Sri Lanka Institute of Information Technology

Module: IE2042

Year 2, Semester 1

Database Design, Implementation and Security

Database Management Systems for Security

B.Sc. (Hons) in Information Technology

Specialized in Cyber Security

Group Details

Group Number	

	Student IT Number	Student Name
1	M.F.M. FARHAN	IT23422070
2	AAZAF RITHA. J	IT23151710
3	S.M.F. HANA	IT23255142
4	R.M.T.P. RASNAYAKE	IT23246546

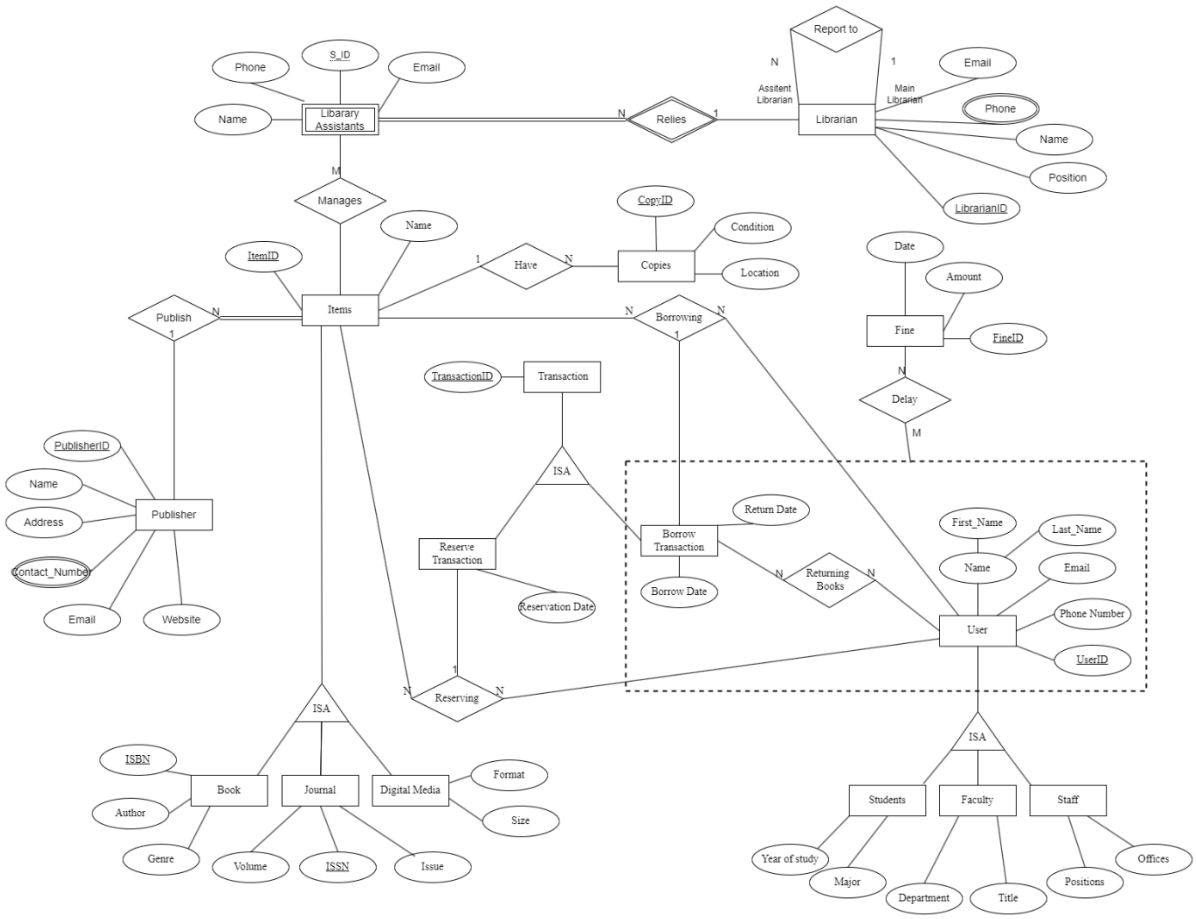
Contents

Part 1 Database Design and Implementation

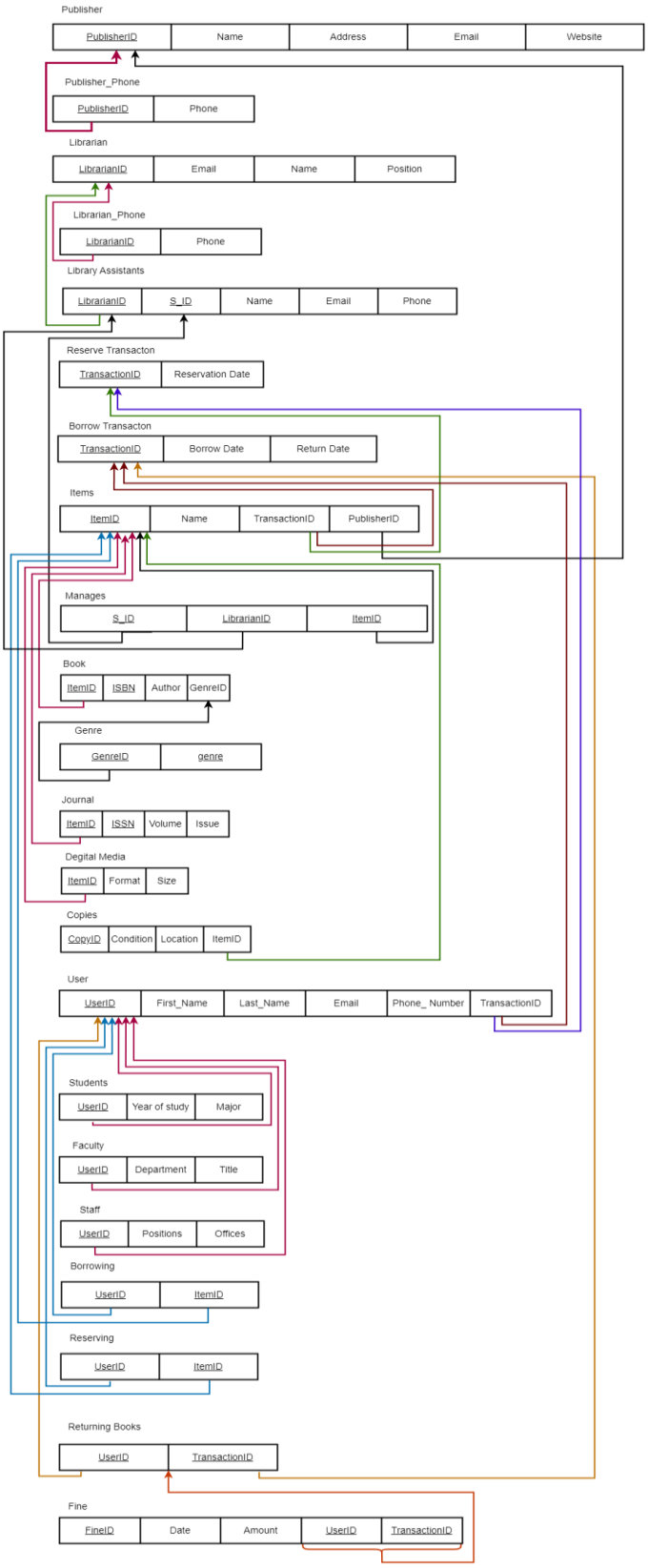
1. Assumptions

1. A library assistant serves only one librarian, assisting with tasks while each librarian has dedicated multiple assistants to help manage their duties.
2. The Assistant Librarians are responsible for compiling and presenting circulation reports, inventory reports, and user activity reports to the main librarian for review and analysis.
3. A library assistant is considered a weak entity as it cannot function. independently without being connected to the Librarian. Their role exists solely to assist the librarian, as without librarian, library assistant has no purpose.
4. When a User borrows an item, any Delay in returning it beyond the due-date triggers a fine.
5. The library implements a fine system for overdue items, where fines are automatically calculated based on the duration of delay.
6. The entity “Items” represent all library materials, with specific subtypes: Books, Journal and Digital media each inheriting common attributes like ISBN and Name
7. The “User” entity represents all the people in the faculty: Student, Faculty and Staff.
8. The “Transaction” entity involves all type of transaction regarding the library items with two specific types such as Borrow Transaction and Reserve Transaction.
9. Phone is taken as a multivalued attribute, reflecting the possibility that Users, Publisher and Librarian may have multiple contact numbers.
10. There is a total participation between Item and Published, meaning that every item must have a Publisher, and no item can exit without being linked to a publisher.

2. ERD (Entity Relation Diagram)



3. Logical Model



Since the database is already in 3rd normal form the normalization phase is not provided

