**A logo of a globe with yellow rings around it

Description automatically generated**

**CT069-3-3-DBS**

**DATABASE SECURITY**

**GROUP ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| INTAKE CODE | : | APD3F2408CS(DA) |
| LECTURER NAME | : | DR. KULOTHUNKAN A/L PALASUNDRAM |
| HAND OUT DATE | : | 12th FEBRUARY 2025 |
| HAND IN DATE | : | 04th MAY 2025 |

|  |  |  |
| --- | --- | --- |
| **GROUP 18** | | |
| **NAME** |  | **TP NUMBER** |
| MUHAMMAD NABIL HAKIM BIN YUSAIDI |  | TP066763 |
| SOO JIUN GUAN |  | TP068687 |
| TAN SHI YING |  | TP065227 |
| TEH YUE FENG |  | TP069429 |

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc197291044)

[1 INTRODUCTION 8](#_Toc197291045)

[1.1 CLASSIFICATION MATRIX 9](#_Toc197291046)

[1.2 AUTHORIZATION MATRIX 11](#_Toc197291047)

[1.3 USER MATRIX 12](#_Toc197291048)

[1.4 AUDIT MATRIX 13](#_Toc197291049)

[2 DATA PROTECTION 16](#_Toc197291050)

[2.1 STUDENT 1 – MUHAMMAD NABIL HAKIM BIN YUSAIDI TP066763 16](#_Toc197291051)

[2.1.1 ISSUE 1 – Appointment Duplication 16](#_Toc197291052)

[2.1.1.1 PROOF 16](#_Toc197291053)

[2.1.1.2 SOLUTION – Use Trigger and Stored Procedure for Appointments Duplication 17](#_Toc197291054)

[2.1.1.3 PROOF OF SOLUTION 17](#_Toc197291055)

[2.1.2 ISSUE 2 – Lack of Appointment Status Management 18](#_Toc197291056)

[2.1.2.1 PROOF 18](#_Toc197291057)

[2.1.2.2 SOLUTION – Use Trigger and Procedure to Manage Appointment Status 18](#_Toc197291058)

[2.1.2.3 PROOF OF SOLUTION 19](#_Toc197291059)

[2.1.3 ISSUE 3 – Unencrypted Data Storage and Transmission 20](#_Toc197291060)

[2.1.3.1 PROOF 20](#_Toc197291061)

[2.1.3.2 SOLUTION – Use TDE and Encryption Key on The Database 21](#_Toc197291062)

[2.1.3.3 PROOF OF SOLUTION 22](#_Toc197291063)

[2.2 STUDENT 2 – SOO JIUN GUAN TP068687 23](#_Toc197291064)

[2.2.1 ISSUE 1 - Exposed Patient Phone Number 23](#_Toc197291065)

[2.2.1.1 PROOF 23](#_Toc197291066)

[2.2.1.2 SOLUTION - Apply dynamic data masking to patient phone numbers with role-based unmasking 23](#_Toc197291067)

[2.2.1.3 PROOF OF SOLUTION 24](#_Toc197291068)

[2.3 STUDENT 3 – TAN SHI YING TP065227 25](#_Toc197291069)

[2.3.1 ISSUE 1- SQL Injection Threat 25](#_Toc197291070)

[2.3.1.1 PROOF 25](#_Toc197291071)

[2.3.1.2 SOLUTION - Parameterized Queries (Stored Procedures), and Input Validation 26](#_Toc197291072)

[2.3.1.3 PROOF OF SOLUTION 27](#_Toc197291073)

[2.3.2 ISSUE 2 - Sensitive Data Exposure (Passport data for patient and staff) 27](#_Toc197291074)

[2.3.2.1 PROOF 28](#_Toc197291075)

[2.3.2.2 SOLUTION - Asymmetric Encryption 28](#_Toc197291076)

[2.3.2.3 PROOF OF SOLUTION 29](#_Toc197291077)

[2.4 STUDENT 4 – TEH YUE FENG TP069429 30](#_Toc197291078)

[2.4.1 ISSUE 1 – Backdated Prescription Entries are Allowed 30](#_Toc197291079)

[2.4.1.1 PROOF 30](#_Toc197291080)

[2.4.1.2 SOLUTION - Prevent insertion of backdated prescriptions using a trigger 31](#_Toc197291081)

[2.4.1.3 PROOF OF SOLUTION 32](#_Toc197291082)

[2.4.2 ISSUE 2 – Plaintext Storage of Payment Information 32](#_Toc197291083)

[2.4.2.1 PROOF 33](#_Toc197291084)

[2.4.2.2 SOLUTION - Data Hashing for CardNumber and CardPinCode 33](#_Toc197291085)

[2.4.2.3 PROOF OF SOLUTION 34](#_Toc197291086)

[2.4.3 ISSUE 2 – No Backup or Disaster Recovery Plan 34](#_Toc197291087)

[2.4.3.1 PROOF 35](#_Toc197291088)

[2.4.3.2 SOLUTION - Implement scheduled backups and recovery processes 35](#_Toc197291089)

[2.4.3.3 PROOF OF SOLUTION 39](#_Toc197291090)

[3 PERMISSION MANAGEMENT 41](#_Toc197291091)

[3.1 STUDENT 1 – MUHAMMAD NABIL HAKIM TP066763 41](#_Toc197291092)

[3.3.1 ISSUE 1 – Lack of Role-Based Access Control 41](#_Toc197291093)

[3.3.1.1 PROOF 41](#_Toc197291094)

[3.3.1.2 SOLUTION – Implement RBAC with Defined Roles 42](#_Toc197291095)

[3.3.1.3 PROOF OF SOLUTION 42](#_Toc197291096)

[3.2 STUDENT 2 – SOO JIUN GUAN TP068687 44](#_Toc197291097)

[3.2.1 ISSUE 1 - Unauthorized Staff Position Changes 44](#_Toc197291098)

[3.2.1.1 PROOF 44](#_Toc197291099)

[3.2.1.2 SOLUTION - Block staff from changing their job position unless they are Admin 45](#_Toc197291100)

[3.2.1.3 PROOF OF SOLUTION 45](#_Toc197291101)

[3.2.2 ISSUE 2 – No Row-Level Access Control for Staff and Patient Records 47](#_Toc197291102)

[3.2.2.1 PROOF 47](#_Toc197291103)

[3.2.2.2 SOLUTION - Enforce row-level security to restrict staff and patient access 48](#_Toc197291104)

[3.2.2.3 PROOF OF SOLUTION 49](#_Toc197291105)

[3.2.3 ISSUE 3 - Unrestricted Modification of Prescription Data by Pharmacists 50](#_Toc197291106)

[3.2.3.1 PROOF 50](#_Toc197291107)

[3.2.3.2 SOLUTION - Restrict prescription status modification via stored procedure for pharmacists 51](#_Toc197291108)

[3.2.3.3 PROOF OF SOLUTION 52](#_Toc197291109)

[3.3 STUDENT 3 – TAN SHI YING TP065227 53](#_Toc197291110)

[3.3.1 ISSUE 1 - Unauthorized Appointment Access and Updates 53](#_Toc197291111)

[3.3.1.1 PROOF 53](#_Toc197291112)

[3.3.1.2 SOLUTION - Centralized Access Control via “ManageAppointment” Procedure 54](#_Toc197291113)

[3.3.1.3 PROOF OF SOLUTION 56](#_Toc197291114)

[3.3.2 ISSUE 2 - Unauthorized Prescription Modification by Doctors 57](#_Toc197291115)

[3.3.2.1 PROOF 57](#_Toc197291116)

[3.3.2.2 SOLUTION - Stored Procedure “ManagePrescription” with Ownership Control 58](#_Toc197291117)

[3.3.2.3 PROOF OF SOLUTION 60](#_Toc197291118)

[3.4 STUDENT 4 – TEH YUE FENG TP069429 61](#_Toc197291119)

[3.4.1 ISSUE 1 - Improper Permission Control for Nurse 61](#_Toc197291120)

[3.4.1.1 PROOF 61](#_Toc197291121)

[3.4.1.2 SOLUTION - Restrict access with grants. 62](#_Toc197291122)

[3.4.1.3 PROOF OF SOLUTION 63](#_Toc197291123)

[4 AUDITING 64](#_Toc197291124)

[4.1 STUDENT 1 – MUHAMMAD NABIL HAKIM TP066763 64](#_Toc197291125)

[4.1.1 ISSUE 1 – Lack of Audit Trail for Critical Operations 64](#_Toc197291126)

[4.1.1.1 PROOF 64](#_Toc197291127)

[4.1.1.2 SOLUTION – Use SQL Server Audit for DML Audit 65](#_Toc197291128)

[4.1.1.3 PROOF OF SOLUTION 66](#_Toc197291129)

[4.2 STUDENT 2 – SOO JIUN GUAN TP068687 67](#_Toc197291130)

[4.2.1 ISSUE 1 – No Audit Trail for Patient and Staff Table Changes 67](#_Toc197291131)

[4.2.1.1 PROOF 67](#_Toc197291132)

[4.2.1.2 SOLUTION - Enable CDC (Change Data Capture) for auditing updates to Patient and Staff tables 68](#_Toc197291133)

[4.2.1.3 PROOF OF SOLUTION 69](#_Toc197291134)

[4.2.2 ISSUE 2 - No Login Activity Is Audited or Tracked 70](#_Toc197291135)

[4.2.2.1 PROOF 70](#_Toc197291136)

[4.2.2.2 SOLUTION - Implement server-level logon auditing with a trigger and store logs in a custom table 71](#_Toc197291137)

[4.2.2.3 PROOF OF SOLUTION 72](#_Toc197291138)

[4.3 STUDENT 3 – TAN SHI YING TP065227 74](#_Toc197291139)

[4.3.1 ISSUE 1 – Staff Data can be Deleted Without Trace 74](#_Toc197291140)

[4.3.1.1 PROOF 74](#_Toc197291141)

[4.3.1.2 SOLUTION – SQL Server Data Definition Language (DDL) Auditing 75](#_Toc197291142)

[4.3.1.3 PROOF OF SOLUTION 76](#_Toc197291143)

[4.4 STUDENT 4 – TEH YUE FENG TP069429 77](#_Toc197291144)

[4.4.1 ISSUE 1 - Lack of Change History Tracking on Patient, Medicine, and Staff tables. 77](#_Toc197291145)

[4.4.1.1 PROOF 78](#_Toc197291146)

[4.4.1.2 SOLUTION - Implement system-versioned temporal tables to track all changes without altering schema logic. 78](#_Toc197291147)

[4.4.1.3 PROOF OF SOLUTION 79](#_Toc197291148)

[4.4.2 ISSUE 2 - No Prescription/Appointment Audit 80](#_Toc197291149)

[4.4.2.1 PROOF 80](#_Toc197291150)

[4.4.2.2 SOLUTION - Implement triggers to record inserts, updates, and deletes. 80](#_Toc197291151)

[4.4.2.3 PROOF OF SOLUTION 83](#_Toc197291152)

[5 CONCLUSIONS 84](#_Toc197291153)

[6 REFERENCES 85](#_Toc197291154)

[APPENDICES 88](#_Toc197291155)

[WORKLOAD MATRIX 88](#_Toc197291156)

# INTRODUCTION

Protecting medical information is paramount in today’s digital health environment. With increasing risk of compromising sensitive health details, medical institutions must ensure their database systems are not only well-protected from internal misuses but also from external compromises. Recently, APU Hospital, being a reputable medical institution in Bukit Jalil, Kuala Lumpur, implemented a centralized database system called the MedicalInfoSystem. The system handles important details like employees' records, patients' data, appointments, prescriptions, and medication stock. While operational, the existing configuration is not equipped with important protective measures for the preservation of data privacies, control of access, and traceability for activities, exposing the system to risks of compromises as well as system failures.

