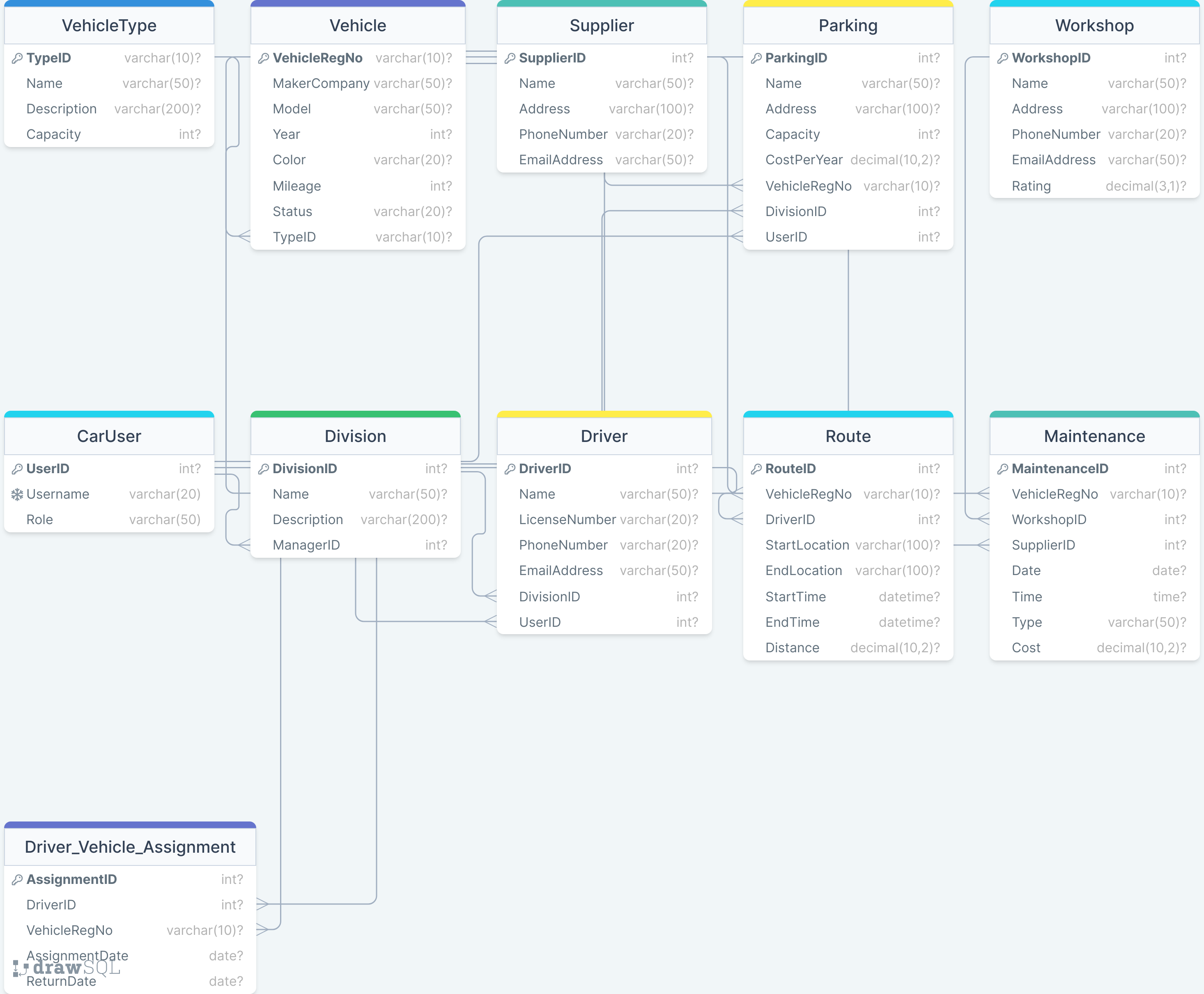
***Vehicle fleet Management System Database***

Vehicle fleet Management System DB using MS SQL Server, which is a Relational Database Management System (RDBMS) developed by Microsoft.

***ER Diagram*** - An ER diagram is a visual representation of the entities (objects or concepts) within a system and the relationships between them.



URL - https://drawsql.app/teams/shyed-shahriar/diagrams/mce-079-05536-shyed-assignment-17-05-2023

***To download Schema diagram of DB tables, please visit*** - [VhicleFleetMangtDB - diagrams.net](https://app.diagrams.net/#G1o5WEAGhWnzGS14jKRA2PK1wotCeQRSYA) or <https://drive.google.com/file/d/1o5WEAGhWnzGS14jKRA2PK1wotCeQRSYA/view?usp=sharing>

**View**

-- A view is a virtual table that shows the result of a query.

