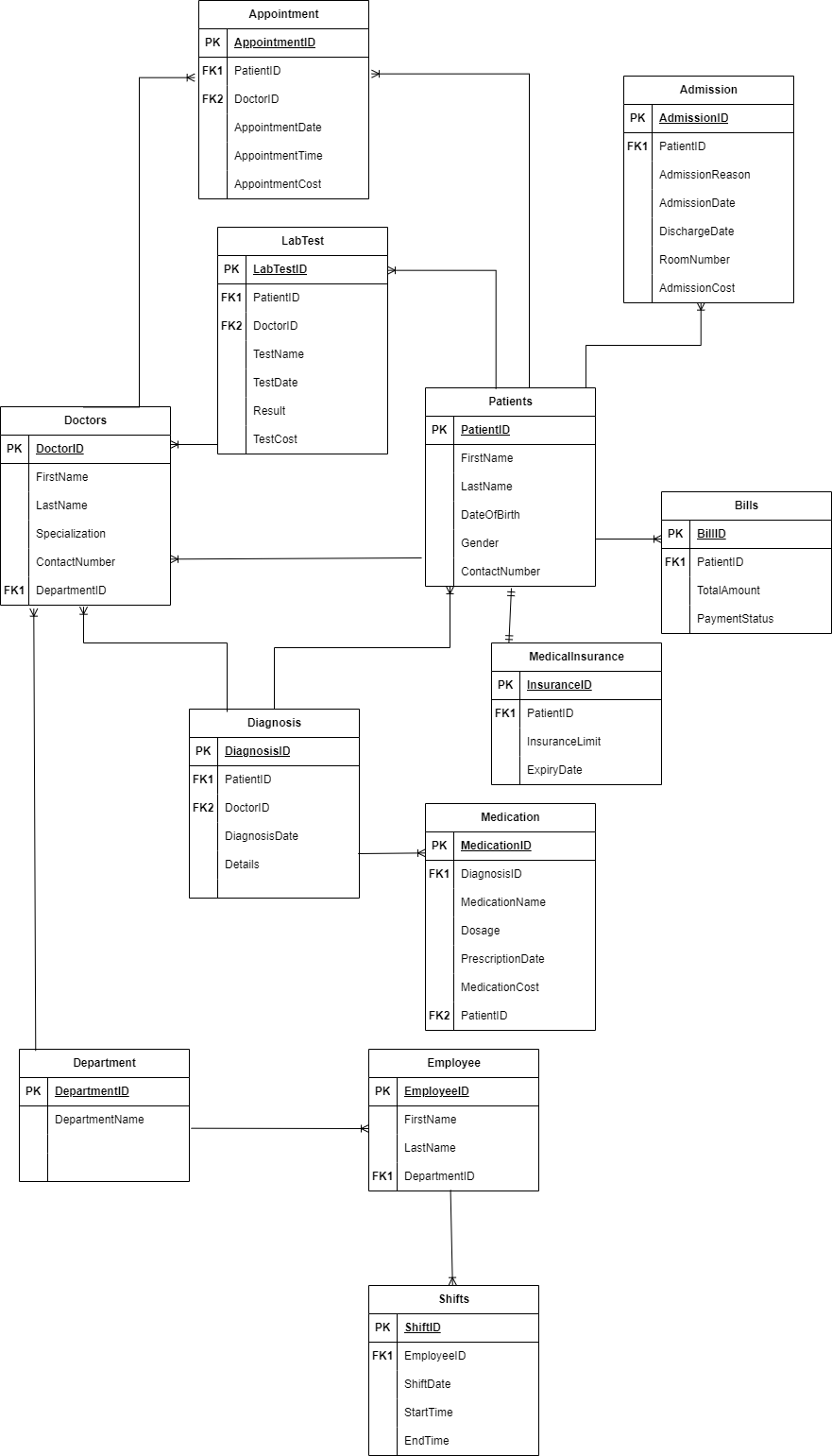
Milestone Project 1

Name:- Milind Sable

Q1.



