NETFLIX TITLES DATASET

BY:

RAJAT TIWARI
KALYAN VANNE
JYOTHSNA LENKA
SAI TEJA MALLADI

PROBLEM STATEMENT

The problem is to analyze the Netflix titles dataset to gain insights into the content available on the platform. This includes understanding the distribution of titles by various attributes such as type (movie or TV show), genre, country, release year, and more.

WHY GRAPH DATABASE?

A graph database like Neo4j is well-suited for this problem because it allows us to model complex relationships between entities easily.

The Netflix titles dataset contains various entities such as titles, directors, actors, etc., and their relationships with each other (e.g., a title directed by a director, a title belonging to a genre, etc.). Graph databases excel at representing such interconnected data and querying relationships efficiently.

MODELING THE GRAPH DATABASE

- ***Title**: Represents a movie or TV show.*
- ***Director**: Represents the director of a title.*
- ***Actor**: Represents an actor/actress appearing in a title.*
- ***Country**: Represents the country associated with a title.*

NODES AND PROPERTIES

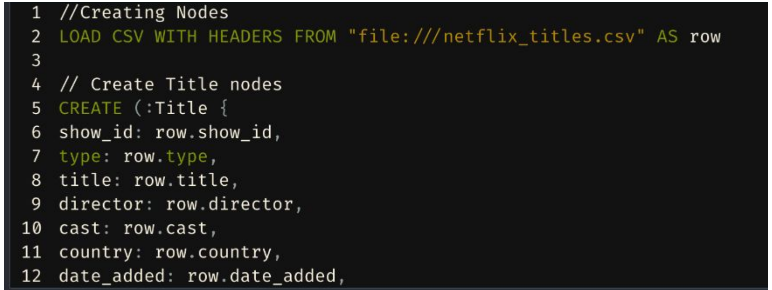
- ***Title****: Properties may include `show_id`, `title`, `type`, `date_added`, `release_year`, `rating`, `duration`, `description`.*
- ***Director****: Property include `name`.*
- ***Actor****: Property include `name`.*
- ***Genre****: Property include `name`.*
- ***Country****: Property include `name`.*

RELATIONSHIPS

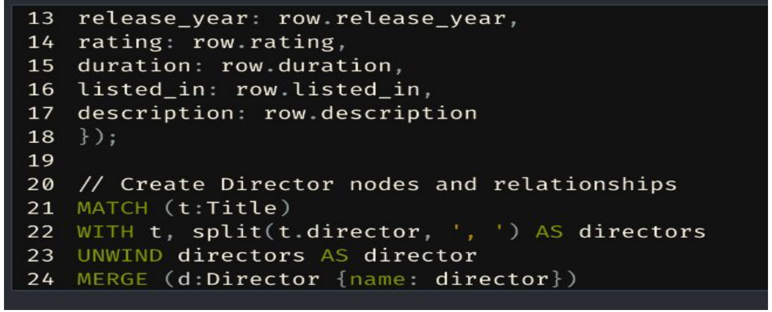
- ***DIRECTED_BY****: Relationship between Title and Director.*
- ***ACTED_BY****: Relationship between Title and Actor.*
- ***AVAILABLE_IN****: Relationship between Title and Country.*

CODING & IMPLEMENTATION

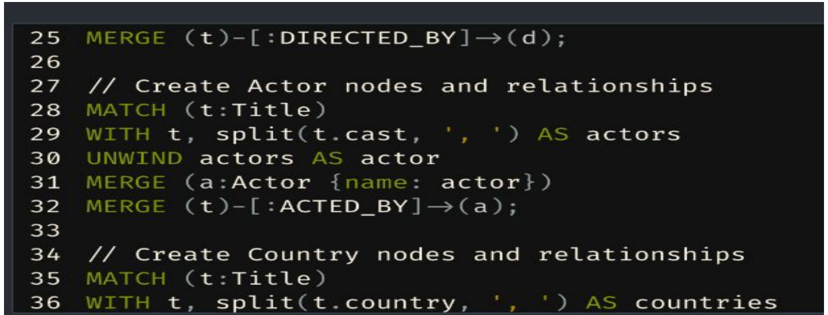
Creating and labeling Nodes and define Relationships



