TUGAS AKHIR – IF184802

PERMODELAN SKEMA PENYIMPANAN DATA SILSILAH KELUARGA MENGGUNAKAN GRAF PADA NEO4J UNTUK MENCARI RELASI ANTAR TOKOH

NUZUL AYU SAFITRI
NRP 05111540000014

Dosen Pembimbing
Nurul Fajrin Ariyani, S.Kom., M.Sc.
Adhatus Solichah Ahmadiyah, S.Kom, M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

PERMODELAN SKEMA PENYIMPANAN DATA SILSILAH KELUARGA MENGGUNAKAN GRAF PADA NEO4J UNTUK MENCARI RELASI ANTAR TOKOH

NUZUL AYU SAFITRI
NRP 05111540000014

Dosen Pembimbing
Nurul Fajrin Ariyani, S.Kom., M.Sc.
Adhatus Solichah Ahmadiyah, S.Kom, M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – IF184802

FAMILY TREE SCHEME MODELING USING NEO4J GRAPH TO FIND RELATIONSHIP BETWEEN CHARACTERS

NUZUL AYU SAFITRI
NRP 05111540000014

Advisor
Nurul Fajrin Ariyani, S.Kom., M.Sc.
Adhatus Solichah Ahmadiyah, S.Kom, M.Sc.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical And Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PERMODELAN SKEMA PENYIMPANAN DATA SILSILAH KELUARGA MENGGUNAKAN GRAF PADA NEO4J UNTUK MENCARI RELASI ANTAR TOKOH

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Managemen Informasi
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
NUZUL AYU SAFITRI
NRP : 05111540000014

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Nurul Fajrin Ariyani, S.Kom., M.Sc. .....
NIP: 19860722 201504 2 003 (pembimbing 1)

Adhatus Solichah Ahmadiyah, S.Kom., M.Sc. .....
NIP: 19850826 201504 2 002 (pembimbing 2)

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN INSTITUT TEKNOLOGI SEPULUH NOPEMBER FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA
JANUARI 2020

[Halaman ini sengaja dikosongkan]

PERMODELAN SKEMA PENYIMPANAN DATA SILSILAH KELUARGA MENGGUNAKAN GRAF PADA NEO4J UNTUK MENCARI RELASI ANTAR TOKOH

Nama Mahasiswa	: NUZUL AYU SAFITRI
NRP	: 05111540000014
Departemen	: Informatika ITS
Dosen Pembimbing I	: Nurul Fajrin Ariyani, S.Kom., M.Sc.
Dosen Pembimbing II	: Adhatus Solichah Ahmadiyah, S.Kom, M.Sc.

Abstrak

Pengetahuan merupakan kumpulkan fakta-fakta atau informasi yang telah diinterpretasikan. Salah satunya terdapat pada informasi biografi atau data diri seseorang. Fakta-fakta atau data-data yang dikemukakan pada biografi tersebut dapat dioleh sehingga ditemukan pengetahuan baru yang mana muncul akibat adanya keterkaitan atau hubungan dengan tokoh lain. Sebagai contoh adanya hubungan kekeluargaan seperti hubungan antara anak dan orang tua, atau suami dan istri.

Hubungan-hubungan antar tokoh tersebut dapat digambarkan pada suatu ontologi yang mana merupakan suatu model untuk merepresentasukan hubungan antar konsep. Ontologi mendefinisikan class, property, instance, *dan relasi antar entitas dalam suatu domain tertentu sehingga memiliki makna (semantik). Ada beberapa terminologi yang dapat diterapkan dalam sebuah ontologi yaitu menggunakan terminologi RDF yang disebut* triplestore *dan basis data graf* (graph database).

Tujuan dari pengerjaan tugas akhir ini adalah memetakan skema penyimpanan data silsilah keluarga dalam bentuk graf menggunakan aplikasi bantuan Neo4j dan library neosematics untuk mengubah data yang berupa RDF menjadi graph database, *dan juga dilakukan reasoning menggunakan Apache Jena dengan* library *Pellet untuk menghasilkan* inferred fact *yang lebih lengkap sebelum dikueri menggunakan* cypher *yang disediakan oleh Neo4j.*

Kata kunci: Neo4j, Basis Data Graf, Silsilah Keluarga, Pellet, Ontologi, Dbpedia, Apache Jena.

[Halaman ini sengaja dikosongkan]

FAMILY TREE SCHEME MODELING USING NEO4J GRAPH TO FIND RELATIONSHIP BETWEEN CHARACTERS

Name	: Nuzul Ayu Safitri
NRP	: 05111540000014
Major	: Informatics Department – ITS
Supervisor I	: Nurul Fajrin Ariyani, S.Kom., M.Sc.
Supervisor II	: Adhatus Solichah Ahmadiyah, S.Kom, M.Sc.

Abstract

Knowledge is gathering facts or information that has been interpreted. One of them is in biographical information or personal data. The facts or data presented in the biography can be obtained so that new knowledge is found which arises due to the association or relationship with other figures. For example, there are family relationships such as the relationship between children and parents, or husband and wife.

The relationships between the characters can be described in an ontology which is a model for representing relationships between concepts. An ontology defines classes, properties, instances, and relations between entities in a particular domain so that it has meaning (semantic). There are several terminologies that can be applied in an ontology that is using RDF terminology called triplestore and graph database.

The purpose of this final project is to map the family genealogy data storage scheme in the form of graphs using the Neo4j help application and the Neosematics library to convert data in the form of RDF into a graph database, and also reasoning using Apache Jena with the Pellet library to produce a more complete inferred fact before queried using cypher provided by Neo4j.

Keywords: Neo4j, Graph Database, Family Tree, Pellet, Ontology, Dbpedia, Apache Jena.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"Permodelan Skema Penyimpanan Data Silsilah Keluarga Menggunakan Graf pada Neo4j untuk Mencari Relasi Antar Tokoh"***.

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis sehingga penulis bisa mendapatkan ilmu lebih dan memanfaatkan semua ilmu yang telah didapat ketika perkuliahan. Serta Tugas Akhir ini juga dapat memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Departemen Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember.

Penulis mengucapkan terima kasih sebanyak-banyaknya kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung kepada penulis selama proses pengerjaan tugas akhir, antara lain:

1. Allah SWT atas segala karunia dan rahmat-Nya yang telah diberikan selama ini.
2. Orang tua, saudara, serta keluarga penulis yang yang selalu mendukung dan memberikan doa agar penulis dapat menyelesaikan tugas akhir dengan lancar lancar.
3. Ibu Nurul Fajrin A., S.Kom., M.Sc. selaku dosen pembimbing I yang telah memberikan bimbingan, arahan serta ilmu-ilmu baru dalam pengerjaan tugas akhir ini.
4. Ibu Adhatus Sholichah A., S.Kom., M.Sc. selaku dosen pembimbing II yang telah banyak memberikan arahan dan bantuan serta waktu untuk berdiskusi sehingga penulis dapat menyelesaikan tugas akhir ini.

5. Bapak Darlis Herimurti, S.Kom, M.Sc. selaku Kepala Program Studi Informatika ITS dan segenap dosen Informatika yang telah memberikan ilmunya.
6. Amelia Velika Yukuri dan Ni Made Lidya Dewi Aristya yang telah memberi banyak semangat dan motivasi kepada penulis selama pengerjaan tugas akhir.
7. Hidayatul Munawaroh, Damai Marisa Bachri, Rozana Firdausi, Cynthia Dewi, Ramdhani Widya Iswara, Wuri Oktaviana, Reny Susanti, Rizky Mei Larasati dan Faiq yang telah memberi dukungan serta berbagi ilmu kepada penulis selama pengerjaan tugas akhir.
8. Cha Jun-ho, Jeon Won-woo, serta Seventeen yang sudah menemani hari-hari penulis selama pengerjaan tugas akhir.
9. Teman-Teman Managemen Informasi dan Algoritama Pemrograman yang sudah memberi dukungan moral bagi Penulis.
10. Teman-teman Informatika ITS angkatan 2015 yang telah membantu, berbagi ilmu dan memberi motivasi kepada penulis, kakak-kakak angkatan 2014, 2013, serta adik-adik angkatan 2016 yang mewarnai kehidupan semasa kuliah penulis.
11. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Januari 2020

Nuzul Ayu Safitri

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan tugas akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Keanekaragaman informasi yang terlahir dari kumpulan data merupakan kunci dari adanya pengetahuan yang telah diinterpretasikan. Sebagai contoh, informasi dapat ditemukan dari sekumpulan data biografi seseorang. Mulai dari nama, tempat tinggal, tanggal lahir, hingga hubungan keluarga dari orang tersebut. Informasi yang telah digali kemudian dapat kita olah kembali secara lebih dalam agar mendapat suatu pengetahuan. Dari informasi tersebut juga dapat ditemukan fakta atau informasi baru yang mana saling berkaitan yang memicu munculnya ikatan atau hubungan yang berkesinambungan sehingga membentuk sebuah pohon atau silsilah keluarga.

Hubungan keterkaitan tokoh satu dengan tokoh yang lain tersebut dapat digambarkan dalam suatu ontologi. Ontologi merupakan suatu model yang berguna untuk merepresentasikan hubungan antar konsep. Ontologi mendefinisikan *class*, *property*, *instance*, dan relasi antar entitas dalam suatu domain tertentu sehingga memiliki makna (semantik). Pada penerapannya ontologi merupakan implementasi metadata yang mana terdiri dari subjek, predikat dan objek atau di dalam terminologi RDF (*resource description framework*) disebut dengan *N-triple*. *N-triple* tersebut disimpan di dalam suatu database yang disebut dengan *triplestore*.

Namun pada realisasinya *triplestore* memiliki sejumlah kelemahan. Beberapa diantaranya adalah tidak dapat melakukan penalaran jika *triple* yang dilakukan *query* tidak ada atau belum tersimpan didalam *triplestore* dan juga mengakibatkan adanya penyimpanan data yang terlalu besar. Masalah ini dapat diatasi menggunakan *graph database*. Berbeda dengan *triplestore*, graph

database menerapkan aturan *built-in* dan *custom rule set* dalam mengambil data, sehingga dapat menciptakan data baru secara otomatis. *Graph database* melakukan penelusuran pada *nodes* dan relasi pada suatu graf, yang mana menghemat waktu agar lebih efisien dan konstan. Terlepas dari ukuran total suatu dataset, *graph database* dapat mengelola data yang saling sangat terhubung dan memiliki *query* yang kompleks. Dengan hanya sebuah pola dan satu titik awal, *graph database* mengekspolasi data tetangga disekitarnya untuk mengumpulkan dan mengagregasi informasi dari jutaan *node* dan relasi, membiarkan data di luar parameter tidak disentuh sehingga waktu yang dibutuhkan menjadi lebih efisien.

Berdasarkan ontologi yang telah disebutkan sebelumnya, pengerjaan tugas akhir ini akan mengobservasi penggunaan *graph database* dalam menangani suatu ontologi yang dalam hal ini ontologi seorang tokoh (silsilah keluarga dari tokoh tersebut) agar implementasi dalam melakukan *query* bisa menjadi lebih mudah dan serta efisien.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

a. Bagaimana memodelkan skema penyimpanan relasi silsilah keluarga pada *graph database?*
b. Bagaimana mendapatkan relasi antar dua tokoh menggunakan *query* pada *graph database*?
c. Bagaimana cara mendapatkan fakta baru tentang relasi antar tokoh yang tidak memiliki properti yang lengkap pada DBpedia?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir memiliki beberapa batasan, yakni sebagai berikut.

a. Metadata yang digunakan bersumber dari informasi yang dimuat dalam laman DBpedia.org mengenai biografi keluarga kandung tokoh.

b. Tugas akhir ini menggunakan asumsi satu tokoh hanya memiliki satu *resource* saja, apabila ada tokoh yang memiliki dua atau lebih *resource* yang berbeda maka *resource-resource* tersebut akan diabaikan.
c. Relasi hanya mengakomodasi *parent, child, spouse*, *grand child*, dan *grand parent*.
d. Pengembangan ontologi tetap menggunakan Protege.
e. *Reasoning engine* yang digunakan adalah Pellet *reasoner*.
f. Aplikasi tidak menyediakan *form* untuk pengelolaan data (tambah, ubah, hapus).
g. Aplikasi hanya untuk menampilkan data silsilah keluarga kandung tokoh saja.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah:

a. Melengkapi relasi keluarga antar tokoh pada *graph database*.
b. Mencari relasi kekeluarga antar 2 tokoh menggunakan *graph database* meningkatkan hasil pencarian relasi antar tokoh pada *graph database*.
c. Memetakan skema penyimpanan data silsilah keluarga dalam bentuk graf.

1.5. Manfaat

Tugas akhir ini diharapkan dapat mempermudah penggalian informasi ketika melakukan *query* pada data silsilah keluarga tokoh dan dapat mengurangi penggunaan memori pada *device* yang digunakan sewaktu menjumpai *query* dengan kasus yang kompleks.

1.6. Metodologi Pembuatan Tugas Akhir

1. Studi literatur

   Tahap studi literatur akan ada pencarian literatur yang berkaitan dengan pengerjakan tugas akhir yang mana dapat membantu dalam proses pengerjaan aplikasi terutama mengenai ontologi dari suatu tokoh yang akan dianalisis, OWL, RDF, DBpedia, *graph database*, LPG, Neo4j, Apache

Jena, *reasoning* secara general dan *reasoning* pada Pellet, Neosematics Java, JavaScript, dan yFiles. Studi literatur akan bersumber dari referensi tesis dan buku. Referensi lain berasal dari dokumentasi resmi internet.

2. Analisis dan desain perangkat lunak

   Pada tahap analisis akan dilakukan identifikasi ontologi sehingga didapatkan konsep yang tepat dan pengidentifikasian tahap-tahap dari setiap bagian perancangan arsitektur sistem secara runtuh. Selanjutnya hasil dari analisis akan digunakan sebagai acuan desain dari perangkat lunak.

3. Implementasi perangkat lunak

   Dalam tahap ini, dilakukan implementasi berdasarkan rancangan yang dibuat dalam tahap sebelumnya. Dimana implementasi itu meliputi implementasi dari setiap tahap pada perancangan perangkat lunak dan penguerian kasus pada Neo4j, termasuk *reasoning* yang dilakukan pada Apache Jena menggunakan *reasoner* Pellet dan visualisasinya pada Web.

4. Pengujian dan evaluasi

   Pada tahap ini dilakukan uji coba aplikasi untuk melihat hasil silsilah keluarga. Evaluasi dilakukan untuk mengetahui apakah hasil kueri dari ontologi keluarga yang diuji dapat keluar secara cepat, tepat, dan efisien.

5. Penyusunan Buku Tugas Akhir

   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini. Pada tahap ini juga disertakan hasil dari implementasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini terdiri atas beberapa bab yang tersusun secara sistematis, yaitu sebagai berikut.

1. BAB I. Pendahuluan

   Bab pendahuluan berisi penjelasan mengenai latar belakang masalah, rumusan masalah, batasan masalah,

tujuan, manfaat, metodologi yang digunakan selama penyusunan dan sistematika penulisan tugas akhir.

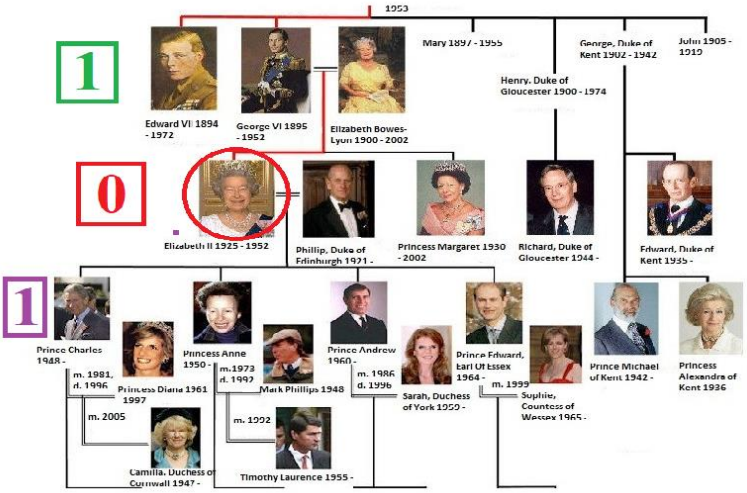
2. BAB II. Tinjauan Pustaka
   Bab tinjauan pustaka berisi hasil studi literatur yang digunakan sebagai dasar untuk menyelesaikan tugas akhir.
3. BAB III. Analsis dan Perancangan
   Bab ini berisi tentang desain perangkat lunak yang dikembangkan yang mana memiliki beberapa tahap, juga terdapat deskripsi umum sistem dan permasalahan yang dikerjakan pada tugas akhir ini. Terdapat pula tahapan dari sistem yang telah dibuat.
4. BAB IV. Implementasi Sistem
   Bab ini membahas implementasi dari pola perancangan yang telah dibuat pada tahap sebelumnya.
5. BAB V. Pengujian dan Evaluasi
   Bab ini membahas pengujian dengan metode pengujian objektif untuk mengetahui kecocokan data dan kekayaan data. Serta pengujuan secara subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.
6. BAB VI. Kesimpulan dan Saran
   Bab ini merupakan bab terakhir yang berisi tentang kesimpulan yang didapat dari proses pembuatan tugas akhir beserta saran-saran untuk pengembangan selanjutnya.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini membahas teori-teori yang mendukung pembuatan tugas akhir. Teori yang mendukung tersebut adalah deskripsi mengenai keluarga, ontologi, OWL, *graph database*, *labeled property graph*, Neo4j, Apache Jena, *reasoning* secara umum, *reasoning* ontologi silsilah keluarga pada Pellet, Neosematics, Java,. PHP, JavaScript, dan yFiles.

2.1. Silsilah Keluarga



Gambar 2.1 Silsilah keluarga Elizabeth

Menurut Duvall dan Logan, keluarga merupakan sekumpulan orang dengan ikatan perkawinan, kelahiran dan adopsi yang bertujuan untuk menciptakan, mempertahankan budaya dan meningkatkan perkembangan fisik, mental, emosional serta sosial dari tiap anggota keluarga [1]. Sedangkan menurut UU. No. 10 Tahun 1992, mendefinisikan keluarga merupakan unit terkecil dari masyarakat yang terdiri dari suami-istri atau suami-istri dan

anaknya atau ayah dan anaknya atau ibu dan anaknya. Sehingga dapat disimpulkan keluarga merupakan unit terkecil dari masyarakat yang memiliki ikatan perkawinan, kelahiran dan adopsi yang bertujuan untuk menciptakan dan mempertahankan budaya.

Silsilah sendiri memiliki arti asal-usul suatu keluarga berupa bagan atau susur galur (keturunan)[2]. Pada silsilah terdapat catatan yang menggambarkan hubungan kekeluarga beberapa generasi. Generasi dapat berupa generasi ke atas dan generasi ke bawah. Istilah generasi memiliki makna untuk menyatakan bahwa sekumpulan orang memiliki waktu atau masa hidup yang sama [3] dan digunakan untuk mempermudah penunjukan silsilah pada keluarga seorang tokoh. Pada Gambar 2.1, angka 1 dengan warna hijau menunjukan generasi pertama ke atas dari silsilah keluarga Elizabeth, sedangkan angka 1 dengan warna ungu merupakan generasi pertama ke bawah dari silsilah keluarga Elizabeth.

2.2. Ontologi

Istilah ontologi *(ontology)* berasal dari bahasa Yunani, yaitu ontos dan logos. Definisi ontologi dalam ilmu komputer menyatakan bahwa *"An ontology is an explicit specification of a conceptualization"* yang artinya adalah sebuah ontologi adalah sebuah spesifikasi yang eksplisit dari suatu konseptualisasi. Dengan demikian ontologi dalam ilmu komputer adalah cara untuk merepresentasikan suatu domain pengetahuan secara eksplisit mengenai suatu konsep dengan cara memberikan makna, properti, serta relasi pada konsep tersebut sehingga terhimpun dalam suatu domain pengetahuan dan membentuk sebuah basis pengetahuan (*knowledge base*).

Ontologi juga membuka kemungkinan suatu sistem manajemen pengetahuan serta membuka kemungkinan untuk berpindah dari pandangan berorientasi dokumen ke arah pengetahuan yang saling terkait, dapat dikombinasikan serta dapat dimanfaatkan kembali secara lebih fleksibel dan dinamis. Model ontologi yang disusun mampu mendeskripsikan informasi secara

sistematis. Hal ini membuat pengguna untuk melakukan pencarian secara lebih mudah [4].

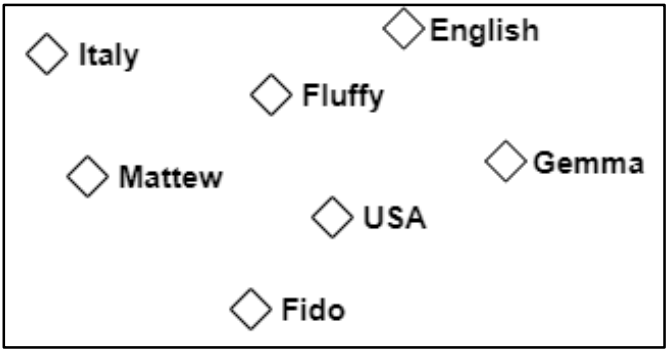
Agar ontologi web semantik dapat dikomputasikan, maka organisasi *World Wide Web Consortium (W3C)* mengeluarkan rekomendasi bahasa yang digunakan untuk mengkomputasikan ontologi. Bahasa tersebut adalah *RDF (Resource Description Framework)* dan *OWL (Web Ontology Language)* yang menggunakan bahasa *XML (Extensible Markup Languange)* sebagai dasar sintaks dalam melakukan pengkodean. *RDF (Resource Description Framework)* digunakan untuk mendefinisikan sumber daya web *(web resource)* dalam bentuk *triple*, sedangkan OWL (Web Ontology Language) digunakan untuk memberi pernyataan yang lebih ekspresif. Dalam melakukan akses pada sumber daya web *(web resource)* dapat dilakukan *query* menggunakan bahasa *SPARQL (SPARQL Protocol and RDF Query Language)* Kegunaan ontologi secara umum sebagai *controller vocabulary, semantic interoperability, knowledge sharing,* dan *reuse* [5]. Davies menjelaskan manfaat ontologi, yaitu:

1) Menjelaskan domain secara eksplisit;
2) Berbagi pemahaman dari informasi yang tersetruktur.
3) Penggunaan ulang domain pengetahuan

Elemen pada ontologi web semantik terdiri atas *instance, property, class* dan *axiom.* Berikut ini akan dijelaskan secara singkat elemen dasar pembentuk ontologi.

a) Instance

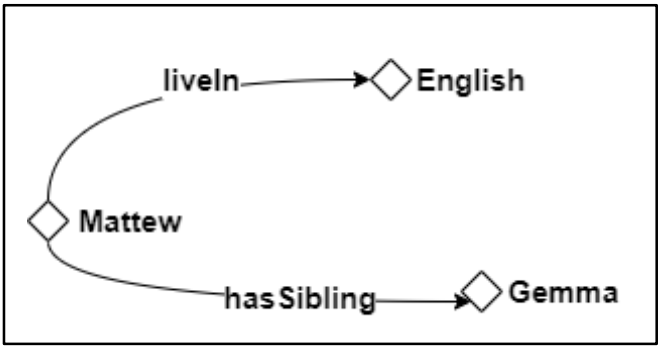
Instance atau disebut juga individual adalah anggota (member) dari *classes*. *Instance* ini dapat dipandang sebagai objek yang ada pada domain yang dibahas. Sebagai contoh seperti pada Gambar 2.2 yaitu Italy, England, Matthew, Fluffy, USA, Gemma, dan Fido.



Gambar 2.2 Contoh *Instance*

b) Property

Property merupakan *binary relation*. Ada dua jenis *property* pada ontologi web semantik, yaitu *object property* dan *datatype property*. *Object property* digunakan untuk menghubungkan objek dengan objek lainnya sedangkan *datatype property* digunakan untuk menghubungkan objek dengan *datatype value* seperti *text, string atau number*.



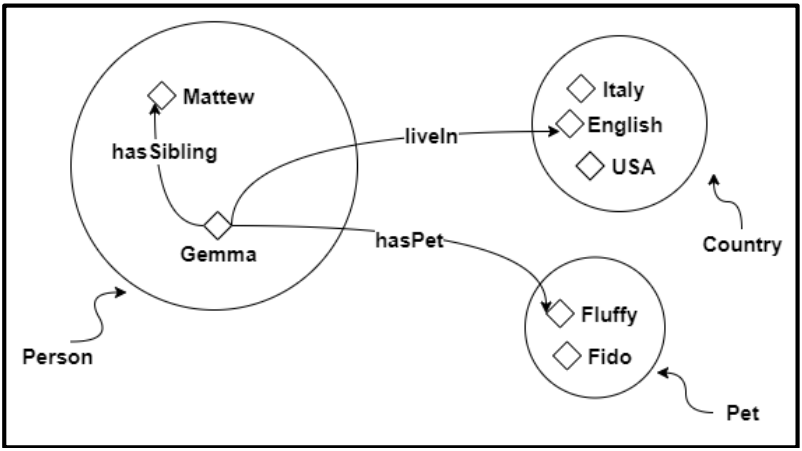
Gambar 2.3 Contoh *Property*

Gambar 2.3 di atas menjelaskan bahwa *instance* Matthew memiliki hubungan *(property)* dengan England

dimana dia tinggal disana serta hubungan Matthew dengan Gemma sebagai saudara kandung.

c) Class

Class merupakan titik pusat ontologi. *Class* menjelaskan sebuah konsep dalam suatu domain yang terdiri dari beberapa *instance*. *Class* juga dikenal sebagai *concept, object* dan *categories*. Sebuah class memiliki *subclass* yang ditujukan untuk menyatakan *concept* lebih spesifik dari *superclass*.



Gambar 2.4 Contoh *Class*

Dari Gambar 2.4 di atas dapat dilihat bahwa class Person memiliki *instance* Gemma dan Matthew.

d) Axiom

Axiom merupakan aturan eksplisit yang digunakan untuk membatasi nilai dari *class* maupun *instance*. *Property* dari relasi adalah jenis *axiom.*

Teori ontologi ini digunakan sebagai pedoman untuk pembuatan ontologi yang memuat silsilah keluarga didalamnya.

2.3. OWL

Ontologi silsilah keluarga yang dibuat menggunakan format OWL untuk penetapan *rules-rules* atau aturan-aturan yang ada didalamnya. *Ontology Own Language* (OWL) sendiri adalah suatu bahasa yang dapat digunakan oleh aplikasi-aplikasi yang bukan sekedar menampilkan informasi tersebut pada manusia, melainkan juga yang perlu memproses isi dari informasi. Dengan menggunakan OWL, kita dapat menambah *vocabulary* tambahan disamping semantika formal yang telah dibuat sebelumnya menggunakan XML, RDF, dan RDF Schema. Hal ini sangat membantu penginterpretasian mesin yang lebih baik terhadap isi Web. OWL ditulis menggunakan W3C XML standart. OWL memiliki 3 elemen utama yaitu *classes, property,* dan *instance* [6].

a) Class

OWL mendefinisikan *root* dari semua yang ada dengan `owl:Thing`. Jadi semua kelas yang dibuat secara implisit merupakan subkelas `owl:Thing`. Pembuatan kelas menggunakan `owl:Class` dan menyatakan subkelas dengan `rdfs:subClassOf.`

b) Instance

Instance atau disebut juga individu adalah anggota *(member)* dari kelas. *Instance* ini dapat dipandang sebagai objek yang ada pada domain yang dibahas. Sama seperti `owl:Class` yang menjadi meta level untuk kelas, begitu pula kelas yang telah didefinisikan menjadi meta level untuk *instance.*

c) Property

Property atau properti merupakan *binary relation*. Ada dua jenis properti pada OWL, yaitu *ObjectProperty* (relasi antara *instance* dari dua kelas) dan *DatatypeProperty* (relasi antara *instance* dengan RDF literal dan tipe data XML Schema). Sama halnya seperti kelas yang dapat dinyatakan secara hierarki, begitu pula properti dapat dinyatakan sebagai `subPropertyOf` dengan `rdfs:subPropertyOf`. Untuk memberikan

batasan pada suatu properti dapat digunakan `rdfs:domain` dan `rdfs:range` yang disebut juga sebagai *global restriction* karena berlaku untuk umum dan tidak terbatas pada kelas tertentu.

