CODING CHALLENGE
Name: C.Sai Anand
Topic: Insurance Management System

Problem Statement:

1: Create a SQL schema from the following classes; use the class attributes for table column names.

```sql
CREATE DATABASE InsuranceManageDB;
USE InsuranceManageDB;
GO

-- Create Users table
CREATE TABLE Users (
   userId INT PRIMARY KEY identity(101,1),
   username NVARCHAR(50) NOT NULL,
   password NVARCHAR(50) NOT NULL,
   role NVARCHAR(50) NOT NULL
);

INSERT INTO Users (username, password, role) VALUES
('megha', 'meghaSecure01', 'admin'),
('rahul', 'rahulAgent99', 'agent'),
('kavita', 'kavita@2024', 'customer'),
('arjun', 'arjun$789', 'customer');

-- Create Clients table
CREATE TABLE Clients (
   clientId INT PRIMARY KEY identity(201,1),
   clientName NVARCHAR(100) NOT NULL,
   contactInfo NVARCHAR(100) NOT NULL,
   policy NVARCHAR(100) NOT NULL
);


INSERT INTO Clients (clientName, contactInfo, policy) VALUES
('Amit Sharma', 'amit.sharma@email.com', 'Life Insurance'),
('Priya Kapoor', 'priya.kapoor@email.com', 'Auto Insurance'),
('Ravi Iyer', '+91 9876543210', 'Health Insurance'),
('Sunita Patel', 'sunita.patel@email.com', 'Home Insurance');

-- Create Claims table
```

```sql
CREATE TABLE Claims (
   claimId INT PRIMARY KEY identity(301,1),
   claimNumber NVARCHAR(100) NOT NULL,
   dateFiled DATE NOT NULL,
   claimAmount DECIMAL(10, 2) NOT NULL,
   status NVARCHAR(50) NOT NULL,
   clientId INT,
   policy NVARCHAR(100),
   FOREIGN KEY (clientId) REFERENCES Clients(clientId)
);

INSERT INTO Claims (claimNumber, dateFiled, claimAmount, status, policy, clientId)
VALUES
('CLM2024005', '2024-10-15', 1800.00, 'Approved', 'Health Insurance', 201),
('CLM2024006', '2024-10-18', 3200.00, 'Pending', 'Vehicle Insurance', 202),
('CLM2024007', '2024-10-20', 500.00, 'Rejected', 'Home Insurance', 203),
('CLM2024008', '2024-10-25', 3900.00, 'Approved', 'Life Insurance', 204);

-- Create Payments table
CREATE TABLE Payments (
   paymentId INT PRIMARY KEY identity(401,1),
   paymentDate DATE NOT NULL,
   paymentAmount DECIMAL(10, 2) NOT NULL,
   clientId INT,
   FOREIGN KEY (clientId) REFERENCES Clients(clientId)
);


INSERT INTO Payments (paymentDate, paymentAmount, clientId) VALUES
('2024-10-05', 1800.00, 201),  -- Payment for Amit Sharma
('2024-10-06', 2700.00, 202),  -- Payment for Priya Kapoor
('2024-10-07', 400.00, 203),   -- Payment for Ravi Iyer
('2024-10-08', 4500.00, 204);  -- Payment for Sunita Patel


-- Create Policies table
CREATE TABLE Policies (
   policyId INT PRIMARY KEY IDENTITY(1,1),
   policyName NVARCHAR(100) NOT NULL,
   policyDescription NVARCHAR(255) NOT NULL
);
```

```
-- Insert into Policies
INSERT INTO Policies (policyName, policyDescription)
VALUES ('Health Insurance', 'Offers coverage for medical costs, hospitalizations, and
surgeries'),
      ('Life Insurance', 'Ensures financial protection for your family in case of
death'),
      ('Vehicle Insurance', 'Provides coverage for vehicle damages and third-party
liabilities'),
      ('Home Insurance', 'Protects your home from damages due to accidents or
disasters');

SELECT * FROM Clients;
SELECT * FROM Users;
SELECT * FROM Claims;
SELECT * FROM Payments
SELECT * FROM Policies;
```

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, and a method to print all the member variables and values.

