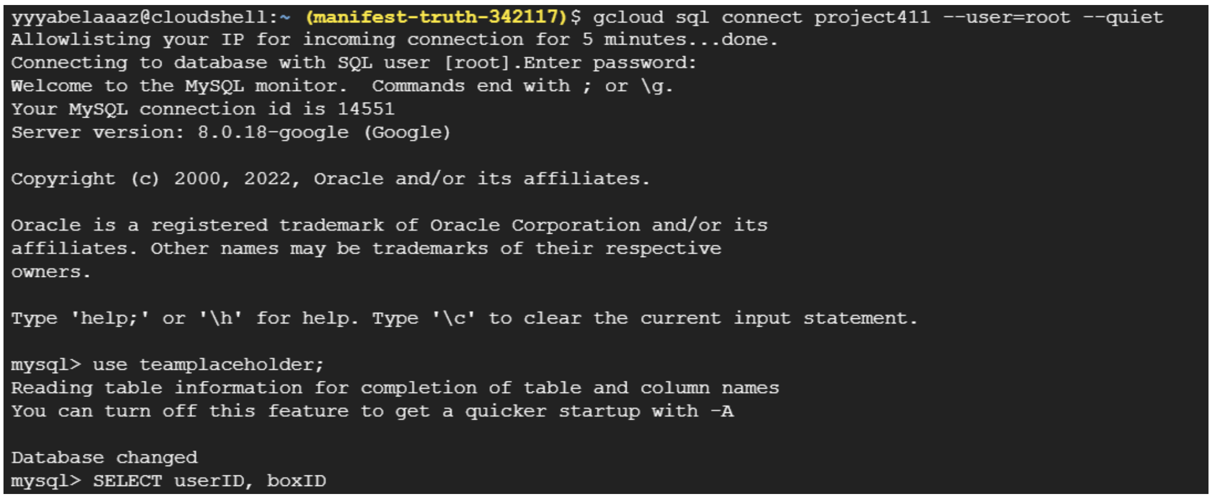
Stage 3-placeholder team40

Connection:

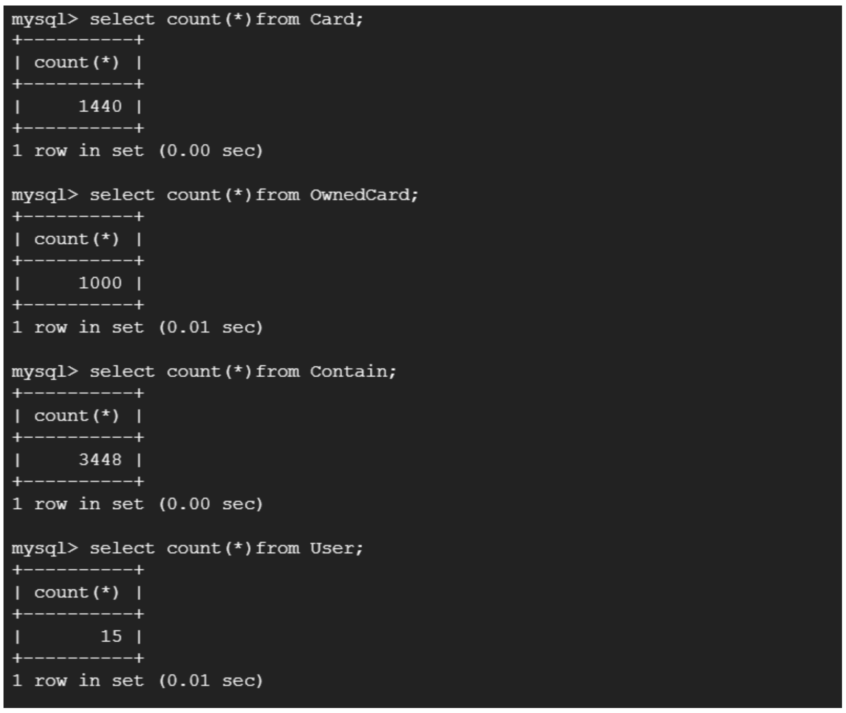


```
yyyabelaaaz@cloudshell:~ (manifest-truth-342117)$ gcloud sql connect project411 --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14551
Server version: 8.0.18-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use teamplaceholder;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT userID, boxID
```

