SQL - BASED DATABASE DESIGN & ANALYSIS

A project of Data Management for Analytics

CARE HOSPITAL DATABASE MANAGEMENT SYSTEM

By:

Uzma Naeem

TABLE OF CONTENTS

1. **INTRODUCTION** 3
2. **ENTITIES & ATTRIBUTIES** 4
3. **DOMAIN CONSTRAINTS** 7
4. **BUSINESS RULES** 9
5. **ENTITY RELATIONSHIP DIAGRAM OF CARE HOSPITAL** 10
6. **3-NF RELATIONAL MODEL OF CARE HOSPITAL** 11
7. **SQL QUERIES** 12
8. **CONCLUSION** 29

INTRODUCTION

Name of the organization: Care Hospital

Sector: Health Care

Description and Focus:

Care Hospital is a fictional health care organization created for the purpose of this project. The database designed for Care Hospital aims to manage and organize various aspects of the healthcare facility, including **patient care**, **communication** among **healthcare professionals**, and **secure storage of medical records**. The design reflects the increasing relevance of technology in healthcare, emphasizing the importance of data accuracy and consistency.

Why did you pick this organization?

By focusing on the below key areas, the Care Hospital database aims to enhance the overall efficiency, coordination, and quality of healthcare services provided to patients. It supports the hospital's commitment to delivering personalized and effective medical care.

In health care sector the data accuracy and consistency play a very important role and designing a database for health care sector is complex and challenging due to the diverse and interconnected nature of data.

This database also reflects the relevance and growing importance of technology in the health care sector.

Data Acquisition Plan:

As our hospital is a fictional organization, data for this project will be simulated by observing similar hospital structure and operations. We will synthesize the data to populate our database, ensuring access to all necessary information without any privacy concerns.

ENTITIES & ATTRIBUTIES

1. Patients Table

Description:

- This table serves as a repository for patient information within the medical system. It includes essential details such as PatientID, which acts as a unique identifier for each patient.
- The FirstName and LastName fields capture the patient's name, while DateOfBirth records the patient's birth date.
- Gender denotes the patient's gender identity, and ContactNumber stores their contact information for communication purposes.

Attributes:

- PatientID (Primary Key)
- FirstName
- LastName
- DateOfBirth
- Gender
- ContactNumber

2. Departments Table

Description:

- This table contains essential information about different departments within the medical facility.
- DepartmentID serves as a unique identifier for each department entry.
- DepartmentName specifies the name of the department, while DoctorID represents the doctor who leads or heads the department.
- Location denotes the physical location of the department, and ContactNumber stores contact information for departmental inquiries.

Attributes:

- DepartmentID (Primary Key)
- DepartmentName
- Location
- ContactNumber

- StaffCount

3. Doctors Table

Description:

- The Doctors table is designed to store comprehensive information about medical practitioners within the healthcare facility.
- DoctorID is a unique identifier for each doctor.
- The FirstName and LastName fields contain the doctor's name, while Specialty describes their area of expertise or medical specialization.
- ContactNumber and Email fields store the doctor's contact details, facilitating communication between medical staff and patients.

Attributes:

- DoctorID (Primary Key)
- DepartmentID
- FirstName
- LastName
- Specialty
- ContactNumber
- Email

4. Appointments Table

Description:

- This table maintains records of appointments scheduled between patients and doctors.
- AppointmentID serves as a unique identifier for each appointment entry.
- AppointmentDate and AppointmentTime fields specify when the appointment is scheduled, and Status indicates the current status of the appointment (e.g., scheduled, completed, canceled).

Attributes:

- AppointmentID (Primary Key)
- PatientID
- DoctorID
- AppointmentDate
- AppointmentTime
- Status (e.g., scheduled, completed, canceled)

5. MedicalRecords Table

Description:

- The MedicalRecords table is crucial for storing detailed medical information about patients' visits to healthcare providers.
- RecordID uniquely identifies each medical record entry.
- DateOfVisit records the date when the patient visited the doctor, while Diagnosis and Prescription fields contain information about the patient's diagnosis and prescribed treatment.

Attributes:

- RecordID (Primary Key)
- PatientID
- DateOfVisit
- Diagnosis
- Prescription

6. Inventory Table

Description:

- The Inventory table is responsible for managing the stock of medical supplies and equipment within each department of the healthcare facility.
- ItemID serves as a unique identifier for each inventory item.
- ItemName describes the name or description of the inventory item, while Quantity specifies the quantity or stock level of the item.

Attributes:

- ItemID (Primary Key)
- ItemName
- Quantity
- ExpiryDate
- UsageHistory

DOMAIN CONSTRAINTS

1. Patients Table

PatientID: Primary Key, INT, unique identifier for each patient.

FirstName: VARCHAR(50), must not be null.

LastName: VARCHAR(50), must not be null.

DateOfBirth: DATE, must not be null, used to calculate age and for medical records.

Gender: CHAR(1), 'M' for male, 'F' for female, or 'O' for other; must not be null.

ContactNumber: VARCHAR(15), must follow a valid phone number format, can be null if not provided.

2. Departments Table

DepartmentID: Primary Key, INT, unique identifier for each department.

DepartmentName: VARCHAR(100), must not be null.

Location: VARCHAR(255), physical location of the department, must not be null.

ContactNumber: VARCHAR(15), must follow a valid phone number format, can be null if not provided.

StaffCount : VARCHAR(15), must not be null

3. Doctors Table

DoctorID: Primary Key, INT, unique identifier for each doctor.

DepartmentID : VARCHAR(50), must not be null.

FirstName: VARCHAR(50), must not be null.

LastName: VARCHAR(50), must not be null.

Specialty: VARCHAR(100), represents the doctor's medical specialty, must not be null.

ContactNumber: VARCHAR(15), must follow a valid phone number format, can be null if not provided.

Email: VARCHAR(25), must follow a valid email format, must not be null.

4. Appointments Table

AppointmentID: Primary Key, INT, unique identifier for each appointment.

PatientID: INT, referencing Patients, must not be null.

DoctorID: INT, referencing Doctors, must not be null.

AppointmentDate: DATE, must not be null.

AppointmentTime: TIME, must not be null.

Status: ENUM('scheduled', 'completed', 'canceled'), status of the appointment, must not be null.

5. MedicalRecords Table

RecordID: Primary Key, INT, unique identifier for each medical record.

PatientID: INT, referencing Patients, must not be null.

DoctorID: INT, referencing Doctors, must not be null.

DateOfVisit: DATE, must not be null.

Diagnosis: TEXT, detailed diagnosis, can be null.

Prescription: TEXT, details of prescribed medication, can be null.

6. Inventory Table

ItemID: Primary Key, INT, unique identifier for each item.
ItemName: VARCHAR(100), must not be null.

Quantity: INT, must be greater than or equal to 0, represents the current stock.

ExpiryDate: DATE, can be null if the item does not expire.

DepartmentID: INT, referencing Departments, specifies where the item is stored, must not be null.

UsageHistory: VARCHAR(50), specifies history of when and where inventory items are used.

