College of Saint Benedict & Saint John's University

Computer Science Department

CSCI 331 Final Project Phase III

Healthcare Management System

Group 3

Team Members: Matt DeRosa, Max O'Brien, Ellie Smith, Mason Meyer, and Evan Quinn

May 7, 2024

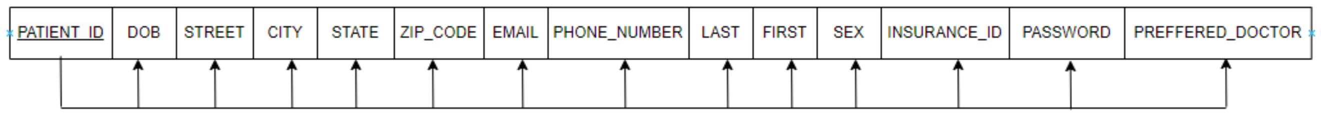
Table of Contents

Introduction

This Healthcare Management System is designed to facilitate various operations within a healthcare setting, allowing different types of users to interact with the system through a graphical user interface (GUI). Each user type (Patient, Doctor, Pharmacy, Insurance Company, etc.) has a dedicated menu that offers specific functionalities relevant to their role.

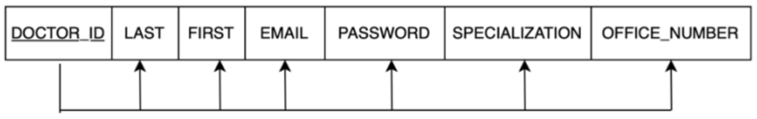
Normalization Analysis

PATIENT Table



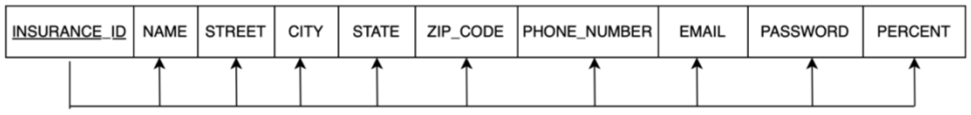
I. PATIENT is in 1NF because PATIENT_ID is the candidate key; it is minimal and derives all other attributes: PATIENT_ID$^+$ = {PATIENT_ID, DOB, STREET, CITY, STATE, ZIP_CODE, EMAIL, PHONE_NUMBER, LAST, FIRST, SEX, INSURANCE_ID, PASSWORD, PREFERRED_DOCTOR}.
II. PATIENT is in 2NF because all non-prime attributes are dependent on PATIENT_ID (single-attribute candidate key).
III. PATIENT is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
  a. Attribute preservation: yes
  b. Dependency preservation: yes
  c. Lossless join: yes

DOCTOR Table



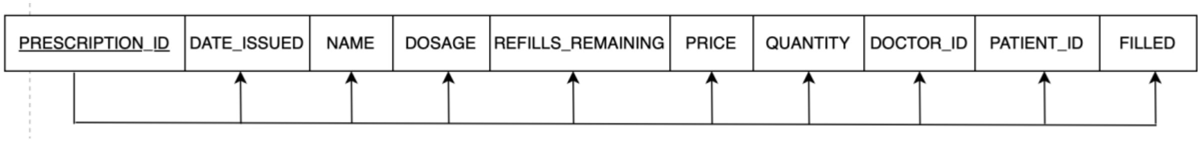
I. DOCTOR is in 1NF because DOCTOR_ID is the candidate key; it is minimal and derives all other attributes: DOCTOR_ID$^+$ = {DOCTOR_ID, LAST, FIRST, EMAIL, PASSWORD, SPECIALIZATION, OFFICE_NUMBER}.
II. DOCTOR is in 2NF because all non-prime attributes are dependent on DOCTOR_ID (single-attribute candidate key).
III. DOCTOR is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
  a. Attribute preservation: yes
  b. Dependency preservation: yes
  c. Lossless join: yes

INSURANCECOMPANY Table



I. INSURANCECOMPANY is in 1NF because INSURANCE_ID is the candidate key; it is minimal and derives all other attributes: INSURANCE _ID$^+$ = {INSURANCE_ID, NAME, STREET, CITY, STATE, ZIP_CODE, PHONE_NUMBER, EMAIL, PASSWORD, PERCENT}.
II. INSURANCECOMPANY is in 2NF because all non-prime attributes are dependent on INSURANCE _ID (single-attribute candidate key).
III. INSURANCECOMPANY is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes

PRESCRIPTION Table

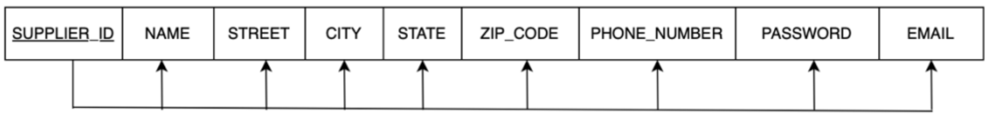


DOCTOR_ID is a FK to DOCTOR_ID in DOCTOR Table

PATIENT_ID is a FK to PATIENT_ID in PATIENT Table

I. PRESCRIPTION is in 1NF because PRESCRIPTION_ID is the candidate key; it is minimal and derives all other attributes: PRESCRIPTION _ID$^+$ = {PRESCRIPTION_ID, DATE_ISSUED, NAME, DOSAGE, REFILLS_REMAINING, PRICE, QUANTITY, DOCTOR_ID, PATIENT_ID, FILLED}.
II. PRESCRIPTION is in 2NF because all non-prime attributes are dependent on PRESCRIPTION_ID (single-attribute candidate key).
III. PRESCRIPTION is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes

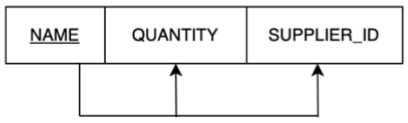
SUPPLIER Table



I. SUPPLIER is in 1NF because SUPPLIER _ID is the candidate key; it is minimal and derives all other attributes: SUPPLIER _ID$^+$ = {SUPPLIER_ID, NAME, STREET, CITY, STATE, ZIP_CODE, PHONE_NUMBER, PASSWORD, EMAIL}.
II. SUPPLIER is in 2NF because all non-prime attributes are dependent on SUPPLIER _ID (single-attribute candidate key).

III. SUPPLIER is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
  a. Attribute preservation: yes
  b. Dependency preservation: yes
  c. Lossless join: yes

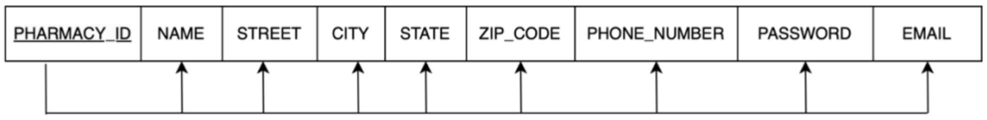
MEDICATION Table



SUPPLIER_ID is a FK to SUPPLIER _ID in SUPPLIER Table

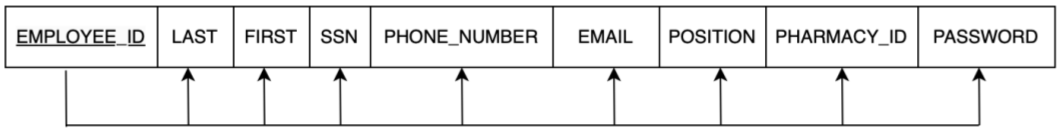
I. MEDICATION is in 1NF because NAME is the candidate key; it is minimal and derives all other attributes: NAME$^+$ = {NAME, QUANTITY, SUPPLIER_ID}.
II. MEDICATION is in 2NF because all non-prime attributes are dependent on NAME (single-attribute candidate key).
III. MEDICATION is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
  a. Attribute preservation: yes
  b. Dependency preservation: yes
  c. Lossless join: yes
V. Improvement: NAME & SUPPLIER_ID as PK or MEDICATION_ID to allow for medication to be supplied by multiple suppliers.

PHARMACY Table



I. PHARMACY is in 1NF because PHARMACY_ID is the candidate key; it is minimal and derives all other attributes: PHARMACY _ID$^+$ = { PHARMACY _ID, NAME, STREET, CITY, STATE, ZIP_CODE, PHONE_NUMBER, PASSWORD, EMAIL}.
II. PHARMACY is in 2NF because all non-prime attributes are dependent on PHARMACY_ID (single-attribute candidate key).
III. PHARMACY is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
  a. Attribute preservation: yes
  b. Dependency preservation: yes
  c. Lossless join: yes

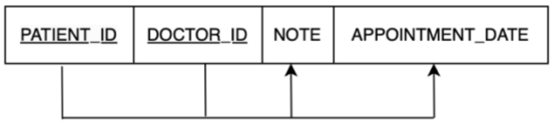
PHARMACYEMPLOYEE Table



PHARMACY_ID is a FK to PHARMACY _ID in PHARMACY Table

I. PHARMACYEMPLOYEE is in 1NF because EMPLOYEE_ID is the candidate key; it is minimal and derives all other attributes: EMPLOYEE _ID$^+$ = {EMPLOYEE _ID, LAST, FIRST, SSN, PHONE_NUMBER, EMAIL, POSITION, PHARMACY_ID, PASSWORD}.
II. PHARMACYEMPLOYEE is in 2NF because all non-prime attributes are dependent on EMPLOYEE _ID (single-attribute candidate key).
III. PHARMACYEMPLOYEE is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes

APPOINTMENT Table

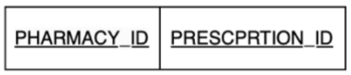


PATIENT_ID is a FK to PATIENT _ID in PATIENT Table

DOCTOR_ID is a FK to DOCTOR _ID in DOCTOR Table

I. APPOINTMENT is in 1NF because {PATIENT_ID, DOCTOR_ID} is the candidate key.
    a. It derives all other attributes: {PATIENT_ID, DOCTOR_ID}$^+$ = {PATIENT_ID, DOCTOR_ID, NOTE, APPOINTMENT_DATE}.
    b. It is minimal: PATIENT _ID$^+$ = {PATIENT_ID} and DOCTOR _ID$^+$ = {DOCTOR _ID}
II. PHARMACYEMPLOYEE is in 2NF because all non-prime attributes are dependent on {PATIENT_ID, DOCTOR_ID}.
III. PHARMACYEMPLOYEE is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes
V. Improvement: this setup currently only allows for one note per patient/doctor combination which is a limitation.

FILLS Table



I. FILLS is in 1NF because {PHARMACY_ID, PRESCRIPTION_ID} is the candidate key.

   a. It derives all other attributes: {PHARMACY_ID, PRESCRIPTION_ID}$^+$ = {PHARMACY_ID, PRESCRIPTION_ID}.
   b. It is minimal: PHARMACY_ID$^+$ = {PHARMACY_ID } and PRESCRIPTION_ID$^+$ = {PRESCRIPTION_ID}

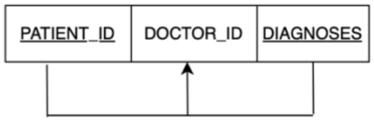
II. FILLS is in 2NF because there are no non-prime attributes.

III. FILLS is in 3NF because there are no non-prime attributes dependent on non-key attributes.

IV. Properties:
   a. Attribute preservation: yes
   b. Dependency preservation: yes
   c. Lossless join: yes

V. Improvement: this table could have been avoided if we just put the PHARMACY_ID attribute in the PRESCPRTION Table.

DIAGNOSES Table



PATIENT_ID is a FK to PATIENT _ID in PATIENT Table

DOCTOR_ID is a FK to DOCTOR _ID in DOCTOR Table

I. DIAGNOSES is in 1NF because {DIAGNOSES, PATIENT_ID} is the candidate key.
   a. It derives all other attributes: {DIAGNOSES, PATIENT_ID}$^+$ = {DIAGNOSES, PATIENT_ID, DOCTOR_ID}.
   b. It is minimal: PATIENT _ID$^+$ = {PATIENT_ID} and DIAGNOSES$^+$ = {DIAGNOSES}

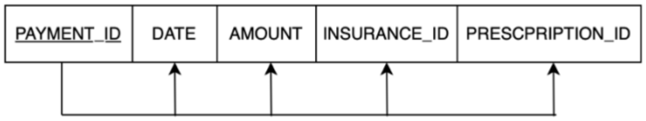
II. DIAGNOSES is in 2NF because all non-prime attributes are dependent on {DIAGNOSES, PATIENT_ID}.

III. DIAGNOSES is in 3NF because there are no non-prime attributes dependent on non-key attributes.

IV. Properties:
   a. Attribute preservation: yes
   b. Dependency preservation: yes
   c. Lossless join: yes

V. Improvement: this setup does not allow a given patient to be diagnosed with the same thing twice

INSURANCEPAYMENT Table



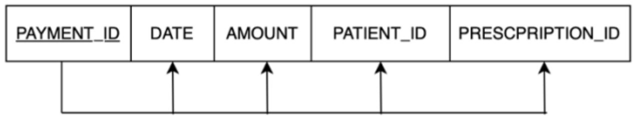
INSURANCE_ID is a FK to INSURANCE _ID in INSURANCE Table

PRESCRIPTION_ID is a FK to PRESCRIPTION _ID in PRESCRIPTION Table

I. INSURANCEPAYMENT is in 1NF because PAYMENT_ID is the candidate key; it is minimal and derives all other attributes: PAYMENT _ID$^+$ = { PAYMENT _ID, DATE, AMOUNT, INSURANCE_ID, PRESCRIPTION_ID}.

II. INSURANCEPAYMENT is in 2NF because all non-prime attributes are dependent on PAYMENT _ID (single-attribute candidate key).
III. INSURANCEPAYMENT is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes

PATIENTPAYMENT Table

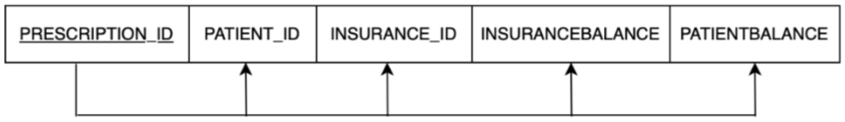


PATIENT_ID is a FK to PATIENT _ID in PATIENT Table

PRESCRIPTION_ID is a FK to PRESCRIPTION _ID in PRESCRIPTION Table

I. PATIENTPAYMENT is in 1NF because PAYMENT_ID is the candidate key; it is minimal and derives all other attributes: PAYMENT $\_ID^+$ = {PAYMENT _ID, DATE, AMOUNT, PATIENT_ID, PRESCRIPTION_ID}.
II. PATIENTPAYMENT is in 2NF because all non-prime attributes are dependent on PAYMENT_ID (single-attribute candidate key).
III. PATIENTPAYMENT is in 3NF because there are no non-prime attributes dependent on non-key attributes.
IV. Properties:
    a. Attribute preservation: yes
    b. Dependency preservation: yes
    c. Lossless join: yes

PRESCRIPTIONBALANCE Table



PRESCRIPTION_ID is a FK to PRESCRIPTION _ID in PRESCRIPTION Table

PATIENT_ID is a FK to PATIENT _ID in PATIENT Table

INSURANCE_ID is a FK to INSURANCE _ID in INSURANCE Table

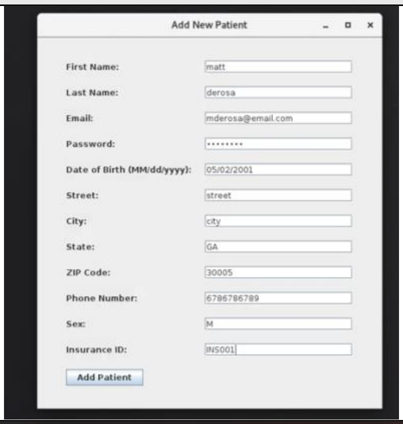
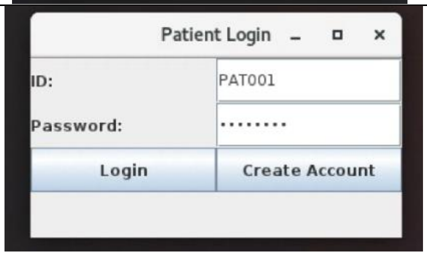
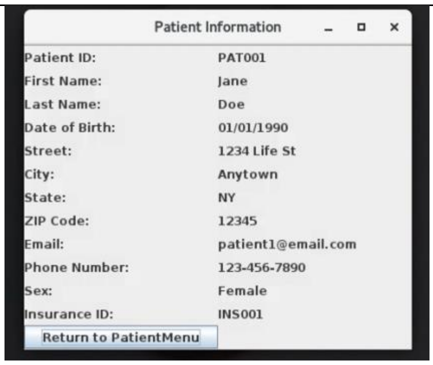
Note: this table was created to split prescription price into INSURANCEBALANCE and PATIENTBALANCE based on the percentage each patient's insurance paid for their prescriptions.

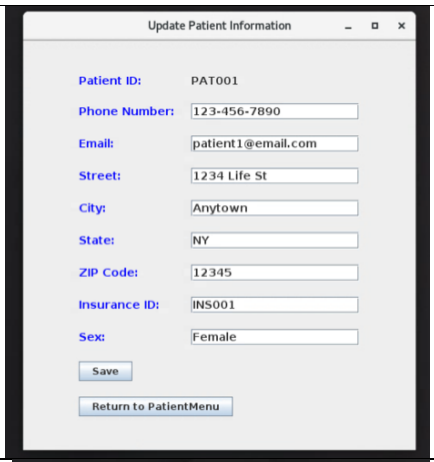
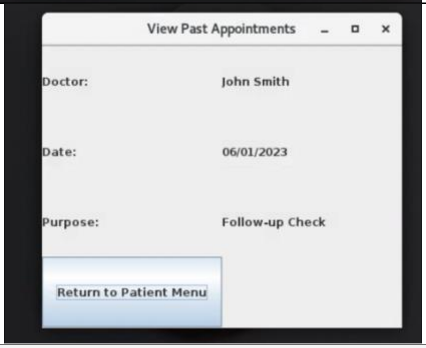
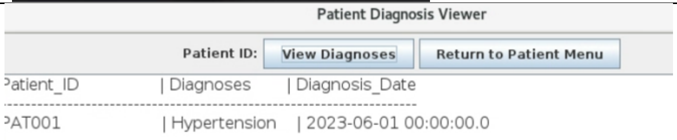
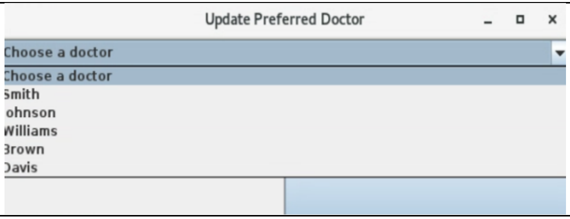
I. PRESCRIPTIONBALANCE is in 1NF because PRESCRIPTION _ID is the candidate key; it is minimal and derives all other attributes: PRESCRIPTION $\_ID^+$ = {PRESCRIPTION _ID, PATIENT_ID, INSURANCE_ID, INSURANCEBALANCE, PATIENTBALANCE}.
II. PRESCRIPTIONBALANCE is in 2NF because all non-prime attributes are dependent on PRESCRIPTION _ID (single-attribute candidate key).

