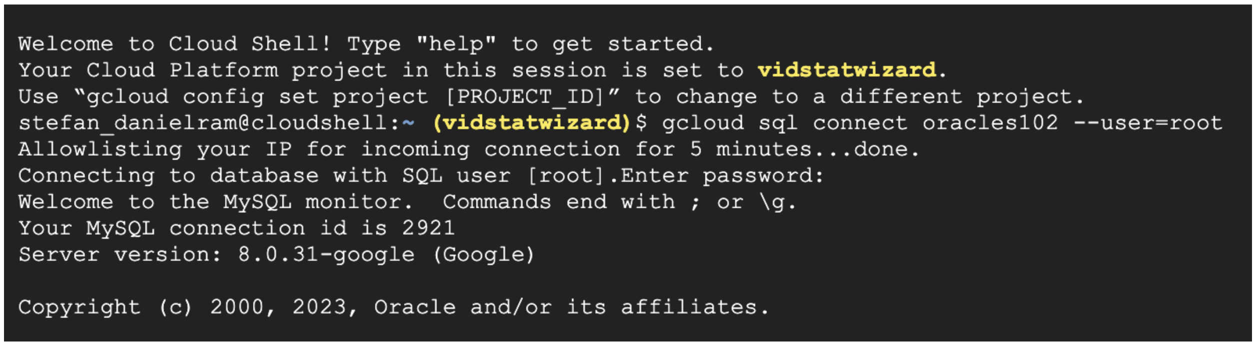
Database Design & Implementation [PT3]

Part 1: Implementation

GCP Connection:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to vidstatwizard.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
stefan_danielram@cloudshell:~ (vidstatwizard)$ gcloud sql connect oracles102 --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 2921
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.
```

DDL Commands:

```
CREATE TABLE User (
      UserId VARCHAR(20),
      Email VARCHAR(255),
      Password VARCHAR(255,
      Name VARCHAR(255),
      PRIMARY KEY (UserId)
);
```

```
CREATE TABLE WatchLaterList (
      UserId VARCHAR(20),
      VideoId VARCHAR(20),
      DateAdded DATE,
      PRIMARY KEY (UserId, VideoId),
      FOREIGN KEY (UserId) REFERENCES User(UserId) ON DELETE CASCADE,
      FOREIGN KEY (VideoId) REFERENCES Video(VideoId) ON DELETE CASCADE
);
```

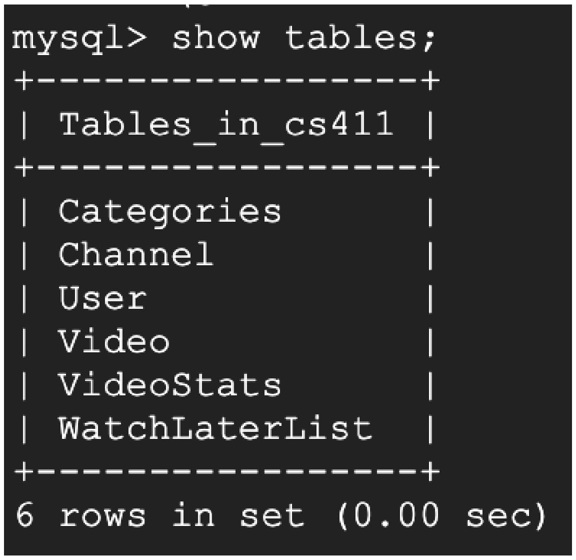
```
CREATE TABLE Video (
      VideoId VARCHAR(20),
      ChannelId VARCHAR(30),
      CategoryId INT,
      Link VARCHAR(255),
      Title VARCHAR(255),
      ThumbnailLink VARCHAR(255),
      Description TEXT,
      Tags TEXT,
      PRIMARY KEY (VideoId),
      FOREIGN KEY (ChannelId) REFERENCES Channel ON DELETE CASCADE,
            FOREIGN KEY (CategoryId) REFERENCES Categories
      );
```

```
      CREATE TABLE Channel (
            ChannelId VARCHAR(30),
            Title VARCHAR(255),
            PRIMARY KEY (ChannelId)
      );
```

```
      CREATE TABLE VideoStats (
            VideoId VARCHAR(20),
            Likes INT,
            Dislikes INT,
            ViewCount INT,
            TrendingDate DATE,
            CommentCount INT,
            PublishedAt DATE,
            PRIMARY KEY (VideoId),
            FOREIGN KEY (VideoId) REFERENCES Video(VideoId) ON DELETE CASCADE
      );
```

