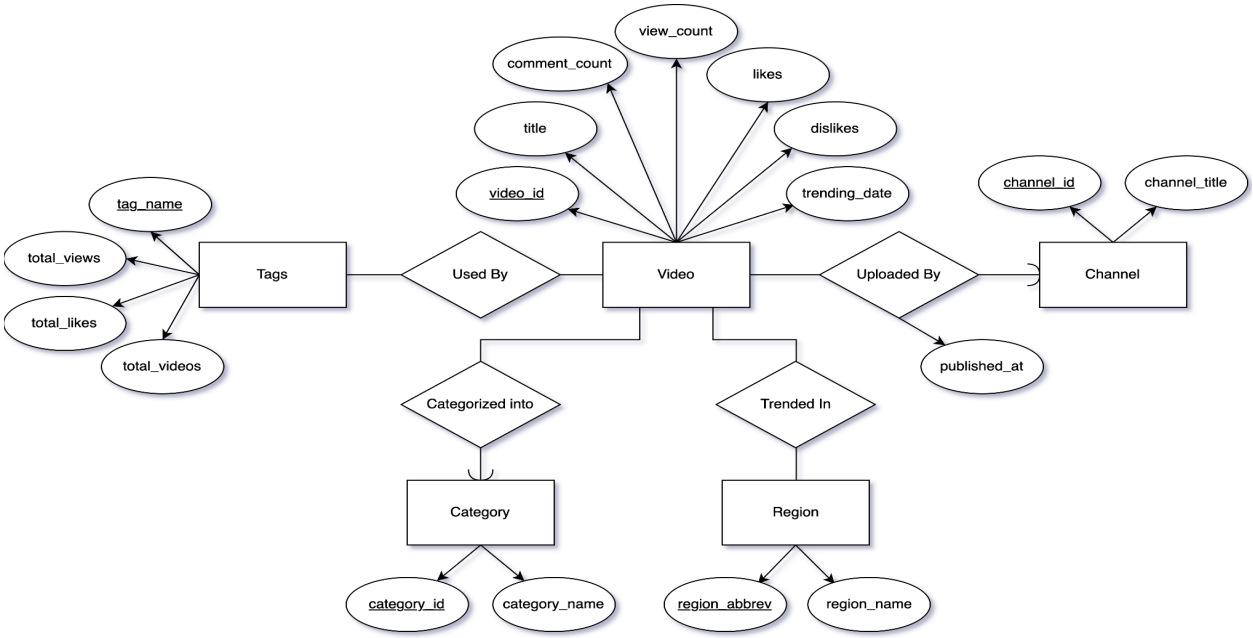
Database Implementation and Indexing



Screenshot of the GCP Connection

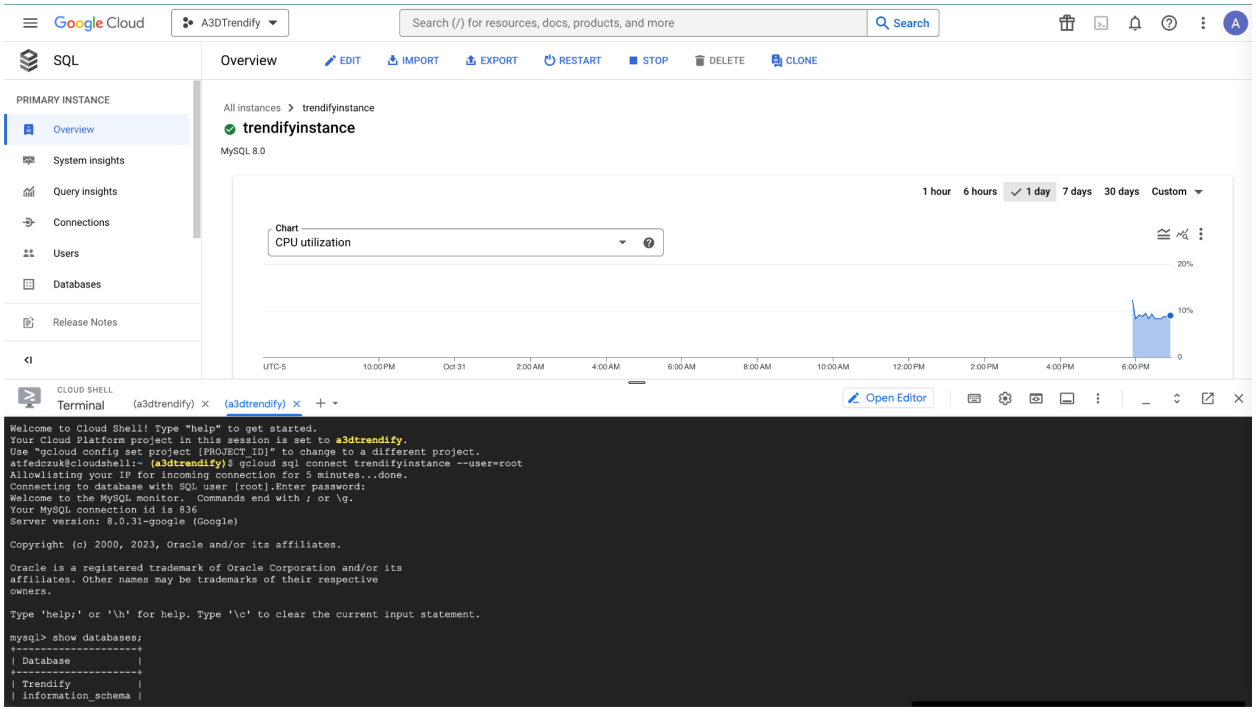
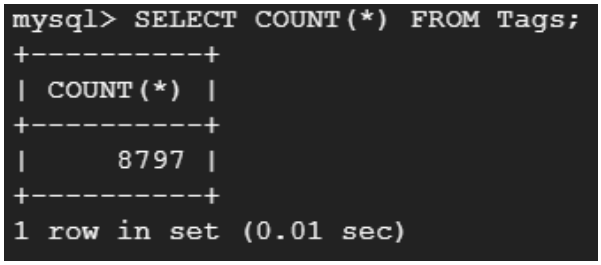


