

Movie Knowledge Question & Answer

Yunzhe GUO

GItHub:<https://github.com/GuoYunZheSE/MovieKnowledge>

Website Accessible: <http://121.40.189.99:4753/#/> (IMPORTANT: Please read the demo description section of the report before trying our demos)

2022/02/06

Data Source and Data Process

Data Craw

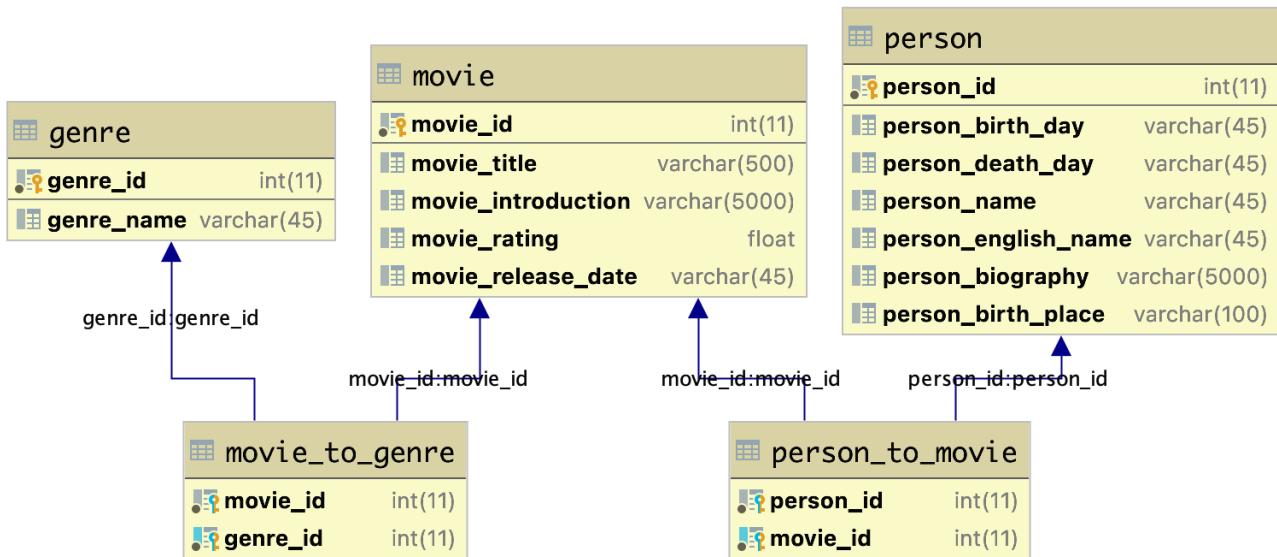
The data is obtained from [The Movie Database \(TMDB\)](#) website, which officially provides API KEY for registered users to query and download the data.

Basic information of actors include: name, English name, date of birth, date of death, place of birth, and personal profile.

Basic information of the movie includes: movie title, movie synopsis, movie rating, movie release date, and movie genre.

In this example, we take **Leonardo DiCaprio** as the initial entry point and get all the movies he has appeared in; then we get all the actors and actresses in these movies; finally we get all the movies in which all the actors and actresses have appeared. After de-duplication, we get **1976** actors' basic information and **29174** movies' basic information. The data is saved in **MySQL**.

The ER graph is :



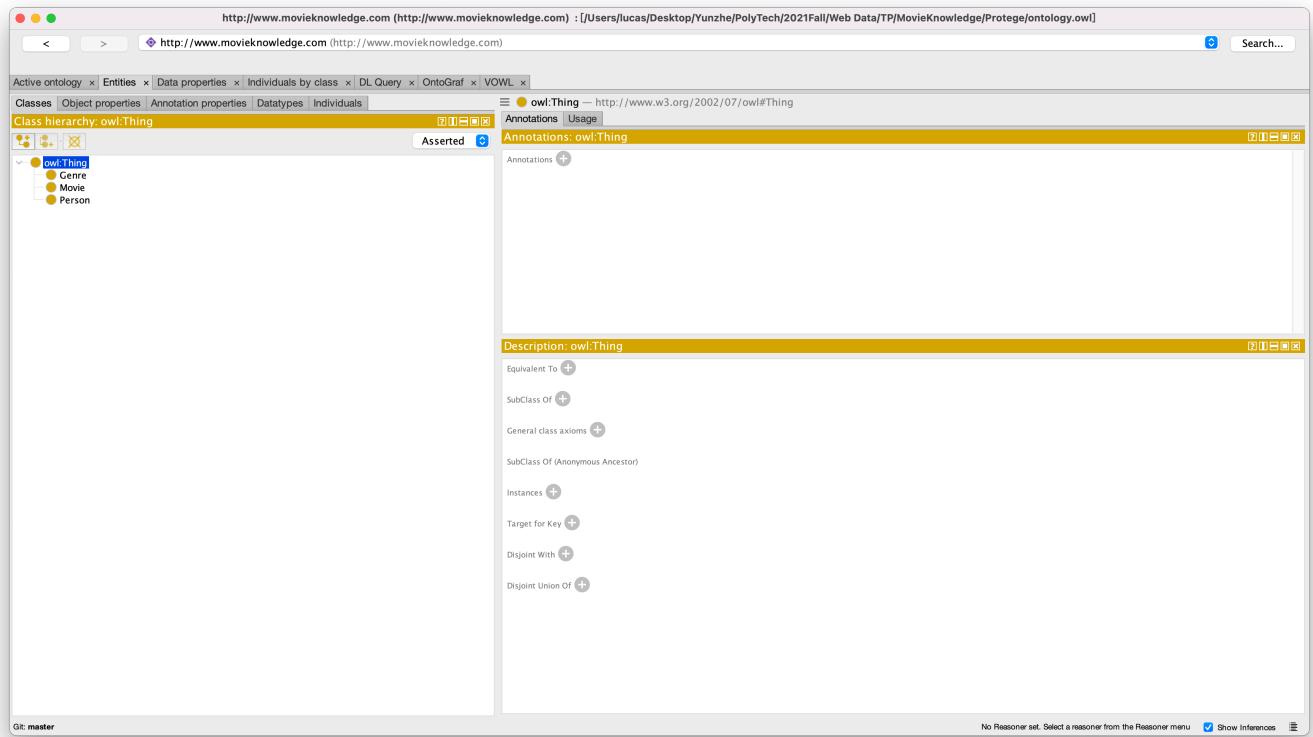
The basic statistics are as follows.

