COURIER MANAGEMENT SYSTEM – ASSIGNMENT

ENTITY:

User.py

```
from util.DBConnUtil import DBConnection

class User(DBConnection):
    def __init__(self):
        super().__init__()
        self.userID = 0
        self.userName = ' '
        self.email = ' '
        self.password = ' '
        self.contactNumber = 0
        self.address = ' '

    #Setters
    def set_userID(self,value):
        self.userID = value

    def set_userName(self,value):
        self.userName = value

    def set_email(self,value):
        self.email = value

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

    def set_contactNumber(self,value):
        self.contactNumber = value

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

    #Getters

    def get_userId(self):
        return self.userID

    def get_userName(self):
        return self.userName

    def get_email(self):
        return self.email

    def get_password(self):
        return self.password

    def get_contactNumber(self):
        return self.contactNumber

    def get_address(self):
        return self.address

    def __str__(self):
        return f'User ID: {self.userID} User Name: {self.userName}\n' \
               f'Email: {self.email} Contact Number: {self.contactNumber} Address:
{self.address}'
```

Courier.py

```
from entity.User import User
class Courier(User):
    def __init__(self):
        super().__init__()
        self.courierID = 0
        self.senderName = ' '
        self.senderAddress = ' '
        self.receiverName = ' '
        self.receiverAddress = ' '
        self.weight = 0.0
        self.status = ' '
        self.trackingNumber = 0
        self.deliveryDate = ' '
        self.userID = 0

    #Setters

    def set_courierID(self,value):
        self.courierID = value

    def set_senderName(self,value):
        self.senderName = value

    def set_senderAddress(self,value):
        self.senderAddress = value

    def set_receiverName(self,value):
        self.receiverName = value

    def set_receiverAddress(self,value):
        self.receiverAddress = value

    def set_weight(self,value):
        self.weight = value

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

    def set_trackingNumber(self,value):
        self.trackingNumber = value

    def set_deliveryDate(self,value):
        self.deliveryDate = value

    def set_userID(self,value):
        self.userID = value

    #Getters

    def get_courierID(self):
        return self.courierID

    def get_senderName(self):
        return self.senderName

    def get_senderAddress(self):
        return self.senderAddress

    def get_receiverName(self):
        return self.receiverName

    def get_receiverAddress(self):
        return self.receiverAddress

    def get_weight(self):
```

```
        return self.weight

    def get_status(self):
        return self.status

    def get_trackingNumber(self):
        return self.trackingNumber

    def get_deliveryDate(self):
        return self.deliveryDate

    def get_userID(self):
        return self.userID

    def __str__(self):
        return f'courierID : {self.courierID}\n' \
               f'senderName: {self.senderName} senderAddress : {self.senderAddress}\n'
\
               f'receiverName : {self.receiverName} receiverAddress :
{self.receiverAddress}\n' \
               f'weight : {self.weight} kg status : {self.status}\n' \
               f'trackingNumber : {self.trackingNumber} deliveryDate :
{self.deliveryDate} userId : {self.userID}'
```

Employee.py

```
from util.DBConnUtil import DBConnection


class Employee(DBConnection):
    def __init__(self):
        super().__init__()
        self.employeeID = 0
        self.employeeName = ''
        self.email = ''
        self.contactNumber = ' '
        self.role = ''
        self.salary = 0.0

    # SETTERS
    def set_employeeID(self, value):
        self.employeeID = value

    def set_employeeName(self, value):
        self.employeeName = value

    def set_email(self, value):
        self.email = value

    def set_contactNumber(self, value):
        self.contactNumber = value

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

    def set_salary(self, value):
        self.salary = value

    # GETTERS
    def get_employeeID(self):
        return self.employeeID

    def get_employeeName(self):
        return self.employeeName

    def get_email(self):
        return self.email

    def get_contactNumber(self):
        return self.contactNumber

    def get_role(self):
        return self.role

    def get_salary(self):
        return self.salary

    def __str__(self):
        return f'Employee ID: {self.employeeID} Name: {self.employeeName}\n' \
               f'Email: {self.email} Contact Number: {self.contactNumber}\n' \
               f'Role: {self.role} Salary: {self.salary}'
```

Location.py

```
from util.DBConnUtil import DBConnection

class Location(DBConnection):
    def __init__(self):
        super().__init__()
        self.locationID = 0
        self.locationName = ''
        self.address = ''

    # Setters

    def set_locationID(self, value):
        self.locationID = value

    def set_locationName(self, value):
        self.locationName = value

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

    # Getters

    def get_locationID(self):
        return self.locationID

    def get_locationName(self):
        return self.locationName

    def get_address(self):
        return self.address

    def __str__(self):
        return f'Location ID: {self.locationID} Name: {self.locationName}\n' \
               f'Address: {self.address}'
```

