Coding Challenge:
Hospital Management System

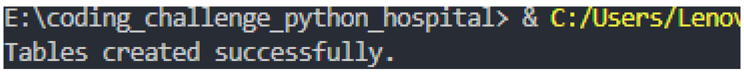
Mohammed Ibrahim Sheriff U

Batch 04

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

Code:

```
import sqlite3

conn = sqlite3.connect("hospital.db")
cursor = conn.cursor()

cursor.execute("""
create table if not exists patient (
    patientid int primary key,
    firstname varchar(50) not null,
    lastname varchar(50) not null,
    dateofbirth date not null,
    gender varchar(10) check (gender in ('male', 'female', 'other')),
    contactnumber varchar(15),
    address varchar(255)
)
""")

cursor.execute("""
create table if not exists doctor (
    doctorid int primary key,
    firstname varchar(50) not null,
    lastname varchar(50) not null,
    specialization varchar(100),
    contactnumber varchar(15)
)
""")

cursor.execute("""
create table if not exists appointment (
    appointmentid int primary key,
    patientid int,
    doctorid int,
    appointmentdate date not null,
    description text,
    foreign key (patientid) references patient(patientid) on delete cascade,
    foreign key (doctorid) references doctor(doctorid) on delete set null
)
""")

conn.commit()
conn.close()
print(" Tables created successfully.")
```



```
E:\coding_challenge_python_hospital> & C:/Users/Lenov
Tables created successfully.
```

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.

1. Define `Patient` class with the following confidential attributes:

a. patientId

b. firstName

c. lastName;

d. dateOfBirth

e. gender

f. contactNumber

g. address;

Code:

```
class Patient:
    def __init__(self, patientid=None, firstname=None, lastname=None,
                 dateofbirth=None, gender=None, contactnumber=None, address=None):
        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 get_firstname(self):
        return self.__firstname

    def get_lastname(self):
        return self.__lastname

    def get_dateofbirth(self):
        return self.__dateofbirth

    def get_gender(self):
        return self.__gender
```

```
def get_contactnumber(self):
    return self.__contactnumber

def get_address(self):
    return self.__address

# Setters
def set_patientid(self, patientid):
    self.__patientid = patientid

def set_firstname(self, firstname):
    self.__firstname = firstname

def set_lastname(self, lastname):
    self.__lastname = lastname

def set_dateofbirth(self, dateofbirth):
    self.__dateofbirth = dateofbirth

def set_gender(self, gender):
    self.__gender = gender

def set_contactnumber(self, contactnumber):
    self.__contactnumber = contactnumber

def set_address(self, address):
    self.__address = address

def display_details(self):
    print("Patient ID:", self.__patientid)
    print("First Name:", self.__firstname)
```

```
        print("Last Name:", self.__lastname)
        print("Date of Birth:", self.__dateofbirth)
        print("Gender:", self.__gender)
        print("Contact Number:", self.__contactnumber)
        print("Address:", self.__address)
```

2. Define ‘Doctor` class with the following confidential attributes:

a. doctorId

b. firstName

c. lastName

d. specialization

e. contactNumber;

Code:

```
class Doctor:
    def __init__(self, doctorid=None, firstname=None, lastname=None,
                 specialization=None, contactnumber=None):
        self.__doctorid = doctorid
        self.__firstname = firstname
        self.__lastname = lastname
        self.__specialization = specialization
        self.__contactnumber = contactnumber

    def get_doctorid(self):
        return self.__doctorid

    def get_firstname(self):
        return self.__firstname

    def get_lastname(self):
        return self.__lastname
    def get_specialization(self):
        return self.__specialization

    def get_contactnumber(self):
        return self.__contactnumber

    def set_doctorid(self, doctorid):
        self.__doctorid = doctorid

    def set_firstname(self, firstname):
        self.__firstname = firstname

    def set_lastname(self, lastname):
        self.__lastname = lastname

    def set_specialization(self, specialization):
        self.__specialization = specialization

    def set_contactnumber(self, contactnumber):
        self.__contactnumber = contactnumber

    def display_details(self):
        print("Doctor ID:", self.__doctorid)
        print("First Name:", self.__firstname)
        print("Last Name:", self.__lastname)
        print("Specialization:", self.__specialization)
        print("Contact Number:", self.__contactnumber)
```

