TUGAS AKHIR – KI141502

PENYIMPANAN DAN PENG-QUERY-AN BASIS DATA RELASIONAL BERBASIS ONTOLOGI UNTUK PERMASALAHAN POHON KELUARGA

KAMALI YAHYA
NRP. 5112 100 107

Dosen Pembimbing 1
Sarwosri, S.Kom., M.T.

Dosen Pembimbing 2
Nurul Fajrin A.,S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

PENYIMPANAN DAN PENG-QUERY-AN BASIS DATA RELASIONAL BERBASIS ONTOLOGI UNTUK PERMASALAHAN POHON KELUARGA

KAMALI YAHYA
NRP. 5112 100 107

Dosen Pembimbing 1
Sarwosri, S.Kom., M.T.

Dosen Pembimbing 2
Nurul Fajrin A.,S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – KI141502

STORING AND QUERYING RELATIONAL DATABASE BASED ON ONTOLOGY FOR FAMILY TREE

KAMALI YAHYA
NRP. 5112 100 107

Supervisor 1
Sarwosri, S.Kom., M.T.

Supervisor 2
Nurul Fajrin A.,S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PENYIMPANAN DAN PENG-QUERY-AN BASIS DATA RELASIONAL BERBASIS ONTOLOGI UNTUK PERMASALAHAN POHON KELUARGA

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
KAMALI YAHYA
NRP. 5112 100 107

Disetujui oleh Pembimbing Tugas Akhir:

1. Sarwosri, S.Kom., M.T. .....
   NIP. 19760809 200112 2 001 (Pembimbing 1)
2. Nurul Fajrin A., S.Kom., M.Sc. .....
   NIP. 19860722 201504 2 003 (Pembimbing 2)

SURABAYA
JANUARI, 2017

(Halaman ini sengaja dikosongkan)

PENYIMPANAN DAN PENG-QUERY-AN BASIS DATA RELASIONAL BERBASIS ONTOLOGI UNTUK PERMASALAHAN POHON KELUARGA

Nama : **Kamali Yahya**
NRP : **5112100107**
Jurusan : **Teknik Informatika**
Fakultas Teknologi Informasi ITS
Dosen Pembimbing I : **Sarwosri, S.Kom., M.T.**
Dosen Pembimbing II : **Nurul Fajrin A.,S.Kom., M.Sc.**

ABSTRAK

Pohon keluarga adalah struktur/susunan keluarga yang ada kaitannya dengan orang lain yang menjadi istri/suaminya dan sanak keluarganya, pohon keluarga tersebut merupakan susunan keluarga dari atas ke bawah dan ke samping dengan menyebut nama keluarganya, pohon keluarga tersebut dibuat menyerupai pohon dengan akar-akarnya atau cabang-cabangnya. Batangnya adalah "saya", cabangnya adalah orang tua, kakek, nenek dan seterusnya keatas. Sedangkan untuk akar-akarnya adalah anak, cucu dan seterusnya ke bawah.

Data Ontologi sebuah pohon keluarga biasanya hanya dapat disimpan dalam bentuk file saja, dan untuk melakukan pencarian data terhadap data ontologi tersebut membutuhkan waktu yang cukup lama. Hal tersebut tidak layak untuk diimplementasikan jika jumlah datanya sangat banyak. Oleh dari itu, maka melalui penulisan ini, akan dilakukan uji coba tentang penyimpanan dan peng-query-an basis data relasional berbasis ontologi untuk permasalahan pohon keluarga, dengan PostgreSQL dan Protégé dengan plugin OWL2RDB dan Pellet sebagai tools yang digunakan, dan juga dengan menggunakan SQL dan SPARQL sebagai bahasa untuk query.

Data awal yang dipakai adalah berupa ontologi hasil invert axioms, data tersebut disimpan ke dalam PostgreSQL dengan dengan menggunakan plugin Protégé berupa OWL2RDB.

Sedangkan untuk peng-query-an (query SPARQL) dibutuhkan sebuah aplikasi sederhana yang bisa memetakan syntax query SPARQL menjadi sebuah syntax query SQL, dan menampilkan data hasil pengolahan query tersebut pada kolom hasil.

Berdasarkan uji coba yang telah dilakukan, penyimpanan ontologi hasil transformasi OWL2RDB sebuah ontologi pohon keluarga berhasil disimpan pada database PostgreSQL menjadi beberapa tabel beserta isinya. Sedangkan untuk pengujian peng-query-an data, semua case yang telah dibangun pada aplikasi ini berhasil menampilkan suatu data hasil query SPARQL seperti yang seharusnya (sama dengan hasil SPARQL Engine Protégé). Dan untuk waktu pemrosesan peng-query-an suatu data pada aplikasi yang dibangun dan pada aplikasi Protégé memberikan hasil bahwa waktu pemrosesan peng-query-an suatu data pada aplikasi yang dibangun lebih cepat atau lebih kecil daripada waktu pemrosesan peng-query-an suatu data pada Protégé

Diharapkan dengan dilakukannya pengerjaan Tugas Akhir ini, maka akan mempercepat seseorang untuk melakukan pencarian data sebuah ontologi pohon keluarga.

Kata kunci: PostgreSQL, Protégé, OWL2RDB, Ontologi, Basis Data Relasional, OWL 2, Query SQL, Query SPARQL.

STORING AND QUERYING RELATIONAL DATABASE BASED ON ONTOLOGY FOR FAMILY TREE

Name	**: Kamali Yahya**
NRP	**: 5112100038**
Department	**: Department of Informatics Faculty of Information Technology ITS**
Supervisor I	**: Sarwosri, S.Kom., M.T.**
Supervisor II	**: Nurul Fajrin A.,S.Kom., M.Sc.**

ABSTRACT

The family's tree is the structure of the family that consist of husband, wife and relatives in which they are interrelated one another. This tree can be described as the arrangement of the family from the various angle because each part has its function. The family's tree is made resemble as tree that consist of rod, branch and root. The rod is the 'me', the branches are the parents, grandparents and forth above. As the roots are children, grandchildren and forth below.

Ontology data of family's tree is generally stored in a file and in order to look for that data, it takes several time due to the form of that data is in the text. Additionally, when the data is huge, it cannot be implemented. Therefore through this paper, it will be tested on the storing and querying relational database based on ontology for family tree by using PostgreSQL and Protégé with plugin OWL2RDB and Pellet as a tool are used and using SQL and SPARQL as the language for querying.

The preliminary data that used is in the form of ontology results of invert axioms, this data is stored in PostgreSQL using plugin Protégé OWL2RDB. Whereas in the query system (query SPARQL), it required a simple application that charted SPARQL

query syntax into SQL query syntax and it can display the result of query processing in the result column.

Based on the trials that was conducted, ontology storage that resulted the transformation of OWL2RDB the ontology family tree succesfully saved in PostgreSQL database into multiple tables and their contents. As for testing for the data query, all the cases which was built on this application has successfully displayed a result of SPARQL data query as it should be (the same as the results of SPARQL Engine Protage). And time for processing the data query in the applications built and on a protégé applications gave the result that processing time of the data query in applications built is faster or smaller than the processing time of the data query on the Protege.

Lastly, it is expected by doing this final project, it will accelerate to seek the ontology data of family's tree.

Keywords: PostgreSQL, Protégé, OWL2RDB, Ontology, Relational Database, OWL 2, SQL Query, SPARQL Query.

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT karena atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"Penyimpanan dan Peng-*query*-an Basis Data Relasional Berbasis Ontologi untuk Permasalahan Pohon Keluarga"

Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama proses pengerjaan Tugas Akhir ini hingga selesai, antara lain:

1. Allah SWT atas segala karunia dan rahmat-Nya yang telah diberikan selama ini.
2. Orang tua, saudara serta keluarga penulis yang tiada henti-hentinya memberikan semangat, perhatian dan doa selama perkuliahan penulis di Jurusan Teknik Informatika ini.
3. Ibu Sarwosri, S.Kom., M.T. selaku dosen pembimbing I yang telah memberikan bimbingan dan arahan dalam pengerjaan Tugas Akhir ini.
4. Ibu Nurul Fajrin A., S.Kom., M.Sc. selaku dosen pembimbing II yang telah banyak memberikan arahan dan bantuan, waktu untuk berdiskusi serta ilmu-ilmu baru sehingga penulis dapat menyelesaikan Tugas Akhir ini.
5. Ibu Dr. Chastine Fatichah,S.Kom.,M.Kom. selaku dosen wali yang telah mengarahkan penulis, serta segenap dosen Teknik Informatika ITS yang telah memberikan ilmunya.

6. Teman diskusi penulis, Alief, Bakhtiar, Madis, dan Wimba yang bersedia meluangkan waktunya untuk berbagi ilmu dengan penulis.
7. Seluruh keluarga TC 2012 yang selalu menemani dan memberi semangat selama 4,5 tahun perkuliahan.
8. Serta semua pihak yang telah memberikan dukungan selama penulis menyelesaikan Tugas Akhir ini.

Penulis mohon maaf apabila terdapat kekurangan dalam penulisan buku Tugas Akhir ini. Kritik dan saran penulis harapkan untuk perbaikan dan pembelajaran di kemudian hari. Semoga Tugas Akhir ini dapat memberikan manfaat yang sebaik-baiknya.

Surabaya, Januari 2017

Penulis

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan hal-hal yang menjadi latar belakang, permasalahan yang dihadapi, batasan masalah, tujuan, metodologi dan sistematika penulisan yang digunakan dalam pembuatan buku Tugas Akhir ini.

1.1. Latar Belakang

Pohon keluarga adalah struktur/susunan keluarga yang ada kaitannya dengan orang lain yang menjadi istri/suaminya dan sanak keluarganya, pohon keluarga tersebut merupakan susunan keluarga dari atas ke bawah dan ke samping dengan menyebut nama keluarganya, pohon keluarga tersebut dibuat menyerupai pohon dengan akar-akarnya atau cabang-cabangnya. Batangnya adalah "saya", cabangnya adalah orang tua, kakek, nenek dan seterusnya ke atas. Sedangkan untuk akar-akarnya adalah anak, cucu dan seterusnya ke bawah.

Data Ontologi sebuah pohon keluarga biasanya hanya dapat disimpan dalam bentuk *file* saja, dan untuk melakukan pencarian data terhadap data ontologi tersebut membutuhkan waktu yang cukup lama. Hal tersebut tidak layak untuk diimplementasikan jika jumlah datanya sangat banyak. Oleh dari itu, maka melalui penulisan ini, akan dilakukan uji coba tentang penyimpanan dan peng-*query*-an basis data relasional berbasis ontologi untuk permasalahan pohon keluarga, dengan PostgreSQL dan Protégé dengan *plugin* OWL2RDB dan Pellet sebagai *tools* yang digunakan, dan juga dengan menggunakan SQL dan SPARQL sebagai bahasa untuk *query*.

Langkah yang dilakukan adalah dengan melakukan normalisasi terlebih dahulu terhadap data ontologi suatu pohon keluarga yang sudah ada. Kemudian melakukan *reasoning* terhadap data ontologi yang telah dinormalisasi tersebut, data hasil *reasoning* tersebut lalu disimpan menjadi ontologi baru dengan memanfaatkan fitur "Export inferred axioms as ontology". Data

ontologi baru inilah yang nantinya akan ditransformasikan ke dalam *Database Management System* (DBMS) PostgreSQL dengan menggunakan *plugin* OWL2RDB. Setelah data tersimpan dalam PostgreSQL, maka akan dibangun suatu aplikasi yang bisa memetakan suatu *syntax* SPARQL menjadi *syntax* SQL, dan menampilkan hasil pemrosesan *syntax* tersebut menjadi sebuah data.

Diharapkan dengan dilakukannya pengerjaan Tugas Akhir ini, maka akan mempercepat seseorang untuk melakukan pencarian data sebuah ontologi pohon keluarga.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana skema yang diterapkan untuk melakukan penyimpanan dan peng-*query*-an basis data relasional berbasis ontologi untuk permasalahan pohon keluarga (*Family Tree)* tersebut?
2. Bagaimana cara penyimpanan ontologi suatu pohon keluarga tersebut ke dalam basis data relasional (PostgreSQL)?
3. Bagaimana cara untuk melakukan peng-*query*-an basis data relasional suatu pohon keluarga tersebut (*query* SPARQL)?
4. Bagaimana perbandingan waktu pemrosesan suatu *query* SPARQL pada aplikasi yang dibangun dan pada Protégé?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. *Tools* yang digunakan adalah Netbeans, PostgreSQL 9.5, dan Protégé 4.3 dengan *plugin* OWL2RDB dan Pellet.
2. Aplikasi yang dibuat tidak menyediakan *form* untuk pengelolaan data (tambah, ubah, hapus).
3. Aplikasi yang dibuat tidak menyediakan fitur *reasoner*.

4. Aplikasi yang dibangun hanya untuk melakukan peng-*query*-an data pohon keluarga tersebut menggunakan *query* SPARQL.
5. Pada klausa *where* dalam peng-*query*-an, predikat merupakan sebuah *data*/*object property*.
6. Jumlah klausa *where* maksimal adalah 1 klausa.
7. Aplikasi ini hanya bisa digunakan untuk data pohon keluarga yang telah ditentukan.

1.4. Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah membuat aplikasi sederhana yang mengimplementasikan peng-*query*-an suatu basis data relasional yang berbasis ontologi dengan menggunakan *query* SPARQL untuk membantu mempercepat dan mempermudah pencarian sebuah data pohon keluarga.

1.5. Metodologi

Ada beberapa tahapan dalam pengerjaan Tugas Akhir ini, yaitu sebagai berikut:

1. Studi Literatur
   Pada tahap ini, akan dilakukan studi mengenai sejumlah referensi yang diperlukan dalam pembuatan aplikasi yaitu mengenai OWL, *Family Relationships Ontology*, SWRL (*Semantic Web Rule Language*), Pellet *Reasoner*, Basis Data Rasional, *Query* SQL, dan *Query* SPARQL.
2. Implementasi
   Pada tahap ini, akan dilakukan implementasi berdasarkan rancangan yang dibuat dalam tahap sebelumnya, yaitu melakukan *query* SPARQL terhadap data ontologi yang sudah ditransformasikan menjadi basis data relasional dan disimpan pada PostgreSQL menggunakan OWL2RDB. Proses tersebut bisa dilakukan dengan cara memetakan *syntax query* SPARQL menjadi *syntax query* SQL. Aplikasi ini dibangun dengan bahasa java menggunakan *tools* NetBeans IDE 8.1.
3. Pengujian dan evaluasi

Tahap ini dilakukan untuk uji coba penyimpanan data ontologi pada PostgreSQL, dengan melihat tabel-tabel yang berhasil dibangun dan isi dari tabel-tabel tersebut. Dan uji coba peng-*query*-an data semua *Case* yang ada pada aplikasi yang dibangun. Dan juga uji coba waktu eksekusi dari beberapa *Case* yang telah dibangun. Evaluasi dilakukan untuk mengetahui hasil penyimpanan, dan juga jalannya sebuah program atas sebuah *syntax query* SPARQL yang diberikan.

4. Penyusunan buku Tugas Akhir
   Tahap ini merupakan tahap penyusunan laporan berupa buku sebagai dokumentasi pengerjaan Tugas Akhir yang mencakup seluruh dasar teori, desain, implementasi serta hasil pengujian yang telah dilakukan.

1.6. Sistematika Penulisan

Sistematika penulisan dibuat bertujuan untuk mendapatkan gambaran umum dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan
Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori
Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah
Bab ini membahas mengenai metode yang digunakan untuk memecahkan masalah yang dipaparkan pada rumusan permasalahan.

Bab IV Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan data, perancangan arsitektur, perancangan sistem, proses dan perancangan antarmuka pada perangkat lunak.

Bab V Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak dan implementasi fitur-fitur penunjang.

Bab VI Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian 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.

Bab VII Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Bab ini akan membahas mengenai dasar teori dan literatur yang menjadi dasar pengerjaan Tugas Akhir ini.

2.1. Genealogi

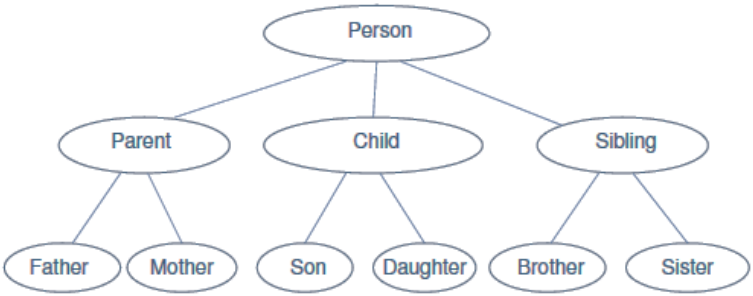
Menurut Kamus Besar Bahasa Indonesia (KBBI), genealogi adalah garis keturunan manusia dalam hubungan keluarga sedarah [1]. Sedangkan menurut sumber lain genealogi adalah "keturunan" dan logos "pengetahuan" adalah kajian tentang keluarga dan penelusuran jalur keturunan serta sejarahnya. Ahli genealogi menggunakan berita dari mulut ke mulut, catatan sejarah, analisis genetik, serta rekaman lain untuk mendapatkan informasi mengenai suatu keluarga dan menunjukkan kekerabatan dan silsilah dari anggota-anggotanya. Hasilnya sering ditampilkan dalam bentuk bagan (disebut bagan silsilah) atau ditulis dalam bentuk narasi. Beberapa ahli membedakan antara genealogi dan sejarah keluarga dan membatasi genealogi hanya pada hubungan perkerabatan, sedangkan "sejarah keluarga" merujuk pada penyediaan detail tambahan mengenai kehidupan dan konteks sejarah keluarga tersebut [2].

2.2. Ontologi

Ontologi adalah spesifikasi eksplisit dari sebuah konseptualisasi. Ketika pengetahuan dari suatu domain direpresentasikan dalam deklarasi formal, satu *set* objek yang terdapat dalam deklarasi tersebut dapat dikatakan sebagai informasi yang *universal*. Objek-objek ini memiliki deskripsi masing-masing dan saling berhubungan satu sama lain yang merefleksikan *knowledge*. Dalam ontologi, informasi yang ditanamkan menggambarkan nama-nama entitas (kelas, relasi, fungsi, dan objek lain) yang dapat dimengerti oleh manusia. Secara formal, ontologi adalah pernyataan dari teori logis [3]. Ontologi digunakan untuk menangkap pengetahuan tentang

beberapa domain yang menarik. Konsep dalam suatu domain tertentu dan hubungan antar konsep-konsep tersebut dapat digambarkan dengan ontologi [4]. Menurut Noy dan McGuinness [5] sebuah domain permasalahan dapat dimodelkan dengan banyak cara, dengan kata lain terdapat berbagai cara untuk membangun sebuah ontologi.

Terdapat beberapa komponen dasar dari ontologi, yaitu *classes* (direpresentasikan dalam susunan taksonomi), *relations* (digunakan untuk menghubungkan antar domain), *axioms* (digunakan untuk memodelkan pernyataan yang selalu benar), *properties* (digunakan untuk menjelaskan karakteristik umum *instance* dari sebuah kelas atau untuk menghubungkan antar *class*) dan *instance* (representasi data spesifik). Ontologi mampu memodelkan data dari suatu domain yang spesifik. Ontologi juga mempunyai fitue *inferences* yang dapat menemukan informasi tersembunyi atau fakta baru di dalam model [6].

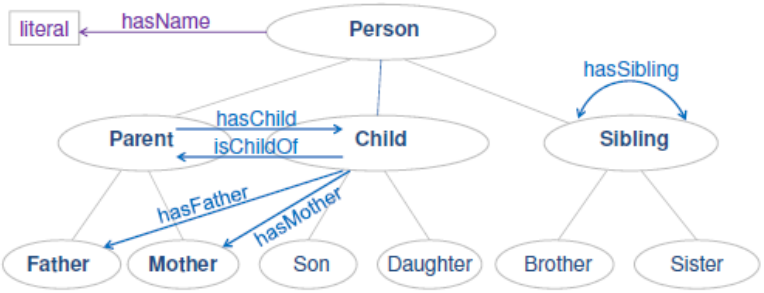


Gambar 2. 1 *Class Hierarchy*
(Sumber : *Introduction to Ontology Concepts and Terminology, Steven J. Miller, 2013)*

Dari beberapa komponen dasar yang telah disebutkan di paragraf sebelumnya, akan dijelaskan mengenai beberapa komponen yang penting dalam pengerjaan Tugas Akhir ini secara lebih spesifik. *Class* menspesifikasikan *property* yang sama dari beberapa *instance* dan mendukung klasifikasi hierarkikal. Selain

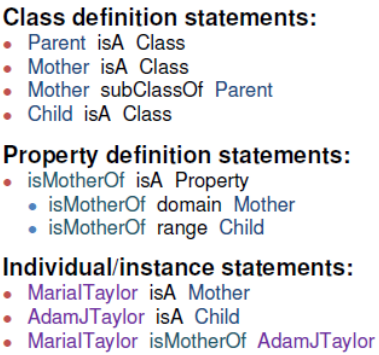
itu, *class* juga mencakup *superclass* dan *subclass*. *Subclass* merupakan level yang lebih khusus dari *superclass*nya. Setiap *subclass* mewarisi operasi dan atribut dari *superclass*nya. *Subclass* mungkin memiliki operasi dan atributnya sendiri (yang tidak dimiliki oleh *superclass*nya). Hubungan antara *subclass* dan *superclass* digambarkan dengan *class hierarchy* seperti yang dicontohkan oleh Miller [7] pada Gambar 2. 1.

Ontologi mendefinisikan sekumpulan *property* yang digunakan dalam domain pengetahuan khusus. Dalam konteks ontologi, *property* menghubungankan *member* dari suatu kelas ke *member* kelas lainnya [7]. *Property* mencakup *subproperties* dan *superproperties*. *Property* juga disebut sebagai "slots" dalam terminologi yang lain. Skema *property* dicontohkan pada Gambar 2. 2.



Gambar 2. 2 *Property*

Instance merujuk pada sesuatu yang teridentifikasi dalam model konseptual. *Instance* juga disebut sebagai individual. Sehingga dapat dikatakan bahwa *instance* menspesifikasikan *member* dari suatu *class*. *Instance* dapat berwujud (alat, bagian, benda, dkk) atau tidak berwujud (sebuah konsep, konstruksi mental). Contoh relasi antara *class*, *property*, dan *instance* dapat dilihat pada Gambar 2. 3.



Gambar 2. 3 Contoh Ontologi

Selain 3 komponen penting yang telah dijelaskan di atas, terdapat beberapa istilah lain yang perlu dipahami dalam konteks ontologi antara lain d*omain* (*member* dari suatu kelas yang dapat menjadi subjek dari *property* yang diberikan), r*ange* (member dari suatu kelas yang dapat menjadi objek dari *property* yang diberikan), c*onstraint* dan *rule* (menentukan batasan dan istilah-istilah teknis untuk mendukung *reasoning*), dan *relationship* (mekanisme inferensi untuk menggenerasi pengetahuan baru).

Dalam *semantic modelling*, ontologi dapat direpresentasikan dengan berbagai bahasa yang sudah memiliki standar seperti RDF, RDFS, atau OWL. Secara umum, kegunaan ontologi adalah sebagai *controlled vocabulary, semantic interoperability, knowledge sharing,* dan *reuse* [8]. Menurut Cristami dan Cuel [9], sebuah model ontologi dapat dikembangkan melalui beberapa tahapan proses sebagai berikut:

- Tahap penentuan domain.
- Tahap penggunaan ulang.
- Tahap penentuan istilah pada ontologi.
- Tahap pendefinisian *class* dan *hierarchy class.*
- Tahap pendefinisian *properties.*
- Tahap pendefinisian *constraints.*
- Tahap pembuatan *instance.*

2.3. OWL

OWL (*Web Ontology Language*) adalah bahasa yang digunakan untuk merepresentasikan definisi dari kosakata dan relasi antar kata sehingga makna suatu informasi menjadi eksplisit. OWL merupakan ekstensi dari RDF Schema. W3C *(World Wide Web Consortium)* merekomendasikan OWL dalam penulisan ontologi untuk web semantik [10]. Kemampuan RDF untuk memodelkan informasi dari sekumpulan data diadaptasi oleh OWL. Model informasi tersebut merupakan tata bahasa untuk mengartikan sekumpulan data menjadi suatu informasi [11].

Elemen pada OWL terdiri atas *classes, properties,* dan *instances*. *Classes* pada OWL mendefinisikan *root* dari semua entitas yang berkaitan dengan `Thing`. Sehingga secara implisit, semua *class* merupakan *subclass* dari `Thing`. Untuk membuat sebuah *class*, digunakan sintaksis `owl:Class`. Sintaksis tersebut sudah menyatakan *subclass* dengan `rdfs:subClassOf`.

Class memiliki beberapa deskripsi yaitu *equivalent classes*, *superclasses*, *members*, dan *disjoint classes*. *Equivalent classes* terjadi jika sebuah kelas memiliki definisi yang sama dengan kelas lainnya namun menggunakan nama atau URI yang berbeda. Contoh dari *equivalent classes* adalah pernyataan sebagai berikut, "*Parent it is any Person who has a child"*. *Superclasses* adalah kelas yang berada satu level di atas kelas tersebut. *Members* menjelaskan individual yang merupakan *member* dari *class* yang dirujuk. Sedangkan *disjoint classes* merupakan kelas pengecualian. *Member* dari *class* tersebut tidak bisa menjadi *member* di *class* yang lain. Biasanya tidak perlu didefinisikan, kecuali jika menggunakan *external reasoning* atau aplikasi lain yang membutuhkan *class* definisi yang lebih eksplisit. Contoh dari *disjoint classes* adalah sebagai berikut, *"Father disjoint class with Mother".*

Property merupakan relasi *binary*. Ada dua jenis *property* pada OWL, yaitu *object property* dan *datatype property*. Sama halnya seperti *class*, *property* juga dapat dinyatakan sebagai `subPropertyOf`. Untuk memberikan batasan suatu *property*,

dapat digunakan *domain* dan *range*, yang disebut juga sebagai *global restriction* karena berlaku untuk umum, tidak terbatas pada *class* tertentu. *Datatype property* pada OWL mendeskripsikan relasi antara *instance* dengan RDF literal atau XML *Schema datatype*. *Datatype property* menghubungkan objek dengan *datatype values*. Berikut *datatype values* yang dapat digunakan:

string normalizedString boolean
decimal float double
integer nonNegativeInteger positiveInteger nonPositiveInteger negativeInteger
long int short byte
unsignedLong unsignedInt unsignedShort unsignedByte
hexBinary base64Binary
dateTime time date gYearMonth gYear gMonthDay gDay gMonth
anyURI token language
NMTOKEN Name NCName

Sedangkan *object properties* berfungsi untuk menghubungkan suatu objek dengan objek lainnya. *Object properties* mendeskripsikan relasi antara *instance* dari dua *classes*. Terdapat beberapa mekanisme yang digunakan untuk penentuan *property*. Tidak menutup kemungkinan bahwa mekanisme tersebut memanfaatkan *object property characteristics* yang menggunakan penalaran. *Object property characteristics* yang dapat digunakan adalah sebagai berikut:

1. *Functional and Inverse functional*
   Functional property hanya boleh memiliki satu *member* di *range*nya untuk semua *member domain*.
   Contoh: `hasGender`
2. *Transitive*
(x property y) and (y property z) implies (x property z)
   Contoh: `hasSibling`
3. *Symmetric*
if (x property y) then (y property x)
   Contoh: `hasSpouse`
4. *Asymmetric*
if (x property y) then (y property x) → *false*
   Contoh: `hasWife`
5. *Reflexive*
   Individu bisa berelasi dengan dirinya sendiri.
6. *Irreflexive*
   Tidak boleh ada individual yang berelasi dengan dirinya sendiri untuk *property* tersebut.
   Contoh: `isParentOf`

Individuals atau disebut juga *instances* adalah anggota dari *classes*. *Instances* dapat dipandang sebagai objek yang ada pada domain tertentu. Sama seperti `owl:Class` yang menjadi *meta level* untuk *class*, begitu juga *class* yang telah didefinisikan menjadi *mata level* untuk *instance*.

Dalam proses *reasoning* pada OWL, dapat digunakan beberapa *reasoner* seperti Pellet, FACT++, HermiT, dan lain sebagainya. *Reasoner* ini digunakan untuk memeriksa konsistensi pada ontologi, melakukan klarifikasi berdasarkan relasi hierarki, dan mendapatkan fakta baru berdasarkan *axioms* dan *rules* [8]. Terdapat beberapa contoh penggunaan ontologi, khususnya OWL seperti yang didokumentasikan oleh W3C [12] sebagai berikut:

- *Web portal*
- *Multimedia collectionns*
- *Corporate web site management*
- *Design documentation*
- *Agents and services*
- *Ubiquitous computing*

2.4. Semantic Web Rule Language (SWRL)

SWRL merupakan bahasa berbentuk *unary* dan *binary rule statement* yang menjadi bagian dari OWL. Pada dasarnya, *rule* terdiri dari *antecedent* dan *consequent*, keduanya terdiri dari pasangan-pasangan atom. Jika *antecedent* bernilai benar, maka *consequent* juga akan bernilai benar [13]. Pada Tabel 2.1 berikut akan dijabarkan bentuk-bentuk atom yang didefinisikan.

Tabel 2.1 Komponen SWRL

Atom	Deskripsi
C(x)	C adalah deklarasi *class* (nama *class*) dan x adalah nama individual atau variabel
D(y)	D adalah deklarasi *data range* dan y adalah variabel atau *data value*
P(x, y)	P adalah data atau *object property*, x dan y adalah variabel atau OWL individual. y adalah sebuah individual jika P adalah *object property*, sedangkan y adalah sebuah *data value* jika P adalah *data property*.
sameAs(x, y)	x dan y adalah variabel atau individual yang menyatakan bahwa keduanya merupakan individu yang sama

Atom	Deskripsi
differentFrom(x, y)	x dan y adalah variabel atau individual yang menyatakan bahwa keduanya merupakan individu yang berbeda

Berikut merupakan contoh SWRL *rule* yang menyatakan bahwa x3 adalah ayah (*father*) dari x1 jika x2 adalah orang tua (*parent*) dari x1 dan x3 adalah istri (*wife*) dari x2.

```
hasParent(?x1, ?x2), hasWife(?x2, ?x3) -> hasFather(?x1, ?x3)
```

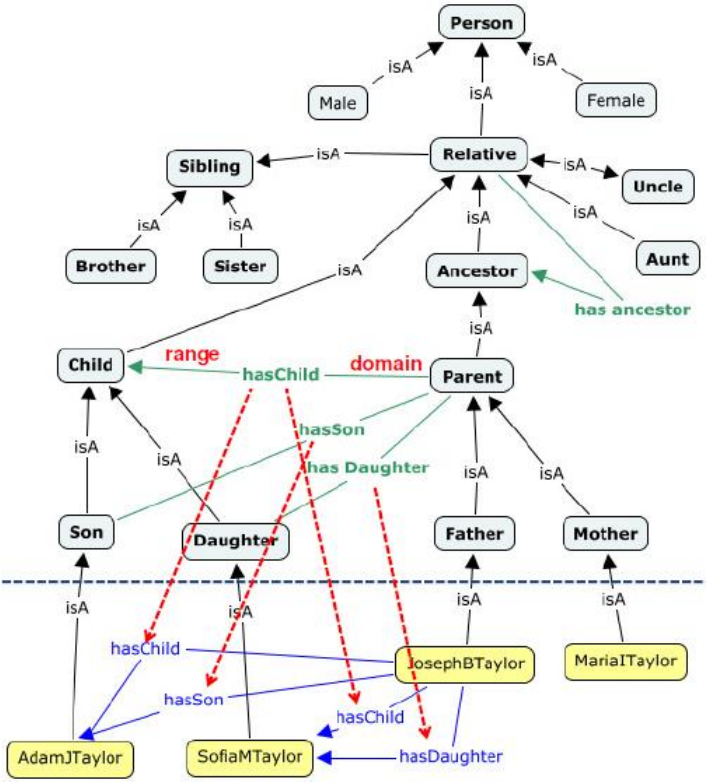
Tanda "->" digunakan sebagai penghubung antara *antecedent* dan *consequent* atom. Sedangkan "," berfungsi sebagai penghubung antar atom. Sebuah variabel ditandai dengan ekspresi "?".

2.5. Family Relationships Ontology

Family relationship umumnya digambarkan dengan terstruktur melalui silsilah keluarga. Manusia membutuhkan informasi tentang silsilah keluarganya untuk berbagai hal, diantaranya adalah untuk memperat ikatan batin antar anggota keluarga, mempermudah keturunannya dalam menelusuri asal usul keluarganya, menentukan pewarisan, perkawinan, dan lain sebagainya. Silsilah keluarga adalah bagan yang menampilkan struktur keluarga dalam bentuk pohon. Silsilah keluarga menyimpan informasi yang mendeskripsikan relasi antar anggota keluarga secara kompleks [14].

Keluarga memiliki struktur garis keturunan yang panjang. Jika relasi keturunan dicari secara manual, maka dibutuhkan waktu dan analisis yang lama. Belum tentu setiap anggota keluarga mengenal kerabatnya, karena pada umunya hanya satu atau dua orang yang mengetahui detail keluarga. Semakin bertambahnya pengetahuan membuat hubungan dalam sebuah keluarga dapat diketahui dengan mudah melalui *Family Relationships Ontology*. Ontologi ini memiliki beberapa kelebihan, diantaranya adalah dapat diketahuinya keakraban, relasi, pewarisan, *domain*, *range*,

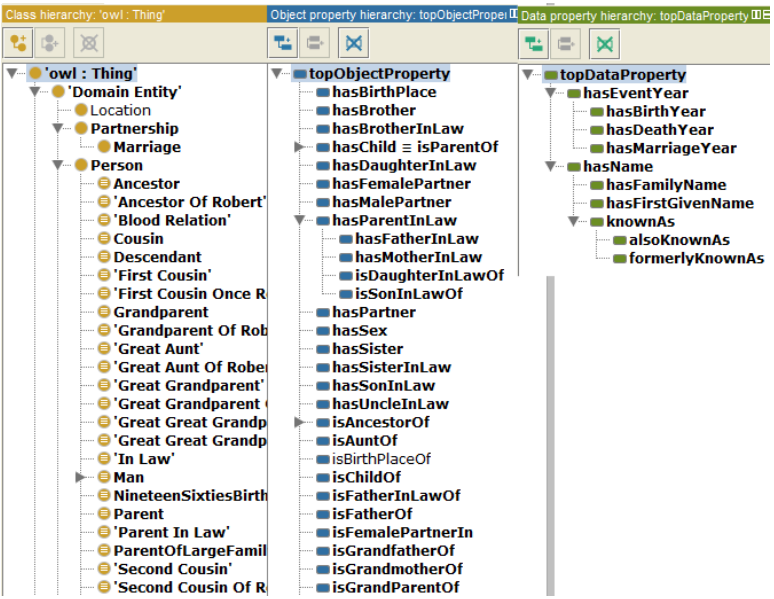
constraint, dan kesimpulan logis dalam sebuah keluarga secara praktis. *Family Relationships Ontology* juga mempermudah untuk memahami ontologi lainnya karena hubungan kekeluargaan dapat digeneralisasikan ke domain pengetahuan lainnya dan konsepnya yang mudah dipahami. Contoh ontologi hubungan keluarga dapat dilihat pada Gambar 2. 4.



Gambar 2. 4 *Family Relationships Map*

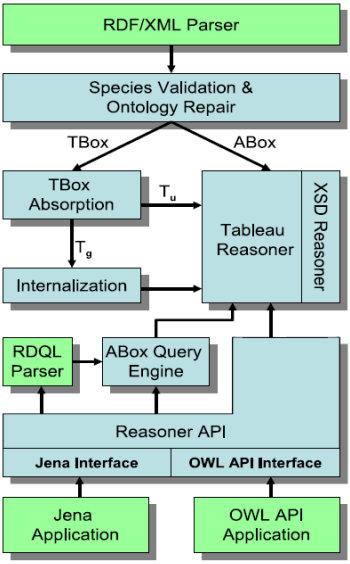
Terdapat banyak ontologi yang telah dibangun menggunakan *domain* keluarga, salah satunya adalah ontologi yang digunakan pada pengerjaan Tugas Akhir ini, yaitu

FamilyTree. Ontologi tersebut didapatkan dari portal The University of Manchester. Ontologi FamilyTree memiliki URI *http://www.co-ode.org/roberts/family-tree.owl* [15]. Ontologi tersebut adalah sebuah ontologi sederhana dengan *domain* hubungan keluarga yang mendeskripsikan keluarga Robert Stevens. FamilyTree merupakan ontologi yang kompleks dan lengkap. Pembangunan ontologi tersebut dimaksudkan untuk menghasilkan suatu ontologi yang meminimalkan *relationships* dan memaksimalkan *inference*. Oleh karena itu, ontologi ini banyak menggunakan *role chain*, *nominal*, dan *properties hierarchy*. Cuplikan kelas, properti, dan individu yang terdapat dalam ontologi tersebut dapat dilihat pada Gambar 2. 5.



Gambar 2. 5 Ontologi *FamilyTree* Keluarga Robert Stevens

2.6. Pellet Reasoner



Gambar 2. 6 Arsitektur Pellet *Reasoner*

(Sumber: Pellet An OWL DL Reasoner, Bijan Parsia & Evren Sirin)

Implementasi OWL *reasoner* yang sudah ada didasarkan pada beberapa pendekatan. *Reasoner* deskripsi logika (seperti Pellet dan RacerPro) menggunakan implementasi algoritma tableaux. Penggunaan algoritma tersebut memanfaatkan penelitian yang telah dilakukan untuk kasus algoritma deskripsi logika pengetahuan berdasar pada formalitas OWL [16]. Pellet didasarkan pada algoritma tableaux yang dikembangkan untuk mengekspresikan *Description Logics*. Pellet mendukung semua konstruksi OWL DL termasuk `owl:oneOf` dan `owl: hasValue`. Saat ini, belum ada algoritma lengkap yang *decidable* dan efektif untuk semua OWL DL (khususnya, penanganan *inverse properties* dan *cardinality restrictions*). Pellet mengkombinasikan algoritma

yang lengkap sebagai reasoner, yaitu OWL DL tanpa *nominals* (SHIN (D)) dan OWL DL tanpa *inverse properties* (SHON (D)). Algoritma ini dikombinasikan untuk mendapatkan penalaran yang lengkap dan berkaitan dengan semua DL. Pellet telah terbukti praktis berguna dalam berbagai pekerjaan saat ini. Gambar 2. 6 menunjukkan komponen utama Pellet *reasoner*.

Ontologi OWL di*parsing* ke dalam RDF dengan pola *triple* (Sintaksis RDF / XML, N3 dan N-Triple yang mendukung). Pellet memvalidasi jenis dari ontologi dimana *triple RDF* dikonversi menjadi pernyataan dan *axiom* berbasis pengetahuan. Jika level ontologi adalah OWL Full karena hilangnya tipe pola *triple*, maka Pellet menggunakan beberapa heuristik untuk memperbaiki ontologi. Misalnya *untyped resource* yang telah digunakan dalam predikat *position* dalam sebuah pola *triple* akan disimpulkan menjadi *datatype property* jika *triple* literal dalam posisi objek.

Pellet menyimpan *axiom* tentang kelas-kelas dalam komponen TBox dan menyimpan pernyataan tentang individu dalam komponen abox. Partisi TBox, adalah tempat penyerapan dan optimasi berlangsung. Tableau reasoner menggunakan *rule* tableau standar dan mencakup berbagai optimasi standar seperti keterkaitan yang diarahkan pada *backjumping*, percabangan semantik dan strategi pemblokiran awal. *Datatype reasoning* untuk *built-in* dan pengambilan XML *Schema datatypes* primitif didukung dalam *reasoner* ini. Pellet diimplementasikan dalam Java dan berada di bawah lisensi MIT [17].

2.7. Basis Data Relasional

Basis data relasional adalah suatu konsep penyimpanan data terstruktur. Teori basis data relasional ini dikemukakan pertamakali oleh Dr. E.F. Codd [16].

Dalam basis data relasional, data disimpan dalam bentuk relasi atau tabel dua dimensi, dan antara tabel satu dengan tabel lainnya terdapat hubungan atau *relationship* sehingga dapat disimpulkan, basis data adalah kumpulan dari sejumlah tabel yang saling hubungan atau saling keterkaitan. Kumpulan dari data yang

diorganisasikan sebagai tabel tadi disimpan dalam bentuk data elektronik di dalam *hardisk* komputer dan dikelompokan secara logis berdasarkan *schema user*.

