TUGAS AKHIR – KS141501

RANCANG BANGUN APLIKASI STORED PROCEDURE DEPENDENCY TOOL UNTUK DATABASE SQL SERVER BERBASIS *GRAPH* NEO4J

GRAPH-BASED STORED PROCEDURE DEPENDENCY TOOL APPLICATION FOR SQL SERVER DATABASE

Aprilia Rizki Rahmawati
NRP 05211440000071

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 STORED PROCEDURE DEPENDENCY TOOL UNTUK DATABASE SQL SERVER BERBASIS *GRAPH* NEO4J

Aprilia Rizki Rahmawati
NRP 05211440000071

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

GRAPH-BASED STORED PROCEDURE DEPENDENCY TOOL APPLICATION FOR SQL SERVER DATABASE

Aprilia Rizki Rahmawati
NRP 05211440000071

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

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

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI STORED PROCEDURE DEPENDENCY TOOL UNTUK DATABASE 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:

APRILIA RIZKI RAHMAWATI
NRP. 05211440000071

Surabaya, Juli 2018

KEPALA
DEPARTEMEN SISTEM INFORMASI

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

LEMBAR PERSETUJUAN

RANCANG BANGUN APLIKASI STORED PROCEDURE DEPENDENCY TOOL UNTUK DATABASE 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 :

APRILIA RIZKI RAHMAWATI
NRP. 05211440000071

Disetujui Tim Penguji : Tanggal Ujian : 9 Juli 2018
Periode Wisuda : September 2018

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

Nur Aini R., S.Kom., M.Sc.Eng., Ph.D **(Penguji I)**

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

RANCANG BANGUN APLIKASI STORED PROCEDURE DEPENDENCY TOOL UNTUK DATABASE SQL SERVER BERBASIS *GRAPH* NEO4J

Nama Mahasiswa	**:**	**Aprilia Rizki Rahmawati**
NRP	**:**	**05211440000071**
Departemen	**:**	**Sistem Informasi**
Dosen Pembimbing	**:**	**Radityo Prasetianto Wibowo, S.Kom., M.Kom.**

ABSTRAK

Dokumentasi sistem perangkat lunak yang konsisten, benar, dan lengkap merupakan hal yang penting bagi pengelola untuk mempermudah pengembangan selanjutnya. Hal ini sangat memudahkan pemahaman dan komunikasi perangkat lunak, memudahkan proses pembelajaran, membuat sistem perangkat lunak menjadi lebih mudah dipelihara, dan akibatnya dapat meningkatkan produktivitas dan kualitas kerja pengelola. Tetapi sebagian besar perusahaan cenderung kurang peduli pada kelayakan dokumentasi sistem.

Dokumentasi SQL sebenarnya dapat diatasi dengan menggunakan aplikasi seperti DBVisualier, RedGate Dependency Tracker, ApexSQL, atau aplikasi yang sejenis. Namun, aplikasi-aplikasi tersebut memiliki keterbatasan fitur dalam menampilkan dependency dari stored procedure dengan objek database lainnya serta bagaimana hubungan interaksi yang ada. Padahal, penggunaan stored procedure tidak dapat dihindari dalam pengelolaan basis data dan seringkali sulit untuk mengetahui dependency dan hubungan interaksinya dengan objek basis data lainnya.

Dengan melihat permasalahan yang ada, dibuatlah aplikasi Stored Procedure Dependency tool yang dapat mempermudah mengetahui hubungan stored procedure dengan objek database lain. Pengembangan aplikasi ini digunnakan untuk database berbasis Microsoft SQL Server. Aplikasi ini juga menggunakan prinsip graph database serta memanfaatkan Neo4J sebagai penyimpanan utamanya.

Hasil dari penelitian ini adalah aplikasi Stored Procedure Dependency Tool berbasis web yang dapat menampilkan graph untuk mempermudah dalam mengetahui hubungan stored procedure dengan objek database lain. Aplikasi memiliki fitur filter berdasarkan relasi, stored procedure, dan radius.

Kata kunci : dokumentasi, parsing, graph database, sql server, visualisasi, neo4j, stored procedure

GRAPH-BASED STORED PROCEDURE DEPENDENCY TOOL APPLICATION FOR SQL SERVER DATABASE

Name	**:**	**Aprilia Rizki Rahmawati**
NRP	**:**	**05211440000071**
Department	**:**	**Sistem Informasi**
Supervisor	**:**	**Radityo Prasetianto Wibowo, S.Kom., M.Kom.**

ABSTRACT

The consistent, correct, and complete documentation of the software system is important for managers to facilitate further development. It greatly facilitates the understanding and communication of software, facilitates the learning process, makes software systems more maintainable, and consequently increases productivity and quality of work managers. But most companies tend to be less concerned with the feasibility of system documentation.

The actual SQL documentation can be solved by using applications such as DBVisualier, RedGate Dependency Tracker, ApexSQL, or similar applications. However, these applications have limited features in displaying dependencies of stored procedures with other database objects and how they interact. In fact, the use of stored procedures can not be avoided in database management and it is often difficult to know the dependencies and their interaction relationships with other database objects.

By looking at the existing problems, made the application Stored Procedure Dependency tool that can facilitate to know the relation of stored procedure with other database object. This application development is used for Microsoft SQL Server

based database. This application also uses the principle graph database and utilize Neo4J as its main storage.

The result of this research is Web-Based application Stored Procedure Dependency Tool which can display graph to simplify in knowing relation of stored procedure with other database object. The app has filter features based on relation, stored procedure, and radius.

Keywords : documentation, parsing, graph database, sql server, visualization, neo4j, stored procedure

KATA PENGANTAR

Segala puji dan syukur pada Allah SWT yang telah melimpahkan ramat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "Rancang Bangun Aplikasi Stored Procedure Dependency Tool untuk Database SQL Server berbasis Graph Neo4J" dengan tepat waktu.

Harapan dari penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi nyata bagi departemen Sistem Informasi, ITS, dan Indonesia.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan yang diterima oleh penulis dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin menyampaikan terimakasih kepada:

1. Kedua orangtua penulis atas dukungan mereka berupa material, moral, dan doa yang tidak pernah putus demi kelancaran dan tercapainya harapan penulis.
2. Bapak Radityo Prasetianto Wibowo, S.Kom, M.Kom. selaku dosen pembimbing yang senantiasa meluangkan waktu, memberikan ilmu dan petunjuk, serta memotivasi untuk pengerjaan tugas akhir penulis.
3. Bapak Faizal Johan Atletiko S.Kom, M.T. selaku dosen wali dan dosen penguji bersama dengan ibu Nur Aini Rakhmawati S.Kom., M.Sc.Eng dalam memberi kritik dan saran untuk pengerjaan tugas akhir penulis.
4. Seluruh dosen Departemen Sistem Informasi ITS yang telah memberikan ilmu dan pengalaman yang sangat bermanfaat dan berharga bagi penulis.
5. Teman-teman penghuni sesi malam yang senantiasa menemani dalam pengerjaan tugas akhir ini.

6. Teman-teman laboratorium ADDI yang senantisa memberikan informasi dan membantu dalam pengerjaan tugas akhir ini.
7. Jwalita, Mutiara, Aldifiati, Anugrah D. Putra, dan Dhimas Dwijo yang telah menemani dan mendukung penulis dalam menjalani kehidupan kampus.
8. Teman-teman OSIRIS yang senantiasa menemani, membantu, dan memberikan motivasi bagi penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir.

Tugas Akhir ini juga masih jauh dari kata sempurna, sehingga penulis mengharapkan saran dan kritik yang membangun dari pembaca untuk perbaikan ke depan. Semga Tugas Akhir ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan semua pihak.

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR KODE

BAB I
PENDAHULUAN

Pada bab pendahuluan ini akan membahas terkait dengan latar belakang masalah, perumusann masalah, Batasan masalah, tujuan penelitina, manfaat penelitian, dan relevansi terhadap pengerjaan tugas akhir.

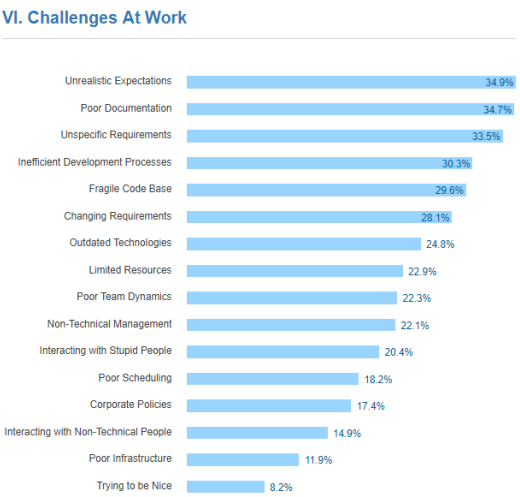
1.1 Latar Belakang

Dokumentasi perangkat lunak dilakukan dengan tujuan untuk mendeskripsikan sistem perangkat lunak dan prosesnya. Dokumentasi sistem perangkat lunak yang konsisten, benar, dan lengkap merupakan hal yang penting bagi pengelola untuk meraih kesuksesan. Ini sangat memudahkan pemahaman dan komunikasi perangkat lunak, memudahkan proses pembelajaran, membuat sistem perangkat lunak menjadi lebih mudah dipelihara, dan akibatnya dapat meningkatkan produktivitas dan kualitas kerja pengelola. Di sisi lain, dokumentasi perangkat lunak yang buruk merupakan alasan utama yang dapat menurunkan kualitas perangkat lunak secara cepat. [1]

Dokumentasi perangkat lunak dibagi menjadi dua, yaitu dokumentasi proses dan dokumentasi produk. Dokumentasi proses adalah pekerjaan dokumentasi yang merekam semua proses pengembangan dan perawatan. Sedangkan dokumentasi produk adalah menggambarkan produk yang sedang dikembangkan. Dokumentasi produk sendiri bisa dilakukan dengan cara *reverse engineering. Reverse Engineering* adalah proses menganalisis subyek sistem untuk mengidentifikasi komponen sistem dan hubunganya satu sama lain serta membuat gambaran suatu sistem kedalam bentuk lain atau level abstraksi yang lebih tinggi. [2]

Sebagian besar perusahaan cenderung kurang peduli pada kelayakan dokumentasi sistem. Berdasarkan survey yang dilakukan oleh StackOverflow yang dilakukan kepada 50.000 pengembang perangkat lunak pada tahun 2016, dokumentasi perangkat lunak yang tidak layak merupakan tantangan kerja

yang banyak dihadapi oleh pengembang perangkat lunak. [3] Bagian sistem informasi Biro Humas dan Hukum Sekretariat Kementrian Pemuda dan Olahraga(Kemenpora) menyatakan bahwa dokumentasi yang minim menyebabkan penggunaan data bersama belum bisa dilakukan sehingga proses sharing masih dilakukan secara manual. [4]



Gambar 1. 1 Hasil survey tantangan kerja berdasarkan Stack Overflow

Didalam sistem informasi, database merupakan komponen penting yang berperan dalam menyediakan data dan informasi untuk pemakai. Penggunaan SQL yang popular menjadikan masalah dokumentasi database SQL menjadi hal yang perlu diperhatikan. Pemahaman atau *knowledge* mengenai database yang tidak terdokumentasi dengan baik akan menjadikanya sebagai *tacit knowledge*. *Tacit Knowledge* adalah pengetahuan yang dihasilkan dari *trial & error* oleh sebagian karyawan dan pengetahuan ini tidak dapat dikelola dengan baik oleh perusahaan karena perusahaan tidak mengetahuinya (O'dell & Grayson, 1998) [5]

Dokumentasi database SQL yang kurang, khususnya SQL Server bisa diatasi dengan menggunakan bantuan aplikasi

visualisasi seperti DBVisualizer, Red-gate: Toolbelt SQL Dependecy Tracker, ApexSQL Analyze, dan aplikasi lainya. Tetapi aplikasi tersebut memiliki kekurangan yaitu tidak dapat memvisualisasikan isi dan relasi yang terdapat pada prosedur atau stored procedure secara spesifik. Padahal stored procedure seringkali digunakan dalam proses pengelolaan database pada banyak kasus database *enterprise*. Sebagai contoh sistem Resource ITS (RESITS) yang memiliki alamat website http://resits.its.ac.id menggunakan banyak stored procedure.

1.2 Perumusan Masalah

Permasalahan yang akan diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana mendapatkan informasi *table, schema, function* pada database SQL Server ?
2. Bagaimana mendapat informasi *Stored Procedure* dalam database SQL Server ?
3. Bagaimana menghubungkan *Stored Procedure* dengan obyek database lainnya ?
4. Bagaimana memvisualisasikan informasi hubungan *database, table, schema, function,* dan *stored procedure* ?

1.3 Batasan Masalah

Batasan masalah didalam pengerjaan tugas akhir ini adalah :

1. Database yang digunakan adalah Microsoft SQL Server
2. Database graph yang digunakan adalah Neo4J yang diinstall pada sistem operasi Windows 10.
3. Hasil keluaran dari tugas akhir ini berupa aplikasi berbasis web yang bisa dijalanakan melalui *browser.*
4. Database yang digunakan adalah database sistem Resource ITS (RESITS)

1.4 Tujuan

Tujuan dari tugas akhir ini adalah membuat aplikasi berbasis web untuk :

1. Mendapatkan informasi mengenai *table*, *schema*, dan *database.*
2. Mendapatkan informasi mengenai seluruh *Stored Procedure* yang terdapat pada database SQL Server.
3. Memvisualisasikan *dependency* yang terdapat pada *Stored Procedure* agar mudah dipahami.

1.5 Manfaat

Manfaat yang diperoleh dari tugas akhir ini antara lain :

1. Memberikan alternative *tool* dokumentasi database yang dapat menyajikan infomasi yang lengkap dan akurat dalam bentuk visualisasi yang menarik dan mudah dimengerti
2. Melengkapi *tool* dokumentasi database dengan menambahkan fitur untuk memetakan *stored procedure* dalam suatu database.

1.6 Relevansi

Topik pada tugas akhir ini adalah mengenai rancang bangun aplikasi visualisasi database sql server berbasis *graph* Neo4j untuk memetakan *dependency* stored procedure terhadap object database.

Tugas akhir ini layak dijadikan sebagai tugas akhir tingkat S1 karena tugas ini mempermudah pengguna untuk mengetahui *dependency* yang terdapat pada *stored procedure* pada database SQL Server. Tugas akhir ini nantinya adalah perangkat lunak yang dirancang untuk bisa digunakan sebagai *tool* untuk visualisasi *dependency* dari stored procedure untuk DBMS SQL Server dengan tampilan yang mudah dipahami.

Selain itu tugas akhir ini berkaitan dengan mata kuliah Analisis dan Desain Perangkat Lunak, Konstruksi dan Pengujian Perangkat Lunak, Interaksi Manusia dan Komputer, serta

Manajemen dan Administrasi Basis Data. Topik tugas akhir adalah akuisisi data dari database dan visualisasi *dependency* dari *stored procedure* yang merupakan bagian dari topik penelitian laboratorium Akuisisi Data dan Diseminasi Informasi.

Halaman ini sengaja dikosongkan

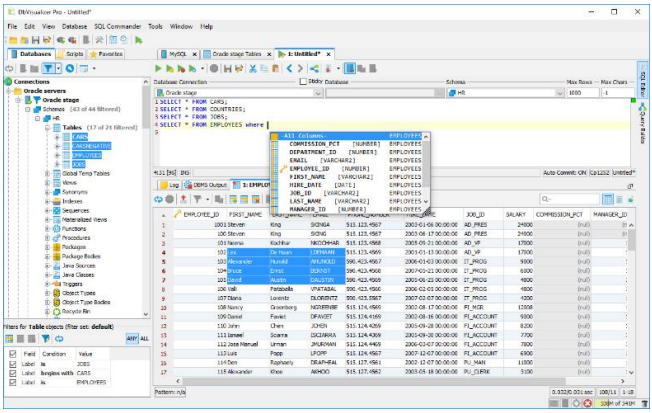
BAB II
TINJAUAN PUSTAKA

Tinjauan pustaka ini berisikan mengenai penjelasan teori-teori, penelitian sebelumnya yang terkait dengan topik tugas akhir, maupun aplikasi yang sejenis.

2.1. Produk Sejenis

Pada subbab ini akan dijeaskan tentang penelitian terkait yang digunakan dalam pengerjaan tugas akhir ini :

2.1.1. DBVisualizer



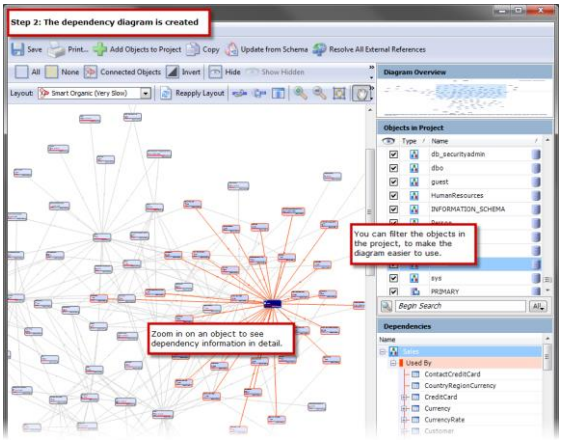
Gambar 2. 1 Tampilan DBVisualizer

DBVisualizer adalah aplikasi manajemen database yang dapat memvisualisasikan schema, table, view, memvisualisasikan aksi seperti CREATE,ALTER,DROP,RENAME, serta dapat membuat, mengedit dan *compile* prosedur, fungsi, dan trigger. Tetapi DBVisualizer memiliki kekurangan yaitu :

- Tidak dapat memvisualisasikan isi dan relasi yang terdapat pada prosedur atau stored procedure secara spesifik. Padahal stored procedure seringkali digunakan dalam proses pengelolaan database pada banyak kasus database *enterprise.*
- Tidak ada hubungan interaksi. Jenis relasi ini mengekspresikan penggunaan objek dalam objek lain.

- Tidak menampilkan SQL dan tidak bisa mengeksekusi SQL

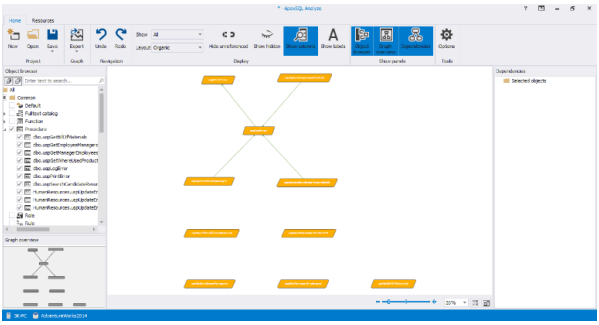
2.1.2. Red-gate: Toolbelt SQL Dependency Tracker



Gambar 2. 2 Tampilan Reg-gate Toolbelt SQL Dependency Tracker

Red-gate adalah aplikasi *SQL Dependency Tracker* yang secara dinamis mendokumentasikan semua dependensi database, dengan menggunakan berbagai layout grafis. Re-gate memiliki fitur seperti menemukan object yang tidak memiliki dependensi terhadap object lain untuk membantu membersihkan database. Red-gate toolbelt SQL Dependency tracker bisa memetakan hubungan berbagai objek dalam database.

2.1.3. ApexSQL Analyze



Gambar 2. 3 Tampilan ApexSQL Analyze

Apex SQL adalah software yang dapat menganalisis dependensi database dan membuat visualisasi secara kustom. ApexSQL Analyze dapat memetakan hubungan hamper semua objek dalam database. Namun, ApexSQL Analyze memiliki kekurangan sebagai berikut :

- Hanya memvisualisasikan hubungan antar objek dalam database. Tidak menampilkan hubungan interaksi.
- Dalam visualisasi dependensi stored procedur, tidak menampilkan transaksi yang ada didalamnya.

Untuk lebih mengetahui perbedaan dengan jelas , berikut adalah tabel perbandingannya :

Tabel 2. 1 Perbandingan Tugas Akhir dengan Aplikasi yang sejenis

	DBVisualizer	ApexSQL	Redgate Toolbelt	Aplikasi yang dibuat
Menampilkan tabel	V	V	V	V
Menampilkan function	X	V	V	V
Menampilkan Stored Procedure	X	V	X	V
Menampilkan hubungan stored procedure dengan obyek lain	X	V	X	V
Menampilkan hubungan interaksi	X	X	X	V

2.2. Penelitian Sebelumnya

Berikut adalah penelitian sebelumnya yang terkait dengan mappirng dependensi objek database.

<table>
<tr><td>Judul</td><td>Database Object Dependency Tool</td></tr>
<tr><td>Penulis</td><td>Ivana Lieskovska Drabikova, Karol Matiasko, Anton Lieskovsky</td></tr>
<tr><td>Metode</td><td>Metode yang digunakan mengambil konsep dasar dari graph. Analogi yang digunakan adalah sebagai berikut :
<table>
<tr><th>Graph</th><th>DB Object Mapper Tool</th></tr>
<tr><td>Node</td><td>DB Object (view, trigger, table, function)</td></tr>
<tr><td>Edge</td><td>Relation of interaction (interaction between DB objects)</td></tr>
</table>
Berdasarkan analogi diatas, dapat didefinisikan basic object untuk DB Object Mapper Tool seperti berikut :
<table>
<tr><td>Class: Node</td><td></td><td></td></tr>
<tr><td rowspan="2"></td><td>Node Name</td><td>Node name (table, trigger, view, procedure, function name)</td></tr>
<tr><td>Node Type</td><td>Node Type (table,</td></tr>
</table>
</td></tr>
</table>

			trigger, view, procedure, function)
		List of Node columns <list>	List of columns name [column name: date type]
		Lists of reference objects (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 the node
		Node_to_Triggers <list>	List of reference triggers related to the node
		Node_to_Procedures <list>	List of reference to procedures related to the node
Kesimpulan	1. Paper memperkenalkan sebuah konsep *software tool* untuk *mapping objects* dan hubungan didalam database. 2. Hubungan dari interaksi disebut dengan *dependencies* atau hubungan yang tidak dibuat dari statement ADD REFERENCES		

	3. Tujuan utama dari software ini adalah untuk merasionalkan proses dari analisis kebutuhan, menjadikan fase dari analisis lebih efisien dan mengoptimasi teknik yang digunakan dalam analisis yang bekerja dengan sistem database yang kompleks. 4. Sumber yang paling relevan dari paper ini adalah pengalaman praktis penulis dengan sistem dan kerja analitis untuk proyek unggulan di bidang perbankan dan telekomunikasi[6]

Berdasarkan penjelasan pada tabel diatas mengenai penelitian sebelumnya yang berjudul *"Database Object Dependency Tool"* dapat diketahui perbedaan dengan Tugas Akhir yang dikerjakan, yaitu:

1. Penelitian sebelumnya menggunakan tools untuk mempermudah analis *software* dalam tahap analisis sistem sedangkan pada tugas akhir *tools* digunakan untuk membantu dokumentasi perangkat lunak dengan cara *reverse engineering*
2. Penelitian sebelumnya menggunakan *systems table* pada Oracle sedangkan pada tugas akhir menggunakan database Microsoft SQL Server
3. Penelitian sebelumnya memetakan semua obyek database (tabel, view, prosedur, dll) sedangkan pada tugas akhir berfokus pada stored procedure dan hubungan interaksinya dengan obyek lain.

