
# GROUP ASSIGNMENT

# ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION

# CT069-3-3 - Database Security

| | |
|---|---|
| Intake: | APD3F2305CS(CYB) |
| Hand Out Date: | Week 3 |
| Submission Date: | 22th December 2023 |

| Group Members | |
|---|---|
| **Student Name** | **ID** |
| TAN YAN SHEN | TP063627 |
| LEE WOON XUN | TP067738 |
| LEE JUN SHENG | TP056821 |

# Table of Contents

# 1.0 Introduction

## 1.1 Project Background

APU Medical Center is a hospital in Bukit Jalil, Kuala Lumpur. A medical information system has been developed to manage its day-to-day operation which includes patient, staff, and medication details. However, it is found that many security features are missing from the database design and implementation of the system.

In the healthcare industry, data security is very important. As there is a lot of personal health information like treatments, medical history, and so on, it is critical to ensure the confidentiality and security of this data. It is not only to align with the regulatory compliance standards but also to build trust and confidence between the patients and medical providers (ECCU, n.d.).

In this project, the security of the APU Medical Center database system will be discussed and enhanced to safeguard sensitive patients and medical data. In order to improve database security, data protection, permission management, and auditing will be discussed and implemented in this project. Furthermore, the potential threats to data security with their solutions will be discussed. All of the implementations will be designed based on the provided functional and security requirements. Lastly, the selected solutions will be implemented for the APU Medical Center database.

## 1.2 Workload Matrix

| | Percentages | | |
|---|---|---|---|
| **Tasks** | **Tan Yan Shen**<br>**TP063627** | **Lee Woon Xun**<br>**TP067738** | **Lee Jun Sheng**<br>**TP056821** |
| **Implementation** | | | |
| **Database Structure** | **30%** | **30%** | **30%** |
| **Data Population** | **30%** | **30%** | **30%** |
| **Permission Management** | **0%** | **100%** | **0%** |
| **Data Protection** | **100%** | **0%** | **0%** |
| **Auditing** | **0%** | **0%** | **100%** |
| **Testing** | **30%** | **30%** | **30%** |
| **Documentation** | | | |
| **Introduction** | **30%** | **30%** | **30%** |
| **Permission Management** | **0%** | **100%** | **0%** |
| **Data Protection** | **100%** | **0%** | **0%** |
| **Auditing** | **0%** | **0%** | **100%** |
| **Summary** | **30%** | **30%** | **30%** |
| **References** | **30%** | **30%** | **30%** |
| | | | |
| ***Total*** | ***33%*** | ***33%*** | ***33%*** |
| ***Signature*** | Shen | WoonXun | JunSheng |

## 1.3 Data Dictionary

The data dictionary for the APU Medical Center Database (MedicalInfoSystem) will be designed as below:

**Table: Staff**

| Purpose: Stores all the staff details | | | |
|---|---|---|---|
| Column Name | Type | Default | Note |
| StaffID | varchar(6) | | PK |
| SName | varchar(100) | | not null |
| SPassportNumber | varchar(50) | | not null |
| SPhone | varchar(20) | | |
| SystemUserID | varchar(10) | | unique, not null |
| Position | varchar(20) | | Doctor, Nurse |

**Table: Patient**

| Purpose: Stores all the registered patient details | | | |
|---|---|---|---|
| Column Name | Type | Default | Note |
| PID | varchar(6) | | PK |
| PName | varchar(100) | | not null |
| PPassportNumber | varchar(50) | | not null |
| PPhone | varchar(20) | | |
| SystemUserID | varchar(10) | | unique, not null |
| PaymentCardNumEncrypted | varbinary(max) | | |
| PaymentCardPinCode | varchar(20) | | |

**Table: Medicine**

| Purpose: Stores all the medicine ID and name relevant to this system | | | |
|---|---|---|---|
| Column Name | Type | Default | Note |
| MID | varchar(10) | | PK |
| MName | varchar(50) | | not null |

**Table: Prescription**

| Purpose: Stores all the prescription details relevant to this system | | | |
|---|---|---|---|
| Column Name | Type | Default | Note |
| PresID | int | | Identity column, PK |
| PatientID | varchar(6) | | FK references Patient(PID) |
| DoctorID | varchar(6) | | FK references Staff(StaffID) |
| PresDateTime | datetime | Current date and time | Getdate(), not null |

**Table: PrescriptionMedicine**

| Purpose: Show the relationship between prescription and medicine (many to many) | | | |
|---|---|---|---|
| Column Name | Type | Default | Note |
| PresID | int | | FK references Prescription(PresID) |
| MedID | varchar(10) | | FK references Medicine(MID) |
| Notes: The primary key of this table is the combination of PresID and MID, (PresID,MID). | | | |

# 2.0 Permission Management

There are many risks and threats in the system. Designers, developers, and even users will worry because it is related to the security of the system. Among these threats, the most important protection method is permission management and access control as it is the most frequently attacked port.

## 2.1 Threats and Importance of Permission Control

Permission control and access control refer to the positioning of account identities and the power control to access the system. If the administrator does not restrict account permissions and access control, then when any account is stolen, the attacker can access any area, including sensitive data, and an intentional person can also do all malicious behaviors, such as tampering or deleting others' information. This threat will have great negative effects on the organization's reputation, impact, and even customer trust, and may even bring economic problems or legal proceedings.

**Unauthorized access to the database**

Unauthorized entrance or usage of a database by people or organizations without the necessary authorizations or credentials is referred to as unauthorized access to the database. As a result, sensitive information kept in the database may be viewed, retrieved, altered, or deleted.

**Unauthorized changes to data / Lack of proper access control**

Unauthorized changes or updates to data inside a database pose a threat. It usually results from inadequate access restrictions that permit users to make changes that they should not, whether on purpose or accidentally.

## 2.2 Authorization Matrix

| Task | Patient | Nurses | Doctor | Admin |
|---|---|---|---|---|
| **Personal Detail** | **Read, Update** | **Read, Update** | **Read, Update** | **Full Access** |
| **Patient' Personal Detail** | **Read, Update** | **Read, Update** | **Read, but not access to update or delete.** | **Full Access** |
| **Medication detail by Own Patient** | **Not Access** | **Read** | **Read, Write, Update, Delete** | **Full Access** |
| **Medication detail by Own Patient** | **Not Access** | **Not Access** | **Not Access to modify another doctor's patient** | **Full Access** |

*Table 1: Example of Authorization Matrix*

| | RoleName | TableName | PermissionType |
|---|---|---|---|
| 1 | DoctorRole | Patient | SELECT |
| 2 | DoctorRole | Prescription | SELECT, INSERT, UPDATE, DELETE |
| 3 | DoctorRole | PrescriptionMedicine | SELECT, INSERT, UPDATE, DELETE |
| 4 | DoctorRole | Medicine | SELECT |
| 5 | DoctorRole | Staff | SELECT, UPDATE |
| 6 | NurseRole | Patient | SELECT, UPDATE |
| 7 | NurseRole | Staff | SELECT, UPDATE |
| 8 | NurseRole | Prescription | SELECT |
| 9 | NurseRole | PrescriptionMedicine | SELECT |
| 10 | NurseRole | Medicine | SELECT |
| 11 | PatientRole | Patient | SELECT,UPDATE |
| 12 | PatientRole | Prescription | SELECT |
| 13 | PatientRole | PrescriptionMedicine | SELECT |
| 14 | PatientRole | Medicine | SELECT |
| 15 | AdminRole | Patient | SELECT,INSERT,UPDATE,DELETE |
| 16 | AdminRole | Prescription | SELECT,INSERT,UPDATE,DELETE |
| 17 | AdminRole | Medicine | SELECT,INSERT,UPDATE,DELETE |

*Figure 1: Present the Authorization matrix in medical system database*

The authorization matrix represents the system's permission identity allocation table, which means that the account allocation structure and permissions can be understood from the table. According to Table 1, the resources of task, patient, nurses, doctor, and admin, and several keywords such as read, write, update, delete, not access, and full access are also included. These keywords are the permission controls where the users are defined for their identities. The permission controls limit the resources that can be accessed, and the types of operations that can be performed based on the given privileges. This method can also largely reduce the occurrence of unauthorized access and operations to the system.

```
126 --------------------Add User Login -------------------------
127 --Doctor Account - SystemUserID: user001
128 CREATE USER user001 FOR LOGIN user001
129 --add to role group [which role] ADD who
130 ALTER ROLE DoctorRole ADD member user001
131 --ALTER ROLE DoctorRole DROP member user001; --Delect user001 from DoctorRole
132
133 --Nurse Account - SystemUserID: user002
134 CREATE USER user002 FOR LOGIN user002
135 ALTER ROLE NurseRole ADD member user002
136
137
138 --Patient Account - SystemUserID: user101
139 CREATE USER user101 FOR LOGIN user101
140 ALTER ROLE PatientRole ADD member user101
```

