

See the slides on Neo4j.

Let's use the movie database included in Neo4j.

Our first goal is to show recommendations for other actors to work with or similar movies to watch. By following the meaningful relationships between actors and movies, we can determine occurrences of actors working together, the frequency of actors working with one another, and the movies they have in common in the graph. This structure forms the basis for many recommendation engines.

Our second goal is to use some graph algorithm.

Let's start the Movies sandbox (<https://sandbox.neo4j.com/>) and let's type the following commands into the Neo4j Browser command line.

Please note that lines starting with // are comments.

```
call db.schema.visualization()
```

```
MATCH (tom:Person {name: 'Tom Hanks'})
RETURN tom
```

```
MATCH (tom:Person {name: 'Tom Hanks'})-[r:ACTED_IN]->(movie:Movie)
RETURN tom, r, movie
```

```
MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(coActor:Person)
RETURN coActor.name
```

```
MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(coActor:Person)
RETURN distinct coActor.name
```

```
MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(movie1:Movie)<-
[:ACTED_IN]-(coActor:Person)-[:ACTED_IN]->(movie2:Movie)<-
[:ACTED_IN]-(coCoActor:Person)
RETURN coCoActor.name
```

// What is wrong with the previous query? Let's improve it:

```
MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(movie1:Movie)<-
[:ACTED_IN]-(coActor:Person)-[:ACTED_IN]->(movie2:Movie)<-
[:ACTED_IN]-(coCoActor:Person)
WHERE tom <> coCoActor AND NOT (tom)-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(coCoActor)
RETURN coCoActor.name
```

```

MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(movie1:Movie)<-
[:ACTED_IN]-(coActor:Person)-[:ACTED_IN]->(movie2:Movie)<-
[:ACTED_IN]-(coCoActor:Person)
WHERE tom <> coCoActor AND NOT (tom)-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(coCoActor)
RETURN coCoActor.name, count(coCoActor) as frequency
ORDER BY frequency DESC
LIMIT 5

```

```

MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(movie1:Movie)<-
[:ACTED_IN]-(coActor:Person)-[:ACTED_IN]->(movie2:Movie)<-
[:ACTED_IN]-(cruise:Person {name: 'Tom Cruise'})
WHERE NOT (tom)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cruise)
RETURN tom, movie1, coActor, movie2, cruise

```

To show also the links on the graph:

```

MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(movie1:Movie)<-
[:ACTED_IN]-(coActor:Person)-[:ACTED_IN]->(movie2:Movie)<-
[:ACTED_IN]-(cruise:Person {name: 'Tom Cruise'})
WHERE NOT (tom)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cruise)
WITH tom, movie1, coActor, movie2, cruise
MATCH path = (tom)-[:ACTED_IN]->(movie1)<-[:ACTED_IN]-(coActor)-
[:ACTED_IN]->(movie2)<-[:ACTED_IN]-(cruise)
RETURN tom, movie1, coActor, movie2, cruise, relationships(path) AS
links;

```

```

// Degree centrality
// Let's find which actor has acted in the most movies using degree
centrality

```

```

// First, create a graph projection

```

```

CALL gds.graph.project(
  'actorGraph',
  ['Person', 'Movie'],
  'ACTED_IN'
);

```

```

// Then, run the degree centrality algorithm on the projected graph

```

```

CALL gds.degree.stream('actorGraph')
YIELD nodeId, score
RETURN
  gds.util.asNode(nodeId).name AS actorName,
  score AS numberOfMoviesActedIn
ORDER BY numberOfMoviesActedIn DESCENDING, actorName LIMIT 5

```

```

// Shortest path algorithm
// What is the shortest path between Kevin Bacon and Clint Eastwood?

```

```

// Create graph projection

CALL gds.graph.project(
  'actorGraph2',
  ['Person','Movie'],
  {
    ACTED_IN:{orientation:'UNDIRECTED'},
    DIRECTED:{orientation:'UNDIRECTED'}
  }
);

// Find shortest path

MATCH (kevin:Person {name: 'Kevin Bacon'})
MATCH (clint:Person {name: 'Clint Eastwood'})
CALL gds.shortestPath.dijkstra.stream(
  'proj2',
  {
    sourceNode: kevin,
    targetNode: clint
  }
)
YIELD sourceNode, targetNode, path
RETURN sourceNode, targetNode, nodes(path) AS path,
relationships(path) AS links;

```