

**IE2042- Database Management Systems for Security- 2025**

**Assignment**

Year 2, Semester 1

**Table of Contents**

**PART 1**

1. ASSUMPTIONS.......................................................................................................................

1. CONCEPTUAL MODEL (EERD). .....................................................................................
2. LOGICAL MODEL (RELATIONAL MODEL). .............................................................................
3. NORMALIZATION. ............................................................................................................

5. IMPLEMENTING THE MODEL INTO THE MYSQ SERVER...............................................

* 1. *creating tables and enforcing constraints. ..............................................................*

* 1. *Sample data. ..........................................................................................................*

6. CONSTRAINTS IDENTIFIED AND ENFORCED. .............................................................

7. DEVELOP THE REQUIRED VIEWS, FUNCTIONS, PROCEDURES, TRIGGERS, AND INDEXES AS SPECIFIED BELOW. .........................

1. *Identify 2 suitable triggers that can be applied to the database and explain and implement them........*
2. *Identify the possible users of this database and create 2 views for them. .............................................*
3. *Based on the below questions identify 2 indexes that will optimize the given queries and implement them......*
4. *Write stored procedures to carry out the below DML functions. .............................................................*

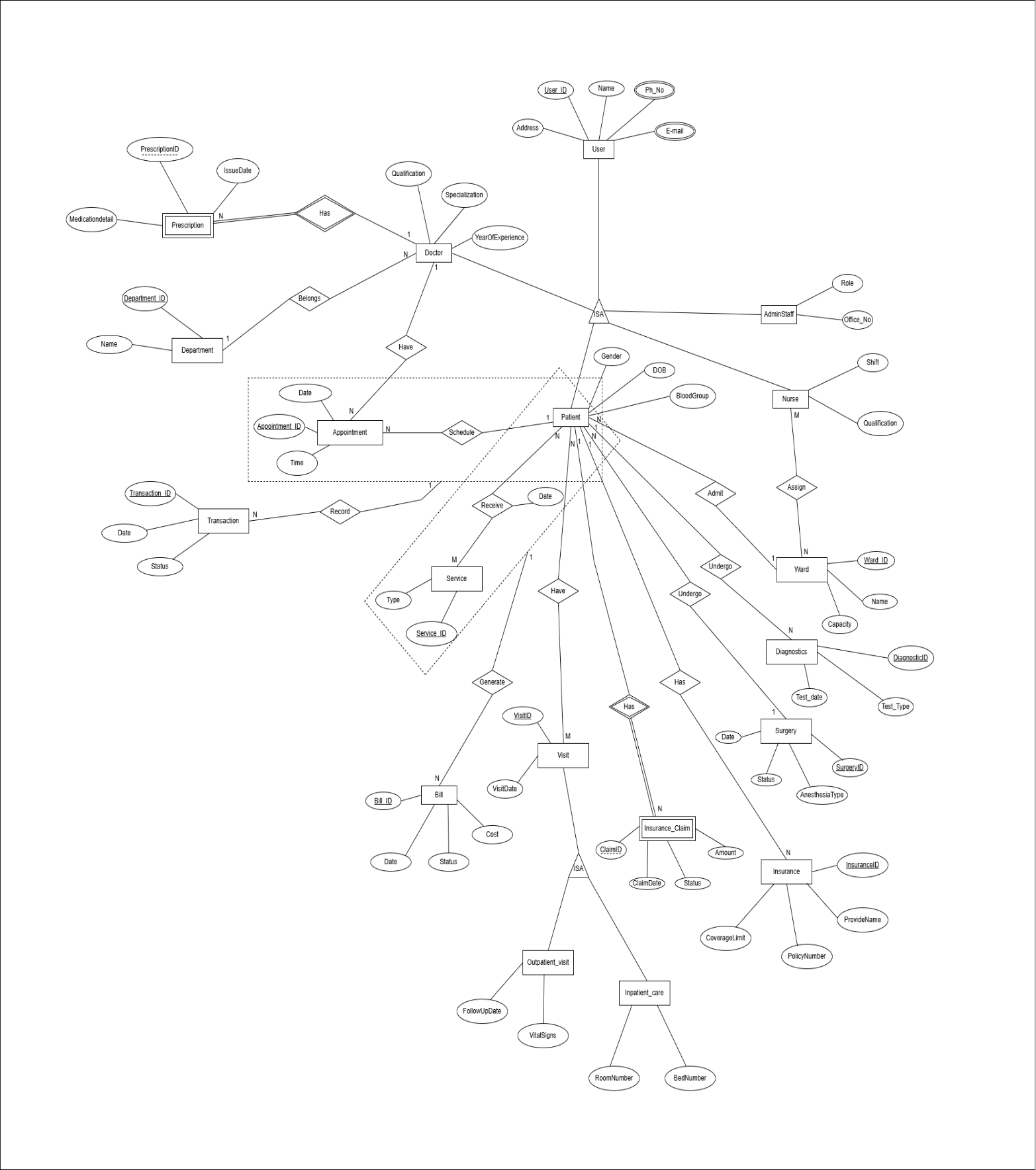
**PART 2**

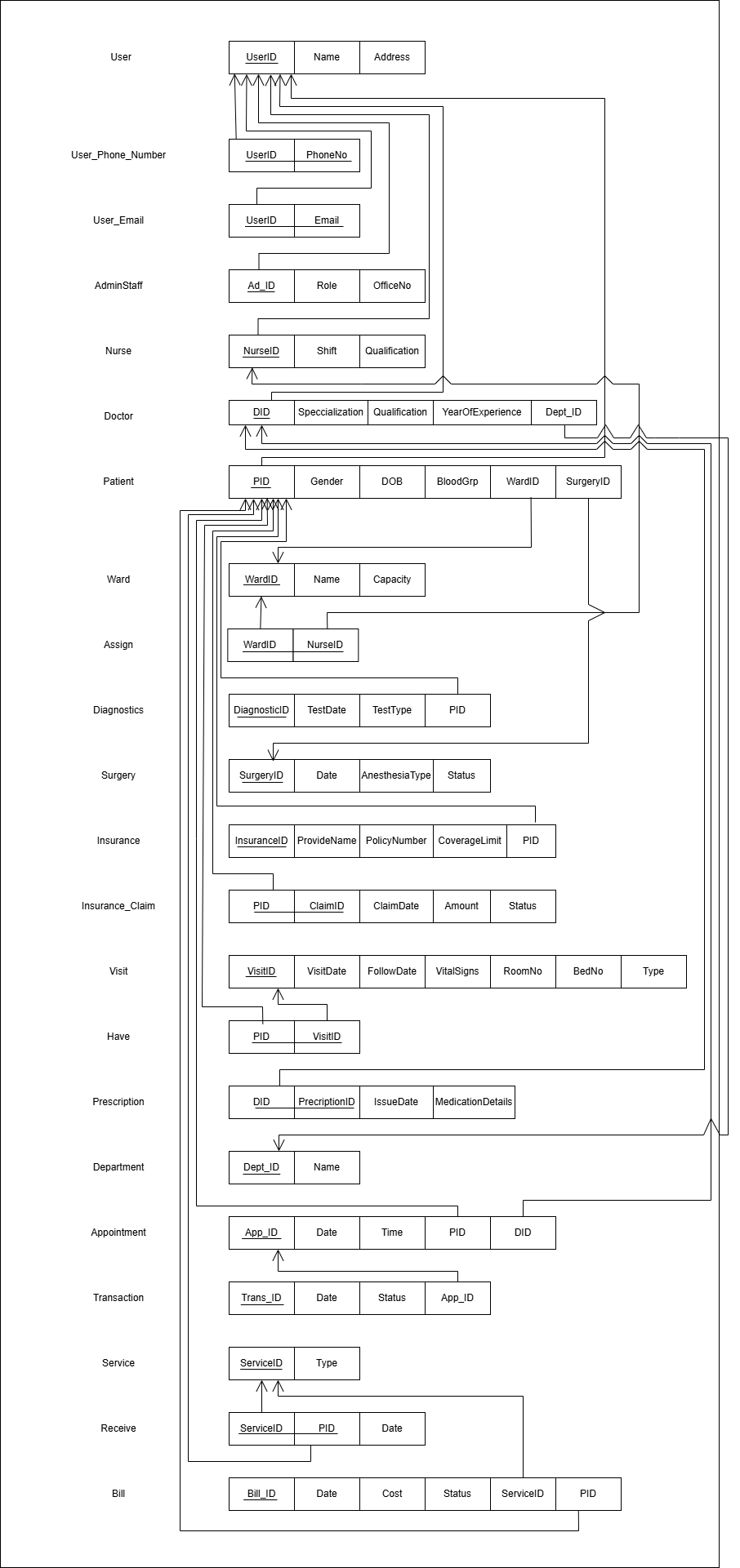
1. DESCRIPTION AND ANALYSIS OF 2 DATABASE VULNERABILITIES .....................................
2. MITIGATION TECHNIQUES AND COUNTERMEASURES SUGGESTIONS: .............................

