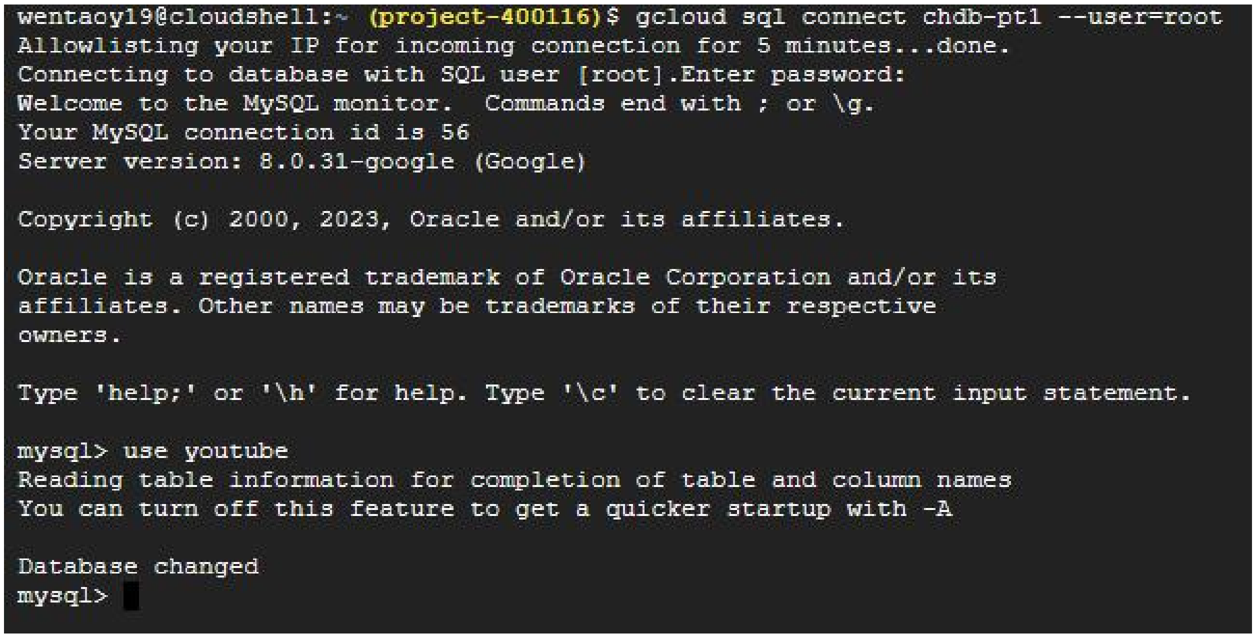
DDL Commands

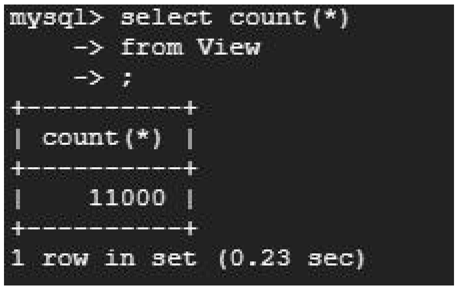
```
use youtube;
CREATE TABLE Nation(
     Nationid INT PRIMARY KEY,
    NationName VARCHAR(50),
    Nation_Mostlike_VideoId VARCHAR(50),
    Play_Volume INT
);
CREATE TABLE Time(
              Trending_Date VARCHAR(50) PRIMARY KEY,
              Time_Mostviewed_VideoId VARCHAR(50),
              Time_Mostlike_VideoId VARCHAR(50),
              Time_Mostdislike_VideoId VARCHAR(50)
);

CREATE TABLE Channels (
   Channel_Id VARCHAR(30) PRIMARY KEY,
   Channel_Title VARCHAR(255),
   Channel_MostLike_VideoId VARCHAR(255),
   Channel_MostDislike_VideoId VARCHAR(255)
);
CREATE TABLE Category (
   Category_Id INT PRIMARY KEY,
   Video_Count INT,
   Category_Name VARCHAR(255),
   Category_MostLike_VideoId VARCHAR(255)
);
CREATE TABLE Video(
    Video_id VARCHAR(255) PRIMARY KEY,
   Title VARCHAR(255),
   Channel_Id VARCHAR(30),
   Category_Id INT,
   Tag_Name VARCHAR(255),
   Publish_time DATE,
   FOREIGN KEY (Channel_Id) REFERENCES Channels(Channel_Id),
   FOREIGN KEY (Category_Id) REFERENCES Category(Category_Id)
);
CREATE TABLE View(
            View_id INT PRIMARY KEY AUTO_INCREMENT,
            Video_id VARCHAR(50),
            Nationid INT,
            Likes INT,
            Dislikes INT,
            Views INT,
               trending_date VARCHAR(50),
               FOREIGN KEY (Video_id) REFERENCES Video(Video_id),
               FOREIGN KEY (Nationid) REFERENCES Nation(Nationid),
               FOREIGN KEY (Trending_Date) REFERENCES Time(Trending_Date)
)
```

