

Project Report

Big Data Management Analytics

**SQL - LIBRARY MANAGEMENT SYSTEM**

Submitted to - Prof. Amarnath Mitra

**FORE School of Management, New Delhi**



**Submitted by -**

Shreeya Yashvi 055045

Kashish Srivastava 055046

## [Table of Contents](#)

1. Problem Statement
2. Objectives
3. Introduction
4. Database Overview
5. Database Creation and Design
6. Tables and Their Relationship
7. Relationship and Cardinality
8. SQL Queries for Data Retrieval and Manipulation
9. Codes and Outputs from the Query Run
10. Normalisation - 1NF, 2NF
11. Stress Test - INSERT, UPDATE, DELETE
12. Managerial Insights
13. Findings and Analysis
14. Conclusion

## **PROBLEM STATEMENT**

The Library Management System (LMS) is a critical tool for managing library operations, including book tracking, member management, book issuance, reservations, fines, and reviews. However, managing such a system manually or with inefficient database structures can lead to data redundancy, inconsistency, and poor performance. The primary challenge is to design and implement a robust, scalable, and efficient database system that ensures data integrity, supports complex queries, and handles large volumes of data without compromising performance. Additionally, the system must enforce referential integrity through proper use of primary and foreign keys, and it must be normalized to avoid data anomalies.

## **OBJECTIVES**

1. To design and create a relational database for the Library Management System that supports all necessary functionalities, including book management, member management, branch management, transactions, fines, and reviews.
2. To develop SQL queries for efficient data retrieval, manipulation, and analysis, including SELECT, INSERT, UPDATE, and DELETE operations, as well as complex queries involving joins, subqueries, and aggregate functions.
3. To ensure the database adheres to normalization principles (1NF, 2NF) to eliminate data redundancy and anomalies, and to enforce data integrity through primary keys, foreign keys, and constraints.
4. To conduct stress tests on the database to evaluate its performance under various operations (INSERT, UPDATE, DELETE) and to ensure that cascading effects are properly handled, maintaining referential integrity between parent and child tables.
5. To provide insights into the database's performance, scalability, and integrity, and to offer recommendations for further improvements and optimizations.

## **INTRODUCTION**

This project involves the creation of a Library Management System using SQL. The system is designed to manage library operations such as book tracking, member management, book issuance, reservations, fines, and reviews. The database is structured to ensure data integrity, scalability, and efficient querying.

The database structure follows the relational database model using primary keys, foreign keys, constraints, and normalization techniques

The report covers:

- Database structure and schema
- Relationships between tables (Cardinality and Logical Constraints)
- SQL queries for data retrieval and manipulation
- Use of functions, logical operators, and arithmetic operations

## **DATABASE OVERVIEW**

The Library Management System database is designed to handle the following key functionalities:

- Book Management: Track books, authors, publishers, and categories.
- Member Management: Manage library members and their details.
- Branch Management: Handle multiple library branches and their staff.
- Transactions: Manage book issuance, returns, and reservations.
- Fines and Reviews: Track fines for overdue books and allow members to review books.

## **DATABASE SCHEMA**

- **Tables** - stores different types of data.
- **Columns** - different tables, specifying attributes and constraints.
- **Primary Keys (PK)** ensures each record is uniquely identifiable.
- **Foreign Keys (FK)** establishing relationships between tables.
- **Constraints** (e.g., NOT NULL, UNIQUE, DEFAULT) to maintain data integrity.

## **DATABASE CREATION AND DESIGN**

The database, **LibraryManagement**, has been created to store essential information about books, customers, branches, employees, book issuance, and returns.

Use database command (put that command/query as per shreeya's suggestion)

Table Name	Description
Books	Stores book details (ISBN, title, category, rental price, author, publisher)
Customer	Holds customer details (ID, name, address, registration date)
Branch	Stores branch details (branch number, manager ID, address, contact)
Employee	Stores employee information (employee ID, name, position, salary, branch association)
Issue_Status	Records issued books (issue ID, issued customer, book name, issue date, ISBN)
Return_Status	Records returned books (return ID, returning customer, book name, return date, ISBN)

Each table includes **primary keys** and **foreign keys** to establish relationships between different entities.

## **TABLES AND THEIR RELATIONSHIPS**

Table Name	Columns	Primary Key	Foreign Keys (FKs) & Relationships
Authors	AuthorID, FirstName, LastName	AuthorID	-
Publishers	PublisherID, PublisherName, Address, Phone	PublisherID	-

Categories	CategoryID, CategoryName	CategoryID	-
Books	BookID, Title, ISBN, PublicationYear, PublisherID, CategoryID	BookID	FK: PublisherID → Publishers, CategoryID → Categories
BookAuthors	BookID, AuthorID	BookID, AuthorID	FK: BookID → Books, AuthorID → Authors
LibraryBranches	BranchID, BranchName, Address, Phone	BranchID	-
BookCopies	CopyID, BookID, BranchID, Status	CopyID	FK: BookID → Books, BranchID → LibraryBranches
Members	MemberID, FirstName, LastName, Email, Phone, Address, MembershipDate	MemberID	-
Issue	IssueID, CopyID, MemberID, LoanDate, DueDate, ReturnDate	IssueID	FK: CopyID → BookCopies, MemberID → Members
Reservations	ReservationID, CopyID, MemberID, ReservationDate, Status	ReservationID	FK: CopyID → BookCopies, MemberID → Members
Fines	FineID, MemberID, IssueID, Amount, DateIssued, DatePaid	FineID	FK: MemberID → Members, IssueID → Issue
Librarians	LibrarianID, FirstName, LastName, Email, Phone, BranchID	LibrarianID	FK: BranchID → LibraryBranches
BookReviews	ReviewID, BookID, MemberID, Rating, Comment, ReviewDate	ReviewID	FK: BookID → Books, MemberID → Members

## RELATIONSHIP AND CARDINALITY

Tables Involved	Types	Cardinality	Logical Constraints
Books - Categories	<b>Many-to-One</b> (Many Books belong to one Category, but one Category can have multiple Books)	<ul style="list-style-type: none"> <li>1 Category → Many Books</li> <li>1 Book → 1 Category</li> </ul>	<p>A book must belong to a category (mandatory).</p> <p>A category can exist without books (optional).</p>
Books - BookCopies	<b>One-to-Many</b> (One Book can have multiple Copies, but each Copy belongs to one Book only)	<ul style="list-style-type: none"> <li>1 Book → Many Copies</li> <li>1 Copy → 1 Book</li> </ul>	<p>A book must have at least one copy (mandatory).</p> <p>A copy cannot exist without a book (mandatory).</p>
Books - Publishers	<b>Many-to-One</b> (Many Books can be published by one Publisher, but one Publisher can publish multiple Books)	<ul style="list-style-type: none"> <li>1 Publisher → Many Books</li> <li>1 Book → 1 Publisher</li> </ul>	<p>A book must have a publisher (mandatory).</p> <p>A publisher can exist without books (optional).</p>
Members - Reservations	<b>One-to-Many</b> (One Member can make multiple Reservations, but each Reservation belongs to one Member only)	<ul style="list-style-type: none"> <li>1 Member → Many Reservations</li> <li>1 Reservation → 1 Member</li> </ul>	<p>A reservation must be linked to a member (mandatory).</p> <p>A member may not have any reservations (optional).</p>
BookCopies - Reservations	<b>One-to-Many</b> (One Book Copy can be reserved multiple)	<ul style="list-style-type: none"> <li>1 Copy → Many Reservations</li> <li>1 Reservation</li> </ul>	<p>A reservation must be linked to a book copy (mandatory).</p>

	times, but each Reservation is for one Book Copy only)	→ 1 Copy	A book copy may not have any reservations (optional).
Reservations - Issue	<b>One-to-One</b> (A Reservation results in one Issue if the book is borrowed, otherwise, it remains unissued)	<ul style="list-style-type: none"> <li>• 1 Reservation → At most 1 Issue</li> <li>• 1 Issue → 1 Reservation</li> </ul>	<p>A reservation may or may not lead to an issue (optional).</p> <p>A book cannot be issued without being reserved first (mandatory).</p>
Issue - Fines	<b>One-to-One</b> (A Fine is issued for a specific Issue, and one Issue can result in at most one Fine)	<ul style="list-style-type: none"> <li>• 1 Issue → At most 1 Fine</li> <li>• 1 Fine → 1 Issue</li> </ul>	<p>A fine must be linked to a book issue (mandatory).</p> <p>A book issue may not always result in a fine (optional).</p>
Members - Issue	<b>One-to-Many</b> (One Member can borrow multiple books, but each Issue belongs to only one Member)	<ul style="list-style-type: none"> <li>• 1 Member → Many Issues</li> <li>• 1 Issue → 1 Member</li> </ul>	<p>A book issue must be linked to a member (mandatory).</p> <p>A member may not have any issued books (optional).</p>
LibraryBranch - BookCopies	<b>One-to-Many</b> (One Library Branch can have multiple Book Copies, but each Copy belongs to only one Branch)	<ul style="list-style-type: none"> <li>• 1 Branch → Many Copies</li> <li>• 1 Copy → 1 Branch</li> </ul>	<p>A book copy must belong to a branch (mandatory).</p> <p>A branch may have no books yet (optional).</p>
Librarians - LibraryBranch	<b>Many-to-One</b> (Many Librarians work in one Branch, but one Branch can have	<ul style="list-style-type: none"> <li>• 1 Branch → Many Librarians</li> <li>• 1 Librarian → 1</li> </ul>	A librarian must be assigned to a branch (mandatory).