Ada 2 (dua) hal terkait dengan properti, yaitu:

- *Characteristic*, memberikan tambahan keterangan untuk properti, yaitu *inversOf*, *TransitiveProperty*, *SymmetricPropery*, *FunctionProperty*, dan *InverseFunctionalProperty*.
- *Restriction*, disebut juga sebagai *local restriction* karena memberikan batasan pada definisi suatu kelas, seperti pada contoh yang diberikan sebelumnya tentang kardinalitas dengan *restriction* dan *onProperty*. Ada tiga macam *restriction*, yaitu *quantifier*, *cardinality*, dan *hasValue*. Untuk menentukan *quantifier* digunakan *allValuesFrom* dan *someValuesFrom*.

2.4. RDF

Resource Description Framework (RDF) adalah sebuah bahasa yang merepresentasikan informasi mengenai sumber pada *World Wide Web* dan juga sebagai model standar untuk pertukaran data pada web. RDF memiliki fitur yang menfasilitasi penggabungan data bahkan jika skema yang mendasarinya berbeda, dan secara spesifik mendukung evolusi skema dari waktu ke waktu tanpa mengharuskan semua data konsumen diubah.

RDF memperluas struktur penautan web dalam menggunakan URI untuk memberi nama pada relasi antar sesuatu dengan sesuatu yang lain yang mana biasanya disebut dengan *triple*. Menggunakan model sederhana ini, memungkinkan data struktur dan semi-terstruktur dapat digabungkan, diekspos, dan dibagikan di berbagai aplikasi [6].

Sebuah *resource* dapat menunjukkan sebuah entitas (*object*), sebuah tipe entitas (*class*), ataupun sebuah relasi. Setiap *resource*

ditandai sebuah *IRI (Internationalized Resource Identifier)*, yang mana dapat membuat entitas tersebut dengan unik dan secara umum dapat teridentifikasi. Setiap pernyataan RDF adalah sebuah *triple*, terdiri dari komponen *subject*, *predicate*, dan *object* [7].

a) *Subject* selalu sebuah IRI atau *blank node* yang menandakan sebuah *resource.*
b) *Predicate* juga sebuah IRI yang mana menerangkan sebuah relasi atau properti dari *subject.*
c) *Object* dapat berbentuk IRI (merujuk pada *resource* lain), sebuah literal, atau *blank node*.

Secara umum IRI pada sebuah graf RDF adalah sebuah Unicode string (UNICODE). IRI dalam sintaksis abstrak RDF harus mutlak, dan dapat berisi pengidentifikasi fragmen. IRI merupakan generalisasi dari URI yang memungkinkan jangkauan karakter Unicode karakter yang lebih luar. Setiap URI dan URL pasti sebuah IRI, namun tidak semua IRI adalah sebuah URI. Ketika IRI digunakan dalam operasi yang hanya ditentukan untuk URI, IRI harus dikonversi terlebih dahulu sesuai dengan pemetaan sebelum digunakan pada operasi yang hanya dapat dikerjakan dengan format URI.

Pada RDF terdapat istilah RDF *vocabulary* yang merupakan sekumpulan IRI yang digunakan pada graf RDF. IRI dalam RDF *vocabulary* sering dimulai dengan substring umum yang dikenal sebagai *namespace* IRI. Beberapa *namespace* IRI dikaitkan oleh konvensi dengan nama pendek yang dikenal sebagai *namespace prefix* seperti contoh pada Tabel 2.1.

Tabel 2.1 Contoh *Namespace* IRI dan *Namespace Prefix*

Namespace prefix	Namespace IRI	RDF vocabulary
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	The RDF built-in vocabulary [RDF11-SCHEMA]

rdfs	http://www.w3.org/2000/01/rdf-schema#	The RDF Schema vocabulary [RDF11-SCHEMA]
xsd	http://www.w3.org/2001/XMLSchema#	The RDF-compatible XSD types

Dalam beberapa format serialisasi, biasanya untuk menyingkat IRI yang dimulai dengan *namespace* IRI dengan menggunakan *namespace prefix* untuk membantu keterbacaan. Sebagai contoh, IRI "http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral" akan disingkat menjadi rdf: XMLLiteral. Namun perlu dicatat bahwa singkatan ini bukan IRI yang valid. *Namespace* IRI dan *namespace prefix* bukan merupakan bagian formal dari model data RDF. Mereka hanyalah sintaksis untuk menyingkat IRI agar nyaman digunakan pada RDF [8].

Tabel 2.2 Contoh Kasus RDF

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988

Dengan contoh kasus seperti pada Tabel 2.2, hasil dari RDF *triplestore* dapat berbentuk seperti pada Tabel 2.3. Pada tabel nomor satu ditunjukan bahwa *subject "*Empire Burlesque" memiliki *predicate "artist"* dengan *object "*Bob Dylan". Pada kasus ini *subject* dan *predicate* merupakan IRI sedangkan object merupakan literal[9].

Tabel 2.3 Contoh dari Hasil Kasus RDF

No.	Subject	Predicate	Object
1	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#artist	"Bob Dylan"
2	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#country	"USA"
3	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#company	"Columbia"
5	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#year	"1985"
6	http://www.recshop.fake/cd/Hide your heart	http://www.recshop.fake/cd#artist	"Bonnie Tyler"

Teori RDF pada subbab ini digunakan sebagai format pada dokumen yang berisi informasi yang memuat tokoh-tokoh yang digunakan sebagai sumber data pada tugas akhir ini. Yang selanjutnya format tersebut juga akan berpengaruh ketika format data diubah menjadi basis data graf.

2.5. DBpedia

Sumber data yang akan digunakan dan dioleh merupakan sumber data yang bersalah dari DBpedia. DBpedia adalah sebuah *knowlegde-base* berskala besar yang mengeksploitasi atau mengekstraksi struktur data dari Wikipedia. DBpedia memiliki cakupan pengetahuan manusia yang luas, keunggulan ini dapat menjadikan adanya hubungan, tidak hanya dengan satu tempat dataset saja tapi juga dapat terhubung dengan dataset lain yang bersifat data terbuka (*open source*), yang mana dari keunggulan tersebut ditemukan keunggulan lain yaitu memungkinkan untuk menghasilkan data DBpedia yang lebih kaya [7].

Seperti yang terlihat pada Gambar 2.5 yaitu laman DBpedia Prince Harry (http://dbpedia.org/page/Prince_Harry), DBpedia

memiliki dua bagian, yaitu *property* dan *value*. *Property* berisi *namespace prefixes* resmi yang dimiliki oleh DBpedia. *Property* dapat berubah menjadi *object property* maupun *datatype property*, dan akan menjadi predikat (*predicate*) jika dijadikan *triplestore*. Sedangkan *value* berisi data berupa URI maupun literal yang akan menjadi objek (*object*) pada *triplestore*. Beberapa *Namespace prefixes* bisa dilihat pada Tabel 2.4.

Gambar 2.5 Laman DBpedia Prince Harry

Tabel 2.4 Daftar *Namespace Prefix pada DBpedia*

Prefix	URL	Description
DBpedia Namespace		

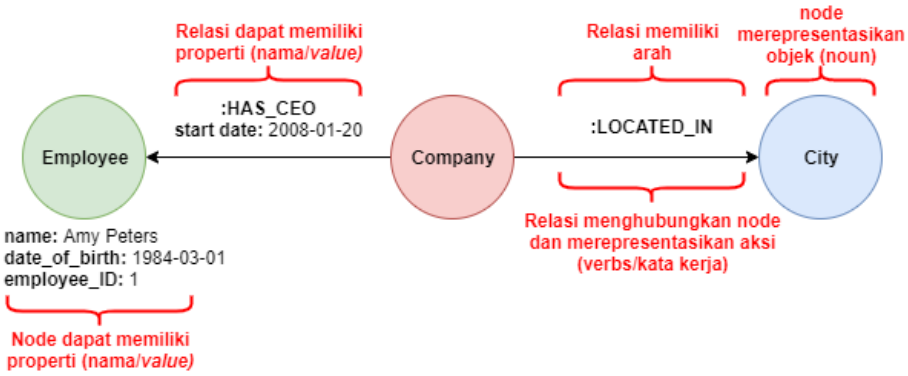
Prefix	URL	Description
dbr	http://dbpedia.org/resource/	*One-to-one mapping* antara artikel Wikipedia dengan sumber daya DBpedia
dbp	http://dbpedia.org/property/	Properti dari raw *infobox extraction*
dbo	http://dbpedia.org/ontology/	Ontologi DBpedia
yago	http://dbpedia.org/class/yago/	*Yet Another Great Ontology*
External namespaces		
dct	http://purl.org/dc/elements/1.1/	*Dublin core*
foaf	http://xmlns.com/foaf/0.1/	*Friend of a friend*
owl	http://www.w3.org/2002/07/owl#	*W3C web ontology language*
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	*Standart W3c RDF vocabulary*
rdfs	http://www.w3.org/2000/01/rdf-schema#	*Extention of the basic RDF vacabulary*

2.6. Graph Database

Graph database adalah suatu basis data yang dirancang untuk memperlakukan relasi antar data menjadi sama pentingnya dengan data itu sendiri. Ini dimaksudkan untuk menyimpan data tanpa membatasinya dengan model yang telah ditentukan sebelumnya. Sebagai gantinya, data disimpan seperti pertama kali dimasukan

atau digambarkan dimana menunjukan bagaimana masing-masing entias terhubung atau terkait dengan yang lain[10].

Seperti kebanyakan teknologi, ada beberapa pendekatan berbeda yang membuat komponen sebuah *graph database* terbentuk. Salah satu pendekatan tersebut adalah model *graph property*, dimana data disusun sebagai *node*, relasi dan proporti (data disimpan pada *node* atau relasi). Berikut adalah komponen-komponen dari model *graph property* yang dapat dilihat pada gambar:



Gambar 2.6 Contoh Kasus pada *Graph Database*

a) *Node*: entitas dalam graf. *Node* dapat menyimpan sejumlah atribut (pasangan *key-value*) yang disebut properti. *Node* dapat ditandai dengan label, mewakili perbedaan *rule* didalam suatu domain. label dari suatu *node* juga dapat berfungsi untuk melampirkan metadata (seperti indeks atau batasan informasi) pada *node* tertentu.
b) Relasi: relasi menyediakan arah, dinamai, relevan secara semantik antara dua entitas atau *node* (misalnya, Employee WORKS_FOR Company seperti pada Gambar 2.6). Sebuah relasi selalu memiliki arah, tipe, *node* awal, dan *node* akhir. Seperti halnya *node*, relasi juga dapat

> memiliki properti. Dalam kebanyakan kasus, relasi memiliki sifat kuantitatif, layaknya bobot, jarak, biaya, peringkat, interval waktu, atau kekuatan. Karena cara efisien hubungan disimpan, dua node dapat berbagi nomor atau jenis hubungan apa pun tanpa mengorbankan kinerja. Meskipun mereka disimpan dalam arah tertentu, relasi selalu dapat diarahkan secara efisien di kedua arah [11].

Teori *graph database* atau basis data graf ini digunakan sebagai format lain dari data selain RDF. Yang mana basis data graf ini merupakan rancangan basis data pada ada pada Neo4j. Sehingga untuk menggunakan dokumen atau sumber data yang awalnya berformat RDF, turtle, n-triples, dan lain sebagainya perlu melakukan konversi data sebelum bisa digunakan pada Neo4j.

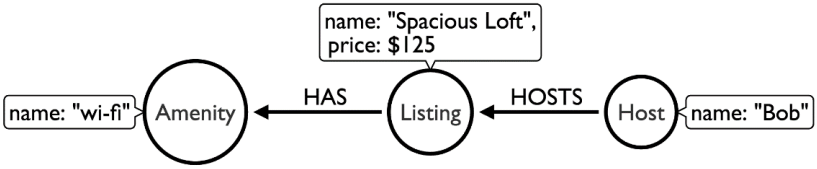
2.7. Labeled Property Graph (LPG)

Labeled property graph adalah sebuah model graf yang terbentuk dari *node*, properti (*property)*, relasi (*relationship)* dan label (*labels*). Yang mana LPG ini digunakan menjadi pedoman dari model pada basis data graf atau *graph database*. *Node* mengandung properti. Sebuah *node* dapat diibaratkan seperti sebuah dokumen yang menyimpan properti dalam bentuk pasangan *key-value* yang dapat dinyatakan secara bebas. *Node* dapat ditandai dengan satu atau lebih label. Dalam hal ini label memberikan keterangan untuk mengelompokkan jenis *node* serta peran yang dimiliki oleh *node* tersebut.

Relasi atau *relationship* berfungsi untuk menghubungkan *nodes* yang ada dengan struktur dari graf. Sebuah relasi selalu memiliki arah, sebuah nama, dan sebuah *start node* dan *end node*. Jika arah dan nama relasi digabungkan akan menambah kejelasan semantik pada proses pembentukan struktur pada *node-node*. Selayaknya *node*, relasi juga dapat memiliki properti. Kemampuan untuk menambah properti pada relasi sangat berguna untuk menyediakan metadata tambahan yang dapat digunakan pada algorithma graf, penambahan semantik tambahan pada relasi

(termasuk kualitas dan berat), dan untuk pembatasan kueri saat *runtime* [10].

Gambar 2.7 menjelaskan tentang salah satu kasus pasa LPG, Host, Listing dan Amenity merupakan *nodes* yang setiap *node*-nya memiliki properti masing-masing seperti host yang memiliki properti *name* "Bob, Listing dengan properti *name* "Spaciouse Losf" dan properti *price* "$125", serta Amenity yang memiliki properti *name* "wi-fi". Sedangkan HAS san HOSTS merupakan relasi yang menghubungkan ketiga *nodes* tersebut. Untuk relasi HOSTS, *node* yang menjadi *start node* adalah Host dengan Listing sebagai *end node*-nya. Dan untuk relasi HAS, *start node*-nya adlah Listing dengan *end node* Amenity. Nodes yang berelasi tersebut terbentuk dan saling berkaitan menjadi sebuah graf. Teori permodelan LPG ini digunakan sebagai model pada basis data graf di tugas akhir ini.



Gambar 2.7 Contoh Kasus LPG

2.8. Neo4j

Neo4j adalah sebuah *open-source*, *NoSQL*, *native graph database* yang menyediakan sebuah *ACID-compliant transaction backend* untuk sebuah aplikasi. Pengembangan awal dimulai dari tahun 2003, tetapi dipublikasi secara umum sejak 2007. *Source code* pada aplikasi ini ditulis dalam Java dan Scala [11].

Neo4j disebut sebagai *native graph database* karena dapat secara efisien mengimplementasikan model *property graph* hingga ke tingkat penyimpanan. Hal ini berarti bahwa data yang dimasukan disimpan sedemikian rupa seperti sebuah papan tulis, dan database menggunakan *pointer* untuk mengarahkan dan

melintasi graf. Berbeda dengan *graph processing* atau *in-memory libararies (triplestore)*, Neo4j juga menyediakan basis data karakter secara lengkap, termasuk *ACID transaction compliance*, *cluster support*, dan *runtime failover* –agar cocok digunakan pada graf untuk data dalam skenario produksi. Berikut adalah beberapa fitur unggulan yang disediakan Neo4j:

a) *Cypher*: sebuah bahasa *query* deklaratif yang mirip dengan SQL. Akan tetapi dioptimalkan untuk graf. Sekarang digunakan oleh database lain seperti SAP, HANA Graph dan Redis graph melalui openCypher.
b) *Constant time traversals*: lintasan waktu yang konstan pada graf yang dalam dan luas dikarenakan untuk mengefisiensi representasi pada *node* dan relasi, memungkinkan peningkatan hingga milyaran *node* pada *hardware*.
c) *Flexible*: skema *property graph* yang fleksibel yang mana dapat beradaptasi dari waktu ke waktu, memungkinkan untuk diwujudhkan dan menambah relasi baru kemudian untuk memintas dan mempercepat domain data ketika bisnis perlu diubah.
d) *Driver*: aplikasi ntuk bahasa populer, termasuk, Java, JavaScript, .Net, Python, dan banyak lagi.

Aplikasi browser Neo4j digunakan sebagai alat yang menyediakan basis data graf dan menjadi salah satu alat untuk mengimplementasikan beberapa tahap perancangan pada tugas akhir ini.

2.9. Apache Jena

Framework Apache Jena digunakan sebagai alat untuk mengimplementasi tahap penalaran untuk mendapatkan fakta-fakta baru dari sejumlah data yang dipakai pada tugas akhir ini. Apache jena sendiri adalah sebuah *Java framework* untuk mengkonstruksi *Sematic Web Applicatons.* Apache Jena menyediakan sebuah lingkungan programatik untuk RDF, RDFS, OWL, SPARQL, dan juga termasuk *rule-based inference engine*

untuk melakukan *reasoning* berdasarkan ontologi OWL dan RDFS, dan sejumlah tempat penyimpanan untuk menyimpan RDF *triples* di memori atau pada *disk*. Data yang sedang diolah direpresentasikan dalam bentuk model. Pengkuerian pada model dapat dilakukan dengan bantuan SPARQL. *Framework* memiliki berbagai jenis *internal reasoners*, salah satunya adalah Pallet *reasoner* [12].

2.10. *Reasoning*

Reasoning atau penalaran adalah proses berpikir tetang berhagai hal dengan cara yang logis dan rasional. Hal ini dianggap sebagai kemampuan bawaan manusia yang telah terbentuk layaknya logika, matematika dan kecerdasan buatan. Proses penalaran digunakan untuk membuat keputusan, memecahkan masalah dan mengevaluasi sesuatu. Sesuatu tersebut dapat berbentuk formal maupun non-formal, *top-down* maupun *bottom-up* dan berdasarkan perbedaan dalam hal penanganan ketidakpastian dan kebenaran yang parsial [13]. Berikut beberapa jenis penalaran:

a) *Deductive Reasoning*

Penalaran deduktif bekerja dari umum ke spesifik berdasarkan fakta-fakta yang ada. Dengan kata lain, penalaran tersebut dimulai dengan teori dan bekerja untuk menemukan pengamatan yang terkonfirmasi. Proses tersebut sering disebut sebagai logika *top-down*. Penalaran deduktif menjamin kesimpulan yang benar jika premis dari argumennya benar, dan argumennya valid (logis). Ada beberapa jenis penalaran deduktif yaitu *modus ponens, modus tollens,* dan *silogisme* [14]. Salah satu contoh penalaran ini adalah sebagai berikut:

- Premis 1: semua mahasiswa adalah lulusan SMA
- Premis 2: saya adalah mahasiswa
- Kesimpulan: saya lulusan SMA

b) *Inductive Reasoning*

Penalaran induktif adalah sebuah logika *bottom-up* yang dilakukan untuk mengambil suatu kesimpulan bersifat umum atau membuat suatu pernyataan baru dari kasus-kasus khusus. Beberapa jenis penalaran induktif adalah generalisasi, analogi dan kausal. Salah satu contoh penalaran ini adalah sebagai berikut:

- Premis 1: Buah mangga berwarna hijau dan rasanya manis.
- Premis 2: Buah Jambu biji berwarna hijau dan rasanya manis.
- Kesimpulan generalisasi: semua buah berwarna hijau rasanya manis

c) *Abductive Reasoning*

Abductive reasoning adalah sebuah bentuk logika yang menebak teori atau kesimpulan untuk menjelaskan pengamatan. Penalaran ini berjenis logika *bottom-up*. *Abductive reasoning* tidak menjamin bahwa suatu teori secara logis benar. Dengan kata lain, penalaran tersebut adalah sebuah metode estimasi atau pembentukan teori. Jika banyak teori atau kesimpulan, kesimpulan yang paling sederhana dan paling mungkin umumnya yang terpilih [15]. Salah satu contoh penalaran ini adalah sebagai berikut:

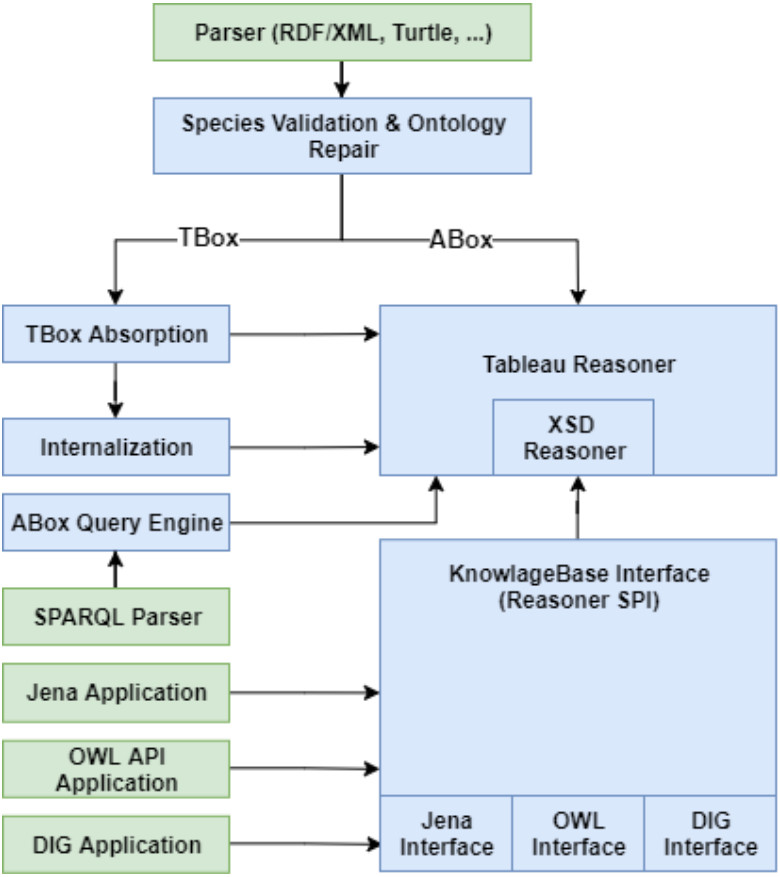
- Jika seseorang tidak makan makanan sehat, maka seseorang itu bisa sakitKesimpulan: seseorang itu sakit, maka bisa disimpulkan seseorang itu tidak makan makanan sehat.

Pada tugas akhir ini, *reasoning* yang digunakan adalah *deductive reasoning.*

2.11. *Reasoning* Ontologi Silsilah Keluarga pada Pellet

Pada dasarnya, Pellet adalah sebuah *Description Logic reasoner*. Bagaimanapun, tidak seperti DL reasoner lainnya, Pellet memang didesain untuk OWL. Pilihan desain tersebut memiliki pengaruh besar dalam keseluruhan arsitektur. Hal tersebut

dipengaruhi oleh bagaimana *tableaux reasoner* diimplementasikan, sebagai contoh kemampuan untuk melakukan *reason* dengan *instance data* (*Abox reasoning)* tanpa menggunakan *Unique Name Assumption* (UNA), dan apa saja modul yang dibutuhkan, misalnya memiliki XML *Schema datatype reasoner* dan sebuah *query engine* [16].



Gambar 2.8 Arsitektur Sistem pada Pellet

Gambar 2.8 menunjukan komponen utama dari Pallet. Inti dari sistem adalah *tableaux reasoner* yang memeriksa kestabilan dari *knowledge base.* Reasoner tersebut berpasangan dengan *datatype* oracle yang juga dapat mengecek kestabilan dari konjugasi yang terbentuk dari XML *Schema simple datatypes.* Ontologi OWL dimuat kedalam *reasoner* setelah proses *species validation* dan *ontology repair.*

Langkah ini memastikan jika semua sumber memiliki tipe *triple* yang tepat dan menambahkan *missing type declarations* berdasarkan beberapa hirarki yang telah ditentukan. Ketika fase *loading* atau memuat data, axiom kelas-kelas diletakkan pada *TBox component* dan pernyataan mengenai *instance* disimpan pada *ABox component.* TBox axiom beralan melalui *standard preprocessing* dari *DL reasoner,* misalnya normalisasi, absorsi dan internalisasi, sebelum proses tersebut dikirim ke *tableaux reasoner.* Sistem menyediakan sebuah lapisan tipis untuk programmatic acces melalui *Service Programming Interface (SPI)* yang menyediakan fungsi yang memudahkan untuk mengakses *reasoning services provided.*

Pellet menyediakan berbagai macam *interface* untuk memuat ontologi. Pellet sendiri tidak memiliki pengurai RDF/OWL tetapi terintegrasi dengan berbagai macam *toolkits* untuk RDF/OWL yang menyediakan *parser* (perngurai). Pada tugas akhir ini Pellet, dengan Apache Jena sebagai *framework* digunakan sebagai alat untuk *reasoning* pada data-data tokoh agar mendapatkan fakta-fakta baru dengan aturan yang bersumber pada ontologi silsilah keluarga.

2.12. Neosematics

Neosematics atau NSMNTX adalaha sebuah *plugin* yang memungkinkan penggunaan RDF pada Neo4j. RDF adalah sebuah model standar W3C untuk penukaran atau *interchange* data. Keefektifitasan memungkinkan neosematics melakukan penyimpanan berupa data RDF pada Neo4j tanpa menghilangkan informasi yang ada atau triple pada pemrosesannya (*lossless manner*) dan dapat pula digunakan untuk mengekspor data

property graph dari Neo4j sebagai bentuk RDF. Fitur lain yang dimiliki oleh Neosematics termasuk *model mapping* dan *inferencing* [17].

Pada tahap impor data, Neosematics akan mengubah *dataType properties* menjadi *property keys*, sedangkan *object properties* diubah menjadi relasi yang menghubungkan *nodes*. Setiap *node* merepresentasikan sebuah sumber data dan memiliki properti dengan sebuah URI didalamnya. Sama halnya dengan rdf:type yang bertransformasi menjadi *node labels*. Terminologi atau *vocabulary* diterapkan secara tetap, dan URI yang mengidentifikasikan elemen-elemen dalam data RDF (*resources*, *properties*, dan lain-lain) memiliki *namespace* yang lebih pendek untuk memudahkan pembacaan untuk kemudian dilakukan kueri.

Sebagai contoh, "http://neo4j.org/vocab/sw#name" disingkat menjadi "ns0__name", begitupun dengan "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" yang disingkat menjadi "rdf__" dan seterusnya. Ini merupakan *namespace prefix* yang secara tetap dikonversi ketika dilakukan pengimporan data.

Pada tugas akhir ini, Neosematics yang merupakan *plugin* yang dipasang pada Neo4j digunakan sebagai *plugin* untuk memuat data-data tokoh dengan format RDF menjadi basis data graf. Selain itu *plugin* ini juga berfungsi sebagai pengubah atau pengekpor data dari basis data graf menjadi format RDF.

2.13. Java

Bahasa pemrograman Java digunakan pada pembuatan *script* untuk melakukan *reasoning* data dengan *framework* Apache Jena dan *reasoner* Pellet. Java sendiri adalah sebuah bahasa permrograman umum yang konkuren, *class-based*, *object-oriented*, dan dirancang khusus untuk memiliki sedikit kemungkinan dependensi pada implementasinya. Hal ini dimaksudkan agar pengembang aplikasi "*write one, run everywhere*" (WORA) yang artinya kode Java yang dikompilasi dapat berjalan di semua platform yang mendukung Java tanpa perlu melakukan kompilasi ulang [18].

