Namrata Raju Patil

Milestone Project 1

Question 1

Create a DB Schema for Hospital Management System.

Table 1: Ward Table

Field	Data Type
wardId	int
wardName	varchar
wardDesc	text

Table 2: Room Table

Field	Data Type
roomId	int
wardId	int
roomNo	int

Table 3: Patient Table

Field	Data Type
patientId	int
roomId	int
firstname	varchar
lastname	varchar
email	varchar
phone	varchar
age	int
gender	Enum (‘Male’, ‘Female’, ’Other’)
address	varchar
city	varchar
zipcode	varchar
state	varchar
allergies	Text
disease	Text
otherhealthDisease	Text
healthInsurance	DATE
insuarnceStatus	varchar

Table 4: Treatment Details Table

Field	Data Type
treatmentDetailsId	int
treatmentName	varchar
description	text
price	Decimal (10, 2)

Table 5: Treatment Table

Field	Data Type
treatmentId	int
patientId	int
treatmentDetailsId	varchar
currentStatus	varchar
treatmentDate	Date

Table 6: Bill Table

Field	Data Type
paymentId	int
patientId	int
treatmentId	int
totalAmount	Decimal (10, 2)
amountPaid	Decimal (10, 2)
amounBalance	Decimal (10, 2)

Table 7: Doctor Table

Field	Data Type
doctorId	int
firstname	varchar
lastname	varchar
specificName	varchar
email	varchar
phone	varchar
age	int
gender	Enum (‘Male’, ‘Female’, ’Other’)
address	varchar
city	varchar
zipcode	varchar
state	varchar

Table 8: Staff Table

Field	Data Type
staffId	int
firstname	varchar
lastname	varchar
jobtitle	varchar
email	varchar
phone	varchar
age	int
gender	Enum (‘Male’, ‘Female’, ’Other’)
address	varchar
city	varchar
zipcode	varchar
state	varchar
salary	Decimal (10, 2)

Table 9: Nurse Assignment Table

Field	Data Type
wardassignmentId	int
staffId	int
wardId	int
shift	Enum (‘Morning’, ‘Afternoon’, ‘Night’)
shiftTime	time

Define the schema along with the constraints indicating the relationships between the entities.

Table 1: Ward Table

Field	Data Type	Relationship
wardId	int	Primary Key
wardName	varchar	Not null
wardDesc	text	-

Table 2: Room Table

Field	Data Type	Relationship
roomId	int	Primary Key
wardId	int	Foreign Key
roomNo	int	Not null

Table 3: Patient Table

Field	Data Type	Relationship
patientId	int	Primary Key
roomId	int	Foreign Key
firstname	varchar	Not null
lastname	varchar	Not null
email	varchar	-
phone	varchar	Not null
age	int	-
gender	Enum (‘Male’, ‘Female’, ’Other’)	Not null
address	varchar	Not null
city	varchar	Not null
zipcode	varchar	Not null
state	varchar	Not null
allergies	Text	-0000000
D00isease	Text	-
otherhealthDisease	Text	-
healthInsurance	DATE	Not null
insuarnceStatus	varchar	Not null

Table 4: Treatment Details Table

Field	Data Type	Relationship

treatmentDetailsId	int	Primary Key
treatmentName	varchar	Not null
description	text	-
price	Decimal (10, 2)	Not null

Table 5: Treatment Table

Field	Data Type	Relationship
treatmentId	int	Primary Key
patientId	int	Foreign Key
treatmentDetailsId	varchar	Foreign Key
currentStatus	varchar	-
treatmentDate	Date	-

Table 6: Bill Table

Field	Data Type	Relationship
paymentId	int	Primary Key
patientId	int	Foreign Key
treatmentId	int	Foreign Key
totalAmount	Decimal (10, 2)	Not null
amountPaid	Decimal (10, 2)	Not null
WardDesc	Decimal (10, 2)	Not null

Table 7: Doctor Table

Field	Data Type	Relationship
doctorId	int	Primary Key
firstname	varchar	Not null
lastname	varchar	Not null
specialityName	varchar	Not null
email	varchar	-
phone	varchar	Not null
age	int	-
gender	Enum (‘Male’, ‘Female’, ’Other’)	Not null
address	varchar	-
city	varchar	-
zipcode	varchar	-
state	varchar	-

