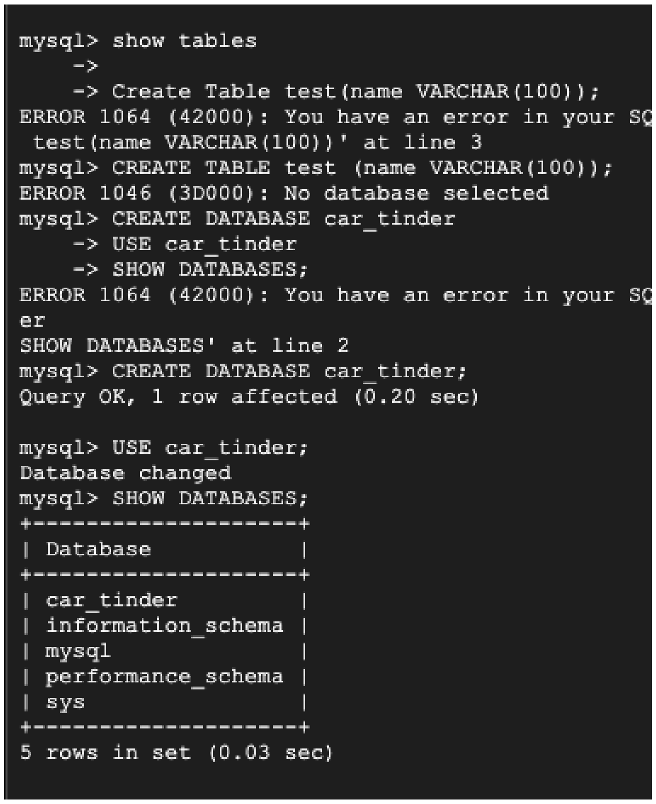
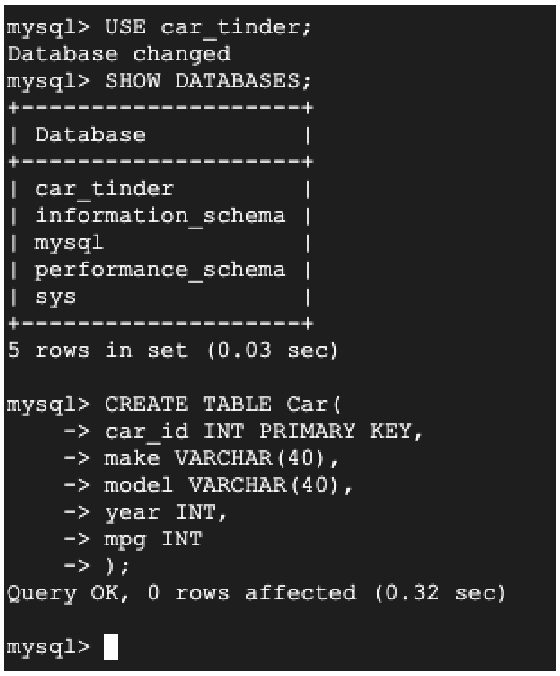
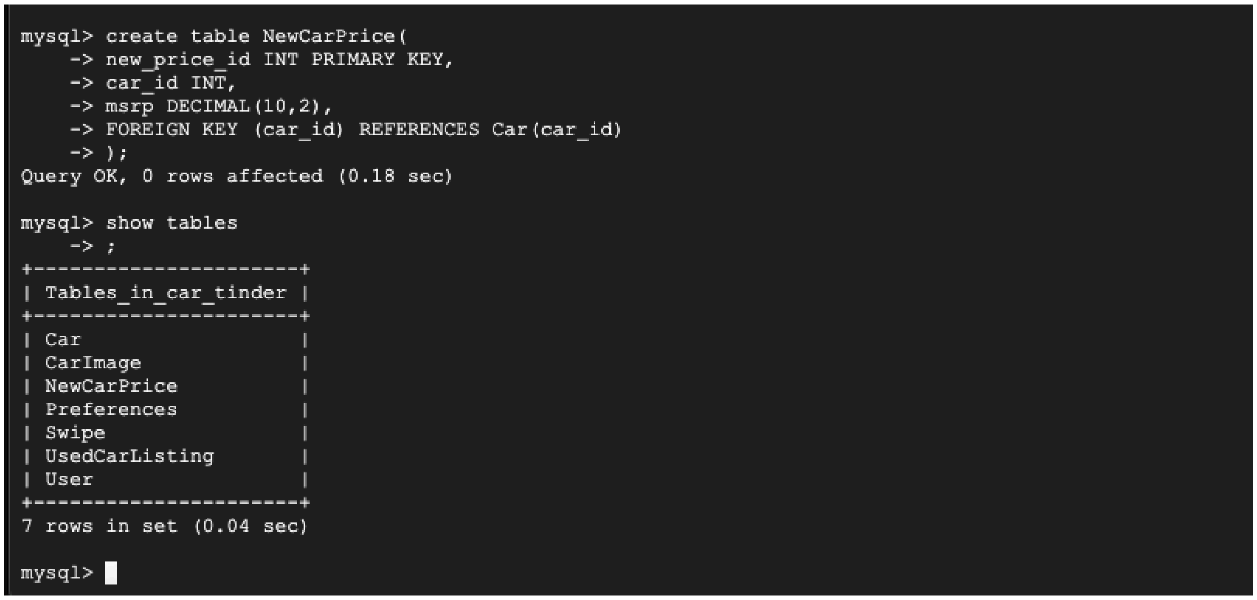
DDL Commands and proof of command line information:



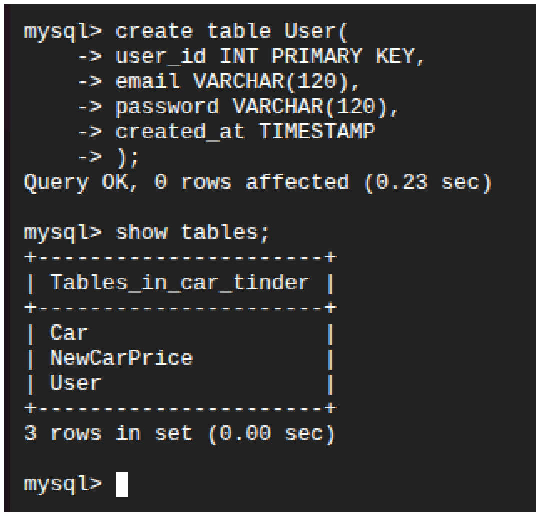
```
mysql> show tables
    ->
    -> Create Table test(name VARCHAR(100));
ERROR 1064 (42000): You have an error in your SQ
 test(name VARCHAR(100))' at line 3
mysql> CREATE TABLE test (name VARCHAR(100));
ERROR 1046 (3D000): No database selected
mysql> CREATE DATABASE car_tinder
    -> USE car_tinder
    -> SHOW DATABASES;
ERROR 1064 (42000): You have an error in your SQ
er
SHOW DATABASES' at line 2
mysql> CREATE DATABASE car_tinder;
Query OK, 1 row affected (0.20 sec)

mysql> USE car_tinder;
Database changed
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| car_tinder         |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.03 sec)
```



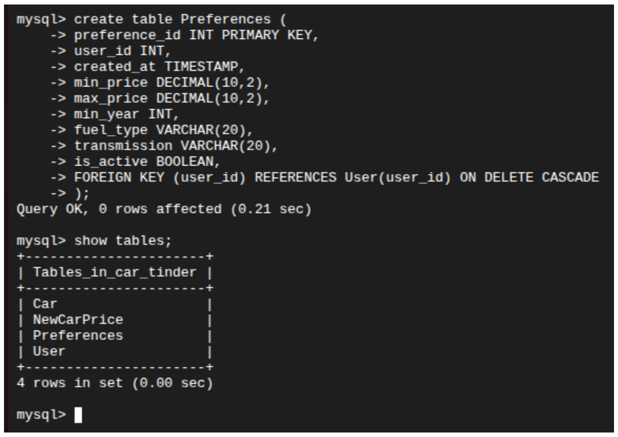
```
mysql> USE car_tinder;
Database changed
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| car_tinder         |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.03 sec)

mysql> CREATE TABLE Car(
    -> car_id INT PRIMARY KEY,
    -> make VARCHAR(40),
    -> model VARCHAR(40),
    -> year INT,
    -> mpg INT
    -> );
Query OK, 0 rows affected (0.32 sec)

mysql>
```



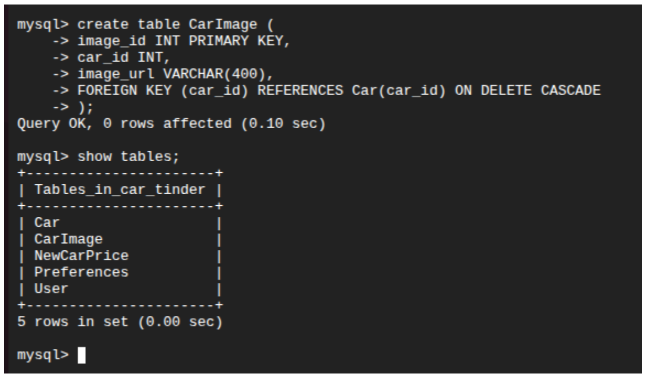
```
mysql> create table NewCarPrice(
    -> new_price_id INT PRIMARY KEY,
    -> car_id INT,
    -> msrp DECIMAL(10,2),
    -> FOREIGN KEY (car_id) REFERENCES Car(car_id)
    -> );
Query OK, 0 rows affected (0.18 sec)

mysql> show tables
    -> ;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| CarImage             |
| NewCarPrice          |
| Preferences          |
| Swipe                |
| UsedCarListing       |
| User                 |
+----------------------+
7 rows in set (0.04 sec)

mysql>
```



