Hospital Management System

Introduction :

Hospital Management Systems (HMS) play a pivotal role in modern healthcare by efficiently managing and organizing a multitude of tasks within a healthcare facility. These systems are designed to streamline processes, enhance patient care, and ensure smooth administrative operations. A well-structured database is the backbone of any Hospital Management System, providing the foundation for storing, retrieving, and managing vast amounts of healthcare-related data.

In this database project, we embark on creating a comprehensive DB Schema for a Hospital Management System, incorporating essential database concepts like Views, Relationships, Indexing, Stored Procedures, and Triggers. The objective is to design a robust and scalable system that not only caters to the current needs of healthcare management but also adapts to future requirements seamlessly.

Objectives :

- Design and implement a robust database schema.
- Utilize database concepts for optimized data retrieval and manipulation.
- Implement a practical solution using a Relational Database System (MySQL).
- Enable HMS to recognize registered patients and user roles.
- Develop necessary queries for user role registration, diagnosis addition, patient details retrieval, and bill preparation.
- Implement optimizations using views/materialized views, indexing, and stored procedures.
- Introduce triggers to indicate when a patient's medical insurance limit has expired.

Database Schema Design:

Entities:

- ➔ Patients:
    - ◆ PatientID (Primary Key), FirstName, LastName, DateOfBirth, Gender, ContactNumber, Address, InsuranceLimit

- UserRoles:
    - RoleID (Primary Key), RoleName
- Users:
    - UserID (Primary Key), Username, Password, RoleID (Foreign Key)
- Diagnosis:
    - DiagnosisID (Primary Key), PatientID (Foreign Key), DiagnosisDate, DiagnosisDetails
- Bills:
    - BillID (Primary Key), PatientID (Foreign Key), BillAmount, CheckoutDate

Constraints:

- Patients Table:
    - PatientID is the primary key.
- UserRoles Table:
    - RoleID is the primary key.
- Users Table:
    - UserID is the primary key.
    - RoleID is a foreign key referencing RoleID in UserRoles.
- Diagnosis Table:
    - DiagnosisID is the primary key.
    - PatientID is a foreign key referencing PatientID in Patients.
- Bills Table:
    - BillID is the primary key.
    - PatientID is a foreign key referencing PatientID in Patients.

Relationships:

- Patients and Diagnosis:
    - One-to-Many relationship between Patients and Diagnosis based on PatientID.
- Patients and Bills:
    - One-to-Many relationship between Patients and Bills based on PatientID.
- Users and UserRoles:
    - One-to-Many relationship between Users and UserRoles based on RoleID.

Views:

- ➔ Patients_view:
  - ◆ A simple view containing all columns from the Patients table.
- ➔ UserRoles_view:
  - ◆ A simple view containing all columns from the UserRoles table.
- ➔ Users_view:
  - ◆ A simple view containing all columns from the Users table.
- ➔ DiagnosisView:
  - ◆ A simple view containing all columns from the Diagnosis table.
- ➔ BillsView:
  - ◆ A simple view containing all columns from the Bills table.
- ➔ PatientDetails:
  - ◆ A view combining patient details with diagnosis and billing information.

Indexing:

- ➔ An index named idx_PatientID is created on the PatientID column of the Patients table for faster retrieval of patient information.

Stored Procedure: GenerateBill

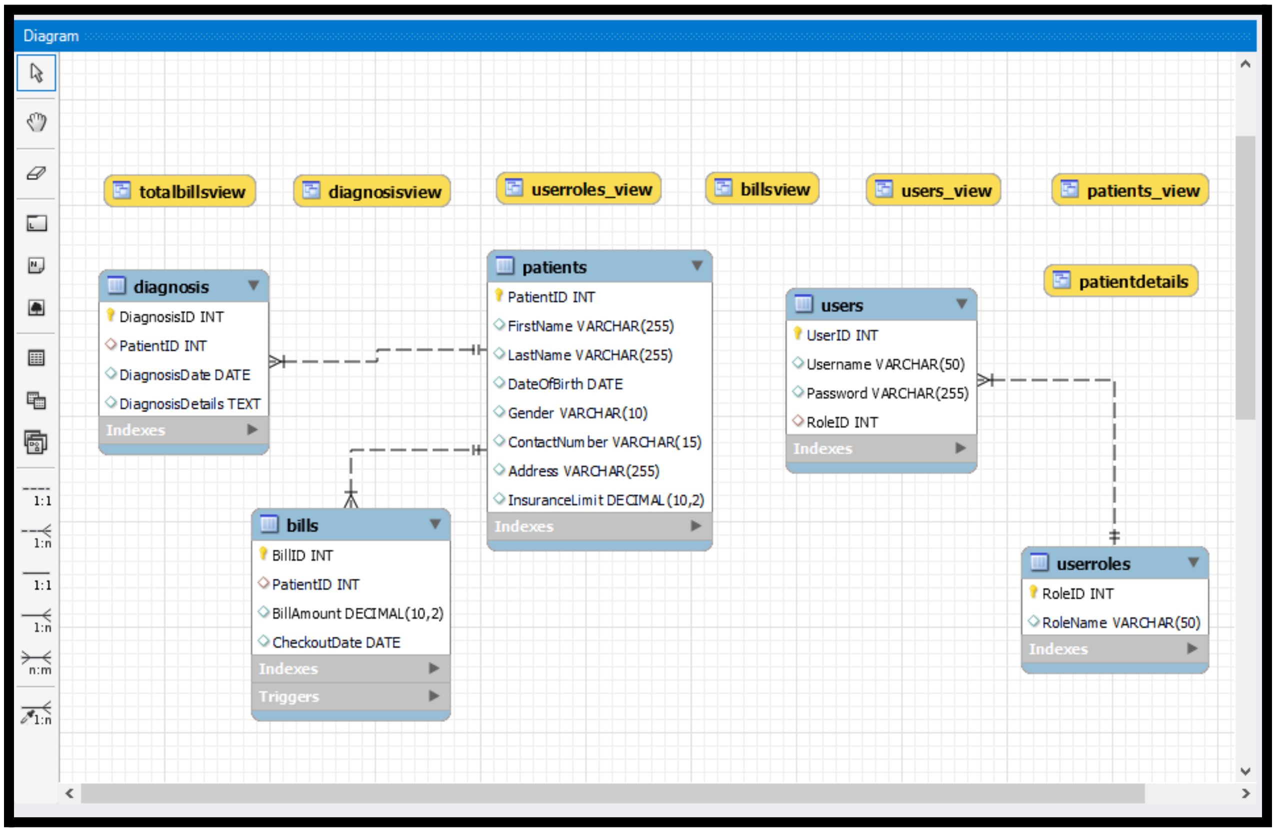
- ➔ Description:
  - ◆ GenerateBill is like a smart helper in the hospital system. It makes billing for patients easier. It puts the patient's ID, the bill amount, and today's date into the Bills list. This smart way helps keep billing organized and quick.
- ➔ Usage:
  - ◆ When you want to bill a patient, just tell GenerateBill the patient's ID and the bill amount. It will take care of adding a new record in the Bills list, making billing easy and in one place.

