CSCI 620/Section 03/Mior,

Introduction to Big Data, Spring 2215

Assignment 2 – SQL

Submitted by –

Prakhar Gupta(pg9349)

Ans 1.

Refer to codes – code 1 to code 4

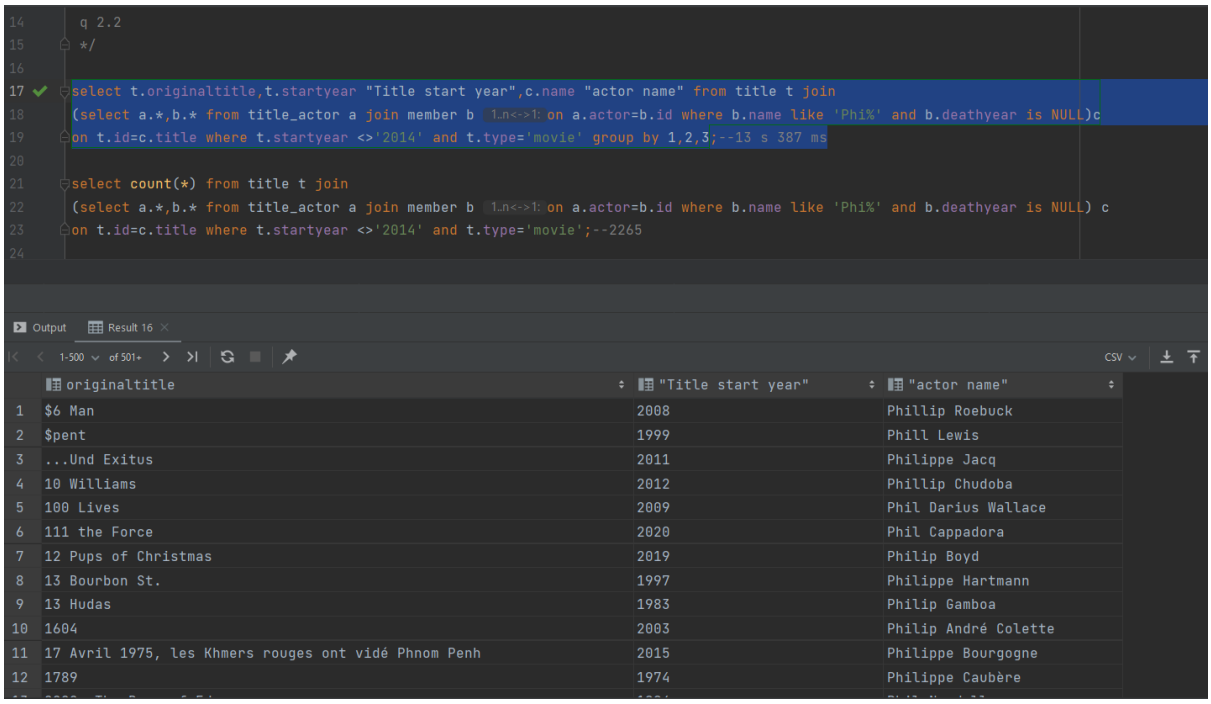
Ans 2

Refer to code – q2.sql

Query 2.1 -

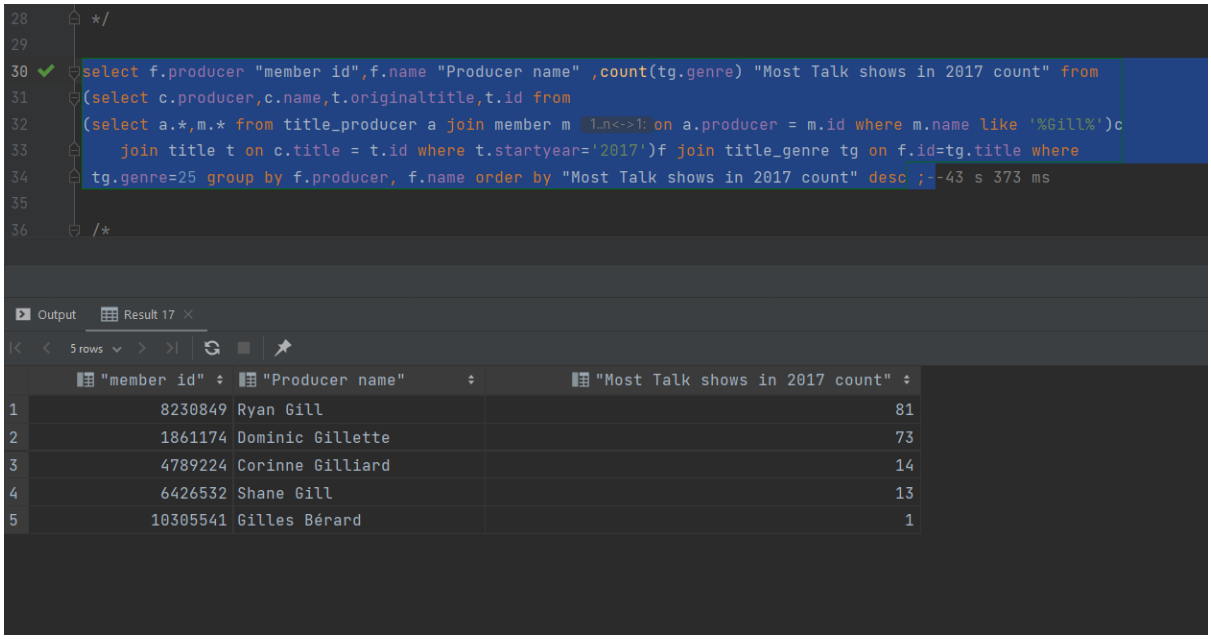
Runtime : 11 s 736 ms

Query 2.2 -



Runtime : 17 s 387 ms

Query 2.3 -



```
*/

select f.producer "member id",f.name "Producer name" ,count(tg.genre) "Most Talk shows in 2017 count" from
(select c.producer,c.name,t.originaltitle,t.id from
(select a.*,m.* from title_producer a join member m  1..n<->1: on a.producer = m.id where m.name like '%Gill%')c
   join title t on c.title = t.id where t.startyear='2017')f join title_genre tg on f.id=tg.title where
tg.genre=25 group by f.producer, f.name order by "Most Talk shows in 2017 count" desc ;--43 s 373 ms

/*
```

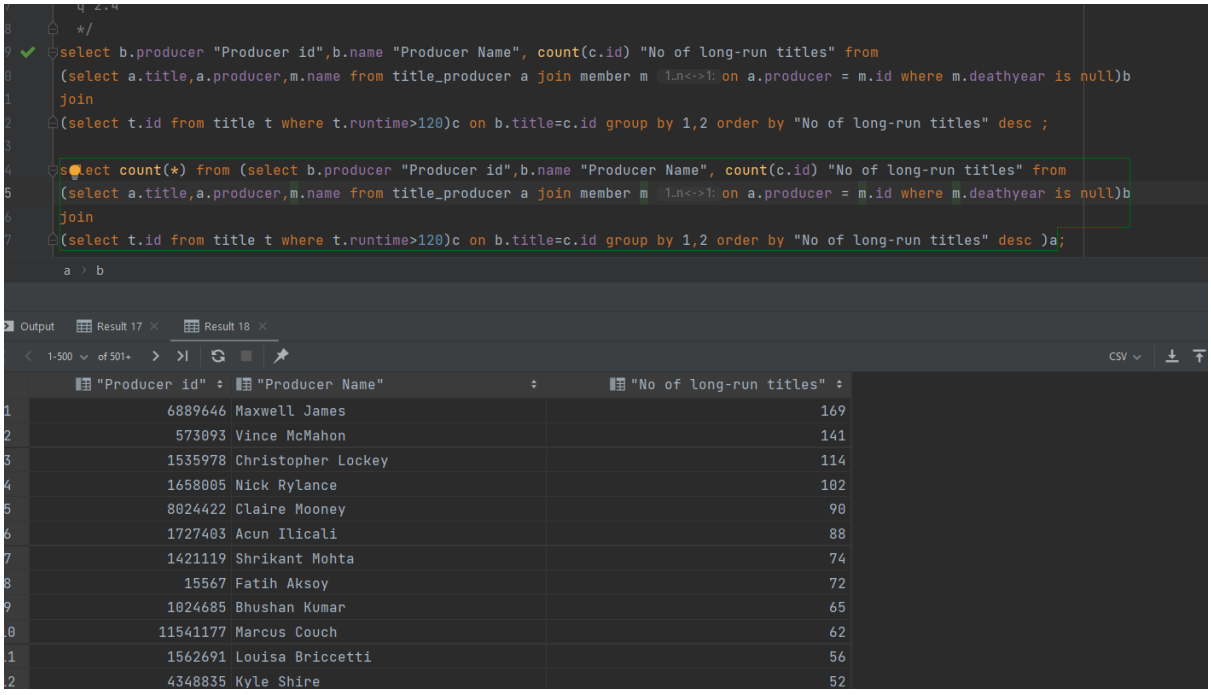
Output | Result 17

5 rows

	"member id"	"Producer name"	"Most Talk shows in 2017 count"
1	8230849	Ryan Gill	81
2	1861174	Dominic Gillette	73
3	4789224	Corinne Gilliard	14
4	6426532	Shane Gill	13
5	10305541	Gilles Bérard	1

Runtime : 55 s 967 ms

Query 2.4 -



```
q 2.4