	multiple Librarians)	Branch	A branch can exist without a librarian (optional).
Books - Authors	<b>Many-to-Many</b> (A Book can have multiple Authors, and an Author can write multiple Books)	<ul style="list-style-type: none"> <li>• 1 Book → Many Authors</li> <li>• 1 Author → Many Books</li> </ul>	<p>A book must have at least one author (mandatory).</p> <p>An author may not have any books yet (optional).</p>

### **SQL QUERIES FOR DATA RETRIEVAL AND MANIPULATION**

Following are the types of queries used for data retrieval and manipulation -

1. Data Retrieval Queries (SELECT Queries)
2. Data Manipulation Queries (INSERT, UPDATE, DELETE)
3. Aggregate and Analytical Queries
4. Joins and Nested Queries

## CODES AND OUTPUTS FROM THE QUERY RUN

### a) Finding most issued books

- Sorts books by the number of issues in descending order and retrieves the top 5 most issued books.
- Result: Harry Potter, Moby-Dick, The catcher in the Rye, The Hobbit are the top most issued books.

The screenshot shows a SQL query window with the following code:

```
-- Most Borrowed Books
-- Find the top 5 most issued books in the library.
SELECT B.Title, COUNT(I.IssueID) AS Total_Issues
FROM Issue I
JOIN BookCopies BC ON I.CopyID = BC.CopyID
JOIN Books B ON BC.BookID = B.BookID
GROUP BY B.Title
ORDER BY Total_Issues DESC
LIMIT 5;
```

The result grid displays the following data:

Title	Total_Issues
Harry Potter and the Sorcerer's Stone	1
Moby-Dick	1
The Catcher in the Rye	1
The Hobbit	1

### b) Finding Books Reserved but Never Issued

- Finds books that were reserved but never actually issued.
- Result: Pride and Prejudice had one reservation but was never issued.

The screenshot shows a SQL query window with the following code:

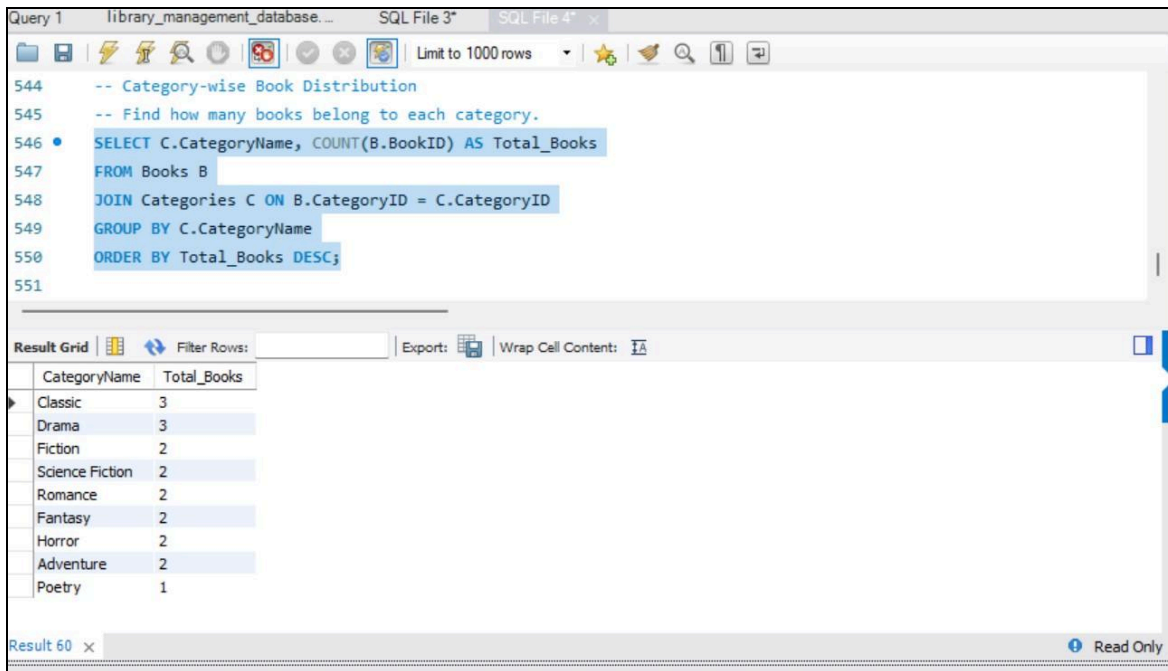
```
-- Books That Are Reserved but Never Issued
-- Identify books that have been reserved but never actually borrowed.
SELECT B.Title, COUNT(R.ReservationID) AS Total_Reservations
FROM Reservations R
JOIN BookCopies BC ON R.CopyID = BC.CopyID
JOIN Books B ON BC.BookID = B.BookID
WHERE R.CopyID NOT IN (SELECT DISTINCT CopyID FROM Issue)
GROUP BY B.Title
ORDER BY Total_Reservations DESC;
```

The result grid displays the following data:

Title	Total_Reservations
Pride and Prejudice	1

### c) Category-wise Book Distribution

- Displays the number of books in each category.
- Categories with the most books: Classic (3), Drama (3), Fiction (2).



```
544 -- Category-wise Book Distribution
545 -- Find how many books belong to each category.
546 • SELECT C.CategoryName, COUNT(B.BookID) AS Total_Books
547 FROM Books B
548 JOIN Categories C ON B.CategoryID = C.CategoryID
549 GROUP BY C.CategoryName
550 ORDER BY Total_Books DESC;
551
```

CategoryName	Total_Books
Classic	3
Drama	3
Fiction	2
Science Fiction	2
Romance	2
Fantasy	2
Horror	2
Adventure	2
Poetry	1

## 1NF

1NF test has been conducted on the Library Management Database to verify compliance with 1NF. The test includes:

- **Test of Atomicity/Multivariate:** Ensuring all columns contain atomic values.
- **Test of Unique Rows (Primary Key):** Ensuring each table has a primary key.
- **Checking for Repeating Groups:** Ensuring no repeating groups exist in the tables.

## Methodology

The following steps were taken to test for 1NF:

1. Identification of Multi-Valued Attributes: Checked for columns that may contain multiple values (e.g., comma-separated values).
2. Checking for Primary Keys: Verified that each table has a primary key.

3. Checking for Repeating Groups: Ensured no repeating groups exist in the tables.
4. Run Queries: Executed SQL queries to detect multi-valued attributes, missing primary keys, and repeating groups.
5. Modifying Tables: If issues were found, modified the tables to ensure compliance with 1NF.
6. Verified Results: Re-run queries to confirm that the tables now comply with 1NF.

## **QUERIES FOR 1NF TEST**

### **1. Condition 1 - Test of Atomicity/ Multivariate**

#### **1.1 BookCopies Table**

- Query: Checked for multi-valued Status column.
- Result: No rows returned, indicating that the Status column contains atomic values.

Query 1 library\_management\_database... SQL File 3\* SQL File 4\* x

Limit to 1000 rows

```

334
335 -- BookCopies Table (Ensuring single-value status)
336 • SELECT * FROM BookCopies WHERE Status LIKE '%,%';
337
338 -- Members Table (Checking for multiple contact details)
339 • SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Address LIKE '%,%';

```

Result Grid

CopyID	BookID	BranchID	Status
NULL	NULL	NULL	NULL

BookCopies 20 x

Output

Action Output

#	Time	Action	Message
187	11:36:49	ALTER TABLE LibraryBranches ADD COLUMN City VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
188	11:37:05	UPDATE LibraryBranches SET City = TRIM(SUBSTRING_INDEX(Address, ',', -1)) WH...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
189	11:37:21	UPDATE LibraryBranches SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1)) ...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
190	11:37:45	SELECT BranchID, BranchName, Address, City, Phone FROM LibraryBranches LIMIT ...	5 row(s) returned
191	11:41:08	SELECT * FROM LibraryBranches WHERE Address LIKE '%,%' OR Phone LIKE '%,%' ...	0 row(s) returned
192	11:41:56	SELECT * FROM BookCopies WHERE Status LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.2 Members Table

- Query: Check for multi-valued Phone, Email, or Address columns.
- Result: No rows returned, confirming that these columns contain atomic values.

