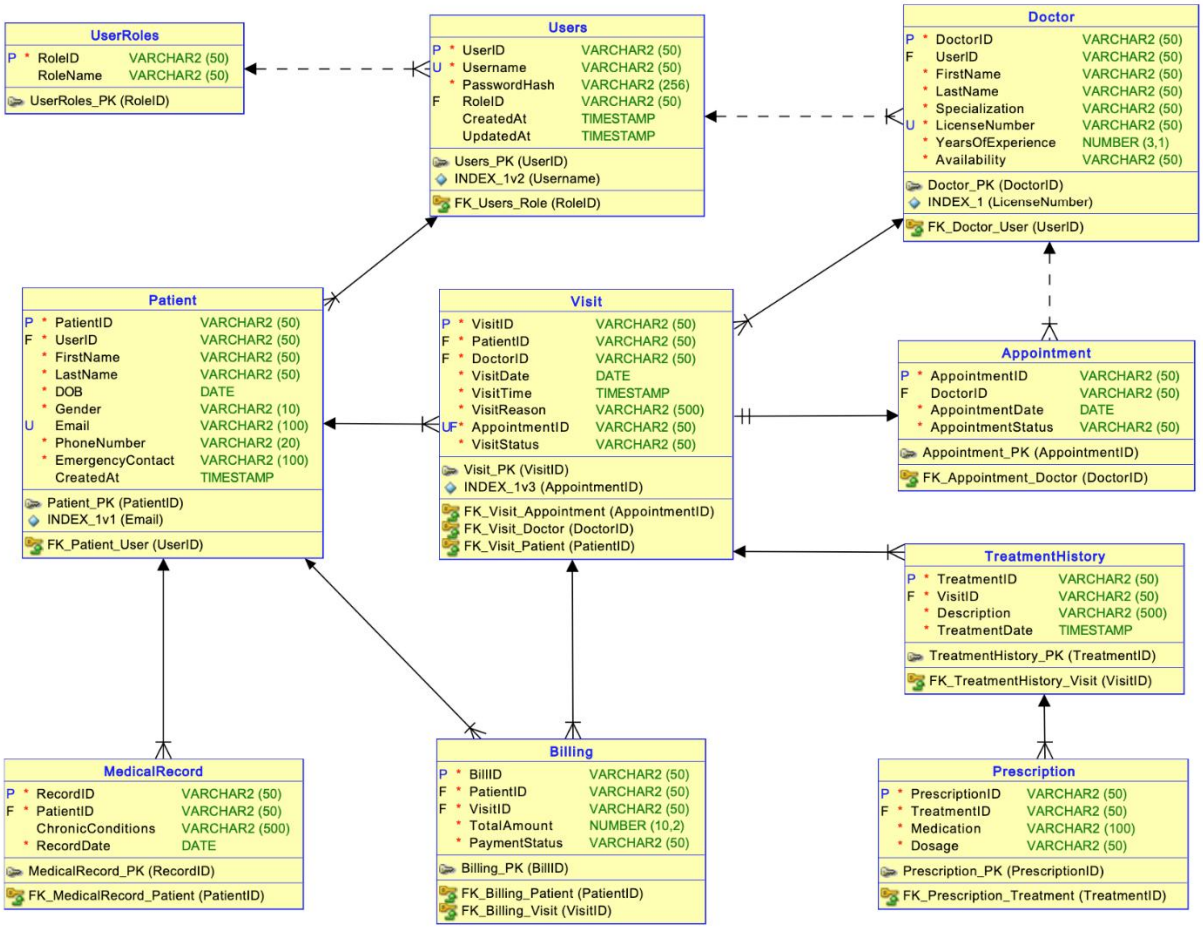
Phase-2 Documentation for H.E.A.L. (Healthcare Efficiency & Assistance Log)

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).

Patient ID	PatientName	Doctor ID	DoctorName	VisitDate	Treatment ID	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:

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

2. **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.

3. **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:-

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.
1. 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**