1. Define the User' class with the following confidential attributes:
a. userId;
b. username;
c. password;
d. Role;

Code for this part
entity/User.py

```
class User:
   def __init__(self, userId=None, username=None, password=None, role=None):
       self.__userId = userId
       self.__username = username
       self.__password = password
       self.__role = role

   # Getters and Setters functions
   def get_userId(self):
       return self.__userId

   def set_userId(self, userId):
       self.__userId = userId

   def get_username(self):
       return self.__username

   def set_username(self, username):
       self.__username = username

   def get_password(self):
       return self.__password

   def set_password(self, password):
       self.__password = password

   def get_role(self):
       return self.__role

   def set_role(self, role):
       self.__role = role


   def __str__(self):
       return f"User [userId={self.__userId}, username={self.__username},
role={self.__role}]"
```

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

a. clientid;

b. clientName;

c. contactinfo;

d. policy;//Represents the policy associated with the client

Code for this part

entity/Client.py

```python
class Client:
   def __init__(self, clientId=None, clientName=None, contactInfo=None, policy=None):
       self.__clientId = clientId
       self.__clientName = clientName
       self.__contactInfo = contactInfo
       self.__policy = policy

   # Getters and Setters functions
   def get_clientId(self):
       return self.__clientId

   def set_clientId(self, clientId):
       self.__clientId = clientId

   def get_clientName(self):
       return self.__clientName

   def set_clientName(self, clientName):
       self.__clientName = clientName

   def get_contactInfo(self):
       return self.__contactInfo

   def set_contactInfo(self, contactInfo):
       self.__contactInfo = contactInfo

   def get_policy(self):
       return self.__policy

   def set_policy(self, policy):
       self.__policy = policy
```

```
    def __str__(self):
        return f"Client [clientId={self.__clientId}, clientName={self.__clientName},
contactInfo={self.__contactInfo}, policy={self.__policy}]"
```

3. Define ' Claim class with the following confidential attributes:

a. claimId;

b. claimNumber;

c. dateFiled;

d. claimAmount;

e. status;

f. policy;//Represents the policy associated with the claim

g. client; // Represents the client associated with the claim

Code for this

entity/Claim.py

```
# entity/Claim.py
class Claim:
    def __init__(self, claimId=None, claimNumber=None, dateFiled=None,
claimAmount=None, status=None, policy=None, clientId=None):
        self.__claimId = claimId
        self.__claimNumber = claimNumber
        self.__dateFiled = dateFiled
        self.__claimAmount = claimAmount
        self.__status = status
        self.__policy = policy
        self.__client = clientId

    # Getters and Setters functions
    def get_claimId(self):
        return self.__claimId

    def set_claimId(self, claimId):
        self.__claimId = claimId

    def get_claimNumber(self):
        return self.__claimNumber
```

```python
    def set_claimNumber(self, claimNumber):
        self.__claimNumber = claimNumber

    def get_dateFiled(self):
        return self.__dateFiled

    def set_dateFiled(self, dateFiled):
        self.__dateFiled = dateFiled

    def get_claimAmount(self):
        return self.__claimAmount

    def set_claimAmount(self, claimAmount):
        self.__claimAmount = claimAmount

    def get_status(self):
        return self.__status

    def set_status(self, status):
        self.__status = status

    def get_policy(self):
        return self.__policy

    def set_policy(self, policy):
        self.__policy = policy

    def get_client(self):
        return self.__client

    def set_client(self, clientId):
        self.__client = clientId

    # String representation
    def __str__(self):
        return (f"Claim [claimId={self.__claimId}, claimNumber={self.__claimNumber},
dateFiled={self.__dateFiled}, "
                f"claimAmount={self.__claimAmount}, status={self.__status},
policy={self.__policy}, client={self.__client}]")
```

4. Define ` payment ` class with the following confidential attributes:

a. paymentId;

b. paymentDate;

c. paymentAmount;

d. client;// Represents the client associated with the payment

Code for this

entity/Payment.py

```
# entity/Payment.py
class Payment:
   def __init__(self, paymentId=None, paymentDate=None, paymentAmount=None,
client=None):
       self.__paymentId = paymentId
       self.__paymentDate = paymentDate
       self.__paymentAmount = paymentAmount
       self.__client = client

   # Getters and Setters
   def get_paymentId(self):
       return self.__paymentId

   def set_paymentId(self, paymentId):
       self.__paymentId = paymentId

   def get_paymentDate(self):
       return self.__paymentDate

   def set_paymentDate(self, paymentDate):
       self.__paymentDate = paymentDate

   def get_paymentAmount(self):
       return self.__paymentAmount

   def set_paymentAmount(self, paymentAmount):
       self.__paymentAmount = paymentAmount

   def get_client(self):
       return self.__client

   def set_client(self, client):
       self.__client = client

   # String representation
   def __str__(self):
       return (f"Payment [paymentId={self.__paymentId},
paymentDate={self.__paymentDate}, "
               f"paymentAmount={self.__paymentAmount}, client={self.__client}]")
```

