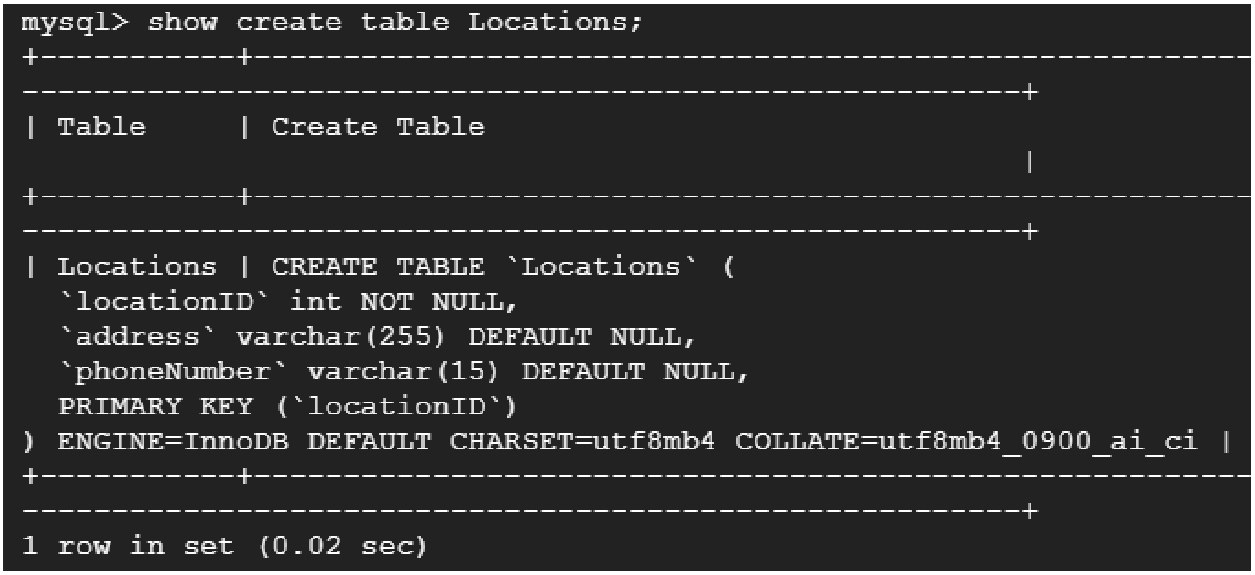
Data Definition Language (DDL)

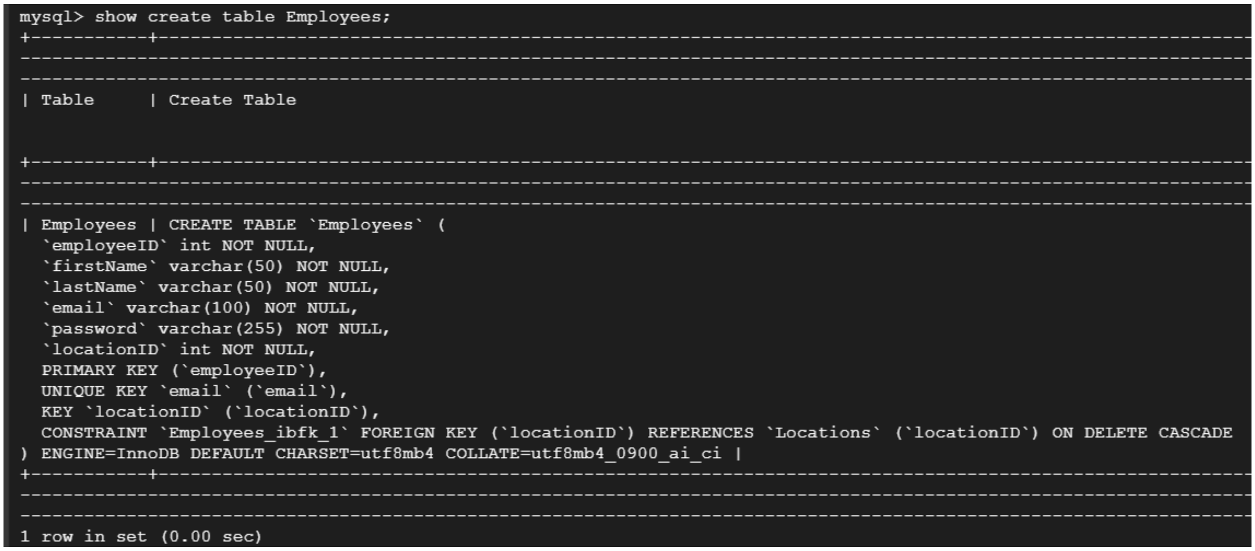
```
CREATE TABLE Locations (
        locationID INT PRIMARY KEY,
        address VARCHAR(255),
        phoneNumber VARCHAR(15)
);

CREATE TABLE Employees (
        employeeID INT PRIMARY KEY,
        firstName VARCHAR(50) NOT NULL,
        lastName VARCHAR(50) NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL,
        password VARCHAR(255) NOT NULL,
        locationID INT NOT NULL,
        FOREIGN KEY (locationID) REFERENCES Locations(locationID) ON DELETE
CASCADE
);

CREATE TABLE Details (
        make VARCHAR(50) NOT NULL,
        model VARCHAR(50) NOT NULL,
        year INT NOT NULL,
        numberOfCylinders INT,
        transmission VARCHAR(50),
        driveWheel VARCHAR(50),
        PRIMARY KEY (make, model, year)
);
```

```
CREATE TABLE Cars (
        VIN VARCHAR(17) PRIMARY KEY,
        color VARCHAR(30),
        price DECIMAL(10, 2),
        mileage INT,
        status VARCHAR(50),
        make VARCHAR(50) NOT NULL,
        model VARCHAR(50) NOT NULL,
        year INT NOT NULL,
        locationID INT NOT NULL,
        lastModifiedBy INT NOT NULL,
        warrantyID INT,
        FOREIGN KEY (make, model, year) REFERENCES Details(make, model, year) ON
DELETE CASCADE,
        FOREIGN KEY (locationID) REFERENCES Locations(locationID) ON DELETE
CASCADE,
        FOREIGN KEY (lastModifiedBy) REFERENCES Employees(employeeID) ON DELETE
CASCADE,
        FOREIGN KEY (warrantyID) REFERENCES Warranties(warrantyID) ON DELETE SET
NULL
);

CREATE TABLE Warranties (
        warrantyID INT PRIMARY KEY,
        startDate DATE NOT NULL,
        endDate DATE NOT NULL,
        coverageDetail VARCHAR(1000),
        VIN VARCHAR(17) NOT NULL,
        reviewID INT NOT NULL,
        FOREIGN KEY (VIN) REFERENCES Cars(VIN) ON DELETE CASCADE,
        FOREIGN KEY (reviewID) REFERENCES Reviews(reviewID) ON DELETE CASCADE
);

CREATE TABLE Reviews (
        reviewID INT PRIMARY KEY,
        rating INT,
        comment VARCHAR(1000),
        make VARCHAR(50) NOT NULL,
        model VARCHAR(50) NOT NULL,
        year INT NOT NULL,
        FOREIGN KEY (make, model, year) REFERENCES Details(make, model, year) ON
DELETE CASCADE
);
```

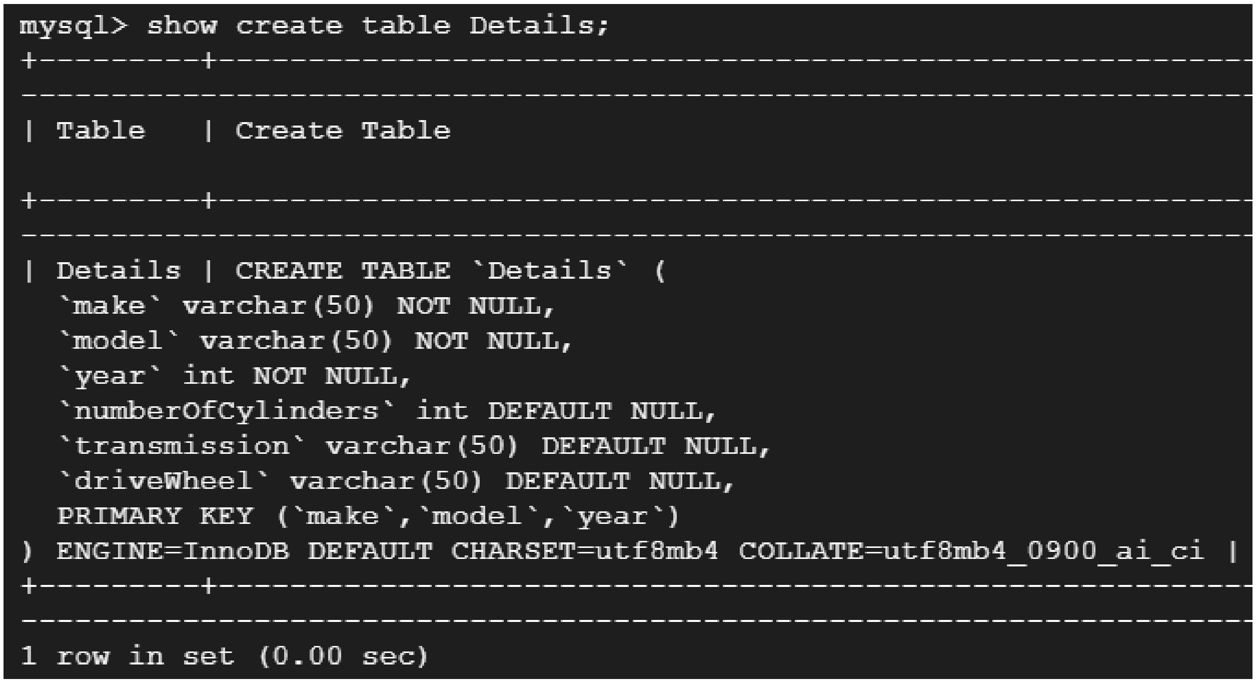
Proof for creating tables



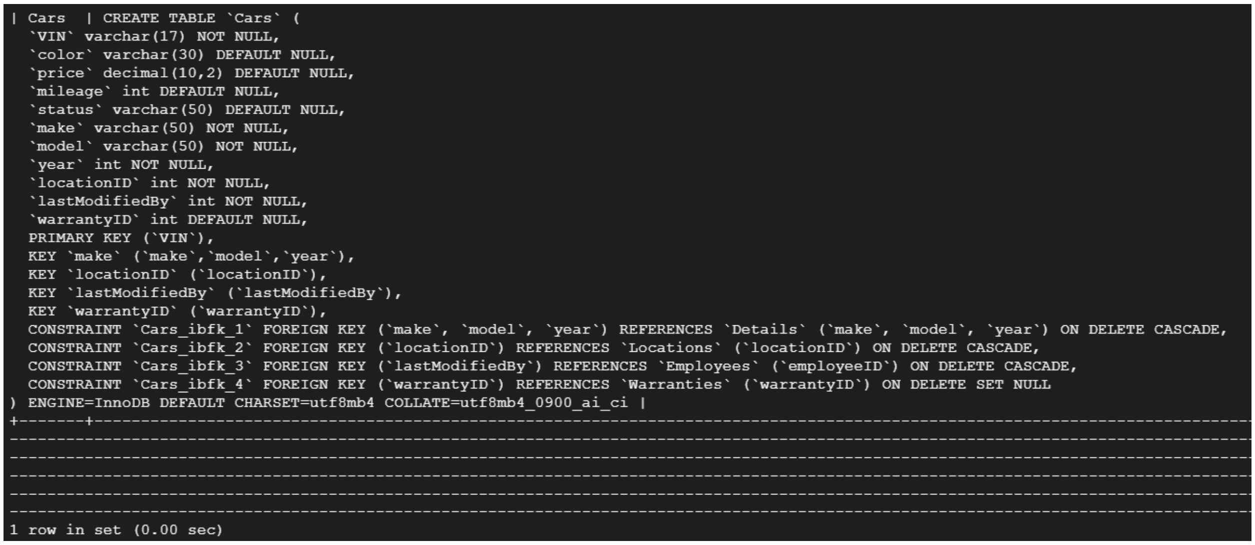
```
mysql> show create table Locations;
+-----------+--------------------------------------------------------------
-----------------------------------------------------------------+
| Table     | Create Table
                                                                 |
+-----------+--------------------------------------------------------------
-----------------------------------------------------------------+
| Locations | CREATE TABLE `Locations` (
  `locationID` int NOT NULL,
  `address` varchar(255) DEFAULT NULL,
  `phoneNumber` varchar(15) DEFAULT NULL,
  PRIMARY KEY (`locationID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-----------+--------------------------------------------------------------
-----------------------------------------------------------------+
1 row in set (0.02 sec)
```



