Documentation for Data Security Project

1.DataBase creation:

-We created database called”SecureStudentDB”

```
CREATE DATABASE SecureStudentDB;
GO
```

---

2. Create keys such as:

Master Key:

Certificate Key:

Symmetric Key:

```
-- Create Master Key
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'StrongMasterKey@2024!'
GO

-- Create Certificate for Encryption
CREATE CERTIFICATE StudentDataCert
WITH SUBJECT = 'Certificate for Student Data Encryption';
GO


-- Create Symmetric Key for AES Encryption
CREATE SYMMETRIC KEY StudentDataKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE StudentDataCert;
GO
```

3.Know our Clearance Level:

-- Clearance Levels:-

-- 0 = Unclassified (Guest)

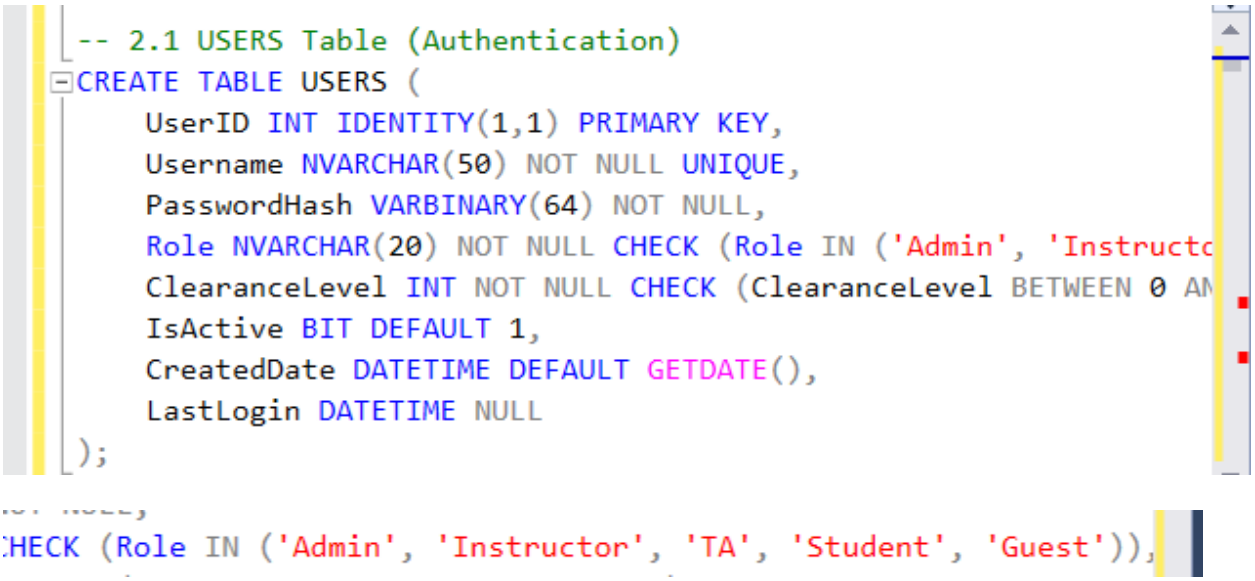
-- 1 = Confidential (Student)

-- 2 = Secret (TA)

-- 3 = Top Secret (Instructor)

-- 4 = Top Secret+ (Admin)

4.Create USERS table: -



```
-- 2.1 USERS Table (Authentication)
CREATE TABLE USERS (
    UserID INT IDENTITY(1,1) PRIMARY KEY,
    Username NVARCHAR(50) NOT NULL UNIQUE,
    PasswordHash VARBINARY(64) NOT NULL,
    Role NVARCHAR(20) NOT NULL CHECK (Role IN ('Admin', 'Instructo
    ClearanceLevel INT NOT NULL CHECK (ClearanceLevel BETWEEN 0 AN
    IsActive BIT DEFAULT 1,
    CreatedDate DATETIME DEFAULT GETDATE(),
    LastLogin DATETIME NULL
);
```



```
HECK (Role IN ('Admin', 'Instructor', 'TA', 'Student', 'Guest')),
```

This table as user should insert some Data such as:

-id

-username (making it NVARCHAR to support different languages as he can assign name in Arabic or English and so on).

-Password Hash: our password is “VARBINARY” to hashing it to binary 0’s and 1’s and cannot get them again but it is for verifying.

-Role: check his Role if he is (guest /student/TA/instructor/Admin).

-then check his clearance level based on his role (it is an authentication and authorization).

-Is active >> to check if his account valid or suspend.

-Created data: he created it when.

-Last Login: check when he last login

If he is the first time so his value is null which is acceptable.

5.Create Student Table confidential:

This table tracks some data about Students such as:

Student-id

Student-id encrypted>>making it encrypted to decrypt when we have the key.

