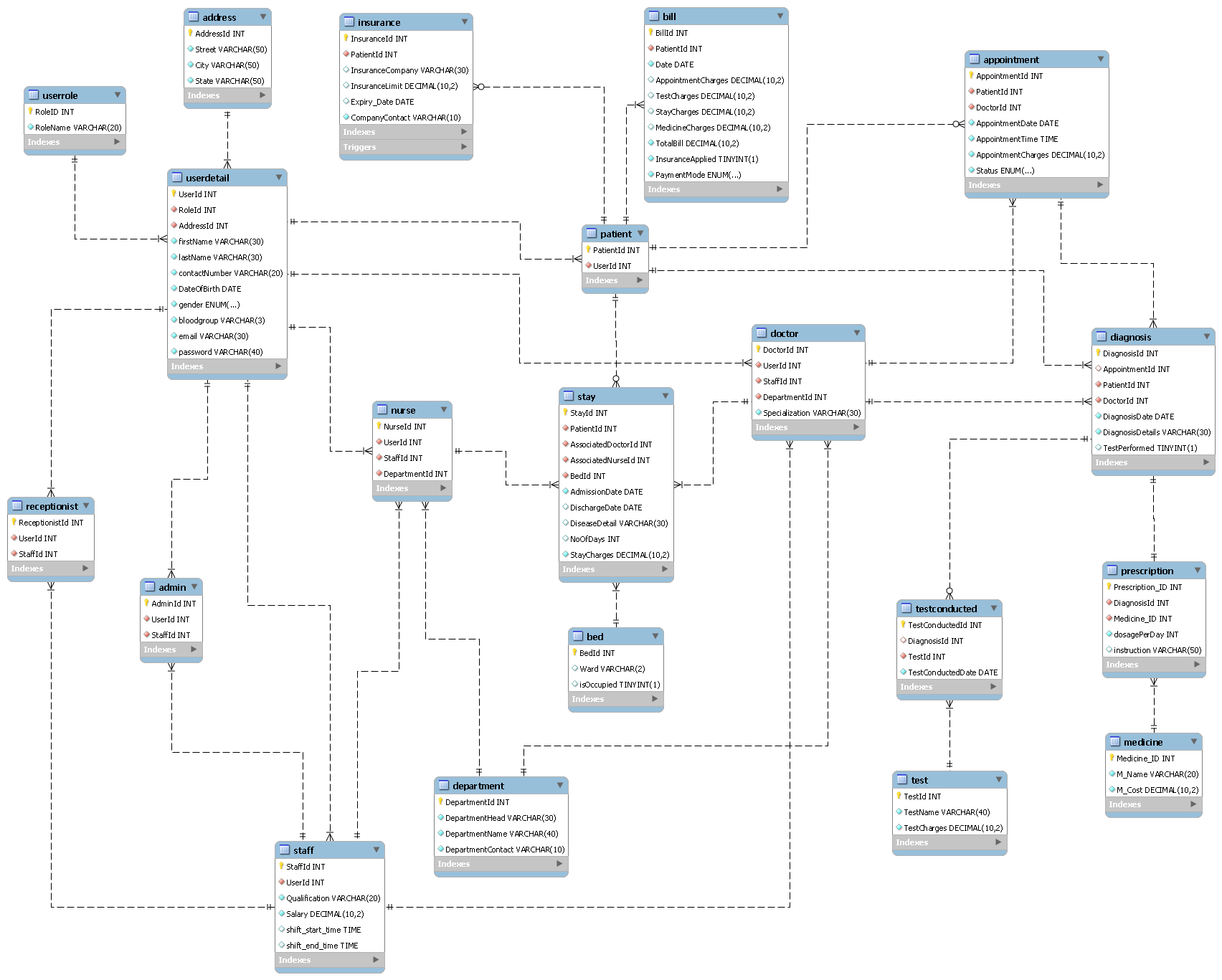
**Name : Akshay Patil**

**Milestone Project 1**

**Q- Create a DB Schema for Hospital Management Systems.**



**Relationships between tables :**

| Expression | Description |
| --- | --- |
|  | A patient may have zero to many insurance policies.  An insurance policy must be associated with one and only one patient |
|  | A User must be associated with one and only one user role.  The same UserRole may be assigned to many Users. |
|  | A patient may have zero to many appointments.  An appointment must be associated with one and only one patient |
|  | A Bill must be associated with one and only one Patient.  A Single Patient may have many Bills. |
|  | A patient may undergo many diagnoses.  A Diagnosis must be associated with one and only one Patient |
|  | Prescription must be associated with one and only one Diagnosis.  Diagnosis must have only one prescription. |
|  | In Diagnosis there may be a requirement of zero or many Tests.  A Test must be associated with one and only one Diagnosis |
|  | A doctor must have one and only one record under staff.  Staff may contain many doctors' information. |
|  | A Nurse must have one and only one record under staff.  Staff may contain many Nurse’s information. |
|  | A Receptionist must have one and only one record under staff.  Staff may contain many Receptionists information. |

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

**UserDetail Table** :

Initially I have created 5 Separate tables for Admin, Receptionist, Patient , Doctor and Nurse. To eliminate redundancy and improve the database's organization, I consolidated common attributes such as FirstName, LastName, BloodGroup, Gender, ContactNo, email, and Password into a single table named UserDetail. This table serves as a centralized repository for shared user information. To establish associations with specific roles (Admin, Receptionist, Patient, Doctor, and Nurse), I implemented foreign key relationships between the UserDetail table and the corresponding role-specific tables. This approach enhances data consistency, reduces duplication, and facilitates easier maintenance and updates in the database structure.

Before Normalization UserDetail table looks like below :



1NF : A table is in the First Normal Form (1NF) since it contains only atomic values in each column, and there are no repeating groups.

2NF : Primary Goal of 2NF is that all non-prime attributes must be fully dependent on Primary Key(UserId) .The table Satisfies this condition.

But there is data redundancy present in the table.While inserting a new record into table everytime you need to specify RoleName which is repetitive task.To reduce the complexity, a Separate UserRole Table is created which contains RoleId along with RoleName. This separation helps us while updating data and it also helps in future to add permission related detail to each user.

It may possible that a single patient may contain multiple address in that case to store 2 address details in same UserDetail table we have to repeat entire detail of User including firstName,lastName,contactNumber,blood group,gender,email and password this will create data redundancy to avoid this a separate Address table is created.

3NF : The table is in the Third Normal Form (3NF) because it has no transitive dependencies. All non-key attributes are directly dependent on the primary key “UserId.”

**After Applying Normalization :**

A single User Detail Table is normalized into 3 separate tables to follow normalization rules and to deal with insertion,updation and deletion anomaly.