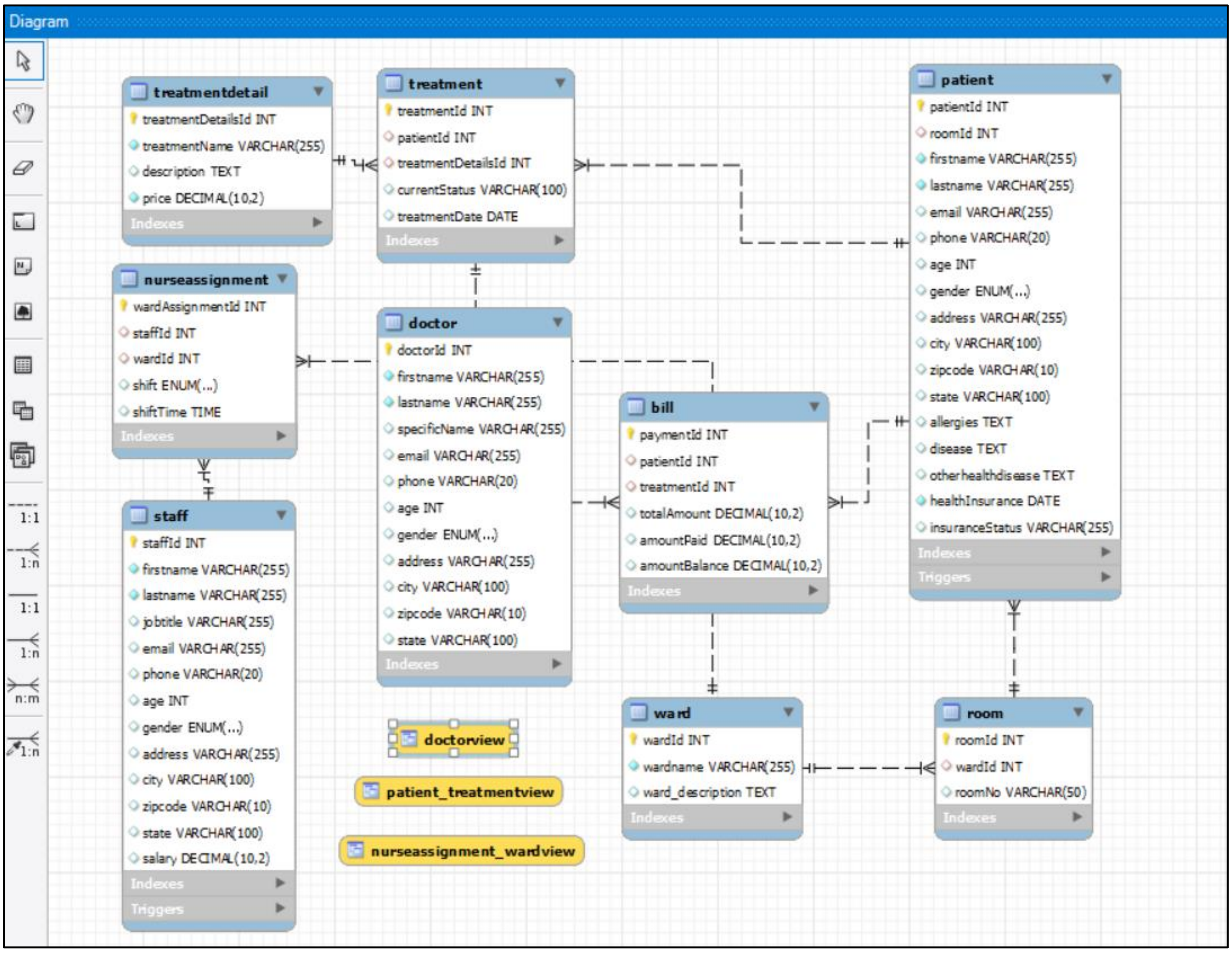
Table 8: Staff Table

Field	Data Type	Relationship
staffId	int	Primary Key
firstname	varchar	Not null
lastname	varchar	Not null
jontitile	varchar	Not null
email	varchar	-
phone	varchar	Not null
age	int	-
gender	Enum (‘Male’, ‘Female’, ’Other’)	Not null

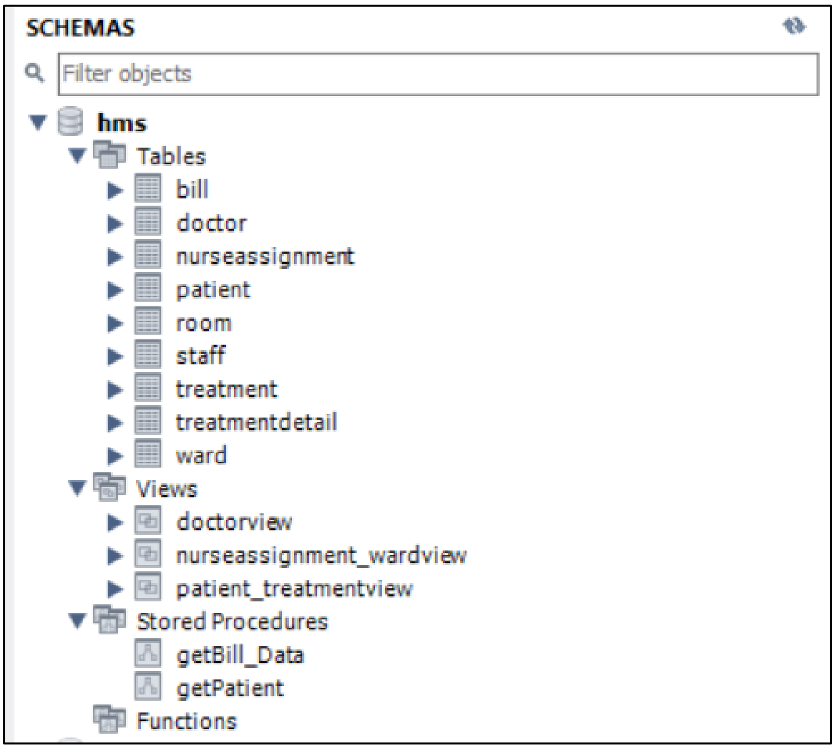
address	varchar	-
city	varchar	-
zipcode	varchar	-
state	varchar	-
salary	Decimal (10, 2)	-

Table 9: Nurse Assignment Table

Field	Data Type	Relationship
wardassignmentId	int	Primary Key
staffId	int	Foreign Key
wardId	int	Foreign Key
shift	Enum (‘Morning’, ‘Afternoon’, ‘Night’)	Not null
shiftTime	time	Not null



Be sure to make use of the database concepts like Views, Relationships, Indexing, Stored Procedure and triggers.



SCHEMAS

Filter objects

- hms
  - Tables
    - bill
    - doctor
    - nurseassignment
    - patient
    - room
    - staff
    - treatment
    - treatmentdetail
    - ward
  - Views
    - doctorview
    - nurseassignment_wardview
    - patient_treatmentview
  - Stored Procedures
    - getBill_Data
    - getPatient
  - Functions

Indicate the Normalization form being used in the schema defined and why you chose to keep it that particular normal form.

The purpose of normalization schemas is to prevent redundant data from being stored, preventing inconsistent data from being stored. because duplicate data is not present in tables that have a normalized schema.
To prevent row-level duplication, I avoided it in my schema definition. I inserted each row with unique data by using the main key of every table that could exist.
Created primary keys can be joined on tables to retrieve data from many sources without continually inputting the same data and can also be used as foreign keys for other tables.
To prevent duplication at the column level. On defined schema, the first normal form (1NF) is utilized.
1NF: If a relation has an atomic value, it is 1NF. It says that a table's attribute cannot have more than one value. It can only contain one-valued First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Applying 1NF prevents the insertion of numerous values into a table, and attributes with non-null constraints help to prevent data ambiguity. Additionally, each tiny piece of necessary data that is kept in a database is kept with a unique schema; the only way to locate data from many tables in a single table in a database is through joins.
Since 1NF is a basic level of normalization with little complexity and reduces data duplication at the row level due to its association with patient and physician data, where each individual requires a unique identification in order to retrieve data with accuracy. For normalization in the schema definition, INF was my choice.