Couriercompany.py

```
from entity.Courier import Courier
from entity.Employee import Employee
from entity.Location import Location

class CourierCompany(Courier, Employee, Location):
    def __init__(self):
        super().__init__()
        self.companyName = ''
        self.courierID = 0
        self.employeeID = 0
        self.locationID = 0

    # SETTERS
    def set_companyName(self, value):
        self.companyName = value

    def set_courierID(self, value):
        self.courierID = value

    def set_employeeID(self, value):
        self.employeeID = value

    def set_locationID(self, value):
        self.locationID = value

    # GETTERS
```

```
    def get_companyName(self):
        return self.companyName

    def get_courierID(self):
        return self.courierID

    def get_employeeID(self):
        return self.employeeID

    def get_locationID(self):
        return self.locationID

    def __str__(self):
        return f'Company Name: {self.companyName} Courier ID: {self.courierID}\n' \
               f'Employee ID: {self.employeeID} Location ID: {self.locationID}'
```

Payment.py

```
from entity.Courier import Courier

class Payment(Courier):
    def __init__(self):
        super().__init__()
        self.paymentID = 0
        self.courierID = 0
        self.amount = 0.0
        self.paymentDate = ''

    # Setters

    def set_paymentID(self, value):
        self.paymentID = value

    def set_courierID(self, value):
        self.courierID = value

    def set_amount(self, value):
        self.amount = value

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

    # Getters

    def get_paymentID(self):
        return self.paymentID

    def get_courierID(self):
        return self.courierID

    def get_amount(self):
        return self.amount

    def get_paymentDate(self):
        return self.paymentDate

    def __str__(self):
        return f'Payment ID: {self.paymentID} Courier ID: {self.courierID}\n' \
               f'Amount: {self.amount} Payment Date: {self.paymentDate}'
```

DAO:

UserDAO:

```
from entity.User import User


class UserDAO(User):
    def __init__(self):
        super().__init__()

    def perform_user_actions(self):
        while True:
            print("(User) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_user_table()
            elif ch == 2:
                print(self.add_user())
            elif ch == 3:
                print(self.update_user())
            elif ch == 4:
                print(self.delete_user())
            elif ch == 5:
                self.select_user()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_user_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS User (
            userID INT PRIMARY KEY,
            userName VARCHAR(50),
            email VARCHAR(50),
            password VARCHAR(50),
            contactNumber INT,
            address VARCHAR(50))'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('User Table Created successfully.')
        except Exception as e:
            print(e)

    def add_user(self):
        try:
            self.open()
            self.userID = int(input('Enter User ID: '))
            self.userName = input('Enter User Name: ')
            self.email = input('Enter Email: ')
            self.password = input('Enter Password: ')
            self.contactNumber = int(input('Enter contact number: '))
            self.address = input('Enter Address: ')
            data = [(self.userID, self.userName, self.email, self.password,
self.contactNumber, self.address)]
            insert_str = '''INSERT INTO User(userID, userName, email, password,
contactNumber, address)
                            VALUES(%s, %s, %s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e
```

```
    def update_user(self):
        try:
            self.open()
            user_id = int(input('Input User ID to be Updated: '))
            self.userName = input('Enter User Name: ')
            self.email = input('Enter Email: ')
            self.password = input('Enter Password: ')
            self.contactNumber = int(input('Enter contact number: '))
            self.address = input('Enter Address: ')
            data = [(self.userName, self.email, self.password, self.contactNumber,
self.address, user_id)]
            update_str = '''UPDATE User SET userName=%s, email=%s, password=%s,
contactNumber=%s, address=%s
                            WHERE userID = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_user(self):
        try:
            self.open()
            user_id = int(input('Input User ID to be Deleted: '))
            delete_str = f'''DELETE FROM User WHERE userID = {user_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_user(self):
        try:
            select_str = '''SELECT * FROM User'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In User Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)
```