**Part 1**

**1. Assumptions**

* + Each user (patient, staff member) has a unique UserID.
  + Users can have multiple phone numbers and email addresses (represented in separate tables).
  + Administrators have unique Ad\_IDs and are assigned roles and office numbers.
  + Nurses have unique NurseIDs, work shifts, and have qualifications.
  + Doctors have unique DoctorIDs (DD), specializations, qualifications, and years of experience.
  + Each patient has a unique PID (Patient ID).
  + Patient records include gender, date of birth, blood group, and are associated with wards and surgeries.
  + Each ward has a unique WardID, name, and capacity.
  + Nurses are assigned to wards through the Assign table.
  + Diagnostics have unique DiagnosticsIDs and are associated with patients, test dates, and test types.
  + Surgeries have unique SurgeryIDs and include date, anesthesia type, and status.
  + Each insurance policy has a unique InsuranceID and is linked to a patient.
  + Insurance claims have ClaimIDs and include claim date, amount, and status.
  + Each visit has a unique VisitID and includes visit date, follow-up date, vital signs, room number, bed number, and type.
  + Appointments have unique App\_IDs and link patients with doctors at specific dates and times.
  + Prescriptions have unique PrescriptionIDs and are issued by doctors to patients.
  + Services have unique ServiceIDs and are received by patients, generating bills.
  + Each bill has a unique Bill\_ID and is associated with a service, patient, date, cost, and status.
  + Departments have unique Dept\_IDs and names.
  + Doctors are associated with departments through Dept\_ID
  + A patient can have multiple visits, diagnostics, surgeries, insurance claims, and bills.
  + A doctor can have multiple appointments and prescriptions.
  + A nurse can be assigned to multiple wards.
  + A ward can have multiple nurses assigned to it.
  + All medical procedures (diagnostics, surgeries) and financial transactions (bills, insurance claims) are recorded with dates and statuses.
  + Patient vitals and visit details are recorded during each visit.

**2. Conceptual model (EERD)**

**3. Logical model (Relational model).**

**4. Normalization.**

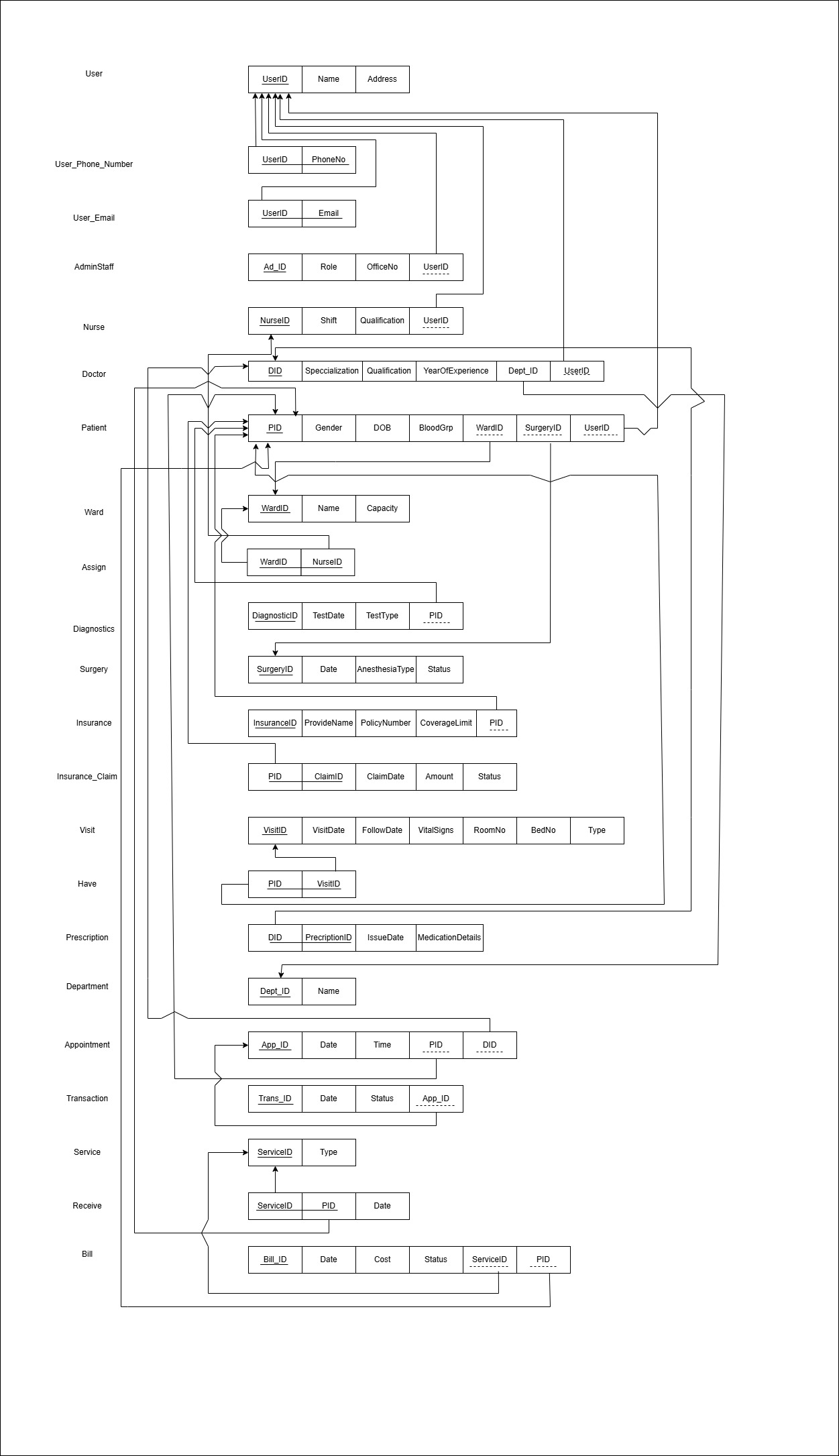
1. Non-atomic values

* There are non-atomic values in the Patient table.  
  For example, the ContactNumbers attribute may store multiple phone numbers in a single field, violating the rule that each field should store a single atomic value.