Trigger : UpdateInsuranceLimit

- ➔ Description:
  - ◆ The UpdateInsuranceLimit trigger ensures accurate insurance details. When a new bill is added, it automatically updates the patient's insurance limit in the Patients list by deducting the new bill amount (NEW.BillAmount). This smart update keeps the insurance limit current at all times.
- ➔ Usage:

◆ Without any manual input, the trigger works seamlessly whenever a new bill is added. It takes care of updating the Patients list, ensuring that insurance details stay accurate and timely after each new bill insertion.

ER Diagram :



Normalize Schema:

First Normal Form (1NF):

- Remove redundancy by splitting the Name field into FirstName and LastName.
- Each column should hold atomic values.
- UserRoles Table ,Users Table,Diagnosis Table,Bills Table already in 1NF

Second Normal Form (2NF):

In 2NF, all table information relies fully on the primary key, preventing partial dependencies. For instance, in the Patients table, details like Date of Birth link solely to the PatientID, ensuring a complete connection.

Third Normal Form (3NF):

In 3NF, transitive dependencies are eliminated, enhancing data integrity. The schema is structured this way to reduce redundancy, enabling efficient data management.

Code :

```
-- Create the 'hospital_management' database
CREATE DATABASE IF NOT EXISTS hospital_management;

-- Switch to the 'hospital_management' database
USE hospital_management;
-- Create Tables
CREATE TABLE Patients (
   PatientID INT PRIMARY KEY,
   FirstName VARCHAR(255),
   LastName VARCHAR(255),
   DateOfBirth DATE,
   Gender VARCHAR(10),
   ContactNumber VARCHAR(15),
   Address VARCHAR(255),
   InsuranceLimit DECIMAL(10,2)
);

CREATE TABLE UserRoles (
   RoleID INT PRIMARY KEY,
   RoleName VARCHAR(50)
);

CREATE TABLE Users (
   UserID INT PRIMARY KEY,
   Username VARCHAR(50),
   Password VARCHAR(255),
   RoleID INT,
   FOREIGN KEY (RoleID) REFERENCES UserRoles(RoleID)
);

CREATE TABLE Diagnosis (
   DiagnosisID INT PRIMARY KEY,
   PatientID INT,
   DiagnosisDate DATE,
   DiagnosisDetails TEXT,
   FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

CREATE TABLE Bills (
   BillID INT PRIMARY KEY,
   PatientID INT,
   BillAmount DECIMAL(10,2),
   CheckoutDate DATE,
   FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

-- Insert Values in tables

INSERT INTO Patients (PatientID, FirstName, LastName, DateOfBirth, Gender,
ContactNumber, Address, InsuranceLimit)
VALUES
(1, 'John', 'Doe', '1990-05-15', 'Male', '1234567890', '123 Main St', 5000.00),
(2, 'Jane', 'Smith', '1985-08-21', 'Female', '9876543210', '456 Oak St', 8000.00);

INSERT INTO UserRoles (RoleID, RoleName)
VALUES
(1, 'Admin'),
(2, 'Doctor'),
(3, 'Nurse');

INSERT INTO Users (UserID, Username, Password, RoleID)
VALUES
(1, 'admin_user', 'admin_password', 1),
(2, 'doctor_user', 'doctor_password', 2),
(3, 'nurse_user', 'nurse_password', 3);
```

```
INSERT INTO Diagnosis (DiagnosisID, PatientID, DiagnosisDate, DiagnosisDetails)
VALUES
(1, 1, '2022-01-10', 'Flu'),
(2, 2, '2022-02-05', 'Broken Arm');

INSERT INTO Bills (BillID, PatientID, BillAmount, CheckoutDate)
VALUES
(1, 1, 100.00, '2022-01-15'),
(2, 2, 500.00, '2022-02-20');

-- Show tables

SELECT * FROM Patients;
SELECT * FROM UserRoles;
SELECT * FROM Users;
SELECT * FROM Diagnosis;
SELECT * FROM Bills;

-- simple view for each table

Create view Patients_view as select * from Patients;
select * from Patients_view;

Create view UserRoles_view as select * from UserRoles;
select * from UserRoles_view;

Create view UserRoles_view as select * from UserRoles;
select * from UserRoles_view;

Create view Users_view as select * from Users;
select * from Users_view;

CREATE VIEW DiagnosisView AS SELECT * FROM Diagnosis;
SELECT * FROM DiagnosisView;

CREATE VIEW BillsView AS SELECT * FROM Bills;
SELECT * FROM BillsView;

-- Combines patient details with diagnosis and billing information.
```