```
      CREATE TABLE Categories (
            CategoryId INT,
            Category VARCHAR(30),
            PRIMARY KEY (CategoryId)
      );
```

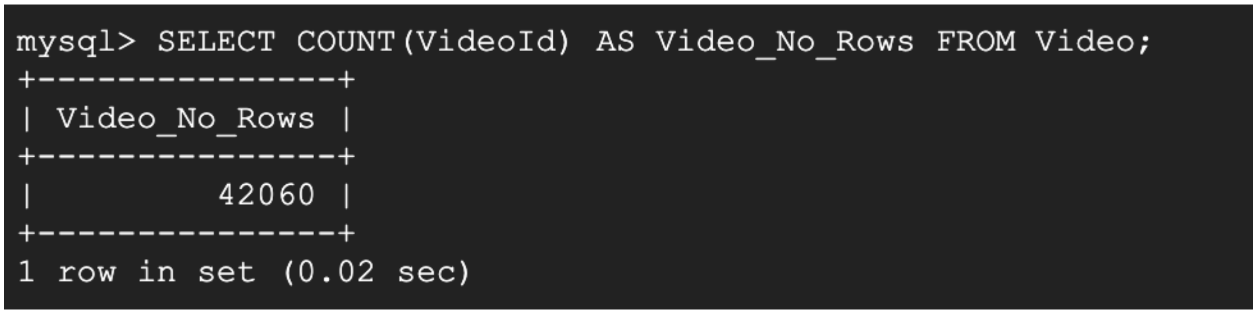
Results in ->



```
mysql> show tables;
+-----------------+
| Tables_in_cs411 |
+-----------------+
| Categories      |
| Channel         |
| User            |
| Video           |
| VideoStats      |
| WatchLaterList  |
+-----------------+
6 rows in set (0.00 sec)
```

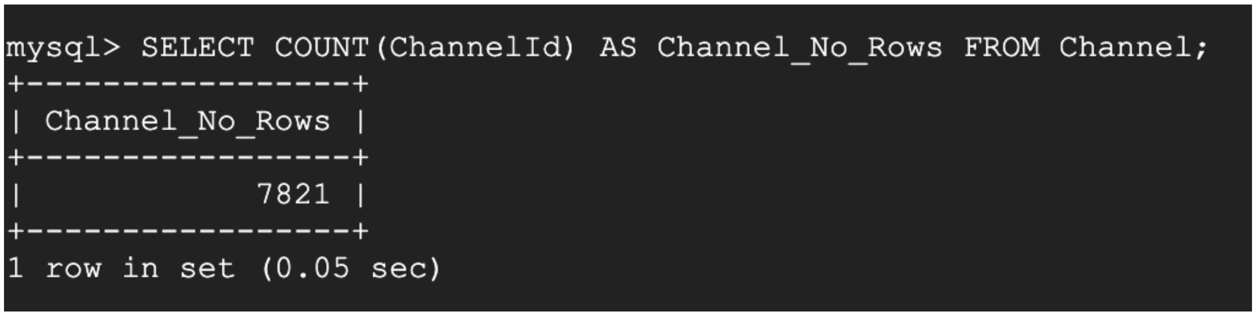
Count Queries to prove that values have been inserted:

Number of rows in Video table:



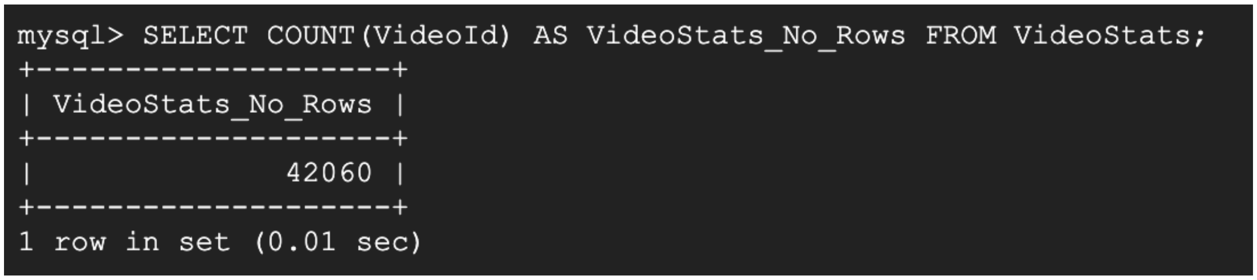
```
mysql> SELECT COUNT(VideoId) AS Video_No_Rows FROM Video;
+---------------+
| Video_No_Rows |
+---------------+
|         42060 |
+---------------+
1 row in set (0.02 sec)
```

Number of rows in Channel table:



```
mysql> SELECT COUNT(ChannelId) AS Channel_No_Rows FROM Channel;
+-----------------+
| Channel_No_Rows |
+-----------------+
|            7821 |
+-----------------+
1 row in set (0.05 sec)
```

Number of rows in VideoStats table:



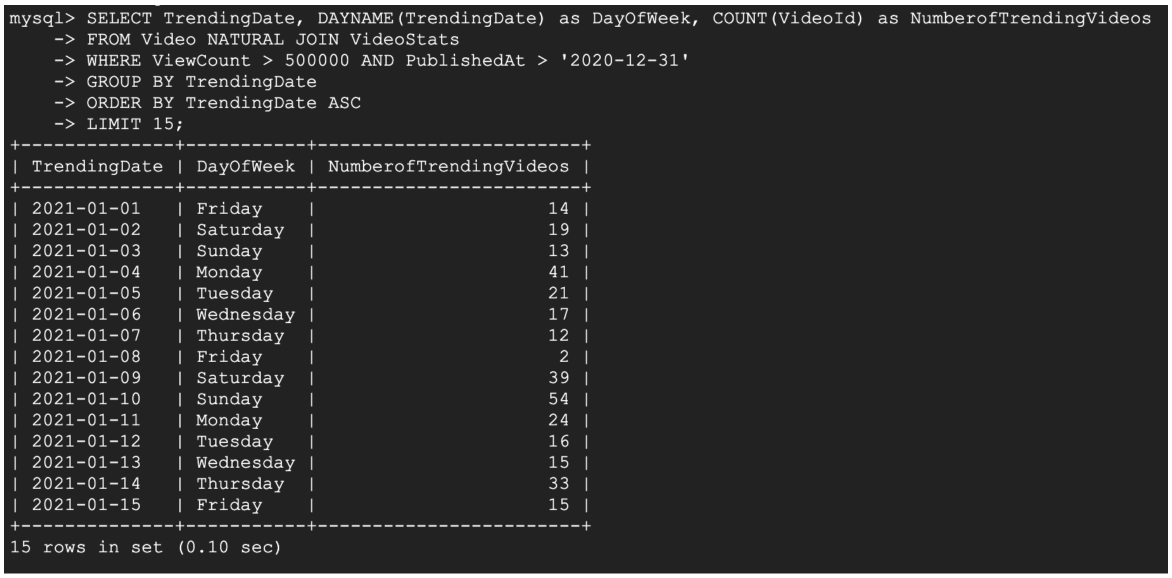
```
mysql> SELECT COUNT(VideoId) AS VideoStats_No_Rows FROM VideoStats;
+--------------------+
| VideoStats_No_Rows |
+--------------------+
|              42060 |
+--------------------+
1 row in set (0.01 sec)
```

Advanced Queries:

AQ1:

Number of videos trending on a certain trending date and day.

```
SELECT TrendingDate, DAYNAME(TrendingDate) as DayOfWeek, COUNT(VideoId) as
NumberofTrendingVideos
FROM Video NATURAL JOIN VideoStats
WHERE ViewCount > 500000 AND PublishedAt > '2020-12-31'
GROUP BY TrendingDate
ORDER BY TrendingDate ASC
LIMIT 15;
```

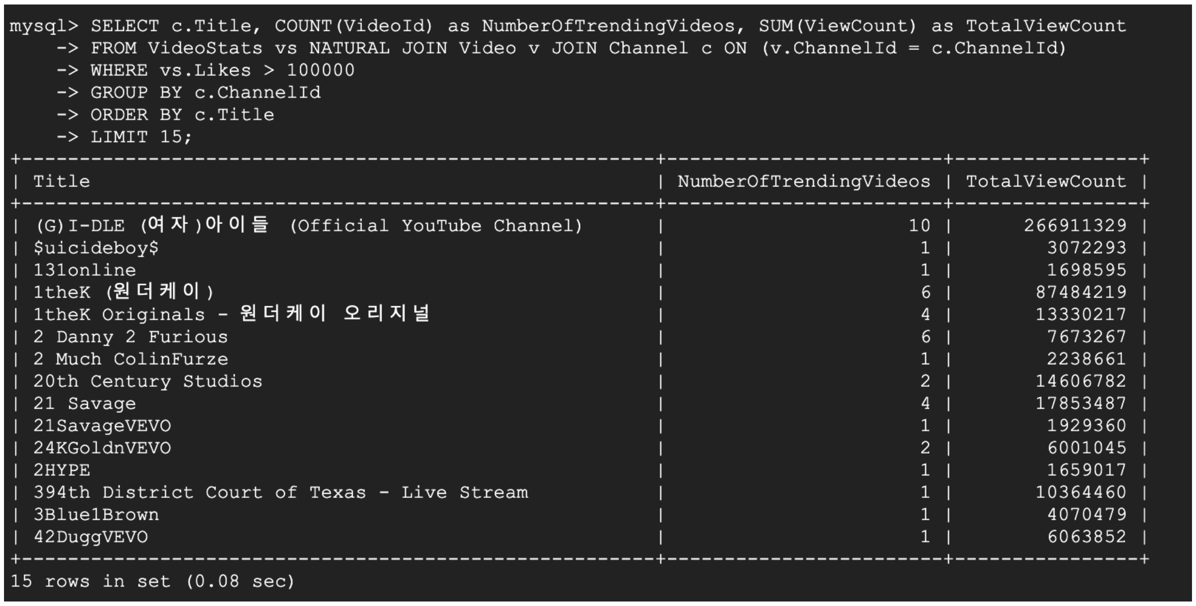


