NAME: PRERANA V. NIWATE

Milestone Project 1

Question 1:

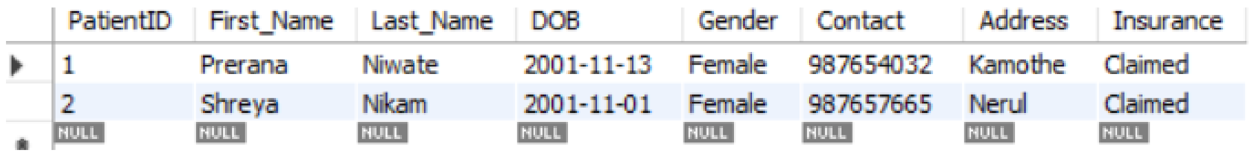
- **Create a DB Schema for Hospital Management System.**
- **Define the schema along with the constraints indicating the relationships between the entities.**
- **Be sure to make use of the database concepts like Views, Relationships, Indexing, Stored Procedure and triggers.**
- **Indicate the Normalization form being used in the schema defined and why you chose to keep it that particular normal form.**
- **Once your schema is well defined, choose any Relational Database system (MySQL, MariaDB, etc) and practically implement the schema so that you are able to perform at least the following operations.**
- **HMS should be capable to recognize already registered patients and user roles.**
- **Write necessary queries to register new user roles and personas**
- **Write necessary queries to add to the list of diagnosis of the patient tagged by date.**
- **Write necessary queries to fetch required details of a particular patient.**
- **Write necessary queries to prepare bill for the patient at the end of checkout.**
- **Write necessary queries to fetch and show data from various related tables (Joins)**
- **Optimize repeated read operations using views/materialized views.**
- **Optimize read operations using indexing wherever required. (Create index on at least 1 table)**
- **Try optimizing bill generation using stored procedures.**
- **Add necessary triggers to indicate when patients medical insurance limit has expired.**

Patients Table:
Table Name: patient_details
- PatientID (Primary Key)
- First_Name
- Last_Name
- DOB
- Gender
- Contact
- Address

NAME: PRERANA V. NIWATE

- Insurance

```
CREATE TABLE `hospital`.`patient_details` (
 `PatientID` INT NOT NULL,
 `First_Name` VARCHAR(45) NOT NULL,
 `Last_Name` VARCHAR(45) NOT NULL,
 `DOB` DATE NOT NULL,
 `Gender` VARCHAR(45) NOT NULL,
 `Contact` INT(10) NOT NULL,
 `Address` VARCHAR(100) NOT NULL,
 `Insurance` VARCHAR(45) NOT NULL,
 PRIMARY KEY (`PatientID`));
```



PatientID	First_Name	Last_Name	DOB	Gender	Contact	Address	Insurance
1	Prerana	Niwate	2001-11-13	Female	987654032	Kamothe	Claimed
2	Shreya	Nikam	2001-11-01	Female	987657665	Nerul	Claimed
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Doctors Table:

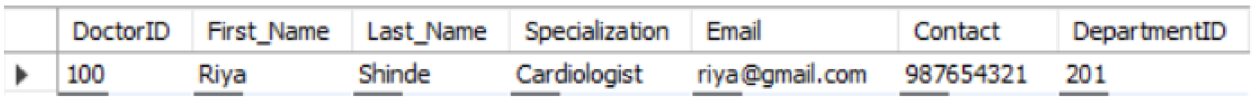
Table Name: doctors_details

- DoctorID (Primary Key)
- First_Name
- Last_Name
- Specialization
- Email
- Contact
- DepartmentID (Foreign Key referencing Departments Table)

```
CREATE TABLE `hospital`.`doctors_details` (

 `Doctor_ID` INT NOT NULL,

 `First_Name` VARCHAR(45) NOT NULL,

 `Last_Name` VARCHAR(45) NOT NULL,

 `Specialization` VARCHAR(45) NOT NULL,

 `Email` VARCHAR(45) NOT NULL,
```

NAME: PRERANA V. NIWATE

```
 `Contact` INT(10) NOT NULL,

 `DepartmentID` INT NOT NULL,

 PRIMARY KEY (`Doctor_ID`),

 INDEX `DepartmentID_idx` (`DepartmentID` ASC) VISIBLE,

 CONSTRAINT `DepartmentID`

   FOREIGN KEY (`DepartmentID`)

   REFERENCES `hospital`.`department_details` (`DepartmentID`)

   ON DELETE NO ACTION

   ON UPDATE NO ACTION);
```



DoctorID	First_Name	Last_Name	Specialization	Email	Contact	DepartmentID
100	Riya	Shinde	Cardiologist	riya@gmail.com	987654321	201