**UserRole table :**

| Primary Key | RoleId int |
| --- | --- |
|  | RoleName varchar |

**Address Table :**

| Primary Key | AddressId int |
| --- | --- |
|  | StreetName varchar  City varchar  State varchar |

**UserDetail Table :**

| Primary Key | UserId int |
| --- | --- |
| Foreign key references UserRole Table | RoleId int |
| Foreign Key references Address Table | AddressId int |
|  | FirstName varchar  LastName varchar  contactNumber varchar  DateOfBirth date  Gender ENUM(‘Male’,’Female’)  Bloodgroup varchar  Email varchar  Password varchar |

Also Admin,Receptionist ,Doctor and Nurse are part of Hospital Staff . Staff of a hospital must contain Salary,qualification and DepartmentName attributes.So differentiation is made by creating a Separate Staff table and Department table. Admin and Receptionist are not part of any Department they belong to complete Hospital.

**Staff Table :**

| Primary Key | StaffId int |
| --- | --- |
| Foreign Key references UserDetail Table | UserId int |
|  | Salary decimal  Qualification varchar  Shift\_start\_time time  Shift\_end\_time time |

**Admin Table :**

| Primary Key | AdminId int |
| --- | --- |
| Foreign Key references Staff Table | StaffId int |
| Foreign Key references UserDetail Table | UserId int |

**Receptionist Table :**

| Primary Key | ReceptionistId int |
| --- | --- |
| Foreign Key references Staff Table | StaffId int |
| Foreign Key references UserDetail Table | UserId int |

**Department Table :**

| Primary Key | DepartmentId int |
| --- | --- |
|  | DepartmentHead varchar  DepartmentName varchar  DepartmentContact varchar |

**Doctor Table :**

| Primary Key | DoctorId int |
| --- | --- |
| Foreign Key references UserDetail Table | UserId int |
| Foreign Key references Staff Table | StaffId int |
| Foreign Key references Department Table | DepartmentId int |
|  | Specialization varchar |

**Nurse Table :**

| Primary Key | NurseId int |
| --- | --- |
| Foreign Key references UserDetail Table | UserId int |
| Foreign Key references Staff Table | StaffId int |
| Foreign Key references Department Table | DepartmentId int |

**Patient Table :**

| Primary Key | PatientId int |
| --- | --- |
| Foreign Key references UserDetail Table | UserId int |

It may Possible for a single Patient to have Multiple insurance policy.So to create such one to many relationship , a Separate Insurance Table is created and they are linked using foreign key.

**Insurance Table :**

| Primary Key | InsuranceId int |
| --- | --- |
| Foreign Key references Patient Table | PatientId int |
|  | InsuranceCompany varchar  InsuranceLimit decimal  Expiry\_Date date  CompanyContact varchar |

All the tables used above are first normalized and then used.They follow 1NF,2NF and 3NF properties.

**The Hospital Management System follows a sequential process:**

* Patients initiate the process by scheduling appointments with doctors for consultations.
* After confirming the appointment, patients proceed to undergo a diagnosis where healthcare professionals provide detailed examination results. In emergency cases, patients may directly undergo diagnosis without booking an appointment.
* Based on the diagnosis details, doctors prescribe medicines to patients. Additionally, patients may be required to undergo specific tests as part of their treatment plan.

Based on the above scenarios below tables are created and normalized.

**Appointment Table :**

| Primary Key | AppointmentId int |
| --- | --- |
| Foreign key References Patient Table | PatientId int |
| Foreign key References Doctor Table | DoctordId int |
|  | AppointmentDate date  AppointmentTime time  AppointmentCharges decimal  Status ENUM(‘Confirmed’,’Cancel’,’Completed’) |

**Diagnosis Table :**

| Primary Key | DiagnosisId int |
| --- | --- |
| Foreign key References Appointment Table | AppointmentId int |
| Foreign key References Patient Table | PatientId int |
| Foreign key References Doctor Table | DoctordId int |
|  | DiagnosisDate date  DiagnsisDetails varchar  TestPerformed boolean |

In Some situations performing a Test is not really important like for Disease Fever,Cold and Cough so in such cases TestPerformed Attribute will be false.

**Test Table :**

| Primary Key | TestId |
| --- | --- |
|  | TestName varchar  TestCharges decimal |

**TestConducted Table :**

| Primary key | TestConductedId int |
| --- | --- |
| Foreign key references Diagnosis Table | DiagnosisId int |
| Foreign key references Test Table | TestId int |
|  | TestConductedDate date |

**Medicine Table :**

| Primary Key | Medicine\_ID int |
| --- | --- |
|  | M\_Name varchar  M\_Cost decimal |

**Prescription Table :**

| Primary Key | PrescriptionId int |
| --- | --- |
| Foreign key references Diagnosis table | DiagnosisId int |
| Foreign key references Medicine table | Medicine\_ID int |
|  | dosagePerDay int  Instruction varchar |

Following a diagnosis, a doctor can prescribe multiple medicines in a single prescription, indicating a one-to-many relationship between diagnosis and medicines.

Example :

DiagnosisDetail : Diabetes Mellitus

TestName : Glycated Hemoglobin Test

Medicines :

Medicine 1 - Metformin (500mg)

Dosage : 2

Instruction : Take with meals in morning and evening

Medicine 2 - Sitagliptin (100mg)

Dosage : 1

Instruction : Can be taken without food in morning

**Bed Table :**

| Primary Key | BedId int |
| --- | --- |
|  | Ward varchar  isOccupied boolean |

**Stay Table :**

| Primary Key | StayId int |
| --- | --- |
| Foreign Key references Patient Table | PatientId int |
| Foreign Key references Doctor Table | AssociatedDoctorId int |
| Foreign Key references Nurse Table | AssociatedNurseId int |
| Foreign Key references Bed Table | BedId int |
|  | AdmissionDate date  DischargeDate date  DiseaseDetail date  NoOfDays int  StayCharges decimal |

**Bill Table :**

| Primary Key | BillId int |
| --- | --- |
| Foreign Key references Patient Table | PatientId int |
|  | Date date  AppointmentCharges decimal  TestCharges decimal  StayCharges decimal  MedicineCharges decimal  TotalBill decimal  InsuranceApplied boolean  PaymentMode ENUM('Cash','Card','UPI','NET Banking') |

While creating every table I have assured that schema adheres to the atleast Third Normal Form (3NF). I normalize the data in such a way that every table satisfies below rules:

* Created a table in such a way that every column must contains atomic values
* Ensuring that all non-prime attributes in every table are fully dependent on the primary key, eliminating any instances of partial dependency.
* Prohibiting transitive dependencies between tables.

All the tables created above are designed to prevent insertion, updating, and deletion anomalies.