*Figure 2: Users at APU Medical Centre*

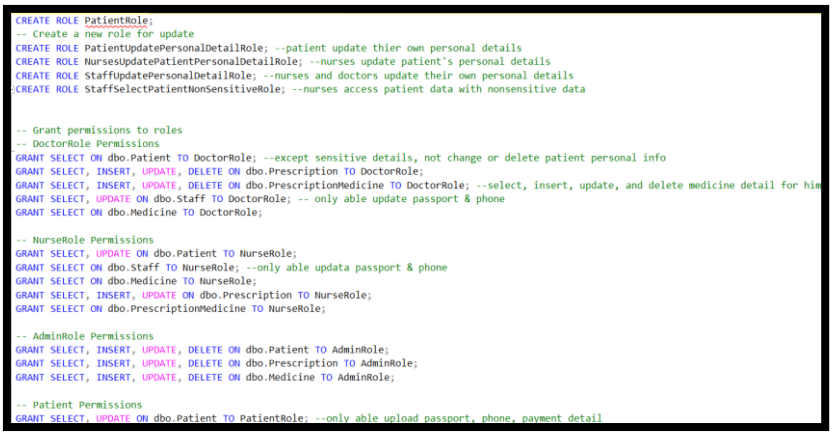
In the case of the APU Medical Center, there are three types of users created which are doctor (user001), nurse (user002), and patient (user101). They are granted the related permissions according to their roles.

## 2.3 User Management Solutions

### 2.3.1 Role-Based Access Control (RBAC)

As one of the developers of the APU Medical Centre database system, he is responsible for the design and implementation of permission management. For the management of this permission, Security and permissions which is applying authorization matrix, and SQL Injection and Unauthorized Access are adopted, all of which use role-based access control (RBAC) technique to configuration. These two plans are adopted to address several risks to the system, which are unauthorized access to the database & sensitive data, unauthorized changes to the data, lack of access control and SQL injection attack.

The first is Security and permissions. According to the elements of several roles involved in the system, when implementing data design and development, the role-users mapping function can be used to divide admin, doctor, nurses, and patients, and then grant the permissions of the roles according to different roles. Utilizing role-users mapping technology can greatly improve the security of the system. It can simplify the permission management of the system and can scale and quickly change permissions when necessary. It is easy to manage and control the configured roles and provides certain convenience in maintenance.



```
CREATE ROLE PatientRole;
-- Create a new role for update
CREATE ROLE PatientUpdatePersonalDetailRole; --patient update thier own personal details
CREATE ROLE NursesUpdatePatientPersonalDetailRole; --nurses update patient's personal details
CREATE ROLE StaffUpdatePersonalDetailRole; --nurses and doctors update their own personal details
CREATE ROLE StaffSelectPatientNonSensitiveRole; --nurses access patient data with nonsensitive data


-- Grant permissions to roles
-- DoctorRole Permissions
GRANT SELECT ON dbo.Patient TO DoctorRole; --except sensitive details, not change or delete patient personal info
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Prescription TO DoctorRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.PrescriptionMedicine TO DoctorRole; --select, insert, update, and delete medicine detail for him
GRANT SELECT, UPDATE ON dbo.Staff TO DoctorRole; -- only able update passport & phone
GRANT SELECT ON dbo.Medicine TO DoctorRole;

-- NurseRole Permissions
GRANT SELECT, UPDATE ON dbo.Patient TO NurseRole;
GRANT SELECT ON dbo.Staff TO NurseRole; --only able updata passport & phone
GRANT SELECT ON dbo.Medicine TO NurseRole;
GRANT SELECT, INSERT, UPDATE ON dbo.Prescription TO NurseRole;
GRANT SELECT ON dbo.PrescriptionMedicine TO NurseRole;

-- AdminRole Permissions
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Patient TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Prescription TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Medicine TO AdminRole;

-- Patient Permissions
GRANT SELECT, UPDATE ON dbo.Patient TO PatientRole; --only able upload passport, phone, payment detail
```

*Figure 3: Implement Role and Grant Permission*

According to Figure 3, the operation of configuring role and grant permission is implemented. The role is configured through the CREATE ROLE, SELECT, UPDATE, INSERT and DELETE keywords used in the query code, and the grant key can be used for grant permission allocation. This is also known as role-based access control (RBAC) technique. Grant operations are performed by creating a role and granting permissions to the role. In addition, other instructions SELECT, INSERT, UPDATE, or DELETE are operation instructions, which represent the authorized operation permissions of the role.

```
-- Create an Authorization Matrix table
CREATE TABLE AuthorizationMatrix (
    RoleName VARCHAR(20),
    TableName VARCHAR(50),
    PermissionType VARCHAR(50)
);
GO

-- Populate the Authorization Matrix
INSERT INTO AuthorizationMatrix (RoleName, TableName, PermissionType)
VALUES
    ('DoctorRole', 'Patient', 'SELECT'),
    ('DoctorRole', 'Prescription', 'SELECT, INSERT, UPDATE, DELETE'),
    ('DoctorRole', 'PrescriptionMedicine', 'SELECT, INSERT, UPDATE, DELETE'),
    ('DoctorRole', 'Medicine', 'SELECT'),
    ('DoctorRole', 'Staff', 'SELECT, UPDATE'),
    ('NurseRole', 'Patient', 'SELECT, UPDATE'),
    ('NurseRole', 'Staff', 'SELECT, UPDATE'),
    ('NurseRole', 'Prescription', 'SELECT'),
    ('NurseRole', 'PrescriptionMedicine', 'SELECT'),
    ('NurseRole', 'Medicine', 'SELECT'),
    ('PatientRole','Patient','SELECT,UPDATE'),
    ('PatientRole','Prescription','SELECT'),
    ('PatientRole','PrescriptionMedicine','SELECT'),
    ('PatientRole','Medicine','SELECT'),
    ('AdminRole', 'Patient', 'SELECT,INSERT,UPDATE,DELETE'),
    ('AdminRole', 'Prescription', 'SELECT,INSERT,UPDATE,DELETE'),
    ('AdminRole', 'Medicine', 'SELECT,INSERT,UPDATE,DELETE');
GO
```

*Figure 4: Implement Authorization Matrix*

The above authorization matrix is to display the permission distribution in the role, so the protection operation has been completed after completing the above RBAC. But the purpose of creating the authorization matrix here is to be able to monitor the configured authorization behaviors. This shows the current power management structure in conjunction with the role-users mapping above.

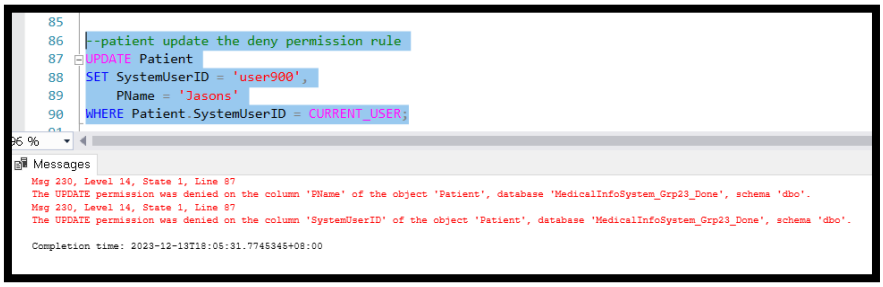



*Figure 5: Patient is not allowed to modify some personal details*

After setting up the permission control, patients are not allowed to modify or change some data like System login userID or patient name.

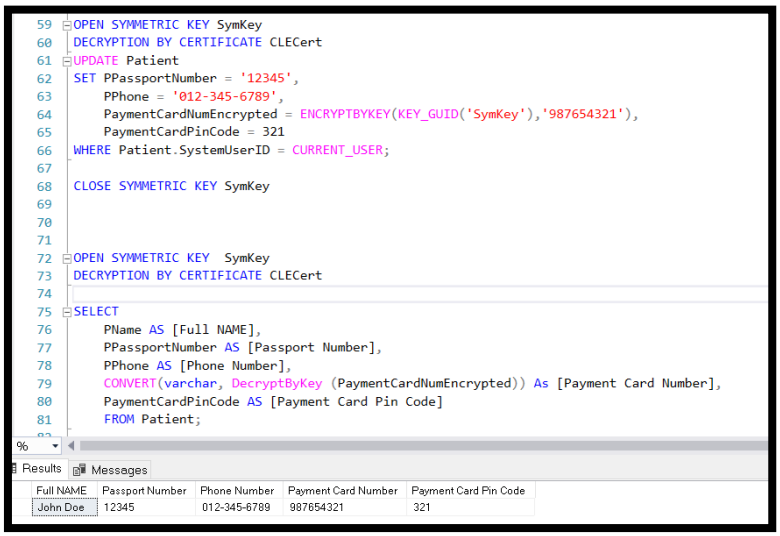



*Figure 6: Patient allow modify other personal details*

However, patients are allowed to modify other personal details like passport number, phone number and payment details.