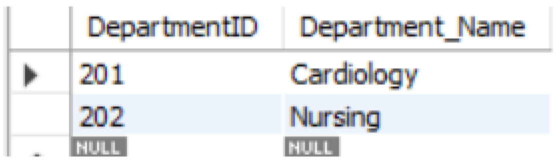
<u>Departments Table:</u>

<u>Table Name:</u> department_details

- DepartmentID (Primary Key)
- DepartmentName

```
CREATE TABLE `hospital`.`department_details` (

 `DepartmentID` INT NOT NULL,

 `Department_Name` VARCHAR(45) NOT NULL,

 PRIMARY KEY (`DepartmentID`));
```

NAME: PRERANA V. NIWATE



	DepartmentID	Department_Name
▶	201	Cardiology
	202	Nursing
	NULL	NULL

Appointments Table:

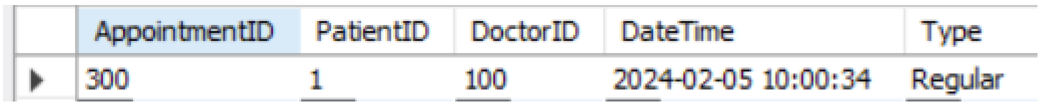
Table Name: appointment_details

- AppointmentID (Primary Key)
- PatientID (Foreign Key referencing Patients Table)
- DoctorID (Foreign Key referencing Doctors Table)
- Date_Time
- Type (e.g., regular check-up, follow-up, etc.)

```
CREATE TABLE `hospital`.`appointment_details` (

 `AppointmentID` INT NOT NULL,

 `PatientID` INT NOT NULL,

 `DoctorID` INT NOT NULL,

 `DateTime` DATETIME NOT NULL,

 `Type` VARCHAR(45) NOT NULL,

 PRIMARY KEY (`AppointmentID`),

 INDEX `PatientID_idx` (`PatientID` ASC) VISIBLE,

 CONSTRAINT `PatientID`

   FOREIGN KEY (`PatientID`)

   REFERENCES `hospital`.`patient_details` (`PatientID`)

   ON DELETE NO ACTION

   ON UPDATE NO ACTION,
```

NAME: PRERANA V. NIWATE

```
 CONSTRAINT
   FOREIGN KEY (`DoctorID`)
   REFERENCES `hospital`.`doctors_details` (`DrID`)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION

);
```

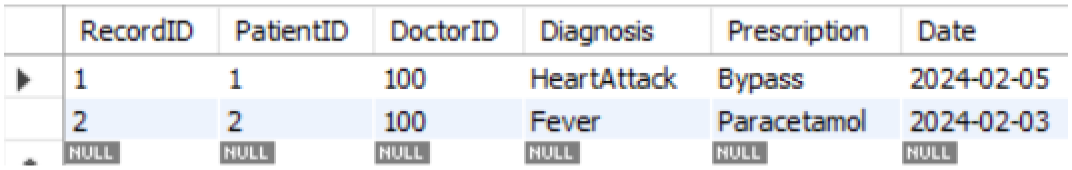


AppointmentID	PatientID	DoctorID	DateTime	Type
300	1	100	2024-02-05 10:00:34	Regular

Medical Records:
Table Name: medical_records

- Record_ID (Primary Key)
- Patient_ID (Foreign Key referencing Patients table)
- Doctor_ID (Foreign Key referencing Doctors table)
- Diagnosis
- Prescription
- Date

```
CREATE TABLE `hospital`.`medical_records` (
  `RecordID` INT NOT NULL,
  `PatientID` INT NOT NULL,
  `DoctorID` INT NOT NULL,
  `Diagnosis` VARCHAR(45) NOT NULL,
  `Prescription` VARCHAR(45) NOT NULL,
  `Date` DATE NOT NULL,
  PRIMARY KEY (`RecordID`),
  INDEX `DoctorID_idx` (`DoctorID` ASC) VISIBLE,
  INDEX `PatientID_idx` (`PatientID` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`PatientID`)
    REFERENCES `hospital`.`patient_details` (`PatientID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT
    FOREIGN KEY (`DoctorID`)
    REFERENCES `hospital`.`doctors_details` (`Doctor_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
);
```



RecordID	PatientID	DoctorID	Diagnosis	Prescription	Date
1	1	100	HeartAttack	Bypass	2024-02-05
2	2	100	Fever	Paracetamol	2024-02-03
NULL	NULL	NULL	NULL	NULL	NULL

