Homework 3

Anikhet Mulky

am9559@g.rit.edu

Q1)

Using the tables from Assignment 2

```
CREATE table movie_genre as(

SELECT Title,"Title_Genre".genre_id

FROM Title

Join "Title_Genre" on "Title_Genre".id = title_main.id

)

CREATE table movie_genre_id as(

SELECT movie_genre.*,"Genre".genre

FROM movie_genre

Join "Genre" on movieandgenre.genre_id = "Genre".genre_id

)

CREATE table movie_genre_id_member as

(SELECT
movie_genre_id.*,Title_Actor_Character.member_id,Title_Actor_Character.characters

FROM Title_Actor_Character

Join movie_genre_id on movie_genre_ide.id = Title_Actor_Character.id

where SELECT member_id

FROM final_character

GROUP BY member_id

HAVING COUNT(*) = 1
```

```
);

CREATE table Movie as

(SELECT movie_genre_id_member.*, "Member"."birthYear"

FROM movie_genre_id_member

join "Member" on "Member".id = movie_genre_id_member.member_id

where "runtimeMinutes" >= 90

)
```

To incorporate all the necessary columns, multiple join commands were necessary and instead of incorporating inside a single query, multiple tables + join statements were used. q1.sql is attached.

Q2)

movie_id —> type

movie_id —> startYear

movie_id —> runtime

movie_id —> avgRating

movie_id —> genre

movie_id —> genre_id

(Every combination of these)

movie_id —> type, startYear, runtime, avgRating , genre , genre_id

member_id ----> movie_id

member_id ----> type

member_id ----> startYear

member_id ----> runtimeMinutes

member_id ----> averageRating

member_id ----> characters

member_id ----> birthYear

(Every combination of these)

member_id ----> movie_id, type, startYear, runtimeMinutes, averageRating, characters, birthYear

genre —> genre_id

etc…..

Q3)

Functional dependencies using a naive (brute force) approach.

```
repeat_detect = {}

for indexMain in range(len(column_set)):
    for index in range(1, len(column_set)):
        for club in itertools.combinations(column_set, r=index):
            attribute_on_the_left = df.loc[:, column_set[indexMain]]
            attribute_on_the_left = attribute_on_the_left.drop_duplicates()
            convert_to_list = list(club)
            attribute_on_the_right = df.loc[:, convert_to_list]
            attribute_on_the_right = attribute_on_the_right.drop_duplicates()
            right_temp = attribute_on_the_right.columns[0]

            count_of_left = attribute_on_the_left.count()

            count_of_right = attribute_on_the_right[right_temp].count()
            element_index = 0
            if count_of_right == count_of_left:
                if club in repeat_detect:
                    break
                repeat_detect[club] = repeat_detect.get(club,0)
                if column_set[indexMain] in convert_to_list:
                    element_index = convert_to_list.index(column_set[indexMain])
                    convert_to_list.pop(element_index)
                if not convert_to_list:
                    break
                print(str(column_set[indexMain]) + " ----> " + str(tuple(convert_to_list)))
```

```
/Users/anikhetmulky/PycharmProjects/pythonProject2/venv/bin/python /Users/anikhetmulky/Library/Application Support/JetBrains/PyCharmCE2022.2/scratches/scratch.py
movie_id ----> ('type',)
movie_id ----> ('startYear',)
movie_id ----> ('runtimeMinutes',)
movie_id ----> ('averageRating',)
movie_id ----> ('type', 'startYear')
movie_id ----> ('type', 'runtimeMinutes')
movie_id ----> ('type', 'averageRating')
movie_id ----> ('startYear', 'runtimeMinutes')
movie_id ----> ('startYear', 'averageRating')
movie_id ----> ('runtimeMinutes', 'averageRating')
movie_id ----> ('type', 'startYear', 'runtimeMinutes')
movie_id ----> ('type', 'startYear', 'averageRating')
movie_id ----> ('type', 'runtimeMinutes', 'averageRating')
```

The idea here is to remove duplicates from the left attribute and the right attribute(s) and compare the count. If it is equal, it is a dependency. A hash-table is used to sieve out redundant repetitive dependencies. It took 15 minutes for the program to finish. The code for the naive approach is attached as naive_approach.py.

Q4)

All the dependencies overlap and the brute force approach introduces many more dependencies which weren't discovered. Every possible combination of dependencies were found. But one dependency

genre → genre_id

was not found in the naive approach which isn’t redundant.

Q5)

Find the candidate keys :

Here movie_id and member_id are the core attributes as they always appear on the left rather than the right.

Canonical cover is basically functional dependencies which cannot be reduced any further

Some are as follows:

movie_id —-> type,startYear,runtimeMinutes,avgRating

genre —-> genre_id

member_id —-> birthYear)

3NF Decomposition :

Table 1 : movie_id, member_id and genre_id are the primary keys

Table 2: movie_id is the primary key, type,startYear,runtime and avgRating

Table 3: Genre_id is the primary key, genre

Table 4: member_id is the primary key, birthYear

Table 5: movie_id and member_id are the primary keys, character