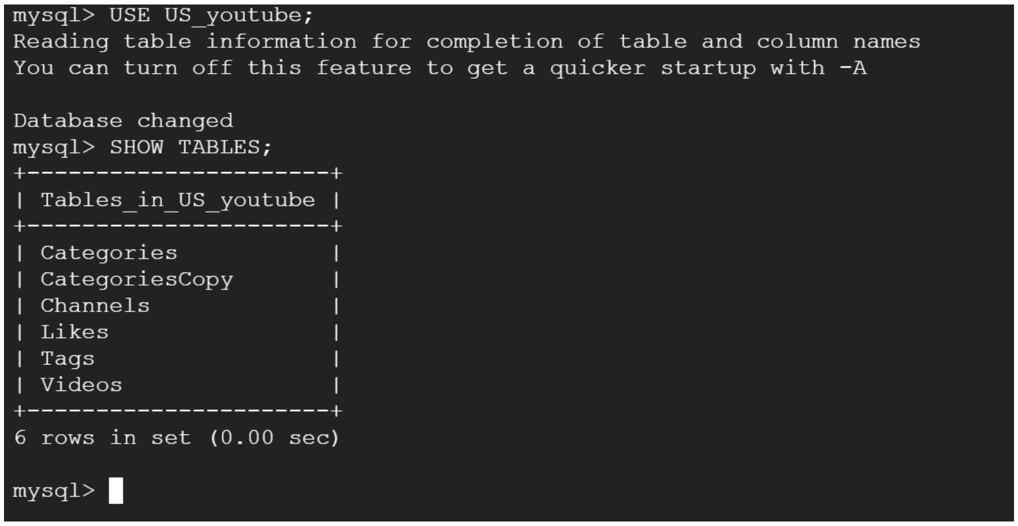
Stage 3: Database Implementation

Screenshot of GCP Connection



```
mysql> USE US_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> SHOW TABLES;
+----------------------+
| Tables_in_US_youtube |
+----------------------+
| Categories           |
| CategoriesCopy       |
| Channels             |
| Likes                |
| Tags                 |
| Videos               |
+----------------------+
6 rows in set (0.00 sec)

mysql> 
```

DDL Commands

Videos(video-id:VARCHAR(255) [PK], channel_id: VARCHAR(255), category_id:VARCHAR(255), title:VARCHAR(255), view_count: INT, likes: INT, published_at: DATE/TIME)

```
CREATE TABLE Videos(
    video_id VARCHAR(255),
    channel_id VARCHAR(255),
    category_id VARCHAR(255),
    title VARCHAR(255),
    view_count INT,
    likes INT,
    published_at DATE,
    PRIMARY KEY(video_id),
    FOREIGN KEY(channel_id) REFERENCES Channels.channelId,
    FOREIGN KEY(categoryid) REFERENCES Categories.categoryId );
```

Channels(channel_id:VARCHAR(255) [PK], video_id:VARCHAR(255) [FK to Videos.video_id], channel_title: VARCHAR(255))

```
CREATE TABLE Channels (
        channelId VARCHAR(255),
        video_id VARCHAR(255),
        channelTitle VARCHAR(255),
        PRIMARY KEY(channelId),
        FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Likes(video_id:VARCHAR(255)[PK], likes:INT, view_count: INT, dislikes: INT)

```
CREATE TABLE Likes(
      video_id VARCHAR(255),
      view_count INT,
      likes INT,
      dislikes INT,
      PRIMARY KEY(video_id),
      FOREIGN KEY(video_id) REFERENCES Videos(video_id)
);
```

Tags(video_id:VARCHAR(255)[PK], tags:VARCHAR(255))

```
 CREATE TABLE Tags (
    video_id VARCHAR(255),
    tags VARCHAR(255),
    PRIMARY KEY(video_id),
    FOREIGN KEY(video_id) REFERENCES Videos(video_id)
);
```

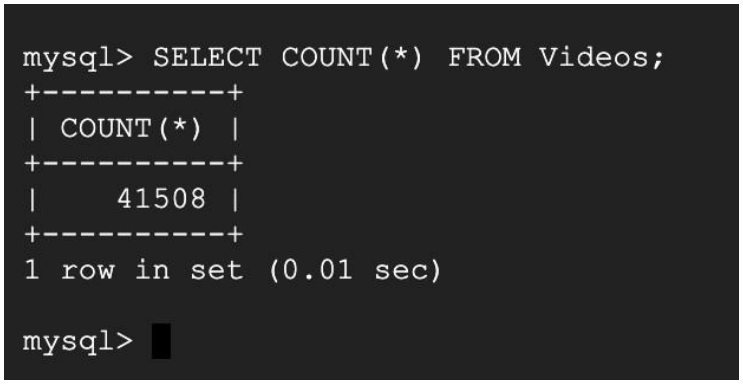
Categories(category_id:VARCHAR(255) [PK], video_id: VARCHAR(255) [FK to Videos.video_id], category_name:VARCHAR(255))