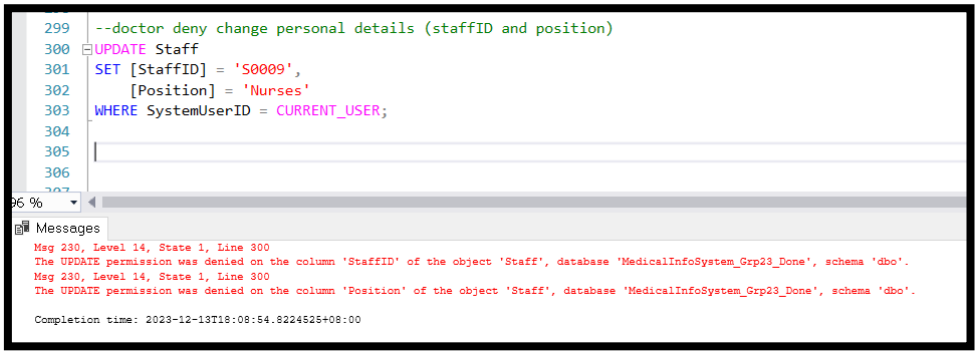



*Figure 7: Nurse or doctor is unable to change some personal details*

Figure 7 shows that doctor and nurse roles are not able to modify StaffID and position inside the database.

```
292 UPDATE Staff
293 SET [SPassportNumber] = '546943',
294     [SPhone] = '011-1352-9814'
295 WHERE SystemUserID = CURRENT_USER;
296
297 SELECT * FROM Staff;
```

Results | Messages

| StaffID | SName | SPassportNumber | SPhone | SystemUserID | Position |
|---|---|---|---|---|---|
| S001 | Dr. Koa | 546943 | 011-1352-9814 | user001 | Doctor |

*Figure 8: Doctor and nurse can update other personal information*

However, they can update their personal information like passport number and phone.

```
506 -- Doctor unable to UPDATE patient personal details
507 UPDATE Patient
508 SET
509     PName = 'John Doe',
510     PPhone = '016-45735-9134'
511 WHERE Patient.PID = 'P001';
512
513
514 --Doctor unable to DELETE patient personal details
515 DELETE FROM Patient WHERE PID = 'P001';
516
```

Messages

```
Msg 229, Level 14, State 5, Line 507
The UPDATE permission was denied on the object 'Patient', database 'MedicalInfoSystem_Grp23_Done', schema 'dbo'.
Msg 229, Level 14, State 5, Line 515
The DELETE permission was denied on the object 'Patient', database 'MedicalInfoSystem_Grp23_Done', schema 'dbo'.

Completion time: 2023-12-13T18:11:51.0528320+08:00
```

*Figure 9: Doctor unable to modify or delete patient personal details*

Based on Figure 9, doctors are only allowed to select or check patient personal details but they are unable to update or delete patient personal details or record. Only nurse role groups are allowed to modify patient names and phone numbers.

```
DECLARE @Username NVARCHAR(50);
DECLARE @Password NVARCHAR(50);

SET @Username = 'userinput';
SET @Password = 'userinput';

SELECT * FROM Users WHERE Username = @Username AND Password = @Password;


CREATE PROCEDURE dbo.AuthenticateUser
    @Username NVARCHAR(50),
    @Password NVARCHAR(50)
AS
BEGIN
    SELECT * FROM Users WHERE Username = @Username AND Password = @Password;
END
```

*Figure 10: Prevention for SQL injection*

In addition to identity permission management and allocation, it is also necessary to prevent some external attacks such as SQL Injection. There is no more effective defense based on the built-in capabilities of the database, but some means can be used to defend against SQL injection. For example, parameter query, storage operation, or the technology of least privilege

principle to defend. According to the above figure, parameterized query control is used, and the input value is converted into query mode to reduce the attacker's direct use of query commands. This technology is to separate the user-input data and SQL code in a logical form. What is used below is the PROCEDURE code. This code has been pre-compiled, so it only needs to be used to get good layered protection. Therefore, the system has an additional level of security when it obtains external resources.

```
572  -- Create the stored procedure with a table-valued parameter
573  CREATE PROCEDURE dbo.ManagePrescription
574      @PatientID NVARCHAR(6),
575      @DoctorID NVARCHAR(10),
576      @MedicineName NVARCHAR(50),
577      @Action VARCHAR(10) -- 'INSERT', 'UPDATE', 'DELETE'
578  AS
579  BEGIN
580
581
582      -- Handle different actions
583      IF @Action = 'INSERT'
584      BEGIN... -- END IF 'Insert'
667
668
669      /* SELECT STATEMETN to check own's patient medicine details*/
670      ELSE IF @Action = 'SELECT'
671      BEGIN... -- END ELSE IF 'SELECT'
698
699
700      ELSE IF @Action = 'UPDATE'
701      BEGIN... --end else if 'UPDATE'
751
752
753      /*DELETE the Medicine*/
754      ELSE IF @Action = 'DELETE'
755      BEGIN... --end else if 'DELETE'
826
827
828
829  END --end whole procedure
830
831  GO
```

*Figure 11: Procedure for allowing doctor position to manage the medicine details*

In this procedure, it is allowed to receive the user input like PatientID, DoctorID, MedicineName and Action to manage the medicinal details or record. The action includes Insert, Select, Update, and Delete.

```
836  -- Grant EXECUTE permission on the stored procedure to the user or role
837  GRANT EXECUTE ON dbo.ManagePrescription TO DoctorRole;
```

*Figure 12: Grant procedure permission to doctor role*

After designing and creating the procedure, the procedure permission is also granted to Doctor Role. It allows doctor access to this procedure.

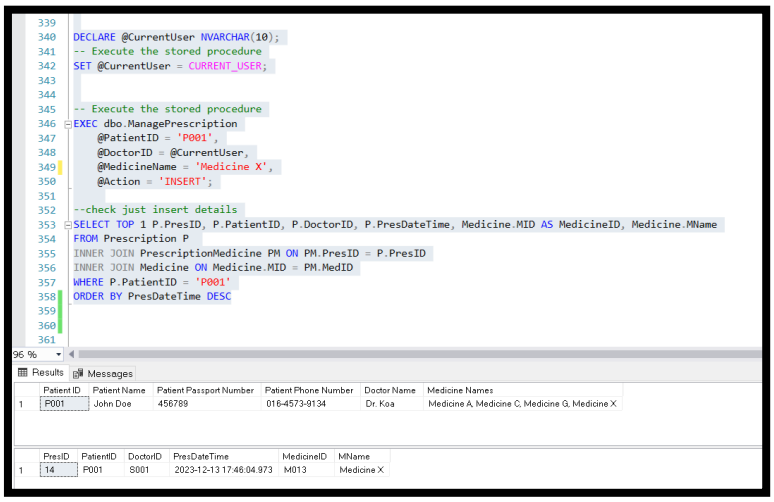



*Figure 13: Result of doctor inserting new medicine record to patient*

According to Figure 16, the doctor inserts new medicine to their patient, and "PresID = 14" are the new Prescription ID, and the Medicine name X will be inserted to the medicine table.

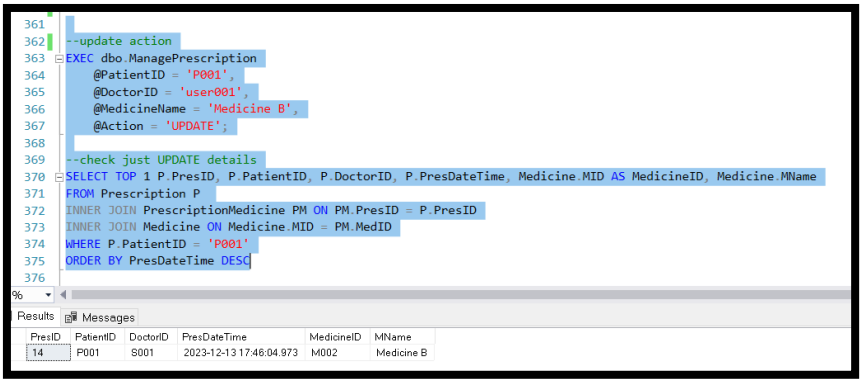



*Figure 14: Doctor update new medicine record at the moment*

If doctor found update wrong medicine record, it can directly key in new medicine name and it was updating the new medicine name inside Prescriptions Medicine table like result showing.

