
TUGAS AKHIR – KS141501

# RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J

## *DEVELOPING USER DEPENDENCY TOOL APPLICATION FOR MS SQL SERVER DATABASE BASED ON NEO4J GRAPH*

HAFIZ PUTRA LUDYANTO
NRP 05211440000087

Dosen Pembimbing
Radityo Prasetianto Wibowo, S.Kom., M.Kom.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

TUGAS AKHIR – KS141501

# RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J

HAFIZ PUTRA LUDYANTO
NRP 05211440000087

Dosen Pembimbing
Radityo Prasetianto Wibowo, S.Kom., M.Kom.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

**FINAL PROJECT – KS141501**

# *DEVELOPING USER DEPENDENCY TOOL APPLICATION FOR MS SQL SERVER DATABASE BASED ON NEO4J GRAPH*

HAFIZ PUTRA LUDYANTO
NRP 05211440000087

Supervisors
Radityo Prasetianto Wibowo, S.Kom., M.Kom.

DEPARTMENT OF INFORMATION SYSTEMS
Faculty of Information and Communication Technology
Sepuluh Nopember Institute of Technology
Surabaya 2018

**LEMBAR PENGESAHAN**

**RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J**

**TUGAS AKHIR**
Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

**HAFIZ PUTRA LUDYANTO**
NRP. 05211440000087

Surabaya, 10 Juli 2018

**KETUA**
**DEPARTEMEN SISTEM INFORMASI**

**Dr. Ir. Aris Tjahyanto, M.Kom.**
**NIP.19650310 199102 1 001**

**LEMBAR PERSETUJUAN**

# RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J

**TUGAS AKHIR**

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :

**HAFIZ PUTRA LUDYANTO**
NRP. 05211440000087

Disetujui Tim Penguji : Tanggal Ujian: 10 Juli 2018
Periode Wisuda: September 2018

**Radityo Prasetianto W., S.Kom., M.Kom.** **(Pembimbing I)**

**Renny Pradina, S.T., M.T.** **(Penguji I)**

**Faizal Johan Atletiko, S.Kom., M.T.** **(Penguji II)**

**RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J**


| | | |
|---|---|---|
| **Nama Mahasiswa** | **:** | **HAFIZ PUTRA LUDYANTO** |
| **NRP** | **:** | **05211440000087** |
| **Departemen** | **:** | **SISTEM INFORMASI FTIF-ITS** |
| **Dosen Pembimbing 1** | **:** | **Radityo Prasetianto Wibowo, S.Kom., M.Kom.** |


## ABSTRAK


*Perusahaan atau organisasi besar memiliki pengguna – pengguna basis data yang beragam. Pengguna – pengguna tersebut terdiri dari aplikasi, pegawai bagian pemasaran, pegawai bagian teknologi informasi, admin basis data, middleware, dan pengguna lainnya. Keragaman pengguna ini ditimbulkan oleh variasi yang terdapat dalam solusi teknologi informasi untuk memenuhi kebutuhan bisnis. Selain itu, keragaman pengguna ini juga disebabkan oleh pengelolaan atau manajemen basis data pada tiap lapisan organisasi, mulai dari admin basis data sampai pengguna basis data di tiap divisi atau departemen. Keragaman pengguna ini meningkatkan kompleksitas konfigurasi akses terhadap basis data, sehingga hal tersebut menimbulkan kesulitan terhadap admin basis data dalam memantau seluruh akses pengguna.*

*Salah satu solusi untuk memantau seluruh akses pengguna adalah menggunakan mekanisme pengelolaan pengguna SQL Server. Kelemahan yang paling utama dalam mekanisme tersebut adalah tampilan GUI (Graphical User Interface) yang terpisah untuk pengguna beserta peran dan objek beserta hak akses. Hal tersebut mengakibatkan admin basis data perlu membuat tampilan (view) yang dihasilkan dari beberapa atau satu eksekusi query* untuk menggabungkan kedua tampilan

yang terpisah. *Metode seperti ini menyebabkan admin untuk membuka SSMS (SQL Server Management Studio) beberapa kali apabila admin membutuhkan data pengguna beserta peran dan hak aksesnya. Metode tersebut menghambat jalannya komunikasi antar pihak admin dan manajemen dalam hal keperluan audit. Metode yang ideal dalam menyajikan data yang standar dan cepat adalah metode yang dapat dipahami oleh admin basis data secara mudah tanpa adanya dukungan atau dokumentasi khusus.*

*Dalam penelitian ini, peniliti bertujuan untuk menampilkan data pengguna basis data beserta peran dan hak aksesnya terhadap objek basis data dalam bentuk visualisasi yang sederhana. Visualisasi tersebut adalah visualisasi berbentuk graph (node dan relasinya). Visualisasi ditampilkan dengan menggunakan basis data Neo4j sebagai penyimpanan graph dalam bentuk situs (web). Visualisasi yang dikembangkan juga disambungkan dengan basis data SQL Server sebagai sumber data untuk proses ekstraksi. Hasil akhir dari tugas akhir ini adalah aplikasi User Dependency Tool yang menampilkan visualisasi pengguna basis data beserta peran, hak akses, objek basis data, dan relasi – relasinya dalam bentuk graph.*

***Kata Kunci : Dokumentasi, Graph, Database, MS SQL Server, Visualisation, User, Neo4jd3.js, Neo4j, Roles, Permissions, Securables.***

**DEVELOPING USER DEPENDENCY TOOL APPLICATION FOR MS SQL SERVER DATABASE BASED ON NEO4J GRAPH**


| | | |
|---|---|---|
| **Name** | **:** | **HAFIZ PUTRA LUDYANTO** |
| **NRP** | **:** | **05211440000087** |
| **Departement** | **:** | **INFORMATION SYSTEM FTIK-ITS** |
| **Supervisor 1** | **:** | **Radityo Prasetianto Wibowo, S.Kom., M.Kom.** |


## ABSTRACT


*Large companies or organizations have multiple database users. These users consist of applications, marketing department employees, information technology department employees, database administrators, middleware, and other users. This diversity of users is caused by the variations contained in information technology solutions to meet business needs. In addition, the diversity of users is also caused by the management or database management on each layer of the organization, ranging from the database admin until the database user in each division or department. This diversity of users increases the complexity of access configuration for the database, thus making it difficult for database administrators to monitor all users's access.*

*One of the solutions to monitor all user's access is to use SQL Server user management mechanism. The main drawback of the mechanism is the separated GUI (Graphical User Interface) displays for users along with roles and objects along with permissions. This results in the database admin needing to create a view that created from multiple or one query execution to combine the two separated displays. This method causes the admin to open SSMS (SQL Server Management Studio) multiple times if the admin requires user data along with its roles and permissions. This method impedes the communication between*

*the admin and management in terms of audit purposes. The ideal method of presenting standard and fast data is a method that can easily be understood by database administrators without any special support or documentation.*

*In this study, the researcher aims to display the data of database users along with their roles and permissions to database objects in the form of simple visualization. This simple visualization is a graph visualization (nodes and relationships). This visualization is displayed using the Neo4j database as a graph storage in the form of a website (web). The developed visualization is also connected to the SQL Server database as the data source for the extraction process. The end result of this final project is the User Dependency Tool application that displays the visualization of database users along with their roles, their permissions, database objects, and their relations in graph.*

***Keywords : Documentation, Graph, Database, MS SQL Server, Visualization, User, Neo4jd3.js, Neo4j, Roles, Permissions, Securables.***

# KATA PENGANTAR

Alhamdulillah atas karunia, rahmat, berkah, hidayah, dan jalan yang telah diberikan oleh Allah SWT selama ini sehingga penulis mendapatkan kelancaran, kemudahan, ilmu, dan kecepatan dalam menyelesaikan tugas akhir dengan judul:

**RANCANG BANGUN APLIKASI USER DEPENDENCY TOOL UNTUK DATABASE MS SQL SERVER BERBASIS GRAPH NEO4J**

Terima kasih atas pihak-pihak yang telah mendukung, memberikan saran, motivasi, semangat, dan bantuan baik materi maupun spiritual demi tercapainya tujuan pembuatan tugas akhir ini. Secara khusus penulis akan menyampaikan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Orang tua penulis dan saudara kandung penulis telah mendokan dan mendukung.
2. Bapak Radityo Prasetianto Wibowo, S.Kom., M.Kom. selaku dosen pembimbing yang meluangkan waktu, memberikan ilmu, petunjuk, dan motivasi untuk kelancaran pengerjaan Tugas Akhir ini.
3. Ibu Renny Pradina, S.T., M.T. dan Bapak Faizal Johan Atletiko, S.Kom., M.T. selaku dosen penguji yang telah memberikan masukan untuk perbaikan tugas akhir ini.
4. Seluruh dosen Jurusan Sistem Informasi ITS yang telah memberikan ilmu yang sangat berharga bagi penulis.
5. Bapak Suyanto selaku paman dari penulis yang telah menyalurkan amalan dan hidayah untuk pengerjaan Tugas Akhir ini dari Allah SWT.
6. Rekan-rekan OSIRIS yang telah berjuang bersama dalam menjalani perkuliahan di Departemen Sistem Informasi ITS.
7. Berbagai pihak yang membantu dalam penyusunan Tugas Akhir ini dan belum dapat disebutkan satu per satu.

Penyusunan laporan ini masih jauh dari sempurna, untuk itu saya menerima adanya kritik dan saran yang membangun untuk

perbaikan di masa mendatang. Semoga buku tugas akhir ini dapat memberikan manfaat bagi semua pembaca.

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

(*Halaman ini sengaja dikosongkan*)

## DAFTAR KODE

(*Halaman ini sengaja dikosongkan*)

# BAB I
# PENDAHULUAN

Pada bab ini akan dijelaskan mengenai gambaran umum terkait tugas akhir yang berisi latar belakang masalah, rumusan masalah, batasan pengerjaan tugas akhir, tujuan, dan manfaat tugas akhir.

## 1.1 Latar Belakang

Perusahaan atau organisasi besar memiliki pengguna – pengguna basis data yang beragam. Pengguna – pengguna tersebut terdiri dari aplikasi, pegawai bagian pemasaran, pegawai bagian teknologi informasi, admin basis data, *middleware*, dan pengguna lainnya. Keragaman pengguna ini ditimbulkan oleh variasi yang terdapat dalam solusi teknologi informasi untuk memenuhi kebutuhan bisnis. Selain itu, keragaman pengguna ini juga disebabkan oleh pengelolaan atau manajemen basis data pada tiap lapisan organisasi, mulai dari admin basis data sampai pengguna basis data di tiap divisi atau departemen.

Keragaman pengguna tersebut meningkatkan kompleksitas konfigurasi akses terhadap basis data, sehingga hal tersebut menimbulkan kesulitan terhadap admin basis data dalam memantau seluruh akses pengguna. Tidak hanya kompleksitas konfigurasi akses saja, dokumentasi pengguna SQL Server yang kurang juga menyulitkan admin basis data dalam memantau seluruh akses pengguna. Dokumentasi yang kurang ini disebabkan oleh tidak adanya aktivitas pencatatan (logging) untuk pengelolaan akses pengguna oleh karyawan, pegawai, dan/atau admin basis data yang lain. Hal tersebut melahirkan *tacit knowledge* dalam suatu perusahaan atau organisasi yang dimiliki oleh hanya sebagian karyawan, pegawai, dan/atau admin basis data. *Tacit knowledge* merupakan pengetahuan yang menempati pikiran orang melalui *trial* dan *error* dan susah atau bahkan tidak mungkin untuk dijelaskan [1].

Salah satu solusi untuk memantau seluruh akses pengguna adalah menggunakan mekanisme pengelolaan pengguna SQL Server [2]. Kelemahan yang paling utama dalam mekanisme tersebut adalah tampilan GUI (Graphical User Interface) yang terpisah untuk pengguna beserta peran dan objek beserta hak akses. Hal tersebut mengakibatkan admin basis data perlu membuat tampilan (view) yang dihasilkan dari beberapa atau satu eksekusi *query* untuk menggabungkan kedua tampilan yang terpisah. Metode seperti ini menyebabkan admin tersebut untuk membuka SSMS (SQL Server Management Studio) beberapa kali apabila admin membutuhkan data pengguna beserta peran dan hak aksesnya. Metode tersebut menghambat jalannya komunikasi antar pihak admin dengan manajemen dalam hal keperluan audit. Metode yang ideal dalam menyajikan data yang standar dan cepat adalah metode yang dapat dipahami oleh admin basis data secara mudah tanpa adanya dukungan atau dokumentasi khusus [3].

Produk atau perangkat lunak yang tersedia sebelumnya telah menyediakan alat untuk menampilkan pengguna beserta tipe transaksi yang dilakukan, seperti ApexSQL Audit [4]. Produk ApexSQL Audit menampilkan jejak audit (audit trail) yang didapatkan dari transaksi yang terjadi di basis data. Permasalahannya adalah bahwa produk tersebut tidak menyediakan suatu tampilan grafis yang dapat memberikan informasi terkait seluruh pengguna beserta peran dan hak aksesnya. Walaupun ApexSQL Audit dapat melihat informasi tersebut dari jejak audit, hal ini tidak mencakup pengguna basis data yang belum pernah melakukan salah satu transaksi dari keempat transaksi yang ada (Select, Insert, Delete, Update). Permasalahan ini juga menyebabkan admin untuk melakukan eksekusi *query* di katalog sistem SQL Server bagi pengguna yang belum pernah melakukan salah satu dari keempatnya.

Dalam penelitian ini, peniliti bertujuan untuk menampilkan data pengguna basis data beserta peran dan hak aksesnya terhadap objek basis data dalam bentuk visualisasi yang sederhana. Visualisasi tersebut adalah visualisasi berbentuk *graph* (node

dan relasinya). Visualisasi ditampilkan dengan menggunakan basis data Neo4j sebagai penyimpanan *graph* dalam bentuk situs (web). Visualisasi yang dikembangkan juga disambungkan dengan basis data SQL Server sebagai sumber data untuk proses ekstraksi. Hasil akhir dari tugas akhir ini adalah aplikasi User Dependency Tool yang menampilkan visualisasi pengguna basis data beserta peran, hak akses, objek basis data, dan relasi – relasinya dalam bentuk *graph*.

## 1.2 Perumusan Masalah

Berdasarkan latar belakang di atas, adapun permasalahan yang dapat diuraikan dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana data pengguna beserta peran dan hak aksesnya terhadap objek didapatkan dari basis data berbasis MS SQL Server?
2. Bagaimana data tersebut diolah untuk membentuk relasi dalam model *graph*?
3. Bagaimana hasil pengolahan data tersebut ditampilkan dalam bentuk visualisasi dengan *graph*?

## 1.3 Batasan Tugas Akhir

Beberapa batasan yang diberlakukan dalam pengerjaan tugas akhir ini adalah:

1. Cakupan untuk *server* atau *instance* yang digunakan hanya terbatas satu.
2. Cakupan untuk basis data yang digunakan adalah basis data yang bukan merupakan basis data sistem, yaitu master, msdb, model, Resource, dan tempdb.
3. Cakupan untuk basis data hanya terbatas pada satu *instance*, tidak pada *instance* yang berbeda – beda.
4. Data yang digunakan untuk pembuatan *graph* terbatas pada data pengguna, peran, hak akses, dan objek basis data yang terkait.
5. Visualisasi tidak mencakup aksi yang dapat merubah data (Create, Update, Delete) dalam basis data.

## 1.4 Tujuan Tugas Akhir

Tujuan pengerjaan tugas akhir ini adalah untuk menampilkan data pengguna beserta peran dan hak aksesnya terhadap objek basis data dalam bentuk visualisasi dengan model *graph*. Aplikasi tersebut memiliki fitur-fitur yakni sebagai berikut:

1. Melakukan ekstraksi *metadata* objek dan pengguna beserta peran dan hak aksesnya dari MS SQL Server.
2. Mengubah hasil ekstraksi pada poin 1 menjadi bentuk *graph* di Neo4j.
3. Menampilkan *graph* ke pengguna dengan platform *web*.
4. Memfilter visualisasi berdasarkan relasi, group, atau kategori lainnya.

## 1.5 Manfaat Tugas Akhir

Tugas akhir ini diharapkan dapat memberikan manfaat antara lain adalah sebagai berikut:

1. Bagi admin basis data, visualisasi ini diharapkan untuk dapat melancarkan komunikasi antar manajemen dan admin dalam keperluan pemantauan akses pengguna basis data (manajemen akses).
2. Bagi akademisi, penelitian ini diharapkan untuk dapat mengembangkan penelitian lainnya dalam bidang *graph* dan memajukan standar atau praktik terbaik dalam hal pemantauan akses pengguna basis data (manajemen akses).
3. Bagi penulis, penelitian ini diharapkan untuk dapat mengajak penulis – penulis lainnya dalam mengembangkan basis data model *graph* di bidang aplikatif lainnya.

## 1.6 Relevansi

Tugas akhir ini memiliki keterkaitan atau relevansi dengan ketentuan topik tugas akhir pada Departemen Sistem Informasi, yakni sebagai berikut:

1. Topik pada tugas akhir ini adalah terkait visualisasi pengguna beserta peran dan hak aksesnya terhadap objek basis data dalam bentuk *graph*, sehingga topik ini berkaitan dengan Laboratorium Akuisisi Data dan Diseminasi Informasi.
2. Tugas akhir ini layak dijadikan sebagai tugas akhir pada tingkat S1, karena produk tugas akhir ini memberikan solusi untuk permasalahan manajemen akses pada basis data dalam hal pemantauan.
3. Tugas akhir ini berkaitan dengan mata kuliah yang terdapat pada Departemen Sistem Informasi, seperti Pemrograman Integratif, Kecerdasan Bisnis, Pengantar Basis Data, Desain Basis Data, dan Analisis dan Desain Perangkat Lunak. Hal tersebut menyatakan bahwa tugas akhir ini memiliki topik yang layak dan sesuai dengan kurikulum pada Departemen Sistem Informasi.

*(Halaman ini sengaja dikosongkan)*

# BAB II
# DASAR TEORI DAN TINJAUAN PUSTAKA

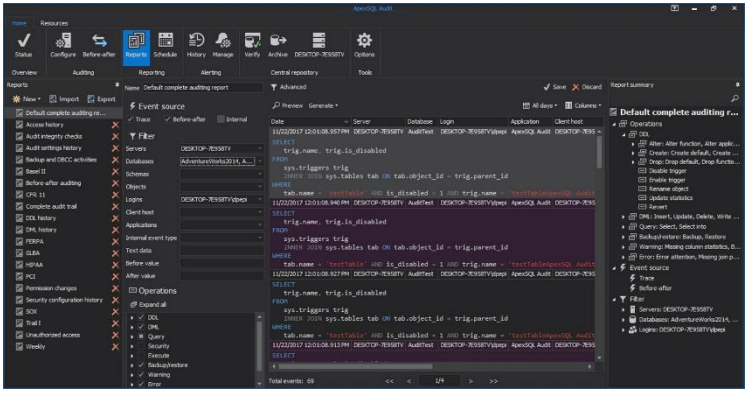
Dasar Teori dan Tinjauan Pustaka merupakan penjelasan mengenai teori-teori terkait yang bersumber dari buku, jurnal, artikel, ataupun penelitian terdahulu yang berfungsi sebagai dasar dalam memahami konsep atau teori penyelesaian permasalahan yang ada dalam pengerjaan tugas akhir. Pada bab ini diberikan uraian meliputi produk sejenis, dasar teori, dan penelitian terdahulu.

## 2.1 Produk Sejenis

Pada bagian ini, produk – produk yang telah menyediakan fitur audit akan dijelaskan dan dibandingkan dengan tugas akhir ini.

### 2.1.1 ApexSQL Audit

ApexSQL Audit adalah aplikasi yang diproduksi oleh ApexSQL, salah satu vendor bersertifikasi emas dari Microsoft. ApexSQL Audit pada Gambar 2.1 digunakan untuk melacak seluruh aktivitas (transaksi pengguna) yang terdapat dalam basis data, manajemen audit, dan memenuhi kebijakan pemerintah terkait dengan privasi data (GDPR). ApexSQL Audit dapat digunakan untuk melihat dan melacak transaksi yang telah dilakukan oleh pengguna. Dari pelacakan tersebut, data pengguna beserta peran dan hak aksesnya dapat diambil dan diolah, tetapi aplikasi ini masih memiliki kekurangan.



**Gambar 2.1 Tampilan Aplikasi ApexSQL Audit**

Kekurangan aplikasi ApexSQL Audit dalam tugas akhir ini adalah sebagai berikut:

1. ApexSQL Audit tidak menampilkan data pengguna beserta peran dan hak aksesnya apabila pengguna tersebut belum pernah melakukan salah satu atau keempat transaksi yang ada (Select, Insert, Update, dan Delete).
2. ApexSQL Audit tidak memberikan visualisasi yang mudah dipahami untuk poin nomor 1.
3. ApexSQL Audit hanya melacak aktivitas yang telah dilakukan oleh pengguna (tidak preventif).

### 2.1.2 Komparasi ApexSQL Audit dengan Tugas Akhir

Pada bagian ini, kekurangan yang terdapat pada aplikasi ApexSQL Audit dapat dibandingkan dengan produk tugas akhir ini. Tabel komparasi ini membandingkan fitur – fitur yang disediakan oleh masing – masing produk. Tabel ini berguna untuk memberikan cakupan dan fokus tugas akhir ini. Tabel komparasi dapat dilihat pada Tabel 2.1.

**Tabel 2.1 Tabel Komparasi ApexSQL, Redgate, dan Produk Tugas Akhir**

| Produk / Fitur | ApexSQL Audit | Tugas Akhir |
|---|---|---|
| Melacak transaksi - transaksi yang dilakukan oleh pengguna | ☑ | ☒ |
| Mendapatkan data pengguna beserta peran dan hak aksesnya dari sistem secara keseluruhan. | ☒ | ☑ |
| Menampilkan data pengguna beserta peran dan hak aksesnya dalam *graph* | ☒ | ☑ |

## 2.2 Dasar Teori

Pada Dasar Teori ini terdiri dari beberapa bagian yang isinya terkait dengan teori-teori dan metode yang digunakan sebagai panduan dalam pengerjaan tugas akhir ini.

### 2.2.1 Katalog Sistem

Katalog Sistem, dalam bahasa Inggrisnya merupakan *System Catalog*, adalah [5] suatu media untuk menyimpan seluruh

informasi detail terkait dengan basis data. Katalog Sistem dalam penelitian [6] juga merupakan media penyimpanan yang digunakan untuk menyimpan informasi terkait dengan *node*, distribusi data, ekstensi pengguna, dan informasi lainnya. Katalog Sistem, dengan nama lain *metastore* [7], merupakan suatu media yang mengandung *metadata* tabel – tabel yang tersimpan, terbentuk, dan termodifikasi di dalam basis data. Katalog Sistem, menurut [8], merupakan suatu media yang digunakan oleh sistem basis data untuk mengakses data – data yang disimpan pengguna. Katalog Sistem dalam artikel [9] bertindak sebagai skema untuk memetakan informasi dari pengguna ke dalam sistem basis data dan menyajikan metode untuk menyimpan informasi ke dalam basis data. Dari definisi - definisi sistem katalog yang telah disebutkan, sistem katalog adalah suatu media penyimpanan yang telah menjadi bawaan dari sistem basis data untuk menyimpan *metadata*, memberikan skema/struktur informasi, menyajikan metode untuk penyimpanan, dan menjadi *framework* dalam suatu basis data.

Katalog Sistem dalam MS SQL Server menyajikan informasi yang digunakan oleh SQL Server *Database Engine* dan ditampilkan dalam bentuk *view* [10]. Katalog Sistem tersebut mengandung informasi *metadata* objek yang dimiliki oleh pengguna. Katalog Sistem dalam SQL Server ditampilkan dalam *view* dengan alasan bahwa metode ini [10] adalah metode yang efisien untuk mendapatkan informasi dalam hal perubahan, penyajian, dan pengumpulan data. Katalog Sistem dalam SQL Server dibagi menjadi tiga kategori [11], yaitu *Catalog Views*, *Information Schema Views*, dan *Dynamic Management Views*. *Catalog Views* lebih berfokus pada informasi yang menyangkut basis data, seperti *mirroring*, pelacakan perubahan, pencarian teks, dan informasi basi data lainnya. *Information Schema Views* mencakup informasi pada objek – objek yang terdapat dalam basis data, seperti tabel, *stored procedure*, *views*, dan objek lainnya. *Dynamic Management Views* mengandung informasi yang berfokus pada

keadaan server dan basis data, seperti performa server, dan lebih berfokus pada aktivitas pemonitoran.

### *2.2.2 Principals*

*Principals* dalam SQL Server, menurut bab [12], [13], merupakan suatu akun yang dapat masuk dan mengakses SQL Server untuk melakukan aksi terhadap objek dalam basis data (securables). Definisi yang lebih jelas dicantumkan pada bab [14], [15], bahwa *principals* adalah suatu individu atau kelompok yang dapat mengakses objek di dalam basis data dan memberikan instruksi kepada basis data untuk mengelola objek tersebut, seperti membuat tabel, menghapus tabel, dan transaksi lainnya. *Principals* memiliki tiga kategori [14], [15], yaitu *windows-level principals*, *SQL Server-level principals*, dan *database-level principals*. *Windows-level principals* adalah pengguna (entitas atau akun) yang mengakses SQL Server dengan menggunakan platform *authentication* Windows. *SQL Server-level principals* adalah pengguna (entitas atau akun) yang menggunakan platform bawaan MS SQL Server untuk mengakses SQL Server. *Database-level principals* adalah pengguna yang mengakses basis data dalam SQL Server berdasarkan akun, peran, dan hak akses yang telah diberikan. *Database-level principals* harus dipetakan terhadap *SQL Server-level principals* atau *Windows-level principals* agar akun bisa digunakan.

### 2.2.3 *Roles* (Peran)

*Roles* adalah sebuah kelompok untuk pengguna yang memiliki kesamaan dalam fungsional tertentu, sehingga hak akses dapat didistribusikan ke kelompok daripada tiap individu pengguna [14], [16], [17]. Dalam penelitian [18], pengertian *roles* mirip dengan definisi pada kalimat sebelumnya dengan perbedaan bahwa *roles* juga dapat dibentuk semacam hirarki (tingkatan). Pengertian *roles* tidak terbatas pada kelompok dari hak akses (permissions) untuk beberapa pengguna, tetapi pengertian *roles* [19] juga mencakup suatu kelompok hak akses yang mendefinisikan akses ke dalam basis data dan operasi – operasi

yang dapat dilakukan dari akses tersebut. *Roles* memiliki hubungan pemetaan dengan pengguna (principals) [20], [21]. Dari definisi – definisi yang telah disebutkan, *roles* merupakan suatu kelompok hak akses yang didistribusikan ke beberapa pengguna untuk mendefinisikan operasi – operasi yang dapat dilakukan terhadap objek dalam basis data.

