Project Report – Logistics Maintenance Database Design

Section 1 – Project Overview

Our team was contracted by **ADR Logistics**, a fleet operator managing vehicles across diverse routes and conditions. The client needs a centralised system to track vehicle status, maintenance history, sensor data, and delivery performance (Laudon and Laudon, 2020). Currently, their data is siloed and underused, limiting timely insight.

To inform our design, we're using a logistics maintenance dataset from Kaggle. Though simulated, it mirrors real fleet data—covering vehicle details, maintenance types, sensor readings, and performance metrics like delivery times and predictive scores. It offers a strong base for developing a practical, scalable database for real-time decision-making.

Our solution centres on a clean, relational structure that enables reporting, analysis, and potential automation. It's built to reduce redundancy, manage varied data inputs, and help the client monitor vehicle health, track trips, and minimise downtime.

Section 2 - Logical Design

The section outlines the logical schema of the database based on the logistics maintenance dataset. The goal of the design is to organize data efficiently to support fleet operations, maintenance tracking, and performance diagnostics. It includes identifying entities, defining their attributes, outlining their relationships, and choosing appropriate data types and formats.

We based our database design on a structured CSV dataset from Kaggle (Kaggle, n.d.). It includes over 25 columns covering everything from vehicle specifications and maintenance logs to sensor readings and delivery performance. We used this to identify four core entities: Vehicle, Maintenance, Trip_Info, and Sensor_Data.

The data helped us shape a model that separates these components into clean tables with clear relationships—using Vehicle_ID as the main link across different tables. While the dataset is simulated, it's well-organised and mirrors what a real transport company would collect. It includes numeric, categorical, and date-based fields, which we've mapped to suitable SQL data types like VARCHAR, FLOAT, and DATE.

We picked this dataset because it's well-labeled, practical, and relevant to the problem we're solving. It also gives us enough variety to think through things like maintenance triggers, downtime tracking, and efficiency metrics—without needing to create hypothetical data from scratch.

2.1 Entity Identification and Purpose

We extracted four core entities from the dataset: Vehicle, Maintenance, Trip_Info, and Sensor_Data. Each entity represents a key component of fleet operation and maintenance. Each table represents a different aspect and type of operational data associated with (logistics/maintenance/our big goal). The 'Vehicle_ID' field would be the primary key for the Vehicle table, and a foreign key to the remaining tables. This one-to-many relationship structure would ensure data integrity and allow for efficient querying across both vehicle usage, maintenance and sensor history.

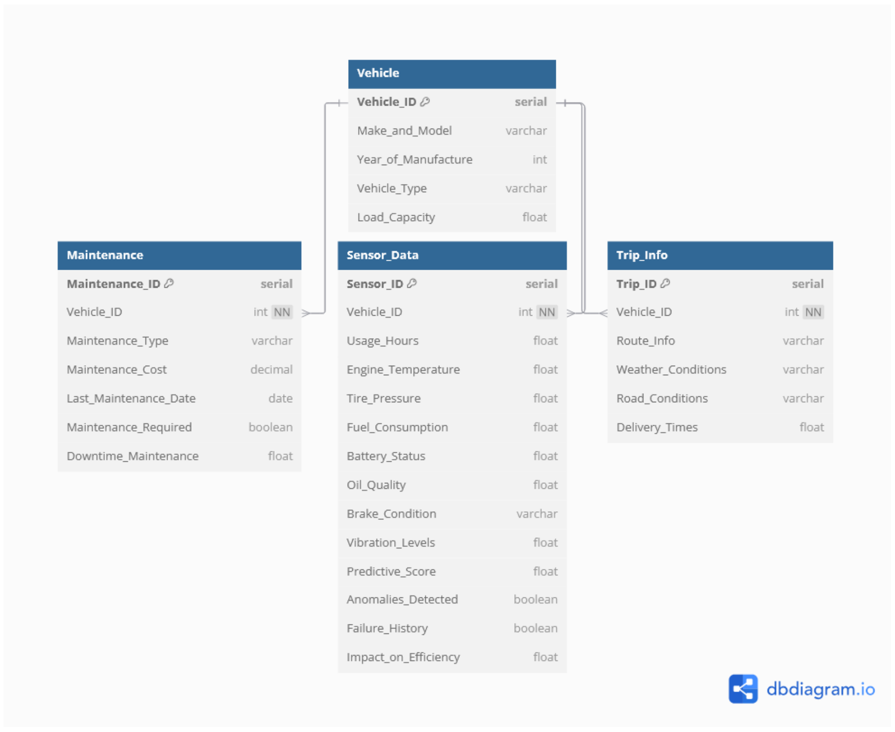


Figure 1. Database relationship diagram showing the connections between the different tables derived from the dataset (Holistics Software, 2025).

