CASE STUDY

CAR RENTAL SYSTEM

SUBMITTED BY – ANUSHA P

INTRODUCTION:

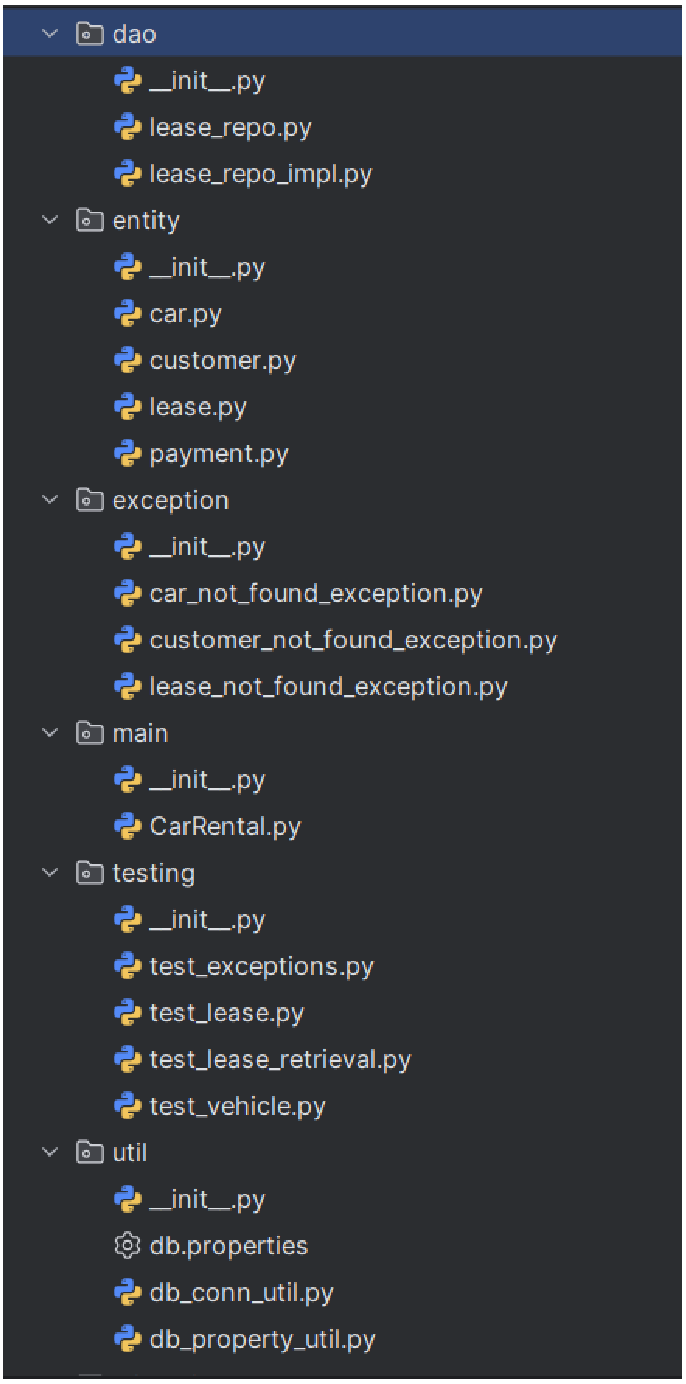
The Car Rental System is a Python-based application integrated with a MySQL database, designed to manage the core operations of a car rental business efficiently. It provides functionalities for adding and managing vehicles, customers, lease agreements, and payments through a user-friendly, menu-driven interface. The system follows a modular architecture with clearly separated packages for data models, database operations, utility functions, and exception handling. By incorporating object-oriented principles, custom exception management, and structured unit testing, this project offers a robust and scalable solution for real-world rental service automation.

OVERVIEW OF THE PROJECT:

The **Car Rental System** is a modular, console-based Python application that simulates the backend operations of a real-world car rental service. It is designed using object-oriented programming principles and integrated with a MySQL database for persistent data storage. The system is logically divided into the following core components:

- **Entity Layer**: This contains Python classes (Vehicle, Customer, Lease, and Payment) representing real-world entities. Each class includes private attributes, constructors (default and parameterized), and appropriate getter and setter methods.
- **DAO Layer (Data Access Object)**: All interactions with the MySQL database are handled here. An interface (ICarLeaseRepository) defines the expected database operations, and its implementation class (ICarLeaseRepositoryImpl) provides the actual logic for CRUD operations, lease handling, and payment management.
- **Util Layer**: This includes utility classes like DBPropertyUtil (to read database configuration from a .properties file) and DBConnUtil (to establish database connections based on configuration).
- **Exception Layer**: Custom exception classes like CarNotFoundException, CustomerNotFoundException, and LeaseNotFoundException are defined to handle invalid input or missing records gracefully.

- **Main Program**: The user interface is menu-driven and categorized into sections such as Car Management, Customer Management, Lease Management, and Payment Handling. Each section provides sub-options to perform operations like adding, searching, removing, and listing data.
- **Testing**: Unit tests are written in separate files using Python's unittest module to validate core functionalities like adding a car, creating a lease, retrieving records, and ensuring proper exception handling.



```
dao
    __init__.py
    lease_repo.py
    lease_repo_impl.py
entity
    __init__.py
    car.py
    customer.py
    lease.py
    payment.py
exception
    __init__.py
    car_not_found_exception.py
    customer_not_found_exception.py
    lease_not_found_exception.py
main
    __init__.py
    CarRental.py
testing
    __init__.py
    test_exceptions.py
    test_lease.py
    test_lease_retrieval.py
    test_vehicle.py
util
    __init__.py
    db.properties
    db_conn_util.py
    db_property_util.py
```

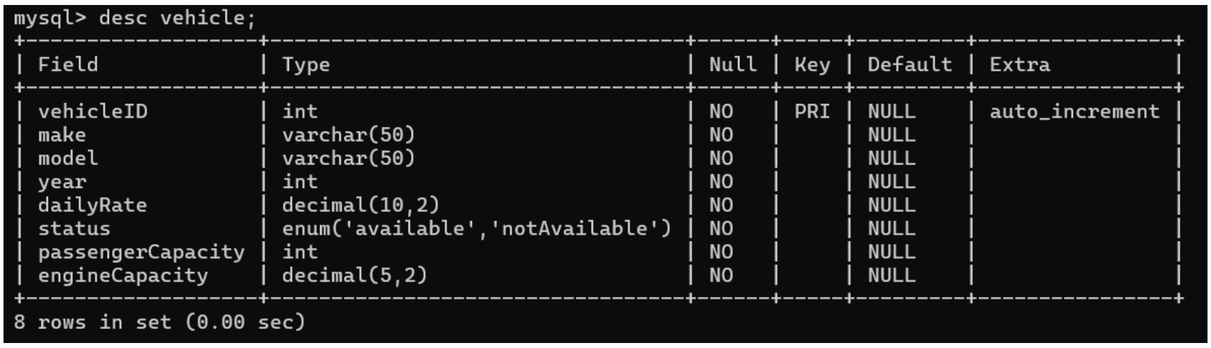
SCHEMA DESIGN :

The database schema consists of four main tables: Vehicle, Customer, Lease, and Payment. These are related through primary and foreign keys to maintain data integrity and ensure normalized relational design.

TABLE CREATION:

