# CS27020 Summer Resit assignment
# Electric Vehicle Database

Released: 10/07/23

Due: 11/08/23

# Table of contents

# Part 1 - UNF

There are many different elements I would include in a database about Electric Vehicles. To create a relation in UNF (unnormalised form) I would use the following attributes:

**JourneyID :**

Int

This is a uniquely generated number that identifies each journey. Because this is a unique number, it works as a primary key. Theoretically you could have two drivers with the same name, driving on the same date, with the same birthday, at the same time in the same model of car and the only thing to differentiate them would be their driver's licence and car registration. By having a JourneyID we have a reliable unique way to identify each journey that is only one attribute long.

**FirstName :**

VarChar[32]

This is the first name of the driver and does not need to be unique. There are probably a lot of drivers with the same first name. It is a type varchar[32] as most of the longest English names are only around 11 characters letters long[1], and assuming this is a UK database and does not accept non-Latin characters that should be enough.

**LastName:**

VarChar[32]

This is the last name of the driver and also does not need to be unique. There are probably a lot of drivers with the same surname. The longest single English surname is 17 letters (Featherstonehaugh)[2], however other nationalities do have longer surnames so it makes sense to me to extend it to 32 characters. This also accounts for double-barrelled names.

**DoB:**

DATE

This is the date of birth of the driver. Assuming this a relation defined in SQL, I can use the DATE type for the date of birth attribute. If it's in SQL I can also set a maximum date which should be 11-08-2007, as anyone who was born after that will not be sixteen when this assignment is due in, and therefore could not drive a car. I could also set the minimum date as 11-04-1907 as that is the birthdate of the current oldest living person[3] and no one will be older than that.

**Address:**

VarChar[150]

This is the address the driver is registered to. This needs to be a very long list of characters as some addresses are ridiculously long. For example, what if someone lived in Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch in Wales? They would need 58 characters just for the name of their village. They would then need the rest of the address too. Therefore as a catch-all I recommend 150 characters to be safe.

**Postcode:**

VarChar[8]

All UK postcodes are two to four characters, a space, and then three more characters.[4]

**LicenceNumber:**

Char[16]

This is the driver's unique licence number from their driving licence. It's always 16 characters long. [5]

**Distance:**

Int

The distance driven on that journey. The length of the UK (from Land's End to John O'Groats[6]) is 847 miles. Doing that twice in one trip would be 1694 miles. Realistically, not any people are going to be travelling more than that in one trip, as you can get pretty much anywhere in the Uk and back again within that distance. Even so the maximum distance this attribute can hold is 9999 miles, which should be enough for any car journey in the UK.

**Date:**

DATE

Assuming this is made in SQL there is an in-built date type. The date is the date of the journey. The maximum should be 11-08-2023 as journeys from the future should not be able to be added. The minimum date should be set as 01-01-1801 as the date of the first ever electric car is disputed and some consider it to be 1828, 1834, 1836, 1859, 1881, 1890 etc... It depends on how you define electric car[7]. What is certain is that there were definitely no electric cars prior to 1801 which is when the first road-steam-powered vehicle was built[8].

**Time:**

TIME

Assuming this is made in SQL then time is also a built in data-type. This attribute refers to the start time of the journey and is necessary because it is possible for one driver to do the same journey in the same car on the same day- however it is not possible for the driver to do two journeys at once.

**Make:**

VarChar[50]

I chose a character list of fifty for this attribute to allow for longer company names.

**Model :**

VarChar[50]

Most car models are a single word and a digit, however some are longer.

**Battery name:**

VarChar[50]

Car battery names tend to be long and complicated.[9]

**Battery dimension :**

List of Int

Because this table is unnormalized, the attributes do not need to be atomic, and can be a list of integers instead.

**Battery voltage :**

Int

The voltage of the battery. Most electric cars run on a voltage between 400V and 800V.[9]

**Registration :**

VarChar[10]

The number plate format was standardised in 2001[10], and plates from before then may not comply with the current format. Additionally, some numberplates are completely custom and do not comply with any format. As a result it's best to store the registration as ten characters.

**Theoretical range :**

Int

The theoretical range of the car in miles.

**Battery percentage before journey :**

Int

All percentages are out of 100, so this will need constraints.

**Battery percentage after journey :**

Int

All percentages are out of 100, so this will need constraints.

**Number of stops:**

Int

This is the number of stops that were made to recharge the car. This is an additional attribute that I added, as the data about battery percentage before and after the journey would be useless if the car was recharged halfway through.

**Percentage charged:**

Int

This is the total percentage that the car was recharged at all stops. For example, if they stopped twice and charged the car 10% and then 30% this number would be 40%. If the number of stops is 0 then this number will also be 0. If this number is over 100% then the number of stops must be at least 2.

# Part 2 – Functional dependencies and candidate keys

**Functional dependencies**

Address -> Postcode

LicenseNumber -> {FirstName, LastName, DoB, Address}

Therefore

LicenseNumber -> {FirstName, LastName, DoB, Address, Postcode}

LicenseNumber is not a candidate key.