BUSINESS RULES

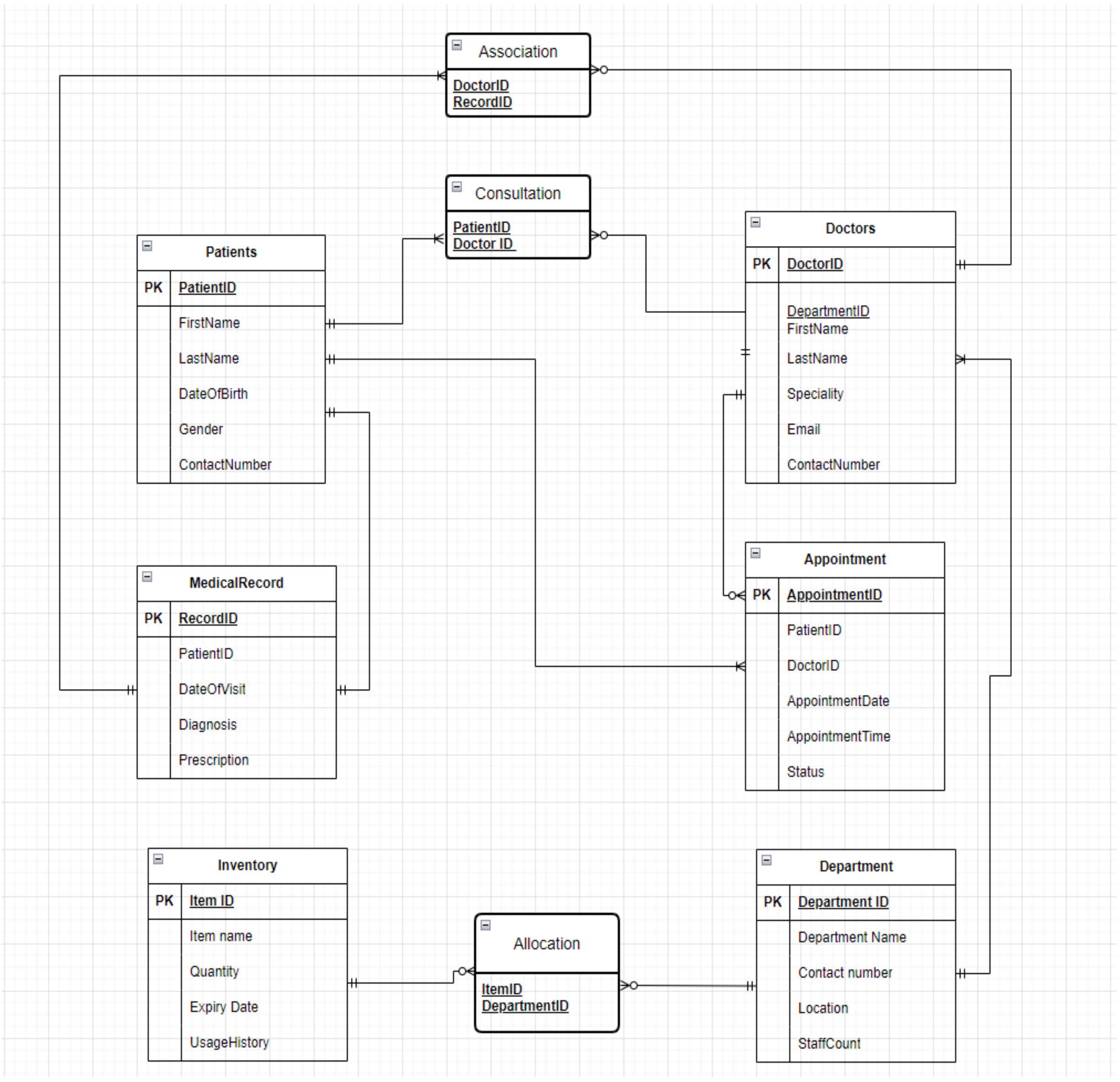
1. Each patient **must** be assigned to at least one doctor.
2. Each doctor **can** have more than one patient.
3. Each patient **must** have one or more appointments.
4. Each appointment **must** be associated to one patient and one doctor.
5. Each doctor **can** have one or more appointments.
6. Each medical record **must** be linked with only one patient.
7. Each patient **must** have a single medical record number.
8. Each medical record **must** have at least one doctor assigned.
9. Each doctor **can** be associated with one or more medical records.
10. Each department **must** have more than one doctor.
11. Each doctor **must** be assigned to only one department.
12. Each department **must** have access to at least one inventory.
13. Each inventory **can** be assigned to one or more departments.

Cardinalities Involving Associative Entities:

Many to Many relationships between entities:

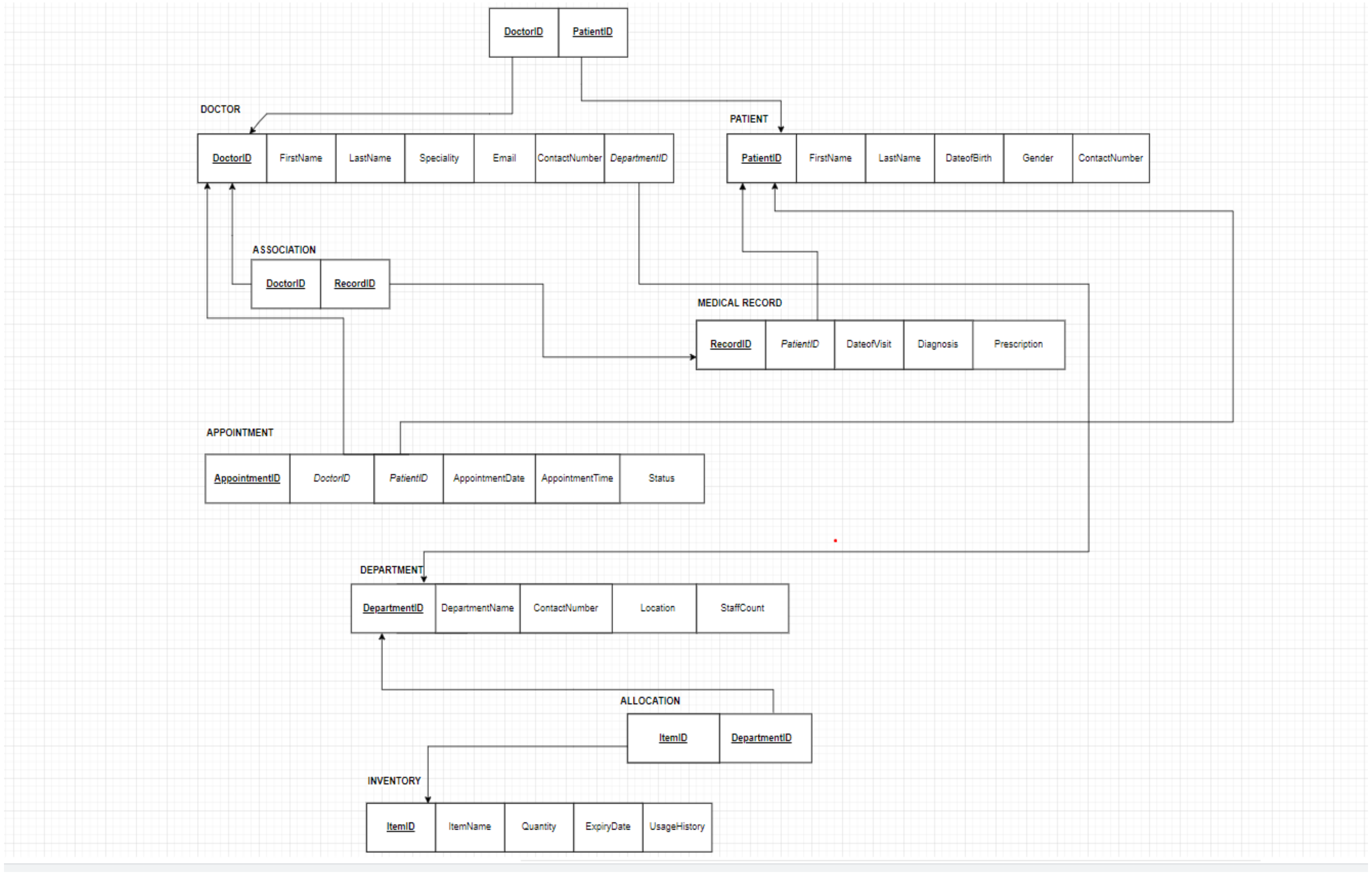
1. Doctor – Patient
2. Doctor – Medical Record
3. Inventory – Department