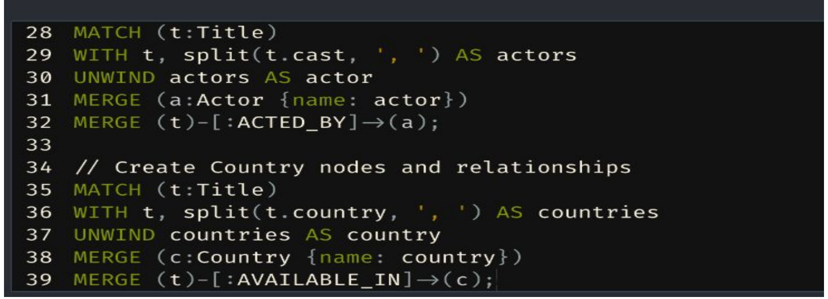
```
//Creating Nodes
LOAD CSV WITH HEADERS FROM "file:///netflix_titles.csv" AS row

// Create Title nodes
CREATE (:Title {
show_id: row.show_id,
type: row.type,
title: row.title,
director: row.director,
cast: row.cast,
country: row.country,
date_added: row.date_added,
```



```
release_year: row.release_year,
rating: row.rating,
duration: row.duration,
listed_in: row.listed_in,
description: row.description
});

// Create Director nodes and relationships
MATCH (t:Title)
WITH t, split(t.director, ', ') AS directors
UNWIND directors AS director
MERGE (d:Director {name: director})
```



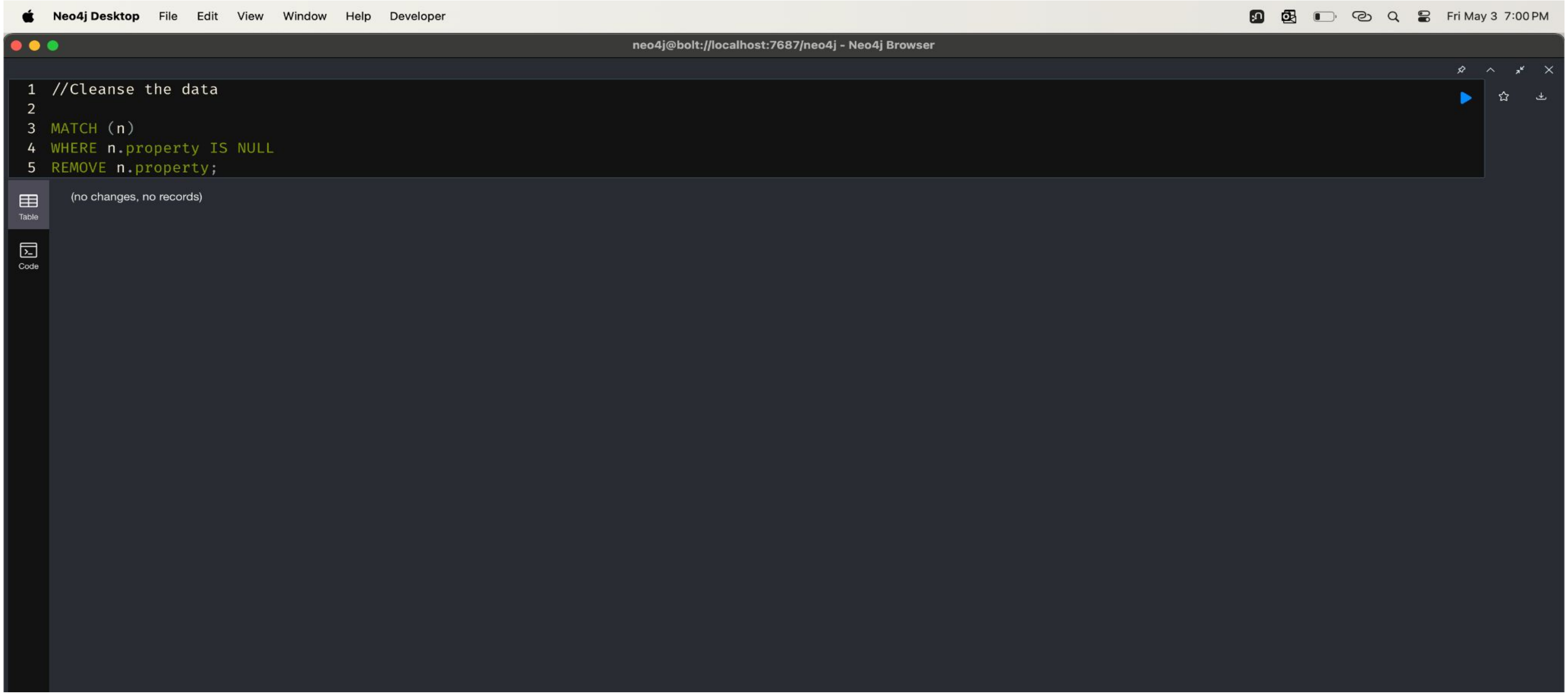
```
MERGE (t)-[:DIRECTED_BY]->(d);

// Create Actor nodes and relationships
MATCH (t:Title)
WITH t, split(t.cast, ', ') AS actors
UNWIND actors AS actor
MERGE (a:Actor {name: actor})
MERGE (t)-[:ACTED_BY]->(a);

// Create Country nodes and relationships
MATCH (t:Title)
WITH t, split(t.country, ', ') AS countries
```



```
MATCH (t:Title)
WITH t, split(t.cast, ', ') AS actors
UNWIND actors AS actor
MERGE (a:Actor {name: actor})
MERGE (t)-[:ACTED_BY]->(a);

// Create Country nodes and relationships
MATCH (t:Title)
WITH t, split(t.country, ', ') AS countries
```

```
UNWIND countries AS country
MERGE (c:Country {name: country})
MERGE (t)-[:AVAILABLE_IN]->(c);
```