Database Connection

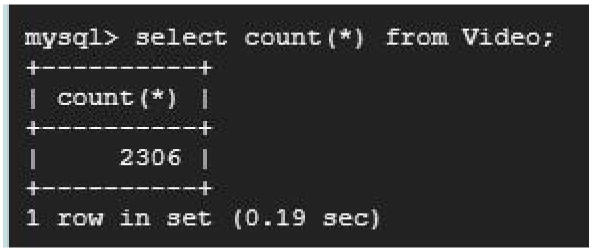


```
wentaoy19@cloudshell:~ (project-400116)$ gcloud sql connect chdb-pt1 --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 56
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2023, 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 youtube
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>
```

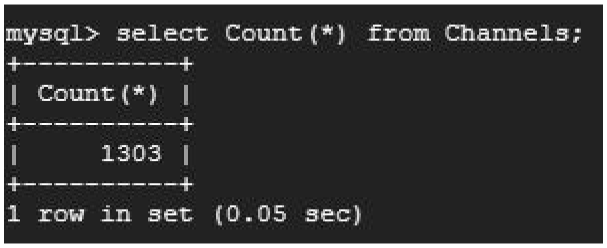
Data Count



```
mysql> select count(*)
    -> from View
    -> ;
+----------+
| count(*) |
+----------+
|    11000 |
+----------+
1 row in set (0.23 sec)
```



```
mysql> select count(*) from Video;
+----------+
| count(*) |
+----------+
|     2306 |
+----------+
1 row in set (0.19 sec)
```



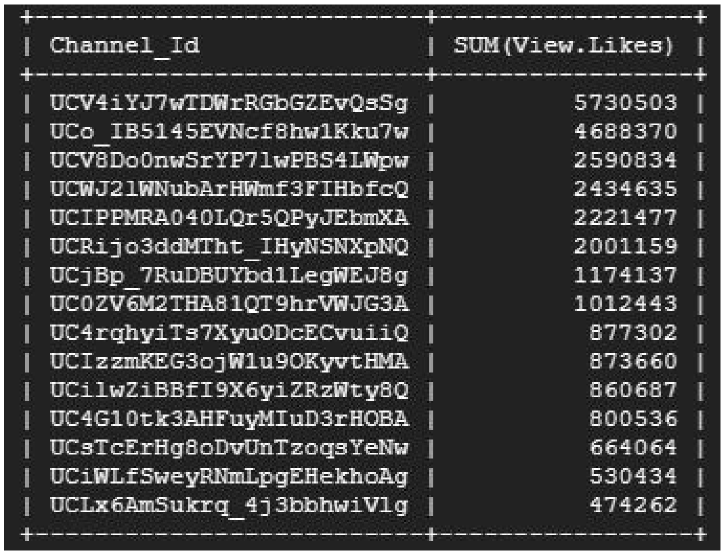
```
mysql> select Count(*) from Channels;
+----------+
| Count(*) |
+----------+
|     1303 |
+----------+
1 row in set (0.05 sec)
```

