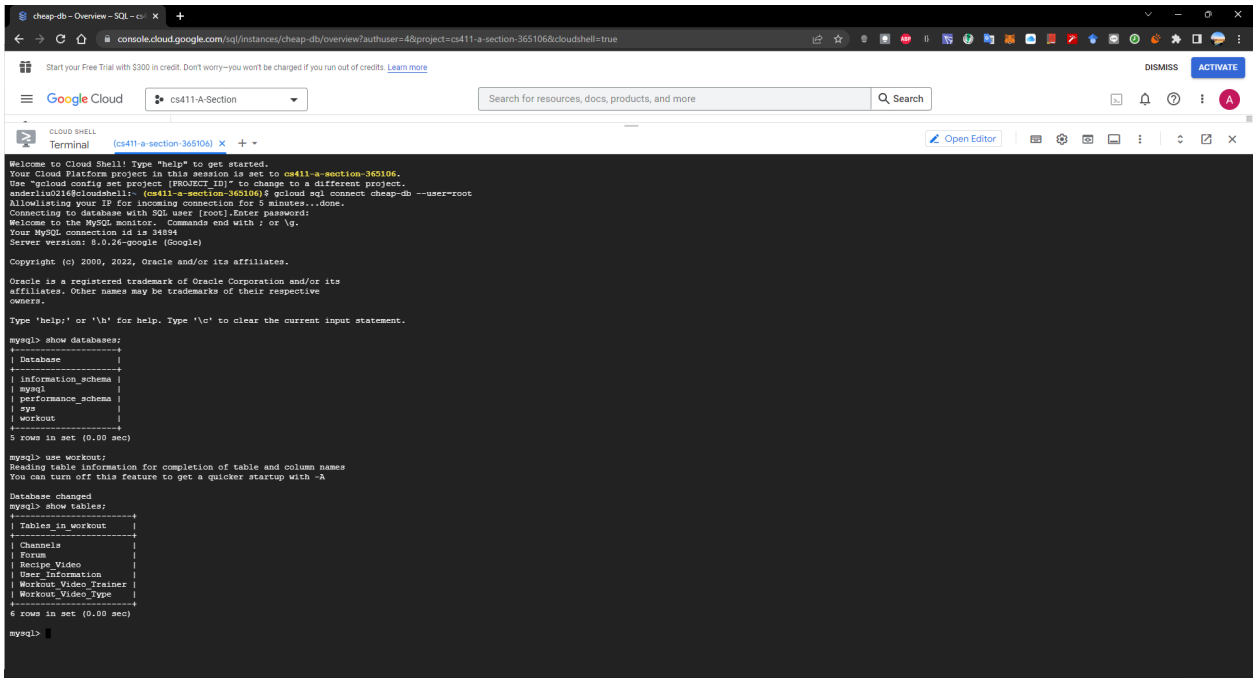
Project Stage 3

1. **Database implementation:**
   a. **MySQL@GCP**

