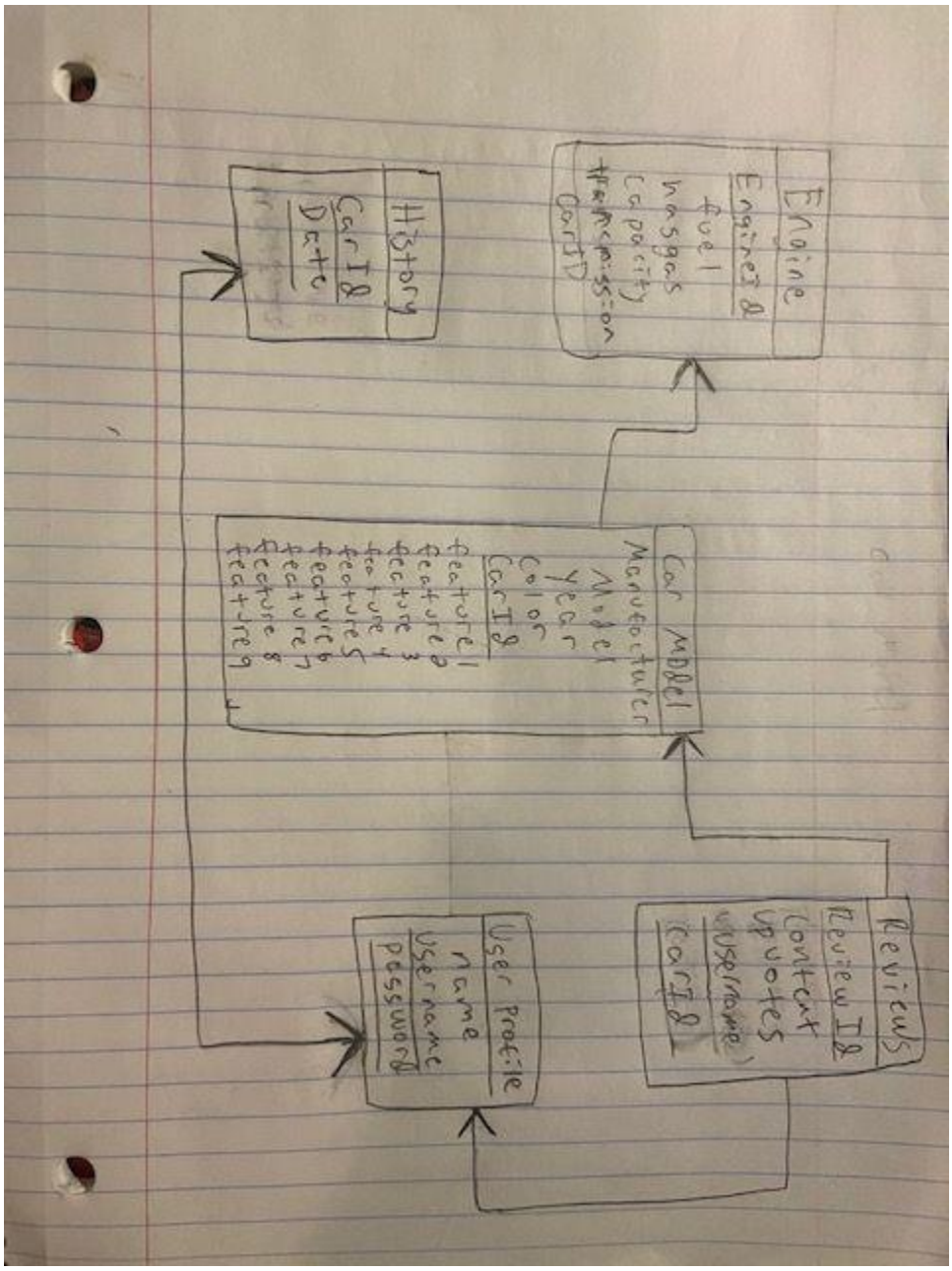
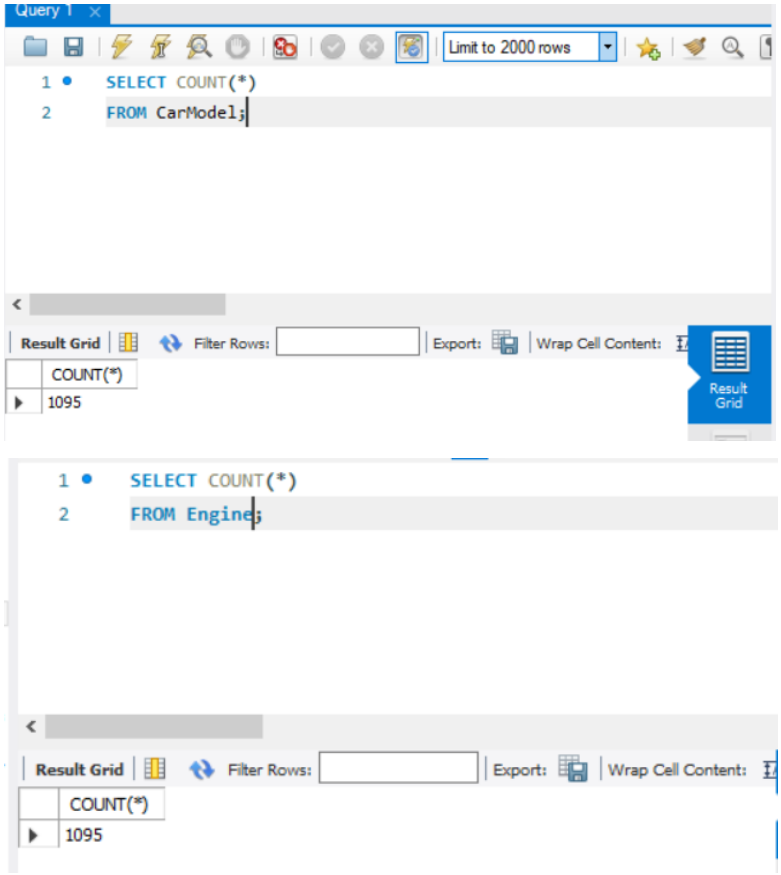
