CIS 552: DATABASE DESIGN

Final Project Summary

Car Rental Management System

Group-12:

Vishnu Vardhan Nagunuri-(53)

Karthik Reddy Suram-(78)

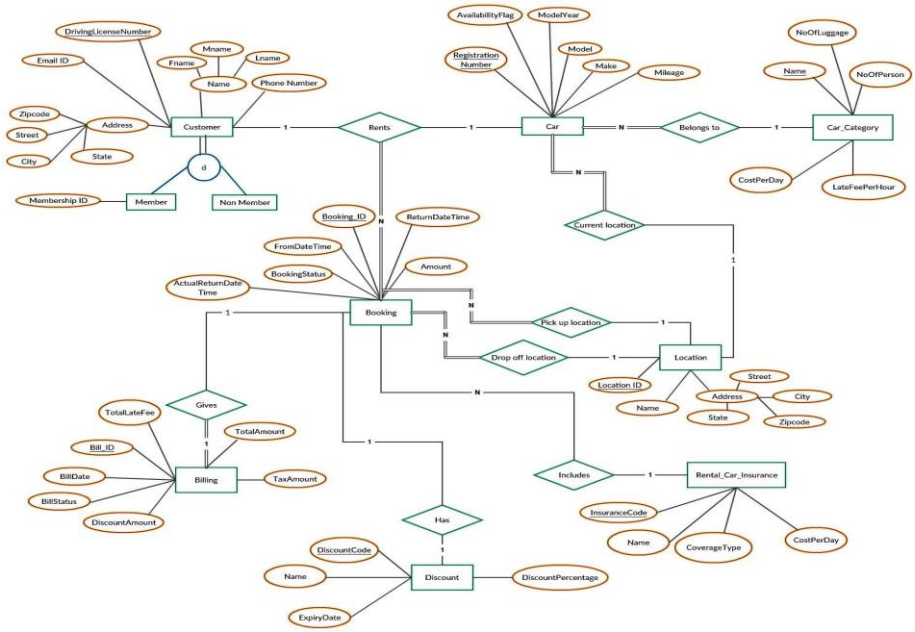
Part 1**: Describe how your Database-driven Application, Describe clearly and succinctly how the planned use-case that the database-driven application will implement extended/revised with the data store paradigms.**

This project presents the design and implementation of a comprehensive car rental management system leveraging MySQL for database management and Python, along with the Streamlit framework, for application development. The system aims to streamline the process of car rental operations, encompassing functionalities such as vehicle inventory management, customer registration, reservation handling, and billing management.

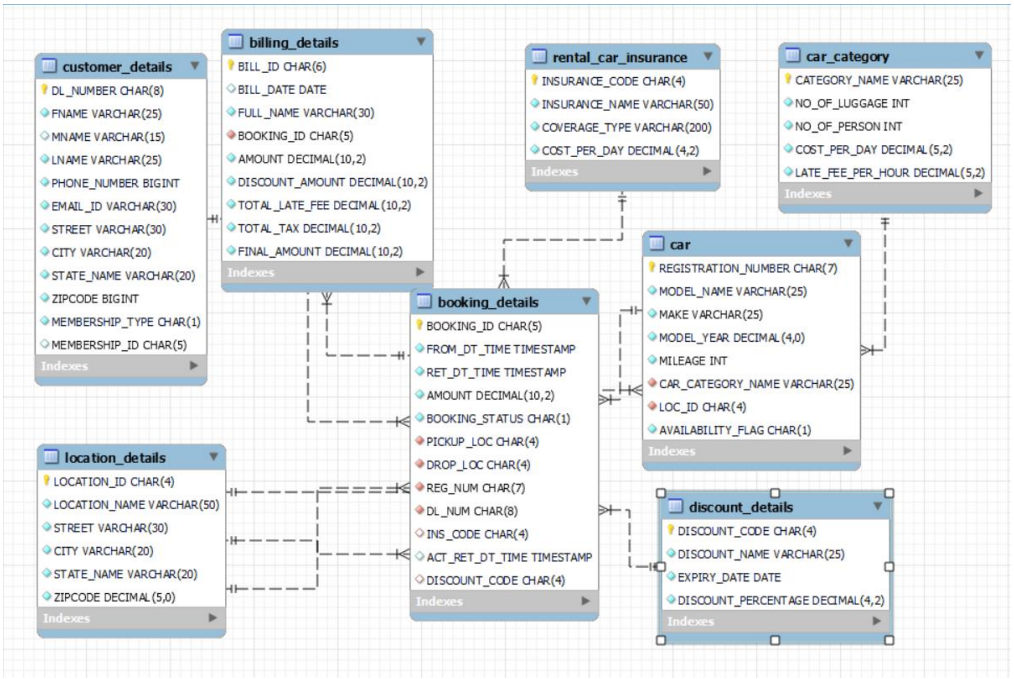
Utilizing MySQL as the backend database allows for efficient storage and retrieval of critical data pertaining to vehicles, customers, reservations. Python serves as the primary programming language for backend logic, ensuring robustness and scalability. Streamlit, a user-friendly web application framework, facilitates the creation of an intuitive and interactive user interface for seamless navigation and user engagement. Our application has key features mentioned below:

- Car Selection Based on Make and Model: Customers can browse through available cars and choose based on their preferred make and model.
- Flexible Pick-up and Drop-off Locations: Our system allows customers to specify different pick-up and drop-off locations, offering convenience and adaptability.
- Late Return Fee Calculation: Automatic calculation and imposition of late fees for rental cars returned beyond the scheduled return date and time.
- Discount Coupon Application: Customers can apply discount coupons towards their final bill, with the system supporting the use of up to one discount coupon per transaction.

ER diagram:



ER Relationship Schema:



Implementation:

The implementation of the car rental management system involves integrating MySQL, Python, and Streamlit to create a comprehensive solution. Firstly, the MySQL database schema is designed and implemented to store essential data such as vehicle details and customer information. Python scripts are then developed to establish a connection to the MySQL database, facilitating data retrieval, insertion, updating, and deletion operations. These scripts utilize libraries like my sql connector for database interaction, ensuring seamless data handling within the system.

