Assignment Number 2
-Deepika Kini

Q.1 please check comments below

- **Title (id, type, title, originalTitle, startYear, endYear, runtime, avgRating, numVotes)**
  Preprocessing: no changes to primary key(unlike previous assignment)
  Joined with ratings table to get extra columns
  File: preprocess.py
  Time for preprocessing and loading: 5 mins
- **Genre (id, genre)**
  Preprocessing: Take unique genres from title and break it down, add a unique identifier
  Time for preprocessing and loading: 1 mins
- **Title_Genre (genre, title)**
  - **genre FK Genre(id)**
  - **title FK Title(id)**

  Preprocessing: map title id with many genre ids(break into many rows using explode
  Time for preprocessing and loading: 2 mins

- **Member (id, name, birthYear, deathYear)**
  Preprocessing:Used name.basics file
  Time for preprocessing and loading: 3 mins

- **Title_Actor (actor, title)**
  - **actor FK Member(id)**
  - **title FK Title(id)**
- **Title_Producer (producer, title)**
  - **producer FK Member(id)**
  - **title FK Title(id)**

  Preprocessing:Used principals file to filter on actor/actress/producer category
  Time for preprocessing and loading: 55 mins

- **Title_Director (director, title)**
  - **director FK Member(id)**
  - **title FK Title(id)**
- **Title_Writer (writer, title)**
  - **writer FK Member(id)**
  - **title FK Title(id)**

  Preprocessing:Used crew file to get columns of writer and director and then spilt them into rows(contained arrays of ids)
  Time for preprocessing and loading: 45 mins

- **Character (id, character)**

Note: this table should contain individual characters

Tables are created in python(unlike my previous implementation of hw1) . here i've use `psycopg2 library`

- **Actor_Title_Character (actor, title, character)**
    - **(actor,title) FK Title_Actor (actor, title)**
    - **character FK Character(id)**

Preprocessing: Used explode to create these tables
Time for preprocessing and loading: 35 mins

Note: For rows in title_actor, … they should be present in both title and principal files. Hence inner joins are run

Tables are created in python(unlike my previous implementation of hw1) . here i've use `psycopg2 library`

Below queries in psql used to load data quicker:(due to slowness later on)

\copy producer(title, producer) FROM '/Users/deepika/Desktop/CSCI620/Assignment_1/IMDB/ModifiedData/producer_mapping_project2.csv'(format csv, delimiter ',', header true);

\copy ACTOR(title, actor) FROM '/Users/deepika/Desktop/CSCI620/Assignment_1/IMDB/ModifiedData/actor_mapping_project2.csv'(format csv, delimiter ',', header true);
Time: 10mins

Wasn't able to load director and writer table(with primary keys applied) since they had duplicates in primary keys which I wasn't able to handle in python.
RUNTIME: 4.25 hours approx

Loading using copy command took time hence pushed data first into sql and added constraints:
SQL:

```
create table atc
(
title varchar(20),
```

```
	actor varchar(20),
character integer
	)

insert into atc(
select actor,title,character from actor_title_character)



create table actor_title_character1(title, actor, character) as
select * from actor_t_c group by title, actor, id

Alter table a_t_c ADD PRIMARY KEY(id, ACTOR, TITLE) ;


Alter table a_t_c ADD FOREIGN KEY(character) REFERENCES Character(id);

Not run:
Alter table a_t_c ADD FOREIGN KEY(TITLE, ACTOR) REFERENCES title_ACTOR(TITLE,
ACTOR);  since extra rows present in this table
(wasn't able to disable disable constraints using the code:
sql = "set constraints all deferred;"
cursor.execute(sql)
conn.commit() )
```

Please refer to main_assignment2_preprocessing.py and hw2.sql for preprocessing and create table codes. Other python files (other than queries_script.py) are used for preprocessing

—----------

Q.2.