4. Database and Constraints Implementation

```
--Create a new database named 'LibraryDataBase'
CREATE DATABASE LibraryDataBase;
--Use the newly created 'LibraryDataBase'
USE LibraryDataBase;


-- 01. Create Table for Publisher
CREATE TABLE Publisher (
    PublisherID VARCHAR(10) PRIMARY KEY NOT NULL,
    Name VARCHAR(100) NOT NULL,
    Address VARCHAR(255),
    Email VARCHAR(100),
    Website VARCHAR(100)
);

-- 02. Create Table for Publisher_Phone with Composite Primary Key
CREATE TABLE Publisher_Phone (
    PublisherID VARCHAR(10) NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (PublisherID, Phone),
    CONSTRAINT Publisher_ID_fk FOREIGN KEY (PublisherID) REFERENCES
Publisher(PublisherID) ON DELETE CASCADE
);

-- 03. Create Table for Librarian
CREATE TABLE Librarian (
    LibrarianID VARCHAR(10) PRIMARY KEY NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Name VARCHAR(100) NOT NULL,
    Position VARCHAR(50)
);

-- 04. Create Table for Librarian_Phone with Composite Primary Key
CREATE TABLE Librarian_Phone (
    LibrarianID VARCHAR(10) NOT NULL,
    Phone VARCHAR(15) NOT NULL,
    PRIMARY KEY (LibrarianID, Phone),
    CONSTRAINT Librarian_Phone_fk FOREIGN KEY (LibrarianID) REFERENCES
Librarian(LibrarianID) ON DELETE CASCADE
);

-- 05. Create Table for Library_Assistants
CREATE TABLE Library_Assistants (
    S_ID VARCHAR(10) NOT NULL,
    LibrarianID VARCHAR(10) NOT NULL,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100),
    Phone VARCHAR(10) UNIQUE,
    PRIMARY KEY (S_ID, LibrarianID),
    CONSTRAINT Library_Assistants_fk FOREIGN KEY (LibrarianID) REFERENCES
Librarian(LibrarianID)
);

-- 06. Create Table for Reserve_Transaction
CREATE TABLE Reserve_Transaction (
    TransactionID VARCHAR(10) PRIMARY KEY,
    ReservationDate DATE
);

-- 07. Create Table for Borrow_Transaction
```

```
CREATE TABLE Borrow_Transaction (
    TransactionID VARCHAR(10) PRIMARY KEY,
    BorrowDate DATE,
    ReturnDate DATE
);

-- 08. Create Table for Items
CREATE TABLE Items (
    ItemID VARCHAR(10) PRIMARY KEY NOT NULL,
    Name VARCHAR(100),
    ReservationTransactionID VARCHAR(10),
    BorrowTransactionID VARCHAR(10),
    PublisherID VARCHAR(10),
    CONSTRAINT Items_Trans_Rese_fk FOREIGN KEY (ReservationTransactionID) REFERENCES
Reserve_Transaction(TransactionID) ON DELETE SET NULL,
    CONSTRAINT Items_Trans_Bor_fk FOREIGN KEY (BorrowTransactionID) REFERENCES
Borrow_Transaction(TransactionID) ON DELETE SET NULL,
    CONSTRAINT Items_Pub_fk FOREIGN KEY (PublisherID) REFERENCES
Publisher(PublisherID)
);

-- 09. Create Table for Manages
CREATE TABLE Manages (
    S_ID VARCHAR(10) NOT NULL,
    LibrarianID VARCHAR(10) NOT NULL,
    ItemID VARCHAR(10) NOT NULL,
    PRIMARY KEY (S_ID, LibrarianID, ItemID),
    CONSTRAINT Manages_Staff_fk FOREIGN KEY (S_ID, LibrarianID) REFERENCES
Library_Assistants(S_ID, LibrarianID),
    CONSTRAINT Manages_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 10. Create Table for Book
CREATE TABLE Book (
    ItemID VARCHAR(10) PRIMARY KEY,
    ISBN VARCHAR(20),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    CONSTRAINT Book_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 11. Create Table for Journal
CREATE TABLE Journal (
    ItemID VARCHAR(10) PRIMARY KEY,
    ISSN VARCHAR(20) NOT NULL,
    Volume INT,
    Issue INT,
    CONSTRAINT Journal_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 12. Create Table for Digital_Media
CREATE TABLE Digital_Media (
    ItemID VARCHAR(10) PRIMARY KEY,
    Format VARCHAR(50),
    Size VARCHAR(20),
    CONSTRAINT Digital_Media_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 13. Create Table for Copies
CREATE TABLE Copies (
    CopyID VARCHAR(10) PRIMARY KEY,
    Condition VARCHAR(50),
```

```
    Location VARCHAR(100),
    ItemID VARCHAR(10),
    CONSTRAINT Copies_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 14. Create Table for Users (Superclass)
CREATE TABLE Users (
    UserID VARCHAR(10) PRIMARY KEY,
    First_Name VARCHAR(100) NOT NULL,
    Last_Name VARCHAR(100),
    Email VARCHAR(100),
    Phone_Number VARCHAR(15),
    ReservationTransactionID VARCHAR(10),
    BorrowTransactionID VARCHAR(10),
    CONSTRAINT Users_Trans_Reser_fk FOREIGN KEY (ReservationTransactionID) REFERENCES
Reserve_Transaction(TransactionID) ON DELETE SET NULL,
    CONSTRAINT Users_Trans_Bor_fk FOREIGN KEY (BorrowTransactionID) REFERENCES
Borrow_Transaction(TransactionID) ON DELETE SET NULL
);

-- 15. Create Table for Students (Subclass)
CREATE TABLE Students (
    UserID VARCHAR(10) PRIMARY KEY NOT NULL,
    Year_of_study INT,
    Major VARCHAR(100),
    CONSTRAINT Students_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- 16. Create Table for Faculty (Subclass)
CREATE TABLE Faculty (
    UserID VARCHAR(10) PRIMARY KEY NOT NULL,
    Department VARCHAR(100),
    Title VARCHAR(50),
    CONSTRAINT Faculty_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- 17. Create Table for Staff (Subclass)
CREATE TABLE Staff (
    UserID VARCHAR(10) PRIMARY KEY NOT NULL,
    Position VARCHAR(100),
    Offices VARCHAR(50),
    CONSTRAINT Staff_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- 18. Create Table for Borrowing
CREATE TABLE Borrowing (
    UserID VARCHAR(10) NOT NULL,
    ItemID VARCHAR(10) NOT NULL,
    PRIMARY KEY (UserID, ItemID),
    CONSTRAINT Borrowing_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID),
    CONSTRAINT Borrowing_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);

-- 19. Create Table for Reserving
CREATE TABLE Reserving (
    UserID VARCHAR(10) NOT NULL,
    ItemID VARCHAR(10) NOT NULL,
    PRIMARY KEY (UserID, ItemID),
    CONSTRAINT Reserving_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID),
    CONSTRAINT Reserving_Item_fk FOREIGN KEY (ItemID) REFERENCES Items(ItemID)
);
```

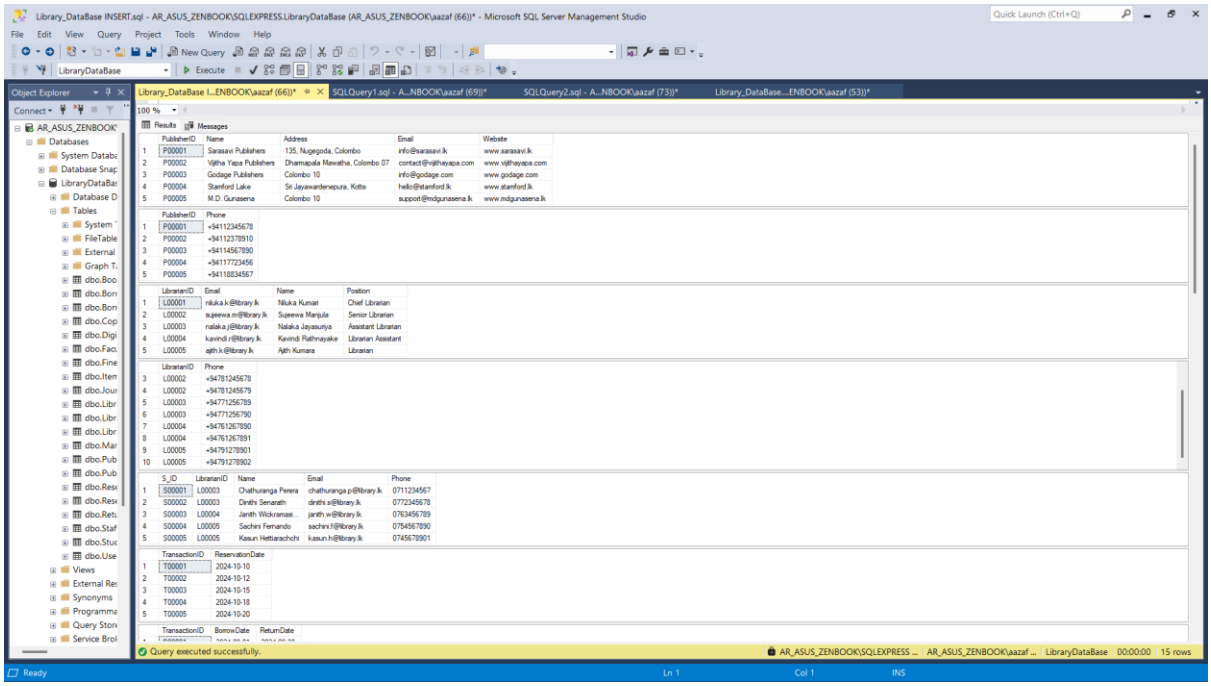
```sql
-- 20. Create Table for Returning_Books
CREATE TABLE Returning_Books (
    UserID VARCHAR(10),
    TransactionID VARCHAR(10),
    PRIMARY KEY (UserID, TransactionID),
    CONSTRAINT Returning_Books_User_fk FOREIGN KEY (UserID) REFERENCES Users(UserID),
    CONSTRAINT Returning_Books_Trans_Borr_fk FOREIGN KEY (TransactionID) REFERENCES
Borrow_Transaction(TransactionID)
);

-- 21. Create Table for Fine
CREATE TABLE Fine (
    FineID VARCHAR(10) PRIMARY KEY,
    Date DATE,
    Amount FLOAT,
    UserID VARCHAR(10),
    TransactionID VARCHAR(10),
    CONSTRAINT Fine_fk FOREIGN KEY (UserID, TransactionID) REFERENCES
Returning_Books(UserID, TransactionID)
);



-- 1. Insert Data into Publisher
INSERT INTO Publisher (PublisherID, Name, Address, Email, Website) VALUES
('P00001', 'Sarasavi Publishers', '135, Nugegoda, Colombo', 'info@sarasavi.lk',
'www.sarasavi.lk'),
('P00002', 'Vijitha Yapa Publishers', 'Dharmapala Mawatha, Colombo 07',
'contact@vijithayapa.com', 'www.vijithayapa.com'),
('P00003', 'Godage Publishers', 'Colombo 10', 'info@godage.com', 'www.godage.com'),
('P00004', 'Stamford Lake', 'Sri Jayawardenepura, Kotte', 'hello@stamford.lk',
'www.stamford.lk'),
('P00005', 'M.D. Gunasena', 'Colombo 10', 'support@mdgunasena.lk',
'www.mdgunasena.lk');


-- 2. Insert Data into Publisher_Phone (Multivalued)
INSERT INTO Publisher_Phone (PublisherID, Phone) VALUES
('P00001', '+94112345678'),
('P00002', '+94112378910'),
('P00003', '+94114567890'),
('P00004', '+94117723456'),
('P00005', '+94118834567');


-- 3. Insert Data into Librarian
INSERT INTO Librarian (LibrarianID, Email, Name, Position) VALUES
('L00001', 'niluka.k@library.lk', 'Niluka Kumari', 'Chief Librarian'),
('L00002', 'sujeewa.m@library.lk', 'Sujeewa Manjula', 'Senior Librarian'),
('L00003', 'nalaka.j@library.lk', 'Nalaka Jayasuriya', 'Assistant Librarian'),
('L00004', 'kavindi.r@library.lk', 'Kavindi Rathnayake', 'Librarian Assistant'),
('L00005', 'ajith.k@library.lk', 'Ajith Kumara', 'Librarian');


-- 4. Insert Data into Librarian_Phone (Multivalued)
INSERT INTO Librarian_Phone (LibrarianID, Phone) VALUES
('L00001', '+94711234567'),
('L00001', '+94711234568'),
('L00002', '+94781245678'),
('L00002', '+94781245679'),
('L00003', '+94771256789'),
('L00003', '+94771256790'),
```