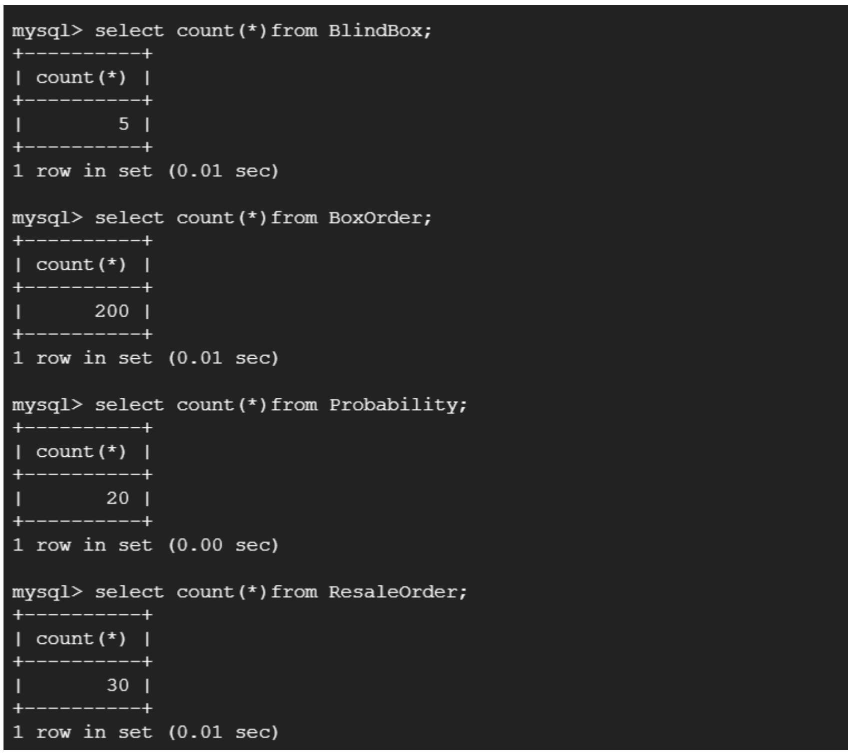
DDL:

```
DROP TABLE IF EXISTS User;
CREATE TABLE User(
userID int(16) NOT NULL,
password   varchar(50) NOT NULL,
u_name  varchar(50) NOT NULL,
PRIMARY KEY(userID)  );

DROP TABLE IF EXISTS Card;
CREATE TABLE Card(
cardNO int(16) NOT NULL,
rarity varchar(50) NOT NULL,
c_name  varchar(50) NOT NULL,
img  varchar(50) NOT NULL,
type  varchar(50) NOT NULL,
PRIMARY KEY(cardNO) );

DROP TABLE IF EXISTS BlindBox;
CREATE TABLE BlindBox(
boxID int(16) NOT NULL,
title   varchar(50) NOT NULL,
b_price  decimal(10,2) NOT NULL,
PRIMARY KEY(boxID) );

DROP TABLE IF EXISTS BoxOrder;
CREATE TABLE BoxOrder(
b_orderID int(16) NOT NULL,
userID  int(16) NOT NULL,
boxID   int(16) NOT NULL,
pay_datetime varchar(50) NOT NULL,
pay_amount  decimal(10,2) NOT NULL,
PRIMARY KEY(b_orderID),
FOREIGN KEY (userID) REFERENCES User(userID) ON DELETE CASCADE,
FOREIGN KEY (boxID) REFERENCES BlindBox(boxID) ON DELETE CASCADE);

DROP TABLE IF EXISTS OwnedCard;
CREATE TABLE OwnedCard(
cardID int(16) NOT NULL,
cardNO  int(16) NOT NULL,
userID  int(16) NOT NULL,
status varchar(50) NOT NULL,
c_price  decimal(10,2) NOT NULL,
PRIMARY KEY(cardID),
FOREIGN KEY (cardNO) REFERENCES Card(cardNO ) ON DELETE CASCADE,
FOREIGN KEY (userID) REFERENCES User(userID ) ON DELETE CASCADE);

DROP TABLE IF EXISTS ResaleOrder;
CREATE TABLE ResaleOrder(
r_orderID int(16) NOT NULL,
sellerID int(16) NOT NULL,
buyerID int(16) NOT NULL,
cardID int(16) NOT NULL,
trade_amount decimal(10, 2) NOT NULL,
trade_datetime varchar(50) NOT NULL,
PRIMARY KEY(r_orderID),
FOREIGN KEY (sellerID) REFERENCES User(userID) ON DELETE CASCADE,
FOREIGN KEY (buyerID) REFERENCES User(userID) ON DELETE CASCADE);

DROP TABLE IF EXISTS Probability;
CREATE TABLE Probability(
ruleNO int(16) NOT NULL,
boxID int(16) NOT NULL,
rarity varchar(50) NOT NULL,
prob  decimal(10, 3)  NOT NULL,
PRIMARY KEY(ruleNO ),
FOREIGN KEY (boxID)  REFERENCES BlindBox(boxID ) ON DELETE CASCADE);

DROP TABLE IF EXISTS Contain;
CREATE TABLE Contain(
cardNO int(16) NOT NULL,
boxID  int(16) NOT NULL,
PRIMARY KEY(cardNO,boxID ),
FOREIGN KEY (cardNO)  REFERENCES Card(cardNO ) ON DELETE CASCADE,
FOREIGN KEY (boxID) REFERENCES BlindBox(boxID ) ON DELETE CASCADE);
```