Pada penggunaannya, *script* yang menggunakan bahasa pemrograman Java mengimpor sejumlah *library* atau *plugin* yang berhubungan dengan Apache Jena dan *reasoner* Pellet. Sehingga program Java yang dibuat dapat menalarkan data dengan format RDF pada *script* Java. *Plugin*, *library*, dan *package* tersebut adalah "apache-log4j-extras-1.2.17" dan "pellet-jena". Fungsi yang digunakan dari kedua *library* tersebut merupakan *class* atau *method* untuk mengubah data menjadi model sebelum menggabungkan keduanya untuk kemudian di reasoning dengan fungsi yang ada pada *library* Pellet.

2.14. Perbandiangan dengan teknologi terdahulu

Sistem yang terdapat pada tugas akhir ini merupakan salah satu pembanding dari teknologi yang sudah ada terdahulu. Diantaranya adalah penelitian tugas akhir yang dikembangkan oleh Madis Saralita [20] dan Faiq [21].

Penelitian tugas akhir yang dimiliki Madis berfokus pada relasi antar tokoh sejarah dengan menggunakan 3 ontologi yang berbeda, yaitu *Family Relationships Ontology*, FOAF, dan BIO. *Family Relationships Ontology* adalah ontologi untuk mengetahui hubungan dalam sebuah keluarga. Sedangkan FOAF (*Friend of a Friend*) adalah sebuah skema berbasis RDF untuk mendefinisikan seseorang dan jaringan sosialnya secara semantik. Ontologi yang lain yaitu BIO *vacubalary* merupakan sebuah dokumen berisi *vocabulary* untuk informasi biografi sehingga biografi hidup seseorang dapat digambarkan dengan ontologi. Skema dari tiga ontologi tersebut kemudian digabungkan dan digunakan sebagai cara untuk mencari relasi antar tokoh sejarah dengan bantuan Apache Jena dan Pellet sebagai *reasoning*-nya [20].

Sedangkan penelitian tugas akhir milik Faiq, adapun kemiripan dengan milik Madis yaitu membuat sistem yang dapat menampilkan pohon keluarga dari seorang tokoh sejarah, yang mana memuat hubungan keluarga antar tokohnya. Pohon keluarga tersebut ditampilkan pada sebuah halaman web untuk menampilkan visualiasi dari relasi-relasi antar tokohnya. Ontologi yang digunakan pada penelitian milik Faiq adalah *Family*

Relatonships Ontology dan *FOAF*, yang mana diketahui bahwa dua ontologi tersebut digunakan pula untuk menjadi acuan pada penelitian pada tugas akhir ini. Sedangkan untuk penyimpanannya, penelitian milik Faiq menggunakan basis data pada Apache Jena Fuseki [21].

Persamaan dari ketiga penelitian yang ada adalah ketiganya sama-sama menggunakan Apache Jena dan *reasoner* Pallet untuk penalarannya, dan juga menggunakan ontologi yang sama untuk pemecahan kasusnya. Adapun perbedaan dari dua sistem sebelumnya adalah penggunaan basis data graf atau *graph database*, disamping penggunaan data berbentuk RDF untuk penyelesaian kasus mencari relasi antar dua tokoh. Bentuk graf tersebut juga dijadikan sebagai basis data pada Web yang digunakan sebagai visualisasi dari kueri yang diimplementasikan untuk penyelesaian kasus. Kekurangan dan kelebihan masing-masing penelitian dapat dilihat pada Tabel 2.5.

Tabel 2.5 Perbandingan Tiga Penelitian

Nama Peneliti	***Ontologi***	***Format Data***	***Kelebihan***	***Kekurangan***
Madis	*Family Relationships Ontology*, FOAF, dan BIO	RDF	*Reasoning* dilakukan pada Apache Jena menggunakan *reasoner* Pellet. Deskripsi dari tokoh jelas terlihat pada	Tidak bisa mengkueri kasus relasi dari dua tokoh yang tidak memiliki relasi yang lengkap

Nama Peneliti	***Ontologi***	***Format Data***	***Kelebihan***	***Kekurangan***
			tampilan antar muka	
Faiq	*Family Relationships Ontology* dan FOAF	RDF	*Reasoning* dilakukan pada Apache Jena menggunakan *reasoner* Pellet. Pohon keluarga inti dari tokoh dapat dilihat jelas.	Tidak bisa mengkueri kasus relasi dari dua tokoh yang tidak memiliki relasi yang lengkap
Nuzul	*Family Relationships Ontology* dan FOAF	Graph Database	Dapat mengkueri kasus relasi dari dua tokoh yang tidak memiliki properti lengkap	Tidak dapat melakukan reasoner secara langsung pada Neo4j.

BAB III
ANALISIS DAN PERANCANGAN

Bab ini membahas tahap analisis permasalahan dan perancangan tugas akhir. Pada bagian awal dibahas mengenai analisis permasalahan yang ingin diselesaikan. Selanjutnya dibahas mengenai pengolahan data dan tahap-tahap yang dilakukan dalam memodelan skema penyimpanan data dengan menggunakan graf, mulai dari menganalisi data hingga proses *reasoning* sebelum data dikueri pada sistem.

3.1. Cakupan Permasalahan

Permasalahan yang diangkat pada tugas akhir ini adalah menentukan cara untuk memodelkan skema penyimpanan relasi silsilah keluarga atau pohon keluarga pada *graph database* suatu tokoh tertentu.

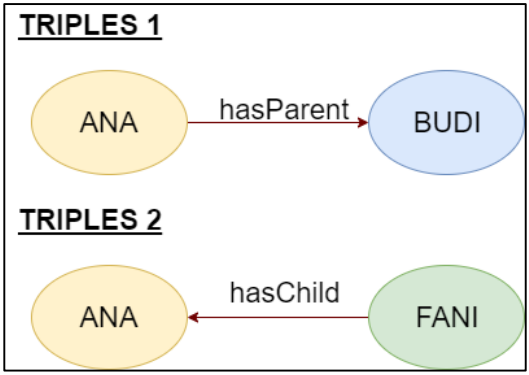
Adapun permasalahan lain yaitu ketidakmampuan data yang memiliki bentuk RDF melakukan kueri untuk dua tokoh yang tidak saling berelasi atau dengan kata lain *triplestore* tidak lengkap sehingga jika dikueri menggunakan SPARQL kemungkinan fakta tidak akan ditemukan.

Sedangkan pada Neo4j yang menggunakan *graph database* sebagai konsep dan LPG sebagai model, permasalahan ini dapat ditanggulangi dengan cara menelusuri *path* atau jalan dari dua *node* atau titik asal yang diinginkan hingga *path* tersebut kemudian bertemu di titik tertentu yang menghubungkan dua titik tersebut yang mana akan muncul relasi tidak langsung antar tokoh.

Sebagai contoh perbandingan antar *triplestore* pada Gambar 3.1 dengan basis data graf pada Gambar 3.2. Pada Gambar 3.1, terlihat bahwa ada 2 *triples* yang tidak saling terhubung seperti yang terlampir pada Tabel 3.1. Jika terdapat contoh kasus untuk mencari hubungan antara BUDI dan FINA. Kueri pada SPARQL tidak akan menemukan hubungan antara keduanya, karena tidak adanya *triples* untuk relasi BUDI dan FINA pada *file* RDF.

Tabel 3.1 Contoh Kasus *Triplestore* dengan *Graph Database*

Triple	S	P	O
Triples 1	ANA	hasParent	BUDI
Triples 2	FANI	hasChild	ANA



Gambar 3.1 Penggambaran Contoh Kasus untuk *Triplestore*

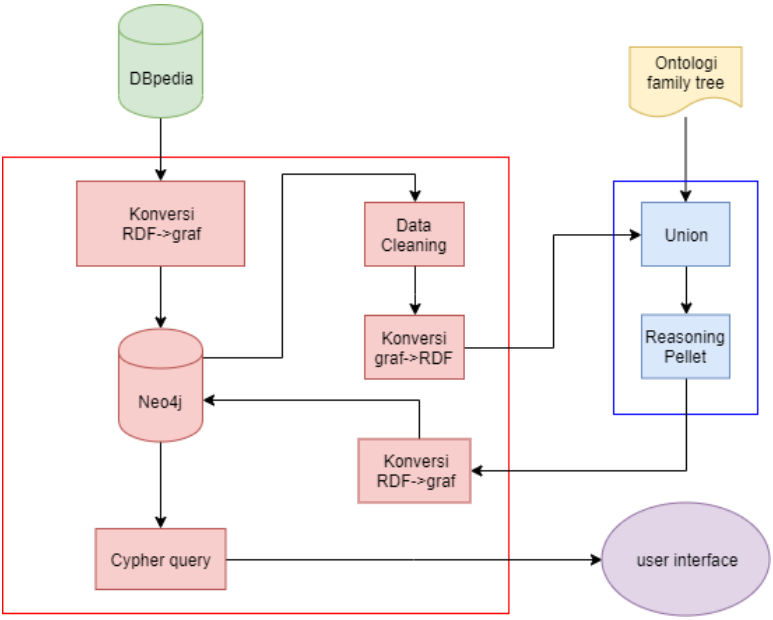
Namun dengan menggunakan model LPG pada basis data graf hubungan antara BUDI dan FANI dapat diketahui meskipun relasi tidak langsung keduanya harus melewati ANA sebagai penghubung.

Gambar 3.2 Penggambaran Contoh Kasus untuk *Graph Database*

Selain itu kelemahan lain jika contoh kasus ini dilakukan pada SPARQL dengan jumlah data yang besar akan terjadi pemborosan waktu dan memungkinkan terjadi *crash* pada *device* yang digunakan karena pengeksekusian kueri yang terlalu berat pada *device*. SPARQL mengeksekusi sebuah kueri dengan mencari setiap baris dari mulai dari subjek, predikat, kemudian objek yang mana hal itu tidak sama dengan pengeksekusian kueri *cypher* pada

basis data graf Neo4j. Penunjukan *node* dan penelusuran relasi pada graf dapat menghemat waktu pengeksekusian kueri.

3.2. Perancangan Arsitektur Sistem



Gambar 3.3 Perancangan Arsitektur Sistem

Dalam perancangannya seperti yang ditunjukan pada Gambar 3.3, arsitektur sistem yang dibuat pada tugas akhir ini terdiri dari beberapa bagian. Pada tahap pertama, data-data yang bersumber dari DBpedia diambil dan diekstrak untuk selanjutnya dimasukan pada Neo4J. Data yang semula berbentuk RDF kemudian diubah menjadi *graph database* pada Neo4j. Selanjutnya dilakukan pembersihan data menggunkan *cypher*, sehingga didapatkan data yang lebih ringan untuk mengurangi waktu pada saat melakukan *reasoning*.

Setelah dilakukan pembersihan data atau *data cleaning*, data akan diubah kembali menjadi RDF untuk dilakukan penggabungan

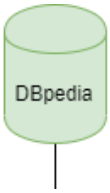
dengan ontologi yang telah dimodelkan dengan cara melakukan *reasoning* pada Apache Jena dengan bantuan *reasoner* Pallet. Data baru yang telah berisi hasil *reasoning* kemudian dikembalikan kembali menjadi *graph database* dengan cara yang sama seperti tahap awal. Terakhir, dilakukan pengkuerian pada data untuk membuktikan adanya hubungan antar tokoh.

3.3. Deskripsi Umum Sistem

Seperti yang sudah terpapar pada Gambar 3.3. Sistem memiliki sejumlah bagian yang mana masing-masing saling terhubung dan berjalan berurutan. Adapun penjelasan dari setiap bagian yaitu:

a) DBpedia

Situs ini menyediakan data-data yang dapat diunduh dan digunakan. Data dapat berbentuk JSON, RDF, N-triples, turtle dan lain-lain. Data tersebut nantinya diunduh dan diekstrasi pada basis data Neo4j. DBpedia ditunjukan seperti pada Gambar 3.4.



Gambar 3.4 Lapisan DBpedia pada Sistem

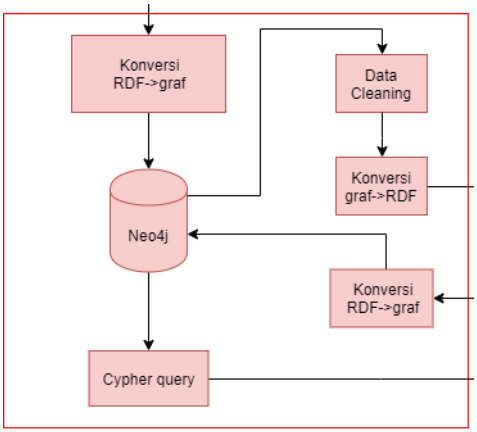
b) Neo4j

Neo4j menyediakan tempat untuk mengolah data menjadi bentuk graf yang mana graf tersebut dapat dikelola melalui Neo4j *browser* (versi antarmuka dari Neo4j *database*). Neo4j juga dapat dijadikan basis data untuk graf tersebut seperti ditunjukan pada Gambar 3.5.

a. Konversi RDF ke graf

Neo4j menyediakan *library* atau *plugin* Neosematics yang mana pada *plugin* tersebut

terdapat kueri yang dapat mengimpor data berbentuk RDF ke bentuk graf.



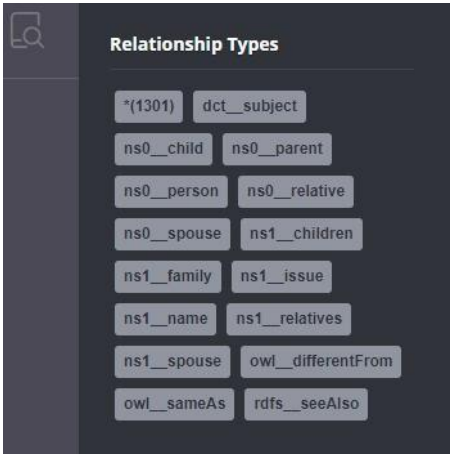
Gambar 3.5 Lapisan Neo4j pada Sistem

b. Data Cleaning
Neo4j memungkinkan melakukan *data cleaning* atau pembersihan data dengan melakukan kueri *cypher* agar besar data berkurang dan lebih ringan ketika pengolahan di tahap selanjutnya. Berikut adalah gambaran dari data sebelum dilakukan *data cleaning* yaitu Gambar 3.6, dan setelah dilakukan *data cleaning* yaitu pada Gambar 3.7.

c. Konversi graf ke RDF
Graf juga dapat dikembali menjadi RDF dengan bantuan sintaks yang dimiliki oleh Neosematics. Hal ini dilakukan untuk penggabungan data dengan Apache Jena di tahap selanjutnya.

d. Cypher query
Untuk melakukan pengkuerian setelah dilakukan *reasoning*, Neo4j menyediakan *cypher query* sebagai sintaks yang dapat memproses pengkuerian.

Gambar 3.6 Relasi Sebelum Dilakukan *Data Cleaning*



Gambar 3.7 Relasi Setelah Dilakukan *Data Cleaning*

c) Ontologi

Pada lapisan yang ditunjukan seperti pada Gambar 3.8, menggunakan ontologi *family tree ontology* yang mana berisi properti seperti *spouse*, *parent*, *child*, dan lain-lain, yang mana ontologi tersebut nantinya digabungkan dengan data dari

Neo4j untuk selanjutnya di *reasoning* pada Apache Jena. Referensi ontologi yang digunakan pada ontologi pada tugas akhir ini adalah milik Faiq dan Madis.

Gambar 3.8 Lapisan Ontologi pada Sistem

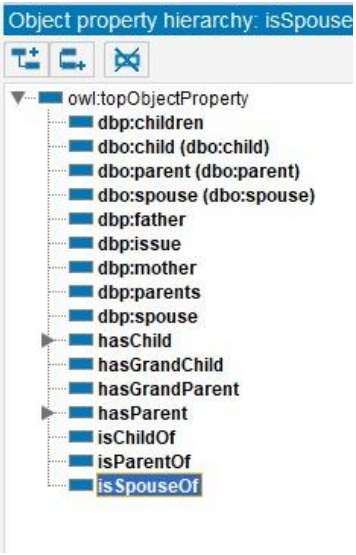
Seperti yang terlihat pada Gambar 3.9, *family tree ontology* yang digunakan oleh Madis dan Faiq memiliki beberapa *object property* diantaranya adalah [20], [21]:

- Dbo:children
- Dbo:child
- Dbo:parent
- Dbp:father
- Dbp:issue
- Dbp:spouse
- hasChild
- hasParent
- isChildOf
- isParentOf
- isSpouse

Kelas (*class*) yang digunakan oleh ontologi ini ada dua yaitu dbo:Agent, dbo:Person, seperti yang bisa dilihat pada Gambar 3.10.

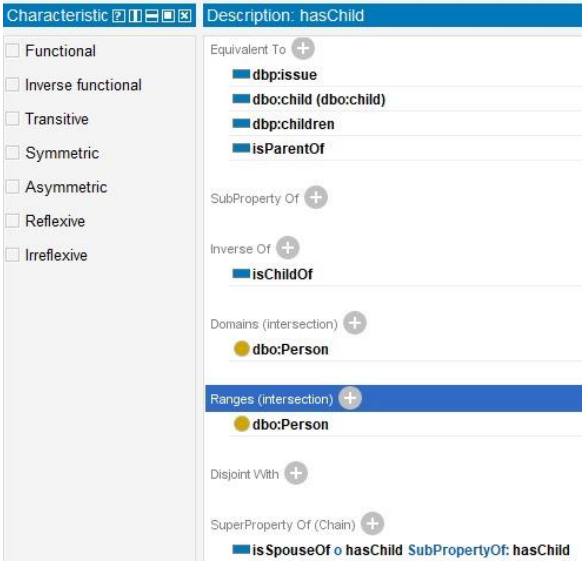
Untuk a*xiom* yang digunakan `dbo:children`, `dbo:child`, `dbp:issue`, `hasChild` dan `isParentOf` memiliki axiom *equivalent to* yang berarti properti tersebut equivalen atau sama. Properti lain yang juga memiliki *axiom equivalent to* adalah `dbo:parent`, `dbp:father`, `dbp:mother`, `isChildOf` dan `hasParent`. *Axiom* lain yang ada pada ontologi ini adalah inverse of yakni saling bertentangan antar properti anak (`dbo:children`, `dbo:child`, `dbp:issue`, `hasChild`, `isParentOf`) dan orang tua (`dbo:parent`, `dbp:father`, `dbp:mother`,

`isChildOf, hasParent`) seperti yang terlihat pada Gambar 3.11.

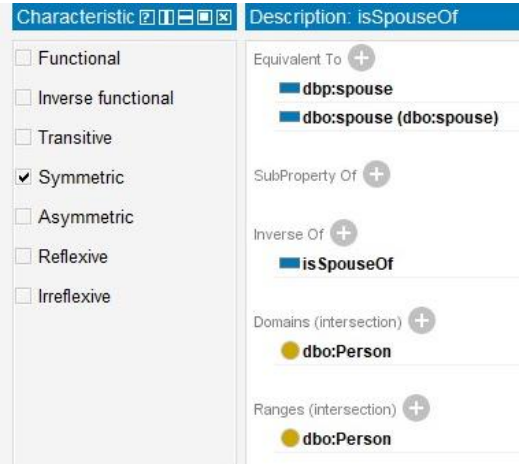


Gambar 3.9 *Object Property* pada *Family Tree Ontology* Madis dan Faiq

Gambar 3.10 Class pada *Family Tree Ontology* Madis dan Faiq



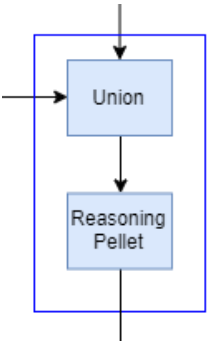
Gambar 3.11 Axiom Properti hasChild pada *Family Tree Ontology* Madis Faiq



Gambar 3.12 Axiom Properti isSpouseOf *Family Tree Ontology* Madis Faiq

Khusus untuk properti `isSpouseOf` seperti yang ditunjukan pada Gambar 3.12, properti ini menggunakan *axiom symmetric, equavalent to* dan *inverse of* dengan `dbp:spouse, dbo:spouse` dan sesama propertinya yaitu `isSpouseOf`.

d) Apache Jena



Gambar 3.13 Lapisan Aplikasi Java dengan Plugin Apache Jena pada Sistem

Plugin dari Apache Jena dimasukan pada sebuah *script* berbasis Java.

a. Union
   Pada *script* ini seperti yang ditunjukkan pada Gambar 3.13, dilakukan penggabungan data antara ontologi dan data dari Neo4j yang telah diproses sebelumnya.
b. *Reasoning*
   Reasoning dilakukan untuk mendapatkan *inferred fact* atau fakta-fakta baru dan melengkapi relasi yang sebelumnya tidak lengkap.

e) User Interface

User interface pada sistem merupakan tampilan visual yang disediakan oleh Neo4J. Bagian user interface digambarkan seperti pada Gambar 3.14.

Gambar 3.14 Lapisan *User Interface* pada Sistem

3.4. Analisis Data

Langkah awal untuk menyelesaikan masalah pada pemetaan atau permodelan skema penyimpanan relasi pada silsilah keluarga suatu tokoh perlu dilakukan analisis data yang akan digunakan. Data yang diambil pada tugas akhir ini menggunakan domain *actor*, yang mana memiliki beberapa ruang lingkup atau *class* seperti *group*, *organization*, *person*, *agent*, dan lain-lain.

Gambar 3.15 Kelas-kelas atau Domain pada DBpedia

Pada tugas akhir ini, data diambil dari ruang lingkup *person* – seperti yang ditunjukan pada Gambar 3.15, yang memiliki bagian data biografi dari suatu tokoh. Keluarga kerajaan Inggris digunakan sebagai dataset dikarenakan ketersediaan datanya yang banyak dan juga adanya keterkaitan antar individu yang kuat. DBbedia Inggris digunakan sebagai sumber data dipilih dengan pertimbangan adanya properti (*property*) yang lebih lengkap dan

lebih terstuktur sehingga lebih mudah dalam pengolahan data yang akan dilakukan pada tahap selanjutnya.

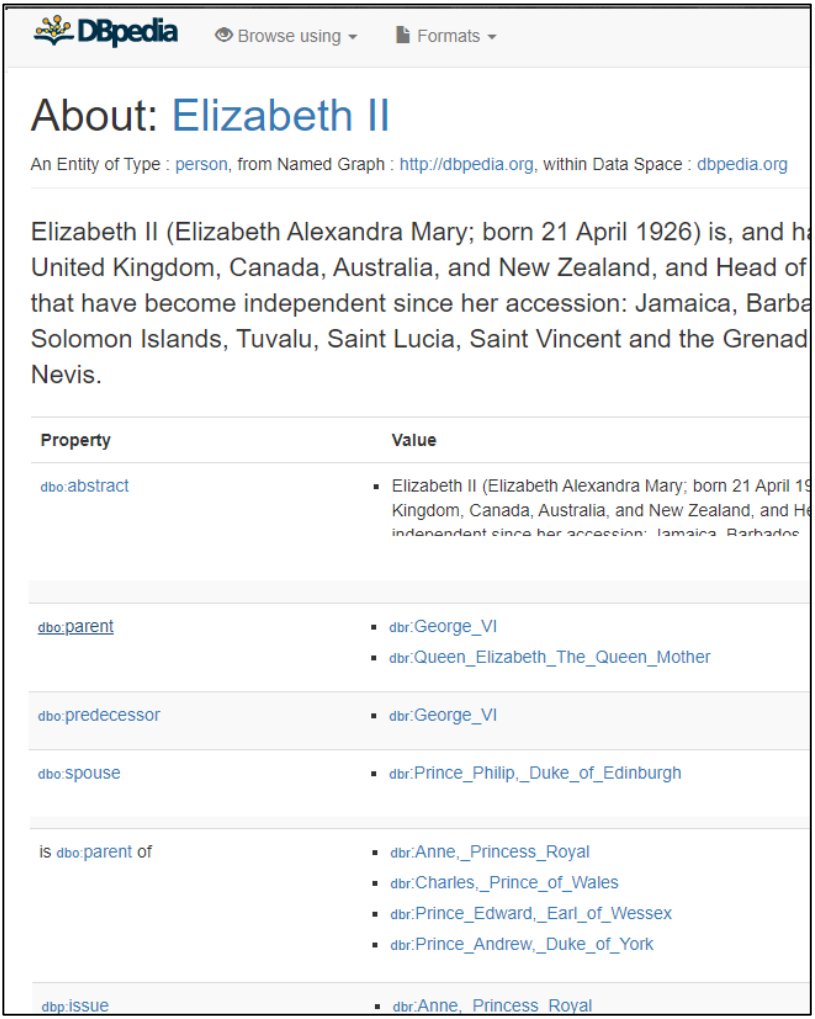
Pada DBpedia, setiap laman data atau *person* memiliki sejumlah properti yang dapat digunakan sebagai indikator dalam menganalisis keterkaitan antar individu seperti *child*, *parent*, *spouse* dan *children*. Beberapa properti yang tersedia pada DBpedia dapat dilihat pada Gambar 3.16 yaitu milik Elizabeth II (http://dbpedia.org/page/Elizabeth_II) .

DBpedia juga memiliki situs alternatif lain atau *resource* lain yang dapat digunakan sebagai pelengkap dari data yang akan diekstrak seperti DBpedia Indonesia, DBpedia Jepang, DBpedia Eropa dan lain-lain yang mana situs atau laman tersebut terhubung melalui properti `owl:sameAs.` Adapun laman dari DBpedia yang tidak memiliki properti yang lengkap sehingga ketika dilakukan analisis pada keterkaitan antar tokohnya tidak bisa terhubung, sebab ketersediaan properti pada salah satu pihak individu adalah syarat suatu individu dapat memiliki relasi dengan individu lain.

Hal lain yang dapat menjadi kendala akibat tidak lengkapnya properti sebagai contohnya adalah ketidaktersediaannya properti nama (`foaf:name, dbpprop-id:name, dbp:name`), ketidaktersediaan properti tersebut akan mempersulit pada saat melakukan kueri berdasarkan properti tersebut dan proses untuk mendapatkan *inferred fact* dari tiap individu akan menurun pada saat *reasoning* terjadi.

Pada paragraf sebelumnya telah dijelaskan bahwa DBpedia memiliki resource bermacam-macam yang terhubung melalui properti `owl:sameAs`. *Resource* sendiri adalah laman-laman alternatif yang memiliki kriteria hampir sama dengan data yang diambil pertama kali tetapi isi didalamnya kemungkinan berbeda dengan properti yang berbeda-beda pula. Kelebihan dari resource meman dapat melengkapi data. Namun, terdapat kekurangan lain yang terbentuk yaitu ketika melakukan kueri dari data DBpedia dapat terjadi redudansi pada data karena adanya *resource* yang bermacam-macam. Sebagai contoh ketika kita ingin mencari data oleh tokoh Elizabeth II seperti pada Gambar 3.27, maka akan

muncul banyak data dengan *resource* berbeda-beda dari tokoh Elizabeth tersebut.



Gambar 3.16 Laman DBpedia Elizabeth II

Pada tugas akhir ini diasumsikan bahwa seorang tokoh hanya akan menggunakan satu resource saja, dan mengabaikan resource lain yang mungkin muncul ketika pemrosesan data termasuk pengkuerian.

Gambar 3.17 Contoh Properti *owl:sameAs* pada Laman Elizabeth II

3.5. Ekstraksi Data sebagai Basis Data Graf pada Neo4j

Setelah dilakukan analisis, data-data dari individu yang ingin diproses pada tahap selanjutnya diunggah dengan format RDF pada lama DBpedia. Proses selanjutnya yaitu menyimpan atau mengimpor data tersebut ke dalam Neo4j sehingga dapat data yang semula berbentuk RDF diubah menjadi *graph database*. Pengimporan dilakukan dengan bantuan *library* yang tersedia pada Neo4j yaitu neosematics, dengan menggunakan sintaks *cypher* seperti Kode Sumber 3.1 yang diperlukan untuk mengimpor data RDF ke Neo4j. *Graph database* dapat terbentuk tanpa menghilangkan properti atau data apapun.

