# Assignment 1 - Knowledge Graph Schema Design - Report

This report describes the design processes of the movie knowledge graph applied in both tasks (task1, task2) and the queries used to build its key elements and assess its functionality.

The scope of both tasks is to expand the existing knowldege graph in order to be able to answer to the below competency questions.

Competency Questions

1. Which movies have been produced in the United States?
2. Which movies are comedies and which are romantic comedies?
3. What rating did a reviewer give to a given movie and what was their review text?

In task 1 the modeling and graph construction must be implemented in Neo4j, while in task 2 it must be done in RDF(S)/OWL - Protege.

In the next two chapters we will look at the process and design decisions made in both environments to achieve the desired results.

## Task 1 - Neo4j

**Which movies have been produced in the United States?**

To do this, we can choose between two ways.

- Create a new attribute for all instances of movies containing the countries in which they were produced.
- Create a new class to include the countries and then a new relationship to link the movie instances to the country instances to indicate where the movies were produced.

The first way is faster and simpler than the other, as it does not require complex implementation steps. When the goal is to quickly record and search for information, it is the best choice. However, if there are cases where you will need to know more information about countries, such as regions and languages, or if you need to create additional relationships between country instances and some other type of instances, then the second way is definitely the way to go.

Because, in our case we do not need to use the country information to answer some other questions and also in Neo4j you have the possibility to assign a list of values to a specific attribute facilitating cases where there is a movie produced in more than one country, it is enough to select and apply the first design concept.

To implement this I used the existing data in the graph and created a new attribute called "produced_in" under the movie instances (labelled as "Movie") and populated it with a list of countries indicating where they were produced, using the Cypher queries as follows. Just to mention that the assigned countries are dummy data and do not correspond to the real countries where they were produced.

```
MATCH (m:Movie)
WHERE m.title IN [
"The Matrix",
"The Matrix Reloaded",
"The Matrix Revolutions",
"The Devil's Advocate",
"A Few Good Men",
"Top Gun",
"Jerry Maguire",
"Stand By Me",
"As Good as It Gets",
"What Dreams May Come",
"Snow Falling on Cedars",
"You've Got Mail",
"Sleepless in Seattle"
]
SET m.produced_in = ["United States", "United Kingdom"]

MATCH (m:Movie)
WHERE m.title IN [
"Joe Versus the Volcano",
"When Harry Met Sally",
"That Thing You Do",
"The Replacements",
"RescueDawn",
"The Birdcage",
"Unforgiven",
"Johnny Mnemonic",
"Cloud Atlas",
"The Da Vinci Code",
"V for Vendetta",
"Speed Racer"
]
SET m.produced_in = ["United States"]

MATCH (m:Movie)
WHERE m.title IN [
"Ninja Assassin",
"The Green Mile",
"Frost/Nixon",
"Hoffa",
"Apollo 13",
"Twister",
"Cast Away",
"One Flew Over the Cuckoo's Nest",
```

```
"Something's Gotta Give",
"Bicentennial Man",
"Charlie Wilson's War",
"The Polar Express",
"A League of Their Own"
]
SET m.produced_in = ["Germany"]
```

In order to verify that the competency question was answered correctly, I wrote the following Cypher query which finds movies produced in the United States and returns the title of the movies and the corresponding country or countries in case there are more than one country.

```
MATCH (m:Movie)
WHERE "United States" IN m.produced_in
RETURN m.title, m.produced_in
```

**Which movies are comedies and which are romantic comedies?**

To answer this question I decided that the best approach is to create a class hierarchy. The term comedy is a type of film while romantic comedy is a type of comedy film such as dark comedy. In general, in the film industry there are certain recognized film genres which can be further broken down into subgenres. For example, within the comedy genre, as I mentioned above, there are subgenres such as romantic comedy and dark comedy, while within the action genre there are other subgenres such as spy action or superhero action and so on.