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

Query:

INSERT INTO Patient

(roomid, firstname, lastname, email, phone, age, gender, address, city, zipcode, state, allergies, disease, otherhealthdisease, healthInsurance)

VALUES (2, 'Namrata', 'Patil', 'namu877patil@example.com', '8779101213', 22, 'Female', '203, Pitru Apartment', 'Navi Mumbai', '400708', 'Maharashtra', 'Peanuts', 'Diabetes', 'Hypertension', '2024-01-01');

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

Query: SELECT p.patientId, td.treatmentName, t.treatmentDate

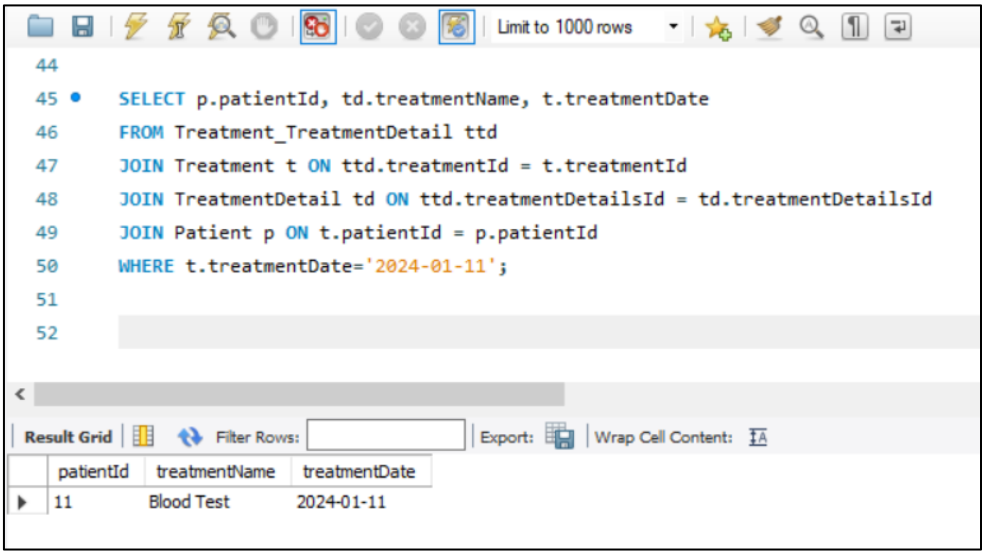
FROM Treatment_TreatmentDetail ttd

JOIN Treatment t ON ttd.treatmentId = t.treatmentId

JOIN TreatmentDetail td ON ttd.treatmentDetailsId = td.treatmentDetailsId

JOIN Patient p ON t.patientId = p.patientId

t.treatmentDate='2024-01-11';

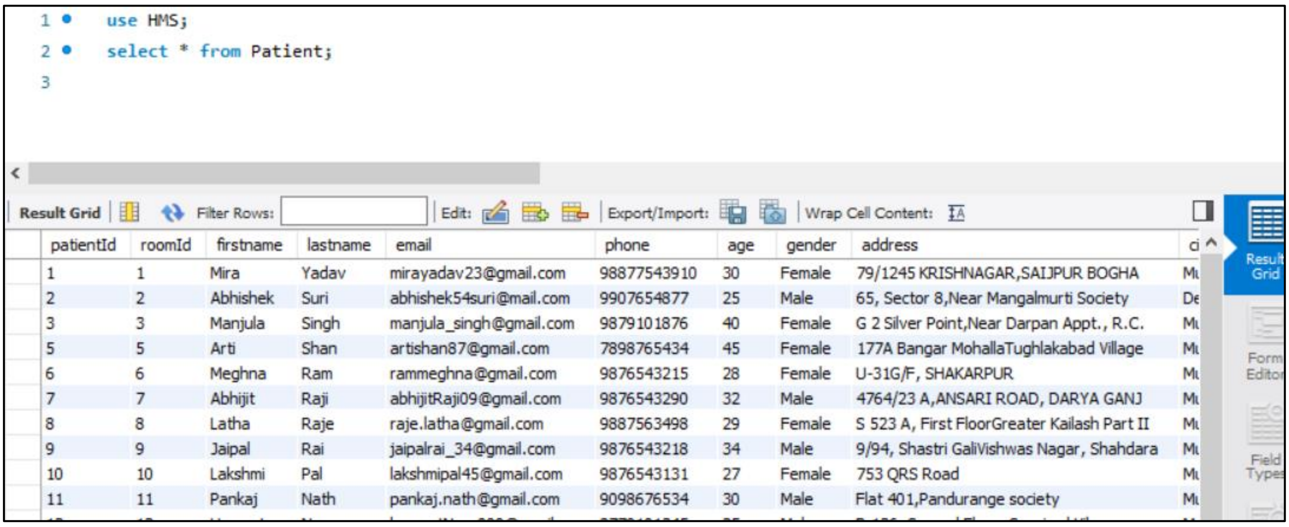


```
SELECT p.patientId, td.treatmentName, t.treatmentDate
FROM Treatment_TreatmentDetail ttd
JOIN Treatment t ON ttd.treatmentId = t.treatmentId
JOIN TreatmentDetail td ON ttd.treatmentDetailsId = td.treatmentDetailsId
JOIN Patient p ON t.patientId = p.patientId
WHERE t.treatmentDate='2024-01-11';
```