I chose Third Normal Form (3NF) for the Hospital Management System database because it minimizes data redundancy, ensuring efficient storage and preventing data inconsistencies. By eliminating transitive dependencies, it simplifies maintenance, supports data integrity, and strikes a balance between normalization and practicality for effective querying in healthcare operations.

In Hospital Management System, data redundancy should be reduced because it can cause errors while updating the data. 3NF form also gives ability to perform queries on database easily without introducing excessive complexity.

1. Write necessary queries to register new user roles and persons.

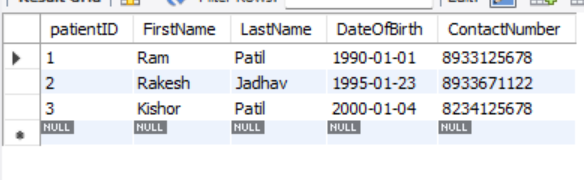
* Inserting New Patient Data : -

INSERT INTO hospitalmanagement.patient (patientID, FirstName, LastName, DateOfBirth, ContactNumber) VALUES

(1, 'Ram', 'Patil', '1990-01-01', '8933125678'),

(2, 'Rakesh', 'Jadhav', '1995-01-23', '8933671122'),

(3, 'Kishor', 'Patil', '2000-01-04', '8234125678')



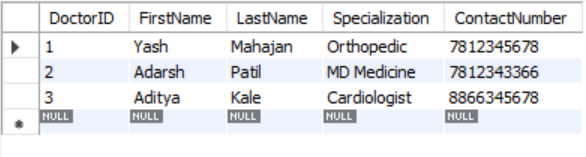
* Inserting New Doctors Data : -

INSERT INTO hospitalmanagement.doctors (DoctorID, FirstName, LastName, Specialization, ContactNumber, DepartmentID) VALUES

(1, 'Yash', 'Mahajan','Orthopedic','7812345678', 2),

(2, 'Adarsh', 'Patil', 'MD Medicine','7812343366', 3),

(3, 'Aditya', 'Kale','Cardiologist','8866345678', 1)



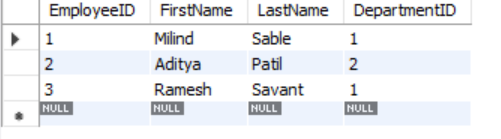
* Inserting New Employee Data : -

INSERT INTO hospitalmanagement.employee (EmployeeID, FirstName, LastName, DepartmentID) VALUES

(1, 'Milind', 'Sable', 1),

(2, 'Aditya', 'Patil', 2),

(3, 'Ramesh', 'Savant', 1)



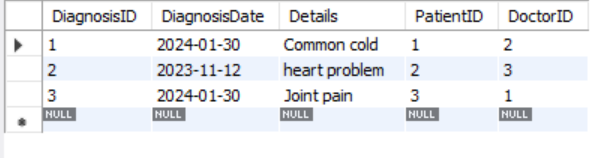
1. Write necessary queries to add to the list of diagnosis of the patient tagged by date.

* INSERT INTO hospitalmanagement.diagnosis (DiagnosisID, DiagnosisDate, Details, PatientID, DoctorID) VALUES

(1, '2024-01-30', 'Common cold',1,2),

(2, '2023-11-12', 'heart problem',2,3),

(3, '2024-01-30', 'Joint pain',3,1)



1. Write necessary queries to fetch required details of a particular patient.

I have used view to fetch all patient details like appointment details, diagnosis details, admission details, etc.

