# Medical Records Database Creation

Sami Saliby, Nada Itani, Dalaa Hammoud, Peter Hajjar

School of Arts and Science

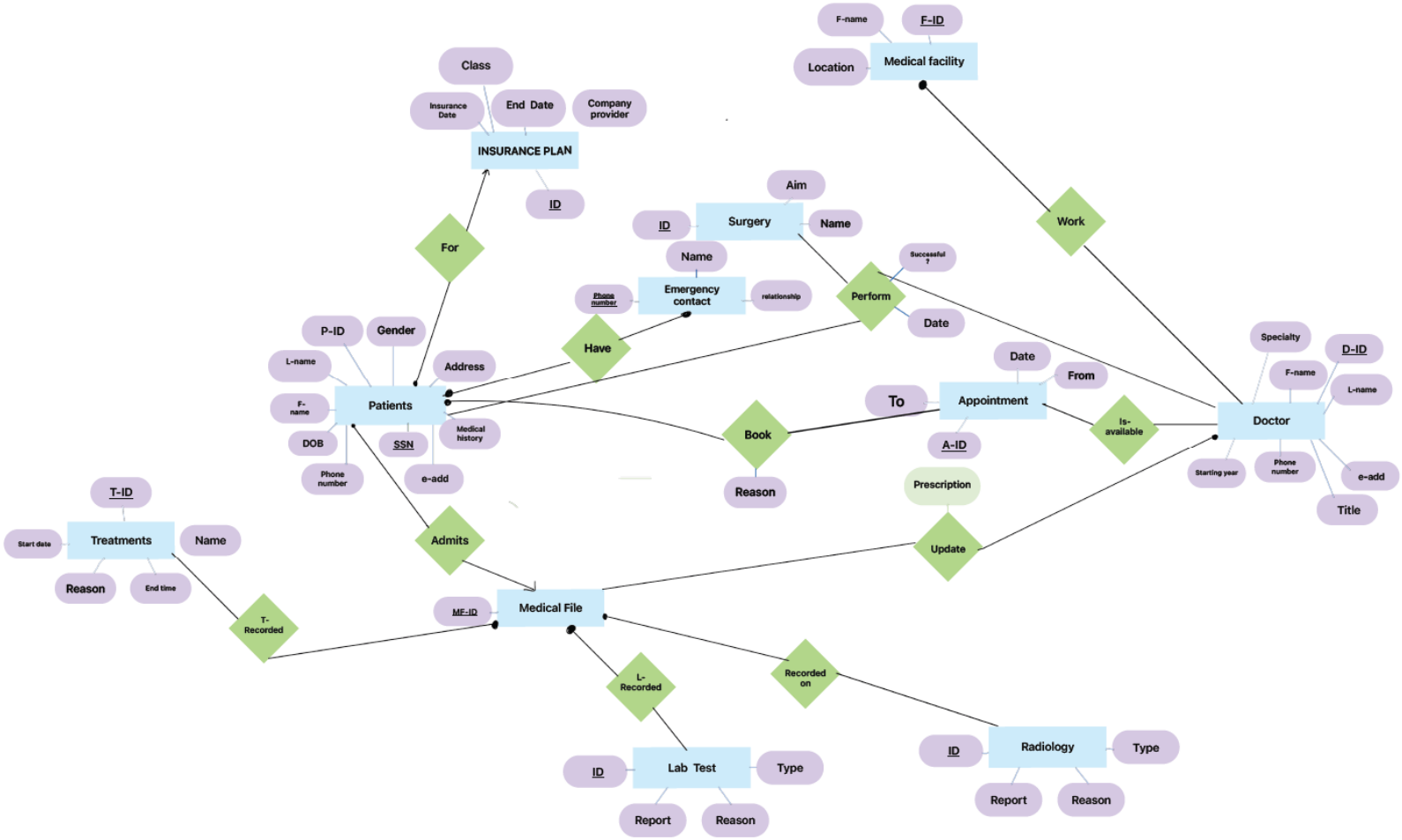
Dr. Ibrahim El Bitar

July 19th, 2024

## Introduction

We created a medical application for both patients and doctors. If the user is a patient, he can view his medical record that includes surgeries, treatments, prescriptions, lab test results, and radiology. He can also add and cancel appointments with many doctors. If the user is a doctor, he can view his schedule, manage appointments, and edit or view a patient's medical record.

## ER-Diagram



## ER Description

### Entity Set

1. **MedicalFacility**
    - **Attributes:** F_ID (Primary Key), Fname, location.
    - **Description:** Represents medical facilities, each identified by a unique facility ID.
2. **Doctor**
    - **Attributes:** Doctor-ID (Primary Key), first-name, last-name, email, phone_nb, title, specialty, starting_Year, F_ID (Foreign Key).
    - **Description:** Represents doctors working in medical facilities. Each doctor is associated with one medical facility.
3. **Patients**
    - **Attributes:** P_SSN (Primary Key), P_ID, DOB, first-name, last-name, email, address, gender, phone_nb, medical_history.
    - **Description:** Represents patients with unique social security numbers.
4. **InsurancePlan**
    - **Attributes:** IP_ID (Primary Key), class, company_provider, issuing_date, end_date, P_SSN (Foreign Key).
    - **Description:** Represents insurance plans associated with patients.
5. **EmergencyContacts**
    - **Attributes:** phone_nb (Primary Key), name, relationship, Patient_SSN (Foreign Key).
    - **Description:** Represents emergency contacts for patients.
6. **Medical_File**
    - **Attributes:** MF_ID (Primary Key), P_SSN (Foreign Key), Doctor-ID (Foreign Key), prescription, desc, date.

- **Description:** Represents medical files, which are associated with patients and doctors.

7. **LabTest**
    - **Attributes:** LT_ID (Primary Key), report, name, date, reason, MF_ID (Foreign Key).
    - **Description:** Represents lab tests associated with medical files.
8. **Radiology**
    - **Attributes:** R_ID (Primary Key), name, date, report, reason, MF_ID (Foreign Key).
    - **Description:** Represents radiology reports associated with medical files.
9. **Surgery**
    - **Attributes:** S_ID (Primary Key), Surgery_name, aim.
    - **Description:** Represents different types of surgeries.
10. **Treatments**
    - **Attributes:** T_ID (Primary Key), name, reason, startDate, endDate, MF_ID (Foreign Key).
    - **Description:** Represents treatments associated with medical files.
11. **Appointment**
    - **Attributes:** A_ID (Primary Key), day, to, from, P_SSN (Foreign Key), reason.
    - **Description:** Represents appointments for patients.

## Relationships

1. **Perform_Surgery**

- **Multiplicity:** Many-to-Many (A doctor can perform many surgeries on many patients, and a patient can have many surgeries performed by many doctors).
- **Justification:** A patient may have multiple surgeries, each potentially performed by different doctors. Similarly, doctors can perform surgeries on multiple patients.

2. **hasAvailability**

- **Multiplicity:** One-to-Many (A doctor can have multiple availability slots, but each slot is associated with one doctor).
- **Justification:** Each doctor can have multiple time slots available for appointments, but each time slot is specific to one doctor.

3. **Doctor to MedicalFacility**

- **Multiplicity:** Many-to-One (Many doctors can work at one medical facility).
- **Justification:** Each doctor is associated with a single medical facility, but a medical facility can have multiple doctors.

4. **Patients to InsurancePlan**

- **Multiplicity:** One-to-Many (One patient can have multiple insurance plans over time).
- **Justification:** Patients may have different insurance plans at different times, but each insurance plan is specific to one patient.

5. **Patients to EmergencyContacts**

- **Multiplicity:** One-to-Many (One patient can have multiple emergency contacts).
- **Justification:** A patient can have multiple people listed as emergency contacts.

**6. Medical_File to Patients**

- **Multiplicity:** Many-to-One (Many medical files can belong to one patient).
- **Justification:** Each patient can have multiple medical files over time, but each medical file is associated with a single patient.

**7. Medical_File to Doctor**

- **Multiplicity:** Many-to-One (Many medical files can be created by one doctor).
- **Justification:** Each doctor can create multiple medical files for different patients, but each medical file is associated with one doctor.

**8. LabTest, Radiology, Treatments to Medical_File**

- **Multiplicity:** Many-to-One (Many lab tests, radiology reports, and treatments can be associated with one medical file).
- **Justification:** Each medical file can have multiple lab tests, radiology reports, and treatments associated with it.

**9. Appointment to Patients**

- **Multiplicity:** Many-to-One (Many appointments can be made for one patient).
- **Justification:** Each patient can have multiple appointments.