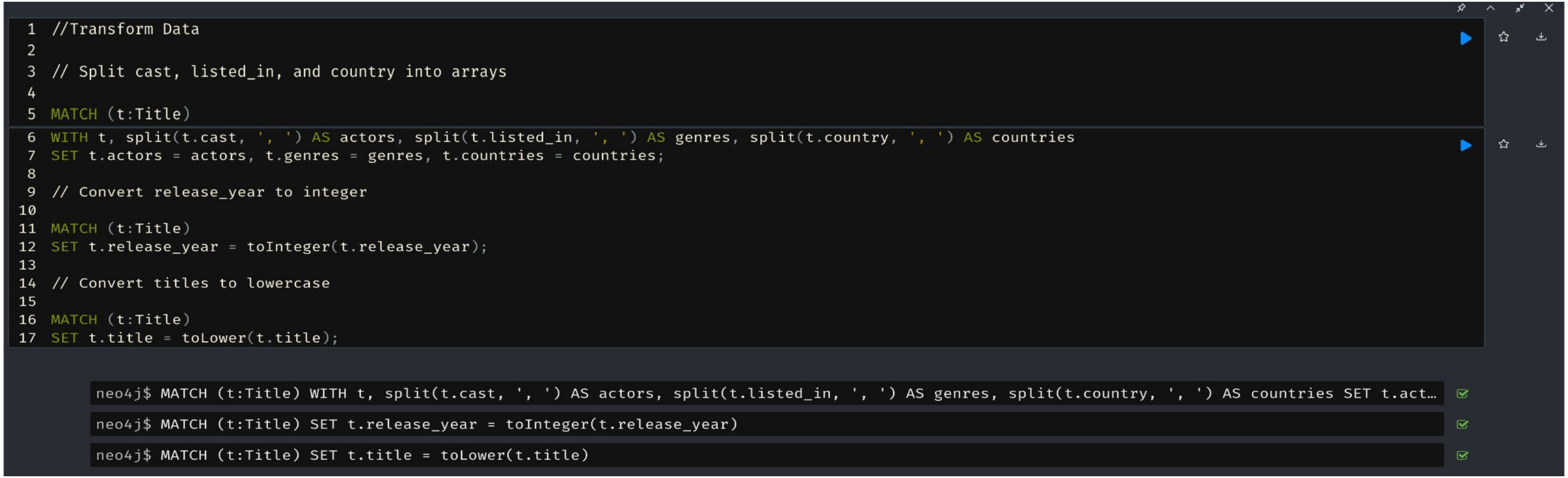
Cleanse data if there are any null values:



```
//Cleanse the data

MATCH (n)
WHERE n.property IS NULL
REMOVE n.property;
```

```
(no changes, no records)
```

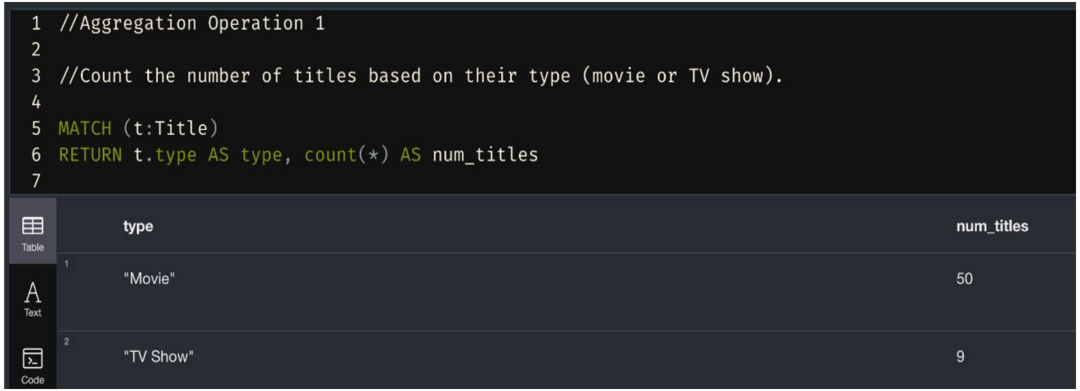
Transform data



```
//Transform Data

// Split cast, listed_in, and country into arrays

MATCH (t:Title)
WITH t, split(t.cast, ', ') AS actors, split(t.listed_in, ', ') AS genres, split(t.country, ', ') AS countries
SET t.actors = actors, t.genres = genres, t.countries = countries;

// Convert release_year to integer

MATCH (t:Title)
SET t.release_year = toInteger(t.release_year);

// Convert titles to lowercase

MATCH (t:Title)
SET t.title = toLower(t.title);
```