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.)

select title, actor from title_actor where concat(title, actor) not in (select concat(title, actor) from atc)

No outcome, just runs

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

```
select m.name, t.title, t.startYear from title_actor a
inner join member m on a.actor = m.id
inner join title t on a.title = t.id
where m.deathYear = 0 and t.startYear != 2014 and endYear != 2014
and lower(m.name) like 'phi%';
```

Time: 19.2sec

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

```
select m.name, count(t.id) from member m
join title_producer p on m.id = p.producer
        join title t on t.id = p.title
        join title_genre tg on tg.title = t.id
    join genre g on tg.genre = g.id
    where m.name like '%gill%' and t.startYear = 2017 and lower(g.genre) like '%talk-show%'
    group by m.name
    order by count(t.id) desc
```

Time:21 secs

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

```
 select m.name, count(t.title)
from title_producer p
join member m on p.producer = m.id
join title t on t.id = p.title
where t.runtime > 120 and m.deathYear != 0
group by m.name
order by count(t.title) desc
```

Time :23 secs

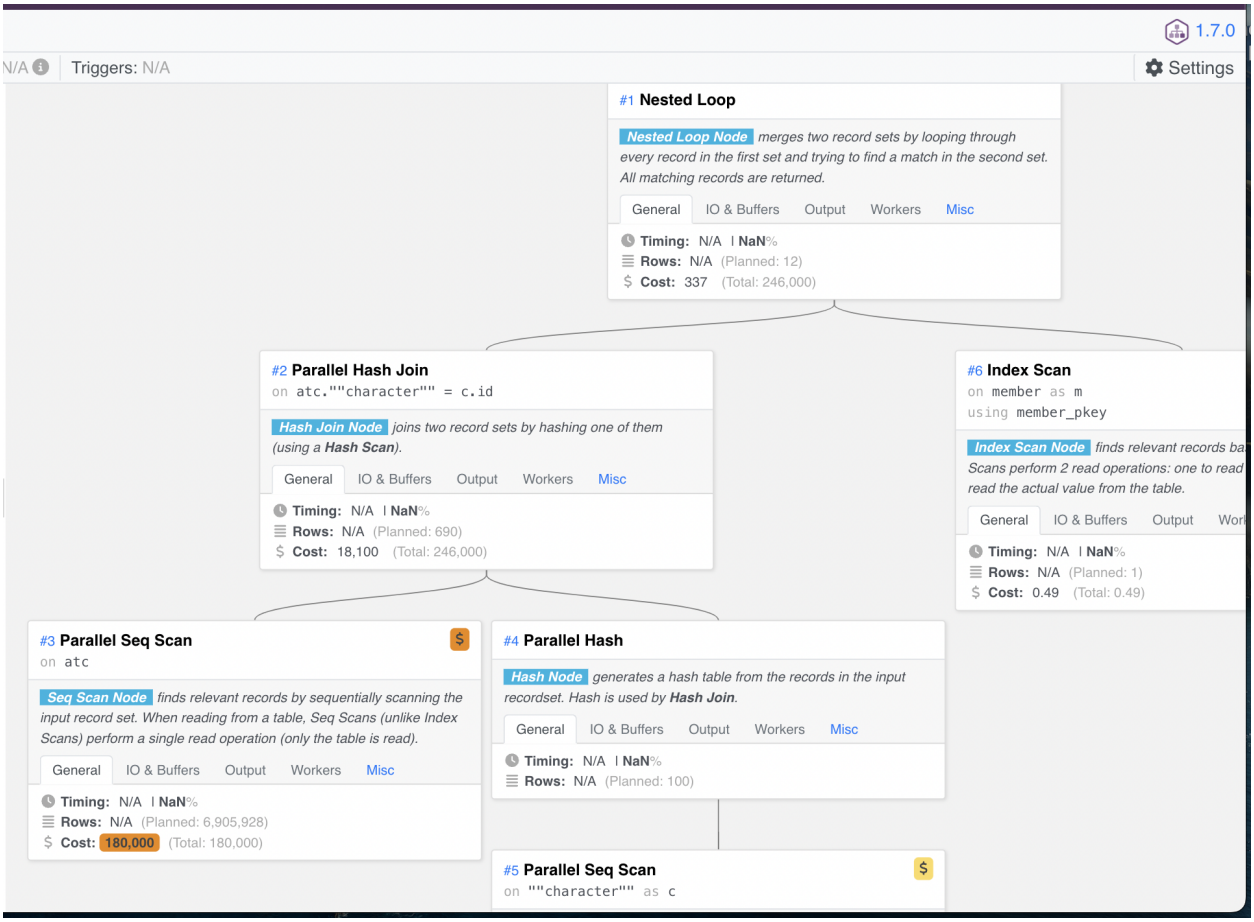
2.5. Alive actors who have portrayed Jesus Christ (simply look for a character with this specific name).