ENTITY RELATIONSHIP DIAGRAM OF CARE HOSPITAL



Note: This is the before and after of the EER diagram as the associative entities are already incorporated in it.

3-NF RELATIONAL MODEL OF CARE HOSPITAL



Key:

1. **<u>Bold Underlined – Primary Keys</u>**
2. *Italicized – Foreign Keys*
3. Rest – Normal Attributes

SQL QUERIES

1. Creation Of Database & Tables:

Create the database

```
CREATE DATABASE IF NOT EXISTS care_hospital;
```

Use the database

```
USE care_hospital;
```

Create Patients Table (1)

```
CREATE TABLE Patients (
   PatientID INT AUTO_INCREMENT PRIMARY KEY,
   FirstName VARCHAR(50) NOT NULL,
   LastName VARCHAR(50) NOT NULL,
   DateOfBirth DATE NOT NULL,
   Gender CHAR(1) NOT NULL,
   ContactNumber VARCHAR (50),
   CONSTRAINT chk_gender CHECK (Gender IN ('M', 'F', 'O'))
);
```

Create Departments Table (2)

```
CREATE TABLE Departments (
   DepartmentID INT AUTO_INCREMENT PRIMARY KEY,
   DepartmentName VARCHAR(100) NOT NULL,
   Location VARCHAR(255) NOT NULL,
   ContactNumber VARCHAR (50),
   StaffCount INT NOT NULL
);
```

Create Doctors Table (3)

```
CREATE TABLE Doctors (
   DoctorID INT AUTO_INCREMENT PRIMARY KEY,
   DepartmentID INT NOT NULL,
   FirstName VARCHAR(50) NOT NULL,
   LastName VARCHAR(50) NOT NULL,
   Specialty VARCHAR(100) NOT NULL,
   ContactNumber VARCHAR (50),
   Email VARCHAR(30) NOT NULL,
   CONSTRAINT chk_email CHECK (Email LIKE '%_@__%.__%'),
   CONSTRAINT Doctors_FK1 FOREIGN KEY (DepartmentID)
REFERENCES Departments(DepartmentID)
);
```

Create Appointments Table (4)

```
CREATE TABLE Appointments (
   AppointmentID INT AUTO_INCREMENT PRIMARY KEY,
   PatientID INT NOT NULL,
   DoctorID INT NOT NULL,
   AppointmentDate DATE NOT NULL,
   AppointmentTime TIME NOT NULL,
   Status ENUM('scheduled', 'completed', 'canceled') NOT NULL,
   CONSTRAINT Appointments_FK1 FOREIGN KEY (PatientID)
REFERENCES Patients(PatientID),
   CONSTRAINT Appointments_FK2 FOREIGN KEY (DoctorID)
REFERENCES Doctors(DoctorID)
);
```

Create MedicalRecords Table (5)

```
CREATE TABLE MedicalRecords (
   RecordID INT AUTO_INCREMENT PRIMARY KEY,
    PatientID INT NOT NULL,
    DoctorID INT NOT NULL,
    DateOfVisit DATE NOT NULL,
    Diagnosis TEXT,
    Prescription TEXT,
    CONSTRAINT MedicalRecords_FK1 FOREIGN KEY (PatientID)
REFERENCES Patients(PatientID),
    CONSTRAINT MedicalRecords_FK2 FOREIGN KEY (DoctorID)
REFERENCES Doctors(DoctorID)
);
```

Create Inventory Table (6)

```
CREATE TABLE Inventory (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    ItemName VARCHAR(100) NOT NULL,
    Quantity INT CHECK (Quantity >= 0),
    ExpiryDate DATE,
    DepartmentID INT NOT NULL,
    UsageHistory VARCHAR(50),
    CONSTRAINT Inventory_FK1 FOREIGN KEY (DepartmentID)
REFERENCES Departments(DepartmentID)
);
```

2. Loading Data Into Tables:

Patients Table (1)

```
INSERT INTO Patients (PatientID, FirstName, LastName, DateOfBirth, Gender,
ContactNumber)
VALUES
    (101, 'John', 'Smith', '1980-05-15', 'M', '123-456-7890'),
    (102, 'Emily', 'Johnson', '1992-09-20', 'F', '456-789-0123'),
    (103, 'Michael', 'Brown', '1975-03-10', 'M', '789-012-3456'),
    (104, 'Sarah', 'Davis', '1988-11-02', 'F', '234-567-8901'),
    (105, 'David', 'Martinez', '1965-07-25', 'M', '567-890-1234');
```

Departments Table (2)

```
INSERT INTO Departments (DepartmentID, DepartmentName, Location,
ContactNumber, StaffCount)
VALUES
    (201, 'Cardiology', "Main Building, 3rd Floor", '123-456-7890', 15),
    (202, 'Pediatrics', "Children's Wing, 2nd Floor", '456-789-0123', 10),
    (203, 'Orthopedics', "East Wing, 1st Floor", NULL, 12),
    (204, 'Dermatology', "West Wing, 2nd Floor", '234-567-8901', 8),
    (205, 'Neurology', "North Wing, 4th Floor", '567-890-1234', 9);
```

Doctors Table (3)

```
INSERT INTO Doctors (DoctorID, DepartmentID, FirstName, LastName,
Specialty, ContactNumber, Email)
VALUES
    (301, 201, 'Robert', 'Johnson', 'Cardiologist', '123-456-7890',
'robert.johnson@hc.com'),
    (302, 202, 'Emily', 'Williams', 'Pediatrician', '456-789-0123',
'emily.williams@hc.com'),
    (303, 203, 'Michael', 'Davis', 'Orthopedic Surgeon', '789-012-3456',
'michael.davis@hc.com'),
    (304, 204, 'Sarah', 'Garcia', 'Dermatologist', '234-567-8901',
'sarah.garcia@hc.com'),
    (305, 205, 'David', 'Martinez', 'Neurologist', '567-890-1234',
'david.martinez@hc.com');
```

