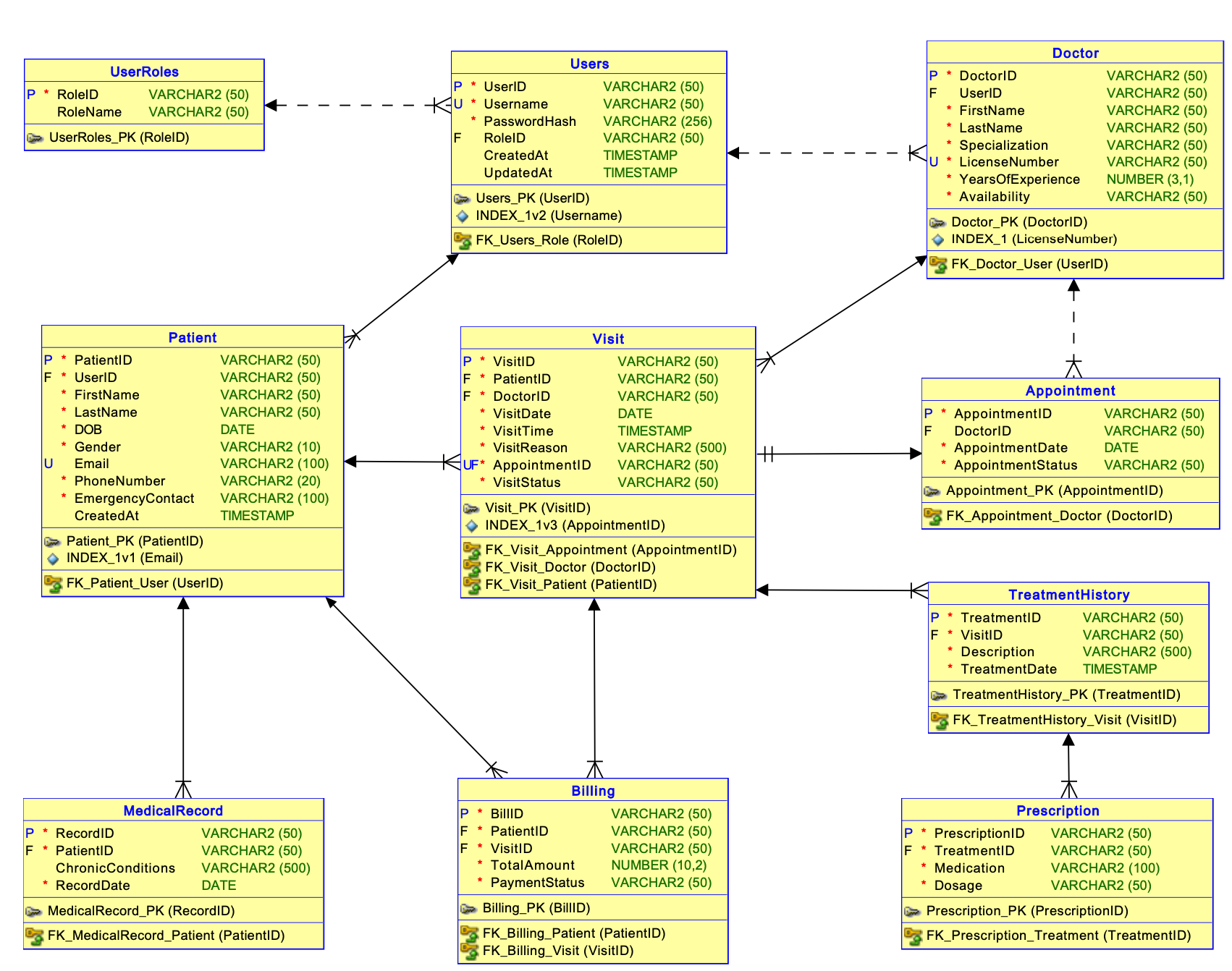
**Phase-2 Documentation for H.E.A.L. (Healthcare Efficiency & Assistance Log)**