select m.name, atc.character, c.character from member m
join atc atc on atc.actor = m.id
join character c on c.id = atc.character
where lower(c.character) like '%jesus christ%' and m.deathYear != 0
Time: 1min 20 sec

Q. 3.

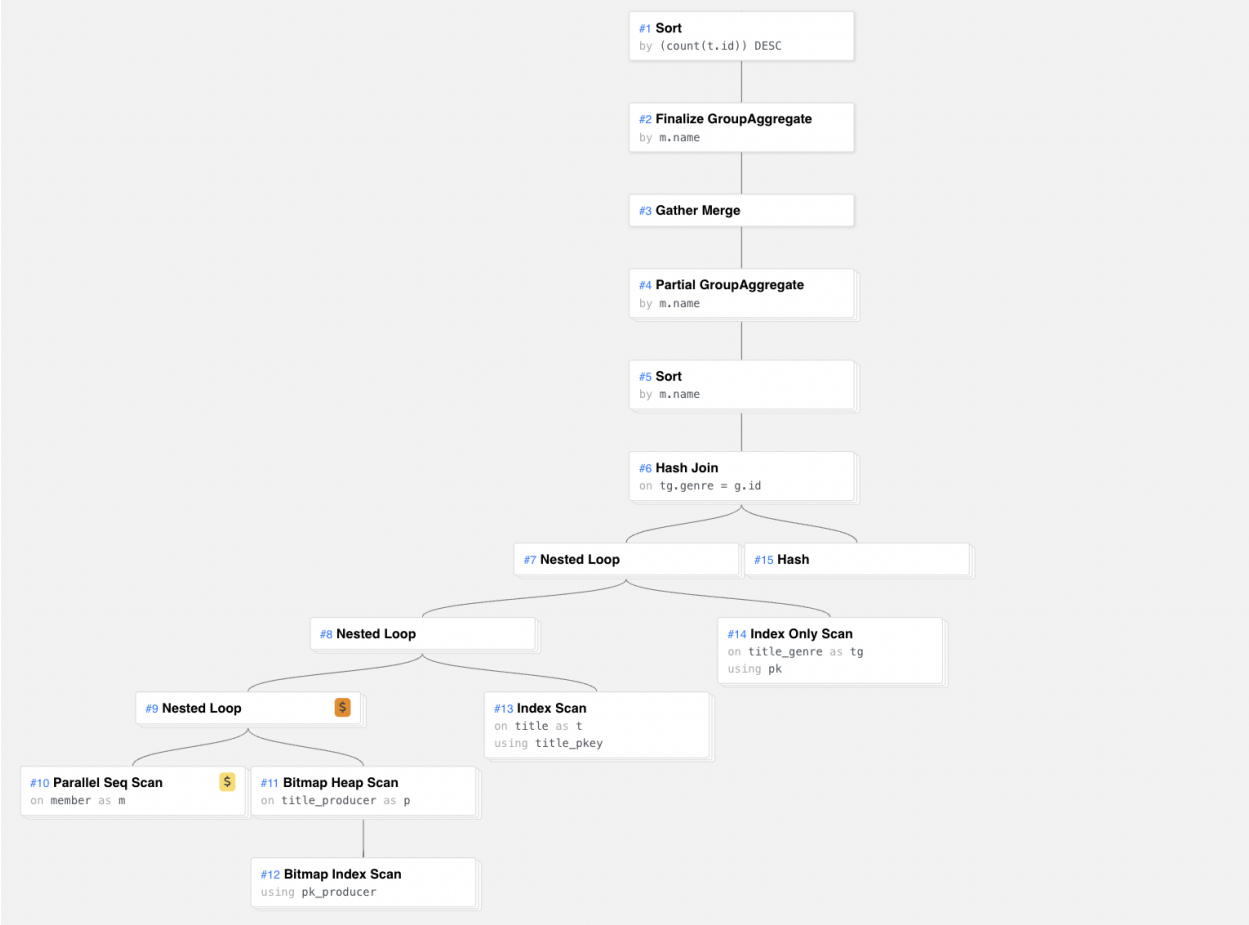
2.1.query
No outcome

2.2



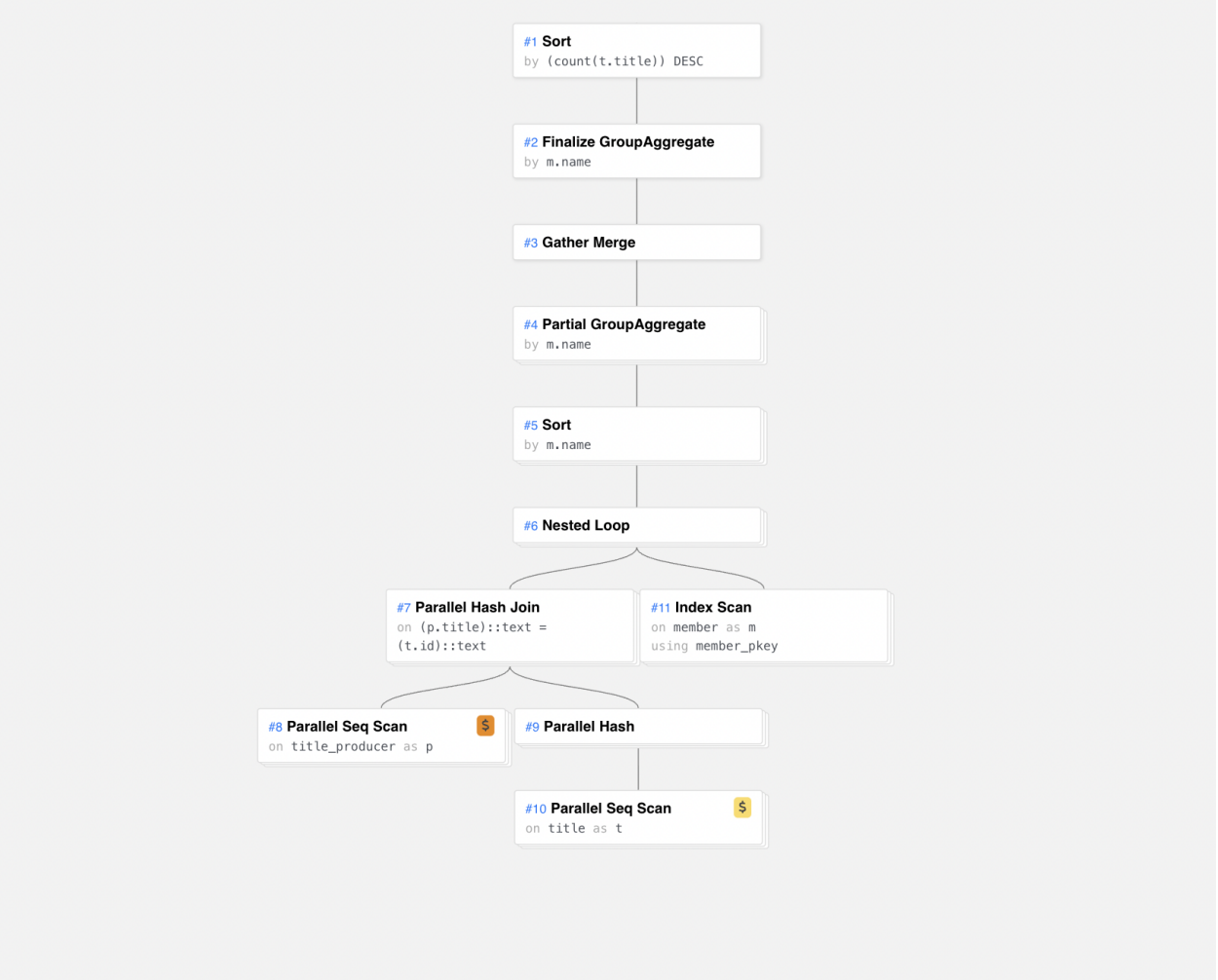
The nested loop is for joins. Member table uses table to filter out year information
Index of title_actor is used to filter out on title_actor for join.
Table scan is basically seq scan and index scan uses primary key as index
Explaining tree structure:
member(pulled by data scan) and title_Actor(scanned on index) join is carried out in the leaf node of tree and nested loop is applied to check on equal values.(line 1 of sql)
Similarly the outcome of this is combined with a join on title_key (line 2 of sql) which gives the 2nd level data. At the end the data is gathered (the columns are chosen)