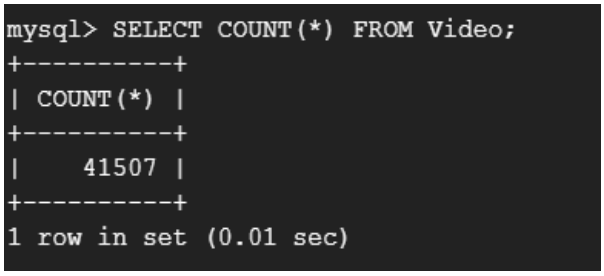
Table Commands (Over 1000 Rows) – Screenshots in the section below

```
CREATE TABLE Tags (
  tag_name VARCHAR(255) Primary Key,
  total_views INT,
  total_likes INT,
  total_videos INT
);

CREATE TABLE Video (
  video_id VARCHAR(255) Primary Key,
  title VARCHAR(255),
  trending_date DATETIME,
  channel_id VARCHAR(255) references Channel(channel_id),
  comment_count INT,
  view_count INT,
  likes INT,
  dislikes INT,
  published_at DATETIME,
  category_id INT references Category(category_id),
  tags VARCHAR(255)
);

CREATE TABLE Channel (
  channel_id VARCHAR(255) Primary Key,
  channel_title VARCHAR(255)
);

CREATE TABLE UsedBy (
  tag_name VARCHAR(255) references Tags(tag_name),
  video_id VARCHAR(255) references Video(video_id),
  PRIMARY KEY (tag_name, video_id)
);

CREATE TABLE TrendingIn (
  region_abbrev VARCHAR(255) references Region(region_abbrev),
  video_id VARCHAR(255) references Video(video_id),
  PRIMARY KEY (region_abbrev, video_id)
);
```

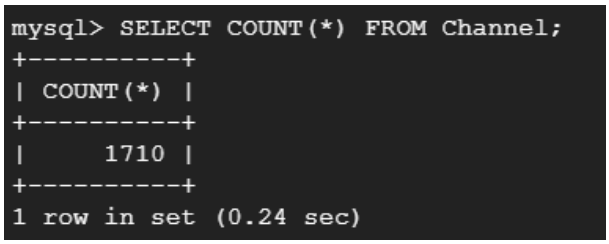
Screenshots of Row Numbers for tables with >1000 rows



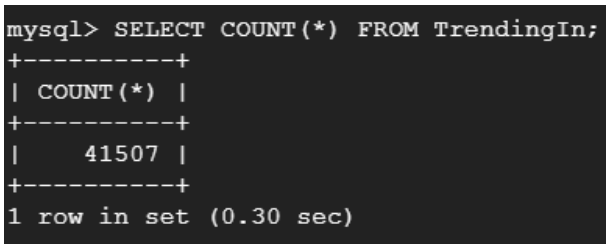
```
mysql> SELECT COUNT(*) FROM Tags;
+----------+
| COUNT(*) |
+----------+
|     8797 |
+----------+
1 row in set (0.01 sec)
```