Model -> {Make, BatteryName, TheoritcalRange}

BatteryName-> {BatteryDimensions, BatteryVoltage}

Therefore

Model-> {Make, BatteryName, BatteryDimensions, BatteryVoltage,TheorticalRange}

Registration -> Model

Therefore

Registration -> {Model, Make, BatteryName, BatteryDimensions, BatteryVoltage}

JourneyID -> {Distance, Date, Time, NumberOfStops, PercentageCharged, BatteryPercentageBeforeJourney, BatteryPercentageAfterJourney,}

JourneyID -> {LicenseNumber, Registration}

Therefore

JourneyID -> {FirstName, LastName, DoB, Address, Postcode, LicenseNumber, Distance, Date, Make, Model, BatteryName, BatteryDimensions, BatteryVoltage, Registration, TheoreticalRange, BatteryPercentageBeforeJourney, BatteryPercentageAfterJourney, NumberOfStops, PercentageCharged}

Therefore JourneyID is a candidate key.

**Candidate keys**

To identify the candidate keys, I first identified the trivial Superkey with all of the attributes.

**Superkey: {**JourneyID, FirstName, LastName, DoB, Address, Postcode, LicenseNumber, Distance, Date, Time, Make, Model, BatteryName, BatteryDimensions, BatteryVoltage, Registration, TheoreticalRange, BatteryPercentageBeforeJourney, BatteryPercentageAfterJourney, NumberOfStops, PercentageCharged}

The candidate keys are:

{JourneyID}

{Date, Time, LicenseNumber, Registration}

# Part 3- UNF > 3NF

**UNF > 1NF**

To take the table from UNF to 1NF we need to make it atomic. The only non-atomic value in this table is BatteryDimsensions. BatteryDimensions is actually a list of three values, and needs to be separated out into width, height and length. Our new attributes are therefore: {JourneyID, FirstName, LastName, DoB, Address, Postcode, LicenseNumber, Distance, Date, Make, Model, BatteryName, BatteryLength, BatteryHeight, BatteryWidth, BatteryVoltage, Registration, TheoreticalRange, BatteryPercentageBeforeJourney, BatteryPercentageAfterJourney, NumberOfStops, PercentageCharged}.

The other attribute that could be non-atomic is the address, however as I have stored it as a single string it is atomic in this table. If it was stored as a series of strings then it would need to be broken down into house number, street, city etc. Or converted into one big string as I have done. The main reason I chose to keep it as one big string is that this table already has a lot of attributes, and I didn't want to make it longer than necessary. Also, if I ended up putting the addresses into their own table, there is no obvious attribute to use as a foreign key.

**1NF > 2NF**

There are a lot of non-prime values that are dependent on parts of composite primary keys in this table. I have to split it into multiple tables.

People

| License Number | First Name | Last Name | Dob | Address | Postcode |
|---|---|---|---|---|---|
| | | | | | |

Car

| Registration | Model | Make | Battery Name | Battery Height | Battery Width | BatteryLength | Battery Voltage | TheoreticalRange |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

Journeys

| JourneyID | Distance | Date | Time | NumberOfStops | PercentageCharged | BatteryPercentageBeforeJourney | BatteryPercentageAfterJourney | LicenseNumber | Registration |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

Journeys.LicenseNumber and Journeys.Registration are foreign keys.

These tables that I have created are not all in 2NF themselves, so I need to make two more.

Addresses

| Address | Postcode |
|---|---|
| | |

Batteries

| BatteryName | BatteryHeight | BatteryWidth | BatteryLength | BatteryVoltage |
|---|---|---|---|---|
| | | | | |

So now I have

People

| License Number | First Name | Last Name | Dob | Address |
|---|---|---|---|---|
| | | | | |

With Address being a foreign key.

Car

| Registration | Model | Make | Battery Name | TheoreticalRange |
|---|---|---|---|---|
| | | | | |

With BatteryName being a foreign key.

Journeys

| JourneyID | Distance | Date | Time | NumberOfStops | PercentageCharged | BatteryPercentageBeforeJourney | BatteryPercentageAfterJourney | License Number | Registration |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

**2NF > 3NF**

To put my tables into 3NF I need to get rid of transitive dependencies. Distance, NumberOfStops, PercentageCharged, BatteryPercentageBeforeJourney and BatteryPercentageAfterJourney are all dependent on {JourneyID}, but not on {Date, Time, License Number, Registration}.

So I create a new table:

JourneyInfo

| JourneyID | Distance | NumberOfStops | Percentage charged | BatteryPercentageBeforeJourney | BatteryPercentageAfterJourney |
|---|---|---|---|---|---|
| | | | | | |

And the other one becomes:

Journey

| JourneyID | Date | Time | LicenseNumber | Registration |
|---|---|---|---|---|
| | | | | |

With JourneyID being a foreign key.