The purpose of this project was to assess the security stance of the MedicalInfoSystem and put in place a set of enhancements strengthening the overall protection. All the database schema, user credentials, and procedural code were examined in detail in order to detect weaknesses that could jeopardize the system’s confidentiality, integrity, and availability. All the vulnerabilities found were countered using SQL Server’s in-built security features as well as custom implementations that fit in with the hospital’s business practices. The focus lay in establishing well-defined boundaries of access, limiting exposure of sensitive information, and being able to audit and trace all key actions with complete accountability.

Security features added in this project are classified under three main categories: permission management, auditing, and data protection. A total of twenty-two issues of security were found and resolved systematically under these three areas. These spanned from SQL injection weaknesses and lack of control over sensitive personal information, through incorrect assignment of privilege, unnoticed deletions, as well as the lack of backup procedures for the data. Each of the issues is backed by clear proof, accompanied by an elaborated solution and test steps that confirm the efficacy of the applied measures.

## 1.1 CLASSIFICATION MATRIX

|  |  |  |
| --- | --- | --- |
| Table Name | Variables Name | Classification |
| Staff | StaffID | Internal |
| SName | Confidential |
| SPassportNumber | Private |
| SPhone | Confidential |
| Position | Public |
| Patient | PID | Internal |
| PName | Confidential |
| PPassportNumber | Private |
| PPhone | Confidential |
| PaymentCardNumber | Restricted |
| PaymentCardPinCode | Restricted |
| Prescription | PresID | Internal |
| PatientID | Confidential |
| DoctorID | Internal |
| PresDateTime | Confidential |
| Status | Internal |
| Medicine | MedID | Internal |
| MedName | Public |
| PrescriptionMedicine | PresID | Internal |
| MedID | Internal |
| PrimaryKey | Internal |
| Appointment | AppointmentID | Internal |
| StaffID | Internal |
| PID | Confidential |
| Date | Confidential |
| Status | Internal |

Table 1: Classification Matrix

## 1.2 AUTHORIZATION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Permission Type | Object | Privilege(s) |
| Doctor | Grant | Table: Patient,  Medicine,  Appointment | Select |
| Table: Prescription | Select, Insert, Update |
| Table: PrescriptionMedicine | Select, Insert |
| Staff | Select, Update |
| Pharmacist | Grant | Table: Patient | Select |
| Table: Prescription,  Staff | Select, Update |
| Table: Medicine | Select, Insert, Update, Delete |
| Table: PrescriptionMedicine | Select, Insert, Update |
| Deny | Table: Prescription | Update |
| Nurse | Grant | Table: Patient | Select |
| Table: Appointment | Select, Insert, Update |
| Staff | Select, Update |
| Patient | Grant | Table: Prescription,  Appointment | Select |
| Patient | Select, Update |
| Deny | Table: Appointment | Update |
| Admin | Grant | MedicalInfoSystem | Full Control |
| Table: AuditLog,  LogonAudit,  cdc.dbo\_Patient\_CT, cdc.dbo\_Staff\_CT, | Select |
| Asymmetric Key,  Symmetric Key,  Certificate | Access |
| All User | Deny | Column: Patient  (CardNumber, CardPinCode, PassportNumber)  Column: Staff (PassportNumber) | Select |

Table 2: Authorization Matrix

## 1.3 USER MATRIX

|  |  |
| --- | --- |
| Roles | Users |
| Doctors | ST001, ST004 |
| Nurses | ST002, ST006 |
| Pharmacist | ST003, ST005 |
| Patients | PT001, PT002, PT003, PT004, PT005, PT006 |
| Admin | AD001, AD002 |

Table 3: User Matrix

## AUDIT MATRIX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL Object Name | SQL Object Type | What to Capture | Mechanism | Audit Log Location |
| Patient | Table | All changes (PID, PName, PPassportNumber, PPhone, PaymentCardNumber, PaymentCardPinCode) with timestamps | Temporal Table (SYSTEM\_VERSIONING = ON) | dbo.PatientHistory |
| Staff | Table | All changes (StaffID, SName, SPassportNumber, SPhone, Position) with timestamps | Temporal Table (SYSTEM\_VERSIONING = ON) | dbo.StaffHistory |
| Medicine | Table | All changes (MedID, MedName, etc.) with timestamps | Temporal Table (SYSTEM\_VERSIONING = ON) | dbo.MedicineHistory |
| Prescription | Table | INSERT: new record, UPDATE: new values, DELETE: deleted data | AFTER Trigger trg\_Prescription\_Audit | dbo.AuditLog (JSON column) |
| Appointment | Table | INSERT, UPDATE, DELETE with ApptID, PID, StaffID, Date, Status | AFTER Trigger trg\_Appointment\_Audit | dbo.AuditLog (JSON column) |
| DDL Changes | Database/Server | CREATE, ALTER, DROP on tables, views, procedures, etc. | Server Audit + Database Audit Specification | File: D:\Temp\AuditLogs |
| Patient (CDC) | Table | INSERT, UPDATE, DELETE (raw changes for sensitive data) | Change Data Capture (CDC) | cdc.dbo\_Patient\_CT, vw\_Patient\_Changes |
| Staff (CDC) | Table | INSERT, UPDATE, DELETE (raw changes for sensitive data) | Change Data Capture (CDC) | cdc.dbo\_Staff\_CT, vw\_Staff\_Changes |
| LogonAudit\_HospitalSystem | Server Trigger | User login attempts, login name, host machine, app, and timestamp | SERVER-LEVEL LOGON TRIGGER | dbo.LogonAudit |
| DML Changes | Database Audit Specification | Deleted Data details | Procedure: usp\_DeleteStaff | File: D:\AuditLogs |

Table 4: Audit Matrix

# 2 DATA PROTECTION

## 2.1 STUDENT 1 – MUHAMMAD NABIL HAKIM BIN YUSAIDI TP066763

### 2.1.1 ISSUE 1 – Appointment Duplication

In the current database, doctors can create multiple conflicting appointments for the same patient at the same or overlapping times. This will lead to scheduling conflicts and potential patient care delays. Without proper validation controls, both application-level functions and direct SQL inserts can create these conflicts, compromising the reliability of APU Hospital's appointment system.

#### 2.1.1.1 PROOF

A close up of a number

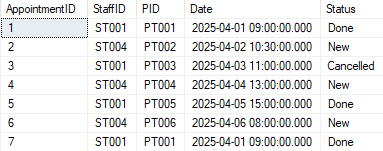
AI-generated content may be incorrect.

Figure 1: Appointment Duplication SQL Query and Results

Based on Figure above, PT001 can be seen at identical and overlapping times and are successfully inserted into the Appointment table without validation. The example INSERT statement can be executed repeatedly which creates multiple conflicting appointments. This confirms the vulnerability that it directly impacts on patients’ schedules, creates confusion among medical staffs and potentially disrupts the hospital's daily operations.

#### 2.1.1.2 SOLUTION – Use Trigger and Stored Procedure for Appointments Duplication

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: Triggers and Procedures to Avoid Duplication

After implementing the solution, the combines INSTEAD OF trigger with a stored procedure to create a dual-layer protection system. The trigger intercepts all appointment insertions, verifying them against a 30-minute overlap window and rejecting conflicts, while the stored procedure provides user-friendly validation messages. This approach maintains scheduling integrity through both database and application-level safeguards while preserving straightforward appointment management for legitimate hospital operations.

#### 2.1.1.3 PROOF OF SOLUTION

A screen shot of a computer

AI-generated content may be incorrect.

Figure 3: Appointment Successfully scheduled

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Appointment failed to scheduled

The testing confirms the solution's effectiveness through systematic validation of multiple scenarios. When executing the schedule appointments stored procedure, the first appointment succeeds as expected, while the subsequent attempt fails with error. Further testing verifies that non-conflicting appointments succeed, and that the system effectively prevents scheduling conflicts while allowing legitimate appointments to proceed normally.

### 2.1.2 ISSUE 2 – Lack of Appointment Status Management

The current database system lacks proper status transition controls for appointments, and it creates significant security and integrity vulnerabilities that could lead to scheduling conflicts, inappropriate cancellations, and potential disruptions to patient care continuity.

#### 2.1.2.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

Figure 5 Appointment Status Update

The medical database allowed inappropriate appointment status changes without validation checks, as shown in Figure 5. This vulnerability exceeded the bound ability of users to modify appointment states improperly thus creating the threat of unauthorized cancellations and disrupted patient care. The risk includes appointment workflow manipulation that could compromise scheduling integrity and patient treatment continuity.

#### 2.1.2.2 SOLUTION – Use Trigger and Procedure to Manage Appointment Status

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect.

Figure 6 Implement Triggers and Procedures for Appointment Management

The implemented solution introduced a trigger and a stored procedure shown in Figure 6 to enforce proper status workflow transitions. This approach enhances the Confidentiality-Integrity-Availability triad by primarily strengthening data integrity through status validation, preventing inappropriate changes and establishing a default "New" status for all appointments.

#### 2.1.2.3 PROOF OF SOLUTION



Figure 7 Default Status is New and Successful Cancellation

After implementing the solutions, new appointments received the default "New" status correctly, and legitimate cancellations succeeded when appointments were in the appropriate state. The system now properly validates status transitions, preventing unauthorized modifications while maintaining necessary operational flexibility for medical staff managing genuine scheduling changes.

### 2.1.3 ISSUE 3 – Unencrypted Data Storage

Data encryption is a fundamental security requirement for healthcare systems that process sensitive patient information. The current database system lacks encryption measures for both data storage and it creates significant security and privacy vulnerabilities that could lead to unauthorized data access and potential breaches of patient confidentiality.

#### 2.1.3.1 PROOF

A table with numbers and a phone payment

AI-generated content may be incorrect.

Figure 8: Unencrypted Data View of Patient Table

Based on figure above, the current database stores sensitive patient data in plaintext format which reveals unencrypted personal and payment information including passport numbers, phone numbers, payment card details, and PIN codes. This severe vulnerability exposes patient data to unauthorized access through database file theft, backup compromise, or network interception during transmission. Without encryption, the hospital fails to comply with healthcare data protection regulations and puts patient privacy at significant risk of exposure through various attack vectors.

#### 2.1.3.2 SOLUTION – Use TDE and Encryption Key on The Database

A computer code on a white background

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

Figure 9: SQL Query for Encrypting the Database using TDE and Encryption Key

The solution uses an encryption strategy using Transparent Data Encryption (TDE) with subsequent certificate and database encryption key creation. The solution includes proper key management with certificate backups and verification procedures that protects sensitive medical data while maintaining system functionality for authorized users. This implementation enhances Confidentiality through encryption at database level maintains Integrity by preventing unauthorized modification of encrypted data ensures Availability through transparent encryption/decryption processes for authenticated users and includes proper disaster recovery through certificate/private key backups using 'BACKUP CERTIFICATE' command with secure password protection.

#### 2.1.3.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: Verification of the Encryption

After applying the solution, the database files are verified as encrypted on disk. The encryption status query returns “ENCRYPTED” status. Furthermore, authorized users can still perform normal operations like SELECT \* FROM Patient with no functional changes further demonstrating that the solution effectively secures patient data without disrupting hospital operations. Certificate verification confirms proper key infrastructure setup. Authorized user access testing demonstrates that legitimate users maintain normal data access privileges through proper authentication channels yet unauthorized access attempts are effectively blocked. This solution enhances Confidentiality through comprehensive encryption without sacrificing Integrity of database operations or Availability for legitimate hospital staff.

## 2.2 STUDENT 2 – SOO JIUN GUAN TP068687

### 2.2.1 ISSUE 1 - Exposed Patient Phone Number

Patient phone numbers are stored in plaintext and accessible to all users without role-based restrictions, breaking access boundaries and exposing sensitive contact information. This broad visibility increases the threat surface, enabling users without legitimate need to access personal identifiers. The lack of visibility controls weakens containment of data within authorized scopes, raising the risk of identity misuse, unsolicited contact, and internal or external exploitation.

