



## Table of contents.

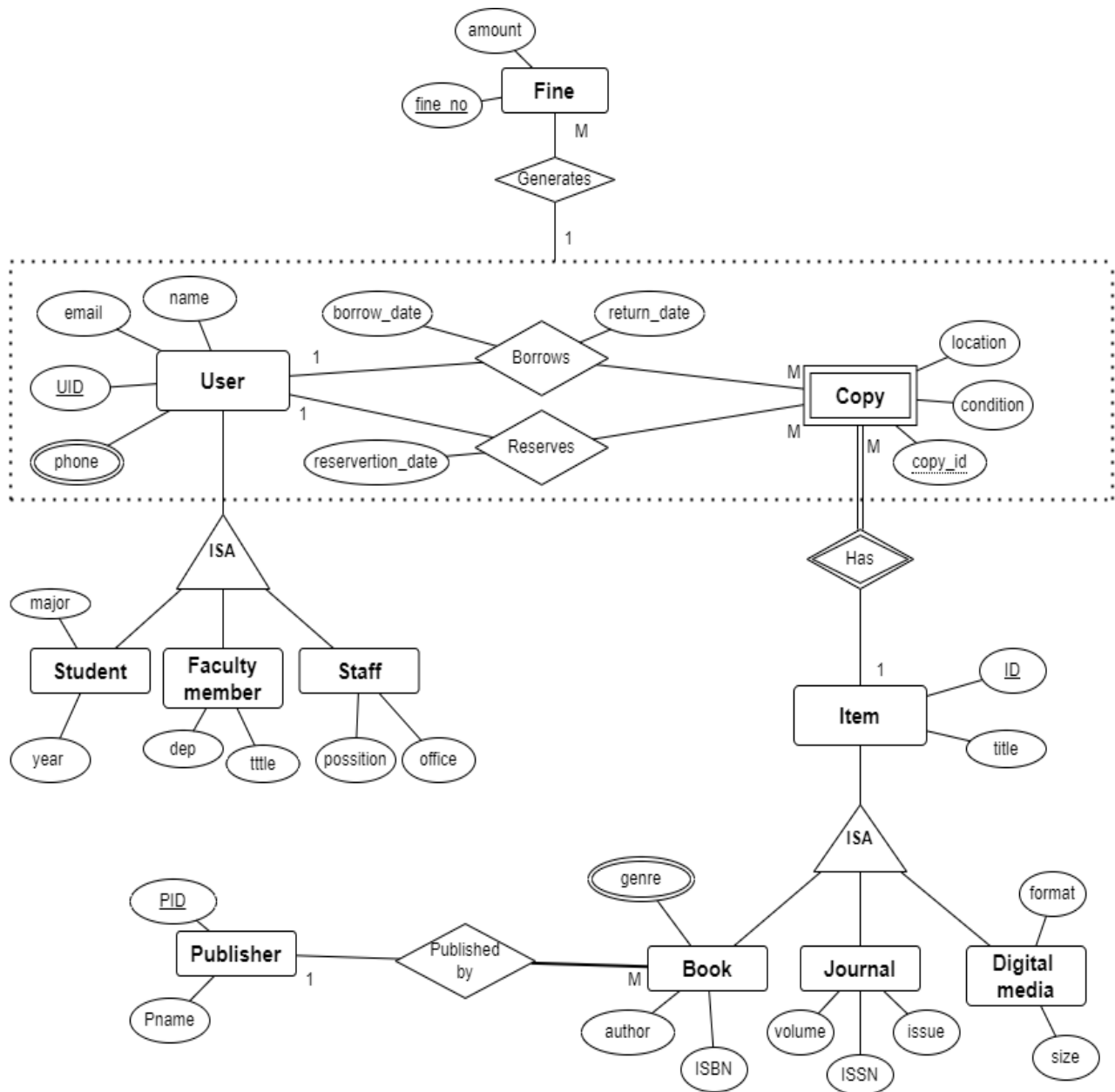
-----	03
<b>PART 1</b> -----	04
1. Assumptions. -----	04
2. Conceptual Model (EERD). -----	05
3. Logical Model (Unnormalized). -----	06
4. Normalization. -----	07
Logical Model (Normalized). -----	08
5. Implementing The Model Into The MySQL Server. -----	09
5.1. Creating Tables and Enforcing Constraints. -----	11
5.2. Sample Data. -----	14
6. Constrains Identified and Enforced. -----	14
7. Develop the required views, functions, procedures, triggers, and indexes as specified below. -----	17
7.1. 2 suitable triggers that can be applied to database. -----	17
7.2. 2 views for possible users. -----	19
7.3. 2 indexes that will optimize the given queries. -----	20
7.4. 2 Stored procedures. -----	21
<b>PART 2</b> -----	
8. Description and Analysis of 2 Database Vulnerabilities. -----	23
9. Mitigation Techniques and Countermeasures Suggestions. -----	24