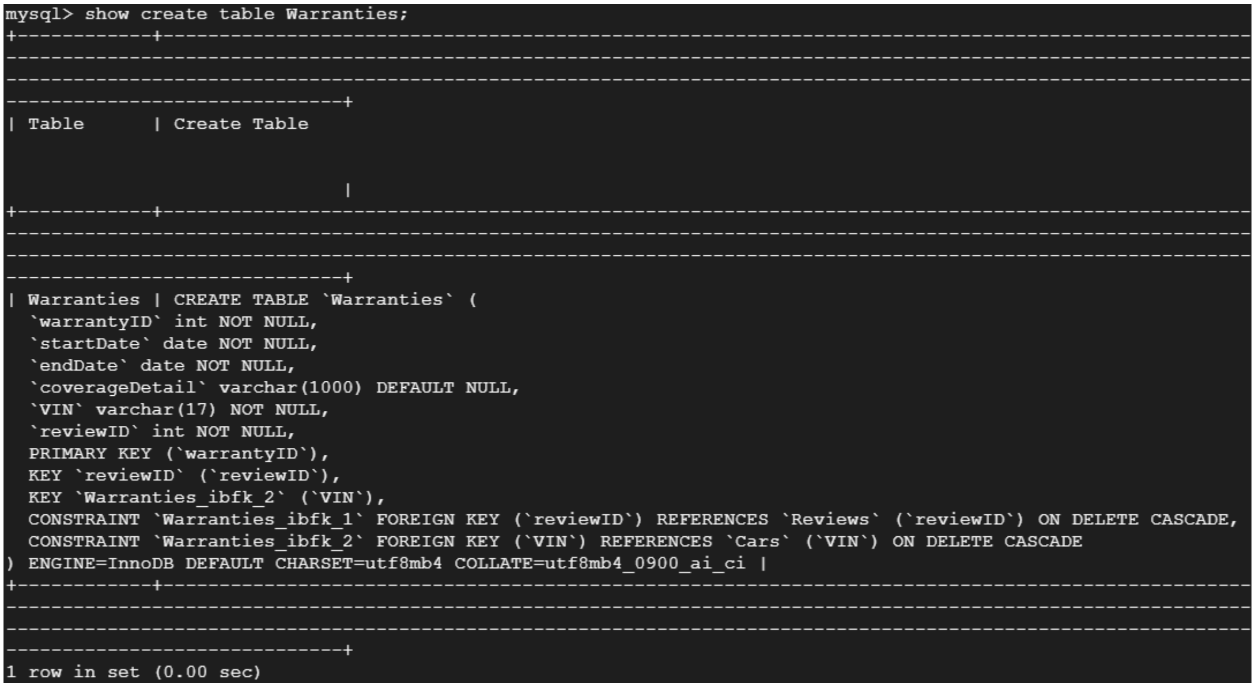
```
mysql> show create table Employees;
+-----------+---------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
| Table     | Create Table


+-----------+---------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
| Employees | CREATE TABLE `Employees` (
  `employeeID` int NOT NULL,
  `firstName` varchar(50) NOT NULL,
  `lastName` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  `locationID` int NOT NULL,
  PRIMARY KEY (`employeeID`),
  UNIQUE KEY `email` (`email`),
  KEY `locationID` (`locationID`),
  CONSTRAINT `Employees_ibfk_1` FOREIGN KEY (`locationID`) REFERENCES `Locations` (`locationID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-----------+---------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```



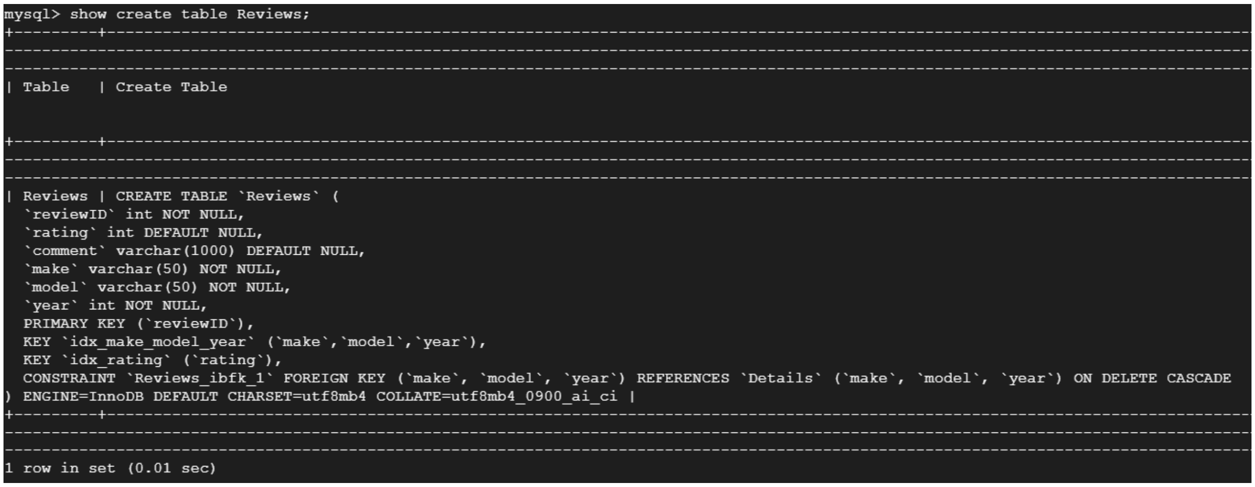
```
mysql> show create table Details;
+---------+---------------------------------------------------------------
----------------------------------------------------------------------------
| Table   | Create Table

+---------+---------------------------------------------------------------
----------------------------------------------------------------------------
| Details | CREATE TABLE `Details` (
  `make` varchar(50) NOT NULL,
  `model` varchar(50) NOT NULL,
  `year` int NOT NULL,
  `numberOfCylinders` int DEFAULT NULL,
  `transmission` varchar(50) DEFAULT NULL,
  `driveWheel` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`make`,`model`,`year`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------+---------------------------------------------------------------
----------------------------------------------------------------------------
1 row in set (0.00 sec)
```



```
| Cars  | CREATE TABLE `Cars` (
  `VIN` varchar(17) NOT NULL,
  `color` varchar(30) DEFAULT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `mileage` int DEFAULT NULL,
  `status` varchar(50) DEFAULT NULL,
  `make` varchar(50) NOT NULL,
  `model` varchar(50) NOT NULL,
  `year` int NOT NULL,
  `locationID` int NOT NULL,
  `lastModifiedBy` int NOT NULL,
  `warrantyID` int DEFAULT NULL,
  PRIMARY KEY (`VIN`),
  KEY `make` (`make`,`model`,`year`),
  KEY `locationID` (`locationID`),
  KEY `lastModifiedBy` (`lastModifiedBy`),
  KEY `warrantyID` (`warrantyID`),
  CONSTRAINT `Cars_ibfk_1` FOREIGN KEY (`make`, `model`, `year`) REFERENCES `Details` (`make`, `model`, `year`) ON DELETE CASCADE,
  CONSTRAINT `Cars_ibfk_2` FOREIGN KEY (`locationID`) REFERENCES `Locations` (`locationID`) ON DELETE CASCADE,
  CONSTRAINT `Cars_ibfk_3` FOREIGN KEY (`lastModifiedBy`) REFERENCES `Employees` (`employeeID`) ON DELETE CASCADE,
  CONSTRAINT `Cars_ibfk_4` FOREIGN KEY (`warrantyID`) REFERENCES `Warranties` (`warrantyID`) ON DELETE SET NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

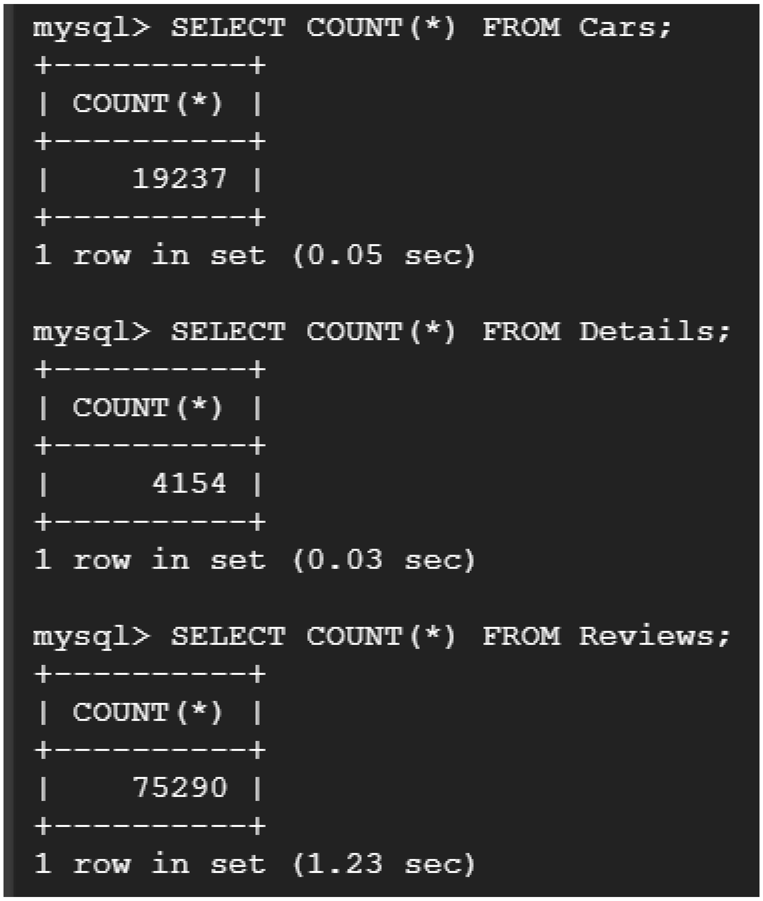


```
mysql> show create table Warranties;
+------------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------------------------+
| Table      | Create Table


                              |
+------------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------------------------+
| Warranties | CREATE TABLE `Warranties` (
  `warrantyID` int NOT NULL,
  `startDate` date NOT NULL,
  `endDate` date NOT NULL,
  `coverageDetail` varchar(1000) DEFAULT NULL,
  `VIN` varchar(17) NOT NULL,
  `reviewID` int NOT NULL,
  PRIMARY KEY (`warrantyID`),
  KEY `reviewID` (`reviewID`),
  KEY `Warranties_ibfk_2` (`VIN`),
  CONSTRAINT `Warranties_ibfk_1` FOREIGN KEY (`reviewID`) REFERENCES `Reviews` (`reviewID`) ON DELETE CASCADE,
  CONSTRAINT `Warranties_ibfk_2` FOREIGN KEY (`VIN`) REFERENCES `Cars` (`VIN`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+------------+-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
------------------------------+
1 row in set (0.00 sec)
```



```
mysql> show create table Reviews;
+---------+-----------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
| Table   | Create Table


+---------+-----------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
| Reviews | CREATE TABLE `Reviews` (
  `reviewID` int NOT NULL,
  `rating` int DEFAULT NULL,
  `comment` varchar(1000) DEFAULT NULL,
  `make` varchar(50) NOT NULL,
  `model` varchar(50) NOT NULL,
  `year` int NOT NULL,
  PRIMARY KEY (`reviewID`),
  KEY `idx_make_model_year` (`make`,`model`,`year`),
  KEY `idx_rating` (`rating`),
  CONSTRAINT `Reviews_ibfk_1` FOREIGN KEY (`make`, `model`, `year`) REFERENCES `Details` (`make`, `model`, `year`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------+-----------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Proof for inserting at least 1000 rows in the three tables