The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```
336 • SELECT * FROM BookCopies WHERE Status LIKE '%,%';
337
338 -- Members Table (Checking for multiple contact details)
339 • SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Address LIKE '%,%';
340 • ALTER TABLE Members ADD COLUMN City VARCHAR(100);
341 • UPDATE Members
```

The bottom pane shows the 'Members' table with 23 rows. The columns are: MemberID, FirstName, LastName, Email, Phone, Address, MembershipDate, and City. The first row shows a member with ID 1, and all other columns are empty.

The 'Output' pane shows the results of the queries:

#	Time	Action	Message
193	11:42:35	SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Address LIKE '%,%';	5 row(s) returned
194	11:43:46	ALTER TABLE Members ADD COLUMN City VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
195	11:43:59	UPDATE Members SET City = TRIM(SUBSTRING_INDEX(Address, ',', -1)) WHERE A...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
196	11:44:17	UPDATE Members SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1)) WHER...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
197	11:44:34	SELECT MemberID, FirstName, LastName, Email, Phone, Address, City, MembershipD...	5 row(s) returned
198	11:45:12	SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Address LIKE '%,%';	0 row(s) returned

## 1.3 Books Table

- Query: Check for a multi-valued Summary column.
- Result: No rows returned, confirming that the Summary column contains atomic values.

The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```
304 -- Books Table (Checking for multi-valued attributes in Summary)
305 • SELECT * FROM Books WHERE Summary LIKE '%,%';
306 • UPDATE Books
307 SET Summary = REPLACE(Summary, ',', '')
308 WHERE BookID = 20;
309
```

The bottom pane shows the 'Books' table with 32 rows. The columns are: BookID, Title, ISBN, PublicationYear, PublisherID, CategoryID, and Summary. The first row shows a book with ID 1, and all other columns are empty.

The 'Output' pane shows the results of the queries:

#	Time	Action	Message
203	11:49:21	SELECT * FROM BookReviews WHERE Rating LIKE '%,%' OR Comment LIKE '%,%' L...	0 row(s) returned
204	11:50:25	SELECT * FROM BookAuthors WHERE BookID LIKE '%,%' OR AuthorID LIKE '%,%' LI...	0 row(s) returned
205	11:51:09	SELECT * FROM Authors WHERE FirstName LIKE '%,%' OR LastName LIKE '%,%' OR...	0 row(s) returned
206	11:52:12	SELECT * FROM Books WHERE Summary LIKE '%,%' LIMIT 0, 1000	1 row(s) returned
207	11:59:20	UPDATE Books SET Summary = REPLACE(Summary, ',', '') WHERE BookID = 20	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
208	11:59:26	SELECT * FROM Books WHERE Summary LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.4 Categories Table

- Query: Check for multi-valued CategoryName columns.
- Result: No rows returned, indicating that the CategoryName column contains atomic values.

The screenshot shows a SQL query window with the following code:

```

314 • SELECT PublisherID, PublisherName, Address FROM Publishers;
315
316 -- Categories Table (Ensuring categories are atomic)
317 • SELECT * FROM Categories WHERE CategoryName LIKE '%,%';
318
319 -- BookAuthors Table (Checking for multi-valued author-book relations)

```

The Results pane shows the 'Categories' table with 14 rows, all of which are NULL. The Output pane shows the execution log:

#	Time	Action	Message
178	11:24:02	ALTER TABLE Publishers ADD COLUMN Country VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
179	11:24:55	UPDATE Publishers SET Country = TRIM(SUBSTRING_INDEX(Address, ',', -1)) WHE...	0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0
180	11:25:32	UPDATE Publishers SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1))	0 row(s) affected Rows matched: 20 Changed: 0 Warnings: 0
181	11:25:58	SELECT PublisherID, PublisherName, Address, Country FROM Publishers LIMIT 0, 1000	20 row(s) returned
182	11:29:29	SELECT PublisherID, PublisherName, Address FROM Publishers LIMIT 0, 1000	20 row(s) returned
183	11:30:12	SELECT * FROM Categories WHERE CategoryName LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.5 BookAuthors Table

- Query: Check for multi-valued BookID or AuthorID columns.
- Result: No rows returned, confirming that these columns contain atomic values.

The screenshot shows a SQL query window with the following code:

```

317 • SELECT * FROM Categories WHERE CategoryName LIKE '%,%';
318
319 -- BookAuthors Table (Checking for multi-valued author-book relations)
320 • SELECT * FROM BookAuthors WHERE BookID LIKE '%,%' OR AuthorID LIKE '%,%';
321
322 -- LibraryBranches Table (Checking for multiple addresses or contacts)

```

The Results pane shows the 'BookAuthors' table with 16 rows, all of which are NULL. The Output pane shows the execution log:

#	Time	Action	Message
180	11:25:32	UPDATE Publishers SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1))	0 row(s) affected Rows matched: 20 Changed: 0 Warnings: 0
181	11:25:58	SELECT PublisherID, PublisherName, Address, Country FROM Publishers LIMIT 0, 1000	20 row(s) returned
182	11:29:29	SELECT PublisherID, PublisherName, Address FROM Publishers LIMIT 0, 1000	20 row(s) returned
183	11:30:12	SELECT * FROM Categories WHERE CategoryName LIKE '%,%' LIMIT 0, 1000	0 row(s) returned
184	11:31:45	SELECT * FROM Publishers WHERE Phone LIKE '%,%' OR Address LIKE '%,%' LIMIT ...	0 row(s) returned
185	11:32:58	SELECT * FROM BookAuthors WHERE BookID LIKE '%,%' OR AuthorID LIKE '%,%' LI...	0 row(s) returned



## 1.6 Issue Table

- Query: Check for multi-valued ReturnDate columns.
- Result: No rows returned, indicating that the ReturnDate column contains atomic values.

The screenshot shows a SQL query window with the following SQL code:

```
350
351 -- Issue Table (Checking if Status column allows multiple values)
352 • SELECT * FROM Issue WHERE ReturnDate LIKE '%,%';
353
354 -- Reservations Table (Checking for multiple status values)
355 • SELECT * FROM Reservations WHERE Status LIKE '%,%';
```

The Results pane shows a single row with all columns (IssueID, CopyID, MemberID, LoanDate, DueDate, ReturnDate) containing NULL values.

The Output pane shows the following messages:

#	Time	Action	Message
194	11:43:46	ALTER TABLE Members ADD COLUMN City VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
195	11:43:59	UPDATE Members SET City = TRIM(SUBSTRING_INDEX(Address, ',', -1)) WHERE A...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
196	11:44:17	UPDATE Members SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1)) WHER...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
197	11:44:34	SELECT MemberID, FirstName, LastName, Email, Phone, Address, City, MembershipD...	5 row(s) returned
198	11:45:12	SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Adre...	0 row(s) returned
199	11:46:13	SELECT * FROM Issue WHERE ReturnDate LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.7 Reservations Table

- Query: Check for multi-valued Status columns.
- Result: No rows returned, confirming that the Status column contains atomic values.

The screenshot shows a SQL query window with the following SQL code:

```
352 • SELECT * FROM Issue WHERE ReturnDate LIKE '%,%';
353
354 -- Reservations Table (Checking for multiple status values)
355 • SELECT * FROM Reservations WHERE Status LIKE '%,%';
356
357 -- Fines Table (Checking for multiple Amount values)
```

The Results pane shows a single row with all columns (ReservationID, CopyID, MemberID, ReservationDate, Status) containing NULL values.

The Output pane shows the following messages:

#	Time	Action	Message
195	11:43:59	UPDATE Members SET City = TRIM(SUBSTRING_INDEX(Address, ',', -1)) WHERE A...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
196	11:44:17	UPDATE Members SET Address = TRIM(SUBSTRING_INDEX(Address, ',', 1)) WHER...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0
197	11:44:34	SELECT MemberID, FirstName, LastName, Email, Phone, Address, City, MembershipD...	5 row(s) returned
198	11:45:12	SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Adre...	0 row(s) returned
199	11:46:13	SELECT * FROM Issue WHERE ReturnDate LIKE '%,%' LIMIT 0, 1000	0 row(s) returned
200	11:46:59	SELECT * FROM Reservations WHERE Status LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.8 Fines Table

- Query: Check for multi-valued Amount columns.
- Result: No rows returned, indicating that the Amount column contains atomic values.

The screenshot shows the SQL Developer interface with a query window titled 'Query 1' containing the following SQL code:

```
354 -- Reservations Table (Checking for multiple status values)
355 • SELECT * FROM Reservations WHERE Status LIKE '%,%';
356
357 -- Fines Table (Checking for multiple Amount values)
358 • SELECT * FROM Fines WHERE Amount LIKE '%,%';
359
```

Below the query window, the 'Result Grid' shows a table with columns: FineID, MemberID, IssueID, Amount, DateIssued, DatePaid. The first row contains NULL values.

