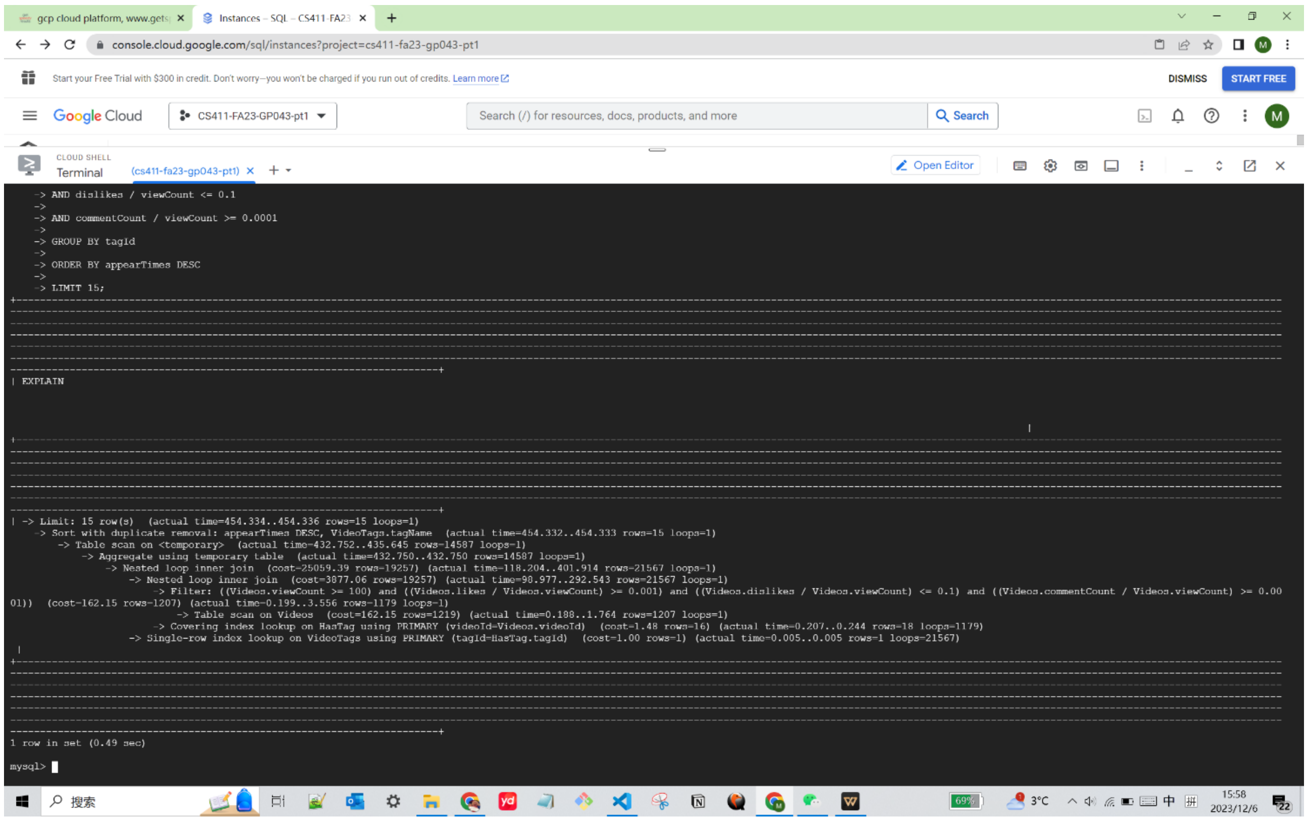
CS411 Stage3 Improvement

Team043-InfelPhira

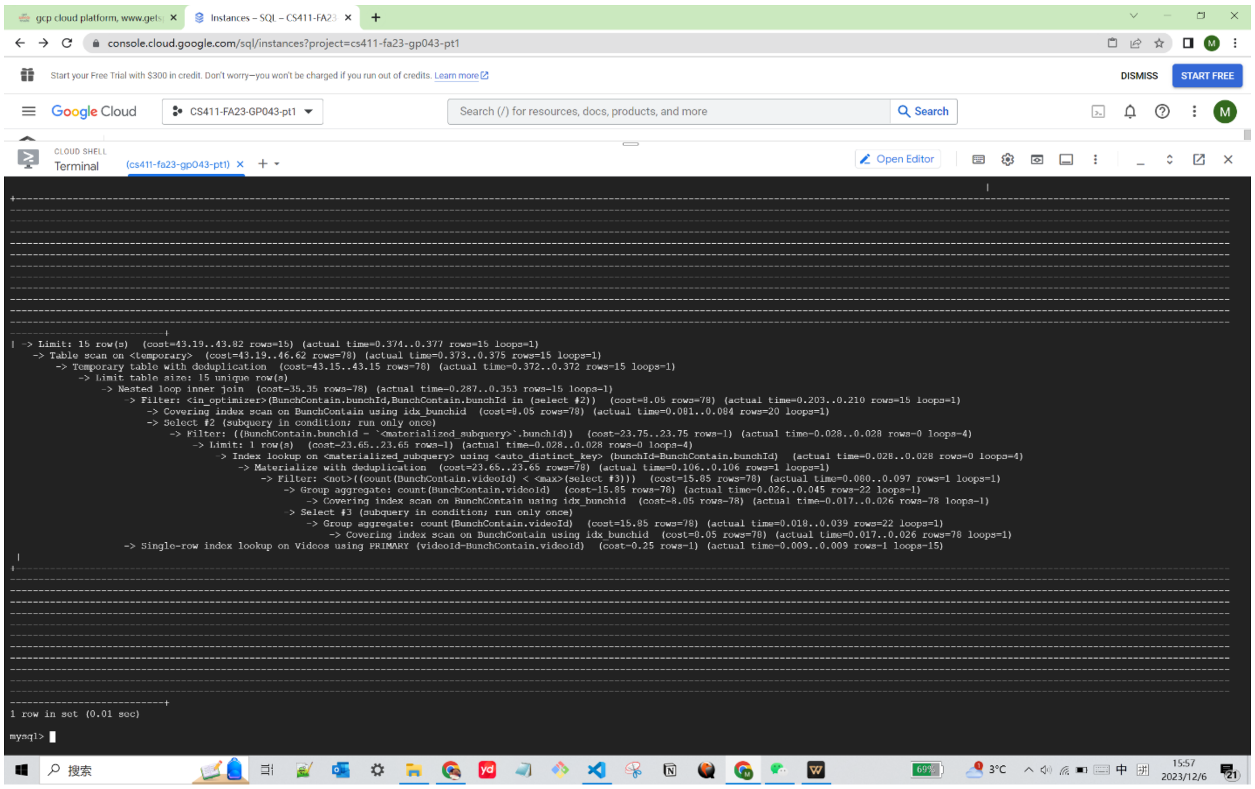
Original screenshots of the explain analyze commands without creating any index.

First Analysis:



```
    -> AND dislikes / viewCount <= 0.1
    ->
    -> AND commentCount / viewCount >= 0.0001
    ->
    -> GROUP BY tagId
    ->
    -> ORDER BY appearTimes DESC
    ->
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=454.334..454.336 rows=15 loops=1)
    -> Sort with duplicate removal: appearTimes DESC, VideoTags.tagName  (actual time=454.332..454.333 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=432.752..435.645 rows=14587 loops=1)
            -> Aggregate using temporary table  (actual time=432.750..432.750 rows=14587 loops=1)
                -> Nested loop inner join  (cost=25059.39 rows=19257) (actual time=118.204..401.914 rows=21567 loops=1)
                    -> Nested loop inner join  (cost=3877.06 rows=19257) (actual time=98.977..292.543 rows=21567 loops=1)
                        -> Filter: ((Videos.viewCount >= 100) and ((Videos.likes / Videos.viewCount) >= 0.001) and ((Videos.dislikes / Videos.viewCount) <= 0.1) and ((Videos.commentCount / Videos.viewCount) >= 0.00
01))  (cost=162.15 rows=1207) (actual time=0.199..3.556 rows=1179 loops=1)
                            -> Table scan on Videos  (cost=162.15 rows=1219) (actual time=0.188..1.764 rows=1207 loops=1)
                        -> Covering index lookup on HasTag using PRIMARY (videoId=Videos.videoId)  (cost=1.48 rows=16) (actual time=0.207..0.244 rows=18 loops=1179)
                    -> Single-row index lookup on VideoTags using PRIMARY (tagId=HasTag.tagId)  (cost=1.00 rows=1) (actual time=0.005..0.005 rows=1 loops=21567)
 |
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
1 row in set (0.49 sec)

mysql> 
```