**1. H.E.A.L ERD:-**



**2. Normalization Steps:-**

**Unnormalized Form (UNF)**

Before applying normalization, the database may contain repeating groups and redundant data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PatientID | PatientName | DoctorID | DoctorName | VisitDate | TreatmentDetails |
| P001 | John Doe | D001 | Dr. Smith | 2024-03-01 | Flu Medication |
| P001 | John Doe | D001 | Dr. Smith | 2024-03-01 | Fever Treatment |

**Issue**: Repeating groups (TreatmentDetails) make it not in 1NF.

**First Normal Form (1NF) - Eliminating Repeating Groups**

A relation is in 1NF if:

* All attributes contain atomic (indivisible) values.
* Each column contains only one value per row.
* Each row has a unique identifier (Primary Key).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PatientID | PatientName | DoctorID | DoctorName | VisitDate | TreatmentID | TreatmentDetails |
| P001 | John Doe | D001 | Dr. Smith | 2024-03-01 | T001 | Flu Medication |
| P001 | John Doe | D001 | Dr. Smith | 2024-03-01 | T002 | Fever Treatment |

**Now in 1NF**: No repeating groups.

**Issue**: **Partial dependency** (DoctorName depends only on DoctorID, not on the full primary key).

**Second Normal Form (2NF) - Removing Partial Dependencies**

A relation is in 2NF if:

* It is already in 1NF.
* All non-key attributes are fully dependent on the entire Primary Key.

**New Tables:**

**Patients Table**

|  |  |
| --- | --- |
| PatientID | PatientName |
| P001 | John Doe |

**Doctors Table**

|  |  |
| --- | --- |
| DoctorID | DoctorName |
| D001 | Dr. Smith |

**Visits Table**

|  |  |  |  |
| --- | --- | --- | --- |
| VisitID | PatientID | DoctorID | VisitDate |
| V001 | P001 | D001 | 2024-03-01 |

**Treatments Table**

|  |  |  |
| --- | --- | --- |
| TreatmentID | VisitID | TreatmentDetails |
| T001 | V001 | Flu Medication |
| T002 | V001 | Fever Treatment |

**Now in 2NF**: Each attribute fully depends on its primary key.

**Issue**: DoctorName is dependent on DoctorID but is not related to the Visit directly.

**Third Normal Form (3NF) - Removing Transitive Dependencies**

A relation is in 3NF if:

* It is already in 2NF.
* There are no transitive dependencies (i.e., non-key attributes must not depend on another non-key attribute).

**Doctors Table Updated**

|  |  |
| --- | --- |
| DoctorID | DoctorName |
| D001 | Dr. Smith |

**Now in 3NF**: No transitive dependencies.

**Boyce-Codd Normal Form (BCNF) - Handling Remaining Anomalies**

A relation is in BCNF if:

* It is already in 3NF.
* Every determinant is a candidate key.

**Steps Taken:**

* Ensured DoctorID is a candidate key in the Doctor table by enforcing unique constraints on LicenseNumber.
* In Users, ensured that Username is unique to prevent redundancy and anomalies.
* In Appointments, prevented scheduling anomalies by enforcing unique constraints on DoctorID and AppointmentDate to avoid duplicate entries.

**3. Business Rules & Constraints:-**

To ensure data integrity and business logic enforcement, the following rules are defined:

1. **General Validations**
   * Email, First Name, and Last Name cannot be blank.
   * Phone Number must follow the standard format (e.g., +1-123-456-7890).
   * Patient cannot have multiple active visits at the same time.
   * Role-based access control (RBAC) restricts user permissions.
2. **Medical & Appointment Rules**
   * Doctors can only have one appointment per patient per visit date.
   * Appointments cannot be scheduled in the past.
   * Doctors must be available on the scheduled appointment date.
3. **Billing & Insurance Rules**
   * Billing must be generated for every completed visit.
   * Insurance claims should be linked to a valid insurance provider.
   * Discounts must be applied before tax calculation.
4. **Data Integrity & Auto-Processing Rules**
   * Auto order placement when medication stock reaches the reorder threshold.
   * Treatment details cannot be added without a valid visit record.
   * Prescriptions must be linked to a valid treatment record.

