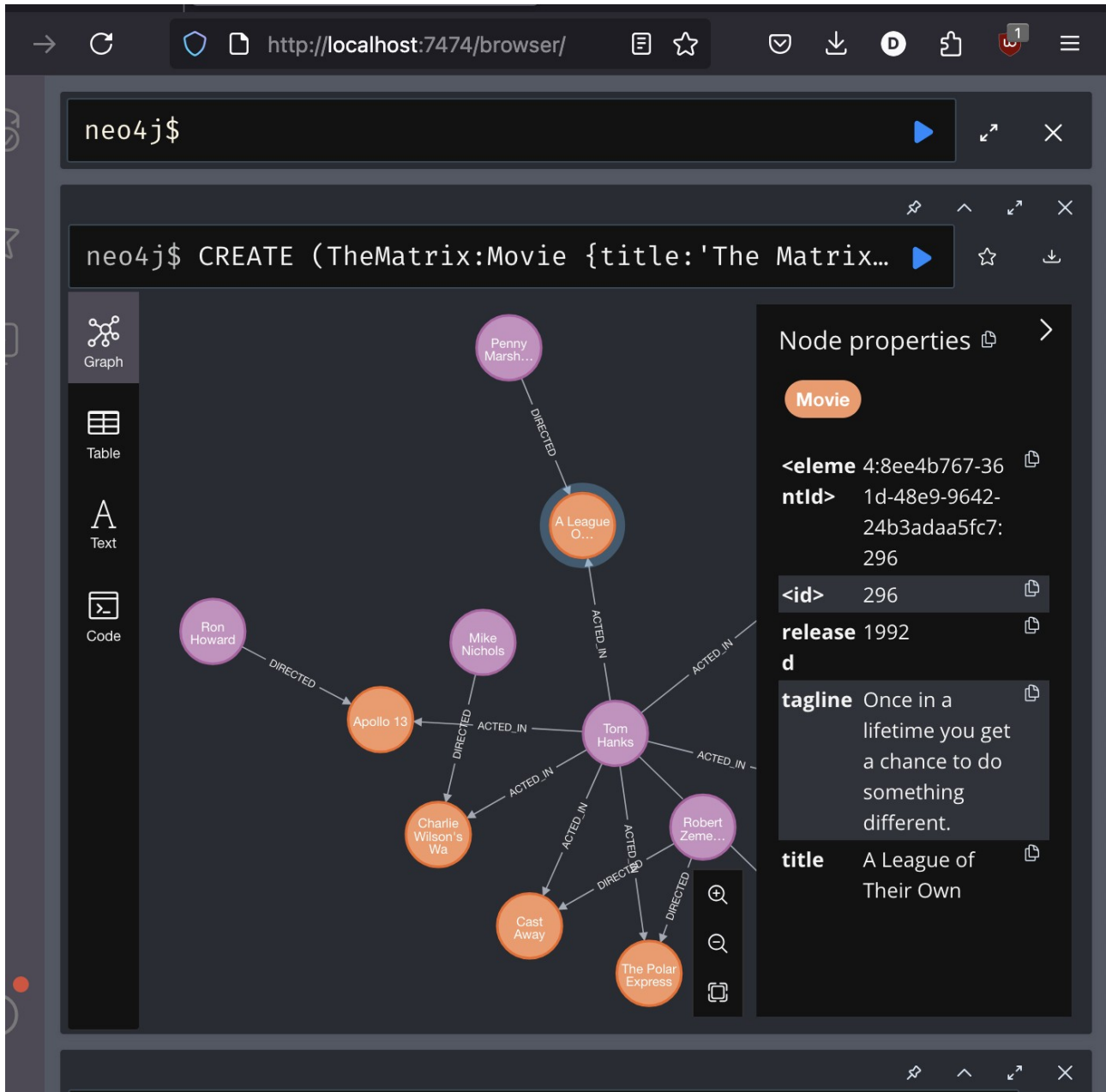


Daniel Soden Assignment 6 neo4j

1.

I started by executing play movie graph and then in the second page, I was prompted to execute this CREATE graph command:

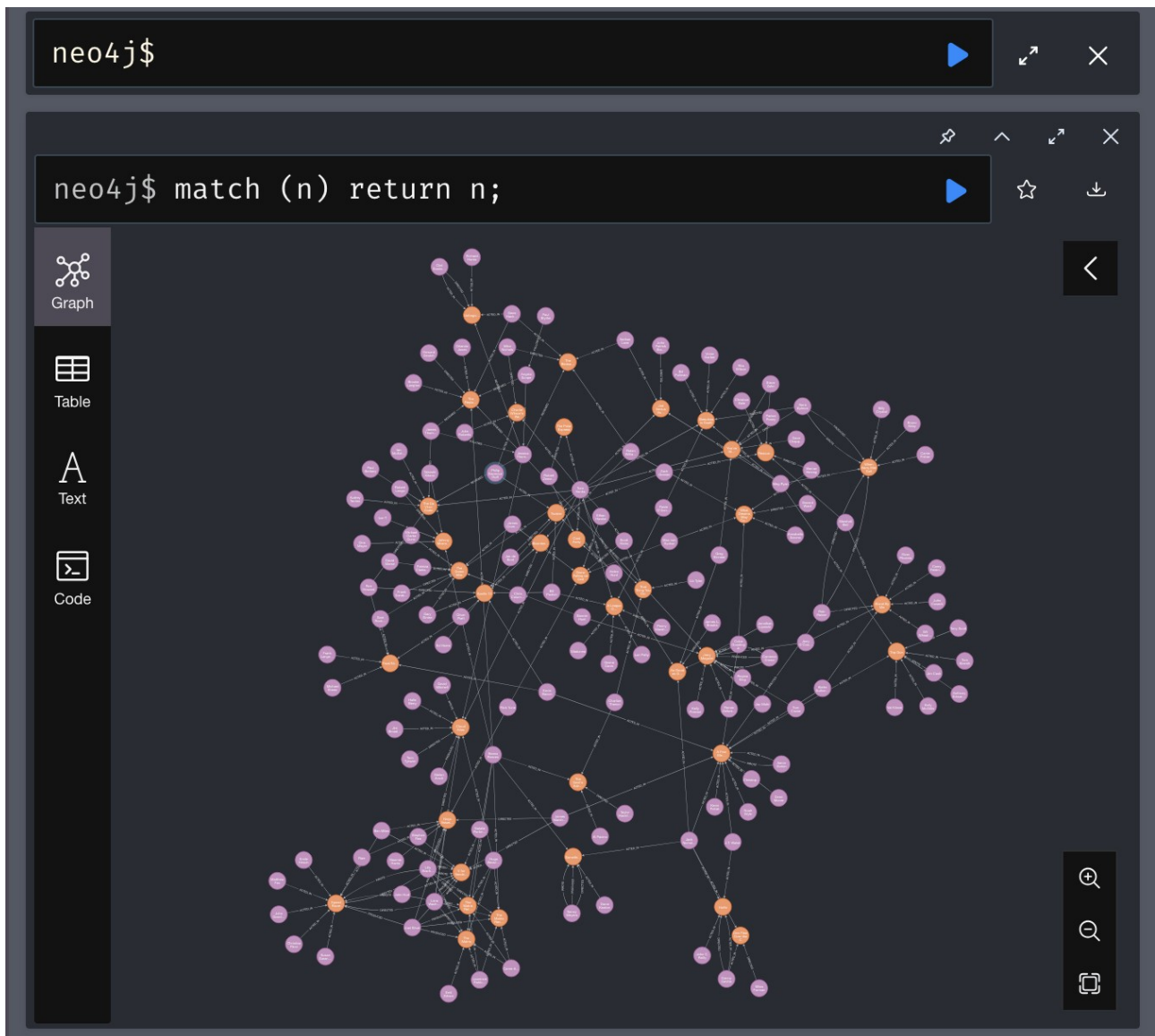


Then after running this command I was able to see the number of vertices, ie the nodes and the number of edges being the amount of relationships in the graph.

The image shows the Neo4j Desktop application interface. At the top, a command bar contains the prompt `neo4j$` and a play button. Below this, a query editor shows the Cypher query `neo4j$ match (n) with count(n) as numVertices mat...` with a play button and icons for saving, undo, redo, and download. The results pane on the left has three tabs: 'Table' (selected), 'Text', and 'Code'. The 'Table' view displays a table with two columns: **numVertices** and **numEdges**. The first row of data shows the value '1' under **numVertices**. The second row shows '171' under **numVertices** and '253' under **numEdges**. At the bottom of the results pane, a status message reads: 'Started streaming 1 records after 17 ms and completed after 18 ms.'

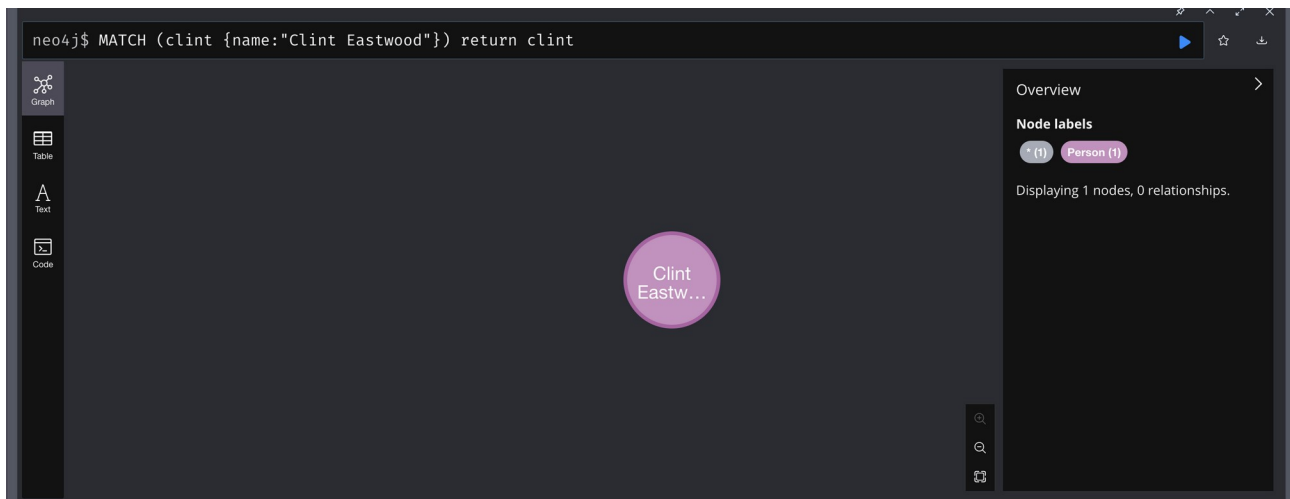
numVertices	numEdges
1	
171	253

Then to view all the nodes and their relationships I ran this command:

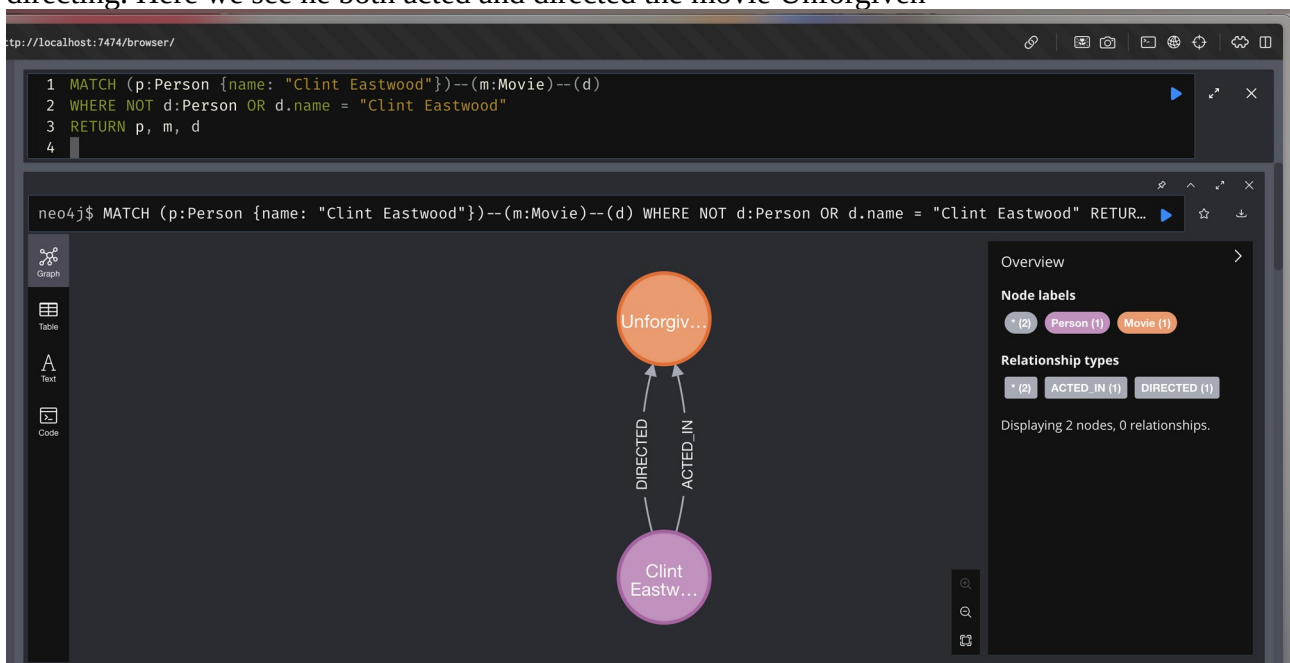


Now that our database is all setup and verified, we can start on the core db queries.

Before inserting the data relating to the movie Bridges of Mason County, I began checking if these nodes actually already existed for the director or actors. As seen above Clint Eastwood is already a person here so it stopped me from creating a dummy node.

