# CODING CHALLENGE

# HOSPITAL MANAGEMENT SYSTEM

Name : Maddaka Tejaswini

Python batch : 1

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

```
create DATABASE Hospital_Management;
use Hospital_Management;
create table Patient(
    patientId varchar(5) PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    dateOfBirth DATE NOT NULL,
    gender VARCHAR(10) NOT NULL,
    contactNumber VARCHAR(15) NOT NULL,
    address VARCHAR(255) NOT NULL);

create table Doctor (
    doctorId varchar(5) PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    specialization VARCHAR(50) NOT NULL,
    contactNumber VARCHAR(15) NOT NULL);

create table Appointment(
     appointmentId int primary key,
      patientId varchar(5) not null,
      doctorId varchar(5) not null,
      appointmentDate DATETIME not null,
       description varchar(50),
      FOREIGN KEY(patientId) REferences Patient(patientId),
      FOREIGN KEY (doctorId) References Doctor(DoctorId)
);

Insert into Patient(patientId,
firstName,lastName,dateOfBirth,gender,contactNumber,address)
VALUES
('p1', 'Anne','John', '2001-10-12','Female','9852654753','14/480,Church
street,Miami'),
('p2', 'Emma','Thomas','1998-01-08', 'Female','8695756984','1C-10,
Lakeview,Portland'),
('p3', 'Noah','Olivia','2000-09-04', 'Male','789654357','12-B,Grifender street,New
York'),
('p4', 'David','Son','1999-02-05', 'Male','7895651423','63/1,Johnson street,San
Jose'),
('p5', 'Martin','Rich','2002-04-06', 'Male','9563285412','56/9,Wainut,Tucson'),
('p6', 'Blue','Harris','1997-10-03', 'Male','6859352946','35-D,Main street,Fort
Worth'),
('p7', 'Kevin','Jose','2003-07-12', 'Male','8534976581','89/7,Cedar,Honolulu'),
('p8', 'Pat','Carol','2001-04-09', 'Male','7689572612','475,Maple,Omaha'),
('p9', 'Amy','Mathew','2004-10-12', 'Female','7654892642','165/1B,Kingston,Las
Vegas'),
('p10', 'Laura','James','1998-03-05', 'Female','9556411791','164,Second
street,Phoenix');
```

```sql
INSERT into Doctor(doctorId,firstName,lastName,specialization,contactNumber)
Values
('d1', 'Dr. Amanda','Stone', 'Cardiologist', '9123456780'),
('d2', 'Dr. Michael','Rivera', 'Neurologist', '6329087654'),
('d3', 'Dr. Olivia ','Henry', 'Surgeon', '98766546543'),
('d4', 'Dr. David','Wong', 'Pediatrician', '8769806547'),
('d5', 'Dr. Emily','Johnson', 'Dermatologist', '9876543210'),
('d6', 'Dr. Benjamin','Carter', 'Oncologist', '8907654762'),
('d7', 'Dr. Lily ','Martinez', 'Dermatologist', '9764789432'),
('d8', 'Dr. William','Lee', 'Rhumetologist', '9876867869'),
('d9', 'Dr. Isabella','Thompson', 'Gastroenterologist', '8769806598'),
('d10', 'Dr. Ethan','Brooks', 'Endocrinologist', '7869087651');

Insert into Appointment(appointmentId,patientId,doctorId,appointmentDate,description)
Values
(1,'p10','d7','2024-10-28','Hair loss'),
(2,'p8','d9','2024-11-02 ','Stomach Ache'),
(3,'p1','d10','2024-10-17','diabetes'),
(4,'p3','d3','2024-10-11','Surgery'),
(5,'p4','d7','2024-10-29','Hair loss'),
(6,'p5','d2','2024-11-03','Migrane'),
(7,'p6','d1','2024-11-01','Hyper Tension'),
(8,'p9','d5','2024-10-30','Hair loss'),
(9,'p2','d4','2024-10-12','Allergy'),
(10,'p7','d8','2024-10-13','Arthritis');
```

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

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

   a. patientId

   b. firstName

   c. lastName

   d. dateOfBirth

   e. gender

   f. contactNumber

   g. address

**entity/patient.py**

```python
class Patient:
    def __init__(self, 
patientId,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

    #setter methods

    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

    #getter methods

    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

    def __str__(self):
        return f"Patient ID: {self.patientId()}, Name: {self.firstName} 
{self.lastName}, " \
               f"DOB: {self.dateOfBirth}, Gender: {self.gender}, Contact: 
{self.contactNumber}, " \
               f"Address: {self.address}"
```

2. Define **Doctor** class with the following confidential attributes:

a. doctorId

b. firstName

c. lastName

d. specialization

e. contactNumber

**entity/doctor.py**

```
class Doctor:
    def
__init__(self,doctorId,firstName,lastName,specialization,contactNumber):
        self.doctorId = doctorId
        self.firstName = firstName
        self.lastName = lastName
        self.specialization = specialization
        self.contactNumber = contactNumber

    #Setter methods

    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

    #Getter methods

    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 __str__(self):
        return f"Doctor ID: {self.doctorId}, Name: {self.firstName}
{self.lastName}, " \
               f"Specialization: {self.specialization}, Contact: 
{self.contactNumber}"
```

3. Appointment Class:

a. appointmentId
b. patientId
c. doctorId
d. appointmentDate
e. description

**entity/appointment.py**

```
class Appointment:
    def 
__init__(self,appointmentId,patientId,doctorId,appointmentDate,description):
        self.patientId = patientId
        self.doctorId = doctorId
        self.appointmentId = appointmentId
        self.appointmentDate = appointmentDate
        self.description = description

#Setter methods

    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

    #Getter methods

    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 __str__(self):
        return f"Appointment ID: {self.appointmentId}, Patient ID: 
{self.patientId}, Doctor ID: {self.doctorId}, " \
               f"Date: {self.appointmentDate}, Description: 
{self.description}"
```

**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. cancelAppointment()

  i. Parameters: AppointmentId

  ii. ReturnType: Boolean

**dao/i_hospital_service.py**

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

class IHospitalService(ABC):
    @abstractmethod
    def getAppointmentById(self, appointmentId) -> Appointment:
        pass

    @abstractmethod
    def getAppointmentsForPatient(self, patientId) -> List[Appointment]:
        pass

    @abstractmethod
    def getAppointmentsForDoctor(self, doctorId) -> List[Appointment]:
        pass

    @abstractmethod
    def scheduleAppointment(self, appointment) -> bool:
        pass

    @abstractmethod
    def updateAppointment(self, appointment) -> bool:
        pass

    @abstractmethod
    def cancelAppointment(self, appointmentId) -> bool:
        pass
```

**Define HospitalServiceImpl class and implement all the methods IHospitalServiceImpl**

**dao/HospitalServiceImpl.py**