2. Transitive Functional Dependencies

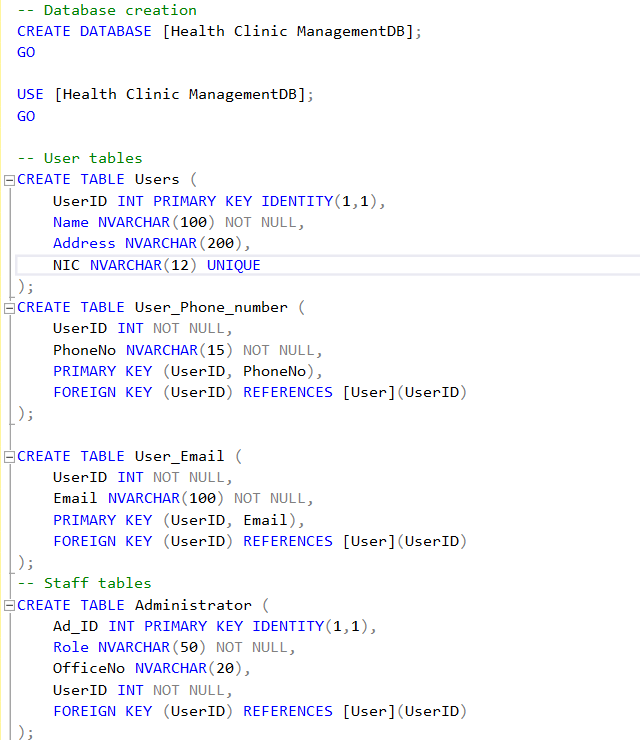
There are transitive functional dependencies in the following ways:

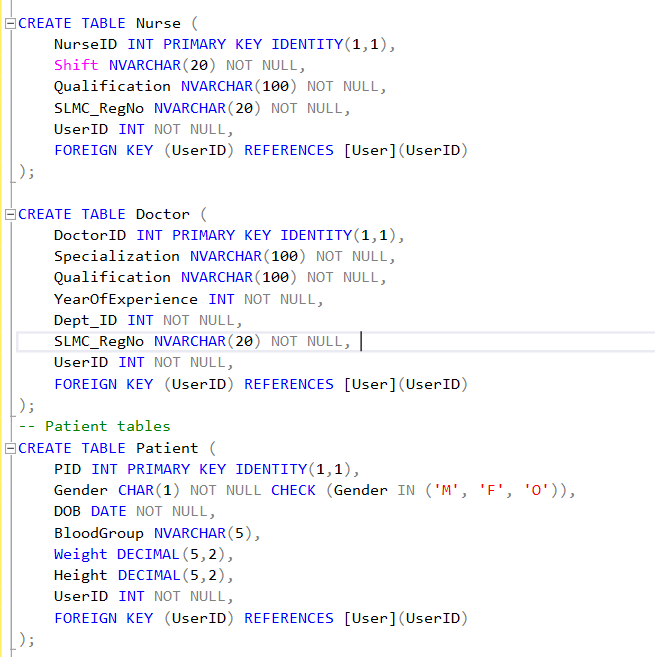
* In the Appointment table:
  + The attributes DoctorName and DoctorSpecialization are transitively dependent on the DoctorID foreign key,  
    which refers to the primary key in the Doctor table through a referential integrity constraint.
* In the Treatment table:
  + The attributes PatientName and PatientDOB are transitively dependent on the PatientID foreign key,  
    which refers to the primary key in the Patient table through a referential integrity constraint.

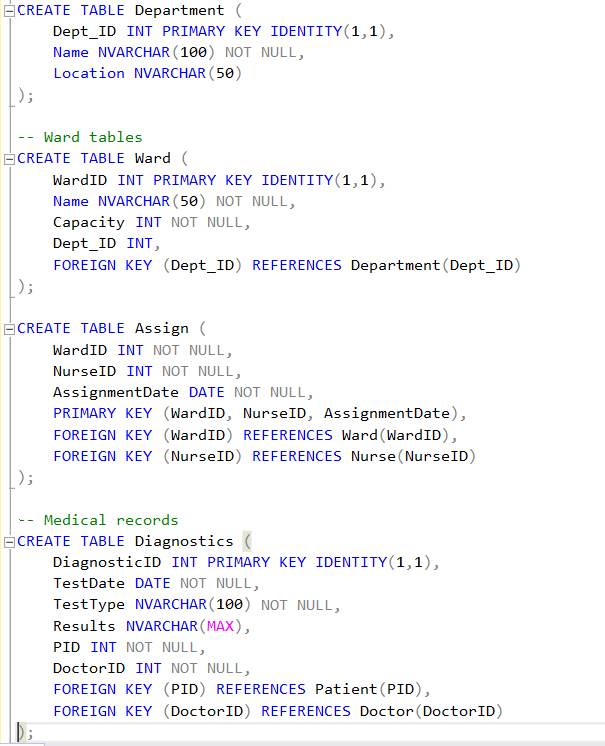


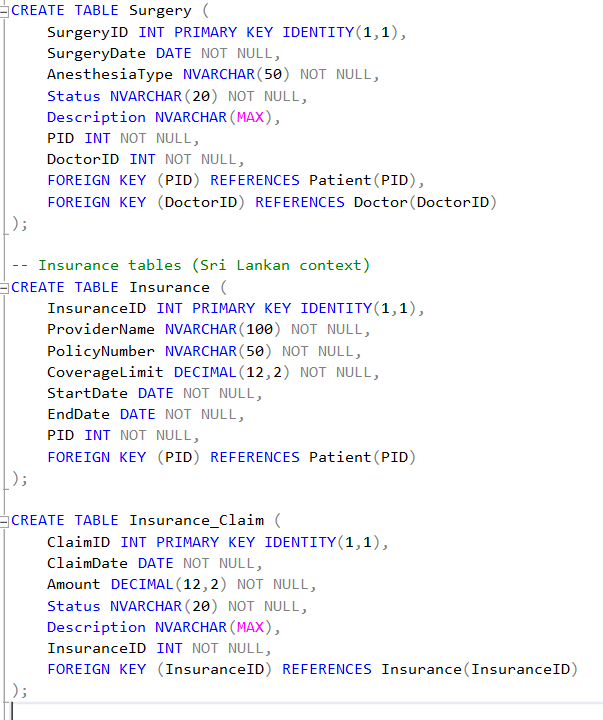
5. **Implementing the model into the MySQL server**