Second Analysis:

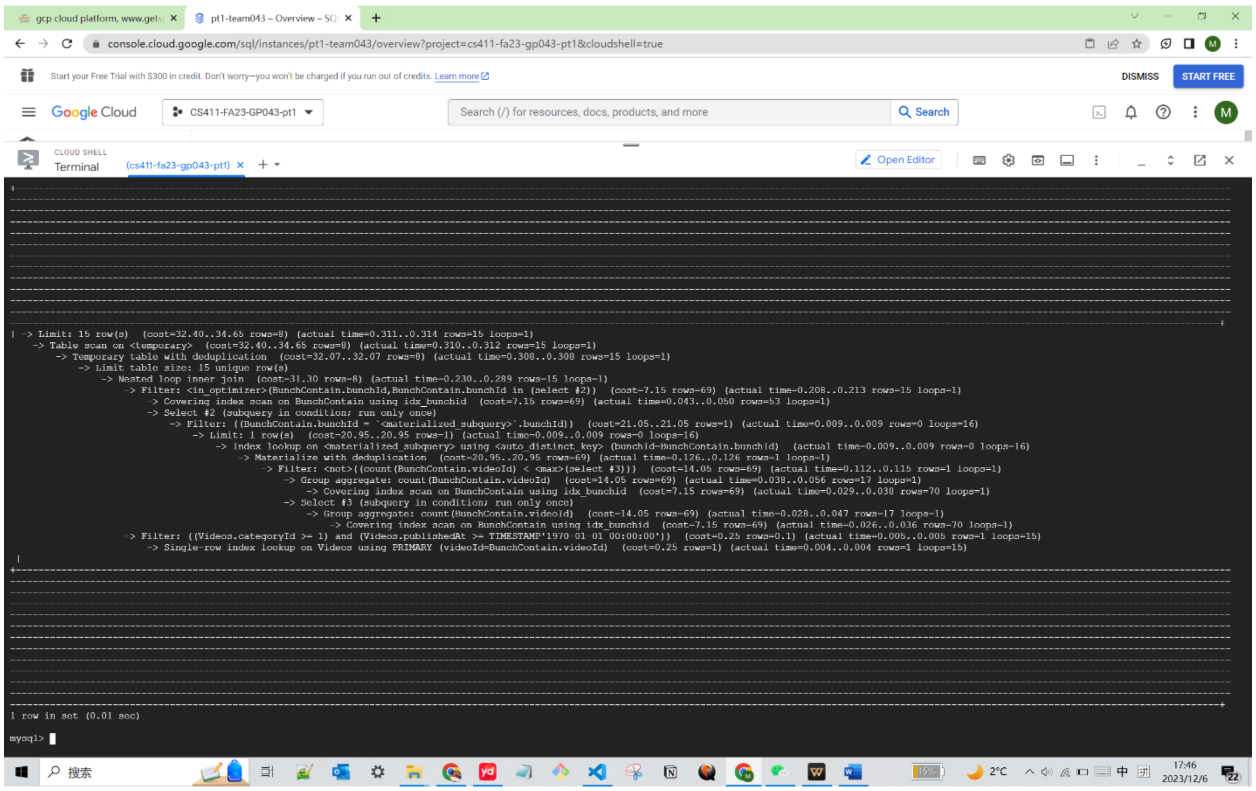


```
+-----------------------------------------------------------------------------------------------
-----------------------------+
| -> Limit: 15 row(s)  (cost=43.19..43.82 rows=15) (actual time=0.374..0.377 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=43.19..46.62 rows=78) (actual time=0.373..0.375 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=43.15..43.15 rows=78) (actual time=0.372..0.372 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=35.35 rows=78) (actual time=0.287..0.353 rows=15 loops=1)
                    -> Filter: <in_optimizer>(BunchContain.bunchId,BunchContain.bunchId in (select #2))  (cost=8.05 rows=78) (actual time=0.203..0.210 rows=15 loops=1)
                        -> Covering index scan on BunchContain using idx_bunchid  (cost=8.05 rows=78) (actual time=0.081..0.084 rows=20 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Filter: ((BunchContain.bunchId = `<materialized_subquery>`.bunchId))  (cost=23.75..23.75 rows=1) (actual time=0.028..0.028 rows=0 loops=4)
                                -> Limit: 1 row(s)  (cost=23.65..23.65 rows=1) (actual time=0.028..0.028 rows=0 loops=4)
                                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (bunchId=BunchContain.bunchId)  (actual time=0.028..0.028 rows=0 loops=4)