CourierDAO:

```
from entity.Courier import Courier


class CourierDAO(Courier):
    def __init__(self):
        super().__init__()

    def perform_courier_actions(self):
        while True:
            print("(Courier) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_courier_table()
            elif ch == 2:
                print(self.add_courier())
            elif ch == 3:
                print(self.update_courier())
            elif ch == 4:
                print(self.delete_courier())
            elif ch == 5:
                self.select_courier()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_courier_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Courier (
            courierID INT PRIMARY KEY,
            senderName VARCHAR(50),
            senderAddress VARCHAR(50),
            receiverName VARCHAR(50),
            receiverAddress VARCHAR(50),
            weight FLOAT,
            status VARCHAR(50),
            trackingNumber INT,
            deliveryDate DATE,
            userID INT,
            FOREIGN KEY(userID) REFERENCES User(userID) ON DELETE CASCADE ON UPDATE
CASCADE)'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('Courier Table Created successfully.')
        except Exception as e:
            print(e)

    def add_courier(self):
        try:
            self.open()
            self.courierID = int(input('Enter Courier ID: '))
            self.senderName = input('Enter Sender Name: ')
            self.senderAddress = input('Enter Sender Address: ')
            self.receiverName = input('Enter Receiver Name: ')
            self.receiverAddress = input('Enter Receiver Address: ')
            self.weight = float(input('Enter Weight: '))
            self.status = input('Enter Status: ')
            self.trackingNumber = int(input('Enter Tracking Number: '))
            self.deliveryDate = input('Enter Delivery Date: ')
            self.userID = int(input('Enter User ID: '))
            data = [(self.courierID, self.senderName, self.senderAddress,
self.receiverName, self.receiverAddress,
                     self.weight, self.status, self.trackingNumber, self.deliveryDate,
self.userID)]
```

```
            insert_str = '''INSERT INTO Courier(courierID, senderName, senderAddress,
receiverName, receiverAddress,
                            weight, status, trackingNumber, deliveryDate, userID)
                            VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def update_courier(self):
        try:
            self.open()
            courier_id = int(input('Input Courier ID to be Updated: '))
            self.senderName = input('Enter Sender Name: ')
            self.senderAddress = input('Enter Sender Address: ')
            self.receiverName = input('Enter Receiver Name: ')
            self.receiverAddress = input('Enter Receiver Address: ')
            self.weight = float(input('Enter Weight: '))
            self.status = input('Enter Status: ')
            self.trackingNumber = int(input('Enter Tracking Number: '))
            self.deliveryDate = input('Enter Delivery Date: ')
            self.userID = int(input('Enter User ID: '))
            data = [(self.senderName, self.senderAddress, self.receiverName,
self.receiverAddress,
                     self.weight, self.status, self.trackingNumber, self.deliveryDate,
self.userID, courier_id)]
            update_str = '''UPDATE Courier SET senderName=%s, senderAddress=%s,
receiverName=%s, receiverAddress=%s,
                            weight=%s, status=%s, trackingNumber=%s, deliveryDate=%s,
userID=%s
                            WHERE courierID = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_courier(self):
        try:
            self.open()
            courier_id = int(input('Input Courier ID to be Deleted: '))
            delete_str = f'''DELETE FROM Courier WHERE courierID = {courier_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_courier(self):
        try:
            select_str = '''SELECT * FROM Courier'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In Courier Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)

    def placeOrder(self):
        pass
```

```
    def getOrderStatus(self, trackingNumber):
        pass

    def cancelOrder(self, trackingNumber):
        pass
```

EmployeeDAO:

```
from entity.Employee import Employee


class EmployeeDAO(Employee):
    def __init__(self):
        super().__init__()

    def perform_employee_actions(self):
        while True:
            print("(Employee) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_employee_table()
            elif ch == 2:
                print(self.add_employee())
            elif ch == 3:
                print(self.update_employee())
            elif ch == 4:
                print(self.delete_employee())
            elif ch == 5:
                self.select_employee()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_employee_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Employee (
            employeeID INT PRIMARY KEY,
            employeeName VARCHAR(50),
            email VARCHAR(50),
            contactNumber VARCHAR(15),
            role VARCHAR(50),
            salary FLOAT)'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('Employee Table Created successfully.')
        except Exception as e:
            print(e)

    def add_employee(self):
        try:
            self.open()
            self.employeeID = int(input('Enter Employee ID: '))
            self.employeeName = input('Enter Employee Name: ')
            self.email = input('Enter email: ')
            self.contactNumber = (input('Enter contact number: '))
            self.role = input('Enter role: ')
            self.salary = float(input('Enter salary: '))
            data = [(self.employeeID, self.employeeName, self.email,
self.contactNumber, self.role, self.salary)]
            insert_str = '''INSERT INTO Employee(employeeID, employeeName, email,
contactNumber, role, salary)
                            VALUES(%s, %s, %s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
```

```
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def update_employee(self):
        try:
            self.open()
            employee_id = int(input('Input Employee ID to be Updated: '))
            self.employeeName = input('Enter Employee Name: ')
            self.email = input('Enter email: ')
            self.contactNumber = (input('Enter contact number: '))
            self.role = input('Enter role: ')
            self.salary = float(input('Enter salary: '))
            data = [(self.employeeName, self.email, self.contactNumber, self.role,
self.salary, employee_id)]
            update_str = '''UPDATE Employee SET employeeName=%s, email=%s,
contactNumber=%s, role=%s, salary=%s
                            WHERE employeeID = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_employee(self):
        try:
            self.open()
            employee_id = int(input('Input Employee ID to be Deleted: '))
            delete_str = f'''DELETE FROM Employee WHERE employeeID = {employee_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_employee(self):
        try:
            select_str = '''SELECT * FROM Employee'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In Employee Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)
```