# Part 1

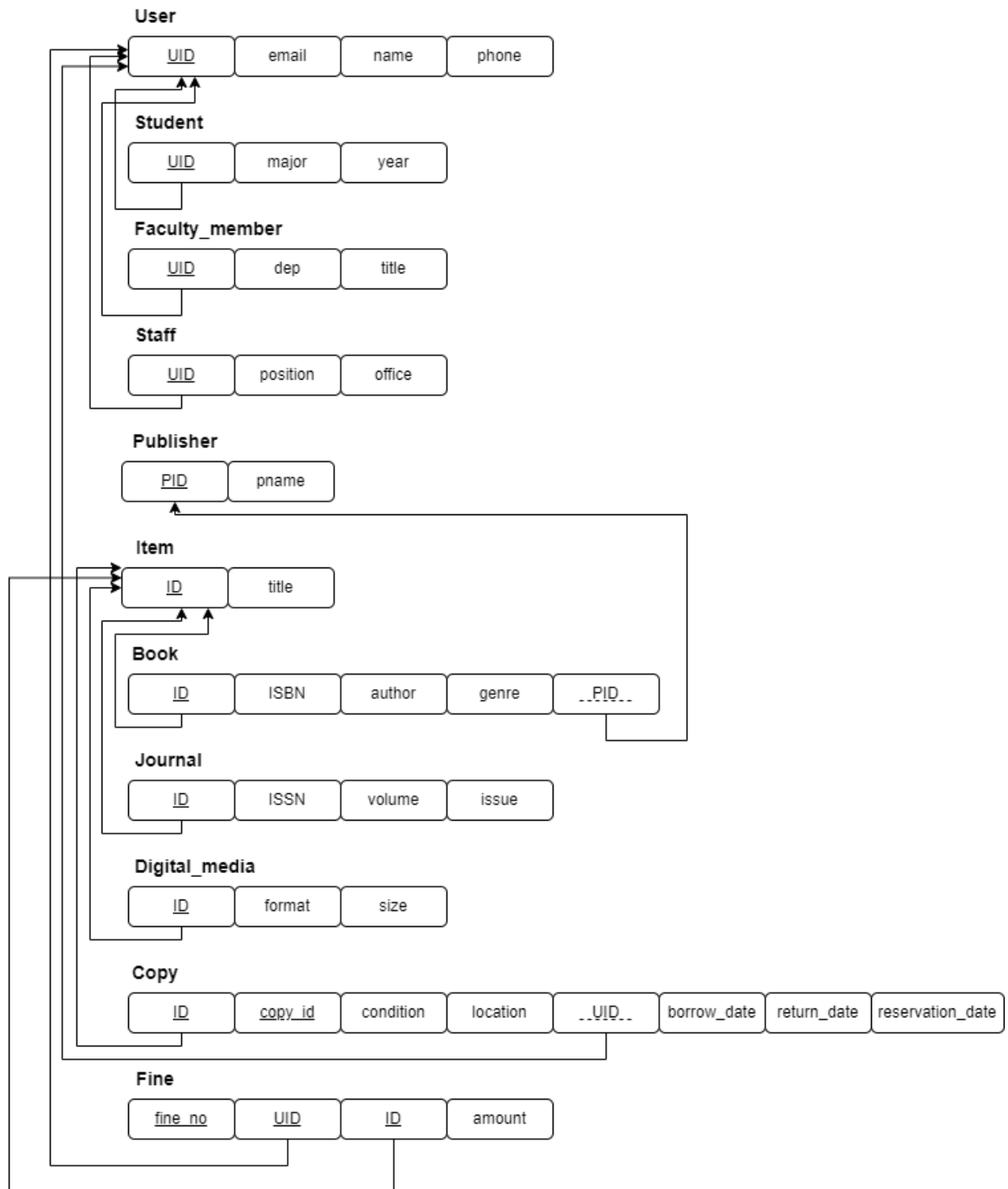
## 1. Assumptions

- Items can be books, journals, digital media.
- Each item type (Book, Journal, Digital Media) inherits a unique identifier from the Item superclass.
- An item can have multiple copies.
- The fines are charged on the user based on the borrowed date and the return date.
- A publisher can publish many books.
- A book must have a publisher.
- Users can only borrow or reserve a specific copy of an item, not the item.
- An item copy can be borrowed/reserved by only one user.
- Items have unique item identifier (ID).
- Users have unique identifier (UID).
- Fines have unique identifier (fine\_no).
- Publishers have unique identifier (PID).
- The borrowed date and the return date are recorded when a user borrows the item copy.
- The reservation date is recorded when a user makes reservation.
- Each user can have more than one phone number.
- A book can have multiple genres.

