**Project Title: Hospital Management System**

**Abstract:** This report presents a SQL-based Hospital Management System (HMS) designed to replace inefficient manual administrative processes. The system utilizes a normalized relational database to manage core entities like patients, doctors, appointments, and billing, ensuring data integrity. Functionality is driven by diverse SQL queries, while stored procedures and triggers are implemented to automate tasks and enhance operational efficiency. It successfully centralizes patient records, streamlines scheduling, and maintains consistency across financial and medical data. Ultimately, the HMS improves overall hospital performance and elevates the quality of patient care.

1. **Introduction**

The Hospital Management System (HMS) is a comprehensive and integrated system designed to The Hospital Management System (HMS) is developed to manage the operations of a hospital in a more efficient and reliable manner. By using database management principles, the system organizes patient records, doctor schedules, medical history, appointments, billing, staff details, medicines, and pharmacy transactions. The primary goal of this project is to reduce manual work, minimize errors, and provide a centralized system that improves hospital service quality and patient care.

1. **Objectives**

The objectives of the Hospital Management System are:

* To store and manage patient details, medical history, and contact information.
* To schedule and manage doctor appointments in a structured way.
* To maintain accurate medical records including diagnosis, treatment, and prescriptions.
* To streamline billing processes and track payment status.
* To manage hospital staff and their responsibilities effectively.
* To maintain medicine stock and pharmacy transactions.
* To provide real-time access to information for doctors, staff, and administrators.
* To reduce redundancy, ensure data integrity, and improve overall hospital service.

1. **Database Design and Table Explanation**

The database for the Hospital Management System consists of multiple tables, each focusing on specific aspects of the hospital's operations. The tables are organized to follow best practices for data normalization, ensuring efficiency and preventing data redundancy. Below is an overview of the primary tables:

**Patients Table:** Stores essential information about patients such as name, contact details, medical history, and date of birth. The patient\_id is the primary key, making each patient unique. This table links to appointments, medical records, and billing.

**Doctors Table:** Holds information about doctors including name, specialty, and contact details. The doctor\_id is the primary key and links to patient appointments and medical records.

**Appointments Table:** Records patient appointments with doctors, including appointment date, time, purpose, and status. Links patients and doctors through patient\_id and doctor\_id.

**Medical Records Table:** Stores the diagnosis, treatment, and prescriptions for each patient visit. It links to the Appointments and Doctors tables to record what happened during an appointment.

**Billing Table:** Manages financial information for appointments, such as the total amount, payment status, and insurance provider. It links to patients and appointments to track payments and billing statuses.

**Medicine Table:** Tracks medicines in the hospital's pharmacy, including medicine name, type, dosage, stock quantity, and expiry date. The medicine\_id serves as the primary key**.**

**Pharmacy Table:** Links patients to the medicines they’ve been prescribed, storing the medicine\_id, patient\_id, quantity, and prescription date. This helps track medication dispensing.

1. **SQL Queries and Operations**

Below is a detailed explanation of the SQL queries used in the Hospital Management System. These queries perform various operations related to patient management, appointments, billing, stored procedures, and triggers

**Get All Table Names in the Database**

SELECT table\_name

FROM information\_schema.tables

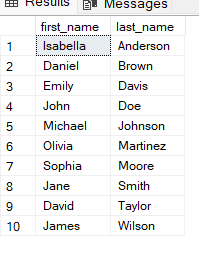
WHERE table\_type = 'BASE TABLE';

This query retrieves a list of all the tables in the database. The information\_schema.tables is a system view that provides metadata about all the tables in the database. The condition WHERE table\_type = 'BASE TABLE' filters out any views and returns only the base tables (actual data tables).

