In [20]:
```python
import numpy as np
import pandas as pd
import psycopg2
import pgspecial
```

In [2]:
```
!docker ps -a
```

```
CONTAINER ID   IMAGE      COMMAND                  CREATED       STATUS
       PORTS                    NAMES
ef19897b58ef   postgres   "docker-entrypoint.s…"   13 days ago   Up 34 minutes
       0.0.0.0:5432->5432/tcp   cats
3f9812e56ca3   postgres   "docker-entrypoint.s…"   2 weeks ago   Exited (0) 35 minutes
ago                             sales
```

In [2]:
```
!docker restart ef19897b58ef
```

```
ef19897b58ef
```

In [3]:
```
# examine port connection
!nc -zv localhost 5432
```

```
found 0 associations
found 1 connections:
     1: flags=82<CONNECTED,PREFERRED>
        outif lo0
        src ::1 port 59490
        dst ::1 port 5432
        rank info not available
        TCP aux info available

Connection to localhost port 5432 [tcp/postgresql] succeeded!
```

In [7]:
```python
# connect database located on docker with psycopg2
try:
    conn = psycopg2.connect("dbname='postgres' user='postgres' host='0.0.0.0' passwor
except:
    print("I am unable to connect to the database")
```

In [9]:
```python
# Open a cursor to perform database operations
cur = conn.cursor()
```

In [3]:
```
%load_ext sql
```

In [4]:
```
# # connect database located on docker with ipython-sql
%sql postgresql://postgres:huyuan3@localhost/postgres
```

Out[4]:
```
'Connected: postgres@postgres'
```

In [11]:
```python
def query_to_df(conn, query, column_names):
    """
    Tranform a SELECT query into a pandas dataframe
    """
    cur = conn.cursor()
    try:
        cur.execute(query)
    except (Exception, psycopg2.DatabaseError) as error:
        print("Error: %s" % error)
        cur.close()
        return 1

    # Naturally we get a list of tuples
    tuples = cur.fetchall()
    cur.close()

    # We just need to turn it into a pandas dataframe
    df = pd.DataFrame(tuples, columns=column_names)
    return df
```

Table Statistics

In [21]:
```python
# load csv to pandas Dataframe
states = pd.read_csv('users.csv')
videos = pd.read_csv('videos.csv')
likes = pd.read_csv('likes.csv')
logins = pd.read_csv('logins.csv')
watches = pd.read_csv('watches.csv')
friends = pd.read_csv('friends.csv')
suggestions = pd.read_csv('suggestions.csv')
```

In [28]:
```
states
```

Out[28]:

	user_id	fname	lname	facebook_id
0	0	Joselito	SMITH	Joselito_SMITH@
1	1	Cynda	SMITH	Cynda_SMITH@
2	2	Eleftheria	SMITH	Eleftheria_SMITH@
3	3	Adley	JOHNSON	Adley_JOHNSON@
4	4	Remy	JOHNSON	Remy_JOHNSON@
...	...	...	...	...
266392	266392	Elda	AALUND	Elda_AALUND@
266393	266393	Gannon	AALUND	Gannon_AALUND@
266394	266394	Tully	AALDERINK	Tully_AALDERINK@
266395	266395	Vera	AALDERINK	Vera_AALDERINK@
266396	266396	Rodolfo	AALDERINK	Rodolfo_AALDERINK@

266397 rows × 4 columns

In [27]:
```
videos
```

Out[27]:

	video_id	video_name
0	0	video_0
1	1	video_1
2	2	video_2
3	3	video_3
4	4	video_4
...	...	...
999995	999995	video_999995
999996	999996	video_999996
999997	999997	video_999997
999998	999998	video_999998
999999	999999	video_999999

1000000 rows × 2 columns

In [26]:
```
likes
```

Out[26]:

	like_id	user_id	video_id	time
0	0	0	606605	2020-01-25
1	1	0	706726	2020-01-09
2	2	0	782062	2020-01-01
3	3	0	818210	2020-01-18
4	4	0	800779	2020-01-04
...	...	...	...	...
2528742	2528753	266396	145751	2020-01-28
2528743	2528754	266396	350362	2020-01-30
2528744	2528755	266396	484309	2020-01-19
2528745	2528756	266396	197010	2020-01-02
2528746	2528757	266396	699516	2020-01-05

