Pre-Computation

# CATS

## Original Query

WITH WeightOfUsers

AS (

SELECT ly.user\_id

,LOG(1 + COUNT(\*)) AS weight

FROM cats.LIKE lx

,cats.LIKE ly

WHERE lx.user\_id = 1

AND lx.video\_id = ly.video\_id

GROUP BY ly.user\_id

)

SELECT l.video\_id

,SUM(w.weight) AS sum\_weight

FROM cats.LIKE l

,WeightOfUsers w

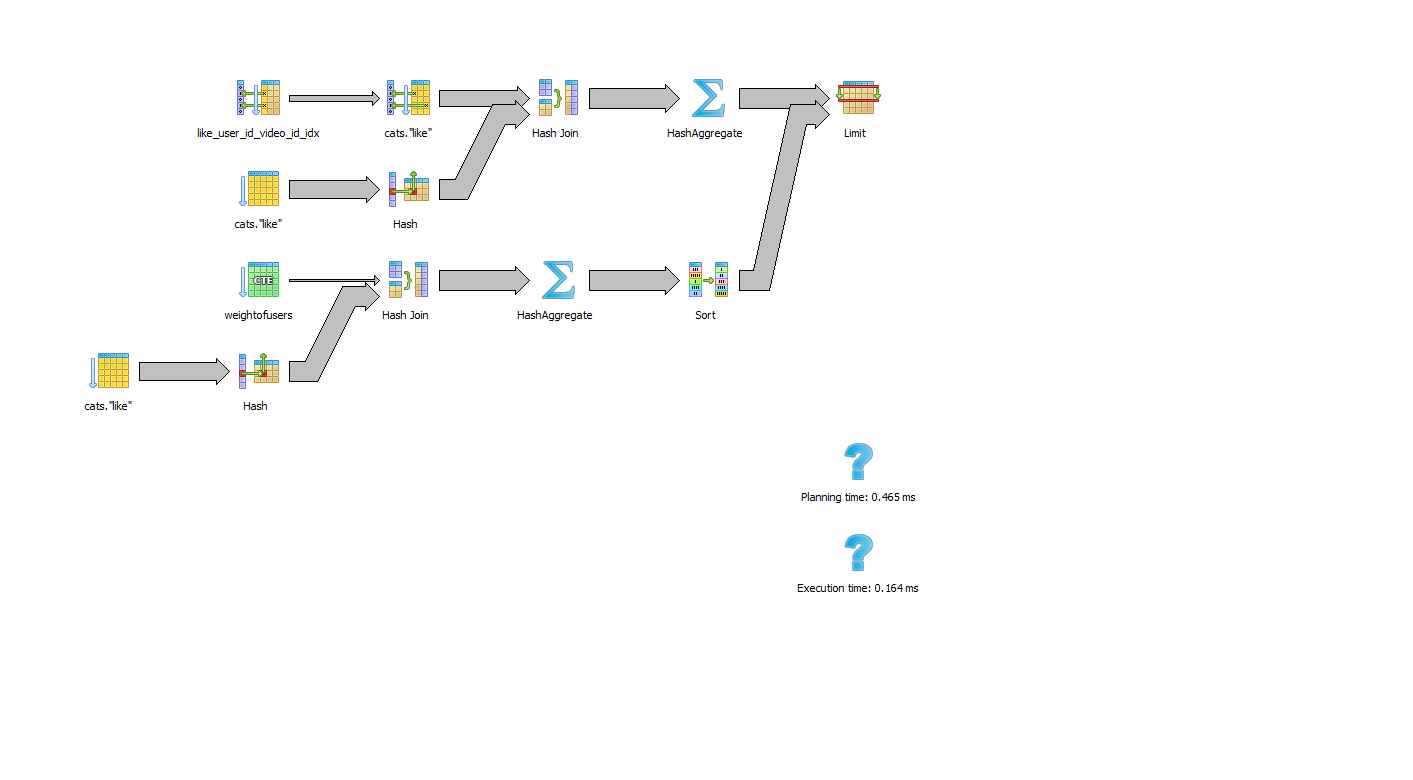
WHERE l.user\_id = w.user\_id

GROUP BY l.video\_id

ORDER BY sum\_weight DESC LIMIT 10

|  |  |  |
| --- | --- | --- |
| Query Cost | Execution Time | Table IO |
| 257881.01..257881.04 | 179.970 ms | 12740 |