```
from dao.i_hospital_service import IHospitalService
from entity.appointment import Appointment
from util.db_connection import DBConnection
from exception.PatientNumberNotFound import PatientNumberNotFoundException
from  tabulate import tabulate

class HospitalServiceImpl(IHospitalService):

    def getAppointmentById(self, appointmentId):
        conn = DBConnection.getConnection()
        cursor=conn.cursor()
        try:
            query = "select * from Appointment where appointmentId = ?"
            cursor.execute(query, (appointmentId,))
            appointment = cursor.fetchone()

            if appointment:
                appointment_details=[
                    ['Appointment ID',appointment[0]],
                    ["Patient ID",appointment[1]],
                    ["Doctor ID",appointment[2]],
                    ["Appointment Date",appointment[3]],
                    ["Description",appointment[4]],
                ]
                print("Appointment Details")
                print(tabulate(appointment_details,tablefmt="grid"))

            else:
                print("Appointment Not Found")
        except Exception as e:
            print(f"Error in fetching appointment: {e}")
            #return None
        finally:
            cursor.close()


    def getAppointmentsForPatient(self, patientId):
        conn = DBConnection.getConnection()
        cursor=conn.cursor()
        try:
            patient_check_query = "select count(*) from Patient where 
patientId = ?"
            cursor.execute(patient_check_query, (patientId,))
            patient_exists = cursor.fetchone()[0]

            if not patient_exists:
                raise PatientNumberNotFoundException(patientId)
            query = "select * from Appointment where patientId = ?"
            cursor.execute(query,(patientId,))
            appointments = []
            for row in cursor.fetchall():
                appointments.append(Appointment(
                    appointmentId=row[0],
                    patientId=row[1],
                    doctorId=row[2],
                    appointmentDate=row[3],
                    description=row[4]
                ))

            return appointments
        finally:
            cursor.close()

    def getAppointmentsForDoctor(self, doctorId):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            doctor_check_query = "select count(*) from Doctor where doctorId =
?"
            cursor.execute(doctor_check_query, (doctorId,))
            doctor_exists = cursor.fetchone()[0]

            if not doctor_exists:
                return None
            query = "select * from Appointment where doctorId = ?"
            cursor.execute(query, (doctorId,))
            doctors_appointments = []
            for result in cursor.fetchall():
                doctors_appointments.append(Appointment(
                    appointmentId=result[0],
                    patientId=result[1],
                    doctorId=result[2],
                    appointmentDate=result[3],
                    description=result[4]
                ))
            return doctors_appointments
        finally:
            cursor.close()


    def scheduleAppointment(self, appointment,appointment_id):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            check_query = "select count(*) from Appointment where
appointmentId = ?"
            cursor.execute(check_query, (appointment_id,))
            exists = cursor.fetchone()[0] > 0

            if exists:
                print("The appointment is full.")
                return
            query = """insert into Appointment(appointmentId, patientId,
doctorId, appointmentDate, description)
                   values (?, ?, ?, ?, ?)"""
            cursor.execute(query, (appointment.get_appointmentId(),
appointment.get_patientId(), appointment.get_doctorId(),
                                   appointment.get_appointmentDate(),
appointment.get_description()))
            conn.commit()
            print('Appointment scheduled')
            return True

        except Exception as e:
            print(f"Error scheduling appointment: {e}")
        finally:
            cursor.close()


    def updateAppointment(self, appointment):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            query = """update Appointment set patientId = ?, doctorId = ?, 
appointmentDate = ?, description = ?
                       where appointmentId = ?"""
            cursor.execute(query, (appointment.get_patientId(), 
appointment.get_doctorId(), appointment.get_appointmentDate(),
                                   appointment.get_description(), 
appointment.appointmentId))
            conn.commit()
            return True
        except Exception as e:
            print(f"Error in updating an appointment: {e}")
            return False
        finally:
            cursor.close()


    def cancelAppointment(self, appointmentId):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            cursor.execute("select count(*) from Appointment where 
appointmentId=?",(appointmentId,))
            count = cursor.fetchone()[0]
            if count==0:
                print("Appointment Not Found")
                return False
            query = "delete from Appointment where appointmentId = ?"
            cursor.execute(query, appointmentId)
            conn.commit()
            return True
        except Exception as e:
            print(f"Error in cancelling an appointment: {e}")
            return False
        finally:
            cursor.close()
```

**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**

**util/db_connection.py**

```python
import pyodbc
from util.property_util import PropertyUtil

class DBConnection:

    @staticmethod
    def getConnection():
        try:
            properties=PropertyUtil.getPropertyString()
            connection=pyodbc.connect(**properties)
            return connection
        except Exception as e:
            print(str(e) + '--Database is not connected--')
            return None
```

**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**

**util/properties.txt**

```
driver = {SQL Server}

server = LAPTOP-Q72Q77L5\SQLEXPRESS

database = Hospital_Management

trusted_connection = yes
```

**util/property_util.py**

```python
class PropertyUtil:
    @staticmethod
    def
getPropertyString(property_file_path=r"C:\Users\Asus\OneDrive\Desktop\Hospital
Management\util\properties.txt"):
        try:
            with open(property_file_path, 'r') as file:
                properties = {}
                for line in file:
                    key, value = line.strip().split('=')
                    properties[key.strip()] = value.strip()
                return properties
        except Exception as e:
            print(f"Error reading property file: {e}")
            return None
```

**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**

**exception/PatientNumberNotFound.py**

```
class PatientNumberNotFoundException(Exception):