2528747 rows × 4 columns

In [25]:
```
logins
```

Out[25]:

	login_id	user_id	time
0	0	199172	2020-01-04
1	1	29048	2020-01-16
2	2	3621	2020-01-26
3	3	264321	2020-01-14
4	4	14089	2020-01-26
...	...	...	...
99995	99995	47064	2020-01-11
99996	99996	58172	2020-01-01
99997	99997	46357	2020-01-04
99998	99998	18675	2020-01-17
99999	99999	30480	2020-01-12

100000 rows × 3 columns

In [24]:
```
watches
```

Out[24]:

	watch_id	user_id	video_id	time
0	0	15625	362900	2020-01-02
1	1	14038	520914	2020-01-16
2	2	152860	366736	2020-01-15
3	3	107271	565628	2020-01-01
4	4	119658	423330	2020-01-27
...	...	...	...	...
534090	534090	141584	197057	2020-01-31
534091	534091	66117	955495	2020-01-21
534092	534092	154244	223846	2020-01-12
534093	534093	150694	803963	2020-01-19
534094	534094	176630	128460	2020-01-06

534095 rows × 4 columns

In [23]:
```
friends
```

Out[23]:

	subject	object
0	0	133554
1	133554	0
2	0	143517
3	143517	0
4	0	165074
...	...	...
2403487	97980	266396
2403488	266396	33903
2403489	33903	266396
2403490	266396	143271
2403491	143271	266396

2403492 rows × 2 columns

In [22]:
```
suggestions
```

Out[22]:

	suggest_id	login_id	video_id
0	0	233384	784951
1	0	233384	455420
2	0	233384	924175
3	0	233384	894830
4	0	233384	235215
...	...	...	...
999995	99999	216301	773409
999996	99999	216301	267102
999997	99999	216301	689943
999998	99999	216301	361671
999999	99999	216301	892770

1000000 rows × 3 columns

Query 1 no indexing

In [12]:
```python
#query 1
query1 = """SELECT video_id, COUNT(*) AS num_like
            FROM cats.likes
            GROUP BY video_id
            ORDER BY num_like DESC
            LIMIT 10"""
result1 = query_to_df(conn, query1, ["video_id","likes_count"])
result1.to_csv("cats_query1_res.csv", index=False)
```

In [13]:
```python
res1 = pd.read_csv("cats_query1_res.csv")
res1.T
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9
video_id	227321	185103	712861	824787	922204	242091	134015	949241	255345	148742
likes_count	13	13	13	13	12	12	12	12	12	12

> QUERY PLAN without index

In [5]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT video_id, COUNT(*) AS num_like
FROM cats.likes
GROUP BY video_id
ORDER BY num_like DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
14 rows affected.
```

Out[5]:

QUERY PLAN
Limit (cost=227142.89..227142.92 rows=10 width=12) (actual time=44844.844..44845.036 rows=10 loops=1)
-> Sort (cost=227142.89..229021.32 rows=751371 width=12) (actual time=44838.798..44838.876 rows=10 loops=1)
Sort Key: (count(*)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=183636.49..210906.03 rows=751371 width=12) (actual time=32779.633..39068.382 rows=920324 loops=1)
Group Key: video_id
Planned Partitions: 32 Batches: 33 Memory Usage: 4113kB Disk Usage: 63272kB
-> Seq Scan on likes (cost=0.00..41394.47 rows=2528747 width=4) (actual time=0.047..15485.360 rows=2528747 loops=1)
Planning Time: 0.384 ms
JIT:
Functions: 9
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 1.045 ms, Inlining 0.000 ms, Optimization 0.582 ms, Emission 7.094 ms, Total 8.722 ms
Execution Time: 44891.813 ms

Query 2 indexing

In [15]:
```python
#query 2
query2 = """SELECT l.video_id, COUNT(*) AS num_like
            FROM cats.friends f, cats.likes l
            WHERE f.user_id=1
            AND f.friend_id=l.user_id
            GROUP BY l.video_id
            ORDER BY num_like DESC
            LIMIT 10"""

result2 = query_to_df(conn, query2, ["video_id","likes_count"])
result2.to_csv("cats_query2_res.csv", index=False)
```

In [16]:
```python
res2 = pd.read_csv("cats_query2_res.csv")
res2.T
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9
video_id	26412	55489	59420	63405	72715	89512	118321	142269	145324	12407
likes_count	1	1	1	1	1	1	1	1	1	1

> QUERY PLAN without index

In [6]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.video_id, COUNT(*) AS num_like
FROM cats.friends f, cats.likes l
WHERE f.user_id=1
AND f.friend_id=l.user_id
GROUP BY l.video_id
ORDER BY num_like DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
25 rows affected.
```

Out[6]:

QUERY PLAN
Limit (cost=24191.02..24191.04 rows=10 width=12) (actual time=91.247..92.377 rows=10 loops=1)
-> Sort (cost=24191.02..24191.30 rows=114 width=12) (actual time=91.229..92.170 rows=10 loops=1)
Sort Key: (count(*)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> Finalize GroupAggregate (cost=24175.01..24188.55 rows=114 width=12) (actual time=87.564..91.430 rows=60 loops=1)
Group Key: l.video_id
-> Gather Merge (cost=24175.01..24186.93 rows=96 width=12) (actual time=87.456..90.229 rows=60 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial GroupAggregate (cost=23174.99..23175.83 rows=48 width=12) (actual time=61.311..62.005 rows=20 loops=3)
Group Key: l.video_id
-> Sort (cost=23174.99..23175.11 rows=48 width=4) (actual time=61.278..61.573 rows=20 loops=3)
Sort Key: l.video_id
Sort Method: quicksort Memory: 26kB
Worker 0: Sort Method: quicksort Memory: 26kB
Worker 1: Sort Method: quicksort Memory: 25kB
-> Nested Loop (cost=0.43..23173.65 rows=48 width=4) (actual time=20.296..61.075 rows=20 loops=3)
-> Parallel Seq Scan on friends f (cost=0.00..23154.50 rows=4 width=4) (actual time=20.203..60.394 rows=2 loops=3)
Filter: (user_id = 1)
Rows Removed by Filter: 801182
-> Index Only Scan using like_const on likes l (cost=0.43..4.66 rows=13 width=8) (actual time=0.067..0.147 rows=10 loops=6)
Index Cond: (user_id = f.friend_id)
Heap Fetches: 0
Planning Time: 0.443 ms
Execution Time: 92.694 ms

Create index on table friend column user_id

In [12]:
```sql
%%sql
CREATE INDEX friend_user_id_idx ON cats.friends(user_id);
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[12]:
```
[]
```

> QUERY PLAN with index

In [13]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.video_id, COUNT(*) AS num_like
FROM cats.friends f, cats.likes l
WHERE f.user_id=1
AND f.friend_id=l.user_id
GROUP BY l.video_id
ORDER BY num_like DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
15 rows affected.
```

Out[13]:

QUERY PLAN
Limit (cost=87.83..87.86 rows=10 width=12) (actual time=3.796..4.181 rows=10 loops=1)
-> Sort (cost=87.83..88.12 rows=114 width=12) (actual time=3.780..3.919 rows=10 loops=1)
Sort Key: (count(*)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=84.23..85.37 rows=114 width=12) (actual time=2.514..3.137 rows=60 loops=1)
Group Key: l.video_id
Batches: 1 Memory Usage: 24kB
-> Nested Loop (cost=0.86..83.66 rows=114 width=4) (actual time=0.120..2.004 rows=60 loops=1)
-> Index Scan using friend_user_id_idx on friends f (cost=0.43..35.79 rows=10 width=4) (actual time=0.050..0.160 rows=6 loops=1)
Index Cond: (user_id = 1)
-> Index Only Scan using like_const on likes l (cost=0.43..4.66 rows=13 width=8) (actual time=0.014..0.109 rows=10 loops=6)
Index Cond: (user_id = f.friend_id)
Heap Fetches: 0
Planning Time: 0.452 ms
Execution Time: 4.425 ms

> Analyze

Index "friend_user_id_idx" reduced the execution time from 92.69ms to 4.43ms. Seq Scan on friends table was replaced by Index Scan.

Query 3 indexing

In [21]:
```python
#query 3
query3 = """SELECT l.vl, COUNT(*)
            FROM

                 (SELECT l.video_id AS vl, l.user_id AS ul
                  FROM cats.friends f, cats.likes l
                  WHERE f.user_id=1 AND f.friend_id=l.user_id

                  UNION

                  SELECT l.video_id AS vl, l.user_id AS ul
                  FROM cats.friends f, cats.friends ff, cats.likes l
                  WHERE f.user_id=1 AND f.friend_id=ff.user_id AND ff.user_id=l.user_
                 ) AS l

            GROUP BY l.vl
            ORDER BY COUNT(*) DESC
            LIMIT 10"""

result3 = query_to_df(conn, query3, ["video_id","likes_count"])
result3.to_csv("cats_query3_res.csv", index=False)
```

In [22]:
```python
res3 = pd.read_csv("cats_query3_res.csv")
res3.T
```

Out[22]:

	0	1	2	3	4	5	6	7	8	9
video_id	55489	350995	758713	59420	145324	514488	663983	342480	634558	142269
likes_count	1	1	1	1	1	1	1	1	1	1

> QUERY PLAN without index

We have to delete previous created index(friend_user_id_idx) to compare the performance of this query

In [46]:
```sql
%%sql
DROP INDEX cats.friend_user_id_idx
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[46]:
```
[]
```

In [7]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.vl, COUNT(*)
FROM

     (SELECT l.video_id AS vl, l.user_id AS ul
      FROM cats.friends f, cats.likes l
      WHERE f.user_id=1 AND f.friend_id=l.user_id

      UNION

      SELECT l.video_id AS vl, l.user_id AS ul
      FROM cats.friends f, cats.friends ff, cats.likes l
      WHERE f.user_id=1 AND f.friend_id=ff.user_id AND ff.user_id=l.user_id
     ) AS l

GROUP BY l.vl
ORDER BY COUNT(*) DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
39 rows affected.
```

Out[7]:

QUERY PLAN
Limit (cost=75794.44..75794.46 rows=10 width=12) (actual time=9445.716..9446.284 rows=10 loops=1)
-> Sort (cost=75794.44..75794.94 rows=200 width=12) (actual time=9445.701..9446.154 rows=10 loops=1)
Sort Key: (count(*)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> GroupAggregate (cost=75778.28..75790.12 rows=200 width=12) (actual time=9444.167..9445.688 rows=60 loops=1)
Group Key: l.vl
-> Sort (cost=75778.28..75781.56 rows=1311 width=4) (actual time=9444.137..9444.856 rows=60 loops=1)
Sort Key: l.vl
Sort Method: quicksort Memory: 27kB
-> Subquery Scan on l (cost=75684.18..75710.40 rows=1311 width=4) (actual time=9442.682..9444.113 rows=60 loops=1)
-> HashAggregate (cost=75684.18..75697.29 rows=1311 width=8) (actual time=9442.668..9443.428 rows=60 loops=1)
Group Key: l_1.video_id, l_1.user_id
Batches: 1 Memory Usage: 73kB
-> Gather (cost=1000.43..75677.62 rows=1311 width=8) (actual time=0.781..9439.256 rows=601 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Append (cost=0.43..74546.52 rows=1311 width=8) (actual time=293.521..9412.089 rows=200 loops=3)
-> Nested Loop (cost=23154.98..51353.21 rows=499 width=8) (actual time=438.094..9351.133 rows=180 loops=3)
Join Filter: (f.friend_id = l_1.user_id)
-> Parallel Hash Join (cost=23154.55..51316.62 rows=44 width=8) (actual time=438.047..9347.633 rows=20 loops=3)
Hash Cond: (ff.user_id = f.friend_id)
-> Parallel Seq Scan on friends ff (cost=0.00..20650.80 rows=1001480 width=4) (actual time=0.027..4648.078 rows=801184 loops=3)
-> Parallel Hash (cost=23154.50..23154.50 rows=4 width=4) (actual time=51.349..51.394 rows=2 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 104kB
-> Parallel Seq Scan on friends f (cost=0.00..23154.50 rows=4 width=4) (actual time=31.194..51.291 rows=2 loops=3)
Filter: (user_id = 1)
Rows Removed by Filter: 801182
-> Index Only Scan using like_const on likes l_1 (cost=0.43..0.67 rows=13 width=8) (actual time=0.012..0.061 rows=9 loops=60)
Index Cond: (user_id = ff.user_id)
Heap Fetches: 0
-> Nested Loop (cost=0.43..23173.65 rows=48 width=8) (actual time=25.709..87.690 rows=30 loops=2)
-> Parallel Seq Scan on friends f_1 (cost=0.00..23154.50 rows=4 width=4) (actual time=25.616..86.641 rows=3 loops=2)
Filter: (user_id = 1)
Rows Removed by Filter: 1201773
-> Index Only Scan using like_const on likes l_2 (cost=0.43..4.66 rows=13 width=8) (actual time=0.060..0.155 rows=10 loops=6)
Index Cond: (user_id = f_1.friend_id)
Heap Fetches: 0
Planning Time: 0.453 ms
Execution Time: 9446.432 ms