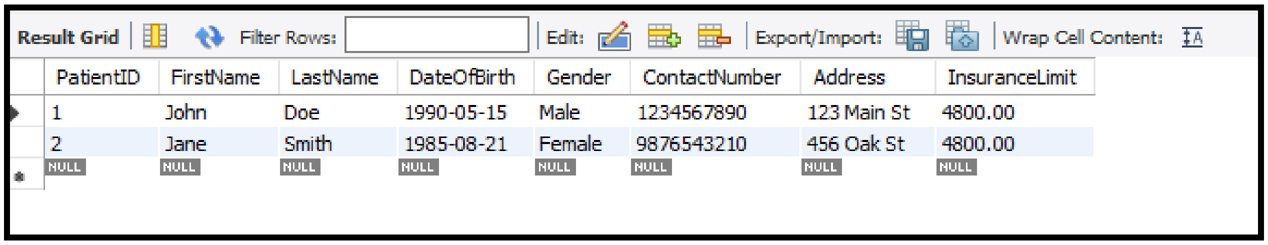
```
CREATE VIEW PatientDetails AS
SELECT Patients.PatientID, FirstName, LastName, DateOfBirth, Gender, ContactNumber,
Address, InsuranceLimit, DiagnosisDetails, BillAmount, CheckoutDate
FROM Patients
LEFT JOIN Diagnosis ON Patients.PatientID = Diagnosis.PatientID
LEFT JOIN Bills ON Patients.PatientID = Bills.PatientID;

select * from PatientDetails;

-- An index on the PatientID column for faster retrieval of patient information.
CREATE INDEX idx_PatientID ON Patients(PatientID);

--  Write necessary queries to register new user roles and personas

select * from UserRoles;
INSERT INTO UserRoles (RoleID, RoleName) VALUES(4, 'interns');

select * from Users;
INSERT INTO Users (userID,Username, Password, RoleID) VALUES (4,'Intern_user',
'Intern_Pass', 4);

SELECT * FROM Users WHERE Username = 'Intern_user';

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

SELECT * FROM Diagnosis;
INSERT INTO Diagnosis (DiagnosisID, PatientID, DiagnosisDate, DiagnosisDetails)
VALUES (3, 1, '2022-02-10', 'Allergy');

select * from Diagnosis where PatientID = 1 and DiagnosisID=3 ;

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

SELECT * FROM Patients WHERE PatientID = 1 ;
select * from PatientDetails where PatientID = 1;

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

select * from Bills where PatientID =1;
```

```
-- Create a view to store the total bill amount for each patient
CREATE VIEW TotalBillsView AS
SELECT PatientID, SUM(BillAmount) AS TotalBill
FROM Bills
GROUP BY PatientID;

select * from Bills;

-- Update the Patients table with the total bill amount
UPDATE Patients
SET InsuranceLimit = InsuranceLimit - IFNULL((SELECT TotalBill FROM TotalBillsView
WHERE TotalBillsView.PatientID = Patients.PatientID), 0)
WHERE PatientID = 1;

SET SQL_SAFE_UPDATES = 0;

UPDATE Patients
SET InsuranceLimit = InsuranceLimit - IFNULL((SELECT TotalBill FROM TotalBillsView
WHERE TotalBillsView.PatientID = Patients.PatientID), 0);

-- Check the updated Patients table
SELECT * FROM Patients;

select * from TotalBillsView where PatientID=1;

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

SELECT Patients.PatientID, FirstName, LastName, DiagnosisDetails, BillAmount,
CheckoutDate
FROM Patients
LEFT JOIN Diagnosis ON Patients.PatientID = Diagnosis.PatientID
LEFT JOIN Bills ON Patients.PatientID = Bills.PatientID;

-- Optimize repeated read operations using views/materialized views.

SELECT * FROM Patients_view;
SELECT * FROM UserRoles_view;
SELECT * FROM Users_view;
SELECT * FROM DiagnosisView;
SELECT * FROM BillsView;
SELECT * FROM PatientDetails;
select * from TotalBillsView;

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

SHOW INDEX FROM Patients;

-- Try optimizing bill generation using stored procedures.

DELIMITER //

CREATE PROCEDURE GenerateBill(IN patient_id INT, IN bill_amount DECIMAL(10,2))
BEGIN
   INSERT INTO Bills (PatientID, BillAmount, CheckoutDate)
   VALUES (patient_id, bill_amount, CURDATE());
END //

DELIMITER ;

SHOW CREATE PROCEDURE GenerateBill;

select * from Bills;

-- Update the remaining insurance limit for the patient
      UPDATE Patients
      SET InsuranceLimit = 4800
      WHERE PatientID = PatientID;

         select * from Patients;
```

OUTPUT :

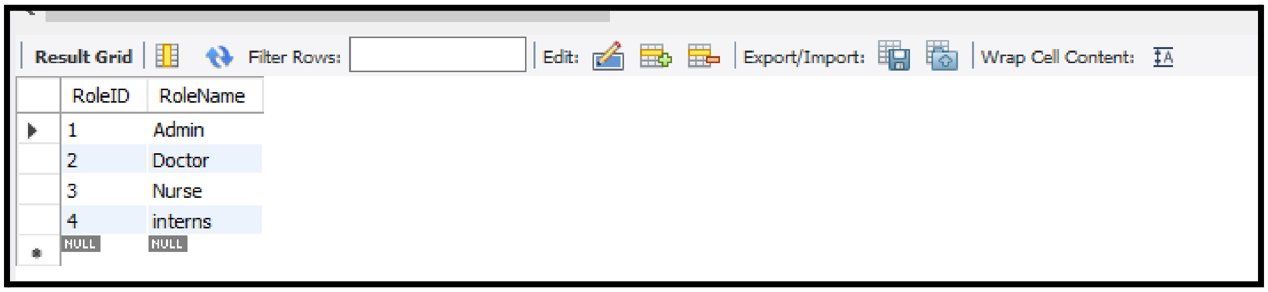
Tables:

- Patients :


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

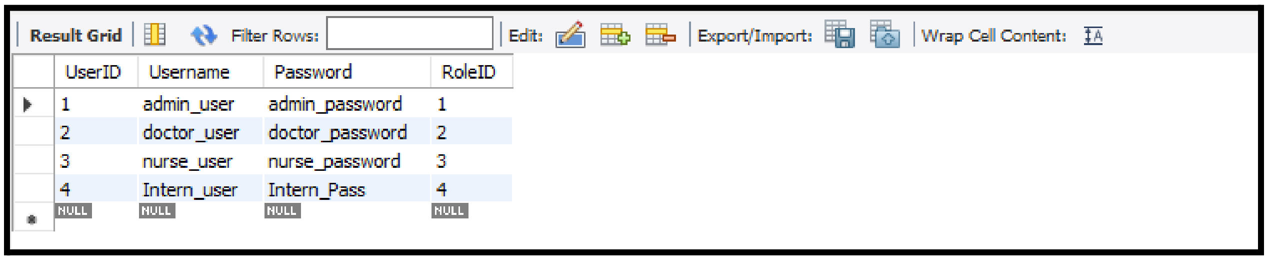
PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00
2	Jane	Smith	1985-08-21	Female	9876543210	456 Oak St	4800.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- UserRoles :


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

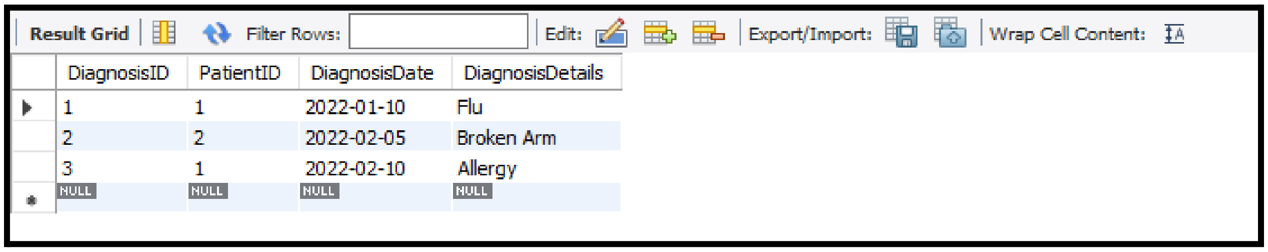
RoleID	RoleName
1	Admin
2	Doctor
3	Nurse
4	interns
NULL	NULL