LocationDAO:

```
from entity.Location import Location

class LocationDAO(Location):
    def __init__(self):
        super().__init__()

    def perform_location_actions(self):
        while True:
            print("(Location) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT")
            ch = int(input("Enter your choice: "))
            if ch == 1:
                self.create_location_table()
            elif ch == 2:
                print(self.add_location())
            elif ch == 3:
                print(self.update_location())
            elif ch == 4:
                print(self.delete_location())
            elif ch == 5:
                print(self.select_location())
            elif ch == 0:
                break
            else:
                print("Invalid choice")
    def create_location_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Location(
            locationID INT PRIMARY KEY,
            locationName VARCHAR(50),
            address VARCHAR(50)
            )'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print("Location table created successfully")
        except Exception as e:
            return e

    def add_location(self):
        try:
            self.open()
            self.locationID = int(input("Enter LocationID: "))
            self.locationName = input("Enter location name: ")
            self.address = input("Enter address: ")
            data = [(self.locationID,self.locationName,self.address)]
            insert_str = '''INSERT INTO Location(locationID,locationName,address)
                            values(%s,%s,%s)'''
            self.stmt.executemany(insert_str,data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return(e)

    def update_location(self):
        try:
            self.open()
            location_id = int(input("Enter location id to be updated: "))
            self.locationName = input("Enter location name: ")
            self.address = input("Enter address: ")
            data = [(self.locationName,self.address,location_id)]
            update_str = '''UPDATE Location
                            SET locationName = %s, address = %s
                            WHERE locationID = %s'''
            self.stmt.executemany(update_str,data)
```

```
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_location(self):
        try:
            self.open()
            location_id = int(input('Input Location ID to be Deleted: '))
            delete_str = f'''DELETE FROM Location WHERE locationID = {location_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_location(self):
        try:
            select_str = '''SELECT * FROM Location'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In Location Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)
```

CourierCompanyDAO:

```
from entity.CourierCompany import CourierCompany


class CourierCompanyDAO(CourierCompany):
    def __init__(self):
        super().__init__()

    def perform_courier_company_actions(self):
        while True:
            print("(CourierCompany) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_courier_company_table()
            elif ch == 2:
                print(self.add_courier_company())
            elif ch == 3:
                print(self.update_courier_company())
            elif ch == 4:
                print(self.delete_courier_company())
            elif ch == 5:
                self.select_courier_company()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_courier_company_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS CourierCompany (
            companyName VARCHAR(50) PRIMARY KEY,
            courierID INT,
```

```
            employeeID INT,
            locationID INT,
            FOREIGN KEY(courierID) REFERENCES Courier(courierID) ON DELETE CASCADE ON
UPDATE CASCADE,
            FOREIGN KEY(employeeID) REFERENCES Employee(employeeID) ON DELETE CASCADE
ON UPDATE CASCADE,
            FOREIGN KEY(locationID) REFERENCES Location(locationID) ON DELETE CASCADE
ON UPDATE CASCADE)'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('CourierCompany Table Created successfully.')
        except Exception as e:
            print(e)

    def add_courier_company(self):
        try:
            self.open()
            self.companyName = input('Enter Company Name: ')
            self.courierID = int(input('Enter Courier ID: '))
            self.employeeID = int(input('Enter Employee ID: '))
            self.locationID = int(input('Enter Location ID: '))
            data = [(self.companyName, self.courierID, self.employeeID,
self.locationID)]
            insert_str = '''INSERT INTO CourierCompany(companyName, courierID,
employeeID, locationID)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def update_courier_company(self):
        try:
            self.open()
            company_name = input('Enter Company Name to be Updated: ')
            self.courierID = int(input('Enter Courier ID: '))
            self.employeeID = int(input('Enter Employee ID: '))
            self.locationID = int(input('Enter Location ID: '))
            data = [(self.courierID, self.employeeID, self.locationID, company_name)]
            update_str = '''UPDATE CourierCompany SET courierID=%s, employeeID=%s,
locationID=%s
                            WHERE companyName = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_courier_company(self):
        try:
            self.open()
            company_name = input('Enter Company Name to be Deleted: ')
            delete_str = f'''DELETE FROM CourierCompany WHERE companyName =
'{company_name}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_courier_company(self):
        try:
```

```
            select_str = '''SELECT * FROM CourierCompany'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In CourierCompany Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)

    def getAssignedOrder(self, employeeID):
        pass
```