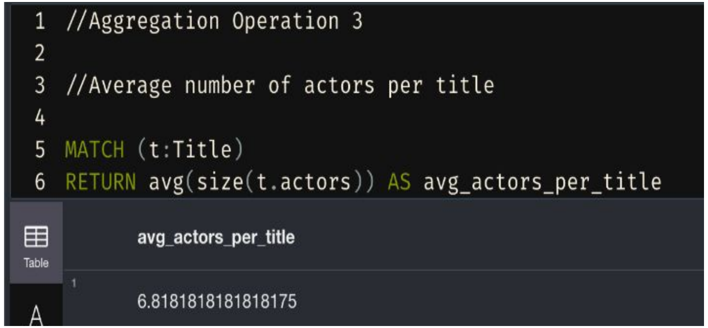
```
neo4j$ MATCH (t:Title) WITH t, split(t.cast, ', ') AS actors, split(t.listed_in, ', ') AS genres, split(t.country, ', ') AS countries SET t.act…
neo4j$ MATCH (t:Title) SET t.release_year = toInteger(t.release_year)
neo4j$ MATCH (t:Title) SET t.title = toLower(t.title)
```

Perform Aggregation Operations:



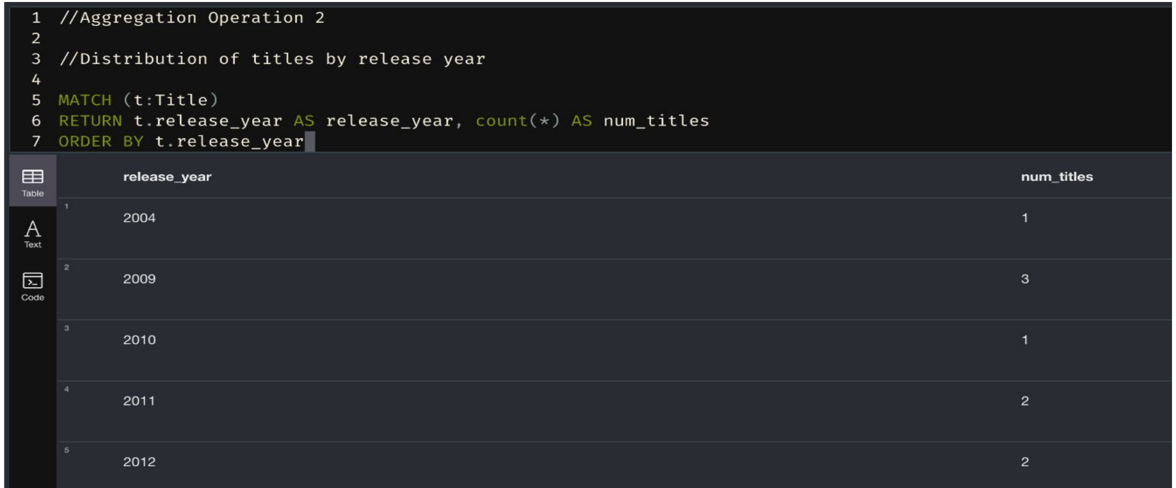
```
//Aggregation Operation 1

//Count the number of titles based on their type (movie or TV show).

MATCH (t:Title)
RETURN t.type AS type, count(*) AS num_titles

```

	type	num_titles
1	"Movie"	50
2	"TV Show"	9



```
//Aggregation Operation 3

//Average number of actors per title

MATCH (t:Title)
RETURN avg(size(t.actors)) AS avg_actors_per_title
```