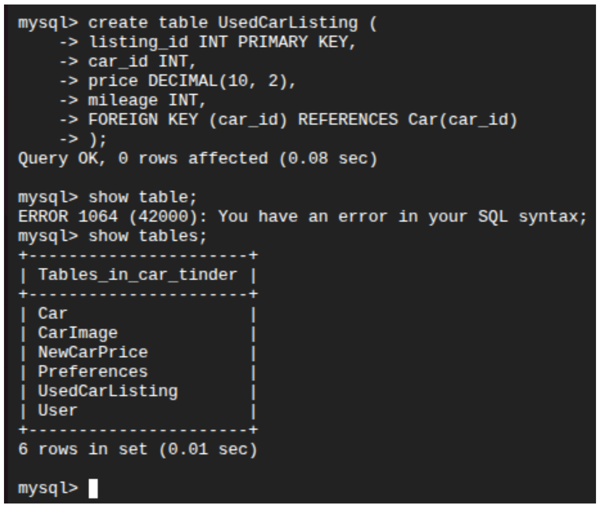
```
mysql> create table User(
    -> user_id INT PRIMARY KEY,
    -> email VARCHAR(120),
    -> password VARCHAR(120),
    -> created_at TIMESTAMP
    -> );
Query OK, 0 rows affected (0.23 sec)

mysql> show tables;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| NewCarPrice          |
| User                 |
+----------------------+
3 rows in set (0.00 sec)

mysql> 
```



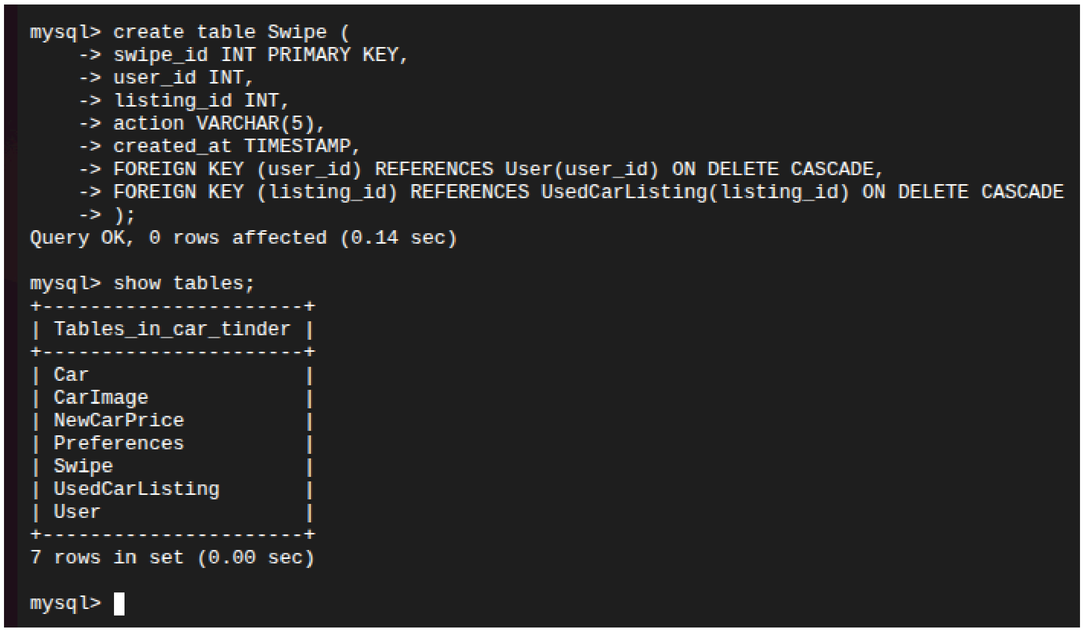
```
mysql> create table Preferences (
    -> preference_id INT PRIMARY KEY,
    -> user_id INT,
    -> created_at TIMESTAMP,
    -> min_price DECIMAL(10,2),
    -> max_price DECIMAL(10,2),
    -> min_year INT,
    -> fuel_type VARCHAR(20),
    -> transmission VARCHAR(20),
    -> is_active BOOLEAN,
    -> FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.21 sec)

mysql> show tables;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| NewCarPrice          |
| Preferences          |
| User                 |
+----------------------+
4 rows in set (0.00 sec)

mysql> 
```



```
mysql> create table CarImage (
    -> image_id INT PRIMARY KEY,
    -> car_id INT,
    -> image_url VARCHAR(400),
    -> FOREIGN KEY (car_id) REFERENCES Car(car_id) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> show tables;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| CarImage             |
| NewCarPrice          |
| Preferences          |
| User                 |
+----------------------+
5 rows in set (0.00 sec)

mysql> 
```

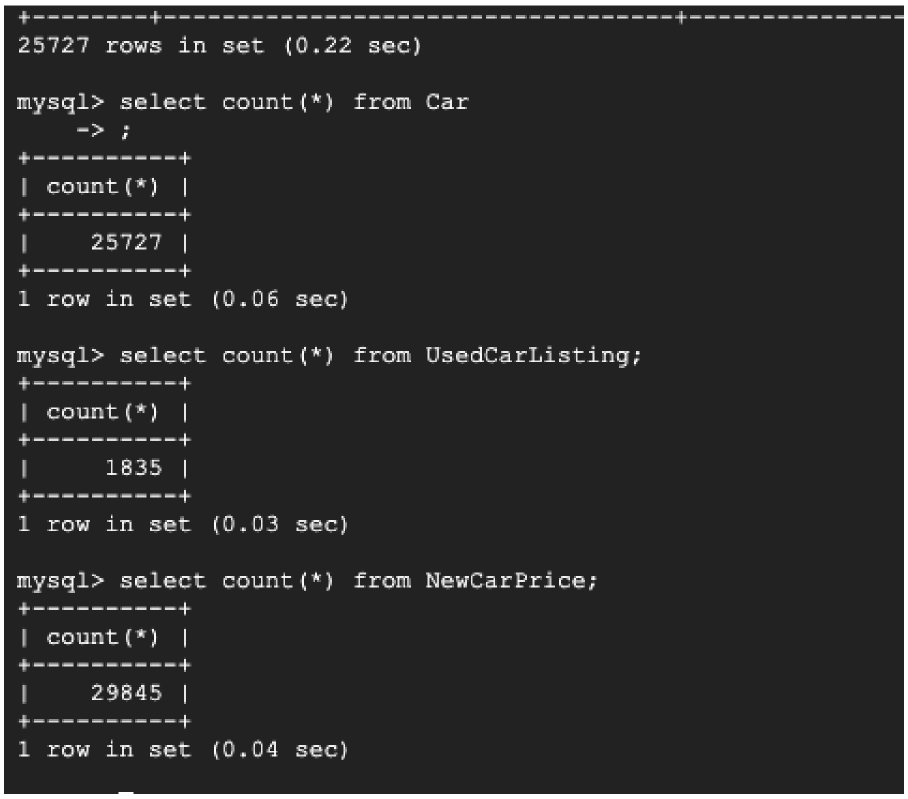


```
mysql> create table UsedCarListing (
    -> listing_id INT PRIMARY KEY,
    -> car_id INT,
    -> price DECIMAL(10, 2),
    -> mileage INT,
    -> FOREIGN KEY (car_id) REFERENCES Car(car_id)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> show table;
ERROR 1064 (42000): You have an error in your SQL syntax;
mysql> show tables;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| CarImage             |
| NewCarPrice          |
| Preferences          |
| UsedCarListing       |
| User                 |
+----------------------+
6 rows in set (0.01 sec)

mysql> 
```



```
mysql> create table Swipe (
    -> swipe_id INT PRIMARY KEY,
    -> user_id INT,
    -> listing_id INT,
    -> action VARCHAR(5),
    -> created_at TIMESTAMP,
    -> FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE,
    -> FOREIGN KEY (listing_id) REFERENCES UsedCarListing(listing_id) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.14 sec)

mysql> show tables;
+----------------------+
| Tables_in_car_tinder |
+----------------------+
| Car                  |
| CarImage             |
| NewCarPrice          |
| Preferences          |
| Swipe                |
| UsedCarListing       |
| User                 |
+----------------------+
7 rows in set (0.00 sec)

mysql> 
```

Proof of data entry:



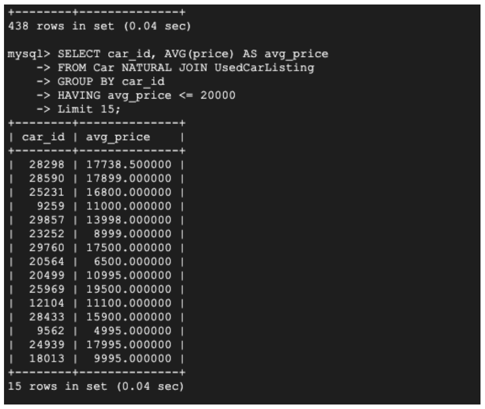
```
+--------+--------------------------------+-------------
25727 rows in set (0.22 sec)

mysql> select count(*) from Car
    -> ;
+----------+
| count(*) |
+----------+
|    25727 |
+----------+
1 row in set (0.06 sec)

mysql> select count(*) from UsedCarListing;
+----------+
| count(*) |
+----------+
|     1835 |
+----------+
1 row in set (0.03 sec)

mysql> select count(*) from NewCarPrice;
+----------+
| count(*) |
+----------+
|    29845 |
+----------+
1 row in set (0.04 sec)
```