- **Full name (not null) - Email (not null) - Phone(encrypted)**
- **DOB =Date of birth**
- **Department (not null)**
- **USERID which is connected to USERS table.**

```
CREATE TABLE STUDENT (
    StudentID INT IDENTITY(1,1) PRIMARY KEY,
    StudentID_Encrypted VARBINARY(256) NOT NULL,
    FullName NVARCHAR(100) NOT NULL,
    Email NVARCHAR(100) NOT NULL,
    Phone_Encrypted VARBINARY(256) NULL,
    DOB DATE NOT NULL,
    Department NVARCHAR(50) NOT NULL,
    ClearanceLevel INT DEFAULT 1,
    UserID INT NULL,
    ClassificationLevel INT DEFAULT 1,
    FOREIGN KEY (UserID) REFERENCES USERS(UserID)
);
GO
```

---

6. Create Instructor Table:(Confidential): -

We will track some data about Instructor

-Instructor ID (pk)" Just for linking with other tables".

-Full Name (Not Null)

-Email (NOT Null)

-Clearence Level(default=3).

-Classification Level (Default =1).

-User-id is foreign key to User-id in Users table.

```
-- 2.3 INSTRUCTOR Table (Confidential)
CREATE TABLE INSTRUCTOR (
    InstructorID INT IDENTITY(1,1) PRIMARY KEY,
    FullName NVARCHAR(100) NOT NULL,
    Email NVARCHAR(100) NOT NULL,
    ClearanceLevel INT DEFAULT 3,
    UserID INT NULL,
    ClassificationLevel INT DEFAULT 1,
    FOREIGN KEY (UserID) REFERENCES USERS(UserID)
);
GO
```

7.Create Course table:-

```
CREATE TABLE COURSE (
    CourseID INT IDENTITY(1,1) PRIMARY KEY,
    CourseName NVARCHAR(100) NOT NULL,
    Description NVARCHAR(MAX) NULL,
    PublicInfo NVARCHAR(MAX) NULL,
    InstructorID INT NULL,
    ClassificationLevel INT DEFAULT 0,
    FOREIGN KEY (InstructorID) REFERENCES INSTRUCTOR(InstructorID)
);
GO
```

Some Data we tracked on Course Table:

-Course-id IDENTITY >> used for tables linking.

-Course-name>> Required NOTNULL

-Description>>may have sensitive data so it has classification level as not any one can access it.

It can be null (not required).

-Public Info: a general info any one can know and it can be null (not required).

-Instructor-id (it can be null) >> it is a foreign key to instructor table.

-classification level: default is 0 as any one can access unless we assign restricted value to classification level.

---

8. Create Grades Table: -

We track some data about grades…such as:

-Garde-id>>identity (used to Join tables).

-student-encrypted-id: To secure student identity.

-Course-id which is a (foreign key) from Course table.

-Grade-value encrypted>>not null

-Date Entered >>Datetime default: keep our date.

-Entered by>> int as it is user-id (not null).

-classification: Entered by person has clearance level 2 or more.

```
-- 2.5 GRADES Table (Secret)
CREATE TABLE GRADES (
    GradeID INT IDENTITY(1,1) PRIMARY KEY,
    StudentID_Encrypted VARBINARY(256) NOT NULL,
    CourseID INT NOT NULL,
    GradeValue_Encrypted VARBINARY(256) NOT NULL,
    DateEntered DATETIME DEFAULT GETDATE(),
    EnteredBy INT NOT NULL,
    ClassificationLevel INT DEFAULT 2,
    FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID)
);
GO
```

9.Create Attendance table:

It is an attendance table as we tracked some data

Such as:

-attendance-id which is a Primary key and identity.

-Student-id (not null)>> required

-course-id (not null)

-Status (not null)

-Date-Recorded>>automatically

-Record-by (not null)

-classification level (default=2)

-Student-id as fk with students table, course-id as fk with courses table.

```
-- 2.6 ATTENDANCE Table (Secret)
CREATE TABLE ATTENDANCE (
    AttendanceID INT IDENTITY(1,1) PRIMARY KEY,
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    Status BIT NOT NULL,
    DateRecorded DATETIME DEFAULT GETDATE(),
    RecordedBy INT NOT NULL,
    ClassificationLevel INT DEFAULT 2,
    FOREIGN KEY (StudentID) REFERENCES STUDENT(StudentID),
    FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID)
);
GO
```

---

10. create TA_ASSIGNMENTS: -

-we will track some data about it such as:

-assignment-id >> used in linking table

-user-id (Not null)

-course-id not null

-assigned date>>automatically filled.

-user-id as fk from users tablet, and course-id as fk from courses table.

```
-- 2.7 TA_ASSIGNMENTS Table
CREATE TABLE TA_ASSIGNMENTS (
    AssignmentID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    CourseID INT NOT NULL,
    AssignedDate DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (UserID) REFERENCES USERS(UserID),
    FOREIGN KEY (CourseID) REFERENCES COURSE(CourseID)
);
GO
```

11.Create Audit Log table:

```
CREATE TABLE AUDIT_LOG (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    Action NVARCHAR(100) NOT NULL,
    TableName NVARCHAR(50) NULL,
    RecordID INT NULL,
    OldValue NVARCHAR(MAX) NULL,
    NewValue NVARCHAR(MAX) NULL,
    Timestamp DATETIME DEFAULT GETDATE(),
    IPAddress NVARCHAR(50) NULL,
    Success BIT DEFAULT 1
);
GO
```

We will track some data about this table such as:

Log id >> identity for Linking with another table.

