

HOMEWORK 2

- Anirudh Narayanan(an9425)

Q1. Provided a separate .txt file with queries and runtime alongside each of them.

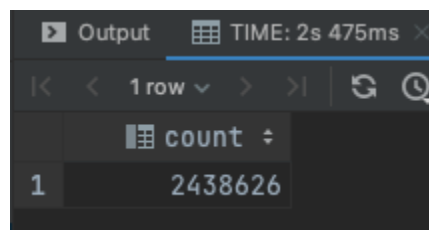
Q2.

2.1. Number of invalid Title_Actor relationships with respect to characters. (That is, entries in Title_Actor which do not appear in Actor_Title_Character.)

Query:

```
select count(*) from Title_Actor
  left join actor_title_character as atc on Title_Actor.actor = atc.actor and
Title_Actor.title = atc.title
where atc.characters is null;
```

Output:



The screenshot shows a database interface with a table titled 'Output'. The table has one row with the value 2438626. The table is named 'count' and the column is 'count'.

| Output | |
|--------|---------|
| 1 | 2438626 |

execution: 7 s 475 ms

2.2 Alive actors whose name starts with “Phi” and did not participate in any title in 2014.

Query:

```
SELECT Distinct name from

  ((select id, name, birthYear, deathYear from member where name like 'Phi%'
and deathYear is null ) as m

      join title_actor t on m.id = t.actor) as p

      left join ((select tid, original_title, startyear from title where startYear
!= '2014') as n

          join title_actor on n.tid = Title_actor.title) as pl

      on pl.tid = p.id ;
```

Output:

| | name |
|----|----------------------|
| 1 | Phi Bulani |
| 2 | Phi Clarke |
| 3 | Phi Hamens Nelson |
| 4 | Phi Hung Nguyen |
| 5 | Phi Huynh |
| 6 | Phi Lan |
| 7 | Phi Nguyen |
| 8 | Phi Palmos |
| 9 | Phi Tran |
| 10 | Phi Truong |
| 11 | Phi VoBa |
| 12 | Phi Vu |
| 13 | Phiasco |
| 14 | Phibs Everfresh Crew |
| 15 | Phichairak Phimonmat |
| 16 | Phichaphop Thongkuea |
| 17 | Phichet Iamchaonaa |
| 18 | Phichet Shaemsaitong |
| 19 | Phichit Putay |
| 20 | Phids Manisy |
| 21 | Phiekthor |
| 22 | Phiinix Jordann |
| 23 | Phiko Magongoma |
| 24 | Phil |
| 25 | Phil 'Skippy' Adams |
| 26 | Phil & John |
| 27 | Phil Aarrestad |
| 28 | Phil Abel |
| 29 | Phil Abernethy |
| 30 | Phil Abrams |
| 31 | Phil Acevedo |
| 32 | Phil Adams |

execution: 10 s 702 ms

2.3 Producers who have produced the most talk shows in 2017 and whose name contains "Gill". (Hint: talk show is a genre)

Query:

```
select counter.name, count(counter.title) as Counts
from (select prod.title, M.name
      from (select gen.title, tp.producer, gen.startYear, gen.genre
            from (select tg.title, t.startYear, tg.genre
                  from (select title, genre
                        from title_genre
                        where genre = (select id
                                      from Genre
                                      where genres = 'Talk-Show')) as tg
                  join title t on t.tid = tg.title
                  where startYear = '2017') as gen
            join Title_Producer tp on tp.title = gen.title) as prod
      join member M on M.ID = prod.producer
      where M.name like '%Gill%') as counter
group by Name
order by Counts desc;
```

Output:

| | name | counts |
|---|------------------|--------|
| 1 | Ryan Gill | 81 |
| 2 | Dominic Gillette | 73 |
| 3 | Corinne Gilliard | 14 |
| 4 | Shane Gill | 13 |
| 5 | Gilles Bérard | 1 |