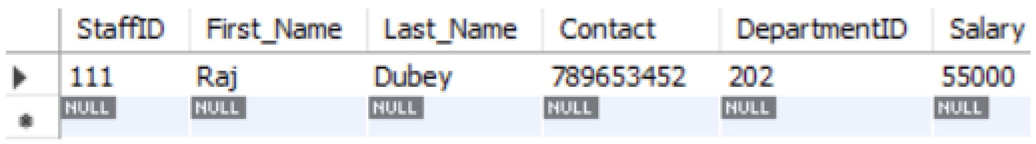
Staff Details:

Table Name: staff_details

- StaffID(Primary key)
- First_Name
- Last_Name
- Contact
- DepartmentID(Foreign Key referencing Department Table)
- Salary

```
CREATE TABLE `hospital`.`staff_details` (
  `StaffID` INT NOT NULL,
  `First_Name` VARCHAR(45) NOT NULL,
  `Last_Name` VARCHAR(45) NOT NULL,
  `Contact` INT NOT NULL,
  `DepartmentID` INT NOT NULL,
  `Salary` INT NOT NULL,
  PRIMARY KEY (`StaffID`),
  INDEX `departement_idx` (`DepartmentID` ASC) VISIBLE,
  CONSTRAINT `departement`
    FOREIGN KEY (`DepartmentID`)
    REFERENCES `hospital`.`department_details` (`DepartmentID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

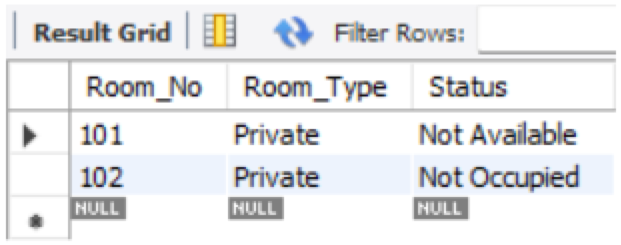
NAME: PRERANA V. NIWATE



StaffID	First_Name	Last_Name	Contact	DepartmentID	Salary
111	Raj	Dubey	789653452	202	55000
NULL	NULL	NULL	NULL	NULL	NULL

<u>ROOM TABLE</u>
<u>Table Name:</u> room_details
Room_No (Primary key)
Room_Type
Status

```
CREATE TABLE `hospital`.`room_details` (
  `Room_No` INT NOT NULL,
  `Room_Type` VARCHAR(45) NOT NULL,
  `Status` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Room_No`));
```



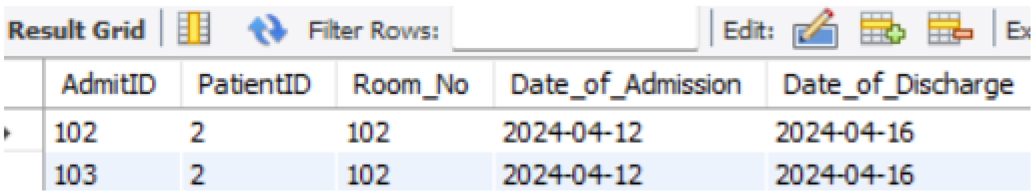
Room_No	Room_Type	Status
101	Private	Not Available
102	Private	Not Occupied
NULL	NULL	NULL

<u>ADMITTED_PATIENT TABLE</u>
<u>Table Name:</u> admitpatient_details
AdmitID
PatientID
Room_No
Date_of_Admission
Date_of_Discharge

```
CREATE TABLE `hospital`.`admitpatient_details` (
  `AdmitID` INT NOT NULL,
  `PatientID` INT NOT NULL,
  `Room_No` INT NOT NULL,
  `Date_of_Admission` DATE NOT NULL,
  `Date_of_Discharge` DATE NOT NULL,
  PRIMARY KEY (`AdmitID`),
  INDEX `patientId_idx` (`PatientID` ASC) VISIBLE,
  INDEX `room_no_idx` (`Room_No` ASC) VISIBLE,
```

NAME: PRERANA V. NIWATE

```
  CONSTRAINT `ptid`
    FOREIGN KEY (`PatientID`)
    REFERENCES `hospital`.`patient_details` (`PatientID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `room_no`
    FOREIGN KEY (`Room_No`)
    REFERENCES `hospital`.`room_details` (`Room_No`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```



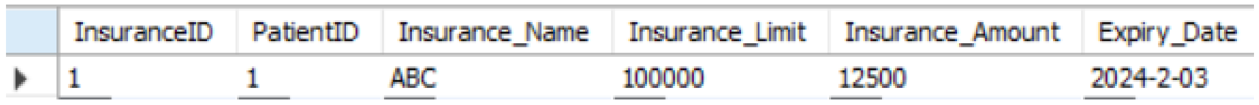
AdmitID	PatientID	Room_No	Date_of_Admission	Date_of_Discharge
102	2	102	2024-04-12	2024-04-16
103	2	102	2024-04-12	2024-04-16