User-id not null

Action not null>>required

Table name may be null

Record id may be null

Time-Stamp a date can be filled automatically

IP address can be null

Success as bit can be 1 or 0

===We finished Table creation =====

Step2: Making RBAC=" Role Based Access Control"

1. **Create Roles:**

```
-- STEP 3: CREATE SQL SERVER ROLES (RBAC)
-- ============================================

CREATE ROLE AdminRole;
CREATE ROLE InstructorRole;
CREATE ROLE TARole;
CREATE ROLE StudentRole;
CREATE ROLE GuestRole;
GO
```

So, we have 5 Roles such as:

Admin

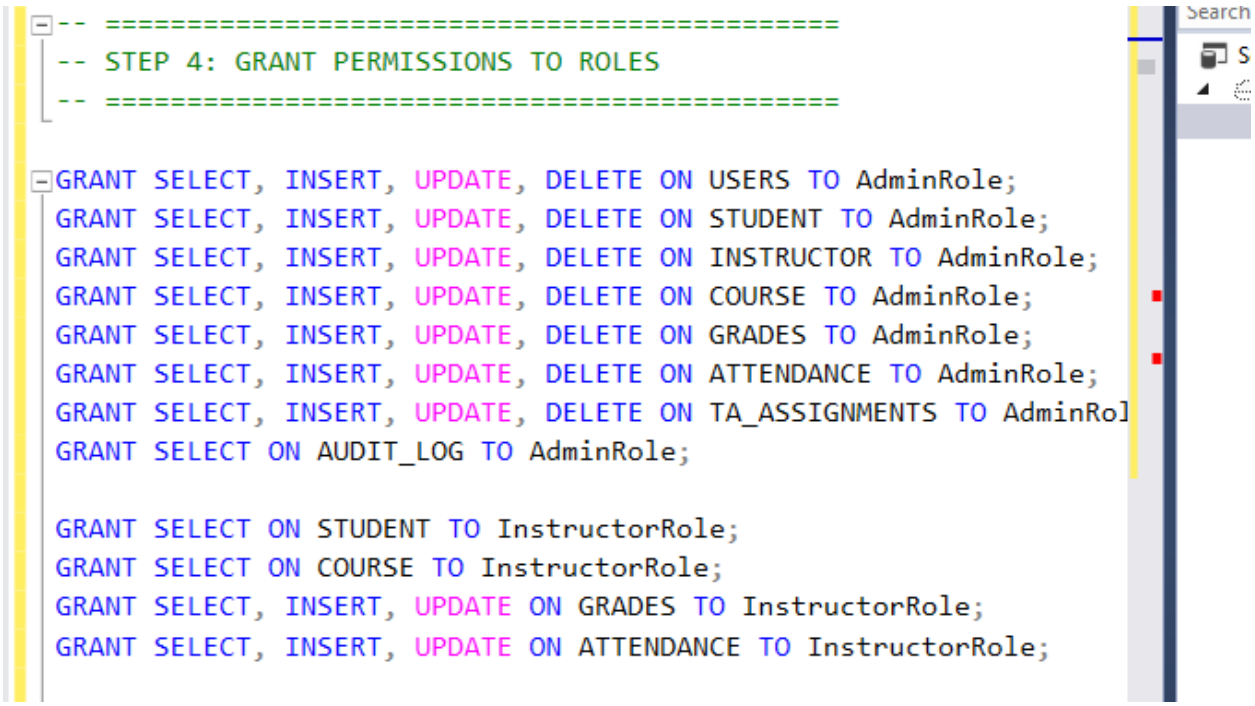
TA

Instructor

Student

Guest

. 2.Grant Permissions to Roles:



```
-- ============================================
-- STEP 4: GRANT PERMISSIONS TO ROLES
-- ============================================

GRANT SELECT, INSERT, UPDATE, DELETE ON USERS TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON STUDENT TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON INSTRUCTOR TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON COURSE TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON GRADES TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON ATTENDANCE TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON TA_ASSIGNMENTS TO AdminRol
GRANT SELECT ON AUDIT_LOG TO AdminRole;

GRANT SELECT ON STUDENT TO InstructorRole;
GRANT SELECT ON COURSE TO InstructorRole;
GRANT SELECT, INSERT, UPDATE ON GRADES TO InstructorRole;
GRANT SELECT, INSERT, UPDATE ON ATTENDANCE TO InstructorRole;
```

Admin has the privilege to (select, insert, update, delete) all tables except Audit-log can select only.