1. Vehicle Table

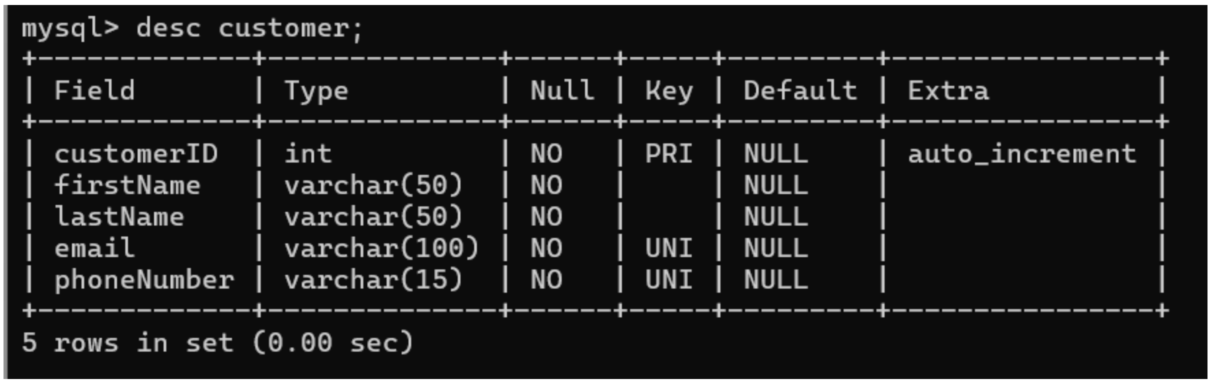
```
create table vehicle (
    vehicleID int primary key auto_increment,
    make varchar(50) not null,
    model varchar(50) not null,
    year int not null,
    dailyrate decimal(10,2) not null,
    status enum('available', 'notavailable') not null,
    passengercapacity int not null,
    enginecapacity decimal(5,2) not null
);
```



```
mysql> desc vehicle;
+-------------------+----------------------------------+------+-----+---------+----------------+
| Field             | Type                             | Null | Key | Default | Extra          |
+-------------------+----------------------------------+------+-----+---------+----------------+
| vehicleID         | int                              | NO   | PRI | NULL    | auto_increment |
| make              | varchar(50)                      | NO   |     | NULL    |                |
| model             | varchar(50)                      | NO   |     | NULL    |                |
| year              | int                              | NO   |     | NULL    |                |
| dailyRate         | decimal(10,2)                    | NO   |     | NULL    |                |
| status            | enum('available','notAvailable') | NO   |     | NULL    |                |
| passengerCapacity | int                              | NO   |     | NULL    |                |
| engineCapacity    | decimal(5,2)                     | NO   |     | NULL    |                |
+-------------------+----------------------------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

2.Customer Table

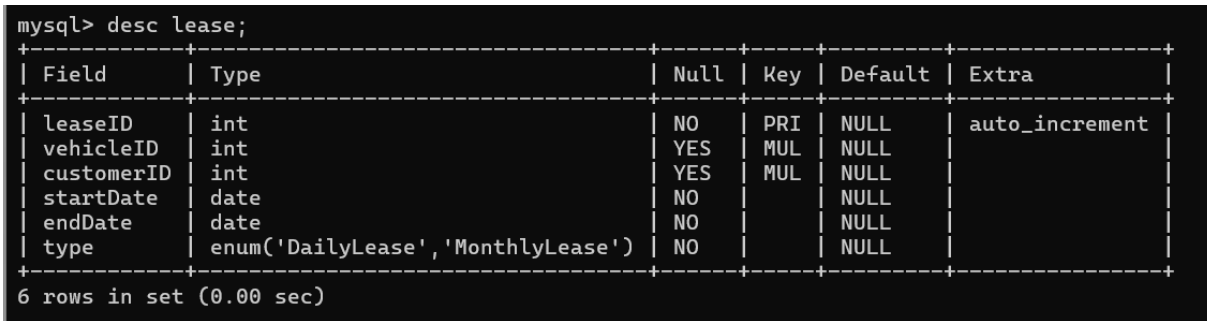
```
create table customer (
    customerID int primary key auto_increment,
    firstname varchar(50) not null,
    lastname varchar(50) not null,
    email varchar(100) unique not null,
    phonenumber varchar(15) unique not null
);
```



```
mysql> desc customer;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| customerID  | int          | NO   | PRI | NULL    | auto_increment |
| firstName   | varchar(50)  | NO   |     | NULL    |                |
| lastName    | varchar(50)  | NO   |     | NULL    |                |
| email       | varchar(100) | NO   | UNI | NULL    |                |
| phoneNumber | varchar(15)  | NO   | UNI | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

3.Lease Table

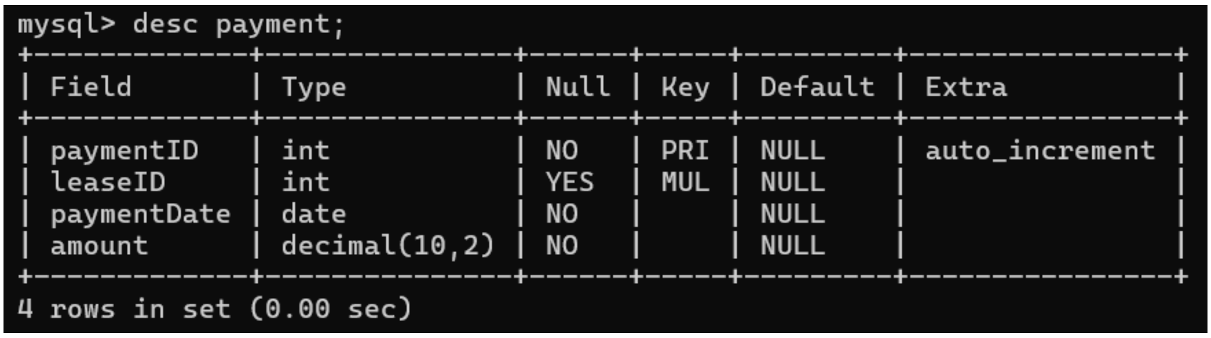
```
create table lease (
    leaseID int primary key auto_increment,
    vehicleID int,
    customerID int,
    startdate date not null,
    enddate date not null,
    type enum('dailylease', 'monthlylease') not null,
    foreign key (vehicleID) references vehicle(vehicleid),
    foreign key (customerID) references customer(customerid)
);
```



```
mysql> desc lease;
+------------+-----------------------------------+------+-----+---------+----------------+
| Field      | Type                              | Null | Key | Default | Extra          |
+------------+-----------------------------------+------+-----+---------+----------------+
| leaseID    | int                               | NO   | PRI | NULL    | auto_increment |
| vehicleID  | int                               | YES  | MUL | NULL    |                |
| customerID | int                               | YES  | MUL | NULL    |                |
| startDate  | date                              | NO   |     | NULL    |                |
| endDate    | date                              | NO   |     | NULL    |                |
| type       | enum('DailyLease','MonthlyLease') | NO   |     | NULL    |                |
+------------+-----------------------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

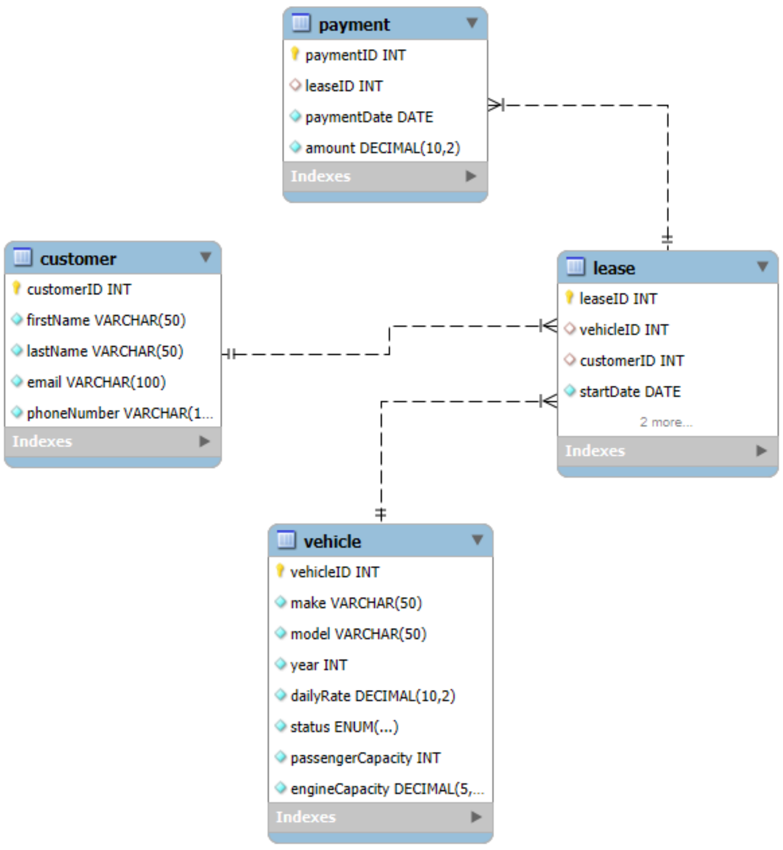
4.Payment Table

```
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY AUTO_INCREMENT,
    leaseID INT,
    paymentDate DATE NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (leaseID) REFERENCES Lease(leaseID)
);
```



```
mysql> desc payment;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| paymentID   | int           | NO   | PRI | NULL    | auto_increment |
| leaseID     | int           | YES  | MUL | NULL    |                |
| paymentDate | date          | NO   |     | NULL    |                |
| amount      | decimal(10,2) | NO   |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

ENTITY RELATIONSHIP DIAGRAM:



Key Functionalities:

1.Customer Management:

- Add new customers
- Update customer information
- Retrieve customer details.