```
mysql> SELECT COUNT(*) FROM Cars;
+----------+
| COUNT(*) |
+----------+
|    19237 |
+----------+
1 row in set (0.05 sec)

mysql> SELECT COUNT(*) FROM Details;
+----------+
| COUNT(*) |
+----------+
|     4154 |
+----------+
1 row in set (0.03 sec)

mysql> SELECT COUNT(*) FROM Reviews;
+----------+
| COUNT(*) |
+----------+
|    75290 |
+----------+
1 row in set (1.23 sec)
```

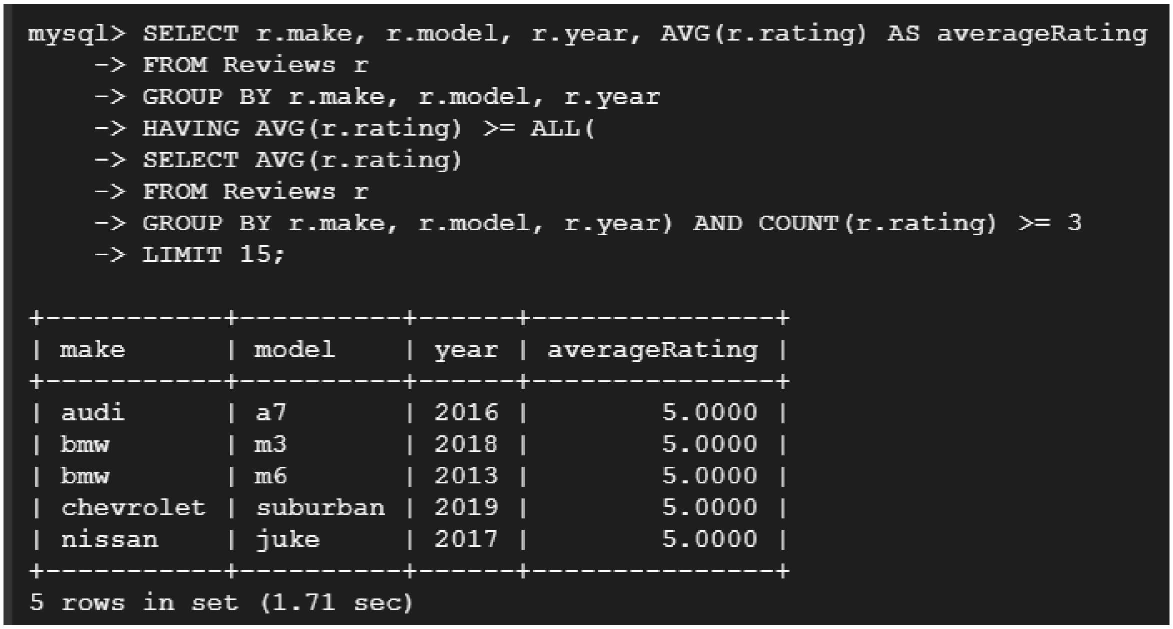
Advanced SQL Queries

Advanced Query 1

Functionality: Returns most popular / highest rated cars based on reviews

```
SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating
FROM Reviews r
GROUP BY r.make, r.model, r.year
HAVING AVG(r.rating) >= ALL(
SELECT AVG(r.rating)
FROM Reviews r
GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3
LIMIT 15;
```

Result of query:



```
mysql> SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating
    -> FROM Reviews r
    -> GROUP BY r.make, r.model, r.year
    -> HAVING AVG(r.rating) >= ALL(
    -> SELECT AVG(r.rating)
    -> FROM Reviews r
    -> GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3
    -> LIMIT 15;

+-----------+----------+------+---------------+
| make      | model    | year | averageRating |
+-----------+----------+------+---------------+
| audi      | a7       | 2016 |        5.0000 |
| bmw       | m3       | 2018 |        5.0000 |
| bmw       | m6       | 2013 |        5.0000 |
| chevrolet | suburban | 2019 |        5.0000 |
| nissan    | juke     | 2017 |        5.0000 |
+-----------+----------+------+---------------+
5 rows in set (1.71 sec)
```

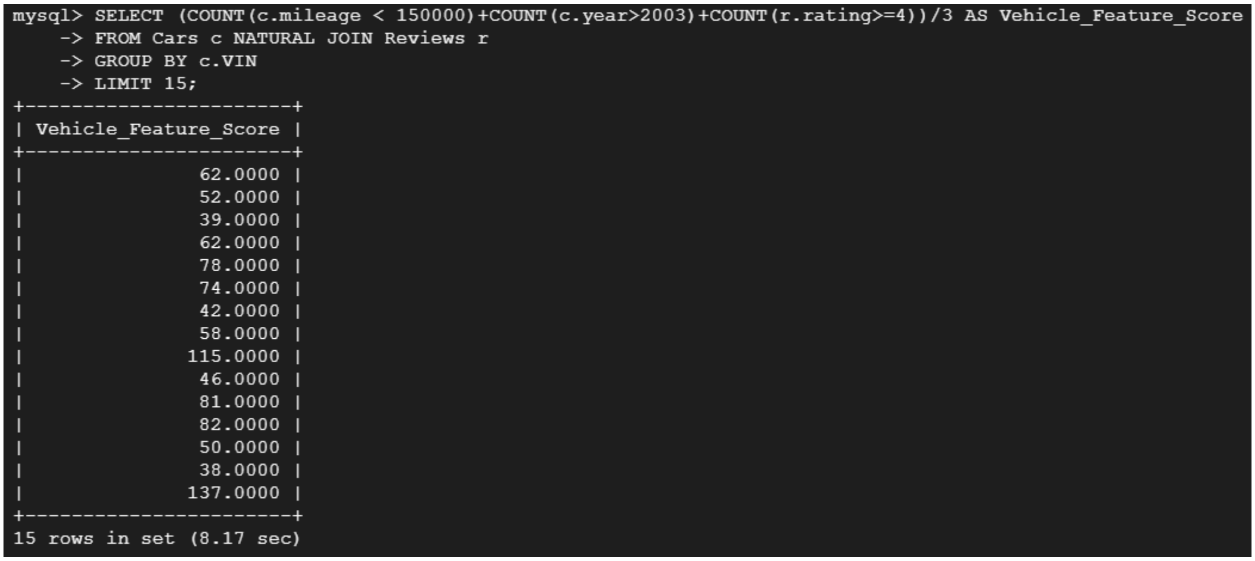
Note: the output is less than 15 rows.

Advanced Query 2

Functionality: Calculates vehicle features score (part of creative component)

```
SELECT (COUNT(c.mileage < 150000) + COUNT(c.year>2003) + COUNT(r.rating>=4))
/ 3 AS Vehicle_Feature_Score
FROM Cars c NATURAL JOIN Reviews r
GROUP BY c.VIN
LIMIT 15;
```

Result of query:



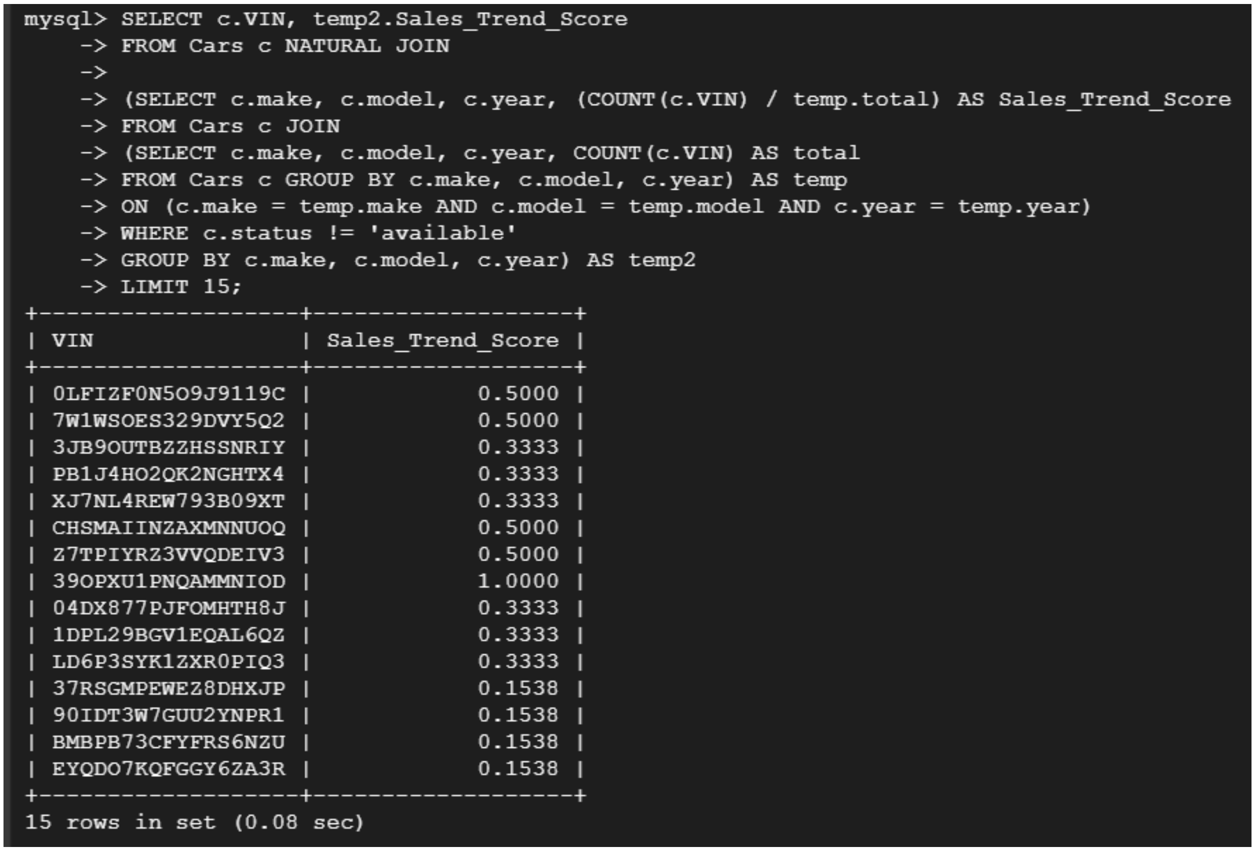
```
mysql> SELECT (COUNT(c.mileage < 150000)+COUNT(c.year>2003)+COUNT(r.rating>=4))/3 AS Vehicle_Feature_Score
    -> FROM Cars c NATURAL JOIN Reviews r
    -> GROUP BY c.VIN
    -> LIMIT 15;
+-----------------------+
| Vehicle_Feature_Score |
+-----------------------+
|               62.0000 |
|               52.0000 |
|               39.0000 |
|               62.0000 |
|               78.0000 |
|               74.0000 |
|               42.0000 |
|               58.0000 |
|              115.0000 |
|               46.0000 |
|               81.0000 |
|               82.0000 |
|               50.0000 |
|               38.0000 |
|              137.0000 |
+-----------------------+
15 rows in set (8.17 sec)
```

Advanced Query 3

Functionality: Calculates sales trend score (part of creative component)