    def __init__(self, patientId):
        super().__init__(f'Patient with ID {patientId} not found')
```

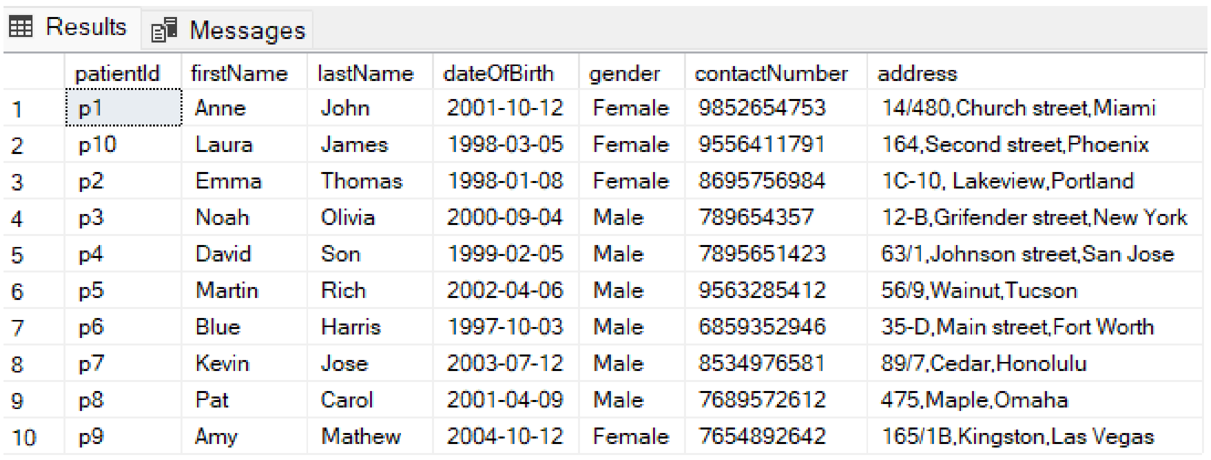
**Create class named MainModule with main method in package mainmodule. Trigger all the methods in service implementation class.**

**main/main_module.py**

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)
from dao.HospitalServiceImpl import HospitalServiceImpl
from entity.appointment import Appointment
from exception.PatientNumberNotFound import PatientNumberNotFoundException
from  tabulate import tabulate

class MainModule:
    def __init__(self):
        self.hospital_service = HospitalServiceImpl()

    def proceed(self):
        while True:
            #self.services()
            data=[
                ["1", "Get Appointment by ID"],
                ["2", "Get Appointments for Patient"],
                ["3", "Get Appointments for Doctor"],
                ["4", "Schedule an Appointment"],
                ["5", "Update an Appointment"],
                ["6", "Cancel an Appointment"],
                ["7", "Exit"]
            ]
            headers=["Option", "Service"]
            print("------Hospital Management System------")
            print(tabulate(data,headers, tablefmt="grid"))
            choice = input("Enter the option from 1 to 7: ")
            if choice == '1':
                self.getAppointmentById()
            elif choice == '2':
                self.getAppointmentsForPatient()
            elif choice == '3':
                self.getAppointmentsForDoctor()
            elif choice == '4':
                self.scheduleAppointment()
            elif choice == '5':
                self.updateAppointment()
            elif choice == '6':
                self.cancelAppointment()
            elif choice == '7':
                print("Exiting...")
                break
            else:
                print("Invalid choice. Please try again...")

    def getAppointmentById(self):
        appointment_id = input("Enter appointment ID: ")
        try:
            int_appointment_id=int(appointment_id)
            appointment = 
self.hospital_service.getAppointmentById(int_appointment_id)
            print(appointment)
        except ValueError as ve:
            print(f"Input type error: Please enter a valid integer for the 
appointment ID.{ve}")
        except Exception as e:
            print(e)

    def getAppointmentsForPatient(self):
        patient_id = input("Enter patient ID: ")
        try:
            appointments = 
self.hospital_service.getAppointmentsForPatient(patient_id)

            if appointments:
                print(f"Appointments for Patient: {patient_id}")
                rows = [[appointment.appointmentId, appointment.doctorId,
appointment.appointmentDate,appointment.description]
                              for appointment in appointments]
                headers = ["Appointment Id", "Doctor Id","Appointment Date",
"Appointment Description"]
                print(tabulate(rows, headers=headers,tablefmt="grid"))
            else:
                print(f'Patient with ID {patient_id} have no appointment')
        except PatientNumberNotFoundException as e:
            print("Exception:",e)

    def getAppointmentsForDoctor(self):
        doctor_id = input("Enter doctor ID: ")
        try:
            appointments =
self.hospital_service.getAppointmentsForDoctor(doctor_id)
            if appointments is None:
                print(f"The doctor ID {doctor_id} does not exist")
            elif appointments:
                print(f"Appointments for Doctor: {doctor_id}")
                table_data = [[appointment.appointmentId,
appointment.patientId, appointment.appointmentDate,appointment.description]
                                  for appointment in appointments]
                headers=["Appointment Id", "Patient Id", "Appointment Date",
"Appointment Description"]
                print(tabulate(table_data, headers=headers,tablefmt="grid"))

            else:
                print(f'Doctor with ID {doctor_id} have no appointments')
        except Exception as e:
            print("Error in fetching details of doctors appointment", e)

    def scheduleAppointment(self):
        appointment_id=int(input('Appointment ID:'))
        patient_id = input("Enter patient ID: ")
        doctor_id = input("Enter doctor ID: ")
        appointment_date = input("Enter appointment date (YYYY-MM-DD): ")
        description = input("Enter appointment description: ")

        appointment = Appointment(
            appointmentId = appointment_id,
            patientId = patient_id,
            doctorId = doctor_id,
            appointmentDate = appointment_date,
            description = description
        )
        success = 
self.hospital_service.scheduleAppointment(appointment,appointment_id)
        if success:
            print()
        else:
            print("Failed to schedule appointment.")

    def updateAppointment(self):
        appointment_id = input("Enter appointment ID: ")
        new_patient_id = input("Enter new patient ID: ")
        new_doctor_id = input("Enter new doctor ID: ")
        new_appointment_date = input("Enter new appointment date (YYYY-MM-DD): 
")
        new_description = input("Enter new appointment description: ")

        appointment = Appointment(
            appointmentId = appointment_id,
            patientId = new_patient_id,
            doctorId = new_doctor_id,
            appointmentDate = new_appointment_date,
            description = new_description
        )

        update = self.hospital_service.updateAppointment(appointment)
        if update:
            print("Appointment updated successfully.")
        else:
            print("Failed to update appointment.")

    def cancelAppointment(self):
        appointment_id = int(input("Enter appointment ID to cancel: "))
        cancel = self.hospital_service.cancelAppointment(appointment_id)
        if cancel:
            print("Appointment cancelled successfully.")
        else:
                print("Failed to cancel appointment.")



if __name__ == "__main__":
    main_module = MainModule()
    main_module.proceed()
```

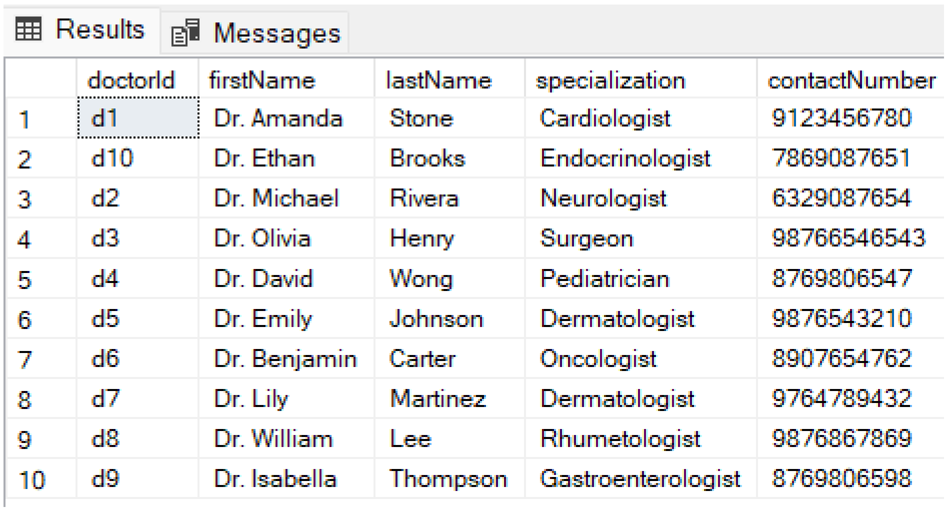
**Outputs of database:**

```
select * from Patient;
```