3. Appointment Class:

a. appointmentId

b. patientId

c. doctorId

d. appointmentDate

e. description

Code:

```
class Appointment:
  def __init__(self,appointmentid = None, patientid = None, doctorid = None, appointmentdate = None, description = None):
    self.__appointmentid = appointmentid
    self.__patientid = patientid
    self.__doctorid = doctorid
    self.__appointmentdate = appointmentdate
    self.__description = description

  def get_appointmentid(self):
    return self.__appointmentid
  def get_patientid(self):
    return self.__patientid
  def get_doctorid(self):
    return self.__doctorid
  def get_appointmentdate(self):
    return self.__appointmentdate
  def get_description(self):
    return self.__description

  def set_appointmentid(self,appointmentid):
    self.__appointmentid = appointmentid
  def set_patientid(self,patientid):
    self.__patientid = patientid
  def set_doctorid(self,doctorid):
    self.__doctorid = doctorid
  def set_appointmentdate(self, appointmentdate):
    self.__appointmentdate = appointmentdate
  def set_description(self,description):
    self.__description = description
```

```
 def display_details(self):
   print("Appointment ID: ",self.__appointmentid)
   print("Patient ID: ", self.__patientid)
   print("Doctor ID: ", self.__doctorid)
   print("Appointment Date: ", self.__appointmentdate)
   print("Description: ", self.__description)
```

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

a. getAppointmentById() i. Parameters: appointmentId ii. ReturnType: Appointment object

b. getAppointmentsForPatient() i. Parameters: patientId ii. ReturnType: List of Appointment objects

c. getAppointmentsForDoctor() i. Parameters: doctorId ii. ReturnType: List of Appointment objects

d. scheduleAppointment() i. Parameters: Appointment Object ii. ReturnType: Boolean

e. updateAppointment() i. Parameters: Appointment Object ii. ReturnType: Boolean

f. ancelAppointment() i. Parameters: AppointmentId ii. ReturnType: Boolean

Code:

```
from abc import ABC, abstractmethod
from entity.appointment import Appointment

class IHospitalService(ABC):
  @abstractmethod
  def get_appointment_by_id(self, appointmentid) -> Appointment:
    pass
  @abstractmethod
  def get_appointments_for_patient(self, patientid) -> list:
    pass
  @abstractmethod
  def get_appointments_for_doctor(self, doctorid) -> list:
    pass
  @abstractmethod
  def schedule_appointment(self, appointment: Appointment) -> bool:
    pass
  @abstractmethod
  def update_appointment(self, appointment: Appointment) -> bool:
    pass
  @abstractmethod
  def cancel_appointment(self, appointmentid) -> bool:
    pass
```

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

Code:

```
import sqlite3
from dao.ihospitalservice import IHospitalService
from entity.appointment import Appointment
from util.dbconnutil import DBConnUtil
from myexceptions.patientnumbernotfoundexception import PatientNumberNotFoundException

class HospitalServiceImpl(IHospitalService):

    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def get_appointment_by_id(self, appointmentid: int) -> Appointment:
        try:
            cursor = self.conn.cursor()
            cursor.execute("select * from appointment where appointmentid = ?", (appointmentid,))
            row = cursor.fetchone()
            if row:
```

```
            return Appointment(*row)
          else:
            raise Exception("Appointment not found")
        except Exception as e:
          print("Error:", e)
          return None

    def get_appointments_for_patient(self, patientid: int) -> List[Appointment]:
        try:
          cursor = self.conn.cursor()
          cursor.execute("select * from appointment where patientid = ?", (patientid,))
          rows = cursor.fetchall()
          if not rows:
            raise PatientNumberNotFoundException("No appointments found for patient ID " +
str(patientid))
          return [Appointment(*row) for row in rows]
        except PatientNumberNotFoundException as pne:
          print(pne)
          return []
        except Exception as e:
          print("Error:", e)
          return []

    def get_appointments_for_doctor(self, doctorid: int) -> List[Appointment]:
        try:
          cursor = self.conn.cursor()
          cursor.execute("select * from appointment where doctorid = ?", (doctorid,))
          rows = cursor.fetchall()
          return [Appointment(*row) for row in rows]
        except Exception as e:
          print("Error:", e)
          return []
    def schedule_appointment(self, appointment: Appointment) -> bool:
        try:
            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
        except Exception as e:
            print("Error: Appointment already exist", e)
            return False


    def update_appointment(self, appointment: Appointment) -> bool:
        try:
            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 cursor.rowcount > 0
        except Exception as e:
            print("Error:", e)
            return False

    def cancel_appointment(self, appointmentid: int) -> bool:
        try:
            cursor = self.conn.cursor()
            cursor.execute("delete from appointment where appointmentid = ?", (appointmentid,))
            self.conn.commit()
            return cursor.rowcount > 0
        except Exception as e:
            print("Error:", e)
            return False
```