# Part 4- ER diagram

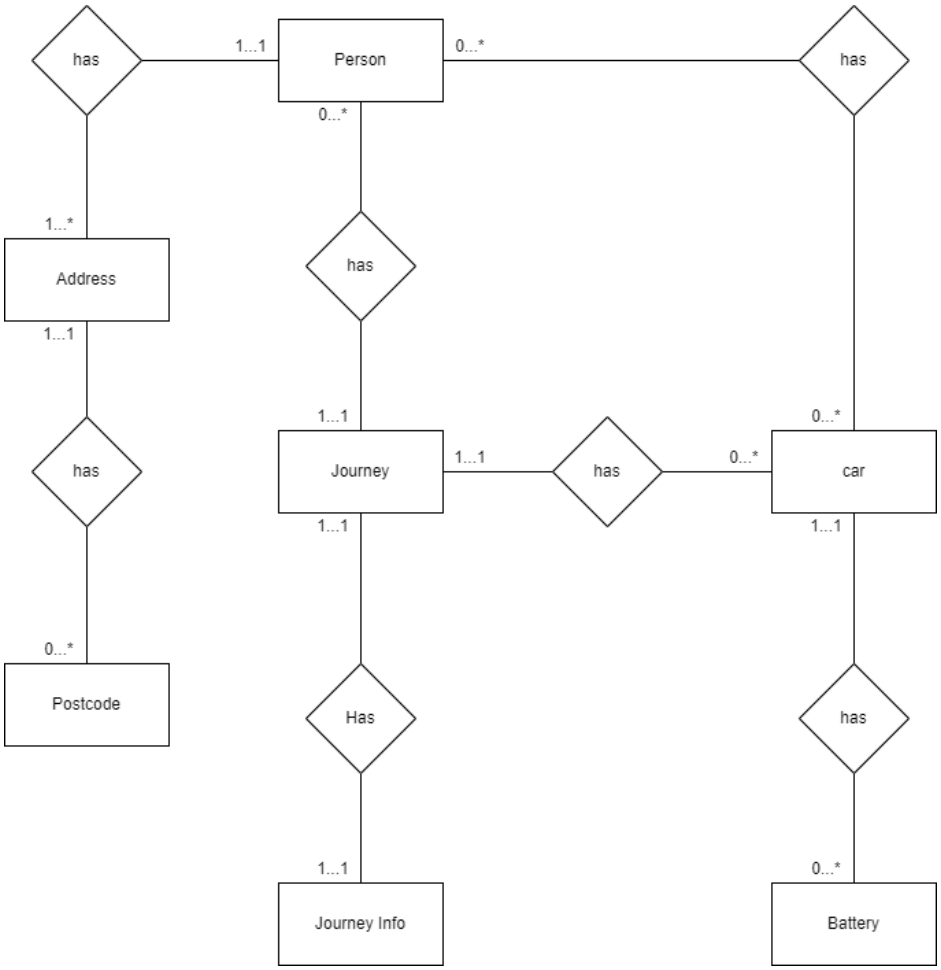


Figure 1.

Every journey must have exactly one person, car and journey information.

A person can be in many journeys and can have many cars. A person must have one address- this is the address on their driver's licence. A person cannot have multiple addresses as they are legally registered to one address.

Every address has a postcode. One postcode can refer to many addresses, or there may be no addresses in the database that are in that postcode.

A car may be used in many journeys, or none at all. If a car is used in a journey then it must have a driver, however if the car is not used in a journey then it does not have to have a driver. A car must have exactly one battery. The same kind of battery may be used in many cars, or in none at all.

# Part 5- PostgrSQL database

### Building the database:

I created the following .pgsql file in VScode, and then ran it on my local machine under a database named cs27020.

```
CREATE TABLE Addresses(
    address VARCHAR(150) UNIQUE,
    postcode VARCHAR(8)

);

CREATE TABLE Batteries(
    battery_name VARCHAR(50) PRIMARY KEY,
    battery_height INTEGER,
    battery_width INTEGER,
    battery_length INTEGER,
    battery_voltage INTEGER

);

CREATE TABLE People(
    license_number CHAR(16) PRIMARY KEY CHECK (license_number ~ '^[0-9]+$'),
    first_name VARCHAR(32),
    surname VARCHAR(32),
    DOB DATE CHECK(DOB >= '1907-04-11' AND DOB<='2007-08-11'),
    address VARCHAR(150) UNIQUE,
    FOREIGN KEY(address) REFERENCES Addresses(address)
);

CREATE TABLE Cars(
    registration VARCHAR(10) PRIMARY KEY,
    model VARCHAR(50),
    make VARCHAR(50),
    battery VARCHAR(50),
    FOREIGN KEY(battery) REFERENCES Batteries(battery_name),
    theoretical_range INTEGER
);

CREATE TABLE JourneyData(
    id SERIAL PRIMARY KEY,
    distance INTEGER,
    number_of_stops INTEGER,
    percentage_charged INTEGER CHECK(percentage_charged >= 0 AND percentage_charged <= 100),
    battery_percentage_before_journey INTEGER CHECK(percentage_charged >= 0 AND
percentage_charged <= 100),
    battery_percentage_after_journey INTEGER CHECK(percentage_charged >= 0 AND
percentage_charged <= 100)


);

CREATE TABLE Journeys(
    Journey INTEGER,
    date DATE CHECK(date >= '1801-01-01' AND date <= '2023-08-11'),
    time TIME,
    license_number CHAR(16),
    registration VARCHAR(10),
    FOREIGN KEY(Journey) REFERENCES JourneyData(id),
    FOREIGN KEY(license_number) REFERENCES People(license_number),
    FOREIGN KEY(registration) REFERENCES Cars(registration)
);
```

I got the following output:

```
null rows created
null rows created
null rows created
null rows created
null rows created
null rows created
```

**Populating the database**

Next I created a quick Python program to generate fictitious data for me. I created some csv files using Excel and Power Query to get data from the web. First, I got the most common names[11-12] and surnames[13]. Then I got a list a of words that could be added together to make fake road names[14]. I also got a list of recent/upcoming car models with their makes[15]- although these are not necessarily EV cars, this data is fictitious, and I just wanted something that would sound realistic. I had a quick look at the Halfords car battery page[9] for inspiration on battery names, but I generated them myself.