Sedangkan kesamaan penelitian sebelumnya dengan tugas akhir adalah menampilkan obyek database dan hubungan interaksi dengan obyek database lainya serta menggunakan prinsip graph database dalam membuat *tools* untuk memetakan hubungan obyek database.

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
Metode	Membuat aplikasi yang dapat memetakan dan memvisualisasikan relasi implisit pada database SQL Server. Visualisasi database SQL Server ke dalam bentuk graph visual dengan memanfaatkan fitur SQL Server yaitu Dynamic Management View. Data hubungan implicit yang didapat dari DMV kemudian diubah dalam bentuk array dan disimpan dalam bentuk graph data dalam Neo4J Graph Database.Informasi yang tersimpan dalam graph data kemudian dipanggil dan diolah menjadi bentuk diagram graph dengan memanfaatkan library D3.js
Kesimpulan	Query yang digunakan didalam aplikasi adalah system catalog dan dynamic management view berhasil memetakan hirarki SQL Server dan mendeteksi relationship yang bersifat eksplisit berbasis foreign key dan implisit berdasarkan JOIN[7] .

Berdasarkan penjelasan pada tabel diatas mengenai penelitian sebelumnya yang berjudul "*Rancang Bangun Aplikasi Visualisasi Database SQL Server dengan Dynamic Management View berbasis Graph Neo4J untuk memetakan relasi implisit pada database"* dapat diketahui perbedaan dengan Tugas Akhir yang dikerjakan, yaitu:

1. Penelitian sebelumnya memanfaatkan query *system catalog* dan *dynamic management view* sedangkan dalam tugas akhir tidak menggunakan query tersebut
2. Visualisasi yang ditampilkan mengenai hubungan eksplisit dan hubungan implisit suatu table dalam suatu database sedangkan tugas akhir ini menampilkan tentang dependency dari Stored Procedure dengan object database lainnya.

Sedangkan kesamaan penelitian sebelumnya dengan tugas akhir adalah menggunakan prinsip graph data dan juga menggunakan sumber yang sama yaitu database RESITS.

2.3. Dasar Teori

Pada subbab ini akan dibahas mengenai pengertian atau penjelasan dasar-dasar teori yang digunakan pada pengerjaan tugas akhir. Penjelasan lebih detail dapat dilihat pada subbab-subbab berikut ini.

2.3.1. Database Manajemen System SQL Server

SQL Server adalah sebuah system manajemen basis data relasional (RDMBS) yang merupakan produk dari Microsoft. Fungsi utama dari SQL Server ini adalah menyimpan dan mengambil data sesuai dengan permintaan dari software aplikasi – yang mungkin berjaan pada komputer yang sama atau komputer yang berbeda yang ada dalam satu jaringan.

Microsoft membuat SQL Server dalam berbagai edisi dengan fitur yang berbeda setiap edisinya dan menargetkan pengguna yang berbeda juga. Edisi yang ada adalah enterprise, standard, web, business intelligence, workgroup, express. Edisi special lainnya adalah azure, compact (SQL CE) developer, embedded, dana lain sebagainya.

Bahasa kuerinya biasa disebut T-SQL atau *transact-SQL* yang merupakan pengembangan dari SQL standar ANSI/ISO. T-SQL adalah merupakan sarana sekunder dari pemrograman dan pengelolaan untuk SQL Server. Keunggulan T-SQL dibanding

dengan PL/SQL adalah T-SQL terdapat sedikit kerentanan daripada PL/SQL dan T-SQL Mudah dikelola. Keunggulan dari T-SQL dibanding dengan SQL Standar adalah T-SQL memiliki fitur stored procedur yang dapat digunakan untuk memakai kembali query yang sering digunakan.

SQL Server kebanyakan digunakan dalam dunia bisnis yang memiliki basis data skala kecil, menengah maupun besar. Keunggulan dari SQL Server sendiri adalah memiliki manajemen password yang baik dan aman, mempunyai fitur backup dan restore, mudah diakses client dan user.

2.3.2. Stored Procedure

Stored procedure adalah salah satu objek yang tersimpan pada database SQL Server dan dapat digunakan untuk menggantikan berbagai kumpulan perintah yang sering digunakan. Stored procedure sangat berguna ketika tidak ingin user mengakses *table* secara langsung, atau dengan kata lain membatasi hak akases user dan mencatat operasi yang dilakukan. Dengan demikian risiko kebocoran dan kerusakan data dapat lebih diminimalisir.

Dalam suatu database bisa terdapat ribuan *stored procedure*. Dalam situasi tertentu, mencari dan menemukan *stored procedure* menjadi sesuatu yang sulit. Beruntungnya, SQL Server menyediakan beberapa metode untuk menemukan *stored procedure* berdasarkan nama, bagian nama, mempunyai text, tabel, atau kolom yang digunakan didalamnya. [8]

Mendapatkan list Stored Procedure

```
1.  SELECT *
2.  FROM information_schema.routines
3.  WHERE routine_type = 'PROCEDURE'
```

Kode 2. 1 Query list Stored Procedure

Selain dapat mencari stored procedure, Microsoft SQL Server bisa juga mencari daftar tabel, function, dan schema dari database. Berikut adalah T-SQL yang dapat digunakan

Mendapatkan daftar server, database, schema dan table

```
1.  SELECT @@SERVERNAME as srv, DB_NAME(DB_ID()) as db, SCHE
    MA_NAME(schema_id) as sch, sys.tables.name as tbl
2.  FROM sys.tables
```

Kode 2. 2 Query list server, db, schema, dan tabel

Mendapatkan daftar function

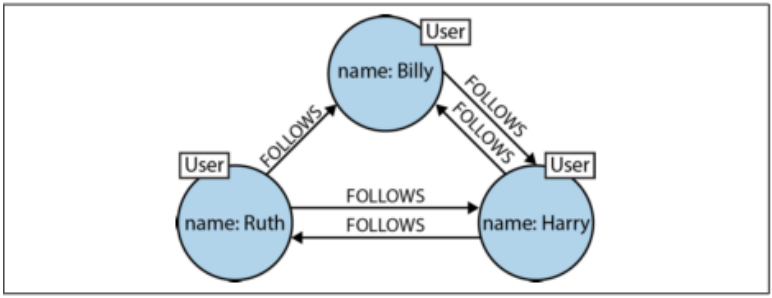
```
1.  SELECT *
2.  FROM information_schema.routines
3.  WHERE routine_type = 'FUNCTION'
```

Kode 2. 3 Query list Function

2.3.3. Parsing & Syntax Analysis

Bahasa pemrograman memiliki aturan yang menentukan struktur *syntax* program yang baik (*well-formed*). Syntax Bahasa pemrograman bisa dideskripsikan dalam bentuk notasi *context-free grammar*. Syntax analysis dalam Bahasa pemrograma biasa disebut juga dengan *parsing* adalah menentukan sebuah *syntax* dapat menghasilkan suatu string dari masukan yang didapat dari tahap *Lexical Analysis*. Sederhananya *parsing* adalah menentukan apakah *syntax* dapat membentuk susunan *string* berbentuk *abstract syntax tree* atau menghasilkan error. [9]

2.3.4. Graph



Gambar 2. 4 Konsep Graph Database

Graph merupakan sekumpulan garis dan *nodes* atau bisa dikatakan sebagai satu set *nodes* dan hubungan yang

menghubungkannya. Graph merepresentasikan entitas sebagai *nodes* dan bagaimana hubungan antara entitasnya. Secara umum, dengan struktur ekprisif ini memungkinkan untuk memodelkan berbagai macam jenis skenario, mulai dari pembuatan sistem jalan hingga riwayat kesahatan untuk suatu populasi [10]. Menurut Gartner (www.gartner.com) menyebutkan bahwa ada 5 jenis graph, yaitu : *social graph, intent graph, consumption graph, interest graph,* dan *mobile graph*. [11]. Contoh sederhana dari graph dapat dilihat pada gambar diatas. Pada gambar diatas terdapat *node* yang diberi label user. Masing-masing node dihubungkan oleh sebuah relationship, yang membentuk konteks semantic: Billy dan Harry saling memfollow. Ruth dan Harry saling memfollow, Ruth memfollow Billy tapi tidak sebaliknya [10]

2.3.5. Neo4J graph DBMS

Gambar 2. 5 Logo Neo4J

Graph database management system (selanjutnya disebut basis data *graph*) adalah sistem manajemen basis data online dengan metode *create, read, update, delete* (CRUD) yang menampilkan sebuah model data *graph* [10]. Terdapat dua properti basis data *graph*,, yaitu:

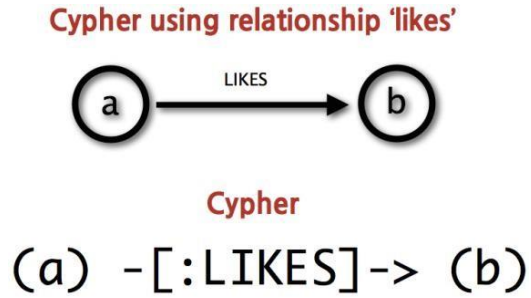
1. **the underlying storage**

   Beberapa basis data *graph* menggunakan *native graph storage* yang dioptimalkan dan dirancang untuk menyimpan dan mengelola *graph.*

2. the processing engine

Beberapa definisi mengharuskan sebuah *graph* menggunakan *index-free adjacency*, yang artinya *nodes* yang terhubung, secara fisikal saling menunjuk (*point*) di dalam basis data. Bukan seperti model basis data *graph* yang dihasilkan dari operasi CRUD.

Neo4J adalah sistem manajemen basis data graph yang dikembangkan oleh Neo4J, Inc., sebuah perusahaan yang berbasis di San Francisco Bay Area, Amerika Serikat. Neo4J adalah sebuah sistem basis data *transactional* dengan *native graph storage* dan *native processing* yang telah memenuhi ACID-*compliant*. Hasil perhitungan peringkat DB-Engines menunjukkan Neo4J menempati peringkat pertama sistem basis data *graph* paling populer mengalahkan Microsoft Azure Cosmos DB, OrientDB, Virtuoso, dan beberapa sistem manajemen basis data *graph* lain. Neo4J hadir dalam 3 lisensi: *community*, *enterprise*, dan *government*. Lisensi yang digunakan pada tugas akhir ini adalah lisensi *community*.



Gambar 2. 6 Konsep Chyper Query

Neo4J memiliki bahasa query bernama Cypher Query Language. Cypher adalah sebuah bahasa deklaratif yang terinspirasi dari SQL untuk mendeskripsikan pola *graph*. Cypher pada awalnya dikembangkan eksklusif untuk Neo4J tapi kemudian dengan dirilisnya openCypher kini Cypher bisa diterapkan pada SAP Hana dan AgensGraph. Cypher Query

mendefinisikan node *relationship* dengan syntax CREATE dan MATCH.

Berikut adalah contoh pendefinisian node You yang berlabel sebagai Person:

```
1.  CREATE (you:Person {name:"You"}) RETURN you
```

Kode 2. 4 Chyper Query Create

Berikut adalah contoh pendefinisian hubungan atau *relationship* antar node:

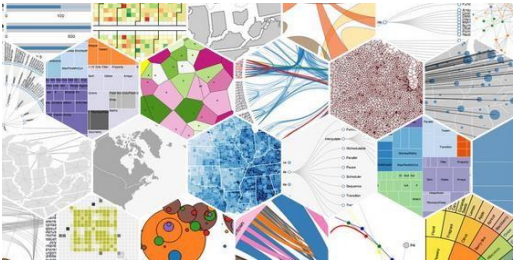
```
1.  MATCH  (you:Person {name:"You"})
2.  CREATE (you)-[like:LIKE]-
    >(neo:Database {name:"Neo4j" }) RETURN you,like,neo)
```

Kode 2. 5 Chyper Query Relationship

2.3.6. D3.js

Gambar 2. 7 Logo D3.js

D3.js adalah library javascript berlisensi BSD untuk manipulasi dokumen berbasis data. D3 adalah kepanjangan dari Data-Driven Documents [23]. D3.js memungkinkan untuk mengikat *arbitrary data* ke sebuah Document Object Model (DOM) yang selanjutnya dapat diterapkan *data-driven transformation.* Sebagai contoh D3.js bisa digunakan untuk membuat tabel berformat HTML dari sebuah array. Dengan data yang sama pula, sebuah diagram batang (*bar chart*) berformat SVG bisa dihasilkan. D3.js bisa diakses melalui repositori GitHub (http://github.com/d3/d3). D3.js mulai dikembangkan pada 18 Februari 2011. Saat ini versi terbaru D3.js adalah 4.11.0 yang dirilis pada 4 Oktober 2017.



Gambar 2. 8 Visualisasi D3.js

Pada versi terbarunya D3.js menyediakan 30 modul yang bisa digunakan antara lain: arrays, geographies, hierarchies, polygons, dll. Contoh langkah-langkah penggunaan D3.js pada aplikasi HTML untuk membuat SVG *circle chart* yaitu:

1. Instalasi D3.js pada folder aplikasi dimana D3.js diterapkan. Apabila menggunakan NPM bisa dengan menuliskan npm install d3. Bisa juga dengan mengunduhnya secara manual pada repositori GitHub (https://github.com/d3/d3/releases/download/v4.11.0/d3. zip) dan kemudian mengekstraknya ke folder aplikasi.
2. Insiasi D3.js pada header HTML <script **src="https://d3js.org/d3.v4.min.js"></script>**
3. Melakukan proses *data-binding* yaitu proses untuk mendefinisikan data yang digunakan.

```
1.  var data = [
2.  {name:"Indonesia",gdp:932,color: "green"},
3.  {name:"Singapore",gdp:297,color: "blue"},  {name:"
    Malaysia",gdp:296,color:"red"}];
```

Kode 2. 6 Kode Data-binding

4. Membuat SVG container beserta attributnya.

```
1.  .attr("width", 250)
2.  .attr("height", 250)
3.  .style("background-color", "#DEDEDE");
```

Kode 2. 7 Kode SVG Container

5. Membuat *circle chart* dengan mengambil data yang telah di-*binding*.

```
1.  svg.selectAll("circle")
2.  .data(data)
3.  .enter()
4.  .append("circle")
5.  .attr("id", function(d) { return d.name })
6.  .attr("cx", function(d) { return d.gdp })
7.  .attr("fill", function(d) { return d.color });
```

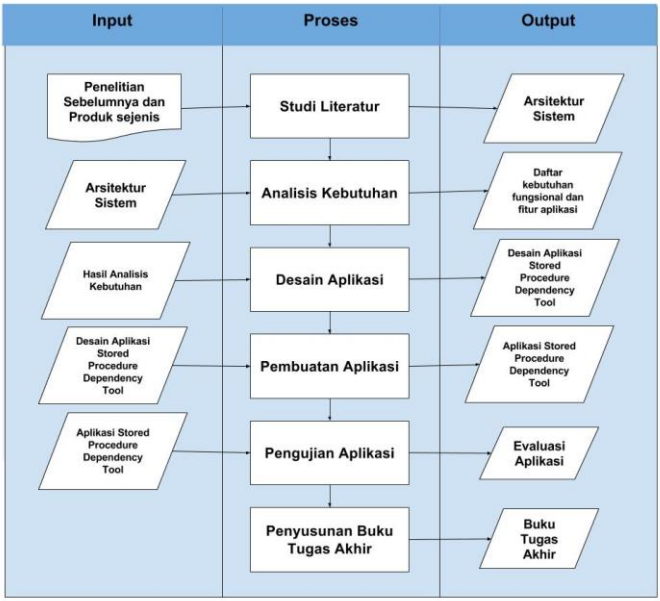
Kode 2. 8 Kode D3.js

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan dijelaskan mengena alur metode penelitian yang akan dilakukan oleh peneliti dalam melakukan penelitian ini. Metode penelitian ini juga digunakan sebagai pedoman untuk melaksanakan penelitian agar terarah dan sistematis.

3.1. Tahapan Pelaksanaan

Pada bagian ini akan dijelaskan mengenai metode yang digunakan dalam menyelesaikan tugas akhir ini. Metodologi pengerjaan tigas akhir ini ditunjukkan oleh gambar dibawah ini.



Gambar 3. 1 Metodologi Penelitian

Berdasarkan gambar diatas berikut adalah penjelasan dari tiap-tiap proses yang dilakukan dalam menyelesaikan tugas akhir ini

3.1.1. Studi Literatur

Pada proses studi literatur dilakukan pencarian terhadap materi-materi yang terkait dengan penelitian tugas akhir. Materi-materi yang dicari meliputi :

1. Konsep Microsoft SQL Server
2. Mendapatkan daftar table, schema, function, stored procedure dari database
3. Konsep Stored Procedure
4. Konsep dan algoritma *parsing*
5. Konsep *graph database* dengan Neo4J
6. Teknik akses Neo4J dan penulisan query Chyper dengan Bahasa PHP
7. Teknik pembuatan graph Neo4J
8. Teknik pembuatan diagram dengan menggunakan D3.js dari graph Neo4J
9. Studi literature juga dilakukan dengan mencari penelitian atau jurnal-jurnal terkait dengan visualisasi *dependency* objek pada database.

Luaran yang dihasilkan dari proses ini adalah mengetahui knowledge gap dan pemahaman berdasarkan literatur yang dibutuhkan serta arsitektur sistem dari aplikasi.

3.1.2. Analisis Kebutuhan

Untuk memastikan pembuatan aplikasi berjalan sesuai dengan kebutuhan pengguna, dilakukan analisis kebutuhan. Berdasarkan rumusan masalah yang berada pada Bab I, berikut adalah analisis kebutuhan fungsional dari aplikasi :

1. Aplikasi dapat mendapatkan daftar tabel, schema, fungsi, dan stored procedure pada database.
2. Aplikasi dapat memparsing query stored procedure.
3. Aplikasi dapat mengidentifikasi *dependency* stored procedure dengan objek database lainya.
4. Aplikasi dapat menyimpan hasil *parsing* kedalam bentuk graph data Neo4J.
5. Aplikasi dapat mengolah data dari graph data Neo4J menjadi bentuk visual berupa relation chart.

3.1.3. Desain Aplikasi

Pada tahap ini dilakukan pembuatan desain aplikasi. Pembuatan desain aplikasi dibuat berdasarakan kebutuhan sistem dan berdasarkan hasil dari studi literatur yang dilakukan. Desain aplikasi dapat dilihat melalui arsitektur sistem dari aplikasi. Pada tahap ini, dibuat desain berupa:

a. Desain alur sistem
   Pada tahap ini dilakukan pembuatan desain dari alur sistem yang berjalan. Alur sistem yang dibuat mulai dari pengguna mengakses aplikasi dan menerima data pengelolaan sistem.
b. Desain antar muka
   Pada tahap ini, dilakukan pembuatan desain yang digunakan untuk tampilan aplikasi. Aplikasi akan menampilkan *relation chart* mengenai stored procedure dan hubunganya dengan objek database lainnya.
c. Desain database
   Pada tahap ini, dilakukan pembuatan desain database Neo4J yang digunakan. Desain database yang dimaksud adalah label apa saja yang digunakan, property yang diperlukan, serta bagaimana relasi antar node.

3.1.4. Pembuatan aplikasi

Pembuatan aplikasi berdasarkan tiga proses utama yang dilakukan oleh aplikasi. Berikut adalah tahapan pembuatan aplikasi :

1. ***Extraction***
   Proses extraction adalah proses mendapatkan informasi mengenai objek database seperti server, database, schema, tabel, function, dan stored procedure. Pada proses ini terdapat satu komponen sebagai berikut.
   a. DB Object Getter
      Query yang digunakan untuk mendapatkan table, schema, function dan stored procedure adalah sebagai berikut:

- Mendapatkan daftar stored procedure

```
1.  SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA
    , SPECIFIC_NAME, ROUTINE_TYPE, ROUTINE_B
    ODY, ROUTINE_DEFINITION, SQL_DATA_ACCESS
    , CREATED, LAST_ALTERED
2.  FROM information_schema.routines
3.  WHERE routine_type = 'PROCEDURE`
```

Kode 3. 1 Query List Stored Procedure

- Mendapatkan daftar table, schema, database, dan server

```
1.  SELECT @@SERVERNAME as srv, DB_NAME(DB_I
    D()) as db, SCHEMA_NAME(schema_id) as sc
    h, sys.tables.name as tbl
2.  FROM sys.tables
```

Kode 3. 2 Query List Tabel, Schema, Database, Server

- Mendapatkan daftar function

```
1.  SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA
    , SPECIFIC_NAME, ROUTINE_TYPE, ROUTINE_B
    ODY, ROUTINE_DEFINITION, SQL_DATA_ACCESS
    , CREATED, LAST_ALTERED
2.  FROM information_schema.routines
3.  WHERE routine_type = 'FUNCTION'
```

Kode 3. 3 Query List Function

2. Parsing

Proses parsing adalah proses mendapatkan hubungan stored procedure dengan objek database lainya berdasarkan hasil *parsing* dari *SQL Definition* dari Stored Procedure. Dalam proses ini terdapat tiga komponen yaitu SP Splitter, SP Parser, dan Translator.

a. SP Splitter

Pada tahap ini, dilakukan penulisan kode untuk memisahkan kata pada *SQL Definition* dari setiap Stored Procedure dan kemudian disimpan dalam array.

b. SP Parser

Pada tahap ini dilakukan penulisan kode parser untuk mem-*parsing* array yang berisi *SQL Definition* dari Stored Procedure. SP Parser ini akan mencari kata kunci yang mengindikasikan hubungan dengan tabel atau Stored Procedure lain seperti FROM, JOIN, INSERT INTO, UPDATE, MERGE, TRUNCATE, dan EXECUTE.

c. Translator

Pada tahap ini, data hasil parser diubah dalam bentuk graph database menggunakan platform Neo4J dengan query Cypher. Dikarenakan aplikasi yang digunakan menggunakan Bahasa PHP dengan memanfaatkkan GraphAware PHP Client, maka penulisan mengikuti aturan yang ada pada GraphAware PHP Client. Penulisan kode sebagai berikut:

3. Visulization

Visualization adalah proses menampilkan data yang telah didapat dari proses extraction dan parsing. Data yang ditampilkan adalah visualisasi hubungan stored procedure dengan database lainnya. Proses ini memiliki dua komponen yaitu metadata dan visualizer.

a. Metadata

Metadata adalah komponen dari visualization yang berisi data mengenai node dan relationship yang diambil dari Neo4J. Metadata ini memiliki tipe data JSON dengan format JSON sudah sesuai dengan format JSON untuk Neo4jd3.js.

b. Visualizer

Penulisan kode visualisasi program memanfaatkan library D3.js. Kode yang digunakan adalah Neo4jd3.js yang merupakan modifikasi dari library D3.js yang mendukung struktur data graph Neo4J. Selain dirancang untuk data dari Neo4J, library tersebut juga mengakomodasi penggunaan node dari text, Font Awesome, dan Twitter Emoji.

