Pharmacy Store

SIDHANT KR AGRAWAL(2021495) || PARAS DHIMAN(2021482)

Relationship Schema:

Patient = (PatientID, Fname, Mname, Lname, Street, City, Zip Code, State, Country, DOB, type CN, Contact No., Gender, Email,Age)

Doctors = (DoctorID, Fname, Mname, Lname, type CN, Contact No., Email)

Prescription = (PrescriptionID, DoctorID, PatientID, Dosage, No_Of_Days, StartDate, EndDate, Date)

Medicine = (MedicineID, Name, price)

Pharmacist = (PharmacistID, Name, typeCN, contactNo., Email, yearOfGraduation, Field of Study, branch ID)

Suppliers = (SupplierID,Name,TypeCN,Contact No.,Email,Address
)

Appointment = (AppointmentID, PatientID, DoctorID, Date, Time, Reason)

Orders = (OrderID,
PatientID,MedicineID,PharmacistID,Quantity,Date,Delivery Status)

Payments = (PaymentID, OrderID, Amount, Payment Method, Date)

Branches = (BranchID, Location, Contact No., Email)

Stock = (StockID,MedicineID,BranchID,Quantity,Threshold,PharmacistID)

Policy Provider = (ProviderID, Name, Contact No., Email, Address, PatientID)

Medical History = (MedicalHistID, PatientID, Description)

QUERIES

1. **Selection:**
   select * from appointment where reason = 'Vaccination';

   Relational Algebra:
   σ(reason = 'Vaccination')(appointment)

   This query uses the following relational algebra operation:
   σ (Selection): Selects only the tuples in the appointment relation where the value of the reason attribute is 'Vaccination'.

2. **Projection:**
   select Doctor_Name_Fname,Doctor_Name_Lname FROM Doctors;

   Relational Algebra:
   π(Doctor_Name_Fname, Doctor_Name_Lname)(Doctors)

   This query uses the following relational algebra operation:
   π (Projection): Projects only the Doctor_Name_Fname and Doctor_Name_Lname attributes from the Doctors relation.

3. **Extended Projection:**
   SELECT CONCAT('Mr. ', Name_Fname, ' ',Name_Mname,' ',Name_Lname) AS full_name FROM Patients;

   Relational Algebra:
   ρ(full_name/('Mr. ' || Name_Fname || ' ' || Name_Mname || ' ' || Name_Lname))(Patients)

   This query uses the following relational algebra operation:
   ρ (Renaming): Renames the resulting relation of the Patients relation, giving the attribute composed of the concatenation of 'Mr. ', Name_Fname, Name_Mname, and Name_Lname the name full_name.
   The resulting relation will have a single attribute, full_name, representing the full name of each patient with the prefix 'Mr.' added. The concatenation operation is represented using the double pipe (||) operator in the relational algebra expression.

4. **Product :**
   SELECT * FROM medicines, Pharmacists;

   Relational Algebra:
   medicines × Pharmacists

   This query uses the following relational algebra operation:

   × (Cartesian Product): Produces a relation that includes all possible combinations of tuples from both the medicines and Pharmacists relations.

5. **Theta-Join:**
   SELECT PrescriptionID, Dosage, StartDate, EndDate
   FROM Prescription JOIN Doctors
   ON Prescription.DoctorID = Doctors.DoctorID
   WHERE Prescription.PatientID > 93;

   Relational Algebra:
   σ(PatientID > 93) (Prescription ⋈ Doctors)

   This query uses the following relational algebra operations:
   σ (Selection): Selects all tuples from the resulting relation of the join operation where the PatientID is greater than 93.
   ⋈ (Join): Performs an inner join between the Prescription and Doctors relations, using the common attribute DoctorID, to produce a relation that includes attributes from both relations.

6. **Natural Join:**
   SELECT Orders.OrderID, Orders.Quantity, Orders.Date, Orders.DeliveryStatus, Patients.Name_Fname, Patients.Name_Lname, Medicines.Name
   FROM Orders NATURAL JOIN Patients NATURAL JOIN Medicines;

   Relational Algebra:
   Orders ⋈ Patients ⋈ Medicines

   This query uses the following relational algebra operation:
   ⋈ (Natural Join): Performs a natural join between the Orders, Patients, and Medicines relations, using the common attributes OrderID and MedicineID, to produce a relation that includes attributes from all three relations.

