Birzeit University

جامعة بيرزيت

BIRZEIT UNIVERSITY

Engineering and **Information** Technology Faculty

Computer Science Department

Data Base Administration

Project Report

Lotfi Qasim:1202064

Dr. Abdallah Alnatsha

Contents

Abstract:

In this project we have applied the methods which we have learned through-out the course in such a way that shows our understanding of the outcomes of the course Starting from Understanding Software requirements of needed Data-Base, to creating necessary Stored procedures, and triggers, adding users to database, creating roles and Authorization matrix, Appling some basic encryption on the some what sensitive data in the database and much more.

Data Base anticipation and assumptions

-- For this database My assumptions were

-- Each staff member can only work in one branch.

-- Each branch has only one Branch Manager.

-- Each supervisor oversees multiple staff members within a branch.

-- Each book copy belongs to a specific branch.

-- Each member can borrow multiple books.

-- Each book can be borrowed by multiple members.

-- Each member can provide feedback for multiple books.

-- Each book can receive feedback from multiple members.

Creating necessary Tables

```
Create table Branch (
	BranchID INT IDENTITY(1,1) PRIMARY KEY,
	Location VARCHAR (50)
);

Create table Staff (
	StaffId INT IDENTITY(1,1) PRIMARY KEY,
	Name VARCHAR (50),
	Position VARCHAR (50),
	BranchID INT,
	CONSTRAINT fk_branch
		FOREIGN KEY (BranchID) REFERENCES Branch (BranchID)
);

Create table BranchManager(
	ManagerID INT IDENTITY(1,1) PRIMARY KEY,
	StaffID INT,
	CONSTRAINT fk_staff
		FOREIGN KEY (StaffID) REFERENCES Staff (StaffID)
);

Create table Supervisor (
	SupervisorID int IDENTITY(1,1) PRIMARY KEY,
	StaffID INT,
	BranchID INT,
	CONSTRAINT fk_supervisor_staff
		FOREIGN KEY (StaffID) REFERENCES Staff (StaffID),
	CONSTRAINT fk_supervisor_branch
		FOREIGN KEY (BranchID) REFERENCES Branch (BranchID)
);

Create TABLE Book (
    BookID INT IDENTITY(1,1) PRIMARY KEY,
```

```
    ISBN VARCHAR(50),
    BookNumber VARCHAR(50),
    BranchID INT,
    CONSTRAINT fk_book_branch FOREIGN KEY (BranchID) REFERENCES Branch (BranchID)
);
ALTER TABLE Book
ADD Copies INT DEFAULT 1;
Select * from BOOK;
Alter TABLE BOOK
ADD Name VARCHAR (50);

Create table Member (
       MemberID INT IDENTITY(1,1) PRIMARY KEY,
       Name VARCHAR(50),
       Address VARCHAR (100),
       ContactNumber VARBINARY(256), -- most numbers are assumed to be at most 20 :)
       Email VARCHAR (50)
);

CREATE TABLE Rental (
    RentalID INT IDENTITY(1,1) PRIMARY KEY,
    MemberID INT,
    BookID INT,
    RentalDate DATE,
    DueDate DATE,
    ReturnDate DATE,
    LateFees DECIMAL(10, 2), --10 digits 2 of them in decimal decimal column to store
the late fees charged for returning the book after the due date.
    CONSTRAINT fk_rental_member
        FOREIGN KEY (MemberID) REFERENCES Member (MemberID),
    CONSTRAINT fk_rental_book
        FOREIGN KEY (BookID) REFERENCES Book (BookID)
);


Create table Feedback (
    FeedbackID INT IDENTITY(1,1) PRIMARY KEY,
    MemberID INT,
    BookID INT,
    Rating INT, --  rating given by the member for the book.
    Comment VARCHAR(255), -- store any optional comments provided by the member for the
book
    CONSTRAINT fk_feedback_member
        FOREIGN KEY (MemberID) REFERENCES Member (MemberID),
    CONSTRAINT fk_feedback_book
        FOREIGN KEY (BookID) REFERENCES Book (BookID)
);


--Implement a historical model to track all the modification happens in Albayan
database. Each group member is required to produce two DML triggers.

--Logs for the track table
CREATE TABLE StaffHistory (
    HistoryID INT IDENTITY(1, 1) PRIMARY KEY,
    StaffID INT,
    Name VARCHAR(50),
    Position VARCHAR(50),
```

```
    BranchID INT,
    Operation VARCHAR(10),
    ModifiedDate DATETIME
);

CREATE TABLE BookHistory (
    HistoryID INT IDENTITY(1, 1) PRIMARY KEY,
    TableName VARCHAR(100),
    Operation VARCHAR(10),
    ModifiedDate DATETIME,
    BookID INT,
    ISBN VARCHAR(50),
    BookNumber VARCHAR(50),
    BranchID INT
);
```

