Stage 3 Indexing Design Changes

Advanced Query 1:
SELECT *
FROM Video v JOIN Category c ON (c.CategoryId = v.CategoryId) JOIN Channel ch ON (ch.ChannelId = v.ChannelId)
WHERE (SELECT TotalViews FROM Channel c2 WHERE (c2.ChannelId = ch.ChannelId)) > 100000 LIMIT 15;

This query selects the top videos per category which have all more than 100,000 in views.

Performance of original advanced query 1:

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=3098.65 rows=15) (actual time=0.226..0.378 rows=15 loops=1)
   -> Nested loop inner join  (cost=3098.65 rows=3843) (actual time=0.207..0.358 rows=15 loops=1)
      -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.161..0.188 rows=19 loops=1)
         -> Table scan on v  (cost=408.55 rows=3843) (actual time=0.090..0.095 rows=19 loops=1)
```

Indexing design for advanced query 1:

1. vid_title_idx on Video(VideoTitle)

```
21 •  CREATE INDEX vid_title_idx on Video(VideoTitle);
22 •  EXPLAIN ANALYZE
23    SELECT *
24    FROM Video v JOIN Category c ON (c.CategoryId = v.CategoryId) JOIN Channel ch ON (ch.ChannelId = v.ChannelId)
25    WHERE (SELECT TotalViews FROM Channel c2 WHERE (c2.ChannelId = ch.ChannelId)) > 100000 LIMIT 15;
26
27
```

100% 49:21

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=3098.65 rows=15) (actual time=0.820..1.004 rows=15 loops=1)
   -> Nested loop inner join  (cost=3098.65 rows=3843) (actual time=0.819..1.002 rows=15 loops=1)
      -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.754..0.790 rows=19 loops=1)
         -> Table scan on v  (cost=408.55 rows=3843) (actual time=0.708..0.718 rows=19 loops=1)
```

2. chan_title_idx on Channel(ChannelTitle);

```
24 •  CREATE INDEX chan_title_idx on Channel(ChannelTitle);
25 •  EXPLAIN ANALYZE
26    SELECT *
27    FROM Video v JOIN Category c ON (c.CategoryId = v.CategoryId) JOIN Channel ch ON (ch.ChannelId = v.ChannelId)
28    WHERE (SELECT TotalViews FROM Channel c2 WHERE (c2.ChannelId = ch.ChannelId)) > 100000 LIMIT 15;
29
30
31
```

100% 28:18

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=3098.65 rows=15) (actual time=0.121..0.271 rows=15 loops=1)
  -> Nested loop inner join  (cost=3098.65 rows=3843) (actual time=0.121..0.269 rows=15 loops=1)
    -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.070..0.098 rows=19 loops=1)
      -> Table scan on v  (cost=408.55 rows=3843) (actual time=0.056..0.061 rows=19 loops=1)
```

3. vid_title_idx on Video(VideoTitle) & chan_title_idx on Channel(ChannelTitle);

```
24 •  CREATE INDEX chan_title_idx on Channel(ChannelTitle);
25 •  CREATE INDEX vid_title_idx on Video(VideoTitle);
26 •  EXPLAIN ANALYZE
27    SELECT *
28    FROM Video v JOIN Category c ON (c.CategoryId = v.CategoryId) JOIN Channel ch ON (ch.ChannelId = v.ChannelId)
29    WHERE (SELECT TotalViews FROM Channel c2 WHERE (c2.ChannelId = ch.ChannelId)) > 100000 LIMIT 15;
30
31
```

100% 19:17

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=3098.65 rows=15) (actual time=0.103..0.273 rows=15 loops=1)
  -> Nested loop inner join  (cost=3098.65 rows=3843) (actual time=0.102..0.271 rows=15 loops=1)
    -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.067..0.094 rows=19 loops=1)
      -> Table scan on v  (cost=408.55 rows=3843) (actual time=0.052..0.057 rows=19 loops=1)
```

Indexing Design Analysis:

For advanced query 1, creating indexes on Video(VideoTitle) and Channel(ChannelTitle) performed best based on the results from the "EXPLAIN ANALYZE" commands. This design choice is primarily focused on optimizing query performance and efficiency. Here are some reasons why we chose this design and why we think it works well:

1. Frequent Searches: Since video and channel titles are likely to be searched frequently by users, having indexes on VideoTitle and ChannelTitle would significantly speed up these search queries. Indexes allow the database to quickly look up rows that match the search criteria, resulting in faster response times.
2. Read-heavy workload: In a scenario where the majority of database operations are read operations such as searching and filtering, having indexes on frequently accessed columns, such as VideoTitle and ChannelTitle, could significantly improve the performance of these operations. Indexes are particularly beneficial in read-heavy workloads as they reduce the time spent scanning full tables to find matching rows.
3. Selective Columns: Indexing columns with high selectivity, like VideoTitle and ChannelTitle, can be very efficient since these columns contain unique or almost unique values. This ensures that

the index lookup can quickly narrow down the search results, reducing the number of rows to be examined.

In summary, our choice to create indexes on Video(VideoTitle) and Channel(ChannelTitle) is based on the results from the "EXPLAIN ANALYZE" commands, which indicate that this design would yield the best query performance. By optimizing for frequent searches and read-heavy workloads, our design ensures efficient access to the data stored in the Video and Channel tables.