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. Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

Code:

```
import sqlite3


class DBPropertyUtil:
    @staticmethod
    def get_property_string(filename):
        try:
            props = {}
            with open(filename, 'r') as file:
```

```
            for line in file:
                if '=' in line:
                    key, value = line.strip().split('=')
                    props[key.strip()] = value.strip()

        # For SQLite, we expect: filename=hospital.db
        db_file = props.get('filename')
        return db_file
    except Exception as e:
        print("Error reading property file:", e)
        return None



class DBConnUtil:
    @staticmethod
    def get_connection():
        try:
            conn_str = DBPropertyUtil.get_property_string("db.properties")
            if conn_str is None:
                raise Exception("Connection string is empty or invalid")
            return sqlite3.connect(conn_str)
        except Exception as e:
            print("Database connection failed:", e)
            return None
```

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, message="Patient number not found in the database."):
        super().__init__(message)    # I have expanded in mainmodule in next step
```

9. Create class named MainModule with main method in package mainmod. Trigger all the methods in service implementation class

Code:

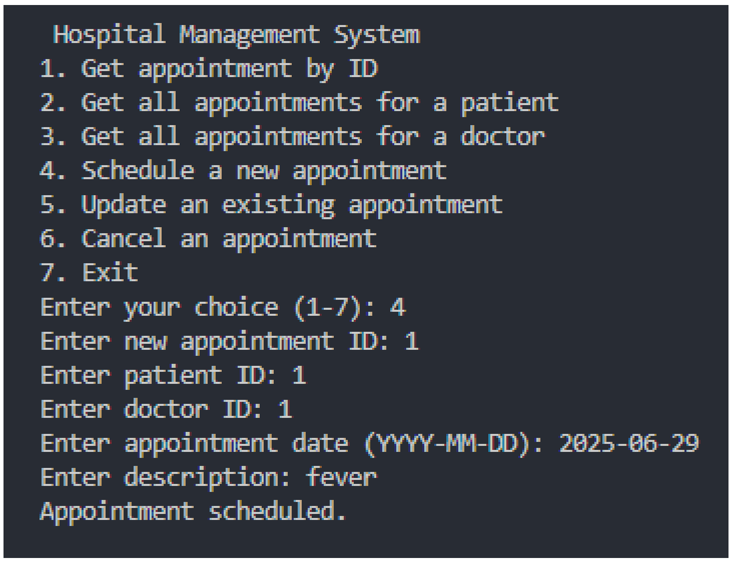
```
from dao.hospitalserviceimpl import HospitalServiceImpl
from entity.appointment import Appointment
from myexceptions.patientnumbernotfoundexception import PatientNumberNotFoundException


def main():
    service = HospitalServiceImpl()

    while True:
        print("\n Hospital Management System ")
        print("1. Get appointment by ID")
        print("2. Get all appointments for a patient")
        print("3. Get all appointments for a doctor")
        print("4. Schedule a new appointment")
        print("5. Update an existing appointment")
        print("6. Cancel an appointment")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == '1':
            aid = int(input("Enter appointment ID: "))
            appointment = service.get_appointment_by_id(aid)
            if appointment:
                appointment.display_details()
            else:
                print("Appointment not found.")
```

```
elif choice == '2':
    pid = int(input("Enter patient ID: "))
    try:
        appointments = service.get_appointments_for_patient(pid)
        for appt in appointments:
            appt.display_details()
            print("-" * 30)
    except PatientNumberNotFoundException as pne:
        print("Exception:", pne)

elif choice == '3':
    did = int(input("Enter doctor ID: "))
    appointments = service.get_appointments_for_doctor(did)
    if appointments:
        for appt in appointments:
            appt.display_details()
            print("-" * 30)
    else:
        print("No appointments found.")

elif choice == '4':
    aid = int(input("Enter new appointment ID: "))
    pid = int(input("Enter patient ID: "))
    did = int(input("Enter doctor ID: "))
    date = input("Enter appointment date (YYYY-MM-DD): ")
    desc = input("Enter description: ")
    appointment = Appointment(aid, pid, did, date, desc)
    success = service.schedule_appointment(appointment)
    print("Appointment scheduled." if success else "Failed to schedule appointment.")