Advanced Queries

Query 1: Car ids with average used listing price of 20,000 or less

Advanced Queries

```
SELECT car_id, make, model, AVG(price) AS avg_price
FROM Car NATURAL JOIN UsedCarListing
GROUP BY car_id, make, model
HAVING avg_price <= 20000;
```

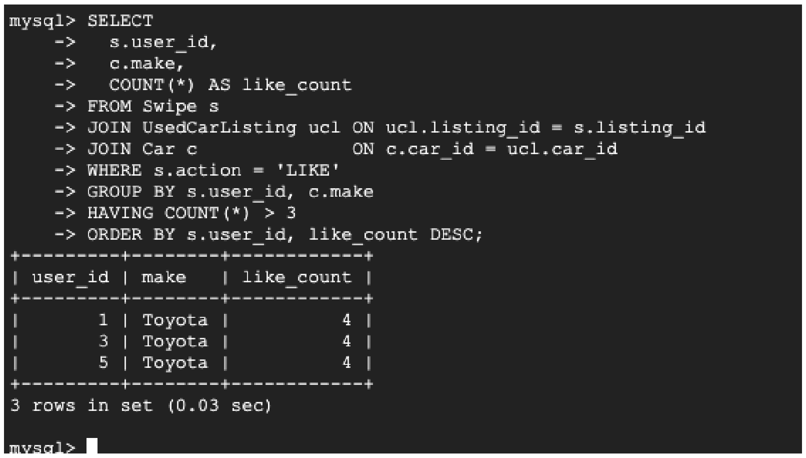


```
+--------+--------------+
438 rows in set (0.04 sec)

mysql> SELECT car_id, AVG(price) AS avg_price
    -> FROM Car NATURAL JOIN UsedCarListing
    -> GROUP BY car_id
    -> HAVING avg_price <= 20000
    -> Limit 15;
+--------+--------------+
| car_id | avg_price    |
+--------+--------------+
|  28298 | 17738.500000 |
|  28590 | 17899.000000 |
|  25231 | 16800.000000 |
|   9259 | 11000.000000 |
|  29857 | 13998.000000 |
|  23252 |  8999.000000 |
|  29760 | 17500.000000 |
|  20564 |  6500.000000 |
|  20499 | 10995.000000 |
|  25969 | 19500.000000 |
|  12104 | 11100.000000 |
|  28433 | 15900.000000 |
|   9562 |  4995.000000 |
|  24939 | 17995.000000 |
|  18013 |  9995.000000 |
+--------+--------------+
15 rows in set (0.04 sec)
```

Justification: We want to identify specific car models in our listings that are affordable, meaning their average used price is $20,000 or less. This query joins the Car and UsedCarListing tables to combine vehicle information with individual sale prices. It groups the data by car_id to calculate the average price for each model and filters the results using the HAVING clause so that only cars with an average price under $20,000 are shown. The results can help users or the recommendation algorithm highlight budget-friendly vehicles and prioritize them for display.

Query 2: Finding Users who have liked the same car make at least 3 times

```
SELECT
  s.user_id,
  c.make,
  COUNT(*) AS like_count
FROM Swipe s
JOIN UsedCarListing ucl ON ucl.listing_id = s.listing_id
JOIN Car c              ON c.car_id = ucl.car_id
WHERE s.action = 'LIKE'
GROUP BY s.user_id, c.make
HAVING COUNT(*) > 3
ORDER BY s.user_id, like_count DESC;
```

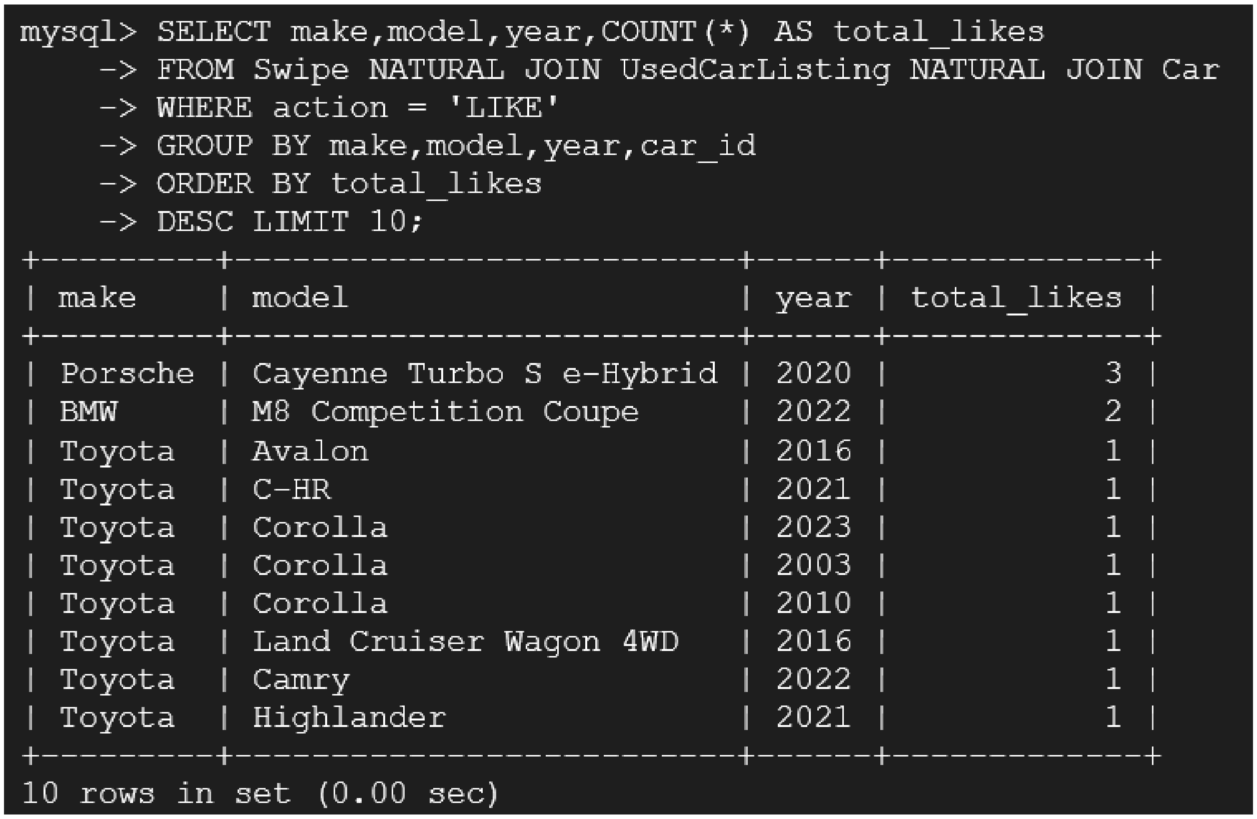

```
mysql> SELECT
    ->   s.user_id,
    ->   c.make,
    ->   COUNT(*) AS like_count
    -> FROM Swipe s
    -> JOIN UsedCarListing ucl ON ucl.listing_id = s.listing_id
    -> JOIN Car c              ON c.car_id = ucl.car_id
    -> WHERE s.action = 'LIKE'
    -> GROUP BY s.user_id, c.make
    -> HAVING COUNT(*) > 3
    -> ORDER BY s.user_id, like_count DESC;
+---------+--------+------------+
| user_id | make   | like_count |
+---------+--------+------------+
|       1 | Toyota |          4 |
|       3 | Toyota |          4 |
|       5 | Toyota |          4 |
+---------+--------+------------+
3 rows in set (0.03 sec)

mysql>
```
Output is less than 15

Justification: We want to identify users who have liked the same make of car at least three times. This query joins the Swipe, UsedCarListing, and Car tables to connect each user’s swipe actions with the specific vehicle makes they interacted with. It filters only the swipes marked as “LIKE,” then groups results by user and car make. Using the HAVING clause ensures we only include makes that were liked more than three times by the same user. This helps reveal strong brand preferences and can be used to personalize future recommendations or highlight patterns in user behavior.

