**Name:** *Yuvraj Singh*

**1) Creating the new Relation**

For this section in the assignment, a new relation was to be created that practically used information from each relation created in the previous assignment. The new relation is the result of joining Movie, Movie\_Genre, Genre, Member, Movie\_Actor, Actor\_Movie\_Role, and Role. The attributes for the new relation are movieId, type, startYear, runtime, avgRating, genreId, genre, memberId, birthYear, role. In the previous assignment, these relations were already created and cleaned into new cvs to be loaded into a relational database which saved some time for this section. Using those cleaned csvs, the relations needed were joined together using by creating a *Pandas DataFrame*. For the resulting relation, a new csv was generated and called “tbl.csv” and placed in the data folder directory. To generate the new relation, copy over the data folder director from assignment 2 and paste it in the home directory of assignment3 and then run the *def create\_new\_relation* method in *assignment3.py*.

**2) Functional Dependencies via Naïve Approach**

In *assignment3.py* the function called *def check\_func\_dep\_naiv* will generate the functional dependencies from the resulting relational from the first section. The relation was loaded into a *Pandas Dataframe* where the data in each column was checked against the data at the same index in a different column. Assuming that the functional dependency for each column combination is true, each row is then checked to see if the data in that row matches the data in the next row. By checking to see if the rows are the same on both left and right side then we can continue to assume that it is a functional dependency. If either side is false, then those columns are not a functional dependency. Running the program for approximately 30 minutes shows that it was only able to verify whether only a few combinations have functional dependencies, thus it’s safe to assume it would take over 5 hours to finish checking everything.

**3) Functional Dependencies via Pruning**

In *assignment3.py* the function called *def check\_func\_dep\_prune* will generate the functional dependencies from the resulting relational from the first section via pruning instead of the naïve approach. What’s different from this approach is that for each combination for the left side, an equivalent class is generated. Then when comparing two columns, the equivalent classes are then compared by checking to see if the indexes of each unique data in the left side is a subset of the indexes of each unique data in the right side. If so, a tuple of the left side column name and right side column name is saved to a list along with all of the other possible combinations of left side columns that have this resulting left column is a part of. Once the program is done computing, the *def save\_func\_deps* method is called to save the list of functional dependencies to a text file.

1. **No Restriction on Actor Title Role**

If the relation was not limited to an actor playing only one role per movie, there would definitely be a drastic change in the outcome. For one, functional dependencies that are a combination of the titleID and actorID may not even be valid anymore. Secondly, dependencies that involve role may also no longer be valid functional dependencies.

1. **3NF Decomposition**

After saving the functional dependencies generated from the third section, a 3NF decomposition of the relation from section one and the functional dependencies is created. When looking at the file of the functional dependencies, we can see the attributes that appear on the left-hand side and never the right, also attributes that never appear in the functional dependency. These attributes make up *Core* of the Candidate Keys. The attributes that belong to the *Core* are memberID and titleID. The attributes that appear on the right-hand side of some of the functional dependencies are the *Exterior*. These attributes are *type, startYearm runtimeMinutes, averageRating, genreID, genre, birthYear, & role*.