# Hospital Management System - Database Design Whitepaper

## Executive Summary

The Hospital Management System is a comprehensive database-driven healthcare management solution built on PostgreSQL 15. This whitepaper documents the database design, implementation, and advanced database features including stored procedures, triggers, and materialized views. The system implements a fully normalized database (3NF) with 20 tables, 10+ stored procedures, 10+ triggers, and 10 materialized views to manage hospital operations efficiently.

---

## 1. Introduction

### 1.1 Problem Statement

Traditional hospital management systems suffer from:

- **Data Redundancy**: Unnormalized databases leading to inconsistency and storage inefficiency
- **Manual Business Logic**: Complex operations requiring multiple manual steps prone to errors
- **Limited Analytics**: Inability to generate real-time reports and insights
- **Data Integrity Issues**: Lack of automated validation and constraint enforcement

### 1.2 Solution Approach

This system addresses these challenges through:

- **Fully Normalized Database (3NF)**: Eliminates redundancy and ensures data consistency
- **Stored Procedures**: Encapsulates complex business logic at database level
- **Triggers**: Automates data integrity checks and business rule enforcement
- **Materialized Views**: Provides fast access to aggregated analytics data
- **Comprehensive Constraints**: Ensures data validity through CHECK, UNIQUE, and foreign key constraints

### 1.3 Project Objectives

1. Design and implement a fully normalized database schema (3NF)
2. Implement automated business logic through stored procedures and triggers
3. Create comprehensive reporting capabilities using materialized views
4. Ensure data integrity through database-level constraints
5. Implement complete audit trail for data tracking and compliance

---

## 2. Database Architecture

### 2.1 Database Technology

- **RDBMS**: PostgreSQL 15
- **Programming Language**: PL/pgSQL for stored procedures and triggers
- **Features Utilized**: Stored Procedures, Triggers, Materialized Views, Indexes, Constraints

### 2.2 Database Structure

```
PostgreSQL 15 Database
│
├── Schema Layer (20 Tables)
│   ├── Core Entity Tables (10)
│   ├── Transactional Tables (5)
│   ├── Junction Tables (2)
│   ├── Allocation Tables (1)
│   └── System Tables (2)
│
├── Stored Procedures Layer (10+)
│   ├── Business Logic Functions
│   ├── Data Validation
│   └── Complex Query Operations
│
├── Triggers Layer (10+)
│   ├── Data Integrity Enforcement
│   ├── Automatic Calculations
│   └── Audit Logging
│
└── Views Layer
    ├── Materialized Views (10) - Analytics & Reporting
    └── Regular Views (4) - Real-time Data Access
```

---

# 3. Database Schema Design

## 3.1 Schema Overview

The database consists of **20 tables** organized into logical groups:

**Core Entity Tables (10 tables):**

1. **Department** - Hospital departments and locations
2. **Specialization** - Medical specializations linked to departments
3. **Doctor** - Doctor profiles, qualifications, and availability
4. **Admin** - Administrative staff information
5. **Patient** - Patient demographics and contact information
6. **Address** - Normalized address information
7. **Disease** - Disease catalog with categories
8. **Medicine** - Medicine inventory with pricing and stock
9. **Room** - Hospital rooms with types and availability
10. **Insurance** - Patient insurance policies

**Transactional Tables (5 tables):**

11. **Appointment** - Appointment scheduling and management
12. **Medical_Record** - Patient visit records and diagnoses
13. **Prescription** - Doctor prescriptions
14. **Billing** - Financial transactions and invoices
15. **Lab_Test** - Laboratory test requests and results

**Junction Tables (2 tables):**

16. **Patient_Disease** - Patient-disease M:N relationship

17. **Prescription_Medicine** - Prescription-medicine M:N relationship

**Allocation Tables (1 table):**

18. **Patient_Room** - Room assignment and admission records

**System Tables (2 tables):**

19. **User_Login** - Authentication and authorization
20. **Audit_Log** - Complete audit trail for compliance

### 3.2 Normalization Steps

**UNF → 1NF:**

```
-- Before: Repeating groups
Patient(Patient_ID, Name, Email, Diseases: "Diabetes, Hypertension")

-- After: Atomic values
Patient(Patient_ID, Name, Email)
Patient_Disease(Patient_ID, Disease_ID, Diagnosis_Date)
```