```
mysql> SELECT TrendingDate, DAYNAME(TrendingDate) as DayOfWeek, COUNT(VideoId) as NumberofTrendingVideos
    -> FROM Video NATURAL JOIN VideoStats
    -> WHERE ViewCount > 500000 AND PublishedAt > '2020-12-31'
    -> GROUP BY TrendingDate
    -> ORDER BY TrendingDate ASC
    -> LIMIT 15;
+--------------+-----------+------------------------+
| TrendingDate | DayOfWeek | NumberofTrendingVideos |
+--------------+-----------+------------------------+
| 2021-01-01   | Friday    |                     14 |
| 2021-01-02   | Saturday  |                     19 |
| 2021-01-03   | Sunday    |                     13 |
| 2021-01-04   | Monday    |                     41 |
| 2021-01-05   | Tuesday   |                     21 |
| 2021-01-06   | Wednesday |                     17 |
| 2021-01-07   | Thursday  |                     12 |
| 2021-01-08   | Friday    |                      2 |
| 2021-01-09   | Saturday  |                     39 |
| 2021-01-10   | Sunday    |                     54 |
| 2021-01-11   | Monday    |                     24 |
| 2021-01-12   | Tuesday   |                     16 |
| 2021-01-13   | Wednesday |                     15 |
| 2021-01-14   | Thursday  |                     33 |
| 2021-01-15   | Friday    |                     15 |
+--------------+-----------+------------------------+
15 rows in set (0.10 sec)
```

AQ2:

Channels with the their total ViewCount and the corresponding number of trending videos.

```
SELECT c.Title, COUNT(VideoId) as NumberOfTrendingVideos, SUM(ViewCount) as
TotalViewCount
FROM VideoStats vs NATURAL JOIN Video v JOIN Channel c ON (v.ChannelId =
c.ChannelId)
WHERE vs.Likes > 100000
GROUP BY c.ChannelId
ORDER BY c.Title
LIMIT 15;
```



```
mysql> SELECT c.Title, COUNT(VideoId) as NumberOfTrendingVideos, SUM(ViewCount) as TotalViewCount
    -> FROM VideoStats vs NATURAL JOIN Video v JOIN Channel c ON (v.ChannelId = c.ChannelId)
    -> WHERE vs.Likes > 100000
    -> GROUP BY c.ChannelId
    -> ORDER BY c.Title
    -> LIMIT 15;
+--------------------------------------------------------+------------------------+----------------+
| Title                                                  | NumberOfTrendingVideos | TotalViewCount |
+--------------------------------------------------------+------------------------+----------------+
| (G)I-DLE (여자)아이들 (Official YouTube Channel)          |                     10 |      266911329 |
| $uicideboy$                                            |                      1 |        3072293 |
| 131online                                              |                      1 |        1698595 |
| 1theK (원더케이)                                         |                      6 |       87484219 |
| 1theK Originals - 원더케이 오리지널                        |                      4 |       13330217 |
| 2 Danny 2 Furious                                      |                      6 |        7673267 |
| 2 Much ColinFurze                                      |                      1 |        2238661 |
| 20th Century Studios                                   |                      2 |       14606782 |
| 21 Savage                                              |                      4 |       17853487 |
| 21SavageVEVO                                           |                      1 |        1929360 |
| 24KGoldnVEVO                                           |                      2 |        6001045 |
| 2HYPE                                                  |                      1 |        1659017 |
| 394th District Court of Texas - Live Stream            |                      1 |       10364460 |
| 3Blue1Brown                                            |                      1 |        4070479 |
| 42DuggVEVO                                             |                      1 |        6063852 |
+--------------------------------------------------------+------------------------+----------------+
15 rows in set (0.08 sec)
```

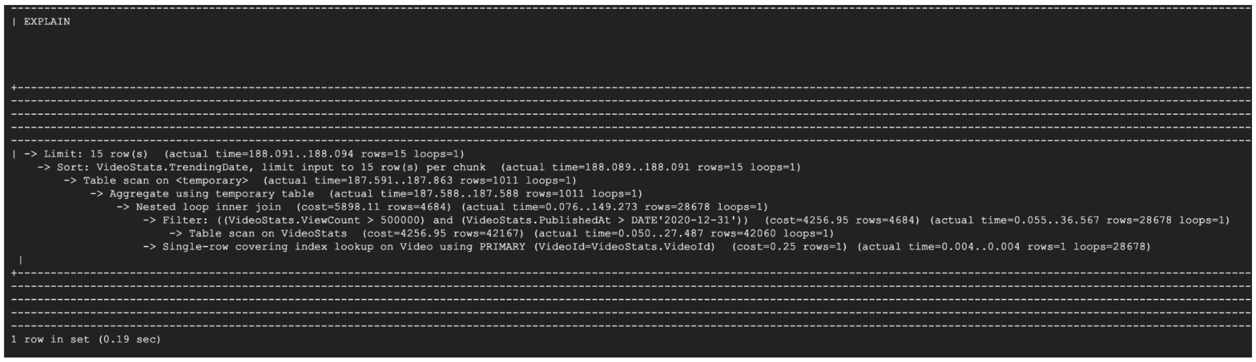
Part 2: Indexing

AQ1
We explored three different index designs for optimizing our query. In the first and second designs, we initially chose to index the "PublishedAt" and "ViewCount" attributes mentioned in the WHERE clause. However, upon analyzing the query execution using the EXPLAIN ANALYZE feature, we discovered that the newly created indexes didn't contribute to better query performance. Instead, the primary key index was consistently used meaning that table scan was used instead of index scan.

For the third indexing design, we focused on the "TrendingDate" attribute used in the GROUP BY and ORDER BY clauses of our query. The EXPLAIN ANALYZE results for this design showed a remarkable improvement. The overall query execution time decreased by more than tenfold, signifying a significant enhancement in performance.

After carefully evaluating all the associated costs and query execution times, we reached the conclusion that the third indexing design, which centered around the "TrendingDate" attribute, proved to be the optimal choice. This decision was driven by its performance improvements and the substantial reduction in query execution time.

Default:



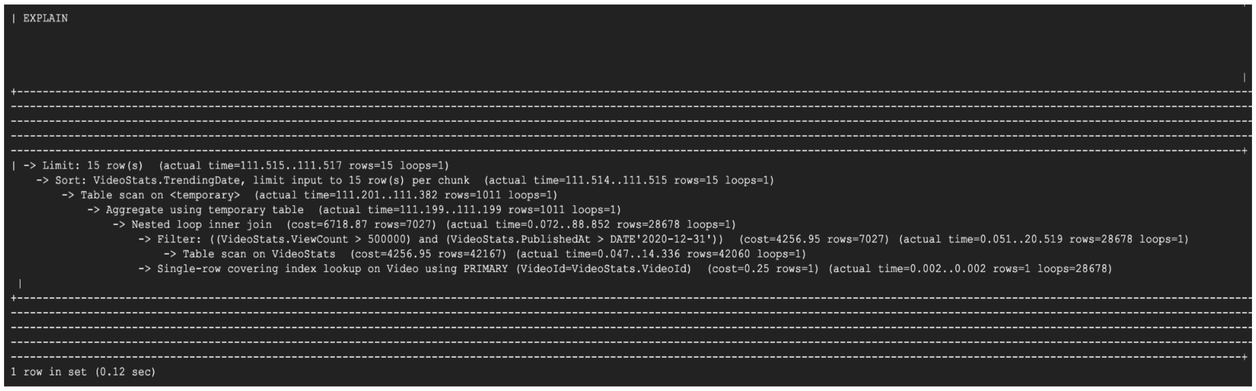
```
------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