1. Number of actors: 1976

2. Number of movies: 29174
3. Film genres: 19 categories
4. Relationship between characters and movies: 48101
5. Relationship between movies and genres: 55968

Ontology modeling

Entities



Data Properties

http://www.movieknowledge.com (http://www.movieknowledge.com) : [/Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web Data/TP/MovieKnowledge/Protege/ontology.owl]

Active ontology Entities Data properties Individuals by class DL Query OntoGraf VOWL

Data property hierarchy: movieIntroduction

Annotations: movieIntroduction

Characteristics: movieIntroduction Description: movieIntroduction

Functional

Equivalent To

SubProperty Of

owl:topDataProperty

Domains (Intersection)

Movie

Ranges

xsd:string

Disjoint With

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Object Properties

http://www.movieknowledge.com (http://www.movieknowledge.com) : [/Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web Data/TP/MovieKnowledge/Protege/ontology.owl]

Active ontology Entities Data properties Individuals by class DL Query OntoGraf VOWL

Classes Object properties Annotation properties Datatypes Individuals

Object property hierarchy: hasActor

Annotations: hasActor

Characteristics: hasActor Description: hasActor

Functional

Inverse functional

Transitive

Symmetric

Asymmetric

Reflexive

Irrreflexive

Equivalent To

SubProperty Of

Inverse Of

hasActedin

Domains (Intersection)

Movie

Ranges (Intersection)