# Relational Model

MedicalFacility (F_ID, Fname, location)

Doctor (Doctor-ID, first-name, last-name, email, phone_nb, title, specialty, starting_Year #F_ID)

Patients (P_SSN, P_ID, DOB, first-name, last-name, email, address, gender, email, phone_nb, medical_history)

InsurancePlan (IP_ID, class, company_provider, issuing_date, end_date, #P_SSN)

EmergencyContacts (phone_nb, name, relationship, #Patient_SSN)

Medical_File (MF_ID,#P_SSN, #Doctor-ID, prescription)

LabTest (LT_ID, report, name, date, reason, #MF_ID)

Radiology (R_ID, name, date, report, reason, #MF_ID)

Surgery (S_ID, Surgery_name, aim)

Treatments (T_ID, name, reason, startDate, endDate, #MF_ID)

Appointment (A_ID, day, to, from, #P_SSN, reason)

Perform_Surgery (#Doctor_ID, #P_SSN, #Surgery_ID,successful?, Date)

hasAvailability (#Doctor_ID, #A_ID)

# Creating tables and data insertion

## Queries to create the tables

```
CREATE TABLE MEDICAL_FACILITY(
    Medical_Facility_ID INTEGER AUTO_INCREMENT UNIQUE,
    Facility_Name VARCHAR (30),
    Facility_Location VARCHAR(70),
    PRIMARY KEY (Medical_Facility_ID)
    );

CREATE TABLE DOCTOR(
    Doctor_ID INTEGER AUTO_INCREMENT,
    First_Name VARCHAR(30),
    Last_Name VARCHAR(30),
    email VARCHAR(30)  UNIQUE,
    Phone_Number VARCHAR(13) UNIQUE  CHECK (Phone_Number LIKE '+961________'),
    Title VARCHAR(10),
    Specialty VARCHAR(50),
    Starting_Year INTEGER,
    Medical_Facility_ID INTEGER,
    PRIMARY KEY (Doctor_ID),
    CONSTRAINT FID FOREIGN KEY (Medical_Facility_ID) REFERENCES
MEDICAL_FACILTIY(Medical_Facility_ID)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);

CREATE TABLE PATIENTS(
    Patient_SSN INTEGER,
    Patient_ID INTEGER UNIQUE AUTO_INCREMENT,
    Date_Of_Birth DATE,
    First_Name VARCHAR(30),
    Last_Name VARCHAR(30),
    email VARCHAR(30)  UNIQUE,
    Phone_Number VARCHAR(13) UNIQUE CHECK (Phone_Number LIKE '+961________'),
    Address VARCHAR(80),
    Gender CHAR(1),
    Medical_History VARCHAR (50000),
    PRIMARY KEY (Patient_SSN)
);

CREATE TABLE INSURANCE_PLAN(
    Insurance_Plan_ID INTEGER AUTO_INCREMENT,
```

```sql
    Company_Provider VARCHAR(30),
    Class VARCHAR(3),
    Issuing_Date DATE,
    End_Date DATE,
    Patient_SSN INTEGER  UNIQUE,
    PRIMARY KEY (Insurance_Plan_ID),
    CONSTRAINT PID FOREIGN KEY (Patient_SSN) REFERENCES PATIENTS(Patient_SSN)
    ON DELETE CASCADE
);

CREATE TABLE EMERGENCY_CONTACTS(
    Phone_Number VARCHAR (13) UNIQUE CHECK (Phone_Number LIKE '+961_______'),
    Name VARCHAR(30),
    Relationship VARCHAR(12),
    Patient_SSN INTEGER,
    PRIMARY KEY (Phone_Number),
    CONSTRAINT SSN FOREIGN KEY (Patient_SSN) REFERENCES PATIENTS(Patient_SSN)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE MEDICAL_FILE(
    Medical_File_ID INTEGER AUTO_INCREMENT,
    Patient_SSN INTEGER UNIQUE,
    Doctor_ID INTEGER,
    Prescription VARCHAR(500),
    PRIMARY KEY (Medical_File_ID, Patient_SSN),
    CONSTRAINT SSN FOREIGN KEY (Patient_SSN) REFERENCES PATIENTS(Patient_SSN)
    ON DELETE SET NULL,
    CONSTRAINT DID FOREIGN KEY (Doctor_ID) REFERENCES DOCTOR(Doctor_ID)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);


CREATE TABLE LAB_TEST(
    Test_ID INTEGER AUTO_INCREMENT,
    Test_Name VARCHAR(30),
    Date DATE,
    Report VARCHAR(1000),
    Reason VARCHAR(500),
    Medical_File_ID INTEGER,
    PRIMARY KEY (Test_ID),
    CONSTRAINT MFID FOREIGN KEY (Medical_File_ID) REFERENCES
MEDICAL_FILE(Medical_File_ID)
```

```sql
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE RADIOLOGY(
    Radiology_ID INTEGER AUTO_INCREMENT,
    Radiology_Name VARCHAR(30),
    Date DATE,
    Report VARCHAR(1000),
    Reason VARCHAR(500),
    Medical_File_ID INTEGER,
    PRIMARY KEY (Radiology_ID),
    CONSTRAINT MFID FOREIGN KEY (Medical_File_ID) REFERENCES
MEDICAL_FILE(Medical_File_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE SURGERY(
    Surgery_ID INTEGER AUTO_INCREMENT,
    Surgery_Name VARCHAR(30),
    Aim VARCHAR(100),
    PRIMARY KEY (Surgery_ID)
);

CREATE TABLE PERFORM_SURGERY(
    Doctor_ID INTEGER,
    Patient_SSN INTEGER,
    Surgery_ID INTEGER AUTO_INCREMENT,
    Successful BOOLEAN,
    Date DATE,
    PRIMARY KEY (Doctor_ID, Patient_SSN, Surgery_ID),
    CONSTRAINT DID FOREIGN KEY (Doctor_ID) REFERENCES DOCTOR(Doctor_ID)
    ON DELETE SET NULL
    ON UPDATE CASCADE,
    CONSTRAINT SSN FOREIGN KEY (Patient_SSN) REFERENCES PATIENTS(Patient_SSN)
    ON DELETE CASCADE,
    CONSTRAINT SID FOREIGN KEY (Surgery_ID) REFERENCES SURGERY(Surgery_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE TREATMENT(
    Treatment_ID INTEGER AUTO_INCREMENT,
    Treatment_Name VARCHAR(30),
```

```sql
    Reason VARCHAR(100),
    Start_Date DATE,
    END_DATE DATE,
    Medical_File_ID INTEGER,
    PRIMARY KEY (Treatment_ID),
    CONSTRAINT MFID FOREIGN KEY (Medical_File_ID) REFERENCES
MEDICAL_FILE(Medical_File_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE APPOINTMENT(
    Appointment_ID INTEGER AUTO_INCREMENT,
    Day DATE,
    Start_Time TIME,
    End_Time TIME,
    Reason VARCHAR(100),
    Patient_SSN INTEGER,
    PRIMARY KEY (Appointment_ID),
    CONSTRAINT SSN FOREIGN KEY (Patient_SSN) REFERENCES PATIENTS(Patient_SSN)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);

CREATE TABLE IS_AVAILABLE(
    Doctor_ID INTEGER,
    Appointment_ID INTEGER,
    PRIMARY KEY (Doctor_ID, Appointment_ID),
    CONSTRAINT DID FOREIGN KEY (Doctor_ID) REFERENCES DOCTOR(Doctor_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    CONSTRAINT AID FOREIGN KEY (Appointment_ID) REFERENCES
APPOINTMENT(Appointment_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

| | Table | Action | | | | | | | Rows | Type | Collation | Size | Overhead |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☐ | appointment | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | doctor | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | emergency_contacts | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | insurance_plan | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | is_available | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | lab_test | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | medical_facility | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | medical_file | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | patients | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | perform_surgery | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | radiology | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | surgery | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| ☐ | treatment | | Browse | Structure | Search | Insert | Empty | Drop | 0 | MyISAM | utf8mb4_0900_ai_ci | 1.0 KiB | - |
| | **13 tables** | **Sum** | | | | | | | **0** | **MyISAM** | **utf8mb4_0900_ai_ci** | **13.0 KiB** | **0 B** |

Check all With selected:

## Queries to create triggers:

```
DELIMITER $$

CREATE TRIGGER PhoneNumber
BEFORE INSERT ON emergency_contacts
FOR EACH ROW
BEGIN
  IF NEW.Phone_Number NOT LIKE '+961_______' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid phone number format';
  END IF;
END;
$$

CREATE TRIGGER PatientSSNOnInsurancePlan
BEFORE INSERT ON insurance_plan
FOR EACH ROW
BEGIN
  IF NEW.Patient_SSN NOT IN (SELECT Patient_SSN FROM patients) THEN
    DELETE FROM insurance_plan WHERE Patient_SSN = new.Patient_SSN;
  END IF;
END;
$$

CREATE TRIGGER PatientSSNOnMedicalFile
BEFORE INSERT ON medical_file
FOR EACH ROW
BEGIN
  IF NEW.Patient_SSN NOT IN (SELECT Patient_SSN FROM patients) THEN
    DELETE FROM medical_file WHERE Patient_SSN = new.Patient_SSN;
  END IF;
END;
$$

CREATE TRIGGER DoctorIDOnMedicalFile
BEFORE INSERT ON medical_file
FOR EACH ROW
BEGIN
  IF NEW.Doctor_ID NOT IN (SELECT Doctor_ID FROM doctor) THEN
    UPDATE new SET Doctor_ID = NULL;
  END IF;
END;
$$

CREATE TRIGGER NoSameName
```

```sql
BEFORE INSERT ON DOCTOR
FOR EACH ROW
BEGIN
  DECLARE count_names INT;
  SELECT COUNT(*) INTO count_names
  FROM DOCTOR
  WHERE First_Name = NEW.First_Name AND Last_Name = NEW.Last_Name;
  IF count_names > 0 THEN
    SET NEW.First_Name = CONCAT(NEW.First_Name, '2.0');
    SET NEW.Last_Name = CONCAT(NEW.Last_Name, '2.0');
  END IF;
END;
$$

CREATE TRIGGER NoSameEmail
BEFORE INSERT ON DOCTOR
FOR EACH ROW
BEGIN
  DECLARE count_emails INT;
  SELECT COUNT(*) INTO count_emails
  FROM DOCTOR
  WHERE email = NEW.email;
  IF count_emails > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Email already exists';
  END IF;
END;
$$