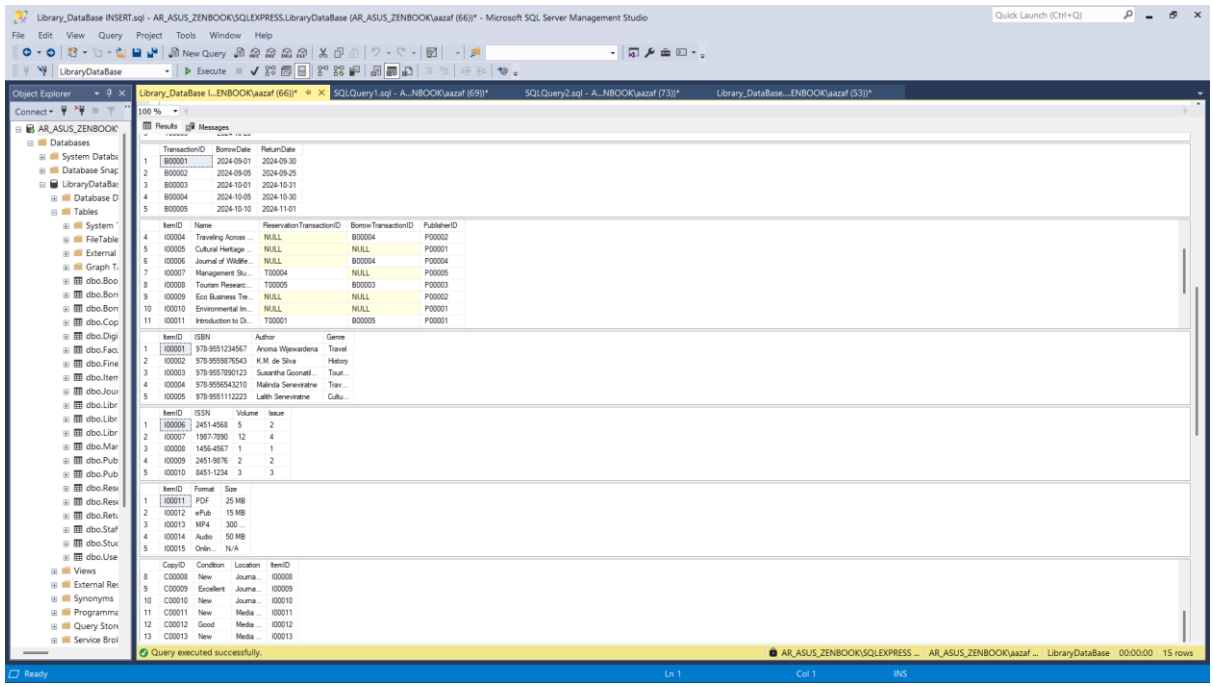
```
('L00004', '+94761267890'),
('L00004', '+94761267891'),
('L00005', '+94791278901'),
('L00005', '+94791278902');


-- 5. Insert Data into Library_Assistants
INSERT INTO Library_Assistants (S_ID, LibrarianID, Name, Email, Phone) VALUES
('S00001', 'L00003', 'Chathuranga Perera', 'chathuranga.p@library.lk', '0711234567'),
('S00002', 'L00003', 'Dinithi Senarath', 'dinithi.s@library.lk', '0772345678'),
('S00003', 'L00004', 'Janith Wickramasinghe', 'janith.w@library.lk', '0763456789'),
('S00004', 'L00005', 'Sachini Fernando', 'sachini.f@library.lk', '0754567890'),
('S00005', 'L00005', 'Kasun Hettiarachchi', 'kasun.h@library.lk', '0745678901');


-- 6. Insert Data into Reserve_Transaction
INSERT INTO Reserve_Transaction (TransactionID, ReservationDate) VALUES
('T00001', '2024-10-10'),
('T00002', '2024-10-12'),
('T00003', '2024-10-15'),
('T00004', '2024-10-18'),
('T00005', '2024-10-20');


-- 7. Insert Data into Borrow_Transaction
INSERT INTO Borrow_Transaction (TransactionID, BorrowDate, ReturnDate) VALUES
('B00001', '2024-09-01', '2024-09-30'),
('B00002', '2024-09-05', '2024-09-25'),
('B00003', '2024-10-01', '2024-10-31'),
('B00004', '2024-10-05', '2024-10-30'),
('B00005', '2024-10-10', '2024-11-01');


-- 8. Insert into Items Table
INSERT INTO Items (ItemID, Name, ReservationTransactionID, BorrowTransactionID,
PublisherID) VALUES
-- Books
('I00001', 'The Road to Kandy', 'T00001', 'B00001', 'P00001'),
('I00002', 'Sri Lanka: A History', 'T00002', 'B00002', 'P00002'),
('I00003', 'Exploring Anuradhapura', 'T00003', 'B00003', 'P00003'),
('I00004', 'Traveling Across Sri Lanka', NULL, 'B00004', 'P00002'),
('I00005', 'Cultural Heritage Guide', NULL, NULL, 'P00001'),

-- Journals
('I00006', 'Journal of Wildlife Research', NULL, 'B00004', 'P00004'),
('I00007', 'Management Studies Journal', 'T00004', NULL, 'P00005'),
('I00008', 'Tourism Research Monthly', 'T00005', 'B00003', 'P00003'),
('I00009', 'Eco Business Trends', NULL, NULL, 'P00002'),
('I00010', 'Environmental Impact Studies', NULL, NULL, 'P00001'),

-- Digital Media
('I00011', 'Introduction to Digital Media', 'T00001', 'B00005', 'P00001'),
('I00012', 'Sri Lankan Recipes eBook', NULL, NULL, 'P00003'),
('I00013', 'Educational Video on History', 'T00002', NULL, 'P00002'),
('I00014', 'Audio Guide to Polonnaruwa', NULL, 'B00003', 'P00004'),
('I00015', 'Environmental Studies Online', NULL, NULL, 'P00005');

-- 9. Insert into  Book Table
INSERT INTO Book (ItemID, ISBN, Author, Genre) VALUES
('I00001', '978-9551234567', 'Anoma Wijewardena', 'Travel'),
('I00002', '978-9559876543', 'K.M. de Silva', 'History'),
('I00003', '978-9557890123', 'Susantha Goonatilake', 'Tourism'),
```