Advanced query 2:

```
SELECT Channel.ChannelTitle, COUNT(*) AS NumOfVideos, SUM(TotalViews) AS
TotalViews
FROM Channel INNER JOIN Video ON Channel.ChannelId = Video.ChannelId
GROUP BY Channel.ChannelTitle
ORDER BY NumOfVideos DESC;
```

This query joins the Channel and Video tables based on the ChannelId column, groups the result by ChannelTitle and calculates the count of videos and the total views for each channel, and then orders the result by the number of videos in descending order.

Performance of original advanced query 2:

EXPLAIN:

```
-> Sort: NumOfVideos DESC  (actual time=21.270..21.376 rows=1239 loops=1)
   -> Table scan on <temporary>  (actual time=0.002..0.175 rows=1239 loops=1)
      -> Aggregate using temporary table  (actual time=20.250..20.495 rows=1239 loops=1)
         -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.090..14.790 rows=3173 loops=1)
```

Indexing design for advanced query 2:

1. num_views_idx on Channel(TotalViews)

```
23    CREATE INDEX num_views_idx on Channel(TotalViews);
24 •  EXPLAIN ANALYZE
25    SELECT Channel.ChannelTitle, COUNT(*) AS NumOfVideos, SUM(TotalViews) AS TotalViews
26    FROM Channel INNER JOIN Video ON Channel.ChannelId = Video.ChannelId
27    GROUP BY Channel.ChannelTitle
28    ORDER BY NumOfVideos DESC;
29
30
31
```

100% 15:18

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Sort: NumOfVideos DESC  (actual time=19.074..19.168 rows=1239 loops=1)
   -> Table scan on <temporary>  (actual time=0.002..0.147 rows=1239 loops=1)
      -> Aggregate using temporary table  (actual time=18.491..18.710 rows=1239 loops=1)
         -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.085..14.948 rows=3173 loops=1)
```

2. vid_title_idx on Video(VideoTitle)

```
24 •  CREATE INDEX vid_title_idx on Video(VideoTitle);
25 •  EXPLAIN ANALYZE
26    SELECT Channel.ChannelTitle, COUNT(*) AS NumOfVideos, SUM(TotalViews) AS TotalViews
27    FROM Channel INNER JOIN Video ON Channel.ChannelId = Video.ChannelId
28    GROUP BY Channel.ChannelTitle
29    ORDER BY NumOfVideos DESC;
```

100% 26:18

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Sort: NumOfVideos DESC  (actual time=18.863..18.957 rows=1239 loops=1)
   -> Table scan on <temporary>  (actual time=0.002..0.138 rows=1239 loops=1)
      -> Aggregate using temporary table  (actual time=18.299..18.508 rows=1239 loops=1)
         -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.108..14.858 rows=3173 loops=1)
```

3. num_views_idx on Channel(TotalViews) & vid_title_idx on Video(VideoTitle)

```
24 •  CREATE INDEX vid_title_idx on Video(VideoTitle);
25 •  CREATE INDEX num_views on Channel(TotalViews);
26 •  EXPLAIN ANALYZE
27    SELECT Channel.ChannelTitle, COUNT(*) AS NumOfVideos, SUM(TotalViews) AS TotalViews
28    FROM Channel INNER JOIN Video ON Channel.ChannelId = Video.ChannelId
29    GROUP BY Channel.ChannelTitle
30    ORDER BY NumOfVideos DESC;
31
```

100% 36:22

Form Editor Navigate: 1 / 1

EXPLAIN:

```
-> Sort: NumOfVideos DESC  (actual time=19.184..19.278 rows=1239 loops=1)
   -> Table scan on <temporary>  (actual time=0.002..0.166 rows=1239 loops=1)
      -> Aggregate using temporary table  (actual time=18.565..18.801 rows=1239 loops=1)
         -> Nested loop inner join  (cost=1753.60 rows=3843) (actual time=0.078..15.024 rows=3173 loops=1)
```

Indexing Design Analysis:

For advanced query 2, we have chosen to create an index on Video(VideoTitle) named vid_title_idx based on the results from the "EXPLAIN ANALYZE" commands. While the advanced query does not directly utilize the VideoTitle column, the index may still offer indirect benefits in terms of performance and efficiency. Here are some reasons why we think this design choice might work well:

1. Supporting Additional Queries: Although the provided advanced query does not explicitly use VideoTitle as a filter or sorting condition, other queries in your application might require searching or filtering based on video titles. By creating an index on Video(VideoTitle), we can speed up those search queries by quickly looking up rows that match the search criteria.
2. Index Scans: The database engine may still use the vid_title_idx index to perform index scans when executing the advanced query. In some cases, an index scan can be more efficient than a full table scan as it can quickly navigate through the index structure and fetch only the relevant rows, resulting in reduced I/O operations.

3. Efficient JOIN operations: Indexes can help optimize JOIN operations by reducing the number of rows that need to be compared when matching records between tables. In our advanced query, there is a JOIN operation between the Channel and Video tables. Although the JOIN condition uses ChannelId, the database engine may still use the VideoTitle index to efficiently fetch matching rows from the Video table, depending on the database's query optimizer.

In summary, our choice to create an index on Video(VideoTitle) named vid_title_idx is based on the results from the "EXPLAIN ANALYZE" commands, which indicate that this design yields the best query performance. Although the advanced query does not directly use the VideoTitle column, the index can still offer indirect performance benefits by supporting additional queries, enabling efficient index scans, and optimizing JOIN operations.