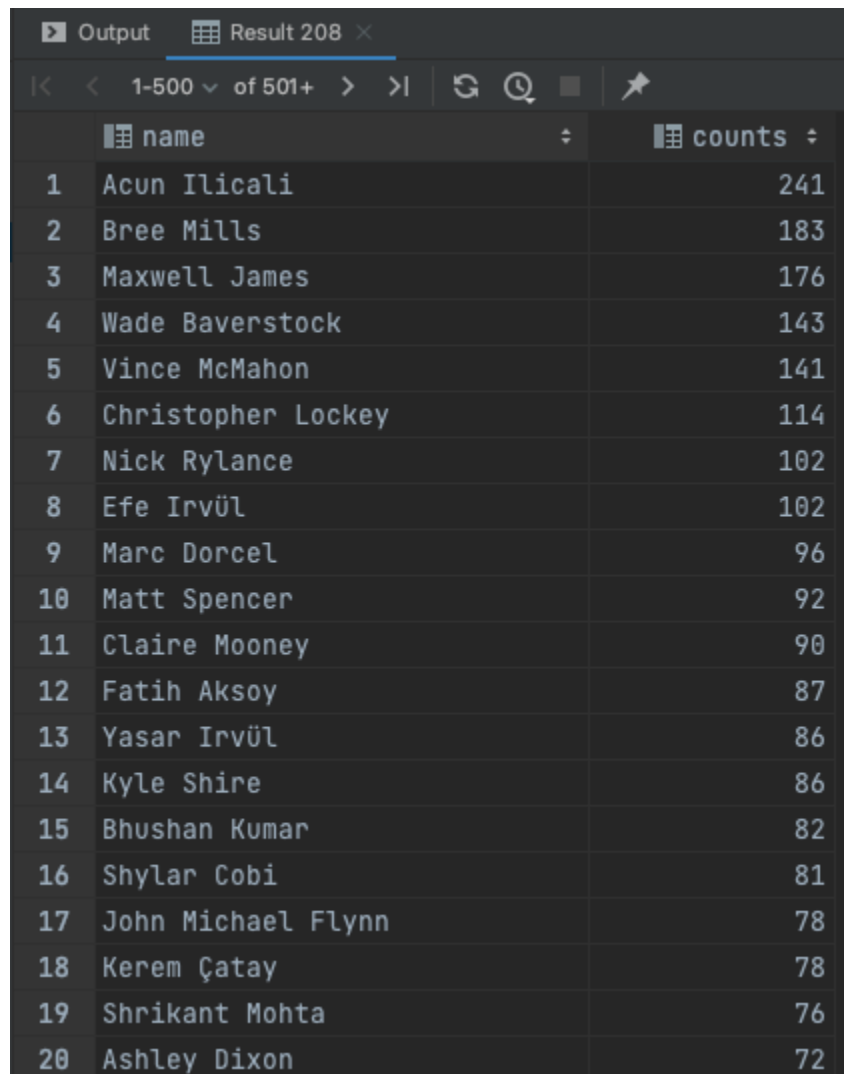
execution: 2 s 604 ms

2.4 Alive producers ordered by the greatest number of long-run titles produced (runtime greater than 120 minutes).

Query:

```
SELECT Name, count(runtimeMinutes) as counts
FROM (SELECT Name, T.runtimeMinutes
      FROM (SELECT Producer, M.name, title
            FROM Title_Producer P
              JOIN Member M on P.Producer = M.id
              WHERE M.deathYear IS NULL) AS pnt
      JOIN Title T on pnt.Title = T.tid
      WHERE cast(T.runtimeminutes as int) > 120) AS nr
GROUP BY Name
ORDER BY counts DESC;
```

Output:



The screenshot shows a database interface with a query result. The result is a table with two columns: 'name' and 'counts'. The table is sorted by 'counts' in descending order. The first 20 rows are visible, showing producers and their respective counts of long-run titles.