CREATE VIEW hospitalmanagement.PatientDetailsView1 AS

SELECT

p.\*,

d.Details,

m.Doasage, m.medicationName,

lt.TestName, lt.Result,

a.appointmentDate, a.DoctorID,

ad.admissionDate, ad.dischargeDate,

b.TotalAmount, b.status,

mi.insuranceLimit, mi.ExpiryDate

FROM

hospitalmanagement.patient p

LEFT JOIN hospitalmanagement.diagnosis d ON p.patientID = d.PatientID

LEFT JOIN hospitalmanagement.medication m ON p.patientID = m.PatientID

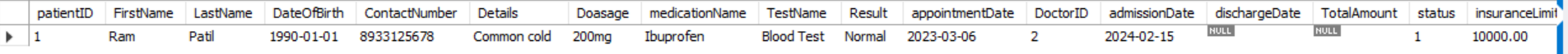
LEFT JOIN hospitalmanagement.labtest lt ON p.patientID = lt.PatientID

LEFT JOIN hospitalmanagement.appointment a ON p.patientID = a.PatientID

LEFT JOIN hospitalmanagement.admission ad ON p.patientID = ad.PatientID

LEFT JOIN hospitalmanagement.bills b ON p.patientID = b.PatientID

LEFT JOIN hospitalmanagement.medicalinsurance mi ON p.patientID = mi.PatientID;



1. Write necessary queries to prepare bill for the patient at the end of checkout.

CREATE PROCEDURE CalculateAndUpdateBill1(IN inPatientID INT)

BEGIN

DECLARE totalCost DECIMAL(10, 2);

SELECT

COALESCE(SUM(ad.admissionCost), 0) +

COALESCE(SUM(a.appointmentCost), 0) +

COALESCE(SUM(lt.TestCost), 0) +

COALESCE(SUM(m.MedicationCost), 0) AS TotalCost

INTO totalCost

FROM

hospitalmanagement.admission ad

LEFT JOIN hospitalmanagement.appointment a ON ad.PatientID = a.PatientID

LEFT JOIN hospitalmanagement.labtest lt ON ad.PatientID = lt.PatientID

LEFT JOIN hospitalmanagement.medication m ON ad.PatientID = m.PatientID

WHERE

ad.PatientID = inPatientID;

UPDATE hospitalmanagement.bills

SET TotalAmount = totalCost

WHERE PatientID = inPatientID;

END //

DELIMITER ;

call CalculateAndUpdateBill1(1);

1. Write necessary queries to fetch and show data from various related tables (Joins)

* Join query for displaying employee details.

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

s.StartTime,

s.endTime,

s.ShiftDate,

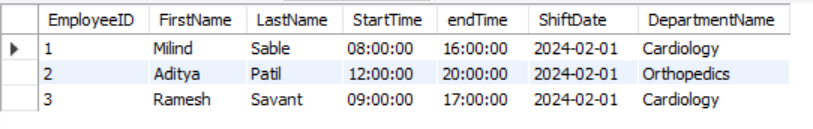
d.DepartmentName

FROM

hospitalmanagement.employee e

LEFT JOIN hospitalmanagement.shifts s ON e.EmployeeID = s.EmployeeID

LEFT JOIN hospitalmanagement.department d ON e.DepartmentID = d.DepartmentID;



* Join query for displaying Doctors details.

SELECT

d.DoctorID,

d.FirstName,

d.LastName,

d.Specialization,

d.ContactNumber,

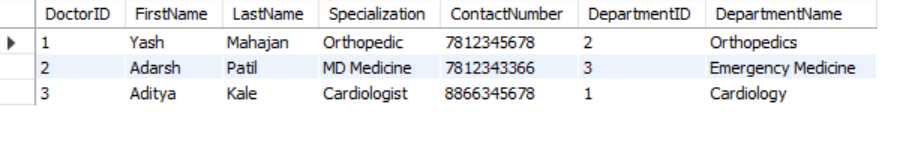
d.DepartmentID,

dt.DepartmentName

FROM

hospitalmanagement.doctors d

LEFT JOIN hospitalmanagement.department dt ON d.DepartmentID = dt.DepartmentID;



1. Optimize read operations using indexing wherever required. (Create index on at least 1 table)

CREATE INDEX idx\_DoctorID ON `hospitalmanagement`.`doctors` (`DoctorID`);

