**Hospital Resources and Room Utilization Database Design**

A PROJECT REPORT

*Submitted by*

## Mukesh A (RA2211056010008)

## Dhivya G (RA2211056010021)

## Riti MATANGI (RA2211056010035)

Under the Guidance of

## Dr. NANDHINI S

Assistant Professor, Department of Data Science and Business Systems

in partial fulfillment of the requirements for the degree of

# BACHELOR OF TECHNOLOGY

**in**

# COMPUTER SCIENCE AND ENGINEERING



**DEPARTMENT OF COMPUTING TECHNOLOGIES COLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR– 603 203**

##### MAY 2024



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603203**

**BONAFIDE CERTIFICATE**

Certified that **21CSC205P** project report titled “**Hospital Resource and Room Utilization Database Design**” is the bonafide work of “**Mukesh [RA22110 56010008], Dhivya G [RA2211056010021], Riti Matangi [RA2211056 010035],** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

|  |  |
| --- | --- |
| Faculty In Charge  Dr. NANDHINI S  Assistant Professor  Department of Data Science and Business Systems  SRMIST | Head of the Department  Dr. M. LAKSHMI  Professor and Head  Department of Data Science and Business Systems  SRMIST |

# ABSTRACT

The capstone project, “Hospital Resources and Room Utilization Management System” is a system designed to streamline the process of managing hospital resources and room utilization. The said project is critical especially now that we are facing a pandemic, there is a need for efficient management of hospital resources and room management. The management efficiency will prevent a shortage in supplies and overcrowding of patients in the hospitals. The researchers aim in developing an automated system to manage hospital resources and room utilization. The development of the system will ease up and simplify the process of determining the availability of hospital resources like medicines, oxygen tanks, and rooms that are used for COVID and non- COVID patients. The researchers will develop the system following the Software Development Life Cycle (SDLC) technique. The researchers will gather a sample size of participating hospitals, staff, and other target users to test and check the system. The system will also be evaluated by IT experts for further enhancement. The researchers will revise the system based on the recommendations of the experts. With the implementation of the system, the manual process of managing hospital resources and room utilization will be eliminated. Different hospitals can now deliver an efficient report on their capacity and availability to accommodate COVID and non- COVID patients. Hospitals can better serve and provide services to the public especially now that we are facing the COVID-19 pandemic.

Keywords: Hospital Resources, Room Utilization, Pandemic, COVID-19,

Efficiency, Automation, Testing, Expert Evaluation, Resource Allocation.

# TABLE OF CONTENTS

**ABSTRACT 3**

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Chapter Name** | **Page No** |
| **1.** | Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the  project | **5** |
| **2.** | Design of Relational Schemas, Creation of Database Tables for the project. | **7** |
| **3.** | Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors. | **8** |
| **4.** | Analyzing the pitfalls, identifying the dependencies, and applying normalizations. | **16** |
| **5.** | Implementation of concurrency control and recovery Mechanisms. | **27** |
| **6.** | Code for the project | **31** |
| **7.** | Result and Discussion (Screen shots of the implementation with front end. | **43** |
| **8.** | Attach the Real Time project certificate / Online course certificate | **50** |

**Chapter 1**

Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project.

A diagram of a company

Description automatically generated

**Problem Understanding**

The COVID-19 pandemic has highlighted the critical need for efficient management of hospital resources and room utilization to ensure optimal patient care while mitigating the risk of shortages and overcrowding. Current manual processes for resource allocation and room management are often cumbersome, leading to inefficiencies and potential errors. To address these challenges, there is an urgent requirement for a sophisticated, automated system that can accurately track and manage hospital resources and room availability in real-time.

This project seeks to develop a comprehensive Hospital Resource and Room Utilization Management System to address the complexities of pandemic response. The system will provide hospitals with the tools necessary to effectively manage essential resources such as medications, oxygen tanks, and patient rooms, facilitating proactive monitoring of availability and utilization. By implementing this system, hospitals can enhance their capacity planning, optimize resource allocation, and deliver timely and appropriate care to patients during the ongoing pandemic.

Key objectives of this project include:

1. Designing and implementing an automated system in accordance with the Software Development Life Cycle (SDLC) methodology, ensuring robust functionality and reliability.
2. Conducting rigorous testing and evaluation of the system with a diverse sample of participating hospitals, staff, and stakeholders to validate usability, accuracy, and effectiveness.
3. Soliciting feedback from IT experts to refine and enhance the system's features and usability, ensuring alignment with industry best practices.
4. Streamlining manual processes and enhancing reporting capabilities to enable hospitals to provide efficient and transparent services, particularly in the context of pandemic response.

**Chapter 2**

Design of Relational Schemas, Creation of Database Tables for the project.

A computer screen shot of a medical chart

Description automatically generated

**Chapter 3**

Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.

In our doctor appointment booking system project, we utilized various SQL concepts including constraints, set operations, joins, and views to ensure data integrity, perform complex queries, and simplify data retrieval. Here's how each of these concepts was applied:

QUERY 1: TABLE CREATION:

-- Drop Medicine table if exists DROP TABLE IF EXISTS Medicine;

-- Create Medicine table CREATE TABLE Medicine (

MedicineID INT PRIMARY KEY,

MedicineName VARCHAR(100), TankID INT,

ǪuantityAvailable INT, Capacity INT,

RefillOptions VARCHAR(255), ValveType CHAR(1)

);

-- Insert data into Medicine table

INSERT INTO Medicine (MedicineID, MedicineName, TankID, ǪuantityAvailable, Capacity, RefillOptions, ValveType)

VALUES

(1, 'Aspirin', 1, 1000, 2000, 'Auto-refill', 'A'),

(2, 'Paracetamol', 2, 800, 1500, 'Manual-refill', 'B'),

(3, 'Ibuprofen', 1, 1200, 1800, 'Auto-refill', 'A'),

(4, 'Amoxicillin', 3, 600, 1000, 'Manual-refill', 'C'),

(5, 'Ciprofloxacin', 3, 700, 1200, 'Auto-refill', 'B'),

(6, 'Loratadine', 1, 500, 800, 'Auto-refill', 'A'),

(7, 'Diazepam', 4, 300, 500, 'Manual-refill', 'B'),

(8, 'Omeprazole', 2, 900, 1400, 'Auto-refill', 'C'),

(9, 'Ranitidine', 3, 750, 1100, 'Manual-refill', 'A'),

(10, 'Levothyroxine', 4, 400, 600, 'Auto-refill', 'C');

A screenshot of a computer

Description automatically generated

A screenshot of a medical report

Description automatically generated

QUERY 2: SELECT MEDICINE NAMES AND QUANTITIES AVAILABLE FOR MEDICINES WITH A REFILL OPTION OF 'AUTO-REFILL'

SELECT MedicineName, ǪuantityAvailable FROM Medicine WHERE RefillOptions = 'Auto-refill';

A screenshot of a computer

Description automatically generated

ǪUERY 3: SELECT MEDICINE NAMES AND TANK IDS WHERE THE QUANTITY AVAILABLE IS LESS THAN 1000

SELECT MedicineName, TankID FROM Medicine WHERE ǪuantityAvailable < 1000;

A screenshot of a computer

Description automatically generated

ǪUERY 4: SELECT THE TOTAL QUANTITY AVAILABLE FOR ALL MEDICINES

SELECT SUM(ǪuantityAvailable) AS TotalǪuantity FROM Medicine;

A screenshot of a computer

Description automatically generated

ǪUERY 5: SELECT MEDICINE NAMES AND QUANTITIES AVAILABLE IN DESCENDING ORDER OF QUANTITY AVAILABLE

SELECT MedicineName, ǪuantityAvailable FROM Medicine ORDER BY ǪuantityAvailable DESC;

A screenshot of a computer

Description automatically generated

ǪUERY 6: UPDATE THE QUANTITY AVAILABLE FOR A SPECIFIC MEDICINE (E.G., ASPIRIN) TO A NEW VALUE (E.G., 1500)

UPDATE Medicine SET ǪuantityAvailable = 1500 WHERE MedicineName = 'Aspirin';

A screenshot of a computer

Description automatically generated

ǪUERY 7: DELETE A SPECIFIC MEDICINE (E.G., IBUPROFEN) FROM THE MEDICINE TABLE

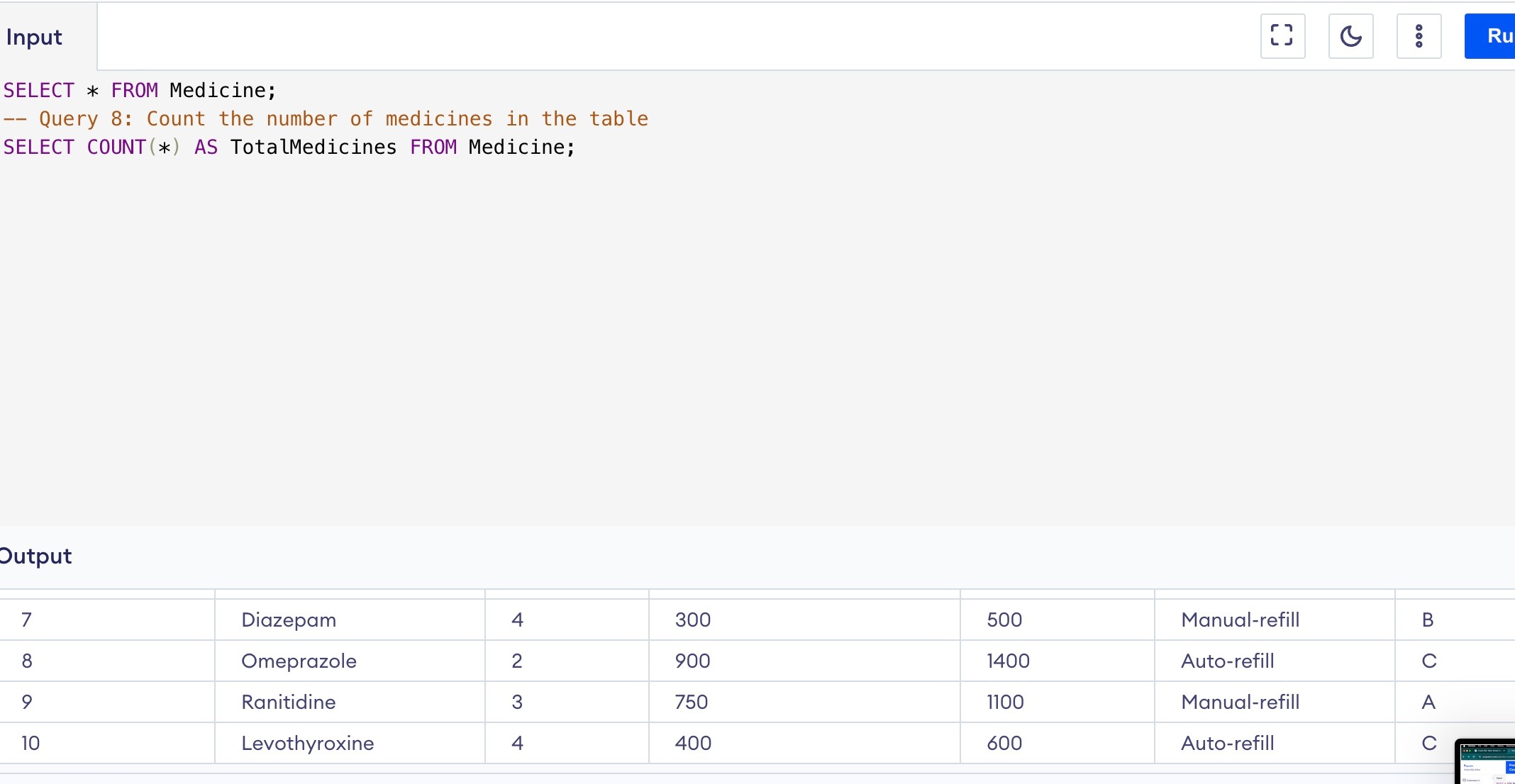
DELETE FROM Medicine WHERE MedicineName = 'Ibuprofen';

A screenshot of a computer

Description automatically generated

ǪUERY 8: COUNT THE NUMBER OF MEDICINES IN THE TABLE

SELECT COUNT(\*) AS TotalMedicines FROM Medicine;



ǪUERY 9: SELECT MEDICINE NAMES AND TANK IDS WHERE THE

MEDICINE NAME STARTS WITH 'A'

SELECT MedicineName, TankID FROM Medicine WHERE MedicineName LIKE 'A%';

A screenshot of a computer

Description automatically generated

ǪUERY 10: SELECT DISTINCT REFILL OPTIONS AVAILABLE IN THE TABLE

SELECT DISTINCT RefillOptions FROM Medicine;

A screenshot of a computer

Description automatically generated

ǪUERY 11: SELECT DISTINCT REFILL OPTIONS AVAILABLE IN THE TABLE

A screenshot of a computer

Description automatically generatedSELECT \* FROM Medicine WHERE ǪuantityAvailable = (SELECT MAX(ǪuantityAvailable) FROM Medicine);

ǪUERY 12: ADD A NEW MEDICINE TO THE TABLE

INSERT INTO Medicine (MedicineID, MedicineName, TankID, ǪuantityAvailable, Capacity, RefillOptions, ValveType)

VALUES (21, 'Dexamethasone', 1, 800, 1200, 'Auto-refill', 'A');

A screenshot of a computer

Description automatically generated

**Chapter 4**

Analyzing the pitfalls, identifying the dependencies, and applying normalizations.