2. 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**

3. 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**

4. Billing & Insurance Processing

Actors: Patient, Admin, Insurance Provider

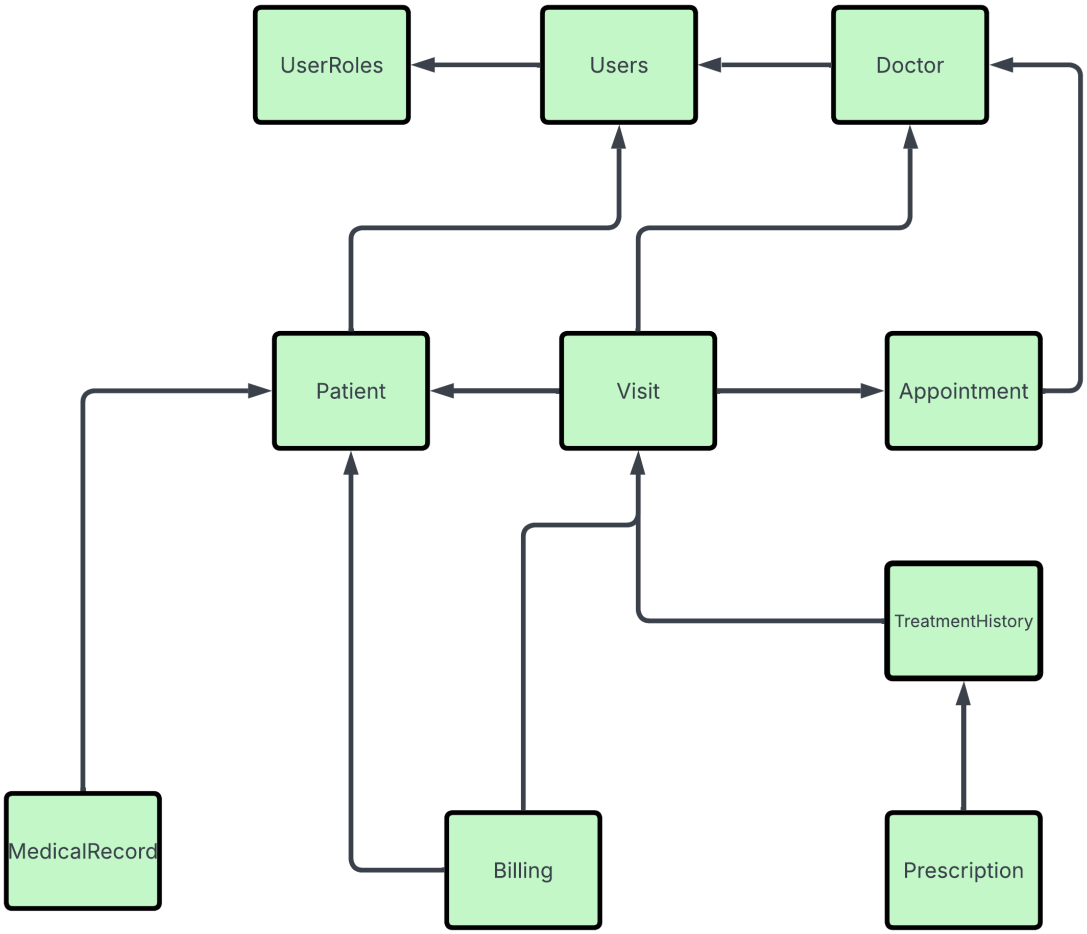
Processes:

- **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**

DATA FLOW DIAGRAM



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**.**

1.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.

2. 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.