## 2. Conceptual Model



### 3. Logical model (Unnormalized)



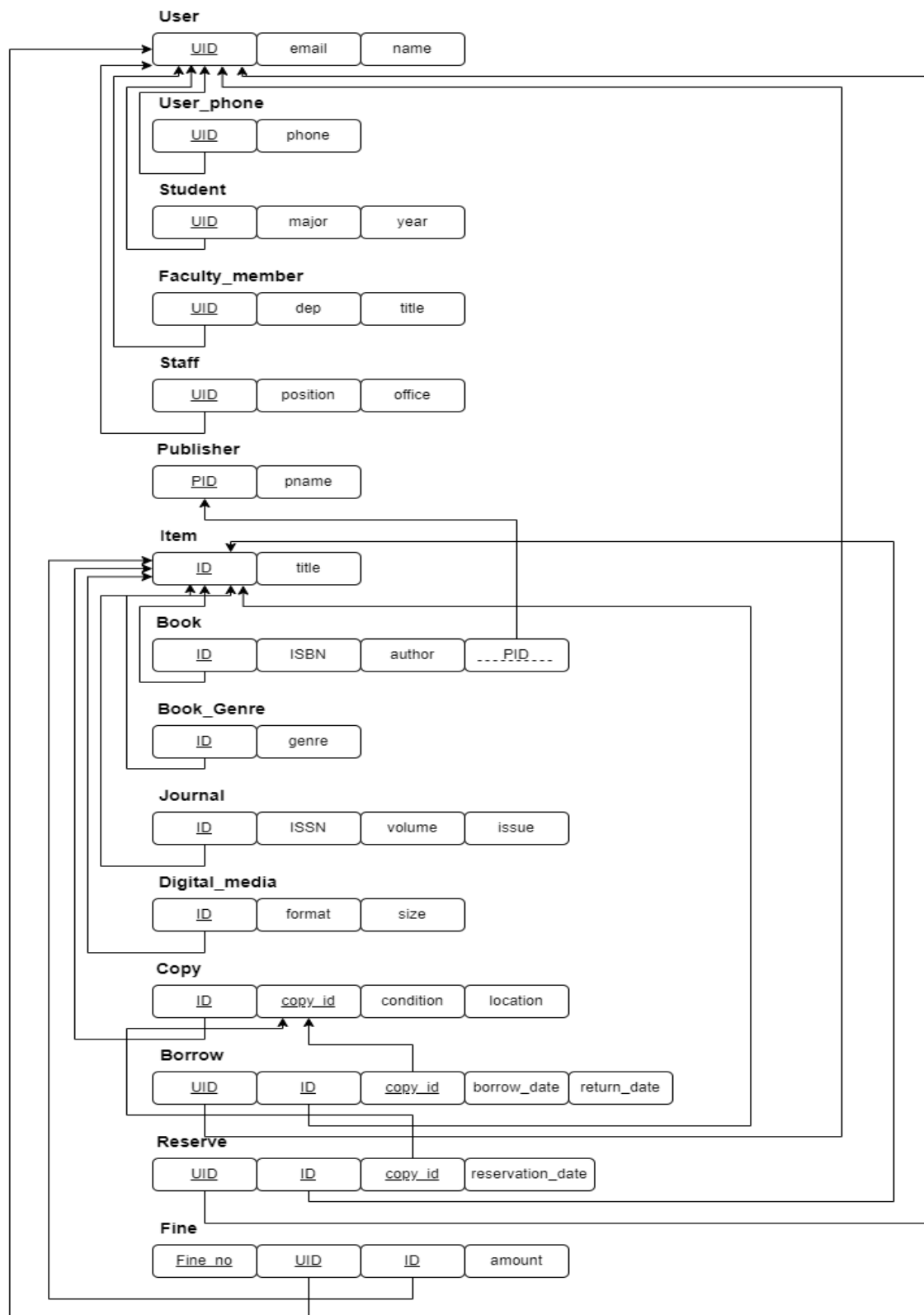
## 4. Normalization

Above implemented logical model is unnormalized and that is in **UNF**.

1. The User table and Book table contain non-atomic values. So, we broke the table. (phone, genre)
  - **Done to achieve 1NF**
2. There are no partial dependencies, so the schema is in **2NF**.
3. Created tables for borrowing and reserving
  - Because in the copy table there are transitive dependencies.
  - borrow\_date and return\_date are transitively dependent.
  - Also, the reservation\_date.
  - **Done to achieve 3NF.**

The model in the next page is normalized to 3NF as there are no non-atomic values, no partial key functional dependencies and no transitive functional dependencies.

## Logical Model (Normalized to 3NF)



## 5. Implement the logical model in the MS SQL server

```
-- Table for User
CREATE TABLE [User] (
    UID INT NOT NULL,
    email VARCHAR(255),
    name VARCHAR(255) NOT NULL,
    CONSTRAINT User_pk PRIMARY KEY (UID)
);

-- Table for User Phone
CREATE TABLE User_phone (
    UID INT NOT NULL,
    phone VARCHAR(15) NOT NULL,
    CONSTRAINT User_Phone_pk PRIMARY KEY (UID, phone),
    CONSTRAINT User_Phone_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Student
CREATE TABLE Student (
    UID INT NOT NULL,
    major VARCHAR(255) ,
    year INT NOT NULL,
    CONSTRAINT Student_pk PRIMARY KEY (UID),
    CONSTRAINT Student_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Faculty Member
CREATE TABLE Faculty_member (
    UID INT NOT NULL,
    dep VARCHAR(255) NOT NULL,
    title VARCHAR(255) ,
    CONSTRAINT Faculty_member_pk PRIMARY KEY (UID),
    CONSTRAINT Faculty_member_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Staff
CREATE TABLE Staff (
    UID INT NOT NULL,
    position VARCHAR(255) ,
    office VARCHAR(255) NOT NULL,
    CONSTRAINT Staff_pk PRIMARY KEY (UID),
    CONSTRAINT Staff_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Publisher
CREATE TABLE Publisher (
    PID INT NOT NULL,
    pname VARCHAR(255) NOT NULL,
    CONSTRAINT Publisher_pk PRIMARY KEY (PID)
);

-- Table for Item
CREATE TABLE Item (
    ID INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    CONSTRAINT Item_pk PRIMARY KEY (ID)
);

-- Table for Book
CREATE TABLE Book (
    ID INT NOT NULL,
    ISBN VARCHAR(20) NOT NULL,
    author VARCHAR(255) ,
    PID INT NOT NULL,
    CONSTRAINT Book_pk PRIMARY KEY (ID),
    CONSTRAINT Book_fk_item FOREIGN KEY (ID) REFERENCES Item(ID),
    CONSTRAINT Book_fk_publisher FOREIGN KEY (PID) REFERENCES Publisher(PID)
);
```



```

-- Table for Book Genre
CREATE TABLE Book_Genre (
    ID INT NOT NULL,
    genre VARCHAR(255) NOT NULL,
    CONSTRAINT Book_Genre_pk PRIMARY KEY (ID, genre),
    CONSTRAINT Book_Genre_fk FOREIGN KEY (ID) REFERENCES Book(ID)
);

-- Table for Journal
CREATE TABLE Journal (
    ID INT NOT NULL,
    ISSN VARCHAR(20) NOT NULL,
    volume INT NOT NULL,
    issue INT NOT NULL,
    CONSTRAINT Journal_pk PRIMARY KEY (ID),
    CONSTRAINT Journal_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Digital Media
CREATE TABLE Digital_media (
    ID INT NOT NULL,
    format VARCHAR(255) NOT NULL,
    size INT NOT NULL,
    CONSTRAINT Digital_media_pk PRIMARY KEY (ID),
    CONSTRAINT Digital_media_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Copy
CREATE TABLE Copy (
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    condition VARCHAR(255) NOT NULL,
    location VARCHAR(255) NOT NULL,
    CONSTRAINT Copy_pk PRIMARY KEY (ID, copy_id),
    CONSTRAINT Copy_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Borrow
CREATE TABLE Borrow (
    UID INT NOT NULL,
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    borrow_date DATE NOT NULL,
    return_date DATE,
    CONSTRAINT Borrow_pk PRIMARY KEY (UID, ID, copy_id),
    CONSTRAINT Borrow_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
    CONSTRAINT Borrow_fk_copy FOREIGN KEY (ID, copy_id) REFERENCES Copy(ID, copy_id)
);

-- Table for Reserve
CREATE TABLE Reserve (
    UID INT NOT NULL,
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    reservation_date DATE NOT NULL,
    CONSTRAINT Reserve_pk PRIMARY KEY (UID, ID, copy_id),
    CONSTRAINT Reserve_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
    CONSTRAINT Reserve_fk_copy FOREIGN KEY (ID, copy_id) REFERENCES Copy(ID, copy_id)
);

-- Table for Fines
CREATE TABLE Fine (
    Fine_no INT NOT NULL,
    UID INT NOT NULL,
    ID INT NOT NULL,
    amount DECIMAL(10, 2),
    CONSTRAINT Fine_pk PRIMARY KEY (Fine_no),
    CONSTRAINT Fine_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
    CONSTRAINT Fine_fk_item FOREIGN KEY (ID) REFERENCES Item(ID)
);