##### First Normal Form (1NF):

The First Normal Form (1NF) is the initial step in database normalization, which aims to organize data efficiently and reduce redundancy.

1. **Atomic Values**: 1NF requires that each attribute in a relation (table) contains only atomic values. Atomic values are indivisible; they cannot be broken down further. This means that each cell in the table should contain a single value, not a set of values or a list.
2. **No Repeating Groups**: 1NF prohibits repeating groups within a table. Each column should contain only a single value, and there should be no repeating sets of columns that represent multiple values for the same attribute.
3. **Homogeneous Data Types**: All values within a column must be of the same data type. This ensures consistency and facilitates operations such as sorting and comparison.
4. **Each Row Unique**: Each row in the table must be unique. This means that there must be a primary key or a combination of columns that uniquely identifies each row.
5. **Elimination of Composite Attributes**: Composite attributes, which contain multiple data elements within a single column, are not allowed in 1NF. Each attribute should have a single value associated with it.

**Medicine Table:**



##### Second Normal Form (2NF):

##### 1NF Compliant: Before achieving 2NF, the table must be in the first normal form (1NF). This means each attribute contains atomic values, there are no repeating groups, and all data types are consistent within columns.

##### Removal of Partial Dependencies: 2NF eliminates partial dependencies within a table. A partial dependency occurs when a non-prime attribute is functionally dependent on only a part of the primary key. In other words, an attribute depends on only a portion of the primary key, not the entire primary key.

##### Primary Key and Candidate Keys: In 2NF, each non-prime attribute must be fully functionally dependent on the entire primary key. If an attribute is functionally dependent on only part of the primary key, it should be moved to a separate table.

##### Normalization into Separate Tables: To achieve 2NF, tables are often normalized by splitting them into multiple tables to eliminate partial dependencies. Each table typically represents a single entity or concept, and the relationships between tables are defined through primary and foreign keys.

##### Ensuring Data Integrity: 2NF ensures data integrity by organizing data into smaller, more manageable tables, reducing redundancy, and minimizing the risk of anomalies such as update, insert, and delete anomalies.

##### Improving Data Structure: By removing partial dependencies, 2NF improves the structure of the database, making it easier to understand, maintain, and query.

##### Dependency on Entire Primary Key: In a 2NF-compliant table, every non-prime attribute should be functionally dependent on the entire primary key, meaning changes in any part of the primary key should not affect these attributes.

##### To achieve the Second Normal Form (2NF), we need to ensure that the table is already in 1NF and then eliminate any partial dependencies.

##### In the given Medicine table:

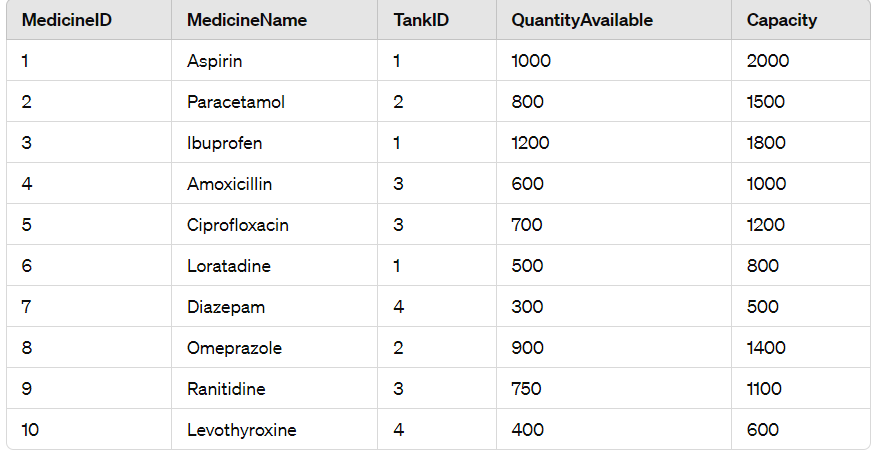
##### 

##### The primary key is MedicineID:

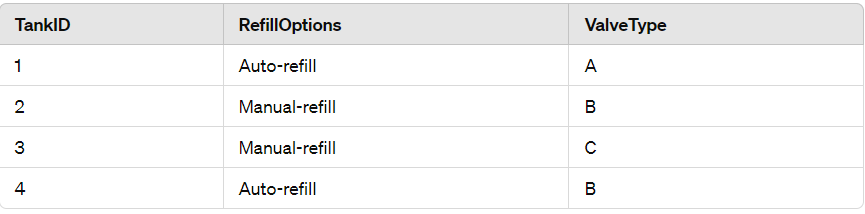
To achieve 2NF, we need to make sure there are no partial dependencies. In this case, we can see that the attributes RefillOptions and ValveType are dependent only on TankID, not on the entire primary key (MedicineID).

To normalize to 2NF, we divide the table into two tables: one for Medicine details and one for Tank details.

**(2NF) Medicine Table:**



**Tank Table:**

****

The Medicine table's attributes are fully dependent on the MedicineID (the primary key), and the Tank table's attributes are fully dependent on TankID (the primary key).

##### Third Normal Form (3NF):

The Third Normal Form (3NF) extends the principles of 1NF and 2NF, aiming to eliminate transitive dependencies within a relational database.

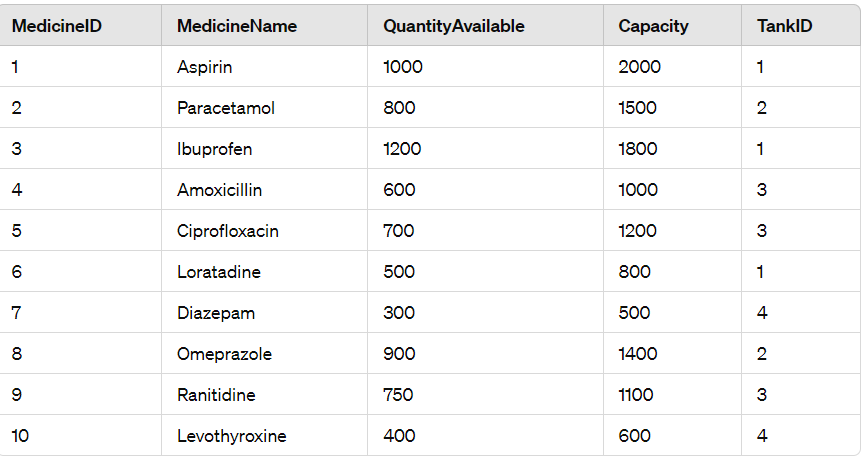
1. **1NF and 2NF Compliant**: Before achieving 3NF, the table must conform to the first two normal forms (1NF and 2NF). This means each attribute contains atomic values, there are no repeating groups, and partial dependencies have been eliminated.
2. **Elimination of Transitive Dependencies**: 3NF eliminates transitive dependencies, where a non-prime attribute depends on another non-prime attribute, rather than directly on the primary key. To achieve this, any non-prime attribute that is functionally dependent on another non-prime attribute must be moved to its own table.
3. **Each Non-Key Attribute Directly Dependent on the Primary Key**: In 3NF, each non-key attribute should be directly dependent on the primary key. This means that every non-key attribute must be dependent on the primary key and not on any other non-key attributes.
4. **Normalization into Separate Tables**: Just like in 2NF, achieving 3NF often involves breaking down tables into smaller, more manageable tables to eliminate transitive dependencies. Each table should represent a single entity or concept.
5. **Maintaining Data Integrity**: By eliminating transitive dependencies, 3NF helps to maintain data integrity, reducing redundancy and the risk of anomalies such as update, insert, and delete anomalies.
6. **Improved Data Structure**: 3NF further improves the structure of the database by ensuring that relationships between attributes are well-defined and that data is organized efficiently.
7. **Avoiding Redundancy**: 3NF helps to minimize redundancy by ensuring that each piece of data is stored in only one place, reducing the chance of inconsistencies and making the database more efficient.

**The functional dependencies:**

* MedicineID -> MedicineName, QuantityAvailable, Capacity.
* TankID -> RefillOptions, ValveType.

we need to split the table into two to remove the transitive dependency of RefillOptions and ValveType on TankID.

**(3NF) Medicine Table:**



**Tank Table:**



The Medicine table's attributes are directly dependent on MedicineID, and the Tank table's attributes are directly dependent on TankID. There are no transitive dependencies, and the tables are normalized and efficient for data storage and management.

##### Boyce-Codd Normal Form (BCNF):

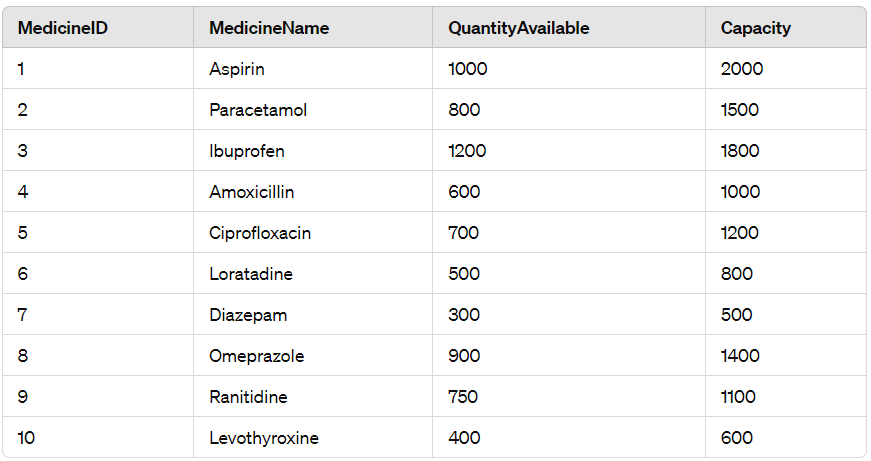
1. **Definition**: BCNF is a property of a relation in a relational database. A relation is in BCNF if and only if for every non-trivial functional dependency (X → Y), X is a superkey. In simpler terms, every determinant (the attribute on the left-hand side of a functional dependency) must be a candidate key.
2. **Elimination of Non-Trivial Dependencies**: BCNF eliminates non-trivial functional dependencies where the determinant is not a superkey. This ensures that each attribute in the table is functionally dependent only on the primary key of the table.
3. **Decomposition**: To achieve BCNF, a table may need to be decomposed into multiple tables to ensure that each table satisfies the BCNF property. This decomposition ensures that dependencies are maintained while removing anomalies.
4. **Candidate Keys and Superkeys**: In BCNF, every determinant must be a superkey, which means it uniquely identifies all other attributes in the table. A candidate key is a minimal superkey, meaning removing any attribute from it would cause it to lose its uniqueness.
5. **Data Integrity**: BCNF helps maintain data integrity by ensuring that each attribute is fully functionally dependent on the primary key and that there are no non-trivial dependencies that could lead to anomalies such as insertion, deletion, or update anomalies.
6. **Normalization**: BCNF is a higher level of normalization compared to 3NF. While 3NF removes transitive dependencies, BCNF goes a step further by ensuring that all functional dependencies are directly related to the candidate keys.
7. **Efficiency**: While BCNF provides better data integrity, achieving it may result in more tables and joins, which could affect query performance and database efficiency. However, it ensures that data anomalies are minimized.

