**COVER PAGE**

**Name**: Kambam Ajithpavanreddy

**Student** Id:1249160  
**Title**: Clinic Appointment Management System  
**Abstract**

**The Clinic Appointment Management System is designed to streamline clinic operations by enabling patients, doctors, and admins to manage appointments, medical records, and doctor schedules. The system is secure, role-based, and provides an efficient way to manage appointments and records, eliminating the need for complex SQL queries.**

**System Architecture**

* **Backend Architecture: The backend is implemented using Flask, which handles all HTTP requests, user authentication, and communication with the MySQL database.**
* **Frontend Architecture: The frontend uses HTML, CSS, and JavaScript for dynamic interactions and presentation.**
* **Database Architecture: MySQL stores all the necessary data, including users, appointments, doctors, patients, and medical records.**

**Phase I: Project Initiation**

**Step 1: Problem Statement**

**Chosen Problem:** **Clinic Management System**

In a healthcare domain, managing a clinic efficiently involves tracking patient appointments, medical records, doctors' schedules, and other relevant data. A clinic management system (CMS) provides a streamlined way to manage these aspects, making the clinic's operations smoother and ensuring that medical records and appointments are handled effectively.

The CMS will include the following features:

1. **User Registration and Login:**
   * Patients and doctors should be able to sign up and log in to the system.
   * Different roles will be assigned, such as "admin," "doctor," and "patient."
2. **Appointment Management:**
   * Patients can view available doctors and book appointments.
   * Doctors can manage their schedules and view patient appointments.
3. **Patient Records Management:**
   * Medical records for each patient will be maintained by doctors, including diagnoses, treatments, and prescriptions.
4. **Doctor Management:**
   * A list of doctors, their specialties, and schedules will be maintained.
5. **Security:**
   * Patient data, appointment information, and medical records will be securely stored and only accessible by authorized users.

**Data Representation:** The following data needs to be represented and captured in the system:

* **Users:** Includes all users of the system, such as patients, doctors, and admins.
  + Attributes: UserID, Username, Password, Role (patient/doctor/admin), Contact, Address
* **Patients:** Details of the patients who visit the clinic.
  + Attributes: PatientID, Name, DateOfBirth, Contact, Address, MedicalHistory
* **Doctors:** Information about the doctors who work at the clinic.
  + Attributes: DoctorID, Name, Specialty, Contact, Address
* **Appointments:** Scheduled appointments between patients and doctors.
  + Attributes: AppointmentID, PatientID, DoctorID, Date, Time, Status
* **Medical Records:** Records of medical diagnoses and treatments for patients.
  + Attributes: RecordID, PatientID, DoctorID, Diagnosis, Treatment, Date

**Relationships:**

* **User** can be either a patient or a doctor (one-to-one relationship with the Users table).
* **Patients** can have multiple **Appointments** (one-to-many relationship between Patients and Appointments).
* **Doctors** can have multiple **Appointments** (one-to-many relationship between Doctors and Appointments).
* **Patients** can have multiple **Medical Records** (one-to-many relationship between Patients and Medical Records).
* **Doctors** can create multiple **Medical Records** for **Patients** (one-to-many relationship between Doctors and Medical Records).

**Step 2: Formulate at Least 8 Realistic Queries**

1. **Query 1:**
   * **Get a list of all upcoming appointments for a specific doctor.**

SELECT Appointments.AppointmentID, Patients.Name, Appointments.Date, Appointments.Time

FROM Appointments

JOIN Patients ON Appointments.PatientID = Patients.PatientID

WHERE Appointments.DoctorID = ? AND Appointments.Date >= CURDATE();

**Query 2:**

* **Get the details of all patients treated by a specific doctor**

SELECT DISTINCT Patients.PatientID, Patients.Name, Patients.Contact

FROM MedicalRecords

JOIN Patients ON MedicalRecords.PatientID = Patients.PatientID

WHERE MedicalRecords.DoctorID = ?;

**Query 3:**

* + **Find the total number of appointments scheduled for a specific date.**

SELECT COUNT(\*) AS TotalAppointments

FROM Appointments

WHERE Date = ?;

**Query 4:**

* + **Get the list of all doctors working on a specific day.**

SELECT Doctors.Name, Doctors.Specialty

FROM Doctors

JOIN Appointments ON Doctors.DoctorID = Appointments.DoctorID

WHERE Appointments.Date = ?;

**Query 5:**

* + **Get the medical record history for a patient, including the doctor who treated them.**

SELECT MedicalRecords.Diagnosis, MedicalRecords.Treatment, Doctors.Name AS DoctorName, MedicalRecords.Date

FROM MedicalRecords

JOIN Doctors ON MedicalRecords.DoctorID = Doctors.DoctorID

WHERE MedicalRecords.PatientID = ?;

**Query 6:**

* + **Get the list of all patients who have an appointment with a specific doctor within the next 7 days.**

SELECT Patients.Name, Appointments.Date, Appointments.Time

FROM Appointments

JOIN Patients ON Appointments.PatientID = Patients.PatientID

WHERE Appointments.DoctorID = ? AND Appointments.Date BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 7 DAY);

**Query 7:**

* + **Get the list of doctors with their specialties and the number of patients they have treated.**

SELECT Doctors.Name, Doctors.Specialty, COUNT(DISTINCT MedicalRecords.PatientID) AS PatientsTreated

FROM Doctors

LEFT JOIN MedicalRecords ON Doctors.DoctorID = MedicalRecords.DoctorID

GROUP BY Doctors.DoctorID;

**Query 8:**

* + **Update the status of an appointment (e.g., cancel, reschedule, etc.).**

UPDATE Appointments

SET Status = ?