#### 2.2.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11: patient's phone number exposure (data protection)

The figure above reveals patient phone numbers in their entirety without role-based filtering and thus emphasizes that data masking is required to keep sensitive information confidential.

#### 2.2.1.2 SOLUTION - Apply dynamic data masking to patient phone numbers with role-based unmasking

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 12: Dynamic masking

In order to minimize this, Dynamic Data Masking (DDM) was implemented against the PPhone column with the partial(0, "XXX-XXX", 2) function, masking all but the last two digits. This is done while keeping the original data and obscuring the sensitive portions during query execution based on user roles (Microsoft, n.d.). Unlike encryption, DDM is lightweight, schema-change free, and is enforced by the SQL Server engine. Only roles with UNMASK permission, e.g., Patient and Nurse, can see unmasked values; others see masked values. This guarantees Confidentiality by limiting access, maintains Integrity by keeping original values, and enables Availability by supporting operations normally in a secure environment.

#### 2.2.1.3 PROOF OF SOLUTION

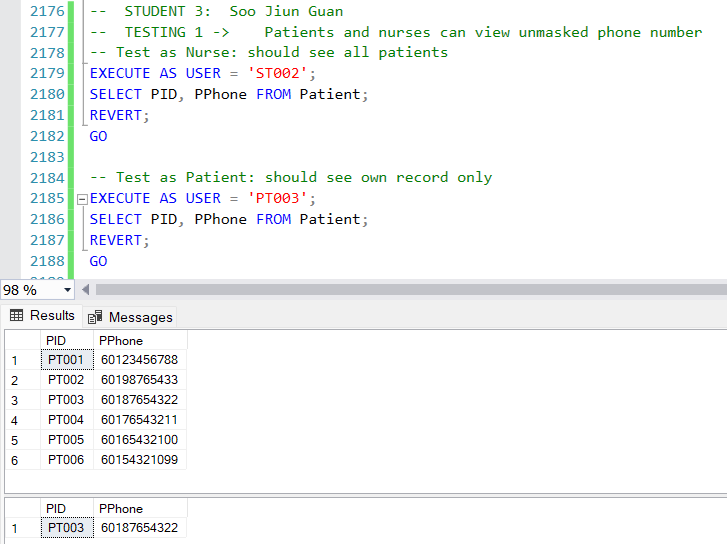


Figure 13: Roles with UNMASK permission checking the masking values

As demonstrated in the above figure, nurses (e.g., ST002) are able to see all patients’ complete unmasked phone numbers, whereas patients (e.g., PT003) are able to see their unmasked phone number, validating that data masking with role-based access has addressed the problem.

## 2.3 STUDENT 3 – TAN SHI YING TP065227

### 2.3.1 ISSUE 1- SQL Injection Threat

The dynamic SQL without input validation in the database is causing vulnerability for attackers to inject malicious code (Hussein, Ibrahim, & Alsalamah, 2021). For instance, with craft input like ‘OR ‘1’ = ‘1’, the logic of SQL can be manipulated to bypass authentication. It will lead to SQL Injection with unauthorized access, where sensitive data (passport numbers and prescription details) from patients, prescriptions, and the staff table, is exposed to data breaches and compliance violations.

#### 2.3.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: SQL Injection Risk (data protection)

As seen in the diagram above, the patient data can be accessed without permission, which causes a serious database security issue regarding data protection within APU Hospital.

#### 2.3.1.2 SOLUTION - Parameterized Queries (Stored Procedures), and Input Validation

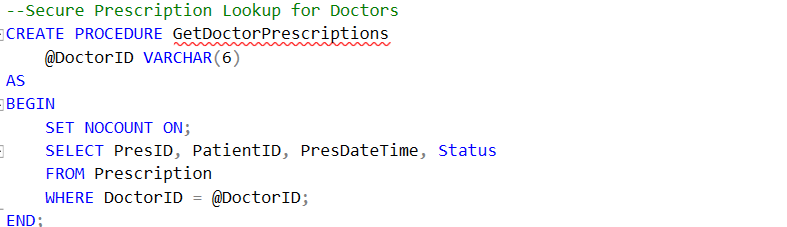


Figure 15: Stored procedure with parameterized queries and input validation

The stored procedures are created as figure above, to isolate the user inputs from the SQL logic, on all the user interactions with sensitive data access, such as staff details, patient login, and prescription lookup. For example, procedure “GetStaffDetails” will safely retrieve staff data, “ValidatePatientLogin” will authenticate patients using their passport number, and “GetDoctorPrescriptions” restricts doctors to view only their prescriptions.

Moreover, with “@PID” and “@PPassportNumber” as parameters, the SQL server will distinguish user input from the command structure, therefore preventing SQL injection attempts such as with 'OR '1'='1' and inaccurate credential details. Hence, the parameterized queries will bind user input securely and avoid query logic alteration. This method will effectively guarantee data confidentiality as unauthorized users cannot access the APU hospital database through crafted injection strings. In addition, the logical structure of the SQL queries will be protected to maintain data integrity, while it also ensures the database’s operational integrity, as the SQL injection threats of server disruption and data corruption are avoided.

#### 2.3.1.3 PROOF OF SOLUTION

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 16: Proof of stored procedure with parameterized queries

From the output above, the use of stored procedures with parameterized queries has eliminated SQL injection effectively, as unauthorized access is blocked, and hospital data is protected.

### 2.3.2 ISSUE 2 - Sensitive Data Exposure (Passport data for patient and staff)

The sensitive data, passport data, is stored as plain text in the patient and staff table, where rise concerns about data risk and privacy breaches. The sensitive data can also be accessed directly by anyone with the APU hospital database, even with read-only permission. Hence, this threat causes a risk of identity theft, fraud, or social engineering attacks, with the sensitive data obtained from the database. As sensitive data is visible to all users, it is a critical risk in data protection that will lead to a violation of privacy regulations, especially given the sensitive healthcare database environment.

#### 2.3.2.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17: Sensitive data exposure (data protection)

From the output above, the patient and staff table store the sensitive data, passport number in plain text, and can be accessed directly.

#### 2.3.2.2 SOLUTION - Asymmetric Encryption

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure 18: Asymmetric encryption

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure 19: Asymmetric encryption

A data-at-rest-protection mechanism with SQL server’s asymmetric encryption is implemented on the sensitive data, the passport field for both the staff and patient tables. Firstly, new encrypted columns 'Encrypted\_PPassportNumber’ and ‘Encrypted\_SPassportNumber’ are added. Then, a master key and asymmetric key, ‘PassportAsymKey’, are created to encrypt and decrypt the data, so that sensitive data is not visible in plain text and protected from unauthorized viewing. Therefore, only authorized users with the accurate private key can encrypt to access the sensitive data, ensuring data confidentiality ( Cloudflare, Inc, 2025). In addition, direct access to sensitive data is restricted, preventing unauthorised manipulation to protect data integrity and secure data availability to authorised users.

#### 2.3.2.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 20: Proof of Asymmetric data protection

The output above shows that the passport value is encrypted, plain text data exposure is eliminated, and sensitive data is stored securely. From the second SQL with the decryption private key, the passport value is shown, thus further confirming that the sensitive data is protected with an Asymmetric key.

## 2.4 STUDENT 4 – TEH YUE FENG TP069429

### 2.4.1 ISSUE 1 – Backdated Prescription Entries are Allowed

The very fact that backdating of prescriptions is allowed stands as a vulnerability against the data validation control of the medical system. This loophole allows anyone to put in a prescription record with a date preceding that of the actual entry time without any checks and validation. This weakness can then easily be exploited by ill-intended or careless users for the fabrication or alteration of medical records, to the disintegration of system integrity, and consequent severe legal, ethical, or medical ramifications to the detriment of the patient. Although an immediate threat may not pose an existential threat, the lack of enforcement of proper date constraints means that the system is inadvertently giving a potential for abuse, an indisputable vulnerability that needs to be remedied in the interest of trust, compliance, and patient safety.

#### 2.4.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 21: Backdate Prescription Entries are Allowed

Before the restrictions on adding terms were put in place, it was possible to add prescriptions that had already expired.

#### 2.4.1.2 SOLUTION - Prevent insertion of backdated prescriptions using a trigger

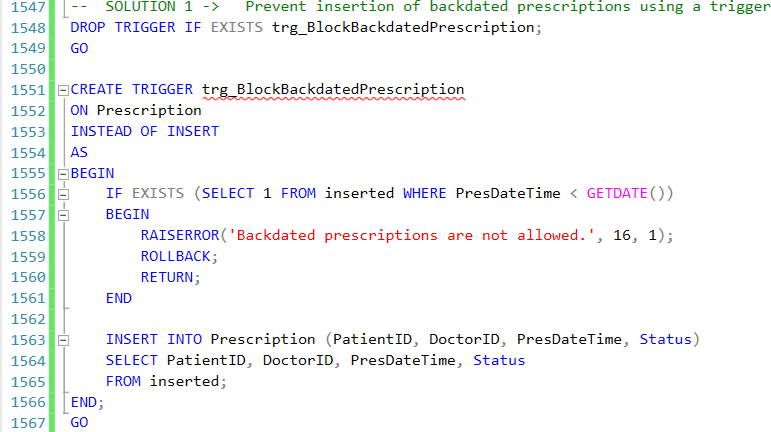


Figure 22: Prevention of Insertion Backdated Prescription

The SQL script depicts the trigger trg\_BlockBackdatedPrescription that avoids the insertion of backdated prescription records. The trigger is initiated before an insert is executed on the Prescription table; it checks if PresDateTime in the inserted table has any record whose timestamp is earlier than the current time (GETDATE()). If this is true, an error "Backdated prescriptions are not allowed.", error code 16, status 1 gets thrown by RAISERROR, and then a ROLLBACK occurs to cancel the operation; if not, the insert operation proceeds with inserting the PatientID, DoctorID, PresDateTime, and Status from the inserted tables into the Prescription table.

#### 2.4.1.3 PROOF OF SOLUTION

A screen shot of a computer

AI-generated content may be incorrect.

Figure 23: Failure after Solution Inserting Backdated Prescription

If we try to add an expiry time that has already passed, the system will not allow it.

### 2.4.2 ISSUE 2 – Plaintext Storage of Payment Information

Payment card numbers and PIN codes stored unmasked inside the Patient table would thus be a strong vector for exploitation since the data is entirely unprotected at rest. Database access can also mean direct access to sensitive financial information: the different roles could be either legitimate permissions, misconfigured roles, or an exploited SQL injection. Thus, this exposes the system to the possible exfiltration of such sensitive information, particularly where access control systems are lax or overly permissive. The absence of encryption further contravenes basic data protection principles and standards such as PCI-DSS, which explicitly call for encryption of cardholder data both at rest and in transit. Therefore, the plaintext storage of data leaves an access road wide open to attack from all kinds of directions. Even exposure through negligence would cripple the organization beyond repair. (Khadidos et al., 2022)

#### 2.4.2.1 PROOF

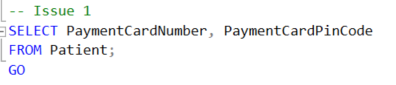
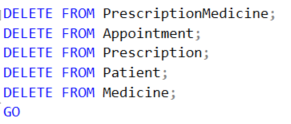


Figure 24: Sensitive Data

Then, the user would select the card number and card PIN code from the payment information and display them in a way that they can be easily seen.

#### 2.4.2.2 SOLUTION - Data Hashing for CardNumber and CardPinCode



A screenshot of a computer

AI-generated content may be incorrect.