3. Define IPolicyService interface/abstract class with following methodsto interact with database

Keep the interfaces and implementation classes in package dao

a. createPolicy()

I. parameters: Policy Object

II. return type: Boolean

b. getPolicy()

I. parameters: policyId

II. return type: Policy Object

c. getAllPolicies()

I. parameters: none

II. return type: Collection of Policy Objects

d. updatePolicy()

I. parameters: Policy Object

II. return type: boolean

e. deletePolicy()

I. parameters: PolicyId

II. return type: boolean

Code for that

dao/IPolicyService.py

```
from abc import ABC, abstractmethod

class IPolicyService(ABC):
    @abstractmethod
    def createPolicy(self, policy):
        pass

    @abstractmethod
    def getPolicy(self, policyId):
        pass

    @abstractmethod
    def getAllPolicies(self):
        pass

    @abstractmethod
    def updatePolicy(self, policy):
        pass

    @abstractmethod
    def deletePolicy(self, policyId):
        pass
```

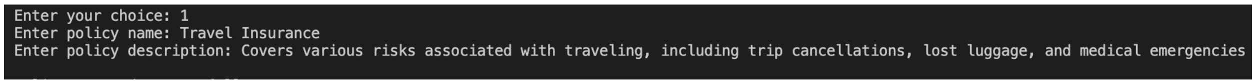
6. Define InsuranceServiceImpl class and implement all the methods InsuranceServiceImpl Code:

dao/PolicyServiceImpl.py

```
import pyodbc
from src.dao.IPolicyService import IPolicyService
from src.entity.Policy import Policy
from src.exception.PolicyNotFoundException import PolicyNotFoundException
from src.util.DBconnection import DBconnection

class PolicyServiceImpl(IPolicyService):
    def __init__(self):
        self.conn = DBconnection.get_connection()

    def createPolicy(self, policy):
        cursor = self.conn.cursor()
        query = "INSERT INTO Policies (policyName, policyDescription) VALUES (?, ?)"
        cursor.execute(query, policy.get_policyName(), policy.get_policyDescription())
        self.conn.commit()
        return True

    def getPolicy(self, policyId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Policies WHERE policyId = ?"
        cursor.execute(query, policyId)
        result = cursor.fetchone()
        if result:
            return Policy(policyId=result.policyId, policyName=result.policyName,
policyDescription=result.policyDescription)
        else:
            raise PolicyNotFoundException(f"Policy with ID {policyId} not found.")

    def getAllPolicies(self):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Policies"
        cursor.execute(query)
        policies = []
        for row in cursor.fetchall():
            policy = Policy(policyId=row.policyId, policyName=row.policyName,
policyDescription=row.policyDescription)
            policies.append(policy)
        return policies

    def updatePolicy(self, policy):
        cursor = self.conn.cursor()
        query = "UPDATE Policies SET policyName = ?, policyDescription = ? WHERE
policyId = ?"
        cursor.execute(query, policy.get_policyName(), policy.get_policyDescription(),
policy.get_policyId())
        self.conn.commit()
        return True

    def deletePolicy(self, policyId):
        cursor = self.conn.cursor()
        query = "DELETE FROM Policies WHERE policyId = ?"
        cursor.execute(query, policyId)
        self.conn.commit()
        return True
```

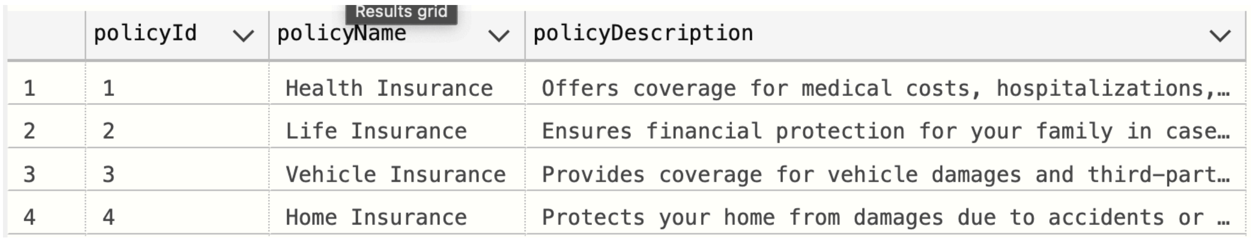
OUTPUTS:

1) Create Policy



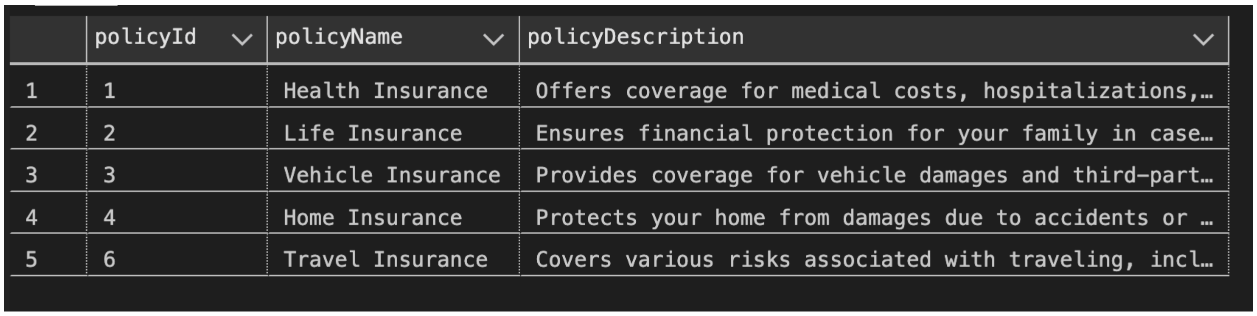
```
Enter your choice: 1
Enter policy name: Travel Insurance
Enter policy description: Covers various risks associated with traveling, including trip cancellations, lost luggage, and medical emergencies
```

DB Before



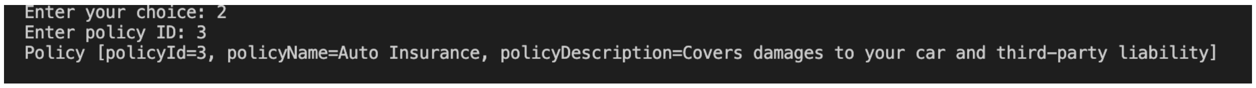
	policyId	policyName	policyDescription
1	1	Health Insurance	Offers coverage for medical costs, hospitalizations,…
2	2	Life Insurance	Ensures financial protection for your family in case…
3	3	Vehicle Insurance	Provides coverage for vehicle damages and third-part…
4	4	Home Insurance	Protects your home from damages due to accidents or …

DB After



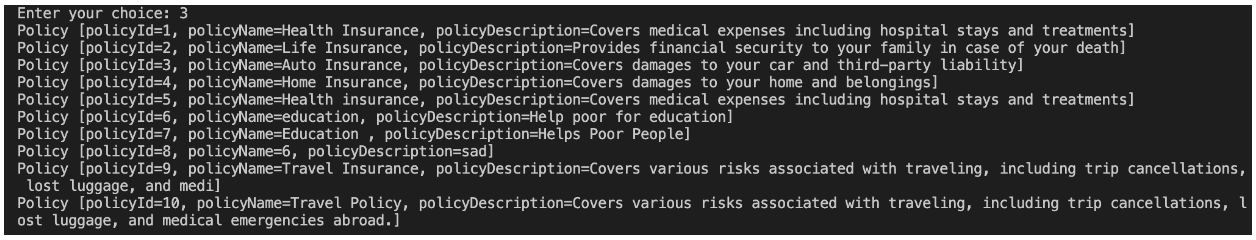
	policyId	policyName	policyDescription
1	1	Health Insurance	Offers coverage for medical costs, hospitalizations,…
2	2	Life Insurance	Ensures financial protection for your family in case…
3	3	Vehicle Insurance	Provides coverage for vehicle damages and third-part…
4	4	Home Insurance	Protects your home from damages due to accidents or …
5	6	Travel Insurance	Covers various risks associated with traveling, incl…

2) Get Policy ID



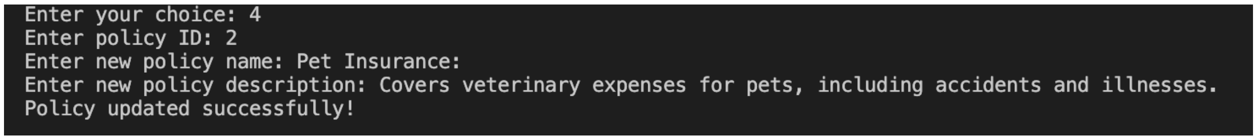
```
Enter your choice: 2
Enter policy ID: 3
Policy [policyId=3, policyName=Auto Insurance, policyDescription=Covers damages to your car and third-party liability]
```