```

## 5.1 Creating tables and enforcing constraints.

```
-- Table for User
CREATE TABLE [User] (
    UID INT NOT NULL,
    email VARCHAR(255),
    name VARCHAR(255) NOT NULL,
    CONSTRAINT User_pk PRIMARY KEY (UID)
);

-- Table for User Phone
CREATE TABLE User_phone (
    UID INT NOT NULL,
    phone VARCHAR(15) NOT NULL,
    CONSTRAINT User_Phone_pk PRIMARY KEY (UID, phone),
    CONSTRAINT User_Phone_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Student
CREATE TABLE Student (
    UID INT NOT NULL,
    major VARCHAR(255) ,
    year INT NOT NULL,
    CONSTRAINT Student_pk PRIMARY KEY (UID),
    CONSTRAINT Student_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Faculty Member
CREATE TABLE Faculty_member (
    UID INT NOT NULL,
    dep VARCHAR(255) NOT NULL,
    title VARCHAR(255) ,
    CONSTRAINT Faculty_member_pk PRIMARY KEY (UID),
    CONSTRAINT Faculty_member_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Staff
CREATE TABLE Staff (
    UID INT NOT NULL,
    position VARCHAR(255) ,
    office VARCHAR(255) NOT NULL,
    CONSTRAINT Staff_pk PRIMARY KEY (UID),
    CONSTRAINT Staff_fk FOREIGN KEY (UID) REFERENCES [User](UID)
);

-- Table for Publisher
CREATE TABLE Publisher (
    PID INT NOT NULL,
    pname VARCHAR(255) NOT NULL,
    CONSTRAINT Publisher_pk PRIMARY KEY (PID)
);

-- Table for Item
CREATE TABLE Item (
    ID INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    CONSTRAINT Item_pk PRIMARY KEY (ID)
```

```

);

-- Table for Book
CREATE TABLE Book (
    ID INT NOT NULL,
    ISBN VARCHAR(20) NOT NULL,
    author VARCHAR(255) ,
    PID INT NOT NULL,
    CONSTRAINT Book_pk PRIMARY KEY (ID),
    CONSTRAINT Book_fk_item FOREIGN KEY (ID) REFERENCES Item(ID),
    CONSTRAINT Book_fk_publisher FOREIGN KEY (PID) REFERENCES Publisher(PID)
);

-- Table for Book Genre
CREATE TABLE Book_Genre (
    ID INT NOT NULL,
    genre VARCHAR(255) NOT NULL,
    CONSTRAINT Book_Genre_pk PRIMARY KEY (ID, genre),
    CONSTRAINT Book_Genre_fk FOREIGN KEY (ID) REFERENCES Book(ID)
);

-- Table for Journal
CREATE TABLE Journal (
    ID INT NOT NULL,
    ISSN VARCHAR(20) NOT NULL,
    volume INT NOT NULL,
    issue INT NOT NULL,
    CONSTRAINT Journal_pk PRIMARY KEY (ID),
    CONSTRAINT Journal_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Digital Media
CREATE TABLE Digital_media (
    ID INT NOT NULL,
    format VARCHAR(255) NOT NULL,
    size INT NOT NULL,
    CONSTRAINT Digital_media_pk PRIMARY KEY (ID),
    CONSTRAINT Digital_media_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Copy
CREATE TABLE Copy (
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    condition VARCHAR(255) NOT NULL,
    location VARCHAR(255) NOT NULL,
    CONSTRAINT Copy_pk PRIMARY KEY (ID, copy_id),
    CONSTRAINT Copy_fk FOREIGN KEY (ID) REFERENCES Item(ID)
);

-- Table for Borrow
CREATE TABLE Borrow (
    UID INT NOT NULL,
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    borrow_date DATE NOT NULL,
    return_date DATE,
    CONSTRAINT Borrow_pk PRIMARY KEY (UID, ID, copy_id),

```

```

        CONSTRAINT Borrow_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
        CONSTRAINT Borrow_fk_copy FOREIGN KEY (ID, copy_id) REFERENCES Copy(ID, copy_id)
    );

-- Table for Reserve
CREATE TABLE Reserve (
    UID INT NOT NULL,
    ID INT NOT NULL,
    copy_id INT NOT NULL,
    reservation_date DATE NOT NULL,
    CONSTRAINT Reserve_pk PRIMARY KEY (UID, ID, copy_id),
    CONSTRAINT Reserve_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
    CONSTRAINT Reserve_fk_copy FOREIGN KEY (ID, copy_id) REFERENCES Copy(ID, copy_id)
);

-- Table for Fines
CREATE TABLE Fine (
    Fine_no INT NOT NULL,
    UID INT NOT NULL,
    ID INT NOT NULL,
    amount DECIMAL(10, 2) ,
    CONSTRAINT Fine_pk PRIMARY KEY (Fine_no),
    CONSTRAINT Fine_fk_user FOREIGN KEY (UID) REFERENCES [User](UID),
    CONSTRAINT Fine_fk_item FOREIGN KEY (ID) REFERENCES Item(ID)
);