2.Car Management:

- Add new cars to the system
- Update car availability
- Retrieve car information.

3.Lease Management:

- Create daily or monthly leases for customers.
- Calculate the total cost of a lease based on the type (Daily or Monthly) and the number of days or months.

4. Payment Handling:

- Record payments for leases.
- Retrieve payment history for a customer.
- Calculate the total revenue from payments.

IMPLEMENTATION – PYTHON PART:

1. ENTITY PACKAGE:

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

Car.py

```
class Vehicle:
    def __init__(self, vehicleID=None, make=None, model=None, year=None,
dailyRate=None, status=None, passengerCapacity=None,
engineCapacity=None):
        self.__vehicleID = vehicleID
        self.__make = make
        self.__model = model
        self.__year = year
        self.__dailyRate = dailyRate
        self.__status = status
        self.__passengerCapacity = passengerCapacity
        self.__engineCapacity = engineCapacity

    # Getters and setters
    def get_vehicleID(self):
        return self.__vehicleID

    def get_make(self):
        return self.__make

    def get_model(self):
        return self.__model
```

```
def get_year(self):
    return self.__year

def get_dailyRate(self):
    return self.__dailyRate

def get_status(self):
    return self.__status

def get_passengerCapacity(self):
    return self.__passengerCapacity

def get_engineCapacity(self):
    return self.__engineCapacity

def set_vehicleID(self, vehicleID):
    self.__vehicleID = vehicleID

def set_make(self, make):
    self.__make = make

def set_model(self, model):
    self.__model = model

def set_year(self, year):
    self.__year = year

def set_dailyRate(self, dailyRate):
    self.__dailyRate = dailyRate

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

def set_passengerCapacity(self, passengerCapacity):
    self.__passengerCapacity = passengerCapacity

def set_engineCapacity(self, engineCapacity):
        self.__engineCapacity = engineCapacity

    def __str__(self):
        return (f"Vehicle[ID={self.__vehicleID}, Make={self.__make}, 
Model={self.__model}, "
                f"Year={self.__year}, Rate={self.__dailyRate}, 
Status={self.__status}, "
                f"Passengers={self.__passengerCapacity}, 
Engine={self.__engineCapacity}L]")
```

Customer.py

```
class Customer:
    def __init__(self, customerID=None, firstName=None, lastName=None, 
email=None, phoneNumber=None):
        self.__customerID = customerID
        self.__firstName = firstName
        self.__lastName = lastName
        self.__email = email
        self.__phoneNumber = phoneNumber

    def get_customerID(self):
        return self.__customerID

    def get_firstName(self):
        return self.__firstName

    def get_lastName(self):
        return self.__lastName

    def get_email(self):
        return self.__email

    def get_phoneNumber(self):
        return self.__phoneNumber

    def set_customerID(self, customerID):
        self.__customerID = customerID
    def set_firstName(self, firstName):
        self.__firstName = firstName

    def set_lastName(self, lastName):
        self.__lastName = lastName

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

    def set_phoneNumber(self, phoneNumber):
        self.__phoneNumber = phoneNumber
```

Lease.py

```
class Lease:
    def __init__(self, leaseID=None, vehicleID=None, customerID=None, 
startDate=None, endDate=None, type=None):
        self.__leaseID = leaseID
        self.__vehicleID = vehicleID
        self.__customerID = customerID
        self.__startDate = startDate
        self.__endDate = endDate
        self.__type = type

    def get_leaseID(self):
        return self.__leaseID

    def get_vehicleID(self):
        return self.__vehicleID

    def get_customerID(self):
        return self.__customerID

    def get_startDate(self):
        return self.__startDate

    def get_endDate(self):
        return self.__endDate

    def get_type(self):
        return self.__type

    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

    def set_vehicleID(self, vehicleID):
        self.__vehicleID = vehicleID

    def set_customerID(self, customerID):
        self.__customerID = customerID

    def set_startDate(self, startDate):
        self.__startDate = startDate

    def set_endDate(self, endDate):
        self.__endDate = endDate

    def set_type(self, type):
        self.__type = type
```

Payment.py

```
class Payment:
    def __init__(self, paymentID=None, leaseID=None, paymentDate=None,
amount=None):
        self.__paymentID = paymentID
        self.__leaseID = leaseID
        self.__paymentDate = paymentDate
        self.__amount = amount

    def get_paymentID(self):
        return self.__paymentID

    def get_leaseID(self):
        return self.__leaseID
    def get_paymentDate(self):
        return self.__paymentDate

    def get_amount(self):
        return self.__amount

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

    def set_leaseID(self, leaseID):
        self.__leaseID = leaseID

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

    def set_amount(self, amount):
        self.__amount = amount
```

2.DAO (Data Access Object) package:

Create Interface for ICarLeaseRepository and add following methods which interact with database.

lease_repo.py

```
from abc import ABC, abstractmethod
from typing import List
from entity.car import Vehicle
from entity.customer import Customer
from entity.lease import Lease

class ICarLeaseRepository(ABC):

    # --- Car Management ---
    @abstractmethod
    def addCar(self, car: Vehicle) -> None:
        pass
```

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

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

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

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

# --- Customer Management ---
@abstractmethod
def addCustomer(self, customer: Customer) -> None:
    pass

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

@abstractmethod
def updateCustomer(self, customer: Customer):
    pass

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

@abstractmethod
def findCustomerById(self, customerID: int) -> Customer:
    pass
    # --- Lease Management ---
    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate, endDate) ->
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 ---
    @abstractmethod
    def recordPayment(self, lease: Lease, amount: float) -> None:
        pass
```

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

lease_repo_impl.py :

```
from typing import List
from dao.lease_repo import ICarLeaseRepository
from entity.car import Vehicle
from entity.customer import Customer
from entity.lease import Lease
from util.db_conn_util import DBConnUtil
from exception.lease_not_found_exception import LeaseNotFoundException
from exception.car_not_found_exception import CarNotFoundException
from exception.customer_not_found_exception import
CustomerrNotFoundException
```

```
class ICarLeaseRepositoryImpl(ICarLeaseRepository):

    def __init__(self):
        self.conn =
DBConnUtil.get_connection(r'C:/Users/anush/PycharmProjects/Car Rental
System/util/db.properties')

    # --- Car Management ---
    def addCar(self, car: Vehicle) -> None:
        cursor = self.conn.cursor()
        sql = """
            INSERT INTO Vehicle (make, model, year, dailyRate, status,
passengerCapacity, engineCapacity)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(sql, (car.get_make(), car.get_model(), car.get_year(),
car.get_dailyRate(),
                             car.get_status(), car.get_passengerCapacity(),
car.get_engineCapacity()))
        self.conn.commit()

    def removeCar(self, carID: int) -> None:
        cursor = self.conn.cursor()
        cursor.execute("DELETE FROM Vehicle WHERE vehicleID = %s",
(carID,))
        self.conn.commit()

    def listAvailableCars(self) -> List[Vehicle]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Vehicle WHERE status = 'available'")
        rows = cursor.fetchall()
        # return [row for row in rows]
        return [Vehicle(*row) for row in rows]

    def listRentedCars(self) -> List[Vehicle]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Vehicle WHERE status = 
'notAvailable'")
        rows = cursor.fetchall()
        return [Vehicle(*row) for row in rows]

    def findCarById(self, carID: int) -> Vehicle:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Vehicle WHERE vehicleID = %s", 
(carID,))
        row = cursor.fetchone()
        if row:
            return Vehicle(*row)
        else:
            raise CarNotFoundException("Car not found")

    # --- Customer Management ---
    def addCustomer(self, customer: Customer) -> None:
        cursor = self.conn.cursor()
        sql = "INSERT INTO Customer (firstName, lastName, email, 
phoneNumber) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, (customer.get_firstName(), customer.get_lastName(),
                             customer.get_email(), customer.get_phoneNumber()))
        self.conn.commit()

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

    def updateCustomer(self, customer: Customer):
        cursor = self.conn.cursor()
        try:
            query = """UPDATE Customer
                       SET firstName=%s, lastName=%s, email=%s, phoneNumber=%s
                       WHERE customerID=%s"""
            cursor.execute(query, (
            customer.get_firstName(), customer.get_lastName(),
customer.get_email(), customer.get_phoneNumber(),
customer.get_customerID()))
            self.conn.commit()
            if cursor.rowcount == 0:
                raise CustomerrNotFoundException(f"Customer with ID
{customer.get_customerID()} not found.")
        finally:
            cursor.close()
            self.conn.close()

    def listCustomers(self) -> List[Customer]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Customer")
        rows = cursor.fetchall()
        return [Customer(*row) for row in rows]

    def findCustomerById(self, customerID: int) -> Customer:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Customer WHERE customerID = %s",
(customerID,))
        row = cursor.fetchone()
        if row:
            return Customer(*row)
        else:
            raise CustomerrNotFoundException("Customer not found")