Secondly, the Streamlit framework is utilized to build a user-friendly web interface for the car rental management system. Streamlit's intuitive design enables the creation of interactive elements such as forms, buttons, and tables, facilitating smooth user interactions. Through Streamlit, functionalities such as vehicle registration, rental booking, and administrative tasks are implemented, providing users with a seamless experience in managing their car rental operations.

Lastly, the integrated system undergoes rigorous testing to validate its functionality, performance, and reliability. Test scenarios are designed to cover various use cases, ensuring that the system

behaves as expected under different conditions. Any issues or bugs identified during testing are addressed through iterative development cycles, refining the system for optimal performance. Once testing is complete, the car rental management system is deployed, making it accessible to users for efficient management of their rental operations.

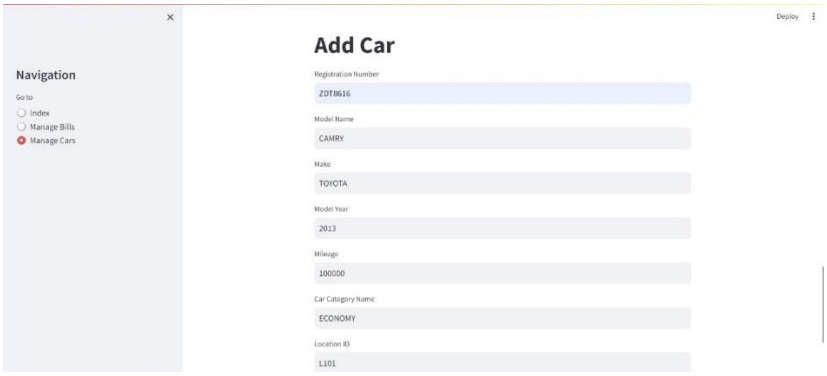
Part 2: Working Incorporation of Data Store (Database Engine)
Provide evidence including screenshots of the working database-driven application including data store software, demonstrating that the application works by (data creation, loading, updating, and querying)

We have performed CRUD operations on CAR table.So,using this application we can add a car,delete a car,update a car and display car details.
To add a car:

The user interface titled “Add Car” presents a clean design for inputting car details. It includes fields for registration number, model name, make, model year, mileage, car category, and economy ID. The light blue and gray color scheme enhances readability.

Sample data from the image:

- **Registration Number**: 2075JHG
- **Model Name**: CAMRY
- **Make**: TOYOTA
- **Model Year**: 2013
- **Mileage**: 100,000
- **Car Category Name**: ECONOMY
- **Economy ID**: L001

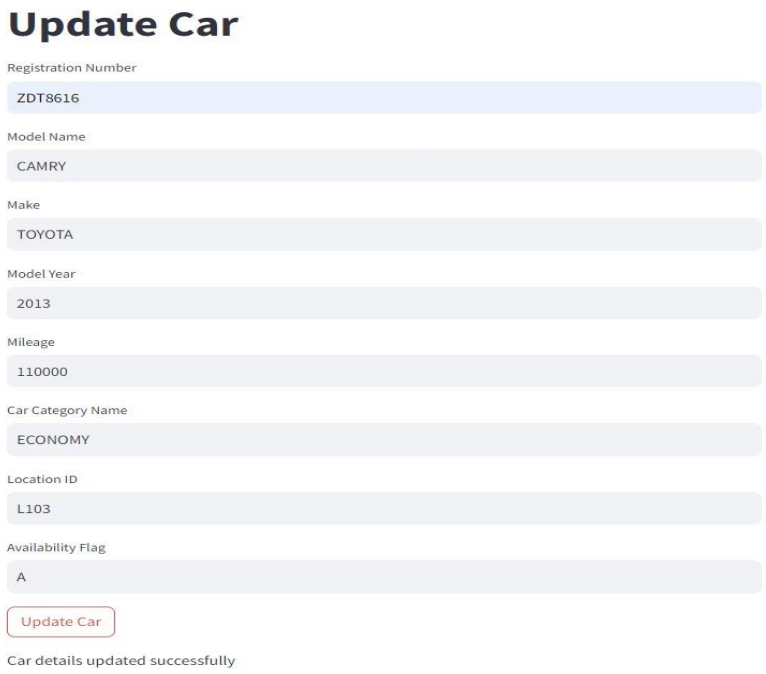



```
336 •  Select * From Car;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

REGISTRATION_NUMBER	MODEL_NAME	MAKE	MODEL_YEAR	MILEAGE	CAR_CATEGORY_NAME	LOC_ID	AVAILABILITY_FLAG
VBN6283	TAURUS	FORD	2015	17500	STANDARD	L101	A
WDV2458	FALCON	FORD	2016	5600	FULL SIZE	L107	A
WER3245	ACCENT	HYUNDAI	2014	12356	ECONOMY	L103	A
WHM7619	AVALON	TOYOTA	2016	7800	LUXURY CAR	L105	A
WIJ6190	EDGE	FORD	2014	18700	STANDARD SUV	L106	A
WKJ7972	SEQUOIA	TOYOTA	2013	14500	FULL SIZE SUV	L103	A
WLZ8955	ESCAPE	FORD	2012	19800	MID SIZE SUV	L106	A
XBM6822	SUBURBAN	CHEVROLET	2016	3400	FULL SIZE SUV	L106	A
YSN1927	PATHFINDER	NISSAN	2014	14390	MID SIZE SUV	L101	A
ZDT8612	TAHOE XL	CHEVROLET	2018	14300	STANDARD SUV	L107	A
ZDT8614	TAHOE	CHEVROLET	2015	14300	STANDARD SUV	L107	A
ZDT8616	CAMRY	TOYOTA	2013	100000	ECONOMY	L101	A
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

As seen in the picture the car added to the database successfully.



The image depicts a **interface** for updating car details. Here are the key points:

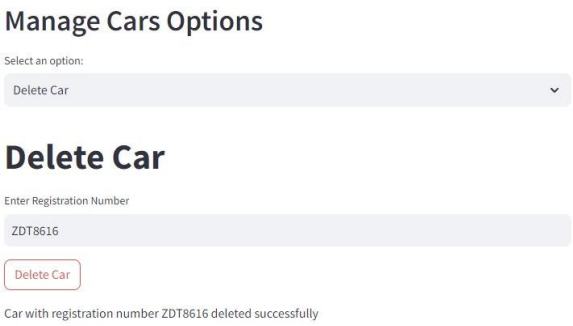
1. **Form Fields**:
   - **Registration Number**: ZDT6816
   - **Model Name**: CAMRY
   - **Make**: TOYOTA
   - **Model Year**: 2013
   - **Mileage**: 110,000
   - **Car Category Name**: ECONOMY
   - **Location ID**: L03
   - **Availability Flag**: A
2. The form includes an **“Update Car”** button, and below it, a confirmation message states **“Car details updated successfully.”**