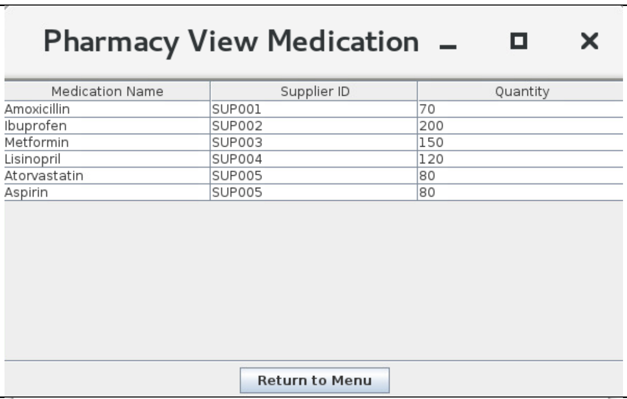
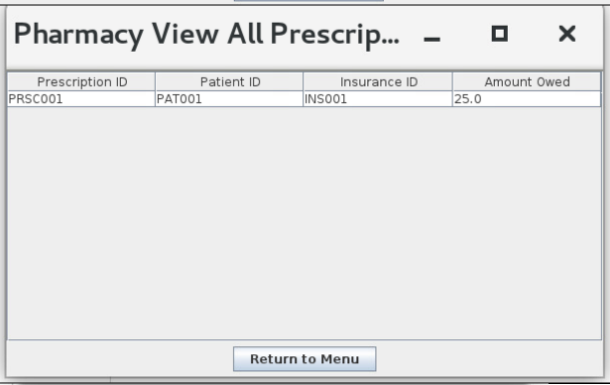
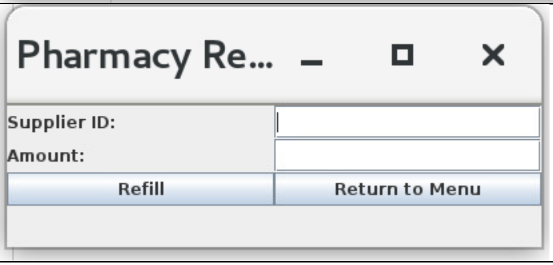
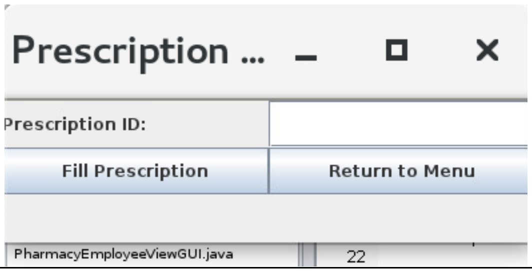
III. PRESCRIPTIONBALANCE is in 3NF because there are no non-prime attributes dependent on non-key attributes.

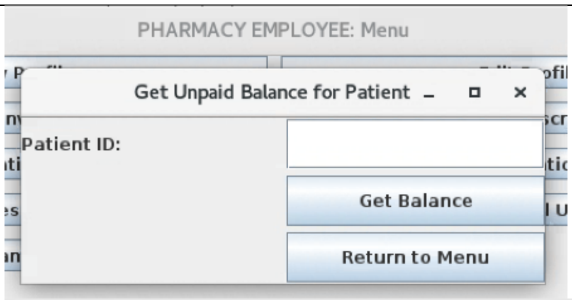
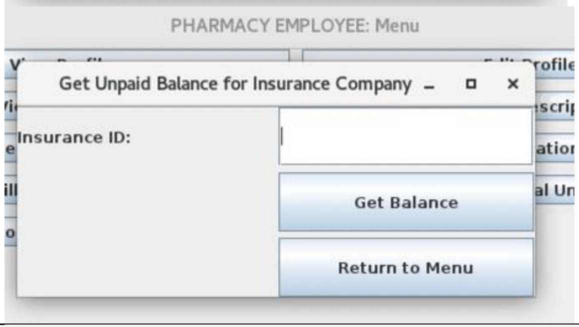
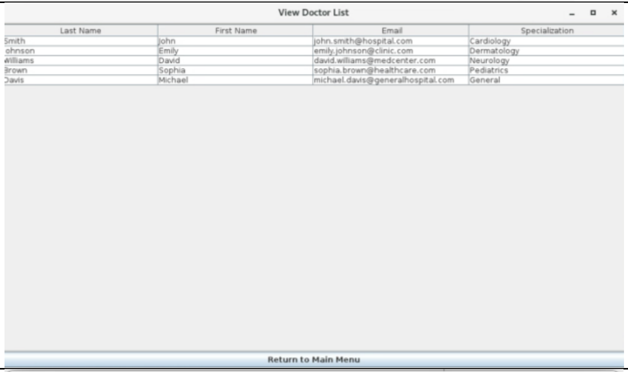
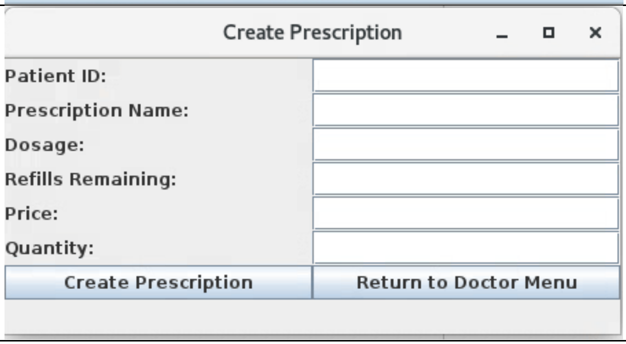
IV. Properties:
   a. Attribute preservation: yes
   b. Dependency preservation: yes
   c. Lossless join: yes

Functionalities Table

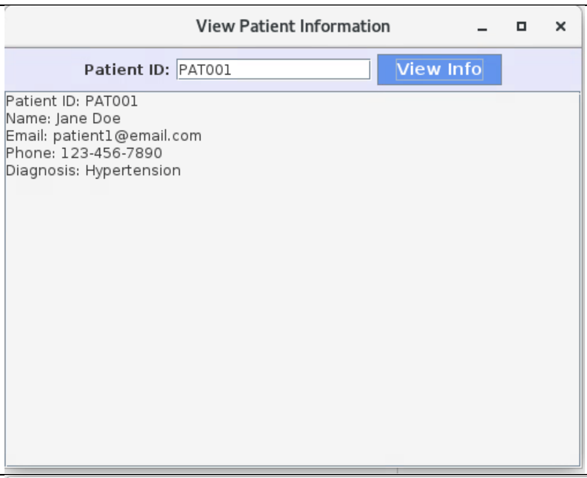
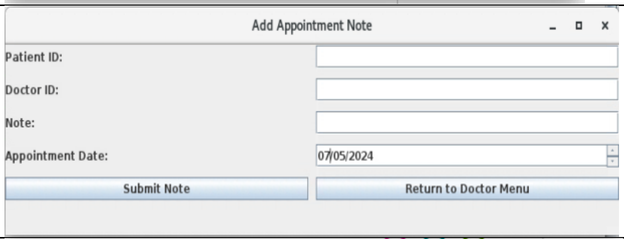
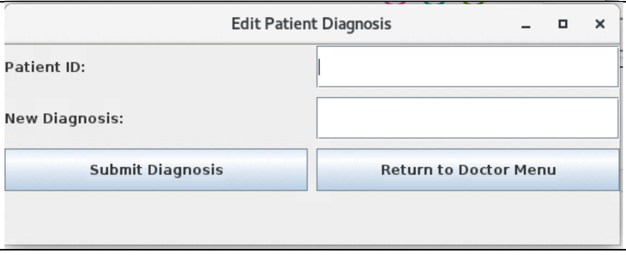
Proposed Functionality	**Member Responsible**	**Brief Description**	**Sample User Interface with data included**	**Successfully Implemented YES or NO (If no, explain why)**
ALL USERS: Create Account	Matt	Allows new users (of all types) to create an account on the management database	 Add New Patient First Name: matt Last Name: derosa Email: mderosa@email.com Password: •••••••• Date of Birth (MM/dd/yyyy): 05/02/2001 Street: street City: city State: GA ZIP Code: 30005 Phone Number: 6786786789 Sex: M Insurance ID: INS001 	Yes working.
ALL USERS: Login	Matt	Allows users of all type to login to the software and will direct them to a page depending on what type of user they are	 Patient Login ID: PAT001 Password: •••••••• 	Yes working
ALL USERS: View/edit profile	Matt	Allow patients to view and edit personal information including insurance information and their primary doctor. Patient Id, DOB, Last, and First and not able to be edited.	 Patient Information Patient ID: PAT001 First Name: Jane Last Name: Doe Date of Birth: 01/01/1990 Street: 1234 Life St City: Anytown State: NY ZIP Code: 12345 Email: patient1@email.com Phone Number: 123-456-7890 Sex: Female Insurance ID: INS001 	Yes view return patient object, edit returns patient object with null fields for uneditables

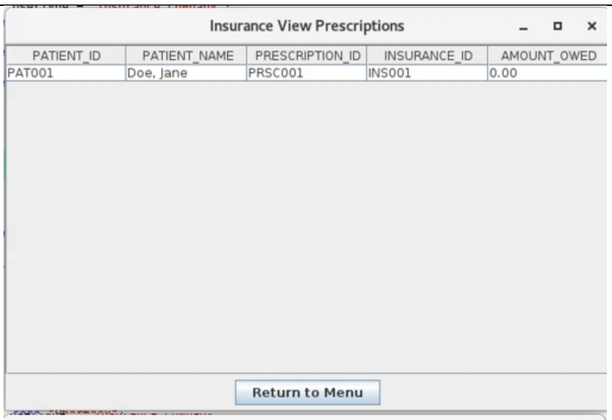
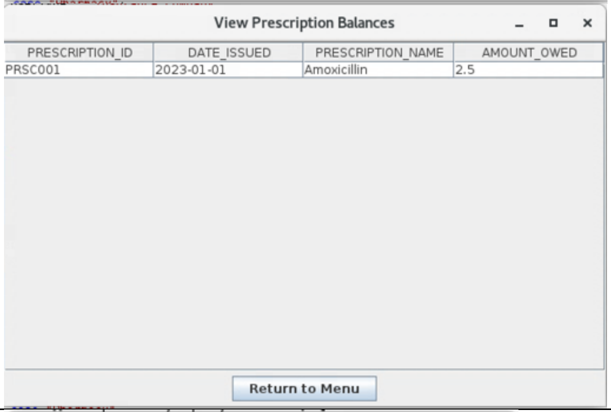
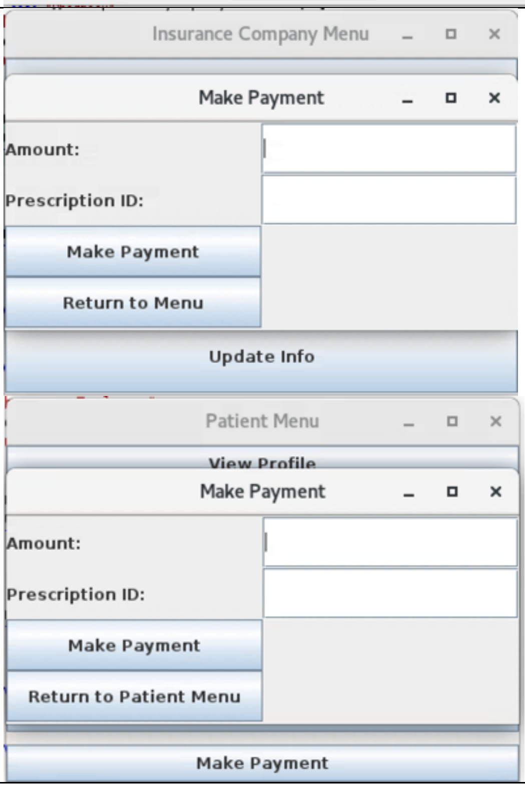
			 Update Patient Information Patient ID: PAT001 Phone Number: 123-456-7890 Email: patient1@email.com Street: 1234 Life St City: Anytown State: NY ZIP Code: 12345 Insurance ID: INS001 Sex: Female Save Return to PatientMenu	
PATIENT: View Appointment Info	Matt	Allows the patient to see the doctor ID number, patient ID number, date of consultation, doctor's notes (in patient description)	 View Past Appointments Doctor: John Smith Date: 06/01/2023 Purpose: Follow-up Check Return to Patient Menu	Yes working in java tests. Returns a list of type appointmentdetials.
PATIENT: View Diagnoses	Max	Patients can view diagnoses that doctors have added to their profile.	 Patient Diagnosis Viewer Patient ID: View Diagnoses Return to Patient Menu Patient_ID | Diagnoses | Diagnosis_Date PAT001 | Hypertension | 2023-06-01 00:00:00.0	Yes
PATIENT: Select Preferred Doctor	Max	Allow patients to view the complete list of doctors and their speci	 Update Preferred Doctor Choose a doctor Choose a doctor Smith Johnson Williams Brown Davis	Yes

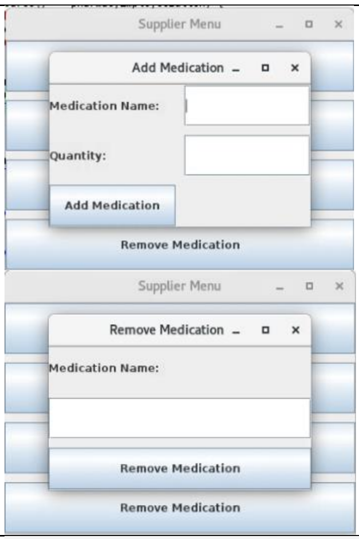
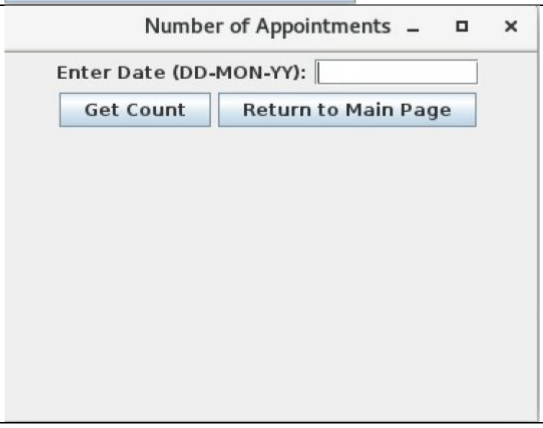
PHARMACY EMPLOYEE: View Inventory	Evan	View all drugs currently available in the pharmacy.	 Pharmacy View Medication Medication Name / Supplier ID / Quantity Amoxicillin / SUP001 / 70 Ibuprofen / SUP002 / 200 Metformin / SUP003 / 150 Lisinopril / SUP004 / 120 Atorvastatin / SUP005 / 80 Aspirin / SUP005 / 80 Return to Menu	Yes
PHARMACY EMPLOYEE: View prescriptions and unpaid balances for a patient	Evan	Allows pharmacy employees to view all prescriptions showing unpaid balances.	 Pharmacy View All Prescrip... Prescription ID / Patient ID / Insurance ID / Amount Owed PRSC001 / PAT001 / INS001 / 25.0 Return to Menu	Yes
PHARMACY EMPLOYEE: Update medication supply	Evan	Allow pharmacy employees to change available quantity of medications.	 Pharmacy Re... Supplier ID: Amount: Refill Return to Menu	Yes
PHARMACY EMPLOYEE: Fill prescriptions	Evan	Allows pharmacy employees to set prescription "FILLED" attribute to YES	Used alter table statements  Prescription ... Prescription ID: Fill Prescription Return to Menu PharmacyEmployeeViewGUI.java	Yes

PHARMACY EMPLOYEE: View patient/insurance company's total unpaid balance	Ellie	Allows pharmacy employees to see total unpaid balances after entering a patient/insurance ID.	 PHARMACY EMPLOYEE: Menu Get Unpaid Balance for Patient Patient ID: Get Balance Return to Menu  PHARMACY EMPLOYEE: Menu Get Unpaid Balance for Insurance Company Insurance ID: Get Balance Return to Menu	Yes
PATIENT: View list of all doctors, their info, and specialties	Max	See a list of all doctors in the database so patients can see what doctor is best to schedule with.		Yes
DOCTOR: Create prescription	Mason	Allows doctors to create prescriptions for their patients.	 Create Prescription Patient ID: Prescription Name: Dosage: Refills Remaining: Price: Quantity: Create Prescription Return to Doctor Menu	Yes

Last Name	First Name	Email	Specialization
Smith	John	john.smith@hospital.com	Cardiology
ohnson	Emily	emily.johnson@clinic.com	Dermatology
Williams	David	david.williams@medcenter.com	Neurology
Brown	Sophia	sophia.brown@healthcare.com	Pediatrics
Davis	Michael	michael.davis@generalhospital.com	General

DOCTOR: View patient info	Mason	Allow doctor users to view certain information from their patients' profiles and view their diagnoses.	 View Patient Information Patient ID: PAT001 View Info Patient ID: PAT001 Name: Jane Doe Email: patient1@email.com Phone: 123-456-7890 Diagnosis: Hypertension	Yes
DOCTOR: Add appointment note	Mason	Allows doctors to leave appointment note and date after seeing a patient.	 Add Appointment Note Patient ID: Doctor ID: Note: Appointment Date: 07/05/2024 Submit Note Return to Doctor Menu	Yes
DOCTOR: Edit patient diagnoses.	Mason	Allows doctors to edit diagnosis for a patient.	 Edit Patient Diagnosis Patient ID: New Diagnosis: Submit Diagnosis Return to Doctor Menu	Yes