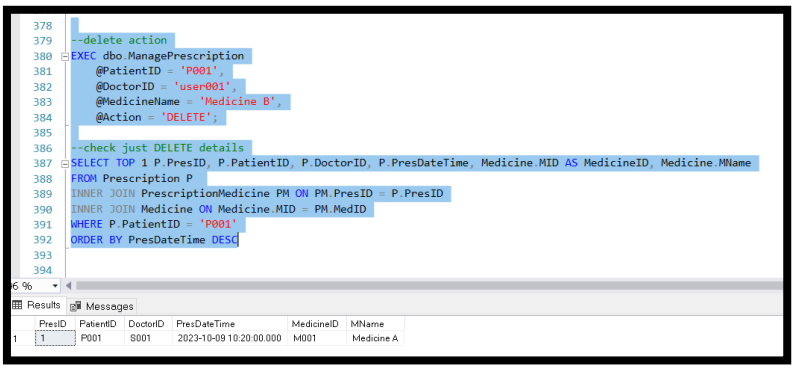
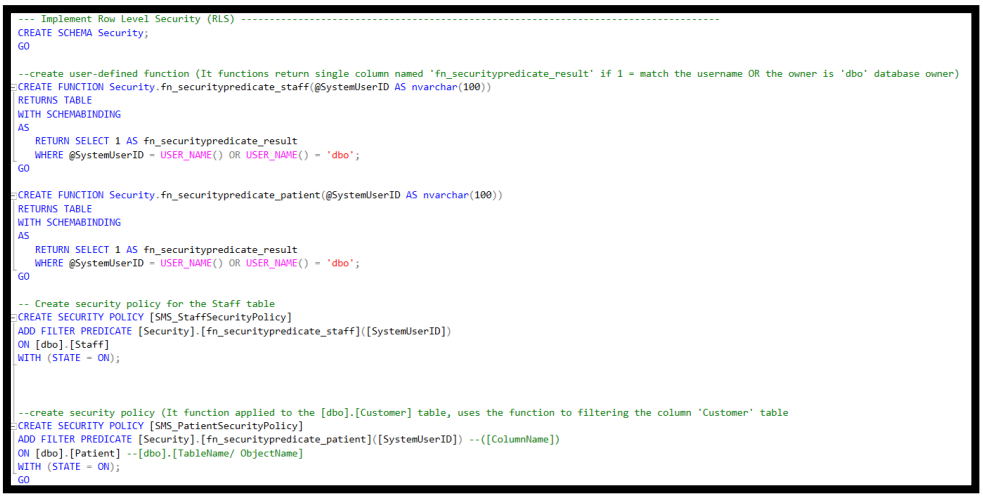



*Figure 15: Doctor deleted the medicine record*

Based on above figure showing doctor are deleted the medicine from the patient P001.

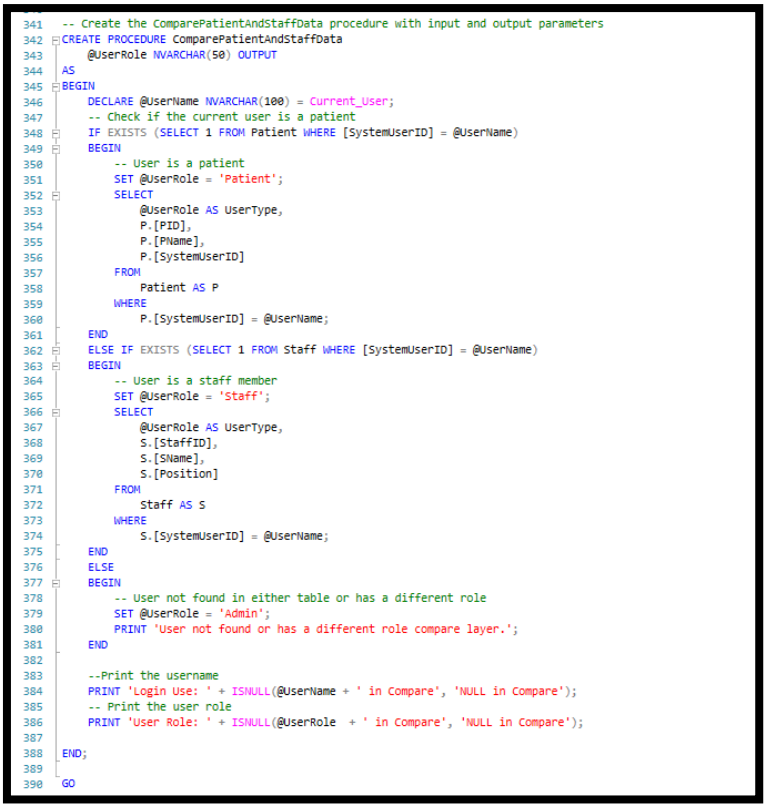
### 2.3.2 Row Level Security (RLS)



```
--- Implement Row Level Security (RLS) ------------------------------------------------------------------------
CREATE SCHEMA Security;
GO

--create user-defined function (It functions return single column named 'fn_securitypredicate_result' if 1 = match the username OR the owner is 'dbo' database owner)
CREATE FUNCTION Security.fn_securitypredicate_staff(@SystemUserID AS nvarchar(100))
RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
    WHERE @SystemUserID = USER_NAME() OR USER_NAME() = 'dbo';
GO

CREATE FUNCTION Security.fn_securitypredicate_patient(@SystemUserID AS nvarchar(100))
RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
    WHERE @SystemUserID = USER_NAME() OR USER_NAME() = 'dbo';
GO

-- Create security policy for the Staff table
CREATE SECURITY POLICY [SMS_StaffSecurityPolicy]
ADD FILTER PREDICATE [Security].[fn_securitypredicate_staff]([SystemUserID])
ON [dbo].[Staff]
WITH (STATE = ON);



--create security policy (It function applied to the [dbo].[Customer] table, uses the function to filtering the column 'Customer' table
CREATE SECURITY POLICY [SMS_PatientSecurityPolicy]
ADD FILTER PREDICATE [Security].[fn_securitypredicate_patient]([SystemUserID]) --([ColumnName])
ON [dbo].[Patient] --[dbo].[TableName/ ObjectName]
WITH (STATE = ON);
GO
```

*Figure 16: Implement RLS*

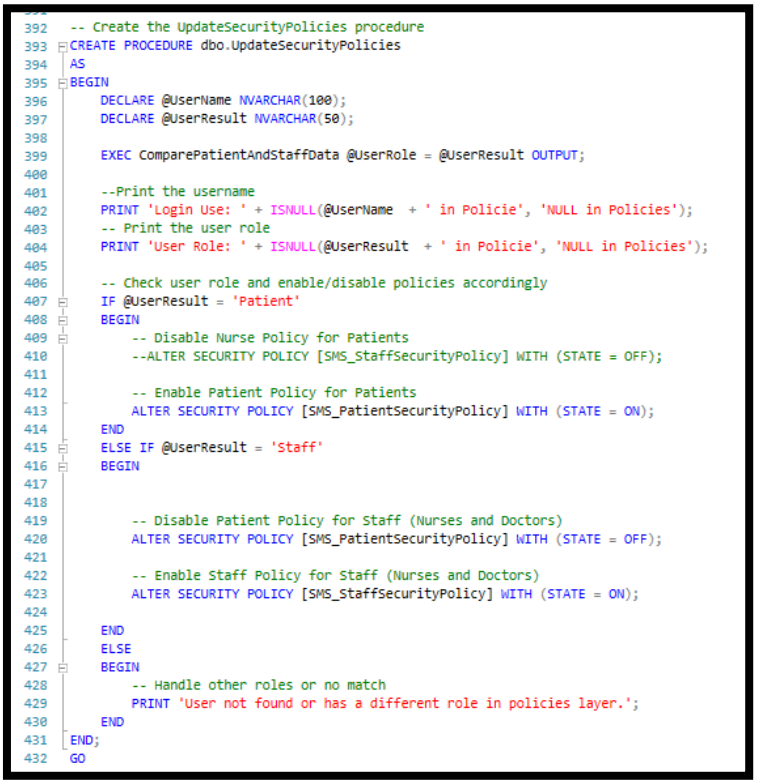
As for Row Level Security (RLS), it allows developers to restrict user access to data based on specific policies. It is usually used for fine-grained access control. It can also be said that RLS is a dynamic access control manager because it adjusts access permissions based on the user's role, attributes, and feedback behavior. This can provide users with the access they want, while also providing greater protection for sensitive data and resources. The field in POLICY represents writing a list to restrict access to a certain table. (For example, dbo.Patient / dbo.Staff) fn_securitypredicate is a function that detects whether the user has the right to access the target table. 1 means that he has the right, and 0 means that he has no right. Secondly, the user will use the identity of the created role to perform access actions. This allows you to restrict access to specific rows based on the user's role or permissions and adds security flexibility to the system.



```
-- Create the ComparePatientAndStaffData procedure with input and output parameters
CREATE PROCEDURE ComparePatientAndStaffData
    @UserRole NVARCHAR(50) OUTPUT
AS
BEGIN
    DECLARE @UserName NVARCHAR(100) = Current_User;
    -- Check if the current user is a patient
    IF EXISTS (SELECT 1 FROM Patient WHERE [SystemUserID] = @UserName)
    BEGIN
        -- User is a patient
        SET @UserRole = 'Patient';
        SELECT
            @UserRole AS UserType,
            P.[PID],
            P.[PName],
            P.[SystemUserID]
        FROM
            Patient AS P
        WHERE
            P.[SystemUserID] = @UserName;
    END
    ELSE IF EXISTS (SELECT 1 FROM Staff WHERE [SystemUserID] = @UserName)
    BEGIN
        -- User is a staff member
        SET @UserRole = 'Staff';
        SELECT
            @UserRole AS UserType,
            S.[StaffID],
            S.[SName],
            S.[Position]
        FROM
            Staff AS S
        WHERE
            S.[SystemUserID] = @UserName;
    END
    ELSE
    BEGIN
        -- User not found in either table or has a different role
        SET @UserRole = 'Admin';
        PRINT 'User not found or has a different role compare layer.';
    END

    --Print the username
    PRINT 'Login Use: ' + ISNULL(@UserName + ' in Compare', 'NULL in Compare');
    -- Print the user role
    PRINT 'User Role: ' + ISNULL(@UserRole  + ' in Compare', 'NULL in Compare');

END;

GO
```