3. 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 (
wq
   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. Views, Indexes, Triggers, Procedures, Package head, body and test cases:-

```
-------------------------------------------------------------
--                  INDEXES
-------------------------------------------------------------
-- Drop Queries to drop the indexes
DROP INDEX ADMIN_USER.idx_appointment_date;
DROP INDEX ADMIN_USER.idx_visit_patient;
DROP INDEX ADMIN_USER.idx_visit_doctor;
DROP INDEX ADMIN_USER.idx_patient_email_phone;
DROP INDEX ADMIN_USER.idx_medicalrecord_patient;

-- Create an index on the Appointment table's AppointmentDate column
CREATE INDEX ADMIN_USER.idx_appointment_date
  ON ADMIN_USER.Appointment(AppointmentDate);

-- Create indexes on the Visit table's foreign key columns for faster joins
CREATE INDEX ADMIN_USER.idx_visit_patient
  ON ADMIN_USER.Visit(PatientID);

-- Create an index on the Visit table's DoctorID column for faster joins
CREATE INDEX ADMIN_USER.idx_visit_doctor
  ON ADMIN_USER.Visit(DoctorID);

-- Create a composite index on the Patient table's Email and PhoneNumber columns
-- for faster lookups when querying by both email and phone number together.
CREATE INDEX ADMIN_USER.idx_patient_email_phone
  ON ADMIN_USER.Patient(Email, PhoneNumber);

-- Create an index on the MedicalRecord table's PatientID column for efficient joins
CREATE INDEX ADMIN_USER.idx_medicalrecord_patient
  ON ADMIN_USER.MedicalRecord(PatientID);



-----------------------------------------------------------
--    TRIGGERS
-----------------------------------------------------------
-- This trigger checks, before any INSERT or UPDATE on the Appointment table,
whether the new AppointmentDate is earlier than today (using TRUNC(SYSDATE) to
ignore the time component)
```

```
CREATE OR REPLACE TRIGGER ADMIN_USER.trg_appointment_date_check
BEFORE INSERT OR UPDATE ON ADMIN_USER.Appointment
FOR EACH ROW
BEGIN
   IF :NEW.AppointmentDate < TRUNC(SYSDATE) THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot schedule appointment in the
past.');
   END IF;
END;
/

-- Auto-update the "UpdatedAt" Column on the Users Table

CREATE OR REPLACE TRIGGER ADMIN_USER.trg_update_user_timestamp
BEFORE UPDATE ON ADMIN_USER.Users
FOR EACH ROW
BEGIN
   :NEW.UpdatedAt := SYSDATE;
END;
/

--Prevent Deletion of Doctors if They Have Associated Appointments
CREATE OR REPLACE TRIGGER ADMIN_USER.trg_prevent_doctor_delete
BEFORE DELETE ON ADMIN_USER.Doctor
FOR EACH ROW
DECLARE
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count
     FROM ADMIN_USER.Appointment
     WHERE DoctorID = :OLD.DoctorID;

   IF v_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Cannot delete doctor: appointments
exist.');
   END IF;
END;
/


------------------------------------------------------------
-- [VIEW CREATION SECTION] - Run as ADMIN_USER
------------------------------------------------------------

SET SERVEROUTPUT ON;
```

```
--------------------------------------------------------------------------
-- 1) CREATE OR REPLACE VIEW: Doctor_Availability
--    Allowed: ADMIN_USER, DOC_USER, BILL_USER
--    Others get "Access Denied."
--------------------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE '
      CREATE OR REPLACE VIEW ADMIN_USER.Doctor_Availability AS
      SELECT
         DoctorID,
         FirstName,
         LastName,
         Specialization,
         Availability
      FROM ADMIN_USER.Doctor
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER"))
           IN ("ADMIN_USER", "DOC_USER", "BILL_USER")

      UNION ALL

      SELECT
         "Access Denied" AS DoctorID,
         "Access Denied" AS FirstName,
         "Access Denied" AS LastName,
         "Access Denied" AS Specialization,
         "Access Denied" AS Availability
      FROM DUAL
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER"))
           NOT IN ("ADMIN_USER", "DOC_USER", "BILL_USER")
   ';
   DBMS_OUTPUT.PUT_LINE('View ADMIN_USER.Doctor_Availability created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating ADMIN_USER.Doctor_Availability: '
|| SQLERRM);
END;
/

-- Grant SELECT on Doctor_Availability to DOC_USER and BILL_USER
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Doctor_Availability
TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Doctor_Availability to
DOC_USER');
```

```
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Doctor_Availability
TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Doctor_Availability to
BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Doctor_Availability: ' ||
SQLERRM);
END;
/

-----------------------------------------------------------------------
-- 2) CREATE OR REPLACE VIEW: Patient_Visit_Summary
--    Allowed: ADMIN_USER, BILL_USER
--    Others get "Access Denied."
-----------------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE '
      CREATE OR REPLACE VIEW ADMIN_USER.Patient_Visit_Summary AS
      SELECT
         TO_CHAR(V.VisitID) AS VisitID,
         P.FirstName || " " || P.LastName AS PatientName,
         D.FirstName || " " || D.LastName AS DoctorName,
         TO_CHAR(V.VisitDate, "YYYY-MM-DD") AS VisitDate,
         V.VisitReason,
         V.VisitStatus
      FROM ADMIN_USER.Visit V
          JOIN ADMIN_USER.Patient P ON V.PatientID = P.PatientID
          JOIN ADMIN_USER.Doctor D ON V.DoctorID = D.DoctorID
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER"))
           IN ("ADMIN_USER", "BILL_USER")

      UNION ALL

      SELECT
         "Access Denied" AS VisitID,
         "Access Denied" AS PatientName,
         "Access Denied" AS DoctorName,
         "Access Denied" AS VisitDate,
         "Access Denied" AS VisitReason,
         "Access Denied" AS VisitStatus
      FROM DUAL
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER"))
           NOT IN ("ADMIN_USER", "BILL_USER")
   ';
```

```
   DBMS_OUTPUT.PUT_LINE('View ADMIN_USER.Patient_Visit_Summary
created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating
ADMIN_USER.Patient_Visit_Summary: ' || SQLERRM);
END;
/
-- Grant SELECT on Patient_Visit_Summary to both BILL_USER and DOC_USER
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON
ADMIN_USER.Patient_Visit_Summary TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Patient_Visit_Summary to
BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Patient_Visit_Summary
to BILL_USER: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON
ADMIN_USER.Patient_Visit_Summary TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Patient_Visit_Summary to
DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Patient_Visit_Summary
to DOC_USER: ' || SQLERRM);
END;
/

--------------------------------------------------------------------
-- 3) CREATE OR REPLACE VIEW: Billing_Insights
--    Allowed: ADMIN_USER only
--    Everyone else gets "Access Denied."
--------------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE '
      CREATE OR REPLACE VIEW ADMIN_USER.Billing_Insights AS
      SELECT
         TO_CHAR(B.BillID)      AS BillID,
         P.FirstName || " " || P.LastName AS PatientName,
         TO_CHAR(V.VisitDate, "YYYY-MM-DD") AS VisitDate,
         TO_CHAR(B.TotalAmount)  AS TotalAmount,
         B.PaymentStatus
```

```
        FROM ADMIN_USER.Billing B
            JOIN ADMIN_USER.Visit V     ON B.VisitID = V.VisitID
            JOIN ADMIN_USER.Patient P  ON B.PatientID = P.PatientID
        WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) =
"ADMIN_USER"

        UNION ALL

        SELECT
            "Access Denied" AS BillID,
            "Access Denied" AS PatientName,
            "Access Denied" AS VisitDate,
            "Access Denied" AS TotalAmount,
            "Access Denied" AS PaymentStatus
        FROM DUAL
        WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) <>
"ADMIN_USER"
    ';
    DBMS_OUTPUT.PUT_LINE('View ADMIN_USER.Billing_Insights created.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error creating ADMIN_USER.Billing_Insights: ' ||
SQLERRM);
END;
/
-- Grant SELECT on Billing_Insights to BILL_USER (so non-admin users see "Access
Denied")
BEGIN
    EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Billing_Insights TO
BILL_USER';
    DBMS_OUTPUT.PUT_LINE('Granted SELECT on Billing_Insights to
BILL_USER');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Billing_Insights to
BILL_USER: ' || SQLERRM);
END;
/

-------------------------------------------------------------------------
-- 4) CREATE OR REPLACE VIEW: Doctor_Only_Patient_Summary
--    Allowed: DOC_USER only (and optionally ADMIN_USER)
-------------------------------------------------------------------------
BEGIN
    EXECUTE IMMEDIATE '
```

```
      CREATE OR REPLACE VIEW ADMIN_USER.Doctor_Only_Patient_Summary
AS
      SELECT
         TO_CHAR(V.VisitID) AS VisitID,
         P.FirstName || " " || P.LastName AS PatientName,
         TO_CHAR(V.VisitDate, "YYYY-MM-DD") AS VisitDate,
         V.VisitReason,
         V.VisitStatus
      FROM ADMIN_USER.Visit V
          JOIN ADMIN_USER.Patient P ON V.PatientID = P.PatientID
          JOIN ADMIN_USER.Doctor D ON V.DoctorID = D.DoctorID
          JOIN ADMIN_USER.Users U ON D.UserID = U.UserID
      WHERE UPPER(U.Username) = UPPER(SYS_CONTEXT("USERENV",
"SESSION_USER"))
        AND UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) IN
("DOC_USER", "ADMIN_USER")

      UNION ALL

      SELECT
         "Access Denied" AS VisitID,
         "Access Denied" AS PatientName,
         "Access Denied" AS VisitDate,
         "Access Denied" AS VisitReason,
         "Access Denied" AS VisitStatus
      FROM DUAL
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) NOT IN
("DOC_USER", "ADMIN_USER")
   ';
   DBMS_OUTPUT.PUT_LINE('View ADMIN_USER.Doctor_Only_Patient_Summary
created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating
ADMIN_USER.Doctor_Only_Patient_Summary: ' || SQLERRM);
END;
/
-- Grant SELECT on Doctor_Only_Patient_Summary to DOC_USER and to
BILL_USER (so they can see "Access Denied")
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON
ADMIN_USER.Doctor_Only_Patient_Summary TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Doctor_Only_Patient_Summary
to DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
```

```
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on
Doctor_Only_Patient_Summary to DOC_USER: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON
ADMIN_USER.Doctor_Only_Patient_Summary TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Doctor_Only_Patient_Summary
to BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on
Doctor_Only_Patient_Summary to BILL_USER: ' || SQLERRM);
END;
/

-------------------------------------------------------------------------
-- 5) CREATE OR REPLACE VIEW: Billing_Only_View
--    Allowed: BILL_USER only (and optionally ADMIN_USER).
-------------------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE '
      CREATE OR REPLACE VIEW ADMIN_USER.Billing_Only_View AS
      SELECT
         TO_CHAR(B.BillID) AS BillID,
         P.FirstName || " " || P.LastName AS PatientName,
         TO_CHAR(V.VisitDate, "YYYY-MM-DD") AS VisitDate,
         TO_CHAR(B.TotalAmount) AS TotalAmount,
         B.PaymentStatus
      FROM ADMIN_USER.Billing B
          JOIN ADMIN_USER.Visit V   ON B.VisitID = V.VisitID
          JOIN ADMIN_USER.Patient P ON B.PatientID = P.PatientID
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) IN
("BILL_USER", "ADMIN_USER")

      UNION ALL

      SELECT
         "Access Denied"  AS BillID,
         "Access Denied"  AS PatientName,
         "Access Denied"  AS VisitDate,
         "Access Denied"  AS TotalAmount,
         "Access Denied"  AS PaymentStatus
      FROM DUAL
      WHERE UPPER(SYS_CONTEXT("USERENV", "SESSION_USER")) NOT IN
("BILL_USER", "ADMIN_USER")
```

```
    ';
    DBMS_OUTPUT.PUT_LINE('View ADMIN_USER.Billing_Only_View created.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error creating ADMIN_USER.Billing_Only_View: '
|| SQLERRM);
END;
/
-- Grant SELECT on Billing_Only_View to BILL_USER (already granted previously in
your code)
BEGIN
    EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Billing_Only_View
TO BILL_USER';
    DBMS_OUTPUT.PUT_LINE('Granted SELECT on Billing_Only_View to
BILL_USER');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Billing_Only_View to
BILL_USER: ' || SQLERRM);
END;
/

------------------------------------------------------------
-- [UNDERLYING TABLE PRIVILEGES SECTION]
------------------------------------------------------------
BEGIN
    EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Billing TO
DOC_USER';