CREATE TRIGGER check_Starting_Year_before_insert
BEFORE INSERT ON doctor
FOR EACH ROW
BEGIN
    IF NEW.Starting_Year < '1955-01-01' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Doctor cannot be born before 1935';
    END IF;
END;
$$

CREATE TRIGGER check_Starting_Year_before_update
BEFORE UPDATE ON doctor
FOR EACH ROW
BEGIN
    IF NEW.Starting_Year < '1955-01-01' THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'We are sorry';
    END IF;
END;
$$

CREATE TRIGGER NoSameNamePatients
BEFORE INSERT ON PATIENTS
FOR EACH ROW
BEGIN
  DECLARE count_names INT;
  SELECT COUNT(*) INTO count_names
  FROM PATIENTS
  WHERE First_Name = NEW.First_Name AND Last_Name = NEW.Last_Name;
  IF count_names > 0 THEN
    SET NEW.First_Name = CONCAT(NEW.First_Name, '2.0');
    SET NEW.Last_Name = CONCAT(NEW.Last_Name, '2.0');
  END IF;
END;
$$

DELIMITER $$
```

| | Name | Table | Time | Event | | | |
|---|---|---|---|---|---|---|---|
| ☐ | **DoctorIDOnMedicalFile** | medical_file | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **NoSameEmail** | doctor | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **NoSameName** | doctor | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **NoSameNamePatients** | patients | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **PatientSSNOnInsurancePlan** | insurance_plan | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **PatientSSNOnMedicalFile** | medical_file | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **PhoneNumber** | emergency_contacts | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **check_Starting_Year_before_insert** | doctor | BEFORE | INSERT | Edit | Export | Drop |
| ☐ | **check_Starting_Year_before_update** | doctor | BEFORE | UPDATE | Edit | Export | Drop |

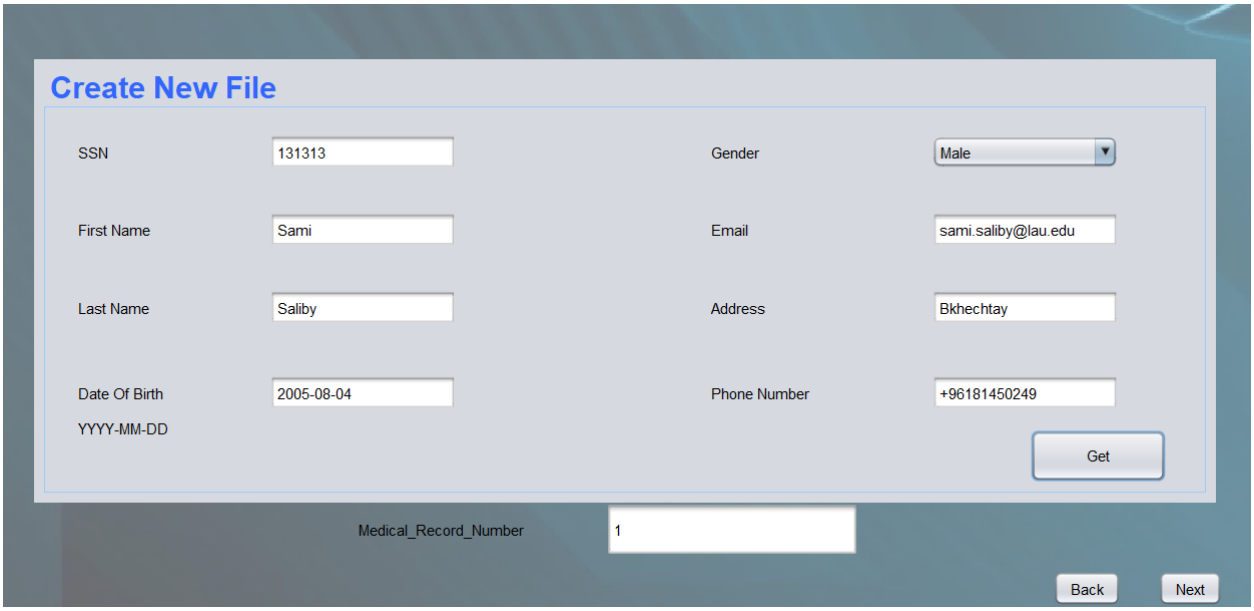
## Queries to Update the Database:

connect: Connects to the database

close: closes the connection to the database

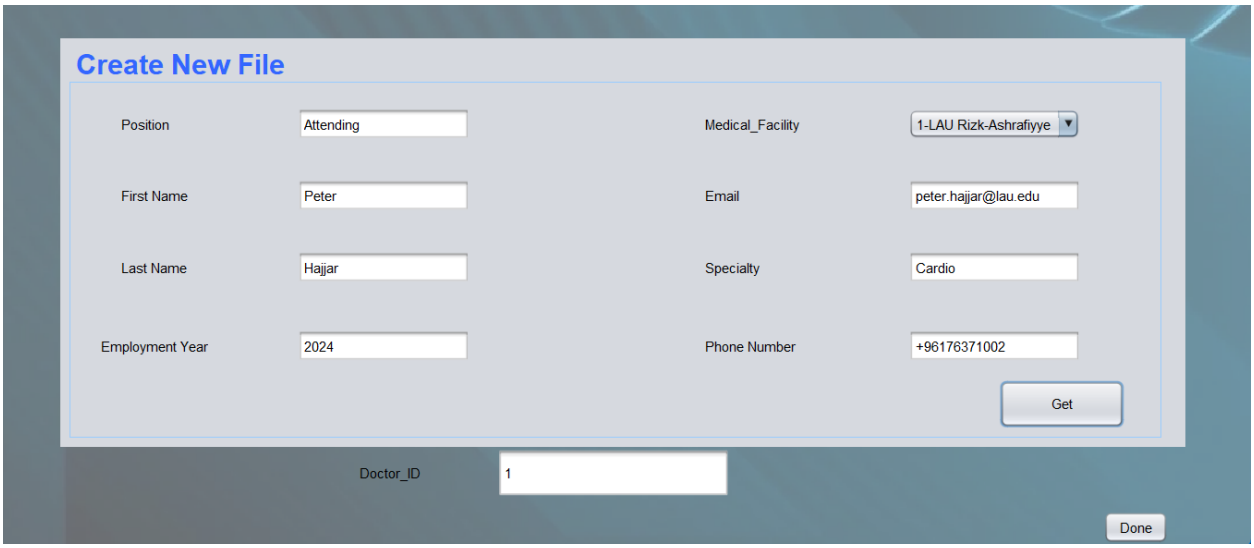
addNewPatient: Inserts a new patient record into the PATIENT table using the details from the Patient object.

```
String query = "INSERT INTO PATIENTS VALUES (" + p.getPatient_SSN() + ", " +
p.getPatient_ID()
        + ", '" + p.getDate_Of_Birth() + "', '" + p.getFirst_Name() + "', '" +
        p.getLast_Name() + "', '" + p.getEmail() +"', '" + p.getPhone_Number()
        + "', '" + p.getAddress() + "', '" + p.getGender() + "', '" +
        p.getMedical_History() + "')";
```




addNewDoctor: Inserts a new doctor record into the DOCTOR table using the details from the Doctor object.

