DATABASE DESIGN

- **For Table User:**
  CREATE TABLE User(
  User_ID INT PRIMARY KEY NOT NULL,
  Name VARCHAR(255),
  Email VARCHAR(255),
  Password VARCHAR(255));

- **For Table Vehicle:**
  CREATE TABLE Vehicle(
  Vehicle_ID INT PRIMARY KEY NOT NULL,
  Brand Varchar(255),
  Model Varchar(100),
  Price Real,
  Average_maintenance_cost REAL,
  Upcoming_Current Varchar(15)CHECK (Upcoming_Current IN ('Upcoming', 'Current')));

- **For Table History:**
  CREATE TABLE History(
  History_ID INT PRIMARY KEY NOT NULL,
  Time TIME,
  User_ID INT,
  Area_Code INT,
  FOREIGN KEY(User_ID) REFERENCES User(User_ID),
  FOREIGN KEY (Area_Code) REFERENCES Location(Area_Code));

- **For Table Charging_Point:**
  CREATE TABLE Charging_Point(
  Charger_ID INT PRIMARY KEY NOT NULL,
  Contact VARCHAR(255),
  Area_code INT,
  Charger_Type VARCHAR(255),
  FOREIGN KEY (Area_Code) REFERENCES Location(Area_Code),
  FOREIGN KEY (Charger_Type) REFERENCES Charger(Charger_Type));

- **For Table Location:**
  CREATE TABLE Location(
  Area_Code INT PRIMARY KEY NOT NULL,
  Name VARCHAR(255));

- **For Table Charger_Type:**
  CREATE TABLE Charger(
  Charger_Type VARCHAR(255) PRIMARY KEY NOT NULL, Cost_per_km REAL);

- **For Table About:**

```
CREATE TABLE About(
History_ID INT,
Vehicle_ID INT,
FOREIGN KEY (History_ID) REFERENCES History(History_ID),
FOREIGN KEY (Vehicle_ID) REFERENCES Vehicle(Vehicle_ID),
PRIMARY KEY(History_ID, Vehicle_ID));
```

- **For Table Compatible_With:**

```
CREATE TABLE Compatible_With(
Vehicle_ID INT,
Charger_Type VARCHAR(255),
FOREIGN KEY (Vehicle_ID) REFERENCES Vehicle(Vehicle_ID),
FOREIGN KEY (Charger_Type) REFERENCES Charger_Type(Charger_Type),
PRIMARY KEY (Vehicle_ID, Charger_Type));
```

DESCRIPTION OF EACH TABLE

```
mysql> DESC User;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| User_ID  | int          | NO   | PRI | NULL    |       |
| Name     | varchar(255) | YES  |     | NULL    |       |
| Email    | varchar(255) | YES  |     | NULL    |       |
| Password | varchar(255) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> DESC Vehicle;
+--------------------------+--------------+------+-----+---------+-------+
| Field                    | Type         | Null | Key | Default | Extra |
+--------------------------+--------------+------+-----+---------+-------+
| Vehicle_ID               | int          | NO   | PRI | NULL    |       |
| Brand                    | varchar(255) | YES  |     | NULL    |       |
| Model                    | varchar(100) | YES  |     | NULL    |       |
| Price                    | double       | YES  |     | NULL    |       |
| Average_maintenance_cost | double       | YES  |     | NULL    |       |
| Upcoming_Current         | varchar(15)  | YES  |     | NULL    |       |
+--------------------------+--------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql> DESC History;
+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| History_ID | int  | NO   | PRI | NULL    |       |
| Time       | time | YES  |     | NULL    |       |
| User_ID    | int  | YES  | MUL | NULL    |       |
| Area_Code  | int  | YES  | MUL | NULL    |       |
+------------+------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> DESC Charging_Point;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| Charger_ID   | int          | NO   | PRI | NULL    |       |
| Contact      | varchar(255) | YES  |     | NULL    |       |
| Area_code    | int          | YES  | MUL | NULL    |       |
| Charger_Type | varchar(255) | YES  | MUL | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> DESC Location;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| Area_Code | int          | NO   | PRI | NULL    |       |
| Name      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> DESC Charger_Type;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| Charger_Type | varchar(255) | NO   | PRI | NULL    |       |
| Cost_per_km  | double       | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql>
```

```
mysql> desc About;
+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| History_ID | int  | NO   | PRI | NULL    |       |
| Vehicle_ID | int  | NO   | PRI | NULL    |       |
+------------+------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> desc Compatible_With;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| Vehicle_ID   | int          | NO   | PRI | NULL    |       |
| Charger_Type | varchar(255) | NO   | PRI | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

QUERIES AND OUTPUTS

- The first query is to find the vehicle id, Brand, and, Model, Charger_type of vehicles that have been produced by Tesla, are charged using an AC Type Charger, or have been produced by BMW, and are charged using a DC Type Charger. We order the results by Vehicle_ID;

  QUERY

