Database Implementation and Indexing

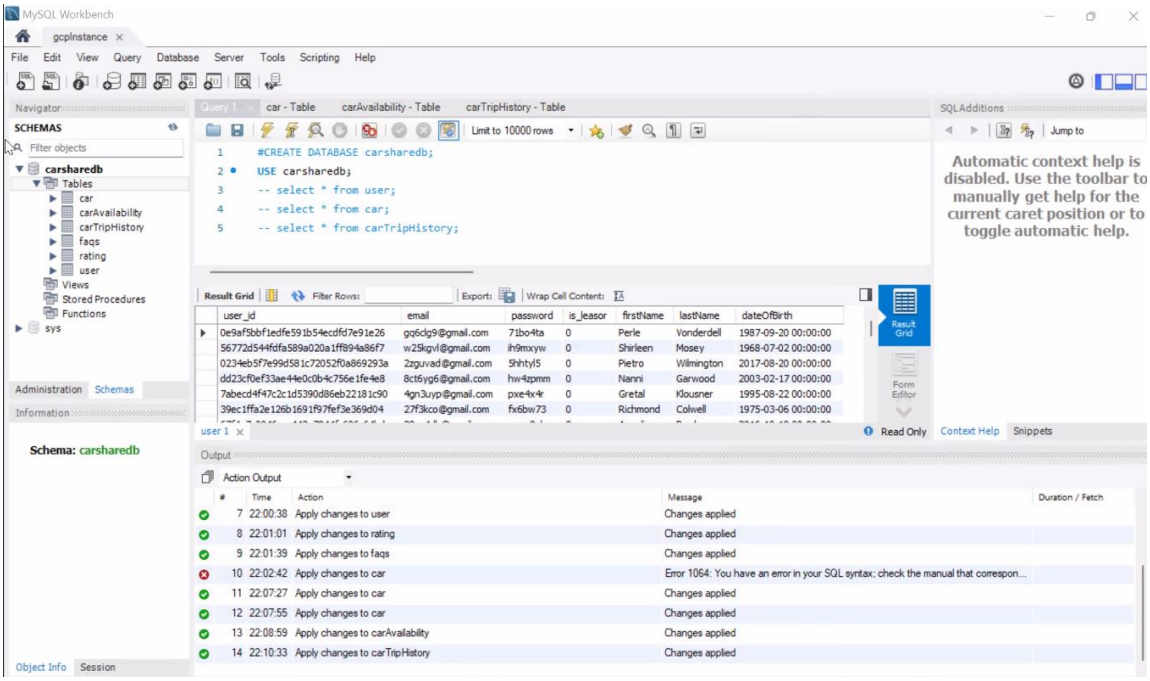
Database Implementation

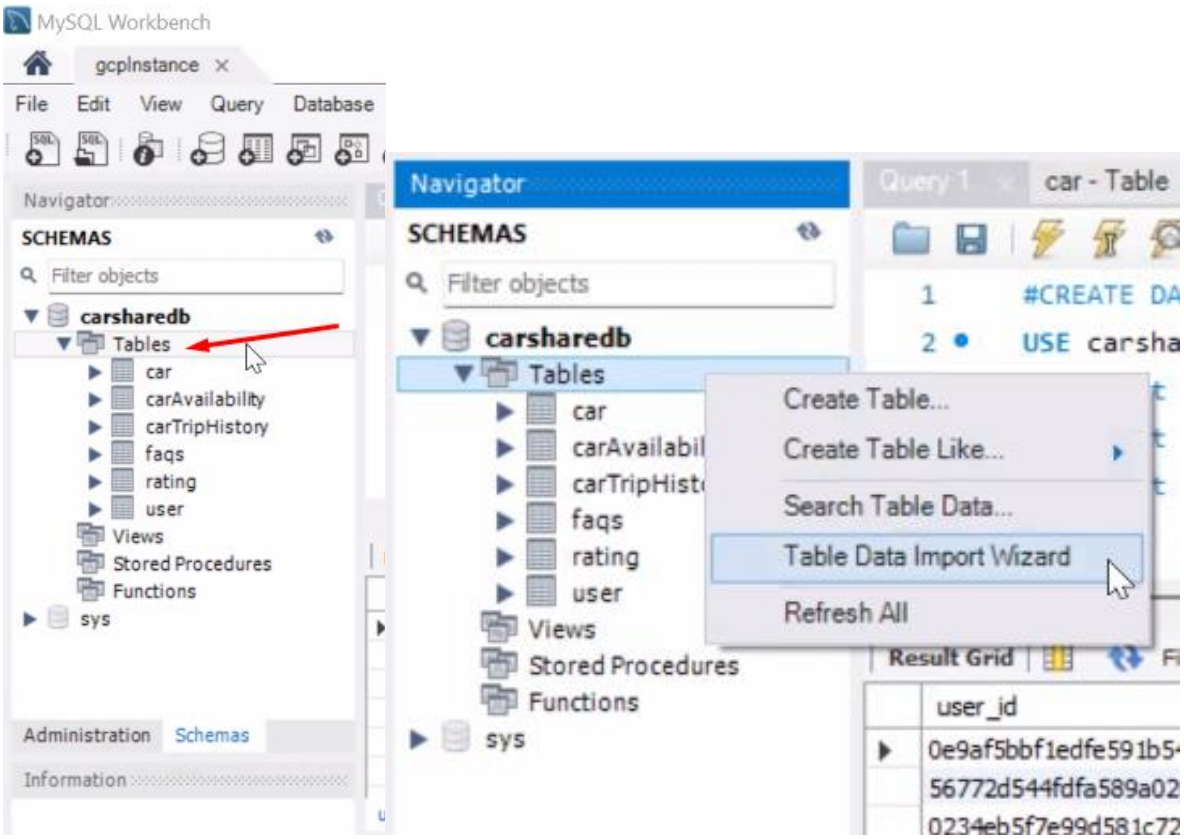
Our group used MySQL Workbench to ease the workflow of uploading our data and to make uploading quick and efficient.