7. **Renaming**:
   SELECT PolicyProviderID AS code, PolicyProvider_Name_Fname FROM PolicyProvider;

   Relational Algebra:
   π(code:PolicyProviderID, PolicyProvider_Name_Fname)(PolicyProvider)

   This query uses the following relational algebra operation:
   π (Projection): Projects the PolicyProviderID attribute as code and the PolicyProvider_Name_Fname attribute from the PolicyProvider relation.

8. **Expression Trees:**
   SELECT SUM(Amount) FROM Payment;

   Relational Algebra:
   γ(total_amount:SUM(Amount))(Payment)

   This query uses the following relational algebra operation:
   γ (Grouping): Calculates the sum of the Amount attribute for all tuples in the Payment relation and stores the result in an attribute named total_amount.

9. **Group By:**
   SELECT PaymentMethod, SUM(Amount) AS total_amount
   FROM Payment
   GROUP BY PaymentMethod;

   Relational Algebra:
   γ(PaymentMethod, total_amount:SUM(Amount))(Payment)

   This query uses the following relational algebra operation:
   γ (Grouping): Groups the tuples of the Payment relation by PaymentMethod and calculates the sum of Amount for each group, which is named as total_amount.

10. **Subquery:**
    SELECT * FROM Prescription
    WHERE PrescriptionID IN

(SELECT PrescriptionID FROM Prescription WHERE Dosage >= 50);

Relational Algebra:
Prescription ⋈ (π(PrescriptionID)(σ(Dosage >= 50)(Prescription)))

This query uses the following relational algebra operations:
σ (Selection): Selects all tuples from the Prescription relation where the Dosage is greater than or equal to 50.
π (Projection): Projects the PrescriptionID attribute from the resulting relation of the previous selection operation.
⋈ (Semi-Join): Performs a semi-join between the Prescription relation and the resulting relation of the projection operation, using the common attribute PrescriptionID, to produce a relation that includes all attributes from the Prescription relation.

11. **ALTER:**

ALTER TABLE Stock DROP PRIMARY KEY;
Relational Algebra:
π (all attributes except the primary key) (Stock)

ALTER TABLE Appointment DROP PRIMARY KEY;
Relational Algebra:
π (all attributes except the primary key) (Appointment)

12. **UPDATE:**
UPDATE Doctors SET Doctor_Name_Lname = 'Doe' WHERE DoctorID = 1;

Some Complex Queries are as follows:-

1. ***The name and contact information of the doctor who prescribed a certain medication to a patient:***

SELECT Doctor_Name_Fname, Doctor_Name_Mname, Doctor_Name_Lname,
Doctors.ContactNumber_number, Doctors.Email_address
FROM Doctors
JOIN Prescription ON Prescription.DoctorID = Doctors.DoctorID
JOIN Medicines ON Medicines.MedicineID = Prescription.MedicineID
JOIN Patients ON Patients.PatientID = Prescription.PatientID
WHERE Patients.PatientID = 4 AND Medicines.Name = 'Paris Parkman';

Relational Algebra:
σ(Patient.PatientID = [patient ID] ∧ Medicine.Name = '[medication name]')
(Doctors ⋈ Prescription ⋈ Medicine ⋈ Patient)π(Doctors.Fname, Doctors.Mname, Doctors.Lname, Doctors.ContactNo, Doctors.Email)

Explanation:

σ(Patient.PatientID = [patient ID] ∧ Medicine.Name = '[medication name]') applies the given conditions to filter out the relevant tuples from the Cartesian product of all tables.
Doctors ⋈ Prescription ⋈ Medicine ⋈ Patient performs a natural join of all the tables based on the join conditions specified in the query.
π(Doctors.Fname, Doctors.Mname, Doctors.Lname, Doctors.ContactNo, Doctors.Email) selects the required columns from the resulting joined relation.