    DBMS_OUTPUT.PUT_LINE('Granted SELECT on Billing to DOC_USER');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Billing to DOC_USER: '
|| SQLERRM);
END;
/
BEGIN
    EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Visit TO
DOC_USER';
    DBMS_OUTPUT.PUT_LINE('Granted SELECT on Visit to DOC_USER');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Visit to DOC_USER: ' ||
SQLERRM);
END;
/
BEGIN
```

```
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Users TO
DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Users to DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Users to DOC_USER: ' ||
SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.MedicalRecord TO
BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on MedicalRecord to BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on MedicalRecord to
BILL_USER: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Visit TO
BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Visit to BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Visit to BILL_USER: ' ||
SQLERRM);
END;
/


-------------------------------------------------------------
-- CREATE PUBLIC SYNONYMS FOR THE VIEWS
-------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'CREATE OR REPLACE PUBLIC SYNONYM
Doctor_Availability FOR ADMIN_USER.Doctor_Availability';
   DBMS_OUTPUT.PUT_LINE('Public synonym Doctor_Availability created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for
Doctor_Availability: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'CREATE OR REPLACE PUBLIC SYNONYM
Patient_Visit_Summary FOR ADMIN_USER.Patient_Visit_Summary';
```

```
   DBMS_OUTPUT.PUT_LINE('Public synonym Patient_Visit_Summary created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for
Patient_Visit_Summary: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'CREATE OR REPLACE PUBLIC SYNONYM
Billing_Insights FOR ADMIN_USER.Billing_Insights';
   DBMS_OUTPUT.PUT_LINE('Public synonym Billing_Insights created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for Billing_Insights: '
|| SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'CREATE OR REPLACE PUBLIC SYNONYM
Doctor_Only_Patient_Summary FOR ADMIN_USER.Doctor_Only_Patient_Summary';
   DBMS_OUTPUT.PUT_LINE('Public synonym Doctor_Only_Patient_Summary
created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for
Doctor_Only_Patient_Summary: ' || SQLERRM);
END;
/
BEGIN
   EXECUTE IMMEDIATE 'CREATE OR REPLACE PUBLIC SYNONYM
Billing_Only_View FOR ADMIN_USER.Billing_Only_View';
   DBMS_OUTPUT.PUT_LINE('Public synonym Billing_Only_View created.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for
Billing_Only_View: ' || SQLERRM);
END;
/
```

```
-- [TEST CASES  FOR THE VIEWS SECTION]
------------------------------------------------------------

--Test Queries for ADMIN_USER
------------------------------------------------------------

SELECT *
FROM ADMIN_USER.Doctor_Availability;
-- Expected: Returns full doctor details.

SELECT *
FROM ADMIN_USER.Patient_Visit_Summary;
-- Expected: Returns all patient visit records with valid details.

SELECT *
FROM ADMIN_USER.Billing_Insights;
-- Expected: Returns all billing records.

--Test Queries for DOC_USER
------------------------------------------------------------
SELECT *
FROM ADMIN_USER.Doctor_Availability;
-- Expected: Returns full doctor details.

SELECT *
FROM ADMIN_USER.Doctor_Only_Patient_Summary;
-- Expected: Returns visit records associated with DOC_USER (or "Access Denied" if
DOC_USER is not matched).

SELECT *
FROM ADMIN_USER.Patient_Visit_Summary;
-- Expected: Depending on your view logic, this could return "Access Denied" or no rows
if Patient_Visit_Summary is not allowed for DOC_USER.


--Test Queries for BILL_USER
-----------------------------------------------------
SELECT *
FROM ADMIN_USER.Billing_Only_View;
-- Expected: Returns billing-specific records for BILL_USER.

SELECT *
FROM ADMIN_USER.Billing_Insights;
-- Expected: Should NOT be accessible to BILL_USER; ideally, it should return "Access
Denied" or raise an error.
```

```
SELECT *
FROM ADMIN_USER.Doctor_Only_Patient_Summary;
-- Expected: Should NOT be accessible to BILL_USER; it should return "Access
Denied" or no data.




------------------------------------------------------------
-- PROCEDURES to be runned in ADMIN_ USER
------------------------------------------------------------
-- Procedure: Update_Visit_Status
CREATE OR REPLACE PROCEDURE Update_Visit_Status (
   p_visit_id IN VARCHAR2,
   p_status   IN VARCHAR2
) AS
BEGIN
   -- Validate the status value
   IF p_status NOT IN ('Pending', 'Completed', 'Canceled') THEN
      RAISE_APPLICATION_ERROR(-20003, 'Invalid visit status. Must be Pending,
Completed, or Canceled.');
   END IF;

   -- Update Visit table
   UPDATE Visit
   SET VisitStatus = p_status
   WHERE VisitID = p_visit_id;

   -- Optionally: COMMIT; -- uncomment if automatic commit is desired

   DBMS_OUTPUT.PUT_LINE('Visit status updated successfully for ' || p_visit_id);
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in Update_Visit_Status: ' || SQLERRM);
      RAISE;
END;
/

-- Procedure: Complete_Payment
CREATE OR REPLACE PROCEDURE Complete_Payment (
   p_bill_id IN VARCHAR2
) AS
BEGIN
   -- Update Billing table to mark payment as complete
   UPDATE Billing
```

```
   SET PaymentStatus = 'Paid'
   WHERE BillID = p_bill_id;

   -- Optionally: COMMIT; -- uncomment if automatic commit is desired

   DBMS_OUTPUT.PUT_LINE('Payment completed for Bill ' || p_bill_id);
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in Complete_Payment: ' || SQLERRM);
      RAISE;
END;
/

-- Procedure: Record_Treatment
CREATE OR REPLACE PROCEDURE Record_Treatment (
   p_visit_id    IN VARCHAR2,
   p_description IN VARCHAR2
) AS
   -- Generate a unique TreatmentID using SYS_GUID (converted to hexadecimal)
   v_treatment_id VARCHAR2(50) := RAWTOHEX(SYS_GUID());
BEGIN
   INSERT INTO ADMIN_USER.TreatmentHistory (TreatmentID, VisitID, Description,
TreatmentDate)
   VALUES (v_treatment_id, p_visit_id, p_description, SYSDATE);

   DBMS_OUTPUT.PUT_LINE('Treatment recorded successfully for Visit ' || p_visit_id
||
                        '. Treatment ID: ' || v_treatment_id);
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in Record_Treatment: ' || SQLERRM);
      RAISE;
END;
/

BEGIN