Sintaks cypher yang digunakan untuk mengimpor RDF ke dalam Neo4J perlu memiliki beberapa argumen inputan agar data tersimpan sempurna yaitu sebagai berikut [22]:

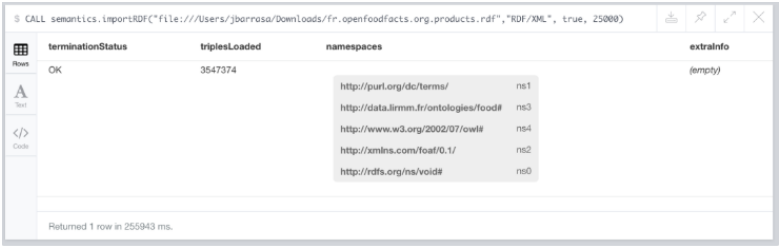
1) Memiliki url data yang diimpor (bisa berupa URL ataupun lokasi data pada penyimpanan *offline*).

2) Memiliki jenis serialisasi atau bentuk data dari RDF seperti JSON-LD, Turtle, RDF/XML, N-Triples dan TriG. Serialisasi yang digunakan pada tugas akhir ini adalah RDF/XML.
3) Periode pada saat komit. Kelipatan setelah komit dijalankan.

```
4)  CREATE INDEX ON :Resource(uri);
5)  CALL semantics.importRDF("http://fr.openfoodfacts.o
    rg/data/fr.openfoodfacts.org.products.rdf","RDF/XML
    ",{commitSize: 500})
```

Kode Sumber 3.1 *Cypher* untuk Mengimpor RDF ke Neo4j

Contoh argumen dapat dilihat pada Kode Sumber 3.1, `CREATE INDEX ON :Resource(uri)` adalah sintak *cypher* untuk membuat indeks baru dalam mapping data sumber dengan (S,P,O) tanpa literal. Sedangkan pada kode baris kedua merupak sintaks *cypher* untuk memuat dokumen yang ingin dimasukan kedalam Neo4j dengan bantuan *plugin* Neosematics, dengan hasil seperti Gambar 3.18.



Gambar 3.18 Hasil *Cypher* untuk Mengimpor RDF ke Neo4j

Pada bagian *namespace prefix*, terdapat perubahan yang mana *namespace prefix* tersebut berpengaruh pada penamaan label pada Neo4j untuk *node* dan relasinya. Keterangan label dapat dilihat pada Tabel 3.2.

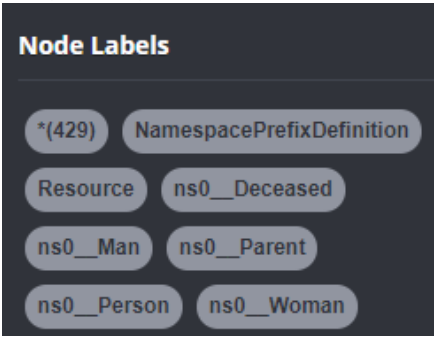
Tabel 3.2 Perubahan *Namespace Prefix* pada Neo4j

Prefix	URL	Description
Ns0	http://dbpedia.org/ontology/	Ontologi DBpedia
Ns1	http://dbpedia.org/property/	Properti dari raw *infobox extraction*
Ns2	http://dbpedia.org/class/yago/	*Yet Another Great Ontology*
Ns3	http://www.wikipedia.org/entity	Halaman Wikipedia
Ns4	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#	*DOLCE+DnS Ultralite ontology*
Ns5	http://umbel.org/umbel/rc/	*Upper Mapping and Binding Exchange Layer*
Ns6	http://xmlns.com/foaf/0.1/	*Friend of a friend*
Ns7	http://w3.org/ns/prov#	PROV *family of document*
Ns8	http://purl.org/linguistics/gold/	*GOLD ontology*
Ns9	http://www.w3.org/2003/01/geo/wgs84_pos#	*WGS84 Geo Positioning*
Ns10	http://dbpedia.org/ontology/Person/	Ontologi DBpedia dengan *class Person*
skos	http://www.w3.org/2004/02/skos/core#	*Simple Knowledge Organization System*
dc	http://purl.org/dc/elements/1.1/	*Dublin core*
sh	http://w3.org/ns/shacl#	*Shapes constaint langauge* (SHACL)

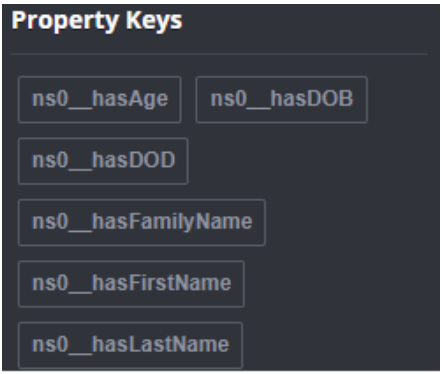
Prefix	URL	Description
sch	http://schema.org/	*Schema*
owl	http://www.w3.org/2002/07/owl#	*W3C web ontology language*
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	*Standart W3c RDF vocabulary*
rdfs	http://www.w3.org/2000/01/rdf-schema#	*Extention of the basic RDF vacabulary*

3.6. *Reasoning* Data menggunakan Neo4j

Telah diketahui bahwa Neo4j dapat digunakan untuk memuat dan menulis data dengan format RDF. Sedangkan untuk melakukan *reasoning* pada RDF atau OWL, *reasoner* yang dapat memprosesnya ialah *reasoner* berjenis *fully-fledged triplestore reasoning engines* atau mesin yang memang khusus digunakan untuk *reasoning* seperti Pellet, HermiT, dan Racer. Namun, menurut pernyataan Thorsten Liebig [25], Neo4j juga dapat digunakan sebagai teknologi *reasoning* yang unik yang mana dapat menghasilkan mesin penalaran yang sangat ekspresif dan sangat kompetitif bagi RDF, RDFS, dan OWL 2 RL.



Gambar 3.19 Perubahan *Class* menjadi *Node Labels*



Gambar 3.20 Perubahan *Data Type* menjadi *Property Keys*

Pada Neo4j, *class* dan *dataType*, atau subjek dan objek dari sebuah *triples* diubah menjadi *node label* dan *property keys* seperti Gambar 3.19 dan Gambar 3.20. Sedangkan untuk *objectProperty* atau predikat dari *triples* pada RDF diubah menjadi relasi atau *relationship type* pada Gambar 3.21.

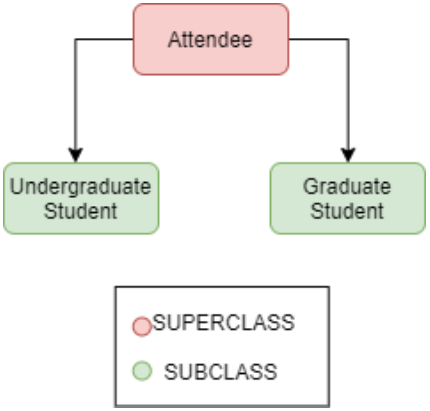
Konsep *reasoning* yang dikemukan Thorsten Liebig memiliki ilustarsi seperti pada Gambar 3.22. Attendee merupakan sebuah *superclass* dengan UnderGraduateStudent dan GraduateStudent sebagai *subclass*-nya. Pada konsep ini Thorsten memisalkan Attendee adalah *superclass* yang belum dinyatakan pada RDF yang telah dimuat pada Neo4j, RDF tersebut hanya ada *subclass* dari Attendee yaitu UnderGraduateStudent dan GraduateStudent. Tahap *reasoning* yang dimaksud ialah menyatakan Attendee atau menambahkan Attendee sebagai *superclass* dari UnderGraduateStudent dan GraduateStudent. Hal tersebut dapat dilakukan dengan menggunakan sintak *cyper* seperti Kode Sumber 3.2. Selain menggunakan *class* untuk *reasoning*, konsep ini juga dapat diterapkan jika *node labels* memiliki *data type* yang sama sehingga dapat dikelompokan atau dilakukan penghirarkian pada *node labels* yang sama.

```
1.  match (x)-[:takesCourse]-
    >() where x:Student or x:UndergraduateStudent
```

```
2.  set x:Attendee
```

Kode Sumber 3.2 *Cypher* untuk *Reasoning SuperClass*

Gambar 3.21 Perubahan *Object Property* menjadi *Relationship Types*



Gambar 3.22 Ilustrasi *Reasoning* oleh Thorsten Liebig

Sedangkan *reasoning* yang diinginkan pada penelitian ini adalah *reasoning* untuk mendapatkan *inferred fact* menggunakan *relationship types* atau *object property* yang mana hal tersebut masih belum dapat dipecahkan oleh Neo4j karena konsep *reasoning* yang dipaparkan oleh Thorsten merupakan konsep untuk *reasoning* dengan menambahkan hirarki sedangkan *object property* tidak hanya berkonsep hirarki saja namun bisa memiliki *rules* lain seperti *inversOf, TransitiveProperty, SymmetricPropery, FunctionProperty, InverseFunctionalProperty* dan *restriction.*

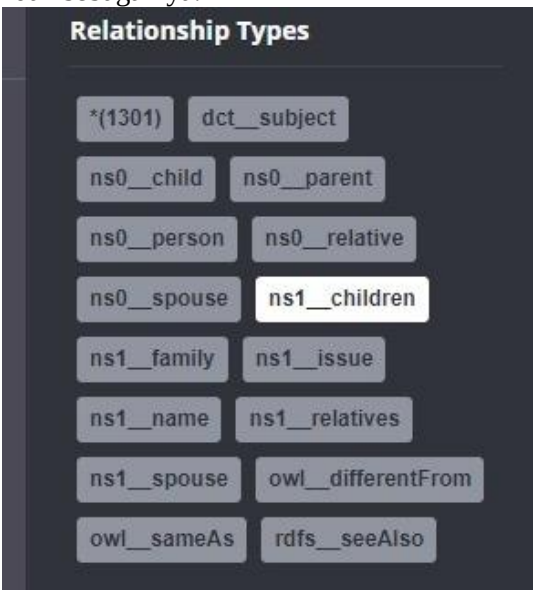
Oleh sebab itu, pada subbab ini *reasoning* pada Neo4j tidak dilakukan atau diimplementasikan setelah mempertimbangkan fakta seperti penjelasan di atas. Sebagai gantinya, implementasi *reasoning* akan menggunakan Apache Jena dengan *reasoner* Pellet yang akan dijelaskan pada subbab 3.8.

3.7. Pembersihan Data pada Basis Data Graf

Gambar 3.23 Contoh Relationship yang Sebelum di *Data Cleaning*

Pembersihan data dilakukan untuk mengurangi besarnya data yang digunakan. Pembersihan data hanya dilakukan untuk

menghapus relasi yang tidak diperlukan pada proses selanjutnya, sebagai contoh adalah relasi seperti pada Gambar 3.23 yaitu ns0_appointer, ns0_birthPlace, ns0_breader, ns0_knownFor, ns0_leader dan sebagainya.



Gambar 3.24 Contoh Relationship yang Setelah di *Data Cleaning*

Sedangkan properti yang digunakan sebagai kunci adanya relasi antar tokoh seperti, *child, spouse, children, issue,* dan *parent* tidak akan dihapus. Pembersihan dilakukan dengan menggunakan sintaks *cypher* yang mana akan menghapus relasi pada tokoh tanpa menghapus label (subjek dan objek jika pada *triplestore*) pada tokoh tersebut.

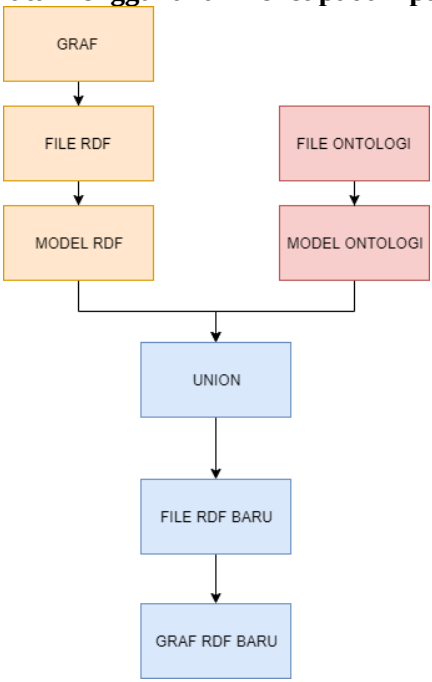
Contoh sintaks *cypher* yang digunakan adalah seperti pada Kode Sumber 3.3. `MATCH` merupakan klausa yang digunakan untuk menentukan pola. Yang mana pola tersebut akan digunakan untuk untuk mendapatkan data dari basis data pada Neo4j. `(n)` menunjukan *node* yang ditunjuk sedangkan `[r:royal]`

merupakan relasi yang ingin dihapus, sedangkan variable **()** menunjuk semua relasi yang berhubungan dengan n dan **DELETE r** adalah klausa untuk menyatakan hapus relasi r. Setelah dijalankan, hasil relasi yang baru seperti pada Gambar 3.24.

```
1.  MATCH (n)-[r:royal]->()
2.  DELETE r;
```

Kode Sumber 3.3 Contoh *Cypher* untuk *Data Cleaning*

3.8. *Reasoning* Data menggunakan Pellet pada Apache Jena



Gambar 3.25 *Flowchat* Perubahan Format Data

Terdapat aliran perubahan data yang terjadi sebelum dan sesudah terjadinya *reasoning.* Seperti yang terlihat pada Gambar 3.25. Data yang sudah ditetapkan bersih kemudian diubah kembali

menggunakan sintaks yang disediakan Neosematics (masih dengan catatan tidak menghilangkan data yang ada pada proses yang telah dilakukan di dalam Neo4j). Sintaks yang digunakan terlihat seperti pada Kode Sumber 3.4. *POST request* dipakai untuk mengambil *payload* dari JSON *map* dengan setidaknya satu *cypher* sebagai nilai kueri yang ingin mengubah kembali objek-objek pada graf seperti *nodes* dengan properti yang dimilikinya serta relasi untuk kemudian diubah menjadi *triplestore* dalam serialisasi RDF.

```
1.  :POST /rdf/cypheronrdf { "cypher":"MATCH (n:Resourc
    e)-[r]-(m) RETURN * " , "format" : "RDF/XML"}
```

Kode Sumber 3.4 *Cypher* untuk Mengeksport data Menjadi RDF

Data yang kembali menjadi RDF tersebut kemudian diubah menjadi model pada *script* Java yang dibuat dengan sintaks seperti Kode Sumber 3.5.

```
1.  Model Instances = FileManager.get().loadModel(READ_
    RDF);
2.  Instances.read(READ_RDF,"RDF/XML");
3.
4.  Model famtree = FileManager.get().loadModel(ONTOLOG
    I);
```

Kode Sumber 3.5 Sintaks untuk Mendeklarasikan Model dengan Apache Jena

Filemanager.get().loadModel merupakan fungsi yang disediakan oleh Apache Jena untuk dapat mengakses dokumen lokal yang memiliki format RDF. Ada dua model yang digunakan dalam proses ini, yaitu model RDF yang berasal dari Neo4j berisi data tokoh dari DBpedia, dan yang kedua adalah model ontologi berformat OWL yang berisi tentang *rules* serta *object property* yang digunakan sebagai *reasoning* nantinya.

Ontologi yang digunakan merupakan ontologi milik Madis dan Faiq yaitu *Family tree ontology*. Pada ontologi tersebut terdapat properti *hasParent, hasChild, hasChildren, isParentOf, isChildOf, isSpouseOf,* dan properti yang juga ada di DBpedia yaitu *child, parent, parents, issue, children, father, mother,* dan *spouse*. Properti-properti tersebut dipilih karena beberapa properti tersebut merupakan properti yang dapat digunakan untuk membentuk silsilah atau pohon keluarga. Adapun tambahan properti yang digunakan yaitu *hasGrandParent, hasGrandChild, hasGrandGrandParent*, dan *hasGrandGrandChild*. Keenam properti tersebut ditambahkan sebagai pembanding saat dilakukan *reasoning*.

Alasan mengapa properti seperti hasSibling, hasUncle, hasAunt, hasNiece dan hasNephew tidak digunakan adalah karena adanya penalaran yang salah atau kurang tepat ketika ingin mendapatkan *inferred fact* dari hasSibling, dimana *reasoning* akan menunjuk seorang individu sebagai saudara atau hasSibling dari individu itu sendiri (menunjuk dirinya sendiri). Sedangkan properti seperti hasUncle, hasAunt, hasNiece, dan hasNephew selain dia perlu mengikutsertakan hasSibling pada penalarannya ketidakadaan *data property* atau *class gender* mempersulit munculnya *inferred fact* baru pada saat *reasoning* sehingga berpotensi munculnya salah penafsiran pada hasil *reasoning.*

Ada beberapa ontologi yang digunakan dalam perbandingan *reasoning*, yaitu ontologi menggunakan *chain property*, ontologi menggunakan SWRL saja, dan ontologi tanpa adanya *chain property* dan SWRL. Macam-macam ontologi tersebut digunakan untuk mengukur kecepatan Apache Jena ketika melakukan *reasoning* terhadap model tokoh yang diolah pada Neo4j sebelumnya.

Setelah dua data tersebut diubah menjadi model, dua model tersebut kemudian digabungkan (union). Model yang terbentuk dari gabungan dua model sebelumnya di *reasoning. Reasoning* dilakukan menggunakan bantuan *library* Pellet yang mana dapat memunculkan fakta-fakta baru untuk melengkapi data awal yang

berasal dari Neo4j. Sintaks yang digunakan untuk mendeklarasi *library* Pallet adalah Kode Sumber 3.6.

```
1. Reasoner pelletReasoner = PelletReasonerFactory.the
   Instance().create();
2. InfModel reasonedModel = ModelFactory.createInfMode
   l(pelletReasoner,union);
```

Kode Sumber 3.6 Penggabungan dan Penalaran pada Pellet

Gambar 3.26 *Namespace Prefix* untuk Properti Hasil *Inferred Facts*

Setelah dijalankan penalaran selesai, fakta-fakta baru akan disimpan pada dokumen baru berbentuk RDF. Selanjutnya, dokumen baru tersebut diimpor kembali isinya ke dalam Neo4j untuk dilakukan kueri dengan sintaks yang sama seperti pada Kode Sumber 3.1.

Pada tahap pengubahan kembali data RDF ke Neo4j menjadi format Graf, akan muncul relasi baru yang didapat dari adanya *reasoning* yang dilakukan. Relasi seperti hasParent, hasChild, isChildOf, isParentOf, isSpouseOf akan mendapatkan *namespace prefix* yang berbeda seperti yang ditunjukan pada Gambar 3.26. Perlu diingat bahwa penamaan pada *namespace prefix* dilakukan secara otomatis pada Neo4j seperti pada Tabel 3.3.

Tabel 3.3 Daftar *Namespace Prefix* Hasil *Reasoning* pada Neo4j

Prefix	URL	Description
Ns0	http://xmlns.com/foaf/0.1/	*Friend of a friend*
Ns1	http://dbpedia.org/ontology/	Ontologi DBpedia
Ns2	http://dbpedia.org/class/yago/	*Yet Another Great Ontology*
Ns3	http://www.wikipedia.org/entity	Halaman Wikipedia
Ns4	http://umbel.org/umbel/rc/	*Upper Mapping and Binding Exchange Layer*
Ns5	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#	*DOLCE+DnS Ultralite ontology*
Ns6	http://dbpedia.org/property/	Properti dari raw *infobox extraction*
Ns7	http://www.co-ode.org/roberts/family-tree.owl#	Robert *family tree ontology*
Ns8	http://purl.org/linguistics/gold/	*GOLD ontology*
Ns9	http://dbpedia.org/ontology/Person/	Ontologi DBpedia dengan *class Person*
skos	http://www.w3.org/2004/02/skos/core#	*Simple Knowledge Organization System*
dc	http://purl.org/dc/elements/1.1/	*Dublin core*
sh	http://w3.org/ns/shacl#	*Shapes constaint langauge* (SHACL)
sch	http://schema.org/	*Schema*

Prefix	URL	Description
owl	http://www.w3.org/2002/07/owl#	*W3C web ontology language*
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	*Standart W3c RDF vocabulary*
Rdfs	http://www.w3.org/2000/01/rdf-schema#	*Extention of the basic RDF vacabulary*

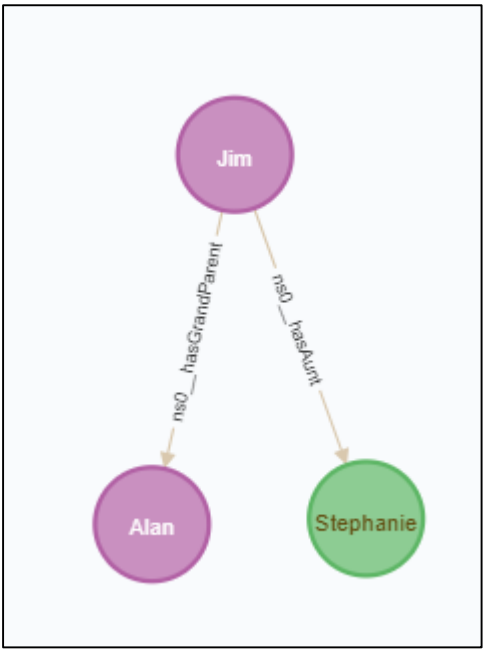
Setelah memuat data baru selesai, tidak dilakukan lagi tahap pembersihan data atau proses *data cleaning*. Karena semua relasi yang ada merupakan relasi kunci yang akan digunakan untuk membentuk pohon keluarga. Apabila data yang telah di *reasoning* tersebut dilakukan *reasoning* kembali, hasil pemuatan data pada Neo4j tidak akan berubah karena basis data graf akan menyatakan semua *triple* dengan properti yang sama sebagai satu *node* dan relasi yang sama pula.

3.9. Penguerian Data

Kueri dilakukan untuk memecahkan permasalahan lain yang dibahas pada tugas akhir ini, yaitu mendapatkan fakta antara dua tokoh yang tidak memiliki relasi satu sama lain. Fakta ini dapat didapatkan jika sedikit-dikitnya ada satu individu yang memiliki hubungan diantara keduanya. Sebagai contoh kasus pada Gambar 3.27.

```
1. MATCH (m:ns0__Person {ns0__hasFirstName:'Stephanie'
   }),(n:ns0__Person {ns0__hasFirstName:'Alan'}),
2. p = shortestPath((m)-[*..30]-(n))
3. RETURN p
```

Kode Sumber 3.7 Menampilkan Relasi antar Dua Tokoh

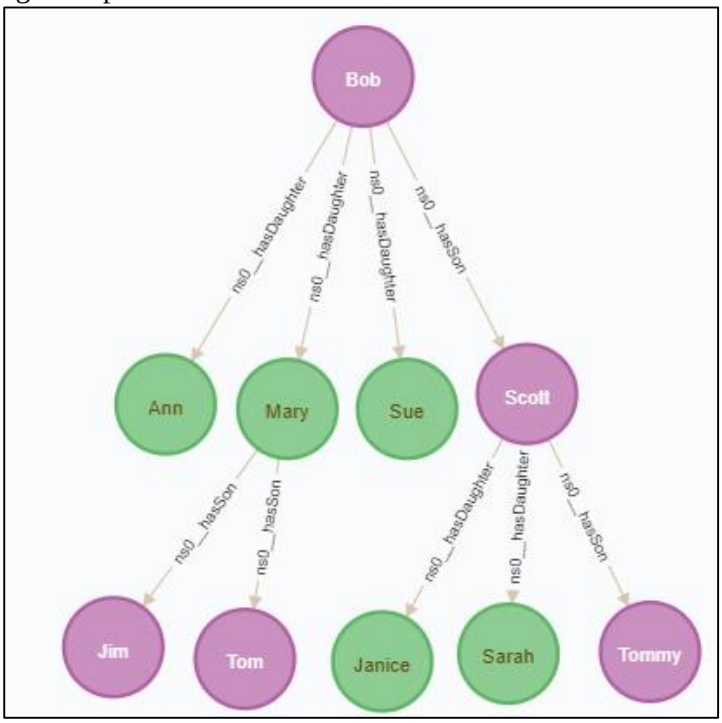


Gambar 3.27 Contoh Kasus Menampilkan Relasi antar Dua Tokoh

Graph database dapat menyelesaikan kasus tersebut dengan catatan A dan C memiliki relasi dengan B meskipun A dan C tidak memiliki relasi apapun yang dapat menghubungkan keduanya. Kueri yang digunakan adalah kueri dari Kode Sumber 3.7. Seperti yang sudah dijelaskan sebelumnya, **`MATCH`** adalah sebuah klausa untuk menentukan pola atau aksi yang akan dilakukan. **`(m:ns0__Person {ns0__hasFirstName:'Stephanie'})`** menjelaskan bahwa node **`m`** merupakan bagian dari domain atau pada Neo4j disebut label **`ns0_Person`** dengan properti **`ns0_hasFirstName 'Stephanie'`**, Dan pada node **`n`** dengan penjelasan sama seperti node m, memiliki properti **`ns0_hasFirstName 'Alan'`**. Selanjutnya dengan variable **`p`**, kedua node tersebut dicari relasinya menggunakan fungsi

shortestPath yang disediakan oleh Neo4j. Dan kemudian di **RETURN p**, untuk menampilkan relasinya.

Hasil dari kueri pertama yaitu Stephanie memiliki hubungan atau relasi dengan Alan melalui Jim yang mana Jim memiliki relasi *hasGrandParent* dengan Alan dan Jim memiliki relasi hasAunt dengan Stephanie.



Gambar 3.28 Contoh kasus Menampilkan Pohon Keluarga Seorang Tokoh

Kueri kedua yang dilakukan adalah mendapatkan beberapa generasi dari suatu individu seperti pada Gambar 3.28. Neo4j dapat mengeksekusi kueri ini dengan batasan tertentu. Dengan Kode Sumber 3.8, kueri tersebut dapat menampilkan sebuah graph dengan satu individu sebagai pusat. Contoh individu nya ialah Bob

yang memiliki empat anak bernama Ann, Mary, Sue, dan Scott, dengan Mary dan Scott yang sudah memiliki keturunan. Mary memiliki dua anak bernama Tom dan Jim, sementara Scott memiliki tiga anak yaitu Janice, sarah, dan Tommy

```
1.  MATCH p=(a:ns0__Person)-
    [:ns0__hasSon|ns0__hasDaughter*]->(allchildren)
2.  WHERE 'Bob' IN a.`ns0__hasFirstName`
3.  RETURN p
4.
```

Kode Sumber 3.8 Menampilkan Pohon Keluarga Seorang Tokoh

BAB IV
IMPLEMENTASI SISTEM

Bab ini membahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijelaskan pada bab sebelumnya. Proses implementasi dari setiap tahap pada perangkat lunak yang dibuat akan diuraikan selengkapnya pada bab ini. Implementasi perangkat lunak yang dibuat menggunakan bahasa pemrograman HTML, PHP, dan JavaScript, yang mana pada implementasinya akan melewati tahapan dengan bantuan aplikasi lain seperti Neo4j dan aplikasi berbasis Java dengan *library* Apache Jena dan Pellet.