PaymentDAO:

```
from entity.Payment import Payment


class PaymentDAO(Payment):
    def __init__(self):
        super().__init__()

    def perform_payment_actions(self):
        while True:
            print("(Payment) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_payment_table()
            elif ch == 2:
                print(self.add_payment())
            elif ch == 3:
                print(self.update_payment())
            elif ch == 4:
                print(self.delete_payment())
            elif ch == 5:
                self.select_payment()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_payment_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Payment (
            paymentID INT PRIMARY KEY,
            courierID INT,
            amount FLOAT,
            paymentDate DATE,
            FOREIGN KEY(courierID) REFERENCES Courier(courierID) ON DELETE CASCADE ON
UPDATE CASCADE)'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('Payment Table Created successfully.')
        except Exception as e:
            print(e)

    def add_payment(self):
        try:
            self.open()
            self.paymentID = int(input('Enter Payment ID: '))
            self.courierID = int(input('Enter Courier ID: '))
            self.amount = float(input('Enter Amount: '))
            self.paymentDate = input('Enter Payment Date: ')
            data = [(self.paymentID, self.courierID, self.amount, self.paymentDate)]
```

```
            insert_str = '''INSERT INTO Payment(paymentID, courierID, amount,
paymentDate)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def update_payment(self):
        try:
            self.open()
            payment_id = int(input('Input Payment ID to be Updated: '))
            self.courierID = int(input('Enter Courier ID: '))
            self.amount = float(input('Enter Amount: '))
            self.paymentDate = input('Enter Payment Date: ')
            data = [(self.courierID, self.amount, self.paymentDate, payment_id)]
            update_str = '''UPDATE Payment SET courierID=%s, amount=%s, paymentDate=%s
                            WHERE paymentID = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def delete_payment(self):
        try:
            self.open()
            payment_id = int(input('Input Payment ID to be Deleted: '))
            delete_str = f'''DELETE FROM Payment WHERE paymentID = {payment_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            self.close()
            return True
        except Exception as e:
            return e

    def select_payment(self):
        try:
            select_str = '''SELECT * FROM Payment'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            self.close()
            print('Records In Payment Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(e)
```

IcourierServiceDAO:

```
from dao.CourierDAO import CourierDAO
from dao.CourierCompanyDAO import CourierCompanyDAO
from exception.TrackingNumberNotFoundException import TrackingNumberNotFoundException
from exception.InvalidEmployeeIdException import InvalidEmployeeIdException


class ICourierUserService(CourierCompanyDAO, CourierDAO):
    def __init__(self):
        super().__init__()

    # PLACE ORDER
    def placeOrder(self):
        print('Enter Courier Details to Place a new courier order: ')
        print(self.add_courier())
        try:
            self.open()
            self.stmt.execute(f'''SELECT trackingNumber FROM Courier WHERE courierID =
{self.courierID}''')
            records = self.stmt.fetchone()[0]
            self.close()
            return records
        except Exception as e:
            return e

    # GET ORDER STATUS
    def getOrderStatus(self, trackingNumber):
        try:
            self.open()
            self.stmt.execute(f'''SELECT COUNT(*) FROM Courier WHERE trackingNumber =
{trackingNumber}''')
            count = self.stmt.fetchone()[0]
            if count == 0:
                raise TrackingNumberNotFoundException(trackingNumber)
            else:
                self.stmt.execute(f'''SELECT status FROM Courier WHERE trackingNumber =
{trackingNumber}''')
                records = self.stmt.fetchone()[0]
                self.close()
                return records
        except TrackingNumberNotFoundException as e:
            return e
        except Exception as e:
            return e

    # CANCEL ORDER
    def cancelOrder(self, trackingNumber):
        try:
            self.open()
            self.stmt.execute(f'''SELECT COUNT(*) FROM Courier WHERE trackingNumber =
{trackingNumber}''')
            count = self.stmt.fetchone()[0]
            if count == 0:
                raise TrackingNumberNotFoundException(trackingNumber)
            else:
                self.stmt.execute(f'''UPDATE Courier SET status = "Cancelled" WHERE
trackingNumber = {trackingNumber}''')
                self.conn.commit()
                self.close()
                return True
        except TrackingNumberNotFoundException as e:
            print(e)
            return False
        except Exception as e:
            print(e)
            return False
```

```
    # GET ASSIGNED ORDER
    def getAssignedOrder(self, employeeID):
        try:
            self.open()
            self.stmt.execute(f'''SELECT COUNT(*) FROM Employee WHERE employeeID =
{employeeID}''')
            count = self.stmt.fetchone()[0]
            if count == 0:
                raise InvalidEmployeeIdException(employeeID)
            else:
                self.stmt.execute(f'''SELECT * FROM Courier AS C JOIN CourierCompany AS
CC ON C.courierID = CC.courierID
                                    WHERE CC.employeeID = {employeeID}''')
                records = self.stmt.fetchall()
                self.close()
                return records
        except InvalidEmployeeIdException as e:
            return e
        except Exception as e:
            return e
```

