Car Fleet Management System

DBMS Project

—

Devanshi Dudhatra | 22BCP171

Heet Dobariya | 22BCP177

G5 | Division 3

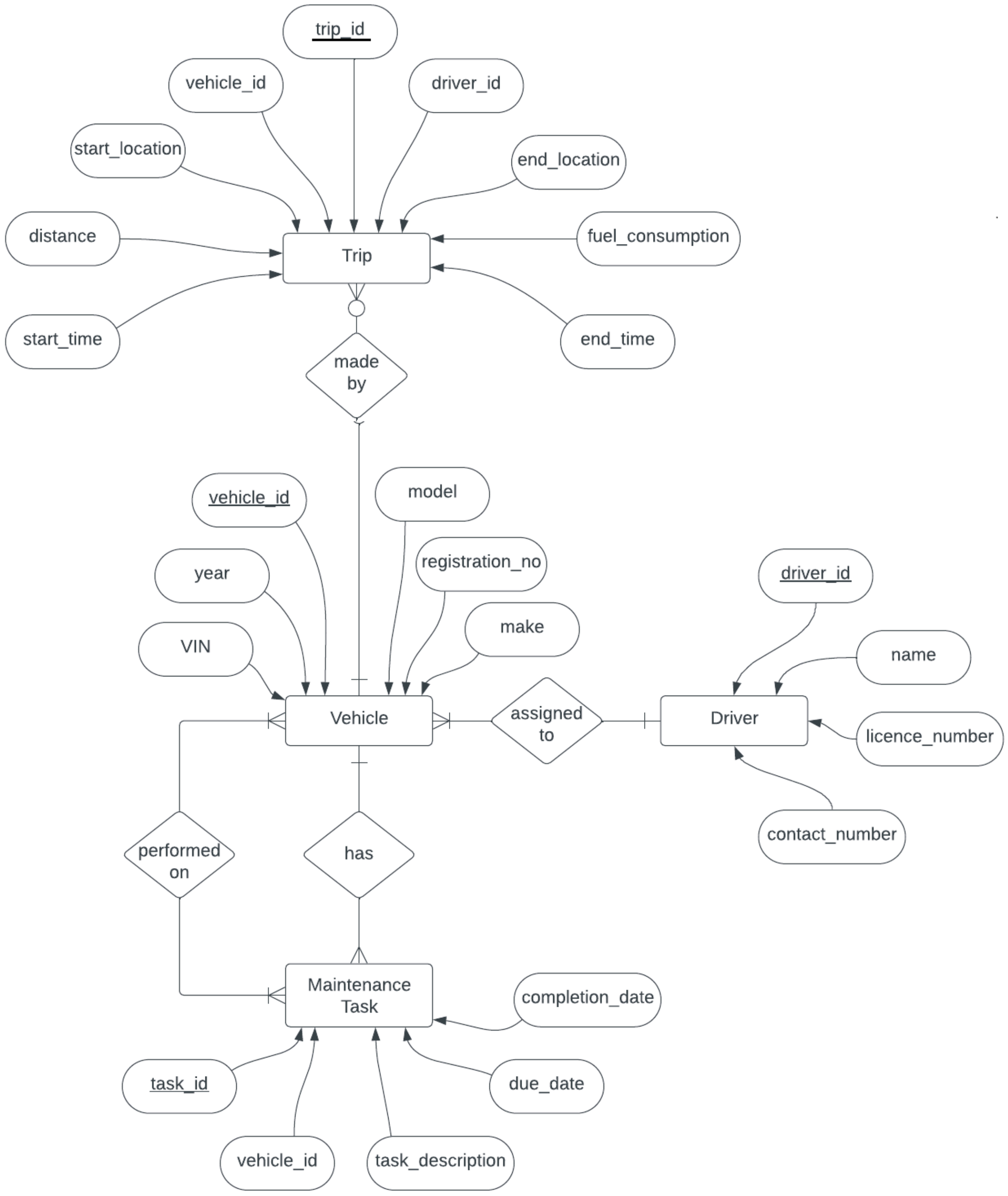
Project Statement

- Project Overview: The project aims to develop a Car Fleet Management System that enables organizations to efficiently manage their vehicle fleet, including tracking vehicle usage, maintenance schedules, fuel consumption, and driver assignments.