Instructor can select from Student and Course

But can insert, select, update, delete from other tables.

```
GRANT SELECT ON STUDENT TO TARole;
GRANT SELECT ON COURSE TO TARole;
GRANT SELECT ON GRADES TO TARole;
GRANT SELECT, INSERT, UPDATE ON ATTENDANCE TO TARole;

GRANT SELECT ON COURSE TO StudentRole;
GRANT SELECT ON COURSE TO GuestRole;
GO
```

Student can select from Course

Guest can select from Course.

Step3: Helper Function such as:

Function1: - Hashing Password:

This function accepts Password (Varchar) and returns Binary varchar (64) in 64 bits.

This password hashed and this hashing put into USERS table.

--Testing for function 1—

For Hashing Password.

```
--another way to execute function1"Hashing Password"--
INSERT INTO USERS (Username, PasswordHash, Role, ClearanceLevel)
VALUES ('student1', dbo.HashPassword('MySecret123'), 'Student', 1);

Select*from USERS;
```

124 %

Results | Messages

	UserID	Username	PasswordHash	Role	ClearanceLevel	IsActive	CreatedDate	LastLogin
1	1	student1	0x4A0DF2AB8A12ABCE6A6A2A9177A09871D816502C6C9131...	Student	1	1	2025-12-20 16:47:36.403	NULL

```
GO
---Testing for function 1---

SELECT dbo.HashPassword('MySecret123') AS HashedValue;
```

Results | Messages

HashedValue
0x4A0DF2AB8A12ABCE6A6A2A9177A09871D816502C6C9131...

Function2:" To check Clearance"

```
-- Clearance check function 2--
CREATE FUNCTION dbo.CheckClearance(@UserClearance INT, @DataClearance INT)
RETURNS BIT
AS
BEGIN
    IF @UserClearance >= @DataClearance
        RETURN 1;
    RETURN 0;
END
GO
```

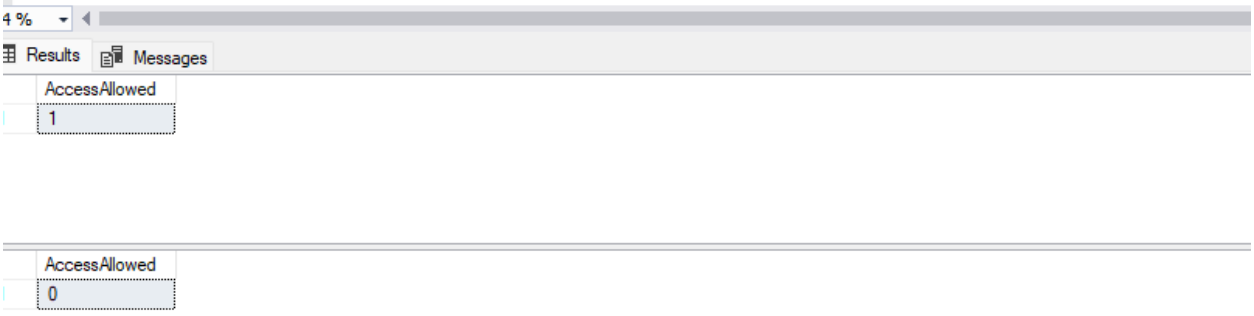
This function accepts User Clearance and Data Clearance…. return Bit 1 if True and 0 if false.

Check if User clearance is larger say yes.

Else no.

To test function 2: -

```
----test function2 Clearance ---
SELECT dbo.CheckClearance(2, 1) AS AccessAllowed;
SELECT dbo.CheckClearance(1, 2) AS AccessAllowed;
```



Stored Procedure: For Encryption:

```
CREATE PROCEDURE sp_EncryptData
    @PlainText NVARCHAR(MAX),
    @EncryptedData VARBINARY(256) OUTPUT
AS
BEGIN
    OPEN SYMMETRIC KEY StudentDataKey
    DECRYPTION BY CERTIFICATE StudentDataCert;

    SET @EncryptedData = ENCRYPTBYKEY(KEY_GUID('StudentDataKey'), @PlainText);

    CLOSE SYMMETRIC KEY StudentDataKey;
END
GO
```

This procedure accepts Plaintext and converts it Encrypted Data (Var binary) …using Symmetric key.

To test this procedure: -

Stored Procedure for Decryption: -

```
---Decrypt Data Procedure--

CREATE PROCEDURE sp_DecryptData
    @EncryptedData VARBINARY(256),
    @PlainText NVARCHAR(MAX) OUTPUT
AS
BEGIN
    OPEN SYMMETRIC KEY StudentDataKey
    DECRYPTION BY CERTIFICATE StudentDataCert;

    SET @PlainText = CONVERT(NVARCHAR(MAX), DECRYPTBYKEY(@EncryptedData));

    CLOSE SYMMETRIC KEY StudentDataKey;
END
GO
```

Accepts Encrypted data then try to decrypt it.

And convert it from Var binary to Varchar.

Step 4: AUTHENTICATION PROCEDURES: -

1. **Procedure for Register and another for Login: -**

   Steps:

   Take data

Then check if it is the first register.

Then encrypt his password.

Add it to Users table.

Login: check the validity of name.