Untuk membuat struktur tabel, mengisi data ke tabel, memperbarui data dan menghapus data dari tabel diperlukan software. Perangkat lunak yang digunakan membuat tabel, isi data, ubah data, dan hapus data disebut "Relational Database Management System" atau yang biasa disingkat dengan RDBMS. Sedangkan perintah yang digunakan untuk membuat tabel, mengisi tabel, mengubah tabel, dan menghapus data disebut perintah SQL yang merupakan singkatan dari "Structure Query Language". Jadi, setiap aplikasi perangkat lunak RDBMS pasti bisa dipakai untuk menjalankan perintah SQL.

Sebenarnya fungsi RDBMS bukan hanya untuk membuat tabel, isi data, ubah data dan hapus data. Untuk manajemen data dalam skala yang besar dan agar bisa mendukung proses bisnis yang berkelanjutan atau terus menerus dan real time suatu "Relational Database Management System" dituntut untuk mempunyai kemampuan manajemen user dan keamanan data yang terjamin, mencadangkan data dan mengembalikan data serta kemampuan lainnya yang berkaitan dengan kecepatan pemrosesan data.

2.8. Query SQL

Structured Query Language (SQL) adalah sekumpulan perintah khusus yang digunakan untuk mengakses data dalam database relasional. SQL merupakan sebuah bahasa komputer yang mengikuti standar ANSI (*American Nasional Standard Institute*) yang digunakan dalam manajemen *database* relasional [18]. Dengan SQL, maka dapat mengakses *database*, menjalankan *query* untuk mengambil data dari *database*, menambahkan data ke *database*, menghapus data di dalam *database*, dan mengubah data di dalam *database*. Saat ini hampir semua *server database* yang ada mendukung SQL untuk melakukan manajemen datanya.

Terdapat 3 (tiga) jenis perintah SQL, yaitu:

1. DDL atau Data Definition Language

   DDL merupakan perintah SQL yang berhubungan dengan pendefinisian suatu struktur database, dalam hal ini database dan table. Perintah SQL yang termasuk dalam DDL antara lain :

   - CREATE
   - ALTER
   - RENAME
   - DROP

2. DML atau Data Manipulation Language

   DML merupakan perintah SQL yang berhubungan dengan manipulasi atau pengolahan data atau *record* dalam *table*. Perintah SQL yang termasuk dalam DML antara lain :

   - SELECT
   - INSERT
   - UPDATE
   - DELETE

3. DCL atau Data Control Language

   DCL merupakan perintah SQL yang berhubungan dengan pengaturan hak akses *user*, baik terhadap *server*, *database*, table maupun *field*. Perintah SQL yang termasuk dalam DCL antara lain :

   - GRANT
   - REVOK

2.9. Query SPARQL

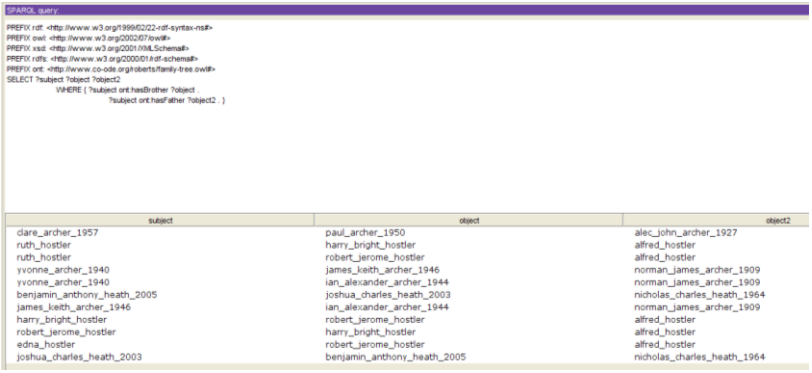
SPARQL adalah Bahasa *query* untuk RDF. *Graph* RDF merupakan terdiri dari *triple* yang terbentuk dari subjek, predikat dan objek, RDF dapat didefenisikan pada RDF konsep dan abstrak *syntax* konsep. *triple* ini dapat datang dari berbagai macam *source*. Untuk *instance* dapat diperoleh secara langsung dari dokumen RDF, dapat disimpulkan dari triple RDF. Ekspresi RDF dapat disimpan dalam format lain seperti XML dan *Database Relational* [19].

SPARQL adalah Bahasa *query* untuk mendapatkan informasi dari *graph* RDF. yang menyediakan fasilitas sebagai berikut :

- Mengekstrak informasi dalam bentuk URI, *blank node* dan literal

- Mengekstrak RDF *Subgraph*
- Membangun *graph* RDF baru berdasarkan *query graph*

sebagai bahasa akses data sangat cocok digunakan untuk *local* maupun *remote*. Contoh *query* SPARQL terdapat pada Gambar 2. 7.


SPARQL query:

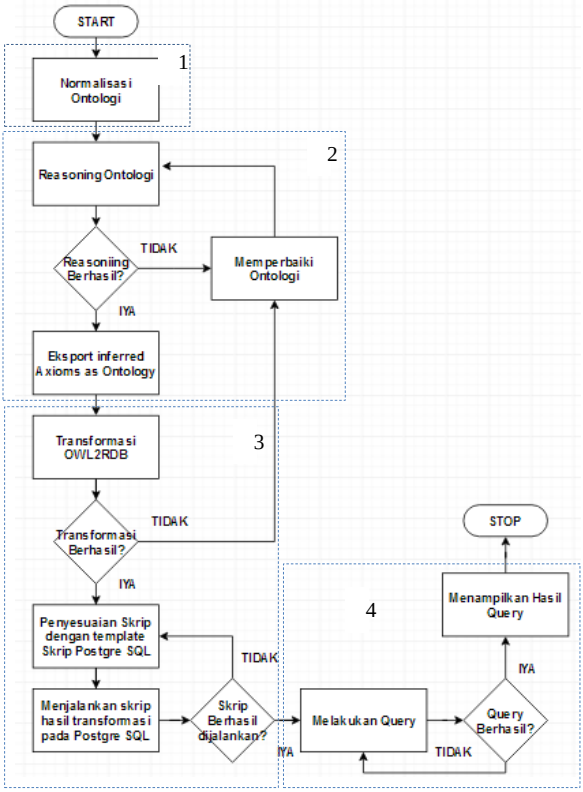
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.co-ode.org/roberts/family-tree.owl#>
SELECT ?subject ?object ?object2
        WHERE { ?subject ont:hasBrother ?object .
                ?subject ont:hasFather ?object2. }
```

subject	object	object2
clare_archer_1957	paul_archer_1950	alec_john_archer_1927
ruth_hostler	harry_bright_hostler	alfred_hostler
ruth_hostler	robert_jerome_hostler	alfred_hostler
yvonne_archer_1940	james_keith_archer_1946	norman_james_archer_1909
yvonne_archer_1940	ian_alexander_archer_1944	norman_james_archer_1909
benjamin_anthony_heath_2005	joshua_charles_heath_2003	nicholas_charles_heath_1964
james_keith_archer_1946	ian_alexander_archer_1944	norman_james_archer_1909
harry_bright_hostler	robert_jerome_hostler	alfred_hostler
robert_jerome_hostler	harry_bright_hostler	alfred_hostler
edna_hostler	robert_jerome_hostler	alfred_hostler
joshua_charles_heath_2003	benjamin_anthony_heath_2005	nicholas_charles_heath_1964

Gambar 2. 7 Penggunaan *Query* SPARQL

BAB III
METODOLOGI PEMECAHAN MASALAH

Pada bab ini dijelaskan mengenai langkah-langkah yang dilakukan untuk melakukan penyimpanan dan peng-*query*-an basis data relasional berbasis ontologi, mulai dari normalisasi ontologi sampai menampilkan hasil *query* basis data relasional. Seperti pada Gambar 3. 1



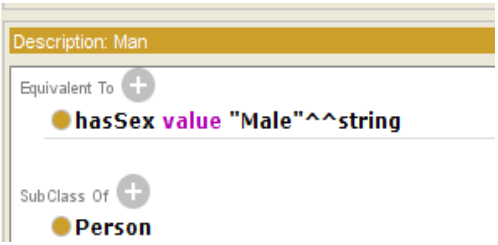
Gambar 3. 1 *Flowchart* Pemecahan Masalah

Sebelum membahas tentang flowchart alur pemecahan masalah yang digunakan, terlebih dahulu akan dijelaskan tentang

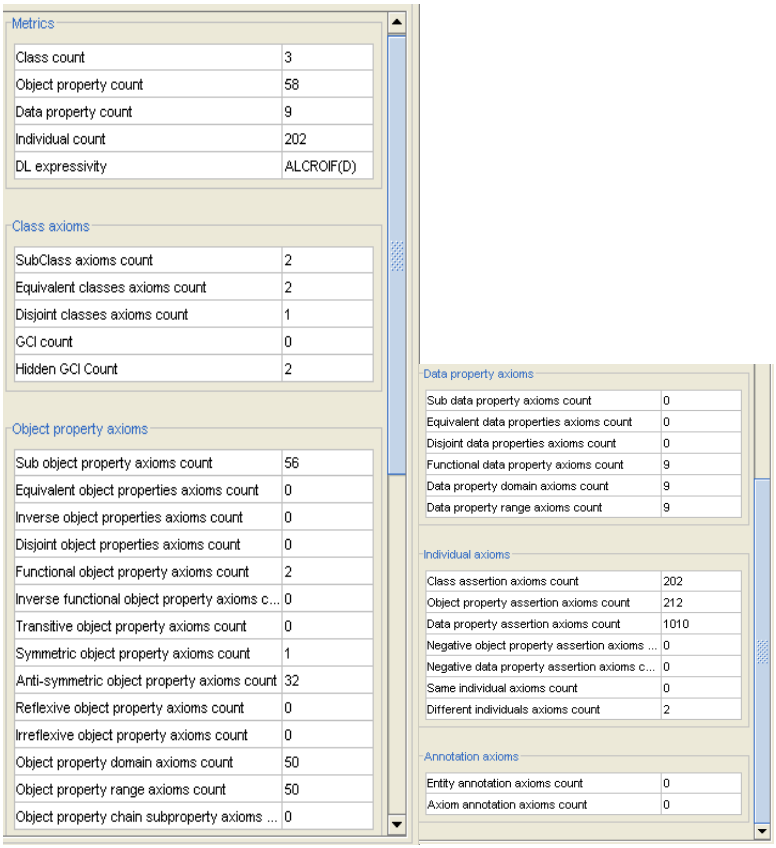
elemen-elemen yang digunakan pada ontologi pohon keluarga dalam Tugas Akhir ini.

3.1. Elemen-Elemen Ontologi Pohon Keluarga

Pada tahap pembuatan suatu ontologi pohon keluarga, beberapa hal yang harus dibangun adalah *class*(tipe), *object property*, *data property*, *individual*, dan *rule*. *Class* ini, berisikan suatu *type* dari ontologi yang telah dibuat, pada *class* ini harus diisikan kolom *equivalent To* seperti pada Gambar 3. 2, agar Protégé bisa menentukan anggota dari tiap *class* secara otomatis. Kemudian pada *object property* dan *data property* harus diisikan *domain* dan *range* dari masing-masing *object property* dan *data property* tersebut, dan jumlah dari *domain* dan *range* dari masing-masing *object property* tersebut maksimal berjumlah satu. Jika *object property* dan *data property* telah terisi lengkap sesuai yang seharusnya, maka selanjutnya bisa membuat suatu individu, dengan mengisikan *class*, *object property*, dan *data property* dari masing-masing individu tersebut, untuk jumlah class dari tiap individu hanya boleh berisikan maksimal satu *class*(tipe) saja, sedangkan untuk jumlah *object property* dan *data property* dari masing-masing individu jumlahnya tak terbatas, sesuai kondisi inputan yang ada. Tahap selanjutnya yaitu pembuatan *rule*. *Rule* ini berfungsi untuk suatu proses *reasoning* yang akan dilakukan. Dengan menggunakan *rule* maka sistem akan mengisikan *object property* dan *data property* dari masing-masing individu secara otomatis. Untuk jumlah data yang digunakan pada Tugas Akhir ini dijelaskan pada Gambar 3. 3.



Gambar 3. 2 *Equivalent To* Pada *Class*

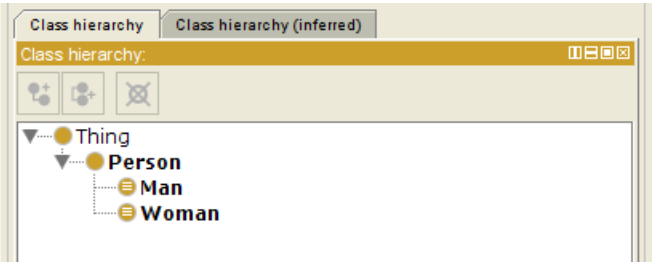


Gambar 3. 3 Data Ontologi Pohon Keluarga

3.1.1. Class

Pada ontologi pohon keluarga ini jumlah *class*(tipe) yang digunakan ada 3 buah *class*, yaitu *class person*, *man*, dan *woman*. Pada kasus ini *man* dan *woman* adalah *subclass* dari person. Pada *class* ini harus mengisikan kolom *equivalen to* dari masing-masing kelas yang tidak mempunyai *subclass*, untuk permasalahan ini yang harus diisi adalah kelas *man* dan *woman*. Hal tersebut berfungsi agar sistem bisa menentukan *member* dari

masing-masing *class* secara otomatis. Gambaran dari kelas tersebut dijelaskan pada Gambar 3. 4.



Gambar 3. 4 Perancangan Kelas

3.1.2. Object Property

Pada *Object Property* ini, jumlah *object property* yang dibangun sebanyak 50 *object property*, kemudian dari masing-masing *object property* tersebut harus diisikan sebuah *domain* dan *range* sesuai kondisi *object property* tersebut. *Object property* yang dibuat pada Tugas Akhir ini adalah seperti yang ada pada Tabel 3. 1 berikut:

Tabel 3. 1 *Object Property*

No	**Nama *Object property***	***Domain***	***Range***
1	*hasRelations*	*Person*	*Person*
2	*hasBloodRelations*	*Person*	*Person*
3	*has Aunt*	*Person*	*Woman*
4	*hasChild*	*Person*	*Person*
5	*hasDaughter*	*Person*	*Woman*
6	*hasSon*	*Person*	*Man*
7	*hasCousin*	*Person*	*Person*
8	*hasGrandChild*	*Person*	*Person*
9	*hasGrandDaughter*	*Person*	*Woman*
10	*hasGrandSon*	*Person*	*Man*
11	*hasGrandParent*	*Person*	*Person*
12	*hasGrandFather*	*Person*	*Man*
13	*hasGrandMother*	*Person*	*Woman*

14	*hasGreatGrandChild*	*Person*	*Person*
15	*hasGreatGrandDaughter*	*Person*	*Woman*
16	*hasGreatGrandSon*	*Person*	*Man*
17	*hasGreatGrandParent*	*Person*	*Person*
18	*hasGreatGrandFather*	*Person*	*Man*
19	*hasGreatGrandMother*	*Person*	*Woman*
20	*hasNephiew*	*Person*	*Man*
21	*hasNiece*	*Person*	*Woman*
22	*hasParent*	*Person*	*Person*
23	*hasFather*	*Person*	*Man*
24	*hasMother*	*Person*	*Woman*
25	*hasSibling*	*Person*	*Person*
26	*hasBrother*	*Person*	*Man*
27	*hasOlderBrother*	*Person*	*Man*
28	*hasYoungerBrother*	*Person*	*Man*
29	*hasSister*	*Person*	*Woman*
30	*hasOlderSister*	*Person*	*Woman*
31	*hasYoungerSister*	*Person*	*Woman*
32	*hasUncle*	*Person*	*Man*
33	*hasInlaw*	*Person*	*Person*
34	*hasAuntInLaw*	*Person*	*Woman*
35	*hasNephiewInLaw*	*Person*	*Man*
36	*hasNieceInLaw*	*Person*	*Woman*
37	*hasParentInLaw*	*Person*	*Person*
38	*hasFatherInLaw*	*Person*	*Man*
39	*hasMotherInLaw*	*Person*	*Woman*
40	*hasSiblingInLaw*	*Person*	*Person*
41	*hasBrotherInLaw*	*Person*	*Man*
42	*hasOlderBrotherInLaw*	*Person*	*Man*
43	*hasYoungerBrotherInLaw*	*Person*	*Man*
44	*hasSisterInLaw*	*Person*	*Woman*
45	*hasOlderSisterInLaw*	*Person*	*Woman*
46	*hasYoungerSisterInLaw*	*Person*	*Woman*
47	*hasSpouse*	*Person*	*Person*

48	*hasHusband*	*Woman*	*Man*
49	*hasWife*	*Man*	*Woman*
50	*hasUncleInLaw*	*Person*	*Man*

3.1.3. Data Property

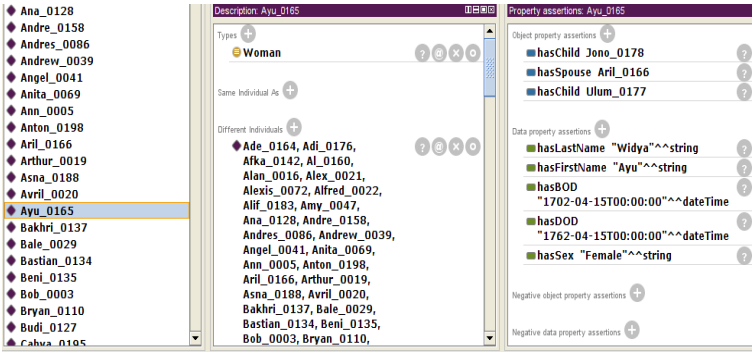
Pada *Data Poperty* ini, terdapat sebanyak 9 *data property*, kemudian dari masing-masing *data property* tersebut harus diisikan sebuah *domain* dan *range* sesuai kondisi *data property* tersebut. *Data property* yang dibangun pada Tugas Akhir ini adalah seperti yang ada pada Tabel 3. 2 berikut:

Tabel 3. 2 *Data Property*

No	**Nama *Object property***	***Domain***	***Range***
1	*BornInYear*	*Person*	*Integer*
2	*DeadInYear*	*Person*	*Integer*
3	*hasAge*	*Person*	*Integer*
4	*hasBOD*	*Person*	*DateTime*
5	*hasDOD*	*Person*	*DateTime*
6	*hasFirstName*	*Person*	*String*
7	*hasLastName*	*Person*	*String*
8	*hasMariageName*	*Person*	*String*
9	*hasSex*	*Person*	*String*

3.1.4. Individual

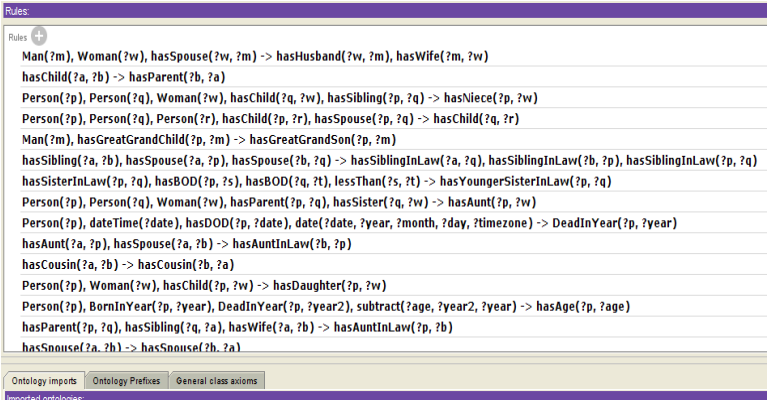
Pada *Individual* ini, data individu yang bangun berisikan tipe, *object property* dan *data property* dari masing-masing individu. Untuk *data property*, tidak harus mengisi 9 *data property* dengan manual, tapi hanya mengisi 5 buah *data property* saja, sisanya akan otomatis diisi oleh sistem saat proses *reasoning*. Kemudian untuk *object property*, dari sejumlah 50 jenis *object property* yang ada, hanya mengisikan 2 jenis *object property* saja, yaitu *hasChild* dan *hasSpouse*. Contoh dari sebuah individu seperti Gambar 3. 5 di bawah ini.



Gambar 3. 5 Individu

3.1.5. Perancangan Rules

Pada *Rules* ini, dibangun sebuah *rule* yang berfungsi untuk proses *reasoning* yang akan dilakukan. *Rule* ini berfungsi agar sistem bisa mengisi *data property*, *object property*, *domain object property*, dll secara otomatis sesuai dengan *rule* yang ada. Contoh dari sebagaian *rule* yang digunakan pada Tugas Akhir ini dijelaskan pada Gambar 3. 6.



Gambar 3. 6 *Rules*

3.2. Pemecahan Masalah

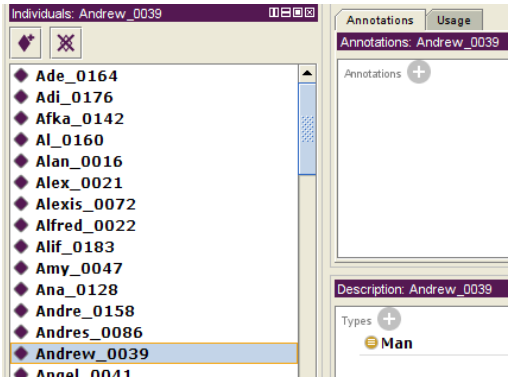
Pemecahan masalah yang digunakan dalam Tugas Akhir ini adalah dengan melakukan proses normalisasi ontologi, *reasoning* ontologi, transformasi data dengan OWL2RDB, dan Pemetaan *syntax query*. Penjelasan lebih lanjut akan dijelaskan pada subbab selanjutnya.

3.2.1. Normalisasi Ontologi

Pada tahapan ini, saat data sudah selesai dibuat dan sebelum dilakukan proses *reasoning*, maka terlebih dahulu perlu dilakukan proses normalisasi. Proses normalisasi ini dilakukan terhadap data *individual, object property* dan *data property* yang sudah dibuat [20].

3.2.1.1. Normalisasi Individual

Pada individu yang sudah dibangun, setiap individu tidak boleh memiliki lebih dari satu *class assertion axiom* (tipe). Biasanya individu hasil *reasoning* mempunyai tipe lebih dari satu, oleh karena itu, untuk menangani proses tersebut, maka setelah proses *reasoning* yaitu pada saat proses "Export inferred axioms as ontology" harus menghilangkan tanda centang pada *class assertions* (*Individual types*). Contoh normalisasi individu sesuai dengan Gambar 3. 7.



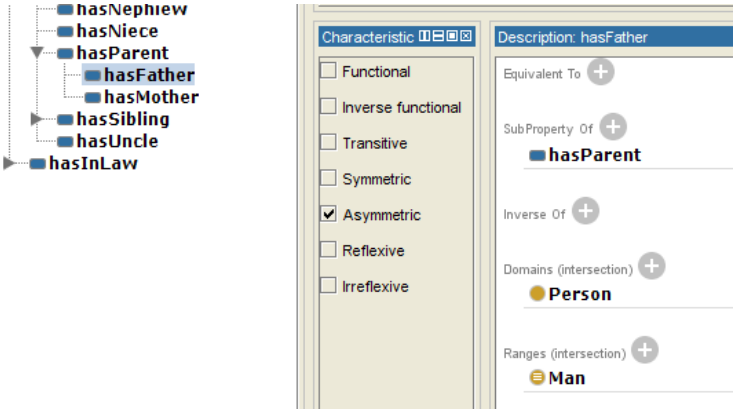
Gambar 3. 7 Normalisasi Individu

3.2.1.2. Normalisasi Object Property

Pada *object property* yang telah dibuat harus dilakukan normalisasi terlebih dahulu, dalam kasus ini normalisasi yang dilakukan yaitu sebanyak 2 kali, normalisasi *domain* dan *range object property*, dan normalisasi karakter *object property*

Normalisasi Domain dan Range Object Property

Pada tahapan normalisasi ini, ada aturan dimana setiap domain dan range dari *object property* yang dibuat haruslah mempuyai tepat satu *domain*, dan satu *range*. Hal ini dilakukan supaya tidak terjadi kesalahan saat dilakukan proses transformasi ontologi ke basis data relasional dengan menggunakan OWL2RDB. Contoh normalisasi *domain* dan *range object property* sesuai dengan Gambar 3. 8.

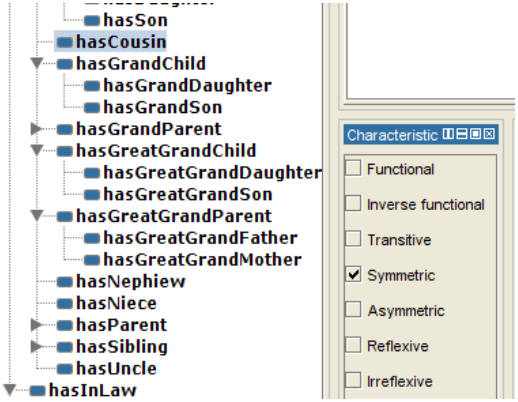


Gambar 3. 8 Normalisasi *Domain* dan *Range Object Property*

Normalisasi Characteristics Object Property

Normalisasi *characteristics* dilakukan dengan cara mengisi karakter dari tiap *object property* tersebut. Apakah itu masuk ke dalam *functional property*, *symetric property*, maupun *asymmetric property*. Apabila *domain* dan *range* dari *object property* tersebut mempunyai relasi *one to one*, maka harus memilih *functional property*. Dan apabila hubungan dari *domain*

ke *range* dari *object property* sama dengan hubungan *range* ke *domain* maka harus memilih *symetric property*, jika berbeda maka harus memilih *asymmetric property*. Contoh normalisasi *characteristics object property* sesuai dengan Gambar 3. 9.



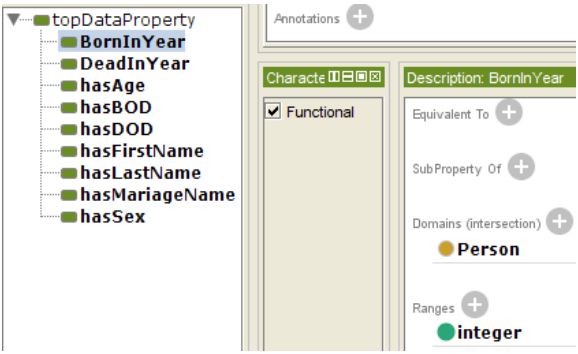
Gambar 3. 9 Normalisasi *Characteristics Object Property*

3.2.1.3. Normalisasi Data Property

Pada *data property* yang telah dibuat harus dilakukan normalisasi terlebih dahulu, dalam kasus ini normalisasi yang dilakukan yaitu sebanyak 2 kali, normalisasi *domain* dan *range data property*, dan normalisasi karakter *data property*

Normalisasi Domain dan Range Data Property

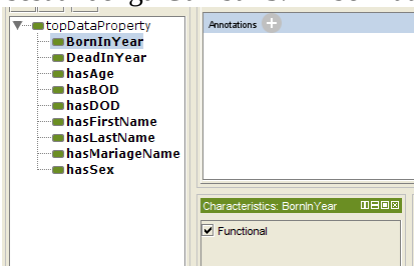
Pada tahapan normalisasi ini, ada aturan dimana setiap *domain* dan *range* dari *data property* yang dibangun haruslah mempuyai tepat satu *domain* dan satu *range*. Hal ini dilakukan supaya tidak terjadi kesalahan saat dilakukan proses transformasi ontologi ke dalam basis data relasional dengan menggunakan OWL2RDB. Contoh normalisasi *domain* dan *range data property* sesuai dengan Gambar 3. 10.



Gambar 3. 10 Normalisasi *Domain* dan *Range Data Property*

Normalisasi Characteristics Data Property

Normalisasi *characteristics* dilakukan dengan cara mengisi karakter dari tiap *data property* tersebut. Pada ontologi yang dibangun ini semua karakter dari setiap *data property* sama, yaitu *functional property*. Contoh normalisasi *characteristics object property* sesuai denga Gambar 3. 11 berikut.

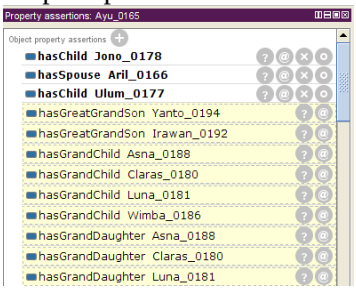


Gambar 3. 11 Normalisasi *Characterisctics Data Property*

3.2.2. Reasoning Ontologi

Pada proses *reasoning* ini menggunakan bantuan sebuah Pellet sebagai *plugin* dari Protégé. Dengan begitu maka sistem akan mengisikan data *object property*, *data property*, *domain object property*, dll secara otomatis sesuai *rule* yang telah dibuat sebelumnya. Setelah proses *reasoning* berhasil maka dilakukan penyimpanan ontologi dengan memanfaatkan fitur "Export inferred axioms as ontology" dengan menghilangkan tanda

centang pada *class assertions* (*Individual types*). Hasil *reasoning* dari data tersebut seperti pada Gambar 3. 12.



Gambar 3. 12 Hasil *Reasoning* Individu Ayu_0165

Pada gambar tersebut bisa dilihat bahwa data awalnya hanya data yang bertuliskan tebal, yaitu individu Ayu_0165 mempunyai pasangan Aril_0166, mempunyai anak Jono_0178 dan mempunyai anak Ulum_0177. Kemudian setelah dilakukan proses *reasoning* maka muncul data hasil pengolahan *rule* yang telah dibangun sebelumnya seperti yang terlihat pada Gambar 3. 12. Dimana tulisan yang mempunyai *background* warna kuning tersebut adalah hasil *reasoning rule* yang dibangun.

Data Ayu_0165 *hasGrandChild* Asna_0188, *hasGrandChild* Claras_0180, *hasGrandChild* Luna_0181, Ayu_0165 *hasGrandChild* Wimba_0186 didapatkan dari *rule* Kode Sumber 3. 1.

```
Person(?p), Person(?q), Person(?r),
hasChild(?q, ?p), hasChild(?r, ?q) ->
hasGrandParent(?p, ?r)
```

```
hasGrandParent(?a, ?b) -> hasGrandChild(?b, ?a)
```

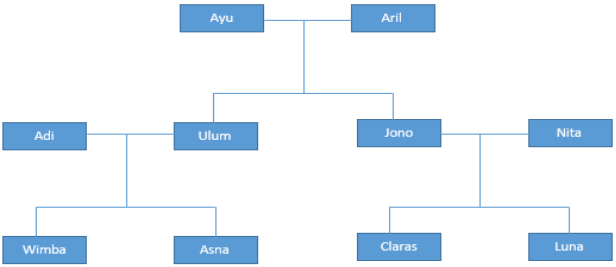
Kode Sumber 3. 1 *Rule hasGrandChild*

Sedangkan untuk Ayu_0165 *hasGrandDaughter* Asna_0188, *hasGrandDaughter* Claras_0180, *hasGrandDaughter* Luna_0181 didapatkan dari *rule* Kode Sumber 3. 2.

```
Woman(?w),        hasGrandChild(?p,        ?w)        ->
hasGrandDaughter(?p, ?w)
```

Kode Sumber 3. 2 *Rule HasGrandDaughter*

Untuk potongan diagram pohon keluarga yang dibangun dapat dilihat pada Gambar 3. 13.



Gambar 3. 13 Potongan Diagram Ontologi Pohon Keluarga

Dan untuk jumlah data setelah dilakukan proses *reasoning* adalah seperti pada Gambar 3. 14 dan Gambar 3. 15.

Data property axioms

Sub data property axioms count	0
Equivalent data properties axioms count	0
Disjoint data properties axioms count	0
Functional data property axioms count	9
Data property domain axioms count	9
Data property range axioms count	9

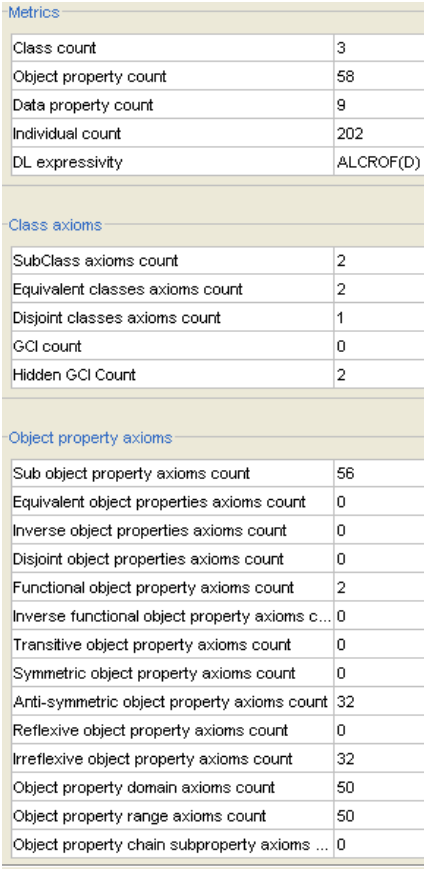
Individual axioms

Class assertion axioms count	202
Object property assertion axioms count	13224
Data property assertion axioms count	1692
Negative object property assertion axioms ...	0
Negative data property assertion axioms c...	0
Same individual axioms count	0
Different individuals axioms count	2

Annotation axioms

Entity annotation axioms count	0
Axiom annotation axioms count	0

Gambar 3. 14 Jumlah Data Hasil *Reasoning*



Metrics

Class count	3
Object property count	58
Data property count	9
Individual count	202
DL expressivity	ALCROF(D)

Class axioms

SubClass axioms count	2
Equivalent classes axioms count	2
Disjoint classes axioms count	1
GCI count	0
Hidden GCI Count	2

Object property axioms

Sub object property axioms count	56
Equivalent object properties axioms count	0
Inverse object properties axioms count	0
Disjoint object properties axioms count	0
Functional object property axioms count	2
Inverse functional object property axioms c...	0
Transitive object property axioms count	0
Symmetric object property axioms count	0
Anti-symmetric object property axioms count	32
Reflexive object property axioms count	0
Irreflexive object property axioms count	32
Object property domain axioms count	50
Object property range axioms count	50
Object property chain subproperty axioms ...	0

Gambar 3. 15 Jumlah Data Hasil *Reasoning*

3.2.3. Transformasi Data dengan OWL2RDB

Pada proses transformasi data ontologi menjadi data relasional digunakan OWL2RDB sebagai *plugin* pada Protégé untuk mengubah secara otomatis data ontologi tersebut menjadi sebuah *syntax query*. *Syntax* tersebut butuh sedikit perubahan agar bisa sesuai dengan *template syntax* yang diinginkan oleh PostgreSQL sebelum dijalankan pada aplikasi PostgreSQL.

Pada *syntax* ini, setiap *class* dijadikan menjadi sebuah tabel sendiri. Sedangkan untuk *object property* atau *data property*, apabila mempunyai *characteristic functional property* maka akan dibuat menjadi sebuah kolom dan dimasukkan ke dalam tabel sesuai nama *domain*-nya. Sedangkan untuk yang memiliki karakter selain *functional prope*rty, maka akan menjadi sebuah tabel sendiri dengan nama "*domain + object property*" atau "*domain + data property*".

Dalam menjalankan *syntax query* tersebut pada SQL Editor PostgreSQL ada urutannya tersendiri. Urutan dalam menjalankan *syntax query* tersebut adalah sebagai berikut:

- Tabel Hasil transformasi ontologi suatu *class.*
- Tabel Hasil transformasi ontologi suatu *object property.*
- Kolom Hasil transformasi ontologi suatu *data property.*
- Isi tabel dan kolom hasil transformasi ontologi suatu *member class, domain* dan *range object property, dan domain dan range data property.*

Penjelasan lebih lanjut seperti pada Tabel 3. 3 dan Tabel 3. 4. Sedangkan untuk hasil penyimpanan data dengan menggunakan *plugin* OWL2RDB bisa dilihat pada Gambar 3. 16.

Tabel 3. 3 Hasil OWL2RDB *Data Property*

No	**Nama *Object property***	***Domain***	***Property***	**Keterangan**
1	*BornInYear*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *Borninyear*
2	*DeadInYear*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *Deadinyear*
3	*has Age*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasage*
4	*hasBOD*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasbod*

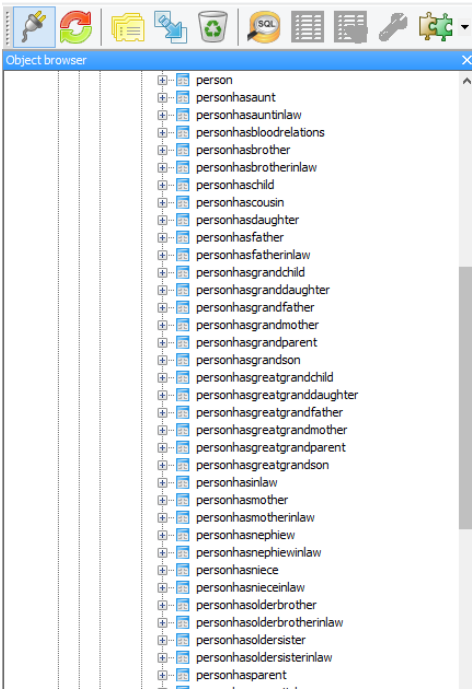
5	*hasDOD*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasdod*
6	*hasFirstName*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasfirstname*
7	*hasLastName*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *haslastname*
8	*hasMariageName*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasmariagename*
9	*hasSex*	*Person*	*Functional*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hassex*

Tabel 3. 4 Hasil OWL2RDB *Object Property*

No	**Nama *Object property***	**Keterangan**
1	*hasRelations*	Membentuk tabel personhasrelations
2	*hasBloodRelations*	Membentuk tabel personhasbloodrelations
3	*has Aunt*	Membentuk tabel personhasaunt
4	*hasChild*	Membentuk tabel personhaschild
5	*hasDaughter*	Membentuk tabel personhasdaughter
6	*hasSon*	Membentuk tabel personhasson
7	*hasCousin*	Membentuk tabel personhascousin
8	*hasGrandChild*	Membentuk tabel personhasgrandchild
9	*hasGrandDaughter*	Membentuk tabel personhasgranddaughter
10	*hasGrandSon*	Membentuk tabel personhasgrandson
11	*hasGrandParent*	Membentuk tabel personhasgrandparent
12	*hasGrandFather*	Membentuk tabel personhasgrandfather
13	*hasGrandMother*	Membentuk tabel personhasgrandmother
14	*hasGreatGrandChild*	Membentuk tabel personhasgreatgrandchild
15	*hasGreatGrandDaughter*	Membentuk tabel personhasgreatgranddaughter

16	*hasGreatGrandSon*	Membentuk tabel personhasgreatgrandson
17	*hasGreatGrandParent*	Membentuk tabel personhasgreatgrandparent
18	*hasGreatGrandFather*	Membentuk tabel personhasgreatgrandfather
19	*hasGreatGrandMother*	Membentuk tabel personhasgreatgrandmother
20	*hasNephiew*	Membentuk tabel personhasnephiew
21	*hasNiece*	Membentuk tabel personhasniece
22	*hasParent*	Membentuk tabel personhasparent
23	*hasFather*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasfather*
24	*hasMother*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasmother*
25	*hasSibling*	Membentuk tabel personhassibling
26	*hasBrother*	Membentuk tabel personhasbrother
27	*hasOlderBrother*	Membentuk tabel personhasolderbrother
28	*hasYoungerBrother*	Membentuk tabel personhasyoungerbrother
29	*hasSister*	Membentuk tabel personhassister
30	*hasOlderSister*	Membentuk tabel personhasoldersister
31	*hasYoungerSister*	Membentuk tabel personhasyoungersister
32	*hasUncle*	Membentuk tabel personhasuncle
33	*hasInlaw*	Membentuk tabel personhasinlaw
34	*hasAuntInLaw*	Membentuk tabel personhasauntinlaw
35	*hasNephiewInLaw*	Membentuk tabel personhasnephiewinlaw
36	*hasNieceInLaw*	Membentuk tabel personhasnieceinlaw
37	*hasParentInLaw*	Membentuk tabel personhasparentinlaw
38	*hasFatherInLaw*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasfatherinlaw*
39	*hasMotherInLaw*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasmotherinlaw*
40	*hasSiblingInLaw*	Membentuk tabel personhassiblinginlaw
41	*hasBrotherInLaw*	Membentuk tabel personhasbrotherinlaw
42	*hasOlderBrotherInLaw*	Membentuk tabel personhasolderbrotherinlaw

43	*hasYoungerBrotherInLaw*	Membentuk tabel personhasyoungerbrotherinlaw
44	*hasSisterInLaw*	Membentuk tabel personhassisterinlaw
45	*hasOlderSisterInLaw*	Membentuk tabel personhasoldersisterinlaw
46	*hasYoungerSisterInLaw*	Membentuk tabel personhasyoungersisterinlaw
47	*hasSpouse*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hasspouse*
48	*hasHusband*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *hashusband*
49	*hasWife*	Digabungkan ke dalam tabel *Person*, menjadi sebuah kolom *haswife*
50	*hasUncleInLaw*	Membentuk tabel personhasuncleinlaw



Gambar 3. 16 Hasil Penyimpanan Data dengan *Plugin* OWL2RDB

3.2.4. Pemetaan Syntax Query

Pada tahap pemetaan *syntax query*, sistem akan memetakan *syntax query* SPARQL menjadi *syntax query* SQL. *Syntax* ini dipetakan berdasarkan pola-pola yang mungkin muncul pada penulisan *syntax query* SPARQL [21] [22].

Dengan begitu pada sebuah basis data relasional pengguna tidak hanya dapat melakukan *query* SQL saja, tapi juga dapat melakukan *query* SPARQL, maka dari itu harus dibuatkan suatu aplikasi yang bisa megubah sebuah *syntax query* SPARQL menjadi sebuah *syntax query* SQL, dan menampilkan data hasil pengolahan *query* SQL tersebut. Dengan demikian maka pengguna dapat melakukan *query* SPARQL untuk basis data relasional.

Berikut ini adalah salah satu contoh pemetaan *syntax query* SPARQL menjadi *syntax query* SQL:

Syntax query SPARQL untuk *data property*:

```
SELECT ?subject ?object
WHERE { Yudi_0162 TA:BornInYear ?object }
```

Kode Sumber 3. 3 *Syntax Query* SPARQL

Syntax query SQL hasil *convert syntax* SPARQL:

```
Select personname, borninyear
From person
Where personname='Yudi_0162'
```

Kode Sumber 3. 4 *Syntax Query* SQL

Untuk penjelasan lebih lanjut mengenai pemetaan *syntax query* SPARQL ini, akan dibahas pada subbab 4.2.

(Halaman ini sengaja dikosongkan)

BAB IV
ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini dijelaskan tentang analisis permasalahan dan perancangan Tugas Akhir. Analisis permasalahan membahas tentang permasalahan yang diangkat dalam Tugas Akhir ini beserta solusi yang ditawarkan. Selanjutnya dibahas juga tentang perancangan sistem yang dibuat.

4.1. Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain cakupan permasalahan, deskripsi umum sistem, kasus penggunaan sistem dan kebutuhan perangkat lunak.

4.1.1. Cakupan Permasalahan

Permasalahan yang diangkat dalam Tugas Akhir ini adalah penyimpanan dan peng-*query*-an suatu basis data relasional yang berbasis ontologi dalam permasalahan pohon keluarga. Studi kasus permasalahan tersebut dapat diselesaikan dengan cara melakukan normalisasi terlebih dahulu terhadap ontologi pohon keluarga yang sudah dibuat sebelumnya, kemudian dilakukan proses *reasoning* dengan menggunakan Pellet *reasoner*. Setelah proses *reasoning* selesai, akan didapatkan fakta-fakta baru yang kemudian disimpan lagi sebagai ontologi baru dalam bentuk RDF dengan memanfaatkan fitur "*Eksport inferred axiom as ontology"* pada Protégé. Ontologi baru tersebut hanya dapat dibuka dengan aplikasi yang mendukung format RDF. Agar ontologi tersebut dapat dibuka dengan aplikasi PostgreSQL yang tidak dapat mendukung format RDF maka dibutuhkan *plugin* OWL2RDB pada Protégé untuk mengubah ontologi tersebut menjadi sebuah *syntax query*. *Syntax* tersebut harus sedikit diperbaiki terlebih dahulu untuk menyesuaikan dengan *syntax* yang diinginkan oleh PostgreSQL sebelum dijalankan. Setelah data berhasil disimpan, maka peng-*query*-an yang bisa dilakukan adalah hanya *query* SQL saja, tidak bisa melakukan *query* SPARQL seperti pada saat data

tersebut masih berbentuk ontologi. Maka dari itu, supaya tidak hanya dapat melakukan *query* SQL *saja* pada basis data relasional, tapi juga bisa melakukan *query* SPARQL maka dibutuhkan sebuah sistem sederhana yang bisa mengubah sebuah *query SPARQL* menjadi sebuah *query* SQL agar bisa menampilkan hasil *query* SPARQL yang dilakukan. Tujuan dari aplikasi ini adalah agar bisa memberi kemudahan bagi pengguna dan juga untuk mempercepat waktu dalam pencarian data. Maka untuk memudahkan pengguna, sistem sederhana tersebut akan dirancang dengan tampilan yang mudah dipahami.

4.1.2. Deskripsi Umum Sistem

Perangkat lunak yang dibangun dalam pengerjaan Tugas Akhir ini diberi nama "SPARQL Relational Database". Nama tersebut diambil dari kata *query* SPARQL dan basis data relasional. Aplikasi sederhana ini dibangun dengan tujuan agar pengguna tidak hanya dapat melakukan *query* SQL pada basis data relasional, tapi pengguna juga dapat melakukan *query* SPARQL. Supaya dapat melakukan *query* SPARQL tersebut, maka perangkat lunak tersebut harus bisa meng-*convert* atau mengubah sebuah *query* SPARQL yang ada menjadi sebuah *query* SQL. "SPARQL Relational Database" dirancang sebagai perangkat lunak berbasis *dekstop* yang menggunakan bahasa pemrograman Java dan *database* PostgreSQL. Perangkat lunak ini memiliki masukan berupa *file* basis data relasional hasil Transformasi dari file OWL2. Berkas OWL2 yang dimaksud adalah ontologi baru yang dihasilkan setelah proses *reasoning*. Sedangkan keluaran dari perangkat lunak "SPARQL Relational Database" adalah aplikasi *dekstop* dengan tampilan *text area result* yang berisi sejumlah baris data hasil *query* yang bersumber dari *file* basis data relasional suatu pohon keluarga.

4.1.3. Spesifikasi Kebutuhan Perangkat Lunak

Bab ini menjelaskan kebutuhan perangkat lunak dalam bentuk diagram kasus dan diagram aktivitas. Masing-masing diagram menjelaskan perilaku atau sifat dari sistem ini.

4.1.3.1. Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan pokok yang harus dipenuhi agar sistem dapat berjalan dengan baik. Daftar kebutuhan fungsional dapat dilihat pada Tabel 4. 1**.**

Tabel 4. 1 Daftar Kebutuhan Fungsional Perangkat Lunak

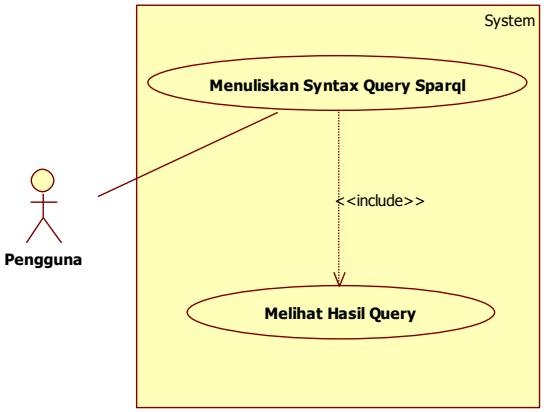
Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
TA-IF0001	Menuliskan *syntax query* SPARQL	Pengguna dapat menuliskan *syntax query* SPARQL pada kolom yang disediakan.
TA-IF0002	Melihat hasil *query*	Pengguna dapat melihat hasil *query* yang dilakukan.

4.1.4. Aktor

Aktor merupakan entitas-entitas yang terlibat dan berinteraksi langsung dengan sistem. Entitas yang dimaksud dapat berupa manusia, sistem, atau perangkat lunak yang lain. Aktor yang berinteraksi dengan Tugas Akhir ini yaitu pengguna yang diasumsikan tidak memahami bahasa pemrograman. Pengguna dapat menuliskan *syntax query* SPARQL pada kolom yang disediakan untuk kemudian dapat melihat data hasil peng-*query*-an di kolom hasil *query*.

4.1.5. Kasus Penggunaan

Kasus penggunaaan dalam Subbab ini akan dijelaskan secara rinci. Kasus penggunaan dijabarkan dalam bentuk spesifikasi kasus penggunaan dan diagram aktivitas. Diagram kasus penggunaan dapat dilihat pada Gambar 4. 1. Daftar kode diagram kasus penggunaan sistem dapat dilihat pada Tabel 4. 2.



Gambar 4. 1 Diagram Kasus Penggunaan Sistem

Tabel 4. 2 Daftar Kode Diagram Kasus Penggunaan

Kode Kasus Penggunaan	Nama
TA-TC0001	Menuliskan *syntax query* SPARQL
TA-TC0002	Melihat hasil *query*

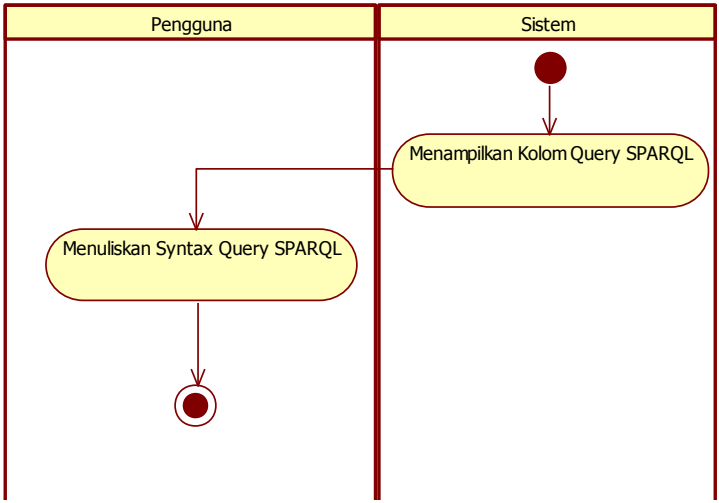
4.1.5.1. Menuliskan Syntax Query SPARQL

Pada kasus penggunaan ini, pengguna dapat menuliskan *syntax query* SPARQL sesuai dengan yang diinginkan pada kolom yang telah disediakan oleh sistem. Spesifikasi kasus penggunaannya dapat dilihat pada Tabel 4. 3 Diagram aktivitasnya dapat dilihat pada Gambar 4. 2

Tabel 4. 3 Spesifikasi Kasus Penggunaan Melakukan *Query* SPARQL

Nama	Melakukan *query* SPARQL
Kode	TA-TC0001
Deskripsi	Menuliskan *syntax query* SPARQL pada kolom yang disediakan.
Tipe	Fungsional

Pemicu	Pengguna menuliskan *syntax query SPARQL* dan menekan tombol "Submit'.
Aktor	Pengguna
Kondisi Awal	Sudah terdapat *file* pohon keluarga pada *database* PostgreSQL.
Aliran: **- Kejadian Normal**	1. Sistem menampilkan kolom sebagai tempat penulisan *query SPARQL*. 2. Pengguna menuliskan *syntax query SPARQL* pada kolom yang disediakan.
Kondisi Akhir	*Query SPARQL* telah siap diproses.
Kebutuhan Khusus	Tidak ada



Gambar 4. 2 Diagram Aktivitas Melakukan Query SPARQL

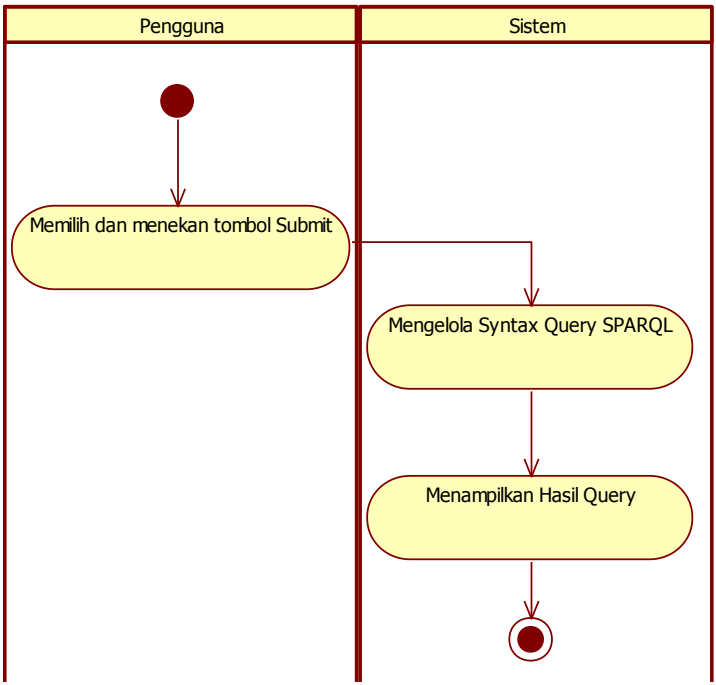
4.1.5.2. Melihat hasil Query

Pada kasus penggunaan ini, *system* akan mengelola *syntax query* SPARQL yang telah dituliskan sebelumnya, kemudian pengguna dapat melihat data hasil dari *syntax query* SPARQL tersebut pada kolom hasil, Spesifikasi kasus penggunaannya dapat

dilihat pada Tabel 4. 4. Diagram aktivitasnya dapat dilihat pada Gambar 4. 3.

Tabel 4. 4 Spesifikasi Kasus Penggunaan Melihat Hasil Q*uery* SPARQL

Nama	Melihat hasil *query* SPARQL
Kode	TA-TC0002
Deskripsi	Melihat hasil dari *syntax query SQL* yang telah dilakukan oleh pengguna.
Tipe	Fungsional
Pemicu	Pengguna memlih dan menekan tombol “Submit”.
Aktor	Pengguna
Kondisi Awal	Sudah terdapat *file* pohon keluarga pada *database* PostgreSQL dan *syntax query SPARQL* sudah selesai ditulis (siap diproses).
Aliran: - **Kejadian Normal**	1. Pengguna memilih dan menekan tambol Submit. 2. Sistem mengelola *syntax query SPARQL* tersebut. 3. Sistem menampilkan data yang ada pada *database PostgreSQL* sesuai dengan *syntax SQL*(hasil *query*).
Kondisi Akhir	Sistem menampilkan data hasil *query*.
Kebutuhan Khusus	Tidak ada



Gambar 4. 3 Diagram Aktivitas Melihat Hasil *Query* SPARQL

4.2. Perancangan Sistem

Bagian ini membahas mengenai perancangan sistem yang akan dibuat. Sistem ini berfungsi untuk memetakan *syntax query* SPARQL yang dituliskan oleh pengguna menjadi *syntax query* SQL, dan menampilkan data hasil pengolahan *sy*ntax SQL tersebut.

Untuk itu diperlukan beberapa *Case* supaya dapat memenuhi macam-macam pola *syntax query* SPARQL yang mungkin terjadi. *Case* yang digunakan pada sistem yang dibangun adalah seperti pada Tabel 4. 5.

Tabel 4. 5 Pemetaan SPARQL *Query* ke dalam SQL *Query*

Case	Klausa Where			Keterangan
	S	P	O	
Case_001	-	v	-	Predikat (P) merupakan *data property* dengan *range integer*.
Case_002	v	v	-	Predikat (P) merupakan *data property* dengan *range integer*.
Case_003	-	v	v	Predikat (P) merupakan *data property* dengan *range integer*.
Case_004	-	v	-	Predikat (P) merupakan *data property* dengan *range* selain *integer*.
Case_005	v	v	-	Predikat (P) merupakan *data property* dengan *range* selain *integer*.
Case_006	-	v	v	Predikat (P) merupakan *data property* dengan *range* selain *integer*.
Case_007	-	v	-	Predikat (P) merupakan *object property* dengan *domain woman/man*. Dan pada klausa *select* terdiri dari subjek, dan objek.
Case_008	-	v	-	Predikat (P) merupakan *object property* dengan *domain woman/man*. Dan pada klausa *select* terdiri dari subjek saja.
Case_009	-	v	-	Predikat (P) merupakan *object property* dengan *domain woman/man*. Dan pada klausa *select* terdiri dari objek saja.
Case_010	v	v	-	Predikat (P) merupakan *object property* dengan *domain woman/man*
Case_011	-	v	v	Predikat (P) merupakan *object property* dengan *domain woman/man*
Case_012	-	v	-	Predikat (P) merupakan *object property* dengan *domain* selain *woman* dan *man*. Dan pada klausa *select* terdiri dari subjek, dan objek.
Case_013	-	v	-	Predikat (P) merupakan *object property* dengan *domain* selain

				woman dan *man*. Dan pada klausa *select* terdiri dari subjek saja.
Case_014	-	v	-	Predikat (P) merupakan *object property* dengan *domain* selain *woman* dan *man*. Dan pada klausa *select* terdiri dari objek saja.
Case_015	v	v	-	Predikat (P) merupakan *object property* dengan *domain* selain *woman* dan *man*
Case_016	-	v	v	Predikat (P) merupakan *object property* dengan *domain* selain *woman* dan *man*
Case_017	v	-	-	Predikat tidak diketahui
Case_018	-	-	v	Predikat tidak diketahui
Case_019	-	-	-	Predikat tidak diketahui
Case_020	v	-	v	Predikat tidak diketahui

Berdasarkan Tabel 4. 5, S adalah sebuah subjek, P adalah sebuah predikat, dan O adalah sebuah objek, sedangkan tanda "-" berarti tidak diketahui, dan tanda "v" berarti diketahui. Untuk penjelasan lebih lanjut beserta contohnya adalah sebagai berikut:

Case_001

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek atau objek atau keduanya, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s ?o
where ?s hasage ?o
```

Syntax query SQL

```
select personname, hasage
from person
where hasage != 0
```

Case_002

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* "where" yang diketahui adalah subjek, dan predikat. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where Yudi_0162 hasage ?o
```

Syntax query SQL

```
select personname, hasage
from person
where personname= 'Yudi_0162'
and hasage != 0
```

Case_003

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikat dan objek. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s
where ?s hasage 70
```

Syntax query SQL

```
select personname, hasage
from person
where hasage = 70
```

Case_004

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek atau objek atau keduanya, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range* selain *integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s ?o
where ?s hasfirstnames ?o
```

Syntax query SQL

```
select personname, hasfirstname
from person
```

Case_005

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* "where" yang diketahui adalah subjek, dan predikat. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range* selain *integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where Yudi_0162 hasfirstname ?o
```

Syntax query SQL

```
select personname, hasfirstname
from person
where personname= 'Yudi_0162'
and hasfirstname != 'Null'
```

Case_006

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikat dan objek. Dimana predikat tersebut merupakan suatu *data property* yang mempunyai *range* selain *integer*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s
where ?s hasfirstname Yudi
```

Syntax query SQL

```
select personname, hasfirstname
from person
where hasfirstname = 'Yudi'
```

Case_007

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek atau objek atau keduanya, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain woman* atau *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s ?o
where ?s haswife ?o
```

Syntax query SQL

```
select distinct personname , (select distinct
person.personname as personhaswife from person where
person.personid= man.haswife)
from person, man
where person.personid= man.manid
and haswife!= 0
```

Case_008

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain woman* atau *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s
where ?s haswife ?o
```

Syntax query SQL

```
select distinct personname
from person, man
where person.personid= man.manid
and haswife!= 0
```

Case_009

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain woman* atau *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where ?s haswife ?o
```

Syntax query SQL

```
select distinct personname
from person, man
where person.personid= man.haswife
and haswife!= 0
```

Case_010

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* "where" yang diketahui adalah subjek dan predikat. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain woman* atau *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where Yudi_0162 haswife ?o
```

Syntax query SQL

```
select distinct personname , (select distinct
person.personname as personhaswife from person where
person.personid= man.haswife)
from person, man
where personname = 'Yudi_0162'
and person.personid= man.manid
and haswife!= 0
```

Case_011

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikat dan objek. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain woman* atau *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?s
where ?s haswife Silfi_0161
```

Syntax query SQL

```
elect distinct (select distinct person.personname
from person where person.personid= man.manid),
personname as personhaswife
from person, man
where personname = 'Silfi_0161'
and person.personid= man.haswife
and haswife!= 0
```

Case_012

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek atau objek atau keduanya, sedangkan pada *syntax* "where" yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain* selain *woman* dan *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s ?o
where ?s haschild ?o
```

Syntax query SQL

```
select distinct personname , (select distinct
person.personname as personhaschild from person
where person.personid=personhaschild.person2id)
from person, personhaschild
where person.personid=personhaschild.person1id
```

Case_013

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada

syntax “where” yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain* selain *woman* dan *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
select ?s
where ?s haschild ?o
```

Syntax query SQL

```
select distinct personname
from person, personhaschild
where person.personid=personhaschild.person1id
```

Case_014

Case ini berjalan ketika kondisi “select” pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* “where” yang diketahui adalah predikatnya saja. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain* selain *woman* dan *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where ?s haschild ?o
```

Syntax query SQL

```
select distinct personname
from person, personhaschild
where person.personid=personhaschild.person2id
```

Case_015

Case ini berjalan ketika kondisi “select” pada *syntax* SPARQL yang dituliskan terdiri dari objek saja, sedangkan pada *syntax* “where” yang diketahui adalah subjek dan predikat. Dimana

predikat tersebut merupakan suatu *object property* yang mempunyai *domain* selain *woman* dan *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?o
where Yudi_0162 haschild ?o
```

Syntax query SQL

```
select distinct personname , (select distinct
person.personname as personhaschild from person
where person.personid=personhaschild.person2id)
from person, personhaschild
where personname = 'Yudi_0162'
and person.personid=personhaschild.person1id
```

Case_016

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek saja, sedangkan pada *syntax* "where" yang diketahui adalah predikat dan objek. Dimana predikat tersebut merupakan suatu *object property* yang mempunyai *domain* selain *woman* dan *man*.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?s
where ?s haschild Yudi_0162
```

Syntax query SQL

```
select distinct (select distinct person.personname
from                person                  where
person.personid=personhaschild.person1id),
personname as personhaschild
from person, personhaschild
where personname = 'Yudi_0162'
and person.personid=personhaschild.person2id
```

Case_017

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari predikat dan objek, sedangkan pada *syntax* "where" yang diketahui adalah subjeknya saja.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?p ?o
where Yudi_0162 ?p ?o
```

Syntax query SQL

Pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mempunyai *range integer*, yaitu sebanyak 3 kali. Dengan cara mengganti *data property borninyear* dengan 2 *data property* lainnya yang mempunyai *range integer*.

```
select personname, borninyear
from person
where personname= 'Yudi_0162'
and borninyear != 0
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mempunyai *range* selain *integer*, yaitu sebanyak 6 kali. Dengan cara mengganti *data property borninyear* dengan 5 *data property* lainnya yang mempunyai *range* selain *integer*.

```
select personname, hasbod
from person
where personname= 'Yudi_0162'
and hasbod != 'Null'
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak 2 kali. Karena hanya untuk mengecek *haswife* dan *hashusband*. Dengan cara mengganti *object property haswife* dengan *object property hashusband*.

```
select distinct personname , (select distinct
person.personname as personhaswife from person where
person.personid= man.haswife)
from person, man
where personname = 'Yudi_0162'
and person.personid= man.manidand person.personid=
man.manid
```

Dan untuk *syntax query* SQL di bawah ini, diulang sebanyak jumlah *object property* dikurangi 2, karena *object property haswife*, dan *hashusband* sudah mempunyai *syntax query* SQL tersendiri, yaitu sebanyak 48 kali. Dengan cara mengganti *object property hasson* dengan 47 *object property* yang lain.

```
select distinct personname , (select distinct
person.personname as personhasson from person where
person.personid=personhasson.manid)
from person, personhasson
where personname = 'Yudi_0162'
and person.personid=personhasson.personid
```

Case_018

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek dan predikat, sedangkan pada *syntax* "where" yang diketahui adalah objeknya saja.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?s ?p
where ?s ?p Yudi_0162
```

Syntax query SQL

Pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mempunyai *range* selain *integer*, yaitu sebanyak 6 kali. Dengan cara mengganti *data property hasbod* dengan 5 *data property* lainnya yang mempunyai *range* selain *integer*.

```
select personname, hasbod
from person
where hasbod = 'Yudi_0162'
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak 2 kali. Karena hanya untuk mengecek *haswife* dan *hashusband*. Dengan cara mengganti *object property hashusband* dengan *object property haswife.*

```
select distinct (select distinct person.personname
from person where person.personid= woman.womanid),
personname as personhashusband
from person, woman
where personname = 'Yudi_0162'
and person.personid= woman.hashusband
```

Dan untuk *syntax query* SQL di bawah ini, diulang sebanyak jumlah *object property* dikurangi 2, karena *object property haswife*, dan *hashusband* sudah mempunyai *syntax query* SQL tersendiri, yaitu sebanyak 48 kali. Dengan cara mengganti *object property hasson* dengan 47 *object property* yang lain.

```
select distinct (select distinct person.personname
from person where person.personid=
personhasfatherinlaw.personid), personname as
personhasfatherinlaw
from person, personhasfatherinlaw
where personname = 'Yudi_0162'
and person.personid=personhasfatherinlaw.manid
```

Case_019

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari subjek, predikat, dan objek, sedangkan pada *syntax* "where" tidak ada yang diketahui.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?s ?p ?o
where ?s ?p ?o
```

Syntax query SQL

Pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mempunyai *range integer*, yaitu sebanyak 3 kali. Dengan cara mengganti *data property borninyear* dengan 2 *data property* lainnya yang mempunyai *range integer*.

```
select personname, borninyear
from person
where borninyear != 0
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mempunyai *range* selain *integer*, yaitu sebanyak 6 kali. Dengan cara mengganti *data property borninyear* dengan 5 *data property* lainnya yang mempunyai *range* selain *integer*.

```
select personname, hasbod
from person
where hasbod != 'Null'
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak 2 kali. Karena hanya untuk mengecek *haswife* dan *hashusband*. Dengan cara mengganti *object property haswife* dengan *object property hashusband.*

```
select distinct personname , (select distinct
person.personname as personhaswife from person where
person.personid= man.haswife)
from person, man
where person.personid= man.manid
```

Dan untuk *syntax query* SQL di bawah ini, diulang sebanyak jumlah *object property* dikurangi 2, karena *object property haswife*, dan *hashusband* sudah mempunyai *syntax query* SQL tersendiri, yaitu sebanyak 48 kali. Dengan cara mengganti *object property hasrelations* dengan 47 *object property* yang lain.

```
select distinct personname , (select distinct
person.personname as personhasrelations from person
where person.personid=personhasrelations.person2id)
from person, personhasrelations
where person.personid=personhasrelations.person1id
```

Case_020

Case ini berjalan ketika kondisi "select" pada *syntax* SPARQL yang dituliskan terdiri dari predikat saja, sedangkan pada *syntax* "where" yang diketahui adalah subjek dan predikatnya.

Untuk penjelasan lebih lanjut, terdapat pada contoh *syntax query* berikut ini:

Syntax query SPARQL

```
Select ?p
where Adi_0176 ?p Asna_0188
```

Syntax query SQL

Pada *syntax query* SQL di bawah ini diulang sebanyak jumlah *data property* yang mepunyai *range* selain *integer*, yaitu sebanyak 6 kali. Dengan cara mengganti *data property hasmariagename* dengan 5 *data property* lainnya yang mempunyai *range* selain *integer*.

```
select personname, hasbod
from person
where personname = 'Adi_0176'
and hasbod= 'Asna_0188'
```

Sedangkan pada *syntax query* SQL di bawah ini diulang sebanyak 2 kali. Karena hanya untuk mengecek *haswife* dan *hashusband*. Dengan cara mengganti *object property haswife* dengan *object property hashusband.*

```
select distinct personname , (select distinct
person.personname as personhaswife from person where
person.personid= man.haswife)
from person, man
where personname = 'Adi_0176'
and person.personid= man.manid
and haswife= (select personid from person where
personname = 'Asna_0188')
and haswife != 0
```

Dan untuk *syntax query* SQL di bawah ini, diulang sebanyak jumlah *object property* dikurangi 2, karena *object property haswife*, dan *hashusband* sudah mempunyai *syntax query*

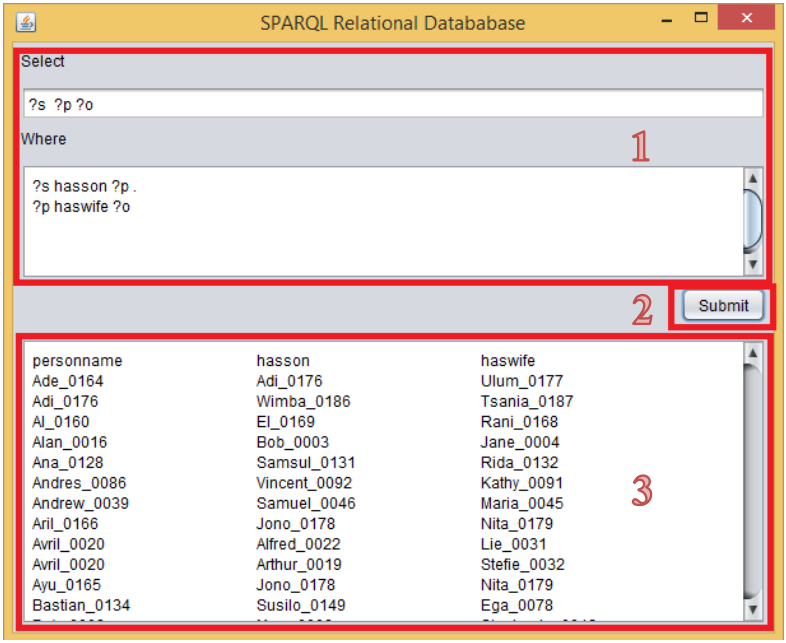
SQL tersendiri, yaitu sebanyak 48 kali. Dengan cara mengganti *object property hasrelations* dengan 47 *object property* yang lain.

```
select distinct personname , (select distinct
person.personname as personhasrelations from person
where person.personid=personhasrelations.person2id)
from person, personhasrelations
where personname = 'Adi_0176'
and person.personid=personhasrelations.person1id
and personhasrelations.person2id = (select personid
from person where personname= 'Asna_0188')
```

4.3. Perancangan Antarmuka Pengguna

Bagian ini membahas mengenai perancangan antarmuka yang akan dibuat. Rancangan antarmuka dibuat agar semudah mungkin dapat dipahami dan digunakan oleh pengguna.

Antarmuka aplikasi ini terdiri dari satu halaman. Pada halaman tersebut terdapat 3 *textArea* dan 1 tombol. Dimana 2 *textArea* untuk tempat penulisan *syntax query SPARQL*, kemudian ada 1 tombol "Submit" yang berfungsi sebagai pemicu pemrosesan *syntax query SPARQL* yang telah ditulis sebelumnya, dan juga ada 1 *textArea* lagi yang berfungsi untuk menampilkan data hasil pemrosesan *syntax query SPARQL*, kolom tersebut berisi sejumlah baris data. Rancangan antarmuka halaman tersebut dapat dilihat pada Gambar 4. 4. Penjelasan mengenai atribut-atribut yang terdapat pada halaman ini bisa dilihat pada Tabel 4. 6.



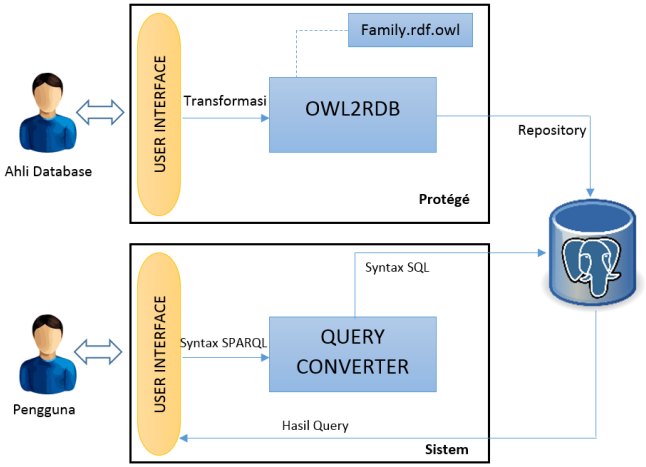
Gambar 4. 4 Antarmuka Halaman Utama

Tabel 4. 6 Spesifikasi Atribut Rancangan Antarmuka

No.	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	*Form Query SPARQL*	*Text Area*	Tempat penulisan *syntax query SPARQL*
2	*Submit Button*	*Button*	Mengeksekusi *request form*
3	*Return Data*	*Text Area*	Menampilkan hasil *syntax query SPARQL*

BAB V
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem yang telah dibuat. Proses implementasi dari setiap fungsi pada perangkat lunak ini akan diuraikan selangkapnya pada bab ini. Implementasi perangkat lunak ini menggunakan bahasa pemrograman java.



Gambar 5. 1 Arsitektur Perangkat Lunak

Arsitektur perangkat lunak yang dibangun pada Tugas Akhir ini dapat dilihat pada Gambar 5. 1. Penyimpanan pada PostgreSQL dilakukan secara manual oleh ahli *database* dengan bantuan *plugin* OWL2RDB pada Protégé. Sedangkan pengguna hanya dapat melakukan *query* SPARQL pada aplikasi yang dibangun.

5.1. Implementasi Fungsi

Pada bagian ini dijelaskan secara terperinci mengenai implementasi fungsi-fungsi yang digunakan dalam membangun sistem.

5.1.1. Fungsi pada Penyimpanan Basis Data Relasional Hasil Transformasi OWL2RDB

Pada subbab ini dijelaskan secara lengkap mengenai fungsi-fungsi yang ada pada *syntax* SQL hasil transformasi data ontologi ke dalam DBMS (*Database Management System*) yaitu PostgreSQL. Dalam menjalankan *syntax query* tersebut pada SQL Editor PostgreSQL ada urutannya tersendiri. Urutan dalam menjalankan *syntax query* tersebut adalah sebagai berikut:

- Tabel hasil transformasi ontologi suatu *class.*
- Tabel hasil transformasi ontologi suatu *object property.*
- Kolom hasil transformasi ontologi suatu *data property.*
- Isi tabel dan kolom hasil transformasi ontologi suatu *member class, domain* dan *range object property, dan domain dan range data property*..

5.1.1.1. Fungsi pada Penyimpanan Tabel Hasil Transformasi Ontologi suatu Class

Terdapat 3 tabel hasil transfromasi ontologi suatu *class*, yaitu tabel *person*, *woman*, dan *man*. Berikut fungsi pada tabel-tabel tersebut:

Create Table Person

Fungsi ini digunakan untuk menyimpan *class person* menjadi sebuah tabel *person*, dimana tabel ini nantinya akan dibuat untuk menyimpan data individu yang bertipe *person*. Seperti Kode Sumber 5. 1.

```
CREATE TABLE Person
(
  personId int NOT NULL
      CONSTRAINT PK_personId PRIMARY KEY
 , annotations int NULL,
      CONSTRAINT FK_person_annotations FOREIGN KEY
(annotations)
      REFERENCES OwlAnnotations(annotationsId)
```

Kode Sumber 5. 1 Fungsi *Create Table Person*

Create Table Woman

Fungsi ini digunakan untuk menyimpan *class woman* menjadi sebuah tabel *woman*, dimana tabel ini nantinya akan dibuat untuk menyimpan id individu yang bertipe *woman*. Seperti pada Kode Sumber 5. 2.

```
CREATE TABLE Woman
(
   womanId int NOT NULL
      CONSTRAINT PK_womanId PRIMARY KEY,
      CONSTRAINT FK_Person_Woman FOREIGN KEY
      (womanId) REFERENCES Person(personId)
)
```

Kode Sumber 5. 2 Fungsi *Create Table Woman*

Create Table Man

Fungsi ini digunakan untuk menyimpan *class man* menjadi sebuah tabel *man*, dimana tabel ini nantinya akan dibuat untuk menyimpan id individu yang bertipe *man*. Seperti pada Kode Sumber 5. 3.

```
CREATE TABLE Man
(
   manId int NOT NULL
      CONSTRAINT PK_manId PRIMARY KEY,
      CONSTRAINT FK_Person_Man FOREIGN KEY
      (manId) REFERENCES Person(personId)
)
```

Kode Sumber 5. 3 Fungsi *Create Table Man*

5.1.1.2. Fungsi pada Penyimpanan Tabel Hasil Transformasi Ontologi suatu Object Property

Terdapat 48 tabel hasil transfromasi ontologi suatu *object property*. Berikut fungsi pada beberapa tabel hasil transformasi *object property* ontologi tersebut:

Create Table PersonHasRelations

Fungsi ini digunakan untuk menyimpan *object property hasRelations* menjadi sebuah tabel *PersonHasRelations*, dimana tabel ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *object property hasRelations*. Seperti pada Kode Sumber 5. 4.

```
CREATE TABLE PersonHasRelations
(
  Id57 Serial Primary Key,
  person1Id int NOT NULL,
  person2Id int NOT NULL,

       CONSTRAINT FK_hasRelations_Person1 FOREIGN
KEY (Person1Id) REFERENCES Person(personId),
       CONSTRAINT FK_hasRelations_Person2 FOREIGN
KEY (Person2Id) REFERENCES Person(personId)
)
```

Kode Sumber 5. 4 Fungsi *Create Table PersonHasRelations*

Create Table PersonHasBloodRelations

Fungsi ini digunakan untuk menyimpan *object property hasBloodRelations* menjadi sebuah tabel *PersonHasBloodRelations*, dimana tabel ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *object property hasBloodRelations.* Seperti pada Kode Sumber 5. 5.

```
CREATE TABLE PersonHasBloodRelations
(
  Id81 Serial Primary Key,
  person1Id int NOT NULL,
  person2Id int NOT NULL,

       CONSTRAINT FK_hasBloodRelations_Person1
FOREIGN KEY (Person1Id) REFERENCES
Person(personId),
       CONSTRAINT FK_hasBloodRelations_Person2
FOREIGN KEY (Person2Id) REFERENCES Person(personId)
)
```

Kode Sumber 5. 5 Fungsi *Create Table PersonHasBloodRelations*

Create Table PersonHasAunt

Fungsi ini digunakan untuk menyimpan *object property hasAunt* menjadi sebuah tabel *PersonHasAunt*, dimana tabel ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *object property hasAunt*. Seperti pada Kode Sumber 5. 6.

```
CREATE TABLE PersonHasAunt
(
  Id11 Serial Primary Key, ]
  personId int NOT NULL,
  womanId int NOT NULL,

       CONSTRAINT FK_hasAunt_Person FOREIGN KEY
(PersonId) REFERENCES Person(personId),
       CONSTRAINT FK_hasAunt_Woman FOREIGN KEY
(womanId) REFERENCES Woman(womanId)
)
```

Kode Sumber 5. 6 Fungsi *Create Table PersonHasAunt*

5.1.1.3. Fungsi pada Penyimpanan Tabel Hasil Transformasi Ontologi suatu Object Property

Terdapat 48 tabel hasil transfromasi ontologi suatu *data property*. Berikut fungsi pada beberapa tabel hasil transformasi *data property* ontologi tersebut:

Alter Table (hasAge)

Fungsi ini digunakan untuk menyimpan *data property hasAge* menjadi sebuah kolom *hasAge*, dan disimpan pada tabel *person*. Kolom ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *data property hasAge*. Seperti pada Kode Sumber 5. 7.

```
ALTER TABLE Person
   ADD hasAge int NULL
```

Kode Sumber 5. 7 Fungsi Create Table *HasAge*

Alter Table (hasBOD)

Fungsi ini digunakan untuk menyimpan *data property hasBOD* menjadi sebuah kolom *hasBOD*, dan disimpan pada tabel *person*. Kolom ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *data property hasBOD*. Seperti pada Kode Sumber 5. 8.

```
ALTER TABLE Person
   ADD BornInYear int NULL
```

Kode Sumber 5. 8 Fungsi *Create Table HasBOD*

Alter Table (hasFirstName)

Fungsi ini digunakan untuk menyimpan *data property hasFirstName* menjadi sebuah kolom *hasfirstname*, dan disimpan pada tabel *person*. Kolom ini nantinya akan dibuat untuk menyimpan id individu yang mempunyai *data property hasFirstName.* Seperti pada Kode Sumber 5. 9.

```
ALTER TABLE Person
   ADD hasAge int NULL
```

Kode Sumber 5. 9 Fungsi *Create Table HasFirstName*

5.1.1.4. Fungsi pada Penyimpanan Isi Tabel Hasil Transformasi Ontologi.

Berikut beberapa fungsi pada penyimpanan isi tabel dari hasil transformasi ontologi tersebut:

Insert Into Person

Fungsi ini digunakan untuk menyimpan nilai dari suatu *class person*, yaitu berisi id dan nama dari suatu individu yang bertipe *person*. Seperti pada Kode Sumber 5. 10.

```
INSERT INTO Person(
PersonId,personName)
```

Kode Sumber 5. 10 Fungsi *Insert Into Person*

Insert Into Woman

Fungsi ini digunakan untuk menyimpan nilai dari suatu *class woman*, yaitu berisi id suatu individu yang bertipe *woman*. Seperti pada Kode Sumber 5. 11

```
INSERT INTO Woman(WomanId)
VALUES(4)
```

Kode Sumber 5. 11 Fungsi *Insert Into Woman*

Insert Into Man

Fungsi ini digunakan untuk menyimpan nilai dari suatu *class man*, yaitu berisi id suatu individu yang bertipe *man*. Seperti pada Kode Sumber 5. 12.