**1NF → 2NF:**

```
-- Before: Partial dependency
Prescription_Medicine(Prescription_ID, Medicine_ID, Medicine_Name, Price, Dosage)

-- After: Removed partial dependency
Medicine(Medicine_ID, Medicine_Name, Price)
Prescription_Medicine(Prescription_ID, Medicine_ID, Dosage, Quantity)
```

**2NF → 3NF:**

```
-- Before: Transitive dependency
Doctor(Doctor_ID, Name, Specialization_Name, Department_Name, Location)

-- After: Removed transitive dependency
Department(Department_ID, Department_Name, Location)
Specialization(Specialization_ID, Specialization_Name, Department_ID)
Doctor(Doctor_ID, Name, Specialization_ID)
```

**Beyond 3NF:** Database complies with BCNF, 4NF, and 5NF through proper junction tables and key constraints.

### 3.3 Normalization Implementation Details

The database design demonstrates normalization principles through specific implementation choices.

**Address Normalization**

- Separate `Address` table with Address_ID as primary key
- Patient table references Address_ID via foreign key
- Multiple patients can share same address without redundancy
- Address updates automatically reflect in all related records

**Department-Specialization Normalization**

- Created `Department` table (Department_ID, Department_Name, Location)
- Created `Specialization` table (Specialization_ID, Specialization_Name, Department_ID)
- Doctor table references only Specialization_ID
- Specialization references Department_ID
- Eliminates transitive dependencies and ensures single source of truth

**Many-to-Many Relationships**

- **Patient-Disease**: Junction table `Patient_Disease` with composite primary key (Patient_ID, Disease_ID, Diagnosis_Date)
- **Prescription-Medicine**: Junction table `Prescription_Medicine` with composite primary key (Prescription_ID, Medicine_ID)

### 3.4 Design Principles Applied

**Referential Integrity:**

- All foreign keys properly defined with appropriate ON DELETE actions
- Cascade deletes for dependent records (patient-related data)
- Set NULL for optional relationships (doctor specialization)
- Prevent deletion for critical records (active appointments)

**Data Consistency:**

- CHECK constraints for valid data ranges (age: 0-150, blood group: A+, A-, B+, etc.)
- UNIQUE constraints for business rules (email, contact numbers, room numbers)
- DEFAULT values for common cases
- NOT NULL constraints for required fields

**Performance Optimization:**

- Strategic indexes on foreign keys (15+ indexes)
- Indexes on frequently queried columns (appointment dates, payment status)
- Composite indexes for multi-column queries
- Materialized views for complex aggregations

**Temporal Data Management:**

- Created_At and Updated_At timestamps on all tables
- Automatic timestamp updates via triggers
- Date-based partitioning capability for historical data

---

## 4. Stored Procedures

### 4.1 Implemented Stored Procedures

1. `book_appointment()`

- Validates doctor availability status from Doctor table
- Checks for time slot conflicts using UNIQUE constraint on (Doctor_ID, Appointment_Date, Appointment_Time)
- Prevents double booking at database level
- Returns success/failure status with descriptive messages

2. `generate_bill()`

- Calculates total charges (consultation, medicine, lab, room)
- Automatically queries Insurance table for active policies
- Applies insurance coverage percentage using SQL calculations
- Calculates discounts and final amount
- Creates billing record with all calculated values

3. `get_patient_history()`

- Joins multiple tables: Appointment, Medical_Record, Prescription, Patient_Disease, Billing
- Aggregates information from multiple sources
- Returns comprehensive patient profile with complete medical history

4. `assign_room()`

- Checks room availability status from Room table
- Validates room type suitability
- Creates admission record in Patient_Room table
- Triggers automatic room status update via trigger

5. `discharge_patient()`

- Calculates total room charges based on duration (Discharge_Date - Admission_Date)
- Retrieves Charges_Per_Day from Room table
- Generates final bill including room charges
- Updates room availability through trigger

6. `cancel_appointment()`

- Validates cancellation rules (time restrictions)
- Updates appointment status to 'Cancelled'
- Creates audit log entry via trigger
- Handles related billing adjustments

7. `get_available_doctors()`

- Lists doctors by specialization with availability filter
- Checks current schedule for conflicts in Appointment table
- Returns real-time availability status