```
CREATE TABLE Categories (
        categoryId VARCHAR(255),
        video_id VARCHAR(255),
        categoryName VARCHAR(255),
        PRIMARY KEY(categoryId)
);
```

CategoriesCopy(category_id:VARCHAR(255) [PK], video_id: VARCHAR(255) [FK to Videos.video_id], category_name:VARCHAR(255))

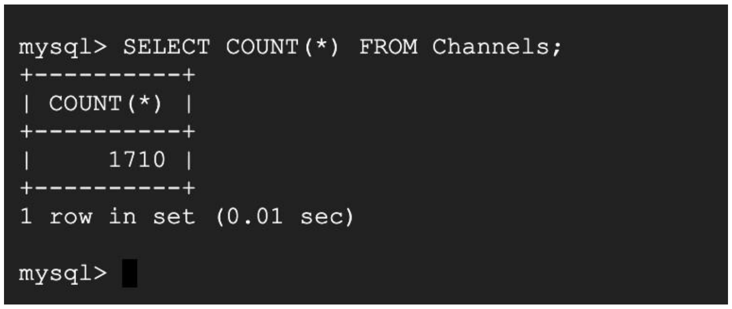
```
CREATE TABLE Categories (
	categoryId VARCHAR(255),
	categoryName VARCHAR(255),
	PRIMARY KEY(categoryId),
	FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Screenshots of >1000 Rows/Table



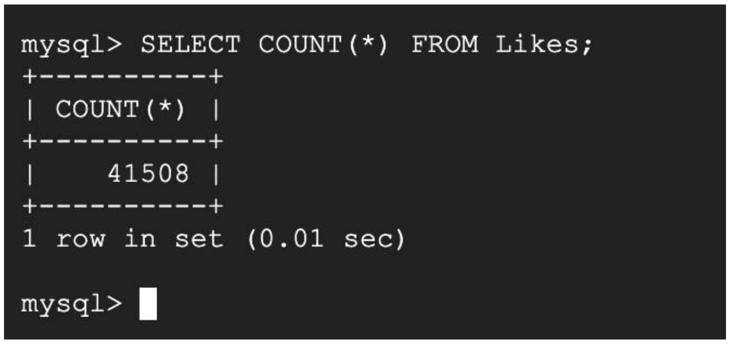
```
mysql> SELECT COUNT(*) FROM Videos;
+----------+
| COUNT(*) |
+----------+
|    41508 |
+----------+
1 row in set (0.01 sec)

mysql> 
```



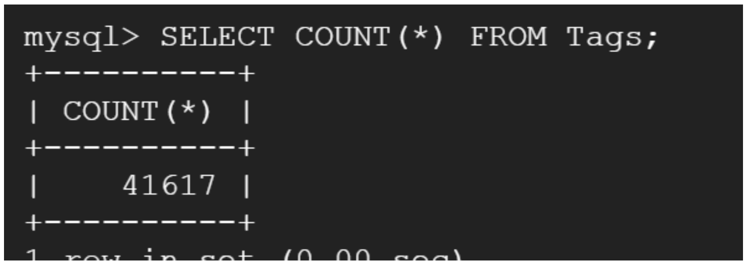
```
mysql> SELECT COUNT(*) FROM Channels;
+----------+
| COUNT(*) |
+----------+
|     1710 |
+----------+
1 row in set (0.01 sec)

mysql> 
```



```
mysql> SELECT COUNT(*) FROM Likes;
+----------+
| COUNT(*) |
+----------+
|    41508 |
+----------+
1 row in set (0.01 sec)

mysql> 
```



```
mysql> SELECT COUNT(*) FROM Tags;
+----------+
| COUNT(*) |
+----------+
|    41617 |
+----------+
1 row in set (0.00 sec)
```

Advanced Queries

1. **Impression on Categories Query**: Retrieve the channel_id & average number of likes across all videos of a given channel