The first python script generated addresses.

```
 1. import csv
 2. import random
 3.
 4. thingspath = 'things.csv'
 5. citiespath = 'cities.csv'
 6. letters = 'abcdefghijklmnopqrstuvwxyz'
 7.
 8. cities = []
 9. things = []
10. with open(thingspath, 'r') as csv_file:
11.     csv_reader = csv.reader(csv_file)
12.     for row in csv_reader:
13.         things.append(row[0])
14.
15. with open(citiespath, 'r') as csv_file:
16.     csv_reader2 = csv.reader(csv_file)
17.     for row in csv_reader2:
18.         cities.append(row[0])
19.
20.
21. filename = 'addresses.txt'
22. adresses = []
23.
24. with open(filename,'w') as file:
25.     for i in range(20):
26.         num = random.randint(1,200)
27.         thing1 = random.choice(things)
28.         thing2 = random.choice(things)
29.         city = random.choice(cities)
30.         add = str(num) +' ' + thing1 + thing2 + ' road ' + city
31.         n = random.randint(2,4)
32.         postcode = ''
33.         for j in range(n):
34.             if random.randint(0,1) == 0:
35.                 postcode += str(random.randint(0,9))
36.             else:
37.                 postcode += str(random.choice(letters)).upper()
38.         postcode += ' '
```

```
        for k in range(3):
            if random.randint(0,1) == 0:
                postcode += str(random.randint(0,9))
            else:
                postcode += str(random.choice(letters)).upper()
        file.write("INSERT INTO Addresses (address, postcode) VALUES ('" + add + "','" +
postcode + "');\n")
        adresses.append(add)

file2 = "addresses.csv"

with open(file2, mode='w', newline='') as file:
    writer = csv.writer(file)
    for item in adresses:
        writer.writerow([item])

```

The next one generated people.

```
import csv
import random

filepath = 'names_only.csv'


boys_names = []
girls_names = []
surnames = []
adresses = []
with open(filepath, 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        throw, boy, throw2, girl, surname = row
        boys_names.append(boy)
        girls_names.append(girl)
        surnames.append(surname)

filepath = 'addresses.csv'
with open(filepath, 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        adresses.append(row[0])

newfile = 'people.txt'
with open(newfile, 'w') as file:
    for i in range(10):
        license = ''
        for j in range(16):
            license += str(random.randint(0,9))

        if random.randint(0,1) == 0:
            name = boys_names[random.randint(1,99)]
        else:
            name = girls_names[random.randint(1,99)]
        name2 = surnames[random.randint(0,99)].lower()
        dob = str(random.randint(1908,2006)) + '-' + str(random.randint(1,12)) + '-' +
str(random.randint(1,28))
        add = random.choice(adresses)
        file.write("INSERT INTO People (license_number, first_name, surname, DOB, address)
VALUES ('" + license + "', '" + name + "', '" + name2 + "', '" +dob +"', '"+add+"');\n")

```

Then I generated my batteries.

```
import random
import csv
companies = ['Halfords', 'Yuasa', 'Citroen', 'Volvo', 'Amazon', 'Tesla']
letters = 'abcdefghijklmnopqrstuvwxyz'
adj = ['platinum', 'silver', 'gold', 'black', 'ultra', 'neon', 'extra']
volts = [6,12,24]

batteries = []

filename = 'batteries.txt'
with open(filename,'w') as file:
    for i in range(10):
        comp = random.choice(companies)
        code = ''
        for j in range(3):
            code += str(random.choice(letters).upper())
        for k in range(3):
            code += str(random.randint(0,9))
        bname = comp + ' ' + code + ' ' + str(random.choice(adj).upper()) + ' Car Battery'
        h = str(random.randint(20,60))
        w = str(random.randint(20,60))
        l = str(random.randint(20,60))
        v = str(random.choice(volts))
        file.write("INSERT INTO Batteries(battery_name, battery_height, battery_width,
battery_length, battery_voltage) VALUES('" + bname + "', '" + h + "', '" + w + "', '" + l + "',
'" + v +"');\n" )
        batteries.append(bname)

file2 = "batteries.csv"

with open(file2, mode='w', newline='') as file:
    writer = csv.writer(file)
    for item in batteries:
        writer.writerow([item])

```

Then I created the journey data. This was when I realised that I didn't actually need to export the text as a file I could copy paste from, as I can copy paste straight from the console.

```
import random

for i in range(10):
    d = random.randint(10,600)
    s = random.randint(0,5)
    pc = random.randint(0,100)
    pb = random.randint(0,100)
    pa = random.randint(0,100)
    print("INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('" + str(d) + "',
'" + str(s) + "', '" + str(pc) + "', '" + str(pb) + "', '" + str(pa) + "');")