Create index on table friend column user_id

In [48]:
```sql
%%sql
CREATE INDEX friend_user_id_idx ON cats.friends(user_id);
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[48]:
```
[]
```

> QUERY PLAN with index

In [14]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.vl, COUNT(*)
FROM

     (SELECT l.video_id AS vl, l.user_id AS ul
      FROM cats.friends f, cats.likes l
      WHERE f.user_id=1 AND f.friend_id=l.user_id

      UNION

      SELECT l.video_id AS vl, l.user_id AS ul
      FROM cats.friends f, cats.friends ff, cats.likes l
      WHERE f.user_id=1 AND f.friend_id=ff.user_id AND ff.user_id=l.user_id
     ) AS l

GROUP BY l.vl
ORDER BY COUNT(*) DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
30 rows affected.
```

Out[14]:

QUERY PLAN
Limit (cost=316.37..316.39 rows=10 width=12) (actual time=35.682..36.215 rows=10 loops=1)
-> Sort (cost=316.37..316.87 rows=200 width=12) (actual time=35.665..35.992 rows=10 loops=1)
Sort Key: (count(*)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=310.04..312.04 rows=200 width=12) (actual time=34.596..35.315 rows=60 loops=1)
Group Key: l.video_id
Batches: 1 Memory Usage: 40kB
-> HashAggregate (cost=277.27..290.38 rows=1311 width=8) (actual time=33.432..34.133 rows=60 loops=1)
Group Key: l.video_id, l.user_id
Batches: 1 Memory Usage: 73kB
-> Append (cost=0.86..270.71 rows=1311 width=8) (actual time=0.085..28.005 rows=601 loops=1)
-> Nested Loop (cost=0.86..83.66 rows=114 width=8) (actual time=0.068..1.847 rows=60 loops=1)
-> Index Scan using friend_user_id_idx on friends f (cost=0.43..35.79 rows=10 width=4) (actual time=0.017..0.080 rows=6 loops=1)
Index Cond: (user_id = 1)
-> Index Only Scan using like_const on likes l (cost=0.43..4.66 rows=13 width=8) (actual time=0.015..0.119 rows=10 loops=6)
Index Cond: (user_id = f.friend_id)
Heap Fetches: 0
-> Nested Loop (cost=1.29..167.39 rows=1197 width=8) (actual time=0.124..16.651 rows=541 loops=1)
Join Filter: (f_1.friend_id = ff.user_id)
-> Nested Loop (cost=0.86..83.66 rows=114 width=12) (actual time=0.054..1.728 rows=60 loops=1)
-> Index Scan using friend_user_id_idx on friends f_1 (cost=0.43..35.79 rows=10 width=4) (actual time=0.014..0.084 rows=6 loops=1)
Index Cond: (user_id = 1)
-> Index Only Scan using like_const on likes l_1 (cost=0.43..4.66 rows=13 width=8) (actual time=0.014..0.103 rows=10 loops=6)
Index Cond: (user_id = f_1.friend_id)
Heap Fetches: 0
-> Index Only Scan using friend_user_id_idx on friends ff (cost=0.43..0.61 rows=10 width=4) (actual time=0.014..0.088 rows=9 loops=60)
Index Cond: (user_id = l_1.user_id)
Heap Fetches: 0
Planning Time: 0.644 ms
Execution Time: 36.385 ms

> Analyze

Index "friend_user_id_idx" reduced the execution time from 9446ms to 36.39ms. Seq Scan on friends table was replaced by Index Scan.

Query 4 indexing

In [49]:
```python
#query 4
query4 = """SELECT l.video_id, COUNT(*)
            FROM cats.likes l
            WHERE l.user_id
            IN
                (SELECT lb.user_id
                 FROM cats.likes la, cats.likes lb
                 WHERE la.user_id=1 AND la.video_id=lb.video_id
                )
            GROUP BY l.video_id
            ORDER BY COUNT(*) DESC
            LIMIT 10"""

result4 = query_to_df(conn, query4, ["video_id","likes_count"])
result4.to_csv("cats_query4_res.csv", index=False)
```

In [50]:
```python
res4 = pd.read_csv("cats_query4_res.csv")
res4.T
```

Out[50]:

	0	1	2	3	4	5	6	7	8	9
video_id	533261	242696	369447	286557	736628	232267	339117	243889	93818	279828
likes_count	7	6	6	5	5	5	4	4	4	4