*/
select b.producer "Producer id",b.name "Producer Name", count(c.id) "No of long-run titles" from
(select a.title,a.producer,m.name from title_producer a join member m  1..n<->1: on a.producer = m.id where m.deathyear is null)b
join
(select t.id from title t where t.runtime>120)c on b.title=c.id group by 1,2 order by "No of long-run titles" desc ;

select count(*) from (select b.producer "Producer id",b.name "Producer Name", count(c.id) "No of long-run titles" from
(select a.title,a.producer,m.name from title_producer a join member m  1..n<->1: on a.producer = m.id where m.deathyear is null)b
join
(select t.id from title t where t.runtime>120)c on b.title=c.id group by 1,2 order by "No of long-run titles" desc )a;
```

a › b

Output | Result 17 | Result 18

1-500 of 501+ | CSV

	"Producer id"	"Producer Name"	"No of long-run titles"
1	6889646	Maxwell James	169
2	573093	Vince McMahon	141
3	1535978	Christopher Lockey	114
4	1658005	Nick Rylance	102
5	8024422	Claire Mooney	90
6	1727403	Acun Ilicali	88
7	1421119	Shrikant Mohta	74
8	15567	Fatih Aksoy	72
9	1024685	Bhushan Kumar	65
10	11541177	Marcus Couch	62
11	1562691	Louisa Briccetti	56
12	4348835	Kyle Shire	52