3.1.5. Pengujian Aplikasi

Pada tahap ini dilakukan pengujian aplikasi untuk mengetahui apakah aplikasi dapat berjalan sesuai dengan rancangan sebelumnya, menemukan bug atau error, serta mengetahui akurasi dari apikasi. Pengujian yang dilakukan berfokus pada hal berikut ini :

a. Performa Aplikasi
   Pengujian ini bertujuan untuk mengetahui performa dari aplikasi. Pengujian dilakukan dengan cara menghitung waktu eksekusi script php yang digunakan untuk melakukan pengumpulan data. Script php dijalankan sebanyak 5 kali dan dihitung rata-rata waktu eksekusi
b. Fungsionalitas
   Pengujian fungsionalitas dilakukan dengan melakukan unit testing pada setiap komponen dari aplikasi. Pengujian dilakukan dengan menggunakan metode *blackbox testing* dan mencatat error setiap komponen. Komponen yang diuji adalah komponen SP Parser dan Visualizer.

3.1.6. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan dokumentasi pada setiap tahapan tugas akhir mulai dari studi literatur hingga pengujian aplikasi. Penyusunan laporan dikerjakan selama penelitian berlangsung dan juga sebagai aktivitas akhir dalam kegiatan Tugas Akhir ini. Didalam buku tersebut mencakup :

a. Bab I Pendahuluan
   Dalam bab ini dijelaskan mengenai latar belakang masalah, perumusan masalah, Batasan masalah, tujuan tugas akhir, manfaat tugas akhir, serta relevansi tugas akhir untuk memberi

gambaran umum permasalahan dan pemecahan masalah pada tugas akhir.

b. Bab II Tinjauan Pustaka
   Dalam bab ini dijelaskan mengenai penelitian sebelumnya, produk yang serupa dan dasar teori yang menunjang permasalahan yang dibahas pada tugas akhir sebagai acuan atau landasan dalam pengerjaan tugas akhir ini.
c. Bab III Metodologi
   Dalam bab ini dijelaskan mngenai gambaran metode dan alur pengerjaan yang dilakukan pada tugas akhir ini.
d. Bab IV Perancangan
   Dalam bab ini dijelaskan mengenai rancangan aplikasi, mulai dari analisis kebutuhan, desain aplikasi, pembuatan aplikasi, serta pengujian aplikasi.
e. Bab V Implementasi
   Dalam bab ini dijelaskan mengenai proses pelakasanna penelitian dan pembuatan aplikasi berdasarkan perancangan yang telah dibuat sebelumnya.
f. Bab VI Hasil dan Pembahasan
   Dalam bab ini dijelaskan mengenai hasil serta analisis dari pengujian aplikasi yang telah dibuat.
g. Bab VII Kesimpulan dan Saran
   Dalam bab ini dijelaskan mengenai kesimpulan dari semua proses yang telah dilakukan dan saran yang dapat diberikan untuk pengembangan yang lebih baik.

Halaman ini sengaja dikosongkan

BAB IV
PERANCANGAN

Pada bab ini akan dibahas mengenai alur perancangan terkait beberapa hal yang diperlukan dalam proses pembuatan aplikasi sesuai dengan alur uang dijelasakan pada bab sebelumnya. Dalam bab perancangan ini akan menjelaskan tentang proses penggalian kebutuhan dan desain sistem.

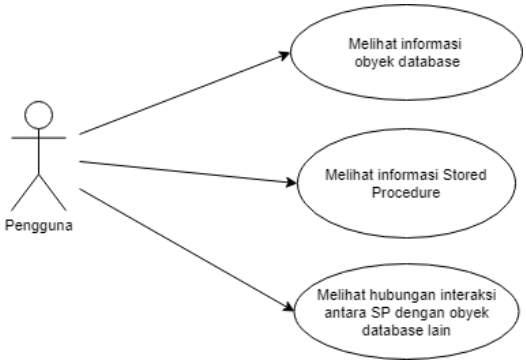
4.1. Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengetahui kebutuhan yang dibutuhkan oleh pengguna. Dari proses analisis kebutuhan akan dihasilkan kebutuhan fungsional dari aplikasi yang kemudian digunakan sebagai dasar pembuatan aplikasi.

4.1.1. Fungsi Utama Perangkat Lunak

Pembuatan aplikasi Stored Procedure Dependency Tool berbasis graph Neo4j ini bertujuan untuk memetakan dan memvisualisasikan hubungan interaksi antara Stored Procedure pada database berbasis SQL Server dengan obyek database lainnya.

4.1.2. Pengguna Aplikasi



Gambar 4. 1 Use Case Diagram Stored Procedure Dependency Tool

Pengguna dari aplikasi Stored Procedure Dependency Tool ini adalah Database Administrator. Berikut adalah pengguna aplikasi Stored Procedure Dependency Tool dan aktivitas-aktivitas yang dapat dilakukan oleh pengguna.

Berdasarkan use case diagram diatas, dapat diketahui bahwa pengguna dapat melakukan 3 hal utama, yaitu :

1. Melihat informasi obyek database
   Pengguna dapat melihat informasi mengenai obyek database. Obyek database yang dimaksud adalah server, database, schema, table, stored procedure dan function.
2. Melihat informasi Stored Procedure
   Pengguna dapat melihat informasi mengenai Stored Procedure yang ada dalam database. Informasi yang bisa didapat seperti server, database, schema, waktu pembuatan, waktu *alter* terakhir dari Stored Procedure.
3. Melihat hubungan interaksi antara Stored Procedure dengan obyek database lain.
   Pengguna dapat melihat informasi hubungan interaksi antara stored procedure dengan obyek database yang lainya.

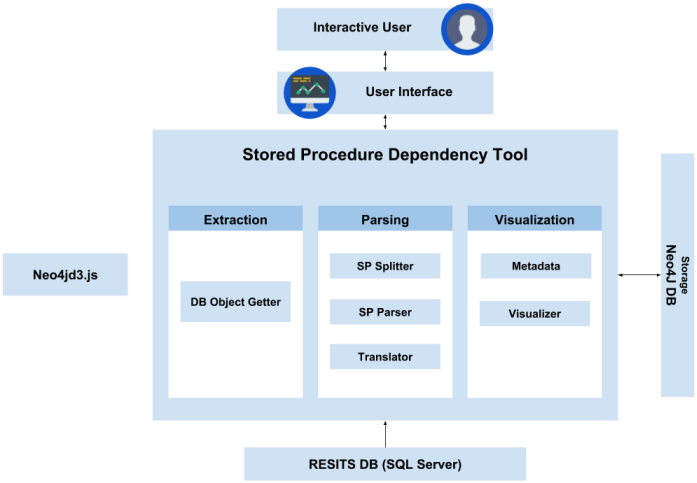
4.1.3. Kebutuhan Fungsional

Proses analisis kebutuhan dilakukan untuk mengetahui fungsi dari perangkat. Berikut adalah kebutuhan fungsional dari aplikasi :

1. Aplikasi bisa mendapatkan daftar tabel, schema, fungsi, dan stored procedure pada database.
2. Aplikasi dapat memparsing query stored procedure.
3. Aplikasi dapat mengidentifikasi *dependency* stored procedure dengan objek database lainya.
4. Aplikasi dapat menyimpan hasil *parsing* kedalam bentuk graph data Neo4J.
5. Aplikasi dapat mengolah data dari graph data Neo4J menjadi bentuk visual berupa relation chart.

4.2. Arsitektur Sistem

Dalam pembuatan perangkat lunak, diperlukan sebuah arsitektur aplikasi. Arsitektur aplikasi adalah sebuah teknologi spesifikasi yang akan digunakan untuk mengimplementasikan sistem informasi. Arsitektur aplikasi menjadi suatu desain aplikasi yang terdiri dari komponen-komponen yang saling berinteraksi antara satu dengan yang lain. Arsitektur aplikasi untuk perangkat lunak *stored procedure dependency tool* lebih jelaskan akan digambarkan seperti gambar 4.2.



Gambar 4. 2 Arsitektur Sistem Stored Procedure Dependency Tool

Pada aplikasi *Stored Procedure Dependency Tool* terdapat 3 proses utama yaitu :

4.2.1. Extraction

Proses extraction bertujuan untuk mendapatkan informasi mengenai database seperti daftar server, database, schema, table, function, dan stored procedure. Setelah informasi tersebut didapat, kemudian disimpan dalam bentuk graph database pada Neo4J.

Berikut adalah flowchart untuk proses extraction :
Dalam proses extraction, komponen yang terlibat, yaitu:

a. DB Object Getter

DB Object Getter berfungsi untuk mendapatkan informasi mengenai objek database yang ada dalam database RESITS. Objek database yang diambil antara lain server, database, schema, table, stored procedure dan function.

- Mendapatkan daftar stored procedure

```
1.  SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA
    , SPECIFIC_NAME, ROUTINE_TYPE, ROUTINE_B
    ODY, ROUTINE_DEFINITION, SQL_DATA_ACCESS
    , CREATED, LAST_ALTERED
2.  FROM information_schema.routines
3.  WHERE routine_type = 'PROCEDURE`
```

Kode 4. 1 Query Daftar Stored Procedure

- Mendapatkan daftar table, schema, database, dan server

```
1.  SELECT @@SERVERNAME as srv, DB_NAME(DB_I
    D()) as db, SCHEMA_NAME(schema_id) as sc
    h, sys.tables.name as tbl
2.  FROM sys.tables
```

Kode 4. 2 Query Dafta tabel, schema, database, server

- Mendapatkan daftar function

```
1.  SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA
    , SPECIFIC_NAME, ROUTINE_TYPE, ROUTINE_B
    ODY, ROUTINE_DEFINITION, SQL_DATA_ACCESS
    , CREATED, LAST_ALTERED
2.  FROM information_schema.routines
3.  WHERE routine_type = 'FUNCTION'
```

Kode 4. 3 Query Daftar Function

4.2.2. Parsing

Proses parsing bertujuan untuk mem-*parsing SQL Definiton* dari Stored Procedure yang terdapat pada database. Dari hasil

parsing tersebut kemudian didefinisikan hubungan antara Stored Procedure dengan obyek database lain.
Komponen yang terlibat pada proses parsing ini, yaitu:

a. SP Splitter

SP Splitter berfungsi untuk membersihkan *comment* pada *SQL Definition* dari Stored Procedure. Selain itu komponen ini juga berfungsi untuk memecah *SQL Definition* ke dalam array. Pemecahan *SQL Definition* menggunakan *regex* dengan berdasarkan beberapa karakter seperti Comma (,), Semicolon (;), Bracket ((,)), serta Whitespace.

b. SP Parser

SP Parser berfungsi untuk mem-*parsing* array hasil dari SP Splitter kemudian mencari nama tabel dan Stored Procedure yang berhubungan dengan Stored Procedure dengan mencari kata kunci FROM, INSERT INTO, MERGE, UPDATE, TRUNCATE, dan EXECUTE. Berikut adalah pseudocode dari method yang digunakan untuk mem-*parsing* array dari SQL Definition.

```
1.  public function KEYWORD($lexer){
2.      $key_word = array_keys($lexer, "keyword");
3.      if (!emptyempty($key_word)) {
4.          $arr = array();
5.          foreach ($key_word as $key) {
6.          $tbl_name = $lexer[$key+1];
7.          $valid = exceptiom;
8.              if ($valid) {
9.              $arr[] = $tbl_name;
10.             }
11.         }
12.     return array_unique($arr);
13.     }
14. }
```

Kode 4. 4 Pseudocode Parser

c. Translator

Translator berfungsi untuk mengidentifikasi hasil *parsing* dari SP Parser untuk disesuaikan dengan nama obyek database yang tersimpan pada Neo4J. Setelah hasil parsing sesuai format, translator akan membuat relasi antara stored

procedure dengan obyek database terkait pada Neo4J. Chyper query untuk membuat relationship antara Stored Procedure dan Objek database lain adalah sebagai berikut :

```
1.  MATCH (a:SP {surname: “SP name” }), (b:Table {surnam
    e:”Table name” })
2.  MERGE (a)-[r:Use]->(b)
3.  ON CREATE SET r.From = "Yes", r.Join = "No", r.Merge
     = "No", r.Truncate = "No", r.Insert = "No", r.Updat
    e = "No"
4.  ON MATCH SET r.From = "Yes"
```

Kode 4. 5 Chyper Query untuk Membuat Relasi

4.2.3. Visualization

Proses visualization bertujuan untuk mengambil data dari graph data dan memvisualisasikan hubungan antara stored procedure dengan objek database lainnya.

a. Metadata

Metadata adalah komponen yang berisi data mengenai node dan relasi yang diambil dari database Neo4J. Metadata ini memiliki tipe data JSON dengan format JSON sudah sesuai dengan format JSON untuk Neo4jd3.js.

Contoh format neo4jd3.js adalah sebagai berikut:

```
1.  {
2.      "results": [
3.          {
4.              "columns": ["user", "entity"],
5.              "data": [
6.                  {
7.                      "graph": {
8.                          "nodes": [
9.                              {
10.                                 "id": "1",
11.                                 "labels": ["User"],
12.                                 "properties": {
13.                                     "userId": "eisman"
14.                                 }
15.                             },
16.                             {
17.                                 "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
                                }
                            }
                        ]
                    }
                }
            ]
        }
    ],
    "errors": []
}
```

Kode 4. 6 Contoh format JSON untuk neo4jd3.js

b. *Visualizer*

Visualizer berfungsi untuk menampilkan data dari *metadata*. Visualizer memanfaatkan library neo4jd3.js untuk menampilkan data.

4.2.4. Storage Neo4J

Komponen storage Neo4J adalah komponen utama untuk penyimpanan data dengan struktur *graph*. Komponen isi menyimpan *node* dan *relationship* yang merupakan hasil dari proses *extraction* dan proses *parsing*. Struktur *node* dan

relationship akan dijelaskan pada subbab desain database. Data-data yang terdapat pada storage neo4j dijadikan sebagai masukan untuk komponen visualization.

4.2.5. RESITS DB

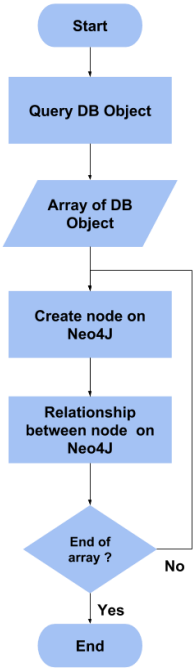
RESITS DB merupakan database sumber yang digunakan. Komponen ini berisi data-data terkait objek database yang kemudian akan diolah dalam proses extraction dan parsing. RESITS DB berbasis Microsoft SQL Server.

4.3. Desain Alur Sistem

Pembuatan desain alur sistem dilakukan untuk memudahkan memahami cara kerja dari sistem yang akan dibuat. Alur sistem menjelaskan tiga proses utama dari sistem yaitu proses extraction, parsing, dan visualization. Penjelasan rinci mengenai masing-masing proses dijelaskan pada subbab berikut.

4.3.1. Proses Extraction

Proses extraction adalah proses pengambilan data mengenai objek database dan kemudian disimpan dalam bentuk graph database pada Neo4J. Objek database yang diambil pada proses ini adalah server yang berhubungan dengan sistem, database, schema, tabel, stored procedure, dan function.

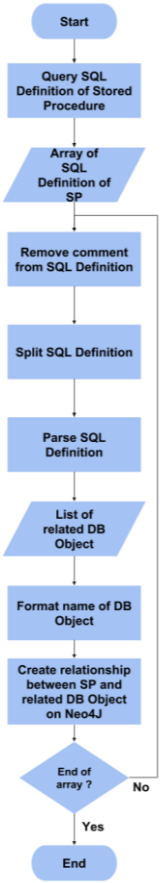


Gambar 4. 3 Flowchart Proses Extraction

Gambar diatas menjelaskan proses extraction. Proses extraction dimulai ketika aplikasi melakukan query untuk mendapatkan daftar objek database yang terdapat pada database berbasis SQL Server. Daftar objek database yang didapat kemudian disimpan dalam array. Objek database yang diambil pada proses ini adalah server/instance, database, schema, tabel, stored procedure, dan function. Setiap objek database kemudian disimpan dalam bentuk *node* pada database neo4J. Setelah *node* objek database terbentuk, dibuat *relationship* antar objek database yang ada. *Relationship* yang dibuat pada proses ini adalah srv, db, sch, tbl, sp, dan func. Setiap *relationship* dijelaskan pada subbab desain database. Proses extraction ini dijalankan secara berkala.

4.3.2. Proses Parsing

Proses parsing adalah proses pengambilan SQL Definition dari stored procedure kemudian mem-*parsing*-nya untuk mengidentifikasi objek database yang terkait denganya. Proses parsing ini dilakukan setelah proses extraction selesai.

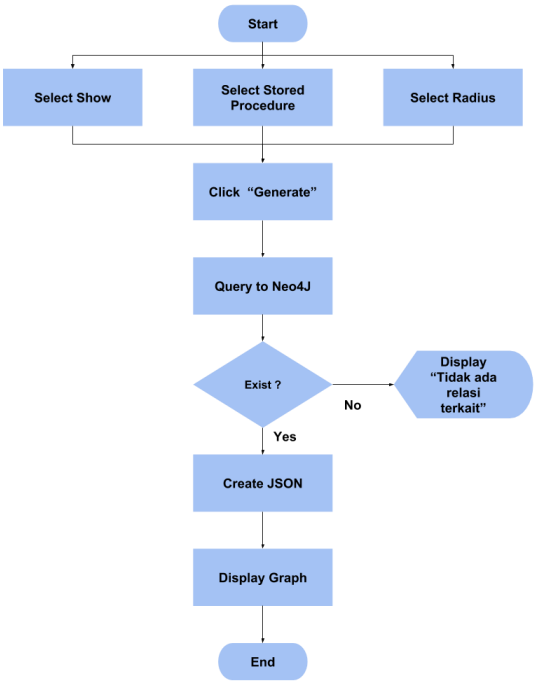


Gambar 4. 4 Flowchart Proses Parsing

Gambar diatas flowchart untuk proses parsing. Proses parsing dimulai ketika aplikasi melakukan *query* untuk mendapatkan *SQL Definition* untuk setiap stored procedure yang ada. Daftar *SQL Definition* dan informasi lain mengenai stored procedure disimpan didalam array. Langkah selanjutnya, komentar pada setiap SQL Definition dihilangkan. Setelah komentar hilang, *SQL Definition* dipecah menjadi array untuk memudahkan *parsing*. Kemudian, dilakukan *parsing* pada array dari *SQL Definition* untuk mengidentifikasi objek database apa saja yang terkait dengan stored procedure. Hasil dari parsing ini adalah daftar objek database apa saja yang berhubungan dengan stored procedure. Nama objek database yang dihasilkan belum sesuai dengan format *surname* objek database yang ada pada neo4j, sehingga dilakukan *formating* nama yang sesuai. Setelah itu, dilakukan *matching* antara stored procedure dengan objek database lain dan dibuat *relationship* diantaranya. *Relationship* yang terbentuk adalah use dan execute. Penjelasan mengenai *relationship* dibahasi pada subbab desain database. Proses parsing ini dilakukan secara berkala.

4.3.3. Proses Visualization

Proses visualization adalah proses menampilkan data objek database dan hubungan diantaranya yang sudah tersimpan pada neo4j. Proses visualization ini dapat dilakukan setelah proses extraction dan proses parsing sudah dilakukan. Penjelasan lebih rinci dapat dilihat pada gambar berikut ini.



Gambar 4. 5 Flowchart Proses Visualization

Proses visualization dimulai ketika pengguna memilih *option* show, memilih stored procedure yang ingin ditampilkan, serta radius yang diinginkan. Ketiganya bisa dipilih secara pararel. Setelah semua pilihan dipilih, pengguna akan meng-klik tombol Generate. Aplikasi akan melakukan *query* ke neo4j sesuai dengan opsi pilihan yang dipilih. Jika *query* tidak mengembalikan data, maka aplikasi akan menampilkan *alert* "Tidak ada relasi terkait". Jika *query* mengembalikan data, maka akan dibuat sebuah variable berformat JSON untuk menyimpan informasi *node* dan *relationship* yang didapat. Variabel ini kemudian digunakan sebagai masukan untuk library neo4jd3.js dalam menampilkan data. Setelah data JSON diproses oleh neo4jd3.js, akan ditampilkan graph sesuai dengan opsi yang dipilih.

4.4. Desain Antarmuka

Desain antarmuka dilakukan untuk membuat rancangan tampilan aplikasi yang akan digunakan. Aplikasi akan menampilkan hubungan antara Stored Procedure dengan obyek database lain dalam bentuk *relation chart.* Selain itu, aplikasi juga dapat melakukan *filtering* dan menampilkan hasilnya. Gambaran desain antarmuka aplikasi dapat dilihat pada gambar 4.7.