2. ***The average age of patients who visit a particular doctor:***

```
SELECT AVG(Patients.Age) AS AvgAge
FROM Patients
JOIN Appointment ON Appointment.PatientID = Patients.PatientID
JOIN Doctors ON Doctors.DoctorID = Appointment.DoctorID
WHERE Doctors.DoctorID = 69;
```

Relational Algebra:
π(AvgAge)(Σ(Patient.Age)(σ(Doctors.DoctorID = [doctor ID])(Patient ⋈ Appointment ⋈ Doctors)))

Explanation:

σ(Doctors.DoctorID = [doctor ID]) applies the given condition to filter out the tuples from the Cartesian product of all tables, such that only the tuples with the specified doctor ID are selected.
Patient ⋈ Appointment ⋈ Doctors performs a natural join of all the tables based on the join conditions specified in the query.
Σ(Patient.Age) calculates the sum of ages of all the selected patients.
Σ(Patient.Age) is then divided by the total number of selected patients to get the average age.
π(AvgAge) selects the resulting column and renames it as AvgAge.

3. ***The total amount of money spent by a patient on medications from a particular supplier***

```
SELECT SUM(Medicines.price * Orders.Quantity) AS TotalSpent
FROM Patients
```

```
JOIN Orders ON Orders.PatientID = Patients.PatientID
JOIN Medicines ON Medicines.MedicineID = Orders.MedicineID
JOIN Pharmacists ON Pharmacists.PharmacistID = Orders.PharmacistID
JOIN Suppliers ON Suppliers.SupplierID = Medicines.SupplierID
WHERE Patients.PatientID = 69 AND Suppliers.Supplier_Name_Fname = 'Gregory';
```

Relational Algebra:
Σ(TotalSpent)(π(Orders.Quantity × Medicine.price)(Patients ⋈ Orders ⋈ Medicines ⋈ Pharmacists ⋈ Suppliers))

Where:
π(Orders.Quantity × Medicine.price) selects the columns Orders.Quantity and Medicine.price and computes their product for each tuple
Patient ⋈ Orders ⋈ Medicine ⋈ Pharmacist ⋈ Suppliers performs a natural join between all the tables, based on the join conditions specified in the SQL query
Σ(TotalSpent) aggregates the result by computing the sum of the product of Orders.Quantity and Medicine.price, and renames the result column as TotalSpent.

4. ***The number of appointments a patient has had with each doctor***

```
SELECT Doctors.DoctorID, Doctors.Doctor_Name_Fname,
Doctors.Doctor_Name_Mname, Doctors.Doctor_Name_Lname, COUNT(*) AS
NumAppointments
FROM Doctors
JOIN Appointment ON Appointment.DoctorID = Doctors.DoctorID
JOIN Patients ON Patients.PatientID = Appointment.PatientID
WHERE Patients.PatientID = 69
GROUP BY Doctors.DoctorID, Doctors.Doctor_Name_Fname,
Doctors.Doctor_Name_Mname, Doctors.Doctor_Name_Lname
ORDER BY NumAppointments DESC;
```

Relational Algebra:
σ(Patient.PatientID = [patient ID])(Doctors ⋈ Appointment ⋈ Patient)
→ γ(DoctorID, Fname, Mname, Lname, COUNT(*) → NumAppointments)
→ ρ(DoctorID, Fname, Mname, Lname, NumAppointments → NumApps)
→ σ(DESC(NumApps))(π(DoctorID, Fname, Mname, Lname, NumApps)(DoctorAppointments))

5. ***The name and contact information of the pharmacist who has the most stock of a certain medication:***

```
SELECT Pharmacists.Pharmacists_Name_Fname,
Pharmacists.ContactNumber_number, Pharmacists.Email_Address
FROM Pharmacists
JOIN Stock ON Stock.PharmacistID = Pharmacists.PharmacistID
JOIN Medicines ON Medicines.MedicineID = Stock.MedicineID
WHERE Medicines.Ingredient = 'Charles Extract'
ORDER BY Stock.Quantity DESC
LIMIT 1;
```

Relational Algebra:

σ(Name, Contact No., Email)(
σ(rownum = 1)(σ(DESC(Quantity))(
σ(Medicine.Name = '[medication name]')(Pharmacist ⋈ Stock ⋈ Medicine))))

6. ***The number of prescriptions issued by each doctor for a particular medication, along with the total dosage prescribed***

```
SELECT  Doctors.Doctor_Name_Fname,  Doctors.Doctor_Name_Mname,
Doctors.Doctor_Name_Lname, COUNT(*) AS NumPrescriptions,
SUM(Prescription.Dosage) AS TotalDosage
FROM Doctors
JOIN Prescription ON Prescription.DoctorID = Doctors.DoctorID
JOIN Medicines ON Medicines.MedicineID = Prescription.MedicineID
WHERE Medicines.Type = 'Tablet'
GROUP BY Doctors.Doctor_Name_Fname, Doctors.Doctor_Name_Mname,
Doctors.Doctor_Name_Lname
ORDER BY NumPrescriptions DESC;
```

Relational Algebra:

σ(Medicine.Name = '[medication name]')(Doctors ⋈ Prescription ⋈ Medicine)
→ γ(Fname, Mname, Lname, COUNT(*) → NumPrescriptions, SUM(Dosage) → TotalDosage)
→ σ(DESC(NumPrescriptions))(DoctorPrescriptions)

7. ***Find the most commonly prescribed medication for each doctor***

```
SELECT Doctors.DoctorID, Doctors.Doctor_Name_Fname,
Doctors.Doctor_Name_Lname, Medicines.Name, COUNT(*) AS PrescriptionCount
FROM Doctors
JOIN Prescription ON Prescription.DoctorID = Doctors.DoctorID
JOIN Medicines ON Medicines.MedicineID = Prescription.MedicineID
GROUP BY Doctors.DoctorID, Medicines.MedicineID
```

```
HAVING COUNT(*) = (
   SELECT MAX(Count) FROM (
      SELECT DoctorID, MedicineID, COUNT(*) AS Count
      FROM Prescription
      GROUP BY DoctorID, MedicineID
   ) AS T
   WHERE T.DoctorID = Doctors.DoctorID
);
```

Relational Algebra:

```
π Doctor.DoctorID, Doctor.Doctor_Name_Fname, Doctor.Doctor_Name_Lname,
Medicines.Name, PrescriptionCount
(ρ Doctor.DoctorID → Prescription.DoctorID, Doctor.Doctor_Name_Fname →
Doctor_Name_Fname, Doctor.Doctor_Name_Lname → Doctor_Name_Lname,
Prescription.MedicineID → Medicines.MedicineID, PrescriptionCount → COUNT(*))
(σ PrescriptionCount =
  (γ DoctorID, MedicineID, MAX(Count))
  (ρ DoctorID → T.DoctorID, MedicineID → T.MedicineID, Count → T.Count)
  (γ DoctorID, MedicineID, COUNT(*) → Count)
  (Prescription)
  (σ Doctor.DoctorID = T.DoctorID)
)
(Doctor ⋈ Prescription ⋈ Medicines)
```

8. ***The top 3 most common medical conditions for patients who have visited a particular doctor:***

```
SELECT Medicalhistory.Description, COUNT(*) AS NumPatients
FROM MedicalHistory
JOIN Patients ON Patients.PatientID = MedicalHistory.PatientID
JOIN Appointment ON Appointment.PatientID = Patients.PatientID
JOIN Doctors ON Doctors.DoctorID = Appointment.DoctorID
WHERE Doctors.DoctorID = 69
GROUP BY MedicalHistory.Description
ORDER BY NumPatients DESC
LIMIT 3;
```

Relational Algebra:

π MedicalHistory.Description, COUNT(*) AS NumPatients (σ Doctors.DoctorID = [doctor ID] ∧ Doctors.DoctorID = Appointment.DoctorID ∧ Appointment.PatientID = Patient.PatientID ∧ Patient.PatientID = MedicalHistory.PatientID (MedicalHistory ⋈ Patient ⋈ Appointment ⋈ Doctors))

