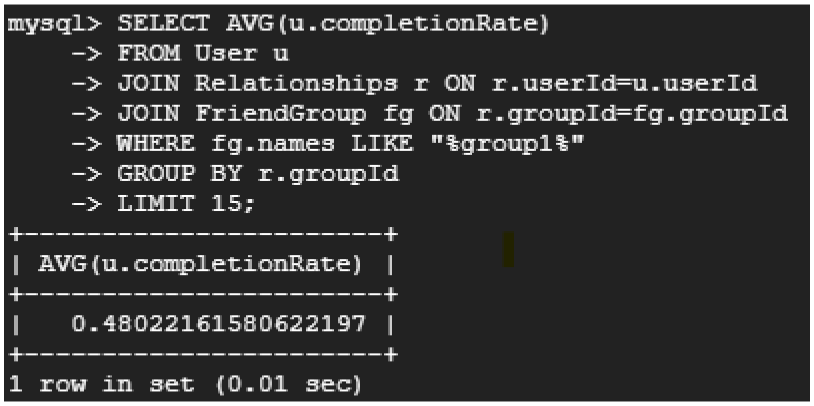
The first Advanced Query is to show the average completion rates of tasks within a selected friend group (as you cannot see strangers' stats). The query is as follows (note the group name at the moment is fixed; would be variable in an application):

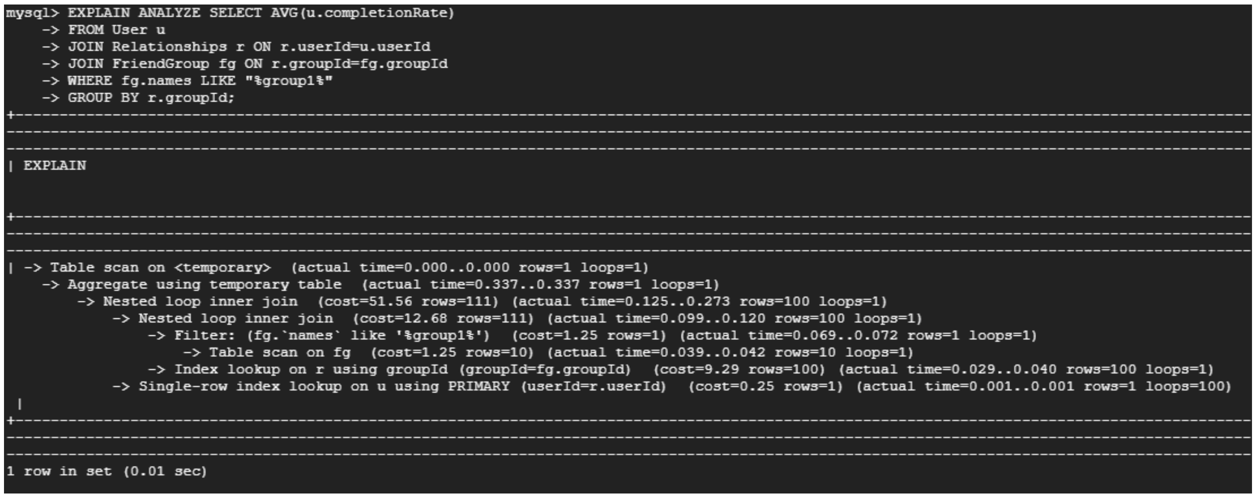
```sql
SELECT AVG(u.completionRate)
FROM User u
JOIN Relationships r ON r.userId=u.userId
JOIN FriendGroup fg ON r.groupId=fg.groupId
WHERE fg.names LIKE "%group1%"
GROUP BY r.groupId
LIMIT 15;
```

The results of running this query is:



```
mysql> SELECT AVG(u.completionRate)
    -> FROM User u
    -> JOIN Relationships r ON r.userId=u.userId
    -> JOIN FriendGroup fg ON r.groupId=fg.groupId
    -> WHERE fg.names LIKE "%group1%"
    -> GROUP BY r.groupId
    -> LIMIT 15;
+-----------------------+
| AVG(u.completionRate) |
+-----------------------+
|   0.48022161580622197 |
+-----------------------+
1 row in set (0.01 sec)
```