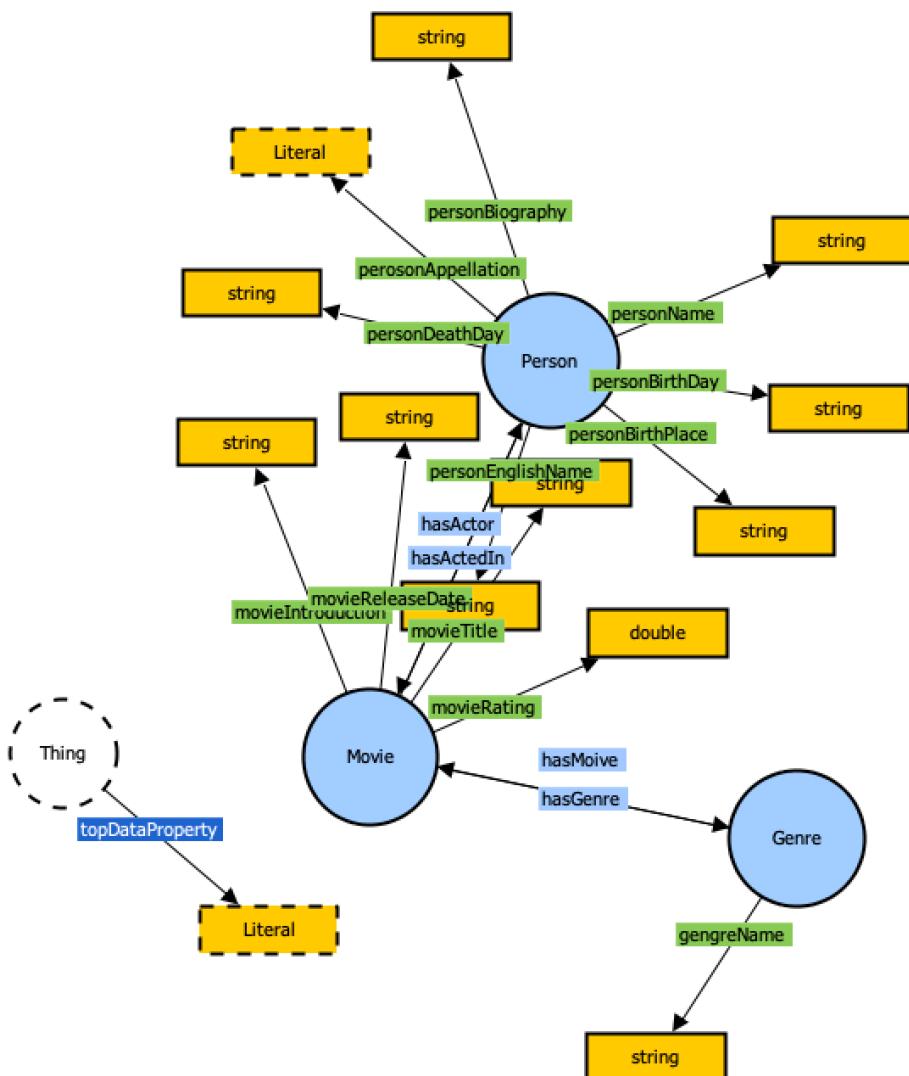
Person

Disjoint With

SuperProperty Of (Chain)

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Over All



Map SQL to RDF by d2rq

D2RQ provides its own mapping language in a form similar to R2RML. d2rq has the convenience of automatically generating a predefined mapping file based on the database, and users can modify this file to map the data to their own ontology.

Generate Mapping File

```
generate-mapping -u root -o movie_mapping.ttl jdbc:mysql://MovieKnowledge
```

After some modification, we can get our mapping file `movie_mapping.ttl` like this:

```
@prefix map: <#> .  
@prefix db: <> .  
@prefix vocab: <vocab/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

```

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .
@prefix : <http://www.movieknowledge.com#> .

map:database a d2rq:Database;
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
  d2rq:jdbcDSN "jdbc:mysql:///MovieKnowledge";
  d2rq:username "root";
  d2rq:password "yourpassword";
  jdbc:zeroDateTimeBehavior "convertToNull";
  jdbc:autoReconnect "true";
.

# Table genre
map:genre a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "genre/@@genre.genre_id@@";
  d2rq:class :Genre;
  d2rq:classDefinitionLabel "genre";
.

map:genre_genre_name a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:genre;
  d2rq:property :genreName;
  d2rq:propertyDefinitionLabel "genre genre_name";
  d2rq:column "genre.genre_name";
.
```

Generate RDF File Automatically

```
./dump-rdf -o movie_mapping.nt ./movie_mapping.ttl
```

The Turtle RDF file contains something like this:

```

<file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#person/2955369> <http://www.movieknowledge.com#PersonBirthDay> "1949-01-12" .

<file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#person/2955369> <http://www.movieknowledge.com#personBiography> "David Luban is a University Professor and Professor of Law and Philosophy. Since 2013, he has also served as Class of 1984 Distinguished Chair in Ethics at the U.S. Naval Academy's Stockdale Center for Ethical Leadership." .

<file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#person/2955369> <http://www.movieknowledge.com#personBirthPlace> "Milwaukee, Wisconsin, USA" .

<file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#person/2955369> <http://www.movieknowledge.com#personEnglishName> "David Luban" .

