Car Rental System

Submitted By : Mridul Bhardwaj

Create following tables in SQL Schema with appropriate class and write the unit test case for the Car Rental application.

Schema Design:

1. Vehicle Table:
   - vehicleID (Primary Key)
   - make
   - model
   - year
   - dailyRate
   - status (available, notAvailable)
   - passengerCapacity
   - engineCapacity

```
class Vehicle:
    def __init__(self, vehicleID : int,
                 make : str,
                 model : str,
                 year : int,
                 dailyRate : float,
                 status : str,
                 passengerCapacity : int,
                 engineCapacity : int):

        self.vehicleID = vehicleID
        self.make = make
        self.model = model
        self.year = year
        self.dailyRate = dailyRate
        self.status = status
        self.passengerCapacity = passengerCapacity
        self.engineCapacity = engineCapacity


    #Getter Methods
    def getVehicleID(self):
        return self.vehicleID

    def getMake(self):
        return self.make

    def getModel(self):
        return self.model
```

```
def getYear(self):
    return self.year

def getDailyRate(self):
    return self.dailyRate

def getStatus(self):
    return  self.status

def getPassengerCapacity(self):
    return self.passengerCapacity

def getEngineCapacity(self):
    return self.engineCapacity


#Setter Methods
def setVehicleID(self,vehicleID):
    self.vehicleID = vehicleID

def setMake(self,make):
    self.make = make

def setModel(self,model):
    self.model = model

def setYear(self,year):
    set.year = year


def setDailyRate(self,dailyRate):
    self.dailyRate = dailyRate

def setStaus(self,status):
    if status not in ['Available','NotAvailable']:
        raise Exception("Status should be Available or NotAvailable")
    self.status = status

def setpassengerCapacity(self,passengerCapacity):
    self.passengerCapacity = passengerCapacity

def setEngineCapacity(self,engineCapacity):
    self.engineCapacity = engineCapacity
```

2. Customer Table:
   - customerID (Primary Key)
   - firstName
   - lastName
   - email
   - phoneNumber

```python
import re
13 usages
class Customer:
    def __init__(self, customerID : int ,
                 firstName : str,
                 lastName : str,
                 email : str,
                 phoneNumber : str):
        self.customerID = customerID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber

    #Getter Methods
    def getCustomerID(self):
        return self.customerID

    def getFirstName(self):
        return self.firstName

    def getLastName(self):
        return self.lastName

    def getEmail(self):
        return self.email

    def getPhoneNumber(self):
        return self.phoneNumber
```

```python
#Setter Methods
def setCustomerID(self,customerID):
    if isinstance(customerID, int) and customerID > 0:
        self.customerID = customerID
    else:
        print("Invalid customer_id. It should be a positive integer.")

    def setFirstName(self, firstName):
        if firstName:
            self.firstName = firstName
        else:
            print("Invalid first_name. It should not be empty.")

    def setLastName(self, lastName):
        if lastName:
            self.lastName = lastName
        else:
            print("Invalid last_name. It should not be empty.")

    def setEmail(self, email):
        if email and re.match( pattern: r"[^@]+@[^@]+\.[^@]+", email):
            self.email = email
        else:
            print("Invalid email format.")

    def setPhoneNumber(self, phoneNumber):
        if phoneNumber and re.match( pattern: r"\d{10}", phoneNumber):
            self.phoneNumber = phoneNumber
        else:
            print("Invalid phone number format.")
```

3. Lease Table:
   - leaseID (Primary Key)
   - vehicleID (Foreign Key referencing Vehicle Table)
   - customerID (Foreign Key referencing Customer Table)
   - startDate
   - endDate
   - type (to distinguish between DailyLease and MonthlyLease)

```
class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate,type):
        self.leaseID = leaseID
        self.vehicleID = vehicleID
        self.customerID = customerID
        self.startDate = startDate
        self.endDate = endDate
        self.type = type

    @property
    def getLeaseID(self):
        return self.leaseID

    @property
    def getVehicleID(self):
        return self.vehicleID

    @property
    def getCustomerID(self):
        return self.customerID

    @property
    def getStartDate(self):
        return self.startDate

    @property
    def getEndDate(self):
        return self.endDate

    @property
    def getType(self):
        return self.type

    def setLeaseID(self, value):
        self.leaseID = value

    def setVehicleID(self, value):
        self.vehicleID = value

    def setCustomerID(self, value):
        self.customerID = value

    def setStartDate(self, value):
        self.startDate = value

    def setEndDate(self, value):
        self.endDate = value

    def setType(self, value):
        self.type = value
```