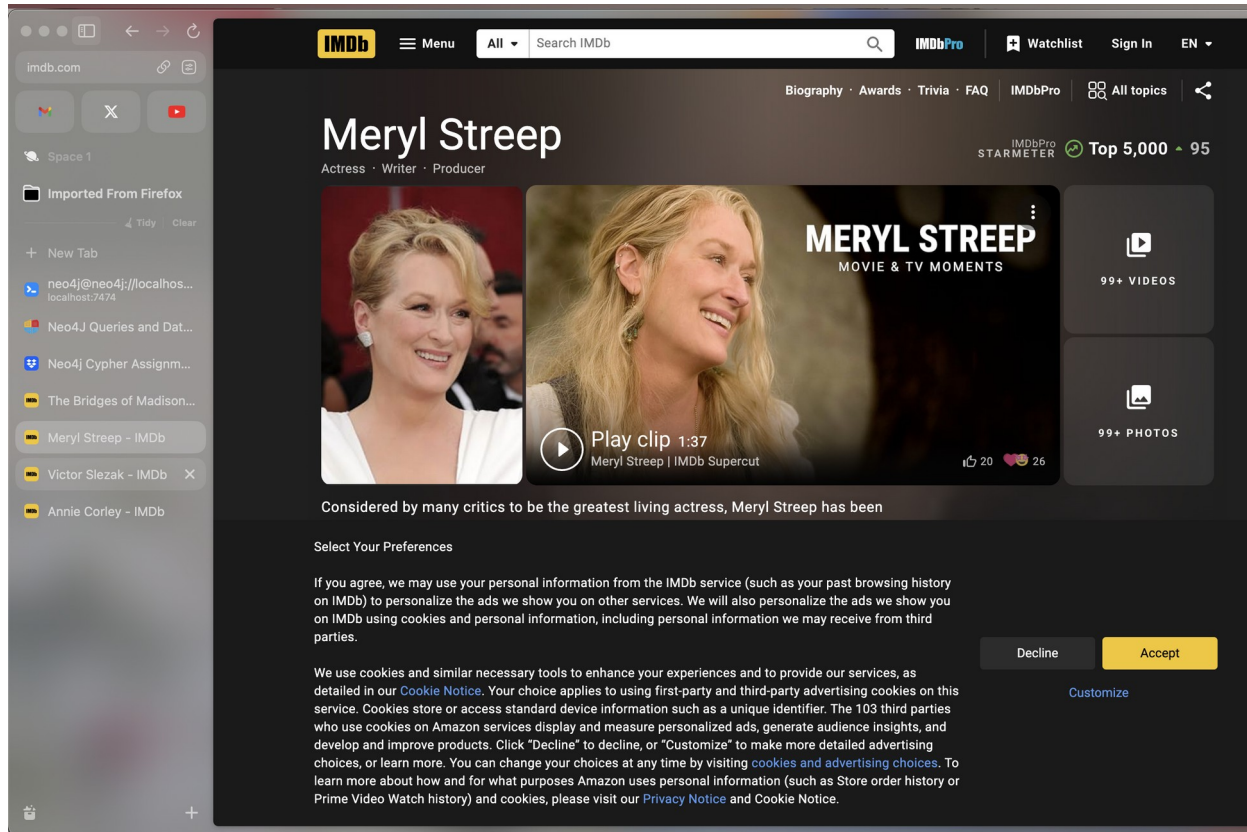


Then I checked what kind of connections Clint Eastwood already had in terms of acting and directing. Here we see he both acted and directed the movie Unforgiven

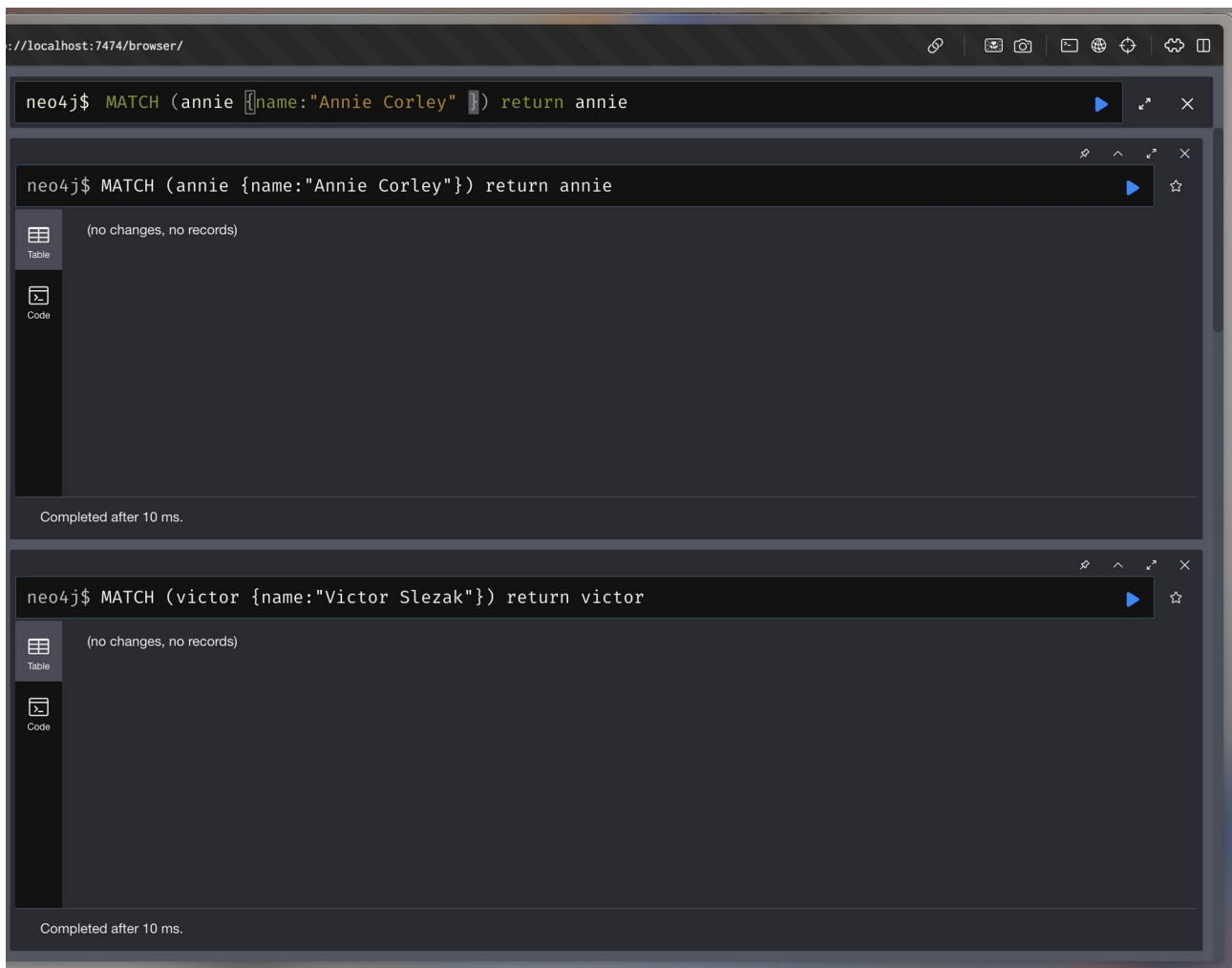


Then as seen below in the main tabs and sub tabs I picked the top three actors in this movie:

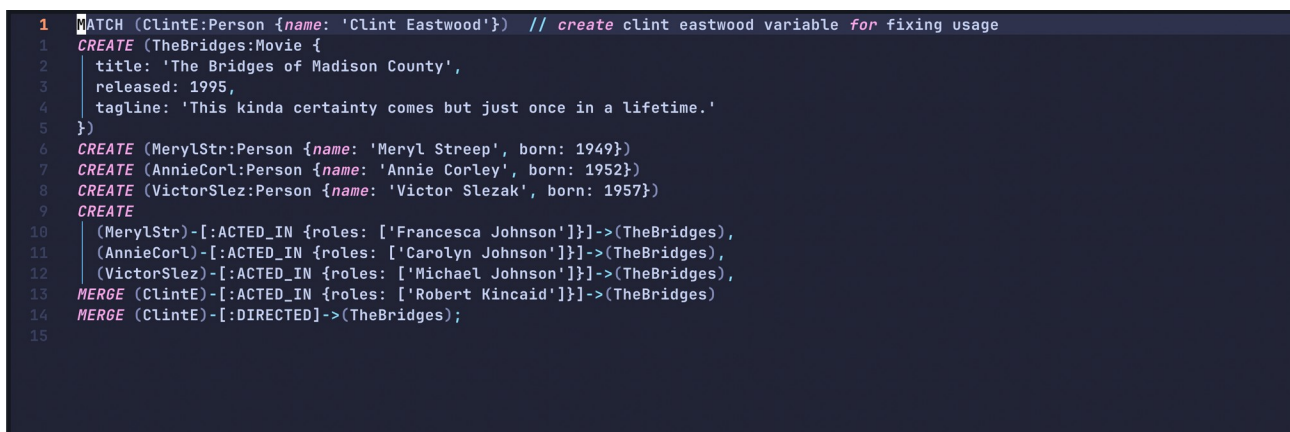
- Meryl Streep
- Victor Sizeak
- Annie Corley



Then I performed the same check as to make sure they wouldn't become dummy nodes:



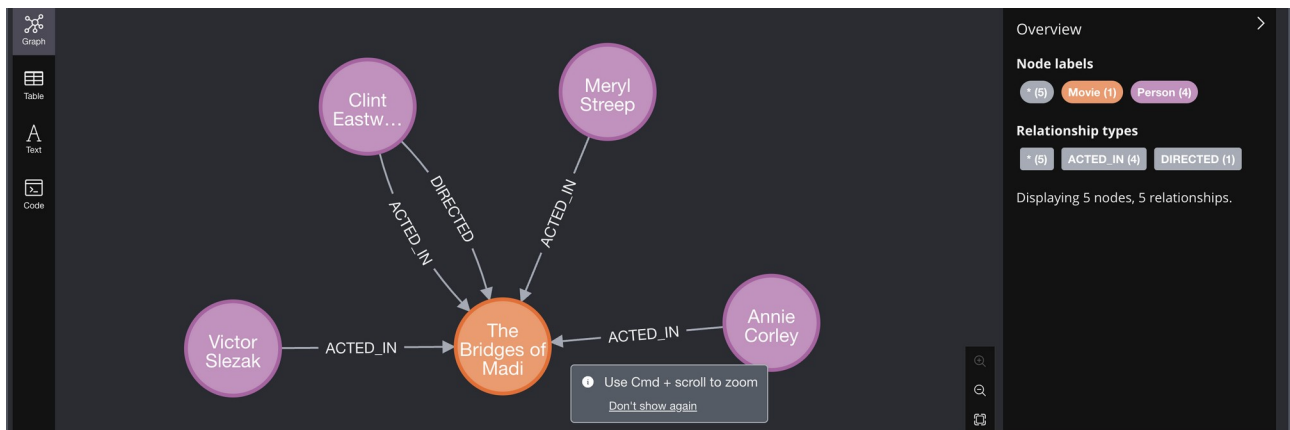
Each of them had no records so I created their vertices and connected them to the new Movie:



Rather than manually adding each entry, I decided in a separate .cypher file, to create the structure of the adding statement and then manually paste this in as to avoid dummy nodes.

```
neo4j$ CREATE (TheBridges:Movie { title: 'The Bridges of Madison County', released: 1995, tagline: 'This kinda certainty c... })
Added 4 labels, created 5 nodes, set 13 properties, created 5 relationships, completed after 104 ms.
```

Successfully created person and movie relationships.



Now we have no dummy nodes through the use of merge and making Clint Eastwood a variable in this query through a match statement.

2.

