St. Joseph's University

Bengaluru, Karnataka - 560027

Graph Database Project executed for Foundations of Data Science Lab (BD2P1)

Submitted by:

Yobah Bertrand Yonkou (222BDA15)

Sinchana R (222BDA48)

Alvina Joanna (222BDA63)

Saba Santhosh (222BDA65)

Submitted to:

Jayati Bhadra

Head of Department

Department of Advanced Computing

St. Joseph's University

Aim

As the world advances, the data we generate and store becomes increasingly complicated and complex. In order to access such data in an intuitive and effective manner, we need data structures that can handle the dynamic nature of growing data. Graph databases utilise the structure of graphs to store data in the form of nodes and edges for querying. One of the best domains to implement this style of storage is while storing information on shows, books, and other media as each instance of an entity (such as movie, book, YouTube channel) will be connected to another instance or another entity (like two movies made by the same directors or two books written by the same author). A graph database is beneficial in creating such complex entities and building relationships between them such that we can query based on relationship or entity.

Domain

Graph databases, Media

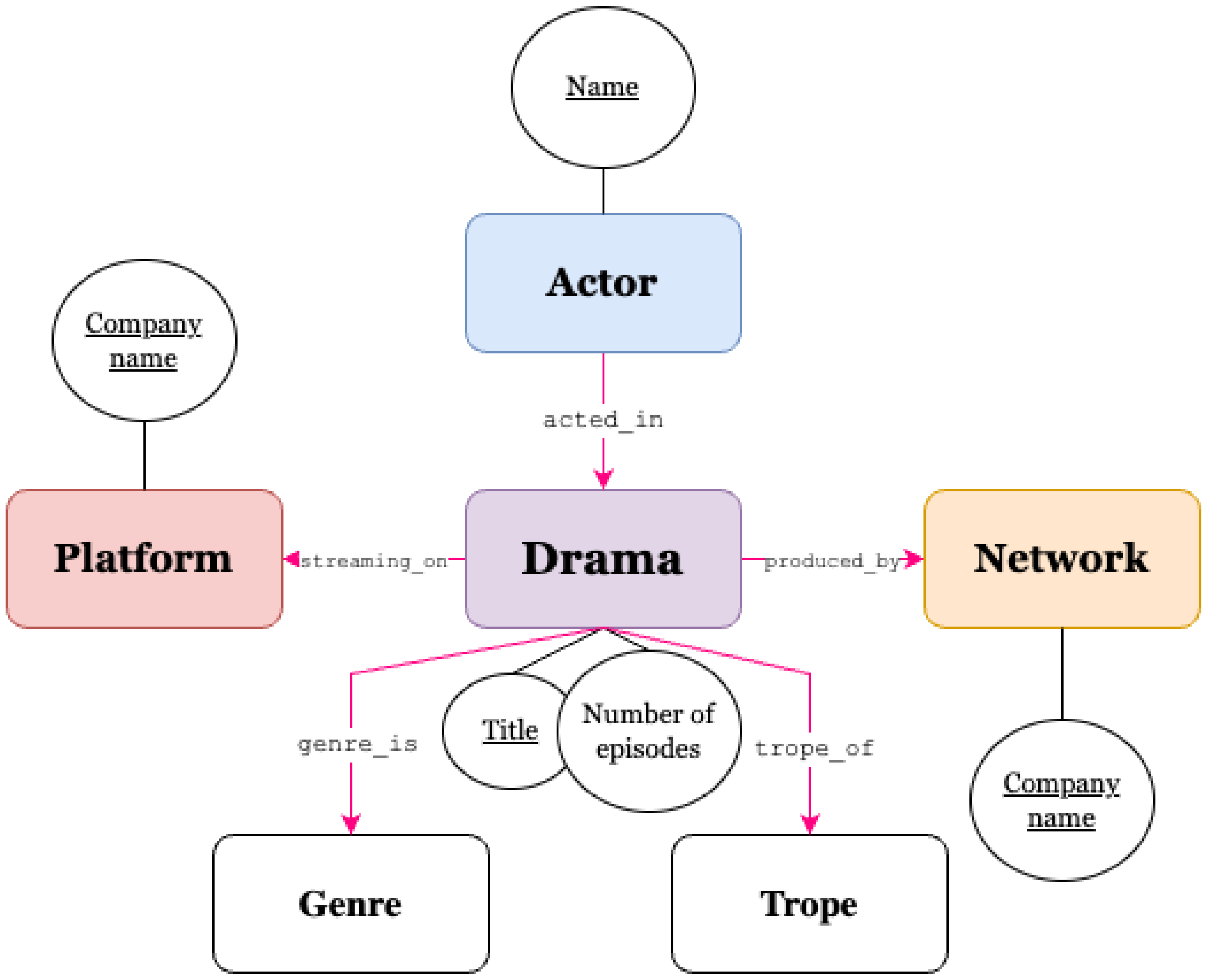
Problem statement

Create and implement a queryable graph database

Introduction

As we pondered over what type of data might be best represented by a graph database, we were excited to explore the possibilities that various domains offered to us to build a database containing the information at hand. Our first attempt was to build a graph database that could implement our family tree and that soon proved to be a difficult feat due to incomplete information that could disrupt our implementation.