*Roles* dalam SQL Server merupakan suatu media untuk mendistribusikan hak akses kepada sekelompok pengguna dibandingkan ke tiap pengguna [22]. *Roles* dalam SQL Server mendefinisikan hal – hal yang berhak atau tidak dilakukan oleh sekelompok pengguna [23]. *Roles* dalam SQL Server dibagi menjadi tiga tipe, yaitu:

1. *Fixed server roles*: *Roles* ini merupakan *roles* bawaan dari SQL Server yang tidak dapat dirubah hak aksesnya dan memiliki cakupan pada level server dan digunakan untuk melakukan akses pada SQL Server [16], [22].Contoh dari *fixed server roles* adalah sysadmin, serveradmin, securityadmin, processadmin, dan *roles* lainnya.
2. *Fixed database roles*: *Roles* ini merupakan *roles* bawaan dari SQL Server yang tidak dapat dirubah hak aksesnya dan memiliki cakupan pada level basis data [16]. *Roles* ini tersedia secara bawaan di tiap basis data yang dibuat [24]. Contoh dari *fixed database roles* adalah db_owner, db_securityadmin, db_accessadmin, db_backupoperator, dan *roles* lainnya.
3. *Application roles*: *Roles* ini merupakan *roles* yang terdapat pada level *database* dan biasa digunakan untuk aplikasi eksternal di luar SQL Server [25]. *Roles* tersebut merupakan *roles* bawaan dari SQL Server yang tidak memiliki anggota dan tidak aktif, sehingga *roles* ini perlu dibuat dan didistribusikan hak aksesnya, juga diaktifkan di dalam SQL Server [16], [25].

### 2.2.4 *Permissions* (Hak Akses)

*Permissions* adalah suatu media untuk menentukan subjek dan operasi yang dapat dilakukan oleh subjek tersebut terhadap objek basis data [26], [27]. Dalam penelitian [28], *permissions* adalah suatu alat untuk membatalkan atau mengijinkan operasi *UPDATE* pada basis data. *Permissions* merupakan hasil distribusi pemetaan izin (hak akses) objek basis data terhadap pengguna [21]. Pada artikel [29], *permissions* merupakan suatu bentuk persetujuan akses pengguna terhadap objek dalam basis data. *Permissions* merupakan suatu hal yang perlu diberikan (GRANT) atau dicabut (REVOKE) oleh pemilik sah *permissions* berdasarkan keputusan yang dimilikinya [27], [30], [31]. Dari definisi – definisi yang telah disebutkan, kesimpulan dari *permissions* adalah suatu media untuk membatasi operasi – operasi yang dilakukan oleh pengguna dalam basis data dan diberikan dari pemilik yang berwenang.

*Permissions* dalam SQL Server merupakan suatu hal yang harus diberikan secara eksplisit terhadap pengguna pada suatu objek agar objek tersebut dapat diakses [32], [33]. *Permissions* pada SQL Server berlaku pada level *server* dan *database* dan berjumlah 237 untuk SQL Server 2017 dan SQL Database [33]. Dalam SQL Server, operasi pada *permissions* menggunakan tiga *statements*, yaitu GRANT, REVOKE, dan DENY. *Statement* GRANT memberikan *permissions* pada pengguna (principals). *Statement* REVOKE mencabut *permissions* pada pengguna tertentu secara tidak penuh. *Statement* DENY mencabut *permissions* pada pengguna secara penuh, sehingga *permissions* tidak dapat diwariskan dari induk (parent roles).

### *2.2.5 Securables*

Menurut penelitian [34], *securables* adalah suatu aset yang menjadi objek dari suatu sistem akses kontrol. Pada artikel [35], [36], *securables* adalah sumber daya yang diregulasikan aksesnya oleh sistem SQL Server. Dari definisi – definisi yang telah disebutkan, *securables* adalah aset SQL Server yang dilindungi oleh suatu mekanisme kontrol akses dan dapat

diakses oleh pengguna yang berhak. *Securables* dibagi menjadi tiga cakupan [35], [36], yaitu *server*, *database*, dan *schema*. Contoh dari cakupan *server* adalah *endpoint*, *login*, *server role*, *database*, dan *securables* cakupan *server* lainnya. Contoh dari cakupan *database* adalah *application role*, *assembly*, *certificate*, *contract*, dan *securables* cakupan *database* lainnya. Contoh dari cakupan *schema* adalah *type*, *XML schema*, *aggregate*, *function*, *procedure*, dan *securables* cakupan *schema* lainnya.

### 2.2.6 DBMS (Database Management System) MS SQL Server

DBMS MS SQL Server adalah sistem manajemen basis data berbasis relasi yang dikembangkan oleh Microsoft. MS SQL Server juga mendukung beberapa ragam pemrosesan transaksi, kecerdasan bisnis, dan aplikasi analitis di lingkungan *enterprise* [37]. MS SQL Server merupakan salah satu produk Microsoft yang lebih berfokus pada pengolahan, manajemen, dan penyimpanan data [38]. MS SQL Server memiliki beberapa edisi, seperti enterprise, developer, dan edisi lainnya, yang ditargetkan untuk segmen pelanggan tertentu. MS SQL Server merupakan salah satu produk SQL unggulan diantara produk unggulan lainnya, berupa Oracle Database, IBM DB2, dan MySQL.

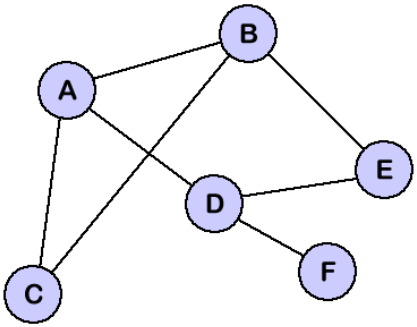
MS SQL Server mendukung ANSI SQL, bahasa SQL standar dan asal. MS SQL Server juga mendukung bahasa T-SQL [37] yang merupakan bahasa SQL implementasi dari Microsoft. Bahasa T-SQL milik Microsoft ini dapat digunakan untuk melakukan transaksi dalam basis data dan memonitor juga mengelola performa dari MS SQL Server. T-SQL ini juga dapat digunakan untuk melakukan operasi pada *instance*/*server* yang berbeda secara bersamaan. Bahasa T-SQL juga digunakan sebagai sebuah *stored procedure* yang disimpan pada *server* dan tidak disimpan di aplikasi eksternal.

MS SQL Server memiliki beberapa teknologi, yaitu Database Engine, Reporting Services, Machine Learning Services, dan

teknologi lainnya [37], [39]. Database Engine merupakan teknologi utama dalam manajemen data, yaitu mengambil, mengolah, dan menyimpan data. Machine Learning Services adalah teknologi yang berfokus pada integrasi dengan bahasa – bahasa *Machine Learning*, seperti R dan Python. Teknologi Reporting Services adalah teknologi yang berguna untuk membuat dan memublikasikan laporan dari basis data ke pihak manajemen. Teknologi – teknologi yang terdapat dalam MS SQL Server memiliki peran masing – masing dalam memenuhi kebutuhan bisnis organisasi atau perusahaan.

### *2.2.7 Graph*

*Graph* adalah suatu komponen yang mengandung *node* dan *edge* [40], [41]. *Graph* dalam penelitian [42] diartikan sebagai seluruh relasi bertipe biner yang memiliki kesamaan jangkauan dan wilayah. Menurut bacaan [43], [44], *graph* diartikan sebagai representasi dari entitas, sebuah objek yang merupakan suatu unit, dan relasi, sebuah koneksi yang menghubungkan dua atau lebih entitas yang berbeda, dari entitas tersebut. *Graph* dikatakan [45] sebagai salah satu struktur yang yang berguna untuk memodelkan objek dengan interaksinya. Maka dari itu, *graph* dapat disimpulkan sebagai suatu komponen yang mengandung *node* dan relasi – relasi di dalamnya, sehingga jaringan struktur data terbentuk dari *node* – *node* yang terhubung.

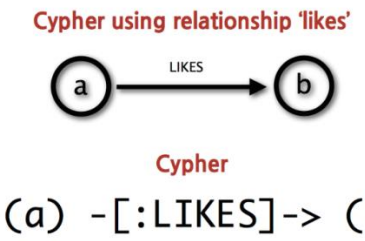



**Gambar 2.2 Contoh Graph (Sumber: https://courses.cs.vt.edu)**

### 2.2.8 Neo4j Graph DBMS


**Gambar 2.3 Logo Neo4j (Sumber: https://neo4j.com)**

Neo4j Graph DBMS adalah suatu basis data NOSQL yang dibuat di atas JVM (Java Virtual Machine) [46]. Neo4j menawarkan fitur *persistence*, performa yang tinggi, skalabilitas, dokumentasi yang baik, dan komunitas yang aktif [47]. Menurut [46], [48], [49], untuk model data yang saling terkoneksi, Neo4j merupakan pilihan yang cocok dalam manajemen data, karena selain performa yang cepat dibandingkan dengan basis data relasi, struktur datanya cocok digunakan untuk data – data berbentuk jaringan, contohnya data sosial, data biologi, dan data – data lainnya. Neo4j mendukung ACID [50] (Atomicity, Consistency, Isolation, Durability) pada level transaksi. Hal tersebut menunjukkan bahwa Neo4j telah mendukung konsistensi penuh dalam hal modifikasi dan pembacaan data.




**Gambar 2.4 Contoh Cypher Query Neo4j (Sumber: https://neo4j.com)**

Neo4j menggunakan beberapa bahasa untuk mengakses *graph* yang terdapat di dalamnya. Dua diantaranya adalah Cypher dan Gremlin [47], [51]. Cypher memiliki kemiripan terhadap bahasa SQL, sehingga bahasa ini lebih mudah dipahami untuk pengguna (high-level language). Gremlin merupakan bahasa asal untuk Neo4j yang digunakan dalam menjelajahi seluruh *graph* dalam basis data. Bahasa Gremlin ini lebih dikhususkan untuk para pengembang. Dengan menggunakan bahasa Cypher, penjelajahan untuk seluruh *graph* dalam basis data tidak perlu dilakukan, karena *query* yang diberikan sudah memiliki bentuk yang efektif [51].

Menurut penelitian [50], Neo4j memiliki tiga bagian arsitektur, yaitu struktur sistem, struktur penyimpanan, dan struktur *cluster HA* (High Availability). Dalam struktur sistem, Neo4j menggunakan dua metode untuk pengaksesan data, agar pengaksesan data semakin cepat. Struktur penyimpanan Neo4j bergantung pada suatu *file* yang bernama *neostore*. *Neostore* ini mengandung dua komponen vital dari Neo4j, yaitu *node* dan relasinya. Struktur *cluster HA* bertindak untuk menjaga Neo4j agar basis data tetap tersedia meskipun jumlah akses bersamaan pada basis data tersebut mencapai puncaknya.

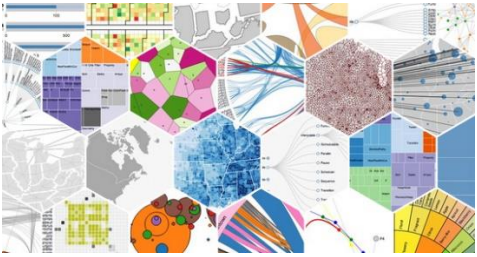
### 2.2.9 D3.js

**Gambar 2.5 Logo D3.js (Sumber: https://dribbble.com)**

D3.js adalah suatu *library* baru yang ditulis dengan menggunakan bahasa Javascript dan pendekatan transparan dan representasi untuk visualisasi dalam *web* [52], [53]. D3.js merupakan salah satu alat yang terkenal [54] dalam membuat visualisasi untuk *web* di antara Google Chart, Flex, dan alat - alat lainnya. D3.js merupakan alat yang mudah untuk dipelajari [53] dan alat tersebut lebih mengutamakan pada manipulasi DOM [52](Document Object Model). Dalam membuat visualisasi di D3.js, alat tersebut menggunakan konsep pemuatan data, pengikatan data, transformasi elemen secara analitis, dan elemen yang berlebih [55]. D3.js tidak memiliki standar atau format tertentu untuk memasukkan data [54], tetapi D3.js menggunakan pemetaan masukan yang dapat dimodifikasi sedemikian rupa oleh pengguna sesuai dengan preferensinya [55]. D3.js juga mendukung visualisasi yang kompleks, seperti geografis, geometris, dan perilaku [56]. Dalam sifat asalnya yang dapat memanipulasi DOM, D3.js bekerja dengan baik untuk HTML5, CSS3, dan *native* Javascript, sehingga D3.js mudah untuk direkayasa oleh para Front-End Developer.

D3.js juga mendukung ekstensi SVG. Ekstensi atau format SVG ini digunakan untuk membuat suatu visualisasi yang terdapat pada HTML5 dan tidak didukung secara penuh oleh CSS3. Format tersebut juga memiliki aturan tersendiri untuk membuat suatu elemen, seperti *path*, *text*, dan aturan lainnya. D3.js mendukung standar W3C Selectors API dan digunakan

untuk memilih elemen dalam hal manipulasi DOM. D3.js adalah alat yang memiliki ukuran ringan karena D3.js hanya mendukung kapabilitas yang dimiliki oleh *browser* baru. Hal tersebut menyatakan bahwa D3.js tidak menggunakan *framework* untuk mendukung *browser* lama, seperti Modernizr. D3.js juga memiliki komunitas yang mendukung alat tersebut berkembang sampai saat ini.



**Gambar 2.6 Contoh Grafik - Grafik yang Dibuat oleh D3.js (Sumber: https://d3js.org)**

## 2.3 Tinjauan Pustaka

Pada bab ini akan dipaparkan beberapa penelitian sebelumnya terkait dengan permasalahan yang diangkat dalam pengerjaan tugas akhir ini. Beberapa penelitian sebelumnya akan dijadikan acuan dan pendukung dalam pengerjaan tugas akhir ini, yaitu sebagai berikut:

**Tabel 2.2 Tinjauan Pustaka (Studi Literatur)**

| | |
|---|---|
| **Judul** | RANCANG BANGUN APLIKASI VISUALISASI DATABASE SQL SERVER DENGAN DYNAMIC MANAGEMENT VIEW BERBASIS GRAPH NEO4J UNTUK MEMETAKAN RELASI IMPLISIT PADA DATABASE |
| **Penulis** | Hufadz Izzudin Robbani |
| **Metodologi** | Dalam penelitian tersebut, penulis merancang dan membangun aplikasi yang dapat memetakan |

| | |
|---|---|
| | relasi implisit pada basis data ke dalam *graph*. Aplikasi tersebut mengambil *query* dari DMV (Dynamic Management View), lalu menyimpan *query* tersebut untuk proses lebih lanjut. Dari *query* yang telah disimpan, aplikasi tersebut melakukan *parsing* untuk mendapatkan relasi implisit, sekaligus *foreign key*, tabel, dan skema yang terdapat dalam hasil *parsing*. Dari hasil *parsing* yang didapatkan, aplikasi menyimpannya ke dalam Neo4j. Setelah itu, aplikasi menampilkan *graph* dari Neo4j menggunakan *library* D3.js. |
| **Keterkaitan dengan Tugas Akhir** | Penelitian ini memiliki hubungan yang erat terkait dengan penggunaan *library* D3.js dan aplikasi Neo4j. *Library* D3.js digunakan untuk menampilkan relasi dari objek dalam basis data dan Neo4j digunakan untuk menyimpan relasi – relasi tersebut. Keterkaitan lain juga terdapat pada pengolahan katalog sistem dalam SQL Server untuk mendapatkan data – data yang dibutuhkan, seperti nama objek, tipe objek, nama *principals*, dan data lainnya. |
| **Judul** | Database Object Dependency Tool |
| **Penulis** | 1. Ivana Lieskovska Drabikova<br>2. Karol Matiasko<br>3. Anton Lieskovsky |
| **Metodologi** | Penelitian ini terkait dengan pengembangan aplikasi DB Object Mapper Tool yang digunakan untuk memetakan objek yang terdapat dalam basis data dan relasinya. Objek yang disebutkan di atas tidak hanya terbatas pada tabel, tetapi objek tersebut mencakup *stored procedure*, *view*, *trigger*, dan objek lainnya. Arsitektur yang |

digunakan dalam mengembangkan aplikasi DB Object Mapper Tool adalah sebagai berikut:

| **Graph** | **DB Object Mapper Tool** |
|---|---|
| Node | DB Object (view, trigger, ta function) |
| Edge | Relation of interaction (inte between DB oQjects) |

Tata cara yang digunakan untuk menamakan objek dalam DB Object Mapper Tool sesuai arsitektur di atas adalah sebagai berikut:

| Class: Node | | |
|---|---|---|
| | None Name | Node name (table, trigger, view, procedure, function name). |
| | Node Type | Node Type (table, trigger, view, procedure, function). |
| | List of Node columns <List> | List of columns name [column |

| | | | |
|---|---|---|---|
| | | | name: date type]. |
| | | Lists of reference objects<br>(reference = relation of interaction) | |
| | | Node_to_Tables <List> | List of reference tables related to the Node. |
| | | Node_to_Views <List> | List of reference views related to<br>the Node. |
| | | Node_to_Triggers <List> | List of reference triggers related to<br>the Node. |
| | | Node_to_Procedures <List> | List of reference procedures related to the Node. |
| **Keterkaitan** | Keterkaitan penelitian ini dengan tugas akhir adalah penggunaan arsitektur dalam memetakan hubungan antar objek. Hal tersebut berguna untuk memetakan hubungan antar pengguna dengan objek dalam tugas akhir. | | |
| **Judul** | Visualisation of Relational Database Structure by Graph Database | | |
| **Penulis** | 1. Przemysław Idziaszek | | |

| | |
|---|---|
| | 2. Wojciech Mueller<br>3. Janina Rudowicz-Nawrocka<br>4. Michał Gruszczy´nski<br>5. Sebastian Kujawa<br>6. Karolina Górna<br>7. Kinga Balcerzak |
| **Metodologi** | Penelitian ini mengembangkan aplikasi yang bernama RELATIONS-Graph. Aplikasi tersebut berguna untuk memetakan seluruh tabel dan relasinya di basis data ke dalam bentuk *graph*. Aplikasi tersebut menggunakan dua metode dalam membentuk *graph*. Metode pertama adalah menggunakan GraphClient *class*. Metode tersebut menggunakan API (Application Programming Interface) yang telah disediakan oleh Neo4jClient, sebuah *package* klien .NET untuk Neo4j. Metode ini langsung merubah objek ke *node* dan relasi ke *edge* dengan menggunakan *method* Cypher yang telah disediakan API. Metode kedua menggunakan *query* Cypher sebagai *string*. *Query* sebagai *string* dimasukkan ke dalam konstruktor *CypherQuery* untuk membentuk *node* dan *edge*. Metode ini berguna untuk merubah struktur *graph* sesuai dengan preferensi pengguna. |
| **Keterkaitan** | Keterkaitan penelitian ini dengan tugas akhir ini adalah penggunaan metode pertama dalam merubah seluruh tabel dan relasinya ke dalam *node* dan *edges* dalam *graph*. Metode pertama ini berguna untuk membentuk *node* dan *edges* untuk pengguna, objek, dan relasinya dalam tugas akhir ini. Metode kedua juga akan digunakan apabila |

| | |
|---|---|
| | pertimbangan dalam penggunaannya adalah penting. |
| **Judul** | VerXCombo: An interactive data visualization of popular library version combinations |
| **Penulis** | 1. Yuki Yano<br>2. Raula Gaikovina Kula<br>3. Takashi Ishio<br>4. Katsuro Inoue |
| **Metodologi** | Penelitian ini mengembangkan aplikasi, bernama VerXCombo, yang digunakan untuk membantu para pengelola sistem dalam memilih kombinasi versi *library* yang akan digunakan dalam sistemnya. VerXCombo menggunakan HTML5 dan Javascript untuk bagian *front-end* dan Neo4j dan Apache Tomcat untuk *back-end*. Metode yang digunakan dalam aplikasi ini dibagi menjadi beberapa langkah. Pertama, pengguna memilih beberapa kumpulan *library* yang terdapat dalam basis data. *Query* dengan data kumpulan *library* yang telah dipilih diteruskan ke dalam sistem. Kedua, data yang dihasilkan dari *query* tersebut ditampilkan dengan D3.js dalam bentuk kumpulan parallel. Ketiga, dari tampilan yang telah dihasilkan, pengguna dapat mengurutkan atau merubah urutan pada kombinasi *library* yang berbeda sesuai dengan popularitas dan kumpulan *release* terakhir. |
| **Keterkaitan** | Keterkaitan penelitian dengan tugas akhir ini adalah penggunaan infrastruktur *back-end* dan *front-end* yang sama, yaitu D3.js untuk *front-end* dan Neo4j untuk *back-end*. Selain itu, visualisasi yang digunakan untuk penelitian ini adalah *web* dan visualisasi tersebut juga menyediakan |

| | |
|---|---|
| | interaksi yang menjadi pertimbangan untuk ditambahkan dalam tugas akhir ini. |

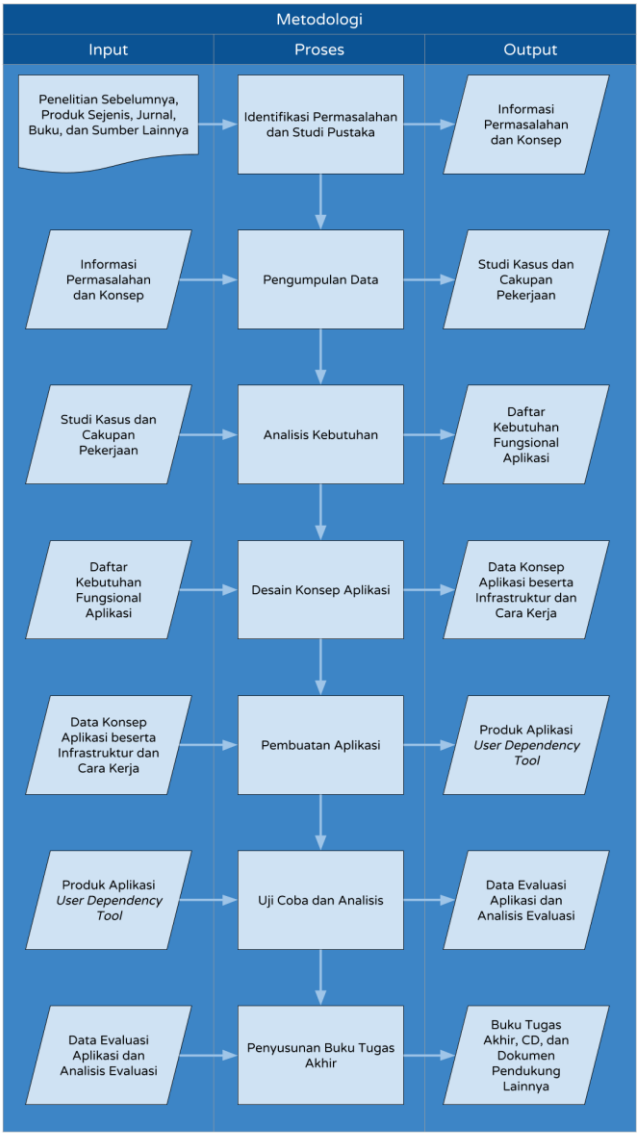
(*Halaman ini sengaja dikosongkan*)

# BAB III
# METODOLOGI PENELITIAN

Pada bagian ini, metodologi dalam mengerjakan tugas akhir ini dijelaskan lebih detail. Metodologi tersebut dijadikan sebagai panduan dasar dalam menyelesaikan tugas akhir ini.

## 3.1 Metode

Proses pengerjaan tugas akhir ini dilakukan melalui beberapa tahapan. Tahapan-tahapan tersebut meliputi Identifikasi Permasalahan dan Studi Pustaka, Pengumpulan Data, Analisis Kebutuhan, Desain Konsep Aplikasi, Pembuatan Aplikasi, Uji Coba dan Analisis, dan Penyusunan Buku Tugas Akhir. Secara sistematis, tahapan – tahapan tersebut dipaparkan dalam Gambar 3.1.




**Gambar 3.1 Tahapan – Tahapan beserta Masukan dan Keluaran Metodologi Pengerjaan Tugas Akhir**

### 3.1.1 Identifikasi Permasalahan dan Studi Pustaka

Pada tahapan ini, penulis menganalisis permasalahan yang berkaitan dengan pengguna RESITS. Dalam analisis yang dilakukan oleh penulis, hasil yang didapatkan dijadikan sebagai bahan dasar pembuatan rumusan masalah dalam tugas akhir ini. Penulis juga menentukan batasan dan tujuan berdasarkan rumusan masalah yang dihasilkan. Batasan tugas akhir berguna untuk membatasi cakupan pekerjaan yang akan dilakukan. Tujuan tugas akhir ini merupakan produk jadi yang dihasilkan dari pengerjaan tugas akhir. Selain itu, penulis juga melakukan studi pustaka terkait dengan penelitian – penelitian sebelumnya dan referensi yang merujuk pada konsep dalam tugas akhir ini. Studi pustaka dilakukan oleh penulis dalam rangka memperkuat konsep dasar dalam pembuatan aplikasi. Penelitian – penelitian sebelumnya juga bermanfaat bagi penulis dalam merancang arsitektur aplikasi. Studi pustaka menggunakan berbagai sumber, seperti internet, jurnal, buku, situs *web*, dan sumber lainnya.

### 3.1.2 Pengumpulan Data

Setelah penulis melakukan identifikasi permasalahan dan studi pustaka, penulis mengumpulkan informasi dan data yang berkaitan dengan tugas akhir ini. Data dan informasi yang dikumpulkan adalah beberapa basis data dengan platform MS SQL Server sebagai studi kasus untuk tugas akhir ini. Basis data yang digunakan dalam tugas akhir ini memiliki kriteria – kriteria tertentu. Kriteria – kriteria tersebut adalah sebagai berikut:

a. Basis data harus memiliki beberapa pengguna (bukan pengguna bawaan).
b. Basis data harus memiliki beberapa objek, seperti tabel, *view*, *stored procedure*, dan objek lainnya.
c. Basis data harus memiliki sampel data.
d. Basis data harus memiliki pengguna yang telah diberikan hak akses atau peran (bukan peran dan/atau hak akses bawaan).

### 3.1.3 Analisis Kebutuhan

Pada tahapan ini, penulis menganalisis kebutuhan yang menjadi dasar fungsional aplikasi. Analisis kebutuhan merujuk pada identifikasi permasalahan pada bagian rumusan masalah, batasan tugas akhir, dan tujuan tugas akhir. Analisis kebutuhan berguna untuk menentukan desain aplikasi yang akan dibuat. Kebutuhan yang didapatkan dari hasil rujukan identifikasi permasalahan pada subbab 1.4 adalah sebagai berikut:

1. Aplikasi dapat melakukan ekstraksi *metadata* objek dan pengguna beserta peran dan hak aksesnya dari MS SQL Server.
2. Aplikasi dapat mengubah hasil ekstraksi pada poin 1 menjadi bentuk *graph* di Neo4j.
3. Aplikasi dapat menampilkan *graph* ke pengguna dengan platform *web*.
4. Aplikasi dapat mengubah data yang berformat JSON ke format visualisasi Neo4jd3.js.
5. Aplikasi dapat menerima masukan dari pengguna untuk menampilkan data – data tertentu yang dipilih berdasarkan preferensi pengguna.