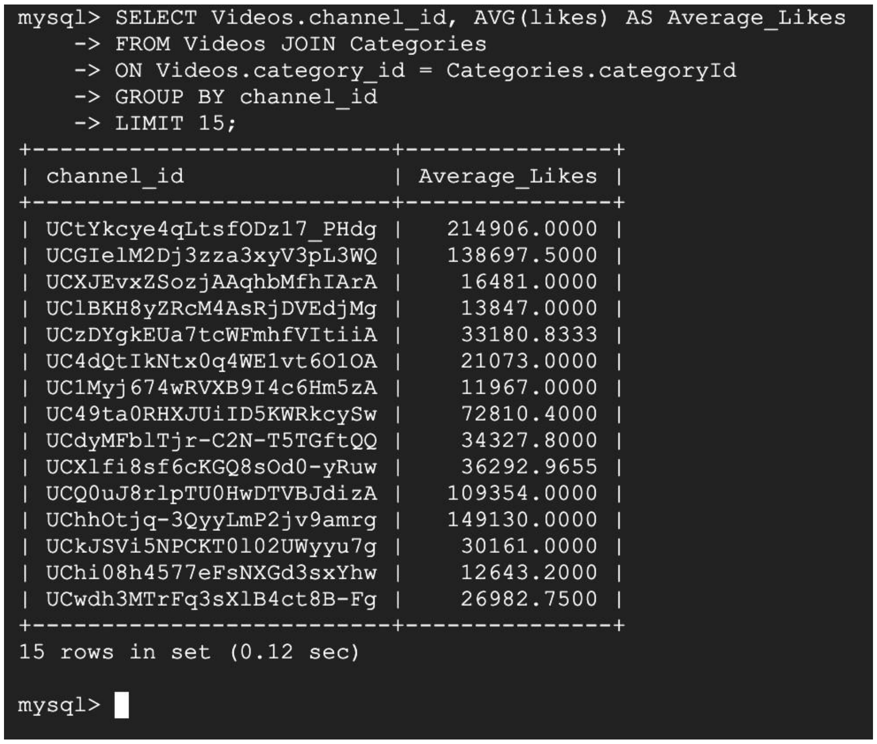
   Tables used: Videos, Categories

   SQL Concepts Used: Join of multiple relations, Aggregation via GROUP BY

   Query:

```
   SELECT Videos.channel_id, AVG(likes) AS Average_Likes,
   COUNT(Categories.categoryId) AS categoryCount
FROM Videos
JOIN Categories ON Videos.category_id = Categories.categoryId
GROUP BY Videos.channel_id;
```

   Query Output:



```
mysql> SELECT Videos.channel_id, AVG(likes) AS Average_Likes
    -> FROM Videos JOIN Categories
    -> ON Videos.category_id = Categories.categoryId
    -> GROUP BY channel_id
    -> LIMIT 15;
+--------------------------+---------------+
| channel_id               | Average_Likes |
+--------------------------+---------------+
| UCtYkcye4qLtsfODz17_PHdg |   214906.0000 |
| UCGIelM2Dj3zza3xyV3pL3WQ |   138697.5000 |
| UCXJEvxZSozjAAqhbMfhIArA |    16481.0000 |
| UClBKH8yZRcM4AsRjDVEdjMg |    13847.0000 |
| UCzDYgkEUa7tcWFmhfVItiiA |    33180.8333 |
| UC4dQtIkNtx0q4WE1vt6O1OA |    21073.0000 |
| UC1Myj674wRVXB9I4c6Hm5zA |    11967.0000 |
| UC49ta0RHXJUiID5KWRkcySw |    72810.4000 |
| UCdyMFblTjr-C2N-T5TGftQQ |    34327.8000 |
| UCXlfi8sf6cKGQ8sOd0-yRuw |    36292.9655 |
| UCQ0uJ8rlpTU0HwDTVBJdizA |   109354.0000 |
| UChhOtjq-3QyyLmP2jv9amrg |   149130.0000 |
| UCkJSVi5NPCKT0l02UWyyu7g |    30161.0000 |
| UChi08h4577eFsNXGd3sxYhw |    12643.2000 |
| UCwdh3MTrFq3sXlB4ct8B-Fg |    26982.7500 |
+--------------------------+---------------+
15 rows in set (0.12 sec)

mysql> █
```

2. **View Count and Likes Query:** Retrieve the channel title and likes for videos that have less than a 1000 views and videos that have more than 10000 views

   Tables Used: Channels, Videos

Query:

```
(SELECT Channels.channelTitle, Videos.likes,
Videos.title
FROM Videos
JOIN Channels ON Videos.channel_id = Channels.channelId
WHERE Videos.view_count < 1000)
UNION
(SELECT Channels.channelTitle, Videos.likes
FROM Videos
JOIN Channels ON Videos.channel_id = Channels.channelId
WHERE Videos.view_count > 10000)
LIMIT 15;
```

Query Output:

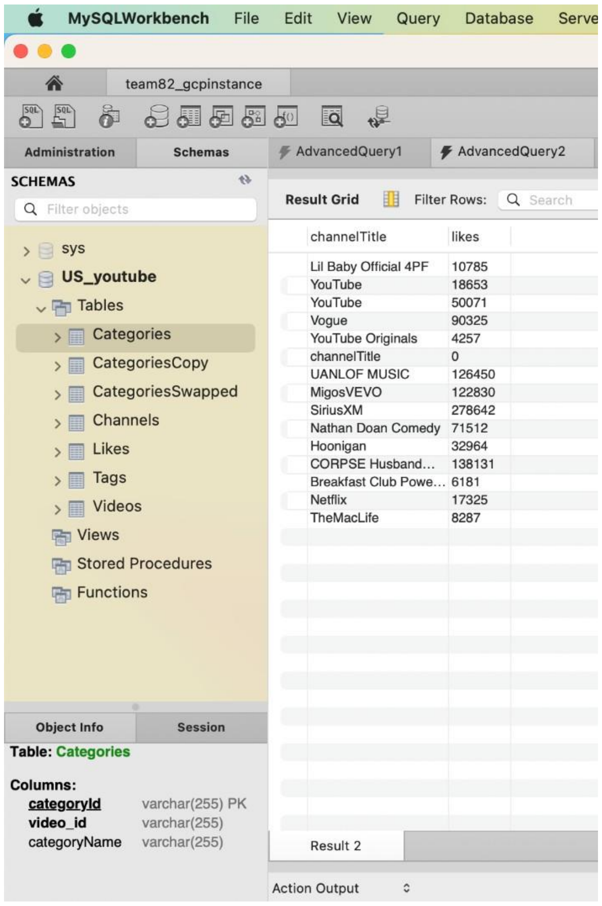


Table: Categories

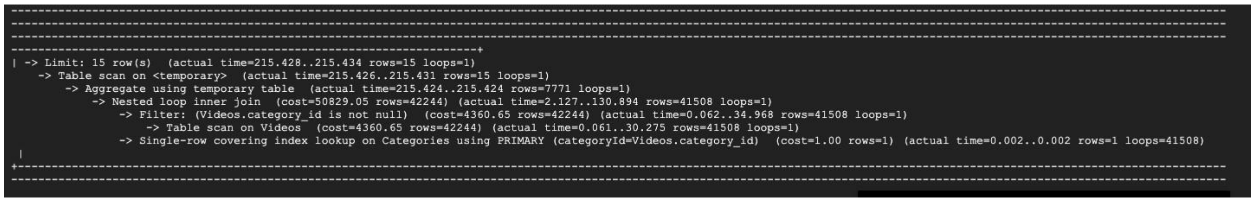
Columns:

categoryId	varchar(255) PK
video_id	varchar(255)
categoryName	varchar(255)

channelTitle	likes
Lil Baby Official 4PF	10785
YouTube	18653
YouTube	50071
Vogue	90325
YouTube Originals	4257
channelTitle	0
UANLOF MUSIC	126450
MigosVEVO	122830
SiriusXM	278642
Nathan Doan Comedy	71512
Hoonigan	32964
CORPSE Husband…	138131
Breakfast Club Powe…	6181
Netflix	17325
TheMacLife	8287

Part 2: Indexing

EXPLAIN ANALYZE:
Query 1:
Before:

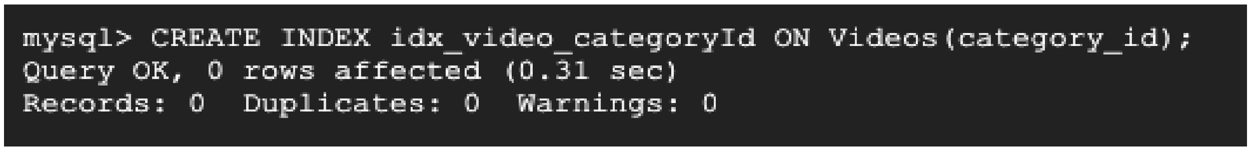


```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=215.428..215.434 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=215.426..215.431 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=215.424..215.424 rows=7771 loops=1)
            -> Nested loop inner join  (cost=50829.05 rows=42244) (actual time=2.127..130.894 rows=41508 loops=1)
                -> Filter: (Videos.category_id is not null)  (cost=4360.65 rows=42244) (actual time=0.062..34.968 rows=41508 loops=1)
                    -> Table scan on Videos  (cost=4360.65 rows=42244) (actual time=0.061..30.275 rows=41508 loops=1)
                -> Single-row covering index lookup on Categories using PRIMARY (categoryId=Videos.category_id)  (cost=1.00 rows=1) (actual time=0.002..0.002 rows=1 loops=41508)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

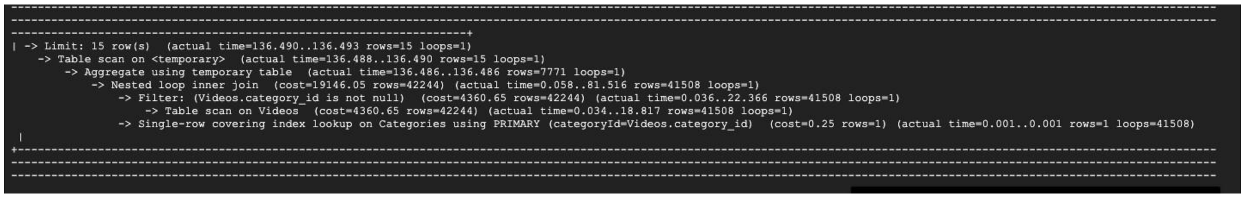
Before indexing, inner join: cost = 50829.05, lower bound = 2.127, upper bound = 130.894

- -

The first index we tried using was Videos(category_id).



```
mysql> CREATE INDEX idx_video_categoryId ON Videos(category_id);
Query OK, 0 rows affected (0.31 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



