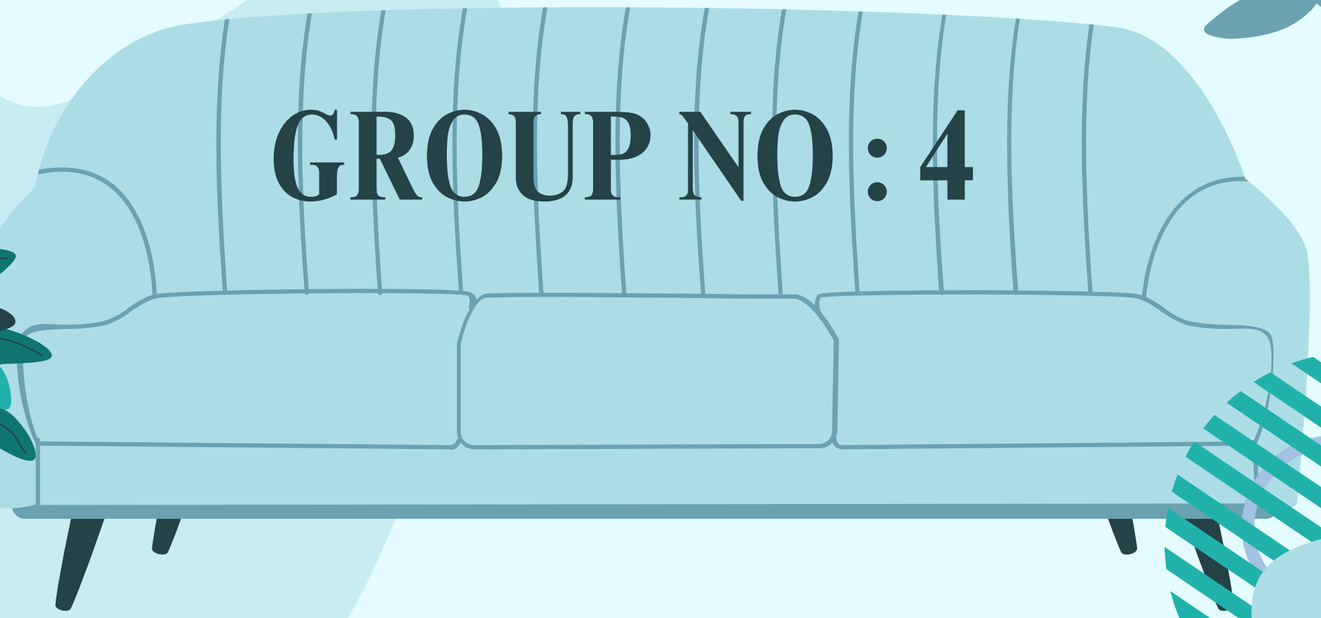
WELCOME

HOSPITAL DB

GROUP NO : 4



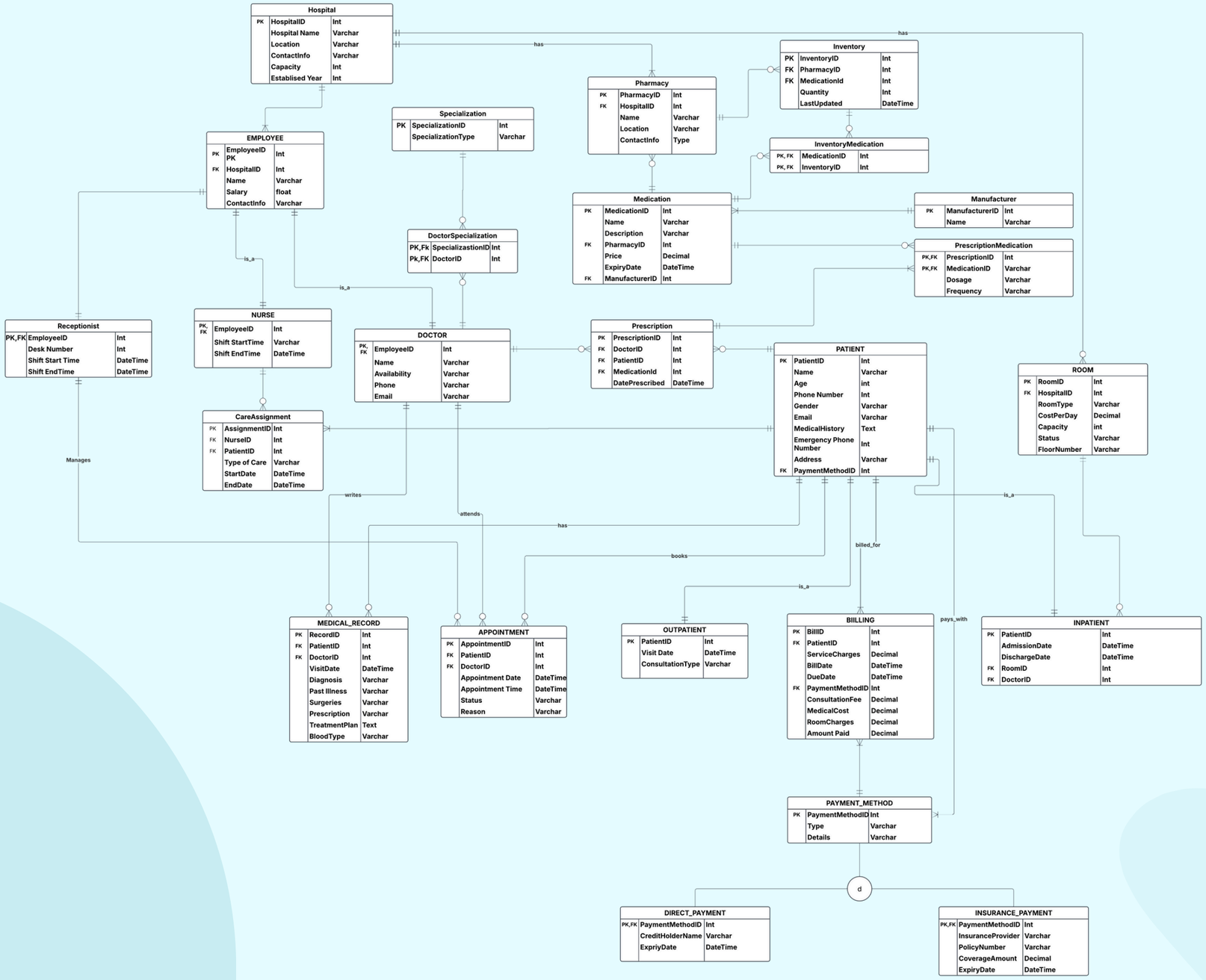

HRISHI PAL
LAVANYA RAJESH
MAYUR PATEL
SALONI MATHURE
MITHRAN EZHILARASAN

PROJECT OVERVIEW

Problem statement:

- Smart Hospitals Start with Smart Databases.
- Hospitals deal with a massive amount of information every day - patient details, doctor schedules, room assignments, prescriptions, bills, and pharmacy stock. When this information is stored in separate places or handled manually, it leads to delays, confusion, billing errors, and poor patient service. Staff may struggle to find available rooms, doctors may be double-booked, or important patient history might go unnoticed.
- To solve these issues, we designed a centralized Hospital Management System (HMS) using Microsoft SQL Server that brings together all key operations in one secure and efficient platform.

ENTITY-RELATIONSHIP DIAGRAM (ERD)

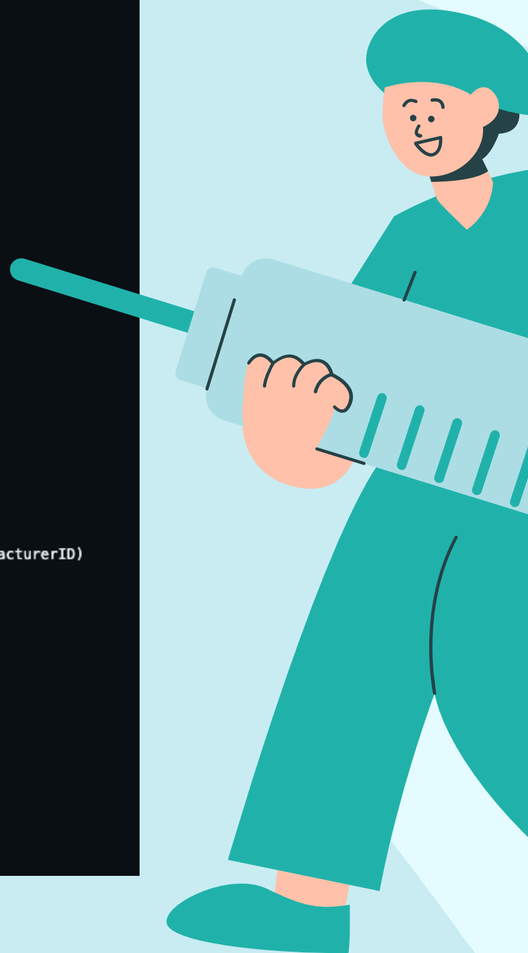


DDL

DDL (Data Definition Language) refers to the set of SQL commands used to create, modify, or remove the structural elements of the HospitalDB database.

```sql
-- Hospital Table
CREATE TABLE Hospital (
    HospitalID INT IDENTITY(1,1) PRIMARY KEY,
    HospitalName VARCHAR(255) NOT NULL,
    Location VARCHAR(255) NOT NULL,
    Capacity INT NOT NULL,
    EstablishedYear INT NOT NULL,
    ContactInfo VARCHAR(255) NOT NULL
);

-- Employee Table with Type Discriminator
CREATE TABLE Employee (
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,
    HospitalID INT,
    Name VARCHAR(255) NOT NULL,
    Salary DECIMAL(10, 2) CHECK (Salary >= 0),
    ContactInfo VARCHAR(255),
    EmployeeType VARCHAR(20) NOT NULL CHECK (EmployeeType IN ('Receptionist', 'Nurse', 'Doctor')),
    FOREIGN KEY (HospitalID) REFERENCES Hospital(HospitalID)
);

-- Receptionist Table with Type Enforcement
CREATE TABLE Receptionist (
    EmployeeID INT PRIMARY KEY,
    EmployeeType VARCHAR(20) NOT NULL CHECK (EmployeeType = 'Receptionist'),
    DeskNumber INT NOT NULL CHECK (DeskNumber > 0),
    ShiftStartTime DATETIME NOT NULL,
    ShiftEndTime DATETIME NOT NULL,
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);

-- Nurse Table with Type Enforcement
CREATE TABLE Nurse (
    EmployeeID INT PRIMARY KEY,
    EmployeeType VARCHAR(20) NOT NULL CHECK (EmployeeType = 'Nurse'),
    ShiftStartTime DATETIME NOT NULL,
    ShiftEndTime DATETIME NOT NULL,
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
);
```

```sql
-- Pharmacy Table
CREATE TABLE Pharmacy (
    PharmacyID INT IDENTITY(1,1) PRIMARY KEY,
    HospitalID INT,
    Name VARCHAR(255),
    Location VARCHAR(255),
    ContactInfo VARCHAR(255),
    FOREIGN KEY (HospitalID) REFERENCES Hospital(HospitalID)
);

-- Manufacturer Table
CREATE TABLE Manufacturer (
    ManufacturerID INT IDENTITY(1,1) PRIMARY KEY,
    Name VARCHAR(255)
);

-- Medication Table
CREATE TABLE Medication (
    MedicationID INT IDENTITY(1,1) PRIMARY KEY,
    PharmacyID INT,
    ManufacturerID INT,
    Name VARCHAR(255),
    Description TEXT,
    Price DECIMAL(10,2) CHECK (Price >= 0),
    ExpiryDate DATE,
    FOREIGN KEY (PharmacyID) REFERENCES Pharmacy(PharmacyID),
    FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID)
);

-- Prescription Table
CREATE TABLE Prescription (
    PrescriptionID INT IDENTITY(1,1) PRIMARY KEY,
    DoctorID INT,
    PatientID INT,
    PrescriptionDate DATE,
    Notes TEXT,
    FOREIGN KEY (DoctorID) REFERENCES Doctor(EmployeeID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);
```



CONSTRAINTS

Foreign key relationships are used to link related data across tables and maintain referential integrity. For example:

- **Employee.HospitalID** links employees to their hospital.
- **Patient.PaymentMethodID** connects patients to their payment method.
- **Appointment.PatientID and DoctorID** ensure each appointment belongs to a valid patient and doctor.
- I**npatient.RoomID** and **DoctorID** ensure correct room and doctor assignments.
- **Prescription.DoctorID** and **PatientID** confirm valid diagnosis mapping.