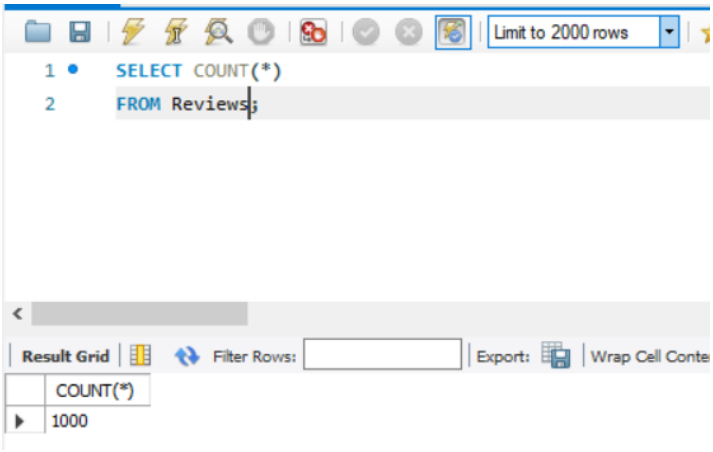
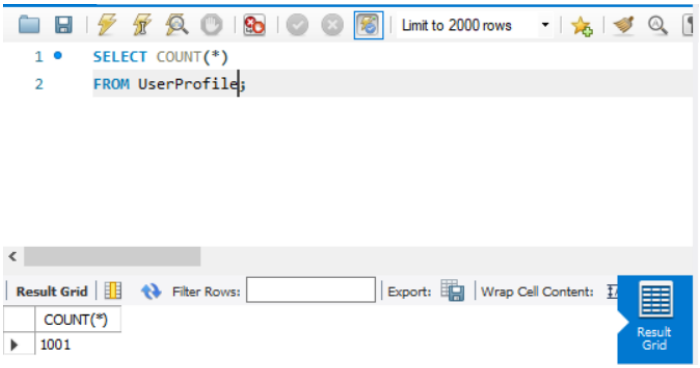
Updated ER diagram:



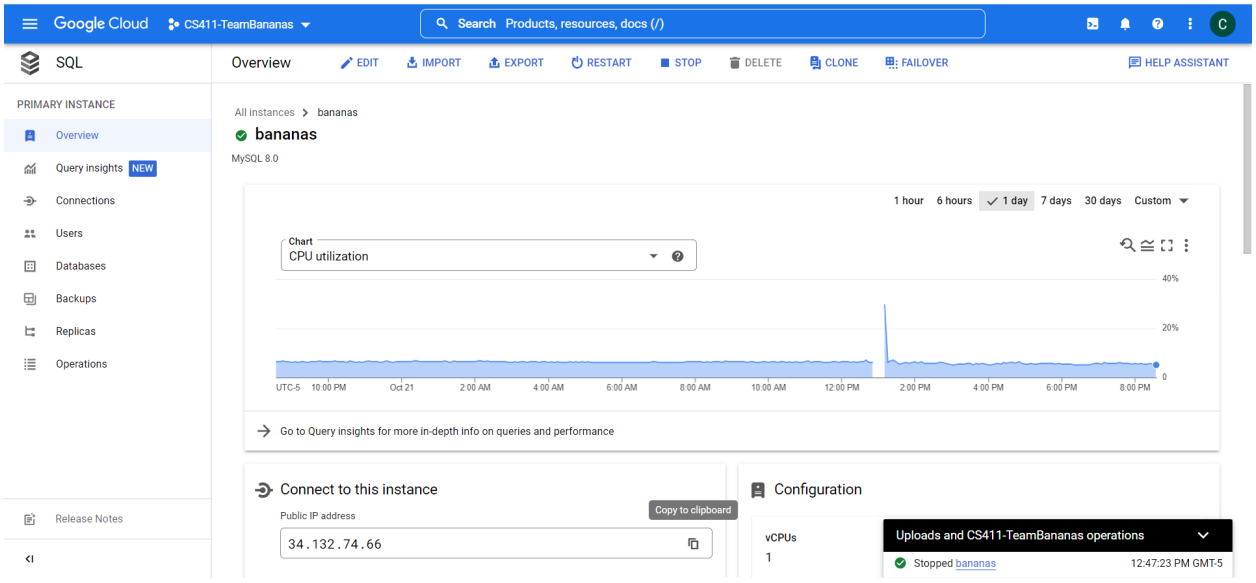
Proof of 1000 rows in 4 tables







Screenshot of connection:



DDL Commands:

Here's the syntax of the CREATE TABLE DDL command:
CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype,...);

```
CREATE TABLE CarModel (
      manufacturer_name VARCHAR(45),
      model_name VARCHAR(45),
      Color VARCHAR(45),
      Year_produced INT,
      CarID INT,
      Feature_0 VARCHAR(45),
      Feature_1 VARCHAR(45),
      Feature_2 VARCHAR(45),
      Feature_3 VARCHAR(45),
      Feature_4 VARCHAR(45),
      Feature_5 VARCHAR(45),
      Feature_6 VARCHAR(45),
      Feature_7 VARCHAR(45),
      Feature_8 VARCHAR(45),
      Feature_9 VARCHAR(45))
      Primary Key(CarID)


CREATE TABLE Engine (
      Engine_fuel VARCHAR(45),
      Engine_has_gas INT,
      Transmission VARCHAR(45),
      Engine_capacity INT,
      EngineId INT,
      Primary Key(EngineId)
)

CREATE TABLE Reviews (
      ReviewId INT,
      Content VARCHAR(45),
      Upvotes INT,
      Username VARCHAR(45),
      CarID INT
      Primary Key(CarID)
      Primary Key(Username)
```

```
)

CREATE TABLE UserProfile (
     Name VARCHAR(45),
     Username VARCHAR(45),
     Password VARCHAR(45),
     Primary Key(Username),
     Primary Key(Password)
)

CREATE TABLE History(
     CarID INT,
     Date VARCHAR(45),
     Primary Key(Date),
     Primary key(CarID)
)
```

Advanced Query #1

JOIN AND HAVING

```
cursor.execute("\
SELECT model_name, year_produced, round(avg(upvotes),2) as avgUpvotes \
FROM CarModel join Reviews using (CarID) \
GROUP BY CarID \
HAVING avg(upvotes) > 20 \
ORDER BY avgUpvotes")
res = cursor.fetchall()
for i in res:
  print(i)
```