**4. Views for Data Analysis:-**

To facilitate efficient data retrieval and reporting, the system incorporates the following database views:

1️. **Doctor Availability** – Provides a quick overview of doctor details, including their specialization and availability status, aiding in appointment scheduling.

2️. **Patient Visit Summary** – Displays a consolidated view of patient visits, including the doctor’s name, visit date, reason, and status, ensuring seamless tracking of medical records.

3️. **Billing Insights** – Summarizes billing transactions, linking patients, visit dates, and total charges with payment status, supporting financial monitoring and reconciliation.

**5. DFD (DATA FLOW) Level 0: Context Diagram:-**

**Overview:**  
The Context Diagram presents the entire H.E.A.L. system as a single process, showcasing interactions with external entities.

**Entities & Interactions:**

* **Patient** – Requests appointments, submits details, and receives billing.
* **Doctor** – Schedules appointments and updates medical records.
* **Admin** – Manages user roles and system access.
* **Billing System** – Handles payments and insurance claims.

**Data Stores:**

* **Patient Database**
* **Doctor Database**
* **Billing Database**
* **Appointment Records**

**Core Processes:**

1. **Manage Appointments**
2. **Handle Patient Records**
3. **Process Billing & Insurance**
4. **Maintain Doctor Availability**

**DFD Level 1: Major Modules**

This level breaks down the core functionalities into detailed processes.

**1️. Patient Registration & Management**

**Actors:** Patient, Admin  
**Processes:**

* **New Patient Registration** – Captures and stores patient data.
* **User Authentication** – Validates login credentials.
* **Manage Patient Profile** – Updates patient details.

**Data Flow:**

* **Patient submits details** → **Data stored in Patient Database**
* **Admin reviews/approves** → **Updated in the system**

**2️. Appointment Scheduling**

**Actors:** Patient, Doctor, System Scheduler  
**Processes:**

* **Schedule Appointment** – Matches doctor availability.
* **Doctor Confirms** – Approves or rejects the request.
* **Update Appointment Status** – Stores confirmation or cancellation.

**Data Flow:**

* **Patient requests appointment** → **System checks availability** → **Doctor confirms/rejects** → **Status saved in Appointment Database**

**3️. Medical Records & Treatment History**

**Actors:** Doctor, Patient  
**Processes:**

* **Record Patient Visit** – Logs visit details.
* **Update Medical Records** – Adds diagnosis, prescriptions, and notes.
* **Retrieve Medical History** – Provides past treatment details.

**Data Flow:**

* **Doctor updates records** → **Data stored in Medical Records Database**
* **Patient views history** → **Data fetched from system**

**4️. Billing & Insurance Processing**

**Actors:** Patient, Admin, Insurance Provider  
**Processes:**

* **Generate Bill** – Calculates total costs.
* **Apply Insurance** – Validates and processes claims.
* **Payment Processing** – Marks payments as Paid or Pending.

**Data Flow:**

* **Patient receives bill** → **Insurance claim processed** → **Payment recorded in Billing Database**

**6. User Creation & Grants:-**

To ensure **security and controlled access** within the system, **role-based access control (RBAC)** is implemented. Different user roles are created with specific permissions to maintain **data integrity and operational efficiency.**

## **1️.User Creation**

The system defines three **primary user roles** with different access levels:

* **Admin User** → Has full access, including **user and database management.**
* **Doctor User** → Can access **patient records, appointments, and medical records**, but cannot manage users or billing.
* **Billing User** → Can access **billing and payment data**, but does not have access to medical records.

Each user is created with **a strong password policy** to meet security requirements.

## **2️. Assigning Permissions**

**Admin - Full Access**  
The **Admin User** has **complete control** over the database, including the ability to create, modify, and manage users and system configurations.