```
INSERT INTO Man(ManId)
VALUES(1)
```

Kode Sumber 5. 12 Fungsi *Insert Into Man*

Insert Into personHasRelation

Fungsi ini digunakan untuk menyimpan nilai dari suatu *object property personHasRelation*, yaitu berisi id *domain* dan id *range* dari *object property* tersebut. Seperti pada Kode Sumber 5. 13.

```
INSERT INTO PersonHasRelations(Person1Id,Person2Id)
VALUES( 1,73)
```

Kode Sumber 5. 13 Fungsi *Insert Into PersonHasRelations*

Insert Into personHasBloodRelations

Fungsi ini digunakan untuk menyimpan nilai dari suatu *object property personHasBloodRelations*, yaitu berisi id *domain* dan id *range* dari *object property* tersebut. Seperti pada Kode Sumber 5. 14.

```
INSERT INTO PersonHasBloodRelations(Person1Id,
Person2Id)
VALUES( 1,111)
```

Kode Sumber 5. 14 Fungsi *Insert Into PersonHasBloodRelations*

Insert Into personHasAunt

Fungsi ini digunakan untuk menyimpan nilai dari suatu *object property hasAunt*, yaitu berisi id *domain* dan id *range* dari *object property* tersebut. Seperti pada **Kode Sumber 5. 15**.

```
INSERT INTO PersonHasAuntInLaw(PersonId,WomanId)
VALUES( 1,40)
```

Kode Sumber 5. 15 Fungsi *Insert Into HasAunt*

Set hasAge

Fungsi ini digunakan untuk menyimpan nilai dari suatu *data property hasAge*, yaitu berisi id suatu individu dan umurnya. Seperti pada Kode Sumber 5. 16.

```
UPDATE Person SET hasAge = 60
where PersonId = 4
```

Kode Sumber 5. 16 Fungsi *Set HasAge*

Set hasBOD

Fungsi ini digunakan untuk menyimpan nilai dari suatu *data property hasBOD*, yaitu berisi id suatu individu dan tanggal kelahirannya. Seperti pada **Kode Sumber 5. 17**.

```
UPDATE Person SET hasBOD = '1897-07-15T09:20:00'
where PersonId = 4
```

Kode Sumber 5. 17 Fungsi Set *HasBOD*

Set hasFirstName

Fungsi ini digunakan untuk menyimpan nilai dari suatu *data property hasFirstName*, yaitu berisi id suatu individu dan tanggal kelahirannya. Seperti pada **Kode Sumber 5. 17**.

```
UPDATE Person SET hasBOD = '1897-07-15T09:20:00'
where PersonId = 4
```

Kode Sumber 5. 18 Fungsi Set *HasFirstName*

5.1.2. Fungsi pada Peng-query-an Data

Pada aplikasi yang dibangun ini, aplikasi mampu memetekan *syntax query* SPARQL ke dalam *syntax query* SQL

untuk kemudian diproses oleh sistem dan ditampilkan data hasil pengolahan *syntax query*. Aplikasi ini dibangun oleh 3 kelas yang masing-masing kelasnya terdiri dari berbarapa fungsi. Berikut kelas-kelas yang dibuat beserta fungsi-fungsinya:

5.1.2.1. Kelas MyDatabase

Pada kelas ini terdapat beberapa fungsi, dimana fungsi-fungsi tersebut nantinya akan digunakan dan dipanggil pada kelas main, fungsi-fungsi tersebut antara lain:

Fungsi Set Object property

Fungsi ini digunakan untuk menuliskan nama-nama *object property* beserta *domain* dan *range*-nya, sesuai dengan data yang ada pada ontologi pohon keluarga. Seperti pada Kode Sumber 5. 19.

```
private void setObjectProperty(){
        objectProperty.add("hasrelations");
        domain.add("person1");
        range.add("person2");
        objectProperty.add("hasbloodrelations");
        domain.add("person1");
        range.add("person2");
        objectProperty.add("hasaunt");
        domain.add("person");
        range.add("woman");
        objectProperty.add("hassister");
        domain.add("person");
        range.add("woman");
        objectProperty.add("hasbrother");
        domain.add("person");
        range.add("man");
              .......
              .......
        objectProperty.add("hashusband");
        domain.add("woman");
        range.add("man");
        objectProperty.add("haswife");
        domain.add("man");
        range.add("woman");}
```

Kode Sumber 5. 19 Fungsi *Set Object Property*

Fungsi Get Object property

Fungsi ini digunakan untuk mendapatkan nilai semua *object property* yang sudah di-*set* sebelumnya. Seperti pada Kode Sumber 5. 20.

```
public String getObjectProperty(int index){
        return objectProperty.get(index);
    }
```

Kode Sumber 5. 20 Fungsi *Get Object Property*

Fungsi Get Domain Object property

Fungsi ini digunakan untuk mendapatkan nilai *domain* dari semua *object property* yang sudah dituliskan sebelumnya. Seperti pada Kode Sumber 5. 21.

```
public String getDomain(String name){
        String dom = "";
        int length = objectProperty.size();
        for (int i=0; i<length; i++){
            if
(objectProperty.get(i).compareTo(name)==0){
                dom = domain.get(i);
            }
        }
        return dom;
}
```

Kode Sumber 5. 21 Fungsi *Get Domain Object Property*

Fungsi Get Range Object property

Fungsi ini digunakan untuk mendapatkan nilai *domain* dari semua *object property* yang sudah dituliskan sebelumnya. Seperti pada Kode Sumber 5. 22.

```
public String getRange(String name){
        String rang = "";
        int length = objectProperty.size();
        for (int i=0; i<length; i++){
            if
(objectProperty.get(i).compareTo(name)==0){
                rang = range.get(i);
            }
        }
        return rang;
    }
```

Kode Sumber 5. 22 Fungsi *Get Range Object Property*

Fungsi Is Object Property

Fungsi ini digunakan untuk mengecek apakah data inputan tersebut merupakan *object property* atau tidak. Seperti pada Kode Sumber 5. 23.

```
public Boolean isObjectProperty(String name){
        Boolean flag = false;
        int length = objectProperty.size();
        for (int i=0; i<length; i++){
            if
(objectProperty.get(i).compareTo(name)==0){
                flag =true;
            }
        }
        return flag;
    }
```

Kode Sumber 5. 23 Fungsi *Is Object Property*

Fungsi Set Data Property

Fungsi ini digunakan untuk menuliskan nama-nama *data property* beserta *domain* dan *range*-nya, sesuai dengan data yang ada pada ontologi pohon keluarga. Seperti pada Kode Sumber 5. 24.

```
private void setDataProperty(){
        dataProperty.add("borninyear");
        domainData.add("person");
        rangeData.add("integer");
        dataProperty.add("deadinyear");
        domainData.add("person");
        rangeData.add("integer");
                    .....
                    .....
                    .....
        domainData.add("person");
        rangeData.add("string");
        dataProperty.add("haslastname");
        domainData.add("person");
        rangeData.add("string");
        dataProperty.add("hasmariagename");
        domainData.add("person");
        rangeData.add("string");
        dataProperty.add("hassex");
        domainData.add("person");
        rangeData.add("string");
}
```

Kode Sumber 5. 24 Fungsi *Set Data Property*

Fungsi Get Data Property

Fungsi ini digunakan untuk mendapatkan nilai semua *data property* yang sudah dituliskan sebelumnya. Seperti pada Kode Sumber 5. 25.

```
public String getDataProperty(int index){
        return dataProperty.get(index);
    }
```

Kode Sumber 5. 25 Fungsi *Get Data Property*

Fungsi Get Domain Data Property

Fungsi ini digunakan untuk mendapatkan nilai *domain* dari semua *data property* yang sudah dituliskan sebelumnya. Seperti pada Kode Sumber 5. 26.

```
public String getDomainData(String name){
        String domDat = "";
        int length = dataProperty.size();
        for (int i=0; i<length; i++){
            if
(dataProperty.get(i).compareTo(name)==0){
                domDat = domainData.get(i);}}
        return domDat;
}
```

Kode Sumber 5. 26 Fungsi *Get Domain Data Property*

Fungsi Get Range Data Property

Fungsi ini digunakan untuk mendapatkan nilai *domain* dari semua *data property* yang sudah dituliskan sebelumnya. Seperti pada Kode Sumber 5. 27.

```
public String getRangeData(String name){
        String rangDat = "";
        int length = dataProperty.size();
        for (int i=0; i<length; i++){
            if
(dataProperty.get(i).compareTo(name)==0){
                rangDat = rangeData.get(i);}}
        return rangDat;}
}
```

Kode Sumber 5. 27 Fungsi *Get Range Data property*

Fungsi Is Data Property

Fungsi ini digunakan untuk mengecek apakah data inputan tersebut merupakan *data property* atau tidak. Seperti pada Kode Sumber 5. 28.

```
public Boolean isDataProperty(String name){
        Boolean flag = false;
        int length = dataProperty.size();
        for (int i=0; i<length; i++){
            if
(dataProperty.get(i).compareTo(name)==0){
                flag =true;
            }
```

Kode Sumber 5. 28 Fungsi *Is Data Property*

5.1.2.2. Kelas DBHandler

Terdapat beberapa fungsi pada kelas ini. Fungsi-fungsi tersebut adalah sebagai berikut:

Fungsi Connect

Fungsi ini digunakan untuk mengkoneksikan data yang ada pada PostgreSQL dengan aplikasi yang dibangun. Seperti pada Kode Sumber 5. 29.

```
public void connect() {

try {
         Class.forName("org.postgresql.Driver");
         con = DriverManager
            .getConnection("jdbc:postgresql://local
host:5432/DataTAFinal",
            "postgres", "Kamali12");
      }
catch (ClassNotFoundException | SQLException e) {

System.err.println(e.getClass().getName()+":
"+e.getMessage());
         System.exit(0);
      }

System.out.println("Opened database successfully");
   }
```

Kode Sumber 5. 29 Fungsi *Connect*

Fungsi Result Set Select

Fungsi ini digunakan untuk menampilkan data hasil pengolahan *syntax query* SQL yang ada. Seperti pada Kode Sumber 5. 30.

```
public ResultSet select(String sql)
     {
        ResultSet rs = null;
       Statement stmt = null;

       try {

       Class.forName("org.postgresql.Driver");
        con.setAutoCommit(false);
        stmt = (Statement) con.createStatement();
        rs = stmt.executeQuery( sql+";" );
       }
      catch ( ClassNotFoundException | SQLException
e ) {

         System.err.println(
e.getClass().getName()+": "+ e.getMessage() );
         System.exit(0);

       }

       return rs;
     }
```

Kode Sumber 5. 30 Fungsi *Result Set Select*

5.1.2.3. Kelas Main

Pada kelas main terdapat beberapa fungsi yang digunakan untuk memetakan *syntax query* SPARQL ke dalam *syntax query* SQL. Fungsi-fungsi pada kelas main adalah sebagai berikut:

Fungsi Case_001

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 31.

```
public String Case_001(String predicate){
        String sql =      "select personname, " +
predicate +"\n" +
                          "from person \n" +
                          "where "+predicate+" != 0" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 31 Fungsi Case_001

Fungsi Case_002

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 32.

```
public   String   Case_002(String   subject,   String
predicate){
        String sql =      "select personname, " +
predicate +"\n" +
                          "from person \n" +
                          "where   personname=   '"   +
subject + "'\n" +
                          "and "+predicate+" != 0" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 32 Fungsi Case_002

Fungsi Case_003

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 33.

```
public String Case_003(String predicate, String
object){
        String sql =      "select personname, " +
predicate + "\n" +
                          "from person \n" +
                          "where " + predicate + " = "
+ object + "\n" +
                          "and "+predicate+" != 0" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 33 Fungsi Case_003

Fungsi Case_004

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 34

```
public String Case_004(String predicate){
        String sql =      "select personname, " +
predicate +"\n" +
                          "from person \n" +
                          "where   "+predicate+"   !=
'Null'" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 34 Fungsi Case_004

Fungsi Case_005

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 35.

```
public String Case_005(String subject, String
predicate){
        String sql =      "select personname, " +
predicate +"\n" +
                          "from person \n" +
                          "where  personname=  '"  +
subject + "'\n" +
                          "and    "+predicate+"    !=
'Null'" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 35 Fungsi Case_005

Fungsi Case_006

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 36.

```
public String Case_006(String predicate, String
object){
        String sql =      "select personname, " +
predicate + "\n" +
                          "from person \n" +
                          "where " + predicate + " =
'" + object + "'\n" +
                          "and    "+predicate+"    !=
'Null'" +
                          "order by personname \n";
        return sql;
    }
```

Kode Sumber 5. 36 Fungsi Case_006

Fungsi Case _007

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 37.

```
public String Case_007(String predicate){
        String sql =   "select distinct personname ,
(select  distinct  person.personname  as  person"  +
predicate + " from person where person.personid= " +
myDatabase.getDomain(predicate)  +  "."  +  predicate
+")\n" +
                        "from       person,       "
+myDatabase.getDomain(predicate) + "\n" +
                        "where  person.personid=  "
+myDatabase.getDomain(predicate)       +       "."
+myDatabase.getDomain(predicate) +"id\n" +
                        "and "+ predicate +"!= 0\n"
+
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 37 Fungsi Case_007

Fungsi Case_008

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 38.

```
public String Case_008(String predicate){
        String sql =    "select distinct personname
\n" +
                        "from       person,       "
+myDatabase.getDomain(predicate) + "\n" +
                        "where  person.personid=  "
+myDatabase.getDomain(predicate)       +       "."
+myDatabase.getDomain(predicate) +"id\n" +
                        "and "+ predicate +"!= 0\n"
+
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 38 Fungsi Case_008

Fungsi Case_009

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 39.

```
public String Case_009(String predicate){
        String sql =    "select distinct personname
\n" +
                        "from       person,       "
+myDatabase.getDomain(predicate) + "\n" +
                        "where  person.personid=  "
+myDatabase.getDomain(predicate)       +       "."
+predicate+"\n" +
                        "and "+ predicate +"!= 0\n"
+
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 39 Fungsi Case_009

Fungsi Case_010

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 40.

```
public String Case_010(String subject, String
predicate){
        String sql =   "select distinct personname ,
(select distinct person.personname as person" +
predicate + " from person where person.personid= "
+myDatabase.getDomain(predicate)+"."+     predicate
+")\n" +
                        "from       person,       "
+myDatabase.getDomain(predicate) + "\n" +
                        "where   personname   =   '"
+subject+ "'\n" +
                        "and   person.personid=    "
+myDatabase.getDomain(predicate)       +       "."
+myDatabase.getDomain(predicate) +"id\n" +
                        "and "+ predicate +"!= 0\n";
        return sql;
```

Kode Sumber 5. 40 Fungsi Case_010

Fungsi Case_011

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 41.

```
public String Case_011(String predicate, String
object){
        String sql =     "select distinct (select
distinct   person.personname   from   person   where
person.personid=                "                +
myDatabase.getDomain(predicate)+       "."       +
myDatabase.getDomain(predicate) +"id), personname as
person" + predicate + "\n" +
                        "from       person,       "
+myDatabase.getDomain(predicate) +  "\n" +
                        "where   personname   =   '"
+object+ "'\n" +
                        "and   person.personid=    "
+myDatabase.getDomain(predicate) + "." + predicate +
"\n" +
                        "and "+ predicate +"!= 0\n";
        return sql;
```

Kode Sumber 5. 41 Fungsi Case_011

Fungsi Case_012

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 42.

```
public String Case_012(String predicate){
        String sql =   "select distinct personname ,
(select distinct person.personname as person" +
predicate     +     "     from     person     where
person.personid=person"+       predicate       +"."+
myDatabase.getRange(predicate) +"id)\n" +
                        "from   person,   person"+
predicate +"\n" +
                        "where
person.personid=person"+       predicate       +"."+
myDatabase.getDomain(predicate) +"id\n" +
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 42 Fungsi Case_012

Fungsi Case_013

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 43.

```
public String Case_013(String predicate){
        String sql =    "select distinct personname
\n" +
                        "from   person,   person"+
predicate +"\n" +
                        "where
person.personid=person"+       predicate       +"."+
myDatabase.getDomain(predicate) +"id\n" +
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 43 Fungsi Case_013

Fungsi Case_014

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 44.

```
public String Case_014(String predicate){
        String sql =    "select distinct personname
\n" +
                        "from    person,    person"+
predicate +"\n" +
                        "where
person.personid=person"+       predicate       +"."+
myDatabase.getRange(predicate) +"id\n" +
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 44 Fungsi Case_014

Fungsi Case_015

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 45.

```
public   String   Case_015(String   subject,   String
predicate){
        String sql =    "select distinct personname ,
(select  distinct  person.personname  as  person"  +
predicate     +     "     from     person     where
person.personid=person"+       predicate       +"."+
myDatabase.getRange(predicate) +"id)\n" +
                        "from    person,    person"+
predicate + "\n" +
                        "where   personname   =   '"
+subject+ "'\n" +
                        "and
person.personid=person"+       predicate       +"."+
myDatabase.getDomain(predicate) +"id\n" +
                        "order by personname";
        return sql;
```

Kode Sumber 5. 45 Fungsi Case_015

Fungsi Case_016

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 46.

```
public String Case_016(String predicate, String
object){
        String sql =      "select distinct (select
distinct person.personname from person where
person.personid=person"+      predicate      +"."+
myDatabase.getDomain(predicate) +"id), personname as
person" + predicate + "\n" +
                        "from   person,   person"+
predicate + "\n" +
                        "where   personname   =   '"
+object+ "'\n" +
                        "and
person.personid=person"+      predicate      +"."+
myDatabase.getRange(predicate) +"id\n" +
                        "order by personname";
        return sql;
    }
```

Kode Sumber 5. 46 Fungsi Case_016

Fungsi Case_017

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 47.

```
public String Case_017(String subject, String tmp){
        String sql = null;
        if (myDatabase.isDataProperty(tmp)){
            if
(myDatabase.getRangeData(tmp).equals("integer")){
                sql =   "select personname, " + tmp
+ "\n" +
                        "from person \n" +
                        "where  personname=  '"  +
subject + "'\n" +
                        "and "+tmp+" != 0" +
                        "order by personname \n";
            }
            else{
                sql =   "select personname, " + tmp
+ "\n" +
                        "from person \n" +
                        "where  personname=  '"  +
subject + "'\n" +
                        "and "+tmp+" != 'Null'" +
                        "order by personname \n";
            }
        }
        else if (myDatabase.isObjectProperty(tmp)){
            if
(myDatabase.getDomain(tmp).equals("woman")||myDatab
ase.getDomain(tmp).equals("man")){
                sql =  "select distinct personname ,
(select distinct person.personname as person" + tmp
+   "   from   person   where   person.personid=   "
+myDatabase.getDomain(tmp)+"."+ tmp +")\n" +
                        "from       person,       "
+myDatabase.getDomain(tmp) +"\n" +
                        "where   personname   =   '"
+subject+ "'\n" +
                        "and "+ tmp +" != 0\n" +
                        "and   person.personid=   "
+myDatabase.getDomain(tmp)          +          "."
+myDatabase.getDomain(tmp) +"id\n" +
                        "order by personname";
            }
             ..........
             ..........
```

```
              ..........
              ..........
              ..........

              else{
                 sql =  "select distinct personname ,
(select distinct person.personname as person" + tmp
+ " from person where person.personid=person"+ tmp
+"."+ myDatabase.getRange(tmp) +"id)\n" +
                        "from person, person"+ tmp +
"\n" +
                        "where   personname   =   '"
+subject+ "'\n" +
                        "and
person.personid=person"+          tmp          +"."+
myDatabase.getDomain(tmp) +"id\n" +
                        "order by personname";
            }
        }
        return sql;
    }
```

Kode Sumber 5. 47 Fungsi Case_017

Fungsi Case_018

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 48.

```
public String Case_018(String object, String tmp){
        String sql = null;

        if (myDatabase.isDataProperty(tmp)){

if(myDatabase.getRangeData(tmp).equals("integer")){
                sql =   "select personname, " + tmp
+ "\n" +
                        "from person \n" +
                        "where " + tmp + " = " +
object + "\n" +
                        "order by personname;";
            }
        }

        if (myDatabase.isDataProperty(tmp)){

if(!myDatabase.getRangeData(tmp).equals("integer"))
{
                sql =   "select personname, " + tmp
+ "\n" +
                        "from person \n" +
                        "where " + tmp + " = '" +
object + "'\n" +
                        "order by personname;";
            }
        }

        else{
            if
(myDatabase.getDomain(tmp).equals("woman")||myDatab
ase.getDomain(tmp).equals("man")){
                sql =    "select distinct (select
distinct   person.personname   from   person   where
person.personid= " + myDatabase.getDomain(tmp)+ "."
+  myDatabase.getDomain(tmp)  +"id),  personname  as
person" + tmp + "\n" +
                                "from     person,     "
+myDatabase.getDomain(tmp) +  "\n" +
                                "where  personname  =  '"
+object+ "'\n" +
                                "and "+ tmp +" != 0\n" +
                    .......
                    .......
```

```
                    .......
                    .......

                           "and  person.personid=  "
                    +myDatabase.getDomain(tmp)     +
                    "." + tmp + "\n" +
                            "order by personname;";
            }
            else{
                sql =     "select distinct (select
distinct   person.personname   from   person   where
person.personid=person"+          tmp          +"."+
myDatabase.getDomain(tmp)   +"id),   personname   as
person" + tmp + "\n" +
                            "from person, person"+
tmp + "\n" +
                            "where personname = '"
+object+ "'\n" +
                            "and
person.personid=person"+          tmp          +"."+
myDatabase.getRange(tmp) +"id\n" +
                            "order by personname;";
            }
        }

        return sql;
    }
```

Kode Sumber 5. 48 Fungsi Case_018

Fungsi Case_019

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 49.

```
public String Case_019(String tmp){
        String sql = null;
        if (myDatabase.isDataProperty(tmp)){
            if
(myDatabase.getRangeData(tmp).equals("integer")){
                sql =   "select personname, " + tmp
+"\n" +
                        "from person \n" +
                        "and "+tmp+" != 0" +
                        "order by personname \n";
            }
            else{
                sql =   "select personname, " + tmp
+"\n" +
                        "from person \n" +
                        "and "+tmp+" != 'Null'" +
                        "order by personname \n";
            }
        }
        else if (myDatabase.isObjectProperty(tmp)){
            if
(myDatabase.getDomain(tmp).equals("woman")||myDatab
ase.getDomain(tmp).equals("man")){
                sql =  "select distinct personname ,
(select distinct person.personname as person" + tmp
+  "  from  person  where  person.personid=  "  +
myDatabase.getDomain(tmp) + "." + tmp +")\n" +
                        "from       person,       "
+myDatabase.getDomain(tmp) + "\n" +
                        "where  person.personid=  "
+myDatabase.getDomain(tmp)          +          "."
+myDatabase.getDomain(tmp) +"id\n" +
                        "and "+ tmp +" != 0\n" +
                        "order by personname";
            }
                    .......
                    .......
                    .......
```

```
                    .......
                    .......
             else{
                sql =  "select distinct personname ,
(select distinct person.personname as person" + tmp
+ " from person where person.personid=person"+ tmp
+"."+ myDatabase.getRange(tmp) +"id)\n" +
                        "from person, person"+ tmp
+"\n" +
                        "where
person.personid=person"+          tmp          +"."+
myDatabase.getDomain(tmp) +"id\n" +
                        "order by personname";
            }
        }
```

Kode Sumber 5. 49 Fungsi Case_019

Fungsi Case_020

Fungsi ini berjalan ketika kondisinya terpenuhi sesuai dengan yang dijelaskan pada Subbab 4.2. Fungsi tersebut dapat dilihat pada Kode Sumber 5. 50

```
public  String  Case_020(String  subject,  String
object, String tmp){
        String sql = null;


        if (myDatabase.isDataProperty(tmp)){
            if
(myDatabase.getRangeData(tmp).equals("integer")){
                sql  =        "select  personname,
"+tmp+"\n" +
                        "from person\n" +
                        "where  personname  =  '"
+subject+ "'\n" +
                        "and  "+tmp+"=  "  +object+
"\n" +
                        "order by personname;";
            }
        }
                    .......
                    .......
```

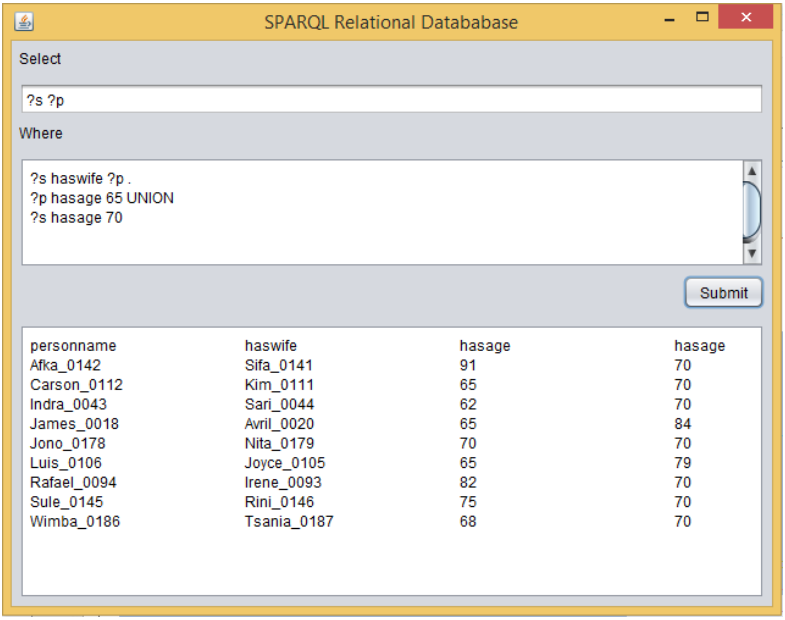
```
                    .......
                    .......
if (myDatabase.isDataProperty(tmp)){
            if
(!myDatabase.getRangeData(tmp).equals("integer")){
                sql   =        "select   personname,
"+tmp+"\n" +
                          "from person\n" +
                          "where   personname   =   '"
+subject+ "'\n" +
                          "and  "+tmp+"=  '"  +object+
"'\n" +;
            }
        }
        else if (myDatabase.isObjectProperty(tmp)){
            if
(myDatabase.getDomain(tmp).equals("woman")||myDatab
ase.getDomain(tmp).equals("man")){
                sql =  "select distinct personname ,
(select distinct person.personname as person"+tmp+"
from       person       where       person.personid=
"+myDatabase.getDomain(tmp)+"."+tmp+")\n" +
                          "from       person,       "
+myDatabase.getDomain(tmp)+ "\n" +
                          "where     personname     =
'"+subject+"'\n" +
                          "and       person.personid=
"+myDatabase.getDomain(tmp)+"."+myDatabase.getDomai
n(tmp)+"id\n" +
                          "and    "+tmp+"=    (select
personid   from   person   where   personname   =
'"+object+"')\n" +
                          "and "+tmp+" != 0\n" ;
            }
            else{
                sql =  "select distinct personname ,
(select distinct person.personname as person"+tmp+"
from                person                  where
person.personid=person"+tmp+"."+myDatabase.getRange
(tmp)+"id)\n" +
                          "from               person,
person"+tmp+"\n" +
                    .......
                    .......
```

```
                    .......
                    .......

                    "where       personname       =
            '"+subject+"'\n" +
                        "and
person.personid=person"+tmp+"."+myDatabase.getDomai
n(tmp)+"id\n" +
                        "and
person"+tmp+"."+myDatabase.getRange(tmp)+"id      =
(select personid from person where personname=
'"+object+"')\n" +
                        "order by personname;";
            }
        }

        return sql;
    }
```

Kode Sumber 5. 50 Fungsi Case_020

5.2. Implementasi Antarmuka Pengguna

Implementasi tampilan antarmuka pengguna pada *dekstop* dilakukan dengan menggunakan dukungan aplikasi NetBeans. NetBeans berfungsi untuk membuat suatu aplikasi *dekstop* dan menghubungkan aplikasi *dekstop* tersebut dengan data yang ada pada PostgreSQL. Implementasi tampilan antarmuka pada aplikasi ini hanya ada satu. Halaman tersebut merupakan implementasi halaman utama dari rancangan antarmuka "SPARQL Relational Database" yang telah dijelaskan pada Subbab 4.2. Halaman utama ini menyediakan kolom tempat penulisan *query* SPARQL, tombol "Submit" dan kolom hasil peng-*query*-an. Tampilan antarmuka halaman utama ini dapat dilihat pada Gambar 5. 2.



Gambar 5. 2 Implementasi Antarmuka Halaman Utama

(Halaman ini sengaja dikosongkan)

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada ontologi yang dikembangkan. Pengujian yang dilakukan adalah pengujian penyimpanan basis data relasional hasil transformasi ontologi, pengujian peng-*query*-an suatu data, dan pengujian waktu pemrosesan *query* SPARQL. Pengujian penyimpanan basis data relasional hasil transformasi ontologi mengacu pada Subbab 3.2.3, sedangkan pengujian peng-*query*-an suatu data mengacu pada subbab 3.2.4. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

6.1. Lingkungan Pengujian

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

Prosesor	: Intel Core i5-4200M CPU @ 2.50GHz
Memori	: 4.00 GB
Jenis *Device*	: Laptop
Sistem Operasi	: Microsoft Windows Embedded 8.1 Industry Pro
Protégé	: Protégé 4.0 dan Protégé 4.3
Reasoner	: Pellet dan OWL2RDB
NetBeans	: NetBeans IDE 8.1
PostgreSQL	: PostgreSQL 9.5

6.2. Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian penyimpanan dilakukan dengan melihat hasil penyimpanan berupa tabel-tabel beserta isinya yang ada pada PostgreSQL. Selanjutnya pengujian peng-*query*-an data dilakukan dengan melakukan *query* SPARQL terhadap semua pola yang telah dibangun pada aplikasi ini. Sedangkan pengujian waktu

pemrosesan *query* SPARQL dilakukan dengan menghitung waktu yang dibutuhkan untuk melakukan *query* SPARQL.

6.2.1. Pengujian Penyimpanan Basis Data Relasional Hasil Transformasi OWL2RDB

Pengujian penyimpanan basis data relasional hasil *transformasi* OWL2RDB merupakan tahap uji penyimpanan data pada PostgreSQL. Pengujian dilakukan secara manual dengan menjalankan *syntax query* SQL hasil *transformasi* pada SQL Editor PostgreSQL, kemudian melihat data yang berhasil disimpan pada PostgreSQL sebagai tolak ukur keberhasilan pengujian. Dimana dalam melakukan penyimpanan data pada SQL Editor PostgreSQL ada urutannya tersendiri. Urutan dalam menjalankan penyimpanan tersebut adalah sebagai berikut:

- Tabel hasil transformasi ontologi suatu *class.*
- Tabel hasil transformasi ontologi suatu *object property.*
- Kolom hasil transformasi ontologi suatu *data property.*
- Isi tabel dan kolom hasil transformasi ontologi suatu *member class, domain* dan *range object property, dan domain dan range data property.*

6.2.1.1. Pengujian Penyimpanan Tabel Hasil Transformasi Ontologi suatu Class

Terdapat 3 tabel hasil transformasi ontologi suatu *class*, yaitu tabel *person*, *man*, dan *woman*. Berikut hasil pengujian penyimpanan tabel-tabel tesebut:

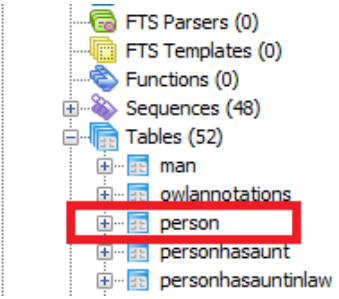
Pengujian Penyimpanan Tabel Person

Pada tahap pengujian penyimpanan tabel *person*, dilakukan pengecekan apakah tabel *person* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *person* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 1.

Tabel 6. 1 Pengujian Penyimpanan Tabel *Person*

ID	TA-UJ.BDC0001
Nama	Pengujian penyimpanan tabel *person*.
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *person* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *person*.
Kondisi Awal	Terdapat *syntax* pembuatan tabel *person* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	1. Pengguna membuka aplikasi PostgreSQL 2. Pengguna menjalankan *syntax* SQL pembutan tabel *person* pada SQL Editor. 3. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel *person* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *person*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *person*.

Untuk hasil pengujian penyimpanan tabel *person* dapat dilihat pada Gambar 6. 1. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *person* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *person* telah berhasil.



Gambar 6. 1 Uji Coba Penyimpanan Tabel *Person*

Pengujian Penyimpanan Tabel Man

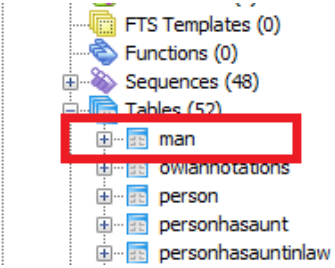
Pada tahap pengujian penyimpanan tabel *man*, dilakukan pengecekan apakah tabel *man* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *man* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 2.

Tabel 6. 2 Pengujian Penyimpanan Tabel *Man*

ID	TA-UJ.BDC0002
Nama	Pengujian penyimpanan tabel *man.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *man* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *man.*
Kondisi Awal	Terdapat *syntax* pembuatan tabel *man* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	1. Pengguna membuka aplikasi PostgreSQL 2. Pengguna menjalankan *syntax* SQL pembutan tabel *man* pada SQL Editor. 3. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel man sudah ada atau belum.

Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *man*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *man*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *man*.

Untuk hasil pengujian penyimpanan tabel *man* dapat dilihat pada Gambar 6. 2. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *man* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *man* telah berhasil.



Gambar 6. 2 Uji Coba Penyimpanan Tabel *Man*

Pengujian Penyimpanan Tabel Woman

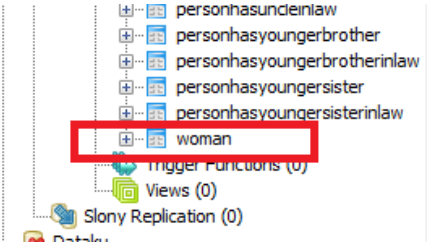
Pada tahap pengujian penyimpanan tabel *woman*, dilakukan pengecekan apakah tabel *woman* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *woman* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 3.

Tabel 6. 3 Pengujian Penyimpanan Tabel *Woman*

ID	TA-UJ.BDC0003
Nama	Pengujian penyimpanan tabel *woman*.

Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *woman* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *woman*.
Kondisi Awal	Terdapat *syntax* pembuatan tabel *woman* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	4. Pengguna membuka aplikasi PostgreSQL 5. Pengguna menjalankan *syntax* SQL pembutan tabel *woman* pada SQL Editor. 6. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel *woman* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *woman*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *woman*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *woman*.

Untuk hasil pengujian penyimpanan tabel *woman* dapat dilihat pada Gambar 6. 3. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *woman* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *woman* telah berhasil.



Gambar 6. 3 Uji Coba Penyimpanan Tabel *Woman*

6.2.1.2. Pengujian Penyimpanan Tabel Hasil Transformasi Ontologi suatu Object Property

Terdapat 48 tabel hasil transformasi ontologi suatu *object property*, seperti *personhasrelations*, *personhasbloodrelations*, *personhasaunt*, dll. Berikut beberapa hasil pengujian penyimpanan tabel-tabel tesebut:

Pengujian Penyimpanan Tabel Personhasrelations

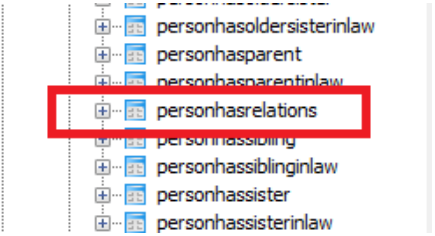
Pada tahap pengujian penyimpanan tabel *personhasrelations*, dilakukan pengecekan apakah tabel *personhasrelations* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasrelations* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 4.

Tabel 6. 4 Pengujian Penyimpanan Tabel *Personhasrelations*

ID	TA-UJ.BDO0001
Nama	Pengujian penyimpanan tabel *personhasrelations.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasrelations* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *personhasrelations*.
Kondisi Awal	Terdapat *syntax* pembuatan tabel *personhasrelations* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	1. Pengguna membuka aplikasi PostgreSQL 2. Pengguna menjalankan *syntax* SQL pembutan tabel *personhasrelations* pada SQL Editor. 3. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel *personhasrelations* sudah ada atau belum.

Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasrelations*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *personhasrelations*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasrelations*.

Untuk hasil pengujian penyimpanan tabel *personhasrelations* dapat dilihat pada Gambar 6. 4. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *personhasrelations* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *personhasrelations* telah berhasil.



Gambar 6. 4 Uji Coba Penyimpanan Tabel *Personhasrelations*

Pengujian Penyimpanan Tabel Personhasbloodrelations

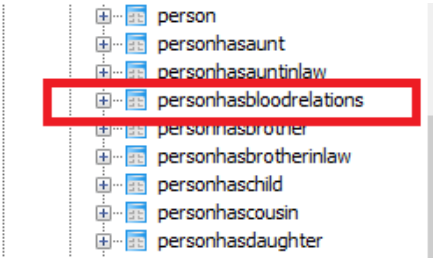
Pada tahap pengujian penyimpanan tabel *personhasbloodrelations*, dilakukan pengecekan apakah tabel *personhasbloodrelations* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasbloodrelations* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 5.

Tabel 6. 5 Pengujian Penyimpanan Tabel *Personhasbloodrelations*

ID	TA-UJ.BDO0002

Nama	Pengujian penyimpanan tabel *personhasbloodrelations.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasbloodrelations* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *personhasbloodrelations*.
Kondisi Awal	Terdapat *syntax* pembuatan tabel *personhasbloodrelations* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	1. Pengguna membuka aplikasi PostgreSQL 2. Pengguna menjalankan *syntax* SQL pembutan tabel *personhasbloodrelations* pada SQL Editor. 3. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel *personhasbloodrelations* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasbloodrelations*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *personhasbloodrelations*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasbloodrelations*.

Untuk hasil pengujian penyimpanan tabel *personhasbloodrelations* dapat dilihat pada Gambar 6. 5. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *personhasbloodrelations* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *personhasbloodrelations* telah berhasil.



Gambar 6. 5 Uji Coba Penyimpanan Tabel *Personhasbloodrelations*

Pengujian Penyimpanan Tabel Personhasaunt

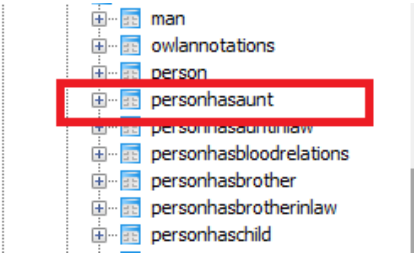
Pada tahap pengujian penyimpanan tabel *personhasaunt*, dilakukan pengecekan apakah tabel *personhasaunt* berhasil dibangun pada *database* PostgreSQL atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasaunt* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 6.

Tabel 6. 6 Pengujian Penyimpanan Tabel *Personhasaunt*

ID	TA-UJ.BDO0003
Nama	Pengujian penyimpanan tabel *personhasaunt.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan tabel *personhasaunt* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu tabel *personhasaunt*.
Kondisi Awal	Terdapat *syntax* pembuatan tabel *personhasaunt* sesuai dengan hasil transformasi OWL2RDB.
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	4. Pengguna membuka aplikasi PostgreSQL 5. Pengguna menjalankan *syntax* SQL pembutan tabel *personhasaunt* pada SQL Editor. 6. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan untuk melihat apakah tabel *personhasaunt* sudah ada atau belum.

Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasaunt*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan tabel *personhasaunt.*
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan tabel *personhasaunt*.

Untuk hasil pengujian penyimpanan tabel *personhasaunt* dapat dilihat pada Gambar 6. 6. Pada gambar tersebut dapat dilihat bahwa ketika uji coba dilakukan maka tabel *personhasaunt* berhasil disimpan pada *database* PostgreSQL. Hasil ini membuktikan bahwa uji coba penyimpanan tabel *personhasaunt* telah berhasil.



Gambar 6. 6 Uji Coba Penyimpanan Tabel *Personhasaunt*

6.2.1.3. Pengujian Penyimpanan Kolom Hasil Transformasi Ontologi suatu Data Property

Semua *data property* (9 data) pada ontologi pohon keluarga mempunyai *characteristic property* berupa *functional*. Jadi *data property* tersebut tidak menjadi suatu tabel sendiri-sendiri, melainkan menjadi kolom suatu tabel sesuai nama *domain data property* tersebut. Berikut beberapa uji coba penyimpanan kolom-kolom hasil transformasi ontologi suatu *data property* tersebut:

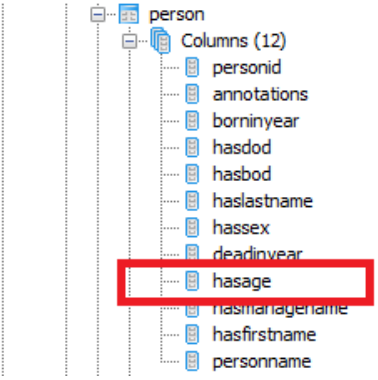
Pengujian Penyimpanan Kolom Hasage

Pada tahap pengujian penyimpanan kolom *hasage*, dilakukan pengecekan apakah kolom *hasage* berhasil dibangun pada tabel sesuai dengan nama *domain*-nya (*person*) atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasage* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 7.