   Record_Treatment('V001', 'Administered flu vaccine');
END;
/



--------------------------------------------------------------------------
--       PACKAGE :- [healthcare_pkg]
--------------------------------------------------------------------------
```

```
-- Package Specification (Header):

CREATE OR REPLACE PACKAGE ADMIN_USER.healthcare_pkg IS
   PROCEDURE Update_Visit_Status(p_visit_id IN VARCHAR2, p_status IN
VARCHAR2);
   PROCEDURE Complete_Payment(p_bill_id IN VARCHAR2);
   PROCEDURE Record_Treatment(p_visit_id IN VARCHAR2, p_description IN
VARCHAR2);
END healthcare_pkg;
/

--Package Body:

CREATE OR REPLACE PACKAGE BODY ADMIN_USER.healthcare_pkg IS

   PROCEDURE Update_Visit_Status(p_visit_id IN VARCHAR2, p_status IN
VARCHAR2) IS
   BEGIN
      -- Validate the status value
      IF p_status NOT IN ('Pending', 'Completed', 'Canceled') THEN
         RAISE_APPLICATION_ERROR(-20003, 'Invalid visit status. Must be Pending,
Completed, or Canceled.');
      END IF;

      UPDATE ADMIN_USER.Visit
      SET VisitStatus = p_status
      WHERE VisitID = p_visit_id;

      DBMS_OUTPUT.PUT_LINE('Visit status updated successfully for ' || p_visit_id);
   EXCEPTION
      WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('Error in Update_Visit_Status: ' || SQLERRM);
         RAISE;
   END Update_Visit_Status;

   PROCEDURE Complete_Payment(p_bill_id IN VARCHAR2) IS
   BEGIN
      UPDATE ADMIN_USER.Billing
      SET PaymentStatus = 'Paid'
      WHERE BillID = p_bill_id;

      DBMS_OUTPUT.PUT_LINE('Payment completed for Bill ' || p_bill_id);
   EXCEPTION
      WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Error in Complete_Payment: ' || SQLERRM);
        RAISE;
    END Complete_Payment;

    PROCEDURE Record_Treatment(p_visit_id IN VARCHAR2, p_description IN
VARCHAR2) IS
        v_treatment_id VARCHAR2(50) := RAWTOHEX(SYS_GUID());  -- it will
generate unique ID
    BEGIN
        INSERT INTO ADMIN_USER.TreatmentHistory (TreatmentID, VisitID,
Description, TreatmentDate)
        VALUES (v_treatment_id, p_visit_id, p_description, SYSDATE);

        DBMS_OUTPUT.PUT_LINE('Treatment recorded successfully for Visit ' ||
p_visit_id ||
                             '. Treatment ID: ' || v_treatment_id);
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error in Record_Treatment: ' || SQLERRM);
            RAISE;
    END Record_Treatment;

END healthcare_pkg;
/




-------------------------------------------------------------
-- Test Package Queries for healthcare_pkg
-------------------------------------------------------------

-- Test Case: Update Visit Status with a valid status.
BEGIN
    -- Calling the procedure with a valid status.
    ADMIN_USER.healthcare_pkg.Update_Visit_Status('V001', 'Completed');
END;
/
-- After running, verify the update:
SELECT VisitID, VisitStatus FROM ADMIN_USER.Visit WHERE VisitID = 'V001';

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

-- Test Case: Update Visit Status with an invalid status.
-- This will trigger the validation and raise an error, which is then caught.
BEGIN
```

```
   -- Calling the procedure with an invalid status (e.g., 'InvalidStatus').
   ADMIN_USER.healthcare_pkg.Update_Visit_Status('V001', 'InvalidStatus');
EXCEPTION
   WHEN OTHERS THEN
      -- The error message should indicate that the status is invalid.
      DBMS_OUTPUT.PUT_LINE('Expected error: ' || SQLERRM);
END;
/
-- Optionally, re-run to confirm the behavior:
BEGIN
   ADMIN_USER.healthcare_pkg.Update_Visit_Status('V001', 'InvalidStatus');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Expected error: ' || SQLERRM);
END;
/

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

-- Test Case: Complete Payment.
-- This will update the Billing record with BillID 'B001' to 'Paid'.
BEGIN
   -- Calling the procedure with a valid BillID.
   ADMIN_USER.healthcare_pkg.Complete_Payment('B001');
END;
/
-- Verify the result:
SELECT BillID, PaymentStatus FROM ADMIN_USER.Billing WHERE BillID =
'B001';

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

-- Test Case: Record Treatment.
-- This will insert a new treatment record into the TreatmentHistory table
-- for the Visit with VisitID 'V001' with the given treatment description.
BEGIN
   -- Calling the procedure with a valid VisitID and treatment description.
   ADMIN_USER.healthcare_pkg.Record_Treatment('V001', 'Administered flu vaccine');
END;
/
-- Verify the treatment record:
SELECT * FROM ADMIN_USER.TreatmentHistory WHERE VisitID = 'V001';
```

```
-----------------------------------------------------------------
-- Constraint Testing
-----------------------------------------------------------------
-- These test cases check data integrity constraints on the underlying tables.
-- They should be executed while connected as ADMIN_USER.
-----------------------------------------------------------------

-- Test Case 1: Insert duplicate email in Patient (should fail)
BEGIN
   INSERT INTO ADMIN_USER.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);
   DBMS_OUTPUT.PUT_LINE('Constraint Test Failed: Duplicate email inserted.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Constraint Test Passed: Duplicate email not allowed
– ' || SQLERRM);
END;
/

-- Test Case 2: Appointment in the past (should fail business logic)
BEGIN
   INSERT INTO ADMIN_USER.Appointment (AppointmentID, DoctorID,
AppointmentDate, AppointmentStatus)
   VALUES ('A999', 'D001', TO_DATE('2023-01-01','YYYY-MM-DD'), 'Scheduled');
   DBMS_OUTPUT.PUT_LINE('Constraint Test Failed: Past appointment inserted.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Constraint Test Passed: Past appointment insertion
blocked – ' || SQLERRM);
END;
/

-- Test Case 3: (Run as ADMIN_USER) Insert a Patient with an invalid Gender
-- (will fail because Gender must be 'Male', 'Female', or 'Other')

BEGIN

   INSERT INTO ADMIN_USER.Patient
     (PatientID, UserID, FirstName, LastName, DOB, Gender, Email, PhoneNumber,
EmergencyContac

-- Test Case 8: (Run as BILL_USER) Billing staff tries to delete a Visit (should fail)
```

```
BEGIN
   DELETE FROM ADMIN_USER.Visit
   WHERE VisitID = 'V001';
   DBMS_OUTPUT.PUT_LINE('Test Case Failed: Billing staff deleted a Visit record.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Test Case Passed: Billing staff cannot delete a Visit
record – ' || SQLERRM);
END;
/
```