INSURANCE TABLE
Table name: insurance_details
InsuranceID(Primary key)
PatientID
Insurance_Name
Insurance_Limit
Insurance_Amount
Expiry_Date

```
CREATE TABLE `hospital`.`insurance_details` (
  `InsuranceID` INT NOT NULL,
  `PatientID` INT NOT NULL,
  `Insurance_Name` VARCHAR(45) NOT NULL,
  `Insurance_Limit` INT NOT NULL,
  `Insurance_Amount` INT NOT NULL,
  `Expiry_Date` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`InsuranceID`),
  INDEX `PID_idx` (`PatientID` ASC) VISIBLE,
  CONSTRAINT `PID`
    FOREIGN KEY (`PatientID`)
    REFERENCES `hospital`.`patient_details` (`PatientID`)
```

NAME: PRERANA V. NIWATE

```
ON DELETE NO ACTION
ON UPDATE NO ACTION);
```



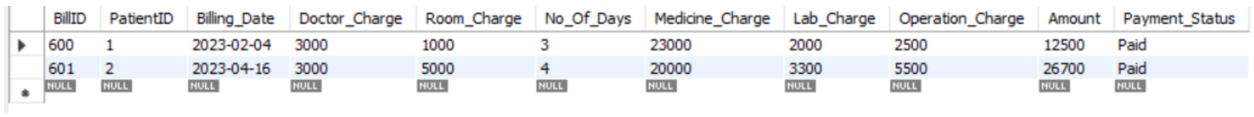
InsuranceID	PatientID	Insurance_Name	Insurance_Limit	Insurance_Amount	Expiry_Date
1	1	ABC	100000	12500	2024-2-03

Billing Table:

Table Name: billing_details

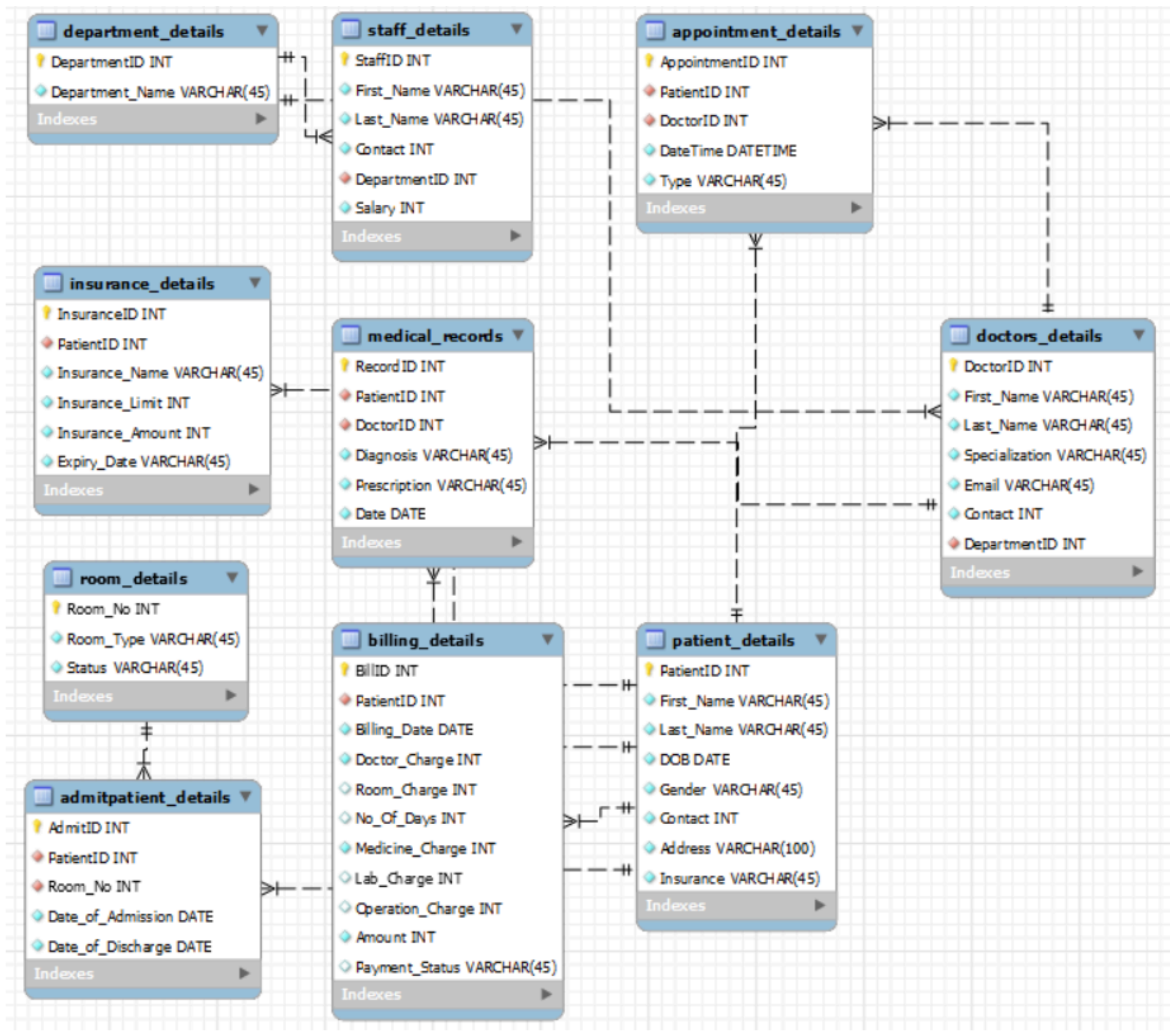
- BillID (Primary Key)
- PatientID (Foreign Key referencing Patients Table)
- BillingDate
- Doctor_Charge
- Room_Charge
- No_of_Days
- Operation_Charge
- Lab_Charge
- BillAmount
- PaymentStatus

```
CREATE TABLE `hospital`.`billing_details` (

 `BillID` INT NOT NULL,

 `PatientID` INT NOT NULL,

 `Billing_Date` DATE NOT NULL,

 `Doctor_Charge` INT NOT NULL,

 `Room_Charge` INT NULL,

 `Medicine_Charge` INT NOT NULL,

 `No_of_Days` INT NULL,

 `Lab_Charge` INT NULL,
 `Operation_Charge` INT NULL,

 `Amount` INT NOT NULL,

 `Payment_Status` VARCHAR(45) NOT NULL,

 PRIMARY KEY (`BillID`),

 INDEX `bill_patientID_idx` (`PatientID` ASC) VISIBLE,

 CONSTRAINT `bill_patientID`

  FOREIGN KEY (`PatientID`)

  REFERENCES `hospital`.`patient_details` (`PatientID`)

  ON DELETE NO ACTION

  ON UPDATE NO ACTION);
```