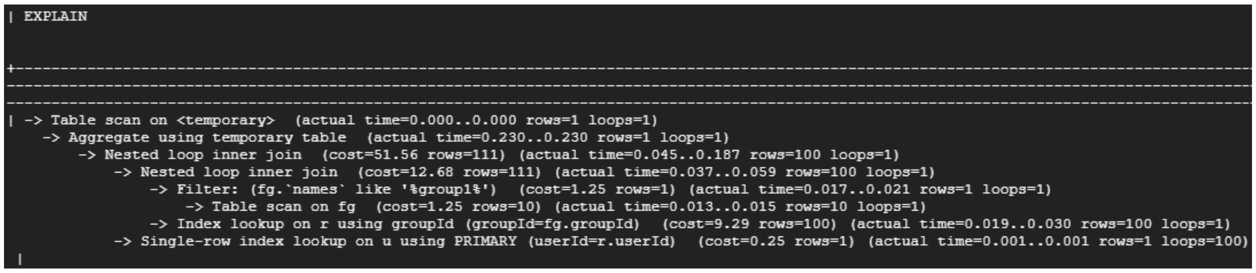
The performance of this query prior to indexing is as follows:



```
mysql> EXPLAIN ANALYZE SELECT AVG(u.completionRate)
    -> FROM User u
    -> JOIN Relationships r ON r.userId=u.userId
    -> JOIN FriendGroup fg ON r.groupId=fg.groupId
    -> WHERE fg.names LIKE "%group1%"
    -> GROUP BY r.groupId;
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=0.000..0.000 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=0.337..0.337 rows=1 loops=1)
        -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.125..0.273 rows=100 loops=1)
            -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.099..0.120 rows=100 loops=1)
                -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.069..0.072 rows=1 loops=1)
                    -> Table scan on fg  (cost=1.25 rows=10) (actual time=0.039..0.042 rows=10 loops=1)
                -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.029..0.040 rows=100 loops=1)
            -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
 |
+------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

From this query, we can see that we are looking for the average of the completionRate by groupID. Thus, we start by indexing this completionRate attribute to make the lookup quicker when using aggregate functions. The results of creating an index on completion rate are as follows:

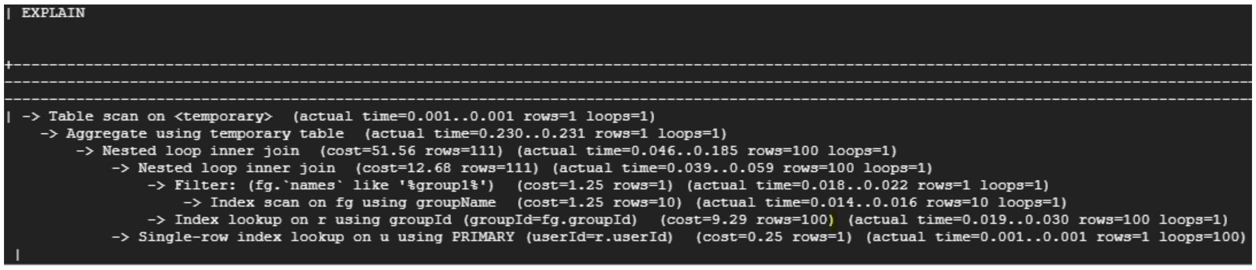


```
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=0.000..0.000 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=0.230..0.230 rows=1 loops=1)
        -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.045..0.187 rows=100 loops=1)
            -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.037..0.059 rows=100 loops=1)
                -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.017..0.021 rows=1 loops=1)
                    -> Table scan on fg  (cost=1.25 rows=10) (actual time=0.013..0.015 rows=10 loops=1)
                -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.019..0.030 rows=100 loops=1)
            -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
 |
```

From the results, we can see that the performance does seem to be very marginally better in the Aggregate section (0.337 to 0.230). This is because the aggregation means a lot of looking up of the groups, thus making lots of accesses to the group names index.

We also can notice that we filter out the results by selecting for only one group name. Thus, we think that indexing can be helpful when looking to see if an entry has the specific group name. The results are as follows:

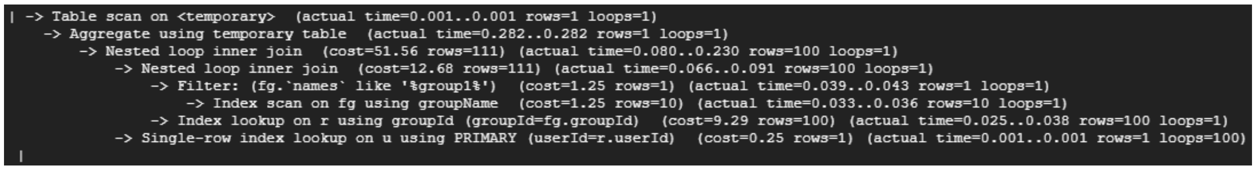


