Data, Systems and Ontology Paper: An Awesome Linked Open Movie Database

Andrei Sili, Saranots Tzortzis

[1] andrei.sili@student.uva.nl
[2] sarantos_tzortzis@icloud.com
https://github.com/sarantinio/Semantic-Web-RDFs.git
April 18th, 2018

Abstract. In this paper we present a system for integrating movie data from 2 non-RDF and 2 RDF sources. We focus mainly on the challenges faced when building the Knowledge Graph, comparing different methods. In achieving this, we develop our own processing pipeline, which can be found at:
https://github.com/AndreiMariusSili/KnowledgeRepresentationOnTheWeb.
In this paper, we use the running example of the Ice Age movie, but all of the steps detailed for this resource applies to any movie or actor resource in our dataset as well.

1 Introduction

Anyone with baseline tech savviness will be aware of the many movie repositories available online. The Internet Movie Database is probably the most famous, but may other such sources exist. Among the open examples we count Rotten Tomatoes, The Movie Database, while other sources such are Netflix, that could provide exceptional insights, remain private. Another very important knowledge base for movies is Wikipedia, as for most other general knowledge topics.

Although so much information is available online about movies, it is rather segmented and contained in silos. IMDB holds great data applicable to any movie, like synopsis, cast, director, financial success. TMDB and Wikipedia hold community curated content, but with different focus. TMDB is more geared toward the viewer preferences, while Wikipedia presents mostly factual information.

Our goal in this paper is to present a proof of concept of how these datasets could be unified to provide for a seamless experience for the user browsing through movie data. With our approach, the user could transition seamlessly between professionally curated content, to rumors and opinions provided by the community. To our knowledge, nothing of the sort has been attempted so far.

The bulk of this paper is focused on describing the process of creating a data warehouse for movies. We explore data sources both in RDF and non-RDF formats. The non-RDF format will be processed into RDF before being brought together with the other datasets. We also observe a need for RDF formats to be cleaned before they can be used properly. This includes removing broken links, and clearing some irrelevant or outdated fields. An overview of this process is provided in Figure 1.

Another critical task is to link the data sources according to some shared unique identifiers. This will prove somewhat difficult, since datasets do not share one identifier across all, so a trade-off solution will need to be adopted. We will achieve this linking through the use of a vocabulary that specifies relationship between properties from different classes, and between properties and the classes to which they belong. A mock-up of such a app is provided in the repository as a YASGUI Data Story.

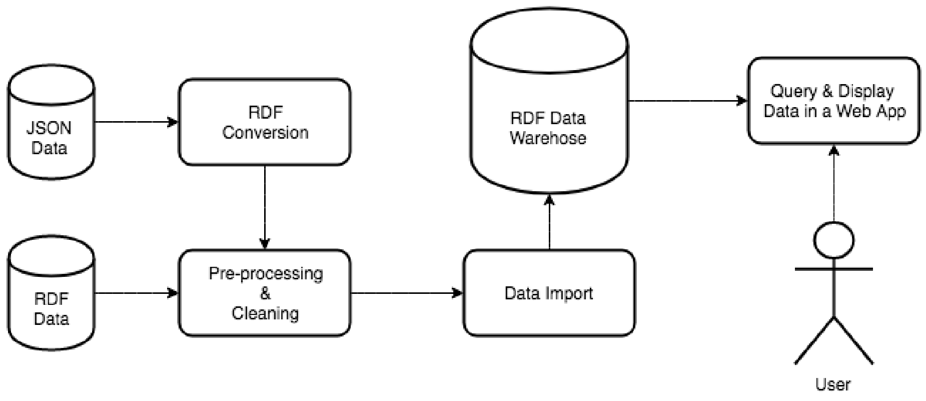


Fig. 1. Data Processing Pipeline

2 Scenario / Use Case

As hinted at in the introduction, our application is geared towards the avid movie-watcher. The app would provide a vast compendium of movie data, that allows a user to research a particular movie in depth but also to explore potential new movies. However, because our collection of movies is also wide, not only deep, a novice movie-watcher can also benefit by getting an overview of different genres.