### 3.1.4 Desain Konsep Aplikasi

Dalam tahapan ini, penulis membuat desain konsep aplikasi berdasarkan kebutuhan yang telah dianalisis. Desain konsep aplikasi ini berguna untuk memberikan dasar, tata cara, petunjuk, atau cara kerja dalam membuat aplikasi. Desain konsep aplikasi menentukan konsep infrastruktur, kombinasi *library*, alur aplikasi, dan tatap muka. Dalam tahapan ini, penulis membagi desain konsep aplikasi menjadi dua bagian, yaitu:

1. Desain Infrastruktur

   Dalam bagian ini, desain infrastruktur dalam aplikasi menentukan platform yang digunakan untuk *front-end* dan *back-end*. Platform yang digunakan untuk *front-end* adalah HTML5, CSS3, dan Javascript. Platform yang digunakan untuk *back-end* adalah PHP dan Apache atau Nginx.

2. Desain Kombinasi Library

Dalam bagian ini, desain kombinasi *library* menentukan beberapa *library* yang digunakan dalam membuat aplikasi. *Library* yang dibutuhkan dalam pembuatan aplikasi ini utamanya adalah SQL Server *Driver* untuk PHP, Neo4jd3.js untuk visualisasi *graph*, Neo4j *Driver* untuk PHP, dan *library* lainnya apabila diperlukan untuk kebutuhan yang tidak teridentifikasi.

3. Desain Alur Sistem

Dalam bagian ini, desain alur sistem menentukan alur yang terjadi dalam aplikasi. Desain alur sistem menentukan tiap langkah atau tahapan mulai dari aplikasi berinteraksi dengan pengguna sampai aplikasi mengembalikan atau menampilkan data berdasarkan hasil interaksi pengguna. Desain alur sistem berguna untuk memberikan prioritas terhadap komponen – komponen yang perlu diselesaikan terlebih dahulu.

4. Desain Antarmuka

Dalam bagian ini, desain antarmuka menentukan tampilan yang akan disajikan pada pengguna dalam aplikasi. Desain ini merupakan bagian utama pada komponen *Visualisation* di arsitektur aplikasi. Desain ini berfokus pada tampilan *graph* dalam aplikasi. Desain ini juga menentukan interaksi – interaksi yang dapat dilakukan oleh pengguna dalam komponen *Visualisation*. Berikut ini merupakan contoh konsep desain antarmuka yang akan ditampilkan kepada pengguna dalam Gambar 3.2.


**Gambar 3.2 Contoh Konsep Desain Antarmuka untuk Pengguna**

### 3.1.5 Pembuatan Aplikasi

Dalam tahapan ini, penulis membuat aplikasi berdasarkan desain konsep aplikasi pada tahapan sebelumnya. Pembuatan aplikasi utamanya terbagi menjadi tiga bagian sesuai dengan bagian yang terdapat pada komponen *User Dependency Tool*. Tiga bagian tersebut adalah sebagai berikut:

1. Visualizer

   Dalam bagian ini, pembuatan *Visualizer* menggunakan *library* Neo4jd3.js yang berdasarkan pada *library* D3.js. *Library* Neo4jd3.js menggunakan format masukan berupa JSON. Hal ini memudahkan pengambilan dan pengolahan data dalam platform *web*. Struktur data untuk format masukan berupa JSON pada Neo4jd3.js didefinisikan pada Gambar 3.3.

```
{
    "nodes": [
        {
            "id": "1",
            "labels": ["User"],
            "properties": {
                "userId": "eisman"
            }
        },
        {
            "id": "8",
            "labels": ["Project"],
            "properties": {
                "name": "neo4jd3",
                "title": "neo4jd3.js",
                "description": "Neo4j graph visualization using D3.js.",
                "url": "https://eisman.github.io/neo4jd3"
            }
        }
    ],
    "relationships": [
        {
            "id": "7",
            "type": "DEVELOPES",
            "startNode": "1",
            "endNode": "8",
            "properties": {
                "from": 1470002400000
            },
            "source": "1",
            "target": "8",
            "linknum": 1
        }
    ]
}
```


**Gambar 3.3 Format JSON untuk Masukan D3.js**

2. Processor

Dalam bagian ini, pembuatan *Processor* menggunakan bahasa PHP. Proses untuk mengubah data hasil *query* bagian *Extractor* ke dalam bentuk *graph* dilakukan dengan menggunakan API yang disediakan oleh Neo4j. Berikut pada Kode 3.1 merupakan contoh kode program untuk memasukkan CypherQuery melalui API Neo4j.

```
$client-
>run('CREATE ($db_name:Database {name:"$db_name"
})');
```

**Kode 3.1 Kode untuk Memasukkan CypherQuery Melalui API Neo4j**

3. Extractor

Dalam bagian ini, pembuatan *Extractor* menggunakan bahasa PHP dengan basis data platform MS SQL Server. *Extractor* menggunakan API MS SQL Server (driver) karena perbedaan bahasa dan platform. *Extractor* menggunakan API sebagai jembatan untuk menyalurkan *query* ke dalam basis data MS SQL Server. Contoh *query* yang digunakan untuk mendapatkan data pengguna beserta peran dan hak aksesnya adalah sebagai berikut:

```
SELECT
        [UserType] = CASE membprinc.[type]
                        WHEN 'S' THEN 'SQL User
'
                        WHEN 'U' THEN 'Windows
User'
                        WHEN 'G' THEN 'Windows
Group'
WHEN 'A' THEN 'Application User'
                    END,
        [DatabaseUserName] = membprinc.[name],
        [LoginName]        = ulogin.[name],
        [Role]             = roleprinc.[name],
        [PermissionType]   = perm.[permission_na
me],
        [PermissionState]  = perm.[state_desc],

        [ObjectType] = CASE perm.[class]
                        WHEN 1 THEN obj.[type
_desc]        -- Schema-contained objects
                        ELSE perm.[class_desc
]             -- Higher-level objects
                    END,
        [Schema] = objschem.[name],
        [ObjectName] = CASE perm.[class]
                        WHEN 3 THEN permschem
.[name]       -- Schemas
```

```
20.                             WHEN 4 THEN imp.[name
    ]              -- Impersonations
21.                             ELSE OBJECT_NAME(perm
    .[major_id])  -- General objects
22.                         END,
23.         [ColumnName] = col.[name]
24.     FROM
25.         --Role/member associations
26.         sys.database_role_members          AS me
    mbers
27.         --Roles
28.         JOIN      sys.database_principals  AS ro
    leprinc ON roleprinc.[principal_id] = members.[r
    ole_principal_id]
29.         --Role members (database users)
30.         JOIN      sys.database_principals  AS me
    mbprinc ON membprinc.[principal_id] = members.[m
    ember_principal_id]
31.         --Login accounts
32.         LEFT JOIN sys.server_principals    AS ul
    ogin    ON ulogin.[sid] = membprinc.[sid]
33.         --Permissions
34.         LEFT JOIN sys.database_permissions AS pe
    rm      ON perm.[grantee_principal_id] = rolepri
    nc.[principal_id]
35.         LEFT JOIN sys.schemas              AS pe
    rmschem ON permschem.[schema_id] = perm.[major_i
    d]
36.         LEFT JOIN sys.objects              AS ob
    j       ON obj.[object_id] = perm.[major_id]
37.         LEFT JOIN sys.schemas              AS ob
    jschem  ON objschem.[schema_id] = obj.[schema_id
    ]
38.         --Table columns
39.         LEFT JOIN sys.columns              AS co
    l       ON col.[object_id] = perm.[major_id]
40.
               AND col.[column_id] = perm.[minor_id]

41.         --Impersonations
42.         LEFT JOIN sys.database_principals  AS im
    p       ON imp.[principal_id] = perm.[major_id]

43.     WHERE
```

```
44.         membprinc.[type] IN ('S','U','G','A')
45.         -- No need for these system accounts
46.         AND membprinc.[name] NOT IN ('sys', 'INF
    ORMATION_SCHEMA')
```

**Kode 3.2 Contoh Potongan Kode Query untuk Mendapatkan Data Pengguna, Peran, dan Hak Aksesnya**

### 3.1.6 Uji Coba dan Analisis

Setelah aplikasi telah dibuat, penulis melakukan tahapan pengujian untuk memastikan aplikasi berjalan sesuai dengan kebutuhan fungsional. Dalam hal ini, penulis juga mengkaji hasil pengujian untuk mendapatkan informasi yang mendukung tujuan tugas akhir. Informasi tersebut juga bermanfaat bagi penulis untuk memberikan saran, perbaikan, dan improvisasi pada aplikasi di tugas akhir ini. Pengujian yang dilakukan oleh penulis terbagi berdasarkan bagian – bagian yang terdapat pada aplikasi *User Dependency Tool*, yaitu:

1. Pengujian Visualizer

   Pengujian ini memastikan bagian *Visualizer* yang terdapat pada aplikasi *User Dependency Tool* megubah (format) data sesuai dengan struktur yang telah ditentukan. Pengujian ini berguna untuk mendeteksi *bug* dan eror yang terdapat pada proses pengubahan data dari Neo4j ke dalam format JSON Neo4jd3.js. Pengujian ini juga memastikan grafik *graph* yang dihasilkan tidak memiliki anomali atau kesalahan relasi. Pengujian ini berfokus pada skema JSON yang dihasilkan sehingga skema tersebut dipahami oleh komponen *Visualisation* secara eksplisit.

2. Pengujian *Processor*

   Pengujian bagian *Processor* memastikan bahwa bagian utama dalam aplikasi mengolah data dari komponen lainnya secara valid. Pengujian ini berguna untuk mendeteksi *bug* dan eror yang terdapat dalam proses pengolahan data. Selain itu, pengujian ini memastikan bahwa data yang disalurkan ke komponen lainnya adalah data yang valid dalam isi maupun strukturnya. Pengujian ini

berkontribusi terhadap terpenuhinya tujuan utama tugas akhir.

3. Pengujian *Extractor*
Pengujian ini memastikan bahwa bagian *Extractor* mengambil data dari komponen basis data secara valid. Pengujian ini berguna untuk mendeteksi *bug* dan eror yang terdapat dalam komponen dalam mengambil data atau *query* yang digunakan. Pengujian ini lebih berfokus pada validasi struktur data yang diambil dari basis data dan struktur *query* yang digunakan. Pengujian ini tidak mencakup pada API MS SQL Server yang digunakan dalam menyalurkan *query* dari komponen *User Dependency Tool* ke komponen *Database*.

### 3.1.7 Penyusunan Buku Tugas Akhir

Tahapan akhir dalam metode adalah penyusunan buku tugas akhir. Dalam tahapan ini, penulis mendokumentasikan hasil penelitian dan pembuatan aplikasi ke dalam dokumen. Dokumen tersebut lalu dicetak dalam buku yang berformat A5 dan memiliki *cover*. Dokumen tersebut juga disertai dengan salinan berbentuk *softcopy* dan lampiran pendukung lainnya. Dokumen tugas akhir yang dibuat mulai dari awal pengerjaan sampai akhir diintegrasikan ke *cloud* oleh penulis untuk menghindari risiko kehilangan data dan membuat cadangan data. Dokumen tugas akhir ini dikumpulkan ke pihak administrasi untuk menandakan bahwa pengerjaan tugas akhir telah selesai dilakukan.

*(Halaman ini sengaja dikosongkan)*

# BAB IV
# PERANCANGAN

Pada bab empat ini akan dijelaskan perancangan aplikasi yang menjadikan materi pada subbab 3.1.3 dan 3.1.4 sebagai dasar untuk rujukan. Kedua subbab tersebut adalah Analisis Kebutuhan dan Desain Konsep Aplikasi secara berturut – turut. Berikut ini merupakan subbab – subbab yang diuraikan dari kedua bab tersebut.

## 4.1 Analisis Kebutuhan

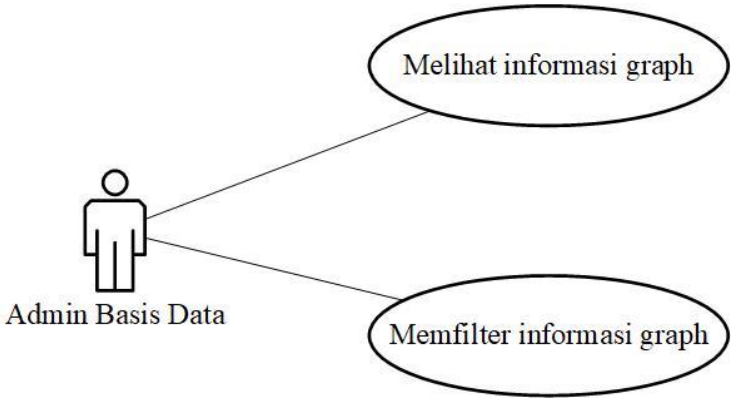
Analisis kebutuhan merupakan proses analisis pada kebutuhan – kebutuhan yang telah dikumpulkan atau digali dari pengguna untuk mendapatkan kebutuhan fungsional aplikasi. Kebutuhan fungsional aplikasi berperan dalam menentukan komponen – komponen dasar dalam aplikasi. Selain itu, kebutuhan fungsional aplikasi digunakan untuk menentukan rancangan terkait dengan arsitektur aplikasi.

### 4.1.1 Fungsi Utama Perangkat Lunak

Fungsi utama aplikasi *User Dependency Tool* adalah menampilkan data pengguna beserta peran dan hak aksesnya terhadap objek basis data dalam bentuk visualisasi dengan model graph sesuai dengan subbab 1.4.

### 4.1.2 Aktivitas Pengguna

Pengguna utama dari aplikasi *User Dependency Tool* adalah admin basis data. Aktivitas pengguna dalam aplikasi *User Dependency Tool* hanya terbatas pada interaksi dengan komponen *Visualisation* pada subbab 4.2.2. Berikut ini merupakan pemetaan aktivitas – aktivitas yang dapat dilakukan oleh pengguna terhadap aplikasi dengan pengguna pada Gambar 4.1.




**Gambar 4.1 Use Case Diagram Aplikasi User Dependency Tool**

Berdasarkan pada Gambar 4.1, berikut ini merupakan penjelasan aktivitas – aktivitas yang dapat dilakukan oleh pengguna utama, yaitu:

1. Melihat informasi *graph*
   Pengguna utama dapat melihat informasi – informasi yang terdapat dalam *graph*. Informasi – informasi tersebut merupakan relasi antar *node*, properti tiap *node*, properti tiap relasi, label tiap *node*, dan tipe tiap relasi. Informasi – informasi tersebut mengandung informasi implisit berupa pemetaan pengguna dengan objek dalam basis data beserta peran dan hak aksesnya dan informasi eksplisit berupa atribut nama, nama julukan, dan atribut lainnya.

2. Memfilter informasi graph
   Pengguna utama dapat memfilter informasi yang ditampilkan dalam *graph*. Hal ini bertujuan agar pengguna dapat membatasi cakupan informasi dan mengekstrak informasi – informasi yang dibutuhkan. Pengguna juga dapat menampilkan semua informasi tanpa memasukkan filter – filter tertentu. Hal tersebut memberikan pengguna pilihan untuk menampilkan informasi *graph* secara general dan keseluruhan. Konstruksi filter terbagi menjadi empat bagian, yaitu bagian *node* asal, relasi, *node* tujuan, dan

jumlah *hop*. Bagian *node* asal menentukan *node* sumber yang harus ditampilkan berdasarkan tipe *node* dan/atau nama julukan. Bagian relasi menentukan relasi – relasi yang harus ditampilkan berdasarkan atribut – atribut yang dipilih. Bagian *node* tujuan menentukan *node* destinasi yang harus ditampilkan berdasarkan tipe *node* dan/atau nama julukan. Bagian jumlah *hop* menentukan lompatan untuk relasi yang didefinisikan antar *node*.

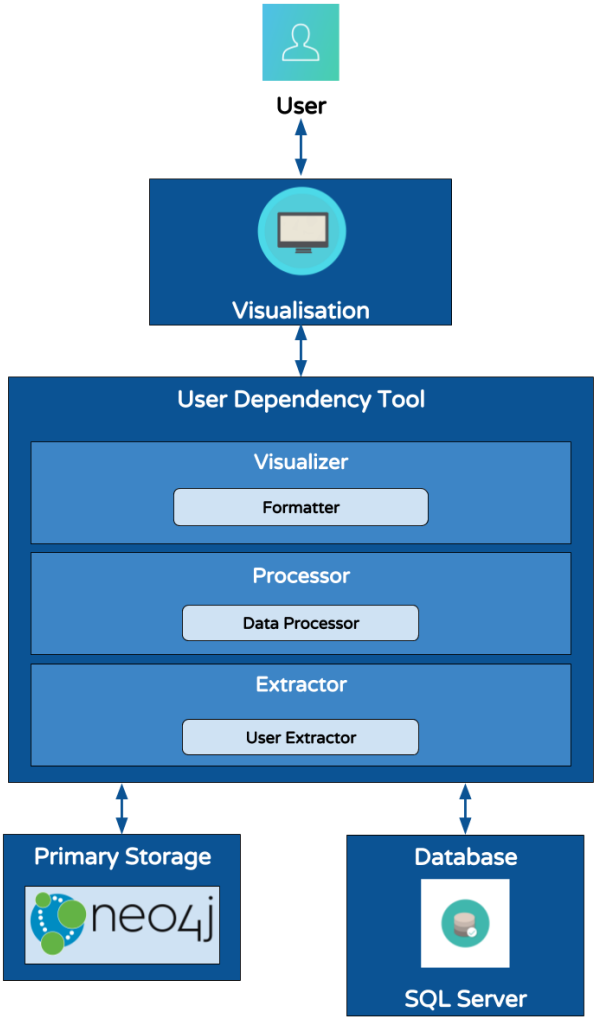
### 4.1.3 Kebutuhan Fungsional

Kebutuhan fungsional dari aplikasi *User Dependency Tool* dalam hal ini didapatkan dari subbab 1.4 dan 4.1.2. Berikut ini merupakan daftar kebutuhan fungsional aplikasi *User Dependency Tool*, yaitu:

1. Melakukan ekstraksi *metadata* objek dan pengguna beserta peran dan hak aksesnya dari MS SQL Server.
2. Mengubah hasil ekstraksi pada poin 1 menjadi bentuk *graph* di Neo4j.
3. Menampilkan *graph* ke pengguna dengan platform *web*.
4. Memfilter visualisasi berdasarkan relasi, group, atau kategori lainnya.

## 4.2 Desain Arsitektur Aplikasi

Pada bagian ini, penulis membuat suatu rancangan arsitektur aplikasi yang merupakan konsep desain dari aplikasi yang akan dibuat. Rancangan arsitektur aplikasi User Dependency Tool ini digunakan untuk melihat gambaran besar dan cara kerja dari aplikasi yang akan dibuat. Rancangan arsitektur aplikasi User Dependency Tool ini memiliki beberapa komponen, baik komponen dalam maupun luar, yaitu *Visualisation*, *User*, *User Dependency Tool, Primary Storage*, dan *Database*.




**Gambar 4.2 Rancangan Arsitektur Aplikasi Tugas Akhir**

### *4.2.1 User*

Komponen *User* adalah komponen yang mewakili pengguna dalam hal memasukkan dan menerima data. Pengguna ini mencakup admin basis data, manajemen, atau pihak lainnya.

Pengguna melakukan interaksi dengan memilih (filter) data – data sesuai dengan kebutuhannya, lalu aplikasi menyediakan tampilan berdasarkan data tersebut.

### *4.2.2 Visualisation*

Komponen *Visualisation* adalah komponen yang memberikan jembatan antar pengguna dengan aplikasi. Komponen *Visualisation* berguna untuk menampilkan visualisasi dan menerima interaksi. Komponen *Visualisation* bekerja baik dengan HTML5, CSS3, dan Javascript karena komponen tersebut menggunakan platform *web*. Komponen *Visualisation* menggunakan *library* Neo4jd3.js untuk menampilkan struktur *graph* yang berisi *node* dan *edge* yang kompleks.

### *4.2.3 User Dependency Tool*

Komponen *User Dependency Tool* merupakan komponen utama dalam aplikasi ini. Komponen ini digunakan untuk melakukan pengolahan terhadap data – data yang terdapat dalam basis data dan menyalurkannya ke komponen *Visualisation*. Komponen ini terbagi menjadi tiga bagian, yaitu *Visualizer, Processor*, dan *Extractor*. Penjelasan terkait dengan ketiga bagian tersebut terdapat pada subbab di bawah ini.

#### *4.2.3.1 Visualizer*

Bagian *Visualizer* ini memiliki komponen utama berupa *Formatter*. Komponen ini digunakan untuk merubah data – data yang dihasilkan dari bagian *Processor* ke dalam bentuk yang mudah dicerna oleh komponen *Visualisation*. Data – data yang mudah dicerna oleh komponen *Visualisation* adalah data yang berformat JSON. Hal tersebut disebabkan oleh komponen *Visualisation* menggunakan Neo4jd3.js sebagai alat untuk menampilkan *graph*, sehingga Javascript cocok digunakan untuk mengubah dan mengolah data dalam bentuk JSON.

#### *4.2.3.2 Processor*

Bagian *Processor* memiliki satu komponen, yaitu komponen *Data Processor*. Komponen *Data Processor* mengambil

seluruh data – data yang didapatkan dari bagian *Extractor*. Komponen tersebut digunakan untuk mengolah data – data terkait dengan seluruh objek dalam basis data, pengguna, hak akses, dan perannya. Selain itu, komponen tersebut juga digunakan untuk mengubah data – data yang telah diolah ke dalam bentuk *graph*. Pembentukan *graph* tersebut menggunakan CypherQuery yang merupakan bawaan dari Neo4j.

#### *4.2.3.3 Extractor*

Bagian *Extractor* digunakan untuk mengekstrak data – data yang terdapat dalam komponen *Database*. Bagian tersebut memiliki satu komponen, yaitu *User Extractor*. Komponen *User Extractor* digunakan untuk mengambil data – data pengguna, hak akses, peran, dan objek untuk membentuk *edge* dan *node* dalam *graph*. Komponen *User Extractor* menggunakan katalog sistem yang telah disediakan oleh MS SQL Server. Komponen ini menggunakan T-SQL *query* untuk mendapatkan data – data tersebut. *Query* yang dieksekusi oleh komponen ini diiterasi untuk tiap basis data yang terdapat dalam satu *instance*.

### *4.2.4 Primary Storage*

Komponen *Primary Storage* adalah komponen utama dalam penyimpanan data dalam struktur *graph*. Komponen ini berisi *node* dan *edge* yang merupakan hasil dari bagian *Processor* dalam pengolahan data. Komponen ini bertindak sebagai jembatan penyimpanan antara komponen *Database* dengan komponen *Visualisation*. Data – data yang didapatkan dari komponen *Database* perlu diolah terlebih dahulu sebelum masuk ke dalam komponen *Visualisasi*. Hasil pengolahan ini tidak efektif apabila disimpan dalam memori, karena memori dalam alat komputasi memiliki kapasitas terbatas dan penggunaannya tidak hanya satu aplikasi saja. Hal tersebut menyebabkan aplikasi membutuhkan alat dalam menyimpan hasil pengolahan dan menyajikannya dengan efektif dan efiisien. Alat tersebut adalah Neo4j Graph DBMS, karena alat

ini menyediakan cara dalam menyimpan dan menyajikan data dalam bentuk *graph*.

### *4.2.5 Database*

Komponen *Database* adalah komponen utama sebagai sumber dalam memberikan data untuk pengolahan. Komponen ini merupakan komponen yang berisi data – data natural, terkait dengan objek dalam basis data, pengguna, hak akses, peran, dan data – data lainnya. Komponen ini menyajikan data – data dalam bentuk mentah, sehingga data – data yang terdapat dalam komponen ini perlu diekstrak dengan menggunakan bagian *Extractor* dalam komponen *User Dependency Tool*. Komponen ini menggunakan platform yang berbeda dengan bagian *Extractor*, sehingga bagian *Extractor* perlu menggunakan API yang telah disediakan oleh platform pada komponen ini. Komponen ini menggunakan platform MS SQL Server.

## 4.3 Desain Infrastruktur

Desain infrastruktur aplikasi *User Dependency Tool* terbagi menjadi dua bagian utama, yaitu *front-end* dan *back-end*. Bagian *front-end* menggunakan HTML5, CSS3, Javascript, dan JQuery. Bagian *back-end* menggunakan PHP dan Nginx. Kedua bagian tersebut mengacu bahasan pada subbab 3.1.4. Perbedaan dalam subbab ini dengan subbab tersebut adalah bagian *back-end*. Pertimbangan dalam penggunaan Nginx daripada Apache adalah kemudahan dalam pengaturan *routing* untuk akses berbasis protokol HTTP.

## 4.4 Desain Kombinasi *Library*

Desain kombinasi *library* aplikasi *User Dependency Tool* tetap mengacu bahasan pada subbab 3.1.4. *Library* utama yang digunakan dalam aplikasi adalah Sql Server *extension* (sqlsrv dan pdo_sqlsrv) untuk PHP, esiman/neo4jd3 untuk visualisasi graph, graphaware/neo4j-php-client untuk interaksi PHP dengan Neo4j. *Library* tambahan lainnya yang digunakan untuk aplikasi ini adalah symfony/var-dumper sebagai pemercantik untuk *dump* variabel, Material Kit sebagai *framework* untuk

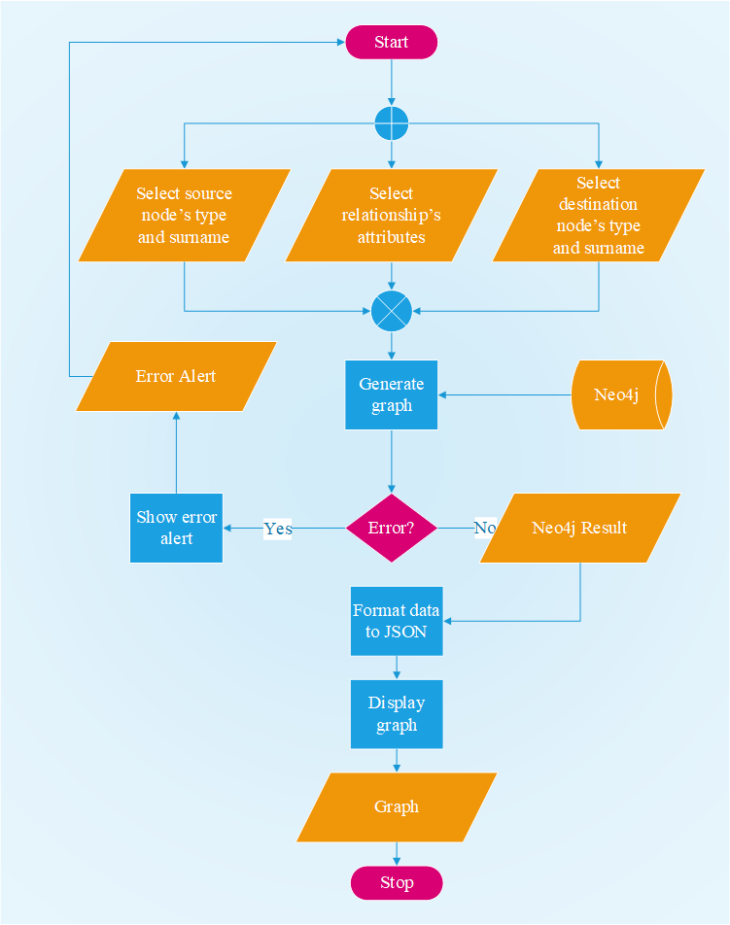
tampilan aplikasi, sweetalert2 untuk menampilkan kotak informasi yang responsif dan mudah, dan bootstrap-select untuk menambahkan beberapa fungsionalitas terhadap elemen *select* dalam *form*.

