**CIS 552: DATABASE DESIGN**

**Final Project Summary**

**Car Rental Management System**

Group-12:

Vishnu Vardhan Nagunuri-(53)

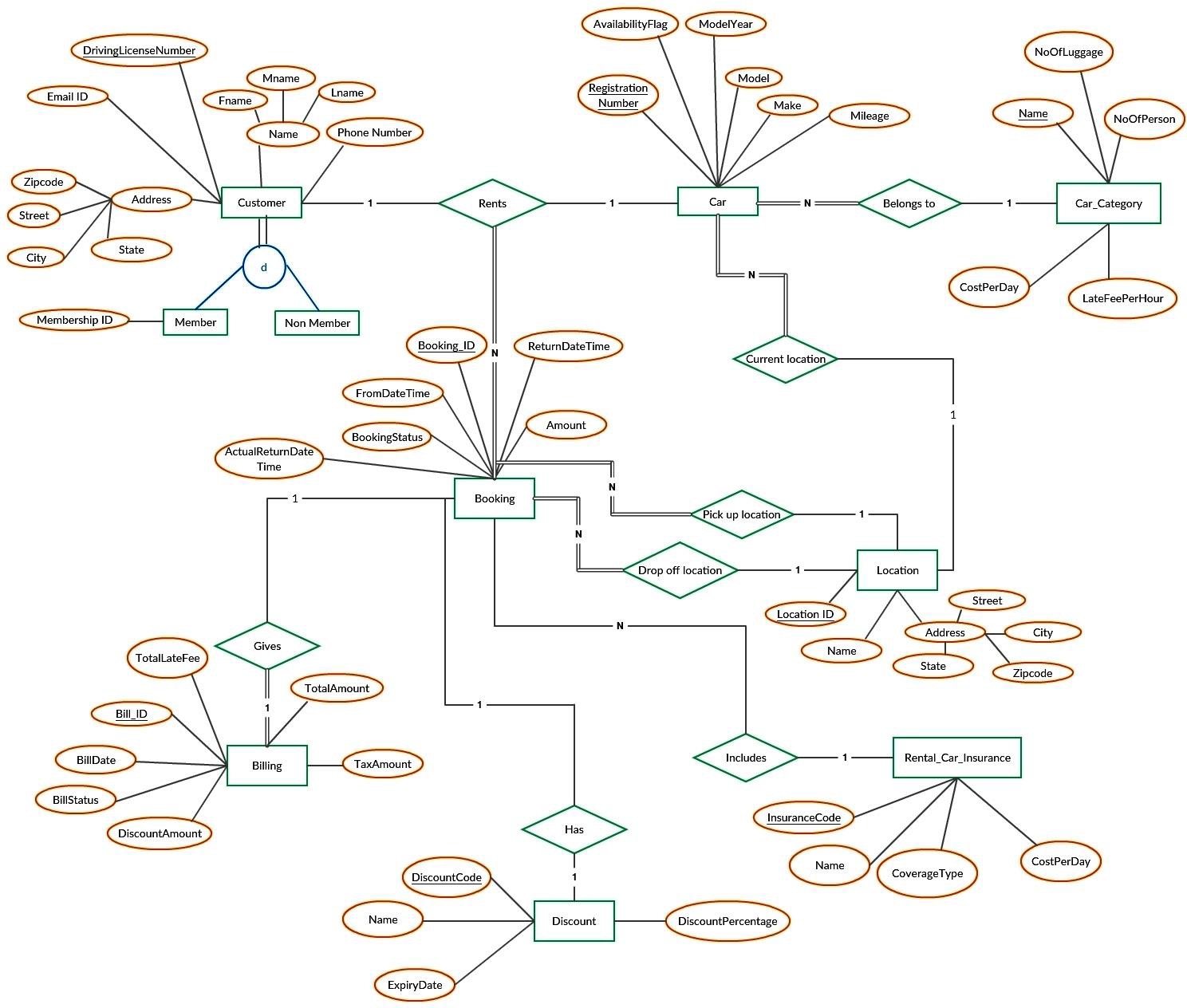
Karthik Reddy Suram-(78)