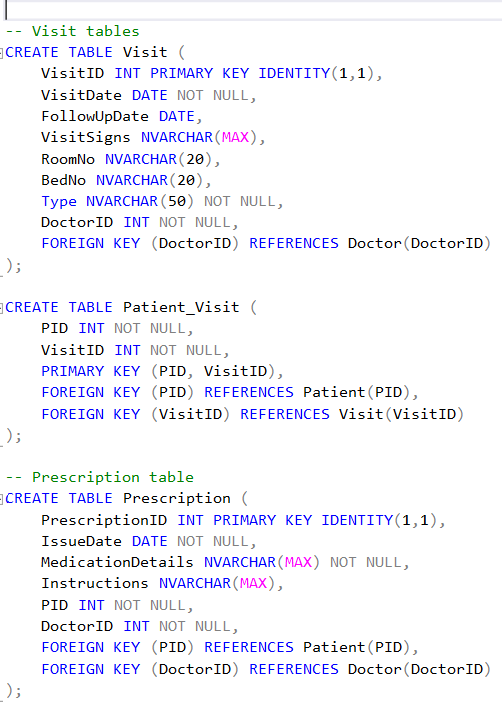
1. creating tables and enforcing constraints

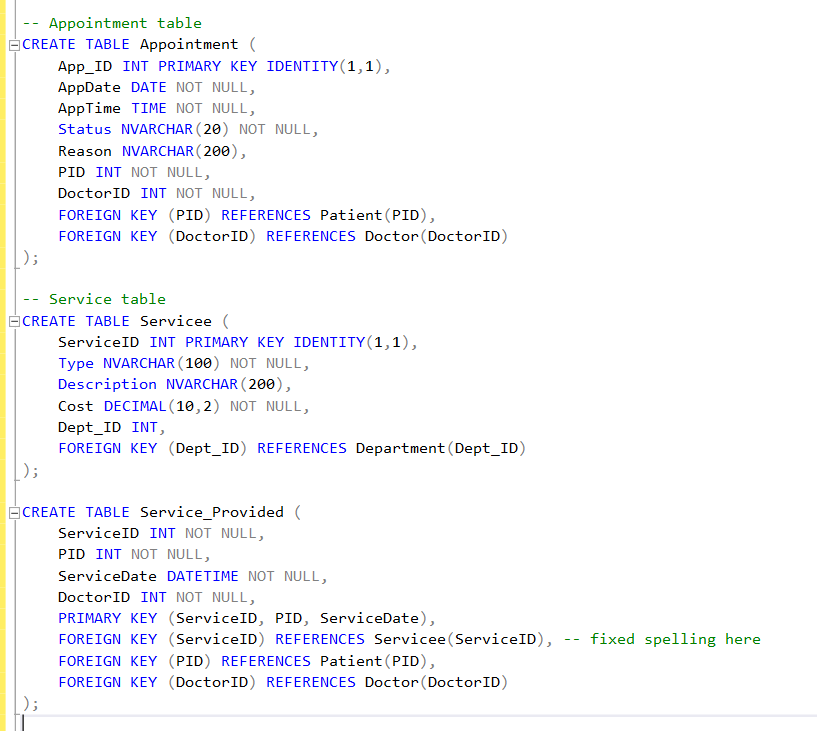
****

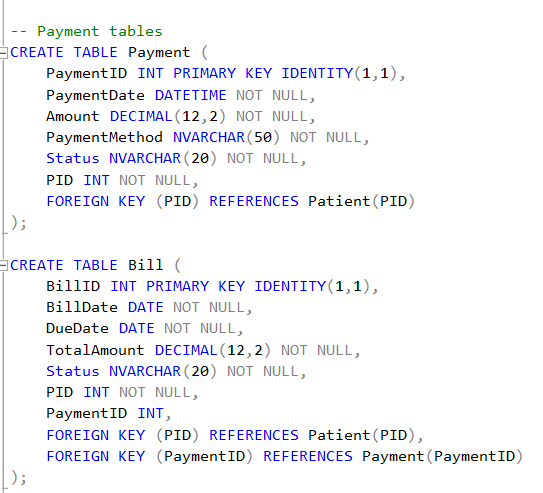
****

****

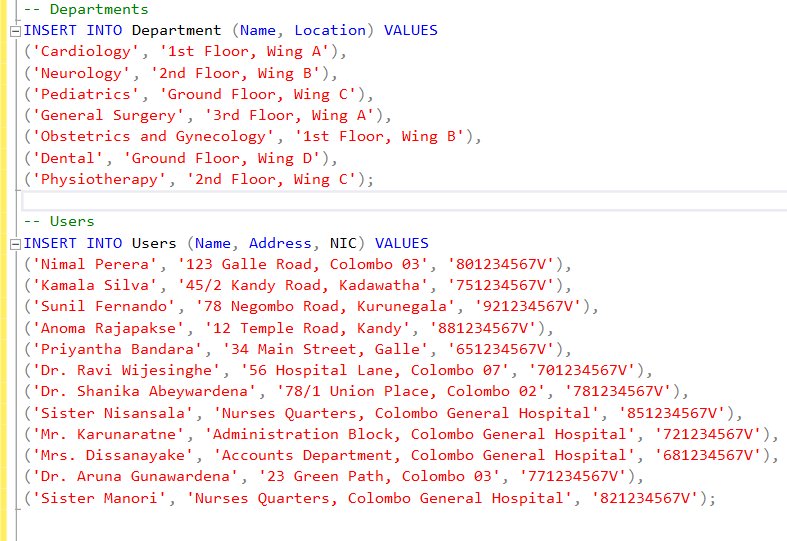
****

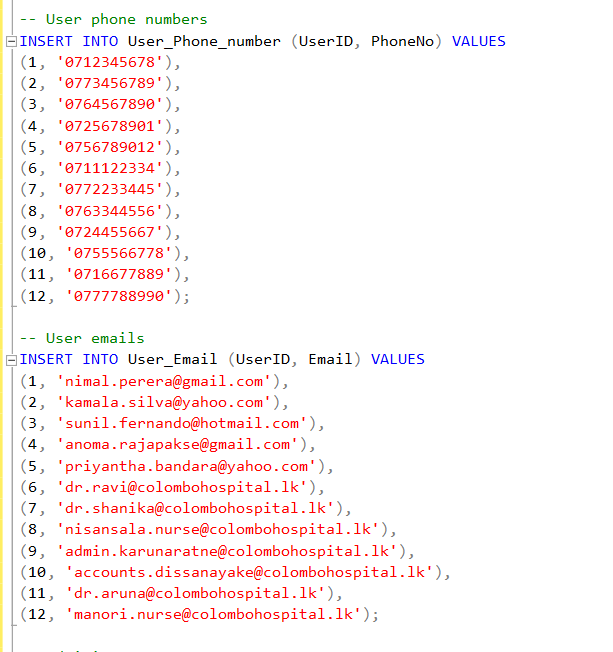
****

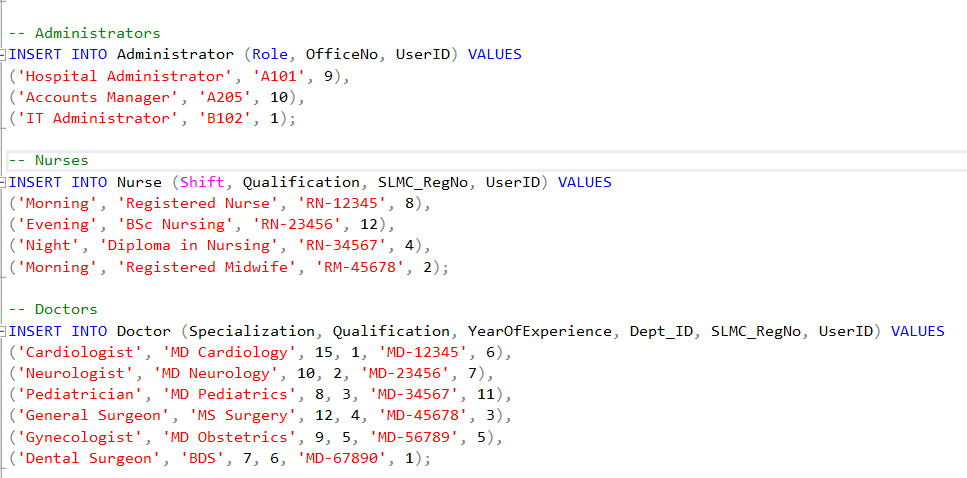


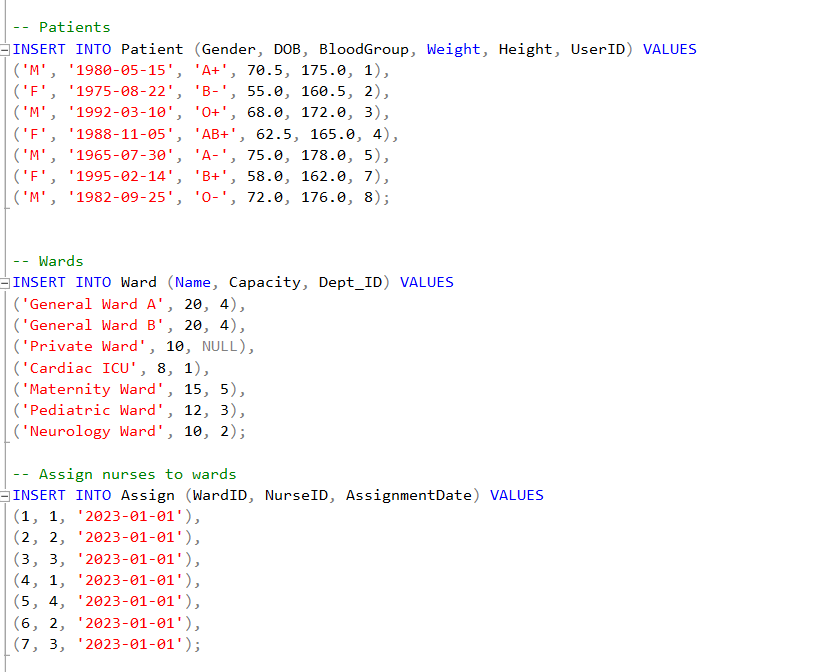


**2. Sample data**

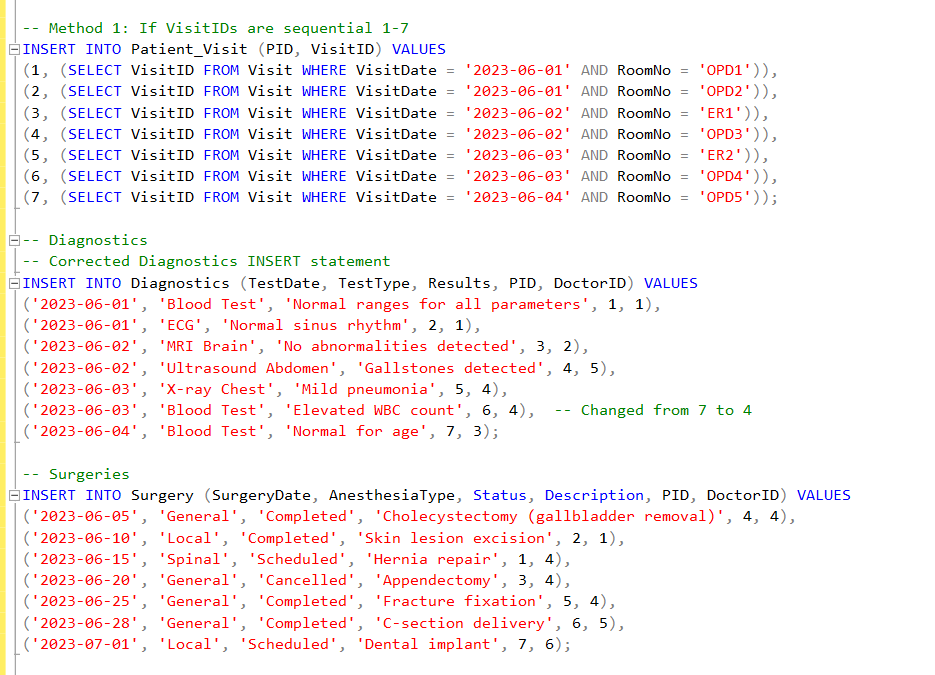


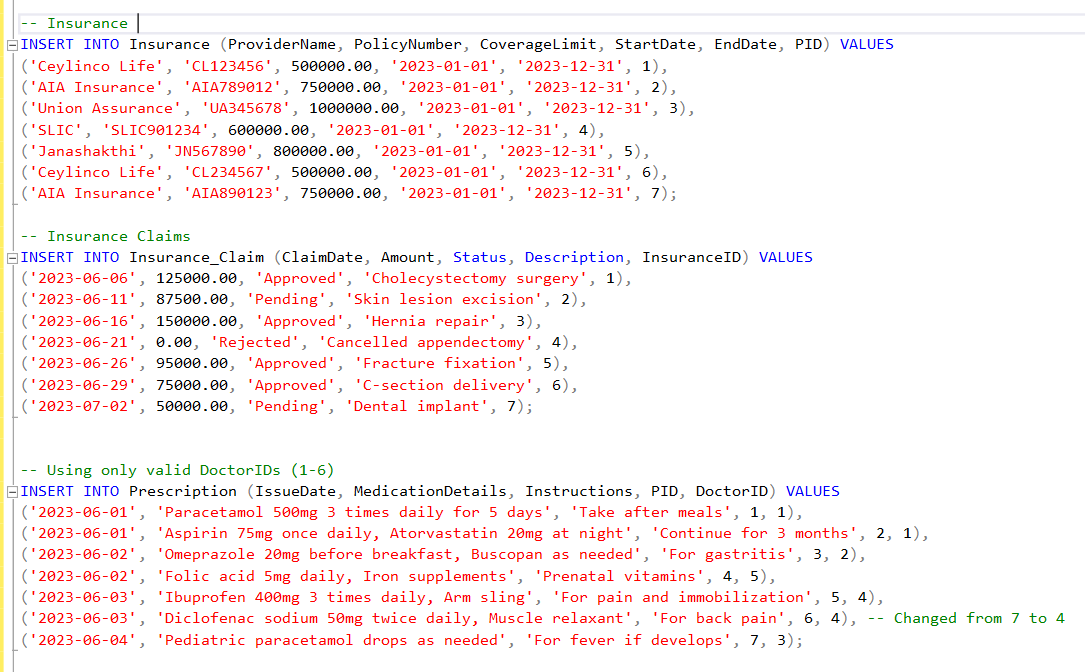


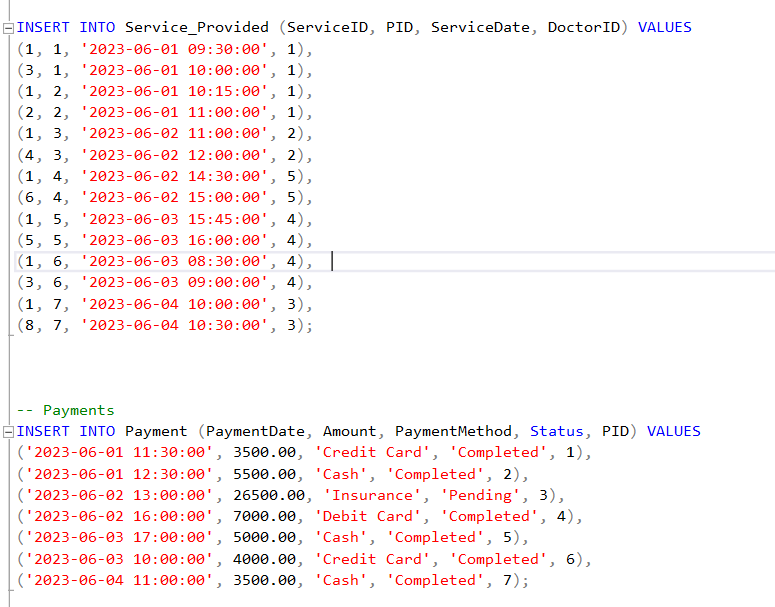


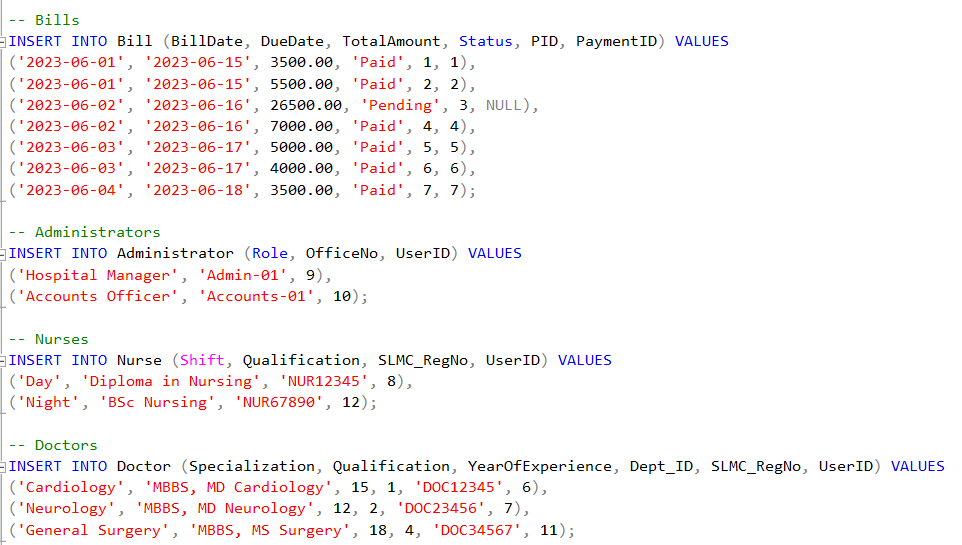


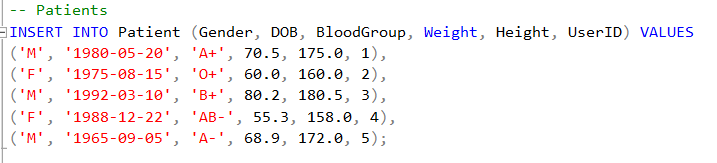












**6.Constraints identified and enforced.**

**Primary Key Constraint**

Ensures that **each record** in a table is **uniquely identifiable**.

**Example:**  
PRIMARY KEY constraint is applied to columns like:

* PatientID in the **Patient** table
* DoctorID in the **Doctor** table
* AppointmentID in the **Appointment** table
* TreatmentID in the **Treatment** table
* BillingID in the **Billing** table