Figure 25: Inserting Data Hashing values

Data hashing implemented a secure algorithm using payment card number and card pin code. This principle states that it directly underpins the confidentiality principle of the CIA triad because the actual PIN values will never be stored in a reversible or readable format. Thus, even if an attacker gains access to the hashed data, they will not be able to retrieve the original PIN without using long, cumbersome, and infeasible brute-force methods. Moreover, it also contributes to Integrity because it enables comparison of values entered without the need to expose the original input. For example, during authentication or verification, the system can compare a newly hashed input with the stored hash to confirm accuracy without having to reveal sensitive information. Hashing does not affect availability directly but based on the reduced complexity and risks associated with the implementation of real-time encryption and decryption operations, it provides indirect support for the same. In other words, hashing ensures that there are fewer storage complexities and risks arising from possible system crashes or performance degradation. Overall, data hashing is an effective, low-overhead, and standards-compliant method to secure static confidential data such as card PINs. (Pan et al., 2021)

#### 2.4.2.3 PROOF OF SOLUTION

A close-up of a number

AI-generated content may be incorrect.

Figure 26: Patient Data with Data Hashing

This shows the Patient chart after the information has been hashed. This way, the Patient's Card Number and Card PIN Code are not displayed.

### 2.4.3 ISSUE 2 – No Backup or Disaster Recovery Plan

A backup and disaster recovery plan would eliminate virtually all risk to the system. Rather, it is the probability of loss or damage due to the flaw that the risk represents. In this case, we threaten the risk of extirpation of data, extensive downtime, or maybe even complete operational denial if any operation is performed, causing corruption, deletion, or ransomware assault on the database. In the heat of being sensitive to any disruption in the system, without a backup or fail-safe plan, the threat becomes much more serious. Risks are heightened in a healthcare system where data is sensitive, constantly being accessed, and where operational continuity is paramount. Without the introduction of countermeasures, the risk will translate into legal liability, reputational damage, and irretrievable loss of patient records. (Nadee & Somwang, 2021)

#### 2.4.3.1 PROOF

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 27: Proof of Not Executing the backup

This section shows whether our database backups have been performed.

#### 2.4.3.2 SOLUTION - Implement scheduled backups and recovery processes

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 28: Execute Normal Backup

First, we need to make sure there's a manual Full Backup of the database. After that, we'll verify that the current state of the database is set to 'Full'.

***Full Backup***

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 29: Full Backup

The SQL code for automating the creation and scheduling of a full database backup job begins its execution by switching to the database context of 'msb' After this, it would create a step called 'AutoFullbackupstep' that executes a SQL command to return a full backup of the 'master' database. This will back up to a specific location with a dynamic filename that has the date in it. Next, it creates a schedule called 'FullBackup\_Every12hours,' which tells this job to run every 12 hours at 8:00 AM and 8:00 PM each day. If there exists a schedule with the same name, this code will delete it first and recreate it. At last, It links this job to the current server and enables it so that the automatic full backups begin according to schedule.

***Differential Backup***

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 30: Differential Backup

The automating SQL code creates and sets up differential backup jobs. It will then add a step, which it can call 'AutoDiffBackupStep', that will execute a SQL command for performing differential backup of the 'master' database. The backup will, hence, be saved to a specified location with a filename secured by a date in dynamically formatted strings. This code also configures a schedule entitled 'DiffBackup\_Every6hours' to enable the running of this job every six hours throughout the day, beginning from midnight until just before the next midnight. If such schedule name already exists, the code goes on to removing the previous one before creating a new one.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 31: Transaction Log Backup

The SQL code is created for automating the transaction log backups of your database. Then the code adds the step 'AutologbackupStep' for that job to invoke a command to back up the transaction log of the 'master' database to a specified location. The code then sets up a schedule 'Logbackup\_Every3hours' to run that backup job every three hours during the day, starting from midnight all the way to just before the next midnight.

The availability piece of the CIA model is instantly improved by establishing a backup and disaster recovery plan. The system allows retrieval of data in the event of hardware failure, corruption of data, or accidental deletion through full, differential, and transaction log backups. This consequently keeps the system uptime and vital healthcare processes running continuously. By allowing restitution to a known good state and making certain that incorrect or incomplete data cannot be used, this approach indirectly supports Integrity even though availability is the primary goal. Backup encryption and storage protected and controlled access area's confidentiality can be kept. (Bohora et al., 2021)

#### 2.4.3.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 32: Check Backup History

This section will show the history of backups, allowing us to confirm that each backup has run successfully.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 33: Status of Backup Run

We need to make sure that every backup completes successfully.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 34: Check Backup Information

This will display how long the entire backup system has been running and its regular schedule.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 35: Proof Restore Successfully in Different Server

After the backup is complete, if further proof is needed, we can restore the backed-up database. This section demonstrates that when I go to another server and restore what was backed up, the process is successful, and the database is operational。

# 3 PERMISSION MANAGEMENT

## 3.1 STUDENT 1 – MUHAMMAD NABIL HAKIM TP066763

### 3.3.1 ISSUE 1 – Lack of Role-Based Access Control

Role-Based Access Control (RBAC) is essential for database systems to ensure users only access information necessary for their roles. The current database lacks proper access control mechanisms where they allow all users to access and modify data beyond their designated responsibilities which creates significant security vulnerabilities.

#### 3.3.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 36: Before Updating Phone Number



Figure 37: After Updating Phone Number

The current database gives all users the same level of access privileges. This vulnerability exceeds bound abilities of users to perform actions outside their authorized roles where it creates the threat of inappropriate data access and modification. The risk includes unauthorized disclosure of patient information, fraudulent prescription modifications, and potential data breaches that violate healthcare privacy regulations and compromise patient trust.

#### 3.3.1.2 SOLUTION – Implement RBAC with Defined Roles

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 38: Granting Permissions for Each Role

The implemented solution establishes comprehensive Role-Based Access Control by creating specific database roles (Doctor, Pharmacist, Patient, Nurse, SystemAdmin) with precisely defined permissions that align with job functions. This approach enhances Confidentiality by restricting data access through commands while protecting integrity by limiting modification rights with appropriate UPDATE permissions. The solution includes proper user mapping to roles, creating a complete security framework that enforces the principle of least privilege.

#### 3.3.1.3 PROOF OF SOLUTION

A table with numbers and a phone payment

AI-generated content may be incorrect.



Figure 39: Successful and unsuccessful Views for Patient Table

After implementing the solutions, we can see that doctor can successfully view patient information and manage prescriptions but receives permission errors when attempting to modify medicine inventory. Similarly, a pharmacist can update prescription status and manage medicines but cannot create new prescriptions. Patient logins can only access their own information through the secure views, while nurses can manage appointments but not prescriptions. These results validate that each role has appropriate access to fulfill their functions while being restricted from unauthorized operations, demonstrating that the solution effectively implements the principle of least privilege while maintaining necessary system functionality.

## 3.2 STUDENT 2 – SOO JIUN GUAN TP068687

### 3.2.1 ISSUE 1 - Unauthorized Staff Position Changes

Any user is permitted to modify their own job title in the system. This is not restricted and is a danger from the security and integrity viewpoint, as there is potential privilege escalation by unprivileged users, such as advancing one's own self into that of an admin or doctor. This is against the principles of role-based access control and has risks associated with it such as abuse of power, alteration of data, or entry into sensitive operations that one is unqualified for. Without control measures, the system is exposed to threats from within and is not accountable in relation to alterations (Gardiner, 2024).

#### 3.2.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 40: Non-admin users can modify their position field

As shown in the above figure, one staff member (e.g., ST002) was able to directly alter their own role to 'Doctor' using an UPDATE statement, showing the lack of boundary checking and the potential vulnerability to privilege abuse.

#### 3.2.1.2 SOLUTION - Block staff from changing their job position unless they are Admin

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 41: Block staff from changing their job position unless they are admin

To address this problem, an INSTEAD OF UPDATE trigger trg\_PreventStaffChangePosition was created on the Staff table to prevent changing the position unless done by a user in the 'Admin' role. This strategy allows fine-grained control in SQL Server, employing IS\_MEMBER('Admin') to impose role-based access control. It maintains Confidentiality by limiting access to sensitive roles, Integrity by not allowing disenfranchised alterations, and Availability by permitting updating of non-sensitive fields. This is unlike application-layer control, which is not able to be bypassed by direct SQL Server access.

#### 3.2.1.3 PROOF OF SOLUTION

A screen shot of a computer

AI-generated content may be incorrect.

Figure 42: Non-admin users try to change their own position

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 43: Admin can modify staff's position

As confirmed in the above figures, if a regular user tries to alter their own account, the system prohibits the updating and yields an exception, whereas the admin user (AD001) has been able to make valid updates, validating that the solution properly implements role-based controls and resolves the problem.

### 3.2.2 ISSUE 2 – No Row-Level Access Control for Staff and Patient Records

The system fails to enforce access boundaries based on user identity or role, allowing staff and patients to access each other’s full personal records. This undermines role segregation and introduces threats such as unauthorized browsing, internal profiling, and exploitation of confidential data. When sensitive information like ID numbers, contact details, and payment records is freely accessible, the risk of identity theft, reputational damage, and data misuse increases. Additionally, the lack of segmentation significantly expands the threat surface and operational risk.

#### 3.2.2.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 44: Staff and patient can see others' personal info

As shown in the figure, performing a simple SELECT \* on the Staff and Patient tables returns complete rows from all users with no access boundaries in evidence, demonstrating a serious security weakness.

#### 3.2.2.2 SOLUTION - Enforce row-level security to restrict staff and patient access

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

Figure 45: Use RLS to restrict staff and patients to only their authorized records

In order to address this problem, Row-Level Security (RLS) was applied through SQL Server’s native policies. Two functions, fn\_RLS\_StaffOwnRecord and fn\_RLS\_PatientOwnRecord were defined to filter by user identity and role and enforced through StaffRowFilter and PatientRowFilter with FILTER and BLOCK predicates (Yaseen, 2019). This method provides fine-grained control at the database level so that users access only rows that matter to them and maintains Confidentiality by hiding unassociated records, Integrity by blocking invalid changes, and Availability by providing valid access without affecting workability. Compared to application-level checks, RLS is enforced in the SQL engine and therefore cannot be bypassed through direct queries.

#### 3.2.2.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 46: Staff and patients view only their own records

As shown in the above figure, testing with a staff user (ST003) and patient (PT002) resulted in each one of them only able to see each of their own individual record, verifying that row-level security has successfully isolated access to approved data alone.

### 3.2.3 ISSUE 3 - Unrestricted Modification of Prescription Data by Pharmacists

Pharmacists currently have complete editing permissions on prescription history, including fields that are reserved purely for physicians like DoctorID, rather than being restricted to updating solely relevant fields like prescription status. This deficiency in boundary control poses severe integrity threats, with the possibility of unauthorized modifications to key medical assignments and allowing potential impersonation or misrepresentation of prescribing responsibility. Without implementing strict field-level control, the system is vulnerable to falsification of data, incorrect treatment recording, and abuse by in-house users.

#### 3.2.3.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 47: Pharmacist modifies DoctorID without restriction

As demonstrated in the illustration, a pharmacist is able to successfully modify the DoctorID field with an explicit UPDATE statement, revealing the system's failure to ensure role-specific update constraints and showcasing the danger of privilege abuse.

#### 3.2.3.2 SOLUTION - Restrict prescription status modification via stored procedure for pharmacists

A screenshot of a computer program

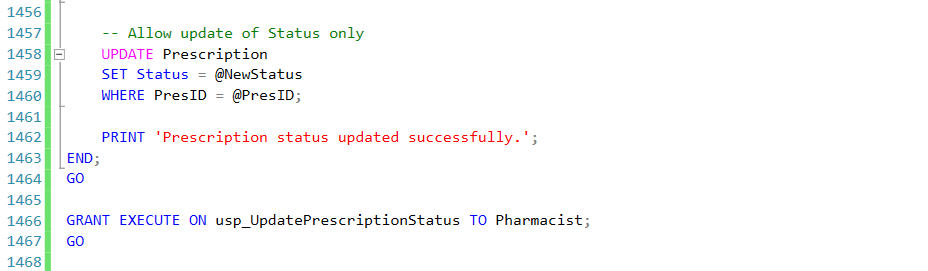
AI-generated content may be incorrect.

