Hospital management system Project- 16

Name: Lokeshwari Bhute

Email:lokeshwareeb@gmail.com

**HMS Project Structure**

1. **Database Creation Script:**
   * Create a SQL script to generate the database schema and initial data.
   * Tables
   * Appointment
   * Doctor
   * Patients
   * PatientsAttendAppointments
   * Medical history
   * PatientFillHistory
   * Medicine Cost

Queries to Solve within the HMS Structure

* **Find the names of patients who have attended appointments scheduled by Dr. John Doe.**

SELECT P.FNAME,P.LNAME FROM PATIENT P

JOIN APPOINTMENT A

ON A.PatientID = P.PatientID

JOIN DOCTOR D

ON D.DoctorID=A.DoctorID

WHERE

d.Fname = 'Dr.John' AND

d.Lname = 'Doe' AND

a.Status = 'Scheduled'

**OUTPUT:** This query is used to find and list the names of patients who have an upcoming scheduled appointment with Dr. John Doe.

* **Calculate the average age of all patients.**

select AVG(AGE) AS 'AVERAGE\_AGE' FROM PATIENT

**OUTPUT**: This average age can provide insights into the overall age distribution of the patient population, which might be useful for understanding demographic trends or planning medical services.

* **Create a stored procedure to get the total number of appointments for a given patient.**

CREATE PROCEDURE GetTotalAppointmentsForPatient

@PatientID VARCHAR(10),

@TotalAppointments INT OUTPUT

AS

BEGIN

-- Calculate the total number of appointments for the given patient

SELECT @TotalAppointments = COUNT(\*)

FROM APPOINTMENT

WHERE PatientID = @PatientID;

END;

GO

DECLARE @TotalAppointments INT;

EXEC GetTotalAppointmentsForPatient

@PatientID = 'P0001', ----IT CAN BE REPLACE BY OTHER PATIENTID

@TotalAppointments = @TotalAppointments OUTPUT;

SELECT @TotalAppointments;

**OUTPUT**: This procedure helps in quickly retrieving appointment counts for a patient, which can be useful for scheduling, tracking patient care, and managing appointment-related data.

* **Create a trigger to update the appointment status to 'Completed' when the appointment date has passed.**

CREATE TRIGGER trgUpdateAppointmentStatus

ON APPOINTMENT

AFTER INSERT

AS

BEGIN

-- Update the appointment status to 'Completed' if the appointment date has passed

UPDATE APPOINTMENT

SET Status = 'Completed'

FROM APPOINTMENT a

JOIN INSERTED i ON a.AppointmentID = i.AppointmentID

WHERE a.Date < GETDATE() AND a.Status <> 'Completed';

END;

select \* from APPOINTMENT

INSERT INTO APPOINTMENT (AppointmentID, PatientID, DoctorID, Date, EndTime, Status)

VALUES ('A0011', 'P0001', 'D0001', '2023-09-01 10:00', '2023-09-01 11:00', 'Scheduled');

SELECT \* FROM APPOINTMENT WHERE AppointmentID = 'A0011';

INSERT INTO APPOINTMENT (AppointmentID, PatientID, DoctorID, Date, EndTime, Status)

VALUES ('A0012', 'P0002', 'D0002', '2024-12-01 10:00', '2024-12-01 11:00', 'Scheduled');

SELECT \* FROM APPOINTMENT WHERE AppointmentID = 'A0012';

**OUTPUT**: This setup automates the process of updating appointment statuses based on their date, which helps maintain accurate status information without manual intervention.

* **Find the names of patients along with their appointment details and the corresponding doctor's name.**

SELECT P.FNAME,P.LNAME,A.\*,D.Fname,D.Lname FROM PATIENT P

JOIN APPOINTMENT A

ON A.PatientID = P.PatientID

JOIN DOCTOR D

ON D.DoctorID=A.DoctorID

* **Find the patients who have a medical history of diabetes and their next appointment is scheduled within the next 7 days.**

SELECT P.PatientID, P.Fname, P.Lname, A.AppointmentID, A.Date, PH.Condition

FROM PATIENT P

JOIN PATIENT\_HISTORY PH ON P.PatientID = PH.PatientID

JOIN APPOINTMENT A ON P.PatientID = A.PatientID

WHERE PH.Condition = 'Diabetes'