Check constraints are used to enforce valid, logical values across the database:

- **Patient.Gender:** Only accepts Male, Female, or Other.
- **PhoneNumber:** Ensures phone numbers are numeric and exactly 10 digits.
- **Room.Status:** Limited to Available, Occupied, or Under Maintenance.
- **Doctor.Availability:** Restricted to Full-time, Part-time, or On-call.
- **Billing and Outpatient numeric fields:** Prevent negative values for charges and fees.
- **Shift timings, CostPerDay, and CoverageAmount:** Validated to ensure realistic entries.

Unique constraints avoid duplication in fields that must remain distinct:

- **Email fields for Patients and Doctors:** Ensures no duplicate email addresses.
- Optionally can be extended to **Pharmacy.Name, Manufacturer.Name, etc.**, for better indexing.

DATA INGESTION

```
-- Insert 30 Employees (10 Receptionists, 10 Nurses, 10 Doctors)
INSERT INTO Employee (HospitalID, Name, Salary, ContactInfo, EmployeeType) VALUES
-- Receptionists (1-10)
(1, 'John Doe', 30000.00, 'john@example.com', 'Receptionist'),
(2, 'Jane Smith', 32000.00, 'jane@example.com', 'Receptionist'),
(3, 'Bob Wilson', 31000.00, 'bob@example.com', 'Receptionist'),
(4, 'Alice Brown', 30500.00, 'alice@example.com', 'Receptionist'),
(5, 'Charlie Green', 31500.00, 'charlie@example.com', 'Receptionist'),
(6, 'Diana White', 32500.00, 'diana@example.com', 'Receptionist'),
(7, 'Ethan Black', 30000.00, 'ethan@example.com', 'Receptionist'),
(8, 'Fiona Gray', 33000.00, 'fiona@example.com', 'Receptionist'),
(9, 'George Blue', 31000.00, 'george@example.com', 'Receptionist'),
(10, 'Hannah Yellow', 32000.00, 'hannah@example.com', 'Receptionist'),
-- Nurses (11-20)
(1, 'Ivy Adams', 45000.00, 'ivy@example.com', 'Nurse'),
(2, 'Jack Carter', 46000.00, 'jack@example.com', 'Nurse'),
(3, 'Karen Davis', 47000.00, 'karen@example.com', 'Nurse'),
(4, 'Liam Evans', 48000.00, 'liam@example.com', 'Nurse'),
(5, 'Mia Foster', 49000.00, 'mia@example.com', 'Nurse'),
(6, 'Noah Green', 50000.00, 'noah@example.com', 'Nurse'),
(7, 'Olivia Hall', 51000.00, 'olivia@example.com', 'Nurse'),
(8, 'Peter Inman', 52000.00, 'peter@example.com', 'Nurse'),
(9, 'Quinn Jones', 53000.00, 'quinn@example.com', 'Nurse'),
(10, 'Rachel King', 54000.00, 'rachel@example.com', 'Nurse'),
-- Doctors (21-30)
(1, 'Samuel Lee', 120000.00, 'samuel@example.com', 'Doctor'),
(2, 'Tina Moore', 130000.00, 'tina@example.com', 'Doctor'),
(3, 'Umar Nazir', 125000.00, 'umar@example.com', 'Doctor'),
(4, 'Victoria Park', 140000.00, 'victoria@example.com', 'Doctor'),
(5, 'Walter Quinn', 135000.00, 'walter@example.com', 'Doctor'),
(6, 'Xena Rhodes', 145000.00, 'xena@example.com', 'Doctor'),
(7, 'Yusuf Shah', 150000.00, 'yusuf@example.com', 'Doctor'),
(8, 'Zara Taylor', 160000.00, 'zara@example.com', 'Doctor'),
(9, 'Aaron Wells', 155000.00, 'aaron@example.com', 'Doctor'),
(10, 'Bella Young', 165000.00, 'bella@example.com', 'Doctor');
GO
```

```
-- Insert 10 Receptionists
INSERT INTO Receptionist (EmployeeID, EmployeeType, DeskNumber, ShiftStartTime, ShiftEndTime) VALUES
(1, 'Receptionist', 1, '2023-01-01 08:00:00', '2023-01-01 16:00:00'),
(2, 'Receptionist', 2, '2023-01-01 09:00:00', '2023-01-01 17:00:00'),
(3, 'Receptionist', 3, '2023-01-01 07:00:00', '2023-01-01 15:00:00'),
(4, 'Receptionist', 4, '2023-01-01 10:00:00', '2023-01-01 18:00:00'),
(5, 'Receptionist', 5, '2023-01-01 08:30:00', '2023-01-01 16:30:00'),
(6, 'Receptionist', 6, '2023-01-01 07:30:00', '2023-01-01 15:30:00'),
(7, 'Receptionist', 7, '2023-01-01 11:00:00', '2023-01-01 19:00:00'),
(8, 'Receptionist', 8, '2023-01-01 12:00:00', '2023-01-01 20:00:00'),
(9, 'Receptionist', 9, '2023-01-01 06:00:00', '2023-01-01 14:00:00'),
(10, 'Receptionist', 10, '2023-01-01 13:00:00', '2023-01-01 21:00:00');
GO

-- Insert 10 Nurses
INSERT INTO Nurse (EmployeeID, EmployeeType, ShiftStartTime, ShiftEndTime) VALUES
(11, 'Nurse', '2023-01-01 07:00:00', '2023-01-01 15:00:00'),
(12, 'Nurse', '2023-01-01 15:00:00', '2023-01-01 23:00:00'),
(13, 'Nurse', '2023-01-01 08:00:00', '2023-01-01 16:00:00'),
(14, 'Nurse', '2023-01-01 16:00:00', '2023-01-01 00:00:00'),
(15, 'Nurse', '2023-01-01 09:00:00', '2023-01-01 17:00:00'),
(16, 'Nurse', '2023-01-01 17:00:00', '2023-01-02 01:00:00'),
(17, 'Nurse', '2023-01-01 10:00:00', '2023-01-01 18:00:00'),
(18, 'Nurse', '2023-01-01 18:00:00', '2023-01-02 02:00:00'),
(19, 'Nurse', '2023-01-01 11:00:00', '2023-01-01 19:00:00'),
(20, 'Nurse', '2023-01-01 19:00:00', '2023-01-02 03:00:00');
GO

-- Insert 10 Doctors
INSERT INTO Doctor (EmployeeID, EmployeeType, Availability, Phone, Email) VALUES
(21, 'Doctor', 'Full-time', '555-1001', 'doc1@example.com'),
(22, 'Doctor', 'Part-time', '555-1002', 'doc2@example.com'),
(23, 'Doctor', 'Full-time', '555-1003', 'doc3@example.com'),
(24, 'Doctor', 'On-call', '555-1004', 'doc4@example.com'),
(25, 'Doctor', 'Full-time', '555-1005', 'doc5@example.com'),
(26, 'Doctor', 'Part-time', '555-1006', 'doc6@example.com'),
(27, 'Doctor', 'On-call', '555-1007', 'doc7@example.com'),
(28, 'Doctor', 'Full-time', '555-1008', 'doc8@example.com'),
(29, 'Doctor', 'Part-time', '555-1009', 'doc9@example.com'),
(30, 'Doctor', 'Full-time', '555-1010', 'doc10@example.com');
GO
```

```
-- Insert 20 Payment Methods (10 Direct, 10 Insurar
INSERT INTO PaymentMethod (Type, Details) VALUES
('Direct', 'Credit Card'),
('Direct', 'Debit Card'),
('Direct', 'Cash'),
('Direct', 'Check'),
('Direct', 'Online Transfer'),
('Direct', 'Mobile Wallet'),
('Direct', 'Bank Draft'),
('Direct', 'Money Order'),
('Direct', 'Cryptocurrency'),
('Direct', 'Gift Card'),
('Insurance', 'HealthPlus Inc.'),
('Insurance', 'MediCare Corp.'),
('Insurance', 'SecureHealth'),
('Insurance', 'Family Shield'),
('Insurance', 'Total Coverage'),
('Insurance', 'Wellness Partners'),
('Insurance', 'Global Health'),
('Insurance', 'Prime Care'),
('Insurance', 'United Health'),
('Insurance', 'LifeGuard');
GO
```