9. ***The total revenue generated by each supplier for a particular medication:***

```
SELECT Suppliers.Supplier_Name_Fname, SUM(Medicines.price * Orders.Quantity) AS TotalRevenue
FROM Suppliers
JOIN Medicines ON Medicines.SupplierID = Suppliers.SupplierID
JOIN Orders ON Orders.MedicineID = Medicines.MedicineID
WHERE Medicines.Name = 'Paris Parkman'
GROUP BY Suppliers.Supplier_Name_Fname
ORDER BY TotalRevenue DESC;
```

Relational Algebra:

π Suppliers.Name, SUM(Medicine.price * Orders.Quantity) AS TotalRevenue (σ Medicine.Name = '[medication name]' ∧ Medicine.MedicineID = Orders.MedicineID ∧ Medicine.SupplierID = Suppliers.SupplierID (Suppliers ⋈ Medicine ⋈ Orders) ÷ Medicine.SupplierID)

10. ***The name and contact information of the pharmacist who has the most overall stock across all medications:***

```
SELECT Pharmacists.Pharmacists_Name_Fname,
Pharmacists.ContactNumber_number, Pharmacists.Email_Address,
SUM(Stock.Quantity) AS TotalStock
FROM Pharmacists
JOIN Stock ON Stock.PharmacistID = Pharmacists.PharmacistID
GROUP BY Pharmacists.Pharmacists_Name_Fname,
Pharmacists.ContactNumber_number, Pharmacists.Email_Address
ORDER BY TotalStock DESC
LIMIT 1;
```

Relational Algebra:

π Pharmacist.Name, Pharmacist.Contact No., Pharmacist.Email, ℱ(SUM(Stock.Quantity)) AS TotalStock (
Pharmacist ⋈ Stock.PharmacistID = Pharmacist.PharmacistID (
ρ ℱ(Stock.Quantity) (σ True(Medicine) ⋈ Stock.MedicineID = Medicine.MedicineID (Medicine)))) ÷ {Pharmacist.Name, Pharmacist.Contact No., Pharmacist.Email} ⋉ TotalStock DESC ⋉ 1;

1 update and one for the alter one

11. Update Command

```
UPDATE Patients p
JOIN PolicyProvider pp ON p.PatientID = pp.PatientID
JOIN MedicalHistory mh ON p.PatientID = mh.PatientID
JOIN Prescription pr ON p.PatientID = pr.PatientID
JOIN Orders o ON p.PatientID = o.PatientID
JOIN Stock s ON o.MedicineID = s.MedicineID AND o.PharmacistID = s.PharmacistID
AND o.PharmacistID = s.PharmacistID
JOIN Pharmacists ph ON s.PharmacistID = ph.PharmacistID
SET p.Email_address = 'newemail@example.com', ph.ContactNumber_number =
'5555555555'
WHERE p.PatientID = 12345;
```