Identify the candidate keys:

* {MedicineID}
* {TankID}

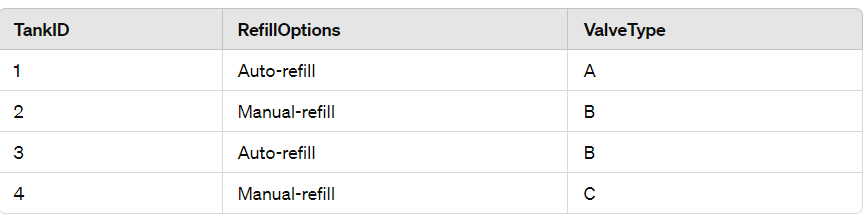
**(BCNF) Medicine Table:**

MedicineID -> MedicineName, QuantityAvailable, Capacity

****

**TankDetails Table:**

TankID -> RefillOptions, ValveType



##### Fourth Normal Form (4NF):

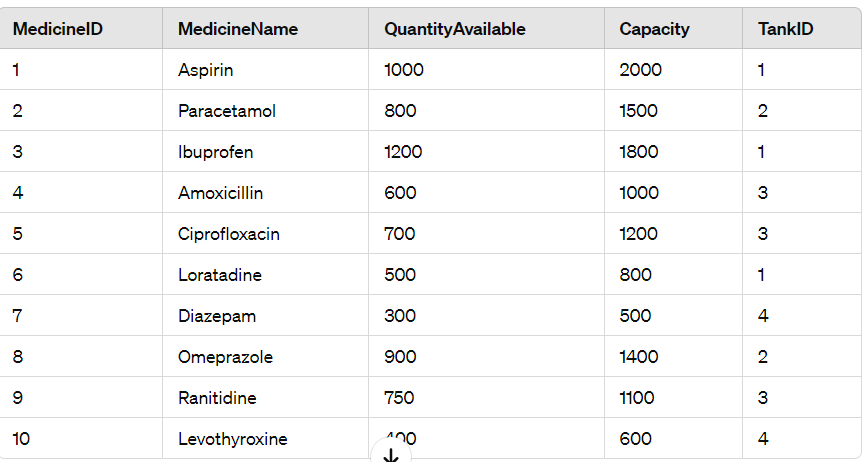
1. Definition: A relation is in 4NF if it is in 3NF and has no non-trivial multi-valued dependencies. In simpler terms, it means that there are no dependencies between non-key attributes.
2. Multi-Valued Dependencies (MVDs): A multi-valued dependency occurs when two or more non-key attributes are dependent on each other, rather than being directly dependent on the primary key. For example, consider a relation R(A, B, C), where B is dependent on A, and C is dependent on A, but B and C are not dependent on each other. This is a multi-valued dependency.
3. Elimination of MVDs: In 4NF, we eliminate MVDs by decomposing the relation into smaller, more atomic relations. Each relation should have no MVDs and should be functionally dependent on the primary key.
4. Decomposition: To achieve 4NF, a relation may need to be decomposed into multiple relations, each addressing a specific multi-valued dependency. This decomposition ensures that the database maintains data integrity and eliminates redundancy.
5. Candidate Keys: In 4NF, the primary key (or candidate key) uniquely identifies each tuple in the relation. Non-key attributes should be fully functionally dependent on the primary key.
6. Benefits: Achieving 4NF reduces redundancy and the risk of anomalies such as insertion, deletion, and update anomalies. It helps to maintain data integrity by organizing the data more efficiently.
7. Efficiency: While 4NF ensures data integrity, achieving it may result in more tables and joins, which could affect query performance and database efficiency. However, it ensures that data anomalies are minimized.
8. Normalization Process: The process of achieving 4NF involves ensuring that each non-key attribute is fully functionally dependent on the primary key and that there are no non-trivial multi-valued dependencies. This may require decomposing the relation into smaller relations until 4NF is satisfied.

**Given Medicine Table:**



Each TankID can have multiple RefillOptions and ValveTypes associated with it.

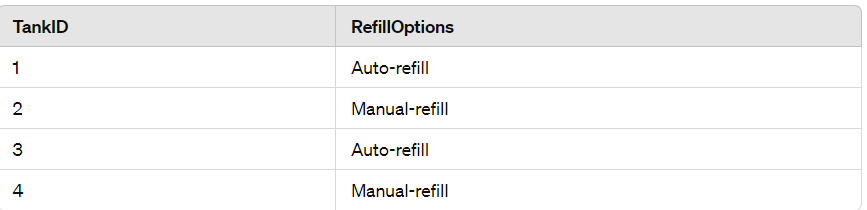
**(4NF) Medicine Table:**

****

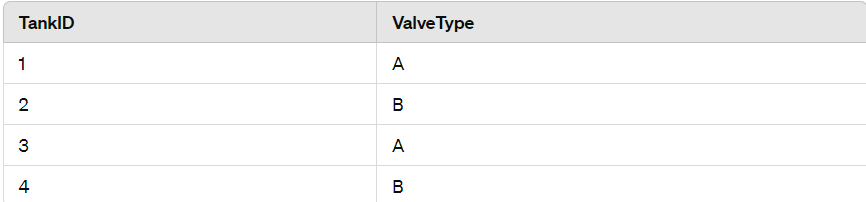
**Tank Table:**

****

**RefillOptions Table:**

****

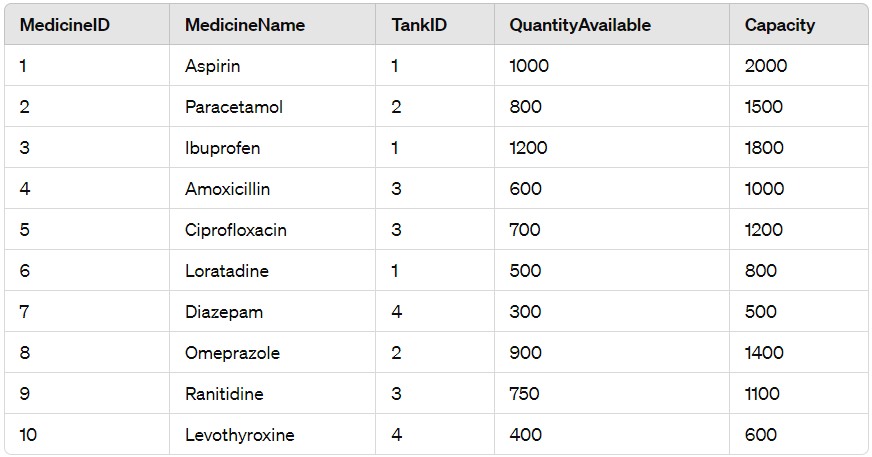
**ValveType Table:**

****

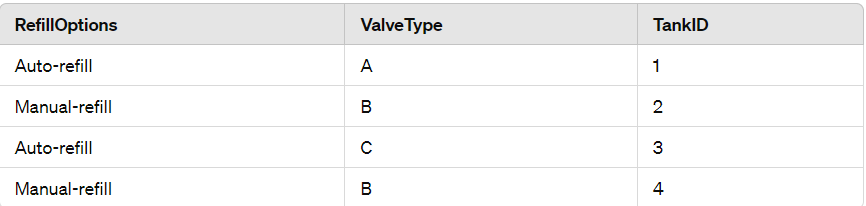
##### Fifth Normal Form (5NF):

##### Fifth Normal Form (5NF), also known as Project-join Normal Form (PJNF), addresses cases where there are join dependencies that cannot be handled by earlier normal forms. It focuses on ensuring that all join dependencies are eliminated.

**(5NF Medicine Table):**

****

**RefillOptions-ValveType Table:**

****

The RefillOptions-ValveType table maintains the join dependency between RefillOptions and ValveType, along with the corresponding TankID.

**Chapter 5**

Implementation of concurrency control and recovery mechanisms.

Code:

**-- Create Medicine table**

**CREATE TABLE IF NOT EXISTS Medicine (**

**MedicineID INTEGER PRIMARY KEY,**

**MedicineName TEXT,**

**QuantityAvailable INTEGER,**

**Capacity INTEGER,**

**TankID INTEGER**

**);**

**-- Create TankDetails table**

**CREATE TABLE IF NOT EXISTS TankDetails (**

**TankID INTEGER PRIMARY KEY,**

**RefillOptions TEXT,**

**ValveType TEXT,**

**UNIQUE(TankID, RefillOptions, ValveType)**

**);**

**-- Insert values into Medicine table**

**INSERT INTO Medicine (MedicineID, MedicineName, QuantityAvailable, Capacity, TankID)**

**VALUES**

**(1, 'Aspirin', 1000, 2000, 1),**

**(2, 'Paracetamol', 800, 1500, 2),**

**(3, 'Ibuprofen', 1200, 1800, 1),**

**(4, 'Amoxicillin', 600, 1000, 3),**

**(5, 'Ciprofloxacin', 700, 1200, 3),**

**(6, 'Loratadine', 500, 800, 1),**

**(7, 'Diazepam', 300, 500, 4),**

**(8, 'Omeprazole', 900, 1400, 2),**

**(9, 'Ranitidine', 750, 1100, 3),**

**(10, 'Levothyroxine', 400, 600, 4);**

**-- Insert values into TankDetails table**

**INSERT INTO TankDetails (TankID, RefillOptions, ValveType)**

**VALUES**

**(1, 'Auto-refill', 'A'),**

**(2, 'Manual-refill', 'B'),**

**(3, 'Auto-refill', 'A'),**

**(4, 'Manual-refill', 'B');**

**-- Update medicine quantity available with concurrency control**

**BEGIN TRANSACTION;**

**UPDATE Medicine**

**SET QuantityAvailable = 950**

**WHERE MedicineID = 1;**

**COMMIT;**

**-- Update tank details with concurrency control**

**BEGIN TRANSACTION;**

**UPDATE TankDetails**

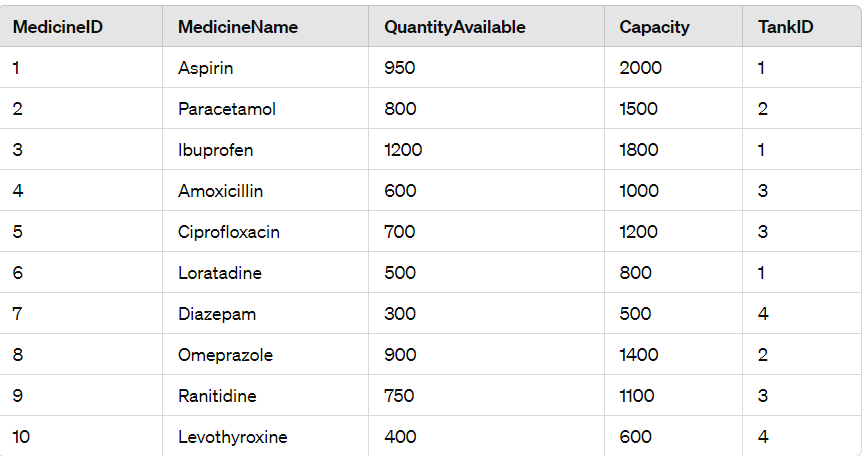
**SET RefillOptions = 'Manual-refill', ValveType = 'B'**

**WHERE TankID = 1;**

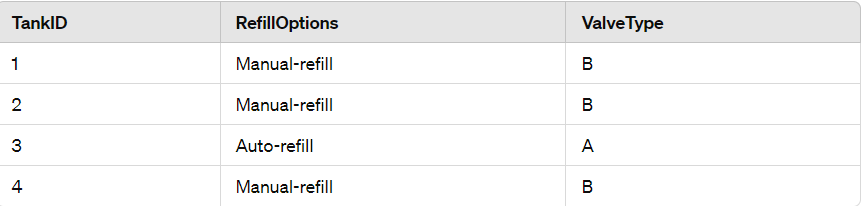
**COMMIT;**

UPDATED TABLE FOR CONCURRENCY CONTROL:

**Medicine Table:**



**TankDetails Table:**

****

These tables represent the current state of the database after implementing concurrency control. The quantity of Aspirin has been updated to 950 units, and Tank 1 has been changed to manual-refill with valve type B.