| | name | counts |
|----|--------------------|--------|
| 1 | Acun Ilıcalı | 241 |
| 2 | Bree Mills | 183 |
| 3 | Maxwell James | 176 |
| 4 | Wade Bayerstock | 143 |
| 5 | Vince McMahon | 141 |
| 6 | Christopher Lockey | 114 |
| 7 | Nick Rylance | 102 |
| 8 | Efe İrvöl | 102 |
| 9 | Marc Dorcel | 96 |
| 10 | Matt Spencer | 92 |
| 11 | Claire Mooney | 90 |
| 12 | Fatih Aksoy | 87 |
| 13 | Yasar İrvöl | 86 |
| 14 | Kyle Shire | 86 |
| 15 | Bhushan Kumar | 82 |
| 16 | Shylar Cobi | 81 |
| 17 | John Michael Flynn | 78 |
| 18 | Kerem Çatay | 78 |
| 19 | Shrikant Mohta | 76 |
| 20 | Ashley Dixon | 72 |

execution: 6 s 559 ms

2.5 Alive actors who have portrayed Jesus Christ (simply look for a character with this specific name).

Query:

```
SELECT id, name
FROM (SELECT DISTINCT(actor)
      FROM actor_title_character,
      (SELECT id FROM Characters where Characters.Characters like 'Jesus
Christ') AS C
      where actor_title_character.characters = C.id) AS actor
      JOIN member ON actor = member.id
WHERE deathYear is NULL;
```

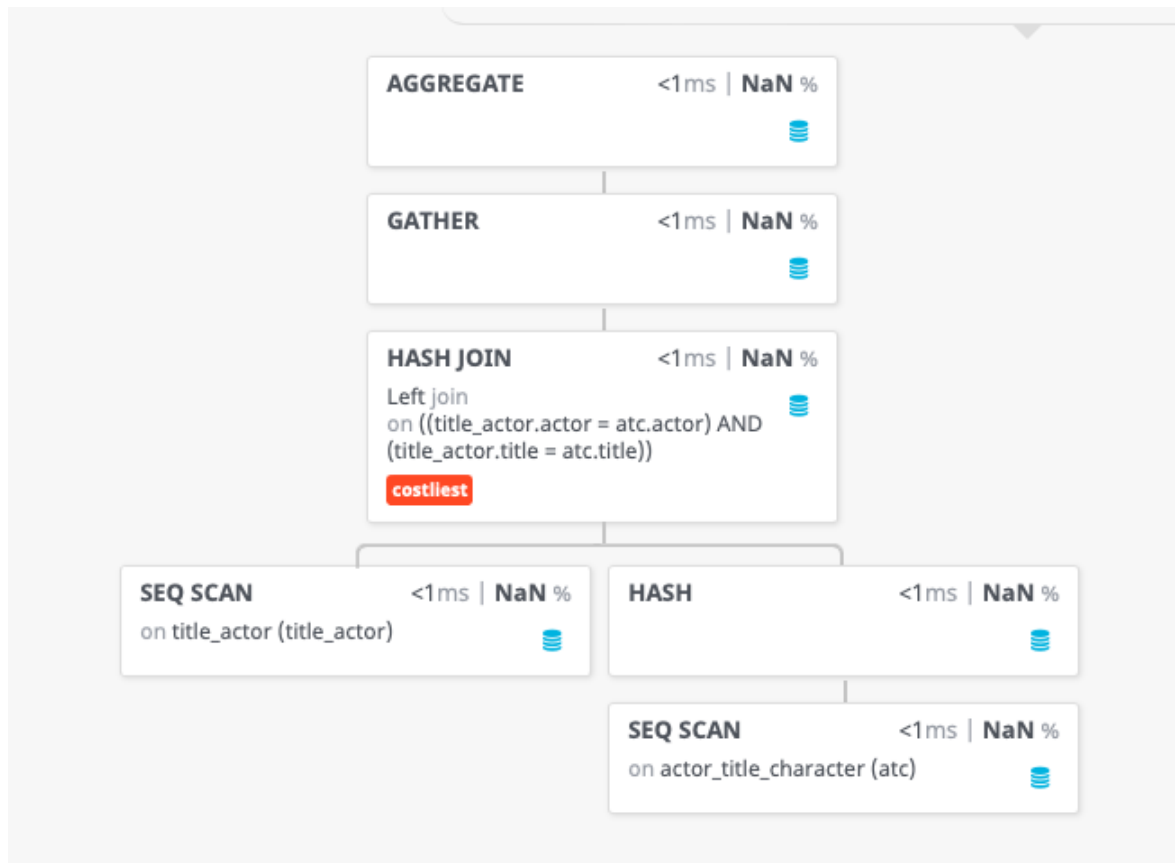
Output:

| |  id ▼ 1 |  name |  |
|----|--|--|---|
| 1 | 14178837 | Jesus Christ | |
| 2 | 12841076 | Joe Henry | |
| 3 | 12599237 | Elyesa Ersoy | |
| 4 | 12260401 | James Wire | |
| 5 | 12102620 | Oscar W. Fitchett | |
| 6 | 11701426 | Uncle Ho | |
| 7 | 11374992 | DavidJohn Wyatt | |
| 8 | 11365230 | Chrystian Tracey | |
| 9 | 11131047 | Fiverr Jesus | |
| 10 | 11104401 | Joe Sciammarella | |
| 11 | 11052773 | Rich Harris | |
| 12 | 10662353 | Kerollos Mathew | |
| 13 | 9926981 | Robinson Alteme | |
| 14 | 9857329 | Anthony Ohnesorgen | |
| 15 | 9299007 | Richard Lambert | |
| 16 | 9208425 | Marco WestWood Gonzalez | |
| 17 | 8991585 | Nick Owens | |
| 18 | 8903025 | Luke Dimyan | |
| 19 | 8734810 | Paolo Vanoli | |
| 20 | 8714396 | John Keller | |
| 21 | 8652775 | Karlos Granada | |
| 22 | 8509764 | Bob DeBartolo | |

execution: 837 ms

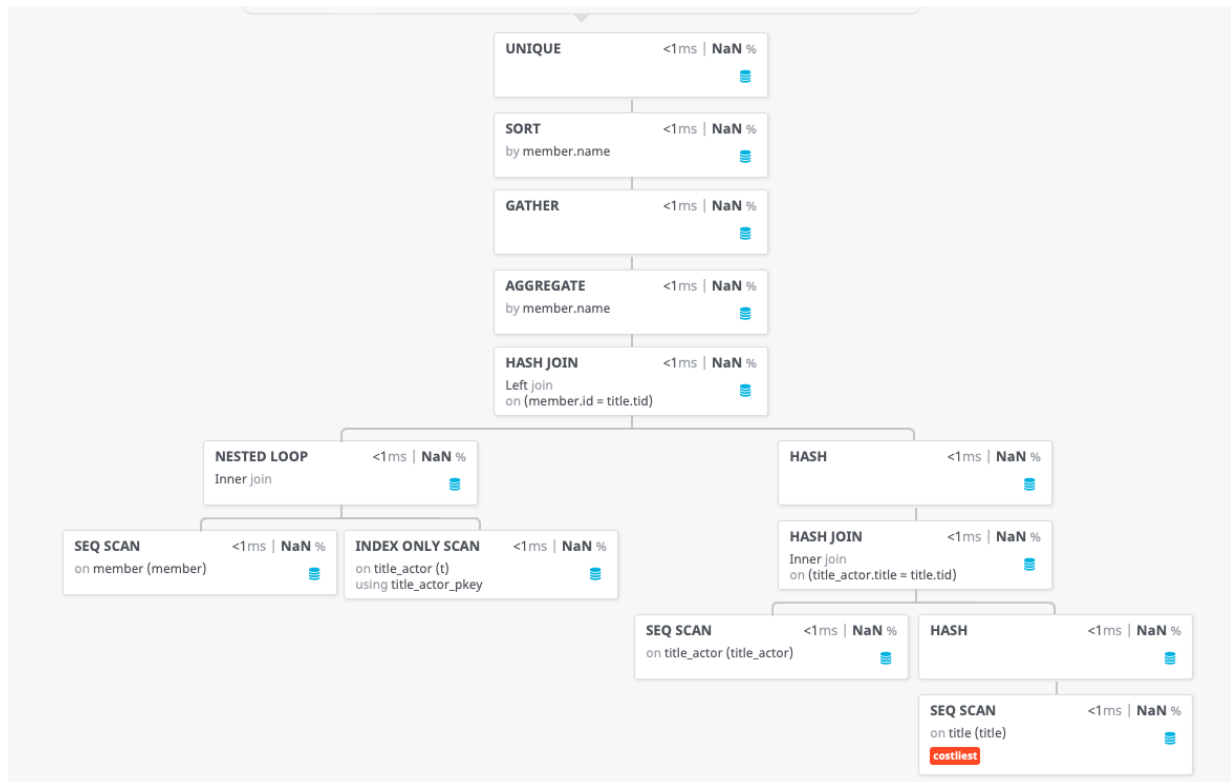
3.