patientId	treatmentName	treatmentDate
11	Blood Test	2024-01-11

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

Query: select * from Patient;



```
use HMS;
select * from Patient;
```

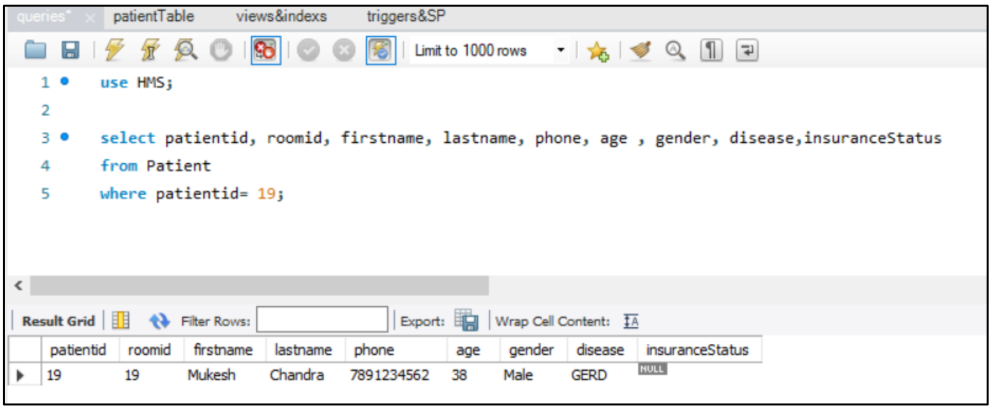
patientId	roomId	firstname	lastname	email	phone	age	gender	address	ci
1	1	Mira	Yadav	mirayadav23@gmail.com	98877543910	30	Female	79/1245 KRISHNAGAR,SAIJPUR BOGHA	Mu
2	2	Abhishek	Suri	abhishek54suri@mail.com	9907654877	25	Male	65, Sector 8,Near Mangalmurti Society	De
3	3	Manjula	Singh	manjula_singh@gmail.com	9879101876	40	Female	G 2 Silver Point,Near Darpan Appt., R.C.	Mu
5	5	Arti	Shan	artishan87@gmail.com	7898765434	45	Female	177A Bangar MohallaTughlakabad Village	Mu
6	6	Meghna	Ram	rammeghna@gmail.com	9876543215	28	Female	U-31G/F, SHAKARPUR	Mu
7	7	Abhijit	Raji	abhijitRaji09@gmail.com	9876543290	32	Male	4764/23 A,ANSARI ROAD, DARYA GANJ	Mu
8	8	Latha	Raje	raje.latha@gmail.com	9887563498	29	Female	S 523 A, First FloorGreater Kailash Part II	Mu
9	9	Jaipal	Rai	jaipalrai_34@gmail.com	9876543218	34	Male	9/94, Shastri GaliVishwas Nagar, Shahdara	Mu
10	10	Lakshmi	Pal	lakshmipal45@gmail.com	9876543131	27	Female	753 QRS Road	Mu
11	11	Pankaj	Nath	pankaj.nath@gmail.com	9098676534	30	Male	Flat 401,Pandurange society	Mu

Query:

select patientid, roomid, firstname, lastname, phone, age , gender, disease,insuranceStatus

from Patient

where patientid= 19;



```
use HMS;

select patientid, roomid, firstname, lastname, phone, age , gender, disease,insuranceStatus
from Patient
where patientid= 19;
```

patientid	roomid	firstname	lastname	phone	age	gender	disease	insuranceStatus
19	19	Mukesh	Chandra	7891234562	38	Male	GERD	NULL

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

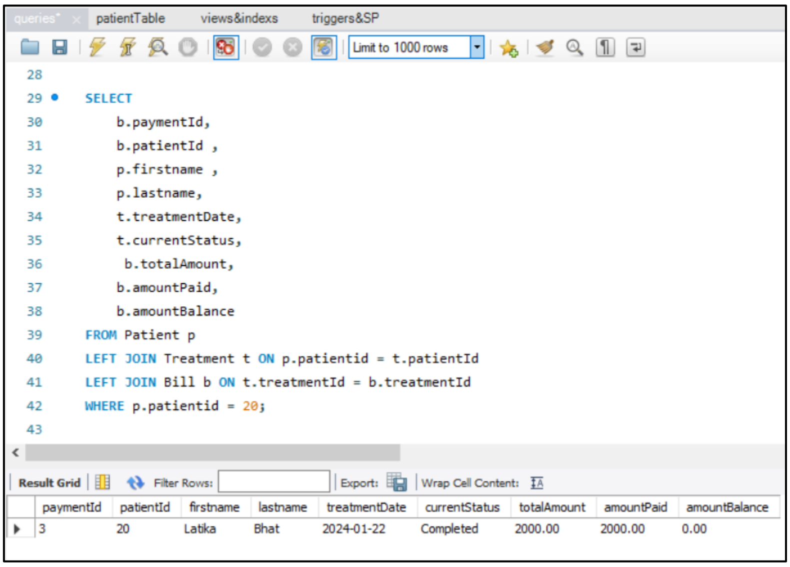
Query: SELECT b.paymentId, b.patientId , p.firstname , p.lastname, t.treatmentDate, t.currentStatus, b.totalAmount, b.amountPaid, b.amountBalance

FROM Patient p

LEFT JOIN Treatment t ON p.patientid = t.patientId

LEFT JOIN Bill b ON t.treatmentId = b.treatmentId

WHERE patientid = 20;



```sql

SELECT
    b.paymentId,
    b.patientId ,
    p.firstname ,
    p.lastname,
    t.treatmentDate,
    t.currentStatus,
     b.totalAmount,
    b.amountPaid,
    b.amountBalance
FROM Patient p
LEFT JOIN Treatment t ON p.patientid = t.patientId
LEFT JOIN Bill b ON t.treatmentId = b.treatmentId
WHERE p.patientid = 20;

```

paymentId	patientId	firstname	lastname	treatmentDate	currentStatus	totalAmount	amountPaid	amountBalance
3	20	Latika	Bhat	2024-01-22	Completed	2000.00	2000.00	0.00

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

Query 1:

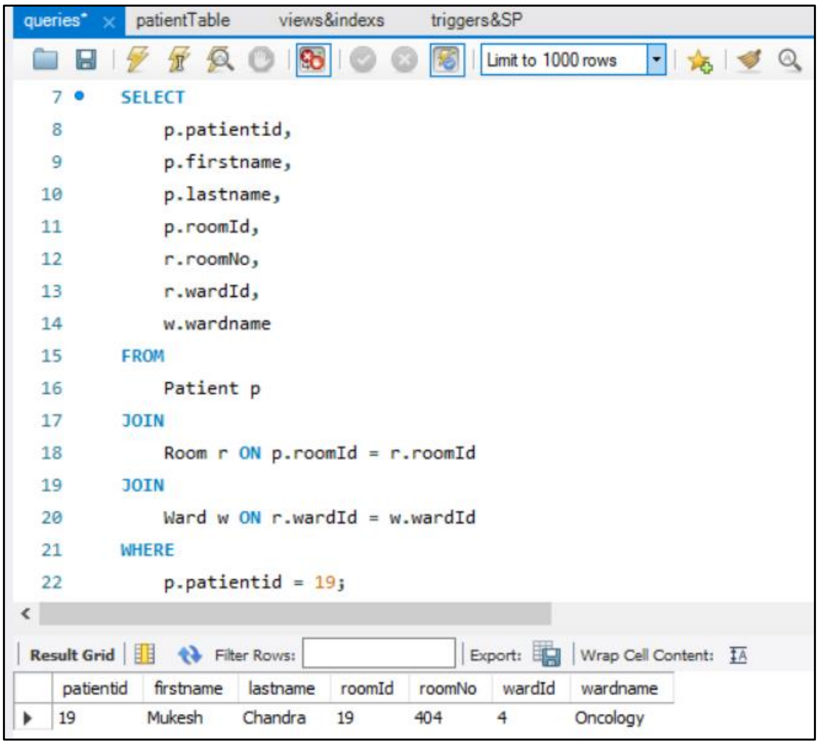
SELECT p.patientid, p.firstname, p.lastname, p.roomId, r.roomNo, r.wardId, w.wardname

FROM Patient p

JOIN Room r ON p.roomId = r.roomId

JOIN Ward w ON r.wardId = w.wardId

WHERE p.patientid = 19;



```sql
SELECT
    p.patientid,
    p.firstname,
    p.lastname,
    p.roomId,
    r.roomNo,
    r.wardId,
    w.wardname
FROM
    Patient p
JOIN
    Room r ON p.roomId = r.roomId
JOIN
    Ward w ON r.wardId = w.wardId
WHERE
    p.patientid = 19;
```

patientid	firstname	lastname	roomId	roomNo	wardId	wardname
19	Mukesh	Chandra	19	404	4	Oncology

QUERY 2:

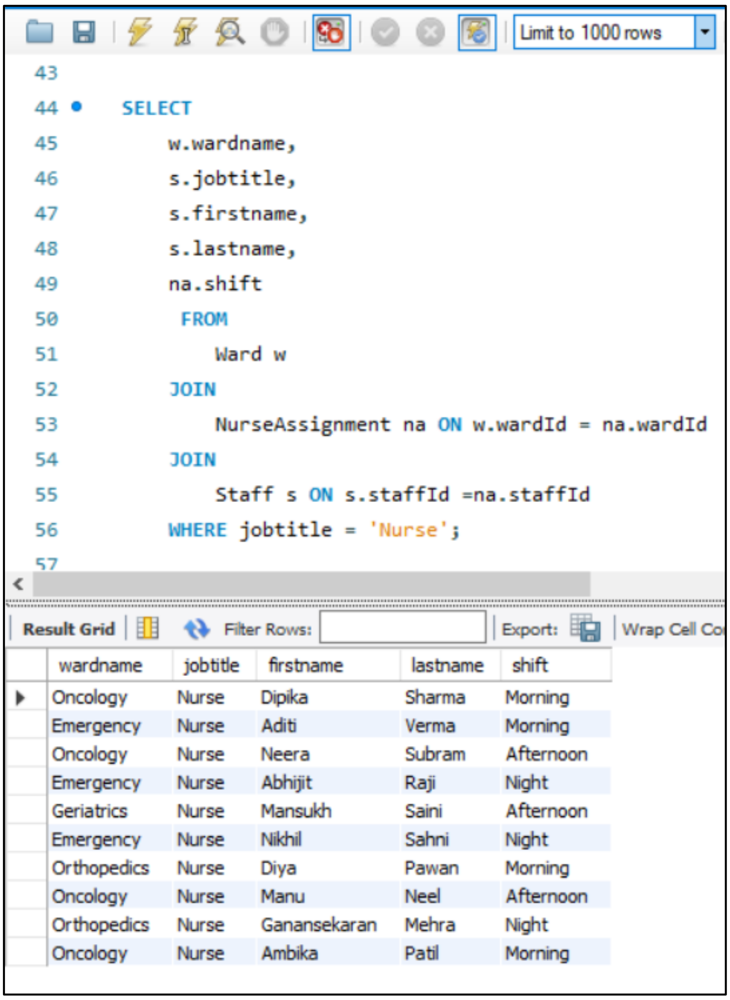
SELECT w.wardname, s.jobtitle, s.firstname,   s.lastname,   na.shift

FROM Ward w

JOIN NurseAssignment na ON w.wardId = na.wardId

JOIN Staff s ON s.staffId =na.staffId

WHERE jobtitle = 'Nurse';



```

SELECT
    w.wardname,
    s.jobtitle,
    s.firstname,
    s.lastname,
    na.shift
     FROM
        Ward w
    JOIN
        NurseAssignment na ON w.wardId = na.wardId
    JOIN
        Staff s ON s.staffId =na.staffId
    WHERE jobtitle = 'Nurse';
```

wardname	jobtitle	firstname	lastname	shift
Oncology	Nurse	Dipika	Sharma	Morning
Emergency	Nurse	Aditi	Verma	Morning
Oncology	Nurse	Neera	Subram	Afternoon
Emergency	Nurse	Abhijit	Raji	Night
Geriatrics	Nurse	Mansukh	Saini	Afternoon
Emergency	Nurse	Nikhil	Sahni	Night
Orthopedics	Nurse	Diya	Pawan	Morning
Oncology	Nurse	Manu	Neel	Afternoon
Orthopedics	Nurse	Ganansekaran	Mehra	Night
Oncology	Nurse	Ambika	Patil	Morning

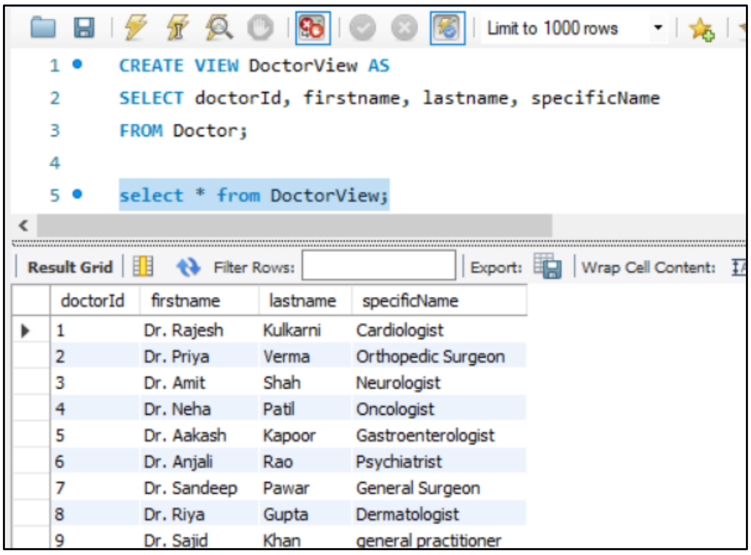
- Optimize repeated read operations using views/materialized views.