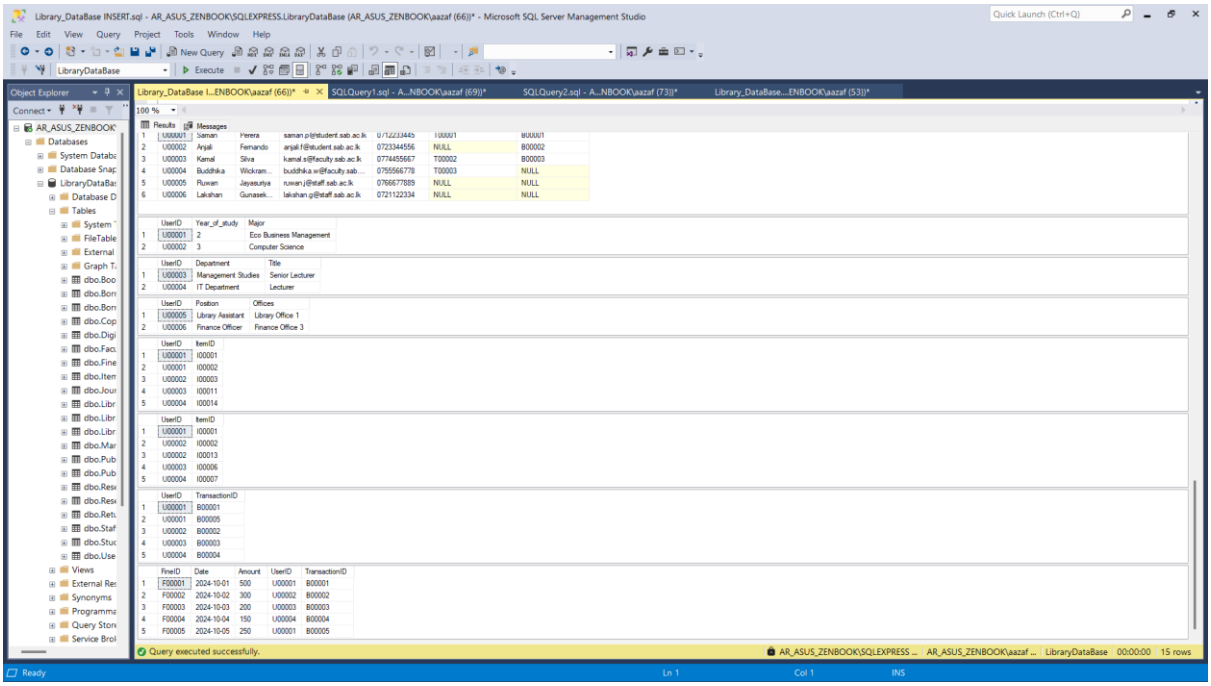
```
('I00004', '978-9556543210', 'Malinda Seneviratne', 'Travel Guide'),
('I00005', '978-9551112223', 'Lalith Seneviratne', 'Cultural Studies');

-- 10. Insert into  Journal Table
INSERT INTO Journal (ItemID, ISSN, Volume, Issue) VALUES
('I00006', '2451-4568', 5, 2),
('I00007', '1987-7890', 12, 4),
('I00008', '1456-4567', 1, 1),
('I00009', '2451-9876', 2, 2),
('I00010', '8451-1234', 3, 3);

-- 11. Insert into  Digital_Media Table
INSERT INTO Digital_Media (ItemID, Format, Size) VALUES
('I00011', 'PDF', '25 MB'),
('I00012', 'ePub', '15 MB'),
('I00013', 'MP4', '300 MB'),
('I00014', 'Audio', '50 MB'),
('I00015', 'Online Access', 'N/A');

-- 12. Insert into  Copies Table
INSERT INTO Copies (CopyID, Condition, Location, ItemID) VALUES
-- Copies of Books
('C00001', 'New', 'Shelf A1', 'I00001'), -- The Road to Kandy
('C00002', 'Good', 'Shelf A2', 'I00002'), -- Sri Lanka: A History
('C00003', 'Fair', 'Shelf B1', 'I00003'), -- Exploring Anuradhapura
('C00004', 'Excellent', 'Shelf C1', 'I00004'), -- Traveling Across Sri Lanka
('C00005', 'New', 'Shelf D2', 'I00005'), -- Cultural Heritage Guide

-- Copies of Journals
('C00006', 'Good', 'Journal Section 1', 'I00006'), -- Journal of Wildlife Research
('C00007', 'Fair', 'Journal Section 2', 'I00007'), -- Management Studies Journal
('C00008', 'New', 'Journal Section 3', 'I00008'), -- Tourism Research Monthly
('C00009', 'Excellent', 'Journal Section 4', 'I00009'), -- Eco Business Trends
('C00010', 'New', 'Journal Section 5', 'I00010'), -- Environmental Impact Studies

-- Copies of Digital Media
('C00011', 'New', 'Media Room 1', 'I00011'), -- Introduction to Digital Media
('C00012', 'Good', 'Media Room 2', 'I00012'), -- Sri Lankan Recipes eBook
('C00013', 'New', 'Media Room 3', 'I00013'), -- Educational Video on History
('C00014', 'Fair', 'Media Room 4', 'I00014'), -- Audio Guide to Polonnaruwa
('C00015', 'New', 'Media Room 5', 'I00015'); -- Environmental Studies Online


-- 13. Insert into  Manages Table
INSERT INTO Manages (S_ID, LibrarianID, ItemID) VALUES
-- Books managed by Library Assistants
('S00001', 'L00003', 'I00001'), -- The Road to Kandy
('S00002', 'L00003', 'I00002'), -- Sri Lanka: A History
('S00001', 'L00003', 'I00003'), -- Exploring Anuradhapura
('S00003', 'L00004', 'I00004'), -- Traveling Across Sri Lanka
('S00003', 'L00004', 'I00005'), -- Cultural Heritage Guide

-- Journals managed by Library Assistants
('S00002', 'L00003', 'I00006'), -- Journal of Wildlife Research
('S00003', 'L00004', 'I00007'), -- Management Studies Journal
('S00004', 'L00005', 'I00008'), -- Tourism Research Monthly
('S00004', 'L00005', 'I00009'), -- Eco Business Trends
('S00005', 'L00005', 'I00010'), -- Environmental Impact Studies

-- Digital Media managed by Library Assistants
('S00001', 'L00003', 'I00011'), -- Introduction to Digital Media
('S00002', 'L00003', 'I00012'), -- Sri Lankan Recipes eBook
```

```
('S00003', 'L00004', 'I00013'), -- Educational Video on History
('S00004', 'L00005', 'I00014'), -- Audio Guide to Polonnaruwa
('S00005', 'L00005', 'I00015'); -- Environmental Studies Online

-- 14. Insert into  Users Table
INSERT INTO Users (UserID, First_Name, Last_Name, Email, Phone_Number,
ReservationTransactionID, BorrowTransactionID) VALUES
-- Students
('U00001', 'Saman', 'Perera', 'saman.p@student.sab.ac.lk', '0712233445', 'T00001',
'B00001'),
('U00002', 'Anjali', 'Fernando', 'anjali.f@student.sab.ac.lk', '0723344556', NULL,
'B00002'),

-- Faculty
('U00003', 'Kamal', 'Silva', 'kamal.s@faculty.sab.ac.lk', '0774455667', 'T00002',
'B00003'),
('U00004', 'Buddhika', 'Wickramasinghe', 'buddhika.w@faculty.sab.ac.lk', '0755566778',
'T00003', NULL),

-- Staff
('U00005', 'Ruwan', 'Jayasuriya', 'ruwan.j@staff.sab.ac.lk', '0766677889', NULL,
NULL),
('U00006', 'Lakshan', 'Gunasekara', 'lakshan.g@staff.sab.ac.lk', '0721122334', NULL,
NULL);

-- 15. Insert into  Student Table
INSERT INTO Students (UserID, Year_of_study, Major) VALUES
('U00001', 2, 'Eco Business Management'),
('U00002', 3, 'Computer Science');

-- 16. Insert into  Faculty Table
INSERT INTO Faculty (UserID, Department, Title) VALUES
('U00003', 'Management Studies', 'Senior Lecturer'),
('U00004', 'IT Department', 'Lecturer');

-- 17. Insert into  Staff Table
INSERT INTO Staff (UserID, Position, Offices) VALUES
('U00005', 'Library Assistant', 'Library Office 1'),
('U00006', 'Finance Officer', 'Finance Office 3');

-- 18. Insert into  Borrowing Table
INSERT INTO Borrowing (UserID, ItemID) VALUES
('U00001', 'I00001'), -- Saman borrowed "The Road to Kandy"
('U00001', 'I00002'), -- Saman borrowed "Sri Lanka: A History"
('U00002', 'I00003'), -- Anjali borrowed "Exploring Anuradhapura"
('U00003', 'I00011'), -- Kamal borrowed "Introduction to Digital Media"
('U00004', 'I00014'); -- Buddhika borrowed "Audio Guide to Polonnaruwa"

-- 19. Insert into  Reserving Table
INSERT INTO Reserving (UserID, ItemID) VALUES
('U00001', 'I00001'), -- Saman reserved "The Road to Kandy"
('U00002', 'I00002'), -- Anjali reserved "Sri Lanka: A History"
('U00003', 'I00006'), -- Kamal reserved "Journal of Wildlife Research"
('U00004', 'I00007'), -- Buddhika reserved "Management Studies Journal"
('U00002', 'I00013'); -- Anjali reserved "Educational Video on History"

-- 20. Insert into  Returning_Books Table
INSERT INTO Returning_Books (UserID, TransactionID) VALUES
('U00001', 'B00001'), -- Saman returned "The Road to Kandy"
('U00002', 'B00002'), -- Anjali returned "Sri Lanka: A History"
('U00003', 'B00003'), -- Kamal returned "Exploring Anuradhapura"
('U00004', 'B00004'), -- Buddhika returned "Traveling Across Sri Lanka"
```

```
('U00001', 'B00005'); -- Saman returned "Introduction to Digital Media"

-- 21. Insert into  Returning_Books Table
INSERT INTO Fine (FineID, Date, Amount, UserID, TransactionID) VALUES
('F00001', '2024-10-01', 500.00, 'U00001', 'B00001'), -- Fine for Saman
('F00002', '2024-10-02', 300.00, 'U00002', 'B00002'), -- Fine for Anjali
('F00003', '2024-10-03', 200.00, 'U00003', 'B00003'), -- Fine for Kamal
('F00004', '2024-10-04', 150.00, 'U00004', 'B00004'), -- Fine for Buddhika
('F00005', '2024-10-05', 250.00, 'U00001', 'B00005'); -- Fine for Saman


--Checking the tables
SELECT * FROM Publisher;
SELECT * FROM Publisher_Phone;
SELECT * FROM Librarian;
SELECT * FROM Librarian_Phone;
SELECT * FROM Library_Assistants;
SELECT * FROM Reserve_Transaction;
SELECT * FROM Borrow_Transaction;
SELECT * FROM Items;
SELECT * FROM Book;
SELECT * FROM Journal;
SELECT * FROM Digital_Media;
SELECT * FROM Copies;
SELECT * FROM Manages;
SELECT * FROM Users;
SELECT * FROM Students;
SELECT * FROM Faculty;
SELECT * FROM Staff;
SELECT * FROM Borrowing;
SELECT * FROM Reserving;
SELECT * FROM Returning_Books;
SELECT * FROM Fine;
```


Library_DataBase INSERT.sql - AR_ASUS_ZENBOOK\SQLEXPRESS.LibraryDataBase (AR_ASUS_ZENBOOK\aazaf (66))* - Microsoft SQL Server Management Studio
Query executed successfully.


Library_DataBase INSERT.sql - AR_ASUS_ZENBOOK\SQLEXPRESS.LibraryDataBase (AR_ASUS_ZENBOOK\aazaf (66))* - Microsoft SQL Server Management Studio
Query executed successfully.