```
SELECT c.VIN, temp2.Sales_Trend_Score
FROM Cars c NATURAL JOIN

(SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score
FROM Cars c JOIN
(SELECT c.make, c.model, c.year, COUNT(c.VIN) AS total
FROM Cars c GROUP BY c.make, c.model, c.year) AS temp
ON (c.make = temp.make AND c.model = temp.model AND c.year = temp.year)
WHERE c.status != 'available'
GROUP BY c.make, c.model, c.year) AS temp2
LIMIT 15;
```

Result of query:



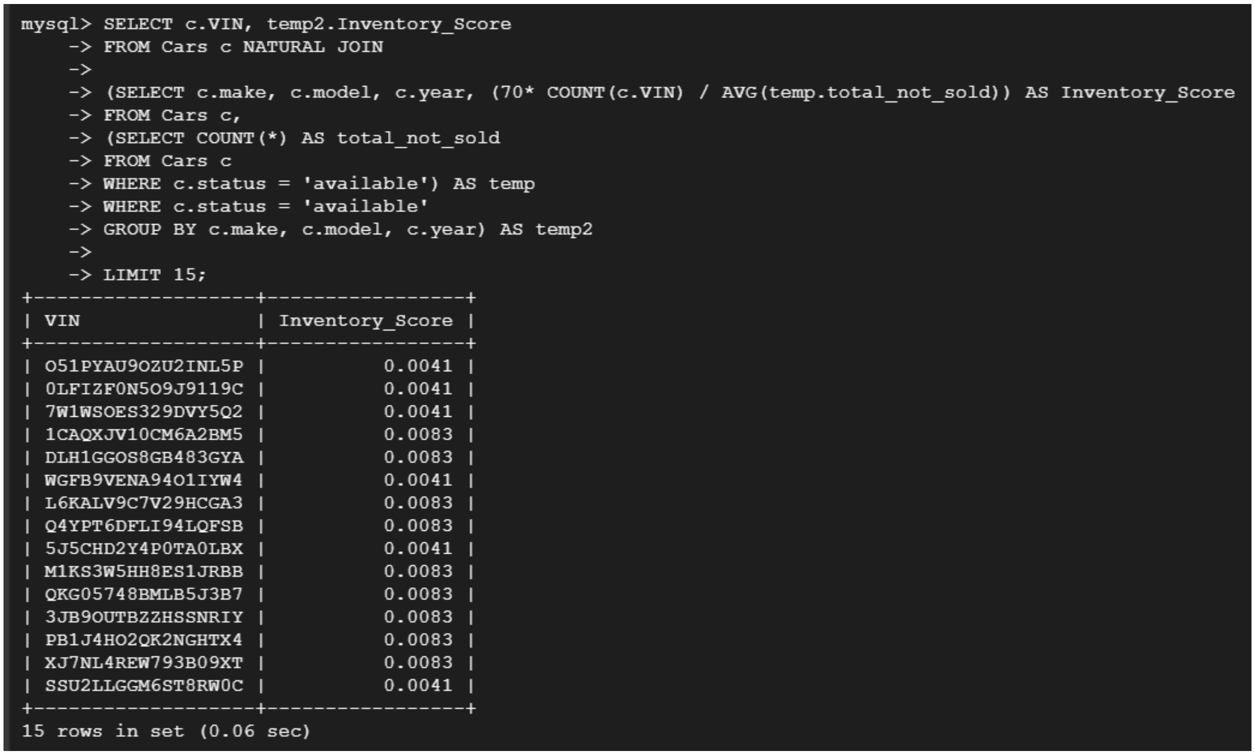
```
mysql> SELECT c.VIN, temp2.Sales_Trend_Score
    -> FROM Cars c NATURAL JOIN
    ->
    -> (SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score
    -> FROM Cars c JOIN
    -> (SELECT c.make, c.model, c.year, COUNT(c.VIN) AS total
    -> FROM Cars c GROUP BY c.make, c.model, c.year) AS temp
    -> ON (c.make = temp.make AND c.model = temp.model AND c.year = temp.year)
    -> WHERE c.status != 'available'
    -> GROUP BY c.make, c.model, c.year) AS temp2
    -> LIMIT 15;
+--------------------+-------------------+
| VIN                | Sales_Trend_Score |
+--------------------+-------------------+
| 0LFIZF0N5O9J9119C |            0.5000 |
| 7W1WSOES329DVY5Q2 |            0.5000 |
| 3JB9OUTBZZHSSNRIY |            0.3333 |
| PB1J4HO2QK2NGHTX4 |            0.3333 |
| XJ7NL4REW793B09XT |            0.3333 |
| CHSMAIINZAXMNNUOQ |            0.5000 |
| Z7TPIYRZ3VVQDEIV3 |            0.5000 |
| 39OPXU1PNQAMMNIOD |            1.0000 |
| 04DX877PJFOMHTH8J |            0.3333 |
| 1DPL29BGV1EQAL6QZ |            0.3333 |
| LD6P3SYK1ZXR0PIQ3 |            0.3333 |
| 37RSGMPEWEZ8DHXJP |            0.1538 |
| 90IDT3W7GUU2YNPR1 |            0.1538 |
| BMBPB73CFYFRS6NZU |            0.1538 |
| EYQDO7KQFGGY6ZA3R |            0.1538 |
+--------------------+-------------------+
15 rows in set (0.08 sec)
```

Advanced Query 4

Functionality: Calculates inventory score (part of creative component)

```
SELECT c.VIN, temp2.Inventory_Score
FROM Cars c NATURAL JOIN

(SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS
Inventory_Score
FROM Cars c,
(SELECT COUNT(*) AS total_not_sold
FROM Cars c
WHERE c.status = 'available') AS temp
WHERE c.status = 'available'
GROUP BY c.make, c.model, c.year) AS temp2