Let's consider the scenario of the avid movie-watcher. Entering the app, he or she might be presented with a list of movies that are that are best suited for his or her taste. Exploring that movie will provide standard IMDB information, such as cast, director, synopsis, and revenue. However, this information will be augmented by more user-centric data coming from TMDB, as well as interesting community curated facts from Wikipedia, such as the historical context of the movie, or in depth analyses of the plot.

On the other hand, a novice watcher might want to first get an overview of the most popular movies by genre, and pick one at a glance. This is easily achievable given our data structure. Other kinds of statistics can also be gathered, such as most popular directors and actors and the movies that they have starred in.

3 Data sources

As mentioned before, we make use of both RDF and non-RDF data. All 4 data sources are imperfect and required some form of pre-processing before being imported into a Stardog repository as described in the following pages.

Non-RDF data is acquired from 2 public facing APIs. These are The Movie Database (TMDB) and the Open Movie Database(OMDB). TMDB is a service that provides, among others, data on viewer preference, while the OMDB provides data scraped from IMDB through an easily consumable API. We chose to obtain OMDB data as well since this is more expansive than the data provided by TMDB alone.

The first hurdle to cross in data collection was that OMDB only allows getting one movie at a time, based on the movie title or IMDB ID. This is what prompted the use of another service for acquiring the collection of movies to search for. We extracted a list of $20,000$ most recent and most popular movies, along with some user preference information, such as the average voting score and a popularity measure from TMDB. These had to be matched to the OMDB resources, which could be done by IMDB ID or by title. Since the IMDB ID was not available to us from TMDB, we chose to match on title, in order to obtain OMDB resources. Table 1 provides some general information about the datasets.

Dataset	Origin	Format	Retrieved resources
TheMovieDatabase	https://www.themoviedb.org/	JSON	20,000
OpenMovieDatabase	http://www.omdbapi.com/	JSON	1,000

Table 1. Non-RDF Data Description

Of course, matching on title was bound to be imperfect, because of different spelling conventions, and different annotations to the name for disambiguation. Overall, we manage to match 976 movies, and we will use this as a starting point. Tables 2 and 3 shows a description of the data retrieved from those services.

Property	Type	Description
title	string	The movie title
popularity	float	Success measure based on viewer implicit and explicit feedback
avg_vote	float	Quality measure based on viewer explicit feedback
vote_count	integer	The number of viewers who have voted on this movie.

Table 2. The Movie Database Data Description

Property	Type	Description
actor	collection	A collection of actors starring in the movie
director	collection	A collection of movie directors
genre	string	The genre of the movie
language	string	The language in which the movie is played
plot	string	A short description of the movie
poster	link	A link to a image of the movie
rated	string	The parental rating
rating	float	The IMDB rating
runtime	integer	The duration of the movie
title	string	The movie title
writer	collection	A collection of movie writers
year	string	The release date of the movie

Table 3. Open Movie Database Data Description

RDF Data is acquired from the Linked Movie Database and from DBPedia, the Wikipedia RDF data warehouse service. Some general information is provided in Table 4.

Dataset	Origin	Format	Retrieved Triples
LMDB	https://old.datahub.io/dataset/linkedmdb	NTUPLES	3,579,614
DBPedia	http://dbpedia.org/page	TURTLE	3,034,317

Table 4. RDF Data Description

The Linked Movie Database also provides IMDB data but in linked open data format and is also much more expansive than its OMDB counterpart. This dataset is acquired from Datahub and consists of a dump file in NTUPLES format. We present the graph for a particular film resource in Figure 2 for reference.