Runtime : 11 s 126 ms

Query 2.5

```
select a1.actor,m.name,a1.character from
(select atc.actor,atc.title,c.character
from character c join actor_title_character atc 1<->1…n: on c.id=atc.character where c.character='Jesus Christ')a1
join member m on a1.actor=m.id where deathyear is null group by 1,2,3;


select count(*) from (select a1.title,m.name from
(select atc.actor,atc.title,c.character
from character c join actor_title_character atc 1<->1…n: on c.id=atc.character where c.character='Jesus Christ')a1
join member m on a1.actor=m.id where deathyear is null group by 1,2)a;
```

Output | Result 19 | Result 20

141 rows

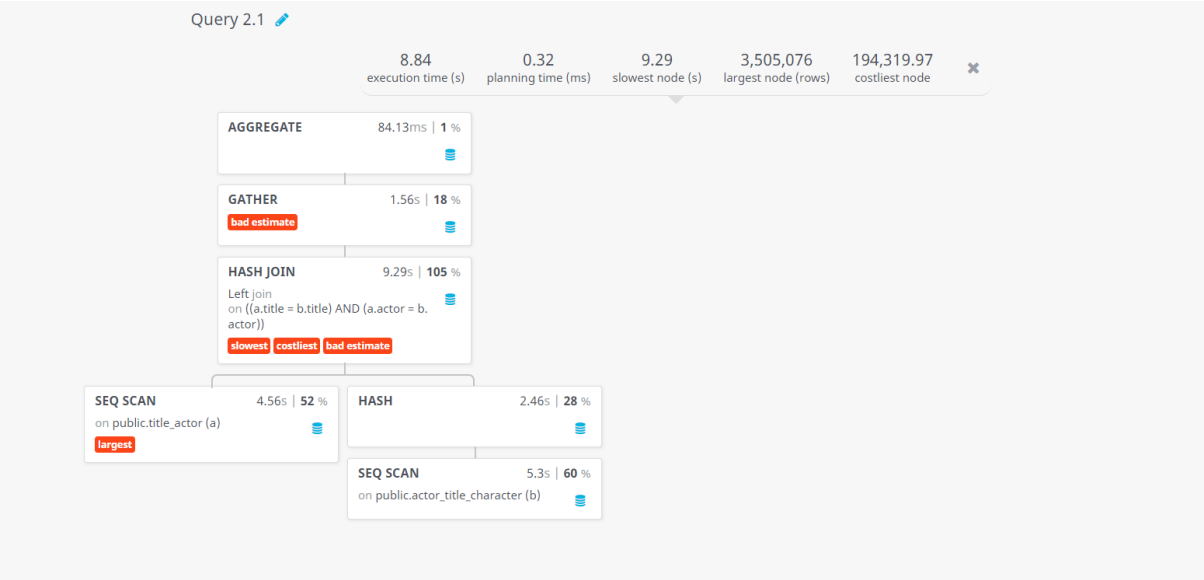
	actor	name	character
1	1293	Seth Green	Jesus Christ
2	1697	Chris Sarandon	Jesus Christ
3	4873	Andy Dick	Jesus Christ
4	5227	Breckin Meyer	Jesus Christ
5	5548	Joel West	Jesus Christ
6	32177	Greg Apparcel	Jesus Christ
7	60150	John Bassberger	Jesus Christ
8	73088	Michael Benyaer	Jesus Christ
9	99584	Russell Boulter	Jesus Christ
10	127438	Joe Caballero	Jesus Christ
11	172301	Emmy Collins	Jesus Christ
12	193738	Henry Ian Cusick	Jesus Christ