4. Payment Table:
   - paymentID (Primary Key)
   - leaseID (Foreign Key referencing Lease Table)
   - paymentDate
   - amount

```python
class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self.paymentID = paymentID
        self.leaseID = leaseID
        self.paymentDate = paymentDate
        self.amount = amount

#Getter Methods
def getPaymentID(self):
    return self.paymentID

def getLeaseID(self):
    return self.leaseID

def getPaymentDate(self):
    return self.paymentDate

def getAmount(self):
    return self.amount

#Setter Methods
def setPaymentID(self,paymentID):
    self.paymentID = paymentID

def setLeaseID(self,leaseID):
    self.leaseID = leaseID

def setPaymentDate(self,paymentDate):
    self.paymentDate = paymentDate

def setAmount(self,amount):
    self.amount = amount
```

5. Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

6. Service Provider Interface/Abstract class: Keep the interfaces and implementation classes in package dao.
   • Create Interface for ICarLeaseRepository and add following methods which interact with database.

   • Car Management
   1. addCar(Car car)
      parameter : Car

return type : void

2. removeCar()
   parameter : carID
   return type : void
3. listAvailableCars() –
   parameter: NIL
   return type: return List of Car
4. listRentedCars() –
   return List of Car parameter: NIL
   return type: return List of Car
5. findCarById(int carID) –
   return Car if found or throw exception
   parameter: NIL
   return type: return List of Car

```python
from abc import ABC,abstractmethod
from typing import List,Optional
from datetime import date
from entity.Customer import Customer
from entity.Vehicle import Vehicle
from entity.Lease import Lease

9 usages
class ICarLeaseRepository(ABC):

    @abstractmethod
    def addCar(self, vehicle: Vehicle) -> None:
        pass

    @abstractmethod
    def removeCar(self, vehicleID: int) -> None:
        pass


    @abstractmethod
    def listAvailableCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def listRentedCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def findCarById(self, vehicleId: int) -> Vehicle:
        pass
```

• Customer Management

1. addCustomer(Customer customer)
   parameter : Customer
   return type : void
2. void removeCustomer(int customerID)
   parameter : CustomerID
   return type : void
3. listCustomers()
   parameter : NIL
   return type : list of customer
4. findCustomerById(int customerID)
   parameter : CustomerID
   return type : Customer

```python
@abstractmethod
def addCustomer(self, customer: Customer) -> None:
    pass

@abstractmethod
def removeCustomer(self, customerID: int) -> None:
    pass

@abstractmethod
def listCustomers(self) -> List[Customer]:
    pass

@abstractmethod
def findCustomerById(self, customerID: int) -> Customer:
    pass
```

• Lease Management

1. createLease()
   parameter : int customerID, int carID, Date startDate, Date endDate
   return type : Lease
2. void returnCar();
   parameter : int leaseID
   return type : Lease info
3. List listActiveLeases();
   parameter : NIL
   return type : Lease list
4. listLeaseHistory();
   parameter : NIL
   return type : Lease list

```
@abstractmethod
def createLease(self, customerID: int, vehicleID: int,
                startDate: date, endDate: date) -> Optional[Lease]:
    pass

@abstractmethod
def returnCar(self, leaseID: int) -> Lease:
    pass

@abstractmethod
def listActiveLeases(self) -> List[Lease]:
    pass

@abstractmethod
def listLeaseHistory(self) -> List[Lease]:
    pass
```

- Payment Handling

1. void recordPayment();
   parameter : Lease lease, double amount
   return type : void

```
@abstractmethod
def recordPayment(self, lease: Lease, amount: float) -> None:
    pass
```