3.1



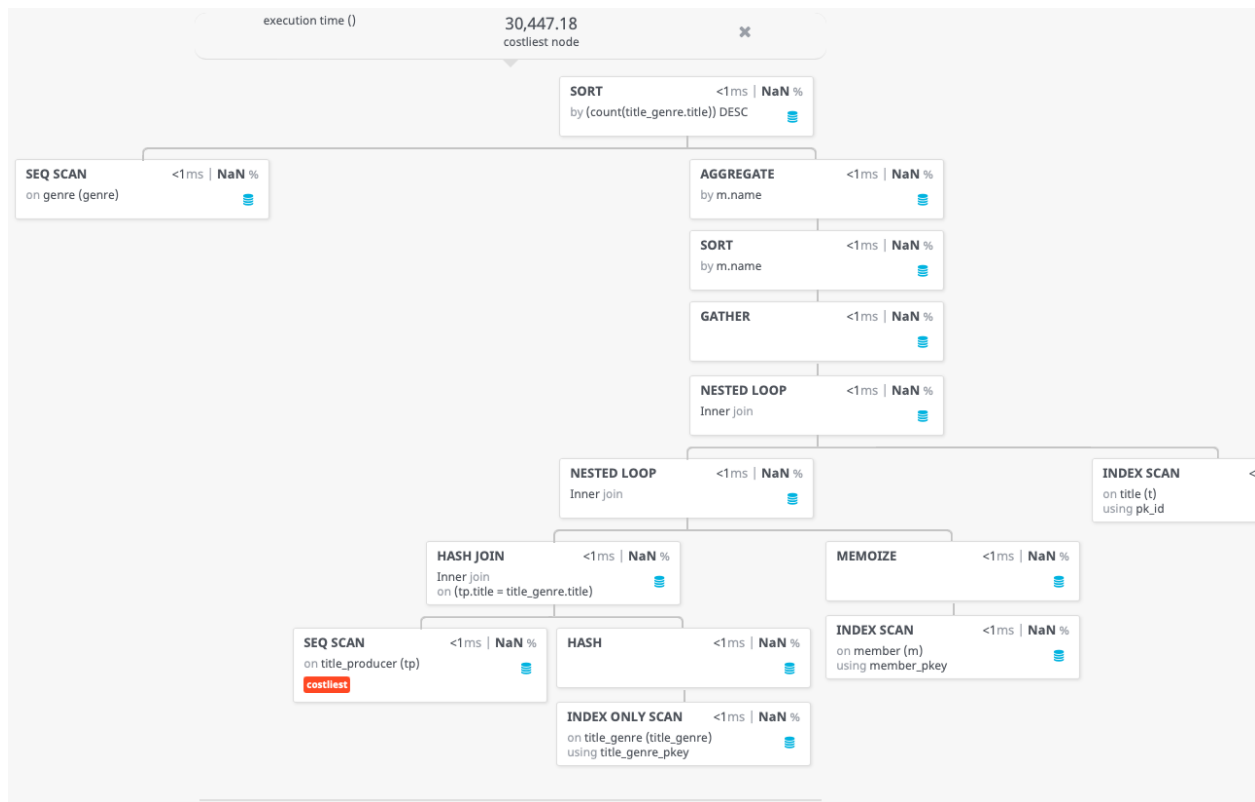
We start with a sequential scan of the atc table, then hash it, then we perform a sequential scan of the title_actor table and then join it with the table after hashing using hash join. Then the result is gathered and aggregated.