INSURANCE COMPANY & PATIENT: View their prescriptions and how much they owe for each	Ellie	Insurance company and patients can view a list of patients they cover along with their unpaid insurance prescription balance.	 Insurance View Prescriptions PATIENT_ID | PATIENT_NAME | PRESCRIPTION_ID | INSURANCE_ID | AMOUNT_OWED PAT001 | Doe, Jane | PRSC001 | INS001 | 0.00  View Prescription Balances PRESCRIPTION_ID | DATE_ISSUED | PRESCRIPTION_NAME | AMOUNT_OWED PRSC001 | 2023-01-01 | Amoxicillin | 2.5 	Yes
INSURANCE COMPANY & PATIENT: Pay balance on prescriptions	Ellie	Insurance company and patients can make payments on patients' prescriptions.	 Insurance Company Menu Make Payment Amount: Prescription ID: Make Payment Return to Menu Update Info Patient Menu Make Payment Amount: Prescription ID: Make Payment Return to Patient Menu Make Payment	Yes

SUPPLIER: edit medications (add/remove)	Ellie	Suppliers can add and remove medications	 Supplier Menu Add Medication Medication Name: Quantity: Add Medication Remove Medication Supplier Menu Remove Medication Medication Name: Remove Medication Remove Medication	Yes
DOCTOR: View Number of Appointments	Max	Doctors can select a date and see the number of appointments they have scheduled for the day	 Number of Appointments Enter Date (DD-MON-YY): Get Count Return to Main Page	

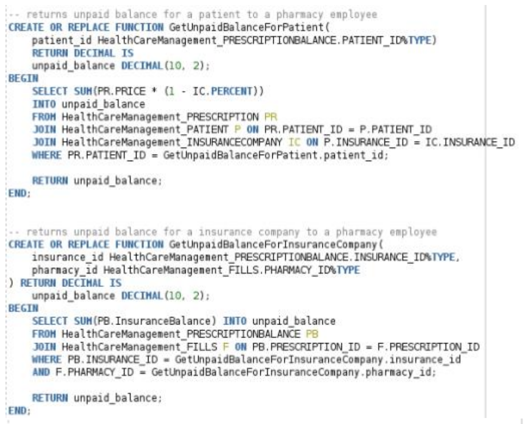
Updated Stored Routines

Below is an explanation of each member's stored routines. Including:

A. Type: Trigger, View, Proc and Function
B. Code
C. Does it work? If so, include sample input and output
D. Functionality (from (2)) in which routine is used with screenshots to prove claim

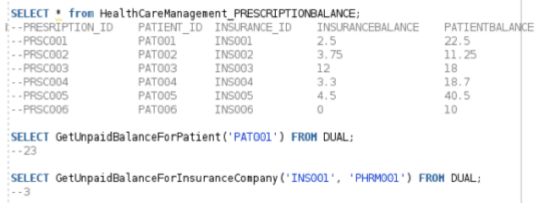
Ellie's Stored Routines

A. Functions GetUnpaidBalanceForInsuranceCompany and GetUnpaidBalanceForPatient



```
-- returns unpaid balance for a patient to a pharmacy employee
CREATE OR REPLACE FUNCTION GetUnpaidBalanceForPatient(
    patient_id HealthCareManagement_PRESCRIPTIONBALANCE.PATIENT_ID%TYPE)
    RETURN DECIMAL IS
    unpaid_balance DECIMAL(10, 2);
BEGIN
    SELECT SUM(PR.PRICE * (1 - IC.PERCENT))
    INTO unpaid_balance
    FROM HealthCareManagement_PRESCRIPTION PR
    JOIN HealthCareManagement_PATIENT P ON PR.PATIENT_ID = P.PATIENT_ID
    JOIN HealthCareManagement_INSURANCECOMPANY IC ON P.INSURANCE_ID = IC.INSURANCE_ID
    WHERE PR.PATIENT_ID = GetUnpaidBalanceForPatient.patient_id;

    RETURN unpaid_balance;
END;


-- returns unpaid balance for a insurance company to a pharmacy employee
CREATE OR REPLACE FUNCTION GetUnpaidBalanceForInsuranceCompany(
    insurance_id HealthCareManagement_PRESCRIPTIONBALANCE.INSURANCE_ID%TYPE,
    pharmacy_id HealthCareManagement_FILLS.PHARMACY_ID%TYPE
) RETURN DECIMAL IS
    unpaid_balance DECIMAL(10, 2);
BEGIN
    SELECT SUM(PB.InsuranceBalance) INTO unpaid_balance
    FROM HealthCareManagement_PRESCRIPTIONBALANCE PB
    JOIN HealthCareManagement_FILLS F ON PB.PRESCRIPTION_ID = F.PRESCRIPTION_ID
    WHERE PB.INSURANCE_ID = GetUnpaidBalanceForInsuranceCompany.insurance_id
    AND F.PHARMACY_ID = GetUnpaidBalanceForInsuranceCompany.pharmacy_id;

    RETURN unpaid_balance;
END;
```

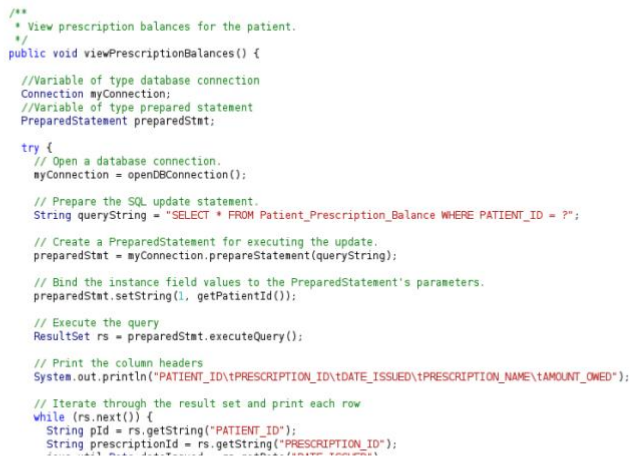
B.



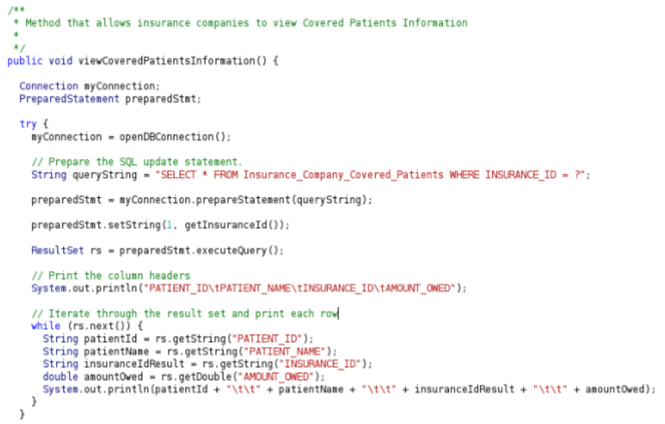
```
SELECT * from HealthCareManagement_PRESCRIPTIONBALANCE;
--PRESRIPTION_ID     PATIENT_ID  INSURANCE_ID    INSURANCEBALANCE     PATIENTBALANCE
--PRSC001             PAT001      INS001          2.5                  22.5
--PRSC002             PAT002      INS002          3.75                 11.25
--PRSC003             PAT003      INS003          12                   18
--PRSC004             PAT004      INS004          3.3                  18.7
--PRSC005             PAT005      INS005          4.5                  40.5
--PRSC006             PAT006      INS006          0                    10

SELECT GetUnpaidBalanceForPatient('PAT001') FROM DUAL;
--23

SELECT GetUnpaidBalanceForInsuranceCompany('INS001', 'PHRM001') FROM DUAL;
--3
```

C.

D. Allows pharmacy employees to view a patient/insurance company's total unpaid balance.

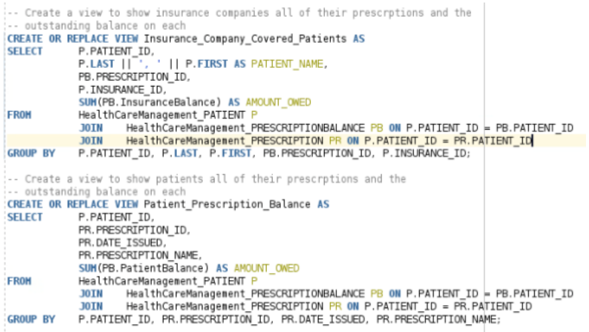


```
/**
 * View prescription balances for the patient.
 */
public void viewPrescriptionBalances() {

  //Variable of type database connection
  Connection myConnection;
  //Variable of type prepared statement
  PreparedStatement preparedStmt;

  try {
    // Open a database connection.
    myConnection = openDBConnection();

    // Prepare the SQL update statement.
    String queryString = "SELECT * FROM Patient_Prescription_Balance WHERE PATIENT_ID = ?";

    // Create a PreparedStatement for executing the update.
    preparedStmt = myConnection.prepareStatement(queryString);

    // Bind the instance field values to the PreparedStatement's parameters.
    preparedStmt.setString(1, getPatientId());

    // Execute the query
    ResultSet rs = preparedStmt.executeQuery();

    // Print the column headers
    System.out.println("PATIENT_ID\tPRESCRIPTION_ID\tDATE_ISSUED\tPRESCRIPTION_NAME\tAMOUNT_OWED");

    // Iterate through the result set and print each row
    while (rs.next()) {
      String pId = rs.getString("PATIENT_ID");
      String prescriptionId = rs.getString("PRESCRIPTION_ID");
```

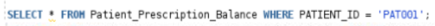


```
/**
 * Method that allows insurance companies to view Covered Patients Information
 *
 */
public void viewCoveredPatientsInformation() {

  Connection myConnection;
  PreparedStatement preparedStmt;

  try {
    myConnection = openDBConnection();

    // Prepare the SQL update statement.
    String queryString = "SELECT * FROM Insurance_Company_Covered_Patients WHERE INSURANCE_ID = ?";

    preparedStmt = myConnection.prepareStatement(queryString);

    preparedStmt.setString(1, getInsuranceId());

    ResultSet rs = preparedStmt.executeQuery();

    // Print the column headers
    System.out.println("PATIENT_ID\tPATIENT_NAME\tINSURANCE_ID\tAMOUNT_OWED");

    // Iterate through the result set and print each row
    while (rs.next()) {
      String patientId = rs.getString("PATIENT_ID");
      String patientName = rs.getString("PATIENT_NAME");
      String insuranceIdResult = rs.getString("INSURANCE_ID");
      double amountOwed = rs.getDouble("AMOUNT_OWED");
      System.out.println(patientId + "\t\t" + patientName + "\t\t" + insuranceIdResult + "\t\t" + amountOwed);
    }
  }
```

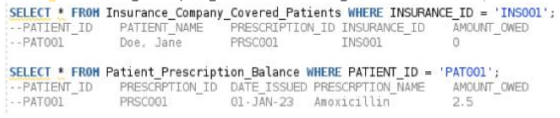
A. Views Patient_Prescription_Balance and Insurance_Company_Covered_Patients



```sql
-- Create a view to show insurance companies all of their prescrptions and the
-- outstanding balance on each
CREATE OR REPLACE VIEW Insurance_Company_Covered_Patients AS
SELECT      P.PATIENT_ID,
            P.LAST || ', ' || P.FIRST AS PATIENT_NAME,
            PB.PRESCRIPTION_ID,
            P.INSURANCE_ID,
            SUM(PB.InsuranceBalance) AS AMOUNT_OWED
FROM        HealthCareManagement_PATIENT P
            JOIN    HealthCareManagement_PRESCRIPTIONBALANCE PB ON P.PATIENT_ID = PB.PATIENT_ID
            JOIN    HealthCareManagement_PRESCRIPTION PR ON P.PATIENT_ID = PR.PATIENT_ID
GROUP BY    P.PATIENT_ID, P.LAST, P.FIRST, PB.PRESCRIPTION_ID, P.INSURANCE_ID;

-- Create a view to show patients all of their prescrptions and the
-- outstanding balance on each
CREATE OR REPLACE VIEW Patient_Prescription_Balance AS
SELECT      P.PATIENT_ID,
            PR.PRESCRIPTION_ID,
            PR.DATE_ISSUED,
            PR.PRESCRIPTION_NAME,
            SUM(PB.PatientBalance) AS AMOUNT_OWED
FROM        HealthCareManagement_PATIENT P
            JOIN    HealthCareManagement_PRESCRIPTIONBALANCE PB ON P.PATIENT_ID = PB.PATIENT_ID
            JOIN    HealthCareManagement_PRESCRIPTION PR ON P.PATIENT_ID = PR.PATIENT_ID
GROUP BY    P.PATIENT_ID, PR.PRESCRIPTION_ID, PR.DATE_ISSUED, PR.PRESCRIPTION_NAME;
```

B. 

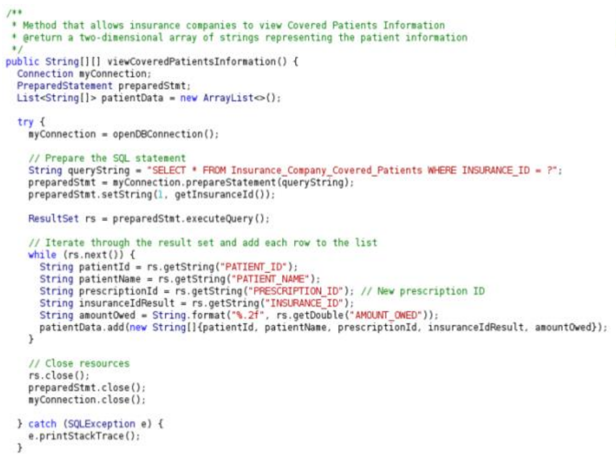
```sql
SELECT * FROM Patient_Prescription_Balance WHERE PATIENT_ID = 'PAT001';
```



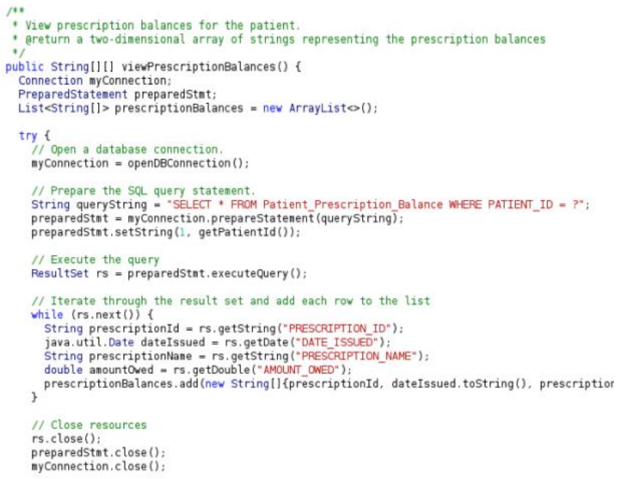
```sql
SELECT * FROM Insurance_Company_Covered_Patients WHERE INSURANCE_ID = 'INS001';
--PATIENT_ID    PATIENT_NAME    PRESCRIPTION_ID INSURANCE_ID    AMOUNT_OWED
--PAT001        Doe, Jane       PRSC001         INS001          0

SELECT * FROM Patient_Prescription_Balance WHERE PATIENT_ID = 'PAT001';
--PATIENT_ID    PRESCRPTION_ID  DATE_ISSUED PRESCRPTION_NAME    AMOUNT_OWED
--PAT001        PRSC001         01-JAN-23   Amoxicillin         2.5
```

C.