                                        -> Materialize with deduplication  (cost=23.65..23.65 rows=78) (actual time=0.106..0.106 rows=1 loops=1)
                                            -> Filter: <not>((count(BunchContain.videoId) < <max>(select #3)))  (cost=15.85 rows=78) (actual time=0.080..0.097 rows=1 loops=1)
                                                -> Group aggregate: count(BunchContain.videoId)  (cost=15.85 rows=78) (actual time=0.026..0.045 rows=22 loops=1)
                                                    -> Covering index scan on BunchContain using idx_bunchid  (cost=8.05 rows=78) (actual time=0.017..0.026 rows=78 loops=1)
                                                -> Select #3 (subquery in condition; run only once)
                                                    -> Group aggregate: count(BunchContain.videoId)  (cost=15.85 rows=78) (actual time=0.018..0.039 rows=22 loops=1)
                                                        -> Covering index scan on BunchContain using idx_bunchid  (cost=8.05 rows=78) (actual time=0.017..0.026 rows=78 loops=1)
                    -> Single-row index lookup on Videos using PRIMARY (videoId=BunchContain.videoId)  (cost=0.25 rows=1) (actual time=0.009..0.009 rows=1 loops=15)
 |
+-----------------------------------------------------------------------------------------------
-----------------------------+
1 row in set (0.01 sec)

mysql> 
```