```
336 •  Select * From Car;
337    On Car.Loc id=location details.LOCATION ID
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

REGISTRATION_NUMBER	MODEL_NAME	MAKE	MODEL_YEAR	MILEAGE	CAR_CATEGORY_NAME	LOC_ID	AVAILABILITY_FLAG
VBN6283	TAURUS	FORD	2015	17500	STANDARD	L101	A
WDV2458	FALCON	FORD	2016	5600	FULL SIZE	L107	A
WER3245	ACCENT	HYUNDAI	2014	12356	ECONOMY	L103	A
WHM7619	AVALON	TOYOTA	2016	7800	LUXURY CAR	L105	A
WIJ6190	EDGE	FORD	2014	18700	STANDARD SUV	L106	A
WKJ7972	SEQUOIA	TOYOTA	2013	14500	FULL SIZE SUV	L103	A
WLZ8955	ESCAPE	FORD	2012	19800	MID SIZE SUV	L106	A
XBM6822	SUBURBAN	CHEVROLET	2016	3400	FULL SIZE SUV	L106	A
YSN1927	PATHFINDER	NISSAN	2014	14390	MID SIZE SUV	L101	A
ZDT8612	TAHOE XL	CHEVROLET	2018	14300	STANDARD SUV	L107	A
ZDT8614	TAHOE	CHEVROLET	2015	14300	STANDARD SUV	L107	A
ZDT8616	CAMRY	TOYOTA	2013	110000	ECONOMY	L103	A
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

As seen in the picture the car details are updated successfully



The image depicts a **user interface** for managing cars within a system. Here are the key details:

1. The interface is titled **“Manage Cars Options”**.
2. **Delete Car Section**:
    - An input field labeled **“Enter Registration Number”** allows users to specify the car they want to delete.
    - Below the input field, there’s a **“Delete Car”** button.
    - A confirmation message at the bottom states: **“Car with registration number ZDT68516 deleted successfully.”**