STORED PROCEDURES

1. sp_AdmitPatient

- Admits a patient to the hospital by assigning them a room and a doctor while ensuring the room is available. It also updates the room status to 'Occupied'.
- *Automates and validates the inpatient admission process with transaction handling.*

```
CREATE PROCEDURE sp_AdmitPatient
    @PatientID INT,
    @RoomID INT,
    @DoctorID INT,
    @AdmissionDate DATETIME,
    @Message NVARCHAR(255) OUTPUT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        IF EXISTS (SELECT 1 FROM Room WHERE RoomID = @RoomID AND Status = 'Available')
        BEGIN
            INSERT INTO Inpatient (PatientID, PatientType, RoomID, DoctorID, AdmissionDate)
            VALUES (@PatientID, 'Inpatient', @RoomID, @DoctorID, @AdmissionDate);

            UPDATE Room SET Status = 'Occupied' WHERE RoomID = @RoomID;

            SET @Message = 'Patient admitted successfully.';
            COMMIT TRANSACTION;
        END
        ELSE
        BEGIN
            SET @Message = 'Room is not available.';
            ROLLBACK TRANSACTION;
        END
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        SET @Message = 'Error during admission: ' + ERROR_MESSAGE();
    END CATCH
END;
GO
```

2. sp_GetDoctorAppointments

- Fetches all appointments for a given doctor based on the appointment status (e.g., Scheduled, Completed). Returns the total number of such appointments.
- *Helps doctors or staff quickly view and count active or specific-status appointments.*

```
CREATE PROCEDURE sp_GetDoctorAppointments
    @DoctorID INT,
    @Status VARCHAR(50),
    @AppointmentCount INT OUTPUT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        SELECT
            AppointmentID, PatientID, Date, AppointmentTime, Reason, Status
        FROM
            Appointment
        WHERE
            DoctorID = @DoctorID AND Status = @Status;

        SELECT @AppointmentCount = COUNT(*)
        FROM Appointment
        WHERE DoctorID = @DoctorID AND Status = @Status;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        SET @AppointmentCount = -1;
    END CATCH
END;
GO
```

USER DEFINED FUNCTIONS

1. fn_GetFullPatientName

- Returns the full display name of a patient in the format [ID] Name, making it easy to identify patients in views and dashboards.
- *Improves readability in reports and helps staff quickly recognize patients.*

```sql
CREATE FUNCTION fn_GetFullPatientName
(
    @PatientID INT
)
RETURNS VARCHAR(300)
AS
BEGIN
    DECLARE @FullName VARCHAR(300)

    SELECT @FullName = '[' + CAST(PatientID AS VARCHAR) + '] ' + Name
    FROM Patient
    WHERE PatientID = @PatientID

    RETURN @FullName
END;

SELECT
    PatientID,
    dbo.fn_GetFullPatientName(PatientID) AS FullName,
    Age,
    dbo.fn_ClassifyPatientAgeGroup(Age) AS AgeGroup,
    dbo.fn_CalculateTotalBill(PatientID) AS TotalBill
FROM Patient
ORDER BY PatientID;
```

2. fn_CalculateTotalBill

- Calculates the total bill for a patient by summing up their service charges, medication charges, room charges, and consultation fees.
- *Used in billing summaries to provide quick and accurate cost insights per patient.*

```sql
Go
CREATE FUNCTION fn_CalculateTotalBill
(
    @PatientID INT
)
RETURNS DECIMAL(10,2)
AS
BEGIN
    RETURN (
        SELECT
            SUM(ISNULL(ServiceCharges, 0) +
                ISNULL(MedicationCharges, 0) +
                ISNULL(RoomCharges, 0) +
                ISNULL(ConsultationFee, 0))
        FROM Billing
        WHERE PatientID = @PatientID
        GROUP BY PatientID
    );
END;

SELECT
    PatientID,
    dbo.fn_CalculateTotalBill(PatientID) AS TotalBill
FROM Patient;
```