As one of the members in our team is a fan of Korean Dramas (henceforth referred to as k-dramas) and wanted to keep track of all that she had watched, we decided to create a graph database that would include information on various k-dramas. This would be a dynamic database that will also have a visual representation of the database that is intuitive and helps to discover any new shows or actors.

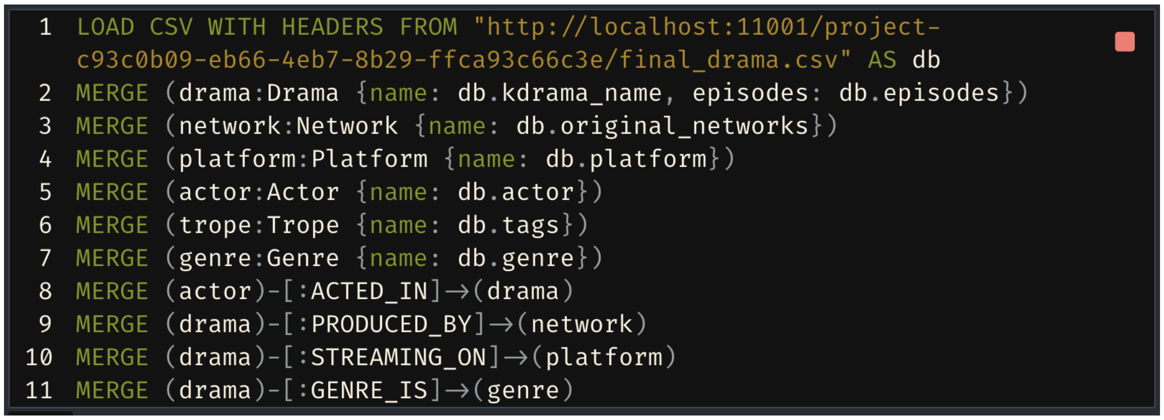


Literature Survey

Our main reference was the Cypher Query Language documentation. All other references have been mentioned in the References section.

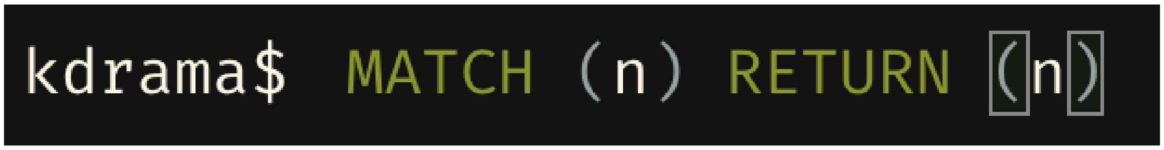
Implementation

- *Data collection:* Our dataset was collected from Kaggle. This dataset consisted of 1278 unique dramas with relevant information regarding each drama. We collected csv files that contained the drama name, number of episodes, tags that explained the tropes of each drama, actors who acted in the drama, platforms that the drama is streaming on and networks that have produced the drama.
- *Data cleaning:* Despite the dataset being relatively clean, we had to clean up the strings, merge and separate files are required and create a consolidated final csv file that contained all the information that we needed in a format that would make the loading of the dataset into neo4j much faster and more efficient.
- *Data loading:* The dataset was loaded into neo4j using the following command. This brought all the necessary data together into one connected graph with the relevant properties and relationships.