Count rows for each table:



```
mysql> select count(*)from Card;
+----------+
| count(*) |
+----------+
|     1440 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*)from OwnedCard;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*)from Contain;
+----------+
| count(*) |
+----------+
|     3448 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*)from User;
+----------+
| count(*) |
+----------+
|       15 |
+----------+
1 row in set (0.01 sec)
```



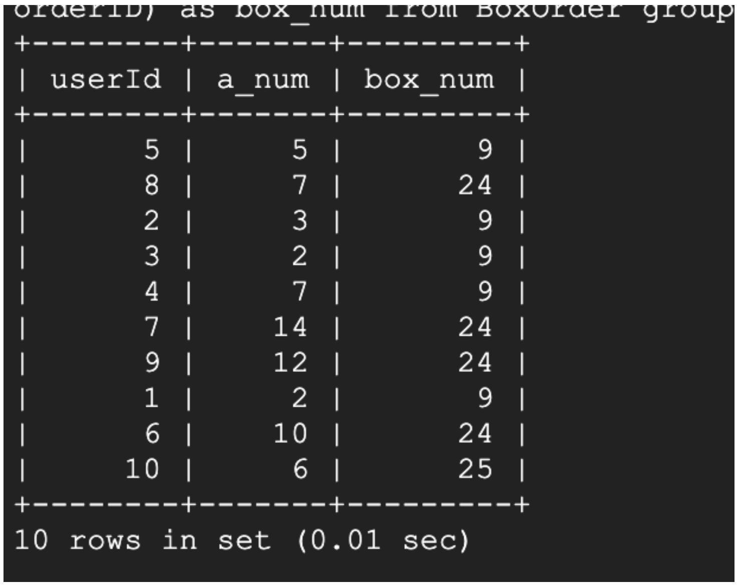
```
mysql> select count(*)from BlindBox;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*)from BoxOrder;
+----------+
| count(*) |
+----------+
|      200 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*)from Probability;
+----------+
| count(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*)from ResaleOrder;
+----------+
| count(*) |
+----------+
|       30 |
+----------+
1 row in set (0.01 sec)
```

Advanced query and index analysis:

1

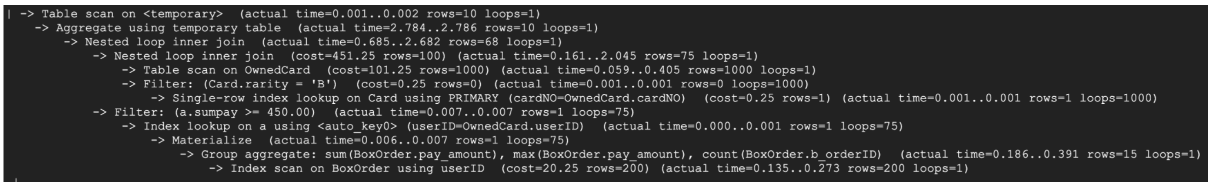
SELECT userId, count(status) as a_num, a.box_num
FROM Card natural join OwnedCard natural join (SELECT userID, sum(pay_amount) as sumpay, max(pay_amount) as maxpay, count(b_orderID) as box_num FROM BoxOrder GROUP BY userID) as a
WHERE a.sumpay >= 450 and rarity = 'B'
GROUP BY userID;

Query result:



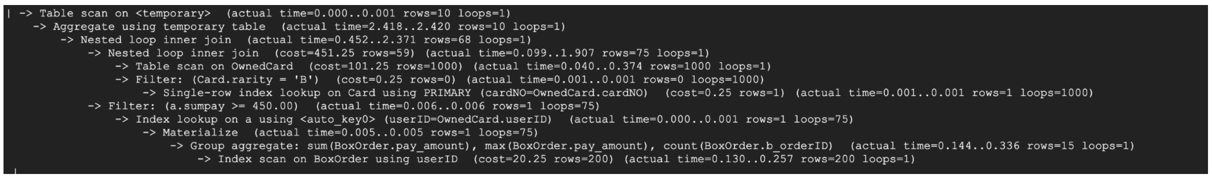
```
orderID) as box_num from BoxOrder group
+--------+-------+---------+
| userId | a_num | box_num |
+--------+-------+---------+
|      5 |     5 |       9 |
|      8 |     7 |      24 |
|      2 |     3 |       9 |
|      3 |     2 |       9 |
|      4 |     7 |       9 |
|      7 |    14 |      24 |
|      9 |    12 |      24 |
|      1 |     2 |       9 |
|      6 |    10 |      24 |
|     10 |     6 |      25 |
+--------+-------+---------+
10 rows in set (0.01 sec)
```

without adding index:



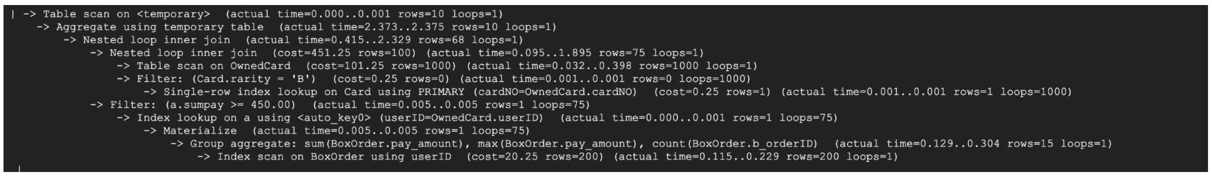
```
| -> Table scan on <temporary>  (actual time=0.001..0.002 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=2.784..2.786 rows=10 loops=1)
        -> Nested loop inner join  (actual time=0.685..2.682 rows=68 loops=1)
            -> Nested loop inner join  (cost=451.25 rows=100) (actual time=0.161..2.045 rows=75 loops=1)
                -> Table scan on OwnedCard  (cost=101.25 rows=1000) (actual time=0.059..0.405 rows=1000 loops=1)
                -> Filter: (Card.rarity = 'B')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1000)
                    -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
            -> Filter: (a.sumpay >= 450.00)  (actual time=0.007..0.007 rows=1 loops=75)
                -> Index lookup on a using <auto_key0> (userID=OwnedCard.userID)  (actual time=0.000..0.001 rows=1 loops=75)
                    -> Materialize  (actual time=0.006..0.007 rows=1 loops=75)
                        -> Group aggregate: sum(BoxOrder.pay_amount), max(BoxOrder.pay_amount), count(BoxOrder.b_orderID)  (actual time=0.186..0.391 rows=15 loops=1)
                            -> Index scan on BoxOrder using userID  (cost=20.25 rows=200) (actual time=0.135..0.273 rows=200 loops=1)
```

after adding index on Card(rarity):



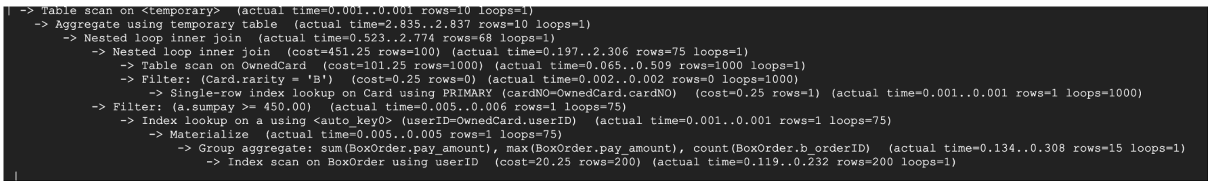
```
| -> Table scan on <temporary>  (actual time=0.000..0.001 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=2.418..2.420 rows=10 loops=1)
        -> Nested loop inner join  (actual time=0.452..2.371 rows=68 loops=1)
            -> Nested loop inner join  (cost=451.25 rows=59) (actual time=0.099..1.907 rows=75 loops=1)
                -> Table scan on OwnedCard  (cost=101.25 rows=1000) (actual time=0.040..0.374 rows=1000 loops=1)
                -> Filter: (Card.rarity = 'B')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1000)
                    -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
            -> Filter: (a.sumpay >= 450.00)  (actual time=0.006..0.006 rows=1 loops=75)
                -> Index lookup on a using <auto_key0> (userID=OwnedCard.userID)  (actual time=0.000..0.001 rows=1 loops=75)
                    -> Materialize  (actual time=0.005..0.005 rows=1 loops=75)
                        -> Group aggregate: sum(BoxOrder.pay_amount), max(BoxOrder.pay_amount), count(BoxOrder.b_orderID)  (actual time=0.144..0.336 rows=15 loops=1)
                            -> Index scan on BoxOrder using userID  (cost=20.25 rows=200) (actual time=0.130..0.257 rows=200 loops=1)
```