LIMIT 15;
```

Result of query:

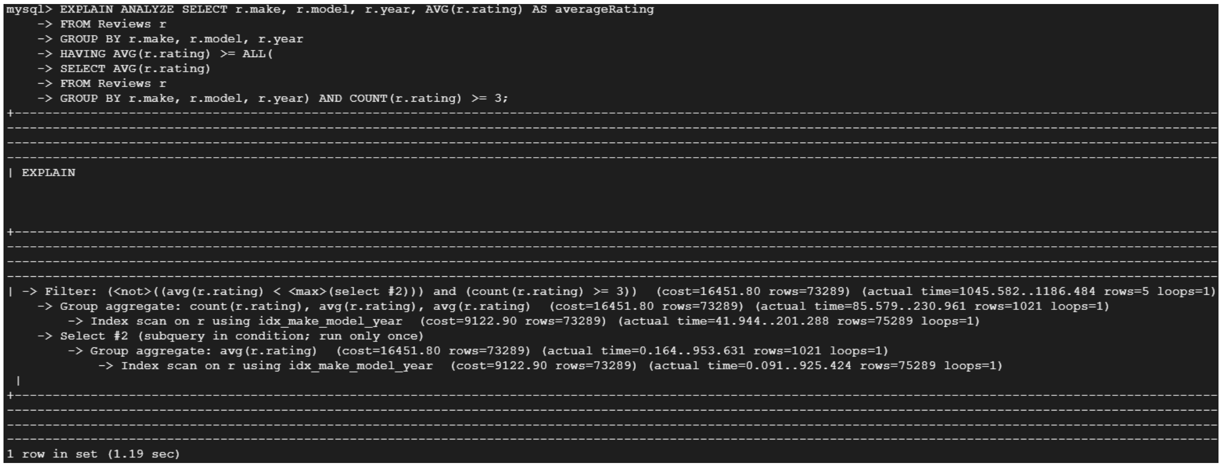


```
mysql> SELECT c.VIN, temp2.Inventory_Score
    -> FROM Cars c NATURAL JOIN
    ->
    -> (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS Inventory_Score
    -> FROM Cars c,
    -> (SELECT COUNT(*) AS total_not_sold
    -> FROM Cars c
    -> WHERE c.status = 'available') AS temp
    -> WHERE c.status = 'available'
    -> GROUP BY c.make, c.model, c.year) AS temp2
    ->
    -> LIMIT 15;
+-------------------+-----------------+
| VIN               | Inventory_Score |
+-------------------+-----------------+
| O51PYAU9OZU2INL5P |          0.0041 |
| 0LFIZF0N5O9J9119C |          0.0041 |
| 7W1WSOES329DVY5Q2 |          0.0041 |
| 1CAQXJV10CM6A2BM5 |          0.0083 |
| DLH1GGOS8GB483GYA |          0.0083 |
| WGFB9VENA94O1IYW4 |          0.0041 |
| L6KALV9C7V29HCGA3 |          0.0083 |
| Q4YPT6DFLI94LQFSB |          0.0083 |
| 5J5CHD2Y4P0TA0LBX |          0.0041 |
| M1KS3W5HH8ES1JRBB |          0.0083 |
| QKG05748BMLB5J3B7 |          0.0083 |
| 3JB9OUTBZZHSSNRIY |          0.0083 |
| PB1J4HO2QK2NGHTX4 |          0.0083 |
| XJ7NL4REW793B09XT |          0.0083 |
| SSU2LLGGM6ST8RW0C |          0.0041 |
+-------------------+-----------------+
15 rows in set (0.06 sec)
```

Indexing

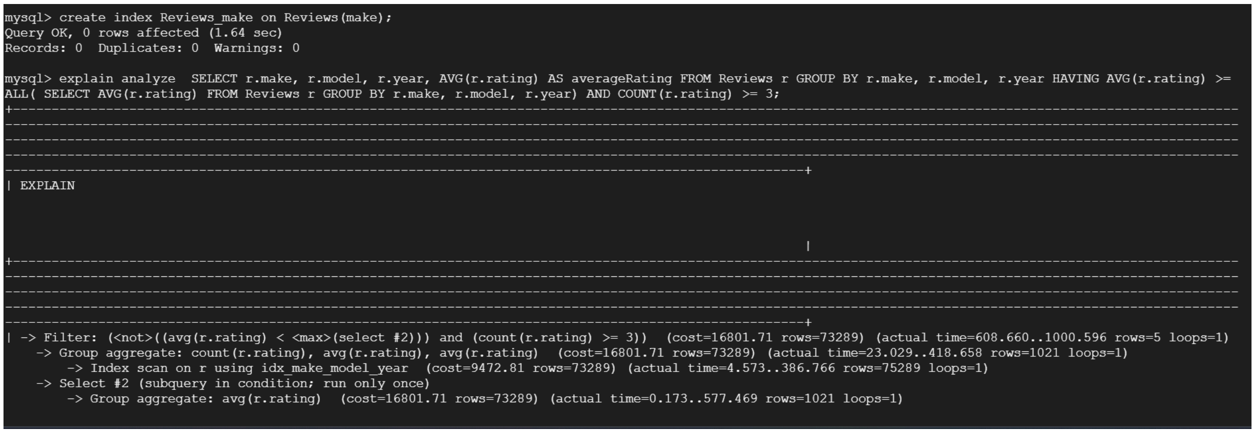
Advanced Query 1

1) Without indexing, Reviews(cost) = 16451.80



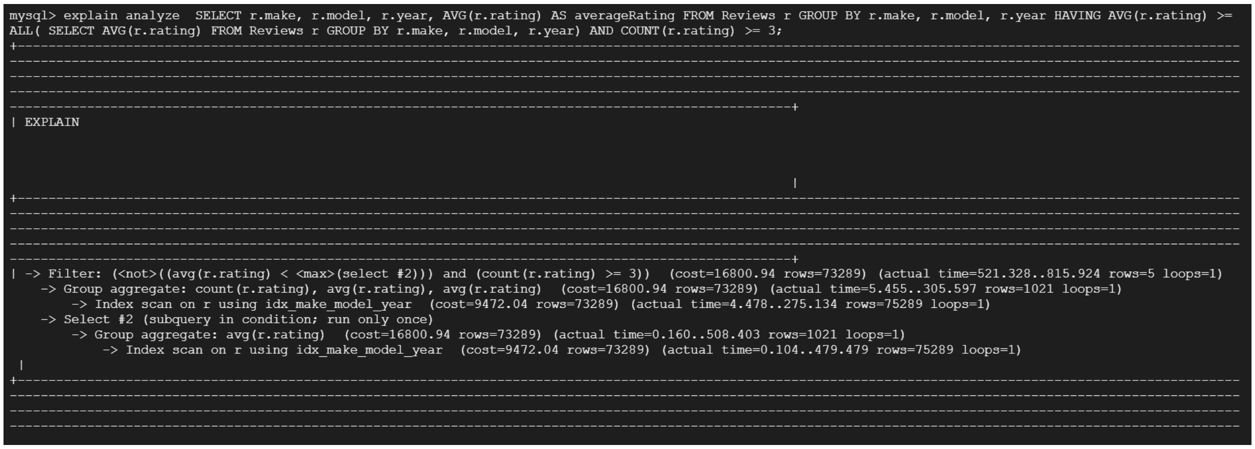
```
mysql> EXPLAIN ANALYZE SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating
    -> FROM Reviews r
    -> GROUP BY r.make, r.model, r.year
    -> HAVING AVG(r.rating) >= ALL(
    -> SELECT AVG(r.rating)
    -> FROM Reviews r
    -> GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3;
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
| -> Filter: (<not>((avg(r.rating) < <max>(select #2))) and (count(r.rating) >= 3))  (cost=16451.80 rows=73289) (actual time=1045.582..1186.484 rows=5 loops=1)
    -> Group aggregate: count(r.rating), avg(r.rating), avg(r.rating)  (cost=16451.80 rows=73289) (actual time=85.579..230.961 rows=1021 loops=1)
        -> Index scan on r using idx_make_model_year  (cost=9122.90 rows=73289) (actual time=41.944..201.288 rows=75289 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Group aggregate: avg(r.rating)  (cost=16451.80 rows=73289) (actual time=0.164..953.631 rows=1021 loops=1)
            -> Index scan on r using idx_make_model_year  (cost=9122.90 rows=73289) (actual time=0.091..925.424 rows=75289 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
1 row in set (1.19 sec)
```

2) With indexing on Reviews(make), cost = 16801.71



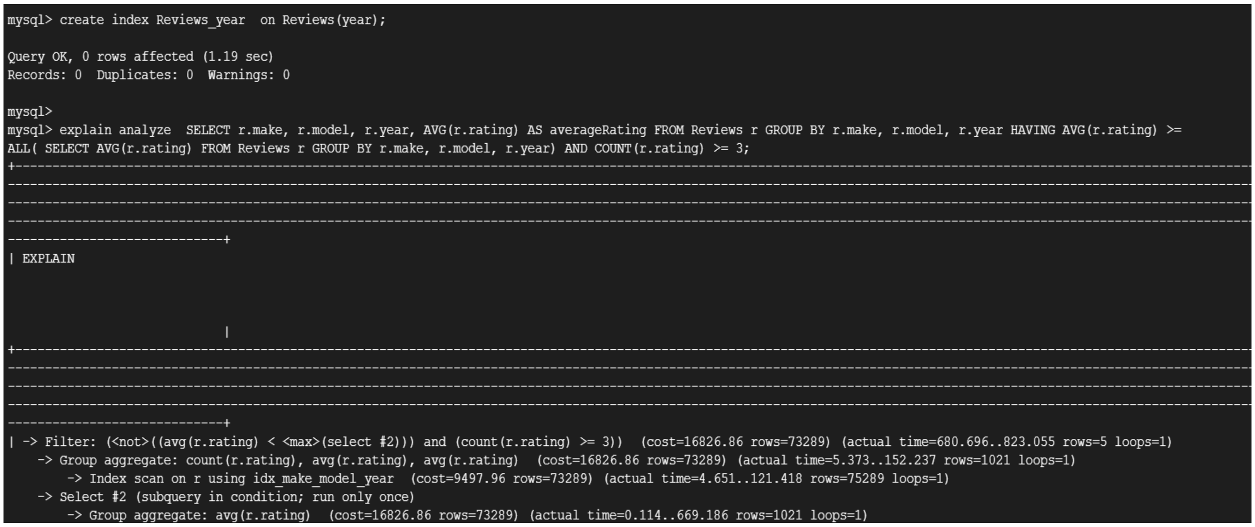
```
mysql> create index Reviews_make on Reviews(make);
Query OK, 0 rows affected (1.64 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze  SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating FROM Reviews r GROUP BY r.make, r.model, r.year HAVING AVG(r.rating) >=
ALL( SELECT AVG(r.rating) FROM Reviews r GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3;
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
| EXPLAIN



                                                                              |
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
| -> Filter: (<not>((avg(r.rating) < <max>(select #2))) and (count(r.rating) >= 3))  (cost=16801.71 rows=73289) (actual time=608.660..1000.596 rows=5 loops=1)
    -> Group aggregate: count(r.rating), avg(r.rating), avg(r.rating)  (cost=16801.71 rows=73289) (actual time=23.029..418.658 rows=1021 loops=1)
        -> Index scan on r using idx_make_model_year  (cost=9472.81 rows=73289) (actual time=4.573..386.766 rows=75289 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Group aggregate: avg(r.rating)  (cost=16801.71 rows=73289) (actual time=0.173..577.469 rows=1021 loops=1)
```

3) With indexing on Reviews(model), cost = 16800.94



```
mysql> explain analyze  SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating FROM Reviews r GROUP BY r.make, r.model, r.year HAVING AVG(r.rating) >=
ALL( SELECT AVG(r.rating) FROM Reviews r GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3;
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
| EXPLAIN



                                                                              |
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
| -> Filter: (<not>((avg(r.rating) < <max>(select #2))) and (count(r.rating) >= 3))  (cost=16800.94 rows=73289) (actual time=521.328..815.924 rows=5 loops=1)
    -> Group aggregate: count(r.rating), avg(r.rating), avg(r.rating)  (cost=16800.94 rows=73289) (actual time=5.455..305.597 rows=1021 loops=1)
        -> Index scan on r using idx_make_model_year  (cost=9472.04 rows=73289) (actual time=4.478..275.134 rows=75289 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Group aggregate: avg(r.rating)  (cost=16800.94 rows=73289) (actual time=0.160..508.403 rows=1021 loops=1)
            -> Index scan on r using idx_make_model_year  (cost=9472.04 rows=73289) (actual time=0.104..479.479 rows=75289 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
```

4) With indexing on Reviews(year), cost = 16826.86



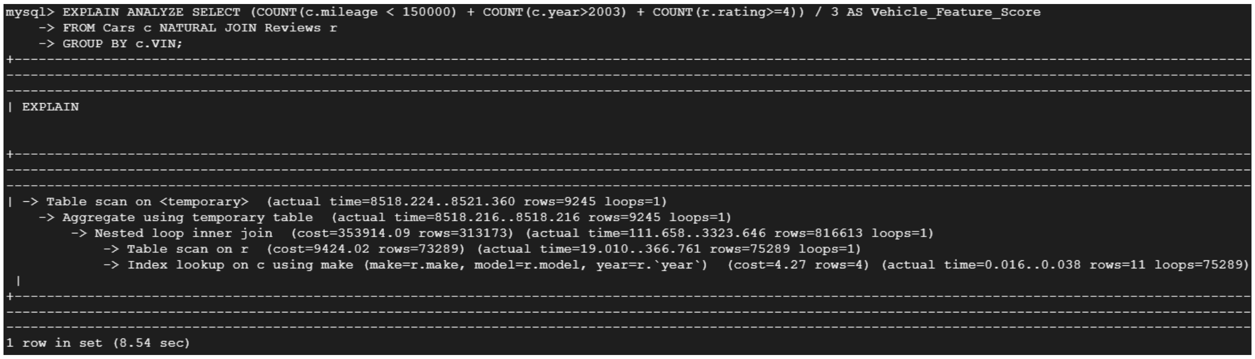
```
mysql> create index Reviews_year  on Reviews(year);

Query OK, 0 rows affected (1.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> explain analyze  SELECT r.make, r.model, r.year, AVG(r.rating) AS averageRating FROM Reviews r GROUP BY r.make, r.model, r.year HAVING AVG(r.rating) >=
ALL( SELECT AVG(r.rating) FROM Reviews r GROUP BY r.make, r.model, r.year) AND COUNT(r.rating) >= 3;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------+
| EXPLAIN