```

I used all of this to create the following .psql file :

```
INSERT INTO Addresses (address, postcode) VALUES ('190 TidePetrified road Worcester','211L
V7X');
INSERT INTO Addresses (address, postcode) VALUES ('175 ReefGems road Peterborough','J6S0
Z6T');
INSERT INTO Addresses (address, postcode) VALUES ('145 RiverSoil road Wolverhampton','IB
71E');
INSERT INTO Addresses (address, postcode) VALUES ('32 VolcanoeVine road Chichester','5BBZ
G0B');
INSERT INTO Addresses (address, postcode) VALUES ('132 OceansRemain road Milton
Keynes','4630 SJB');
```

```
 6. INSERT INTO Addresses (address, postcode) VALUES ('150 ViruseDarkness road Preston','57IX
A01');
 7. INSERT INTO Addresses (address, postcode) VALUES ('24 SeaBeache road Nottingham','RH5I
3MW');
 8. INSERT INTO Addresses (address, postcode) VALUES ('138 BaysBush road Peterborough','O2R
K24');
 9. INSERT INTO Addresses (address, postcode) VALUES ('179 RootIce road Truro','80V WAQ');
10. INSERT INTO Addresses (address, postcode) VALUES ('145 StormHerb road Manchester','SZBD
A25');
11. INSERT INTO Addresses (address, postcode) VALUES ('60 PetrifiedEgg road Derby','9I 67H');
12. INSERT INTO Addresses (address, postcode) VALUES ('115 SoilMammal road Portsmouth','KD33
782');
13. INSERT INTO Addresses (address, postcode) VALUES ('32 WildfireInsect road York','PYW2 CPY');
14. INSERT INTO Addresses (address, postcode) VALUES ('186 MountainMoss road Cambridge','OC
UFB');
15. INSERT INTO Addresses (address, postcode) VALUES ('38 ViruseRainforest road Wakefield','VB6
5T4');
16. INSERT INTO Addresses (address, postcode) VALUES ('123 RainGems road Birmingham','03L SF8');
17. INSERT INTO Addresses (address, postcode) VALUES ('117 AuroraDust road Ely','W4K 7I0');
18. INSERT INTO Addresses (address, postcode) VALUES ('174 AnimalVegetable road Sunderland','T9
39F');
19. INSERT INTO Addresses (address, postcode) VALUES ('138 CarnivoreVegetable road Carlisle','1M
SN3');
20. INSERT INTO Addresses (address, postcode) VALUES ('184 HerbGrassland road Milton Keynes','R9
2R5');
21.
22.
23. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Yuasa WEV042 NEON Car Battery', '26', '53', '21', '6');
24. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Citroen ZDO820 BLACK Car Battery', '46', '35', '30', '6');
25. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Yuasa TWP822 GOLD Car Battery', '32', '33', '27', '6');
26. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Tesla QAU517 SILVER Car Battery', '48', '26', '46', '12');
27. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Halfords CJU109 NEON Car Battery', '50', '38', '34', '6');
28. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Halfords KCZ784 ULTRA Car Battery', '57', '42', '31', '6');
29. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Volvo IYC732 GOLD Car Battery', '53', '34', '27', '12');
30. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Halfords JAY718 SILVER Car Battery', '29', '49', '45', '6');
31. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Volvo UTA833 EXTRA Car Battery', '60', '24', '50', '24');
32. INSERT INTO Batteries(battery_name, battery_height, battery_width, battery_length,
battery_voltage) VALUES('Volvo MWB564 BLACK Car Battery', '35', '53', '26', '12');
33.
34. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('3060864955198754', ' Aria', 'gray', '1965-12-20', '60 PetrifiedEgg road Derby');
35. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('1274518666150693', ' Ellie', 'jenkins', '1980-12-1', '60 PetrifiedEgg road Derby');
36. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('6842535293489716', ' Max', 'adams', '1925-2-15', '145 StormHerb road Manchester');
37. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('2166507962655243', ' Luca', 'campbell', '1935-7-13', '186 MountainMoss road Cambridge');
38. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('3528000155923387', ' Liam', 'hill', '1953-5-24', '24 SeaBeache road Nottingham');
39. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('3161053397674525', ' Emma', 'harris', '1968-8-21', '38 ViruseRainforest road Wakefield');
40. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('4314693819423859', ' Poppy', 'young', '1978-10-10', '184 HerbGrassland road Milton Keynes');
41. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('7018522436054916', ' Isaac', 'hughes', '1943-8-27', '38 ViruseRainforest road Wakefield');
42. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('4256015482204389', ' Darcie', 'harris', '1946-10-12', '132 OceansRemain road Milton Keynes');
43. INSERT INTO People (license_number, first_name, surname, DOB, address) VALUES
('9071955581711710', 'Olivia', 'dixon', '2003-6-20', '174 AnimalVegetable road Sunderland');
44.
45.
```

```
46. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('AN25
VRE','i4','BMW','Halfords JAY718 SILVER Car Battery','261');
47. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('JB54
RLM','1500 Quad Cab','Ram','Yuasa WEV042 NEON Car Battery','478');
48. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('DP62
DAJ','Ridgeline','Honda','Tesla QAU517 SILVER Car Battery','261');
49. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('PY13
QQI','Maxima','Nissan','Volvo UTA833 EXTRA Car Battery','290');
50. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('GH67
REX','Camaro','Chevrolet','Yuasa WEV042 NEON Car Battery','216');
51. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('KK27
KXM','ID.Buzz','Volkswagen','Citroen ZDO820 BLACK Car Battery','458');
52. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('BE52
SKB','Envista','Buick','Tesla QAU517 SILVER Car Battery','184');
53. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('WP38
TEF','296 GTS','Ferrari','Halfords KCZ784 ULTRA Car Battery','393');
54. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('TJ12
HLL','Sierra 2500 HD Regular Cab','GMC','Volvo MWB564 BLACK Car Battery','136');
55. INSERT INTO Cars (registration, model, make, battery, theoretical_range) VALUES ('DD51
GJO','Navigator L','Lincoln','Tesla QAU517 SILVER Car Battery','334');
56.
57. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('398', '4', '41',
'69', '1');
58. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('137', '4', '50',
'70', '19');
59. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('358', '5', '43',
'25', '26');
60. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('560', '0', '98',
'19', '97');
61. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('443', '5', '49',
'59', '16');
62. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('557', '1', '68',
'54', '77');
63. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('10', '2', '70',
'58', '56');
64. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('442', '4', '79',
'73', '76');
65. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('482', '5', '41',
'78', '24');
66. INSERT INTO JourneyData (distance, number_of_stops, percentage_charged,
battery_percentage_before_journey, battery_percentage_after_journey) VALUES ('565', '1', '41',
'53', '26');
67.
```