Appointments Table (4)

```
INSERT INTO Appointments (AppointmentID, PatientID, DoctorID,
AppointmentDate, AppointmentTime, Status)

VALUES

    (401, 101, 301, '2024-05-12', '10:00:00', 'scheduled'),

    (402, 102, 302, '2024-05-13', '11:00:00', 'scheduled'),

    (403, 103, 303, '2024-05-14', '12:00:00', 'completed'),

    (404, 104, 304, '2024-05-15', '13:00:00', 'canceled'),

    (405, 105, 305, '2024-05-16', '14:00:00', 'scheduled');
```

MedicalRecords Table (5)

```
INSERT INTO MedicalRecords (RecordID, PatientID, DoctorID, DateOfVisit,
Diagnosis, Prescription)

VALUES

    (501, 101, 301, '2024-05-12', 'Hypertension', 'Medication A'),

    (502, 102, 302, '2024-05-13', 'Common Cold', 'Rest and Fluids'),

    (503, 103, 303, '2024-05-14', 'Fractured Arm', 'Surgery recommended'),

    (504, 104, 304, '2024-05-15', 'Skin Allergy', 'Topical cream prescribed'),

    (505, 105, 305, '2024-05-16', 'Migraine', 'Pain relievers');
```

Inventory Table (6)

```
INSERT INTO Inventory (ItemID, ItemName, Quantity, ExpiryDate,
DepartmentID, UsageHistory)

VALUES

    (601, 'Bandages', 200, '2025-12-31', 203, 'Orthopedics'),

    (602, 'Aspirin', 500, '2023-06-30', 201, 'Cardiology'),

    (603, 'Antibiotics', 100, '2024-09-30', 202, 'Pediatrics'),

    (604, 'Anti-Allergics', 50, '2025-03-31', 204, 'Dermatology'),

    (605, 'Apomorphine', 300, '2024-12-31', 205, 'Neurology');
```

This Query No. A is to bring visible difference in number of appointments retrieved in descending order in Query No.1.

QUERY NO. A - INSERT

Add a new appointment for a doctor with DoctorID = 301

INSERT INTO Appointments (PatientID, DoctorID, AppointmentDate, AppointmentTime, Status)

VALUES (101, 301, '2024-05-17', '15:00:00', 'completed');

3. Retrieval Of Data Select Queries:

QUERY NO. 1 - SELECT, CONCAT, COUNT, INNER JOIN, GROUP BY, HAVING, ORDER BY

Get a count of appointments per doctor, only showing those doctors who have appointments, sort by appointment count in descending order:

Providing a count of appointments per doctor helps in analyzing workload and scheduling efficiency, which is crucial for operational planning and resource allocation.

SELECT

CONCAT (Doctors.FirstName, ' ' , Doctors.LastName) AS DoctorName,

COUNT(Appointments.AppointmentID) AS AppointmentCount

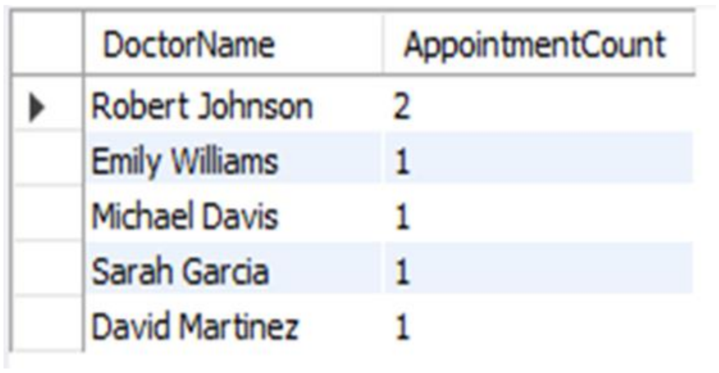
FROM Appointments

INNER JOIN Doctors ON Appointments.DoctorID = Doctors.DoctorID

GROUP BY DoctorName

HAVING AppointmentCount > 0

ORDER BY AppointmentCount DESC;



	DoctorName	AppointmentCount
▶	Robert Johnson	2
	Emily Williams	1
	Michael Davis	1
	Sarah Garcia	1
	David Martinez	1

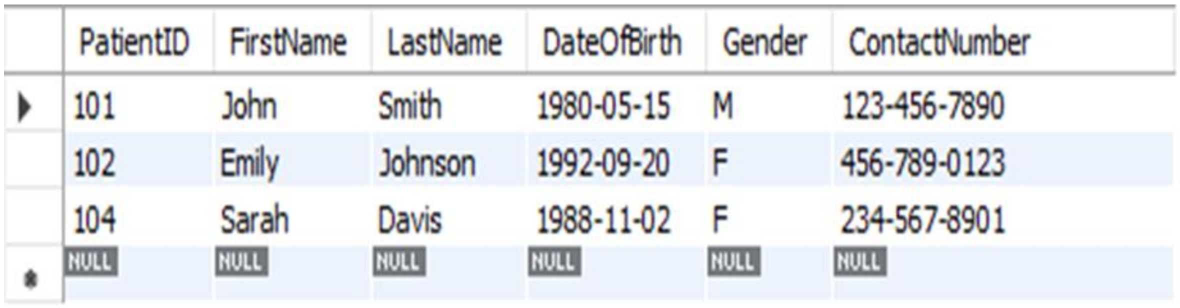
QUERY NO. 2: WHERE

Pick a patient who is born after 1980

Filtering patients born after 1980 can be useful for studies or treatments that target a specific demographic group.

SELECT * FROM Patients

WHERE DateOfBirth > '1980-01-01';



	PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber
▶	101	John	Smith	1980-05-15	M	123-456-7890
	102	Emily	Johnson	1992-09-20	F	456-789-0123
	104	Sarah	Davis	1988-11-02	F	234-567-8901
*	NULL	NULL	NULL	NULL	NULL	NULL

QUERY NO. 3 - SELECT, INNER JOIN

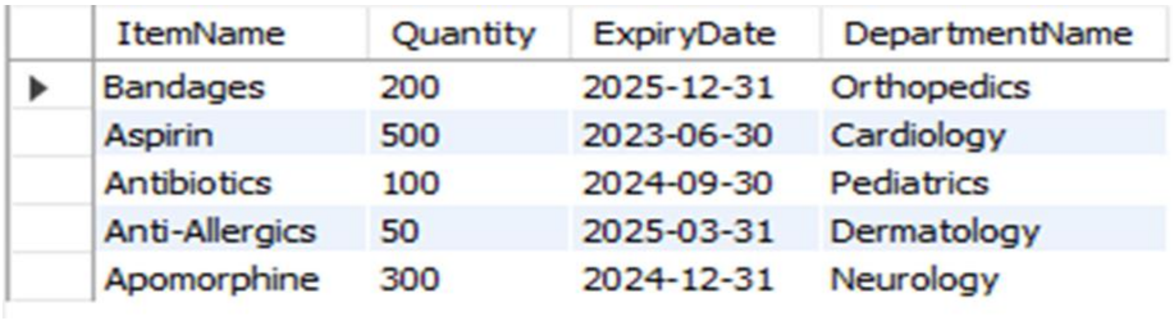
Get a list of items in the inventory along with their quantities and the department they belong to.