Runtime : 1 s 406 ms

Ans 3

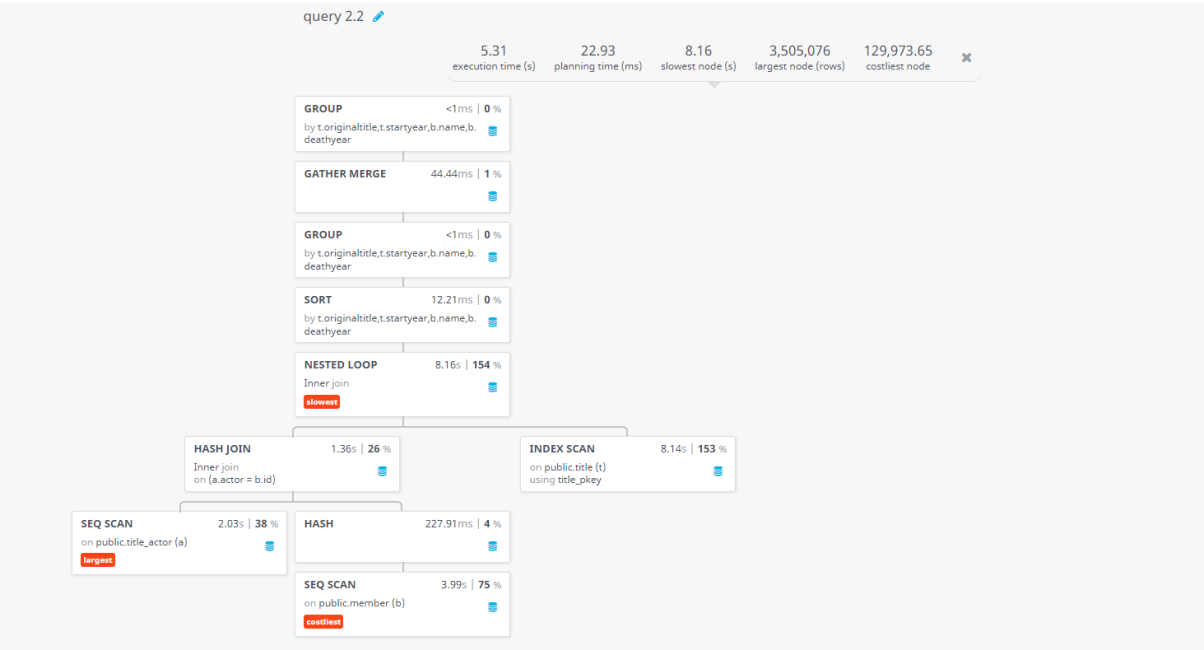
(Refer to q3.sql)

Query 1



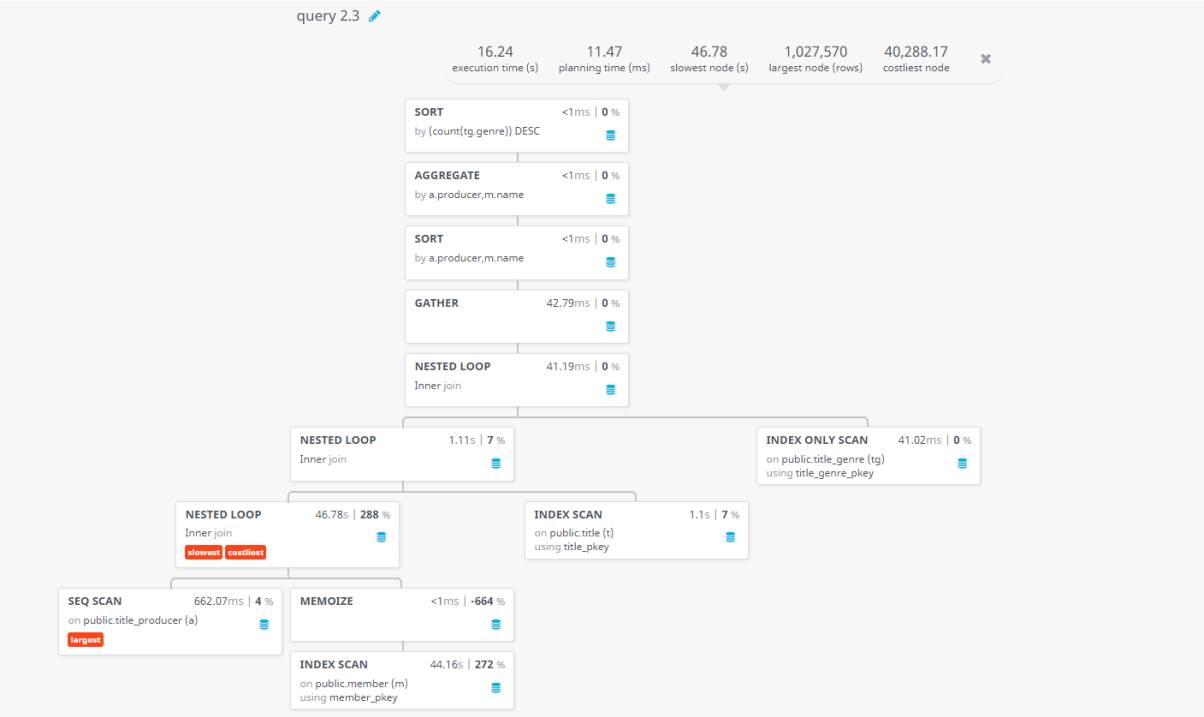
We left join title actor with actor_title_character table on title and actor fields. This is represented by the diagram by hash join (left) which is the slowest node. For hash join to take place one of the table columns need to be hashed hence after sequential read of actor_title_character hashing takes place. The results is checked for tuples where character field is null and finally we count number of rows in the aggregate function above.