Tabel 6. 7 Pengujian Penyimpanan Kolom *Hasage*

ID	TA-UJ.BDD0001
Nama	Pengujian penyimpanan kolom *hasage.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasage* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu kolom *hasage*.
Kondisi Awal	Terdapat *syntax* pembuatan kolom *hasage* sesuai dengan hasil transformasi OWL2RDB
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	1. Pengguna membuka aplikasi PostgreSQL 2. Pengguna menjalankan *syntax* SQL pembutan kolom *hasage* pada SQL Editor. 3. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan, kemudian memilih tabel *person* untuk melihat apakah kolom *hasage* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasage*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan kolom *hasage*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasage*.

Untuk hasil pengujian penyimpanan kolom *hasage* dapat dilihat pada Gambar 6. 7. Pada gambar tersebut dapat dilihat bahwa ketika uji coba telah dilakukan maka kolom *hasage* berhasil disimpan pada tabel *person*. Hasil ini membuktikan bahwa uji coba penyimpanan kolom *hasage* telah berhasil.



Gambar 6. 7 Uji Coba Penyimpanan Kolom *hasage*

Pengujian Penyimpanan Kolom Hasbod

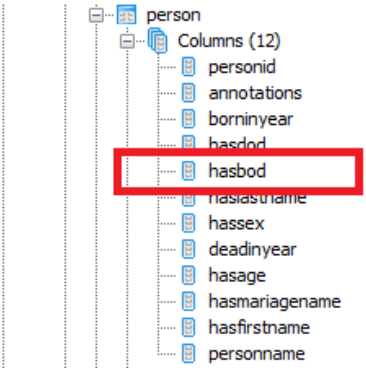
Pada tahap pengujian penyimpanan kolom *hasbod*, dilakukan pengecekan apakah kolom *hasbod* berhasil dibangun pada tabel sesuai dengan nama *domain*-nya (*person*) atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasbod* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 8.

Tabel 6. 8 Pengujian Penyimpanan Kolom *Hasbod*

ID	TA-UJ.BDD0002
Nama	Pengujian penyimpanan kolom *hasbod.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasbod* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu kolom *hasbod*.

Kondisi Awal	Terdapat *syntax* pembuatan kolom *hasbod* sesuai dengan hasil transformasi OWL2RDB
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	4. Pengguna membuka aplikasi PostgreSQL 5. Pengguna menjalankan *syntax* SQL pembutan kolom *hasbod* pada SQL Editor. 6. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan, kemudian memilih tabel *person* untuk melihat apakah kolom *hasbod* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasbod*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan kolom *hasbod*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasbod*.

Untuk hasil pengujian penyimpanan kolom *hasbod* dapat dilihat pada Gambar 6. 8. Pada gambar tersebut dapat dilihat bahwa ketika uji coba telah dilakukan maka kolom *hasbod* berhasil disimpan pada tabel *person*. Hasil ini membuktikan bahwa uji coba penyimpanan kolom *hasbod* telah berhasil.



Gambar 6. 8 Uji Coba Penyimpanan Kolom *hasbod*

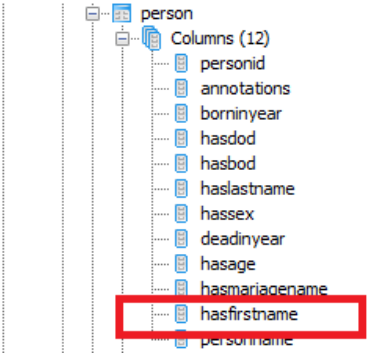
Pengujian Penyimpanan Kolom Hasfirstname

Pada tahap pengujian penyimpanan kolom *hasfirstname*, dilakukan pengecekan apakah kolom *hasfirstname* berhasil dibangun pada tabel sesuai dengan nama *domain*-nya (*person*) atau belum. Tujuannya adalah untuk mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasfirstname* berjalan atau tidak. Rincian pengujian ini dapat dilihat pada Tabel 6. 9.

Tabel 6. 9 Pengujian Penyimpanan Kolom *Hasfirstname*

ID	TA-UJ.BDD0003
Nama	Pengujian penyimpanan kolom *hasfirstname.*
Tujuan Pengujian	Mengetahui apakah *syntax* SQL hasil transformasi OWL2RDB berupa pembuatan kolom *hasfirstname* berjalan atau tidak.
Skenario 1	Menjalankan *syntax* SQL hasil dari transformasi OWL2RDB berupa pembuatan suatu kolom *hasfirstname.*
Kondisi Awal	Terdapat *syntax* pembuatan kolom *hasfirstname* sesuai dengan hasil transformasi OWL2RDB
Data Uji	Data uji merupakan data ontologi pohon keluarga yang telah ditransformasikan ke dalam *syntax* SQL.
Langkah Pengujian	7. Pengguna membuka aplikasi PostgreSQL 8. Pengguna menjalankan *syntax* SQL pembutan kolom *hasfirstname* pada SQL Editor. 9. Pengguna membuka *database* sesuai dengan nama *database* yang digunakan, kemudian memilih tabel *person* untuk melihat apakah kolom *hasfirstname* sudah ada atau belum.
Hasil Yang Diharapkan	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasfirstname*.
Hasil Yang Didapat	*Database* pada PostgreSQL sudah menyimpan kolom *hasfirstname.*
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pada PostgreSQL, seharusnya sudah berhasil menyimpan kolom *hasfirstname*.

Untuk hasil pengujian penyimpanan kolom *hasfirstname* dapat dilihat pada Gambar 6. 9. Pada gambar tersebut dapat dilihat bahwa ketika uji coba telah dilakukan maka kolom *hasfirstname* berhasil disimpan pada tabel *person*. Hasil ini membuktikan bahwa uji coba penyimpanan kolom *hasfirstname* telah berhasil.



Gambar 6. 9 Uji Coba Penyimpanan Kolom *hasfirstname*

6.2.2. Pengujian Peng-query-an Data

Pengujian peng-*query*-an data merupakan tahap uji setelah data berhasil disimpan pada PostgreSQL. Pengujian ini bertujuan untuk melihat apakah semua pola yang dibangun pada aplikasi ini telah berhasil atau belum.

6.2.2.1. Pengujian Peng-query-an Data Case_001

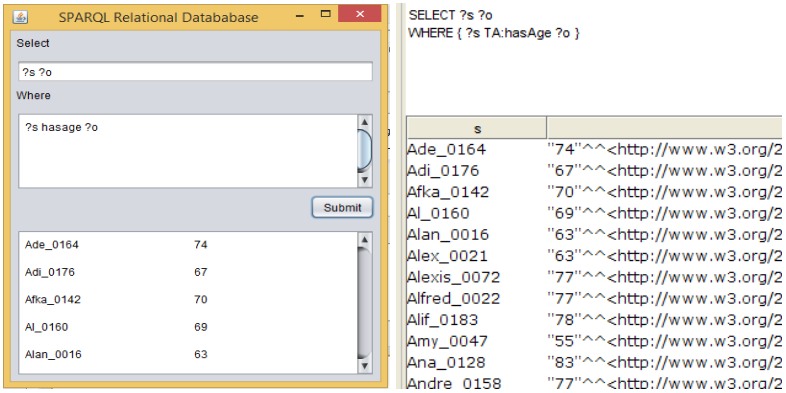
Pada tahap pengujian peng-*query*-an data Case_001, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_001 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 10.

Tabel 6. 10 Pengujian Peng-*query*-an Case_001

ID	TA-Q.BD0001
Nama	Pengujian peng-*query*-an data Case_001.

Tujuan Pengujian	Menguji apakah Case_001 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_001, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_001. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_001 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_001 berupa data *person* dan *data property*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_001 berupa data *person* dan *data property*.

Untuk hasil pengujian peng-*query*-an data Case_001 dapat dilihat pada Gambar 6. 10. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_001, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_001 telah berhasil.



Gambar 6. 10 Uji Coba Peng-*query*-an Case_001

6.2.2.2. Pengujian Peng-query-an Data Case_002

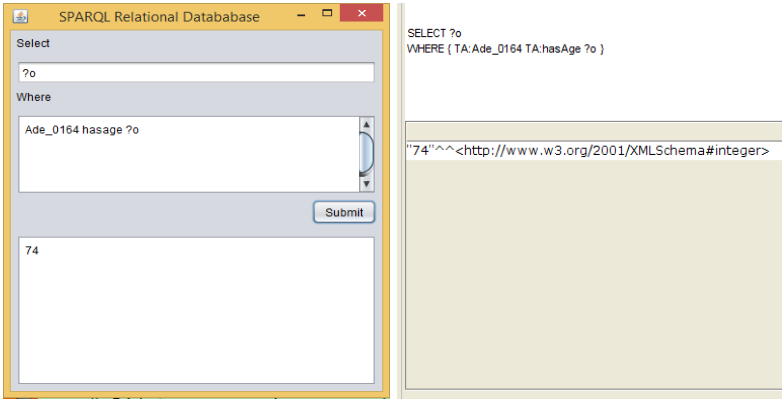
Pada tahap pengujian peng-*query*-an data Case_002, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_002 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 11.

Tabel 6. 11 Pengujian Peng-*query*-an Case_002

ID	TA-Q.BD0005
Nama	Pengujian peng-*query*-an data Case_002.
Tujuan Pengujian	Menguji apakah Case_002berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_002, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_002. 2. Pengguna menekan tombol “submit”.

	3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_002 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_002 berupa *data property*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_002 berupa *data property*.

Untuk hasil pengujian peng-*query*-an data Case_002 dapat dilihat pada Gambar 6. 11. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_002, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_002 telah berhasil.



Gambar 6. 11 Uji Coba Peng-*query*-an Case_002

6.2.2.3. Pengujian Peng-query-an Data Case_003

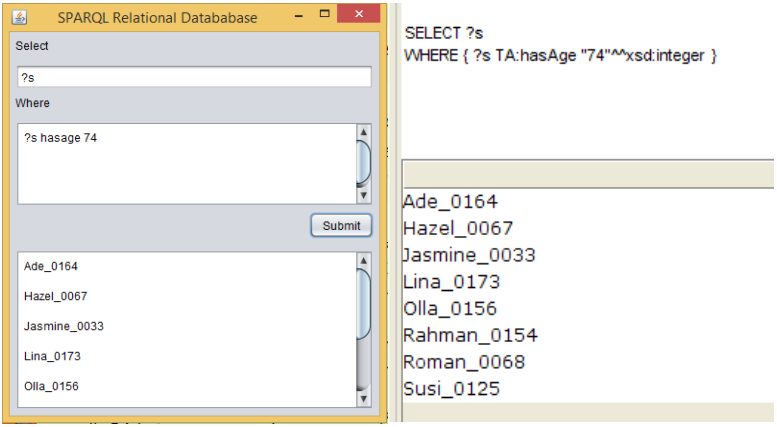
Pada tahap pengujian peng-*query*-an data Case_003, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_003 ini berfungsi untuk memetakan *syntax query*

SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 12.

Tabel 6. 12 Pengujian Peng-*query*-an Case_003

ID	TA-Q.BD0006
Nama	Pengujian peng-*query*-an data Case_003.
Tujuan Pengujian	Menguji apakah Case_003berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_003, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_003. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_003 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_003 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_003 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_003 dapat dilihat pada Gambar 6. 12. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_003, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_003 telah berhasil.



Gambar 6. 12 Uji Coba Peng-*query*-an Case_003

6.2.2.4. Pengujian Peng-query-an Data Case_004

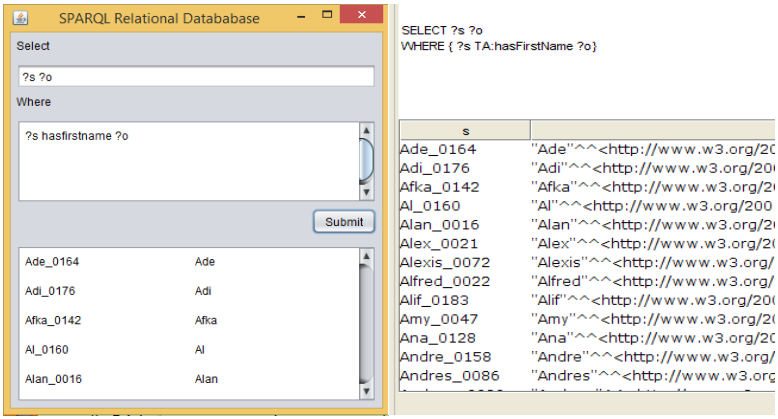
Pada tahap pengujian peng-*query*-an data Case_004, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_004 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 13.

Tabel 6. 13 Pengujian Peng-*query*-an Case_004

ID	TA-Q.BD0007
Nama	Pengujian peng-*query*-an data Case_004.
Tujuan Pengujian	Menguji apakah Case_004 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_004, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_004.

	2. Pengguna menekan tombol “submit”. 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_004 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_004 berupa data *person* dan *data property*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_004 berupa data *person* dan *data property*.

Untuk hasil pengujian peng-*query*-an data Case_004 dapat dilihat pada Gambar 6. 13. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_004, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_004 telah berhasil.



Gambar 6. 13 Uji Coba Peng-*query*-an Case_004

6.2.2.5. Pengujian Peng-query-an Data Case_005

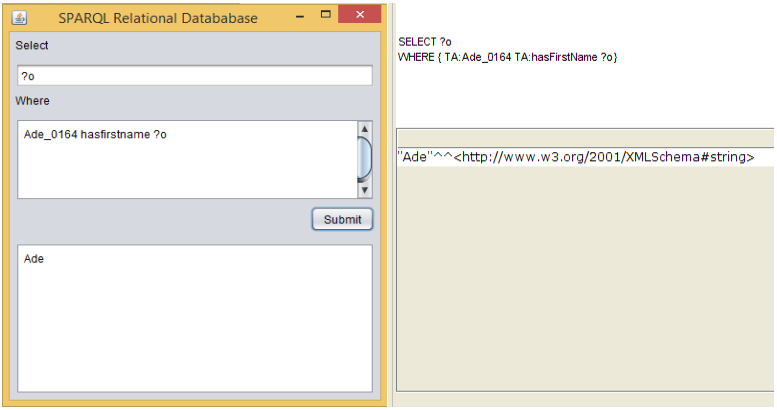
Pada tahap pengujian peng-*query*-an data Case_005, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada

Subbab 4.2. Case_005 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQLdan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 14.

Tabel 6. 14 Pengujian Peng-*query*-an Case_005

ID	TA-Q.BD00011
Nama	Pengujian peng-*query*-an data Case_005.
Tujuan Pengujian	Menguji apakah Case_005 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_005, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_005. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_005 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_005 berupa *data property*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_005 berupa *data property*.

Untuk hasil pengujian peng-*query*-an data Case_005 dapat dilihat pada Gambar 6. 14. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_005, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_005 telah berhasil.



Gambar 6. 14 Uji Coba Peng-*query*-an Case_005

6.2.2.6. Pengujian Peng-query-an Data Case_006

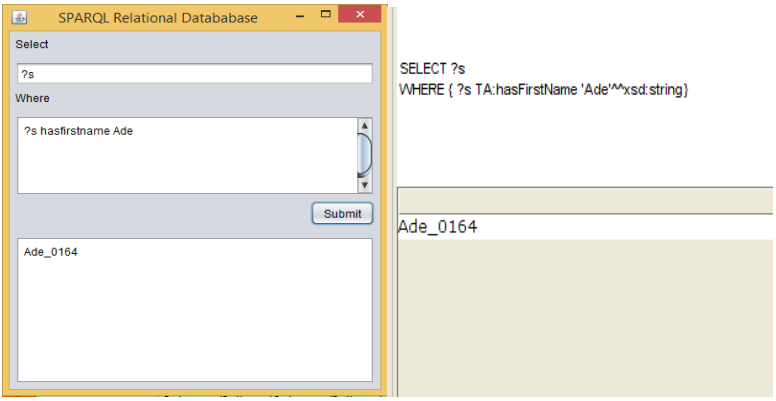
Pada tahap pengujian peng-*query*-an data Case_006, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_006 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan syntax tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 15.

Tabel 6. 15 Pengujian Peng-*query*-an Case_006

ID	TA-Q.BD0012
Nama	Pengujian peng-*query*-an data Case_006.
Tujuan Pengujian	Menguji apakah Case_006 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_006, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_006. 2. Pengguna menekan tombol “submit”.

	3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_006 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_006 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_006 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_006 dapat dilihat pada Gambar 6. 15. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_006 , maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_006 telah berhasil.



Gambar 6. 15 Uji Coba Peng-*query*-an Case_006

6.2.2.7. Pengujian Peng-query-an Data Case_007

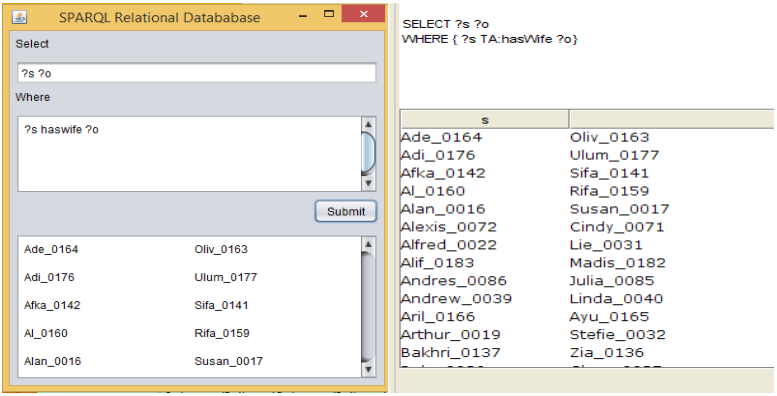
Pada tahap pengujian peng-*query*-an data Case_007, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_007 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil

pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 16

Tabel 6. 16 Pengujian Peng-*query*-an Case_007

ID	TA-Q.BD0013
Nama	Pengujian peng-*query*-an data Case_007.
Tujuan Pengujian	Menguji apakah Case_007 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_007, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_007. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_007 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_007 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_007 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_007 dapat dilihat pada Gambar 6. 16. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_007, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_007 telah berhasil.



Gambar 6. 16 Uji Coba Peng-*query*-an Case_007

6.2.2.8. Pengujian Peng-query-an Data Case_008

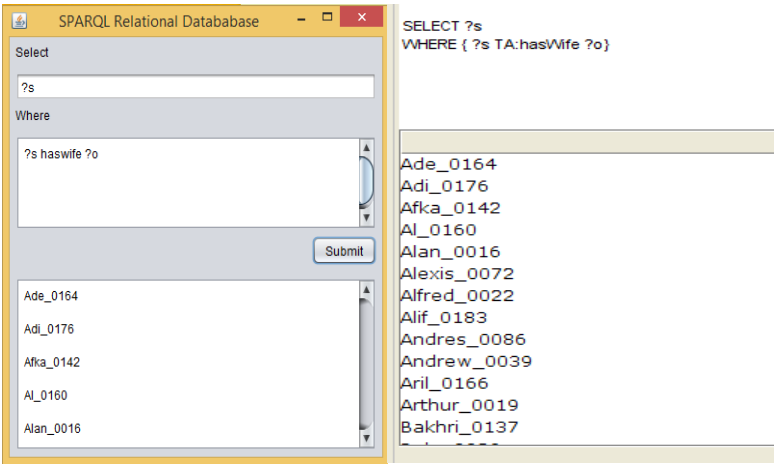
Pada tahap pengujian peng-*query*-an data Case_008, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_008 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi q*uery* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 17.

Tabel 6. 17 Pengujian Peng-*query*-an Case_008

ID	TA-Q.BD0015
Nama	Pengujian peng-*query*-an data Case_008.
Tujuan Pengujian	Menguji apakah Case_008 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_008, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_008. 2. Pengguna menekan tombol “submit”. 3. Pengguna melihat hasil *query* pada kolom hasil.

Hasil Yang Diharapkan	Data hasil *query* Case_008 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_008 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_008 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_008 dapat dilihat pada Gambar 6. 17. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_008, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_008 telah berhasil.



Gambar 6. 17 Uji Coba Peng-*query*-an Case_008

6.2.2.9. Pengujian Peng-query-an Data Case_009

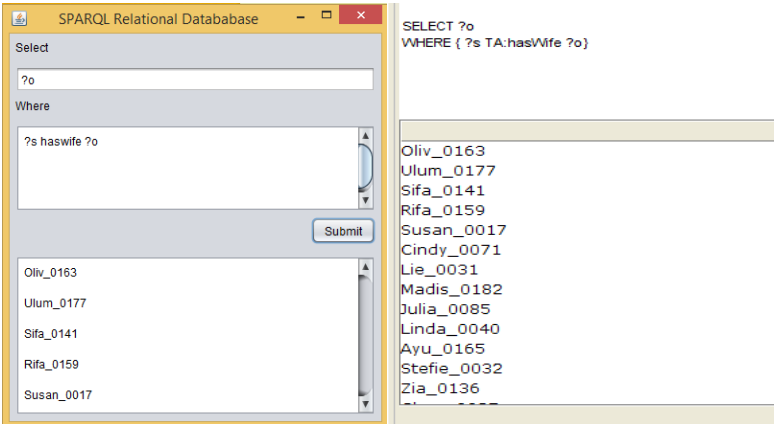
Pada tahap pengujian peng-*query*-an data Case_009, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_009 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil

pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 18.

Tabel 6. 18 Pengujian Peng-*query*-an Case_009

ID	TA-Q.BD0016
Nama	Pengujian peng-*query*-an data Case_009.
Tujuan Pengujian	Menguji apakah Case_009 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_009, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_009. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_009 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_009 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_009 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_009 dapat dilihat pada Gambar 6. 18. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_009, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_009 telah berhasil.



Gambar 6. 18 Uji Coba Peng-*query*-an Case_009

6.2.2.10. Pengujian Peng-query-an Data Case_010

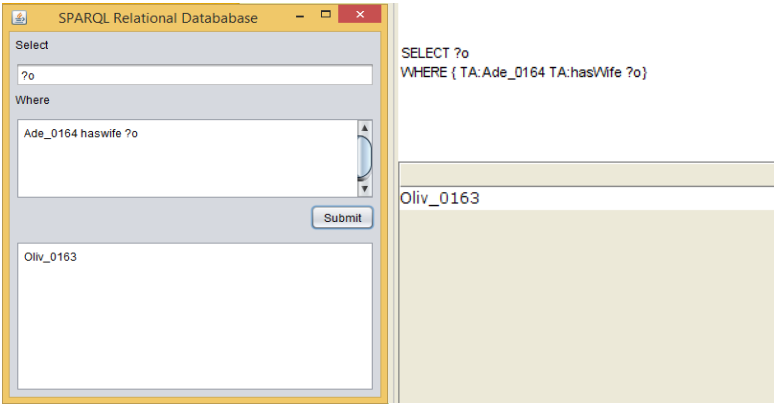
Pada tahap pengujian peng-*query*-an data Case_010, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_010 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 19.

Tabel 6. 19 Pengujian Peng-*query*-an Case_010

ID	TA-Q.BD0017
Nama	Pengujian peng-*query*-an data Case_010.
Tujuan Pengujian	Menguji apakah Case_010 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_010, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_010.

	2. Pengguna menekan tombol “submit”. 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_010 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_010 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_010 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_010 dapat dilihat pada Gambar 6. 19. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_010, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_010 telah berhasil.



Gambar 6. 19 Uji Coba Peng-*query*-an Case_010

6.2.2.11. Pengujian Peng-query-an Data Case_011

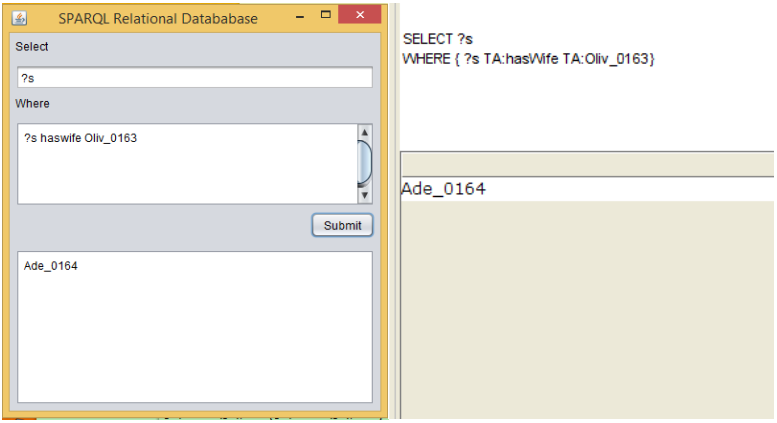
Pada tahap pengujian peng-*query*-an data Case_011, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_011 ini berfungsi untuk memetakan *syntax query*

SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 20.

Tabel 6. 20 Pengujian Peng-*query*-an Case_011

ID	TA-Q.BD0018
Nama	Pengujian peng-*query*-an data Case_011.
Tujuan Pengujian	Menguji apakah Case_011berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_011, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_011. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_011 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_011 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_011 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_011 dapat dilihat pada Gambar 6. 20. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_011, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_011 telah berhasil.



Gambar 6. 20 Uji Coba Peng-*query*-an Case_011

6.2.2.12. Pengujian Peng-query-an Data Case_012

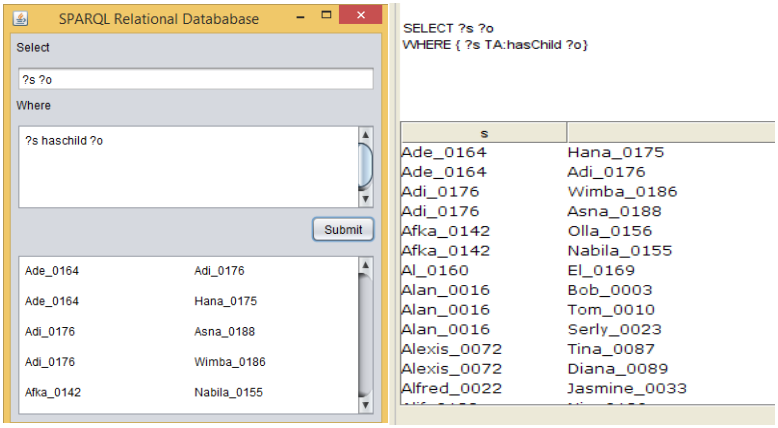
Pada tahap pengujian peng-*query*-an data Case_012, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_012 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 21.

Tabel 6. 21 Pengujian Peng-*query*-an Case_012

ID	TA-Q.BD0019
Nama	Pengujian peng-*query*-an data Case_012.
Tujuan Pengujian	Menguji apakah Case_012 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_012, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_012. 2. Pengguna menekan tombol “submit”.

	3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_012 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_012 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_012 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_012 dapat dilihat pada Gambar 6. 21. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_012, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_012 telah berhasil.



Gambar 6. 21 Uji Coba Peng-*query*-an Case_012

6.2.2.20 Pengujian Peng-query-an Data Case_013

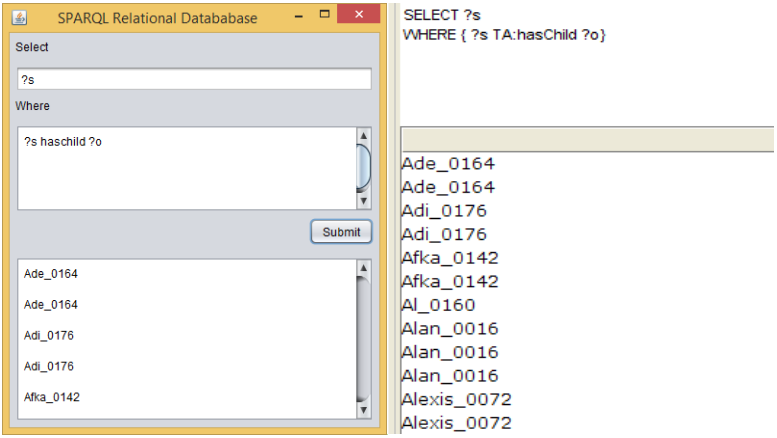
Pada tahap pengujian peng-*query*-an data Case_013, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_013 ini berfungsi untuk memetakan *syntax query*

SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 22.

Tabel 6. 22 Pengujian Peng-*query*-an Case_013

ID	TA-Q.BD0021
Nama	Pengujian peng-*query*-an data Case_013.
Tujuan Pengujian	Menguji apakah Case_013 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_013, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_013. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_013 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_013 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_013 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_013 dapat dilihat pada Gambar 6. 22. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_013, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_013 telah berhasil.



Gambar 6. 22 Uji Coba Peng-*query*-an Case_013

6.2.2.21 Pengujian Peng-query-an Data Case_014

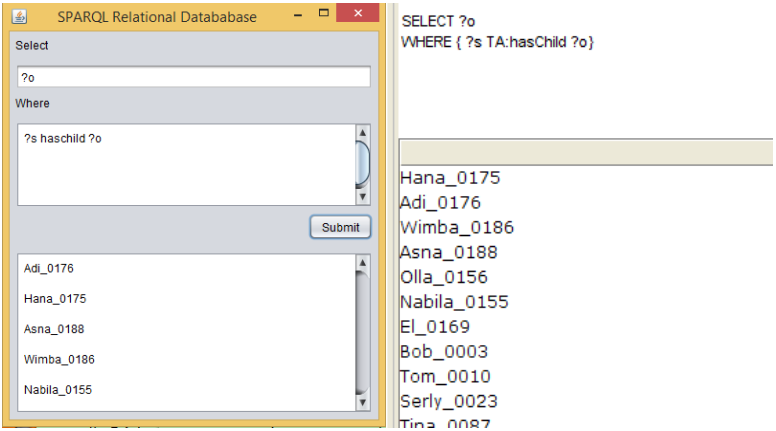
Pada tahap pengujian peng-*query*-an data Case_014, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_014 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 23.

Tabel 6. 23 Pengujian Peng-*query*-an Case_014

ID	TA-Q.BD0022
Nama	Pengujian peng-*query*-an data Case_014.
Tujuan Pengujian	Menguji apakah Case_014 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_014, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_014.

	2. Pengguna menekan tombol “submit”. 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_014 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_014 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_014 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_014 dapat dilihat pada Gambar 6. 23. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_014, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_014 telah berhasil.



Gambar 6. 23 Uji Coba Peng-*query*-an Case_014

6.2.2.22 Pengujian Peng-query-an Data Case_015

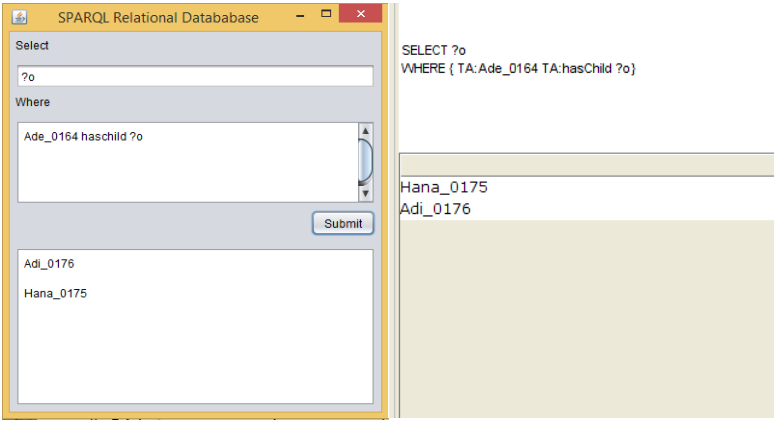
Pada tahap pengujian peng-*query*-an data Case_015, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_015 ini berfungsi untuk memetakan *syntax query*

SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 24.

Tabel 6. 24 Pengujian Peng-*query*-an Case_015

ID	TA-Q.BD0023
Nama	Pengujian peng-*query*-an data Case_015.
Tujuan Pengujian	Menguji apakah Case_015 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_015, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_015. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_015 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_015 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_015 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_015 dapat dilihat pada Gambar 6. 24. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_015, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_015 telah berhasil.



Gambar 6. 24 Uji Coba Peng-*query*-an Case_015

6.2.2.23 Pengujian Peng-query-an Data Case_016

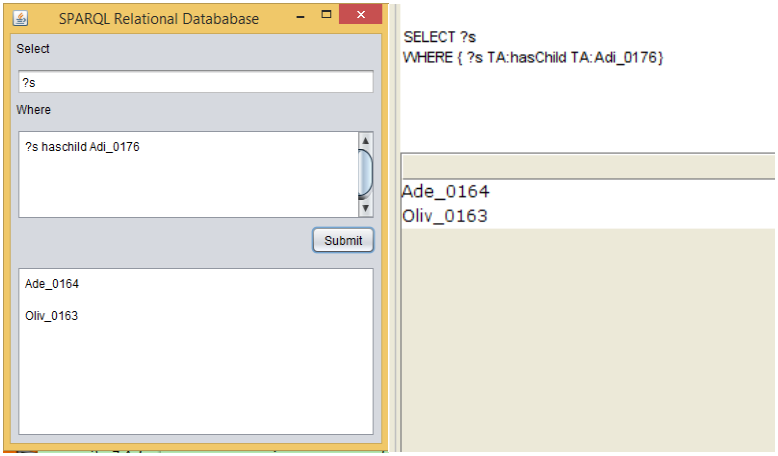
Pada tahap pengujian peng-*query*-an data Case_016, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_016 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi q*uery* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 25.

Tabel 6. 25 Pengujian Peng-*query*-an Case_016

ID	TA-Q.BD0024
Nama	Pengujian peng-*query*-an data Case_016.
Tujuan Pengujian	Menguji apakah Case_016 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_016, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_016. 2. Pengguna menekan tombol “submit”.

	3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_016 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_016 berupa data *person*.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_016 berupa data *person*.

Untuk hasil pengujian peng-*query*-an data Case_016 dapat dilihat pada Gambar 6. 25. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_016, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_016 telah berhasil.



Gambar 6. 25 Uji Coba Peng-*query*-an Case_016

6.2.2.24 Pengujian Peng-query-an Data Case_017

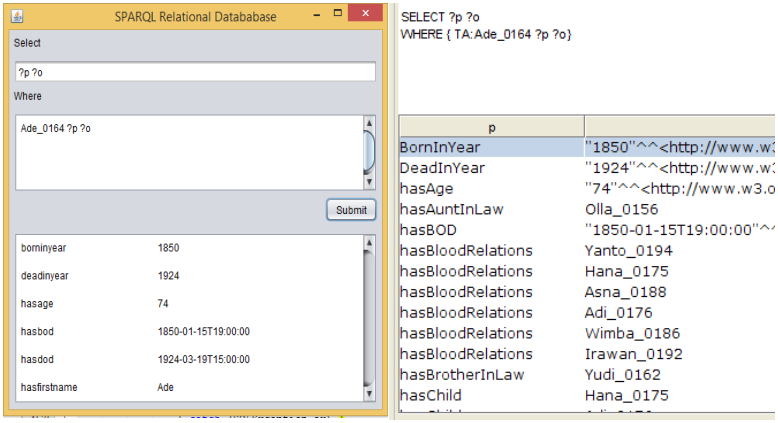
Pada tahap pengujian peng-*query*-an data Case_017, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada

Subbab 4.2. Case_017 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan syntax tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 26

Tabel 6. 26 Pengujian Peng-*query*-an Case_017

ID	TA-Q.BD0025
Nama	Pengujian peng-*query*-an data Case_017.
Tujuan Pengujian	Menguji apakah Case_017 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_017, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_017. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_017 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_017 berupa data *person* dan *property*-nya.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_017 berupa data *person* dan *property*-nya.

Untuk hasil pengujian peng-*query*-an data Case_017 dapat dilihat pada Gambar 6. 26. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_017, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_017 telah berhasil.



Gambar 6. 26 Uji Coba Peng-*query*-an Case_017

6.2.2.25 Pengujian Peng-query-an Data Case_018

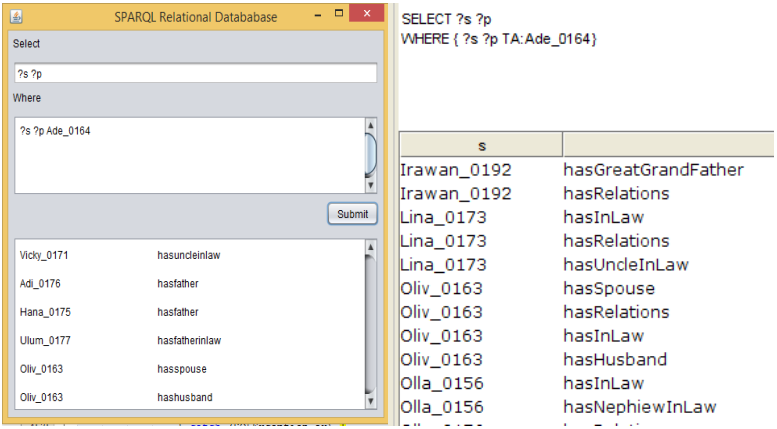
Pada tahap pengujian peng-*query*-an data Case_018, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_018 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 27.

Tabel 6. 27 Pengujian Peng-*query*-an Case_018

ID	TA-Q.BD0029
Nama	Pengujian peng-*query*-an data Case_018.
Tujuan Pengujian	Menguji apakah Case_018 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_018, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_018. 2. Pengguna menekan tombol “submit”.

	3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_018 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_018 berupa data *person* dan *property*-nya.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_018 berupa data *person* dan *property*-nya.

Untuk hasil pengujian peng-*query*-an data Case_018 dapat dilihat pada Gambar 6. 27. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_018, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_018 telah berhasil.



Gambar 6. 27 Uji Coba Peng-*query*-an Case_018

6.2.2.26 Pengujian Peng-query-an Data Case_019

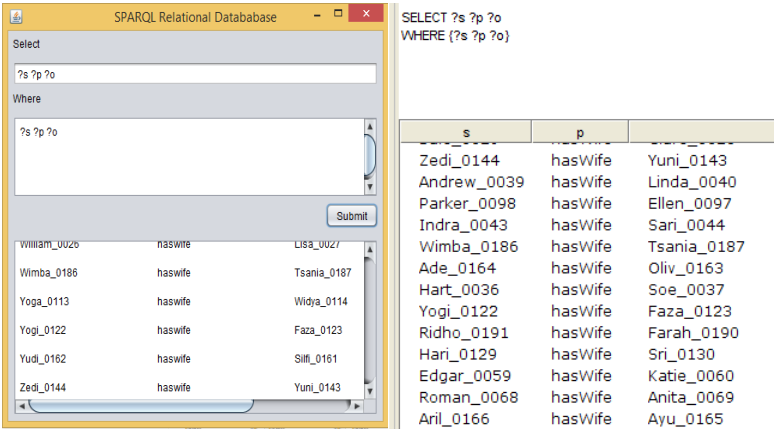
Pada tahap pengujian peng-*query*-an data Case_019, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_019 ini berfungsi untuk memetakan *syntax query*

SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 28.

Tabel 6. 28 Pengujian Peng-*query*-an Case_019

ID	TA-Q.BD0033
Nama	Pengujian peng-*query*-an data Case_019.
Tujuan Pengujian	Menguji apakah Case_019 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_019, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_019. 2. Pengguna menekan tombol "submit". 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_019 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_019 berupa data *person* dan *property*-nya.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_019 berupa data *person* dan *property*-nya.

Untuk hasil pengujian peng-*query*-an data Case_019 dapat dilihat pada Gambar 6. 28. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_019, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_019 telah berhasil.



Gambar 6. 28 Uji Coba Peng-*query*-an Case_019

6.2.2.47 Pengujian Peng-query-an Data Case_020

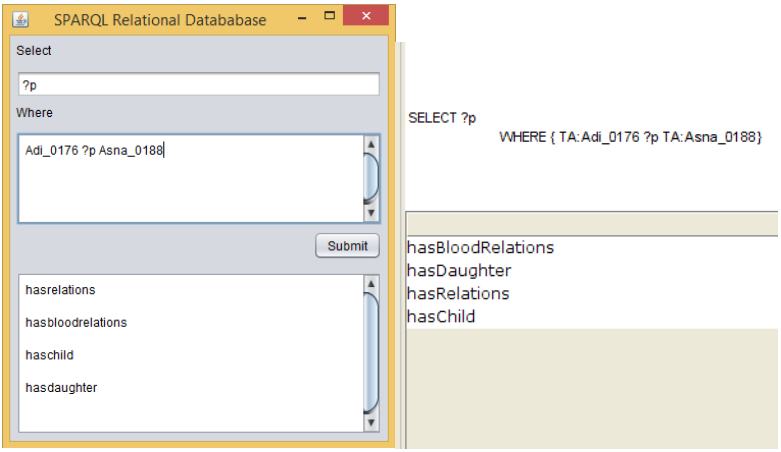
Pada tahap pengujian peng-*query*-an data Case_020, berjalan ketika kondisinya terpenuhi seperti yang dijelaskan pada Subbab 4.2. Case_020 ini berfungsi untuk memetakan *syntax query* SPARQL menjadi *query* SQL dan menampilkan data hasil pengolahan *syntax* tersebut. Rincian pengujian perbandingan data ini dapat dilihat pada Tabel 6. 29.