- Users :


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

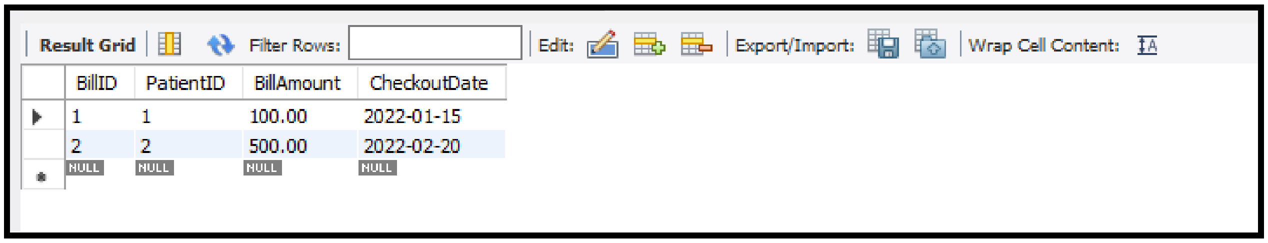
UserID	Username	Password	RoleID
1	admin_user	admin_password	1
2	doctor_user	doctor_password	2
3	nurse_user	nurse_password	3
4	Intern_user	Intern_Pass	4
NULL	NULL	NULL	NULL

- Diagnosis :


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

DiagnosisID	PatientID	DiagnosisDate	DiagnosisDetails
1	1	2022-01-10	Flu
2	2	2022-02-05	Broken Arm
3	1	2022-02-10	Allergy
NULL	NULL	NULL	NULL

- Bills :

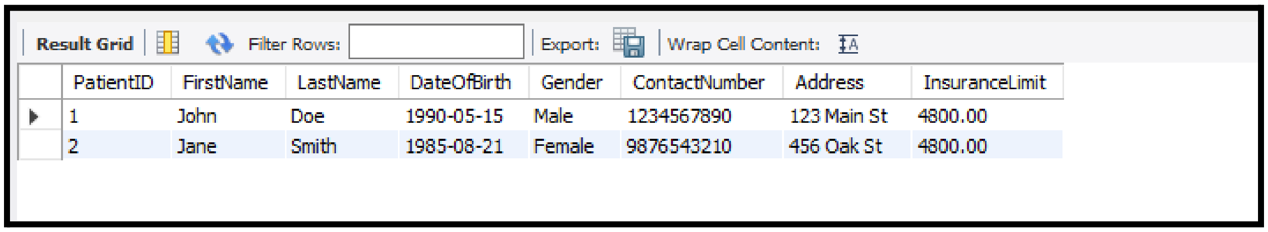


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

BillID	PatientID	BillAmount	CheckoutDate
1	1	100.00	2022-01-15
2	2	500.00	2022-02-20
NULL	NULL	NULL	NULL

Views :

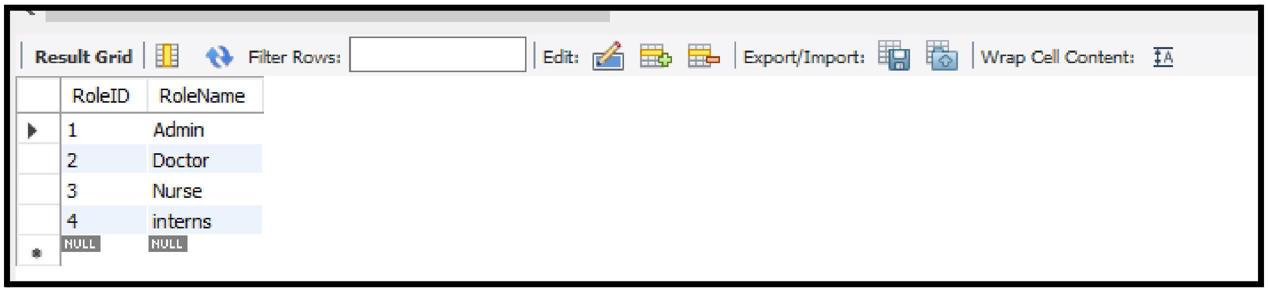
- Patient_View:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00
2	Jane	Smith	1985-08-21	Female	9876543210	456 Oak St	4800.00

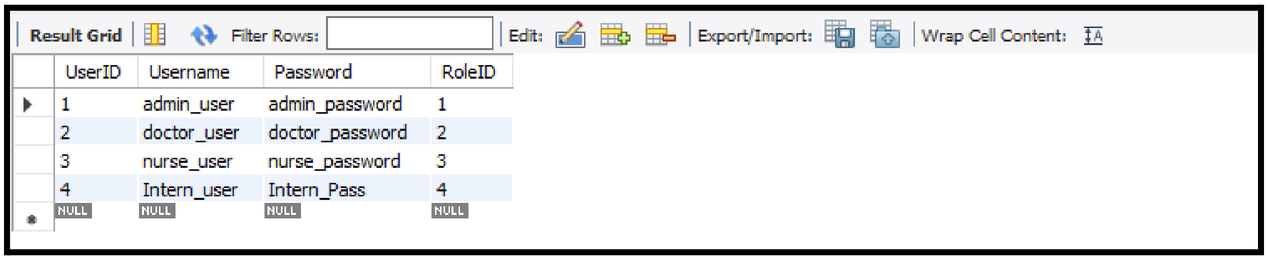
- UserRoles_View :



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

RoleID	RoleName
1	Admin
2	Doctor
3	Nurse
4	interns
NULL	NULL

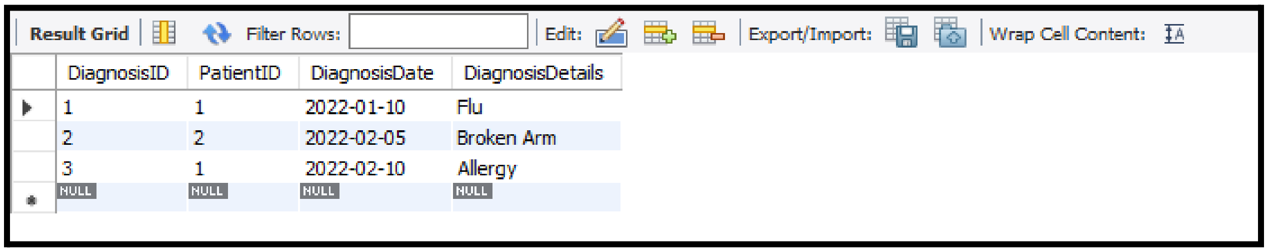
- Users_View :