Constraints:-

```
CREATE TABLE Patients (
  PatientID INT PRIMARY KEY,
  Name_Fname VARCHAR(50),
  Name_Mname VARCHAR(50),
  Name_Lname VARCHAR(50),
  Address_street_number VARCHAR(50),
  Address_street_name VARCHAR(50),
  Address_apt_number VARCHAR(50),
  Address_city VARCHAR(50),
  Address_zipCode INT,
  Address_state VARCHAR(50),
  Address_country VARCHAR(50),
  DOB DATE,
  ContactNumber_type VARCHAR(50),
  ContactNumber_number INT,
  Gender CHAR(1),
  Email_type VARCHAR(50),
  Email_address VARCHAR(100),
  Age INT
);

CREATE TABLE Doctors
(
   DoctorID INT PRIMARY KEY,
   Doctor_Name_Fname VARCHAR(50) NOT NULL,
   Doctor_Name_Mname VARCHAR(50),
```

```
    Doctor_Name_Lname VARCHAR(50) NOT NULL,
    Specialisation_Description VARCHAR(100) NOT NULL,
    ContactNumber_type VARCHAR(20) NOT NULL,
    ContactNumber_number INT NOT NULL,
    Email_type VARCHAR(10) NOT NULL,
    Email_address VARCHAR(100) NOT NULL
);

CREATE TABLE Suppliers
(
    SupplierID INT PRIMARY KEY,
    Supplier_Name_Fname VARCHAR(50) NOT NULL,
    Supplier_Name_Mname VARCHAR(50) NOT NULL,
    Supplier_Name_Lname VARCHAR(50) NOT NULL,
    ContactNumber_type VARCHAR(20) NOT NULL,
    ContactNumber_number VARCHAR(20) NOT NULL,
    Email_Address VARCHAR(50) NOT NULL,
    Email_Type VARCHAR(20) NOT NULL,
    Address VARCHAR(100) NOT NULL,
    Address_Type VARCHAR(20) NOT NULL
);

CREATE TABLE Medicines
(
    MedicineID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Composition VARCHAR(50) NOT NULL,
    Ingredient VARCHAR(50) NOT NULL,
    Type VARCHAR(20) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    Cost DECIMAL(10, 2) NOT NULL,
    Currency VARCHAR(10) NOT NULL,
    SupplierID INT NOT NULL,
    FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)
);

CREATE TABLE Prescription
(
    PrescriptionID INT PRIMARY KEY,
    DoctorID INT NOT NULL,
    PatientID INT NOT NULL,
    MedicineID INT NOT NULL,
    Dosage INT NOT NULL,
    Duration_Days INT NOT NULL,
```

```
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    Date DATE NOT NULL,
    DateWritten DATE NOT NULL,
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID)
);

CREATE TABLE Pharmacists (
    PharmacistID INT PRIMARY KEY,
    Pharmacists_Name_Fname VARCHAR(255),
    Pharmacists_Name_Mname VARCHAR(255),
    Pharmacists_Name_Lname VARCHAR(255),
    ContactNumber_type VARCHAR(255),
    ContactNumber_number VARCHAR(255),
    Email_Address VARCHAR(255),
    Email_Type VARCHAR(255),
    Qualification_yearOfGraduation INT,
    Qualification_FieldOfStudy VARCHAR(255)
);

CREATE TABLE MedicalHistory (
    MedicalHistID INT PRIMARY KEY,
    PatientID INT,
    Description VARCHAR(255),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

CREATE TABLE Payment (
  PaymentID INT PRIMARY KEY,
  OrderID INT,
  Amount DECIMAL(10, 2) NOT NULL,
  PaymentMethod VARCHAR(50) NOT NULL,
  PaymentDate DATE NOT NULL
);

CREATE TABLE Branches (
  BranchID INT PRIMARY KEY,
  street_number VARCHAR(255) NOT NULL,
  street_name VARCHAR(255) NOT NULL,
  apt_number VARCHAR(255),
  city VARCHAR(255) NOT NULL,
  zipCode VARCHAR(255) NOT NULL,
```

```
  state VARCHAR(255) NOT NULL,
  country VARCHAR(255) NOT NULL,
  ContactNumber VARCHAR(255) NOT NULL,
  Email VARCHAR(255) NOT NULL
);

CREATE TABLE Stock (
    StockID INT PRIMARY KEY,
    MedicineID INT,
    BranchID INT,
    Quantity INT,
    Threshold INT,
    PharmacistID INT,
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID),
    FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),
    FOREIGN KEY (PharmacistID) REFERENCES Pharmacists(PharmacistID)
);

CREATE TABLE Appointment (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    Date DATE,
    Time TIME,
    Reason VARCHAR(255),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    PatientID INT,
    MedicineID INT,
    PharmacistID INT,
    Quantity INT,
    Date DATE,
    DeliveryStatus VARCHAR(255),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID),
    FOREIGN KEY (PharmacistID) REFERENCES Pharmacists(PharmacistID)
);

CREATE TABLE PolicyProvider (
  PolicyProviderID INT PRIMARY KEY,
```