WHERE AppointmentID = ?;

**Step 3: Assumptions About the Database**

1. **Entities:**
   * **Users:** Stores details of all users (patients, doctors, and admins) of the system.
   * **Patients:** Stores details about the patients who visit the clinic.
   * **Doctors:** Stores details about the doctors working at the clinic.
   * **Appointments:** Stores the appointment details, linking patients and doctors.
   * **MedicalRecords:** Stores medical diagnoses and treatment records for each patient.
2. **Attributes:**
   * **Users:** UserID, Username, Password, Role
   * **Patients:** PatientID, Name, DateOfBirth, Contact, MedicalHistory
   * **Doctors:** DoctorID, Name, Specialty, Contact
   * **Appointments:** AppointmentID, PatientID, DoctorID, Date, Time, Status
   * **MedicalRecords:** RecordID, PatientID, DoctorID, Diagnosis, Treatment, Date
3. **Keys:**
   * **Primary Keys:**
     + UserID for the Users table
     + PatientID for the Patients table
     + DoctorID for the Doctors table
     + AppointmentID for the Appointments table
     + RecordID for the MedicalRecords table
   * **Foreign Keys:**
     + PatientID in Appointments (references Patients.PatientID)
     + DoctorID in Appointments (references Doctors.DoctorID)
     + PatientID in MedicalRecords (references Patients.PatientID)
     + DoctorID in MedicalRecords (references Doctors.DoctorID)
4. **Relationships:**
   * A **patient** can have multiple **appointments**.
   * A **doctor** can have multiple **appointments**.
   * A **patient** can have multiple **medical records**, but each medical record is created by one **doctor**.
   * A **doctor** can create multiple **medical records** for different patients.

**Step 4: EER Modeling**

The Enhanced Entity-Relationship (EER) model for the Clinic Appointment Management System consists of the following components:

**Entities and Attributes:**

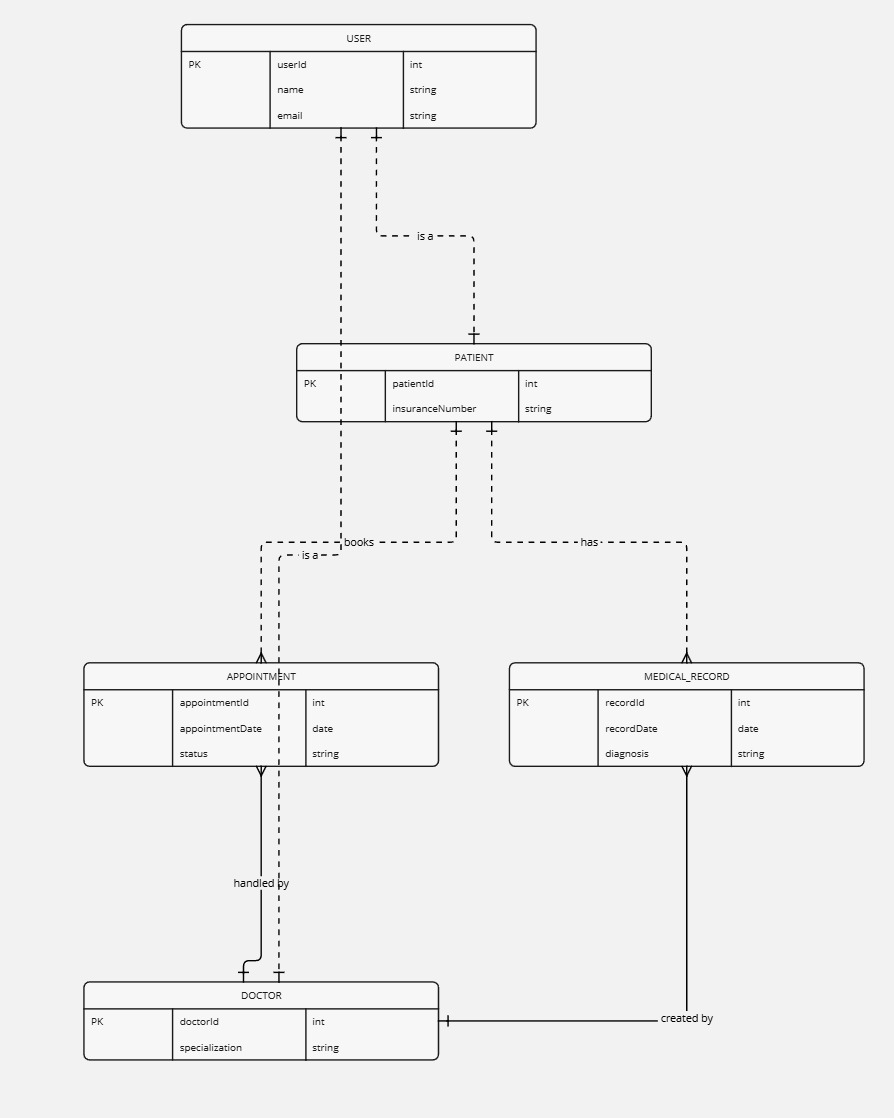
* Users (UserID, Username, Password, Role)
* Patients (PatientID, Name, DOB, Contact, Address, MedicalHistory, UserID)
* Doctors (DoctorID, Name, Specialty, Contact, Address, UserID)
* Appointments (AppointmentID, PatientID, DoctorID, Date, Time, Status)
* MedicalRecords (RecordID, PatientID, DoctorID, Diagnosis, Treatment, Date)

**Relationships:**

* User Specialization: A User can be either a Patient or a Doctor (Generalization/Specialization)
* Patient-Appointments Relationship: A Patient can book multiple appointments (1:M)
* Doctor-Appointments Relationship: A Doctor handles multiple appointments (1:M)
* Patient-MedicalRecords Relationship: A Patient can have multiple medical records (1:M)
* Doctor-MedicalRecords Relationship: A Doctor creates multiple medical records (1:M)
* Additional Features: Weak Entity: MedicalRecords (dependent on both Patients and Doctors)

**Attributes on Relationships:**

The Appointment entity includes a Status attribute indicating whether it is scheduled, completed, or canceled.



