**Coding Challenge: Hospital Management System**

1Create SQL Schema from the following classes class, use the class attributes for table column names.

**Create Database HospitalManagementSystem;**

**use HospitalManagementSystem;**

CREATE TABLE Patient (

patientId INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

dateOfBirth DATE,

gender VARCHAR(10),

contactNumber VARCHAR(15),

address VARCHAR(255)

);

CREATE TABLE Doctor (

doctorId INT PRIMARY KEY,

firstName VARCHAR(50),

lastName VARCHAR(50),

specialization VARCHAR(100),

contactNumber VARCHAR(15)

);

CREATE TABLE Appointment (

appointmentId INT PRIMARY KEY,

patientId INT,

doctorId INT,

appointmentDate DATE,

description VARCHAR(255),

FOREIGN KEY (patientId) REFERENCES Patient(patientId),

FOREIGN KEY (doctorId) REFERENCES Doctor(doctorId)

);

A screenshot of a computer

AI-generated content may be incorrect.

1. Create the following model/entity classes within package entity with variables declared private, constructors(default and parametrized,getters,setters and toString())

2. Implement the following for all model classes. Write default constructors and overload the

constructor with parameters, getters and setters, method to print all the member variables and

values.

**#entity/patient.py:**

class Patient:

def \_\_init\_\_(self, patientId=0, firstName="", lastName="", dateOfBirth="", gender="", contactNumber="", address=""):

self.\_\_patientId = patientId

self.\_\_firstName = firstName

self.\_\_lastName = lastName

self.\_\_dateOfBirth = dateOfBirth

self.\_\_gender = gender

self.\_\_contactNumber = contactNumber

self.\_\_address = address

def get\_patientId(self): return self.\_\_patientId

def set\_patientId(self, value): self.\_\_patientId = value

def get\_firstName(self): return self.\_\_firstName

def set\_firstName(self, value): self.\_\_firstName = value

def get\_lastName(self): return self.\_\_lastName

def set\_lastName(self, value): self.\_\_lastName = value

def get\_dateOfBirth(self): return self.\_\_dateOfBirth

def set\_dateOfBirth(self, value): self.\_\_dateOfBirth = value

def get\_gender(self): return self.\_\_gender

def set\_gender(self, value): self.\_\_gender = value

def get\_contactNumber(self): return self.\_\_contactNumber

def set\_contactNumber(self, value): self.\_\_contactNumber = value

def get\_address(self): return self.\_\_address

def set\_address(self, value): self.\_\_address = value

def \_\_str\_\_(self):

return f"{self.\_\_patientId}, {self.\_\_firstName}, {self.\_\_lastName}, {self.\_\_dateOfBirth}, {self.\_\_gender}, {self.\_\_contactNumber}, {self.\_\_address}"

**#entity/doctor.py:**

class Doctor:

def \_\_init\_\_(self, doctorId=0, firstName="", lastName="", specialization="", contactNumber=""):

self.\_\_doctorId = doctorId

self.\_\_firstName = firstName

self.\_\_lastName = lastName

self.\_\_specialization = specialization

self.\_\_contactNumber = contactNumber

def get\_doctorId(self): return self.\_\_doctorId

def set\_doctorId(self, value): self.\_\_doctorId = value

def get\_firstName(self): return self.\_\_firstName

def set\_firstName(self, value): self.\_\_firstName = value

def get\_lastName(self): return self.\_\_lastName

def set\_lastName(self, value): self.\_\_lastName = value

def get\_specialization(self): return self.\_\_specialization

def set\_specialization(self, value): self.\_\_specialization = value

def get\_contactNumber(self): return self.\_\_contactNumber

def set\_contactNumber(self, value): self.\_\_contactNumber = value

def \_\_str\_\_(self):

return f"{self.\_\_doctorId}, {self.\_\_firstName}, {self.\_\_lastName}, {self.\_\_specialization}, {self.\_\_contactNumber}"