```
String query = "INSERT INTO DOCTOR VALUES (" + d.getDoctorId() + ", '" +
d.getFirstName()
        + "', '" + d.getLastName() + "', '" + d.getEmail() + "', '" + d.getPhoneNumber()
        + "', '" + d.getTitle() + "', '" + d.getSpecialty() + "', " + d.getStartingYear()
        + ", " + d.getMedicalFacilityId()+ ")";
```




| | Medical_File_ID | Patient_SSN | Doctor_ID | Prescription |
|---|---|---|---|---|
| Edit Copy Delete | 1 | 131313 | NULL | NULL |

getDoctorsBySpecialty: Retrieves a list of doctor first names and last names based on a given specialty.

String query = "SELECT First_Name, Last_Name FROM DOCTOR WHERE specialty = '' + specialty;

getAllSpecialties: Retrieves a list of all unique specialties from the DOCTOR table.

SELECT Specialty FROM DOCTOR

String q = "SELECT Specialty FROM DOCTOR";

getApptInfoFromDoctor: Retrieves all appointment information for a doctor based on their first name and last name.

String q = "SELECT * FROM APPOINTMENT WHERE Appointment_ID IN (SELECT Appointment_ID FROM IS_AVAILABLE NATURAL JOIN DOCTOR d WHERE d.First_Name = '" + fn + "', AND Last_Name = '" + ln + "';";

WHERE First_Name = ? AND Last_Name = ?

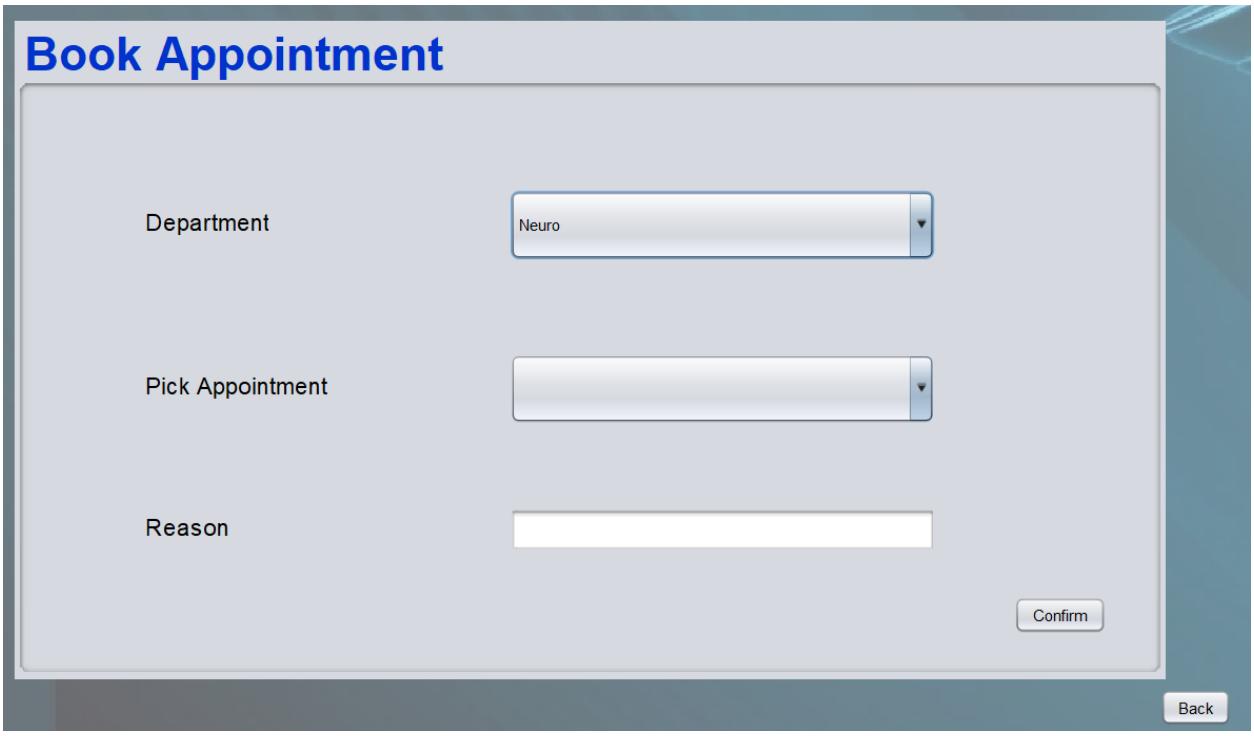
)

removeApptFromAvailability: Deletes an appointment from the IS_AVAILABLE table based on the appointment ID.

String q = "DELETE FROM IS_AVAILABLE WHERE Appointment_ID = '' + appointmentId + ";";

addAppttoBookAppt: Inserts a new record into the BOOK_APPOINTMENT table using the details from the Book_Appointment object.

```
String q = "INSERT INTO BOOK_APPOINTMENT VALUES (" + a.getPatient_SSN()
        + ", " + a.getAppointment_ID() + ", '" + a.getReason() + "';";
```

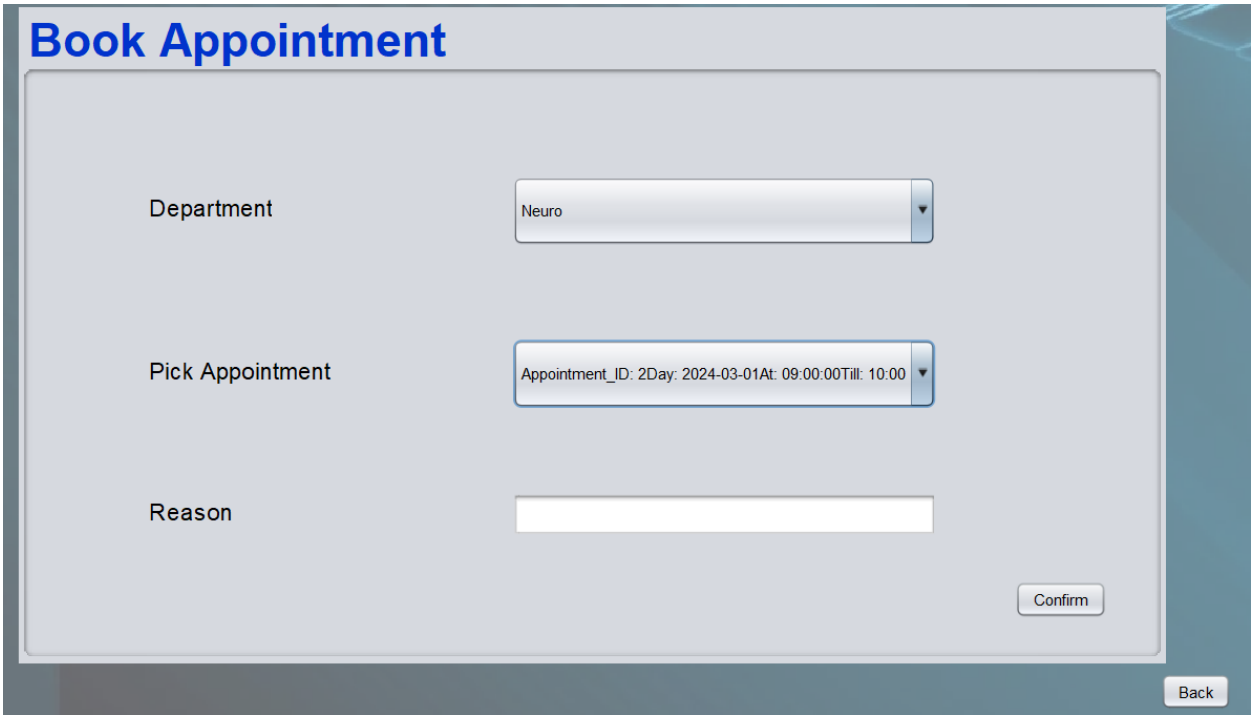



getApptFromSSN: Retrieves all appointments associated with a specific patient's SSN.

```
String q = "SELECT * FROM APPOINTMENT NATURAL JOIN BOOK_APPOINTMENT
WHERE Patient_SSN = " + ssn + ";";
```

removeApptFromBooked: Deletes a booked appointment from the BOOK_APPOINTMENT table based on the appointment ID.

```
String q = "DELETE FROM BOOK_APPOINTMENT WHERE Appointment_ID = " +
chosenAppointment.getAppointmentId() + ";";
```



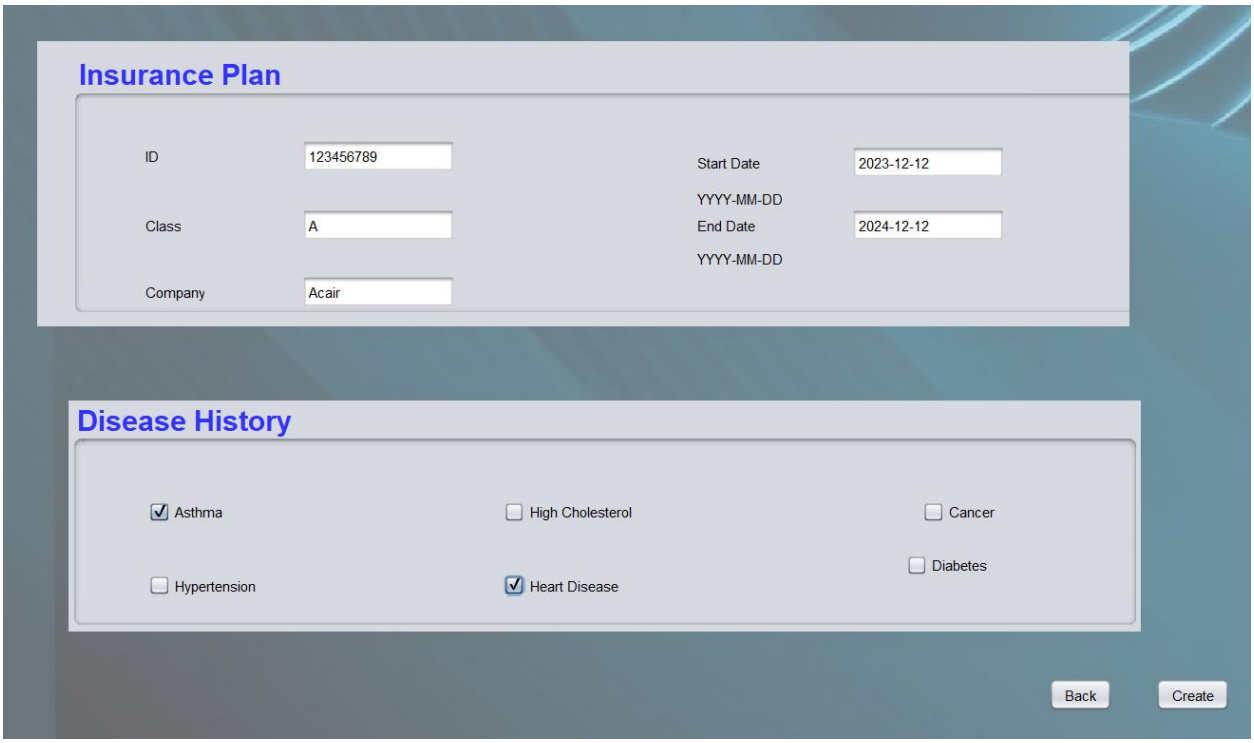

addAppttoAvailable: Inserts a new appointment into the IS_AVAILABLE table using the details from the Appointment object.


createNewInsurancePlan: Inserts a new insurance plan record into the INSURANCE_PLAN table using the details from the Insurance_Plan object.

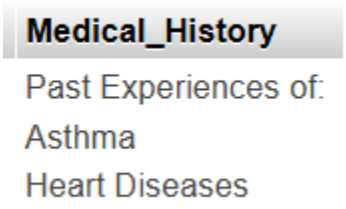
```
String q = "INSERT INTO INSURANCE_PLAN VALUES (" + ip.getInsurancePlanId()
        + ", '" + ip.getCompanyProvider() + "', '" + ip.getInsuranceClass()
        + "', '" + ip.getIssuingDate() + "', '" + ip.getEndDate()
        + "', " + ip.getPatientSSN() + ");";
```

updateMedicalHistory: Updates the medical history of a patient in the PATIENTS table based on the patient's ID (MRN).