7. Implement the above interface in a class called ICarLeaseRepositoryImpl in package dao.

```
import mysql.connector
from mysql.connector import Error

from dao.ICarLeaseRepository import ICarLeaseRepository
from entity.lease import Lease
from datetime import date
```

4 usages

```
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self, connection_params):
        self.connection_params = connection_params
        self.connection = self.create_connection()
```

```python
def create_connection(self):
    try:
        connection = mysql.connector.connect(**self.connection_params)
        if connection.is_connected():
            print("Congratulations!!!")
            print("Database Connected Successfully..")
            return connection
    except Error as e:
        print(f"Error: {e}")
        return None
```

1 usage

```python
def close_connection(self):
    if self.connection.is_connected():
        self.connection.close()
        print("Connection Over....")
```

```python
def addCar(self, car):
    try:
        cursor = self.connection.cursor()
        query = ("INSERT INTO vehicle (vehicleID, make, model, year,"
                 " dailyRate, status, passengerCapacity, engineCapacity) "
                 "VALUES (%s, %s, %s, %s, %s, %s, %s, %s)")
        values = (car.vehicleID, car.make, car.model, car.year, car.dailyRate, car.status, car.passengerCapacity
        cursor.execute(query, values)
        self.connection.commit()
        print("Car added successfully!!")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

1 usage

```python
def removeCar(self, carID) :
    try:
        cursor = self.connection.cursor()
        query = "DELETE FROM vehicle WHERE vehicleID = %s"
        values = (carID,)
        cursor.execute(query, values)
        self.connection.commit()
        print("Car removed successfully!!")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```
def listAvailableCars(self):
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE status = 'available'"
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

1 usage

```
def listRentedCars(self) :
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE status = 'notAvailable'"
        cursor.execute(query)
        result = cursor.fetchall()

        return result
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```
def findCarById(self, carID):
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM vehicle WHERE vehicleID = %s"
        values = (carID,)
        cursor.execute(query, values)
        result = cursor.fetchone()
        if result:
            return result
        else:
            raise Exception(f"Car with ID {carID} not found..")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```python
def addCustomer(self, customer):
    try:
        cursor = self.connection.cursor()
        query = ("INSERT INTO customer (customerID, firstName, lastName, email, phoneNumber) "
                 "VALUES (%s, %s, %s, %s, %s)")
        values = (customer.customerID, customer.firstName, customer.lastName,
                  customer.email, customer.phoneNumber)
        cursor.execute(query, values)
        self.connection.commit()
        print("Customer added successfully!!!")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

1 usage

```python
def removeCustomer(self, customerID: int) :
    try:
        cursor = self.connection.cursor()
        query = "DELETE FROM customer WHERE customerID = %s"
        values = (customerID,)
        cursor.execute(query, values)
        self.connection.commit()
        print("Customer removed successfully!!")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```python
def listCustomers(self) :
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM customer"
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()

def findCustomerById(self, customerID: int):
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM customer WHERE customerID = %s"
        values = (customerID,)
        cursor.execute(query, values)
        result = cursor.fetchone()
        if result:
            return result
        else:
            raise Exception(f"Customer with ID {customerID} not found")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```python
def createLease(self, customerID: int, carID: int, startDate, endDate,type:str):
    try:
        cursor = self.connection.cursor()

        # Fetch the current maximum leaseID from the database
        cursor.execute("SELECT MAX(leaseID) FROM lease")
        max_lease_id = cursor.fetchone()[0]

        # Increment the max_lease_id by 1 to get the new leaseID
        new_lease_id = max_lease_id + 1 if max_lease_id is not None else 1

        # Insert the new lease record with the calculated leaseID
        query = ("INSERT INTO lease (leaseID, vehicleID, customerID, startDate, endDate,type) "
                 "VALUES (%s, %s, %s, %s, %s, %s)")
        values = (new_lease_id, carID, customerID, startDate, endDate,type)
        cursor.execute(query, values)
        self.connection.commit()

        # Return the Lease object with the calculated leaseID
        return Lease(new_lease_id, carID, customerID, startDate, endDate,type)

    except Error as e:
        print(f"Error: Lease cannot be created")
    finally:
        cursor.close()
```