3) Get All Policy Output



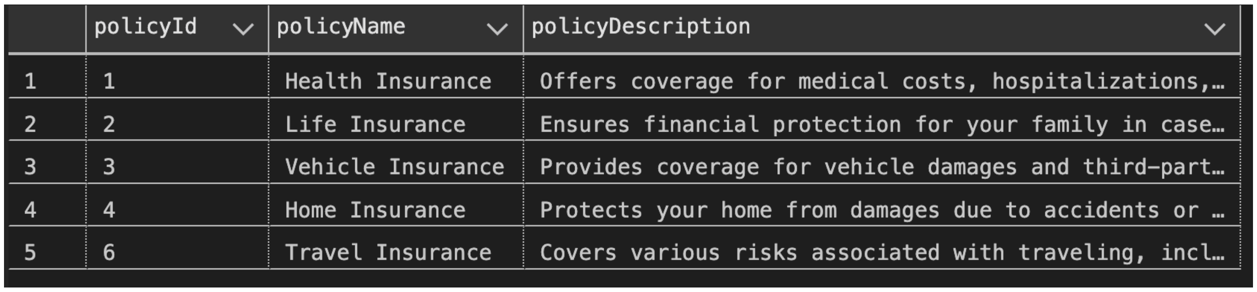
```
Enter your choice: 3
Policy [policyId=1, policyName=Health Insurance, policyDescription=Covers medical expenses including hospital stays and treatments]
Policy [policyId=2, policyName=Life Insurance, policyDescription=Provides financial security to your family in case of your death]
Policy [policyId=3, policyName=Auto Insurance, policyDescription=Covers damages to your car and third-party liability]
Policy [policyId=4, policyName=Home Insurance, policyDescription=Covers damages to your home and belongings]
Policy [policyId=5, policyName=Health insurance, policyDescription=Covers medical expenses including hospital stays and treatments]
Policy [policyId=6, policyName=education, policyDescription=Help poor for education]
Policy [policyId=7, policyName=Education , policyDescription=Helps Poor People]
Policy [policyId=8, policyName=6, policyDescription=sad]
Policy [policyId=9, policyName=Travel Insurance, policyDescription=Covers various risks associated with traveling, including trip cancellations,
 lost luggage, and medi]
Policy [policyId=10, policyName=Travel Policy, policyDescription=Covers various risks associated with traveling, including trip cancellations, l
ost luggage, and medical emergencies abroad.]
```

4) Update Policy



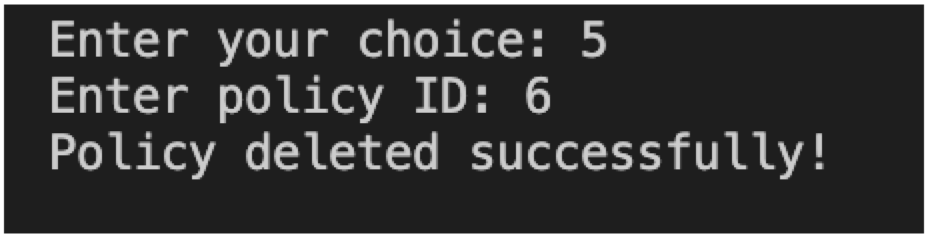
```
Enter your choice: 4
Enter policy ID: 2
Enter new policy name: Pet Insurance:
Enter new policy description: Covers veterinary expenses for pets, including accidents and illnesses.
Policy updated successfully!
```

DB before updating policy



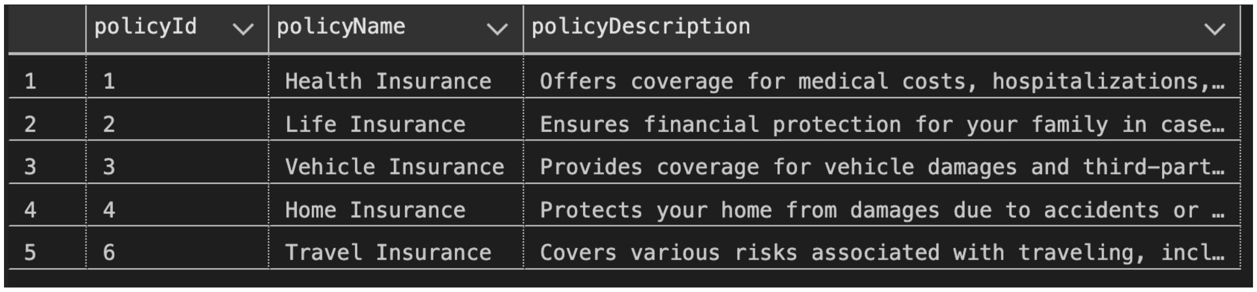
	policyId	policyName	policyDescription
1	1	Health Insurance	Offers coverage for medical costs, hospitalizations,…
2	2	Life Insurance	Ensures financial protection for your family in case…
3	3	Vehicle Insurance	Provides coverage for vehicle damages and third-part…
4	4	Home Insurance	Protects your home from damages due to accidents or …
5	6	Travel Insurance	Covers various risks associated with traveling, incl…