Figure 48: Stored procedure to restrict status updates

To address this, a stored procedure (usp\_UpdatePrescriptionStatus) was implemented, and a REVOKE UPDATE command was used to remove direct table updating permissions from pharmacists. The procedure checks the user's identity and role first and only allows updating after that, specifically only in the Status column. This maintains Confidentiality by keeping unauthorized role edits from occurring, Integrity by limiting updating of important data only to permitted users, and Availability by ensuring pharmacists are still able to execute required updates in security. This is enforced at the database layer and cannot be side-stepped even with direct SQL commands (i.e., UPDATE Prescription SET ...), making it safer than implementing validation in the application.

#### 3.2.3.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 49: Successful status update using stored procedure

A screenshot of a computer

AI-generated content may be incorrect.

Figure 50: Direct update blocked by permission control

As demonstrated in the above two screenshots, the pharmacist only succeeded in updating the Status column with the help of the procedure, whereas any explicit updating effort to DoctorID resulted in permission failure, ascertaining that the solution strictly enforces restricted updates.

## 3.3 STUDENT 3 – TAN SHI YING TP065227

### 3.3.1 ISSUE 1 - Unauthorized Appointment Access and Updates

The appointment system in the APU hospital database allowed full access to all users, exposing scheduling data and risking integrity violations. It causes a significant privacy breach as patients can see others’ data, and medical errors caused by unauthorised appointment status changes, such as from “new” to “done,” will cause a huge mistake in the healthcare system. Not only that, but the appointment table in the database is also exposed to the risk of data integrity loss resulting from unauthorized insertions, deletions, or changes, and the disruption of appointment scheduling or impersonating another role.

#### 3.3.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 51: Proof of unauthorised appointment access and updates

From the SQL above, the patients can view all the appointments in the database, including those of other patients, due to the open-access data of the APU hospital database. In addition, duplicated appointment data can be added to the database as well as updating appointment status directly without any permission needed, which causes data integrity risk in the database.

#### 3.3.1.2 SOLUTION - Centralized Access Control via “ManageAppointment” Procedure

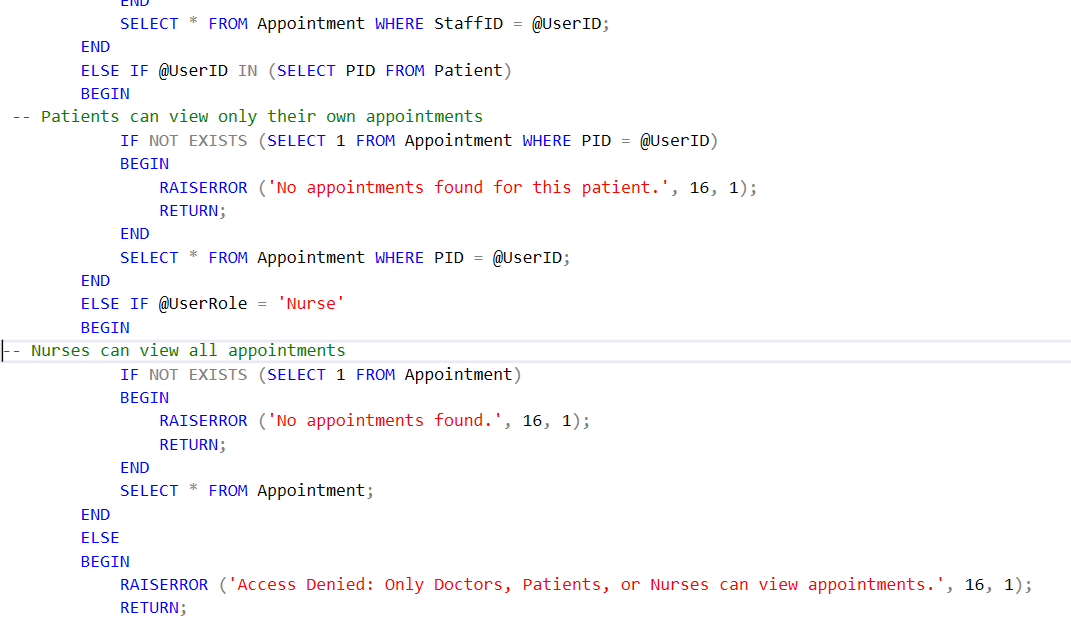
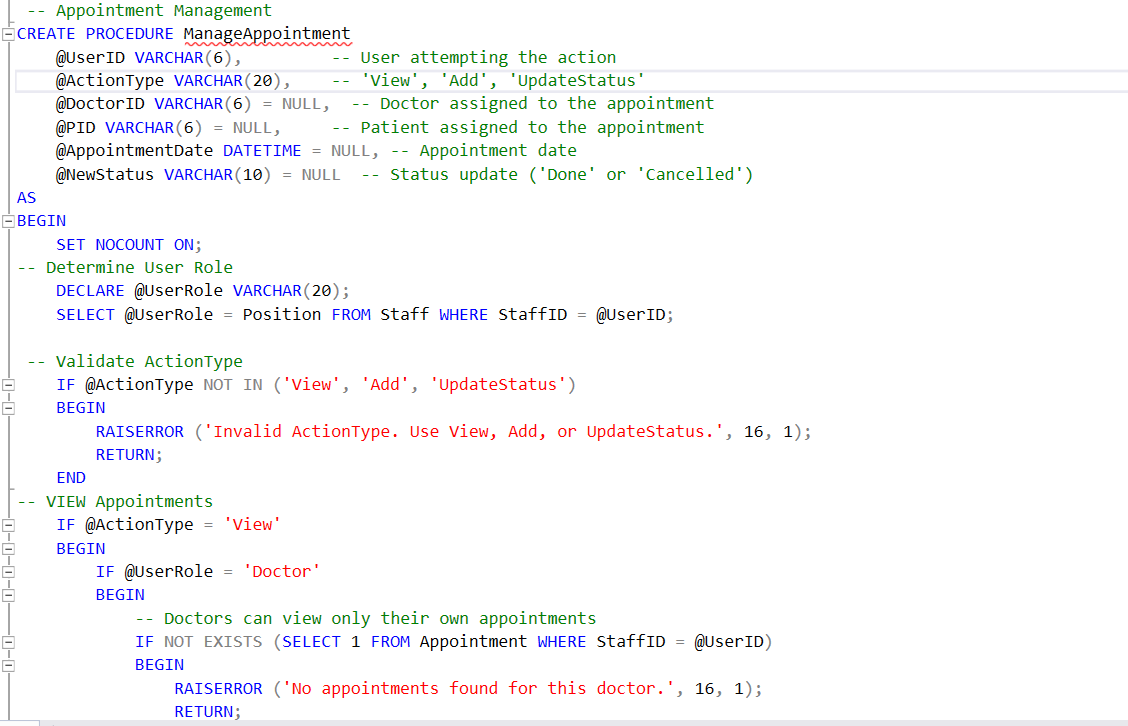


Figure 52 : 'ManageAppointment' stored procedure 1

The “ManageAppointment” procedure created a controlled access layer that restricts appointment visibility based on user role, which directly solves the problem of unauthorized access by ensuring that only doctors, nurses, or the actual patient can view relevant data. For instance, it will check the staff ID and validate the patient, ensuring tight, role-specific data access that aligns with privacy principles in healthcare sector.

A screenshot of a computer

AI-generated content may be incorrect.

Figure :53 'ManageAppointment' stored procedure 2

From the ‘ManageAppointment’ stored procedure, the ‘add’ logic is designed to prevent unauthorized users from adding new appointments and ensure that only validated, non-duplicate records are added by authorized nurses. This solution demonstrates strict permission control, therefore eliminating errors from duplicate scheduling, and ensures the hospital appointment system remains accurate and secure.

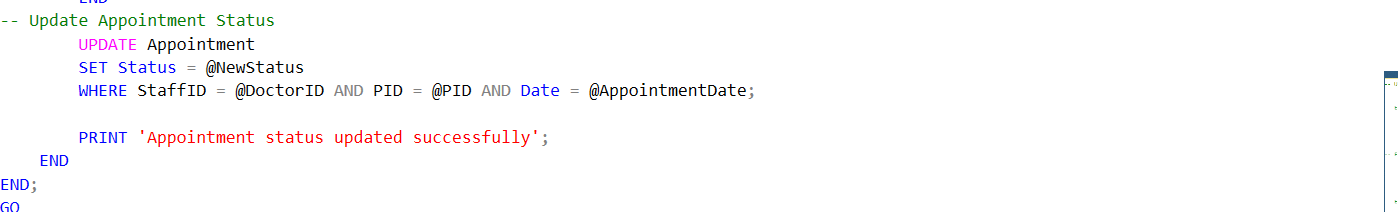
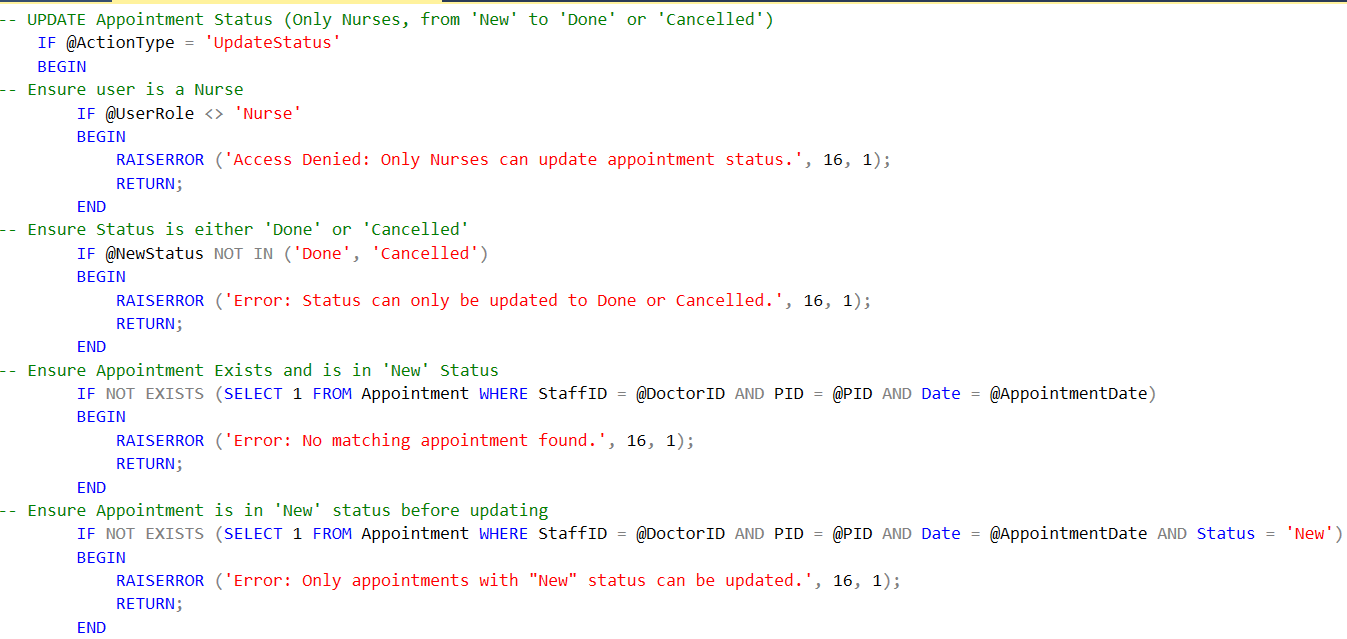


Figure :54 'ManageAppointment' stored procedure 3

For the ‘UpdateStatus’ in the same ‘ManageAppointment' stored procedure, it ensures that only authorized nurses can update the appointment status, and only for existing, active (‘New’) appointments. Hence, the appointment table in the APU hospital database is protected from unauthorized tampering, data inconsistency, and ensures that appointment statuses reflect true, authorized changes in the workflow. The Appointment’ stored procedure authorises the user to access the appointment table, guarantees database confidentiality and integrity while aligning with the business rule of APU hospital.