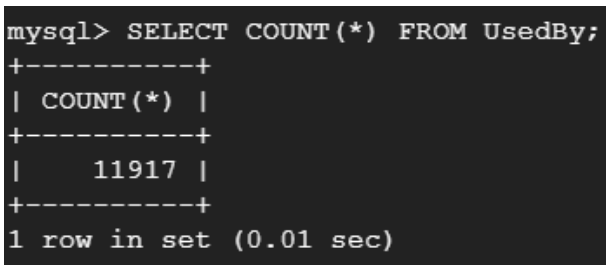
```
mysql> SELECT COUNT(*) FROM Video;
+----------+
| COUNT(*) |
+----------+
|    41507 |
+----------+
1 row in set (0.01 sec)
```



```
mysql> SELECT COUNT(*) FROM Channel;
+----------+
| COUNT(*) |
+----------+
|     1710 |
+----------+
1 row in set (0.24 sec)
```



```
mysql> SELECT COUNT(*) FROM TrendingIn;
+----------+
| COUNT(*) |
+----------+
|    41507 |
+----------+
1 row in set (0.30 sec)
```



```
mysql> SELECT COUNT(*) FROM UsedBy;
+----------+
| COUNT(*) |
+----------+
|    11917 |
+----------+
1 row in set (0.01 sec)
```

All Other Commands for Non-Main Tables (<1000 Rows)

```
CREATE TABLE Category (
  category_id    INT Primary Key,
  category_name    VARCHAR(255)
);

CREATE TABLE Region (
  region_abbrev    VARCHAR(255) Primary Key,
  region_name    VARCHAR(255)
);
```

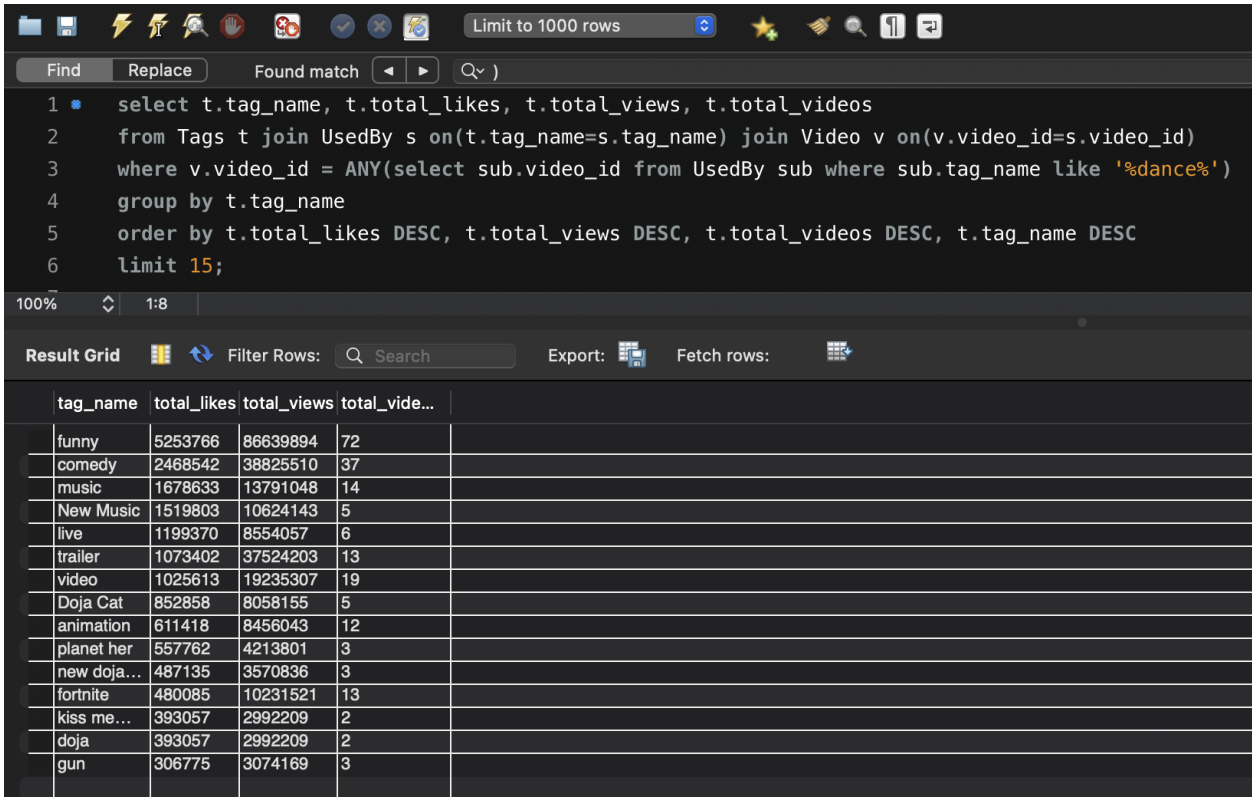
Advanced Queries

Query 1:

Using the imputed word(s), returns all the other tags that appeared on videos within the chosen category (including the tag keyword and tags with the keyword inside of it)
User Inputs:

- userkey (varchar) – for our purposes in this stage, it is “dance”

```
select t.tag_name, t.total_likes, t.total_views, t.total_videos
from Tags t join UsedBy s on(t.tag_name=s.tag_name) join Video v on(v.video_id=s.video_id)
where v.video_id = ANY(select sub.video_id from UsedBy sub where sub.tag_name like '%dance%')
group by t.tag_name
order by t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC
limit 15;
```



tag_name	total_likes	total_views	total_vide...
funny	5253766	86639894	72
comedy	2468542	38825510	37
music	1678633	13791048	14
New Music	1519803	10624143	5
live	1199370	8554057	6
trailer	1073402	37524203	13
video	1025613	19235307	19
Doja Cat	852858	8058155	5
animation	611418	8456043	12
planet her	557762	4213801	3
new doja…	487135	3570836	3
fortnite	480085	10231521	13
kiss me…	393057	2992209	2
doja	393057	2992209	2
gun	306775	3074169	3

Indexing Analysis for Query 1:

Query analysis after running the initial Explain Analyze (only default indexes present)

```
mysql> EXPLAIN ANALYZE select t.tag_name, t.total_likes, t.total_views, t.total_videos
    -> from Tags t join UsedBy s on(t.tag_name=s.tag_name) join Video v on(v.video_id=s.video_id)
    -> where v.video_id = ANY(select sub.video_id from UsedBy sub where sub.tag_name like '%dance%')
    -> group by t.tag_name
    -> order by t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC
    -> limit 15;
+-----------------------------------------------------------------------------------------------+
| EXPLAIN
       |
+-----------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=14.844..14.846 rows=15 loops=1)
    -> Sort: t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC, limit input to 15 row(s) per chunk  (actual time=14.843..14.844 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1749033.31..1767236.30 rows=1456040) (actual time=14.748..14.775 rows=159 loops=1)
            -> Temporary table with deduplication  (cost=1749033.30..1749033.30 rows=1456040) (actual time=14.745..14.745 rows=159 loops=1)
                -> Nested loop inner join  (cost=1603429.30 rows=1456040) (actual time=6.246..14.620 rows=173 loops=1)
                    -> Inner hash join (s.video_id = `<subquery2>`.video_id)  (cost=1457507.29 rows=1456040) (actual time=6.235..12.922 rows=173 loops=1)
                        -> Covering index scan on s using PRIMARY  (cost=138.94 rows=11448) (actual time=0.032..3.470 rows=11917 loops=1)
                        -> Hash
                            -> Nested loop inner join  (cost=1442.81 rows=1272) (actual time=6.120..6.162 rows=8 loops=1)
                                -> Table scan on <subquery2>  (cost=1296.25..1314.62 rows=1272) (actual time=6.094..6.095 rows=8 loops=1)
                                    -> Materialize with deduplication  (cost=1296.24..1296.24 rows=1272) (actual time=6.090..6.090 rows=8 loops=1)
                                        -> Filter: (sub.tag_name like '%dance%')  (cost=1169.05 rows=1272) (actual time=0.057..6.062 rows=12 loops=1)
                                            -> Covering index scan on sub using PRIMARY  (cost=1169.05 rows=11448) (actual time=0.048..3.733 rows=11917 loops=1)
                                -> Single-row covering index lookup on v using PRIMARY (video_id=`<subquery2>`.video_id)  (cost=1.10 rows=1) (actual time=0.008..0.008 rows=1 loops=8)
                    -> Single-row index lookup on t using PRIMARY (tag_name=s.tag_name)  (cost=0.28 rows=1) (actual time=0.010..0.010 rows=1 loops=173)
 |
```