To implement this approach I took the following steps (NOTE: the data used are dummy):

1. I created a new class called "MovieType". Each movie genre is a subclass of the "MovieType" class and the instances of each genre correspond to its subgenres. Through the following Cypher query I created the class "MovieType", the subclasses "Comedy", "Drama", "Action" and "Adventure" and some instances of them.
   ```
   CREATE
   (t1:MovieType:Comedy {name:"Romantic Comedy"}),
   (t2:MovieType:Comedy {name:"Dark Comedy"}),
   (t3:MovieType:Drama {name:"Legal Drama"}),
   (t4:MovieType:Action {name:"Spy Action"}),
   (t5:MovieType:Action {name:"Superhero Action"}),
   (t6:MovieType:Adventure {name:"Space Adventure"})
   ```

2. I created a new relation called "IS_TYPE_OF", which connects the instances of movies to the instances of the classes "Comedy", "Drama", "Action" and "Adventure", which, due to the class hierarchy, are also instances of the class "MovieType". Through the following Cypher query I achieved the above and also for each existing movie I applied a link to some "MovieType" instances.
   ```
   MATCH (m:Movie), (t:Comedy)
   WHERE m.title IN [
   "The Matrix",
   ```

```
"The Matrix Reloaded",
"The Matrix Revolutions",
"The Devil's Advocate",
"A Few Good Men",
"Top Gun",
"Jerry Maguire",
"Stand By Me",
"As Good as It Gets",
"What Dreams May Come",
"Snow Falling on Cedars",
"You've Got Mail",
"Sleepless in Seattle"
] AND
t.name = "Romantic Comedy"
CREATE (m)-[r:IS_TYPE_OF]->(t)

MATCH (m:Movie), (t:Comedy)
WHERE m.title IN [
"Joe Versus the Volcano",
"When Harry Met Sally",
"That Thing You Do",
"The Replacements",
"RescueDawn"
] AND
t.name = "Dark Comedy"
CREATE (m)-[r:IS_TYPE_OF]->(t)

MATCH (m:Movie), (t:Drama)
WHERE m.title IN [
"The Birdcage",
"Unforgiven",
"Johnny Mnemonic",
"Cloud Atlas",
"The Da Vinci Code",
"V for Vendetta",
"Speed Racer"
]
CREATE (m)-[r:IS_TYPE_OF]->(t)

MATCH (m:Movie), (t:Action)
WHERE m.title IN [
"Ninja Assassin",
"The Green Mile",
"Frost/Nixon",
"Hoffa"
] AND
t.name = "Spy Action"
CREATE (m)-[r:IS_TYPE_OF]->(t)

MATCH (m:Movie), (t:Action)
WHERE m.title IN [
"Apollo 13",
```

```
"Twister",
"Cast Away",
"One Flew Over the Cuckoo's Nest",
"Something's Gotta Give"
] AND
t.name = "Superhero Action"
CREATE (m)-[r:IS_TYPE_OF]->(t)

MATCH (m:Movie), (t:Adventure)
WHERE m.title IN [
"Bicentennial Man",
"Charlie Wilson's War",
"The Polar Express",
"A League of Their Own"
]
CREATE (m)-[r:IS_TYPE_OF]->(t)
```

In order to verify that the competency question was answered correctly, I wrote the following Cypher queries, where the first one finds the romantic comedies and returns the title and the type of the movies, while the second one finds all comedies (romantic and dark) and returns again the title and the type of the movies.

```
MATCH (m:Movie)-[r:IS_TYPE_OF]->(t:Comedy)
WHERE t.name = "Romantic Comedy"
RETURN m.title, t.name

MATCH (m:Movie)-[r:IS_TYPE_OF]->(t:Comedy)
RETURN m.title, t.name
```

**What rating did a reviewer give to a given movie and what was their review text?**

To answer to this question i decided to create a new relation called "REVIEWED" linking "Person" instances with "Movie" instances. The relation has two attributes where the first one is called "rating" and holds the rating that the reviewer gave to a specific movie, while the second one called "review_text" and stores the corresponding review from the reviewer for the movie. To do this i run the following Cypher query creating the relation and some dummy data.

```
MATCH (p:Person), (m:Movie)
WHERE p.name IN
[
"Keanu Reeves",
"Carrie-Anne Moss",
"Laurence Fishburne",
"Hugo Weaving",
"Emil Eifrem",
"Charlize Theron",
"Al Pacino",
"Tom Cruise",
"Jack Nicholson",
```

```
"Demi Moore"
] AND
m.title IN
[
"The Matrix",
"The Matrix Reloaded",
"The Matrix Revolutions",
"The Devil's Advocate",
"A Few Good Men",
"Top Gun"
]
CREATE (p)-[r:REVIEWED {rating: 8, review_text:"Interesting
movie!"}]->(m)
```

I decided to link the instances of the "Person" class directly to those of the "Movie" class and not to create a subclass of "Person" named for example "Reviewer" and use its instances, as I noticed that for the "DIRECTED" and "PRODUCED" relations the respective classes were not created but the original one which is "Person" was used directly. This allows a "Person" instance that plays many roles (director, actor, reviewer etc) not to be repeated in our graph and to distinguish each time its role from the relation with which it is connected to the other instances of the classes.