3.2



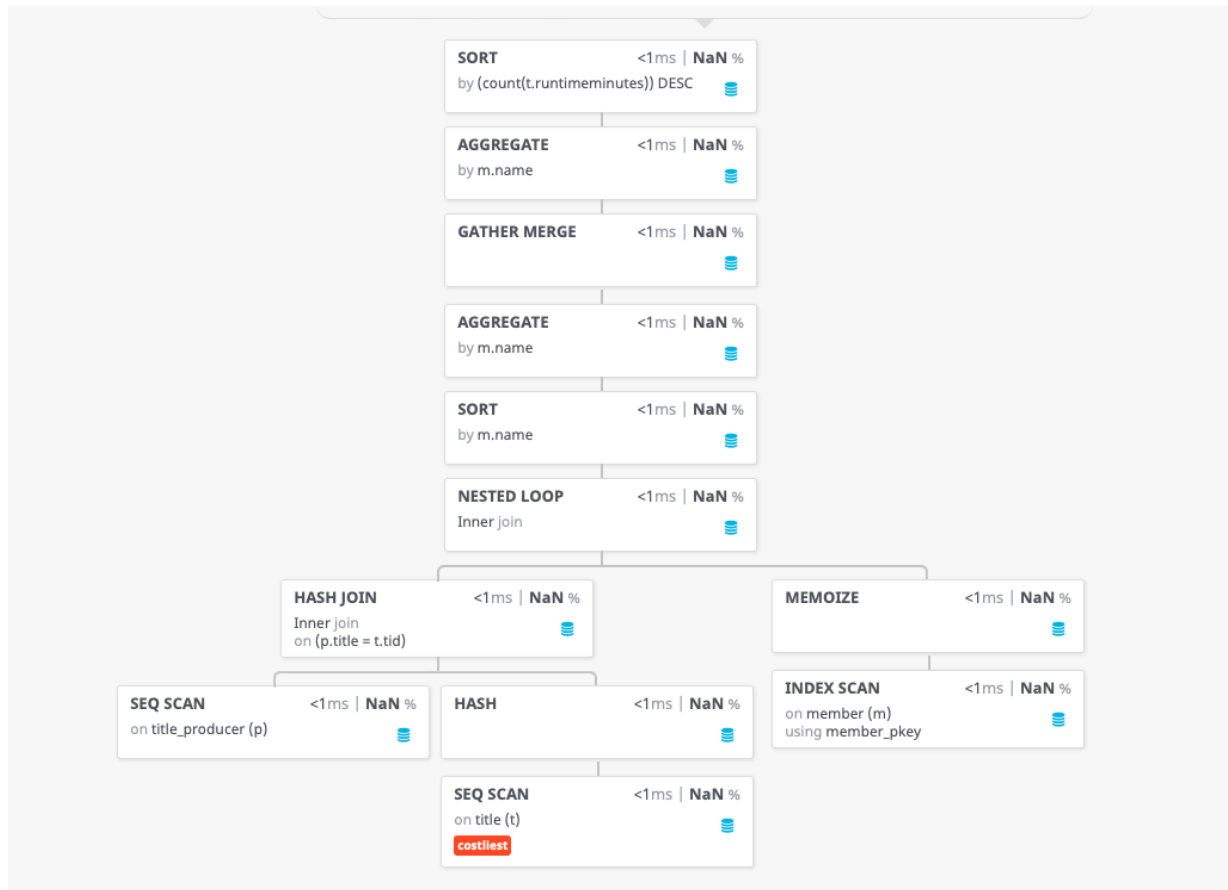
We start with a seq scan of the title table, then hash it, then seq scan the title_actor table and join with hash join, then we perform seq scan of the member table and index scan of the title_actor_pkkey in the nested loop. Then we join this table and the previously joined table. Then we aggregate, gather , sort and choose unique values.