### Query Plan:



## New Query

WITH WeightOfUsers

AS (

SELECT y\_user\_Id AS user\_id

,log(sum(1 + weight)) AS weight

FROM cats.PRE\_WeightOfUsers

WHERE x\_user\_id = 49976

GROUP BY y\_user\_id

)

SELECT l.video\_id

,SUM(w.weight) AS sum\_weight

FROM cats.LIKE l

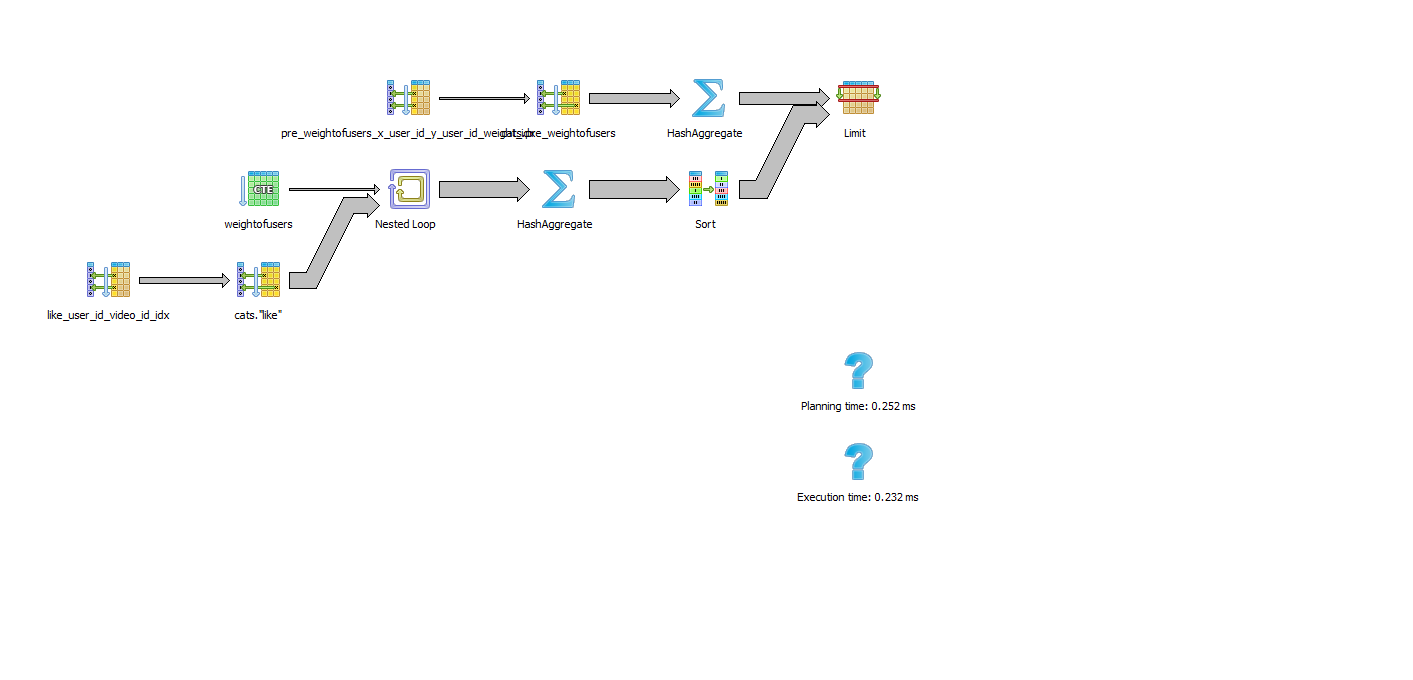
,WeightOfUsers w

WHERE l.user\_id = w.user\_id

GROUP BY l.video\_id

ORDER BY sum\_weight DESC LIMIT 10

|  |  |  |
| --- | --- | --- |
| Query Cost | Execution Time | Table IO |
| 10678.39..10678.42 | 0.339 ms | 15 |



## Pre-Computation

DROP TABLE IF EXISTS cats.PRE\_WeightOfUsers;