**Show All Records from the Patients Table**

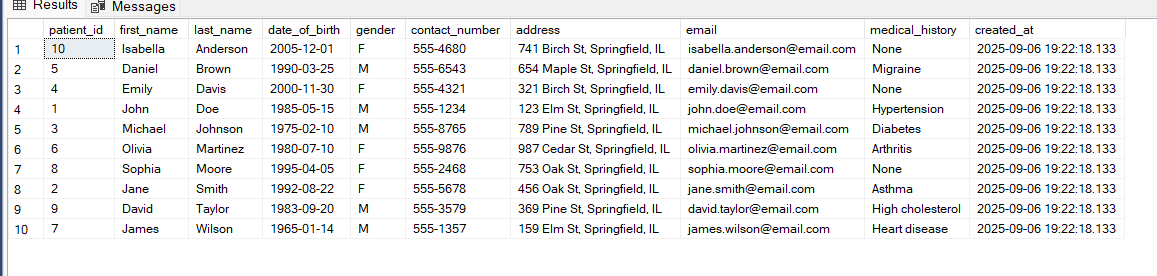
SELECT first\_name, last\_name FROM Patients;

This query selects only the first\_name and last\_name columns from the Patients table. It is useful when you only need to retrieve the names of all patients without any other information.



**Show Records from the Patients Table Ordered by Last Name**

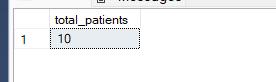
SELECT \* FROM Patients ORDER BY last\_name ASC;

****

This query retrieves all columns from the Patients table, but it orders the records alphabetically by the last\_name column in ascending order (ASC). This is helpful when you need to see the list of patients sorted by their last names.

**Count the Total Number of Patients**

SELECT COUNT(\*) AS total\_patients FROM Patients;

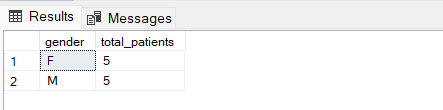
****

This query counts the total number of patients in the Patients table. The COUNT(\*) function counts all rows in the table. The result is returned with the alias total\_patients, providing a summary of the number of patients in the system.

**Count the Number of Patients Grouped by Gender**

SELECT gender, COUNT(\*) AS total\_patients FROM Patients

GROUP BY gender;

****

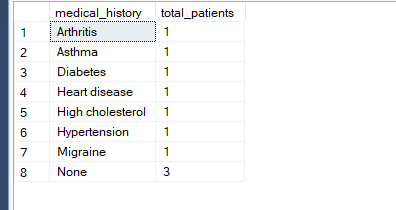
This query groups patients by their gender and counts how many patients fall under each gender category. The GROUP BY clause groups rows with the same values in the gender column, and the COUNT(\*) function returns the number of patients for each group

**Count the Number of Patients with Specific Medical History**

SELECT medical\_history, COUNT(\*) AS total\_patients FROM Patients

WHERE medical\_history IS NOT NULL

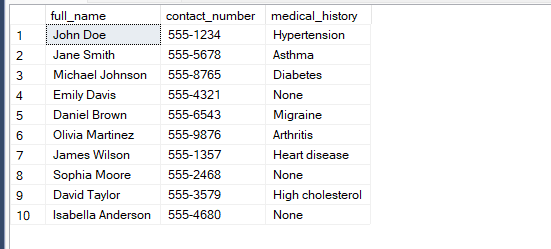
GROUP BY medical\_history;

****

This query counts the number of patients who have a specific medical history (non-null entries). It uses the WHERE clause to filter out patients without a medical history (medical\_history IS NOT NULL). The GROUP BY clause groups the patients by their medical\_history, and COUNT(\*) counts the number of patients within each group.

**Concatenate First and Last Name as Full Name and Show Contact Number and Medical History**

SELECT first\_name + ' ' + last\_name AS full\_name, contact\_number, medical\_history FROM Patients;

****

This query concatenates the first\_name and last\_name columns into a single column called full\_name using the + operator. It also retrieves the contact\_number and medical\_history columns from the Patients table. This query is useful for generating a more readable format of patient names.