Query analysis after indexing total_likes in the Tags table

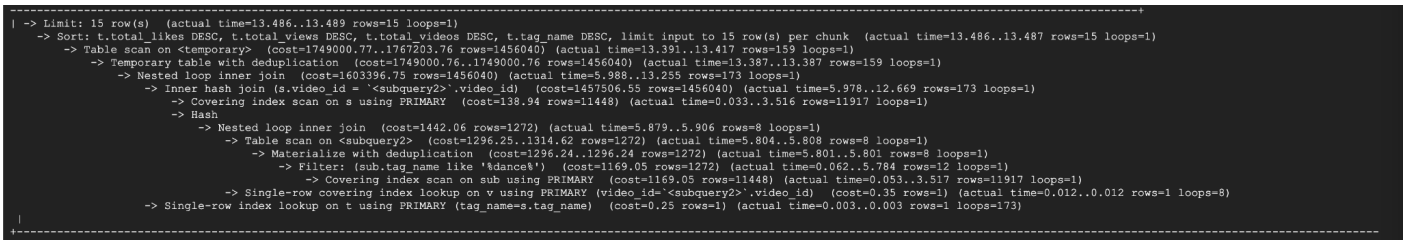
Index created using “CREATE INDEX totallikes on Tags(total_likes);”

Note: Explain Analyze command not included in following screenshots for Query 1 since it stays the same

```
-----------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=13.461..13.463 rows=15 loops=1)
    -> Sort: t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC, limit input to 15 row(s) per chunk  (actual time=13.460..13.461 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1749000.77..1767203.76 rows=1456040) (actual time=13.362..13.388 rows=159 loops=1)
            -> Temporary table with deduplication  (cost=1749000.76..1749000.76 rows=1456040) (actual time=13.358..13.358 rows=159 loops=1)
                -> Nested loop inner join  (cost=1603396.75 rows=1456040) (actual time=5.962..13.235 rows=173 loops=1)
                    -> Inner hash join (s.video_id = `<subquery2>`.video_id)  (cost=1457506.55 rows=1456040) (actual time=5.930..12.749 rows=173 loops=1)
                        -> Covering index scan on s using PRIMARY  (cost=138.94 rows=11448) (actual time=0.035..3.579 rows=11917 loops=1)
                        -> Hash
                            -> Nested loop inner join  (cost=1442.06 rows=1272) (actual time=5.804..5.842 rows=8 loops=1)
                                -> Table scan on <subquery2>  (cost=1296.25..1314.62 rows=1272) (actual time=5.776..5.778 rows=8 loops=1)
                                    -> Materialize with deduplication  (cost=1296.24..1296.24 rows=1272) (actual time=5.773..5.773 rows=8 loops=1)
                                        -> Filter: (sub.tag_name like '%dance%')  (cost=1169.05 rows=1272) (actual time=0.055..5.757 rows=12 loops=1)
                                            -> Covering index scan on sub using PRIMARY  (cost=1169.05 rows=11448) (actual time=0.046..3.397 rows=11917 loops=1)
                                -> Single-row covering index lookup on v using PRIMARY (video_id=`<subquery2>`.video_id)  (cost=0.35 rows=1) (actual time=0.008..0.008 rows=1 loops=8)
                    -> Single-row index lookup on t using PRIMARY (tag_name=s.tag_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=173)
 |
+-----------------------------------------------------------------------------------------------
```

Analysis Report: The actual time that the query takes is similar to the regular one with only the default indexes. However, looking at the results after indexing total likes in the Tags table, we see that the cost has decreased by about 33. We used total_likes because it is the first thing we sort by and the most heavily weighed. The very slight decrease in cost could be due to a few instances where a video has one tag, thus the number of total likes might have been unique to that tag (using this dataset and this specific situation). With this indexing, it would have been easier to locate those singular instances. However, the few values themselves just happened to be unique, so with different datasets of the same sizes, this indexing could lead to no improvement if tags like these do not exist.