1) Doctor view

Query: CREATE VIEW DoctorView AS

SELECT doctorId, firstname, lastname, specificName

FROM Doctor;



```sql
CREATE VIEW DoctorView AS
SELECT doctorId, firstname, lastname, specificName
FROM Doctor;

select * from DoctorView;
```

doctorId	firstname	lastname	specificName
1	Dr. Rajesh	Kulkarni	Cardiologist
2	Dr. Priya	Verma	Orthopedic Surgeon
3	Dr. Amit	Shah	Neurologist
4	Dr. Neha	Patil	Oncologist
5	Dr. Aakash	Kapoor	Gastroenterologist
6	Dr. Anjali	Rao	Psychiatrist
7	Dr. Sandeep	Pawar	General Surgeon
8	Dr. Riya	Gupta	Dermatologist
9	Dr. Sajid	Khan	general practitioner

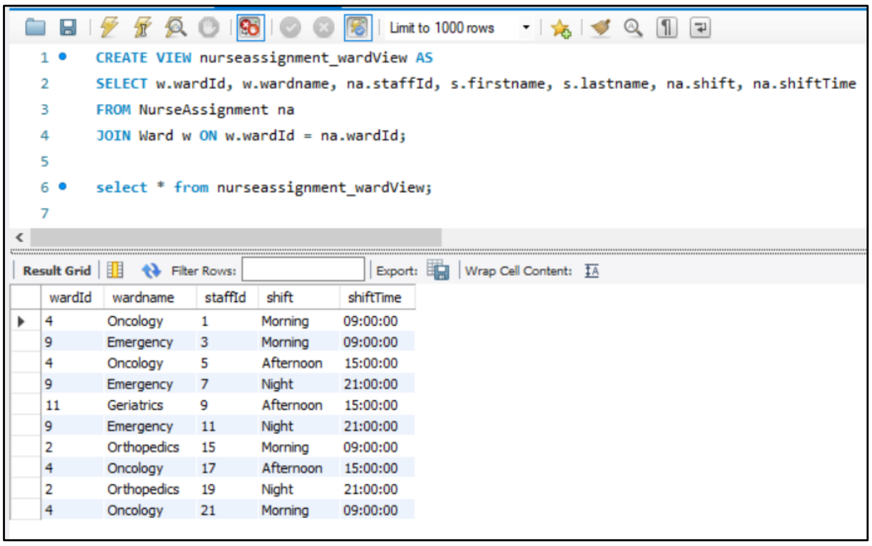
2) Nurse and Ward assignment view

Query: CREATE VIEW nurseassignment_wardView AS

SELECT w.wardId, w.wardname, na.staffId, s.firstname, s.lastname, na.shift, na.shiftTime

FROM NurseAssignment na

JOIN Ward w ON w.wardId = na.wardId;



```sql
CREATE VIEW nurseassignment_wardView AS
SELECT w.wardId, w.wardname, na.staffId, s.firstname, s.lastname, na.shift, na.shiftTime
FROM NurseAssignment na
JOIN Ward w ON w.wardId = na.wardId;

select * from nurseassignment_wardView;

```

wardId	wardname	staffId	shift	shiftTime
4	Oncology	1	Morning	09:00:00
9	Emergency	3	Morning	09:00:00
4	Oncology	5	Afternoon	15:00:00
9	Emergency	7	Night	21:00:00
11	Geriatrics	9	Afternoon	15:00:00
9	Emergency	11	Night	21:00:00
2	Orthopedics	15	Morning	09:00:00
4	Oncology	17	Afternoon	15:00:00
2	Orthopedics	19	Night	21:00:00
4	Oncology	21	Morning	09:00:00

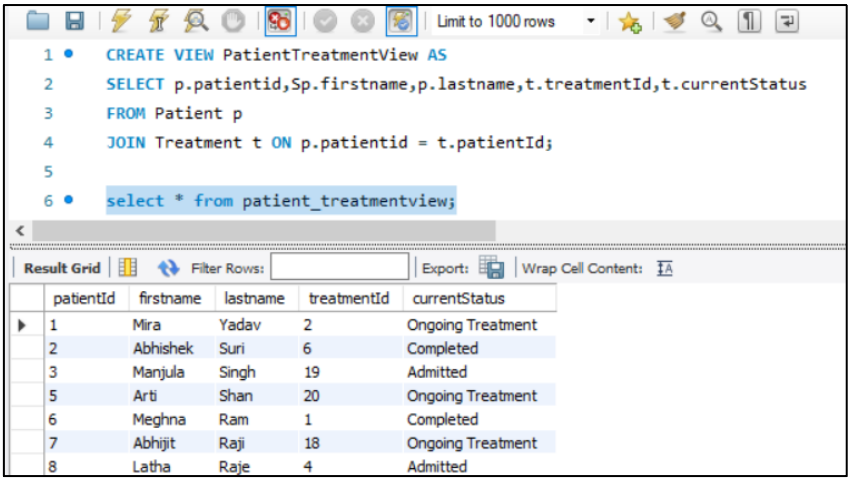
3) Patient_treatment view

Query: CREATE VIEW PatientTreatmentView AS

SELECT p.patientid,Sp.firstname,p.lastname,t.treatmentId,t.currentStatus

FROM Patient p

JOIN Treatment t ON p.patientid = t.patientId;



```
CREATE VIEW PatientTreatmentView AS
SELECT p.patientid,Sp.firstname,p.lastname,t.treatmentId,t.currentStatus
FROM Patient p
JOIN Treatment t ON p.patientid = t.patientId;

select * from patient_treatmentview;
```

patientId	firstname	lastname	treatmentId	currentStatus
1	Mira	Yadav	2	Ongoing Treatment
2	Abhishek	Suri	6	Completed
3	Manjula	Singh	19	Admitted
5	Arti	Shan	20	Ongoing Treatment
6	Meghna	Ram	1	Completed
7	Abhijit	Raji	18	Ongoing Treatment
8	Latha	Raje	4	Admitted

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

Query:

CREATE INDEX idx_specificname ON doctor(specificname);

CREATE INDEX idx_firstname ON Patient(firstname);

- Try optimizing bill generation using stored procedures.