```
String q = "UPDATE PATIENTS SET Medical_History = '" + MedHis
        + "' WHERE Patient_ID = " + MRN;
```




| | Insurance_Plan_ID | Company_Provider | Class | Issuing_Date | End_Date | Patient_SSN |
|---|---|---|---|---|---|---|
| Edit Copy Delete | 123456789 | Acair | A | 2023-12-12 | 2024-12-12 | 131313 |




**<u>Querying the Database</u>:**

exists: Checks if a patient with a given SSN exists in the PATIENTS table.

SELECT * FROM PATIENTS WHERE Patient_SSN =  SSN + “;”;

getSSNFromMRN: Retrieves the SSN of a patient based on their medical record number (MRN). Executes an SQL SELECT query and returns the SSN.

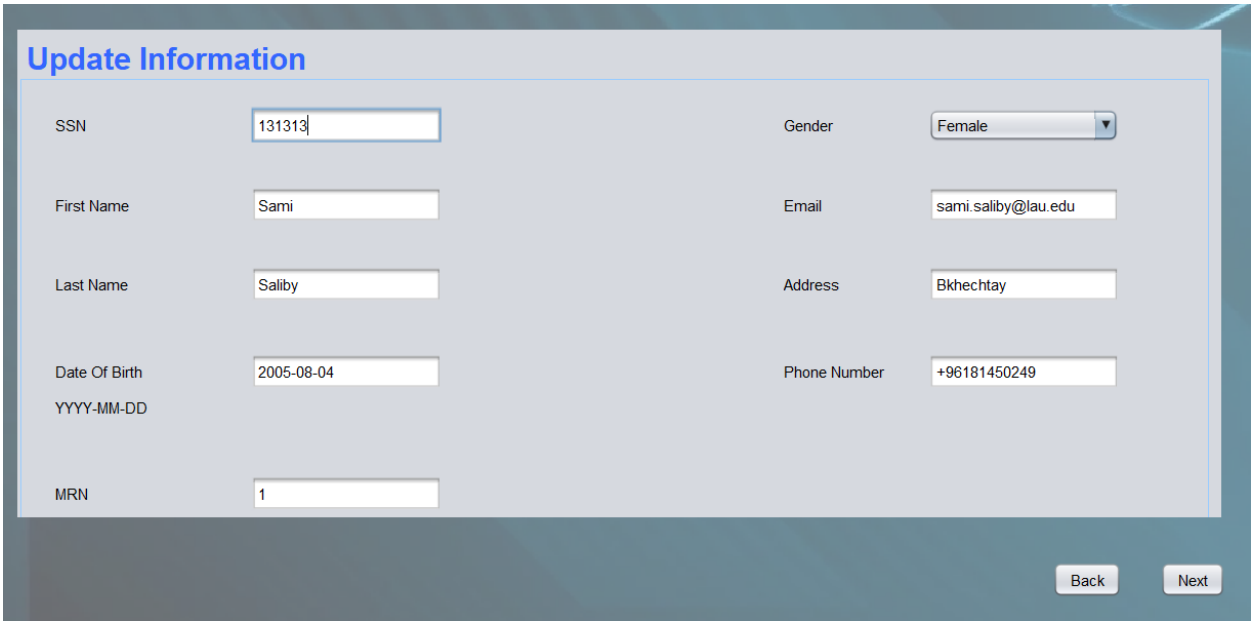
```
String q = "SELECT Patient_SSN FROM PATIENTS WHERE Patient_ID = " + MRNs + ";";

    int SSNN = 0;
```

getPatientFromMRN: Retrieves all information of a patient based on their medical record number (MRN).

```
String q = "SELECT * FROM PATIENTS WHERE Patient_ID = " + MRN;
```

updatePatientBySSN: Updates the details of a patient in the PATIENTS table based on their SSN.

```
String q = "UPDATE PATIENTS SET Patient_SSN = " + p.getPatient_SSN()
        + ", Patient_ID = " + p.getPatient_ID() + ", Date_Of_Birth = '" + p.getDate_Of_Birth()
        + "', First_Name = '" + p.getFirst_Name() + "', Last_Name = '" +
        p.getLast_Name() + "', email = '" + p.getEmail() + "', Phone_Number = '" +
        p.getPhone_Number() + "', Address = '" + p.getAddress() + "', Gender = '"
        + p.getGender() + "' "
        + "WHERE Patient_SSN = " + p.getPatient_SSN() + ";";
```



getInsuranceInfoFromMRN: This method retrieves insurance plan information for a patient identified by their MRN (Medical Record Number).

```
String q = "SELECT * FROM INSURANCE_PLAN WHERE Patient_SSN = "
        + "(SELECT PATIENT_SSN FROM PATIENTS WHERE PATIENT_ID = " + mrn
        + ");";
```

UpdateInsurancePlan : This method updates an existing insurance plan in the INSURANCE_PLAN table.

```
String q = "UPDATE INSURANCE_PLAN SET Insurance_Plan_ID = " + ip.getInsurancePlanId()
        + ", Company_Provider = '" + ip.getCompanyProvider()
        + "', Class = '" + ip.getInsuranceClass() + "', Issuing_Date = '"
        + ip.getIssuingDate() + "', End_Date = '" + ip.getEndDate() + "', Patient_SSN = "
        + ip.getPatientSSN()
        + " WHERE Insurance_Plan_ID = " + ip.getInsurancePlanId() + ";";
```

getPerformFromSurgery: This method retrieves the performance details of a surgery.

```
String q = "SELECT * FROM PERFORM_SURGERY WHERE Surgery_ID = "
        + "(SELECT SURGERY_ID FROM SURGERY WHERE SURGERY_ID = "
        + s.getSurgeryId();
```

retrieveSurgeriesbyMRN: This method retrieves all surgeries associated with a patient identified by their MRN.

```
String q = "SELECT * FROM SURGERY NATURAL JOIN PERFORM_SURGERY NATUAL JOIN PATIENTS WHERE "
        + "Patient_ID = " + mrn + ";";
```

getMRNFromSSN: This method retrieves the MRN (Medical Record Number) of a patient based on their SSN (Social Security Number).

```
String q = "SELECT Patient_ID FROM PATIENTS WHERE Patient_SSN = " + p.getPatient_SSN() + ";";
```

retrieveLabbyMRN: This method retrieves all lab tests associated with a patient identified by their MRN.

```
String q = "SELECT * FROM LAB_TEST NATURAL JOIN MEDICAL_FILE NATURAL JOIN PATIENTS "
            + "WHERE Patient_ID = " + mrnOfPatient;
```

retrieveRadiologybyMRN: This method retrieves all radiology reports associated with a patient identified by their MRN.

```
String q = "SELECT * FROM RADIOLOGY NATURAL JOIN MEDICAL_FILE  WHERE MEDICAL_FILE_ID = " + mrnOfPatient + ";";
```

retrieveTreatmentsbyMRN: This method retrieves all treatments associated with a patient identified by their MRN.

```
String q = "SELECT * FROM TREATMENT NATURAL JOIN MEDICAL_FILE NATURAL JOIN PATIENTS WHERE Patient_ID = " + mrnOfPatient + ";";
```

getMedicalFileIdFromSSN: This method retrieves the medical file ID associated with a patient's SSN.

```
String q = "SELECT Prescription FROM MEDICAL_FILE NATURAL JOIN PATIENTS WHERE "
            + "Patient_ID = " + mrnOfPatient + ";";
```

retrievePrescriptionFromMedicalFile: This method retrieves the prescription information associated with a patient's MRN (Medical Record Number).

```
String q = "SELECT Prescription FROM MEDICAL_FILE NATURAL JOIN PATIENTS WHERE "
            + "Patient_ID = " + mrnOfPatient + ";";
```

createTheirMedicalFile: This method creates a new medical file for a patient by inserting a new record into the MEDICAL_FILE table.

```
String q = "INSERT INTO MEDICAL_FILE (Medical_File_ID, Date_Of_Creation, Patient_SSN) " +
              "VALUES (" + p.getPatient_ID() + ", '2024-07-22', '" + p.getPatient_SSN() + "');";
```

createApptInAppointment: This method creates a new appointment by inserting a record into the APPOINTMENT table.

```
String q = "INSERT INTO APPOINTMENT (Day, Start_Time, End_Time) VALUES ('"
            + a.getDay() + "', '" + a.getStartTime() + "', '" + a.getEndTime() + "')";
```

getDocIDFromApptID: This method retrieves the doctor ID associated with a specific appointment ID.

String q = "SELECT Doctor_ID FROM IS_AVAILABLE WHERE Appointment_ID = " + appointmentId;

addAppttoAvailable: This method adds an appointment to the IS_AVAILABLE table, associating it with a specific doctor ID.