5) Delete Policy



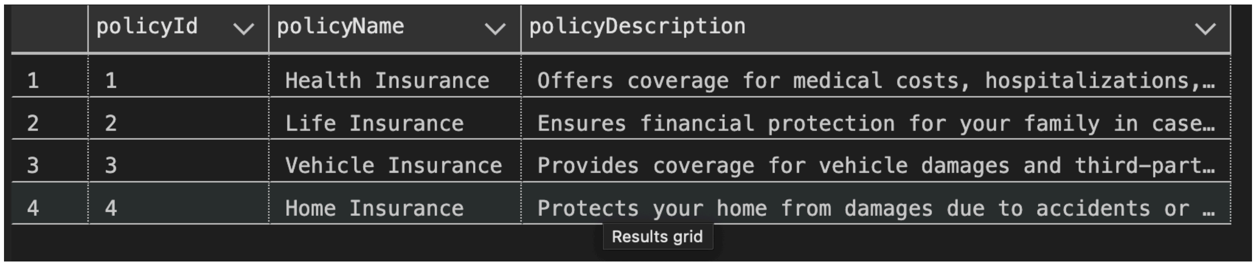
```
Enter your choice: 5
Enter policy ID: 6
Policy deleted successfully!
```

DB Before Deletion



	policyId	policyName	policyDescription
1	1	Health Insurance	Offers coverage for medical costs, hospitalizations,…
2	2	Life Insurance	Ensures financial protection for your family in case…
3	3	Vehicle Insurance	Provides coverage for vehicle damages and third-part…
4	4	Home Insurance	Protects your home from damages due to accidents or …
5	6	Travel Insurance	Covers various risks associated with traveling, incl…

DB After Deletion of policy (Policyid 6 deleted Travel Insurance)



	policyId	policyName	policyDescription
1	1	Health Insurance	Offers coverage for medical costs, hospitalizations,…
2	2	Life Insurance	Ensures financial protection for your family in case…
3	3	Vehicle Insurance	Provides coverage for vehicle damages and third-part…
4	4	Home Insurance	Protects your home from damages due to accidents or …

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.

```python
# src/util/db_connection.py
import pyodbc

class db_connection:
   @staticmethod
   def get_connection():

       connection_string = (
           "DRIVER={ODBC Driver 18 for SQL Server};"
           "SERVER=localhost;"
           "DATABASE=InsuranceManagementDB;"
           "TrustServerCertificate=yes;"
           "UID=sa;"
           "PWD=Raman63@;"
       )
       return pyodbc.connect(connection_string)

# Example usage
try:
   conn = db_connection.get_connection()
```

```
    print("Connection successful")
except Exception as e:
    print("Error connecting to the database:", e)
finally:
    if 'conn' in locals():
        conn.close()
```