**Practical Implementation of Schema using MYSQL.**

SQL commands for Database Creation :

Table Creation :

| CREATE TABLE UserRole(  RoleID int primary key,  RoleName varchar(20) NOT NULL  );  CREATE TABLE Address (  AddressId INT PRIMARY KEY AUTO\_INCREMENT,  Street VARCHAR(50) not null,  City VARCHAR(50) not null,  State VARCHAR(50) not null  );  CREATE TABLE UserDetail(  UserId int primary key auto\_increment,  RoleId int not null,  AddressId int not null,  firstName varchar(30) NOT NULL,  lastName varchar(30) NOT NULL,  contactNumber varchar(20) NOT NULL unique,  DateOfBirth date not null,  gender enum('Male','Female','Other')not null,  bloodgroup varchar(3) not null,  email varchar(30) not null unique,  password varchar(40) not null,  foreign key(RoleId) references UserRole(RoleID),  foreign key(AddressId) references Address(AddressId)  );  CREATE TABLE Staff (  StaffId INT PRIMARY KEY AUTO\_INCREMENT,  UserId INT NOT NULL,  Qualification VARCHAR(20) NOT NULL,  Salary DECIMAL(10, 2) NOT NULL,  shift\_start\_time time,  shift\_end\_time time,  FOREIGN KEY (UserId) REFERENCES UserDetail(UserId)  );  CREATE TABLE Admin(  AdminId int primary key ,  UserId int not null,  foreign key(UserId) references UserDetail(UserId)  );  CREATE TABLE Department(  DepartmentId int primary key,  DepartmentHead varchar(30) not null,  DepartmentName varchar(40) not null unique,  DepartmentContact varchar(10) not null unique  );  CREATE TABLE Patient(  PatientId int primary key auto\_increment,  UserId int not null,  foreign key(UserId) references UserDetail(UserId)  );  CREATE TABLE Insurance  (  InsuranceId int primary key auto\_increment,  PatientId int not null,  InsuranceCompany varchar(30),  InsuranceLimit Decimal(10,2),  Expiry\_Date date,  CompanyContact varchar(10) not null unique,  foreign key(PatientId) references Patient(PatientId)  );  CREATE TABLE Doctor(  DoctorId int primary key auto\_increment,  UserId int not null,  StaffId int not null,  DepartmentId int not null,  Specialization varchar(30) not null,  foreign key(UserId) references UserDetail(UserId),  foreign key(StaffId) references Staff(StaffId),  foreign key(DepartmentId) references Department(DepartmentId)  );  CREATE TABLE Nurse(  NurseId int primary key auto\_increment,  UserId int not null,  StaffId int not null,  DepartmentId int not null,  foreign key(UserId) references UserDetail(UserId),  foreign key(StaffId) references Staff(StaffId),  foreign key(DepartmentId) references Department(DepartmentId)  );  CREATE TABLE Receptionist(  ReceptionistId int primary key auto\_increment,  UserId int not null,  StaffId int not null,  foreign key(UserId) references UserDetail(UserId),  foreign key(StaffId) references Staff(StaffId)  );  CREATE TABLE Appointment  (  AppointmentId int primary key auto\_increment,  PatientId int not null,  DoctorId int not null,  AppointmentDate date not null,  AppointmentTime time not null,  AppointmentCharges decimal(10,2) not null,  Status Enum('Confirmed','Cancel','Completed') not null,  foreign key(PatientId) references Patient(PatientId),  foreign key(DoctorId) references Doctor(DoctorId),  CONSTRAINT UC\_AppointmentDateTime UNIQUE (DoctorId,AppointmentDate, AppointmentTime)  );  CREATE TABLE Diagnosis  (  DiagnosisId int primary key auto\_increment,  AppointmentId int,  PatientId int not null,  DoctorId int not null,  DiagnosisDate date not null,  DiagnosisDetails varchar(30) not null,  TestPerformed boolean default false,  foreign key(AppointmentId) references Appointment(AppointmentId),  foreign key(PatientId) references Patient(PatientId),  foreign key(DoctorId) references Doctor(DoctorId)  );  CREATE TABLE Test  (  TestId int primary key auto\_increment,  TestName varchar(40) not null unique,  TestCharges decimal(10,2) not null  );  CREATE TABLE TestConducted  (  TestConductedId int primary key auto\_increment,  DiagnosisId int,  TestId int not null,  TestConductedDate date not null,  foreign key(DiagnosisId) references Diagnosis(DiagnosisId),  foreign key(TestId) references Test(TestId)  );  CREATE TABLE Medicine (  Medicine\_ID INT primary key NOT NULL,  M\_Name VARCHAR(20) NOT NULL unique,  M\_Cost Decimal(10,2) not null  );  CREATE TABLE Prescription (  Prescription\_ID INT primary key NOT NULL,  DiagnosisId int not null,  Medicine\_ID INT NOT NULL,  dosagePerDay int not null,  instruction varchar(30) not null,  FOREIGN KEY (DiagnosisId) REFERENCES Diagnosis(DiagnosisId),  FOREIGN KEY (Medicine\_ID) REFERENCES Medicine (Medicine\_ID)  );  CREATE TABLE Bed  (  BedId int primary key,  Ward varchar(2),  isOccupied boolean default false  );  CREATE TABLE Stay  (  StayId int primary key auto\_increment,  PatientId int not null,  AssociatedDoctorId int not null,  AssociatedNurseId int not null,  BedId int not null,  AdmissionDate date not null,  DischargeDate date,  DiseaseDetail varchar(30),  NoOfDays int ,  StayCharges decimal(10,2) not null,  FOREIGN KEY (PatientId) REFERENCES Patient (PatientId),  FOREIGN KEY (AssociatedDoctorId) REFERENCES Doctor (DoctorId),  FOREIGN KEY (AssociatedNurseId) REFERENCES Nurse (NurseId),  FOREIGN KEY (BedId) REFERENCES Bed(BedId),  CHECK(DischargeDate >= AdmissionDate),  CHECK(BedId>0),  CHECK(NoOfDays>=0)  );  CREATE TABLE Bill  (  BillId int primary key auto\_increment,  PatientId int not null,  Date date not null,  AppointmentCharges decimal(10,2),  TestCharges decimal(10,2),  StayCharges decimal(10,2),  MedicineCharges decimal(10,2),  TotalBill decimal(10,2) not null,  InsuranceApplied boolean not null,  PaymentMode ENUM('Cash','Card','UPI','NET Banking') not null,  FOREIGN KEY (PatientId) REFERENCES Patient (PatientID)  ); |
| --- |

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

UserRole Insertion :

INSERT INTO UserRole (RoleID, RoleName) VALUES

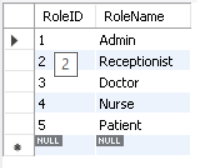
(1, 'Admin'),

(2, 'Receptionist'),

(3, 'Doctor'),

(4, 'Nurse'),

(5, 'Patient');