BillID	PatientID	Billing_Date	Doctor_Charge	Room_Charge	No_Of_Days	Medicine_Charge	Lab_Charge	Operation_Charge	Amount	Payment_Status
600	1	2023-02-04	3000	1000	3	23000	2000	2500	12500	Paid
601	2	2023-04-16	3000	5000	4	20000	3300	5500	26700	Paid
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

NAME: PRERANA V. NIWATE



Normalization is the process of reducing Data Redundancy.

All the tables are in 2NF(Second Normal Form) because it gives us the clear understanding of data as well as dependency of one table on another.
Partial dependency is not present as well as the table satisfies all rules of 1NF.

Write necessary queries to register new user roles and personas

```
Insert into patient_details(PatientID,First_Name, Last_Name, DOB, Gender,
Contact, Address, Insurance) Values(1,'Prerana','Niwate',
'2001-11-13','Female',987654032,'Kamothe','Claimed');
```

NAME: PRERANA V. NIWATE

```
Insert into doctors_details(DoctorID,First_Name, Last_Name, Specialization,
Email, Contact, DepartmentID) Values(100,'Riya','Shinde',
'Cardiologist','riya@gmail.com',987654321,201);

Insert into staff_details(111, ‘Raj’,’Dubey’, 789653452, 202,5500

Insert into department_details(DepartmentID,Department_Name)
Values(201,'Cardiology');

Insert into appointment_details(AppointmentID,PatientID, DoctorID, DateTime,
Type) Values(300,1,100,'2024-02-05 10:00:34','Regular');

insert into room_details (Room_No, Room_Type,Status) Value(102, 'Private',
'Available');

insert into admitpatient_details
(AdmitID,PatientID,Room_No,Date_of_Admission,Date_of_Discharge)
Value(102, '2', '102', '2024-04-12','2024-04-16');

Insert into medical_records(RecordID,PatientID, DoctorID, Diagnosis,
Prescription,Date) Values(2, 2, 100,'Fever','Paracetamol','2024-2-03');

Insert into insurance_details (InsuranceID,PatientID, Insurance_Name,
Insurance_Limit, Insurance_Amount,Expiry_Date) Values(1, 1,
'ABC',100000,'12500','2024-2-03');

Insert into billing_details(BillID,PatientID, Billing_Date,Doctor_Charge,
Room_Charge, No_of_Days, Medicine_Charge, lab_Charge, Operation_Charge,
Amount, Payment_Status)
values(602,2,'2023-04-16',3000,5000,4,20000,3300,5500, 0, 'Paid');
```