*Figure 17: Compare the login user role group*

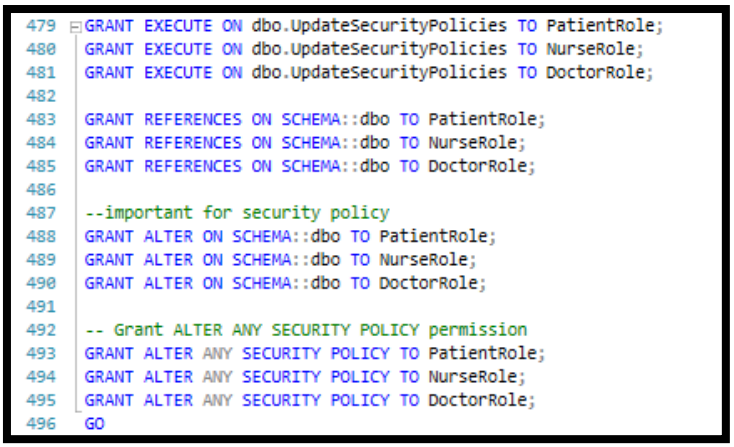
The purpose of this procedure is to identify the login user's identity and under which role group and passing the result to below UpdateSecurityPolicies procedures.



```
-- Create the UpdateSecurityPolicies procedure
CREATE PROCEDURE dbo.UpdateSecurityPolicies
AS
BEGIN
    DECLARE @UserName NVARCHAR(100);
    DECLARE @UserResult NVARCHAR(50);

    EXEC ComparePatientAndStaffData @UserRole = @UserResult OUTPUT;

    --Print the username
    PRINT 'Login Use: ' + ISNULL(@UserName  + ' in Policie', 'NULL in Policies');
    -- Print the user role
    PRINT 'User Role: ' + ISNULL(@UserResult  + ' in Policie', 'NULL in Policies');

    -- Check user role and enable/disable policies accordingly
    IF @UserResult = 'Patient'
    BEGIN
        -- Disable Nurse Policy for Patients
        --ALTER SECURITY POLICY [SMS_StaffSecurityPolicy] WITH (STATE = OFF);

        -- Enable Patient Policy for Patients
        ALTER SECURITY POLICY [SMS_PatientSecurityPolicy] WITH (STATE = ON);
    END
    ELSE IF @UserResult = 'Staff'
    BEGIN


        -- Disable Patient Policy for Staff (Nurses and Doctors)
        ALTER SECURITY POLICY [SMS_PatientSecurityPolicy] WITH (STATE = OFF);

        -- Enable Staff Policy for Staff (Nurses and Doctors)
        ALTER SECURITY POLICY [SMS_StaffSecurityPolicy] WITH (STATE = ON);

    END
    ELSE
    BEGIN
        -- Handle other roles or no match
        PRINT 'User not found or has a different role in policies layer.';
    END
END;
GO
```

*Figure 18: Based on above compared to switch the policies*

This procedure was based on comparePatientAndStaffData procedure to return result and switch on and off of the RLS policies to access the database. Like Patient login execute this procedure, it is only able to see personal record inside the Patient table nurse and doctor also. But the only difference is nurse and doctor login, they can view personal record in Staff table, also can view all the patient record.



```
GRANT EXECUTE ON dbo.UpdateSecurityPolicies TO PatientRole;
GRANT EXECUTE ON dbo.UpdateSecurityPolicies TO NurseRole;
GRANT EXECUTE ON dbo.UpdateSecurityPolicies TO DoctorRole;

GRANT REFERENCES ON SCHEMA::dbo TO PatientRole;
GRANT REFERENCES ON SCHEMA::dbo TO NurseRole;
GRANT REFERENCES ON SCHEMA::dbo TO DoctorRole;

--important for security policy
GRANT ALTER ON SCHEMA::dbo TO PatientRole;
GRANT ALTER ON SCHEMA::dbo TO NurseRole;
GRANT ALTER ON SCHEMA::dbo TO DoctorRole;

-- Grant ALTER ANY SECURITY POLICY permission
GRANT ALTER ANY SECURITY POLICY TO PatientRole;
GRANT ALTER ANY SECURITY POLICY TO NurseRole;
GRANT ALTER ANY SECURITY POLICY TO DoctorRole;
GO
```

*Figure 19: Grant permission to each role group to access procedures*

The purpose of Figure 19 is to allow each role has permission to access the given procedures.

# 3.0 Data protection

## 3.1 Threats and Importance of Data and Database Protection

In the medical context, there is a lot of sensitive data like personally identifiable information (PII), personal health information (PHI), and so on. To gain advantages from it, malicious actors always try to find any vulnerabilities to get this sensitive information. Thus, it causes many risks and threats if there is no protection for data and databases. There are some common threats to data and databases as below:

**Data Breach / Data Theft**

Data breach / Data theft is one of the potential threats, especially in the healthcare sector. It can be caused by a lot of reasons. They might be an insider who either purposefully or accidentally discloses data, credential-stealing malware, phishing, and so on (Center for Internet Security, n.d.). As a result of data breach / data theft, if the sensitive patient information falls into the wrong hands, it will cause irreparable harm to the individuals as well as the organisation's reputation.

**Data Loss**

In this digital world, most data including medical data has been moved and stored in digital form. Thus, data loss may occur in several ways like hardware failures, human error, data corruption or natural disasters. This threat may disrupt healthcare services, hinder research, and lead to financial losses (Digital Guardian, n.d.).

Data and database protection are not only mandatory to meet the requirements of compliance like HIPAA (Health Insurance Portability and Accountability Act), GDPR (General Data Protection Regulation), and so on but also to maintain trust between the organization and patients. It is important to protect patients' data and privacy no matter in transit or at rest as well as to provide high-quality services and operations continuously to the customers (cprime, n.d.). Hence, APU Medical Center needs to enhance the data protection of its database system.

## 3.2 Data Protection Solutions

For data protection, there are some common practices that can safeguard the data and database of the APU Medical Center.

### 3.2.1 Encryption


*Figure 20: Encryption (cloudflare, n.d.)*

Encryption is a protection that uses an algorithm to transform readable data into a ciphertext that is unreadable by humans. Encrypted data can only be decrypted by the key generated by the algorithm in order to retrieve usable information. With encryption, the data can only be read by the users who have the right encryption keys (N-able, 2019).

The data can be encrypted in different encryption methods and the different lengths of the encrypted key. The encryption methods can be symmetric or asymmetric. A symmetric encryption uses the same key for encryption and decryption while an asymmetric encryption uses different keys for both. On the other hand, the length of the key will affect the strength of protection. Using a longer key will make it harder to be decrypted by brute forcing if without the key. For example, the encryption, Advanced Encryption Standard 256 (AES-256), uses a 256-bits key length to encrypt and decrypt the data, and has $2^{256}$ possible key combinations. In this case, it is mostly unable to be cracked by malicious actors (Kananda, 2022).

In database security, encryption is a redundant protection when other security measures like access control work well. However, it is still essential to be implemented as it can protect data to be read if data is stolen (N-able, 2019). Thus, to prevent data breaches, leaks, theft, and so on, encryption is considered to enhance the data protection for the APU Medical Center.