```
String q = "INSERT INTO IS_AVAILABLE (Appointment_ID, Doctor_ID) VALUES ("
            + chosen.getAppointmentId() + ", " + doc_ID + ")";
```

getDocIDFromDoc: This method retrieves the doctor ID based on the doctor's first and last name.

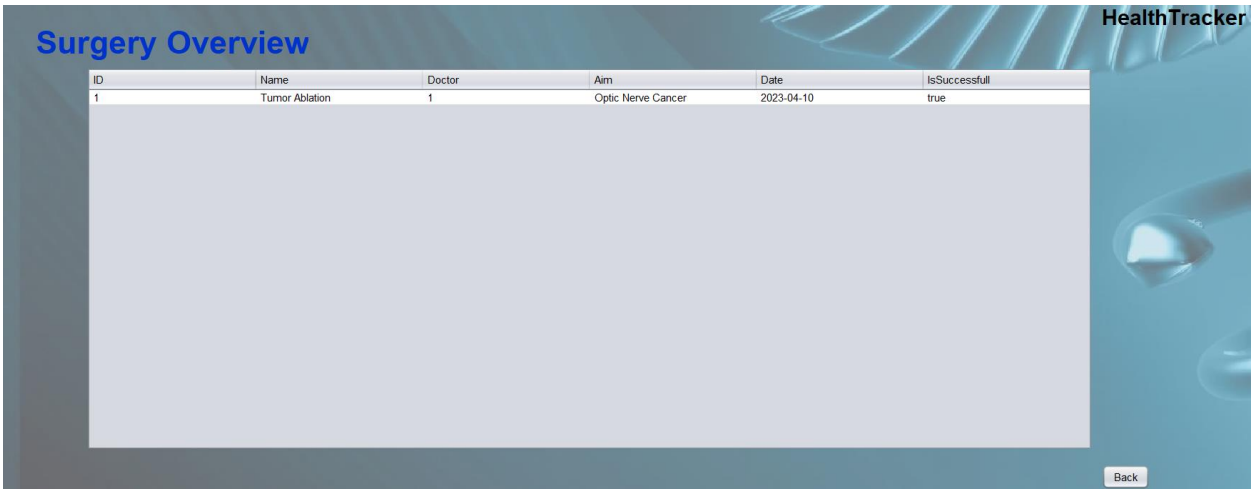
```
String q = "SELECT Doctor_ID FROM DOCTOR WHERE First_Name = '" + d.getFirstName()
            + "' AND Last_Name = '" + d.getLastName() + "'";
```

getAllMedicalFacilities: This method retrieves all medical facilities from the MEDICAL_FACILITY table

String q = "SELECT * FROM MEDICAL_FACILITY";

getSurgeriesFromMRN: This method retrieves all surgeries associated with a patient's MRN (Medical Record Number).

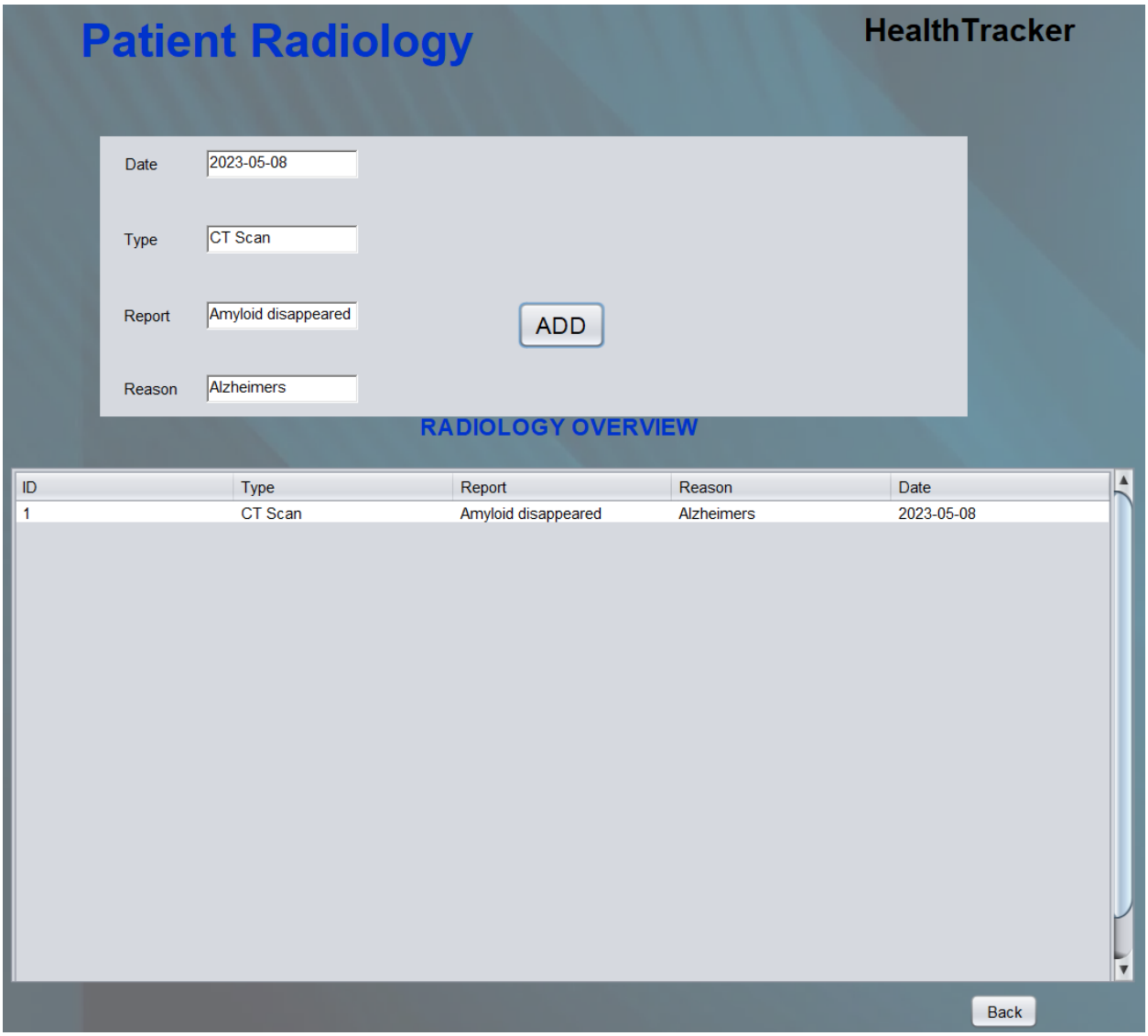
String q = "SELECT * FROM SURGERY NATURAL JOIN PERFORM_SURGERY WHERE Patient_SSN = (SELECT Patient_SSN FROM PATIENTS WHERE Patient_ID = " + mrnOfPatient + ");";




createNewRadiologyOnMedicalFile: This method creates a new radiology record for a patient by inserting a record into the RADIOLOGY table.

String query = "INSERT INTO RADIOLOGY (Radiology_Name, Date, Report, Reason, Medical_File_ID) VALUES (?, ?, ?, ?, ?)";

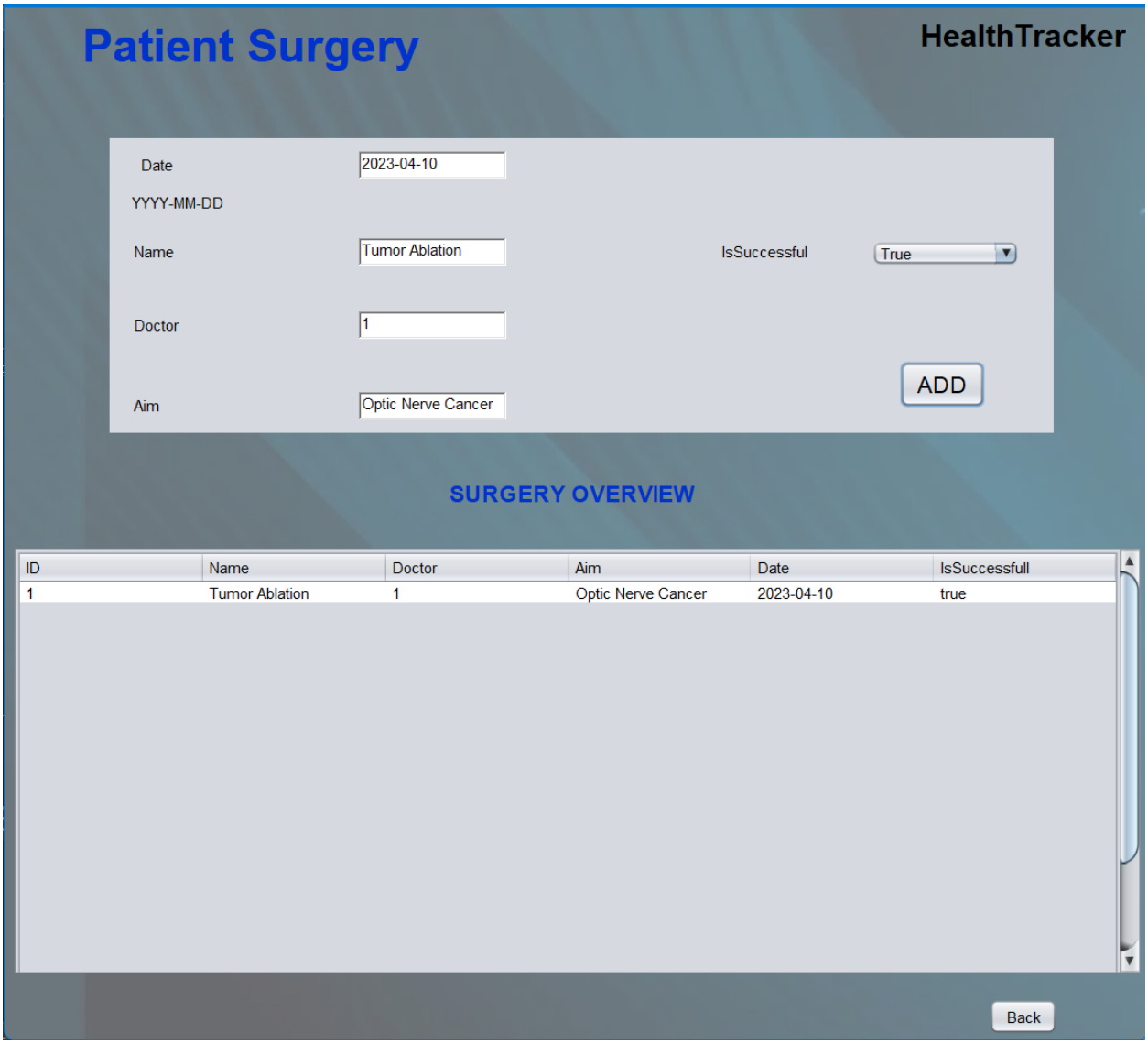

Search patients...
Enter patient MRN:
10
Search


Patient Radiology
HealthTracker
Date
2023-05-08
Type
CT Scan
Report
Amyloid disappeared
Reason
Alzheimers
ADD
RADIOLOGY OVERVIEW
ID
Type
Report
Reason
Date
1
CT Scan
Amyloid disappeared
Alzheimers
2023-05-08
Back

treatme

addNewSurgery: This method adds a new surgery record for a patient by inserting a record into the PERFORM_SURGERY table.

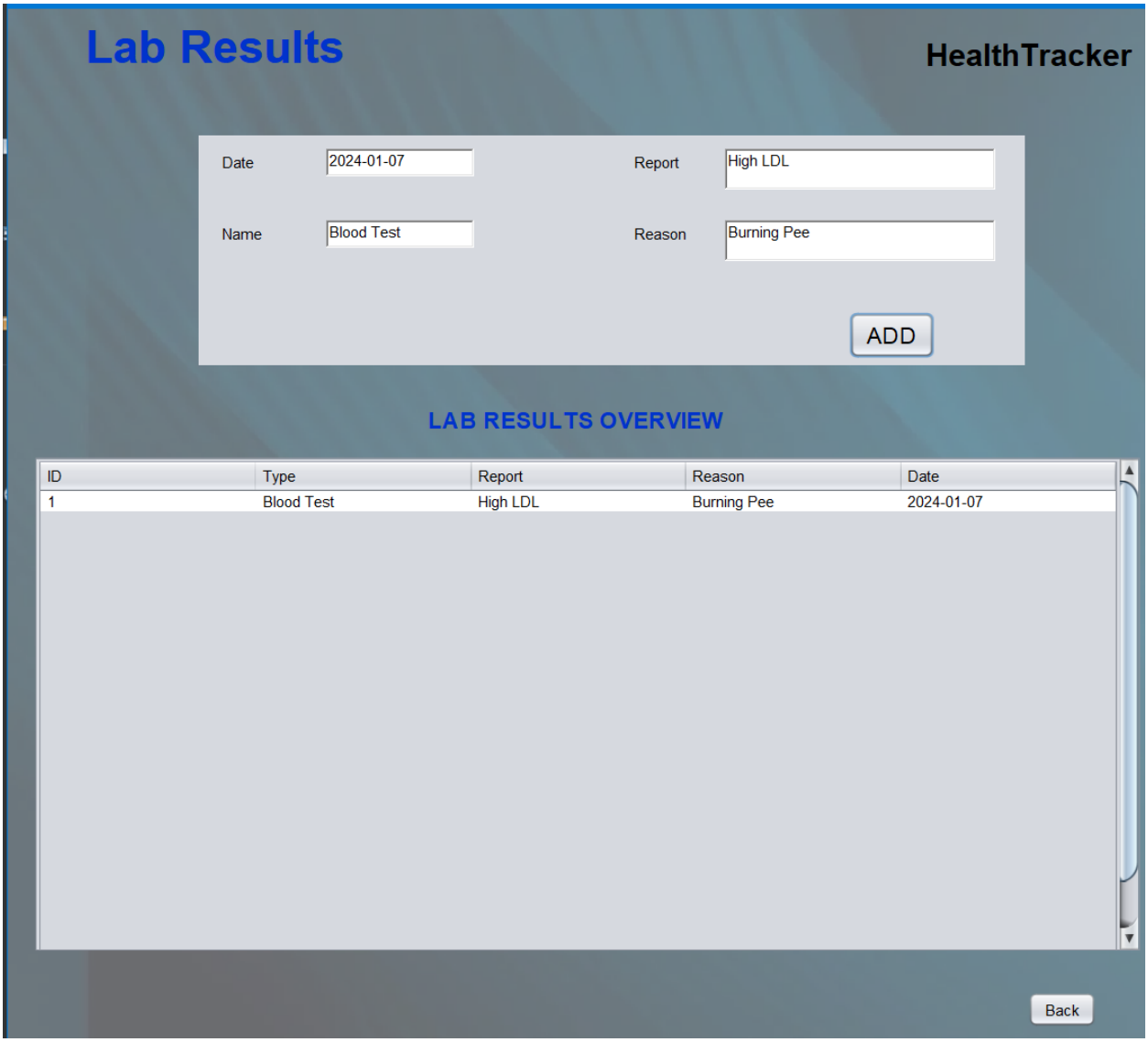
```
String q = "INSERT INTO PERFORM_SURGERY(Doctor_ID, Patient_SSN, Successful, Date) VALUES ("
        + s.getDoctor_ID() + ", '"
        + getSSNFromMRN(patientMRN) + "', "
        + s.isSuccessful() + ", '"
        + s.getDate() + "')";