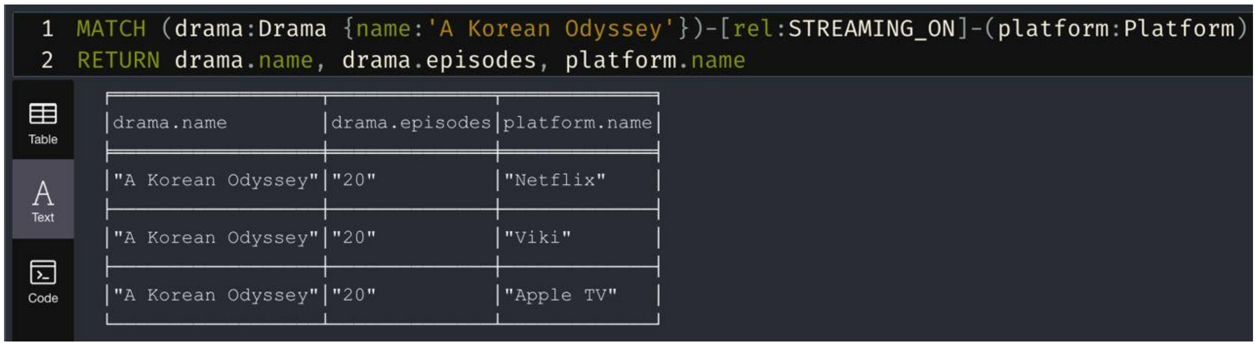
```
LOAD CSV WITH HEADERS FROM "http://localhost:11001/project-
c93c0b09-eb66-4eb7-8b29-ffca93c66c3e/final_drama.csv" AS db
MERGE (drama:Drama {name: db.kdrama_name, episodes: db.episodes})
MERGE (network:Network {name: db.original_networks})
MERGE (platform:Platform {name: db.platform})
MERGE (actor:Actor {name: db.actor})
MERGE (trope:Trope {name: db.tags})
MERGE (genre:Genre {name: db.genre})
MERGE (actor)-[:ACTED_IN]->(drama)
MERGE (drama)-[:PRODUCED_BY]->(network)
MERGE (drama)-[:STREAMING_ON]->(platform)
MERGE (drama)-[:GENRE_IS]->(genre)
```

- *Data visualisation:* Neo4j provides us with a beautiful and interactive visualisation of the entire graph. We have used that to view our graph with all the node classes and relationships.



```
kdrama$ MATCH (n) RETURN (n)
```

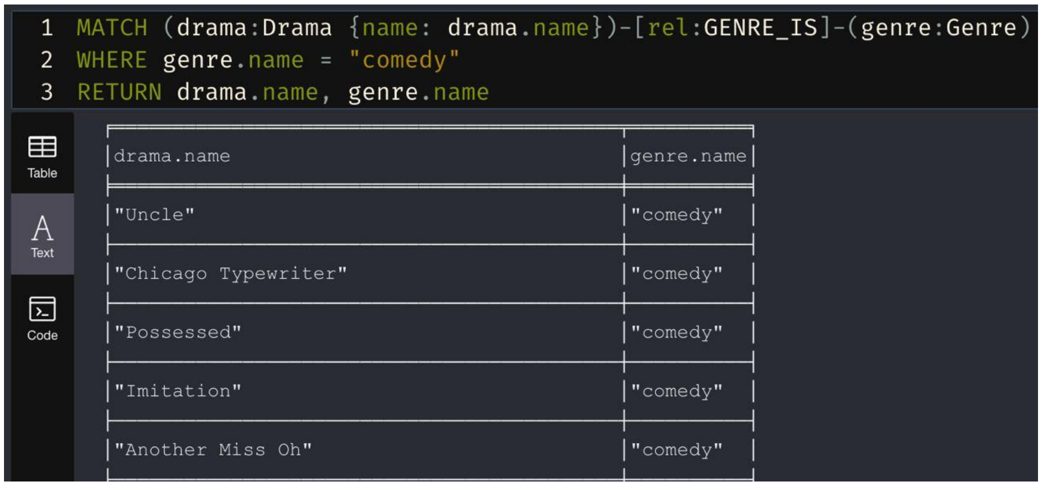
- *Data querying:* We were able to query for a desired output in the neo4j browser using their Cypher Query Language.



```
MATCH (drama:Drama {name:'A Korean Odyssey'})-[rel:STREAMING_ON]-(platform:Platform)
RETURN drama.name, drama.episodes, platform.name
```

```
╒══════════════════╤══════════════╤═════════════╕
│drama.name        │drama.episodes│platform.name│
╞══════════════════╪══════════════╪═════════════╡
│"A Korean Odyssey"│"20"          │"Netflix"    │
├──────────────────┼──────────────┼─────────────┤
│"A Korean Odyssey"│"20"          │"Viki"       │
├──────────────────┼──────────────┼─────────────┤
│"A Korean Odyssey"│"20"          │"Apple TV"   │
└──────────────────┴──────────────┴─────────────┘
```

