To create a relational database that meets the business needs I first had to normalize the data

Initially all the data was in this one table in an unnormalized form.

So to make it in the first normal form had to ensure that each field has a single atomic value and remove repeating groups.

So I split the data into two tables, service and Employee, where the service table contains details about each service and the employee table lists the employees working on services, and the customer and employee names were split into forenames and surnames.

For it to be in the second normal form I had to remove partial dependencies which are fields that only depend on part of a composite primary key.

The time spent in the employee table depended on both service ID and employee id.

So I created a new table Service Employee, to link service ID and employee ID **with time spent as an attribute of the relationship.**

For it to be in the third normal form I had to remove transitive dependencies which are fields that depend on non primary key attributes.

So I split out attributes into their own tables when they were not directly dependent on the primary key.

This led to the customer table and the car table and I then updated the service table to include only foreign keys to the customer and car tables

**and I also added a new table**, Employee Availability, which was required for the 5th business process of making an employee unavailable for a period of time.

################################################################

This is my entity relationship diagram for the car service database.

The relationship between car and service is a one to many relationship as a car can have many services over time and each service must be linked to exactly one car.

It's a partial full constraint as a car may or may not have a service which is partial participation, whereas every service must be associated with a car which is full participation.

The relationship between customer and service is a one to many relationship as a customer can book many services and each service is associated with exactly one customer.

It's a partial full constraint as a customer may or may not have booked services, which is partial participation, and every service must be linked to a customer which is full participation.

The relationship between employee and service is a many to many relationship as a service can involve many employees and an employee can work on many services.

It's a partial partial participation constraint as not every employee is guaranteed to work on services and not every service requires associated employees.

The relationship between employee and employee availability is a 1 to many, as an employee can have many unavailability periods, whereas each record in the employee availability table must be associated with a employee.

It's a partial full constraint as an employee may or may not be available, which is partial, whereas every record in the employee availability table must correspond to an employee which is full participation.

################################################################

For the service table the primary key service ID is of data type variable character 10 as the service IDs consist of both letters and number characters and the longest one in the data provided was 9 characters long. This applies for all the primary keys.

Drop of date is of data type date as the data is in a DATE form

Drop off time is of data type time as the data is in a TIME form

Work required is of data type variable character max as it will consist of both letters, spaces and commas primarily and it's of max length which allows it to be descriptive and list many issues

Mileage is of data type integer as mileage is always recorded as integer numbers so it is the most appropriate data type for this column.

Then in the car table

the make is of Data type Variable Character 10 as the longest in the provided data was 10 characters long.

Model is of data type Varchar 20 as it consists of both letters and space characters, and the longest model’s name in that column characters long.

Date of manufacture Data type is date as it's the most appropriate for this column as it's the date when the car was made.

Then for the customer table

the customer forenames is variable character 10 as the longest forename in the provided data was 9 characters long and the

customer surname is a variable character 10 as the longest provided name surname in the Data is of 9 characters long.

customer email is variable character max as email addresses are generally a mix of letters, numbers and special characters and they can vary in length a lot and

customer phone number is of variable character 15 as it consists of numbers and special characters and the longest provided in the data was 15 characters long.

In the employee table,

Grade is of variable character 15 as the longest provided in the data is 15 characters long.

For 1st NF, Ensure each field has a single, atomic value and remove repeating groups.

Split the data into two tables: SERVICE and EMPLOYEE.

* **SERVICE Table:** Contains details about each service, including service\_id, dropoff\_date, work\_required, registration (for the car), and customer\_id.
* **EMPLOYEE Table:** Lists the employees working on services, with fields like service\_id, employee\_id, emp\_name, and time\_spent

Each record now represents one service or one employee assignment, with no repeating groups.

For 2nd NF, Remove partial dependencies (fields that only depend on part of a composite primary key).

* Identified that time\_spent in the EMPLOYEE table depended on both service\_id and employee\_id.
* Created a new SERVICEEMPLOYEE table to link service\_id and employee\_id, with time\_spent as an attribute of this relationship.

For 3rd NF, Remove transitive dependencies (fields that depend on non-primary key attributes).

* Split out attributes into their own tables when they were not directly dependent on the primary key:
  + **CUSTOMER Table:** Contains customer\_id, cust\_name, cust\_email, and cust\_phone.
  + **CAR Table:** Contains registration, make, model, and date\_of\_manufacture.
* Updated the SERVICE table to include only references (foreign keys) to the CUSTOMER and CAR tables.
* Added a new table, EMPLOYEEAVAILABILITY, to manage employee schedules separately.

Some Assumptions were made for the relational database to function correctly.

Services are associated with exactly one car (registration) and one customer (customer\_id).

It is assumed that a car (registration) is linked to a single customer at any given time.

An employee (employee\_id) can be assigned to multiple services, which justifies the creation of the SERVICEEMPLOYEE table to handle this many-to-many relationship.

Customer information (name, email, phone) is assumed to remain stable for a given customer ID.

when it came to populating the tables with data, I decided to make a python script to help automate this process

what I did was I put all the necessary data into a text file and each column was separated by five hashtags

then in the python script I would specify the table name and Collins of the table.

Then I have the functions to formate the date into the correct order (y, m, d)

And formatting values if they needed quotation marks or not

Then all the data was made into an sql insert statement, and then saved to an output file.

I did this for all the tables.

Explanation of the Trigger

Trigger Type: AFTER INSERT ensures the trigger fires after a new record is successfully added to the Service table.

inserted Table: The inserted table is a special, temporary table available in triggers. It contains the rows that were just inserted into the target table (Service).

Action: The INSERT statement copies the service\_id from the inserted table into ServiceEmployee with:

employee\_id defaulting to 'UNASSIGNED'.

time\_spent defaulting to NULL.

##################################################################

Existence Check: Uses an IF NOT EXISTS condition to check if a car with the provided registration already exists in the Car table.

Action: If no matching record exists, an INSERT statement adds the new car to the table.

Outcome: Prevents duplicate entries while ensuring new cars are added seamlessly.

1. *Add details of a new service booking including the car involved, drop of date and time, but without specifying the mechanic(s) who will work on it.*
2. *Get the name of any mechanics who have worked on previous services of a car involved in a new service booking, together with the number of jobs in which they are already involved on the day the car is dropped off. If the number of jobs they are already involved is zero on the day the car is dropped off, then they should not be returned in the result.*
3. *List the total time spent servicing cars per mechanic over a 24hr period between two given dates.*
4. *List the name and email address of customers who own a car where the date of next service falls between two given dates. The list should include the make and registration of the car and the date of next service.*
5. *Make a mechanic temporarily unavailable for a period of time (e.g., because of illness or vacation) and get a list of any services they may be involved in on the days they are unavailable.*

so for business process one which was add details of a new service booking including the car involved drop off date and time without specifying the mechanics who will on it

I firstly connected to the database and then specified the customer and car variables, with a new customer and car that are not in the database

when a query is run to see if the customer has a ID, if not it’s assumed this is a new customer so a new customer ID is randomly generated, then the stored procedure customer if no exist is ran, which will insert the new customer into the table, then the stored procedure, car no exist, it’s called too check if the car is in your car table if not it will insert it

this query then retrieve the latest service ID from the service table, which is then passed into python, to increase the numerical part by 1, for the new service id

finally the information is inserted into the service table, which will trigger the trigger