Tabel 6. 29 Pengujian Peng-*query*-an Case_020

ID	TA-Q.BD0047
Nama	Pengujian peng-*query*-an data Case_020.
Tujuan Pengujian	Menguji apakah Case_020 berjalan atau tidak.
Skenario 1	Melakukan *query* SPARQL untuk Case_020, dan melihat hasil pengolahan *syntax query* tersebut.
Kondisi Awal	Kolom penulisan *query* SPARQL dan hasil masih kosong.
Data Uji	Data uji merupakan basis data pohon keluarga yang sudah diolah dan disimpan pada PostgreSQL.
Langkah Pengujian	1. Pengguna menuliskan *syntax query* SPARQL untuk Case_020.

	2. Pengguna menekan tombol “submit”. 3. Pengguna melihat hasil *query* pada kolom hasil.
Hasil Yang Diharapkan	Data hasil *query* Case_020 berhasil ditampilkan.
Hasil Yang Didapat	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_020 berupa data *person* dan *property*-nya.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Didapatkan hasil pengolahan *syntax* SPARQL sesuai dengan Case_020 berupa data *person* dan *property*-nya.

Untuk hasil pengujian peng-*query*-an data Case_020 dapat dilihat pada Gambar 6. 29. Pada gambar tersebut dapat dilihat bahwa ketika dilakukan *query* SPARQL yang memenuhi kondisi Case_020, maka aplikasi akan memunculkan data hasil pengolahan *syntax* tersebut. Hasil tersebut membuktikan bahwa uji coba peng-*query*-an Case_020 telah berhasil.



Gambar 6. 29 Uji Coba Peng-*query*-an Case_020

6.2.3. Pengujian Waktu Pemrosesan Query SPARQL

Pengujian waktu pemrosesan *query* SPARQL merupakan tahap uji yang bertujuan untuk mengetahui berapa lama waktu yang dibutuhkan oleh sistem untuk melakukan *query* SPARQL.

6.2.3.1. Pengujian Waktu Pemrosesan Query SPARQL Case_017

Pada tahap pengujian ini dilakukan pengukuran waktu yang dibutuhkan dalam pemrosesan *query* SPARQL untuk Case_017 pada aplikasi yang dibangun. Rincian pengujian *query* SPARQL ini dapat dilihat pada Tabel 6. 30

Tabel 6. 30 Pengujian Waktu Pemrosesan *Query* SPARQL Case_017

ID	TA-UJ.KQ0001
Nama	Pengujian waktu pemrosesan *query* SPARQL Case_017
Tujuan Pengujian	Menguji berapa lama waktu yang dibutuhkan sitem untuk melakukan *query* SPARQL.
Skenario 1	Menghitung waktu yang dibutuhkan sistem untuk melakukan *query* SPARQL.
Kondisi Awal	*Syntax query* SPARQL telah dituliskan pada aplikasi yang dibangun.
Data Uji	Data uji berupa *file* ontology yang telah disimpan dalam DBMS PostgreSQL
Langkah Pengujian	1. Pengguna menekan tombol "submit" 2. Sistem menampilkan data hasil *query* dan waktu pemrosesannya.
Hasil Yang Diharapkan	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 0,5 s.
Hasil Yang Didapat	Didapatkan hasil pengukuran bahwa proses *query* SPARQL membutuhkan waktu 53 ms.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 0,5 ms.

6.2.3.2. Pengujian Waktu Pemrosesan Query SPARQL Case_019

Pada tahap pengujian ini dilakukan pengukuran waktu yang dibutuhkan dalam pemrosesan *query* SPARQL untuk Case_019 pada aplikasi yang dibangun. Rincian pengujian *query* SPARQL ini dapat dilihat pada Tabel 6. 31.

Tabel 6. 31 Pengujian Waktu Pemrosesan *Query* SPARQL Case_019

ID	TA-UJ.KQ0002
Nama	Pengujian waktu pemrosesan *query* SPARQL Case_019
Tujuan Pengujian	Menguji berapa lama waktu yang dibutuhkan sistem untuk melakukan *query* SPARQL.
Skenario 1	Menghitung waktu yang dibutuhkan sistem untuk melakukan *query* SPARQL.
Kondisi Awal	*Syntax query* SPARQL telah dituliskan pada aplikasi yang dibangun.
Data Uji	Data uji berupa *file* ontology yang telah disimpan dalam DBMS PostgreSQL
Langkah Pengujian	1. Pengguna menekan tombol "submit" 2. Sistem menampilkan data hasil *query* dan waktu pemrosesannya.
Hasil Yang Diharapkan	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 1 s.
Hasil Yang Didapat	Didapatkan hasil pengukuran bahwa proses *query* SPARQL membutuhkan waktu 896 ms.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 1 s.

6.2.3.3. Pengujian Waktu Pemrosesan Query SPARQL Case_020

Pada tahap pengujian ini dilakukan pengukuran waktu yang dibutuhkan dalam pemrosesan *query* SPARQL untuk

Case_020 pada aplikasi yang dibangun. Rincian pengujian *query* SPARQL ini dapat dilihat pada Tabel 6. 32.

Tabel 6. 32 Pengujian Waktu Pemrosesan *Query* SPARQL Case_020

ID	TA-UJ.KQ0003
Nama	Pengujian waktu pemrosesan *query* SPARQL Case_020
Tujuan Pengujian	Menguji berapa lama waktu yang dibutuhkan sistem untuk melakukan *query* SPARQL.
Skenario 1	Menghitung waktu yang dibutuhkan sistem untuk melakukan *query* SPARQL.
Kondisi Awal	*Syntax query* SPARQL telah dituliskan pada aplikasi yang dibangun.
Data Uji	Data uji berupa *file* ontology yang telah disimpan dalam DBMS PostgreSQL
Langkah Pengujian	1. Pengguna menekan tombol "submit" 2. Sistem menampilkan data hasil *query* dan waktu pemrosesannya.
Hasil Yang Diharapkan	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 0,5 s.
Hasil Yang Didapat	Didapatkan hasil pengukuran bahwa proses *query* SPARQL membutuhkan waktu 44 ms.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Waktu yang dibutuhkan sistem untuk pemrosesan *query* SPARQL kurang dari 0,5 s.

6.3. Evaluasi Pengujian

Pada subbab ini akan diberikan hasil evaluasi dari pengujian-pengujian yang telah dilakukan. Evaluasi yang diberikan meliputi evaluasi pengujian penyimpanan data yang telah dijelaskan pada Subbab 6.2.1, evaluasi pengujian peng-*query*-an data yang telah dijelaskan pada Subbab 6.2.2, dan evaluasi pengujian waktu pemrosesan *query* SPARQL yang telah dijelaskan pada Subbab 6.2.3.

6.3.1. Evaluasi Pengujian Penyimpanan Basis Data Relasional Hasil Transformasi OWL2RDB

Rangkuman mengenai hasil pengujian penyimpanan data dapat dilihat pada Tabel 6. 33. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga bisa ditarik kesimpulan bahwa penyimpanan basis data telah sesuai dengan yang diharapkan.

Tabel 6. 33 Rangkuman Hasil Pengujian Penyimpanan Basis Data Relasional Hasil Transformasi OWL2RDB

ID	Nama	Skenario	Hasil
TA-UJ.BDC0001	Pengujian Kevalidan Penyimpanan *Object property hasRelations*	Skenario 1	Berhasil
TA-UJ.BDC0002	Pengujian Kevalidan Penyimpanan *Object property hasBloodRelations*	Skenario 1	Berhasil
TA-UJ.BDC0003	Pengujian Kevalidan Penyimpanan *Object property hasAunt*	Skenario 1	Berhasil
TA-UJ.BDO0001	Pengujian Kevalidan Penyimpanan *Object property hasChild*	Skenario 1	Berhasil
TA-UJ.BDO0002	Pengujian Kevalidan Penyimpanan *Object property hasDaughter*	Skenario 1	Berhasil
TA-UJ.BDO0003	Pengujian Kevalidan Penyimpanan *Object property hasSon*	Skenario 1	Berhasil
TA-UJ.BDD0001	Pengujian Kevalidan Penyimpanan *Object property hasCousin*	Skenario 1	Berhasil
TA-UJ.BDD0002	Pengujian Kevalidan Penyimpanan *Object property hasGrandChild*	Skenario 1	Berhasil
TA-UJ.BDD0003	Pengujian Kevalidan Penyimpanan *Object property hasGrandDaughter*	Skenario 1	Berhasil

6.3.2. Evaluasi Pengujian Peng-*query*-an Data

Rangkuman mengenai hasil pengujian peng-*query*-an data dapat dilihat pada Tabel 6. 34. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga bisa ditarik kesimpulan bahwa aplikasi yang dibangun telah sesuai dengan yang diharapkan.

Tabel 6. 34 Rangkuman Hasil Pengujian Peng-*query*-an Data

ID	Nama	Skenario	Hasil
TA-Q.BD0001	Pengujian Peng-*query*-an untuk Case_001	Skenario 1	Berhasil
TA-Q.BD0002	Pengujian Peng-*query*-an untuk Case_002	Skenario 1	Berhasil
TA-Q.BD0003	Pengujian Peng-*query*-an untuk Case_003	Skenario 1	Berhasil
TA-Q.BD0004	Pengujian Peng-*query*-an untuk Case_004	Skenario 1	Berhasil
TA-Q.BD0005	Pengujian Peng-*query*-an untuk Case_005	Skenario 1	Berhasil
TA-Q.BD0006	Pengujian Peng-*query*-an untuk Case_006	Skenario 1	Berhasil
TA-Q.BD0007	Pengujian Peng-*query*-an untuk Case_007	Skenario 1	Berhasil
TA-Q.BD0008	Pengujian Peng-*query*-an untuk Case_008	Skenario 1	Berhasil
TA-Q.BD0009	Pengujian Peng-*query*-an untuk Case_009	Skenario 1	Berhasil
TA-Q.BD0010	Pengujian Peng-*query*-an untuk Case_010	Skenario 1	Berhasil
TA-Q.BD0011	Pengujian Peng-*query*-an untuk Case_011	Skenario 1	Berhasil
TA-Q.BD0012	Pengujian Peng-*query*-an untuk Case_012	Skenario 1	Berhasil
TA-Q.BD0013	Pengujian Peng-*query*-an untuk Case_013	Skenario 1	Berhasil
TA-Q.BD0014	Pengujian Peng-*query*-an untuk Case_014	Skenario 1	Berhasil
TA-Q.BD0015	Pengujian Peng-*query*-an untuk Case_015	Skenario 1	Berhasil
TA-Q.BD0016	Pengujian Peng-*query*-an untuk Case_016	Skenario 1	Berhasil
TA-Q.BD0017	Pengujian Peng-*query*-an untuk Case_017	Skenario 1	Berhasil
TA-Q.BD0018	Pengujian Peng-*query*-an untuk Case_018	Skenario 1	Berhasil
TA-Q.BD0019	Pengujian Peng-*query*-an untuk Case_019	Skenario 1	Berhasil

ID	Nama	Skenario	Hasil
TA-Q.BD0020	Pengujian Peng-*query*-an untuk Case_020	Skenario 1	Berhasil

6.3.3. Evaluasi Pengujian Waktu Pemrosesan SPARQL

Rangkuman mengenai hasil pengujian waktu pemrosesan *query* SPARQL dapat dilihat pada Tabel 6. 35. Berdasarkan data pada tabel tersebut, semua skenario pengujian berhasil. Sehingga dapat ditarik kesimpulan bahwa SPARQL yang dikembangkan sesuai dengan yang diharapkan.

Tabel 6. 35 Rangkuman Hasil Pengujian Waktu Pemrosesan SPARQL

ID	Nama	Skenario	Hasil
TA-UJ.KQ0001	Pengujian waktu pemrosesan *query* SPARQL Case_017	Skenario 1	Berhasil
TA-UJ.KQ0002	Pengujian waktu pemrosesan *query* SPARQL Case_019	Skenario 1	Berhasil
TA-UJ.KQ0003	Pengujian waktu pemrosesan *query* SPARQL Case_020	Skenario 1	Berhasil

Perbandingan waktu pemrosesan *query* SPARQL aplikasi yang dibangun dan *query engine* Protégé dapat dilihat pada tabel Tabel 6. 36.

Tabel 6. 36 Perbandingan Waktu Pemrosesan *Query* SPARQL

ID	Nama	SPARQL Relational Database	Query Engine Protégé
TA-UJ.KQ0001	Pengujian waktu pemrosesan *query* SPARQL Case_017	53 ms	± 800 s
TA-UJ.KQ0002	Pengujian waktu pemrosesan *query* SPARQL Case_019	896 ms	± 10 s

ID	Nama	SPARQL Relational Database	Query Engine Protégé
TA-UJ.KQ0003	Pengujian waktu pemrosesan *query* SPARQL Case_020	44 ms	± 750 ms

Dari tabel tersebut dapat disimpulkan bahwa waktu pemrosesan *query* SPARQL dari sistem yang dibangun lebih kecil daripada waktu pemrosesan *query* SPARQL pada Protégé. Padahal jumlah memory yang dipakai pada saat proses peng-*query*-an oleh Protégé lebih besar daripada yang dipakai pada aplikasi yang dibangun (NetBeans dan Java). Dimana Protégé membutuhkan *memory* 800-900 MB saat proses peng-*query*-an, sedangkan NetBeans dan java hanya membutuhkan membutuhkan *memory* kurang dari 600 MB pada saat proses peng-*query*-an. Jadi aplikasi yang dikembangkan ini bisa dikatakan berhasil.

(Halaman ini sengaja dikosongkan)

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa untuk dikembangkan dari Tugas Akhir ini.

7.1. Kesimpulan

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

1. Skema yang diterapkan untuk melakukan penyimpanan dan peng-*query*-an basis data adalah dengan melakukan normalisasi terlebih dahulu terhadap data ontologi tersebut, kemudian melakukan *reasoning* terhadap data tersebut. Data hasil *reasoning* ini yang akan ditransformasikan menggunakan OWL2RDB ke dalam DMBS PostgreSQL. Setelah data berhasil disimpan ke dalam postgreSQL baru kemudian membangun aplikasi yang bisa memetakan *syntax query* SPARQL menjadi *syntax query* SQL, dan menampilkan hasil pengolahan *syntax query* tersebut.
2. Penyimpanan ontologi pohon keluarga ke dalam basis data relasional dapat dilakukan dengan menggunakan *plugin* Protégé berupa OWL2RDB, dengan *plugin* tersebut bisa didapatkan *syntax* SQL hasil transformasi ontologi pohon keluarga. Dengan begitu *syntax* tersebut tinggal dijalankan pada SQL Editor aplikasi PostgreSQL, dan akan menghasilkan beberapa tabel beserta isinya.

3. Peng-*query*-an (*query* SPARQL) basis data pohon keluarga dapat dilakukan dengan cara menuliskan *syntax query* SPARQL pada kolom yang disediakan oleh aplikasi yang dibangun. Sistem akan memetakan *syntax query* SPARQL tersebut ke dalam *syntax query* SQL berdasarkan pola-pola yang sudah dibangun pada aplikasi ini. Dan nantinya sistem akan menampilkan hasil pengolahan *syntax query* tersebut pada kolom hasil aplikasi yang dibangun.
4. Sesuai dengan hasil uji coba yang telah dilakukan, waktu pemrosesan yang dibutuhkan untuk peng-*query*-an data pada aplikasi yang dibangun lebih kecil daripada waktu pemrosesan yang dibutuhkan untuk peng-*query*-an data pada Protégé, perbedaan itu terlihat apabila jumlah data yang digunakan semakin banyak. Dan aplikasi yang dibangun ini juga membutuhkan memori yang lebih kecil apabila dibandingkan dengan Protégé.

7.2. Saran

Membutuhkan penelitian lebih lanjut supaya jumlah klausa *where* dalam penulisan *syntax query* SPARQL bisa lebih banyak lagi.

DAFTAR PUSTAKA

[1] "Genealogi," Kemdikbud (Pusat Bahasa), 2012-2016. [Online]. Available: http://kbbi.web.id/genealogi 15 01 2017. [Accessed 15 01 2017].

[2] "Genealogi," [Online]. Available: https://id.wikipedia.org/wiki/Genealogi. [Accessed 15 01 2017].

[3] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *International Journal of Human-Computer Studies,* vol. 43, no. 5-6, p. 907–928, 1995.

[4] M. Horridge, A Practical Guide To Building OWL Ontologies Using Prot´eg´e 4 and CO-ODE Tools Edition 1.2, The University Of Manchester, 2009.

[5] N. F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880* , 2001.

[6] C. H. d. Nascimento, F. S. Ferraz, R. E. Assad, D. L. e. Silva and V. H. d. Rocha, "OntoLog: Using Web Semantic and Ontology for Security Log Analysis," in *ICSEA 2011 : The Sixth International Conference on Software Engineering Advances*, Brazil, 2011.

[7] S. J. Miller, Introduction to Ontology Concepts and Terminology, Lisbon, Portugal: University of Wisconsin-Milwaukee, 2013.

[8] M. A. Ramadhanie, Penerapan Ontologi Objek Pembelajaran Untuk Kebutuhan Personalisasi E-Learning Berbasis Semantic Web, Depok: Universitas Indonesia, 2009.

[9] M. Cristani and R. Cuel, "A Survey on Ontology Creation Methodologies," *Int'l Journal on Semantic Web & Information Systems,* vol. 1, no. 2, pp. 48-68, April-June 2005.

[10] D. L. McGuinness and F. v. Harmelen, "OWL Web Ontology Language Overview," [Online]. Available: https://www.w3.org/TR/owl-features/. [Accessed 06 January 2016].

[11] I. N. Bagus Caturbawa, "Case Tool untuk Pemodelan Data Semantik dalam Web Ontology Language (OWL)," Institut Teknologi Sepuluh Nopember, Surabaya, 2009.

[12] J. Heflin, "OWL Web Ontology Language Use Cases and Requirements," [Online]. Available: https://www.w3.org/TR/webont-req/. [Accessed 06 January 2016].

[13] S. Nikles, "Expressiveness of Enterprise Modelling Languages," University of Applied Sciences Northwestern Switzerland, Basel, 2010.

[14] C. Candrabiantara, D. O. Siahaan and U. L. Yuhana, "Rancang Bangun Aplikasi Visualisasi Silsilah Keluarga Berbasis Ontologi," *Jurnal Teknik POMITS,* vol. 2, no. 1, 2013.

[15] "Professor Robert Stevens," [Online]. Available: http://www.cs.man.ac.uk/~stevensr/ontology/family.rdf.owl . [Accessed 06 January 2016].

[16] G. Meditskos and N. Bassiliades, "A Rule-Based Object-Oriented OWL Reasoner," *IEEE Transactions on Knowledge and Data Engineering,* vol. 20, no. 3, pp. 397-410, 2008.

[17] B. Parsia and E. Sirin, "Pellet: An OWL DL Reasoner," University of Maryland, College Park.

[18] "SQL," [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed 20 Nopember 2015].

[19] "SPARQL," [Online]. Available: https://en.wikipedia.org/wiki/SPARQL. [Accessed 20 Nopember 2015].

[20] E. Vysniauskas, N. Lina, A. Sukys and B. Paradauskus, "Preserving Semantics of OWL 2 Ontologies in Relational Database Using Hybrid Approach," *Information Technology and Control,* vol. 41, no. 2, pp. 103-115, 2012.

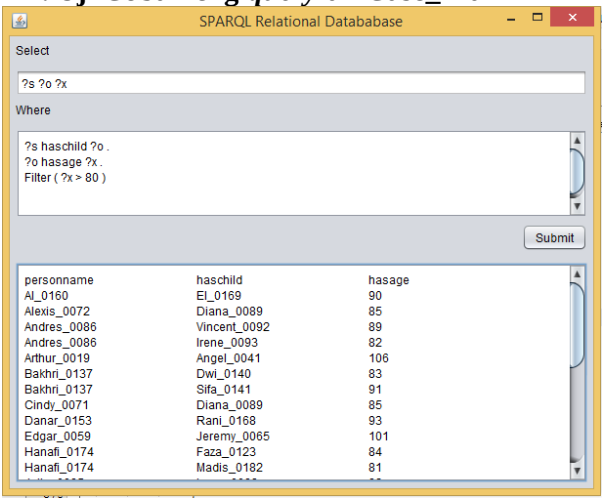
[21] E. Vysniauskas, N. Lina, A. Sukys and B. Paradauskus, "Enhancing Connection Between Ontologies and Database with OWL 2 Concept and SPARQL," *Proccedings og the 16th International Conference on Information and Software Technologies,* pp. 350-357, 2010.

[22] E. Vysniauskas, N. Lina, A. Sukys and B. Paradauskus, "A Hybrid Approach for Relating OWL 2 Ontologies and Relational Database," *Perspective in Businnes Informatics Research Proceddings of the 90th International Conference (BIR'2010),* pp. 86-101, 2010.

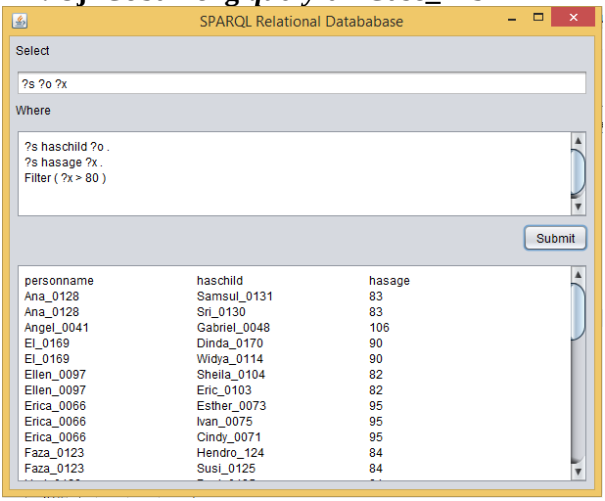
(Halaman ini sengaja dikosongkan)

LAMPIRAN

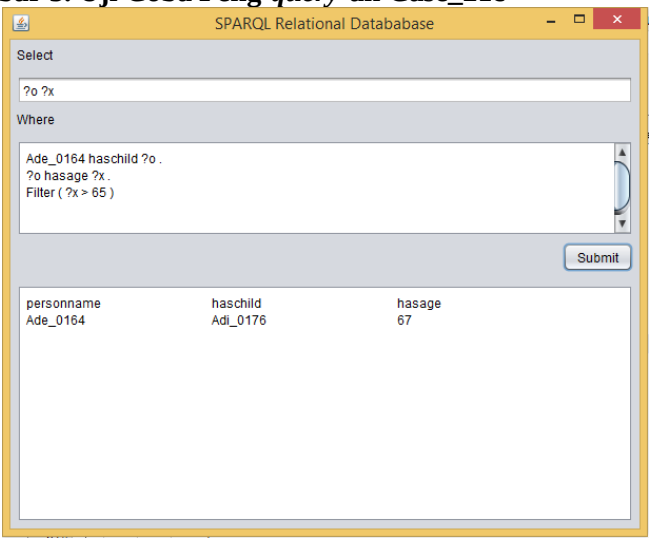
Gambar 1. Uji Coba Peng-*query*-an Case_21a



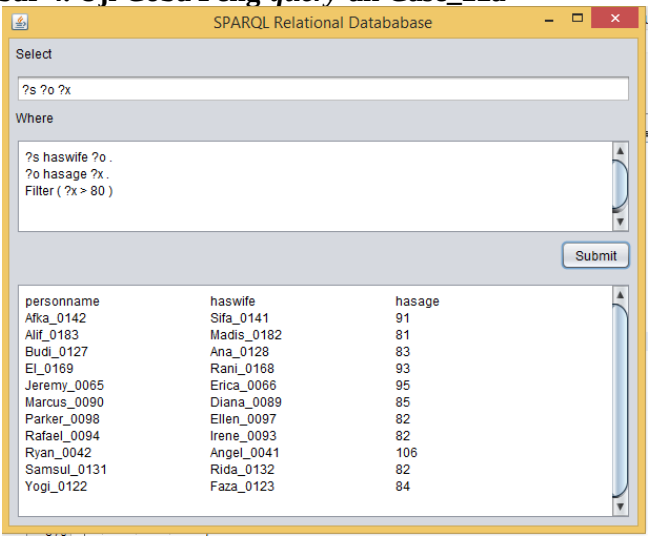
Gambar 2. Uji Coba Peng-*query*-an Case_21b



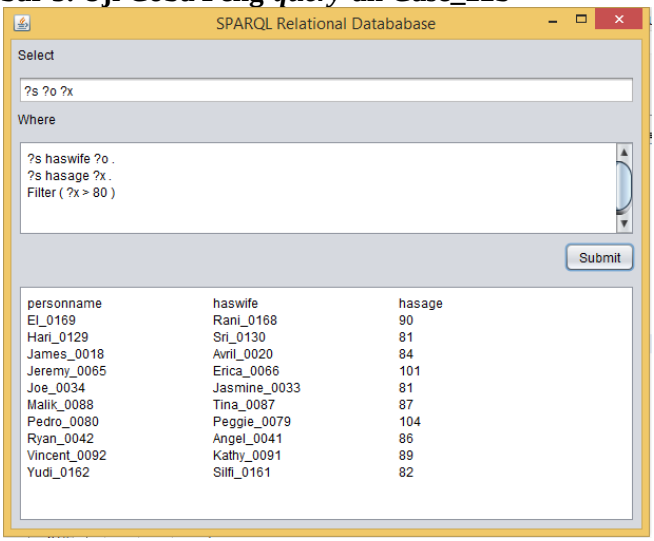
Gambar 3. Uji Coba Peng-*query*-an Case_21c



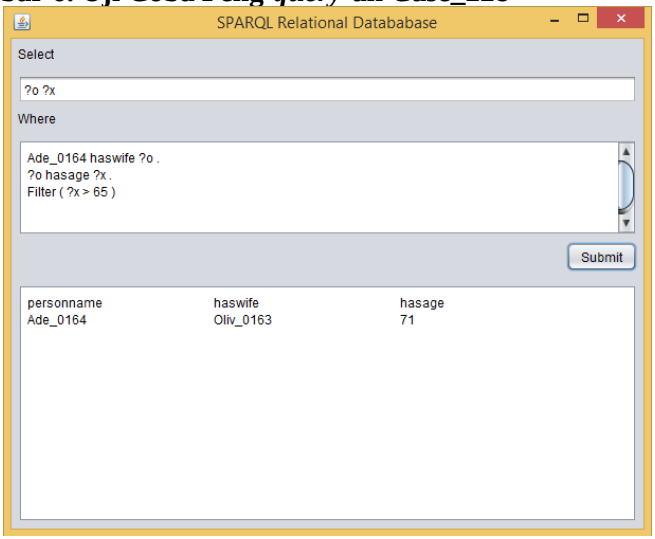
Gambar 4. Uji Coba Peng-*query*-an Case_22a



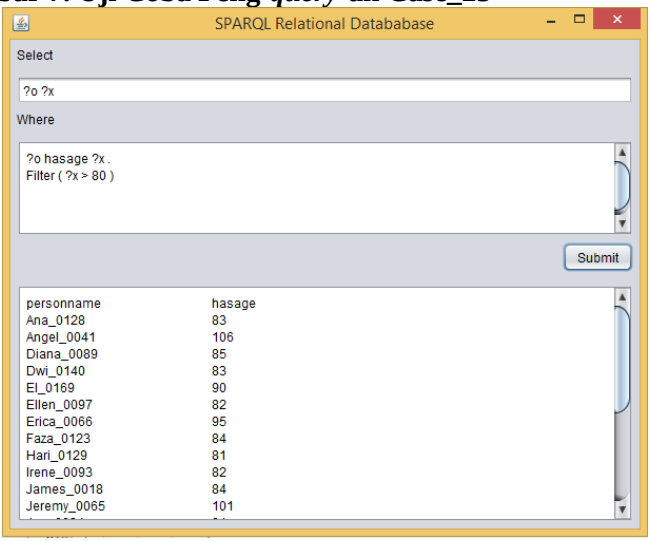
Gambar 5. Uji Coba Peng-*query*-an Case_22b



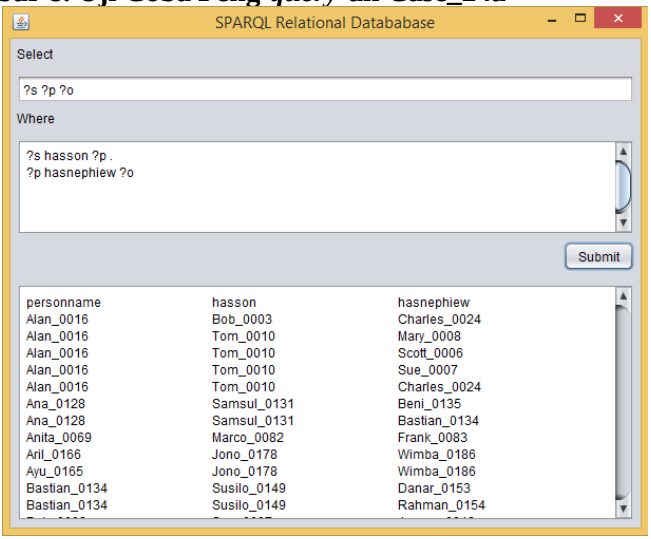
Gambar 6. Uji Coba Peng-*query*-an Case_22c



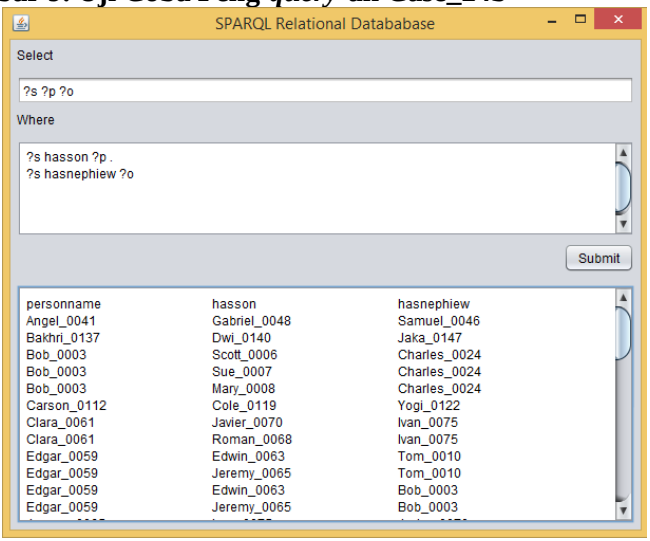
Gambar 7. Uji Coba Peng-*query*-an Case_23



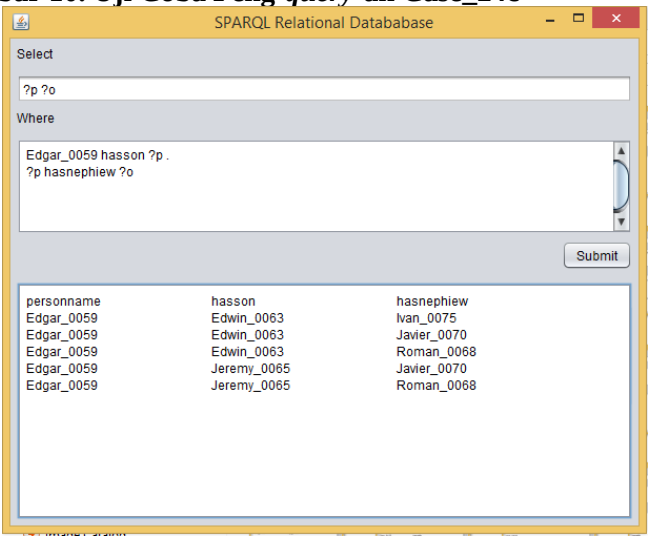
Gambar 8. Uji Coba Peng-*query*-an Case_24a



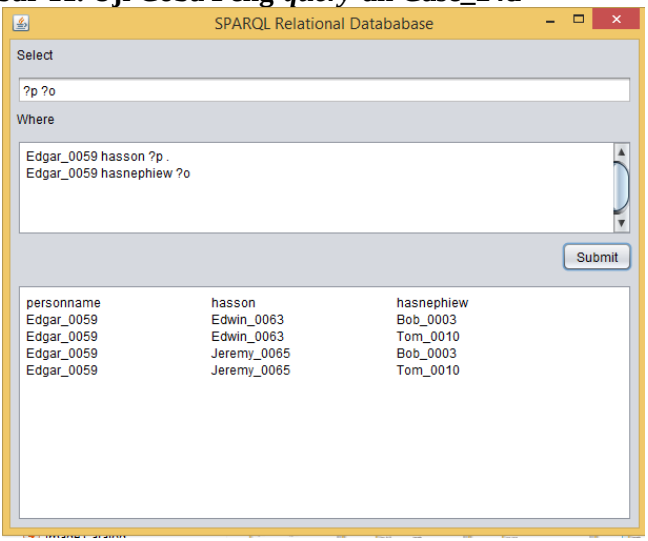
Gambar 9. Uji Coba Peng-*query*-an Case_24b



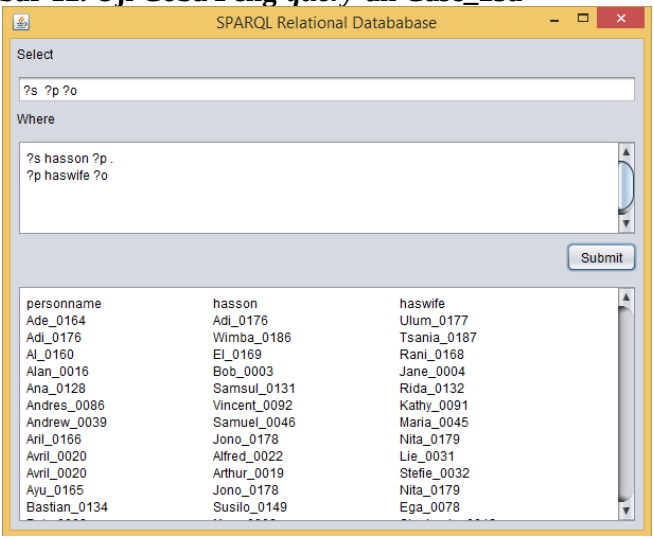
Gambar 10. Uji Coba Peng-*query*-an Case_24c



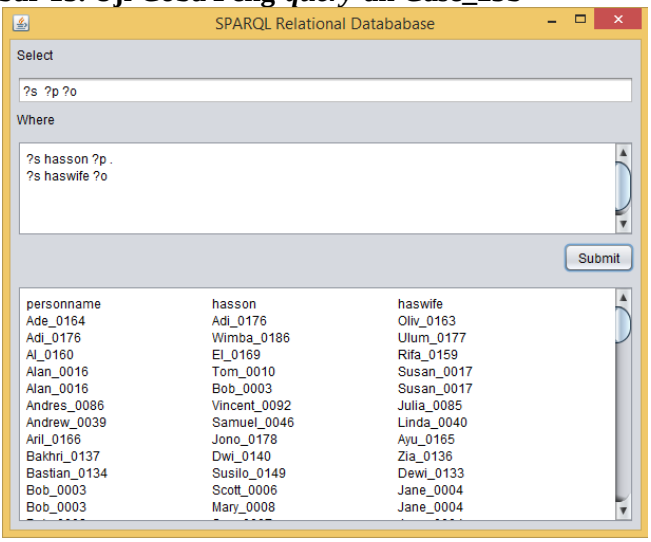
Gambar 11. Uji Coba Peng-*query*-an Case_24d



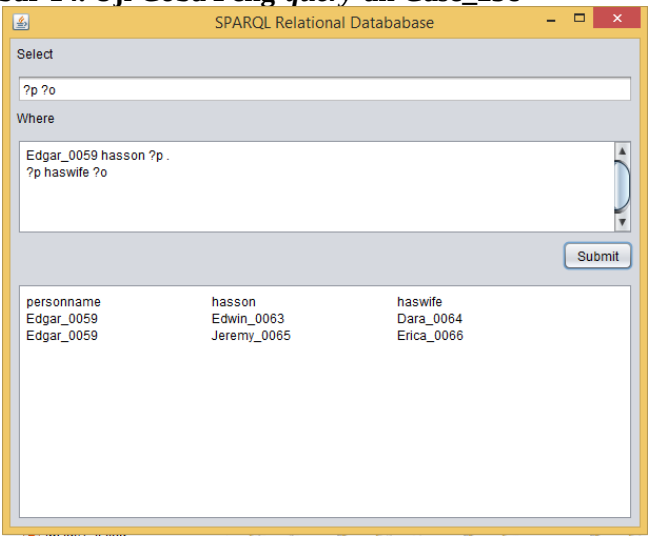
Gambar 12. Uji Coba Peng-*query*-an Case_25a



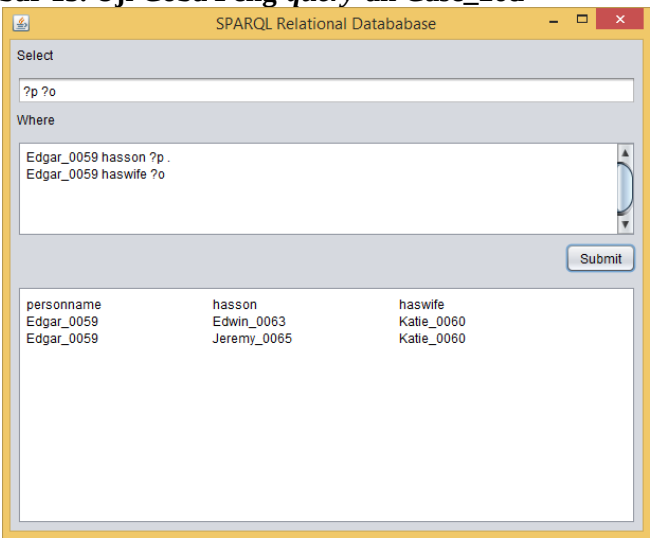
Gambar 13. Uji Coba Peng-*query*-an Case_25b



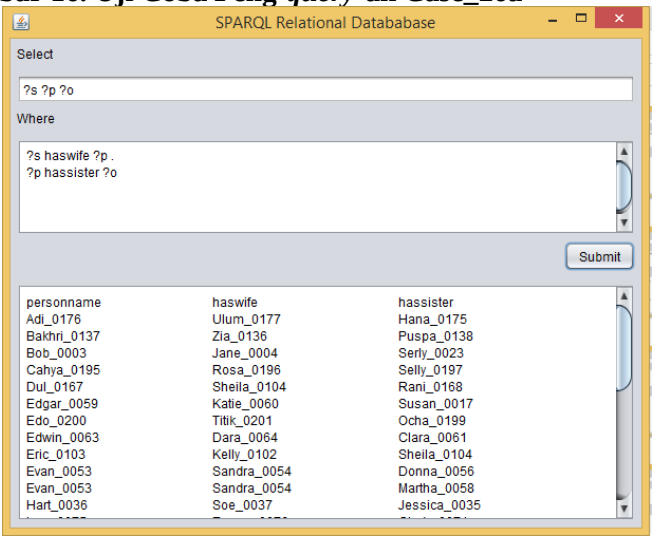
Gambar 14. Uji Coba Peng-*query*-an Case_25c



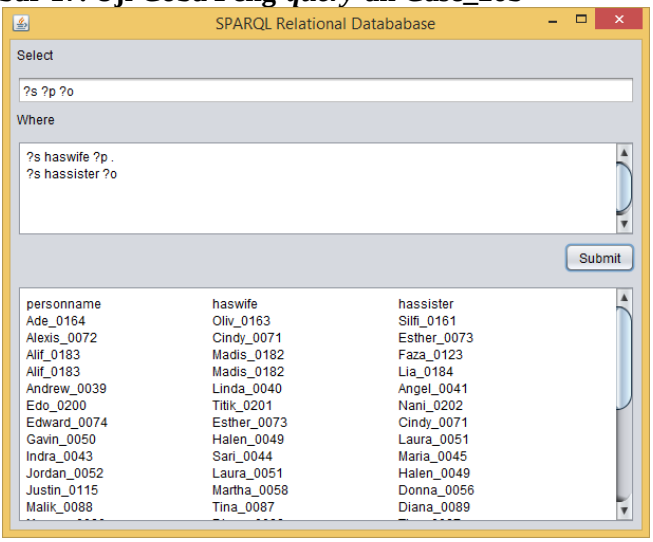
Gambar 15. Uji Coba Peng-*query*-an Case_26d