We would like to import this entire dump into our data warehouse, but this is not readily possible because the dataset presents broken, or malformed URIs. Most notably, special characters such as parentheses, quotation marks or apostrophes are expressed in quoted format. For example, a left parenthesis is represented by the sequence of characters '%28'. Stardog does not allow such malformed URIs by default, so we explicitly tell it to parse the dump non-strictly. This allows for importing the entire set, but will provide challenges for data integration down the road.

We cannot import all of DBPedia. This would be impractical but also useless, since most of the data in there is not relevant to movies. However, we can use the LMDB 'sameAs' property that points to a DBPedia resource to pick and choose what we will import. This is done for movies and actors. Because of the malformed URIs, this will present a challenge.

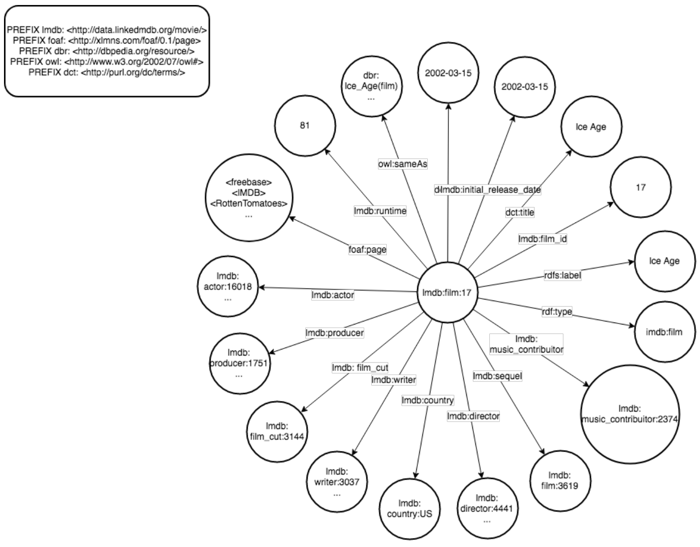


Fig. 2. A Film Resource from the LMDB Dataset. *A representation of the lmdb:film resource. Resource objects are represented by a prefix URI. Literals are represented as plain text. A '...' placeholder is set in case the resource has the same relation with multiple other resources. Each resource node potential has more links to other resource that are not represented here.*

Finally, we would have liked to acquire data on user reviews of movies and actors, but this seemed hard to come by. This would complement the data nicely with subjective opinions but was not realizable given the limitations of this project.

4 Methodology: Building Knowledge Graphs

To build the knowledge graph, we first start by converting the JSON data obtained from TMDB and OMDB into RDF. We take a simplistic approach to this conversion. Each dataset gets its own name space which is just the base URI of the origin and all predicates are names according to the key in the JSON data. Each OMDB movie resource is identified by the IMDB ID and each TMDB resource is identified by the TMDB ID, which is not transferable. All properties are recorded with a prefix according to their provenance and values of those properties are recorded as literals since at this stage we do not have a means to corroborate, say, an actor name with his or her resource specification. Taking this approach, we obtain a graph as shown in Figure 3. It should be observed that at this point that the graphs live in separate silos, since there is no link established between them.

As mentioned previously, for the LMDB data, we simply import the dump as is, using non-strict parsing, to preserve all triples. After the import, a critical