I then went into the shell and used the SELECT function to get the license numbers and car registrations. With this data I created one final Python script:

```
 1. import random
 2.
 3. license_nums = ['3060864955198754',
 4.     '1274518666150693',
 5.     '6842535293489716',
 6.     '2166507962655243',
 7.     '3528000155923387',
 8.     '3161053397674525',
 9.     '4314693819423859',
10.     '7018522436054916',
11.     '4256015482204389',
```

```
12.     '9071955581711710',]
13.
14. regs = [
15.     'AN25 VRE',
16.     'JB54 RLM',
17.     'DP62 DAJ',
18.     'PY13 QQI',
19.     'GH67 REX',
20.     'KK27 KXM',
21.     'BE52 SKB',
22.     'WP38 TEF',
23.     'TJ12 HLL',
24.     'DD51 GJO',
25.
26. ]
27.
28.
29. for i in range (10):
30.     id = str(i+1)
31.     date = str(random.randint(2007,2022)) + '-' + str(random.randint(1,12)) + '-' +
str(random.randint(1,28))
32.     time = str(random.randint(00,24)) + ':' + str(random.randint(0,60)) + ':' +
str(random.randint(0,60))
33.     lic = str(license_nums[i])
34.     reg = str(regs[i-1])
35.     all = "'" + id + "', '" + date + "', '" + time +  "', '" + lic + "', '" + reg +"');"
36.     print("INSERT INTO Journeys (Journey, date, time, license_number, registration) VALUES
(" + all)
37.
38.
39.
```

These Python scripts are not perfect. They do not check to see if there is more than one person with the same name and date of birth for example, but I manually checked all the data for integrity before I ran my queries.

**Output**

Running the DESCRIBE TABLE (note: I used the shorthand "\d" for this) command for each of my tables in the database outputs the following:

```
cs27020=# \d addresses;
                    Table "public.addresses"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 address  | character varying(150) |           |          |
 postcode | character varying(8)   |           |          |
Indexes:
    "addresses_address_key" UNIQUE CONSTRAINT, btree (address)
Referenced by:
    TABLE "people" CONSTRAINT "people_address_fkey" FOREIGN KEY (address) REFERENCES
addresses(address)


cs27020=# \d batteries;
                       Table "public.batteries"
     Column      |         Type          | Collation | Nullable | Default
-----------------+-----------------------+-----------+----------+---------
 battery_name    | character varying(50) |           | not null |
 battery_height  | integer               |           |          |
 battery_width   | integer               |           |          |
 battery_length  | integer               |           |          |
 battery_voltage | integer               |           |          |
Indexes:
    "batteries_pkey" PRIMARY KEY, btree (battery_name)
Referenced by:
```

```
    TABLE "cars" CONSTRAINT "cars_battery_fkey" FOREIGN KEY (battery) REFERENCES
batteries(battery_name)


cs27020=# \d cars
                          Table "public.cars"
      Column       |         Type          | Collation | Nullable | Default
-------------------+-----------------------+-----------+----------+---------
 registration      | character varying(10) |           | not null |
 model             | character varying(50) |           |          |
 make              | character varying(50) |           |          |
 battery           | character varying(50) |           |          |
 theoretical_range | integer               |           |          |
Indexes:
    "cars_pkey" PRIMARY KEY, btree (registration)
Foreign-key constraints:
    "cars_battery_fkey" FOREIGN KEY (battery) REFERENCES batteries(battery_name)
Referenced by:
    TABLE "journeys" CONSTRAINT "journeys_registration_fkey" FOREIGN KEY (registration)
REFERENCES cars(registration)


cs27020=# \d people
                        Table "public.people"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 license_number | character(16)          |           | not null |
 first_name     | character varying(32)  |           |          |
 surname        | character varying(32)  |           |          |
 dob            | date                   |           |          |
 address        | character varying(150) |           |          |
Indexes:
    "people_pkey" PRIMARY KEY, btree (license_number)
Check constraints:
    "people_dob_check" CHECK (dob >= '1907-04-11'::date AND dob <= '2007-08-11'::date)
    "people_license_number_check" CHECK (license_number ~ '^[0-9]+$'::text)
Foreign-key constraints:
    "people_address_fkey" FOREIGN KEY (address) REFERENCES addresses(address)
Referenced by:
    TABLE "journeys" CONSTRAINT "journeys_license_number_fkey" FOREIGN KEY (license_number)
REFERENCES people(license_number)


cs27020=# \d journeydata
                                      Table "public.journeydata"
              Column               |  Type   | Collation | Nullable |                 Default
-----------------------------------+---------+-----------+----------+--------------------------
--------------
 id                                | integer |           | not null |
nextval('journeydata_id_seq'::regclass)
 distance                          | integer |           |          |
 number_of_stops                   | integer |           |          |
 percentage_charged                | integer |           |          |
 battery_percentage_before_journey | integer |           |          |
 battery_percentage_after_journey  | integer |           |          |
Indexes:
    "journeydata_pkey" PRIMARY KEY, btree (id)
Check constraints:
    "journeydata_percentage_charged_check" CHECK (percentage_charged >= 0 AND percentage_charged
<= 100)
    "journeydata_percentage_charged_check1" CHECK (percentage_charged >= 0 AND
percentage_charged <= 100)
    "journeydata_percentage_charged_check2" CHECK (percentage_charged >= 0 AND
percentage_charged <= 100)
Referenced by:
    TABLE "journeys" CONSTRAINT "journeys_journey_fkey" FOREIGN KEY (journey) REFERENCES
journeydata(id)


cs27020=# \d journeys
```

```
                        Table "public.journeys"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 journey        | integer                |           |          |
 date           | date                   |           |          |
 time           | time without time zone |           |          |
 license_number | character(16)          |           |          |
 registration   | character varying(10)  |           |          |
Check constraints:
    "journeys_date_check" CHECK (date >= '1801-01-01'::date AND date <= '2023-08-11'::date)
Foreign-key constraints:
    "journeys_journey_fkey" FOREIGN KEY (journey) REFERENCES journeydata(id)
    "journeys_license_number_fkey" FOREIGN KEY (license_number) REFERENCES
people(license_number)
    "journeys_registration_fkey" FOREIGN KEY (registration) REFERENCES cars(registration)
```