<file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#person/2955369> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.movieknowledge.com#person> .

```

We can see the content more clearly by [OpenRefine](#):

Movie Release Date	Title	Introduction	Genre
Paper Bullets	Alcoholic cop John Rourke finds a trail of corruption after a gunman opens fire on a police conference.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/60	
Betrayal	When one of her hits goes wrong, a professional assassin ends up with a suitcase full of a million dollars belonging to a mob boss ...	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/28	
Percentage	After a drug deal gone wrong, two New York City hustlers relocate to Miami, where they set up a lucrative credit card fraud operation. Unfortunately, their new business not only attracts the police, but the city's top criminals -- and both want a percentage of the take.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/60	
Kisses in the Dark	Four independent short films comprise this quirky anthology. "Coriolis Effect" (1994) is an offbeat love story involving storm chasers. In the Oscar-nominated "Sally's Diner" (1979), a homeless man (Larry Hankin, who also directs) witnesses a woman (Diane Ladd) being beaten and然後殺死。And the dialogue-free "J'been" (1997) features David Aaron Baker as a psychiatric patient searching for enlightenment.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/35	
Mystery 101: Dead Talk	Amy heads to Seattle to give a TED Talk-style lecture. When a tech genius dies, Amy suspects foul play. When local cops shut her out, Travis joins forces with Amy to figure out whodunit.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/18	
Behind the Moons	Interviews with the English language voice cast of Moons and the Comet Chase."	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/648	
Shadow of Doubt	An attorney uncovers a political conspiracy in the brutal killing of a wealthy young woman.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/69	
Whiplash	Under the direction of a ruthless instructor, a talented young drummer begins to pursue perfection at any cost, even his humanity.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/53	
Breakin' All the Rules	Inspired by his fiancée (who dumped him), a man publishes a break-up handbook for men, becoming a bestselling author in the process.	file:///Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web%20Data/TP/MovieKnowledge/movie_mapping.nt#genre/10402	

Apache Jena SPARQL Endpoint

The section we will use in this subsection are: **TDB, Jena and Fuseki**.

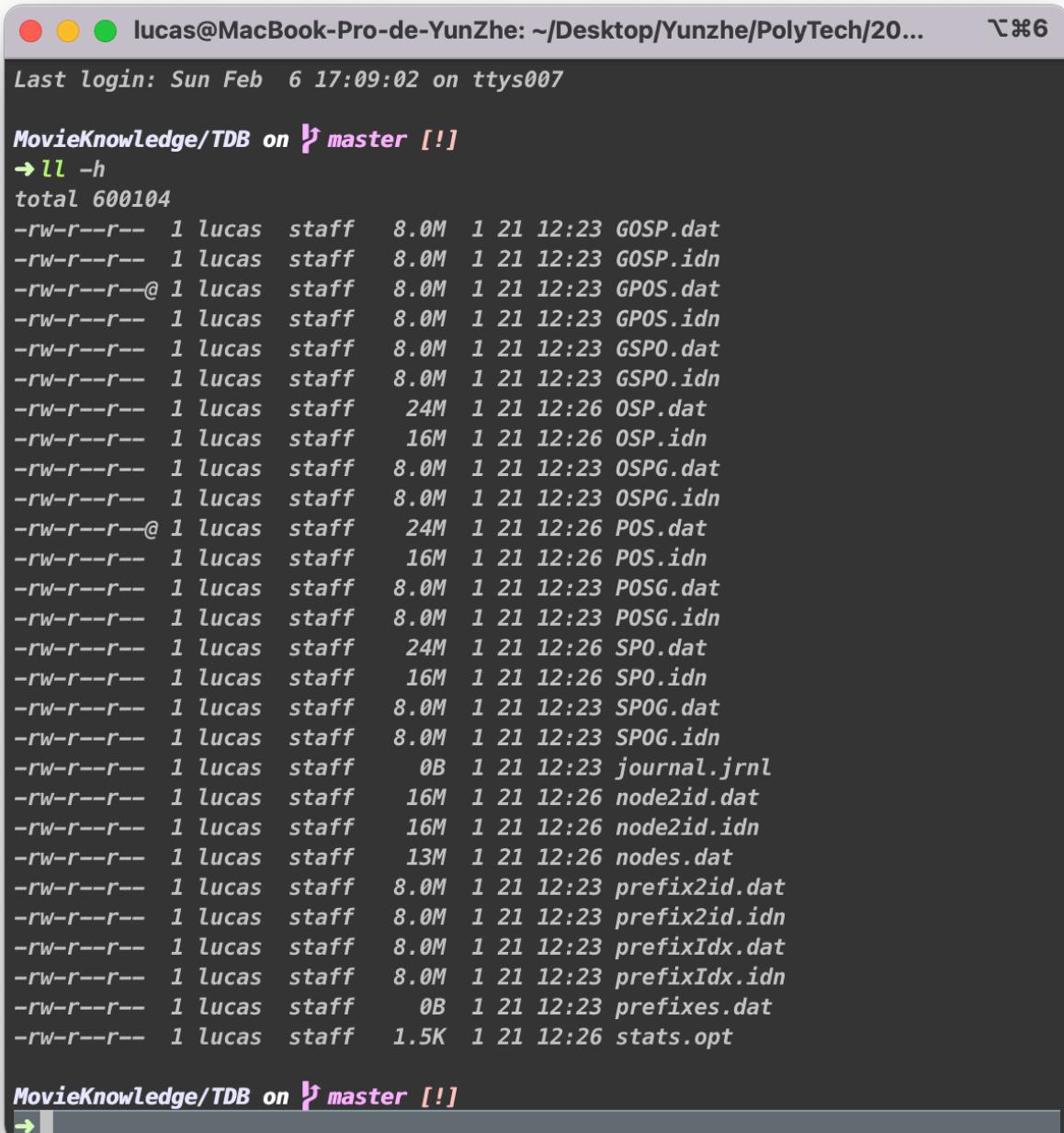
1. TDB is a component used by Jena to store RDF, and is a technology that belongs to the storage level. It can provide very high RDF storage performance in standalone case. The latest version of TDB is currently TDB2 and is not compatible with TDB1. 2.
2. Jena provides RDFS, OWL and generic rule reasoner. In fact, Jena's RDFS and OWL reasoner are also

implemented by Jena's own general-purpose rule reasoner.

3. Fuseki is a SPARQL server, or SPARQL endpoint, provided by Jena, which provides four modes of operation: standalone, as a service of the system, as a web application, or as an embedded server. Movie title, movie synopsis, movie rating, movie release date, and movie genre.

Generate TDB Storage By Jena

```
./tdbloader --loc=". /MovieKnowledge/TDB" ". /MovieKnowledge/movie_mapping.nt"
```



The screenshot shows a terminal window on a Mac OS X desktop. The title bar reads "lucas@MacBook-Pro-de-YunZhe: ~/Desktop/Yunzhe/PolyTech/20... ⌘⌘6". The terminal output is as follows:

```
Last login: Sun Feb 6 17:09:02 on ttys007

MovieKnowledge/TDB on ↵ master [!]
→ ll -h
total 600104
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 GOSP.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 GOSP.idn
-rw-r--r--@ 1 lucas staff 8.0M 1 21 12:23 GPOS.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 GPOS.idn
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 GSPO.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 GSPO.idn
-rw-r--r-- 1 lucas staff 24M 1 21 12:26 OSP.dat
-rw-r--r-- 1 lucas staff 16M 1 21 12:26 OSP.idn
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 OSPG.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 OSPG.idn
-rw-r--r--@ 1 lucas staff 24M 1 21 12:26 POS.dat
-rw-r--r-- 1 lucas staff 16M 1 21 12:26 POS.idn
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 POSG.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 POSG.idn
-rw-r--r-- 1 lucas staff 24M 1 21 12:26 SPO.dat
-rw-r--r-- 1 lucas staff 16M 1 21 12:26 SPO.idn
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 SPOG.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 SPOG.idn
-rw-r--r-- 1 lucas staff 0B 1 21 12:23 journal.jrn1
-rw-r--r-- 1 lucas staff 16M 1 21 12:26 node2id.dat
-rw-r--r-- 1 lucas staff 16M 1 21 12:26 node2id.idn
-rw-r--r-- 1 lucas staff 13M 1 21 12:26 nodes.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 prefix2id.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 prefix2id.idn
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 prefixIdx.dat
-rw-r--r-- 1 lucas staff 8.0M 1 21 12:23 prefixIdx.idn
-rw-r--r-- 1 lucas staff 0B 1 21 12:23 prefixes.dat
-rw-r--r-- 1 lucas staff 1.5K 1 21 12:26 stats.opt

MovieKnowledge/TDB on ↵ master [!]
```

Start Fuseki Service

1. Go to the "apache-jena-fuseki" folder, run `./fuseki-server`, and exit. The program will automatically create the "run" folder in the current directory for us.
2. Move our ontology file "ontology.owl" to the "databases" folder under the "run" folder and change the "owl" extension to "ttl".
3. In the "configuration" folder under the "run" folder, create a text file named "fuseki_conf.ttl" , and add the configuration.

```
@prefix :      <http://base/#> .
@prefix tdb:   <http://jena.hpl.hp.com/2008/tdb#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:    <http://jena.hpl.hp.com/2005/11/Assembler#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix fuseki: <http://jena.apache.org/fuseki#> .

:service1      a          fuseki:Service ;
                  fuseki:dataset <#dataset> ;
                  fuseki:name      "MovieKnowledge" ;
                  fuseki:serviceQuery "query" , "sparql" ;
                  fuseki:serviceReadGraphStore "get" ;
                  fuseki:serviceReadWriteGraphStore "data" ;
                  fuseki:serviceUpdate      "update" ;
                  fuseki:serviceUpload      "upload" .

<#dataset> rdf:type ja:RDFDataset ;
  ja:defaultGraph <#model_inf> ;
  .

<#model_inf> a ja:InfModel ;
  ja:MemoryModel <#tdbGraph> ;

  #本体文件的路径
  ja:content [ ja:externalContent <file:///Applications/apache-jena-fuseki-
4.3.2/run/databases/ontology.ttl> ] ;

<#tdbGraph> rdf:type tdb:GraphTDB ;
  tdb:dataset <#tdbDataset> ;
  .

<#tdbDataset> rdf:type tdb:DatasetTDB ;
  tdb:location "/Users/lucas/Desktop/Yunzhe/PolyTech/2021Fall/Web
Data/TP/MovieKnowledge/TDB" ;
  ja:context[ ja:cxName "arqTimeout"; ja:cxtValue "1000"]
  .
```

The service is hosted on port `3030`

Website Demo

I used a separate front-end and back-end development approach, using vue.js for the front-end, mainly responsible for sending requests and displaying results.

The backend uses `django` and uses the `SPARQLWrapper` package to further wrap our SPARQL query, so I will focus on our backend implementation in this section.

FrontEnd

I used `element-ui` in order to develop the interface more conveniently.

I also used `axios` for request.

```
import axios from 'axios';
import { API_HOST } from '../config.json';

export function sendMessage(sender, message) {
  return axios.post(` ${API_HOST}/Answer/`, { sender, message });
}

export function QuryActor(queryParams) {
  return axios.get(` ${API_HOST}/QueryActor/`, { params: queryParams });
}