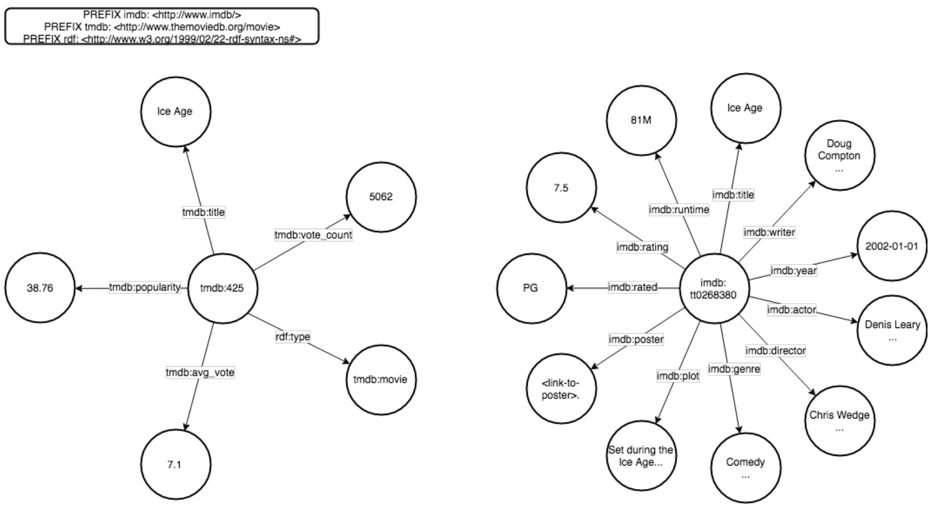


Fig. 3. TMDB and OMDB Data in RDF Format. *The same conventions apply for this figure as before. The resources live in separate silos for now, since there is no link between properties of different objects.*

step is data cleaning. Each film and actor resource from LMDB has 'sameAs' references to the DBPedia and YAGO knowledge bases. We remove the YAGO reference since we do not intend to use it. Also, we observe that for DBPedia, the URI references are poorly encoded and badly formed. The URIs contain apostrophes, and quotation marks, which are typically not allowed. We also note that all URI are quoted and some special characters are poorly encoded. For example, a URI in our LMDB dataset might look like 'http://dbpedia.org/resource/LÃ©on_%28film%29' should actually be represented as 'http://dbpedia.org/resource/Lèon_(film)'. We go ahead and unquote the URIs to take care of encodings such as '%28', and we remove any characters that are not allwoed from the URI, but we cannot find a way to fix the poorly encoded characters. This will produce some issues down the road.

Linking DBPedia data to our existing Knowledge Graph could be achieved in 2 ways. The first is to rely on SPARQL federated queries. That is, we can load the DBPedia endpoint along our own and use the 'sameAs' pointers in our LMDB dataset to retrieve DBPedia data on the fly. The second is to pre-import all existing DBPedia data in our own Stardog repository and allow the reasoner to make use of the 'sameAs' relations. The first has one major advantage over the other, namely that method ensures that any DBPedia data retrieved will be up to date, since we have a live connection to the DBPedia servers. However, such queries can become expensive quickly.

Imagine trying to get data on 50 movies from DBPedia starting from the LMDB data. A query would need to be submitted for each one of the 50 movies from LMDB, which is hardly a good solution. Second of all, federated search also includes another layer of complexity which is difficult to handle for an inexperienced technical user that is trying to explore the capabilities of our dataset.

For these reasons, we opt for the more efficient, less prohibiting, but also more risk prone solution of pre-importing DBPedia data. We first take all the movies and actors from our imported LMDB dataset and we extract the (cleaned) URI that points to the DBPedia resource from the 'sameAs' property. These URI strings are the unqoted and converted to an RDF URI.

At this point, we are in principle ready to match all DBPedia resources that have the same URI. However, as note earlier some URIs are poorly encoded, which means that some resource will not have a match in DBPedia. This is a limitation that we could not overcome. To actually import the data, we run a CONSTRUCT query over the DBPedia endpoint, saving the results in a TURTLE file for later import into our Stardog repository.

Finally we note that in order to not flood the DBPedia SPARQL endpoint with requests, we chunk our URIs into sets of 50 and ask DBPedia to find all triples that contain any of the 50 URIS as subjects. The entire process of creating the Knowledge Graph is depicted in Figure 4.