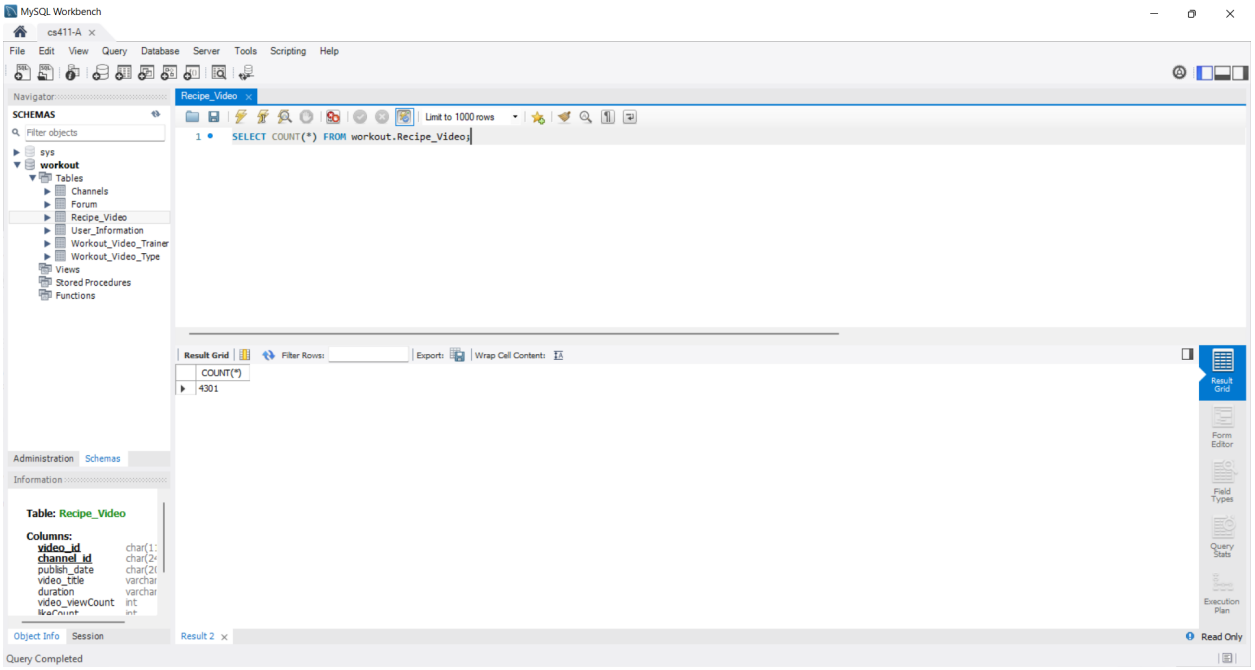


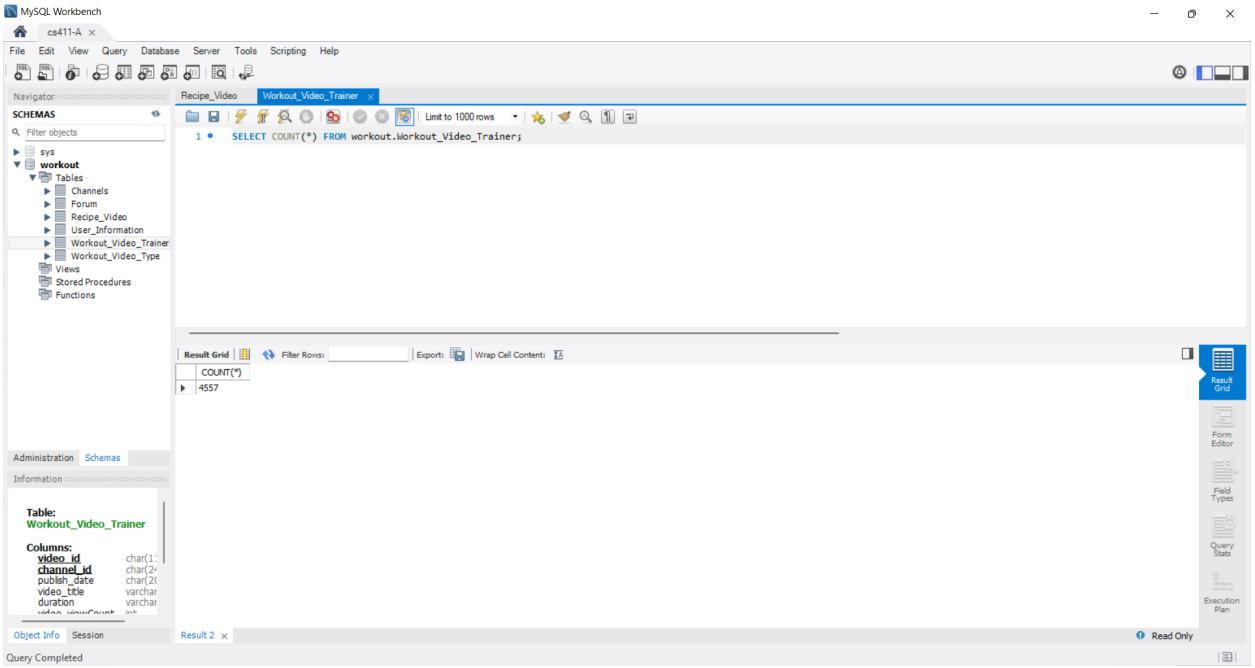
   b. **Data Definition Language (DDL)**