In which the above database has resulted in the following Data base Diagram:

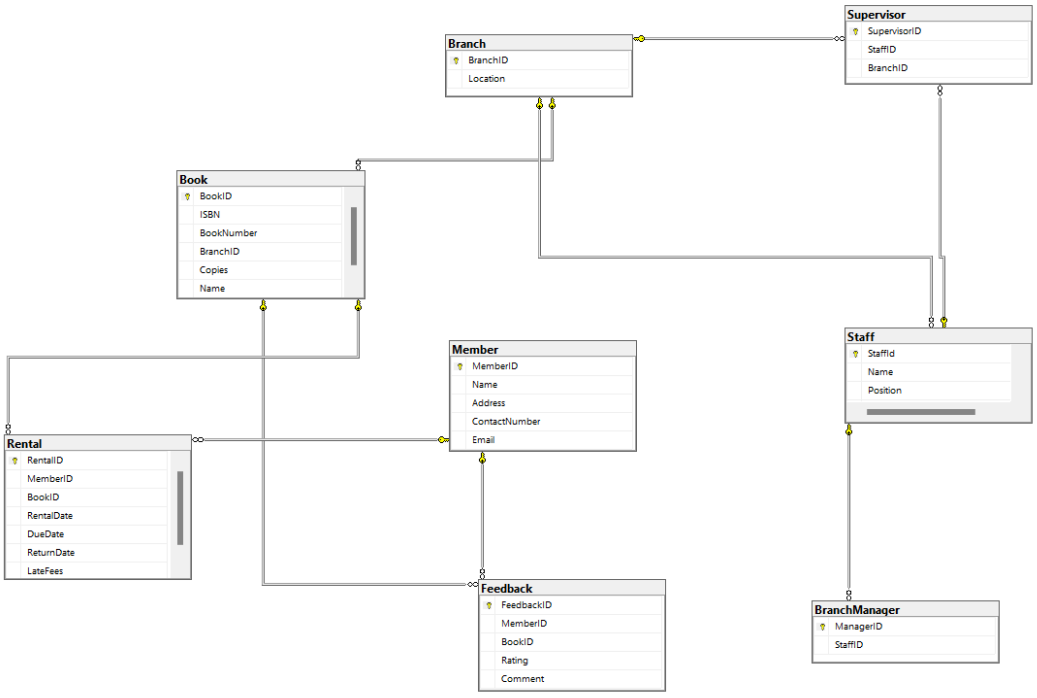

Figure 0-1Data base entity diagram

Creating needed Stored Procedures:

Where for stored procedure we have added what we thought was the most important and needed stored procedures which acted as the most important constrains to the data base:

```
- Trigger 1: Capture INSERT operations on the Staff table
GO
CREATE TRIGGER Staff_InsertTrigger
ON Staff
AFTER INSERT
AS
BEGIN
    INSERT INTO StaffHistory (StaffID, Name, Position, BranchID, Operation,
ModifiedDate)
    SELECT StaffID, Name, Position, BranchID, 'INSERT', GETDATE()
    FROM inserted;
END;

GO
-- Trigger 2: Capture UPDATE operations on the Staff table
CREATE TRIGGER Staff_UpdateTrigger
ON Staff
AFTER UPDATE
AS
BEGIN
    INSERT INTO StaffHistory (StaffID, Name, Position, BranchID, Operation,
ModifiedDate)
    SELECT StaffID, Name, Position, BranchID, 'UPDATE', GETDATE()
    FROM deleted;
END;
Select * from dbo.StaffHistory;
-- Trigger 3 :for INSERT operation on the book table
CREATE TRIGGER Book_InsertTrigger
ON Book
AFTER INSERT
AS
BEGIN
    INSERT INTO BookHistory (TableName, Operation, ModifiedDate, BookID, ISBN,
BookNumber, BranchID)
    SELECT 'Book', 'INSERT', GETDATE(), BookID, ISBN, BookNumber, BranchID
    FROM inserted;
END;

-- Trigger 4: for UPDATE operation on the book table
CREATE TRIGGER Book_InsertTrigger
ON Book
AFTER UPDATE
AS
BEGIN
    INSERT INTO BookHistory (TableName, Operation, ModifiedDate, BookID, ISBN,
BookNumber, BranchID)
    SELECT 'Book', 'UPDATE', GETDATE(), Book.BookID, Book.ISBN, Book.BookNumber,
Book.BranchID
    FROM inserted
    INNER JOIN Book ON inserted.BookID = Book.BookID;
```

```sql
END;

--  stored procedure for adding feedback with a check for existing feedback
CREATE PROCEDURE AddFeedback
    @MemberID INT,
    @BookID INT,
    @Rating INT,
    @Comment VARCHAR(255)
AS
BEGIN
    -- Check if feedback already exists for the same member and book
    IF EXISTS (
        SELECT 1
        FROM Feedback
        WHERE MemberID = @MemberID AND BookID = @BookID
    )
    BEGIN
        RAISERROR ('Feedback already exists for the same member and book.', 16, 1);
        RETURN;
    END

    -- Check if the rating is more than 5
    IF @Rating > 5
    BEGIN
        RAISERROR ('Invalid rating. Rating cannot be more than 5.', 16, 1);
        RETURN;
    END

    -- Insert the feedback into the Feedback table
    INSERT INTO Feedback (MemberID, BookID, Rating, Comment)
    VALUES (@MemberID, @BookID, @Rating, @Comment);
END;



--Stored procedure that allows Members to rent a book based on specified requirements
--
--
--
GO
CREATE PROCEDURE BorrowBook
    @MemberID INT,
    @BookID INT,
    @RentalDate DATE
AS
BEGIN
    -- Check if the member has reached the maximum limit of book rentals
    IF (SELECT COUNT(*) FROM Rental WHERE MemberID = @MemberID AND ReturnDate IS NULL)
>= 5
    BEGIN
        RAISERROR('You have reached the maximum limit of book rentals.', 16, 1)
        RETURN
    END

    -- Check if the book is available for rental and has copies available
    IF EXISTS (
        SELECT *
        FROM Book
```

```
        WHERE BookID = @BookID
            AND BranchID IN (1, 2, 3)
            AND Copies > 0
            AND BookID NOT IN (SELECT BookID FROM Rental WHERE ReturnDate IS NULL)
    )
    BEGIN
        -- Decrement the number of copies for the borrowed book