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. PolicyNotFoundException :throw this exception when user enters an invalid patient number which doesn't exist in db

```
# exception/PolicyNotFoundException.py
class PolicyNotFoundException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

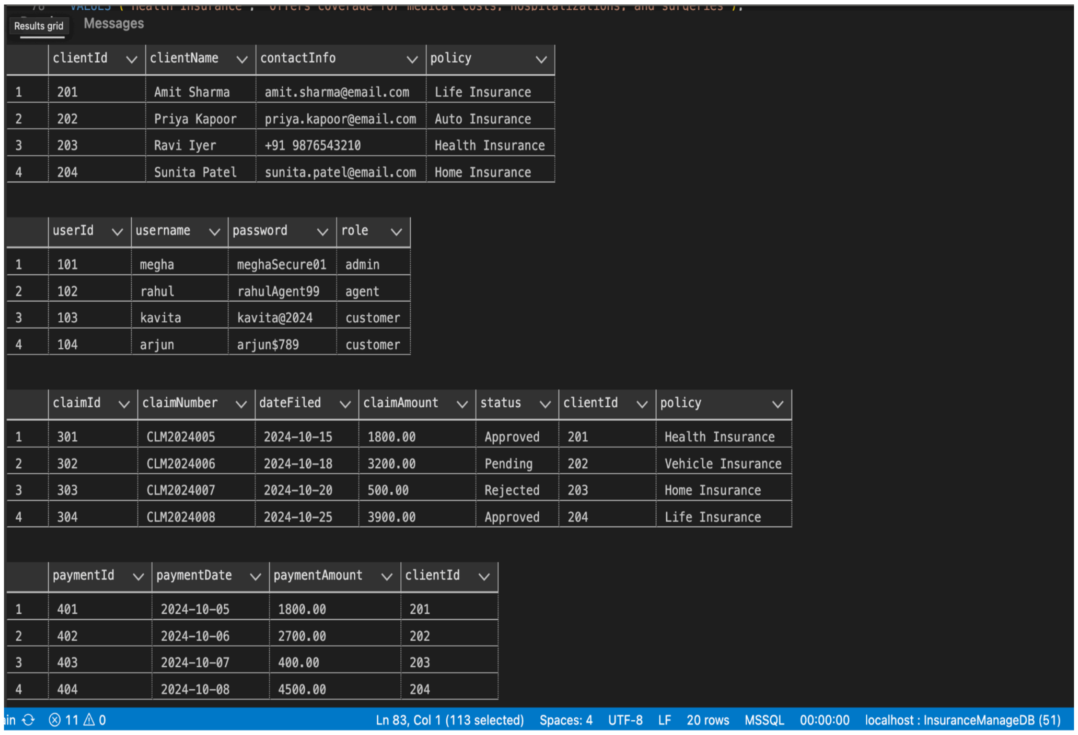
9. Create class named MainModule with main method in package mainmod. Trigger all the methods in service implementation class.

```
import sys
import os


path_to_insurance_management = os.path.abspath(os.path.join(os.path.dirname(__file__),
'..', '..'))
print("Path to InsuranceManagement:", path_to_insurance_management)

sys.path.append(path_to_insurance_management)

print(sys.path)

from src.dao.PolicyServiceImpl import PolicyServiceImpl
from src.dao.ClientServiceImpl import ClientServiceImpl
from src.dao.ClaimServiceImpl import ClaimServiceImpl
from src.dao.UserServiceImpl import UserServiceImpl
from src.dao.PaymentServiceImpl import PaymentServiceImpl
from src.entity.Policy import Policy
from src.entity.Client import Client
```

```python
from src.entity.Claim import Claim
from src.entity.User import User
from src.entity.Payment import Payment
from src.exception.PolicyNotFoundException import PolicyNotFoundException

if __name__ == "__main__":
   # Create instances of service classes
   policy_service = PolicyServiceImpl()
   client_service = ClientServiceImpl()
   claim_service = ClaimServiceImpl()
   user_service = UserServiceImpl()
   payment_service = PaymentServiceImpl()

   while True:
       print("\nInsurance Management System\n")
       print("1.  Create Policy")
       print("2.  Get Policy")
       print("3.  Get All Policies")
       print("4.  Update Policy")
       print("5.  Delete Policy")
       print("6.  Create Client")
       print("7.  Create User")
       print("8.  Get User")
       print("9.  Get All Users")
       print("10. Update User")
       print("11. Delete User")
       print("12. Create Claim")
       print("13. Get Claim")
       print("14. Get All Claims")
       print("15. Create Payment")
       print("16. Get Payment")
       print("17. Exit")

       choice = input("Enter your choice: ")

       if choice == '1':  # Create Policy
           policyName = input("Enter policy name: ")
           policyDescription = input("Enter policy description: ")
           policy = Policy(policyName=policyName, policyDescription=policyDescription)
           policy_service.createPolicy(policy)
           print("Policy created successfully!")
```

```
        elif choice == '2':  # Get Policy
            policyId = int(input("Enter policy ID: "))
            try:
                policy = policy_service.getPolicy(policyId)
                print(policy)
            except PolicyNotFoundException as e:
                print(e)

        elif choice == '3':  # Get All Policies
            policies = policy_service.getAllPolicies()
            for policy in policies:
                print(policy)

        elif choice == '4':  # Update Policy
            policyId = int(input("Enter policy ID: "))
            policyName = input("Enter new policy name: ")
            policyDescription = input("Enter new policy description: ")
            policy = Policy(policyId=policyId, policyName=policyName,
policyDescription=policyDescription)
            policy_service.updatePolicy(policy)
            print("Policy updated successfully!")

        elif choice == '5':  # Delete Policy
            policyId = int(input("Enter policy ID: "))
            policy_service.deletePolicy(policyId)
            print("Policy deleted successfully!")