The 'Output' window shows the 'Action Output' table with columns: #, Time, Action, Message. The last row of the output is:

#	Time	Action	Message
201	11:47:44	SELECT * FROM Fines WHERE Amount LIKE '%,%' LIMIT 0, 1000	0 row(s) returned

## 1.9 Librarians Table

- Query: Check for multi-valued Phone or Email columns.
- Result: No rows returned, confirming that these columns contain atomic values.

The screenshot shows the SQL Developer interface with a query window titled 'Query 1' containing the following SQL code:

```
357 -- Fines Table (Checking for multiple Amount values)
358 • SELECT * FROM Fines WHERE Amount LIKE '%,%';
359
360 -- Librarians Table (Checking for multiple roles or contacts)
361 • SELECT * FROM Librarians WHERE Phone LIKE '%,%' OR Email LIKE '%,%';
362
```

Below the query window, the 'Result Grid' shows a table with columns: LibrarianID, FirstName, LastName, Email, Phone, BranchID. The first row contains NULL values.

The 'Output' window shows the 'Action Output' table with columns: #, Time, Action, Message. The last row of the output is:

#	Time	Action	Message
202	11:48:30	SELECT * FROM Librarians WHERE Phone LIKE '%,%' OR Email LIKE '%,%' LIMIT 0, ...	0 row(s) returned



## 1.10 BookReviews Table

- Query: Check for multi-valued Rating or Comment columns.
- Result: No rows returned, indicating that these columns contain atomic values.

The screenshot shows a SQL query window with the following text:

```

360 -- Librarians Table (Checking for multiple roles or contacts)
361 • SELECT * FROM Librarians WHERE Phone LIKE '%,%' OR Email LIKE '%,%';
362
363 -- BookReviews Table (Ensuring a single review per row)
364 • SELECT * FROM BookReviews WHERE Rating LIKE '%,%' OR Comment LIKE '%,%';
365

```

Below the query, the 'Result Grid' is empty, showing columns: ReviewID, BookID, MemberID, Rating, Comment, ReviewDate.

The 'Output' pane shows the following messages:

#	Time	Action	Message
198	11:45:12	SELECT * FROM Members WHERE Phone LIKE '%,%' OR Email LIKE '%,%' OR Address LIKE '%,%'...	0 row(s) returned
199	11:46:13	SELECT * FROM Issue WHERE ReturnDate LIKE '%,%' LIMIT 0, 1000	0 row(s) returned
200	11:46:59	SELECT * FROM Reservations WHERE Status LIKE '%,%' LIMIT 0, 1000	0 row(s) returned
201	11:47:44	SELECT * FROM Fines WHERE Amount LIKE '%,%' LIMIT 0, 1000	0 row(s) returned
202	11:48:30	SELECT * FROM Librarians WHERE Phone LIKE '%,%' OR Email LIKE '%,%' LIMIT 0, ...	0 row(s) returned
203	11:49:21	SELECT * FROM BookReviews WHERE Rating LIKE '%,%' OR Comment LIKE '%,%' L...	0 row(s) returned

## 2. Condition 2 - Test of Unique Rows (Primary Key)

### 2.1 Check for Missing Primary Keys

- Query: Identify tables without a primary key.
- Result: No rows returned, indicating that all tables in the database have a primary key.

The screenshot shows a SQL query window with the following text:

```

371 • SELECT TABLE_NAME
372 FROM INFORMATION_SCHEMA.TABLES
373 WHERE TABLE_SCHEMA = 'LibraryManagement'
374 AND TABLE_NAME NOT IN (
375     SELECT DISTINCT TABLE_NAME
376     FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
377     WHERE CONSTRAINT_NAME = 'PRIMARY'
378 )

```

Below the query, the 'Result Grid' is empty, showing the column: TABLE\_NAME.

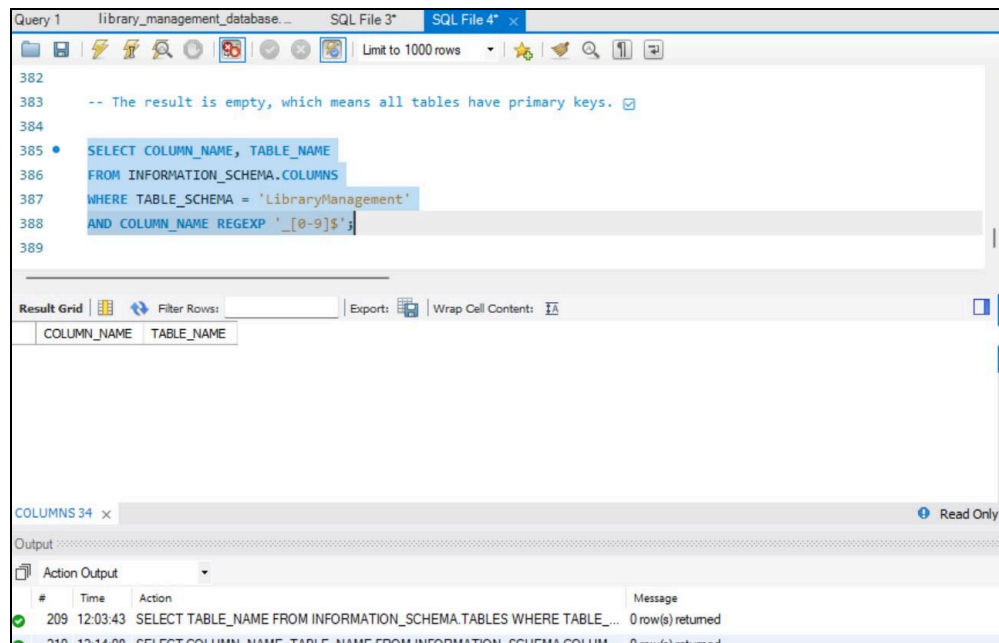
The 'Output' pane shows the following messages:

#	Time	Action	Message
204	11:50:25	SELECT * FROM BookAuthors WHERE BookID LIKE '%,%' OR AuthorID LIKE '%,%' LI...	0 row(s) returned
205	11:51:09	SELECT * FROM Authors WHERE FirstName LIKE '%,%' OR LastName LIKE '%,%' OR...	0 row(s) returned

### 3. Condition 3 - Checking for Repeating Groups

#### 3.1 Checking for Repeating Groups

- Query: Identify columns with names ending in numbers (indicating potential repeating groups).
- Result: No rows returned, indicating that no repeating groups exist in the database.



### RESULT

- **Atomicity/Multivariate Test:**
  - All tables passed the atomicity test and hence complies with 1NF.
- **Primary Key Test:**
  - All tables have a primary key, ensuring unique rows.
- **Repeating Groups Test:**
  - No repeating groups were found in the database.

Library Management Database successfully passed the First Normal Form (1NF) test. All tables were found to contain atomic values, have primary keys, and lack repeating groups. The database is compliant with 1NF, ensuring a solid foundation for further normalization.

## 2NF

2NF test has been conducted on the Library Management Database to verify compliance with 2NF. The test includes:

- **Identifying Composite Primary Keys:** Ensuring tables with composite keys are properly structured.
- **Checking for Partial Dependencies:** Ensuring no non-prime attributes depend on a subset of the composite key.

## Methodology

The following steps were taken to test for 2NF:

1. Identify Composite Primary Keys: Locate tables with composite keys.
2. Check for Partial Dependencies: Verify that non-prime attributes are fully dependent on the entire composite key.
3. Run Queries: Execute SQL queries to detect partial dependencies.
4. Modify Tables: If partial dependencies are found, modify the tables to ensure compliance with 2NF.
5. Verify Results: Re-run queries to confirm that the tables now comply with 2NF.

## QUERIES FOR 2NF TEST

### 1. Condition 1 - Identifying Composite Primary Key

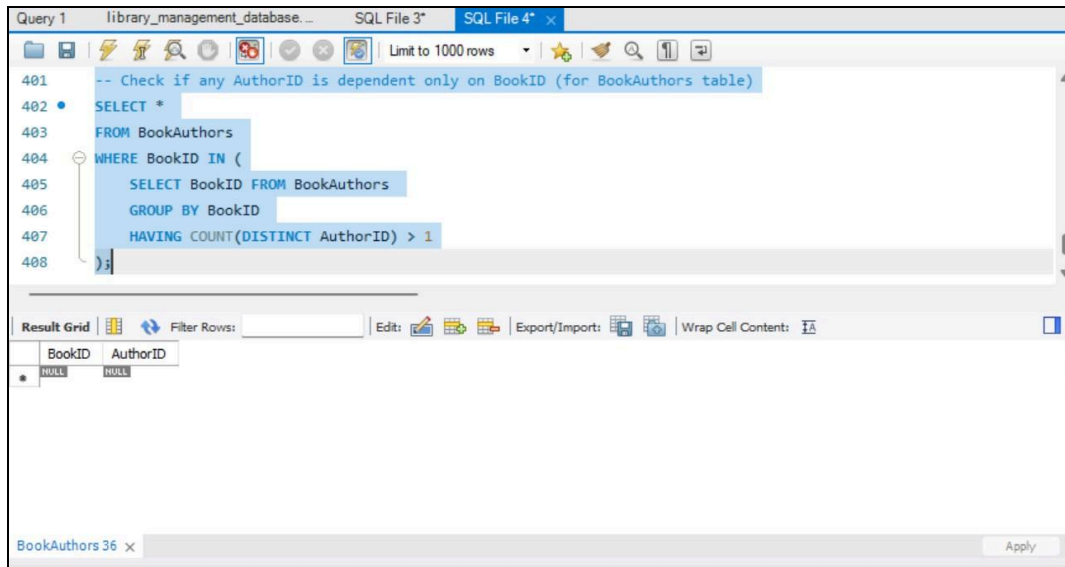
The following tables were identified as having composite primary keys:

- BookAuthors: Composite key (BookID, AuthorID).
- Reservations: Composite key (ReservationID, CopyID).
- Issue: Composite key (IssueID, CopyID).