2.2 Relationships and Cardinality

Relationships between the entities have been carefully defined to maintain referential integrity and support query efficiency (Elmasri and Navathe, 2016):

One vehicle has many Maintenance records(1:M)

One Vehicle has many Trip_Info records(1:M)

One Vehicle has many Sensor_Data records(1:M)

2.3 Justification for Data Formats

Data types were chosen to optimize accuracy and storage (Connolly & Begg, 2015):

INT for unique IDs, usage counters, and other whole number fields that don't require decimal precision. Examples of the variables include vehicle_id, maintenance_id, year_of_manufacture

VARCHAR for text-based labels. Examples of the variables include make_and_model, vehichle_type, route_info, weather_conditions.

FLOAT for numerical and currency fields. Examples of the variables include capacity, delivery_times, maintenance_cost.

DATE for timestamps. The only variable with this format is last_maintenance_date.

BOOLEAN for binary conditions. Examples of the variables include maintenance_required, anomalies_detected

3. Proposed Database Build

3.1 Database Model Design

The proposed database is a normalized relational model consisting of four main tables: Vehicle, Maintenance, Trip_Info, and Sensor_Data. The schema eliminates redundancy, enforces referential integrity, and supports scalability.

3.1.1 Table Structure & Relationships

We structured the schema using Third Normal Form (3NF) to remove redundant data and keep relationships clean and efficient, following standard database design principles (Connolly and Begg, 2015).

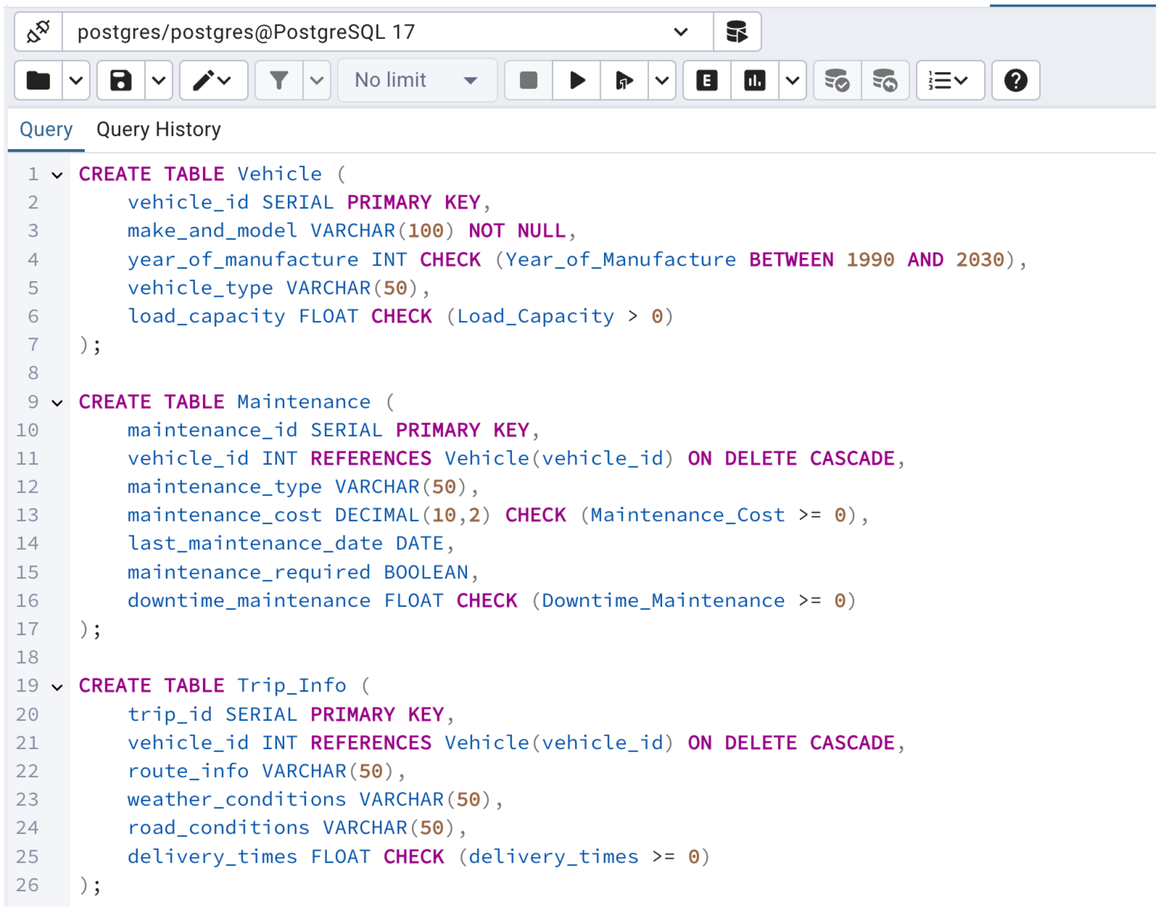
Relational Overview:

Each table has primary keys (PK) and foreign key (FK) constraints to enforce entity relationships and maintain data integrity.