8. `create_prescription()`

- Validates doctor-patient relationship
- Creates prescription record in Prescription table
- Links medicines with dosage information in Prescription_Medicine junction table
- Triggers automatic inventory update via trigger

9. `get_doctor_schedule()`

- Returns doctor appointments for specified date range
- Joins with Patient table for patient information
- Sorted by date and time
- Shows appointment status

10. `search_patients()`

- Multi-criteria patient search (name, contact, email, ID)
- Supports partial matching using LIKE operator

- Returns matching patients with key information

---

# 5. Triggers

## 5.1 Implemented Triggers

1. `check_doctor_availability`
   - **Event**: BEFORE INSERT/UPDATE on Appointment
   - **Purpose**: Prevent double booking
   - **Logic**: Queries Appointment table for conflicting appointments before allowing insert/update
   - **Action**: Raises exception if conflict detected
2. `generate_bill_auto`
   - **Event**: AFTER INSERT/UPDATE on Appointment
   - **Purpose**: Auto-generate bills when appointment completed
   - **Logic**: Detects status change to 'Completed', retrieves Consultation_Charges from Doctor table, creates bill in Billing table
   - **Action**: Inserts billing record with consultation charges
3. `update_room_status`
   - **Event**: AFTER INSERT/UPDATE on Patient_Room
   - **Purpose**: Keep room status synchronized with occupancy
   - **Logic**: Sets room to 'Occupied' on admission (INSERT), 'Available' on discharge (UPDATE with Discharge_Date)
   - **Action**: Updates Room table Availability_Status
4. `log_admin_actions`
   - **Event**: AFTER INSERT/UPDATE/DELETE on Admin table
   - **Purpose**: Track all administrative changes for audit
   - **Logic**: Captures old and new values in JSONB format
   - **Action**: Inserts audit log entry in Audit_Log table with complete change history
5. `update_inventory`
   - **Event**: AFTER INSERT on Prescription_Medicine
   - **Purpose**: Automatically update medicine stock
   - **Logic**: Decreases Stock_Quantity by prescribed Quantity from Prescription_Medicine table
   - **Action**: Updates Medicine table Stock_Quantity
6. `validate_appointment_date`
   - **Event**: BEFORE INSERT/UPDATE on Appointment
   - **Purpose**: Ensure appointment dates are valid
   - **Logic**: Prevents past date appointments (except for historical records)
   - **Action**: Raises exception if date validation fails
7. `update_patient_age`
   - **Event**: BEFORE INSERT/UPDATE on Patient
   - **Purpose**: Auto-calculate age from date of birth
   - **Logic**: Calculates age from DOB field using PostgreSQL date functions
   - **Action**: Updates Age field automatically in Patient table

8. `log_critical_changes`

- **Event**: AFTER UPDATE on critical tables (Patient, Doctor, Billing)
- **Purpose**: Track changes to critical data
- **Logic**: Compares old and new values using JSONB, logs differences
- **Action**: Creates detailed audit log entries in Audit_Log table

9. `prevent_appointment_deletion`

- **Event**: BEFORE DELETE on Appointment
- **Purpose**: Prevent deletion of active appointments
- **Logic**: Checks appointment status and date from Appointment table
- **Action**: Raises exception if deletion not allowed

10. `update_payment_date`

- **Event**: BEFORE UPDATE on Billing
- **Purpose**: Auto-set payment date on bill payment
- **Logic**: Detects payment status change to 'Paid' in Billing table
- **Action**: Sets Payment_Date to CURRENT_DATE

---

# 6. Materialized Views

## 6.1 Implemented Materialized Views

1. `mv_doctor_performance`

- Aggregates doctor appointment statistics from Appointment and Billing tables
- Joins Doctor, Specialization, and Appointment tables
- Calculates revenue generated per doctor
- Metrics: Total appointments, completed appointments, unique patients, total revenue, average bill amount

2. `mv_patient_visit_summary`

- Comprehensive patient visit history joining Patient, Appointment, Medical_Record, Patient_Disease, and Billing tables
- Aggregates medical records count
- Tracks disease history
- Calculates billing summary with pending bills