To find this we need to use a match statement that outputs three elements: - the movie itself, the actors in this movie and a person acting and directing.

This is my query for that:

```
(VictorSlez)-[:ACTED_IN {roles: ['Michael Johnson']}]>-(TheBridges),
MERGE (ClintE)-[:ACTED_IN {roles: ['Robert Kincaid']}]>-(TheBridges)
MERGE (ClintE)-[:DIRECTED]>-(TheBridges);

MATCH (m:Movie {title: 'The Bridges of Madison County'})-[r]-(p)
RETURN m, r, p;

-- Q 2

MATCH (justActor:Person)-[:ACTED_IN]>-(m:Movie)<-[:ACTED_IN]-(both:Person)
WHERE exists( (both)-[:DIRECTED]>-(m) )
RETURN justActor.name as Actor, both.name as `bothActingDirecting`, m.title as Movie
```

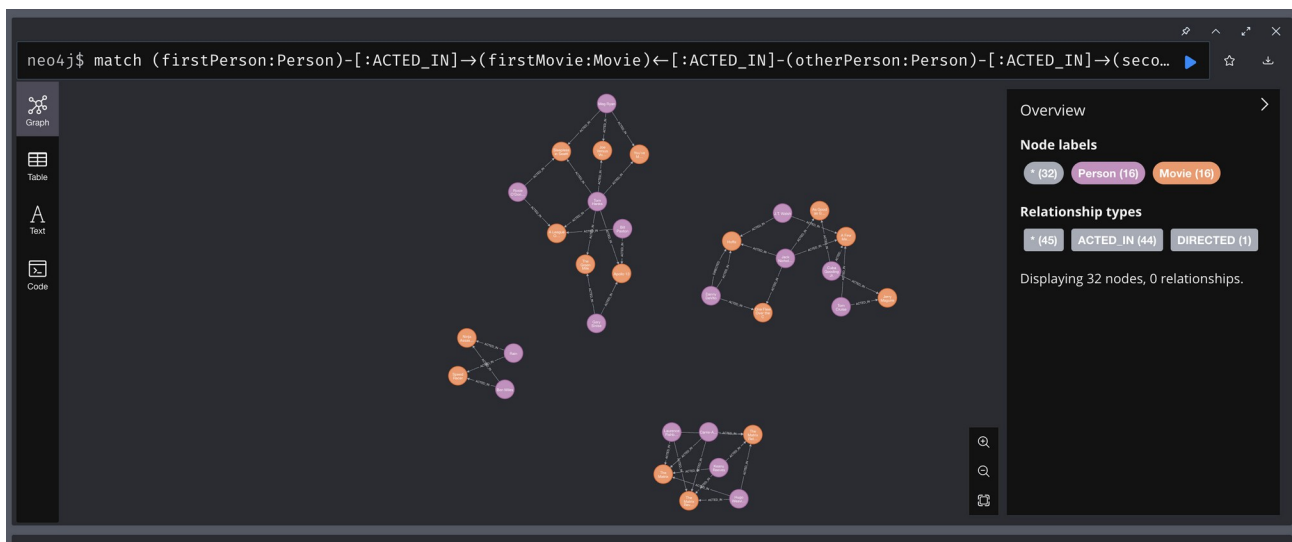
```
neo4j$ MATCH (justActor:Person)-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(both:Person) WHERE exists( (both)-[:DIRECTED]->(m...
```

	Actor	bothActingDirecting	Movie
1	"Charlize Theron"	"Tom Hanks"	"That Thing You Do"
2	"Liv Tyler"	"Tom Hanks"	"That Thing You Do"
3	"Gene Hackman"	"Clint Eastwood"	"Unforgiven"
4	"Richard Harris"	"Clint Eastwood"	"Unforgiven"
5	"Jack Nicholson"	"Danny DeVito"	"Hoffa"
6	"J.T. Walsh"	"Danny DeVito"	"Hoffa"
7			

Started streaming 10 records after 35 ms and completed after 37 ms.

This output does provide duplicates and is only available in text/table format, but in a node format, it would be increasingly difficult to read.

3.



```
-- Q3
MATCH (firstPerson:Person)-[:ACTED_IN]->(firstMovie:Movie)<-[:ACTED_IN]-(otherPerson:Person)-[:ACTED_IN]->(secondMovie:Movie)<-[:ACTED_IN]
return firstPerson,firstMovie,secondMovie,otherPerson

-- Q5
MATCH
```

main collegeWork command.sql 87% 29:5 12:

4.

```
MATCH (keanu:Person {name: "Keanu Reeves"})-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(co_actor:Person)
WITH m.title AS movieTitle, collect(DISTINCT co_actor.name) AS coActors
RETURN movieTitle, coActors
```



```
neo4j$ MATCH (keanu:Person {name: "Keanu Reeves"})-[:ACTED_IN]->(m:Movie)-[:ACTED_IN]-(co_actor:Person) WITH m.title AS movieTitle, collect(DISTINCT co_actor.name) AS coActors RETURN movieTitle, coActors
```

movieTitle	coActors
"Something's Gotta Give"	["Diane Keaton", "Jack Nicholson"]
"The Replacements"	["Brooke Langton", "Orlando Jones", "Gene Hackman"]
"Johnny Mnemonic"	["Ice-T", "Dina Meyer", "Takeshi Kitano"]
"The Devil's Advocate"	["Al Pacino", "Charlize Theron"]
"The Matrix Revolutions"	["Hugo Weaving", "Laurence Fishburne", "Carrie-Anne Moss"]
"The Matrix Reloaded"	["Hugo Weaving", "Laurence Fishburne", "Carrie-Anne Moss"]

Started streaming 7 records after 7 ms and completed after 10 ms.

