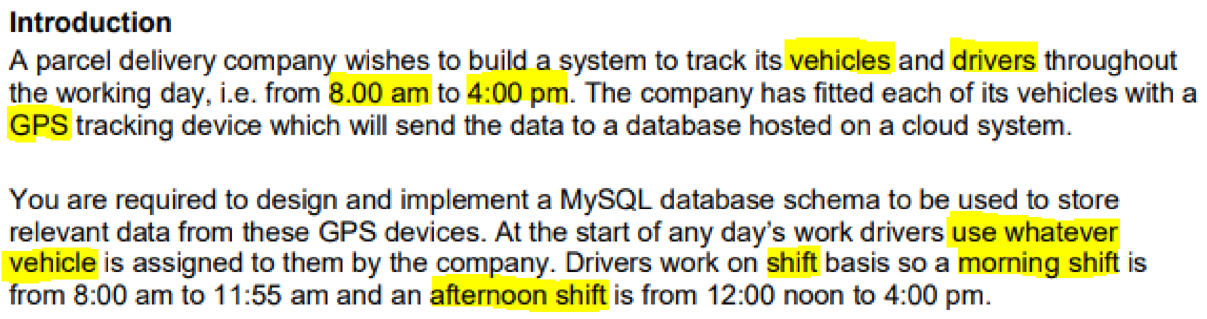
Scalable Database Systems Assignment 1

Database Design and Implementation Report

1 The Design Process

1.1 Planning

To begin my design process, I took a copy of the database design brief and highlighted all the keywords that stood out to me as being important to the database design, specifically nouns and verbs.



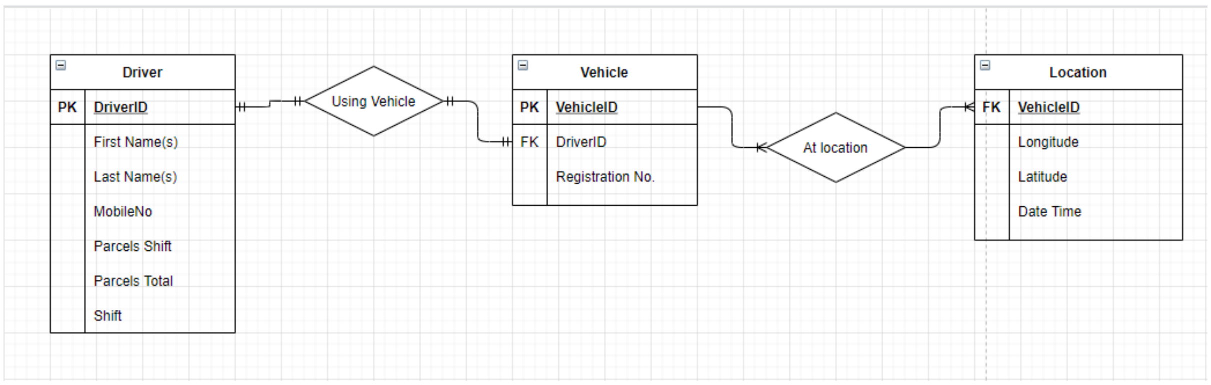
Introduction

A parcel delivery company wishes to build a system to track its vehicles and drivers throughout the working day, i.e. from 8.00 am to 4:00 pm. The company has fitted each of its vehicles with a GPS tracking device which will send the data to a database hosted on a cloud system.

You are required to design and implement a MySQL database schema to be used to store relevant data from these GPS devices. At the start of any day's work drivers use whatever vehicle is assigned to them by the company. Drivers work on shift basis so a morning shift is from 8:00 am to 11:55 am and an afternoon shift is from 12:00 noon to 4:00 pm.

This helped me to begin forming an ER diagram to aid in the design of my database. The nouns in the brief are objects that we want to specifically keep a track of, for instance the "drivers" and "vehicles". There are physical things that the database needs to hold information about. The verbs are actions that these things are doing that need to be tracked, such as "use whatever vehicle", showing that there has to be a relationship between the drivers and the vehicles. This was reflected in my ER diagram as shown:

1.2 The ER Diagram



1.2.1 Tables

Each table represents a separate object that needs to be tracked in the database, whilst each relationship shows how the objects interact with each other, loosely based around the verbs in the brief.