This query retrieves a list of inventory items along with their quantities and associated departments. It helps in understanding how resources are distributed across different parts of the hospital. By linking inventory items with departments, the hospital can enforce accountability for resource usage, ensuring that departments manage their supplies responsibly.

SELECT ItemName, Quantity, ExpiryDate, DepartmentName

FROM Inventory

INNER JOIN Departments ON Inventory.DepartmentID = Departments.DepartmentID;



	ItemName	Quantity	ExpiryDate	DepartmentName
▶	Bandages	200	2025-12-31	Orthopedics
	Aspirin	500	2023-06-30	Cardiology
	Antibiotics	100	2024-09-30	Pediatrics
	Anti-Allergics	50	2025-03-31	Dermatology
	Apomorphine	300	2024-12-31	Neurology

QUERY NO. 4 - CREATE, VIEW, SELECT, CONCAT, AS, WHERE, AND - MULTITABLE VIEW

Create a view that displays the patient's full name, appointment date and time, doctor's full name, and department name for all scheduled appointments.

Creating a view that consolidates scheduled appointments with relevant details enhances the usability of the database for daily operations and planning.

CREATE VIEW Scheduled_Appointments AS

SELECT CONCAT(Patients.FirstName, ' ' ,Patients.LastName) AS PatientName,

Appointments.AppointmentDate, Appointments.AppointmentTime,

CONCAT (Doctors.FirstName, ' ' ,Doctors.LastName) AS DoctorName, Departments.DepartmentName

FROM Patients,Appointments,Doctors,Departments

WHERE Departments.DepartmentID = Doctors.DepartmentID

AND Appointments.DoctorID = Doctors.DoctorID

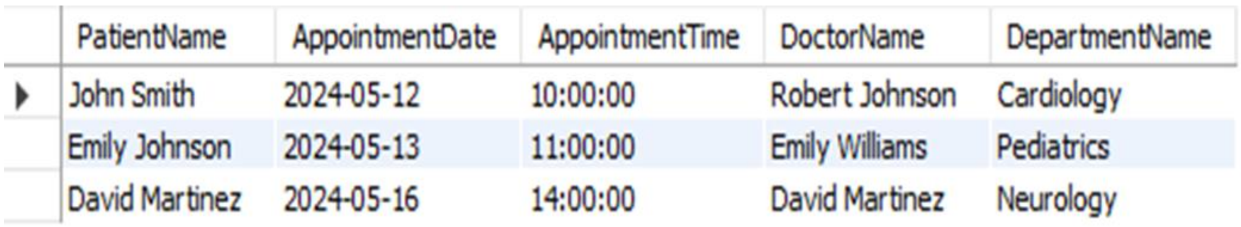
AND Appointments.PatientID = Patients.PatientID

AND Appointments.Status = 'scheduled';

Check Query No 4

Show the details of the view created in Query No. 4

Select * From Scheduled_Appointments;



	PatientName	AppointmentDate	AppointmentTime	DoctorName	DepartmentName
▶	John Smith	2024-05-12	10:00:00	Robert Johnson	Cardiology
	Emily Johnson	2024-05-13	11:00:00	Emily Williams	Pediatrics
	David Martinez	2024-05-16	14:00:00	David Martinez	Neurology

QUERY NO. 5: UPDATE, SET, WHERE, SELECT

Update an appointment of a patient using the appointment ID:

Updating an appointment's status to 'completed' allows for real-time tracking of patient appointments, essential for both operational management and historical records.

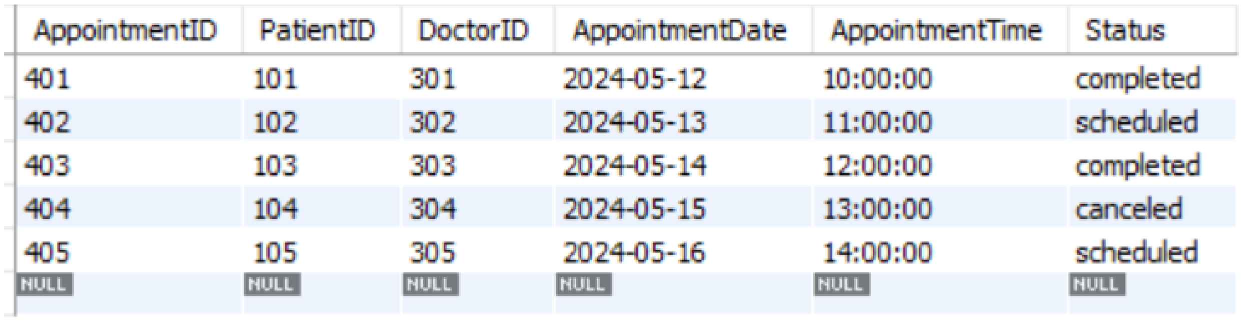
UPDATE Appointments

SET Status = 'completed'

WHERE AppointmentID = 401;

Check Query No.5

SELECT * FROM Appointments;



AppointmentID	PatientID	DoctorID	AppointmentDate	AppointmentTime	Status
401	101	301	2024-05-12	10:00:00	completed
402	102	302	2024-05-13	11:00:00	scheduled
403	103	303	2024-05-14	12:00:00	completed
404	104	304	2024-05-15	13:00:00	canceled
405	105	305	2024-05-16	14:00:00	scheduled
NULL	NULL	NULL	NULL	NULL	NULL

4. Different Function Queries:

QUERY NO 6: ALTER

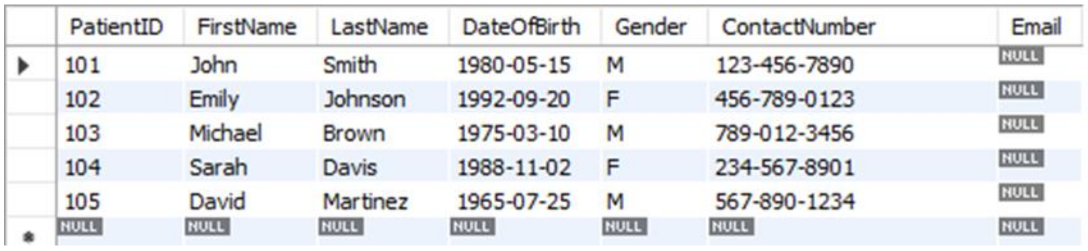
Alter Patients table to add a column:

Adding an email column to the Patients table allows the hospital to communicate electronically with patients, enhancing communication efficiency.

ALTER TABLE Patients ADD Column Email VARCHAR(255);

Check Query No.6

Select * From Patients;



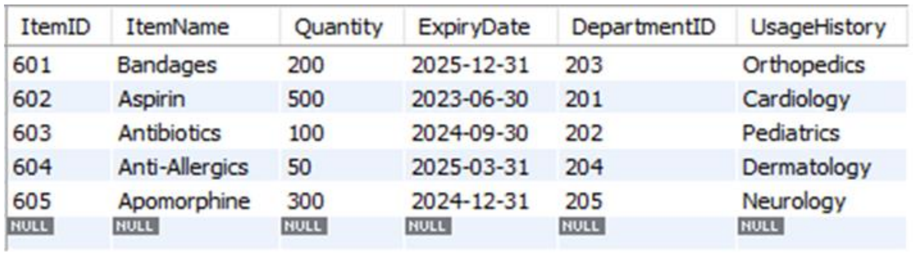
PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Email
101	John	Smith	1980-05-15	M	123-456-7890	NULL
102	Emily	Johnson	1992-09-20	F	456-789-0123	NULL
103	Michael	Brown	1975-03-10	M	789-012-3456	NULL
104	Sarah	Davis	1988-11-02	F	234-567-8901	NULL
105	David	Martinez	1965-07-25	M	567-890-1234	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

QUERY NO. 7 DROP

Drop an unused column:

Removing an unused column can streamline database operations, reducing complexity and improving performance.