export function QueryMovie(queryParams) {
  return axios.get(` ${API_HOST}/QueryMovie/`, { params: queryParams });
}
```

BackEnd

We have implemented two functions based on the knowledge graph, one is to display the data in the form of a table where users can find and filter, we call it **Custom Search**:

The screenshot shows a web-based movie search application. At the top, there is a navigation bar with links to Home, Actor, Movie, and other categories. Below this is a search form titled "Movie Query" containing fields for "Movie Title" (with a dropdown for "Science Fiction"), "Rating Descen[...]" (with a dropdown for "Rating Descen[...]"), and a "Search" button. The main content area displays a table of movie results:

ID	Title	Release Date	Rating	Introduction
1	Star Trek: Inside the Roddenberry Vault	2016-12-05	10.	Fifty years after the original Star Trek first arrived on...
2	Gerry Anderson's Firestorm		10.	Filmed in 'Ultramarionation' Gerry Anderson's FIRE...
3	Shadow on the Land	1968-12-04	10.	Patriotic freedom fighters struggle against a fascist ...
4	Transformers: The Rebirth	1987-11-09	10.	Since his return as Autobot leader, Optimus Prime h...

At the bottom, there is a pagination control showing "共 1656 条" (Total 1656 items), "20条/页" (20 items per page), and a series of numbered buttons from 1 to 83, with "1" being highlighted in blue. There are also buttons for "前往" (Go to) and "1 页" (Page 1).

The other is **Question and Answer**:

Movie QA

[What can I do?](#)

Hello, I'm your Movie assistant~

Titanic

124

Which movies did Leonardo DiCaprio and Andie Hicks star in together?

How many movies has Harrison Ford starred in?

The first letter of the actor and the movie need to be capitalized, e.g. Titanic, Leonardo DiCaprio

[Send](#)

Implementation of Custom Search

This implementation is simple, we pre-define some template and fill them according to the request send to server:

```
SPARQL_PREFIX = u"""
PREFIX : <http://www.movieknowledge.com#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
"""

SPARQL_SELECT_TEMPLATE = """
{prefix}
SELECT DISTINCT {select} WHERE {{ 
    {expression}
    {filter}
}}
{pagination}
"""

SPARQL_SELECT_TEMPLATE = SPARQL_SELECT_TEMPLATE.replace("{prefix}", SPARQL_PREFIX)
SPARQL_SELECT_TEMPLATE = SPARQL_SELECT_TEMPLATE.replace("{select}", "SELECT ?label")
SPARQL_SELECT_TEMPLATE = SPARQL_SELECT_TEMPLATE.replace("{expression}", "WHERE ?label a Person")
SPARQL_SELECT_TEMPLATE = SPARQL_SELECT_TEMPLATE.replace("{filter}", "FILTER (str(?label) != '')")
SPARQL_SELECT_TEMPLATE = SPARQL_SELECT_TEMPLATE.replace("{pagination}", "LIMIT 20 OFFSET 0")
```

The search follows this pattern: <http://127.0.0.1:8000/api/QueryActor/?page=1&limit=20&sort=-PersonBirthDay>

Implementation of Question and Answer

This feature uses regular expressions to do semantic parsing. We need a third-party library to do the initial natural language processing (POS entity recognition), and then use a library that supports word-level regular matching to do the subsequent semantic matching.

We use spacy for POS and entity recognition (names of people and movies). In our demo, the accuracy of entity recognition is not acceptable for human names and movie titles. Therefore, we directly export the names of people and movies in the database as external dictionaries; load the external dictionaries when using spacy, so that the problem of entity recognition can be solved:

```
def train_ER(self, file_path: str, entity_type: str, entity_pattern):
    ruler = EntityRuler(self.nlp, overwrite_ents=True)
    jsonData = None
    with open(file_path) as f:
        jsonData = json.load(f)
    patterns = [{"label": entity_type, "pattern": each[entity_pattern]} for each in
jsonData]
    ruler.add_patterns(patterns)
    ruler.name = f"{entity_type}_ruler"
    self.nlp.add_pipe(ruler)
```

After converting natural language to word-based basic units, we use **REFo** (Regular Expressions for Objects) to complete semantic matching.

```
rules = [
    Rule(condition_num=2, condition=person_entity + Star(Any(), greedy=False) + movie +
Star(Any(), greedy=False), action=QuestionSet.has_movie_question),
    Rule(condition_num=2, condition=(movie_entity + Star(Any(), greedy=False) + actor +
Star(Any(), greedy=False)) | (actor + Star(Any(), greedy=False) + movie_entity +
Star(Any(), greedy=False)), action=QuestionSet.has_actor_question),
    Rule(condition_num=3, condition=person_entity + Star(Any(), greedy=False) +
person_entity + Star(Any(), greedy=False) + (movie | Star(Any(), greedy=False)),
action=QuestionSet.has_cooperation_question),
]
```

After a successful match, we get its corresponding SPARQL template we pre-wrote, then send a query to **Fuseki** server, and finally print out the result.

Demo Explanation

IMPORTANT: The demo is host in a alibaba cloud server, which is in China and has only 4 GB memory. Therefore, the demo may be slow.

Because the full movie data is too big and the server memory is small, please DO NOT use the Movie Custom Search part.

You can find the demo: <http://121.40.189.99:4753/#/>

Custom Search

1. Request URL: <http://127.0.0.1:8000/api/QueryActor/?page=1&limit=20&sort=%2BPersonBirthDay>

2. BackEnd Log:

```
PREFIX : <http://www.movieknowledge.com#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT * WHERE {

  ?x :personEnglishName ?personEnglishName .

  optional
  {
    ?x :PersonBirthDay ?PersonBirthDay .
    ?x :personDeathDay ?personDeathDay .
    ?x :personBiography ?personBiography .
    ?x :personBirthPlace ?personBirthPlace .
  }

  ORDER BY ?PersonBirthDay LIMIT 20 OFFSET 0
}
```

[06/Feb/2022 19:23:21] "GET /api/QueryActor/?page=1&limit=20&sort=%2BPersonBirthDay HTTP/1.1" 200 2901

3. Fuseki Log:

```
20:23:21 INFO Fuseki      :: [591] 200 OK (179 ms)
20:23:21 INFO Fuseki      :: [591] GET http://127.0.0.1:3030/MovieKnowledge/query?query=%0A+++++PREFIX%3Ahttp%3A//www.movieknowledge.com%23%3E%0A++PREFIX+rdf%3A%3Chttp%3A//www.w3.org/1999/02/22-rdf-syntax-ns%23%3E%0A++PREFIX+rdfs%3A%3Chttp%3A//www.w3.org/2000/01/rdf-schema%23%3E%0A%0A+++++SELECT+DISTINCT+?x+WHERE+?x%0A+++++optional%0A+++++?B%0A+++++?3Fx++%3APersonEnglishName%3FpersonEnglishName+.%0A%0A+optional%0A+++++?3Fx++%3APersonBirthDay%3FpersonBirthDay+.%0A+++++?3Fx++%3APersonDeathDay%3FpersonDeathDay+.%0A+++++?3Fx++%3APersonBiography%3FpersonBiography+.%0A+++++?3Fx++%3APersonBirthPlace%3FpersonBirthPlace+.%0A+++++?7D%0A+++++?D%0A+++++?0ORDER+BY%3FpersonBirthDay+LIMIT+20+OFFSET+0%0A&format=json&output=json&results=json
20:23:21 INFO Fuseki      :: [592] Query = PREFIX : <http://www.movieknowledge.com#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT DISTINCT * WHERE { ?x :personEnglishName ?personEnglishName . optional { ?x :PersonBirthDay ?PersonBirthDay. ?x :personDeathDay ?personDeathDay. ?x :personBiography ?personBiography. ?x :personBirthPlace ?personBirthPlace. } } ORDER BY ?PersonBirthDay LIMIT 20 OFFSET 0
20:23:21 INFO Fuseki      :: [592] 200 OK (133 ms)
```

4. Result:

Actor Query

Actor Name From Older to

ID	Actor Name	Actor Bi rthday	Actor D eathDay	Actor Bi rthPlace	Actor Biography
1	A. Lee Morris	None	None	None	None
2	A. O. Scott	None	None	None	None
3	AJ McLean	None	None	None	None
4	AJ Taylor	None	None	None	None
5	Aaron Glaser	None	None	None	None
6	Aaron James C ash	None	None	None	None

共 1976 条 < 2 3 4 5 6 ... 99 > 前往 页

Question and Answer

1. Question:

```
{"sender": "2022-02-06T19:25:53.837Z6BIMK8QTI8", "message": "What movies did Leonardo DiCaprio act in?"}
```

2. BackEnd:

1. POS:

```
Leonardo DiCaprio Person
What DET
movies NOUN
did AUX
act VERB
in ADP
? PUNCT
```

2. SPARQL Matched:

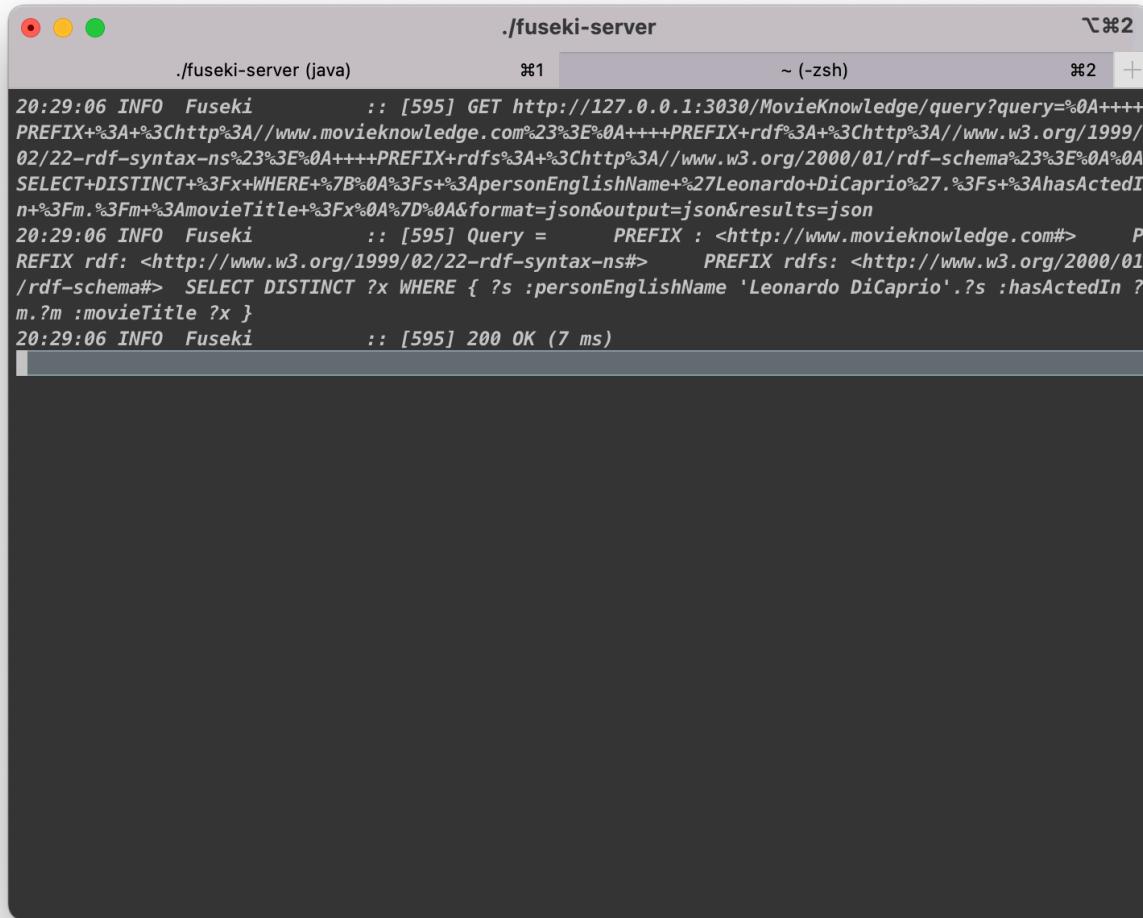
SPARQL:

```
PREFIX : <http://www.movieknowledge.com#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?x WHERE {
?s :personEnglishName 'Leonardo DiCaprio'.?s :hasActedIn ?m.?m :movieTitle ?x
}

[06/Feb/2022 19:26:01] "POST /api/Answer/ HTTP/1.1" 200 1373
```

3. Fuseki:



The screenshot shows a terminal window titled '.fuseki-server' running on a Mac OS X system. The window has three tabs: tab 1 contains the command '.fuseki-server (java)', tab 2 contains the log output, and tab 3 contains the command '~ (-zsh)'. The log output shows the following:

```
20:29:06 INFO  Fuseki      :: [595] GET http://127.0.0.1:3030/MovieKnowledge/query?query=%0A+++
PREFIX+>%3A+>%3Chttp%3A//www.movieknowledge.com%23%3E%0A+++
PREFIX+rdf%3A+>%3Chttp%3A//www.w3.org/1999/02/22-rdf-syntax-ns%23%3E%0A+++
PREFIX+rdfs%3A+>%3Chttp%3A//www.w3.org/2000/01/rdf-schema%23%3E%0A%0A
SELECT+DISTINCT+>%3Fx+WHERE+>%7B%0A%3Fs+>%3ApersonEnglishName+>%27Leonardo+DiCaprio%27.%3Fs+>%3AhasActedIn+>%3Fm.%3Fm+>%3AmovieTitle+>%3Fx%0A%7D%0A&format=json&output=json&results=json
20:29:06 INFO  Fuseki      :: [595] Query = PREFIX : <http://www.movieknowledge.com#> P
REFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT DISTINCT ?x WHERE { ?s :personEnglishName 'Leonardo DiCaprio'.?s :hasActedIn ?m.?m :movieTitle ?x }
20:29:06 INFO  Fuseki      :: [595] 200 OK (7 ms)
```

4. Result:

Movie QA

[What can I do?](#)

Hello, I'm your Movie assistant~

What movies did Leonardo DiCaprio act in?

Notes on an American Film Director at Work、The Wolf of Wall Street、Everything's Cool、Kid 90、Critters 3、Titanic、X-Girl、The Black Hand、Spielberg、What's Eating Gilbert Grape、Catch Me If You Can: Behind the Camera、Portrait of Leonardo: The Kid Who Took Hollywood、The Audition、Inception: The Cobol Job、Killers of the Flower Moon、Gangs of New York、Don't Look Up、The Basketball Diaries、The Great Gatsby、Before the Flood、The Quick and the Dead、The 11th Hour、Another Round、The Concert for New York City、JAPANARAMA! Psycho TV From Japan Vol. 1、The Aviator、Tesla's War、Blood Diamond、Salvator Mundi、Revolutionary Road、Marvin's Room、The Foot Shooting Party、Total Eclipse、This Boy's Life、Inception、Celebrity、The Lost Leonardo、Reflections on Titanic、Body of Lies、Mickey's Safety Club: Street Safe、Street Smart、Shutter Island、IMAX Hubble、Ice on Fire、A World Unseen: The Revenant、The Wolf

The first letter of the actor and the movie need to be capitalized, e.g. Titanic, Leonardo DiCaprio

Send