I normalised my database to 2NF, meaning most of the objects within the brief have been given their own table with only data that is meaningful to that particular object stored within itself. A few fields however, such as “Shift”, are part of larger tables like “Driver” instead of having their own table. This is done for the convenience of keeping queries simple and not over-complicating the database itself.

1.2.1.1 Fields

Each table stores information in separate fields. These fields have to be assigned data types in order to function properly, and to prevent users from inputting invalid data.
In the “Driver” table, fields such as “First Name”, “Last Name”, and “MobileNo” are of type `varchar(length)` allowing any alphanumeric input less than the length provided. Specifically, the names are up to 255 characters long, since there can be no guarantee as to the specific length of any person’s name or names, whilst “MobileNo” is limited to 11 characters to prevent people from inputting numbers that are too long accidentally. Other fields such as “Parcels Shift” and “DriverID” are integers, allowing only whole numbers to be entered, since you cannot deliver a fraction of a package. It is also important to keep the DriverID as an integer, since it is an auto-incrementing field, meaning it will automatically increment by one every time a new driver is added. Features such as this make my database much more future proof.

1.2.2 Relationships

This database is designed around the assumption that the data stored within is only held for a single day. After that day, data could potentially be stored in a separate database for archival purproses. The only exception to this rule is the “Parcels Total” field in the “Driver” table which stores the total amount of parcels the driver has delivered. This could be implemented as being the total for any time period, however in this instance it is there only as a placeholder.
Due to this, in my database, one driver can only be assigned to one vehicle, since that is their only vehicle assigned during a day, and one vehicle can be given many instances of GPS locations.

2 Database Implementation

2.1 Queries

Upon creation of the framework for my database, I populated my tables with data generated randomly online. This allowed me to test all the queries/procedures I implemented in my design.
I implemented all the queries mentioned in the design brief into my database, being:

ID - SQL Script	*Output*

(1)
 	
SELECT driver.*, vehicle.VehicleID, vehicle.RegistrationNo, location.Date_Time from driver
INNER JOIN vehicle ON driver.DriverID = vehicle.DriverID
INNER JOIN location ON location.VehicleID = vehicle.VehicleID WHERE location.Date_Time >= '2021-01-01 09:00:00' AND location.Date_Time < '2021-01-01 10:00:00';
	1, Ayush, Rigby, 7700900380, 3, 79, Morning, 1, MW72BAD, 2021-01-01 09:39:25
	1, Ayush, Rigby, 7700900380, 3, 79, Morning, 1, MW72BAD, 2021-01-01 09:42:01

(2) SELECT \`driver\`.* FROM \`driver\` WHERE \`driver\`.\`ParcelsShift\` > 2;	1, Ayush, Rigby, 7700900380, 3, 79, Morning 2, Antonina, Madden, 1144960587, 4, 87, Afternoon 3, Neriah, Huerta, 1144960587, 4, 100, Afternoon 5, Leja, Cottrel, 1134960747, 6, 91, Morning 7, Ruairidh, Clegg, 1134960511, 3, 78, Afternoon
(3) SELECT \`driver\`.* FROM \`driver\`;	1, Ayush, Rigby, 7700900380, 3, 79, Morning 2, Antonina, Madden, 1144960587, 4, 87, Afternoon 3, Neriah, Huerta, 1144960587, 4, 100, Afternoon 4, Alysia, Villanueva, 1414960432, 2, 77, Afternoon 5, Leja, Cottrel, 1134960747, 6, 91, Morning 6, Nimrah, Healy, 1214960968, 1, 65, Morning 7, Ruairidh, Clegg, 1134960511, 3, 78, Afternoon 8, Willis, Lester, 2079460708, 2, 61, Morning 9, Mindy, Whitehouse, 1174960951, 2, 52, Afternoon 10, Aneesa, Bradley, 2079460381, 1, 55, Morning