    # --- Lease Management ---
    def createLease(self, customerID: int, carID: int, startDate, endDate) ->
Lease:
        cursor = self.conn.cursor()
        cursor.execute("""
            INSERT INTO Lease (vehicleID, customerID, startDate, endDate, type)
            VALUES (%s, %s, %s, %s, %s)
        """, (carID, customerID, startDate, endDate, 'DailyLease'))
        self.conn.commit()
        lease_id = cursor.lastrowid
        return Lease(lease_id, carID, customerID, startDate, endDate,
'DailyLease')

    def returnCar(self, leaseID: int) -> Lease:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Lease WHERE leaseID = %s",
(leaseID,))
        row = cursor.fetchone()
        if not row:
            raise LeaseNotFoundException("Lease not found")
        lease = Lease(*row)
        cursor.execute("UPDATE Vehicle SET status = 'available' WHERE
vehicleID = %s", (lease.get_vehicleID(),))
        self.conn.commit()
        return lease

    def listActiveLeases(self) -> List[Lease]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Lease WHERE endDate >=
CURDATE()")
        rows = cursor.fetchall()
        return [Lease(*row) for row in rows]

    def listLeaseHistory(self) -> List[Lease]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Lease")
        rows = cursor.fetchall()
        return [Lease(*row) for row in rows]

    # --- Payment Handling ---
    def recordPayment(self, lease: Lease, amount: float) -> None:
        cursor = self.conn.cursor()
        sql = "INSERT INTO Payment (leaseID, paymentDate, amount) VALUES
(%s,curdate(), %s)"
        cursor.execute(sql, (lease.get_leaseID(), amount))
        self.conn.commit()
```

3.UTIL PACKAGE(DATABASE CONNECTION):

- **Connect your application to the SQL database and write code to establish a connection to your SQL database. 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.**

db_conn_util.py:

```
import mysql.connector
from util.db_property_util import DBPropertyUtil

class DBConnUtil:
    @staticmethod
    def get_connection(prop_file_name: str):
        try:
            # Get the full connection string
            conn_str = DBPropertyUtil.get_connection_string(prop_file_name)

            # Parse the connection string into a dictionary
            conn_params = {}
            for item in conn_str.split(';'):
                if '=' in item:
                    key, value = item.split('=', 1)
                    conn_params[key.strip()] = value.strip()

            # Connect to the MySQL database
            conn = mysql.connector.connect(
                host=conn_params.get('host'),
                user=conn_params.get('user'),
                password=conn_params.get('password'),
                database=conn_params.get('database')
            )
            return conn

        except mysql.connector.Error as err:
            print(f"Database connection error: {err}")
            return None
        except Exception as e:
            print(f"Unexpected error: {e}")
            return None
```

db_property_util.py:

```
class DBPropertyUtil:
    @staticmethod
    def get_connection_string(prop_file_name: str) -> str:
        props = {}
        try:
            with open(prop_file_name, 'r') as file:
                for line in file:
                    line = line.strip()
                    if line and not line.startswith('#'):
                        key_value = line.split('=')
                        if len(key_value) == 2:
                            key, value = key_value
                            props[key.strip()] = value.strip()
        except FileNotFoundError:
            print(f"Property file '{prop_file_name}' not found.")
        except Exception as e:
            print(f"Error reading property file: {e}")

        # Build connection string from properties
        connection_string = (
            f"host={props.get('host')};"
            f"user={props.get('user')};"
            f"password={props.get('password')};"
            f"database={props.get('database')}"
        )
        return connection_string
```

4.EXCEPTION PACKAGE:

Create the exceptions in package **myexceptions** and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

- **CarNotFoundException**: throw this exception when user enters an invalid car id which doesn’t exist in db.
- **LeaseNotFoundException**: throw this exception when user enters an invalid lease id which doesn’t exist in db.
- **CustomerrNotFoundException**: throw this exception when user enters an invalid customer id which doesn’t exist in db.

car_not_found_exception.py:

```
class CarNotFoundException(Exception):
    def __init__(self, message="Car with the given ID was not found."):
        super().__init__(message)
```

customer_not_found_exception.py:

```
class CustomerrNotFoundException(Exception):
    def __init__(self, message="Customer with the given ID was not found."):
        super().__init__(message)
```

lease_not_found_exception.py:

```
class LeaseNotFoundException(Exception):
    def __init__(self, message="Lease with the given ID was not found."):
        super().__init__(message)
```

5.MAIN PACKAGE:

Carrental.py:

```
from util.db_conn_util import DBConnUtil

from dao.lease_repo_impl import ICarLeaseRepositoryImpl
from entity.car import Vehicle
from entity.customer import Customer
from exception.car_not_found_exception import  CarNotFoundException
from exception.customer_not_found_exception import
CustomerrNotFoundException
```

```
from exception.lease_not_found_exception import  LeaseNotFoundException
from datetime import date


def main():
    repo = ICarLeaseRepositoryImpl()

    while True:
        print("\n===== Car Rental System Menu =====")
        print("1. Car Management")
        print("2. Customer Management")
        print("3. Lease Management")
        print("4. Payment Handling")
        print("5. Exit")
        category = input("Enter your choice: ")

        try:
            if category == "1":
                print("\n--- Car Management ---")
                print("1. Add Car")
                print("2. Remove Car")
                print("3. List Available Cars")
                print("4. List Rented Cars")
                print("5. Find Car by ID")
                sub_choice = input("Enter your choice: ")

                if sub_choice == "1":
                    make = input("Enter make: ")
                    model = input("Enter model: ")
                    year = int(input("Enter year: "))
                    rate = float(input("Enter daily rate: "))
                    status = "available"
                    passenger_capacity = int(input("Enter passenger capacity: "))
                    engine_capacity = float(input("Enter engine capacity: "))
                    car = Vehicle(None, make, model, year, rate, status,
passenger_capacity, engine_capacity)
                    repo.addCar(car)
                    print("Car added successfully.")

                elif sub_choice == "2":
                    car_id = int(input("Enter car ID to remove: "))
                    repo.removeCar(car_id)
                    print("Car removed successfully.")

                elif sub_choice == "3":
                    cars = repo.listAvailableCars()
                    for car in cars:
                        print(car)

                elif sub_choice == "4":
                    cars = repo.listRentedCars()
                    for car in cars:
                        print(car)

                elif sub_choice == "5":
                    car_id = int(input("Enter car ID: "))
                    car = repo.findCarById(car_id)
                    print(car)

                else:
                    print("Invalid choice.")

            elif category == "2":
                print("\n--- Customer Management ---")
                print("1. Add Customer")
                print("2. Remove Customer")
                print("3. List Customers")
                print("4. Find Customer by ID")
                print("5. Update Customer")
                sub_choice = input("Enter your choice: ")

                if sub_choice == "1":
                    first_name = input("Enter first name: ")
                    last_name = input("Enter last name: ")
                    email = input("Enter email: ")
                    phone = input("Enter phone number: ")
                    customer = Customer(None, first_name, last_name, email, phone)
                    repo.addCustomer(customer)
                    print("Customer added successfully.")

                elif sub_choice == "2":
                    customer_id = int(input("Enter customer ID to remove: "))
                    repo.removeCustomer(customer_id)
                    print("Customer removed successfully.")

                elif sub_choice == "3":
                    customers = repo.listCustomers()
                    for cust in customers:
                        print(cust)

                elif sub_choice == "4":
                    customer_id = int(input("Enter customer ID: "))
                    customer = repo.findCustomerById(customer_id)
                    print(customer)

                elif sub_choice == "5":
                    customer_id = int(input("Enter customer ID to update: "))
                    first_name = input("Enter new first name: ")
                    last_name = input("Enter new last name: ")
                    email = input("Enter new email: ")
                    phone = input("Enter new phone number: ")
                    customer = Customer(customer_id, first_name, last_name, email,
        phone)
                    repo.updateCustomer(customer)
                    print("Customer information updated successfully.")

                else:
                    print("Invalid choice.")