This printed a drama along with all the platforms that it streaming on ⬆️

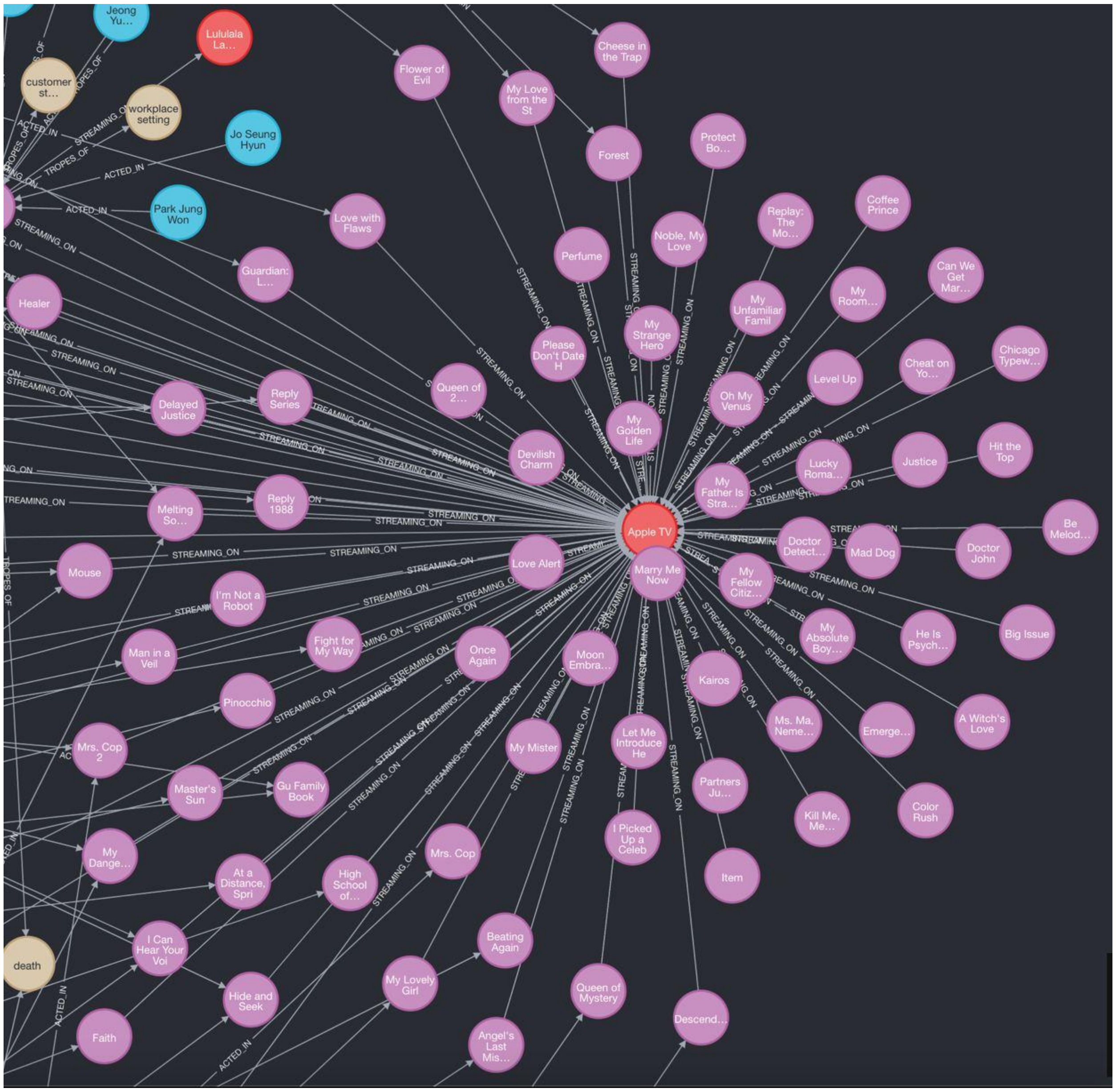


```
MATCH (drama:Drama {name: drama.name})-[rel:GENRE_IS]-(genre:Genre)
WHERE genre.name = "comedy"
RETURN drama.name, genre.name
```