#### 3.3.1.3 PROOF OF SOLUTION

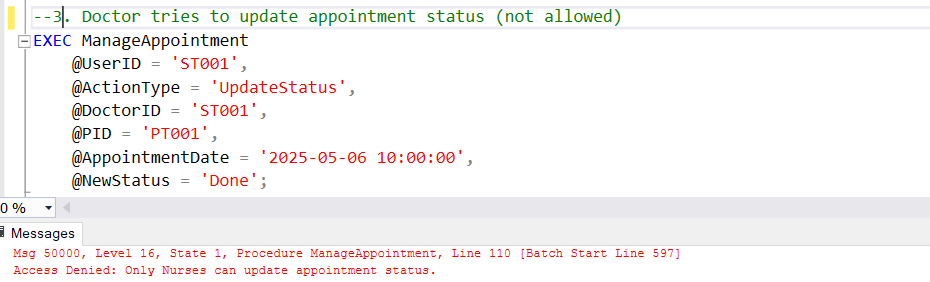
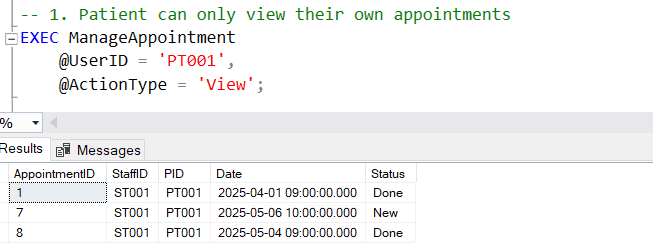


Figure :55 Proof of stored procedure “ManageAppointment”

From the test cases above, the ManageAppointment procedure has successfully enforced strict permission management rules based on the user roles and business rules in APU Hospital. It ensures only authorized actions are permitted, blocking privacy breaches, duplicate appointments, and unauthorized status updates, thereby securing the confidentiality, integrity, and availability of the appointment data. In short, store procedure is a secure access control mechanism to enforce permission management at the database level, effectively preventing unauthorized users from viewing or modifying appointment data based on their role (Kime, 2023).

### 3.3.2 ISSUE 2 - Unauthorized Prescription Modification by Doctors

In the APU hospital database, the doctor can access the prescription table directly, even unintentionally or maliciously modify prescriptions that they do not own. Consequently, it will cause a risk of prescription duplication, conflict, or cancellation errors, which will then lead to inaccurate medication data that might result in serious clinical harm (Todd, 2025). Without proper permission management in the prescription data, data integrity is compromised as prescription ownership is not enforced. Furthermore, a medical liability due to unauthorized prescription changes will occur, as well as a violation of professional boundaries and hospital policy.

#### 3.3.2.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure :56 Unauthorized Prescription Modification by Doctors

The test case above shows that any doctor can modify all the prescriptions, including another doctor’s prescriptions, which can lead to cancelling or reassigning treatment plans and adding prescriptions for patients under different doctors.

#### 3.3.2.2 SOLUTION - Stored Procedure “ManagePrescription” with Ownership Control

A screen shot of a computer

AI-generated content may be incorrect.

Figure :57 Stored procedure “Manage Prescription” 1

A stored procedure “ManagePrescription” is created to grant ownership control over the prescription data. First, the stored procedure will allow only two actions: add a prescription or update the prescription status. Then, it will validate whether the user is a doctor, with StaffID, allowing only Doctors’ access.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure :58 Stored procedure “Manage Prescription” 2

Moving on, the stored procedure will validate the patient to restrict 'AddPrescription' so a doctor can only prescribe for their respective patients. This action ensures prescription data confidentiality and integrity, as other doctors cannot access and modify someone else’s prescription/

A screenshot of a computer program

AI-generated content may be incorrect.

Figure :59 Stored procedure “Manage Prescription” 3

Furthermore, the stored procedure restricts 'UpdateStatus', allowing only the owning doctor of a prescription to update its status, and allows status changes to 'New' or 'Cancelled' only. Hence, the stored procedure successfully validates permission on prescription data, blocking cross-editing of prescriptions.

#### 3.3.2.3 PROOF OF SOLUTION

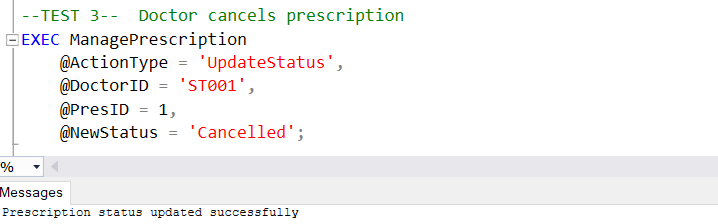
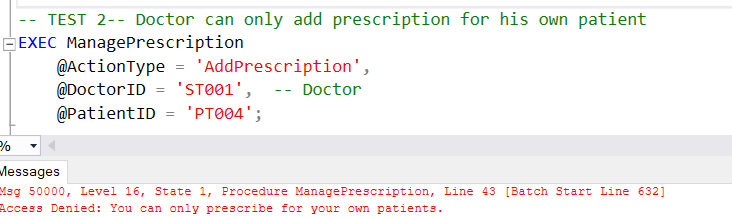


Figure :60 Proof of "ManagePrescription"

The test cases above confirm that the ManagePrescription procedure successfully enforces role and ownership validation, a doctor-based access control. Unauthorized prescription additions or updates are denied, while valid, ownership-based actions are permitted. Hence, the prescription data are safeguarded with appropriate user permission management within the APU hospital database to ensure data privacy and maintain clinical accuracy.

## 3.4 STUDENT 4 – TEH YUE FENG TP069429

### 3.4.1 ISSUE 1 - Improper Permission Control for Nurse

This specific issue has become a straightforward vulnerability case in the permissions architecture of the system. The Nurse doesn’t incorporate restrictions for access to and changes on the sensitive tables, such as the Patient. Absence of explicit boundary permissions allows unauthorized access to data beyond that which is related to nurses, such as seeing other patients’ prescriptions or changes in appointment statuses. The core problematic issue is the wrongful configuration in access controls, a design flaw within the system architecture, not a present threat or realized attack. Although the vulnerability may be exploited by some internal threat actors, that is, malicious or careless nurses, the problem itself does not represent a deliberate attack or measure of misuse probability or impact. Therefore, it is not a threat requiring an actor and intention or a risk that weighs up likelihood and consequence, but a vulnerability, a weakness in how the system is configured to elevate the potential for compromise to future levels.

#### 3.4.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 61: Test Access for Nurse Role

This is where a user logs in. If they log in as a nurse and view the Patient table, they will be able to see all the information, including sensitive data.

#### 3.4.1.2 SOLUTION - Restrict access with grants.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 62: Solution of Solving Nurse Over-permission

To address the over-permission vulnerability, the solution implemented was to explicitly grant only the necessary access privileges to the Nurse, using the GRANT statement. This approach aligns with the Confidentiality principle in that nurses may only see non-sensitive data-for example, patient name, phone number, and patient assignments, should have no means to access sensitive patient financial records. Through mitigating exposure, it also dampens the potential for leakage or internal misuse of data. This method also strengthens Integrity limits on a nurse's ability to effect unauthorized changes in any capacity (e.g., unauthorized UPDATEs or DELETEs) such that the correctness of data would be compromised. Even though Availability will not be directly compromised, the application would remain stable and functional for the nurse within that limited scope to prevent the nurse from doing their work while being given unnecessary access. Overall, through GRANT permission scoping, RBAC was assured, thereby protecting the system from insider threat and unintentional misuse.

#### 3.4.1.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 63: Proof Nurse Access

After logging in as a nurse, viewing the Patient table again will display the non-sensitive information

A screenshot of a computer

AI-generated content may be incorrect.

Figure 64: Access Denies for Nurse

However, even if a nurse is logged in, they won't be able to view or change information in the patient table. Access to that has been restricted, and they can only see non-sensitive details.

# 4 AUDITING

## 4.1 STUDENT 1 – MUHAMMAD NABIL HAKIM TP066763

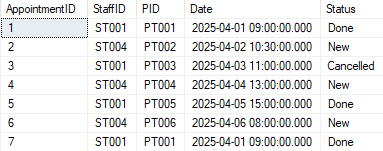
### 4.1.1 ISSUE 1 – Lack of Audit Trail for Critical Operations

Audit trails are crucial security components in healthcare database systems, providing accountability and tracking for sensitive operations. The current database lacks audit mechanisms for tracking critical actions like record deletions and as a result, creating significant security gaps in accountability and traceability.

#### 4.1.1.1 PROOF

A screen shot of a computer

AI-generated content may be incorrect.

 A screenshot of a computer

AI-generated content may be incorrect.

Figure :65 Deletion of Data from Appointment Table

The current database allows administrators to delete critical records without any tracking or accountability which permanently removes staff data without logging the operation. This vulnerability exceeds the bound ability of administrators to perform destructive actions without oversight, creating the threat of undetected malicious deletions. The risk includes permanent loss of critical medical staff records, potential disruption of hospital operations due to missing data, and inability to investigate security incidents or data tampering, thus ultimately compromising regulatory compliance and data governance capabilities.

#### 4.1.1.2 SOLUTION – Use SQL Server Audit for DML Audit

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure :66 Stored Procedures and Auditing Setup for Appointment and Patient Deletion

The solution creates an audit framework using SQL Server AUDIT and database-specific audit specifications targeting Staff table operations. This approach enhances Confidentiality by tracking who accesses and modifies sensitive data, protects Integrity through the stored procedure usp\_DeletePatient and usp\_DeleteAppointment which prevents deletion of referenced records and logs all deletion attempts, and maintains Availability by ensuring critical data relationships remain intact. The implementation includes detailed server-level auditing with proper file management and rotation supporting both security incident investigation and regulatory compliance requirements.

#### 4.1.1.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 67: SQL Audit from File Location

After implementing the solution, the testing validates the effectiveness of the audit system through specific deletion scenarios. When executing deletion procedures, the system properly checks for dependencies logs attempted operations and either blocks deletions where referenced data exists or captures complete record details before removal. Audit records successfully store user identity timestamp and operation details in secure audit logs. These results demonstrate comprehensive accountabilities for all high-impact operations enhancing confidentiality through complete tracking of sensitive data access Integrity through prevention of orphaned records and Availability by ensuring critical patient-appointment relationships remain intact supporting both operational reliability and regulatory compliance with healthcare data protection standards.

## 4.2 STUDENT 2 – SOO JIUN GUAN TP068687

### 4.2.1 ISSUE 1 – No Audit Trail for Patient and Staff Table Changes

The system of the database fails to record modifications that have been made to tables like Staff and Patient, resulting in no visibility whatsoever into who made what and when. This lack of an audit record undermines accountability by making it hard to identify unauthorized changes or resolve disputes regarding data. Without these changes being stored, the system is open to hidden tampering, compromising the certainty of stored data and incapacitating effective monitoring by data custodians (Vicente, n.d.).

#### 4.2.1.1 PROOF

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 68: No CDC enabled on database and target tables

As in the figure, CDC is not enabled on the MedicalInfoSystem database or on the Patient and Staff tables, meaning that no change history is recorded on these sensitive tables.