The main tables which contain critical application information that we implemented are:

1. Car (Stores information such as make, model, year, VIN number)
2. CarAvailability (Stores information on when the car is available to rent)
3. CarTripHistory (Stores information including the number of trips the car has taken)
4. FAQs (Stores questions and answers for the user to view)
5. Ratings (Stores ratings for each car)

We followed MySQL workbench in order to upload our data. The instructions we used in order to upload this data are in the form of images below:




MySQL Workbench
gcpInstance
File Edit View Query Database
Navigator
SCHEMAS
Filter objects
carsharedb
Tables
car
carAvailability
carTripHistory
faqs
rating
user
Views
Stored Procedures
Functions
sys
Administration Schemas
Information
Navigator
SCHEMAS
Filter objects
carsharedb
Tables
car
faqs
rating
user
Views
Stored Procedures
Functions
sys
Create Table...
Create Table Like...
Search Table Data...
Table Data Import Wizard
Refresh All
Query 1
car - Table
1 #CREATE DA
2 USE carsha
Result Grid
user_id

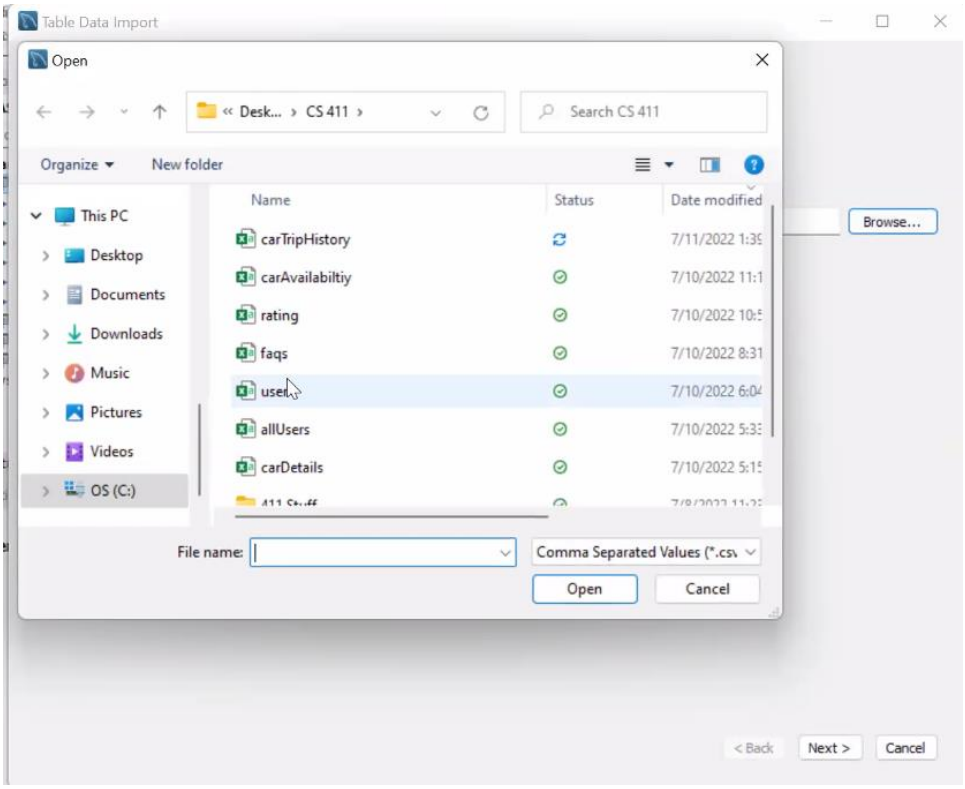


Table Data Import
Open
Desk... › CS 411 ›
Search CS 411
Organize
New folder
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
OS (C:)
Name
Status
Date modified
carTripHistory
carAvailabiltiy
rating
faqs
user
allUsers
carDetails
Browse...
File name:
Comma Separated Values (*.csv
Open
Cancel
< Back
Next >
Cancel

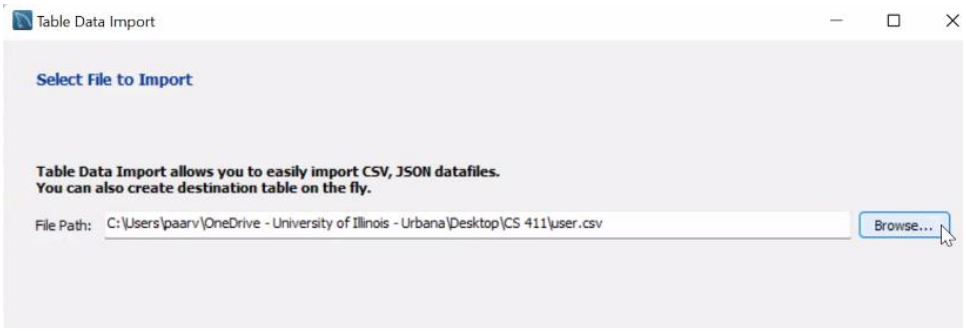


Table Data Import
Select File to Import
Table Data Import allows you to easily import CSV, JSON datafiles.
You can also create destination table on the fly.
File Path:
C:\Users\paarv\OneDrive - University of Illinois - Urbana\Desktop\CS 411\user.csv
Browse...