Advance Query & Index Analysis

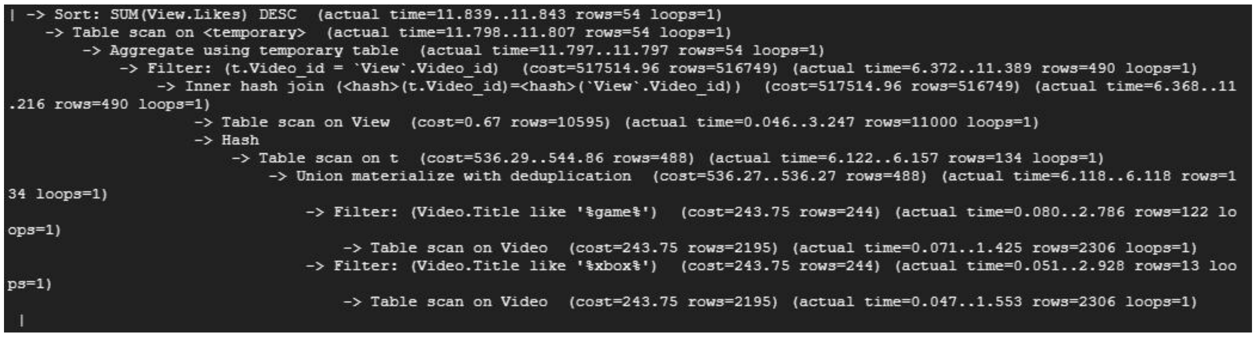
Query1:

```
SELECT t.Channel_Id, SUM(View.Likes)

FROM

(

  (

    SELECT *

    FROM Video

    WHERE Title LIKE '%game%'

  )

  UNION

  (

    SELECT *

    FROM Video

    WHERE Title LIKE '%xbox%'

  )

) as t

INNER JOIN View on t.Video_id=View.Video_id

GROUP BY t.Channel_Id

ORDER BY SUM(View.Likes)
```

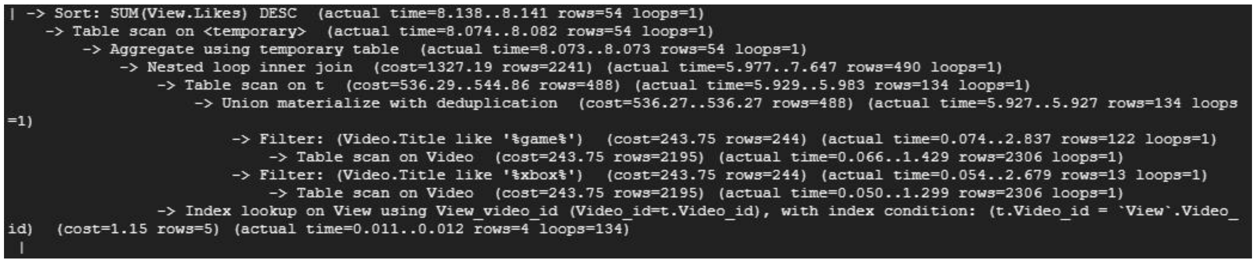
Top 15 Output:


```
+------------------------+-----------------+
| Channel_Id             | SUM(View.Likes) |
+------------------------+-----------------+
| UCV4iYJ7wTDWrRGbGZEvQsSg |         5730503 |
| UCo_IB5145EVNcf8hw1Kku7w |         4688370 |
| UCV8Do0nwSrYP7lwPBS4LWpw |         2590834 |
| UCWJ2lWNubArHWmf3FIHbfcQ |         2434635 |
| UCIPPMRA040LQr5QPyJEbmXA |         2221477 |
| UCRijo3ddMTht_IHyNSNXpNQ |         2001159 |
| UCjBp_7RuDBUYbd1LegWEJ8g |         1174137 |
| UC0ZV6M2THA81QT9hrVWJG3A |         1012443 |
| UC4rqhyiTs7XyuODcECvuiiQ |          877302 |
| UCIzzmKEG3ojW1u9OKyvtHMA |          873660 |
| UCilwZiBBfI9X6yiZRzWty8Q |          860687 |
| UC4G10tk3AHFuyMIuD3rHOBA |          800536 |
| UCsTcErHg8oDvUnTzoqsYeNw |          664064 |
| UCiWLfSweyRNmLpgEHekhoAg |          530434 |
| UCLx6AmSukrq_4j3bbhwiVlg |          474262 |
+------------------------+-----------------+
```