Address Insertion :

INSERT INTO Address (Street, City, State) VALUES

('123 Main St', 'Kalyan', 'Maharashtra'),

('456 Oak St', 'Chandigarh', 'Chandigarh'),

('56 Oak St', 'Raipur', 'Chattisgarh'),

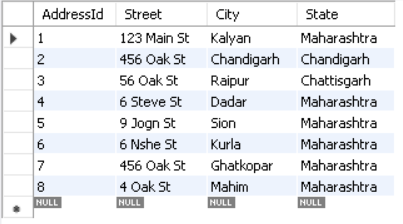
('6 Steve St', 'Dadar', 'Maharashtra'),

('9 Jogn St', 'Sion', 'Maharashtra'),

('6 Nshe St', 'Kurla', 'Maharashtra'),

('456 Oak St', 'Ghatkopar', 'Maharashtra'),

('4 Oak St', 'Mahim', 'Maharashtra');



UserDetail Insertion :

INSERT INTO UserDetail (RoleId, AddressId, FirstName, LastName, ContactNumber, DateOfBirth,Email, Password, bloodgroup) VALUES

(1,1,'Balaji','Satpute','7894561237','1992-02-14','balaji@email.com','12345678','O-'),

(2,2,'Navin','Modi','8945612371','1995-07-11','navin@org.com','32165498','B+'),

(3,3,'Abhijeet','Patil','9987456123','1994-03-19','patil@gmail.com','45612378','O+'),

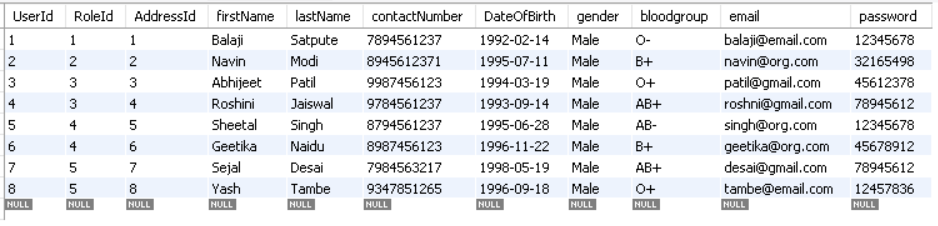
(3,4,'Roshini','Jaiswal','9784561237','1993-09-14','roshni@gmail.com','78945612','AB+'),

(4,5,'Sheetal','Singh','8794561237','1995-06-28','singh@org.com','12345678','AB-'),

(4,6,'Geetika','Naidu','8987456123','1996-11-22','geetika@org.com','45678912','B+'),

(5,7,'Sejal','Desai','7984563217','1998-05-19','desai@gmail.com','78945612','AB+'),

(5,8,'Yash','Tambe','9347851265','1996-09-18','tambe@email.com','12457836','O+');



Staff Record Insertion :

INSERT INTO Staff (UserId, Qualification, Salary,shift\_start\_time,shift\_end\_time) VALUES

(1, 'MCA', 125000.00,'09:00:00',’19:00:00’),

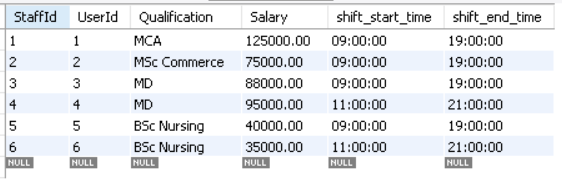
(2, 'MSc Commerce', 75000.00,'09:00:00',’19:00:00’),

(3, 'MD', 88000.00,'09:00:00',’19:00:00’),

(4, 'MD', 95000.00,'11:00:00',’21:00:00’),

(5, 'BSc Nursing', 40000.00,'09:00:00',’19:00:00’),

(6, 'BSc Nursing', 35000.00,'11:00:00',’21:00:00’);



Department Record Insertion :

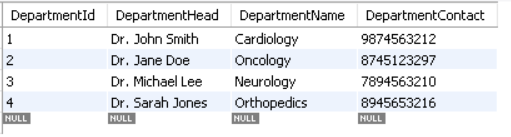
INSERT INTO Department (DepartmentId,DepartmentHead, DepartmentName,DepartmentContact) VALUES

(1,'Dr. John Smith', 'Cardiology',"9874563212"),

(2,'Dr. Jane Doe', 'Oncology',"8745123297"),

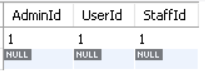
(3,'Dr. Michael Lee', 'Neurology',"7894563210"),

(4,'Dr. Sarah Jones', 'Orthopedics',"8945653216");



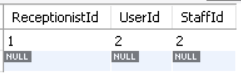
Admin Record Insertion :

INSERT INTO Admin(AdminId,UserId,StaffId) values(1,1,1);



Receptionist Record Insertion :

INSERT INTO Receptionist (ReceptionistId,UserId, StaffId) VALUES(1,2, 2);

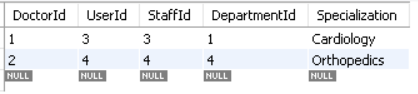


Doctor Record Insertion :

INSERT INTO Doctor (DoctorId, UserId, StaffId, DepartmentId, Specialization) VALUES

(1, 3, 3, 1, 'Cardiology'),

(2, 4, 4, 4, 'Orthopedics');

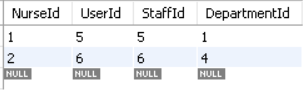


Nurse Record Insertion :

INSERT INTO Nurse (NurseId, UserId, StaffId, DepartmentId) VALUES

(1, 5, 5, 1),

(2, 6, 6, 4);

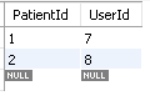


Patient Record Insertion :

INSERT INTO Patient(PatientId,UserId) VALUES

(1,7),

(2,8);

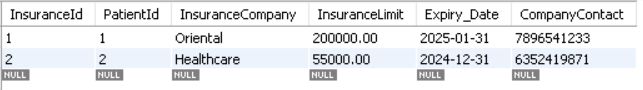


Insurance Record Insertion :

INSERT INTO Insurance (InsuranceId, PatientId, InsuranceCompany, InsuranceLimit, Expiry\_Date, CompanyContact) VALUES

(1, 1, 'Oriental', 200000.00, '2025-01-31',"7896541233"),

(2, 2, 'Healthcare', 55000.00, '2024-12-31',"6352419871");



**Q-2 Write a necessary to recognize already registered patients and user roles.**

I’m trying to insert the same patient which is already registered into Database.It will throw an error because I have added a unique constraint for email id and password.