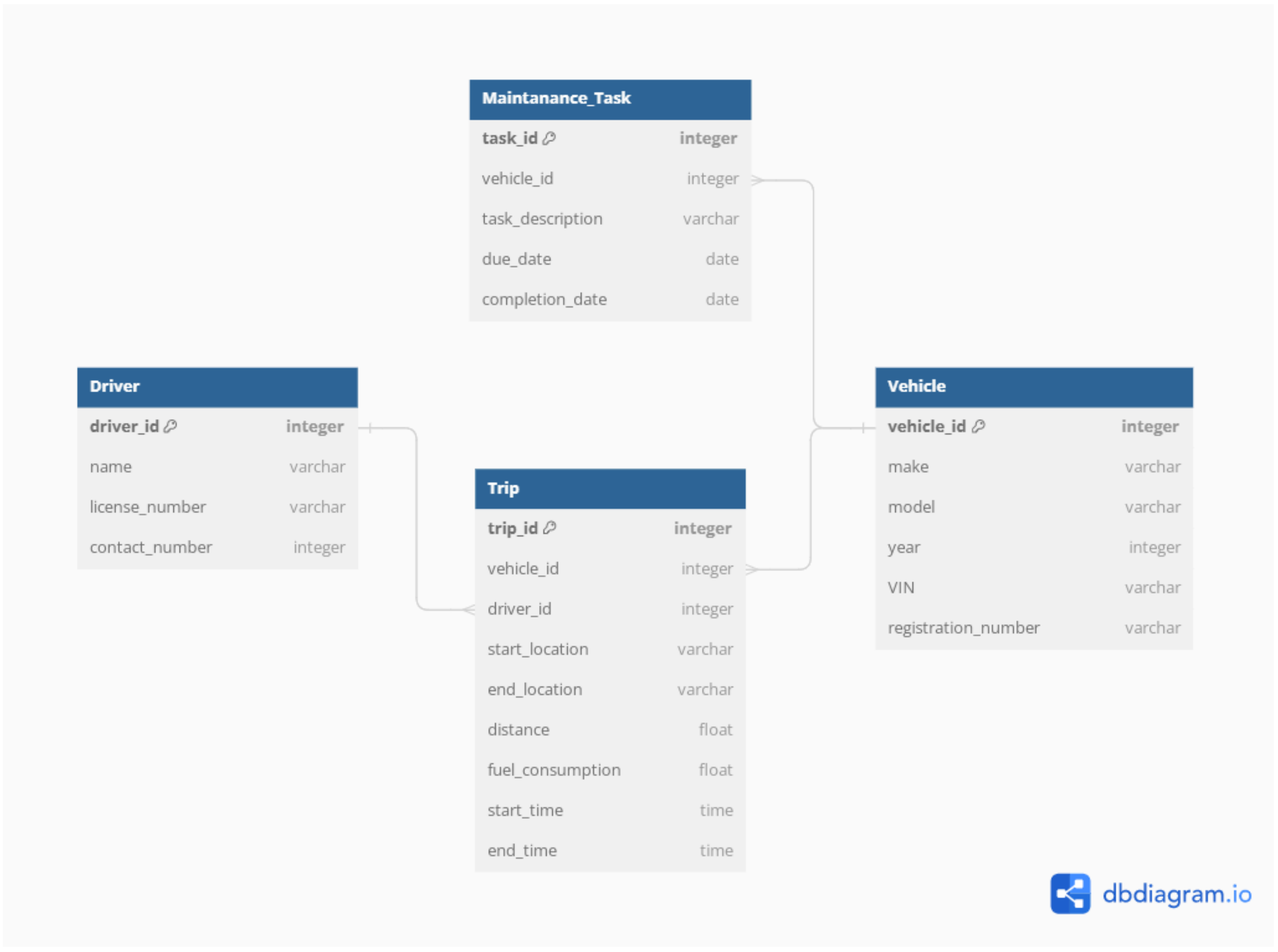
- Use Cases (Applications):
    - Vehicle Management: Add, update, and retire vehicles from the fleet, track vehicle details such as make, model, year, VIN, and registration information.
    - Driver Management: Assign drivers to vehicles, track driver details such as name, license information, and contact details.
    - Maintenance Management: Schedule and track maintenance tasks for vehicles, record maintenance history, and generate alerts for upcoming services.
    - Trip Management: Track trips made by vehicles, including start and end locations, distance traveled, fuel consumption, and driver information.