**Join Patients Table with Appointments Table to Get Patient Details Along with Appointment Dates**

SELECT p.first\_name, p.last\_name, a.appointment\_date, a.appointment\_time

FROM Patients p

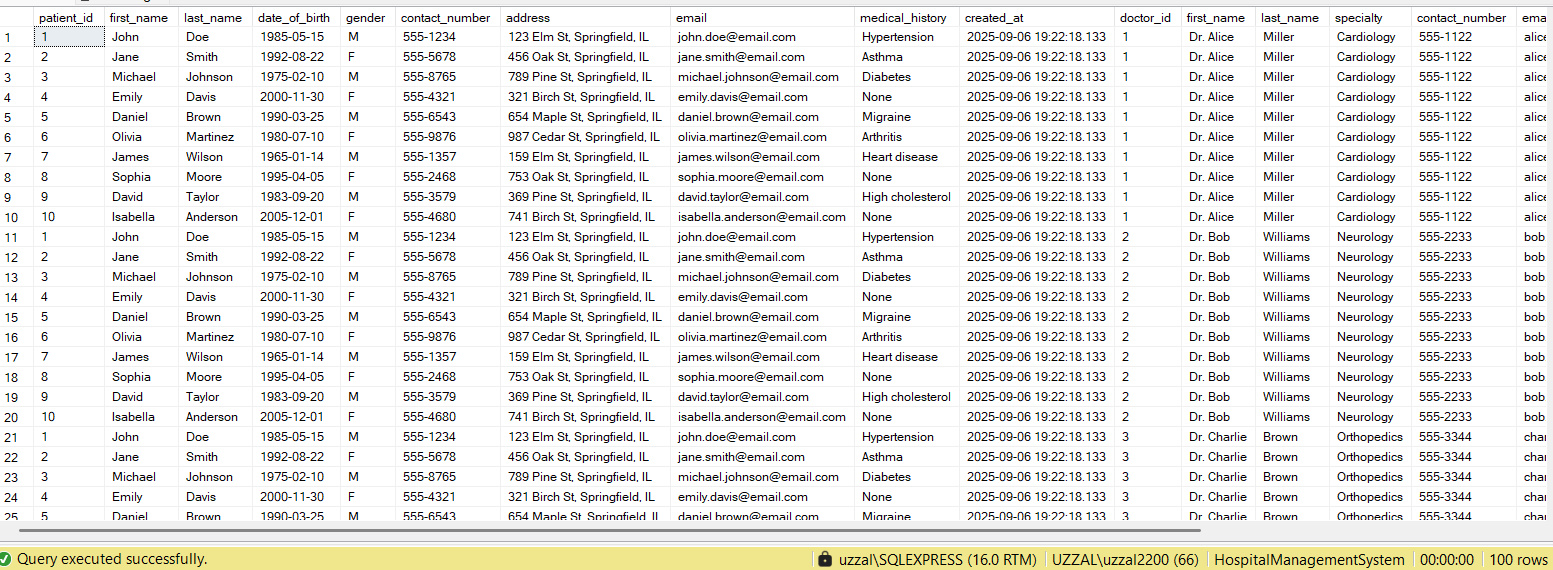
INNER JOIN Appointments a ON p.patient\_id = a.patient\_id;

****

This query performs an INNER JOIN between the Patients and Appointments tables. It links the two tables based on the patient\_id column, and retrieves the patient's first\_name, last\_name, along with the associated appointment\_date and appointment\_time. This query is used to get the list of patients along with their scheduled appointments

**Get the Cartesian Product of Patients and Doctors Tables**

SELECT \* FROM Patients, Doctors;

****

This query retrieves the Cartesian product of the Patients and Doctors tables. It combines every row in the Patients table with every row in the Doctors table, resulting in all possible combinations of patients and doctors. This type of query is typically not used in practice because it creates a large number of results, but it can be useful in certain situations where you want to check all possible combinations.