Query :

INSERT INTO UserDetail (RoleId, AddressId, FirstName, LastName, ContactNumber, DateOfBirth,Email, Password,bloodgroup) VALUES

(5,8,'Yash','Tambe','9347851265','1996-09-18','tambe@email.com','12457836','O+');



**Q-3 Write necessary queries to add to the list of diagnosis of the patient tagged by date.**

Diagnosis Has Been done only after taking doctors Appointment.So first we have to insert record into appointment table:

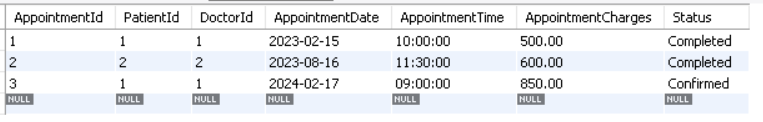
Appointment Record Insertion :

INSERT INTO Appointment (PatientId, DoctorId, AppointmentDate, AppointmentTime, AppointmentCharges, Status) VALUES

(1, 1, '2023-02-15', '10:00:00', 500.00, 'Completed'),

(2, 2, '2023-08-16', '11:30:00', 600.00, 'Completed'),

(1, 1, '2024-02-17', '09:00:00', 850.00, 'Confirmed');

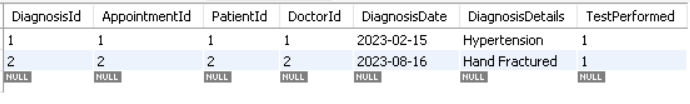


Diagnosis Record Insertion :

INSERT INTO Diagnosis (DiagnosisId, AppointmentId, PatientId, DoctorId, DiagnosisDate, DiagnosisDetails, TestPerformed) VALUES

(1, 1, 1, 1, '2023-02-15', 'Hypertension', true),

(2, 2, 2, 2, '2023-08-16', 'Hand Fractured', true);



Based On Diagnosis doctor suggest Prescription and Test to Patient :

Test Record Insertion :

INSERT INTO Test (TestName, TestCharges) VALUES

('Blood test', 1200.00),

('Electrocardiogram (ECG)', 3000.00),

('Blood Pressure Test', 1000.00),

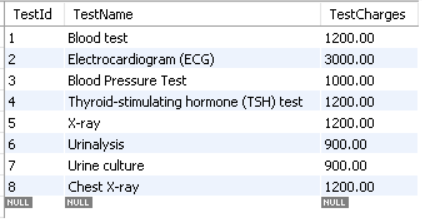
('Thyroid-stimulating hormone (TSH) test', 1200.00),

('X-ray', 1200.00),

('Urinalysis', 900.00),

('Urine culture', 900.00),

('Chest X-ray', 1200.00);

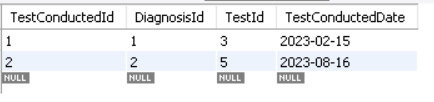


TestConducted Record Insertion :

INSERT INTO TestConducted (TestconductedId, DiagnosisId, TestId, TestConductedDate) VALUES

(1, 1, 3, '2023-02-15'),

(2, 2, 5, '2023-08-16');



Medicine Record Insertion :

INSERT INTO Medicine (Medicine\_ID, M\_Name, M\_Cost) VALUES

(1, 'Paracetamol', 55.00),

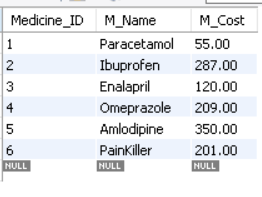
(2, 'Ibuprofen', 287.00),

(3, 'Enalapril', 120.00),

(4, 'Omeprazole', 209.00),

(5, 'Amlodipine', 350.0),

(6, 'PainKiller', 201.00);



Prescription Record Insertion :

INSERT INTO Prescription (Prescription\_ID, DiagnosisId, Medicine\_ID, dosagePerDay, instruction) VALUES

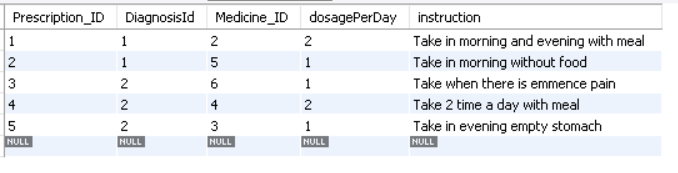
(1, 1, 2,2,"Take in morning and evening with meal"),

(2, 1, 5, 1,"Take in morning without food"),

(3, 2, 6, 1,"Take when there is emmense pain"),

(4, 2, 4, 2, "Take 2 time a day with meal"),

(5, 2, 3, 1, "Take in evening empty stomach");



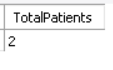
**Q-4. Write necessary queries to fetch required details of a particular patient.**

* Query to find count of patient in the system :

SELECT COUNT(\*) AS TotalPatients

FROM UserDetail

WHERE RoleId = 5;



* Query to find Confirmed Appointment details of Patient along with date ,time and Doctors Name

SELECT

A.AppointmentDate,

A.AppointmentTime,

CONCAT(DU.FirstName, ' ', DU.LastName) AS DoctorName

FROM

Appointment A

JOIN

Patient P ON A.PatientId = P.PatientId

JOIN

Doctor D ON A.DoctorId = D.DoctorId

JOIN

UserDetail DU ON D.UserId = DU.UserId

WHERE

P.PatientId = 1

AND A.Status = 'Confirmed';



* Query to find Patients Insurance detail with Name.

SELECT

P.PatientId,

CONCAT(UD.FirstName, ' ', UD.LastName) AS PatientName,

I.InsuranceId,

I.InsuranceCompany,

I.InsuranceLimit,

I.Expiry\_Date,

I.CompanyContact

FROM

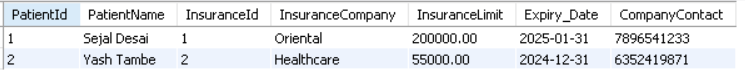
Patient P

JOIN

Insurance I ON P.PatientId = I.PatientId

JOIN

UserDetail UD ON P.UserId = UD.UserId;



* Query to find out Patient=”Yash” Diagnosis detail with prescription , Medicine Names and Test Conducted

SELECT

P.PatientId,

CONCAT(UD.FirstName, ' ', UD.LastName) AS PatientName,

D.DiagnosisDate,

D.DiagnosisDetails,

T.TestName,

TC.TestConductedDate,

M.M\_Name,

P.dosagePerDay,

P.instruction

FROM

Patient P

JOIN

Diagnosis D ON P.PatientId = D.PatientId

JOIN

TestConducted TC ON D.DiagnosisId = TC.DiagnosisId

JOIN

Test T ON TC.TestId = T.TestId

JOIN

Prescription P ON D.DiagnosisId = P.DiagnosisId

JOIN

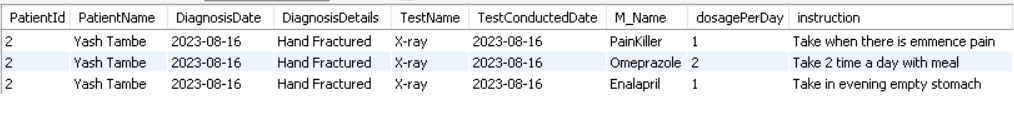
Medicine M ON P.Medicine\_ID = M.Medicine\_ID

JOIN

UserDetail UD ON P.UserId = UD.UserId

WHERE

UD.FirstName = 'Yash';



**Q-5. Write necessary queries to prepare a bill for the patient at the end of checkout.**

For each patient we have to calculate AppointmentCharges , Test Charges, Stay Charges and Medicine Charges.Once it's found we can sum all charges into total\_till and insert into Bill Table.