In order to verify that the competency question was answered correctly, I wrote the following Cypher query, which finds the review that Emil Eifrem gave to the movie The Matrix and returns the corrsponding rating and review text.

```
MATCH (p:Person)-[r:REVIEWED]->(m:Movie)
WHERE p.name = "Emil Eifrem" AND m.title = "The Matrix"
RETURN r.rating AS Rating, r.review_text AS Review
```

# Task 2 - RDF(S)/OWL - Protege

**Which movies have been produced in the United States?**

Here, unlike the first task, I decided to create a "Country" class and link the instances to those of the "Movie" class using an appropriate relation, as I noticed that it is not so easy for a data property to take a list of values as a value, which I had done in Neo4j. After creating the class, I added some "Country" instances consisting of a data property called "name" containing the name of the country, some instances under the "Movie" class consisting of three data properties called "title", "tagline" and "released" and finally i defined a new relation called "PRODUCED_IN" that takes as domains the "Movie" instances and as ranges the instances from the "Country" class and connected the instances.

To verify that the competency question was answered correctly, I wrote the following SPAQL query, which finds the movies that were produced in United States and prints their title.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:<http://neo4j.com/voc/movies#>

SELECT ?movie
WHERE {
    ?movie rdf:type ont:Movie.
    ?movie ont:PRODUCED_IN ont:United_States.
}
```

**Which movies are comedies and which are romantic comedies?**

As i explained in the relevant question of the previous task, here i created a new Class called "MovieType" and under this i added two new sub-classes called "Comedy" and "Action". After that, i made some instances under the sub-classes (which also are instances of the parent class "MovieType") consisting of one data attribute called "name" that holds the name of the film type and then i defined the corresponding relation called "IS_TYPE_OF" connecting "Movie" instances (domains) to "MovieType" instances (ranges) and connected them. Just to mention that the data used are dummy.

To verify that the competency question was answered correctly, I wrote the following SPAQL queries, where the first one finds the comedy movies and prints their title while the second one fnds the romantic comedy films and returns their title.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:<http://neo4j.com/voc/movies#>

SELECT ?movie
WHERE {
    ?movie rdf:type ont:Movie.
    ?comedy rdf:type ont:Comedy.
    ?movie ont:IS_TYPE_OF ?comedy.
}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:<http://neo4j.com/voc/movies#>

SELECT ?movie
WHERE {
    ?movie rdf:type ont:Movie.
```

```
    ?movie ont:IS_TYPE_OF ont:Romantic_Comedy.
}
```

**What rating did a reviewer give to a given movie and what was their review text?**

Here, unlike the first task, I decided to create a new class under the "Person" class (just another design strategy) called "Reviewer". I added some instances that have only one data property called "name" containing the name of the reviewer. Because, we need to know the rating and review each reviewer gave to the movies and RDF(S)/OWL does not offer the ability to have properties under relationships like in Neo4j, to achieve this I had to define a new class called "MovieRating" that holds the reviewers' movie reviews. Its instances have two data properties called "rating" and "review" that store the rating and the review text respectively. Now, to assign reviews to reviewers and movies, I had to create two new relationships. The first one connects the "Person" (parent class of "Reviewer") instances (domains) to the "MovieRating" instances (ranges) and is called "REVIEWED", while the second one connects the "MovieRating" instances (domains) to the "Movie" instances (ranges) and is called "REVIEWED_BY". Just to mention that the data used are dummy.

To verify that the competency question was answered correctly, I wrote the following SPAQL query, that finds the review that Reviewer 1 gave to the movie "The Matrix" and returns the name of the reviewer, the movie title, the rating and the review text.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:<http://neo4j.com/voc/movies#>

SELECT ?reviewer ?movie ?rating ?review
WHERE {
    ?reviewer rdf:type ont:Reviewer.
    ?reviewer ont:name "Reviewer 1".

    ?movieRating rdf:type ont:MovieRating.
    ?reviewer ont:REVIEWED ?movieRating.

    ?movie rdf:type ont:Movie.
    ?movieRating ont:REVIEWED_BY ?movie.
    ?movie ont:title "The Matrix".

    ?movieRating ont:rating ?rating.
    ?movieRating ont:review ?review.
}
```