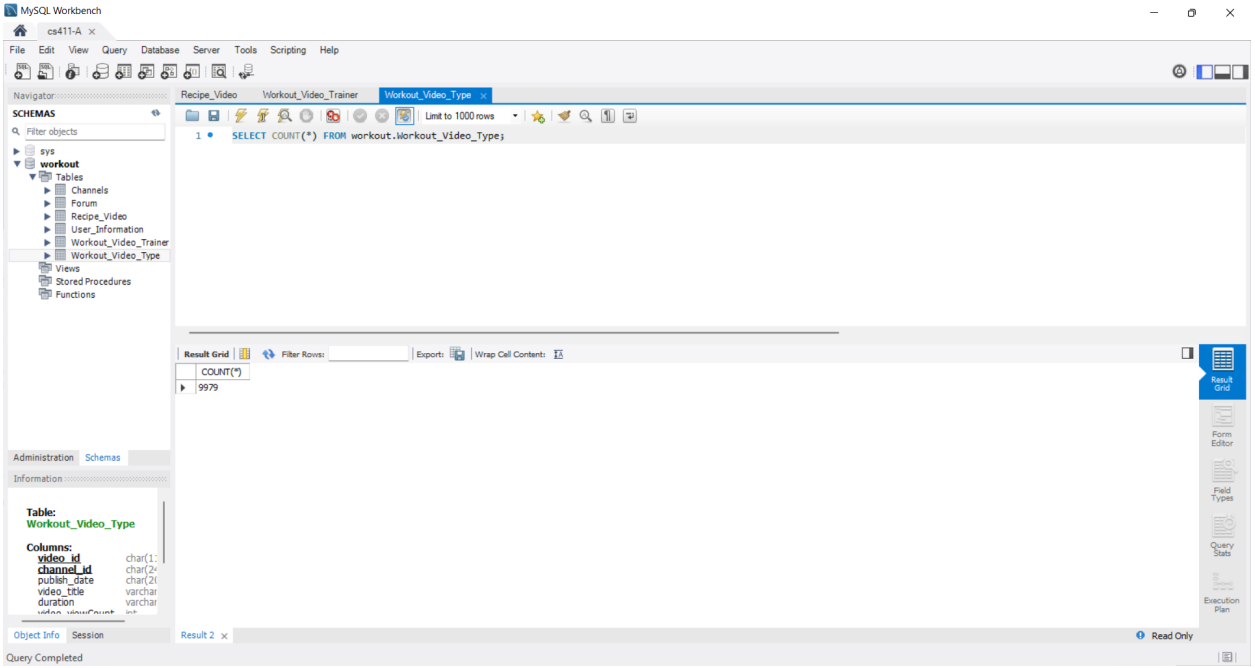
```
CREATE TABLE Channels(
        channel_id CHAR(24) NOT NULL,
        channel_title VARCHAR(255) NOT NULL,
        subscriberCount INT NOT NULL,
        channel_viewCount BIGINT NOT NULL,
        videoCount INT NOT NULL,
        PRIMARY KEY (channel_id));

CREATE TABLE Workout_Video_Trainer(
        video_id CHAR(11) NOT NULL,
        channel_id CHAR(24) NOT NULL,
        publish_date CHAR(20) NOT NULL,
        video_title VARCHAR(255) NOT NULL,
        duration VARCHAR(255) NOT NULL,
        video_viewCount INT NOT NULL,
        likeCount INT NOT NULL,
        trainer VARCHAR(255),
        PRIMARY KEY (video_id, channel_id),
```

```
	FOREIGN KEY(channel_id) REFERENCES Channels(channel_id) ON DELETE
CASCADE);

CREATE TABLE Workout_Video_Type(
	video_id CHAR(11) NOT NULL,
	channel_id CHAR(24) NOT NULL,
	publish_date CHAR(20) NOT NULL,
	video_title VARCHAR(255) NOT NULL,
	duration VARCHAR(255) NOT NULL,
	video_viewCount INT NOT NULL,
	likeCount INT NOT NULL,
	workout_type VARCHAR(255) NOT NULL,
	PRIMARY KEY (video_id, channel_id),
	FOREIGN KEY (channel_id) REFERENCES Channels(channel_id) ON DELETE
CASCADE);

CREATE TABLE Recipe_Video(
	video_id CHAR(11) NOT NULL,
	channel_id CHAR(24) NOT NULL,
	publish_date CHAR(20) NOT NULL,
	video_title VARCHAR(255) NOT NULL,
	duration VARCHAR(255) NOT NULL,
	video_viewCount INT NOT NULL,
	likeCount INT NOT NULL,
	PRIMARY KEY (video_id, channel_id),
	FOREIGN KEY(channel_id) REFERENCES Channels(channel_id) ON DELETE
CASCADE);

CREATE TABLE User_Information(
	user_id INT NOT NULL,
	email VARCHAR(255) NOT NULL,
	phone_number VARCHAR(10) NOT NULL,
	password VARCHAR(255) NOT NULL,
	PRIMARY KEY (user_id));

CREATE TABLE Forum(
	post_id INT NOT NULL,
	user_id INT NOT NULL,
	post_timestamp VARCHAR(255) NOT NULL,
	post_title VARCHAR(255) NOT NULL,
	post_content VARCHAR(5000) NOT NULL,
	PRIMARY KEY (post_id , user_id),
	FOREIGN KEY(user_id) REFERENCES User_Information(user_id) ON DELETE
CASCADE);
```

c. Inserting at least 1000 rows in the tables







2. **Advanced Queries**
   a. **If a user is interested in HIIT&crossfit and wants to know the number of million views video of each channel. For trainer videos, they want to filter the video publish after 2020 (inclusive).**

```
(SELECT channel_title, COUNT(video_id) AS cnt
FROM workout.Workout_Video_Trainer NATURAL JOIN workout.Channels
WHERE video_viewCount > 1000000 AND publish_date LIKE "202%"
```