elif choice == '5':
    aid = int(input("Enter appointment ID to update: "))
```

```
            pid = int(input("Enter updated patient ID: "))
            did = int(input("Enter updated doctor ID: "))
            date = input("Enter updated appointment date (YYYY-MM-DD): ")
            desc = input("Enter updated description: ")
            appointment = Appointment(aid, pid, did, date, desc)
            success = service.update_appointment(appointment)
            print("Appointment updated." if success else "Failed to update appointment.")

        elif choice == '6':
            aid = int(input("Enter appointment ID to cancel: "))
            success = service.cancel_appointment(aid)
            print("Appointment cancelled." if success else "Appointment not found.")

        elif choice == '7':
            print("Exiting system.")
            break

        else:
            print("Invalid choice. Please select from 1 to 7.")

if __name__ == "__main__":
    main()
```

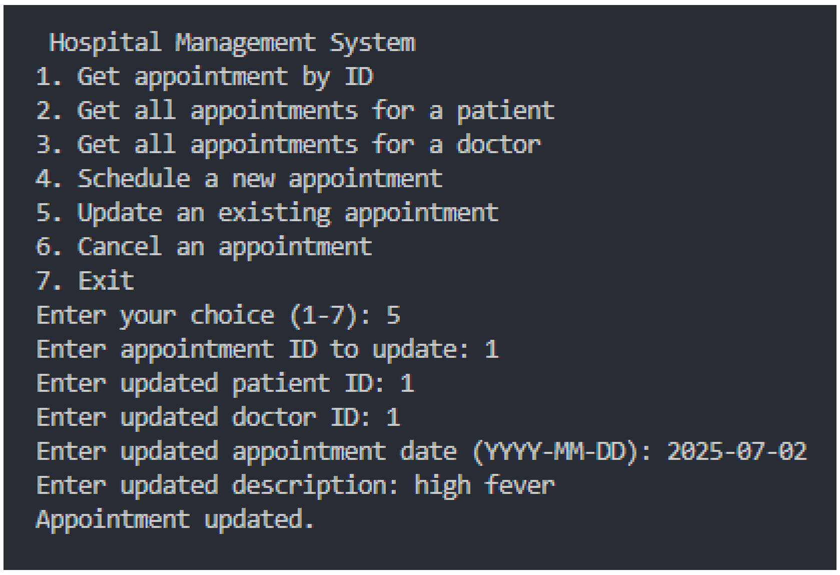
OUTPUTS:

Scheduling an appointment:



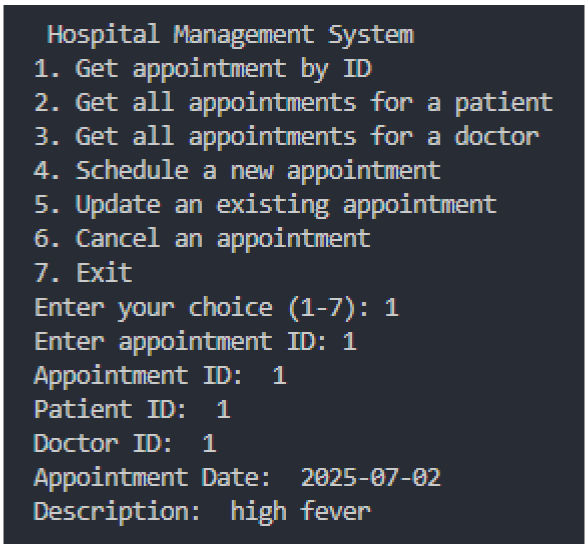
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 4
Enter new appointment ID: 1
Enter patient ID: 1
Enter doctor ID: 1
Enter appointment date (YYYY-MM-DD): 2025-06-29
Enter description: fever
Appointment scheduled.
```

Updating an appointment:



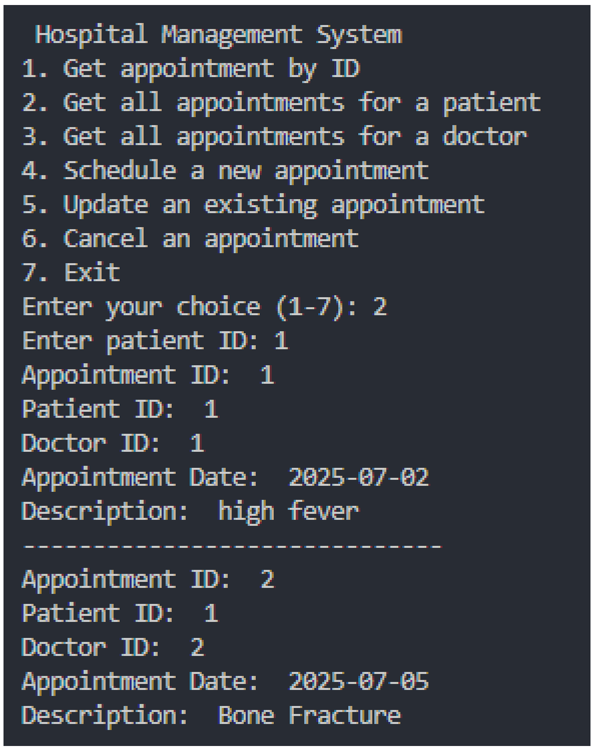
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 5
Enter appointment ID to update: 1
Enter updated patient ID: 1
Enter updated doctor ID: 1
Enter updated appointment date (YYYY-MM-DD): 2025-07-02
Enter updated description: high fever
Appointment updated.
```

Get Appointment Details:



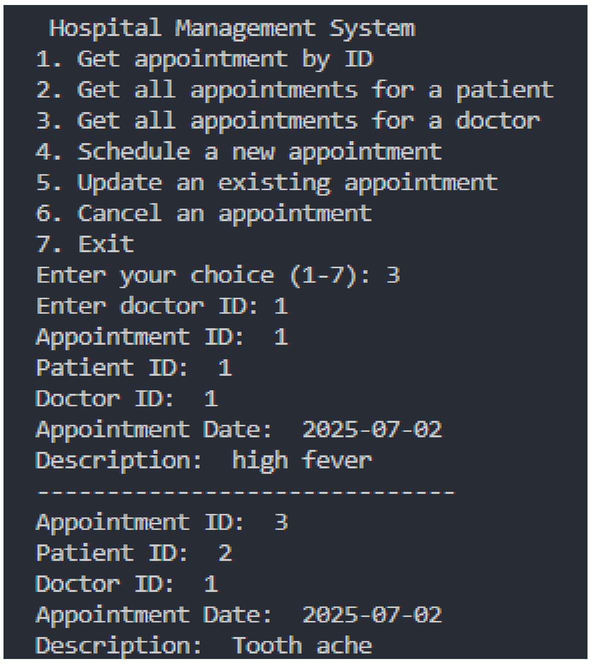
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 1
Enter appointment ID: 1
Appointment ID:  1
Patient ID:  1
Doctor ID:  1
Appointment Date:  2025-07-02
Description:  high fever
```

All Appointments same patient has:



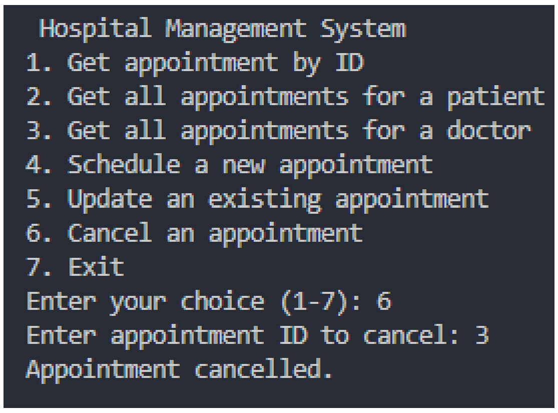
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 2
Enter patient ID: 1
Appointment ID:  1
Patient ID:  1
Doctor ID:  1
Appointment Date:  2025-07-02
Description:  high fever
------------------------------
Appointment ID:  2
Patient ID:  1
Doctor ID:  2
Appointment Date:  2025-07-05
Description:  Bone Fracture
```

All Appointments a doctor has:



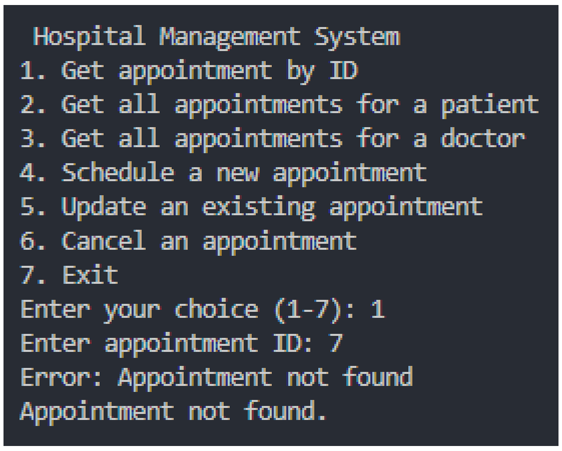
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 3
Enter doctor ID: 1
Appointment ID:  1
Patient ID:  1
Doctor ID:  1
Appointment Date:  2025-07-02
Description:  high fever
------------------------------
Appointment ID:  3
Patient ID:  2
Doctor ID:  1
Appointment Date:  2025-07-02
Description:  Tooth ache
```