after adding index BoxOrder(pay_amount):



```
| -> Table scan on <temporary>  (actual time=0.000..0.001 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=2.373..2.375 rows=10 loops=1)
        -> Nested loop inner join  (actual time=0.415..2.329 rows=68 loops=1)
            -> Nested loop inner join  (cost=451.25 rows=100) (actual time=0.095..1.895 rows=75 loops=1)
                -> Table scan on OwnedCard  (cost=101.25 rows=1000) (actual time=0.032..0.398 rows=1000 loops=1)
                -> Filter: (Card.rarity = 'B')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=0 loops=1000)
                    -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
            -> Filter: (a.sumpay >= 450.00)  (actual time=0.005..0.005 rows=1 loops=75)
                -> Index lookup on a using <auto_key0> (userID=OwnedCard.userID)  (actual time=0.000..0.001 rows=1 loops=75)
                    -> Materialize  (actual time=0.005..0.005 rows=1 loops=75)
                        -> Group aggregate: sum(BoxOrder.pay_amount), max(BoxOrder.pay_amount), count(BoxOrder.b_orderID)  (actual time=0.129..0.304 rows=15 loops=1)
                            -> Index scan on BoxOrder using userID  (cost=20.25 rows=200) (actual time=0.115..0.229 rows=200 loops=1)
```

adding index OwnedCard(status):



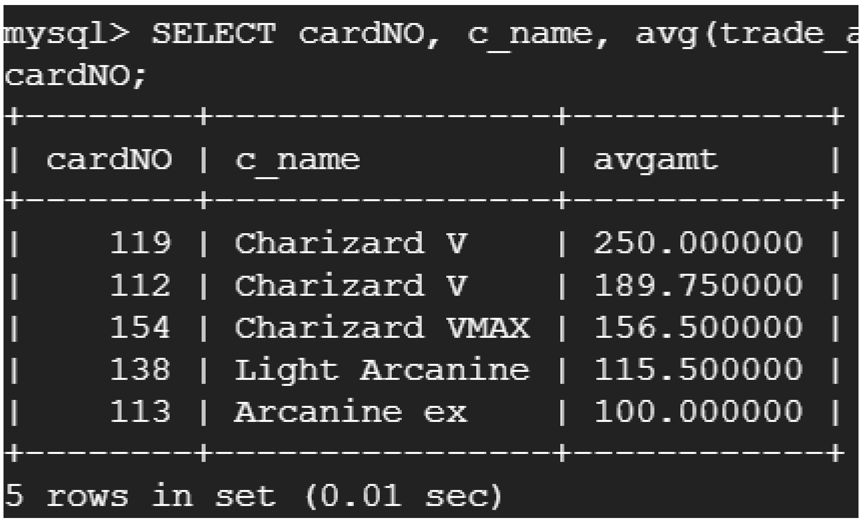
```
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=2.835..2.837 rows=10 loops=1)
        -> Nested loop inner join  (actual time=0.523..2.774 rows=68 loops=1)
            -> Nested loop inner join  (cost=451.25 rows=100) (actual time=0.197..2.306 rows=75 loops=1)
                -> Table scan on OwnedCard  (cost=101.25 rows=1000) (actual time=0.065..0.509 rows=1000 loops=1)
                -> Filter: (Card.rarity = 'B')  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=1000)
                    -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
            -> Filter: (a.sumpay >= 450.00)  (actual time=0.005..0.006 rows=1 loops=75)
                -> Index lookup on a using <auto_key0> (userID=OwnedCard.userID)  (actual time=0.001..0.001 rows=1 loops=75)
                    -> Materialize  (actual time=0.005..0.005 rows=1 loops=75)
                        -> Group aggregate: sum(BoxOrder.pay_amount), max(BoxOrder.pay_amount), count(BoxOrder.b_orderID)  (actual time=0.134..0.308 rows=15 loops=1)
                            -> Index scan on BoxOrder using userID  (cost=20.25 rows=200) (actual time=0.119..0.232 rows=200 loops=1)
 |
```

Analysis:
We've created three indexes for this query, they are Card(rarity), BoxOrder(pay_amount) and OwnedCard(status). Card(rarity) is used in outer where statement, BoxOrder(pay_amount) is used in nested query and aggregate functions like sum and count, and OwnedCard(status) is just used for outer's count, we want use this index to test whether it will improve the efficiency of the query. And through the results we can find when the adding index on BoxOrder(pay_amount), The efficiency improvement of sql statements is the most obvious. Especially the nested loop inner join. The second most obvious improvement in efficiency is when adding index on Card(rarity), the total time has decreased very obvious, but not show a clear difference in the query where (rarity) is. And for the index on OwnedCard(status), it even slower than no index query, we assume that because it is just used in aggregate function one time will not slow down the search.