Results:

```
('Caliber', 2007, Decimal('34.0000'))
('Legacy', 2001, Decimal('31.0000'))
('Impreza', 2012, Decimal('22.0000'))
('Vesta', 2018, Decimal('35.0000'))
('Rio', 2017, Decimal('21.0000'))
('Tribeca', 2006, Decimal('24.0000'))
('Caravan', 2000, Decimal('38.0000'))
('XRAY', 2018, Decimal('42.0000'))
('Outback', 2008, Decimal('23.0000'))
('Caravan', 1999, Decimal('45.0000'))
('Grand Caravan', 1999, Decimal('48.0000'))
('Stratus', 2004, Decimal('28.5000'))
('Forester', 2012, Decimal('47.0000'))
('Kalina', 2015, Decimal('20.5000'))
('Forester', 2008, Decimal('32.0000'))
('Carnival', 2002, Decimal('38.0000'))
('Forester', 2007, Decimal('27.5000'))
("Cee'd", 2008, Decimal('28.0000'))
('Kalina', 2012, Decimal('27.0000'))
('Grand Caravan', 2008, Decimal('37.0000'))
('469', 1998, Decimal('23.0000'))
('Grand Caravan', 2001, Decimal('29.0000'))
```

Indexing:

Original EXPLAIN ANALYZE OUTPUT:

```
Sort: round(avg(Reviews.upvotes),2)
(actual time=2.895..2.932 rows=382 loops=1)

Filter: (avg(Reviews.upvotes) > 20)
(actual time=2.408..2.671 rows=382 loops=1)

Table scan on <temporary>
(actual time=0.001..0.063 rows=654 loops=1)
```

```
Aggregate using temporary table
(actual time=2.404..2.502 rows=654 loops=1)

Nested loop inner join  (cost=1108.00 rows=1000)
(actual time=0.072..1.630 rows=1000 loops=1)

Filter: (Reviews.CarID is not null) (cost=101.75 rows=1000)
(actual time=0.051..0.383 rows=1000 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000)
(actual time=0.050..0.305 rows=1000 loops=1)

Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=1000))
```

Schema 1: Add "CREATE INDEX upvotesasc ON Reviews(upvotes asc);"
Adding index on upvotes in Reviews Table

EXPLAIN ANALYZE OUTPUT:

```
Sort: round(avg(Reviews.upvotes),2)
(actual time=3.032..3.070 rows=382 loops=1)

Filter: (avg(Reviews.upvotes) > 20)
(actual time=2.550..2.794 rows=382 loops=1)

Table scan on <temporary>
(actual time=0.002..0.067 rows=654 loops=1)

Aggregate using temporary table
(actual time=2.544..2.646 rows=654 loops=1)

Nested loop inner join  (cost=1108.00 rows=1000)
(actual time=0.063..1.711 rows=1000 loops=1)

Filter: (Reviews.CarID is not null)  (cost=101.75 rows=1000)
(actual time=0.050..0.392 rows=1000 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000)
(actual time=0.049..0.316 rows=1000 loops=1)

Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=1000)\n',)
```

Analysis: We find on this schema that there is very little change to the performance of the query. This makes sense because since our upvote data was randomly generated, and our query is finding the average of the upvotes, it does not really matter if we index on the reviews, because it will likely not change our performance.

Schema 2: Add "CREATE INDEX model_name ON CarModel(model_name);"
Adding index on model_name in CarModel

Explain Analyze output:

```
Sort: round(avg(Reviews.upvotes),2)
(actual time=2.996..3.038 rows=382 loops=1)

Filter: (avg(Reviews.upvotes) > 20)
(actual time=2.526..2.769 rows=382 loops=1)

Table scan on <temporary>
(actual time=0.001..0.065 rows=654 loops=1)

Aggregate using temporary table
(actual time=2.520..2.621 rows=654 loops=1)

Nested loop inner join  (cost=1108.00 rows=1000)
(actual time=0.062..1.723 rows=1000 loops=1)

Filter: (Reviews.CarID is not null)  (cost=101.75 rows=1000)
(actual time=0.051..0.413 rows=1000 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000)
(actual time=0.050..0.335 rows=1000 loops=1)

Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=1000)
```

Analysis: It makes sense that indexing on model name did not improve performance because in our data currently there is no correlation between the model name and the average rating. However, with more realistic data, we would expect this to be more effective.