Query to create stored procedure:

DELIMITER //

CREATE PROCEDURE getBill_Data(IN id INT)

BEGIN

SELECT b.paymentId, b.patientId , p.firstname , p.lastname, t.treatmentDate, t.currentStatus, b.totalAmount, b.amountPaid, b.amountBalance

FROM Patient p

LEFT JOIN Treatment t ON p.patientid = t.patientId

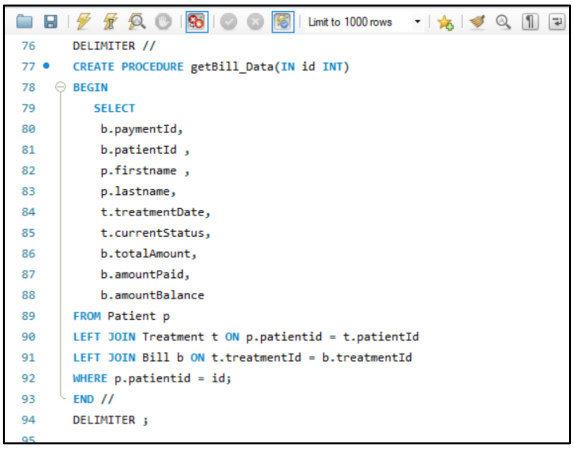
LEFT JOIN Bill b ON t.treatmentId = b.treatmentId

WHERE p.patientid = id;

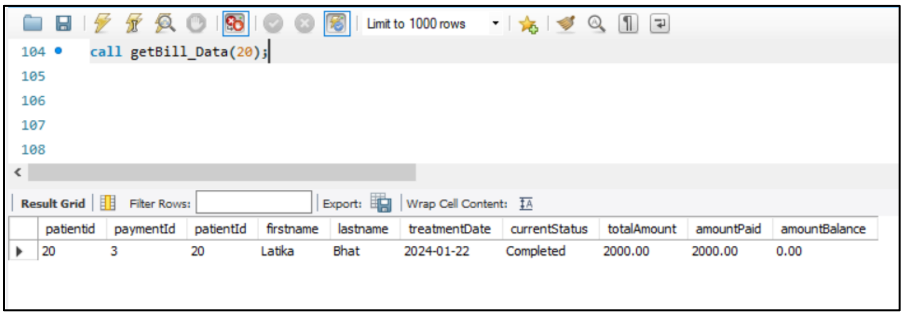
END //

DELIMITER;

call getBill_Data(20);



```
DELIMITER //
CREATE PROCEDURE getBill_Data(IN id INT)
BEGIN
   SELECT
    b.paymentId,
    b.patientId ,
    p.firstname ,
    p.lastname,
    t.treatmentDate,
    t.currentStatus,
    b.totalAmount,
    b.amountPaid,
    b.amountBalance
FROM Patient p
LEFT JOIN Treatment t ON p.patientid = t.patientId
LEFT JOIN Bill b ON t.treatmentId = b.treatmentId
WHERE p.patientid = id;
END //
DELIMITER ;
```



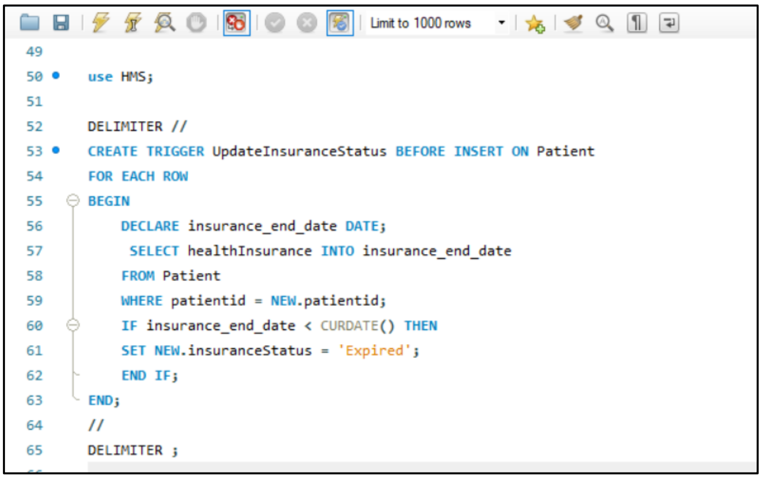
```
call getBill_Data(20);
```

patientid	paymentId	patientId	firstname	lastname	treatmentDate	currentStatus	totalAmount	amountPaid	amountBalance
20	3	20	Latika	Bhat	2024-01-22	Completed	2000.00	2000.00	0.00

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

Query:

```
DELIMITER //
CREATE TRIGGER UpdateInsuranceStatus BEFORE INSERT ON Patient
FOR EACH ROW
BEGIN
   DECLARE insurance_end_date DATE;
   SELECT healthInsurance INTO insurance_end_date
   FROM Patient
   WHERE patientid = NEW.patientid;
   IF insurance_end_date < CURDATE() THEN
      SET NEW.insuranceStatus = 'Expired';
   END IF;
END;
//
DELIMITER ;
```