- Objective: The objective is to create a comprehensive database management system that streamlines fleet operations, enhances vehicle utilization, and reduces maintenance costs.

Entity Relationship Diagram



Relational Model



Normalization

- **For Vehicle (vehicle_id, make, model, year, VIN, registration_number):**
  - Functional Dependencies -

    vehicle_id → make, model, year, VIN, registration_number

    VIN → vehicle_id

    registration_number → vehicle_id

- Primary Key: vehicle_id (correctly identifies each vehicle)
- No partial dependencies on the primary key. Therefore, it is already in 2NF.
- Now as it is in 2NF and it does not have any transitive dependencies i.e. no non-key attribute is dependent on any other non-key attribute. Therefore, it is also in 3NF.
- Highest Normal Form: 3NF

❖ **For Driver (driver_id, name, license_number, contact_number):**

➢ Functional Dependencies -

driver_id → name, license_number, contact_number

license_number → driver_id

- Primary Key: driver_id (correctly identifies each driver)
- No partial dependencies on the primary key. Therefore, it is already in 2NF.
- Now as it in 2NF and it does not have any transitive dependencies i.e. no non-key attribute is dependent on any other non-key attribute. Therefore , it is also in 3NF.
- Highest Normal Form : 3NF

❖ **For MaintenanceTask (task_id, vehicle_id, task_description, due_date, completion_date):**

➢ Functional Dependencies -

task_id → vehicle_id, task_description, due_date, completion_date

vehicle_id → task_id

due_date → task_id

completion_date → task_id

- Primary Key: task_id (uniquely identifies each task)
- Foreign Key: vehicle_id references Vehicle(vehicle_id) (establishes link with a vehicle)

- No partial dependencies on the primary key or foreign key. Therefore it is in 2NF.
- It is also in 3NF.
- Highest Normal Form : 3NF

❖ **For Trip (trip_id, vehicle_id, driver_id, start_location, end_location, distance, fuel_consumption, start_time, end_time):**

➢ Functional Dependencies -

trip_id → vehicle_id, driver_id, start_location, end_location, distance, fuel_consumption, start_time, end_time

vehicle_id → trip_id, driver_id, start_location, end_location, distance, fuel_consumption, start_time, end_time

driver_id → trip_id, vehicle_id, start_location, end_location, distance, fuel_consumption, start_time, end_time

start_location, end_location, start_time → trip_id

end_time → trip_id, start_time

- Primary Key: trip_id (uniquely identifies each trip)
- Foreign Key: vehicle_id references Vehicle(vehicle_id) (establishes link with a vehicle)
- Foreign Key: driver_id references Driver(driver_id) (establishes link with a driver)
- No partial dependencies on the primary key or foreign keys.
- Therefore it is in 2NF.
- Also there are no transitive dependencies. Therefore it is in 3NF also.
- Highest Normal Form : 3NF