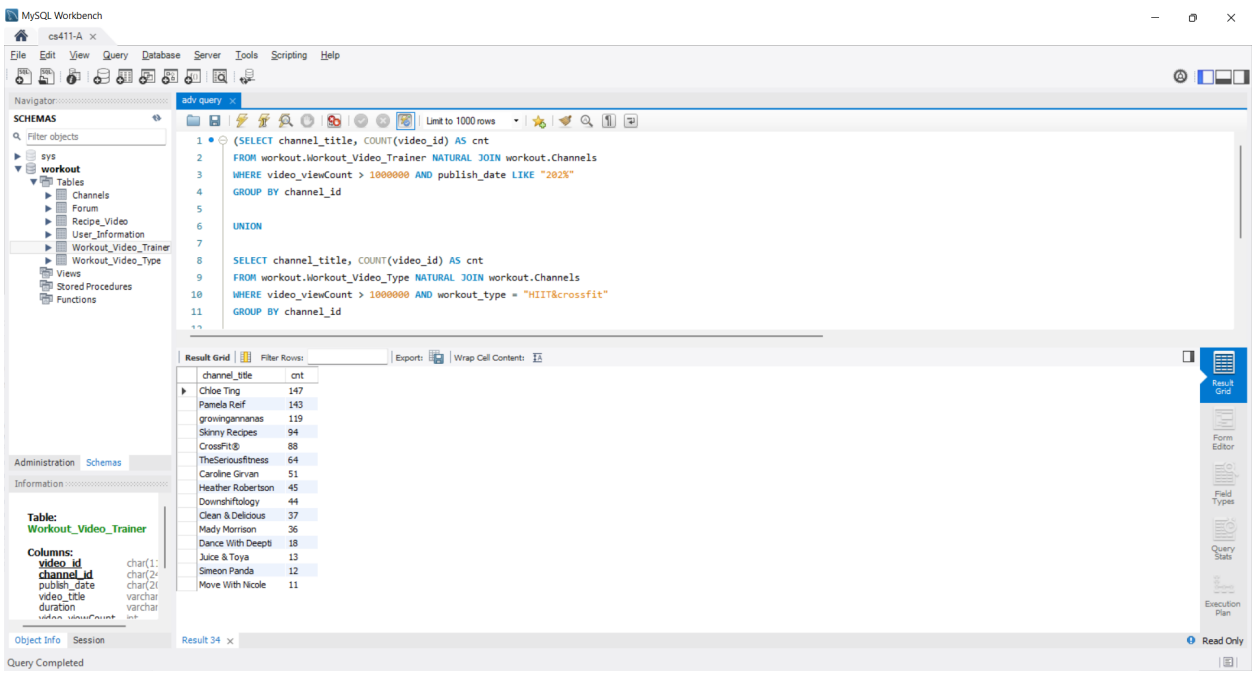
GROUP BY channel_id

UNION

SELECT channel_title, COUNT(video_id) AS cnt
FROM workout.Workout_Video_Type NATURAL JOIN workout.Channels
WHERE video_viewCount > 1000000 AND workout_type = "HIIT&crossfit"
GROUP BY channel_id