D. Allows INSURANCE COMPANY & PATIENTs to view their prescriptions and how much they owe for each.

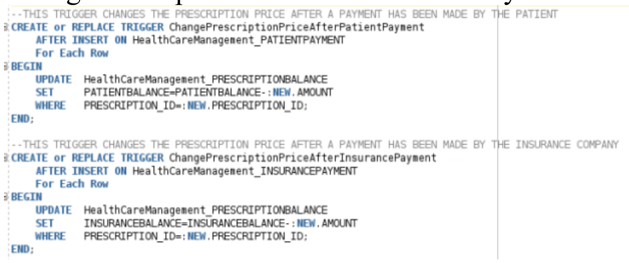


```java
/**
 * Method that allows insurance companies to view Covered Patients Information
 * @return a two-dimensional array of strings representing the patient information
 */
public String[][] viewCoveredPatientsInformation() {
  Connection myConnection;
  PreparedStatement preparedStmt;
  List<String[]> patientData = new ArrayList<>();

  try {
    myConnection = openDBConnection();

    // Prepare the SQL statement
    String queryString = "SELECT * FROM Insurance_Company_Covered_Patients WHERE INSURANCE_ID = ?";
    preparedStmt = myConnection.prepareStatement(queryString);
    preparedStmt.setString(1, getInsuranceId());

    ResultSet rs = preparedStmt.executeQuery();

    // Iterate through the result set and add each row to the list
    while (rs.next()) {
      String patientId = rs.getString("PATIENT_ID");
      String patientName = rs.getString("PATIENT_NAME");
      String prescriptionId = rs.getString("PRESCRIPTION_ID"); // New prescription ID
      String insuranceIdResult = rs.getString("INSURANCE_ID");
      String amountOwed = String.format("%.2f", rs.getDouble("AMOUNT_OWED"));
      patientData.add(new String[]{patientId, patientName, prescriptionId, insuranceIdResult, amountOwed});
    }

    // Close resources
    rs.close();
    preparedStmt.close();
    myConnection.close();

  } catch (SQLException e) {
    e.printStackTrace();
  }
```



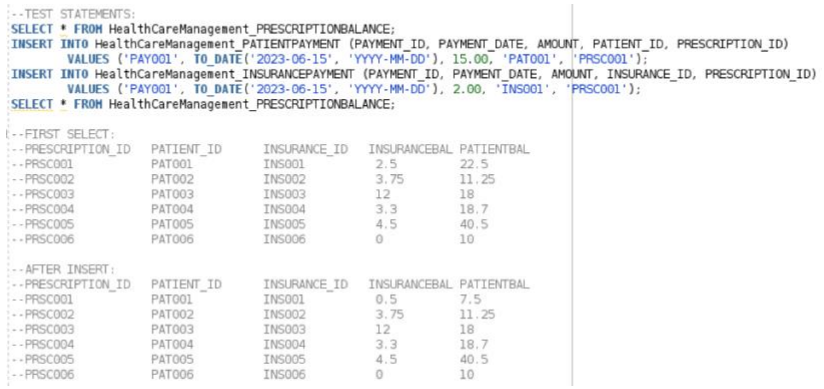
```java
/**
 * View prescription balances for the patient.
 * @return a two-dimensional array of strings representing the prescription balances
 */
public String[][] viewPrescriptionBalances() {
  Connection myConnection;
  PreparedStatement preparedStmt;
  List<String[]> prescriptionBalances = new ArrayList<>();

  try {
    // Open a database connection.
    myConnection = openDBConnection();

    // Prepare the SQL query statement.
    String queryString = "SELECT * FROM Patient_Prescription_Balance WHERE PATIENT_ID = ?";
    preparedStmt = myConnection.prepareStatement(queryString);
    preparedStmt.setString(1, getPatientId());

    // Execute the query
    ResultSet rs = preparedStmt.executeQuery();

    // Iterate through the result set and add each row to the list
    while (rs.next()) {
      String prescriptionId = rs.getString("PRESCRIPTION_ID");
      java.util.Date dateIssued = rs.getDate("DATE_ISSUED");
      String prescriptionName = rs.getString("PRESCRIPTION_NAME");
      double amountOwed = rs.getDouble("AMOUNT_OWED");
      prescriptionBalances.add(new String[]{prescriptionId, dateIssued.toString(), prescriptior
    }

    // Close resources
    rs.close();
    preparedStmt.close();
    myConnection.close();
```

A. Triggers ChangePrescriptionPriceAfterPayment and ChangePrescriptionBalanceAfterPatientPayment



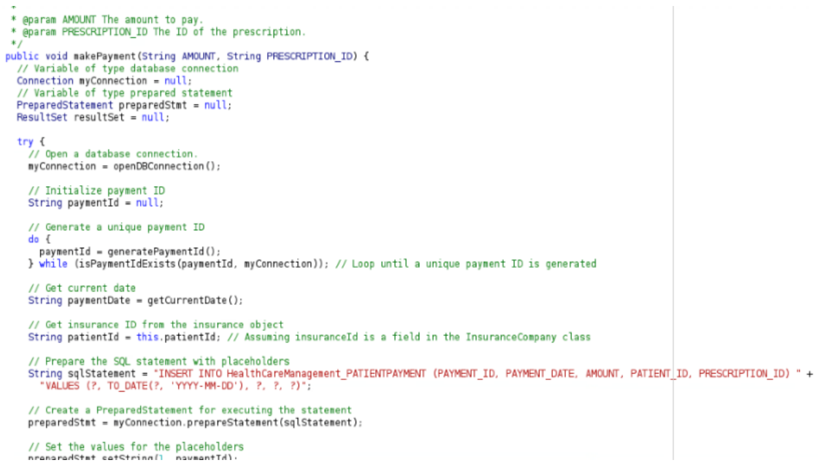
```sql
--THIS TRIGGER CHANGES THE PRESCRIPTION PRICE AFTER A PAYMENT HAS BEEN MADE BY THE PATIENT
CREATE or REPLACE TRIGGER ChangePrescriptionPriceAfterPatientPayment
    AFTER INSERT ON HealthCareManagement_PATIENTPAYMENT
    For Each Row
BEGIN
    UPDATE  HealthCareManagement_PRESCRIPTIONBALANCE
    SET     PATIENTBALANCE=PATIENTBALANCE-:NEW.AMOUNT
    WHERE   PRESCRIPTION_ID=:NEW.PRESCRIPTION_ID;
END;

--THIS TRIGGER CHANGES THE PRESCRIPTION PRICE AFTER A PAYMENT HAS BEEN MADE BY THE INSURANCE COMPANY
CREATE or REPLACE TRIGGER ChangePrescriptionPriceAfterInsurancePayment
    AFTER INSERT ON HealthCareManagement_INSURANCEPAYMENT
    For Each Row
BEGIN
    UPDATE  HealthCareManagement_PRESCRIPTIONBALANCE
    SET     INSURANCEBALANCE=INSURANCEBALANCE-:NEW.AMOUNT
    WHERE   PRESCRIPTION_ID=:NEW.PRESCRIPTION_ID;
END;
```

B.

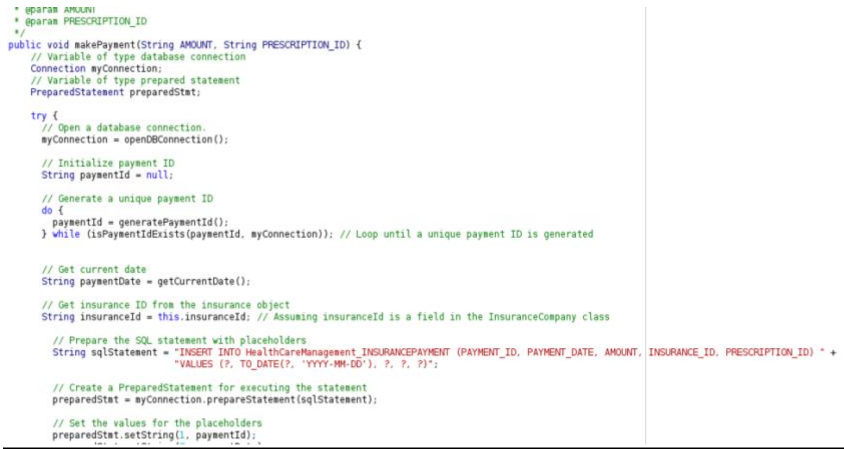
C. 

```
--TEST STATEMENTS:
SELECT * FROM HealthCareManagement_PRESCRIPTIONBALANCE;
INSERT INTO HealthCareManagement_PATIENTPAYMENT (PAYMENT_ID, PAYMENT_DATE, AMOUNT, PATIENT_ID, PRESCRIPTION_ID)
        VALUES ('PAY001', TO_DATE('2023-06-15', 'YYYY-MM-DD'), 15.00, 'PAT001', 'PRSC001');
INSERT INTO HealthCareManagement_INSURANCEPAYMENT (PAYMENT_ID, PAYMENT_DATE, AMOUNT, INSURANCE_ID, PRESCRIPTION_ID)
        VALUES ('PAY001', TO_DATE('2023-06-15', 'YYYY-MM-DD'), 2.00, 'INS001', 'PRSC001');
SELECT * FROM HealthCareManagement_PRESCRIPTIONBALANCE;

--FIRST SELECT:
--PRESCRIPTION_ID   PATIENT_ID      INSURANCE_ID   INSURANCEBAL PATIENTBAL
--PRSC001           PAT001          INS001          2.5         22.5
--PRSC002           PAT002          INS002          3.75        11.25
--PRSC003           PAT003          INS003          12          18
--PRSC004           PAT004          INS004          3.3         18.7
--PRSC005           PAT005          INS005          4.5         40.5
--PRSC006           PAT006          INS006          0           10

--AFTER INSERT:
--PRESCRIPTION_ID   PATIENT_ID      INSURANCE_ID   INSURANCEBAL PATIENTBAL
--PRSC001           PAT001          INS001          0.5         7.5
--PRSC002           PAT002          INS002          3.75        11.25
--PRSC003           PAT003          INS003          12          18
--PRSC004           PAT004          INS004          3.3         18.7
--PRSC005           PAT005          INS005          4.5         40.5
--PRSC006           PAT006          INS006          0           10
```

D. This trigger updates the remaining balances for insurance companies and patients when they make a payment on a prescription.

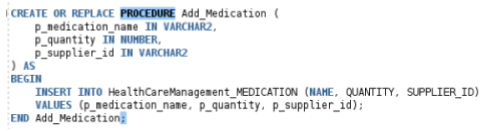


```
  *
  * @param AMOUNT The amount to pay.
  * @param PRESCRIPTION_ID The ID of the prescription.
  */
public void makePayment(String AMOUNT, String PRESCRIPTION_ID) {
  // Variable of type database connection
  Connection myConnection = null;
  // Variable of type prepared statement
  PreparedStatement preparedStmt = null;
  ResultSet resultSet = null;

  try {
    // Open a database connection.
    myConnection = openDBConnection();

    // Initialize payment ID
    String paymentId = null;

    // Generate a unique payment ID
    do {
      paymentId = generatePaymentId();
    } while (isPaymentIdExists(paymentId, myConnection)); // Loop until a unique payment ID is generated

    // Get current date
    String paymentDate = getCurrentDate();

    // Get insurance ID from the insurance object
    String patientId = this.patientId; // Assuming insuranceId is a field in the InsuranceCompany class

    // Prepare the SQL statement with placeholders
    String sqlStatement = "INSERT INTO HealthCareManagement_PATIENTPAYMENT (PAYMENT_ID, PAYMENT_DATE, AMOUNT, PATIENT_ID, PRESCRIPTION_ID) " +
      "VALUES (?, TO_DATE(?, 'YYYY-MM-DD'), ?, ?, ?)";

    // Create a PreparedStatement for executing the statement
    preparedStmt = myConnection.prepareStatement(sqlStatement);

    // Set the values for the placeholders
    preparedStmt.setString(1, paymentId);
```

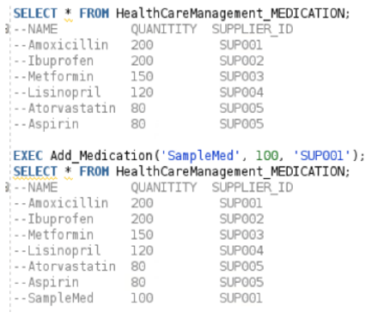


```
 * @param AMOUNT
 * @param PRESCRIPTION_ID
 */
public void makePayment(String AMOUNT, String PRESCRIPTION_ID) {
    // Variable of type database connection
    Connection myConnection;
    // Variable of type prepared statement
    PreparedStatement preparedStmt;

    try {
      // Open a database connection.
      myConnection = openDBConnection();

      // Initialize payment ID
      String paymentId = null;

      // Generate a unique payment ID
      do {
        paymentId = generatePaymentId();
      } while (isPaymentIdExists(paymentId, myConnection)); // Loop until a unique payment ID is generated


      // Get current date
      String paymentDate = getCurrentDate();

      // Get insurance ID from the insurance object
      String insuranceId = this.insuranceId; // Assuming insuranceId is a field in the InsuranceCompany class

        // Prepare the SQL statement with placeholders
        String sqlStatement = "INSERT INTO HealthCareManagement_INSURANCEPAYMENT (PAYMENT_ID, PAYMENT_DATE, AMOUNT, INSURANCE_ID, PRESCRIPTION_ID) " +
                              "VALUES (?, TO_DATE(?, 'YYYY-MM-DD'), ?, ?, ?)";

        // Create a PreparedStatement for executing the statement
        preparedStmt = myConnection.prepareStatement(sqlStatement);

        // Set the values for the placeholders
        preparedStmt.setString(1, paymentId);
```

A. Procedure Add_Medication

B. 

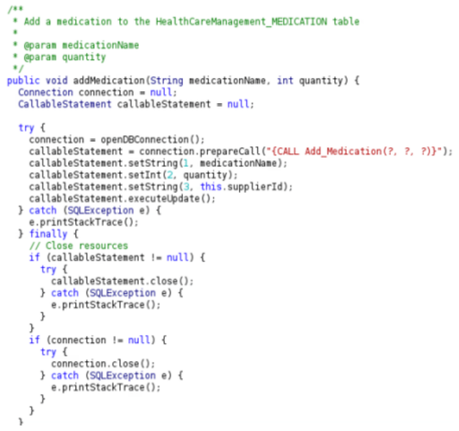
```
CREATE OR REPLACE PROCEDURE Add_Medication (
    p_medication_name IN VARCHAR2,
    p_quantity IN NUMBER,
    p_supplier_id IN VARCHAR2
) AS
BEGIN
    INSERT INTO HealthCareManagement_MEDICATION (NAME, QUANTITY, SUPPLIER_ID)
    VALUES (p_medication_name, p_quantity, p_supplier_id);
END Add_Medication;
```



```
SELECT * FROM HealthCareManagement_MEDICATION;
--NAME          QUANITITY  SUPPLIER_ID
--Amoxicillin   200        SUP001
--Ibuprofen     200        SUP002
--Metformin     150        SUP003
--Lisinopril    120        SUP004
--Atorvastatin  80         SUP005
--Aspirin       80         SUP005

EXEC Add_Medication('SampleMed', 100, 'SUP001');
SELECT * FROM HealthCareManagement_MEDICATION;
--NAME          QUANITITY  SUPPLIER_ID
--Amoxicillin   200        SUP001
--Ibuprofen     200        SUP002
--Metformin     150        SUP003
--Lisinopril    120        SUP004
--Atorvastatin  80         SUP005
--Aspirin       80         SUP005
--SampleMed     100        SUP001
```

C.

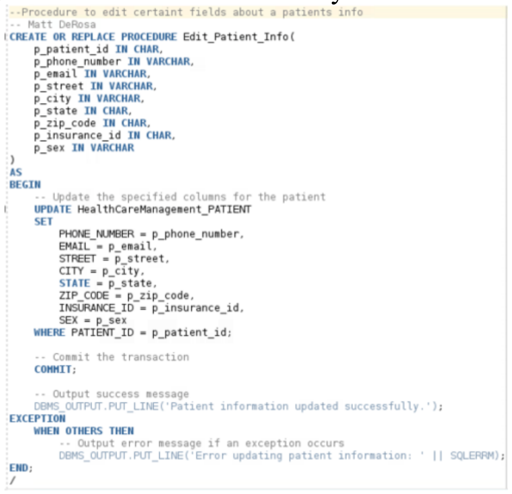
D. Allows suppliers to add and remove medications.



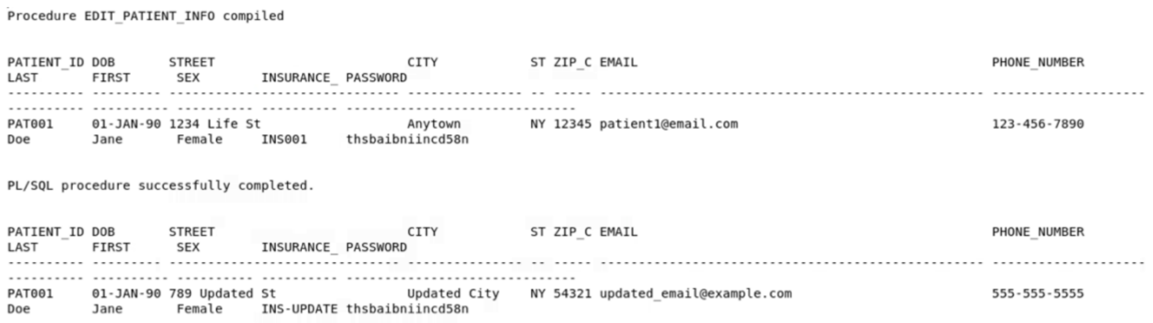
```java
/**
 * Add a medication to the HealthCareManagement_MEDICATION table
 *
 * @param medicationName
 * @param quantity
 */
public void addMedication(String medicationName, int quantity) {
  Connection connection = null;
  CallableStatement callableStatement = null;

  try {
    connection = openDBConnection();
    callableStatement = connection.prepareCall("{CALL Add_Medication(?, ?, ?)}");
    callableStatement.setString(1, medicationName);
    callableStatement.setInt(2, quantity);
    callableStatement.setString(3, this.supplierId);
    callableStatement.executeUpdate();
  } catch (SQLException e) {
    e.printStackTrace();
  } finally {
    // Close resources
    if (callableStatement != null) {
      try {
        callableStatement.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
    if (connection != null) {
      try {
        connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
  }
```

Matt's Stored Routines

a. Procedure for editing a user. Users include Patient, Doctor, Pharmacy, Pharmacy Employee, and Supplier. All of the Users have this procedure but slightly different based on the fields they have and the fields that they can edit

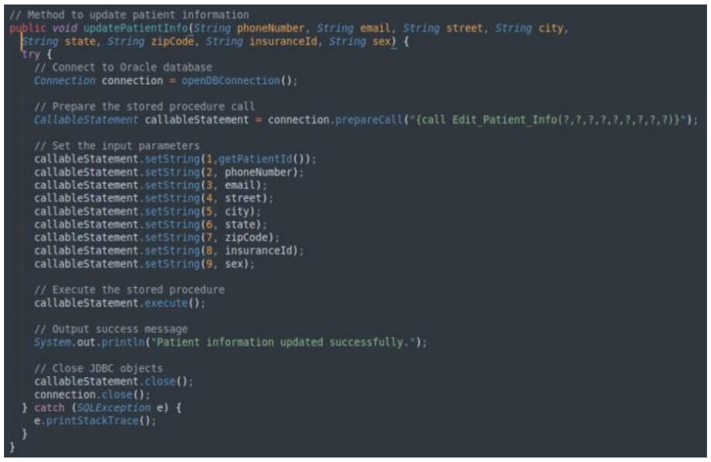


```sql
--Procedure to edit certaint fields about a patients info
-- Matt DeRosa
CREATE OR REPLACE PROCEDURE Edit_Patient_Info(
    p_patient_id IN CHAR,
    p_phone_number IN VARCHAR,
    p_email IN VARCHAR,
    p_street IN VARCHAR,
    p_city IN VARCHAR,
    p_state IN CHAR,
    p_zip_code IN CHAR,
    p_insurance_id IN CHAR,
    p_sex IN VARCHAR
)
AS
BEGIN
    -- Update the specified columns for the patient
    UPDATE HealthCareManagement_PATIENT
    SET
        PHONE_NUMBER = p_phone_number,
        EMAIL = p_email,
        STREET = p_street,
        CITY = p_city,
        STATE = p_state,
        ZIP_CODE = p_zip_code,
        INSURANCE_ID = p_insurance_id,
        SEX = p_sex
    WHERE PATIENT_ID = p_patient_id;

    -- Commit the transaction
    COMMIT;

    -- Output success message
    DBMS_OUTPUT.PUT_LINE('Patient information updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        -- Output error message if an exception occurs
        DBMS_OUTPUT.PUT_LINE('Error updating patient information: ' || SQLERRM);
END;
/
```



```
Procedure EDIT_PATIENT_INFO compiled

PATIENT_ID DOB       STREET                    CITY          ST ZIP_C EMAIL                                             PHONE_NUMBER
LAST       FIRST      SEX        INSURANCE_ PASSWORD
---------- --------- ------------------------- ------------- -- ----- ------------------------------------------------- --------------------
--------- --------- --------- --------- ---------------------------
PAT001     01-JAN-90 1234 Life St              Anytown       NY 12345 patient1@email.com                                123-456-7890
Doe        Jane       Female     INS001     thsbaibniincd58n

PL/SQL procedure successfully completed.

PATIENT_ID DOB       STREET                    CITY          ST ZIP_C EMAIL                                             PHONE_NUMBER
LAST       FIRST      SEX        INSURANCE_ PASSWORD
---------- --------- ------------------------- ------------- -- ----- ------------------------------------------------- --------------------
--------- --------- --------- --------- ---------------------------
PAT001     01-JAN-90 789 Updated St            Updated City  NY 54321 updated_email@example.com                         555-555-5555
Doe        Jane       Female     INS-UPDATE thsbaibniincd58n
```

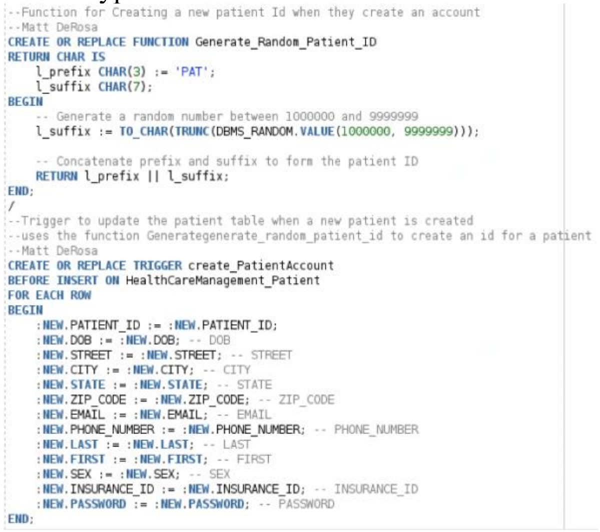
a. Java JDBC method to call the procedures, again the JDBC files a little different based on the fields they are setting.