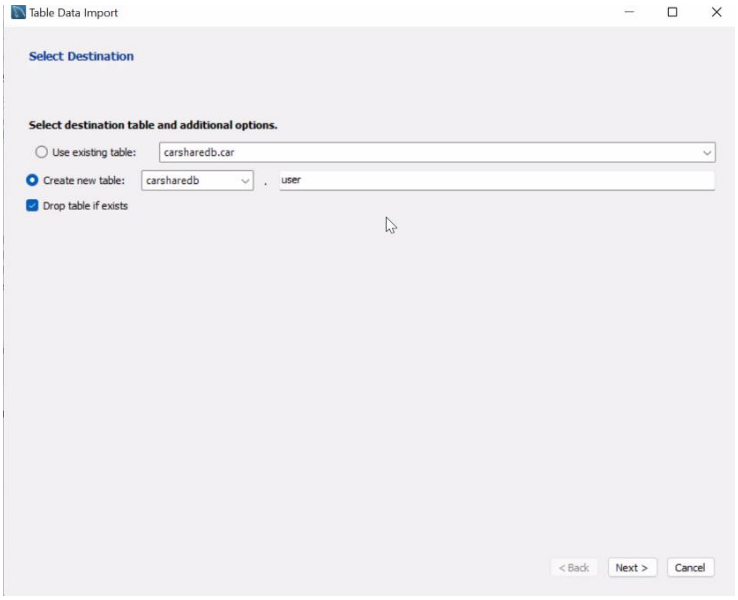


Table Data Import
Select Destination
Select destination table and additional options.
Use existing table:
carsharedb.car
Create new table:
carsharedb
user
Drop table if exists
< Back
Next >
Cancel

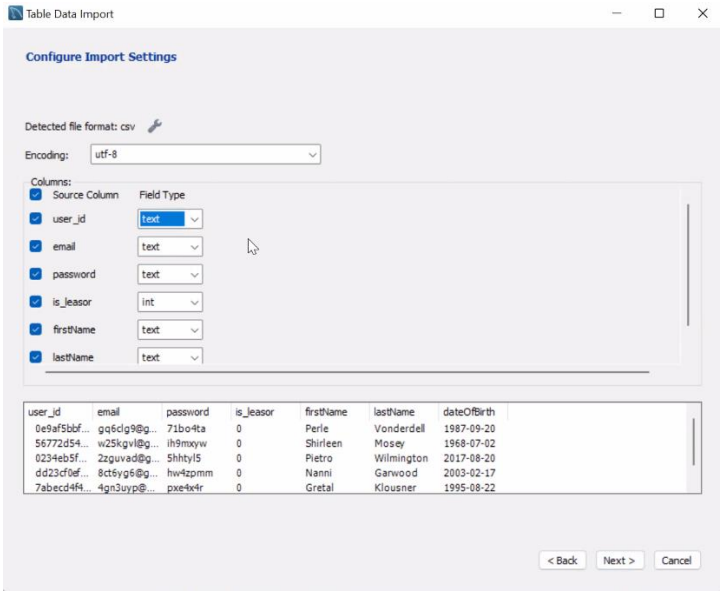
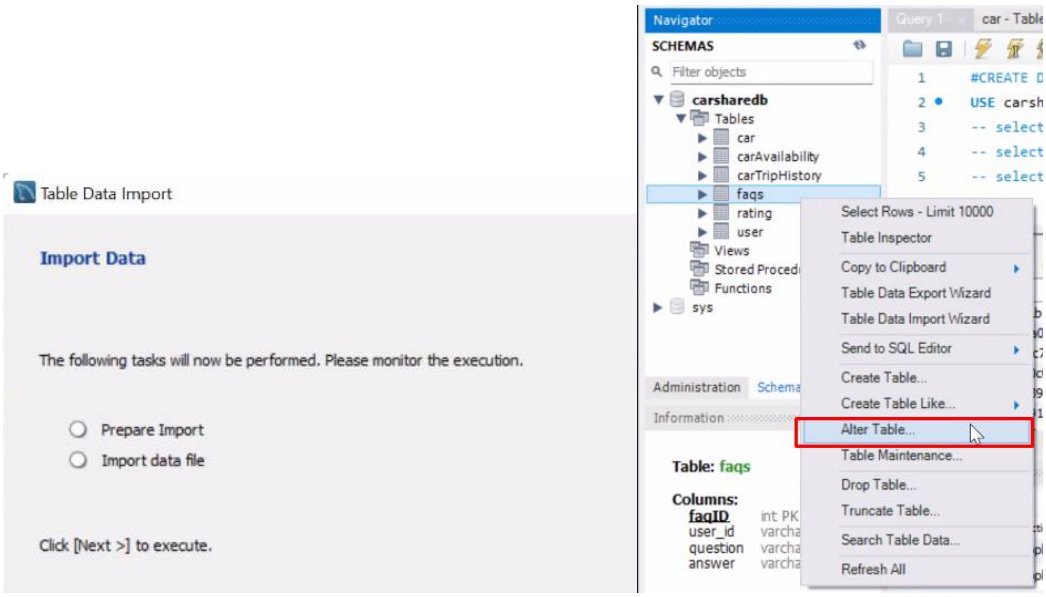
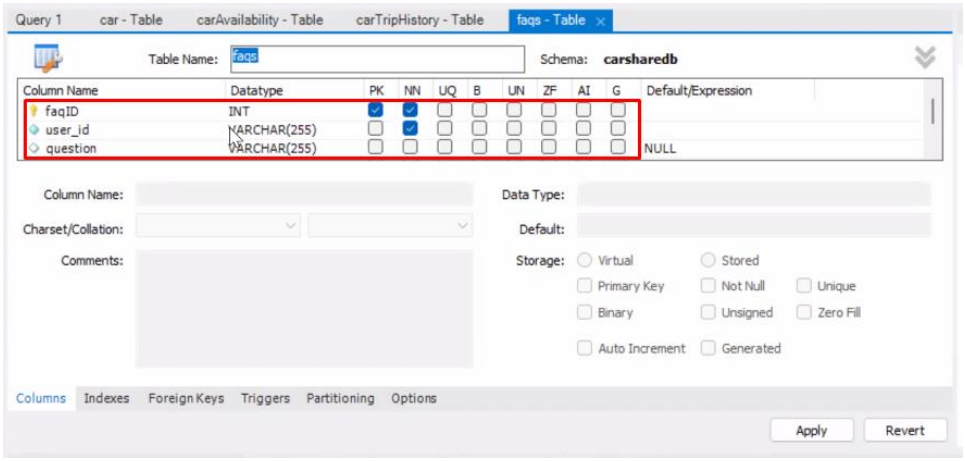