Query 3: Find the top 10 most liked car_id across all users

SELECT make,model,year,COUNT(*) AS total_likes
FROM Swipe NATURAL JOIN UsedCarListing NATURAL JOIN Car
WHERE action = 'LIKE'
GROUP BY make,model,year,car_id
ORDER BY total_likes
DESC LIMIT 10;

Query Example



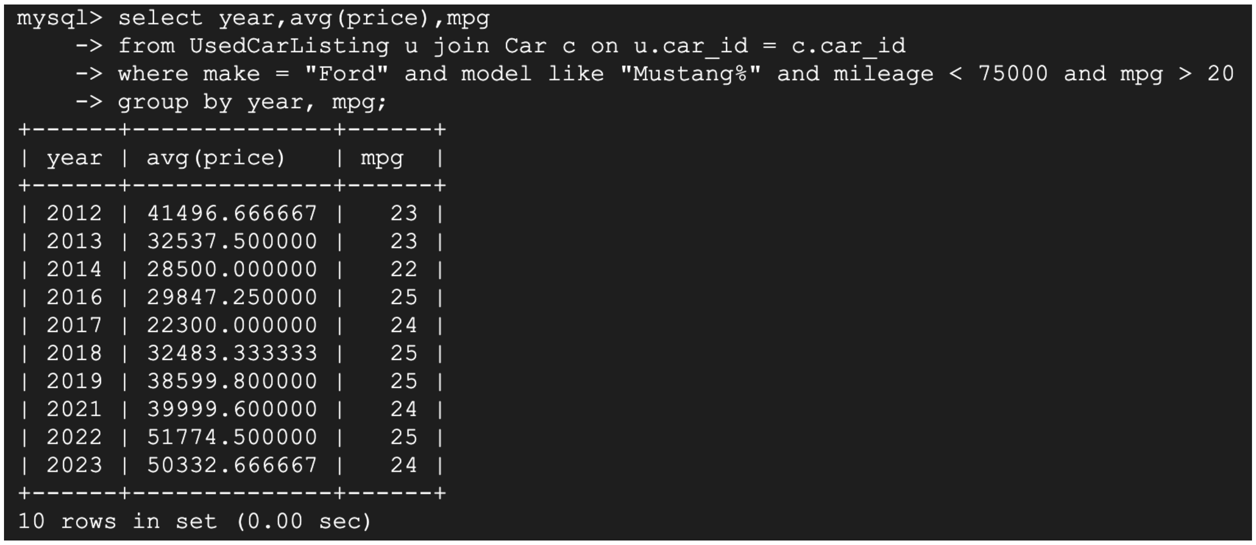
```
mysql> SELECT make,model,year,COUNT(*) AS total_likes
    -> FROM Swipe NATURAL JOIN UsedCarListing NATURAL JOIN Car
    -> WHERE action = 'LIKE'
    -> GROUP BY make,model,year,car_id
    -> ORDER BY total_likes
    -> DESC LIMIT 10;
+---------+-------------------------+------+-------------+
| make    | model                   | year | total_likes |
+---------+-------------------------+------+-------------+
| Porsche | Cayenne Turbo S e-Hybrid | 2020 |           3 |
| BMW     | M8 Competition Coupe    | 2022 |           2 |
| Toyota  | Avalon                  | 2016 |           1 |
| Toyota  | C-HR                    | 2021 |           1 |
| Toyota  | Corolla                 | 2023 |           1 |
| Toyota  | Corolla                 | 2003 |           1 |
| Toyota  | Corolla                 | 2010 |           1 |
| Toyota  | Land Cruiser Wagon 4WD  | 2016 |           1 |
| Toyota  | Camry                   | 2022 |           1 |
| Toyota  | Highlander              | 2021 |           1 |
+---------+-------------------------+------+-------------+
10 rows in set (0.00 sec)
```

Output is less than 15 rows because our swipes table only has ~10 entries

Justification: This query identifies which car models receive the most “like” actions across all users. It joins the Swipe, UsedCarListing, and Car tables to group swipes by make, model, year, and car_id and counts how many times each car has been liked. In implementation, this data can power the app’s initial recommendation engine, especially for new users who have not yet swiped enough for personalized suggestions. When a user first joins and sets broad or minimal preferences, the system can use these aggregated popularity metrics to recommend cars that are widely liked, ensuring that early recommendations are relevant and engaging even before individual behavior data is collected.

Query 4: A user wishes to see the yearly average prices for used Ford Mustangs with a mileage less than 75,000 and a fuel efficiency greater than 20 mpg.

SELECT year, AVG(price), mpg
FROM UsedCarListing u JOIN Car c on u.car_id = c.car_id
WHERE make = ‘Ford' AND model = ‘Mustang’ AND mileage < 75000 AND mpg > 20
GROUP BY year, mpg;



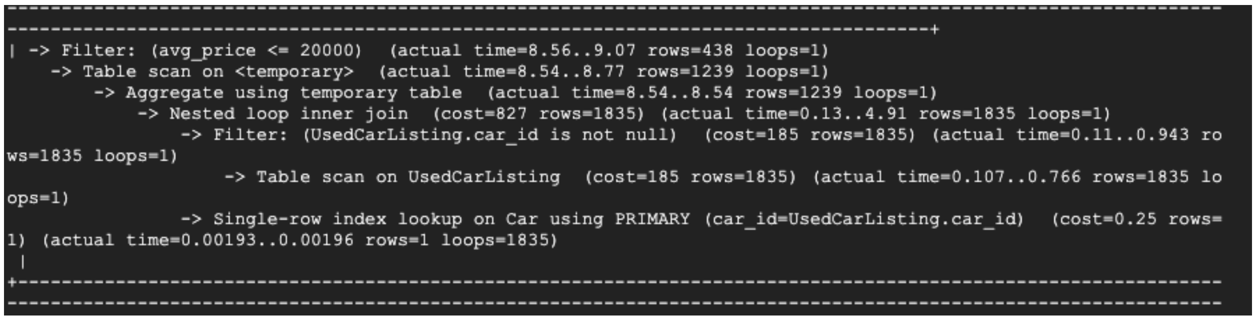
```
mysql> select year,avg(price),mpg
    -> from UsedCarListing u join Car c on u.car_id = c.car_id
    -> where make = "Ford" and model like "Mustang%" and mileage < 75000 and mpg > 20
    -> group by year, mpg;
+------+--------------+------+
| year | avg(price)   | mpg  |
+------+--------------+------+
| 2012 | 41496.666667 |   23 |
| 2013 | 32537.500000 |   23 |
| 2014 | 28500.000000 |   22 |
| 2016 | 29847.250000 |   25 |
| 2017 | 22300.000000 |   24 |
| 2018 | 32483.333333 |   25 |
| 2019 | 38599.800000 |   25 |
| 2021 | 39999.600000 |   24 |
| 2022 | 51774.500000 |   25 |
| 2023 | 50332.666667 |   24 |
+------+--------------+------+
10 rows in set (0.00 sec)
```

Justification: This query is the app's core personalization feature for filtering cars based on preferences. By joining the UsedCarListing and Car tables, we join each individual car listing with key identifiers, specifically make, model, and fuel efficiency, which are the preferences the hypothetical user is providing. Such a prompt is key for a user's research purposes, allowing them to compare the prices of a used car's model over different years filtering for their desired fuel efficiency, communicated through the WHERE clause. In this case, we are able to group by mpg without aggregating since the fuel efficiency is standardized for make and model by year.

Indexing

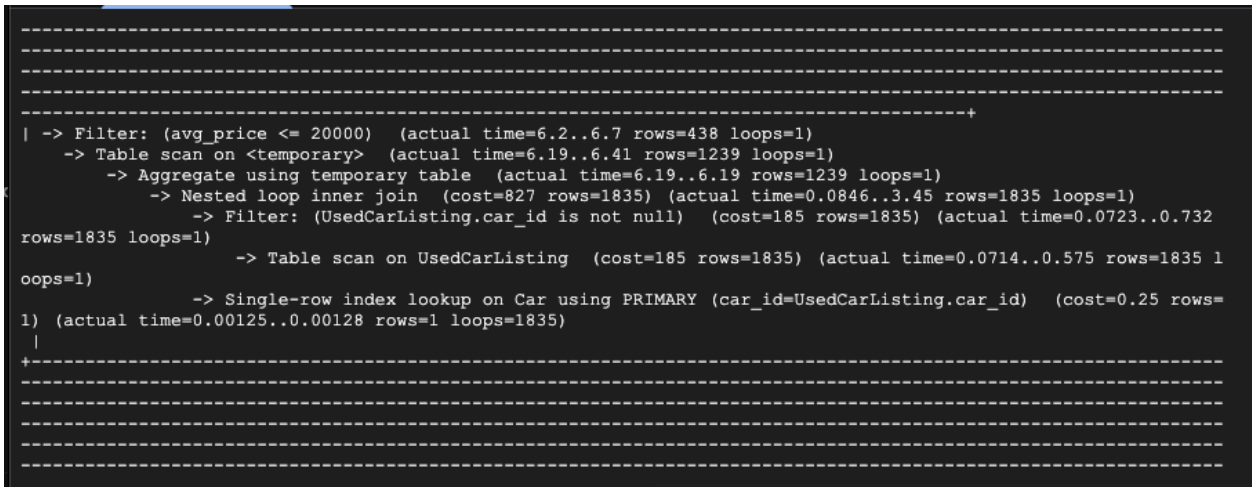
Query 1

No index:



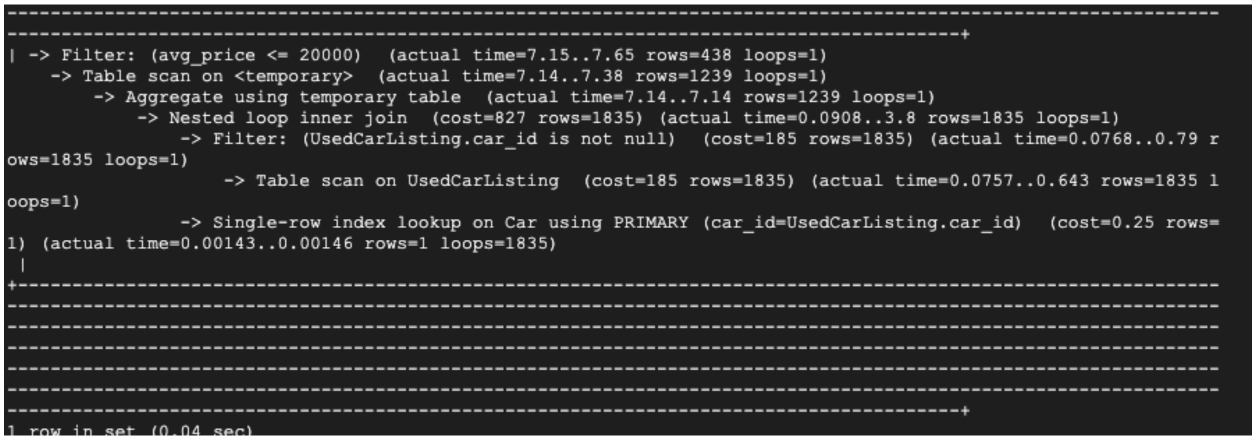
```
---------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| -> Filter: (avg_price <= 20000)  (actual time=8.56..9.07 rows=438 loops=1)
    -> Table scan on <temporary>  (actual time=8.54..8.77 rows=1239 loops=1)
        -> Aggregate using temporary table  (actual time=8.54..8.54 rows=1239 loops=1)
            -> Nested loop inner join  (cost=827 rows=1835) (actual time=0.13..4.91 rows=1835 loops=1)
                -> Filter: (UsedCarListing.car_id is not null)  (cost=185 rows=1835) (actual time=0.11..0.943 ro
ws=1835 loops=1)
                    -> Table scan on UsedCarListing  (cost=185 rows=1835) (actual time=0.107..0.766 rows=1835 lo
ops=1)
                -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.25 rows=
1) (actual time=0.00193..0.00196 rows=1 loops=1835)
 |
+--------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------
```

Adding index for Car(make)



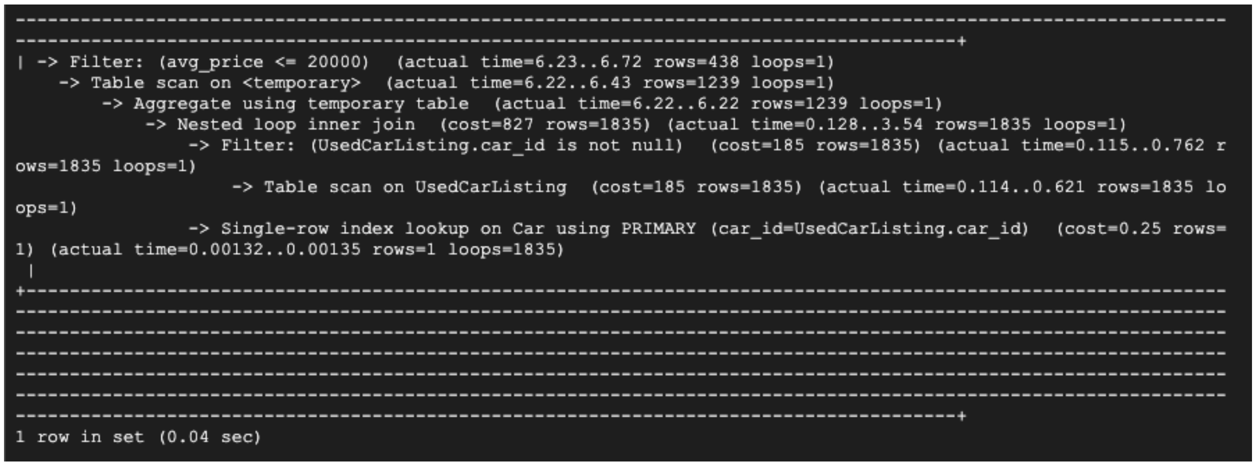
```
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| -> Filter: (avg_price <= 20000)  (actual time=6.2..6.7 rows=438 loops=1)
    -> Table scan on <temporary>  (actual time=6.19..6.41 rows=1239 loops=1)
        -> Aggregate using temporary table  (actual time=6.19..6.19 rows=1239 loops=1)
            -> Nested loop inner join  (cost=827 rows=1835) (actual time=0.0846..3.45 rows=1835 loops=1)
                -> Filter: (UsedCarListing.car_id is not null)  (cost=185 rows=1835) (actual time=0.0723..0.732
rows=1835 loops=1)
                    -> Table scan on UsedCarListing  (cost=185 rows=1835) (actual time=0.0714..0.575 rows=1835 l
oops=1)
                -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.25 rows=
1) (actual time=0.00125..0.00128 rows=1 loops=1835)
 |
+----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
```

Adding index for Car(model)



```
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| -> Filter: (avg_price <= 20000)  (actual time=7.15..7.65 rows=438 loops=1)
    -> Table scan on <temporary>  (actual time=7.14..7.38 rows=1239 loops=1)
        -> Aggregate using temporary table  (actual time=7.14..7.14 rows=1239 loops=1)
            -> Nested loop inner join  (cost=827 rows=1835) (actual time=0.0908..3.8 rows=1835 loops=1)
                -> Filter: (UsedCarListing.car_id is not null)  (cost=185 rows=1835) (actual time=0.0768..0.79 r
ows=1835 loops=1)
                    -> Table scan on UsedCarListing  (cost=185 rows=1835) (actual time=0.0757..0.643 rows=1835 l
oops=1)
                -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.25 rows=
1) (actual time=0.00143..0.00146 rows=1 loops=1835)
 |
+----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

Adding index for Car(make, model)



```
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| -> Filter: (avg_price <= 20000)  (actual time=6.23..6.72 rows=438 loops=1)
    -> Table scan on <temporary>  (actual time=6.22..6.43 rows=1239 loops=1)
        -> Aggregate using temporary table  (actual time=6.22..6.22 rows=1239 loops=1)
            -> Nested loop inner join  (cost=827 rows=1835) (actual time=0.128..3.54 rows=1835 loops=1)
                -> Filter: (UsedCarListing.car_id is not null)  (cost=185 rows=1835) (actual time=0.115..0.762 r
ows=1835 loops=1)
                    -> Table scan on UsedCarListing  (cost=185 rows=1835) (actual time=0.114..0.621 rows=1835 lo
ops=1)
                -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.25 rows=
1) (actual time=0.00132..0.00135 rows=1 loops=1835)
 |
+----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

Cost: 827

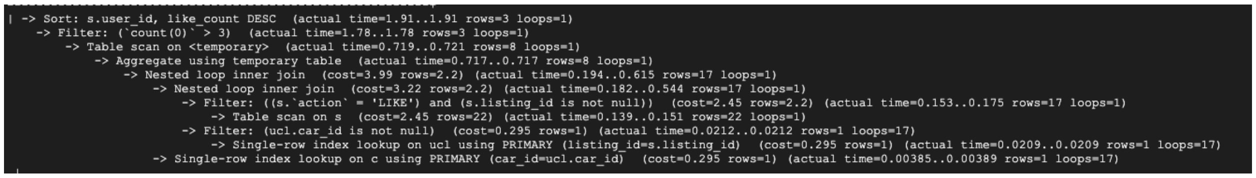
Index Strategy	Cost

None	827
Car(make,model)	827
Car(make)	827
Car(model)	827

Justification: We did not observe any cost reduction with different indexing strategies. This is to be expected, as 1. There is no “WHERE” clause to reduce cardinality before execution, 2. The “HAVING” clause applies after computing AVG(price), so it cannot help with pruning, 3. The “JOIN” is already happening on primary keys/foreign keys which are already indexed, and 4. The “GROUP BY” spans two tables, preventing index-only grouping. For these reasons we see no reason to use an additional indexing strategy for this query.