> QUERY PLAN without index

In [8]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.video_id, COUNT(*)
FROM cats.likes l
WHERE l.user_id
IN
    (SELECT lb.user_id
     FROM cats.likes la, cats.likes lb
     WHERE la.user_id=500000 AND la.video_id=lb.video_id
    )
GROUP BY l.video_id
ORDER BY COUNT(*) DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
29 rows affected.
```

Out[8]:

QUERY PLAN
Limit (cost=30510.81..30510.83 rows=10 width=12) (actual time=36.739..40.855 rows=0 loops=1)
-> Sort (cost=30510.81..30512.29 rows=594 width=12) (actual time=36.723..40.821 rows=0 loops=1)
Sort Key: (count(*)) DESC
Sort Method: quicksort Memory: 25kB
-> GroupAggregate (cost=30487.58..30497.97 rows=594 width=12) (actual time=36.704..40.788 rows=0 loops=1)
Group Key: l.video_id
-> Sort (cost=30487.58..30489.06 rows=594 width=4) (actual time=36.689..40.757 rows=0 loops=1)
Sort Key: l.video_id
Sort Method: quicksort Memory: 25kB
-> Nested Loop (cost=30419.04..30460.21 rows=594 width=4) (actual time=36.670..40.725 rows=0 loops=1)
-> HashAggregate (cost=30418.61..30419.05 rows=44 width=4) (actual time=36.655..40.693 rows=0 loops=1)
Group Key: lb.user_id
Batches: 1 Memory Usage: 24kB
-> Gather (cost=1004.82..30418.50 rows=44 width=4) (actual time=36.637..40.660 rows=0 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Hash Join (cost=4.82..29414.10 rows=18 width=4) (actual time=0.154..0.250 rows=0 loops=3)
Hash Cond: (lb.video_id = la.video_id)
-> Parallel Seq Scan on likes lb (cost=0.00..26643.45 rows=1053645 width=8) (actual time=0.035..0.042 rows=1 loops=3)
-> Hash (cost=4.66..4.66 rows=13 width=4) (actual time=0.036..0.102 rows=0 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 8kB
-> Index Only Scan using like_const on likes la (cost=0.43..4.66 rows=13 width=4) (actual time=0.021..0.028 rows=0 loops=3)
Index Cond: (user_id = 500000)
Heap Fetches: 0
-> Index Only Scan using like_const on likes l (cost=0.43..0.81 rows=13 width=8) (never executed)
Index Cond: (user_id = lb.user_id)
Heap Fetches: 0
Planning Time: 0.312 ms
Execution Time: 40.940 ms

Seq scan exists on user_id in table likes.

Create index on user_id in likes table

In [17]:
```sql
%%sql
CREATE INDEX like_user_id_idx ON cats.likes(user_id);
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[17]:
```
[]
```

> QUERY PLAN with index

In [18]:
```sql
%%sql
EXPLAIN ANALYZE
SELECT l.video_id, COUNT(*)
FROM cats.likes l
WHERE l.user_id
IN
    (SELECT lb.user_id
     FROM cats.likes la, cats.likes lb
     WHERE la.user_id=500000 AND la.video_id=lb.video_id
    )
GROUP BY l.video_id
ORDER BY COUNT(*) DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
29 rows affected.
```

Out[18]:

QUERY PLAN
Limit (cost=30510.81..30510.83 rows=10 width=12) (actual time=36.026..40.666 rows=0 loops=1)
-> Sort (cost=30510.81..30512.29 rows=594 width=12) (actual time=36.010..40.635 rows=0 loops=1)
Sort Key: (count(*)) DESC
Sort Method: quicksort Memory: 25kB
-> GroupAggregate (cost=30487.58..30497.97 rows=594 width=12) (actual time=35.991..40.582 rows=0 loops=1)

QUERY PLAN
Group Key: l.video_id
-> Sort (cost=30487.58..30489.06 rows=594 width=4) (actual time=35.975..40.546 rows=0 loops=1)
Sort Key: l.video_id
Sort Method: quicksort Memory: 25kB
-> Nested Loop (cost=30419.04..30460.21 rows=594 width=4) (actual time=35.957..40.500 rows=0 loops=1)
-> HashAggregate (cost=30418.61..30419.05 rows=44 width=4) (actual time=35.942..40.470 rows=0 loops=1)
Group Key: lb.user_id
Batches: 1 Memory Usage: 24kB
-> Gather (cost=1004.82..30418.50 rows=44 width=4) (actual time=35.927..40.438 rows=0 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Hash Join (cost=4.82..29414.10 rows=18 width=4) (actual time=0.159..0.276 rows=0 loops=3)
Hash Cond: (lb.video_id = la.video_id)
-> Parallel Seq Scan on likes lb (cost=0.00..26643.45 rows=1053645 width=8) (actual time=0.018..0.025 rows=1 loops=3)
-> Hash (cost=4.66..4.66 rows=13 width=4) (actual time=0.051..0.076 rows=0 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 8kB
-> Index Only Scan using like_const on likes la (cost=0.43..4.66 rows=13 width=4) (actual time=0.035..0.045 rows=0 loops=3)
Index Cond: (user_id = 500000)
Heap Fetches: 0
-> Index Only Scan using like_const on likes l (cost=0.43..0.81 rows=13 width=8) (never executed)
Index Cond: (user_id = lb.user_id)
Heap Fetches: 0
Planning Time: 0.599 ms
Execution Time: 40.751 ms

> ### Analyze

Performance doesn't change too much as the Seq Scan was selected as the optimal solution for the rendering data entries.