VIEWS

1. vw_CurrentInpatients

- Displays patients who are currently admitted (i.e., not discharged), along with their room and doctor details.
- *Useful for front desk, nurse staff, or daily admission/discharge reports.*

```sql
CREATE VIEW vw_CurrentInpatients AS
SELECT
    ip.PatientID,
    p.Name AS PatientName,
    ip.AdmissionDate,
    r.RoomID,
    r.RoomType,
    d.EmployeeID AS DoctorID,
    d.Email AS DoctorEmail
FROM Inpatient ip
JOIN Patient p ON ip.PatientID = p.PatientID
JOIN Room r ON ip.RoomID = r.RoomID
JOIN Doctor d ON ip.DoctorID = d.EmployeeID
WHERE ip.DischargeDate IS NULL;

SELECT * FROM vw_CurrentInpatients;
```

	PatientID	PatientName	AdmissionDate	RoomID	RoomType	DoctorID	DoctorEmail
1	5	Patient E	2023-05-01 00:00:00.000	5	Single	25	doc5@example.com
2	6	Patient F	2023-06-01 00:00:00.000	6	Double	26	doc6@example.com
3	7	Patient G	2023-07-01 00:00:00.000	7	ICU	27	doc7@example.com
4	8	Patient H	2023-08-01 00:00:00.000	8	Emergency	28	doc8@example.com
5	9	Patient I	2023-09-01 00:00:00.000	9	Single	29	doc9@example.com
6	10	Patient J	2023-10-01 00:00:00.000	10	Double	30	doc10@example.c…
7	11	Patient K	2025-03-30 00:00:00.000	6	Double	24	doc4@example.com
8	20	Patient T	2025-03-29 00:00:00.000	1	Single	21	doc1@example.com

2. vw_DoctorAppointments

- Lists all scheduled appointments for doctors, including patient names, timing, and status.
- *Helps doctors and admin staff see upcoming visits and plan accordingly.*

```sql
CREATE VIEW vw_DoctorAppointments AS
SELECT
    d.EmployeeID AS DoctorID,
    d.Email AS DoctorEmail,
    p.PatientID,
    p.Name AS PatientName,
    a.Date,
    a.AppointmentTime,
    a.Status
FROM Appointment a
JOIN Doctor d ON a.DoctorID = d.EmployeeID
JOIN Patient p ON a.PatientID = p.PatientID
WHERE a.Status = 'Scheduled';

SELECT * FROM vw_DoctorAppointments;
```

	DoctorID	DoctorEmail	PatientID	PatientName	Date	AppointmentTime	Status
1	21	doc1@example.com	11	Patient K	2024-01-01	09:00:00	Scheduled
2	22	doc2@example.com	12	Patient L	2024-02-01	10:00:00	Scheduled
3	23	doc3@example.com	13	Patient M	2024-03-01	11:00:00	Scheduled
4	24	doc4@example.com	14	Patient N	2024-04-01	14:00:00	Scheduled
5	25	doc5@example.com	15	Patient O	2024-05-01	15:00:00	Scheduled
6	26	doc6@example.com	16	Patient P	2024-06-01	16:00:00	Scheduled
7	27	doc7@example.com	17	Patient Q	2024-07-01	17:00:00	Scheduled
8	28	doc8@example.com	18	Patient R	2024-08-01	18:00:00	Scheduled

VIEWS

3. vw_PatientBillingSummary

- Shows each patient's name, age group, and total bill using UDFs.
- *Provides a ready-to-use billing overview for finance and insurance processing.*

```sql
CREATE VIEW vw_PatientBillingSummary AS
SELECT
    p.PatientID,
    p.Name AS PatientName,
    p.Age,
    dbo.fn_ClassifyPatientAgeGroup(p.Age) AS AgeGroup,
    dbo.fn_CalculateTotalBill(p.PatientID) AS TotalBill
FROM Patient p;

SELECT * FROM vw_PatientBillingSummary;
```

	PatientID	PatientName	Age	AgeGroup	TotalBill
1	1	Patient A	45	Adult	750.00
2	2	Patient B	32	Adult	960.00
3	3	Patient C	28	Adult	1170.00
4	4	Patient D	50	Adult	1380.00
5	5	Patient E	60	Senior	1590.00
6	6	Patient F	22	Adult	1800.00
7	7	Patient G	38	Adult	2010.00
8	8	Patient H	41	Adult	2220.00

4. vw_PatientMedicationDetails

- Displays all prescriptions, medicines, dosages, and doctors associated with a patient.
- *Assists pharmacy, nurses, and doctors in tracking ongoing treatments and prescriptions.*

```sql
CREATE VIEW vw_PatientMedicationDetails AS
SELECT
    p.PatientID,
    p.Name AS PatientName,
    d.EmployeeID AS DoctorID,
    d.Email AS DoctorEmail,
    pr.PrescriptionDate,
    m.Name AS MedicationName,
    pm.Dosage,
    pm.Frequency
FROM Prescription pr
JOIN Patient p ON pr.PatientID = p.PatientID
JOIN Doctor d ON pr.DoctorID = d.EmployeeID
JOIN PrescriptionMedication pm ON pr.PrescriptionID = pm.PrescriptionID
JOIN Medication m ON pm.MedicationID = m.MedicationID;


SELECT * FROM vw_PatientMedicationDetails;
```

	PatientID	PatientName	DoctorID	DoctorEmail	PrescriptionDate	MedicationName	Dosage	Frequency
1	1	Patient A	21	doc1@example.com	2023-01-01	PainAway	500mg	Once Daily
2	2	Patient B	22	doc2@example.com	2023-02-01	ColdFix	10ml	Twice Daily
3	3	Patient C	23	doc3@example.com	2023-03-01	AlleriStop	1 Tablet	Every 6 Hours
4	4	Patient D	24	doc4@example.com	2023-04-01	DigestEase	2 Capsules	With Meals
5	5	Patient E	25	doc5@example.com	2023-05-01	FlexiJoint	Apply Thinly	As Needed
6	6	Patient F	26	doc6@example.com	2023-06-01	SleepWell	1 Spray	Nightly
7	7	Patient G	27	doc7@example.com	2023-07-01	ImmuneBoost	1 Drop	Morning
8	8	Patient H	28	doc8@example.com	2023-08-01	HeartGuard	2 Puffs	Every 12 Hours