```python
def returnCar(self, leaseID: int):
    try:
        cursor = self.connection.cursor()
        query = "UPDATE lease SET endDate = CURRENT_DATE WHERE leaseID = %s"
        values = (leaseID,)
        cursor.execute(query, values)
        self.connection.commit()

        # Fetch the updated lease information
        query_select = "SELECT * FROM lease WHERE leaseID = %s"
        cursor.execute(query_select, values)
        result = cursor.fetchone()
        if result:
            return result
        else:
            raise Exception(f"Lease with ID {leaseID} not found")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

```
def listActiveLeases(self,date):
    cur = self.connection.cursor()
    query = "select * from Lease where endDate > '{}'".format(date)
    cur.execute(query)
    result = cur.fetchall()
    if result:
        for record in result:
            print(record)
        self.connection.commit()
        print("Active leases fetched ...")
    else:
        print("No records found for Active Leases..")
```

1 usage

```
def listLeaseHistory(self,date):
    cur = self.connection.cursor()
    query = "select * from Lease where endDate < '{}'".format(date)
    cur.execute(query)
    result = cur.fetchall()
    if result:
        for record in result:
            print(record)
        self.connection.commit()
        print("Lease History fetched ...")
    else:
        print("No records found for Lease History..")
```

```
def recordPayment(self, leaseID: int, amount: float):
    try:
        cursor = self.connection.cursor()
        query = "INSERT INTO payment (leaseID, paymentDate, amount) VALUES (%s, CURRENT_DATE, %s)"
        values = (leaseID, amount)
        cursor.execute(query, values)
        self.connection.commit()
        print("Payment recorded successfully!!!")
    except Error as e:
        print(f"Error: {e}")
    finally:
        cursor.close()
```