SQL Implementation

a. Create tables with necessary integrity constraints using SQL DDL statements.

```
CREATE TABLE Vehicle (
    vehicle_id INT PRIMARY KEY,
    make VARCHAR(50) NOT NULL,
    model VARCHAR(50) NOT NULL,
    year INT NOT NULL,
    VIN VARCHAR(17) UNIQUE NOT NULL,
    registration_number VARCHAR(20) UNIQUE NOT NULL
);

CREATE TABLE Driver (
    driver_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    license_number VARCHAR(20) UNIQUE NOT NULL,
    contact_number VARCHAR(20) NOT NULL
);

CREATE TABLE MaintenanceTask (
    task_id INT PRIMARY KEY,
    vehicle_id INT NOT NULL,
    task_description VARCHAR(200) NOT NULL,
    due_date DATE NOT NULL,
    completion_date DATE,
    FOREIGN KEY (vehicle_id) REFERENCES Vehicle(vehicle_id)
);

CREATE TABLE Trip (
    trip_id INT PRIMARY KEY,
    vehicle_id INT NOT NULL,
    driver_id INT NOT NULL,
    start_location VARCHAR(100) NOT NULL,
    end_location VARCHAR(100) NOT NULL,
    distance DECIMAL(10, 2) NOT NULL,
    fuel_consumption DECIMAL(10, 2) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES Vehicle(vehicle_id),
    FOREIGN KEY (driver_id) REFERENCES Driver(driver_id)
);
```

b. Populate tables with relevant data.

```
INSERT INTO Vehicle VALUES (1, 'Toyota', 'Camry', 2022, '1HGCM82633A004352',
'ABC123');
INSERT INTO Vehicle VALUES (2, 'Honda', 'Civic', 2020, 'JHMEH61600S215688',
'XYZ456');
INSERT INTO Vehicle VALUES (3, 'Ford', 'F-150', 2019, '1FTEW1EP9KFA25866',
'DEF789');
INSERT INTO Vehicle VALUES (4, 'Chevrolet', 'Tahoe', 2021,
'1GNSKBKC0KR105746', 'GHI987');
INSERT INTO Vehicle VALUES (5, 'Nissan', 'Altima', 2018, '1N4AL3AP9JC167719',
'JKL654');

INSERT INTO Driver VALUES (1, 'John Doe', '123456', '9876543210');
INSERT INTO Driver VALUES (2, 'Jane Smith', '654321', '1234567890');
INSERT INTO Driver VALUES (3, 'David Johnson', '789012', '2345678901');
```

INSERT INTO Driver VALUES (4, 'Emily Davis', '890123', '3456789012');

INSERT INTO Driver VALUES (5, 'Michael Brown', '901234', '4567890123');

INSERT INTO Driver VALUES (6, 'Mukesh', '901553', '4565455323');

INSERT INTO MaintenanceTask VALUES (1, 1, 'Oil Change', '2024-03-20', NULL);

INSERT INTO MaintenanceTask VALUES (2, 2, 'Tire Rotation', '2024-03-22', NULL);

INSERT INTO MaintenanceTask VALUES (3, 3, 'Brake Inspection', '2024-03-25', NULL);

INSERT INTO MaintenanceTask VALUES (4, 4, 'Oil Change', '2024-03-28', NULL);

INSERT INTO MaintenanceTask VALUES (5, 5, 'Fluid Check', '2024-03-30', NULL);

INSERT INTO Trip VALUES (1, 1, 1, 'City A', 'City B', 100.5, 10.2, '2024-03-14 09:00:00', '2024-03-14 12:00:00');

INSERT INTO Trip VALUES (2, 2, 2, 'City C', 'City D', 120.8, 9.8, '2024-03-15 10:00:00', '2024-03-15 13:00:00');

INSERT INTO Trip VALUES (3, 3, 3, 'City E', 'City F', 80.3, 12.5, '2024-03-16 11:00:00', '2024-03-16 14:00:00');

INSERT INTO Trip VALUES (4, 4, 4, 'City G', 'City H', 150.2, 11.2, '2024-03-17 12:00:00', '2024-03-17 15:00:00');

INSERT INTO Trip VALUES (5, 5, 5, 'City I', 'City J', 200.5, 10.0, '2024-03-18 13:00:00', '2024-03-18 16:00:00');

c. Write various SQL queries for data retrieval related to the application.

-- Retrieve all vehicles assigned to a specific driver.