-- Create a view that shows the total cost of maintenance for each vehicle

go

CREATE VIEW MaintenanceCost AS

SELECT VehicleRegNo, SUM(Cost) AS TotalCost

FROM Maintenance

GROUP BY VehicleRegNo;

go

--to see the view table

select \* from MaintenanceCost;

go

-- Create a view that shows the details of each driver and their division

CREATE VIEW DriverDetails AS

SELECT d.DriverID, d.Name, d.LicenseNumber, d.PhoneNumber, d.EmailAddress, dv.Name AS DivisionName, dv.Description AS DivisionDescription

FROM Driver d

JOIN Division dv ON d.DivisionID = dv.DivisionID;

go

select \* from DriverDetails;

go

-- Query the MaintenanceCost view to get the total cost of maintenance for vehicle 1

SELECT TotalCost FROM MaintenanceCost WHERE VehicleRegNo = 'DHK-3456';

-- Query the DriverDetails view to get the name and division of driver 2

SELECT Name, DivisionName FROM DriverDetails WHERE DriverID = 2;

-- Create a view that shows the route summary

go

CREATE VIEW RouteSummary AS

SELECT r.VehicleRegNo, COUNT(\*) AS RouteCount, SUM(r.Distance) AS TotalDistance

FROM Route r

GROUP BY r.VehicleRegNo;

go

-- Query the RouteSummary view to get the number of routes and total distance for vehicle 'DHK-7890'

SELECT RouteCount, TotalDistance

FROM RouteSummary

WHERE VehicleRegNo = 'DHK-7890';

**Trigger**

A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server.

go

CREATE TRIGGER CheckVehicleStatus

ON Route

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS(

SELECT 1

FROM inserted i

JOIN Vehicle v ON i.VehicleRegNo = v.VehicleRegNo

WHERE v.Status <> 'available'

)

BEGIN

RAISERROR('Vehicle is not available for route', 16, 1);

ROLLBACK TRANSACTION;

END

ELSE

BEGIN

INSERT INTO Route (VehicleRegNo, DriverID, StartLocation, EndLocation, StartTime, EndTime, Distance)

SELECT VehicleRegNo, DriverID, StartLocation, EndLocation, StartTime, EndTime, Distance

FROM inserted;

END;

END;

go

CREATE TRIGGER UpdateVehicleMileage ON Route

AFTER INSERT

AS

BEGIN

UPDATE Vehicle SET Mileage = Mileage + i.Distance FROM Vehicle JOIN inserted i ON Vehicle.VehicleRegNo = i.VehicleRegNo;

END;

go

INSERT INTO Route (VehicleRegNo, DriverID, StartLocation, EndLocation, StartTime, EndTime, Distance)

VALUES ('DHK-7890', 1, 'A', 'B', '2022-01-01 10:00:00', '2022-01-01 11:00:00', 50);

go

SELECT table\_name FROM INFORMATION\_SCHEMA.TABLES WHERE table\_type = 'BASE TABLE' -- list all the tables in the database.

go

CREATE TRIGGER UpdateVehicleMileage ON Route

AFTER INSERT

AS

BEGIN

UPDATE Vehicle SET Mileage = Mileage + i.Distance FROM Vehicle JOIN inserted i ON Vehicle.VehicleRegNo = i.VehicleRegNo;

END;

GO

INSERT INTO Route (VehicleRegNo, DriverID, StartLocation, EndLocation, StartTime, EndTime, Distance)

VALUES ('DHK-6789', 1, 'A', 'B', '2022-01-01 10:00:00', '2022-01-01 11:00:00', 150);

go

CREATE TRIGGER ScheduleMaintenance ON Vehicle

AFTER UPDATE AS

BEGIN

IF UPDATE(Mileage)

INSERT INTO Maintenance (VehicleRegNo, WorkshopID, SupplierID, Date, Time, Type, Cost)

SELECT i.VehicleRegNo, 1, 1, GETDATE(), GETDATE(), 'Full service', 5000.00

FROM inserted i JOIN deleted d ON i.VehicleRegNo = d.VehicleRegNo

WHERE i.Mileage >= 100000 AND d.Mileage < 100000;

END;

GO

CREATE TRIGGER ScheduleMaintenance2 ON Vehicle

AFTER UPDATE AS

BEGIN

IF UPDATE(Mileage)

INSERT INTO Maintenance (VehicleRegNo, WorkshopID, SupplierID, Date, Time, Type, Cost)