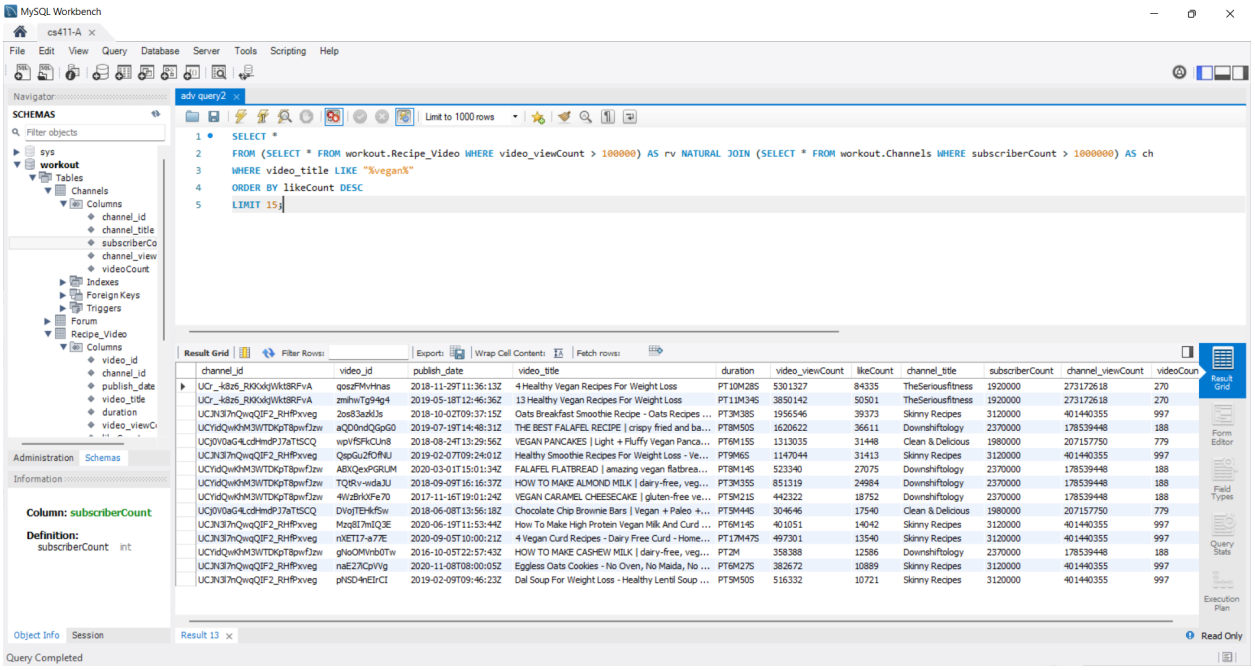
UNION

SELECT channel_title, COUNT(video_id) AS cnt
FROM workout.Recipe_Video NATURAL JOIN workout.Channels
WHERE video_viewCount > 1000000
GROUP BY channel_id)
ORDER BY cnt DESC
LIMIT 15;



b. Suppose vegan people want to know the recipe videos which have at least 100000 views and the channel has millions of subscribers. We want to do subquery first and then join, to avoid the expensive cost of joins.

SELECT *
FROM (SELECT * FROM workout.Recipe_Video WHERE video_viewCount > 100000) AS rv
NATURAL JOIN (SELECT * FROM workout.Channels WHERE subscriberCount > 1000000) AS ch
WHERE video_title LIKE "%vegan%";



3. Indexing Analysis

a. Query1

-> Limit: 15 row(s) (cost=2.50 rows=0) (actual time=0.029..0.031 rows=15 loops=1)
-> Sort: cnt DESC, limit input to 15 row(s) per chunk (cost=2.50 rows=0) (actual time=0.028..0.029 rows=15 loops=1)
-> Table scan on <union temporary> (cost=2.50 rows=0) (actual time=0.001..0.005 rows=21 loops=1)
-> Union materialize with deduplication (cost=2.50..2.50 rows=0) (actual time=11.993..11.996 rows=21 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.003 rows=10 loops=1)
-> Aggregate using temporary table (actual time=3.256..3.258 rows=10 loops=1)
-> Nested loop inner join (cost=562.96 rows=177) (actual time=0.109..2.933 rows=472 loops=1)
-> Filter: ((Workout_Video_Trainer.video_viewCount > 1000000) and (Workout_Video_Trainer.publish_date like '202%')) (cost=501.15 rows=177) (actual time=0.089..2.337 rows=472 loops=1)
-> Table scan on Workout_Video_Trainer (cost=501.15 rows=4769) (actual time=0.075..1.875 rows=4557 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Trainer.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=472)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=3 loops=1)
-> Aggregate using temporary table (actual time=6.588..6.589 rows=3 loops=1)
-> Nested loop inner join (cost=1129.66 rows=325) (actual time=0.066..6.422 rows=220 loops=1)

-> Filter: ((Workout_Video_Type.workout_type = 'HIIT&crossfit') and (Workout_Video_Type.video_viewCount > 1000000)) (cost=1015.85 rows=325) (actual time=0.057..6.209 rows=220 loops=1)
-> Table scan on Workout_Video_Type (cost=1015.85 rows=9756) (actual time=0.050..3.640 rows=9979 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Type.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=220)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=8 loops=1)
-> Aggregate using temporary table (actual time=2.089..2.090 rows=8 loops=1)
-> Nested loop inner join (cost=1039.50 rows=1562) (actual time=0.058..1.918 rows=248 loops=1)
-> Filter: (Recipe_Video.video_viewCount > 1000000) (cost=492.85 rows=1562) (actual time=0.052..1.639 rows=248 loops=1)
-> Table scan on Recipe_Video (cost=492.85 rows=4686) (actual time=0.048..1.396 rows=4301 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Recipe_Video.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=248)