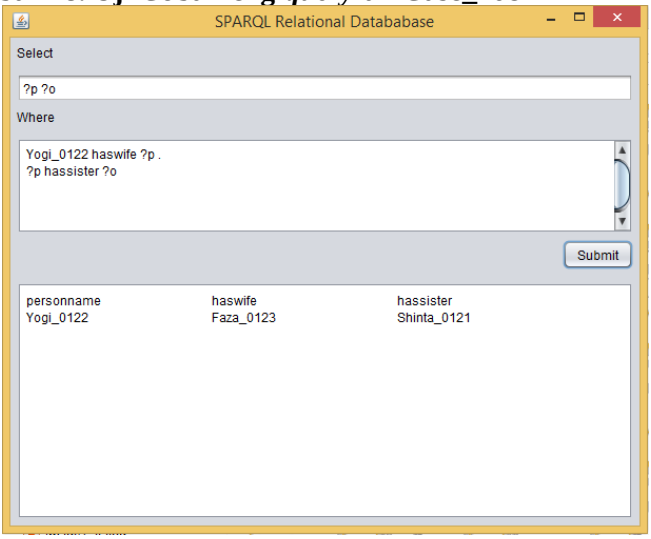
Gambar 16. Uji Coba Peng-*query*-an Case_26a



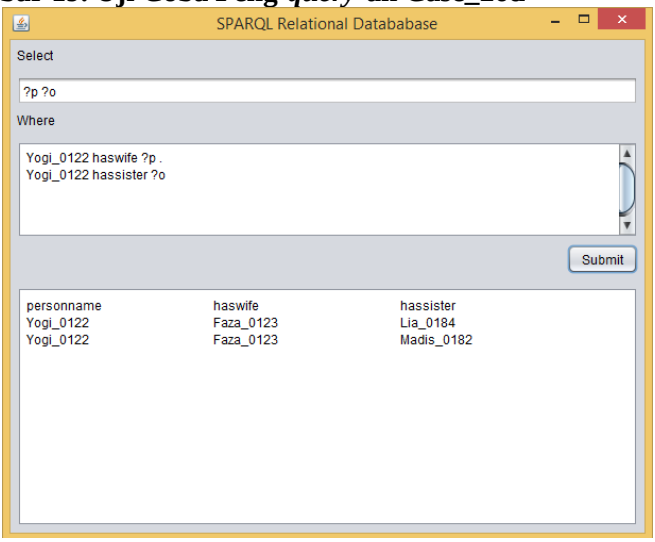
Gambar 17. Uji Coba Peng-*query*-an Case_26b



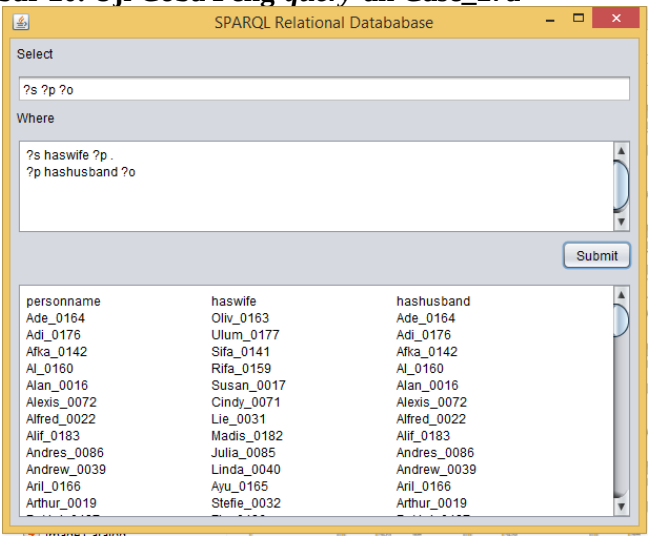
Gambar 18. Uji Coba Peng-*query*-an Case_26c



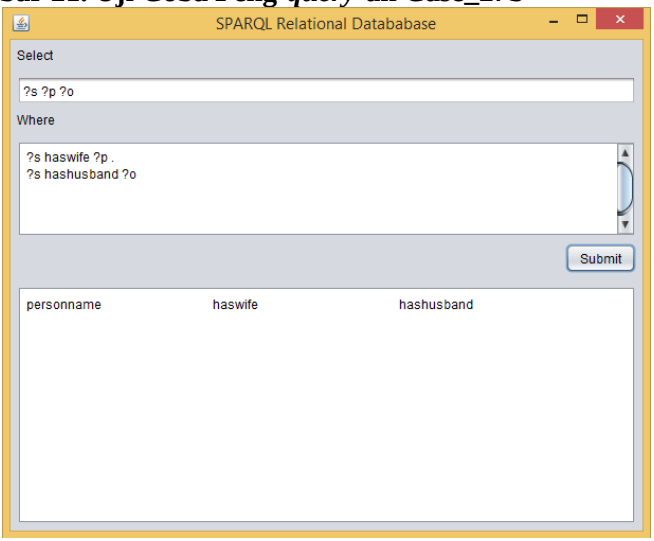
Gambar 19. Uji Coba Peng-*query*-an Case_26d



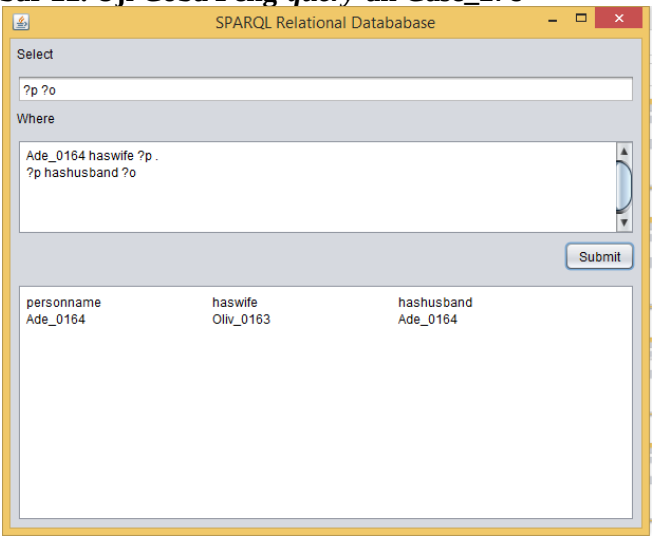
Gambar 20. Uji Coba Peng-*query*-an Case_27a



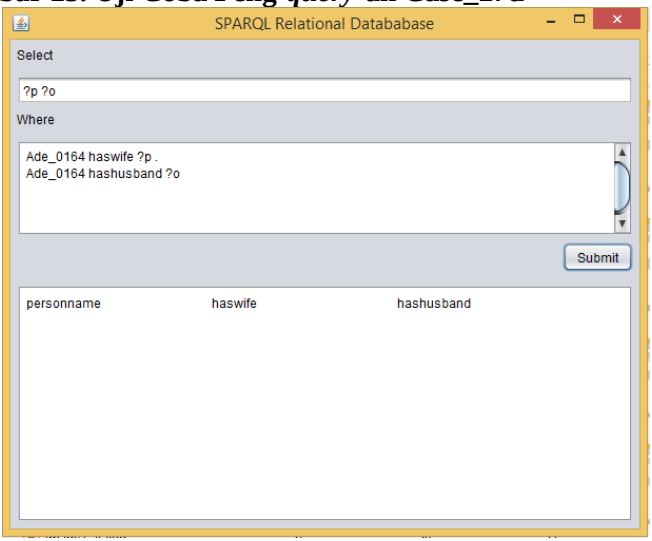
Gambar 21. Uji Coba Peng-*query*-an Case_27b



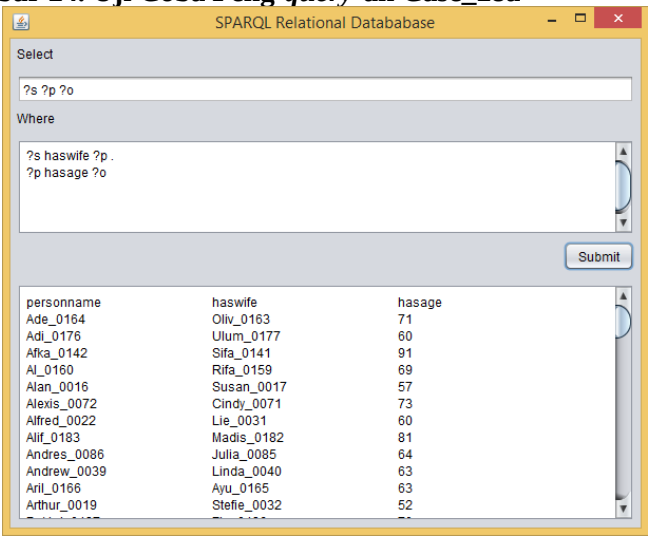
Gambar 22. Uji Coba Peng-*query*-an Case_27c



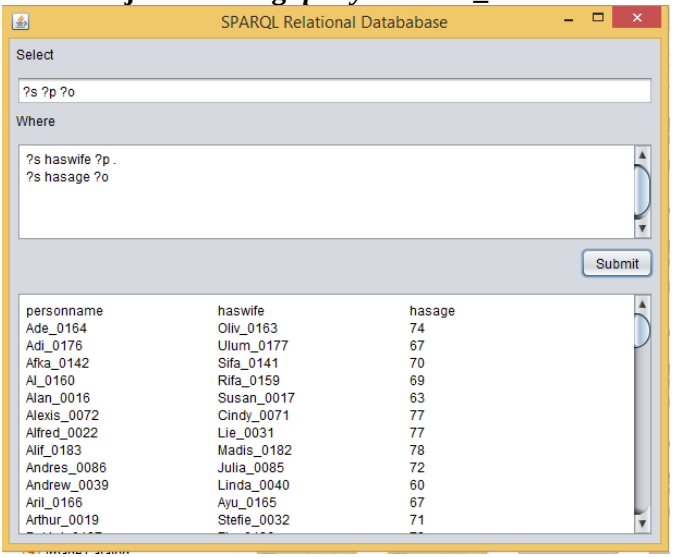
Gambar 23. Uji Coba Peng-*query*-an Case_27d



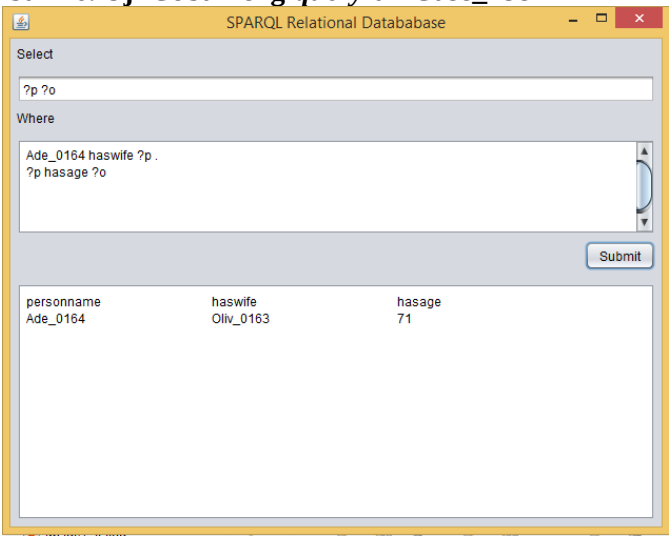
Gambar 24. Uji Coba Peng-*query*-an Case_28a



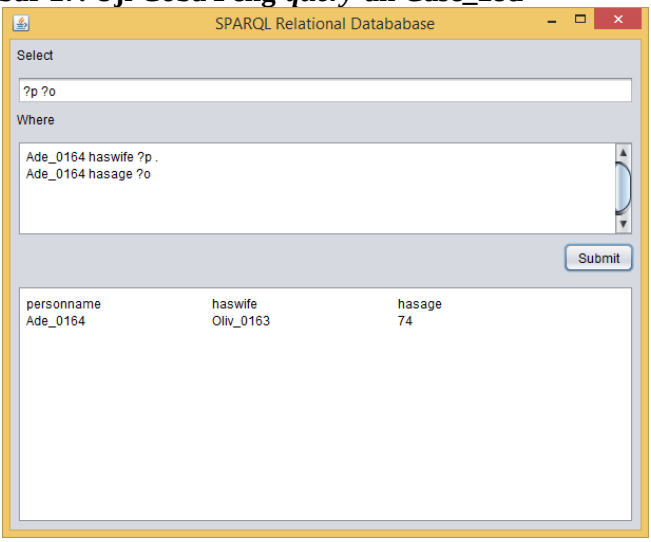
Gambar 25. Uji Coba Peng-*query*-an Case_28b



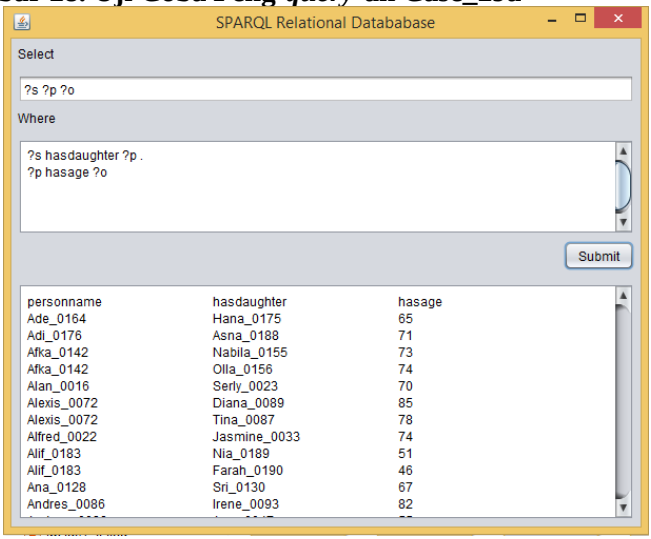
Gambar 26. Uji Coba Peng-*query*-an Case_28c



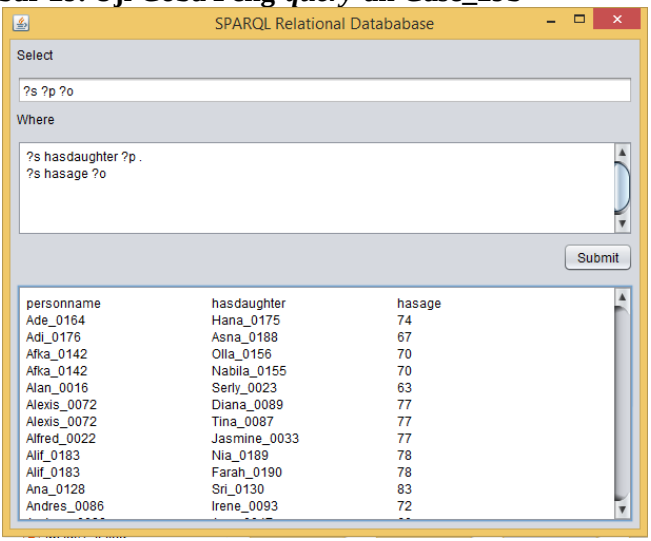
Gambar 27. Uji Coba Peng-*query*-an Case_28d



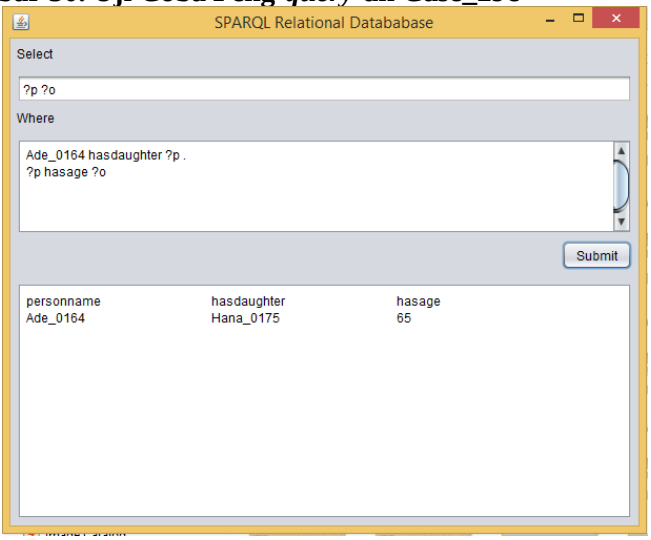
Gambar 28. Uji Coba Peng-*query*-an Case_29a



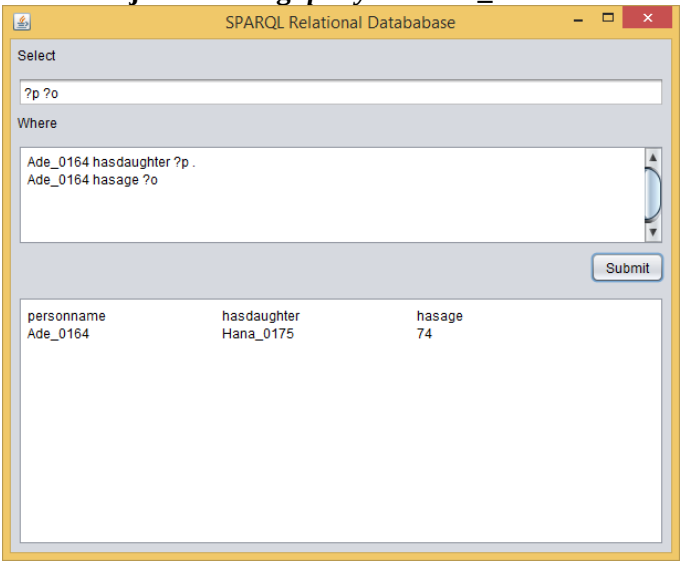
Gambar 29. Uji Coba Peng-*query*-an Case_29b



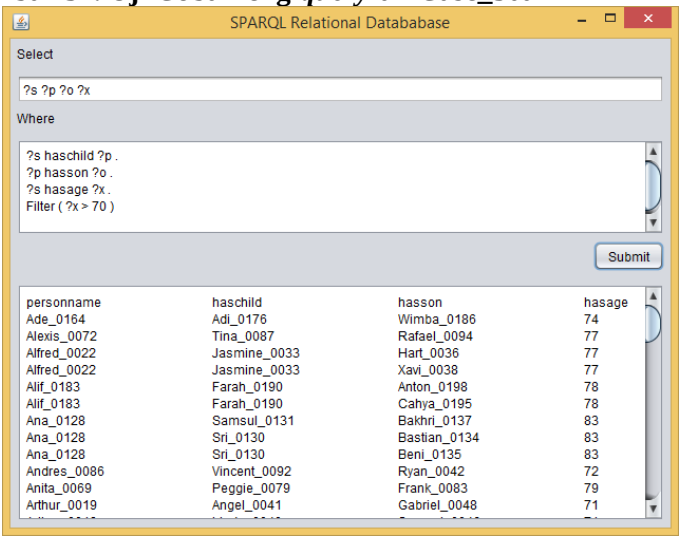
Gambar 30. Uji Coba Peng-*query*-an Case_29c



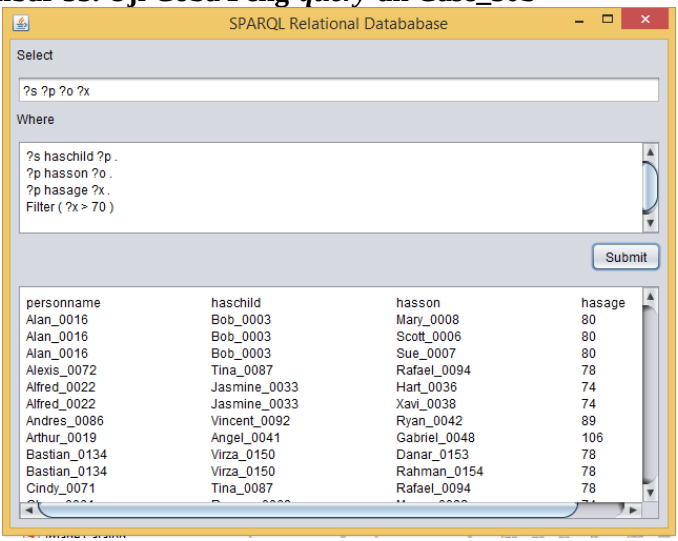
Gambar 31. Uji Coba Peng-*query*-an Case_29d



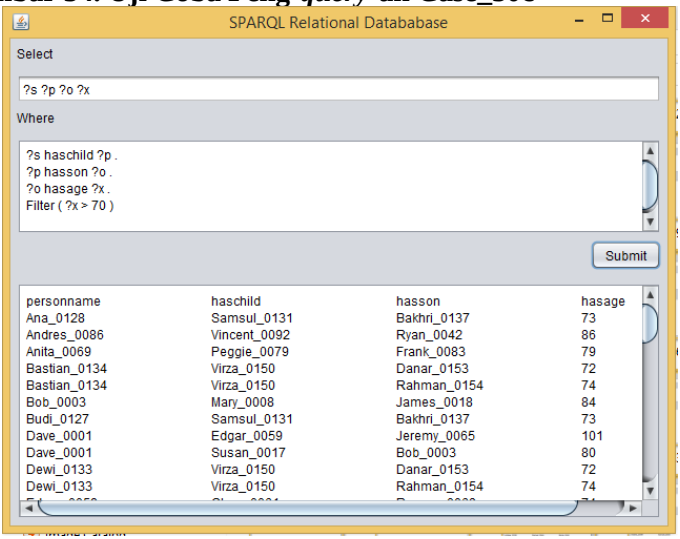
Gambar 32. Uji Coba Peng-*query*-an Case_30a



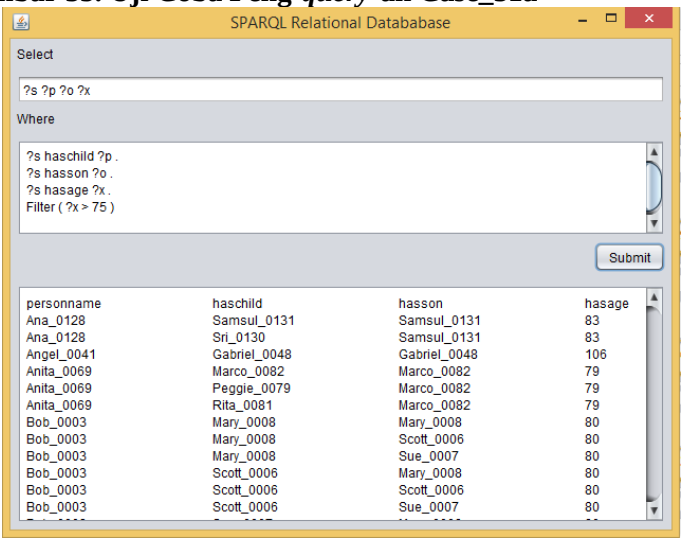
Gambar 33. Uji Coba Peng-*query*-an Case_30b



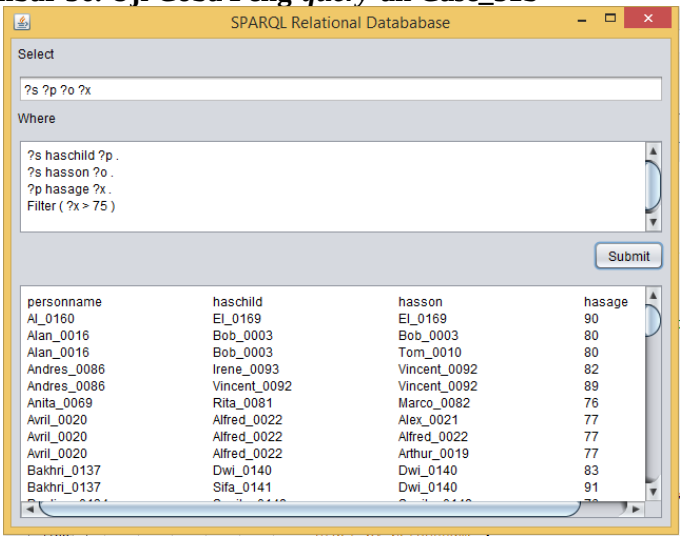
Gambar 34. Uji Coba Peng-*query*-an Case_30c



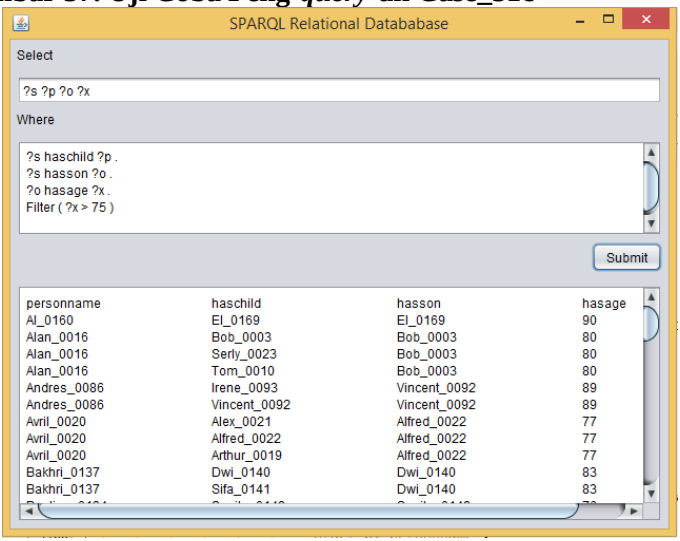
Gambar 35. Uji Coba Peng-*query*-an Case_31a



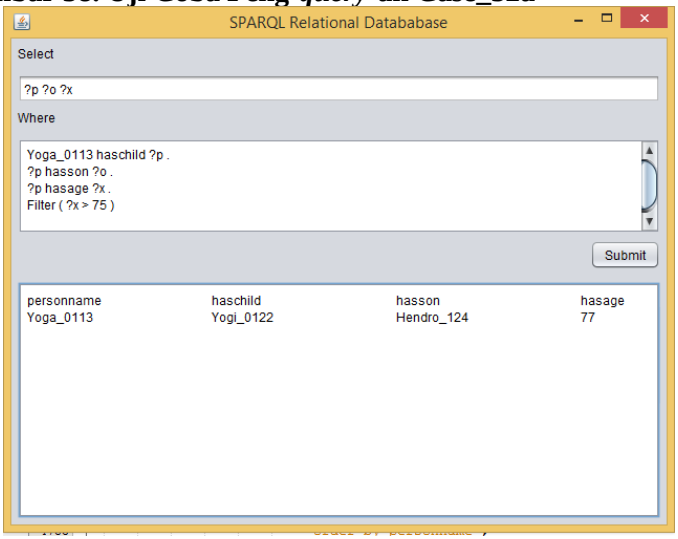
Gambar 36. Uji Coba Peng-*query*-an Case_31b



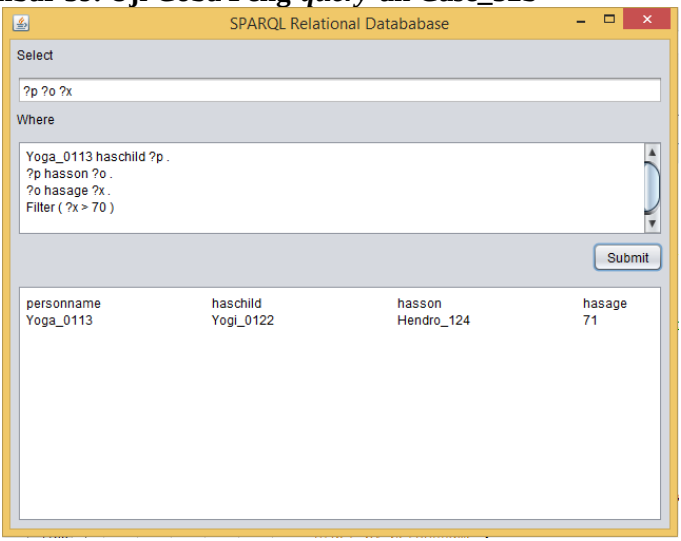
Gambar 37. Uji Coba Peng-*query*-an Case_31c



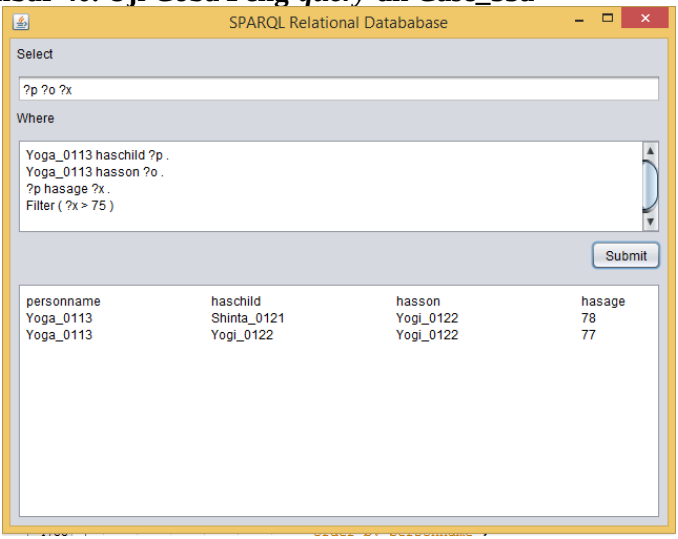
Gambar 38. Uji Coba Peng-*query*-an Case_32a



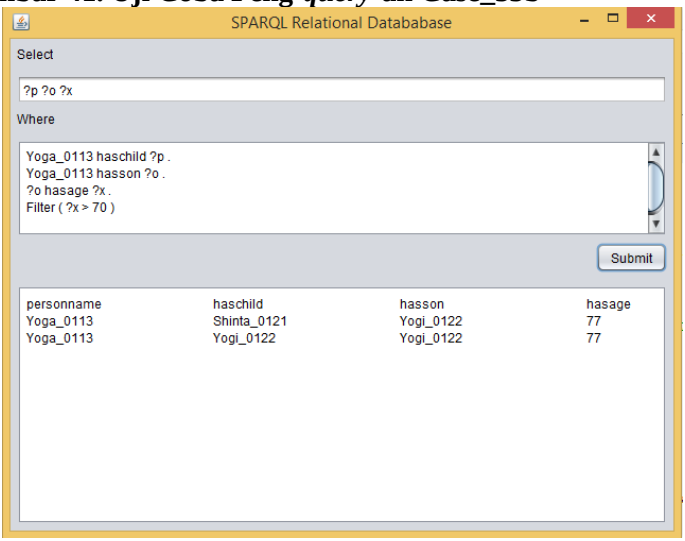
Gambar 39. Uji Coba Peng-*query*-an Case_32b



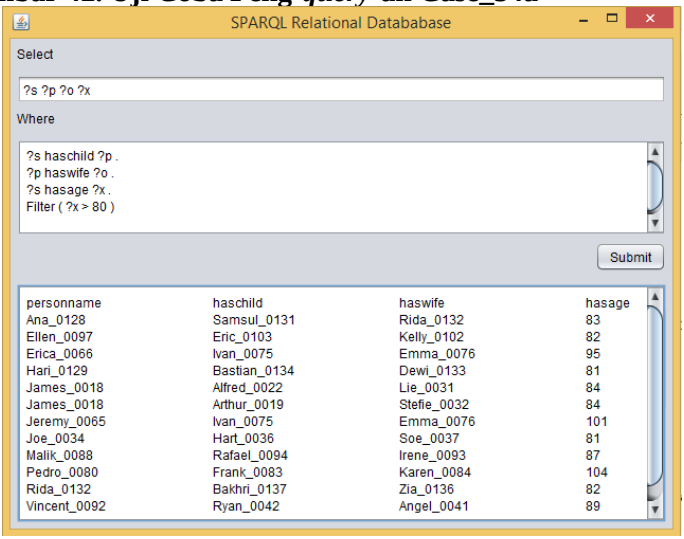
Gambar 40. Uji Coba Peng-*query*-an Case_33a



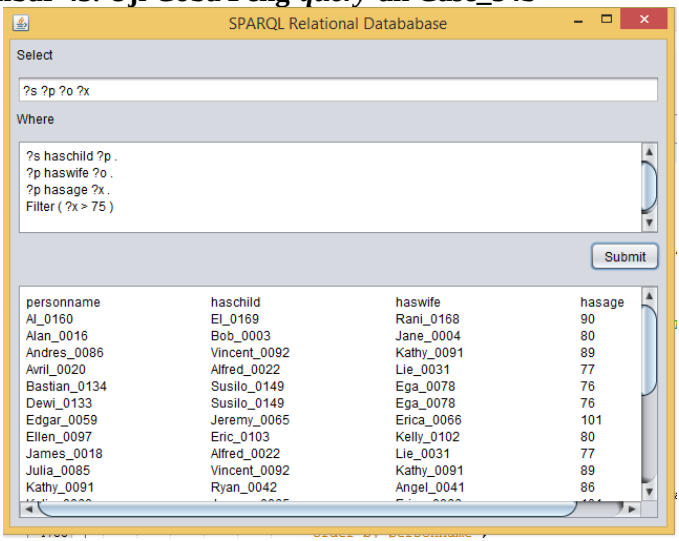
Gambar 41. Uji Coba Peng-*query*-an Case_33b



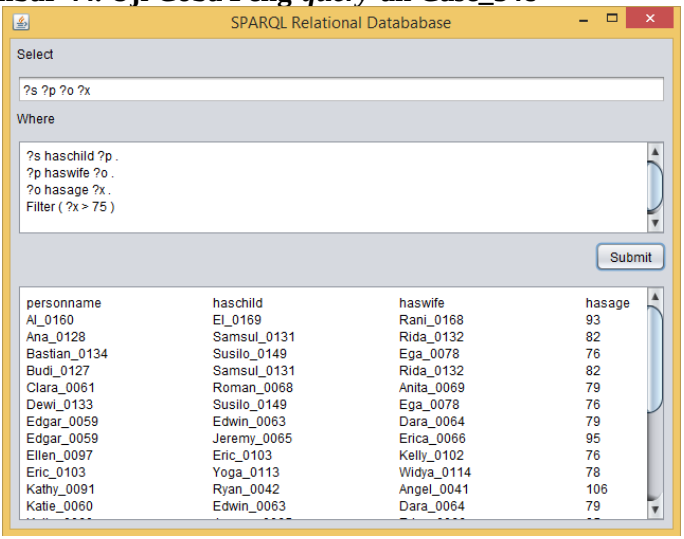
Gambar 42. Uji Coba Peng-*query*-an Case_34a



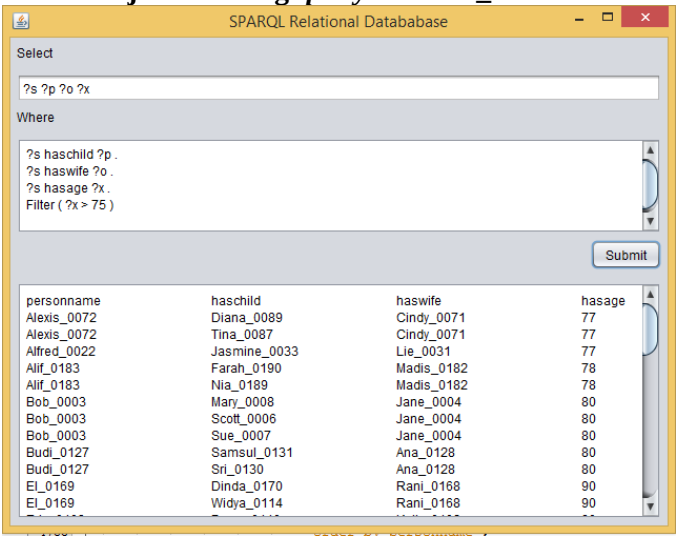
Gambar 43. Uji Coba Peng-*query*-an Case_34b



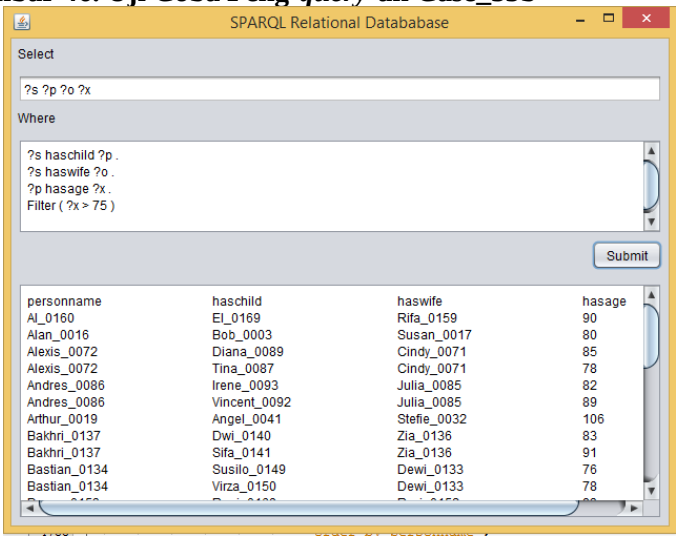
Gambar 44. Uji Coba Peng-*query*-an Case_34c



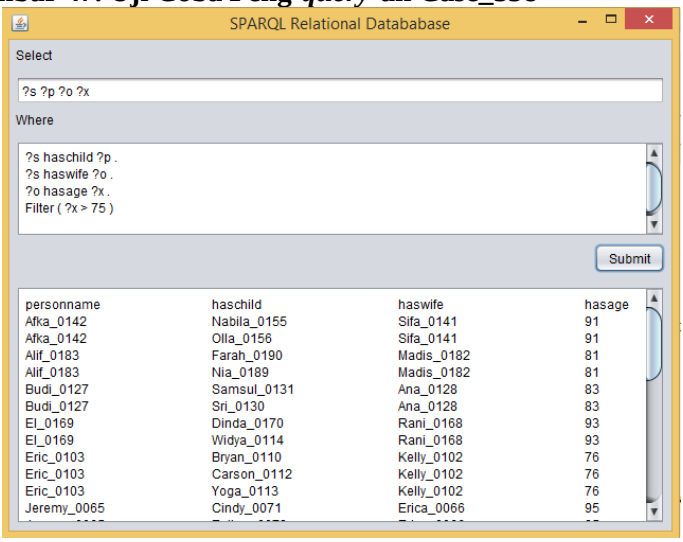
Gambar 45. Uji Coba Peng-*query*-an Case_35a



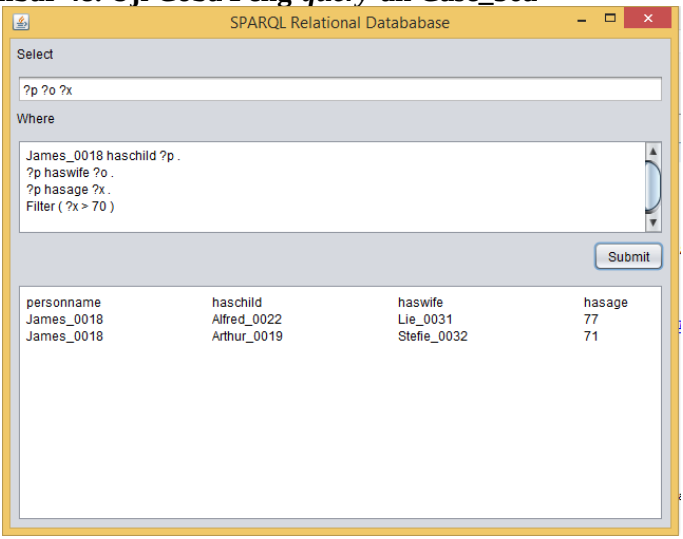
Gambar 46. Uji Coba Peng-*query*-an Case_35b



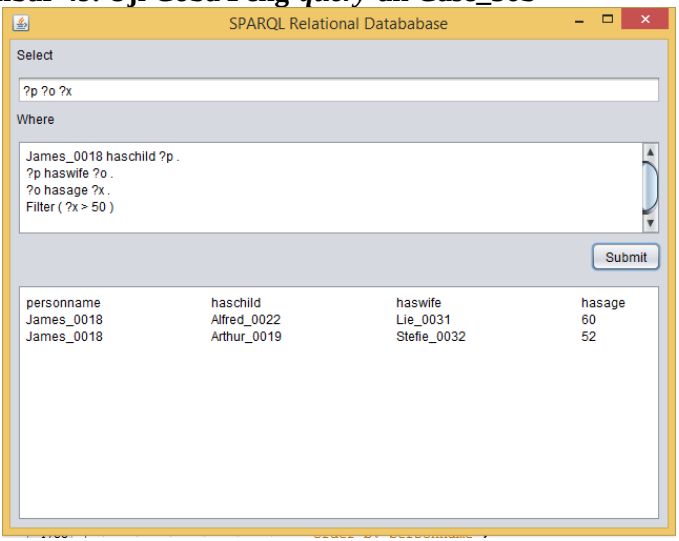
Gambar 47. Uji Coba Peng-*query*-an Case_35c