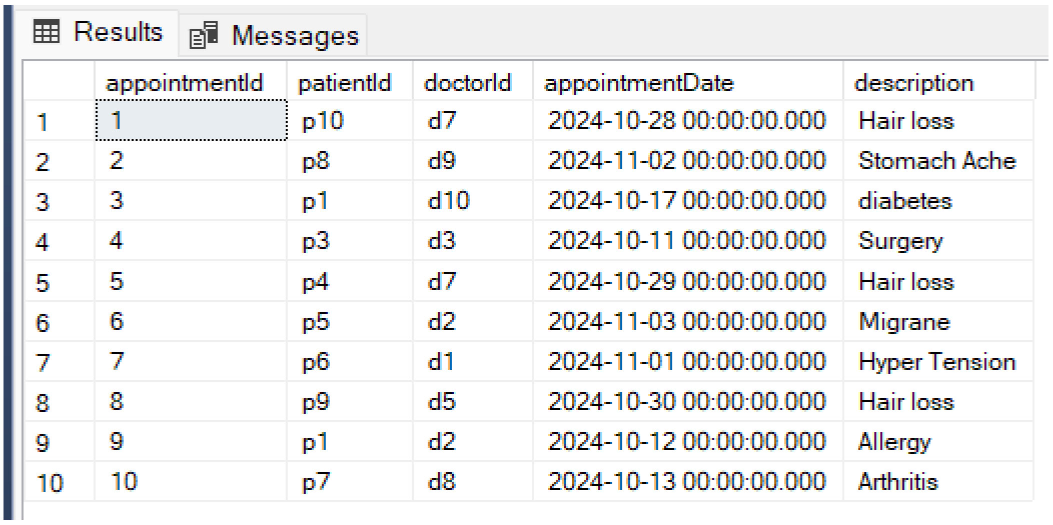
| | patientId | firstName | lastName | dateOfBirth | gender | contactNumber | address |
|---|---|---|---|---|---|---|---|
| 1 | p1 | Anne | John | 2001-10-12 | Female | 9852654753 | 14/480,Church street,Miami |
| 2 | p10 | Laura | James | 1998-03-05 | Female | 9556411791 | 164,Second street,Phoenix |
| 3 | p2 | Emma | Thomas | 1998-01-08 | Female | 8695756984 | 1C-10, Lakeview,Portland |
| 4 | p3 | Noah | Olivia | 2000-09-04 | Male | 789654357 | 12-B,Grifender street,New York |
| 5 | p4 | David | Son | 1999-02-05 | Male | 7895651423 | 63/1,Johnson street,San Jose |
| 6 | p5 | Martin | Rich | 2002-04-06 | Male | 9563285412 | 56/9,Wainut,Tucson |
| 7 | p6 | Blue | Harris | 1997-10-03 | Male | 6859352946 | 35-D,Main street,Fort Worth |
| 8 | p7 | Kevin | Jose | 2003-07-12 | Male | 8534976581 | 89/7,Cedar,Honolulu |
| 9 | p8 | Pat | Carol | 2001-04-09 | Male | 7689572612 | 475,Maple,Omaha |
| 10 | p9 | Amy | Mathew | 2004-10-12 | Female | 7654892642 | 165/1B,Kingston,Las Vegas |

```
select * from Doctor;
```



| | doctorId | firstName | lastName | specialization | contactNumber |
|---|---|---|---|---|---|
| 1 | d1 | Dr. Amanda | Stone | Cardiologist | 9123456780 |
| 2 | d10 | Dr. Ethan | Brooks | Endocrinologist | 7869087651 |
| 3 | d2 | Dr. Michael | Rivera | Neurologist | 6329087654 |
| 4 | d3 | Dr. Olivia | Henry | Surgeon | 98766546543 |
| 5 | d4 | Dr. David | Wong | Pediatrician | 8769806547 |
| 6 | d5 | Dr. Emily | Johnson | Dermatologist | 9876543210 |
| 7 | d6 | Dr. Benjamin | Carter | Oncologist | 8907654762 |
| 8 | d7 | Dr. Lily | Martinez | Dermatologist | 9764789432 |
| 9 | d8 | Dr. William | Lee | Rhumetologist | 9876867869 |
| 10 | d9 | Dr. Isabella | Thompson | Gastroenterologist | 8769806598 |

```
select * from Appointment;
```

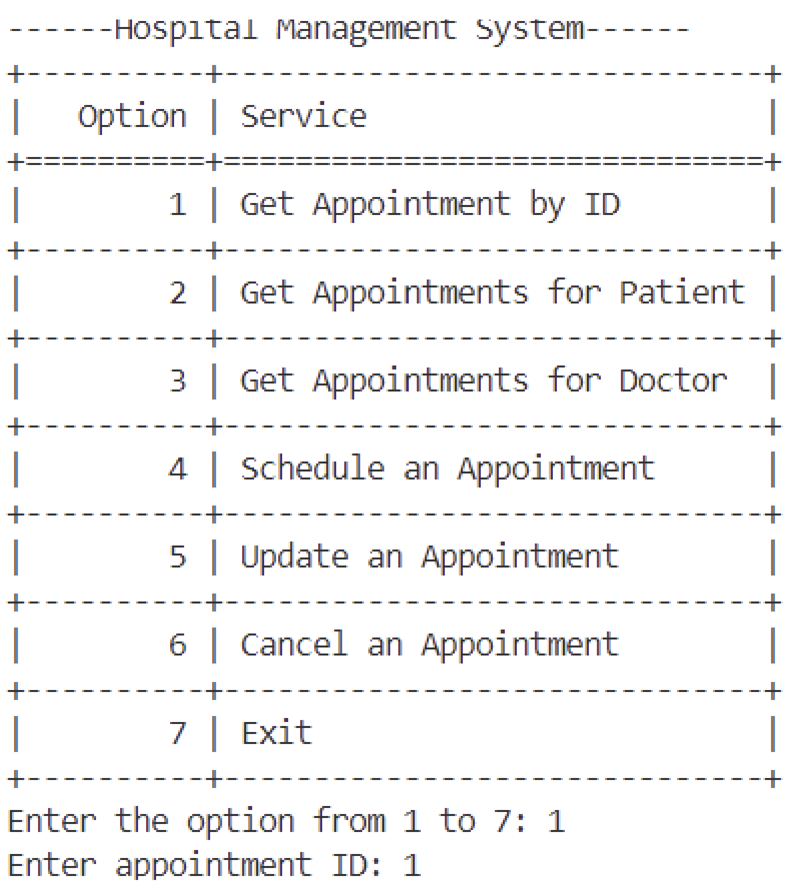

Results | Messages

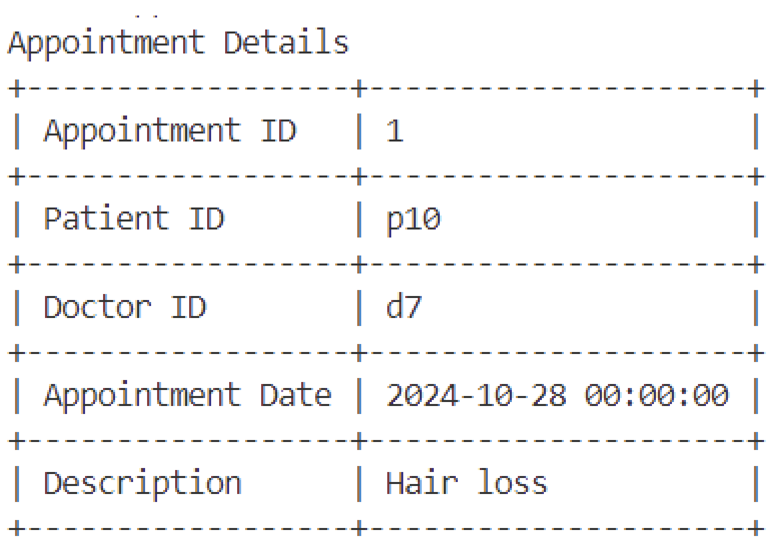
| | appointmentId | patientId | doctorId | appointmentDate | description |
|---|---|---|---|---|---|
| 1 | 1 | p10 | d7 | 2024-10-28 00:00:00.000 | Hair loss |
| 2 | 2 | p8 | d9 | 2024-11-02 00:00:00.000 | Stomach Ache |
| 3 | 3 | p1 | d10 | 2024-10-17 00:00:00.000 | diabetes |
| 4 | 4 | p3 | d3 | 2024-10-11 00:00:00.000 | Surgery |
| 5 | 5 | p4 | d7 | 2024-10-29 00:00:00.000 | Hair loss |
| 6 | 6 | p5 | d2 | 2024-11-03 00:00:00.000 | Migrane |
| 7 | 7 | p6 | d1 | 2024-11-01 00:00:00.000 | Hyper Tension |
| 8 | 8 | p9 | d5 | 2024-10-30 00:00:00.000 | Hair loss |
| 9 | 9 | p1 | d2 | 2024-10-12 00:00:00.000 | Allergy |
| 10 | 10 | p7 | d8 | 2024-10-13 00:00:00.000 | Arthritis |