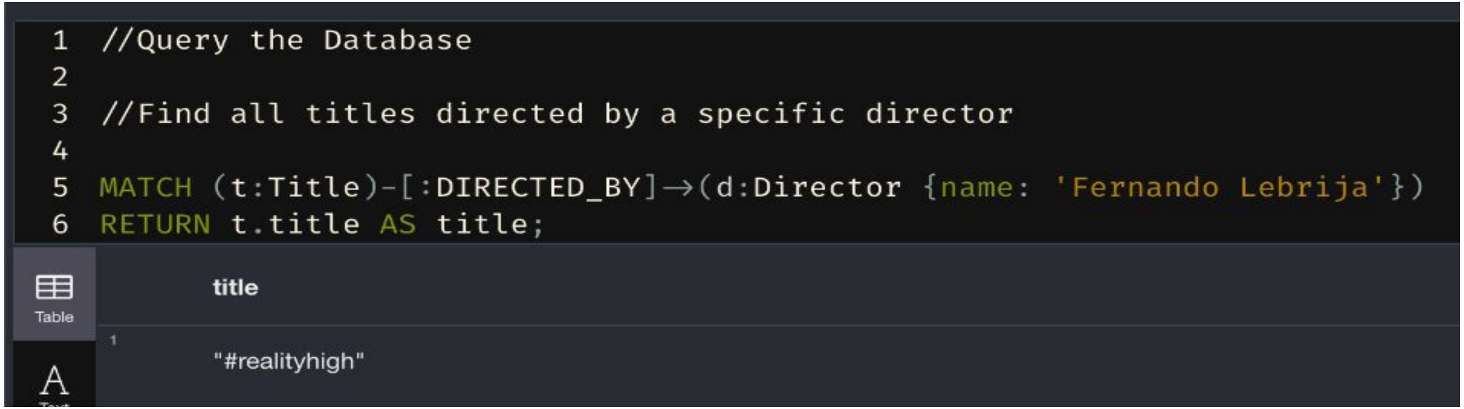
	avg_actors_per_title
1	6.8181818181818175



```
//Aggregation Operation 2

//Distribution of titles by release year

MATCH (t:Title)
RETURN t.release_year AS release_year, count(*) AS num_titles
ORDER BY t.release_year
```

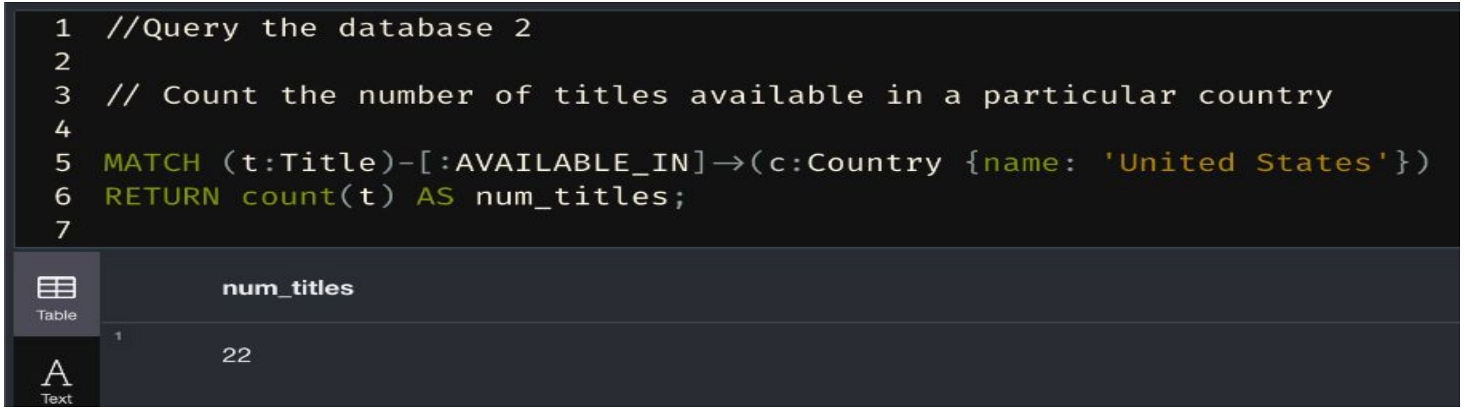
	release_year	num_titles
1	2004	1
2	2009	3
3	2010	1
4	2011	2
5	2012	2

Query your database:



```
//Query the Database

//Find all titles directed by a specific director

MATCH (t:Title)-[:DIRECTED_BY]->(d:Director {name: 'Fernando Lebrija'})
RETURN t.title AS title;
```