**Concurrency Control:**

Concurrency control is a fundamental aspect of database management systems (DBMS) that ensures data integrity and consistency in multi-user environments where multiple transactions may access or modify the same data concurrently. Concurrency control mechanisms prevent anomalies such as lost updates, dirty reads, and non-repeatable reads. Here are some key points about concurrency control:

1. **Isolation**: Concurrency control ensures that transactions execute as if they were isolated from each other, even though they may be executed concurrently. Each transaction appears to execute in isolation, maintaining the illusion of serial execution.
2. **Serializability**: Transactions should produce the same result as if they were executed serially, preserving consistency and correctness. This is achieved through techniques such as locking, timestamp-based ordering, or optimistic concurrency control.
3. **Locking Mechanisms**: Locking is one of the most common concurrency control mechanisms. Transactions acquire locks on data items to prevent other transactions from accessing or modifying them concurrently. Locks can be exclusive (write locks) or shared (read locks), and they are released when the transaction completes.
4. **Timestamp-based Concurrency Control**: Transactions are assigned unique timestamps, and conflicts between transactions are resolved based on these timestamps. Older transactions may take precedence over newer ones to maintain serializability.
5. **Optimistic Concurrency Control (OCC)**: This approach allows transactions to proceed without locking resources initially. Before committing, the DBMS checks for conflicts. If conflicts are detected, the transaction is aborted and restarted.

**Recovery Mechanisms:**

Recovery mechanisms ensure that the database remains in a consistent state even after system failures, crashes, or errors. These mechanisms are crucial for data durability and maintainability. Here are the key components of recovery mechanisms:

1. **Logging**: Logging is the process of recording changes made to the database in a log file before they are applied to the actual database. This includes recording transactional changes, such as updates, inserts, and deletes, as well as checkpoint information.
2. **Write-Ahead Logging (WAL)**: In WAL, changes are first written to the log file before being applied to the database. This ensures that the changes are durable even if the system crashes before they are written to disk.
3. **Checkpointing**: Periodically, the DBMS performs a checkpoint to write all modified data pages from memory to disk and record information about the checkpoint in the log. Checkpoints help speed up recovery by reducing the amount of log replay required after a crash.
4. **Rollback and Rollforward**: Rollback involves undoing the changes made by an incomplete transaction, restoring the database to its previous state. Rollforward involves applying changes from the log to recover the database to a consistent state after a crash.
5. **Transaction Undo and Redo**: The undo phase of recovery undoes the changes made by transactions that were not committed at the time of failure. The redo phase reapplies changes made by committed transactions from the log to ensure consistency.

**Chapter 6**

Code for the project.

**Sample code:**

-- phpMyAdmin SQL Dump

-- version 5.0.2

-- https://www.phpmyadmin.net/

--

-- Host: 127.0.0.1

-- Generation Time: Jul 05, 2020 at 04:35 AM

-- Server version: 10.4.13-MariaDB

-- PHP Version: 7.4.7

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8mb4 \*/;

--

-- Database: `hms`

--

-- --------------------------------------------------------

--

-- Table structure for table `admin`

--

CREATE TABLE `admin` (

  `id` int(11) NOT NULL,

  `username` varchar(255) NOT NULL,

  `password` varchar(255) NOT NULL,

  `updationDate` varchar(255) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `admin`

--

INSERT INTO `admin` (`id`, `username`, `password`, `updationDate`) VALUES

(1, 'admin', 'Test@12345', '28-12-2016 11:42:05 AM');

-- --------------------------------------------------------

--

-- Table structure for table `appointment`

--

CREATE TABLE `appointment` (

  `id` int(11) NOT NULL,

  `doctorSpecialization` varchar(255) DEFAULT NULL,

  `doctorId` int(11) DEFAULT NULL,

  `userId` int(11) DEFAULT NULL,

  `consultancyFees` int(11) DEFAULT NULL,

  `appointmentDate` varchar(255) DEFAULT NULL,

  `appointmentTime` varchar(255) DEFAULT NULL,

  `postingDate` timestamp NULL DEFAULT current\_timestamp(),

  `userStatus` int(11) DEFAULT NULL,

  `doctorStatus` int(11) DEFAULT NULL,

  `updationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `appointment`

--

INSERT INTO `appointment` (`id`, `doctorSpecialization`, `doctorId`, `userId`, `consultancyFees`, `appointmentDate`, `appointmentTime`, `postingDate`, `userStatus`, `doctorStatus`, `updationDate`) VALUES

(3, 'Demo test', 7, 6, 600, '2019-06-29', '9:15 AM', '2019-06-23 18:31:28', 1, 0, '0000-00-00 00:00:00'),

(4, 'Ayurveda', 5, 5, 8050, '2019-11-08', '1:00 PM', '2019-11-05 10:28:54', 1, 1, '0000-00-00 00:00:00'),

(5, 'Dermatologist', 9, 7, 500, '2019-11-30', '5:30 PM', '2019-11-10 18:41:34', 1, 0, '2019-11-10 18:48:30'),

(6, 'Physician', 11, 2, 2000, '2020-07-14', '10:15 AM', '2020-07-05 02:12:37', 1, 1, NULL),

(7, 'General Physician', 3, 2, 1200, '2020-07-05', '10:15 AM', '2020-07-05 02:14:49', 1, 1, NULL);

-- --------------------------------------------------------

--

-- Table structure for table `doctors`

--

CREATE TABLE `doctors` (

  `id` int(11) NOT NULL,

  `specilization` varchar(255) DEFAULT NULL,

  `doctorName` varchar(255) DEFAULT NULL,

  `address` longtext DEFAULT NULL,

  `docFees` varchar(255) DEFAULT NULL,

  `contactno` bigint(11) DEFAULT NULL,

  `docEmail` varchar(255) DEFAULT NULL,

  `password` varchar(255) DEFAULT NULL,

  `creationDate` timestamp NULL DEFAULT current\_timestamp(),

  `updationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `doctors`

--

INSERT INTO `doctors` (`id`, `specilization`, `doctorName`, `address`, `docFees`, `contactno`, `docEmail`, `password`, `creationDate`, `updationDate`) VALUES

(1, 'Dentist', 'Lyndon Bermoy', 'New Delhi', '500', 8285703354, 'anuj.lpu1@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2016-12-29 06:25:37', '2020-07-05 01:53:19'),

(2, 'Homeopath', 'Sarita Pandey', 'Varanasi', '600', 2147483647, 'sarita@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2016-12-29 06:51:51', '0000-00-00 00:00:00'),

(3, 'General Physician', 'Nitesh Kumar', 'Ghaziabad', '1200', 8523699999, 'nitesh@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2017-01-07 07:43:35', '0000-00-00 00:00:00'),

(4, 'Homeopath', 'Vijay Verma', 'New Delhi', '700', 25668888, 'vijay@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2017-01-07 07:45:09', '0000-00-00 00:00:00'),

(5, 'Ayurveda', 'Sanjeev', 'Gurugram', '8050', 442166644646, 'sanjeev@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2017-01-07 07:47:07', '0000-00-00 00:00:00'),

(6, 'General Physician', 'Amrita', 'New Delhi India', '2500', 45497964, 'amrita@test.com', 'f925916e2754e5e03f75dd58a5733251', '2017-01-07 07:52:50', '0000-00-00 00:00:00'),

(7, 'Demo test', 'abc ', 'New Delhi India', '200', 852888888, 'test@demo.com', 'f925916e2754e5e03f75dd58a5733251', '2017-01-07 08:08:58', '2019-06-23 18:17:25'),

(8, 'Ayurveda', 'Test Doctor', 'Xyz Abc New Delhi', '600', 1234567890, 'test@test.com', '202cb962ac59075b964b07152d234b70', '2019-06-23 17:57:43', '2019-06-23 18:06:06'),

(11, 'Physician', 'Jonah Juarez', 'Surigao Philippines', '2000', 123456789, 'jjuarez@gmail.com', '25f9e794323b453885f5181f1b624d0b', '2020-07-05 02:06:00', '2020-07-05 02:06:48');

-- --------------------------------------------------------

--

-- Table structure for table `doctorslog`

--

CREATE TABLE `doctorslog` (

  `id` int(11) NOT NULL,

  `uid` int(11) DEFAULT NULL,

  `username` varchar(255) DEFAULT NULL,

  `userip` binary(16) DEFAULT NULL,

  `loginTime` timestamp NULL DEFAULT current\_timestamp(),

  `logout` varchar(255) DEFAULT NULL,

  `status` int(11) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `doctorslog`

--

INSERT INTO `doctorslog` (`id`, `uid`, `username`, `userip`, `loginTime`, `logout`, `status`) VALUES

(20, 7, 'test@demo.com', 0x3a3a3100000000000000000000000000, '2020-07-05 01:50:01', NULL, 1),

(21, NULL, 'juarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:02:51', NULL, 0),

(22, NULL, 'juarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:03:03', NULL, 0),

(23, NULL, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:04:02', NULL, 0),

(24, NULL, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:04:38', NULL, 0),

(25, 11, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:06:19', NULL, 1),

(26, 11, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:06:38', NULL, 1),

(27, 11, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:08:18', NULL, 1),

(28, 11, 'jjuarez@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:15:25', NULL, 1);

-- --------------------------------------------------------

--

-- Table structure for table `doctorspecilization`

--

CREATE TABLE `doctorspecilization` (

  `id` int(11) NOT NULL,

  `specilization` varchar(255) DEFAULT NULL,

  `creationDate` timestamp NULL DEFAULT current\_timestamp(),

  `updationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `doctorspecilization`

--

INSERT INTO `doctorspecilization` (`id`, `specilization`, `creationDate`, `updationDate`) VALUES

(1, 'Gynecologist/Obstetrician', '2016-12-28 06:37:25', '0000-00-00 00:00:00'),

(2, 'General Physician', '2016-12-28 06:38:12', '0000-00-00 00:00:00'),

(3, 'Dermatologist', '2016-12-28 06:38:48', '0000-00-00 00:00:00'),

(4, 'Homeopath', '2016-12-28 06:39:26', '0000-00-00 00:00:00'),

(5, 'Ayurveda', '2016-12-28 06:39:51', '0000-00-00 00:00:00'),

(6, 'Dentist', '2016-12-28 06:40:08', '0000-00-00 00:00:00'),

(7, 'Ear-Nose-Throat (Ent) Specialist', '2016-12-28 06:41:18', '0000-00-00 00:00:00'),

(9, 'Demo test', '2016-12-28 07:37:39', '0000-00-00 00:00:00'),

(10, 'Bones Specialist demo', '2017-01-07 08:07:53', '0000-00-00 00:00:00'),

(11, 'Test', '2019-06-23 17:51:06', '2019-06-23 17:55:06'),

(12, 'Dermatologist', '2019-11-10 18:36:36', '2019-11-10 18:36:50'),

(13, 'Physician', '2020-07-05 01:59:00', NULL);

-- --------------------------------------------------------

--

-- Table structure for table `tblcontactus`

--

CREATE TABLE `tblcontactus` (

  `id` int(11) NOT NULL,

  `fullname` varchar(255) DEFAULT NULL,

  `email` varchar(255) DEFAULT NULL,

  `contactno` bigint(12) DEFAULT NULL,

  `message` mediumtext DEFAULT NULL,

  `PostingDate` timestamp NULL DEFAULT current\_timestamp(),

  `AdminRemark` mediumtext DEFAULT NULL,

  `LastupdationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp(),

  `IsRead` int(11) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `tblcontactus`

--

INSERT INTO `tblcontactus` (`id`, `fullname`, `email`, `contactno`, `message`, `PostingDate`, `AdminRemark`, `LastupdationDate`, `IsRead`) VALUES