## 2. Condition 2 - Checking for Partial Dependencies

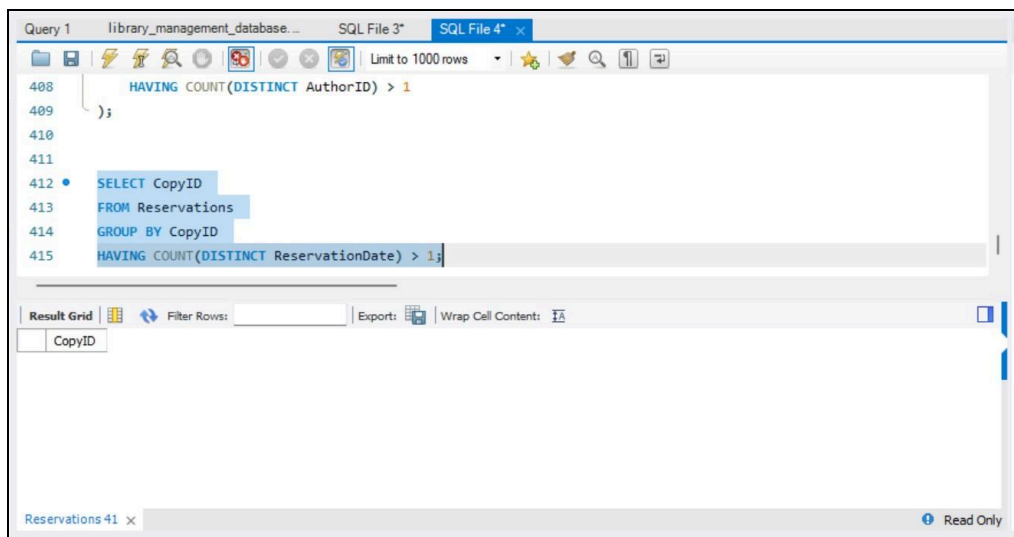
### 2.1 BookAuthors Table

- Dependency Check: Check if AuthorID is partially dependent on BookID.
- Result: No rows returned, indicating that AuthorID is not partially dependent on BookID. The BookAuthors table complies with 2NF.



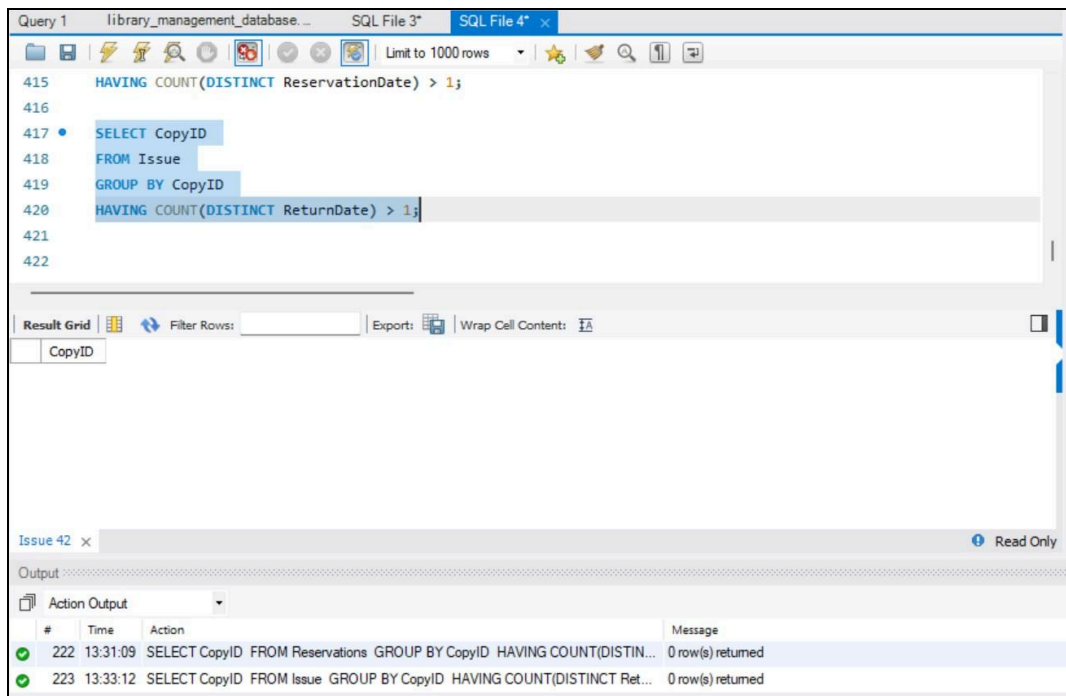
### 2.2 Reservations Table

- Dependency Check: Check if ReservationDate is partially dependent on CopyID.
- Result: No rows returned, indicating that ReservationDate is not partially dependent on CopyID. The Reservations table complies with 2NF.



### 2.3 Issue Table

- Dependency Check: Check if ReturnDate is partially dependent on CopyID.
- Result: No rows returned, indicating that ReturnDate is not partially dependent on CopyID. The Issue table complies with 2NF.



## RESULTS

- The tables with composite keys (BookAuthors, Reservations, and Issue) are well-structured, and no partial dependencies were found.
- The absence of partial dependencies ensures that non-prime attributes are fully dependent on the entire composite key, maintaining data integrity.
- The database is well-structured for future scalability, as it adheres to 2NF principles, ensuring that non-prime attributes are fully functionally dependent on the primary key.

The Library Management Database successfully passed the Second Normal Form (2NF) test. All tables with composite keys were found to be free from partial dependencies. The database is now compliant with 2NF.

## **STRESS TEST**

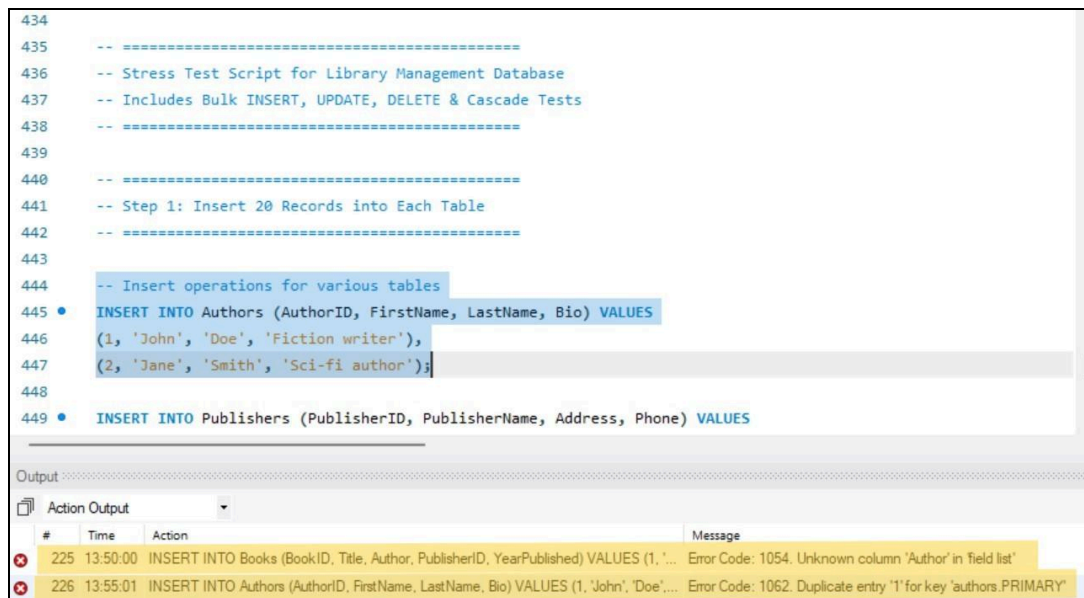
The stress test was conducted to evaluate the performance and integrity of the library management database under various operations, including bulk inserts, updates, and deletes. The test also aimed to identify any cascading effects between parent and child tables and to detect any errors that might arise during these operations.