Check user is valid…. update last login table and Audit trial.

```
CREATE PROCEDURE sp_Login
    @Username NVARCHAR(50),
    @Password NVARCHAR(100),
    @UserID INT OUTPUT,
    @Role NVARCHAR(20) OUTPUT,
    @ClearanceLevel INT OUTPUT,
    @Success BIT OUTPUT,
    @Message NVARCHAR(200) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @StoredHash VARBINARY(64);
    DECLARE @InputHash VARBINARY(64);

    SELECT @StoredHash = PasswordHash,
           @UserID = UserID,
           @Role = Role,
           @ClearanceLevel = ClearanceLevel
```

```
EXEC sp_RegisterUser 'dr_smith', 'Instructor@123', 'Instructor', 3, @UserID OUTPUT, @Result OUT
PRINT 'Instructor: ' + @Message;

EXEC sp_RegisterUser 'ta_john', 'TA@123', 'TA', 2, @UserID OUTPUT, @Result OUTPUT, @Message OUT
PRINT 'TA: ' + @Message;

EXEC sp_RegisterUser 'student1', 'Student@123', 'Student', 1, @UserID OUTPUT, @Result OUTPUT, @
PRINT 'Student1: ' + @Message;

EXEC sp_RegisterUser 'student2', 'Student@123', 'Student', 1, @UserID OUTPUT, @Result OUTPUT, @
PRINT 'Student2: ' + @Message;

EXEC sp_RegisterUser 'guest', 'Guest@123', 'Guest', 0, @UserID OUTPUT, @Result OUTPUT, @Message
PRINT 'Guest: ' + @Message;
```

Messages

```
Admin: User registered successfully
Instructor: User registered successfully
TA: User registered successfully
Student1: Username already exists
Student2: User registered successfully
Guest: User registered successfully

Completion time: 2025-12-20T18:37:42.1380057+02:00
```

```
CREATE PROCEDURE sp_RegisterUser
    @Username NVARCHAR(50),
    @Password NVARCHAR(100),
    @Role NVARCHAR(20),
    @ClearanceLevel INT,
    @UserID INT OUTPUT,
    @Result INT OUTPUT,
    @Message NVARCHAR(200) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS (SELECT 1 FROM USERS WHERE Username = @Username)
        BEGIN
            SET @Result = 0;
            SET @Message = 'Username already exists';
            SET @UserID = NULL;
            RETURN;
        END

        DECLARE @PasswordHash VARBINARY(64);
```

Multilevel views: -

```sql
--First view--
CREATE VIEW vw_UnclassifiedCourses
AS
SELECT CourseID, CourseName, PublicInfo
FROM COURSE
WHERE ClassificationLevel = 0;
GO

--Second view--
CREATE VIEW vw_ConfidentialStudentData
AS
SELECT s.StudentID, s.FullName, s.Email, s.Department
FROM STUDENT s
WHERE s.ClassificationLevel <= 1;
GO
--Third view--
CREATE VIEW vw_SecretGradeData
AS
SELECT g.GradeID, g.CourseID, g.DateEntered
FROM GRADES g
WHERE g.ClassificationLevel <= 2;
GO
```

Inference control: firstly, check Access Control: as clearance>=2. (TA …. up).

+ Inference control as minimum size check count<3

+Data Encryption.

--Flow control:

Prevents users from writing data to a lower classification level (“no write-down”), enforcing mandatory access control (MAC). Uses the inserted pseudo-table to validate rows before insertion or update, and rolls back any unauthorized attempts to ensure data integrity and security.

--Select *from instructors;

```
Select*from INSTRUCTOR;

--Insert in course table--
INSERT INTO COURSE (CourseName, Description, PublicInfo, InstructorID)
```

124 %

Results | Messages

	InstructorID	FullName	Email	ClearanceLevel	UserID	ClassificationLevel
1	1	Dr. John Smith	dr.smith@university.edu	3	3	1

Select*from Course;

```
--To check--
Select*from COURSE;
```

24 %

Results | Messages

	CourseID	CourseName	Description	PublicInfo	InstructorID	ClassificationLevel
1	1	Database Security	Advanced database security concepts	Learn database security fundamentals	1	0
2	2	Network Security	Network security and cryptography	Explore network security protocols	1	0
3	3	Web Security	Web application security	Secure web development practices	1	0
4	4	Database Security	Advanced database security concepts	Learn database security fundamentals	1	0
5	5	Network Security	Network security and cryptography	Explore network security protocols	1	0
6	6	Web Security	Web application security	Secure web development practices	1	0

View Public Course Procedure:

```
--View Public Course procedure--

EXEC sp_ViewPublicCourses;
GO
```

124 %

Results | Messages

	CourseID	CourseName	PublicInfo
1	1	Database Security	Learn database security fundamentals
2	2	Network Security	Explore network security protocols
3	3	Web Security	Secure web development practices
4	4	Database Security	Learn database security fundamentals
5	5	Network Security	Explore network security protocols
6	6	Web Security	Secure web development practices