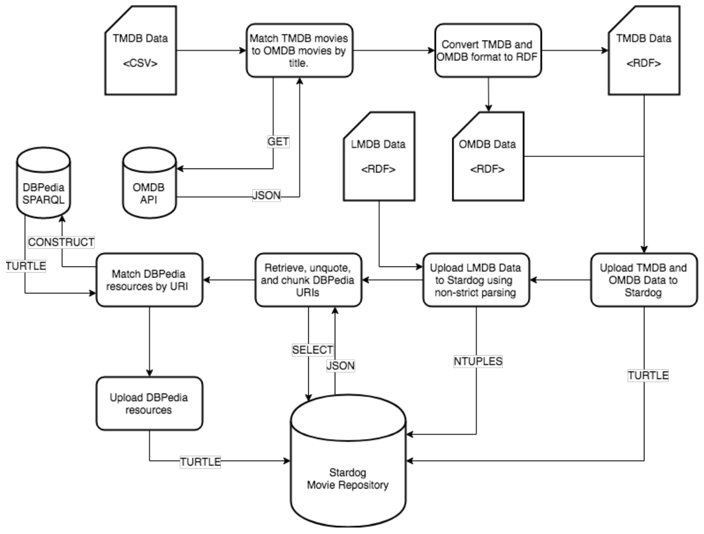

TMDB Data
<CSV>
Match TMDB movies to OMDB movies by title.
Convert TMDB and OMDB format to RDF
TMDB Data
<RDF>
GET
LMDB Data
<RDF>
OMDB Data
<RDF>
DBPedia SPARQL
OMDB API
JSON
CONSTRUCT
TURTLE
Match DBPedia resources by URI
Retrieve, unquote, and chunk DBPedia URIs
Upload LMDB Data to Stardog using non-strict parsing
Upload TMDB and OMDB Data to Stardog
SELECT
JSON
NTUPLES
TURTLE
Upload DBPedia resources
TURTLE
Stardog Movie Repository

Fig. 4. Building the Stardog Repository. *The process is done sequentially. Arrow labels indicate the type of request / response sent between separate entities. Actions are represented by rectangles, remote services by cylinders and local files by cards.*

At this point we have 3 datasets brought under one roof, but they all live in separate silos. The next step is to generate a vocabulary that allows us to link these silos according to some properties.

5 Results

5.1 The Dataset

The dataset is now formed. Some rules are also added in our ontology that they will be explained in the next section. It is also important that the final constructed dataset meets the 5 star model for Linked Data. This means the below:

⋆ **Data is available on the Web.** All our datasets were obtained via the web. (e.g. linked-data from DBPedia, JSON format from OMDB). While we do not publish a SPARQL endpoint for the purpose of this project, this can be easily achievable using Stardog.

⋆⋆ **Available as machine-readable structured data.** This criterion is also met, as our data were downloaded through API requests in json format, or directly in RDF format.

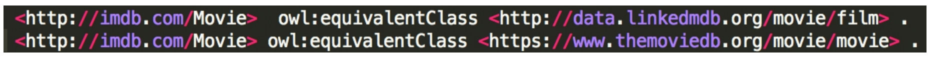
⋆⋆⋆ **Available in a non-proprietary format, (i.e, CSV).** This is already achieved using our RDF representation.

⋆⋆⋆⋆ **Published using open standards (RDF and SPARQL).** As well described at the previous sections our final dataset is in RDF.

⋆⋆⋆⋆⋆ **All of the above and links to other Linked Open Data.** he final criterion is also met as our data also provide links to other sources. An example is the *linkedMDB:actor* which also provides links to the DBPedia about this actor.

5.2 The Ontology

As mentioned above, when the RDF of the combined data was created there were not sufficient keys in order to match directly the different resources. (e.g *imdb:id* != *tmdb:id*). For this reason an ontology was designed to wrap these with some rules. The first rules applied are illustrated at the Figure 5 .