Query 2



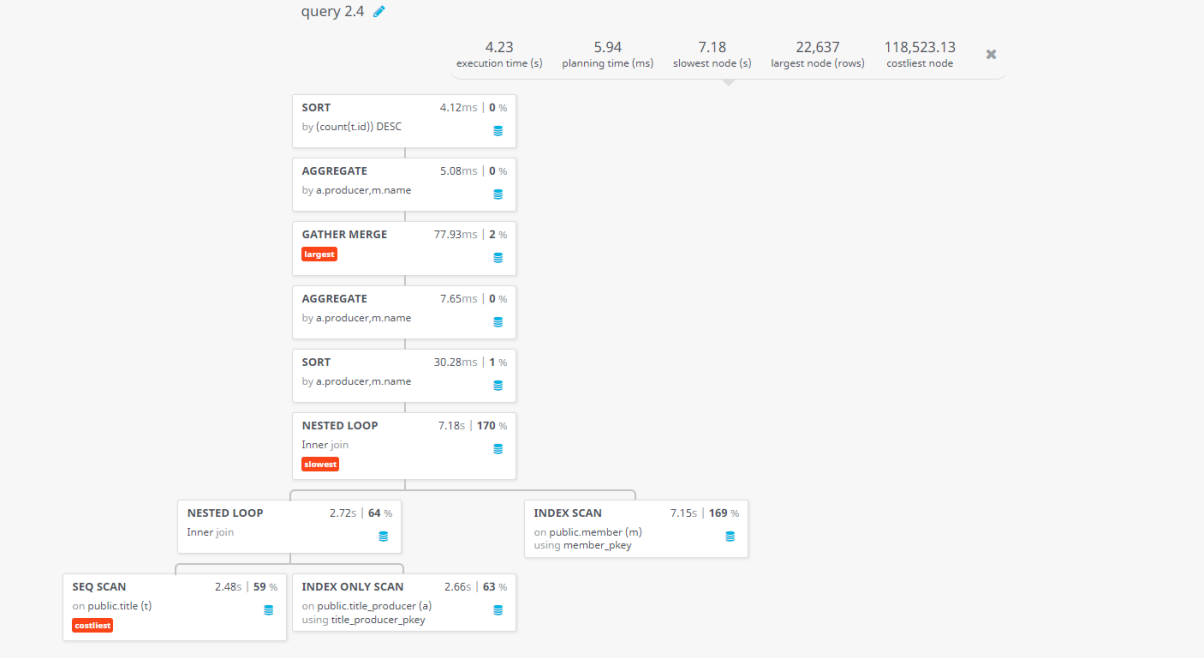
We join both title actor and member table on actor and id respectively also filtering that member name is like ‘Phi%’. This resulted in inner joined with title table on title id primary key. At this point instead of seq scan index scan runs as id is primary key in title table. Here we check that title startyear is not 2014 and it’s a movie. We then group by all the columns to prevent any duplicate records being printed (safety measure).

Query 3



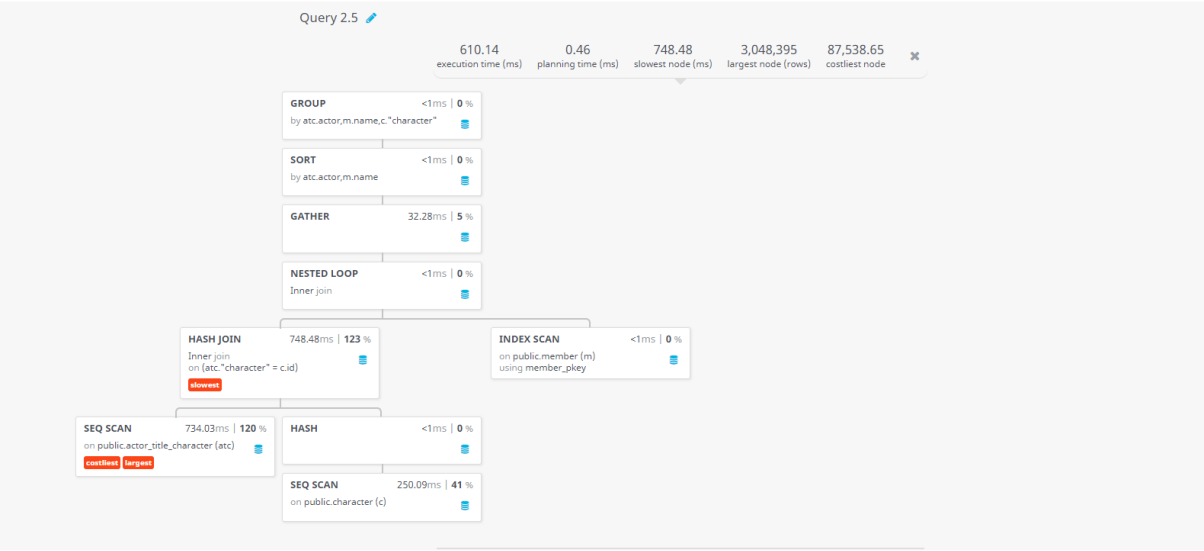
Firstly we join title_producer with member table where name is like ‘%Gill%’ represented by the nested loop node above. The results are then joined with title title on primary key id where startyear is ‘2017’. This is aggregated(group by) over producer name and id with count of number of titles they produced.