CREATE INDEX video_viewCount_idx ON workout.Workout_Video_Trainer(video_viewCount);
CREATE INDEX video_viewCount_idx ON workout.Workout_Video_Type(video_viewCount);
CREATE INDEX video_viewCount_idx ON workout.Recipe_Video(video_viewCount);

-> Limit: 15 row(s) (cost=2.50 rows=0) (actual time=0.022..0.024 rows=15 loops=1)
-> Sort: cnt DESC, limit input to 15 row(s) per chunk (cost=2.50 rows=0) (actual time=0.022..0.023 rows=15 loops=1)
-> Table scan on <union temporary> (cost=2.50 rows=0) (actual time=0.001..0.003 rows=21 loops=1)
-> Union materialize with deduplication (cost=2.50..2.50 rows=0) (actual time=18.440..18.443 rows=21 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.004 rows=10 loops=1)
-> Aggregate using temporary table (actual time=9.019..9.023 rows=10 loops=1)
-> Nested loop inner join (cost=374.75 rows=85) (actual time=0.399..8.649 rows=472 loops=1)
-> Filter: (Workout_Video_Trainer.publish_date like '202%') (cost=344.96 rows=85) (actual time=0.385..8.042 rows=472 loops=1)
-> Index range scan on Workout_Video_Trainer using video_viewCount_idx, with index condition: (Workout_Video_Trainer.video_viewCount > 1000000) (cost=344.96 rows=766) (actual time=0.381..7.896 rows=766 loops=1)

-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Trainer.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=472)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=3 loops=1)
-> Aggregate using temporary table (actual time=8.644..8.645 rows=3 loops=1)
-> Nested loop inner join (cost=364.01 rows=75) (actual time=0.284..8.418 rows=220 loops=1)
-> Filter: (Workout_Video_Type.workout_type = 'HIIT&crossfit') (cost=337.76 rows=75) (actual time=0.278..8.140 rows=220 loops=1)
-> Index range scan on Workout_Video_Type using video_viewCount_idx, with index condition: (Workout_Video_Type.video_viewCount > 1000000) (cost=337.76 rows=750) (actual time=0.275..7.984 rows=750 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Type.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=220)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=8 loops=1)
-> Aggregate using temporary table (actual time=0.715..0.717 rows=8 loops=1)
-> Nested loop inner join (cost=141.44 rows=248) (actual time=0.050..0.492 rows=248 loops=1)
-> Filter: (Recipe_Video.video_viewCount > 1000000) (cost=54.64 rows=248) (actual time=0.043..0.164 rows=248 loops=1)
-> Index range scan on Recipe_Video using video_viewCount_idx (cost=54.64 rows=248) (actual time=0.041..0.137 rows=248 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Recipe_Video.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=248)

CREATE INDEX publish_date_idx ON workout.Workout_Video_Trainer(publish_date);

-> Limit: 15 row(s) (cost=2.50 rows=0) (actual time=0.037..0.039 rows=15 loops=1)
-> Sort: cnt DESC, limit input to 15 row(s) per chunk (cost=2.50 rows=0) (actual time=0.028..0.029 rows=15 loops=1)
-> Table scan on <union temporary> (cost=2.50 rows=0) (actual time=0.002..0.006 rows=21 loops=1)
-> Union materialize with deduplication (cost=2.50..2.50 rows=0) (actual time=12.368..12.370 rows=21 loops=1)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=10 loops=1)
-> Aggregate using temporary table (actual time=3.120..3.121 rows=10 loops=1)
-> Nested loop inner join (cost=779.26 rows=795) (actual time=0.090..2.795 rows=472 loops=1)
-> Filter: ((Workout_Video_Trainer.video_viewCount > 1000000) and (Workout_Video_Trainer.publish_date like '202%')) (cost=501.15 rows=795) (actual time=0.075..2.214 rows=472 loops=1)

-> Table scan on Workout_Video_Trainer (cost=501.15 rows=4769) (actual time=0.063..1.776 rows=4557 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Trainer.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=472)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=3 loops=1)
-> Aggregate using temporary table (actual time=7.077..7.077 rows=3 loops=1)
-> Nested loop inner join (cost=1129.66 rows=325) (actual time=0.049..6.885 rows=220 loops=1)
-> Filter: ((Workout_Video_Type.workout_type = 'HIIT&crossfit') and (Workout_Video_Type.video_viewCount > 1000000)) (cost=1015.85 rows=325) (actual time=0.044..6.654 rows=220 loops=1)
-> Table scan on Workout_Video_Type (cost=1015.85 rows=9756) (actual time=0.037..3.813 rows=9979 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Type.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=220)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=8 loops=1)
-> Aggregate using temporary table (actual time=2.104..2.105 rows=8 loops=1)
-> Nested loop inner join (cost=1039.50 rows=1562) (actual time=0.062..1.926 rows=248 loops=1)
-> Filter: (Recipe_Video.video_viewCount > 1000000) (cost=492.85 rows=1562) (actual time=0.053..1.637 rows=248 loops=1)
-> Table scan on Recipe_Video (cost=492.85 rows=4686) (actual time=0.050..1.381 rows=4301 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Recipe_Video.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=248)