-- Calculate Appointment Charges

SET @appointment\_charges = (

SELECT COALESCE(SUM(AppointmentCharges), 0)

FROM Appointment

WHERE PatientId = patient\_id

);

-- Calculate Test Charges

SET @test\_charges = (

SELECT COALESCE(SUM(TestCharges), 0)

FROM TestConducted

JOIN Test ON TestConducted.TestId = Test.TestId

JOIN Diagnosis ON TestConducted.DiagnosisId = Diagnosis.DiagnosisId

WHERE Diagnosis.PatientId = patient\_id

);

-- Calculate Stay Charges

SET @stay\_charges = (

SELECT COALESCE(StayCharges, 0)

FROM Stay

WHERE PatientId = patient\_id

);

-- Calculate Medicine Charges

SET @medicine\_charges = (

SELECT COALESCE(SUM(M\_Cost), 0)

FROM Prescription

JOIN Medicine ON Prescription.Medicine\_ID = Medicine.Medicine\_ID

JOIN Diagnosis ON Prescription.DiagnosisId = Diagnosis.DiagnosisId

WHERE Diagnosis.PatientId = patient\_id

);

-- Calculate Total Bill

SET @total\_bill = COALESCE(@appointment\_charges, 0) + COALESCE(@test\_charges, 0) + COALESCE(@stay\_charges, 0) + COALESCE(@medicine\_charges, 0);

Scenario 1 : Patient Sejal had hypertension, which was treated with medication, and she did not require a hospital stay. Therefore, the bill was calculated for her based on appointment charges, test charges, and medicine charges.

INSERT INTO Bill (PatientId, Date, AppointmentCharges, TestCharges, StayCharges, MedicineCharges, TotalBill, InsuranceApplied, PaymentMode)

VALUES (

1,

"2023-02-15",

@appointment\_charges,

@test\_charges,

@stay\_charges,

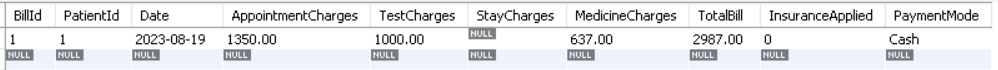
@medicine\_charges,

@total\_bill,

0,

'Cash'

);



Scenario 2 : Patient Yash was diagnosed with a fracture during the doctor's consultation. He opted for treatment at the hospital, and as a result, the bill was calculated for him, taking into account appointment charges, test charges, medicine charges, and stay charges.

INSERT INTO Bill (PatientId, Date, AppointmentCharges, TestCharges, StayCharges, MedicineCharges, TotalBill, InsuranceApplied, PaymentMode)

VALUES (

2,

"2023-02-15",

@appointment\_charges,

@test\_charges,

@stay\_charges,

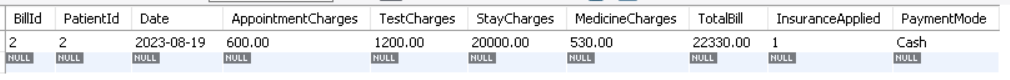
@medicine\_charges,

@total\_bill,

0,

'Cash'

);



This above process of calculating Bill may take many steps but it will be reduced by using stored procedures further.

**Q-6 Write necessary queries to fetch and show data from various related tables (Joins)**

* Query to find the ward and bed where a patient is present in the hospital along with the patient's name

SELECT

S.BedId,

B.Ward,

U.FirstName,

U.LastName

FROM

Stay S

JOIN

Bed B ON S.BedId = B.BedId

JOIN

Patient P ON S.PatientId = P.PatientId

JOIN

UserDetail U ON P.UserId = U.UserId;



* Query to find Doctor and Nurse Associated with Patient “Yash” in Hospital while in treatment along with their Department Name.

SELECT

S.StayId,

CONCAT(U.FirstName, ' ', U.LastName) AS PatientName,

CONCAT(Du.FirstName, ' ', Du.LastName) AS DoctorName,

D.Specialization AS Specialization,

CONCAT(Nu.FirstName, ' ', Nu.LastName) AS NurseName,

Dp.DepartmentName AS Department

FROM

Patient P

JOIN

Stay S ON P.PatientId = S.PatientId

JOIN

Doctor D ON S.AssociatedDoctorId = D.DoctorId

JOIN

Nurse N ON S.AssociatedNurseId = N.NurseId

JOIN

Department Dp ON D.DepartmentId = Dp.DepartmentId

JOIN

UserDetail Du ON D.UserId = Du.UserId

JOIN

UserDetail Nu ON N.UserId = Nu.UserId

JOIN

UserDetail U ON P.UserId=U.UserId

WHERE

U.FirstName ="Yash" and U.RoleId=5;



**Q-7 .Optimize repeated read operations using views/materialized views.**

The information related to appointment details which is not yet completed is frequently accessed by both doctors and patients. Therefore, it is beneficial to encapsulate this query in a view, allowing it to be called whenever needed by either party.

**View Creation :**

CREATE VIEW AppointmentDetailsView AS

SELECT

A.AppointmentId,

CONCAT(U\_Patient.firstName, ' ', U\_Patient.lastName) AS PatientName,

CONCAT(U\_Doctor.firstName, ' ', U\_Doctor.lastName) AS DoctorName,

A.AppointmentDate,

A.AppointmentTime,

A.AppointmentCharges,

A.Status

FROM

Appointment A

JOIN

Patient P ON A.PatientId = P.PatientId

JOIN

Doctor D ON A.DoctorId = D.DoctorId

JOIN

UserDetail U\_Patient ON P.UserId = U\_Patient.UserId

JOIN

UserDetail U\_Doctor ON D.UserId = U\_Doctor.UserId

WHERE

A.Status <> 'Completed' OR (A.AppointmentDate = CURRENT\_DATE() AND A.AppointmentTime > CURRENT\_TIME());

**View Call :**

SELECT \* FROM AppointmentDetailsView;

****

**Q-8 Optimize read operations using indexing wherever required. (Create index on at least 1 table)**

It is essential to create an index on the UserDetail table, particularly when querying user details frequently. This is crucial because the queries often involve joins and filtering based on the first name and last name columns. By having an index on the UserId column of the UserDetail table, read operations can be significantly optimized, resulting in reduced query execution times and improved overall performance.