+-----------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (actual time=188.091..188.094 rows=15 loops=1)
    -> Sort: VideoStats.TrendingDate, limit input to 15 row(s) per chunk  (actual time=188.089..188.091 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=187.591..187.863 rows=1011 loops=1)
            -> Aggregate using temporary table  (actual time=187.588..187.588 rows=1011 loops=1)
                -> Nested loop inner join  (cost=5898.11 rows=4684) (actual time=0.076..149.273 rows=28678 loops=1)
                    -> Filter: ((VideoStats.ViewCount > 500000) and (VideoStats.PublishedAt > DATE'2020-12-31'))  (cost=4256.95 rows=4684) (actual time=0.055..36.567 rows=28678 loops=1)
                        -> Table scan on VideoStats  (cost=4256.95 rows=42167) (actual time=0.050..27.487 rows=42060 loops=1)
                    -> Single-row covering index lookup on Video using PRIMARY (VideoId=VideoStats.VideoId)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=28678)
 |
+-----------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
1 row in set (0.19 sec)
```

Indexing Design 1: Indexing on VideoStats(PublishedAt)

```
CREATE INDEX idx_pa ON VideoStats (PublishedAt);
```



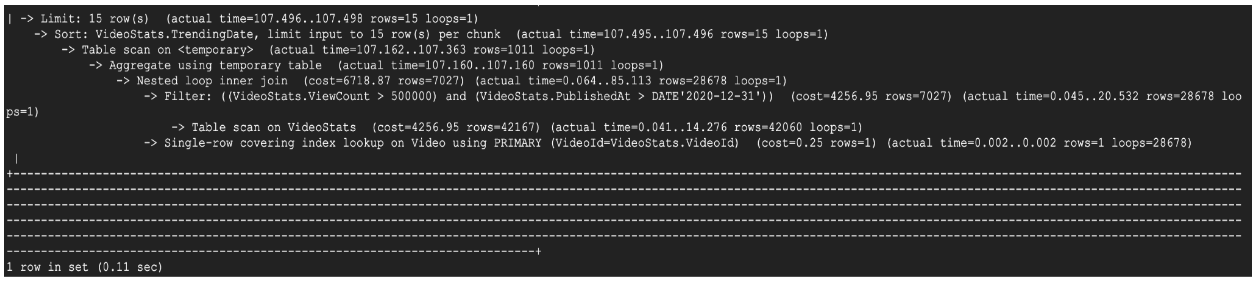
```
| EXPLAIN