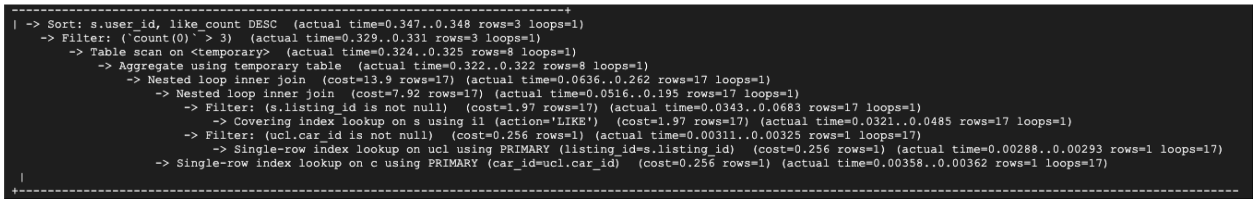
Query 2

No Index



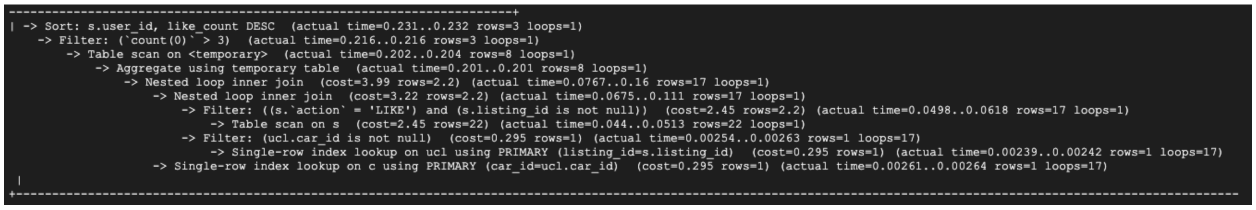
```
| -> Sort: s.user_id, like_count DESC  (actual time=1.91..1.91 rows=3 loops=1)
    -> Filter: (`count(0)` > 3)  (actual time=1.78..1.78 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.719..0.721 rows=8 loops=1)
            -> Aggregate using temporary table  (actual time=0.717..0.717 rows=8 loops=1)
                -> Nested loop inner join  (cost=3.99 rows=2.2) (actual time=0.194..0.615 rows=17 loops=1)
                    -> Nested loop inner join  (cost=3.22 rows=2.2) (actual time=0.182..0.544 rows=17 loops=1)
                        -> Filter: ((s.`action` = 'LIKE') and (s.listing_id is not null))  (cost=2.45 rows=2.2) (actual time=0.153..0.175 rows=17 loops=1)
                            -> Table scan on s  (cost=2.45 rows=22) (actual time=0.139..0.151 rows=22 loops=1)
                        -> Filter: (ucl.car_id is not null)  (cost=0.295 rows=1) (actual time=0.0212..0.0212 rows=1 loops=17)
                            -> Single-row index lookup on ucl using PRIMARY (listing_id=s.listing_id)  (cost=0.295 rows=1) (actual time=0.0209..0.0209 rows=1 loops=17)
                    -> Single-row index lookup on c using PRIMARY (car_id=ucl.car_id)  (cost=0.295 rows=1) (actual time=0.00385..0.00389 rows=1 loops=17)
```

Adding index on Swipe(action);



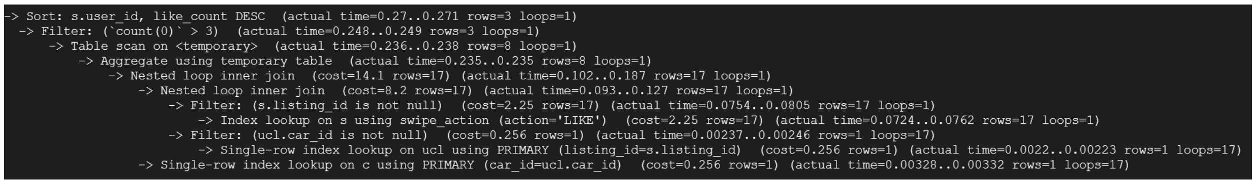
```
| -> Sort: s.user_id, like_count DESC  (actual time=0.347..0.348 rows=3 loops=1)
    -> Filter: (`count(0)` > 3)  (actual time=0.329..0.331 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.324..0.325 rows=8 loops=1)
            -> Aggregate using temporary table  (actual time=0.322..0.322 rows=8 loops=1)
                -> Nested loop inner join  (cost=13.9 rows=17) (actual time=0.0636..0.262 rows=17 loops=1)
                    -> Nested loop inner join  (cost=7.92 rows=17) (actual time=0.0516..0.195 rows=17 loops=1)
                        -> Filter: (s.listing_id is not null)  (cost=1.97 rows=17) (actual time=0.0343..0.0683 rows=17 loops=1)
                            -> Covering index lookup on s using i1 (action='LIKE')  (cost=1.97 rows=17) (actual time=0.0321..0.0485 rows=17 loops=1)
                        -> Filter: (ucl.car_id is not null)  (cost=0.256 rows=1) (actual time=0.00311..0.00325 rows=1 loops=17)
                            -> Single-row index lookup on ucl using PRIMARY (listing_id=s.listing_id)  (cost=0.256 rows=1) (actual time=0.00288..0.00293 rows=1 loops=17)
                    -> Single-row index lookup on c using PRIMARY (car_id=ucl.car_id)  (cost=0.256 rows=1) (actual time=0.00358..0.00362 rows=1 loops=17)
 |
```

Adding index on Car(make);



```
| -> Sort: s.user_id, like_count DESC  (actual time=0.231..0.232 rows=3 loops=1)
    -> Filter: (`count(0)` > 3)  (actual time=0.216..0.216 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.202..0.204 rows=8 loops=1)
            -> Aggregate using temporary table  (actual time=0.201..0.201 rows=8 loops=1)
                -> Nested loop inner join  (cost=3.99 rows=2.2) (actual time=0.0767..0.16 rows=17 loops=1)
                    -> Nested loop inner join  (cost=3.22 rows=2.2) (actual time=0.0675..0.111 rows=17 loops=1)
                        -> Filter: ((s.`action` = 'LIKE') and (s.listing_id is not null))  (cost=2.45 rows=2.2) (actual time=0.0498..0.0618 rows=17 loops=1)
                            -> Table scan on s  (cost=2.45 rows=22) (actual time=0.044..0.0513 rows=22 loops=1)
                        -> Filter: (ucl.car_id is not null)  (cost=0.295 rows=1) (actual time=0.00254..0.00263 rows=1 loops=17)
                            -> Single-row index lookup on ucl using PRIMARY (listing_id=s.listing_id)  (cost=0.295 rows=1) (actual time=0.00239..0.00242 rows=1 loops=17)
                    -> Single-row index lookup on c using PRIMARY (car_id=ucl.car_id)  (cost=0.295 rows=1) (actual time=0.00261..0.00264 rows=1 loops=17)
 |
```

Adding index on Car(make), Swipe(action);



```
-> Sort: s.user_id, like_count DESC  (actual time=0.27..0.271 rows=3 loops=1)
  -> Filter: (`count(0)` > 3)  (actual time=0.248..0.249 rows=3 loops=1)
      -> Table scan on <temporary>  (actual time=0.236..0.238 rows=8 loops=1)
          -> Aggregate using temporary table  (actual time=0.235..0.235 rows=8 loops=1)
              -> Nested loop inner join  (cost=14.1 rows=17) (actual time=0.102..0.187 rows=17 loops=1)
                  -> Nested loop inner join  (cost=8.2 rows=17) (actual time=0.093..0.127 rows=17 loops=1)
                      -> Filter: (s.listing_id is not null)  (cost=2.25 rows=17) (actual time=0.0754..0.0805 rows=17 loops=1)
                          -> Index lookup on s using swipe_action (action='LIKE')  (cost=2.25 rows=17) (actual time=0.0724..0.0762 rows=17 loops=1)
                      -> Filter: (ucl.car_id is not null)  (cost=0.256 rows=1) (actual time=0.00237..0.00246 rows=1 loops=17)
                          -> Single-row index lookup on ucl using PRIMARY (listing_id=s.listing_id)  (cost=0.256 rows=1) (actual time=0.0022..0.00223 rows=1 loops=17)
                  -> Single-row index lookup on c using PRIMARY (car_id=ucl.car_id)  (cost=0.256 rows=1) (actual time=0.00328..0.00332 rows=1 loops=17)
```

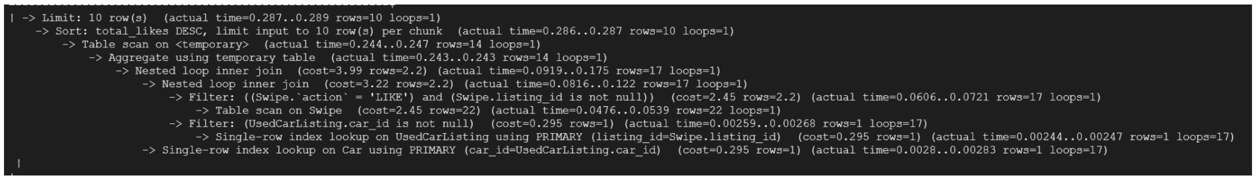
Index	Cost

None	3.99
Swipe(action)	13.9
Car(make)	3.99
Car(make), Swipe(action)	14.1

Justification: We did not observe a cost reduction from any indexing strategies, and some increased the cost, namely those involving an index on Swipe(action). This is because the condition “action=’LIKE’” does not filter very many rows (4/22), while indexing incurs additional work. Indexing Car(make) did not improve performance because “make” is not used to filter rows. Overall these results are expected for low-selectivity conditions and a cross table aggregation that requires a temporary table and a filesort. For these reasons we see no reason to use an additional indexing strategy for this query.