                             |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------+
| -> Filter: (<not>((avg(r.rating) < <max>(select #2))) and (count(r.rating) >= 3))  (cost=16826.86 rows=73289) (actual time=680.696..823.055 rows=5 loops=1)
    -> Group aggregate: count(r.rating), avg(r.rating), avg(r.rating)  (cost=16826.86 rows=73289) (actual time=5.373..152.237 rows=1021 loops=1)
        -> Index scan on r using idx_make_model_year  (cost=9497.96 rows=73289) (actual time=4.651..121.418 rows=75289 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Group aggregate: avg(r.rating)  (cost=16826.86 rows=73289) (actual time=0.114..669.186 rows=1021 loops=1)
```

Report:

There is no significant difference between which indexing model we use since we have the similar costs of around 16400-16800. It is due to the fact that this query involves both grouping and aggregation on multiple columns, with a condition on the average rating that requires a full scan of the data. Since the query calculates AVG(r.rating) across grouped combinations of make, model, and year, the database must evaluate the aggregate function over each unique combination, making the indexes we chose ineffective. Without indexes tailored for optimizing aggregation, those indexes do not help significantly, resulting in similar costs with or without indexing.

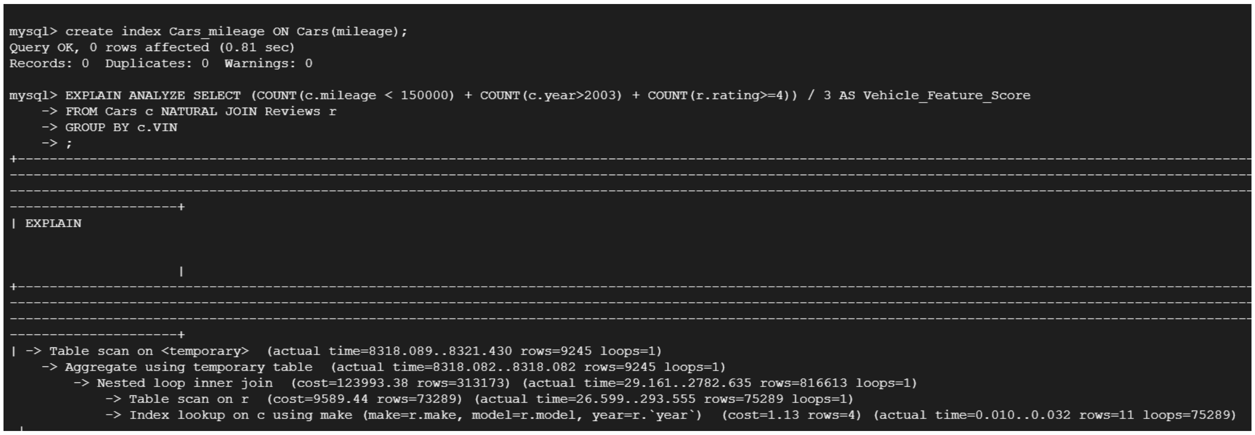
Advanced Query 2

1) Without indexing, cost = 353914.09



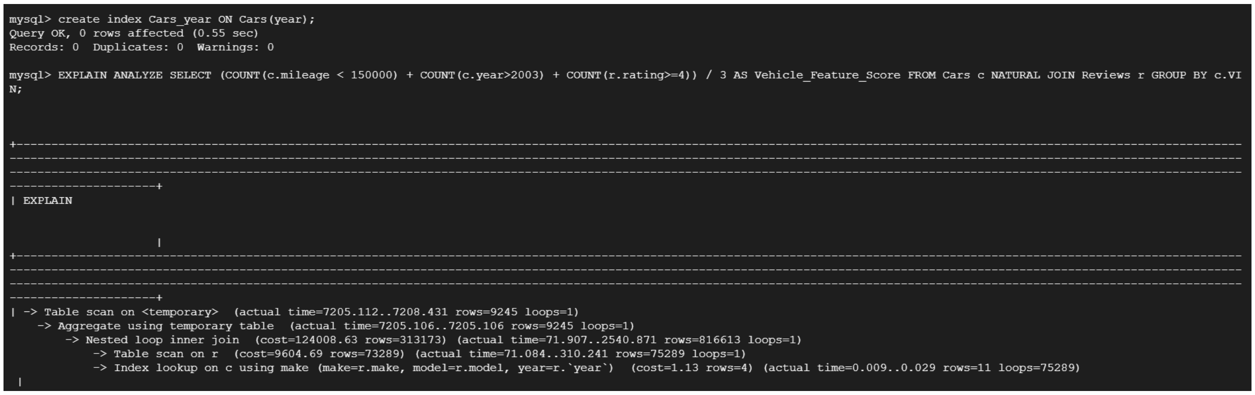
```
mysql> EXPLAIN ANALYZE SELECT (COUNT(c.mileage < 150000) + COUNT(c.year>2003) + COUNT(r.rating>=4)) / 3 AS Vehicle_Feature_Score
    -> FROM Cars c NATURAL JOIN Reviews r
    -> GROUP BY c.VIN;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=8518.224..8521.360 rows=9245 loops=1)
    -> Aggregate using temporary table  (actual time=8518.216..8518.216 rows=9245 loops=1)
        -> Nested loop inner join  (cost=353914.09 rows=313173) (actual time=111.658..3323.646 rows=816613 loops=1)
            -> Table scan on r  (cost=9424.02 rows=73289) (actual time=19.010..366.761 rows=75289 loops=1)
            -> Index lookup on c using make (make=r.make, model=r.model, year=r.`year`)  (cost=4.27 rows=4) (actual time=0.016..0.038 rows=11 loops=75289)
 |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
1 row in set (8.54 sec)
```

2) With indexing on Cars(mileage), cost = 123993.38



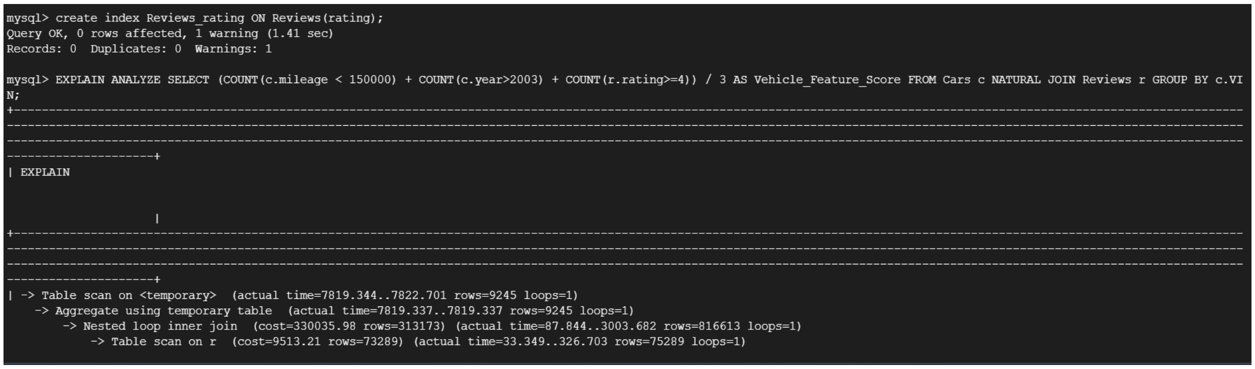
```
mysql> create index Cars_mileage ON Cars(mileage);
Query OK, 0 rows affected (0.81 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT (COUNT(c.mileage < 150000) + COUNT(c.year>2003) + COUNT(r.rating>=4)) / 3 AS Vehicle_Feature_Score
    -> FROM Cars c NATURAL JOIN Reviews r
    -> GROUP BY c.VIN
    -> ;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| EXPLAIN


                     |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| -> Table scan on <temporary>  (actual time=8318.089..8321.430 rows=9245 loops=1)
    -> Aggregate using temporary table  (actual time=8318.082..8318.082 rows=9245 loops=1)
        -> Nested loop inner join  (cost=123993.38 rows=313173) (actual time=29.161..2782.635 rows=816613 loops=1)
            -> Table scan on r  (cost=9589.44 rows=73289) (actual time=26.599..293.555 rows=75289 loops=1)
            -> Index lookup on c using make (make=r.make, model=r.model, year=r.`year`)  (cost=1.13 rows=4) (actual time=0.010..0.032 rows=11 loops=75289)
```

3) With indexing on Cars(year), cost = 124008.63



```
mysql> create index Cars_year ON Cars(year);
Query OK, 0 rows affected (0.55 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT (COUNT(c.mileage < 150000) + COUNT(c.year>2003) + COUNT(r.rating>=4)) / 3 AS Vehicle_Feature_Score FROM Cars c NATURAL JOIN Reviews r GROUP BY c.VI
N;



+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| EXPLAIN


                     |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| -> Table scan on <temporary>  (actual time=7205.112..7208.431 rows=9245 loops=1)
    -> Aggregate using temporary table  (actual time=7205.106..7205.106 rows=9245 loops=1)
        -> Nested loop inner join  (cost=124008.63 rows=313173) (actual time=71.907..2540.871 rows=816613 loops=1)
            -> Table scan on r  (cost=9604.69 rows=73289) (actual time=71.084..310.241 rows=75289 loops=1)
            -> Index lookup on c using make (make=r.make, model=r.model, year=r.`year`)  (cost=1.13 rows=4) (actual time=0.009..0.029 rows=11 loops=75289)
 |
```

4) With indexing on Reviews(rating), cost = 330035.98



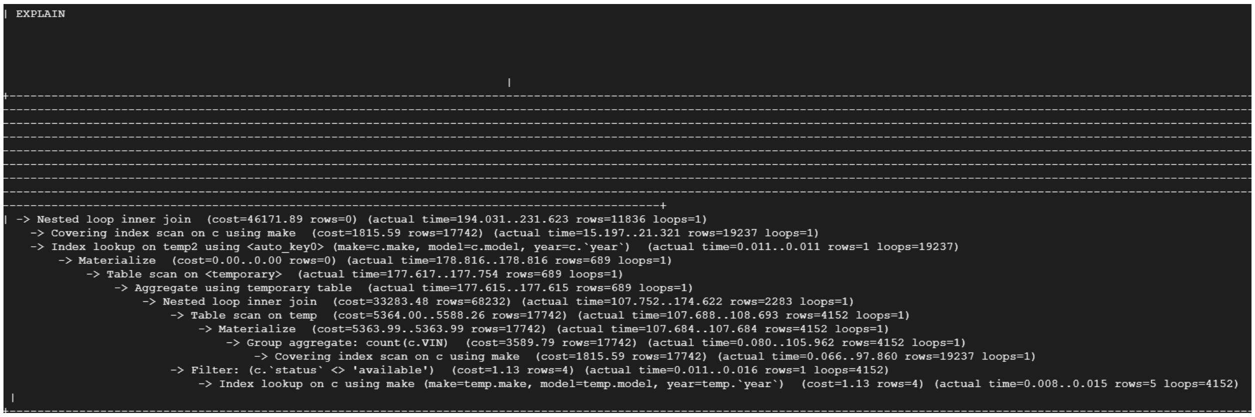
```
mysql> create index Reviews_rating ON Reviews(rating);
Query OK, 0 rows affected, 1 warning (1.41 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT (COUNT(c.mileage < 150000) + COUNT(c.year>2003) + COUNT(r.rating>=4)) / 3 AS Vehicle_Feature_Score FROM Cars c NATURAL JOIN Reviews r GROUP BY c.VI
N;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| EXPLAIN


                     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------+
| -> Table scan on <temporary>  (actual time=7819.344..7822.701 rows=9245 loops=1)
    -> Aggregate using temporary table  (actual time=7819.337..7819.337 rows=9245 loops=1)
        -> Nested loop inner join  (cost=330035.98 rows=313173) (actual time=87.844..3003.682 rows=816613 loops=1)
            -> Table scan on r  (cost=9513.21 rows=73289) (actual time=33.349..326.703 rows=75289 loops=1)
```

Report:

We choose indexing model #2 (indexing on Cars(mileage)) because we have the lowest cost of 123993.38 compared to the index on Cars(year), and Reviews(rating) which have a higher cost. Doing so resulted in the lowest cost because the query frequently filters and evaluates conditions on mileage (c.mileage < 150000). Since mileage is one of the primary filtering conditions in this query, the database can leverage the index on mileage to access only relevant rows. This speeds up the evaluation process and lowers execution cost.

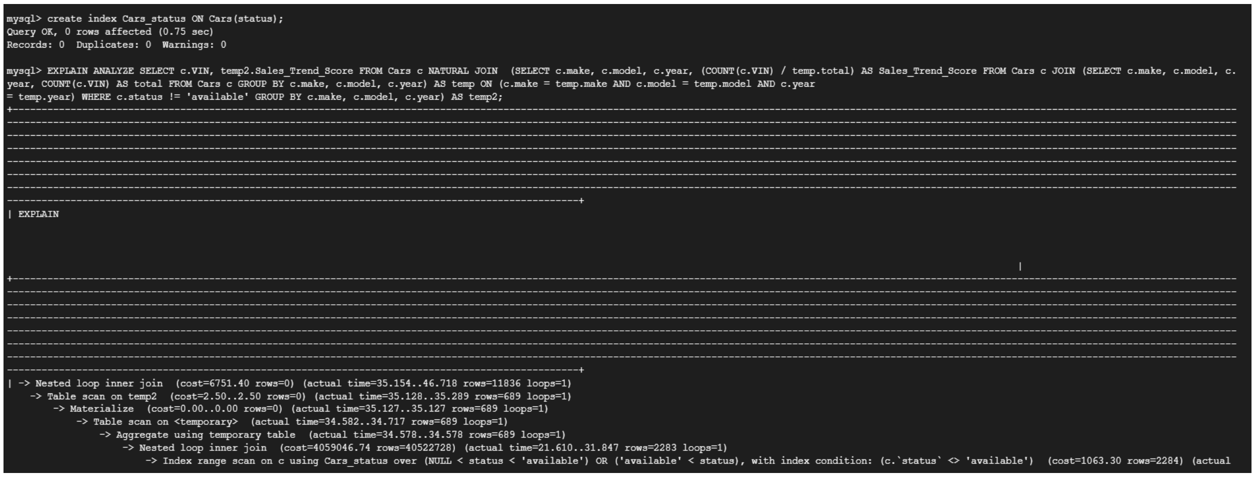
Advanced Query 3

1) Without indexing, cost = 46171.89



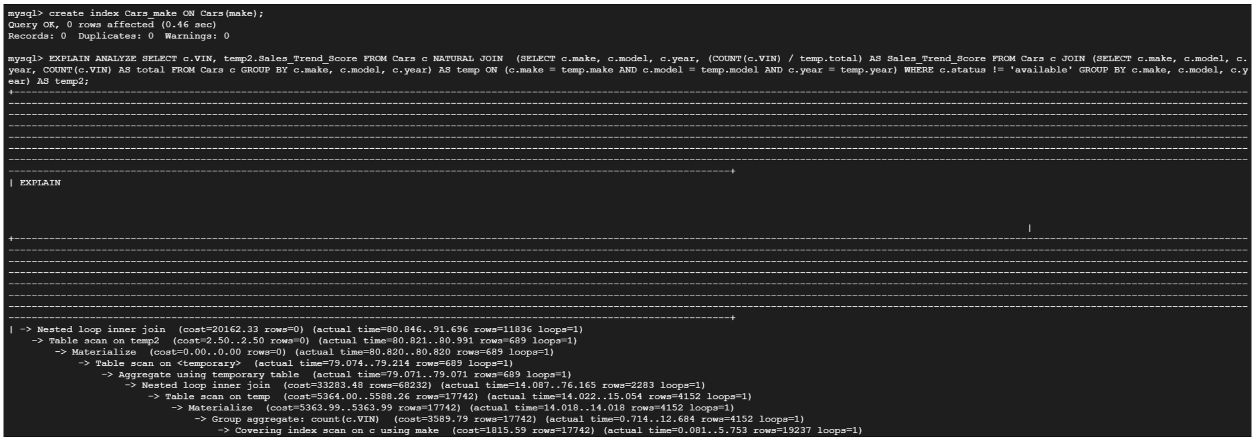
```
| EXPLAIN