**Doctor - Limited Access (Patient & Medical Records Only)**  
The **Doctor User** can access **patient information, appointments, and medical records**, ensuring they **can update and retrieve** medical data but **cannot modify user or billing data.**

**Billing Staff - Limited Access (Billing & Payments Only)**  
The **Billing User** can manage **billing and payments** but **does not have access to medical records**, ensuring **privacy and data separation**.

## **3️. Restricting User Privileges**

To **enhance security and prevent unauthorized actions, doctors and billing staff are restricted from performing critical modifications**, such as dropping tables.

Additionally, **automated privilege revocation** ensures that users **do not retain permissions that were never assigned**, reducing security risks.

**7. DDL SCRIPT:-**

-- Drop existing tables if they exist to ensure re-execution does not fail

DROP TABLE Users CASCADE CONSTRAINTS;

DROP TABLE UserRoles CASCADE CONSTRAINTS;

DROP TABLE Doctor CASCADE CONSTRAINTS;

DROP TABLE Patient CASCADE CONSTRAINTS;

DROP TABLE Appointment CASCADE CONSTRAINTS;

DROP TABLE Visit CASCADE CONSTRAINTS;

DROP TABLE MedicalRecord CASCADE CONSTRAINTS;

DROP TABLE TreatmentHistory CASCADE CONSTRAINTS;

DROP TABLE Prescription CASCADE CONSTRAINTS;

DROP TABLE Billing CASCADE CONSTRAINTS;

-- UserRoles Table

CREATE TABLE UserRoles (

RoleID VARCHAR2(50) PRIMARY KEY,

RoleName VARCHAR2(50) CHECK (RoleName IN ('Admin', 'Doctor', 'BillingStaff'))

);

-- Users Table

CREATE TABLE Users (

UserID VARCHAR2(50) PRIMARY KEY,

Username VARCHAR2(50) UNIQUE NOT NULL,

PasswordHash VARCHAR2(256) NOT NULL,

RoleID VARCHAR2(50),

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UpdatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_Users\_Role FOREIGN KEY (RoleID) REFERENCES UserRoles(RoleID)

);

-- Doctor Table

CREATE TABLE Doctor (

DoctorID VARCHAR2(50) PRIMARY KEY,

UserID VARCHAR2(50) UNIQUE NOT NULL,

FirstName VARCHAR2(50) NOT NULL,

LastName VARCHAR2(50) NOT NULL,

Specialization VARCHAR2(50) NOT NULL,

LicenseNumber VARCHAR2(50) UNIQUE NOT NULL,

YearsOfExperience NUMBER(3,1) CHECK (YearsOfExperience >= 0),

Availability VARCHAR2(50) NOT NULL,

CONSTRAINT FK\_Doctor\_User FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

-- Patient Table

CREATE TABLE Patient (

PatientID VARCHAR2(50) PRIMARY KEY,

UserID VARCHAR2(50) UNIQUE NOT NULL,

FirstName VARCHAR2(50) NOT NULL,

LastName VARCHAR2(50) NOT NULL,

DOB DATE NOT NULL,

Gender VARCHAR2(10) CHECK (Gender IN ('Male', 'Female', 'Other')),

Email VARCHAR2(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR2(20) NOT NULL,

EmergencyContact VARCHAR2(100) NOT NULL,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_Patient\_User FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

-- Appointment Table

CREATE TABLE Appointment (

AppointmentID VARCHAR2(50) PRIMARY KEY,

DoctorID VARCHAR2(50) NOT NULL,

AppointmentDate DATE NOT NULL,

AppointmentStatus VARCHAR2(50) CHECK (AppointmentStatus IN ('Scheduled', 'Completed', 'Canceled')),

CONSTRAINT FK\_Appointment\_Doctor FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

);

-- Visit Table

CREATE TABLE Visit (

VisitID VARCHAR2(50) PRIMARY KEY,

PatientID VARCHAR2(50) NOT NULL,

DoctorID VARCHAR2(50) NOT NULL,

VisitDate DATE NOT NULL,

VisitTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

VisitReason VARCHAR2(500) NOT NULL,

AppointmentID VARCHAR2(50) UNIQUE,

VisitStatus VARCHAR2(50) CHECK (VisitStatus IN ('Pending', 'Completed', 'Canceled')),