Write necessary queries to add to the list of diagnosis of the patient tagged by date.
→Insert into medical_records (RecordID,PatientID, DoctorID, Diagnosis, Prescription,Date) Values(2, 2, 100,'Fever','Paracetamol','2024-2-03');

Write necessary queries to fetch required details of a particular patient
→ Select * from patient_details where patientID = 1;

NAME: PRERANA V. NIWATE

Optimize read operations using indexing whenever required(create index on atleast 1 table)

```
        Create Index idx_appointment ON appointment_details(Date Time)
```

STORED PROCEDURE FOR SCHEDULING APPOINTMENT

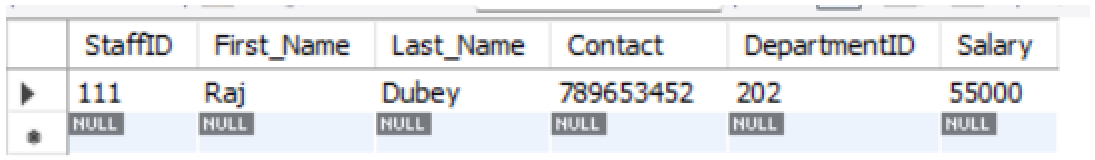
```
DELIMITER //

CREATE PROCEDURE schedule_appointment(
  aAppointmentID int,
  aPatientID int,
  aDoctorID int,
  aDateTime datetime,
  aType varchar(45)
)
BEGIN
   IF EXISTS(SELECT * FROM patient_details WHERE PatientID = aPatientID) AND
EXISTS(SELECT * FROM doctors_details WHERE DoctorID = aDoctorID)
  THEN
                INSERT INTO appointment_details(AppointmentID,PatientID, DoctorID,
AppointmentDateTime, Type)
        VALUES(aAppointmentID,aPatientID, aDoctorID, aDateTime, aType);
  ELSE
      SELECT 'INVALID PatientID or DoctorID';
  END IF;
END

DELIMITER ;
```

STORED PROCEDURE FOR UPDATE SALARY

```
DELIMITER //

CREATE PROCEDURE update_salary (StaffID INT, NewSalary INT)
BEGIN
    UPDATE staff_details
    SET Salary = NewSalary
    WHERE StaffID = StaffID;
END//
```

NAME: PRERANA V. NIWATE

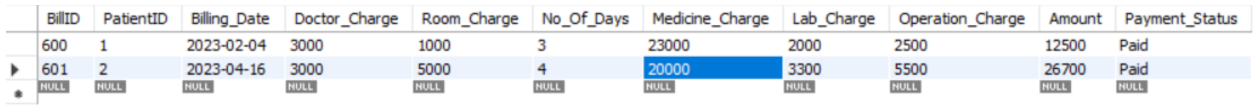
```
DELIMITER ;
CALL update_salary(111, 55000);
```



StaffID	First_Name	Last_Name	Contact	DepartmentID	Salary
111	Raj	Dubey	789653452	202	55000
NULL	NULL	NULL	NULL	NULL	NULL

STORED PROCEDURE FOR UPDATE PAYMENT_STATUS

```
DELIMITER //

CREATE PROCEDURE update_payStatus (BillID INT, NewStatus Varchar(45))
BEGIN
   UPDATE billing_details
   SET Payment_Status = NewStatus
   WHERE BillID = BillID;
END//

DELIMITER ;

CALL update_payStatus(600, 'Paid');
```



BillID	PatientID	Billing_Date	Doctor_Charge	Room_Charge	No_Of_Days	Medicine_Charge	Lab_Charge	Operation_Charge	Amount	Payment_Status
600	1	2023-02-04	3000	1000	3	23000	2000	2500	12500	Paid
601	2	2023-04-16	3000	5000	4	20000	3300	5500	26700	Paid
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

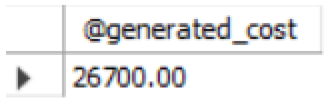
Write necessary queries to prepare bill for the patient at the end of checkout

→ STORED PROCEDURE FOR GENERATING BILL

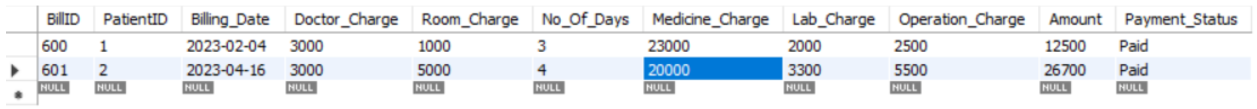
```
DELIMITER //

CREATE PROCEDURE genere_bill(IN _PatientID INT, OUT TotalCost DECIMAL(10,2))
BEGIN
   DECLARE v_DoctorCharge DECIMAL(10,2);
   DECLARE v_RoomCharge DECIMAL(10,2);
   DECLARE v_OpCharge DECIMAL(10,2);
   DECLARE v_LabCharge DECIMAL(10,2);
   DECLARE v_NumOfDays TINYINT;
```