        UPDATE Book
        SET Copies = Copies - 1
        WHERE BookID = @BookID

        -- Calculate the due date (14 days from the rental date)
        DECLARE @DueDate DATE
        SET @DueDate = DATEADD(DAY, 14, @RentalDate)

        -- Insert the rental record with initial LateFees of 0.0
        INSERT INTO Rental (MemberID, BookID, RentalDate, DueDate, LateFees)
        VALUES (@MemberID, @BookID, @RentalDate, @DueDate, 0.0)

        PRINT 'Book borrowed successfully.'
    END
    ELSE
    BEGIN
        RAISERROR('The requested book is not available for rental or has no copies
available.', 16, 1)
        RETURN
    END

    -- Check for any overdue books and calculate late fees
    DECLARE @OverdueBooksCount INT, @LateFees DECIMAL(10, 2), @LateFeePerDay
DECIMAL(10, 2)
    SELECT @OverdueBooksCount = COUNT(*)
    FROM Rental
    WHERE MemberID = @MemberID AND ReturnDate IS NULL AND DueDate < GETDATE()

    SET @LateFees = 0.0
    SET @LateFeePerDay = 2.0 -- Assuming a late fee of $2.00 per day for all books

    IF (@OverdueBooksCount > 0)
    BEGIN
        -- Calculate the late fees based on the number of late days and the fixed late
fee per day
        SELECT @LateFees = @LateFeePerDay * DATEDIFF(DAY, DueDate, GETDATE())
        FROM Rental
        WHERE MemberID = @MemberID AND ReturnDate IS NULL AND DueDate < GETDATE()

        -- Update the LateFees column for overdue books
        UPDATE Rental
        SET LateFees = @LateFees
        WHERE MemberID = @MemberID AND ReturnDate IS NULL AND DueDate < GETDATE()

        PRINT 'You have ' + CAST(@OverdueBooksCount AS VARCHAR) + ' overdue book(s).'
        PRINT 'Late fees: $' + CAST(@LateFees AS VARCHAR)
    END
END
```

```
GO
--Rent a book
DECLARE @MemberID INT, @BookID INT, @RentalDate DATE;

-- Set the parameter values
SET @MemberID = 1;
SET @BookID = 1;
SET @RentalDate = GETDATE();

-- Call the stored procedure
EXECUTE BorrowBook @MemberID, @BookID, @RentalDate;

Select * From Rental;


--Return a book
GO
Create PROCEDURE ReturnBook
    @RentalID INT
AS
BEGIN
    DECLARE @LateFees DECIMAL(10, 2)
    DECLARE @LateDays INT
    DECLARE @BookID INT

    -- Update the ReturnDate to the current date only if it is null
    UPDATE Rental
    SET ReturnDate = GETDATE()
    WHERE RentalID = @RentalID AND ReturnDate IS NULL

    -- Check if the ReturnDate was updated
    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'The book has already been returned.'
        RETURN
    END

    -- Retrieve the BookID associated with the rental
    SELECT @BookID = BookID
    FROM Rental
    WHERE RentalID = @RentalID

    -- Increment the number of copies for the returned book
    UPDATE Book
    SET Copies = Copies + 1
    WHERE BookID = @BookID

    -- Calculate the late fees
    SELECT @LateDays = DATEDIFF(DAY, DueDate, GETDATE())
    FROM Rental
    WHERE RentalID = @RentalID

    SET @LateFees = @LateDays * 2.0 -- Assuming a late fee of $2.00 per day

    -- Update the LateFees column
    IF (@LateFees > 0)
    BEGIN
```

```
        UPDATE Rental
        SET LateFees = @LateFees
        WHERE RentalID = @RentalID
        PRINT 'Late fees: $' + CAST(@LateFees AS VARCHAR)
    END
    ELSE
    BEGIN
        PRINT 'No Late fees'
    END
END
```