---

10. User Creation & Grants:-

```
-- Enable DBMS_OUTPUT for logging messages
SET SERVEROUTPUT ON;

-------------------------------------------------------------
-- SECTION 1: Drop Existing Users (if they exist)
-------------------------------------------------------------
BEGIN
   FOR user_rec IN (SELECT username FROM dba_users
                    WHERE username IN ('ADMIN_USER', 'DOC_USER', 'BILL_USER'))
LOOP
      BEGIN
         EXECUTE IMMEDIATE 'DROP USER ' || user_rec.username || ' CASCADE';
         DBMS_OUTPUT.PUT_LINE('Dropped user: ' || user_rec.username);
      EXCEPTION
         WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error dropping user ' || user_rec.username || ': ' ||
SQLERRM);
      END;
   END LOOP;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in dropping users block: ' || SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 2: Create Users with Strong Passwords
-------------------------------------------------------------
```

```
BEGIN
   EXECUTE IMMEDIATE 'CREATE USER ADMIN_USER IDENTIFIED BY
"Admin@Secure#1234"';
   DBMS_OUTPUT.PUT_LINE('Created user: ADMIN_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating ADMIN_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE USER DOC_USER IDENTIFIED BY
"Doctor@Secure#1234"';
   DBMS_OUTPUT.PUT_LINE('Created user: DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating DOC_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE USER BILL_USER IDENTIFIED BY
"Billing@Secure#1234"';
   DBMS_OUTPUT.PUT_LINE('Created user: BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating BILL_USER: ' || SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 3: Grant Basic Database Access (CONNECT, RESOURCE)
-------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'GRANT CONNECT, RESOURCE TO ADMIN_USER';
   DBMS_OUTPUT.PUT_LINE('Granted CONNECT, RESOURCE to ADMIN_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting CONNECT, RESOURCE to
ADMIN_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT CONNECT, RESOURCE TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted CONNECT, RESOURCE to DOC_USER');
```

```
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting CONNECT, RESOURCE to
DOC_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT CONNECT, RESOURCE TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted CONNECT, RESOURCE to BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting CONNECT, RESOURCE to
BILL_USER: ' || SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 4: Grant Administrative Privileges to ADMIN_USER
-------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'GRANT CREATE SESSION, CREATE TABLE, CREATE
VIEW, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO
ADMIN_USER';
   DBMS_OUTPUT.PUT_LINE('Granted DDL privileges to ADMIN_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting DDL privileges to ADMIN_USER: ' ||
SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT CREATE USER, ALTER USER, DROP USER TO
ADMIN_USER';
   DBMS_OUTPUT.PUT_LINE('Granted user management privileges to
ADMIN_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting user management privileges to
ADMIN_USER: ' || SQLERRM);
END;
/

BEGIN
```

```
   EXECUTE IMMEDIATE 'GRANT CREATE ROLE, GRANT ANY ROLE TO
ADMIN_USER';
   DBMS_OUTPUT.PUT_LINE('Granted role management privileges to
ADMIN_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting role management privileges to
ADMIN_USER: ' || SQLERRM);
END;
/

ALTER USER admin_user QUOTA UNLIMITED ON DATA;
GRANT CREATE PUBLIC SYNONYM TO ADMIN_USER;

-------------------------------------------------------------
-- SECTION 5: Grant Privileges to DOC_USER (Doctor Role)
-------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'GRANT CREATE SESSION TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted CREATE SESSION to DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting CREATE SESSION to DOC_USER: '
|| SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT, INSERT, UPDATE ON
ADMIN_USER.Patient TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted DML privileges on Patient to DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting privileges on Patient to DOC_USER: '
|| SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT, INSERT, UPDATE ON
ADMIN_USER.Appointment TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted DML privileges on Appointment to
DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Error granting privileges on Appointment to
DOC_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT, INSERT, UPDATE ON
ADMIN_USER.MedicalRecord TO DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted DML privileges on MedicalRecord to
DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting privileges on MedicalRecord to
DOC_USER: ' || SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT ON ADMIN_USER.Doctor TO
DOC_USER';
   DBMS_OUTPUT.PUT_LINE('Granted SELECT on Doctor to DOC_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting SELECT on Doctor to DOC_USER: '
|| SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 6: Grant Privileges to BILL_USER (Billing Role)
-------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'GRANT CREATE SESSION TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted CREATE SESSION to BILL_USER');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting CREATE SESSION to BILL_USER: '
|| SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'GRANT SELECT, INSERT, UPDATE ON
ADMIN_USER.Billing TO BILL_USER';
   DBMS_OUTPUT.PUT_LINE('Granted DML privileges on Billing to BILL_USER');
EXCEPTION
```

```
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error granting privileges on Billing to BILL_USER:
' || SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 7: Prevent Unassigned Privileges (Revoke DROP ANY TABLE if granted)
-------------------------------------------------------------
DECLARE
   v_count NUMBER;
BEGIN
   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';
      DBMS_OUTPUT.PUT_LINE('Revoked DROP ANY TABLE from DOC_USER');
   ELSE
      DBMS_OUTPUT.PUT_LINE('No DROP ANY TABLE privilege found for
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';
      DBMS_OUTPUT.PUT_LINE('Revoked DROP ANY TABLE from BILL_USER');
   ELSE
      DBMS_OUTPUT.PUT_LINE('No DROP ANY TABLE privilege found for
BILL_USER');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in revoking privileges: ' || SQLERRM);
END;
/

-------------------------------------------------------------
-- SECTION 8: Create Public Synonyms (Accessible by all users)
-------------------------------------------------------------
-- These commands must be executed while connected as ADMIN_USER
BEGIN
   EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM Patient FOR
ADMIN_USER.Patient';
   DBMS_OUTPUT.PUT_LINE('Created public synonym: Patient');
EXCEPTION
```

```
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for Patient: ' ||
SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM Doctor FOR
ADMIN_USER.Doctor';
   DBMS_OUTPUT.PUT_LINE('Created public synonym: Doctor');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for Doctor: ' ||
SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM Appointment FOR
ADMIN_USER.Appointment';
   DBMS_OUTPUT.PUT_LINE('Created public synonym: Appointment');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for Appointment: ' ||
SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM MedicalRecord FOR
ADMIN_USER.MedicalRecord';
   DBMS_OUTPUT.PUT_LINE('Created public synonym: MedicalRecord');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for MedicalRecord: '
|| SQLERRM);
END;
/

BEGIN
   EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM Billing FOR
ADMIN_USER.Billing';
   DBMS_OUTPUT.PUT_LINE('Created public synonym: Billing');
EXCEPTION
   WHEN OTHERS THEN
```

```
      DBMS_OUTPUT.PUT_LINE('Error creating public synonym for Billing: ' ||
SQLERRM);
END;
/
```