Cancelling an Appointment:



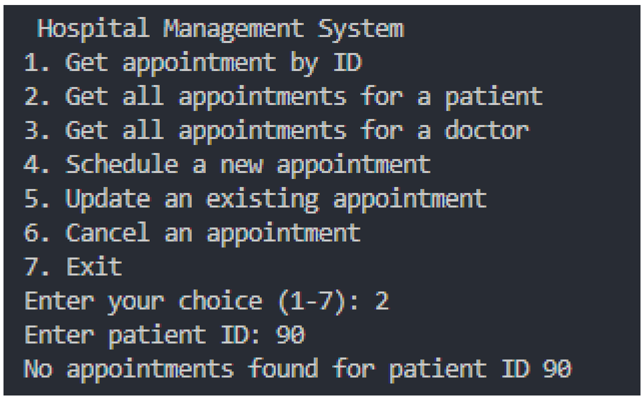
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 6
Enter appointment ID to cancel: 3
Appointment cancelled.
```

Error Handling for non existing appointment:



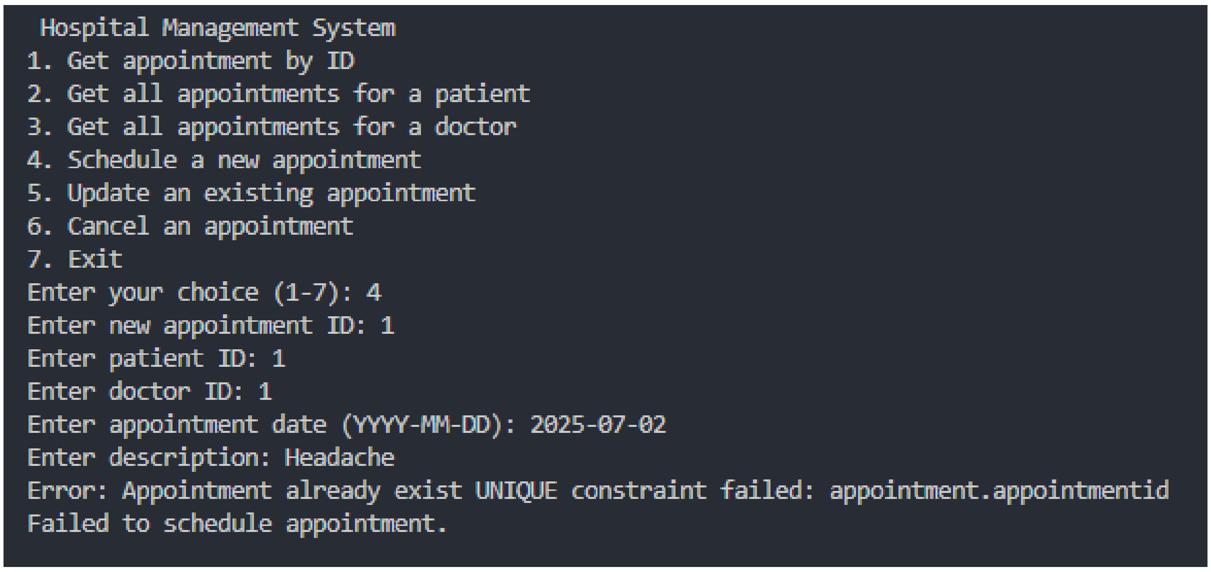
```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 1
Enter appointment ID: 7
Error: Appointment not found
Appointment not found.
```

Error handling for getting details for non existing patient:



```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 2
Enter patient ID: 90
No appointments found for patient ID 90
```

Error Handling for re-registering existing appointment:



```
 Hospital Management System
1. Get appointment by ID
2. Get all appointments for a patient
3. Get all appointments for a doctor
4. Schedule a new appointment
5. Update an existing appointment
6. Cancel an appointment
7. Exit
Enter your choice (1-7): 4
Enter new appointment ID: 1
Enter patient ID: 1
Enter doctor ID: 1
Enter appointment date (YYYY-MM-DD): 2025-07-02
Enter description: Headache
Error: Appointment already exist UNIQUE constraint failed: appointment.appointmentid
Failed to schedule appointment.
```