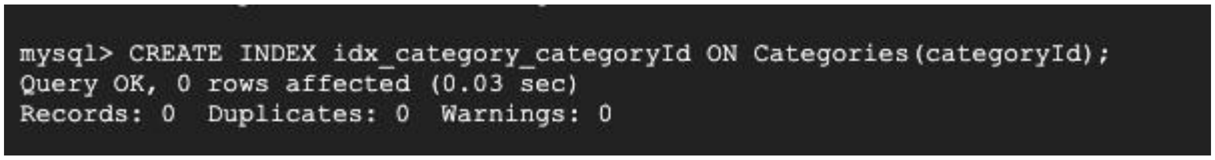
```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=136.490..136.493 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=136.488..136.490 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=136.486..136.486 rows=7771 loops=1)
            -> Nested loop inner join  (cost=19146.05 rows=42244) (actual time=0.058..81.516 rows=41508 loops=1)
                -> Filter: (Videos.category_id is not null)  (cost=4360.65 rows=42244) (actual time=0.036..22.366 rows=41508 loops=1)
                    -> Table scan on Videos  (cost=4360.65 rows=42244) (actual time=0.034..18.817 rows=41508 loops=1)
                -> Single-row covering index lookup on Categories using PRIMARY (categoryId=Videos.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41508)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Inner join: cost = 19146.05, lower bound = .058, upper bound = 81.516

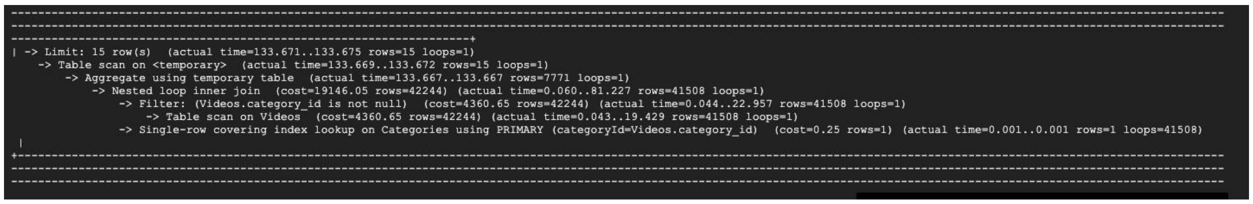
This increased our performance because we performed a join in our 1st Advance query on category_id. By indexing category_id, we make it easily accessible for the query. Instead of a full table scan, the system can now identify the category, which is used in the WHERE clause of the query. This improves performance as the category id in Videos is the main index of the table.

- -

The second index we tried using was Categories(category_id).



```
mysql> CREATE INDEX idx_category_categoryId ON Categories(categoryId);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



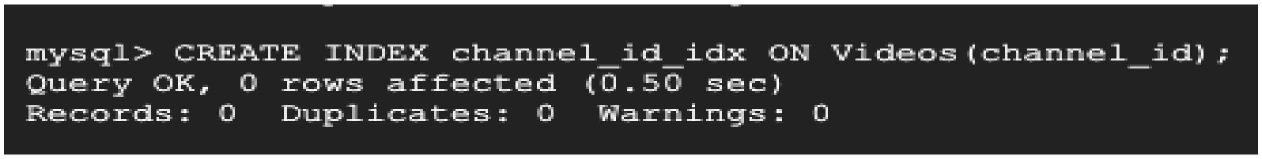
```
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=133.671..133.675 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=133.669..133.672 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=133.667..133.667 rows=7771 loops=1)
            -> Nested loop inner join  (cost=19146.05 rows=42244) (actual time=0.060..81.227 rows=41508 loops=1)
                -> Filter: (Videos.category_id is not null)  (cost=4360.65 rows=42244) (actual time=0.044..22.957 rows=41508 loops=1)
                    -> Table scan on Videos  (cost=4360.65 rows=42244) (actual time=0.043..19.429 rows=41508 loops=1)
                -> Single-row covering index lookup on Categories using PRIMARY (categoryId=Videos.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=41508)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Inner join: cost = 19146.05, Time lower bound = .060, upper bound = 81.227

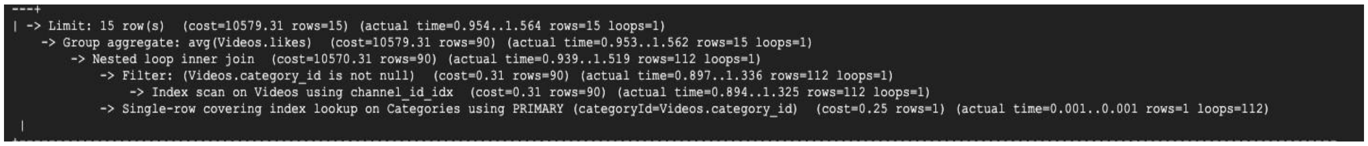
This increased our performance because we perform a join in our 1st Advance query on category_id. By indexing category_id, we make it easily accessible for the query. Here it is similar to our second indexing as we are performing indexing on category_id, however this time it is for the Categories table. This is similar to the Videos(category_id) index, but we wanted to explore if changing the table would have an effect on the performance, which had little impact.

– –

The third index we tried using was Videos(channel_id).



```
mysql> CREATE INDEX channel_id_idx ON Videos(channel_id);
Query OK, 0 rows affected (0.50 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