## 4.5 Desain Alur Sistem

Desain alur sistem dalam aplikasi *User Dependency Tool* dibagi berdasarkan bagian – bagian pada subbab 4.2.3. Tiga bagian tersebut merupakan *Visualizer*, *Processor*, dan *Extractor*. Pertimbangan pembagian alur sistem terhadap tiga bagian tersebut terletak pada peran penting komponen *User Dependency Tool* sebagai komponen utama dalam aplikasi. Penjelasan alur terkait dengan masing – masing bagian terletak pada subbab – subbab di bawah ini.

### 4.5.1 Alur Bagian *Visualizer*

Bagian *Visualizer* berguna untuk mengubah data atau masukan yang didapatkan pada bagian *Processor* dari basis data Neo4j ke dalam format JSON. Alur pada bagian *Visualizer* dittunjukkan pada Gambar 4.3 berikut ini.




**Gambar 4.3 Flowchart Alur Bagian Visualizer**

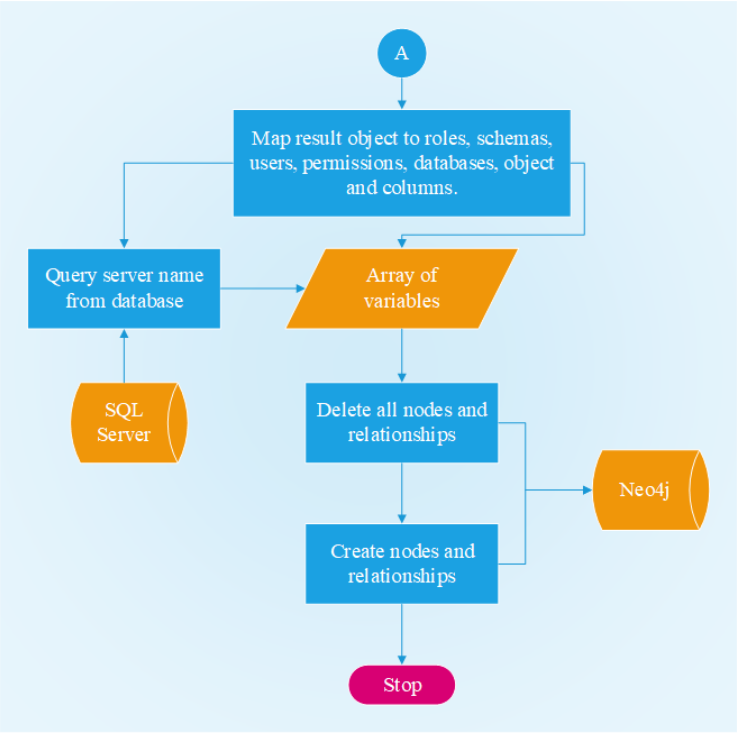
Penjelasan terkait dengan *flowchart* alur bagian *Visualizer* pada Gambar 4.3 adalah sebagai berikut:

1. Alur pada bagian *Visualizer* dimulai pada saat pengguna memilih filter pada aplikasi dan pemilihan tersebut bersifat opsional untuk *node* asal, *node* tujuan, relasi, dan/atau jumlah *hop*. Pemilihan filter dilakukan pada komponen *Visualisation*.

2. Setelah pengguna memilih filter sesuai dengan preferensinya, maka pengguna menekan tombol "Submit" untuk melakukan proses *generate graph* dengan CypherQuery pada basis data Neo4j.
3. Jika aplikasi mengalami eror, setelah proses *generate graph*, maka aplikasi akan menampilkan eror dalam bentuk peringatan terhadap pengguna. Jika aplikasi tidak mengalami eror, setelah proses *generate graph*, maka aplikasi akan menghasilkan data berupa *Neo4j Result*. Data tersebut merupakan hasil yang dibundel dalam objek PHP dari *library* graphaware/neo4j-php-client. Peringatan eror akan ditampilkan pada komponen *Visualisation*.
4. Setelah itu, aplikasi mengekstrak data dari *Neo4j Result* dan memetakannya ke dalam format JSON.
5. Kemudian, data yang berformat JSON disalurkan ke dalam komponen *Visualisation* pada subbab 4.2.2 untuk membentuk tampilan berupa *graph* di bagian *front-end*.

### 4.5.2 Alur Bagian *Processor*

Bagian *Processor* merupakan bagian yang berperan dalam mengubah data yang didapatkan dari bagian *Extractor* menjadi data dalam bentuk *node* dan relasi antar *node* untuk membentuk suatu *graph* dan menyimpannya ke dalam Neo4j dengan CypherQuery. Alur pada bagian *Processor* ditunjukkan pada Gambar 4.4 di bawah ini.




**Gambar 4.4 Flowchart Alur Bagian Processor**

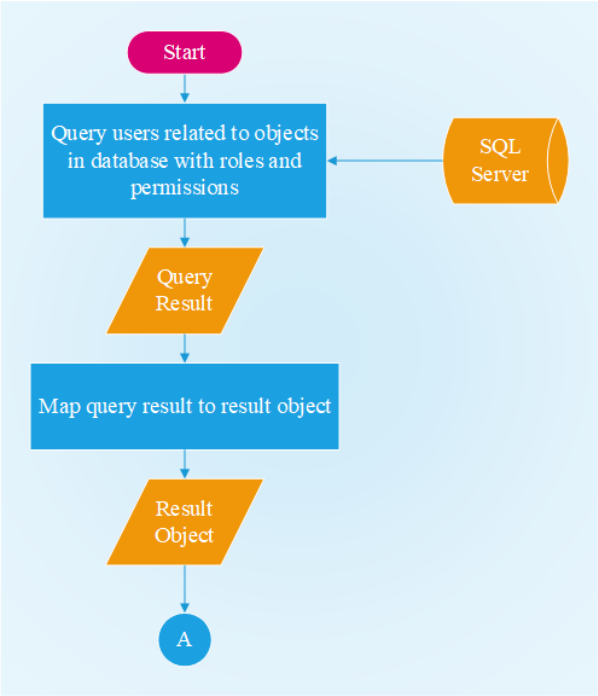
Penjelasan *flowchart* alur bagian *Processor* pada Gambar 4.4 dijabarkan sebagai berikut:

1. Alur *flowchart* dimulai dari *flowchart* pada bagian *Extractor* pada saat masukan terakhir yang diterima berupa *Result Object*.
2. Masukan data berupa *Result Object* dipetakan ke dalam masing – masing variabel, berupa *roles*, *schemas*, *users*, *permissions*, *databases*, *objects*, and *columns*.
3. Kemudian, aplikasi menggunakan T-SQL *query* untuk mendapatkan nama *server* dari basis data SQL Server. Nama server yang telah didapatkan dari hasil *query* juga dipetakan ke dalam variabel.

4. Variabel yang telah dipetakan tersebut dibentuk beberapa *array* yang mengandung metadata sebagai atribut dari *node* dan relasi.
5. Lalu, aplikasi menggunakan CypherQuery untuk menghapus keseluruhan *nodes* dan relasi yang terdapat dalam *graph*. Hal ini bertujuan untuk menghilangkan inkonsistensi pada *graph*, apabila *nodes* dan relasi baru dimasukkan ke dalam basis data.
6. Kemudian, aplikasi memasukkan *nodes* dan relasi yang baru dengan data dari *array* – *array* metadata yang telah terbentuk. Proses memasukkan *nodes* dan relasi yang baru juga menggunakan CypherQuery untuk menyimpan *nodes* dan relasi tersebut dalam bentuk *graph*.

### 4.5.3 Alur Bagian *Extractor*

Bagian *Extractor* memiliki peran untuk mengekstrak objek, pengguna, hak akses, peran, skema, *server*, dan data lainnya dari basis data Sql Server dengan menggunakan T-SQL Query. Alur pada bagian *Extractor* ditunjukkan pada Gambar 4.5 di bawah ini.



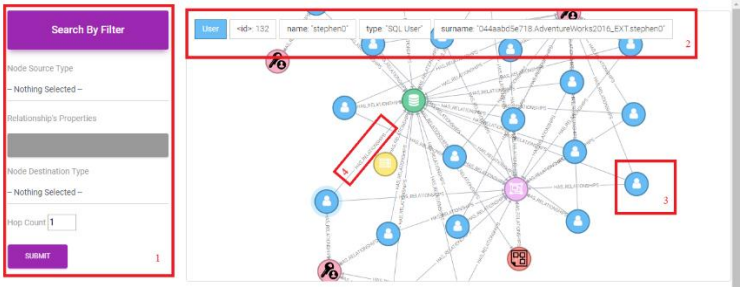

**Gambar 4.5 Flowchart Alur Bagian Extractor**

Penjelasan *flowchart* alur bagian *Extractor* pada Gambar 4.5 dijabarkan sebagai berikut:

1. Aplikasi bermula melakukan *query* ke SQL Server dengan *query* khusus untuk ekstraksi pemetaan pengguna beserta peran dan hak aksesnya terhadap objek dalam basis data. Hasil *query* yang didapatkan disimpan ke dalam *Query Result* yang merupakan objek bawaan untuk *Library* PDO dari PHP.
2. Lalu, aplikasi memetakan *Query Result* ke dalam objek PHP berupa *Result Object*. *Result Object* merupakan objek buatan khusus untuk menampung informasi – informasi terkait nama basis data, *query* yang digunakan, koneksi, dan hasil dari eksekusi *query*.

3. Kemudian, alur aplikasi berlanjut dari bagian *Extractor* ke bagian *Processor*. Data *Result Object* langsung disalurkan ke bagian *Processor* sebagai bahan utama dalam pembuatan *nodes* dan relasi untuk pembentukan *graph*.

## 4.6 Desain Antarmuka




**Gambar 4.6 Desain Antarmuka Aplikasi User Dependency Tool**

Desain antarmuka aplikasi *User Dependency Tool* terdapat pada Gambar 4.6. Desain antarmuka menjelaskan tampilan yang akan ditampilkan pada pengguna. Desain antarmuka ini mencakup interaksi yang dapat dilakukan oleh pengguna, sekaligus keluaran atau informasi yang diterima oleh pengguna. Informasi utama yang disajikan oleh aplikasi ini berupa tampilan *graph* yang berisi pemetaan pengguna dengan peran dan hak aksesnya terhadap objek basis data.

Penjabaran terkait dengan bagian – bagian yang berkotak merah dalam Gambar 4.6 untuk desain antarmuka aplikasi adalah sebagai berikut:

1. Bagian 1: Bagian ini merupakan bagian utama aplikasi *User Dependency Tool* dalam menerima masukan dari pengguna. Masukan tersebut berupa filter untuk informasi *graph* yang akan ditampilkan. Filter ini terdiri dari empat bagian, yaitu bagian *node* asal, *node* tujuan, relasi, dan jumlah *hop*. Penjelasan terkait

dengan bagian – bagian tersebut terdapat pada subbab 4.1.2.

2. Bagian 2: Bagian ini merupakan bagian yang mengandung deskripsi dari suatu *node* atau relasi. Data yang ditampilkan untuk *node* berupa label dan propertinya, sedangkan data yang ditampilkan untuk relasi berupa tipe dan propertinya. Data – data tersebut ditampilkan apabila *cursor mouse* diarahkan ke atas *node* atau relasi.
3. Bagian 3: Bagian ini menunjukkan tampilan *node* dalam *graph*. *Node* merepresentasikan *user*, *schema*, *role*, *database*, *object*, *server*, dan *column*.
4. Bagian 4: Bagian ini menunjukkan tampilan relasi dalam *graph*. Relasi tersebut menampilkan pemetaan antar *user*, *schema*, *role*, *database*, *object*, *server*, dan *column*.

## 4.7 Desain *Database*

Desain *database* aplikasi *User Dependency Tool* menjelaskan konsep *graph* yang digunakan. Konsep *graph* tersebut terbagi menjadi dua bagian, yaitu bagian *node* dan relasi. Penjabaran terkait dengan kedua bagian tersebut dalam konsep *graph* terdapat pada subbab – subbab berikut ini.

### 4.7.1 Bagian *Node*

Bagian ini menjabarkan *node* dalam *graph* di basis data Neo4j. Dalam aplikasi ini, *node* merepresentasikan *user*, *object*, *schema*, *server*, *column*, *role*, dan *database* yang terdapat pada basis data SQL Server. *Node* memiliki label sebagai tipe dari *node* tersebut. Keseluruhan label yang digunakan dalam aplikasi ini terdapat pada Tabel 4.1.

**Tabel 4.1 Keseluruhan Label dalam Aplikasi User Dependency Tool**

| Label | Keterangan |
|---|---|
| User | Label ini merujuk pada pengguna dalam basis data SQL Server. Pengguna yang dirujuk merupakan pengguna dengan tingkatan basis data. |
| Object | Label ini merujuk pada objek – objek yang terdapat dalam basis data SQL Server. Objek ini meliputi tabel, *stored procedure*, *view*, dan objek lainnya. |
| Database | Label ini merujuk pada basis data yang terdapat pada SQL Server. Satu *instance* SQL Server dapat memiliki lebih dari satu basis data dan satu basis data memiliki lebih dari satu skema. |
| Schema | Label ini merujuk pada skema yang terdapat dalam basis data SQL Server. Satu skema memiliki beberapa objek. |
| Role | Label ini merujuk pada peran yang dimiliki oleh pengguna. Peran ini merupakan kumpulan hak akses dan metode pengelompokan atau pengkategorian pengguna. |
| Server | Label ini merujuk pada *server* yang merupakan tempat berjalannya basis data. *Server* juga dapat merujuk pada istilah *instance* di SQL Server. |
| Column | Label ini merujuk pada kolom yang terdapat pada objek basis data tertentu. Objek basis data tersebut memiliki karakteristik berupa tabel, seperti tabel dan *view*. |

Selain label, *node* juga memiliki properti. Properti digunakan untuk memberikan deskripsi atau keterangan tambahan terkait dengan *node*. Seluruh *node* yang terdapat dalam aplikasi ini

memiliki beberapa properti yang sama. Berikut ini merupakan penjabaran terkait dengan properti umum untuk seluruh *node* pada Tabel 4.2.

**Tabel 4.2 Properti Umum dari Node dalam Aplikasi User Dependency Tool**

| Properti | Keterangan |
|---|---|
| name | Properti ini merujuk pada nama dari suatu *node*. Properti tersebut diambil dari nama yang terdapat dalam basis data SQL Server. |
| surname | Properti ini merujuk pada nama julukan dari suatu *node*. Properti tersebut diambil dari nama lengkap yang terdapat dalam basis data SQL Server. |

Selain properti umum, beberapa *node* juga memiliki properti khusus. Properti khusus hanya dimiliki oleh *node* yang berlabel *User* dan *Object*. Properti tersebut adalah properti type. Properti ini merujuk pada kategori dari *node*, hanya khusus berlabel *User* dan *Object*. Jika properti ini terdapat pada *node* dengan label *User*, maka nilai properti ini diambil dari tipe pengguna dalam basis data SQL Server, begitu juga untuk kasus *node* yang berlabel *Object*.

### 4.7.2 Bagian Relasi

Bagian ini menjabarkan relasi antar *node* dalam *graph* di basis data Neo4j. Dalam aplikasi ini, relasi merepresentasikan hubungan pemetaan antar pengguna beserta peran dan hak aksesnya dengan objek basis data dalam SQL Server. Apabila *node* memiliki label sebagai tipe dari *node* tersebut, maka relasi juga memiliki tipe. Tipe relasi yang digunakan dalam aplikasi ini hanya satu, yaitu HAS_RELATIONSHIPS. Tipe tersebut merupakan tipe relasi yang merepresentasikan seluruh hubungan atau pemetaan yang terdapat antar dua *node*. Seluruh relasi dengan tipe tersebut dalam aplikasi ini memiliki properti

yang bervariasi, sehingga tiap relasi antar dua *node* memiliki hubungan yang berbeda – beda. Berikut ini merupakan penjabaran terkait dengan properti – properti yang dapat diidentifikasikan dan dimiliki oleh relasi dengan tipe HAS_RELATIONSHIPS pada Tabel 4.3.

**Tabel 4.3 Beberapa Properti dari Relasi Bertipe HAS_RELATIONSHIPS**

| Properti | Keterangan |
|---|---|
| DATABASE | Properti ini merujuk pada hubungan antar *node* yang berlabel *Server* dengan *node* yang berlabel *Database*. Properti ini memiliki makna bahwa *server* memiliki basis data. |
| USER_OF | Properti ini merujuk pada hubungan antar *node* yang berlabel *User* dengan *node* yang berlabel *Database*. Properti ini memiliki makna bahwa *user* merupakan pengguna basis data. |
| MEMBER_OF | Properti ini merujuk pada hubungan antar *node* yang berlabel *User* dengan *node* yang berlabel *Role*. Properti ini memiliki makna bahwa pengguna merupakan bagian atau anggota dari peran tertentu. |
| SCHEMA | Properti ini merujuk pada hubungan antar *node* yang berlabel *Database* dengan *node* yang berlabel *Schema*. Properti ini memiliki makna bahwa basis data memiliki skema. |
| OWNS | Properti ini merujuk pada hubungan antar *node* yang berlabel *Schema* dengan *node* yang berlabel *Object*. Properti ini memiliki makna bahwa skema memiliki beberapa objek. |

| | |
|---|---|
| COLUMN | Properti ini merujuk pada hubungan antar *node* yang berlabel *Object* dengan *node* yang berlabel *Column*. Properti ini memiliki makna bahwa objek mengandung kolom. |

## 4.8 Desain Pengujian Aplikasi

Desain pengujian aplikasi menjabarkan metode yang digunakan untuk menguji aplikasi *User Dependency Tool*. Metode yang digunakan dalam pengujian ini adalah metode *black box testing*. Pengujian aplikasi dibagi menjadi dua bagian, yaitu bagian performa dan bagian fungsional. Bagian performa lebih menekankan pada kinerja aplikasi, sedangkan bagian fungsional lebih cenderung mengarah pada kegunaan atau fungsionalitas aplikasi. Bahan pengujian untuk aplikasi *User Dependency Tool* merupakan komponen *User Dependency Tool* dan komponen *Visualisation* yang terdapat pada subbab 4.2.

Pengujian fungsionalitas aplikasi mencakup komponen *User Dependency Tool* untuk seluruh bagian, yaitu bagian *Visualizer*, *Processor*, dan *Extractor* dan komponen *Visualisation*. Berikut ini merupakan penjelasan terkait dengan pengujian fungsionalitas kedua komponen tersebut pada Tabel 4.4.

**Tabel 4.4 Pengujian Fungsionalitas Aplikasi User Dependency Tool**

| **Bagian /Komponen** | **ID** | **Cara Pengujian** | **Hasil yang Diharapkan** |
|---|---|---|---|
| Visualisation | A01 | Menampilkan *form* filter untuk interaksi pengguna. | Aplikasi menampilkan *form* filter. |
| | A02 | Memfilter informasi yang | Aplikasi hanya menampilkan |

| | | | |
|---|---|---|---|
| | | ditampilkan dalam *graph*. | sebagian dari *graph*. |
| | A03 | Menampilkan *graph* dari data yang berformat JSON. | Aplikasi menampilkan *graph* sesuai dengan data yang disalurkan. |
| | A04 | Menampilkan pesan eror saat terjadi eror dalam proses filter. | Aplikasi menampilkan pesan eror. |
| Visualizer | B01 | Mengubah data *graph* Neo4j ke format JSON. | Aplikasi mendapatkan data yang berformat JSON untuk *node* dan relasinya. |
| Processor | C01 | Menghapus seluruh *node* dan relasi dalam graph dengan CypherQuery. | Aplikasi menghapus *graph*, termasuk seluruh *node* dan relasi. |
| | C02 | Mengirim T-SQL *query* untuk mendapatkan nama *server*. | Aplikasi mendapatkan hasil *query* nama *server*. |
| | C03 | Membuat *node* dan relasinya dengan menggunakan data pada *result object* yang didapatkan dari | Aplikasi membuat *graph* baru pada basis data Neo4j dengan data *node* dan relasi dari *result object*. |

| | | | |
|---|---|---|---|
| | | D02 dan CypherQuery. | |
| | C04 | Menambahkan indeks pada seluruh label *node* untuk properti name dan surname dengan CypherQuery. | Aplikasi menambahkan indeks pada *graph* di basis data Neo4j. |
| Extractor | D01 | Mengirim T-SQL *query* untuk mendapatkan pemetaan pengguna beserta peran dan hak aksesnya dengan objek basis data. | Aplikasi mendapatkan hasil *query*, berupa pemetaan pengguna beserta peran dan hak aksesnya dengan objek basis data. |
| | D02 | Memetakan hasil *query* dari D01 ke dalam *result object*. | Aplikasi mendapatkan *result object* yang berisi data – data, berupa basis data, koneksi, *query*, dan data lainnya. |

Pengujian performa atau kinerja aplikasi pada komponen *User Dependency Tool* mencakup dua bagian saja, yaitu bagian *Processor* dan *Extractor*. Pengujian tersebut dimulai dari awal program sampai berakhirnya program (Awal Kode sampai Akhir Kode). Pengujian performa menggunakan *microtime* yang merupakan bawaan dari PHP. Hasil akhir dari pengujian performa merupakan rata – rata dari percobaan eksekusi sebanyak enam kali. Rata – rata bagian *Processor* dikurangkan dengan rata – rata bagian *Extractor* karena bagian *Extractor*

merupakan bagian vital yang memiliki data masukan untuk bagian *Processor*.

# BAB V
# IMPLEMENTASI

Bab ini menjelaskan implementasi terhadap perancangan yang telah dijabarkan pada bab sebelumnya. Implementasi pengembangan aplikasi *User Dependency Tool* mencakup lingkungan implementasi, konfigurasi aplikasi, konfigurasi inisiasi koneksi aplikasi, implementasi aplikasi, dan konfigurasi otomatisasi.

## 5.1 Lingkungan Implementasi

Lingkungan implementasi mendefinisikan lingkungan yang digunakan untuk mengembangkan dan menguji perangkat lunak. Lingkungan implementasi mencakup infrastruktur, perangkat keras, perangkat lunak, jaringan, dan/atau *library*. Dalam konteks tugas akhir ini, lingkungan implementasi merupakan sumber daya yang disediakan oleh penulis. Berikut ini merupakan spesifikasi perangkat keras yang dijabarkan pada Tabel 5.1.

**Tabel 5.1 Spesifikasi Perangkat Keras**

| Data Perangkat | Spesifikasi |
|---|---|
| Processor | Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz, 3504 Mhz |
| Memory | 16.0 GB |
| Operating System | Microsoft Windows 10 Pro |

Pengembangan aplikasi ini membutuhkan beberapa teknologi pendukung. Teknologi pendukung ini digunakan untuk memudahkan dan mempercepat pengembangan aplikasi. Teknologi pendukung dapat mencakup perangkat lunak, *library*, dan perangkat pihak ketiga. Berikut ini merupakan

daftar teknologi pendukung yang digunakan dalam pengembangan aplikasi ini pada Tabel 5.2.

**Tabel 5.2 Daftar Teknologi Pendukung**

| | |
|---|---|
| **Webserver** | Nginx 1.14.0 |
| **Language** | • PHP<br>• HTML5<br>• CSS3<br>• Javascript |
| **Database** | • Neo4j 3.4.1 Enterprise<br>• SQL Server 2017 |
| **Text Editor** | Visual Studio Code 1.24.0 |
| **Browser** | Google Chrome Version 67.0.3396.87 |
| **Library** | • Font Awesome 4.7.0<br>• Material Kit v2.0.3<br>• Bootstrap-Select v1.12.4<br>• eisman/neo4jd3<br>• graphaware/neo4j-php-client<br>• symfony/var-dumper<br>• sweetalert2/sweetalert2<br>• sqlsrv dan pdo_sqlsrv 5.2.0 |
| **OS Component** | • Windows Task Scheduler<br>• Command Prompt (cmd.exe) |
| **Virtualization** | Docker |

Secara bawaan, *webserver* dengan nginx menjalankan *web* pada *port* 80. Hal tersebut menyebabkan gangguan pada lingkungan

lokal penulis. Solusi untuk mengatasi permasalahan tersebut adalah mengganti *port* 80 ke *port* 81 dengan mengubah beberapa konfigurasi pada *webserver*. Teknologi virtualisasi dalam lingkungan implementasi digunakan untuk menyimulasikan basis data Neo4j dan SQL Server tanpa melakukan instalasi terlebih dahulu pada lingkungan lokal.