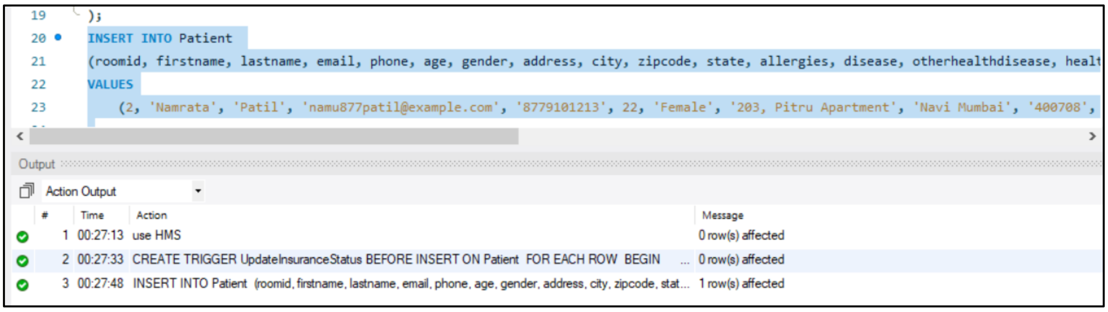
```
use HMS;

DELIMITER //
CREATE TRIGGER UpdateInsuranceStatus BEFORE INSERT ON Patient
FOR EACH ROW
BEGIN
    DECLARE insurance_end_date DATE;
     SELECT healthInsurance INTO insurance_end_date
    FROM Patient
    WHERE patientid = NEW.patientid;
    IF insurance_end_date < CURDATE() THEN
    SET NEW.insuranceStatus = 'Expired';
    END IF;
END;
//
DELIMITER ;
```

Query:

INSERT INTO Patient (roomid, firstname, lastname, email, phone, age, gender, address, city, zipcode, state, allergies, disease, otherhealthdisease, healthInsurance)

VALUES (2, 'Namrata', 'Patil', 'namu877patil@example.com', '8779101213', 22, 'Female', '203, Pitru Apartment', 'Navi Mumbai', '400708', 'Maharashtra', 'Peanuts', 'Diabetes', 'Hypertension', '2024-01-01');



```
);
INSERT INTO Patient
(roomid, firstname, lastname, email, phone, age, gender, address, city, zipcode, state, allergies, disease, otherhealthdisease, healt
VALUES
    (2, 'Namrata', 'Patil', 'namu877patil@example.com', '8779101213', 22, 'Female', '203, Pitru Apartment', 'Navi Mumbai', '400708',
```

```
   #  Time      Action                                                                        Message
   1  00:27:13  use HMS                                                                       0 row(s) affected
   2  00:27:33  CREATE TRIGGER UpdateInsuranceStatus BEFORE INSERT ON Patient  FOR EACH ROW  BEGIN   ...  0 row(s) affected
   3  00:27:48  INSERT INTO Patient  (roomid, firstname, lastname, email, phone, age, gender, address, city, zipcode, stat...  1 row(s) affected
```

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: The process of converting code into readable, interactive web content is known as rendering in web development. In order to do this, programming language code must be interpreted and displayed as an interactive webpage for consumers to engage with. Either the user or the server can do this.

Patterns of Rendering

One kind of design pattern that is particularly focused on the rendering of web pages is the rendering pattern. They offer a means of structuring and organizing the code that creates the HTML, CSS, and JavaScript that are given to the browser.

The process of rendering HTML, CSS, and JavaScript on the server before delivering it to the browser is known as server-side rendering, or SSR. As a result, the web page may load faster because the browser does not need to perform any rendering work. On the other hand, SSR may also be more challenging to integrate and update the website.

Client-side rendering (CSR): This refers to how the browser renders HTML, CSS, and JavaScript. Because the modifications may be done immediately in the browser, this may facilitate updating the website. But because the browser needs to work harder, CSR might sometimes be slower than SSR.

The method of creating the HTML, CSS, and JavaScript for a web page at build time is known as static site generation, or SSG. Since the browser doesn't need to perform any rendering, this can happen very quickly. However, because the build process must be modified, SSG may make it more challenging to update the website.

Design patterns:

Design patterns are reusable fixes for typical issues that arise during the software design process. They act as models or guides for resolving particular design problems. Developers can decrease errors, produce scalable and maintainable code, and enhance code organization by utilizing design patterns.

Types of Rendering Patterns

1) Observer Pattern:

The observer pattern is applicable in rendering scenarios where multiple objects need to be notified about changes in the rendering state. Apply the Observer Pattern when you need to implement event handling or keep multiple components synchronized with a common data source.

Use Case: In a game engine, various entities might need to be updated and re-rendered based on changes in the game world. The observer pattern facilitates efficient communication and synchronization between these entities.

2) Factory Pattern:

   The factory pattern is commonly used in rendering systems for creating and managing different types of rendering objects and resources. It provides a centralized mechanism for creating instances of renderable objects, such as meshes, textures, shaders, or materials. The factory pattern allows for easy extensibility, as new types of renderable objects can be added without modifying the existing codebase.

   Use case: In a game engine, we need to support various types of textures such as diffuse maps, normal maps, specular maps, etc.

3) Command Pattern:

   The command pattern is well-suited for rendering tasks that require user interaction and dynamic behaviour. It decouples the invoker of the rendering command from the object implementing the rendering logic. This pattern allows for easily adding, modifying, or removing rendering commands without impacting other parts of the system.

   Use case: In a 2D game, we need to handle various user input commands such as moving the player character, shooting projectiles, or activating power-ups. We can create different command to perform particular action.

4) Composite Pattern:

   The composite pattern is suitable for rendering tasks that involve hierarchical or nested structures. For instance, in scene graphs, where objects are organized based on their relationships, the composite pattern enables efficient rendering of the entire scene and handling changes at different levels of the hierarchy.

   Use Case: In a 3D game engine, we need to render a scene consisting of various objects, such as models, lights, cameras, etc., each with its transformation and properties. These objects are related to each other in a hierarchical structure, and the scene's organization plays a significant role in rendering performance.

5) State Pattern:

   The state pattern is particularly useful in rendering scenarios where the behaviour of an object or scene depends on its current state. For example, in a video player application, the rendering process might differ based on the playback state (play, pause, stop). The state pattern enables easy management and transitions between these rendering states.

   Use Case: Developing a graphic design software where users can create and edit various shapes. The rendering of each shape may vary based on its current state, such as selected, deselected, or hovered over.

Conclusion:

Rendering is the process of generating a visual output from data, and design patterns provide structured solutions for rendering challenges. The command, observer, factory, composite, and state patterns are well-suited for various rendering tasks, such as user interaction, dynamic updates, resource creation, hierarchical structures, and managing rendering states. Applying the appropriate rendering pattern ensures efficient and flexible rendering systems tailored to specific use cases.