# HOMEWORK 03:

- Anirudh Narayanan(an9425)

Q1.

```
CREATE TABLE Q1(
   movieId varchar,
   type varchar,
   startYear varchar,
   runtime varchar,
   avgRating float,
   genreId varchar,
   genres varchar,
   memberId varchar,
   birthYear varchar,
   character varchar
);
drop table Q1;

INSERT INTO Q1 (movieId, type, startYear, runtime, avgRating, genreId, genres,
memberId, birthYear, character)
SELECT t.tid, t.titletype, t.startyear, t.runtimeminutes, t.averagerating,
tg.genre, g.genres, m.id, m.birthyear, atc2.characters
FROM title AS t
JOIN title_genre AS tg ON t.tid = tg.title
JOIN genre AS g ON g.id = tg.genre
JOIN (
   SELECT at.actor, at.title, atc.characters
   FROM (
       SELECT actor, title, count(*) AS count
       FROM title_actor
       GROUP BY actor, title
       HAVING count(*) = 1
   ) AS at
   JOIN actor_title_character AS atc ON at.title = atc.title AND at.actor =
atc.actor
) AS atc2 ON t.tid = atc2.title
JOIN member AS m ON atc2.actor = m.id
WHERE CAST(t.runtimeminutes AS INT) >= 90 AND titletype = 'movie';
```

Run SQL query to create Q1 table with combined information.

Q2.

Run Q2.py.

Runtime:
Ran the program for approximately an hour.
Expect the program to complete in 4 to 5 hours as it has completed about 20% of the possible dependency check. Used print statement to keep track.

Q3.
Run Q3.py

Used pruning approach
Stopped check if any of the following conditions were met:

IF A → B, B →C, C→D, then conclusion : A → C
IF A → B, B →C, C→D, then conclusion : (A,B) → D also satisfied.

Dependencies found:
'movieid' → 'type,startyear,runtime,avgrating',
'Genreid' → 'genres', 'memberid': 'birthyear',
'movieid, type' → 'type,startyear,runtime,avgrating,

Here are the dependencies found that I am using in Q5.

Q4.

If we do not include the condition for characters == 1 per title, then the joined table would contain a lot more columns and each movieID will have multiple character values and finding functional dependencies will take much longer as there will be duplicates in the dataset formed, which causes data redundancy. This will cause update anomalies.

Q5.
Candidate Keys: movieID, GenreID, memberID

Canonical Cover:

'movieid, type' -> 'startyear, runtime, avgrating’
'Genreid' -> 'genres'
'memberid' -> 'birthyear, character'

Final Decomposition of Q1 Query: (Primary Key **- BOLD**)

Table1:
movie_info(**movieid, type,** startyear, runtime, avgrating)

Table2:
member_info(**memberid,** birthyear, character)

Table3:
genre_info(**genreid**, genres)

Table4:
movie_member(**movieid, memberid**) # For relation between movie_info and member_info

Table5:
movie_genre(**movieid, genreid**) # For relation between movie_info and genre_info