**Step 5: Relational Schema**

Based on the EER model, the following relational schema is designed:

**Tables and Schema:**

1. Users (UserID (PK), Username, Password, Role)
2. Patients (PatientID (PK), Name, DOB, Contact, Address, MedicalHistory, UserID (FK))
3. Doctors (DoctorID (PK), Name, Specialty, Contact, Address, UserID (FK))
4. Appointments (AppointmentID (PK), PatientID (FK), DoctorID (FK), Date, Time, Status)
5. MedicalRecords (RecordID (PK), PatientID (FK), DoctorID (FK), Diagnosis, Treatment, Date)

**Primary and Foreign Keys:**

**Primary Keys:**

* UserID, PatientID, DoctorID, AppointmentID, RecordID

**Foreign Keys:**

1. PatientID in Appointments references Patients (PatientID)
2. DoctorID in Appointments references Doctors (DoctorID)
3. PatientID in MedicalRecords references Patients (PatientID)
4. DoctorID in MedicalRecords references Doctors (DoctorID)

**Step 6: Relational Algebra Queries**

The following relational algebra expressions define critical queries in the system:

1. Get a list of all upcoming appointments for a specific doctor:

π AppointmentID, Name, Date, Time (σ DoctorID = 'X' ∧ Date >= TODAY (Appointments ⨝ Patients))

1. Get the details of all patients treated by a specific doctor:

π PatientID, Name, Contact (σ DoctorID = 'X' (MedicalRecords ⨝ Patients))

1. Find the total number of appointments scheduled for a specific date:

γ COUNT(AppointmentID) → TotalAppointments (σ Date = 'Y' (Appointments))

1. Get the list of all doctors working on a specific day:

π Name, Specialty (σ Date = 'Z' (Doctors ⨝ Appointments))

1. Retrieve the medical record history for a patient, including the doctor who treated them:

π Diagnosis, Treatment, DoctorName, Date (MedicalRecords ⨝ Doctors ⨝ Patients)

1. List all patients who have an appointment with a specific doctor within the next 7 days:

π Name, Date, Time (σ DoctorID = 'X' ∧ Date BETWEEN TODAY AND TODAY+7 (Appointments ⨝ Patients))

1. Get the list of doctors with their specialties and the number of patients they have treated:

γ Name, Specialty, COUNT(PatientID) → PatientsTreated (Doctors ⨝ MedicalRecords)

1. Update the status of an appointment (e.g., cancel, reschedule, etc.):

δ Status = 'NewStatus' (σ AppointmentID = 'A' (Appointments))

**STEP 7:** **SQL DDL script**

**dbDDL.sql:**

**-- Creating Users Table**

CREATE TABLE Users (

UserID INT PRIMARY KEY,

Username VARCHAR (50) NOT NULL UNIQUE,

Password VARCHAR (255) NOT NULL,

Role VARCHAR (20) CHECK (Role IN ('Admin', 'Doctor', 'Patient'))

);

**-- Creating Patients Table**

CREATE TABLE Patients (

PatientID INT PRIMARY KEY,

Name VARCHAR (100),

DOB DATE,

Contact VARCHAR (15),

Address VARCHAR (255),

MedicalHistory TEXT,

UserID INT,

FOREIGN KEY (UserID) REFERENCES Users (UserID) ON DELETE CASCADE

);

**-- Creating Doctors Table**

CREATE TABLE Doctors (

DoctorID INT PRIMARY KEY,

Name VARCHAR (100),

Specialty VARCHAR (100),

Contact VARCHAR (15),

Address VARCHAR (255),

UserID INT,

FOREIGN KEY (UserID) REFERENCES Users (UserID) ON DELETE CASCADE

);

**-- Creating Appointments Table**

CREATE TABLE Appointments (

AppointmentID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

Date DATE NOT NULL,

Time TIME NOT NULL,

Status VARCHAR (20) DEFAULT 'Scheduled',

FOREIGN KEY (PatientID) REFERENCES Patients (PatientID) ON DELETE CASCADE,

FOREIGN KEY (DoctorID) REFERENCES Doctors (DoctorID) ON DELETE CASCADE

);

**-- Creating Medical Records Table**

CREATE TABLE MedicalRecords (

RecordID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

Diagnosis TEXT NOT NULL,

Treatment TEXT NOT NULL,

Date DATE NOT NULL,

FOREIGN KEY (PatientID) REFERENCES Patients (PatientID) ON DELETE CASCADE,

FOREIGN KEY (DoctorID) REFERENCES Doctors (DoctorID) ON DELETE CASCADE

);

**-- Creating Admins Table (Additional Table)**

CREATE TABLE Admins (

AdminID INT PRIMARY KEY,

Name VARCHAR (100),

Contact VARCHAR (15),

UserID INT,

FOREIGN KEY (UserID) REFERENCES Users (UserID) ON DELETE CASCADE

);

**-- Creating Billing Table (Additional Table)**

CREATE TABLE Billing (

BillID INT PRIMARY KEY,

AppointmentID INT,

Amount DECIMAL (10, 2) NOT NULL,

PaymentStatus VARCHAR (20) DEFAULT 'Pending',

FOREIGN KEY (AppointmentID) REFERENCES Appointments (AppointmentID) ON DELETE CASCADE

);

**-- Creating Specializations Table (Additional Table)**

CREATE TABLE Specializations (

SpecialtyID INT PRIMARY KEY,

SpecialtyName VARCHAR(100) NOT NULL

);

**-- Creating an Index on Appointments for faster searching by DoctorID**

CREATE INDEX idx\_doctor ON Appointments (DoctorID);

**-- Creating an Index on Appointments for faster searching by PatientID**

CREATE INDEX idx\_patient ON Appointments (PatientID);

**-- Creating a View: All Active Appointments**

CREATE VIEW ActiveAppointments AS

SELECT

a.AppointmentID, p.Name AS PatientName, d.Name AS DoctorName, a.Date, a.Time, a.Status

FROM

Appointments a

JOIN

Patients p ON a.PatientID = p.PatientID

JOIN

Doctors d ON a.DoctorID = d.DoctorID

WHERE

a.Status = 'Scheduled';

**-- Creating a Trigger: Automatically Update Appointment Status after Appointment Date**

CREATE TRIGGER UpdateAppointmentStatus

AFTER UPDATE ON Appointments

FOR EACH ROW

WHEN (NEW.Date < CURRENT\_DATE AND NEW.Status = 'Scheduled')

BEGIN

UPDATE Appointments

SET Status = 'Completed'

WHERE AppointmentID = NEW.AppointmentID;

END;

**Description of DDL Script:**

* Tables:
  + There are 8 tables: Users, Patients, Doctors, Appointments, MedicalRecords, Admins, Billing, and Specializations.
  + Each table includes appropriate Primary Keys and Foreign Key constraints to maintain relationships between the entities.
* Indexes:
  + Two indexes are created on DoctorID and PatientID in the Appointments table to improve query performance for searching appointments.
* View:
  + ActiveAppointments is a view that shows all active appointments where the status is 'Scheduled', displaying patient and doctor names along with appointment details.
* Trigger:
  + The UpdateAppointmentStatus trigger automatically updates the status of appointments to 'Completed' if the appointment date has passed, ensuring the system's status is always up-to-date.