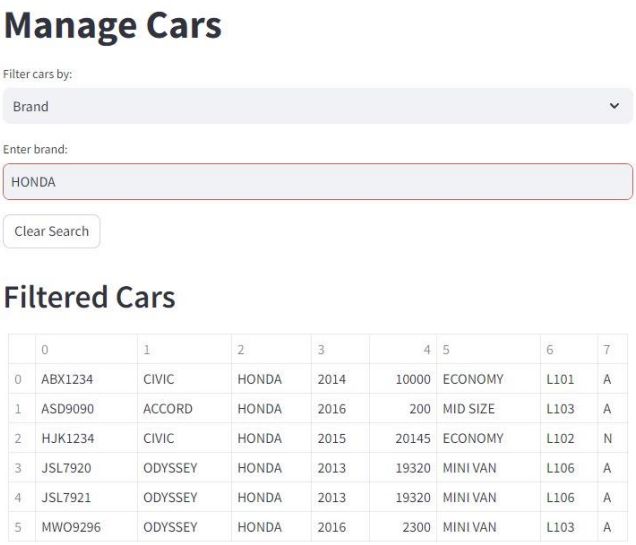
This interface streamlines car management tasks, enabling users to remove specific cars from the system.

```
335
336 •  Select * From Car;
337     On Car.Loc id=location details.LOCATION ID
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

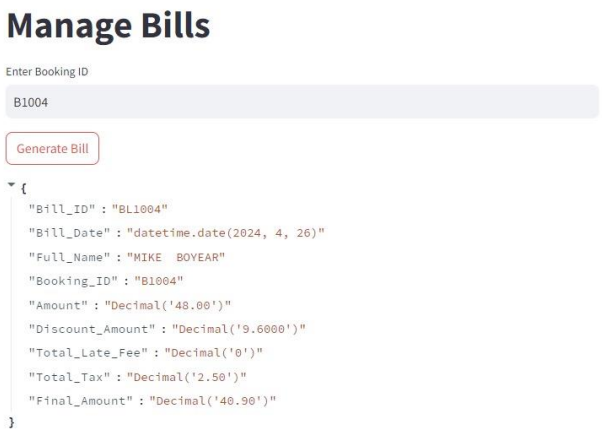
REGISTRATION_NUMBER	MODEL_NAME	MAKE	MODEL_YEAR	MILEAGE	CAR_CATEGORY_NAME	LOC_ID	AVAILABILITY_FLAG
UYT3981	LEGACY	SUBARU	2013	16750	MID SIZE	L104	A
VBN6283	TAURUS	FORD	2015	17500	STANDARD	L101	A
WDV2458	FALCON	FORD	2016	5600	FULL SIZE	L107	A
WER3245	ACCENT	HYUNDAI	2014	12356	ECONOMY	L103	A
WHM7619	AVALON	TOYOTA	2016	7800	LUXURY CAR	L105	A
WIJ6190	EDGE	FORD	2014	18700	STANDARD SUV	L106	A
WKJ7972	SEQUOIA	TOYOTA	2013	14500	FULL SIZE SUV	L103	A
WLZ8955	ESCAPE	FORD	2012	19800	MID SIZE SUV	L106	A
XBM6822	SUBURBAN	CHEVROLET	2016	3400	FULL SIZE SUV	L106	A
YSN1927	PATHFINDER	NISSAN	2014	14390	MID SIZE SUV	L101	A
ZDT8612	TAHOE XL	CHEVROLET	2018	14300	STANDARD SUV	L107	A
ZDT8614	TAHOE	CHEVROLET	2015	14300	STANDARD SUV	L107	A
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

As seen in this picture the car with registration number ZDT8616 deleted successfully.



The image depicts a **user interface** for managing cars within a system. Here are the key details:

1. The interface is titled **“Manage Cars Options”**.
2. **Delete Car Section**:
    - An input field labeled **“Enter Registration Number”** allows users to specify the car they want to delete.
    - Below the input field, there’s a **“Delete Car”** button.
    - A confirmation message at the bottom states: **“Car with registration number ZDT68516 deleted successfully.”**



The image depicts a **user interface** for managing bills within a system. Here are the key details:

1. The interface is titled **“Manage Bills”**.
2. **Input Section**:
    - An input field labeled **“Enter Booking ID”** allows users to specify the booking for which they want to generate a bill.
    - Below the input field, there’s a **“Generate Bill”** button.
3. **Output Section** (in JSON format):
    - The generated bill includes details such as:
        - **Bill ID**: B1004
        - **Bill Date**: April 26, 2024
        - **Full Name**: MIKE BOVEARI