Change Second Analysis:

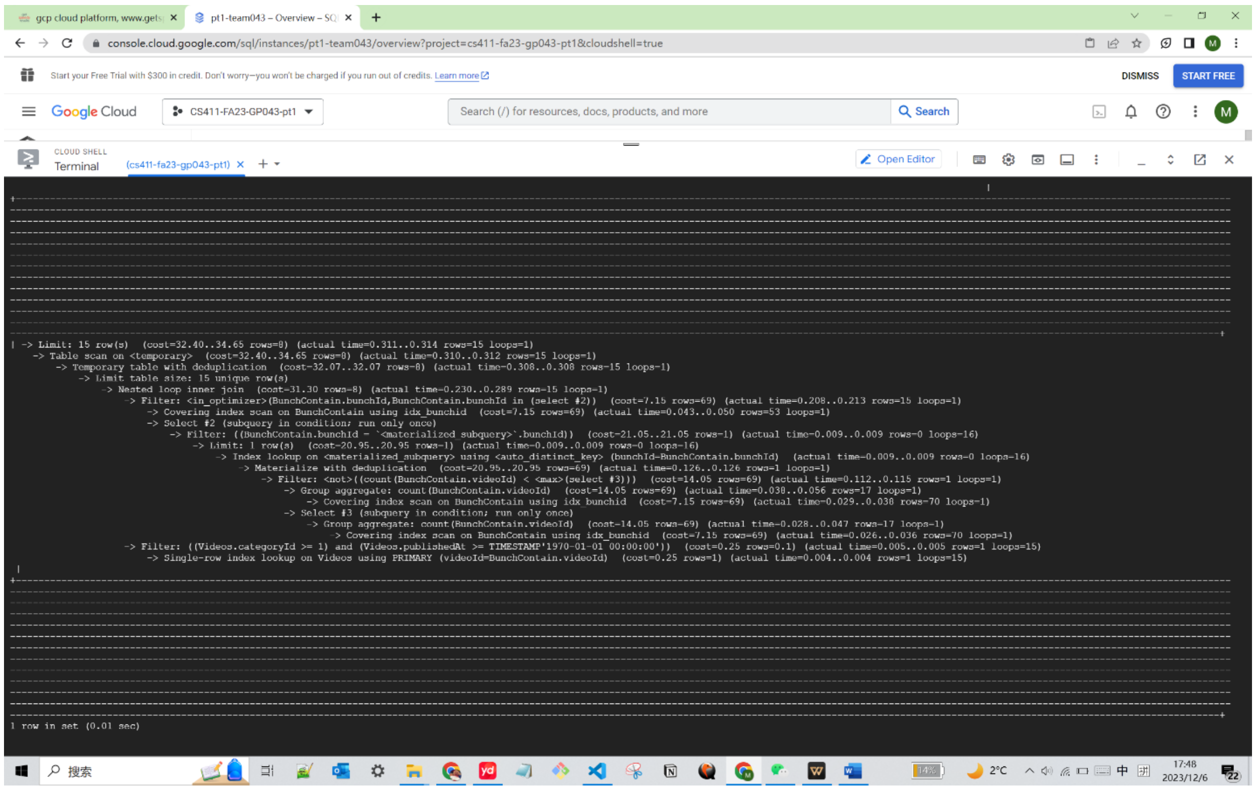
```sql
EXPLAIN ANALYZE
SELECT DISTINCT title
FROM Videos
JOIN BunchContain ON Videos.videoId = BunchContain.videoId
WHERE bunchId IN (
    SELECT bunchId
    FROM BunchContain
    GROUP BY bunchId
    HAVING COUNT(videoId) >= ALL(
        SELECT COUNT(videoId)
        FROM BunchContain
        GROUP BY bunchId
    )
)
AND categoryId >= 1
AND publishedAt >= '1970-01-01'
LIMIT 15;
CREATE INDEX idx_categoryId ON Videos (categoryId);
CREATE INDEX idx_publishedAt ON Videos (publishedAt);
CREATE INDEX idx_category_publishedAt ON Videos (categoryId, publishedAt);
```