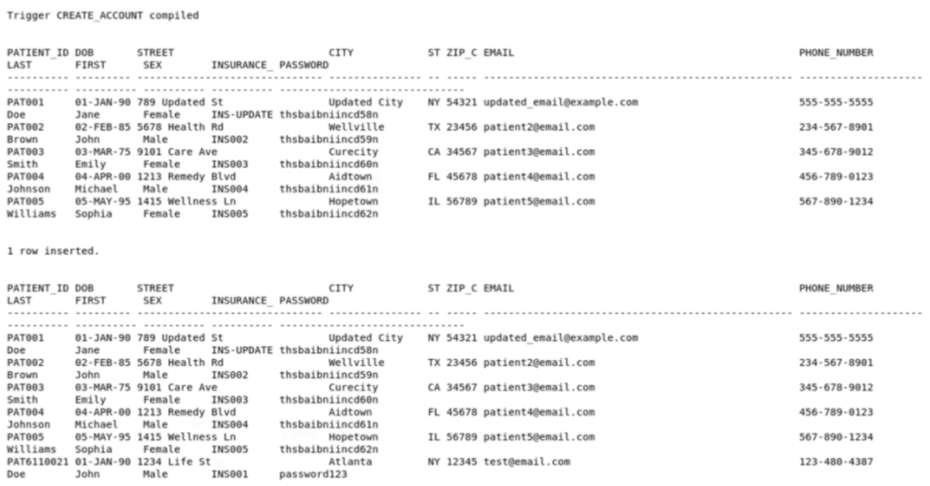
```java
// Method to update patient information
public void updatePatientInfo(String phoneNumber, String email, String street, String city,
  String state, String zipCode, String insuranceId, String sex) {
  try {
    // Connect to Oracle database
    Connection connection = openDBConnection();

    // Prepare the stored procedure call
    CallableStatement callableStatement = connection.prepareCall("{call Edit_Patient_Info(?,?,?,?,?,?,?,?,?)}");

    // Set the input parameters
    callableStatement.setString(1,getPatientId());
    callableStatement.setString(2, phoneNumber);
    callableStatement.setString(3, email);
    callableStatement.setString(4, street);
    callableStatement.setString(5, city);
    callableStatement.setString(6, state);
    callableStatement.setString(7, zipCode);
    callableStatement.setString(8, insuranceId);
    callableStatement.setString(9, sex);

    // Execute the stored procedure
    callableStatement.execute();

    // Output success message
    System.out.println("Patient information updated successfully.");

    // Close JDBC objects
    callableStatement.close();
    connection.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

b. Function creates a randomly generated Id number for Users. Users include Patient, Doctor, Pharmacy, Pharmacy Employee, and Supplier. All of the Users have this function but slightly different based on their type and characters remaining after identifier. For example, Patient has PAT Char(3) with Char(7) remaining for digits and Pharmacy has PHRM Char(4) with Char(6) remaining for digits.

b. Trigger waits for a new user to that users respective table. Again, this trigger is implemented for all user types and different based on fields.

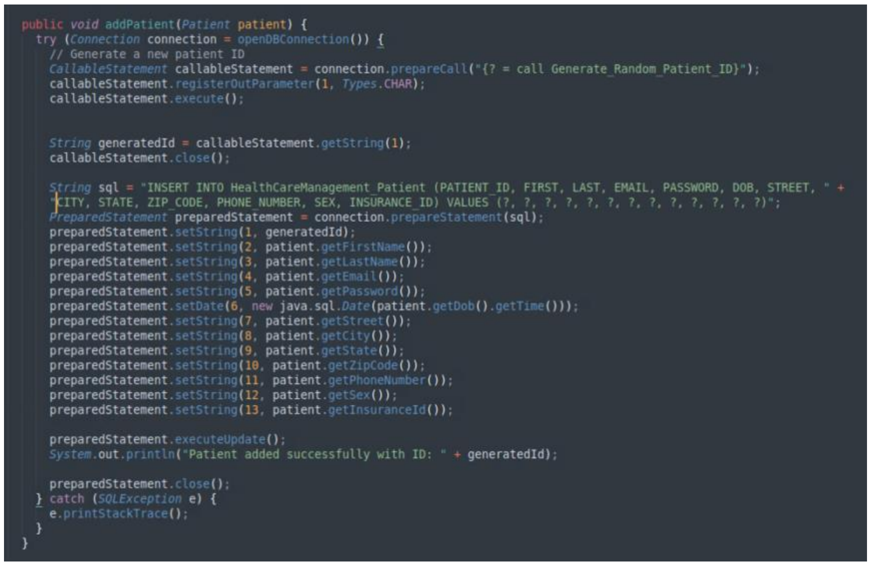


```sql
--Function for Creating a new patient Id when they create an account
--Matt DeRosa
CREATE OR REPLACE FUNCTION Generate_Random_Patient_ID
RETURN CHAR IS
    l_prefix CHAR(3) := 'PAT';
    l_suffix CHAR(7);
BEGIN
    -- Generate a random number between 1000000 and 9999999
    l_suffix := TO_CHAR(TRUNC(DBMS_RANDOM.VALUE(1000000, 9999999)));

    -- Concatenate prefix and suffix to form the patient ID
    RETURN l_prefix || l_suffix;
END;
/
--Trigger to update the patient table when a new patient is created
--uses the function Generategenerate_random_patient_id to create an id for a patient
--Matt DeRosa
CREATE OR REPLACE TRIGGER create_PatientAccount
BEFORE INSERT ON HealthCareManagement_Patient
FOR EACH ROW
BEGIN
    :NEW.PATIENT_ID := :NEW.PATIENT_ID;
    :NEW.DOB := :NEW.DOB; -- DOB
    :NEW.STREET := :NEW.STREET; -- STREET
    :NEW.CITY := :NEW.CITY; -- CITY
    :NEW.STATE := :NEW.STATE; -- STATE
    :NEW.ZIP_CODE := :NEW.ZIP_CODE; -- ZIP_CODE
    :NEW.EMAIL := :NEW.EMAIL; -- EMAIL
    :NEW.PHONE_NUMBER := :NEW.PHONE_NUMBER; -- PHONE_NUMBER
    :NEW.LAST := :NEW.LAST; -- LAST
    :NEW.FIRST := :NEW.FIRST; -- FIRST
    :NEW.SEX := :NEW.SEX; -- SEX
    :NEW.INSURANCE_ID := :NEW.INSURANCE_ID; -- INSURANCE_ID
    :NEW.PASSWORD := :NEW.PASSWORD; -- PASSWORD
END;
```



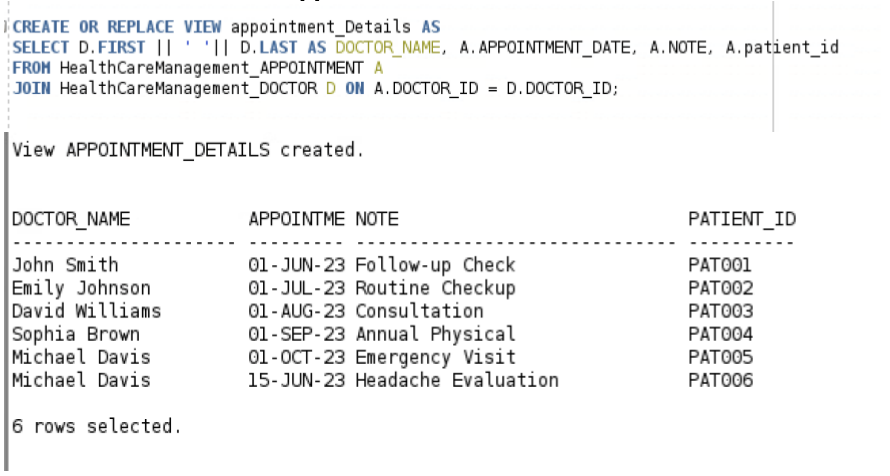
```
Trigger CREATE_ACCOUNT compiled

PATIENT_ID DOB       STREET                        CITY           ST ZIP_C EMAIL                                              PHONE_NUMBER
LAST       FIRST      SEX       INSURANCE_ PASSWORD
---------- --------- ------------------------------ -------------- -- ----- -------------------------------------------------- --------------------
---------- ---------- ---------- ---------- ------------------------------
PAT001     01-JAN-90 789 Updated St                Updated City   NY 54321 updated_email@example.com                          555-555-5555
Doe        Jane       Female    INS-UPDATE thsbaibniincd58n
PAT002     02-FEB-85 5678 Health Rd                Wellville      TX 23456 patient2@email.com                                 234-567-8901
Brown      John       Male      INS002     thsbaibniincd59n
PAT003     03-MAR-75 9101 Care Ave                 Curecity       CA 34567 patient3@email.com                                 345-678-9012
Smith      Emily      Female    INS003     thsbaibniincd60n
PAT004     04-APR-00 1213 Remedy Blvd              Aidtown        FL 45678 patient4@email.com                                 456-789-0123
Johnson    Michael    Male      INS004     thsbaibniincd61n
PAT005     05-MAY-95 1415 Wellness Ln              Hopetown       IL 56789 patient5@email.com                                 567-890-1234
Williams   Sophia     Female    INS005     thsbaibniincd62n


1 row inserted.


PATIENT_ID DOB       STREET                        CITY           ST ZIP_C EMAIL                                              PHONE_NUMBER
LAST       FIRST      SEX       INSURANCE_ PASSWORD
---------- --------- ------------------------------ -------------- -- ----- -------------------------------------------------- --------------------
---------- ---------- ---------- ---------- ------------------------------
PAT001     01-JAN-90 789 Updated St                Updated City   NY 54321 updated_email@example.com                          555-555-5555
Doe        Jane       Female    INS-UPDATE thsbaibniincd58n
PAT002     02-FEB-85 5678 Health Rd                Wellville      TX 23456 patient2@email.com                                 234-567-8901
Brown      John       Male      INS002     thsbaibniincd59n
PAT003     03-MAR-75 9101 Care Ave                 Curecity       CA 34567 patient3@email.com                                 345-678-9012
Smith      Emily      Female    INS003     thsbaibniincd60n
PAT004     04-APR-00 1213 Remedy Blvd              Aidtown        FL 45678 patient4@email.com                                 456-789-0123
Johnson    Michael    Male      INS004     thsbaibniincd61n
PAT005     05-MAY-95 1415 Wellness Ln              Hopetown       IL 56789 patient5@email.com                                 567-890-1234
Williams   Sophia     Female    INS005     thsbaibniincd62n
PAT6110021 01-JAN-90 1234 Life St                  Atlanta        NY 12345 test@email.com                                     123-480-4387
Doe        John       Male      INS001     password123
```

b. Java JDBC method to call the function to generate a random Id for a user and then pass the generated Id to the creation of a new user. The same but different fields for all users.



```java
public void addPatient(Patient patient) {
  try (Connection connection = openDBConnection()) {
    // Generate a new patient ID
    CallableStatement callableStatement = connection.prepareCall("{? = call Generate_Random_Patient_ID}");
    callableStatement.registerOutParameter(1, Types.CHAR);
    callableStatement.execute();


    String generatedId = callableStatement.getString(1);
    callableStatement.close();

    String sql = "INSERT INTO HealthCareManagement_Patient (PATIENT_ID, FIRST, LAST, EMAIL, PASSWORD, DOB, STREET, " +
    "CITY, STATE, ZIP_CODE, PHONE_NUMBER, SEX, INSURANCE_ID) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setString(1, generatedId);
    preparedStatement.setString(2, patient.getFirstName());
    preparedStatement.setString(3, patient.getLastName());
    preparedStatement.setString(4, patient.getEmail());
    preparedStatement.setString(5, patient.getPassword());
    preparedStatement.setDate(6, new java.sql.Date(patient.getDob().getTime()));
    preparedStatement.setString(7, patient.getStreet());
    preparedStatement.setString(8, patient.getCity());
    preparedStatement.setString(9, patient.getState());
    preparedStatement.setString(10, patient.getZipCode());
    preparedStatement.setString(11, patient.getPhoneNumber());
    preparedStatement.setString(12, patient.getSex());
    preparedStatement.setString(13, patient.getInsuranceId());

    preparedStatement.executeUpdate();
    System.out.println("Patient added successfully with ID: " + generatedId);

    preparedStatement.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

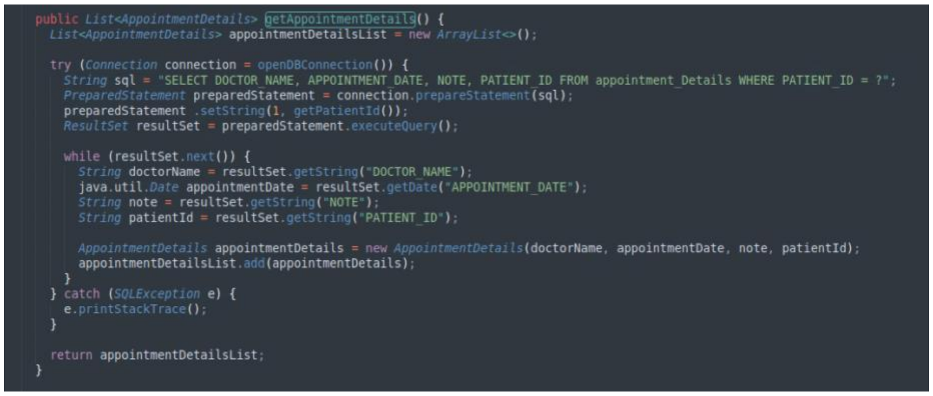
c. View to create a table for patients to be able to see all of their past appointments and details correlated to the appointment.



```sql
CREATE OR REPLACE VIEW appointment_Details AS
SELECT D.FIRST || ' '|| D.LAST AS DOCTOR_NAME, A.APPOINTMENT_DATE, A.NOTE, A.patient_id
FROM HealthCareManagement_APPOINTMENT A
JOIN HealthCareManagement_DOCTOR D ON A.DOCTOR_ID = D.DOCTOR_ID;
```

```
View APPOINTMENT_DETAILS created.


DOCTOR_NAME           APPOINTME NOTE                           PATIENT_ID
--------------------- --------- ------------------------------ ----------
John Smith            01-JUN-23 Follow-up Check                PAT001
Emily Johnson         01-JUL-23 Routine Checkup                PAT002
David Williams        01-AUG-23 Consultation                   PAT003
Sophia Brown          01-SEP-23 Annual Physical                PAT004
Michael Davis         01-OCT-23 Emergency Visit                PAT005
Michael Davis         15-JUN-23 Headache Evaluation            PAT006

6 rows selected.
```

c. Java JDBC method for getting a patient’s appointments and the appointment details and getting them in an array list.