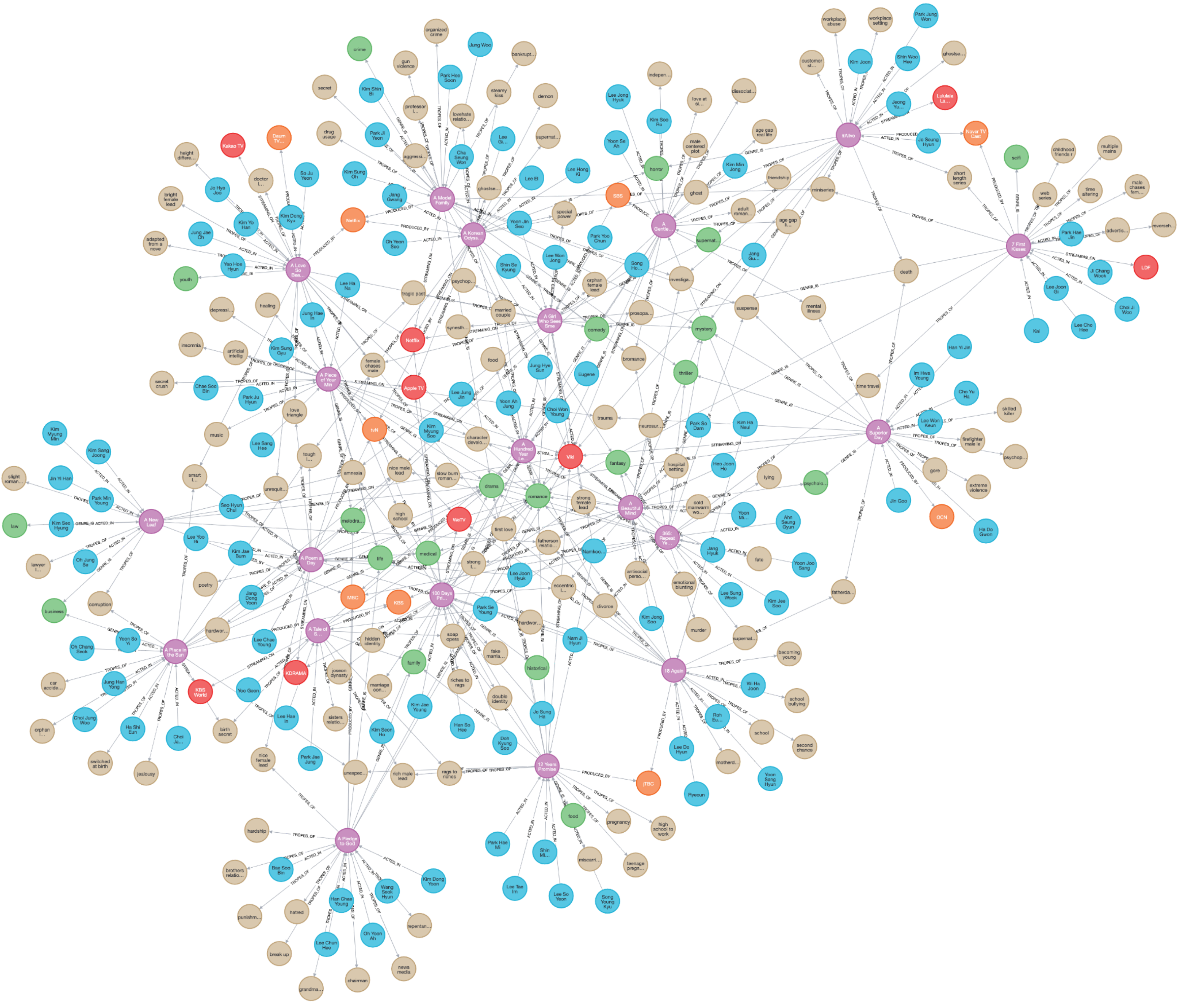
```
╒════════════════════════════════════════════╤══════════╕
│drama.name                                  │genre.name│
╞════════════════════════════════════════════╪══════════╡
│"Uncle"                                     │"comedy"  │
├────────────────────────────────────────────┼──────────┤
│"Chicago Typewriter"                        │"comedy"  │
├────────────────────────────────────────────┼──────────┤
│"Possessed"                                 │"comedy"  │
├────────────────────────────────────────────┼──────────┤
│"Imitation"                                 │"comedy"  │
├────────────────────────────────────────────┼──────────┤
│"Another Miss Oh"                           │"comedy"  │
├────────────────────────────────────────────┼──────────┤
```