2

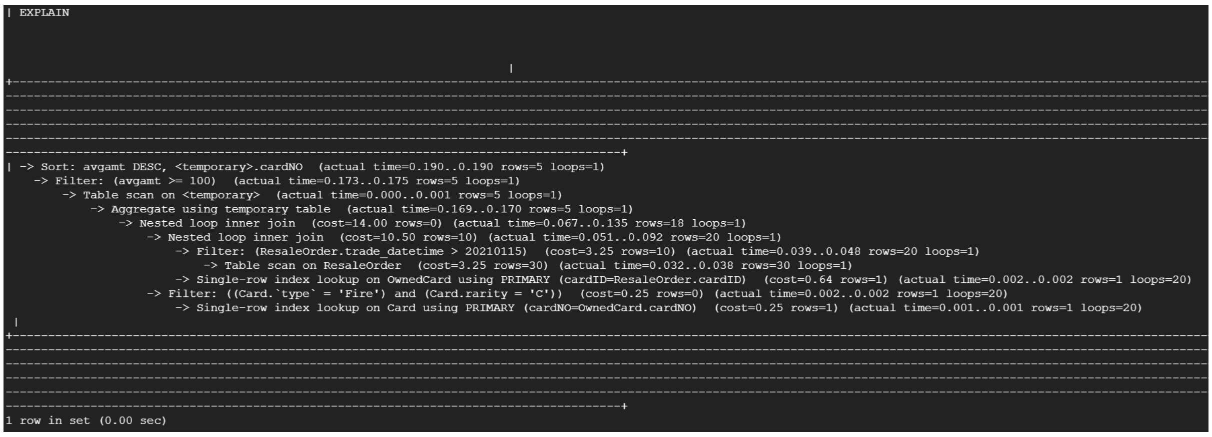
SELECT cardNO, c_name, avg(trade_amount) avgamt
FROM ResaleOrder NATURAL JOIN OwnedCard NATURAL JOIN Card
WHERE type = 'Fire' and rarity = 'C' and trade_datetime > 20210115
GROUP BY cardNO
HAVING avgamt >= 100
ORDER BY avgamt desc, cardNO

Query Result:



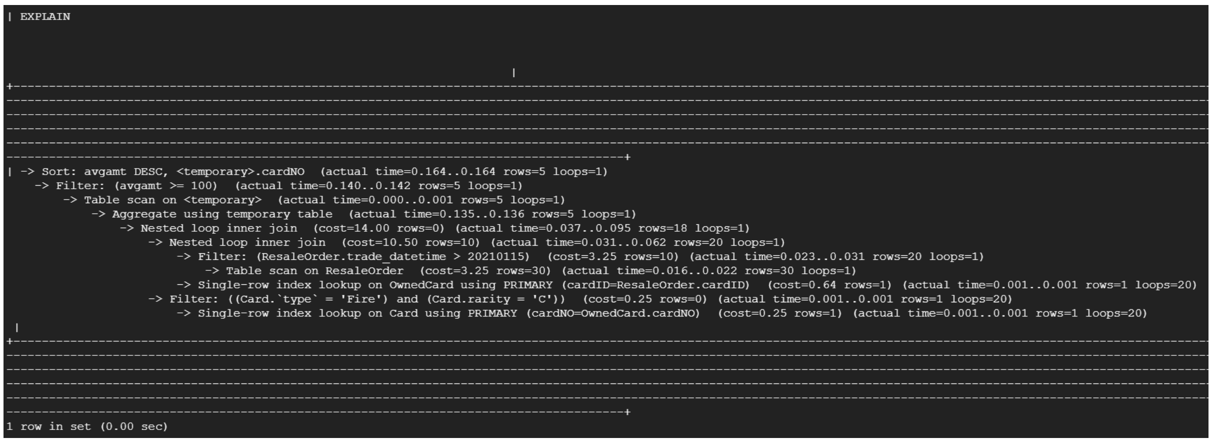
```
mysql> SELECT cardNO, c_name, avg(trade_a
cardNO;
+--------+----------------+------------+
| cardNO | c_name         | avgamt     |
+--------+----------------+------------+
|    119 | Charizard V    | 250.000000 |
|    112 | Charizard V    | 189.750000 |
|    154 | Charizard VMAX | 156.500000 |
|    138 | Light Arcanine | 115.500000 |
|    113 | Arcanine ex    | 100.000000 |
+--------+----------------+------------+
5 rows in set (0.01 sec)
```

Without adding index:



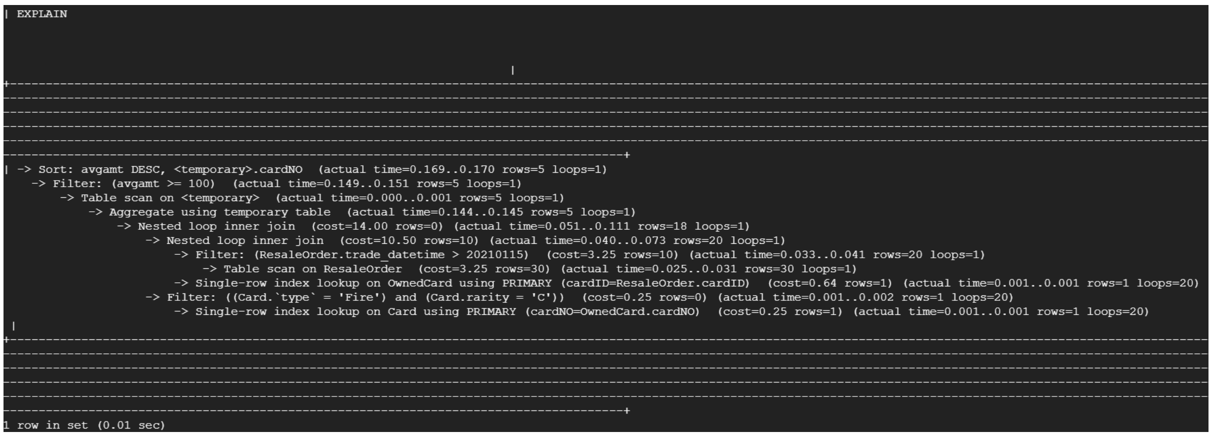
```
| EXPLAIN

                                                                     |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| -> Sort: avgamt DESC, <temporary>.cardNO  (actual time=0.190..0.190 rows=5 loops=1)
    -> Filter: (avgamt >= 100)  (actual time=0.173..0.175 rows=5 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.001 rows=5 loops=1)
            -> Aggregate using temporary table  (actual time=0.169..0.170 rows=5 loops=1)
                -> Nested loop inner join  (cost=14.00 rows=0) (actual time=0.067..0.135 rows=18 loops=1)
                    -> Nested loop inner join  (cost=10.50 rows=10) (actual time=0.051..0.092 rows=20 loops=1)
                        -> Filter: (ResaleOrder.trade_datetime > 20210115)  (cost=3.25 rows=10) (actual time=0.039..0.048 rows=20 loops=1)
                            -> Table scan on ResaleOrder  (cost=3.25 rows=30) (actual time=0.032..0.038 rows=30 loops=1)
                        -> Single-row index lookup on OwnedCard using PRIMARY (cardID=ResaleOrder.cardID)  (cost=0.64 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
                    -> Filter: ((Card.`type` = 'Fire') and (Card.rarity = 'C'))  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=1 loops=20)
                        -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
 |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

add an index on Card(type):



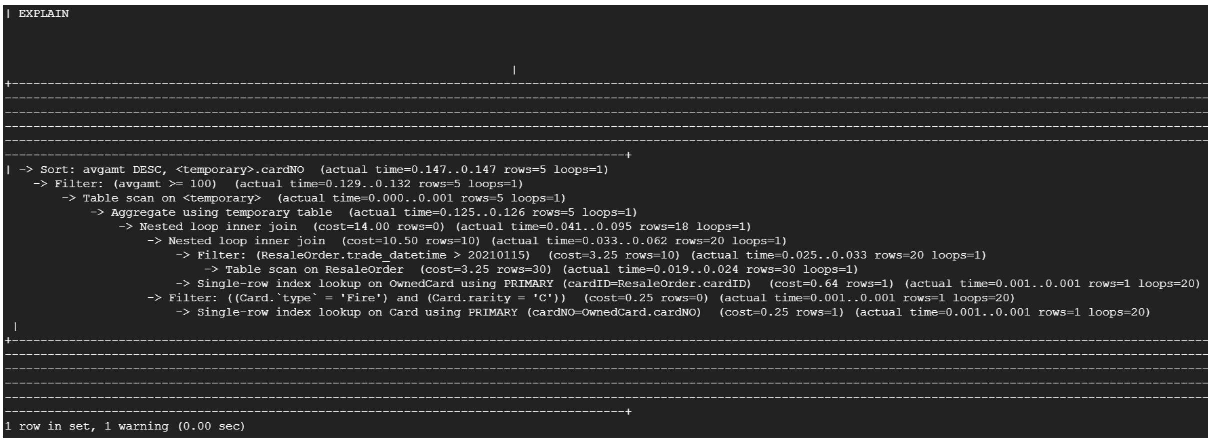
```
| EXPLAIN