            elif category == "3":
                print("\n--- Lease Management ---")
                print("1. Create Lease")
                print("2. Return Car")
                print("3. List Active Leases")
                print("4. List Lease History")
                sub_choice = input("Enter your choice: ")

                if sub_choice == "1":
                    customer_id = int(input("Enter customer ID: "))
                    car_id = int(input("Enter car ID: "))
                    start_date = input("Enter lease start date (YYYY-MM-DD): ")
                    end_date = input("Enter lease end date (YYYY-MM-DD): ")
                    lease = repo.createLease(customer_id, car_id,
        date.fromisoformat(start_date), date.fromisoformat(end_date))
                    print("Lease created successfully:", lease)

                elif sub_choice == "2":
                    lease_id = int(input("Enter lease ID to return car: "))
                    lease = repo.returnCar(lease_id)
                    print("Car returned successfully:", lease)

                elif sub_choice == "3":
                    leases = repo.listActiveLeases()
                    for lease in leases:
                        print(lease)

                elif sub_choice == "4":
                    leases = repo.listLeaseHistory()
                    for lease in leases:
                        print(lease)

                else:
                    print("Invalid choice.")

            elif category == "4":
                print("\n--- Payment Handling ---")
                lease_id = int(input("Enter lease ID: "))
                amount = float(input("Enter payment amount: "))
```

```
            lease = repo.findLeaseById(lease_id)
            repo.recordPayment(lease, amount)
            print("Payment recorded successfully.")

        elif category == "5":
            print("Exiting... Goodbye!")
            break

        else:
            print("Invalid category. Please try again.")

    except CarNotFoundException as e:
        print(f"Error: {e}")

    except CustomerrNotFoundException as e:
        print(f"Error: {e}")

    except LeaseNotFoundException as e:
        print(f"Error: {e}")

    except Exception as e:
        print(f"Unexpected error: {e}")


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

6.TESTING PACKAGE:

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.**
- **Write test case to test lease is retrieved successfully or not.**

- **Write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database.**

test_exceptions.py:

```
import unittest
from dao.lease_repo_impl import ICarLeaseRepositoryImpl
from exception.car_not_found_exception import CarNotFoundException
from exception.customer_not_found_exception import
CustomerrNotFoundException
from exception.lease_not_found_exception import LeaseNotFoundException

class TestExceptionHandling(unittest.TestCase):
    def setUp(self):
        self.repo = ICarLeaseRepositoryImpl()

    def test_car_not_found(self):
        with self.assertRaises(CarNotFoundException):
            self.repo.findCarById(-1)

    def test_customer_not_found(self):
        with self.assertRaises(CustomerrNotFoundException):
            self.repo.findCustomerById(-1)

    def test_lease_not_found(self):
        with self.assertRaises(LeaseNotFoundException):
            self.repo.returnCar(-1)

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

test_lease.py

```
import unittest
from dao.lease_repo_impl import ICarLeaseRepositoryImpl
from entity.car import Vehicle
from entity.customer import Customer
from datetime import date
```

```
class TestLeaseFunctions(unittest.TestCase):
    def setUp(self):
        self.repo = ICarLeaseRepositoryImpl()

    def test_create_lease_success(self):
        customer = Customer(None, "Lease", "User", "lease@example.com",
"9876543210")
        self.repo.addCustomer(customer)
        customer_id = self.repo.listCustomers()[-1].get_customerID()

        car = Vehicle(None, "LeaseMake", "LeaseModel", 2023, 90.0, "available",
4, 2.0)
        self.repo.addCar(car)
        car_id = self.repo.listAvailableCars()[-1].get_vehicleID()

        lease = self.repo.createLease(customer_id, car_id, date.today(),
date.today())
        self.assertIsNotNone(lease)

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

test_lease_retrieval.py:

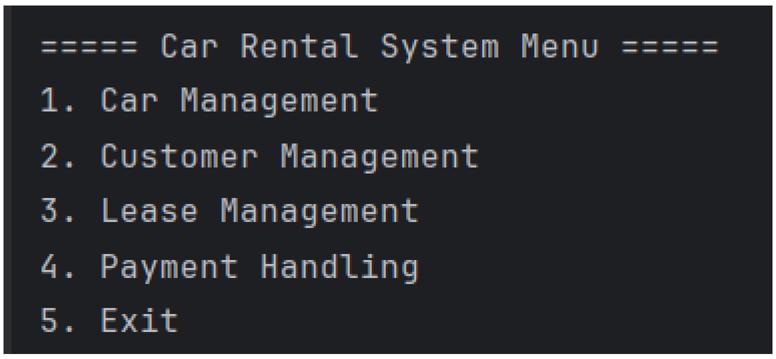
```
import unittest
from dao.lease_repo_impl import ICarLeaseRepositoryImpl

class TestLeaseRetrieval(unittest.TestCase):
    def setUp(self):
        self.repo = ICarLeaseRepositoryImpl()

    def test_retrieve_leases_success(self):
        leases = self.repo.listActiveLeases()
        self.assertIsInstance(leases, list)
        if leases:
            self.assertTrue(hasattr(leases[0], "get_leaseID"))
```

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

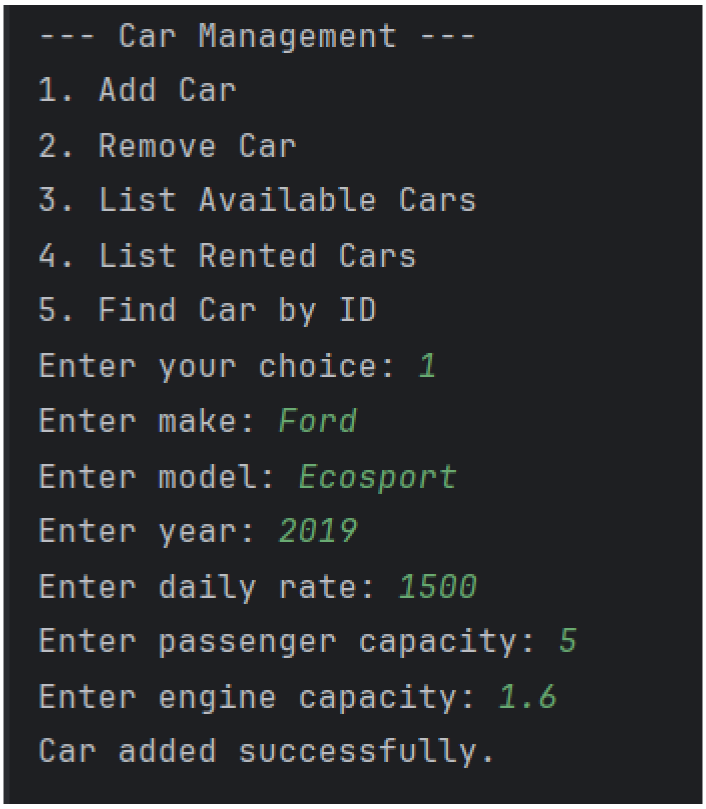
test_vehicle.py:

```
import unittest
from dao.lease_repo_impl import ICarLeaseRepositoryImpl
from entity.car import Vehicle

class TestVehicleFunctions(unittest.TestCase):
    def setUp(self):
        self.repo = ICarLeaseRepositoryImpl()

    def test_add_car_success(self):
        car = Vehicle(None, "TestMake", "TestModel", 2024, 75.0, "available", 4,
2.2)
        self.repo.addCar(car)
        cars = self.repo.listAvailableCars()
        self.assertTrue(any(c.get_make() == "TestMake" and c.get_model() ==
"TestModel" for c in cars))

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

OUTPUT:

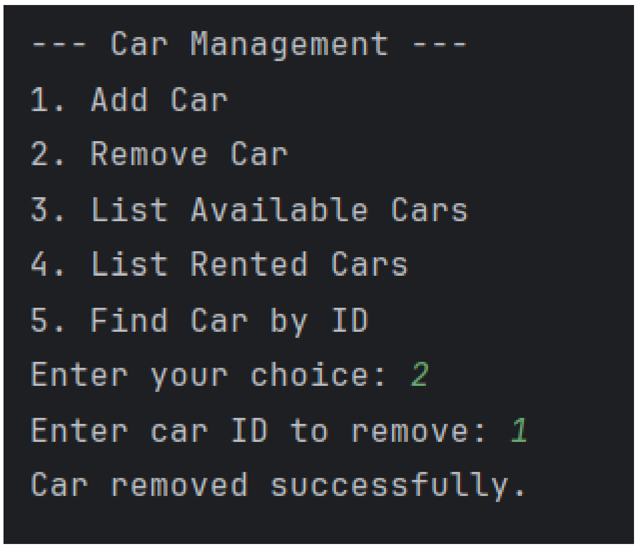