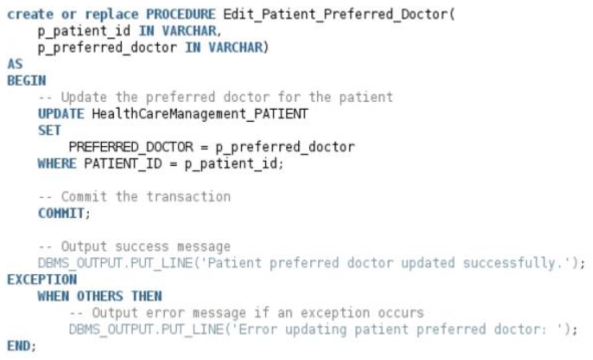


```java
public List<AppointmentDetails> getAppointmentDetails() {
  List<AppointmentDetails> appointmentDetailsList = new ArrayList<>();

  try (Connection connection = openDBConnection()) {
    String sql = "SELECT DOCTOR_NAME, APPOINTMENT_DATE, NOTE, PATIENT_ID FROM appointment_Details WHERE PATIENT_ID = ?";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement .setString(1, getPatientId());
    ResultSet resultSet = preparedStatement.executeQuery();

    while (resultSet.next()) {
      String doctorName = resultSet.getString("DOCTOR_NAME");
      java.util.Date appointmentDate = resultSet.getDate("APPOINTMENT_DATE");
      String note = resultSet.getString("NOTE");
      String patientId = resultSet.getString("PATIENT_ID");

      AppointmentDetails appointmentDetails = new AppointmentDetails(doctorName, appointmentDate, note, patientId);
      appointmentDetailsList.add(appointmentDetails);
    }
  } catch (SQLException e) {
    e.printStackTrace();
  }

  return appointmentDetailsList;
}
```

Max’s Stored Routines

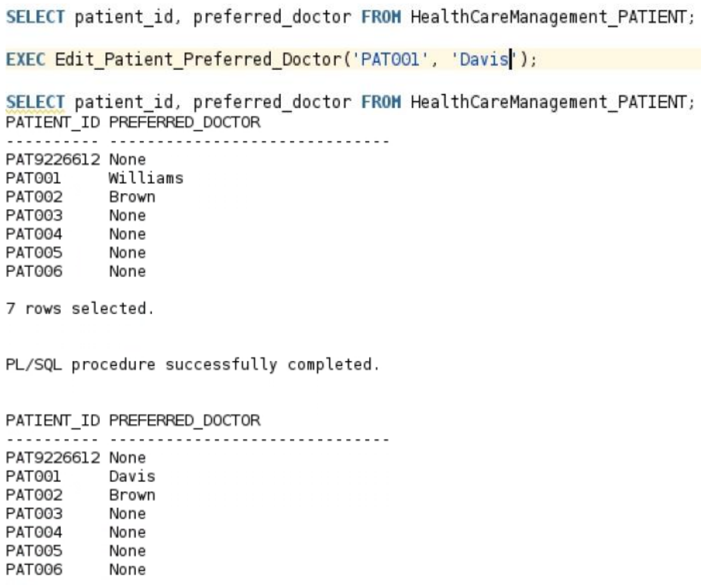
A. PROCEDURE Edit_Patient_Preferred_Doctor allows for users to add/update their preferred doctor attribute after viewing list of doctors.

B.



```sql
create or replace PROCEDURE Edit_Patient_Preferred_Doctor(
    p_patient_id IN VARCHAR,
    p_preferred_doctor IN VARCHAR)
AS
BEGIN
    -- Update the preferred doctor for the patient
    UPDATE HealthCareManagement_PATIENT
    SET
        PREFERRED_DOCTOR = p_preferred_doctor
    WHERE PATIENT_ID = p_patient_id;

    -- Commit the transaction
    COMMIT;

    -- Output success message
    DBMS_OUTPUT.PUT_LINE('Patient preferred doctor updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        -- Output error message if an exception occurs
        DBMS_OUTPUT.PUT_LINE('Error updating patient preferred doctor: ');
END;
```

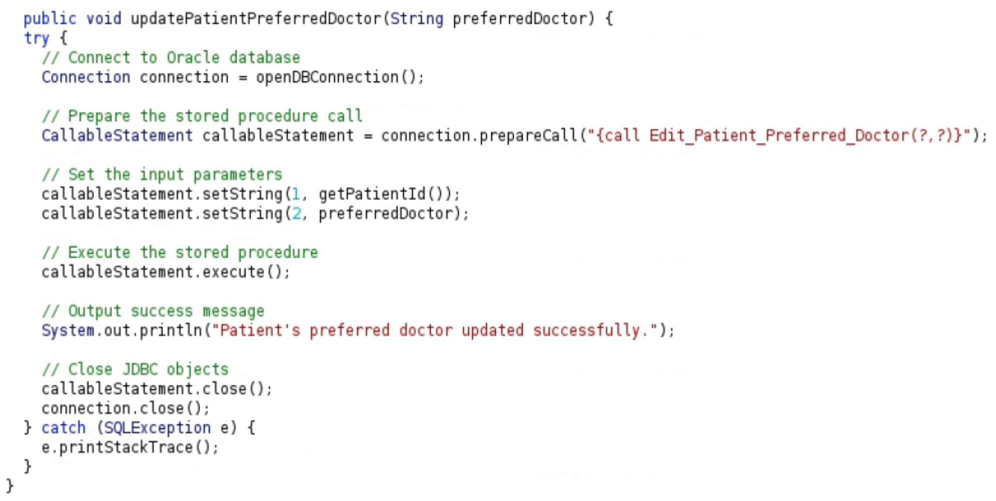
C.



```sql
SELECT patient_id, preferred_doctor FROM HealthCareManagement_PATIENT;

EXEC Edit_Patient_Preferred_Doctor('PAT001', 'Davis');

SELECT patient_id, preferred_doctor FROM HealthCareManagement_PATIENT;
```

```
PATIENT_ID PREFERRED_DOCTOR
---------- ------------------------------
PAT9226612 None
PAT001     Williams
PAT002     Brown
PAT003     None
PAT004     None
PAT005     None
PAT006     None

7 rows selected.


PL/SQL procedure successfully completed.


PATIENT_ID PREFERRED_DOCTOR
---------- ------------------------------
PAT9226612 None
PAT001     Davis
PAT002     Brown
PAT003     None
PAT004     None
PAT005     None
PAT006     None
```

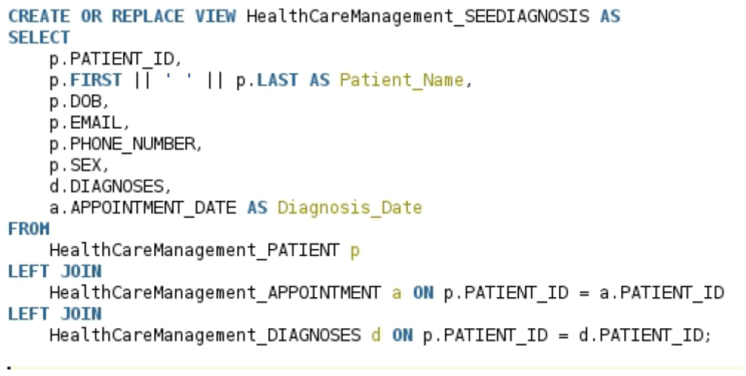
D.



```
public void updatePatientPreferredDoctor(String preferredDoctor) {
  try {
    // Connect to Oracle database
    Connection connection = openDBConnection();

    // Prepare the stored procedure call
    CallableStatement callableStatement = connection.prepareCall("{call Edit_Patient_Preferred_Doctor(?,?)}");

    // Set the input parameters
    callableStatement.setString(1, getPatientId());
    callableStatement.setString(2, preferredDoctor);

    // Execute the stored procedure
    callableStatement.execute();

    // Output success message
    System.out.println("Patient's preferred doctor updated successfully.");

    // Close JDBC objects
    callableStatement.close();
    connection.close();
  } catch (SQLException e) {
    e.printStackTrace();
  }
}
```

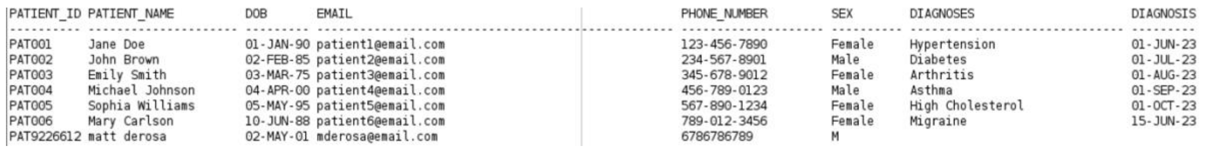
A. VIEW HealthCareManagement_SEEDIAGNOSIS provides an overview of patients, their general info, and a list of their diagnoses from previous appointments. The dates of the diagnoses are also listed by joining patient, appointment, and diagnosis data.

B.



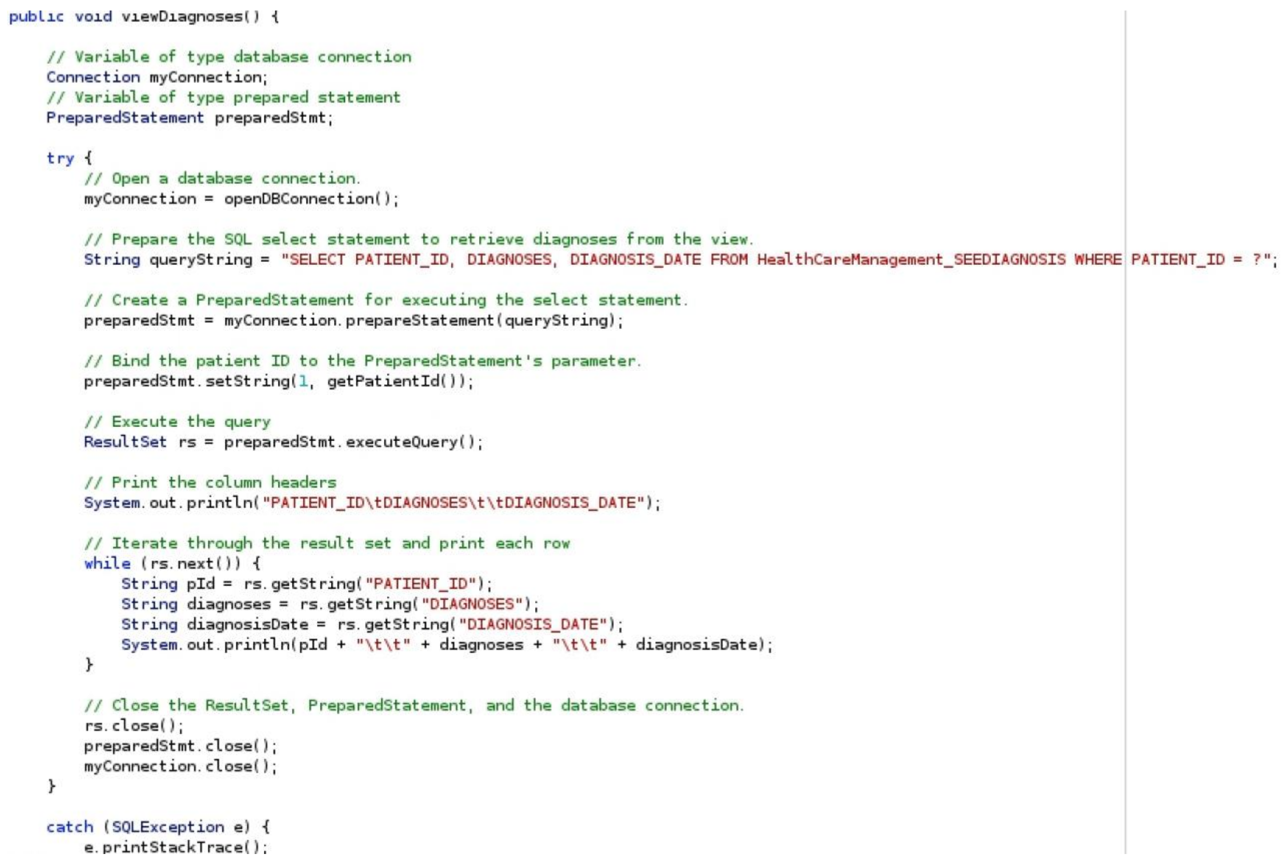
```
CREATE OR REPLACE VIEW HealthCareManagement_SEEDIAGNOSIS AS
SELECT
    p.PATIENT_ID,
    p.FIRST || ' ' || p.LAST AS Patient_Name,
    p.DOB,
    p.EMAIL,
    p.PHONE_NUMBER,
    p.SEX,
    d.DIAGNOSES,
    a.APPOINTMENT_DATE AS Diagnosis_Date
FROM
    HealthCareManagement_PATIENT p
LEFT JOIN
    HealthCareManagement_APPOINTMENT a ON p.PATIENT_ID = a.PATIENT_ID
LEFT JOIN
    HealthCareManagement_DIAGNOSES d ON p.PATIENT_ID = d.PATIENT_ID;
```

C.



```
PATIENT_ID PATIENT_NAME         DOB       EMAIL                                                       PHONE_NUMBER         SEX       DIAGNOSES                      DIAGNOSIS
---------- -------------------- --------- ------------------------------------------------------------ -------------------- --------- ------------------------------ ---------
PAT001     Jane Doe             01-JAN-90 patient1@email.com                                          123-456-7890         Female    Hypertension                   01-JUN-23
PAT002     John Brown           02-FEB-85 patient2@email.com                                          234-567-8901         Male      Diabetes                       01-JUL-23
PAT003     Emily Smith          03-MAR-75 patient3@email.com                                          345-678-9012         Female    Arthritis                      01-AUG-23
PAT004     Michael Johnson      04-APR-00 patient4@email.com                                          456-789-0123         Male      Asthma                         01-SEP-23
PAT005     Sophia Williams      05-MAY-95 patient5@email.com                                          567-890-1234         Female    High Cholesterol               01-OCT-23
PAT006     Mary Carlson         10-JUN-88 patient6@email.com                                          789-012-3456         Female    Migraine                       15-JUN-23
PAT9226612 matt derosa          02-MAY-01 mderosa@email.com                                           6786786789           M
```

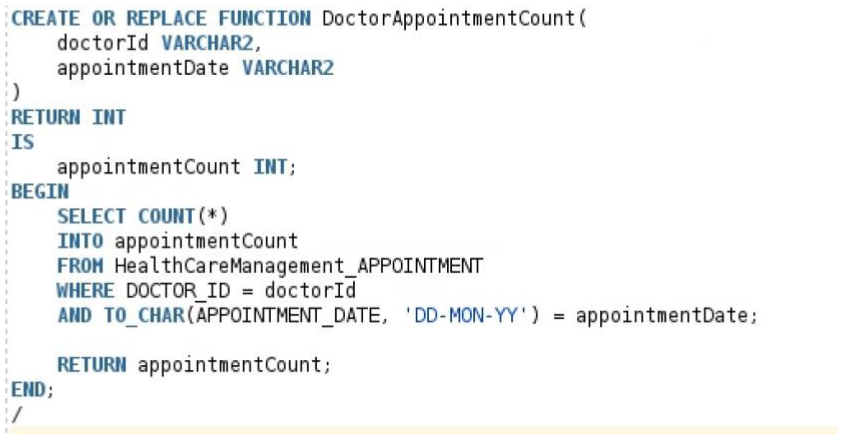
D.



```
public void viewDiagnoses() {

    // Variable of type database connection
    Connection myConnection;
    // Variable of type prepared statement
    PreparedStatement preparedStmt;

    try {
        // Open a database connection.
        myConnection = openDBConnection();

        // Prepare the SQL select statement to retrieve diagnoses from the view.
        String queryString = "SELECT PATIENT_ID, DIAGNOSES, DIAGNOSIS_DATE FROM HealthCareManagement_SEEDIAGNOSIS WHERE PATIENT_ID = ?";

        // Create a PreparedStatement for executing the select statement.
        preparedStmt = myConnection.prepareStatement(queryString);

        // Bind the patient ID to the PreparedStatement's parameter.
        preparedStmt.setString(1, getPatientId());

        // Execute the query
        ResultSet rs = preparedStmt.executeQuery();

        // Print the column headers
        System.out.println("PATIENT_ID\tDIAGNOSES\t\tDIAGNOSIS_DATE");

        // Iterate through the result set and print each row
        while (rs.next()) {
            String pId = rs.getString("PATIENT_ID");
            String diagnoses = rs.getString("DIAGNOSES");
            String diagnosisDate = rs.getString("DIAGNOSIS_DATE");
            System.out.println(pId + "\t\t" + diagnoses + "\t\t" + diagnosisDate);
        }

        // Close the ResultSet, PreparedStatement, and the database connection.
        rs.close();
        preparedStmt.close();
        myConnection.close();
    }

    catch (SQLException e) {
        e.printStackTrace();
```

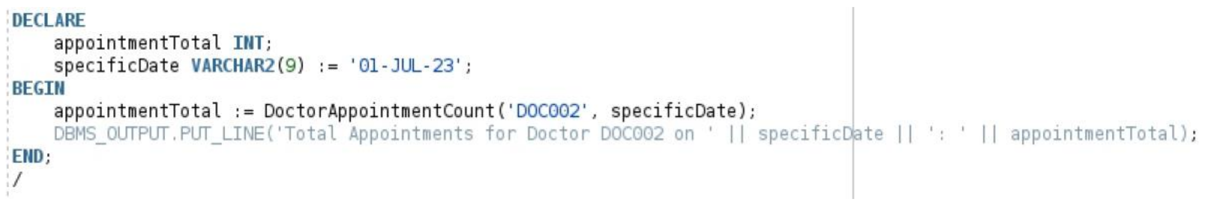
A. Function: DoctorAppointmentCount counts the number of appointments for a specific doctor on a given date. This allows doctors to see their schedule/capacity for a certain day.

B.

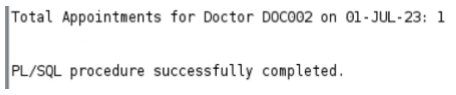


```
CREATE OR REPLACE FUNCTION DoctorAppointmentCount(
    doctorId VARCHAR2,
    appointmentDate VARCHAR2
)
RETURN INT
IS
    appointmentCount INT;
BEGIN
    SELECT COUNT(*)
    INTO appointmentCount
    FROM HealthCareManagement_APPOINTMENT
    WHERE DOCTOR_ID = doctorId
    AND TO_CHAR(APPOINTMENT_DATE, 'DD-MON-YY') = appointmentDate;

    RETURN appointmentCount;
END;
/
```

C.