                                                                                                                                   |
+----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=111.515..111.517 rows=15 loops=1)
    -> Sort: VideoStats.TrendingDate, limit input to 15 row(s) per chunk  (actual time=111.514..111.515 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=111.201..111.382 rows=1011 loops=1)
            -> Aggregate using temporary table  (actual time=111.199..111.199 rows=1011 loops=1)
                -> Nested loop inner join  (cost=6718.87 rows=7027) (actual time=0.072..88.852 rows=28678 loops=1)
                    -> Filter: ((VideoStats.ViewCount > 500000) and (VideoStats.PublishedAt > DATE'2020-12-31'))  (cost=4256.95 rows=7027) (actual time=0.051..20.519 rows=28678 loops=1)
                        -> Table scan on VideoStats  (cost=4256.95 rows=42167) (actual time=0.047..14.336 rows=42060 loops=1)
                    -> Single-row covering index lookup on Video using PRIMARY (VideoId=VideoStats.VideoId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=28678)
 |
+----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.12 sec)
```

There is no overall difference in performance with the default indexing.

Indexing Design 2: Indexing on Video(PublishedAt)

`CREATE INDEX idx_vc ON VideoStats (ViewCount);`

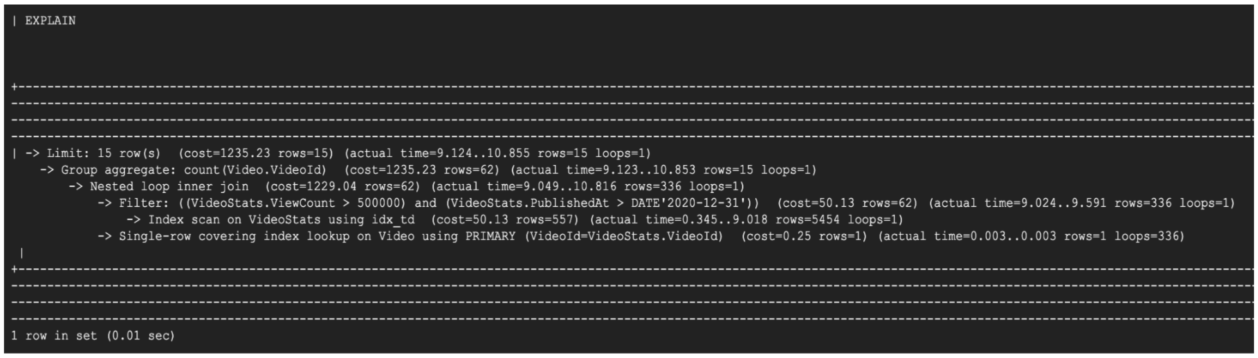


```
| -> Limit: 15 row(s)  (actual time=107.496..107.498 rows=15 loops=1)
    -> Sort: VideoStats.TrendingDate, limit input to 15 row(s) per chunk  (actual time=107.495..107.496 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=107.162..107.363 rows=1011 loops=1)
            -> Aggregate using temporary table  (actual time=107.160..107.160 rows=1011 loops=1)
                -> Nested loop inner join  (cost=6718.87 rows=7027) (actual time=0.064..85.113 rows=28678 loops=1)
                    -> Filter: ((VideoStats.ViewCount > 500000) and (VideoStats.PublishedAt > DATE'2020-12-31'))  (cost=4256.95 rows=7027) (actual time=0.045..20.532 rows=28678 loo
ps=1)
                        -> Table scan on VideoStats  (cost=4256.95 rows=42167) (actual time=0.041..14.276 rows=42060 loops=1)
                    -> Single-row covering index lookup on Video using PRIMARY (VideoId=VideoStats.VideoId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=28678)
 |
+---------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
1 row in set (0.11 sec)
```

There is no overall difference in performance with the default indexing.

Indexing Design 3: `Indexing on VideoStats(TrendingDate)`

`CREATE INDEX idx_td ON VideoStats (TrendingDate);`



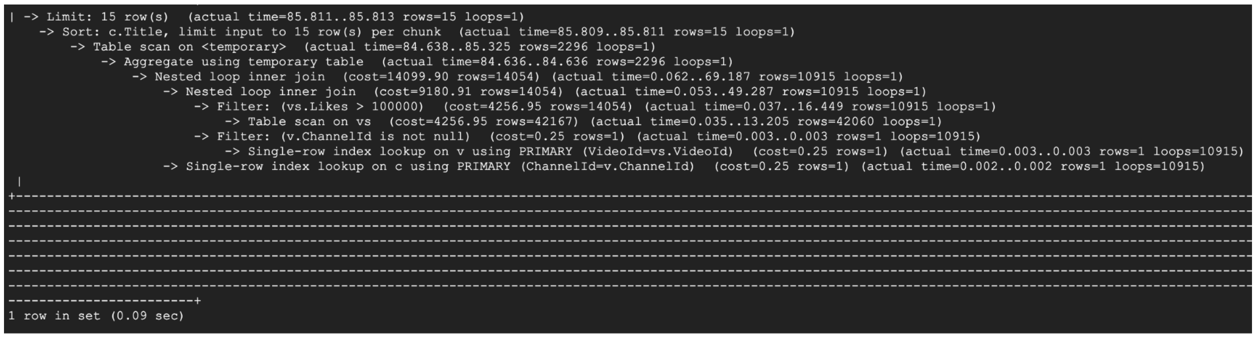
```
| EXPLAIN

+----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=1235.23 rows=15) (actual time=9.124..10.855 rows=15 loops=1)
    -> Group aggregate: count(Video.VideoId)  (cost=1235.23 rows=62) (actual time=9.123..10.853 rows=15 loops=1)
        -> Nested loop inner join  (cost=1229.04 rows=62) (actual time=9.049..10.816 rows=336 loops=1)
            -> Filter: ((VideoStats.ViewCount > 500000) and (VideoStats.PublishedAt > DATE'2020-12-31'))  (cost=50.13 rows=62) (actual time=9.024..9.591 rows=336 loops=1)
                -> Index scan on VideoStats using idx_td  (cost=50.13 rows=557) (actual time=0.345..9.018 rows=5454 loops=1)
            -> Single-row covering index lookup on Video using PRIMARY (VideoId=VideoStats.VideoId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=336)
 |
+----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

There is a significant difference in performance with a faster runtime compared to the default indexing.