    String q1 = "INSERT INTO SURGERY VALUES ("
        + "(SELECT MAX(Surgery_ID) FROM PERFORM_SURGERY), '"
        + s.getSurgery_Name() + "', '"
        + s.getAim() + "')";
```




addNewTreatment: This method adds a new treatment record for a patient by inserting a record into the TREATMENT table.

String query = "INSERT INTO TREATMENT (Treatment_Name, Reason, Start_Date, End_Date, Medical_File_ID) VALUES (?, ?, ?, ?, ?)";


addNewLabTest: This method adds a new lab test record for a patient by inserting a record into the LAB_TEST table.



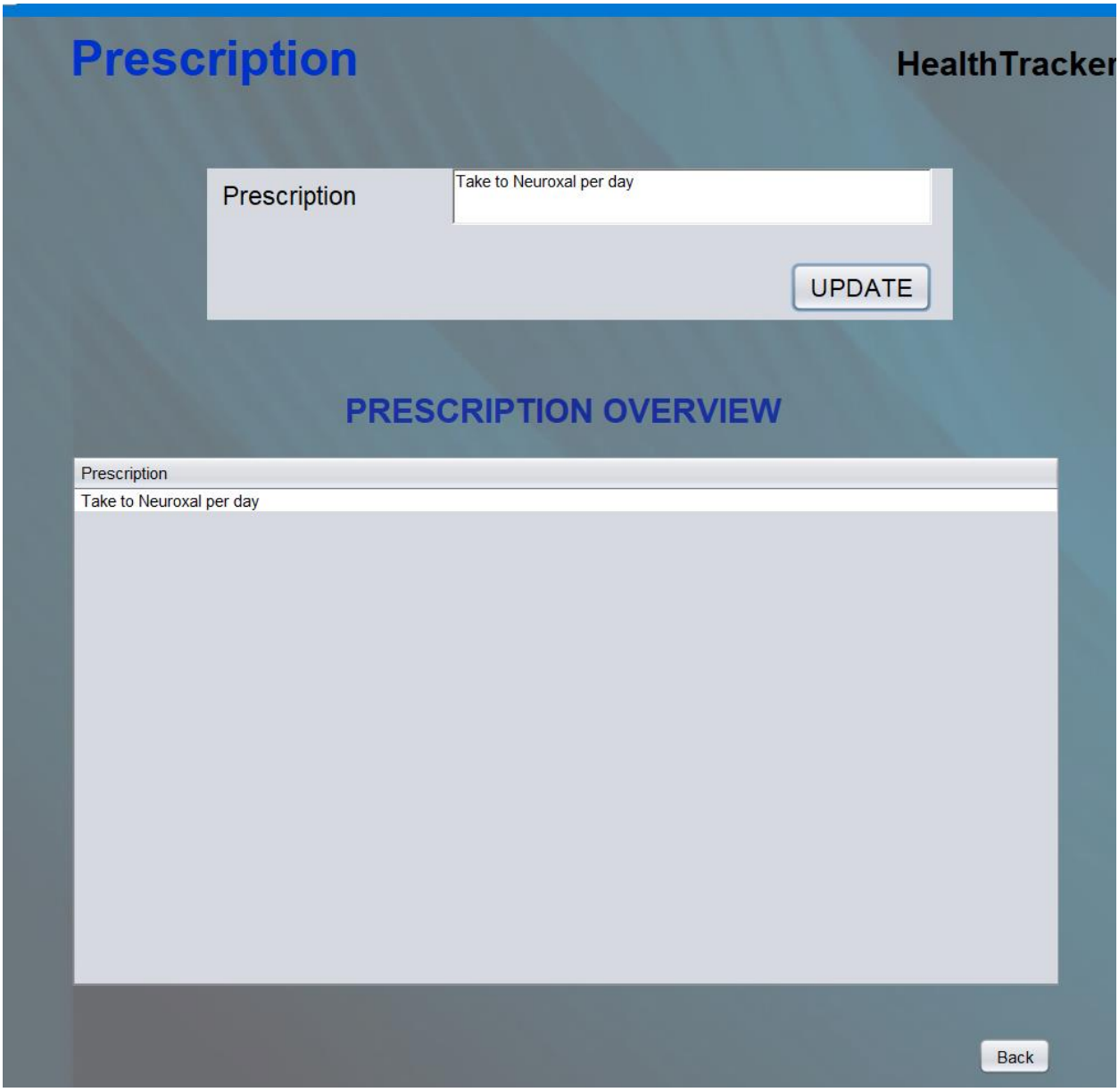

String query = "INSERT INTO LAB_TEST (Test_Name, Date, Report, Reason, Medical_File_ID) VALUES (?, ?, ?, ?, ?)";

getLabTestsOfPatientsByMRN: This method retrieves all lab tests associated with a patient's MRN (Medical Record Number).

String query = "SELECT * FROM LAB_TEST WHERE Medical_File_ID = ?";

updateMedicalFileWithNewPrescriptionUsingMRN: This method updates a patient's medical file with a new prescription based on the provided medical record number (MRN).

String q = "UPDATE MEDICAL_FILE SET Prescription = '" + prescription + "' WHERE Medical_File_ID = " + patientMRN;




addEmergencyContactToMRN: This method adds a new emergency contact for a patient identified by their medical record number (MRN).