(1, 'test user', 'test@gmail.com', 2523523522523523, ' This is sample text for the test.', '2019-06-29 19:03:08', 'Test Admin Remark', '2019-06-30 12:55:23', 1),

(2, 'Lyndon Bermoy', 'serbermz2020@gmail.com', 1111111111111111, ' This is sample text for testing.  This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing. This is sample text for testing.', '2019-06-30 13:06:50', 'Answered', '2020-07-05 02:13:25', 1),

(3, 'fdsfsdf', 'fsdfsd@ghashhgs.com', 3264826346, 'sample text   sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  sample text  ', '2019-11-10 18:53:48', 'vfdsfgfd', '2019-11-10 18:54:04', 1),

(4, 'demo', 'demo@gmail.com', 123456789, ' hi, this is a demo', '2020-07-05 01:57:20', 'answered', '2020-07-05 01:57:46', 1);

-- --------------------------------------------------------

--

-- Table structure for table `tblmedicalhistory`

--

CREATE TABLE `tblmedicalhistory` (

  `ID` int(10) NOT NULL,

  `PatientID` int(10) DEFAULT NULL,

  `BloodPressure` varchar(200) DEFAULT NULL,

  `BloodSugar` varchar(200) NOT NULL,

  `Weight` varchar(100) DEFAULT NULL,

  `Temperature` varchar(200) DEFAULT NULL,

  `MedicalPres` mediumtext DEFAULT NULL,

  `CreationDate` timestamp NOT NULL DEFAULT current\_timestamp() ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `tblmedicalhistory`

--

INSERT INTO `tblmedicalhistory` (`ID`, `PatientID`, `BloodPressure`, `BloodSugar`, `Weight`, `Temperature`, `MedicalPres`, `CreationDate`) VALUES

(2, 3, '120/185', '80/120', '85 Kg', '101 degree', '#Fever, #BP high\r\n1.Paracetamol\r\n2.jocib tab\r\n', '2019-11-06 04:20:07'),

(3, 2, '90/120', '92/190', '86 kg', '99 deg', '#Sugar High\r\n1.Petz 30', '2019-11-06 04:31:24'),

(4, 1, '125/200', '86/120', '56 kg', '98 deg', '# blood pressure is high\r\n1.koil cipla', '2019-11-06 04:52:42'),

(5, 1, '96/120', '98/120', '57 kg', '102 deg', '#Viral\r\n1.gjgjh-1Ml\r\n2.kjhuiy-2M', '2019-11-06 04:56:55'),

(6, 4, '90/120', '120', '56', '98 F', '#blood sugar high\r\n#Asthma problem', '2019-11-06 14:38:33'),

(7, 5, '80/120', '120', '85', '98.6', 'Rx\r\n\r\nAbc tab\r\nxyz Syrup', '2019-11-10 18:50:23');

-- --------------------------------------------------------

--

-- Table structure for table `tblpatient`

--

CREATE TABLE `tblpatient` (

  `ID` int(10) NOT NULL,

  `Docid` int(10) DEFAULT NULL,

  `PatientName` varchar(200) DEFAULT NULL,

  `PatientContno` bigint(10) DEFAULT NULL,

  `PatientEmail` varchar(200) DEFAULT NULL,

  `PatientGender` varchar(50) DEFAULT NULL,

  `PatientAdd` mediumtext DEFAULT NULL,

  `PatientAge` int(10) DEFAULT NULL,

  `PatientMedhis` mediumtext DEFAULT NULL,

  `CreationDate` timestamp NULL DEFAULT current\_timestamp(),

  `UpdationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `tblpatient`

--

INSERT INTO `tblpatient` (`ID`, `Docid`, `PatientName`, `PatientContno`, `PatientEmail`, `PatientGender`, `PatientAdd`, `PatientAge`, `PatientMedhis`, `CreationDate`, `UpdationDate`) VALUES

(1, 1, 'Manisha Jha', 4558968789, 'test@gmail.com', 'Female', '\"\"J&K Block J-127, Laxmi Nagar New Delhi', 26, 'She is diabetic patient', '2019-11-04 21:38:06', '2019-11-06 06:48:05'),

(4, 7, 'Manav Sharma', 9888988989, 'sharma@gmail.com', 'Male', 'L-56,Ashok Nagar New Delhi-110096', 45, 'He is long suffered by asthma', '2019-11-06 14:33:54', '2019-11-06 14:34:31'),

(5, 9, 'John', 1234567890, 'john@test.com', 'male', 'Test ', 25, 'THis is sample text for testing.', '2019-11-10 18:49:24', NULL),

(6, 0, 'Don Bermoy', 123456789, 'serbermz2020@gmail.com', 'male', 'Surigao Philippines', 35, 'Diagnosed of High Blood Pressure', '2020-07-05 02:08:09', NULL);

-- --------------------------------------------------------

--

-- Table structure for table `userlog`

--

CREATE TABLE `userlog` (

  `id` int(11) NOT NULL,

  `uid` int(11) DEFAULT NULL,

  `username` varchar(255) DEFAULT NULL,

  `userip` binary(16) DEFAULT NULL,

  `loginTime` timestamp NULL DEFAULT current\_timestamp(),

  `logout` varchar(255) DEFAULT NULL,

  `status` int(11) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `userlog`

--

INSERT INTO `userlog` (`id`, `uid`, `username`, `userip`, `loginTime`, `logout`, `status`) VALUES

(24, 2, 'test@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 01:50:24', NULL, 1),

(25, NULL, 'serbermz2020@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:09:18', NULL, 0),

(26, NULL, 'serbermz2020@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:11:05', NULL, 0),

(27, NULL, 'test@demo.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:11:24', NULL, 0),

(28, NULL, 'serbermz2020@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:11:46', NULL, 0),

(29, 2, 'test@gmail.com', 0x3a3a3100000000000000000000000000, '2020-07-05 02:12:00', NULL, 1);

-- --------------------------------------------------------

--

-- Table structure for table `users`

--

CREATE TABLE `users` (

  `id` int(11) NOT NULL,

  `fullName` varchar(255) DEFAULT NULL,

  `address` longtext DEFAULT NULL,

  `city` varchar(255) DEFAULT NULL,

  `gender` varchar(255) DEFAULT NULL,

  `email` varchar(255) DEFAULT NULL,

  `password` varchar(255) DEFAULT NULL,

  `regDate` timestamp NULL DEFAULT current\_timestamp(),

  `updationDate` timestamp NULL DEFAULT NULL ON UPDATE current\_timestamp()

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `users`

--

INSERT INTO `users` (`id`, `fullName`, `address`, `city`, `gender`, `email`, `password`, `regDate`, `updationDate`) VALUES

(2, 'Demo User', 'Manila, Philippines', 'Delhi', 'female', 'test@gmail.com', 'f925916e2754e5e03f75dd58a5733251', '2016-12-30 05:34:39', '2020-07-05 01:55:24');

--

-- Indexes for dumped tables

--

--

-- Indexes for table `admin`

--

ALTER TABLE `admin`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `appointment`

--

ALTER TABLE `appointment`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `doctors`

--

ALTER TABLE `doctors`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `doctorslog`

--

ALTER TABLE `doctorslog`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `doctorspecilization`

--

ALTER TABLE `doctorspecilization`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `tblcontactus`

--

ALTER TABLE `tblcontactus`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `tblmedicalhistory`

--

ALTER TABLE `tblmedicalhistory`

  ADD PRIMARY KEY (`ID`);

--

-- Indexes for table `tblpatient`

--

ALTER TABLE `tblpatient`

  ADD PRIMARY KEY (`ID`);

--

-- Indexes for table `userlog`

--

ALTER TABLE `userlog`

  ADD PRIMARY KEY (`id`);

--

-- Indexes for table `users`

--

ALTER TABLE `users`

  ADD PRIMARY KEY (`id`),

  ADD KEY `email` (`email`);

--

-- AUTO\_INCREMENT for dumped tables

--

--

-- AUTO\_INCREMENT for table `admin`

--

ALTER TABLE `admin`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

--

-- AUTO\_INCREMENT for table `appointment`

--

ALTER TABLE `appointment`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=8;

--

-- AUTO\_INCREMENT for table `doctors`

--

ALTER TABLE `doctors`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=12;

--

-- AUTO\_INCREMENT for table `doctorslog`

--

ALTER TABLE `doctorslog`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=29;

--

-- AUTO\_INCREMENT for table `doctorspecilization`

--

ALTER TABLE `doctorspecilization`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=14;

--

-- AUTO\_INCREMENT for table `tblcontactus`

--

ALTER TABLE `tblcontactus`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=5;

--

-- AUTO\_INCREMENT for table `tblmedicalhistory`

--

ALTER TABLE `tblmedicalhistory`

  MODIFY `ID` int(10) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=8;

--

-- AUTO\_INCREMENT for table `tblpatient`

--

ALTER TABLE `tblpatient`

  MODIFY `ID` int(10) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=7;

--

-- AUTO\_INCREMENT for table `userlog`

--

ALTER TABLE `userlog`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=30;

--

-- AUTO\_INCREMENT for table `users`

--

ALTER TABLE `users`

  MODIFY `id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=8;