1 usage

```
def total_revenue(self):
    cursor = self.connection.cursor()
    query = "select SUM(amount) AS money FROM Payment"
    cursor.execute(query)
    result = cursor.fetchone()
    if result:
        amt = result[0]
    else:
        amt = 0
    cursor.close()
    print("Total Revenue is :", amt)
```

```
from dao.CarLeaseService import ICarLeaseRepository
from typing import List,Optional
from datetime import date

from entity import Vehicle,Customer,Lease

from myExeptions.Exception import CarNotFoundException
from myExeptions.Exception import CustomerNotFoundException
import mysql.connector


class CarService(ICarLeaseRepository):
    def __init__(self,connection):
        self.connection = connection

    def addCar(self, vehicle: Vehicle) -> None:
        cursor = self.connection.cursor()
        query = ("INSERT INTO Vehicle (make, model, year, dailyRate,"
                 " status, passengerCapacity, engineCapacity) "
                 "VALUES (%s, %s, %s, %s, %s, %s, %s)")
        data = (vehicle.make, vehicle.model, vehicle.year, vehicle.dailyRate,
                vehicle.status, vehicle.passengerCapacity,
                vehicle.engineCapacity)
        cursor.execute(query, data)
        self.connection.commit()
        cursor.close()
```

```python
def removeCar(self, vehicleID: int) -> None:
    cursor = self.connection.cursor()
    query = "DELETE FROM Vehicle WHERE vehicleID = %s"
    cursor.execute(query, (vehicleID))
    self.connection.commit()
    cursor.close()

def listAvailableCars(self) -> List[Vehicle]:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Vehicle WHERE status = 'available'"
    cursor.execute(query)
    vehicles = cursor.fetchall()
    cursor.close()
    return vehicles

def listRentedCars(self) -> List[Vehicle]:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Vehicle WHERE status = 'notAvailable'"
    cursor.execute(query)
    rented_cars = cursor.fetchall()
    cursor.close()
    return rented_cars
```

```python
def findCarById(self, vehicleID: int) -> Vehicle:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Vehicle WHERE vehicleID = %s"
    cursor.execute(query, (vehicleID,))
    vehicle = cursor.fetchone()
    cursor.close()
    if vehicle:
        return vehicle
    else:
        raise CarNotFoundException("Car not found with ID: {}".format(vehicleID))
```

```python
def addCustomer(self, customer: Customer) -> None:
    cursor = self.connection.cursor()
    query = ("INSERT INTO Customer (firstName, lastName, email, phoneNumber) "
             "VALUES (%s, %s, %s, %s)")
    data = (customer.firstName, customer.lastName,
            customer.email, customer.phoneNumber)
    cursor.execute(query, data)
    self.connection.commit()
    cursor.close()

def removeCustomer(self, customerID: int) -> None:
    cursor = self.connection.cursor()
    query = "DELETE FROM Customer WHERE customerID = %s"
    cursor.execute(query, (customerID,))
    self.connection.commit()
    cursor.close()
```

```
def listCustomers(self) -> List[Customer]:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Customer"
    cursor.execute(query)
    customers = cursor.fetchall()
    cursor.close()
    return customers

def findCustomerById(self, customerID: int) -> Customer:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Customer WHERE customerID = %s"
    cursor.execute(query, (customerID,))
    customer_data = cursor.fetchone()
    cursor.close()
    if customer_data:
        return customer_data
    else:
        raise CustomerNotFoundException("Customer not found with ID: {}".format(customerID))

def createLease(self, customerID: int, vehicleID: int,
                startDate: date, endDate: date) -> Optional[Lease]:
    cursor = self.connection.cursor()
    query = ("INSERT INTO Lease (vehicleID, customerID, startDate, endDate, type) "
             "VALUES (%s, %s, %s, %s, %s)")
    data = (vehicleID, customerID, startDate, endDate, type)
    cursor.execute(query, data)
    self.connection.commit()
    cursor.close()

def returnCar(self, leaseID: int) -> Optional[Lease]:
    cursor = self.connection.cursor()
    query = "UPDATE Lease SET endDate = CURDATE() WHERE leaseID = %s"
    cursor.execute(query, (leaseID,))
    self.connection.commit()
    cursor.close()

def listActiveLeases(self) -> List[Lease]:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Lease WHERE CURDATE() >= startDate AND CURDATE() <= endDate"
    cursor.execute(query)
    active_leases = cursor.fetchall()
    cursor.close()
    return active_leases

def listLeaseHistory(self) -> List[Lease]:
    cursor = self.connection.cursor()
    query = "SELECT * FROM Lease"
    cursor.execute(query)
    lease_history = cursor.fetchall()
    cursor.close()
    return lease_history

def recordPayment(self, lease: Lease, amount: float) -> None:
    cursor = self.connection.cursor()
    query = ("INSERT INTO Payment (leaseID, paymentDate, amount) VALUES (%s, %s, %s)")
    data = (lease, date, amount)
    cursor.execute(query, data)
    self.connection.commit()
    cursor.close()
```

Main Class:

Here, The Menu of Car Rental System:

```
from dao.ICarLeaseImpl import ICarLeaseRepositoryImpl
from entity.customer import Customer
from entity.vehicle import Vehicle

1 usage
def menu():

    print("1. Add Car")
    print("2. Remove Car")
    print("3. List Available Cars")
    print("4. List Rented Cars")
    print("5. Add Customer")
    print("6. Remove Customer")
    print("7. List Customers")
    print("8. Create Lease")
    print("9. List Active Leases")
    print("10. Lease History")
    print("11. Record Payment for a Lease")
    print("12. Return Car")
    print("13. Total Revenue")
    print("0. Exit")
```

```python
def main():
    # Replace with your actual MySQL connection details
    connection_params = {
        "host": "localhost",
        "user": "root",
        "password": "9927559686",
        "database": "CarRentalSystem"
    }

    car_repository = ICarLeaseRepositoryImpl(connection_params)

    while True:
        print("\n--- Welcome to Car Rental System------")
        menu()
        choice = input("Enter your choice (0-13): ")

        if choice == "1":
            new_car = Vehicle(vehicleID=int(input("Enter Vehicle ID: ")),
                              make=input("Enter Make: "),
                              model=input("Enter Model: "),
                              year=int(input("Enter Year: ")),
                              dailyRate=float(input("Enter Daily Rate: ")),
                              status=input("Enter Status (available/notAvailable): "),
                              passengerCapacity=int(input("Enter Passenger Capacity: ")),
                              engineCapacity=int(input("Enter Engine Capacity: ")))

            car_repository.addCar(new_car)
elif choice == "2":
    car_id = int(input("Enter the car ID:"))
    car_repository.removeCar(car_id)

elif choice == "3":
    available_cars = car_repository.listAvailableCars()
    print("Available Cars:")
    for car in available_cars:
        print(car)

elif choice == "4":
    rented_cars = car_repository.listRentedCars()
    print("Rented Cars:")
    for car in rented_cars:
        print(car)

elif choice == "5":
    new_customer = Customer(customerID=int(input("Enter Customer ID: ")),
                            firstName=input("Enter First Name: "),
                            lastName=input("Enter Last Name: "),
                            email=input("Enter Email: "),
                            phoneNumber=input("Enter Phone Number: "))

    car_repository.addCustomer(new_customer)
```

```
elif choice == "6":                                                    ⚠3 ⚠11
    cus_id = int(input("Enter the customer id: "))
    car_repository.removeCustomer(cus_id)

elif choice == "7":
    customers = car_repository.listCustomers()
    print("Customers:")
    for customer in customers:
        print(customer)

elif choice == "8":
    new_lease = car_repository.createLease(customerID=int(input("Enter Customer ID: ")),
                                           carID=int(input("Enter Car ID: ")),
                                           startDate=input("Enter Start Date date(yy-mm-dd): "),
                                           endDate=input("Enter End Date date(yy-mm-dd): "),
                                           type = input("Enter the type(monthly or daily): "))
    print("Lease created:", new_lease)

elif choice == "9":
   search_date = input("Enter today's date to search for Active Leases :")
   car_repository.listActiveLeases(search_date)

elif choice == "10":
    date = input("Enter today's date to search for Lease History:")
    car_repository.listLeaseHistory(date)

        elif choice == "11":
            lease_id = int(input("Enter Lease ID: "))
            payment_amount = float(input("Enter Payment Amount: "))
            car_repository.recordPayment(leaseID=lease_id, amount=payment_amount)
            print("Payment recorded successfully!!")

        elif choice == "12":
            lease_id = int(input("Enter Lease ID: "))
            returned_lease = car_repository.returnCar(leaseID=lease_id)
            print("Car returned. Updated Lease information:", returned_lease)

        elif choice == "13":
            car_repository.total_revenue()


        elif choice == "0":
            print("Signing offf Car Rental System. Thanks for Visiting..")
            car_repository.close_connection()
            break

        else:
            print("Invalid choice. Please enter a number between 0 and 13 ")

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

Unit Testing:

10. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system.

Following questions to guide the creation of Unit test cases:

• Write test case to test car created successfully or not.

• Write test case to test lease is created successfully or not.

```python
import unittest
from datetime import datetime
from entity.vehicle import Vehicle
from entity.lease import Lease
from dao.ICarLeaseImpl import ICarLeaseRepositoryImpl

class TestCarRentalSystem(unittest.TestCase):
    def setUp(self):
        # Replace with your actual MySQL connection details
        connection_params = {
            "host": "localhost",
            "user": "root",
            "password": "9927559686",
            "database": "CarRentalSystem"
        }
        self.car_repository = ICarLeaseRepositoryImpl(connection_params)

    def test_add_car(self):
        ''' new_car = Vehicle(vehicleID=17, make="Toyota", model="Corolla", year=2023, dailyRate=50.00,
                          status="available", passengerCapacity=5, engineCapacity=1500)
        self.car_repository.addCar(new_car)
        # Add assertions to verify if the car was added successfully
        self.assertEqual(new_car.vehicleID, 17)
        self.assertEqual(new_car.make, 'Toyota')
        self.assertEqual(new_car.model, 'Corolla')
        self.assertEqual(new_car.year, 2019)
        self.assertEqual(new_car.dailyRate, 50)
        self.assertEqual(new_car.status, 'available')
```