This printed the dramas that are tagged under the genre of "comedy" ⬆️

And the below image illustrates how neo4j browser's graph visualisation can interactively show us grouping in the graph. We asked it to group all dramas that are streamed on Apple TV platform and it pulled together the drama titles that have the relationships 'streaming_on' with the specific streaming platform. ⬇️


Jeong Yu...
Lululala La...
customer st...
workplace setting
Jo Seung Hyun
Park Jung Won
Flower of Evil
My Love from the St
Cheese in the Trap
Forest
Protect Bo...
Love with Flaws
Guardian: L...
Perfume
Noble, My Love
Replay: The Mo...
Coffee Prince
Healer
My Unfamiliar Famil
My Room...
Can We Get Mar...
Please Don't Date H
My Strange Hero
Level Up
Cheat on Yo...
Chicago Typew...
Delayed Justice
Reply Series
Queen of 2...
Oh My Venus
My Golden Life
Devilish Charm
Lucky Roma...
Justice
Hit the Top
My Father Is Stra...
Melting So...
Reply 1988
Apple TV
Doctor Detect...
Mad Dog
Doctor John
Be Melod...
Mouse
Love Alert
Marry Me Now
My Fellow Citiz...
I'm Not a Robot
Man in a Veil
Fight for My Way
Once Again
Moon Embra...
My Absolute Boy...
He Is Psych...
Big Issue
Kairos
Pinocchio
Mrs. Cop 2
My Mister
Let Me Introduce He
Ms. Ma, Neme...
Emerge...
A Witch's Love
Master's Sun
Gu Family Book
Partners Ju...
Kill Me, Me...
Color Rush
My Dange...
I Picked Up a Celeb
Mrs. Cop
At a Distance, Spri
High School of...
Item
Beating Again
death
I Can Hear Your Voi
Hide and Seek
My Lovely Girl
Queen of Mystery
Descend...
Faith
Angel's Last Mis...
STREAMING_ON
ACTED_IN
TROPES_OF

The full graph that we generated from our dataset



Conclusion & Inferences

In this project, we took a database of information on Korean Dramas and converted it into an intuitive and easily query-able graph database. Our journey through this project has enabled us to see a whole new world of graph databases that are widely being used. With even tech giants like Microsoft using their own graph database named Microsoft Graph that is a large knowledge graph that contains the data that is to be utilised by their recently released AI - Copilot. Thus, we can infer that having an intimate knowledge of how these databases can work will give us an edge in the future.

Future work

This project can be further expanded to include many more details of dramas and other important entities related to dramas and the other node classes that we have. Enabling ardent fans to have access to such a database would be an amazing way to grow this database while also giving them an easy tool to find what they want more efficiently.

Key Learnings

During the execution of this project, we had the opportunity to dig a little bit into *graph databases*. Our focus was on the Neo4J graph database management system where we learned how to use the Cypher Query language to convert data (in the form of CSV files) to create nodes, define relationships between nodes as well as visualizing the graph created from the data. More so, it was an amazing experience to see how data can be represented using graphs and how graph databases are prolific at representing the relationships between multiple nodes. Furthermore, this project presented us with an opportunity to go through certain data preprocessing steps using Python one of which would be merging multiple files to form a dataset that would be suitable for our problem statement. In conclusion, working on this project gave us the immense opportunity to learn more about graph databases and how data can be represented using graphs which will be a valuable skill as a data scientist.

References

1. Dataset source
   [Korean drama list [about 1278 unique dramas] | Kaggle]

2. Cypher Query Language - Neo4J official documentation
   [Cypher Query Language - Developer Guides]

3. Tutorial: Import Data - Neo4J official documentation
   https://neo4j.com/docs/getting-started/cypher-intro/load-csv/

4. Explanation of error "Cannot merge node using null property value for"
   https://neo4j.com/developer/kb/explanation-of-error-cannot-merge-node-using-null-property-value

5. Show all Nodes and Relationships in Data Browser Tab
   [Show all Nodes and Relationships in Data Browser Tab - Stack Overflow]

6. Example project that we referenced
   [Neo4j for Bollywood. A fun project to learn graph database | by Sixing Huang | Towards Data Science]

Start Date: April 20, 2023

End Date: April 28, 2023