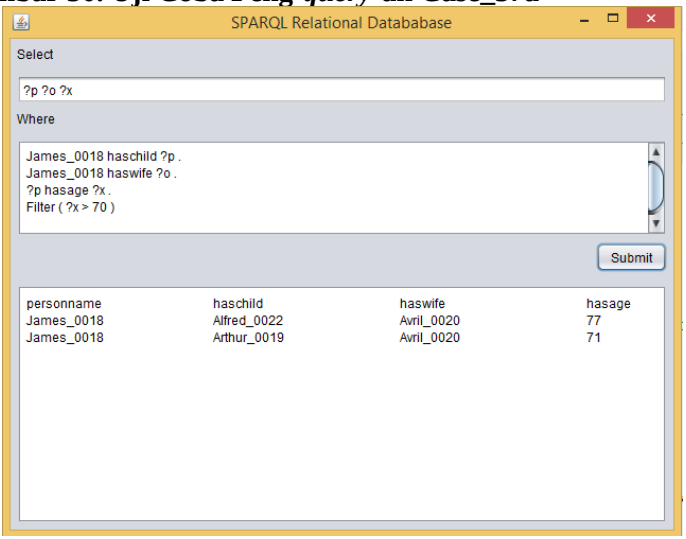
Gambar 48. Uji Coba Peng-*query*-an Case_36a



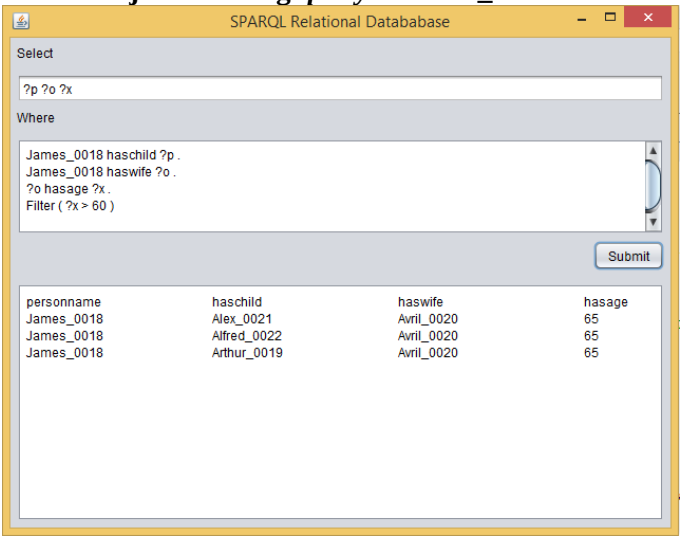
Gambar 49. Uji Coba Peng-*query*-an Case_36b



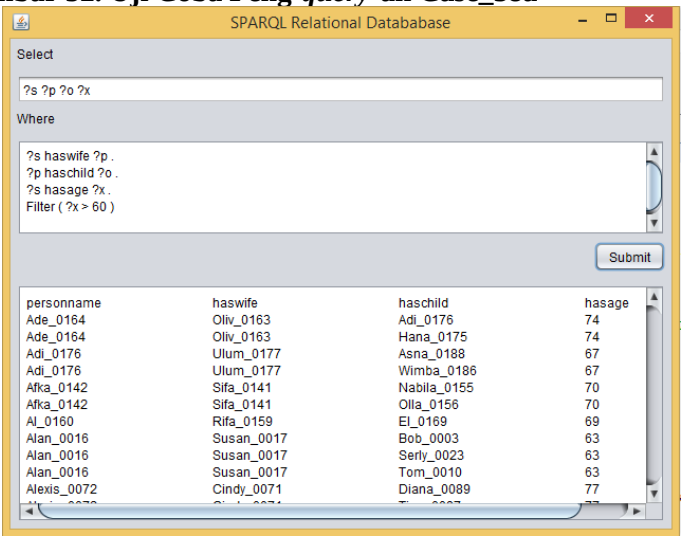
Gambar 50. Uji Coba Peng-*query*-an Case_37a



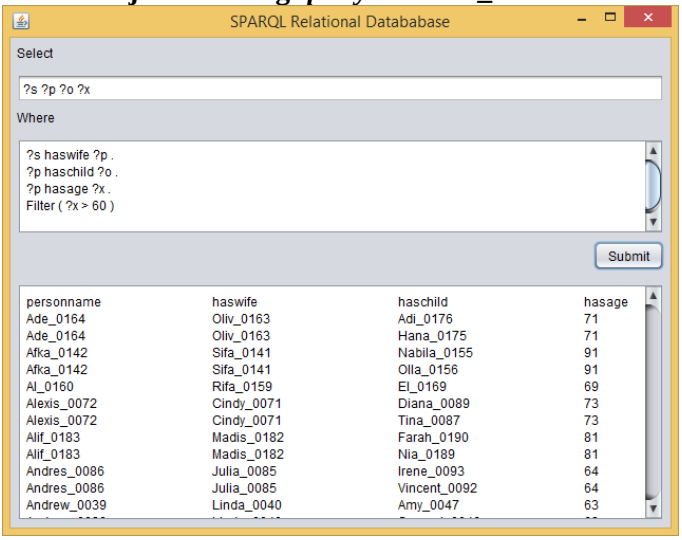
Gambar 51. Uji Coba Peng-*query*-an Case_37b



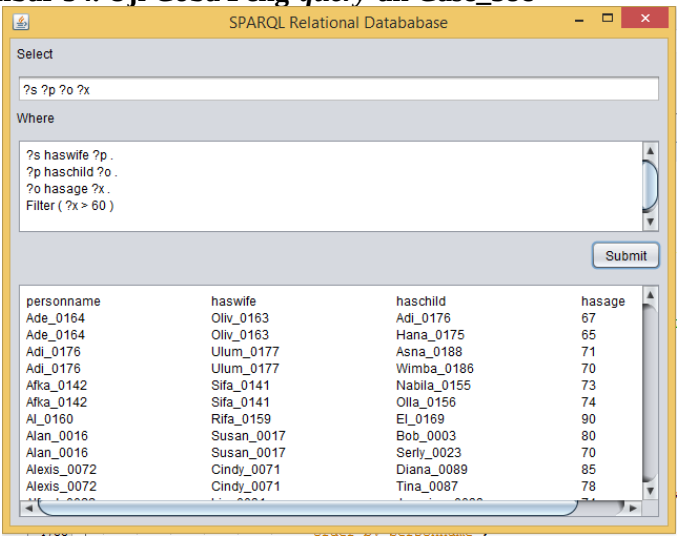
Gambar 52. Uji Coba Peng-*query*-an Case_38a



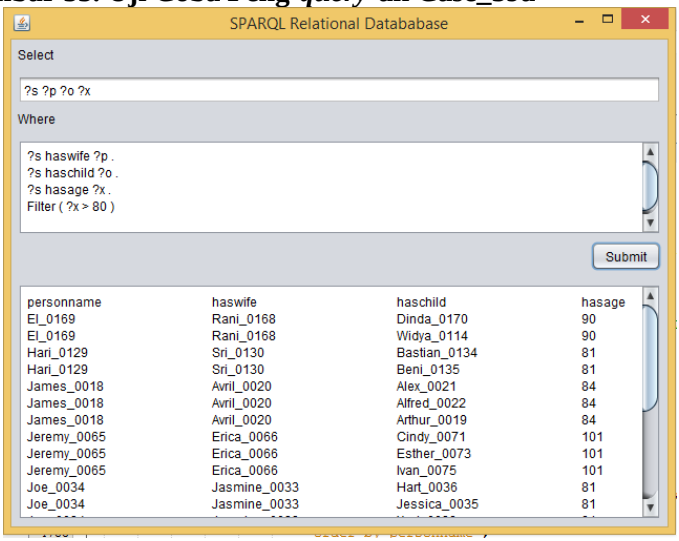
Gambar 53. Uji Coba Peng-*query*-an Case_38b



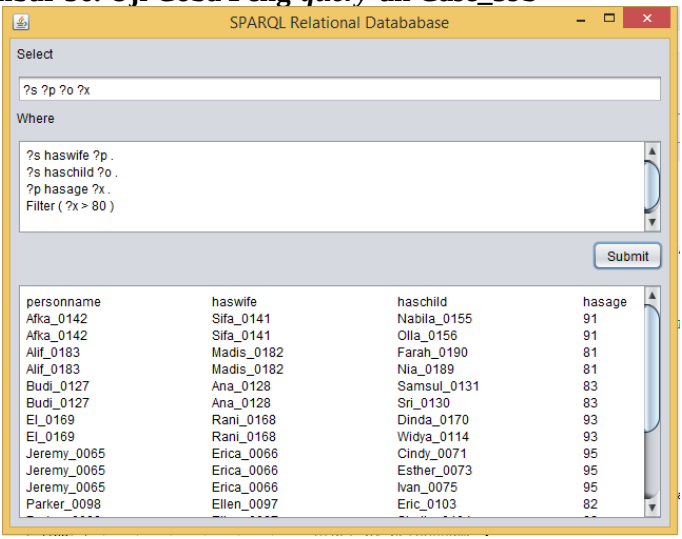
Gambar 54. Uji Coba Peng-*query*-an Case_38c



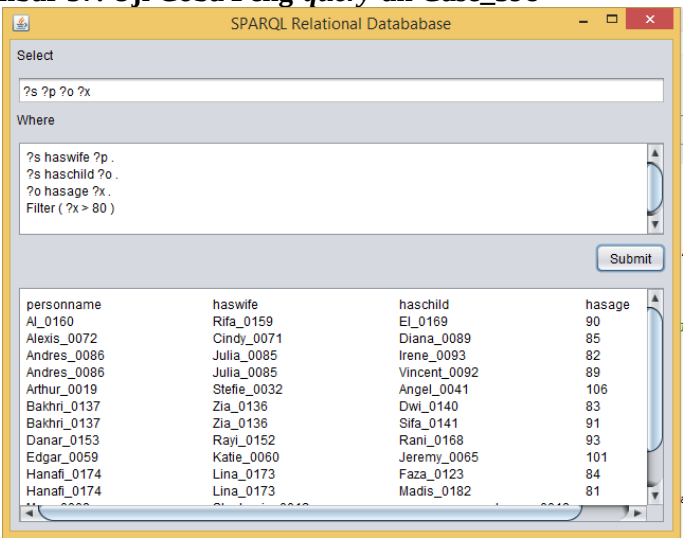
Gambar 55. Uji Coba Peng-*query*-an Case_39a



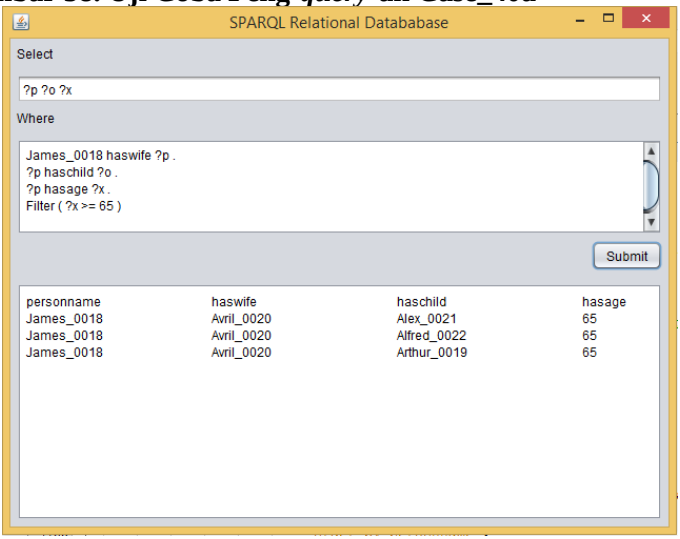
Gambar 56. Uji Coba Peng-*query*-an Case_39b



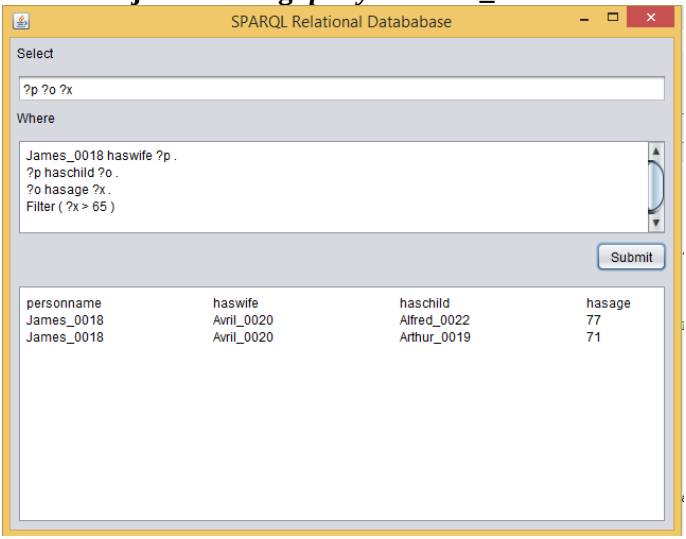
Gambar 57. Uji Coba Peng-*query*-an Case_39c



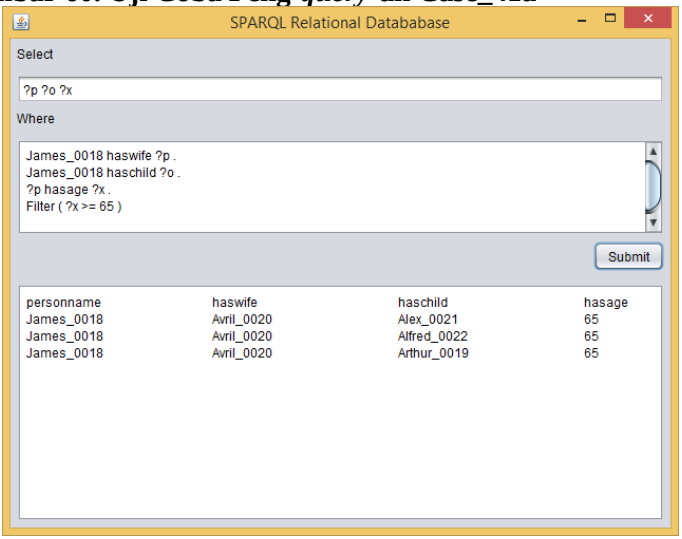
Gambar 58. Uji Coba Peng-*query*-an Case_40a



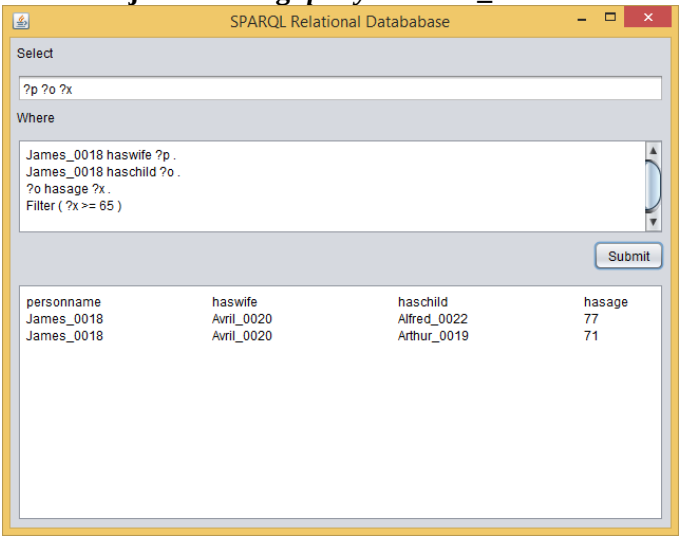
Gambar 59. Uji Coba Peng-*query*-an Case_40b



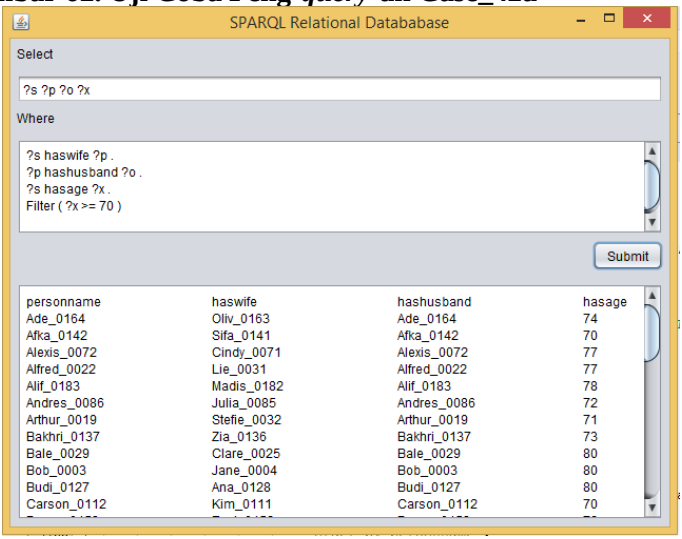
Gambar 60. Uji Coba Peng-*query*-an Case_41a



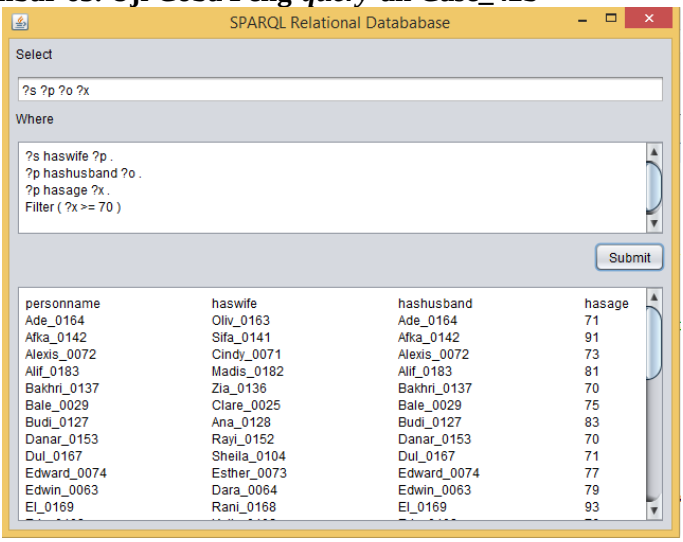
Gambar 61. Uji Coba Peng-*query*-an Case_41b



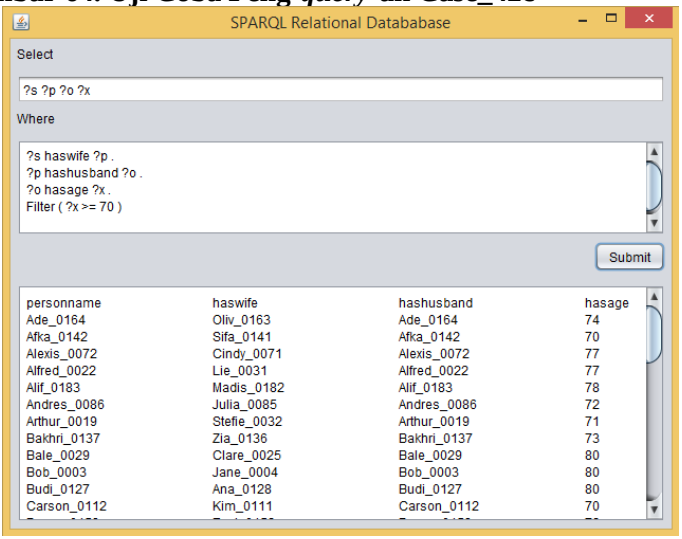
Gambar 62. Uji Coba Peng-*query*-an Case_42a



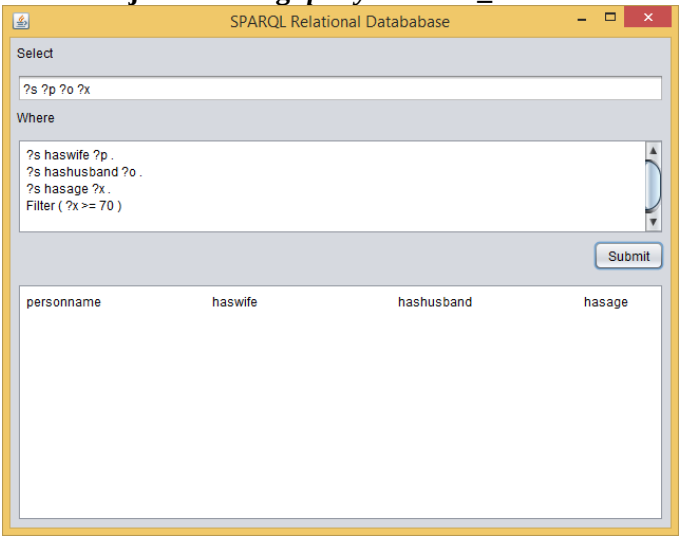
Gambar 63. Uji Coba Peng-*query*-an Case_42b



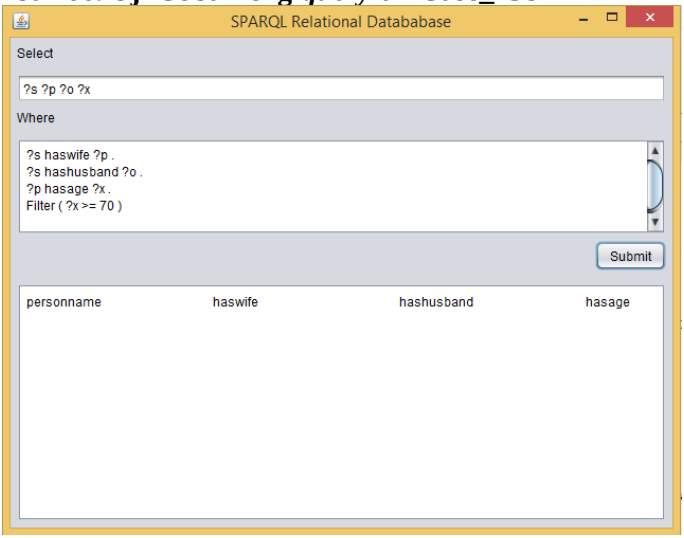
Gambar 64. Uji Coba Peng-*query*-an Case_42c



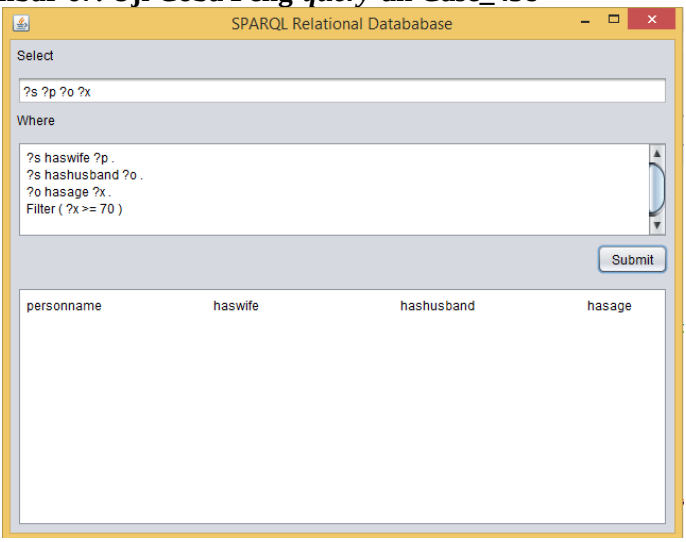
Gambar 65. Uji Coba Peng-*query*-an Case_43a



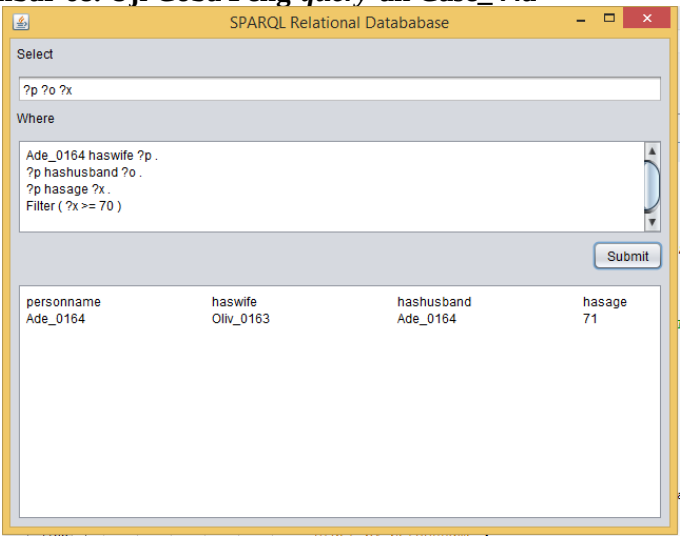
Gambar 66. Uji Coba Peng-*query*-an Case_43b



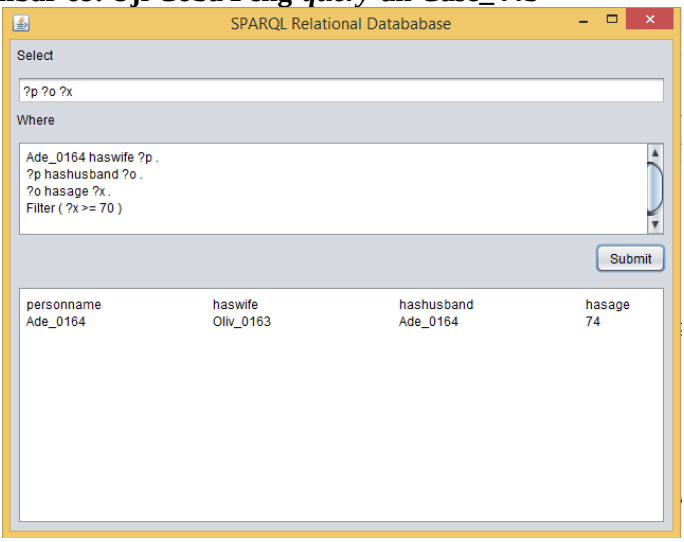
Gambar 67. Uji Coba Peng-*query*-an Case_43c



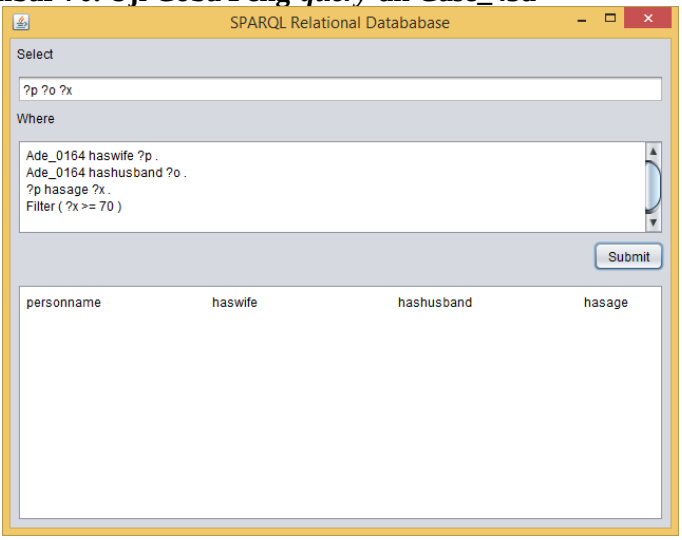
Gambar 68. Uji Coba Peng-*query*-an Case_44a



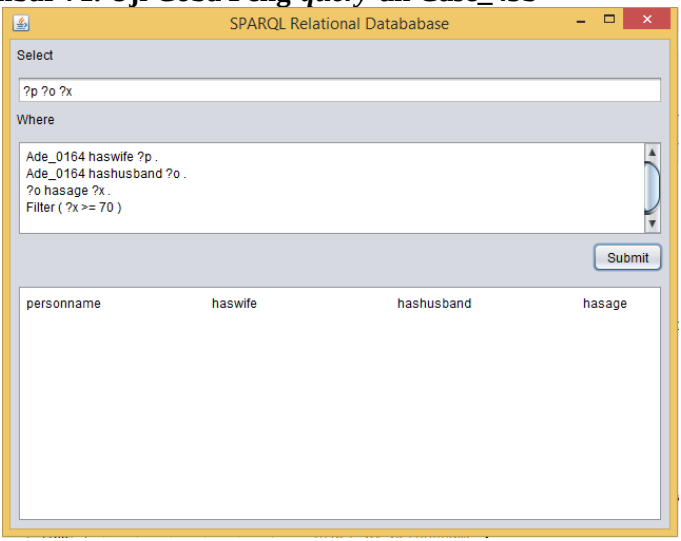
Gambar 69. Uji Coba Peng-*query*-an Case_44b



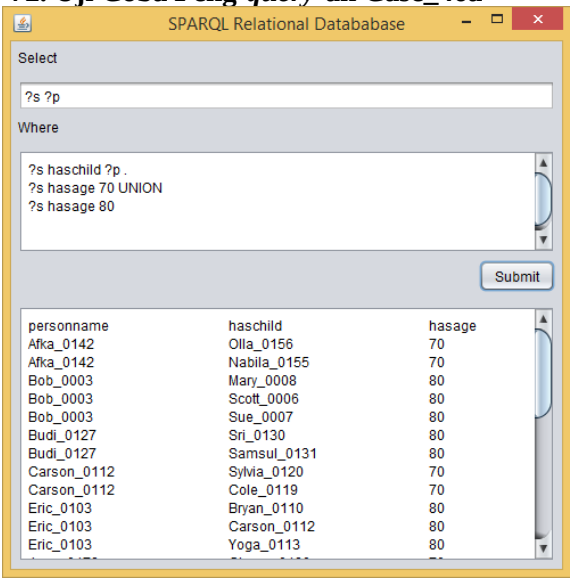
Gambar 70. Uji Coba Peng-*query*-an Case_45a



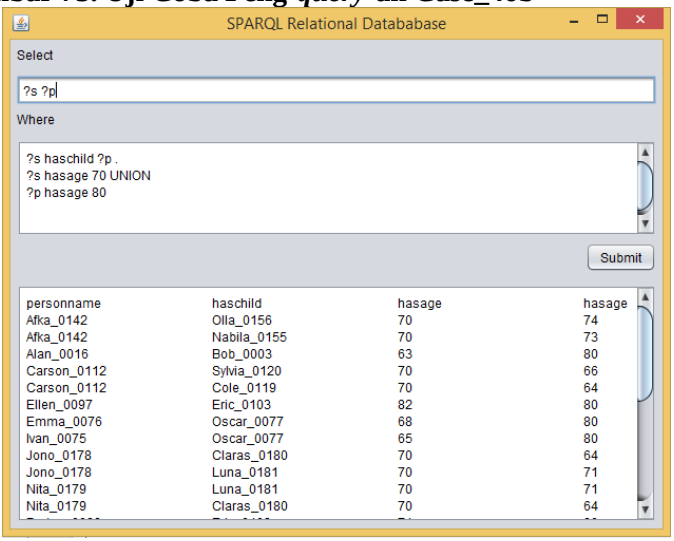
Gambar 71. Uji Coba Peng-*query*-an Case_45b



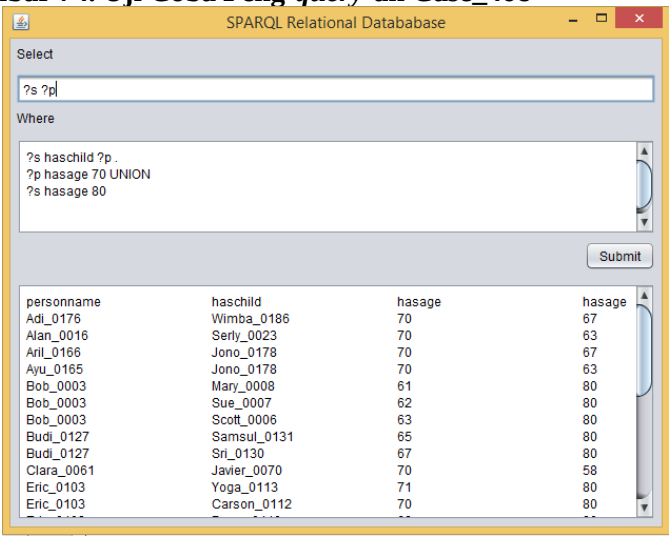
Gambar 72. Uji Coba Peng-*query*-an Case_46a



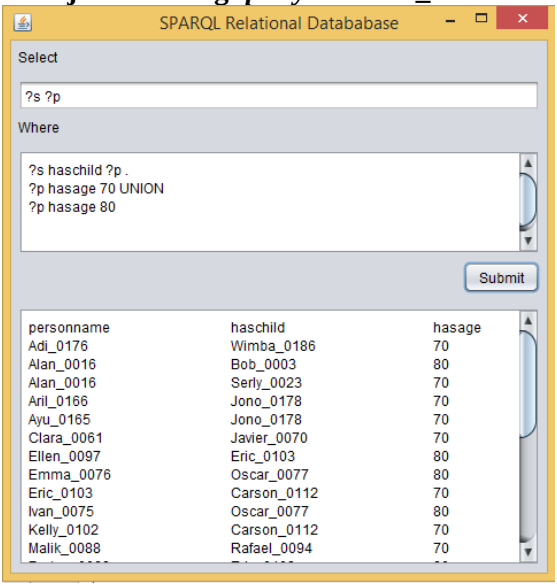
Gambar 73. Uji Coba Peng-*query*-an Case_46b



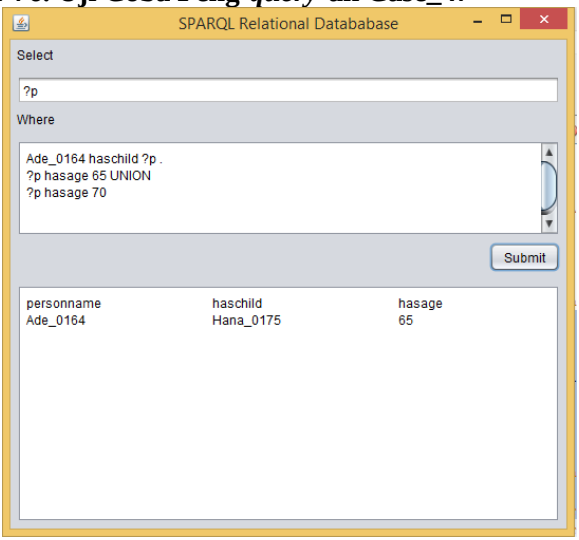
Gambar 74. Uji Coba Peng-*query*-an Case_46c



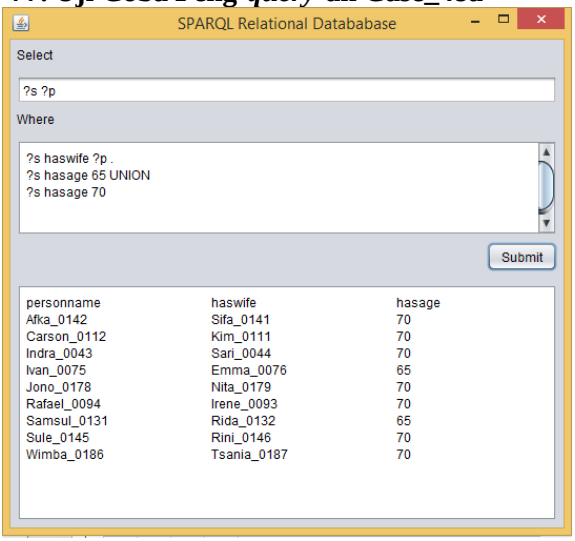
Gambar 75. Uji Coba Peng-*query*-an Case_46d



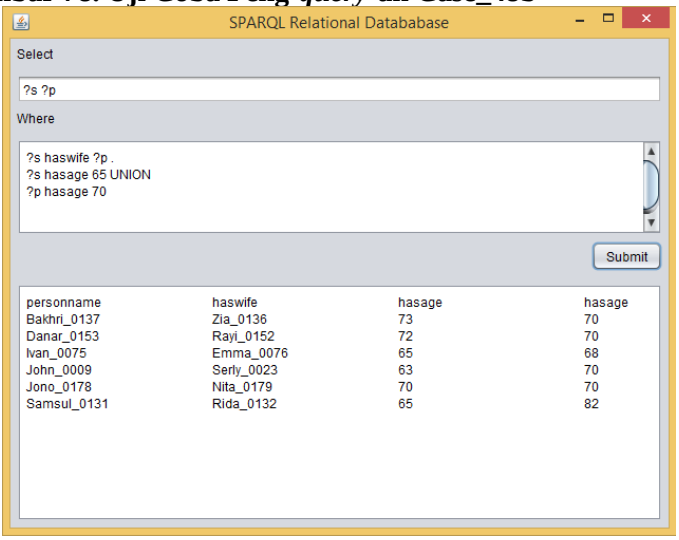
Gambar 76. Uji Coba Peng-*query*-an Case_47



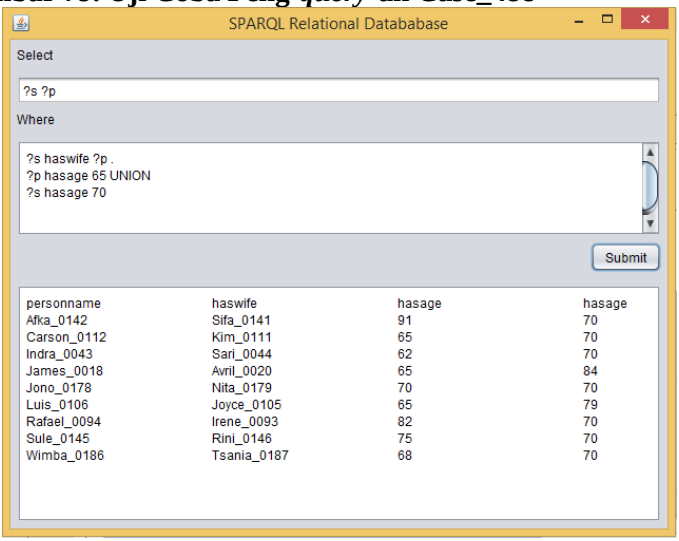
Gambar 77. Uji Coba Peng-*query*-an Case_48a



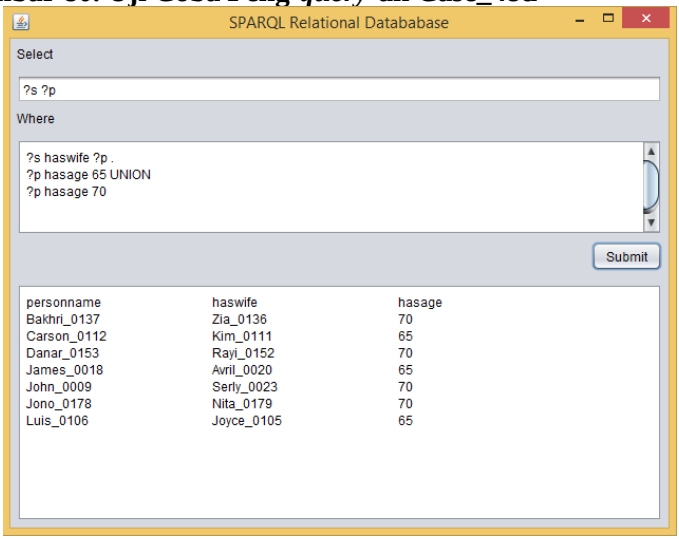
Gambar 78. Uji Coba Peng-*query*-an Case_48b



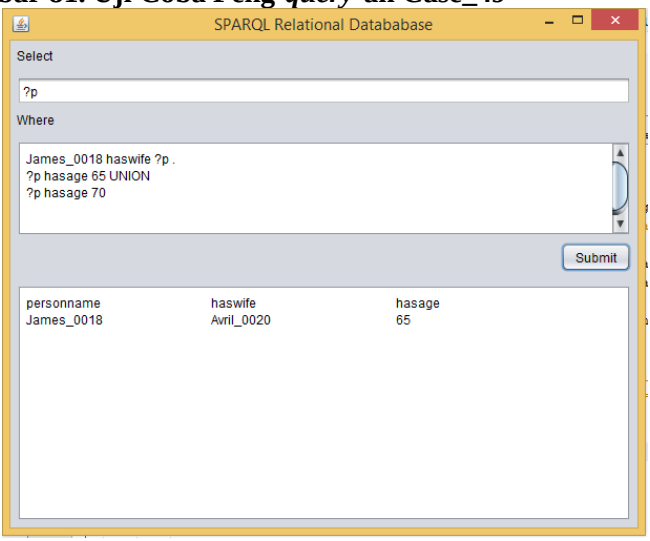
Gambar 79. Uji Coba Peng-*query*-an Case_48c



Gambar 80. Uji Coba Peng-*query*-an Case_48d



Gambar 81. Uji Coba Peng-*query*-an Case_49



(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Kamali Yahya, lahir pada tanggal 15 Februari 1994 di Lamongan. Penulis pernah menempuh pendidikan di MI Muhammadiyah 13 Sendangagung, Paciran, Lamongan (2000-2006), SMP Muhammadiyah 12 Paciran, Lamongan (2006-2009), dan MA Al-Islah Paciran, Lamongan (2009-2012). Saat ini penulis sedang menempuh pendidikan perguruan tinggi di Institut Teknologi Sepuluh Nopember Surabaya di jurusan Teknik Informatika Fakultas Teknologi Informasi angkatan tahun 2012. Dalam menyelesaikan pendidikan S1 penulis mengambil bidang minat Manajemen Informasi (MI). Terlibat aktif dalam organisasi kemahasiswaan selama perkuliahan, yaitu Ikatan Mahasiswa Muhammadiyah (IMM), dan Generasi Baru Indonesia (GenBI) 2014-2015. Penulis dapat dihubungi melalui alamat *email* kamali.yahya12@gmail.com