CREATE INDEX idx\_PatientID ON `hospitalmanagement`.`patient` (`PatientID`);

CREATE INDEX idx\_AppointmentID ON `hospitalmanagement`.`appointment` (`AppointmentID`);

CREATE INDEX idx\_EmployeeID ON `hospitalmanagement`.`employee` (`EmployeeID`);

1. Add necessary triggers to indicate when patients medical insurance limit has expired.

DELIMITER //

CREATE TRIGGER check\_medical\_insurance\_limit

BEFORE INSERT ON `hospitalmanagement`.`medicalinsurance`

FOR EACH ROW

BEGIN

IF NEW.ExpiryDate IS NOT NULL AND NEW.ExpiryDate < CURDATE() THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Medical insurance limit has expired.';

END IF;

END;

//

DELIMITER ;

1. Views to fetch different information

* View to fetch details of patient their bill and medical insurance

CREATE VIEW hospitalmanagement.PatientDetailsWithInsuranceView1 AS

SELECT

p.PatientID,

p.FirstName,

p.LastName,

p.DateOfBirth,

p.ContactNumber,

b.TotalAmount,

mi.insuranceLimit,

mi.ExpiryDate

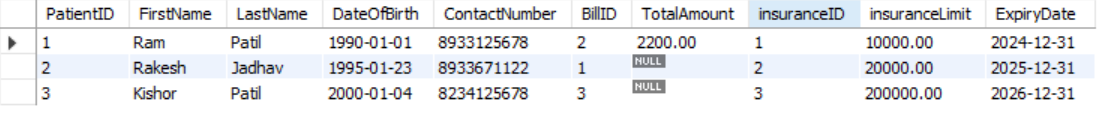
FROM

hospitalmanagement.patient p

LEFT JOIN hospitalmanagement.bills b ON p.PatientID = b.PatientID

LEFT JOIN hospitalmanagement.medicalinsurance mi ON p.PatientID = mi.PatientID;

select \* from hospitalmanagement.PatientDetailsWithInsuranceView1;



* View for fetching details about patient appointment, diagnosis and lab test details.

CREATE VIEW PatientDetailsWithHistoryView AS

SELECT

p.patientID,

p.FirstName,

p.LastName,

p.DateOfBirth,

p.ContactNumber,

a.appointmentID,

a.appointmentDate,

d.DiagnosisID,

d.Details,

m.medicationName,

lt.LabTestID,

lt.TestName,

lt.Result

FROM

hospitalmanagement.patient p

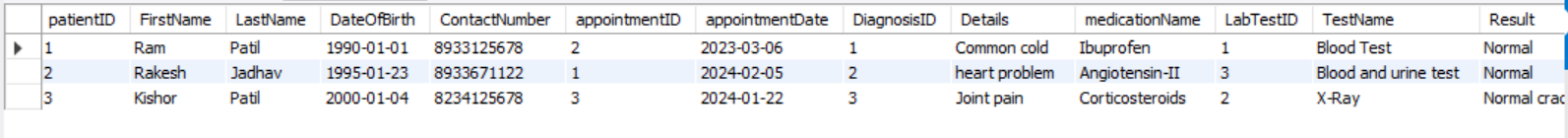
LEFT JOIN hospitalmanagement.appointment a ON p.PatientID = a.PatientID

LEFT JOIN hospitalmanagement.diagnosis d ON p.PatientID = d.PatientID

LEFT JOIN hospitalmanagement.medication m ON p.PatientID = m.PatientID

LEFT JOIN hospitalmanagement.labtest lt ON p.PatientID = lt.PatientID;

select \* from PatientDetailsWithHistoryView;



Q2. Write a report on your understanding of Rendering patterns and Design Patterns. Mention and elaborate where a particular Rendering pattern is applicable and is well suited for which use case.­­­

Ans:- Rendering and Design patterns are fundamental concepts in Computer Science. Where Rendering patterns tells us about how the user interface is rendered and how data is presented to users and Design patterns are nothing but reusable solutions to common problems that can arise during software development, these are not complete finished codes but rather templates.