SELECT * FROM Vehicle WHERE vehicle_id IN (SELECT vehicle_id FROM Trip WHERE driver_id = 1);

-- Retrieve maintenance tasks due for a specific vehicle.

SELECT * FROM MaintenanceTask WHERE vehicle_id = 1 AND due_date <= CURRENT_DATE;

-- Calculate total distance traveled by a specific vehicle.

SELECT SUM(distance) AS total_distance FROM Trip WHERE vehicle_id = 1;

-- Find average fuel consumption for all trips made by a specific vehicle.

SELECT AVG(fuel_consumption) AS average_fuel_consumption FROM Trip WHERE vehicle_id = 1;

UI

```
import streamlit as st

import sqlite3

from PIL import Image


# Connect to SQLite database

def connect_to_db():

    return sqlite3.connect("FleetManagement.db")


# Function to create tables

def create_tables():

    conn = connect_to_db()

    cursor = conn.cursor()


    cursor.execute("""

    CREATE TABLE IF NOT EXISTS Vehicle (
        vehicle_id INTEGER PRIMARY KEY,
        make TEXT NOT NULL,
        model TEXT NOT NULL,
        year INTEGER NOT NULL,
        VIN TEXT UNIQUE NOT NULL,
        registration_number TEXT UNIQUE NOT NULL
    )
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS Driver (
        driver_id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        license_number TEXT UNIQUE NOT NULL,
        contact_number TEXT NOT NULL
    )
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS MaintenanceTask (
        task_id INTEGER PRIMARY KEY,
        vehicle_id INTEGER NOT NULL,
        task_description TEXT NOT NULL,
        due_date DATE NOT NULL,
        completion_date DATE,
        FOREIGN KEY (vehicle_id) REFERENCES Vehicle(vehicle_id)
    )
```

```
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS Trip (
        trip_id INTEGER PRIMARY KEY,
        vehicle_id INTEGER NOT NULL,
        driver_id INTEGER NOT NULL,
        start_location TEXT NOT NULL,
        end_location TEXT NOT NULL,
        distance DECIMAL(10, 2) NOT NULL,
        fuel_consumption DECIMAL(10, 2) NOT NULL,
        start_time TIMESTAMP NOT NULL,
        end_time TIMESTAMP NOT NULL,
        FOREIGN KEY (vehicle_id) REFERENCES Vehicle(vehicle_id),
        FOREIGN KEY (driver_id) REFERENCES Driver(driver_id)
    )
    """)

    conn.commit()
    conn.close()

# Function to insert data into a table
def insert_data(table_name, data):
    conn = connect_to_db()
    cursor = conn.cursor()
    columns = ', '.join(data.keys())
    placeholders = ', '.join(['?'] * len(data))
    query = f"INSERT INTO {table_name} ({columns}) VALUES
({placeholders})"
    cursor.execute(query, list(data.values()))
    conn.commit()
    conn.close()

# Function to delete data from a table
def delete_data(table_name, condition):
    conn = connect_to_db()
    cursor = conn.cursor()
    query = f"DELETE FROM {table_name} WHERE {condition}"
    cursor.execute(query)
    conn.commit()
    conn.close()

# Function to search data from a table
def search_data(table_name, search_criteria):
    conn = connect_to_db()
    cursor = conn.cursor()
    query = f"SELECT * FROM {table_name} WHERE {search_criteria}"
    cursor.execute(query)
    data = cursor.fetchall()
    conn.commit()
    conn.close()
    return data

# Streamlit UI
def main():
    st.title("Car Fleet Management Database")

    # Sidebar for selecting operation
    operation = st.sidebar.selectbox("Select Operation", ("View ER
Diagram","View Data", "Add Data", "Delete Data","Search Data",))

    if operation == "View Data":
        st.subheader("View Data")

        table_name = st.selectbox("Select Table", ("Vehicle", "Driver",
"MaintenanceTask", "Trip"))

        conn = connect_to_db()
        cursor = conn.cursor()

        if table_name == "Vehicle":
            cursor.execute("SELECT * FROM Vehicle")
            data = cursor.fetchall()
            columns = [description[0] for description in
cursor.description]  # Fetch column names
            st.table([columns] + data)  # Concatenate column names with
data and display

        elif table_name == "Driver":
            cursor.execute("SELECT * FROM Driver")
            data = cursor.fetchall()
            columns = [description[0] for description in
cursor.description]  # Fetch column names
```

```
            st.table([columns] + data)  # Concatenate column names with
data and display

        elif table_name == "MaintenanceTask":
            cursor.execute("SELECT * FROM MaintenanceTask")
            data = cursor.fetchall()
            columns = [description[0] for description in