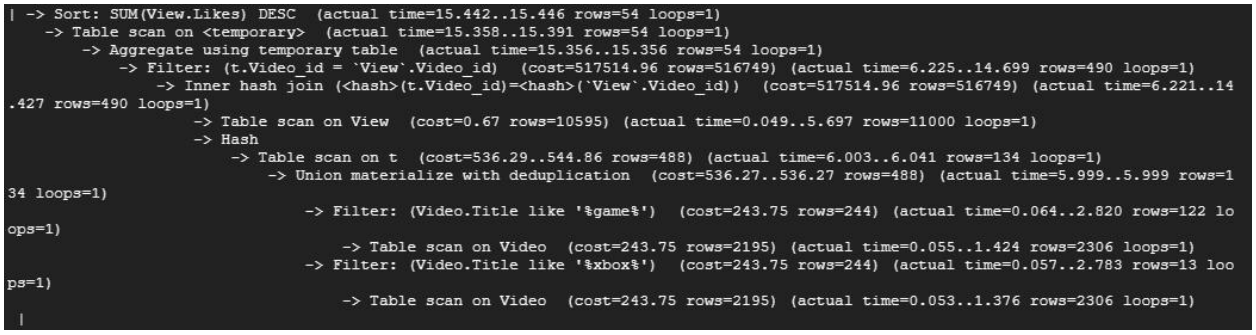
Default Indexing:


```
| -> Sort: SUM(View.Likes) DESC  (actual time=11.839..11.843 rows=54 loops=1)
    -> Table scan on <temporary>  (actual time=11.798..11.807 rows=54 loops=1)
        -> Aggregate using temporary table  (actual time=11.797..11.797 rows=54 loops=1)
            -> Filter: (t.Video_id = `View`.Video_id)  (cost=517514.96 rows=516749) (actual time=6.372..11.389 rows=490 loops=1)
                -> Inner hash join (<hash>(t.Video_id)=<hash>(`View`.Video_id))  (cost=517514.96 rows=516749) (actual time=6.368..11
.216 rows=490 loops=1)
                    -> Table scan on View  (cost=0.67 rows=10595) (actual time=0.046..3.247 rows=11000 loops=1)
                    -> Hash
                        -> Table scan on t  (cost=536.29..544.86 rows=488) (actual time=6.122..6.157 rows=134 loops=1)
                            -> Union materialize with deduplication  (cost=536.27..536.27 rows=488) (actual time=6.118..6.118 rows=1
34 loops=1)
                                -> Filter: (Video.Title like '%game%')  (cost=243.75 rows=244) (actual time=0.080..2.786 rows=122 lo
ops=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.071..1.425 rows=2306 loops=1)
                                -> Filter: (Video.Title like '%xbox%')  (cost=243.75 rows=244) (actual time=0.051..2.928 rows=13 loo
ps=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.047..1.553 rows=2306 loops=1)
 |
```

Indexing View(Video_id):

