Milestone 4 - Ryan Inghilterra - Cats

Precomputed tables

```
CREATE TABLE user_video AS
   SELECT
      v.video_id,
      u.user_id
   FROM video v CROSS JOIN users u;
```

the cost of maintenance for this table would that it need to be updated any time there is a new user or video added, removed, or updated.

New Query 5 with the precomputed table: user_video

```
PREPARE Q5 (int) AS

   WITH liked AS (
   SELECT u.video_id, u.user_id, CAST(COALESCE(l.like_id,0) AS boolean) AS liked
   FROM user_video u LEFT OUTER JOIN likes l
   ON u.video_id = l.video_id
   AND u.user_id = l.user_id
   WHERE u.user_id != $1
   ),
   iliked AS (
   SELECT u.video_id, u.user_id, CAST(COALESCE(l.like_id,0) AS boolean) AS iliked
   FROM user_video u LEFT OUTER JOIN likes l
   ON u.video_id = l.video_id
   AND u.user_id = l.user_id
   WHERE u.user_id = $1
   ),
   t1 AS (
   SELECT $1 AS the_user, u.user_id AS other_user
   from users u
   WHERE u.user_id != $1
   ),
   t2 AS (
   SELECT v.video_id, t1.the_user, t1.other_user
   FROM video v CROSS JOIN t1
   ),
   t3 AS (
   SELECT t2.video_id, t2.other_user, l.liked
   FROM t2 FULL OUTER JOIN liked l
   ON t2.video_id = l.video_id
   AND t2.other_user = l.user_id
   ),
```

```
 t4 AS (
   SELECT t3.video_id, t3.other_user, t3.liked, l.iliked, (t3.liked AND l.iliked) AS
liked_by_both
   FROM t3 FULL OUTER JOIN iliked l
   ON t3.video_id = l.video_id
   ),
   t5 AS (
   SELECT other_user, log(1 + SUM(CAST(liked_by_both as int))) AS lc_score
   FROM t4
   GROUP BY other_user
   ),
   t6 AS (
   SELECT t3.video_id, t3.liked, SUM(t5.lc_score) AS rank
   FROM t3 FULL OUTER JOIN t5
   ON t3.other_user = t5.other_user
   GROUP BY t3.video_id, t3.liked
   ),
   t7 AS (
   SELECT video_id, rank
   FROM t6
   WHERE liked = true
   ),
   t8 AS (
   SELECT v.video_id, COALESCE(rank, 0) AS rank
   FROM video v LEFT OUTER JOIN t7
   ON v.video_id = t7.video_id
   ),
   iwatched AS (
   SELECT u.video_id, u.user_id, CAST(COALESCE(w.watch_id,0) AS boolean) AS
iwatched
   FROM user_video u LEFT OUTER JOIN watch w
   ON u.video_id = w.video_id
   AND u.user_id = w.user_id
   WHERE u.user_id = $1
   ),
   t9 AS (
   SELECT t8.video_id, t8.rank, l.iliked, w.iwatched
   FROM t8 LEFT OUTER JOIN iliked l
   ON t8.video_id = l.video_id
   LEFT OUTER JOIN iwatched w
   ON t8.video_id = w.video_id
   )
   SELECT video_id as vid, rank FROM t9
   ORDER BY iliked ASC, iwatched ASC, RANK DESC
   LIMIT 10;
```

updated query 5 cost
Limit (cost=10564712.67..10564712.70 rows=10 width=14) (actual time=5182.181..5182.182 rows=10 loops=1)

original query 5 cost
Limit (cost=33811219.61..33811219.64 rows=10 width=14) (actual time=6319.787..6319.789 rows=10 loops=1)

a large improvement in performance! The sequential scan of from a cross join can be expensive. For example in user_video, if there are 10,000 users and 10,000 videos, then the cross join is 10,000 x 10,000 rows which is an extremely large amount. So by moving user_video from a CTE in the original query to a precomputed table, it speeds up my query 5 significantly.

After trying some different options of other precomputed tables and trying to add some different indices, I decide the performance benefits were not significant at all and that the maintenance cost would outweigh the improvement performance.

So overall I only decided to add the user_video precomputed table, but this alone made a big performance difference