Encryption:

Encryption process and explaination:

--I decided to use column encryption in the Member table for the ContactNumber column.

-- I chose the common approach of DMK -> certificate -> symmetric key -> column data.

-- Reasons for using this encryption:

1. Ease of implementation: The common approach is easier to implement compared to asymmetric key encryption as it uses a single encryption key for both encryption and decryption.

2. Performance: Symmetric encryption algorithms are typically faster than asymmetric keys encryption. As I have decided to encrypt the ContactNumber column, which will be accessed frequently in the database, we need a fast decryption method.

```
USE AlBayan_Library;

-- Lotfi,

-- Step 1: Create Data Master Key (DMK)
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1111';

-- Step 2: Create a certificate protected with DMK
CREATE CERTIFICATE contactNumberCertificate WITH SUBJECT = 'Contact Number';

-- Step 3: Create a symmetric key protected by the certificate
-- Note: The symmetric key is protected by the certificate 'contactNumberCertificate'
--       The algorithm AES_256 is the default
CREATE SYMMETRIC KEY contactNumberSymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE contactNumberCertificate;

-- Step 4: Open the symmetric key before encrypting or decrypting
OPEN SYMMETRIC KEY contactNumberSymmetricKey
    DECRYPTION BY CERTIFICATE contactNumberCertificate;

-- Update already entered data into Member
UPDATE Member
SET ContactNumber = ENCRYPTBYKEY(KEY_GUID('contactNumberSymmetricKey'),
CONVERT(VARBINARY(256), ContactNumber));

-- Close the symmetric key
CLOSE SYMMETRIC KEY contactNumberSymmetricKey;

-- For decryption
OPEN SYMMETRIC KEY contactNumberSymmetricKey
    DECRYPTION BY CERTIFICATE contactNumberCertificate;

SELECT *,
       CAST(DECRYPTBYKEY(ContactNumber) AS VARCHAR(20)) AS ContactNumber
FROM Member;

CLOSE SYMMETRIC KEY contactNumberSymmetricKey;

-- Instead of trigger for inserting data into Member to encrypt it
```

```
GO
CREATE TRIGGER insertEncryptedMember
ON Member
INSTEAD OF INSERT
AS
BEGIN
    OPEN SYMMETRIC KEY contactNumberSymmetricKey
        DECRYPTION BY CERTIFICATE contactNumberCertificate;

    INSERT INTO Member (Name, Address, ContactNumber, Email)
    SELECT Name, Address, ENCRYPTBYKEY(KEY_GUID('contactNumberSymmetricKey'),
CONVERT(VARBINARY(256), ContactNumber)), Email
    FROM inserted;

    CLOSE SYMMETRIC KEY contactNumberSymmetricKey;
END;


INSERT INTO Member (Name, Address, ContactNumber, Email)
VALUES ('ahmads', 'Birzeit', CONVERT(VARBINARY(256), '0569999889'), 'ahmad@gmail.com');

SELECT * FROM Member;

--DROP Trigger insertEncryptedMember;
--Seeing values of contact number
OPEN SYMMETRIC KEY contactNumberSymmetricKey
    DECRYPTION BY CERTIFICATE contactNumberCertificate;
--Note running with closed key returns value null

SELECT *,
       CAST(DECRYPTBYKEY(ContactNumber) AS VARCHAR(20)) AS DecryptedContactNumber
FROM Member;

CLOSE SYMMETRIC KEY contactNumberSymmetricKey;

-- Populate Member table
INSERT INTO Member (Name, Address, ContactNumber, Email)
VALUES ('John Doe', '123 Main Street', CONVERT(VARBINARY(256),'555-1234'),
'johndoe@example.com'),
       ('Jane Smith', '456 Elm Avenue', CONVERT(VARBINARY(256),'555-5678'),
'janesmith@example.com'),
       ('Michael Johnson', '789 Oak Drive',CONVERT(VARBINARY(256), '555-9012'),
'michaeljohnson@example.com');
```

Encryption:

Encryption process and explaination:

--I decided to use columns encryption in the feedback table for the Rating column and comment column.

-- I chose the common approach of DMK -> certificate -> symmetric key -> column data.

-- Reasons for using this encryption:

This can enhance trust and confidence among customers, knowing that their feedback is handled with care and their personal information remains secure.

```
USE AlBayan_Library;

CREATE CERTIFICATE feedbackCertificate WITH SUBJECT = 'Feedback Encryption';

CREATE SYMMETRIC KEY feedbackSymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE feedbackCertificate;

OPEN SYMMETRIC KEY feedbackSymmetricKey
    DECRYPTION BY CERTIFICATE feedbackCertificate;

UPDATE Feedback
SET Rating = ENCRYPTBYKEY(KEY_GUID('feedbackSymmetricKey'), CONVERT(VARBINARY(256),
Rating)),
    Comment = ENCRYPTBYKEY(KEY_GUID('feedbackSymmetricKey'), CONVERT(VARBINARY(256),
Comment));

CLOSE SYMMETRIC KEY feedbackSymmetricKey;


OPEN SYMMETRIC KEY feedbackSymmetricKey
    DECRYPTION BY CERTIFICATE feedbackCertificate;

SELECT FeedbackID,
       MemberID,
       BookID,
       CAST(DECRYPTBYKEY(Rating) AS INT) AS Rating,
       CAST(DECRYPTBYKEY(Comment) AS VARCHAR(255)) AS Comment
FROM Feedback;

CLOSE SYMMETRIC KEY feedbackSymmetricKey;

GO
CREATE TRIGGER insertEncryptedFeedback
ON Feedback
INSTEAD OF INSERT
AS
BEGIN
    OPEN SYMMETRIC KEY feedbackSymmetricKey
        DECRYPTION BY CERTIFICATE feedbackCertificate;

    INSERT INTO Feedback (MemberID, BookID, Rating, Comment)
    SELECT MemberID,
           BookID,
```

```
            ENCRYPTBYKEY(KEY_GUID('feedbackSymmetricKey'), CONVERT(VARBINARY(256),
Rating)),
            ENCRYPTBYKEY(KEY_GUID('feedbackSymmetricKey'), CONVERT(VARBINARY(256),
Comment))
    FROM inserted;

    CLOSE SYMMETRIC KEY feedbackSymmetricKey;
END;
```