CONSTRAINT FK\_Visit\_Appointment FOREIGN KEY (AppointmentID) REFERENCES Appointment(AppointmentID),

CONSTRAINT FK\_Visit\_Doctor FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID),

CONSTRAINT FK\_Visit\_Patient FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)

);

-- MedicalRecord Table

CREATE TABLE MedicalRecord (

RecordID VARCHAR2(50) PRIMARY KEY,

PatientID VARCHAR2(50) NOT NULL,

ChronicConditions VARCHAR2(500),

RecordDate DATE NOT NULL,

CONSTRAINT FK\_MedicalRecord\_Patient FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)

);

-- TreatmentHistory Table

CREATE TABLE TreatmentHistory (

TreatmentID VARCHAR2(50) PRIMARY KEY,

VisitID VARCHAR2(50) NOT NULL,

Description VARCHAR2(500) NOT NULL,

TreatmentDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_TreatmentHistory\_Visit FOREIGN KEY (VisitID) REFERENCES Visit(VisitID)

);

-- Prescription Table

CREATE TABLE Prescription (

PrescriptionID VARCHAR2(50) PRIMARY KEY,

TreatmentID VARCHAR2(50) NOT NULL,

Medication VARCHAR2(100) NOT NULL,

Dosage VARCHAR2(50) NOT NULL,

CONSTRAINT FK\_Prescription\_Treatment FOREIGN KEY (TreatmentID) REFERENCES TreatmentHistory(TreatmentID)

);

-- Billing Table

CREATE TABLE Billing (

BillID VARCHAR2(50) PRIMARY KEY,

PatientID VARCHAR2(50) NOT NULL,

VisitID VARCHAR2(50) NOT NULL,

TotalAmount NUMBER(10,2) CHECK (TotalAmount >= 0) NOT NULL,

PaymentStatus VARCHAR2(50) CHECK (PaymentStatus IN ('Paid', 'Pending', 'Failed')) NOT NULL,

CONSTRAINT FK\_Billing\_Patient FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),

CONSTRAINT FK\_Billing\_Visit FOREIGN KEY (VisitID) REFERENCES Visit(VisitID)

);

**8. DML SCRIPT:-**

-- DATA for DMDD PROJECT

-- Insert Data for UserRoles

INSERT INTO UserRoles VALUES ('R001', 'Admin');

INSERT INTO UserRoles VALUES ('R002', 'Doctor');

INSERT INTO UserRoles VALUES ('R003', 'BillingStaff');

-- Insert Data for Users

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U001', 'admin\_user', 'hashed\_password', 'R001', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U002', 'doctor\_smith', 'hashed\_password', 'R002', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U003', 'billing\_staff', 'hashed\_password', 'R003', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U004', 'doctor\_jane', 'hashed\_password', 'R002', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U005', 'billing\_mary', 'hashed\_password', 'R003', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

-- Add Users for Patients & New Doctor

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U006', 'bob\_miller', 'hashed\_password', NULL, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U007', 'emma\_wilson', 'hashed\_password', NULL, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U008', 'michael\_johnson', 'hashed\_password', NULL, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U009', 'sophia\_lee', 'hashed\_password', NULL, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

INSERT INTO Users (UserID, Username, PasswordHash, RoleID, CreatedAt, UpdatedAt)

VALUES ('U010', 'robert\_brown', 'hashed\_password', 'R002', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

-- Insert Data for Doctors

INSERT INTO Doctor VALUES ('D001', 'U002', 'John', 'Smith', 'Cardiology', 'DOC12345', 10, 'Available');

INSERT INTO Doctor VALUES ('D002', 'U004', 'Jane', 'Doe', 'Pediatrics', 'DOC67890', 8, 'Available');

INSERT INTO Doctor VALUES ('D003', 'U010', 'Robert', 'Brown', 'Neurology', 'DOC78901', 15, 'Available');