cursor.description]  # Fetch column names
            st.table([columns] + data)  # Concatenate column names with
data and display

        elif table_name == "Trip":
            cursor.execute("SELECT * FROM Trip")
            data = cursor.fetchall()
            columns = [description[0] for description in
cursor.description]  # Fetch column names
            st.table([columns] + data)  # Concatenate column names with
data and display

        conn.close()

    elif operation == "Add Data":
        st.subheader("Add Data")

        table_name = st.selectbox("Select Table", ("Vehicle", "Driver",
"MaintenanceTask", "Trip"))

        if table_name == "Vehicle":
            st.subheader("Add Vehicle")
            make = st.text_input("Make")

            model = st.text_input("Model")

            year = st.number_input("Year", min_value=1900, max_value=2100)

            VIN = st.text_input("VIN")

            registration_number = st.text_input("Registration Number")


            if st.button("Add Vehicle"):

                insert_data("Vehicle", {

                    "make": make,

                    "model": model,

                    "year": year,

                    "VIN": VIN,

                    "registration_number": registration_number

                })

                st.success("Vehicle added successfully!")


        elif table_name == "Driver":

            name = st.text_input("Name")

            license_number = st.text_input("License Number")

            contact_number = st.text_input("Contact Number")


            if st.button("Add Driver"):

                insert_data("Driver", {

                    "name": name,

                    "license_number": license_number,

                    "contact_number": contact_number

                })
                st.success("Driver added successfully!")

        elif table_name == "MaintenanceTask":
            vehicle_id = st.number_input("Vehicle ID", min_value=1)
            task_description = st.text_input("Task Description")
            due_date = st.date_input("Due Date")

            if st.button("Add Maintenance Task"):
                insert_data("MaintenanceTask", {
                    "vehicle_id": vehicle_id,
                    "task_description": task_description,
                    "due_date": due_date
                })

                st.success("Maintenance Task added successfully!")

        elif table_name == "Trip":
            vehicle_id = st.number_input("Vehicle ID", min_value=1)
            driver_id = st.number_input("Driver ID", min_value=1)
            start_location = st.text_input("Start Location")
            end_location = st.text_input("End Location")
            distance = st.number_input("Distance")
            fuel_consumption = st.number_input("Fuel Consumption")
            start_time = st.text_input("Start Time")
            end_time = st.text_input("End Time")
```

```
            if st.button("Add Trip"):
                insert_data("Trip", {
                    "vehicle_id": vehicle_id,
                    "driver_id": driver_id,
                    "start_location": start_location,
                    "end_location": end_location,
                    "distance": distance,
                    "fuel_consumption": fuel_consumption,
                    "start_time": start_time,
                    "end_time": end_time
                })

                st.success("Trip added successfully!")


    elif operation == "Delete Data":
        st.subheader("Delete Data")

        table_name = st.selectbox("Select Table", ("Vehicle", "Driver",
"MaintenanceTask", "Trip"))

        conn = connect_to_db()
        cursor = conn.cursor()

        if table_name == "Vehicle":
            st.subheader("Delete Vehicle")
            vehicle_id = st.number_input("Vehicle ID", min_value=1)
            if st.button("Delete Vehicle"):
                delete_data("Vehicle", f"vehicle_id = {vehicle_id}")
                st.success("Vehicle deleted successfully!")

        elif table_name == "Driver":
            st.subheader("Delete Driver")
            name = st.text_input("Name")
            if st.button("Delete Driver"):
                delete_data("Driver", f"name = '{name}'")
                st.success("Driver deleted successfully!")

        elif table_name == "MaintenanceTask":
            st.subheader("Delete Maintenance Task")
            vehicle_id = st.number_input("Vehicle ID", min_value=1)