TRIGGERS

1. trg_CascadeDeletePatient

- **Automatically deletes all related records (appointments, prescriptions, billing, inpatient/outpatient info, etc.) when a patient is deleted — while also updating room status if applicable.**
- ***Ensures clean deletion of patient data across the system and prevents orphan records.***

```sql
CREATE TRIGGER trg_CascadeDeletePatient
ON Patient
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- Only update rooms for inpatients
    UPDATE Room
    SET Status = 'Available'
    FROM Room
    INNER JOIN Inpatient ON Room.RoomID = Inpatient.RoomID
    INNER JOIN deleted ON Inpatient.PatientID = deleted.PatientID
    WHERE deleted.PatientType = 'Inpatient';

    -- Delete PrescriptionMedication links only if they exist
    DELETE pm
    FROM PrescriptionMedication pm
    INNER JOIN Prescription p ON pm.PrescriptionID = p.PrescriptionID
    INNER JOIN deleted d ON p.PatientID = d.PatientID;

    -- Delete Prescriptions if they exist
    DELETE FROM Prescription WHERE PatientID IN (SELECT PatientID FROM deleted);

    -- Delete other related records
    DELETE FROM MedicalRecord WHERE PatientID IN (SELECT PatientID FROM deleted);
    DELETE FROM Appointment WHERE PatientID IN (SELECT PatientID FROM deleted);
    DELETE FROM CareAssignment WHERE PatientID IN (SELECT PatientID FROM deleted);
    DELETE FROM Billing WHERE PatientID IN (SELECT PatientID FROM deleted);

    -- Delete Inpatient or Outpatient records conditionally
    DECLARE @PatientCursor CURSOR;
    DECLARE @PatientID INT;
    DECLARE @PatientType VARCHAR(20);

    SET @PatientCursor = CURSOR FOR
        SELECT PatientID, PatientType
        FROM deleted;

    OPEN @PatientCursor;
    FETCH NEXT FROM @PatientCursor INTO @PatientID, @PatientType;
```

2. trg_UpdateRoomStatusOnDischarge

- **Fires after an inpatient's discharge date is updated, and automatically sets their room status back to 'Available'.**
- ***Maintains real-time room availability and removes the need for manual updates.***

```sql
CREATE TRIGGER trg_UpdateRoomStatusOnDischarge
ON Inpatient
AFTER UPDATE
AS
BEGIN

    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN deleted d ON i.PatientID = d.PatientID
        WHERE i.DischargeDate IS NOT NULL AND d.DischargeDate IS NULL
    )
    BEGIN
        UPDATE Room
        SET Status = 'Available'
        WHERE RoomID IN (
            SELECT i.RoomID
            FROM inserted i
            JOIN deleted d ON i.PatientID = d.PatientID
            WHERE i.DischargeDate IS NOT NULL AND d.DischargeDate IS NULL
        );
    END
END;

-- Testing trigger

UPDATE Inpatient
SET DischargeDate = '2025-04-01'
WHERE PatientID = 3;

SELECT RoomID FROM Inpatient WHERE PatientID = 3;
SELECT Status FROM Room WHERE RoomID = 3;
```

	PatientID	RoomID	RoomStatusAfter	AdmissionDate	DischargeDate
1	1	1	Available	2023-01-01 00:00:00.000	2023-04-11 00:00:00.000
2	6	6	Available	2023-06-01 00:00:00.000	2023-04-11 00:00:00.000

ENCRYTION

- **Columns Encrypted:**
  - `Patient.Email → EncryptedEmail`
  - `Patient.PhoneNumber → EncryptedPhoneNumber`
  - `Employee.ContactInfo → EncryptedContactInfo`
- **Encryption Method Used:**

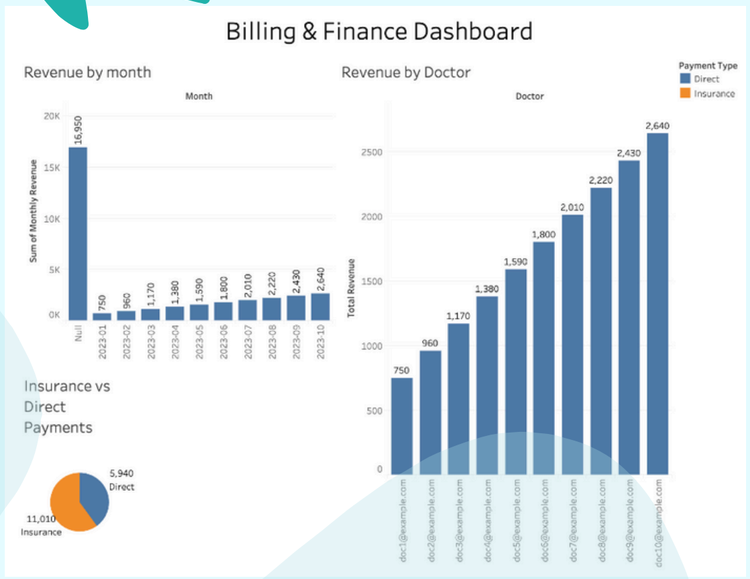
AES-256 encryption using a symmetric key, protected by a digital certificate.