```
        self.assertEqual(new_car.passengerCapacity, 5)
        self.assertEqual(new_car.engineCapacity, 2)
'''
        new_car = Vehicle(vehicleID=23, make="Audi", model="6", year=2023, dailyRate=65.00,
                          status="available", passengerCapacity=4, engineCapacity=1500)
        self.car_repository.addCar(new_car)
        # Add assertions to verify if the car was added successfully
        self.assertEqual(new_car.vehicleID, second: 23)
        self.assertEqual(new_car.make, second: 'Audi')
        self.assertEqual(new_car.model, second: '6')
        self.assertEqual(new_car.year, second: 2023)
        self.assertEqual(new_car.dailyRate, second: 65)
        self.assertEqual(new_car.status, second: 'available')
        self.assertEqual(new_car.passengerCapacity, second: 4)
        self.assertEqual(new_car.engineCapacity, second: 1500)

    def test_lease_creation(self):
        new_lease = Lease(leaseID = 17, vehicleID = 8,customerID =8,
                          startDate = "2024-02-18", endDate = "2024-02-20", type = "daily")
        #lease_data = Lease(1, 1, 1, '2024-2-5', '2024-2-10', 'monthly')
        self.assertEqual(new_lease.leaseID, second: 17)
        self.assertEqual(new_lease.vehicleID, second: 8)
        self.assertEqual(new_lease.customerID, second: 8)
        self.assertEqual(new_lease.startDate, second: '2024-02-18')
        self.assertEqual(new_lease.endDate, second: '2024-02-20')
        self.assertEqual(new_lease.type, second: 'daily')