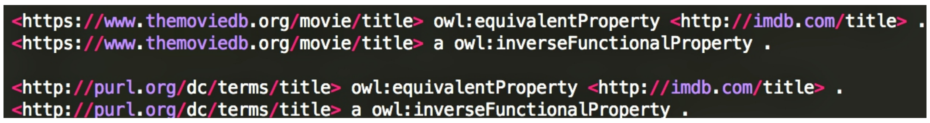


```
<http://imdb.com/Movie>  owl:equivalentClass <http://data.linkedmdb.org/movie/film> .
<http://imdb.com/Movie> owl:equivalentClass <https://www.themoviedb.org/movie/movie> .
```

Fig. 5. equivalentClass

The *owl:equivalentClass* rule was used to match the different movie entities. That was helpful in order to retrieve information from all the different datasets when asking in a SPARQL query for an *imdb:Movie.*

The next necessary matching was the movie title, so it is able to obtain information from all the datasets asking only for an *imdb:title.* The Figure 6 shows the rules applied for this purpose.

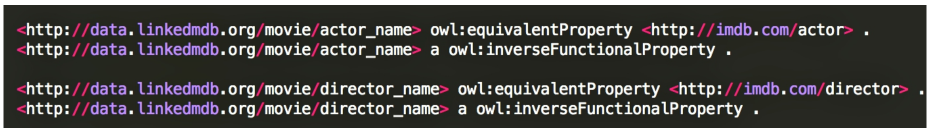


```
<https://www.themoviedb.org/movie/title> owl:equivalentProperty <http://imdb.com/title> .
<https://www.themoviedb.org/movie/title> a owl:inverseFunctionalProperty .

<http://purl.org/dc/terms/title> owl:equivalentProperty <http://imdb.com/title> .
<http://purl.org/dc/terms/title> a owl:inverseFunctionalProperty .
```

Fig. 6. inverseFunctionalProperty

The pair of *owl:equivalentProperty* and *owl:inverseFunctionalProperty* provide us with this ability. We are able now to access info about a movie from the datasets by only providing the *imdb:title*.

The last case in which a matching deemed appropriate is the linking between actor/director/writer.. name among all the datasets. More specifically, the *imdb:actor* provide us information about the actor names only. While if we load the same movie via the *linkedmdb* only the actorss ids are visible. So in order to load more information about the actor himself from the *linkedmdb*, it was essential to match the *imdb:actor* with the *linkedmdb:actor_name*. As a result of this matching, providing the *imdb:actor* we find the *linkedmdb:actor* that has an *linkedmdb:actor_name* = *imdb:actor* and all the data for this actor is loaded. The figure 7 show some examples of these rules added, carrying similar properties as the previous rules.