# Part 6- SQL Queries

## i. Query for battery capacity and range for all vehicles.

Query:

```
 1. SELECT
 2.     c.registration,
 3.     c.model,
 4.     c.make,
 5.     b.battery_name,
 6.     b.battery_voltage AS battery_capacity,
 7.     c.theoretical_range
 8. FROM
 9.     Cars c
10. JOIN
11.     Batteries b ON c.battery = b.battery_name;
12.
13.
```

Output:

10 rows returned

| | registration character varying | model character varying | make character varying | battery_name character varying | battery_capacity integer | theoretical_range integer |
|---|---|---|---|---|---|---|
| 1 | GH67 REX | Camaro | Chevrolet | Yuasa WEV042 NEON Car Battery | 6 | 216 |
| 2 | JB54 RLM | 1500 Quad Cab | Ram | Yuasa WEV042 NEON Car Battery | 6 | 478 |
| 3 | KK27 KXM | ID.Buzz | Volkswagen | Citroen ZDO820 BLACK Car Battery | 6 | 458 |
| 4 | DD51 GJO | Navigator L | Lincoln | Tesla QAU517 SILVER Car Battery | 12 | 334 |
| 5 | BE52 SKB | Envista | Buick | Tesla QAU517 SILVER Car Battery | 12 | 184 |
| 6 | DP62 DAJ | Ridgeline | Honda | Tesla QAU517 SILVER Car Battery | 12 | 261 |
| 7 | WP38 TEF | 296 GTS | Ferrari | Halfords KCZ784 ULTRA Car Battery | 6 | 393 |
| 8 | AN25 VRE | i4 | BMW | Halfords JAY718 SILVER Car Battery | 6 | 261 |
| 9 | PY13 QQI | Maxima | Nissan | Volvo UTA833 EXTRA Car Battery | 24 | 290 |
| 10 | TJ12 HLL | Sierra 2500 HD Regular Cab | GMC | Volvo MWB564 BLACK Car Battery | 12 | 136 |

This is an image because I ran it in VScode, and when I try and copy paste the output it just gives me the following:

```
    Batteries b ON c.battery = b.battery_name;
```

### ii. Calculating the average journey distance for all cars

Query:

```
 1. WITH JourneyCharge AS (
 2.     SELECT
 3.         jd.id,
 4.         jd.distance,
 5.         jd.battery_percentage_before_journey,
 6.         jd.battery_percentage_after_journey,
 7.         c.registration,
 8.         c.make,
 9.         c.model,
10.         c.battery,
11.         b.battery_voltage
12.     FROM
13.         JourneyData jd
14.     JOIN
15.         Journeys j ON jd.id = j.Journey
16.     JOIN
17.         Cars c ON j.registration = c.registration
18.     JOIN
19.         Batteries b ON c.battery = b.battery_name
20. )
21. SELECT
22.     jc.make,
23.     jc.model,
24.     ROUND(AVG(jc.distance) )AS avg_journey_distance,
25.     ROUND(AVG((jc.battery_percentage_before_journey - jc.battery_percentage_after_journey) *
jc.battery_voltage / 100.0), 2) AS avg_charge_used_kWh
26. FROM
27.     JourneyCharge jc
28. GROUP BY
29.     jc.make, jc.model;
30.
```

Output:

10 rows returned

| | make character varying | model character varying | avg_journey_distance numeric | avg_charge_used_kwh numeric |
|---|---|---|---|---|
| 1 | Buick | Envista | 442 | -0.36 |
| 2 | Ferrari | 296 GTS | 482 | 3.24 |
| 3 | Nissan | Maxima | 443 | 10.32 |
| 4 | BMW | i4 | 137 | 3.06 |
| 5 | Chevrolet | Camaro | 557 | -1.38 |
| 6 | GMC | Sierra 2500 HD Regular Cab | 565 | 3.24 |
| 7 | Lincoln | Navigator L | 398 | 8.16 |
| 8 | Volkswagen | ID.Buzz | 10 | 0.12 |
| 9 | Ram | 1500 Quad Cab | 358 | -0.06 |
| 10 | Honda | Ridgeline | 560 | -9.36 |

### iii. Amount of CO2 emissions saved

Query:

```
SELECT
    p.first_name,
    p.surname,
    ROUND(SUM(jd.distance * 0.310 / 1000),5) AS total_CO2_saved_kg
FROM
    People p
JOIN
    Journeys j ON p.license_number = j.license_number
JOIN
    JourneyData jd ON j.Journey = jd.id
GROUP BY
    p.first_name,
    p.surname;

```

Output:

10 rows returned

| | first_name character varying | surname character varying | total_co2_saved_kg numeric |
|---|---|---|---|
| 1 | Ellie | jenkins | 0.04247 |
| 2 | Isaac | hughes | 0.13702 |
| 3 | Darcie | harris | 0.14942 |
| 4 | Liam | hill | 0.13733 |
| 5 | Poppy | young | 0.00310 |
| 6 | Luca | campbell | 0.17360 |
| 7 | Olivia | dixon | 0.17515 |
| 8 | Aria | gray | 0.12338 |
| 9 | Emma | harris | 0.17267 |
| 10 | Max | adams | 0.11098 |

## Part 7- Self-evaluation

Part 1: I hope to get full marks for this section as I have done research, referenced it, and explained all of my choices in detail.

Part 2: I did not explain myself as well in plain English in this section, but I would still hope to get at least 13/15 because it should all be correct.

Part 3: I worked hard on this section and so long as it's all correct I would except to get full marks.

Part 4: I did not explain the ER diagram in plain English as I thought it was pretty self-explanatory, but assuming the diagram is correct, I would expect to get 13 or 14 out of 15.

Part 5: I really enjoyed this section, and feel like I went above and beyond the requirements by creating Python scripts to populate my database. I worked very hard to make my database, and I really enjoyed doing it. However, the Python scripts, while not required, are also not perfect, so I'm expecting to get 23/25.

Part 6: This should all be correct, and I feel like it does not need much explanation, so I'm expecting to get the full 20 marks.

Overall I really loved this module and this assignment. Databases are just so fun, and I have discovered a passion for manipulating data. I hope that you are as pleased with my work as I am.

## References

[1]

C. Brooke, '25 Baby Names That Push The Character Count', *Business 2 Community*, Nov. 10, 2015. https://www.business2community.com/travel-leisure/25-baby-names-that-push-the-character-count-01374798 (accessed Aug. 10, 2023).

[2]

'Longest single English surname', *Guinness World Records*. https://www.guinnessworldrecords.com/world-records/73269-longest-single-english-surname (accessed Aug. 10, 2023).

[3]

'List of the verified oldest people', *Wikipedia*. Aug. 10, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_the_verified_oldest_people&oldid=1169578505

[4]

'The UK Postcode Format', *IdealPostcodes*. https://ideal-postcodes.co.uk (accessed Aug. 10, 2023).

[5]

J. Gibbs, 'Driving licence numbers explained'. https://www.confused.com/car-insurance/guides/how-to-check-your-driving-licence (accessed Aug. 10, 2023).

[6]

'Land's End to John o' Groats', *Wikipedia*. Jul. 01, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Land%27s_End_to_John_o%27_Groats&oldid=1162761547

[7]

'History of the electric vehicle', *Wikipedia*. Aug. 02, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=History_of_the_electric_vehicle&oldid=1168380863

[8]

'History of steam road vehicles', *Wikipedia*. Jul. 09, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=History_of_steam_road_vehicles&oldid=1164594100

[9]

'Car Batteries | Halfords UK'. https://www.halfords.com/motoring/batteries/car-batteries/ (accessed Aug. 10, 2023).

[10]

'Number plate format - how UK licence plates work | AA'. https://www.theaa.com/car-buying/number-plates (accessed Aug. 10, 2023).

[11]

E. Kendall, ‘Current top 100 baby boy names in the UK - see full list’, *Manchester Evening News*, May 09, 2022. https://www.manchestereveningnews.co.uk/news/parenting/current-top-100-baby-boy-23907877 (accessed Aug. 10, 2023).

[12]

C. U. B. N.-A.-C. Ltd, ‘Top UK baby girl names 2023’. https://www.ukbabynames.com/girls/top (accessed Aug. 10, 2023).

[13]

‘Appendix:English surnames (England and Wales)’, *Wiktionary, the free dictionary*. Apr. 03, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://en.wiktionary.org/w/index.php?title=Appendix:English_surnames_(England_and_Wales)&oldid=72584622

[14]

‘92 Natural Things’, *Simplicable*. https://simplicable.com/world/natural-things (accessed Aug. 10, 2023).

[15]

‘New Car Model/Make Reference List’. https://www.kbb.com/car-make-model-list/new/view-all/ (accessed Aug. 10, 2023).