NAME: PRERANA V. NIWATE

```
        SELECT Doctor_Charge, Room_Charge, Operation_Charge, Lab_Charge,
NO_OF_DAYS
  INTO v_DoctorCharge, v_RoomCharge, v_OpCharge, v_LabCharge, v_NumOfDays
  FROM billing_details
  WHERE PatientID = _PatientID;

   SET TotalCost := v_DoctorCharge + v_RoomCharge + v_OpCharge + v_LabCharge *
v_NumOfDays;

  UPDATE billing_details
  SET Amount = TotalCost
  WHERE PatientID = _PatientID;

 END//

DELIMITER ;
CALL genere_bill(2,@generated_cost);
select @generated_cost;
```



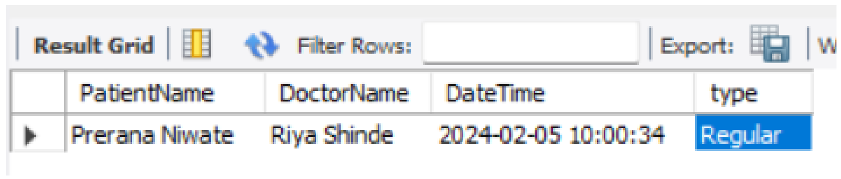
@generated_cost
26700.00



BillID	PatientID	Billing_Date	Doctor_Charge	Room_Charge	No_Of_Days	Medicine_Charge	Lab_Charge	Operation_Charge	Amount	Payment_Status
600	1	2023-02-04	3000	1000	3	23000	2000	2500	12500	Paid
601	2	2023-04-16	3000	5000	4	20000	3300	5500	26700	Paid
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

VIEW FOR PATIENT APPOINTMENT

```
CREATE VIEW patient_appointment AS
SELECT CONCAT(p.First_Name, ' ', p.Last_Name) AS PatientName,
     CONCAT(d.First_Name, ' ', d.Last_Name) AS DoctorName,
     a.DateTime,
     a.type
FROM patient_details p
JOIN appointment_details a ON p.PatientID = a.PatientID
JOIN doctors_details d ON a.DoctorID = d.DoctorID;
```



PatientName	DoctorName	DateTime	type
Prerana Niwate	Riya Shinde	2024-02-05 10:00:34	Regular

NAME: PRERANA V. NIWATE

Write necessary queries to fetch and show data from various related tables[joins]

```
SELECT p.PatientID, pa.PatientName, ra.Room_Type, ma.Diagnosis, ma.Prescription,
ap.Date_of_Admission, ap.Date_of_Discharge
FROM admitpatient_details ap
JOIN patient_details p ON ap.PatientID = p.PatientID
JOIN medical_records ma ON p.PatientID = ma.PatientID
JOIN (
   SELECT PatientID, CONCAT(First_Name, ' ', Last_Name)  AS PatientName, Contact,
Address
   FROM patient_details
) pa ON p.PatientID = pa.PatientID
JOIN room_details ra ON ap.Room_No = ra.Room_No
ORDER BY ap.Date_of_Admission DESC;
```

TRIGGER FOR DUPLICATE ENTRY

```
DELIMITER $$

CREATE TRIGGER duplicate_id
BEFORE INSERT ON patient_details
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT * FROM patient_details WHERE PatientID = NEW.PatientID)
THEN
        SET MESSAGE_TEXT = "Duplicate Entry";
   END IF;
END$$

DELIMITER ;
```

TRIGGER TO UPDATE ROOM AVAILABILTY

```
DELIMITER $$

CREATE TRIGGER trg_UpdateRoomStatus
AFTER INSERT ON admitpatient_details FOR EACH ROW
BEGIN
   UPDATE room_details
   SET Status = 'Not Available''
   WHERE Room_No = NEW.Room_No;
END$$

DELIMITER ;
DELIMITER //
```

TRIGGER FOR INSURANCE EXPIRY

```
DELIMITER $$

CREATE TRIGGER InsuranceExpiryTrigger
AFTER INSERT ON insurance_details
FOR EACH ROW
BEGIN
   IF New.Expiry_Date < CURDATE() THEN
      INSERT INTO expired_insurance (InsuranceID,PatientID, Expiry_Date)
      VALUES (NEW.InsuranceID,NEW.PatientID, NEW.Expiry_Date);
   END IF;
END$$

DELIMITER //
```

Question 2:

Write a report on your understanding of rendering and design patterns. Mention and elaborate where a particular rendering pattern is applicable and is well suited for which use case.