#### 4.2.1.2 SOLUTION - Enable CDC (Change Data Capture) for auditing updates to Patient and Staff tables

A screenshot of a computer program

AI-generated content may be incorrect.A computer screen shot of a code

AI-generated content may be incorrect. A close-up of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 69: Enable CDC for Patient and Staff Tables and Grant Access to Admin

In order to address this, Change Data Capture (CDC) was enabled at the database level and applied on both the Patient and Staff tables. Native SQL Server feature that captures insert, update, and delete operations asynchronously in dedicated CDC tables (Patient\_CT, Staff\_CT) tracks changes (Richman, 2025). The system is architected to support Confidentiality by allowing authorized roles (i.e., Admin) to query CDC logs, Integrity by storing original and changed values to be verified by audits, and Availability by utilizing CDC's nonintrusive architecture that has minimal performance overhead. Unlike triggers or app-level logging, CDC is not subject to tamper and takes place independent of client apps, providing consistent data monitoring at the SQL engine level.

#### 4.2.1.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 70: CDC captures and logs data modifications

As seen in the above diagram, once the data of patient is changed, the modification is successfully stored in the CDC table, ensuring that CDC is properly capturing and storing all the modifications.

### 4.2.2 ISSUE 2 - No Login Activity Is Audited or Tracked

There is no logging system to record user login activity or access usage, and hence there is no detection of suspicious behavior or unauthorized intrusion. This blind area in boundary monitoring lowers the system's resistance to privilege misuse and credential abuse. There is no method to authenticate the user, detect suspicious behavior, or impose responsibility for abuse in the absence of server-side logging, and there is thus a major security hole in the entry point (OWASP Foundation, 2021).

#### 4.2.2.1 PROOF

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 71: No Login Activity Tracked

As indicated in the above figure, there is no LogonAudit table and there is no trace left behind by test logins, demonstrating that SQL Server has no functioning audit trail facility in place for logins.

#### 4.2.2.2 SOLUTION - Implement server-level logon auditing with a trigger and store logs in a custom table

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect. 

Figure 72: Implement Server-Level Logon Trigger with Secure Audit Table Access

To solve the issue of login activity logging, a server-level trigger called trg\_LogonAudit\_HospitalSystem was created for logging every login attempt in its own audit table, LogonAudit, in the database named MedicalInfoSystem. The trigger makes use of SQL Server-built-in functions like ORIGINAL\_LOGIN(), HOST\_NAME(), and APP\_NAME() in order to capture key session metadata, such as the user's login, host machine, client app, and login time. To provide additional control, the solution also features a session-limited capability that caps users' concurrent sessions at three, thereby reducing the risk of access abuses made with credentials (Gupta, 2021). In doing so, it adds an extra layer of Confidentiality through limiting exposure of access activity, of Integrity through blocking simultaneous unauthorized sessions, as well as of Availability through allowing legit users within prescribed bounds. The audit data is tightly secured, which is accessible only for the Admin role, such that login details are shielded from unauthorized inspection or manipulation. As this mechanism is enforced natively within the SQL Server engine, it provides an informed, tamper-evident solution independent of applications' control.

#### 4.2.2.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 73 Logon Event Captured

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 74: Logon failed due to exceed maximum sessions

As can be clearly demonstrated from the above figures, user logins like ST001 are successfully logged in the LogonAudit table with complete metadata, whereas unauthorized user login activities above session thresholds (like the second login of ST001) are thwarted immediately through the trigger, verifying successful server-side login auditing and control.

## 4.3 STUDENT 3 – TAN SHI YING TP065227

### 4.3.1 ISSUE 1 – Staff Data can be Deleted Without Trace

The APU hospital database has the following auditing vulnerabilities, as it allows users to drop, alter, or create objects such as the staff table, without any logging or audit trail. Hence, this vulnerability causes the threat of the tables in the database being deleted or modified, such as ‘DROP TABLE Staff’ or alter schema ‘ALTER TABLE’ and ‘CREATE’, without being detected by the staff member or attacker with elevated privileges. Subsequently, it will result in audit failure during investigation, and difficulties in identifying the source and impact of unauthorized schema changes.

#### 4.3.1.1 PROOF

A screenshot of a computer

AI-generated content may be incorrect.

Figure 75: Proof of no auditing on logging

The figure above shows the attempt to query "sys.fn\_get\_audit\_file()", which shows no results, indicating the absence of an audit log in the APU Hospital database. This reflects a critical database security issue in auditing, exposing the system to untraceable schema modifications, violating the database integrity and accountability.

#### 4.3.1.2 SOLUTION – SQL Server Data Definition Language (DDL) Auditing

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 76: DDL Auditing Solution 1

Firstly, a server-level audit is created, with the audit log being saved in the local file (C:\Temp\AuditLogs\), and any changes made at the server level will be tracked in the file (Microsoft, 2024).

A screenshot of a computer program

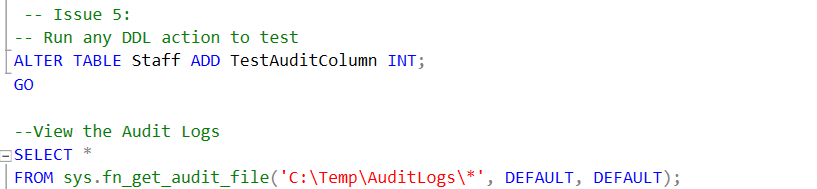
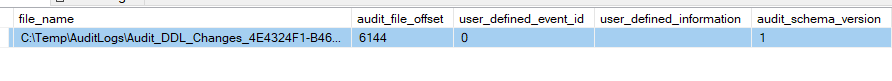
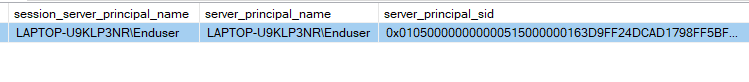
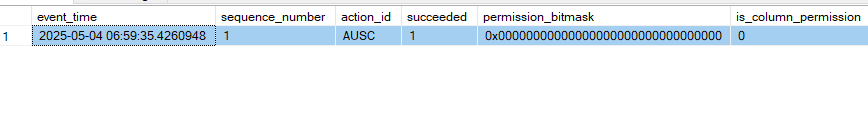
AI-generated content may be incorrect.

Figure 77: DDL Auditing Solution 2

Furthermore, a database-level audit specification is created as shown above, it is used to capture all DDL changes, including the “CREATE”, “ALTER”, and “DROP” activities such as those on the Staff table in the APU Hospital database.

#### 4.3.1.3 PROOF OF SOLUTION

Figure 78: Proof of DDL Auditing



The SQL above activates and verifies the auditing by allowing visibility into who did what and when on the APU Hospital database. Actions such as who dropped or modified a table, the time, and the identity of the user (IP/user), and what statement was executed will be recorded in the logs. The sample output above shows a part of the auditing log, displaying the alter table command along with user information and timestamp data, and details the affected object. The logged row confirms that DDL auditing operates correctly in the APU Hospital database.

The solution activates DDL auditing through SQL Server so that schema modifications will be logged and traceable (IBM Corporation, 2023). Therefore, the healthcare database is secured with accountability, improved forensic investigation, and maintains vital data security auditing principles.

## 4.4 STUDENT 4 – TEH YUE FENG TP069429

### 4.4.1 ISSUE 1 - Lack of Change History Tracking on Patient, Medicine, and Staff tables.

System-versioned temporal tables (SYSTEM\_VERSIONING = ON) are an exemplary best practice in using auditing changes in the Patient, Medicine, and Staff tables and making the events traceable. If, on the other hand of these temporal tables, such as PatientHistory or StaffHistory, they are misconfigured, it will open a backdoor to the same auditing mechanism. For instance, if these history tables didn't have a proper access control like DENY SELECT or DENY DELETE for unauthorized roles, a malicious user could easily read or write to a historical record. In addition, there are also performance degradation and reliability issues with an audit trail of things like missing indexes, uncontrolled data growth, or failure to validate ValidFrom and ValidTo fields. This does not present a threat, for it is not a way to talk about an active or intentional attempt to attack the system. Likewise, it is not a risk because it does not evaluate the likelihood or impact of data loss or manipulation. It is rather a failure in design toward an implementation of audit—a textbook definition of a vulnerability that could be exploited if not resolved. (Anwar et al., 2021)

#### 4.4.1.1 PROOF

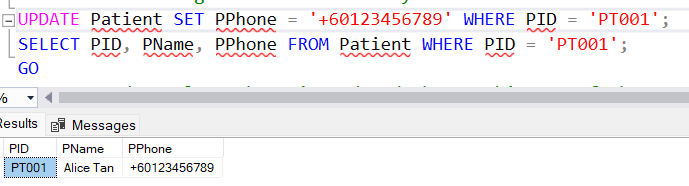
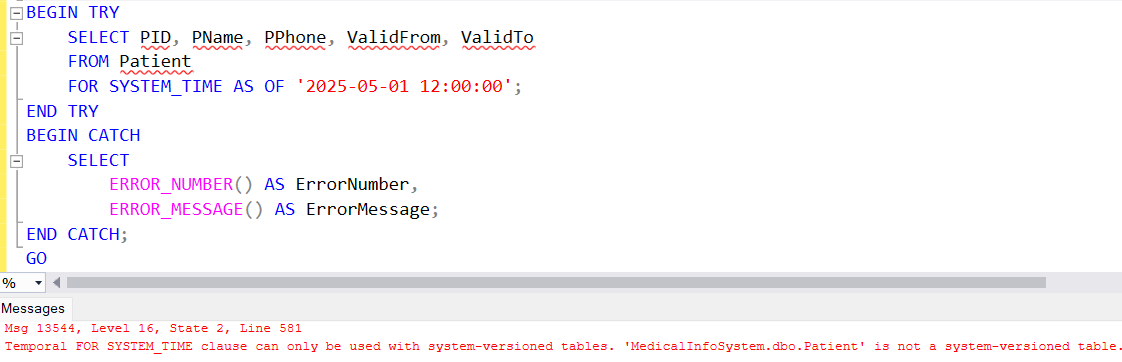


Figure 79: Proof of Missing Temporal Table

Initially, the system will show whether Temporal Tables exist. It will also indicate that when a user updates data, there are currently no recorded changes.

#### 4.4.1.2 SOLUTION - Implement system-versioned temporal tables to track all changes without altering schema logic.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 80: Solution Inserting System Temporal Table

To mitigate the vulnerability posed by inadequate audit tracking along with potential gaps in configuration, system-versioned temporal tables were enforced on the Patient, Medicine, and Staff tables. This applies automatic recording of any change in data along with the previous value, timestamps (ValidFrom, ValidTo), and history of modifications to all records. Thus, it gives full traceability of records independent of any manual logging and triggers. Integrity is reinforced via temporal tables which provide an immutable audit trail for all updates. Under ordinary circumstances, the audit trail updates can never be altered. Under appropriate access control measures that restrict access to the history tables to auditors or admins, this ends up further contributing to Confidentiality. Availability applies here while performance considerations were kept in mind in the sense of ensuring that key columns are indexed, and growth storage is monitored.

#### 4.4.1.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 81: Temporal Table Result

This section will provide several ways to demonstrate that the temporal table truly exists. For example, it will show when a user deletes their information, when a user updates their own details, and the effective dates for the information.

### 4.4.2 ISSUE 2 - No Prescription/Appointment Audit

Incomplete accountability is a vulnerability brought in due to lack of audit tracking important tables like Prescription or Appointment. Hence, no historical records could identify what has been modified, who has modified it, and when. Post-event investigations become almost impossible due to this fact, and trust regarding the integrity of the data gets broken. In absence of audit logs or versioning, illegal modifications may remain unnoticed for a long period, gradually contaminating system data.