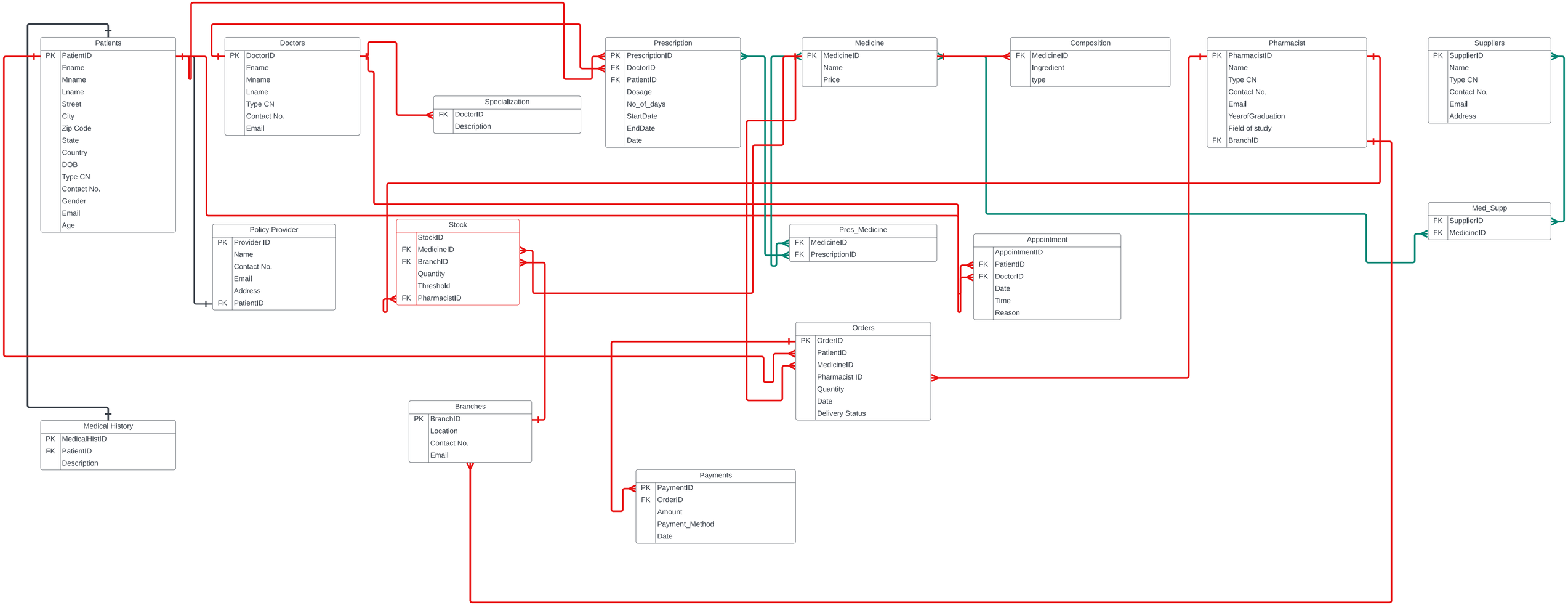
```
  PolicyProvider_Name_Fname VARCHAR(255) NOT NULL,
  PolicyProvider_Name_Mname VARCHAR(255),
  PolicyProvider_Name_Lname VARCHAR(255) NOT NULL,
  ContactNumber VARCHAR(255) NOT NULL,
  Email VARCHAR(255) NOT NULL,
  Address VARCHAR(255) NOT NULL,
  PatientID INT NOT NULL,
  FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);
CREATE TABLE Pres_medicine (
    PMID int NOT NULL PRIMARY KEY,
    MedicineID int NOT NULL,
    PrescriptionID int NOT NULL,
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID),
    FOREIGN KEY (PrescriptionID) REFERENCES Prescription(PrescriptionID)
);
CREATE TABLE Med_Supp(
    MSID int NOT NULL PRIMARY KEY,
    MedicineID int NOT NULL,
    SupplierID int NOT NULL,
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID),
    FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)
);
```

```
ALTER TABLE Suppliers
MODIFY ContactNumber_number BIGINT;
ALTER TABLE Pharmacists
MODIFY ContactNumber_number BIGINT;
ALTER TABLE Branches
MODIFY ContactNumber BIGINT;
ALTER TABLE PolicyProvider
MODIFY ContactNumber BIGINT NOT NULL;
```

```
ALTER TABLE Patients ADD CONSTRAINT check_zipCode CHECK (Address_zipCode
BETWEEN 10000 AND 999999);
ALTER TABLE Patients ADD CONSTRAINT check_email_address CHECK
(Email_address LIKE '%@%.%');
ALTER TABLE Patients ADD CONSTRAINT check_age CHECK (Age >= 0 AND Age <=
150);
ALTER TABLE Doctors ADD CONSTRAINT doc_check_email_address CHECK
(Email_address LIKE '%@%.%');
ALTER TABLE Doctors ADD CONSTRAINT unique_email UNIQUE (Email_address);
```

```
ALTER TABLE Doctors ADD CONSTRAINT unique_contact UNIQUE
(ContactNumber_number);
ALTER TABLE Suppliers ADD CONSTRAINT supp_check_email_address CHECK
(Email_Address LIKE '%@%.%');
ALTER TABLE Suppliers ADD CONSTRAINT sup_unique_email UNIQUE
(Email_Address);
ALTER TABLE Suppliers ADD CONSTRAINT sup_unique_contact UNIQUE
(ContactNumber_number);
ALTER TABLE Suppliers ADD CONSTRAINT sup_check_contact_type CHECK
(ContactNumber_type IN ('Mobile', 'Home', 'Work'));
ALTER TABLE Medicines ADD CONSTRAINT unique_medicine_name UNIQUE
(Name);
ALTER TABLE Medicines ADD CONSTRAINT med_price_positive CHECK (Price >= 0);
ALTER TABLE Medicines ADD CONSTRAINT med_cost_positive CHECK (Cost >= 0);
ALTER TABLE Pharmacists ADD CONSTRAINT pharma_unique_contact_number
UNIQUE (ContactNumber_number);
ALTER TABLE Stock ADD CONSTRAINT check_threshold CHECK (Threshold >= 0);
ALTER TABLE Stock ADD CONSTRAINT check_quantity CHECK (Quantity >= 0);
ALTER TABLE Orders ADD CONSTRAINT check_orders_quantity CHECK (Quantity >=
0);
ALTER TABLE PolicyProvider ADD CONSTRAINT pp_check_email_address CHECK
(Email LIKE '%@%.%');
ALTER TABLE PolicyProvider ADD CONSTRAINT pp_unique_email UNIQUE (Email);

ALTER TABLE Stock DROP PRIMARY KEY;
ALTER TABLE Appointment DROP PRIMARY KEY;
```