CREATE INDEX workout_type_idx ON workout.Workout_Video_Type(workout_type);

-> Limit: 15 row(s) (cost=2.50 rows=0) (actual time=0.023..0.025 rows=15 loops=1)
-> Sort: cnt DESC, limit input to 15 row(s) per chunk (cost=2.50 rows=0) (actual time=0.023..0.023 rows=15 loops=1)
-> Table scan on <union temporary> (cost=2.50 rows=0) (actual time=0.001..0.004 rows=21 loops=1)
-> Union materialize with deduplication (cost=2.50..2.50 rows=0) (actual time=24.575..24.578 rows=21 loops=1)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=10 loops=1)
-> Aggregate using temporary table (actual time=3.177..3.179 rows=10 loops=1)
-> Nested loop inner join (cost=562.96 rows=177) (actual time=0.086..2.832 rows=472 loops=1)

-> Filter: ((Workout_Video_Trainer.video_viewCount > 1000000) and (Workout_Video_Trainer.publish_date like '202%')) (cost=501.15 rows=177) (actual time=0.071..2.257 rows=472 loops=1)
-> Table scan on Workout_Video_Trainer (cost=501.15 rows=4769) (actual time=0.058..1.800 rows=4557 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Trainer.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=472)
-> Table scan on <temporary> (actual time=0.002..0.003 rows=3 loops=1)
-> Aggregate using temporary table (actual time=18.684..18.684 rows=3 loops=1)
-> Nested loop inner join (cost=852.38 rows=1626) (actual time=0.156..18.463 rows=220 loops=1)
-> Filter: (Workout_Video_Type.video_viewCount > 1000000) (cost=283.33 rows=1626) (actual time=0.150..18.183 rows=220 loops=1)
-> Index lookup on Workout_Video_Type using workout_type_idx (workout_type='HIIT&crossfit') (cost=283.33 rows=4878) (actual time=0.146..17.748 rows=7416 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Workout_Video_Type.channel_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=220)
-> Table scan on <temporary> (actual time=0.001..0.002 rows=8 loops=1)
-> Aggregate using temporary table (actual time=2.653..2.655 rows=8 loops=1)
-> Nested loop inner join (cost=1039.50 rows=1562) (actual time=0.055..2.442 rows=248 loops=1)
-> Filter: (Recipe_Video.video_viewCount > 1000000) (cost=492.85 rows=1562) (actual time=0.048..1.928 rows=248 loops=1)
-> Table scan on Recipe_Video (cost=492.85 rows=4686) (actual time=0.044..1.653 rows=4301 loops=1)
-> Single-row index lookup on Channels using PRIMARY (channel_id=Recipe_Video.channel_id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=248)

Filter: ((Workout_Video_Trainer.video_viewCount > 1000000) and (Workout_Video_Trainer.publish_date like '202%')) (cost=501.15 rows=177) (actual time=0.089..2.337 rows=472 loops=1)
Filter: ((Workout_Video_Type.workout_type = 'HIIT&crossfit') and (Workout_Video_Type.video_viewCount > 1000000)) (cost=1015.85 rows=325) (actual time=0.057..6.209 rows=220 loops=1)
Filter: (Recipe_Video.video_viewCount > 1000000) (cost=492.85 rows=1562) (actual time=0.052..1.639 rows=248 loops=1)

Before adding the index on video_viewCount, the cost of each subquery is 501.15, 1015.85, and 492.85 separately. After adding the index on video_viewCount, the cost reduces to

344.96, 337.76, and 54.64, which improves the performance by over 60%. Adding the index on publish_date doesn't affect the analyzed result because indexing cannot be used on forms of LIKE "abc%". On the other hand, adding the index on workout_type is very successful, reducing cost from 1015.85 to 283.33, decreasing over 70% of the cost. This is because we only have four types of workout_type, so indexing on it would be very useful.