Before Dropping



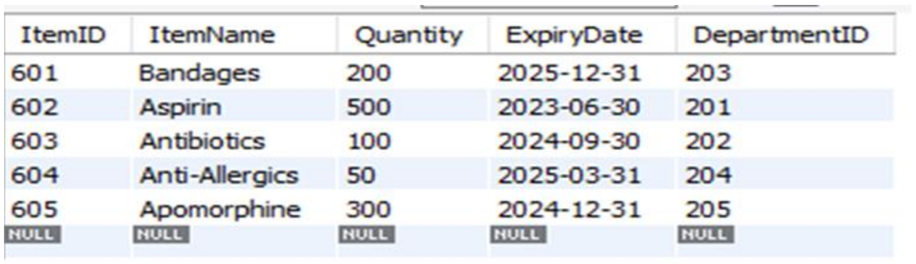
ItemID	ItemName	Quantity	ExpiryDate	DepartmentID	UsageHistory
601	Bandages	200	2025-12-31	203	Orthopedics
602	Aspirin	500	2023-06-30	201	Cardiology
603	Antibiotics	100	2024-09-30	202	Pediatrics
604	Anti-Allergics	50	2025-03-31	204	Dermatology
605	Apomorphine	300	2024-12-31	205	Neurology
NULL	NULL	NULL	NULL	NULL	NULL

ALTER TABLE Inventory

DROP COLUMN UsageHistory;

Check Query. 7

SELECT * FROM Inventory;



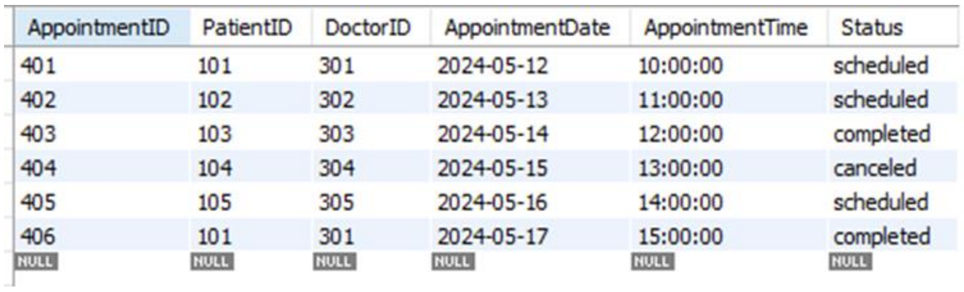
ItemID	ItemName	Quantity	ExpiryDate	DepartmentID
601	Bandages	200	2025-12-31	203
602	Aspirin	500	2023-06-30	201
603	Antibiotics	100	2024-09-30	202
604	Anti-Allergics	50	2025-03-31	204
605	Apomorphine	300	2024-12-31	205
NULL	NULL	NULL	NULL	NULL

QUERY NO. 8 - DELETE, WHERE.

Delete an appointment where appointment ID is 404:

Deleting an appointment (e.g., when canceled) keeps the system up-to-date, ensuring that resource allocation and scheduling remain accurate.

Before Deleting



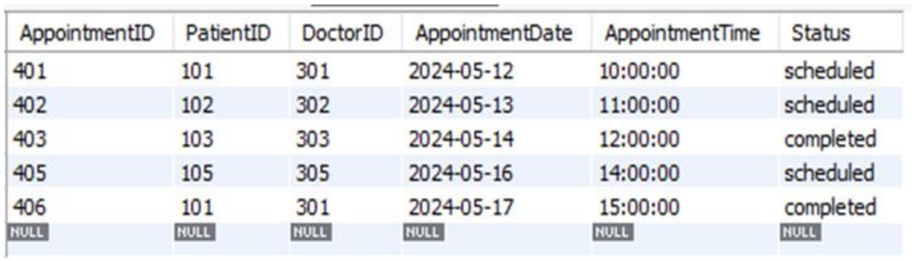
AppointmentID	PatientID	DoctorID	AppointmentDate	AppointmentTime	Status
401	101	301	2024-05-12	10:00:00	scheduled
402	102	302	2024-05-13	11:00:00	scheduled
403	103	303	2024-05-14	12:00:00	completed
404	104	304	2024-05-15	13:00:00	canceled
405	105	305	2024-05-16	14:00:00	scheduled
406	101	301	2024-05-17	15:00:00	completed
NULL	NULL	NULL	NULL	NULL	NULL

DELETE FROM Appointments

WHERE AppointmentID = 404;

Check Query No. 8

Select * From Appointments;



AppointmentID	PatientID	DoctorID	AppointmentDate	AppointmentTime	Status
401	101	301	2024-05-12	10:00:00	scheduled
402	102	302	2024-05-13	11:00:00	scheduled
403	103	303	2024-05-14	12:00:00	completed
405	105	305	2024-05-16	14:00:00	scheduled
406	101	301	2024-05-17	15:00:00	completed
NULL	NULL	NULL	NULL	NULL	NULL

QUERY NO. 9 – VIEW, SELECT, JOIN

View for doctor schedules

This view provides a clear schedule for doctors, integrating patient and appointment information, which is essential for managing doctor availability and workload

CREATE VIEW DoctorSchedules AS

SELECT Doctors.FirstName AS DoctorFirstName, Doctors.LastName AS DoctorLastName, Doctors.Specialty,

Patients.FirstName AS PatientFirstName, Patients.LastName AS PatientLastName,

Appointments.AppointmentDate, Appointments.AppointmentTime, Appointments.Status

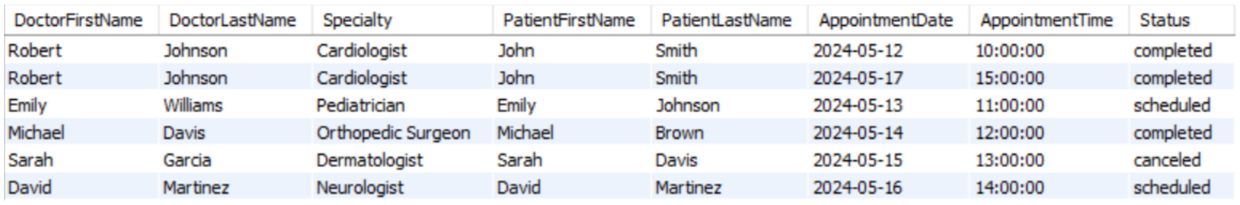
FROM Doctors

JOIN Appointments ON Doctors.DoctorID = Appointments.DoctorID

JOIN Patients ON Appointments.PatientID = Patients.PatientID;

Check QUERY NO. 9

SELECT * FROM DoctorSchedules;



DoctorFirstName	DoctorLastName	Specialty	PatientFirstName	PatientLastName	AppointmentDate	AppointmentTime	Status
Robert	Johnson	Cardiologist	John	Smith	2024-05-12	10:00:00	completed
Robert	Johnson	Cardiologist	John	Smith	2024-05-17	15:00:00	completed
Emily	Williams	Pediatrician	Emily	Johnson	2024-05-13	11:00:00	scheduled
Michael	Davis	Orthopedic Surgeon	Michael	Brown	2024-05-14	12:00:00	completed
Sarah	Garcia	Dermatologist	Sarah	Davis	2024-05-15	13:00:00	canceled
David	Martinez	Neurologist	David	Martinez	2024-05-16	14:00:00	scheduled

QUERY NO. 10 - TRIGGER, IF, THEN, UPDATE, UNION, SET

Implement a trigger that automatically updates the staff count of a department in the Departments table whenever a new doctor is added or removed.