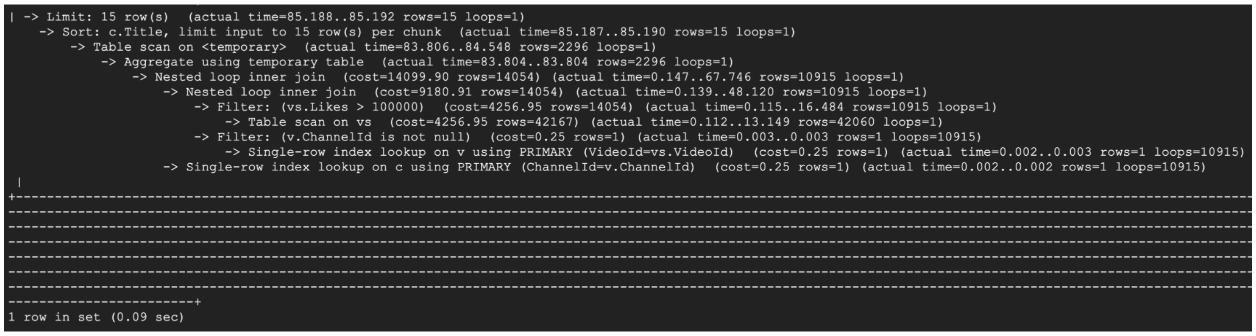
AQ2:
We chose to test indexing on Channel.Title, Video.Likes, and also both. Channel.Title is used in the ORDER BY clause, thus it would be beneficial if we try to index on this attribute. Similarly, Video.Likes was used in the WHERE clause, which would most likely improve the query performance if we index on this attribute. However, based on the observations, there is no improvement in adding these indexes. The query processor consistently opted for the table scan or primary key lookups instead of an index scan on the newly created indexes. We believe that the nature of our Video entity that we joined, which consists of a primary key and 2 foreign keys, does the job in having the most optimized query performance because there are 3 indexes directly available on it, facilitating efficient data retrieval and join.

Default:



```
| -> Limit: 15 row(s)  (actual time=85.811..85.813 rows=15 loops=1)
    -> Sort: c.Title, limit input to 15 row(s) per chunk  (actual time=85.809..85.811 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=84.638..85.325 rows=2296 loops=1)
            -> Aggregate using temporary table  (actual time=84.636..84.636 rows=2296 loops=1)
                -> Nested loop inner join  (cost=14099.90 rows=14054) (actual time=0.062..69.187 rows=10915 loops=1)
                    -> Nested loop inner join  (cost=9180.91 rows=14054) (actual time=0.053..49.287 rows=10915 loops=1)
                        -> Filter: (vs.Likes > 100000)  (cost=4256.95 rows=14054) (actual time=0.037..16.449 rows=10915 loops=1)
                            -> Table scan on vs  (cost=4256.95 rows=42167) (actual time=0.035..13.205 rows=42060 loops=1)
                        -> Filter: (v.ChannelId is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=10915)
                            -> Single-row index lookup on v using PRIMARY (VideoId=vs.VideoId)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=10915)
                    -> Single-row index lookup on c using PRIMARY (ChannelId=v.ChannelId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.09 sec)
```

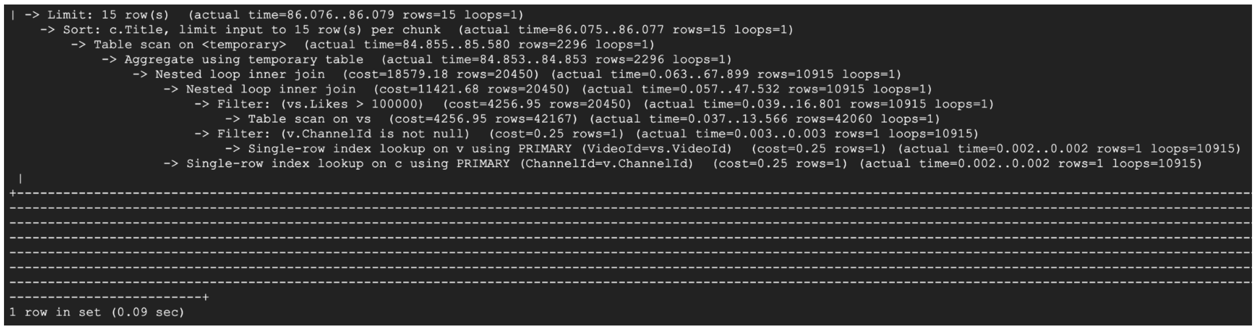
Indexing Design 1: Indexing on Channel(Title)
`CREATE INDEX idx_c on Channel (Title);`



```
| -> Limit: 15 row(s)  (actual time=85.188..85.192 rows=15 loops=1)
    -> Sort: c.Title, limit input to 15 row(s) per chunk  (actual time=85.187..85.190 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=83.806..84.548 rows=2296 loops=1)
            -> Aggregate using temporary table  (actual time=83.804..83.804 rows=2296 loops=1)
                -> Nested loop inner join  (cost=14099.90 rows=14054) (actual time=0.147..67.746 rows=10915 loops=1)
                    -> Nested loop inner join  (cost=9180.91 rows=14054) (actual time=0.139..48.120 rows=10915 loops=1)
                        -> Filter: (vs.Likes > 100000)  (cost=4256.95 rows=14054) (actual time=0.115..16.484 rows=10915 loops=1)
                            -> Table scan on vs  (cost=4256.95 rows=42167) (actual time=0.112..13.149 rows=42060 loops=1)
                        -> Filter: (v.ChannelId is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=10915)
                            -> Single-row index lookup on v using PRIMARY (VideoId=vs.VideoId)  (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=10915)
                    -> Single-row index lookup on c using PRIMARY (ChannelId=v.ChannelId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.09 sec)
```

