Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)
School of Computing
B.Tech. – Computer Science and Engineering

VTR UGE2021- (CBCS)

Academic Year: 2025–2026

SUMMER SEMESTER - SS2526

Course Code : 10211CS207

Course Name : Database Management Systems

Slot No : S1L4

DBMS TASK - 12 REPORT

Title: Conceptual Design through FTR

Submitted by:

VTUNO	REGISTER NUMBER	STUDENT NAME
VTU29513	24UECS0301	SAMUDRALA SUPRAJITH

Design entities and relationships for vehicles, owners, services, and technicians.

Aim

To design an **Entity – Relationship (ER) model** for a *Vehicle Service and Maintenance Tracker* that records details of vehicles, owners, services, and technicians.

Entities and Attributes

Owner

- Owner_ID (PK)
- Name
- Phone
- Email
- Address

Vehicle

- Vehicle_ID (PK)
- Owner_ID (FK)
- Make
- Model
- Year
- Registration_No
- VIN
- Mileage

Technician

- Technician_ID (PK)
- Name
- Skill_Specialization
- Contact_No

- Experience_Years

Service

- Service_ID (PK)
- Vehicle_ID (FK)
- Technician_ID (FK)
- Service_Date
- Service_Type
- Service_Details
- Cost
- Next_Service_Date

Relationships

Owner → Vehicle: One-to-Many (An owner can have multiple vehicles)
Vehicle → Service: One-to-Many (A vehicle can have multiple service records)
Technician → Service: One-to-Many (A technician can perform many services)

Normalize the Schema and List All Functional Dependencies Identified

Aim:

To identify all functional dependencies in the Vehicle Service and Maintenance Tracker database and normalize the schema up to Third Normal Form (3NF) to eliminate redundancy and ensure data integrity.

1. Functional Dependencies

Owner

Owner_ID → Name, Phone, Email, Address

Vehicle

Vehicle_ID → Owner_ID, Make, Model, Year, Registration_No, VIN, Mileage

Technician

Technician_ID → Name, Skill_Specialization, Contact_No, Experience_Years

Service

Service_ID → Vehicle_ID, Technician_ID, Service_Date, Service_Type, Service_Details, Cost, Next_Service_Date

2. Normalization Steps

First Normal Form (1NF)

All attributes are atomic (no repeating or multivalued fields).
Each entity already satisfies 1NF.

Second Normal Form (2NF)

All non-key attributes are fully dependent on the primary key.
Each table has a single-column primary key, so there is no partial dependency.
All entities satisfy 2NF.

Third Normal Form (3NF)

No transitive dependencies (non-key attributes depend only on the primary key).
Each non-key attribute depends directly on its table’s primary key.
Therefore, all entities satisfy 3NF.

3. Final Normalized Tables

Table Name	Primary Key	Foreign Key(s)	Description

Table Name	Primary Key	Foreign Key(s)	Description
Owner	Owner_ID	—	Stores owner details
Vehicle	Vehicle_ID	Owner_ID	Stores vehicle details
Technician	Technician_ID	—	Stores technician details
Service	Service_ID	Vehicle_ID, Technician_ID	Stores service records

Result

All tables are normalized up to **Third Normal Form (3NF)**, which removes redundancy, avoids anomalies, and ensures data consistency.

SQL Queries to Record Maintenance Activities and Retrieve Vehicle History

Aim

To write SQL queries for recording vehicle maintenance activities and retrieving complete vehicle service history from the Vehicle Service and Maintenance Tracker database.

1. Record a Maintenance Activity

The following SQL query inserts a new service record into the **Service** table:

```
INSERT INTO Service (
    Service_ID, Vehicle_ID, Technician_ID, Service_Date,
    Service_Type, Service_Details, Cost, Next_Service_Date
)VALUES( 'SRV1001', 'VH2001', 'TECH01', '2025-10-25',
    'Oil Change', 'Engine oil and filter replaced', 120.00, '2026-04-25');
```

2. Retrieve Vehicle Service History

This query retrieves the complete service history of a specific vehicle:

```
SELECT
    V.Vehicle_ID,
    V.Make,
    V.Model,
    S.Service_ID,
    S.Service_Date,
    S.Service_Type,
    S.Service_Details,
    S.Cost,
    T.Name AS Technician_Name
FROM
    Vehicle V
JOIN
    Service S ON V.Vehicle_ID = S.Vehicle_ID
JOIN
    Technician T ON S.Technician_ID = T.Technician_ID
WHERE
    V.Vehicle_ID = 'VH2001'
```

```
ORDER BY
    S.Service_Date DESC;
```

3. List Vehicles Due for Service

This query lists all vehicles whose next service date has arrived or already passed:

```
SELECT
    V.Vehicle_ID,
    V.Make,
    V.Model,
    S.Next_Service_Date
FROM
    Vehicle V
JOIN
    Service S ON V.Vehicle_ID = S.Vehicle_ID
WHERE
    S.Next_Service_Date <= CURDATE();
```

Transaction Management During Simultaneous Service Updates

Aim

To explain how transaction management ensures data consistency, accuracy, and reliability when multiple users simultaneously update service records in the Vehicle Service and Maintenance Tracker database

1. Introduction

When multiple users (such as technicians or managers) update service records at the same time, it may cause problems like **data inconsistency**, **lost updates**, or **dirty reads**.
Transaction management and concurrency control techniques are used to prevent these issues and ensure the database remains accurate and reliable.