**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.**

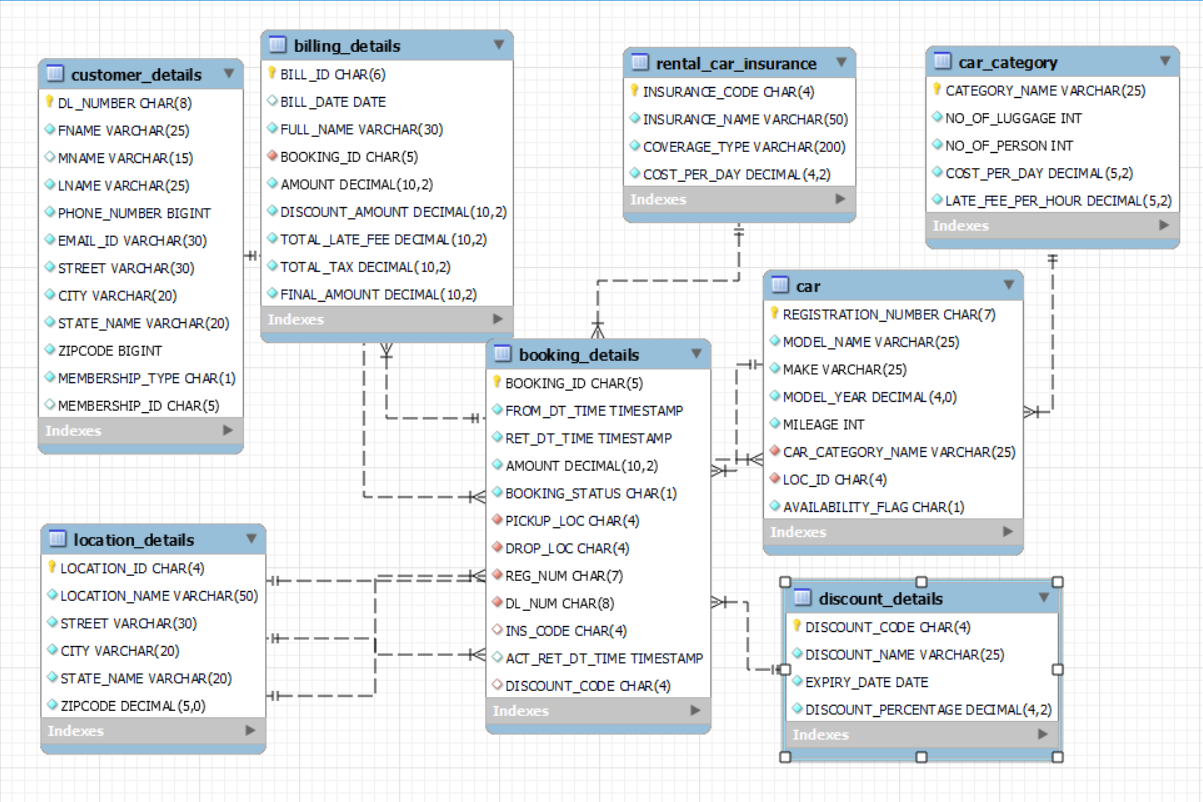
This project presents the design and implementation of a comprehensive car rental management system leveraging MySQL for database management and Python, along with the Streamlit framework, for application development. The system aims to streamline the process of car rental operations, encompassing functionalities 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 likemy 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

A screenshot of a car

Description automatically generated

A screenshot of a computer

Description automatically generated

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

A screenshot of a update car

Description automatically generated

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.”**

A screenshot of a computer

Description automatically generated

As seen in the picture the car details are updated successfully

A screenshot of a website

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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

A screenshot of a car registration form

Description automatically generated

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.”**

A screenshot of a computer program

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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. - A close-up of a car rental system

Description automatically generated

This is the interface used to manage cars:

A screenshot of a car

Description automatically generated

A screenshot of a update car

Description automatically generated

A screenshot of a website

Description automatically generated

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

Below is the interface to manage billing:

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

**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.