Roles And members Added to Data Base:

```
USE AlBayan_Library;
EXECUTE sp_addrole @rolename = 'BranchManager';
EXECUTE sp_addrole @rolename = 'Supervisor';
EXECUTE sp_addrole @rolename = 'Staff';
EXECUTE sp_addrole @rolename = 'Administrator';
EXECUTE sp_addrole @rolename = 'Member';

SELECT *
FROM sys.database_principals
WHERE type = 'R' AND is_fixed_role = 0;

/--Permissions to give still not excecuted need to be revisited and checked
-- Grant permissions to Branch Manager role
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Branch TO BranchManager;
--GRANT EXECUTE ON dbo.StoredProc TO BranchManager;

-- Grant permissions to Supervisor role
GRANT SELECT, UPDATE ON dbo.Staff TO Supervisor;

-- Grant permissions to Staff role
GRANT SELECT, INSERT, UPDATE ON dbo.Member TO Staff;
-- Add more permissions as needed

-- Grant permissions to Member role
GRANT SELECT, INSERT, UPDATE ON dbo.Rental TO Member;
Revoke Select,Insert,Update On dbo.Rental from Member;
GRANT execute on dbo.BorrowBook TO Member;
GRANT execute on dbo.ReturnBook TO Member;

-- Grant permissions to Administrator role
-- Grant all privileges on the Branch table
GRANT ALL PRIVILEGES ON Branch TO Administrator with Grant Option;

-- Grant all privileges on the Staff table
GRANT ALL PRIVILEGES ON Staff TO Administrator with Grant Option;

-- Grant all privileges on the BranchManager table
GRANT ALL PRIVILEGES ON BranchManager TO Administrator with Grant Option;

-- Grant all privileges on the Supervisor table
GRANT ALL PRIVILEGES ON Supervisor TO Administrator with Grant Option;

-- Grant all privileges on the Book table
GRANT ALL PRIVILEGES ON Book TO Administrator with Grant Option;

-- Grant all privileges on the Member table
GRANT ALL PRIVILEGES ON Member TO Administrator with Grant Option;

-- Grant all privileges on the Rental table
GRANT ALL PRIVILEGES ON Rental TO Administrator with Grant Option;

-- Grant all privileges on the Feedback table
GRANT ALL PRIVILEGES ON Feedback TO Administrator with Grant Option;
```

```
-- Grant all privileges on the StaffHistory table
GRANT ALL PRIVILEGES ON StaffHistory TO Administrator;
--Grant execute privliges on all stored procedure to Administrator
GRANT EXECUTE TO Administrator with Grant Option;
```