Rendering Patterns

A rendering pattern refers to the way in which the HTML, CSS, and JavaScript code is all processed and rendered in a web application or website.
The most common rendering patterns in web development are:

1. Server-side rendering (SSR): In SSR, the web server generates the HTML content of a web page on the server-side and sends it to the client's browser. This approach can improve initial loading times and SEO (search engine optimization) but can be

slower for dynamic content. When a website's JavaScript is rendered on the website's server, a fully rendered page is sent to the client and the client's JavaScript bundle engages and enables the Single Page Application framework to operate. SSR is commonly used for content-heavy websites, where fast initial loads and effective SEO optimization are crucial, such blogs and news websites.

2. Client-side rendering (CSR): In CSR, the client's browser generates the HTML content of a web page on the client-side using JavaScript. This approach can provide a fast and interactive user experience but can be slower for initial loading times and bad for SEO. Websites using CSR are Facebook , Twitter, Gmail. The server sends the first HTML file and any necessary JavaScript files when a user requests a page. The client then reloads the page only when required by using JavaScript to update the page as needed. CSR is used whenever Web applications that demand a high level of interaction, such social media platforms or e-commerce websites, frequently use CSR.
3. Static site generation (SSG): In SSG, the HTML content of a web page is generated at build time and served to the client as a static file. This approach can provide excellent performance and security but can be less flexible for dynamic content. The server creates a static HTML file and any necessary JavaScript files for a page on user request. At that point, the client doesn't have to wait for any more server requests to show the page. The page can be updated as needed by using the client-side JavaScript. SSG is frequently used for static websites that need to load quickly at first and have some interactivity, like landing pages and portfolios. More sophisticated apps that don't need real-time updates can also use it.

Design Patterns

Design patterns provide solutions to common problems encountered during software development. They offer proven architectural approaches that can improve code quality, reduce complexity, and promote reusability.

Factory Method Pattern: The Factory Method pattern involves creating objects through an interface rather than directly instantiating them within client code. A factory method pattern could be applied in a hospital management system to simplify the creation of different types of appointments without duplicating code.

NAME: PRERANA V. NIWATE

MVC (Model-View-Controller): MVC is also one of the rendering patterns. In the Hospital management system the MVC rendering pattern can be applied to the user interface for managing patient records, appointments, and billing. For example, when a user interacts with the system to schedule an appointment, the controller would handle the input, update the model (patient and appointment data), and then update the view to reflect the changes.

Observer Pattern:In this pattern, objects subscribe to updates from other objects called subjects. When the subject undergoes state changes, observers receive notifications about these events.When the subject undergoes state changes, observers receive notifications about these events. This pattern is commonly used in real-time systems, such as stock market applications
The stock object (subject) updates its price. The stock object notifies all registered observers (portfolio trackers, stock ticker displays, analytics modules) about the change. Each observer receives the notification and updates its state accordingly. For example:

- The portfolio tracker may update the value of the "XYZ Corp" stock in the user's portfolio.
- The stock ticker display may update the price of "XYZ Corp" being displayed.
- The analytics module may perform calculations or generate reports based on the new price.

Singleton Pattern: The Singleton pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. It is commonly used in scenarios where only a single instance of a class is needed, such as in logging, thread pools, and database connections. In Hospital Management System the Singleton pattern could be applied to ensure that there is a single, centralized instance for managing the system's configuration settings, or for handling access to a shared resource such as a patient's medical records.

Repository Pattern: A repository serves as an intermediary for the application logic layer and the data storage layer in the Repository Pattern. It offers a single point of entry for data retrieval and manipulation, separating the complex structure of the data storage layer from the rest of the application. Because of this, it's simpler to modify the data storage layer without having an impact on the application overall. The Repository Pattern is frequently used in software development for applications that require interaction with a data source, such a database. This design allows the business logic of the application to be maintained in addition to the specifics of data storage and retrieval. This facilitates the process of switching between data sources without interfering with the functionality of the program, and it also makes the code easier to maintain and test.

Dependency Injection Pattern: It is used to increase the code's flexibility, testability, and maintainability while reducing coupling between components. Instead of being built inside a component, dependencies are injected into it using the dependency injection pattern. Because of this, it is simpler to change or modify dependencies without affecting the component itself because components can be constructed independently of their dependencies.

Code reusability can be achieved by design patterns. By following a standard structure, developers can easily reuse code in different parts of an application or even in different applications altogether. Design patterns are tried and tested solutions to common problems. By using these patterns, developers can avoid common errors and pitfalls that might arise when writing code from scratch.