### **→ INSERT**

#### **1.1 Primary Key issue**

##### **1.1.1 Duplicate Entry**

- Query: `INSERT INTO Authors (AuthorID, FirstName, LastName, Bio) VALUES (1, 'John', 'Doe', 'Fiction writer');`
- Issue: The error indicates a duplicate entry for the primary key AuthorID. This suggests that the AuthorID value 1 already exists in the Authors table.



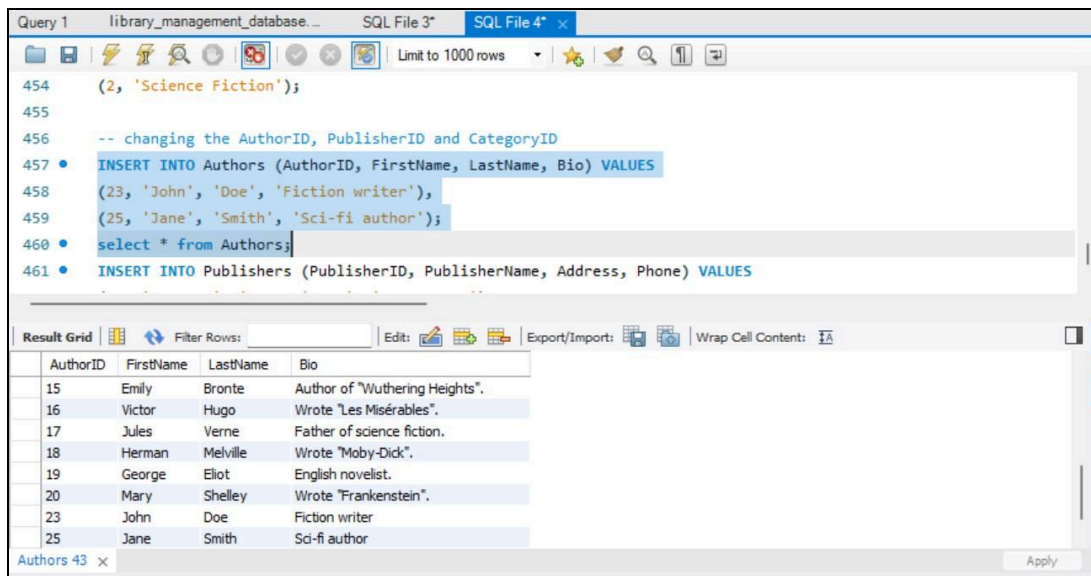
```
434
435 -- =====
436 -- Stress Test Script for Library Management Database
437 -- Includes Bulk INSERT, UPDATE, DELETE & Cascade Tests
438 -- =====
439
440 -- =====
441 -- Step 1: Insert 20 Records into Each Table
442 -- =====
443
444 -- Insert operations for various tables
445 • INSERT INTO Authors (AuthorID, FirstName, LastName, Bio) VALUES
446   (1, 'John', 'Doe', 'Fiction writer'),
447   (2, 'Jane', 'Smith', 'Sci-fi author');
448
449 • INSERT INTO Publishers (PublisherID, PublisherName, Address, Phone) VALUES
```

Output :

#	Time	Action	Message
225	13:50:00	INSERT INTO Books (BookID, Title, Author, PublisherID, YearPublished) VALUES (1, '...	Error Code: 1054. Unknown column 'Author' in field list
226	13:55:01	INSERT INTO Authors (AuthorID, FirstName, LastName, Bio) VALUES (1, 'John', 'Doe', ...	Error Code: 1062. Duplicate entry '1' for key 'authors.PRIMARY'

## 1.2 Authors Table

- Query: INSERT INTO Authors (AuthorID, FirstName, LastName, Bio) VALUES (23, 'John', 'Doe', 'Fiction writer');
- Result: The record was successfully inserted without any errors.



The screenshot shows a SQL IDE window with a query window at the top and a result grid at the bottom. The query window contains the following SQL code:

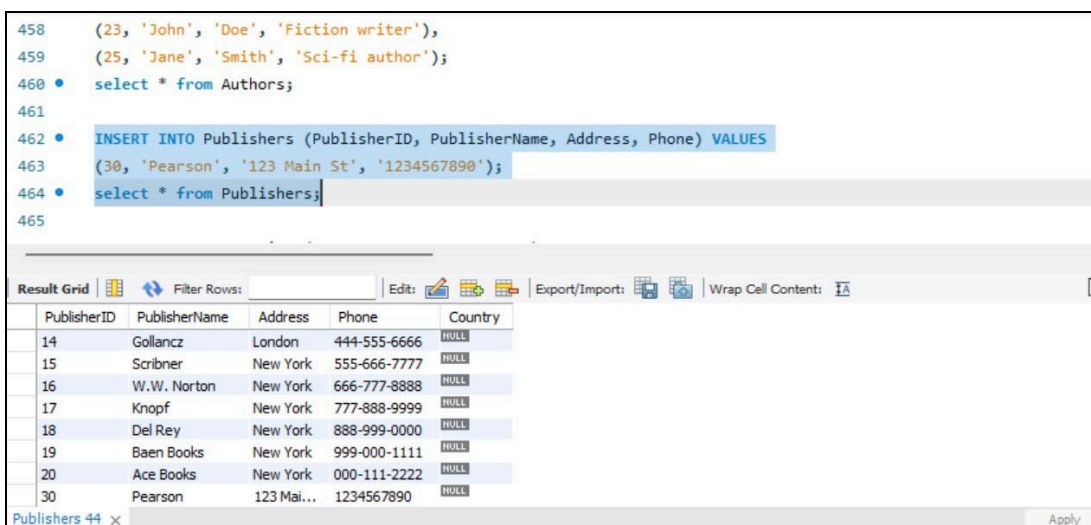
```
454 (2, 'Science Fiction');
455
456 -- changing the AuthorID, PublisherID and CategoryID
457 INSERT INTO Authors (AuthorID, FirstName, LastName, Bio) VALUES
458 (23, 'John', 'Doe', 'Fiction writer'),
459 (25, 'Jane', 'Smith', 'Sci-fi author');
460 select * from Authors;
461 INSERT INTO Publishers (PublisherID, PublisherName, Address, Phone) VALUES
```

The result grid shows the following data:

AuthorID	FirstName	LastName	Bio
15	Emily	Bronte	Author of "Wuthering Heights".
16	Victor	Hugo	Wrote "Les Misérables".
17	Jules	Verne	Father of science fiction.
18	Herman	Melville	Wrote "Moby-Dick".
19	George	Eliot	English novelist.
20	Mary	Shelley	Wrote "Frankenstein".
23	John	Doe	Fiction writer
25	Jane	Smith	Sci-fi author

## 1.3 Publishers Table:

- Query: INSERT INTO Publishers (PublisherID, PublisherName, Address, Phone) VALUES (30, 'Pearson', '123 Main St', '1234567890');
- Result: The record was successfully inserted without any errors.



The screenshot shows a SQL IDE window with a query window at the top and a result grid at the bottom. The query window contains the following SQL code:

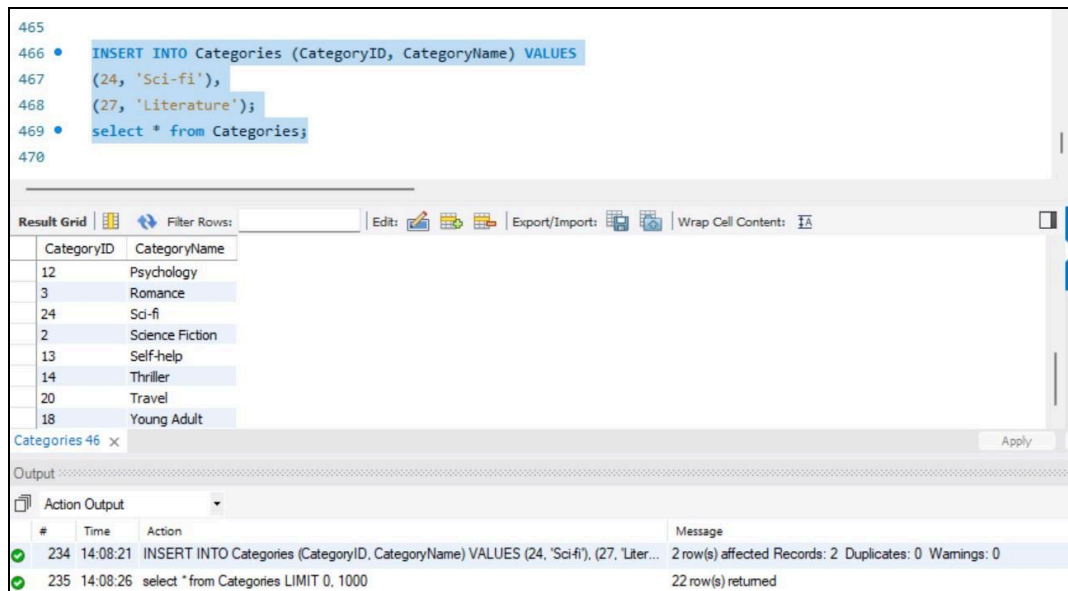
```
458 (23, 'John', 'Doe', 'Fiction writer'),
459 (25, 'Jane', 'Smith', 'Sci-fi author');
460 select * from Authors;
461
462 INSERT INTO Publishers (PublisherID, PublisherName, Address, Phone) VALUES
463 (30, 'Pearson', '123 Main St', '1234567890');
464 select * from Publishers;
465
```