            if st.button("Delete Maintenance Task"):
                delete_data("MaintenanceTask", f"vehicle_id =
{vehicle_id}")
                st.success("Task deleted successfully!")

        elif table_name == "Trip":
            st.subheader("Delete Trip")
            start_location = st.text_input("Start Location")
            end_location = st.text_input("End Location")
            if st.button("Delete Trip"):
                delete_data("Trip", f"start_location = '{start_location}'
and end_location = '{end_location}'")
                st.success("Trip deleted successfully!")

    elif operation == "Search Data":
        st.subheader("Search Data")

        table_name = st.selectbox("Select Table", ("Vehicle", "Driver",
"MaintenanceTask", "Trip"))

        conn = connect_to_db()
        cursor = conn.cursor()

        if table_name == "Vehicle":
            st.subheader("Search Vehicle")
            vehicle_id = st.number_input("Vehicle ID")
            model = st.text_input("Model")
            VIN = st.text_input("VIN")
            registration_number = st.text_input("Registration Number")
            if st.button("Search Vehicle"):
                data = search_data("Vehicle", f"vehicle_id =
{vehicle_id}")
                model_data = search_data("Vehicle" , f"model = '{model}'")
                vin_data = search_data("Vehicle" , f"VIN = '{VIN}'")
                reg_data = search_data("Vehicle" , f"registration_number =
'{registration_number}' ")
                if data:
                    st.table(data)
                elif model_data:
                    st.table(model_data)
                elif vin_data:
                    st.table(vin_data)
                elif reg_data:
                    st.table(reg_data)
                else:
                    st.warning("Vehicle not found.")

        elif table_name == "Driver":
            st.subheader("Search Driver")
            driver_id = st.number_input("Driver ID")
            name = st.text_input("Name")
            license_number = st.text_input("License Number")
            contact_number = st.text_input("Contact Number")

            if st.button("Search Driver"):
                data = search_data("Driver", f"driver_id = {driver_id}")
                name_data = search_data("Driver" , f"name = '{name}' ")
                lic_data = search_data("Driver" , f"license_number =
'{license_number}'")
                contact_data = search_data("Driver" , f"contact_number =
'{contact_number}'")
                if data:
                    st.table(data)
                elif name_data:
                    st.table(name_data)
                elif lic_data:
                    st.table(lic_data)
                elif contact_data:
                    st.table(contact_data)
                else:
                    st.warning("Driver not found.")
        elif table_name == "MaintenanceTask":
            st.subheader("Search Maintenance Task")
            vehicle_id = st.number_input("Vehicle ID")
            due_date = st.date_input("Due Date")
            if st.button("Search Maintenance Task"):
                data = search_data("MaintenanceTask", f"vehicle_id =
{vehicle_id}")
                date_data = search_data("MaintenanceTask" , f"due_date =
{due_date}")
                if data:
                    st.table(data)
                elif date_data:
                    st.table(date_data)
                else:
                    st.warning("Maintenance task not found.")

        elif table_name == "Trip":
            st.subheader("Search Trip")
            start_location = st.text_input("Start Location")
            end_location = st.text_input("End Location")
            if st.button("Search Trip"):
                data = search_data("Trip", f"start_location =
'{start_location}'")
                end_data = search_data("Trip", f"end_location =
'{end_location}'")
                if data:
                    st.table(data)
                elif end_data:
                    st.table(end_data)
                else:
                    st.warning("Trip not found.")
            conn.close()

    elif operation == "View ER Diagram":
        img = Image.open("ER Diagram.png")
        st.image(img, width=700)




if __name__ == "__main__":
    # Set background color using set_page_config
    st.set_page_config(layout="wide",page_title ="Fleet Management App",
page_icon=":car:",
                   initial_sidebar_state="expanded")

    create_tables()
    main()
```