3.3



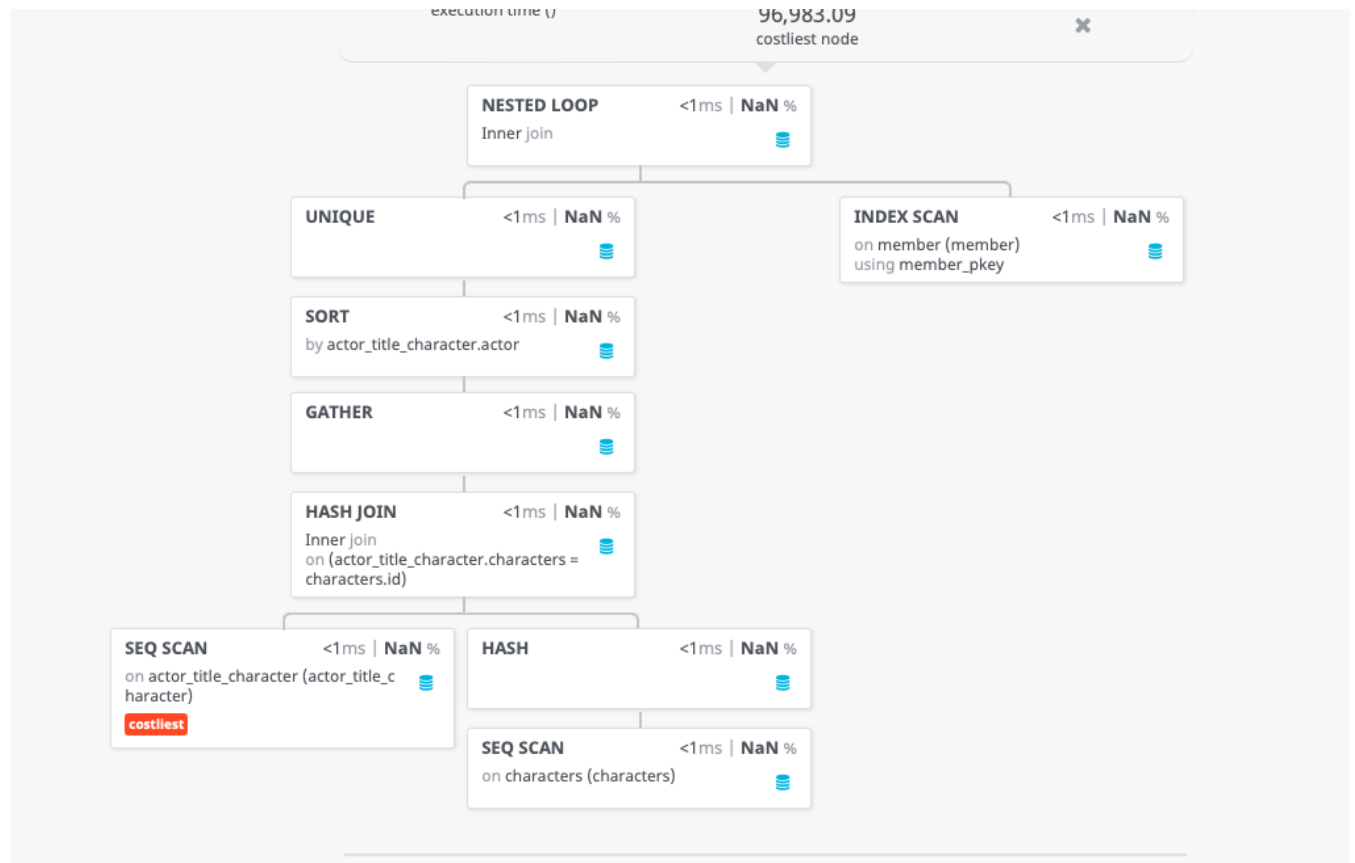
First, we perform an index only scan on the title_genre primary key, then hash it. Then we perform a seq scan on the title_producer then hash join it with the previous table in a nested loop. This result is then joined with members and then merged with the index scan of the title primary key. Then it is gathered, sorted and aggregated. Then we perform a seq scan of the genre and merge them and sort by count in descending order.