**STEP 8:DML SCRIPT**

**dbDML.sql:**

**-- Inserting sample data into Users Table**

INSERT INTO Users (UserID, Username, Password, Role) VALUES

(1, 'admin1', 'adminpass', 'Admin'),

(2, 'doctor1', 'docpass1', 'Doctor'),

(3, 'doctor2', 'docpass2', 'Doctor'),

(4, 'patient1', 'patpass1', 'Patient'),

(5, 'patient2', 'patpass2', 'Patient'),

(6, 'patient3', 'patpass3', 'Patient'),

(7, 'admin2', 'adminpass2', 'Admin'),

(8, 'patient4', 'patpass4', 'Patient');

**-- Inserting sample data into Patients Table**

INSERT INTO Patients (PatientID, Name, DOB, Contact, Address, MedicalHistory, UserID) VALUES

(1, 'John Doe', '1990-05-20', '555-1234', '123 Elm St', 'Diabetes', 4),

(2, 'Jane Smith', '1985-07-15', '555-5678', '456 Maple Ave', 'Asthma', 5),

(3, 'Michael Brown', '1975-03-30', '555-4321', '789 Pine Blvd', 'Hypertension', 6),

(4, 'Lisa Ray', '1992-11-22', '555-9876', '135 Oak Dr', 'None', 8),

(5, 'Tom Hanks', '1988-12-05', '555-6543', '246 Birch Ct', 'Anxiety', 8);

**-- Inserting sample data into Doctors Table**

INSERT INTO Doctors (DoctorID, Name, Specialty, Contact, Address, UserID) VALUES

(1, 'Dr. Alice Johnson', 'Cardiology', '555-1111', '101 Heart St', 2),

(2, 'Dr. Bob Anderson', 'Neurology', '555-2222', '202 Brain Ln', 3),

(3, 'Dr. Emily Green', 'Dermatology', '555-3333', '303 Skin Ave', 2);

**-- Inserting sample data into Appointments Table**

INSERT INTO Appointments (AppointmentID, PatientID, DoctorID, Date, Time, Status) VALUES

(1, 1, 1, '2025-03-30', '09:00:00', 'Scheduled'),

(2, 2, 1, '2025-03-31', '10:00:00', 'Scheduled'),

(3, 3, 2, '2025-03-30', '11:00:00', 'Scheduled'),

(4, 1, 2, '2025-03-28', '14:00:00', 'Completed'),

(5, 4, 3, '2025-04-01', '09:30:00', 'Scheduled'),

(6, 2, 3, '2025-04-02', '10:30:00', 'Scheduled'),

(7, 5, 1, '2025-03-29', '15:00:00', 'Completed');

**-- Inserting sample data into MedicalRecords Table**

INSERT INTO MedicalRecords (RecordID, PatientID, DoctorID, Diagnosis, Treatment, Date) VALUES

(1, 1, 1, 'Hypertension', 'Medication prescribed', '2025-03-01'),

(2, 2, 1, 'Asthma', 'Inhaler prescribed', '2025-03-10'),

(3, 3, 2, 'Chronic Headaches', 'MRI recommended', '2025-02-28'),

(4, 4, 3, 'Skin Rash', 'Ointment prescribed', '2025-03-15'),

(5, 1, 2, 'Mild Anxiety', 'Counseling recommended', '2025-03-20'),

(6, 2, 3, 'Eczema', 'Cream prescribed', '2025-03-25'),

(7, 5, 1, 'Diabetes', 'Diet and exercise plan', '2025-03-05');

**-- Inserting sample data into Admins Table**

INSERT INTO Admins (AdminID, Name, Contact, UserID) VALUES

(1, 'Sarah Lee', '555-8765', 1),

(2, 'David Wong', '555-9987', 7);

**-- Inserting sample data into Billing Table**

INSERT INTO Billing (BillID, AppointmentID, Amount, PaymentStatus) VALUES

(1, 1, 150.00, 'Paid'),

(2, 2, 200.00, 'Pending'),

(3, 3, 250.00, 'Paid'),

(4, 4, 300.00, 'Paid'),

(5, 5, 180.00, 'Pending'),

(6, 6, 160.00, 'Pending'),

(7, 7, 120.00, 'Paid');

**--** **Inserting** **sample data into Specializations Table**

INSERT INTO Specializations (SpecialtyID, SpecialtyName) VALUES

(1, 'Cardiology'),

(2, 'Neurology'),

(3, 'Dermatology'),

(4, 'Pediatrics'),

(5, 'Orthopedics');

**-- Updating the status of an appointment**

UPDATE Appointments

SET Status = 'Completed'

WHERE AppointmentID = 5;

**-- Deleting a patient record**

DELETE FROM Patients

WHERE PatientID = 5;

**Description of DML Script:**

* Insert Statements:
  + Populates the Users, Patients, Doctors, Appointments, MedicalRecords, Admins, Billing, and Specializations tables with 7-10 sample entries per table.
* Update Statement:
  + Updates the status of an appointment from 'Scheduled' to 'Completed' for AppointmentID = 5.
* Delete Statement:
  + Deletes the patient with PatientID = 5 from the Patients table.