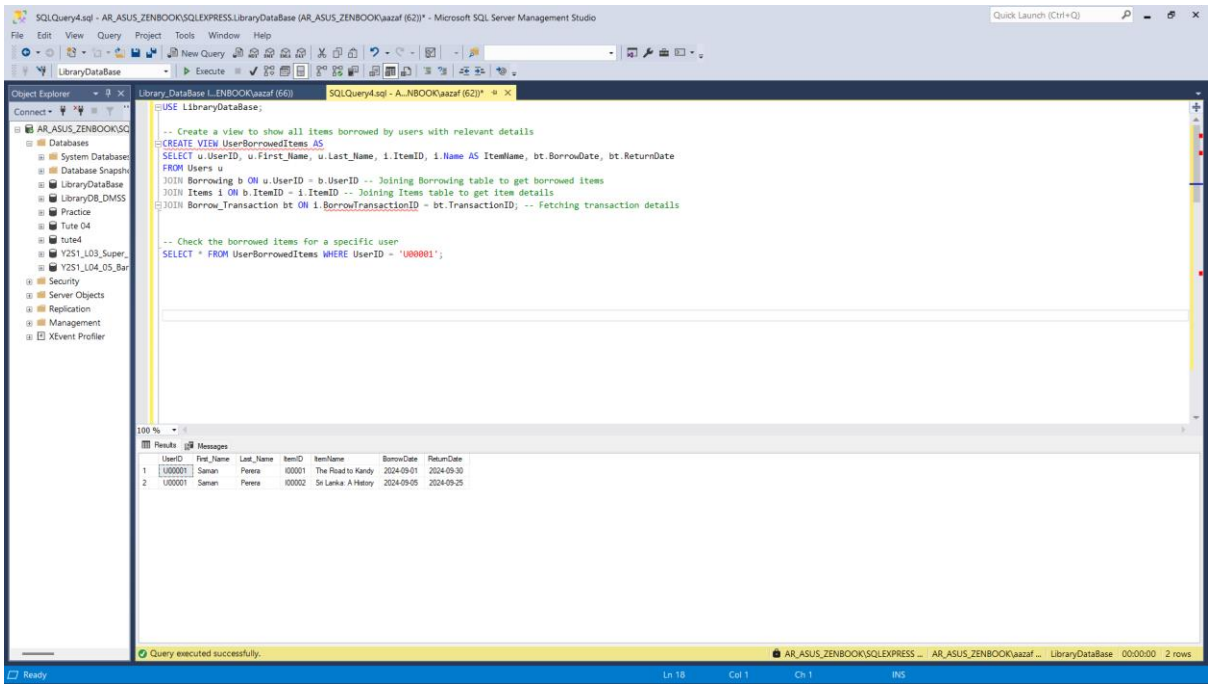

Library_DataBase INSERT.sql - AR_ASUS_ZENBOOK\SQLEXPRESS.LibraryDataBase (AR_ASUS_ZENBOOK\aazaf (66))* - Microsoft SQL Server Management Studio
Query executed successfully.
Ready

5. Views, Functions, Procedures, Triggers and Indexes

1.1 Views

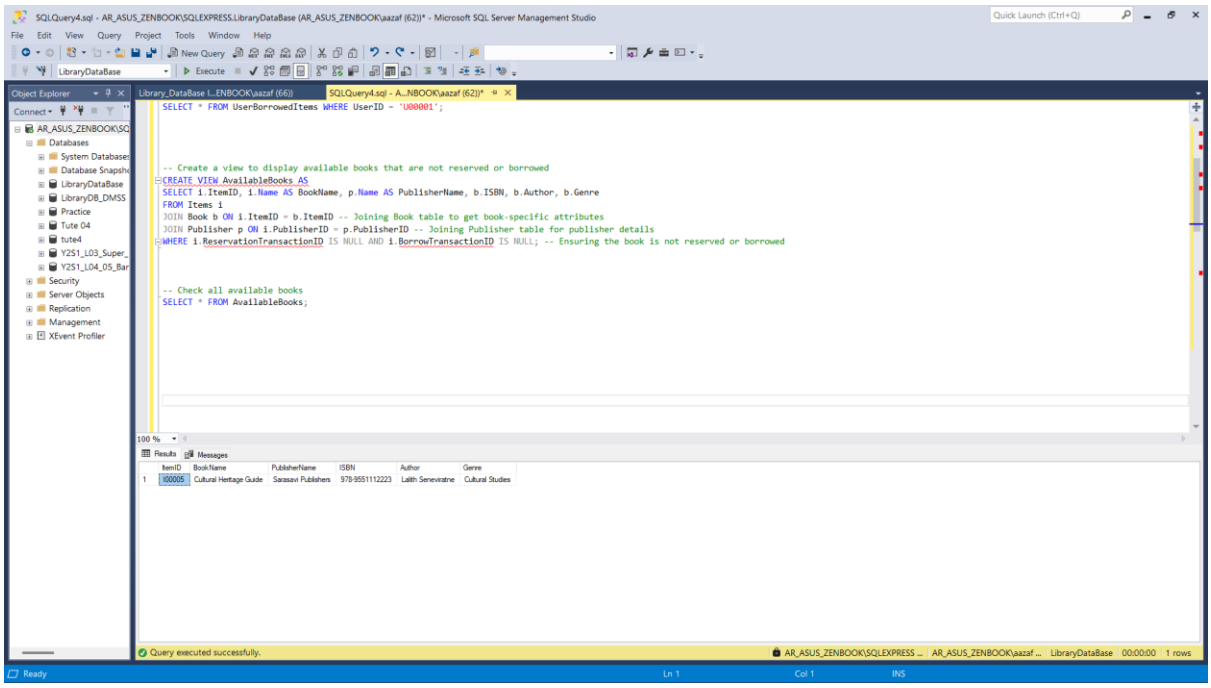
Creating a View for Borrowed Items by a User

```
-- Create a view to show all items borrowed by users with relevant details
CREATE VIEW UserBorrowedItems AS
SELECT u.UserID, u.First_Name, u.Last_Name, i.ItemID, i.Name AS ItemName,
bt.BorrowDate, bt.ReturnDate
FROM Users u
JOIN Borrowing b ON u.UserID = b.UserID -- Joining Borrowing table to get borrowed
items
JOIN Items i ON b.ItemID = i.ItemID -- Joining Items table to get item details
JOIN Borrow_Transaction bt ON i.BorrowTransactionID = bt.TransactionID; -- Fetching
transaction details


-- Check the borrowed items for a specific user
SELECT * FROM UserBorrowedItems WHERE UserID = 'U00001';
```



Create a View for Available Books

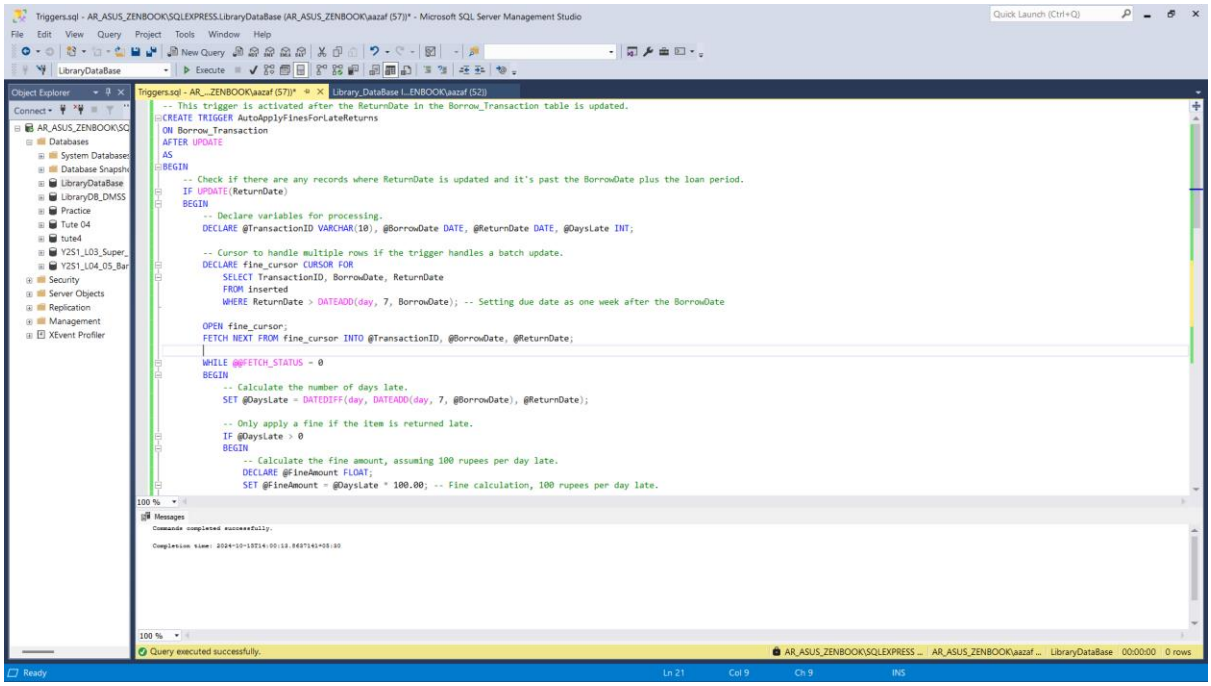
```
-- Create a view to display available books that are not reserved or borrowed
CREATE VIEW AvailableBooks AS
SELECT i.ItemID, i.Name AS BookName, p.Name AS PublisherName, b.ISBN, b.Author,
b.Genre
FROM Items i
JOIN Book b ON i.ItemID = b.ItemID -- Joining Book table to get book-specific
attributes
JOIN Publisher p ON i.PublisherID = p.PublisherID -- Joining Publisher table for
publisher details
WHERE i.ReservationTransactionID IS NULL AND i.BorrowTransactionID IS NULL; --
Ensuring the book is not reserved or borrowed



-- Check all available books
SELECT * FROM AvailableBooks;
```