        elif choice == '6':  # Create Client
            clientName = input("Enter client name: ")
            contactInfo = input("Enter contact info: ")
            policy = input("Enter policy: ")
            client = Client(clientName=clientName, contactInfo=contactInfo,
policy=policy)
            client_service.createClient(client)
            print("Client created successfully!")

        elif choice == '7':  # Create User
            username = input("Enter username: ")
            password = input("Enter password: ")
            role = input("Enter role (admin/user): ")
            user = User(username=username, password=password, role=role)
            user_service.createUser(user)
            print("User created successfully!")

        elif choice == '8':  # Get User
            userId = int(input("Enter user ID: "))
            user = user_service.getUser(userId)
            if user:
                print(user)
            else:
                print("User not found.")

        elif choice == '9':  # Get All Users
            users = user_service.getAllUsers()
            for user in users:
                print(user)

        elif choice == '10':  # Update User
            userId = int(input("Enter user ID: "))
            username = input("Enter new username: ")
            password = input("Enter new password: ")
            role = input("Enter new role (admin/user): ")
            user = User(userId=userId, username=username, password=password, role=role)
            user_service.updateUser(user)
            print("User updated successfully!")

        elif choice == '11':  # Delete User
            userId = int(input("Enter user ID: "))
            user_service.deleteUser(userId)
            print("User deleted successfully!")

        elif choice == '12':  # Create Claim
            claimNumber = input("Enter claim number: ")
            dateFiled = input("Enter date filed (YYYY-MM-DD): ")
            claimAmount = float(input("Enter claim amount: "))
            status = input("Enter status: ")
            policy = input("Enter associated policy: ")
            clientId = input("Enter associated client: ")
            claim = Claim(claimNumber=claimNumber, dateFiled=dateFiled,
claimAmount=claimAmount, status=status,
                          policy=policy, clientId=clientId)
            claim_service.createClaim(claim)
            print("Claim created successfully!")
        elif choice == '13':  # Get Claim
            claimId = int(input("Enter claim ID: "))
            claim = claim_service.getClaim(claimId)
            if claim:
                print(claim)
            else:
                print("Claim not found.")

        elif choice == '14':  # Get All Claims
            claims = claim_service.getAllClaims()
            for claim in claims:
                print(claim)

        elif choice == '15':  # Create Payment
            paymentDate = input("Enter payment date (YYYY-MM-DD): ")
            paymentAmount = float(input("Enter payment amount: "))
            client = input("Enter associated client: ")
            payment = Payment(paymentDate=paymentDate, paymentAmount=paymentAmount,
client=client)
            payment_service.createPayment(payment)
            print("Payment created successfully!")

        elif choice == '16':  # Get Payment
            paymentId = int(input("Enter payment ID: "))
            payment = payment_service.getPayment(paymentId)
            if payment:
                print(payment)
            else:
                print("Payment not found.")

        elif choice == '17':  # Exit
            print("Exiting...")
            break
        else:
            print("Invalid choice! Please try again.")
```

ENTIRE DATABSE



```
VALUES ('Health Insurance', 'Offers coverage for medical costs, hospitalizations, and surgeries');
```

	clientId	clientName	contactInfo	policy
1	201	Amit Sharma	amit.sharma@email.com	Life Insurance
2	202	Priya Kapoor	priya.kapoor@email.com	Auto Insurance
3	203	Ravi Iyer	+91 9876543210	Health Insurance
4	204	Sunita Patel	sunita.patel@email.com	Home Insurance

	userId	username	password	role
1	101	megha	meghaSecure01	admin
2	102	rahul	rahulAgent99	agent
3	103	kavita	kavita@2024	customer
4	104	arjun	arjun$789	customer

	claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy
1	301	CLM2024005	2024-10-15	1800.00	Approved	201	Health Insurance
2	302	CLM2024006	2024-10-18	3200.00	Pending	202	Vehicle Insurance
3	303	CLM2024007	2024-10-20	500.00	Rejected	203	Home Insurance
4	304	CLM2024008	2024-10-25	3900.00	Approved	204	Life Insurance

	paymentId	paymentDate	paymentAmount	clientId
1	401	2024-10-05	1800.00	201
2	402	2024-10-06	2700.00	202
3	403	2024-10-07	400.00	203
4	404	2024-10-08	4500.00	204

```
ⓧ 11 ⚠ 0    Ln 83, Col 1 (113 selected)   Spaces: 4   UTF-8   LF   20 rows   MSSQL   00:00:00   localhost : InsuranceManageDB (51)
```