(4) `SELECT `driver`.* FROM `driver` WHERE `driver`.`Shift` = "Morning";`	1, Ayush, Rigby, 7700900380, 3, 79, Morning 5, Leja, Cottrel, 1134960747, 6, 91, Morning 6, Nimrah, Healy, 1214960968, 1, 65, Morning 8, Willis, Lester, 2079460708, 2, 61, Morning 10, Aneesa, Bradley, 2079460381, 1, 55, Morning

In query (1), I am able to return the location of a vehicle and it's driver between a given date and time. I take every field from the drivers table, the VehicleID and RegistrationNo from the vehicle table, and the Date_Time field from the location table and join them together on the VehicleID and the DriverID. I include a "WHERE" statement to only include the values where the Date_Time field of the location table is between the two given dates and times.
In query (2), I am simply selecting every field of the driver table where the "ParcelsShift" field is greater than 2. This number can be changed, and is easily changeable in the procedure version of this query.
Similarly, in query (3), I return the entirety of the "driver" table without any checks or statements to narrow the results.
And finally, the fourth query (4) is selecting all of the fields from "driver", but only where the shift field matches "Morning" identically. I have implemented the shift field as a `varchar()`,meaning any checks against this field have to be written as specifics. To improve this in the future, I could write the procedure version of this query to have the user enter something different, such as a "1" or "0", corresponding to being "Morning" or "Afternoon", and have the procedure decide which shift to lookup. This, whilst being less easily human readable, would help eliminate some of the human error that could come from the incorrect typing of the shifts.

2.2 Procedures

All of the queries I have created I also have procedures for. Procedures help to neaten up the queries that would be most often used in my database, as well as give a nicer user interface for inputting specific search terms. For instance, my procedure for getting a driver and their vehicle's location at a specific time allows the user to enter a Date-Time range between which to search.

```
DELIMITER //


CREATE PROCEDURE findDriverAtTime ( IN timeOne DateTime, IN timeTwo DateTime )

BEGIN

       SELECT driver.*, vehicle.VehicleID, vehicle.RegistrationNo,
location.Date_Time from driver

INNER JOIN vehicle ON driver.DriverID = vehicle.DriverID
INNER JOIN location ON location.VehicleID = vehicle.VehicleID WHERE
location.Date_Time >= timeOne AND location.Date_Time < timeTwo;

END//


DELIMITER ;



CALL findDriverAtTime('2021-01-01 09:00:00', '2021-01-01 10:00:00');
```

As can be seen from the call function, the user is able to request data very accurately from within custom time periods, and have all relevant information presented to them.

Alongside all of the queries in procedure form, I also created an extra procedure that I thought would be helpful to a potential database user. "`dropOffCount(int DriverID)`" is used to get the number of parcels delivered by any driver during that day, equating to the number of stops they've made during that day too.

```
DELIMITER //


CREATE PROCEDURE dropOffCount( IN driverIDNum int )

BEGIN

       SELECT COUNT(*) from driver INNER JOIN vehicle ON driver.DriverID =
vehicle.DriverID INNER JOIN location ON location.VehicleID = vehicle.VehicleID
WHERE driver.DriverID = driverIDNum;

END//


DELIMITER ;


CALL dropOffCount(1);
```

2.3 Security

I also took database security into consideration when creating my database. By default, MySQL doesn't include any password protection on the user accounts used to access the database, and as such leaves my database open to malicious invasion. I created a new user named "databaseUser" with a password that was only allowed permissions to edit the "delivery" database.

```
FLUSH PRIVILEGES;

DROP USER 'databaseUser'@'localhost';

CREATE USER 'databaseUser'@'localhost' IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON `delivery`.* TO 'databaseUser'@'localhost';

FLUSH PRIVILEGES;
```

Whilst this user is not very secure as setup here, it could easily be altered on a commercial database to have a more descriptive name, and secure password. Currently, my “root” user is also not password protected, and in a commercial environment it would be vital to ensure it was.