The result grid shows the following data:

PublisherID	PublisherName	Address	Phone	Country
14	Gollancz	London	444-555-6666	NULL
15	Scribner	New York	555-666-7777	NULL
16	W.W. Norton	New York	666-777-8888	NULL
17	Knopf	New York	777-888-9999	NULL
18	Del Rey	New York	888-999-0000	NULL
19	Baen Books	New York	999-000-1111	NULL
20	Ace Books	New York	000-111-2222	NULL
30	Pearson	123 Mai...	1234567890	NULL

## 1.4 Categories Table:

- Query: INSERT INTO Categories (CategoryID, CategoryName) VALUES (24, 'Sci-fi'), (27, 'Literature');
- Result: The records were successfully inserted without any errors.



The successful inserts show cascading effects, indicating that the database maintains referential integrity under normal operations.

## → UPDATE

### 2.1 Books Table Update

- Query: UPDATE Books SET Summary = 'Updated Fictional Story' WHERE BookID = 1;
- Result: The Summary field for BookID 1 was successfully updated.
- Cascading Effect: The update in the Books table cascaded to the BookCopies and Issue tables, as evidenced by the updated Summary field in these tables.



Query 1 library\_management\_database... SQL File 3\* SQL File 4\* x

```

474 -- UPDATE Books SET Summary = 'Updated Fictional Story' WHERE BookID = 1;
475 -- Step 1: Update Summary in Books Table
476 • UPDATE Books
477   SET Summary = 'Updated Fictional Story'
478   WHERE BookID = 1;
479
480 -- Step 2: Display Updated Record from Books Table
481 • SELECT * FROM Books WHERE BookID = 1;

```

Result Grid

BookID	Title	ISBN	PublicationYear	PublisherID	CategoryID	Summary
1	1984	978-0451524935	1949	1	1	Updated Fictional Story

Books 47 x

Output

Action Output

#	Time	Action	Message
237	14:11:57	UPDATE Books SET Summary = 'Updated Fictional Story' WHERE BookID = 1	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
238	14:11:57	SELECT * FROM Books WHERE BookID = 1 LIMIT 0, 1000	1 row(s) returned

## 2.2 BookCopies Table

- Query: `SELECT BC.*, B.Summary FROM BookCopies BC JOIN Books B ON BC.BookID = B.BookID WHERE BC.BookID = 1;`
- Result: The Summary field in the BookCopies table was updated to reflect the change in the Books table.

Query 1 library\_management\_database... SQL File 3\* SQL File 4\* x

```

481 • SELECT * FROM Books WHERE BookID = 1;
482
483 -- Step 3: Display Relevant Child Records Referencing BookID
484 -- Check BookCopies table if it has BookID
485 • SELECT BC.*, B.Summary
486   FROM BookCopies BC
487   JOIN Books B ON BC.BookID = B.BookID
488   WHERE BC.BookID = 1;

```

Result Grid

CopyID	BookID	BranchID	Status	Summary
1	1	1	Available	Updated Fictional Story
2	1	2	On Loan	Updated Fictional Story

Result 48 x

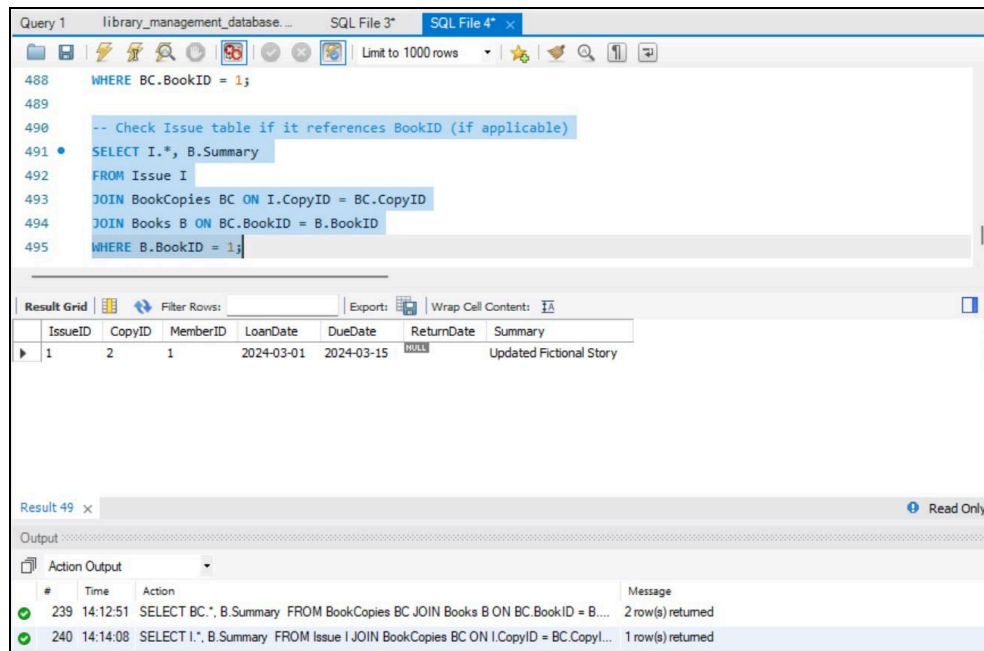
Output

Action Output

#	Time	Action	Message
238	14:11:57	SELECT * FROM Books WHERE BookID = 1 LIMIT 0, 1000	1 row(s) returned
239	14:12:51	SELECT BC.*, B.Summary FROM BookCopies BC JOIN Books B ON BC.BookID = B....	2 row(s) returned

## 2.3 Issue Table

- Query: `SELECT I.*, B.Summary FROM Issue I JOIN BookCopies BC ON I.CopyID = BC.CopyID JOIN Books B ON BC.BookID = B.BookID WHERE B.BookID = 1;`
- Result: The Summary field in the Issue table was updated to reflect the change in the Books table.

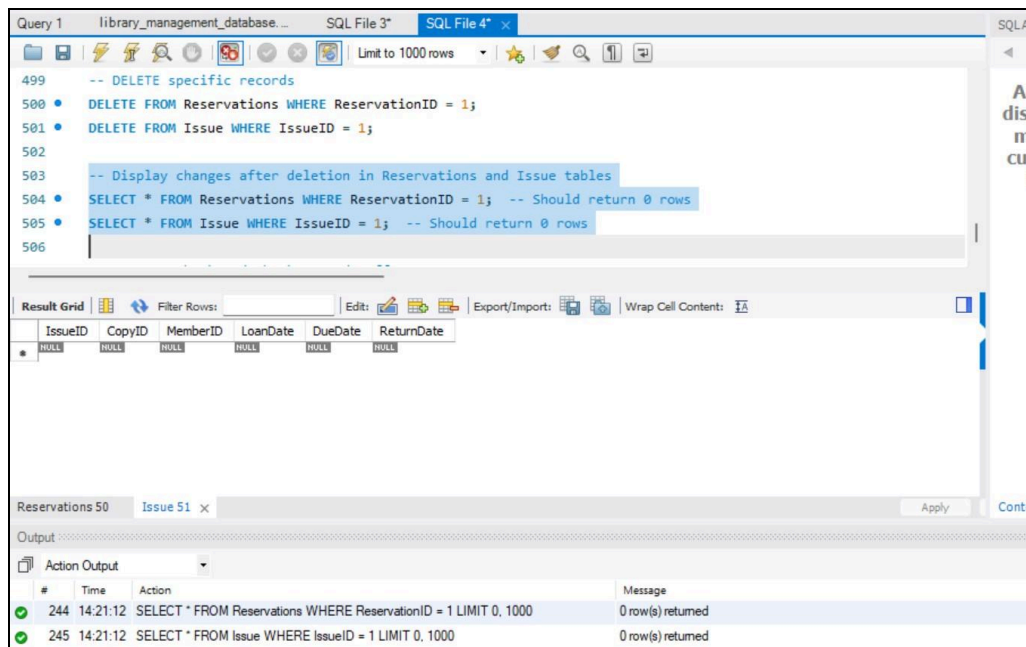


The update operation on the Books table successfully cascaded to the BookCopies and Issue tables. The foreign key constraints are properly enforced, ensuring referential integrity between parent and child tables.