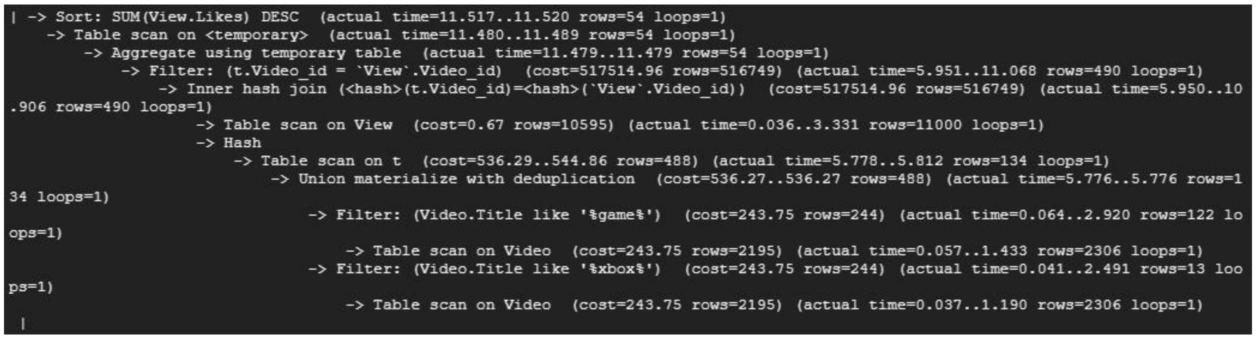

```
| -> Sort: SUM(View.Likes) DESC  (actual time=8.138..8.141 rows=54 loops=1)
    -> Table scan on <temporary>  (actual time=8.074..8.082 rows=54 loops=1)
        -> Aggregate using temporary table  (actual time=8.073..8.073 rows=54 loops=1)
            -> Nested loop inner join  (cost=1327.19 rows=2241) (actual time=5.977..7.647 rows=490 loops=1)
                -> Table scan on t  (cost=536.29..544.86 rows=488) (actual time=5.929..5.983 rows=134 loops=1)
                    -> Union materialize with deduplication  (cost=536.27..536.27 rows=488) (actual time=5.927..5.927 rows=134 loops
=1)
                        -> Filter: (Video.Title like '%game%')  (cost=243.75 rows=244) (actual time=0.074..2.837 rows=122 loops=1)
                            -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.066..1.429 rows=2306 loops=1)
                        -> Filter: (Video.Title like '%xbox%')  (cost=243.75 rows=244) (actual time=0.054..2.679 rows=13 loops=1)
                            -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.050..1.299 rows=2306 loops=1)
                -> Index lookup on View using View_video_id (Video_id=t.Video_id), with index condition: (t.Video_id = `View`.Video_
id)  (cost=1.15 rows=5) (actual time=0.011..0.012 rows=4 loops=134)
 |
```

Indexing Video(Title):



```
| -> Sort: SUM(View.Likes) DESC  (actual time=15.442..15.446 rows=54 loops=1)
    -> Table scan on <temporary>  (actual time=15.358..15.391 rows=54 loops=1)
        -> Aggregate using temporary table  (actual time=15.356..15.356 rows=54 loops=1)
            -> Filter: (t.Video_id = `View`.Video_id)  (cost=517514.96 rows=516749) (actual time=6.225..14.699 rows=490 loops=1)
                -> Inner hash join (<hash>(t.Video_id)=<hash>(`View`.Video_id))  (cost=517514.96 rows=516749) (actual time=6.221..14
.427 rows=490 loops=1)
                    -> Table scan on View  (cost=0.67 rows=10595) (actual time=0.049..5.697 rows=11000 loops=1)
                    -> Hash
                        -> Table scan on t  (cost=536.29..544.86 rows=488) (actual time=6.003..6.041 rows=134 loops=1)
                            -> Union materialize with deduplication  (cost=536.27..536.27 rows=488) (actual time=5.999..5.999 rows=1
34 loops=1)
                                -> Filter: (Video.Title like '%game%')  (cost=243.75 rows=244) (actual time=0.064..2.820 rows=122 lo
ops=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.055..1.424 rows=2306 loops=1)
                                -> Filter: (Video.Title like '%xbox%')  (cost=243.75 rows=244) (actual time=0.057..2.783 rows=13 loo
ps=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.053..1.376 rows=2306 loops=1)
 |
```