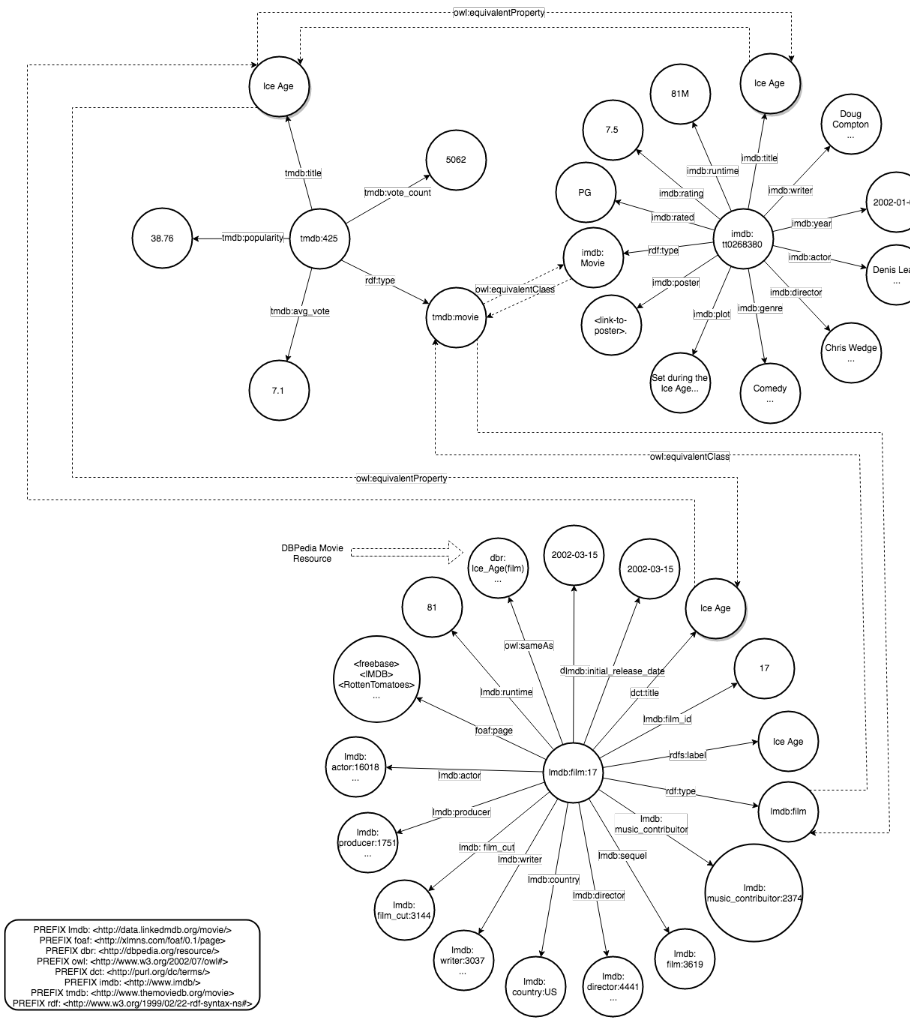
```
<http://data.linkedmdb.org/movie/actor_name> owl:equivalentProperty <http://imdb.com/actor> .
<http://data.linkedmdb.org/movie/actor_name> a owl:inverseFunctionalProperty .

<http://data.linkedmdb.org/movie/director_name> owl:equivalentProperty <http://imdb.com/director> .
<http://data.linkedmdb.org/movie/director_name> a owl:inverseFunctionalProperty .
```

Fig. 7. Rules to match actors, directors etc.

6 Evaluation

Overall, we observe satisfactory results. Over the course of this endeavour we have managed to link 4 datasets with each other to provide for a rich body of knowledge on movies. While all these 4 datasets resided as silos on the web, they are not integrated in a way that allows cross dataset querying easily. Examples of such queries are presented in our Data Stories mock-up, which is available online. As described above, what allowed for the datasets to function as one unified set was the vocabulary that we defined to link properties and classes and to express properties that are unique identifiers for some classes. A representation of a linked resource is provided in Figure 8

Nonetheless, we do observe some less than satisfactory results. For example, although the tmdb:movie and imdb:movie are equivalent classes, when querying for properties from both sets, we still need to explicitly add a UNION clause to our SPARQL queries in order to get some results and the triples returned



```
PREFIX lmdb: <http://data.linkedmdb.org/movie/>
PREFIX foaf: <http://xlmns.com/foaf/0.1/page>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX imdb: <http://www.imdb/>
PREFIX tmdb: <http://www.themoviedb.org/movie>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Fig. 8. Full Linked Model Representation.

are separated according to the class to which the property belongs. We find this counter-intuitive since we defined the classes to be equivalent and we could not find a solution to this problem.

7 Discussion

To conclude, our work in this project was focused on bringing multiple dataset under one roof and linking them. Along the way, we have come to appreciate the Web of Data technologies according to their potential. We are content with our results, although we wish we had more time to further develop our project further. Among the top priorities for the future are devising an ontology that allows for easy recovery of properties from different equivalent classes, and adding more non-RDF data to the repository, since at this point we only have about 1000 resources of this origin. It is clear to us that this technology holds potential, but we also recognize that data integration still present great challenges, especially if the available data is not proper.