### 3.2.2 Data Backup and Restore

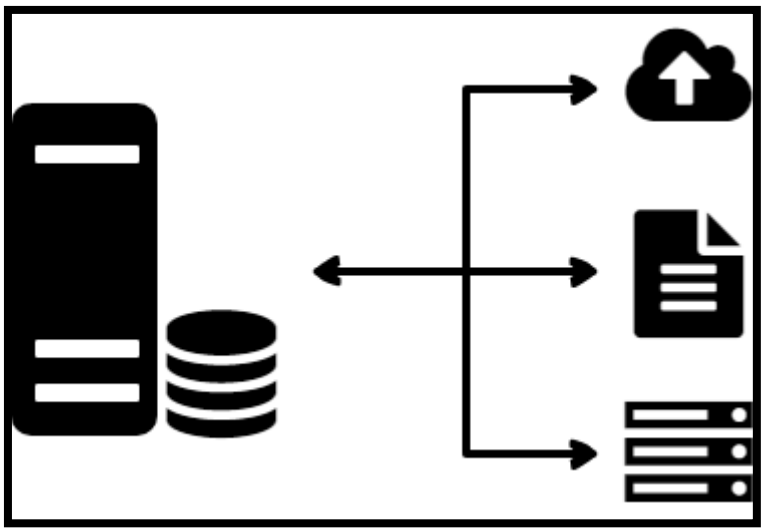


*Figure 201: Backup and Restore (Jayaram, 2018)*

Backup and recovery is a protection that creates and stores copies of data in order to prevent data loss. When a data loss occurs, the backup will be used to recover the data to the original place or other places for restoring the lost or damaged data. In general, a backup copy will be stored in a separate medium or system like tape, cloud storage, and so on to safeguard against data loss (NetApp, n.d.).

With backup and restore, the business can be recovered when its primary data fails to work. Data failure might be caused by a lot of reasons like data corruption, malicious attacks, accidental deletion, hardware, software failure, and so on. Thus, backup and restore is important to recover the affected business from an unplanned event (NetApp, n.d.).

In the scenario of the APU medical center, it is essential to implement reliable data backup and restore mechanisms to maintain operational continuity and ensure the integrity and availability of patient data. The backups of the APU Medical Center database can be the database structures, data, protections, and so on. All of the backups can reduce the effect if there is a data loss. Regular backups, both on-site and off-site, will be the best practice in case of data loss or corruption.

## 3.3 Implemented Solutions

To enhance data protection for the APU Medical Center, the following solutions have been chosen and implemented in its database:

### 3.3.1 Encryption – Transparent Data Encryption (TDE)

Transparent Data Encryption (TDE), one of the data encryption methods for databases, has been chosen for the APU Medical Center database. TDE ensures that the data is encrypted both in transit and at rest. It is important for the APU Medical Center to protect sensitive data like health records, payment card information, and so on. By encrypting data at rest, it can ensure that even if unauthorised access is gained to the database files, the data remains unreadable. It is the additional layer of security to safeguard patient privacy and ensure compliance with data protection regulations (Microsoft, 2023).

Besides the powerful protection of TDE, it is chosen as it seamlessly integrates into the database management system like SQL Server Management Studio. As the data is encrypted at the file level, the data is encrypted before they are written to disk and are decrypted when read into memory. Hence, it is transparent to applications and users which makes itself a straightforward solution for protecting data at rest (Etienne, 2022). Although there are other encryption solutions, TDE aligns well with the database environment and ensures data confidentiality for the APU Medical Center.

```
--SOLUTION 1: ENCRYPTION
--1.1 Enable Transparent Data Encryption (TDE) for the database
USE master;
GO

--Create a master key
CREATE MASTER KEY ENCRYPTION BY
PASSWORD = 'Grp23_MasterK3y';
GO

--Create certificate
CREATE CERTIFICATE Cert_APUMC
WITH SUBJECT = 'APU MC Cert'
GO

USE MedicalInfoSystem_Grp23_Done;
GO

-- Create a database encryption key (DEK)
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE Cert_APUMC;
GO

-- Enable encryption on the database
ALTER DATABASE MedicalInfoSystem_Grp23_Done
SET ENCRYPTION ON;
-- close the encryption
--ALTER DATABASE MedicalInfoSystem_Grp23_Done
--SET ENCRYPTION OFF;
GO
```

*Figure 212: TDE*

Figure 22 shows the implementation of TDE for APU Medical Center's database. Firstly, a master key is created to protect the following created certificate in the server layer. After that, a database encryption key is created and protected by the certificate. Lastly, the database is set to use encryption and protection.

### 3.3.2 Encryption – Column Level Encryption (CLE)

Another encryption method that has been implemented is Column Level Encryption (CLE). It is used for specific columns that contain highly sensitive information. In the case of APU Medical Center, the column-level encryption will protect the confidentiality and integrity of patient-sensitive information like payment card details.

CLE is a cryptographic technique that allows for the selective encryption of individual columns within a database table. Unlike TDE which encrypts the whole database, CLE enables a more targeted security strategy. Each encrypted column has its own encryption key which provides a higher level of control over access to sensitive data (Shiftan, 2023).

The implementation of CLE in the APU Medical Center focuses on the encryption of payment card numbers within the Patient table. The decision to encrypt payment card numbers individually addresses the specific security and compliance requirements related to financial and personal information. With CLE, the risk of unauthorized access and potential financial fraud will be reduced. Furthermore, regulatory requirements like the Health Insurance Portability and Accountability Act (HIPAA) and so on, which are crucial to healthcare institutions will be met. Lastly, the patient's trust will be enhanced as the confidentiality of their personal and financial details is ensured.

```
-- 1.2: Column Level Encryption for customers' sensitive data
USE MedicalInfoSystem_Grp23_Done;
GO

ALTER TABLE Patient
ADD PaymentCardNumEncrypted VARBINARY(MAX)

--Step 1 - Create Master Key
CREATE master key encryption by password = 'CLE_MasterK3y'
go
select * from sys.symmetric_keys
go

--Step 2 - Create Certificate to protect sym key
CREATE CERTIFICATE CLECert WITH SUBJECT = 'CLE Cert';
GO

--Step 3 - Create symmetric key
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE CLECert;
GO

OPEN SYMMETRIC KEY SymKey
DECRYPTION BY CERTIFICATE CLECert

UPDATE Patient
Set PaymentCardNumEncrypted =
ENCRYPTBYKEY(KEY_GUID('SymKey'),PaymentCardNumber)

CLOSE SYMMETRIC KEY SymKey
```

| | PID | PName | PaymentCardNumEncrypted |
|---|---|---|---|
| 1 | P001 | John Doe | 0x00CAEEDA3C832E4D93673279085C9F1002000000B8D5E2F... |
| 2 | P002 | Jane Smith | 0x00CAEEDA3C832E4D93673279085C9F10020000002DDE333... |
| 3 | P003 | Wade | 0x00CAEEDA3C832E4D93673279085C9F10020000008733234... |
| 4 | P004 | Dave | 0x00CAEEDA3C832E4D93673279085C9F1002000000357490E... |
| 5 | P005 | Seth | 0x00CAEEDA3C832E4D93673279085C9F1002000000DBBDBF... |
| 6 | P006 | Ivan | 0x00CAEEDA3C832E4D93673279085C9F100200000078A5C1A... |

*Figure 223: CLE with its result*

Figure 23 shows the implementation of column-level encryption. First of all, a database master key and certificate are created. The purpose of the master key and certificate are the same as the TDE which are used for protecting each and others. Then, a symmetric key is created to encrypt the column data. In the case of the APU Medical Center, the patient's payment card number is encrypted and saved in another column. Lastly, the original column which stores the unencrypted data is dropped to prevent any data breach.

### 3.3.3 Backup and Recovery

As data protection encompasses disaster recovery, backup and recovery has been chosen as the solution to mitigate data loss. Data loss is a significant threat in the healthcare sector and can result from unforeseen events like natural disasters, hardware failures, accidental deletions and so on. Thus, it is essential to implement reliable data backup and restore mechanisms to maintain the APU Medical Center's operational continuity and ensure the data integrity and availability. Furthermore, by implementing backup and restore solutions, the APU Medical Center can recover from data loss events efficiently, minimize the impact of data loss, and prevent potential operational disruptions (NetApp, n.d.).