## → DELETE

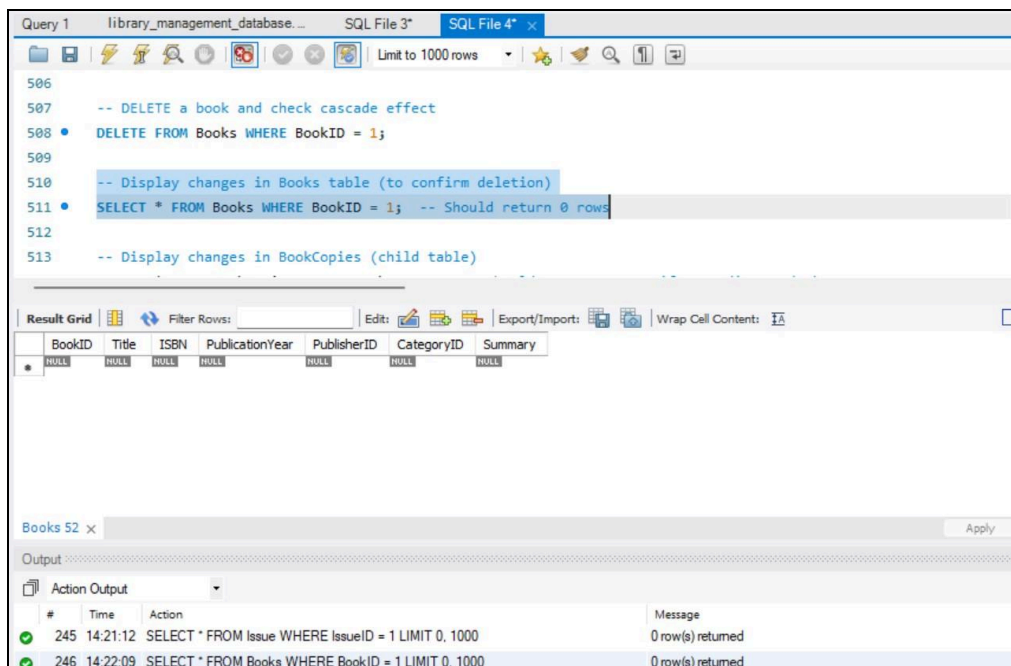
### 3.1 Reservations and Issue Tables

- ◆ Query: `DELETE FROM Reservations WHERE ReservationID = 1; DELETE FROM Issue WHERE IssueID = 1;`
- ◆ Result: The specified records were successfully deleted from the Reservations and Issue tables.
- ◆ Cascading Effect: No cascading effect was observed as these tables do not have child tables dependent on them.



### 3.2 Books Table

- Query: DELETE FROM Books WHERE BookID = 1;
- Result: The specified book record was successfully deleted from the Books table.
- Cascading Effect: The deletion cascaded to the BookCopies and BookAuthors tables, as evidenced by the absence of records with BookID 1 in these tables.



### 3.3 BookCopies Table

- Query: `SELECT * FROM BookCopies WHERE BookID = 1;`
- Result: No records were returned, indicating successful cascading deletion.

The screenshot shows the SQL Developer interface with a query window titled 'Query 1' containing the following SQL code:

```
507 -- DELETE a book and check cascade effect
508 • DELETE FROM Books WHERE BookID = 1;
509
510 -- Display changes in Books table (to confirm deletion)
511 • SELECT * FROM Books WHERE BookID = 1; -- Should return 0 rows
512
513 -- Display changes in BookCopies (child table)
514 • SELECT * FROM BookCopies WHERE BookID = 1; -- Should return 0 rows if cascading worked
```

Below the query window, the 'Result Grid' shows the execution results for the first two queries:

CopyID	BookID	BranchID	Status
*	NULL	NULL	NULL

The 'Output' pane shows the 'Action Output' for the first two queries:

#	Time	Action	Message
246	14:22:09	SELECT * FROM Books WHERE BookID = 1 LIMIT 0, 1000	0 row(s) returned
247	14:22:40	SELECT * FROM BookCopies WHERE BookID = 1 LIMIT 0, 1000	0 row(s) returned

### 3.4 BookAuthors Table

- Query: `SELECT * FROM BookAuthors WHERE BookID = 1;`
- Result: No records were returned, indicating successful cascading deletion.

The screenshot shows the SQL Developer interface with a query window titled 'Query 1' containing the following SQL code:

```
510 -- Display changes in Books table (to confirm deletion)
511 • SELECT * FROM Books WHERE BookID = 1; -- Should return 0 rows
512
513 -- Display changes in BookCopies (child table)
514 • SELECT * FROM BookCopies WHERE BookID = 1; -- Should return 0 rows if cascading worked
515
516 -- Display changes in BookAuthors (child table)
517 • SELECT * FROM BookAuthors WHERE BookID = 1; -- Should return 0 rows if cascading worked
```

Below the query window, the 'Result Grid' shows the execution results for the first two queries:

BookID	AuthorID
*	NULL

The 'Output' pane shows the 'Action Output' for the first two queries:

#	Time	Action	Message
247	14:22:40	SELECT * FROM BookCopies WHERE BookID = 1 LIMIT 0, 1000	0 row(s) returned
248	14:23:52	SELECT * FROM BookAuthors WHERE BookID = 1 LIMIT 0, 1000	0 row(s) returned

The delete operation on the Books table successfully cascaded to the BookCopies and BookAuthors tables. The foreign key constraints are properly enforced, ensuring referential integrity between parent and child tables.

The successful inserts, updates, and deletes, along with the observed cascading effects, indicate that the database maintains referential integrity under normal operations.

## **FINDINGS AND ANALYSIS**

1. The database successfully enforces data integrity through the use of primary keys, foreign keys, and constraints. The normalization process (1NF and 2NF) ensures that the database is free from data redundancy and anomalies, providing a solid foundation for future scalability.
2. The SQL queries developed for data retrieval and manipulation are optimized for performance. The use of joins, subqueries, and aggregate functions allows for efficient data analysis and reporting, which is crucial for decision-making in library management.
3. The stress tests demonstrated that the database handles cascading effects effectively. Updates and deletes in parent tables (e.g., Books) correctly propagate to child tables (e.g., BookCopies, BookAuthors), ensuring referential integrity. This is critical for maintaining consistent and accurate data across the system.
4. The database design supports scalability, allowing for the addition of new tables, columns, and relationships without compromising performance. Regular schema validation and error handling mechanisms are recommended to maintain the database's integrity and performance over time.
5. The database's ability to handle large volumes of data and complex transactions efficiently translates to improved operational efficiency for the library. This includes faster book issuance, accurate tracking of reservations and fines, and better management of library resources.

## **MANAGERIAL INSIGHTS**

- **Ensure Accurate Reporting:** Managers can rely on the database's integrity to generate accurate reports on book availability, member activity, and financial transactions (e.g., fines). This ensures that decisions are based on reliable data.
- **Minimize Errors:** By understanding that the database enforces referential integrity, managers can reduce errors in operations such as book issuance, returns, and reservations, leading to smoother library operations.
- **Optimize Resource Allocation:** Managers can use the efficient query performance to analyze data quickly, such as identifying popular books, tracking overdue books, or evaluating member activity. This helps in allocating resources (e.g., purchasing more copies of high-demand books) effectively.
- **Improve Member Experience:** Fast query performance allows librarians to quickly retrieve information for members, such as book availability or reservation status, enhancing the overall member experience.
- **Monitor Key Metrics:** Managers can use SQL queries to monitor key performance indicators (KPIs) like book circulation rates, member engagement, and fine collection, enabling data-driven decision-making.
- **Simplify Maintenance:** By understanding cascading effects, managers can simplify database maintenance tasks, as they do not need to manually update or delete related records in multiple tables.
- **Reduce Operational Risks:** The cascading effect ensures that critical operations like book deletions or updates do not leave orphaned records, reducing the risk of data inconsistencies or errors in library operations.
- **Streamline Library Operations:** Managers can use the database's efficiency to streamline processes like book issuance, returns, and reservations, reducing wait times and improving member satisfaction.
- **Improve Staff Productivity:** By automating repetitive tasks (e.g., updating book statuses, generating reports), managers can free up staff time for more value-added activities, such as member engagement or organizing library events.

## **CONCLUSION**

The Library Management System database project successfully addresses the challenges of managing library operations through a well-designed, normalized, and efficient relational database. The database ensures data integrity, supports complex queries, and handles large volumes of data with ease. The stress tests confirmed that the database maintains referential integrity through proper cascading effects, and the normalization process eliminated data redundancy and anomalies.

The SQL queries developed for data retrieval and manipulation provide valuable insights into library operations, enabling **better decision-making and resource management**. The database's **scalability and performance** make it a robust solution for libraries of all sizes.

In conclusion, the Library Management System database is a reliable and efficient tool for managing library operations, ensuring data integrity, and supporting complex queries. The project demonstrates the importance of proper database design, normalization, and stress testing in building a scalable and maintainable system.