**STEP 9 : Creating Drop Script**

**dbDROP.sql:**

**-- Dropping the Trigger**

DROP TRIGGER IF EXISTS UpdateAppointmentStatus;

**-- Dropping the View**

DROP VIEW IF EXISTS ActiveAppointments;

**-- Dropping the Indexes**

DROP INDEX IF EXISTS idx\_doctor ON Appointments;

DROP INDEX IF EXISTS idx\_patient ON Appointments;

**-- Dropping the Foreign Key Constraints and Tables**

ALTER TABLE MedicalRecords DROP CONSTRAINT IF EXISTS fk\_MedicalRecords\_PatientID;

ALTER TABLE MedicalRecords DROP CONSTRAINT IF EXISTS fk\_MedicalRecords\_DoctorID;

ALTER TABLE Appointments DROP CONSTRAINT IF EXISTS fk\_Appointments\_PatientID;

ALTER TABLE Appointments DROP CONSTRAINT IF EXISTS fk\_Appointments\_DoctorID;

ALTER TABLE Patients DROP CONSTRAINT IF EXISTS fk\_Patients\_UserID;

ALTER TABLE Doctors DROP CONSTRAINT IF EXISTS fk\_Doctors\_UserID;

ALTER TABLE Admins DROP CONSTRAINT IF EXISTS fk\_Admins\_UserID;

ALTER TABLE Billing DROP CONSTRAINT IF EXISTS fk\_Billing\_AppointmentID;

**-- Dropping the Tables**

DROP TABLE IF EXISTS Billing;

DROP TABLE IF EXISTS MedicalRecords;

DROP TABLE IF EXISTS Appointments;

DROP TABLE IF EXISTS Patients;

DROP TABLE IF EXISTS Doctors;

DROP TABLE IF EXISTS Admins;

DROP TABLE IF EXISTS Specializations;

DROP TABLE IF EXISTS Users;

**Description of dbDROP.sql:**

* Order of Dropping:
  + The trigger (UpdateAppointmentStatus) is dropped first because it is dependent on the Appointments table.
  + The view (ActiveAppointments) is dropped next because it depends on the Appointments, Patients, and Doctors tables.
  + The indexes are dropped before the tables.
  + Foreign key constraints are removed before dropping tables to avoid dependency errors.
  + Finally, all the tables are dropped in the appropriate order, starting with those that don't have dependencies.

**STEP 10: Creating SQL Script  
Query 1: Join Query (Patients and their Appointments with Doctor's Name)**

**-- This query retrieves the list of patients with their appointment details, along with the doctor's name.**

SELECT

p.Name AS PatientName,

a.Date AS AppointmentDate,

a.Time AS AppointmentTime,

d.Name AS DoctorName

FROM

Patients p

JOIN

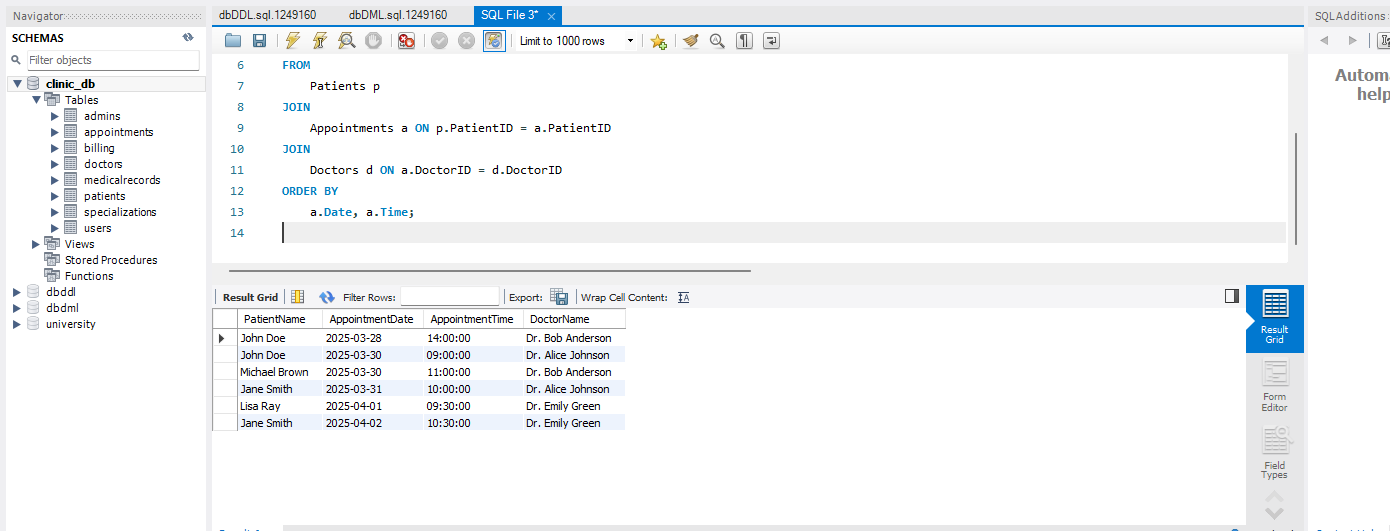
Appointments a ON p.PatientID = a.PatientID

JOIN

Doctors d ON a.DoctorID = d.DoctorID

ORDER BY

a.Date, a.Time;



**-- Query 2: Aggregate Query with GROUP BY, HAVING, and ORDER BY**

**-- This query calculates the total number of appointments scheduled for each doctor, only showing those doctors with more than 1 appointment, and orders them by the total number of appointments.**