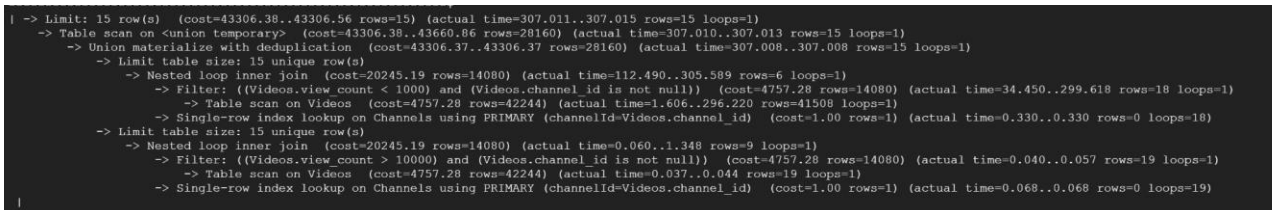
```
---+
| -> Limit: 15 row(s)  (cost=10579.31 rows=15) (actual time=0.954..1.564 rows=15 loops=1)
    -> Group aggregate: avg(Videos.likes)  (cost=10579.31 rows=90) (actual time=0.953..1.562 rows=15 loops=1)
        -> Nested loop inner join  (cost=10570.31 rows=90) (actual time=0.939..1.519 rows=112 loops=1)
            -> Filter: (Videos.category_id is not null)  (cost=0.31 rows=90) (actual time=0.897..1.336 rows=112 loops=1)
                -> Index scan on Videos using channel_id_idx  (cost=0.31 rows=90) (actual time=0.894..1.325 rows=112 loops=1)
            -> Single-row covering index lookup on Categories using PRIMARY (categoryId=Videos.category_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=112)
 |
```

Inner join: cost =10570.31, lower bound = .939, upper bound = 1.519
Explanation for improvement:
As we indexed in channel_id and our first advance query uses 'GROUP BY channel_id', it, as we are finding the average likes for each channel. Using this index increases performance significantly, since multiple values with the same channel_id are aggregated in order because of the group by, and using it as an index allows the system to quickly access each same channel_id without having to move around different rows.

We decided to use the third index on Videos(channel_id) since we are using and aggregate function on using the 'GROUP BY' on channel_id. Additionally, since category_id is essentially the same for both videos and categories, there is no marginal benefit indexing category_id. As our first ans second indexing, have the same cost and time, and our third index is significantly better that is why we chose it.

Query 2:

Before:



```
| -> Limit: 15 row(s)  (cost=43306.38..43306.56 rows=15) (actual time=307.011..307.015 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=43306.38..43660.86 rows=28160) (actual time=307.010..307.013 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=43306.37..43306.37 rows=28160) (actual time=307.008..307.008 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=20245.19 rows=14080) (actual time=112.490..305.589 rows=6 loops=1)
                    -> Filter: ((Videos.view_count < 1000) and (Videos.channel_id is not null))  (cost=4757.28 rows=14080) (actual time=34.450..299.618 rows=18 loops=1)
                        -> Table scan on Videos  (cost=4757.28 rows=42244) (actual time=1.606..296.220 rows=41508 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId=Videos.channel_id)  (cost=1.00 rows=1) (actual time=0.330..0.330 rows=0 loops=18)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=20245.19 rows=14080) (actual time=0.060..1.348 rows=9 loops=1)
                    -> Filter: ((Videos.view_count > 10000) and (Videos.channel_id is not null))  (cost=4757.28 rows=14080) (actual time=0.040..0.057 rows=19 loops=1)
                        -> Table scan on Videos  (cost=4757.28 rows=42244) (actual time=0.037..0.044 rows=19 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId=Videos.channel_id)  (cost=1.00 rows=1) (actual time=0.068..0.068 rows=0 loops=19)
 |
```

We added an index on channelTitle by using:

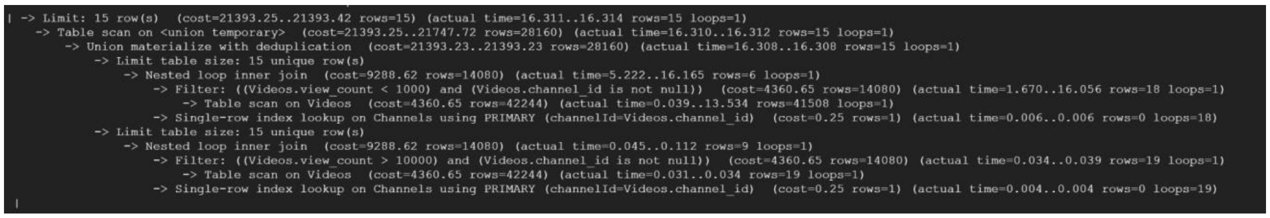
```
mysql> CREATE INDEX idx_channelTitle ON Channels(channelTitle);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Table Scan on Videos:
Before indexing, the table scan on "Videos" had a cost of 4757.28 to produce 42244 rows. The time taken was between 1.606 to 296.220 ms and produced 41508 rows in one loop.
After indexing, the table scan on "Videos" had the same cost and rows. The time taken reduced to between 0.037 to 0.044 ms and produced only 19 rows.

Single Row Index Lookup on Channels:

Before indexing: Time: 0.330 ms in one instance and 0.068 ms in another. Both produced 0 rows

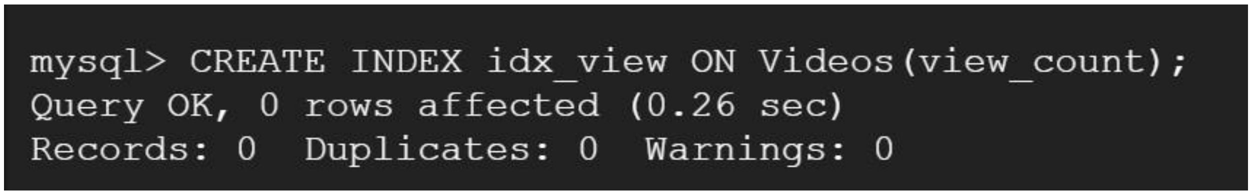
After indexing: Time: 0.330 ms in one instance and 0.068 ms in another Both produced 0 rows