## 5.2 Konfigurasi Aplikasi

Konfigurasi aplikasi menjabarkan hal – hal terkait dengan pengaturan dasar aplikasi untuk menjalankan beberapa fungsi atau bagian tertentu. Konfigurasi aplikasi terletak pada folder **config** di *directory* utama folder aplikasi. Folder tersebut berisi file – file PHP yang mengembalikan nilai *array* yang berisi *key* dan *value* terkait dengan pengaturan koneksi basis data, *query*, dan pengaturan lainnya jika dibutuhkan. Dalam aplikasi ini, pengaturan wajib yang digunakan dan dikemas dalam *file* PHP berjumlah dua, yaitu pengaturan *query* dan pengaturan koneksi. Pengaturan koneksi yang digunakan sebagai basis dalam inisiasi koneksi ke basis data terdapat pada Kode 5.1.

```
1.  <?php
2.
3.  return [
4.      'connections' => [
5.          'sqlsrv' => [
6.              'host' => '10.199.2.66',
7.              'port' => 1433,
8.              'database' =>   'resits',
9.              'username' => 'monitoring',
10.             'password' => 'monitor',
11.             'charset' => 'utf8',
12.             'prefix' => 'sqlsrv',
13.         ],
14.         'neo4j' => [
15.             'sp' => [
16.                 'host' => 'localhost',
17.                 'port' => 7687,
18.                 'username' => 'neo4j',
19.                 'password' => 'secret',
20.             ],
21.             'user' => [
```

```
                'host' => '192.168.99.100',
                'port' => 7687,
                'username' => 'neo4j',
                'password' => 'neo4j123',
            ],
            'username_read' => 'neo4j_read',
            'password_read' => 'neo4jread123',
        ]
    ]
];
```

**Kode 5.1 Konfigurasi Koneksi Basis Data**

Pengaturan koneksi pada Kode 5.1 mengandung nilai – nilai, berupa *server*/*host*, *port*, *username*, dan *password*. Nilai – nilai tersebut merupakan nilai yang umum digunakan untuk menginisiasi koneksi ke dalam basis data. Selain pengaturan koneksi, pengaturan *query* yang terdapat dalam aplikasi ini ditampilkan pada Kode 5.2 berikut ini.

```
<?php

return [
    'sqlserver' => [
        'extract_user' => "
            DECLARE @Command nvarchar(MAX)
            SET @Command = '
            SELECT
                [userType] = CASE princ.[type]
                                WHEN ''S'' THEN ''SQL User''
                                WHEN ''U'' THEN ''Windows User''
                                WHEN ''G'' THEN ''Windows Group''
                                WHEN ''A'' THEN ''Application User''
                             END,
                [databaseUserName] = princ.[name],
                [loginName]        = ulogin.[name],
                [role]             = NULL,
```

```
18.                 [permissionType]   = perm.[permi
    ssion_name],
19.                 [permissionState]  = perm.[state
    _desc],
20.                 [objectType] = CASE perm.[class]

21.                                    WHEN 1 THEN obj.
    [type_desc]
22.                                    ELSE perm.[class
    _desc]
23.                                END,
24.                 [schema] = objschem.[name],
25.                 [objectName] = CASE perm.[class]

26.                                    WHEN 3 THEN perm
    schem.[name]
27.                                    WHEN 4 THEN imp.
    [name]
28.                                    ELSE OBJECT_NAME
    (perm.[major_id])
29.                                END,
30.                 [columnName] = col.[name]
31.             FROM
32.                 sys.database_principals
       AS princ
33.                 LEFT JOIN sys.server_principals
       AS ulogin    ON ulogin.[sid] = princ.[sid]
34.                 LEFT JOIN sys.database_permissio
    ns AS perm      ON perm.[grantee_principal_id] =
     princ.[principal_id]
35.                 LEFT JOIN sys.schemas
       AS permschem ON permschem.[schema_id] = perm.
    [major_id]
36.                 LEFT JOIN sys.objects
       AS obj       ON obj.[object_id] = perm.[major
    _id]
37.                 LEFT JOIN sys.schemas
       AS objschem  ON objschem.[schema_id] = obj.[s
    chema_id]
38.                 LEFT JOIN sys.columns
       AS col       ON col.[object_id] = perm.[major
    _id]
```

```
39.
                    AND col.[column_id] = perm.[mino
    r_id]
40.                 LEFT JOIN sys.database_principal
    s  AS imp       ON imp.[principal_id] = perm.[ma
    jor_id]
41.             WHERE
42.                 princ.[type] IN (''S'',''U'',''G
    '',''A'')
43.                 AND princ.[name] NOT IN (''sys''
    , ''INFORMATION_SCHEMA'')
44.             UNION
45.             SELECT
46.                 [userType] = CASE membprinc.[typ
    e]
47.                                 WHEN ''S'' THEN
    ''SQL User''
48.                                 WHEN ''U'' THEN
    ''Windows User''
49.                                 WHEN ''G'' THEN
    ''Windows Group''
50.                                 WHEN ''A'' THEN
    ''Application User''
51.                             END,
52.                 [databaseUserName] = membprinc.[
    name],
53.                 [loginName]        = ulogin.[nam
    e],
54.                 [role]             = roleprinc.[
    name],
55.                 [permissionType]   = perm.[permi
    ssion_name],
56.                 [permissionState]  = perm.[state
    _desc],
57.                 [objectType] = CASE perm.[class]

58.                                 WHEN 1 THEN obj.
    [type_desc]
59.                                 ELSE perm.[class
    _desc]
60.                             END,
61.                 [schema] = objschem.[name],
62.                 [objectName] = CASE perm.[class]
```

```
63.                                     WHEN 3 THEN perm
   schem.[name]
64.                                     WHEN 4 THEN imp.
   [name]
65.                                     ELSE OBJECT_NAME
   (perm.[major_id])
66.                                END,
67.                 [columnName] = col.[name]
68.             FROM
69.                 sys.database_role_members
      AS members
70.                 JOIN      sys.database_principal
   s  AS roleprinc ON roleprinc.[principal_id] = me
   mbers.[role_principal_id]
71.                 JOIN      sys.database_principal
   s  AS membprinc ON membprinc.[principal_id] = me
   mbers.[member_principal_id]
72.                 LEFT JOIN sys.server_principals
      AS ulogin    ON ulogin.[sid] = membprinc.[sid
   ]
73.                 LEFT JOIN sys.database_permissio
   ns AS perm      ON perm.[grantee_principal_id] =
    roleprinc.[principal_id]
74.                 LEFT JOIN sys.schemas
      AS permschem ON permschem.[schema_id] = perm.
   [major_id]
75.                 LEFT JOIN sys.objects
      AS obj       ON obj.[object_id] = perm.[major
   _id]
76.                 LEFT JOIN sys.schemas
      AS objschem  ON objschem.[schema_id] = obj.[s
   chema_id]
77.                 LEFT JOIN sys.columns
      AS col       ON col.[object_id] = perm.[major
   _id]
78.
                    AND col.[column_id] = perm.[mino
   r_id]
79.                 LEFT JOIN sys.database_principal
   s  AS imp       ON imp.[principal_id] = perm.[ma
   jor_id]
80.             WHERE
81.                 membprinc.[type] IN (''S'',''U''
   ,''G'',''A'')
```

```
82.                 AND membprinc.[name] NOT IN (''s
    ys'', ''INFORMATION_SCHEMA'')
83.             UNION
84.             SELECT
85.                 [userType]         = ''public'',

86.                 [databaseUserName] = ''public'',

87.                 [loginName]        = ''public'',

88.                 [role]             = roleprinc.[
    name],
89.                 [permissionType]   = perm.[permi
    ssion_name],
90.                 [permissionState]  = perm.[state
    _desc],
91.                 [objectType] = CASE perm.[class]

92.                                    WHEN 1 THEN obj.
    [type_desc]        -- schema-contained objects
93.                                    ELSE perm.[class
    _desc]             -- Higher-level objects
94.                               END,
95.                 [schema] = objschem.[name],
96.                 [objectName] = CASE perm.[class]

97.                                    WHEN 3 THEN perm
    schem.[name]       -- schemas
98.                                    WHEN 4 THEN imp.
    [name]             -- Impersonations
99.                                    ELSE OBJECT_NAME
    (perm.[major_id])  -- General objects
100.                              END,
101.                [columnName] = col.[name]
102.            FROM
103.                sys.database_principals
       AS roleprinc
104.                LEFT JOIN sys.database_permissio
    ns AS perm      ON perm.[grantee_principal_id] =
     roleprinc.[principal_id]
105.                LEFT JOIN sys.schemas
       AS permschem ON permschem.[schema_id] = perm.
    [major_id]
```

```
106.                 JOIN      sys.objects
       AS obj       ON obj.[object_id] = perm.[major
    _id]
107.                 LEFT JOIN sys.schemas
       AS objschem  ON objschem.[schema_id] = obj.[s
    chema_id]
108.                 LEFT JOIN sys.columns
       AS col       ON col.[object_id] = perm.[major
    _id]
109.
                     AND col.[column_id] = perm.[mino
    r_id]
110.                 LEFT JOIN sys.database_principal
    s  AS imp        ON imp.[principal_id] = perm.[ma
    jor_id]
111.             WHERE
112.                 roleprinc.[type] = ''R''
113.                 AND roleprinc.[name] = ''public'
    '
114.                 AND obj.[is_ms_shipped] = 0
115.             ORDER BY
116.             [userType],
117.             [databaseUserName],
118.             [loginName],
119.             [role],
120.             [schema],
121.             [objectName],
122.             [columnName],
123.             [permissionType],
124.             [permissionState],
125.             [objectType] '
126.             EXEC sp_executesql @Command
127.         ",
128.         'extract_database' => "
129.             SELECT name
130.             FROM sys.sysdatabases WHERE name NOT
     IN ('master', 'tempdb','model','msdb')
131.         ",
132.         'extract_servername' => "
133.             SELECT @@SERVERNAME as server
134.         ",
135.     ],
136.
```

```
137. ];
```

**Kode 5.2 Konfigurasi Query Aplikasi**

Konfigurasi *query* mengandung nilai – nilai *query* SQL Server. Nilai – nilai *query* SQL Server digunakan untuk mengekstrak data – data yang terdapat pada SQL Server. Konfigurasi *query* berperan untuk menjadikan *query* SQL Server terpusat, sehingga *query* yang digunakan dalam aplikasi tidak ditulis secara *hard code* pada tiap *file* PHP dan pengubahan *query* efektif dan mudah dilakukan dalam satu *file* PHP saja.

## 5.3 Konfigurasi Inisiasi Koneksi Aplikasi

Konfigurasi inisiasi koneksi aplikasi menjabarkan metode yang digunakan untuk memulai koneksi dari aplikasi ke basis data. Basis data yang digunakan dalam aplikasi ada dua, yaitu Neo4j dan SQL Server. Inisiasi koneksi aplikasi dengan SQL Server menggunakan ekstensi PDO SQL Server, sedangkan inisiasi koneksi aplikasi dengan Neo4j menggunakan *library* graphaware/neo4j-php-client. Inisiasi koneksi aplikasi dengan basis data SQL Server ditunjukkan pada Kode 5.3 berikut.

```
1.  if (! function_exists('createSQLServerConnection
    ')) {
2.      function createSQLServerConnection(string $s
    erver = null, int $port = null, string $username
     = null, string $password = null, string $databa
    se = null, string $prefix = 'sqlsrv') {
3.          $serverName = 'tcp:'.$server.','.$port;

4.          $conn = new PDO("$prefix:server=$serverN
    ame ; Database=$database", $username, $password)
    ;
5.          $conn-
    >setAttribute(PDO::SQLSRV_ATTR_ENCODING, PDO::SQ
    LSRV_ENCODING_UTF8);
6.
7.          return $conn;
8.      }
9.
```

```
10. }
```

**Kode 5.3 Fungsi Inisiasi Koneksi SQL Server**

Inisiasi koneksi aplikasi ke basis data SQL Server menggunakan fungsi yang telah didefinisikan pada *file* bootstrap/helpers.php. File ini berfungsi untuk mendaftarkan fungsi – fungsi PHP secara global saat *file* PHP yang terkait mengimpor *file* vendor/autoload.php. Prinsip penggunaan fungsi untuk inisiasi koneksi basis data SQL Server juga digunakan pada inisiasi koneksi basis data Neo4j. Hal tersebut dibuktikan pada Kode 5.4 berikut.

```
1.  if (! function_exists('createNeo4jConnection'))
    {
2.      function createNeo4jConnection(string $usern
    ame = null, string $password = null, string $hos
    t = null, int $port = null) {
3.          return ClientBuilder::create()
4.                      -
    >addConnection('bolt', "bolt://$username:$passwo
    rd@$host:$port")
5.                      ->build();
6.      }
7.  }
```

**Kode 5.4 Fungsi Inisiasi Koneksi Neo4j**

## 5.4 Implementasi Aplikasi

Implementasi aplikasi menjabarkan kode – kode yang digunakan untuk menjalankan aplikasi pada bagian atau komponen tertentu. Implementasi aplikasi berfokus pada komponen *Visualisation* dan *User Dependency Tool* yang telah dijabarkan pada subbab 4.2. Penjabaran terkait dengan implementasi kedua komponen tersebut terdapat pada subbab – subbab berikut ini.

### 5.4.1 Implementasi Komponen *User Dependency Tool*

Implementasi komponen *User Dependency Tool* berfokus pada implementasi ketiga bagian yang merupakan pembentuk dari

komponen tersebut. Ketiga bagian tersebut merupakan bagian *Processor*, *Extractor*, dan *Visualizer*. Penjelasan terkait dengan implementasi ketiga bagian tersebut terdapat pada subbab – subbab berikut ini.

#### 5.4.1.1 Implementasi Bagian *Extractor*

Implementasi bagian *Extractor* berfokus pada pengambilan data dari basis data SQL Server. Data utama yang diambil merupakan data pemetaan pengguna beserta hak akses dan perannya dengan objek basis data. Sesuai dengan *flowchart* yang telah dijelaskan pada subbab 4.5.3, berikut ini merupakan kode program untuk mengimplementasikan bagian *Extractor* pada Kode 5.5.

```
1.  <?php
2.
3.  namespace Dependency\Components;
4.
5.  use Dependency\Database\SqlConnectionMapper;
6.
7.  $resultColl = collect("resits")-
    >mapInto(SqlConnectionMapper::class);
8.  return $resultColl;
```

**Kode 5.5 Kode Program Bagian Extractor**

Kode 5.5 berfungsi untuk membuat dan mengembalikan koleksi (Collection) yang berisi objek SqlConnectionMapper. Pembuatan koleksi (Collection) dilakukan dengan menggunakan fungsi collect. Koleksi tersebut merupakan *Result Object* pada *flowchart* Gambar 4.5. Objek SqlConnectionMapper berguna untuk mengirim *query* ekstraksi pemetaan pengguna beserta peran dan hak akses terhadap objek basis data dan menyimpan hasil *query* ke dalam objek tersebut. Selain itu, objek tersebut menyimpan data – data lain yang diperlukan, seperti *query*, koneksi, dan nama basis data. Implementasi kelas SqlConnectionMapper terdapat pada Kode 5.6.

```
1.  <?php
```

```
2.
3.  namespace Dependency\Database;
4.
5.  use PDO;
6.
7.  class SqlConnectionMapper
8.  {
9.      protected $connection;
10.     protected $result;
11.     protected $query;
12.     protected $database;
13.
14.     public function __construct(string $db)
15.     {
16.         $databaseConfig = require config_path('d
    atabase.php');
17.         $queryConfig = require config_path('quer
    y.php');
18.         $dbConfig = $databaseConfig['connections
    ']['sqlsrv'];
19.         $this-
    >connection = createSQLServerConnection($dbConfi
    g['host'], $dbConfig['port'], $dbConfig['usernam
    e'], $dbConfig['password'], $db);
20.         $this-
    >query = $queryConfig['sqlserver']['extract_user
    '];
21.         $this->queryResult();
22.         $this->database = $db;
23.     }
24.
25.     private function queryResult()
26.     {
27.         $this->result = tap($this->connection-
    >prepare($this->query))->execute()-
    >fetchAll(PDO::FETCH_OBJ);
28.     }
29.
30.     public function getResult()
31.     {
32.         return $this->result;
33.     }
34.
35.     public function getDatabase()
```

```
36.     {
37.         return $this->database;
38.     }
39. }
```

**Kode 5.6 Kode Program Kelas SqlConnectionMapper (Result Object)**

### 5.4.1.2 Implementasi Bagian *Processor*

Implementasi bagian *Processor* berfokus pembuatan *graph* pada basis data Neo4j berdasarkan data yang disalurkan oleh bagian *Extractor*. Data tersebut dijuluki sebagai *Result Object* sesuai dengan *flowchart* pada Gambar 4.5. Kode program untuk implementasi bagian *Processor* dibagi menjadi beberapa bagian.

Bagian pertama merupakan bagian pembentukan *stack*, penghapusan seluruh *node* dan relasi, pembentukan *user* atau pengguna basis data Neo4j baru, dan pembuatan *node* yang berlabel *Server*. Penghapusan keseluruhan *node* dan relasi berguna untuk menjaga konsistensi *graph* dalam basis data Neo4j. Hal ini disebabkan oleh kemungkinan basis data Neo4j menyimpan *graph* yang dihasilkan dari proses sebelumnya. Bagian ini juga merupakan bagian awal pada *stack* CypherQuery dalam pembuatan *node* yang pertama. Seluruh aktivitas yang berhubungan dengan *graph*, seperti pembentukan *node*, penghapusan *node* dan/atau relasi, dan aktivitas lainnya pada bagian pertama sampai dengan seterusnya pada bagian Processor menggunakan CypherQuery yang dimasukkan ke dalam *stack*, terkecuali untuk pembuatan *user* atau pengguna basis data Neo4j baru. Hal ini bertujuan untuk meningkatkan efektifitas dan performa aplikasi dalam mengirim CypherQuery. Keseluruhan pembuatan *node* pada bagian *Processor* juga menyertakan pembuatan indeks dalam CypherQuerynya. Seluruh indeks pada bagian *Processor* merupakan indeks gabungan untuk atribut name dan surname. Pengguna basis data Neo4j baru yang dibuat pada bagian ini hanya diberikan peran berupa “reader”. Hal ini bertujuan supaya pengambilan data pada komponen *Visualisation* tidak

memerlukan pengguna yang memiliki peran “admin” dan kemanan aplikasi lebih terjaga. Berikut ini merupakan implementasi bagian pertama pada Kode 5.7.

```
1.  $extractServer  = $queryConfig['sqlserver']['ext
    ract_servername'];
2.
3.  $serverName = tap($sqlsrv-
    >prepare($extractServer))
4.  ->execute()
5.  ->fetchObject()
6.  ->server;
7.
8.  $serverArray  = [
9.      'serverName' => $serverName
10. ];
11. $stack = tap($neo4j->stack())-
    >push("MATCH (n) DETACH DELETE n");
12. $user = $neo4jAllConfig['username_read'];
13. $password = $neo4jAllConfig['password_read'];
14. try {
15.     $neo4j->run("
16.     CALL dbms.security.deleteUser('$user')
17.     CALL dbms.security.createUser('$user', '$pas
    sword', false)
18.     CALL dbms.security.addRoleToUser('reader', '
    $user')
19.     MATCH (n)
20.     RETURN n");
21. } catch (\Exception $e) {
22.     $neo4j->run("
23.     CALL dbms.security.createUser('$user', '$pas
    sword', false)
24.     CALL dbms.security.addRoleToUser('reader', '
    $user')
25.     MATCH (n)
26.     RETURN n");
27. }
28. $stack-
    >push("MERGE (s:Server {name: {serverName}, surn
    ame: {serverName}})", $serverArray);
29. $stack-
    >push("CREATE INDEX ON :Server(name, surname)");
```

```
30.  $serverPrefix = "{$serverName}.";
```

**Kode 5.7 Kode Program Bagian 1 pada Bagian Processor**

Bagian kedua merupakan bagian permulaan iterasi koleksi objek SqlConnectionMapper. Bagian ini juga berperan untuk membuat *node* yang berlabel Server dan Database, serta membentuk relasi antar kedua *node* dengan atribut DATABASE. Implementasi bagian kedua terdapat pada Kode 5.8 berikut.

```
1.   foreach ($extractResult as $result) {
2.       $databaseName = $result->getDatabase();
3.       $databasePrefix = "{$serverPrefix}{$database
     Name}.";
4.       $databaseArray = [
5.           'databaseName' => $databaseName,
6.           'databaseSurname' => rtrim($databasePref
     ix, "."),
7.       ];
8.       $serverToDbArray =  array_merge($serverArray
     , $databaseArray);
9.       $stack-
     >push("MERGE (s:Server {name: {serverName}, surn
     ame: {serverName}})
10.          MERGE (d:Database {name: {databaseName},
      surname: {databaseSurname}})
11.          MERGE (s)-[y:HAS_RELATIONSHIPS]->(d)
12.          SET y.DATABASE = 'Yes'", $serverToDbArra
     y);
13.      $stack-
     >push("CREATE INDEX ON :Database(name, surname)"
     );
```

**Kode 5.8 Kode Program Bagian 2 pada Bagian Processor**

Bagian ketiga merupakan bagian iterasi hasil *query* dan pemetaan hasil *query* terhadap variabel lokal. Bagian ini tidak memasukkan CypherQuery ke dalam *stack*, tetapi bagian ini hanya menginisiasikan variabel – variabel yang didapatkan dari hasil *query*. Variabel tersebut adalah tipe pengguna, pengguna,

peran, hak akses, tipe hak akses, status hak akses, tipe objek, nama objek, skema, dan nama kolom. Implementasi bagian ketiga terdapat pada Kode 5.9 berikut ini.

```
1.  $resultMapping = $result->getResult();
2.  foreach ($resultMapping as $eachMapping) {
3.      $userType = $eachMapping->userType;
4.      $databaseUserName = $eachMapping-
    >databaseUserName;
5.      $loginName = $eachMapping->loginName;
6.      $role = $eachMapping->role;
7.      $permissionType = $eachMapping-
    >permissionType ? str_replace(" ", "_", $eachMap
    ping->permissionType) : null;
8.      $permissionState = $eachMapping-
    >permissionState;
9.      $objectType = $eachMapping->objectType;
10.     $schema = $eachMapping->schema;
11.     $objectName = $eachMapping->objectName;
12.     $columnName = $eachMapping->columnName;
```

**Kode 5.9 Kode Program Bagian 3 pada Bagian Processor**

Bagian keempat merupakan bagian pembuatan *node* yang berlabel Database dan User dan relasi antar keduanya. Relasi yang dihasilkan dari bagian ini memiliki atribut USER_OF. Implementasi bagian keempat terdapat pada Kode 5.10 berikut.

```
1.  $dbUserArray = [
2.              'databaseUserName' => $databaseUserNam
    e,
3.              'databaseUserSurname' => "{$databasePr
    efix}{$databaseUserName}",
4.              'userType' => $userType,
5.          ];
6.          $dbToUserArray = array_merge($dbUserArray,
     $databaseArray);
7.          $stack-
    >push("MERGE (d:Database {name: {databaseName},
    surname: {databaseSurname}})
8.              MERGE (u:User {name: {databaseUserName
    }, type: {userType}, surname: {databaseUserSurna
    me}})
```

```
9.              MERGE (d)<-[y:HAS_RELATIONSHIPS]-(u)
10.             SET y.USER_OF = 'Yes'", $dbToUserArray
    );
11.         $stack-
    >push("CREATE INDEX ON :User(name, surname)");
```

**Kode 5.10 Kode Program Bagian 4 pada Bagian Processor**

Bagian kelima merupakan bagian pembentukan *node* yang berlabel Role dan User. Pembentukan *node* yang berlabel Role dan User juga diikuti pembentukan relasi antar kedua *node* tersebut. Relasi yang dibentuk dari bagian ini merupakan relasi yang beratribut MEMBER_OF. Pembentukan *node* dan relasi ini hanya terjadi apabila nilai $role atau peran pengguna tidak kosong atau bernilai *null*. Implementasi bagian kelima terdapat pada Kode 5.11 berikut ini.

```
1.  if ($role !== null) {
2.          $roleArray = [
3.              'role' => $role,
4.              'roleSurname' => "{$databasePrefix}
    {$role}"
5.          ];
6.
7.          $roleToUserArray = array_merge($roleArr
    ay, $dbUserArray);
8.
9.          $stack-
    >push("MERGE (r:Role {name: {role}, surname: {ro
    leSurname}})
10.             MERGE (u:User {name: {databaseUserN
    ame}, type: {userType}, surname: {databaseUserSu
    rname}})
11.             MERGE (r)<-[y:HAS_RELATIONSHIPS]-
    (u)
12.             SET y.MEMBER_OF = 'Yes'", $roleToUs
    erArray);
13.
14.         $stack-
    >push("CREATE INDEX ON :Role(name, surname)");
15.
```

```
16.       }
```

**Kode 5.11 Kode Program Bagian 5 pada Bagian Processor**

Bagian keenam merupakan bagian pembuatan *node* yang berlabel Database dan Schema. Pembentukan kedua *node* tersebut juga diikuti dengan pembentukan relasi antar kedua *node* tersebut. Relasi tersebut adalah relasi yang beratribut SCHEMA. Pembentukan kedua *node* tersebut beserta relasinya akan terjadi apabila nilai $schema atau skema tidak bernilai *null*. Implementasi bagian keenam terdapat pada Kode 5.12 berikut.

```
1.  $schemaPrefix = "{$databasePrefix}{$schema}.";
2.        if ($schema !== null) {
3.            $schemaArray = [
4.                'schema' => $schema,
5.                'schemaSurname' => rtrim($schemaPr
    efix, ".")
6.            ];
7.            $schemaToDbArray = array_merge($schema
    Array, $databaseArray);
8.            $stack-
    >push("MERGE (o:Database {name: {databaseName},
    surname:{databaseSurname}})
9.                MERGE (sc:Schema {name: {schema},
    surname:{schemaSurname}})
10.               MERGE (o)-[y:HAS_RELATIONSHIPS]-
    >(sc)
11.               SET y.SCHEMA = 'Yes'", $schemaToDb
    Array);
12.           $stack-
    >push("CREATE INDEX ON :Schema(name, surname)");

13.                }
```

**Kode 5.12 Kode Program Bagian 6 pada Bagian Processor**

Bagian ketujuh merupakan bagian yang kompleks, karena bagian ini terdiri dari beberapa proses dan pengecekan. Proses pertama dalam bagian ini merupakan pengecekan terhadap nilai $permissionType. Jikai nilai tersebut bernilai *null*, maka kode

program pada bagian ini tidak berlanjut atau jalan. Selanjutnya, bagian ini mengecek nilai $objectName atau nama objek. Untuk kasus nama objek yang tidak bernilai *null*, pembuatan *node* yang berlabel Object dan Schema akan dilakukan dan diikuti dengan pembentukan relasi antar keduanya. Relasi yang dibentuk antar kedua *node* tersebut merupakan relasi yang beratribute OWNS. Selanjutnya jika nilai $columnName atau nama kolom tidak bernilai *null*, maka pembuata *node* yang berlabel Column dan Object akan dilakukan dan diikuti dengan pembentukan relasi yang beratribut COLUMN antar kedua *node* tersebut. Selanjutnya, pembentukan dua *node* yang berlabel Column dan User akan dilakukan dan diikuti pembentukan relasi yang beratribut tipe hak akses yang bernilai status hak akses (Yes/No) antar kedua *node* tersebut. Jikai nilai $columnName atau nama kolom bernilai *null*, maka proses yang telah disebutkan sebelumnya tidak terjadi. Kemudian, untuk kasus nama objek bernilai *null*, maka kasus tersebut berlanjut dengan nilai $objectType atau tipe objek. Jika tipe objek berupa DATABASE, maka pembentukan *node* yang berlabel User dan Database akan dilakukan dan diikuti dengan pembentukan relasi yang beratribut tipe hak akses dengan nilai status hak akses. Jika tipe objek bernilai selain nilai yang telah disebutkan, maka pembentukan *node* yang berlabel Objek menggunakan nilai $objectType sebagai atribut name dan nilai $objectPrefix yang baru sebagai atribut surname dari *node* tersebut. Selanjutnya, pembentukan *node* yang berlabel User dan Object akan dilakukan dan diikuti dengan pembentukan relasi yang beratribut tipe hak akses dengan nilai status hak akses antar kedua *node* tersebut. Implementasi bagian ini terdapat pada Kode 5.13 berikut.

```
1.  if ($permissionType !== null) {
2.              $permissionState = $permissionState
    === "GRANT" ? "Yes": "No";
3.              $query = "MERGE (u:User {name: {data
    baseUserName}, type: {userType}, surname: {datab
    aseUserSurname}})
```

```
4.                  MERGE (o:Object {name: {objectNa
    me}, type: {objectType}, surname:{objectSurname}
    })
5.                  MERGE (u)-[p:HAS_RELATIONSHIPS]-
    >(o)
6.                  SET p.$permissionType = '$permis
    sionState'";
7.              if ($objectName !== null) {
8.                  $objectPrefix = "{$schemaPrefix}
    {$objectName}.";
9.                  $objectArray = [
10.                     'objectName' => $objectName,
11.                     'objectType' => $objectType,
12.                     'objectSurname' => rtrim($ob
    jectPrefix, ".")
13.                 ];
14.                 $schemaToObjArray = array_merge(
    $schemaArray, $objectArray);
15.                 $stack-
    >push("MERGE (o:Object {name: {objectName}, type
    : {objectType}, surname:{objectSurname}})
16.                     MERGE (sc:Schema {name: {sch
    ema}, surname:{schemaSurname}})
17.                     MERGE (o)<-
    [y:HAS_RELATIONSHIPS]-(sc)
18.                     SET y.OWNS = 'Yes'", $schema
    ToObjArray);
19.                 $stack-
    >push("CREATE INDEX ON :Object(name, surname)");
20.                 if ($columnName !== null) {
21.                     $columnArray = [
22.                         'columnName' => $columnN
    ame,
23.                         'columnSurname' => "{$ob
    jectPrefix}{$columnName}"
24.                     ];
25.                     $objToColumnArray = array_me
    rge($objectArray, $columnArray);
26.                     $stack-
    >push("MERGE (o:Object {name: {objectName}, type
    : {objectType}, surname:{objectSurname}})
```

```
27.                             MERGE (c:Column {name: {
    columnName}, surname:{columnSurname}})
28.                             MERGE (o)-
    [y:HAS_RELATIONSHIPS]->(c)
29.                             SET y.COLUMN = 'Yes'", $
    objToColumnArray);
30.                         $stack-
    >push("CREATE INDEX ON :Column(name, surname)");

31.                         $objectArray = $columnArray;

32.                         $query = "MERGE (u:User {nam
    e: {databaseUserName}, type: {userType}, surname
    : {databaseUserSurname}})
33.                             MERGE (o:Column {name: {
    columnName}, surname:{columnSurname}})
34.                             MERGE (u)-
    [p:HAS_RELATIONSHIPS]->(o)
35.                             SET p.$permissionType =
    '$permissionState'";
36.                     }
37.                 } else {
38.                     switch ($objectType) {
39.                         case "DATABASE":
40.                             $objectArray = [
41.                                 'objectName' => $dat
    abaseName,
42.                                 'objectType' => $obj
    ectType,
43.                                 'objectSurname' => r
    trim($databasePrefix, ".")
44.                             ];
45.                             $query = "MERGE (u:User
    {name: {databaseUserName}, type: {userType}, sur
    name: {databaseUserSurname}})
46.                                 MERGE (o:Database {n
    ame: {objectName}, surname:{objectSurname}})
47.                                 MERGE (u)-
    [p:HAS_RELATIONSHIPS]->(o)
48.                                 SET p.$permissionTyp
    e = '$permissionState'";
49.                         break;
50.                         default:
```

```
51.                             $objectPrefix = "{$schem
    aPrefix}{$objectType}.";
52.                             $objectArray = [
53.                                 'objectName' => $obj
    ectType,
54.                                 'objectType' => $obj
    ectType,
55.                                 'objectSurname' => r
    trim($objectPrefix, ".")
56.                             ];
57.                     }
58.                 }
59.                 $userToPermToObjArray = array_merge(
    $dbUserArray, $objectArray);
60.                 $stack-
    >push($query, $userToPermToObjArray);
61.                 $stack-
    >push("CREATE INDEX ON :Object(name, surname)");

62.             }
```