SELECT

d.Name AS DoctorName,

COUNT(a.AppointmentID) AS TotalAppointments

FROM

Doctors d

JOIN

Appointments a ON d.DoctorID = a.DoctorID

GROUP BY

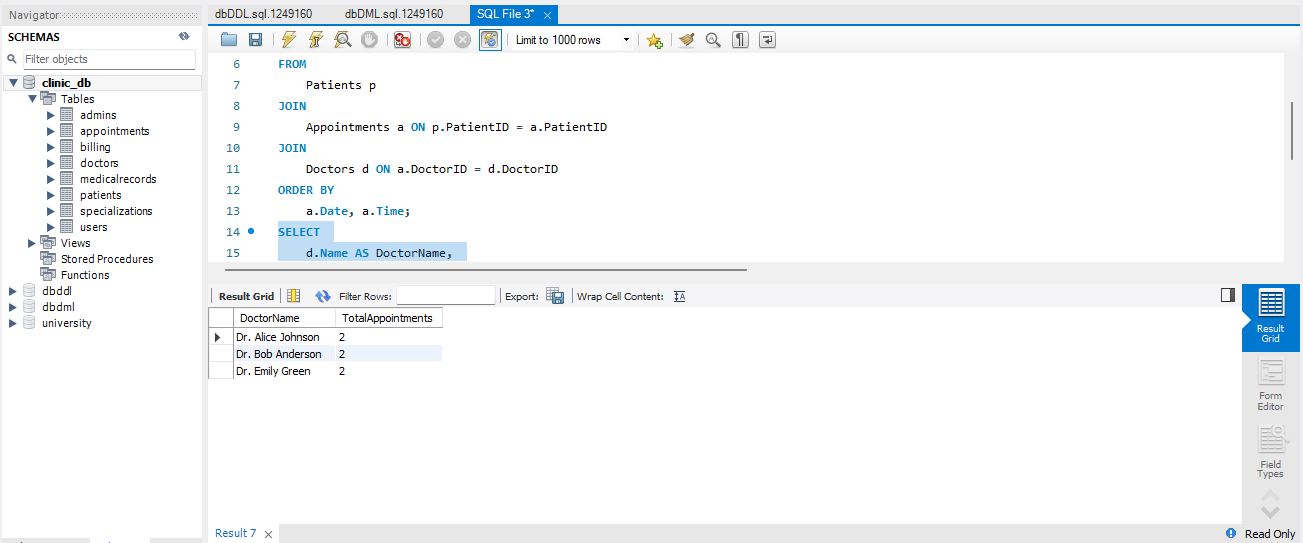
d.Name

HAVING

COUNT(a.AppointmentID) > 1

ORDER BY

TotalAppointments DESC;



**-- Query 3: Nested Subquery**

**-- This query retrieves the list of patients who have had more than one appointment with the same doctor.**

SELECT

p.Name AS PatientName,

d.Name AS DoctorName

FROM

Patients p

JOIN

Appointments a ON p.PatientID = a.PatientID

JOIN

Doctors d ON a.DoctorID = d.DoctorID

WHERE

p.PatientID IN (SELECT

a2.PatientID

FROM

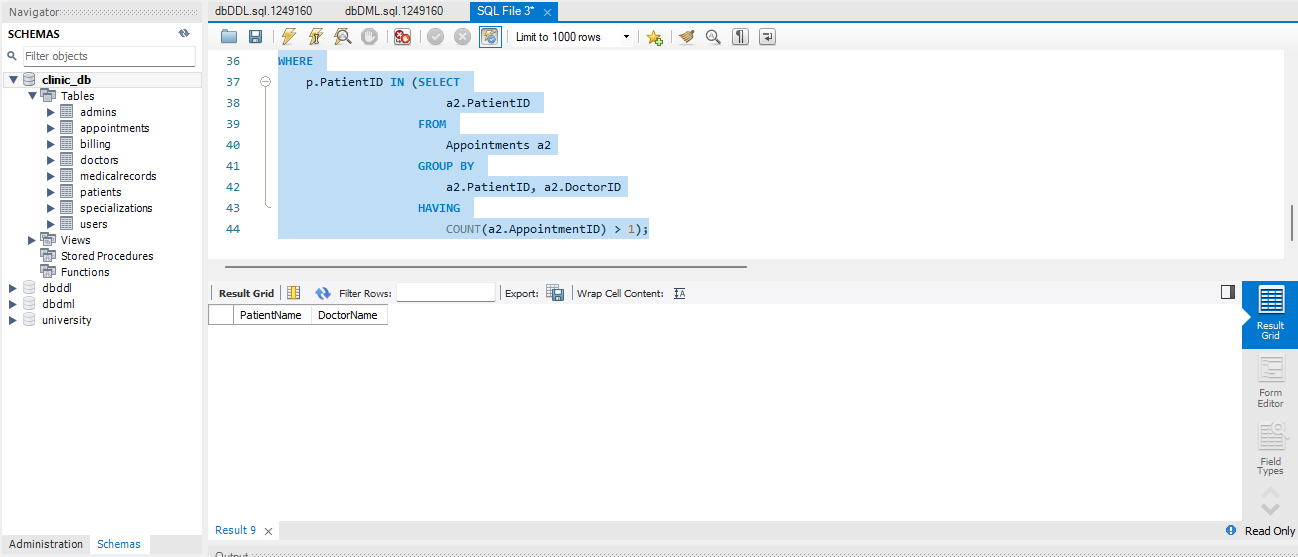
Appointments a2

GROUP BY

a2.PatientID, a2.DoctorID

HAVING

COUNT(a2.AppointmentID) > 1);



**-- Query 4: Join with 3 Relations and Conditional Filtering**

**-- This query retrieves the medical records along with the patient name, doctor name, and appointment date for patients who have been diagnosed with 'Hypertension'.**