Penjelasan dari desain antarmuka aplikasi sebagai berikut :

1. Bagian ini merupakan *filter*. Terdapat tiga pilihan filter, yaitu *show* dan *Stored Procedure.* Filter show terdapat 3 pilihan yaitu *All, Execute,* dan *Use*. All akan menampilkan semua relasi. Execute akan menampilkan hanya yang memiliki relasi execute. Sedangkan Use akan menampilkan hanya yang memiliki relasi use. Untuk filter Stored Procedure berisi list Stored Procedure yang ada pada database. Filter yang ketiga adalah filter radius. Radius yang bisa dipilih adalah 1-2 radius relasi.
   Pada bagian bawah terdapat tombol GENERATE untuk mengeksekusi kriteria *filtering* yang telah dipilih.
2. Bagian ini akan menampilkan informasi tentang label dan property objek yang dipilih. Bagian ini akan muncul ketika pengguna mengarahkan pointer pada objek database.
3. Lingkaran menunjukkan objek database. Setiap objek database ditunjukkan dengan *icon* yang berbeda. Penjelasan icon adalah sebagai berikut :

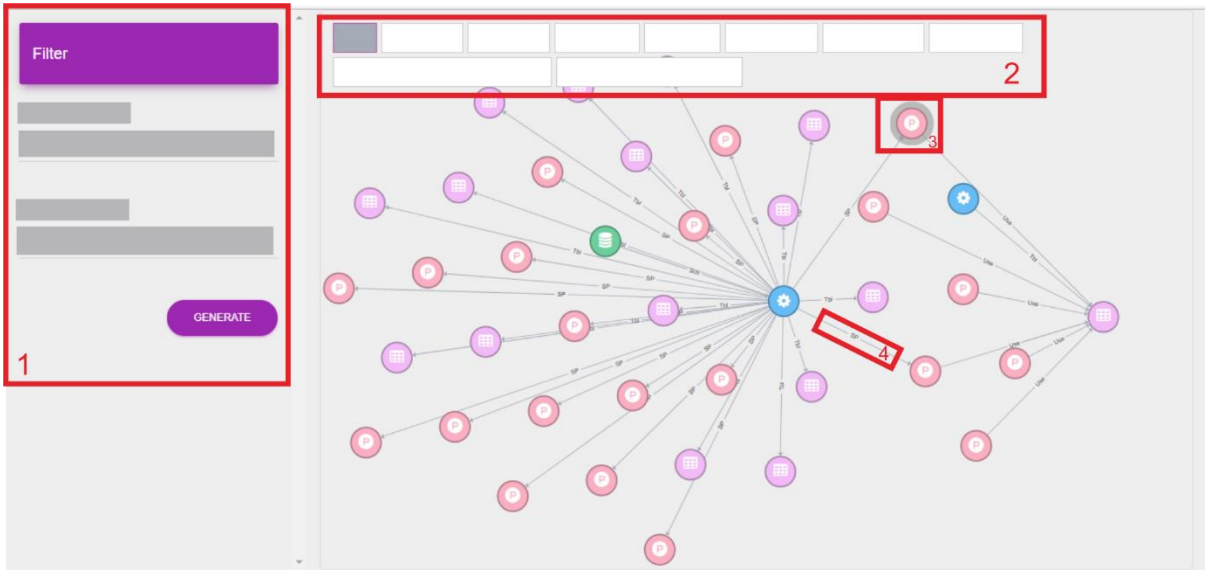
Tabel 4. 1 Penjelasan simbol aplikasi

Icon	Keterangan
	Instance / Server

Icon	Keterangan
	Database
	Schema
	Table
	Stored Procedure
	Function

Ketika pengguna memilih salah satu stored procedure untuk ditampilkan, maka *node* dari *stored procedure* tersebut

Garis menunjukkan hubungan interaksi antar objek database. Arah panah menunjukkan objek sumber dan objek target.



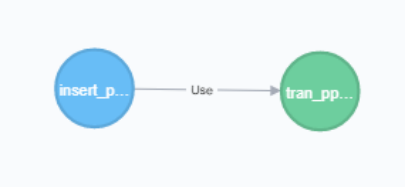
Gambar 4. 6 Desain antarmuka aplikasi

4.5. Desain Database

Sebuah graph database dapat menyimpan berbagai macam data menggunakan konsep sederhana. Tiga komponen penting pada graph database adalah :

1. Nodes – *records* dari graph database
2. Relationship – menghubungkan antar node
3. Property – *named data values*

Contoh sederhana dari graph yang digunakan dapa gambar dibawah ini.



Gambar 4. 7 Contoh graph data

Pada gambar, terdapat *node* yang memiliki label SP dan Table. Node dihubungkan oleh sebuah relationship yang membentuk kontes semantic: Stored Procedure insert_ppm_pengabdian_masyarakat menggunakan tabel tran_ppm_dosen.

Penggunaan masing-masing komponen dijelaskan pada tabel berikut ini :

Tabel 4. 2 Keterangan komponen graph database

Komponen	Keterangan
Node	Objek Database
Relationship	Interaksi antar objek database
Property	Informasi mengenai objek database

Setiap node memiliki memiliki property masing-masing. Property untuk setiap node berbeda-beda tergantung pada tipe/label yang dimiliki.

Tabel 4. 3 Property yang dimiliki setiap label

Label	Property yang dimiliki
Server	ID, name
Database	ID, Name, Surname, Server
Schema	ID, Name, Surname, Server, Database
Table	ID, Name, Surname, Server, Database, Schem, Column, PK
Function	ID, Name, Surname, Server, Database, Schema, Created, Last_altered
SP	ID, Name, Surname, Server, Database, Schema, Created, Last_altered

Penjelasan mengenai property untuk masing-masing node dapat dilihat pada tabel berikut ini.

Tabel 4. 4 Penjelasan property node

Property	Keterangan
ID	ID dari node
Label	Label/tipe dari node atau *DB Object* (Server, Database, Schema, Table, Function, SP)
Name	Nama dari *DB Object*
Surname	Nama identifikasi dari *DB Object*
Server	Server dari *DB Object*

Property	Keterangan
Database	Database dari *DB Object*
Schema	Schema dari *DB Object*
Column	Column yang dimiliki oleh tabel (khusus tabel)
PK	Primary Key yang dimiliki oleh tabel (khusus tabel)
Created	Waktu pembuatan (khusus function dan Stored Procedure)
Last_altered	Waktu terakhir akses (khusus function dan stored procedure)

Relationship berfungsi untuk menghubungkan antar node. Selain itu relationship juga dapat menggambarkan hubungan antar objek database yang ada. Setiap relationship memiliki tipe yang berbeda-beda bergantung pada jenis hubungan antar *node*. Macam-macam tipe relationship dapat dilihat pada tabel berikut.

Tabel 4. 5 Tipe relationship

Tipe Relasi	Keterangan
Db	Relasi antara server dengan database
Sch	Relasi antara database dengan schema
Tbl	Relasi antara schema dengan tabel
SP	Relasi schema dengan Stored Procedure
Func	Relasi schema dengan Function
Use	Relasi antara Stored Procedure dengan Table
Execute	Relasi antara Stored Procedure dengan Stored Procedure lain

Tidak hanya node, relationship juga memiliki property. Property pada relationship berfungsi untuk memberikan informasi detail mengenai hubungan antar node. Property untuk masing-masing relationship berbeda bergantung pada tipenya.

Tabel 4. 6 Property untuk relationship

Relationship	Property yang dimiliki
Db	ID
Sch	ID
Tbl	ID
Func	ID
SP	ID
Use	ID, From, Insert, Join, Merge, Truncate, Update
Execute	ID

Uraian lebih jelas mengenai property relationship dapat dilihat pada tabel berikut ini.

Tabel 4. 7 Penjelasan property untuk relationship

Property	Keterangan
ID	ID dari relasi
Type	Tipe relasi
FK	Foreign Key yang menghubungkan tabel
Par_tbl	Parent table
Ref_tbl	Reference table
From	Interaksi from (value : yes/no)
Insert	Interaksi insert (value : yes/no)

Property	Keterangan
Join	Interaksi join (value : yes/no)
Merge	Interaksi merge (value : yes/no)
Truncate	Interaksi truncate (value : yes/no)
Update	Interaksi update (value : yes/no)

4.6. Desain Parser

Pada subbab ini akan dijelaskan mengenai desain komponen-komponen yang digunakan pada proses *parsing*. Proses *parsing* pada penelitian kali ini menggunakan parser yang dikembangkan sendiri oleh penulis berdasarkan standar penulisan syntax T-SQL dan observasi terhadap stored procedure. Pada proses *parsing* terdapat tiga komponen yaitu, SP Splitter, SP Parser, dan Translator. Penjelasan lebih rinci desain setiap komponen terdapat pada subbab-subbab berikut ini.

4.6.1 Desain SP Splitter

Komponen SP splitter memiliki fungsi untuk memecah *SQL definition* dari *Stored Procedure* menjadi array. Pemecahan dilakukan untuk mempermudah proses *parsing* yang dilakukan oleh komponen SP Parser. Hasil array dari komponen ini akan menjadi masukan untuk komponen SP Parser.

Sebelum dilakukan pemecahan SQL Definition, dilakukan penghapusan komentar. Komentar yang terdapat pada SQL Definition tidak dibaca oleh server. Adanya komentar dapat mengganggu proses *parsing*. Standar penulisan *syntax* untuk komentar pada T-SQL berdasarkan dokumentasi Ms. SQL Server [12][13] adalah sebagai berikut :

```
1.  /*
2.  text_of_comment
3.  */
```

Kode 4. 7 Standar Penulisan Syntax Block Comment

```
1.  -- text_of_comment
```

Kode 4. 8 Standar Penulisan Syntax Comment

Berdasarkan standar penulisan *syntax* diatas, jika pada SQL Definition terdapat *syntax* dengan awalan – atau diantara /* dan /* dianggap sebagai komentar dan dihapus.

Setelah penghapusan komentar, dilakukan penggantian beberapa kata kunci. Penggantian kata kunci ini dilakukan karena terkadang penulisan query tidak sesuai dengan standar yang ada. Penggantian kata kunci juga dilakukan untuk menstandardisasi *query* yang akan dideteksi untuk proses *parsing*. Kata kunci yang diganti adalah sebagai berikut.

- PROCEDURE
- PROC
- FROM
- MERGE
- JOIN
- TABLE
- INTO
- UPDATE
- EXECUTE
- EXEC

Kata kunci diatas akan diganti menjadi *lowercase* supaya memudahkan proses *parsing* karena parser yang dibuat *case sensitive*. Pemilihan kata kunci diatas akan dijelaskan pada subbab desain SP parser.

Setelah komentar dihapus dan kata kunci diganti, selanjutnya adalah melakukan pemecahan SQL Definition menjadi array. Berdasarkan Transact-SQL Syntax Conventions [14] yang membahas tentang standar penulisan syntax T-SQL, pemecahan SQL Definition menjadi array didasarkan oleh beberapa karakter berikut:

- Koma (,)
- Titik koma (;)
- Tanda kurung (( ))

- Kurung Siku ([ ])
- Whitespace

Karakter-karakter diatas akan memisahkan setiap *syntax* dan menyimpannya dalam array. Array yang merupakan pecahan dari *SQL Definition* ini kemudian menjadi masukan untuk komponen SP Parser.

4.6.2 Desain SP Parser

Komponen SP Parser bertugas untuk melakukan *parsing*. *Parsing* dilakukan untuk mengidentifikasi hubungan *Stored Procedure* dengan objek database lain. Hubungan diidentifikasi dari *query* yang digunakan dalam *SQL Definition*. Setelah dilakukan observasi pada *Stored Procedure* yang terdapat pada database RESITS, *query* yang sering digunakan adalah *from*, *merge*, *join*, *truncate*, *insert*, *update*, dan *execute*.

Cara mengidentifikasi hubungan objek database berdasarkan query yaitu dengan melihat pola penulisan *syntax*-nya. Berdasarkan dokumentasi Microsoft SQL Server [15], berikut adalah standar penulisan *syntax* untuk *query* yang telah dipilih.

a. From

Query from biasanya digunakan untuk menentukan tabel, tampilan, tabel turunan, dan tabel gabungan yang digunakan dalam pernyataaan delete, select, dan update pada SQL Server.

Menurut dokumentasi T-SQL, standar penulisan *query* untuk *syntax* from adalah sebagai berikut.

```
1.    [ FROM { <table_source> } [ ,...n ] ]
2.   <table_source> ::=
3.   {
4.        table_or_view_name [ [ AS ] table_alias ]
5.        | @variable [ [ AS ] table_alias ]
6.            …
```

Kode 4. 9 Standar Query Syntax From

Berdasarkan standar penulisan pada Kode 4.9, dapat diketahui jika setelah *syntax* from diikuti dengan nama tabel sumber yang dituju. Hal ini menandakan bahwa *syntax* from dapat digunakan untuk mengidentifikasi hubungan *stored procedure* dengan tabel. Selain nama tabel, setelah *syntax* from juga bisa diikuti dengan nama variable. Nama variabel pada database biasanya berawalan dengan '@'. Untuk menghindari parser mendeteksi variabel sebagai objek database, parser memiliki filter untuk tidak meloloskan *value* yang memiliki awalan '@'.

b. Merge

Query merge biasanya digunakan untuk melakukan operasi insert, update, atau delete pada tabel yang ditargetkan berdasarkan hasil dari join dengan tabel sumber. Hal ini menunjukan bahwa query merge dapat digunakan untuk mengidentifikasi hubungan stored procedure dengan tabel. Menurut dokumentasi T-SQL, standar penulisan query untuk syntax merge adalah sebagai berikut.

```
1.  [ WITH <common_table_expression> [,...n] ]
2.  MERGE
3.      …
4.  <target_table> ::=
5.  {
6.      [ database_name . schema_name . | schema_name . ]
7.    target_table
8.  }
```

Kode 4. 10 Standar Query Syntax Merge

Seperti dapat dilihat pada Kode 4.10 setelah *syntax* merge diikuti dengan nama tabel target yang diinginkan. Dari hal tersebut dapat dilihat pola *query* untuk mengidentifikasi tabel yang berhubungan.

c. Join

Query join dipakai untuk melakukan penggabungan antar tabel atau view. Hal ini menandakan bahwa *query* join dapat

digunakan untuk mengidentifikasi tabel yang berhubungan dengan *stored procedure*.

Menurut dokumentasi T-SQL, standar penulisan *query* untuk *syntax* join adalah sebagai berikut.

```
1.    [ FROM { <table_source> } [ ,...n ] ]
2.   <table_source> ::=
3.   {
4.       …
5.       | <joined_table>
6.   }
7.   …
8.
9.   <joined_table> ::=
10.  {
11.      <table_source> <join_type> <table_source> ON <search_conditi
     on>
12.      | <table_source> CROSS JOIN <table_source>
13.      | left_table_source { CROSS | OUTER } APPLY right_table_sour
     ce
14.      | [ ( ] <joined_table> [ ) ]
15.  }
16.  <join_type> ::=
17.      [ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } [ <join_
     hint> ] ]
18.      JOIN
```

Kode 4. 11 Standar Query Syntax Join

Pada Kode 4.11 baris ke-11 dapat dilihat standar penulisan *query* join. Sesuai standar, setelah penulisan *syntax* join selanjutnya akan diikut dengan nama tabel sumber yang digunakan. Hal ini dapat digunakan sebagai dasar untuk mengidentifikasi tabel yang berhubungan dengan *stored procedure*. Terkadang *syntax* join juga diikuti dengan *query* select. Untuk menghindari parser mendeteksi query select sebagai objek database, parser memiliki filter untuk tidak meloloskan *value* yang mengandung kata select sebagai objek database yang berhubungan.

d. Truncate

Query truncate digunakan untuk menghapus semua baris pada tabel atau bagian tertentu pada tabel, tanpa *logging* penghapusan baris individu. Hal ini menandakan *query* truncate dapat digunakan untuk mengidentifikasi hubungan stored procedure dengan tabel.

Menurut dokumentasi T-SQL, standar penulisan query untuk *syntax* truncate adalah sebagai berikut.

```
1.  TRUNCATE TABLE
2.      [ { database_name .[ schema_name ] . | schema_name . } ]
3.      table_name
4.  …
```

Kode 4. 12 Standar Query Syntax Truncate

Dapat dilihat pada Kode 4.12 bahwa setelah *syntax* truncate table diikuti dengan nama tabel target. Dari pola ini, dapat digunakan sebagai acuan untuk mengidentifikasi nama tabel yang berhubungan melalui *query* ini.

e. Insert

Query insert digunakan untuk menambah satu atau lebih baris ke tabel atau view pada SQL Server. Hal ini menandakan bahwa melalui *query* insert dapat diidentifikasi tabel yang berhubungan dengan *stored procedure*.

Menurut dokumentasi T-SQL, standar penulisan query untuk syntax insert adalah sebagai berikut.

```
1.   [ WITH <common_table_expression> [ ,...n ] ]
2.  INSERT
3.  {
4.          …
5.          [ INTO ]
6.          { <object> | rowset_function_limited
7.            …
8.  }
9.  [;]
```

```
10.
11.  <object> ::=
12.  {
13.      [ server_name . database_name . schema_name .
14.        | database_name .[ schema_name ] .
15.        | schema_name .
16.      ]
17.    table_or_view_name
18.  }
```

Kode 4. 13 Standar Query Syntax Insert

Dapat dilihat pada Kode 4.13 bahwa setelah *syntax* insert into adalah nama tabel atau view. Pola *syntax* ini dapat digunakan sebagai acuan untuk mengidentifikasi nama tabel yang berhubungan.

f. Update

Query update digunakan untuk mengubah data yang sudah ada pada tabel atau view pada SQL Server. Hal ini menandakan bahwa *query* ini dapat digunakan untuk membantu mengidentifikasi tabel yang berhubungan dengan *stored procedure.*

Menurut dokumentasi T-SQL, standar penulisan *query* untuk *syntax* update adalah sebagai berikut.

```
1.   [ WITH <common_table_expression> [...n] ]
2.   UPDATE
3.       …
4.       { { table_alias | <object> | rowset_function_limited
     …
5.       }
6.       SET
7.        …
8.   [ ; ]
9.
10.  <object> ::=
11.  {
12.      [ server_name . database_name . schema_name .
13.      | database_name .[ schema_name ] .
14.      | schema_name .
15.      ]
```

```
16.     table_or_view_name}
```

Kode 4. 14 Standar Query Syntax Update

Dapat dilihat pada Kode 4.14 bahwa setelah *syntax* update adalah nama tabel atau view. Pola *syntax* ini dapat digunakan sebagai acuan untuk mengidentifikasi nama tabel yang berhubungan.

g. Execute

Query execute digunakan untuk menjalankan string perintah atau string karakter dalam batch T-SQL, atau menjalankan *system stored procedure*, *user-defined stored procedure*, atau *function*. Hal ini menandakan bahwa dari *query* execute dapat digunakan untuk mengidentifikasi stored procedure lain yang berhubungan.
Menurut dokumentasi T-SQL, standar penulisan *query* untuk *syntax* execute adalah sebagai berikut.

```
1.  -- Execute a stored procedure
2.  [ { EXEC | EXECUTE } ]
3.      procedure_name
4.          [ { value | @variable [ OUT | OUTPUT ] } ] [ ,...n ] }

5.  [;]
```

Kode 4. 15 Standar Query Syntax Execute

Dapat dilihat pada Kode 4.15 bahwa setelah *syntax* execute atau exec adalah nama *stored procedure* atau *function*. Pola *syntax* ini dapat digunakan sebagai acuan untuk mengidentifikasi nama *stored procedure* yang berhubungan.

h. Procedure

Stored procedure mirip dengan prosedur dalam bahasa pemrograman lain karena mereka dapat:

- Menerima parameter input dan mengembalikan beberapa nilai dalam bentuk parameter output ke prosedur pemanggilan atau batch

- Berisi *programming statement* yang melakukan operasi dalam database, termasuk memanggil prosedur lain
- Mengembalikan nilai status ke batch untuk menunjukkan keberhasilan atau kegagalan (dan alasan kegagalan).

T-SQL yang digunakan untuk *stored procedure* adalah sebagai berikut.

```
1.  CREATE [ OR ALTER ] { PROC | PROCEDURE }
2.      [schema_name.] procedure_name [ ; number ]
3.      [ { @parameter [ type_schema_name. ] data_type }
4.          [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]

5.      ] [ ,...n ]
6.  [ WITH <procedure_option> [ ,...n ] ]
7.  [ FOR REPLICATION ]
8.  AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
9.  [;]
10.
11. <procedure_option> ::=
12.     [ ENCRYPTION ]
13.     [ RECOMPILE ]
14.     [ EXECUTE AS Clause ]
```

Kode 4. 16 Standar Query Syntax Procedure

Dapat dilihat pada Kode 4.16 bahwa setelah *syntax* create atau alter procedure adalah nama *stored procedure*. Pola *syntax* ini dapat digunakan sebagai acuan untuk mengidentifikasi nama *stored procedure* yang di-*parsing*.

4.6.3 Desain Translator

Komponen translator ini bertugas untuk melakukan *formatting* nama objek database sebelum pembentukan relasi pada *graph database*. *Formatting* dilakukan untuk menspesifikan objek database yang berhubungan karena dalam satu server bisa saja terdapat nama objek yang sama namun berada pada database atau schema yang berbeda. Kemungkinan nama objek yang redundan ini akan mengurangi keakurasian relasi yang dibuat.

Dalam melakukan formatting nama, penulis mengacu pada *T-SQL Conventions* mengenai format nama objek database[14]. Berdasarkan itu, semua T-SQL yang berhubungan dengan nama objek database dapat menggunakan *four-part name* dalam bentuk berikut:

```
1.  server_name .[database_name].[schema_name].object_name
2.  | database_name.[schema_name].object_name
3.  | schema_name.object_name
4.  | object_name
```

Kode 4. 17 Standar Penamaan Objek Database T-SQL

Keterangan dari Kode 4.17 adalah sebagai berikut:

Server_name
Menentukan nama linked server atau remote server.

Database_name
Menentukan nama database pada lingkungan lokal SQL Server.

Schema_name
Menentukan nama schema dimana objek database berada.

Object_name
Mengacu pada nama objek database.

Berdasarkan T-SQL Conventions, tabel berikut menunjukan format nama objek yang valid.