Original:



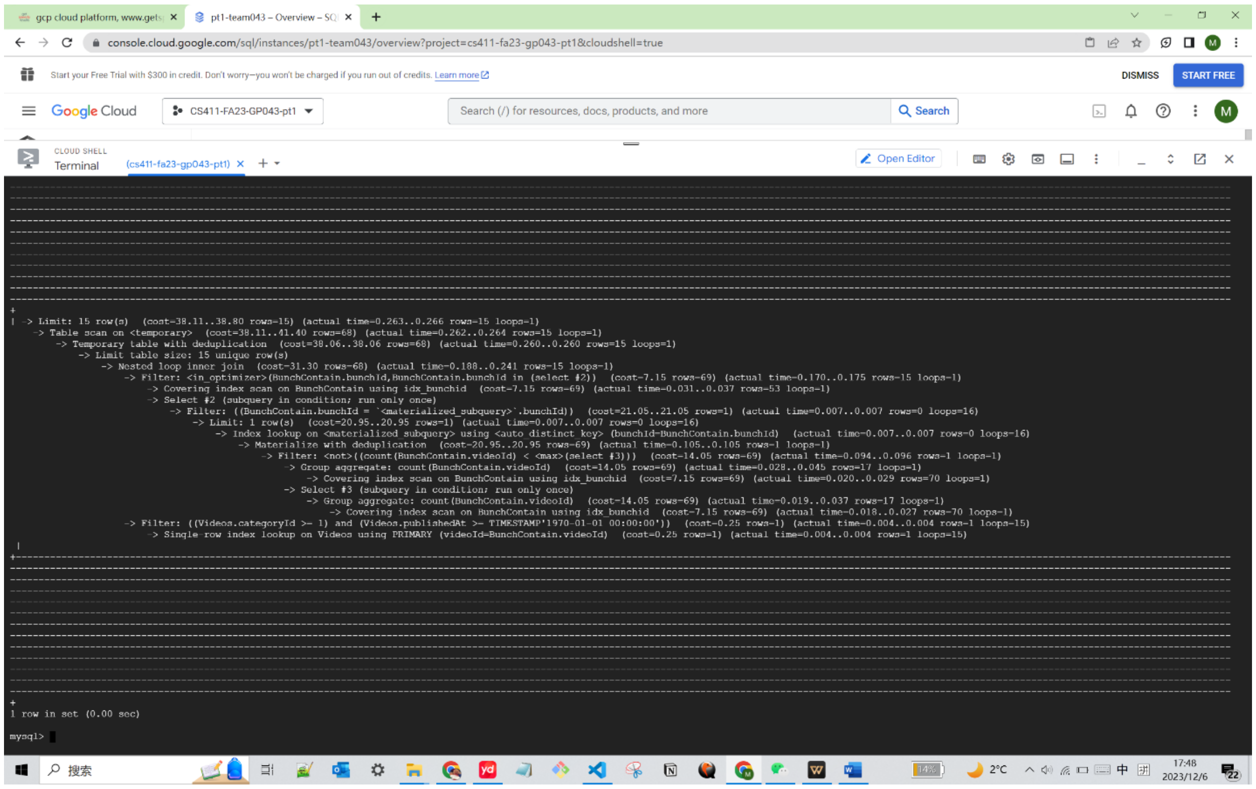
```
+------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=32.40..34.65 rows=8) (actual time=0.311..0.314 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=32.40..34.65 rows=8) (actual time=0.310..0.312 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=32.07..32.07 rows=8) (actual time=0.308..0.308 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=31.30 rows=8) (actual time=0.230..0.289 rows=15 loops=1)
                    -> Filter: <in_optimizer>(BunchContain.bunchId,BunchContain.bunchId in (select #2))  (cost=7.15 rows=69) (actual time=0.208..0.213 rows=15 loops=1)
                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.043..0.050 rows=53 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Filter: ((BunchContain.bunchId = `<materialized_subquery>`.bunchId))  (cost=21.05..21.05 rows=1) (actual time=0.009..0.009 rows=0 loops=16)
                                -> Limit: 1 row(s)  (cost=20.95..20.95 rows=1) (actual time=0.009..0.009 rows=0 loops=16)
                                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (bunchId=BunchContain.bunchId)  (actual time=0.009..0.009 rows=0 loops=16)
                                        -> Materialize with deduplication  (cost=20.95..20.95 rows=69) (actual time=0.126..0.126 rows=1 loops=1)
                                            -> Filter: <not>((count(BunchContain.videoId) < <max>(select #3)))  (cost=14.05 rows=69) (actual time=0.112..0.115 rows=1 loops=1)
                                                -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.038..0.056 rows=17 loops=1)
                                                    -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.029..0.038 rows=70 loops=1)
                                                -> Select #3 (subquery in condition; run only once)
                                                    -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.028..0.047 rows=17 loops=1)
                                                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.026..0.036 rows=70 loops=1)
                    -> Filter: ((Videos.categoryId >= 1) and (Videos.publishedAt >= TIMESTAMP'1970-01-01 00:00:00'))  (cost=0.25 rows=0.1) (actual time=0.005..0.005 rows=1 loops=15)
                        -> Single-row index lookup on Videos using PRIMARY (videoId=BunchContain.videoId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
 |
+------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)

mysql>
```