3.2 SQL Schema Implementation

The SQL schema was designed to reflect the logical structure of the data, with primary and foreign keys used to manage relationships between tables and constraints added to ensure data integrity (Rob and Coronel, 2007).

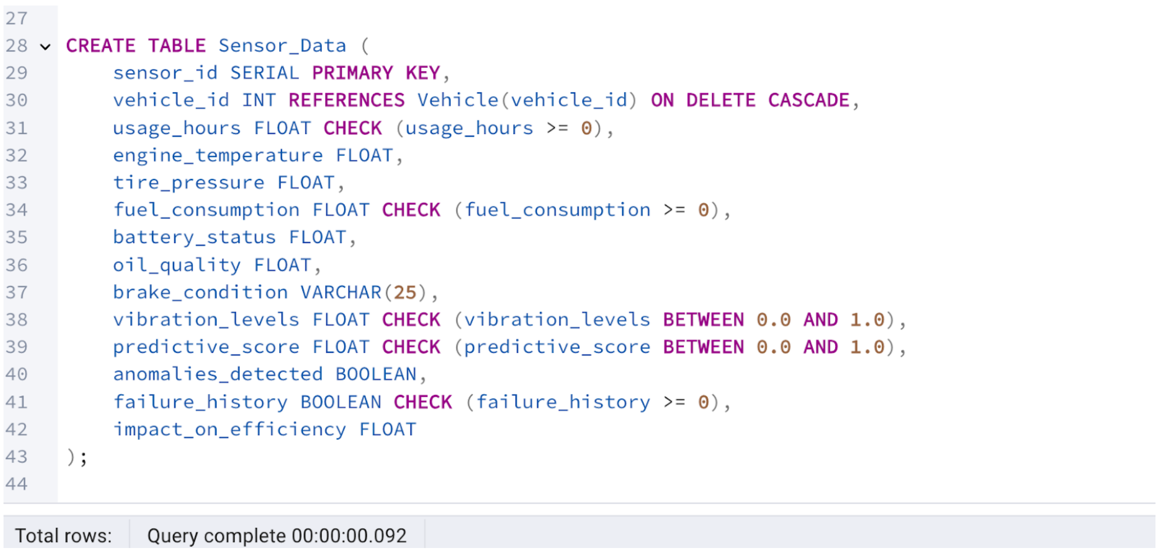
Below is the SQL CREATE TABLE script with constraints and relationships.



```sql
CREATE TABLE Vehicle (
    vehicle_id SERIAL PRIMARY KEY,
    make_and_model VARCHAR(100) NOT NULL,
    year_of_manufacture INT CHECK (Year_of_Manufacture BETWEEN 1990 AND 2030),
    vehicle_type VARCHAR(50),
    load_capacity FLOAT CHECK (Load_Capacity > 0)
);

CREATE TABLE Maintenance (
    maintenance_id SERIAL PRIMARY KEY,
    vehicle_id INT REFERENCES Vehicle(vehicle_id) ON DELETE CASCADE,
    maintenance_type VARCHAR(50),
    maintenance_cost DECIMAL(10,2) CHECK (Maintenance_Cost >= 0),
    last_maintenance_date DATE,
    maintenance_required BOOLEAN,
    downtime_maintenance FLOAT CHECK (Downtime_Maintenance >= 0)
);

CREATE TABLE Trip_Info (
    trip_id SERIAL PRIMARY KEY,
    vehicle_id INT REFERENCES Vehicle(vehicle_id) ON DELETE CASCADE,
    route_info VARCHAR(50),
    weather_conditions VARCHAR(50),
    road_conditions VARCHAR(50),
    delivery_times FLOAT CHECK (delivery_times >= 0)
);
```



```sql

CREATE TABLE Sensor_Data (
    sensor_id SERIAL PRIMARY KEY,
    vehicle_id INT REFERENCES Vehicle(vehicle_id) ON DELETE CASCADE,
    usage_hours FLOAT CHECK (usage_hours >= 0),
    engine_temperature FLOAT,
    tire_pressure FLOAT,
    fuel_consumption FLOAT CHECK (fuel_consumption >= 0),
    battery_status FLOAT,
    oil_quality FLOAT,
    brake_condition VARCHAR(25),
    vibration_levels FLOAT CHECK (vibration_levels BETWEEN 0.0 AND 1.0),
    predictive_score FLOAT CHECK (predictive_score BETWEEN 0.0 AND 1.0),
    anomalies_detected BOOLEAN,
    failure_history BOOLEAN CHECK (failure_history >= 0),
    impact_on_efficiency FLOAT
);