```
String q = "INSERT INTO EMERGENCY_CONTACT VALUES ('"
        + ec.getPhoneNumber() + "', '"
        + ec.getName() + "', '"
        + ec.getRelationship() + ", "
        + ec.getPatientSSN() + ");";
```

getApptsBySpecialty: This method retrieves all available appointments for a specific medical specialty.

```
String q = "SELECT * FROM APPOINTMENT a NATURAL JOIN IS_AVAILABLE NATURAL JOIN DOCTOR"
            + "WHERE Specialty = '" + chosenSpecialty + "' AND a.Patient_SSN IS NULL;";
```

getApptFromSSN: This method retrieves all appointments for a patient identified by their Social Security Number (SSN).

```
String q = "SELECT * FROM APPOINTMENT WHERE Patient_SSN = " + SSN + ";";
```

updateAppointment: This method updates an appointment to set the Patient_SSN field to NULL, marking the appointment as unbooked.

```
String q = "UPDATE APPOINTMENT SET Patient_SSN = NULL WHERE Appointment_ID = " + a.getAppointmentId() + ";";
```

getBookedAppointments: This method retrieves all booked appointments for a specific doctor identified by their doctor ID.

```
String q = "SELECT * FROM APPOINTMENT NATURAL JOIN IS_AVAILABLE WHERE"
            + "Doctor_ID = " + DocID + " AND Patient_SSN IS NOT NULL;";
```

getPatientFromAppt: This method retrieves the patient information for a given appointment

```
String q = "SELECT * FROM PATIENT WHERE Patient_SSN = (SELECT"
            + "Patient_SSN FROM APPOINTMENT WHERE Appointment_ID = " + ap.getAppointmentId() + ";";
    Patient p = null;
```

NewApptID: This method retrieves the maximum appointment ID from the APPOINTMENT table to determine the most recently created appointment's ID.

```
String q = "SELECT MAX(Appointment_ID) AS Appointment_ID FROM APPOINTMENT";
```

getSurgeriesFromSSN: This method retrieves all surgeries performed on a patient based on their Social Security Number (SSN).

```
String q = "SELECT * FROM SURGERY NATURAL JOIN PERFORM_SURGERY WHERE Patient_SSN = " + SSnOfPatient + ";";
```

EnsureEmail: This method checks if the provided email address matches the email address associated with a specific doctor in the database.

```
String q = "SELECT email FROM DOCTOR WHERE Doctor_ID = " + iDofDoctor + ";";
```

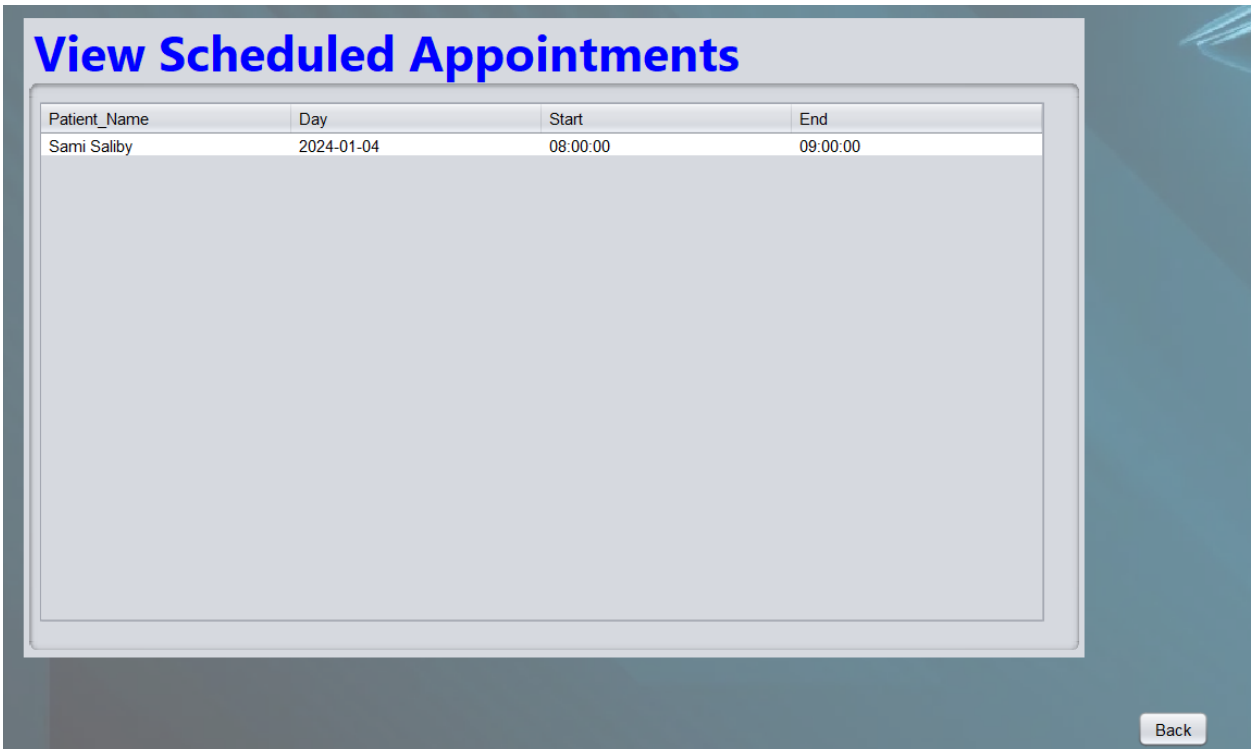
EnsureEmailofPatient:

This method checks if the provided email address matches the email address associated with a specific patient in the database.

String q = "SELECT email FROM PATIENTS WHERE Patient_ID = " + parseInt + ";";

bookAppt: This method updates the details of an appointment in the database, setting the patient assigned to the appointment and the reason for the appointment.

String q = "UPDATE APPOINTMENT SET Patient_SSN = " + a.getPatient_SSN() + ", REASON = '" + a.getReason() + "' WHERE  Appointment_ID = " + a.getAppointmentId();




getDoctors: This method retrieves all doctors from the database.

String query = "SELECT * FROM DOCTOR;";

addDRtoMedicalFile: This method assigns a specified doctor to a patient's medical file by updating the Doctor_ID field in the MEDICAL_FILE table.

String q = "UPDATE MEDICAL_FILE SET Doctor_ID = " + chosenDR

+ " WHERE Medical_File_ID = " + mrnOfPatient + ";";

# Layout

- AddingANewDoctor:  we can add a new doctor.
- BookAppointment: patients can book appointment.
- DoctorLabResults: Doctor can view and edit a patient's lab results.
- DoctorPrescrip: Doctor can view and edit a patient's prescription.
- DoctorRadiology: Doctor can view and edit a patient's radiology.
- DoctorSurgery: Doctor can view and edit a patient's surgery.
- DoctorTreatment: Doctor can view and edit a patient's Treatment.
- DoctorSchedule: Doctor can view booked appointments.
- Emergency: Patient can add an emergency contact.
- NewFile: Patient can create a new medical file by filling personal information.
- NewFile2: Patient can fill his insurance information and previous diseases.
- UpdateEmergency: Patient can update his emergency contact.
- UpdateFile: Patient can update his medical file including his personal information.
- UpdateInsurance: Patient can update his insurance information or common diseases that he suffers from.
- ViewAppointment: patient to view and cancel appointments
- availability: doctor can open availability.
- doctor_homepage: doctor he can view his schedule, search for patients, and manage appointments.
- doctorlogin: doctor can log in.
- doctorpatient:doctor  can view the patient medical file
- labresults:Patient can view his lab results.
- main_page:Patients can sign in or sign up, and doctors can sign up and sign in.
- Patient_homepage:Patient can view surgeries, treatments, prescriptions, lab test results, and radiology
- Patientpage:Patient can sign in.
- Prescription: allow patient to view his prescription
- Radio_page: patient can view his radiology.
- Surgeries_page: patient can view his surgeries.
- Treatment_page: patient can view his treatment