Also use Union to combine two Queries for both insertion and deletion.

Automating updates to the staff count when doctors are added or removed ensures accurate and up-to-date information, critical for resource management and compliance with staffing requirements.

DELIMITER $$

CREATE TRIGGER UPDATE_STAFFCOUNT_AFTER_INSERT

AFTER INSERT ON DOCTORS

FOR EACH ROW

BEGIN

IF NEW.DEPARTMENTID IS NOT NULL THEN

UPDATE DEPARTMENTS

SET STAFFCOUNT = STAFFCOUNT + 1

WHERE DEPARTMENTID = NEW.DEPARTMENTID;

END IF;

END;

$$

DELIMITER ;

UNION

DELIMITER $$

CREATE TRIGGER UPDATE_STAFFCOUNT_AFTER_DELETE

AFTER DELETE ON DOCTORS

FOR EACH ROW

BEGIN

IF OLD.DEPARTMENTID IS NOT NULL THEN

UPDATE DEPARTMENTS

SET STAFFCOUNT = STAFFCOUNT - 1

WHERE DEPARTMENTID = OLD.DEPARTMENTID;

END IF;

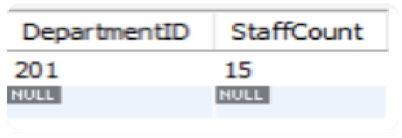
END;

$$

DELIMITER ;

Check query 10 if the trigger is working

Display staff count first



DepartmentID	StaffCount
201	15
NULL	NULL

SELECT DepartmentID, StaffCount FROM Departments WHERE DepartmentID = 201;

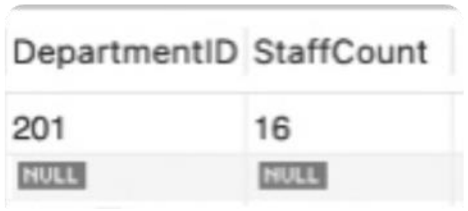
Insert a new doctor into the department with DepartmentID 201

INSERT INTO Doctors (DepartmentID, FirstName, LastName, Specialty, ContactNumber, Email)

VALUES (201, 'Test', 'Doctor', 'Test Specialty', '123-456-7890', 'test.doctor@hc.com');

Check the updated staff count for the department

SELECT DepartmentID, StaffCount FROM Departments WHERE DepartmentID = 201;



DepartmentID	StaffCount
201	16
NULL	NULL

-- staff count updated.

Query No 11

Trigger Logs every time an appointment status is updated, which is helpful for auditing and tracking patient flow.

This query creates a system to log every change in the status of appointments. This is crucial for auditing purposes, allowing the hospital to maintain records of all changes, ensuring accountability and transparency. The logs provide insights into patient flow and appointment management, helping to identify patterns or issues such as frequent cancellations or rescheduling.

Creating the Log table : AppointmentLog

```
CREATE TABLE AppointmentLog (
   LogID INT AUTO_INCREMENT PRIMARY KEY,
   AppointmentID INT,
   OldStatus ENUM('scheduled', 'completed', 'canceled'),
   NewStatus ENUM('scheduled', 'completed', 'canceled'),
   UpdateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Trigger to update appointment log when appointment table is updated.

```
DELIMITER $$
CREATE TRIGGER LogAppointmentUpdate
AFTER UPDATE ON Appointments
FOR EACH ROW
BEGIN
   IF OLD.Status != NEW.Status THEN
      INSERT INTO AppointmentLog (AppointmentID, OldStatus, NewStatus)
```

```
        VALUES (OLD.AppointmentID, OLD.Status, NEW.Status);
    END IF;
END;
$$
DELIMITER ;
```

Updating an appointment's status to test the trigger

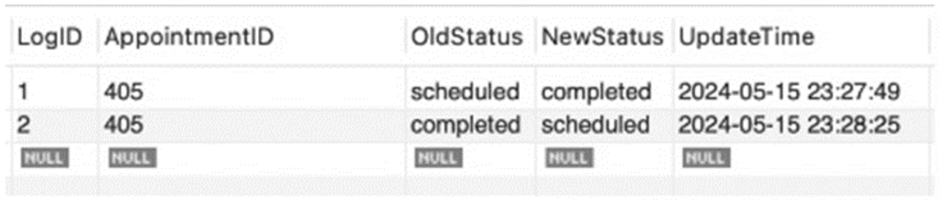
```
SELECT * FROM Appointments ;


UPDATE Appointments SET Status = 'completed' WHERE AppointmentID =
405;

UPDATE Appointments SET Status = 'scheduled' WHERE AppointmentID =
405;
```

Checking query 11 the log to see if the update has been recorded

```
SELECT * FROM AppointmentLog;
```



LogID	AppointmentID	OldStatus	NewStatus	UpdateTime
1	405	scheduled	completed	2024-05-15 23:27:49
2	405	completed	scheduled	2024-05-15 23:28:25
NULL	NULL	NULL	NULL	NULL

QUERY NO. 12

Trigger checks inventory levels after an update and logs a warning if the quantity of any item falls below a predefined threshold, aiding in inventory management.

This trigger helps manage inventory by logging warnings when the quantity of any item falls below a predefined threshold. This is vital for ensuring that the hospital does not run out of crucial medical supplies. By automating the monitoring of inventory levels, the hospital can maintain optimal stock, reducing the risk of shortages that could affect patient care.

Create the log table

```
CREATE TABLE InventoryLog (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    ItemID INT,
    Quantity INT,
    WarningMessage VARCHAR(255),
    LogTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Trigger

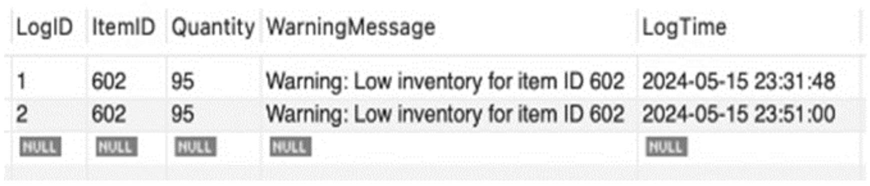
```
DELIMITER $$
CREATE TRIGGER CheckInventoryAfterUpdate
AFTER UPDATE ON Inventory
FOR EACH ROW
BEGIN
    IF NEW.Quantity < 100 THEN
        INSERT INTO InventoryLog (ItemID, Quantity, WarningMessage)
        VALUES (NEW.ItemID, NEW.Quantity, CONCAT('Warning: Low
inventory for item ID ', NEW.ItemID));
    END IF;
END;
$$
DELIMITER ;
```