Besides the full database backup, the creations of the APU Medical Center database which include database structure and initial data population will also be backup. With these backups, the recovery of the APU Medical Center database will be quick and efficient.

```
169  --SOLUTION 2: BACKUP AND RESTORE
170  -- Create a full database backup
171  BACKUP DATABASE MedicalInfoSystem_Grp23_Done
172  TO DISK = N'C:\APU_MC_Backup\MedicalInfoSystem_Grp23_full.bak';
173
174  -- Optionally, create transaction log backups for point-in-time recovery
175  BACKUP LOG MedicalInfoSystem_Grp23_Done
176  TO DISK = N'C:\APU_MC_Backup\MedicalInfoSystem_Grp23_log_1.bak';
177
178  -- Repeat the transaction log backup for additional log files if needed
179  -- BACKUP LOG MedicalInfoSystem TO DISK = 'C:\Backup\MedicalInfoSystem_log_2.bak';
180
181  -- To Restore the database
182  --RESTORE DATABASE MedicalInfoSystem_Grp23_Done
183  --FROM DISK = N'C:\APU_MC_Backup\MedicalInfoSystem_Grp23_full.bak'
```

*Figure 234: Backup and Recovery*

Figure 24 shows the implementation of backup and recovery. The APU Medical Center database is fully backed up and stored on a disk. Besides the backup of the database, the important keys and certificates used to implement TDE and CLE are also backed up.

# 4.0 Auditing

## 4.1 Problems and Threats

### 4.1.1 Insider Threats

The insiders within an organization that can be assumed in APU Medical Center may use the authorized access for bad intentions, thus making it a severe problem in the SQL Server insider risks (Smith, 2021). These include doctors, nurses, and privileged insiders as a single entity who can purposefully or inadvertently breach data confidentiality, integrity, and system security as a whole. Motivation for insider threats vary from espionage, economic gain, revenge, and simple curiousness (Jones et al., 2020). The insider threat life span includes phases that include recruiting, unauthorized access, information gathering, and identification and reaction by the organization (Brown & Johnson, 2019). These include credential sharing, SQL injection, unsanctioned access, and backdoor development.

To identify insider threats successfully, APU Medical Center should be on the lookout for technological and behavioral clues like irregular access times, excessive data downloads, frequent login failures, and mass data alterations action done by employees (Doe & Roe, 2022). To mitigate the problem data encryption, incident response plan, behavioral analysis, frequent access audits, and user training is important and needed (Smith & Johnson, 2020). It might become problematic to discriminate intentionally from unintentional dangers, to handle shifting insider behavior patterns, and to reconcile privacy and security needs as they relate to the insider threats' mitigations (Brown, 2021). In addition to other issues of regulatory compliance, strong audit trials and comprehensive security measures are essential, especially standards such as GDPR and HIPAA (Garcia et al., 2019). The combined efforts of the IT security, HR, legal, and management teams of APU Medical Center are necessary as a strategy to build a strong wall of defense against the different vulnerabilities that insiders present and to combat insider threats.

### 4.1.2 Unauthorized Data Modification

The security of data stored in SQL Server databases of APU Medical Center is very critical as it serves the role of data in a medical's operations which can affect life and medicine supply. Therefore, data modification by unauthorized persons is a serious concern as such modification encompasses a variety of undesirable acts aimed at altering the database contents without authorization and permission. In addition, unauthorize data modification comes from both internal database users and external sources and employ strategies like insider abuse and SQL injection attacks, challenging the security features of SQL Server databases. Unauthorized data alteration could cause data corruption, loss of confidentiality, financial losses, damage to reputation, and affect the operation of the medical center. Therefore, it is important to understand the mechanisms behind data modification due to channels like SQL injection attacks, credential compromises, and unsecured database connections to ensure the server always runs in a secure manner.

There should be comprehensive mitigation against hazards associated with unauthorized data change through proactive resolution of vulnerabilities. This involves implementing robust authentication measures, adhering to access controls, and server auditing. Proactive monitoring using SQL Server Audit allowed frequent security audits and through input validation to detect early threats and mitigation. A change management procedure is put in place to ensure that all changes to databases are made to adhere to organizational guidelines and are properly approved.

### 4.1.3 Privilege Escalation

SQL Server has major security problems called privilege escalation issues that make it hard to keep important data safe and private. By using SQL Server environment flaws, these risks allow people who are not supposed to have access obtain access to more advanced levels than they were given at first such as what patient can do. To get higher rights, people use things like SQL injection attacks, taking advantage of security holes that have not been fixed, wrongly set access limits, and bad authentication. These types of risks have very bad effects, like security holes that let people who aren't supposed to be their access secret records without any problems, putting data's privacy and trustworthiness at risk. When unauthorized people with high-level access change data, mess with processes, and do management work in a SQL Server environment, there are big risks to its overall security and stability.

To lower the risks of power escalation, servers of APU Medical Center need to be thorough and vigilant. To do this, APU Medical Center need to strengthen security methods of server with tools like multi-factor authentication, regularly update SQL Server software to fix bugs, and use the least privilege principle to make sure users only have the access they need. By using tools for real-time tracking, reporting, and database activity monitoring, businesses can quickly deal with any security risks. These tools can help find strange actions that could be signs of attempts to gain more privileges.

## 4.2 Implemented Solution with Auditing

### 4.2.1 Configuration of Server Level Audit Login and Logout Event Group

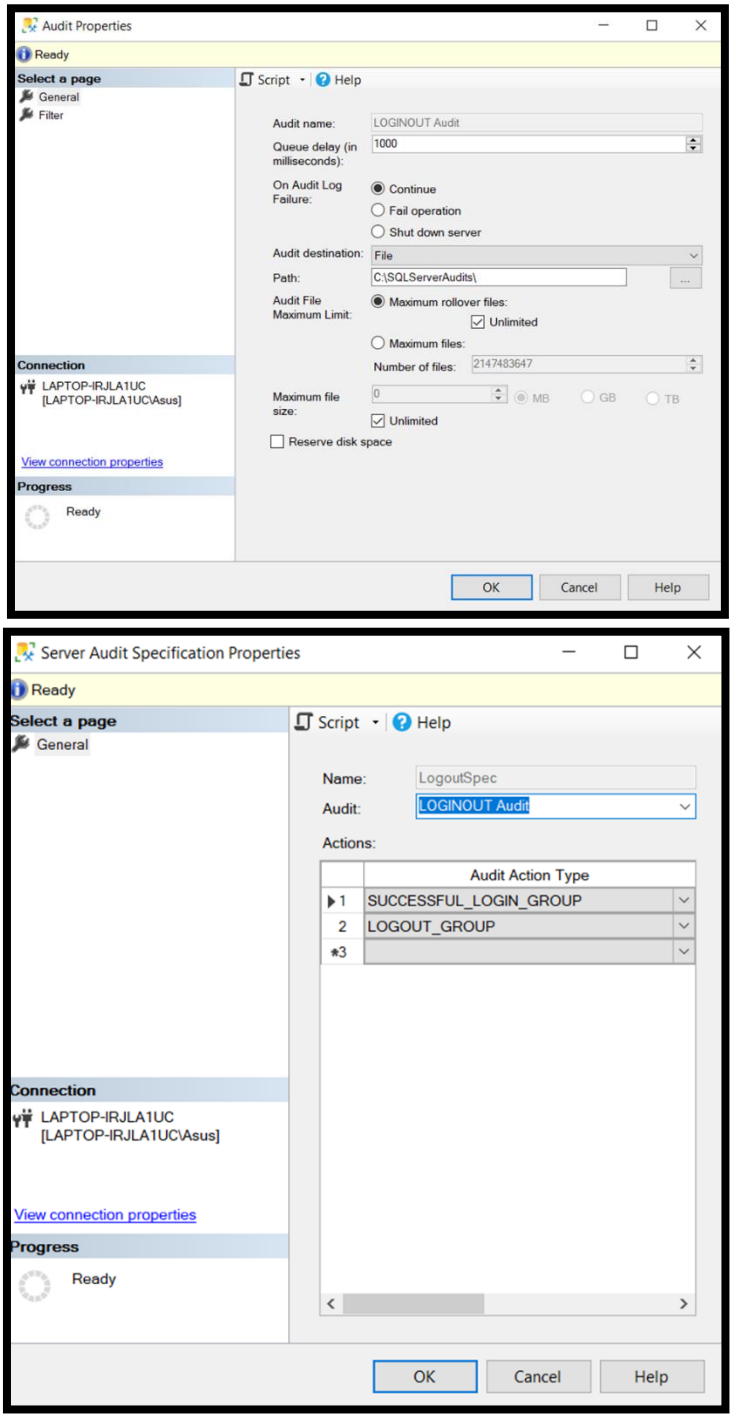



*Figure 24: Database Auditing*

Server Audit Login and Logout groups are important parts of SQL Server systems of APU Medical Center that defense against insider threats (Smith, 2021). These audit groups make it possible to keep an eye on and analyze what users such as doctors and nurses are doing which

makes the organization much safer and prevent malicious access. By keeping a close eye on both successful and failed login attempts (SUCCESSFUL_LOGIN_GROUP) and (FAILED_LOGIN_GROUP) which created as shown in the diagram above. (Brown & Johnson, 2019). The security team of APU Medical Center can quickly spot problems and possible insider illegal access, like password misuse or multiple failed login attempts.