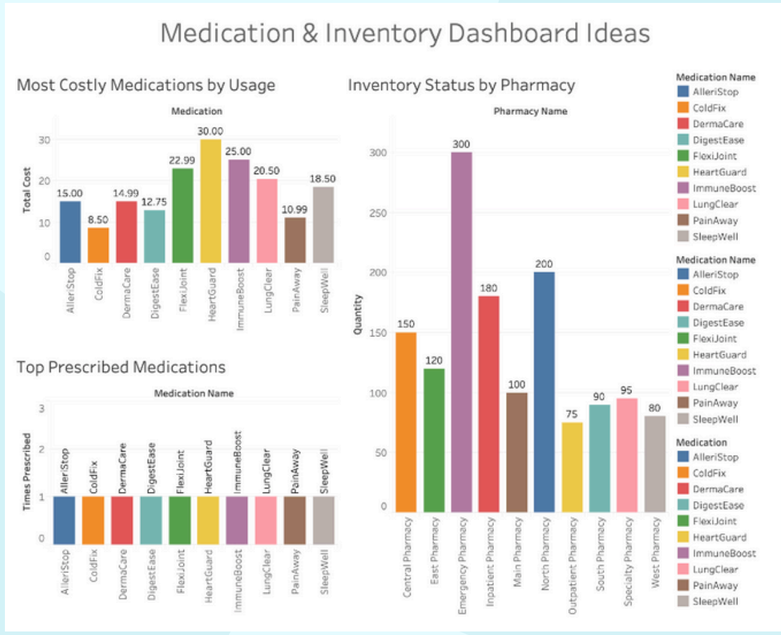
- **Steps Taken:**

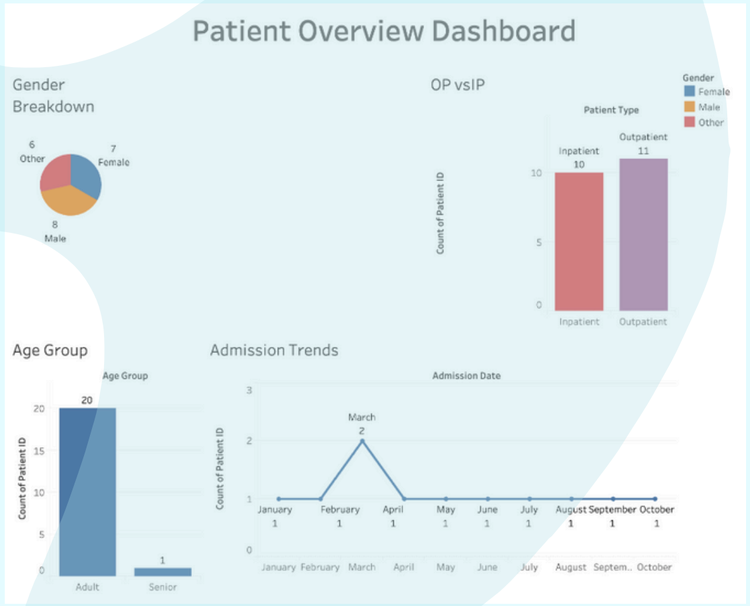
a. A master key was created if not already existing.
b. A certificate (HospitalCert) was created to secure the symmetric key.
c. A symmetric key (HospitalSymmetricKey) was generated with AES_256 encryption algorithm.
d. The symmetric key was opened for use, and actual data was encrypted using ENCRYPTBYKEY().
e. Encrypted data was stored in separate VARBINARY(MAX) columns for secure storage.
f. For verification or access, decryption was done using DECRYPTBYKEY() when the key was open.

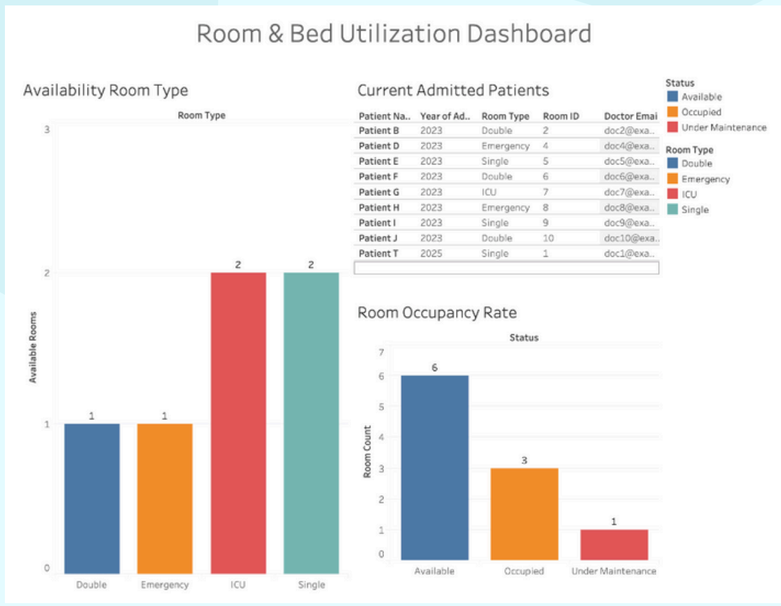
```sql
ALTER TABLE Employee
ADD EncryptedContactInfo VARBINARY(MAX);
GO

OPEN SYMMETRIC KEY HospitalSymmetricKey
DECRYPTION BY CERTIFICATE HospitalCert;

UPDATE Employee
SET EncryptedContactInfo = CASE
    WHEN ContactInfo IS NOT NULL
    THEN ENCRYPTBYKEY(KEY_GUID('HospitalSymmetricKey'), CAST(ContactInfo AS VARCHAR(255)))
    ELSE NULL
END;

CLOSE SYMMETRIC KEY HospitalSymmetricKey;
GO


-- [Optional] Decryption Test

OPEN SYMMETRIC KEY HospitalSymmetricKey
DECRYPTION BY CERTIFICATE HospitalCert;

SELECT
    EmployeeID,
    Name,
    ContactInfo,
    CONVERT(VARCHAR, DECRYPTBYKEY(EncryptedContactInfo)) AS DecryptedContactInfo
FROM Employee;

CLOSE SYMMETRIC KEY HospitalSymmetricKey;
```