Query 5 indexing

In [55]:

```
#query 5
query5 = """WITH UserWeight AS
            (
             SELECT lb.user_id, LOG(1+COUNT(*)) AS weight
             FROM cats.likes la, cats.likes lb
             WHERE la.user_id=1 AND la.video_id=lb.video_id
             GROUP BY lb.user_id
             )
        SELECT l.video_id, SUM(w.weight) AS sum_weight FROM cats.likes l,  UserWe
        WHERE l.user_id=w.user_id
        GROUP BY l.video_id
        ORDER BY sum_weight DESC
        LIMIT 10"""

result5 = query_to_df(conn, query5, ["video_id","likes_count"])
result5.to_csv("cats_query5_res.csv", index=False)
```

In [56]:

```
res5 = pd.read_csv("cats_query4_res.csv")
res5.T
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9
video_id	533261	242696	369447	286557	736628	232267	339117	243889	93818	279828
likes_count	7	6	6	5	5	5	4	4	4	4

> #### QUERY PLAN without index

Similarly, we drop the previous created like_user_id_idx first.

In [29]:

```
%%sql
DROP INDEX cats.like_user_id_idx;
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[29]:

```
[]
```

In [9]:

```
%%sql
EXPLAIN ANALYZE
WITH UserWeight AS
     (
      SELECT lb.user_id, LOG(1+COUNT(*)) AS weight
      FROM cats.likes la, cats.likes lb
      WHERE la.user_id=9999 AND la.video_id=lb.video_id
      GROUP BY lb.user_id
      )
SELECT l.video_id, SUM(w.weight) AS sum_weight FROM cats.likes l,  UserWeight w
WHERE l.user_id=w.user_id
GROUP BY l.video_id
ORDER BY sum_weight DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
35 rows affected.
```

Out[9]:

QUERY PLAN
Limit (cost=30681.60..30681.63 rows=10 width=12) (actual time=9824.635..9825.006 rows=10 loops=1)
-> Sort (cost=30681.60..30683.09 rows=594 width=12) (actual time=9824.619..9824.874 rows=10 loops=1)
Sort Key: (sum((log(((1 + count(*)))::double precision)))) DESC
Sort Method: top-N heapsort Memory: 25kB
-> GroupAggregate (cost=30658.37..30668.77 rows=594 width=12) (actual time=9809.714..9820.999 rows=575 loops=1)
Group Key: l.video_id
-> Sort (cost=30658.37..30659.86 rows=594 width=12) (actual time=9809.685..9813.375 rows=618 loops=1)
Sort Key: l.video_id
Sort Method: quicksort Memory: 53kB
-> Nested Loop (cost=30414.92..30631.00 rows=594 width=12) (actual time=9791.890..9806.095 rows=618 loops=1)
-> Finalize GroupAggregate (cost=30414.49..30419.91 rows=44 width=12) (actual time=9791.811..9793.150 rows=43 loops=1)
Group Key: lb.user_id
-> Gather Merge (cost=30414.49..30418.96 rows=36 width=12) (actual time=9791.739..9792.560 rows=43 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial GroupAggregate (cost=29414.47..29414.79 rows=18 width=12) (actual time=9767.435..9767.731 rows=14 loops=3)
Group Key: lb.user_id
-> Sort (cost=29414.47..29414.52 rows=18 width=4) (actual time=9767.330..9767.489 rows=20 loops=3)
Sort Key: lb.user_id
Sort Method: quicksort Memory: 25kB
Worker 0: Sort Method: quicksort Memory: 26kB
Worker 1: Sort Method: quicksort Memory: 25kB
-> Hash Join (cost=4.82..29414.10 rows=18 width=4) (actual time=562.370..9767.223 rows=20 loops=3)
Hash Cond: (lb.video_id = la.video_id)
-> Parallel Seq Scan on likes lb (cost=0.00..26643.45 rows=1053645 width=8) (actual time=0.025..4882.784 rows=842916 loops=3)
-> Hash (cost=4.66..4.66 rows=13 width=4) (actual time=0.301..0.317 rows=17 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Index Only Scan using like_const on likes la (cost=0.43..4.66 rows=13 width=4) (actual time=0.031..0.162 rows=17 loops=3)
Index Cond: (user_id = 9999)
Heap Fetches: 0
-> Index Only Scan using like_const on likes l (cost=0.43..4.66 rows=13 width=8) (actual time=0.024..0.116 rows=14 loops=43)
Index Cond: (user_id = lb.user_id)
Heap Fetches: 0
Planning Time: 0.272 ms
Execution Time: 9825.151 ms