Query 4



Title table title_producer table and member table each are joined to using inner join(nested loop) where title have more than 120 minutes of runtime joined on title and id in title table with producers which have null in deathyear(alive producers). This is aggregated over and count of titles is used for each of the above producers.

Query 5



The actor_title_character table is joined with character table where character is ‘Jesus Christ’ on character.id and actor_title_character.character . This gives actors who have played role of jesus Christ. This is then joined with the member table on member id vs actor id . The rows are filtered for actors where deathyear is null.

Ans 4)

4.1) $\Pi_{\text{count}(\text{title\_actor.title})}\left(\Pi_{\text{title\_actor.title}, \text{title\_actor.actor}}(\text{title\_actor}) - \Pi_{\text{actor\_title\_character.title}, \text{actor\_title\_character.actor}}(\text{actor\_title\_character})\right)$

4.2)

$$\Pi_{member.name,\ member.id}\Big(\sigma_{title.id = title\_actor.title \wedge title.start\_year != '2014' \wedge title.type = 'movie'}\Big(title \times \Big(\sigma_{title\_actor.actor = member.id \wedge member.name\ like\ ('Phi\%') \wedge (death\_year\ is\ null)}\Big(title\_actor \times member\Big)\Big)\Big)\Big)$$

4.3) Assumption: we know "talkshow" genre id, its 25 in our case

$$\Pi_{member.id,\ member.name}\Big(\sigma_{title\_genre.title=titl e.id \wedge title\_genre.genre=25}\Big(\sigma_{producer.title=title.id \wedge title.startyear='2017'}\Big(\sigma_{title\_producer.producer=member.id \wedge member.name\ like('\%Gill\%')}\big((title\_producer \times member) \times title\big) \times title\_genre\big)\Big)\Big)$$

4.4) $\Pi_{title.originaltitle}\left(\sigma_{title\_producer.title = title.id}\left(\right.\right.$

$\left(\sigma_{title\_producer.producer = member.id \wedge member.deathyear}\right.$

$\text{is null}\left(title\_producers \times member\right)\Big) \times$

$\left.\left.\left(\sigma_{runtime>120}(titles)\right)\right)\right)$

4.5) $\Pi_{member.name}\left(\sigma_{member.id = actor\_title\_character.actor}\right.$

$\left(\left(\sigma_{actor = member.id \wedge death\_year \text{ is null}}\left(title\_actor \times member\right)\right)\right.$

$\times$

$\left(\left(\sigma_{character.id = actor\_title\_character.character \wedge}\right.\right.$

$character.character = \text{'Jesus'}\left(character \times actor\_title\_character\right)\Big)$

$\cup$

$\left(\sigma_{character.id = actor\_title\_character.character \wedge}\right.$

$character = \text{"christ"}\left(character \times actor\_title\_character\right)\Big)\Big)\Big)$

Ans 5

(Refer to q5.sql)

After indexing increment in performance was observed. Below are the values regarding the increase in performance –

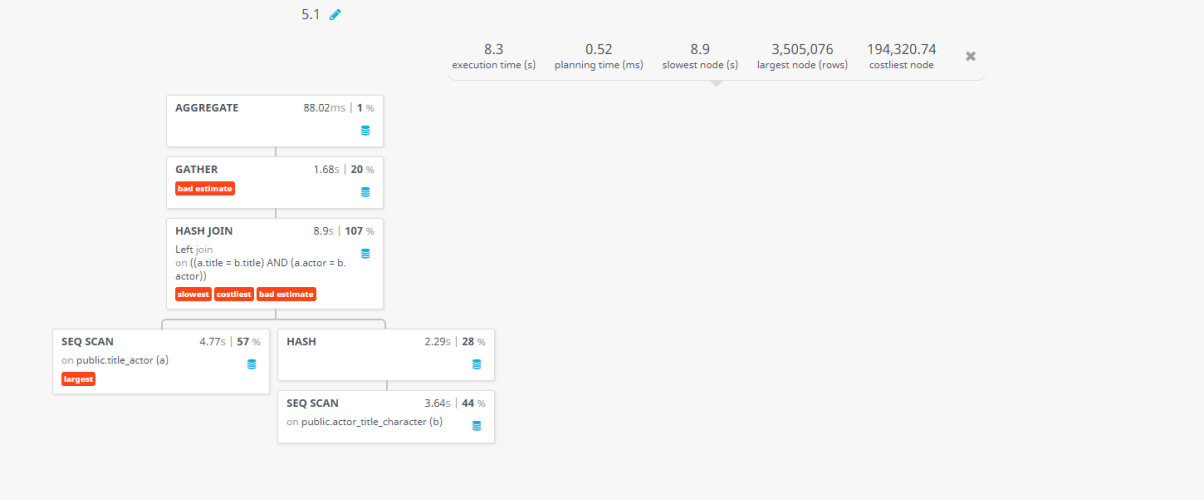
- Query 1 – Runtime: 10 s 515 ms
- Query 2 – Runtime: 11 s 892 ms
- Query 3 – Runtime: 42 s 497 ms
- Query 4 – Runtime: 7 s 480 ms
- Query 5 – Runtime: 203 ms

Indexing was done on the selected columns only. One approach was to use index was to apply indexing on columns which are used to join two tables. The idea was that through indexing the joining of two tables could be faster as their respective indexes could be used to join them. This did result in reduction of query time. Second approach was to apply indexing on the columns which were used for filtering. Minor changes were noticed. Indexing the columns which were showing as slowest nodes did help to speed queries up.