**Kode 5.13 Kode Program Bagian 7 pada Bagian Processor**

Bagian kedelapan merupakan bagian akhir dari program. Bagian ini menjalankan seluruh CypherQuery yang terdapat dalam *stack*. Semua CypherQuery yang menggunakan kata "MERGE" di dalamnya akan membuat node baru apabila node yang didefinisikan tidak ada dan mengembalikan nilai *node* dari *node* yang terbuat atau *node* yang ditemukan. Hal tersebut juga berlaku untuk relasi. Setelah itu, bagian ini men-*dump* kata "SUCCEDED" dan mengakhiri program. Implementasi bagian kedelapan ini terdapat pada Kode 5.14.

```
1.
2.     }
3.
4.
5.  }
6.
7.  $neo4j->runStack($stack);
8.
```

```
9.   dd("SUCCEDED");
```

**Kode 5.14 Kode Program Bagian 8 pada Bagian Processor**

#### 5.4.1.3 Implementasi Bagian *Visualizer*

Implementasi bagian *visualizer* berfokus pada pemrosesan data dari CypherQuery yang mengembalikan nilai *graph* ke dalam data yang berformat JSON. Format JSON yang dibutuhkan sesuai dengan rujukan pada Gambar 3.3 dalam subbab 3.1.5. Kode program implementasi bagian *Visualizer* terdapat pada Kode 5.15 berikut ini.

```
1.   $resultNeo4jCollection = collect([
2.           'results' => [
3.               [
4.                   'data' => [
5.                       [
6.                           'graph' => [
7.                               'nodes'=> array_valu
     es($nodeResColl->toArray()),
8.                               'relationships' => a
     rray_values($relResColl->toArray()),
9.                           ]
10.                      ]
11.                  ]
12.              ]
13.          ],
14.      ]);
```

**Kode 5.15 Kode Program Implementasi Bagian Visualizer**

### 5.4.2 Implementasi Komponen *Visualisation*

Implementasi komponen *Visualisation* berfokus pada implementasi tampilan untuk pengguna. Implementasi komponen ini terdiri dari beberapa bagian. Penjabaran implementasi tiap bagian pada komponen *Visualisation* terdapat pada paragraf – paragraf di bawah ini.

Bagian inisiasi koleksi merupakan proses pemetaan hasil dari eksekusi CypherQuery untuk mendapatkan seluruh *node* dan

relasinya ke dalam variabel – variabel koleksi. Variabel – variabel tersebut merupakan koleksi relasi, koleksi label, dan koleksi *node*. Implementasi terkait dengan bagian inisiasi koleksi terdapat pada Kode 5.16 berikut.

```
1.  $stack = tap($neo4j->stack())-
    >push('MATCH (n) RETURN labels(n) as labels, n')
    ;
2.  $stack->push('MATCH (x)-[y]-(z) RETURN y');
3.  $results = $neo4j->runStack($stack);
4.  $labelsCollection = collect();
5.  $nodesCollection = collect();
6.  $relsCollection = collect();
7.  foreach($results as $result){
8.      foreach($result-
    >getRecords() as $record) {
9.          $labels = $record->get('labels');
10.         $nodes = $record->get('n');
11.         $rel = $record->get('y');
12.         if ($labels) {
13.             $labelsCollection->push($labels);
14.         }
15.
16.         if ($nodes) {
17.             $nodesCollection->push($nodes);
18.         }
19.
20.         if ($rel) {
21.             $relsCollection->push($rel);
22.         }
23.
24.     }
25.
26. }
27.
28. $relsCollection = $relsCollection-
    >map(function($value, $key){
29.     return $value->values();
30. })->flatMap(function($value){
31.     return array_keys($value);
32. })->unique();
33.
```

```
34. $labelsCollection = $labelsCollection-
    >flatten()->uniqueStrict();
```

**Kode 5.16 Kode Program Pemetaan Seluruh Node dan Relasi Terhadap Koleksi**

Bagian selanjutnya merupakan bagian *form* filter. Bagian ini merupakan bagian yang menjadi jembatan interaksi antara pengguna dengan aplikasi. *Form* filter berisi isian berupa label *node* asal, nama julukan *node* asal, atribut relasi, label *node* tujuan, nama julukan *node* tujuan, dan jumlah *hop*. Nilai maksimal jumlah *hop* merupakan dua dan nilai minimalnya adalah satu. Implementasi bagian *form* filter terdapat pada Kode 5.17 berikut ini.

```
1.  <form enctype="multipart/form-
    data" action="./user_viewer.php" method="post">

2.      <input type="hidden" name="search_mode" valu
    e="filter" />
3.      <div class="form-group">
4.          <label for="from_node_type" class="bmd-
    label-floating">Node Source Type</label>
5.          <select name="from_node_type" id="from_n
    ode_type" class="form-control select-node-
    type" data-point="from">
6.              <option value="">-
    - Nothing Selected --</option>
7.              <?php
8.              foreach ($labelsCollection as $label
    ) {
9.          ?>
10.                 <option value="<?php echo $label
    ; ?>">
11.                     <?php echo $label; ?>
12.                 </option>
13.                 <?php
14.             }
15.         ?>
16.         </select>
17.         <span class="bmd-
    help">Please select a node type!</span>
18.     </div>
```

```
19.      <?php
20.          foreach($labelsCollection as $label) {
21.      ?>
22.          <div class="form-group from-
    input" id="<?php echo 'from_'.$label; ?>" style=
    "display:none">
23.              <label for="from_node" class="bmd-
    label-floating">Node Source Surname</label>
24.              <select name="from_node[]" class="fo
    rm-control from_node">
25.                  <option value="">-
    - Nothing Selected --</option>
26.                  <?php
27.                  foreach($nodesCollection-
    >filter(function($value, $key) use($label){
28.                      return $value-
    >hasLabel($label);
29.                  }) as $node){
30.              ?>
31.                      <option value="<?php echo $n
    ode->identity();?>">
32.                          <?php echo $node-
    >value('surname');?>
33.                      </option>
34.                      <?php
35.              }
36.              ?>
37.              </select>
38.              <span class="bmd-
    help">Please select a node!</span>
39.          </div>
40.          <?php
41.          }
42.      ?>
43.              <div class="form-group">
44.                  <label for="relationships" class
    ="bmd-label-
    floating">Relationship's Properties</label>
45.                  <select multiple="multiple" name
    ="relationships[]" id="relationships" class="for
    m-control selectpicker show-tick">
46.                      <option value="">-
    - Nothing Selected --</option>
47.                      <?php
```

```
48.                     foreach ($relsCollection as
    $rel) {
49.
50.                 ?>
51.                     <option value="<?php echo $r
    el;?>">
52.                         <?php echo $rel;?>
53.                     </option>
54.                     <?php
55.                     }
56.                 ?>
57.                 </select>
58.                 <span class="bmd-
    help">Please select the relationships!</span>
59.             </div>
60.             <div class="form-group">
61.                 <label for="to_node_type" class=
    "bmd-label-
    floating">Node Destination Type</label>
62.                 <select name="to_node_type" id="
    to_node_type" class="form-control select-node-
    type" data-point="to">
63.                     <option value="">-
    - Nothing Selected --</option>
64.                     <?php
65.             foreach ($labelsCollection as $label
    ) {
66.         ?>
67.                         <option value="<?php ech
    o $label; ?>">
68.                             <?php echo $label; ?
    >
69.                         </option>
70.                         <?php
71.             }
72.         ?>
73.                 </select>
74.                 <span class="bmd-
    help">Please select a node type!</span>
75.             </div>
76.             <?php
77.             foreach($labelsCollection as $label)
     {
78.         ?>
```

```
79.                 <div class="form-group to-
    input" id="<?php echo 'to_'.$label; ?>" style="d
    isplay:none">
80.                     <label for="to_node" class="
    bmd-label-
    floating">Node Destination Surname</label>
81.                     <select name="to_node[]" cla
    ss="form-control to_node">
82.                         <option value="">-
    - Nothing Selected --</option>
83.                         <?php
84.                         foreach($nodesCollection
    ->filter(function($value, $key) use($label){
85.                             return $value-
    >hasLabel($label);
86.                         }) as $node){
87.                     ?>
88.                             <option value="<?php
     echo $node->identity();?>">
89.                                 <?php echo $node
    ->value('surname');?>
90.                             </option>
91.                             <?php
92.                         }
93.                     ?>
94.                     </select>
95.                     <span class="bmd-
    help">Please select a node!</span>
96.                 </div>
97.                 <?php
98.                 }
99.             ?>
100.                <div class="form-group">
101.                    <label for="hop_count" class
    ="bmd-label-floating">Radius</label>
102.                    <input type="number" name="h
    op_count" min="1" max="2" step="1" value="1" />

103.                    <span class="bmd-
    help">Specify the radius!</span>
104.                </div>
105.                <button type="submit" class="btn
     btn-primary">Generate</button>
```

```
106. </form>
```

**Kode 5.17 Implementasi Form Filter**

Bagian inisiasi filter merupakan bagian awal untuk implementasi fungsi filter pada aplikasi. Bagian ini menginisiasikan variabel – variabel lokal yang dipetakan dari variabel global $_POST. Implementasi bagian inisiasi filter terdapat pada Kode 5.18 berikut.

```
1.  try {
2.      $postVar = (object) $_POST;
3.      $from_node_type = $postVar-
    >from_node_type;
4.      $fromColl = collect($postVar->from_node)-
    >filter(function($value, $key){
5.          return $value !== "";
6.      });
7.      $to_node_type = $postVar->to_node_type;
8.      $toColl = collect($postVar->to_node)-
    >filter(function($value, $key){
9.          return $value !== "";
10.     });
11.     $relColl = collect($postVar-
    >relationships);
12.     $hop = $postVar->hop_count;
```

**Kode 5.18 Implementasi Inisiasi Filter**

Bagian berikutnya merupakan bagian implementasi *query* filter dan eksekusi *query* tersebut. Pembentukan *query* filter didasarkan pada masukan yang diterima oleh aplikasi dari pengguna. Pembentukan *query* tersebut menggabungkan beberapa *query* jadi satu *query*, sehingga hasil dari eksekusi *query* hanya berjumlah satu. Hal tersebut berbeda dengan menggunakan *stack* yang dapat menghasilkan lebih dari satu hasil eksekusi *query*. Implementasi bagian ini terdapat pada Kode 5.19 berikut.

```
1.  $query = "MATCH (n)-
    [x:HAS_RELATIONSHIPS*..$hop]->(y) ";
2.  if ($from_node_type !== "") {
```

```
3.      $query .= "WHERE n:$from_node_type ";
4.  }
5.  if ($to_node_type !== "") {
6.      $query .= "AND y:$to_node_type ";
7.  }
8.  if (!$fromColl->isEmpty()) {
9.      $from_node = $fromColl->first();
10.     $query .= "AND id(n) = $from_node ";
11. }
12. if (!$toColl->isEmpty()) {
13.     $to_node = $toColl->first();
14.     $query .= "AND id(y) = $to_node ";
15. }
16. $query .= "UNWIND x AS rel ";
17. if (!$relColl->isEmpty()) {
18.     foreach($relColl as $relationship) {
19.         $query .= "WITH * ";
20.         $query .= "WHERE exists(rel.$relationshi
    p) ";
21.     }
22. }
23. $query .= "RETURN DISTINCT n, rel, y";
24.
25. $results = $neo4j->run($query);
```

**Kode 5.19 Implementasi Query Filter dan Eksekusinya**

Selanjutnya, bagian yang akan dibahas merupakan bagian pemetaan hasil filter terhadap koleksi. Hasil filter yang dipetakan merupakan *node* dan relasi – relasinya. Pemetaan hasil filter tersebut melakukan iterasi terhadap hasil eksekusi *query* pada bagian sebelumnya dan memasukkan *node* dan relasi ke dalam koleksi yang terkait. Koleksi yang dipetakan tersebut dipastikan memiliki elemen *node* dan relasi yang unik. Implementasi bagian pemetaan hasil filter terhadap koleksi terdapat pada Kode 5.20 berikut ini.

```
1.  $nodeResColl = collect();
2.  $relResColl = collect();
3.  foreach($results->records() as $record) {
4.      $nodeSource = $record->get('n');
5.      $nodeDest = $record->get('y');
6.      $relationship = $record->get('rel');
```

```
7.      $nodeResColl->push($nodeSource);
8.      $nodeResColl->push($nodeDest);
9.      $relResColl->push($relationship);
10. }
11. if(!$nodeResColl->isEmpty()) {
12.     $nodeResColl = $nodeResColl-
    >uniqueStrict(function($value) {
13.         return $value->identity();
14.     })->map(function ($value) {
15.         return [
16.             'id' => $value->identity(),
17.             'labels' => $value->labels(),
18.             'properties' => $value->values()
19.         ];
20.     });
21. }
22. if(!$relResColl->isEmpty()){
23.     $relResColl = $relResColl-
    >uniqueStrict(function($value) {
24.         return $value->identity();
25.     })->map(function ($value) {
26.         return [
27.             'id' => $value->identity(),
28.             'type' => $value->type(),
29.             'startNode' => $value-
    >startNodeIdentity(),
30.             'endNode' => $value-
    >endNodeIdentity(),
31.             'properties' => $value->values()
32.         ];
33.     });
34. }
```

**Kode 5.20 Implementasi Pemetaan Node dan Relasi dari Hasil Filter Terhadap Koleksi**

Bagian selanjutnya merupakan bagian penanganan eror pada komponen *Visualisation*. Penanganan ini bertujuan untuk mencegah terminasi program apabila eror terjadi pada saat proses filter. Penanganan ini hanya berupa alokasi nilai *true* pada variabel $error dan nilai pesan eror pada variabel $errorMessage. Implementasi penanganan eror pada komponen *Visualisation* terdapat pada Kode 5.21 berikut ini.

```
} catch (\Exception $e) {
    $error = true;
    $errorMessage = $e->getMessage();
}
```

**Kode 5.21 Implementasi Error Handling pada Filter**

Bagian berikutnya merupakan bagian tampilan eror. Bagian ini bertugas untuk menampilkan eror apabila bagian sebelumnya berfungsi atau terjadi eror pada aplikasi. Bagian ini hanya menampilkan kotak peringatan eror kepada pengguna beserta pesan yang telah ditetapkan pada bagian sebelumnya. Implementasi bagian tampilan eror terdapat pada Kode 5.22 berikut.

```
<?php
            if($error) {
        ?>
        swal({
            type: 'error',
            title: 'Alert!',
            text: '<?php echo $errorMessage; ?>'
,
            showCloseButton: true,
            showConfirmButton: false,
        });
```

**Kode 5.22 Implementasi Tampilan Eror**

Berikutnya, bagian yang akan dibahas adalah bagian tampilan *graph*. Bagian ini bertujuan untuk menampilkan *graph* dengan data yang berformat. Pengubahan data dari hasil eksekusi *query* filter ke data yang berformat JSON menggunakan fungsi toJson. Implementasi bagian ini dan pengubahan hasil eksekusi *query* ke data yang berformat JSON terdapat pada Kode 5.23 berikut.

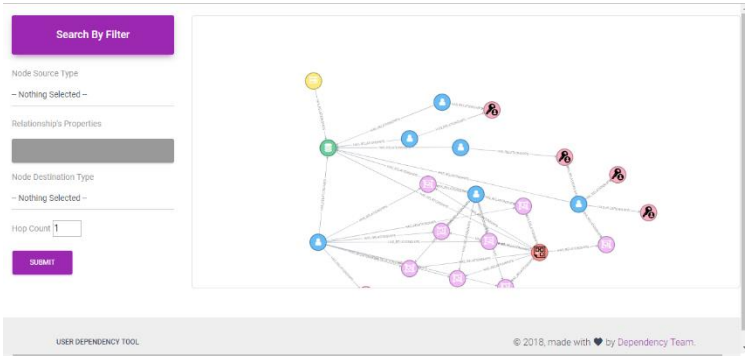
```
<?php
            } else {
        ?>
        let neo4jd3 = new Neo4jd3('#neo4jd3', {

            icons: {
```

```
6.                  'Server': 'server',
7.                  'Database': 'database',
8.                  'Column': 'columns',
9.                  'Object': 'object-group',
10.                 'User': 'user'
11.             },
12.             images: {
13.                 'Schema': './assets/icons/schema
    .svg',
14.                 'Role': './assets/icons/role.svg
    '
15.             },
16.             minCollision: 60,
17.             neo4jData: <?php echo $resultNeo4jCo
    llection->toJson(); ?>,
18.             nodeRadius: 20,
19.             zoomFit: true,
20.         });
21.         <?php
22.             }
23.         ?>
```

**Kode 5.23 Implementasi Tampilan Graph**

Bagian terakhir merupakan bagian tampilan antarmuka. Pembahasan terkait dengan bagian ini terdapat pada subbab 4.6. Implementasi bagian tampilan antarmuka sama dengan desain antarmuka pada Gambar 4.6. Bagian ini hanya menambahkan keterangan terkait dengan gambar atau *icon* yang digunakan untuk merepresentasikan label *node* pada tampilan *graph*. Implementasi bagian tampilan antarmuka terdapat pada Gambar 5.1 berikut ini. Penambahan keterangan terkait dengan gambar atau *icon* terdapat pada Tabel 5.3 berikut.




**Gambar 5.1 Impementasi Tampilan Antarmuka**

**Tabel 5.3 Tabel Daftar Logo dan Keterangannya**

| Logo | Arti |
|---|---|
| | Server |
| | Database |
| | Column |
| | Object |
| | User |

| | |
|---|---|
|  | Role |
|  | Schema |

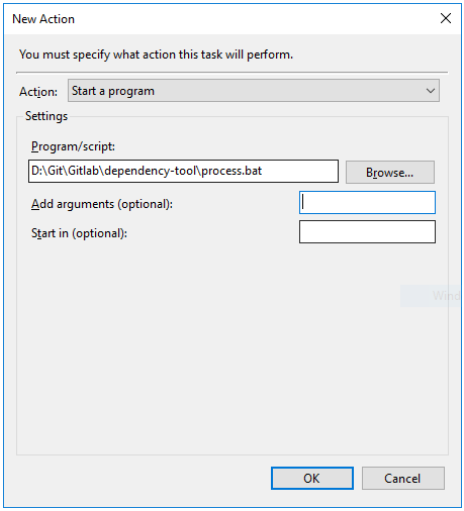
## 5.5 Konfigurasi Otomatisasi

Konfigurasi otomatisasi menjelaskan eksekusi kode program atau skrip yang digunakan untuk menjalankan bagian *Processor* dan bagian *Extractor* secara otomatis. Konfigurasi ini menggunakan kode program atau skrip *file* process.bat untuk menjalankan ekstraksi dan pembentukan *graph* dalam *background*. Implementasi kode program *file* process.bat terdapat pada Kode 5.24 berikut.

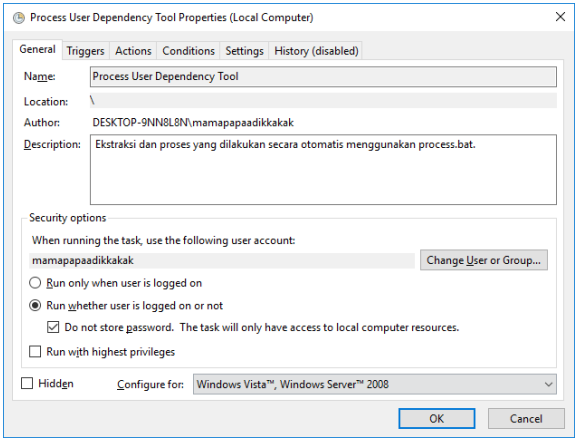
```
php -f D:\Git\Gitlab\dependency-
tool\app\Components\user_process.php
```

**Kode 5.24 Kode Program File "process.bat" (Eksekusi Proses dan Ekstraksi)**

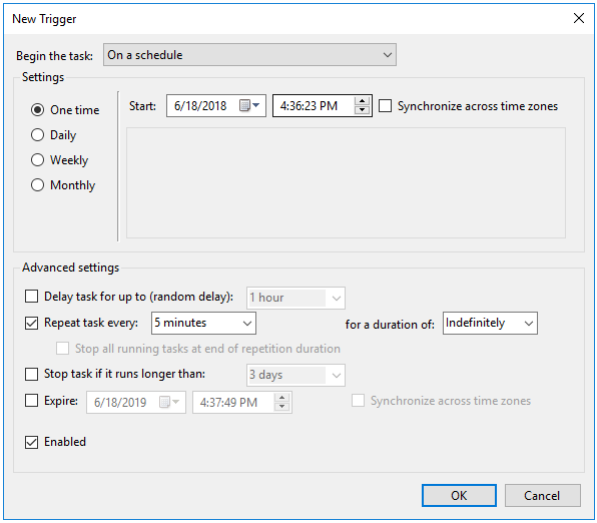
Konfigurasi ini juga mendefinisikan penjadwalan dengan menggunakan *Task Scheduler* di Windows. Penjadwalan ini bertujuan untuk mengeksekusi file process.bat pada tiap lima menit secara otomatis. Implementasi penjadwalan dengan menggunakan *Task Scheduler* terdapat pada Gambar 5.2, Gambar 5.3, dan Gambar 5.4 berikut ini.




**Gambar 5.2 Konfigurasi Action Task Scheduler**




**Gambar 5.3 Konfigurasi Umum Task Scheduler**




**Gambar 5.4 Konfigurasi Trigger Task Scheduler**

# BAB VI
# HASIL DAN PEMBAHASAN

Bab ini akan menjelaskan mengenai data dan pembahasan hasil pengujian aplikasi. Pembahasan hasil pengujian aplikasi mencakup hasil implementasi yang telah dijelaskan pada bab sebelumnya. Penjabaran lebih detail terkait dengan data dan pembahasan hasil pengujian aplikasi terdapat pada subbab – subbab berikut ini.

## 6.1 Data Hasil Pengujian

Data hasil pengujian aplikasi *User Dependency Tool* dibagi menjadi dua bagian, yaitu data hasil pengujian fungsionalitas dan data hasil pengujian performa. Pembagian ini menyesuaikan dengan bahasan desain pengujian aplikasi yang terdapat pada subbab 4.8. Data hasil pengujian tidak mencakup keseluruhan komponen dalam aplikasi *User Dependency Tool*, karena data hasil pengujian hanya merujuk pada komponen dan/atau bagian yang dibahas pada subbab tersebut.

### 6.1.1 Data Hasil Pengujian Fungsionalitas

Data hasil pengujian fungsionalitas aplikasi *User Dependency Tool* menunjukkan status berhasil untuk keseluruhan kasus. Kasus dengan huruf A berlaku untuk komponen *Visualisation*. Kasus dengan huruf B berlaku untuk bagian *Visualizer*. Kasus dengan huruf C dan D berlaku untuk bagian *Processor* dan *Extractor* secara berturut – turut. Data hasil pengujian fungsionalitas yang ditunjukkan pada Tabel 6.1 menyatakan bahwa seluruh pengujian bagi seluruh bagian atau komponen berhasil dilakukan dan memberikan hasil sesuai yang diharapkan.

**Tabel 6.1 Data Hasil Pengujian Fungsionalitas Berdasarkan Kasus - Kasus**

| Bagian /Komponen | | ID | Cara Pengujian | Status |
|---|---|---|---|---|
| Visualisation | | A01 | Menampilkan *form* filter untuk interaksi pengguna. | Berhasil |
| | | A02 | Memfilter informasi yang ditampilkan dalam *graph*. | Berhasil |
| | | A03 | Menampilkan *graph* dari data yang berformat JSON. | Berhasil |
| | | A04 | Menampilkan pesan eror saat terjadi eror dalam proses filter. | Berhasil |
| Visualizer | | B01 | Mengubah data *graph* Neo4j ke format JSON. | Berhasil |
| Processor | | C01 | Menghapus seluruh *node* dan relasi dalam graph dengan CypherQuery. | Berhasil |
| | | C02 | Mengirim T-SQL *query* untuk mendapatkan nama *server*. | Berhasil |
| | | C03 | Membuat *node* dan relasinya dengan menggunakan data pada *result object* yang didapatkan dari D02 dan CypherQuery. | Berhasil |

| | | | | |
|---|---|---|---|---|
| | | C04 | Menambahkan indeks pada seluruh label *node* untuk properti name dan surname dengan CypherQuery. | Berhasil |
| Extractor | | D01 | Mengirim T-SQL *query* untuk mendapatkan pemetaan pengguna beserta peran dan hak aksesnya dengan objek basis data. | Berhasil |
| | | D02 | Memetakan hasil *query* dari D01 ke dalam *result object*. | Berhasil |

### 6.1.2 Data Hasil Pengujian Performa

Data hasil pengujian performa untuk kedua bagian *Extractor* dan *Processor* merupakan rata – rata waktu eksekusi yang dibutuhkan untuk masing – masing bagian tersebut. Rata – rata waktu eksekusi didapatkan dari percobaan eksekusi masing – masing bagian selama enam kali. Rata – rata waktu eksekusi bagian *Extractor* adalah 0,29 detik. Rata – rata waktu eksekusi bagian *Processor* adalah 1,83 detik. Kedua rata – rata tersebut didapatkan dari Tabel 6.2. Waktu eksekusi kedua bagian tiap percobaan selama enam kali juga terdapat pada Tabel 6.2.

