Luisa Baldino, Dan Haub, Finn White
Rao Ali
CPSC408
18 December 2021

Final Write Up

Name of App: Six Degrees of IMDB

Catch Line: Bring the fun to the next Dodge Party

Problem Addressed:

Movie trivia games are not common in the game market and the ones that do exist get boring quickly because they all feel the same. Games on the market today are also hard for people with little movie expertise because they mostly consist of guessing where a scene or quote is from. We sought out to change that by introducing a new form of movie trivia game. It is inspired by the Wikipedia game where you must connect two seemingly random pages through hyperlinks. Or 6 degrees of Kevin Bacon, but more Hollywood oriented. In our game, it is your goal to create a path between two defined actors. At each step along the way you are presented with either all the movies the current actor has been in or all the actors that are in the current movie. Because the movies and actors are presented to the user at each step, they do not have to be an expert to go through the game. While there's no guarantee you will find the end, it is guaranteed to make you smile. In addition, our game is connected to a database with 80k+ movies and 250k+ actors, so it can also be used if the user just wants to learn about movies. Can you connect Steve Buscemi to Jason Schwartzman in only 6 steps? Play our game to find out!

Databases:

The database side of our project is currently using both a MySQL instance and a Neo4j instance (See Figure 1 for MySQL and Figure 2 for Neo4j).

The MySQL database is composed of 5 different tables: Movies, MoviePeople, Cast, Game, and Player. The Movies and MoviePeople tables are self-explanatory and store data on movies and people who were a part of movies. One aspect that is used heavily throughout our app is the Popularity attribute. This is the average number of ratings all the movies a person has been a part of have received. This allows us to create games and query people that are the most well-known. This makes the game side of the app more interesting because instead of being given a random unknown actor, we can provide a big name for the game to start with. The Cast table is what connects them and stores the data about who was in what movies and what their role in the movie was. Game and Player are used to keep track of player data and all the games players have played. Because a game starts and ends with an actor, the game table has two foreign keys to the MoviePeople table. The game table also keeps track of the path taken by a user to complete the game. This is stored as a blob object and on the app side it is simply a serialized int array of alternating Movie IDs and MoviePeople IDs. The most interesting part of the MySQL database is the way in which we used stored procedures. We ensured that any form of data the app will need has a corresponding stored procedure. This ensures that all written queries reside in the MySQL instance and nowhere else.

The Neo4j database is a scaled down version of the MySQL database in graph format. It is storing the 2000 most popular movies and all actors related to them. It is also only storing the ID of the actor/movie from the MySQL DB and the Title/StageName respectively. The DB is then used to create games that have guaranteed connections. It also allows for finding the optimal path of a certain game (minimum number of steps between two actors) and through that a difficulty can be applied in the future. For example, a game with an optimal path of 3 is most likely easier than a game with a optimal path of 10. Without the use of the Neo4j database creating games that have a guaranteed optimal path would have been much less trivial and less efficient.

Database Interaction:

The API that interacts with the MySQL database works through stored procedures. Any query that is required for the application has an equivalent stored procedure. This includes modification, creation, and selection of records. There were a few reasons we chose to do it in this way. Firstly, it allows for the API to be readable and concise. Instead of having hard coded SQL queries in the API, it now just consists of a single callProc(nameOfProc) function for interaction with the database. It also increases security because all interactions with the database are only coded as procedure names instead of the exact query that is being run. When a procedure is called the API then parses the results into a JSON. The JSON object is then what is passed to the GUI for it to display. We chose JSON because it works seamlessly with an implementation of a GUI. The API also handles the interaction with the Neo4j database to find

game end points. To do this it is given a distance (or number of steps) that the game would like to be created on and then returns a random start and end point for the GUI to display to the user. It also has seeding functionality built in so games can be replayed in the future, and it is not all up to randomness.

Front End:

The front end of our application is built using the tKinter library as well as the command line. The command line is reserved for admins who would like to make quick changes to the database like deleting one actor or updating a movie's rating. This design choice is to hide the editing process from the user. A user will never need to edit records in the database, so presenting it in the GUI is unnecessary. We chose to use tKinter for our implementation because it has clear documentation and there is a lot of information about it available online.

Each major functionality designed for this project is split in the GUI as: Browse, Game, and Edit. Browse allows you to search and view info in actors, movies, or character names from movies. Game organizes the chaining of actors and movies in a user-friendly manner (See Figure 3), and if you are successful, your score is calculated and saved in the database. Edit sends you to the command line where each editing option-- Add, Delete, Edit, Save (Commit), and Undo (Rollback)-- are neatly numbered and displayed. The GUI also supports a simple login system to enter as a user (prompted to enter username) or as administrator (prompted for password).