Outputs from repository:

**OPTION 1:**

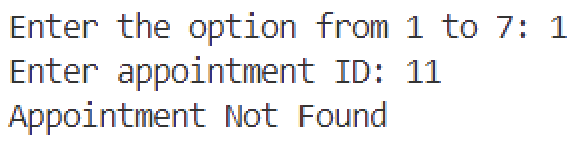
When appointment is present in db:


```
------Hospital Management System------
+----------+----------------------------+
|   Option | Service                    |
+==========+============================+
|        1 | Get Appointment by ID      |
+----------+----------------------------+
|        2 | Get Appointments for Patient |
+----------+----------------------------+
|        3 | Get Appointments for Doctor |
+----------+----------------------------+
|        4 | Schedule an Appointment    |
+----------+----------------------------+
|        5 | Update an Appointment      |
+----------+----------------------------+
|        6 | Cancel an Appointment      |
+----------+----------------------------+
|        7 | Exit                       |
+----------+----------------------------+
Enter the option from 1 to 7: 1
Enter appointment ID: 1
```



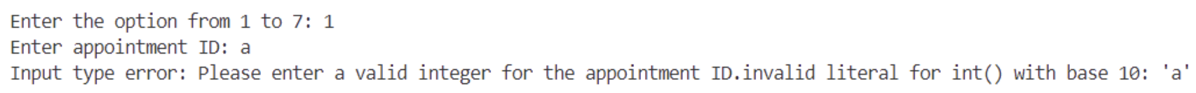
```
Appointment Details
+------------------+---------------------+
| Appointment ID   | 1                   |
+------------------+---------------------+
| Patient ID       | p10                 |
+------------------+---------------------+
| Doctor ID        | d7                  |
+------------------+---------------------+
| Appointment Date | 2024-10-28 00:00:00 |
+------------------+---------------------+
| Description      | Hair loss           |
+------------------+---------------------+
```

When appointment does not exists in db:



```
Enter the option from 1 to 7: 1
Enter appointment ID: 11
Appointment Not Found
```

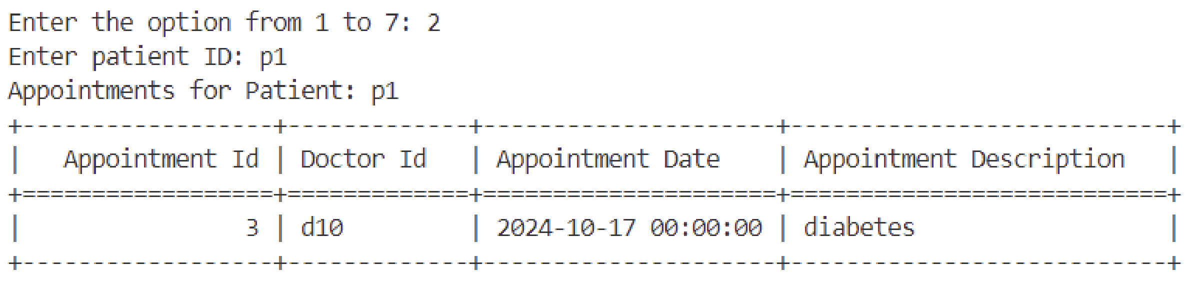
When input type has given incorrect:



```
Enter the option from 1 to 7: 1
Enter appointment ID: a
Input type error: Please enter a valid integer for the appointment ID.invalid literal for int() with base 10: 'a'
```

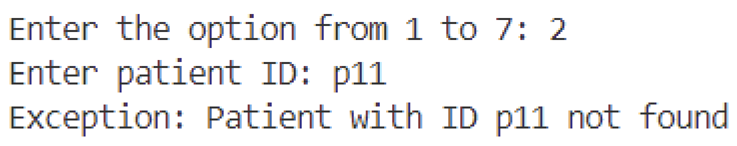
**OPTION 2:**

When patient id is present in db:



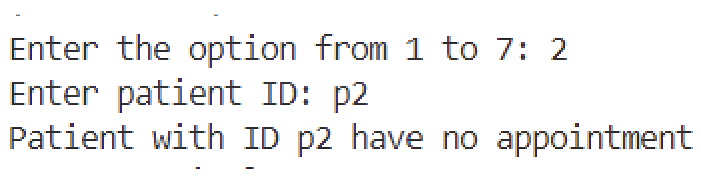
```
Enter the option from 1 to 7: 2
Enter patient ID: p1
Appointments for Patient: p1
+------------------+-------------+---------------------+--------------------------+
|   Appointment Id | Doctor Id   | Appointment Date    | Appointment Description  |
+==================+=============+=====================+==========================+
|                3 | d10         | 2024-10-17 00:00:00 | diabetes                 |
+------------------+-------------+---------------------+--------------------------+
```

Exception handling:



```
Enter the option from 1 to 7: 2
Enter patient ID: p11
Exception: Patient with ID p11 not found
```

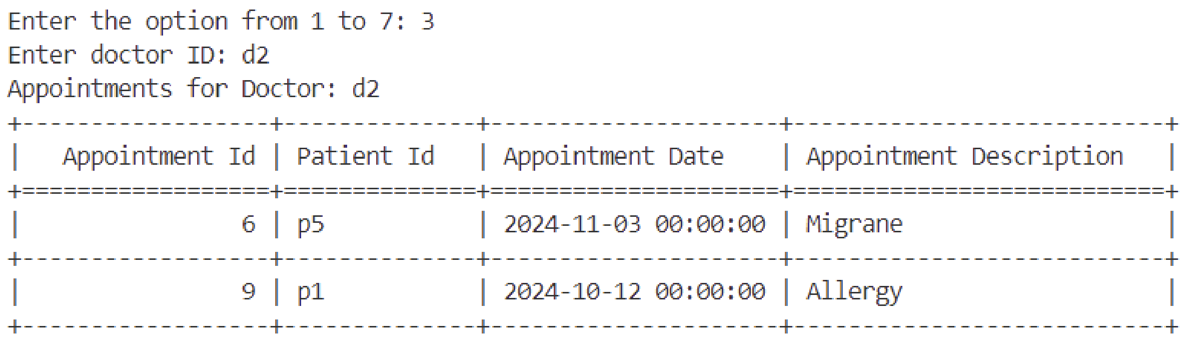
When patient does not have any appointment:



```
Enter the option from 1 to 7: 2
Enter patient ID: p2
Patient with ID p2 have no appointment
```