Table Data Import
Configure Import Settings
Detected file format: csv
Encoding:
utf-8
Columns:
Source Column
Field Type
user_id
text
email
text
password
text
is_leasor
int
firstName
text
lastName
text
user_id email password is_leasor firstName lastName dateOfBirth
0e9af5bbf... gq6clg9@g... 71bo4ta 0 Perle Vonderdell 1987-09-20
56772d54... w25kgvl@g... ih9mxyw 0 Shirleen Mosey 1968-07-02
0234eb5f... 2zguvad@g... 5hhtyl5 0 Pietro Wilmington 2017-08-20
dd23cf0ef... 8ct6yg6@g... hw4zpmm 0 Nanni Garwood 2003-02-17
7abecd4f4... 4gn3uyp@... pxe4x4r 0 Gretal Klousner 1995-08-22
< Back
Next >
Cancel





The **SQL DDL** commands we would use otherwise to create our tables would be as follows:

CREATE TABLE Car(car_id VARCHAR(8), car_lat DECIMAL(3,10), car_long DECIMAL(3,10), make VARCHAR(255), model VARCHAR(255), year INTEGER(4), vinVARCHAR(255));

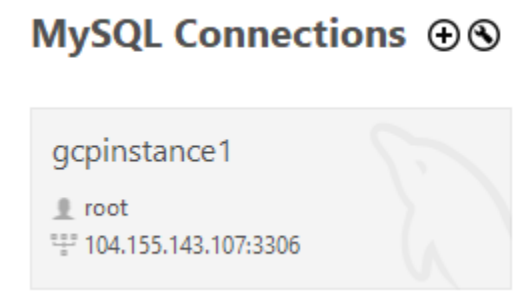
CREATE TABLE CarTripHistory(journey_id VARCHAR(255), lesser_id VARCHAR(255), car_id VARCHAR(255), pickup_datetime DATETIME, dropoff_datetime DATETIME, price INTEGER(8), rating INTEGER(8));

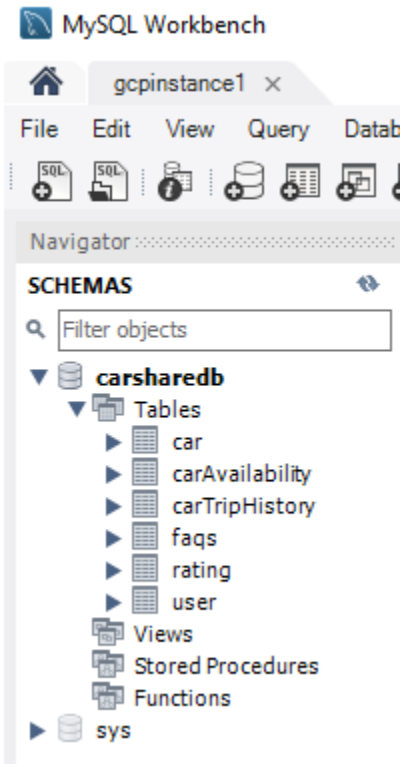
CREATE TABLE Carvailability(availability_id VARCHAR(255), car_id VARCHAR(255), available_from DATETIME, available_till DATETIME);

CREATE TABLE FAQs(faq_ID VARCHAR(255), user_id VARCHAR(255), question VARCHAR(255), answer VARCHAR(255));

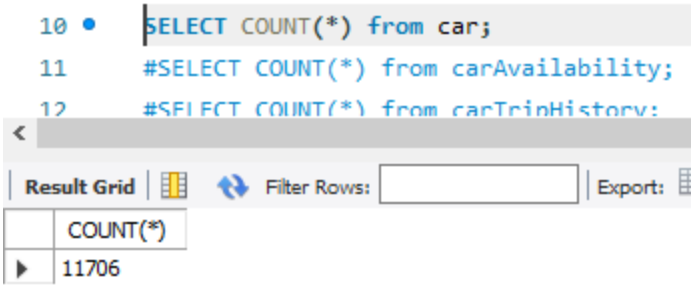
CREATE TABLE Rating(rating INTEGER(8), message VARCHAR(255));

Screenshot of connection to MySQL workbench/Google GCP: We created a google GCP instance and then we connected to it through MySQL workbench. We then created a database called “carshareddb” to store all our data.