Query :

CREATE INDEX idx\_userdetail\_userid ON UserDetail (UserId);

**Q-9. Try optimizing bill generation using stored procedures.**

DELIMITER //

CREATE PROCEDURE GenerateBill(

IN p\_PatientId INT,

IN bill\_Date Date,

IN p\_AppointmentId INT,

IN p\_DiagnosisId INT,

IN p\_TestConductedId INT,

IN p\_StayId INT,

IN p\_InsuranceApplied BOOLEAN,

IN p\_PaymentMode varchar(10)

)

BEGIN

DECLARE v\_AppointmentCharges DECIMAL(10,2);

DECLARE v\_DiagnosisCharges DECIMAL(10,2);

DECLARE v\_TestCharges DECIMAL(10,2);

DECLARE v\_MedicineCharges DECIMAL(10,2);

DECLARE v\_StayCharges DECIMAL(10,2);

DECLARE v\_TotalBill DECIMAL(10,2);

-- Get Appointment Charges

SELECT AppointmentCharges INTO v\_AppointmentCharges

FROM Appointment

WHERE AppointmentId = p\_AppointmentId;

-- Get Test Charges

SELECT coalesce(SUM(T.TestCharges),0)

INTO v\_TestCharges

FROM

TestConducted TC

JOIN

Test T ON TC.TestId = T.TestId

WHERE

TC.TestConductedId = p\_TestConductedId;

-- Get Medicine Charges

SELECT SUM(M.M\_Cost) INTO v\_MedicineCharges

FROM

Prescription P

JOIN

Medicine M ON P.Medicine\_ID = M.Medicine\_ID

WHERE

P.DiagnosisId = p\_DiagnosisId

GROUP BY

P.DiagnosisId;

-- Get Stay Charges

SELECT StayCharges INTO v\_StayCharges

FROM Stay

WHERE StayId = p\_StayId;

-- Calculate Total Bill

SET v\_TotalBill = COALESCE(v\_AppointmentCharges, 0) + COALESCE(v\_TestCharges, 0) + COALESCE(v\_MedicineCharges, 0) + COALESCE(v\_StayCharges, 0);

-- Insert into Bill table

INSERT INTO Bill (PatientId, Date, AppointmentCharges,TestCharges, MedicineCharges, StayCharges, TotalBill, InsuranceApplied, PaymentMode)

VALUES (p\_PatientId, bill\_Date, v\_AppointmentCharges, v\_TestCharges, v\_MedicineCharges, v\_StayCharges, v\_TotalBill, p\_InsuranceApplied, p\_PaymentMode);

- - Print Bill Records

SELECT

CONCAT(U.firstName, ' ', U.lastName) AS PatientName,

IFNULL(S.AdmissionDate, null) AS AdmissionDate,

IFNULL(S.DischargeDate, null) AS DischargeDate,

B.BillId,

B.Date,

B.AppointmentCharges,

B.TestCharges,

B.StayCharges,

B.MedicineCharges,

B.TotalBill,

B.InsuranceApplied,

B.PaymentMode

FROM

Patient P

LEFT JOIN

Stay S ON P.PatientId = S.PatientId

LEFT JOIN

Bill B ON P.PatientId = B.PatientId

JOIN

UserDetail U ON U.UserId = P.UserId

WHERE

P.PatientId = p\_PatientId;

END //

DELIMITER ;

– Call Procedure

CALL GenerateBill(

2, -- PatientId

"2023-08-19", -- Date oF Bill

2, -- AppointmentId

2, -- DiagnosisId

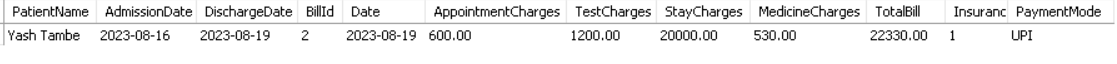
2, -- TestConductedId

1, -- StayId

true, -- InsuranceApplied status

"UPI" -- PaymentMode

);



**Q-10. - Add necessary triggers to indicate when a patient's medical insurance limit has expired.**

**Query 1**: Create a trigger , that automatically triggers a message to indicate if a patient's medical insurance has expired when inserting a new record into the Insurance table. The trigger should compare the current date with the Insurance Expiry date while inserting a record.

DELIMITER //

CREATE TRIGGER CheckInsuranceExpiry

BEFORE INSERT ON Insurance

FOR EACH ROW

BEGIN

DECLARE today DATE;

DECLARE insuranceExpired BOOLEAN;

SET today = CURDATE();

SET insuranceExpired = NEW.Expiry\_Date < today;

IF insuranceExpired THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insurance has expired for Patient';

END IF;

END;

//

DELIMITER ;

**Trying to insert Insurance detail with invalid date.**

Insert into Insurance (PatientId, InsuranceCompany, InsuranceLimit, Expiry\_Date,CompanyContact) values (1, 'WellCare', 82000.00, '2023-09-30',"9087364560");



**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.**

**Design Pattern :**

As we dive into the world of application development,software developers encounter a bunch of challenges. These range from figuring out how to structure our system effectively to dealing with issues like repeating code in different places, making sure our objects communicate properly, adding new updates without messing up existing code, and trying to test individual parts of our program.

If developers do not use any generalized solution then they might need to undergo all such difficulties.To avoid all these problems Design pattern helps us.

Design patterns are general and reusable solutions to commonly occurring problems in software development. Design patterns are not specific to any technology but rather it's a logical way of how to design and organize an application to avoid such recurring problems.

There are a number of design patterns present. Each one has its own advantages. One cant say that use of a specific design pattern is good to use in a project.Rather its completely depends on project requirements and scenarios.

It's not true that in a single project you can use only one pattern, you can use multiple design patterns in a single project. Suppose you have an application where you want store each user activity record in log file and also you have an requirement that to hide object creation process from user then in such cases you may have to use 2 separate design patterns.Singleton design pattern is useful for log scenario where you just need one global object which can insert all activities of user in that log file and for hiding object creation you may use factory design pattern.

It's not like every developer may know all design patterns but there are some important patterns that most of the time fits into project requiremnt.Its like only 20% pattern used for 80% of the time. But having just a basic understanding of all patterns helps developers to analyze client requirements and basis on that deciding design pattern to use.

One exciting aspect of design patterns is their ability to promote loose coupling between components. This not only makes our code more adaptable to changes but also facilitates the replacement or extension of parts without causing a domino effect.