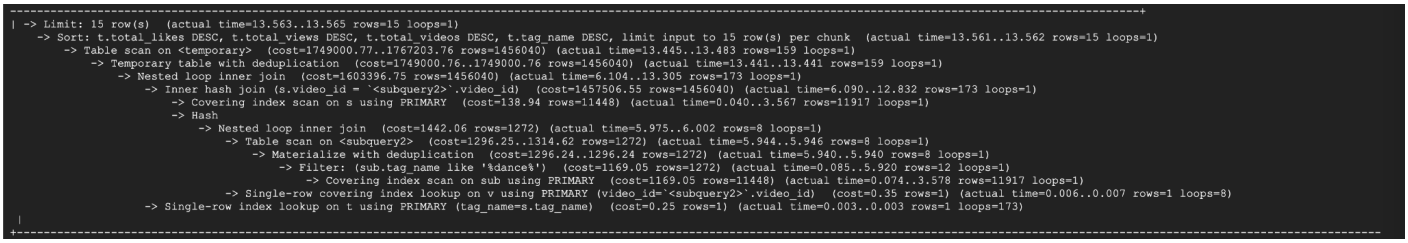
Query analysis after indexing total_views in the Tags table
Index created using “CREATE INDEX totalviews on Tags(total_views);”



```
-> Limit: 15 row(s)  (actual time=13.486..13.489 rows=15 loops=1)
    -> Sort: t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC, limit input to 15 row(s) per chunk  (actual time=13.486..13.487 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1749000.77..1767203.76 rows=1456040) (actual time=13.391..13.417 rows=159 loops=1)
            -> Temporary table with deduplication  (cost=1749000.76..1749000.76 rows=1456040) (actual time=13.387..13.387 rows=159 loops=1)
                -> Nested loop inner join  (cost=1603396.75 rows=1456040) (actual time=5.988..13.255 rows=173 loops=1)
                    -> Inner hash join (s.video_id = `<subquery2>`.video_id)  (cost=1457506.55 rows=1456040) (actual time=5.978..12.669 rows=173 loops=1)
                        -> Covering index scan on s using PRIMARY  (cost=138.94 rows=11448) (actual time=0.033..3.516 rows=11917 loops=1)
                        -> Hash
                            -> Nested loop inner join  (cost=1442.06 rows=1272) (actual time=5.879..5.906 rows=8 loops=1)
                                -> Table scan on <subquery2>  (cost=1296.25..1314.62 rows=1272) (actual time=5.804..5.808 rows=8 loops=1)
                                    -> Materialize with deduplication  (cost=1296.24..1296.24 rows=1272) (actual time=5.801..5.801 rows=8 loops=1)
                                        -> Filter: (sub.tag_name like '%dance%')  (cost=1169.05 rows=1272) (actual time=0.062..5.784 rows=12 loops=1)
                                            -> Covering index scan on sub using PRIMARY  (cost=1169.05 rows=11448) (actual time=0.053..3.517 rows=11917 loops=1)
                                -> Single-row covering index lookup on v using PRIMARY (video_id=`<subquery2>`.video_id)  (cost=0.35 rows=1) (actual time=0.012..0.012 rows=1 loops=8)
                    -> Single-row index lookup on t using PRIMARY (tag_name=s.tag_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=173)
```

Analysis Report: Total_likes was chosen to see if the second attribute we order by would make a difference. The costs are exactly the same as before and it seems to be for the same reason that we see in the previous indexing design.

Query analysis after indexing total_views in the Tags table
Index created using “CREATE INDEX totalvideos on Tags(total_videos);”