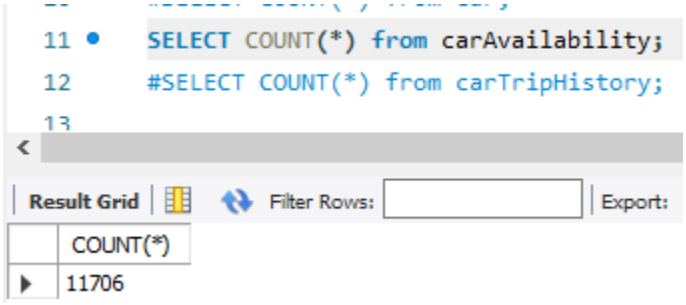




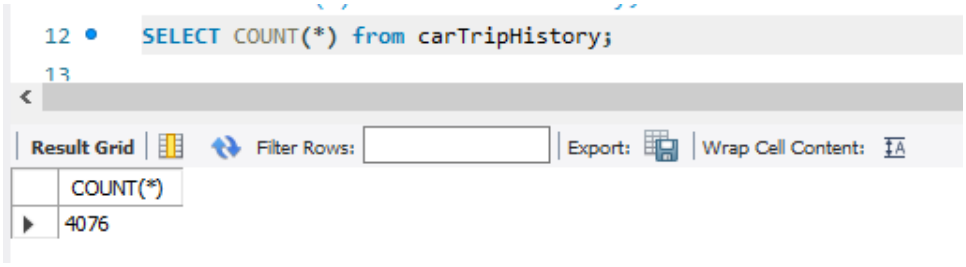
Car Table Count



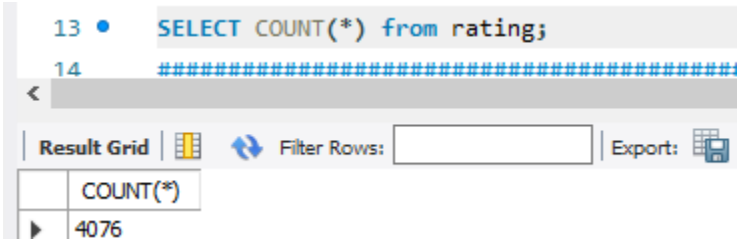
Car Availability Count



CarTripHistory Table Count



Rating Table Count

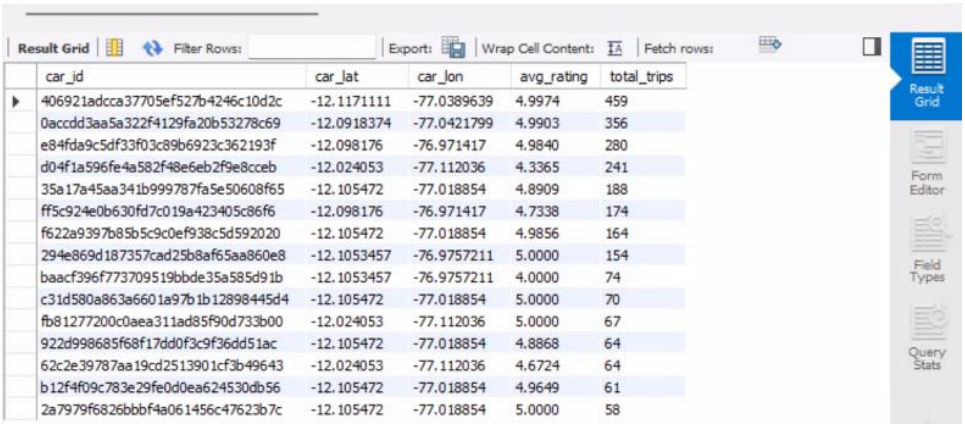


Two Advanced SQL Queries:

Query 1: (Car location details with > 4.0 rating and > 50 trips)

```sql
# Finding car location details of those car which has more than 4.0 average user ratings
#  and had more than 50 trips
USE carsharedb;
SELECT
    car.car_id,
    car_lat,
    car_lon,
    AVG(rating) AS avg_rating,
    COUNT(*) AS total_trips
FROM car
JOIN carTripHistory ON car.car_id = carTripHistory.car_id
GROUP BY car.car_id
HAVING avg_rating >=4 AND total_trips >=50
ORDER BY total_trips DESC, avg_rating DESC
LIMIT 15;
```

Top 15 Results:



Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

car_id	car_lat	car_lon	avg_rating	total_trips
406921adcca37705ef527b4246c10d2c	-12.1171111	-77.0389639	4.9974	459
0accdd3aa5a322f4129fa20b53278c69	-12.0918374	-77.0421799	4.9903	356
e84fda9c5df33f03c89b6923c362193f	-12.098176	-76.971417	4.9840	280
d04f1a596fe4a582f48e6eb2f9e8cceb	-12.024053	-77.112036	4.3365	241
35a17a45aa341b999787fa5e50608f65	-12.105472	-77.018854	4.8909	188
ff5c924e0b630fd7c019a423405c86f6	-12.098176	-76.971417	4.7338	174
f622a9397b85b5c9c0ef938c5d592020	-12.105472	-77.018854	4.9856	164
294e869d187357cad25b8af65aa860e8	-12.1053457	-76.9757211	5.0000	154
baacf396f773709519bbde35a585d91b	-12.1053457	-76.9757211	4.0000	74
c31d580a863a6601a97b1b12898445d4	-12.105472	-77.018854	5.0000	70
fb81277200c0aea311ad85f90d733b00	-12.024053	-77.112036	5.0000	67
922d998685f68f17dd0f3c9f36dd51ac	-12.105472	-77.018854	4.8868	64
62c2e39787aa19cd2513901cf3b49643	-12.024053	-77.112036	4.6724	64
b12f4f09c783e29fe0d0ea624530db56	-12.105472	-77.018854	4.9649	61
2a7979f6826bbbf4a061456c47623b7c	-12.105472	-77.018854	5.0000	58

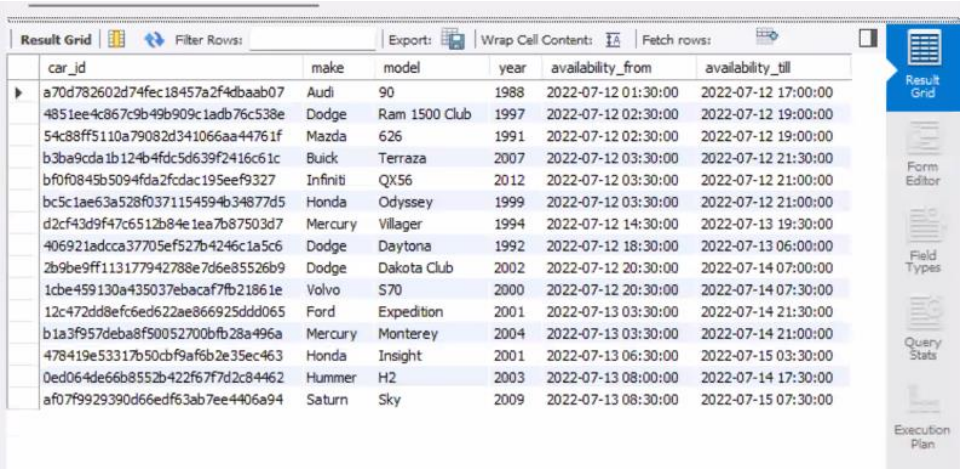
Result Grid | Form Editor | Field Types | Query Stats

Justification: This query will be a critical part of our website. What the query does is it returns important car information (such as its ID and location) for cars that meet certain requirements. These requirements are cars that have a rating that is greater than 4.0 and that have more than 50 trips. Our criteria for the thresholds are tailored so that the user has an easier time searching for cars that are reliable and that have a high rating. We decided that cars that have 50 trips or more while still holding an average rating that is greater than 4.0 would be the cars that most users would want to rent, so this query will assist in the user's search.

Query 2: (Show car availability of cars without any trip history)

```sql
# Finding out available cars in user defined timeframe
# which did not have any trip history (unused/new cars)
USE carsharedb;
SELECT
    car.car_id, make, model, year, availability_from, availability_till
FROM carAvailability
JOIN car ON car.car_id = carAvailability.car_id
WHERE availability_from >= CURDATE() AND availability_till <=ADDDATE(CURDATE(), 5)
 AND car.car_id NOT IN (SELECT DISTINCT car_id FROM carTripHistory)
ORDER BY availability_from ASC
LIMIT 15;
```

Top 15 Results:



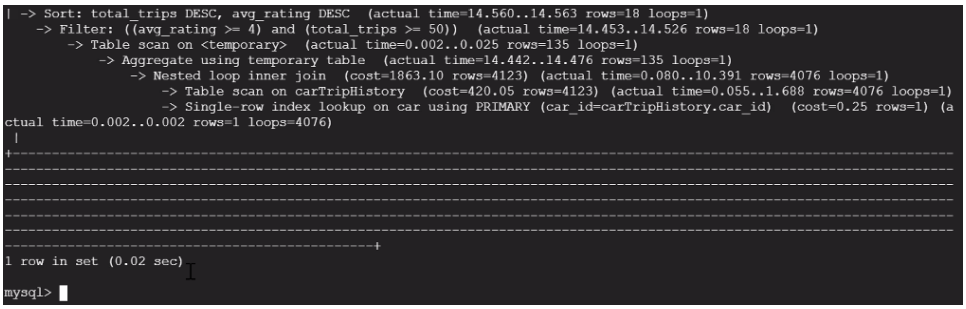
car_id	make	model	year	availability_from	availability_till
a70d782602d74fec18457a2f4dbaab07	Audi	90	1988	2022-07-12 01:30:00	2022-07-12 17:00:00
4851ee4c867c9b49b909c1adb76c538e	Dodge	Ram 1500 Club	1997	2022-07-12 02:30:00	2022-07-12 19:00:00
54c88ff5110a79082d341066aa44761f	Mazda	626	1991	2022-07-12 02:30:00	2022-07-12 19:00:00
b3ba9cda1b124b4fdc5d639f2416c61c	Buick	Terraza	2007	2022-07-12 03:30:00	2022-07-12 21:30:00
bf0f0845b5094fda2fcdac195eef9327	Infiniti	QX56	2012	2022-07-12 03:30:00	2022-07-12 21:00:00
bc5c1ae63a528f0371154594b34877d5	Honda	Odyssey	1999	2022-07-12 03:30:00	2022-07-12 21:00:00
d2cf43d9f47c6512b84e1ea7b87503d7	Mercury	Villager	1994	2022-07-12 14:30:00	2022-07-13 19:30:00
406921adcca37705ef527b4246c1a5c6	Dodge	Daytona	1992	2022-07-12 18:30:00	2022-07-13 06:00:00
2b9be9ff113177942788e7d6e85526b9	Dodge	Dakota Club	2002	2022-07-12 20:30:00	2022-07-14 07:00:00
1cbe459130a435037ebacaf7fb21861e	Volvo	S70	2000	2022-07-12 20:30:00	2022-07-14 07:30:00
12c472dd8efc6ed622ae866925ddd065	Ford	Expedition	2001	2022-07-13 03:30:00	2022-07-14 21:30:00
b1a3f957deba8f50052700bfb28a496a	Mercury	Monterey	2004	2022-07-13 03:30:00	2022-07-14 21:00:00
478419e53317b50cbf9af6b2e35ec463	Honda	Insight	2001	2022-07-13 06:30:00	2022-07-15 03:30:00
0ed064de66b8552b422f67f7d2c84462	Hummer	H2	2003	2022-07-13 08:00:00	2022-07-14 17:30:00
af07f9929390d66edf63ab7ee4406a94	Saturn	Sky	2009	2022-07-13 08:30:00	2022-07-15 07:30:00