**#entity/appointment.py:**

class Appointment:

def \_\_init\_\_(self, appointmentId=0, patientId=0, doctorId=0, appointmentDate="", description=""):

self.\_\_appointmentId = appointmentId

self.\_\_patientId = patientId

self.\_\_doctorId = doctorId

self.\_\_appointmentDate = appointmentDate

self.\_\_description = description

def get\_appointmentId(self): return self.\_\_appointmentId

def set\_appointmentId(self, value): self.\_\_appointmentId = value

def get\_patientId(self): return self.\_\_patientId

def set\_patientId(self, value): self.\_\_patientId = value

def get\_doctorId(self): return self.\_\_doctorId

def set\_doctorId(self, value): self.\_\_doctorId = value

def get\_appointmentDate(self): return self.\_\_appointmentDate

def set\_appointmentDate(self, value): self.\_\_appointmentDate = value

def get\_description(self): return self.\_\_description

def set\_description(self, value): self.\_\_description = value

def \_\_str\_\_(self):

return f"{self.\_\_appointmentId}, {self.\_\_patientId}, {self.\_\_doctorId}, {self.\_\_appointmentDate}, {self.\_\_description}"

3. Define IHospitalService interface/abstract class with following methods to interact with database Keep the interfaces and implementation classes in package dao

Dao/ihospital\_service.py:

from abc import ABC, abstractmethod

class IHospitalService(ABC):

    @abstractmethod

    def getAppointmentById(*self*, *appointmentId*):

        pass

    @abstractmethod

    def getAppointmentsForPatient(*self*, *patientId*):

        pass

    @abstractmethod

    def getAppointmentsForDoctor(*self*, *doctorId*):

        pass

    @abstractmethod

    def scheduleAppointment(*self*, *appointment*):

        pass

    @abstractmethod

    def updateAppointment(*self*, *appointment*):

        pass

    @abstractmethod

    def cancelAppointment(*self*, *appointmentId*):

        pass

6. Define HospitalServiceImpl class and implement all the methods IHospitalServiceImpl .

import pyodbc

from dao.ihospital\_service import IHospitalService

from entity.appointment import Appointment

from myexceptions.patient\_not\_found import PatientNumberNotFoundException

from util.db\_conn\_util import DBConnUtil

class HospitalServiceImpl(IHospitalService):

def \_\_init\_\_(self):

self.conn = DBConnUtil.getConnection()

def getAppointmentById(self, appointmentId):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM Appointment WHERE appointmentId = ?", appointmentId)

row = cursor.fetchone()

if row:

return Appointment(\*row)

return None

def getAppointmentsForPatient(self, patientId):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM Appointment WHERE patientId = ?", patientId)

rows = cursor.fetchall()

if not rows:

raise PatientNumberNotFoundException(f"No appointments found for patient ID {patientId}")

return [Appointment(\*row) for row in rows]

def getAppointmentsForDoctor(self, doctorId):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM Appointment WHERE doctorId = ?", doctorId)

rows = cursor.fetchall()

return [Appointment(\*row) for row in rows]

def scheduleAppointment(self, appointment):

cursor = self.conn.cursor()

cursor.execute(

"INSERT INTO Appointment (appointmentId, patientId, doctorId, appointmentDate, description) VALUES (?, ?, ?, ?, ?)",

(

appointment.get\_appointmentId(),

appointment.get\_patientId(),

appointment.get\_doctorId(),

appointment.get\_appointmentDate(),

appointment.get\_description()

)

)

self.conn.commit()

return True

def updateAppointment(self, appointment):

cursor = self.conn.cursor()

cursor.execute(

"UPDATE Appointment SET patientId = ?, doctorId = ?, appointmentDate = ?, description = ? WHERE appointmentId = ?",

(

appointment.get\_patientId(),

appointment.get\_doctorId(),

appointment.get\_appointmentDate(),

appointment.get\_description(),

appointment.get\_appointmentId()

)

)

self.conn.commit()

return True

def cancelAppointment(self, appointmentId):

cursor = self.conn.cursor()

cursor.execute("DELETE FROM Appointment WHERE appointmentId = ?", appointmentId)

self.conn.commit()

return True

7. Create a utility class DBConnection in a package util with a static variable connection of Type

Connection and a static method getConnection() which returns connection.