SELECT i.VehicleRegNo, 1, 1, GETDATE(), GETDATE(), 'Full service', 5000.00

FROM inserted i JOIN deleted d ON i.VehicleRegNo = d.VehicleRegNo

WHERE i.Mileage >= 1000 AND d.Mileage < 1000;

END;

GO

UPDATE Vehicle

SET Mileage = 100000

WHERE VehicleRegNo = 'DHK-7890';

UPDATE Vehicle

SET Mileage = 1000

WHERE VehicleRegNo = 'DHK-6789';

UPDATE Vehicle SET Mileage = 100000 WHERE VehicleRegNo = 'DHK-1234';

go

--CREATE TRIGGER ScheduleMaintenance ON Vehicle AFTER UPDATE AS - This line starts the creation of a new trigger named ScheduleMaintenance that will be executed on the Vehicle table after an UPDATE operation.

--IF UPDATE(Mileage) - This line checks if the Mileage column was updated during the UPDATE operation.

--INSERT INTO Maintenance (VehicleRegNo, WorkshopID, SupplierID, Date, Time, Type, Cost) - This line specifies the columns of the Maintenance table that will be inserted with new data.

--SELECT i.VehicleRegNo, 1, 1, GETDATE(), GETDATE(), 'Full service', 5000.00 - This line specifies the values to be inserted into the columns specified in the previous line. In this case, the VehicleRegNo column will be set to the VehicleRegNo value of the row being updated, the WorkshopID and SupplierID columns will be set to 1, the Date and Time columns will be set to the current date and time using the GETDATE() function, the Type column will be set to 'Full service', and the Cost column will be set to 5000.00.

--FROM inserted i JOIN deleted d ON i.VehicleRegNo = d.VehicleRegNo - This line specifies that the inserted and deleted tables will be joined on the VehicleRegNo column, which will allow the trigger to compare the old and new values of the Mileage column.

--WHERE i.Mileage >= 100000 AND d.Mileage < 100000 - This line specifies that the trigger will only insert a new row into the Maintenance table if the updated Mileage value is greater than or equal to 100000 and the previous Mileage value was less than 100000.

--END; - This line marks the end of the trigger code.

--GO - This line signals the end of the CREATE TRIGGER statement and executes the query.

--Overall, this trigger will check if the Mileage column of a Vehicle row was updated to a value greater than or equal to 100000, and if so, it will insert a new row into the Maintenance table with information about the vehicle, workshop, supplier, date, time, type of service, and cost.

--To fire the trigger that inserts a new row into the Maintenance table when a vehicle's mileage reaches 100000,

UPDATE Vehicle SET Mileage = 100000 WHERE VehicleRegNo = 2;

go

**Stored procedure**

A stored procedure is a named and pre-compiled collection of SQL statements and procedural logic that is stored in a database. It is a reusable and self-contained unit of code that can be called and executed multiple times.

-- Create stored procedure

CREATE PROCEDURE GetVehicleTypeById

@TypeID VARCHAR(10)

AS

BEGIN

-- Select the vehicle type based on the given TypeID

SELECT \* FROM VehicleType WHERE TypeID = @TypeID;

END;

-- Execute the stored procedure

EXEC GetVehicleTypeById '1'; ---show store procedure

--This stored procedure is named GetVehicleTypeById and takes a parameter @TypeID of type VARCHAR(10). It retrieves the vehicle type information from the VehicleType table based on the provided TypeID. To execute the stored procedure and get the vehicle type with TypeID 1, you can use the EXEC

--A stored procedure in MS SQL Server is a named group of SQL statements that are compiled and stored in the database. It allows you to encapsulate a series of SQL statements into a single program unit, which can be executed repeatedly. Stored procedures enhance performance, security, and code reusability.

--Here's an example of a stored procedure that incorporates your previous query for creating a database named "VehicleFleetMngtDB":

-- Create a stored procedure

CREATE PROCEDURE CreateVehicleFleetDatabase

AS

BEGIN

-- Check if the database already exists

IF NOT EXISTS (SELECT 1 FROM sys.databases WHERE name = 'VehicleFleetMngtDB')

BEGIN

-- Create the database

CREATE DATABASE VehicleFleetMngtDB;

PRINT 'VehicleFleetMngtDB created successfully.';

END

ELSE

BEGIN

PRINT 'VehicleFleetMngtDB already exists.';

END

END;

--In this stored procedure, we first check if the database "VehicleFleetMngtDB" already exists. If it doesn't exist, we create the database and print a success message. If the database already exists, we print a message indicating that it already exists.

-- Execute the stored procedure

EXEC CreateVehicleFleetDatabase;

--Executing the stored procedure will create the "VehicleFleetMngtDB" database if it doesn't already exist, or print a message indicating that it already exists.