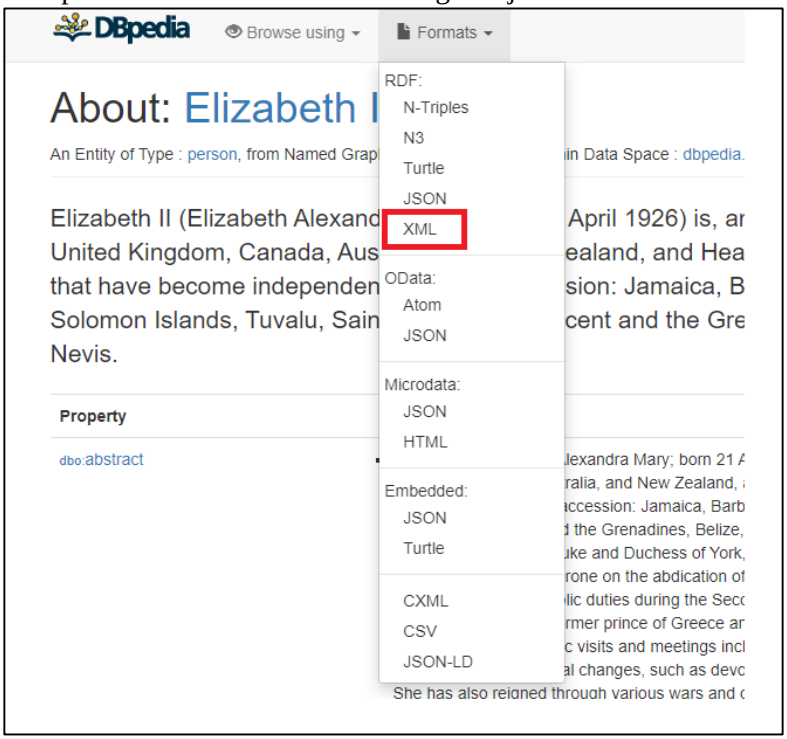
4.1. Implementasi Ekstaksi Data sebagai Basis Data Graf pada Neo4j

Pada subbab ini akan dipaparkan secara rinci mengenai implementasi dari tahapan pertama dalam pembuatan sistem yaitu implementasi dari pengekstasian data berasal dari DBpedia. Adapun tahapan ekstrasi data yang dibagi menjadi dua bagian, yaitu pengunduhan data pada DBpedia dan pengimporan data RDF ke Neo4j.

4.1.1. Pengunduhan Data pada DBpedia

Pengunduhan dilakukan secara manual, yaitu dengan cara mencari laman tokoh yang ingin dijadikan dataset. Setelah itu menekan *drop-down* "Formats" yang tersedia pada laman DBpedia tersebut, selanjutnya pilih salah satu format yaitu RDF-XML, seperti pada Gambar 4.1. *Property* yang ada pada laman DBpedia akan berubah menjadi predikat, sedangkan *value* dapat berubah menjadi subjek atau objek seperti pada Kode Sumber 4.1 dengan bentuk awal seperti Gambar 4.2. Pada kode sumber tersebut "`<rdf:Description rdf:about="http://dbpedia.org/resource/Prince_Philip,_Duke_of_Edinburgh`" adalah IRI dari Prince Philip, Duke of Edinburgh sebagai subjek, "`<dbo:spouse>`" merupakan predikat dari *property spouse*. Sedangkan

“rdf:resource="http://dbpedia.org/resource/Elizabeth_II"” merupakan IRI dari Elizabeth II sebagai objek.



Gambar 4.1 Mengunduh RDF-XML pada Laman DBpedia milik Elizabeth

```
1.  <rdf:Description rdf:about="http://dbpedia.org/reso
    urce/Prince_Philip,_Duke_of_Edinburgh">
2.    <dbo:spouse rdf:resource="http://dbpedia.org/reso
    urce/Elizabeth_II" />
3.  </rdf:Description>
```

Kode Sumber 4.1 Baris RDF dengan Properti Spouse milik Elizabeth II

Pengunduhan untuk tokoh selanjutnya didapat dari *value* yang memiliki properti *parent*, *childen*, *child*, *issue*, dan *spouse*. Yang mana *property-property* tersebut seperti dbo:parent, dbo:spouse, is dbo:spouse of, is dbo:parent of, dbo:child, dan sebagainya.

Gambar 4.2 Kolom Properti Spouse milik Elizabeth II

4.1.2. Pengimporan Data RDF ke Neo4j

Data yang telah diunduh ditahap sebelumnya selanjutnya satu-persatu dimasukan ke dalam Neo4j sehingga didapatkan basis data graf. Data yang semula *triplestore* diubah menjadi *node-node* berlabel yang memiliki relasi dan membentuk sebuah graf. Sintaks *cypher* ini bekerja dengan bantuan *library* Neosematics yang mana menyediakan sintaks yang dapat mengubah RDF ke bentuk graf tanpa menghilangkan data-data didalamnya, sehingga data tetap sama seperti sebelum diubah menjadi ekstensi lain yaitu bentuk graf. Pengimporan ini juga dapat menyatukan dataset yang sebelumnya hanya perindividu menjadi terkelompok menjadi satu label dan satu relationship seperti. Sintaks pengimporan data dapat dilihat seperti pada Kode Sumber 4.2.

```
1. CREATE INDEX ON :Resource(uri);
2.
3. CALL semantics.importRDF("file:///C:/Users/MI/Downl
   oads/tokoh3/Elizabeth_II.rdf","RDF/XML",{commitSize
   : 10});
4.
5. CALL semantics.importRDF("file:///C:/Users/MI/Downl
   oads/tokoh3/James,_Viscount_Severn.rdf","RDF/XML",{
   commitSize: 10});
6.
7. CALL semantics.importRDF("file:///C:/Users/MI/Downl
   oads/tokoh3/Lady_Louise_Windsor.rdf","RDF/XML",{com
   mitSize: 10});
```

Kode Sumber 4.2 *Cypher* untuk Mengimport Data RDF menjadi Basis Data Graf

Data awal yang berbentuk RDF seperti Kode Sumber 4.3. Setiap baris dari dokumen pada RDF akan diubah menjadi basis data graf. *Object property* yang merupakan predikat akan diubah menjadi *relationship keys*. Sedangkan subjek akan diubah menjadi *node labels*, dan *data type* akan diubah menjadi *property keys*. Namespace prefix akan secara otomatis diubah dan dikelompokan seperti pada Tabel 4.1.

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <rdf:RDF
3.      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#"
4.      xmlns:rdfs="http://www.w3.org/2000/01/rdf-
    schema#"
5.      xmlns:owl="http://www.w3.org/2002/07/owl#"
6.      xmlns:dbo="http://dbpedia.org/ontology/"
7.      xmlns:dct="http://purl.org/dc/terms/"
8.      xmlns:foaf="http://xmlns.com/foaf/0.1/"
9.      xmlns:dbp="http://dbpedia.org/property/"
10.     xmlns:prov="http://www.w3.org/ns/prov#" >
11.    <rdf:Description rdf:about="http://dbpedia.org/re
    source/British_Armed_Forces">
12.     <dbp:commanderInChief rdf:resource="http://dbpe
    dia.org/resource/Elizabeth_II" />
13.    </rdf:Description>
14.    <rdf:Description rdf:about="http://dbpedia.org/re
    source/Eric_Williams">
15.     <dbo:monarch rdf:resource="http://dbpedia.org/r
    esource/Elizabeth_II" />
16.    </rdf:Description>
17.    <rdf:Description rdf:about="http://dbpedia.org/re
    source/George_VI">
18.     <dbo:successor rdf:resource="http://dbpedia.org
    /resource/Elizabeth_II" />
19.     <dbp:issue rdf:resource="http://dbpedia.org/res
    ource/Elizabeth_II" />
20.    </rdf:Description>
```

```
21.   <rdf:Description rdf:about="http://dbpedia.org/re
    source/James_Callaghan">
22.     <dbo:monarch rdf:resource="http://dbpedia.org/r
    esource/Elizabeth_II" />
23.   </rdf:Description>
24.   <rdf:Description rdf:about="http://dbpedia.org/re
    source/John_Gorton">
25.     <dbo:monarch rdf:resource="http://dbpedia.org/r
    esource/Elizabeth_II" />
26.   </rdf:Description>
27.   <rdf:Description rdf:about="http://dbpedia.org/re
    source/John_McEwen">
28.     <dbo:monarch rdf:resource="http://dbpedia.org/r
    esource/Elizabeth_II" />
29.   </rdf:Description>
30.   <rdf:Description rdf:about="http://dbpedia.org/re
    source/Monarchy_of_Canada">
31.     <dbp:incumbent rdf:resource="http://dbpedia.org
    /resource/Elizabeth_II" />
32.   </rdf:Description>
33.   <rdf:Description rdf:about="http://dbpedia.org/re
    source/Thabo_Mbeki">
34.     <!--
35.     <p: xmlns:p="http://dbpedia.org/property/1named
    ata" rdf:resource="http://dbpedia.org/resource/Eliz
    abeth_II" />
36.     -->
37.   </rdf:Description>
```

Kode Sumber 4.3 RDF milik Elizabeth II

Tabel 4.1 Daftar Namespace Prefix milik Elizabeth II pada Neo4j

prefix	namespace
"ns8"	"http://purl.org/linguistics/gold/"
"ns4"	"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#"
"ns5"	"http://umbel.org/umbel/rc/"
"ns6"	"http://xmlns.com/foaf/0.1/"
"ns7"	"http://www.w3.org/ns/prov#"

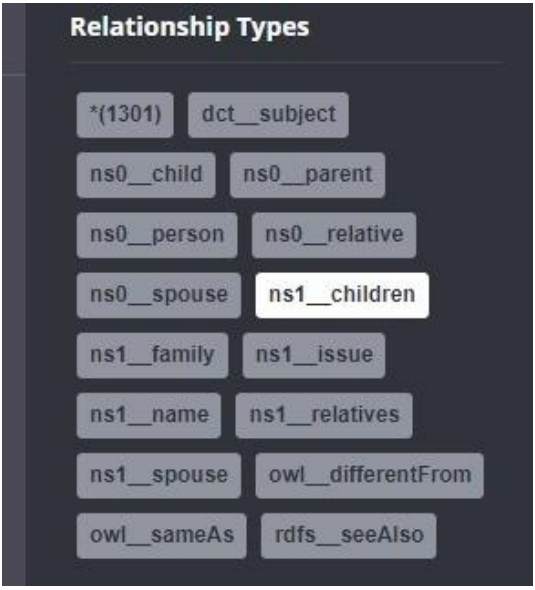
prefix	namespace
"rdfs"	"http://www.w3.org/2000/01/rdf-schema#"
"ns1"	"http://dbpedia.org/property/"
"ns2"	"http://dbpedia.org/class/yago/"
"ns3"	"http://www.wikidata.org/entity/"
"ns0"	"http://dbpedia.org/ontology/"
"skos"	"http://www.w3.org/2004/02/skos/core#"
"sch"	"http://schema.org/"
"sh"	"http://www.w3.org/ns/shacl#"
"dc"	"http://purl.org/dc/elements/1.1/"
"dct"	"http://purl.org/dc/terms/"
"rdf"	"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
"owl"	"http://www.w3.org/2002/07/owl#"

4.2. Implementasi Pembersihan Data pada Basis Data Graf

Gambar 4.3 Relasi Sebelum Dilakukan *Data Cleaning*

Pembersihan data atau data cleaning diperuntukan untuk mendapatkan data yang lebih ringan sehingga pada tahap selanjutnya data dapat lebih mudah diolah. *Data cleaning*

dilakukan terhadap relasi-relasi yang sekiranya tidak diperlukan pada tahap selanjutnya. Adapun relasi yang diperlukan yaitu relasi yang menyangkut dengan ontologi *family tree* yaitu *parent, child, spouse, dan issue*. *Data cleaning* yang dilakukan yaitu menggunakan sintaks *cypher* seperti pada Kode Sumber 4.4, sintaks tersebut dapat menghapus relasi tanpa menghapus *node* yang ada.



Gambar 4.4 Relasi Setelah Dilakukan Data Cleaning

```
1.  MATCH (n)-[r:ns0__almaMater]->()
2.  DELETE r;
3.
4.  MATCH (n)-[r:ns0__appointer]->()
5.  DELETE r;
6.
7.  MATCH (n)-[r:ns0__author]->()
8.  DELETE r;
```

Kode Sumber 4.4 *Cypher* Delete untuk *Data Cleaning*

Gambar 4.3 adalah gambar sebelum dilakukan *data cleaning* dan Gambar 4.4adalah gambar setelah dilakukan *data cleaning.* Pembersihan ini tidak akan menghilangkan IRI ataupun literal yang memiliki sifat seperti subjek dan predikat pada *triplestore*, pembersihan ini hanya akan menghapus relasi atau predikat saja. Penggambaran data cleaning dapat dilihat seperti pada gambar.

4.3. Implementasi *Reasoning* Data menggunakan Pellet pada Apache Jena

Pada subbab ini akan dijelaskan mengenai proses pengeksporan data dari Graf ke RDF dan proses *reasoning* pada aplikasi Java dengan *library* Apache Jena dan Pellet, serta pengimporan data kembali dari RDF ke basis data Graf.

4.3.1. Pengeksporan Basis Data Graf ke RDF

Pengeksporan basis data graf ke RDF dilakukan guna mengubah kembali *node-node* dan relasinya ke bentuk *triplestore* yang mana merupakan syarat agar Apache Jena dapat membaca data yang akan dilakukan *reasoning*. Sintaks *cypher* yang digunakan ialah seperti Kode Sumber 4.5 yang mana masih menggunakan bantuan library Neosematics dalam pengimplementasiannya. Sintaks tersebut berfungsi menarik semua data yang ada pada graf yang terbentuk didalam Neo4j tersebut.

Data awal yang merupakan graf kemudian akan diubah menjadi RDF seperti pada gambar. *Node labels, relationship keys*, hingga *property keys* akan berbentuk menjadi *triplestore* kembali yang mana terdiri dari subjek, predikat, dan objek. Dengan *namespace prefix* sesuatu standar RDF dan DBpedia seperti pada Tabel 4.2.

```
1.  :POST /rdf/cypheronrdf { "cypher":"MATCH (n:Resourc
    e)-[r]-(m) RETURN * " , "format" : "RDF/XML"}
```

Kode Sumber 4.5 *Cypher* untuk Mengekspor Basis Data Graf ke RDF

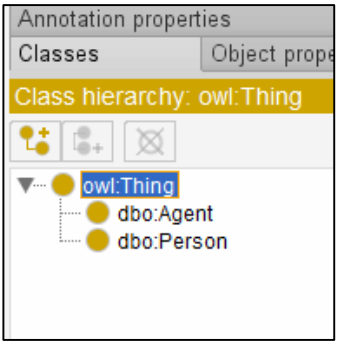
Tabel 4.2 Daftar *Namespace Prefix* standar pada RDF

Prefix	URL	Description
DBpedia Namespace		
dbr	http://dbpedia.org/resource/	*One-to-one mapping* antara artikel Wikipedia dengan sumber daya DBpedia
dbp	http://dbpedia.org/property/	Properti dari raw *infobox extraction*
dbo	http://dbpedia.org/ontology/	Ontologi DBpedia
yago	http://dbpedia.org/class/yago/	*Yet Another Great Ontology*
External namespaces		
dct	http://purl.org/dc/elements/1.1/	*Dublin core*
foaf	http://xmlns.com/foaf/0.1/	*Friend of a friend*
owl	http://www.w3.org/2002/07/owl#	*W3C web ontology language*
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	*Standart W3c RDF vocabulary*
rdfs	http://www.w3.org/2000/01/rdf-schema#	*Extention of the basic RDF vacabulary*

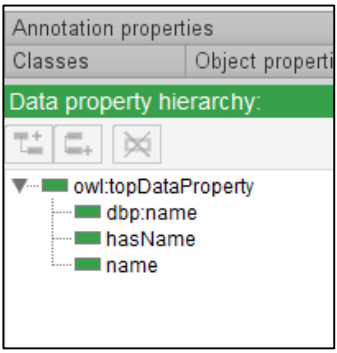
4.3.2. Pembuatan Ontologi *Family Tree*

Ontologi yang terbentuk merupakan ontologi yang juga digunakan oleh penelitian tugas akhir milik Madis dan Faiq. Pada penelitian tugas akhir milik Madis dan Faiq, ontologi yang

digunakan bersumber pada ontologi *Family Relationships Ontology* milik Robert Stevens. Ontologi tersebut berisi sejumlah *class*, *data property*, *individual*, *data properties*, dan *object properties* yang dapat membentuk pohon keluarga. Namun, tidak semua hal yang ada pada *Family Relationships Ontology* digunakan sepenuhnya. Pada ontologi ini, hanya beberapa properti saja yang diambil dan digunakan. Beberapa entitas tersebut dapat dilihat pada Gambar 4.5, Gambar 4.6, dan Gambar 4.7.



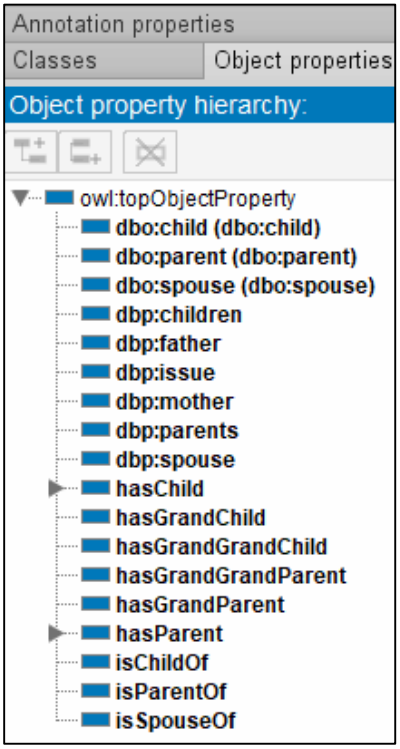
Gambar 4.5 Daftar *Class* pada Ontologi *Family Tree*



Gambar 4.6 Daftar *Data Properties* pada Ontologi *Family Tree*

Properti seperti *hasChild*, *hasParent*, *isChildOf*, *isParentOf*, *dan isSpouseOf* adalah properti yang bersumber dari *Family*

Relationships Ontology dan digunakan untuk pembuatan ontologi ini. Sedangkan properti *hasGrandParent, hasGrandChild, hasGrandGrandChild*, dan *hasGrandGrandParent* merupakan properti tambahan yang merupakan *property chain* yang terbentuk dari hubungan properti-properti yang telah ada. Properti seperti *dbo:child, dbo:parent, dbo:spouse, dbp:children, dbp:father, dbp:mother, dbp: issue, dpb:parents*, dan *dbp:spouse* merupakan properti yang bersumber dari DBpedia. Daftar properti tersebut dapat dilihat pada Tabel 4.3.



Gambar 4.7 Daftar *Object Properties* pada Ontologi *Family Tree*

Untuk properti yang memiliki persamaan dengan properti lain akan akan diatur sebagai *equivalent class*, seperti properti *hasParent, dbo:parent, dbp:parents, dbp:father, dbp:mother*, dan *isChildof* yang terlampir pada Tabel 4.4. Adapun aturan-aturan lain yang mana tertulis pada Tabel seperti *inverseOf* dan *symmetric*. Sedangkan penggunaan class dan data properties dapat dilihat pada Tabel 4.5 dan Tabel 4.6.

Tabel 4.3 Daftar *Object Properties* pada Ontologi *Family Tree*

Object property	URL	Karakteristik	Keterangan
dbo:child	dbpedia.org/ontology/child	-	ekuivalen dengan hasChild
dbo:parent	dbpedia.org/ontology/parent	-	ekuivalen dengan hasParent
dbo:spouse	dbpedia.org/ontology/spouse	symmetric	ekuivalen dengan isSpouseOf
dbp:children	dbpedia.org/property/children	-	ekuivalen dengan hasChild
dbp:issue	dbpedia.org/property/issue	-	ekuivalen dengan hasChild
hasChild	co-ode.org/roberts/family-tree.owl#haschild	-	ekuivalen dengan dbo:child, dbp:children, dbp:issue, dan isParentof

Object property	URL	Karakteristik	Keterangan
hasGrandChild	co-ode.org/roberts/family-tree.owl#hasgrandchild	-	SuperProperty dari 'hasChild o hasChild'
hasParent	co-ode.org/roberts/family-tree.owl#hasparent	-	ekuivalen dengan dbo:parent
hasGrandParent	co-ode.org/roberts/family-tree.owl#hasgrandparent	-	SuperProperty dari 'hasParent o hasParent'
isChildOf	co-ode.org/roberts/family-tree.owl#ischildof	-	inverse dari hasChild
isParentOf	co-ode.org/roberts/family-tree.owl#isparentof	-	ekuivalen dengan hasChild
isSpouseOf	co-ode.org/roberts/family-tree.owl#isspouseof	-	ekuivalen dengan dbo:spouse

Tabel 4.4 Pemetaan pada Ontologi *Family Tree*

Property DBpedia	Property Ontologi Family Tree
foaf:name	foaf:name
dbo:child	hasChild, isParentOf, dbp:children, dbp:issue, dbo:child

Property DBpedia	Property Ontologi Family Tree
dbo:spouse	isSpouseOf, dbo:spouse, dbp:spouse
dbo:parent	hasParent, dbo:parent, dbp:parents, isChildOf, dbp:father, dbp:mother

Tabel 4.5 Daftar *Class* pada Ontologi *Family Tree*

Class	URL	Karakteristik	Keterangan
Owl:Thing	http://www.w3.org/2002/07/owl#thing	-	Class untuk menjelaskan suatu hal
Dbo:agent	http://dbpedia.org/ontology/agent	-	Class untuk menjelaskan *agent*
Dbo:Person	http://dbpedia.org/ontology/person	-	Class untuk menjelaskan seseorang

Tabel 4.6 Daftar *Data Properties* pada Ontologi *Family Tree*

Data property	URL	Karakteristik	Keterangan
Dbp:name	dbpedia.org/property/name	-	Menerangkan data nama

Data property	URL	Karakteristik	Keterangan
hasName	co-ode.org/roberts/family-tree.owl#hasname	-	Menerangkan data nama
Label	rdfs:label	-	Menerangkan data label

4.3.3. Penggabungan Data RDF dengan Model Ontologi

```
1. String READ_RDF = "c:/Users/MI/Downloads/neo4jseb/103seb.rdf";
2. String ONTOLOGI = "c:/Users/MI/Downloads/ontologi_reason.owl";
```

Kode Sumber 4.6 Inisialisai Variable yang Digunakan

Data yang kembali menjadi RDF tersebut kemudian di *reasoning* pada aplikasi berbasis Java yang memiliki *plugin* Apache Jena dan *reasoner* Pallet. Pada tahap awal dilakukan inisialisasi variable untuk membaca data yang telah diekspor pada Neo4j (READ_RDF) dan ontologi (ONTOLOGI) yang digunakan seperti pada Kode Sumber 4.6.

Setelah variable-variable dari lokasi ontologi dan data diinisialisasi, langkah selanjutnya adalah memodelkan ontologi tersebut dengan menggunakan fungsi yang dimiliki oleh Apache Jena seperti pada Kode Sumber 4.7. READ_RDF dimodelkan dengan nama variable *Instances*, sedangkan ONTOLOGI dimodelkan dengan nama variable *famtree*

```
5. FileManager.get().addLocatorClassLoader(Main.class.
   getClassLoader());
6. Model Instances = FileManager.get().loadModel(READ_
   RDF);
7. Instances.read(READ_RDF,"RDF/XML");
```

```
8.
9.  Model famtree = FileManager.get().loadModel(ONTOLOG
    I);
```

Kode Sumber 4.7 Inisialisasi Model-Model pada Apache Jena

Pada Kode Kode Sumber 4.8, dilakukan penggabungan pada model yang telah diinisialisasi sebelumnya yaitu *Instances* dan *famtree*, yang mana hasilnya tersimpan pada variable *union.*

```
1.  final Model union = ModelFactory.createUnion(Instan
    ces,famtree);
```

Kode Sumber 4.8 Implementasi Penggabungan Model

```
1.  Reasoner pelletReasoner = PelletReasonerFactory.the
    Instance().create();
2.  InfModel reasonedModel = ModelFactory.createInfMode
    l(pelletReasoner,union);
```

Kode Sumber 4.9 Implementasi *Reasoning* pada Model yang telah Digabung

Reasoning dilakukan menggunakan bantuan *library* Pellet yang mana dapat memunculkan fakta-fakta baru untuk melengkapi data awal yang berasal dari Neo4j. Sintaks yang digunakan untuk mendeklarasi *library* Pallet adalah Kode Sumber 4.9.

Kemudian setelah dilakukan *reasoning*, data dan fakta-fakta yang baru disimpan pada suatu dokumen RDF menggunakan sintaks Kode Sumber 4.10 untuk selanjutnya dokumen tersebut diimpor pada tahap selanjutnya.

```
1.  if(fileRDF.delete())
2.  {
3.      System.out.println("The old result.rdf file del
    eted successfully");
4.  }
```

```
5.  else
6.  {
7.      System.out.println("Creating new result as RDF
    File");
8.  }
9.  PrintStream fileStream = new PrintStream("result103
    swrl.rdf");
10. System.setOut(fileStream);
11.
12. reasonedModel.write( System.out, "RDF/XML" );
```

Kode Sumber 4.10 Implementasi Pembuatan Dokumen bagi Data dan Fakta-Fakta Baru

Kode Sumber 4.11 merupakan data sebelum dilakukan *reasoning*. Hasil dari *reasoning* akan berbentuk seperti pada Kode Sumber 4.12 yaitu ada penambahan fakta baru. Fakta-fakta tersebut berasal dari ontologi yang telah dibuat dengan rules yang juga telah ada didalamnya seperti yang dijelaskan pada subbab 4.3.2.

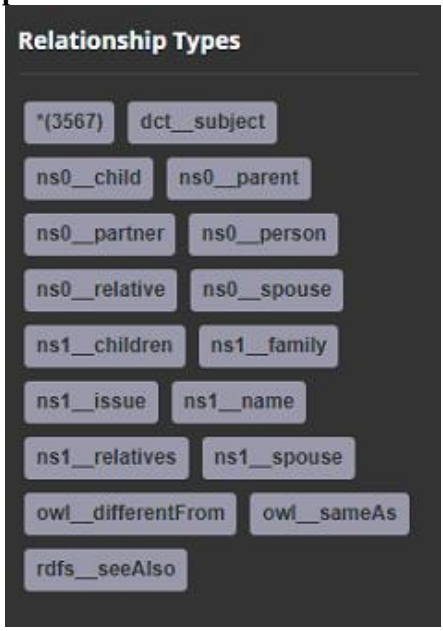
```
1.  <rdf:Description rdf:about="http://dbpedia.org/reso
    urce/Prince_Edward,_Earl_of_Wessex">
2.    <dbo:parent rdf:resource="http://dbpedia.org/reso
    urce/Prince_Philip,_Duke_of_Edinburgh" />
3.  <rdf:Description rdf:about="http://dbpedia.org/reso
    urce/Charles,_Prince_of_Wales">
4.    <dbo:parent rdf:resource="http://dbpedia.org/reso
    urce/Prince_Philip,_Duke_of_Edinburgh" />
5.  <rdf:Description rdf:about="http://dbpedia.org/reso
    urce/Prince_Andrew,_Duke_of_York">
6.    <dbo:parent rdf:resource="http://dbpedia.org/reso
    urce/Prince_Philip,_Duke_of_Edinburgh" />
7.  <rdf:Description rdf:about="http://dbpedia.org/reso
    urce/Anne,_Princess_Royal">
8.    <dbo:parent rdf:resource="http://dbpedia.org/reso
    urce/Prince_Philip,_Duke_of_Edinburgh" />
```

Kode Sumber 4.11 RDF dari Prince Philip dengan Properti dbo:parent sebelum *reasoning*

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Prince_Philip,_Duke_of_Edinburgh">
    <hasParent rdf:resource="http://dbpedia.org/resource/Princess_Alice_of_Battenberg"/>
    <hasChild rdf:resource="http://nl.dbpedia.org/resource/Anne_Mountbatten-Windsor"/>
    <hasChild rdf:resource="http://nl.dbpedia.org/resource/Edward,_graaf_van_Wessex"/>
    <hasChild rdf:resource="http://de.dbpedia.org/resource/Charles,_Prince_of_Wales"/>
    <hasChild rdf:resource="http://de.dbpedia.org/resource/Anne,_Princess_Royal"/
```

Kode Sumber 4.12 Hasil *reasoning* dengan *eqivalent class* sama dengan dbo:parent milik Prince Phillip

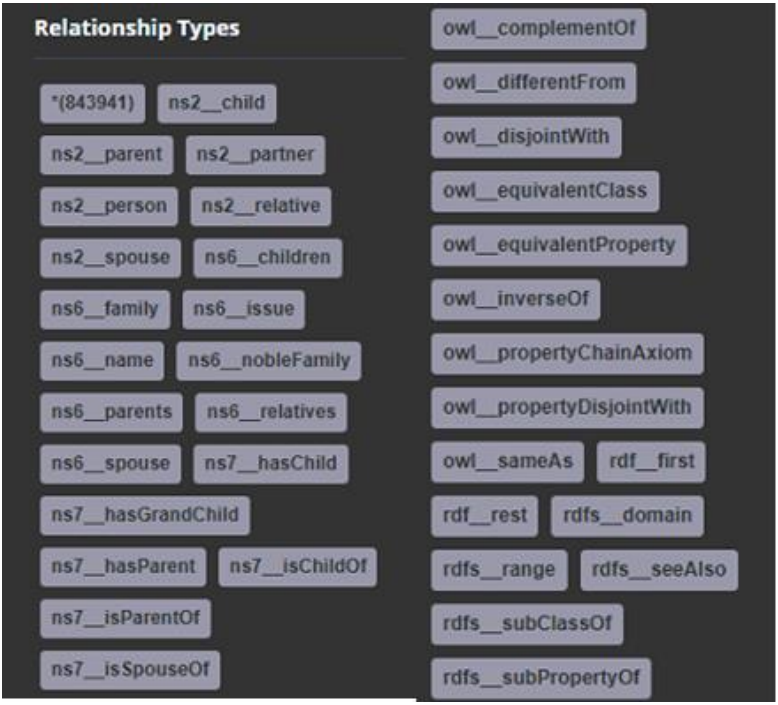
4.3.4. Pengimporan kembali RDF ke Basis Data Graf



Gambar 4.8 *Relationship Types* sebelum Dilakukan *Reasoning*

```
1. CREATE INDEX ON :Resource(uri);
2. CALL semantics.importRDF("file:///C:/Users/MI/Downl
   oads/neo4jfile/result33.rdf","RDF/XML",{commitSize:
   10});
```

Kode Sumber 4.13 *Cypher* untuk Mengimpor dari RDF ke Basis Data Graf Setelah *Reasoning*



Gambar 4.9 *Relationship Types* setelah Dilakukan *Reasoning*

Seperti halnya pada tahap pengimporan sebelumnya, pada tahap ini sintaks *cypher* yang digunakan juga masih menggunakan sintaks yang sama, yaitu terlampir pada Kode Sumber 4.13. Hasil pengimporan kan berbeda dengan hasil yang yang terlampir pada subbab 4.1.2, pada pengimporan pada tahap ini akan ada

penambahan data pada label dan relasinya, perubahan tersebut terlihat pada Gambar 4.8 yang merupakan sebelum dilakukan *reasoning* dengan Gambar 4.9 yaitu hasil setelah dilakukan *reasoning*. Properti seperti *hasGrandParent* dan *hasGrandChild* akan menjadi *relationship keys* dan beberapa aturan seperti *inverseOf* dan lainya akan ditambah dan diubah dengan *namespace prefix owl_inverseOf* dan lain sebagainya.