#### 4.4.2.1 PROOF

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 82: Availability of Audit Log Table

This indicates that the audit log table for appointments and prescriptions has not been created yet.

#### 4.4.2.2 SOLUTION - Implement triggers to record inserts, updates, and deletes.

Without audit tracking for any important tables, such as Prescription, Patient, or Appointment, there is an inherent vulnerability arising out of the weakness in accountability. Once changes are made-whether maliciously or by mistake-there would exist no historical records to identify what changed, who made that change, or when the alteration was made. This, in turn, creates distrust in data integrity and almost ruins the possibility of investigating the matter. The security implications of unauthorized modifications are such that without audit logs and/or version tracking, they could go unnoticed and remain undetected for long periods until many system data are corrupted under their influence. (Alexander & Denis, 2021)

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 83: Create AuditLog Table

To address the update on table prescription and appointment, create an audit log table containing AuditID, TableName, Operation, RecordID, ChangedColumns, ChangedBy, and ChangedAt.

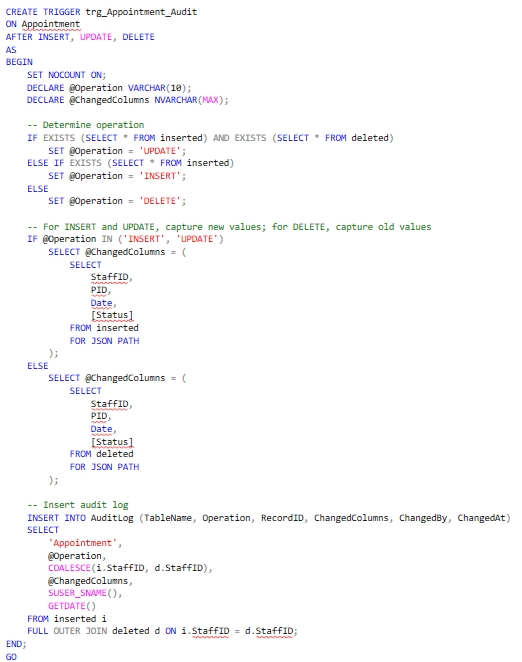
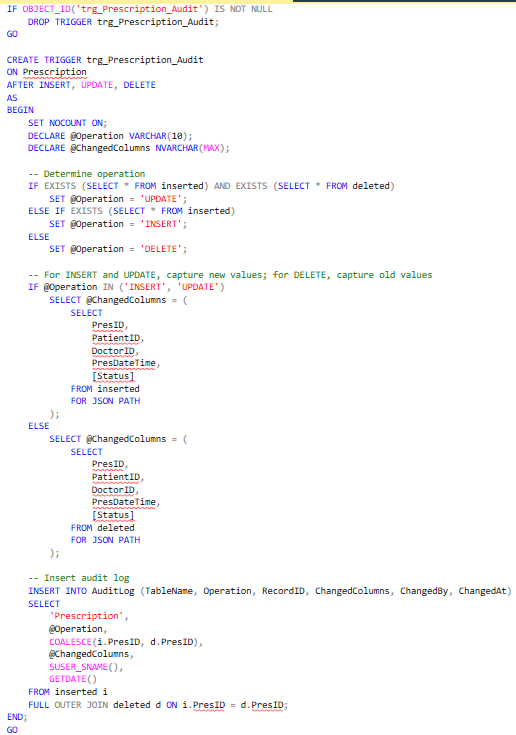


Figure 84: Trigger Audit on Prescription and Appointment

It is a script for the creation of a trigger called 'trg\_Prescription\_Audit' and 'trg\_Appointment\_Audit' which is fired automatically in the case of INSERT, UPDATE, or DELETE on 'Prescription' and 'Appointment'. The type of operation in concern is captured along with the exact columns that were modified (i.e., PresID, PatientID, DoctorID, PresDateTime, Status). Otherwise, Appointment involved column(StaffID, PatientID, Date, Status). In the case of an insert or a delete, it will pull the necessary values from temp tables 'inserted' or 'deleted', respectively. Details like the name of the table, 'Prescription' or 'Appointment' the operation type, the changed record ID, changes columns, who did the change, and when it occurred is inserted into the 'AuditLog' table, thereby enabling tracking of any change made into the 'Prescription' and 'Appointment' table. Basically, therefore, all changes regarding the two tables should be logged.

#### 4.4.2.3 PROOF OF SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.

Figure 85: Backup History

This section will only allow users with Admin privileges to view the entire AuditLog table.

A number with numbers on it

AI-generated content may be incorrect.

A number with numbers on it

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 86: Successfully Back Up

# 5 CONCLUSIONS

In conclusion, this project fundamentally upgraded the security stance of the APU Hospital’s MedicalInfoSystem as 22 specific vulnerabilities with serious implications for the confidentiality, integrity, and availability of healthcare information were remediated. The issues were methodically put in categories of data protection, permission management, and auditing, thereby allowing for remediation through a focused, structured approach.

For data protection risk, data fields such as passport numbers, card numbers, and telephone numbers were protected using dynamic data masking, transparent data encryption (TDE), asymmetric encryption, and hashing methods, protecting against the exposure of data. In addition, for the database’s permission management, it was restricted through strict role-based access controls using stored procedures, triggers, and user-role mapping in order to ensure users could only read and update data for their line of work. For auditing, features such as Change Data Capture (CDC), temporal tables, DDL auditing, as well as server-level logon triggers, were utilized in order to trace all updates of the data as well as access events, for complete accountability and traceability. These solutions not only worked technically but were also aligned with hospital workflow realities in the real world, with little disruption while strengthening its defense. As such, the database system of APU Hospital is now on a secure as well as compliance foundation with the capability for protection of sensitive medical data, ensuring operational integrity, as well as the capability for withstanding possible internal as well as external attacks in today’s healthcare environment.

# 6 REFERENCES

Alexander, B., & Denis, M. (2021). SECURITY AUDIT LOGGING IN MICROSERVICE-BASED SYSTEMS: SURVEY OF ARCHITECTURE PATTERNS. *Вопросы кибербезопасности*, *2 (42)*. https://cyberleninka.ru/article/n/security-audit-logging-in-microservice-based-systems-survey-of-architecture-patterns

Anwar, M., Panjaitan, R., & Supriati, R. (2021). Implementation Of E-Business Information System In Indonesia : Prospects And Challenges. *International Journal of Cyber and IT Service Management (IJCITSM)*, *1*(2), 197–205. https://d1wqtxts1xzle7.cloudfront.net/82301340/v1\_n2\_06-libre.pdf?1647576231=&response-content-disposition=inline%3B+filename%3DImplementation\_Of\_Database\_Auditing\_By\_S.pdf&Expires=1746345450&Signature=Jqzw8~vP08PeEedvZq08WzRPFs5tYu8-gsH6uufCPfDrzv5hHa636lVto0pDsWONdMEWDCGUiDqrs-WRuODJBb~FwTR1oQuNC3mQWi~Ps7VaJd6JSMUMv~S5F1AmpNwmpsyOSOUeEhXaVIX7KmWT62RYLTZeqtuez9kNVyYDnJ6EOjMtCHPs7ds24RD6Xn~Y830yZtgrIUclTgS5Yw47H3a4BnpZXC6fx7uHxsecML8eVaxosQFuGNcAx2-0SIZGUGJ091KUKzFSxhlWIwib2dXIDzjh9IvvSWOO03wBiOargVa-LyJiz6b670COUnymHTrKxywAAVympNRM5X6JLw\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

‌Bohora, K., Bothe, A., Sheth, D., & Chopade, R. (2021). Backup and Recovery Mechanisms of Cassandra Database: A Review. *The Journal of Digital Forensics, Security and Law*. https://doi.org/10.15394/jdfsl.2021.1613

Cloudflare, Inc. (2025). *What is asymmetric encryption?* Retrieved from https://www.cloudflare.com/learning/ssl/what-is-asymmetric-encryption/

‌ Gardiner, M. (2024, January 24). *Privilege escalation attack: How attackers gain unauthorized access*. Proofpoint. https://www.proofpoint.com/us/blog/identity-threat-defense/privilege-escalation-attack

Gupta, R. (2021, April 7). *An overview of Logon triggers in SQL Server*. SQL Shack - Articles About Database Auditing, Server Performance, Data Recovery, and More. https://www.sqlshack.com/an-overview-of-logon-triggers-in-sql-server/

Hussein, D., Ibrahim, D. M., & Alsalamah, M. (2021). A Review Study on SQL Injection Attacks, Prevention, and Detection. *Vol 13*(Number 3), pg 1-9. doi:10.22042/ISECURE.2021.0.0.0

IBM Corporation. (10 10, 2023). *Data definition language*. Retrieved from https://www.ibm.com/docs/en/i/7.3.0?topic=programming-data-definition-language

Kime, C. (16 May, 2023). *How to Prevent SQL Injection: 5 Key Prevention Methods*. Retrieved from https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/

Khadidos, A. O., S. Shitharth, Khadidos, A. O., K. Sangeetha, & Alyoubi, K. H. (2022). Healthcare Data Security Using IoT Sensors Based on Random Hashing Mechanism. *Journal of Sensors*, *2022*, 1–17. https://doi.org/10.1155/2022/8457116

Microsoft. (9 7, 2024). *Create a server audit and database audit specification.* Retrieved from https://learn.microsoft.com/en-us/sql/relational-databases/security/auditing/create-a-server-audit-and-database-audit-specification?view=sql-server-ver16

Microsoft. (n.d.). *Dynamic data masking*. Microsoft Learn. https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16

Nadee, P., & Somwang, P. (2021). Efficient incremental data backup of unison synchronize approach. *Bulletin of Electrical Engineering and Informatics*, *10*(5), 2707–2715. https://doi.org/10.11591/eei.v10i5.2212

OWASP Foundation. (2021). *A09:2021 – Security logging and monitoring failures*. OWASP. https://owasp.org/Top10/A09\_2021-Security\_Logging\_and\_Monitoring\_Failures/

Pan, W., Wang, M., Qin, J., & Zhou, Z. (2021). Improved CNN-Based Hashing for Encrypted Image Retrieval. *Security and Communication Networks*, *2021*, 1–8. https://doi.org/10.1155/2021/5556634

Richman, J. (2025, February 25). How to enable SQL Server change data Capture in 5 steps. *Estuary*. https://estuary.dev/blog/enable-sql-server-change-data-capture/

Todd, C. (24 Jan, 2025). *What You Need to Know About Data Protection in Healthcare*. Retrieved from https://www.hycu.com/blog/what-is-data-protection-healthcare

Vicente, V. (2024, May 10). *What is an audit trail? Everything you need to know*. https://auditboard.com/blog/what-is-an-audit-trail/

‌ Yaseen, A. (2019, November 20). *How to filter and block the data access using SQL Server 2016 Row-Level Security*. SQL Shack - Articles About Database Auditing, Server Performance, Data Recovery, and More. https://www.sqlshack.com/filter-block-data-access-using-sql-server-2016-row-level-security/

‌

# APPENDICES

## WORKLOAD MATRIX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Muhammad Nabil Hakim Bin Yusaidi**  **(TP066763)** | **Soo Jiun Guan (TP068687)** | **Tan Shi Ying (TP065227)** | **Teh Yue Feng**  **(TP069429)** |
| Classification Matrix | 25% | 25% | 25% | 25% |
| Authorization Matrix | 25% | 25% | 25% | 25% |
| User Matrix | 25% | 25% | 25% | 25% |
| Auditing Matrix | 25% | 25% | 25% | 25% |
| Data Protection | 25% | 25% | 25% | 25% |
| Permission Management | 25% | 25% | 25% | 25% |
| Auditing | 25% | 25% | 25% | 25% |
| Digital Signature | Nabil | Soo | shiying | yuefeng |