1.2 Triggers

Trigger 1: Late Return Fine Trigger

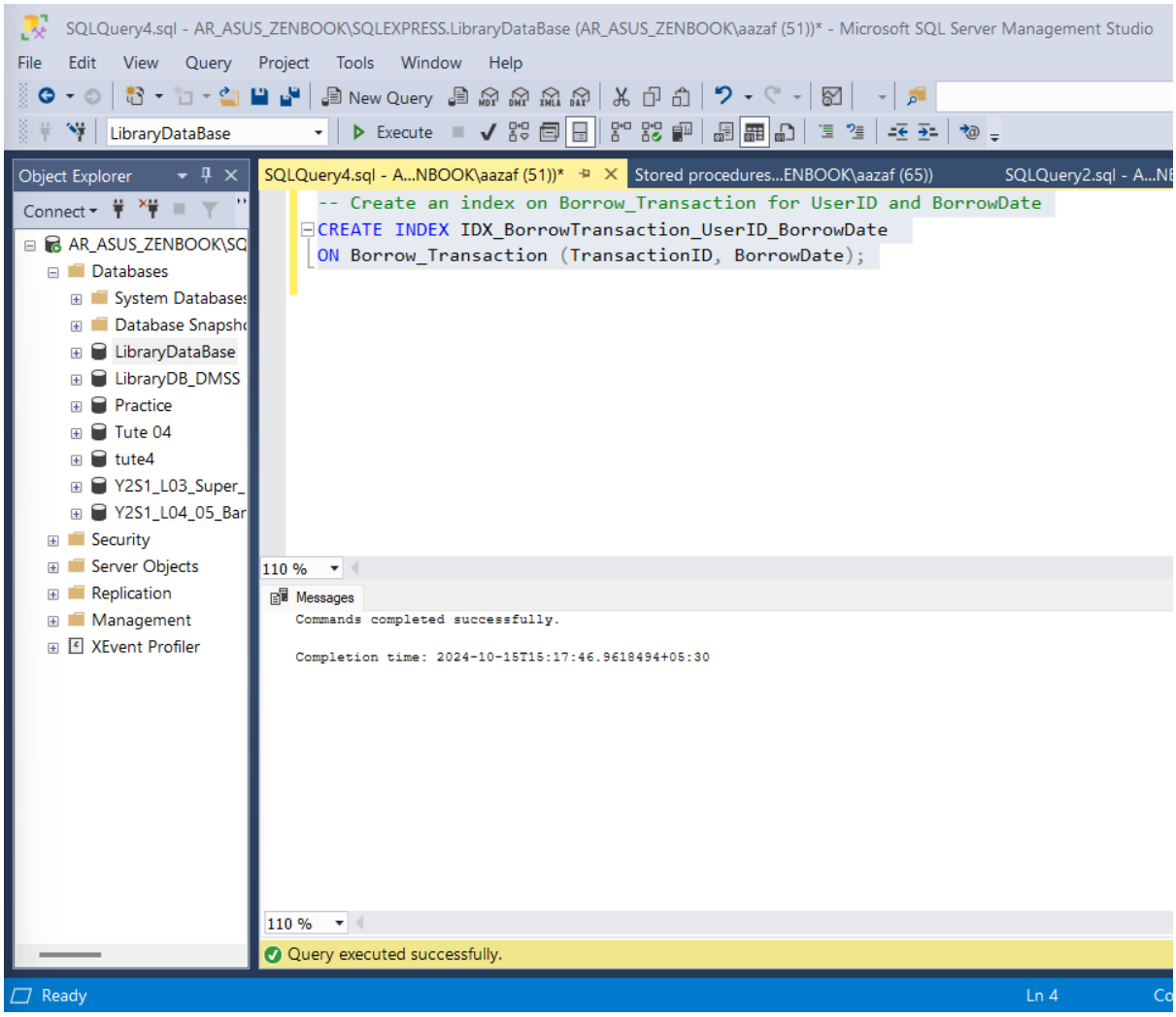
```
-- This trigger is activated after the ReturnDate in the Borrow_Transaction table is
updated.
CREATE TRIGGER AutoApplyFinesForLateReturns
ON Borrow_Transaction
AFTER UPDATE
AS
BEGIN
    -- Check if there are any records where ReturnDate is updated and it's past the
BorrowDate plus the loan period.
    IF UPDATE(ReturnDate)
    BEGIN
        -- Declare variables for processing.
        DECLARE @TransactionID VARCHAR(10), @BorrowDate DATE, @ReturnDate DATE,
@DaysLate INT;

        -- Cursor to handle multiple rows if the trigger handles a batch update.
        DECLARE fine_cursor CURSOR FOR
            SELECT TransactionID, BorrowDate, ReturnDate
            FROM inserted
            WHERE ReturnDate > DATEADD(day, 7, BorrowDate); -- Setting due date as one
week after the BorrowDate

        OPEN fine_cursor;
        FETCH NEXT FROM fine_cursor INTO @TransactionID, @BorrowDate, @ReturnDate;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Calculate the number of days late.
            SET @DaysLate = DATEDIFF(day, DATEADD(day, 7, @BorrowDate), @ReturnDate);

            -- Only apply a fine if the item is returned late.
            IF @DaysLate > 0
            BEGIN
                -- Calculate the fine amount, assuming 100 rupees per day late.
                DECLARE @FineAmount FLOAT;
                SET @FineAmount = @DaysLate * 100.00; -- Fine calculation, 100 rupees
per day late.

                -- Insert the fine into the Fine table.
                INSERT INTO Fine (TransactionID, Date, Amount)
                VALUES (@TransactionID, GETDATE(), @FineAmount);
            END

            FETCH NEXT FROM fine_cursor INTO @TransactionID, @BorrowDate, @ReturnDate;
        END;

        CLOSE fine_cursor;
        DEALLOCATE fine_cursor;
    END
END;
```


Triggers.sql - AR_ASUS_ZENBOOK\SQLEXPRESS.LibraryDataBase (AR_ASUS_ZENBOOK\aazaf (57))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
Execute
LibraryDataBase
Object Explorer
Connect
AR_ASUS_ZENBOOK\SQ
Databases
System Databases
Database Snapsh
LibraryDataBase
LibraryDB_DMSS
Practice
Tute 04
tute4
Y2S1_L03_Super_
Y2S1_L04_05_Bar
Security
Server Objects
Replication
Management
XEvent Profiler
Triggers.sql - AR_...ZENBOOK\aazaf (57))*
Library_DataBase I...ENBOOK\aazaf (52))
-- This trigger is activated after the ReturnDate in the Borrow_Transaction table is updated.
CREATE TRIGGER AutoApplyFinesForLateReturns
ON Borrow_Transaction
AFTER UPDATE
AS
BEGIN
-- Check if there are any records where ReturnDate is updated and it's past the BorrowDate plus the loan period.
IF UPDATE(ReturnDate)
BEGIN
-- Declare variables for processing.
DECLARE @TransactionID VARCHAR(10), @BorrowDate DATE, @ReturnDate DATE, @DaysLate INT;
-- Cursor to handle multiple rows if the trigger handles a batch update.
DECLARE fine_cursor CURSOR FOR
SELECT TransactionID, BorrowDate, ReturnDate
FROM inserted
WHERE ReturnDate > DATEADD(day, 7, BorrowDate); -- Setting due date as one week after the BorrowDate
OPEN fine_cursor;
FETCH NEXT FROM fine_cursor INTO @TransactionID, @BorrowDate, @ReturnDate;
WHILE @@FETCH_STATUS = 0
BEGIN
-- Calculate the number of days late.
SET @DaysLate = DATEDIFF(day, DATEADD(day, 7, @BorrowDate), @ReturnDate);
-- Only apply a fine if the item is returned late.
IF @DaysLate > 0
BEGIN
-- Calculate the fine amount, assuming 100 rupees per day late.
DECLARE @FineAmount FLOAT;
SET @FineAmount = @DaysLate * 100.00; -- Fine calculation, 100 rupees per day late.
100 %
Messages
Commands completed successfully.
Completion time: 2024-10-18T14:00:13.8687141+05:30
100 %
Query executed successfully.
AR_ASUS_ZENBOOK\SQLEXPRESS ...
AR_ASUS_ZENBOOK\aazaf ...
LibraryDataBase
00:00:00
0 rows
Ready
Ln 21
Col 9
Ch 9
INS

1.3 Indexes

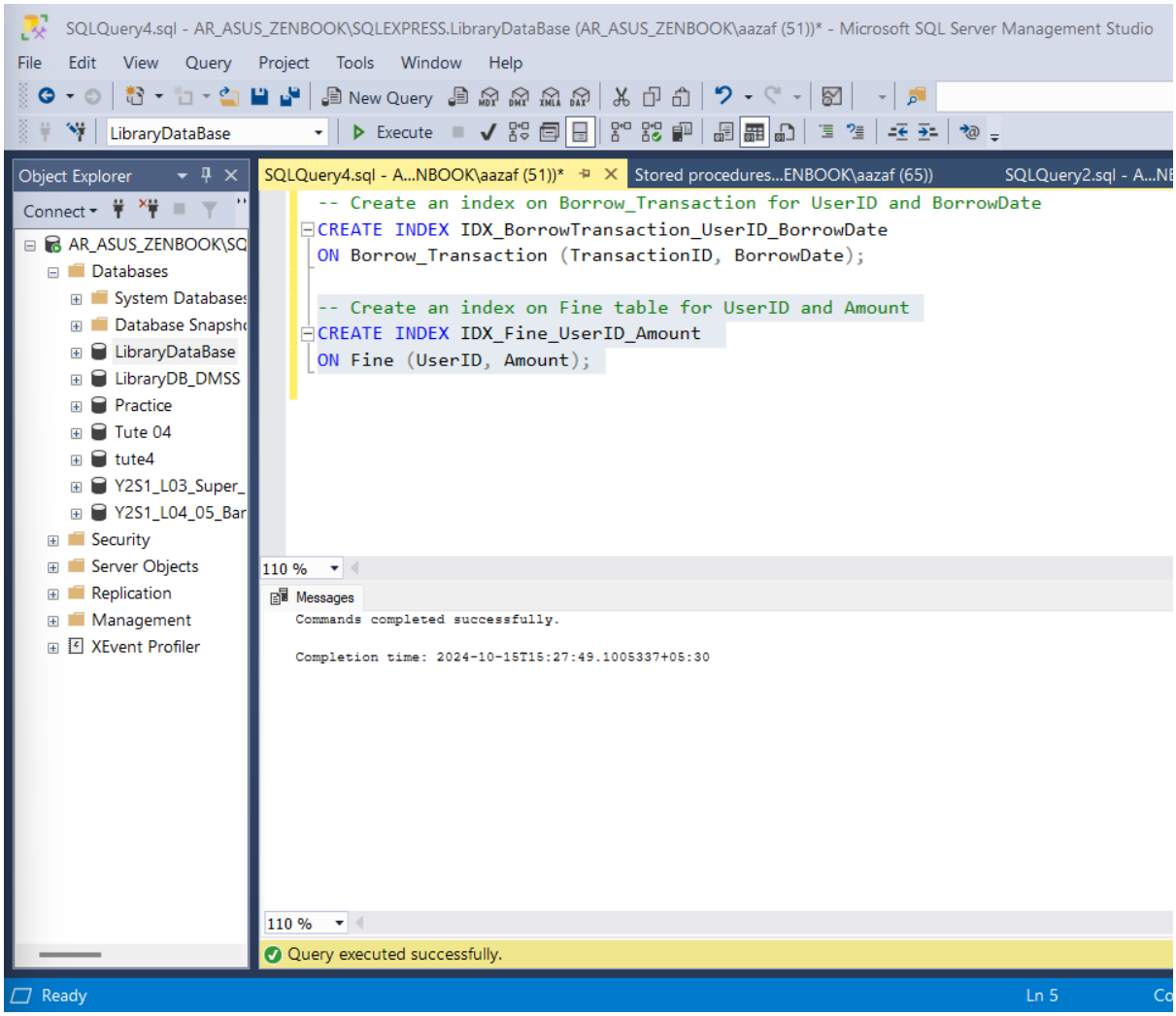
01. Borrow Date Exists in Borrow_Transaction Table

```
CREATE INDEX IDX_BorrowTransaction_UserID_BorrowDate
ON Borrow_Transaction (TransactionID, BorrowDate);
```