-- Insert Data for Patients

INSERT INTO Patient VALUES ('P001', 'U003', 'Alice', 'Johnson', TO\_DATE('1990-05-10', 'YYYY-MM-DD'), 'Female', 'alice@example.com', '1234567890', 'Emergency Contact', CURRENT\_TIMESTAMP);

INSERT INTO Patient VALUES ('P002', 'U006', 'Bob', 'Miller', TO\_DATE('1985-02-14', 'YYYY-MM-DD'), 'Male', 'bob@example.com', '9876543210', 'Spouse Contact', CURRENT\_TIMESTAMP);

INSERT INTO Patient VALUES ('P003', 'U007', 'Emma', 'Wilson', TO\_DATE('1995-08-21', 'YYYY-MM-DD'), 'Female', 'emma@example.com', '7894561230', 'Mother Contact', CURRENT\_TIMESTAMP);

INSERT INTO Patient VALUES ('P004', 'U008', 'Michael', 'Johnson', TO\_DATE('1982-07-03', 'YYYY-MM-DD'), 'Male', 'michael@example.com', '4567891230', 'Brother Contact', CURRENT\_TIMESTAMP);

INSERT INTO Patient VALUES ('P005', 'U009', 'Sophia', 'Lee', TO\_DATE('1998-12-15', 'YYYY-MM-DD'), 'Female', 'sophia@example.com', '8529637410', 'Father Contact', CURRENT\_TIMESTAMP);

-- Insert Data for Appointments

INSERT INTO Appointment VALUES ('A001', 'D001', TO\_DATE('2024-03-20', 'YYYY-MM-DD'), 'Scheduled');

INSERT INTO Appointment VALUES ('A002', 'D002', TO\_DATE('2024-03-21', 'YYYY-MM-DD'), 'Completed');

INSERT INTO Appointment VALUES ('A003', 'D003', TO\_DATE('2024-03-22', 'YYYY-MM-DD'), 'Scheduled');

INSERT INTO Appointment VALUES ('A004', 'D001', TO\_DATE('2024-03-23', 'YYYY-MM-DD'), 'Scheduled');

INSERT INTO Appointment VALUES ('A005', 'D002', TO\_DATE('2024-03-24', 'YYYY-MM-DD'), 'Scheduled');

-- Insert Data for Visits

INSERT INTO Visit VALUES ('V001', 'P001', 'D001', TO\_DATE('2024-03-20', 'YYYY-MM-DD'), CURRENT\_TIMESTAMP, 'Regular Checkup', 'A001', 'Pending');

INSERT INTO Visit VALUES ('V002', 'P002', 'D002', TO\_DATE('2024-03-21', 'YYYY-MM-DD'), CURRENT\_TIMESTAMP, 'Annual Checkup', 'A002', 'Completed');

INSERT INTO Visit VALUES ('V003', 'P003', 'D003', TO\_DATE('2024-03-22', 'YYYY-MM-DD'), CURRENT\_TIMESTAMP, 'Neurology Consultation', 'A003', 'Pending');

INSERT INTO Visit VALUES ('V004', 'P004', 'D001', TO\_DATE('2024-03-23', 'YYYY-MM-DD'), CURRENT\_TIMESTAMP, 'Follow-up', 'A004', 'Pending');

INSERT INTO Visit VALUES ('V005', 'P005', 'D002', TO\_DATE('2024-03-24', 'YYYY-MM-DD'), CURRENT\_TIMESTAMP, 'General Consultation', 'A005', 'Pending');

-- Insert Data for Billing

INSERT INTO Billing VALUES ('B001', 'P001', 'V001', 250.00, 'Pending');

INSERT INTO Billing VALUES ('B002', 'P002', 'V002', 180.00, 'Paid');

INSERT INTO Billing VALUES ('B003', 'P003', 'V003', 300.00, 'Pending');

INSERT INTO Billing VALUES ('B004', 'P004', 'V004', 150.00, 'Pending');

INSERT INTO Billing VALUES ('B005', 'P005', 'V005', 200.00, 'Pending');