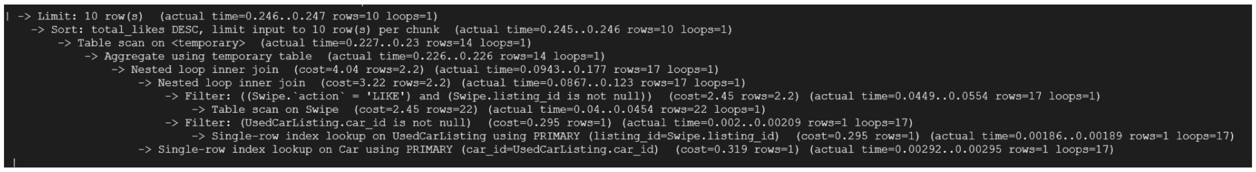
Query 3

No Index:



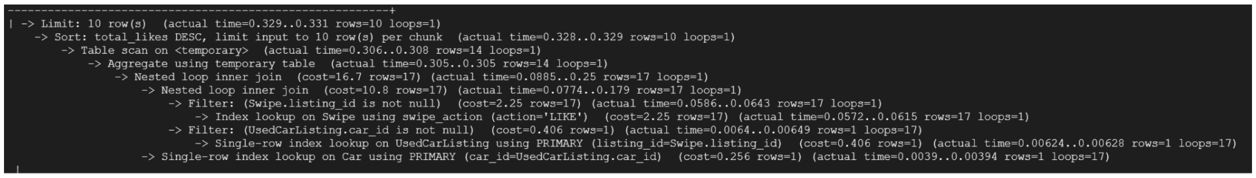
```
| -> Limit: 10 row(s)  (actual time=0.287..0.289 rows=10 loops=1)
    -> Sort: total_likes DESC, limit input to 10 row(s) per chunk  (actual time=0.286..0.287 rows=10 loops=1)
        -> Table scan on <temporary>  (actual time=0.244..0.247 rows=14 loops=1)
            -> Aggregate using temporary table  (actual time=0.243..0.243 rows=14 loops=1)
                -> Nested loop inner join  (cost=3.99 rows=2.2) (actual time=0.0919..0.175 rows=17 loops=1)
                    -> Nested loop inner join  (cost=3.22 rows=2.2) (actual time=0.0816..0.122 rows=17 loops=1)
                        -> Filter: ((Swipe.`action` = 'LIKE') and (Swipe.listing_id is not null))  (cost=2.45 rows=2.2) (actual time=0.0606..0.0721 rows=17 loops=1)
                            -> Table scan on Swipe  (cost=2.45 rows=22) (actual time=0.0476..0.0539 rows=22 loops=1)
                        -> Filter: (UsedCarListing.car_id is not null)  (cost=0.295 rows=1) (actual time=0.00259..0.00268 rows=1 loops=17)
                            -> Single-row index lookup on UsedCarListing using PRIMARY (listing_id=Swipe.listing_id)  (cost=0.295 rows=1) (actual time=0.00244..0.00247 rows=1 loops=17)
                    -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.295 rows=1) (actual time=0.0028..0.00283 rows=1 loops=17)
 |
```

Adding Index on Car(make):



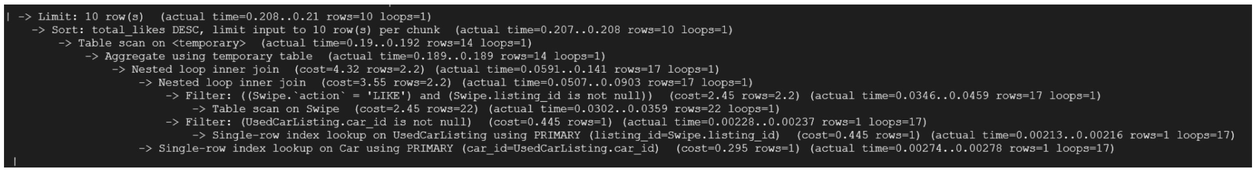
```
| -> Limit: 10 row(s)  (actual time=0.246..0.247 rows=10 loops=1)
    -> Sort: total_likes DESC, limit input to 10 row(s) per chunk  (actual time=0.245..0.246 rows=10 loops=1)
        -> Table scan on <temporary>  (actual time=0.227..0.23 rows=14 loops=1)
            -> Aggregate using temporary table  (actual time=0.226..0.226 rows=14 loops=1)
                -> Nested loop inner join  (cost=4.04 rows=2.2) (actual time=0.0943..0.177 rows=17 loops=1)
                    -> Nested loop inner join  (cost=3.22 rows=2.2) (actual time=0.0867..0.123 rows=17 loops=1)
                        -> Filter: ((Swipe.`action` = 'LIKE') and (Swipe.listing_id is not null))  (cost=2.45 rows=2.2) (actual time=0.0449..0.0554 rows=17 loops=1)
                            -> Table scan on Swipe  (cost=2.45 rows=22) (actual time=0.04..0.0454 rows=22 loops=1)
                        -> Filter: (UsedCarListing.car_id is not null)  (cost=0.295 rows=1) (actual time=0.002..0.00209 rows=1 loops=17)
                            -> Single-row index lookup on UsedCarListing using PRIMARY (listing_id=Swipe.listing_id)  (cost=0.295 rows=1) (actual time=0.00186..0.00189 rows=1 loops=17)
                    -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.319 rows=1) (actual time=0.00292..0.00295 rows=1 loops=17)
 |
```

Index on Swipe(action):



```
--------------------------------------------------------+
| -> Limit: 10 row(s)  (actual time=0.329..0.331 rows=10 loops=1)
    -> Sort: total_likes DESC, limit input to 10 row(s) per chunk  (actual time=0.328..0.329 rows=10 loops=1)
        -> Table scan on <temporary>  (actual time=0.306..0.308 rows=14 loops=1)
            -> Aggregate using temporary table  (actual time=0.305..0.305 rows=14 loops=1)
                -> Nested loop inner join  (cost=16.7 rows=17) (actual time=0.0885..0.25 rows=17 loops=1)
                    -> Nested loop inner join  (cost=10.8 rows=17) (actual time=0.0774..0.179 rows=17 loops=1)
                        -> Filter: (Swipe.listing_id is not null)  (cost=2.25 rows=17) (actual time=0.0586..0.0643 rows=17 loops=1)
                            -> Index lookup on Swipe using swipe_action (action='LIKE')  (cost=2.25 rows=17) (actual time=0.0572..0.0615 rows=17 loops=1)
                        -> Filter: (UsedCarListing.car_id is not null)  (cost=0.406 rows=1) (actual time=0.0064..0.00649 rows=1 loops=17)
                            -> Single-row index lookup on UsedCarListing using PRIMARY (listing_id=Swipe.listing_id)  (cost=0.406 rows=1) (actual time=0.00624..0.00628 rows=1 loops=17)
                    -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.256 rows=1) (actual time=0.0039..0.00394 rows=1 loops=17)
 |
```

Index on Car(make,model):



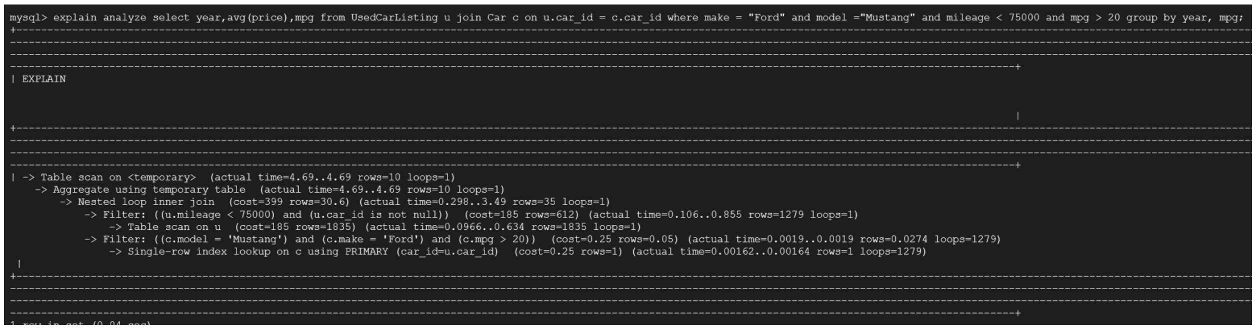
```
| -> Limit: 10 row(s)  (actual time=0.208..0.21 rows=10 loops=1)
    -> Sort: total_likes DESC, limit input to 10 row(s) per chunk  (actual time=0.207..0.208 rows=10 loops=1)
        -> Table scan on <temporary>  (actual time=0.19..0.192 rows=14 loops=1)
            -> Aggregate using temporary table  (actual time=0.189..0.189 rows=14 loops=1)
                -> Nested loop inner join  (cost=4.32 rows=2.2) (actual time=0.0591..0.141 rows=17 loops=1)
                    -> Nested loop inner join  (cost=3.55 rows=2.2) (actual time=0.0507..0.0903 rows=17 loops=1)
                        -> Filter: ((Swipe.`action` = 'LIKE') and (Swipe.listing_id is not null))  (cost=2.45 rows=2.2) (actual time=0.0346..0.0459 rows=17 loops=1)
                            -> Table scan on Swipe  (cost=2.45 rows=22) (actual time=0.0302..0.0359 rows=22 loops=1)
                        -> Filter: (UsedCarListing.car_id is not null)  (cost=0.445 rows=1) (actual time=0.00228..0.00237 rows=1 loops=17)
                            -> Single-row index lookup on UsedCarListing using PRIMARY (listing_id=Swipe.listing_id)  (cost=0.445 rows=1) (actual time=0.00213..0.00216 rows=1 loops=17)
                    -> Single-row index lookup on Car using PRIMARY (car_id=UsedCarListing.car_id)  (cost=0.295 rows=1) (actual time=0.00274..0.00278 rows=1 loops=17)
 |
```

Index	Cost
None	3.99
Car(make)	4.04
Swipe(action)	16.7
Car(make,model)	4.32

Justification: Similarly to the previous queries, we only join on already indexed attributes, so the JOIN is not affected by indexing. For the same reason as query 2 the Swipe(action) index does not perform almost any filtering while incurring additional load, and as the query is already cheap this represents a meaningful cost increase. We also aggregate to compute COUNT(*) and then sort by that aggregate, so no index can help with ORDER BY total_likes because it doesn't exist until after aggregation. For these reasons we see no reason to use an additional indexing strategy for this query.