There is no overall difference in performance with the default indexing.

Indexing Design 2: Indexing on Video(Likes)
`CREATE INDEX idx_l on Video (Likes);`

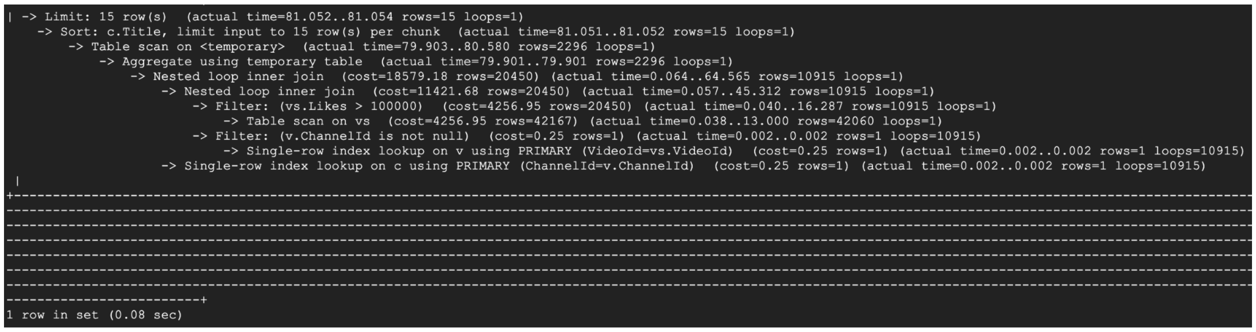


```
| -> Limit: 15 row(s)  (actual time=86.076..86.079 rows=15 loops=1)
    -> Sort: c.Title, limit input to 15 row(s) per chunk  (actual time=86.075..86.077 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=84.855..85.580 rows=2296 loops=1)
            -> Aggregate using temporary table  (actual time=84.853..84.853 rows=2296 loops=1)
                -> Nested loop inner join  (cost=18579.18 rows=20450) (actual time=0.063..67.899 rows=10915 loops=1)
                    -> Nested loop inner join  (cost=11421.68 rows=20450) (actual time=0.057..47.532 rows=10915 loops=1)
                        -> Filter: (vs.Likes > 100000)  (cost=4256.95 rows=20450) (actual time=0.039..16.801 rows=10915 loops=1)
                            -> Table scan on vs  (cost=4256.95 rows=42167) (actual time=0.037..13.566 rows=42060 loops=1)
                        -> Filter: (v.ChannelId is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=10915)
                            -> Single-row index lookup on v using PRIMARY (VideoId=vs.VideoId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
                    -> Single-row index lookup on c using PRIMARY (ChannelId=v.ChannelId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.09 sec)
```

There is no overall difference in performance with the default indexing.

Indexing Design 3: Indexing on Channel(Title) and Video(Likes)

```
CREATE INDEX idx_c on Channel (Title);
CREATE INDEX idx_l on Video (Likes);
```



```
| -> Limit: 15 row(s)  (actual time=81.052..81.054 rows=15 loops=1)
    -> Sort: c.Title, limit input to 15 row(s) per chunk  (actual time=81.051..81.052 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=79.903..80.580 rows=2296 loops=1)
            -> Aggregate using temporary table  (actual time=79.901..79.901 rows=2296 loops=1)
                -> Nested loop inner join  (cost=18579.18 rows=20450) (actual time=0.064..64.565 rows=10915 loops=1)
                    -> Nested loop inner join  (cost=11421.68 rows=20450) (actual time=0.057..45.312 rows=10915 loops=1)
                        -> Filter: (vs.Likes > 100000)  (cost=4256.95 rows=20450) (actual time=0.040..16.287 rows=10915 loops=1)
                            -> Table scan on vs  (cost=4256.95 rows=42167) (actual time=0.038..13.000 rows=42060 loops=1)
                        -> Filter: (v.ChannelId is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
                            -> Single-row index lookup on v using PRIMARY (VideoId=vs.VideoId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
                    -> Single-row index lookup on c using PRIMARY (ChannelId=v.ChannelId)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10915)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.08 sec)
```

There is no overall difference in performance with the default indexing.