CREATE INDEX idx_categoryId on Videos(categoryId);



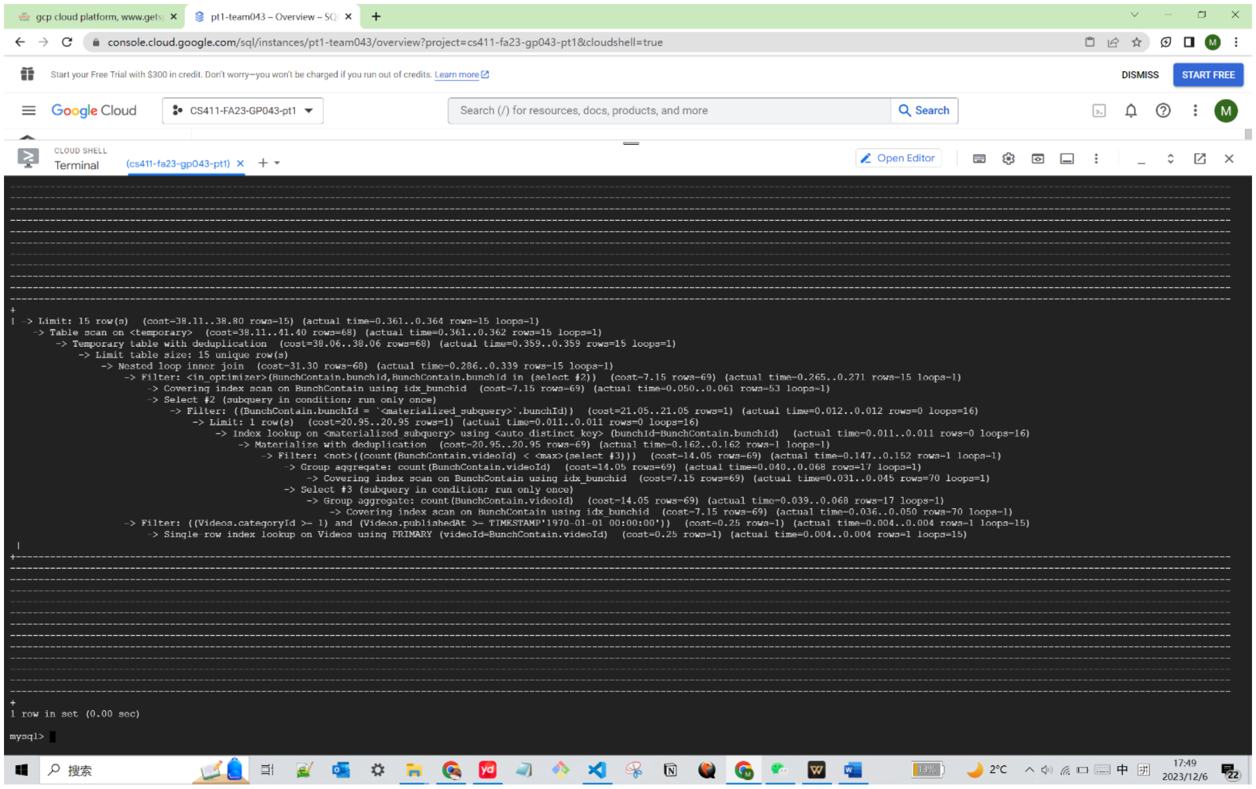
```
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=32.40..34.65 rows=8) (actual time=0.311..0.314 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=32.40..34.65 rows=8) (actual time=0.310..0.312 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=32.07..32.07 rows=8) (actual time=0.308..0.308 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=31.30 rows=8) (actual time=0.230..0.289 rows=15 loops=1)
                    -> Filter: <in_optimizer>(BunchContain.bunchId,BunchContain.bunchId in (select #2))  (cost=7.15 rows=69) (actual time=0.208..0.213 rows=15 loops=1)
                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.043..0.050 rows=53 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Filter: ((BunchContain.bunchId = `<materialized_subquery>`.bunchId))  (cost=21.05..21.05 rows=1) (actual time=0.009..0.009 rows=0 loops=16)
                                -> Limit: 1 row(s)  (cost=20.95..20.95 rows=1) (actual time=0.009..0.009 rows=0 loops=16)
                                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (bunchId=BunchContain.bunchId)  (actual time=0.009..0.009 rows=0 loops=16)
                                        -> Materialize with deduplication  (cost=20.95..20.95 rows=69) (actual time=0.126..0.126 rows=1 loops=1)
                                            -> Filter: <not>((count(BunchContain.videoId) < <max>(select #3)))  (cost=14.05 rows=69) (actual time=0.112..0.115 rows=1 loops=1)
                                                -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.038..0.056 rows=17 loops=1)
                                                    -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.029..0.038 rows=70 loops=1)
                                                -> Select #3 (subquery in condition; run only once)
                                                    -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.028..0.047 rows=17 loops=1)
                                                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.026..0.036 rows=70 loops=1)
                    -> Filter: ((Videos.categoryId >= 1) and (Videos.publishedAt >= TIMESTAMP'1970-01-01 00:00:00'))  (cost=0.25 rows=0.1) (actual time=0.005..0.005 rows=1 loops=15)
                        -> Single-row index lookup on Videos using PRIMARY (videoId=BunchContain.videoId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

CREATE INDEX idx_publishedAt on Videos(publishedAt);