3.4



We start off with a seq scan of the title and then hash it. Then we perform a seq scan of the title_producer and join it with the previous table using hash join. Then we perform an index scan of the member table primary key and store . Then we sort, aggregate, gather, sort by count(runtime) in descending order.

3.5



We start off with a seq scan of the character and then hash it, then we perform a seq scan on the actor_title_character table and hash join it with the previous table. Then we gather, sort and choose unique values from the table. Then we perform an index scan of the member table primary key. The results are merged.

Q4.

4.1, 4.2

Q4.1]

$$\pi(\text{count}(*)) (\sigma_{\text{atc.character} = \text{Null}} (\text{Title_Actor} \bowtie \text{atc} \begin{matrix} \text{atc.actor} = \text{title_actor.actor} \\ \text{atc.title} = \text{title_actor.title} \end{matrix}))$$

Q4.2]

$$\rho_p(\pi_{\text{name}}) \rho_m(\pi_{\text{id, name, birthyear, deathyear}} (\sigma_{\text{name like 'Phiz'} \wedge \text{deathyear is Null}}) \bowtie \rho_t(\text{title_actor}) \begin{matrix} \text{m.id} = \text{t.actor} \end{matrix})$$

$$\bowtie \begin{matrix} \text{p.id} = \text{pl.tid} \end{matrix}$$

$$\rho_p(\rho_n(\pi_{\text{t.tid, t.title, t.startyear}} (\sigma_{\text{startyear} \neq 2014}) \bowtie \text{title_actor}) \begin{matrix} \text{n.id} = \text{title_actor.title} \end{matrix})$$

Q 4.3

σ_{name} like "%GILL%" (σ_{gen} (title, startyear, genre))

$C_{\text{startyear}='2017'}(S_{\text{ty}}(\text{title, genre}(C_{\text{genres}='talkshow'})))$

$\Delta \text{ title } (t) \quad \Delta \text{ tp} \quad \Delta \text{ m}$
 $t.\text{id} = t.g.\text{title} \quad t.\text{title} = \text{gen.\text{title}} \quad m.\text{id} = \text{prod.\text{producer}}$

$$\pi_{\text{name}, \sum_{\text{count}} (\text{count}(\text{sumtime}))} \left(\sum_n (\text{sumtime}) \right)$$

$(\pi_{name, runtime} (\rho_{pt} (producer, name, title)))$

C & deathyear is NULL \bowtie Title +
pt. title (where runtime ≤ 120)
T. tid

4.5

Q4.5)

$\pi_{id, name} \left(\sigma_{Cid = c} \left(\sigma_{id = 'id'} \right) \right)$
actor c character = 'Jesus christ'

member

member.id = actor

where deathyear = Null

5.

Query1: We have created the index on (actor, title.)

```
create index "index" on title_actor(actor, title);  
-- IMPROVED RUNTIME: 5s 232ms
```

Query2: We have indexed startyear, so that we can search through it faster.

```
create index "index1" on title(startyear);  
--IMPROVED RUNTIME: 8s 366ms
```

Query3: We have indexed the genre, so that we can search through it faster.

```
create index "index2" on title_genre(genre);  
--IMPROVED RUNTIME: 2s 234ms
```

Query4: We have indexed deathyear, so that we can search through it faster.

```
create index "index3" on member(deathYear);  
--IMPROVED RUNTIME: 5s 312ms
```

Query5: We have indexed character, so that we can search through it faster.

```
create index "index4" on characters(characters);  
--IMPROVED RUNTIME: 577ms
```