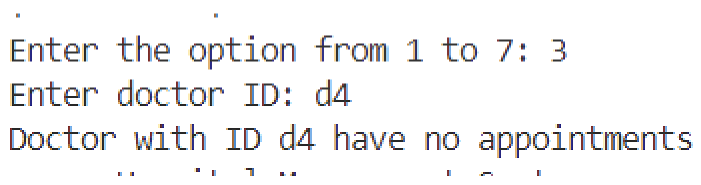
**OPTION 3:**

When doctor have appointments to check:



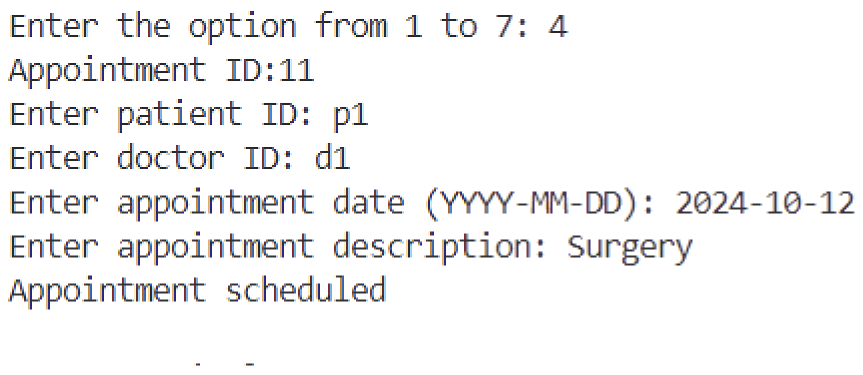
```
Enter the option from 1 to 7: 3
Enter doctor ID: d2
Appointments for Doctor: d2
+-----------------+-------------+--------------------+-------------------------+
|  Appointment Id | Patient Id  | Appointment Date   | Appointment Description |
+=================+=============+====================+=========================+
|               6 | p5          | 2024-11-03 00:00:00 | Migrane                |
+-----------------+-------------+--------------------+-------------------------+
|               9 | p1          | 2024-10-12 00:00:00 | Allergy                |
+-----------------+-------------+--------------------+-------------------------+
```

When doctor have no appointments to check:

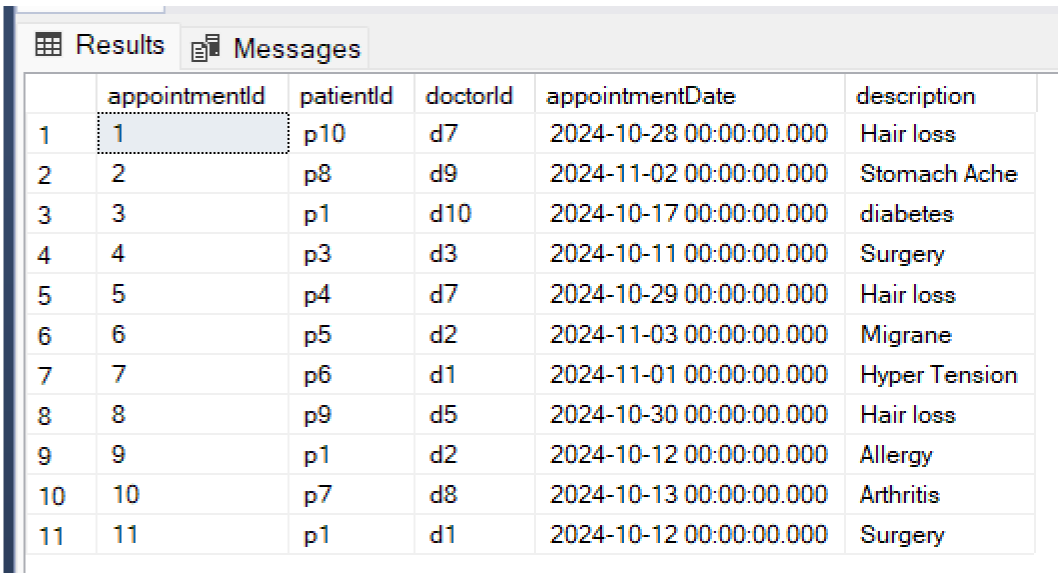


```
Enter the option from 1 to 7: 3
Enter doctor ID: d4
Doctor with ID d4 have no appointments
```

**OPTION 4:**

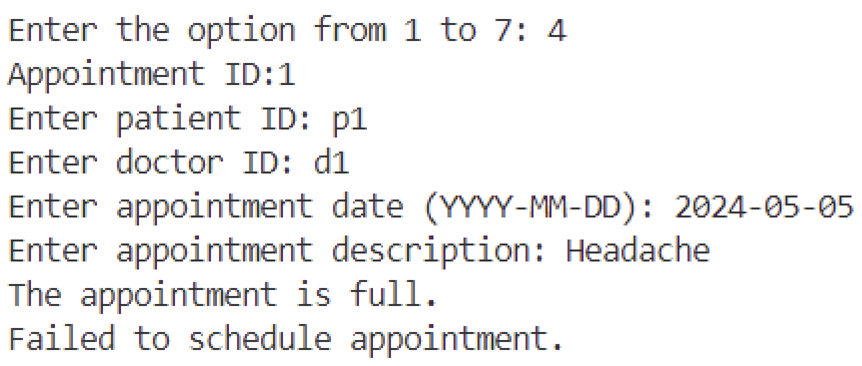


```
Enter the option from 1 to 7: 4
Appointment ID:11
Enter patient ID: p1
Enter doctor ID: d1
Enter appointment date (YYYY-MM-DD): 2024-10-12
Enter appointment description: Surgery
Appointment scheduled
```