```
+
| -> Limit: 15 row(s)  (cost=38.11..38.80 rows=15) (actual time=0.263..0.266 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=38.11..41.40 rows=68) (actual time=0.262..0.264 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=38.06..38.06 rows=68) (actual time=0.260..0.260 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=31.30 rows=68) (actual time=0.188..0.241 rows=15 loops=1)
                    -> Filter: <in_optimizer>(BunchContain.bunchId,BunchContain.bunchId in (select #2))  (cost=7.15 rows=69) (actual time=0.170..0.175 rows=15 loops=1)
                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.031..0.037 rows=53 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Filter: ((BunchContain.bunchId = `<materialized_subquery>`.bunchId))  (cost=21.05..21.05 rows=1) (actual time=0.007..0.007 rows=0 loops=16)
                                -> Limit: 1 row(s)  (cost=20.95..20.95 rows=1) (actual time=0.007..0.007 rows=0 loops=16)
                                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (bunchId=BunchContain.bunchId)  (actual time=0.007..0.007 rows=0 loops=16)
                                        -> Materialize with deduplication  (cost=20.95..20.95 rows=69) (actual time=0.105..0.105 rows=1 loops=1)
                                            -> Filter: <not>((count(BunchContain.videoId) < <max>(select #3)))  (cost=14.05 rows=69) (actual time=0.094..0.096 rows=1 loops=1)
                                                -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.028..0.045 rows=17 loops=1)
                                                    -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.020..0.029 rows=70 loops=1)
                                                -> Select #3 (subquery in condition; run only once)
                                                    -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.019..0.037 rows=17 loops=1)
                                                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.018..0.027 rows=70 loops=1)
                    -> Filter: ((Videos.categoryId >= 1) and (Videos.publishedAt >= TIMESTAMP'1970-01-01 00:00:00'))  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
                        -> Single-row index lookup on Videos using PRIMARY (videoId=BunchContain.videoId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)

mysql>
```

CREATE INDEX idx_category_publishedAt on Videos(categoryId, publishedAt);



```
+
| -> Limit: 15 row(s)  (cost=38.11..38.80 rows=15) (actual time=0.361..0.364 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=38.11..41.40 rows=68) (actual time=0.361..0.362 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=38.06..38.06 rows=68) (actual time=0.359..0.359 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=31.30 rows=68) (actual time=0.286..0.339 rows=15 loops=1)
                    -> Filter: <in_optimizer>(BunchContain.bunchId,BunchContain.bunchId in (select #2))  (cost=7.15 rows=69) (actual time=0.265..0.271 rows=15 loops=1)
                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.050..0.061 rows=53 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Filter: ((BunchContain.bunchId = `<materialized_subquery>`.bunchId))  (cost=21.05..21.05 rows=1) (actual time=0.012..0.012 rows=0 loops=16)
                                -> Limit: 1 row(s)  (cost=20.95..20.95 rows=1) (actual time=0.011..0.011 rows=0 loops=16)
                                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (bunchId=BunchContain.bunchId)  (actual time=0.011..0.011 rows=0 loops=16)
                                        -> Materialize with deduplication  (cost=20.95..20.95 rows=69) (actual time=0.162..0.162 rows=1 loops=1)
                                            -> Filter: <not>((count(BunchContain.videoId) < <max>(select #3)))  (cost=14.05 rows=69) (actual time=0.147..0.152 rows=1 loops=1)
                                                -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.040..0.068 rows=17 loops=1)
                                                    -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.031..0.045 rows=70 loops=1)
                                                -> Select #3 (subquery in condition; run only once)
                                                    -> Group aggregate: count(BunchContain.videoId)  (cost=14.05 rows=69) (actual time=0.039..0.068 rows=17 loops=1)
                                                        -> Covering index scan on BunchContain using idx_bunchid  (cost=7.15 rows=69) (actual time=0.036..0.050 rows=70 loops=1)
                    -> Filter: ((Videos.categoryId >= 1) and (Videos.publishedAt >= TIMESTAMP'1970-01-01 00:00:00'))  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
                        -> Single-row index lookup on Videos using PRIMARY (videoId=BunchContain.videoId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)

mysql>
```

Analyses:

idx_categoryId on Videos(categoryId):

This index is beneficial for the categoryId >= 1 condition. It allows the database engine to efficiently retrieve rows where the categoryId is greater than or equal to 1. There are a large number of distinct categoryId values and queries often filter based on this column, the index is appropriate

idx_publishedAt on Videos(publishedAt):

This index supports the publishedAt >= '1970-01-01' condition. It allows for efficient retrieval of videos published after a specific date. Since queries frequently filter videos based on the publication date, this index is beneficial.

idx_category_publishedAt on Videos(categoryId, publishedAt):

This composite index covers both the categoryId and publishedAt conditions used in the WHERE clause. It is helpful when filtering based on both categoryId and publishedAt, potentially improving the performance of queries with combined conditions.

Conclusion:

Since our biggest bunch contains data that is not enough(only hundreds), the time it executed had little differences. idx_category_publishedAt on Videos(categoryId, publishedAt) should be good when filtering based on both categoryId and publishedAt.