COMMIT;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

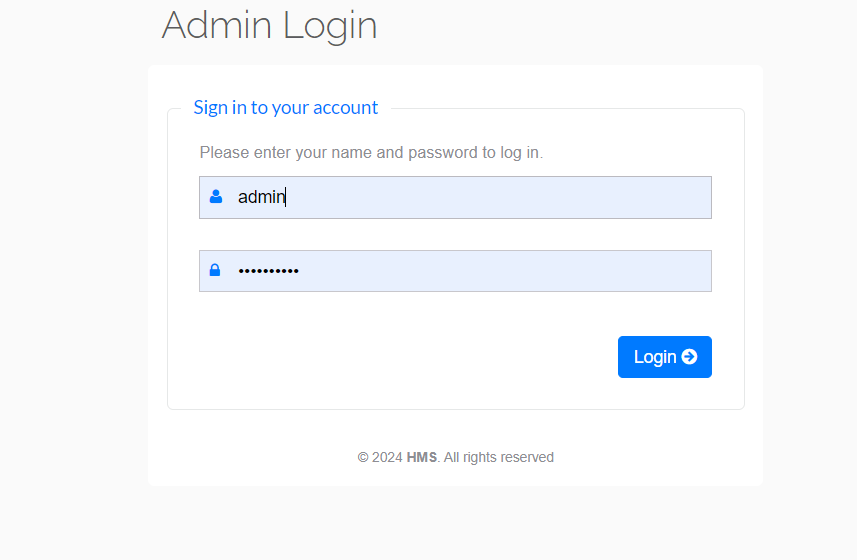
**Chapter 7**

**Result and Discussion (Screen shots of the implementation with front end).**

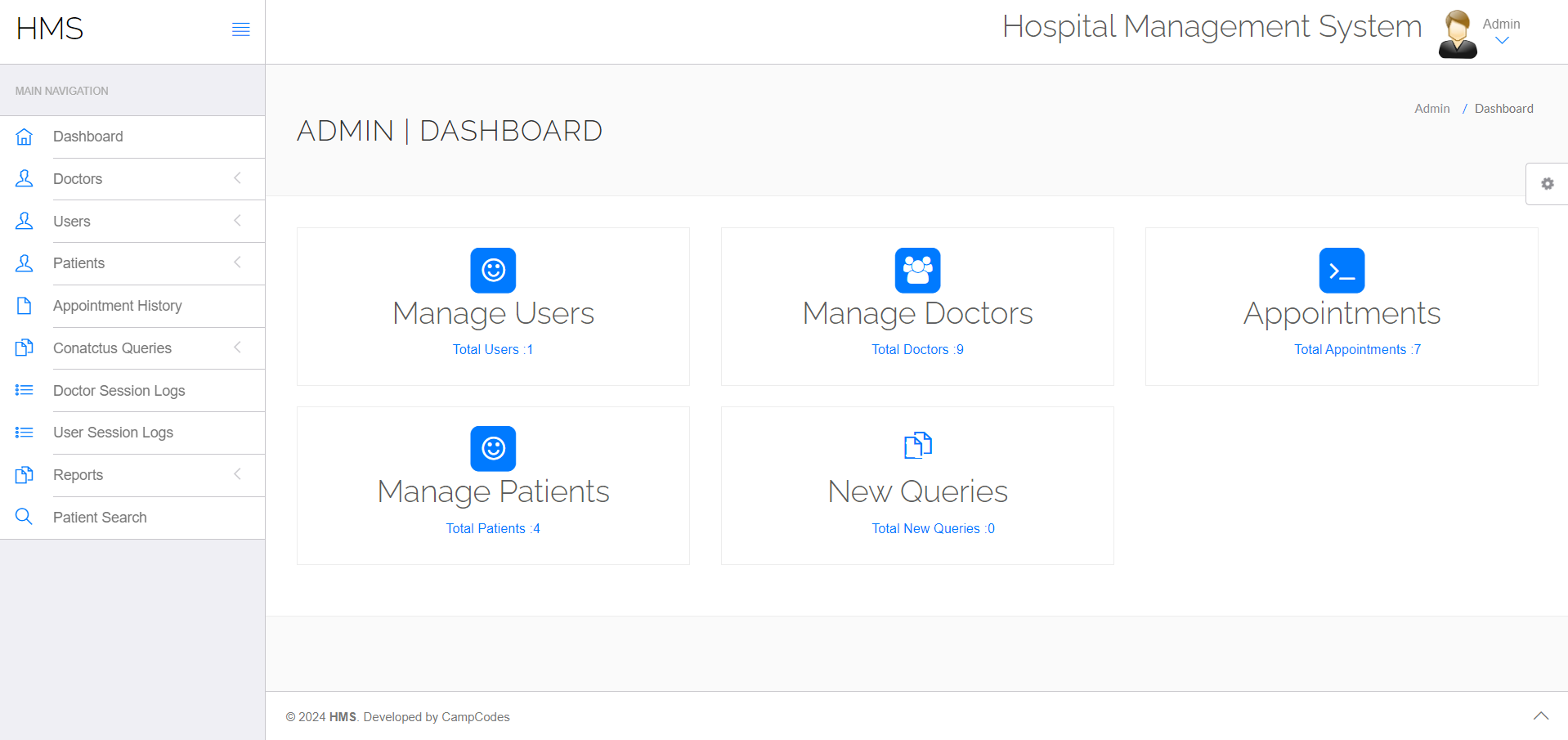
1. **Home Page:**

****

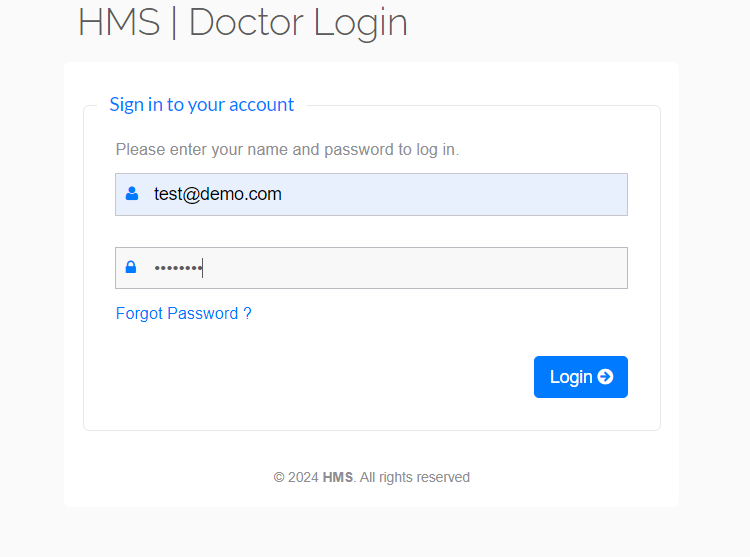
1. **Admin Login:**

****

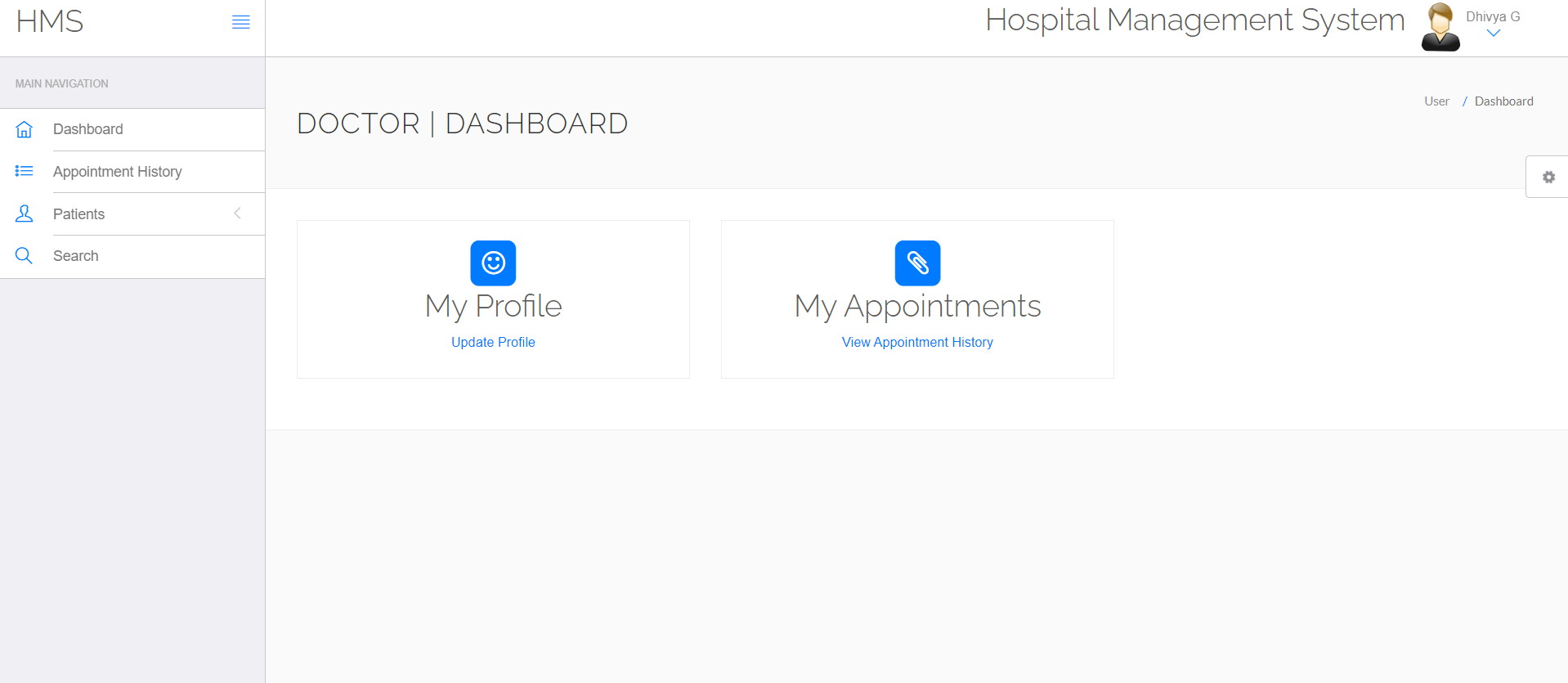
1. **Admin Dashboard:**

****

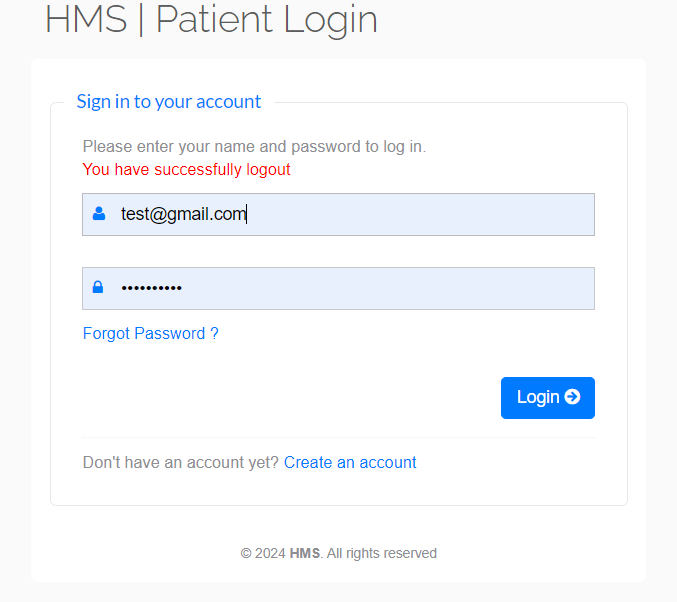