TABLEAU DASHBOARD

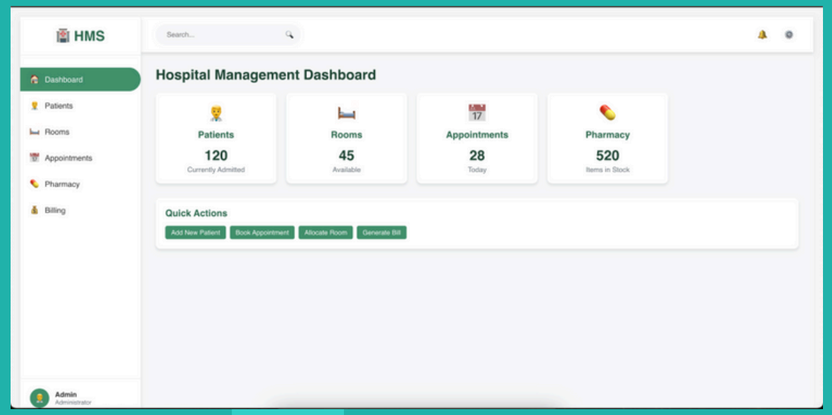


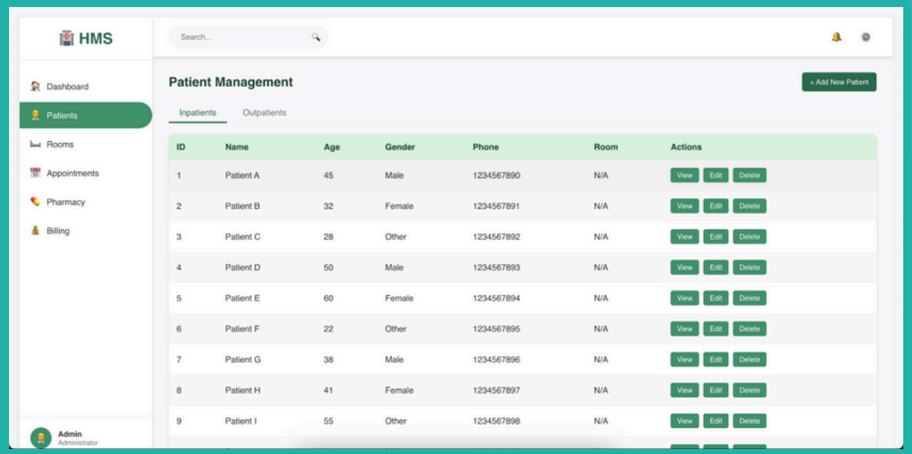


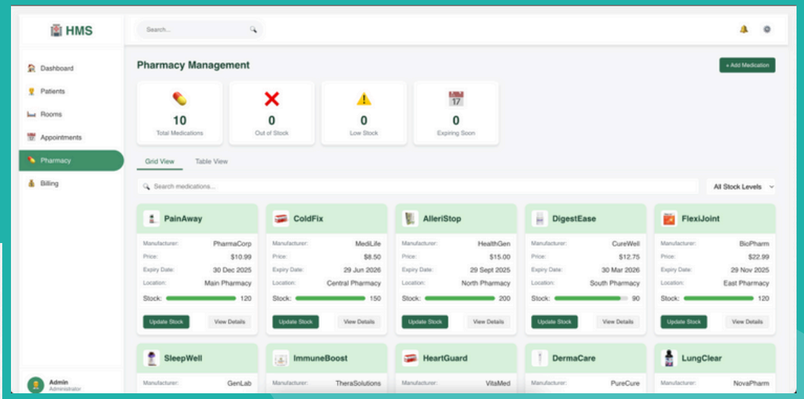


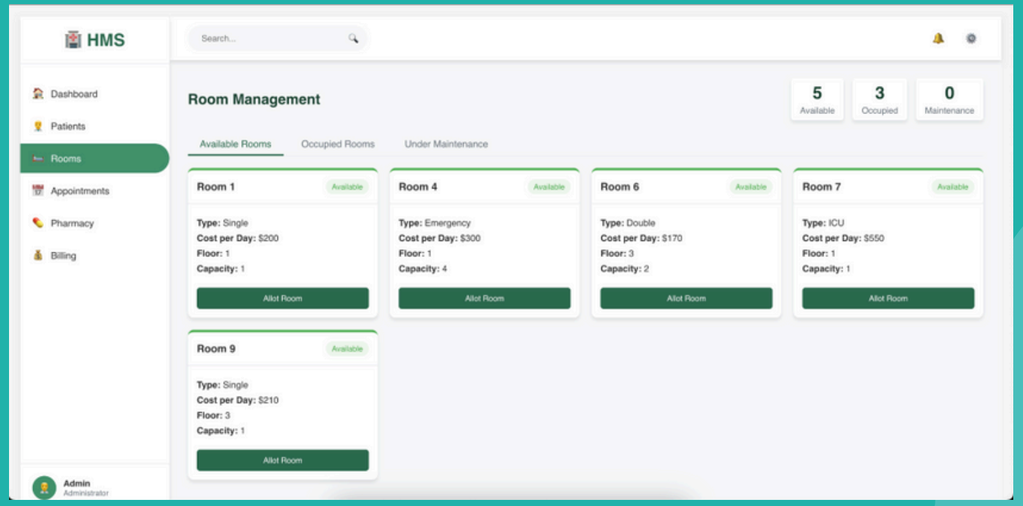


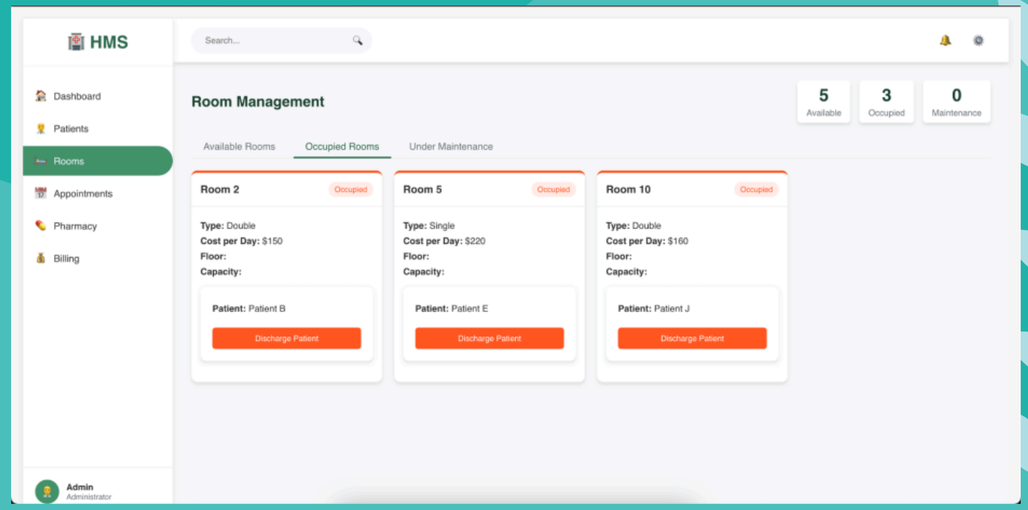
GUI - SCREENSHOT











Thank you