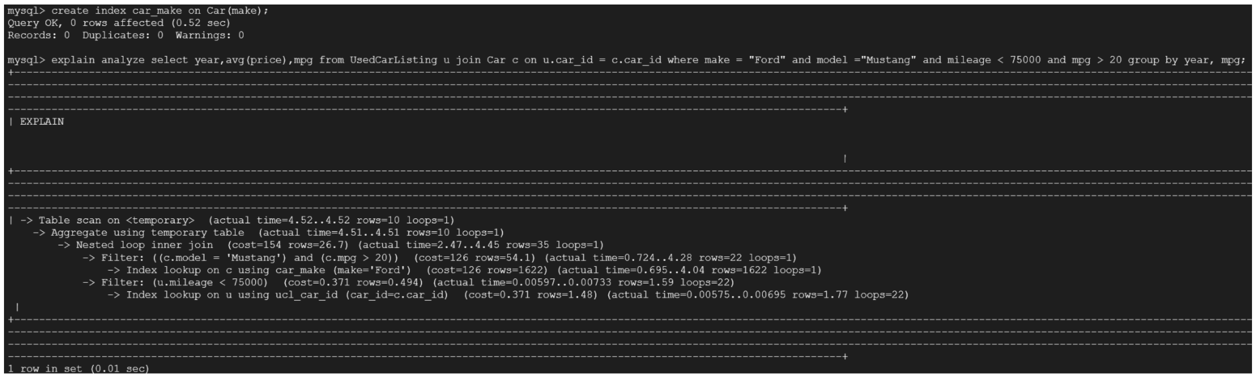
Query 4

No Index:



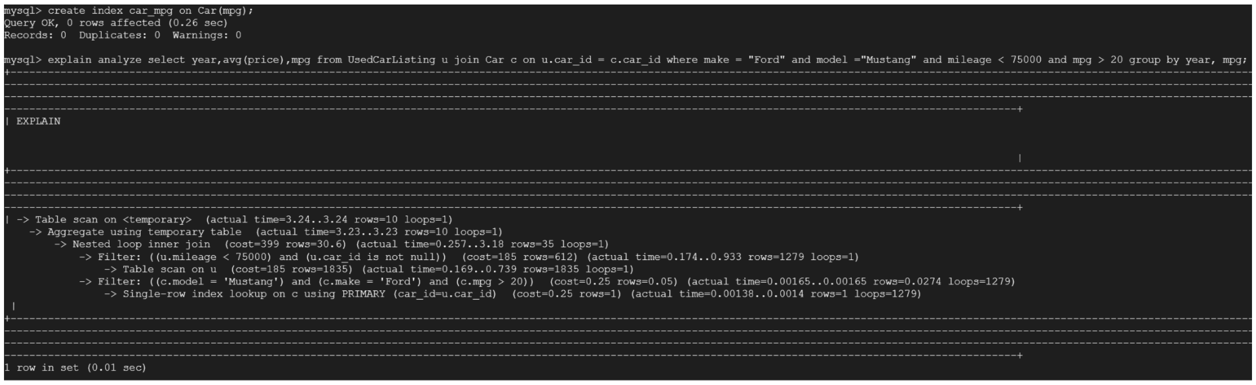
```
mysql> explain analyze select year,avg(price),mpg from UsedCarListing u join Car c on u.car_id = c.car_id where make = "Ford" and model ="Mustang" and mileage < 75000 and mpg > 20 group by year, mpg;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                                                |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=4.69..4.69 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=4.69..4.69 rows=10 loops=1)
        -> Nested loop inner join  (cost=399 rows=30.6) (actual time=0.298..3.49 rows=35 loops=1)
            -> Filter: ((u.mileage < 75000) and (u.car_id is not null))  (cost=185 rows=612) (actual time=0.106..0.855 rows=1279 loops=1)
                -> Table scan on u  (cost=185 rows=1835) (actual time=0.0966..0.634 rows=1835 loops=1)
            -> Filter: ((c.model = 'Mustang') and (c.make = 'Ford') and (c.mpg > 20))  (cost=0.25 rows=0.05) (actual time=0.0019..0.0019 rows=0.0274 loops=1279)
                -> Single-row index lookup on c using PRIMARY (car_id=u.car_id)  (cost=0.25 rows=1) (actual time=0.00162..0.00164 rows=1 loops=1279)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------+
```

Index on Car(Make):



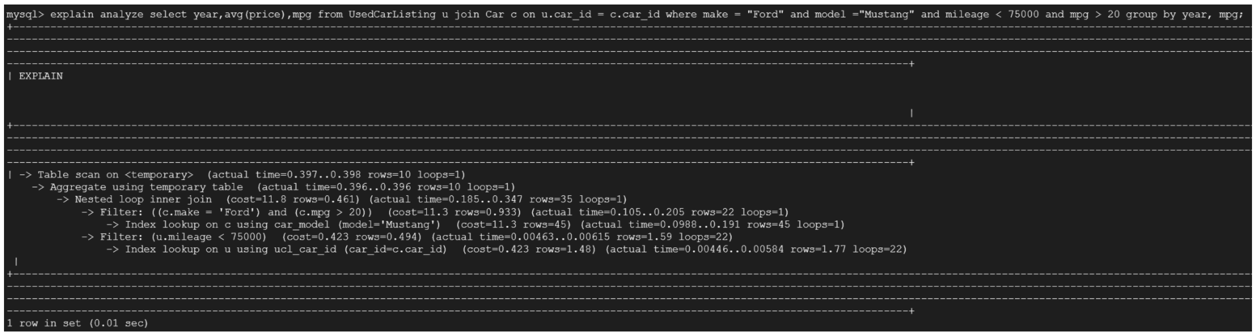
```
mysql> create index car_make on Car(make);
Query OK, 0 rows affected (0.52 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze select year,avg(price),mpg from UsedCarListing u join Car c on u.car_id = c.car_id where make = "Ford" and model ="Mustang" and mileage < 75000 and mpg > 20 group by year, mpg;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                         |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=4.52..4.52 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=4.51..4.51 rows=10 loops=1)
        -> Nested loop inner join  (cost=154 rows=26.7) (actual time=2.47..4.45 rows=35 loops=1)
            -> Filter: ((c.model = 'Mustang') and (c.mpg > 20))  (cost=126 rows=54.1) (actual time=0.724..4.28 rows=22 loops=1)
                -> Index lookup on c using car_make (make='Ford')  (cost=126 rows=1622) (actual time=0.695..4.04 rows=1622 loops=1)
            -> Filter: (u.mileage < 75000)  (cost=0.371 rows=0.494) (actual time=0.00597..0.00733 rows=1.59 loops=22)
                -> Index lookup on u using ucl_car_id (car_id=c.car_id)  (cost=0.371 rows=1.48) (actual time=0.00575..0.00695 rows=1.77 loops=22)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index on Car(mpg):



```
mysql> create index car_mpg on Car(mpg);
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze select year,avg(price),mpg from UsedCarListing u join Car c on u.car_id = c.car_id where make = "Ford" and model ="Mustang" and mileage < 75000 and mpg > 20 group by year, mpg;
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                       |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=3.24..3.24 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=3.23..3.23 rows=10 loops=1)
        -> Nested loop inner join  (cost=399 rows=30.6) (actual time=0.257..3.18 rows=35 loops=1)
            -> Filter: ((u.mileage < 75000) and (u.car_id is not null))  (cost=185 rows=612) (actual time=0.174..0.933 rows=1279 loops=1)
                -> Table scan on u  (cost=185 rows=1835) (actual time=0.169..0.739 rows=1835 loops=1)
            -> Filter: ((c.model = 'Mustang') and (c.make = 'Ford') and (c.mpg > 20))  (cost=0.25 rows=0.05) (actual time=0.00165..0.00165 rows=0.0274 loops=1279)
                -> Single-row index lookup on c using PRIMARY (car_id=u.car_id)  (cost=0.25 rows=1) (actual time=0.00138..0.0014 rows=1 loops=1279)
 |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index on Car(make, model):



```
mysql> explain analyze select year,avg(price),mpg from UsedCarListing u join Car c on u.car_id = c.car_id where make = "Ford" and model ="Mustang" and mileage < 75000 and mpg > 20 group by year, mpg;
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                             |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.397..0.398 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=0.396..0.396 rows=10 loops=1)
        -> Nested loop inner join  (cost=11.8 rows=0.461) (actual time=0.185..0.347 rows=35 loops=1)
            -> Filter: ((c.make = 'Ford') and (c.mpg > 20))  (cost=11.3 rows=0.933) (actual time=0.105..0.205 rows=22 loops=1)
                -> Index lookup on c using car_model (model='Mustang')  (cost=11.3 rows=45) (actual time=0.0988..0.191 rows=45 loops=1)
            -> Filter: (u.mileage < 75000)  (cost=0.423 rows=0.494) (actual time=0.00463..0.00615 rows=1.59 loops=22)
                -> Index lookup on u using ucl_car_id (car_id=c.car_id)  (cost=0.423 rows=1.48) (actual time=0.00446..0.00584 rows=1.77 loops=22)
 |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Index	Cost
None	399
Car(Make)	154
Car(MPG)	399
Car(Make,Model)	11.8

Justification: This query was the only one that sees meaningful performance improvements from indexing. Car(MPG) did not reduce costs as most vehicles have mpg>20, so indexing produces very little benefit. Car(Make) does improve performance by reducing the search space, but it cannot eliminate non-Mustang rows at the index step, so the gains are limited. Once we add an index for Car(Model) our performance increases dramatically. We can filter the large Car table for a small number of rows, and then join with the relatively smaller UsedCarListing table. This results in a table with much smaller cardinality, which leads to a large speedup.