```

## 5.2 Sample data.

```
INSERT INTO [User] (UID, email, name) VALUES
(110, 'john.doe@example.com', 'John Doe'),
(111, 'jane.smith@example.com', 'Jane Smith'),
(112, 'alice.johnson@example.com', 'Alice Johnson'),
(113, 'bob.brown@example.com', 'Bob Brown');

INSERT INTO User_phone (UID, phone) VALUES
(110, '112-345-0980'),
(111, '235-568-9901'),
(112, '346-688-9042'),
(113, '404-727-2108');

INSERT INTO Student (UID, major, year) VALUES
(110, 'Computer Science', 3),
(111, 'Information Systems', 2);

INSERT INTO Faculty_member (UID, dep, title) VALUES
(112, 'Computer Science', 'Professor');

INSERT INTO Staff (UID, position, office) VALUES
(113, 'Librarian', 'Office 101');

INSERT INTO Publisher (PID, pname) VALUES
(200, 'TechBooks Publishing'),
(201, 'Educational Press'),
(202, 'KnowledgeSource Publishers'),
(203, 'ScienceWorks Press');

INSERT INTO Item (ID, title) VALUES
(300, 'Introduction to Algorithms'),
(301, 'Data Science Insights'),
(302, 'Modern Operating Systems'),
(303, 'Advanced Databases');

INSERT INTO Book (ID, ISBN, author, PID) VALUES
(300, '978-0262033848', 'Thomas H. Cormen', 200),
(301, '978-0136091813', 'Elmasri & Navathe', 201);

INSERT INTO Book_Genre (ID, genre) VALUES
(300, 'Computer Science'),
(303, 'Databases');

INSERT INTO Journal (ID, ISSN, volume, issue) VALUES
(301, '2345-6789', 1, 1);

INSERT INTO Digital_media (ID, format, size) VALUES
(302, 'MP4', 5120);

INSERT INTO Copy (ID, copy_id, condition, location) VALUES
(300, 1, 'Good', 'Library A'),
(300, 2, 'New', 'Library B'),
(301, 1, 'Excellent', 'Library A'),
(302, 1, 'Fair', 'Library C');

INSERT INTO Borrow (UID, ID, copy_id, borrow_date, return_date) VALUES
(110, 300, 1, '2024-10-01', '2024-10-15'),
(111, 301, 1, '2024-09-15', '2024-10-01'),
(112, 302, 1, '2024-09-20', '2024-10-05'),
(113, 300, 2, '2024-10-10', '2024-10-20');
```

```

INSERT INTO Reserve (UID, ID, copy_id, reservation_date) VALUES
(110, 300, 1, '2024-10-05'),
(112, 300, 1, '2024-09-20'),
(113, 301, 1, '2024-09-30');

INSERT INTO Fine (Fine_no, UID, ID, amount) VALUES
(1, 110, 300, 100.00),
(2, 111, 301, 50.00),
(3, 112, 302, 150.00),
(4, 113, 300, 200.00);

```

```

INSERT INTO [User] (UID, email, name) VALUES
(110, 'john.doe@example.com', 'John Doe'),
(111, 'jane.smith@example.com', 'Jane Smith'),
(112, 'alice.johnson@example.com', 'Alice Johnson'),
(113, 'bob.brown@example.com', 'Bob Brown');

```

```

INSERT INTO User_phone (UID, phone) VALUES
(110, '112-345-0980'),
(111, '235-568-9901'),
(112, '346-688-9042'),
(113, '404-727-2108');

```

```

INSERT INTO Student (UID, major, year) VALUES
(110, 'Computer Science', 3),
(111, 'Information Systems', 2);

```

```

INSERT INTO Faculty_member (UID, dep, title) VALUES
(112, 'Computer Science', 'Professor');

```

```

INSERT INTO Staff (UID, position, office) VALUES
(113, 'Librarian', 'Office 101');

```

```

INSERT INTO Publisher (PID, pname) VALUES
(200, 'TechBooks Publishing'),
(201, 'Educational Press'),
(202, 'KnowledgeSource Publishers'),
(203, 'ScienceWorks Press');

```

```

INSERT INTO Item (ID, title) VALUES
(300, 'Introduction to Algorithms'),
(301, 'Data Science Insights'),
(302, 'Modern Operating Systems'),
(303, 'Advanced Databases');

```

```

INSERT INTO Book (ID, ISBN, author, PID) VALUES
(300, '978-0262033848', 'Thomas H. Cormen', 200),
(301, '978-0136091813', 'Elmasri & Navathe', 201);

```

```

INSERT INTO Book_Genre (ID, genre) VALUES
(300, 'Computer Science'),
(301, 'Databases');

```

```

INSERT INTO Journal (ID, ISSN, volume, issue) VALUES
(301, '2345-6789', 1, 1);

```

```

INSERT INTO Digital_media (ID, format, size) VALUES
(302, 'MP4', 5120);

INSERT INTO Copy (ID, copy_id, condition, location) VALUES
(300, 1, 'Good', 'Library A'),
(300, 2, 'New', 'Library B'),
(301, 1, 'Excellent', 'Library A'),
(302, 1, 'Fair', 'Library C');

INSERT INTO Borrow (UID, ID, copy_id, borrow_date, return_date) VALUES
(110, 300, 1, '2024-10-01', '2024-10-15'),
(111, 301, 1, '2024-09-15', '2024-10-01'),
(112, 302, 1, '2024-09-20', '2024-10-05'),
(113, 300, 2, '2024-10-10', '2024-10-20');

INSERT INTO Reserve (UID, ID, copy_id, reservation_date) VALUES
(110, 300, 1, '2024-10-05'),
(112, 300, 1, '2024-09-20'),
(113, 301, 1, '2024-09-30');

INSERT INTO Fine (Fine_no, UID, ID, amount) VALUES
(1, 110, 300, 100.00),
(2, 111, 301, 50.00),
(3, 112, 302, 150.00),
(4, 113, 300, 200.00);

```

## 6. Constraints identified and enforced.

- **Primary Key Constraints:**

Ensures that each record on a table is uniquely identifiable.

- **Foreign Key Constraints:**

Maintains referential integrity between two related tables.

- **NOT NULL Constraints:**

Ensure that a column cannot contain a NULL value.