Patients
PK PatientID
Fname
Mname
Lname
Street
City
Zip Code
State
Country
DOB
Type CN
Contact No.
Gender
Email
Age
Doctors
PK DoctorID
Fname
Mname
Lname
Type CN
Contact No.
Email
Specialization
FK DoctorID
Description
Prescription
PK PrescriptionID
FK DoctorID
FK PatientID
Dosage
No_of_days
StartDate
EndDate
Date
Medicine
PK MedicineID
Name
Price
Composition
FK MedicineID
Ingredient
type
Pharmacist
PK PharmacistID
Name
Type CN
Contact No.
Email
YearofGraduation
Field of study
FK BranchID
Suppliers
PK SupplierID
Name
Type CN
Contact No.
Email
Address
Med_Supp
FK SupplierID
FK MedicineID
Policy Provider
PK Provider ID
Name
Contact No.
Email
Address
FK PatientID
Stock
StockID
FK MedicineID
FK BranchID
Quantity
Threshold
FK PharmacistID
Pres_Medicine
FK MedicineID
FK PrescriptionID
Appointment
AppointmentID
FK PatientID
FK DoctorID
Date
Time
Reason
Orders
PK OrderID
PatientID
MedicineID
Pharmacist ID
Quantity
Date
Delivery Status
Branches
PK BranchID
Location
Contact No.
Email
Payments
PK PaymentID
FK OrderID
Amount
Payment_Method
Date
Medical History
PK MedicalHistID
FK PatientID
Description