**Tabel 6.2 Data Hasil Pengujian Performa**

| Percobaan ke- | Bagian | |
|---|---|---|
| | Extractor (s) | Processor (s) |
| 1 | 0.39443111419678 | 2.0522000789642 |
| 2 | 0.24651885032654 | 1.9167962074280 |
| 3 | 0.24666213989258 | 1.7466199398041 |

| | | |
|---|---|---|
| 4 | 0.25508117675781 | 1.7147779464722 |
| 5 | 0.26723909378052 | 1.9117012023926 |
| 6 | 0.20542287826538 | 1.6770081520081 |
| Rata - rata | 0.26922587553660 | 1.8365172545115 |

## 6.2 Pembahasan Hasil Pengujian

Subbab ini menjabarkan hasil – hasil yang didapatkan dari pengujian aplikasi *User Dependency Tool* untuk dua bagian yang telah dibahas pada subbab 6.1. Subbab ini bertujuan untuk menambahkan keterangan atau penjelasan terkait dengan data hasil pengujian. Keterangan atau penjelasan tersebut dapat berupa bukti pengujian, skenario, dan/atau keluaran aplikasi. Penjabaran lebih detail terkait dengan pembahasan hasil pengujian diuraikan pada subbab – subbab berikut ini.

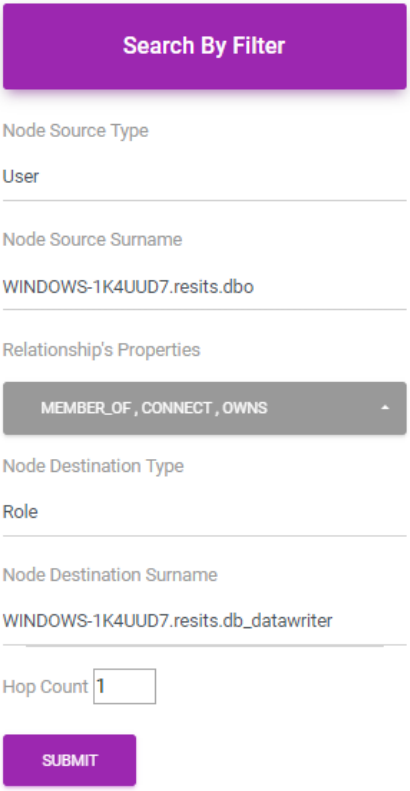
### 6.2.1 Pembahasan Hasil Pengujian Fungsionalitas

Penjabaran hasil pengujian fungsionalitas tidak berfokus pada komponen atau bagian, tetapi penjabaran hasil pengujian fungsionalitas berfokus pada masing – masing kasus. Hal tersebut bertujuan untuk memberikan keterangan atau penjelasan tambahan yang lebih mendetail pada tiap kasus. Penjabaran hasil pengujian fungsionalitas untuk masing – masing kasus diuraikan pada subbab – subbab berikut ini.

#### 6.2.1.1 Pembahasan Hasil Pengujian A01

Kasus pengujian A01 merupakan kasus yang diuji dengan menampilkan *form* filter untuk interaksi pengguna. Hasil dari kasus tersebut terdapat pada Gambar 6.1 berikut. Hasil yang ditampakkan pada gambar tersebut berupa *form* filter untuk memfilter tampilan *graph*. *Form* filter yang ditunjukkan pada gambar tersebut berhasil menampilkan isian tipe *node* asal dan nama julukan tipe *node* asal yang dipilih. Isian tipe *node* mengandung seluruh label yang terdapat dalam *graph* dan isian nama julukan mengandung seluruh nama julukan yang terdapat dalam *graph* sesuai dengan tipe *node* yang dipilih. Hal tersebut

juga berlaku untuk isian tipe *node* tujuan dan nama julukan tipe *node* tujuan yang dipilih. Selain keempat isian tersebut, *form* filter berhasil menunjukkan seluruh atribut yang terdapat pada seluruh relasi HAS_RELATIONSHIPS dan terdapat pada isian relasi. *Form* filter juga berhasil menunjukkan isian jumlah *hop* dengan nilai maksimum satu dan nilai minimum dua.



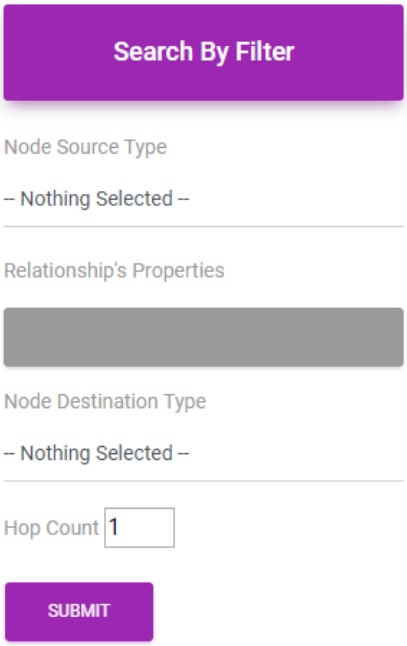

**Gambar 6.1 Hasil Kasus Pengujian A01**

Berikut ini merupakan keterkaitan isian dengan *form* filter yang ditunjukkan pada Gambar 6.1. Isian tipe *node* asal ditunjukkan dengan nama/label Node Source Type. Isian nama julukan tipe *node* asal ditunjukkan dengan nama/label Node Source

Surname. Isian relasi ditunjukkan dengan nama/label Relationships's Properties. Isian tipe *node* tujuan ditunjukkan dengan nama/label Node Destination Type. Isian nama julukan tipe *node* tujuan ditunjukkan dengan nama/label Node Destination Surname. Isian jumlah *hop* ditunjukkan dengan nama/label Hop Count.

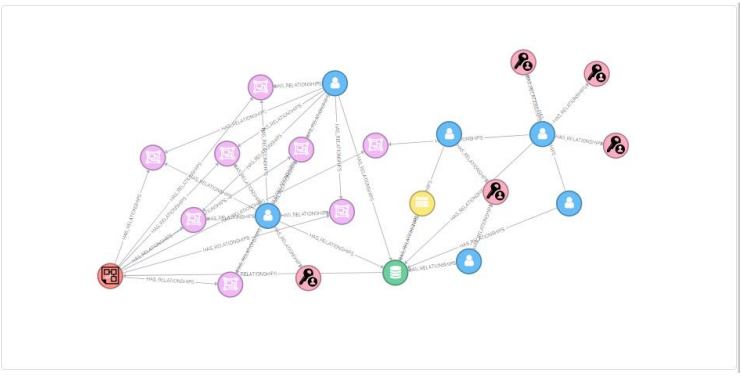
#### 6.2.1.2 Pembahasan Hasil Pengujian A02

Kasus pengujian A02 merupakan kasus yang diuji dengan memfilter informasi yang ditampilkan dalam *graph*. Kasus pengujian A02 dibagi menjadi beberapa skenario. Hal ini bertujuan untuk mendapatkan hasil pengujian yang berbeda pada tiap skenario. Perbedaan hasil pengujian menunjukkan keberhasilan proses filter pada aplikasi *User Dependency Tool*.

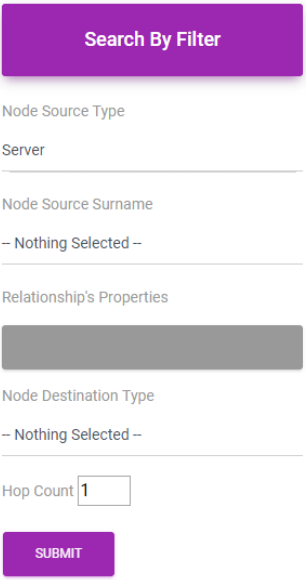



**Gambar 6.2 Filter Skenario Pertama Kasus Pengujian A02**

Skenario pertama merupakan skenario filter tanpa mengubah masukan apapun pada filter. Skenario ini mengharapkan tampilan *graph* untuk keseluruhan *node* dan relasi – relasinya. Filter yang digunakan pada skenario pertama terdapat pada Gambar 6.2. Hasil dari skenario pertama ini ditunjukkan pada Gambar 6.3. Hasil skenario pada Gambar 6.3 dengan hasil yang diharapkan pada skenario ini sesuai, sehingga skenario ini berhasil dilakukan.



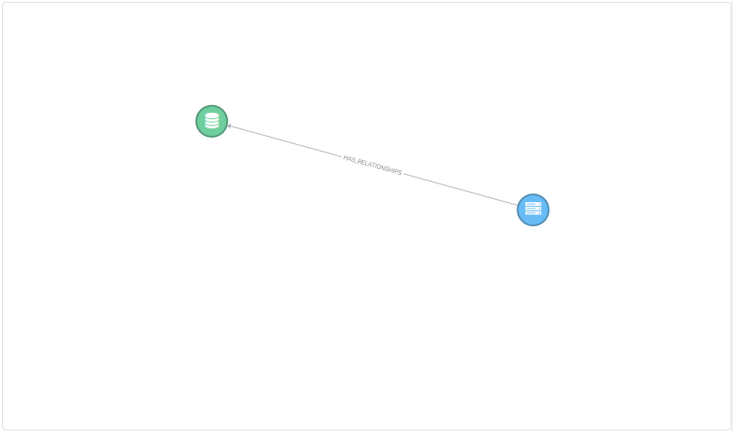
**Gambar 6.3 Hasil Skenario Pertama Kasus Pengujian A02**



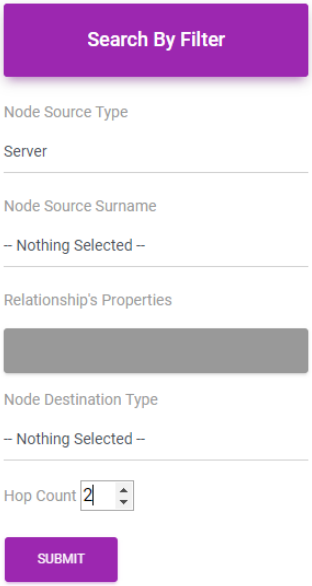

**Gambar 6.4 Filter Skenario Kedua Kasus Pengujian A02**

Skenario kedua merupakan skenario filter dengan menggunakan tipe *node* asal berlabel Server. Skenario ini mengharapkan tampilan *graph* yang berisi *node* yang berlabel Server dengan *node* lainnya dalam satu *hop* (lompatan) relasi. Filter yang digunakan dalam skenario terdapat pada Gambar

6.4. Hasil skenario kedua ditunjukkan pada Gambar 6.5 berikut. Hasil skenario tersebut sesuai dengan hasil yang diharapkan pada skenario ini, sehingga skenario ini berhasil dilakukan.



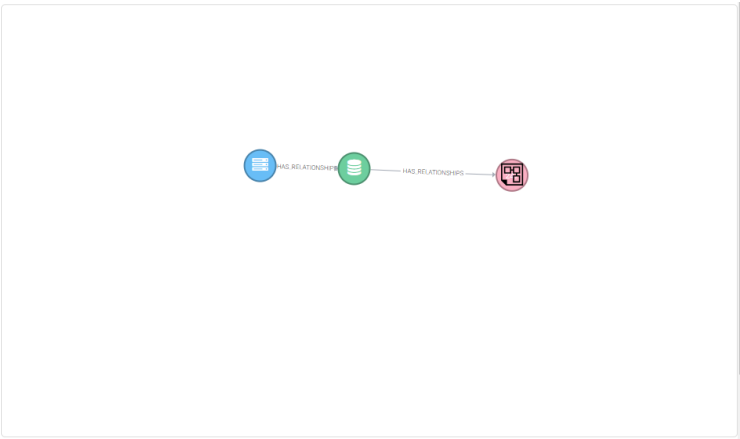

**Gambar 6.5 Hasil Skenario Kedua Kasus Pengujian A02**



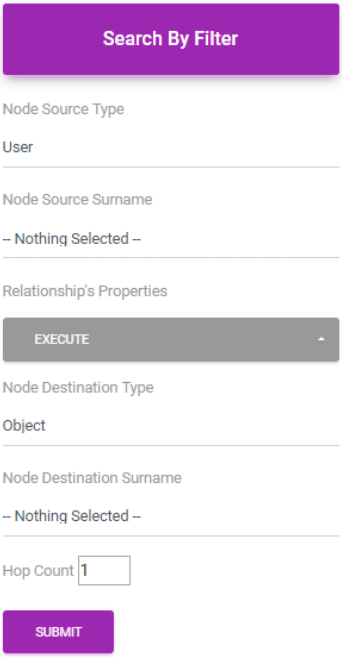

**Gambar 6.6 Filter Skenario Ketiga Kasus Pengujian A02**

Skenario ketiga merupakan skenario filter yang sama dengan skenario kedua dengan perbedaan pada jumlah *hop*. Jumlah *hop* yang digunakan pada skenario ini adalah dua. Hasil yang

diharapkan dari skenario ini sama dengan skenario sebelumnya dengan perbedaan pada penggunaan dua lompatan (hop) relasi. Filter yang digunakan pada skenario ini terdapat pada Gambar 6.6. Hasil skenario ini terdapat pada Gambar 6.7 berikut. Hasil skenario tersebut dengan hasil yang diharapkan pada skenario ini sesusai. Hal tersebut menunjukkan skenario ini berhasil dilakukan.

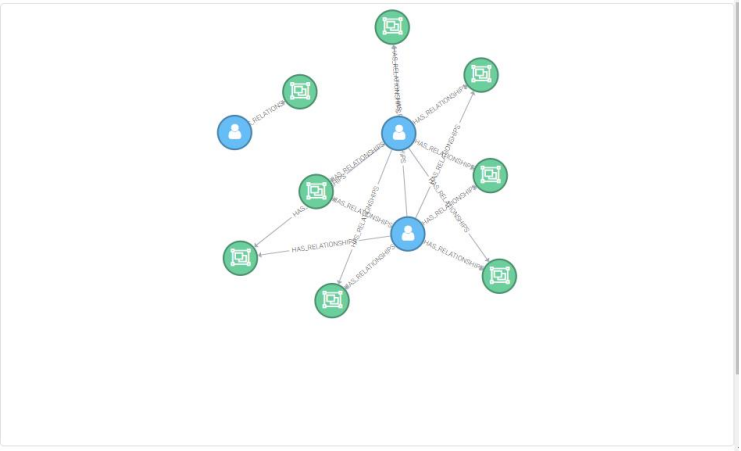



**Gambar 6.7 Hasil Skenario Ketiga Kasus Pengujian A02**

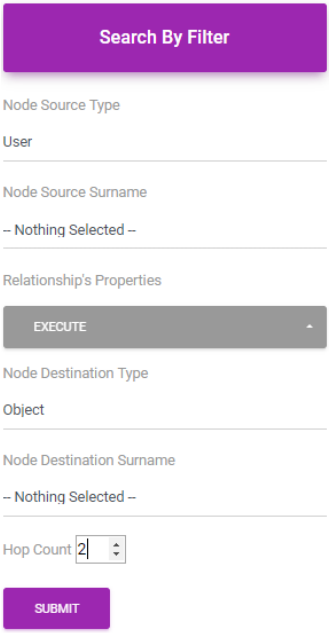



**Gambar 6.8 Filter Skenario Keempat Kasus Pengujian A02**

Skenario keempat merupakan skenario dengan filter tipe *node* asal dan tujuan beserta atribut relasi yang terdapat pada relasi antar kedua tipe *node* tersebut. Hasil yang diharapkan pada skenario ini adalah tampilan *graph* yang hanya memiliki *node* dengan tipe User dan Object pada satu lompatan relasi. Filter yang digunakan pada skenario ini terdapat pada Gambar 6.8. Hasil skenario keempat ditunjukkan pada Gambar 6.9 berikut. Hasil skenario tersebut sesuai dengan hasil yang diharapkan, sehingga skenario ini berhasil dilakukan.

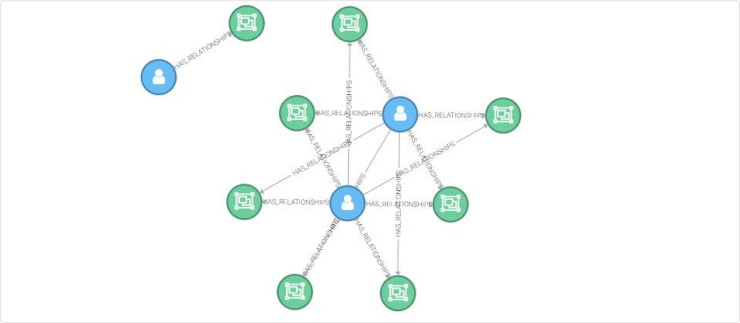


**Gambar 6.9 Hasil Skenario Keempat Kasus Pengujian A02**

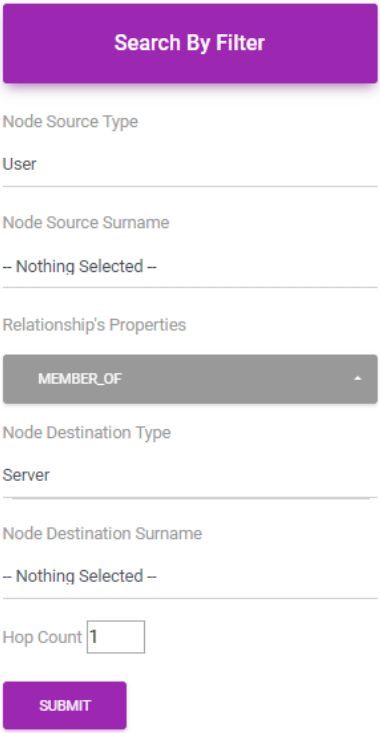



**Gambar 6.10 Filter Skenario Kelima Kasus Pengujian A02**

Skenario kelima merupakan skenario filter yang sama dengan skenario sebelumnya atau skenario keempat. Perbedaan yang mendasar pada skenario kelima dan keempat hanya terletak pada jumlah *hop* yang digunakan untuk filter dan hasil yang diharapkan. Skenario ini menggunakan dua jumlah *hop*. Filter yang digunakan pada skenario ini terdapat pada Gambar 6.10. Hasil dari skenario kelima ditampakkan pada Gambar 6.11 berikut. Hasil yang ditampakkan pada Gambar 6.11 berbeda dengan hasil yang diharapkan, tetapi hasil tersebut sama dengan hasil pada Gambar 6.9. Hal tersebut disebabkan oleh data atau sifat dari data tersebut dan tidak dari fungsi filter aplikasi. Dalam basis data Neo4j, lompatan hubungan setelah *node* yang berlabel User ke Object tidak ditemukan. Hal tersebut yang menjadi alasan bahwa skenario ini berhasil dilakukan.



**Gambar 6.11 Hasil Skenario Kelima Kasus Pengujian A02**




**Gambar 6.12 Filter Skenario Keenam Kasus Pengujian A02**

Skenario keenam merupakan skenario yang terakhir dalam kasus pengujian A02. Skenario ini menggunakan filter tipe *node* asal dan tujuan beserta atribut relasi yang tidak terdapat pada relasi antar kedua *node* tersebut. Hasil yang diharapkan pada skenario ini merupakan tampilan *graph* yang kosong. Hal ini diakibatkan oleh tidak ada data *node* dan relasi yang dikembalikan dari basis data Neo4j. Fitler yang digunakan dalam skenario ini terdapat pada Gambar 6.12. Hasil skenario terakhir ini terdapat pada Gambar 6.13 berikut. Hasil skenario pada Gambar 6.13 dengan hasil yang diharapkan sesuai, sehingga skenario ini berhasil dilakukan.



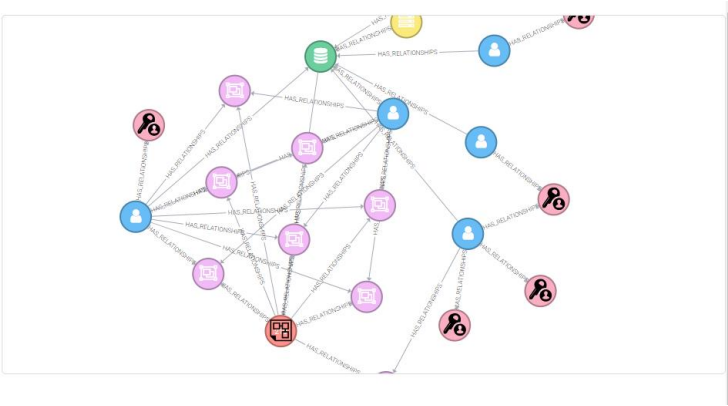
**Gambar 6.13 Hasil Skenario Keenam Kasus Pengujian A02**

Keenam skenario yang terdapat pada kasus pengujian A02 telah dijabarkan. Meskipun salah satu skenario tidak memberikan hasil yang sesuai dengan hasil yang diharapkan, hal tersebut tidak diakibatkan oleh segi fungsionalitas, tetapi hal tersebut diakibatkan oleh segi data. Keenam skenario tersebut telah berhasil dilakukan. Hal tersebut menandakan bahwa kasus pengujian A02 telah berhasil dilakukan.

#### 6.2.1.3 Pembahasan Hasil Pengujian A03

Kasus pengujian A03 merupakan kasus yang diuji dengan menampilkan *graph* dari data yang berformat JSON. Hasil dari kasus pengujian ini terdapat pada Gambar 6.14. Data yang disalurkan dari bagian *Visualizer* dalam komponen *User Dependency Tool* berhasil diolah menjadi tampilan *graph* pada

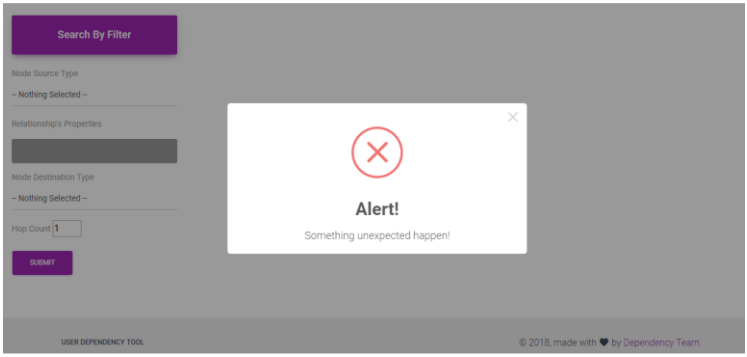
Gambar 6.14. Dalam kasus pengujian ini, *filter* yang digunakan tidak ada, karena kasus pengujian ini lebih berfokus pada tampilan *graph* yang telah dihasilkan. Keterangan – keterangan terkait dengan gambar atau *icon* yang merepresentasikan label dari suatu *node* telah dibahas pada subbab 5.4.2. Tampilan *graph* yang dihasilkan juga berhasil dalam menampilkan seluruh *node* beserta relasinya yang terkait. Hal tersebut bermaksud bahwa tidak ada *node* yang tidak memiliki relasi pada Gambar 6.14.



**Gambar 6.14 Hasil Kasus Pengujian A03**

### 6.2.1.4 Pembahasan Hasil Pengujian A04

Kasus pengujian A04 merupakan kasus yang diuji dengan menampilkan pesan eror saat terjadi eror dalam proses filter. Hasil dari kasus pengujian A04 terdapat pada Gambar 6.15 berikut. Dalam Gambar 6.15, eror ditampilkan dengan kotak peringatan beserta dengan pesannya. Pembahasan pada subbab 5.4.2 menjelaskan pesan yang ditampilkan pada kotak peringatan eror, seperti “Something unexpected happen!” pada Gambar 6.15.




**Gambar 6.15 Hasil Kasus Pengujian A04**

### 6.2.1.5 Pembahasan Hasil Pengujian B01

Kasus pengujian B01 merupakan kasus yang diuji dengan mengubah data *graph* Neo4j ke format JSON. Hasil kasus pengujian B01 terdapat pada Gambar 6.16 berikut. Hasil tersebut menampakkan data *graph* yang berisi id, label, dan properti *node* beserta id, tipe, dan label relasi. Gambar 6.16 menyatakan bahwa aplikasi berhasil mendapatkan data *graph* dalam format JSON Neo4jd3.


```
"This is neo4j data in JSON format: "

"{"results":[{"data":[{"graph":{"nodes":[{"id":11,"labels":
["User"],"properties":{"name":"dbo","type":"SQL User","surname":"WINDOWS-
1K4UUD7.resits.dbo"}},{"id":216,"labels":["Database"],"properties":
{"name":"resits","surname":"WINDOWS-1K4UUD7.resits"}},{"id":274,"labels":
["Role"],"properties":{"name":"db_owner","surname":"WINDOWS-
1K4UUD7.resits.db_owner"}},{"id":35,"labels":["User"],"properties":
{"name":"resits","type":"SQL User","surname":"WINDOWS-
1K4UUD7.resits.resits"}},{"id":33,"labels":["Role"],"properties":
{"name":"db_datareader","surname":"WINDOWS-1K4UUD7.resits.db_datareader"}},
{"id":36,"labels":["Role"],"properties":
{"name":"db_datawriter","surname":"WINDOWS-1K4UUD7.resits.db_datawriter"}},
{"id":118,"labels":["Role"],"properties":
{"name":"web_app","surname":"WINDOWS-1K4UUD7.resits.web_app"}},
{"id":211,"labels":["Object"],"properties":
```


**Gambar 6.16 Hasil Kasus Pengujian B01**

### 6.2.1.6 Pembahasan Hasil Pengujian C01

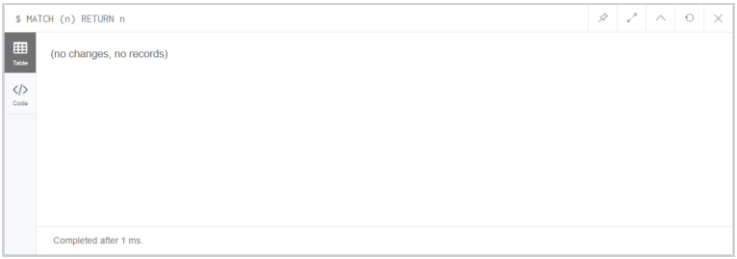
Kasus pengujian C01 merupakan kasus yang diuji dengan menghapus seluruh *node* dan relasi dalam graph dengan

CypherQuery. Hasil kasus pengujian C01 dalam aplikasi ditunjukkan pada Gambar 6.17. Hasil pengujian tersebut menunjukkan pesan atau informasi yang ditampilkan dari aplikasi bahwa seluruh *node* dan relasi – relasinya dalam *graph* pada basis data Neo4j telah dihapus.

```
"All neo4j nodes and relationships has been
deleted successfully."
```

**Gambar 6.17 Hasil Kasus Pengujian C01 dalam Aplikasi User Dependency Tool**

Informasi penghapusan *node* dan relasi – relasinya dalam basis data yang ditampilkan oleh aplikasi divalidasi terlebih dahulu. Validasi dilakukan pada basis data Neo4j dengan menggunakan Neo4j Browser. Hasil dari CypherQuery untuk menampilkan seluruh *node* atau *graph* yang berisi *node* dan relasi – relasinya menunjukkan bahwa tidak ada yang ditampilkan atau *node* yang dikembalikan. Hasil tersebut ditunjukkan pada Gambar 6.18 berikut ini. Hal tersebut menunjukkan bahwa pengujian kasus C01 berhasil dilakukan.