3. `mv_department_statistics`

- Department-wise performance metrics
- Joins Department, Specialization, Doctor, Appointment, and Billing tables
- Doctor count per department
- Appointment and patient statistics
- Revenue contribution by department

4. `mv_revenue_report`

- Monthly revenue breakdown using DATE_TRUNC function on Billing table
- Payment method analysis (Cash, Card, UPI, Insurance)
- Pending vs collected amounts
- Discount tracking

5. `mv_room_occupancy`

- Room utilization statistics joining Room and Patient_Room tables
- Occupancy rates by room type
- Average stay duration calculation (Discharge_Date - Admission_Date)
- Revenue from room charges

6. `mv_disease_prevalence`

- Disease occurrence tracking joining Disease and Patient_Disease tables
- Patient count per disease
- Age group distribution from Patient table
- Trend analysis data

7. `mv_lab_test_report`

- Lab test statistics from Lab_Test table
- Test completion rates
- Revenue from lab tests
- Test type distribution

8. `mv_appointment_trends`

- Appointment booking trends from Appointment table
- Peak hours analysis
- Cancellation rates
- No-show statistics

9. `mv_medicine_inventory`

- Current stock status from Medicine table
- Low stock alerts using WHERE Stock_Quantity < threshold
- Medicine usage patterns from Prescription_Medicine table
- Reorder recommendations

10. `mv_insurance_summary`

- Insurance claim statistics joining Insurance and Billing tables
- Coverage utilization
- Claim approval rates
- Insurance provider analysis

### 6.2 Regular Views

- `v_patient_complete_info` : Complete patient profile joining Patient, Address, Insurance tables
- `v_doctor_details` : Doctor information with specialization and department details joining Doctor, Specialization, Department tables
- `v_upcoming_appointments` : Real-time list of scheduled appointments with status filter from Appointment table
- `v_pending_bills` : Current pending bills with patient information joining Billing and Patient tables

---

## 7. Database Operations and Workflows

### 7.1 Appointment Management

**Booking Process:**

1. Stored procedure `book_appointment()` validates doctor availability from Doctor table

2. Procedure checks for time conflicts using UNIQUE constraint on Appointment table
3. Trigger `check_doctor_availability` prevents double booking at database level
4. Appointment created with 'Scheduled' status
5. Trigger logs appointment creation in Audit_Log table

**Completion Process:**

1. Appointment status updated to 'Completed' in Appointment table
2. Trigger `generate_bill_auto` automatically generates bill in Billing table
3. Consultation charges retrieved from Doctor table
4. Medical record can be created in Medical_Record table
5. Prescription can be added using `create_prescription()` procedure

### 7.2 Billing and Insurance

**Bill Generation:**

1. Trigger `generate_bill_auto` fires on appointment completion
2. Stored procedure `generate_bill()` calculates total charges
3. Insurance coverage automatically applied by querying Insurance table
4. Coverage percentage applied using SQL calculations
5. Final amount computed using GENERATED column: `Final_Amount = Total_Amount - Discount_Amount`

**Payment Processing:**

1. Payment status updated to 'Paid' in Billing table
2. Trigger `update_payment_date` sets Payment_Date automatically
3. Transaction logged in Audit_Log table

### 7.3 Room Management

**Admission Process:**

1. Stored procedure `assign_room()` validates room availability from Room table
2. Patient_Room record created with admission date
3. Trigger `update_room_status` sets room to 'Occupied' in Room table
4. Room charges begin accumulating based on Charges_Per_Day from Room table

**Discharge Process:**

1. Discharge date set in Patient_Room record
2. Trigger `update_room_status` sets room to 'Available' in Room table
3. Stored procedure `discharge_patient()` calculates total room charges
4. Bill generated including room charges in Billing table

### 7.4 Prescription Management

**Prescription Creation:**

1. Doctor creates prescription using `create_prescription()` procedure
2. Medicines added to Prescription_Medicine junction table
3. Trigger `update_inventory` automatically decreases medicine stock in Medicine table
4. Medicine charges calculated and added to bill in Billing table

---

# 8. Database Constraints and Security

## 8.1 Database Constraints

**Foreign Key Constraints:**

- All foreign keys properly defined with appropriate ON DELETE actions
- Cascade deletes for dependent records (patient-related data)
- Set NULL for optional relationships (doctor specialization)
- Prevent deletion for critical records (active appointments)