Additionally, maintaining updated on logout events (LOGOUT_GROUP) shows how much time user sessions last and assists in detecting rapid or frequent logouts, which could mean that an insider is trying to avoid being caught and detected (Garcia, 2018). By looking at use trends and session time, security team can spot strange behavior, like long sessions or access to private data outside of normal business hours, that could be a sign of insider threats (Doe & Roe, 2022). Server Audit Login and Logout events allowed security teams to know right away if there are problems, so they can move quickly (Smith & Johnson, 2020). Brown and Johnson (2019) present that identifying users through successful login events helps hold users accountable and makes investigative studies easier during investigations.

The comprehensive auditing tools set up by the security expert for APU Medical Center also make sure they follow the regulations when it comes to monitoring and keeping track of the person or employees that has access to private information (Garcia et al., 2019). In addition, Server Audit Login and Logout groups and role-based access control (RBAC) provide a function for APU Medical Center to find and detect hidden threats and lower the risks they pose. By doing regular audits and reading over audit logs while needed, server admin of APU Medical Center can keep up a preventative security stance (Smith, 2021).

### 4.2.2 DML Audit Specification

```
-- Create Server Audit
CREATE SERVER AUDIT Grp23DatabaseAudit
TO FILE (FILEPATH = 'C:\SQLServerAudits')
WITH (ON_FAILURE = CONTINUE);
GO

-- Enable Server Audit
ALTER SERVER AUDIT Grp23DatabaseAudit WITH (STATE = ON);
GO

-- Database Audit Start
USE MedicalInfoSystem_Grp23;
GO

-- Create Database Audit DML Specification (Data Level)
CREATE DATABASE AUDIT SPECIFICATION Grp23DatabaseDMLSpec
FOR SERVER AUDIT Grp23DatabaseAudit
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::MedicalInfoSystem_Grp23 BY public),
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::MedicalInfoSystem_Grp23 BY DoctorRole),
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::MedicalInfoSystem_Grp23 BY NurseRole),
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::MedicalInfoSystem_Grp23 BY PatientRole),
ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::MedicalInfoSystem_Grp23 BY AdminRole)
WITH (STATE = ON);
```

*Figure 25: DML Audit Specification*

Create a database-level audit for Data Manipulation Language (DML) actions for APU Medical Center in the given SQL code. This is one of the most important things security experts can do in order to lower the risk of Unauthorized Data Modification in SQL Server. The Grp23DatabaseDMLSpec standard is mostly about auditing and focus on certain DML which is data manipulation actions that happened or present in MedicalInfoSystem_Grp23 database. The audit specification includes the SELECT, INSERT, UPDATE, and DELETE steps to make sure that any possible unauthorized data changes are fully covered and audited. The meaning is also made clearer by connecting these tasks to different jobs, like public, DoctorRole, NurseRole, PatientRole, and AdminRole to differentiate the user group and the action performed. This level of detail helps detect and discover illegal behavior by making it easier to connect changes to data to specific user jobs when security event detected. By adding the line WITH (STATE = ON) to this audit definition, organizations can be sure that the system constantly watches and records important DML events. This makes the system a useful tool for responding to and keeping an eye on any possible changes made to the chosen database without permission.

### 4.2.3 DDL Audit Specification

```
-- Enable Server Audit
ALTER SERVER AUDIT Grp23ServerAudit WITH (STATE = ON);
GO

-- Create a server audit specification
CREATE SERVER AUDIT SPECIFICATION ServerStateAudit
FOR SERVER AUDIT Grp23ServerAudit
ADD (SUCCESSFUL_LOGIN_GROUP),
ADD (FAILED_LOGIN_GROUP),
ADD (SERVER_PERMISSION_CHANGE_GROUP),
ADD (DATABASE_CHANGE_GROUP),
ADD (SERVER_ROLE_MEMBER_CHANGE_GROUP);

-- Enable the server audit specification
ALTER SERVER AUDIT SPECIFICATION ServerStateAudit WITH (STATE = ON);
GO
```

```
-- Create Database Level Audit with event group (DDL)
CREATE DATABASE AUDIT SPECIFICATION Grp23DatabaseDDLSpec
FOR SERVER AUDIT Grp23DatabaseAudit
ADD (DATABASE_OBJECT_CHANGE_GROUP),
ADD (DATABASE_PERMISSION_CHANGE_GROUP),
ADD (DATABASE_OBJECT_PERMISSION_CHANGE_GROUP),
ADD (SCHEMA_OBJECT_CHANGE_GROUP),
ADD (SCHEMA_OBJECT_PERMISSION_CHANGE_GROUP),
ADD (DATABASE_PRINCIPAL_CHANGE_GROUP)
WITH (STATE=ON)
Go
```

*Figure 26: DDL Audit Specification*

As the figure above shown, the snipped code shows the configuration of ServerStateAudit and Grp23DatabaseDDLSpec which are two important database level specification and is one of the important parts of lowering the risks of Privilege Escalation in SQL Server of APU Medical Center. The ServerStateAudit standard is related to the Grp23ServerAudit server audit. This specification is meant to keep track of important server-level actions like successful logins attempt to know the last login user, changes to server permissions to detect anyone who try to perform a permission escalation and all changes to databases and server role memberships. If server of APU Medical Center maintain the safeguard on these server-level actions, which show changes to job memberships, server settings, and user authentication, it may be better able to discover and denied the possible attempts at Privilege Escalation. At the same time, the Grp23DatabaseDDLSpec standard discusses the events in the database that happen because of

Data Definition Language (DDL), such as changes to database objects, rights, models, and database owners. It has been set up for the server audit Grp23DatabaseAudit. By making it easier to spot illegal changes at the database level, this detailed audit description adds an important layer of defense against Privilege Escalation attacks. By setting certain audit conditions with the line WITH (STATE=ON), the SQL code makes sure that the system constantly records and keeps track of these important events. This lets managers quickly deal with any problems or illegal behavior by checking audit logs on a regular basis.

# 5.0 Summary

In summary, this report covers all of the security enhancements for the APU Medical Center's database and focuses on addressing threats related to permission management, data protection, and auditing. The report provides a comprehensive overview of the proposed security measures and their practical implementations. Besides, it also emphasizes the importance of role-based access control, row-level security, encryption, data backup and restore, and robust auditing mechanisms. All in all, the implemented solutions aim to create a secure environment for managing patient data as well as ensure confidentiality, integrity, and availability for APU Medical Center.

# 6.0 References


Center for Internet Security. (n.d.). *Data Breaches: In the Healthcare Sector*. Retrieved from cisecurity: https://www.cisecurity.org/insights/blog/data-breaches-in-the-healthcare-sector

cloudflare. (n.d.). *What is encryption?* Retrieved from cloudflare: https://www.cloudflare.com/learning/ssl/what-is-encryption/

cprime. (n.d.). *The Importance of Healthcare Data Security*. Retrieved from cprime: https://www.cprime.com/resources/blog/the-importance-of-healthcare-data-security/

Digital Guardian. (n.d.). *HEALTHCARE*. Retrieved from Digital Guardian: https://www.digitalguardian.com/solutions/healthcare

ECCU. (n.d.). *The Importance of Data Security in Electronic Health Records*. Retrieved from ECCU: https://www.eccu.edu/blog/cybersecurity/the-importance-of-data-security-in-electronic-health-records/

Etienne. (2022, August 19). *Transparent Data Encryption (TDE)*. Retrieved from SQL Land: https://sqland.wordpress.com/2022/08/19/transparent-data-encryption-tde/

Jayaram, P. (2018, March 1). *An overview of the process of SQL Server backup-and-restore*. Retrieved from SQLShack: https://www.sqlshack.com/overview-sql-server-backup-restore-process/

Kananda, V. (2022, June 22). *Why You Should Use AES 256 Encryption to Secure Your Data*. Retrieved from Progress: https://www.ipswitch.com/blog/use-aes-256-encryption-secure-data

Microsoft. (2023, August 30). *Transparent data encryption (TDE)*. Retrieved from Microsoft: https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver16

N-able. (2019, May 10). *Types of Database Encryption Methods*. Retrieved from N-able: https://www.n-able.com/blog/types-database-encryption-methods

NetApp. (n.d.). *What Is Backup and Recovery?* Retrieved from NetApp: https://www.netapp.com/cyber-resilience/data-protection/data-backup-recovery/what-is-backup-recovery/

Shiftan, A. (2023, June 20). *Column-Level Encryption 101: What is It, implementation & Benefits*. Retrieved from PiiANO: https://www.piiano.com/blog/column-level-encryption