Schema 3: `CREATE INDEX model_name ON CarModel(model_name);`
`CREATE INDEX upvotesasc ON Reviews(upvotes asc);`

Explain Analyze output:

```
Sort: round(avg(Reviews.upvotes),2)
(actual time=3.156..3.195 rows=382 loops=1)

(avg(Reviews.upvotes) > 20)
(actual time=2.677..2.925 rows=382 loops=1

 Table scan on <temporary>
(actual time=0.001..0.076 rows=654 loops=1)

Aggregate using temporary table
(actual time=2.672..2.783 rows=654 loops=1)

Nested loop inner join  (cost=1108.00 rows=1000)
(actual time=0.072..1.758 rows=1000 loops=1)

Filter: (Reviews.CarID is not null)  (cost=101.75 rows=1000)
(actual time=0.060..0.420 rows=1000 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000)
(actual time=0.059..0.334 rows=1000 loops=1)
```

```
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=1000)
```

Analysis: Since we saw no improvement from indexing on model_name and upvotes, it is not surprising indexing on both did not change anything either. We are curious and would like to eventually find out if this would help with more realistic data where there is a correlation between those two variables.

Advanced Query #2

JOIN AND UNION

```
cursor.execute("\
SELECT CarID, color, model_name \
FROM CarModel natural join Reviews \
WHERE upvotes > 25 \
Union \
SELECT CarID, color, model_name \
FROM CarModel natural join Reviews \
WHERE content like '%great%' \
ORDER BY CarID")
res = cursor.fetchall()
for i in res:
  print(i)
```

```
(828, 'brown', 'Optima')
(833, 'silver', 'Sportage')
(834, 'silver', 'Carens')
(836, 'white', 'Sorento')
(837, 'silver', 'Rio')
(840, 'silver', 'Carnival')
(842, 'black', 'Sorento')
(847, 'blue', 'Rio')
(849, 'white', 'Sportage')
(851, 'black', 'Optima')
(853, 'silver', 'Magentis')
(854, 'black', 'Rio')
(856, 'black', 'Rio')
(860, 'black', "Cee'd")
(861, 'silver', 'Rio')
(864, 'blue', 'Carens')
(868, 'red', 'Sportage')
(869, 'blue', 'Sportage')
(870, 'silver', "Cee'd")
(871, 'red', 'Rio')
(874, 'silver', 'Rio')
(877, 'white', 'Sportage')
(880, 'red', 'Rio')
```

Indexing:

Original EXPLAIN ANALYZE Output:

```
Sort: CarID  (cost=2.50 rows=0)
(actual time=0.155..0.181 rows=371 loops=1)

 Table scan on <union temporary>  (cost=2.50 rows=0)
(actual time=0.001..0.033 rows=371 loops=1)
```

```
Union materialize with deduplication  (cost=845.24..847.74 rows=580)
(actual time=2.252..2.300 rows=371 loops=1)

Nested loop inner join  (cost=573.68 rows=469)
(actual time=0.051..1.070 rows=469 loops=1)

 Filter: ((Reviews.upvotes > 25) and (Reviews.CarID is not null))  (cost=101.75 rows=469)
(actual time=0.040..0.401 rows=469 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000) (actual time=0.038..0.297 rows=1000 loops=1)

Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=469)

Nested loop inner join  (cost=213.54 rows=111)
(actual time=0.722..0.722 rows=0 loops=1)

Filter: ((Reviews.content like '%great%') and (Reviews.CarID is not null))  (cost=101.75
rows=111)
(actual time=0.721..0.721 rows=0 loops=1)

Table scan on Reviews  (cost=101.75 rows=1000)
(actual time=0.031..0.249 rows=1000 loops=1

Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(never executed)\n",)
```