Users added to different roles

```
USE AlBayan_Library;
Select * from dbo.Staff;
--User 1
EXECUTE sp_addlogin @loginame =  'Lotfi Qasim', @passwd = '@passwd123';
EXECUTE sp_adduser 'Lotfi Qasim', 'lotfi';
execute sp_addrolemember @rolename = 'Administrator',@membername = 'lotfi';
--User 2
EXECUTE sp_addlogin @loginame = 'Mohmamad', @passwd = '@passwd123';
EXECUTE sp_adduser 'Mohmamad', 'Mohmamad';
execute sp_addrolemember @rolename = 'Staff',@membername = 'Mohmamad';
--User 3
EXECUTE sp_addlogin @loginame = 'Ali', @passwd = '@passwd123';
EXECUTE sp_adduser 'Ali', 'Ali';
execute sp_addrolemember @rolename = 'Member',@membername = 'Ali';
--User 4
EXECUTE sp_addlogin @loginame = 'amr', @passwd = '@passwd123';
EXECUTE sp_adduser 'amr', 'amr';
execute sp_addrolemember @rolename = 'Supervisor',@membername = 'amr';
--user 5
EXECUTE sp_addlogin @loginame = 'qusi', @passwd = '@passwd123';
EXECUTE sp_adduser 'qusi', 'qusi';
execute sp_addrolemember @rolename = 'BranchManager',@membername = 'qusi';
```

Authorization Matrices generated

Administrator Role permissions

On Book

Explicit | Effective

Permission	Grantor	Grant	With Gr...	Deny
Delete	dbo	☑	☑	☐
Insert		☐	☐	☐
Insert	dbo	☑	☑	☐
References		☐	☐	☐
References	dbo	☑	☑	☐
Select		☐	☐	☐
Select	dbo	☑	☑	☐
Take ownership		☐	☐	☐
Unmask		☐	☐	☐
Update		☐	☐	☐
Update	dbo	☑	☑	☐
View change tracking		☐	☐	☐
View definition		☐	☐	☐

Figure 0-1Administrator Role permissions on BOOKs

On Branch table:

Permission	Grantor	Grant	With Gr...	Deny
Delete	dbo	☑	☑	☐
Insert		☐	☐	☐
Insert	dbo	☑	☑	☐
References		☐	☐	☐
References	dbo	☑	☑	☐
Select		☐	☐	☐
Select	dbo	☑	☑	☐
Take ownership		☐	☐	☐
Unmask		☐	☐	☐
Update		☐	☐	☐
Update	dbo	☑	☑	☐
View change tracking		☐	☐	☐
View definition		☐	☐	☐

Figure 0-2Administrator Role permissions on Branch

On Branch Manager Table

Permission	Grantor	Grant	With Gr...	Deny
Delete	dbo	☑	☑	☐
Insert		☐	☐	☐
Insert	dbo	☑	☑	☐
References		☐	☐	☐
References	dbo	☑	☑	☐
Select		☐	☐	☐
Select	dbo	☑	☑	☐
Take ownership		☐	☐	☐
Unmask		☐	☐	☐
Update		☐	☐	☐
Update	dbo	☑	☑	☐
View change tracking		☐	☐	☐
View definition		☐	☐	☐

Figure 0-3 Administrator Role permissions on Branch Manager

On FeedBack

Permission	Grantor	Grant	With Gr...	Deny
Delete	dbo	☑	☑	☐
Insert		☐	☐	☐
Insert	dbo	☑	☑	☐
References		☐	☐	☐
References	dbo	☑	☑	☐
Select		☐	☐	☐
Select	dbo	☑	☑	☐
Take ownership		☐	☐	☐
Unmask		☐	☐	☐
Update		☐	☐	☐
Update	dbo	☑	☑	☐
View change tracking		☐	☐	☐
View definition		☐	☐	☐

Figure 0-4Administrator Role permissions on FeedBack

On Member table

Permission	Grantor	Grant	With Gr...	Deny
Delete	dbo	☑	☑	☐
Insert		☐	☐	☐
Insert	dbo	☑	☑	☐
References		☐	☐	☐
References	dbo	☑	☑	☐
Select		☐	☐	☐
Select	dbo	☑	☑	☐
Take ownership		☐	☐	☐
Unmask		☐	☐	☐
Update		☐	☐	☐
Update	dbo	☑	☑	☐
View change tracking		☐	☐	☐
View definition		☐	☐	☐

Figure 0-5 Administrator Role permissions on Member

And much more authorization tables Which were not shown here but can be understood by looking at the code provided.

Conclusion:

Throughout the course we have learned how to think more rationally and got a better understanding of what and how Data Base works, we have been introduced to the shallowest point of the depth of a database administrator work, and we have been introduced to stored procedure triggers and much more which we have implied and practiced throughout this project.