2. ACID Properties of Transactions

Property	Description
Atomicity	Ensures that all operations within a transaction are completed successfully. If any part fails, the entire transaction is rolled back.
Consistency	Ensures that the database remains in a valid state before and after the transaction.
Isolation	Ensures that concurrent transactions do not interfere with each other’s operations.
Durability	Ensures that once a transaction is committed, the changes remain permanent even after a system crash.

3. Example of Transaction Management

The following SQL code demonstrates a transaction that updates a service record and logs the update operation:

```
START TRANSACTION;

UPDATE Service
SET Cost = 150.00
WHERE Service_ID = 'SRV1001';

INSERT INTO ServiceLog (Service_ID, Update_Timestamp, Updated_By)
VALUES ('SRV1001', NOW(), 'AdminUser');

COMMIT;
```

4. Isolation Levels

SQL provides isolation levels to control how and when changes made by one transaction are visible to others:

Isolation Level	Description	Prevents
READ UNCOMMITTED	Transactions can read uncommitted changes from others.	None
READ COMMITTED	Only committed data can be read.	Dirty Reads
REPEATABLE READ	Ensures consistent data during a single transaction.	Non-repeatable Reads
SERIALIZABLE	Executes transactions one after another, ensuring full consistency.	All anomalies

5. Importance in Vehicle Service Management System

Prevents conflicts when two technicians update the same service record.
Ensures accurate billing and prevents duplicate entries.
Maintains consistent history of all vehicle services.
Provides auditability through transaction logs.

Perform CRUD operations in a document-based database for vehicle records

Aim

To demonstrate how CRUD (Create, Read, Update, Delete) operations can be performed on vehicle records using a document-based database such as **MongoDB** for the Vehicle Service and Maintenance Tracker.

1. Introduction

A document-based database like **MongoDB** stores data in flexible, JSON-like documents instead of relational tables.
This approach allows complex, nested data (such as vehicle details and service history) to be stored in a single document, making it ideal for applications involving dynamic and hierarchical data structures.

2. Example Database Structure

Database: `vehicleServiceDB`
Collection: `vehicles`

Each vehicle document stores information about the owner, vehicle details, and service records.

3. CRUD Operations

A. Create – Insert a New Vehicle Record

```
db.vehicles.insertOne({
  vehicle_id: "VH2001",
  owner: {
    owner_id: "OWN01",
    name: "John Doe",
    contact: "9876543210"
  },
  make: "Toyota",
  model: "Corolla",
  year: 2020,
  services: [
    {
      service_id: "SRV1001",
      date: "2025-10-25",
      type: "Oil Change",
      technician: { tech_id: "TECH01", name: "Alex Smith" },
      cost: 120,
      next_service_date: "2026-04-25"
    }
  ]
});
```

Explanation:
Inserts a new vehicle record with owner details and an array of service records.

B. Read – Retrieve Vehicle Service History

```
db.vehicles.find(
  { vehicle_id: "VH2001" },
  { _id: 0, services: 1 }
);
```

Explanation:
Retrieves all service details for the vehicle with ID VH2001.

C. Update – Add a New Service Record

```
db.vehicles.updateOne(
  { vehicle_id: "VH2001" },
  {
    $push: {
      services: {
        service_id: "SRV1002",
        date: "2025-11-20",
        type: "Brake Inspection",
        technician: { tech_id: "TECH02", name: "Maria Lee" },
        cost: 200,
        next_service_date: "2026-05-20"
      }
    }
  }
);
```

Explanation:
Adds a new service record to the services array for the existing vehicle.

D. Delete – Remove a Vehicle Record

```
db.vehicles.deleteOne({ vehicle_id: "VH2001" });
```

Explanation:
Removes the entire vehicle record (including all its service details) from the collection.

4. Advantages of Using a Document-Based Database

Feature	Description
Flexibility	Data structure can evolve without schema changes.
Performance	Faster read/write for hierarchical or nested data.
Scalability	Easily supports large-scale, distributed systems.
Embedded Relationships	Stores related data (like owner and service info) together in one document.