Connection properties supplied in the connection string should be read from a property file.

#db\_property\_util.py:

class PropertyUtil:

    @staticmethod

    def get\_property\_string():

        return (

            r"DRIVER={ODBC Driver 17 for SQL Server};"

            r"SERVER=HP\_PAVILION\SQLEXPRESS01;"

            r"DATABASE=HospitalManagementSystem;"

            r"Trusted\_Connection=yes;"

        )

#db.con.py:

import pyodbc

from db\_property\_util import PropertyUtil

class DBConnUtil:

    @staticmethod

    def get\_connection():

        conn\_str = PropertyUtil.get\_property\_string()

        return pyodbc.connect(conn\_str)

8. Create the exceptions in package myexceptions

Define the following custom exceptions and throw them in methods whenever needed. Handle all the

exceptions in main method,

1. PatientNumberNotFoundException :throw this exception when user enters an invalid patient

number which doesn’t exist in db

class PatientNumberNotFoundException(Exception):

    def \_\_init\_\_(*self*, *patient\_id*):

*self*.patient\_id = *patient\_id*

*self*.message = f"Patient with ID {*patient\_id*} not found in the database"

        super().\_\_init\_\_(*self*.message)

9. Create class named MainModule with main method in package mainmod.

Trigger all the methods in service implementation class.

import sys

import os

import datetime

*# Adding the root directory to sys.path to resolve imports*

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from dao.hospital\_service\_imple import HospitalServiceImpl

from entity.appointment import Appointment

from exception.patient\_not\_found import PatientNumberNotFoundException

def validate\_date(*date\_str*):

*"""Validate that the date string is in the format YYYY-MM-DD"""*

    try:

        datetime.datetime.strptime(*date\_str*, '%Y-%m-%d')

        return True

    except ValueError:

        return False

def get\_valid\_integer(*prompt*):

*"""Get a valid integer input from the user"""*

    while True:

        try:

            value = int(input(*prompt*))

            return value

        except ValueError:

            print("Please enter a valid number.")

def main():

    service = HospitalServiceImpl()

    while True:

        print("\n--- Hospital Management System ---")

        print("1. Get Appointment by ID")

        print("2. Get Appointments for Patient")

        print("3. Get Appointments for Doctor")

        print("4. Schedule Appointment")

        print("5. Update Appointment")

        print("6. Cancel Appointment")

        print("7. Exit")

        choice = input("Enter choice: ")

        try:

            if choice == '1':

                aid = get\_valid\_integer("Enter Appointment ID: ")

                appt = service.getAppointmentById(aid)

                if appt:

                    print("\nAppointment Details:")

                    print(appt)

                else:

                    print("Appointment not found.")

            elif choice == '2':

                pid = get\_valid\_integer("Enter Patient ID: ")

                try:

                    appts = service.getAppointmentsForPatient(pid)

                    if appts:

                        print(f"\nAppointments for Patient #{pid}:")

                        for a in appts:

                            print(a)

                    else:

                        print(f"No appointments found for Patient #{pid}.")

                except PatientNumberNotFoundException as e:

                    print(f"Error: Patient with ID {pid} not found.")

            elif choice == '3':

                did = get\_valid\_integer("Enter Doctor ID: ")

                try:

                    appts = service.getAppointmentsForDoctor(did)

                    if appts:

                        print(f"\nAppointments for Doctor #{did}:")

                        for a in appts:

                            print(a)

                    else:

                        print(f"No appointments found for Doctor #{did}.")

                except Exception as e:

                    print(f"Error retrieving appointments for Doctor #{did}: {e}")

            elif choice == '4':

                try:

                    aid = get\_valid\_integer("Enter Appointment ID: ")

*# Check if appointment ID already exists*

                    if service.getAppointmentById(aid):

                        print(f"Error: Appointment with ID {aid} already exists.")