**9. View SCRIPT:-**

**CODE:-**

-- SECTION A: General Views

-- ---------------------------

-- View: Doctor Availability (Visible to all roles)

CREATE OR REPLACE VIEW Doctor\_Availability AS

SELECT DoctorID, FirstName, LastName, Specialization, Availability

FROM Doctor;

-- View: Patient Visit Summary (Admin only)

CREATE OR REPLACE VIEW Patient\_Visit\_Summary AS

SELECT V.VisitID, P.FirstName || ' ' || P.LastName AS PatientName,

D.FirstName || ' ' || D.LastName AS DoctorName,

V.VisitDate, V.VisitReason, V.VisitStatus

FROM Visit V

JOIN Patient P ON V.PatientID = P.PatientID

JOIN Doctor D ON V.DoctorID = D.DoctorID;

-- View: Billing Insights (Admin only)

CREATE OR REPLACE VIEW Billing\_Insights AS

SELECT B.BillID, P.FirstName || ' ' || P.LastName AS PatientName,

V.VisitDate, B.TotalAmount, B.PaymentStatus

FROM Billing B

JOIN Visit V ON B.VisitID = V.VisitID

JOIN Patient P ON B.PatientID = P.PatientID;

-- SECTION B: Role-Specific Views

-- ---------------------------

-- View: Doctor-Only Patient Visit Summary

CREATE OR REPLACE VIEW Doctor\_Only\_Patient\_Summary AS

SELECT V.VisitID, P.FirstName || ' ' || P.LastName AS PatientName,

V.VisitDate, V.VisitReason, V.VisitStatus

FROM Visit V

JOIN Patient P ON V.PatientID = P.PatientID

WHERE V.DoctorID = (SELECT DoctorID FROM Doctor WHERE UserID =

(SELECT UserID FROM Users WHERE Username = SYS\_CONTEXT('USERENV','SESSION\_USER')));

-- View: Billing-Only Insights

CREATE OR REPLACE VIEW Billing\_Only\_View AS

SELECT B.BillID, P.FirstName || ' ' || P.LastName AS PatientName,

V.VisitDate, B.TotalAmount, B.PaymentStatus

FROM Billing B

JOIN Visit V ON B.VisitID = V.VisitID

JOIN Patient P ON B.PatientID = P.PatientID;

-- SECTION C: Test Cases for Role Enforcement

-- ---------------------------

-- Test Case: Doctor tries to update Billing (Should fail)

BEGIN

UPDATE Billing SET TotalAmount = 999.99 WHERE BillID = 'B001';

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Doctor cannot update billing – ' || SQLERRM);

END;

/

-- Test Case: Billing Staff tries to SELECT from MedicalRecord (Should fail)

BEGIN

FOR rec IN (SELECT \* FROM MedicalRecord) LOOP

NULL;

END LOOP;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Billing staff not allowed to view medical records – ' || SQLERRM);

END;

/

-- Test Case: Patient tries to insert Visit (Should fail)

BEGIN

INSERT INTO Visit (VisitID, PatientID, DoctorID, VisitDate, VisitReason, VisitStatus)

VALUES ('V999', 'P001', 'D001', SYSDATE, 'Unauthorized Visit', 'Pending');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Unauthorized insert into Visit table – ' || SQLERRM);

END;

/

-- Test Case: Doctor tries to delete a user (Should fail)

BEGIN

DELETE FROM Users WHERE Username = 'admin\_user';

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Doctor cannot delete users – ' || SQLERRM);

END;

/

-- SECTION D: Constraint Testing

-- ---------------------------

-- Test Case: Insert duplicate email in Patient (Should fail)

BEGIN

INSERT INTO Patient (PatientID, UserID, FirstName, LastName, DOB, Gender, Email, PhoneNumber, EmergencyContact, CreatedAt)

VALUES ('P999', 'U001', 'Test', 'Duplicate', SYSDATE, 'Male', 'alice@example.com', '1231231234', 'Test Contact', SYSDATE);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Duplicate email not allowed – ' || SQLERRM);