Schema 1:

```
CREATE INDEX upvotes ON Reviews(upvotes);
```

Explain Analyze output:

```
("-> Sort: CarID  (cost=2.50 rows=0) (actual time=0.175..0.206 rows=371 loops=1)\n    -> Table
scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.043 rows=371 loops=1)\n
-> Union materialize with deduplication  (cost=845.24..847.74 rows=580) (actual
time=2.252..2.305 rows=371 loops=1)\n            -> Nested loop inner join  (cost=573.68
rows=469) (actual time=0.068..1.055 rows=469 loops=1)\n                -> Filter:
((Reviews.upvotes > 25) and (Reviews.CarID is not null))  (cost=101.75 rows=469) (actual
time=0.055..0.405 rows=469 loops=1)\n                    -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.052..0.292 rows=1000 loops=1)\n                ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=469)\n            -> Nested loop inner join
(cost=213.54 rows=111) (actual time=0.723..0.723 rows=0 loops=1)\n                -> Filter:
((Reviews.content like '%great%') and (Reviews.CarID is not null))  (cost=101.75 rows=111)
(actual time=0.723..0.723 rows=0 loops=1)\n                    -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.026..0.251 rows=1000 loops=1)\n                ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(never executed)
```

Analysis: We see that this indexing schema is not making a large difference, and it may even have made the performance slightly worse. Similar to the last scenario, we believe this has to do with the data not being correlated

Schema 2: CREATE INDEX model_name ON CarModel(model_name);
CREATE INDEX color ON CarModel(color);

Explain Analyze output:

```
("-> Sort: CarID  (cost=2.50 rows=0) (actual time=0.150..0.177 rows=371 loops=1)\n    -> Table
scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.035 rows=371 loops=1)\n
-> Union materialize with deduplication  (cost=845.24..847.74 rows=580) (actual
time=2.320..2.368 rows=371 loops=1)\n           -> Nested loop inner join  (cost=573.68
rows=469) (actual time=0.123..1.140 rows=469 loops=1)\n               -> Filter:
((Reviews.upvotes > 25) and (Reviews.CarID is not null))  (cost=101.75 rows=469) (actual
time=0.113..0.478 rows=469 loops=1)\n                   -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.110..0.359 rows=1000 loops=1)\n               ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.001..0.001 rows=1 loops=469)\n           -> Nested loop inner join
(cost=213.54 rows=111) (actual time=0.732..0.732 rows=0 loops=1)\n               -> Filter:
((Reviews.content like '%great%') and (Reviews.CarID is not null))  (cost=101.75 rows=111)
(actual time=0.731..0.731 rows=0 loops=1)\n                   -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.030..0.248 rows=1000 loops=1)\n               ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(never executed)\n",)
```

Analysis: This schema did not change the performance much. We indexed on car model name here, which likely has no correlation to the review of the car.

Schema 3:
CREATE INDEX model_name ON CarModel(model_name);
CREATE INDEX color ON CarModel(color);
CREATE INDEX upvotes ON Reviews(upvotes);

Explain Analyze output:

```
("-> Sort: CarID  (cost=2.50 rows=0) (actual time=0.229..0.256 rows=371 loops=1)\n    -> Table
scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.009..0.046 rows=371 loops=1)\n
-> Union materialize with deduplication  (cost=845.24..847.74 rows=580) (actual
time=2.930..2.978 rows=371 loops=1)\n           -> Nested loop inner join  (cost=573.68
rows=469) (actual time=0.088..1.292 rows=469 loops=1)\n               -> Filter:
((Reviews.upvotes > 25) and (Reviews.CarID is not null))  (cost=101.75 rows=469) (actual
time=0.076..0.486 rows=469 loops=1)\n                   -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.074..0.365 rows=1000 loops=1)\n               ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(actual time=0.002..0.002 rows=1 loops=469)\n           -> Nested loop inner join
(cost=213.54 rows=111) (actual time=0.999..0.999 rows=0 loops=1)\n               -> Filter:
((Reviews.content like '%great%') and (Reviews.CarID is not null))  (cost=101.75 rows=111)
(actual time=0.998..0.998 rows=0 loops=1)\n                   -> Table scan on Reviews
(cost=101.75 rows=1000) (actual time=0.036..0.271 rows=1000 loops=1)\n               ->
Single-row index lookup on CarModel using PRIMARY (CarID=Reviews.CarID)  (cost=0.91 rows=1)
(never executed)\n",)
```

Analysis: This schema made our performance worse. Cleary indexing on upvotes has a negative impact on performance, and we suspect the other two indexes had not much impact.