## 7. Develop the required views, functions, procedures, triggers, and indexes as specified below.

### 7.1. 2 suitable triggers that can be applied to database

#### Trigger 1: Prevent borrowing non-existing copy.

**Explanation:** It creates a trigger to make sure there is a copy of a book, before allowing any record to be inserted into the Borrow table. If the Copy with a given ID and copy\_id is not found, then the trigger raises an error and performs a rollback on the transaction. If the copy is exist, the insert is executed.

```
CREATE OR ALTER TRIGGER Prevent_Borrow_Invalid_Copy
ON Borrow
INSTEAD OF INSERT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Copy WHERE ID = (SELECT ID FROM inserted) AND copy_id = (SELECT copy_id FROM inserted))
    BEGIN
        RAISERROR('Cannot borrow a copy that does not exist.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Borrow(UUID, ID, copy_id, borrow_date, return_date)
        SELECT UUID, ID, copy_id, borrow_date, return_date FROM inserted;
    END
END;
```

```
CREATE OR ALTER TRIGGER Prevent_Borrow_Invalid_Copy
ON Borrow
INSTEAD OF INSERT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Copy WHERE ID = (SELECT ID FROM inserted) AND
copy_id = (SELECT copy_id FROM inserted))
    BEGIN
        RAISERROR('Cannot borrow a copy that does not exist.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Borrow(UUID, ID, copy_id, borrow_date, return_date)
        SELECT UUID, ID, copy_id, borrow_date, return_date FROM inserted;
    END
END;
```



## Trigger 2: Prevent duplicate reservation.

**Explanation:** This trigger would verify if the reservation of the same copy by the same user already exists. If it does, it makes errors and rolls back the transaction, preventing duplicate reservations. If it doesn't exist, the insert is executed.

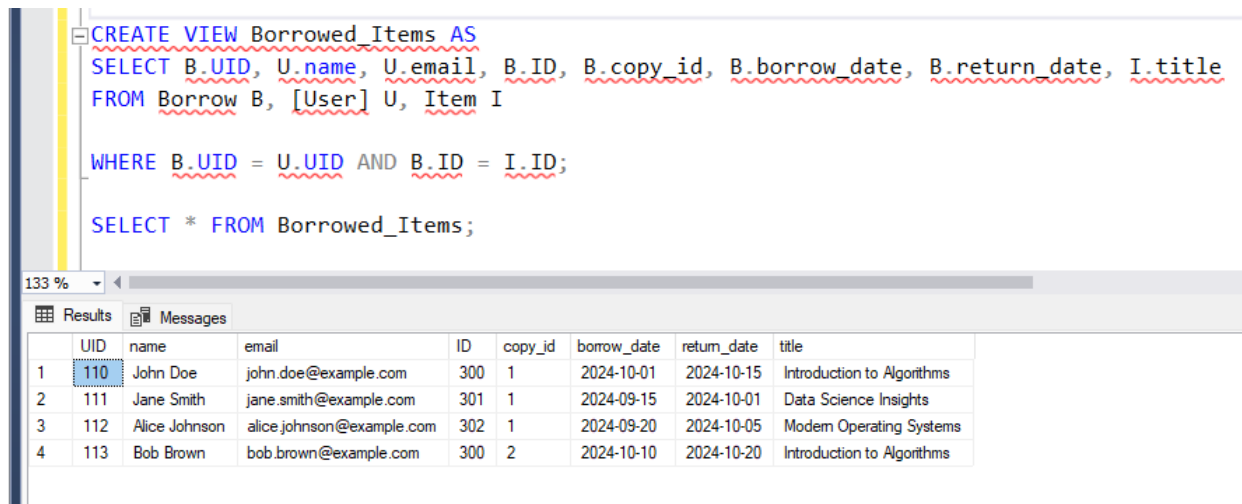
```
CREATE OR ALTER TRIGGER Prevent_Duplicate_Reservation
ON Reserve
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Reserve WHERE UID = (SELECT UID FROM inserted) AND ID = (SELECT ID FROM inserted) AND copy_id = (SELECT copy_id FROM inserted))
    BEGIN
        RAISERROR('Duplicate reservation is not allowed.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Reserve(UID, ID, copy_id, reservation_date)
        SELECT UID, ID, copy_id, reservation_date FROM inserted;
    END
END;
```

```
CREATE OR ALTER TRIGGER Prevent_Duplicate_Reservation
ON Reserve
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Reserve WHERE UID = (SELECT UID FROM inserted) AND ID =
(SELECT ID FROM inserted) AND copy_id = (SELECT copy_id FROM inserted))
    BEGIN
        RAISERROR('Duplicate reservation is not allowed.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Reserve(UID, ID, copy_id, reservation_date)
        SELECT UID, ID, copy_id, reservation_date FROM inserted;
    END
END;
```

## 7.2. Creating 2 views for possible users

### View 1: Borrowed items view

This view shows the UID, user name, email, item title, borrow date, and return date for borrowed items, making it useful for tracking borrowings.



```
CREATE VIEW Borrowed_Items AS
SELECT B.UID, U.name, U.email, B.ID, B.copy_id, B.borrow_date, B.return_date, I.title
FROM Borrow B, [User] U, Item I
WHERE B.UID = U.UID AND B.ID = I.ID;

SELECT * FROM Borrowed_Items;
```

	UID	name	email	ID	copy_id	borrow_date	return_date	title
1	110	John Doe	john.doe@example.com	300	1	2024-10-01	2024-10-15	Introduction to Algorithms
2	111	Jane Smith	jane.smith@example.com	301	1	2024-09-15	2024-10-01	Data Science Insights
3	112	Alice Johnson	alice.johnson@example.com	302	1	2024-09-20	2024-10-05	Modern Operating Systems
4	113	Bob Brown	bob.brown@example.com	300	2	2024-10-10	2024-10-20	Introduction to Algorithms