2.3



Bitmap index filters data based on index of the table. Similar to above query outcome

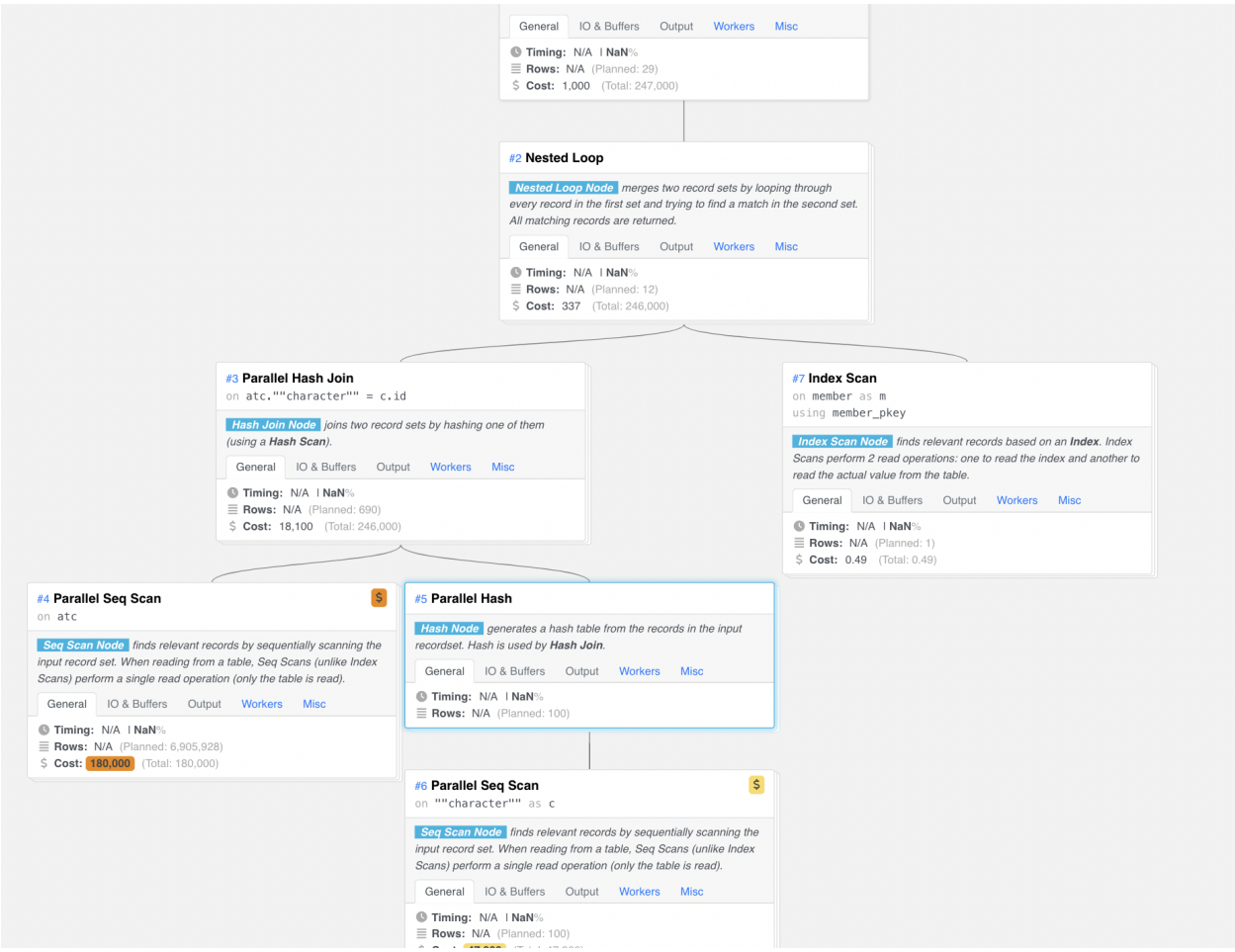
2.4



Where and join takes place first, then aggregate works, and then sort (based on count). For **text columns, hash** is used as seen above. M.name is sorted first and then applied the filter to