Tabel 4. 8 Format Nama Objek Database

Object reference format	Description
server . database . schema . object	*Four-part name*
server . database .. object	Nama schema dihilangkan
server .. schema . object	Nama database dihilangkan
server .. object	Nama database dan schema dihilangkan
database . schema . object	Nama server dihilangkan

Object reference format	Description
database .. object	Nama server dan schema dihilangkan
schema . object	Nama server dan database dihilangkan
object	Nama server, database, dan schema dihilangkan

Berdasarkan standar nama objek database pada T-SQL yang telah dijelaskan diatas, dilakukan analisis *formatting* nama objek database. Analisis *formatting* berdasarkan jumlah titik yang terdapat pada nama objek database yang dideteksi oleh parser. Cara identifikasinya adalah sebagai berikut :

- Nama objek database tidak mengandung titik (.)
  Dapat dianalisis jika penulisan nama objek hasil *parsing* tidak mengandung titik maka berasal dari schema, database, dan server yang sama dengan *stored procedure.* Sehingga nama objek database ditambah dengan nama server, database, dan schema dari stored procedure.
- Nama objek database mengandung satu titik(.)
  Jika nama objek database hasil *parsing* mengandung satu titik (.) pada penulisannya, berdasarkan standar penulisan yang ditunjukkan pada Kode 4.17 baris ke-3, sudah terdapat nama schema dari objek dan objek berasal dari database dan server yang sama dengan stored procedure. Sehingga nama objek database ditambah dengan nama server dan database dari stored procedure.
- Nama objek database mengandung dua titik(.)
  Dapat dianalisis jika nama objek database hasil *parsing* terdapat dua titik (.), berdasarkan standar penulisan yang ditunjukka pada Kode 4.17 baris ke-2, objek database berasal dari server yang sama dengan stored

procedure. Sehingga nama objek database akan ditambah dengan nama server dari stored procedure.

- Nama objek database mengandung tiga titik(.)
  Jika nama objek database hasil *parsing* mengandung tiga titik (.) seperti yang ditunjukkan Kode 4.17 baris ke-1, maka nama objek database tidak diubah.

Nama objek database yang telah di-*format* akan digunakan untuk mencocokan data objek database pada *graph database* yang telah dibuat sebelumnya dan membuat relasi antara objek database tersebut dengan *stored procedure*.

Pembuatan relasi antara objek database dengan *stored procedure* menggunakan *query chyper*. *Query chyper* yang digunakan adalah MATCH dan MERGE. Untuk query MERGE, menggunakan ON CREATE dan ON MATCH. ON CREATE digunakan ketika relasi antara stored procedure dan objek database belum dibuat. ON MATCH digunakan ketika relasi antara stored procedure dan objek database sudah terbuat dan hanya mengubah value dari property yang ada. Berikut contoh penggunaan chyper query untuk MATCH dan MERGE.

```
1.  MERGE (var:Label { property: 'value' })
2.  ON CREATE SET property_key = value
3.  MERGE (var:Label)
4.  ON MATCH SET property_key = value
```

Kode 4. 18 Query Chyper Merge

```
1.  MATCH (:Label { property_key: 'value' })-[relationship]->(var)
2.  RETURN relationship
```

Kode 4. 19 Query Chyper Match

4.7. Desain Pengujian Aplikasi

Pada tahap ini akan dikelaskan mengenai desain pengujian aplikasi. Pengujian dilakukan dengan metode *black box testing*. Hal-hal yang diuji adalah performa aplikasi, ketepatan hasil *parsing*, dan ketepatan aplikasi dalam menampilkan data.

4.7.1 Pengujian Performa Aplikasi

Pengujian ini bertujuan untuk mengetahui performa dari aplikasi. Pengujian dilakukan dengan cara menghitung waktu eksekusi *script* php yang digunakan untuk melakukan pengumpulan data. *Script* php dijalankan sebanyak 5 kali dan dihitung rata-rata waktu eksekusi.

4.7.2 Pengujian Fungsionalitas Aplikasi

Pengujian fungsionalitas dilakukan dengan melakukan unit testing pada setiap komponen dari aplikasi. Pengujian dilakukan dengan menggunakan metode *blackbox testing* dan mencatat error setiap komponen. Komponen yang diuji adalah komponen SP Parser dan Visualizer. Rancangan pengujian fungsionalitas untuk komponen SP Parser dapat dilihat pada Tabel 4.9 berikut.

Tabel 4. 9 Rancangan Pengujian Fungsional Komponen SP Parser

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan
S01	Mendeteksi objek database yang berhubungan from	SP: insert_ppm_penelitian	Tabel: • its-dw.ppm.penelitian • resits.dbo.tran_publikasi_dosen_tetap
		SP: generate_ jumlah_ scopus	Tabel: • its-dw.ppm.scopus_jurnal • its-dw.ppm.scopus_seminar

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan
			• resits.dbo.tmst_pegawai
S02	Mendeteksi objek database yang berhubungan join	SP: mapping_fakultas_jurusan_recent	Tabel: • its-dw.akademik.fakultas • resits.dbo.tmst_fakultas • resits.dbo.tmst_jurusan
		SP: mapping_dosen_recent	Tabel: resits.dbo.tmst_pegawai
S03	Mendeteksi objek database yang berhubungan merge	SP: mapping_dosen_recent	Tabel: resits.dbo.mapping_dosen
		SP: insert_jurusan_itsdw	Tabel: resits.dbo.tmst_jurusan_baru
S04	Mendeteksi objek database yang berhubungan truncate	SP: generate_mapping_temp_dosen	Tabel: resits.dbo.mapping_temp_dosen
		SP: tran_temp_publikasi_recent	Tabel: • resits.dbo.tran_temp_publikasi • resits.dbo.tran_temp_publikasi_distinct • resits.dbo.tran_temp_publikasi_mapping

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan
S05	Mendeteksi objek database yang berhubungan insert	SP: insert_publikasi_itsdw_buku	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		SP: insert_publikasi_itsdw_seminar	Tabel: resits.dbo.tran_publikasi_dosen_tetap
S06	Mendeteksi objek database yang berhubungan update	SP: insert_publikasi_itsdw_seminar	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		SP: insert_publikasi_itsdw_jurnal	Tabel: resits.dbo.tran_publikasi_dosen_tetap
S07	Mendeteksi objek database yang berhubungan execute	SP: master_run	SP: • insert_fakultas_itsdw • insert_jurusan_itsdw • insert_ppm_pegawai_dosen • insert_laboratorium_ppm • insert_ppm_buku • insert_ppm_penelitian • insert_publikasi_itsdw_jurnal_gs • insert_publikasi_itsdw_jurnal_scopus

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan
			• insert_publikasi_itsdw_paten • insert_ppm_pengabdian_masyarakat • insert_publikasi_itsdw_seminar_gs • insert_publikasi_itsdw_seminar_scopus • distinct_publikasi_dosen_tetap • tran_temp_publikasi_recent • generate_mapping_temp_dosen • generate_mapping_publikasi_dosen • update_laboratorium_fakjur_itsdw • mapping_dosen_recent
		SP: sp_helpdiagrams	Tidak memunculkan hasil

Berikut adalah rancangan pengujian fungsional untuk komponen visualizer.

Tabel 4. 10 Rancangan Pengujian Fungsional Komponen SP Parser

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan
V01	Mengubah data *graph* Neo4J ke dalam format JSON	Hasil Chyper Query pada Neo4J	Aplikasi berhasil membuat data yang berformat JSON untuk *node* dan relasinya
V02	Menampilkan form filter untuk interaksi pengguna	-	Aplikasi menampilkan form filter
V03	Menampilkan *graph* dari format JSON	JSON berisi graph data	Aplikasi menampilkan *graph* sesuai dengan data.
V04	Menampilkan *graph* sesuai filter yang dipilih pengguna	JSON dan form filter	Aplikasi menampilkan *graph* sesuai filter yang dipilih pengguna
V05	Menampilkan pesan error jika tidak ada *graph* yang sesuai filter	Query Chyper dari Neo4J	Aplikasi menampilkan pesan error ketika tidak ada *graph* sesuai filter.

Setiap ID pengujian akan dilakukan pengecekan sesuai dengan hasil yang diharapkan. Jika hasil pengujian sesuai dengan hasil yang diharapkan, maka setiap uji berstatus “Sukses”. Jika hasil pengujian tidak sesuai dengan hasil yang diharapkan, maka berstatus “Gagal”.

BAB V
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai terkait proses implementasi pada perangkat lunak sesuai dengan perancangan sistem yang dilakukan pada bab sebelumnya.

5.1. Lingkungan Implementasi

Pada bagian ini dibahas mengenai lingkungan implementasi yang digunakan dalam pembuatan dan pengujian aplikasi. Lingkungan implementasi terkait dengan perangkat keras dan perangkat lunak yang digunakan. Tabel 5.1 berikut berisikan tentang spesifkasi perang keras yang digunakan.

Tabel 5. 1 Spesifikasi Perangkat Keras

Perangkat	Spesifikasi
Jenis	ASUS A442UR
Processor	Intel Core i5-8250u 3.4GHz
Memory	8 GB
Storage	1 TB

Selain perangkat keras, pembuatan aplikasi juga menggunakan beberapa teknologi pendukung seperti yang dijelaskan pada tabel 5.2.

Tabel 5. 2 Teknologi pendukung pengembangan aplikasi

Webserver	XAMPP v3.2.2
Bahasa Pemrograman	• PHP 7.1.15 with Composer • JavaScript
RDMS	SQL Server Express Edition

Graph DBMS	Neo4J 3.3.4 Enterprise
Text Editor	Sublime 3.1.1
Web Browser	Google Chrome 66
Library	• Bootstrap 3.3.7 • Font Awesome 5 • D3.js 4.12.0 • Neo4jD3.js by eismann • PDO & ODBC 11 • GraphAware PHP Client
OS Component	Command Prompt (cmd.exe)

Lingkungan implementasi pada tabel 5.1 dan tabel 5.2 harus dipastikan berjalan dengan baik agar pembuatan dan pengujian aplikasi tidak mengalami hambatan.

5.2. Implementasi

Pada bagian ini akan dijelaskan mengenai implementasi kode untuk masing-masing komponen dari aplikasi. Mulai dari konfigurasi yang dibutuhkan aplikasi, implementasi kode untuk masing-masing komponen, serta konfigurasi otomasi proses.

5.2.1. Konfigurasi Aplikasi

Untuk memastikan aplikasi dapat terhubung dengan DBMS, dilakukan konfigurasi koneksi yang membutuhkan alamat server, informasi port, nama database, username dan password. Konfigurasi ini terdapat pada file config/database.php dan bootstrap/helper.php.

Aplikasi ini memiliki dua macam konfigurasi karena menggunakan dua DBMS yang berbeda yaitu Microsoft SQL Server dan Neo4J.

Sebelum melakukan konfigurasi, perlu dilakukan instalasi library PDO untuk koneksi ke SQL Server dan instalasi library GraphAware untuk koneksi ke Neo4J. Instalasi PDO untuk sql server dapat dilakukan dengan mengunduh library melalui website resmi dari Microsoft, kemudian menginstalnya pada folder C:\xampp\php\ext\. Instalasi graphaware untuk koneksi antara php dengan database neo4j dapat dilakukan melaui composer dengan menjalankan command dibawah ini.

```
composer require graphaware/neo4j-php-client:^4.0
```

Kode 5. 1 Command instalasi graphaware

Neo4j client for PHP perlu dipanggil dengan kode berikut

```
require_once 'vendor/autoload.php';
use GraphAware\Neo4j\Client\ClientBuilder;
```

Kode 5. 2 Penggunaan Library Graphaware

Pada implementasinya dalam aplikasi ini, dibuat file php khusus yang digunakan untuk memanggil autoload dan file php ini akan dipanggil pada setiap file php yang membutuhkannya. Script php dalam file tersebut adalah sebagai berikut.

```
namespace Dependency;
require_once __DIR__.'/../vendor/autoload.php';
```

Kode 5. 3 Konfigurasi autoload

Untuk membuat koneksi, dilakukan pembuatan method createSQLServerConnection untuk koneksi ke database SQL Server dan method createNeo4jConnection untuk koneksi ke database Neo4J.

Berikut adalah *function* untuk koneksi ke sql server. *Function* tersebut memiliki 6 parameter, yaitu parameter server, port, username, password, database, dan prefix.

```
1.  function createSQLServerConnection(string $server = null, int $p
    ort = null, string $username = null, string $password = null, st
    ring $database = null, string $prefix = 'sqlsrv') {
2.      $serverName = 'tcp:'.$server.','.$port;
3.      $conn = new PDO("$prefix:server=$serverName ; Database=$dat
    abase", $username, $password);
4.      $conn-
    >setAttribute(PDO::SQLSRV_ATTR_ENCODING, PDO::SQLSRV_ENCODING_UT
    F8);
5.
6.      return $conn;
7.   }
```

Kode 5. 4 Fungsi Inisiasi Koneksi SQL Server

Berikut adalah *function* untuk koneksi ke neo4j. *Function* ini memiliki 4 parameter yaitu, username, password, host, dan port.

```
1.  function createNeo4jConnection(string $username = null, string $
    password = null, string $host = null, int $port = null) {
2.      return ClientBuilder::create()
3.              -
    >addConnection('bolt', "bolt://$username:$password@$host:$port")

4.              ->build();
5.  }
```

Kode 5. 5 Fungsi Inisiasi Koneksi Neo4J

Setiap *function* diatas akan dipanggil pada file php yang membutuhkan koneksi ke sql server atau neo4j.

Pada file config/database.php mengembalikan array yang berisi konfigurasi yang dibutuhkan untuk koneksi ke database. Konfigurasi untuk koneksi sql server, dibutuhkan alamat host, port, nama database, username, password. Host merupakan alamat IP dari server olap.its.ac.id yaitu 10.199.2.66. Port yang digunakan adalah 1433. Serta database yang digunakan adalah database resits.

Sedangkan untuk koneksi ke neo4j dibutuhkan informasi mengenai host, port, username, dan password. Host yang digunakan untuk pengembangan aplikasi ini menggunakan localhost. Neo4J bisa melakukan koneksi melaui bolt di port 7687, http di 7474, dan https di 7473. Namun pada

implementasi aplikasi ini, dilakukan koneksi melalui bolt dengan port 7687. Konfigurasi koneksi untuk sql server terdapat pada baris ke-3 hingga baris ke-10. Sedangkan konfigurasi untuk koneksi database Neo4J terdapat pada baris ke-14 hingga baris ke-17.

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

Kode 5. 6 Parameter Konfigurasi Database

Contoh konfigurasi koneksi untuk setiap script php adalah sebagai berikut.

```
1.  require_once __DIR__.'/../hihi.php';
2.  //initiation
3.  $databaseConfig = require config_path('database.php');
4.  $neo4jConfig = $databaseConfig['connections']['neo4j']['sp'];
5.  $sqlSrvConfig = $databaseConfig['connections']['sqlsrv'];
```

Kode 5. 7 Konfigurasi Koneksi Database (1)

Hal pertama yang dilakukan adalah memanggil file php yang menggunakan autoload yang ditunjukan pada baris 1. Setelah itu, memanggil array pada database.php yang ditunjukkan pada baris ke-3 hingga ke-5.

```
1.  //connection to  neo4j
2.  $neo = createNeo4jConnection($neo4jConfig['username'], $neo4jCon
    fig['password'], $neo4jConfig['host'], $neo4jConfig['port']);
3.  //connection to sql server
4.  $sqlsrv = createSqlServerConnection($sqlSrvConfig['host'], $sqlS
    rvConfig['port'], $sqlSrvConfig['username'], $sqlSrvConfig['pass
    word'], $sqlSrvConfig['database']);
```

Kode 5. 8 Konfigurasi Koneksi Database (2)

Selanjutnya, membuat koneksi pada neo4j dengan variabel bernama neo yang ditunjukan pada baris ke-2. Serta membuat koneksi pada sql server dengan variabel bernama sqlsrv yang ditunjukan pada baris ke-4.

5.2.2. Pembuatan Komponen DB Object getter

Komponen DB Object Getter berfungsi untuk mengambil data objek database dari database RESITS kemudian menyimpanya dalam bentu graph database pada Neo4J. Data mengenai objek database diambil dengan melakukan query dari system catalog.

Hal pertama yang dilakukan adalah mengambil daftar nama server yang berhubungan dengan sistem. Query yang digunaka untuk mengambil informasi server adalah sebagai berikut.

```
1.  //get System Catalog Server
2.  $sql_srv=$sqlsrv-
    >prepare('SELECT name as srv from sys.servers');
3.  $sql_srv->execute();
```

Kode 5. 9 Query Pengambilan Daftar Server

Selanjutnya dilakukan pengambilan daftar database, schema, dan tabel yang ada pada database RESITS. Pengambilan tersebut menggunakan query berikut ini.

```
1.  //Get System Catalog (db,schema,table)
2.  $sql_syscat = $sqlsrv->prepare('
3.      SELECT @@SERVERNAME as srv,DB_NAME(DB_ID()) as db,SCHEMA_NAM
    E(schema_id) as sch, sys.tables.name as tbl FROM sys.tables');
4.  $sql_syscat->execute();
```

Kode 5. 10 Query Pengambilan Daftar Database, Schema, dan Tabel

Setelah itu, hasil query dari *system catalog* disimpan ke dalam graph database pada neo4j dengan menggunakan *chyper query* yang dijalankan dengan library graphaware. *Query* yang dijalankan menggunakan fitur stack. Fitur stack adalah fitur dari library graphaware yang memungkinkan untuk mengumpulkan dulu *query-query* yang akan dijalankan dan menjalankanya secara keseluruhan di akhir bagian. Cara penulisan kode PHP ditunjukkan pada kode 5.11.

```
1.  $stack-
    >push('MERGE(x:Database{name:{name2}, surname:{name}, server:{sr
    v}})',['name'=>$dbname, 'name2' => $db, 'srv' => $srv]);
2.  $stack-
    >push('MATCH (a:Table { surname: {table} }) SET a += {colname}',
     ['table' => $tblname, 'colname' => ['name' => $tbl, 'database'
    => $db, 'server' => $srv, 'schema' => $sch ] ]);
3.  //create relationship Schema to Table
4.  $stack-
    >push('MATCH (a:Schema {surname: {name1} }),(b:Table {surname: {
    name2} }) MERGE (a)-[:Tbl]-
    >(b)',['name1'=>$schname,'name2'=>$tblname]);
5.  $neo->runStack($stack);
```

Kode 5. 11 Kode Pembuatan Node dan Relationship

Kode pada baris 1 digunakan untuk membuat *node* database jika belum terdapat pada graph database. Query chyper ini juga digunakan untuk membuat *node* schema sesuai dengan parameter masing-masing.

Kode pada baris ke-2 digunakan untuk membuat *node* tabel jika belum terdapat pada graph database. Perbedaan dengan chyper query yang digunakan pada baris ke-1 adalah cara menginisiasi property.

Kode pada baris ke-4 digunakan untuk membuat relasi antar objek database. Relasi yang terbentuk dari proses ini adalah srv, db, sch, dan tbl.

Setiap tabel memiliki kolom. Setiap nama-nama kolom akan dijadikan property untuk *node* tabel.

```
1.  $sql_col = $sqlsrv->prepare("
2.      SELECT @@SERVERNAME as srv,DB_NAME(DB_ID()) as db,SCHEMA_NAM
    E(schema_id) as sch, sys.tables.name as tbl,sys.columns.name as
    col FROM sys.tables inner join sys.columns on sys.tables.object_
    id = sys.columns.object_id
3.      ");
4.  $sql_col->execute();
```

Kode 5. 12 Query Mengambil Daftar Kolom

Kode 5.12 adalah query untuk mengambil daftar kolom dari *sys.columns* dan daftar tabel dari *sys.tables*. Setelah itu, dilakukan pengelompokan nama-nama kolom sesuai tabel masing-masing. Pengelompokan kolom memanfaatkan dua array yang berisi daftar nama tabel dan daftar nama kolom. Untuk masing-masing nama tabel pada array akan dicocokkan dengan nama tabel dan nama kolom pada array yang lain dengan menggunakan nested for. Penulisan kode php ditunjukkan dengan kode 5.13 berikut.

```
1.  for ($i=0; $i < count($a) ; $i++) {
2.      $col = '';
3.      for ($j=0; $j < count($b) ; $j++) {
4.          if ($a[$i]['tbl'] === $b[$j]['tbl']) {
5.              $col .= $b[$j]['col']." ";
6.           }
7.      }
8.
9.      $a[$i] = array( 'srv' => $a[$i]['srv'], 'db' =>$a[$i
    ]['db'] , 'sch' =>$a[$i]['sch'] ,'tbl' => $a[$i]['tbl'],
     'col' => $col  );
10. }
```

Kode 5. 13 Kode Perulangan Matching Kolom

Nama-nama kolom untuk setiap tabel yang sudah didapatkan kemudian ditambahkan menjadi *property column* pada *node* tabel pada neo4j dengan menggunakan *chyper query* seperti ditunjukkan kode dibawah ini.

```
1.  $neo-
    >run('MATCH (a:Table { surname: {table}}) SET a += {colname}', [
    'table' => $tbl, 'colname' => ['column' => $col ] ])
```

Kode 5. 14 Kode Penambahan Property Kolom

Setiap tabel juga memiliki *primary key* masing-masing. *Primary key* ini akan disimpan menjadi property pada *node* tabel. Sebelum itu, dilakukan *query* dari *sys.object* untuk mendapatkan daftar *primary key* yang dimiliki masing-masing tabel. Query yang digunakan ditunjukkan pada baris ke-1 – 6 pada Kode 5.15. Primary key kemudian disimpan menjadi *property* pada *node* tabel dengan *chyper query* yang ditunjukkan pada baris ke-13-14 Kode 5.15 berikut.

```
1.  $sql_pk = $sqlsrv->prepare("
2.      SELECT OBJECT_NAME(OBJECT_ID) AS pk,
3.      SCHEMA_NAME(schema_id) AS sch,
4.      OBJECT_NAME(parent_object_id) AS tbl
5.      FROM sys.objects
6.      WHERE type_desc = 'PRIMARY_KEY_CONSTRAINT'");
7.  $sql_pk->execute();
8.
9.  while ($row=$sql_pk->fetch(PDO::FETCH_ASSOC)) {
10.     $pk = $row['pk'];
11.     $tbl = $row['tbl'];
12.
13.     $neo-
    >run('MATCH (a:Table { name: {table} }) SET a += {prop}', ['tabl
    e' => $tbl, 'prop' => ['PK' => $pk ]]);
14. }
```

Kode 5. 15 Kode Penambahan Property Primary Key

Stored procedure dan *function* diambil dari *information schema*. Informasi yang diambil adalah *specific_schema*, *specific_name*, *routine_type*, *routine_body*, *routine_definition*, *sql_data_access*, *created*, dan *last altered*. *Query* untuk mengambil informasi tersebut ditunjukkan pada Kode 5.16 berikut.

```
1.  $sql_sp = $sqlsrv->prepare("
2.      SELECT @@SERVERNAME as srv, SPECIFIC_CATALOG as db, SPECIFIC
    _SCHEMA as sch, SPECIFIC_NAME as sp_name, ROUTINE_TYPE, ROUTINE_
```

```
    BODY, ROUTINE_DEFINITION as sql, SQL_DATA_ACCESS, CREATED, LAST_
    ALTERED
3.          FROM information_schema.routines
4.      ");
5.  $sql_sp->execute();
```

Kode 5. 16 Query untuk Mengambil SP dan Function

Setelah itu, *stored procedure* dan *function* disimpan pada database neo4j melalui *chyper query* yang ditunjukkan Kode 5.17 berikut.

```
1.  $stack-
    >push('MERGE (x:Function {name: {name}, surname:{surname}, serve
    r:{srv}, database:{db}, schema:{sch}, created:{create}, last_alt
    ered:{last}  })',['name' => $name, 'surname' => $surname, 'srv'
    => $srv, 'db' => $db,  'sch' => $sch, 'create' => $create, 'last
    ' => $last]);
2.  $stack-
    >push('MATCH (a:Schema {surname: {sch} }), (b:Function {surname:
     {name} }) MERGE (a)-[:Func]-
    >(b)',['sch'=>$schname,'name'=>$surname]);
```

Kode 5. 17 Chyper Query untuk Menyimpan SP dan Function

Kode pada baris 1 menujukan *chyper query* yang digunakan untuk membuat *node Stored Procedure* dan *Function*. Sedangkan kode pada baris ke-2 menujukan *chyper query* yang digunakan untuk membuat relasi antar SP atau *function* dengan *schema*-nya.