**Nested query**

-- Operation: Get all patients who have an appointment on a specific date

SELECT \* FROM Patients

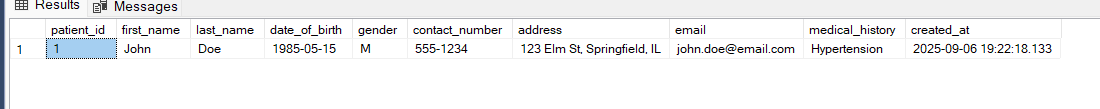
WHERE patient\_id IN (

SELECT patient\_id

FROM Appointments

WHERE appointment\_date = '2024-11-20'

);

****

**Create the stored procedure**

CREATE PROCEDURE GetAllPatients

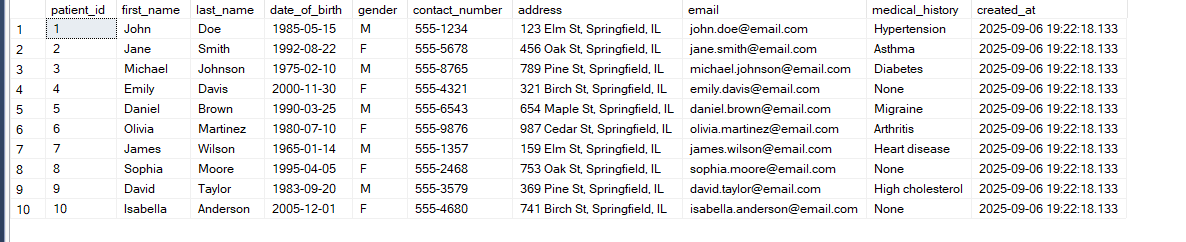
AS

BEGIN

SELECT \* FROM Patients;

END;

EXEC GetAllPatients;

****

This stored procedure is created to retrieve all records from the Patients table. Stored procedures are used to group multiple SQL statements into a single unit, which can be executed with a single command. This is useful for repetitive tasks.

**Execute the Stored Procedure to Get All Patient Records**

This command executes the GetAllPatients stored procedure, which retrieves all records from the Patients table. Stored procedures are executed using the EXEC command.

**Create a Parameterized Stored Procedure to Fetch Patients by Medical History**

CREATE PROCEDURE GetPatientsByHistory (@history NVARCHAR(255))

AS

BEGIN

SELECT \* FROM Patients WHERE medical\_history = @history;

END;

EXEC GetPatientsByHistory @history = 'Hypertension';

This parameterized stored procedure is created to fetch patients based on a specific medical history. The @history parameter allows you to dynamically specify the medical condition, and the procedure retrieves all patients who have that medical history**.** This command executes the GetPatientsByHistory stored procedure, passing in the value 'Hypertension' as the medical history. It retrieves all patients who have been diagnosed with hypertension

**Trigger to Update Billing Status When an Appointment Status is Updated**

CREATE TRIGGER UpdateBillingStatus

ON Appointments

AFTER UPDATE

AS

BEGIN

IF UPDATE(status)

BEGIN

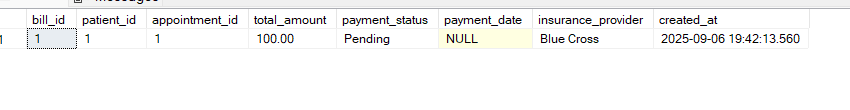
UPDATE Billing

SET payment\_status = 'Pending'

WHERE appointment\_id IN (SELECT appointment\_id FROM inserted);