**CHECK Constraints:**

- Age validation: CHECK (Age >= 0 AND Age <= 150) for Patient, CHECK (Age >= 23 AND Age <= 100) for Doctor
- Blood group: CHECK (Blood_Group IN ('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'))
- Status values: CHECK (Status IN ('Active', 'Recovered', 'Chronic', 'Under Treatment'))
- Gender: CHECK (Gender IN ('Male', 'Female', 'Other'))

**UNIQUE Constraints:**

- Email addresses (Doctor, Patient, Admin tables)
- Contact numbers (Doctor, Patient, Admin tables)
- Room numbers (Room table)
- Policy numbers (Insurance table)
- Composite UNIQUE on (Doctor_ID, Appointment_Date, Appointment_Time) in Appointment table

**NOT NULL Constraints:**

- Required fields enforced: First_Name, Last_Name, Email in all entity tables
- Critical fields: Appointment_Date, Appointment_Time in Appointment table

## 8.2 Transaction Support

- ACID properties ensure data consistency
- Transaction isolation prevents data corruption
- Rollback capability for error recovery
- Stored procedures execute within transactions

## 8.3 Audit Trail

**Audit_Log Table Structure:**

- Log_ID (Primary Key)
- User_ID (Foreign Key to User_Login)
- Action_Type (INSERT, UPDATE, DELETE)
- Table_Affected
- Record_ID
- Old_Values (JSONB format)
- New_Values (JSONB format)
- Action_Timestamp

**Automatic Logging:**

- Triggers on critical tables (Patient, Doctor, Billing, Admin)
- Captures old and new values in JSONB format
- Enables point-in-time restoration capability

- Supports compliance reporting

---

# 9. Database Performance Optimization

## 9.1 Indexing Strategy

- Primary keys indexed (all 20 tables)
- Foreign keys indexed: Doctor(Specialization_ID), Patient(Address_ID), Appointment(Doctor_ID, Patient_ID), Medical_Record(Patient_ID), Billing(Patient_ID), Lab_Test(Patient_ID)
- Query columns indexed: Appointment(Appointment_Date), Billing(Payment_Status)
- Composite and unique indexes for multi-column queries and constraints