Justification: This query is similar to the previous query, but it returns cars without any trip history associated with it in a certain timeframe. This will be another important piece on our website that will be used extensively if the user wants to rent a car that has not been used before in terms of renting.

Indexing Analysis:

Query 2:

- Default Index: car_id, availability _id
    - Time to execute: 0.02 sec
    - Query Cost:



```
| -> Sort: total_trips DESC, avg_rating DESC  (actual time=14.560..14.563 rows=18 loops=1)
    -> Filter: ((avg_rating >= 4) and (total_trips >= 50))  (actual time=14.453..14.526 rows=18 loops=1)
        -> Table scan on <temporary>  (actual time=0.002..0.025 rows=135 loops=1)
            -> Aggregate using temporary table  (actual time=14.442..14.476 rows=135 loops=1)
                -> Nested loop inner join  (cost=1863.10 rows=4123) (actual time=0.080..10.391 rows=4076 loops=1)
                    -> Table scan on carTripHistory  (cost=420.05 rows=4123) (actual time=0.055..1.688 rows=4076 loops=1)
                    -> Single-row index lookup on car using PRIMARY (car_id=carTripHistory.car_id)  (cost=0.25 rows=1) (a
ctual time=0.002..0.002 rows=1 loops=4076)
 |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
1 row in set (0.02 sec)

mysql>
```

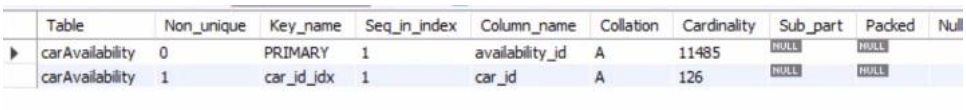
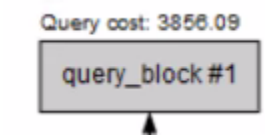


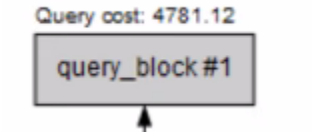
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null
carAvailability	0	PRIMARY	1	availability_id	A	11485	NULL	NULL	
carAvailability	1	car_id_idx	1	car_id	A	126	NULL	NULL	



- Index Design #1: car_id, availability_id, availability _from
    - Time to execute: 0.01 sec