```
===== Car Rental System Menu =====
1. Car Management
2. Customer Management
3. Lease Management
4. Payment Handling
5. Exit
```

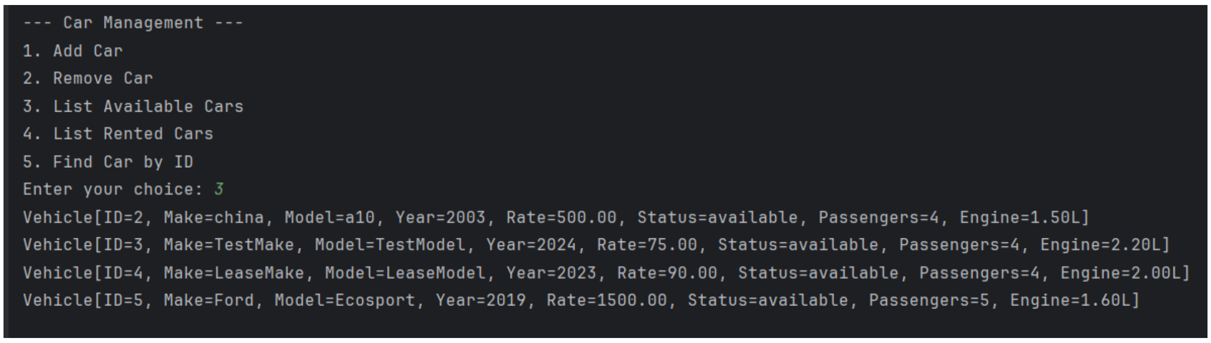


```
--- Car Management ---
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
Enter your choice: 1
Enter make: Ford
Enter model: Ecosport
Enter year: 2019
Enter daily rate: 1500
Enter passenger capacity: 5
Enter engine capacity: 1.6
Car added successfully.
```

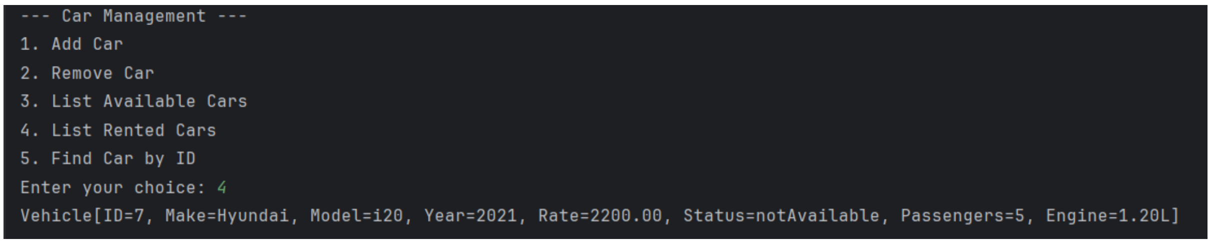
1.Car Management:



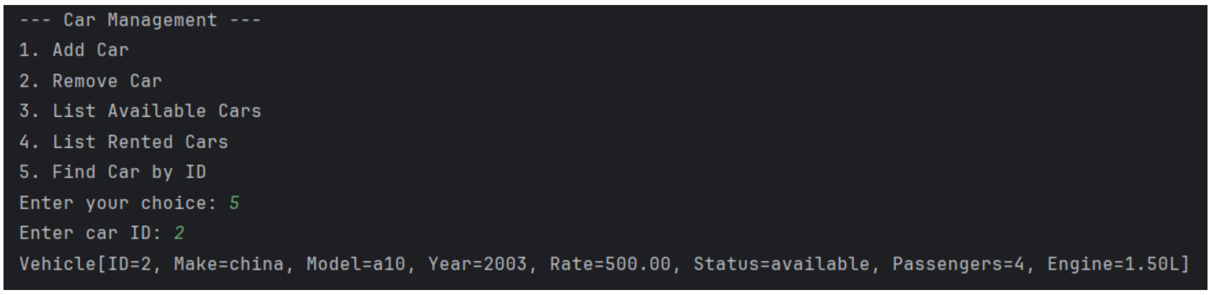
```
--- Car Management ---
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
Enter your choice: 2
Enter car ID to remove: 1
Car removed successfully.
```



```
--- Car Management ---
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
Enter your choice: 3
Vehicle[ID=2, Make=china, Model=a10, Year=2003, Rate=500.00, Status=available, Passengers=4, Engine=1.50L]
Vehicle[ID=3, Make=TestMake, Model=TestModel, Year=2024, Rate=75.00, Status=available, Passengers=4, Engine=2.20L]
Vehicle[ID=4, Make=LeaseMake, Model=LeaseModel, Year=2023, Rate=90.00, Status=available, Passengers=4, Engine=2.00L]
Vehicle[ID=5, Make=Ford, Model=Ecosport, Year=2019, Rate=1500.00, Status=available, Passengers=5, Engine=1.60L]
```

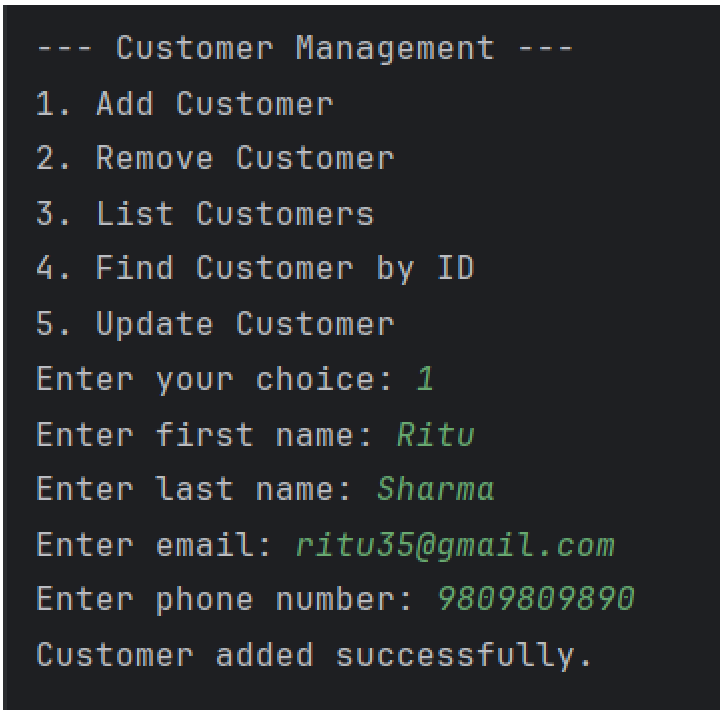


```
--- Car Management ---
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
Enter your choice: 4
Vehicle[ID=7, Make=Hyundai, Model=i20, Year=2021, Rate=2200.00, Status=notAvailable, Passengers=5, Engine=1.20L]
```

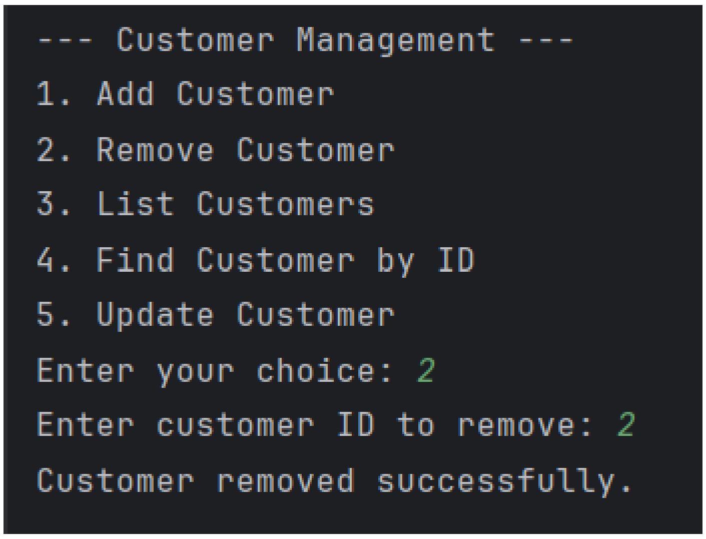