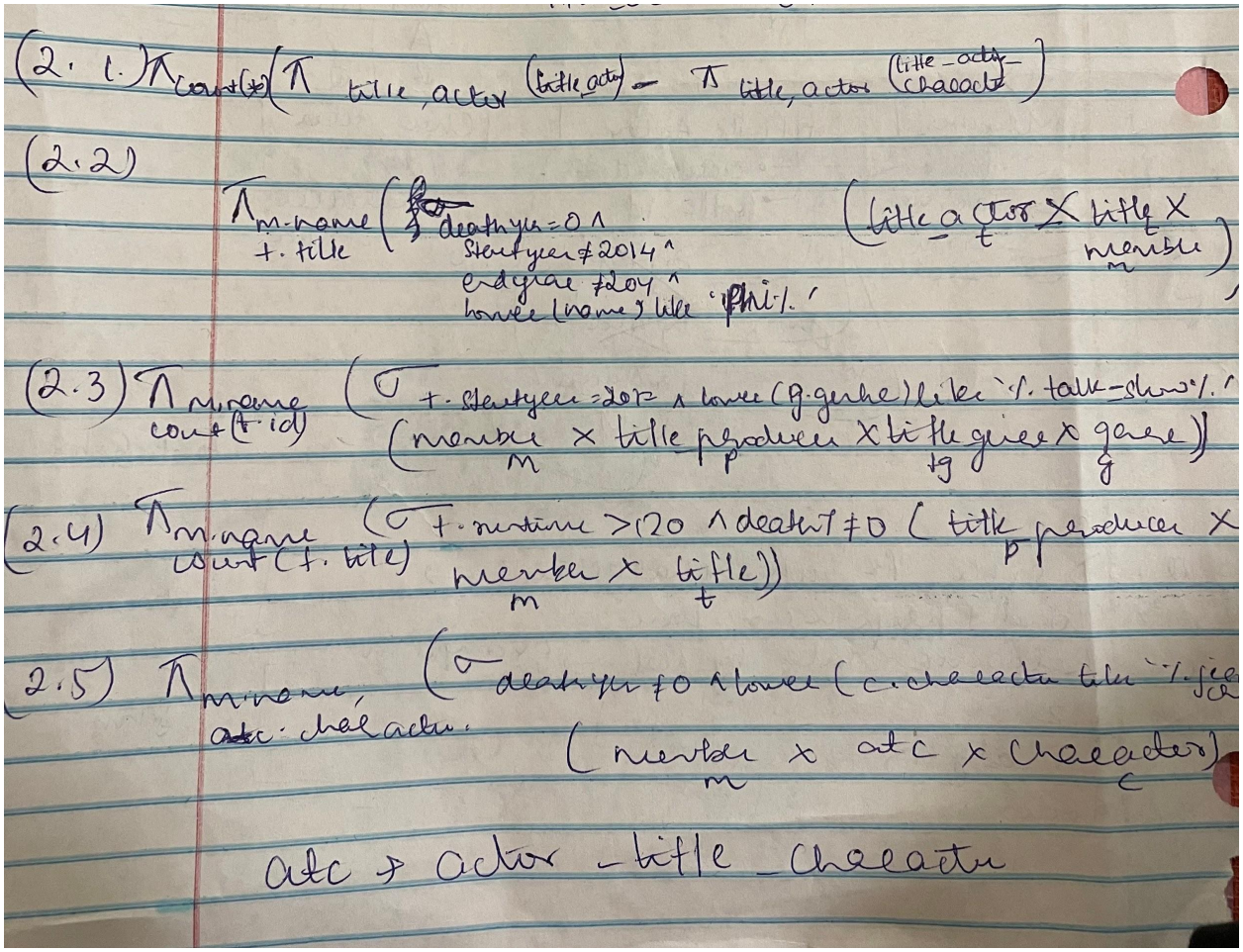
2.5

GATHER

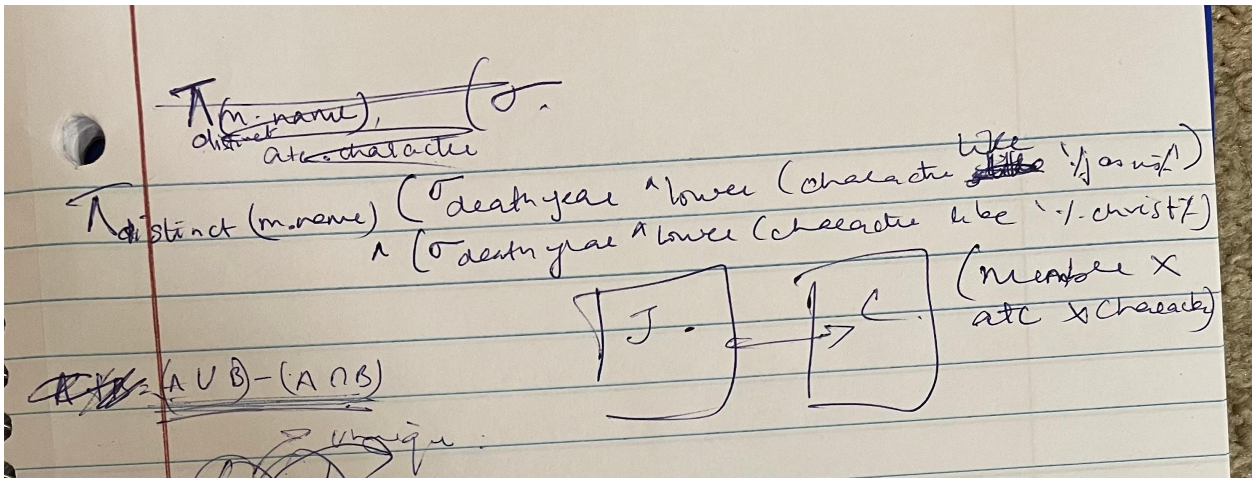


Q.4.

Below are the relational algebra queries corresponding to sql queries:



2.5 with different jesus and christ terms:



Q.5

Applied index to columns which are used frequently in either join, where or select clauses. Q2, 3 ,4 and 5 run faster when indexed manually

Query 2.1:

select title, actor from title_actor where concat(title, actor) not in (select concat(title, actor) from atc)

Taking time to run

Query 2.2:

name	title	startyear
Philip Martin Brown	The Birdman	1994
Philip Bretherton	The Car Washer	2015
Philippa Baker	Episode #1.97	1972
Philipp Seiser	Die Beförderung	1983
Philipp Stadler	Mirakel	1990
Phill Jupitus	The Great Water	2000
Phil Brooks	Episode #10.53	2009
Philip Hawthorn	Episode #5.28	2000
Phillip J. Roth	Chimes of Freedom	2019

Time: 12.2 sec

Query 2.3:

output:

"Name" "count"
"Robert Shergill" 1

Time: 3 seconds

Query 2.4 :

Sample rows:

"name" "count"