Indexing Video(Channel_Id):



```
| -> Sort: SUM(View.Likes) DESC  (actual time=11.517..11.520 rows=54 loops=1)
    -> Table scan on <temporary>  (actual time=11.480..11.489 rows=54 loops=1)
        -> Aggregate using temporary table  (actual time=11.479..11.479 rows=54 loops=1)
            -> Filter: (t.Video_id = `View`.Video_id)  (cost=517514.96 rows=516749) (actual time=5.951..11.068 rows=490 loops=1)
                -> Inner hash join (<hash>(t.Video_id)=<hash>(`View`.Video_id))  (cost=517514.96 rows=516749) (actual time=5.950..10
.906 rows=490 loops=1)
                    -> Table scan on View  (cost=0.67 rows=10595) (actual time=0.036..3.331 rows=11000 loops=1)
                    -> Hash
                        -> Table scan on t  (cost=536.29..544.86 rows=488) (actual time=5.778..5.812 rows=134 loops=1)
                            -> Union materialize with deduplication  (cost=536.27..536.27 rows=488) (actual time=5.776..5.776 rows=1
34 loops=1)
                                -> Filter: (Video.Title like '%game%')  (cost=243.75 rows=244) (actual time=0.064..2.920 rows=122 lo
ops=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.057..1.433 rows=2306 loops=1)
                                -> Filter: (Video.Title like '%xbox%')  (cost=243.75 rows=244) (actual time=0.041..2.491 rows=13 loo
ps=1)
                                    -> Table scan on Video  (cost=243.75 rows=2195) (actual time=0.037..1.190 rows=2306 loops=1)
 |
```

The reason why we choose these three keys is because they exist in the step of JOIN, GROUP in WHERE conditions.

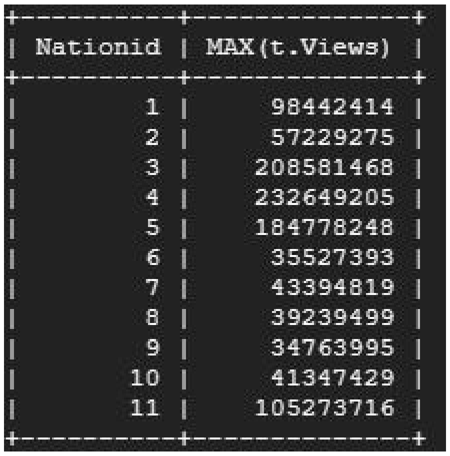
Based on the analysis result, both Video.Title and Video.Channel_Id indexing will not improve the performance, while indexing on View.Video_id will improve the query execution performance. For indexing on Video.Title, since the condition is on whether a string exists in that attribute, it is hard to make it construct a BTREE index to accelerate the search. And for Video.Channel_Id, this will not influence the performance since it is just used in displaying the final result by table scan, not used in a condition. And for View.Video_id, this will improve the performance of execution because this attribute is used for join Video table, so indexing on that will make the joining step much faster.

Query 2:

```
SELECT t.Nationid, MAX(t.Views)
FROM (
   SELECT *
   FROM View
   WHERE Likes>400000
) as t
```

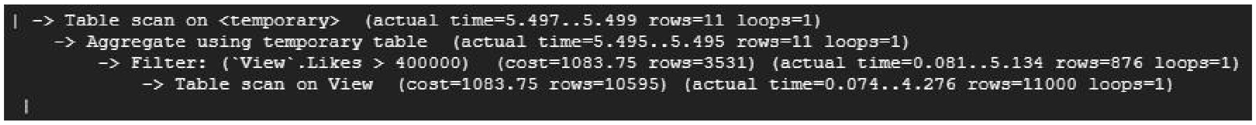
GROUP BY Nationid;

Output(less than 15):