```
SELECT v.Vehicle_ID, Brand, Model, w.Charger_Type
FROM Vehicle v NATURAL JOIN Compatible_With w
WHERE Brand ='Tesla' and w.Charger_Type Like 'AC%'
UNION
SELECT v1.Vehicle_ID, Brand, Model, w1.Charger_Type
FROM Vehicle v1 NATURAL JOIN Compatible_With w1
WHERE Brand ='BMW' and w1.Charger_Type Like 'DC%'
ORDER BY Vehicle_ID LIMIT 15;
```

```
+------------+-------+------------------+--------------+
| Vehicle_ID | Brand | Model            | Charger_Type |
+------------+-------+------------------+--------------+
|          2 | BMW   | 5-Series Plug in | DC_CHAdeMo   |
|          8 | BMW   | i8               | DC_CHAdeMo   |
|         31 | Tesla | Model S          | AC_Type_2    |
|         47 | Tesla | Model S          | AC_Type_1    |
|         80 | BMW   | i3               | DC_CHAdeMo   |
|         83 | BMW   | i4               | DC_CHAdeMo   |
|         85 | Tesla | Model S          | AC_Type_2    |
|        100 | BMW   | i3 REx           | DC_CCS       |
|        133 | BMW   | i3 REx           | DC_CHAdeMo   |
|        138 | BMW   | 3-Series Plug in | DC_CHAdeMo   |
|        168 | BMW   | i8               | DC_CHAdeMo   |
|        213 | BMW   | iX               | DC_CCS       |
|        216 | Tesla | Model Y          | AC_Type_1    |
|        240 | Tesla | Model S          | AC_Type_2    |
|        273 | Tesla | Model S          | AC_Type_2    |
+------------+-------+------------------+--------------+
```

- The second query is to display the Brands, and average Price of vehicles that have average price less than the average price of all Tesla Electric Vehicles.

QUERY

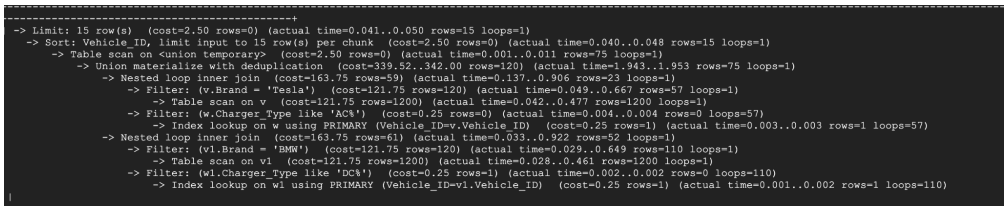
```
SELECT Brand, avg(Price)
FROM Vehicle
GROUP BY Brand
HAVING avg(Price) < (SELECT avg(price)
                     FROM Vehicle
                     WHERE Brand = 'Tesla')
LIMIT 15;
```

Brand	avg(Price)
Audi	55796.15748810017
Miles	36628.370525681865
Fisker	53349.433910579464
BYD	33912.24225955426
Hummer	51446.72480271526
Land	36806.53016853441
Volkswagen	47341.635372441975
Proterra	38091.5267126654