4.4. Implementasi Penguerian Data

Pada subbab ini akan dipaparkan kueri apa saja yang digunakan untuk memecahkan kasus uji yang dilakukan pada tugas akhir ini.

4.4.1. Penguerian Data Menampilkan Pohon Keluarga Seorang Tokoh

Implementasi kueri yang pertama dilakukan untuk menampilkan pohon keluarga seorang tokoh, dimana tokoh tersebut dipilih untuk menjadi center dari *node-node* lain yang mana terdapat relasi diantara keduanya. Sintaks cypher yang digunakan adalah pada Kode Sumber 4.14.

```
5.  MATCH p=(a:ns0__Person)-[:ns7_hasChild*]-
    >(allchildren)
6.  WHERE ' George V' IN a.` ns0__name`
7.  RETURN p LIMIT 150
```

Kode Sumber 4.14 *Cypher* untuk Menampilkan Pohon Keluarga Seorang Tokoh Bernama Price Andrew

4.4.2. Penguerian Data Menampilkan Relasi dari Dua Tokoh

```
1.  MATCH (n:ns0__Person{ns6__name:'Prince Andrew'}),(m
    :ns0__Person{ns6__name:'Sophie Helen'}),p = shortes
    tPath shortestPath((m)-[*..30]-(n))
```

```
2.  RETURN p
```

Kode Sumber 4.15 *Cypher* untuk Menampilkan Relasi dari Dua Tokoh

Implementasi kueri yang kedua adalah implementasi untuk menampilkan relasi dari dua tokoh. Yang mana, kedua tokoh tidak memiliki relasi antara keduanya, tetapi relasi tersebut dapat dicari dengan minimal satu *node* lain yang menghubungkan keduanya. Sintaks yang digunakan seperti yang terlampir pada Kode Sumber 4.15.

[Halaman ini sengaja dikosongkan]

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas mengenai pengujian dan evaluasi pada ontologi yang dikembangkan pada basis data graf. Pengujian yang dilakukan meliputi *reasoning* ontologi, pengujian penggunaan memori, pengujian relasi pada dua tokoh secara tidak langsung, dan pengujian pohon keluarga dari seorang tokoh. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas sebagai berikut:

Prosesor : Intel(R) Core™ i3-3240 CPU @ 3.40GHz
Memori : 10.00 GB
Jenis *Device* : Desktop
Sistem Operasi : Microsoft Windows 10 Home 64-bit
Reasoner : Pellet
Browser : Google Chrome
Database Server: Neo4J

5.2. Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian yang dilakukan yaitu dengan membandingkan data bentuk *graph database* atau basis data graf dengan data yang berbentuk *triplestore*.

Pengujian meliputi pengujian untuk menyelesaikan beberapa kasus pengujian meliputi pengujian untuk menghitung waktu ketika melakukan *reasoning*, pengujian penggunaan memori ketika melakukan kueri, dan menyelesaikan dua kasus yaitu mengetahui relasi dua tokoh dan mencari pohon keluarga dari suatu tokoh.

5.2.1. Pengujian *Reasoning*

Pada subbab ini pengujian dilakukan untuk menguji *reasoning* dari beberapa skenario yang ada pada Tabel 5.1.

Skenario pengujian *reasoning* tersebut bertujuan untuk membandingkan waktu atau *runtime* dari setiap skenario. Pada skenario kode uji R-01 merupakan skenario tanpa adanya relasi tambahan pada ontologi, sedangkan R-02 hingga R-07 memiliki sejumlah tambahan relasi dengan dua metode yang berbeda yaitu *property chain* dan SWRL.

Tabel 5.1 Skenario Pengujian *Reasoning*

Kode Uji	Nama Pengujian	Keterangan
R-01	*Reasoning* 2 generasi ke atas dan 2 generasi ke bawah	Untuk mengujikan waktu *reasoning* pada 2 generasi saja atau tanpa properti cicit dan buyut
R-02	*Reasoning* 3 generasi ke bawah dengan *property chain*	Untuk mengujikan waktu *reasoning* menggunakan properti cicit (3 generasi ke bawah) dengan *property chain*
R-03	*Reasoning* 3 generasi ke bawah dengan SWRL	Untuk mengujikan waktu *reasoning* menggunakan properti cicit (3 generasi ke bawah) dengan SWRL
R-04	*Reasoning* 3 generasi ke atas dengan *property chain*	Untuk mengujikan waktu *reasoning* menggunakan properti buyut (3 generasi ke atas) dengan *property chain*
R-05	*Reasoning* 3 generasi ke atas dengan SWRL	Untuk mengujikan waktu *reasoning* menggunakan properti buyut (3 generasi ke atas) dengan SWRL
R-06	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan *property chain*	Untuk mengujikan waktu *reasoning* menggunakan properti cicit dan buyut (3 generasi ke bawah dan ke

Kode Uji	Nama Pengujian	Keterangan
		atas) dengan *property chain*
R-07	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan SWRL	Untuk mengujikan waktu *reasoning* menggunakan properti cicit dan buyut (3 generasi ke bawah dan ke atas) dengan SWRL

5.2.2. Pengujian Penggunaan Memori

Pada subbab ini pengujian dilakukan untuk mengujikan perbandingan penggunaan memori antara *cypher query* dengan sparql *query*. Diujikan pula runtime dari masing-masing skenario yang diujikan, skenario tersebut terlampir pada Tabel 5.2.

Tabel 5.2 Skenario Pengujian Penggunaan Memori

Kode Uji	Nama Pengujian	Keterangan
M-01	Perhitungan memori pada penguerian Neo4j	Untuk mengukur runtime dan *memory usage* saat melakukan kueri pada Neo4j
M-02	Perhitungan memori pada penguerian Jena	Untuk mengukur runtime dan *memory usage* saat melakukan kueri pada Jena

5.2.3. Pengujian Relasi pada Dua Tokoh

Pada subbab ini pengujian dilakukan untuk mengujikan beberapa skenario yang berhubungan dengan relasi pada dua tokoh dengan pengkuerian yang dilakukan pada Neo4j menggunakan *cypher query*. Skenario yang diuji memiliki dua kondisi yaitu sebelum dilakukan *reasoning* dan setelah dilakukan *reasoning*. yang mana bertujuan untuk membuktikan apakah terdapat relasi pada tokoh-tokoh tersebut dan apakah relasi yang menghubungkan keduanya akan sama atau tidak seperti yang terlampir pada Tabel 5.3.

Tabel 5.3 Skenario Pengujian Relasi pada Dua Tokoh

Kode Uji	Nama Pengujian	Keterangan
T-01	Kondisi memiliki relasi sebelum di *reasoning*	Untuk menguji graf ketika dua tokoh berelasi secara tidak langsung sebelum di *reasoning*
T-02	Kondisi memiliki relasi sesudah di *reasoning*	Untuk menguji graf ketika dua tokoh berelasi secara tidak langsung setelah di *reasoning*
T-03	Kondisi tidak memiliki relasi sebelum di *reasoning*	Untuk menguji graf ketika dua tokoh tidak berelasi sama sekali sebelum di *reasoning*
T-04	Kondisi tidak memiliki relasi sesudah di *reasoning*	Untuk menguji graf ketika dua tokoh tidak berelasi sama sekali setelah di *reasoning*

5.2.4. Pengujian Pencarian Pohon Keluarga Seorang Tokoh

Pada subbab ini, skenario yang diujikan adalah pengujian untuk mencari pohon keluarga dari seorang tokoh seperti yang terlampir pada Tabel 5.4. Untuk seberapa dalam penelusuran dari pohon keluarga tersebut yaitu hingga generasi terakhir dari keluarga tokoh tersebut atau hingga tidak ada node lagi yang terhubung.

Tabel 5.4 Skenario Pengujian Pencarian Pohon Keluarga Seorang Tokoh

Kode Uji	Nama Pengujian	Keterangan
P-01	Pengujian pencarian pohon keluarga seorang tokoh	Untuk menguji relasi secara rekursi dari seorang tokoh

5.3. Hasil Pengujian

Pada subbab ini akan dipaparkan hasil dari pengujian-pengujian yang telah dilakukan. Hasil yang diberikan meliputi hasil pengujian *reasoning* dan hasil pengujian relasi antar dua tokoh serta pengujian satu tokoh menjadi pusat.

5.3.1. Hasil Pengujian *Reasoning*

Pada subbab ini akan terlampir hasil dari pengujian *reasoning* yang telah dilakukan. Tujuh poin pengujian mulai dari R-01 hingga R-07 terpapar pada Tabel 5.5- Tabel 5.11.

Tabel 5.5 Pengujian *Reasoning* 2 Generasi ke Atas dan 2 Generasi ke Bawah

Kode Uji	R-01
Nama Pengujian	*Reasoning* 2 generasi ke atas dan 2 generasi ke bawah
Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan instance	Lampiran 2
Keterangan	Menghitung *runtime* dari *reasoning* data 2 generasi saja atau tanpa properti cicit dan buyut dengan jumlah data 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.6 Pengujian *Reasoning* 3 Generasi ke Bawah dengan *Property Chain*

Kode Uji	R-02
Nama Pengujian	*Reasoning* 3 generasi ke bawah dengan *property chain*

Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan 33 instance	Lampiran 3
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti cicit (3 generasi ke bawah) dengan *property chain*. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.7 Pengujian *Reasoning* 3 Generasi ke Bawah dengan SWRL

Kode Uji	R-03
Nama Pengujian	*Reasoning* 3 generasi ke bawah dengan SWRL
Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan instance	Lampiran 4
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti cicit (3 generasi ke bawah) dengan SWRL. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.8 Pengujian *Reasoning* 3 Generasi ke Atas dengan *Property Chain*

Kode Uji	R-04
Nama Pengujian	*Reasoning* 3 generasi ke atas dengan *property chain*
Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan 33 instance	Lampiran 5
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti buyut (3 generasi ke atas) dengan *property chain*. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.9 Pengujian *Reasoning* 3 Generasi ke Atas dengan SWRL

Kode Uji	R-05
Nama Pengujian	*Reasoning* 3 generasi ke atas dengan SWRL
Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan instance	Lampiran 6
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti buyut (3

	generasi ke atas) dengan SWRL. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.10 Pengujian *Reasoning* 3 Generasi ke Atas dan 3 Generasi ke Bawah dengan *Property Chain*

Kode Uji	R-06
Nama Pengujian	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan *property chain*
Kondisi sebelum *reasoning*	Lampiran 1
Kondisi sesudah *reasoning* dengan instance	Lampiran 7
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti cicit dan buyut (3 generasi ke bawah dan ke atas) dengan *property chain*. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

Tabel 5.11 Pengujian *Reasoning* 3 Generasi ke Atas Dan 3 Generasi ke Bawah dengan SWRL

Kode Uji	R-07
Nama Pengujian	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan SWRL
Kondisi sebelum *reasoning*	Lampiran 1

Kondisi sesudah *reasoning* dengan instance	Lampiran 8
Keterangan	Menghitung *runtime* dari *reasoning* data menggunakan properti cicit dan buyut (3 generasi ke bawah dan ke atas) dengan SWRL. Jumlah data yaitu 33, 53, 103, 153, 203.
Hasil	Berhasil

5.3.2. Hasil Pengujian Penggunaan Memori

Pada subbab ini akan terlampir hasil dari pengujian penggunaan memori yang telah dilakukan. Pengujian mulai dari M-01 dan M-02 terpapar pada Tabel 5.12 dan Tabel 5.13.

Tabel 5.12 Pengujian Penggunaan Memori pada Pengkuerian Neo4j

Kode Uji	M-01
Nama Pengujian	Perhitungan memori pada penguerian Neo4j
Hasil	Lampiran 16, Lampiran 17
Keterangan	Untuk mengukur runtime dan *memory usage* saat melakukan kueri pada Neo4j dengan jumlah data 30, 50, 100, 150, 200.
Nilai	Berhasil

Tabel 5.13 Pengujian Penggunaan Memori pada Pengkuerian Sparql

Kode Uji	M-02
Nama Pengujian	Perhitungan memori pada penguerian Jena
Hasil	Lampiran 16, Lampiran 17

Keterangan	Untuk mengukur runtime dan *memory usage* saat melakukan kueri pada Jena dengan jumlah data 30, 50, 100, 150, 200.
Hasil	Berhasil

5.3.3. Hasil Pengujian Relasi antar Dua Tokoh

Pada subbab ini akan terlampir hasil dari pengujian relasi antar dua tokoh yang telah dilakukan. Tiap poin pengujian mulai dari T-01 hingga T-04 terpapar pada Tabel 5.14 hingga Tabel 5.17.

Tabel 5.14 Pengujian Relasi antar Dua Tokoh Sebelum Dilakukan *Reasoning*

Kode Uji	T-01
Nama Pengujian	Kondisi memiliki relasi sebelum di *reasoning*
Hasil	Lampiran 9
Keterangan	Contoh relasi dari Prince Andrew dengan Sophie Helen sebelum dilakukan *reasoning*
Hasil	Berhasil

Tabel 5.15 Pengujian Relasi antar Dua Tokoh Sesudah Dilakukan *Reasoning*

Kode Uji	T-02
Nama Pengujian	Kondisi memiliki relasi sesudah di *reasoning*
Hasil	Lampiran 10
Keterangan	Contoh relasi dari Prince Andrew dengan Sophie Helen setelah dilakukan *reasoning*
Hasil	Berhasil

Tabel 5.16 Pengujian Tidak adanya Relasi antar Dua Tokoh Sebelum Dilakukan *Reasoning*

Kode Uji	T-03
Nama Pengujian	Kondisi tidak memiliki relasi sebelum di *reasoning*
Hasil	Lampiran 11
Keterangan	Contoh relasi dari Prince Andrew dengan Sukarno sebelum dilakukan *reasoning*
Hasil	Berhasil

Tabel 5.17 Pengujian Tidak adanya Relasi antar Dua Tokoh Sesudah Dilakukan *Reasoning*

Kode Uji	T-04
Nama Pengujian	Kondisi tidak memiliki relasi sesudah di *reasoning*
Hasil	Lampiran 12
Keterangan	Contoh relasi dari Prince Andrew dengan Sukarno sesudah dilakukan *reasoning*
Hasil	Berhasil

5.3.4. Hasil Pengujian Pengujian Pencarian Pohon Keluarga Seorang Tokoh

Pada subbab ini akan terlampir hasil dari pengujian *reasoning* yang telah dilakukan, hasil dapat dilihat pada Tabel 5.18.

Tabel 5.18 Pengujian Pengujian Pencarian Pohon Keluarga Seorang Tokoh

Kode Uji	P-01
Nama Pengujian	Pengujian pencarian pohon keluarga seorang tokoh
Hasil	Lampiran 13 dan Lampiran 14
Keterangan	Untuk menguji silsilah keluarga dari George V dan keluarga Mark Phillips

	dengan menggunakan relasi hasChild atau menelususi generasi ke bawah.
Hasil	Berhasil

5.4. Evaluasi Pengujian

Pada subbab ini akan diberikan hasil evaluasi dari pengujian-pengujian yang telah dilakukan. Evaluasi yang diberikan meliputi evaluasi pengujian *reasoning* yang telah dijelaskan pada Subbab 5.2, evaluasi pengujian efisiensi redudansi data, evaluasi pengujian relasi antar dua tokoh yang telah dijelaskan pada, dan evaluasi pengujian pencarian pohon keluarga seorang tokoh yang telah dijelaskan pada.

5.4.1. Evaluasi Pengujian *Reasoning*

Rangkuman mengenai hasil pengujian *reasoning* dapat dilihat pada Tabel 5.19. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga bisa ditarik disimpulkan bahwa ontologi yang dikembangkan telah sesuai dengan yang diharapkan.

Tabel 5.19 Evaluasi Pengujian *Reasoning*

Kode Uji	Nama Pengujian	Hasil
R-01	*Reasoning* 2 generasi ke atas dan 2 generasi ke bawah	Berhasil
R-02	*Reasoning* 3 generasi ke bawah dengan property chain	Berhasil
R-03	*Reasoning* 3 generasi ke bawah dengan SWRL	Berhasil
R-04	*Reasoning* 3 generasi ke atas dengan property chain	Berhasil
R-05	*Reasoning* 3 generasi ke bawah dengan SWRL	Berhasil
R-06	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan property chain	Berhasil

Kode Uji	Nama Pengujian	Hasil
R-07	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan SWRL	Berhasil

Untuk perbandingan waktu hasil runtime pengujian *reasoning* pada subbab 5.3.1 dapat dilihat pada Tabel 5.20. Didapatkan bahwa perbandingan runtime dari setiap jumlah data dengan skenario uji yang berbeda memiliki penggambaran grafik seperti Lampiran. Berdasarkan perbandingan waktu hasil runtime pengujian *reasoning* didapatkan jika semakin besar data semakin terlihat perbedaan antara *rules* menggunakan SWRL dan menggunakan *rules property chain*, yang mana disimpulkan bahwa SWRL memiliki waktu lebih cepat dalam melakukan *reasoning*. Grafik dari hasil runtime dapat dilihat pada Lampiran 15.

Tabel 5.20 Evaluasi Hasil Runtime *Reasoning*

Kode Uji	Nama Pengujian	Jumlah Data				
		33	53	103	153	203
R-01	*Reasoning* 2 generasi ke atas dan 2 generasi ke bawah	53 detik	58 detik	2 menit 28 detik	4 menit 37 detik	8 menit 36 detik
R-02	*Reasoning* 3 generasi ke bawah dengan property chain	32 detik	59 detik	2 menit 46 menit	4 menit 29 detik	7 menit 45 detik
R-03	*Reasoning* 3 generasi ke bawah	38 detik	59 detik	2 menit 39 detik	4 menit 55 detik	7 menit 24 detik

Kode Uji	Nama Pengujian	Jumlah Data				
		33	53	103	153	203
	dengan SWRL					
R-04	*Reasoning* 3 generasi ke atas dengan property chain	38 detik	1 menit 3 detik	2 menit 36 detik	6 menit 1 detik	8 menit 33 detik
R-05	*Reasoning* 3 generasi ke bawah dengan SWRL	40 detik	1 menit 7 detik	2 menit 31 detik	5 menit 4 detik	8 menit 1 detik
R-06	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan property chain	40 detik	1 menit	2 menit 39 detik	7 menit 15 detik	9 menti 5 detik
R-07	*Reasoning* 3 generasi ke atas dan 3 generasi ke bawah dengan SWRL	36 detik	58 detik	2 menit 24 detik	4 menit 33 detik	7 menit 30 detik

5.4.2. Evaluasi Pengujian Penggunaan Memori

Rangkuman mengenai hasil pengujian penggunaan memori dapat diliat pada Tabel 5.21. Sedangkan untuk pengukuran

waktu saat dilakukan pengkuerian dapat dilihat pada Tabel 5.22 Berdasarkan data pada tabel tersebut dapat dilihat bawah memori yang digunakan ketika melakukan kueri pada Jena berbentuk parabola, semakin banyak tokoh maka semakin banyak memori yang digunakan. Berbeda dengan neo4j, grafik yang dihasilkan terlihat stabil dengan catatan waktu yang kecil pada pengkueriannya seperti terlampir pada Lampiran 16. Untuk grafik penggunaan memori dapat dilihat pada Lampiran 17.

Tabel 5.21 Evaluasi Pengujian Penggunaan Memori

Jumlah tokoh	Neo4j	Spark
30	10407208	74244832
50	7999240	8175520
100	8216568	8176208
150	8419552	8186216
200	11099096	311414760

Tabel 5.22 Evaluasi Pengujian *Runtime* Kueri

Jumlah tokoh	neo4j	spark
30	359	2883
50	248	3754
100	239	3730
150	352	2668
200	418	3061

5.4.3. Evaluasi Pengujian Relasi pada Dua Tokoh

Rangkuman mengenai hasil pengujian relasi pada dua tokoh dapat dilihat pada Tabel 5.23. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga bisa ditarik disimpulkan bahwa ontologi yang dikembangkan telah sesuai dengan yang diharapkan.

Tabel 5.23 Evaluasi Pengujian Realasi pada Dua Tokoh

Kode Uji	Nama Pengujian	Hasil
T-01	Kondisi memiliki relasi sebelum di *reasoning*	Berhasil
T-02	Kondisi memiliki relasi sesudah di *reasoning*	Berhasil
T-03	Kondisi tidak memiliki relasi sebelum di *reasoning*	Berhasil
T-04	Kondisi tidak memiliki relasi sesudah di *reasoning*	Berhasil

5.4.4. Evaluasi Pengujian Pencarian Pohon Keluarga Seorang Tokoh

Rangkuman mengenai hasil pengujian pencarian pohon keluarga seorang tokoh dapat dilihat pada Tabel 5.24. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga bisa ditarik disimpulkan bahwa ontologi yang dikembangkan telah sesuai dengan yang diharapkan.

Tabel 5.24 Evaluasi Pengujian Pohon Keluarga Seorang Tokoh

Kode Uji	Nama Pengujian	Hasil
P-01	Pengujian pencarian pohon keluarga seorang tokoh	Berhasil

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1) Skema penyimpanan relasi silsilah keluarga berhasil dimodelkan dengan mengindeksi sumber data yang diimpor pada Neo4j *graph database* dan mengubah data RDF yang didapat menjadi *node*, *relationship*, dan *property*. Node sebagai *person*, *relationship* sebagai relasi, dan *property* sebagai literal atau data yang melengkapi seorang *person.*
2) Dengan *graph database*, relasi antar dua tokoh berhasil didapatkan dengan menelusuri relasi dari tokoh dengan algoritma *shortest path.* Algoritma tersebut bekerja dengan cara mencari *node* dari dua tokoh yang dicari lalu mencari relasi yang paling mungkin dengan jalur yang paling pendek.
3) Fakta baru berhasil didapat dari *reasoning* dari model ontologi dan model data sumber yang digabungkan menjadi satu. *Reasoning* tersebut dilakukan dengan bantuan *library* Jena dan *reasoner* Pellet. Hasil *reasoning* kemudian disimpan pada *graph database*, yang selanjutnya dapat dimanfaatkan untuk menjawab kueri.

6.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada

hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1) Mengembangkan sistem agar dapat mengambil dan memilih satu *resource* diantara banyak *resource* yang dimiliki oleh seorang tokoh.
2) Penggunaan perangkat uji coba dengan spesifikasi kapasitas memori yang lebih besar agar waktu yang dibutuhkan untuk proses *reasoning* dan pengimporan data lebih cepat serta pengkuerian dapat berjalan lancar.

DAFTAR PUSTAKA

[1] "Arti kata keluarga - Kamus Besar Bahasa Indonesia (KBBI) Online." [Daring]. Tersedia pada: https://kbbi.web.id/keluarga. [Diakses: 29-Nov-2019].

[2] "Arti kata silsilah - Kamus Besar Bahasa Indonesia (KBBI) Online." [Daring]. Tersedia pada: https://kbbi.web.id/silsilah. [Diakses: 29-Des-2019].

[3] "Arti kata generasi - Kamus Besar Bahasa Indonesia (KBBI) Online." [Daring]. Tersedia pada: https://kbbi.web.id/generasi. [Diakses: 29-Des-2019].

[4] Y. F. Badron, F. Agus, dan H. R. Hatta, "Studi Tentang Pemodelan Ontologi Web Semantik dan Prospek Penerapan pada Bibliografi Artikel Jurnal Ilmiah," dalam *Prosiding Seminar Ilmu Komputer dan Teknologi Informasi*, 2017, hlm. 164–169.

[5] "What is an Ontology?" [Daring]. Tersedia pada: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html. [Diakses: 20-Jan-2020].

[6] "RDF - Semantic Web Standards." [Daring]. Tersedia pada: https://www.w3.org/RDF/. [Diakses: 29-Nov-2019].

[7] K. Balog, *Entity-Oriented Search*. Springer, 2018.

[8] "RDF 1.1 Concepts and Abstract Syntax." [Daring]. Tersedia pada: https://www.w3.org/TR/rdf11-concepts/#dfn-iri. [Diakses: 30-Des-2019].

[9] "RDF contoh." [Daring]. Tersedia pada: http://www.w3big.com/id/rdf/rdf-example.html. [Diakses: 29-Nov-2019].

[10] I. Robinson, J. Webber, dan E. Eifrem, *Graph databases*. O'Reilly Media, Inc., 2013.

[11] "What Is a Graph Database and Property Graph | Neo4j," *Neo4j Graph Database Platform*. [Daring]. Tersedia pada: https://neo4j.com/developer/graph-database/. [Diakses: 29-Nov-2019].