02.Index on Fine Table for UserID and Amount

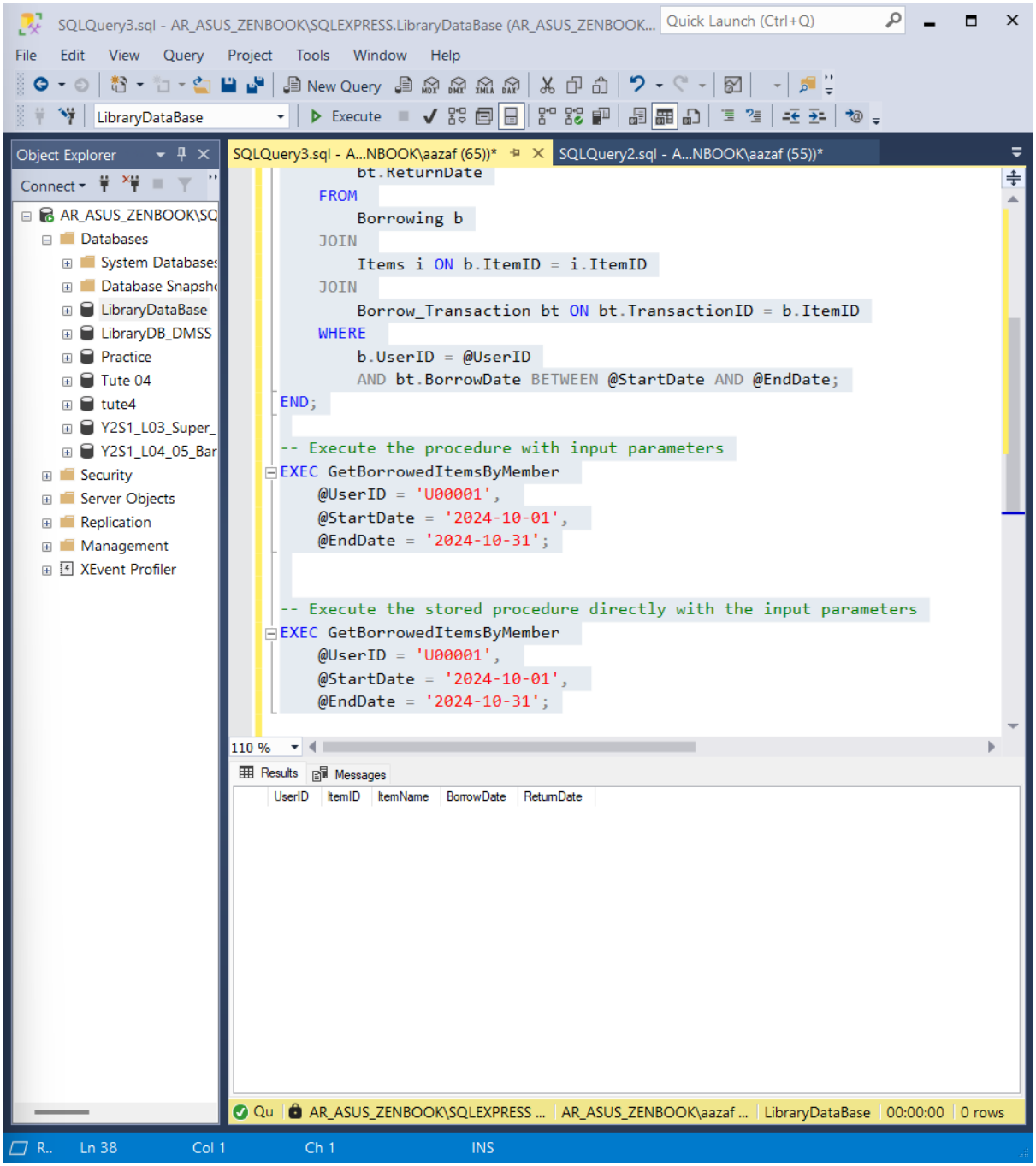
```
CREATE INDEX IDX_Fine_UserID_Amount
ON Fine (UserID, Amount);
```


SQLQuery4.sql - AR_ASUS_ZENBOOK\SQLEXPRESS.LibraryDataBase (AR_ASUS_ZENBOOK\aazaf (51))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
LibraryDataBase
Execute
Object Explorer
Connect
AR_ASUS_ZENBOOK\SQ
Databases
System Databases
Database Snapsh
LibraryDataBase
LibraryDB_DMSS
Practice
Tute 04
tute4
Y2S1_L03_Super_
Y2S1_L04_05_Bar
Security
Server Objects
Replication
Management
XEvent Profiler
SQLQuery4.sql - A...NBOOK\aazaf (51))*
Stored procedures...ENBOOK\aazaf (65))
SQLQuery2.sql - A...NB
-- Create an index on Borrow_Transaction for UserID and BorrowDate
CREATE INDEX IDX_BorrowTransaction_UserID_BorrowDate
ON Borrow_Transaction (TransactionID, BorrowDate);
-- Create an index on Fine table for UserID and Amount
CREATE INDEX IDX_Fine_UserID_Amount
ON Fine (UserID, Amount);
110 %
Messages
Commands completed successfully.
Completion time: 2024-10-15T15:27:49.1005337+05:30
110 %
Query executed successfully.
Ready
Ln 5

1.4 Stored Procedures

01. Stored Procedure: Get Borrowed Items by a Member within a Given Period

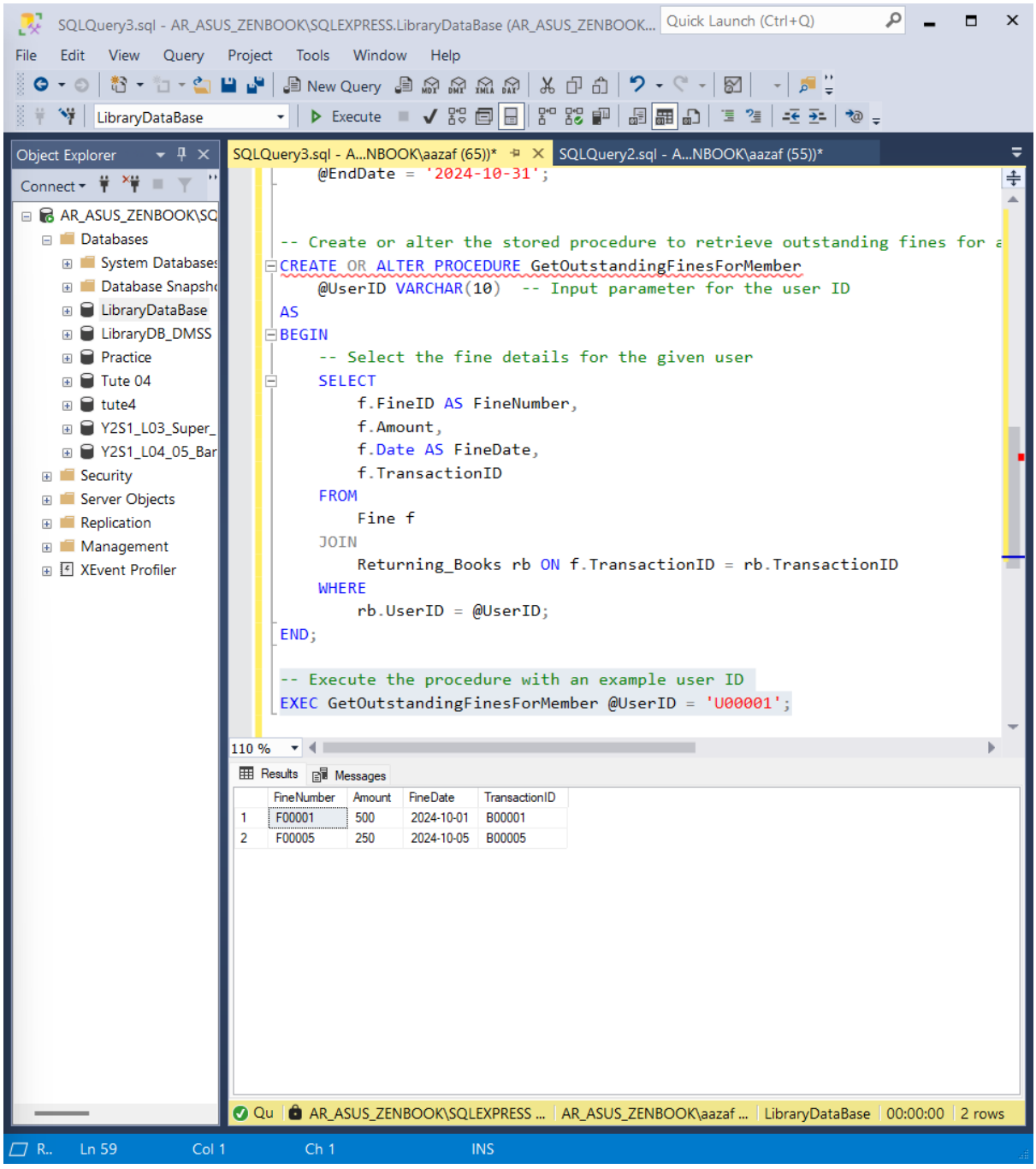
```
-- Create or alter the stored procedure to retrieve borrowed items by a member within
a specific period
CREATE OR ALTER PROCEDURE GetBorrowedItemsByMember
    @UserID VARCHAR(10),      -- Input parameter for the user ID
    @StartDate DATE,          -- Start date for the borrowing period
    @EndDate DATE             -- End date for the borrowing period
AS
BEGIN
    -- Select the borrowed items with relevant details
    SELECT
        b.UserID,
        i.ItemID,
        i.Name AS ItemName,
        bt.BorrowDate,
        bt.ReturnDate
    FROM
        Borrowing b
    JOIN
        Items i ON b.ItemID = i.ItemID
    JOIN
        Borrow_Transaction bt ON bt.TransactionID = b.ItemID
    WHERE
        b.UserID = @UserID
        AND bt.BorrowDate BETWEEN @StartDate AND @EndDate;
END;

-- Execute the procedure with input parameters
EXEC GetBorrowedItemsByMember
    @UserID = 'U00001',
    @StartDate = '2024-10-01',
    @EndDate = '2024-10-31';


-- Execute the stored procedure directly with the input parameters
EXEC GetBorrowedItemsByMember
    @UserID = 'U00001',
    @StartDate = '2024-10-01',
    @EndDate = '2024-10-31';
```