        - **Booking ID**: B1004
        - Other relevant information related to the bil

Manage Bills

Enter Booking ID

B1004

Generate Bill

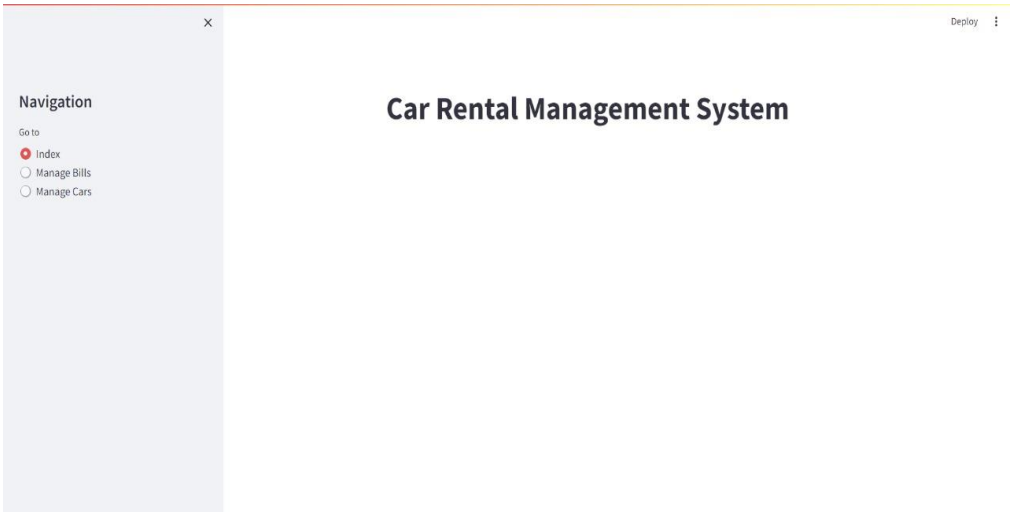
	0	1	2	3	4	5	6	7	8
0	Bill ID	Bill Date	full_name	Booking ID	Amount	Discount Amount	Late Fee	Total Tax	Final Amount
1	BL1001	2024-04-20	NAVEEN RAJ	B1001	150.00	15.00	0.00	8.78	143.78
2	BL1002	2024-04-20	DANISH HASSAN	B1008	630.00	63.00	43.20	39.66	649.86
3	BL1003	2024-04-20	MARK HUFF	B1003	450.00	0.00	0.00	29.25	479.25
4	BL1004	2024-04-26	MIKE BOYEAR	B1004	48.00	9.60	0.00	2.50	40.90

The image depicts a **user interface** for managing bills within a system. Here are the key details:

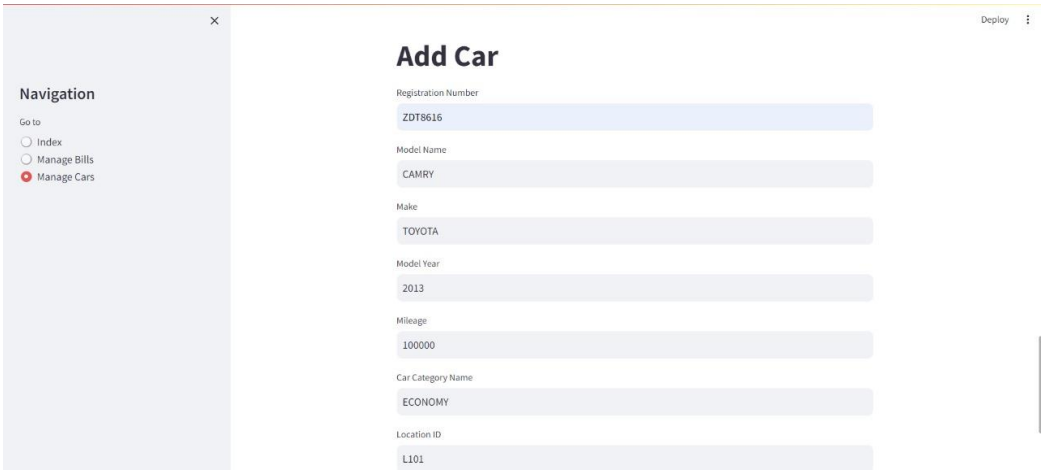
1. The interface is titled **“Manage Bills Options”**.
2. By entering the booking Id in given space and by clicking on generate bill. The billing details are displayed.

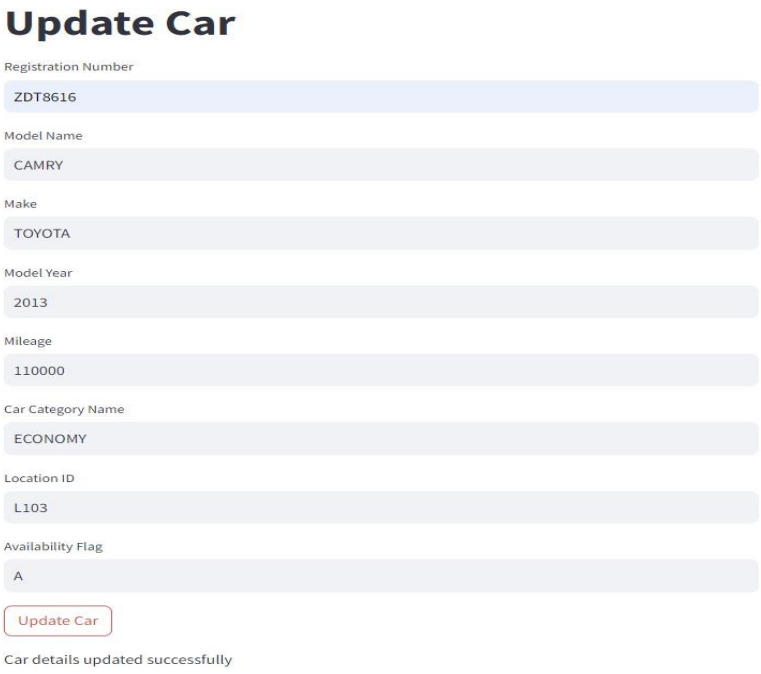
Part 3: Your front-hand Interface: Provide evidence including screenshots of the front-end interface

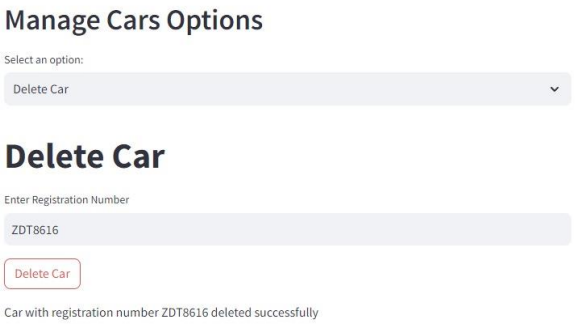
This is the interface of the car rental management system. -



This is the interface used to manage cars:

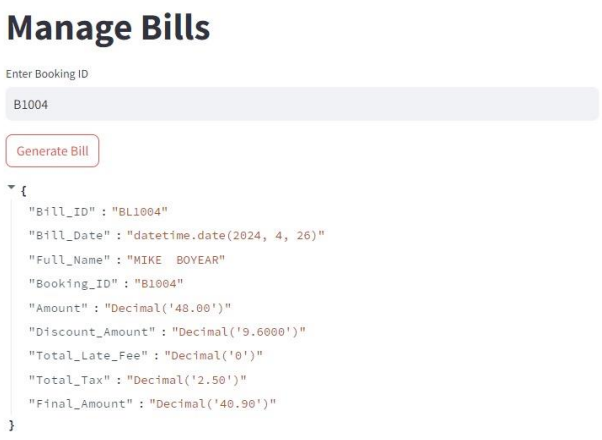


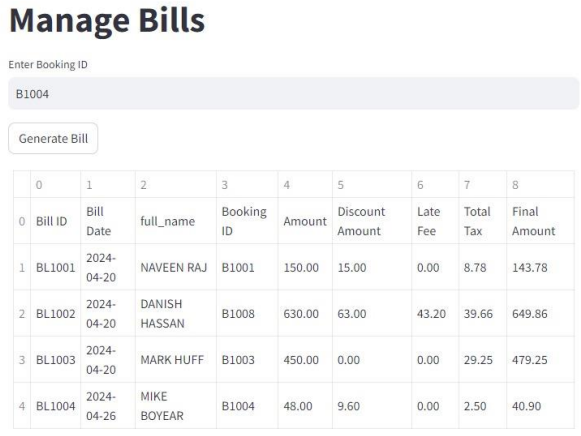




This interface streamlines car management tasks, enabling users to remove specific cars from the system.

Below is the interface to manage billing:





	0	1	2	3	4	5	6	7	8
0	Bill ID	Bill Date	full_name	Booking ID	Amount	Discount Amount	Late Fee	Total Tax	Final Amount
1	BL1001	2024-04-20	NAVEEN RAJ	B1001	150.00	15.00	0.00	8.78	143.78
2	BL1002	2024-04-20	DANISH HASSAN	B1008	630.00	63.00	43.20	39.66	649.86
3	BL1003	2024-04-20	MARK HUFF	B1003	450.00	0.00	0.00	29.25	479.25
4	BL1004	2024-04-26	MIKE BOYEAR	B1004	48.00	9.60	0.00	2.50	40.90

Part 4: Your source code: Provide evidence including screenshots of the code that manages your interface and communicates with your database. (code of interface, database querying and all related CURD operations.)

CODE:

Below is the code to perform CRUD Operations written in python.Mysql database has been connected using mysql-connector-python library of python.Below is the code to perform that:

```
import mysql.connector as connector
class CarRental:
    def __init__(self):
        self.con = connector.connect(
            host="localhost",
            port="3306",
            user="root",
            password="Vishnu123!",
            database="car_rental"
        )
        # create table
         query = 'create table if not exists CAR(REGISTRATION_NUMBER CHAR(7) NOT NULL,MODEL_NAME VARCHAR(25) NOT NULL,MAKE VARCHAR(25) NOT NULL,MODEL_YEAR DECIMAL(4) NOT NULL,MILEAGE INTEGER NOT NULL,CAR_CATEGORY_NAME VARCHAR(25) NOT NULL,LOC_ID CHAR(4) NOT NULL,AVAILABILITY_FLAG CHAR(1) NOT NULL,CONSTRAINT CARPK PRIMARY KEY (REGISTRATION_NUMBER))'
        cur = self.con.cursor()
        cur.execute(query)
```

Using above code we connected mysql database with python.Here we are using car_rental Database that is already created in the mysql.In the above code we are creating the CAR table if it does not exist.

Using below code we performed the Create,Read,Update,Delete(CRUD) operations using python.

```
    def insert_car(self, REGISTRATION_NUMBER, MODEL_NAME, MAKE, MODEL_YEAR, MILEAGE, CAR_CATEGORY_NAME, LOC_ID, AVAILABILITY_FLAG):
        try:
             # Check if the car with the given registration number already exists
            cur = self.con.cursor()
             cur.execute("SELECT 1 FROM CAR WHERE REGISTRATION_NUMBER = %s", (REGISTRATION_NUMBER,))
            if cur.fetchone():
               return ("Car with the same registration number already exists.")
```

```
            # Insert the car data into the database
             query = "INSERT INTO CAR(REGISTRATION_NUMBER, MODEL_NAME, MAKE,
MODEL_YEAR, MILEAGE, CAR_CATEGORY_NAME, LOC_ID, AVAILABILITY_FLAG) VALUES (%s,
%s, %s, %s, %s, %s, %s, %s)"
               values = (REGISTRATION_NUMBER, MODEL_NAME, MAKE, MODEL_YEAR,
MILEAGE, CAR_CATEGORY_NAME, LOC_ID, AVAILABILITY_FLAG)
            cur.execute(query, values)
            self.con.commit()
            return "Car added to database successfully."
        except connector.Error as e:
               return "An error occurred while inserting the car into the
database:", e


    # read data
    def read_car(self):
        try:
            query = "SELECT * FROM CAR"
            cur = self.con.cursor()
            cur.execute(query)
            cars = cur.fetchall()  # Fetch all rows from the result set
            # for row in cars:
            #     print(row[0])
            return cars
        except Exception as e:
            print("An error occurred while reading car data:", e)
            return None  # Return None in case of error

    # delete car
    def delete_car(self, REGISTRATION_NUMBER):
        try:
            query = "DELETE FROM CAR WHERE REGISTRATION_NUMBER = %s"
            cur = self.con.cursor()
            cur.execute(query, (REGISTRATION_NUMBER,))
            self.con.commit()
            if cur.rowcount > 0:
                        return "Car with registration number {} deleted
successfully".format(REGISTRATION_NUMBER)
            else:
                         return "No car found with registration number
{}".format(REGISTRATION_NUMBER)
        except connector.Error as e:
            print("An error occurred:", e)
```

```
    # update car details
      def update_car(self, REGISTRATION_NUMBER, MODEL_NAME=None, MAKE=None,
MODEL_YEAR=None,    MILEAGE=None,    CAR_CATEGORY_NAME=None,    LOC_ID=None,
AVAILABILITY_FLAG=None):
        try:
            update_fields = []
            values = []

            if MODEL_NAME is not None:
                update_fields.append("MODEL_NAME = %s")
                values.append(MODEL_NAME)
            if MAKE is not None:
                update_fields.append("MAKE = %s")
                values.append(MAKE)
            if MODEL_YEAR is not None:
                update_fields.append("MODEL_YEAR = %s")
                values.append(MODEL_YEAR)
            if MILEAGE is not None:
                update_fields.append("MILEAGE = %s")
                values.append(MILEAGE)
            if CAR_CATEGORY_NAME is not None:
                update_fields.append("CAR_CATEGORY_NAME = %s")
                values.append(CAR_CATEGORY_NAME)
            if LOC_ID is not None:
                update_fields.append("LOC_ID = %s")
                values.append(LOC_ID)
            if AVAILABILITY_FLAG is not None:
                update_fields.append("AVAILABILITY_FLAG = %s")
                values.append(AVAILABILITY_FLAG)

            if not update_fields:
                return "No fields provided for update"

           query = "UPDATE CAR SET {} WHERE REGISTRATION_NUMBER = %s".format(",
".join(update_fields))
            values.append(REGISTRATION_NUMBER)

            cur = self.con.cursor()
            cur.execute(query, tuple(values))
            self.con.commit()

            if cur.rowcount > 0:
```

```
                return "Car details updated successfully"
            else:
                        return  "No  car  found  with  registration  number
{}".format(REGISTRATION_NUMBER)
        except connector.Error as e:
            print("An error occurred:", e)
```

Below is the code to generate a bill based on the booking details and it will calculate the late fee,Discount amount,Tax and total amount of the booking id entered.

```
import mysql.connector as connector
import decimal;
from datetime import datetime

class Billing:
    def __init__(self):
        self.con = connector.connect(
            host="localhost",
            port="3306",
            user="root",
            password="Vishnu123!",
            database="car_rental"
        )
        # create table
    def get_all_bills(self):
        try:
            query="SELECT * FROM BILLING_DETAILS"
            cur=self.con.cursor()
            cur.execute(query)
            bills=cur.fetchall()
            return bills
        except Exception as e:
            print("An error occurred while reading bills:", e)
            return None  # Return None in case of error
    def create_final_bill(self, booking_id):
        try:
            # Check if the database connection is available
            if not self.con:
                print("Error: MySQL Connection not available.")
                return
             # Check if a bill with the same booking ID already exists
            cur = self.con.cursor()
```