Updating an inventory item to reduce its quantity below the threshold to test the trigger

```
UPDATE Inventory SET Quantity = 95 WHERE ItemID = 602;
```

Check Query 12 - the inventory log to see if the warning has been recorded

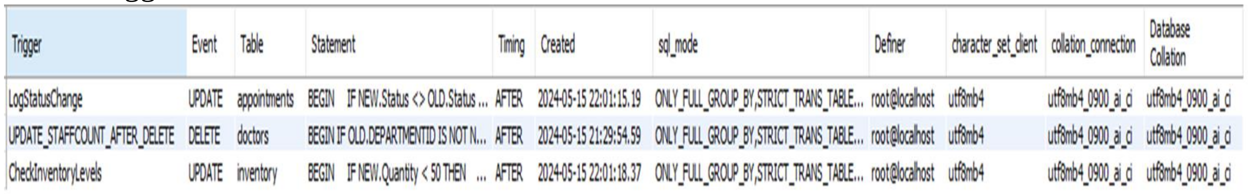
```
SELECT * FROM InventoryLog;
```



LogID	ItemID	Quantity	WarningMessage	LogTime
1	602	95	Warning: Low inventory for item ID 602	2024-05-15 23:31:48
2	602	95	Warning: Low inventory for item ID 602	2024-05-15 23:51:00
NULL	NULL	NULL	NULL	NULL

Check Query No 10, 11 & 12

Show Triggers;

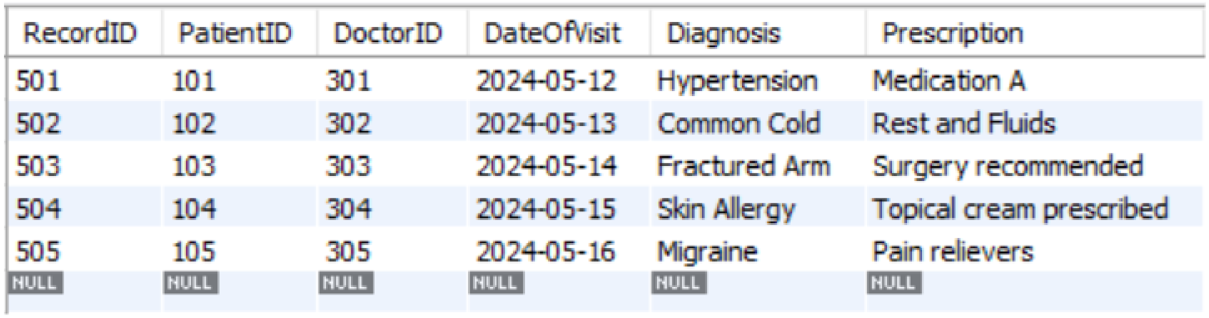


Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
LogStatusChange	UPDATE	appointments	BEGIN IF NEW.Status <> OLD.Status ...	AFTER	2024-05-15 22:01:15.19	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	root@localhost	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci
UPDATE_STAFFCOUNT_AFTER_DELETE	DELETE	doctors	BEGIN IF OLD.DEPARTMENTID IS NOT N...	AFTER	2024-05-15 21:29:54.59	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	root@localhost	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci
CheckInventoryLevels	UPDATE	inventory	BEGIN IF NEW.Quantity < 50 THEN ...	AFTER	2024-05-15 22:01:18.37	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	root@localhost	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

QUERY NO. 13 - UPDATE, SET, WHERE.

Before Update

Select * From MedicalRecords;



RecordID	PatientID	DoctorID	DateOfVisit	Diagnosis	Prescription
501	101	301	2024-05-12	Hypertension	Medication A
502	102	302	2024-05-13	Common Cold	Rest and Fluids
503	103	303	2024-05-14	Fractured Arm	Surgery recommended
504	104	304	2024-05-15	Skin Allergy	Topical cream prescribed
505	105	305	2024-05-16	Migraine	Pain relievers
NULL	NULL	NULL	NULL	NULL	NULL

UPDATE MedicalRecords

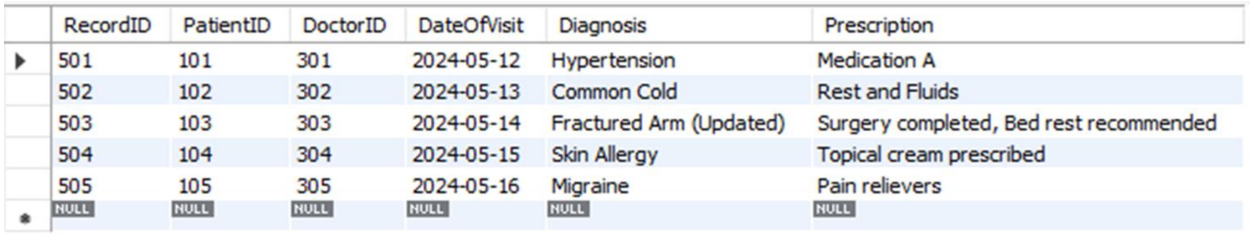
SET Diagnosis = 'Fractured Arm (Updated)',

Prescription = 'Surgery completed, Bed rest recommended'

WHERE RecordID = 503;

Check QUERY NO. 13

Select * From MedicalRecords;



RecordID	PatientID	DoctorID	DateOfVisit	Diagnosis	Prescription
501	101	301	2024-05-12	Hypertension	Medication A
502	102	302	2024-05-13	Common Cold	Rest and Fluids
503	103	303	2024-05-14	Fractured Arm (Updated)	Surgery completed, Bed rest recommended
504	104	304	2024-05-15	Skin Allergy	Topical cream prescribed
505	105	305	2024-05-16	Migraine	Pain relievers
NULL	NULL	NULL	NULL	NULL	NULL

CONCLUSION

The development of the Care Hospital Database Management System provided us with invaluable insights into database design and management. This project not only enhanced our technical skills but also deepened our understanding of organizational structures within the healthcare sector.

Benefits Gained:

Through the development of the Care Hospital Database Management System, we gained significant benefits, including a deeper understanding of database design and management principles. This project enhanced our knowledge of relational databases, SQL query formulation, and the intricate structure of a healthcare organization. We learned to model complex relationships and ensure data integrity, which is crucial in a sensitive field like healthcare.

Challenges Encountered:

We encountered challenges related to query formulation and data insertion. Ensuring that data types matched those in our original domain constraints required meticulous attention to detail. Proper placement of foreign keys, considering cardinalities, and maintaining database integrity were significant hurdles. Additionally, correctly using SQL functions where required posed challenges. We addressed these issues by consulting academic resources which were provided to us during classes, seeking guidance from our instructor, and iteratively refining our design through peer reviews and feedback. This methodical approach enabled us to overcome these obstacles and develop a robust and functional database system.