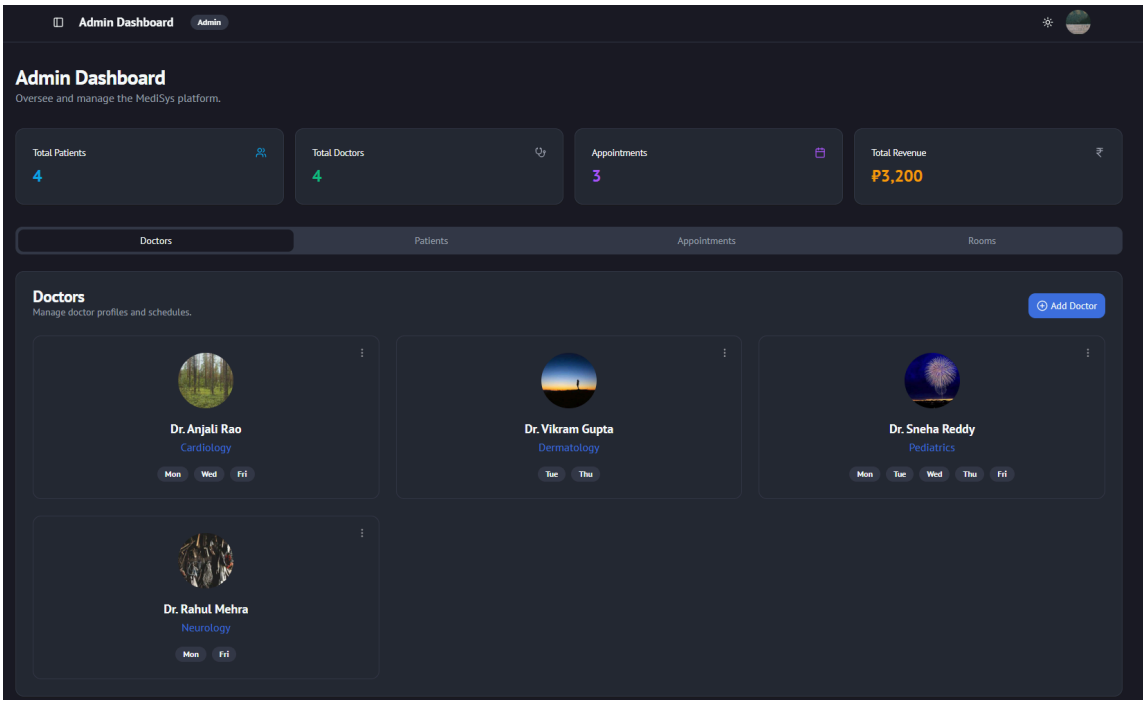
## 9.2 Query Optimization Techniques

- Materialized views (10) for expensive aggregations
- Optimized JOIN strategies and WHERE clauses using indexed columns
- Stored procedures for complex operations
- LIMIT and DATE_TRUNC for efficient data retrieval