AND A.Status = 'Scheduled'

AND A.Date BETWEEN GETDATE() AND DATEADD(DAY, 7, GETDATE());

**OUTPUT**: This query can be useful for scheduling follow-ups or managing appointments for patients with specific medical conditions

* **Find patients who have multiple appointments scheduled.**

SELECT PatientID, COUNT(AppointmentID) AS AppointmentCount

FROM APPOINTMENT

WHERE Status = 'Scheduled'

GROUP BY PatientID

HAVING COUNT(AppointmentID) > 1;

**OUTPUT**: This information can help in identifying patients with multiple upcoming appointments, which could be useful for managing schedules or understanding patient engagement.

* **Calculate the average duration of appointments for each doctor.**

SELECT

A.DoctorID,

D.Fname AS DoctorFirstName,

D.Lname AS DoctorLastName,

AVG(DATEDIFF(MINUTE, A.Date, A.EndTime)) AS AvgAppointmentDurationInMinutes

FROM

APPOINTMENT A

JOIN

DOCTOR D ON A.DoctorID = D.DoctorID

GROUP BY

A.DoctorID, D.Fname, D.Lname;

**OUTPUT**: This information can be useful for evaluating doctors' appointment times and understanding how long, on average, their appointments last.

* **Find Patients with Most Appointments**

SELECT PatientID, COUNT(AppointmentID) AS AppointmentCount

FROM PATIENT\_APPOINTMENT

GROUP BY PatientID

HAVING COUNT(AppointmentID) > 1;

**OUTPUT**: This query helps in identifying patients with multiple appointments, which can be useful for analyzing patient engagement, managing appointment schedules, or targeting follow-ups.

* **Calculate the total cost of medication for each patient.**

SELECT

P.PatientID,

SUM(M.Cost\_in\_Dollars) AS TotalMedicationCost

FROM PRESCRIPTION PR

JOIN PATIENT\_HISTORY PH ON PR.HistoryID = PH.HistoryID

JOIN MEDICATION M ON PH.Medication = M.Medication

JOIN PATIENT P ON PR.PatientID = P.PatientID

GROUP BY P.PatientID;

**OUTPUT:** This information helps in understanding the total expenditure on medications for each patient, which can be useful for billing, financial analysis, or managing medication costs**.**

* **Create a stored procedure named CalculatePatientBill that calculates the total bill for a patient based on their medical history and medication costs. The procedure should take the PatientID as a parameter and calculate the total cost by summing up the medication costs and applying a charge of $50 for each surgery in the patient's medical history. If the patient has no medical history, the procedure should still return a basic charge of $50.**

CREATE PROCEDURE CalculatePatientBill

@PatientID VARCHAR(10)

AS

BEGIN

DECLARE @TotalCost DECIMAL(10, 2) = 0;

DECLARE @SurgeryCount INT = 0;

DECLARE @MedicationCost DECIMAL(10, 2) = 0;

-- Calculate the cost of medications for the patient

SELECT @MedicationCost = SUM(M.Cost\_in\_Dollars)

FROM PATIENT\_HISTORY PH

JOIN MEDICATION M ON PH.Medication = M.Medication

WHERE PH.PatientID = @PatientID;

-- Calculate the number of surgeries for the patient

SELECT @SurgeryCount = COUNT(\*)

FROM PATIENT\_HISTORY

WHERE PatientID = @PatientID AND Surgeries <> 'None';

-- Calculate total cost

SET @TotalCost = ISNULL(@MedicationCost, 0) + (@SurgeryCount \* 50);

-- If no medical history, return a basic charge of $50

IF @TotalCost = 0

BEGIN

SET @TotalCost = 50;

END

-- Return the total cost

SELECT @TotalCost AS TotalBill;

END;

EXEC CalculatePatientBill @PatientID = 'P0001';

**OUTPUT**: The final bill reflects the sum of medication costs and applicable surgery charges (or a basic charge if no surgeries are present).This procedure ensures that the patient's bill accurately reflects both their medication costs and any additional charges due to their surgical history.

Note: Kindly create a Docx file and paste the queries there with a short brief as a conclusion para, and also keep your SQL server open at the time of presentation.

Kindly Paste your Queries Here

Conclusions (Summary):