[ERD Diagram inserted here based on database schema – includes all entities, relationships, keys, and constraints]

11. Triggers & Indexes (NEW)

To enhance data integrity and performance, the system includes the following triggers and indexes:

• Trigger: Auto-update visit status after appointment completion

```
CREATE OR REPLACE TRIGGER trg_update_visit_status
AFTER UPDATE OF AppointmentStatus ON Appointment
FOR EACH ROW
BEGIN
   IF :NEW.AppointmentStatus = 'Completed' THEN
      UPDATE Visit SET VisitStatus = 'Completed'
      WHERE AppointmentID = :NEW.AppointmentID;
   END IF;
END;
```

• Trigger: Prevent deletion of doctors with existing appointments

```
CREATE OR REPLACE TRIGGER trg_prevent_doctor_delete
BEFORE DELETE ON Doctor
FOR EACH ROW
DECLARE
   v_count INTEGER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM Appointment WHERE DoctorID =
:OLD.DoctorID;
   IF v_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot delete doctor with scheduled
appointments.');
   END IF;
END;
```

• Indexes:

```
- CREATE INDEX idx_visit_date ON Visit(VisitDate);
```

- CREATE INDEX idx_patient_email ON Patient(Email);
- CREATE INDEX idx_doctor_specialization ON Doctor(Specialization);

12. Sample Outputs & Query Results (NEW)

Examples of the output produced by views and stored procedures:

• View: Doctor_Availability

DoctorID	FirstName	LastName	Specialization	Availability
D001	John	Smith	Cardiology	Available

• Procedure Execution: Update_Visit_Status('V001', 'Completed')
→ Affected row: Visit V001 marked as Completed.

• View: Billing_Insights

BillID	PatientName	VisitDate	TotalAmount	PaymentStatus
B002	Bob Miller	2024-03-21	180.00	Paid

Phase-2 Documentation for H.E.A.L. (Healthcare Efficiency & Assistance Log)
1. H.E.A.L ERD:-
[ERD Diagram inserted here based on database schema – includes all entities, relationships, keys, and constraints]
11. Triggers & Indexes (NEW)

To enhance data integrity and performance, the system includes the following triggers and indexes:

• Trigger: Auto-update visit status after appointment completion

```
CREATE OR REPLACE TRIGGER trg_update_visit_status
AFTER UPDATE OF AppointmentStatus ON Appointment
FOR EACH ROW
BEGIN
   IF :NEW.AppointmentStatus = 'Completed' THEN
      UPDATE Visit SET VisitStatus = 'Completed'
      WHERE AppointmentID = :NEW.AppointmentID;
   END IF;
END;
```

• Trigger: Prevent deletion of doctors with existing appointments

```
CREATE OR REPLACE TRIGGER trg_prevent_doctor_delete
BEFORE DELETE ON Doctor
```

```
FOR EACH ROW
DECLARE
   v_count INTEGER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM Appointment WHERE DoctorID =
:OLD.DoctorID;
   IF v_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Cannot delete doctor with scheduled
appointments.');
   END IF;
END;
```

• Indexes:
- CREATE INDEX idx_visit_date ON Visit(VisitDate);
- CREATE INDEX idx_patient_email ON Patient(Email);
- CREATE INDEX idx_doctor_specialization ON Doctor(Specialization);

12. Sample Outputs & Query Results (NEW)

Examples of the output produced by views and stored procedures:

• View: Doctor_Availability

DoctorID	FirstName	LastName	Specialization	Availability
D001	John	Smith	Cardiology	Available

• Procedure Execution: Update_Visit_Status('V001', 'Completed')
→ Affected row: Visit V001 marked as Completed.

• View: Billing_Insights

BillID	PatientName	VisitDate	TotalAmount	PaymentStatus
B002	Bob Miller	2024-03-21	180.00	Paid

13. Additional Views (NEW)

• Doctor_Only_Patient_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'))
);
```

• Billing_Only_View:

```
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;
```

14. Additional Procedures (NEW)

• 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;
```

15. Test Cases for Security & Constraints (NEW)

• Doctor tries to update billing:
→ Should trigger an exception.

• Attempt to insert duplicate patient email:

→ Constraint violation (unique email).

• Appointment scheduled in the past:
→ Should fail due to business logic validation.

16. Extended User Grants (NEW)

• Admin User
- CREATE USER admin_user IDENTIFIED BY "Admin@Secure#1234";
- Full access to database objects, roles, and configurations.

• Doctor User
- Access to Patient, Appointment, and MedicalRecord tables (SELECT, INSERT, UPDATE).

• Billing User
- Access to Billing table only (SELECT, INSERT, UPDATE).

• Privilege Safety Check
- Prevent misuse of 'DROP ANY TABLE' using PL/SQL block.

DATA FLOW DIAGRAM: -