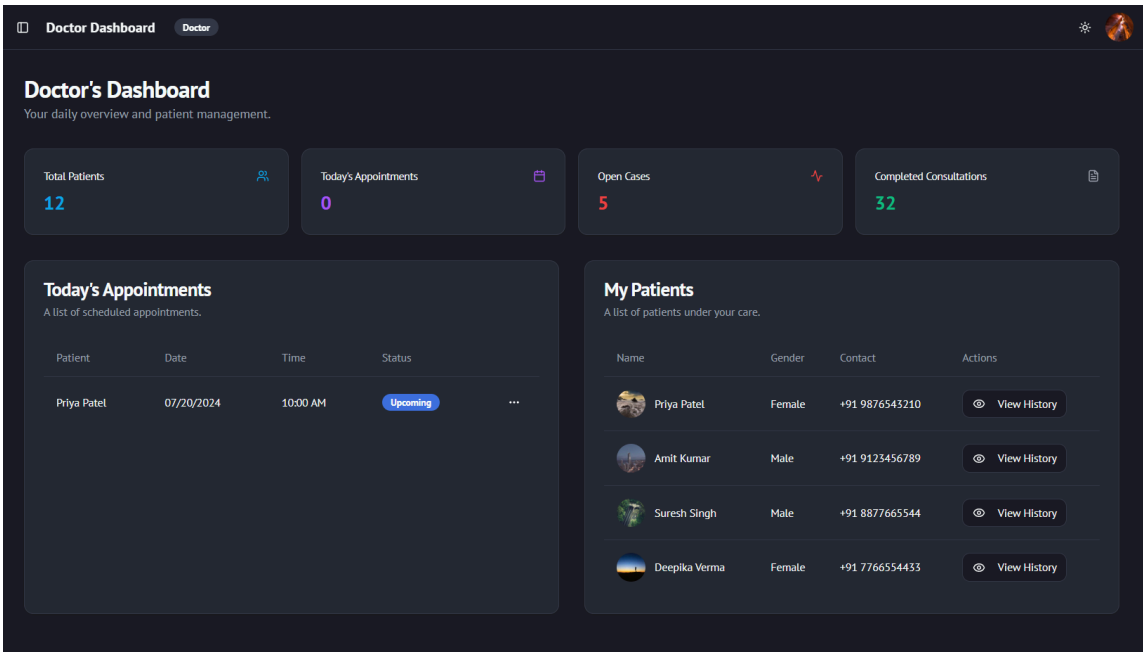
---

# 10. System Screenshots

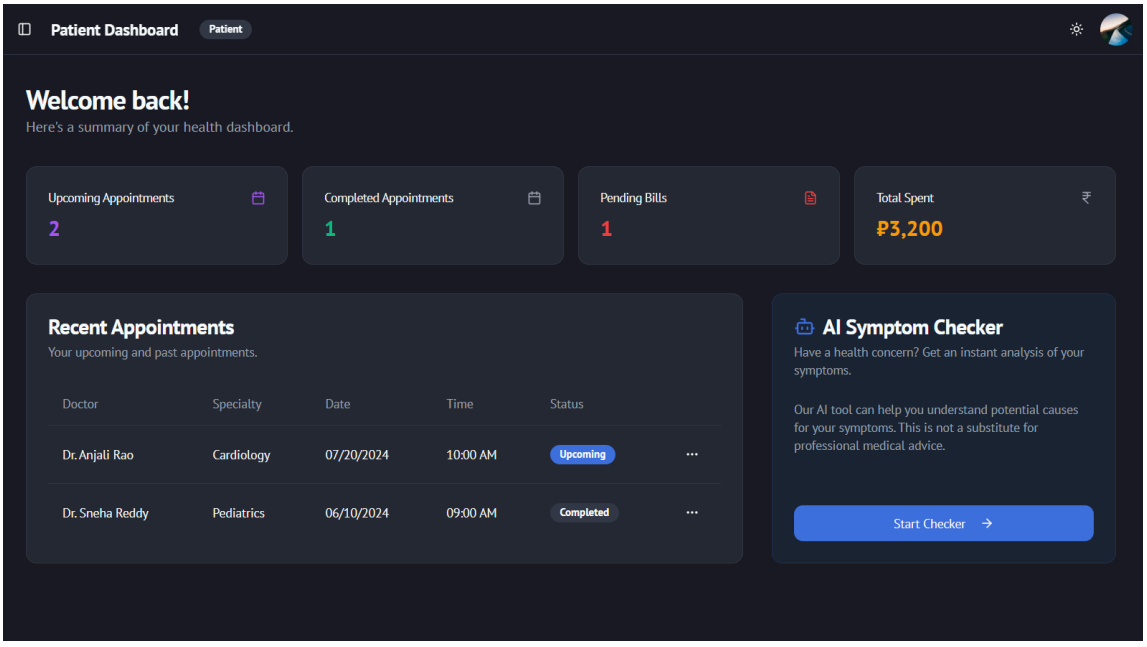
## 10.1 Admin Interface



### 10.2 Doctor Interface




### 10.3 Patient Interface




---

## 11. Conclusion

The Hospital Management System demonstrates comprehensive database design and implementation. The fully normalized database (3NF) eliminates redundancy and ensures data consistency. Through stored procedures, triggers, and materialized views, the system provides automation, data integrity, and performance optimization.

**Key Database Achievements:**

1. **Fully Normalized Database (3NF)**: Eliminates redundancy and ensures data consistency
2. **Automated Business Logic**: 10+ stored procedures handle complex operations at database level
3. **Data Integrity Enforcement**: 10+ triggers automatically enforce business rules and maintain data consistency
4. **Comprehensive Reporting**: 10 materialized views provide fast access to analytics and insights
5. **Complete CRUD Operations**: All entities support full Create, Read, Update, Delete operations through stored procedures
6. **Security and Audit**: Comprehensive audit logging via triggers on critical tables
7. **Performance Optimization**: Strategic indexing (15+ indexes) and materialized views ensure efficient query performance

**Technical Implementation:**

- 20 normalized tables with proper relationships
- 10+ stored procedures for business logic
- 10+ triggers for automation and integrity
- 10 materialized views for analytics
- 15+ indexes for performance
- Comprehensive constraints for data validation

The database design provides a robust foundation for healthcare management operations with production-ready features including constraint enforcement, automated business logic, performance optimization, and complete audit trails.

---

# Appendix A: Database Schema Summary

## Entity Relationships

**One-to-Many Relationships:**

- Department → Specialization → Doctor
- Address → Patient
- Doctor → Appointment, Medical_Record, Prescription
- Patient → Appointment, Medical_Record, Prescription, Insurance, Billing, Lab_Test
- Room → Patient_Room

**Many-to-Many Relationships:**

- Patient ↔ Disease (via Patient_Disease junction table)
- Prescription ↔ Medicine (via Prescription_Medicine junction table)
- Patient ↔ Room (via Patient_Room allocation table)

**System Tables:**

- User_Login (authentication and authorization)
- Audit_Log (compliance and tracking)

---

# Appendix B: Database Metrics

- **Total Tables**: 20
- **Stored Procedures**: 10+
- **Triggers**: 10+

- **Materialized Views**: 10
- **Regular Views**: 4
- **Indexes**: 15+
- **Foreign Keys**: 25+
- **CHECK Constraints**: 20+
- **Unique Constraints**: 10+
- **Database Normalization Level**: 3NF (Third Normal Form)

---