**Gambar 6.18 Pengecekan Hasil Kasus Pengujian C01**

### 6.2.1.7 Pembahasan Hasil Pengujian C02

Kasus pengujian C02 merupakan kasus yang diuji dengan mengirim T-SQL *query* untuk mendapatkan nama *server*. Hasil kasus pengujian C02 tersebut terdapat pada Gambar 6.19 berikut. Gambar 6.19 tersebut menunjukkan informasi yang ditampilkan oleh aplikasi. Informasi yang ditampilkan

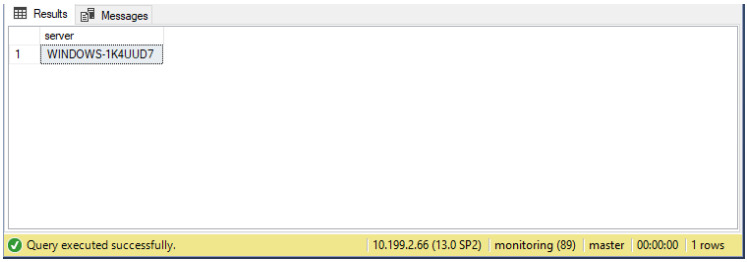
merupakan nama *server* SQL Server yang telah diinisiasikan koneksinya oleh aplikasi. Informasi yang ditampilkan pada Gambar 6.19 bergantung pada konfigurasi koneksi sesuai bahasan pada subbab 5.2.

```
"This is the server name of SQL Server:"

"WINDOWS-1K4UUD7"
```

**Gambar 6.19 Hasil Kasus Pengujian C02**

Informasi yang terdapat pada Gambar 6.19 divalidasi langsung ke basis data yang terdapat konfigurasi aplikasi. Validasi dilakukan dengan melakukan eksekusi *query* terhadap basis data SQL Server melalui SSMS (SQL Server Management Studio). Hasil dari eksekusi *query* untuk mendapatkan nama *server* ditampakkan pada Gambar 6.20 berikut ini. Hasil yang ditampakkan pada SSMS dan aplikasi sama, sehingga kasus pengujian C02 ini berhasil dilakukan.

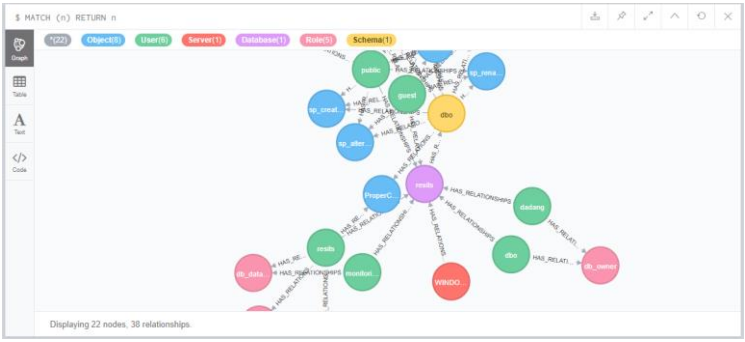



**Gambar 6.20 Pengecekan Hasil Kasus Pengujian C02**

### 6.2.1.8 Pembahasan Hasil Pengujian C03

Kasus pengujian C03 merupakan kasus yang diuji dengan membuat *node* dan relasinya dengan menggunakan data pada *result object* yang didapatkan dari D02 dan CypherQuery. Hasil kasus pengujian tersebut ditampilkan pada Gambar 6.21 berikut ini. Pengujian dilakukan secara langsung pada Neo4j Browser

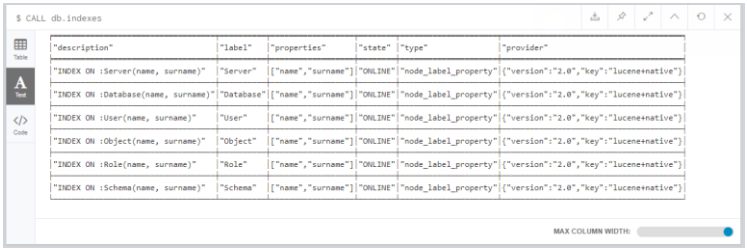
dengan menggunakan CypherQuery “MATCH (n) RETURN n”. CypherQuery tersebut bertujuan untuk menampilkan *graph* pada mode *graph* dan menampilkan seluruh *node* pada mode tabel. Hasil kasus pengujian yang ditampilkan pada Gambar 6.21 menunjukkan 22 *node* dan 38 relasi – relasinya dalam basis data Neo4j. Gambar 6.21 membuktikan bahwa *node* dan relasi – relasinya yang dibuat dari aplikasi dengan menggunakan CypherQuery dan data yang diekstraksi berhasil dilakukan.




**Gambar 6.21 Hasil Kasus Pengujian C03**

### 6.2.1.9 Pembahasan Hasil Pengujian C04

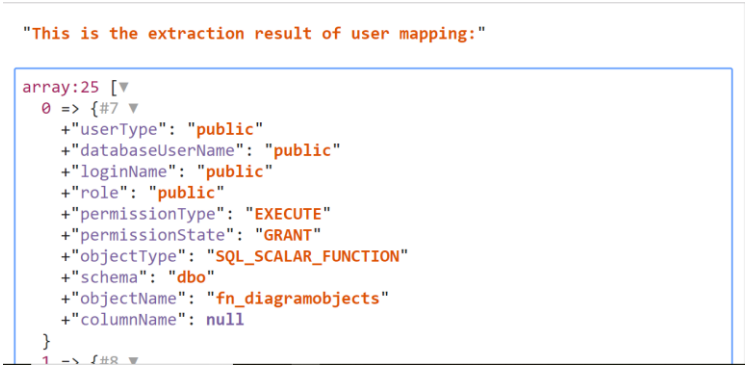
Kasus pengujian C04 merupakan kasus yang diuji dengan menambahkan indeks pada seluruh label *node* untuk properti name dan surname dengan CypherQuery. Hasil kasus pengujian C04 ditunjukkan pada Gambar 6.22 berikut. Hasil tersebut didapatkan dari penggunaan CypherQuery pada Neo4j Browser untuk melihat seluruh indeks dalam basis data Neo4j. CypherQuery yang digunakan berupa “CALL db.indexes”. Indeks yang ditampilkan merupakan indeks untuk *node* yang berlabel Server, Database, User, Object, Role, dan Schema dengan atribut gabungan name dan surname. Indeks untuk *node* yang berlabel Column tidak ditemukan, karena *node* yang berlabel Column tidak ada pada basis data Neo4j. Indeks yang dibuat hanya untuk *node* yang terbentuk saja. Kasus pengujian C04 berdasarkan pembahasan tersebut telah berhasil dilakukan.




**Gambar 6.22 Hasil Kasus Pengujian C04**

### 6.2.1.10 Pembahasan Hasil Pengujian D01

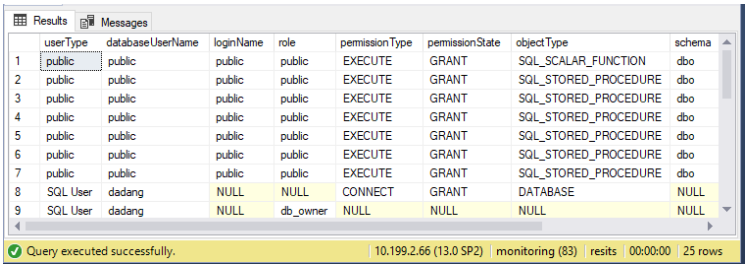
Kasus pengujian D01 merupakan kasus yang diuji dengan mengirim T-SQL *query* untuk mendapatkan pemetaan pengguna beserta peran dan hak aksesnya dengan objek basis data. Hasil kasus pengujian D01 terdapat pada Gambar 6.23 berikut. Hasil kasus pengujian tersebut menunjukkan data ekstraksi pengguna beserta peran dan hak aksesnya dengan objek basis data berhasil didadapatkan dari basis data SQL Server. Data – data yang didapatkan berupa tipe pengguna, pengguna, nama *login* pengguna, peran, tipe hak akses, status hak akses, tipe objek, skema, nama objek, dan nama kolom.




**Gambar 6.23 Hasil Kasus Pengujian D01**

Gambar 6.23 menunjukkan hasil ekstraksi yang diterima oleh aplikasi. Hasil tersebut perlu divalidasikan dengan basis data SQL Server. Eksekusi *query* untuk mendapatkan ekstraksi yang

sama perlu dilakukan dengan menggunakan SSMS dan *query* yang sama. Hasil dari eksekusi *query* tersebut ditampakkan pada Gambar 6.24. Gambar 6.24 menunjukkan hasil *query* dengan nama kolom dan jumlah baris yang sama terhadap Gambar 6.23. Hal tersebut menandakan bahwa kasus pengujian D01 berhasil dilakukan.



Results | Messages

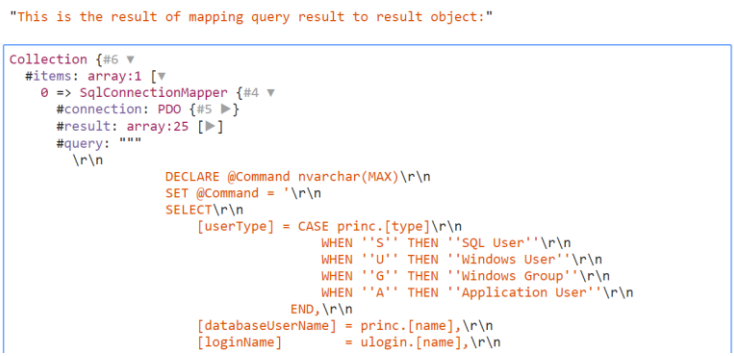
| | userType | databaseUserName | loginName | role | permissionType | permissionState | objectType | schema |
|---|---|---|---|---|---|---|---|---|
| 1 | public | public | public | public | EXECUTE | GRANT | SQL_SCALAR_FUNCTION | dbo |
| 2 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 3 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 4 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 5 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 6 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 7 | public | public | public | public | EXECUTE | GRANT | SQL_STORED_PROCEDURE | dbo |
| 8 | SQL User | dadang | NULL | NULL | CONNECT | GRANT | DATABASE | NULL |
| 9 | SQL User | dadang | NULL | db_owner | NULL | NULL | NULL | NULL |

Query executed successfully. | 10.199.2.66 (13.0 SP2) | monitoring (83) | resits | 00:00:00 | 25 rows

**Gambar 6.24 Pengecekan Hasil Kasus Pengujian D01**

### 6.2.1.11 Pembahasan Hasil Pengujian D02

Kasus pengujian D02 merupakan kasus yang diuji dengan memetakan hasil *query* dari D01 ke dalam *result object*. Hasil dari kasus pengujian D02 ditampakkan pada Gambar 6.25 berikut. Gambar 6.25 menunjukkan pemetaan hasil *query* dari D01 pada kelas SqlConnectionMapper. Hasil *query* tersebut terdapat pada atribut result dalam kelas SqlConnectionMapper sesuai dengan Gambar 6.25. Jumlah elemen yang terdapat dalam *array* pada atribut result sesuai dengan jumlah elemen pada Gambar 6.23 dan jumlah baris pada Gambar 6.24. Hal tersebut menandakan bahwa kasus pengujian D02 telah berhasil dilakukan.




**Gambar 6.25 Hasil Kasus Pengujian D02**

### 6.2.2 Pembahasan Hasil Pengujian Performa

Pembahasan hasil pengujian performa berfokus pada kedua bagian dalam komponen *User Dependency Tool*. Kedua bagian tersebut merupakan bagian *Processor* dan *Extractor*. Pengujian pada masing – masing bagian dilakukan selama enam kali percobaan dan hasil pengujian didapatkan dari rata – rata seluruh percobaan tersebut. Hasil pengujian berupa rata – rata dan hasil tiap percobaannya telah dibahas pada subbab 6.1.1. Informasi yang dihasilkan dari tiap percobaan untuk bagian *Processor* ditunjukkan pada Gambar 6.26. Informasi yang dihasilkan dari tiap percobaan untuk bagian *Extractor* ditunjukkan pada Gambar 6.27 berikut. Kedua informasi tersebut mengandung waktu eksekusi untuk masing – masing bagian tiap percobaan.

```
"The execution time for processing:"

1.6770081520081

"SUCCEDED"
```

**Gambar 6.26 Informasi Waktu Eksekusi Bagian Processor**

```
"The execution time for extraction:"

0.24756717681885
```

**Gambar 6.27 Informasi Waktu Eksekusi Bagian Extractor**

*(Halaman ini sengaja dikosongkan)*

# BAB VII
# KESIMPULAN DAN SARAN

Bab ini menjabarkan kesimpulan dari implementasi dan hasil implementasi tugas akhir dan saran untuk pengembangan atau implementasi tugas akhir selanjutnya.

## 7.1 Kesimpulan

Subbab ini menyertakan beberapa kesimpulan yang dapat diambil dari implementasi dan hasil implementasi tugas akhir ini. Kesimpulan yang diambil tersebut mengacu pada tujuan tugas akhir ini yang telah didefinisikan pada subbab 1.4. Berikut ini merupakan kesimpulan dari pengerjaan tugas akhir ini:

1. Aplikasi berhasil melakukan ekstraksi metadata objek dan pengguna beserta peran dan hak aksesnya dari MS SQL Server. Kasus pengujian D01 dan D02 mendukung kesimpulan ekstraksi ini.
2. Aplikasi berhasil mengubah hasil ekstraksi pada poin 1 menjadi bentuk graph di Neo4j. Kasus pengujian B01 dan C03 mendukung kesimpulan pengubahan hasil ektraksi ini.
3. Aplikasi berhasil menampilkan graph ke pengguna dengan platform web. Kasus pengujian A03 mendukung kesimpulan penampilan *graph* ini.
4. Aplikasi berhasil memfilter visualisasi berdasarkan relasi, group, atau kategori lainnya. Kasus pengujian A01, A02, dan A04 mendukung kesimpulan proses filter ini.

## 7.2 Saran

Saran yang dapat diberikan oleh penulis pada implementasi atau pengerjaan tugas akhir ini adalah sebagai berikut:

1. Komponen *Visualisation* sebaiknya ditambahkan fitur filter dengan menggunakan CypherQuery.

2. Tampilan *graph* sebaiknya menggunakan Neo4j Browser yang dikustomisasi, karena Neo4jd3 memiliki *bug* pada relasi yang saling tumpang tindih apabila dua *node* memiliki lebih dari satu relasi.
3. Apabila aplikasi ini menjadi semakin kompleks dengan kebutuhan yang banyak, maka aplikasi ini perlu menggunakan *framework*, seperti CI, Laravel, Yii2, dan *framework* lainnya.
4. Aplikasi ini perlu dibuatkan API (Application Programming Interface) apabila aplikasi ini akan diintegrasikan dengan lingkungan atau aplikasi eksternal.
5. Aplikasi ini perlu dikembangkan fitur pemantauan untuk lamanya waktu eksekusi *query* ke SQL Server pada bagian *Processor* dan *Extractor* dalam komponen *User Dependency Tool*.
6. Aplikasi ini perlu dikembangkan fitur CDC (Change Data Capture) sehingga eksekusi *query* untuk ekstraksi tidak mencakup keseluruhan data tetapi ekstraksi hanya mencakup data – data yang berubah saja.

# DAFTAR PUSTAKA


[1] W. R. King, "Knowledge management and organizational learning," in *Knowledge management and organizational learning*, Springer, 2009, pp. 3–13.

[2] E. Macauley, D. Laudenschlager, and C. Guyer, "Securing SQL Server," *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/securing-sql-server. [Accessed: 09-Feb-2018].

[3] H. F. Ledgard, M. D. Good, J. a Whiteside, D. R. Wixon, and S. J. Jones, "Building a User . Derived Interface," *Hum. Factors*, vol. 27, no. 10, pp. 1032–1043, 1984.

[4] ApexSQL software, *An introduction to ApexSQL Audit*. YouTube, 2017.

[5] C. S. Mullins, "The Importance of the Relational System Catalog," *Database Trends and Applications*, 2017. [Online]. Available: http://www.dbta.com/Columns/DBA-Corner/The-Importance-of-the-Relational-System-Catalog-116713.aspx. [Accessed: 23-Feb-2018].

[6] The SciDB Development Team, "Overview of SciDB - Large Scale Array Storage , Processing and Analysis," *SIGMOD '10 Proc. 2010 ACM SIGMOD Int. Conf. Manag. data*, pp. 963–968, 2010.

[7] A. Thusoo *et al.*, "Hive - A Warehousing Solution Over a Map-Reduce Framework," *Sort*, vol. 2, pp. 1626–1629, 2009.

[8] J. F. Hornibrook, J. E. Lumby, W. Rjaibi, and C. P. Zuzarte, "System and process for evaluating the performance of a database system." Google Patents, 2003.

[9] W.-S. Han, K.-H. Lee, and B. S. Lee, “An XML storage system for object-oriented/object-relational DBMSs,” *J. Object Technol.*, vol. 2, no. 3, pp. 113–126, 2003.

[10] E. Macauley and C. Guyer, “System Catalog Views (Transact-SQL),” *Microsoft*, 2016. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/catalog-views-transact-sql. [Accessed: 23-Feb-2018].

[11] R. Sheldon, “SQL Server System Views: The Basics,” *Redgate Hub*, 2016. [Online]. Available: https://www.red-gate.com/simple-talk/sql/learn-sql-server/sql-server-system-views-the-basics/. [Accessed: 23-Feb-2018].

[12] J. Sack, “Securables and Permissions,” *SQL Serv. 2005 T-SQL Recipes A Probl. Approach*, pp. 433–458, 2006.

[13] J. Sack, “Securables, Permissions, and Auditing,” *SQL Serv. 2008 Trans. Recipes*, pp. 501–545, 2008.

[14] R. Dobson, “Managing SQL Server Express Security,” *Begin. SQL Serv. 2005 Express Database Appl. with Vis. Basic Express Vis. Web Dev. Express From Novice to Prof.*, pp. 307–368, 2006.

[15] K. Simmons and S. Carstarphen, “Managing Security Within the Database Engine,” *Pro SQL Serv. 2008 Adm.*, pp. 209–249, 2009.

[16] E. Bertino, D. Bruschi, S. Franzoni, I. Nai-Fovino, and S. Valtolina, “Threat modelling for SQL servers: Designing a secure database in a web application,” *IFIP Adv. Inf. Commun. Technol.*, vol. 175, 2005.

[17] a. L. Pereira, V. Muppavarapu, and S. M. Chung, “Role-Based Access Control for Grid Database Services Using the Community Authorization Service,” *IEEE Trans. Dependable Secur. Comput.*, vol. 3, no. 2, pp. 156–166, 2006.

[18] L. Notargiacomo, “Role-based access control in ORACLE7 and Trusted ORACLE7,” in *Proceedings of the first ACM Workshop on Role-based access control*, 1996, p. 17.

[19] G. Buhle and R. R. Wessman, “Authentication and authorization in a multi-tier relational database management system.” Google Patents, 2001.

[20] Y. Laura-Silva and W. G. Aref, “Realizing Privacy-Preserving Features in Hippocratic Databases,” in *2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007, pp. 198–206.

[21] R. A. Kagalwala and J. P. Thompson, “Representing database permissions as associations in computer schema.” Google Patents, 2004.

[22] D. Laudenschlager, C. Guyer, M. Jones, and M. Hoffman, “Server and Database Roles in SQL Server,” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/server-and-database-roles-in-sql-server. [Accessed: 24-Feb-2018].

[23] S. Harkins, “Understanding roles in SQL Server security,” *TechRepublic*, 2004. [Online]. Available: https://www.techrepublic.com/article/understanding-roles-in-sql-server-security/. [Accessed: 24-Feb-2018].

[24] E. Macauley, D. Laudenschlager, and C. Guyer, “Database-Level Roles,” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles. [Accessed: 24-Feb-2018].

[25] E. Macauley and C. Guyer, “Application Roles,” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/application-roles. [Accessed: 24-Feb-2018].

[26] F. Chong, G. Carraro, and R. Wolter, “Multi-tenant data architecture,” *MSDN Libr. Microsoft Corp.*, pp. 14–30, 2006.

[27] P. Youn, “Method and apparatus for facilitating role-based cryptographic key management for a database.” Google Patents, 2011.

[28] B. Mackay and P. Wolpe, “System and method for controlling the release of updates to a database configuration.” Google Patents, 2008.

[29] S. Osborn, R. Sandhu, and Q. Munawer, “Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 85–106, May 2000.

[30] T. Priebe and G. Pernul, “Towards OLAP Security Design - Survey and Research Issues,” in *Proceedings of the 3rd ACM International Workshop on Data Warehousing and OLAP*, 2000, pp. 33–40.

[31] C. L. Sweeney, S. A. Stodghill, K. A. DeShazer, and A. Marimuthu, “Application and database security and integrity system and method.” Google Patents, 1999.

[32] D. Laudenschlager, C. Guyer, M. Jones, and M. Hoffman, “Authorization and Permissions in SQL Server,” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/authorization-and-permissions-in-sql-server. [Accessed: 25-Feb-2018].

[33] E. Macauley and C. Guyer, “Permissions (Database Engine),” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/permissions-database-engine. [Accessed: 25-Feb-2018].

[34] M. J. Melone, “Access Privilege Analysis for a Securable Asset.” Google Patents, 2017.

[35] E. Macauley and C. Guyer, “Securables,” *Microsoft*, 2016. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/securables. [Accessed: 25-Feb-2018].

[36] R. Sheldon, “SQL Server Access Control: The Basics,” *Redgate Hub*, 2016. [Online]. Available: https://www.red-gate.com/simple-talk/sql/database-administration/sql-server-access-control-basics/. [Accessed: 25-Feb-2018].

[37] M. Rouse, A. Hughes, and C. Stedman, “Microsoft SQL Server,” *TechTarget*, 2017. [Online]. Available: http://searchsqlserver.techtarget.com/definition/SQL-Server. [Accessed: 25-Feb-2018].

[38] D. Schlichting, “What is SQL Server,” *Database Journal*, 2008. [Online]. Available: https://www.databasejournal.com/features/mssql/article.php/3769211/What-is-SQL-Server.htm. [Accessed: 25-Feb-2018].

[39] C. Guyer, J. Roth, M. B, A. Pasic, and E. Asner, “SQL Server Documentation,” *Microsoft*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation. [Accessed: 25-Feb-2018].

[40] K. Schmid, M. Sevenich, S. Hong, and H. Chafi, “Graph database system that dynamically compiles and executes custom graph analytic programs written in high-level, imperative programing language.” Google Patents, 2016.

[41] A. C. Allison, M. P. Atkinson, S. L. PeytonJones, and C. J. van Rijsbergen, “User Interfaces to Data Modelling Systems,” 1994.

[42] P. Klint, T. v. d. Storm, and J. Vinju, “RASCAL: A Domain Specific Language for Source Code Analysis

and Manipulation," in *2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, 2009, pp. 168–177.

[43] R. Angles and C. Gutierrez, "Survey of Graph Database Models," *ACM Comput. Surv.*, vol. 40, no. 1, p. 1:1--1:39, Feb. 2008.

[44] I. Robinson, J. Webber, and E. Eifrem, *Graph databases*. " O'Reilly Media, Inc.," 2013.

[45] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective," in *Proceedings of the 48th Annual Southeast Regional Conference*, 2010, p. 42:1--42:6.

[46] J. Webber, "A Programmatic Introduction to Neo4J," in *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, 2012, pp. 217–218.

[47] F. Holzschuher and R. Peinl, "Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4J," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 2013, pp. 195–204.

[48] J. J. Miller, "Graph database applications and concepts with Neo4j," in *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, 2013, vol. 2324, p. 36.

[49] G. Drakopoulos, A. Baroutiadi, and V. Megalooikonomou, "Higher order graph centrality measures for Neo4j," in *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, 2015, pp. 1–6.

[50] H. Huang and Z. Dong, "Research on architecture and query performance based on distributed graph database Neo4j," in *2013 3rd International Conference on*

*Consumer Electronics, Communications and Networks*, 2013, pp. 533–536.

[51] S. Batra and C. Tyagi, “Comparative analysis of relational and graph databases,” *Int. J. Soft Comput. Eng.*, vol. 2, no. 2, pp. 509–512, 2012.

[52] M. Bostock, V. Ogievetsky, and J. Heer, “D #x0B3; Data-Driven Documents,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[53] A. Jain, “Data Visualization with the D3.JS Javascript Library,” *J. Comput. Sci. Coll.*, vol. 30, no. 2, pp. 139–141, Dec. 2014.

[54] K. Ono, B. Demchak, and T. Ideker, “Cytoscape tools for the web age: D3.js and Cytoscape.js exporters,” *F1000Research*, vol. 3. London, UK, 2014.

[55] F. Bao and J. Chen, “Visual framework for big data in d3.js,” in *2014 IEEE Workshop on Electronics, Computer and Applications*, 2014, pp. 47–50.

[56] S. Lee, J. Y. Jo, and Y. Kim, “Performance testing of web-based data visualization,” in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 1648–1653.

*(Halaman ini sengaja dikosongkan)*

## BIODATA PENULIS

Penulis lahir di Surabaya pada tanggal 18 September 1996. Penulis merupakan anak pertama dari dua bersaudara. Penulis telah menempuh pendidikan formal untuk jenjang SD (Sekolah Dasar) di SD Ta'miriyah Surabaya. Penulis juga telah menempuh pendidikan formal di sekolah negeri mulai dari SMPN 1 Surabaya dan SMAN 2 Surabaya. Setelah lulus, penulis melanjutkan ke jenjang perguruan tinggi negeri di Surabaya, yakni Jurusan Sistem Informasi Institut Teknologi Sepuluh Nopember Surabaya. Penulis menjadi mahasiswa aktif dalam kepanitiaan di Jurusan, seperti ISE, dan keanggotaan di tingkat UKM. Penulis menjadi anggota UKM Shorinji Kempo dari semester 2 hingga semester 5. Penulis juga pernah melakukan kerja praktik di Departemen Sistem Informasi PDAM Surya Sembada Surabaya selama 1,5 bulan pada tahun 2017. Penulis mengambil laboratorium ADDI (Akuisisi Data dan Diseminasi Informasi) sebagai bidang minat untuk tugas akhir. Penulis memiliki minat pada topik Visualisasi dalam laboratorium tersebut. Untuk kepentingan tertentu, penulis juga menyertakan alamat email sebagai kontak, yaitu hafizludyanto13@gmail.com.

*(Halaman ini sengaja dikosongkan)*

# LAMPIRAN A