[12] "Apache Jena - Semantic Web Standards." [Daring]. Tersedia pada: https://www.w3.org/2001/sw/wiki/Apache_Jena. [Diakses: 29-Nov-2019].
[13] S. Tamba dan E. Surya, "PENGEMBANGAN KEMAMPUAN PENALARAN MATEMATIS (REASONING MATHEMATICS ABILITY) DALAM PENALARAN DEDUKTIF DAN INDUKTIF."
[14] "Penalaran Deduktif, Induktif, dan Salah Nalar Halaman all - Kompasiana.com." [Daring]. Tersedia pada: https://www.kompasiana.com/bugarsinar/5c040d9cab12ae425772afd3/penalaran-deduktif-induktif-dan-salah-nalar?page=all. [Diakses: 29-Nov-2019].
[15] "7 Types of Reasoning," *Simplicable*. [Daring]. Tersedia pada: https://simplicable.com/new/reasoning. [Diakses: 29-Nov-2019].
[16] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, dan Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 5, no. 2, hlm. 51–53, 2007.
[17] "NSMNTX User Guide." [Daring]. Tersedia pada: https://neo4j.com/docs/labs/nsmntx/current/. [Diakses: 29-Nov-2019].
[18] "What is Java programming language?," *HowToDoInJava*. [Daring]. Tersedia pada: https://howtodoinjava.com/java/basics/what-is-java-programming-language/. [Diakses: 29-Nov-2019].
[19] "What is Java? A Beginner's Guide to Java and its Evolution," *Edureka*, 19-Apr-2017. [Daring]. Tersedia pada: https://www.edureka.co/blog/what-is-java/. [Diakses: 29-Nov-2019].
[20] M. SARALITA, "PENCARIAN RELASI ANTAR TOKOH SEJARAH INDONESIA MENGGUNAKAN ONTOLOGI," ITS, Surabaya, 2016.

[21] FAIQ, "RANCANG BANGUN APLIKASI BERBASIS WEB UNTUK VISUALISASI POHON KELUARGA TOKOH SEJARAH INDONESIA MENGGUNAKAN ONTOLOGI DBPEDIA DAN PELLET REASONER," ITS, Surabaya, 2019.

[22] J.Barrasa, "Importing RDF data into Neo4j," *Jesús Barrasa*, 07-Jun-2016. [Daring]. Tersedia pada: https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/. [Diakses: 08-Des-2019].

[23] "Neo4j: A Reasonable RDF Graph Database & Reasoning Engine [Community Post]," *Neo4j Graph Database Platform*, 27-Feb-2018. [Daring]. Tersedia pada: https://neo4j.com/blog/neo4j-rdf-graph-database-reasoning-engine/. [Diakses: 30-Des-2019].

[Halaman ini sengaja dikosongkan]

LAMPIRAN A

Nama Tokoh	URL
Adolphus Cambridge, 1st Marquess of Cambridge	http://dbpedia.org/page/Adolphus_Cambridge,_1st_Marquess_of_Cambridge
Albert, Prince Consort	http://dbpedia.org/page/Albert,_Prince_Consort
Alexandra Feodorovna (Charlotte of Prussia)	http://dbpedia.org/page/Alexandra_Feodorovna_(Charlotte_of_Prussia)
Alexandra of Denmark	http://dbpedia.org/page/Alexandra_of_Denmark
Alfred, Duke of Saxe-Coburg and Gotha	http://dbpedia.org/page/Alfred,_Duke_of_Saxe-Coburg_and_Gotha
Amelia of Nassau-Weilburg	http://dbpedia.org/page/Amelia_of_Nassau-Weilburg
Anne, Princess Royal	http://dbpedia.org/page/Anne,_Princess_Royal
Anne, Princess Royal and Princess of Orange	http://dbpedia.org/page/Anne,_Princess_Royal_and_Princess_of_Orange
Anne Caroline Salisbury	http://dbpedia.org/page/Anne_Caroline_Salisbury
Augustus, Duke of Saxe-Gotha-Altenburg	http://dbpedia.org/page/Augustus,_Duke_of_Saxe-Gotha-Altenburg
Autumn Phillips	http://dbpedia.org/page/Autumn_Phillips

Nama Tokoh	URL
Charlotte of Mecklenburg-Strelitz	http://dbpedia.org/page/Charlotte_of_Mecklenburg-Strelitz
Christian IX of Denmark	http://dbpedia.org/page/Christian_IX_of_Denmark
Christian VII of Denmark	http://dbpedia.org/page/Christian_VII_of_Denmark
Claude Bowes-Lyon, 13th Earl of Strathmore and Kinghorne	http://dbpedia.org/page/Claude_Bowes-Lyon,_13th_Earl_of_Strathmore_and_Kinghorne
Claude Bowes-Lyon, 14th Earl of Strathmore and Kinghorne	http://dbpedia.org/page/Claude_Bowes-Lyon,_14th_Earl_of_Strathmore_and_Kinghorne
Countess Augusta Reuss of Ebersdorf	http://dbpedia.org/page/Countess_Augusta_Reuss_of_Ebersdorf
Countess Caroline Felizitas of Leiningen-Dagsburg	http://dbpedia.org/page/Countess_Caroline_Felizitas_of_Leiningen-Dagsburg
Countess Charlotte of Dohna-Leistenau	http://dbpedia.org/page/Countess_Charlotte_of_Dohna-Leistenau
Countess Claudine Rhédey von Kis-Rhéde	http://dbpedia.org/page/Countess_Claudine_Rhédey_von_Kis-Rhéde
Countess Friederike of Schlieben	http://dbpedia.org/page/Countess_Friederike_of_Schlieben
Countess Karoline Ernestine of Erbach-Schönberg	http://dbpedia.org/page/Countess_Karoline_Ernestine_of_Erbach-Schönberg
Diana, Princess of Wales	http://dbpedia.org/page/Diana,_Princess_of_Wales

Nama Tokoh	URL
Duchess Amelia of Württemberg	http://dbpedia.org/page/Duchess_Amelia_of_Württemberg
Duchess Charlotte Frederica of Mecklenburg-Schwerin	http://dbpedia.org/page/Duchess_Charlotte_Frederica_of_Mecklenburg-Schwerin
Duchess Charlotte Georgine of Mecklenburg-Strelitz	http://dbpedia.org/page/Duchess_Charlotte_Georgine_of_Mecklenburg-Strelitz
Duchess Elisabeth Alexandrine of Württemberg	http://dbpedia.org/page/Duchess_Elisabeth_Alexandrine_of_Württemberg
Duchess Elisabeth of Württemberg	http://dbpedia.org/page/Duchess_Elisabeth_of_Württemberg
Duchess Frederica of Württemberg	http://dbpedia.org/page/Duchess_Frederica_of_Württemberg
Duchess Louise Charlotte of Mecklenburg-Schwerin	http://dbpedia.org/page/Duchess_Louise_Charlotte_of_Mecklenburg-Schwerin
Duchess Sophia Frederica of Mecklenburg-Schwerin	http://dbpedia.org/page/Duchess_Sophia_Frederica_of_Mecklenburg-Schwerin
Duke Alexander of Württemberg (1804–1885)	http://dbpedia.org/page/Duke_Alexander_of_Württemberg_(1804–1885)
Duke Charles Louis Frederick of Mecklenburg	http://dbpedia.org/page/Duke_Charles_Louis_Frederick_of_Mecklenburg

Nama Tokoh	URL
Duke Eugen of Württemberg (1758–1822)	http://dbpedia.org/page/Duke_Eugen_of_Württemberg_(1758–1822)
Duke Louis of Mecklenburg-Schwerin	http://dbpedia.org/page/Duke_Louis_of_Mecklenburg-Schwerin
Duke William Frederick Philip of Württemberg	http://dbpedia.org/page/Duke_William_Frederick_Philip_of_Württemberg
Edward VII	http://dbpedia.org/page/Edward_VII
Edward VIII	http://dbpedia.org/page/Edward_VIII
Edwyn Burnaby (courtier)	http://dbpedia.org/page/Edwyn_Burnaby_(courtier)
Elizabeth II	http://dbpedia.org/page/Elizabeth_II
Ernest Augustus, King of Hanover	http://dbpedia.org/page/Ernest_Augustus,_King_of_Hanover
Ernest I, Duke of Saxe-Coburg and Gotha	http://dbpedia.org/page/Ernest_I,_Duke_of_Saxe-Coburg_and_Gotha
Ernest II, Duke of Saxe-Coburg and Gotha	http://dbpedia.org/page/Ernest_II,_Duke_of_Saxe-Coburg_and_Gotha
Ernest II, Duke of Saxe-Gotha-Altenburg	http://dbpedia.org/page/Ernest_II,_Duke_of_Saxe-Gotha-Altenburg
Family of Catherine, Duchess of Cambridge	http://dbpedia.org/page/Family_of_Catherine,_Duchess_of_Cambridge

Nama Tokoh	URL
Ferdinand, Hereditary Prince of Denmark	http://dbpedia.org/page/Ferdinand,_Hereditary_Prince_of_Denmark
Frances Bowes-Lyon, Countess of Strathmore and Kinghorne	http://dbpedia.org/page/Frances_Bowes-Lyon,_Countess_of_Strathmore_and_Kinghorne
Frances Shand Kydd	http://dbpedia.org/page/Frances_Shand_Kydd
Francis, Duke of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Francis,_Duke_of_Saxe-Coburg-Saalfeld
Francis, Duke of Teck	http://dbpedia.org/page/Francis,_Duke_of_Teck
Frederick, Duke of Saxe-Altenburg	http://dbpedia.org/page/Frederick,_Duke_of_Saxe-Altenburg
Frederick, Hereditary Prince of Denmark	http://dbpedia.org/page/Frederick,_Hereditary_Prince_of_Denmark
Frederick, Prince of Wales	http://dbpedia.org/page/Frederick,_Prince_of_Wales
Frederick Francis I, Grand Duke of Mecklenburg-Schwerin	http://dbpedia.org/page/Frederick_Francis_I,_Grand_Duke_of_Mecklenburg-Schwerin
Frederick II, Landgrave of Hesse-Kassel	http://dbpedia.org/page/Frederick_II,_Landgrave_of_Hesse-Kassel
Frederick II Eugene, Duke of Württemberg	http://dbpedia.org/page/Frederick_II_Eugene,_Duke_of_Württemberg

Nama Tokoh	URL
Frederick IV, Duke of Saxe-Gotha-Altenburg	http://dbpedia.org/page/Frederick_IV,_Duke_of_Saxe-Gotha-Altenburg
Frederick V of Denmark	http://dbpedia.org/page/Frederick_V_of_Denmark
Frederick VIII of Denmark	http://dbpedia.org/page/Frederick_VIII_of_Denmark
Frederick William, Margrave of Brandenburg-Schwedt	http://dbpedia.org/page/Frederick_William,_Margrave_of_Brandenburg-Schwedt
Frederick William, Prince of Nassau-Weilburg	http://dbpedia.org/page/Frederick_William,_Prince_of_Nassau-Weilburg
Frederick William III of Prussia	http://dbpedia.org/page/Frederick_William_III_of_Prussia
Friedrich, Duke of Schleswig-Holstein-Sonderburg-Glücksburg	http://dbpedia.org/page/Friedrich,_Duke_of_Schleswig-Holstein-Sonderburg-Glücksburg
Friedrich Karl Ludwig, Duke of Schleswig-Holstein-Sonderburg-Beck	http://dbpedia.org/page/Friedrich_Karl_Ludwig,_Duke_of_Schleswig-Holstein-Sonderburg-Beck
Friedrich Wilhelm, Duke of Schleswig-Holstein-Sonderburg-Glücksburg	http://dbpedia.org/page/Friedrich_Wilhelm,_Duke_of_Schleswig-Holstein-Sonderburg-Glücksburg
Georg, Duke of Saxe-Altenburg	http://dbpedia.org/page/Georg,_Duke_of_Saxe-Altenburg
George I of Great Britain	http://dbpedia.org/page/George_I_of_Great_Britain
George I of Greece	http://dbpedia.org/page/George_I_of_Greece

Nama Tokoh	URL
George II of Great Britain	http://dbpedia.org/page/George_II_of_Great_Britain
George IV of the United Kingdom	http://dbpedia.org/page/George_IV_of_the_United_Kingdom
George V	http://dbpedia.org/page/George_V
George VI	http://dbpedia.org/page/George_VI
Grand Duke Konstantin Nikolayevich of Russia	http://dbpedia.org/page/Grand_Duke_Konstantin_Nikolayevich_of_Russia
Heinrich XXIV, Count Reuss of Ebersdorf	http://dbpedia.org/page/Heinrich_XXIV,_Count_Reuss_of_Ebersdorf
Henrietta Mildred Hodgson	http://dbpedia.org/page/Henrietta_Mildred_Hodgson
Isla Phillips	http://dbpedia.org/page/Isla_Phillips
James, Viscount Severn	http://dbpedia.org/page/James,_Viscount_Severn
John Spencer, 8th Earl Spencer	http://dbpedia.org/page/John_Spencer,_8th_Earl_Spencer
Joseph, Duke of Saxe-Altenburg	http://dbpedia.org/page/Joseph,_Duke_of_Saxe-Altenburg
Juliana Maria of Brunswick-Wolfenbüttel	http://dbpedia.org/page/Juliana_Maria_of_Brunswick-Wolfenbüttel

Nama Tokoh	URL
Karl, Duke of Schleswig-Holstein-Sonderburg-Glücksburg	http://dbpedia.org/page/Karl,_Duke_of_Schleswig-Holstein-Sonderburg-Glücksburg
Lady Louise Windsor	http://dbpedia.org/page/Lady_Louise_Windsor
Leopold I of Belgium	http://dbpedia.org/page/Leopold_I_of_Belgium
Louis, Duke of Württemberg	http://dbpedia.org/page/Louis,_Duke_of_Württemberg
Louisa Cavendish-Bentinck	http://dbpedia.org/page/Louisa_Cavendish-Bentinck
Louise, Princess Royal	http://dbpedia.org/page/Louise,_Princess_Royal
Louise of Great Britain	http://dbpedia.org/page/Louise_of_Great_Britain
Louise of Hesse-Kassel	http://dbpedia.org/page/Louise_of_Hesse-Kassel
Louise of Mecklenburg-Strelitz	http://dbpedia.org/page/Louise_of_Mecklenburg-Strelitz
Margravine Elisabeth Louise of Brandenburg-Schwedt	http://dbpedia.org/page/Margravine_Elisabeth_Louise_of_Brandenburg-Schwedt
Margravine Friederike of Brandenburg-Schwedt	http://dbpedia.org/page/Margravine_Friederike_of_Brandenburg-Schwedt
Margravine Philippine of Brandenburg-Schwedt	http://dbpedia.org/page/Margravine_Philippine_of_Brandenburg-Schwedt
Maria Feodorovna (Dagmar of Denmark)	http://dbpedia.org/page/Maria_Feodorovna_(Dagmar_of_Denmark)

Nama Tokoh	URL
Maria Feodorovna (Sophie Dorothea of Württemberg)	http://dbpedia.org/page/Maria_Feodorovna_(Sophie_Dorothea_of_Württemberg)
Maria Feodorovna (Sophie Dorothea of Württemberg)	http://dbpedia.org/page/Maria_Feodorovna_(Sophie_Dorothea_of_Württemberg)
Marie of Hesse-Kassel	http://dbpedia.org/page/Marie_of_Hesse-Kassel
Marie of Saxe-Altenburg	http://dbpedia.org/page/Marie_of_Saxe-Altenburg
Mark Phillips	http://dbpedia.org/page/Mark_Phillips
Mary, Princess Royal and Countess of Harewood	http://dbpedia.org/page/Mary,_Princess_Royal_and_Countess_of_Harewood
Mary Elphinstone, Lady Elphinstone	http://dbpedia.org/page/Mary_Elphinstone,_Lady_Elphinstone
Mary of Teck	http://dbpedia.org/page/Mary_of_Teck
Maud of Wales	http://dbpedia.org/page/Maud_of_Wales
Mike Tindall	http://dbpedia.org/page/Mike_Tindall
Nicholas I of Russia	http://dbpedia.org/page/Nicholas_I_of_Russia
Olga Constantinovna of Russia	http://dbpedia.org/page/Olga_Constantinovna_of_Russia
Patrick Bowes-Lyon, 15th Earl of Strathmore and Kinghorne	http://dbpedia.org/page/Patrick_Bowes-Lyon,_15th_Earl_of_Strathmore_and_Kinghorne
Paul I of Russia	http://dbpedia.org/page/Paul_I_of_Russia
Peter Phillips	http://dbpedia.org/page/Peter_Phillips

Nama Tokoh	URL
Prince Adolphus, Duke of Cambridge	http://dbpedia.org/page/Prince_Adolphus,_Duke_of_Cambridge
Prince Albert Victor, Duke of Clarence and Avondale	http://dbpedia.org/page/Prince_Albert_Victor,_Duke_of_Clarence_and_Avondale
Prince Alfred of Great Britain	http://dbpedia.org/page/Prince_Alfred_of_Great_Britain
Prince Andrew, Duke of York	http://dbpedia.org/page/Prince_Andrew,_Duke_of_York
Prince Arthur, Duke of Connaught and Strathearn	http://dbpedia.org/page/Prince_Arthur,_Duke_of_Connaught_and_Strathearn
Prince Augustus Frederick, Duke of Sussex	http://dbpedia.org/page/Prince_Augustus_Frederick,_Duke_of_Sussex
Prince Charles of Hesse-Kassel	http://dbpedia.org/page/Prince_Charles_of_Hesse-Kassel
Prince Christian of Hesse	http://dbpedia.org/page/Prince_Christian_of_Hesse
Prince Eduard of Saxe-Altenburg	http://dbpedia.org/page/Prince_Eduard_of_Saxe-Altenburg
Prince Edward, Duke of Kent and Strathearn	http://dbpedia.org/page/Prince_Edward,_Duke_of_Kent_and_Strathearn
Prince Edward, Duke of York and Albany	http://dbpedia.org/page/Prince_Edward,_Duke_of_York_and_Albany

Nama Tokoh	URL
Prince Edward, Earl of Wessex	http://dbpedia.org/page/Prince_Edward,_Earl_of_Wessex
Prince Ferdinand of Saxe-Coburg and Gotha	http://dbpedia.org/page/Prince_Ferdinand_of_Saxe-Coburg_and_Gotha
Prince Francis of Teck	http://dbpedia.org/page/Prince_Francis_of_Teck
Prince Frederick, Duke of York and Albany	http://dbpedia.org/page/Prince_Frederick,_Duke_of_York_and_Albany
Prince Frederick of Great Britain	http://dbpedia.org/page/Prince_Frederick_of_Great_Britain
Prince Frederick of Hesse-Kassel	http://dbpedia.org/page/Prince_Frederick_of_Hesse-Kassel
Prince Frederik of Hesse	http://dbpedia.org/page/Prince_Frederik_of_Hesse
Prince George, Duke of Cambridge	http://dbpedia.org/page/Prince_George,_Duke_of_Cambridge
Prince George, Duke of Kent	http://dbpedia.org/page/Prince_George,_Duke_of_Kent
Prince George of Cambridge	http://dbpedia.org/page/Prince_George_of_Cambridge
Prince George William of Great Britain	http://dbpedia.org/page/Prince_George_William_of_Great_Britain
Prince Harry	http://dbpedia.org/page/Prince_Harry

Nama Tokoh	URL
Prince Henry, Duke of Cumberland and Strathearn	http://dbpedia.org/page/Prince_Henry,_Duke_of_Cumberland_and_Strathearn
Prince Henry, Duke of Gloucester	http://dbpedia.org/page/Prince_Henry,_Duke_of_Gloucester
Prince John of the United Kingdom	http://dbpedia.org/page/Prince_John_of_the_United_Kingdom
Prince Julius of Schleswig-Holstein-Sonderburg-Glücksburg	http://dbpedia.org/page/Prince_Julius_of_Schleswig-Holstein-Sonderburg-Glücksburg
Prince Karl Anton August of Schleswig-Holstein-Sonderburg-Beck	http://dbpedia.org/page/Prince_Karl_Anton_August_of_Schleswig-Holstein-Sonderburg-Beck
Prince Leopold, Duke of Albany	http://dbpedia.org/page/Prince_Leopold,_Duke_of_Albany
Prince Octavius of Great Britain	http://dbpedia.org/page/Prince_Octavius_of_Great_Britain
Prince Philip, Duke of Edinburgh	http://dbpedia.org/page/Prince_Philip,_Duke_of_Edinburgh
Prince Valdemar of Denmark	http://dbpedia.org/page/Prince_Valdemar_of_Denmark
Prince William, Duke of Cambridge	http://dbpedia.org/page/Prince_William,_Duke_of_Cambridge

Nama Tokoh	URL
Prince William, Duke of Cumberland	http://dbpedia.org/page/Prince_William,_Duke_of_Cumberland
Prince William Henry, Duke of Gloucester and Edinburgh	http://dbpedia.org/page/Prince_William_Henry,_Duke_of_Gloucester_and_Edinburgh
Prince William of Hesse-Kassel	http://dbpedia.org/page/Prince_William_of_Hesse-Kassel
Princess Alexandra of Saxe-Altenburg	http://dbpedia.org/page/Princess_Alexandra_of_Saxe-Altenburg
Princess Alice of the United Kingdom	http://dbpedia.org/page/Princess_Alice_of_the_United_Kingdom
Princess Amelia of Great Britain	http://dbpedia.org/page/Princess_Amelia_of_Great_Britain
Princess Amelia of the United Kingdom	http://dbpedia.org/page/Princess_Amelia_of_the_United_Kingdom
Princess Antoinette of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Princess_Antoinette_of_Saxe-Coburg-Saalfeld
Princess Augusta of Cambridge	http://dbpedia.org/page/Princess_Augusta_of_Cambridge
Princess Augusta of Great Britain	http://dbpedia.org/page/Princess_Augusta_of_Great_Britain

Nama Tokoh	URL
Princess Augusta of Hesse-Kassel	http://dbpedia.org/page/Princess_Augusta_of_Hesse-Kassel
Princess Augusta of Saxe-Gotha	http://dbpedia.org/page/Princess_Augusta_of_Saxe-Gotha
Princess Augusta Sophia of the United Kingdom	http://dbpedia.org/page/Princess_Augusta_Sophia_of_the_United_Kingdom
Princess Beatrice of the United Kingdom	http://dbpedia.org/page/Princess_Beatrice_of_the_United_Kingdom
Princess Beatrice of York	http://dbpedia.org/page/Princess_Beatrice_of_York
Princess Carolina of Orange-Nassau	http://dbpedia.org/page/Princess_Carolina_of_Orange-Nassau
Princess Caroline of Great Britain	http://dbpedia.org/page/Princess_Caroline_of_Great_Britain
Princess Caroline of Nassau-Usingen	http://dbpedia.org/page/Princess_Caroline_of_Nassau-Usingen
Princess Charlotte of Cambridge	http://dbpedia.org/page/Princess_Charlotte_of_Cambridge
Princess Charlotte of Denmark	http://dbpedia.org/page/Princess_Charlotte_of_Denmark
Princess Charlotte of Saxe-Hildburghausen	http://dbpedia.org/page/Princess_Charlotte_of_Saxe-Hildburghausen

Nama Tokoh	URL
Princess Charlotte of Saxe-Meiningen	http://dbpedia.org/page/Princess_Charlotte_of_Saxe-Meiningen
Princess Charlotte Sophie of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Princess_Charlotte_Sophie_of_Saxe-Coburg-Saalfeld
Princess Elisabeth Albertine of Saxe-Hildburghausen	http://dbpedia.org/page/Princess_Elisabeth_Albertine_of_Saxe-Hildburghausen
Princess Elisabeth of Saxe-Altenburg (1826–1896)	http://dbpedia.org/page/Princess_Elisabeth_of_Saxe-Altenburg_(1826–1896)
Princess Elizabeth of Great Britain	http://dbpedia.org/page/Princess_Elizabeth_of_Great_Britain
Princess Elizabeth of the United Kingdom	http://dbpedia.org/page/Princess_Elizabeth_of_the_United_Kingdom
Princess Eugenie of York	http://dbpedia.org/page/Princess_Eugenie_of_York
Princess Friederike of Schleswig-Holstein-Sonderburg-Glücksburg	http://dbpedia.org/page/Princess_Friederike_of_Schleswig-Holstein-Sonderburg-Glücksburg
Princess Helena of the United Kingdom	http://dbpedia.org/page/Princess_Helena_of_the_United_Kingdom
Princess Henriette of Nassau-Weilburg	http://dbpedia.org/page/Princess_Henriette_of_Nassau-Weilburg
Princess Juliane of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Princess_Juliane_of_Saxe-Coburg-Saalfeld

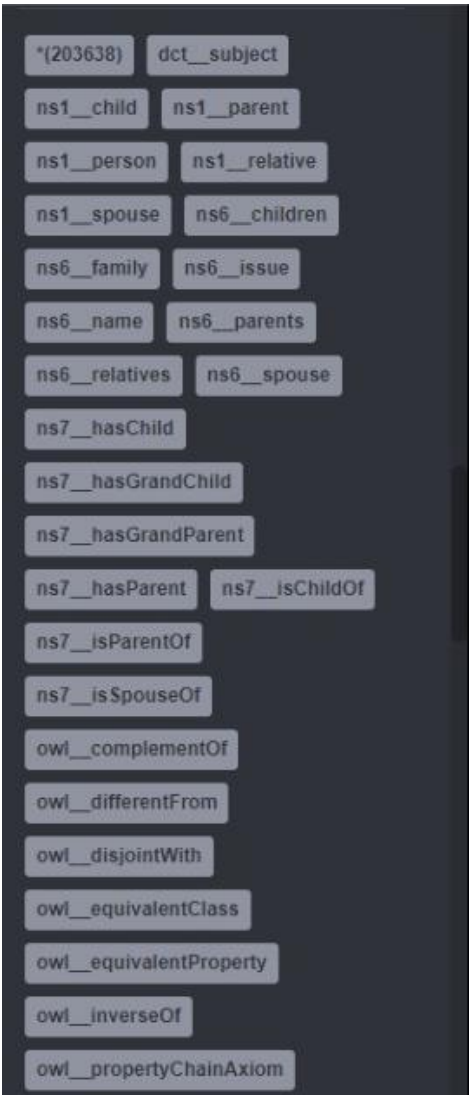
Nama Tokoh	URL
Princess Louisa of Great Britain	http://dbpedia.org/page/Princess_Louisa_of_Great_Britain
Princess Louise, Duchess of Argyll	http://dbpedia.org/page/Princess_Louise,_Duchess_of_Argyll
Princess Louise Caroline of Hesse-Kassel	http://dbpedia.org/page/Princess_Louise_Caroline_of_Hesse-Kassel
Princess Louise of Denmark (1750–1831)	http://dbpedia.org/page/Princess_Louise_of_Denmark_(1750–1831)
Princess Louise of Saxe-Gotha-Altenburg (1756–1808)	http://dbpedia.org/page/Princess_Louise_of_Saxe-Gotha-Altenburg_(1756–1808)
Princess Louise of Saxe-Gotha-Altenburg (1800–1831)	http://dbpedia.org/page/Princess_Louise_of_Saxe-Gotha-Altenburg_(1800–1831)
Princess Louise of Saxe-Hildburghausen	http://dbpedia.org/page/Princess_Louise_of_Saxe-Hildburghausen
Princess Margaret, Countess of Snowdon	http://dbpedia.org/page/Princess_Margaret,_Countess_of_Snowdon
Princess Marie Auguste of Thurn and Taxis	http://dbpedia.org/page/Princess_Marie_Auguste_of_Thurn_and_Taxis
Princess Marie Luise Charlotte of Hesse-Kassel	http://dbpedia.org/page/Princess_Marie_Luise_Charlotte_of_Hesse-Kassel