b. Query2

```
-> Limit: 15 row(s)  (cost=1004.83 rows=15) (actual time=5.650..5.684 rows=15 loops=1)
    -> Nested loop inner join  (cost=1004.83 rows=1562) (actual time=5.649..5.682 rows=15
loops=1)
        -> Sort: Recipe_Video.likeCount DESC  (cost=492.85 rows=4686) (actual
time=5.614..5.617 rows=19 loops=1)
            -> Filter: ((Recipe_Video.video_title like '%vegan%') and
(Recipe_Video.video_viewCount > 100000))  (cost=492.85 rows=4686) (actual
time=0.197..5.556 rows=98 loops=1)
                -> Table scan on Recipe_Video  (cost=492.85 rows=4686) (actual time=0.086..2.080
rows=4301 loops=1)
        -> Filter: (Channels.subscriberCount > 1000000)  (cost=0.25 rows=0) (actual
time=0.003..0.003 rows=1 loops=19)
            -> Single-row index lookup on Channels using PRIMARY
(channel_id=Recipe_Video.channel_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=19)
```

CREATE INDEX video_viewCount_idx ON workout.Recipe_Video(video_viewCount);

```
-> Limit: 15 row(s)  (cost=1001.50 rows=15) (actual time=5.610..5.645 rows=15 loops=1)
    -> Nested loop inner join  (cost=1001.50 rows=1562) (actual time=5.609..5.643 rows=15
loops=1)
        -> Sort: Recipe_Video.likeCount DESC  (cost=492.85 rows=4686) (actual
time=5.576..5.579 rows=19 loops=1)
            -> Filter: ((Recipe_Video.video_title like '%vegan%') and
(Recipe_Video.video_viewCount > 100000))  (cost=492.85 rows=4686) (actual
time=0.225..5.518 rows=98 loops=1)
                -> Table scan on Recipe_Video  (cost=492.85 rows=4686) (actual time=0.072..2.057
rows=4301 loops=1)
        -> Filter: (Channels.subscriberCount > 1000000)  (cost=0.25 rows=0) (actual
time=0.003..0.003 rows=1 loops=19)
            -> Single-row index lookup on Channels using PRIMARY
(channel_id=Recipe_Video.channel_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=19)
```

CREATE INDEX subscriberCount_idx ON workout.Channels(subscriberCount);

```
-> Limit: 15 row(s)  (cost=1004.83 rows=15) (actual time=5.657..5.691 rows=15 loops=1)
    -> Nested loop inner join  (cost=1004.83 rows=2570) (actual time=5.656..5.689 rows=15
loops=1)
        -> Sort: Recipe_Video.likeCount DESC  (cost=492.85 rows=4686) (actual
time=5.626..5.629 rows=19 loops=1)
            -> Filter: ((Recipe_Video.video_title like '%vegan%') and
(Recipe_Video.video_viewCount > 100000))  (cost=492.85 rows=4686) (actual
time=0.185..5.568 rows=98 loops=1)
                -> Table scan on Recipe_Video  (cost=492.85 rows=4686) (actual time=0.066..2.076
rows=4301 loops=1)
        -> Filter: (Channels.subscriberCount > 1000000)  (cost=0.25 rows=1) (actual
time=0.003..0.003 rows=1 loops=19)
            -> Single-row index lookup on Channels using PRIMARY
(channel_id=Recipe_Video.channel_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=19)
```

CREATE INDEX video_title_idx ON workout.Recipe_Video(video_title);

```
-> Limit: 15 row(s)  (cost=1004.83 rows=15) (actual time=6.060..6.096 rows=15 loops=1)
    -> Nested loop inner join  (cost=1004.83 rows=1562) (actual time=6.059..6.094 rows=15
loops=1)
        -> Sort: Recipe_Video.likeCount DESC  (cost=492.85 rows=4686) (actual
time=6.019..6.022 rows=19 loops=1)
            -> Filter: ((Recipe_Video.video_title like '%vegan%') and
(Recipe_Video.video_viewCount > 100000))  (cost=492.85 rows=4686) (actual
time=0.185..5.953 rows=98 loops=1)
                -> Table scan on Recipe_Video  (cost=492.85 rows=4686) (actual time=0.085..2.207
rows=4301 loops=1)
        -> Filter: (Channels.subscriberCount > 1000000)  (cost=0.25 rows=0) (actual
time=0.003..0.004 rows=1 loops=19)
            -> Single-row index lookup on Channels using PRIMARY
(channel_id=Recipe_Video.channel_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1
loops=19)
```

Because we use LIKE “%abc%” in the where clause, any indexing on Recipe_Video would be useless. Moreover, every subscriberCount only has three significant digits, which causes them usually end with zeros, so indexing on subscriberCount is useless as well. In conclusion, three kinds of indexing do not affect the analyzed result, which means they do not work at all.