```
            cur.execute("SELECT COUNT(*) FROM billing_details WHERE booking_id
= %s", (booking_id,))
            existing_bills_count = cur.fetchone()[0]
            if existing_bills_count > 0:
                print("Bill already exists for the booking ID:", booking_id)
                return
            # Generate bill ID
            def generate_bill_ID():
                try:
                     # Query to get the last bill number from billing_details
table
                       query = "SELECT bill_ID FROM billing_details ORDER BY
bill_ID DESC LIMIT 1"
                    cur = self.con.cursor()
                    cur.execute(query)
                    last_bill_ID = cur.fetchone()

                    if last_bill_ID:
                        # Extract numeric part of the last bill number
                        last_bill_ID_numeric = int(last_bill_ID[0][2:])
                        # Increment the numeric part
                        new_bill_ID_numeric = last_bill_ID_numeric + 1
                         # Generate the new bill number by concatenating "BL"
with the incremented numeric part
                        new_bill_ID = f"BL{new_bill_ID_numeric:04d}"  # Format
to ensure 4 digits after "BL"
                    else:
                        # If no previous bill numbers exist, start with BL1001
                        new_bill_ID = "BL1001"

                    return new_bill_ID

                except Exception as e:
                    print("Error generating bill number:", e)
                    return None
            bill_id=generate_bill_ID()
            # Get current date for bill_date
            bill_date = datetime.now().date()

            # Fetch necessary values
            def fetch_booking_data():
                try:
```

```
                    query = "SELECT amount, RET_DT_TIME, ACT_RET_DT_TIME,
discount_code FROM booking_details WHERE Booking_id = %s"
                    cur = self.con.cursor()
                    cur.execute(query, (booking_id,))
                    return cur.fetchone()
                except Exception as e:
                    print("Error fetching booking data:", e)
                    return None

            booking_data = fetch_booking_data()

            if not booking_data:
                print("No booking data found.")
                return

            initial_amount = decimal.Decimal(booking_data[0])
            Return_Date = booking_data[1]
            Act_Return_Date = booking_data[2]
            discount_code = booking_data[3]

            # Calculate discount amount
            def calculate_discount_amount():
                try:
                    if not discount_code:
                        return decimal.Decimal('0.00')

                    query = "SELECT Discount_percentage FROM discount_details
WHERE discount_code = %s"
                    cur = self.con.cursor()
                    cur.execute(query, (discount_code,))
                    discount_percentage = cur.fetchone()


                                                    return    initial_amount    *
(decimal.Decimal(discount_percentage[0]) / 100)
                except Exception as e:
                    print("Error calculating discount amount:", e)
                    return None

            discount_amount = calculate_discount_amount()

            # Calculate late fee
            def calculate_late_fee():
                try:
```

```
                    # Fetch Return Date, Actual Return Date, and Late fee per
hour
                        query = "SELECT BD.RET_DT_TIME, BD.ACT_RET_DT_TIME,
CC.LATE_FEE_PER_HOUR FROM BOOKING_DETAILS AS BD JOIN CAR AS C ON BD.REG_NUM =
C.REGISTRATION_NUMBER JOIN CAR_CATEGORY AS CC ON C.CAR_CATEGORY_NAME =
CC.CATEGORY_NAME WHERE BD.Booking_id = %s"
                    cur = self.con.cursor()
                    cur.execute(query, (booking_id,))
                    result = cur.fetchone()
                    if result:
                        Return_Date = result[0]
                        Act_Return_Date = result[1]
                        LatefeePerHR = float(result[2])
                    else:
                        print("No Details Found")
                        return None

                    # Check if Return_Date and Act_Return_Date are not None
                    if Return_Date and Act_Return_Date:
                   # Calculate late fee if actual return is later than expected
return
                        if Act_Return_Date > Return_Date:
                                    hour_difference = (Act_Return_Date -
Return_Date).total_seconds() / 3600
                            late_fee = hour_difference * LatefeePerHR
                            return late_fee
                        else:
                            late_fee = decimal.Decimal(0.0)
                            return late_fee
                    else:
                          print("Error: Return date or actual return date is
None")
                        return None
                except Exception as e:
                    print("Error calculating late fee:", e)
                    return None
            # Calculate amount before tax
            late_fee=calculate_late_fee()
                    amount_before_tax = decimal.Decimal(initial_amount) +
decimal.Decimal(late_fee) - decimal.Decimal(discount_amount)
            amount_before_tax=round(amount_before_tax,2)

            # Calculate total tax
```

```
            def calculate_total_tax():
                try:
                    tax_rate = decimal.Decimal('0.065')  # Tax rate of 6.5%
                    return amount_before_tax * tax_rate
                except Exception as e:
                    print("Error calculating total tax:", e)
                    return None

            total_tax = calculate_total_tax()
            total_tax=round(total_tax,2)


            # Calculate final amount
            final_amount = amount_before_tax + total_tax
            final_amount=round(final_amount,2)
            try:
                query="SELECT CD.FNAME,CD.MNAME,CD.LNAME FROM BOOKING_DETAILS
AS  BD  JOIN  CUSTOMER_DETAILS  AS  CD  ON  CD.DL_NUMBER=BD.DL_NUM  WHERE
BOOKING_ID=%s"
                cur=self.con.cursor()
                cur.execute(query,(booking_id,))
                result=cur.fetchone()
                if result:
                    f_name=result[0]
                    m_name=result[1]
                    l_name=result[2]
                else:
                    return "Name Not Found"
                   full_name = f"{f_name} {' ' + m_name if m_name else ''}
{l_name}"
            except Exception as e:
                    print("Error calculating late fee:", e)
                    return None


            # Insert into billing_details table
           query = "INSERT INTO billing_details (bill_ID, bill_date,full_name,
booking_id,amount, discount_amount, total_late_fee, total_tax, final_amount)
VALUES (%s, %s, %s,%s, %s, %s, %s, %s, %s)"
           values = (bill_id, bill_date,full_name, booking_id,initial_amount,
discount_amount, late_fee, total_tax, final_amount)
            cur = self.con.cursor()
            cur.execute(query, values)
            self.con.commit()
```

```
            bill_details = {
            'Bill_ID': bill_id,  # Example bill ID
            'Bill_Date': bill_date,  # Example bill date
            'Full_Name':full_name,
            'Booking_ID': booking_id,
            'Amount':initial_amount,
            'Discount_Amount': discount_amount,  # Example initial amount
            'Total_Late_Fee': late_fee,  # Example late fee
            'Total_Tax': total_tax,  # Example total tax
            'Final_Amount': final_amount  # Example final amount
            }

            return bill_details
        except Exception as e:
            print("Error generating bill:", e)
            return None

billing=Billing()
billing.get_all_bills()
```

Below is the code to connect to streamlit and display a web page to manage cars and billing.