Figures:

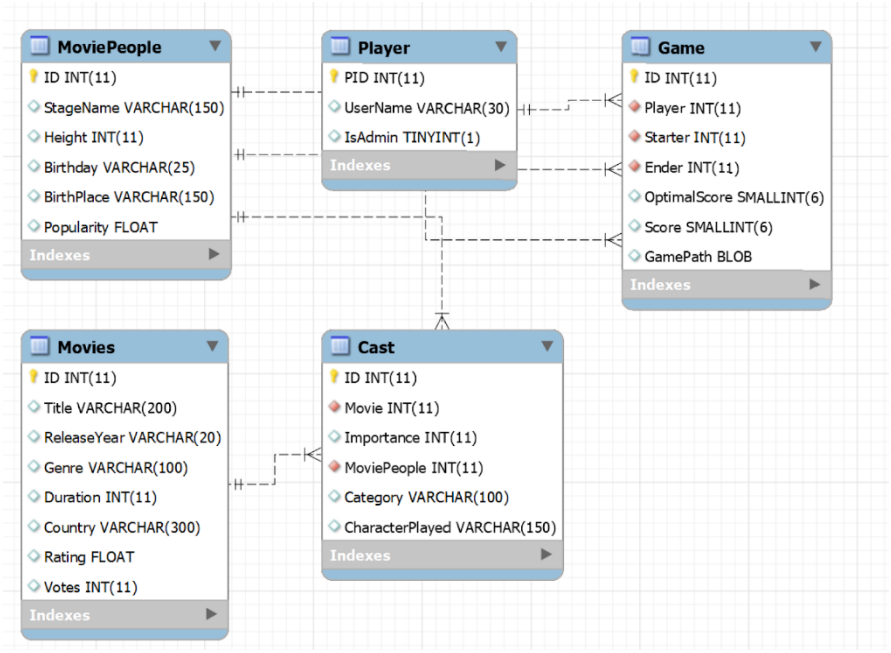

Figure 1. ER diagram of MySQL database.

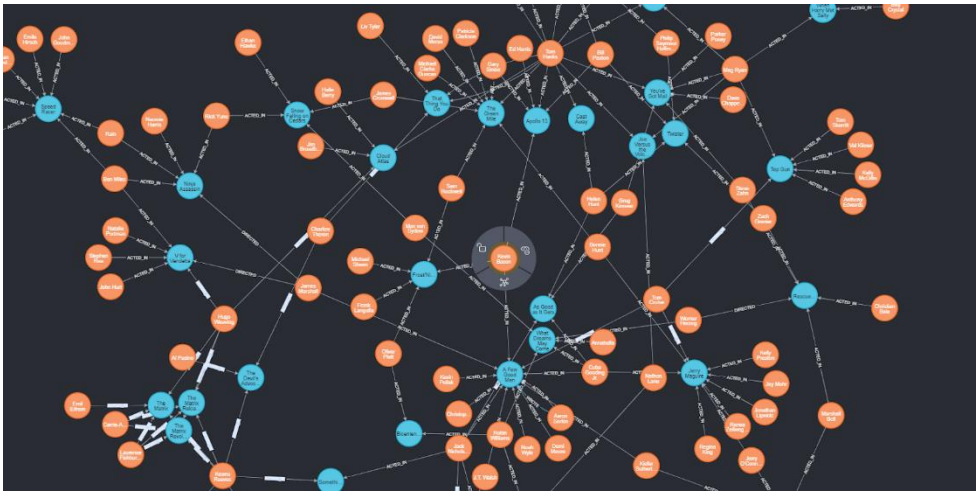

Figure 2. Example set of Neo4j database.

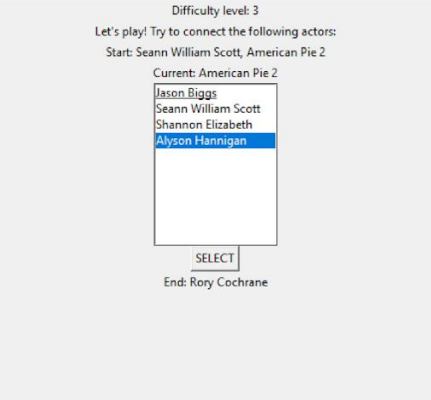

Figure 3. Example of Game Window in tKinter