Properly applied design patterns can enhance system performance by promoting efficient code organization and reducing redundancy. However, poorly chosen or overused patterns may introduce unnecessary complexity, potentially impacting performance.

Overusing design patterns without considering the context and complexity of the project can lead to unnecessarily complex code. It’s important to apply design patterns judiciously, where they add value and improve code structure.

Design patterns are categorized into 3 types :

* Creational design pattern
* Structural design pattern
* Behavioral design pattern

**Creational Design Pattern** : Creational design patterns provide different ways of creating objects,it's like a blueprint that guides us on how to create instances of objects in a more flexible and reusable way.

Types of Creational Design Pattern : Singleton, Prototype, Factory, Abstract Factory, Builder design pattern.

**Structural Design Pattern** : Structural design patterns are concerned with how classes and objects can be combined to form larger structures, and they address issues related to object composition, interface design, and class inheritance.

Type of Structural design pattern : Adapter, Bridge, Composite, Decorator, Facade, Flyweight and proxy design pattern.

**Behavioral design pattern** : This design pattern deals with interaction and responsibilities of objects.These patterns focus on how objects communicate and work together to achieve common tasks.

Types of Behavioral design pattern : Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.

**Rendering Pattern :**

Rendering patterns are ways in which we decide how HTML,CSS and Javascript code is all processed and rendered in a web application.There are certain important factors we have to consider before deciding which pattern tobe used.Factors include :

1. Render Time
2. Dynamic Content
3. Search Engine Optimization
4. Build Time
5. Content Updation

Based on client requirements and uses of end user applications we have to decide a pattern. How different patterns are helpful and in which scenario i have explained below :

**Client Side Rendering (CSR):**

CSR is a type of rendering in which a web page is primarily handled on the client's browser rather than on the server.Whenever User sends request to server for web page access, server sends barebone HTML document along with CSS and javascript files to the client and the client's browser takes on the responsibility of rendering the page, fetching data, and updating the user interface dynamically. It’s a responsibility of javascript to build a web page at client browser as per the need. CSR is used by React.As we have many benefits of CSR, but still it presents some challenges in Search engine optimization. As in CSR only basic HTML skeleton is provided along with css and javascript scripts tag to client first, it may look like there is not much HTML content present in the webpage and search engines ignore the page to improve ranking and search indexing will also be reduced.

Application **:**

* Single Page Application : CSR is often associated with Single Page Application where a single HTML page is loaded initially, and subsequent interactions of the client dynamically update the content without full page reloads.Basically SPA follows component based architecture where only part of website is loaded as per request by changing dom elements.
* Highly Interactive applications : For applications that need a lot of interaction and quick updates, Client-Side Rendering (CSR) is quite helpful. CSR utilizes asynchronous programming, which means it can update parts of a webpage without putting too much strain on the server or using too much server resources.
* Social media feeds : CSR allows for seamless updates to the content of social media feeds as new posts, comments, or likes occur without requiring a full page reload.

**Server Side Rendering (SSR) :**

In Server-Side Rendering (SSR), the whole process of creating web pages occurs on the server before sending them to the user's device. Unlike Client-Side Rendering (CSR), where the browser does the work, in SSR, the server has everything it needs for each request. It gets the data, arranges the page's look, and builds the entire webpage. After making sure everything is in place, the server sends the complete page to the user's device. So, SSR is like having the server do all the heavy lifting before giving you the finished webpage. SSR reduces the amount of Javascript loaded and executed on the client's browser.

Additionally, SSR also enables web crawlers and Search engine optimizer to index page ranking,

as the complete HTML content is available on the initial page load.

Server SIde rendering has Faster initial load time than the Client side rendering.

Example : NextJs follows the Server side rendering pattern.

Application : Financial and Banking websites always prioritize security, reliability, and efficient content delivery. SSR is more secure for rendering sensitive financial data on the server side before sending it to the client. Sensitive information, such as account balances, transaction history, and personal details, can be controlled and managed on the server, reducing the risk of exposing critical data on the client side.

If we use Client side rendering here then additional precautions are required to prevent data leakage and ensure the protection of user information.

**Static Site Generation (SSG) :**

Static site generation (SSG) is a technique for building web pages by pre-generating HTML, CSS, and JavaScript files at build time instead of rendering them on the server or client-side.

As we don't have any dynamic content to serve to users. It can be loaded from a web server or CDN(Content delivery network) to reduce latency for users present in different geographical locations and it can also be deployed on Serverless architecture.

Applications :

* We can use Static Site Generation (SSG) for websites that mostly have fixed, unchanging content to show users. Pages like "About Us," "Contact Us," and "Product Landing Page" don't usually have things that change a lot, so putting them on an SSG network makes them load really fast.
* For blogs or websites with articles, where the templates (the look and feel of the pages) and all the things like content, pictures, videos, and audio are pretty much the same for every page, using SSG is a great idea. It helps these websites load quickly and work effectively.

SSG has some limitations in terms of dynamic content and interactivity. Since the content is generated at build time, any website which requires real time updates and frequent changes like social media sites and ecommerce sites with frequent change in availability of product cannot be served using Static site generation.

Another disadvantage is Build time may take a significant amount of time while generating pages.But once pages are built it can be rendered from server very fast.

**Incremental Static Regeneration :**

Incremental Static regeneration is an extended version of Static site generation.While Both involves Pre-rendering the pages at build time, ISR introduces the ability to update the build pages on fly, without requiring a full build.

Consider a example of Blog where articles are written and published. In Static site generation, if you add a new article or update an existing one, you need to rebuild the entire blog to reflect the changes. But in case of ISR, you can add a new article or make changes to an existing one, and only the affected pages are regenerated in the background. Readers visiting the blog may see the updates without the need for a full rebuild.

Use Case : ISR is useful for websites that need a mix of static and frequently updated content. It's especially beneficial for scenarios where you want to provide dynamic updates without compromising the performance benefits of static rendering.

Application : Content Management System where content creators frequently update their articles, blogs or other pieces of information.

**Streaming Server Side Rendering :**

Streaming server side rendering is a rendering pattern for web development that involves sending server generated HTML to the client as soon as it's available, instead of waiting for the entire page to be rendered before sending it.

With traditional Server Side Rendering, the server would wait for the entire page to be rendered before sending it to the client, resulting in a longer time to first byte (TTFB) and a slower user experience.Streaming SSR allows the server to send the HTML to the client in chunks as it is generated, providing a faster TTFB and a more responsive user experience.

One challenge with streaming SSR is ensuring that the chunks of HTML are sent to the client in the correct order and that the page remains coherent as it is rendered.

**Best for** :

* Applications with large pages or media that require a long time to load.
* Applications that require a smooth user experience with minimal waiting time.