Results
Messages

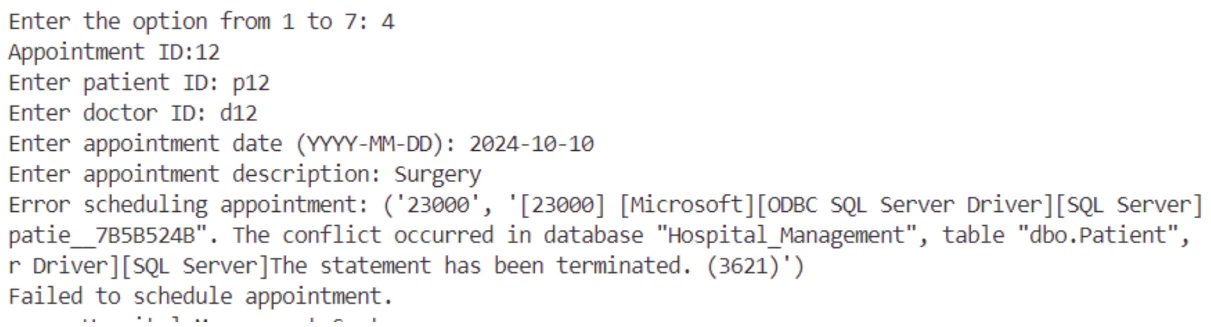
| | appointmentId | patientId | doctorId | appointmentDate | description |
|---|---|---|---|---|---|
| 1 | 1 | p10 | d7 | 2024-10-28 00:00:00.000 | Hair loss |
| 2 | 2 | p8 | d9 | 2024-11-02 00:00:00.000 | Stomach Ache |
| 3 | 3 | p1 | d10 | 2024-10-17 00:00:00.000 | diabetes |
| 4 | 4 | p3 | d3 | 2024-10-11 00:00:00.000 | Surgery |
| 5 | 5 | p4 | d7 | 2024-10-29 00:00:00.000 | Hair loss |
| 6 | 6 | p5 | d2 | 2024-11-03 00:00:00.000 | Migrane |
| 7 | 7 | p6 | d1 | 2024-11-01 00:00:00.000 | Hyper Tension |
| 8 | 8 | p9 | d5 | 2024-10-30 00:00:00.000 | Hair loss |
| 9 | 9 | p1 | d2 | 2024-10-12 00:00:00.000 | Allergy |
| 10 | 10 | p7 | d8 | 2024-10-13 00:00:00.000 | Arthritis |
| 11 | 11 | p1 | d1 | 2024-10-12 00:00:00.000 | Surgery |

When appointment is already exists:



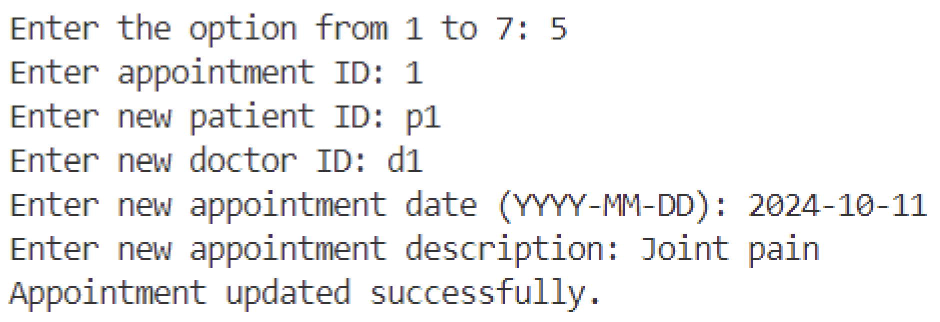
```
Enter the option from 1 to 7: 4
Appointment ID:1
Enter patient ID: p1
Enter doctor ID: d1
Enter appointment date (YYYY-MM-DD): 2024-05-05
Enter appointment description: Headache
The appointment is full.
Failed to schedule appointment.
```

Exception:

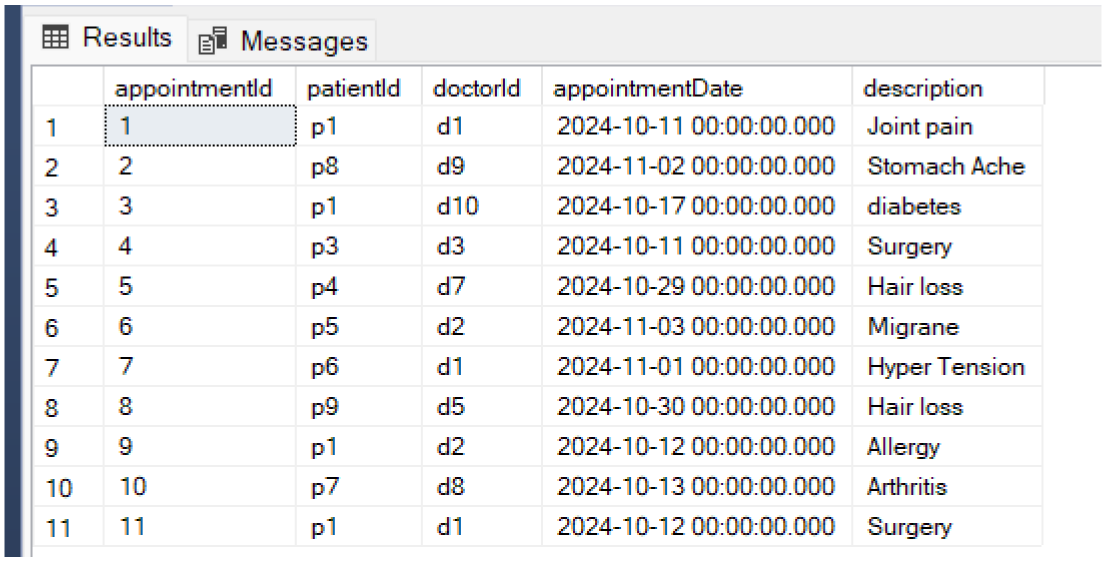


```
Enter the option from 1 to 7: 4
Appointment ID:12
Enter patient ID: p12
Enter doctor ID: d12
Enter appointment date (YYYY-MM-DD): 2024-10-10
Enter appointment description: Surgery
Error scheduling appointment: ('23000', '[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]
patie__7B5B524B". The conflict occurred in database "Hospital_Management", table "dbo.Patient",
r Driver][SQL Server]The statement has been terminated. (3621)')
Failed to schedule appointment.
```

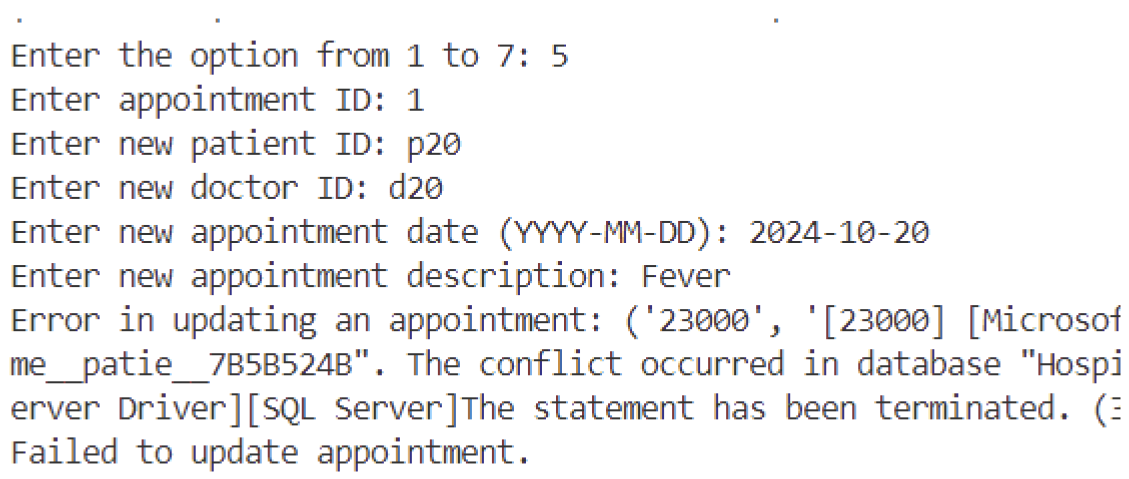
**OPTION 5:**



```
Enter the option from 1 to 7: 5
Enter appointment ID: 1
Enter new patient ID: p1
Enter new doctor ID: d1
Enter new appointment date (YYYY-MM-DD): 2024-10-11
Enter new appointment description: Joint pain
Appointment updated successfully.
```