```
| -> Limit: 15 row(s)  (cost=21393.25..21393.42 rows=15) (actual time=16.311..16.314 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=21393.25..21747.72 rows=28160) (actual time=16.310..16.312 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=21393.23..21393.23 rows=28160) (actual time=16.308..16.308 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=9288.62 rows=14080) (actual time=5.222..16.165 rows=6 loops=1)
                    -> Filter: ((Videos.view_count < 1000) and (Videos.channel_id is not null))  (cost=4360.65 rows=14080) (actual time=1.670..16.056 rows=18 loops=1)
                        -> Table scan on Videos  (cost=4360.65 rows=42244) (actual time=0.039..13.534 rows=41508 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId=Videos.channel_id)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=0 loops=18)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=9288.62 rows=14080) (actual time=0.045..0.112 rows=9 loops=1)
                    -> Filter: ((Videos.view_count > 10000) and (Videos.channel_id is not null))  (cost=4360.65 rows=14080) (actual time=0.034..0.039 rows=19 loops=1)
                        -> Table scan on Videos  (cost=4360.65 rows=42244) (actual time=0.031..0.034 rows=19 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId=Videos.channel_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=0 loops=19)
 |
```

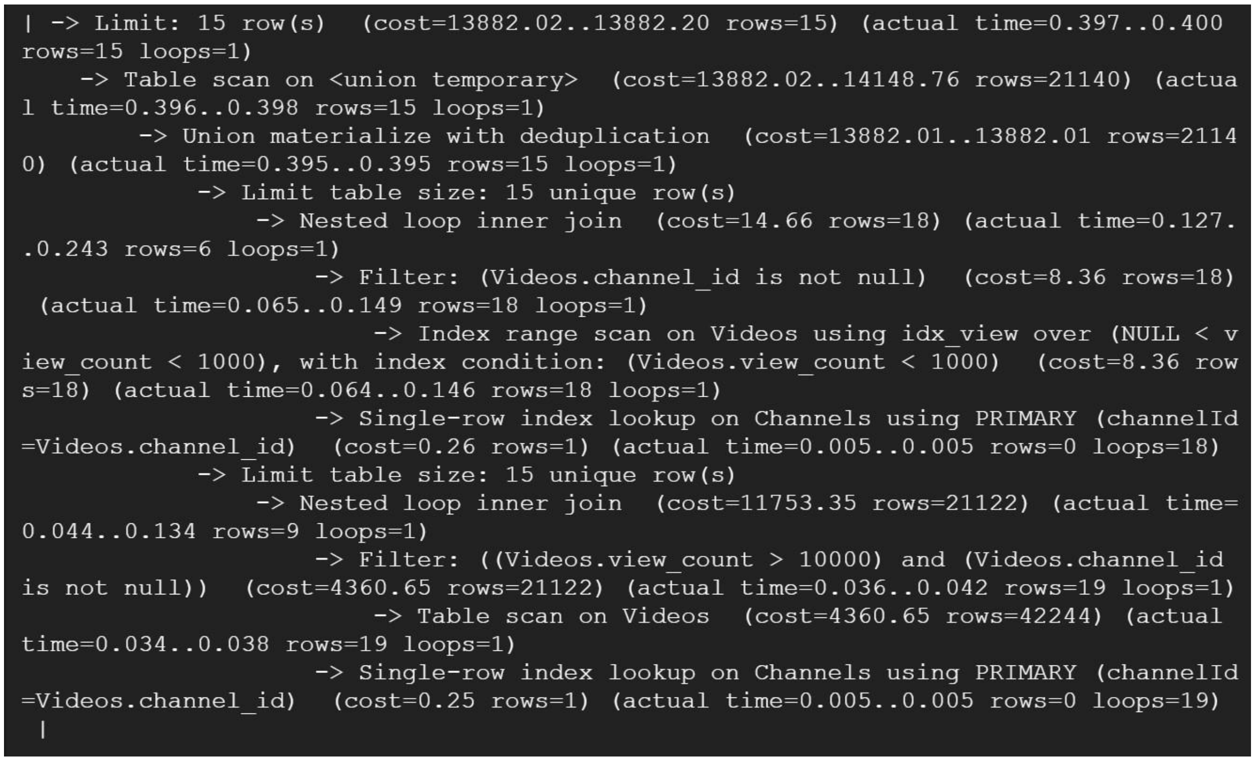
The time for table scan on "Videos" when filtering for '>1000 view_count' was significantly reduced after adding the index. However, the time remained similar for the filter. Cost remained unchanged.

Adding an index on “channelTitle” didn’t significantly change the performance, which is expected since “channelTitle” was a select variable and doesn’t directly relate to the filters provided.