```
--- Car Management ---
1. Add Car
2. Remove Car
3. List Available Cars
4. List Rented Cars
5. Find Car by ID
Enter your choice: 5
Enter car ID: 2
Vehicle[ID=2, Make=china, Model=a10, Year=2003, Rate=500.00, Status=available, Passengers=4, Engine=1.50L]
```

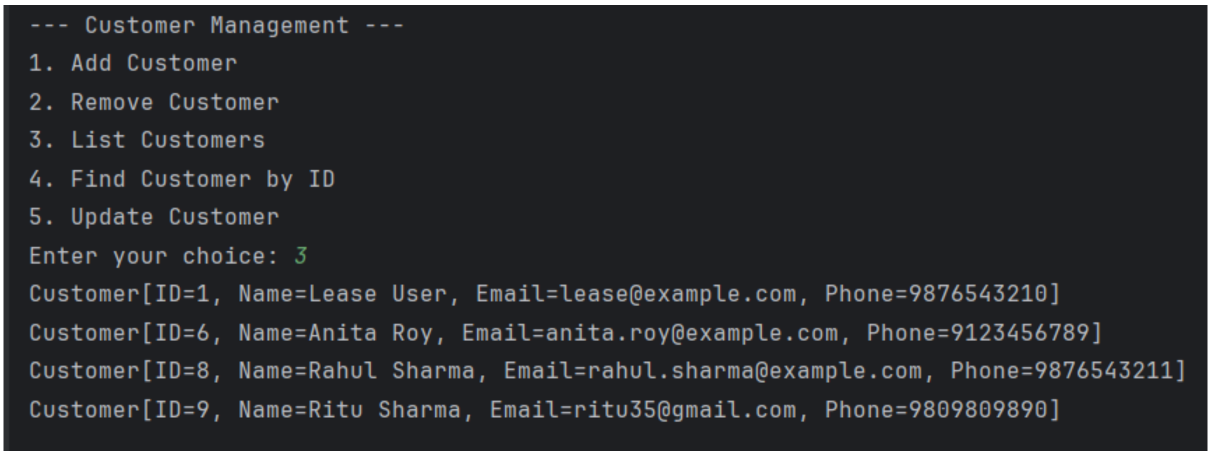
2.Customer Management:



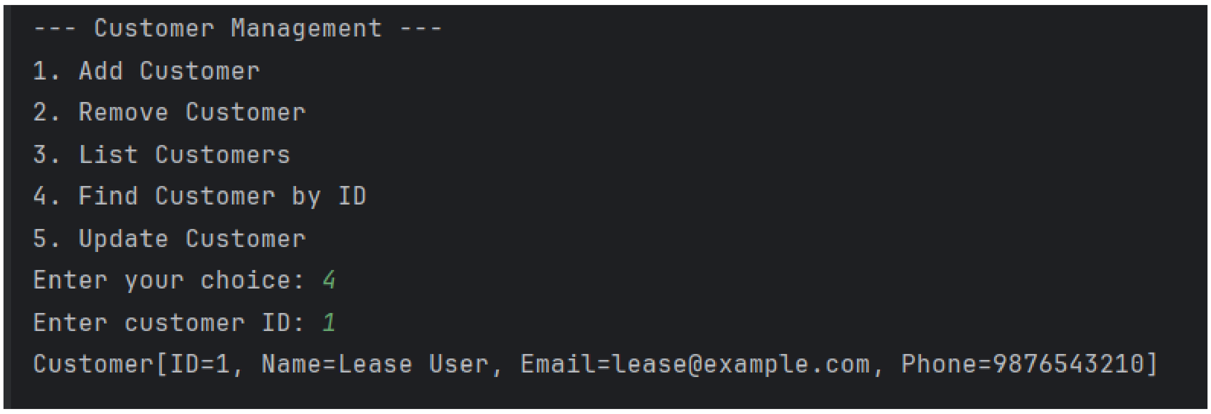
```
--- Customer Management ---
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Update Customer
Enter your choice: 1
Enter first name: Ritu
Enter last name: Sharma
Enter email: ritu35@gmail.com
Enter phone number: 9809809890
Customer added successfully.
```



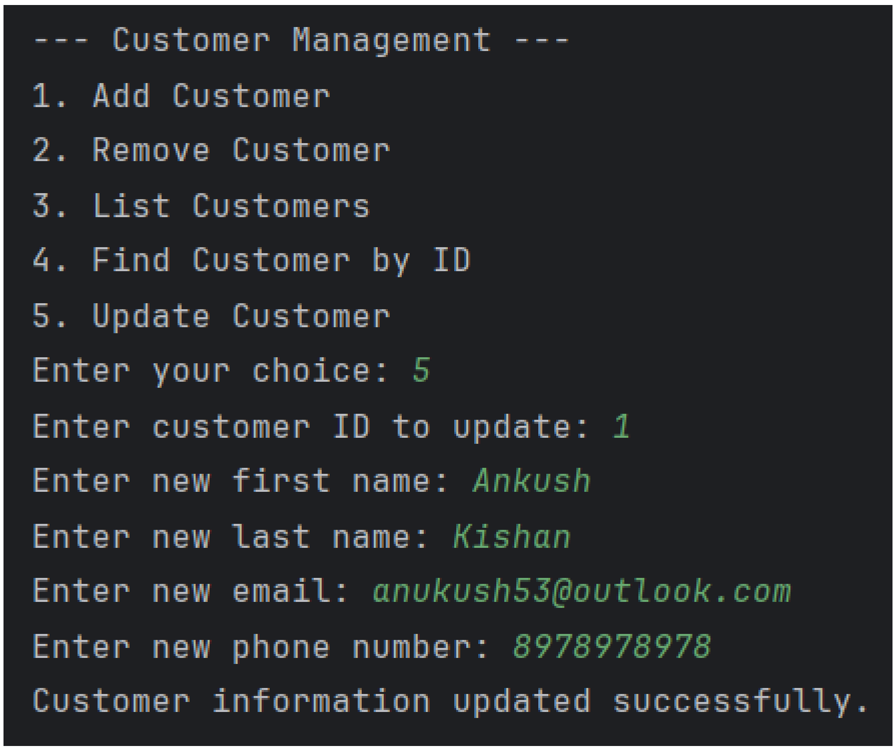
```
--- Customer Management ---
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Update Customer
Enter your choice: 2
Enter customer ID to remove: 2
Customer removed successfully.
```



```
--- Customer Management ---
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Update Customer
Enter your choice: 3
Customer[ID=1, Name=Lease User, Email=lease@example.com, Phone=9876543210]
Customer[ID=6, Name=Anita Roy, Email=anita.roy@example.com, Phone=9123456789]
Customer[ID=8, Name=Rahul Sharma, Email=rahul.sharma@example.com, Phone=9876543211]
Customer[ID=9, Name=Ritu Sharma, Email=ritu35@gmail.com, Phone=9809809890]
```

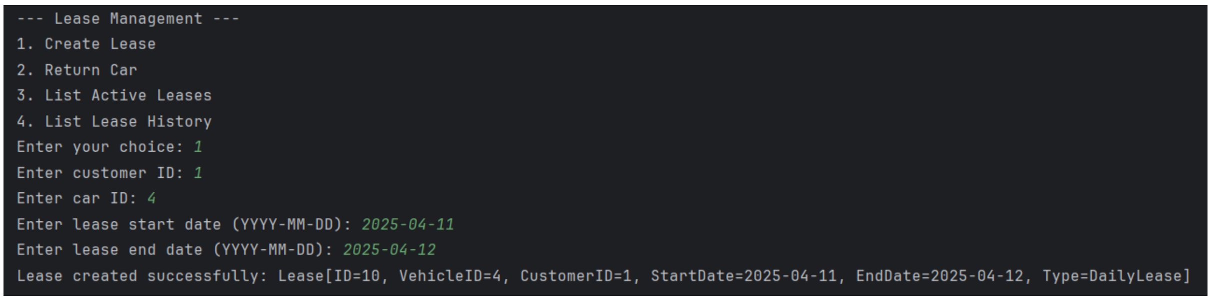


```
--- Customer Management ---
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Update Customer
Enter your choice: 4
Enter customer ID: 1
Customer[ID=1, Name=Lease User, Email=lease@example.com, Phone=9876543210]
```

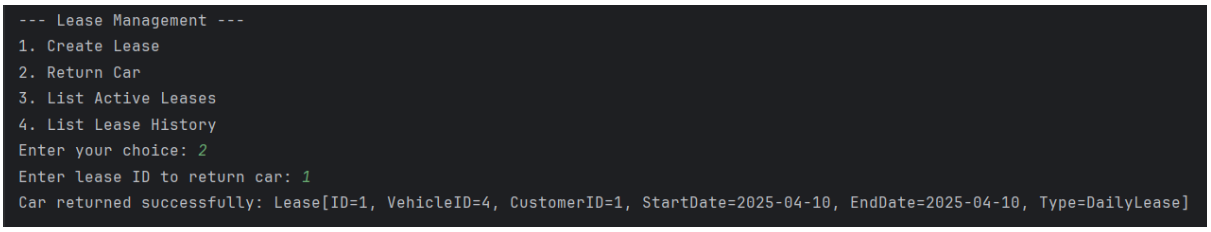