                                                                     |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| -> Sort: avgamt DESC, <temporary>.cardNO  (actual time=0.164..0.164 rows=5 loops=1)
    -> Filter: (avgamt >= 100)  (actual time=0.140..0.142 rows=5 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.001 rows=5 loops=1)
            -> Aggregate using temporary table  (actual time=0.135..0.136 rows=5 loops=1)
                -> Nested loop inner join  (cost=14.00 rows=0) (actual time=0.037..0.095 rows=18 loops=1)
                    -> Nested loop inner join  (cost=10.50 rows=10) (actual time=0.031..0.062 rows=20 loops=1)
                        -> Filter: (ResaleOrder.trade_datetime > 20210115)  (cost=3.25 rows=10) (actual time=0.023..0.031 rows=20 loops=1)
                            -> Table scan on ResaleOrder  (cost=3.25 rows=30) (actual time=0.016..0.022 rows=30 loops=1)
                        -> Single-row index lookup on OwnedCard using PRIMARY (cardID=ResaleOrder.cardID)  (cost=0.64 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
                    -> Filter: ((Card.`type` = 'Fire') and (Card.rarity = 'C'))  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=1 loops=20)
                        -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
 |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

add an index on Card(rarity):



```
| EXPLAIN

                                                                     |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| -> Sort: avgamt DESC, <temporary>.cardNO  (actual time=0.169..0.170 rows=5 loops=1)
    -> Filter: (avgamt >= 100)  (actual time=0.149..0.151 rows=5 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.001 rows=5 loops=1)
            -> Aggregate using temporary table  (actual time=0.144..0.145 rows=5 loops=1)
                -> Nested loop inner join  (cost=14.00 rows=0) (actual time=0.051..0.111 rows=18 loops=1)
                    -> Nested loop inner join  (cost=10.50 rows=10) (actual time=0.040..0.073 rows=20 loops=1)
                        -> Filter: (ResaleOrder.trade_datetime > 20210115)  (cost=3.25 rows=10) (actual time=0.033..0.041 rows=20 loops=1)
                            -> Table scan on ResaleOrder  (cost=3.25 rows=30) (actual time=0.025..0.031 rows=30 loops=1)
                        -> Single-row index lookup on OwnedCard using PRIMARY (cardID=ResaleOrder.cardID)  (cost=0.64 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
                    -> Filter: ((Card.`type` = 'Fire') and (Card.rarity = 'C'))  (cost=0.25 rows=0) (actual time=0.001..0.002 rows=1 loops=20)
                        -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
 |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

added an index on ResaleOrder(trade_datetime):



```
| EXPLAIN



                                                                        |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------+
| -> Sort: avgamt DESC, <temporary>.cardNO  (actual time=0.147..0.147 rows=5 loops=1)
    -> Filter: (avgamt >= 100)  (actual time=0.129..0.132 rows=5 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.001 rows=5 loops=1)
            -> Aggregate using temporary table  (actual time=0.125..0.126 rows=5 loops=1)
                -> Nested loop inner join  (cost=14.00 rows=0) (actual time=0.041..0.095 rows=18 loops=1)
                    -> Nested loop inner join  (cost=10.50 rows=10) (actual time=0.033..0.062 rows=20 loops=1)
                        -> Filter: (ResaleOrder.trade_datetime > 20210115)  (cost=3.25 rows=10) (actual time=0.025..0.033 rows=20 loops=1)
                            -> Table scan on ResaleOrder  (cost=3.25 rows=30) (actual time=0.019..0.024 rows=30 loops=1)
                        -> Single-row index lookup on OwnedCard using PRIMARY (cardID=ResaleOrder.cardID)  (cost=0.64 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
                    -> Filter: ((Card.`type` = 'Fire') and (Card.rarity = 'C'))  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=1 loops=20)
                        -> Single-row index lookup on Card using PRIMARY (cardNO=OwnedCard.cardNO)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

Analysis:
We created indexes for the three where conditions of this query respectively and compared them with the original query. We found that the time displayed in the lower left corner is 0.00 except when rarity has an index which is 0.01. This may be because our query takes too little time to reflect the difference. And each time running the same query takes different time. From the details of explain, the speed of queries with indexes is slightly faster. Among them, the query with index on trade_datetime is the fastest, and the query with index on rarity is the slowest, this may be because we only have one rarity of the cards in the table ResaleOrder, the index is not fully working for the query, however, the index of trade_datetime can help select the data. We're not sure why adding an index doesn't reduce the cost of the query, It may be that we did not select the appropriate columns to add indexes. We have considered adding indexes to the columns of GROUP BY or JOIN, but the columns of these operations are all primary keys or foreign keys, and already have indexes.