EXCEPTIONS:

InvalidEmployeeIdException:

```
class InvalidEmployeeIdException(Exception):
    def __init__(self,employee_id):
        super() .__init__(f'EmployeeID {employee_id} is not found in the system')
```

TrackingNumberNotFoundException:

```
class TrackingNumberNotFoundException(Exception):
    def __init__(self,tracking_Number):
        super().__init__(f'TrackingNumber {tracking_number} is not found in the
system')
```

UTIL:

DBConnUtil:

```
#import sys
import mysql.connector as sql
from util.DBPropertyUtil import PropertyUtil

class DBConnection:
    def open(self):
        try:
            # print('--Database Is Connected:--')
            connection_properties = PropertyUtil.getConnectionString()
            self.conn = sql.connect(**connection_properties)
            self.stmt = self.conn.cursor()
        except Exception as e:
            print(str(e) + ' --Database Is Not Connected:--')
            sys.exit(1)

    def close(self):
        self.conn.close()
        # print('--Connection Is Closed:--')
```

DBPropertyUtil:

```
class PropertyUtil:
    connection_properties = None

    @staticmethod
    def getConnectionString():
        if PropertyUtil.connection_properties is None:
            host = 'localhost'
            database = 'courier_managementdb'
            user = 'root'
            password = 'Mahitharsha@1'
            PropertyUtil.connection_properties = {'host': host, 'database': database,
'user': user, 'password': password}
        return PropertyUtil.connection_properties
```

MAIN:

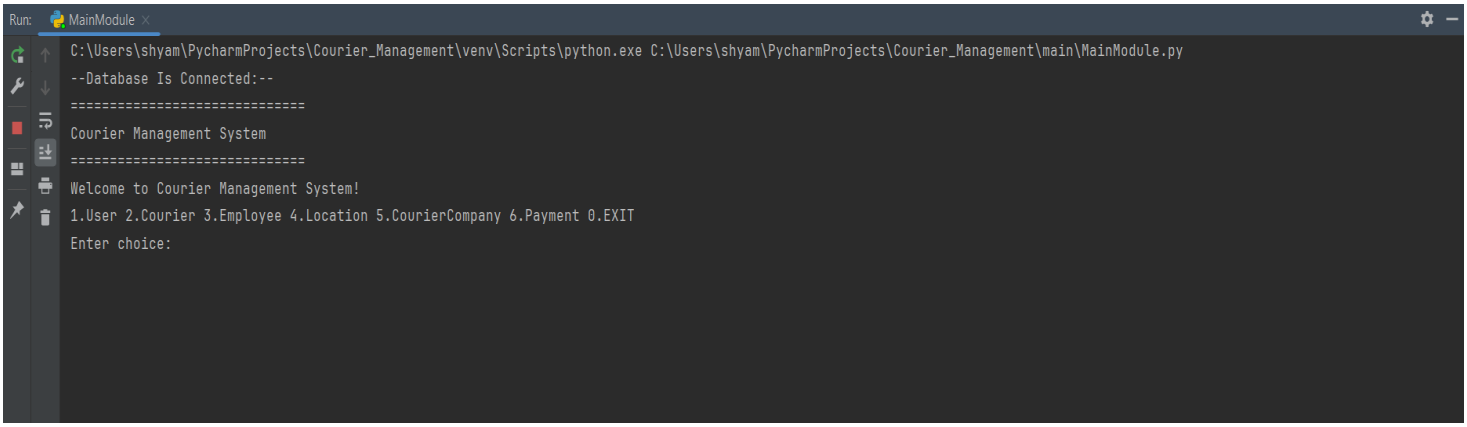
Main.py (Executable file)

```
from dao.ICourierUserService import ICourierUserService
from dao.UserDAO import UserDAO
from dao.CourierDAO import CourierDAO
from dao.EmployeeDAO import EmployeeDAO
from dao.LocationDAO import LocationDAO
from dao.CourierCompanyDAO import CourierCompanyDAO
from dao.PaymentDAO import PaymentDAO
from exception.TrackingNumberNotFoundException import TrackingNumberNotFoundException
from exception.InvalidEmployeeIdException import InvalidEmployeeIdException
from util.DBConnUtil import DBConnection


def main():

    dbconnection = DBConnection()

    try:
        dbconnection.open()
        print("--Database Is Connected:--")
    except Exception as e:
        print(e)

    try:
        print("=" * 30)
        print("Courier Management System")
        print("=" * 30)
        print("Welcome to Courier Management System!")



        while True:
            print("1.User 2.Courier 3.Employee 4.Location 5.CourierCompany 6.Payment
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                u = UserDAO()
                u.perform_user_actions()
            elif ch == 2:
                c = CourierDAO()
                c.perform_courier_actions()
            elif ch == 3:
                e = EmployeeDAO()
                e.perform_employee_actions()
```

```
            elif ch == 4:
                l = LocationDAO()
                l.perform_location_actions()
            elif ch == 5:
                cc = CourierCompanyDAO()
                cc.perform_courier_company_actions()
            elif ch == 6:
                p = PaymentDAO()
                p.perform_payment_actions()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

        courier_management_system = ICourierUserService()

        while True:
            print("=" * 10)
            print("---MENU---")
            print("=" * 10)

print("1.placeOrder\n2.getOrderStatus\n3.cancelOrder\n4.getAssignedOrder\n0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                print(f'Tracking Number of newly placed Order is
{courier_management_system.placeOrder()}')
            elif ch == 2:
                print(courier_management_system.getOrderStatus(int(input('Enter
Tracking Number of the Courier to get Order Status: '))))
            elif ch == 3:
                print(courier_management_system.cancelOrder(int(input('Enter Tracking
Number of the Courier to be Cancelled: '))))
            elif ch == 4:
                print(courier_management_system.getAssignedOrder(int(input('Enter
Employee ID to list the courier orders assigned to him/her: '))))
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    except TrackingNumberNotFoundException as e:
        print(e)

    except InvalidEmployeeIdException as e:
        print(e)

    except Exception as e:
        print(e)

    finally:
        dbconnection.close()
        print("Thankyou for visiting Courier Management System!")
        print("--Connection Is Closed:--")


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

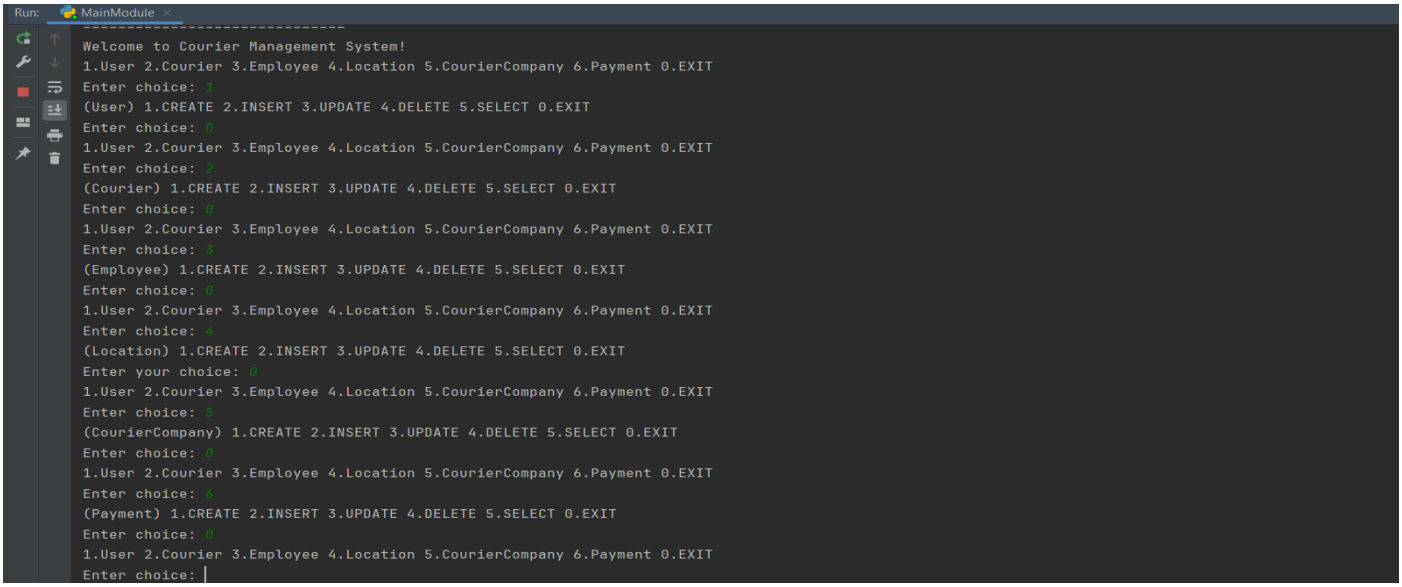
OUTPUTS:

#Running the Main file



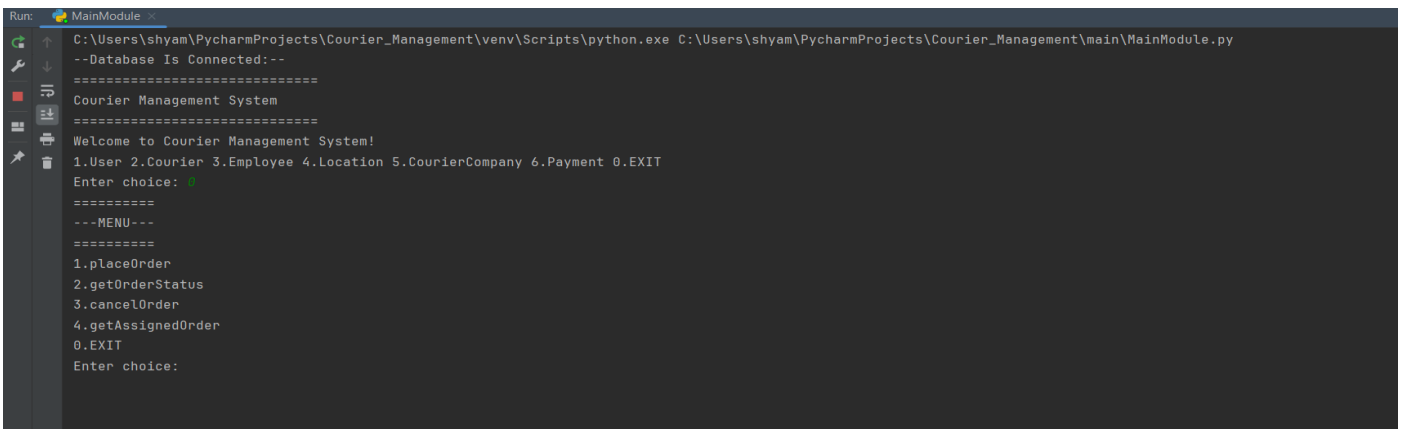
➔ A menu driven program that asks for user input to select between above option to perform CRUD operations

#Selecting the options



➔ By selecting respective options from above we can perform CREATE,INSERT,UPDATE,DELETE and SELECT operations that automatically reflects in our database.

#Directly exit from the loop to enter into the menu



#To place Order