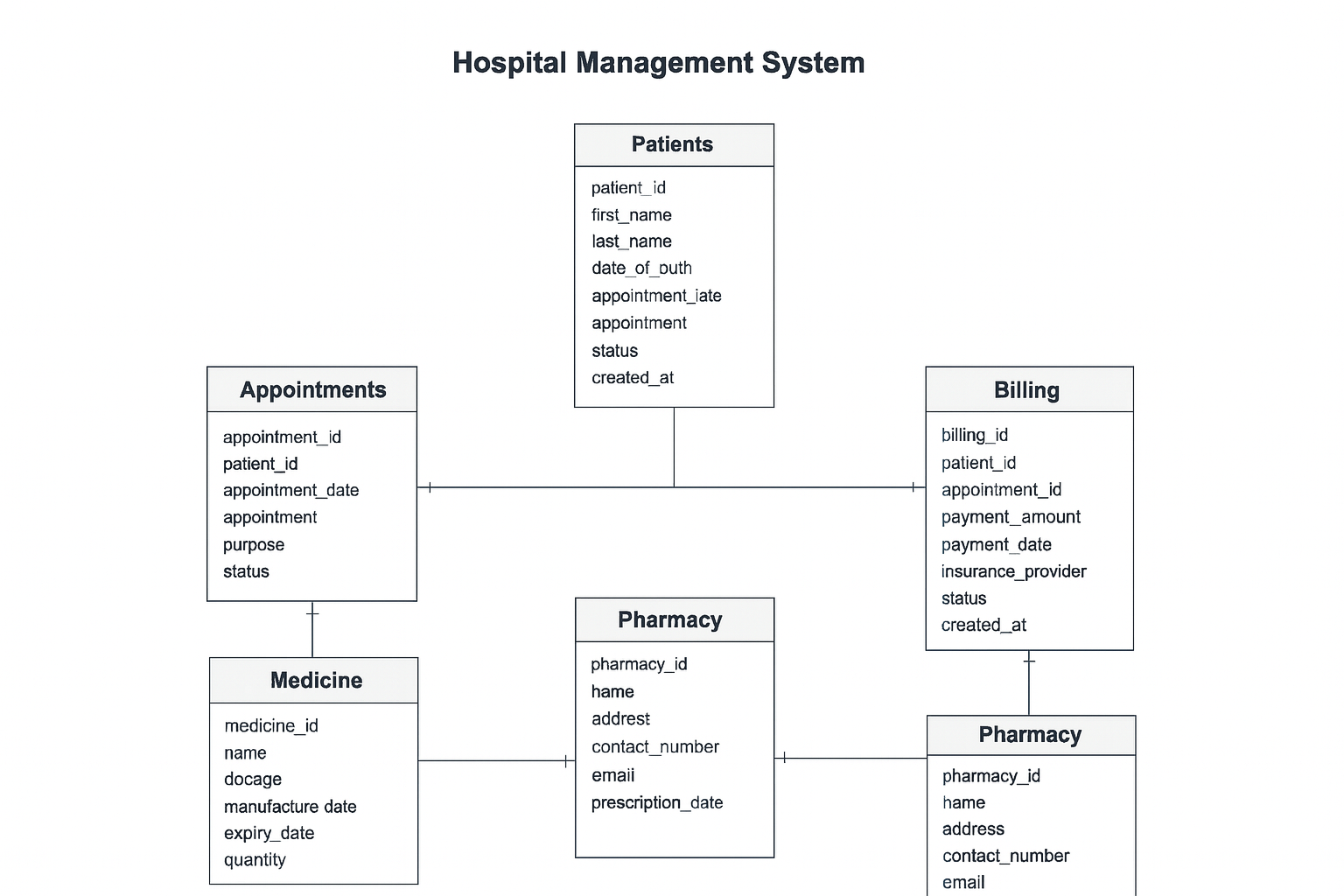
END

END;

****

This trigger automatically updates the billing status to "Pending" whenever the status of an appointment is updated. The AFTER UPDATE clause ensures that the trigger runs after an update operation on the Appointments table. The INSERTED table contains the updated rows, and the trigger checks for changes in the status column.

1. **ER Diagram**

****

**Figure 01: ER diagram of Hospital Management System**

**6. Results and Discussion**

The implemented Hospital Management System allows hospitals to handle patient data, doctor schedules, billing, and medicine inventory more efficiently. Queries can retrieve meaningful insights such as the total number of patients, patients grouped by gender, or those with specific medical histories. Stored procedures simplify operations like checking if a patient exists or fetching all patients with a particular disease. Triggers automate billing updates, ensuring consistency without manual intervention.

**7. Conclusion**

The Hospital Management System provides a structured and reliable solution to the problems faced by hospitals in managing operations manually. By integrating patient details, appointments, medical records, billing, staff, and pharmacy into one system, it ensures consistency, accuracy, and efficiency. The database supports automation, reporting, and real-time updates, which ultimately improves patient care and hospital performance.