Antar tabel bisa berhubungan satu sama lain melalui *foreign key*. Untuk mendapatkan *foreign key* dan tabel yang berhubungan, dilakukan *query* seperti kode 5.18 berikut.

```
1.  $sql_fk = $sqlsrv->prepare('
2.      SELECT
3.      @@SERVERNAME as srv,DB_NAME(DB_ID()) as db,
4.          obj.name AS fk_rel,
5.          sch2.name AS par_sch,
6.          tab2.name AS par_tbl,
7.          col2.name AS par_col,
8.          sch1.name AS ref_sch,
9.          tab1.name AS ref_tbl,
```

```
10.          col1.name AS ref_col
11.      FROM sys.foreign_key_columns fkc
12.      INNER JOIN sys.objects obj
13.          ON obj.object_id = fkc.constraint_object_id
14.      INNER JOIN sys.tables tab1
15.          ON tab1.object_id = fkc.parent_object_id
16.      INNER JOIN sys.columns col1
17.          ON col1.column_id = parent_column_id AND col1.object_id
    = tab1.object_id
18.      INNER JOIN sys.tables tab2
19.          ON tab2.object_id = fkc.referenced_object_id
20.      INNER JOIN sys.columns col2
21.          ON col2.column_id = referenced_column_id AND col2.object
    _id = tab2.object_id
22.      INNER JOIN sys.schemas sch1
23.          ON tab1.schema_id = sch1.schema_id
24.      INNER JOIN sys.schemas sch2
25.          ON tab2.schema_id = sch2.schema_id');
26. $sql_fk->execute();
```

Kode 5. 18 Query untuk mendapatkan Foreign Key

Foreign key digunakan untuk membuat relasi antar tabel yang berhubungan. Berikut adalah *chyper query* untuk membuat relasi antar tabel yang berhubungan melalui *foreign key*.

```
1.  $neo-
    >run('MATCH (a:Table {surname: {name1} }),(b:Table {surname: {na
    me2} }) MERGE (a)-
    [:fk_rel {FK: {name3}, par_tbl: {name4}, ref_tbl: {name5} }]-
    >(b)',['name1'=>$par_col,'name2'=>$ref_col, 'name3' => $fk, 'nam
    e4' => $par, 'name5' => $ref]);
```

Kode 5. 19 Kode Penambahan Relasi FK

Foreign key, *reference table*, dan *parent table* disimpan menjadi property dari relasi. *Reference table* adalah tabel referensi, yang memiliki FK. *Parent table* adalah tabel asal atau tabel yang memiliki PK.

5.2.3. Pembuatan Komponen SP Splitter

Komponen splitter memiliki fungsi untuk melakukan *pre-processing* pada *SQL Definition* dari *stored procedure* sebelum di-*parsing*.

SQL Definition dari stored procedure bisa memiliki komentar didalamnya. Komentar ini dapat mengganggu proses parsing. Oleh karena itu, komentar harus dihilangkan dengan memanfaatkan regex yang ditunjukkan pada baris ke-3 Kode 5.20.

Setelah itu, kurung siku ([,]) juga dihilangkan untuk mempermudah parsing yang ditunjukkan pada baris ke-5 Kode 5.20.

Selanjutnya, dilakukan mengganti beberapa keyword yang dibutuhkan untuk melakukan parsing. Penggantian ini bertujuan untuk mempermudah parsing karena parser yang dibuat *case sensitive*. Penggantian juga bertujuan untuk menyamakan *syntax* karena dalam penulisan *SQL Definition* oleh database administrator terkadang tidak sama satu sama lain. Keyword yang diganti adalah procedure, proc, from, merge, join, table, into, update, execute, serta exec. Penggantian kata-kata kunci ini memanfaat fungsi str_replace milik PHP yang implementasinya bisa dilihat pada baris ke-7 hingga 16 pada Kode 5.20.

Setelah menghapus komentar, menghapus kurung siku dan mengubah beberapa keyword, *SQL Definition* kemudian dipecah ke dalam array. Pemecahan dilakukan berdasarkan koma (,) , titik koma (;), tanda kurung, kurung siku, dan whitespace. Pemecahan *syntax* memanfaatkan *regex* yang implementasinya dapat dilihat pada baris ke-18 Kode 5.20.

```
1.  class SP_splitter {
2.      public function split($sql){
3.          $uncommented = trim( preg_replace( '@(([\'"]).*?[^\\\]\2
    )|((?:\#|--
    ).*?$|/\*(?:[^/*]|/(?!\*)|\*(?!/)|(?R))*\*\/)\s*|(?<=;)\s+@ms',
    '$1', $sql ) );
4.
5.          $nobracket = str_replace(array('[',']'), '',$uncommented
    );
6.
7.          $rep = str_replace("PROCEDURE", "proc", $nobracket);
8.          $rep2 = str_replace("FROM", "from", $rep);
```

```
9.          $rep3 = str_replace("MERGE", "merge", $rep2);
10.         $rep4 = str_replace("JOIN", "join", $rep3);
11.         $rep5 = str_replace("TABLE", "table", $rep4);
12.         $rep6 = str_replace("INTO", "into", $rep5);
13.         $rep7 = str_replace("UPDATE", "update", $rep6);
14.         $rep8 = str_replace("EXECUTE", "exec", $rep7);
15.         $rep9 = str_replace("EXEC", "exec", $rep8);
16.         $rep10 = str_replace("PROC", "proc", $rep9);
17.
18.         $keywords = preg_split("/[,;(\s)]+/", $rep10);
19.
20.         return $keywords;
21.     }
22. }
```

Kode 5. 20 Kode Komponen SP Splitter

5.2.4. Pembuatan Komponen SP Parser

Komponen parser berfungsi mem-*parsing SQL Definiton* untuk mendapatkan objek database yang berhubungan dengan *stored procedure*. Cara mengidentifikasi objek database adalah dengan menganalisis pola penulisan *syntax* untuk *query-query* yang bisa mengindikasi hubungan stored procedure dengan objek database lain. Analisis pola sudah dibahas pada subbab desain SP Parser .

Parser yang dibuat memanfaatkan berbagai fungsi untuk array PHP. Fungsi array yang digunakan adalah sebagai berikut.

- Array_keys
  Array_keys digunakan untuk mencari index berapa saja yang mengandung *keyword* yang ditentukan. Kembalian dari array_keys adalah array yang berisi index yang mengandung keyword.
- Array_unique
  Array_unique digunakan untuk menghapus *value* yang redundan yang terdapat pada array. Kembalian dari array_unique adalah array yang memiliki value *unique*.

Setelah dilakukan parsing, terkadang kembalian yang muncul tidak sesuai dengan harapan. Untuk mengatasi hal tersebut, dilakukan filter yang terangkum dalam variable bernama $valid. Syarat hasil parsing merupakan objek database adalah sebagai berikut:

- Bukan openquery
- Bukan select
- Bukan set
- Tidak mengandung @
- Mengandung titik (.)
- Mengandung kata ‘sys’

Parser yang dibuat dapat mendeteksi objek database yang berhubungan melalui *syntax* from, join, merge, truncate, insert, update, dan execute. Setiap syntax dibuat menjadi satu method tersendiri. Yang berbeda antara satu method dengan yang lain adalah kata kunci yang dipakai dalam implementasi fungsi array_keys.

Komponen parser hanya bisa menganalis hubungan objek database dari *query* from, join, merge, truncate, insert, update, dan execute. Parses ini belum dapat melakukan *parsing* untuk *query* yang sangat kompleks. Parser ini juga belum mampu untuk membaca openquery.

Implementasi pada kode PHP ditunjukan oleh kode 5.21 berikut.

```
1.  public function from($lexer){
2.      $key_from = array_keys($lexer, "from");
3.          if (!emptyempty($key_from)) {
4.              $from = array();
5.              foreach ($key_from as $key) {
6.                  $tbl_name = $lexer[$key+1];
7.                  $valid = $tbl_name !== 'openquery' && $tbl_name !
    == 'select' && $tbl_name !== 'set' && (strncmp($tbl_name, '@', 1
    ) === 1 && strpos($tbl_name, '.') !== false || strpos($tbl_name,
     'sys') !== false);
8.                      if ($valid) {
9.                          $from[] = $tbl_name;
```

```
                    }
                }
        return array_unique($from);
            }
        }
```

Kode 5. 21 Kode Komponen SP Parser

5.2.5. Pembuatan Komponen Translator

Komponen translator memiliki fungsi untuk melakukan *formatting* nama objek database sebelum disimpan dalam neo4j serta membuat relationship antara stored procedure dan objek database yang berhubungan dengannya.

Sebelumnya, dilakukan konfigurasi file php yang digunakan. File php yang diperlukan adalah SP_Splitter dan SP_Parser. Konfigurasi tersebut ditunjukkan kode 5.22 berikut.

```
use Dependency\Parser\SP_parser;
use Dependency\Parser\SP_splitter;

$sp_pars = new SP_parser();
$sp_split = new SP_splitter();
```

Kode 5. 22 Inisiasi File PHP

Untuk mendapatkan informasi mengenai *stored procedure*, dilakukan *query* dari *information_schema* seperti ditunjukkan pada kode 5.23 berikut.

```
$sql_sp = $sqlsrv->prepare("
            SELECT @@SERVERNAME as srv, SPECIFIC_CATALOG as db, SPECIFIC_SCHEMA as sch, SPECIFIC_NAME as sp_name, ROUTINE_TYPE, ROUTINE_BODY, ROUTINE_DEFINITION as sql, SQL_DATA_ACCESS, CREATED, LAST_ALTERED
                FROM information_schema.routines
                WHERE routine_type = 'PROCEDURE'
            ");
$sql_sp->execute();
```

Kode 5. 23 Query untuk Mendapatkan SP

SQL Definition yang didapatkan kemudian dipecah menjadi array dengan memanfaatkan komponen SP_splitter yang telah

dibuat sebelumnya. Array hasil dari SP_splitter kemudian di-*parsing* dengan menggunakan SP_parser.

```
1.  $raw = $sp_split->split($row['sql']);
```

Kode 4. 20 Penggunaan SP Splitter

Dari array hasil SP Splitter, dapat dilakukan identifikasi nama *stored procedure* yang sedang di-*parsing.* Analisis identifikasi nama *stored procedure* dapat dilihat pada subbab desain SP Parser pada bab 4. Implementasi kode PHP untuk identifikasi nama *stored procedure* dapat dilihat pada Kode 5.24 berikut.

```
1.  $key = array_search("proc", $raw);
2.  $sp = strafter($raw[$key+1],".");
3.  $spname = $row['srv']."."$row['db']."."$row['sch']."."
    .$sp;
```

Kode 5. 24 Kode PHP untuk Identifikasi SP

Setiap objek database hasil *parsing* akan dilakukan *formatting* nama. Analisis format nama telah dibahas pada subbab desain translator di bab 4. Implementasi kode PHP untuk *formatting* nama objek database bisa dilihat pada Kode 5.25 baris ke-5 hingga baris ke-15.

Setelah nama objek database sudah sesuai format, maka akan dibuat relasi antara *stored procedure* dengan objek database yang terkait melalui *query chyper* yang dapat dilihat pada baris ke-18 hingga 21 Kode 5.25. Berikut adalah implementasi kode php untuk komponen translator.

```
1.  $from = $sp_pars->from($raw);
2.  if (isset($from)) {
3.      echo "From: <br>";
4.      foreach ($from as $value) {
5.      if (substr_count($value, "." ) == 0) {
6.          $value = $row['srv']."."$row['db']."."$row['sch']."."
    $value;
7.      }
8.      elseif (substr_count($value, "." ) == 1) {
9.          $value = $row['srv']."."$row['db']."."$value;
10.     }
11.     elseif (substr_count($value, ".") == 2) {
```

```
12.          $value = $row['srv']."."$value;
13.      }
14.      elseif (substr_count($value, ".") == 3) {
15.          $value = $value;
16.      }
17.
18.      $stack-
     >push('MATCH (a:SP {surname: {name1} }), (b:Table {surname:{name
     2} })
19.                  MERGE (a)-[r:Use]->(b)
20.                  ON CREATE SET r.From = "Yes", r.Join = "No", r.M
     erge = "No", r.Truncate = "No", r.Insert = "No", r.Update = "No"

21.                  ON MATCH SET r.From = "Yes"',['name1'=>$spname,'
     name2'=>$value]);
22.      echo $value."<br> \n" ;
23.
24.      }
25.  }
```

Kode 5. 25 Kode Komponen Translator

5.2.6. Pembuatan Komponen Visualizer

Komponen visualizer berfokus pada pembuatan tampilan untuk pengguna. Hal yang dilakukan sebelum implementasi adalah melakukan konfigurasi database untuk neo4j serta mengimpor file css dan javascript yang dibutuhkan. Konfigurasi dapat dilihat pada kode 5.25 dan 5.26 berikut.

```
1.  require_once __DIR__.'\hihi.php';
2.  require_once __DIR__.'\SP_vis.php';
3.  use Dependency\Components\SP_vis;
4.
5.  $databaseConfig = require config_path('database.php');
6.
7.  $neo4jConfig = $databaseConfig['connections']['neo4j']['sp'];
8.  $neo = createNeo4jConnection($neo4jConfig['username'], $neo4jCon
    fig['password'], $neo4jConfig['host'], $neo4jConfig['port']);
```

Kode 5. 26 Konfigurasi Koneksi Neo4j Komponen Visualizer

```
<link rel="stylesheet" type="text/css" href="assets/css/neo4jd3.min.css">
<link rel="stylesheet" type="text/css" href="assets/css/font-awesome.min.css">
<link rel="stylesheet" href="assets/css/material-kit.css?v=2.0.3">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.1.0/css/all.css" integrity="sha384-lKuwvrZot6UHsBSfcMvOkWwlCMgc0TaWr+30HWe3a4ltaBwTZhyTEggF5tJv8tbt" crossorigin="anonymous">
<script type="text/javascript" src='assets/js/d3.min.js'></script>
<script type="text/javascript" src='assets/js/neo4jd3.js'></script>
   <style>
       body,
       html,
       .neo4jd3 {
           height: 100%;
           overflow: hidden;
       }
   </style>
<link rel="icon" href="assets/img/server_Iq4_icon.ico">
```

Kode 5. 27 Konfigurasi CSS dan Javascript yang Dibutuhkan

Komponen visualisasi ini mengimpor SP_vis.php untuk mendapatkan daftar *stored porcedure* yang terdapat pada database sql server. *Query* yang dijalankan pada SP_vis.php adalah sebagai berikut.

```
$sql_sp = $sqlsrv->prepare("
        SELECT @@SERVERNAME as srv, SPECIFIC_CATALOG as db, SPECIFIC_SCHEMA as sch, SPECIFIC_NAME as sp_name
                FROM information_schema.routines
                WHERE routine_type = 'PROCEDURE'
                ORDER BY sp_name ASC
            ");
$sql_sp->execute();
```

Kode 5. 28 Query untuk Mendapatkan daftar SP

Setelah konfigurasi selesai, selanjutnya adalah membuat form untuk filter. Filter yang tersedia adalah filter jenis relasi, nama *stored procedure*, serta jumlah hops atau radius relasi. Nilai

minimal hops adalah satu dan maksimal dua. Implementasi kode php untuk form filter ditunjukkan kode 5.28.

Baris ke-3 hingga 16 merupakan kode HTML untuk pembuatan filter show. Baris ke-17 hingga 33 merupakan kode HTML untuk pembuatan filter stored procedure. Sedangkan baris ke-34 hingga 44 merupakan kode untuk pembuatan filter radius.

```
1.  <form method="post">
2.  <div class="form-group">
3.      <label>Show</label>
4.      <select class="form-control" name="show" id="show">
5.          <option value="All" <?php if (isset($show) && $show == 'All') {
6.              echo 'selected';
7.          } ?> >All</option>
8.          <option value="Execute" <?php if (isset($show) && $show == 'Execute
    ') {
9.              echo 'selected';
10.         } ?>>Execute</option>
11.         <option value="Use" <?php if (isset($show) && $show == 'Use') {
12.             echo 'selected';
13.         } ?>>Use</option>
14.
15.     </select>
16. </div><br>
17. <div class="form-group">
18.     <label for="exampleFormControlSelect1">Stored Procedure</label>
19.     <select class="form-control" name="sp_select">
20.         <option value="All">Select Procedure (All) </option>
21.         <?php
22.             foreach ($sp as $value) {
23.                 $opt = $value['sp_name'];
24.                 $val = $value['srv']."."$value['db']."."$value['sch']."."
    .$value['sp_name'];
25.         ?>
26.       <option value="<?= $val; ?>" <?php if (isset($sp_select) && $sp_selec
    t == $val) {
27.             echo 'selected';
28.         } ?>><?= $opt; ?></option>
29.       <?php
30.             }
31.       ?>
32.     </select>
33. </div><br>
```

```
34.  <div class="form-group">
35.      <label>Radius</label>
36.      <select class="form-control" name="hops" id="hops">
37.          <option value="1" <?php if (isset($hops) && $hops == '1') {
38.              echo 'selected';
39.          } ?> >1</option>
40.          <option value="2" <?php if (isset($hops) && $hops == '2') {
41.              echo 'selected';
42.          } ?>>2</option>
43.      </select>
44.  </div><br>
45.  <div>
46.      <div>
47.      <button type='submit' class='btn btn-primary btn-round pull-
     right' name='btn-gen' id="btn-gen">GENERATE</button>
48.      </div>
49.  </div>
50.  </form>
51.
```

Kode 5. 29 Kode Form *Filter*

Masukan filter digunakan sebagai dasar untuk melakukan *query chyper* pada neo4j untuk mendapatkan data. Masing-masing filter akan menjalankan *query* yang berbeda-beda.

Implementasi kode php ditunjukkan pada kode 5.30 hingga 5.32.

```
1.  $resnode = $neo-
    >run('MATCH (n) RETURN n.name as name,n.surname as surname,n.ser
    ver as server, n.database as database, n.schema as schema, n.PK
    as PK, n.column as column ,n.created as created, n.last_altered
    as last_altered, id(n) as id,labels(n) as label');
2.  $resrel = $neo->run('MATCH (a)-[r]-
    >(b) return id(r) as id,id(a) as start,id(b) as end, type(r) as
    type, r.FK as FK, r.From as from, r.Insert as insert, r.Join as
    join, r.Merge as merge, r.Truncate as truncate, r.Update as upda
    te, a.name as node_from, b.name as node_to');
```

Kode 5. 30 Kode Proses *Filtering* (1)

Kode 5.30 merupakan *chyper query* yang digunakan ketika filter show dan filter stored procedure yang dipilih adalah All.

```
1.  $resnode = $neo->run('MATCH (a)-[r:Use]-
    >(b) RETURN DISTINCT a.name as name, a.surname as surname, a.ser
    ver as server, a.database as database, a.schema as schema, a.PK
    as PK, a.column as column ,a.created as created, a.last_altered
    as last_altered, id(a) as id,labels(a) as label');
2.  $resnode2 = $neo->run('MATCH (a)-[r:Use]-
    >(b) RETURN DISTINCT b.name as name, b.surname as surname, b.ser
    ver as server, b.database as database, b.schema as schema, b.PK
    as PK, b.column as column ,b.created as created, b.last_altered
    as last_altered, id(b) as id,labels(b) as label');
3.  $resrel= $neo->run('MATCH (a)-[r:Use]-
    >(b) RETURN id(r) as id,id(a) as start,id(b) as end, type(r) as
    type, r.FK as FK, r.From as from, r.Insert as insert, r.Join as
    join, r.Merge as merge, r.Truncate as truncate, r.Update as upda
    te, a.name as node_from, b.name as node_to');
```

Kode 5. 31 Kode Proses *Filtering* (2)

Kode 5.31 merupakan *chyper query* yang dijalankan ketika filter show yang dipilih adalah use atau execute serta filter stored procedure yang dipilih adalah All.

```
1.  $resnode = $neo->run('MATCH path = (a:SP {surname:{sp}})-
    [r*..'.$hops.']-(b)
2.      WITH startnode(LAST(r)) as x
3.      RETURN DISTINCT x.name as name,x.surname as surname,x.server
     as server, x.database as database, x.schema as schema, x.PK as
    PK, x.column as column ,x.created as created, x.last_altered as
    last_altered, id(x) as id,labels(x) as label',['sp' => $sp_selec
    t]);
4.  $resnode2=$neo->run('MATCH path = (a:SP {surname: {sp} })-
    [r*..'.$hops.']-(b)
5.      WITH endnode(LAST(r)) as y
6.      RETURN DISTINCT y.name as name,y.surname as surname,y.server
     as server, y.database as database, y.schema as schema, y.PK as
    PK, y.column as column ,y.created as created, y.last_altered as
    last_altered, id(y) as id,labels(y) as label',['sp' => $sp_selec
    t]);
7.  $resrel = $neo->run('MATCH path = (a:SP {surname: {sp} })-
    [r*..'.$hops.']-(b)
8.      WITH LAST(r) as lr, startnode(LAST(r)) as x, endnode(LAST(r)
    ) as y
9.      RETURN id(lr) as id, id(x) as start,id(y) as end, type(lr) a
    s type, lr.FK as FK, lr.From as from, lr.Insert as insert, lr.Jo
    in as join, lr.Merge as merge, lr.Truncate as truncate, lr.Updat
```

```
    e as update, x.name as node_from, y.name as node_to',['sp' => $s
    p_select]);
```

Kode 5. 32 Kode Proses *Filtering* (3)

Kode 5.32 merupakan *chyper query* yang dijalankan ketika filter *stored procedure* dan filter radius dipilih.