One noticeable change in the execution plans was use of BitMap Heap Scan and BitMap Index scan instead of Seq Scans on table nodes.

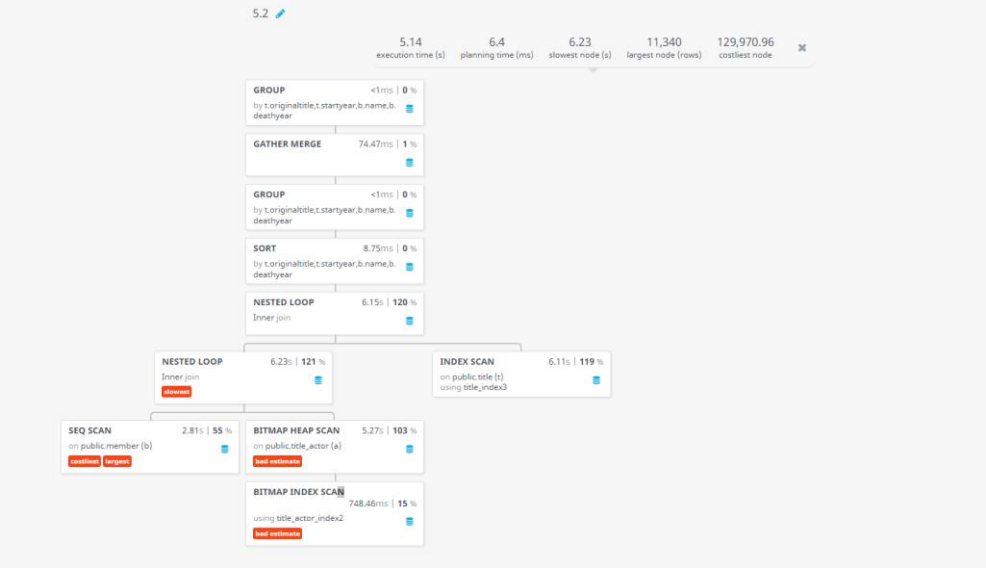
Below is the execution plans after indexing-

Query 1-



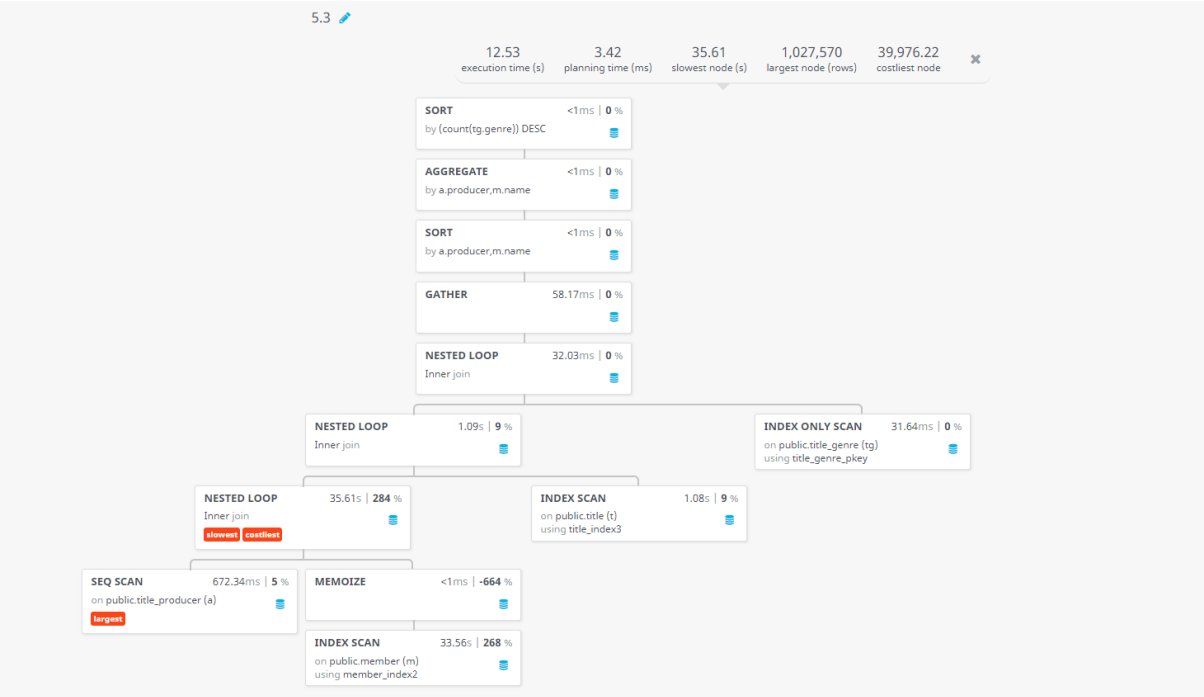
No major change was observed except reduction of time.