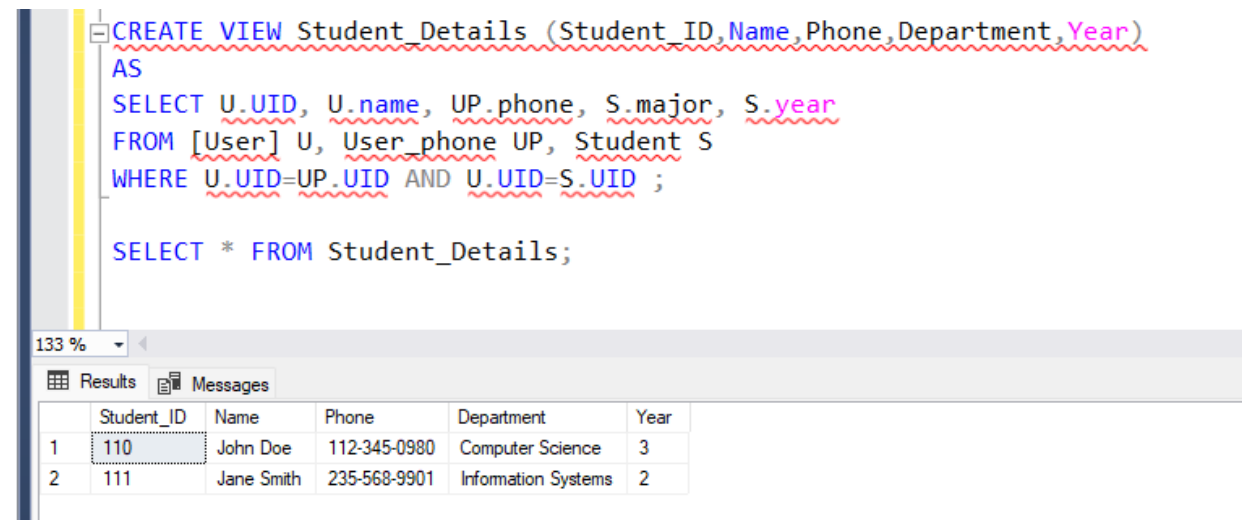
```
CREATE VIEW Borrowed_Items AS

SELECT B.UID, U.name, U.email, B.ID, B.copy_id, B.borrow_date, B.return_date, I.title

FROM Borrow B, [User] U, Item I
WHERE B.UID = U.UID AND B.ID = I.ID;
```

### View 2: Student details view

The view can be used to retrieve Registered Users details (details of student) .



```
CREATE VIEW Student_Details (Student_ID, Name, Phone, Department, Year)
AS
SELECT U.UID, U.name, UP.phone, S.major, S.year
FROM [User] U, User_phone UP, Student S
WHERE U.UID=UP.UID AND U.UID=S.UID ;

SELECT * FROM Student_Details;
```

	Student_ID	Name	Phone	Department	Year
1	110	John Doe	112-345-0980	Computer Science	3
2	111	Jane Smith	235-568-9901	Information Systems	2

```
CREATE VIEW Student_Details (Student_ID, Name, Phone, Department, Year)
AS
SELECT U.UID, U.name, UP.phone, S.major, S.year
FROM [User] U, User_phone UP, Student S
WHERE U.UID=UP.UID AND U.UID=S.UID ;
```

### **7.3. Identify 2 indexes that will optimize the given queries and implement them.**

- Retrieve the details of all the items borrowed by a given member within a given period.
- Retrieve the outstanding fines for a given member.

#### **1. Index for UID and Borrow Date**

This index optimizes queries that search for items borrowed by a specific user (UID) within a specific date range (borrow\_date).

```
CREATE INDEX IDX_Borrow_UID_BorrowDate ON Borrow(UID, borrow_date);
```

```
CREATE INDEX IDX_Borrow_UID_BorrowDate ON Borrow(UID, borrow_date);
```

#### **2. Index on fine table.**

This index speeds up queries that retrieve fines for a specific user.

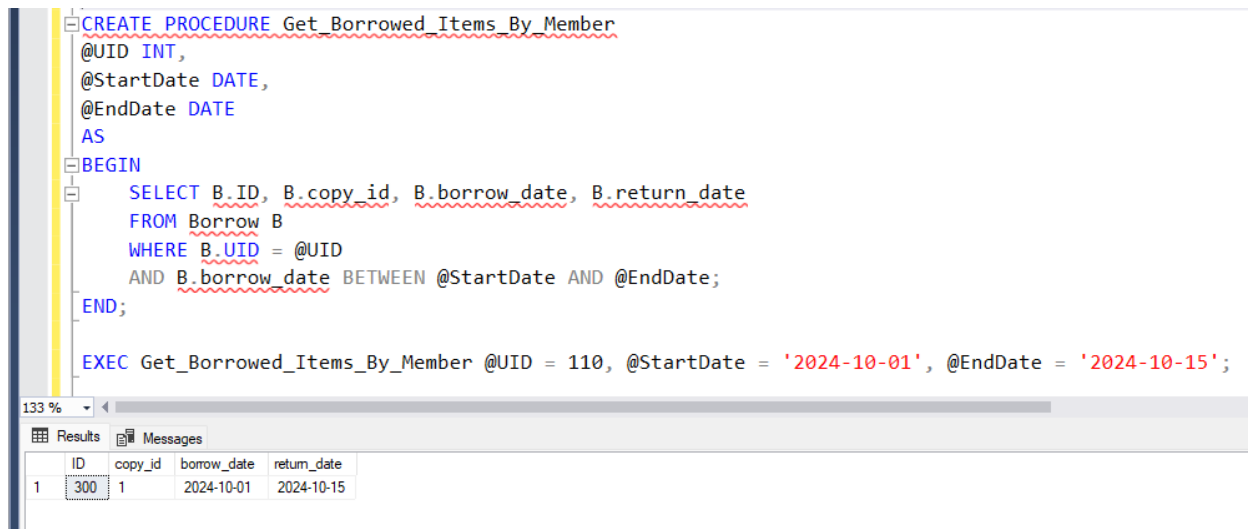
```
CREATE INDEX IDX_Fine_UID ON Fine(UID);
```

```
CREATE INDEX IDX_Fine_UID ON Fine(UID);
```

## 7.4. Stored procedures to carry out the below DML functions

### Procedure 1:

Retrieve the details of all the items borrowed by a given member within a given period.



```
CREATE PROCEDURE Get_Borrowed_Items_By_Member
@UID INT,
@StartDate DATE,
@EndDate DATE
AS
BEGIN
    SELECT B.ID, B.copy_id, B.borrow_date, B.return_date
    FROM Borrow B
    WHERE B.UID = @UID
    AND B.borrow_date BETWEEN @StartDate AND @EndDate;
END;

EXEC Get_Borrowed_Items_By_Member @UID = 110, @StartDate = '2024-10-01', @EndDate = '2024-10-15';
```