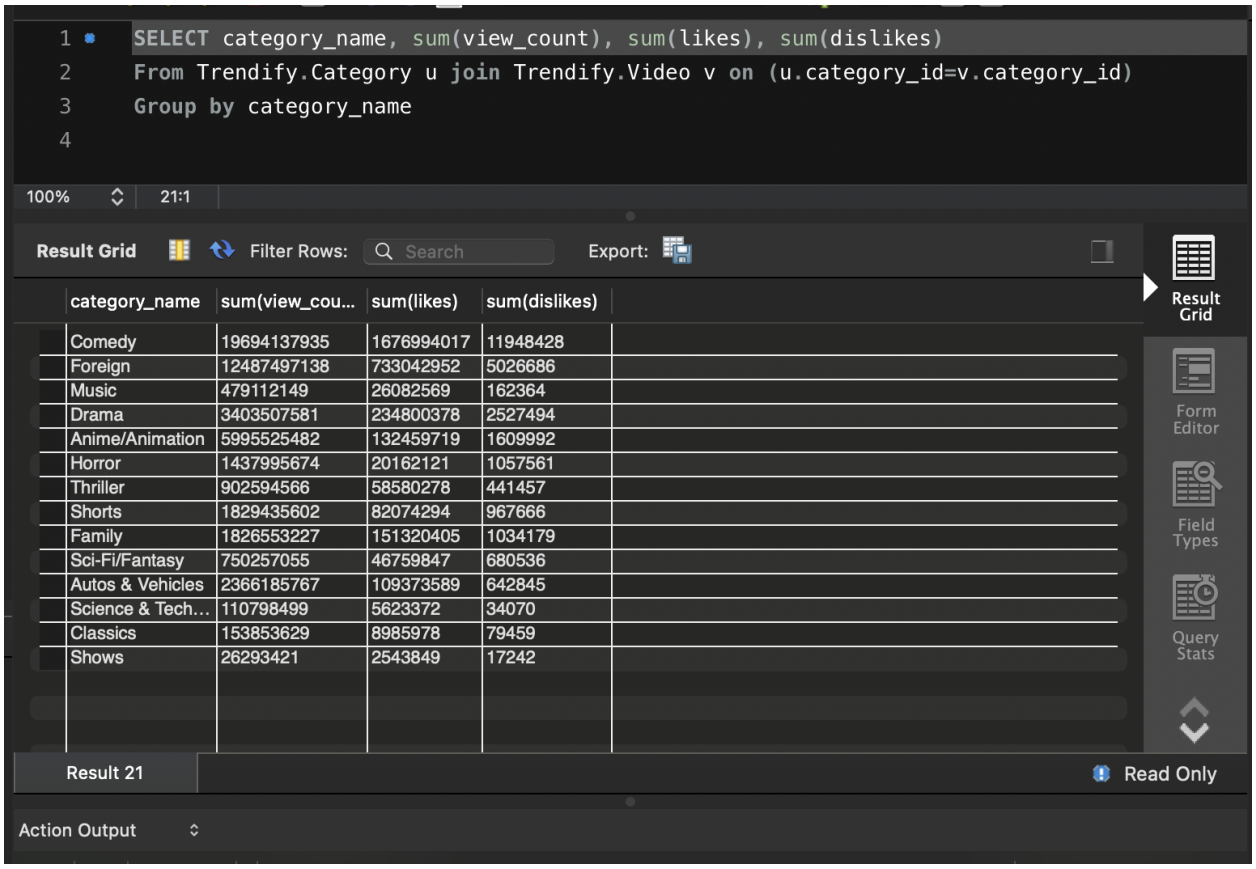
```
-> Limit: 15 row(s)  (actual time=13.563..13.565 rows=15 loops=1)
    -> Sort: t.total_likes DESC, t.total_views DESC, t.total_videos DESC, t.tag_name DESC, limit input to 15 row(s) per chunk  (actual time=13.561..13.562 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1749000.77..1767203.76 rows=1456040) (actual time=13.445..13.483 rows=159 loops=1)
            -> Temporary table with deduplication  (cost=1749000.76..1749000.76 rows=1456040) (actual time=13.441..13.441 rows=159 loops=1)
                -> Nested loop inner join  (cost=1603396.75 rows=1456040) (actual time=6.104..13.305 rows=173 loops=1)
                    -> Inner hash join (s.video_id = `<subquery2>`.video_id)  (cost=1457506.55 rows=1456040) (actual time=6.090..12.832 rows=173 loops=1)
                        -> Covering index scan on s using PRIMARY  (cost=138.94 rows=11448) (actual time=0.040..3.567 rows=11917 loops=1)
                        -> Hash
                            -> Nested loop inner join  (cost=1442.06 rows=1272) (actual time=5.975..6.002 rows=8 loops=1)
                                -> Table scan on <subquery2>  (cost=1296.25..1314.62 rows=1272) (actual time=5.944..5.946 rows=8 loops=1)
                                    -> Materialize with deduplication  (cost=1296.24..1296.24 rows=1272) (actual time=5.940..5.940 rows=8 loops=1)
                                        -> Filter: (sub.tag_name like '%dance%')  (cost=1169.05 rows=1272) (actual time=0.085..5.920 rows=12 loops=1)
                                            -> Covering index scan on sub using PRIMARY  (cost=1169.05 rows=11448) (actual time=0.074..3.578 rows=11917 loops=1)
                                -> Single-row covering index lookup on v using PRIMARY (video_id=`<subquery2>`.video_id)  (cost=0.35 rows=1) (actual time=0.006..0.007 rows=1 loops=8)
                    -> Single-row index lookup on t using PRIMARY (tag_name=s.tag_name)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=173)
```

Analysis Report: We see the same result as the previous two indexing designs. Total_videos is the last attribute we decided to index on as the third attribute. The situation here is the same as the previous two it seems.

A final note for these 3 Indexing Designs: While they technically improve the cost, the improvement is incredibly minuscule for such a large dataset that it is not much better than using the default in all 3 situations. Other values (outside of the table primary keys) would not have assisted in indexing for this query since the query only really looks at the primary keys of the tables to see if they fit requirements, outside of the values returned. Also, whole strings are parsed to look for words like “dance” so we cannot shorten them.