To check sp_AddStudent:

```
Select*from STUDENT;
```

Results | Messages

	StudentID	StudentID_Encrypted	FullName	Email	Phone_Encrypted
1	1	0x001CC72EB5012D4BBE49EDF0D30BDBD002000000D3AB6E...	Alice Johnson	alice.johnson@university.edu	0x001CC72EB5012D4BBE49EDF0D30BDBD

Part B in Project:

Request table:

```
CREATE TABLE ROLE_REQUESTS (
    RequestID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    CurrentRole NVARCHAR(20) NOT NULL,
    RequestedRole NVARCHAR(20) NOT NULL CHECK (RequestedRole IN ('TA', 'Instructor', 'Admin')),
    Reason NVARCHAR(500) NOT NULL,
    Comments NVARCHAR(MAX) NULL,
    Status NVARCHAR(20) NOT NULL DEFAULT 'Pending' CHECK (Status IN ('Pending', 'Approved', 'De
    RequestDate DATETIME DEFAULT GETDATE(),
    ReviewedBy INT NULL,
    ReviewDate DATETIME NULL,
    ReviewComments NVARCHAR(MAX) NULL,
    FOREIGN KEY (UserID) REFERENCES USERS(UserID),
```

he ROLE_REQUESTS table is used to manage role escalation requests within the system.
It allows users to request higher-privilege roles (such as TA, Instructor, or Admin) in a controlled and auditable manner.
Each request must be reviewed and approved or denied by an authorized administrator, ensuring proper access control and preventing unauthorized privilege escalation.

Step 2: to accelerate this process and making it fast: - (using Indexing).

Indexes are created on the Status and UserID columns of the ROLE_REQUESTS table to improve query performance.
These indexes optimize frequent operations such as retrieving pending requests for review and fetching role requests for a specific user.

Step3: Creating Procedure to sp_SubmitRoleRequest:-

1.check user-existence:

IF NOT EXISTS (SELECT 1 FROM USERS WHERE UserID = @UserID).

2.Enforce him to choose a role from our List (Admin, TA, Guest, Instructor, Std).

3. Prevent **Role Downgrade**

4.Reason is mandatory

5.we cannot make a request for the second time as it is pending.

6.update Audit log

Step4: Create Stored Procedure called sp_ViewMyRoleRequests: -

This stored procedure allows a user to securely view all their role request submissions.
It ensures that users can only access their own requests, provides full request history with review details, and records the access operation in the audit log for accountability.

Horizontal Privilege Escalation

No one can see others' requests.

New requests appear Firstly.

Step5: Create sp_ViewPendingRoleRequests: -

MAC is applied as only admin can see this Page.

WHERE rr.Status = 'Pending' (show only pending Requests).

We make (inner Join) to check that every request has real user.

Make left join with Students…

DaysPending: calculate pending days then reorder requests Asc.

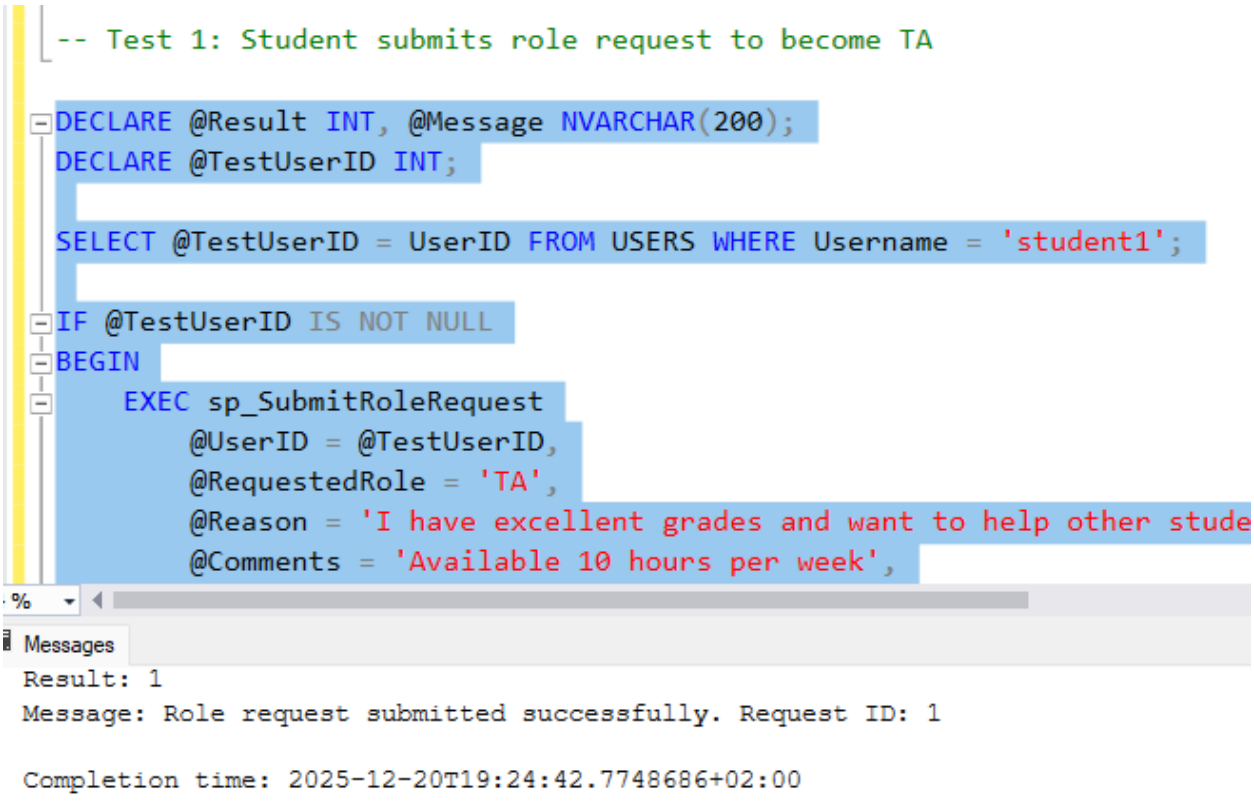
Step5: Propcedure: **ApproveRoleRequest:-**