1. **Doctor Login:**

****

1. **Doctor Dashboard:**

****

1. **Patient Login:**

****

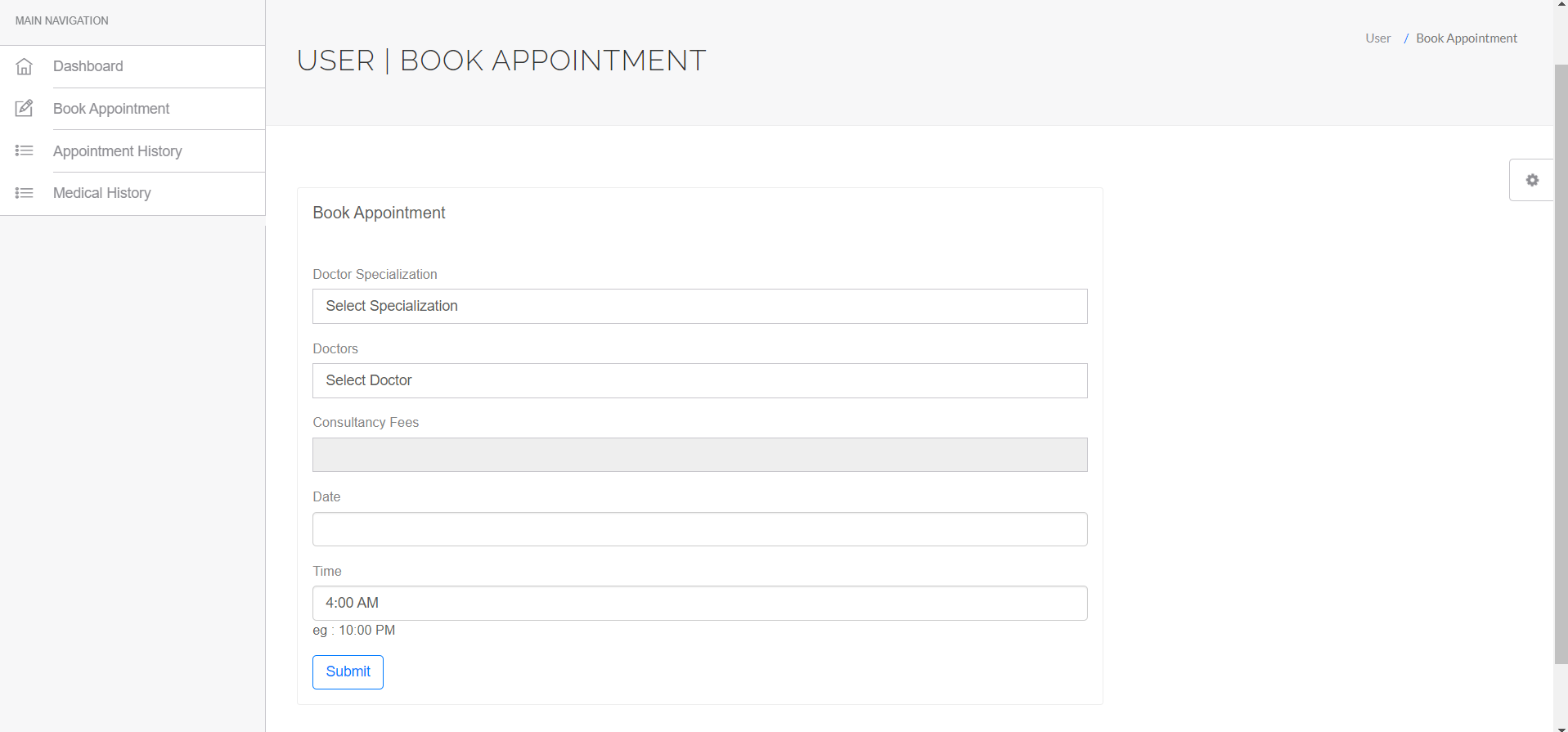
1. **Patient Dashboard:**

****

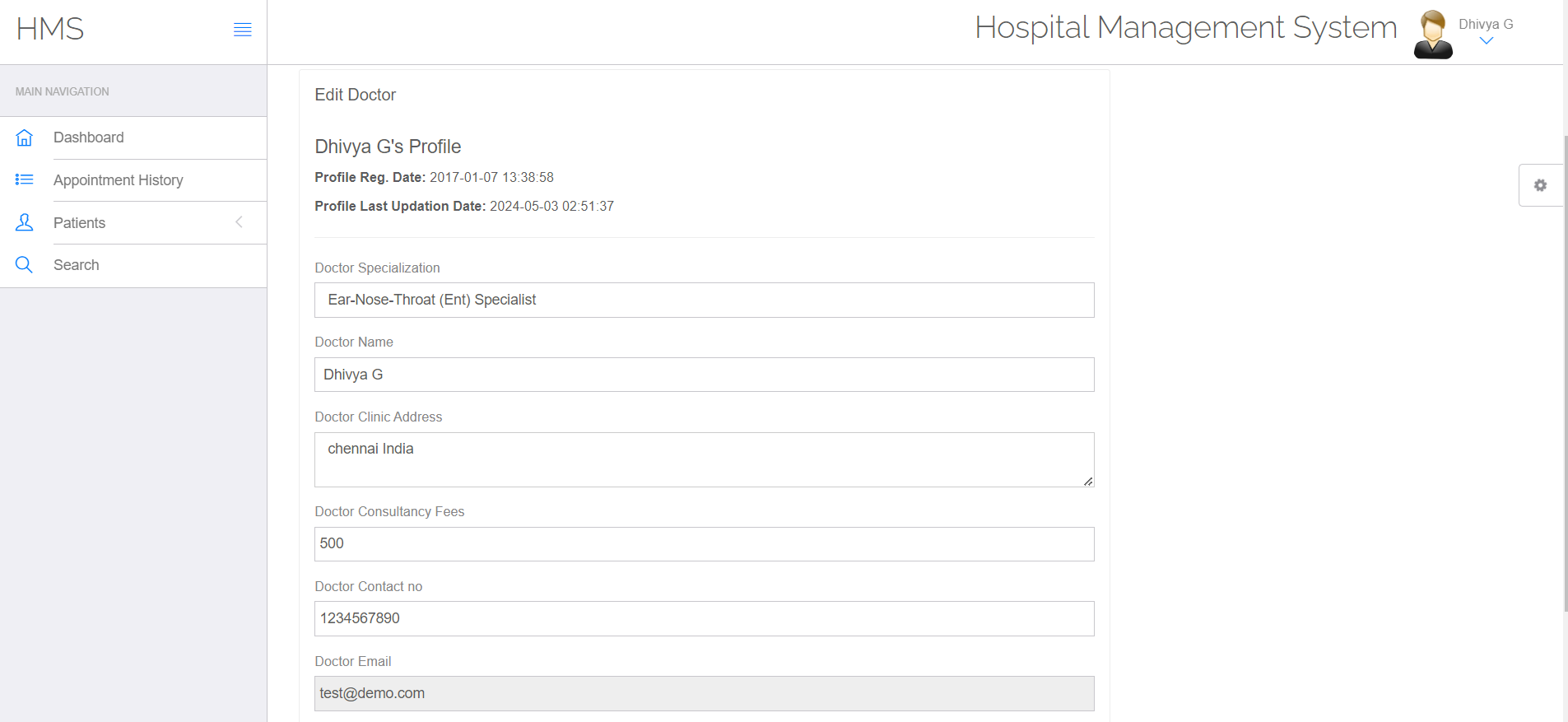
1. **My Profile:**

****

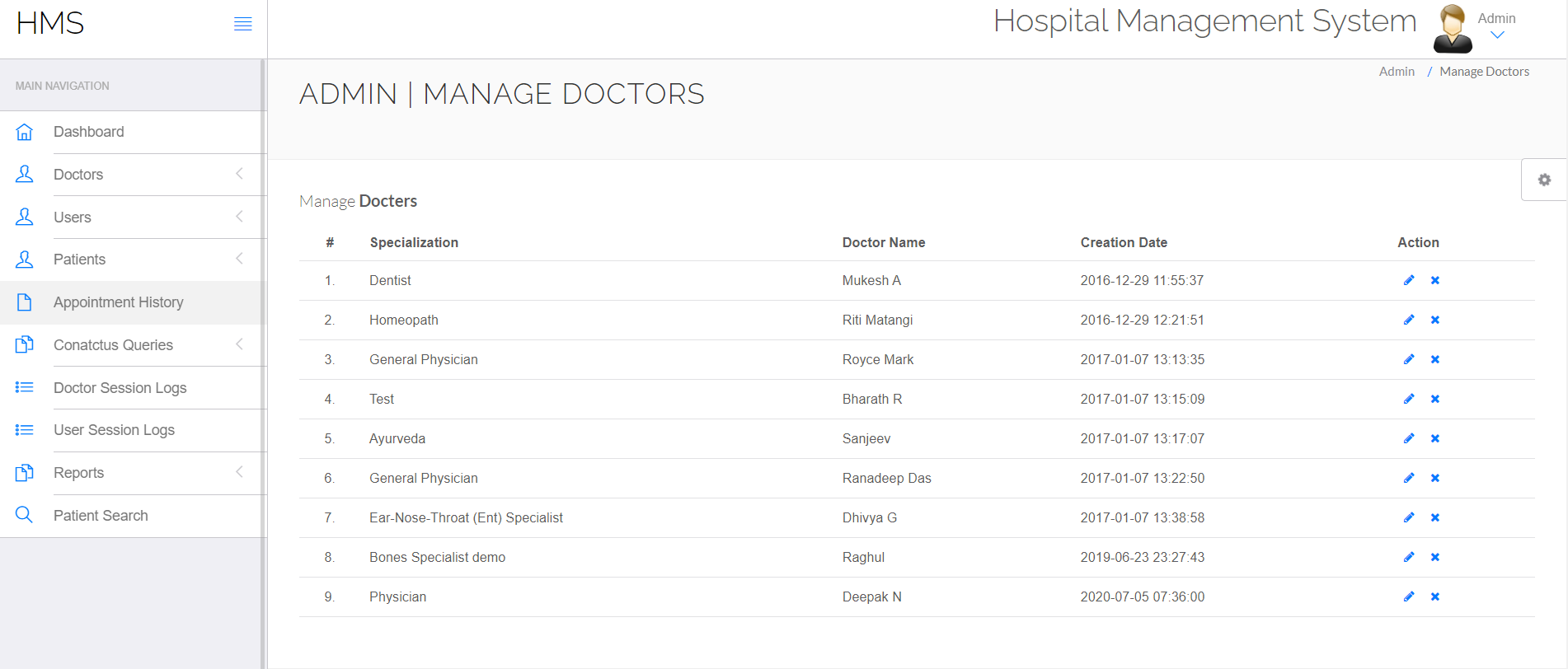
1. **Appointment Booking Page:**

****

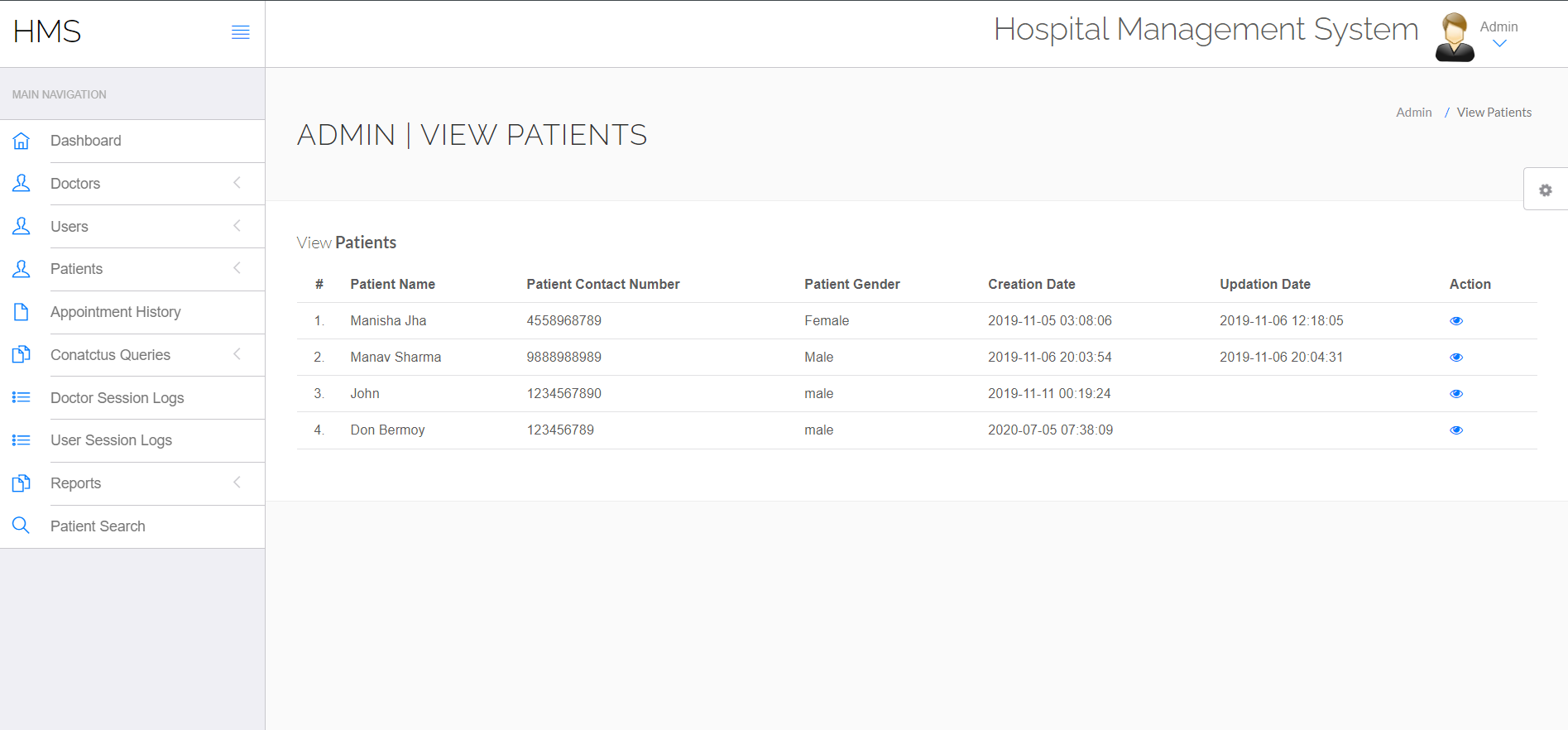
1. **Doctor Profile:**

****

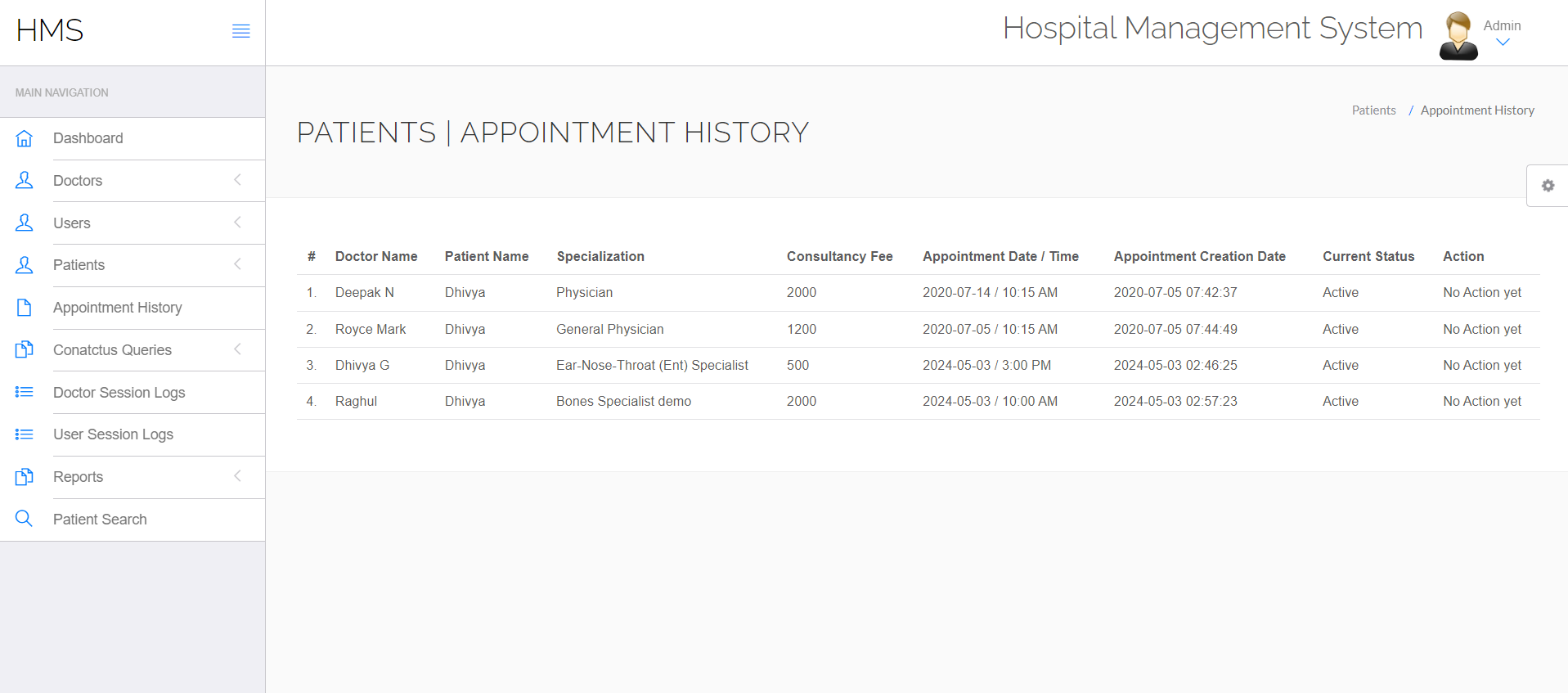
1. **Manage Doctor:**

****

1. **View Patient:**