Nama Tokoh	URL
Princess Mary Adelaide of Cambridge	http://dbpedia.org/page/Princess_Mary_Adelaide_of_Cambridge
Princess Mary of Great Britain	http://dbpedia.org/page/Princess_Mary_of_Great_Britain
Princess Sophia Dorothea of Prussia	http://dbpedia.org/page/Princess_Sophia_Dorothea_of_Prussia
Princess Sophia of the United Kingdom	http://dbpedia.org/page/Princess_Sophia_of_the_United_Kingdom
Princess Sophie Antoinette of Brunswick-Wolfenbüttel	http://dbpedia.org/page/Princess_Sophie_Antoinette_of_Brunswick-Wolfenbüttel
Princess Sophie of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Princess_Sophie_of_Saxe-Coburg-Saalfeld
Princess Thyra of Denmark	http://dbpedia.org/page/Princess_Thyra_of_Denmark
Princess Victoria of Saxe-Coburg-Saalfeld	http://dbpedia.org/page/Princess_Victoria_of_Saxe-Coburg-Saalfeld
Princess Victoria of the United Kingdom	http://dbpedia.org/page/Princess_Victoria_of_the_United_Kingdom
Princess Wilhelmina Caroline of Denmark	http://dbpedia.org/page/Princess_Wilhelmina_Caroline_of_Denmark

Nama Tokoh	URL
Queen Elizabeth The Queen Mother	http://dbpedia.org/page/Queen_Elizabeth_The_Queen_Mother
Queen Victoria	http://dbpedia.org/page/Queen_Victoria
Ronald Ferguson	http://dbpedia.org/page/Ronald_Ferguson
Rose Leveson-Gower, Countess Granville	http://dbpedia.org/page/Rose_Leveson-Gower,_Countess_Granville
Sarah, Duchess of York	http://dbpedia.org/page/Sarah,_Duchess_of_York
Savannah Phillips	http://dbpedia.org/page/Savannah_Phillips
Sophia Dorothea of Celle	http://dbpedia.org/page/Sophia_Dorothea_of_Celle
Sophia Dorothea of Hanover	http://dbpedia.org/page/Sophia_Dorothea_of_Hanover
Sophia Magdalena of Denmark	http://dbpedia.org/page/Sophia_Magdalena_of_Denmark
Sophie, Countess of Wessex	http://dbpedia.org/page/Sophie,_Countess_of_Wessex
Susan Barrantes	http://dbpedia.org/page/Susan_Barrantes
Therese of Saxe-Hildburghausen	http://dbpedia.org/page/Therese_of_Saxe-Hildburghausen
Thomas Lyon-Bowes, Lord Glamis	http://dbpedia.org/page/Thomas_Lyon-Bowes,_Lord_Glamis
Timothy Laurence	http://dbpedia.org/page/Timothy_Laurence

Nama Tokoh	URL
Victoria, Princess Royal	http://dbpedia.org/page/Victoria,_Princess_Royal
William I, Elector of Hesse	http://dbpedia.org/page/William_I,_Elector_of_Hesse
William IV, Prince of Orange	http://dbpedia.org/page/William_IV,_Prince_of_Orange
William IV of the United Kingdom	http://dbpedia.org/page/William_IV_of_the_United_Kingdom
William V, Prince of Orange	http://dbpedia.org/page/William_V,_Prince_of_Orange
Zara Phillips	http://dbpedia.org/page/Zara_Phillips

Lampiran 1 Gambar Sebelum Dilakukan *Reasoning*



Lampiran 2 Gambar Hasil Pengujian *Reasoning* R-01

Lampiran 3 Gambar Hasil Pengujian *Reasoning* R-02

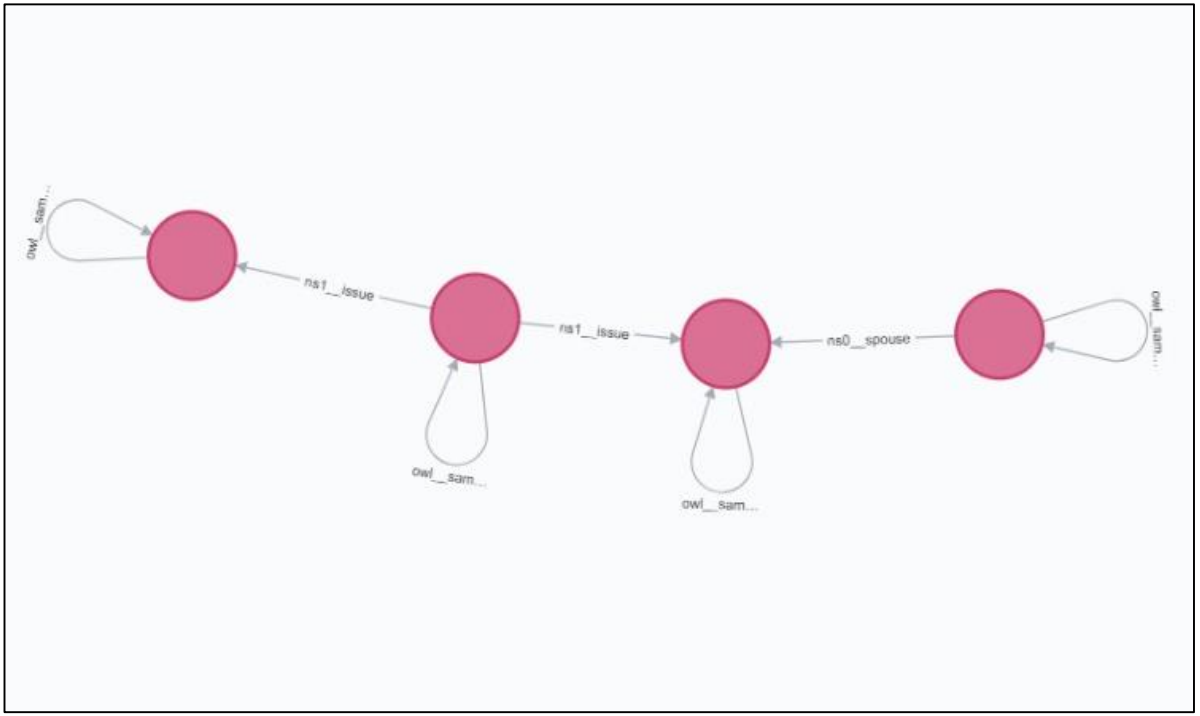
Lampiran 4 Gambar Hasil Pengujian *Reasoning* R-03

Lampiran 5 Gambar Hasil Pengujian *Reasoning* R-04

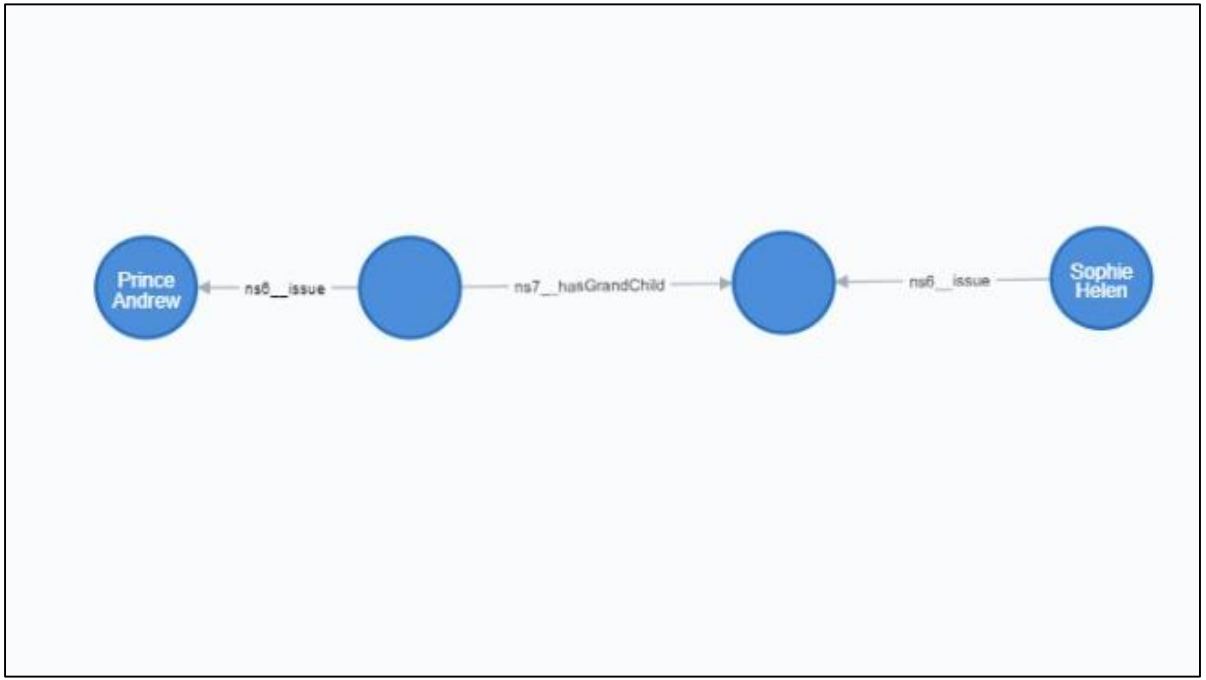
Lampiran 6 Gambar Hasil Pengujian *Reasoning* R-05

Lampiran 7 Gambar Hasil Pengujian *Reasoning* R-06

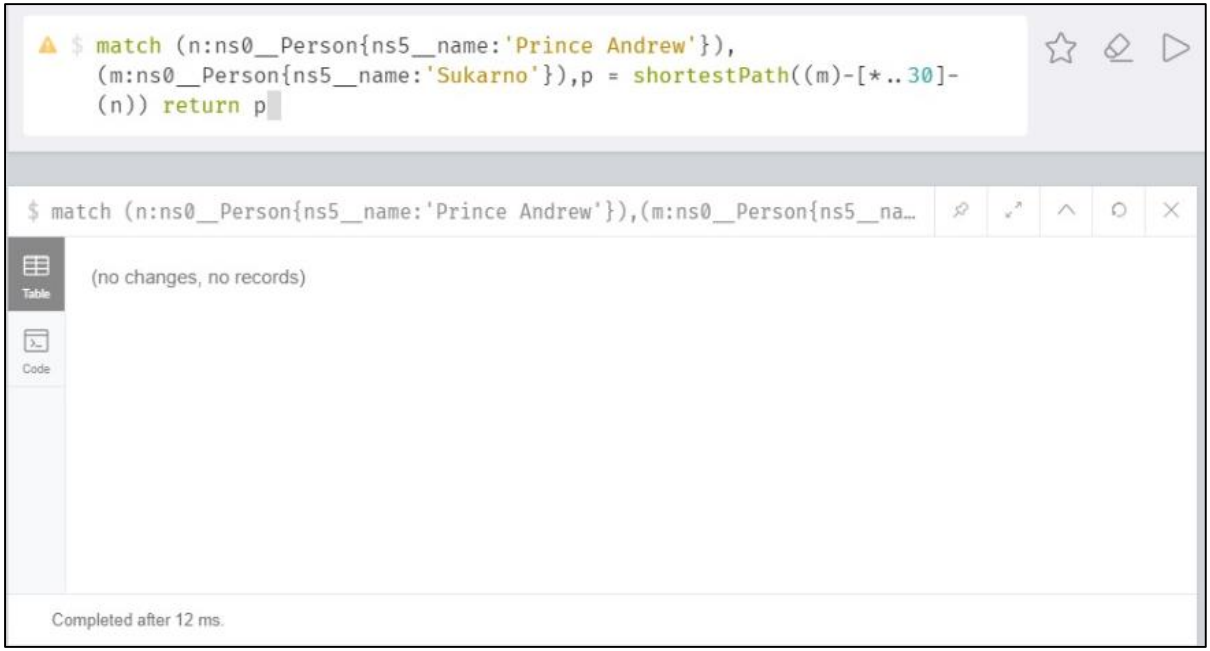
Lampiran 8 Gambar Hasil Pengujian *Reasoning* R-07



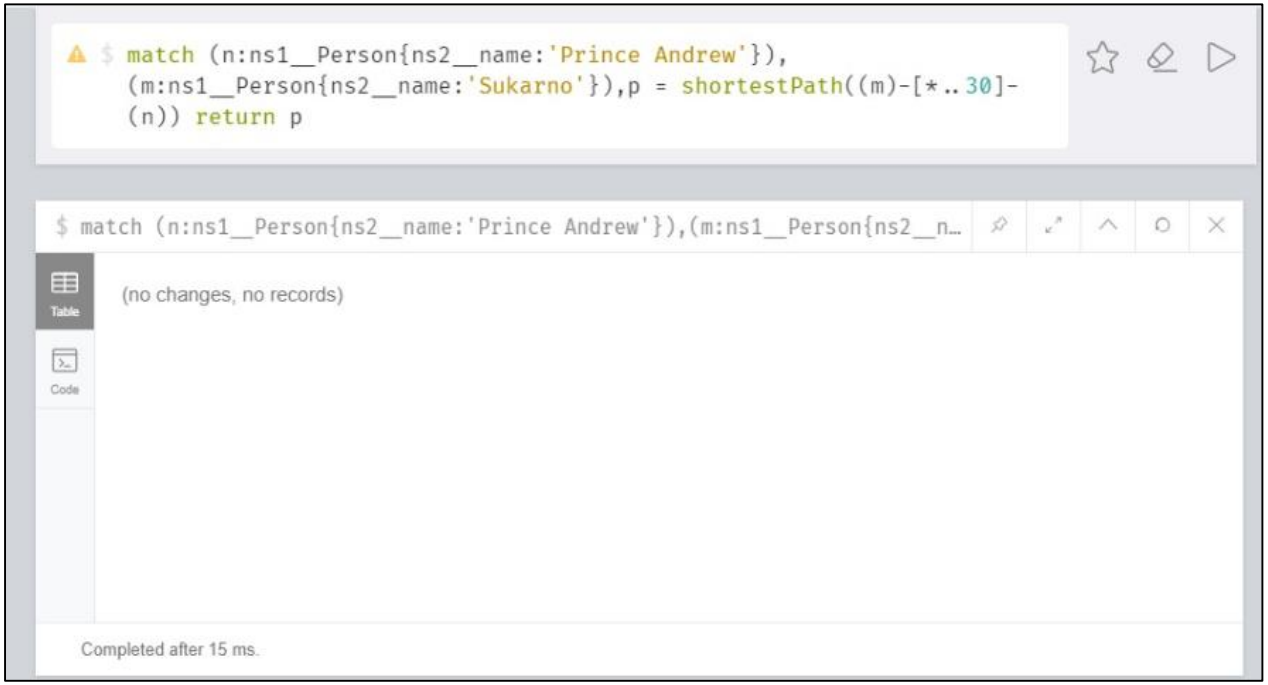
Lampiran 9 Relasi antar Dua Tokoh Sebelum Dilakukan *Reasoning*



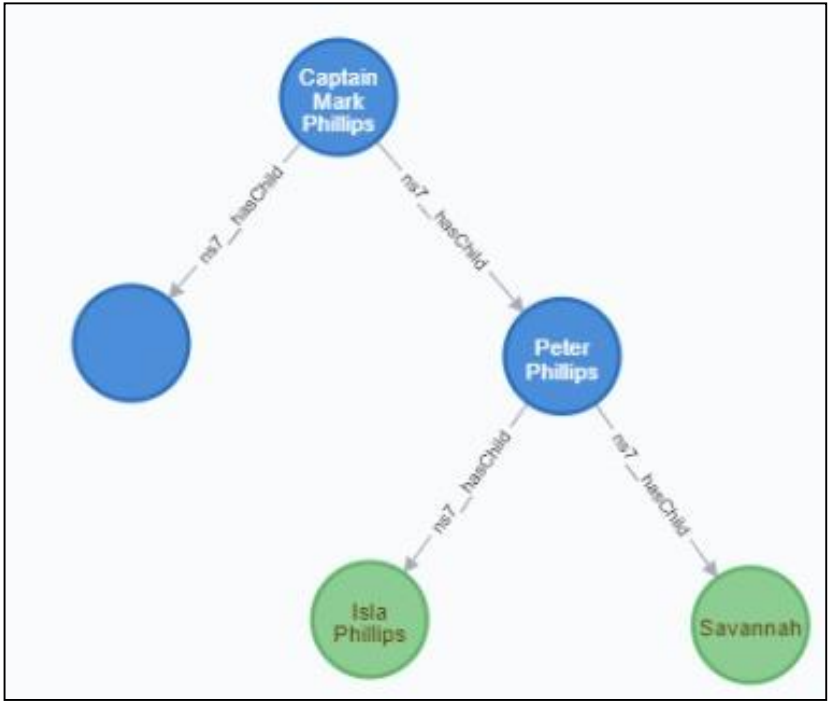
Lampiran 10 Relasi antar Dua Tokoh Setelah Dilakukan *Reasoning*



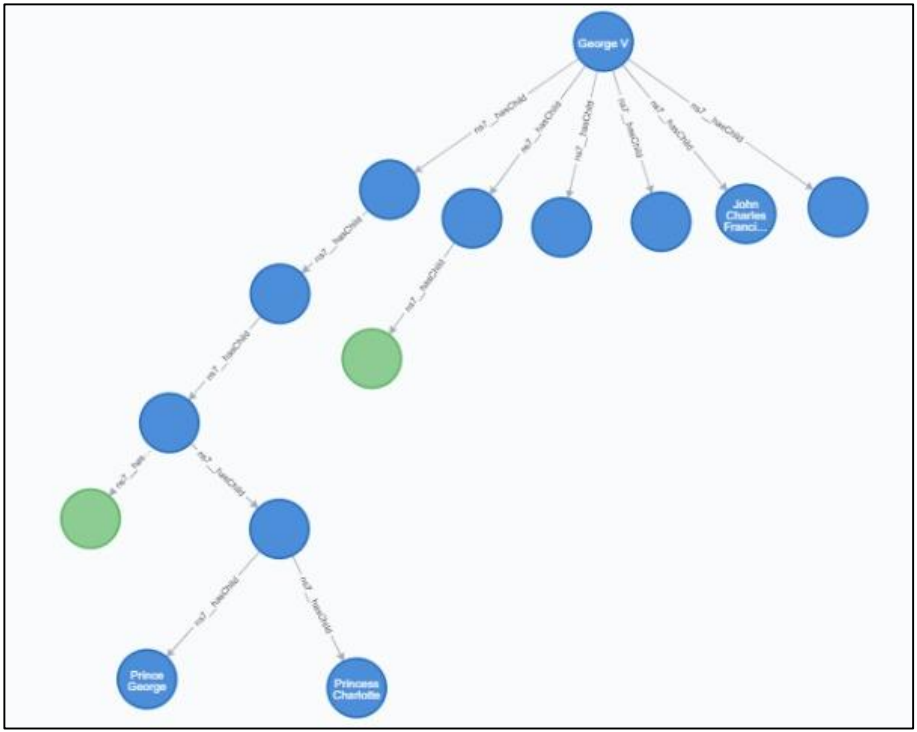
Lampiran 11 Tidak adanya Relasi antar Dua Tokoh Sebelum Dilakukan *Reasoning*



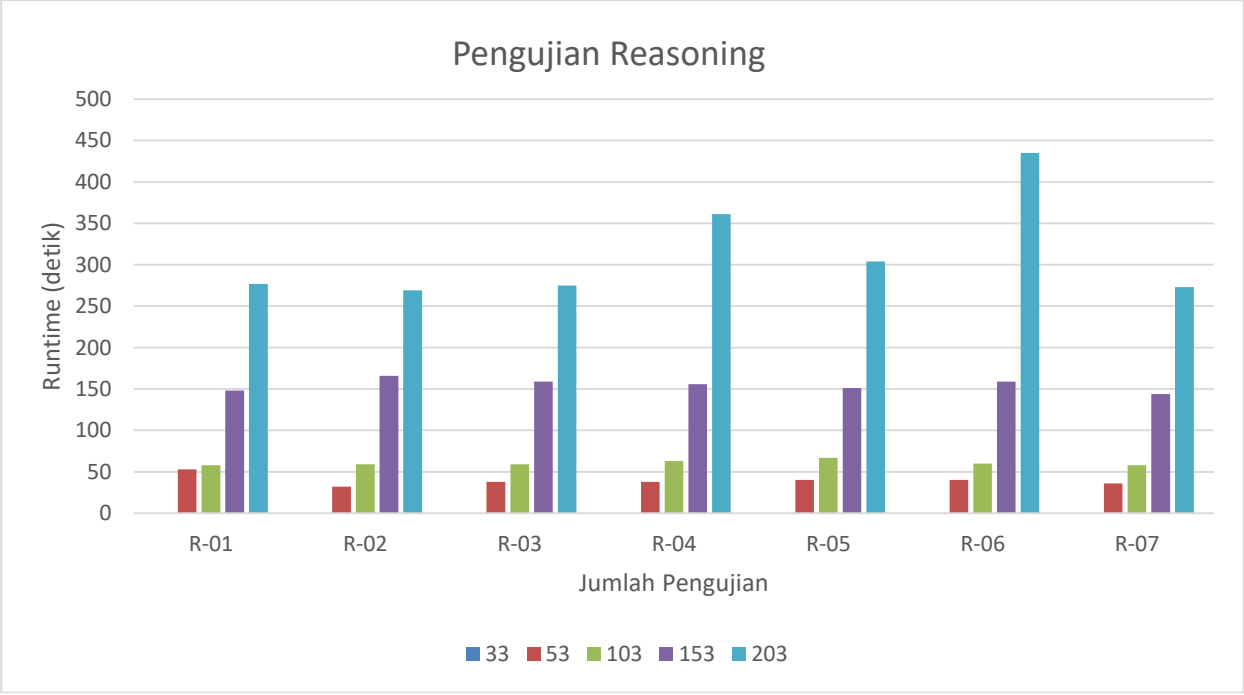
Lampiran 12 Tidak adanya Relasi antar Dua Tokoh Setelah Dilakukan *Reasoning*



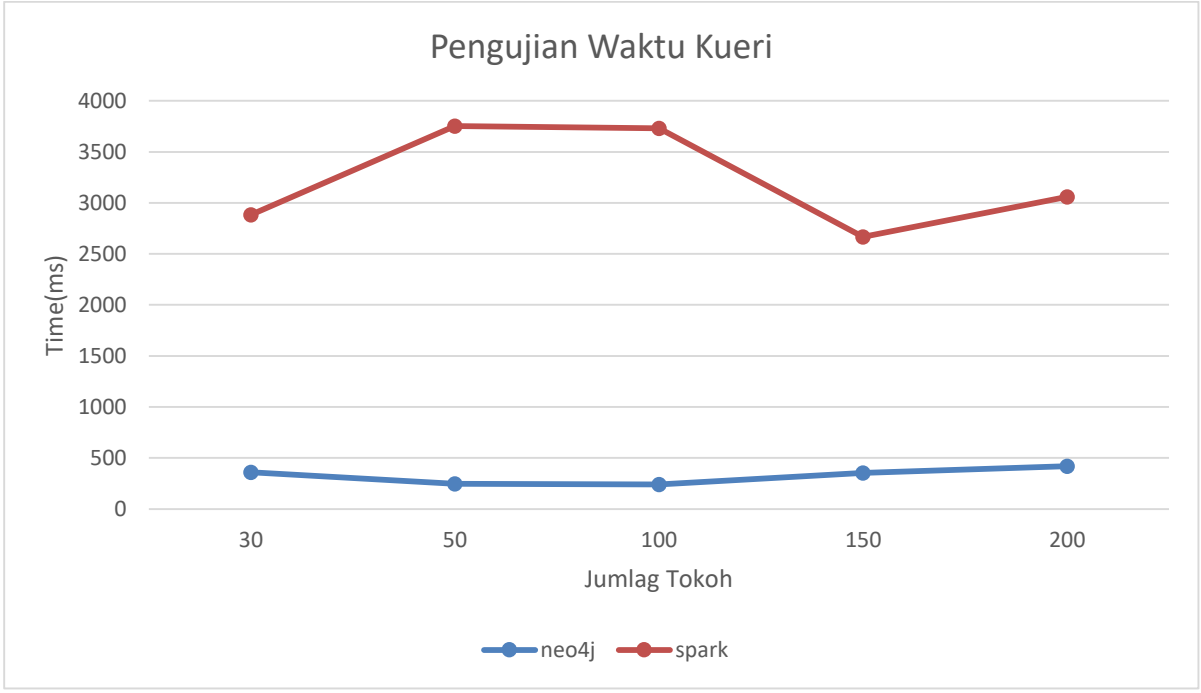
Lampiran 13 Hasil Pengujian Pencarian Pohon Keluarga Seorang Tokoh Mark Phillips



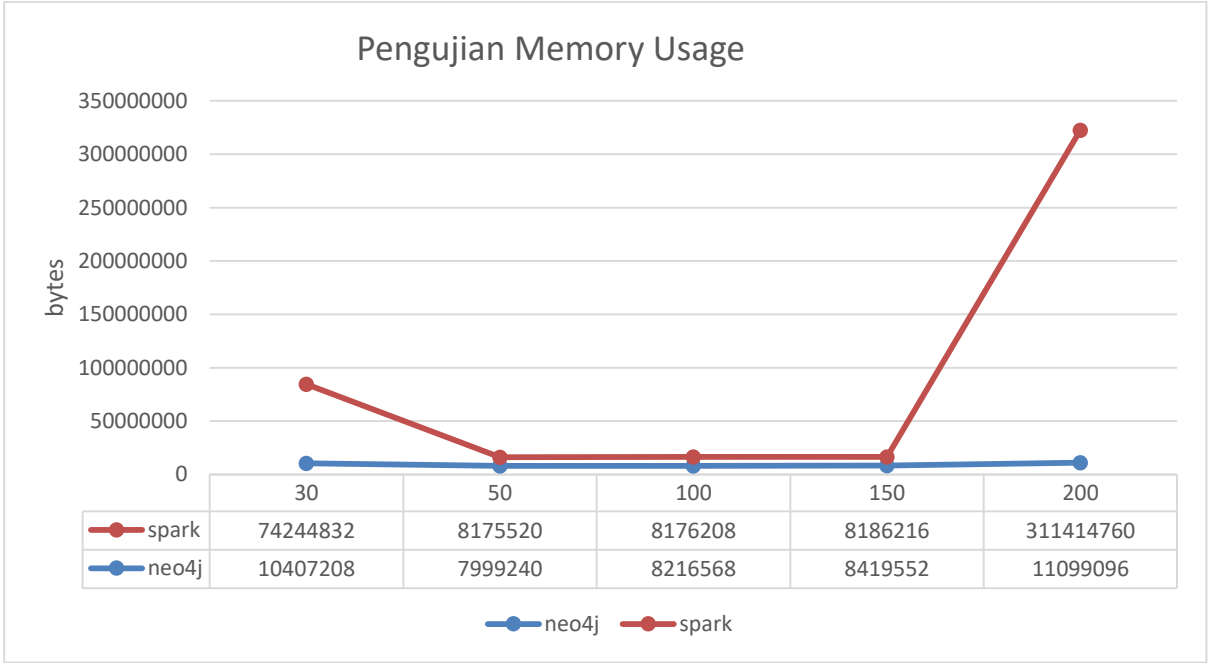
Lampiran 14 Hasil Pengujian Pencarian Pohon Keluarga Seorang Tokoh George V



Lampiran 15 Grafik Perbandingan waktu pengujian reasoning



Lampiran 16 Grafik Perbandingan Waktu Kueri



Lampiran 17 Grafik Perbandingan Penggunaan Memori

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Nuzul Ayu Safitri, lahir di Kebumen, 24 Januari 1997. Penulis menempuh pendidikan mulai dari SD Negeri 3 Kedawung, Pejagoan, Kebumen, Jawa Tengah (2003-2009), SMP Negeri 3 Kebumen, Jawa Tengah (2009-2012), SMA Negeri 1 Kebumen, Jawa Tengah (2012-2015), dan sekarang menempuh pendidikan S1 Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember, Surabaya. Selama perkuliahan, penulis aktif dalam organisasi HMTC. Diantaranya menjadi staff Pengembangan Profesi HMTC.

Dalam menyelesaikan pendidikan S1 penulis mengambil bidang minat Manajemen Informasi (MI). Di sisi profesionalitas, penulis pernah melakukan kerja praktek di Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI) – ITS. Penulis dapat dihubungi melalui alamat *email* nuzulayusafitri@gmail.com.