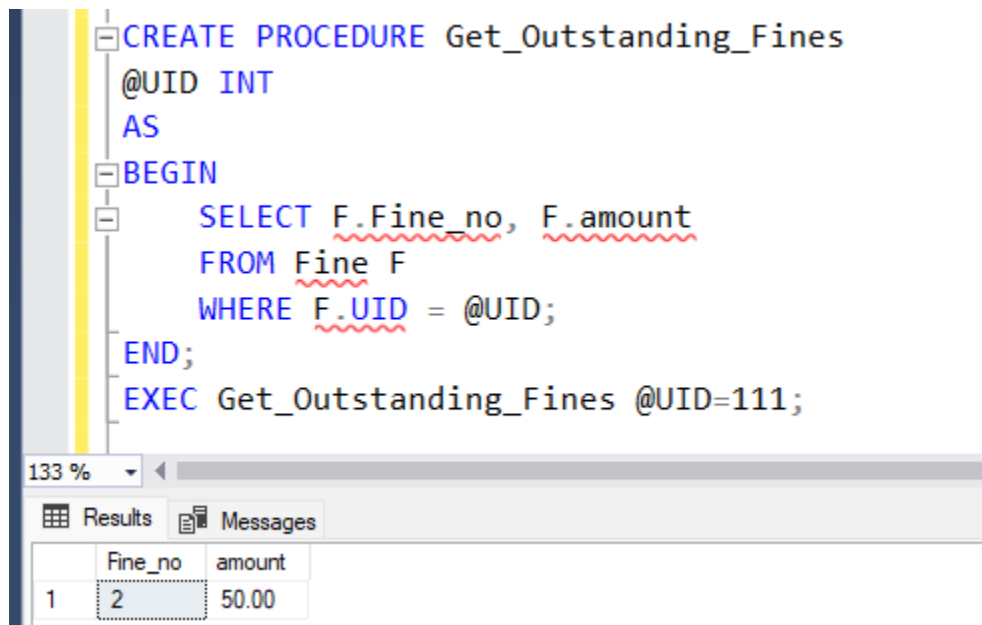
The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the SQL script for creating and executing the stored procedure. The bottom pane shows the results of the execution, which is a table with four columns: ID, copy\_id, borrow\_date, and return\_date. The table contains one row of data.

ID	copy_id	borrow_date	return_date
1	300	2024-10-01	2024-10-15

```
CREATE PROCEDURE Get_Borrowed_Items_By_Member
@UID INT,
@StartDate DATE,
@EndDate DATE
AS
BEGIN
    SELECT B.ID, B.copy_id, B.borrow_date, B.return_date
    FROM Borrow B
    WHERE B.UID = @UID
    AND B.borrow_date BETWEEN @StartDate AND @EndDate;
END;
```

## Procedure 2:

Retrieve the outstanding fines for a given member.



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays the following T-SQL code:

```
CREATE PROCEDURE Get_Outstanding_Fines
@UID INT
AS
BEGIN
    SELECT F.Fine_no, F.amount
    FROM Fine F
    WHERE F.UID = @UID;
END;
EXEC Get_Outstanding_Fines @UID=111;
```

The bottom pane shows the 'Results' tab with a zoom level of 133%. It displays a single row of data from the 'Fine' table:

	Fine_no	amount
1	2	50.00

```
CREATE PROCEDURE Get_Outstanding_Fines
@UID INT
AS
BEGIN
    SELECT F.Fine_no, F.amount
    FROM Fine F
    WHERE F.UID = @UID;
END;
```

## 8. Description and Analysis of 2 Database Vulnerabilities

### 1. SQL Injection Vulnerability

It is one of the most frequent attacks-a hacker injects malicious SQL code into a query, which compromises the security of the database. This happens when the input provided by a user is not well validated or sanitized. By tampering with such inputs, an attacker may eventually manipulate queries into extracting, modifying, or deleting sensitive data.

**Example:** A login form without input validation may allow an attacker to provide a SQL statement, such as ' OR 1=1 --, thereby successfully bypassing authentication and gaining unauthorized access.

**Impact:** Attackers are able to conduct security controls bypass, read sensitive information, or modify information, probably with full control over the database.

### 2. Weak Authentication Mechanisms

Poor authentication mechanisms or bad access control can let unauthorized users gain permission to access the database. A good example is using default credentials, which leads to not enforcing strong passwords. In turn, this exposes the database to brute-force attack.

- **Example:** An attacker may compromise a password if the database administrator does not enforce a strong password or two-factor authentication.
- **Impact:** This may result in unauthorized access to the database, whereby hackers can view or modify data inside the database. This may also lead to loss of confidentiality, integrity, and availability.

## 9. Mitigation Techniques and Countermeasures Suggestions

### 1. SQL Injection Prevention

- **Input Validation and Sanitization:**

All users' inputs shall be validated to ensure type, length, format, and range. Prepared statements or parameterized queries shall be used where user input should never be added directly to SQL.

- **ORM Frameworks:**

Object-Relational Mapping frameworks, such as Hibernate and Entity Framework, are able to abstract direct SQL queries and reduce the possibility of injection vulnerabilities.

- **Web Application Firewalls:**

Use WAFs, which can detect and block malicious SQL injection attempts before they reach the application.

- **Regular Security Audits:**

Perform regular SQL Injection security audits and vulnerability checks of your applications and databases, so that you can detect and remedy SQL Injection vulnerabilities.

- **Error Handling:**

Take appropriate measures to prevent displaying the sensitive data in error messages. Some helpful generic error messages should be displayed to users to prevent the hacker from inferring the structure of your database.

### 2. Hardening Authentication

- **Enforce Strong Password Policies:**

Ensure that the passwords are strong, comprising a mix of letters, numbers, and special characters, and these should be regularly changed.

- **MFA:**

Using multi-factor authentication-for example-will have users verify an additional factor, such as through SMS, email, or with an authenticator app, to reduce the possibility of unauthorized access.

- **Limit Login Attempts:**

Implement account lockout mechanisms after a certain number of attempts have failed to prevent brute-force attacks.

- **Session Timeout:**

Implement session timeouts that log the user out after some period of inactivity. This will reduce the possibility of an unattended session leading to unauthorized access.

- **Password Hashing:**

Passwords need to be in storage by hashing using strong hashing algorithms, such as bcrypt or Argon2, so that in case of any database breach, the passwords remain unreadable to attackers, and it would be computationally infeasible to obtain the original password.

Among the techniques discussed above, there will be efforts toward reducing the identified risks of each database vulnerability, thereby providing a better security posture to the entire system.