Query 2:

Returns the total view count, likes, and dislikes received in each category. This query can then be used to quickly retrieve data when the user inputs a category.



```
1  SELECT category_name, sum(view_count), sum(likes), sum(dislikes)
2  From Trendify.Category u join Trendify.Video v on (u.category_id=v.category_id)
3  Group by category_name
4
```

100% 21:1

Result Grid | Filter Rows: Search | Export:

category_name	sum(view_cou...	sum(likes)	sum(dislikes)
Comedy	19694137935	1676994017	11948428
Foreign	12487497138	733042952	5026686
Music	479112149	26082569	162364
Drama	3403507581	234800378	2527494
Anime/Animation	5995525482	132459719	1609992
Horror	1437995674	20162121	1057561
Thriller	902594566	58580278	441457
Shorts	1829435602	82074294	967666
Family	1826553227	151320405	1034179
Sci-Fi/Fantasy	750257055	46759847	680536
Autos & Vehicles	2366185767	109373589	642845
Science & Tech...	110798499	5623372	34070
Classics	153853629	8985978	79459
Shows	26293421	2543849	17242

Result Grid | Form Editor | Field Types | Query Stats

Result 21 | Read Only

Action Output

Indexing Analysis for Query 2:

Query analysis after running the initial Explain analyze

```
mysql> EXPLAIN ANALYZE SELECT category_name, sum(view_count), sum(likes), sum(dislikes) From Category u join Video v on (u.category_id=v.category_id) Group by category_name;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=106.699..106.702 rows=14 loops=1)
    -> Aggregate using temporary table  (actual time=106.697..106.697 rows=14 loops=1)
        -> Nested loop inner join  (cost=19629.95 rows=42856) (actual time=0.080..59.589 rows=41507 loops=1)
            -> Filter: (v.category_id is not null)  (cost=4630.35 rows=42856) (actual time=0.066..24.045 rows=41507 loops=1)
                -> Table scan on v  (cost=4630.35 rows=42856) (actual time=0.065..20.477 rows=41507 loops=1)
            -> Single-row index lookup on u using PRIMARY (category_id=v.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41507)
 |
```

Query analysis after indexing video_id in the video table

```
mysql> Create index video_idx on Video(video_id)
    -> ;
Query OK, 0 rows affected (0.46 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT category_name, sum(view_count), sum(likes), sum(dislikes) From Category u join Video v on (u.category_id=v.category_id) Group by category_name;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=149.095..149.098 rows=14 loops=1)
    -> Aggregate using temporary table  (actual time=149.093..149.093 rows=14 loops=1)
        -> Nested loop inner join  (cost=19629.95 rows=42856) (actual time=0.071..81.612 rows=41507 loops=1)
            -> Filter: (v.category_id is not null)  (cost=4630.35 rows=42856) (actual time=0.053..32.110 rows=41507 loops=1)
                -> Table scan on v  (cost=4630.35 rows=42856) (actual time=0.052..27.661 rows=41507 loops=1)
            -> Single-row index lookup on u using PRIMARY (category_id=v.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41507)
 |
+------------------------------------------------------------------------------------------------------------------------------------
```

After indexing category_id in video table

```
mysql> Create index category_idx1 on Video(category_id);
Query OK, 0 rows affected (0.42 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT category_name, sum(view_count), sum(likes), sum(dislikes) From Category u join Video v on (u.category_id=v.category_id) Group by category_name;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=106.988..106.997 rows=14 loops=1)
    -> Aggregate using temporary table  (actual time=106.985..106.985 rows=14 loops=1)
        -> Nested loop inner join  (cost=19629.95 rows=42856) (actual time=0.140..59.667 rows=41507 loops=1)
            -> Filter: (v.category_id is not null)  (cost=4630.35 rows=42856) (actual time=0.120..24.074 rows=41507 loops=1)
                -> Table scan on v  (cost=4630.35 rows=42856) (actual time=0.117..20.570 rows=41507 loops=1)
            -> Single-row index lookup on u using PRIMARY (category_id=v.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41507)
 |
```

After indexing category_id in category table

```
mysql> Create index category_idx2 on Category(category_id);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT category_name, sum(view_count), sum(likes), sum(dislikes) From Category u join Video v on (u.category_id=v.category_id) Group by category_name;
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=110.925..110.928 rows=14 loops=1)
    -> Aggregate using temporary table  (actual time=110.922..110.922 rows=14 loops=1)
        -> Nested loop inner join  (cost=19629.95 rows=42856) (actual time=0.107..62.666 rows=41507 loops=1)
            -> Filter: (v.category_id is not null)  (cost=4630.35 rows=42856) (actual time=0.088..25.812 rows=41507 loops=1)
                -> Table scan on v  (cost=4630.35 rows=42856) (actual time=0.087..22.320 rows=41507 loops=1)
            -> Single-row index lookup on u using PRIMARY (category_id=v.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41507)
 |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
```

We notice that in all three indexing strategies, the time and number of rows scanned remain the same despite indexing different columns which means this query cannot be optimized further using indexing. This is because we are joining the Video and Category tables and going through that temporary table in our query, which makes it difficult to index the new temporary table. Indexing either the Video or the Category or both the tables does not improve anything either, and in fact increases the time taken in some cases as seen in the above results.