```
import streamlit as st
from carrental import CarRental
from billing import Billing

car_rental = CarRental()
bill_generate = Billing()

def index():
    st.title("Car Rental Management System")

def generate_bill():
    st.title("Manage Bills")
    booking_id = st.text_input("Enter Booking ID")
    if st.button("Generate Bill"):
        output = bill_generate.create_final_bill(booking_id)
        if output:
            st.write(output)
        else:
            st.error(f"Bill already exists for Booking ID: {booking_id}")
    bills=bill_generate.get_all_bills()
    if bills:
```

```
        bill_table=[]
        for bill in bills:
                    bill_row   =   [bill[0],   bill[1],   bill[2],   bill[3],
bill[4],bill[5],bill[6],bill[7],bill[8]]
            bill_table.append(bill_row)

            bill_table.insert(0,['Bill  ID','Bill  Date','full_name','Booking
ID','Amount','Discount Amount','Late Fee','Total Tax','Final Amount'])
        st.table(bill_table)
    else:
        st.write("No Bills Available")


import streamlit as st

def manage_cars():
    st.title("Manage Cars")
    cars = car_rental.read_car()
    filtered_cars = None  # Initialize filtered_cars variable

    if cars:
        filter_option = st.selectbox(
            "Filter cars by:",
            ["Brand", "Type"]
        )

        if filter_option == "Brand":
            brand = st.text_input("Enter brand:")
               filtered_cars  =  [car  for  car  in  cars  if  car[2].lower()  ==
brand.lower()]
            # if filtered_cars:
            #     st.table(filtered_cars)
            # else:
            #     st.write("No cars found with the specified brand.")
        elif filter_option == "Type":
            car_type = st.text_input("Enter car type:")
               filtered_cars  =  [car  for  car  in  cars  if  car[5].lower()  ==
car_type.lower()]
            # if filtered_cars:
            #     st.table(filtered_cars)
            # else:
            #     st.write("No cars found with the specified type.")
    else:
        st.write("No cars available.")
```

```
    # Option to clear search and display all rows
    if st.button("Clear Search"):
        filtered_cars = None

    # Display filtered cars or all cars
    if filtered_cars:
        st.header("Filtered Cars")
        st.table(filtered_cars)
    elif cars:
        st.header("All Available Cars")
        st.table(cars)

    st.header("Manage Cars Options")

    option = st.selectbox(
        "Select an option:",
        ["Add Car", "Update Car", "Delete Car"]
    )

    if option == "Add Car":
        add_car()
    elif option == "Update Car":
        update_car()
    elif option == "Delete Car":
        delete_car()



def add_car():
    st.title("Add Car")
    registration_number = st.text_input("Registration Number")
    model_name = st.text_input("Model Name")
    make = st.text_input("Make")
    model_year = st.text_input("Model Year")
    mileage = st.text_input("Mileage")
    car_category_name = st.text_input("Car Category Name")
    loc_id = st.text_input("Location ID")
    availability_flag = st.text_input("Availability Flag")
    if st.button("Add Car"):
        output = car_rental.insert_car(registration_number, model_name, make,
model_year, mileage, car_category_name, loc_id, availability_flag)
        st.write(output)
```

```
def display_cars():
    st.title("Manage Cars")
    st.header("Available Cars")
    cars = car_rental.read_car()
    if cars:
        # Display the list of cars in a tabular format
        st.table(cars)
    else:
        st.write("No cars available.")


def delete_car():
    st.title("Delete Car")
    registration_number = st.text_input("Enter Registration Number")
    if st.button("Delete Car"):
        output=car_rental.delete_car(registration_number)
        st.write(output)

def update_car():
    st.title("Update Car")
    registration_number = st.text_input("Registration Number")
    model_name = st.text_input("Model Name")
    make = st.text_input("Make")
    model_year = st.text_input("Model Year")
    mileage = st.text_input("Mileage")
    car_category_name = st.text_input("Car Category Name")
    loc_id = st.text_input("Location ID")
    availability_flag = st.text_input("Availability Flag")
    if st.button("Update Car"):
         output=car_rental.update_car(registration_number, model_name, make,
model_year, mileage, car_category_name, loc_id, availability_flag)
        st.write(output)

def main():
    pages = {
        "Index": index,
        "Manage Bills": generate_bill,
        "Manage Cars": manage_cars,
    }
    st.sidebar.title("Navigation")
    selection = st.sidebar.radio("Go to", list(pages.keys()))
    pages[selection]()

if __name__ == "__main__":
```

```
main()
```

Conclusion:

To conclude, car rental management system is a full fledged application developed using mysql as database python for backend and streamlit as backend.We can perform all the CRUD operations on the CAR table i.e.,inserting,updating and deleting the cars.Also we can manage bills by adding new bill and deleting them using the application.We built the frontend using streamlit which is a powerful and user-friendly platform for building interactive web applications with Python. e've demonstrated how Streamlit can be utilized to create dynamic interfaces for managing cars and billing details within a rental system.

By integrating Streamlit with database operations and utilizing its intuitive components such as text inputs, buttons, and tables, we've showcased the seamless interaction between user input and backend functionality. From displaying data in tabular formats to enabling users to generate bills and perform CRUD operations on car records, Streamlit offers flexibility and ease of development. Moreover, by leveraging Streamlit's capabilities alongside standard Python libraries like Pandas and database connectors, developers can create comprehensive solutions that cater to various use cases efficiently.