Query 2-



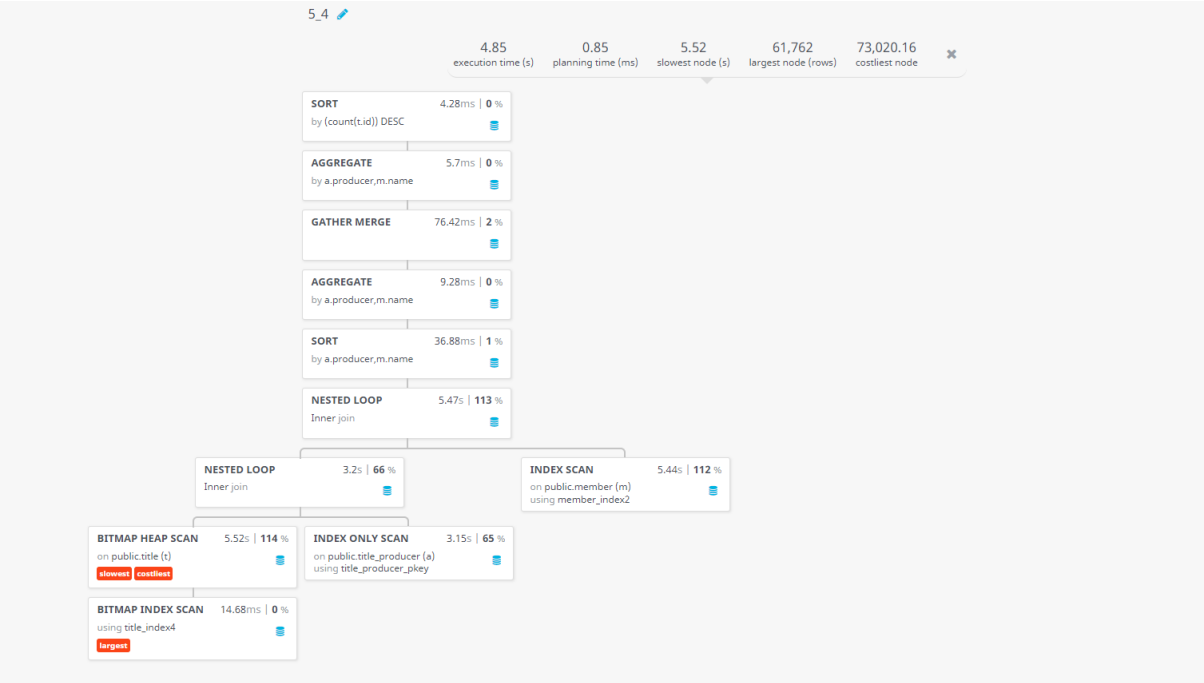
BitMap replaced the Seq scan and Hashing which was present without indexing

Query 3-



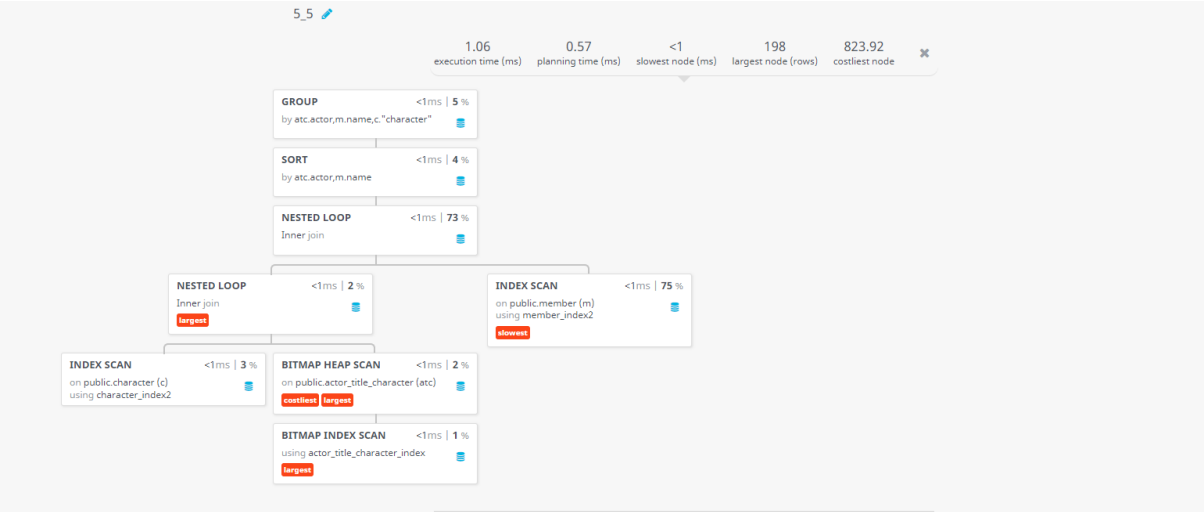
Slowest node’s time reduced by 10 seconds after indexing

Query 4-



BitMap Scans replaced Seq Scans on title table.

Query 5-



Bitmap scans which take less than 1 ms replaced seq scans which was taking 250 ms without indexing.