This stored procedure allows authorized administrators to securely approve role upgrade requests.
It enforces clearance-level access control, validates request state, updates user roles and clearance levels atomically, and records all actions in the audit log to ensure accountability and traceability.

Step 6:Deny Role Request:

This procedure allows administrators to securely deny role upgrade requests.
It enforces clearance-level access, validates request state, requires mandatory review comments, updates the request status, and logs all actions in the audit log for accountability.

Sending Request for the first time:



For the second time:

It said there is already a pending Role.

```
        @Reason = 'I have excellent grades and want to help other
        @Comments = 'Available 10 hours per week',
        @Result = @Result OUTPUT,
        @Message = @Message OUTPUT;

    PRINT 'Result: ' + CAST(@Result AS NVARCHAR(10));
    PRINT 'Message: ' + @Message;
END
GO
```

124 %

Messages

```
Result: 0
Message: You already have a pending role request

Completion time: 2025-12-20T19:25:25.8169811+02:00
```

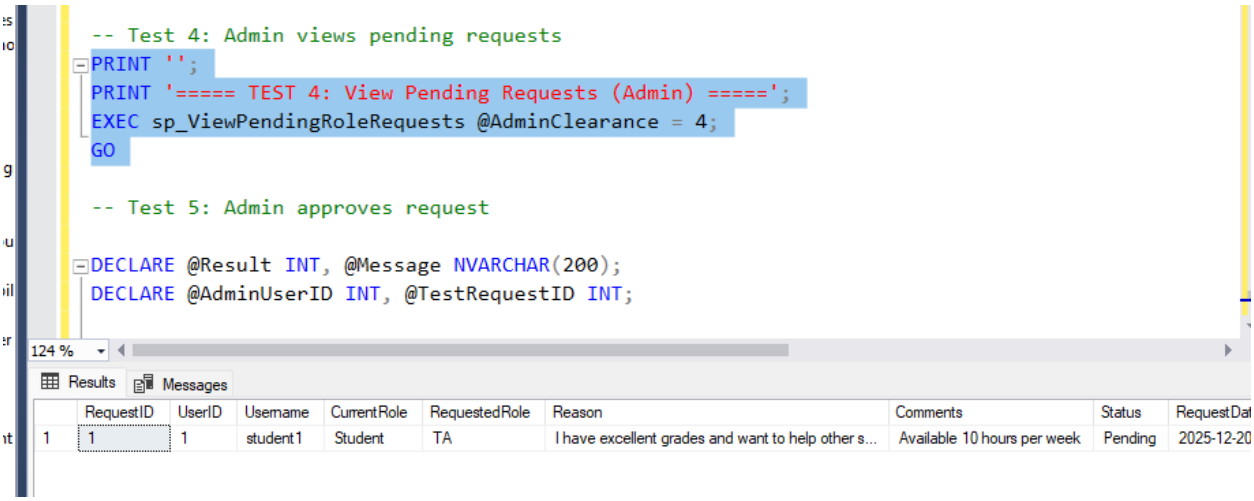
View my request Role:

```
-- Test 3: View my role requests
DECLARE @TestUserID INT;
SELECT @TestUserID = UserID FROM USERS WHERE Username = 'student1';

IF @TestUserID IS NOT NULL
    EXEC sp_ViewMyRoleRequests @UserID = @TestUserID;
GO

-- Test 4: Admin views pending requests
PRINT '';
PRINT '===== TEST 4: View Pending Requests (Admin) =====';
EXEC sp_ViewPendingRoleRequests @AdminClearance = 4;
GO
```

Results | Messages

RequestID	CurrentRole	RequestedRole	Reason	Comments	Status	RequestDate	ReviewI
1	Student	TA	I have excellent grades and want to help other s...	Available 10 hours per week	Pending	2025-12-20 19:24:42.767	NULL