                                                                          |
+--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=46171.89 rows=0) (actual time=194.031..231.623 rows=11836 loops=1)
    -> Covering index scan on c using make  (cost=1815.59 rows=17742) (actual time=15.197..21.321 rows=19237 loops=1)
    -> Index lookup on temp2 using <auto_key0> (make=c.make, model=c.model, year=c.`year`)  (actual time=0.011..0.011 rows=1 loops=19237)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=178.816..178.816 rows=689 loops=1)
            -> Table scan on <temporary>  (actual time=177.617..177.754 rows=689 loops=1)
                -> Aggregate using temporary table  (actual time=177.615..177.615 rows=689 loops=1)
                    -> Nested loop inner join  (cost=33283.48 rows=68232) (actual time=107.752..174.622 rows=2283 loops=1)
                        -> Table scan on temp  (cost=5364.00..5588.26 rows=17742) (actual time=107.688..108.693 rows=4152 loops=1)
                            -> Materialize  (cost=5363.99..5363.99 rows=17742) (actual time=107.684..107.684 rows=4152 loops=1)
                                -> Group aggregate: count(c.VIN)  (cost=3589.79 rows=17742) (actual time=0.080..105.962 rows=4152 loops=1)
                                    -> Covering index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.066..97.860 rows=19237 loops=1)
                        -> Filter: (c.`status` <> 'available')  (cost=1.13 rows=4) (actual time=0.011..0.016 rows=1 loops=4152)
                            -> Index lookup on c using make (make=temp.make, model=temp.model, year=temp.`year`)  (cost=1.13 rows=4) (actual time=0.008..0.015 rows=5 loops=4152)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------
```

2) With indexing Cars(status), cost = 6751.40



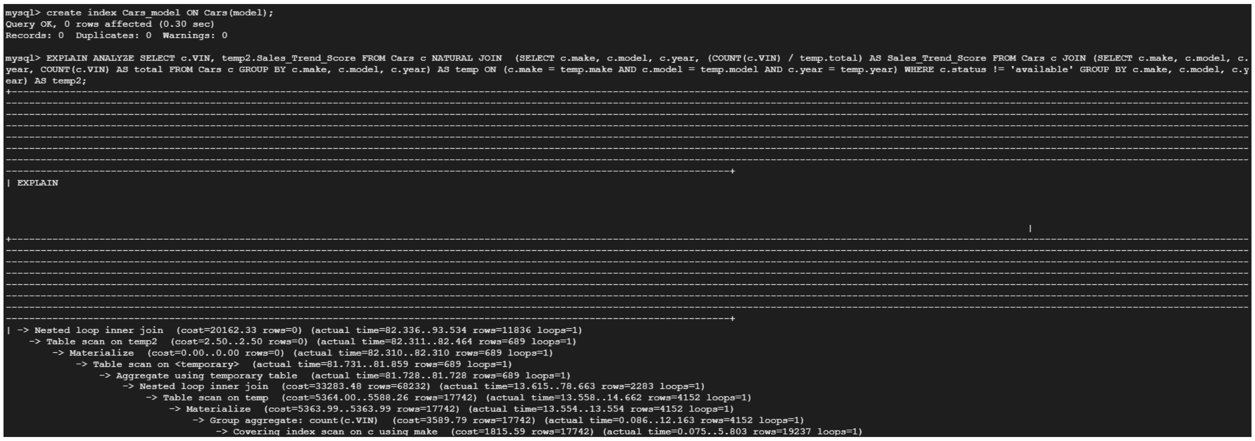
```
mysql> create index Cars_status ON Cars(status);
Query OK, 0 rows affected (0.75 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Sales_Trend_Score FROM Cars c NATURAL JOIN  (SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score FROM Cars c JOIN (SELECT c.make, c.model, c.
year, COUNT(c.VIN) AS total FROM Cars c GROUP BY c.make, c.model, c.year) AS temp ON (c.make = temp.make AND c.model = temp.model AND c.year
= temp.year) WHERE c.status != 'available' GROUP BY c.make, c.model, c.year) AS temp2;
+--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                        |
+--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=6751.40 rows=0) (actual time=35.154..46.718 rows=11836 loops=1)
    -> Table scan on temp2  (cost=2.50..2.50 rows=0) (actual time=35.128..35.289 rows=689 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=35.127..35.127 rows=689 loops=1)
            -> Table scan on <temporary>  (actual time=34.582..34.717 rows=689 loops=1)
                -> Aggregate using temporary table  (actual time=34.578..34.578 rows=689 loops=1)
                    -> Nested loop inner join  (cost=4059046.74 rows=40522728) (actual time=21.610..31.847 rows=2283 loops=1)
                        -> Index range scan on c using Cars_status over (NULL < status < 'available') OR ('available' < status), with index condition: (c.`status` <> 'available')  (cost=1063.30 rows=2284) (actual
```

3) With indexing Cars(make), cost = 20162.33



```
mysql> create index Cars_make ON Cars(make);
Query OK, 0 rows affected (0.46 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Sales_Trend_Score FROM Cars c NATURAL JOIN  (SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score FROM Cars c JOIN (SELECT c.make, c.model, c.
year, COUNT(c.VIN) AS total FROM Cars c GROUP BY c.make, c.model, c.year) AS temp ON (c.make = temp.make AND c.model = temp.model AND c.year = temp.year) WHERE c.status != 'available' GROUP BY c.make, c.model, c.y
ear) AS temp2;
+--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                                        |
+--------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=20162.33 rows=0) (actual time=80.846..91.696 rows=11836 loops=1)
    -> Table scan on temp2  (cost=2.50..2.50 rows=0) (actual time=80.821..80.991 rows=689 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=80.820..80.820 rows=689 loops=1)
            -> Table scan on <temporary>  (actual time=79.074..79.214 rows=689 loops=1)
                -> Aggregate using temporary table  (actual time=79.071..79.071 rows=689 loops=1)
                    -> Nested loop inner join  (cost=33283.48 rows=68232) (actual time=14.087..76.165 rows=2283 loops=1)
                        -> Table scan on temp  (cost=5364.00..5588.26 rows=17742) (actual time=14.022..15.054 rows=4152 loops=1)
                            -> Materialize  (cost=5363.99..5363.99 rows=17742) (actual time=14.018..14.018 rows=4152 loops=1)
                                -> Group aggregate: count(c.VIN)  (cost=3589.79 rows=17742) (actual time=0.714..12.684 rows=4152 loops=1)
                                    -> Covering index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.081..5.753 rows=19237 loops=1)
```

4) With indexing Cars(model), cost = 20162.33



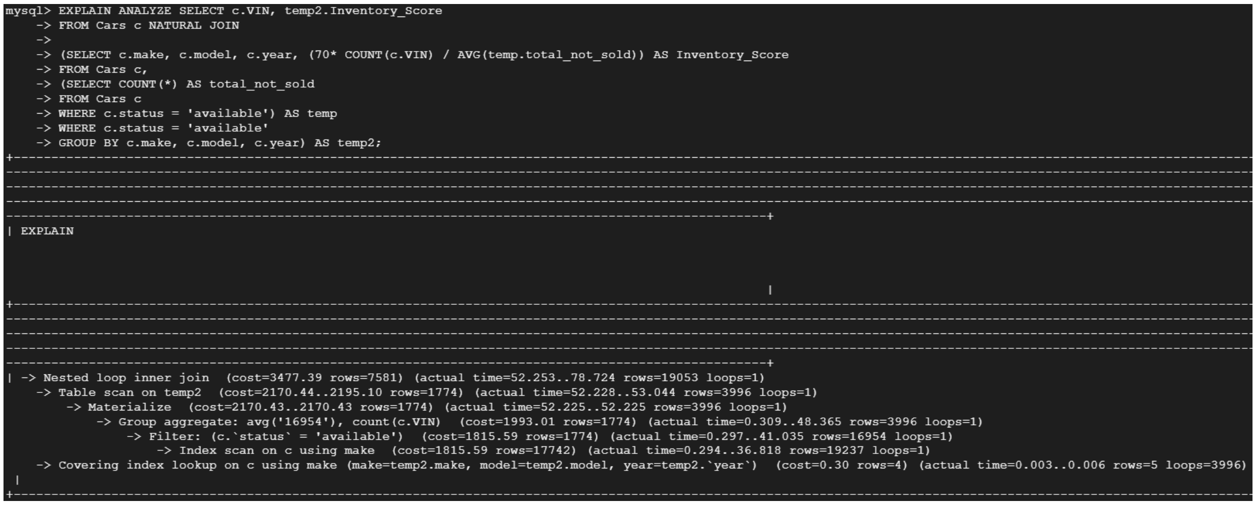
```
mysql> create index Cars_model ON Cars(model);
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Sales_Trend_Score FROM Cars c NATURAL JOIN  (SELECT c.make, c.model, c.year, (COUNT(c.VIN) / temp.total) AS Sales_Trend_Score FROM Cars c JOIN (SELECT c.make, c.model, c.
year, COUNT(c.VIN) AS total FROM Cars c GROUP BY c.make, c.model, c.year) AS temp ON (c.make = temp.make AND c.model = temp.model AND c.year = temp.year) WHERE c.status != 'available' GROUP BY c.make, c.model, c.y
ear) AS temp2;
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| EXPLAIN