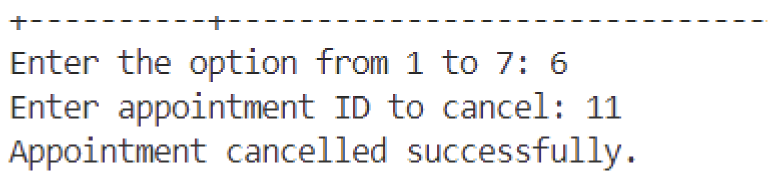


|  | appointmentId | patientId | doctorId | appointmentDate | description |
|---|---|---|---|---|---|
| 1 | 1 | p1 | d1 | 2024-10-11 00:00:00.000 | Joint pain |
| 2 | 2 | p8 | d9 | 2024-11-02 00:00:00.000 | Stomach Ache |
| 3 | 3 | p1 | d10 | 2024-10-17 00:00:00.000 | diabetes |
| 4 | 4 | p3 | d3 | 2024-10-11 00:00:00.000 | Surgery |
| 5 | 5 | p4 | d7 | 2024-10-29 00:00:00.000 | Hair loss |
| 6 | 6 | p5 | d2 | 2024-11-03 00:00:00.000 | Migrane |
| 7 | 7 | p6 | d1 | 2024-11-01 00:00:00.000 | Hyper Tension |
| 8 | 8 | p9 | d5 | 2024-10-30 00:00:00.000 | Hair loss |
| 9 | 9 | p1 | d2 | 2024-10-12 00:00:00.000 | Allergy |
| 10 | 10 | p7 | d8 | 2024-10-13 00:00:00.000 | Arthritis |
| 11 | 11 | p1 | d1 | 2024-10-12 00:00:00.000 | Surgery |

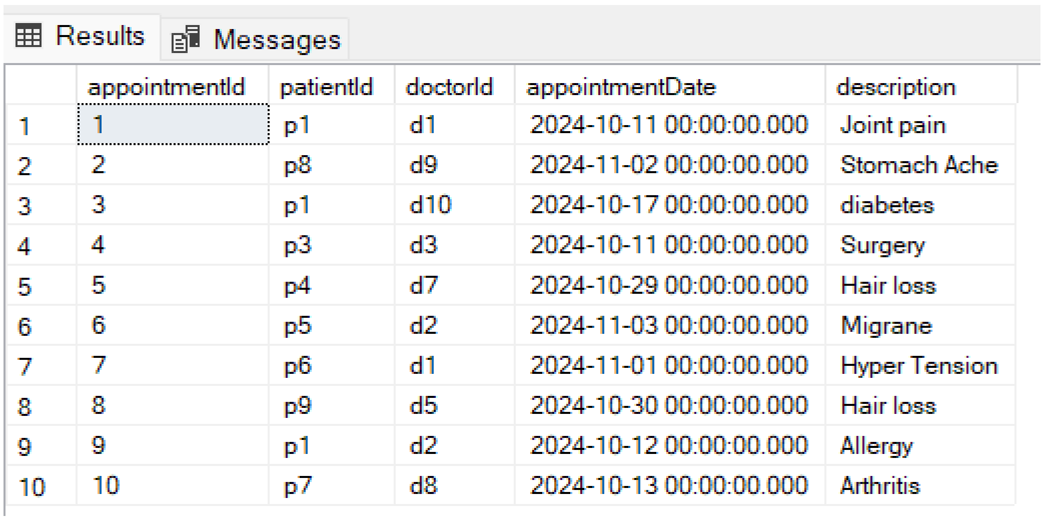


```
Enter the option from 1 to 7: 5
Enter appointment ID: 1
Enter new patient ID: p20
Enter new doctor ID: d20
Enter new appointment date (YYYY-MM-DD): 2024-10-20
Enter new appointment description: Fever
Error in updating an appointment: ('23000', '[23000] [Microsof
me__patie__7B5B524B". The conflict occurred in database "Hospi
erver Driver][SQL Server]The statement has been terminated. (
Failed to update appointment.
```

OPTION 6:

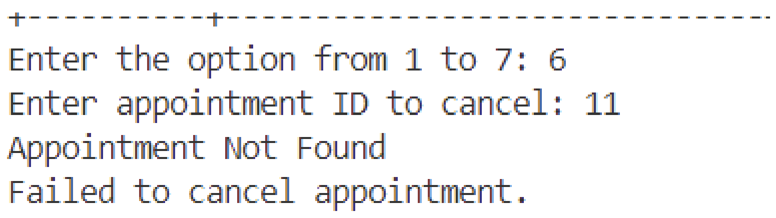


```
+----------+------------------------------
Enter the option from 1 to 7: 6
Enter appointment ID to cancel: 11
Appointment cancelled successfully.
```

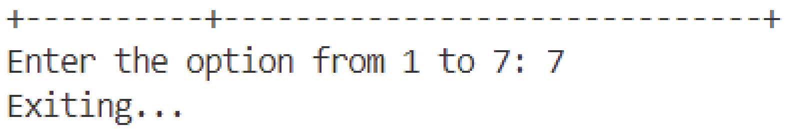


Results | Messages

| | appointmentId | patientId | doctorId | appointmentDate | description |
|---|---|---|---|---|---|
| 1 | 1 | p1 | d1 | 2024-10-11 00:00:00.000 | Joint pain |
| 2 | 2 | p8 | d9 | 2024-11-02 00:00:00.000 | Stomach Ache |
| 3 | 3 | p1 | d10 | 2024-10-17 00:00:00.000 | diabetes |
| 4 | 4 | p3 | d3 | 2024-10-11 00:00:00.000 | Surgery |
| 5 | 5 | p4 | d7 | 2024-10-29 00:00:00.000 | Hair loss |
| 6 | 6 | p5 | d2 | 2024-11-03 00:00:00.000 | Migrane |
| 7 | 7 | p6 | d1 | 2024-11-01 00:00:00.000 | Hyper Tension |
| 8 | 8 | p9 | d5 | 2024-10-30 00:00:00.000 | Hair loss |
| 9 | 9 | p1 | d2 | 2024-10-12 00:00:00.000 | Allergy |
| 10 | 10 | p7 | d8 | 2024-10-13 00:00:00.000 | Arthritis |

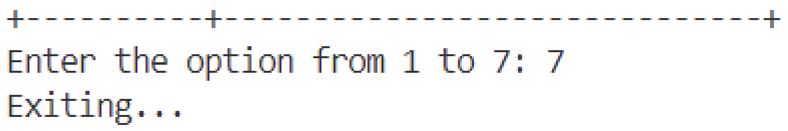


```
+----------+-----------------------------
Enter the option from 1 to 7: 6
Enter appointment ID to cancel: 11
Appointment Not Found
Failed to cancel appointment.
```



```
+----------+----------------------------+
Enter the option from 1 to 7: 7
Exiting...
```

**OPTION 7:**



```
+----------+----------------------------+
Enter the option from 1 to 7: 7
Exiting...
```