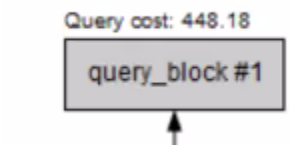
```
           -> Filter: (carAvailability.availability_from >= <cache>(curdate()))  (cost=87.56 rows=194) (actual time=8.10
7..10.800 rows=129 loops=1)
               -> Index range scan on carAvailability using idx_avail_till, with index condition: (carAvailability.avail
ability_till <= <cache>((curdate() + interval 5 day)))  (cost=87.56 rows=194) (actual time=8.076..10.762 rows=194 loops=1
)
       -> Single-row index lookup on car using PRIMARY (car_id=carAvailability.car_id)  (cost=0.25 rows=1) (actual time=
0.003..0.003 rows=1 loops=129)
   -> Index lookup on carTripHistory using fk_car_idx (car_id=carAvailability.car_id)  (cost=36.79 rows=31) (actual time
=0.011..0.011 rows=1 loops=129)
 |
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------+
1 row in set (0.01 sec)
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
carAvailability	0	PRIMARY	1	availability_id	A	11485	NULL	NULL
carAvailability	1	car_id_idx	1	car_id	A	126	NULL	NULL
carAvailability	1	idx_avail_from	1	availability_from	A	8393	NULL	NULL



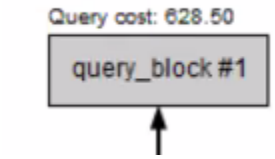
- Index Design #2: car_id, availability_id, availability_till

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Nu
carAvailability	0	PRIMARY	1	availability_id	A	11485	NULL	NULL	
carAvailability	1	car_id_idx	1	car_id	A	126	NULL	NULL	
carAvailability	1	idx_avail_till	1	availability_till	A	8394	NULL	NULL	



- Index Design #3: car_id, availability_id, availability _From availability_till

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
carAvailability	0	PRIMARY	1	availability_id	A	11485	NULL	NULL
carAvailability	1	car_id_idx	1	car_id	A	126	NULL	NULL
carAvailability	1	idx_avail_from	1	availability_from	A	8393	NULL	NULL
carAvailability	1	idx_avail_till	1	availability_till	A	8394	NULL	NULL



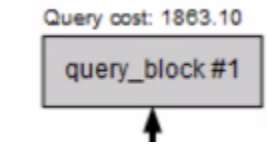
Justification: For this query, when we ran the EXPLAIN ANALYZE command to see execution times, we did not see any significant differences between the times. Sometimes there was only a difference of 0.01 seconds between the different index designs that we chose. Moving forward, we decided to use the visual tool in order to compare query costs. We found that by using the indexes “car_id,

availability_id, availability_till" we got the lowest query cost. Therefore, we will be using this indexing design in our implementation for the next stages.

Query 1:

- Default:

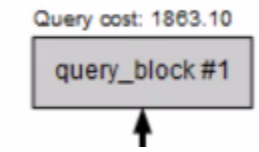
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
carTripHistory	0	PRIMARY	1	journey_id	A	4123	NULL	NULL
carTripHistory	1	fk_lesser_idx	1	lesser_id	A	742	NULL	NULL
carTripHistory	1	fk_car_idx	1	car_id	A	135	NULL	NULL



- Index Design #1: card_id, owner_id, VIN

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

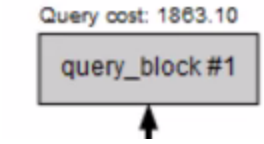
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
car	0	PRIMARY	1	car_id	A	213	NULL	NULL
car	1	fk_owner_idx	1	owner_id	A	168	NULL	NULL
car	1	idx_vin	1	VIN	A	213	NULL	NULL



- Index Design #2:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

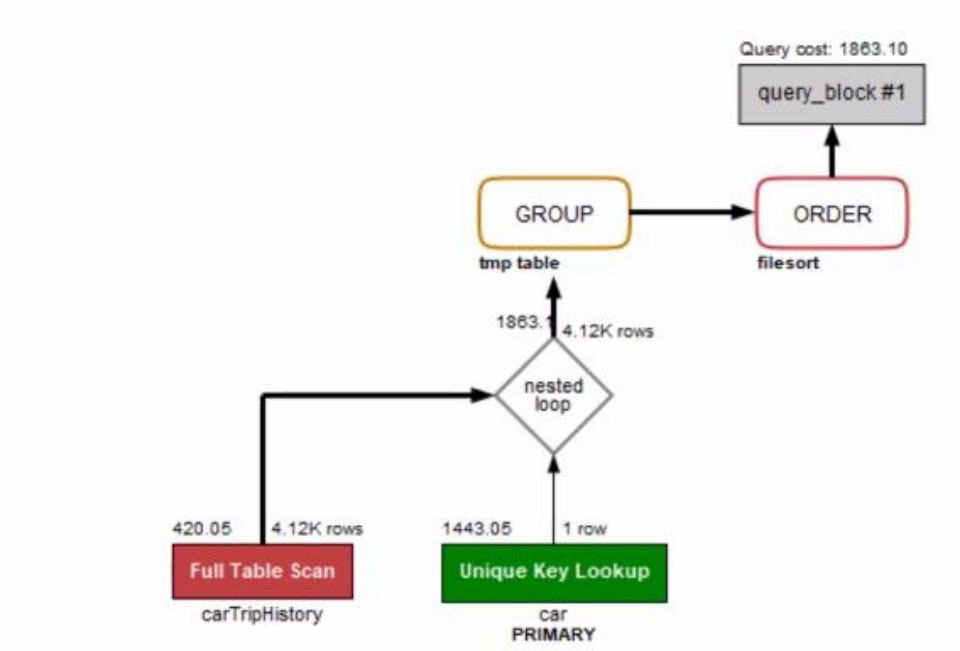
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
carTripHistory	0	PRIMARY	1	journey_id	A	4123	NULL	NULL
carTripHistory	1	fk_lesser_idx	1	lesser_id	A	742	NULL	NULL
carTripHistory	1	fk_car_idx	1	car_id	A	135	NULL	NULL
carTripHistory	1	idx_pickup	1	pickup_datetime	A	4002	NULL	NULL



- Index Design #3:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
carTripHistory	0	PRIMARY	1	journey_id	A	4123	NULL	NULL
carTripHistory	1	fk_lesser_idx	1	lesser_id	A	742	NULL	NULL
carTripHistory	1	fk_car_idx	1	car_id	A	135	NULL	NULL
carTripHistory	1	idx_dropoff	1	dropoff_datetime	A	4048	NULL	NULL



Justification: For this query, when we ran the EXPLAIN ANALYZE command to see execution times, we did not see any significant differences between the times, nor did we see any differences between the query costs. This behavior persisted despite trying numerous different indexing designs. We provided a visual explain result as context. One possible explanation for this behavior is that our query already runs in the most optimal manner due to our primary key choice (journey_id).