SELECT

p.Name AS PatientName,

d.Name AS DoctorName,

m.Diagnosis,

m.Treatment,

m.Date AS RecordDate

FROM

MedicalRecords m

JOIN

Patients p ON m.PatientID = p.PatientID

JOIN

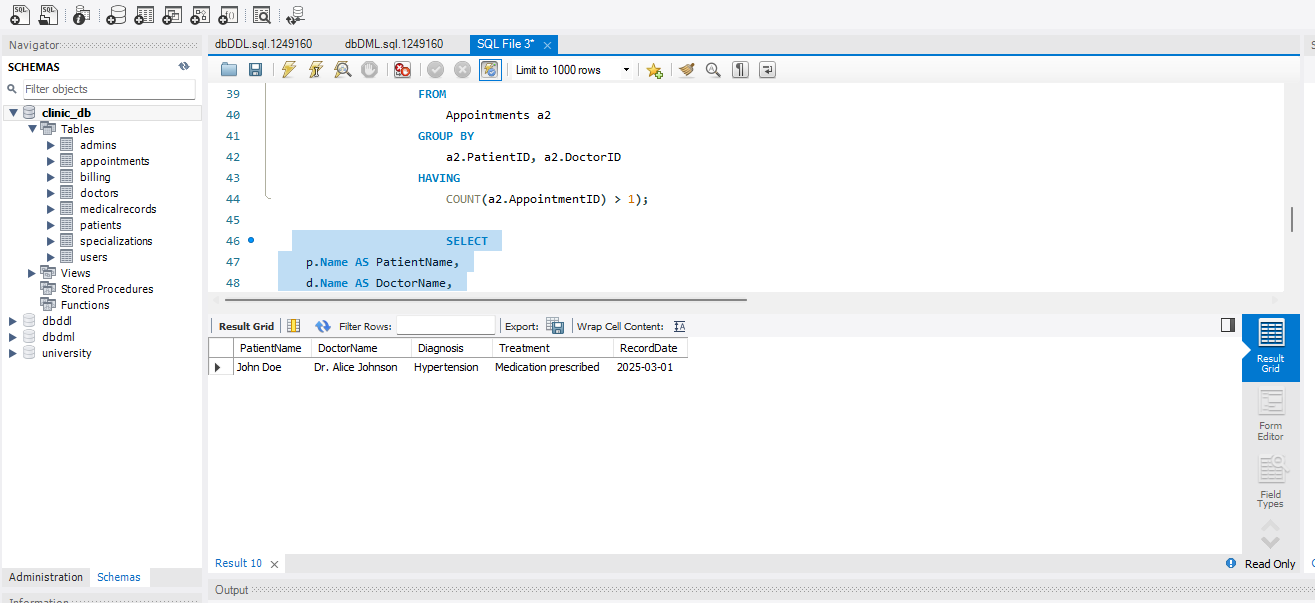
Doctors d ON m.DoctorID = d.DoctorID

WHERE

m.Diagnosis = 'Hypertension'

ORDER BY

m.Date;



**-- Query 5: Aggregate Query with a Join and SUM**

**-- This query calculates the total amount billed to each patient and orders the result by the total bill amount.**

SELECT

p.Name AS PatientName,

SUM(b.Amount) AS TotalBilled

FROM

Patients p

JOIN

Appointments a ON p.PatientID = a.PatientID

JOIN

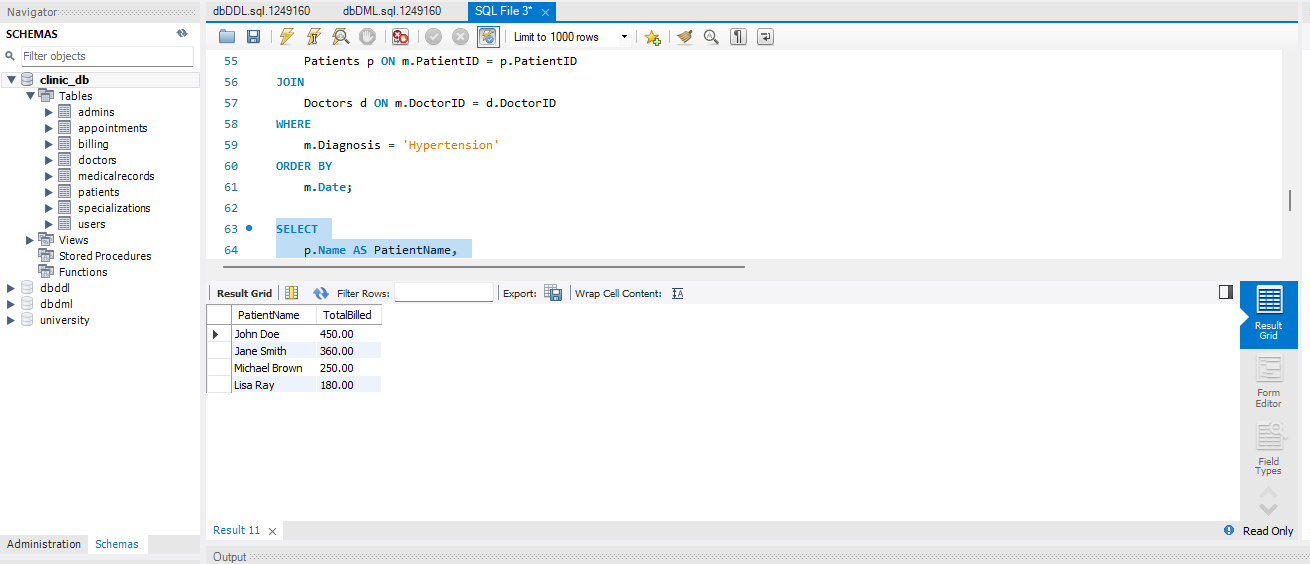
Billing b ON a.AppointmentID = b.AppointmentID

GROUP BY

p.Name

ORDER BY

TotalBilled DESC;



**Description of dbSQL.sql Queries:**

1. **Query 1 (Join Query):**  
   Joins the Patients, Appointments, and Doctors tables to list all appointments, including the patient's name and doctor's name. The result is sorted by the appointment date and time.
2. **Query 2 (Aggregate Query with GROUP BY, HAVING, and ORDER BY):**  
   Counts the total appointments for each doctor, filters the results to show only those doctors who have more than 1 appointment, and orders the result by the total number of appointments.
3. **Query 3 (Nested Subquery):**  
   Uses a nested subquery to find patients who have had more than one appointment with the same doctor.
4. **Query 4 (Join with 3 Relations and Conditional Filtering):**  
   Joins the MedicalRecords, Patients, and Doctors tables and filters for patients diagnosed with "Hypertension", displaying their treatment details.
5. **Query 5 (Aggregate Query with SUM):**  
   Calculates the total amount billed to each patient by joining the Patients, Appointments, and Billing tables and summing the Amount. The result is sorted by the total billed amount in descending order.