**• Foreign Key Constraint**

Maintains **referential integrity** between **related tables**.

**Example:**  
FOREIGN KEY constraint is applied to columns like:

* PatientID in the **Appointment** table referencing PatientID in the **Patient** table
* DoctorID in the **Appointment** table referencing DoctorID in the **Doctor** table
* AppointmentID in the **Treatment** table referencing AppointmentID in the **Appointment** table
* PatientID in the **Billing** table referencing PatientID in the **Patient** table

**• Unique Constraint**

Ensures that **values in a column** or a **group of columns** are **unique across the table**.

**Example:**  
UNIQUE constraint is applied to:

* Email column in the **Patient** table
* LicenseNumber column in the **Doctor** table (each doctor’s license number must be unique)

**• Not Null Constraint**

Ensures that a **column cannot contain a NULL value**.

**Example:**  
NOT NULL constraint is applied to:

* PatientName in the **Patient** table
* DoctorName in the **Doctor** table
* AppointmentDate in the **Appointment** table
* BillingAmount in the **Billing** table

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

**3.Identify 2 suitable triggers that can be applied to the database and explain and implement them.**

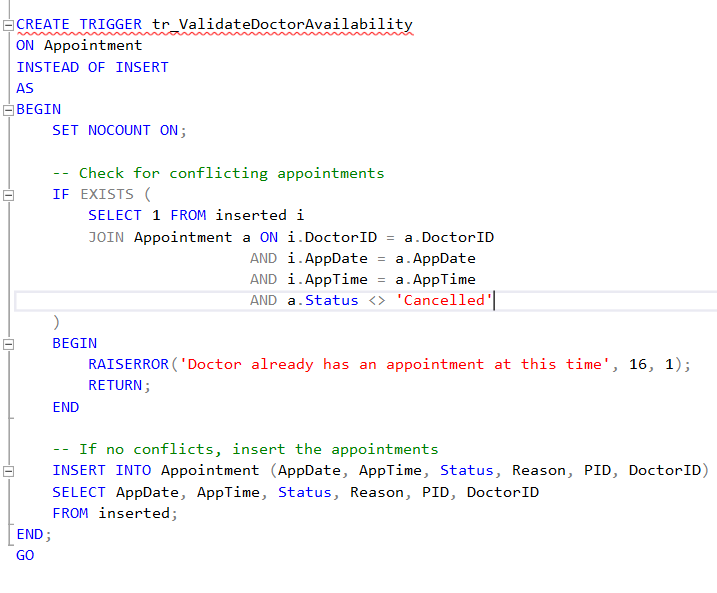
**Trigger 1: Validate Doctor Availability Trigger**

**Explanation:**  
This trigger ensures that **when a new appointment is inserted into the "Appointment" table**, the system first checks if the **Doctor** already has an **existing appointment** scheduled at the **same date and time** (that is **not cancelled**).

* If a conflicting appointment is found, the trigger **raises an error** and **prevents** the new appointment from being inserted.
* If no conflict is detected, the new appointment is **successfully inserted** into the Appointment table.

**Purpose:**

* Prevent **double-booking** of doctors.
* Maintain **accurate appointment scheduling** without manual checking.