Berikutnya setelah query berhasil dieksekusi, dibentuk JSON sebagai masukan untuk visualisasi. Setiap hasil eksekusi query diiterasi untuk mendapatkan *node* dan relasinya. *Node* disimpan dalam array bernama $resnode dan $resnode2. Sedangkan relasi disimpan dalam array bernama $resrel. Array-array ini kemudian dijadikan variable bertipe JSON dengan format yang sesuai dengan masukan untuk library neo4jd3.js Kode implementasi ditunjukan oleh Kode 5.33 hingga Kode 5.36 berikut.

```
1.  foreach ($resnode->getRecords() as $record) {
2.          $property = array(
3.              "name"=> $record->value('name'),
4.              "surname" => $record->value('surname'),
5.              "server" => $record->value('server'),
6.              "database" => $record->value('database'),
7.              "schema" => $record->value('schema'),
8.              "column" => $record->value('column'),
9.              "PK" => $record->value('PK'),
10.             "created" => $record->value('created'),
11.             "last altered" => $record->value('last_altered')
12.         );
13.         $filtered = array_filter($property);
14.
15.         $nodes[] = ["id"=>$record->value('id'),
16.                     "labels"=>$record->value('label'),
17.                     "properties"=>
18.                         $filtered
19.                         //taruh value lain di sini (jika ada)
20.                     ];
21. }
```

Kode 5. 33 Kode Pembuatan Metadata (1)

Kode 5.33 diatas merupakan implementasi kode untuk pembuatan metadata dari data *node* yang didapatkan dari *chyper query* yang berhasil dijalankan.

```
if ($resnode2 != '') {
    foreach ($resnode2->getRecords() as $record) {
        $property = array(
            "name"=> $record->value('name'),
            "surname" => $record->value('surname'),
            "server" => $record->value('server'),
            "database" => $record->value('database'),
            "schema" => $record->value('schema'),
            "column" => $record->value('column'),
            "PK" => $record->value('PK'),
            "created" => $record->value('created'),
            "last altered" => $record->value('last_altered')
        );
        $filtered = array_filter($property);

        $nodes[] = ["id"=>$record->value('id'),
                    "labels"=>$record->value('label'),
                    "properties"=>
                        $filtered
                        //taruh value lain di sini (jika ada)
                    ];
}
}
```

Kode 5. 34 Kode Pembuatan Metadata (2)

Kode 5.34 diatas merupakan implementasi kode untuk pembuatan metadata dari data *node* tujuan yang didapatkan dari *chyper query* yang berhasil dijalankan. Kode ini akan dijalankan ketika *filter* show yang dipilih adalah All.

```
foreach ($resrel->getRecords() as $record) {
    $fk = array(
        "FK" => $record->value('FK'),
        "From" => $record->value('from'),
        "Insert" => $record->value('insert'),
        "Join" => $record->value('join'),
        "Merge" => $record->value('merge'),
        "Truncate" => $record->value('truncate'),
        "Update" => $record->value('update'),
```

```
10.         "node_from" => $record->value('node_from'),
11.         "node_to" => $record->value('node_to')
12.     );
13.     $filtered_fk = array_filter($fk);
14.       $rel[] = ["id"=>$record->value('id'),
15.                 "type"=>$record->value('type'),
16.                 "startNode"=>$record->value('start'),
17.                 "endNode"=>$record->value('end'),
18.                 "properties"=>
19.                     $filtered_fk
20.                 ];
21.     }
```

Kode 5. 35 Kode Pembuatan Metadata (3)

Kode 5.35 diatas merupakan implementasi kode untuk pembuatan metadata dari data relasi yang didapatkan dari *chyper query* yang berhasil dijalankan.

```
1.  if (isset($nodes) && isset($rel)) {
2.      $json = ["results" => array([
3.              "data" => array([
4.                  "graph" => array(
5.                      "nodes" => $nodes,
6.                      "relationships" => $rel
7.                      )])])];
8.      $result = json_encode($json);
9.  }
10. else{
11.    echo '<script type="text/javascript">';
12.    echo 'alert("Tidak ada relasi terkait")';
13.    echo '</script>';
14. }
```

Kode 5. 36 Kode Pembuatan Metadata (3)

Kode 5.36 baris ke 1-9 menunjukan implementasi kode untuk pembuatan JSON dari data yang dihasilkan oleh Kode 5.34 dan Kode 5.35.

Jika query mengembalikan hasil kosong, maka aplikasi akan menanganinya dengan menampilkan pesan peringatan. Implementasi penanganan error ini terdapat pada Kode 5.36 baris ke-10 hingga 13.

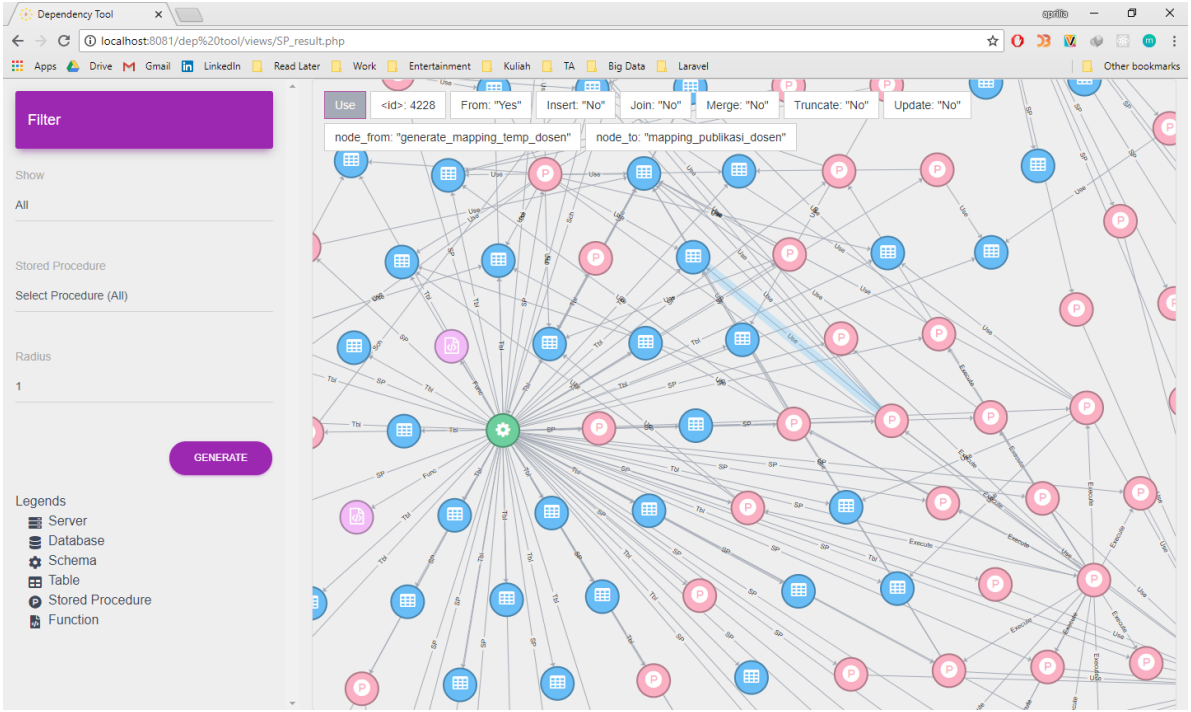
Selanjutnya adalah bagian kode untuk menampilkan *graph*. Untuk menampilkan *graph* memanfaatkan library neo4jd3.js. Library tersebut membutuhkan variabel bertipe JSON yang berisi data *node* dan relasi sebagai masukan. Neo4jd3.js juga dapat melakukan beberapa konfigurasi seperti konfigurasi *icon* yang ingin digunakan, radius untuk *node*, jarak antar *node*, *node* yang di-*highlight*, serta fungsi-fungsi lainya. Implementasi kode tampilan *graph* terdapat pada Kode 5.38 berikut ini.

```
<script type="text/javascript">
var neo4jd3 = new Neo4jd3('#neo4jd3', {
    icons: {
        'Server': 'server',
        'Database': 'database',
        'Schema': 'gear',
        'Table': 'table',
        'Column': 'columns',
        'SP' : 'f288',
        'Function' : 'f1c9'
    },
    minCollision: 60,
    neo4jData: <?= $result; ?> ,
    nodeRadius: 20,
    highlight:
    [
        {
            class: 'SP',
            property: 'surname',
            value: '<?= $sp_select; ?>',
        }
    ],
    zoomFit: true,
    });
</script>
```

Kode 5. 37 Kode untuk Tampilan *Graph*

Terakhir adalah implementasi antarmuka aplikasi. Pembahasan antarmuka aplikasi terdapat pada subbab desain antarmuka pada bab 4. Implementasi bagian antarmuka aplikasi sesuai dengan Gambar 4.7. Pada implementasinya, ditambahkan keterangan *icon* yang digunakan untuk merepresentasi *node* pada tampilan *graph*. Pada sisi kiri terdapat menu filter yang dapat dipilih

pengguna. Filter yang tersedia adalah filter berdasarkan relasi, filter berdasarkan Stored Procedure, dan pilihan hops atau radius relasi. Sisi kanan merupakan tampilan *graph* yang sesuai dengan filter yang dipilih oleh pengguna. Pada bagian bawah filter terdapat keterangan *icon* yang digunakan untuk merepresentasikan setiap *node*. Hasil implementasi antarmuka aplikasi dapat dilihat pada Gambar 5.1 berikut.



Gambar 5. 1 Implementasi Visualisasi

5.2.7. Konfigurasi Otomasi

Konfigurasi otomasi menjelaskan mengenai pengaturan pada *task scheduler* untuk menjalankan program atau *script* untuk proses extraction dan parsing secara otomatis. Konfigurasi ini menggunakan *file* extraction.bat untuk menjalankan proses extraction dan *file* parsing.bat untuk menjalankan proses parsing pada background proses. Implementasi kode extraction.bat dan parsing.bat dapat dilihat pada kode 5.39 dan 5.40.

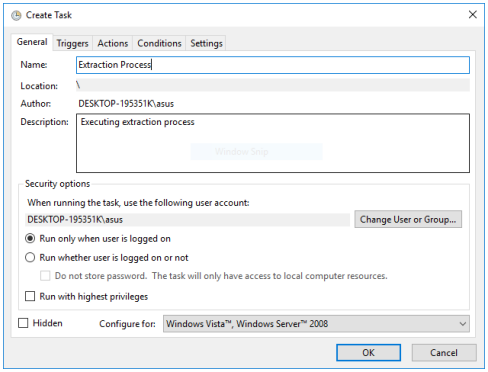
```
1.   php -f C:\xampp\htdocs\dep tool\app\Components\SP_syscat.php
```

Kode 5. 38 Kode Program File extraction.bat

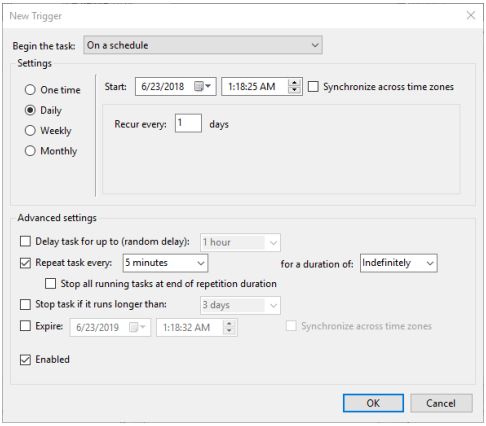
```
1.   php -f C:\xampp\htdocs\dep tool\app\Components\SP_parsing.php
```

Kode 5. 39 Kode Program File parsing.bat

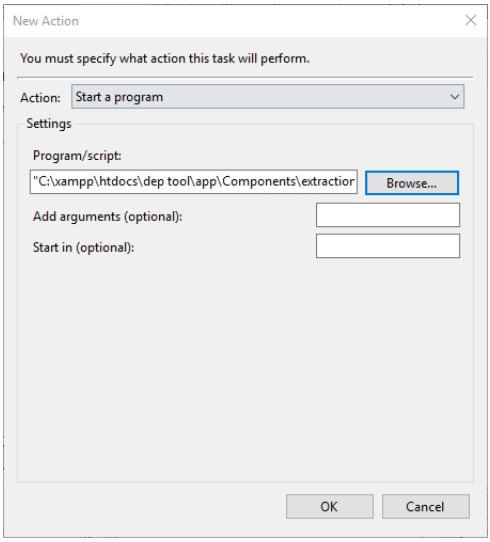
Selanjutnya kedua file .bat ini diatur agar dijalankan setiap lima menit secara otomatis. Karena lingkungan implementasi aplikasi menggunakan windows, maka penjadwalan dilakukan melalui *task scheduler*. Implementasi konfigurasi pada *task scheduler* terdapat pada Gambar 5.2 - 5.7 berikut ini.



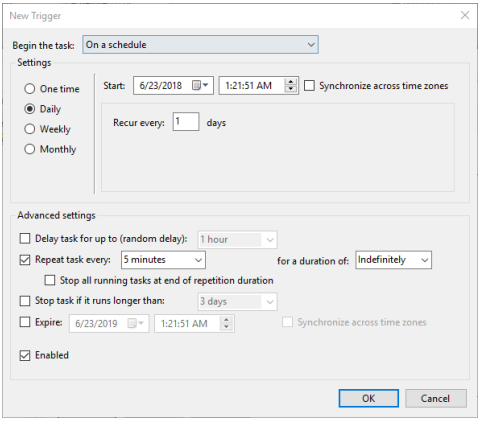
Gambar 5. 2 Konfigurasi Task Extraction.bat



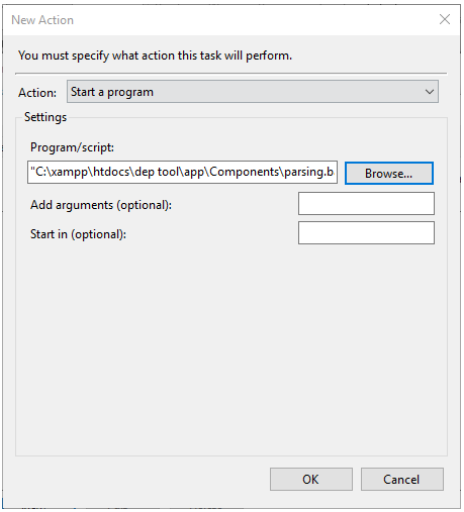
Gambar 5. 3 Konfigurasi Trigger Extraction.bat



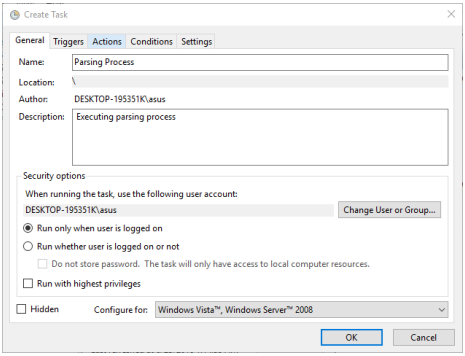
Gambar 5. 4 Konfigurasi Action Extraction.bat



Gambar 5. 5 Konfigurasi Task Parsing.bat



Gambar 5. 6 Konfigurasi Trigger Parsing.bat



Gambar 5. 7 Konfigurasi Action Parsing.bat

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan mengenai hasil pengujian dan pembahasan hasil pengujian dari aplikas. Pembahasan mengenai hasil pengujian aplikasi mencakup hasil dari implementasi yang telah dilakukan pada bab sebelumnya. Penjelasan lebih detail mengenai hasil pengujian dan pembahasan hasil pengujian terdapat pada subbab-subbab berikut ini.

6.1. Data Hasil Pengujian

Data hasil pengujian dibagi menjadi dua yaitu data hasil pengujian performa aplikasi dan data hasil pengujian fungsionalitas. Hal ini sesuai dengan rancangan pengujian aplikasi yang telah dibahas pada subbab 4.6.

6.1.1. Pengujian Performa Aplikasi

Pengujian performa aplikasi bertujuan untuk mengetahui rata-rata waktu eksekusi dari proses extraction dan proses parsing untuk jumlah data tertentu. Data hasil pengujian performa aplikasi adalah rata-rata waktu eksekusi yang dibutuhkan untuk masing-masing proses. Rata-rata waktu proses diambil dari percobaan sebanyak lima kali untuk masing-masing proses. Jumlah data yang diolah pada proses extraction sebanyak 870 objek dengan uraian ditunjukan dengan tabel berikut.

Tabel 6. 1 Rincian Jumlah Objek Database

Objek Database	Jumlah
Server	14
Database	1
Schema	3
Tabel	64
Kolom	707
Stored Procedure	59
Function	2
Primary Key	33

Komponen pada proses extraction dapat mengeksekusi rata-rata 18,94 detik. Sedangkan proses parsing melakukan parsing terhadap 59 stored procedure yang terdapat pada database. Proses parsing rata-rata membutuhkan waktu eksekusi selama 6,36 detik. Waktu eksekusi kedua proses untuk setiap percobaan dapat dilihat pada Tabel 6.2.

Tabel 6. 2 Data Hasil Pengujian Performa

Percobaan ke-	Proses	
	Extraction (s)	**Parsing (s)**
1	22.51	6.32
2	18.56	6.90
3	18.81	6.57
4	19.25	6.21
5	15.59	5.83
Rata-rata	**18.94**	**6.36**

6.1.2. Pengujian Fungsionalitas

Data hasil pengujian fungsionalitas aplikasi *Stored Procedure Dependency Tool* menunjukkan hasil yang sesuai dengan yang diharapkan untuk semua kasus pengujian. Kasus pengujian yang memiliki kode S untuk komponen SP Parser. Kasus pengujian yang memiliki kode V untuk komponen Visualizer. Data hasil pengujian fungsionalitas dapat dilihat pada Tabel 6.3 dan Tabel 6.4. Dari data tersebut dapat dilihat bahwa seluruh pengujian untuk komponen berhasil dilakukan dan menujukkan status sukses.

6.1.2.1 Hasil Pengujian Komponen SP Parser

Pada komponen SP Parser dilakukan pengujian dalam proses *parsing* yang dilakukan oleh komponen ini. Tujuan dari pengujian ini adalah mengetahui akurasi hasil *parsing* dari komponen SP Parser yang dibuat. Hasil pengujian komponen SP Parser dapat dilihat pada Tabel 6.2 berikut ini.

Tabel 6. 3 Hasil Pengujian Komponen SP Parser

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan
S01	Mendeteksi objek database yang berhubungan from	SP: insert_ppm_penelitian	Tabel: • its-dw.ppm.penelitian • resits.dbo.tran_publikasi_dosen_tetap	Tabel: • its-dw.ppm.penelitian resits.dbo.tran_publikasi_dosen_tetap
		Pengujian berhasil dilakukan.		
		SP: generate_jumlah_scopus	Tabel: • its-dw.ppm.scopus_jurnal • its-dw.ppm.scopus_seminar • resits.dbo.tmst_pegawai	Tabel: • its-dw.ppm.scopus_jurnal • its-dw.ppm.scopus_seminar resits.dbo.tmst_pegawai
		Pengujian berhasil dilakukan.		
S02	Mendeteksi objek database yang berhubungan join	SP: mapping_fakultas_jurusan_recent	Tabel: • its-dw.akademik.fakultas • resits.dbo.tmst_fakultas • resits.dbo.tmst_jurusan	Tabel: • its-dw.akademik.fakultas • resits.dbo.tmst_fakultas resits.dbo.tmst_jurusan
		Pengujian berhasil dilakukan.		

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan
		SP: mapping_dosen_recent	Tabel: resits.dbo.tmst_pegawai	Tabel: resits.dbo.tmst_pegawai
		Pengujian berhasil dilakukan.		
S03	Mendeteksi objek database yang berhubungan merge	SP: mapping_dosen_recent	Tabel: resits.dbo.mapping_dosen	Tabel: resits.dbo.mapping_dosen
		Pengujian berhasil dilakukan.		
		SP: insert_jurusan_itsdw	Tabel: resits.dbo.tmst_jurusan_baru	Tabel: resits.dbo.tmst_jurusan_baru
		Pengujian berhasil dilakukan.		
S04	Mendeteksi objek database yang berhubungan truncate	SP: generate_mapping_temp_dosen	Tabel: resits.dbo.mapping_temp_dosen	Tabel: resits.dbo.mapping_temp_dosen
		Pengujian berhasil dilakukan.		
		SP: tran_temp_publikasi_recent	Tabel: • resits.dbo.tran_temp_publikasi • resits.dbo.tran_temp_publikasi_distinct • resits.dbo.tran_temp_pu	Tabel: • resits.dbo.tran_temp_publikasi • resits.dbo.tran_temp_publikasi_distinct resits.dbo.tran_temp_publikasi_mapping

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan
			blikasi_mapping	
		Pengujian berhasil dilakukan.		
S05	Mendeteksi objek database yang berhubungan insert	SP: insert_publikasi_itsdw_buku	Tabel: resits.dbo.tran_publikasi_dosen_tetap	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		Pengujian berhasil dilakukan.		
		SP: insert_publikasi_itsdw_seminar	Tabel: resits.dbo.tran_publikasi_dosen_tetap	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		Pengujian berhasil dilakukan.		
S06	Mendeteksi objek database yang berhubungan update	SP: insert_publikasi_itsdw_seminar	Tabel: resits.dbo.tran_publikasi_dosen_tetap	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		Pengujian berhasil dilakukan.		
		SP: insert_publikasi_itsdw_jurnal	Tabel: resits.dbo.tran_publikasi_dosen_tetap	Tabel: resits.dbo.tran_publikasi_dosen_tetap
		Pengujian berhasil dilakukan.		
S07	Mendeteksi objek database yang	SP: master_run	SP: • insert_fakultas_itsdw • insert_jurusan_itsdw	SP: • insert_fakultas_itsdw • insert_jurusan_itsdw

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan
	berhubungan execute		• insert_ppm_pegawai_dosen • insert_laboratorium_ppm • insert_ppm_buku • insert_ppm_penelitian • insert_publikasi_itsdw_jurnal_gs • insert_publikasi_itsdw_jurnal_scopus • insert_publikasi_itsdw_paten • insert_ppm_pengabdian_masyarakat • insert_publikasi_itsdw_seminar_gs • insert_publikasi_itsdw_seminar_scopus • distinct_publikasi_dosen_tetap	• insert_ppm_pegawai_dosen • insert_laboratorium_ppm • insert_ppm_buku • insert_ppm_penelitian • insert_publikasi_itsdw_jurnal_gs • insert_publikasi_itsdw_jurnal_scopus • insert_publikasi_itsdw_paten • insert_ppm_pengabdian_masyarakat • insert_publikasi_itsdw_seminar_gs • insert_publikasi_itsdw_seminar_scopus • distinct_publikasi_dosen_tetap

ID	Cara Pengujian	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan
			• tran_temp_publikasi_recent • generate_mapping_temp_dosen • generate_mapping_publikasi_dosen • update_laboratorium_fakjur_itsdw • mapping_dosen_recent	• tran_temp_publikasi_recent • generate_mapping_temp_dosen • generate_mapping_publikasi_dosen • update_laboratorium_fakjur_itsdw mapping_dosen_recent
		Pengujian berhasil dilakukan.		
		SP: sp_helpdiagrams	Tidak memunculkan hasil	Tidak memunculkan hasil
		Pengujian berhasil dilakukan.		