```
--- Customer Management ---
1. Add Customer
2. Remove Customer
3. List Customers
4. Find Customer by ID
5. Update Customer
Enter your choice: 5
Enter customer ID to update: 1
Enter new first name: Ankush
Enter new last name: Kishan
Enter new email: anukush53@outlook.com
Enter new phone number: 8978978978
Customer information updated successfully.
```

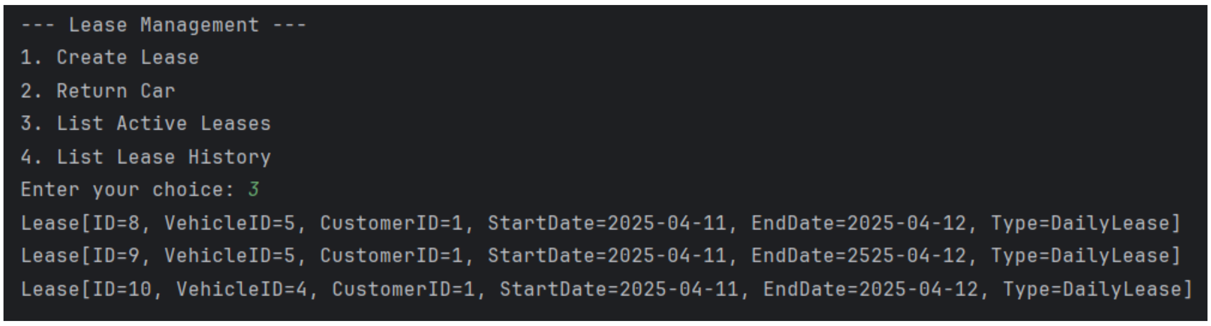
3.Lease Management:



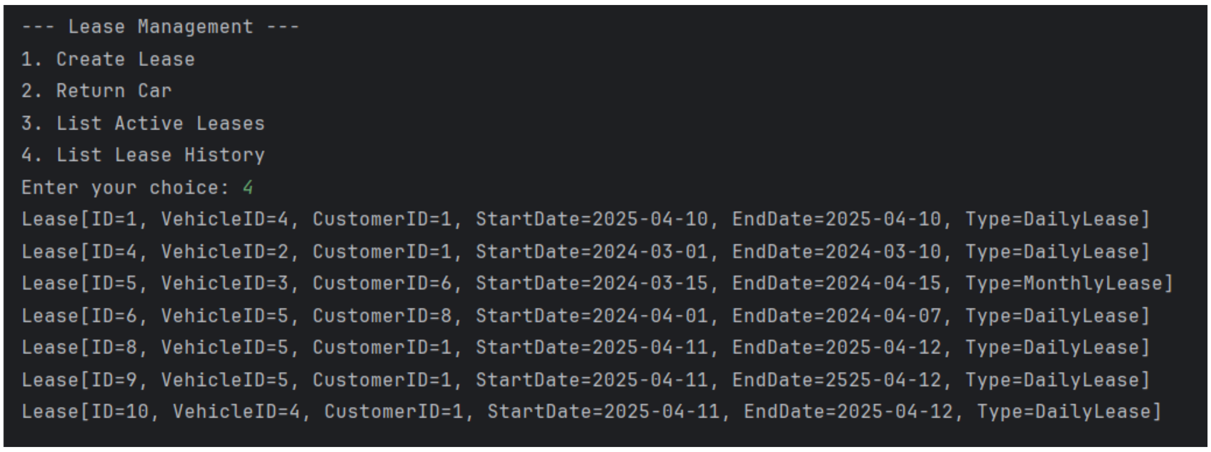
```
--- Lease Management ---
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
Enter your choice: 1
Enter customer ID: 1
Enter car ID: 4
Enter lease start date (YYYY-MM-DD): 2025-04-11
Enter lease end date (YYYY-MM-DD): 2025-04-12
Lease created successfully: Lease[ID=10, VehicleID=4, CustomerID=1, StartDate=2025-04-11, EndDate=2025-04-12, Type=DailyLease]
```



```
--- Lease Management ---
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
Enter your choice: 2
Enter lease ID to return car: 1
Car returned successfully: Lease[ID=1, VehicleID=4, CustomerID=1, StartDate=2025-04-10, EndDate=2025-04-10, Type=DailyLease]
```

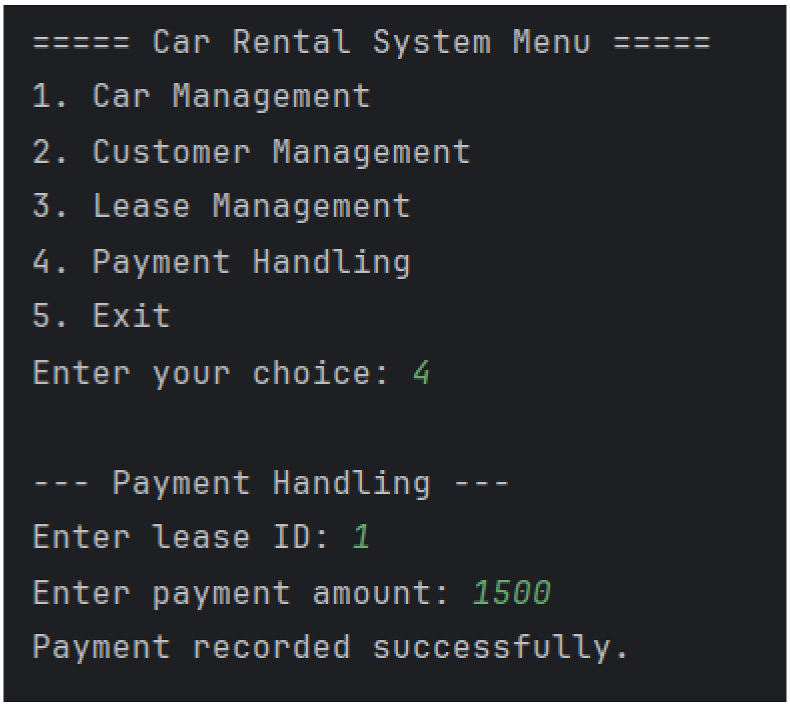


```
--- Lease Management ---
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
Enter your choice: 3
Lease[ID=8, VehicleID=5, CustomerID=1, StartDate=2025-04-11, EndDate=2025-04-12, Type=DailyLease]
Lease[ID=9, VehicleID=5, CustomerID=1, StartDate=2025-04-11, EndDate=2525-04-12, Type=DailyLease]
Lease[ID=10, VehicleID=4, CustomerID=1, StartDate=2025-04-11, EndDate=2025-04-12, Type=DailyLease]
```



```
--- Lease Management ---
1. Create Lease
2. Return Car
3. List Active Leases
4. List Lease History
Enter your choice: 4
Lease[ID=1, VehicleID=4, CustomerID=1, StartDate=2025-04-10, EndDate=2025-04-10, Type=DailyLease]
Lease[ID=4, VehicleID=2, CustomerID=1, StartDate=2024-03-01, EndDate=2024-03-10, Type=DailyLease]
Lease[ID=5, VehicleID=3, CustomerID=6, StartDate=2024-03-15, EndDate=2024-04-15, Type=MonthlyLease]
Lease[ID=6, VehicleID=5, CustomerID=8, StartDate=2024-04-01, EndDate=2024-04-07, Type=DailyLease]
Lease[ID=8, VehicleID=5, CustomerID=1, StartDate=2025-04-11, EndDate=2025-04-12, Type=DailyLease]
Lease[ID=9, VehicleID=5, CustomerID=1, StartDate=2025-04-11, EndDate=2525-04-12, Type=DailyLease]
Lease[ID=10, VehicleID=4, CustomerID=1, StartDate=2025-04-11, EndDate=2025-04-12, Type=DailyLease]
```

4.PAYMENT HANDLING:



```
===== Car Rental System Menu =====
1. Car Management
2. Customer Management
3. Lease Management
4. Payment Handling
5. Exit
Enter your choice: 4

--- Payment Handling ---
Enter lease ID: 1
Enter payment amount: 1500
Payment recorded successfully.
```

CONCLUSION:

The Car Rental System project successfully demonstrates how to design and develop a full-stack Python application integrated with a MySQL database. It includes core modules for vehicle, customer, lease, and payment management, allowing users to perform real-time operations such as renting a car, returning it, managing customers, and handling payments. With proper schema design, data access layers (DAO), custom exceptions, and modular structure, the project ensures scalability, maintainability, and a clean separation of concerns. It provides a practical and extensible foundation for understanding real-world software architecture and database-driven application development.