EVI	31322.56121717842
Coda	42991.39302133594
Smith	50979.00286670663
Lucid	45838.89335174021
Jaguar	57267.905293714095
Chrysler	39888.00307455342

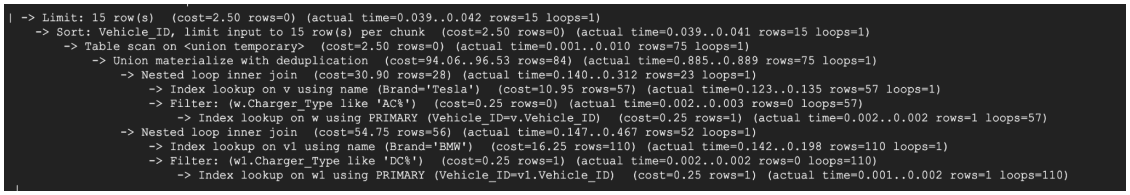
INDEXING

Running Explain Analyze Command on both queries with and without indexing.

1. **QUERY 1:**
   - No Indexing:

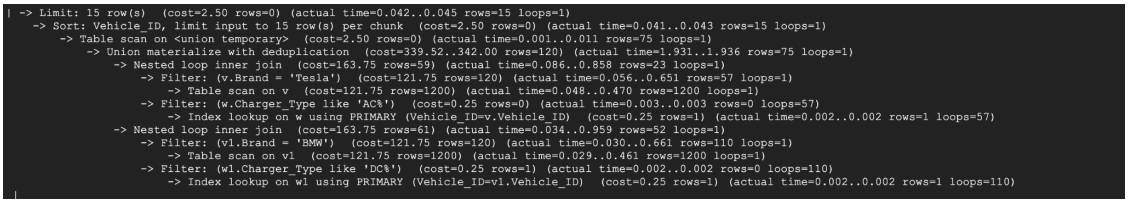


   - Creating Index on Brand



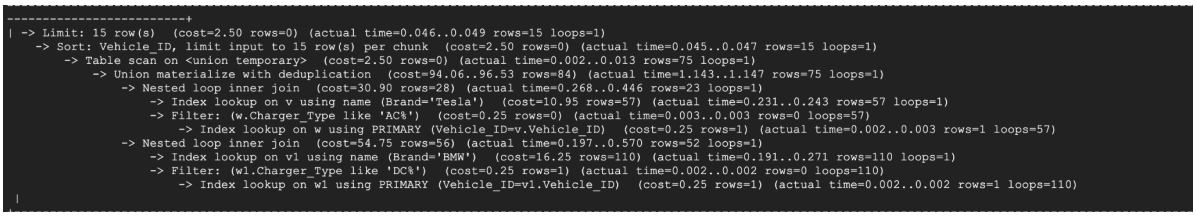
This indexing is a process that is searching two things which are both BMW and Tesla. So we actually still need to go through most of the stuff and return their value. Mostly this is because the process of Join of the two table. The order we created for VehicleID doesn't still keep when we JOIN it with another table as this is only for Vehicle_table. It even for some reason makes the process more complex and more time consuming.

- Creating Index on Compatible_With(Charger_Type)



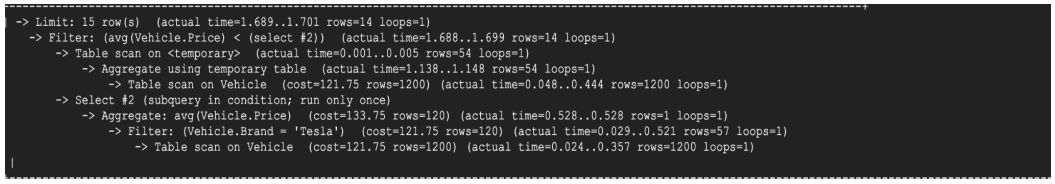
Still like the one before, we don't see improvement in behavior when it comes to filter the type. That is because after "JOIN" action actually make the table lose its characteristics as it is a indexing for certain table Vehicle not for the one after joining and thus that won't improved the behavior of the whole process. We could even see that it somehow even make some procedure using more time which might due to it being more disordered.

- Creating Index on Vehicle(Brand) and Compatible_With(Charger_Type)



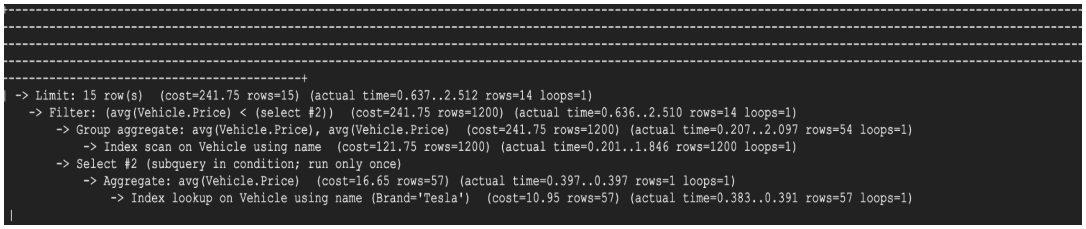
Even after pairing up the brand from Vehicle and Charger_Type from Compatible_With, the results were not significantly different. We believe that it happens due to two reasons: 1. Ordering by Brand doesn't really help as the two Brands are stacked on extreme ends of the memory. 2. Since Charger_Type is already a primary key in the Compatible_With table, indexing using Charger_Type doesn't produce any difference. Hence, the disordering generated by Brand indexing makes the result worse.

2. **QUERY 2:**
    - No Indexing:



```
-> Limit: 15 row(s)  (actual time=1.689..1.701 rows=14 loops=1)
  -> Filter: (avg(Vehicle.Price) < (select #2))  (actual time=1.688..1.699 rows=14 loops=1)
      -> Table scan on <temporary>  (actual time=0.001..0.005 rows=54 loops=1)
          -> Aggregate using temporary table  (actual time=1.138..1.148 rows=54 loops=1)
              -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.048..0.444 rows=1200 loops=1)
      -> Select #2 (subquery in condition; run only once)
          -> Aggregate: avg(Vehicle.Price)  (cost=133.75 rows=120) (actual time=0.528..0.528 rows=1 loops=1)
              -> Filter: (Vehicle.Brand = 'Tesla')  (cost=121.75 rows=120) (actual time=0.029..0.521 rows=57 loops=1)
                  -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.024..0.357 rows=1200 loops=1)
```

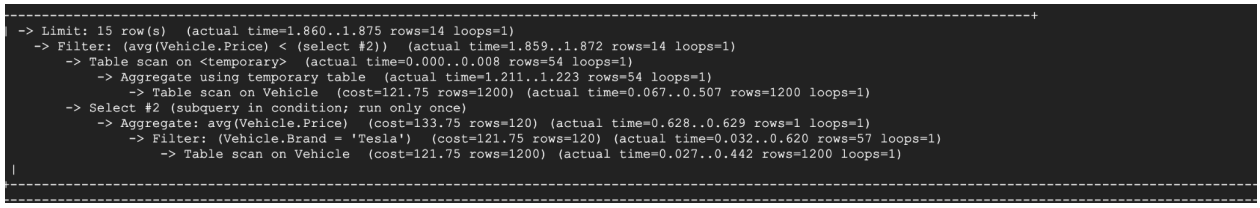
    - Creating Index on Brand:



```
-> Limit: 15 row(s)  (cost=241.75 rows=15) (actual time=0.637..2.512 rows=14 loops=1)
  -> Filter: (avg(Vehicle.Price) < (select #2))  (cost=241.75 rows=1200) (actual time=0.636..2.510 rows=14 loops=1)
      -> Group aggregate: avg(Vehicle.Price), avg(Vehicle.Price)  (cost=241.75 rows=1200) (actual time=0.207..2.097 rows=54 loops=1)
          -> Index scan on Vehicle using name  (cost=121.75 rows=1200) (actual time=0.201..1.846 rows=1200 loops=1)
      -> Select #2 (subquery in condition; run only once)
          -> Aggregate: avg(Vehicle.Price)  (cost=16.65 rows=57) (actual time=0.397..0.397 rows=1 loops=1)
              -> Index lookup on Vehicle using name (Brand='Tesla')  (cost=10.95 rows=57) (actual time=0.383..0.391 rows=57 loops=1)
```

We can see clearly that in this case the cost, time, and, the steps for query all reduce in case of indexing. We believe that by indexing by BrandName, it makes the lookup more efficient as the system has the memory location of each brand, and hence while calculating the average, the system can access different vehicles of the same brand more efficiently and faster.

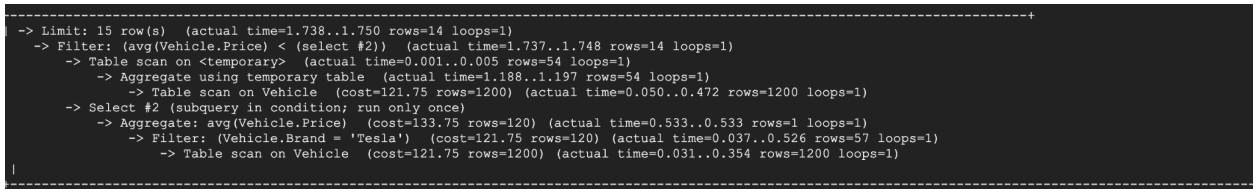
- Creating Index on Price:



```
-> Limit: 15 row(s)  (actual time=1.860..1.875 rows=14 loops=1)
  -> Filter: (avg(Vehicle.Price) < (select #2))  (actual time=1.859..1.872 rows=14 loops=1)
      -> Table scan on <temporary>  (actual time=0.000..0.008 rows=54 loops=1)
          -> Aggregate using temporary table  (actual time=1.211..1.223 rows=54 loops=1)
              -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.067..0.507 rows=1200 loops=1)
      -> Select #2 (subquery in condition; run only once)
          -> Aggregate: avg(Vehicle.Price)  (cost=133.75 rows=120) (actual time=0.628..0.629 rows=1 loops=1)
              -> Filter: (Vehicle.Brand = 'Tesla')  (cost=121.75 rows=120) (actual time=0.032..0.620 rows=57 loops=1)
                  -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.027..0.442 rows=1200 loops=1)
```

We observed that by indexing by Price the performance of the query became worse. We believe that this happens because indexing the location of prices only makes the query more disordered. Since the results are aggregated by the Brand, ordering in the order of price doesn't really fasten the process of calculating the price average. This can possibly happen due to vehicles from one brand being on the extreme ends of the price spectrum, hence separating vehicles of similar brands.

- Creating Index on Model:



```
-> Limit: 15 row(s)  (actual time=1.738..1.750 rows=14 loops=1)
  -> Filter: (avg(Vehicle.Price) < (select #2))  (actual time=1.737..1.748 rows=14 loops=1)
      -> Table scan on <temporary>  (actual time=0.001..0.005 rows=54 loops=1)
          -> Aggregate using temporary table  (actual time=1.188..1.197 rows=54 loops=1)
              -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.050..0.472 rows=1200 loops=1)
      -> Select #2 (subquery in condition; run only once)
          -> Aggregate: avg(Vehicle.Price)  (cost=133.75 rows=120) (actual time=0.533..0.533 rows=1 loops=1)
              -> Filter: (Vehicle.Brand = 'Tesla')  (cost=121.75 rows=120) (actual time=0.037..0.526 rows=57 loops=1)
                  -> Table scan on Vehicle  (cost=121.75 rows=1200) (actual time=0.031..0.354 rows=1200 loops=1)
```

As expected, indexing by model also didn't improve the performance of the query by a lot. We believe that this happens because different brands might have vehicles with similar names, and hence vehicles from different brands will be stored together, thus slowing down the process of calculating average prices by brands. If the query was aggregated by Model instead, this indexing would have produced better results.