****

1. **Manage Patient Appointments:**

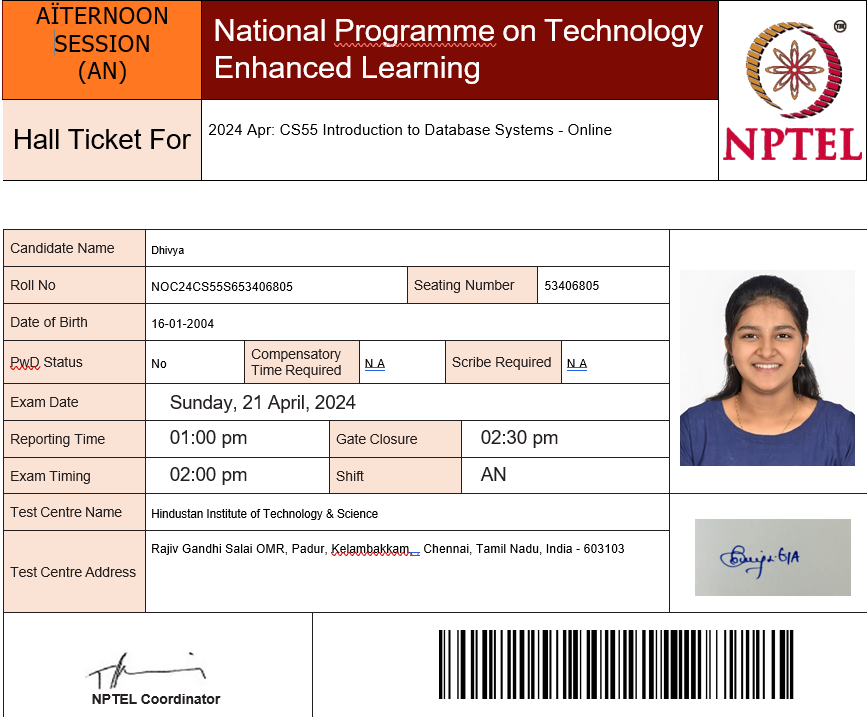
****

**Chapter 8**

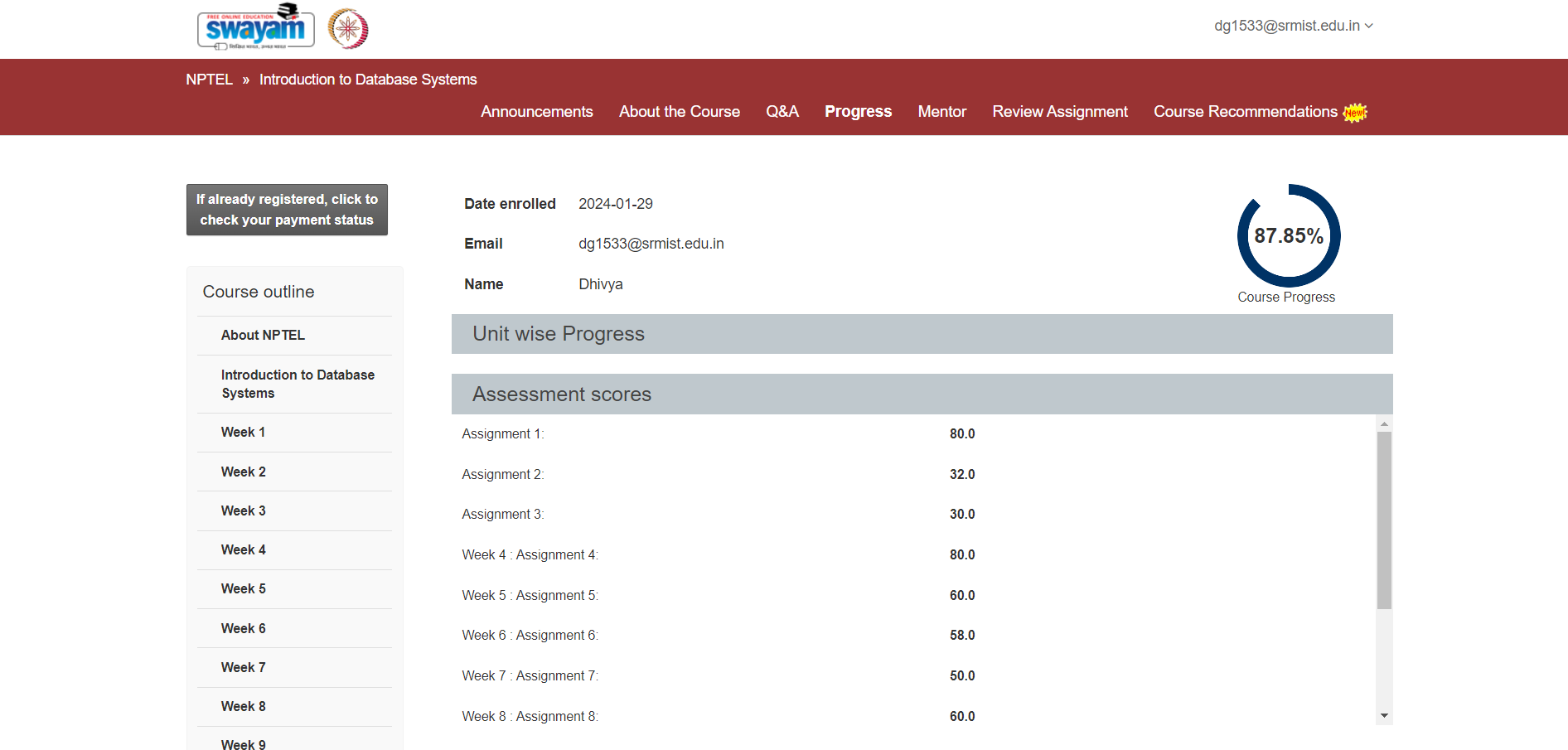
**Attach the Real Time project certificate / Online course certificate.**

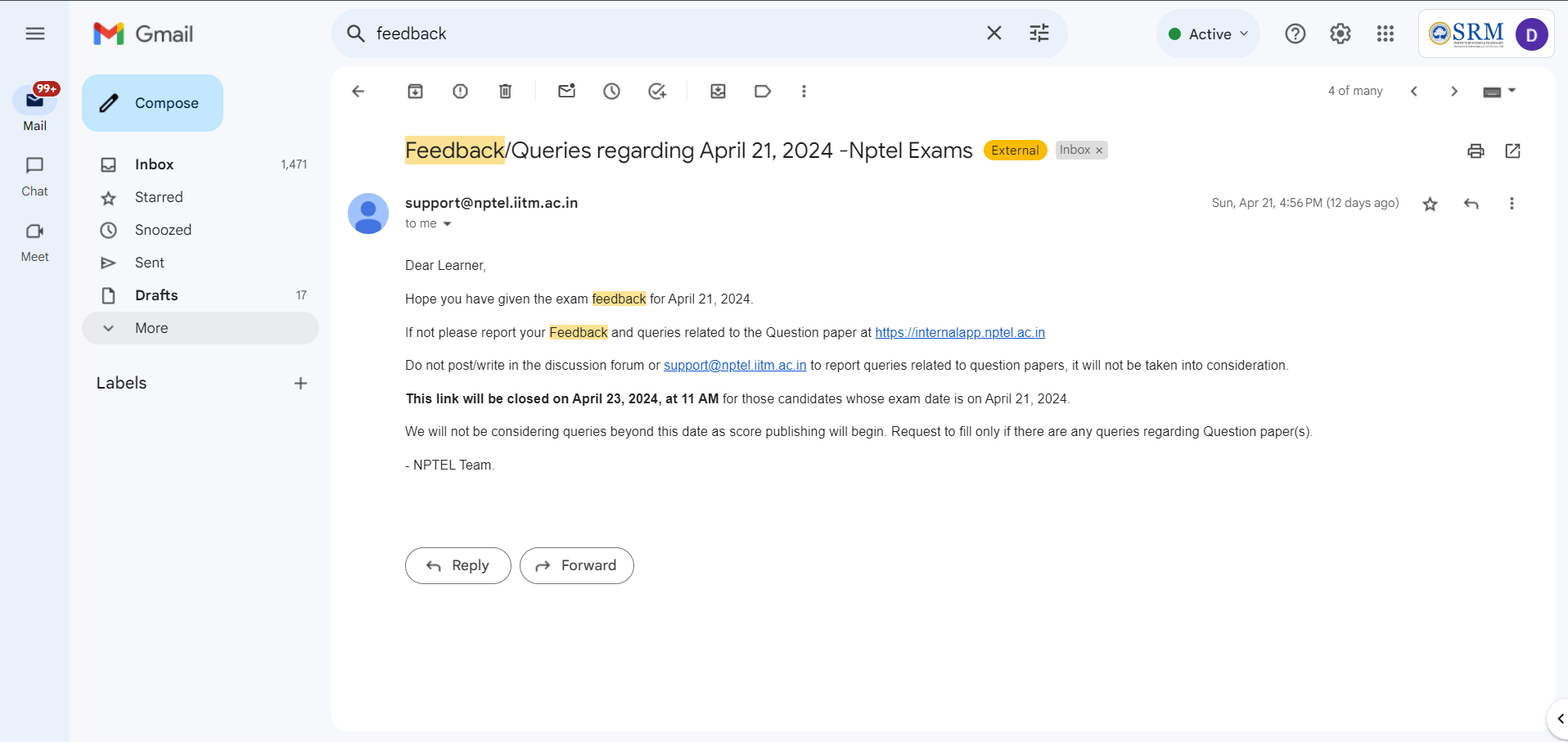
**NPTEL Hall Ticket:**

* 1. **Dhivya G (RA2211056010021)**

****

**NPTEL Progress:**

****

****

* 1. **Mukesh A(RA2211056010008)**

****

* 1. **Riti Matangi (RA2211056010035):**