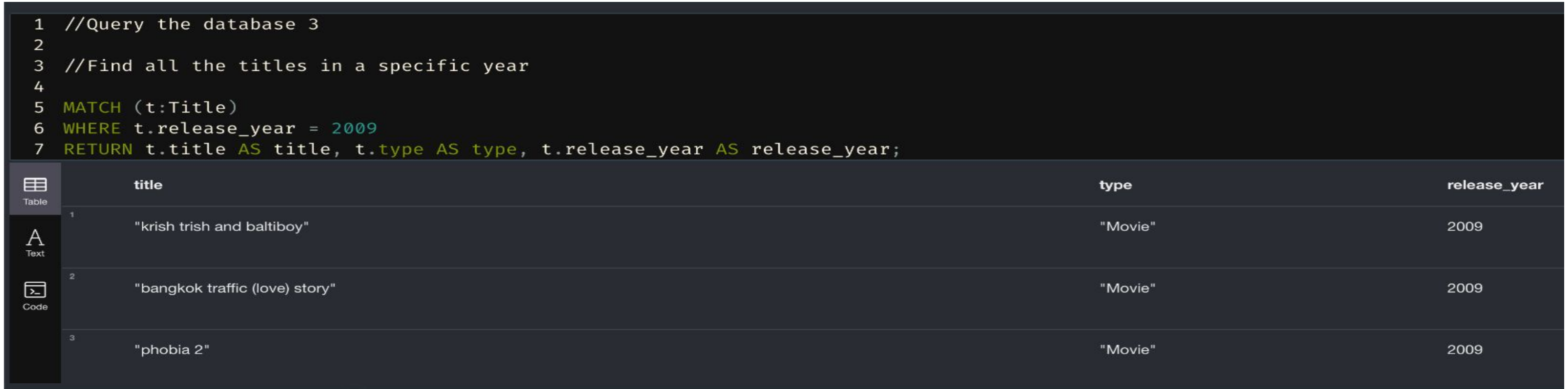
```
        title
1       "#realityhigh"
```



```
//Query the database 2

// Count the number of titles available in a particular country

MATCH (t:Title)-[:AVAILABLE_IN]->(c:Country {name: 'United States'})
RETURN count(t) AS num_titles;

```

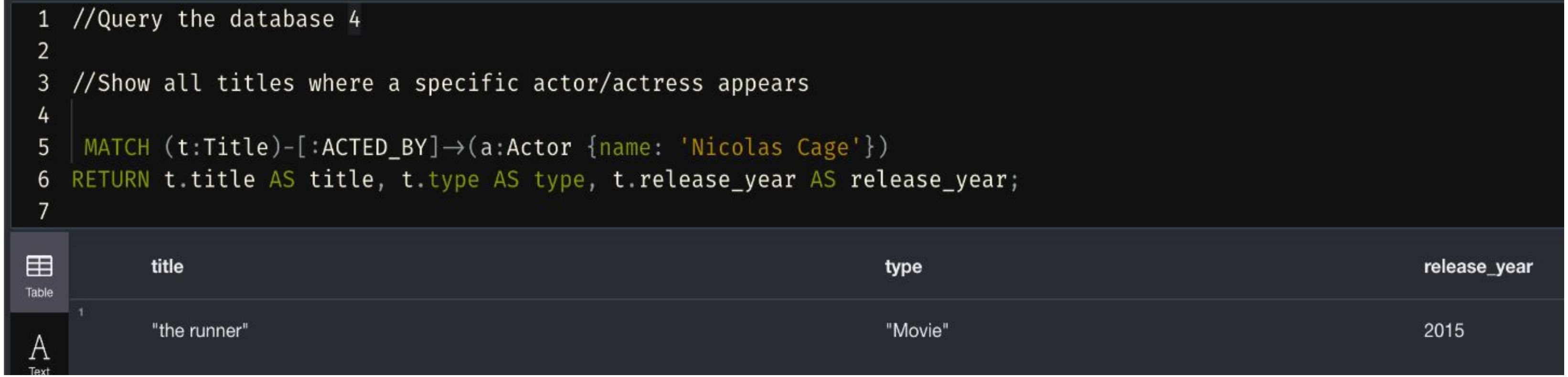
```
        num_titles
1       22
```

Query your database:



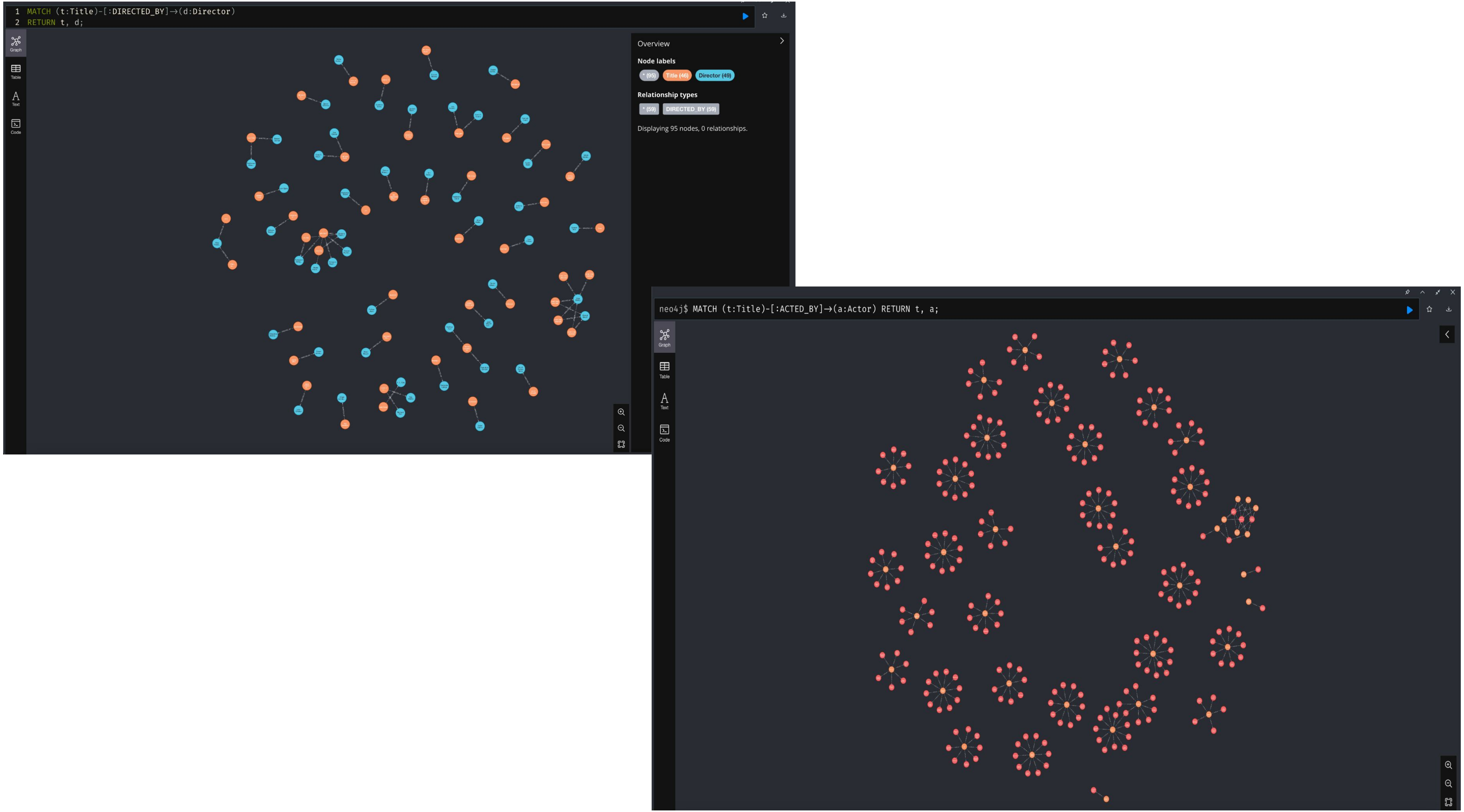
```
//Query the database 3

//Find all the titles in a specific year

MATCH (t:Title)
WHERE t.release_year = 2009
RETURN t.title AS title, t.type AS type, t.release_year AS release_year;
```

	title	type	release_year
1	"krish trish and baltiboy"	"Movie"	2009
2	"bangkok traffic (love) story"	"Movie"	2009
3	"phobia 2"	"Movie"	2009



```
//Query the database 4

//Show all titles where a specific actor/actress appears

 MATCH (t:Title)-[:ACTED_BY]->(a:Actor {name: 'Nicolas Cage'})
RETURN t.title AS title, t.type AS type, t.release_year AS release_year;
```

	title	type	release_year
1	"the runner"	"Movie"	2015



```
MATCH (t:Title)-[:DIRECTED_BY]→(d:Director)
RETURN t, d;
```