02. Stored Procedure: Get Outstanding Fines for a Member

```
-- Create or alter the stored procedure to retrieve outstanding fines for a member
CREATE OR ALTER PROCEDURE GetOutstandingFinesForMember
    @UserID VARCHAR(10)  -- Input parameter for the user ID
AS
BEGIN
    -- Select the fine details for the given user
    SELECT
        f.FineID AS FineNumber,
        f.Amount,
        f.Date AS FineDate,
        f.TransactionID
```

```
    FROM
        Fine f
    JOIN
        Returning_Books rb ON f.TransactionID = rb.TransactionID
    WHERE
        rb.UserID = @UserID;
END;

-- Execute the procedure with an example user ID
EXEC GetOutstandingFinesForMember @UserID = 'U00001';
```



Database Security Vulnerabilities and Countermeasures

6. Database Vulnerabilities

A Database is a collection of related data. Database security is the processes, tools, and controls that secure databases against accidental and intentional threats. Database security systems are created to safeguard against misuse, damage, and intrusion not just the data contained inside the database but also the data management system as a whole and any application that utilizes it.

There are so many database vulnerabilities. Among them are SQL injections, Weak Authentication and Authorization (Privilege Escalation), Denial-of-Service attacks, Buffer overflows, and Extensive user and group privileges, etc. are the most common and prominent vulnerabilities in a database system.

SQL Injection (SQLi)

I. Introduction

SQL injection is a very dangerous and common web application security vulnerability. This SQL injection vulnerability allows attackers to manipulate SQL queries that a database executes. So as well this vulnerability also gives unauthorized access to data, manipulates or deletes data, and gains administrative control over the database.

By inserting malicious SQL statements into an entry field for execution, the SQL injection technique in computing is used to exploit data-driven applications (e.g., to dump the database contents to the attacker). SQL injections must take advantage of a security flaw in the software of the application, such as when user input is either not strongly typed and unexpectedly executed or when user input is erroneously checked for string literal escape characters encoded in SQL statements.

II. Types of SQL injection

There are three primary categories exist for SQL injection (SQLi): Out-of band SQL injection, inferential SQL injection (also known as blind SQLi) and in band SQL injection (also known as classic SQLi). The way an attack is executed, and information is obtained varies depending on the kind.

III. In Band SQL Injection

This kind of sqli is the most basic and prevalent the reason its termed in band is that the attacker conducts the attack and collects data over the same communication channel. It has been appreciated since it is simple to use and highly efficient. There are two basic variations of in band SQLi:

- Error based SQL injection

  Using this technique, the attacker manipulates the database to display error messages that reveal important details about the database structure. attackers may be able to learn more about how to further abuse the system thanks to these error messages.

- Union based SQL injection

  In this instance the attacker joins the result set of multiple SQL statements using the SQL UNION operator. This allows them to retrieve data from several database tables and compile it into a single response.

IV. Inferential SQL injection (Blind SQLi)

In this kind of attack the attacker mounts the server and transmits data payloads to it. He then watches the server's response, but he can't see the precise database result. It was originally referred to as "blind" since the attacker could not wait for a response from the server and had to make assumption about the appearance of a database based on the server's response. Even while blind SQLi might be slower than in band SQLi it is still extremely risky. Two categories of inferential SQLi exist

- Boolean-based Blind SQLi

  The attacker sends a query which returns true or false values. Based on applications response, the attacker determines the validity of the query

- Time-based SQLi

  The attacker sends a query that returns true or false which could be determined by the time it takes for the server to respond. Although this method doesn't depend on visual data, the attacker can determine the success of the attack by the response time delay

V. Out of band SQL Injection

Out-of-band SQLi is used when the attacker can't use the same channel for the attack and gathering results, or when the server is too slow or unstable for the other types of attacks. This method relies on the server's ability to make external DNS or HTTP requests to send information to the attacker. Out of band SQLi is only possible when specific functionalities are enabled on the database server is SQLi possible.

VI. SQL injection examples

There are many types of sql injection vulnerabilities, and techniques that can be used in diverse contexts. Some instances of SQL injection include the following.

- Access hidden data: Modifying a SQL query to show more result than intended
- Altering application behavior: Changing a query to disrupt the regular function of the application
- Using UNION to combine data: Retrieve data from multiple database tables with just one query
- Gathering information about the database: Extracting details like the database version and structure
- Blind SQL injection: Controlling a query without being able to see the results directly in the application's response.

VII. How does SQL injection attack work

An SQL query is a request sent to a database asking it to perform a specific task. Queries can also run commands on the operating system. When a user submits a query, parameters are used to make sure only the relevant data is shown. However, by a technique known as SQL injection attackers exploit this by injecting harmful code into the query through the input form.

The first step in a SQL injection attack is to test how the database works. Attackers do this by entering random data into the query to observe how the server responds. Once they understand the database's behavior, they create a query that the server will interpret as a legitimate SQL command.

For example, a database might store customer details and assign each one a unique ID. Instead of looking for a particular customer ID, an attacker could input something like "CustomerID = 1000 OR 1=1" into the query. The query would return all customer IDs and

relevant information because “1=1” is always true. This gives the attacker access to sensitive data even with administrator privileges and allows them to get past security measures.

In addition to retrieving illegal data, SQL injection attacks can be used to erase the entire database, get around password constraints, change, records or insert malicious material.

7. Mitigation and Countermeasures

The following actions are among the most effective ways to avoid SQL injection (SQLI): The first process is data cleansing, also referred to as data scrubbing or input sanitizing. This means creating programming that can detect incorrect user input. Input validation is always a good practice but it is not perfect. In many situations, it becomes rather challenging to set up a check that would consider all potential valid and invalid input. When it is done, it may result in many false positives, which can both negatively affect a user's experience and, on the other hand, hinder the functioning of an application. Since input validation is not always ideal, a WAF is put in place to filter out SQLI attacks and other online threats. Incoming traffic is compared to a large and ever-growing database of attack patterns by a WAF. The purpose of these signatures is to prevent harmful SQL queries, and the collection of signatures is regularly updated to guard against emerging dangers. To strengthen security measures, modern WAFs are integrated with other security systems. For example, a WAF may make a differential decision to accept or reject inputs based on the idea of IP reputation. Before completely blocking, the WAF may check to see if the IP address linked to the input has a blacklisted string if it doesn't appear to be harmful.

The Imperva cloud-based WAF functions similarly to a WAF but emphasizes methods like IP reputation and signature recognition to prevent SQL injections with few false alarms. It also employs IncapRules as a tool of setting security in a way that special instances can be provided for. This enables users to define the rules that govern specific security needs.

Additionally, Imperva's WAF structure uses crowdsourcing to disseminate information about emerging threats to all users, enabling prompt responses to emerging vulnerabilities and attacks.

Weak Authentication and Authorization (Privilege Escalation)

VIII. Introduction

Weak authentication and authorization problems can result in privilege escalation, which is when an attacker gains access to higher levels of control within a system than they are supposed to have. These vulnerabilities often come from insecure login processes, weak passwords, or poorly set up user roles. Attackers can take advantage of these issues to gain extra permissions, potentially leading to control over sensitive areas or data.

IX. Types of Privilege Escalation

(1) Vertical Privilege Escalation

This happens when an attacker moves up the system and gains more privileges, like going from a regular user to an admin.

(2) Horizontal Privilege Escalation

In this case, the attacker gets access to another user's data or resources, even though they are on the same privilege level, without having the correct permissions.

X. Examples of privilege Escalation

- Vertical Example -: A normal user finds a way to access files meant only for the system admin.
- Horizontal Example -: A customer gets access to another customer's data through poorly protected user ID settings.

XI. How does a Privilege Escalation work?

- Finding Weak Spots in Authentication -: Attackers discover weaknesses in the login system, such as using weak passwords or insecure password recovery methods.
- Bypassing Authorization Checks -: After getting authenticated, the attacker might alter cookies, tokens, or URL parameters to reach areas they shouldn't have access.
- Gaining Admin Rights -: Once the attacker escalates privileges, they can perform admin-level actions like changing user roles, deleting records, or accessing confidential data.

XII. Mitigation and Countermeasures

- Strong Passwords -: Enforce the use of strong, complex passwords and require multi-factor authentication (MFA).
- Role-based Access Control (RBAC) -: Make sure users are assigned appropriate roles and only have access to the resources they need. Regularly check and adjust user permissions.
- Session Management -: Use secure tokens for sessions and avoid exposing sensitive information like user credentials in URLs or cookies.
- Patch Vulnerabilities -: Regularly update software to fix known security issues and prevent attackers from exploiting them.
- Limit Access -: Follow the principle of least privilege (POLP), giving users only the access they need for their roles.