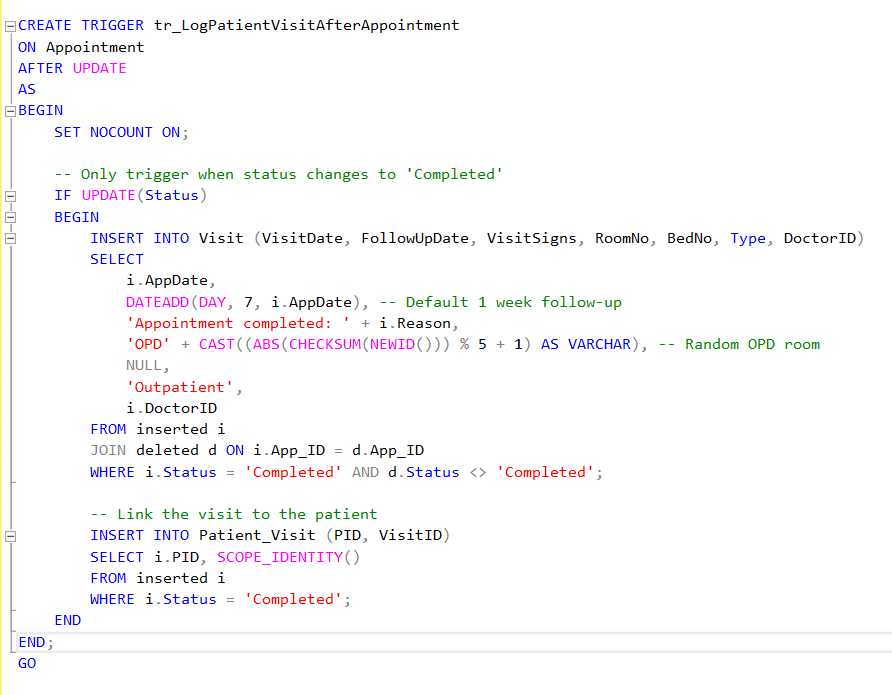
**Trigger 2: Log Patient Visit After Appointment Trigger**

**Explanation:**  
This trigger ensures that **after an appointment’s status is updated to "Completed"** in the Appointment table, a corresponding **patient visit record** is automatically created in the Visit and Patient\_Visit tables.

* It **inserts a new visit record** with relevant details like visit date, follow-up date (one week later by default), visit signs, room number, and doctor.
* It **links the patient** to the newly created visit.
* The trigger **only fires** when the appointment's Status **changes to "Completed"**.

**Purpose:**

* Automates the **recording of patient visits** after completed appointments.
* Ensures **consistency** and **integrity** between appointments and visits.



**4. Identify the possible users of this database and create 2 views for them.**

**User 1: - Doctor**

Creating a View for Doctors to See Patient Appointments and Treatment Details

Explanation:  
This view, vw\_DoctorPatientDetails, provides doctors with a complete overview of their patients, appointments, and treatment history.

* Displays doctor details (ID, name, specialization).
* Displays patient details (ID, name, blood group).
* Shows appointment information (date, time, status, reason).
* Shows visit records (visit ID, date, signs recorded).
* Shows prescription details (prescription ID, issue date, medication information).

Purpose:

* Allows doctors to easily access all necessary information about their patients' appointments and treatments in one place.
* Helps doctors to track medical histories, follow up with patients, and manage treatment plans effectively.

**User 2: - Administrative Staff**

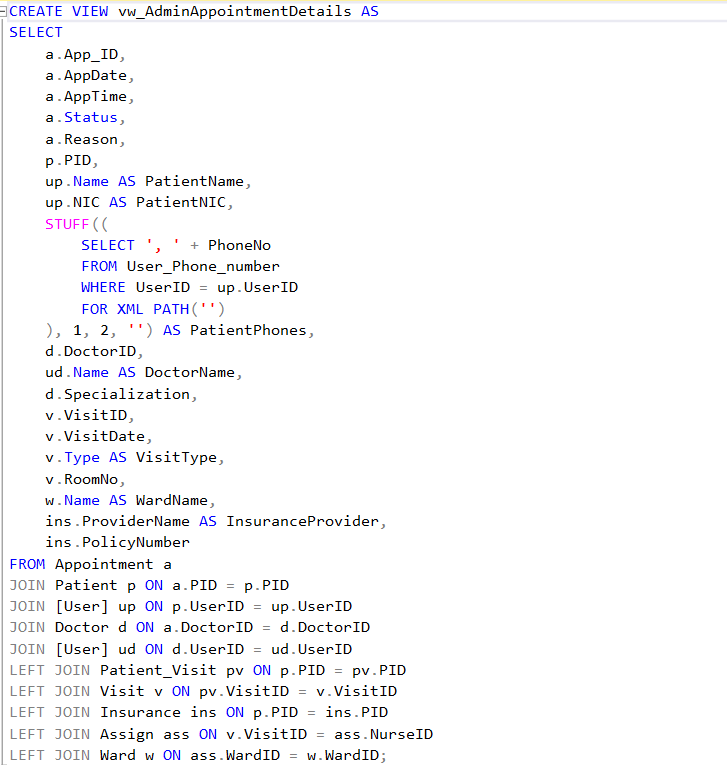
**Creating a View for Administrative Staff to Manage Appointments and Patient Details**

**Explanation:**  
This view, vw\_AdminAppointmentDetails, provides administrative staff with **comprehensive information** about patient appointments, assigned doctors, visit records, and insurance details.

* Displays **appointment details** (ID, date, time, status, reason).
* Displays **patient information** (ID, name, NIC, phone numbers).
* Displays **doctor information** (ID, name, specialization).
* Shows **visit information** (visit ID, date, type, room number, ward name if assigned).
* Shows **insurance details** (provider name, policy number).

**Purpose:**

* Allows administrative staff to **efficiently monitor** and **manage appointments**.
* Helps in **coordinating between patients, doctors, wards, and insurance providers**.
* Improves hospital operations by giving a **centralized view** of all important appointment and patient-related information.



**5.Based on the below questions identify 2 indexes that will optimize the given queries and implement them.**

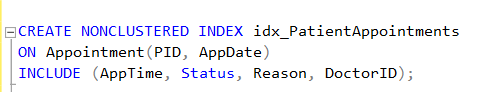
**Index 1: Patient Appointment Index**

**Explanation:**  
As we need to **retrieve the details of all appointments for a given patient within a given period**, we can create a **non-clustered index** on the PID (Patient ID) and AppDate columns of the **Appointment** table.

* This index helps **quickly locate** appointments for a specific patient across a specific time range.
* We **include** other important columns (AppTime, Status, Reason, DoctorID) in the index, so they can be retrieved without an extra lookup (**covering index**).

**Purpose:**

* Boosts query performance by avoiding full table scans on large **Appointment** tables.
* Reduces lookup times significantly when **filtering by patient and appointment date**.



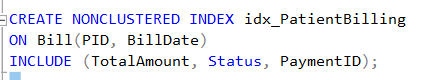
**Index 2: Patient Billing Index**

**Explanation:**  
As we need to **retrieve billing records for a patient based on the billing date**, we can create a **non-clustered index** on the PID (Patient ID) and BillDate columns of the **Bill** table.