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

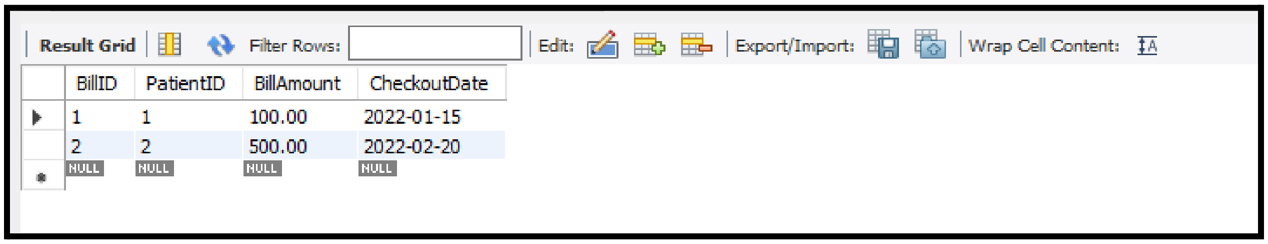
UserID	Username	Password	RoleID
1	admin_user	admin_password	1
2	doctor_user	doctor_password	2
3	nurse_user	nurse_password	3
4	Intern_user	Intern_Pass	4
NULL	NULL	NULL	NULL

- Diagnosis_View :



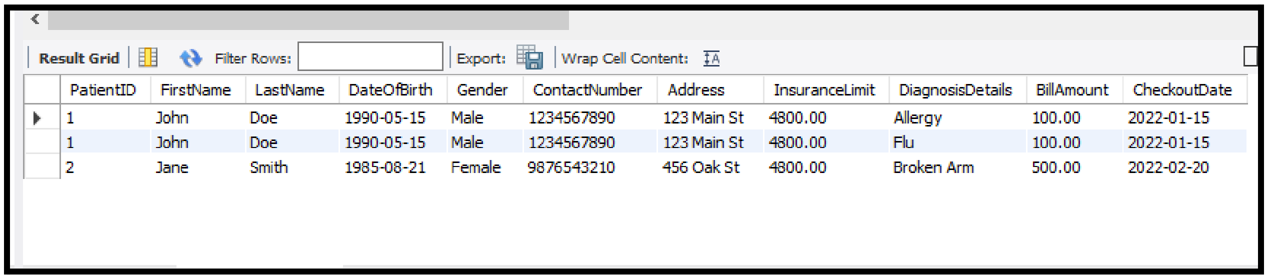
DiagnosisID	PatientID	DiagnosisDate	DiagnosisDetails
1	1	2022-01-10	Flu
2	2	2022-02-05	Broken Arm
3	1	2022-02-10	Allergy
NULL	NULL	NULL	NULL

- Bills_View :



BillID	PatientID	BillAmount	CheckoutDate
1	1	100.00	2022-01-15
2	2	500.00	2022-02-20
NULL	NULL	NULL	NULL

- Combines patient details with diagnosis and billing information:



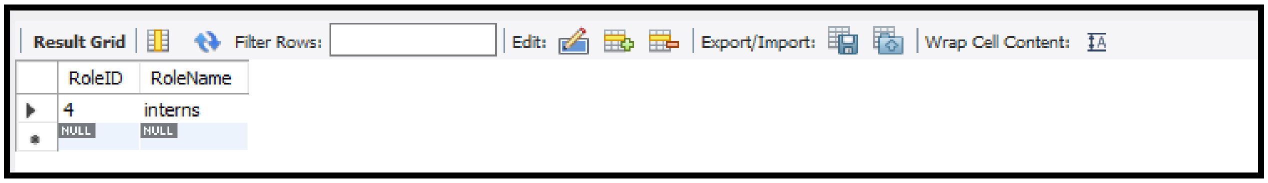
PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit	DiagnosisDetails	BillAmount	CheckoutDate
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Allergy	100.00	2022-01-15
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Flu	100.00	2022-01-15
2	Jane	Smith	1985-08-21	Female	9876543210	456 Oak St	4800.00	Broken Arm	500.00	2022-02-20

Queries :

1. **Write necessary queries to register new user roles and personas**

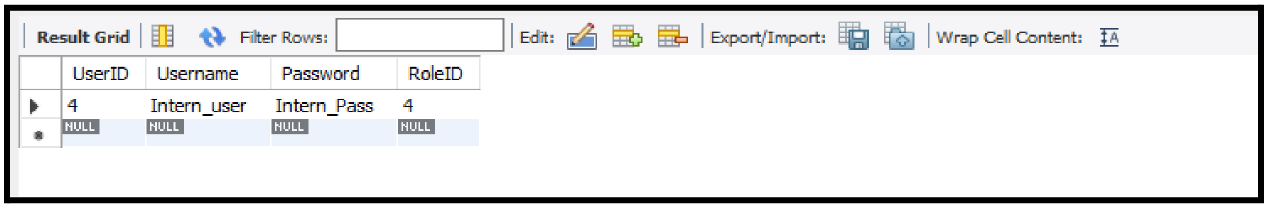
   Ans:

   1.1 UserRole


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

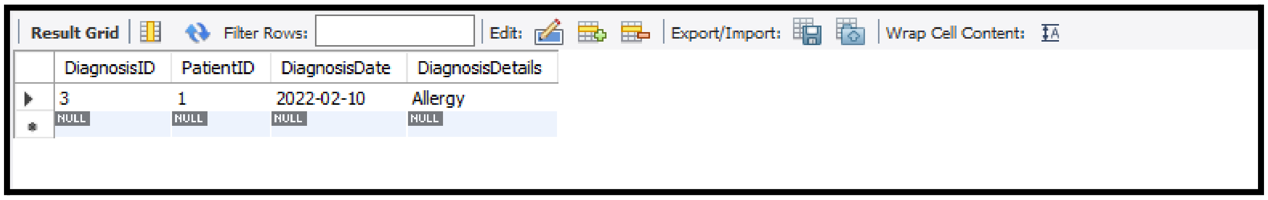
RoleID	RoleName
4	interns
NULL	NULL

1.2 Person


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

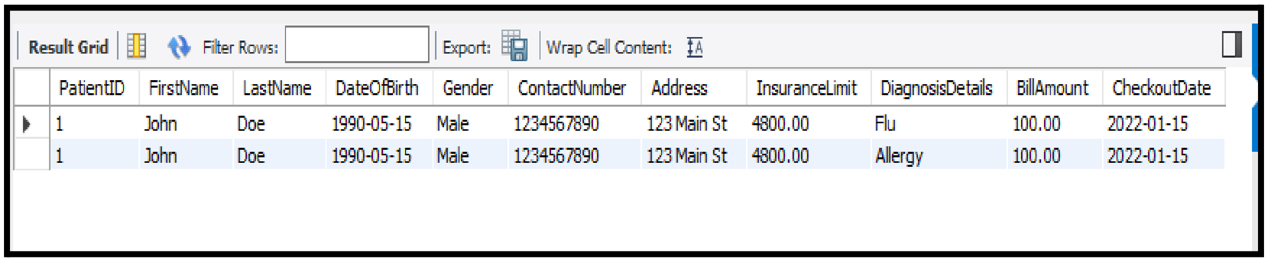
UserID	Username	Password	RoleID
4	Intern_user	Intern_Pass	4
NULL	NULL	NULL	NULL

2. **Write necessary queries to add to the list of diagnosis of the patient tagged by date.**
   Ans:


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

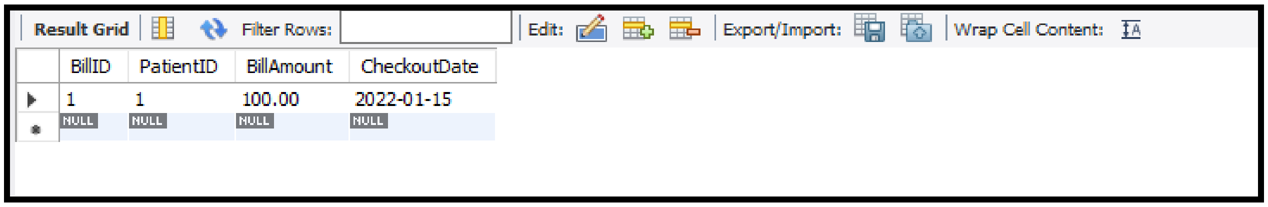
DiagnosisID	PatientID	DiagnosisDate	DiagnosisDetails
3	1	2022-02-10	Allergy
NULL	NULL	NULL	NULL

3. **Write necessary queries to fetch required details of a particular patient.**
   Ans:


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

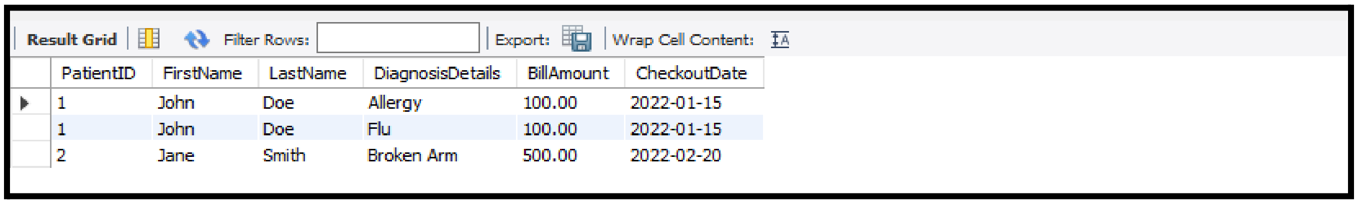
PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit	DiagnosisDetails	BillAmount	CheckoutDate
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Flu	100.00	2022-01-15
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Allergy	100.00	2022-01-15

4. **Write necessary queries to prepare a bill for the patient at the end of checkout.**
   Ans:


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

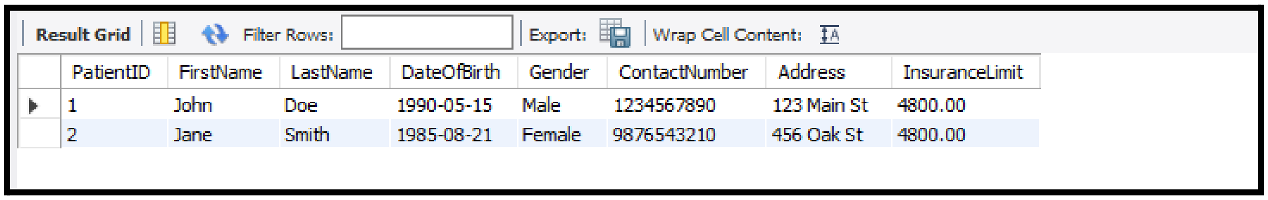
BillID	PatientID	BillAmount	CheckoutDate
1	1	100.00	2022-01-15
NULL	NULL	NULL	NULL

5. **Write necessary queries to fetch and show data from various related tables (Joins)**
 Ans:

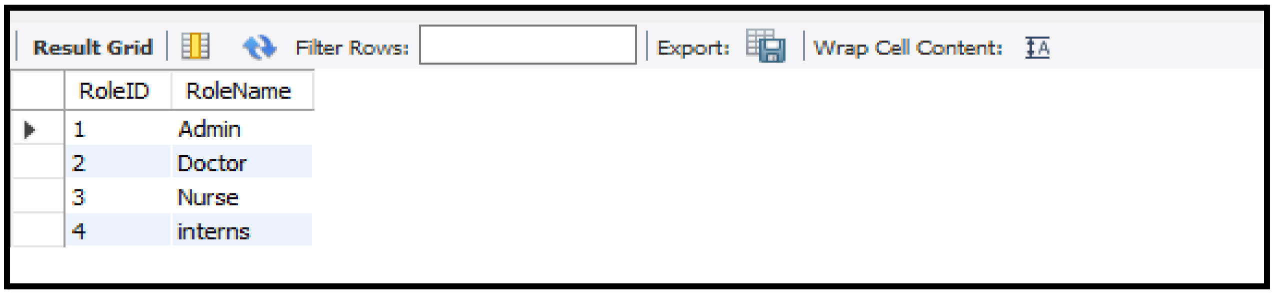

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PatientID	FirstName	LastName	DiagnosisDetails	BillAmount	CheckoutDate
1	John	Doe	Allergy	100.00	2022-01-15
1	John	Doe	Flu	100.00	2022-01-15
2	Jane	Smith	Broken Arm	500.00	2022-02-20