                        continue

                    pid = get\_valid\_integer("Enter Patient ID: ")

                    did = get\_valid\_integer("Enter Doctor ID: ")

                    while True:

                        date = input("Enter Appointment Date (YYYY-MM-DD): ")

                        if validate\_date(date):

                            break

                        print("Invalid date format. Please use YYYY-MM-DD format.")

                    description = input("Enter Appointment Description: ")

                    appointment = Appointment(aid, pid, did, date, description)

                    success = service.scheduleAppointment(appointment)

                    print("Appointment scheduled successfully." if success else "Failed to schedule appointment.")

                except Exception as e:

                    print(f"Error scheduling appointment: {e}")

            elif choice == '5':

                try:

                    aid = get\_valid\_integer("Enter Appointment ID to update: ")

                    appt = service.getAppointmentById(aid)

                    if not appt:

                        print(f"Error: Appointment with ID {aid} not found.")

                        continue

                    print("\nCurrent Appointment Details:")

                    print(appt)

                    print("\nEnter new details (press Enter to keep current value):")

*# Patient ID*

                    pid\_input = input(f"Enter New Patient ID [{appt.get\_patientId()}]: ")

                    pid = int(pid\_input) if pid\_input else appt.get\_patientId()

*# Doctor ID*

                    did\_input = input(f"Enter New Doctor ID [{appt.get\_doctorId()}]: ")

                    did = int(did\_input) if did\_input else appt.get\_doctorId()

*# Date*

                    while True:

                        date\_input = input(f"Enter New Appointment Date [{appt.get\_appointmentDate()}]: ")

                        date = date\_input if date\_input else appt.get\_appointmentDate()

                        if not date\_input or validate\_date(date):

                            break

                        print("Invalid date format. Please use YYYY-MM-DD format.")

*# Description*

                    desc\_input = input(f"Enter New Description [{appt.get\_description()}]: ")

                    description = desc\_input if desc\_input else appt.get\_description()

*# Confirm changes*

                    print("\nNew Appointment Details:")

                    print(f"Appointment ID: {aid}")

                    print(f"Patient ID: {pid}")

                    print(f"Doctor ID: {did}")

                    print(f"Date: {date}")

                    print(f"Description: {description}")

                    confirm = input("\nSave these changes? (y/n): ").lower()

                    if confirm == 'y':

*# Create a new appointment object with updated values*

                        updated\_appointment = Appointment(aid, pid, did, date, description)

                        success = service.updateAppointment(updated\_appointment)

                        print("Appointment updated successfully." if success else "Failed to update appointment.")

                    else:

                        print("Update cancelled.")

                except Exception as e:

                    print(f"Error updating appointment: {e}")

            elif choice == '6':

                try:

                    aid = get\_valid\_integer("Enter Appointment ID to cancel: ")

                    appt = service.getAppointmentById(aid)

                    if not appt:

                        print(f"Error: Appointment with ID {aid} not found.")

                        continue

                    print("\nAppointment to cancel:")

                    print(appt)

                    confirm = input("\nAre you sure you want to cancel this appointment? (y/n): ").lower()

                    if confirm == 'y':

                        success = service.cancelAppointment(aid)

                        print("Appointment cancelled successfully." if success else "Failed to cancel appointment.")

                    else:

                        print("Cancellation aborted.")

                except Exception as e:

                    print(f"Error cancelling appointment: {e}")

            elif choice == '7':

                print("Thank you for using the Hospital Management System. Goodbye!")

                break

            else:

                print("Invalid choice. Please enter a number between 1 and 7.")

        except PatientNumberNotFoundException as e:

            print(f"Error: {e}")

        except Exception as e:

            print(f"Unexpected Error: {str(e)}")

*# Close the database connection when exiting*

    try:

        service.conn.close()

        print("Database connection closed.")

    except:

        pass

if \_\_name\_\_ == '\_\_main\_\_':

    main()