Overview

Node labels

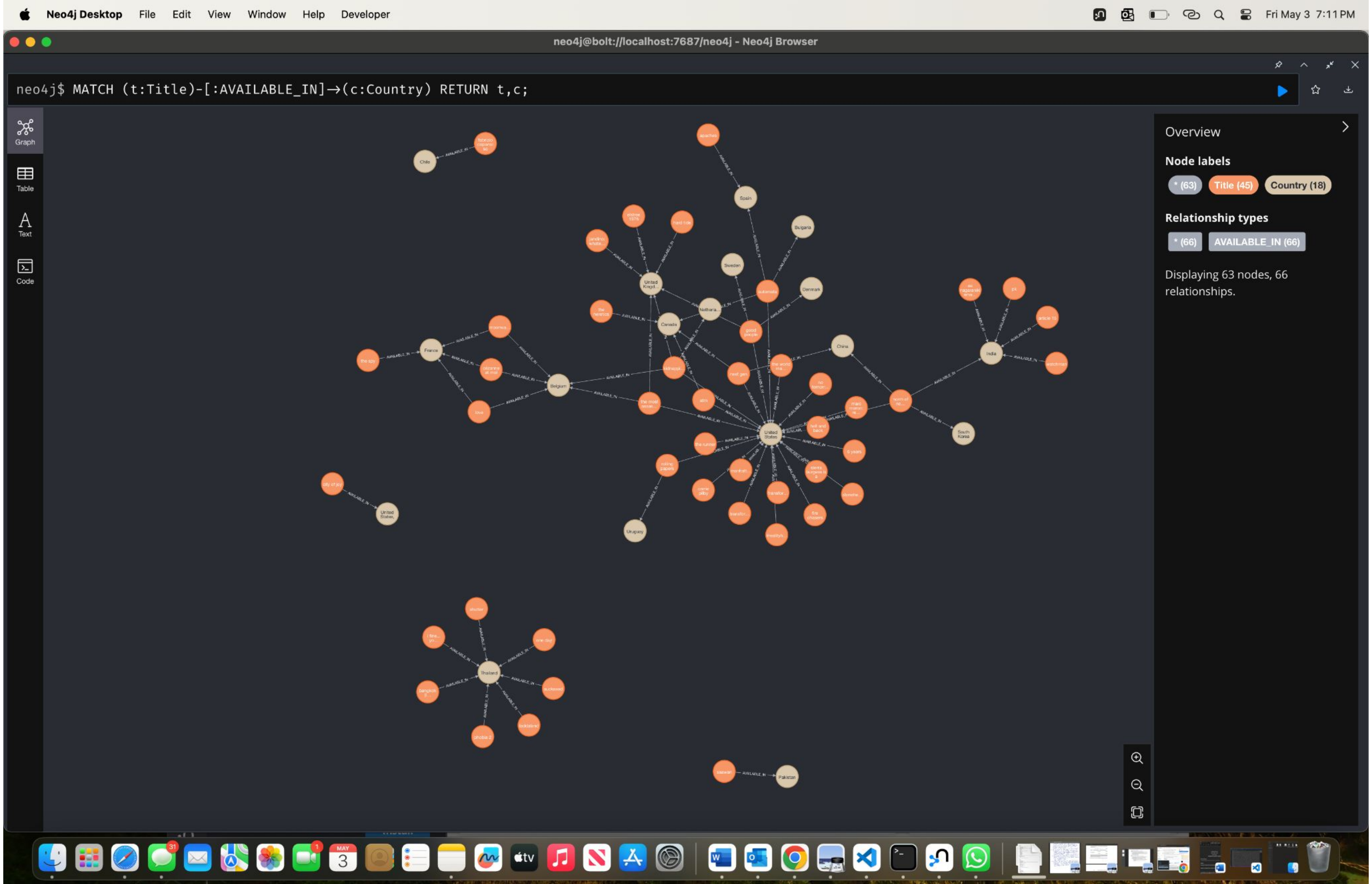
* (95) Title (46) Director (49)

Relationship types

* (59) DIRECTED_BY (59)

Displaying 95 nodes, 0 relationships.

```
neo4j$ MATCH (t:Title)-[:ACTED_BY]→(a:Actor) RETURN t, a;
```



```
neo4j$ MATCH (t:Title)-[:AVAILABLE_IN]→(c:Country) RETURN t,c;
```

Overview

Node labels

* (63) Title (45) Country (18)

Relationship types

* (66) AVAILABLE_IN (66)

Displaying 63 nodes, 66 relationships.

CONCLUSION

In this project, we explored the Netflix titles dataset using Neo4j, a graph database. We leveraged the graph database model to represent the relationships between different entities such as titles, actors, directors, genres, and countries. Through a series of steps, we performed data modeling, data cleansing, transformation, aggregation operations, and querying to gain insights into the dataset.

The use of Neo4j proved to be advantageous for this project due to its native support for graph data modeling and querying. It allowed us to represent complex relationships between entities in a natural and intuitive way, making it easier to analyze and query the dataset.

Throughout the project, we utilized Cypher queries to interact with the graph database, performing various tasks such as retrieving nodes and relationships, cleansing data, transforming data, and performing aggregation operations. We visualized the data using the Neo4j Browser interface, which provided an interactive way to explore the graph and gain insights into the relationships between different entities.