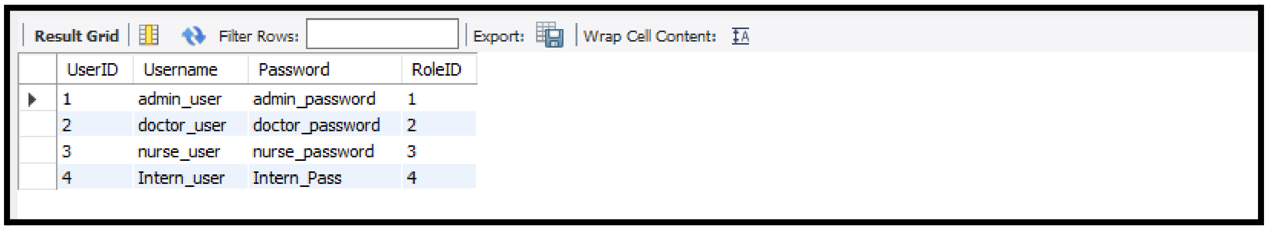
6. **Optimize repeated read operations using views/materialized views.**
 Ans:


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

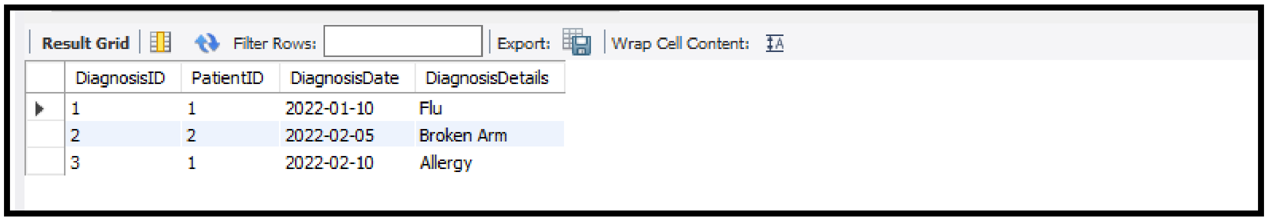
PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00
2	Jane	Smith	1985-08-21	Female	9876543210	456 Oak St	4800.00


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

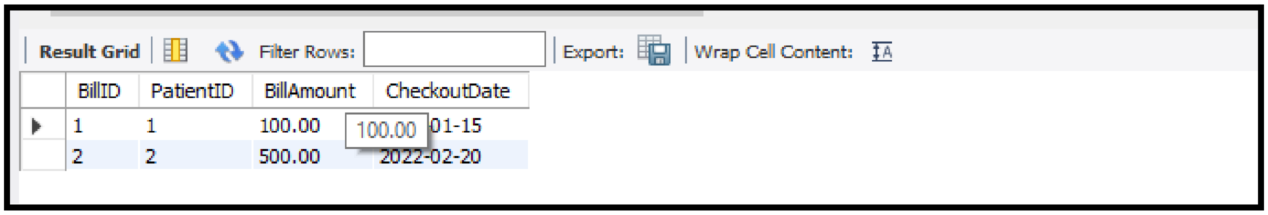
RoleID	RoleName
1	Admin
2	Doctor
3	Nurse
4	interns


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

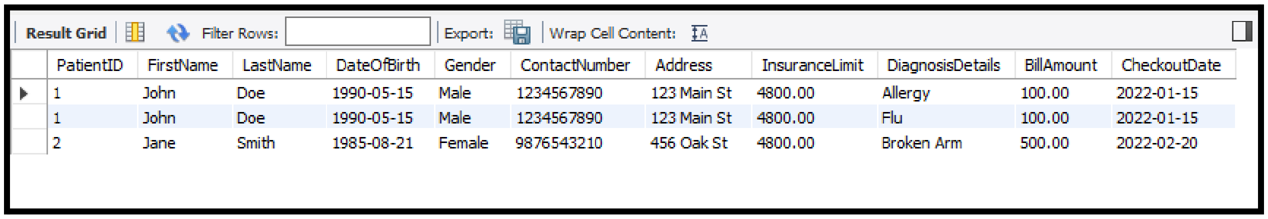
UserID	Username	Password	RoleID
1	admin_user	admin_password	1
2	doctor_user	doctor_password	2
3	nurse_user	nurse_password	3
4	Intern_user	Intern_Pass	4



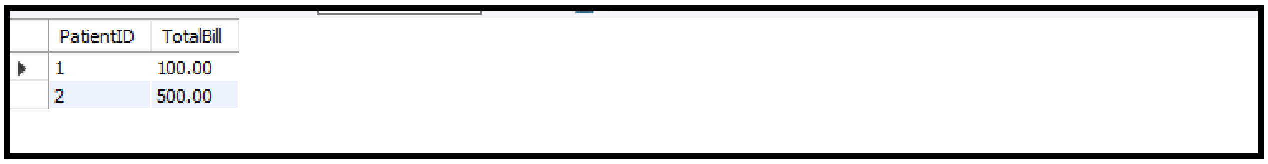
DiagnosisID	PatientID	DiagnosisDate	DiagnosisDetails
1	1	2022-01-10	Flu
2	2	2022-02-05	Broken Arm
3	1	2022-02-10	Allergy



BillID	PatientID	BillAmount	CheckoutDate
1	1	100.00	01-15
2	2	500.00	2022-02-20



PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit	DiagnosisDetails	BillAmount	CheckoutDate
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Allergy	100.00	2022-01-15
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00	Flu	100.00	2022-01-15
2	Jane	Smith	1985-08-21	Female	9876543210	456 Oak St	4800.00	Broken Arm	500.00	2022-02-20