"Aachi Manorama" 172
"N.T. Rama Rao" 164
"Nagesh" 157
"Shivaji Ganesan" 141

Time: 17.9 sec

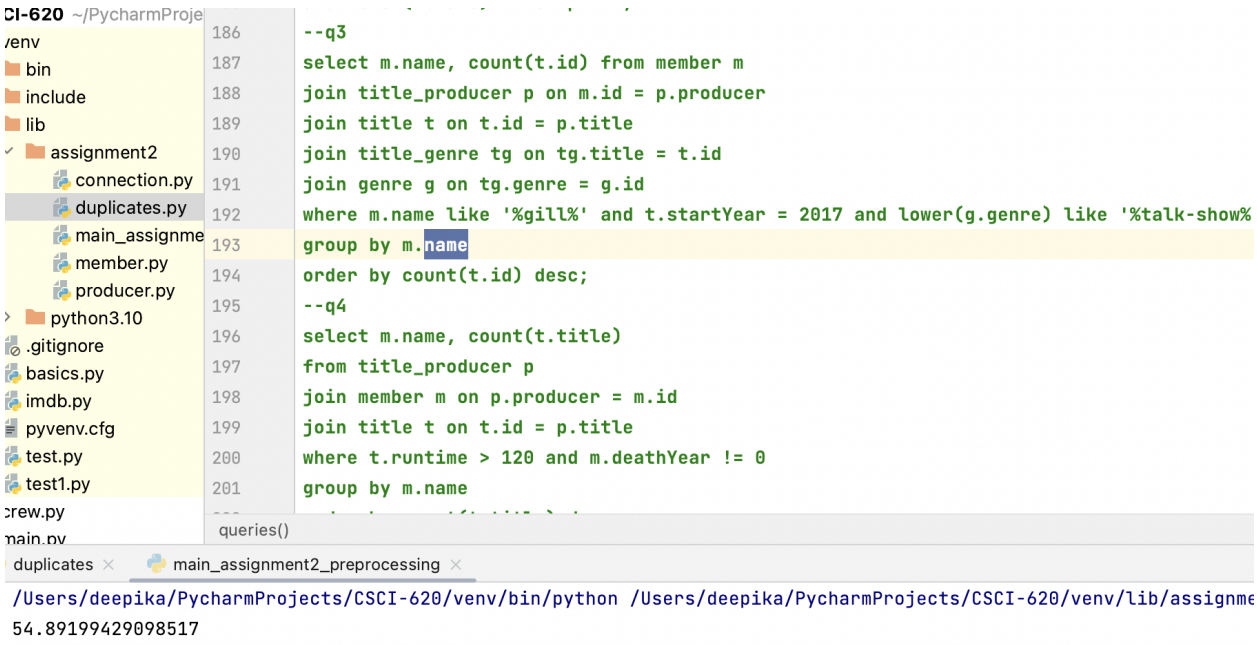
Query 2.5:

Time: 7.3 seconds

Sample rows:

name	*character*	*character-2*
Claudio Brook	*1598433*	*["Jesucristo (Jesus Christ)"]*
Jon Shepodd	*228330*	*["Jesus Christ"]*
Nelson Leigh	*228330*	*["Jesus Christ"]*
Peter Chown	*228330*	*["Jesus Christ"]*
Arsenio Corsellas	*228330*	*["Jesus Christ"]*
Hans-Reinhard Müller	*988461*	*["Jesus Christus"]*
Michael Gwynn	*228330*	*["Jesus Christ"]*
Sydney Ayres	*228330*	*["Jesus Christ"]*
Nelson Leigh	*1248861*	*["Jesus","Jesus Christ"]*
Nelson Leigh	*228330*	*["Jesus Christ"]*

When run in python code: queries_script.py, it takes 54 secs for indexed data and 72 sec for non indexed (not manually indexed by me) data



Index sql statements:(in **hw2.sql file)**

create index q2 on member(name)

create index q3 on title(id, endYear, startYear)

create index q4 on genre(genre)

create index q5 on title_actor(title, actor)

create index q6 on character(id, character)

create index q7 on actor_title_character(actor, character)

—----------END—----------