* This index will **quickly locate** all bills issued to a particular patient within a certain time frame.
* We **include** other important billing columns (TotalAmount, Status, PaymentID) in the index, allowing queries to retrieve all needed information without extra lookups (**covering index**)

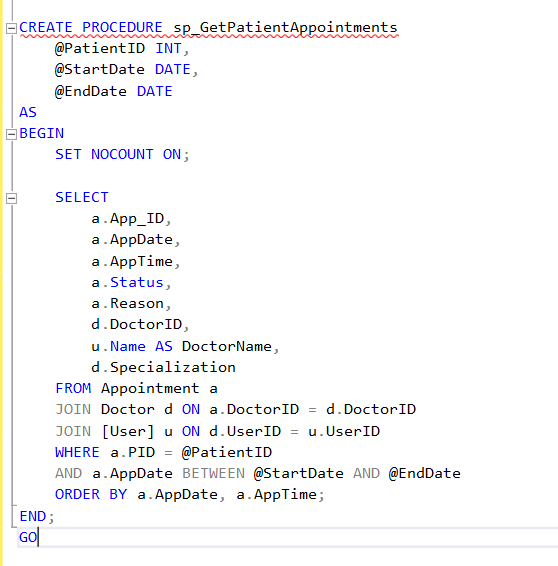
**Purpose:**

* Improves performance by enabling fast lookups of **billing records per patient**.
* Helps avoid full table scans when filtering or sorting by patient and billing date.

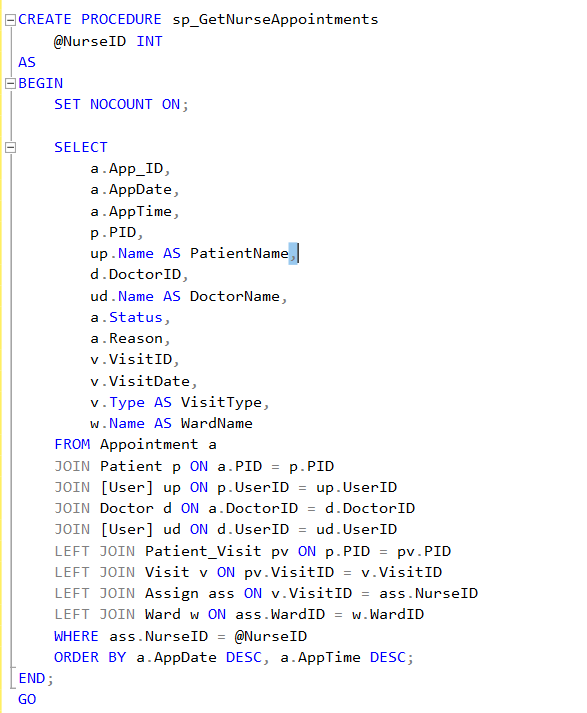


**6.Write stored procedures to carry out the below DML functions.**

The stored procedure retrieves a list of patient appointments for a given patient within a specified date range**.**

****

Retrieve the details of all appointments assigned to a specific nurse, including the appointment ID, date, time, patient information (ID and name), doctor information (ID and name), appointment status, reason, visit details (ID, date, and type), and the ward name, ordered by appointment date and time in descending order.



**Part 2**

**8.Description and analysis of 2 database vulnerabilities**

**SQL Injection (SQLi)**

**Techniques:**

SQL injection is a critical web application vulnerability that occurs when SQL statements contain unsanitary or invalid user input. Attackers exfiltrate data by injecting malicious SQL commands using input field, URL parameters, or other data entry methods. SQL strings commonly used for SQL attacks contains SQL keywords, which are SELECT , UNION , INSERT , UPDATE , DELETE and logical operations , which can be used to easily operations, which can be used to easily manipulate queries in databases. These attacks exploit input formats from or insecure URL types, resulting in application logic failure to distinguish between SQL commands, user, data and SQL

**Impact:**

A successful SQL injection attack can be lead to unauthorized access to sensitive information such as user's proprietary information or user credentials. Attackers have the ability to modify or delete data after accessing the database. Attackers can execute system - level commands through SQL injection attackers in extreme cases, compromising the entire server. Such breaches can cause organizations significant reputational damage, financial losses, legal liabilities and regulatory non- compliance ,beyond technical harm.

**Cross-Site Scripting (XSS)**

**Techniques:**

Cross-Site Scripting (XSS) is a security risk that occurs when an application is included in its output without proper validation or completion of unreliable data attempts. Disablers hang a corrupt JavaScript code on the website and the same code runs in other users' browsers. XSS attacks are typically classified as reflected, stored, or DOM-based. Reflected XSS is when a malicious script has an instant effect through a URL parameter, Stored XSS, meanwhile, is meant to show users a corrupt script while preserving the database stable. Attacks primarily target intolerant input fields, comment sections, or message boards.

**Impact:**

XSS vulnerabilities can be exploited by employees to steal cookies, hijack user sessions, capture keystrokes, perform phishing attacks, and send users to malicious websites. Short beforehand, disablers may even distort the website or perform activities that are not permitted in the user's name, thereby breaking user trust and integrity of the app. Such risks pose a serious threat to the confidentiality and integrity of user data.

**Remote Code Execution (RCE)**

**Techniques:**

RCE (Remote Code Execution) risks occur when an application does not properly complete the input information provided by the user. This allows disabled workers to run arbitrary code on a remote server.

The reasons for such risks can be:

- input validation vulnerabilities (input validation flaws)

- Unsafe deserialization

- Vulnerabilities in the lateral parts of the server

Workers often take advantage of such risks by sending specifically processed input, explaining and executing those inputs as commands by the server. This also allows them to control system processes.

**Impact:**

Disablers have the ability to install malware, read or modify sensitive data, increase power, and construct activities such as encrypting secure data on the network.

RCE, as it has features that affect the overall IT infrastructure, results in service threats and uploading data (data breaches), and a number as one of the most relevant threats.

**9. Mitigation Techniques and Countermeasure Suggestions**

**• SQL Injection**

**Input Validation:**

Strict validation rules should be applied to all input fields and URL parameters. Only pre-defined data types and models should be accepted.

**Parameterized Queries:**

In order to prevent malicious SQL entries, the user should use parameterized queries or prepared statements to separate the SQL code from the input.

**Least Privilege Principle:**

Only minimal database privileges should be provided for users and apps. Do not use administrator accounts for day-to-day operations.

**Escaping Special Characters:**

Minimum necessary database privileges should be assigned to the users and applications.

**Web Application Firewall (WAF):**

Installing WAF (Web Application Firewalls) to prevent malicious SQL injection attacks and prevent such an attack from being accessed by the app prior to being halted.

**Regular Security Audits:**

Security testing and vulnerability scanning are performed on a regular basis, and SQL injection vulnerabilities are found and resolved.

**Error Handling:**

Create error messages to prevent displaying data storage structures. Prevent users from displaying sensitive information.

**Input Sanitization:**

Sanitize all user input, benefits, cookies, hideovers, and URL parameters to filter out as much risky characters as possible

**Patch and Update:**

Keep the Database Management System (DBMS) and application frameworks updated with the latest security patches.

**Education and Training:**

Provide administrators and engineers security code writing techniques and skills training to avoid SQL injections.

**Database Configuration:**

Secure databases by disabling activation of unnecessary facilities and services offered to employees.