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

Output

```
Congratulations!!!
Database Connected Successfully..

--- Welcome to Car Rental System------
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Add Customer
6. Remove Customer
7. List Customers
8. Create Lease
9. List Active Leases
10. Lease History
11. Record Payment for a Lease
12. Return Car
13. Total Revenue
0. Exit
Enter your choice (0-13):


Enter your choice (0-13): 0
Signing offf Car Rental System. Thanks for Visiting..
Connection Over....

Process finished with exit code 0

 Enter your choice (0-13): 1
 Enter Vehicle ID: 19
 Enter Make: Mercedes
 Enter Model: Benz
 Enter Year: 2020
 Enter Daily Rate: 75
 Enter Status (available/notAvailable): available
 Enter Passenger Capacity: 2
 Enter Engine Capacity: 1500
 Car added successfully!!
```

```
Enter your choice (0-13): 2
Enter the car ID:19
Car removed successfully!!

Enter your choice (0-13): 3
Available Cars:
(1, 'Toyota', 'Camry', 2022, Decimal('50.00'), 'available', 4, 1450)
(4, 'Nissan', 'Altima', 2023, Decimal('52.00'), 'available', 7, 1200)
(5, 'Chevrolet', 'Malibu', 2022, Decimal('47.00'), 'available', 4, 1800)
(7, 'BMW', '3 Series', 2023, Decimal('60.00'), 'available', 7, 2499)
(8, 'Mercedes', 'C-Class', 2022, Decimal('68.00'), 'available', 8, 2599)
(10, 'Lexus', 'ES', 2023, Decimal('54.00'), 'available', 4, 2500)
(17, 'Toyota', 'Corolla', 2023, Decimal('50.00'), 'available', 5, 1500)
(18, 'maruti', 'suzuki', 2005, Decimal('55.00'), 'available', 4, 800)
(20, 'Audi', '5', 2023, Decimal('65.00'), 'available', 4, 1500)
(21, 'Audi', '5', 2023, Decimal('65.00'), 'available', 4, 1500)
(23, 'Audi', '6', 2023, Decimal('65.00'), 'available', 4, 1500)

Enter your choice (0-13): 4
Rented Cars:
(3, 'Ford', 'Focus', 2022, Decimal('48.00'), 'notAvailable', 4, 1400)
(6, 'Hyundai', 'Sonata', 2023, Decimal('49.00'), 'notAvailable', 7, 1400)
(9, 'Audi', 'A4', 2022, Decimal('55.00'), 'notAvailable', 4, 2500)
(15, 'Honda', 'City', 2015, Decimal('60.00'), 'notAvailable', 6, 1200)


Enter your choice (0-13): 5
Enter Customer ID: 21
Enter First Name: Satish
Enter Last Name: Shah
Enter Email: shah@abc.com
Enter Phone Number: 7845120369
Customer added successfully!!!

Enter your choice (0-13): 6
Enter the customer id: 21
Customer removed successfully!!
```

```
Enter your choice (0-13): 7
Customers:
(1, 'John', 'Doe', 'johndoe@example.com', '555-555-5555')
(3, 'Robert', 'Johnson', 'robert@example.com', '555-789-1234')
(4, 'Sarah', 'Brown', 'sarah@example.com', '555-456-7890')
(5, 'David', 'Lee', 'david@example.com', '555-987-6543')
(6, 'Laura', 'Hall', 'laura@example.com', '555-234-5678')
(7, 'Michael', 'Davis', 'michael@example.com', '555-876-5432')
(8, 'Emma', 'Wilson', 'emma@example.com', '555-432-1098')
(9, 'William', 'Taylor', 'william@example.com', '555-321-6547')
(10, 'Olivia', 'Adams', 'olivia@example.com', '555-765-4321')
(14, 'A', 'S', 'a@s.com', '7894512036')

Enter your choice (0-13): 9
Enter today's date to search for Active Leases :2024-02-15
(15, 18, 14, datetime.date(2024, 2, 10), datetime.date(2024, 2, 18), None)
Active leases fetched ...


Enter your choice (0-13): 10
Enter today's date to search for Lease History:2024-02-13
(1, 1, 1, datetime.date(2023, 1, 1), datetime.date(2023, 1, 5), 'DailyLease')
(3, 3, 3, datetime.date(2023, 3, 10), datetime.date(2023, 3, 15), 'DailyLease')
(4, 4, 4, datetime.date(2023, 4, 20), datetime.date(2023, 4, 30), 'MonthlyLease')
(5, 5, 5, datetime.date(2023, 5, 5), datetime.date(2023, 5, 10), 'DailyLease')
(6, 4, 3, datetime.date(2023, 6, 15), datetime.date(2023, 6, 30), 'MonthlyLease')
(7, 7, 7, datetime.date(2023, 7, 1), datetime.date(2023, 7, 10), 'DailyLease')
(8, 8, 8, datetime.date(2023, 8, 12), datetime.date(2023, 8, 15), 'MonthlyLease')
(9, 3, 3, datetime.date(2023, 9, 7), datetime.date(2023, 9, 10), 'DailyLease')
(10, 10, 10, datetime.date(2023, 10, 10), datetime.date(2023, 10, 31), 'MonthlyLease')
(11, 18, 14, datetime.date(2024, 2, 5), datetime.date(2024, 2, 8), None)
(12, 18, 14, datetime.date(2024, 2, 5), datetime.date(2024, 2, 8), None)
(13, 15, 14, datetime.date(2024, 2, 5), datetime.date(2024, 2, 10), None)
(14, 18, 14, datetime.date(2024, 2, 5), datetime.date(2024, 2, 8), None)
Lease History fetched ...

Enter your choice (0-13): 11
Enter Lease ID: 15
Enter Payment Amount: 4500
Error: 1054 (42S22): Unknown column 'paymentDate' in 'field list'
Payment recorded successfully!!
```

```
Enter your choice (0-13): 12
Enter Lease ID: 15
Car returned. Updated Lease information: (15, 18, 14, datetime.date(2024, 2, 10), datetime.date(2024, 2, 12), None)
```

```
Enter your choice (0-13): 13
Total Revenue is : 5155.00
```

```
============================= test session starts ==============================
collecting ... collected 2 items

testMain.py::TestCarRentalSystem::test_add_car
testMain.py::TestCarRentalSystem::test_lease_creation

============================== 2 passed in 0.37s ===============================
PASSED                    [ 50%]Congratulations!!!
Database Connected Successfully..
Error: 1062 (23000): Duplicate entry '23' for key 'vehicle.PRIMARY'
PASSED             [100%]Congratulations!!!
Database Connected Successfully..

Process finished with exit code 0
```