```

```
Total rows:    Query complete 00:00:00.092
```

4. The Data Management Pipeline Process

4.1 Data Capture/Source

The database development process would begin with transforming a raw dataset comprising sensor reading, maintenance records, and route logs, typical of real-world fleet management systems. It includes data from vehicles, trips, sensors, and maintenance history, captured via automated and semi-automated sources like IoT sensors, GPS tracking, and service logging systems (Zanella et al., 2014). While currently static, our proposed design would enable integration of dynamic data streams (Hofmann and Rüsch, 2017).

4.2 Data Cleaning

A structured cleaning process would be performed in Python using pandas to ensure the data is accurate and ready for database import (McKinney, 2022).

4.2.1 Exploration and text/format standardisation

The CSV file would be imported and explored to identify missing values, inconsistent formats, and type mismatches using descriptive summaries. Text fields (i.e Vehicle_Type and Brake_Condition) would be cleaned by removing extra spaces and symbols, standardising case (e.g. title case), and unifying label formats (Huxley, 2020).

4.2.2 – Validation and missing records

Dates would be standardised, and fields like Predictive_Score validated. Incompatible entries would be removed and less critical gaps imputed using means/medians (Little and Rubin, 2019).

4.2.3 – SQL integration

Referential integrity across tables would be verified with the correct import order, following relational dependencies. The database will enforce schema constraints, report errors, and remove duplicate entries, ensuring clean, consistent data for reliable querying and analysis.

4.3 Evaluation

Our proposed database design supports the client’s goal of centralising fleet data and enabling insights. The 3NF schema separates entities into dedicated tables, reducing redundancy and improving query efficiency (Connolly & Begg, 2015).

Data cleaning, done in Python with pandas, prepares the data for import through typecasting, format standardisation, and handling of missing records. This is essential in logistics where IoT sensor and log data can be inconsistent (Rahm and Do, 2000).

Database constraints enforce referential integrity, handle missing or duplicate data and catch other integrity errors. While the original dataset is static, our design allows for scalable, real-time ingestion from telemetry and tracking platforms.

Overall, our system offers a robust, scalable foundation for our client’s operations, supporting advanced analytics, reporting, and future automation in fleet management.

References:

Connolly, T. and Begg, C. (2015) *Database systems: a practical approach to design, implementation, and management*. 6th edn. Harlow: Pearson Education. Accessed from UoE Library.

Elmasri, R. and Navathe, S.B. (2016) *Fundamentals of Database Systems*. 7th edn. Boston: Pearson.

Hofmann, E. and Rüsch, M. (2017). ‘Industry 4.0 and the current status as well as future prospects on logistics’. *Computers in Industry*, 89, pp.23-34. Doi: https://doi.org/10.1016/j.compind.2017.04.002

Holistics Software. (2025) *dbdiagram.io*. Available at: https://dbdiagram.io/home (Accessed: 6 June 2025).

Huxley, K. (2020) ‘Data Cleaning’, *SAGE Research Methods Foundations*, Available at: https://doi.org/10.4135/9781526421036842861

Kaggle (n.d.) *Logistics Vehicle Maintenance History Dataset*. Available at: https://www.kaggle.com/datasets/datasetengineer/logistics-vehicle-maintenance-history-dataset?resource=download (Accessed: 27 May 2025).

Laudon, K.C. and Laudon, J.P. (2020) *Management Information Systems: Managing the Digital Firm*. 16th edn. Harlow: Pearson.

Little, R.J.A. and Rubin, D.B. (2019) *Statistical Analysis with Missing Data*. 3rd edn. Hoboken, NJ: Wiley. Accessed from UoE Library.

McKinney, W. (2022) *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter*. 3rd edn. Sebastopol, CA: O'Reilly Media. Accessed from UoE library.

Rob, P. and Coronel, C. (2007) ‘Database Systems: Design, Implementation, and Management’. 8th edn. Boston: Cengage Learning.

Zanella, A. et al. (2014) 'Internet of Things for Smart Cities', *IEEE Internet of Things Journal*, 1(1), pp.22–32. Available at**:** 10.1109/JIOT.2014.2306328