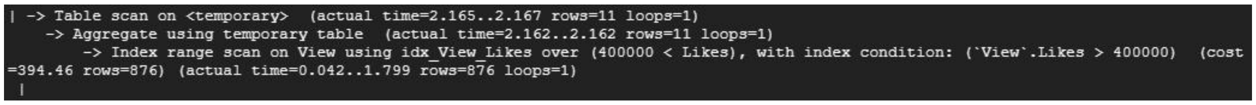
```
+----------+--------------+
| Nationid | MAX(t.Views) |
+----------+--------------+
|        1 |     98442414 |
|        2 |     57229275 |
|        3 |    208581468 |
|        4 |    232649205 |
|        5 |    184778248 |
|        6 |     35527393 |
|        7 |     43394819 |
|        8 |     39239499 |
|        9 |     34763995 |
|       10 |     41347429 |
|       11 |    105273716 |
+----------+--------------+
```

Baseline:



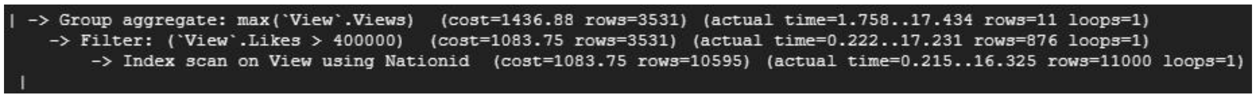
```
| -> Table scan on <temporary>  (actual time=5.497..5.499 rows=11 loops=1)
    -> Aggregate using temporary table  (actual time=5.495..5.495 rows=11 loops=1)
        -> Filter: (`View`.Likes > 400000)  (cost=1083.75 rows=3531) (actual time=0.081..5.134 rows=876 loops=1)
            -> Table scan on View  (cost=1083.75 rows=10595) (actual time=0.074..4.276 rows=11000 loops=1)
 |
```

Having index View(Likes):



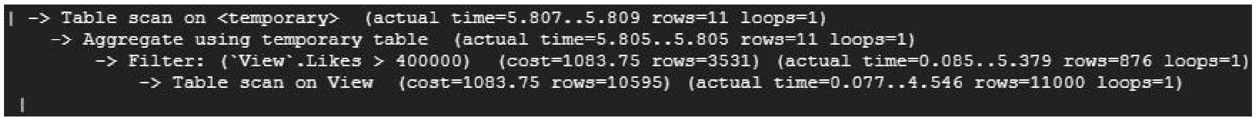
```
| -> Table scan on <temporary>  (actual time=2.165..2.167 rows=11 loops=1)
    -> Aggregate using temporary table  (actual time=2.162..2.162 rows=11 loops=1)
        -> Index range scan on View using idx_View_Likes over (400000 < Likes), with index condition: (`View`.Likes > 400000)  (cost
=394.46 rows=876) (actual time=0.042..1.799 rows=876 loops=1)
 |
```

Having index View(Nationid):



```
| -> Group aggregate: max(`View`.Views)  (cost=1436.88 rows=3531) (actual time=1.758..17.434 rows=11 loops=1)
    -> Filter: (`View`.Likes > 400000)  (cost=1083.75 rows=3531) (actual time=0.222..17.231 rows=876 loops=1)
        -> Index scan on View using Nationid  (cost=1083.75 rows=10595) (actual time=0.215..16.325 rows=11000 loops=1)
 |
```

Having index View(Views):



```
| -> Table scan on <temporary>  (actual time=5.807..5.809 rows=11 loops=1)
    -> Aggregate using temporary table  (actual time=5.805..5.805 rows=11 loops=1)
        -> Filter: (`View`.Likes > 400000)  (cost=1083.75 rows=3531) (actual time=0.085..5.379 rows=876 loops=1)
            -> Table scan on View  (cost=1083.75 rows=10595) (actual time=0.077..4.546 rows=11000 loops=1)
 |
```

The reason why we choose these three keys is that they are used for WHERE condition, GROUP and MAX operation.

Based on the results, using View.Likes and View.Nationid as index can improve the query execution performance, while View.Views will not improve the performance. That is because View.Views are not for a search key condition, while both View.Likes and View.Nationid are in the condition of search and grouping, which will improve the performance in indexing.

We have modified Stage 2 (Add one table for many-to-many relation, The reason why we choose BCNF, And the description of entities), Please regrade it, thanks !