CREATE TABLE cats.PRE\_WeightOfUsers AS

SELECT lx.user\_id AS x\_user\_id,

ly.user\_id AS y\_user\_id,

COUNT(\*) AS weight

FROM cats.LIKE lx

,cats.LIKE ly

WHERE lx.video\_id = ly.video\_id

GROUP BY lx.user\_id, ly.user\_id;

## Trigger

## CREATE or replace FUNCTION cats.FN\_PRE\_WeightOfUsers() RETURNS trigger AS $FN\_PRE\_WeightOfUsers$

## BEGIN

## CREATE TEMP TABLE users as

## SELECT user\_id

## FROM cats.LIKE

## WHERE video\_id = NEW.video\_id

## UNION

## SELECT NEW.user\_id;

## 

## WITH U AS (

## SELECT user\_id

## FROM users

## )

## UPDATE CATS.PRE\_WeightOfUsers

## SET weight = weight + 1

## WHERE (

## x\_user\_id IN (

## SELECT user\_id

## FROM U

## )

## AND y\_user\_id = NEW.user\_id

## )

## OR (

## y\_user\_id IN (

## SELECT user\_id

## FROM U

## )

## AND x\_user\_id = NEW.user\_id

## );

## WITH U AS (

## SELECT user\_id

## FROM users

## ), I as (

## SELECT u.user\_id as x\_user\_id, NEW.user\_id as y\_user\_id

## FROM U

## 

## UNION

## 

## SELECT NEW.user\_id as x\_user\_id , u.user\_id as y\_user\_id

## FROM U

## 

## UNION

## 

## SELECT NEW.user\_id as x\_user\_id, NEW.user\_id as y\_user\_id

## ), X as (

## SELECT \*

## FROM I

## EXCEPT

## SELECT x\_user\_id

## ,y\_user\_id

## FROM Cats.PRE\_WeightOfUsers

## )

## INSERT INTO cats.PRE\_WeightOfUsers (

## x\_user\_id

## ,y\_user\_id

## ,weight

## )

## SELECT \*

## ,1

## FROM X;

## 

## RETURN NEW;

## END;

## $FN\_PRE\_WeightOfUsers$ LANGUAGE plpgsql;

## DROP TRIGGER IF EXISTS trigger\_LIKE ON CATS.LIKE ;

## CREATE TRIGGER trigger\_LIKE

## BEFORE INSERT ON CATS.LIKE

## FOR EACH ROW EXECUTE PROCEDURE cats.FN\_PRE\_WeightOfUsers();

## Maintenance Costs

The trigger used to maintain the pre-computed table has 3 parts: temp table, update and insert. The cost for each provided below.

|  |  |
| --- | --- |
|  | Cost |
| Temp Table | 1363.68..1367.60 |
| Update | 3175.68..4097.24 |
| Insert | 5403.67..5413.29 |
| Total | 9943.03..10878.13 |

## Indexes

The following indexes were used to the trigger to improve maintenance cost or to improve the cost of the final query.

### pre\_weightofusers\_x\_user\_id\_y\_user\_id\_idx

CREATE INDEX pre\_weightofusers\_x\_user\_id\_idx

ON cats.pre\_weightofusers

USING btree

(x\_user\_id);

### pre\_weightofusers\_x\_user\_id\_y\_user\_id\_weight\_idx

CREATE INDEX pre\_weightofusers\_x\_user\_id\_y\_user\_id\_weight\_idx

ON cats.pre\_weightofusers

USING btree

(x\_user\_id, y\_user\_id, weight);

### like\_video\_id\_user\_id\_idx

CREATE INDEX like\_video\_id\_user\_id\_idx

ON cats."like"

USING btree

(video\_id, user\_id);

# Conclusion

|  |  |  |  |
| --- | --- | --- | --- |
|  | Query Cost | Execution Time | Table IO |
| Original Query | 257881.01..257881.04 | 179.970 ms | 12740 |
| Using Pre-compute | 10678.39..10678.42 | 0.339 ms | 15 |
| Difference | (247202.61..247202.62) | (179.631) ms | (12725) |

Maintenance cost for n=1 (1 new like): 9943.03..10878.13

Given the maintenance execution cost of 10,878.13, using the pre-computation strategy described would be optimal for values of n between 2 and 22.