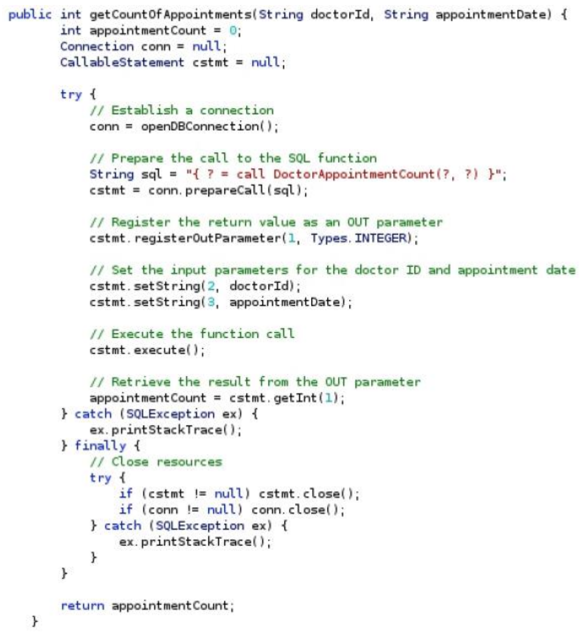


```
DECLARE
    appointmentTotal INT;
    specificDate VARCHAR2(9) := '01-JUL-23';
BEGIN
    appointmentTotal := DoctorAppointmentCount('DOC002', specificDate);
    DBMS_OUTPUT.PUT_LINE('Total Appointments for Doctor DOC002 on ' || specificDate || ': ' || appointmentTotal);
END;
/
```



```
Total Appointments for Doctor DOC002 on 01-JUL-23: 1


PL/SQL procedure successfully completed.
```

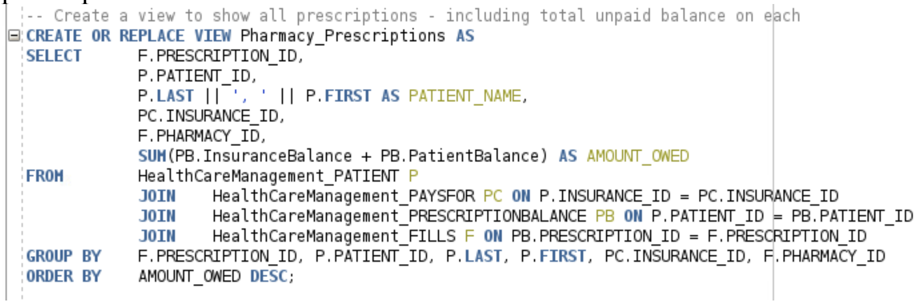
D.



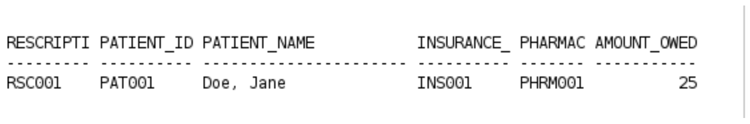
```
public int getCountOfAppointments(String doctorId, String appointmentDate) {
        int appointmentCount = 0;
        Connection conn = null;
        CallableStatement cstmt = null;

        try {
            // Establish a connection
            conn = openDBConnection();

            // Prepare the call to the SQL function
            String sql = "{ ? = call DoctorAppointmentCount(?, ?) }";
            cstmt = conn.prepareCall(sql);

            // Register the return value as an OUT parameter
            cstmt.registerOutParameter(1, Types.INTEGER);

            // Set the input parameters for the doctor ID and appointment date
            cstmt.setString(2, doctorId);
            cstmt.setString(3, appointmentDate);

            // Execute the function call
            cstmt.execute();

            // Retrieve the result from the OUT parameter
            appointmentCount = cstmt.getInt(1);
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            // Close resources
            try {
                if (cstmt != null) cstmt.close();
                if (conn != null) conn.close();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }

        return appointmentCount;
    }
```

Evan's Stored Routines

A. VIEW Pharmacy_Prescriptions allows pharmacy employees to view information about a patients prescriptions. Works.

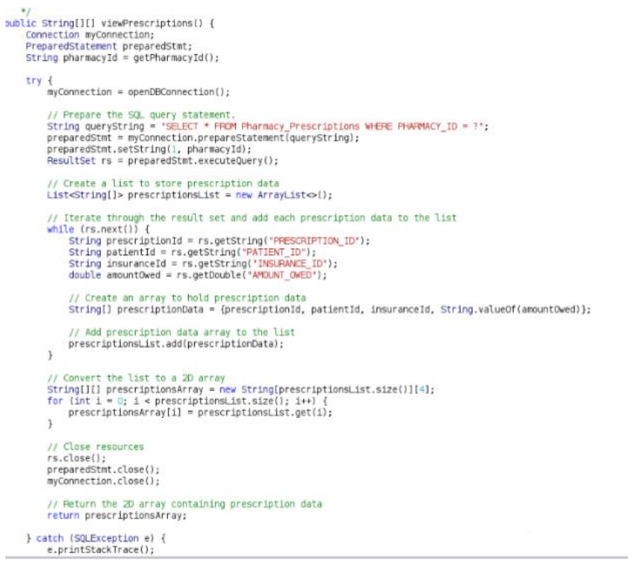


```
-- Create a view to show all prescriptions - including total unpaid balance on each
CREATE OR REPLACE VIEW Pharmacy_Prescriptions AS
SELECT      F.PRESCRIPTION_ID,
            P.PATIENT_ID,
            P.LAST || ', ' || P.FIRST AS PATIENT_NAME,
            PC.INSURANCE_ID,
            F.PHARMACY_ID,
            SUM(PB.InsuranceBalance + PB.PatientBalance) AS AMOUNT_OWED
FROM        HealthCareManagement_PATIENT P
            JOIN    HealthCareManagement_PAYSFOR PC ON P.INSURANCE_ID = PC.INSURANCE_ID
            JOIN    HealthCareManagement_PRESCRIPTIONBALANCE PB ON P.PATIENT_ID = PB.PATIENT_ID
            JOIN    HealthCareManagement_FILLS F ON PB.PRESCRIPTION_ID = F.PRESCRIPTION_ID
GROUP BY    F.PRESCRIPTION_ID, P.PATIENT_ID, P.LAST, P.FIRST, PC.INSURANCE_ID, F.PHARMACY_ID
ORDER BY    AMOUNT_OWED DESC;
```



```
RESCRIPTI PATIENT_ID PATIENT_NAME           INSURANCE_ PHARMAC AMOUNT_OWED
--------- ---------- ---------------------- ---------- ------- -----------
RSC001    PAT001     Doe, Jane              INS001     PHRM001          25
```

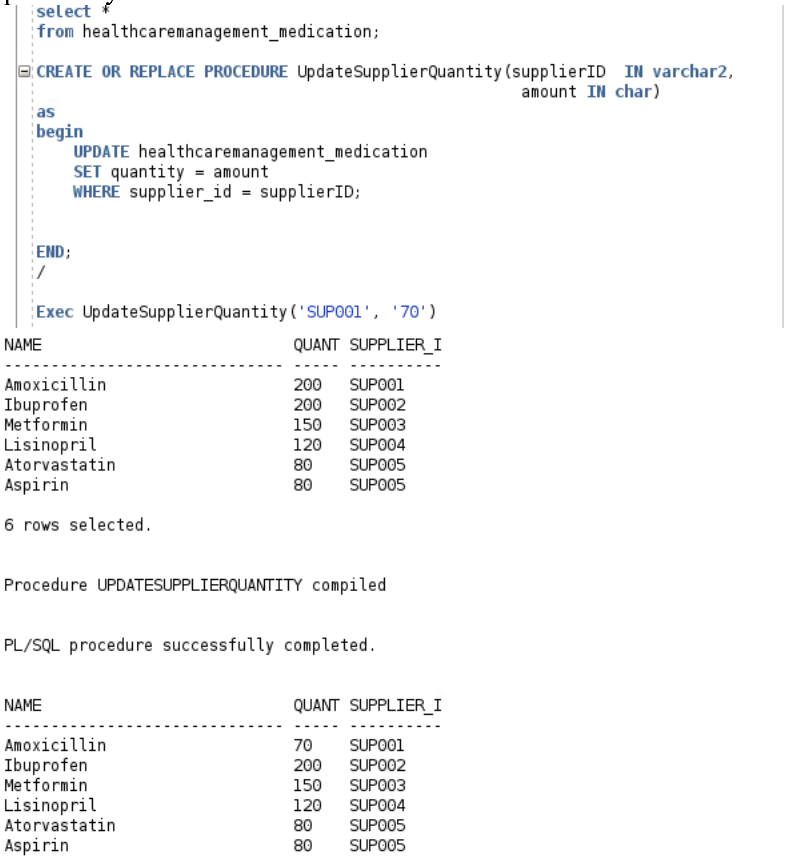
B.



```
    */
public String[][] viewPrescriptions() {
    Connection myConnection;
    PreparedStatement preparedStmt;
    String pharmacyId = getPharmacyId();

    try {
        myConnection = openDBConnection();

        // Prepare the SQL query statement.
        String queryString = "SELECT * FROM Pharmacy_Prescriptions WHERE PHARMACY_ID = ?";
        preparedStmt = myConnection.prepareStatement(queryString);
        preparedStmt.setString(1, pharmacyId);
        ResultSet rs = preparedStmt.executeQuery();

        // Create a list to store prescription data
        List<String[]> prescriptionsList = new ArrayList<>();

        // Iterate through the result set and add each prescription data to the list
        while (rs.next()) {
            String prescriptionId = rs.getString("PRESCRIPTION_ID");
            String patientId = rs.getString("PATIENT_ID");
            String insuranceId = rs.getString("INSURANCE_ID");
            double amountOwed = rs.getDouble("AMOUNT_OWED");

            // Create an array to hold prescription data
            String[] prescriptionData = {prescriptionId, patientId, insuranceId, String.valueOf(amountOwed)};

            // Add prescription data array to the list
            prescriptionsList.add(prescriptionData);
        }

        // Convert the list to a 2D array
        String[][] prescriptionsArray = new String[prescriptionsList.size()][4];
        for (int i = 0; i < prescriptionsList.size(); i++) {
            prescriptionsArray[i] = prescriptionsList.get(i);
        }

        // Close resources
        rs.close();
        preparedStmt.close();
        myConnection.close();

        // Return the 2D array containing prescription data
        return prescriptionsArray;

    } catch (SQLException e) {
        e.printStackTrace();
```

C.

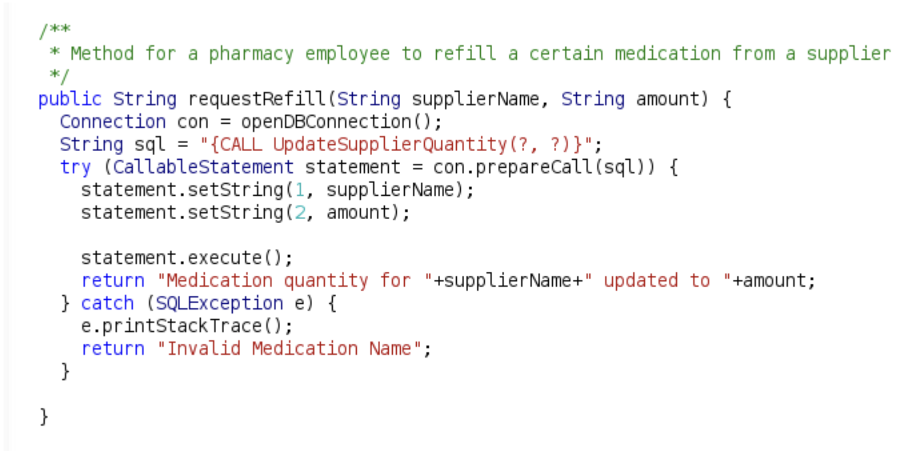
A. Procedure UpdateSupplierQuantity for updated the quantity of medicine available in the pharmacy. Works.



```
select *
from healthcaremanagement_medication;

CREATE OR REPLACE PROCEDURE UpdateSupplierQuantity(supplierID  IN varchar2,
                                                   amount IN char)
as
begin
    UPDATE healthcaremanagement_medication
    SET quantity = amount
    WHERE supplier_id = supplierID;


END;
/

Exec UpdateSupplierQuantity('SUP001', '70')
```

```
NAME                           QUANT SUPPLIER_I
------------------------------ ----- ----------
Amoxicillin                    200   SUP001
Ibuprofen                      200   SUP002
Metformin                      150   SUP003
Lisinopril                     120   SUP004
Atorvastatin                   80    SUP005
Aspirin                        80    SUP005

6 rows selected.


Procedure UPDATESUPPLIERQUANTITY compiled


PL/SQL procedure successfully completed.


NAME                           QUANT SUPPLIER_I
------------------------------ ----- ----------
Amoxicillin                    70    SUP001
Ibuprofen                      200   SUP002
Metformin                      150   SUP003
Lisinopril                     120   SUP004
Atorvastatin                   80    SUP005
Aspirin                        80    SUP005
```

B.

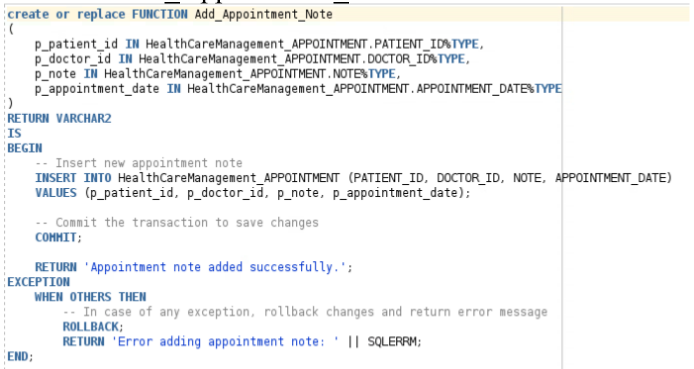


```java
/**
 * Method for a pharmacy employee to refill a certain medication from a supplier
 */
public String requestRefill(String supplierName, String amount) {
  Connection con = openDBConnection();
  String sql = "{CALL UpdateSupplierQuantity(?, ?)}";
  try (CallableStatement statement = con.prepareCall(sql)) {
    statement.setString(1, supplierName);
    statement.setString(2, amount);

    statement.execute();
    return "Medication quantity for "+supplierName+" updated to "+amount;
  } catch (SQLException e) {
    e.printStackTrace();
    return "Invalid Medication Name";
  }

}
```

C.

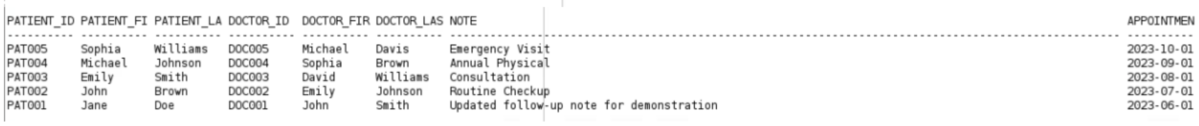
Mason's Stored Routines

A. Function Add_Appointment_Note



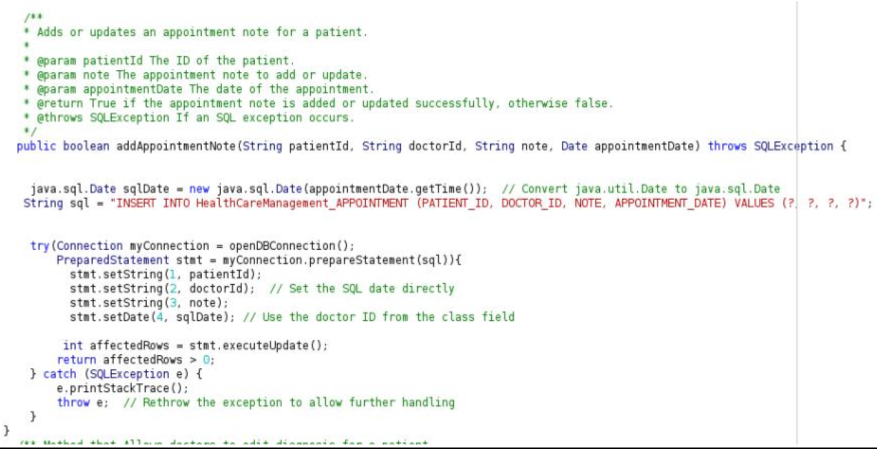
```sql
create or replace FUNCTION Add_Appointment_Note
(
    p_patient_id IN HealthCareManagement_APPOINTMENT.PATIENT_ID%TYPE,
    p_doctor_id IN HealthCareManagement_APPOINTMENT.DOCTOR_ID%TYPE,
    p_note IN HealthCareManagement_APPOINTMENT.NOTE%TYPE,
    p_appointment_date IN HealthCareManagement_APPOINTMENT.APPOINTMENT_DATE%TYPE
)
RETURN VARCHAR2
IS
BEGIN
    -- Insert new appointment note
    INSERT INTO HealthCareManagement_APPOINTMENT (PATIENT_ID, DOCTOR_ID, NOTE, APPOINTMENT_DATE)
    VALUES (p_patient_id, p_doctor_id, p_note, p_appointment_date);

    -- Commit the transaction to save changes
    COMMIT;

    RETURN 'Appointment note added successfully.';
EXCEPTION
    WHEN OTHERS THEN
        -- In case of any exception, rollback changes and return error message
        ROLLBACK;
        RETURN 'Error adding appointment note: ' || SQLERRM;
END;
```

B.



```
PATIENT_ID PATIENT_FI PATIENT_LA DOCTOR_ID  DOCTOR_FIR DOCTOR_LAS NOTE                                                                                                        APPOINTMEN
---------- ---------- ---------- ---------- ---------- ---------- ---------------------------------------------------------------------------------------------------------- ----------
PAT005     Sophia     Williams   DOC005     Michael    Davis      Emergency Visit                                                                                             2023-10-01
PAT004     Michael    Johnson    DOC004     Sophia     Brown      Annual Physical                                                                                             2023-09-01
PAT003     Emily      Smith      DOC003     David      Williams   Consultation                                                                                                2023-08-01
PAT002     John       Brown      DOC002     Emily      Johnson    Routine Checkup                                                                                             2023-07-01
PAT001     Jane       Doe        DOC001     John       Smith      Updated follow-up note for demonstration                                                                    2023-06-01
```

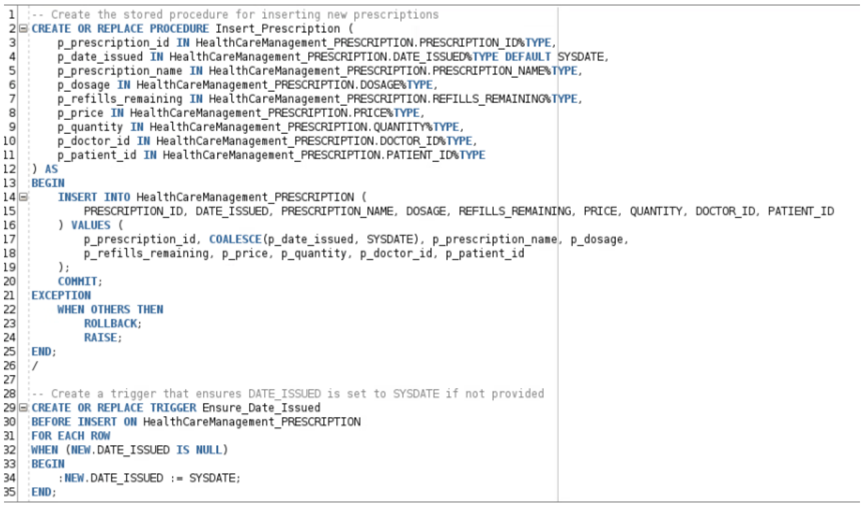
C.



```java
/**
 * Adds or updates an appointment note for a patient.
 *
 * @param patientId The ID of the patient.
 * @param note The appointment note to add or update.
 * @param appointmentDate The date of the appointment.
 * @return True if the appointment note is added or updated successfully, otherwise false.
 * @throws SQLException If an SQL exception occurs.
 */
public boolean addAppointmentNote(String patientId, String doctorId, String note, Date appointmentDate) throws SQLException {

  java.sql.Date sqlDate = new java.sql.Date(appointmentDate.getTime());  // Convert java.util.Date to java.sql.Date
 String sql = "INSERT INTO HealthCareManagement_APPOINTMENT (PATIENT_ID, DOCTOR_ID, NOTE, APPOINTMENT_DATE) VALUES (?, ?, ?, ?)";

  try(Connection myConnection = openDBConnection();
      PreparedStatement stmt = myConnection.prepareStatement(sql)){
        stmt.setString(1, patientId);
        stmt.setString(2, doctorId);  // Set the SQL date directly
        stmt.setString(3, note);
        stmt.setDate(4, sqlDate); // Use the doctor ID from the class field

       int affectedRows = stmt.executeUpdate();
      return affectedRows > 0;
  } catch (SQLException e) {
      e.printStackTrace();
      throw e;  // Rethrow the exception to allow further handling
  }
}
 /** Method that Allows doctors to edit diagnosis for a patient
```

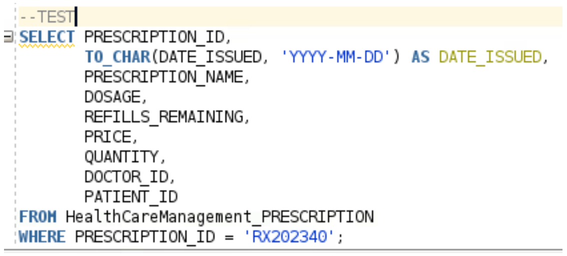
D.