                                                                                                                    |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------+
| -> Nested loop inner join  (cost=20162.33 rows=0) (actual time=82.336..93.534 rows=11836 loops=1)
    -> Table scan on temp2  (cost=2.50..2.50 rows=0) (actual time=82.311..82.464 rows=689 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=82.310..82.310 rows=689 loops=1)
            -> Table scan on <temporary>  (actual time=81.731..81.859 rows=689 loops=1)
                -> Aggregate using temporary table  (actual time=81.728..81.728 rows=689 loops=1)
                    -> Nested loop inner join  (cost=33283.48 rows=68232) (actual time=13.615..78.663 rows=2283 loops=1)
                        -> Table scan on temp  (cost=5364.00..5588.26 rows=17742) (actual time=13.558..14.662 rows=4152 loops=1)
                            -> Materialize  (cost=5363.99..5363.99 rows=17742) (actual time=13.554..13.554 rows=4152 loops=1)
                                -> Group aggregate: count(c.VIN)  (cost=3589.79 rows=17742) (actual time=0.086..12.163 rows=4152 loops=1)
                                    -> Covering index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.075..5.803 rows=19237 loops=1)
```

Report:

We choose indexing model #2 (indexing on Cars(status)) because we have the lowest cost of 6751.40 compared to the index on Cars(make) and Cars(model) which both have a cost of 20162.33 . Doing so lowers the query cost because it optimizes the filtering condition in the subquery WHERE c.status != 'available'. This part of the query identifies cars that are not available, and with an index on status, the database can quickly locate only the relevant rows instead of scanning through the entire Cars table.

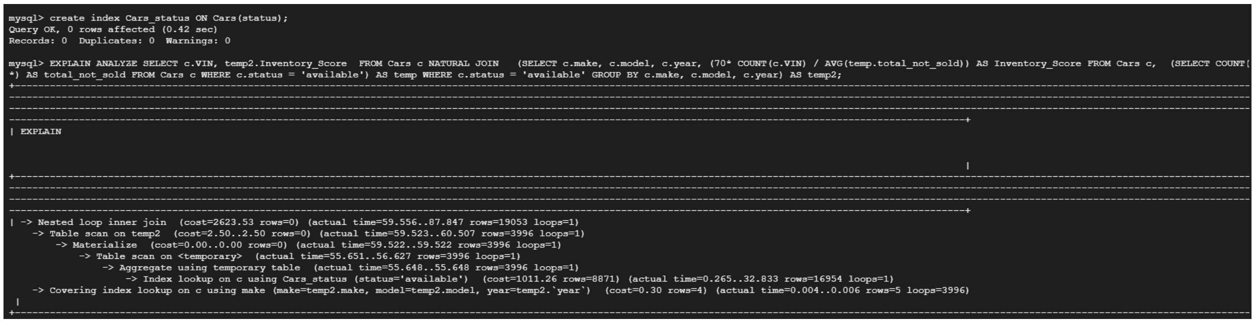
Advanced Query 4

1) Without indexing, cost = 3477.39



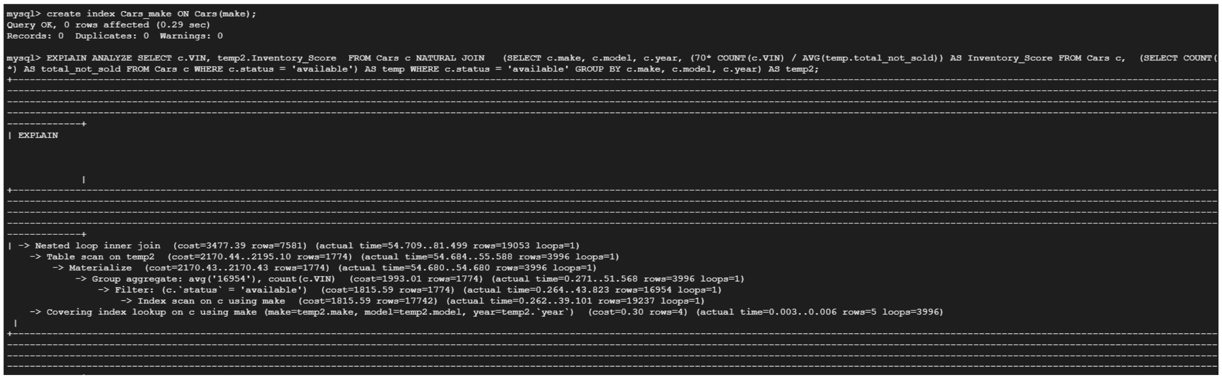
```
mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Inventory_Score
    -> FROM Cars c NATURAL JOIN
    ->
    -> (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS Inventory_Score
    -> FROM Cars c,
    -> (SELECT COUNT(*) AS total_not_sold
    -> FROM Cars c
    -> WHERE c.status = 'available') AS temp
    -> WHERE c.status = 'available'
    -> GROUP BY c.make, c.model, c.year) AS temp2;
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| EXPLAIN



                                                                                                     |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=3477.39 rows=7581) (actual time=52.253..78.724 rows=19053 loops=1)
    -> Table scan on temp2  (cost=2170.44..2195.10 rows=1774) (actual time=52.228..53.044 rows=3996 loops=1)
        -> Materialize  (cost=2170.43..2170.43 rows=1774) (actual time=52.225..52.225 rows=3996 loops=1)
            -> Group aggregate: avg('16954'), count(c.VIN)  (cost=1993.01 rows=1774) (actual time=0.309..48.365 rows=3996 loops=1)
                -> Filter: (c.`status` = 'available')  (cost=1815.59 rows=1774) (actual time=0.297..41.035 rows=16954 loops=1)
                    -> Index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.294..36.818 rows=19237 loops=1)
    -> Covering index lookup on c using make (make=temp2.make, model=temp2.model, year=temp2.`year`)  (cost=0.30 rows=4) (actual time=0.003..0.006 rows=5 loops=3996)
 |
+------------------------------------------------------------------------------------------------------------
```

2) With indexing on Cars(status), cost = 2623.53



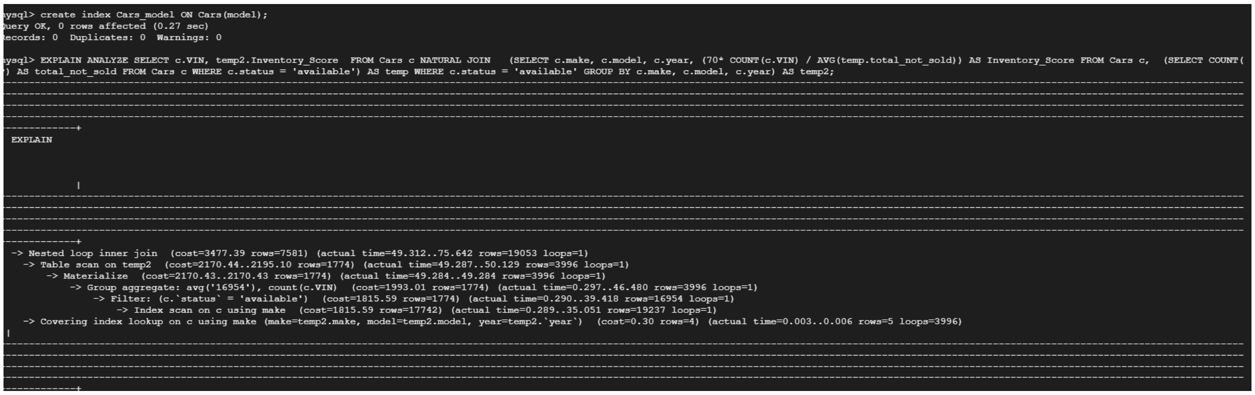
```
mysql> create index Cars_status ON Cars(status);
Query OK, 0 rows affected (0.42 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Inventory_Score  FROM Cars c NATURAL JOIN   (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS Inventory_Score FROM Cars c,  (SELECT COUNT(
*) AS total_not_sold FROM Cars c WHERE c.status = 'available') AS temp WHERE c.status = 'available' GROUP BY c.make, c.model, c.year) AS temp2;
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                     |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=2623.53 rows=0) (actual time=59.556..87.847 rows=19053 loops=1)
    -> Table scan on temp2  (cost=2.50..2.50 rows=0) (actual time=59.523..60.507 rows=3996 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=59.522..59.522 rows=3996 loops=1)
            -> Table scan on <temporary>  (actual time=55.651..56.627 rows=3996 loops=1)
                -> Aggregate using temporary table  (actual time=55.648..55.648 rows=3996 loops=1)
                    -> Index lookup on c using Cars_status (status='available')  (cost=1011.26 rows=8871) (actual time=0.265..32.833 rows=16954 loops=1)
    -> Covering index lookup on c using make (make=temp2.make, model=temp2.model, year=temp2.`year`)  (cost=0.30 rows=4) (actual time=0.004..0.006 rows=5 loops=3996)
 |
+------------------------------------------------------------------------------------------------------------
```

3) With indexing on Cars(make), cost = 3477.39



```
mysql> create index Cars_make ON Cars(make);
Query OK, 0 rows affected (0.29 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Inventory_Score  FROM Cars c NATURAL JOIN   (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS Inventory_Score FROM Cars c,  (SELECT COUNT(
*) AS total_not_sold FROM Cars c WHERE c.status = 'available') AS temp WHERE c.status = 'available' GROUP BY c.make, c.model, c.year) AS temp2;
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------+
| EXPLAIN


            |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------+
| -> Nested loop inner join  (cost=3477.39 rows=7581) (actual time=54.709..81.499 rows=19053 loops=1)
    -> Table scan on temp2  (cost=2170.44..2195.10 rows=1774) (actual time=54.684..55.588 rows=3996 loops=1)
        -> Materialize  (cost=2170.43..2170.43 rows=1774) (actual time=54.680..54.680 rows=3996 loops=1)
            -> Group aggregate: avg('16954'), count(c.VIN)  (cost=1993.01 rows=1774) (actual time=0.271..51.568 rows=3996 loops=1)
                -> Filter: (c.`status` = 'available')  (cost=1815.59 rows=1774) (actual time=0.264..43.823 rows=16954 loops=1)
                    -> Index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.262..39.101 rows=19237 loops=1)
    -> Covering index lookup on c using make (make=temp2.make, model=temp2.model, year=temp2.`year`)  (cost=0.30 rows=4) (actual time=0.003..0.006 rows=5 loops=3996)
 |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
```

4) With indexing on Cars(model), cost = 3477.39



```
ysql> create index Cars_model ON Cars(model);
uery OK, 0 rows affected (0.27 sec)
ecords: 0  Duplicates: 0  Warnings: 0

ysql> EXPLAIN ANALYZE SELECT c.VIN, temp2.Inventory_Score  FROM Cars c NATURAL JOIN   (SELECT c.make, c.model, c.year, (70* COUNT(c.VIN) / AVG(temp.total_not_sold)) AS Inventory_Score FROM Cars c,  (SELECT COUNT(
) AS total_not_sold FROM Cars c WHERE c.status = 'available') AS temp WHERE c.status = 'available' GROUP BY c.make, c.model, c.year) AS temp2;
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
 EXPLAIN



           |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
 -> Nested loop inner join  (cost=3477.39 rows=7581) (actual time=49.312..75.642 rows=19053 loops=1)
   -> Table scan on temp2  (cost=2170.44..2195.10 rows=1774) (actual time=49.287..50.129 rows=3996 loops=1)
       -> Materialize  (cost=2170.43..2170.43 rows=1774) (actual time=49.284..49.284 rows=3996 loops=1)
           -> Group aggregate: avg('16954'), count(c.VIN)  (cost=1993.01 rows=1774) (actual time=0.297..46.480 rows=3996 loops=1)
               -> Filter: (c.`status` = 'available')  (cost=1815.59 rows=1774) (actual time=0.290..39.418 rows=16954 loops=1)
                   -> Index scan on c using make  (cost=1815.59 rows=17742) (actual time=0.289..35.051 rows=19237 loops=1)
   -> Covering index lookup on c using make (make=temp2.make, model=temp2.model, year=temp2.`year`)  (cost=0.30 rows=4) (actual time=0.003..0.006 rows=5 loops=3996)
|
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
```

Report:

Index on Cars(make) and Cars(model) have the same cost as without indexing so we don’t choose them. We chose indexing model #2 (indexing on Cars(status)) because we have the lowest cost of 2623.53. Indexing on status in this query is essential for performance because it allows the database to quickly filter only the rows where status = 'available'. This index enables faster access to relevant rows improving efficiency in grouping and aggregation operations on make, model, and year. By focusing processing only on relevant rows, this indexing lowers computational cost.