Create index on user_id in likes table

In [10]:

```
%%sql
CREATE INDEX like_user_id_idx ON cats.likes(user_id);
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[10]:

```
[]
```

> #### QUERY PLAN with index

In [11]:

```
%%sql
EXPLAIN ANALYZE
WITH UserWeight AS
     (
      SELECT lb.user_id, LOG(1+COUNT(*)) AS weight
      FROM cats.likes la, cats.likes lb
      WHERE la.user_id=1 AND la.video_id=lb.video_id
      GROUP BY lb.user_id
      )
SELECT l.video_id, SUM(w.weight) AS sum_weight
FROM cats.likes l,  UserWeight w
WHERE l.user_id=w.user_id
GROUP BY l.video_id
ORDER BY sum_weight DESC
LIMIT 10
```

```
 * postgresql://postgres:***@localhost/postgres
35 rows affected.
```

Out[11]:

QUERY PLAN
Limit (cost=30681.60..30681.63 rows=10 width=12) (actual time=9764.907..9765.270 rows=10 loops=1)
-> Sort (cost=30681.60..30683.09 rows=594 width=12) (actual time=9764.893..9765.140 rows=10 loops=1)
Sort Key: (sum((log(((1 + count(*)))::double precision)))) DESC
Sort Method: top-N heapsort Memory: 25kB
-> GroupAggregate (cost=30658.37..30668.77 rows=594 width=12) (actual time=9748.046..9761.234 rows=618 loops=1)
Group Key: l.video_id
-> Sort (cost=30658.37..30659.86 rows=594 width=12) (actual time=9748.017..9752.235 rows=670 loops=1)
Sort Key: l.video_id
Sort Method: quicksort Memory: 56kB
-> Nested Loop (cost=30414.92..30631.00 rows=594 width=12) (actual time=9728.393..9743.921 rows=670 loops=1)
-> Finalize GroupAggregate (cost=30414.49..30419.91 rows=44 width=12) (actual time=9728.357..9730.259 rows=53 loops=1)
Group Key: lb.user_id
-> Gather Merge (cost=30414.49..30418.96 rows=36 width=12) (actual time=9728.283..9729.397 rows=53 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial GroupAggregate (cost=29414.47..29414.79 rows=18 width=12) (actual time=9702.984..9703.352 rows=18 loops=3)
Group Key: lb.user_id
-> Sort (cost=29414.47..29414.52 rows=18 width=4) (actual time=9702.883..9703.064 rows=24 loops=3)
Sort Key: lb.user_id
Sort Method: quicksort Memory: 27kB
Worker 0: Sort Method: quicksort Memory: 25kB
Worker 1: Sort Method: quicksort Memory: 25kB
-> Hash Join (cost=4.82..29414.10 rows=18 width=4) (actual time=613.999..9702.721 rows=24 loops=3)
Hash Cond: (lb.video_id = la.video_id)
-> Parallel Seq Scan on likes lb (cost=0.00..26643.45 rows=1053645 width=8) (actual time=0.030..4851.307 rows=842916 loops=3)
-> Hash (cost=4.66..4.66 rows=13 width=4) (actual time=0.384..0.400 rows=19 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Index Only Scan using like_const on likes la (cost=0.43..4.66 rows=13 width=4) (actual time=0.039..0.199 rows=19 loops=3)
Index Cond: (user_id = 1)
Heap Fetches: 0
-> Index Only Scan using like_const on likes l (cost=0.43..4.66 rows=13 width=8) (actual time=0.020..0.097 rows=13 loops=53)
Index Cond: (user_id = lb.user_id)
Heap Fetches: 0
Planning Time: 0.431 ms
Execution Time: 9765.399 ms

> ### Analyze

Performance doesn't change too much as the Seq Scan was still selected as the optimal solution for the rendering data entries.