* **Rendering Patterns : -**

Rendering patterns refers to different approaches for generating and displaying the contents on a website or web application. It shows us the way in which the HTML, CSS and JavaScript code is all processed and rendered in a web application or website. Different rendering patterns are used to achieve different performance and user experience goals.

1. **Client-side Rendering (CSR) : -** The rendering process occurs on the client’s browser using JavaScript. The server only sends required HTML, CSS and JavaScript files to client browser and then browser fetches and renders data dynamically.
2. **Server-side Rendering (SSR) :** - In Server-side rendering pattern, the server renders all the required data itself, embed it with the given HTML and sent the pre-rendered HTML files to the client, it reduces client-side processing.
3. **Static Site Generation (SSG) : -** Static Site Generation pattern generates static HTML pages while building phase, eliminating the need for server-side or client-side processing during runtime.
4. **Rehydration : -** A combination of server-side rendering and client-side rendering. Initially, a small, static HTML payload is sent, and client-side JavaScript rehydrates it with dynamic data.
5. **Incremental Static Generation : -** The Incremental Static Generation pattern was introduced as an upgrade to SSG, It allows you to update existing pages and add new ones by pre-rendering a subset of pages in the background even while fresh requests for pages are coming in and without rebuilding the entire site.

* **Design Patterns : -**

A design pattern in programming is a reusable solution to a common problem that occurs during software design and development. It provides a structured approach to solving specific design or implementation issues, allowing developers to create more maintainable, flexible and scalable code. Each pattern is like a blueprint that you can customize to solve a particular design problem in our code.

Design patterns can be categorized into three main types : -

1. **Creational Patterns : -** These patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the application.

Examples – Singleton, Factory Method, Abstract Factory, etc.

1. **Structural Patterns : -** These patterns explain how to assemble objects and classes into larger structures while keeping these structures flexible and efficient.

Examples – Adapter, Decorator, Bridge, etc.

1. **Behavioral Patterns : -** These patterns are concerned with the interaction and responsibility of objects.

Examples – Command, Iterator, Mediator, Interpreter, etc.

* **Client-side Rendering (CSR) : -**

The Client-side Rendering pattern is mostly applicable on Single-page applications which are highly interactive and dynamic. Sites with complex user interaction that change often based on user actions and sites for which SEO is not a top priority and initial load time can be addressed through optimization techniques. This pattern is ideal where real-time updates and user-specific content are essential.

**Use-cases :-** Ideal for dynamic web applications where interactivity and frequent updates are crucial like Single-page applications as Gmail, Facebook, etc.

* **Server-side Rendering Pattern (SSR) : -**

The Server-side Rendering Pattern is well suited for content-heavy websites with frequent updates. The applications which requires SEO optimization for better search engine visibility and where the initial load speed is important.

**Use-case : -** Ideal for applications where content security, dynamic updates and fast initial load is required like news portals, e-commerce websites, etc.

* **Static Site Generation (SSG) : -**

Suitable for websites with relatively static content. Best for content driven websites with infrequent updates and offers fast initial load time and low server load and content that doesn’t require frequent user interaction.

**Use-case : -** Ideal for websites where frequent content updation is not required and efficient CDN distribution is must. Best for websites offering fast initial load times like blogs, portfolios websites.

* **Rehydration : -**

Suitable for balancing initial page load speed and dynamic content updates. Ideal for applications that want a quick stary by sending a basic page structure first and applications where user interactivity can be given to the components as needed.

**Use-case : -** Websites with dynamic content that can be updated on the client-side after initial load.

* **Incremental Static Generation : -**

Suitable for websites where there is not much content which needs to be dynamically updated and rest content is static and where certain pages can be regenerated at runtime based on user interactions.

**Use-case : -** Websites with a mix of static and dynamic content like News websites with real-time updates where certain pages needs to reflect the latest information.