Admin views Pending Requests:



```
-- Test 4: Admin views pending requests
PRINT '';
PRINT '===== TEST 4: View Pending Requests (Admin) =====';
EXEC sp_ViewPendingRoleRequests @AdminClearance = 4;
GO

-- Test 5: Admin approves request

DECLARE @Result INT, @Message NVARCHAR(200);
DECLARE @AdminUserID INT, @TestRequestID INT;
```

124 %

Results | Messages

	RequestID	UserID	Username	CurrentRole	RequestedRole	Reason	Comments	Status	RequestDat
1	1	1	student1	Student	TA	I have excellent grades and want to help other s...	Available 10 hours per week	Pending	2025-12-20

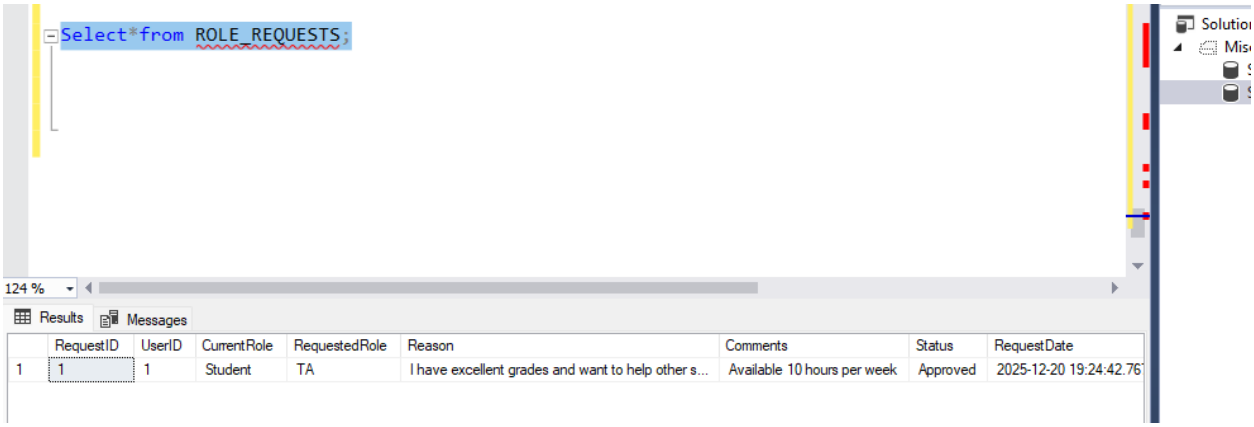
Admin approve requests:

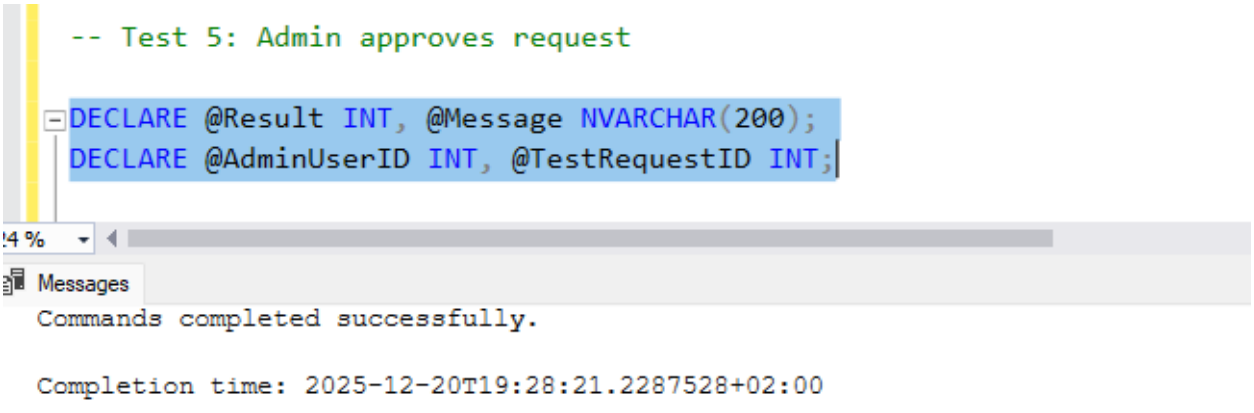
```
-- Test 5: Admin approves request

DECLARE @Result INT, @Message NVARCHAR(200);
DECLARE @AdminUserID INT, @TestRequestID INT;
```

Messages

```
Commands completed successfully.

Completion time: 2025-12-20T19:28:21.2287528+02:00
```

Admin Approve request:





Security Metrics:

Function / Data	Admin	Instructor	TA	Student	Guest
View own profile	✓	✓	✓	✓	✕
Edit own profile	✓	✓	✓	✕	✕
View grades	✓	✓	✕	✕	✕
Edit grades	✓	✓	✕	✕	✕
View attendance	✓	✓	✓	✓ (own)	✕
Edit attendance	✓	✓	✓	✕	✕
Manage users	✓	✕	✕	✕	✕
View public course info.	✓	✓	✓	✓	✓

Team members:

Salma Salah

Mariam Adelfattah

Fatema Tarek

Nada Waleed

Ahmed Tarek