PatientID	TotalBill
1	100.00
2	500.00

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

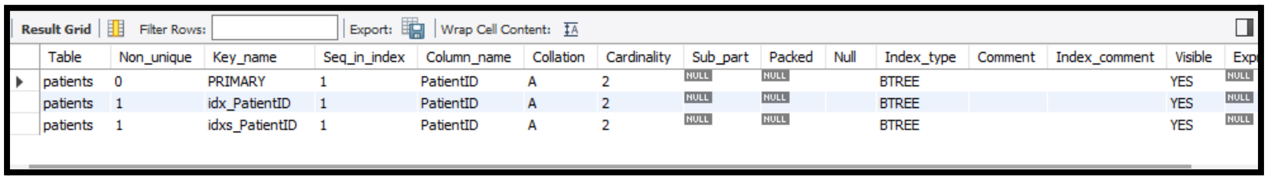
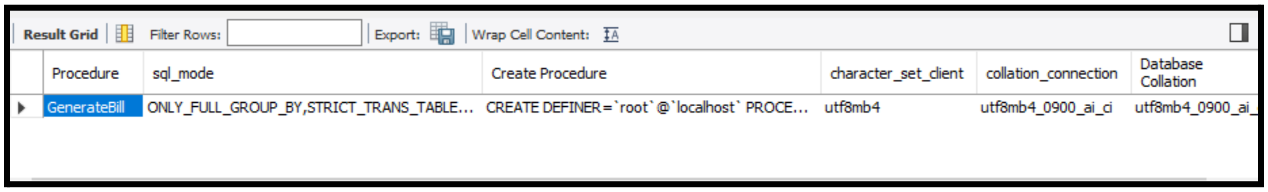


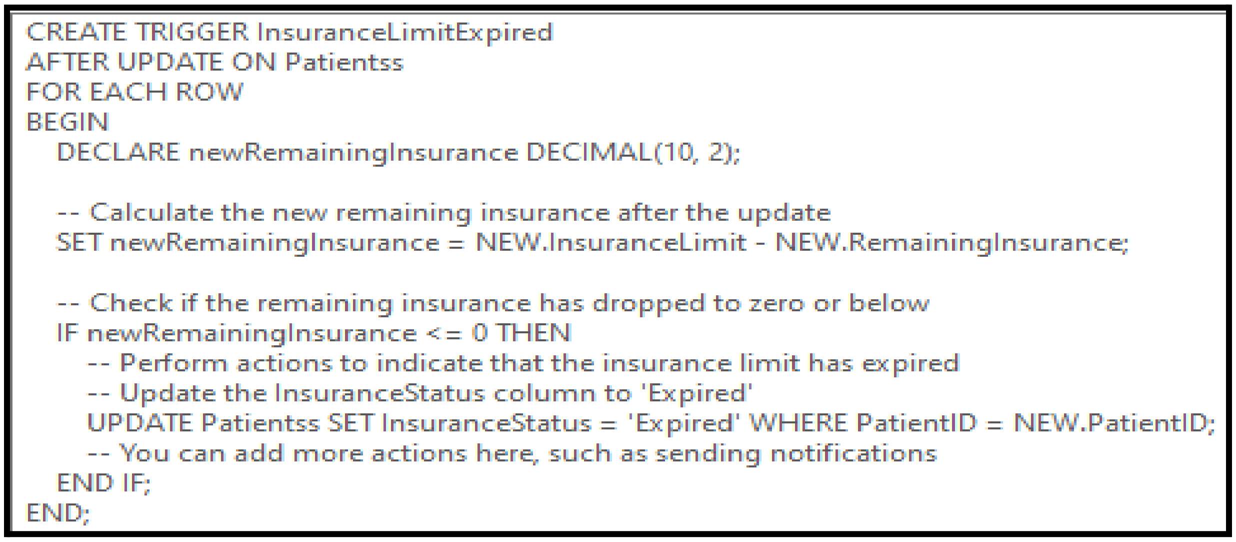
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Exp
patients	0	PRIMARY	1	PatientID	A	2	NULL	NULL		BTREE			YES	NULL
patients	1	idx_PatientID	1	PatientID	A	2	NULL	NULL		BTREE			YES	NULL
patients	1	idxs_PatientID	1	PatientID	A	2	NULL	NULL		BTREE			YES	NULL

8. **Try optimizing bill generation using stored procedures**
   Ans:

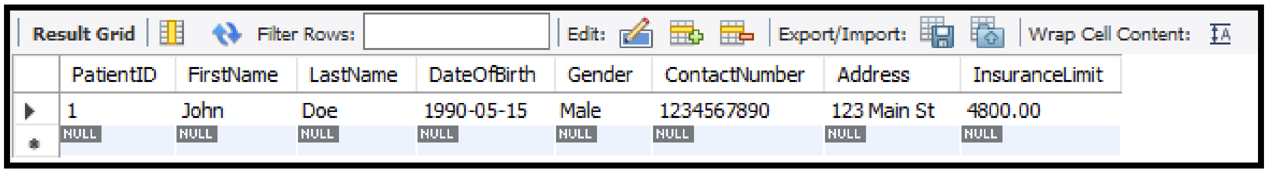


Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database Collation
GenerateBill	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	CREATE DEFINER=`root`@`localhost` PROCE...	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_

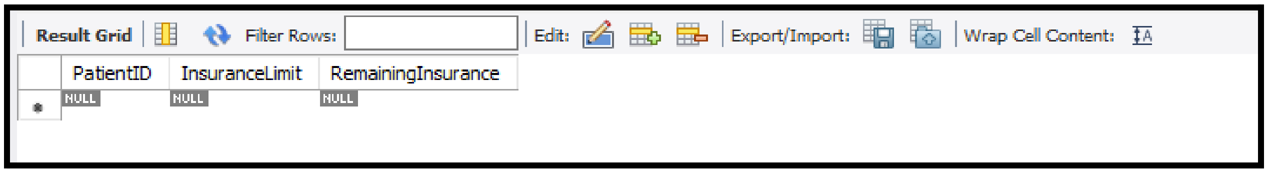
9. **Add necessary triggers to indicate when a patient's medical insurance limit has expired.**
   Ans:



```
CREATE TRIGGER InsuranceLimitExpired
AFTER UPDATE ON Patientss
FOR EACH ROW
BEGIN
  DECLARE newRemainingInsurance DECIMAL(10, 2);

  -- Calculate the new remaining insurance after the update
  SET newRemainingInsurance = NEW.InsuranceLimit - NEW.RemainingInsurance;

  -- Check if the remaining insurance has dropped to zero or below
  IF newRemainingInsurance <= 0 THEN
    -- Perform actions to indicate that the insurance limit has expired
    -- Update the InsuranceStatus column to 'Expired'
    UPDATE Patientss SET InsuranceStatus = 'Expired' WHERE PatientID = NEW.PatientID;
    -- You can add more actions here, such as sending notifications
  END IF;
END;
```



PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address	InsuranceLimit
1	John	Doe	1990-05-15	Male	1234567890	123 Main St	4800.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



PatientID	InsuranceLimit	RemainingInsurance
NULL	NULL	NULL

Conclusion :
In conclusion, the Hospital Management System project efficiently organizes patient data, streamlines billing processes, and ensures real-time insurance updates. Stored procedures, triggers, and views enhance system accuracy and efficiency.