END;

/

-- Test Case: Appointment in the past (Should fail business logic)

BEGIN

INSERT INTO Appointment (AppointmentID, DoctorID, AppointmentDate, AppointmentStatus)

VALUES ('A999', 'D001', TO\_DATE('2023-01-01','YYYY-MM-DD'), 'Scheduled');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(' ERROR CAUGHT: Cannot schedule appointment in the past – ' || SQLERRM);

END;

/

-- SECTION E: Additional Procedures

-- ---------------------------

-- Procedure: Update Visit Status

CREATE OR REPLACE PROCEDURE Update\_Visit\_Status (

p\_visit\_id VARCHAR2,

p\_status VARCHAR2

) AS

BEGIN

IF p\_status NOT IN ('Pending', 'Completed', 'Canceled') THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid visit status.');

END IF;

UPDATE Visit

SET VisitStatus = p\_status

WHERE VisitID = p\_visit\_id;

END;

/

-- Procedure: Complete Payment

CREATE OR REPLACE PROCEDURE Complete\_Payment (

p\_bill\_id VARCHAR2

) AS

BEGIN

UPDATE Billing

SET PaymentStatus = 'Paid'

WHERE BillID = p\_bill\_id;

END;

/

**10. User Creation & Grants:-**

**Code:-**

-- Drop existing users to avoid conflicts

BEGIN

FOR user\_rec IN (SELECT username FROM dba\_users WHERE username IN ('ADMIN\_USER', 'DOC\_USER', 'BILL\_USER')) LOOP

EXECUTE IMMEDIATE 'DROP USER ' || user\_rec.username || ' CASCADE';

END LOOP;

END;

/

-- Create Users with Strong Passwords (At least 12 characters, uppercase, numbers, special characters)

CREATE USER admin\_user IDENTIFIED BY "Admin@Secure#1234";

CREATE USER doc\_user IDENTIFIED BY "Doctor@Secure#1234";

CREATE USER bill\_user IDENTIFIED BY "Billing@Secure#1234";

-- Grant Basic Database Access

GRANT CONNECT, RESOURCE TO admin\_user;

GRANT CONNECT, RESOURCE TO doc\_user;

GRANT CONNECT, RESOURCE TO bill\_user;

-- Grant Admin Permissions

GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO admin\_user;

GRANT CREATE USER, ALTER USER, DROP USER TO admin\_user;

GRANT CREATE ROLE, GRANT ANY ROLE TO admin\_user;

-- Grant Doctor User Permissions

GRANT CREATE SESSION TO doc\_user;

GRANT SELECT, INSERT, UPDATE ON Patient TO doc\_user;

GRANT SELECT, INSERT, UPDATE ON Appointment TO doc\_user;

GRANT SELECT, INSERT, UPDATE ON MedicalRecord TO doc\_user;

GRANT SELECT ON Doctor TO doc\_user;

-- Grant Billing User Permissions

GRANT CREATE SESSION TO bill\_user;

GRANT SELECT, INSERT, UPDATE ON Billing TO bill\_user;

GRANT SELECT ON Billing TO bill\_user;

-- Prevent Revoking Unassigned Privileges

DECLARE

v\_count NUMBER;

BEGIN

-- Check and revoke privilege only if granted

SELECT COUNT(\*) INTO v\_count FROM DBA\_SYS\_PRIVS WHERE GRANTEE = 'DOC\_USER' AND PRIVILEGE = 'DROP ANY TABLE';

IF v\_count > 0 THEN

EXECUTE IMMEDIATE 'REVOKE DROP ANY TABLE FROM doc\_user';

END IF;

SELECT COUNT(\*) INTO v\_count FROM DBA\_SYS\_PRIVS WHERE GRANTEE = 'BILL\_USER' AND PRIVILEGE = 'DROP ANY TABLE';

IF v\_count > 0 THEN

EXECUTE IMMEDIATE 'REVOKE DROP ANY TABLE FROM bill\_user';

END IF;

END;

/