A. Procedure and Trigger for Create_Prescription



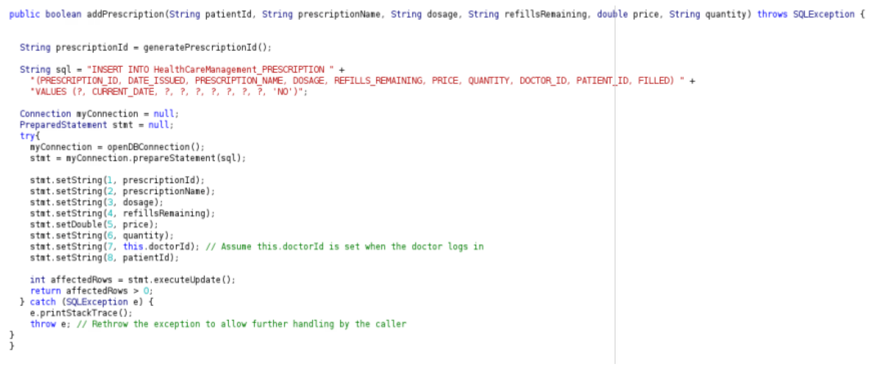
```
-- Create the stored procedure for inserting new prescriptions
CREATE OR REPLACE PROCEDURE Insert_Prescription (
    p_prescription_id IN HealthCareManagement_PRESCRIPTION.PRESCRIPTION_ID%TYPE,
    p_date_issued IN HealthCareManagement_PRESCRIPTION.DATE_ISSUED%TYPE DEFAULT SYSDATE,
    p_prescription_name IN HealthCareManagement_PRESCRIPTION.PRESCRIPTION_NAME%TYPE,
    p_dosage IN HealthCareManagement_PRESCRIPTION.DOSAGE%TYPE,
    p_refills_remaining IN HealthCareManagement_PRESCRIPTION.REFILLS_REMAINING%TYPE,
    p_price IN HealthCareManagement_PRESCRIPTION.PRICE%TYPE,
    p_quantity IN HealthCareManagement_PRESCRIPTION.QUANTITY%TYPE,
    p_doctor_id IN HealthCareManagement_PRESCRIPTION.DOCTOR_ID%TYPE,
    p_patient_id IN HealthCareManagement_PRESCRIPTION.PATIENT_ID%TYPE
) AS
BEGIN
    INSERT INTO HealthCareManagement_PRESCRIPTION (
        PRESCRIPTION_ID, DATE_ISSUED, PRESCRIPTION_NAME, DOSAGE, REFILLS_REMAINING, PRICE, QUANTITY, DOCTOR_ID, PATIENT_ID
    ) VALUES (
        p_prescription_id, COALESCE(p_date_issued, SYSDATE), p_prescription_name, p_dosage,
        p_refills_remaining, p_price, p_quantity, p_doctor_id, p_patient_id
    );
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

-- Create a trigger that ensures DATE_ISSUED is set to SYSDATE if not provided
CREATE OR REPLACE TRIGGER Ensure_Date_Issued
BEFORE INSERT ON HealthCareManagement_PRESCRIPTION
FOR EACH ROW
WHEN (NEW.DATE_ISSUED IS NULL)
BEGIN
    :NEW.DATE_ISSUED := SYSDATE;
END;
```

B.



```
--TEST
SELECT PRESCRIPTION_ID,
       TO_CHAR(DATE_ISSUED, 'YYYY-MM-DD') AS DATE_ISSUED,
       PRESCRIPTION_NAME,
       DOSAGE,
       REFILLS_REMAINING,
       PRICE,
       QUANTITY,
       DOCTOR_ID,
       PATIENT_ID
FROM HealthCareManagement_PRESCRIPTION
WHERE PRESCRIPTION_ID = 'RX202340';
```

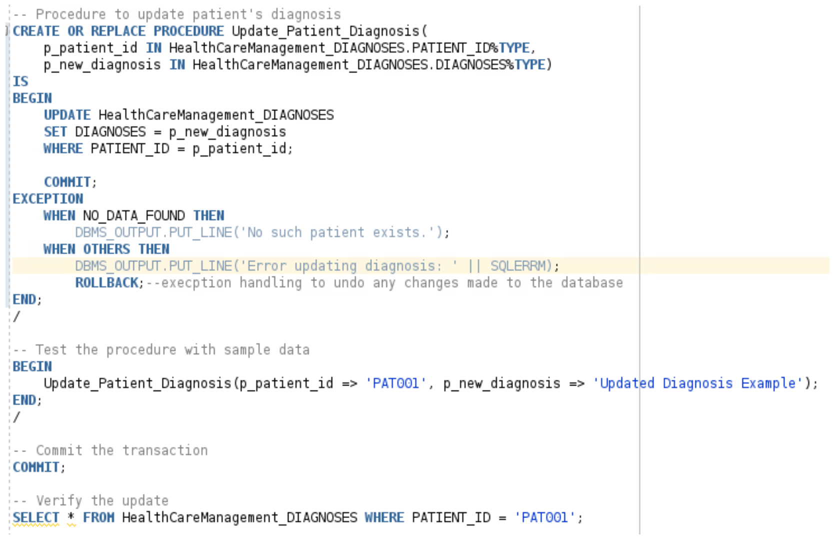
C.



```
public boolean addPrescription(String patientId, String prescriptionName, String dosage, String refillsRemaining, double price, String quantity) throws SQLException {

  String prescriptionId = generatePrescriptionId();

  String sql = "INSERT INTO HealthCareManagement_PRESCRIPTION " +
    "(PRESCRIPTION_ID, DATE_ISSUED, PRESCRIPTION_NAME, DOSAGE, REFILLS_REMAINING, PRICE, QUANTITY, DOCTOR_ID, PATIENT_ID, FILLED) " +
    "VALUES (?, CURRENT_DATE, ?, ?, ?, ?, ?, ?, ?, 'NO')";

  Connection myConnection = null;
  PreparedStatement stmt = null;
  try{
    myConnection = openDBConnection();
    stmt = myConnection.prepareStatement(sql);

    stmt.setString(1, prescriptionId);
    stmt.setString(2, prescriptionName);
    stmt.setString(3, dosage);
    stmt.setString(4, refillsRemaining);
    stmt.setDouble(5, price);
    stmt.setString(6, quantity);
    stmt.setString(7, this.doctorId); // Assume this.doctorId is set when the doctor logs in
    stmt.setString(8, patientId);

    int affectedRows = stmt.executeUpdate();
    return affectedRows > 0;
  } catch (SQLException e) {
    e.printStackTrace();
    throw e; // Rethrow the exception to allow further handling by the caller
  }
}
```

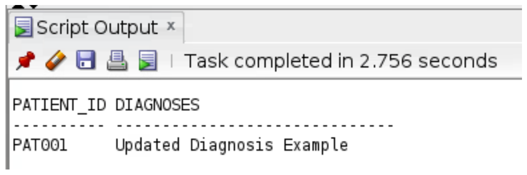
D.

A. Update Patient Diagnosis Procedure



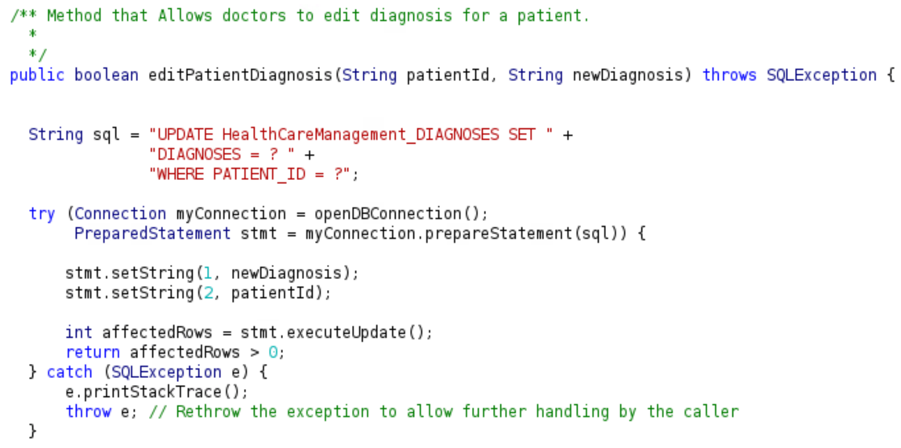
```
-- Procedure to update patient's diagnosis
CREATE OR REPLACE PROCEDURE Update_Patient_Diagnosis(
    p_patient_id IN HealthCareManagement_DIAGNOSES.PATIENT_ID%TYPE,
    p_new_diagnosis IN HealthCareManagement_DIAGNOSES.DIAGNOSES%TYPE)
IS
BEGIN
    UPDATE HealthCareManagement_DIAGNOSES
    SET DIAGNOSES = p_new_diagnosis
    WHERE PATIENT_ID = p_patient_id;

    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No such patient exists.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error updating diagnosis: ' || SQLERRM);
        ROLLBACK;--execption handling to undo any changes made to the database
END;
/

-- Test the procedure with sample data
BEGIN
    Update_Patient_Diagnosis(p_patient_id => 'PAT001', p_new_diagnosis => 'Updated Diagnosis Example');
END;
/

-- Commit the transaction
COMMIT;

-- Verify the update
SELECT * FROM HealthCareManagement_DIAGNOSES WHERE PATIENT_ID = 'PAT001';
```

B.



```
PATIENT_ID DIAGNOSES
---------- ------------------------------
PAT001     Updated Diagnosis Example
```

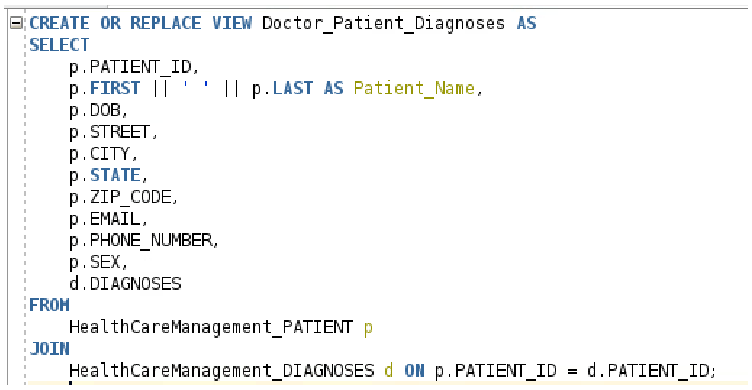
C.



```
/** Method that Allows doctors to edit diagnosis for a patient.
  *
  */
public boolean editPatientDiagnosis(String patientId, String newDiagnosis) throws SQLException {


  String sql = "UPDATE HealthCareManagement_DIAGNOSES SET " +
               "DIAGNOSES = ? " +
               "WHERE PATIENT_ID = ?";

  try (Connection myConnection = openDBConnection();
       PreparedStatement stmt = myConnection.prepareStatement(sql)) {

      stmt.setString(1, newDiagnosis);
      stmt.setString(2, patientId);

      int affectedRows = stmt.executeUpdate();
      return affectedRows > 0;
  } catch (SQLException e) {
      e.printStackTrace();
      throw e; // Rethrow the exception to allow further handling by the caller
  }
```

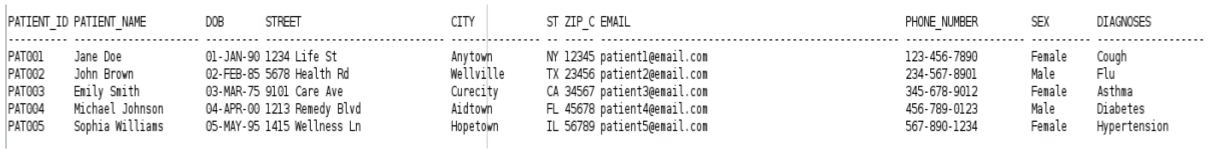
D.

A. View in GetPatientDetails

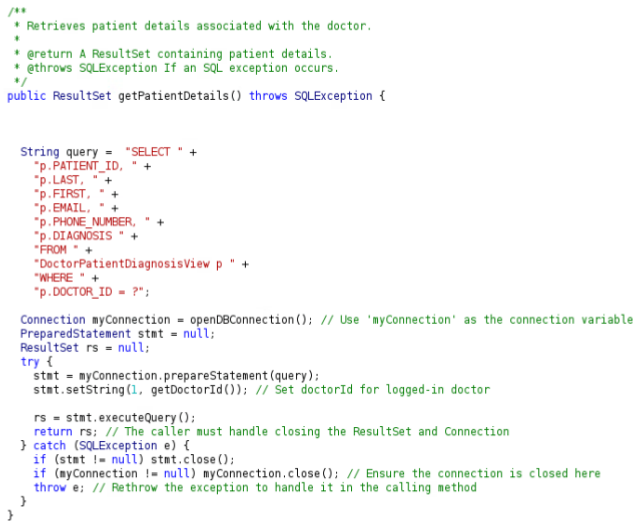


```
CREATE OR REPLACE VIEW Doctor_Patient_Diagnoses AS
SELECT
    p.PATIENT_ID,
    p.FIRST || ' ' || p.LAST AS Patient_Name,
    p.DOB,
    p.STREET,
    p.CITY,
    p.STATE,
    p.ZIP_CODE,
    p.EMAIL,
    p.PHONE_NUMBER,
    p.SEX,
    d.DIAGNOSES
FROM
    HealthCareManagement_PATIENT p
JOIN
    HealthCareManagement_DIAGNOSES d ON p.PATIENT_ID = d.PATIENT_ID;
```

B.

C. 

```
PATIENT_ID PATIENT_NAME         DOB       STREET                         CITY          ST ZIP_C EMAIL                                              PHONE_NUMBER         SEX        DIAGNOSES
---------- -------------------- --------- ------------------------------ ------------- -- ----- -------------------------------------------------- -------------------- ---------- ------------------
PAT001     Jane Doe             01-JAN-90 1234 Life St                   Anytown       NY 12345 patient1@email.com                                 123-456-7890         Female     Cough
PAT002     John Brown           02-FEB-85 5678 Health Rd                 Wellville     TX 23456 patient2@email.com                                 234-567-8901         Male       Flu
PAT003     Emily Smith          03-MAR-75 9101 Care Ave                  Curecity      CA 34567 patient3@email.com                                 345-678-9012         Female     Asthma
PAT004     Michael Johnson      04-APR-00 1213 Remedy Blvd               Aidtown       FL 45678 patient4@email.com                                 456-789-0123         Male       Diabetes
PAT005     Sophia Williams      05-MAY-95 1415 Wellness Ln               Hopetown      IL 56789 patient5@email.com                                 567-890-1234         Female     Hypertension
```

D. 

```java
/**
 * Retrieves patient details associated with the doctor.
 *
 * @return A ResultSet containing patient details.
 * @throws SQLException If an SQL exception occurs.
 */
public ResultSet getPatientDetails() throws SQLException {


  String query =  "SELECT " +
    "p.PATIENT_ID, " +
    "p.LAST, " +
    "p.FIRST, " +
    "p.EMAIL, " +
    "p.PHONE_NUMBER, " +
    "p.DIAGNOSIS " +
    "FROM " +
    "DoctorPatientDiagnosisView p " +
    "WHERE " +
    "p.DOCTOR_ID = ?";

  Connection myConnection = openDBConnection(); // Use 'myConnection' as the connection variable
  PreparedStatement stmt = null;
  ResultSet rs = null;
  try {
    stmt = myConnection.prepareStatement(query);
    stmt.setString(1, getDoctorId()); // Set doctorId for logged-in doctor

    rs = stmt.executeQuery();
    return rs; // The caller must handle closing the ResultSet and Connection
  } catch (SQLException e) {
    if (stmt != null) stmt.close();
    if (myConnection != null) myConnection.close(); // Ensure the connection is closed here
    throw e; // Rethrow the exception to handle it in the calling method
  }
}
```