5.

```
neo4j$ MATCH (p:Person)-[:REVIEWED]->(m:Movie) WITH min(r.rating) AS minRating MATCH (p:Person)-[:REVIEWED]->(m:Movie) WHERE r.rating = minRating WITH m LIMIT 1 DETACH DELETE m;
```

Deleted 1 node, deleted 5 relationships, completed after 10 ms.

Deleted 1 node, deleted 5 relationships, completed after 10 ms.

For deleting the movie with the lowest rating, we need to delete both the movie itself, and the leftover relationships:

I kept a limit of 1 just so it only deletes the movie with THE lowest rating.

```
7 -- Q5
8
9 MATCH (p:Person)-[:REVIEWED]->(m:Movie)
10 WITH min(r.rating) AS minRating
11 MATCH (p:Person)-[:REVIEWED]->(m:Movie)
12 WHERE r.rating = minRating
13 WITH m
14 LIMIT 1
15 DETACH DELETE m;
```

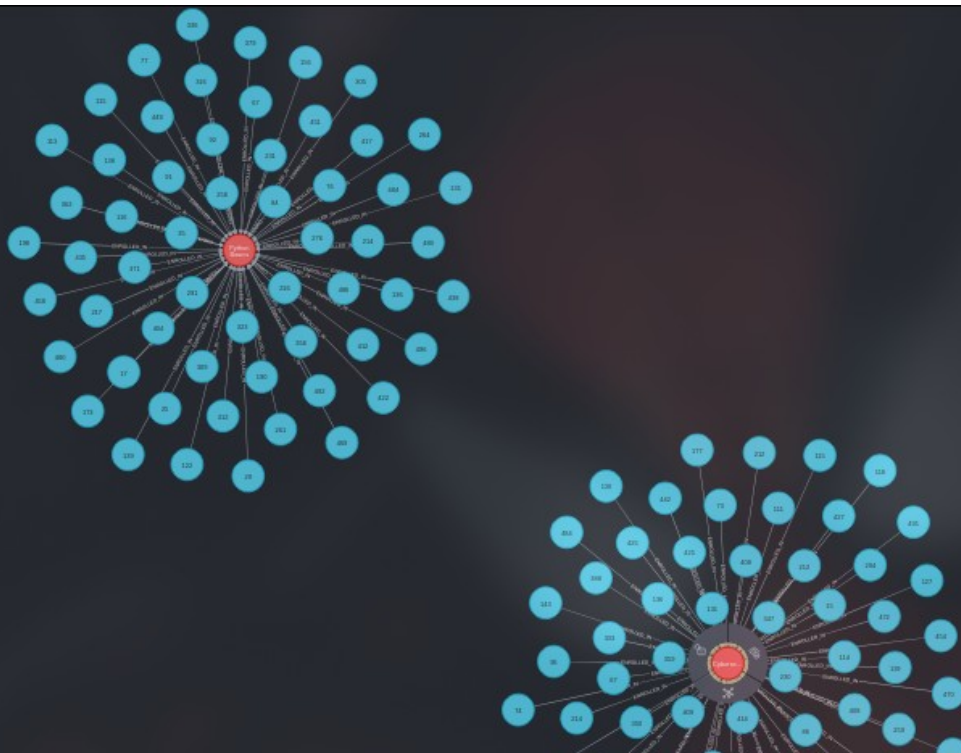
Part 2 (Personalized Learning Database)

- Before simply importing this dataset, we first need to decipher the csv data. From the pdf, we have two distinct entities, students and courses. I initially assumed there may be occurrences of students being listed for multiple courses, but upon looking through the file, the ids for students only ever appear once. I was able to then use this command and import the csv:

```

47 -- Import for next question
48 LOAD CSV WITH HEADERS FROM "file:///personalized_learning_dataset.csv" as row
49 MERGE (s:Student {
50     studentId: row.Student_ID,
51     age: toInteger(row.Age),
52     gender: row.Gender,
53     educationLevel: row.Education_Level,
54     timeSpentOnVideos: toInteger(row.`Time_Spent_on_Videos`),
55     quizAttempts: toInteger(row.Quiz_Attempts),
56     quizScores: toFloat(row.Quiz_Scores) / 100,
57     forumParticipation: toInteger(row.Forum_Participation),
58     assignmentCompletionRate: toFloat(row.Assignment_Completion_Rate) / 100,
59     engagementLevel: row.Engagement_Level,
60     finalExamScore: toFloat(row.Final_Exam_Score) / 100,
61     learningStyle: row.Learning_Style,
62     feedbackScore: toFloat(row.Feedback_Score),
63     dropOutChance: row.Dropout_Likelihood
64 })
65 MERGE (c:Course {name: row.Course_Name})
66 CREATE (s)-[:ENROLLED_IN]->(c);

```



```

"file:///personalized_learning_dataset.csv" as row MERGE (s:Student { student
.40005 properties, created 10000 relationships, completed after 40248 ms.