```
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=0.230..0.231 rows=1 loops=1)
        -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.046..0.185 rows=100 loops=1)
            -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.039..0.059 rows=100 loops=1)
                -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.018..0.022 rows=1 loops=1)
                    -> Index scan on fg using groupName  (cost=1.25 rows=10) (actual time=0.014..0.016 rows=10 loops=1)
                -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.019..0.030 rows=100 loops=1)
            -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
 |
```

From the data above, we see that in the Filter section, there is a significant time boost (0.069 to 0.014). This is likely because every time we see a line, we have to look up its groupName attribute. Thus, having this index helps speed up the access.

We see that the common trend is for groupId and userId being sufficient. However, we notice that in the Relationships Table, the groupId and userId form to be one primary key. However, the groupId does not necessarily need to be unique in this joined table, only the userId. Thus, creating an index on groupId in the Relationships table may speed up by making it non-unique. The results are as follow:



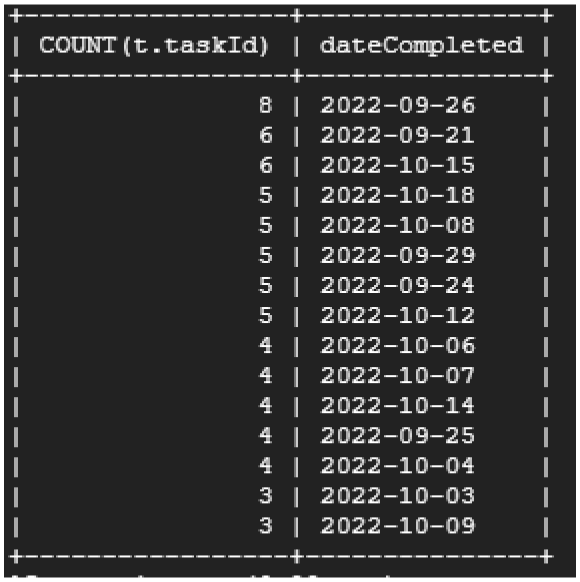
```
| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=0.282..0.282 rows=1 loops=1)
        -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.080..0.230 rows=100 loops=1)
            -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.066..0.091 rows=100 loops=1)
                -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.039..0.043 rows=1 loops=1)
                    -> Index scan on fg using groupName  (cost=1.25 rows=10) (actual time=0.033..0.036 rows=10 loops=1)
                -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.025..0.038 rows=100 loops=1)
            -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
 |
```

We see that there is no real significant increase in the nested loop performance in total. We see that there are two options: the userId Primary index in User Table and groupId Primary index in FriendGroup Table is good enough for this option, or these other indices cause more overhead than benefits.

The second Advanced Query is to show the days with the most completed tasks within a selected friend group (as you cannot see strangers' stats). The query is as follows (note the group name at the moment is fixed; would be variable in an application):

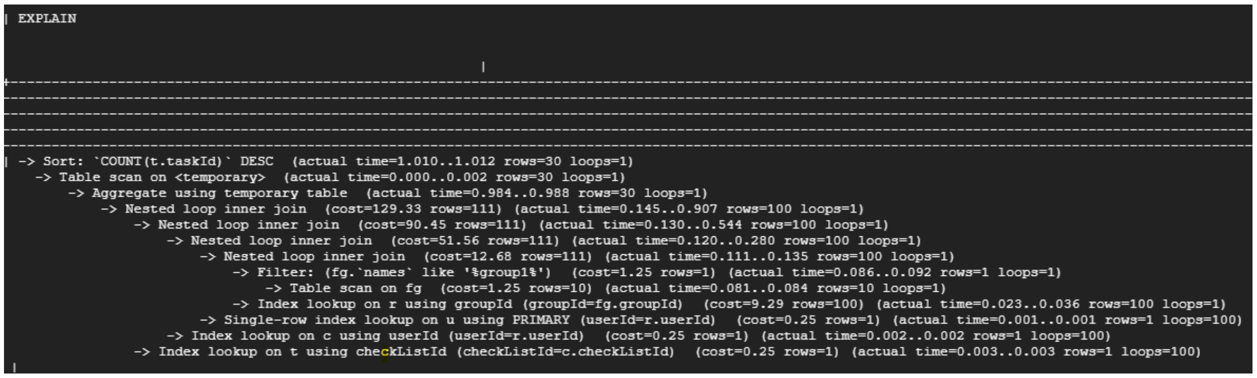
```sql
SELECT COUNT(t.taskId), dateCompleted
FROM User u
JOIN Checklist c ON c.userId=u.userId
JOIN Tasks t ON t.checkListId=c.checkListId
JOIN Relationships r ON r.userId=u.userId
JOIN FriendGroup fg ON r.groupId=fg.groupId
WHERE fg.names LIKE "%group1%"
GROUP BY dateCompleted
ORDER BY COUNT(t.taskId) DESC
LIMIT 15;
```

The results of this query are as follows:



```
+-----------------+---------------+
| COUNT(t.taskId) | dateCompleted |
+-----------------+---------------+
|               8 | 2022-09-26    |
|               6 | 2022-09-21    |
|               6 | 2022-10-15    |
|               5 | 2022-10-18    |
|               5 | 2022-10-08    |
|               5 | 2022-09-29    |
|               5 | 2022-09-24    |
|               5 | 2022-10-12    |
|               4 | 2022-10-06    |
|               4 | 2022-10-07    |
|               4 | 2022-10-14    |
|               4 | 2022-09-25    |
|               4 | 2022-10-04    |
|               3 | 2022-10-03    |
|               3 | 2022-10-09    |
+-----------------+---------------+
```

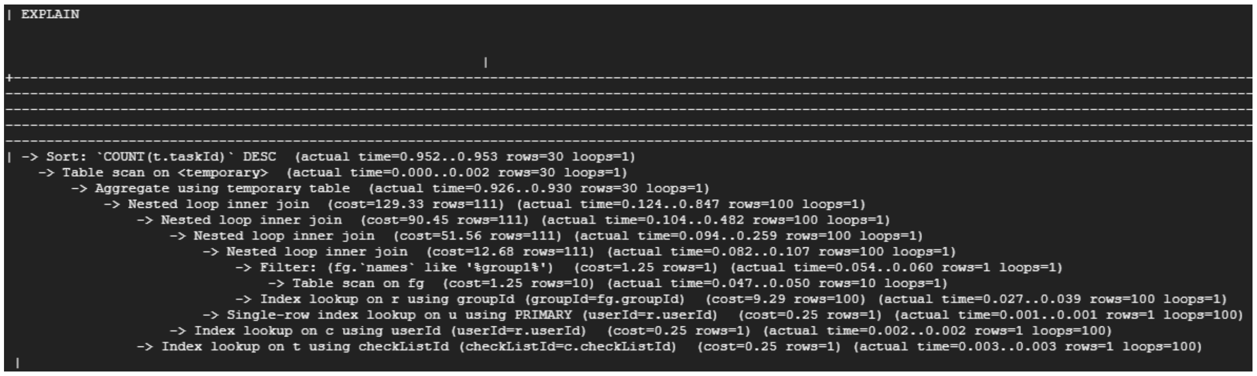
The performance of this query normally is:



```
| EXPLAIN

                                                  |
+-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
| -> Sort: `COUNT(t.taskId)` DESC  (actual time=1.010..1.012 rows=30 loops=1)
    -> Table scan on <temporary>  (actual time=0.000..0.002 rows=30 loops=1)
        -> Aggregate using temporary table  (actual time=0.984..0.988 rows=30 loops=1)
            -> Nested loop inner join  (cost=129.33 rows=111) (actual time=0.145..0.907 rows=100 loops=1)
                -> Nested loop inner join  (cost=90.45 rows=111) (actual time=0.130..0.544 rows=100 loops=1)
                    -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.120..0.280 rows=100 loops=1)
                        -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.111..0.135 rows=100 loops=1)
                            -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.086..0.092 rows=1 loops=1)
                                -> Table scan on fg  (cost=1.25 rows=10) (actual time=0.081..0.084 rows=10 loops=1)
                            -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.023..0.036 rows=100 loops=1)
                        -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
                    -> Index lookup on c using userId (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=100)
                -> Index lookup on t using checkListId (checkListId=c.checkListId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=100)
 |
```

First, we notice that we are searching and aggregating over the dateCompleted attribute in Tasks. So naturally, to make the lookups for this faster, we index it. The results are:

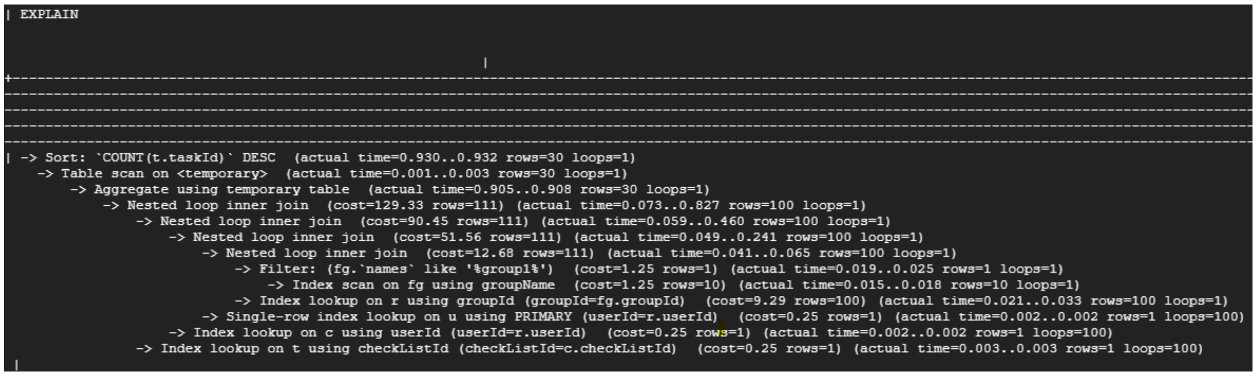


```
| EXPLAIN


                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: `COUNT(t.taskId)` DESC  (actual time=0.952..0.953 rows=30 loops=1)
    -> Table scan on <temporary>  (actual time=0.000..0.002 rows=30 loops=1)
        -> Aggregate using temporary table  (actual time=0.926..0.930 rows=30 loops=1)
            -> Nested loop inner join  (cost=129.33 rows=111) (actual time=0.124..0.847 rows=100 loops=1)
                -> Nested loop inner join  (cost=90.45 rows=111) (actual time=0.104..0.482 rows=100 loops=1)
                    -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.094..0.259 rows=100 loops=1)
                        -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.082..0.107 rows=100 loops=1)
                            -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.054..0.060 rows=1 loops=1)
                                -> Table scan on fg  (cost=1.25 rows=10) (actual time=0.047..0.050 rows=10 loops=1)
                            -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.027..0.039 rows=100 loops=1)
                        -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
                    -> Index lookup on c using userId (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=100)
                -> Index lookup on t using checkListId (checkListId=c.checkListId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=100)
 |
```

In the Aggregate using temporary table section, we do not see a significant time performance. This is likely because an entire scan is happening upon the table, so having a special index for dateCompleted is insignificant.

Next, we see that we are only looking for groups of certain names. Thus, indexing them may make the search faster in the WHERE clause of the query. The results are:

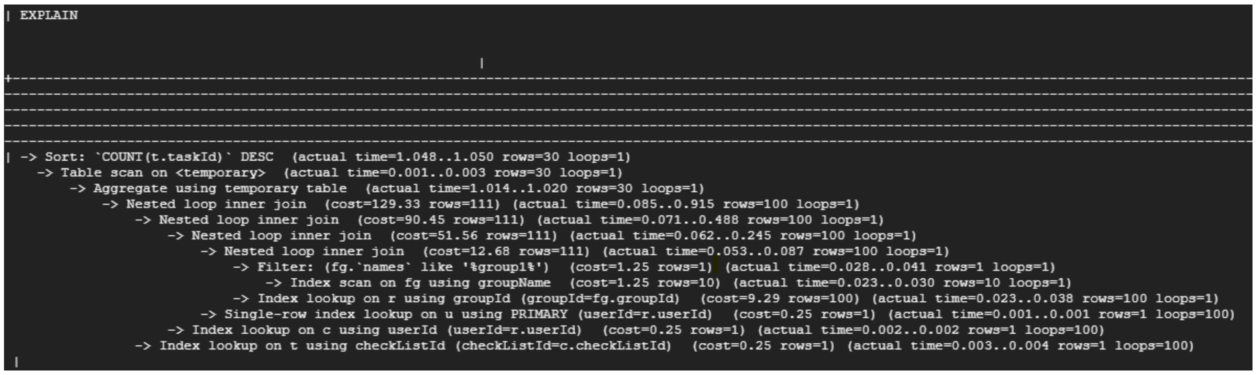