```
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 1
Enter Courier Details to Place a new courier order:
Enter Courier ID: 11
Enter Sender Name: QQQ
Enter Sender Address: QQ Street
Enter Receiver Name: WWW
Enter Receiver Address: WW Street
Enter Weight: 2.8
Enter Status: Ordered
Enter Tracking Number: 13001
Enter Delivery Date: 2024-03-10
Enter User ID: 4
True
Tracking Number of newly placed Order is 13001
```

#To Get Order Status

```
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 2
Enter Tracking Number of the Courier to get Order Status: 13001
Ordered
```

To Cancel Order

```
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 3
Enter Tracking Number of the Courier to be Cancelled: 13001
True
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 2
Enter Tracking Number of the Courier to get Order Status: 13001
Cancelled
```

#To Get Assigned Order

```
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 4
Enter Employee ID to list the courier orders assigned to him/her: 6

[(2, 'DEF', 'DEF STREET', 'USER1', 'USER1 STREET', 3.0, 'IN TRANSIT', 1123, datetime.date(2024, 3, 8), 1, 'BLACK DART', 2, 6, 5), (6, 'PQR', 'PQR STREET', 'USER2', 'USER2 STREET
```

#Finally

```
==========
---MENU---
==========
1.placeOrder
2.getOrderStatus
3.cancelOrder
4.getAssignedOrder
0.EXIT
Enter choice: 0
Thankyou for visiting Courier Management System!
--Connection Is Closed:--

Process finished with exit code 0
```