```
mysql> CREATE INDEX idx_view ON Videos(view_count);
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Next, we are adding index on view_count



```
| -> Limit: 15 row(s)  (cost=13882.02..13882.20 rows=15) (actual time=0.397..0.400
rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=13882.02..14148.76 rows=21140) (actua
l time=0.396..0.398 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=13882.01..13882.01 rows=2114
0) (actual time=0.395..0.395 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=14.66 rows=18) (actual time=0.127.
.0.243 rows=6 loops=1)
                    -> Filter: (Videos.channel_id is not null)  (cost=8.36 rows=18)
 (actual time=0.065..0.149 rows=18 loops=1)
                        -> Index range scan on Videos using idx_view over (NULL < v
iew_count < 1000), with index condition: (Videos.view_count < 1000)  (cost=8.36 row
s=18) (actual time=0.064..0.146 rows=18 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId
=Videos.channel_id)  (cost=0.26 rows=1) (actual time=0.005..0.005 rows=0 loops=18)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=11753.35 rows=21122) (actual time=
0.044..0.134 rows=9 loops=1)
                    -> Filter: ((Videos.view_count > 10000) and (Videos.channel_id
is not null))  (cost=4360.65 rows=21122) (actual time=0.036..0.042 rows=19 loops=1)
                        -> Table scan on Videos  (cost=4360.65 rows=42244) (actual
time=0.034..0.038 rows=19 loops=1)
                    -> Single-row index lookup on Channels using PRIMARY (channelId
=Videos.channel_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=0 loops=19)
 |
```

Before:

Cost for “Videos” table scan: 4757.28

Time for “Videos” table scan: 34.450 ms - 299.618 ms

After:

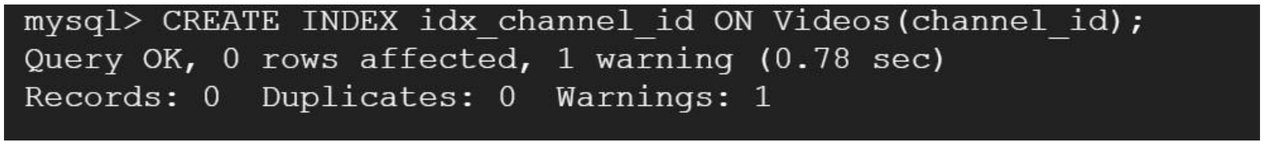
Cost for “Index range scan on Videos using idx_view”: 8.36

Time: 0.064 ms - 0.146 ms

Cost and time both drastically decreased when accessing the “Videos” table.

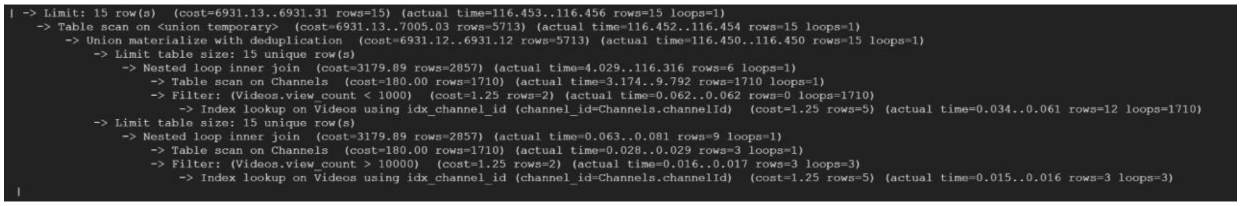
Reasons for improvement:

The presence of the index means that conditions like ‘Videos.view_count < 1000’ can be evaluated more efficiently using the indexed column.



```
mysql> CREATE INDEX idx_channel_id ON Videos(channel_id);
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

We created index on channel_id



```
| -> Limit: 15 row(s)  (cost=6931.13..6931.31 rows=15) (actual time=116.453..116.456 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=6931.13..7005.03 rows=5713) (actual time=116.452..116.454 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=6931.12..6931.12 rows=5713) (actual time=116.450..116.450 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=3179.89 rows=2857) (actual time=4.029..116.316 rows=6 loops=1)
                    -> Table scan on Channels  (cost=180.00 rows=1710) (actual time=3.174..9.792 rows=1710 loops=1)
                    -> Filter: (Videos.view_count < 1000)  (cost=1.25 rows=2) (actual time=0.062..0.062 rows=0 loops=1710)
                        -> Index lookup on Videos using idx_channel_id (channel_id=Channels.channelId)  (cost=1.25 rows=5) (actual time=0.034..0.061 rows=12 loops=1710)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=3179.89 rows=2857) (actual time=0.063..0.081 rows=9 loops=1)
                    -> Table scan on Channels  (cost=180.00 rows=1710) (actual time=0.028..0.029 rows=3 loops=1)
                    -> Filter: (Videos.view_count > 10000)  (cost=1.25 rows=2) (actual time=0.016..0.017 rows=3 loops=3)
                        -> Index lookup on Videos using idx_channel_id (channel_id=Channels.channelId)  (cost=1.25 rows=5) (actual time=0.015..0.016 rows=3 loops=3)
 |
```

The execution time saw a notable decrease and went down from around 307.011 ms to 116.452 ms in the new one.
The actual time for scanning “Videos” decreased from 1.606-299.620 ms to 0.034-0.061 ms and 0.015-0.016 seconds for different conditions.
In summary, adding the index on ‘channel_id’ resulted in a significant decrease in both cost and time of execution for the query.