```
| EXPLAIN


                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: `COUNT(t.taskId)` DESC  (actual time=0.930..0.932 rows=30 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=30 loops=1)
        -> Aggregate using temporary table  (actual time=0.905..0.908 rows=30 loops=1)
            -> Nested loop inner join  (cost=129.33 rows=111) (actual time=0.073..0.827 rows=100 loops=1)
                -> Nested loop inner join  (cost=90.45 rows=111) (actual time=0.059..0.460 rows=100 loops=1)
                    -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.049..0.241 rows=100 loops=1)
                        -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.041..0.065 rows=100 loops=1)
                            -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.019..0.025 rows=1 loops=1)
                                -> Index scan on fg using groupName  (cost=1.25 rows=10) (actual time=0.015..0.018 rows=10 loops=1)
                            -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.021..0.033 rows=100 loops=1)
                        -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=100)
                    -> Index lookup on c using userId (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=100)
                -> Index lookup on t using checkListId (checkListId=c.checkListId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=100)
 |
```

We see that once again the filter time of fg.names seems to go down by a significant margin (0.054 to 0.019). This may not be significant, but the percentage decrease is over half. Thus, it seems to work in this scenario.

Finally, we try to index by groupId as a Non-unique index in the Relationship clause as before. The results are:



```
| EXPLAIN


                                                           |
+--------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: `COUNT(t.taskId)` DESC  (actual time=1.048..1.050 rows=30 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=30 loops=1)
        -> Aggregate using temporary table  (actual time=1.014..1.020 rows=30 loops=1)
            -> Nested loop inner join  (cost=129.33 rows=111) (actual time=0.085..0.915 rows=100 loops=1)
                -> Nested loop inner join  (cost=90.45 rows=111) (actual time=0.071..0.488 rows=100 loops=1)
                    -> Nested loop inner join  (cost=51.56 rows=111) (actual time=0.062..0.245 rows=100 loops=1)
                        -> Nested loop inner join  (cost=12.68 rows=111) (actual time=0.053..0.087 rows=100 loops=1)
                            -> Filter: (fg.`names` like '%group1%')  (cost=1.25 rows=1) (actual time=0.028..0.041 rows=1 loops=1)
                                -> Index scan on fg using groupName  (cost=1.25 rows=10) (actual time=0.023..0.030 rows=10 loops=1)
                            -> Index lookup on r using groupId (groupId=fg.groupId)  (cost=9.29 rows=100) (actual time=0.023..0.038 rows=100 loops=1)
                        -> Single-row index lookup on u using PRIMARY (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100)
                    -> Index lookup on c using userId (userId=r.userId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=100)
                -> Index lookup on t using checkListId (checkListId=c.checkListId)  (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=100)
 |
```

We see that in the inner loop for joining, there does not seem to be any increase in performance. This is probably because the groupId and userId having their own primary indices in their own tables is more efficient for the performance.