Phase IV: Application Development

**Application Implementation**

This phase involves the development of a front-end application that interacts with the **Clinic Appointment Management System**'s backend (Flask) and MySQL database. The primary goal is to enable users (patients, doctors, and admins) to manage appointments, medical records, and doctor schedules, without needing to write SQL queries. The user interface (UI) is designed to be intuitive and responsive.

**HTML**: Used for the structure of the web pages.

* **CSS**: Styling the web pages and ensuring responsive design with frameworks like **Bootstrap**.
* **JavaScript**: For dynamic interactions and real-time operations (like AJAX calls for fetching and submitting data).

**Libraries/Frameworks**:

* **Bootstrap**: Used for responsive design, ensuring the web application works well on mobile and desktop.
* **jQuery** : Simplifies JavaScript tasks such as AJAX requests.

**Core Functionalities of the Application**

**User Registration and Login**

* **Patients, Doctors, and Admins** can sign up and log in securely.
* **Flask-Login** is used to manage user sessions and restrict access based on roles.
  + **Admin**: Full access to manage users, appointments, and doctors.
  + **Doctor**: Can manage their schedule and medical records for their patients.
  + **Patient**: Can view and book appointments, and access their own medical records.

**Appointment Management**

* **Patients**: Can view available doctors, book appointments, and cancel or reschedule them.
* **Doctors**: Can view their scheduled appointments and manage them.
* **Admins**: Can view and manage appointments across all doctors and patients.

**Medical Records Management**

* **Doctors**: Create, view, and update medical records for patients.
* **Patients**: Can view their medical history, but cannot modify records.
* **Admins**: Can access all medical records across the system.

**Doctor Management**

* **Admins**: Add, remove, and update doctor profiles (name, specialty, contact information).
* **Doctors**: Can view and update their profiles but cannot modify others’ information.

**Frontend Design and Navigation**

**Pages and Features:**

1. **Home Page**:
   * The landing page features the **Login** and **Sign-up** options.
   * After logging in, users are redirected to their respective dashboards (patient, doctor, or admin).
2. **Login and Signup Pages**:
   * Users can either log in with their existing credentials or sign up for a new account.
   * Passwords are securely hashed using **Werkzeug**.
   * After login, the user is redirected to their respective role-based dashboard.
3. **Dashboard Pages**:
   * **Admin Dashboard**: Full access to manage users, appointments, and doctors.
   * **Doctor Dashboard**: Displays scheduled appointments and patient records.
   * **Patient Dashboard**: Displays upcoming appointments, medical history, and an option to book new appointments.
4. **Appointment Booking Page**:
   * Patients can see available slots for doctors and book appointments. This is handled dynamically via AJAX calls to the backend.

**Connecting Front-End to Back-End**

The front-end uses **AJAX** or **Fetch API** to interact with the Flask backend, allowing the application to perform operations without page reloads. Here's a breakdown of how the interactions work:

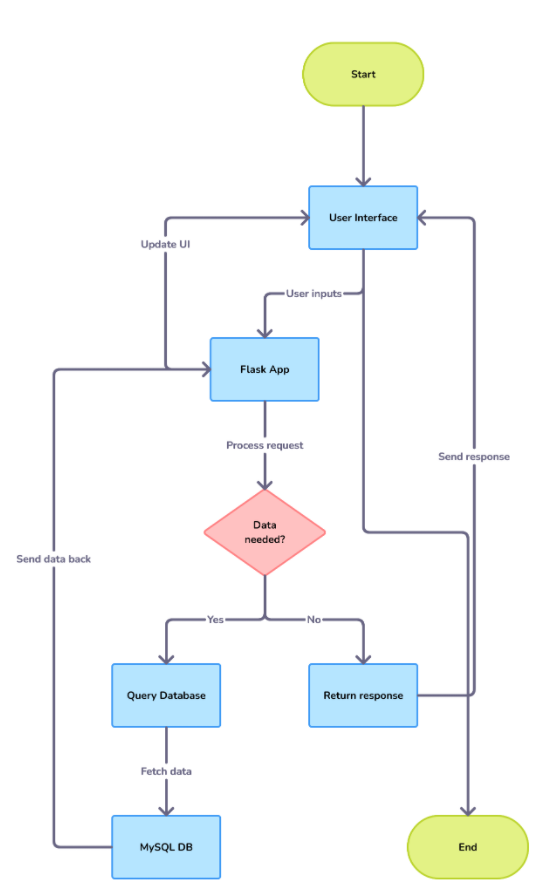
* **Login**:  
  When a user submits their username and password, a POST request is sent to the Flask backend. The backend checks if the credentials are valid, creates a session, and sends a success response.
* **Appointments**:  
  When a patient books an appointment, a POST request is made to the backend, which inserts the appointment into the MySQL database. The front-end updates the list of appointments dynamically without refreshing the page.
* **Medical Records**:  
  Doctors can create and update medical records, which are sent to the backend for storage in the database. Patients can view their medical records but cannot modify them.
* **Role-Based Access**:  
  Based on the role (admin, doctor, patient), the front-end hides or displays certain content. Admins have access to all features, while doctors and patients can only access their own data.

**Testing and Debugging**

Testing ensures the system works as expected, and various types of testing are carried out:

* **Unit Testing**: Testing individual components of the system, such as appointment booking or login functionality.
* **Integration Testing**: Ensuring that the front-end, backend, and database work seamlessly together.
* **User Acceptance Testing (UAT)**: Ensuring that the system meets user requirements and is easy to use. This includes testing the system’s usability by users with different roles.
* **Security Testing**: Ensuring that user data is securely stored, and features like login and user management work as expected.

System Configuration Diagram



**Conclusion**

This project provided a comprehensive understanding of developing a full-stack web application. The system integrates various functionalities such as role-based access control, appointment management, and medical record tracking, which are essential for a clinic's day-to-day operations. Future improvements could include adding features like email notifications for appointments or integrating with external healthcare management systems.