```

As seen below, we have hundreds of students each connected to python basics and cyber-security respectfully.

1.

2.

neo4j\$ MATCH (s:Student)-[:ENROLLED_IN]→(c:Course) WHERE s.engagementLevel < "Low" WITH c.nam... ▶ ☆ ⬇

Table

Text

Code

"Data Science"	"Undergraduate"	0.99
"Data Science"	"High School"	0.99
"Data Science"	"Postgraduate"	0.99
"Web Development"	"Postgraduate"	0.99
"Web Development"	"High School"	0.99
"Web Development"	"Undergraduate"	0.99
"Cybersecurity"	"Undergraduate"	0.99
"Cybersecurity"	"High School"	0.99
"Cybersecurity"	"Postgraduate"	0.99
"Machine Learning"	"Undergraduate"	0.99
"Machine Learning"	"Postgraduate"	0.99

MAX COLUMN WIDTH:

Table

Text

Code

(:Student {timeSpentOnVideos: 53,quizAttempts: 3,gender: "Female",learningStyle: "Visual",finalExamScore: 0.42,studentId:
(:Student {timeSpentOnVideos: 171,quizAttempts: 4,gender: "Female",learningStyle: "Visual",finalExamScore: 0.51,studentId:
(:Student {timeSpentOnVideos: 174,quizAttempts: 2,gender: "Female",learningStyle: "Visual",finalExamScore: 0.53,studentId:
(:Student {timeSpentOnVideos: 356,quizAttempts: 2,gender: "Female",learningStyle: "Visual",finalExamScore: 0.78,studentId:
(:Student {timeSpentOnVideos: 423,quizAttempts: 4,gender: "Female",learningStyle: "Visual",finalExamScore: 0.54,studentId:
(:Student {timeSpentOnVideos: 341,quizAttempts: 3,gender: "Female",learningStyle: "Visual",finalExamScore: 0.99,studentId:
(:Student {timeSpentOnVideos: 137,quizAttempts: 3,gender: "Female",learningStyle: "Visual",finalExamScore: 0.62,studentId:
(:Student {timeSpentOnVideos: 186,quizAttempts: 3,gender: "Female",learningStyle: "Visual",finalExamScore: 0.6,studentId:
(:Student {timeSpentOnVideos: 87,quizAttempts: 2,gender: "Female",learningStyle: "Visual",finalExamScore: 0.96,studentId:
(:Student {timeSpentOnVideos: 417,quizAttempts: 2,gender: "Female",learningStyle: "Visual",finalExamScore: 0.79,studentId:

MAX COLUMN WIDTH:

```
MATCH (s:Student)-[:ENROLLED_IN]→(c:Course)
WHERE s.engagementLevel < "Low"
WITH c.name AS CourseName, s.educationLevel AS EducationLevel, MAX(s.assignmentCompletionRate) AS completionRate
RETURN DISTINCT CourseName, EducationLevel, MaxCompletionRate
ORDER BY completionRate DESC;
```

3.

```
1 MATCH (s:Student)-[:ENROLLED_IN]-(c:Course)
2 ORDER BY s.feedbackScore DESC
3 LIMIT 1
4 RETURN c.name, s.feedbackScore;
5
```

	c.name	s.feedbackScore
1	"Cybersecurity"	5.0

This simply gives the score any has achieved however a more accurate depiction would be an average:

```
1 MATCH (s:Student)-[:ENROLLED_IN]-(c:Course)
2 WITH c,AVG(s.feedbackScore) as avgFbck
3 ORDER BY avgFbck DESC
4 LIMIT 1
5 RETURN c.name, avgFbck;
6
```

	c.name	avgFbck
1	"Python Basics"	3.045135406218659

Started streaming 1 records after 1 ms and completed after 7 ms.

4.

```
1 MATCH (s:Student)-[:ENROLLED_IN]→(c:Course)
2   WHERE c.name = "Cybersecurity" AND s.engagementLevel="Low"
3   |   AND s.finalExamScore < 0.4 AND s.gender = "Male"
4   return COUNT(s);
```

neo4j\$ MATCH (s:Student)-[:ENROLLED_IN]→(c:Course) WHERE c.name = "Cybersecurity" AND s.engage...

	COUNT(s)
1	29

Started streaming 1 records after 5 ms and completed after 6 ms.

5.

```
1 MATCH (s:Student)
2   WHERE s.dropOutChance = "Yes" AND s.engagementLevel="Low"
3   WITH s,collect(DISTINCT s.learningStyle) as learnStyle, collect(DISTINCT s.educationLevel)
4   as edLevel
   return learnStyle, edLevel, COUNT(s);
```

	learnStyle	edLevel	COUNT(s)
7	["Kinesthetic"]	["Postgraduate"]	23
8	["Auditory"]	["Postgraduate"]	28
9	["Reading/Writing"]	["High School"]	30
10	["Reading/Writing"]	["Undergraduate"]	42
11	["Kinesthetic"]	["High School"]	26
12	["Visual"]	["Postgraduate"]	13

Started streaming 12 records after 7 ms and completed after 18 ms.