6.1.2.2 Hasil Pengujian Komponen Visualizer

Pada komponen visualizer dilakukan pengujian dalam proses penyajian data. Tujuan dari pengujian ini adalah mengetahui apakah komponen visualizer dapat memproses dan menyajikan visualisasi dari data yang ada pada neo4j. Hasil pengujian komponen visualizer dapat dilihat pada Tabel 6.4 berikut ini.

Tabel 6. 4 Hasil Pengujian Komponen Visualizer

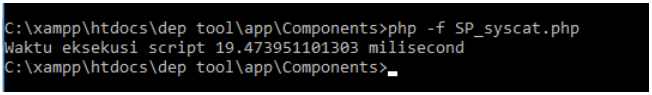
ID	Masukan	Hasil yang Diharapkan	Hasil yang Didapatkan	Status
V01	Hasil Chyper Query pada Neo4J	Aplikasi berhasil membuat data yang berformat JSON untuk *node* dan relasinya	Aplikasi berhasil membuat data yang berformat JSON untuk *node* dan relasinya	Sukses
V02	-	Aplikasi menampilkan form filter	Aplikasi menampilkan form filter	Sukses
V03	JSON berisi graph data	Aplikasi menampilkan *graph* sesuai dengan data.	Aplikasi menampilkan *graph* sesuai dengan data.	Sukses
V04	JSON dan form filter	Aplikasi menampilkan *graph* sesuai filter yang dipilih pengguna	Aplikasi menampilkan *graph* sesuai filter yang dipilih pengguna	Sukses
V05	Query Chyper dari Neo4J	Aplikasi menampilkan pesan peringatan ketika terjadi error	Aplikasi menampilkan pesan peringatan ketika terjadi error	Sukses

6.2. Pembahasan Hasil Pengujian

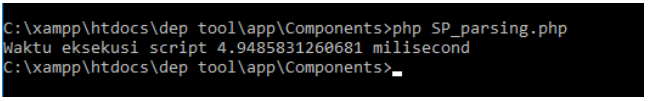
Pada bagian ini akan dibahas mengenai hasil pengujian dari aplikasi *Stored Procedure Dependency Tool*. Hal-hal yang dibahas adalah keterangan atau penjelasan terkait hasil dari pengujian seperti bukti pengujian, skenarion pengujian, dan keluaran dari aplikasi. Subbab ini akan dibagai menjadi dua bagian yang akan membahas pengujian perfoma aplikasi dan pengujian fungsionalitas aplikasi. Penjelasan lebih detail dapat dilihat pada subbab berikut ini.

6.2.1. Pengujian Performa Aplikasi

Pembahasan hasil pengujian performa aplikasi berfokus pada proses yang terdapat pada aplikasi *Stored Procedure Dependency Tool*. Proses yang menjalani pengujian perfoma aplikasi adalah proses *extraction* dan *parsing*. Pengujian dilakukan sebanyak lima kali dan didapatkan rata-rata waktu eksekusi yang telah dijelaskan pada subbab 6.1.1. Informasi hasil eksekusi untuk proses *extraction* dapat dilihat pada Gambar 6.1. Sedangkan informasi mengenai hasil eksekusi untuk proses *parsing* dapat dilihat pada Gambar 6.2.



Gambar 6. 1 Informasi waktu eksekusi proses Extraction



Gambar 6. 2 Informasi waktu eksekusi proses Parsing

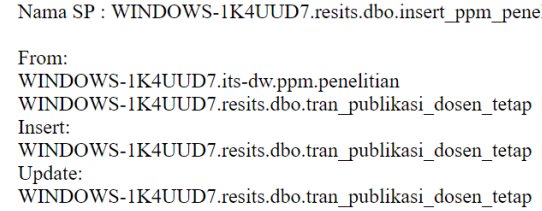
6.2.2. Pengujian Fungsionalitas

Pembahasan mengenai hasil pengujian fungsionalitas berfokus pada penjabaran masing-masing kasus pengujian. Pembahasan kali ini bertujuan untuk memeberi penjelasan mengenai hasil mendetail setiap kasus pengujian. Pada masing-masing kasus pengujian komponen SP Parser diberikan dua skenario yang berbeda. Penjelasan masing-masing kasus dapat dilihat lebih detail pada subbab-subbab berikut ini.

6.2.2.1 Pembahasan Hasil Pengujian S01

Kasus pengujian S01 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan from. Pengujian kali ini menggunakan dua stored procedure yang dapat dilihat pada Kode A.1 dan Kode A.2 pada lampiran.

Skenario pertama pengujian ini menggunakan stored procedure yang terdapat pada Kode A.1. Pada stored procedure tersebut terdapat query from. Query from dapat digunakan untuk mengidentifikasi tabel yang berhubungan dengan stored procedure. Setelah dilakukan *parsing*, hasil yang didapatkan ditunjukan dengan Gambar 6.3 berikut.

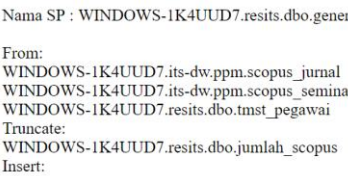


Gambar 6. 3 Hasil Pengujian S01 skenario pertama

Gambar diatas menunjukkan bahwa SP Parser dapat mengembalikan nama objek database yang berhubungan melalui from sesuai dengan hasil yang diharapkan pada pengujian. Hal ini menunjukkan bahwa SP Parser dapat mendeteksi objek database yang berhubungan melalui from

dengan tepat dan kasus pengujian S01 skenario pertama berhasil dilakukan.

Skenario kedua menggunakan stored procedure pada Kode A.2 sebagai masukan. Hasil *parsing* yang didapatkan dapat dilihat pada gambar berikut.



Gambar 6. 4 Hasil Pengujian S01 skenario kedua

Gambar 6.4 menunjukkan bahwa SP Parser dapat mengembalikan nama objek database yang berhubungan melalui from sesuai dengan hasil yang diharapkan pada pengujian. Hal ini menunjukkan bahwa SP Parser dapat mendeteksi objek database yang berhubungan melalui from dengan tepat dan kasus pengujian S01 skenario pertama berhasil dilakukan.

6.2.2.2 Pembahasan Hasil Pengujian S02

Kasus pengujian S02 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan join. Stored procedure yang digunakan pada pengujian ini dapat dilihat pada Kode A.3 dan Kode A.4 yang terdapat pada lampiran.

Skenario pertama pengujian kali ini menggunakan stored procedure pada Kode A.3 sebagai masukan. Stored procedure tersebut memiliki query join. Query join dapat digunakan untuk mengidentifikasi objek database yang berhubungan dengan stored procedure. Hasil *parsing* dari stored procedure dapat dilihat pada Gambar 6.5 berikut ini.

Nama SP : WINDOWS-1K4UUD7.resits.deleted.mapping_fakultas_jurusan_recent

From:
WINDOWS-1K4UUD7.its-dw.akademik.jurusan
Join:
WINDOWS-1K4UUD7.its-dw.akademik.fakultas
WINDOWS-1K4UUD7.resits.dbo.tmst_fakultas
WINDOWS-1K4UUD7.resits.dbo.tmst_jurusan
Merge:
WINDOWS-1K4UUD7.resits.dbo.mapping_fakultas_jurusan
Update:

Gambar 6. 5 Hasil Pengujian S02 skenario pertama

Gambar menunjukkan bahwa SP Parser dapat mengembalikan array yang berisi nama objek database yang berhubungan melalui query join dengan tepat sesuai dengan hasil yang diharapkan dari pengujian. Hal ini menunjukkan bahwa pengujian S02 skenario pertama berhasil dilakukan.

Skenario kedua menggunakan stored procedure pada Kode A.4 sebagai masukan. Hasil *parsing* dapat dilihat pada gambar berikut ini.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.mapping_dosen_recent

From:
WINDOWS-1K4UUD7.its-dw.kepegawaian.profil_dosen
WINDOWS-1K4UUD7.simpel.dbo.peneliti'
Join:
WINDOWS-1K4UUD7.resits.dbo.tmst_pegawai
Merge:
WINDOWS-1K4UUD7.resits.dbo.mapping_dosen
Update:

Gambar 6. 6 Hasil Pengujian S02 skenario kedua

Gambar menunjukkan bahwa SP Parser dapat mengembalikan array yang berisi nama objek database yang berhubungan melalui query join dengan tepat sesuai dengan analisis yang dilakukan sebelumnya. Hal ini menunjukkan bahwa pengujian S02 skenario kedua berhasil dilakukan.

6.2.2.3 Pembahasan Hasil Pengujian S03

Kasus pengujian S03 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang

berhubungan merge. Stored procedure yang digunakan untuk pengujian ini dapat dilihat pada Kode A.4 dan Kode A.5 pada lampiran.

Skenario pertama menggunakan stored procedure pada Kode A.4. Pada stored procedure tersebut terdapat query merge. Query merge dapat digunakan untuk mengidentifikasi hubungan objek database dengan stored procedure. Setelah dilakukan *parsing*, didapatkan hasil seperti Gambar 6.7 berikut.

```
Nama SP : WINDOWS-1K4UUD7.resits.dbo.mapping_dosen_recent

From:
WINDOWS-1K4UUD7.its-dw.kepegawaian.profil_dosen
WINDOWS-1K4UUD7.simpel.dbo.peneliti'
Join:
WINDOWS-1K4UUD7.resits.dbo.tmst_pegawai
Merge:
WINDOWS-1K4UUD7.resits.dbo.mapping_dosen
Update:
```

Gambar 6. 7 Hasil Pengujian S03 skenario pertama

Gambar diatas menujukan bahwa parser dapat mengembalikan nama objek database yang berhubungan merge dengan stored procedure. Hal ini membuktikan bahwa pengujian S03 skenario pertama berhasil dilakukan.

Skenario kedua menggunakan stored procedure pada Kode A.5. Hasil *parsing* dari stored procedure tersebut dapat dilihat pada Gambar 6.8 berikut.

```
Nama SP : WINDOWS-1K4UUD7.resits.dbo.insert_jurusan_itsdw

From:
WINDOWS-1K4UUD7.its-dw.ppm.jurusan
Merge:
WINDOWS-1K4UUD7.resits.dbo.tmst_jurusan_baru
Update:
```

Gambar 6. 8 Hasil Pengujian S03 skenario kedua

Pada gambar terlihat bahwa parser dapat mengembalikan array dengan nama objek database yang berhubungan dengan SP

yang sama dengan hasil yang diharapkan dari pengujian. Hal ini menandakan pengujian S03 skenario kedua berhasil dilakukan.

6.2.2.4 Pembahasan Hasil Pengujian S04

Kasus pengujian S04 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan truncate. Stored procedure yang digunakan pada pengujian kali ini dapat dilihat pada Kode A.6 dan Kode A.7 yang terdapat pada lampiran.

Skenario pertama pengujian S04 menggunakan stored procedure yang ditunjukkan dengan Kode A.6. Pada stored procedure tersebut terdapat query truncate. Query truncate dapat digunakan untuk mengidentifikasi hubungan stored procedure dengan objek lain. Setelah dilakukan *parsing* didapatkan hasil seperti pada Gambar 6.9 berikut.

```
Nama SP : WINDOWS-1K4UUD7.resits.dbo.generate_mapping_temp_dosen

From:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
WINDOWS-1K4UUD7.resits.dbo.mapping_publikasi_dosen
WINDOWS-1K4UUD7.resits.dbo.tmst_pegawai
Join:
WINDOWS-1K4UUD7.resits.dbo.tmst_jurusan_baru
WINDOWS-1K4UUD7.resits.dbo.tmst_fakultas_baru
Truncate:
WINDOWS-1K4UUD7.resits.dbo.mapping_temp_dosen
Insert:
```

Gambar 6. 9 Hasil Pengujian S04 skenario pertama

Gambar diatas menunjukkan hasil yang sesuai dengan hasil yang diharapkan pada pengujian. Hal ini menandakan bahwa pengujian S04 skenario pertama berhasil dilakukan.

Skenario kedua pengujian ini menggunakan stored procedure Kode A.7. Dari stored procedure tersebut didapatkan hasil yang dapat dilihat pada Gambar 6.10 berikut ini.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.tran_temp_publikasi_recent

From:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Join:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
WINDOWS-1K4UUD7.resits.dbo.tmst_pegawai
WINDOWS-1K4UUD7.resits.dbo.tmst_jurusan_baru
WINDOWS-1K4UUD7.resits.dbo.tran_temp_publikasi
Truncate:
WINDOWS-1K4UUD7.resits.dbo.tran_temp_publikasi
WINDOWS-1K4UUD7.resits.dbo.tran_temp_publikasi_distinct
WINDOWS-1K4UUD7.resits.dbo.tran_temp_publikasi_mapping
Insert:

Gambar 6. 10 Hasil Pengujian S04 skenario kedua

Dari gambar diatas dapat dilihat bahwa hasil parsing yang dilakukan SP parser sudah sesuai dengan query yang dujicobakan. Hal ini menandakan pengujian S04 skenario kedua berhasil dilakukan.

6.2.2.5 Pembahasan Hasil Pengujian S05

Kasus pengujian S05 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan insert. Stored procedure yang digunakan pada pengujian kali ini dapat dilihat pada Kode A.8 dan Kode A.9 yang terdapat pada lampiran.

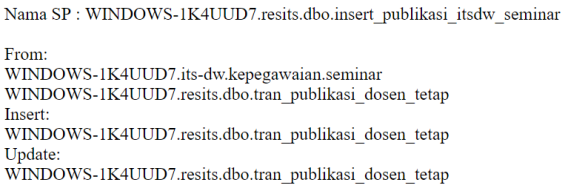
Skenario pertama pengujian S05 menggunakan Kode A.8 sebagai masukan. Pada stored procedure tersebut terdapat query insert. Dari query insert dapat diidentifikasi nama objek database yang berhubungan dengan stored procedure. Setelah dilakukan *parsing* didapatkan hasil yang ditunjukkan Gambar 6.11 berikut.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.insert_publikasi_itsdw_buku

From:
WINDOWS-1K4UUD7.its-dw.kepegawaian.buku
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Insert:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Update:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap

Gambar 6. 11 Hasil Pengujian S05 skenario pertama

Gambar diatas menujukkan bahwa parser dapat mengembalikan nama objek database yang berhubungan melalui query insert sesuai dengan hasil yang diharapkan pada pengjian. Hal ini membuktikan bahwa pengujian S05 skenario pertama berhasil dilakukan.
Pada skenario kedua, stored procedure yang digunakan untuk pengujian adalah Kode A.9. Hasil *parsing* dari stored procedure tersebut dapat dilihat pada Gambar 6.12 berikut.



Gambar 6. 12 Hasil Pengujian S05 skenario kedua

Pada Gambar 6.12 menunjukkan pada SP parser dapat mengembalikan hasil yang sesuai berdasarkan query yang diujikan. Hal ini menunjukkan pengujian S05 skenario kedua berhasil dilakukan.

6.2.2.6 Pembahasan Hasil Pengujian S06

Kasus pengujian S06 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan update. Stored procedure yang digunakan untuk pengujian kali ini dapat dilihat pada Kode A.9 dan Kode A.10

Masukan untuk skenario pertama adalah stored procedure Kode A.9. Pada stored procedure terdapat query update. Setelah dilakukan *parsing* didapatkan hasil yang ditunjukkan gambar berikut.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.insert_publikasi_itsdw_seminar

From:
WINDOWS-1K4UUD7.its-dw.kepegawaian.seminar
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Insert:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Update:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap

Gambar 6. 13 Hasil Pengujian S06 skenario pertama

Gambar diatas menujukkan hasil *parsing* yang sesuai dengan yang diharapkan dari pengujian. Hal ini membuktikan bahwa pengujian S06 skenario pertama berhasil dilakukan.

Pada skenario kedua, digunakan stored procedure yang ditunjukkan oleh Kode A.10. Hasil *parsing* dari stored procedure tersebut ditunjukkan oleh gambar berikut.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.insert_publikasi_itsdw_jurnal

From:
WINDOWS-1K4UUD7.its-dw.kepegawaian.jurnal
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Insert:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap
Update:
WINDOWS-1K4UUD7.resits.dbo.tran_publikasi_dosen_tetap

Gambar 6. 14 Hasil Pengujian S06 skenario kedua

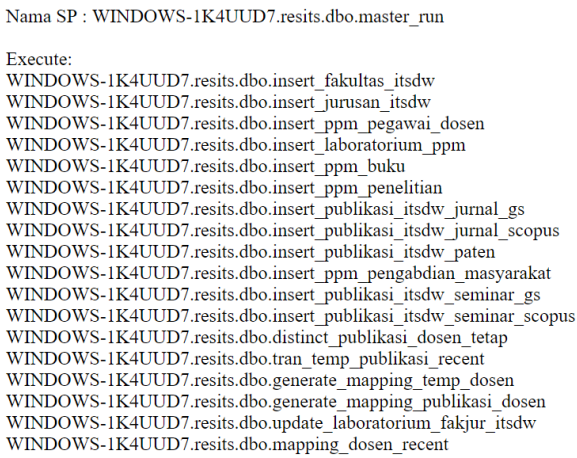
Gambar diatas menujukkan parser dapat mengembalikan nama objek database yang terkait dengan stored procedure. Hal ini menunjukkan bahwa pengujian S06 skenario berhasil dilakukan.

6.2.2.7 Pembahasan Hasil Pengujian S07

Kasus pengujian S07 adalah untuk menguji apakah aplikasi dapat mengembalikan array berisi objek database yang berhubungan execute. Pada pengujian kali ini menggunakan dua stored procedure yang berbeda. Stored procedure yang

digunakan dapat dilihat pada Kode A.11 dan Kode A.12 pada lampiran.

Stored procedure yang digunakan untuk pengujian S07 skenario pertama ditunjukan oleh Kode A.11. Stored procedure tersebut memiliki query execute atau exec. Dari query execute dapat diidentifikasi objek database apa aja yang terkait dengan stored procedure. Objek database yang berhubungan melaui query ini adalah stored procedure atau fungsi lain. Berdasarkan masukan, didapatkan hasil yang ditunjukan oleh gambar berikut ini.



Gambar 6. 15 Hasil Pengujian S07 skenario pertama

Gambar 6.15 menujukkan hasil parsing yang sesuai dengan hasil yang diharapkan. Hal ini menujukkan bahwa pengujian S07 skenario pertama berhasil dilakukan.

Stored procedure yang digunakan untuk pengujian S07 skenario kedua dapa dilihat pada Kode A.12. Pada stored procedure terdapat query execute tetapi tidak digunakan untuk mengeksekusi stored procedure lain secara langsung, sehingga tidak didapatkan objek database. Hasil *parsing* dapat dilihat pada gambar berikut.

Nama SP : WINDOWS-1K4UUD7.resits.dbo.sp_helpdiagrams

From:
WINDOWS-1K4UUD7.resits.dbo.sysdiagrams
Execute:

Gambar 6. 16 Hasil Pengujian S07 skenario kedua

Gambar diatas menujukkan hasil *parsing* yang sesuai dengan hasil yang diharapkan pada pengujian. Hal ini menujukkan bahwa pengujian S07 skenario kedua berhasil dilakukan.

6.2.2.8 Pembahasan Hasil Pengujian V01

Kasus pengujian V01 adalah untuk menguji apakah aplikasi berhasil membuat data yang berformat JSON untuk node dan relasi yang berasal dari hasil query neo4j. Hasil pengujian dapat dilihat pada Kode 6.1 berikut.

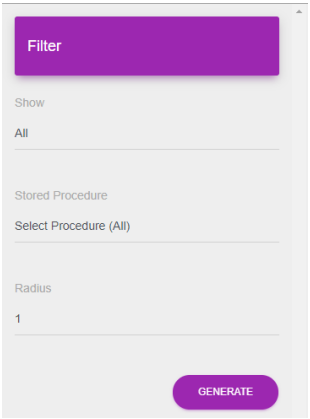

```
{
  "results": [
    {
      "data": [
        {
          "graph": {
            "nodes": [
              {
                "id": 496,
                "labels": [
                  "Table"
                ],
                "properties": {
                  "name": "tran_temp_dosen",
                  "surname": "WINDOWS-1K4UUD7.resits.deleted.tran_temp_dosen",
                  "server": "WINDOWS-1K4UUD7",
                  "database": "resits",
                  "schema": "deleted",
                  "column": "best_term books conferences email gelar_akademik_belakang gelar_akademik_depan id id_node journals kode_fakultas kode_jurusan kode_pegawai kualifikasi_ahli nama_dosen nama_fakultas_en nama_jurusan_en nip paten research skor_spmi thesis ",
                  "PK": "deleted.PK_Dosen"
                }
```

```
22.               },
23.               {
24.                 "id": 497,
25.                 "labels": [
26.                   "Table"
27.                 ],
28.                 "properties": {
29.                   "name": "mapping_fakultas_jurusan",
30.                   "surname": "WINDOWS-
    1K4UUD7.resits.dbo.mapping_fakultas_jurusan",
31.                   "server": "WINDOWS-1K4UUD7",
32.                   "database": "resits",
33.                   "schema": "dbo",
34.                   "column": "fakultas_id_itsdw fakultas_id_resits fakultas_
    id_silacak fakultas_id_simpel jurusan_id_itsdw jurusan_id_resits jurusan_id
    _silacak jurusan_id_simpel "
35.                 }
36.               },
```


Kode 6. 1 Data JSON

6.2.2.9 Pembahasan Hasil Pengujian V02

Kasus pengujian V02 adalah untuk menguji apakah aplikasi dapat menampilkan form filter. Hasil pengujian dapat dilihat pada Gambar 6.17 berikut.

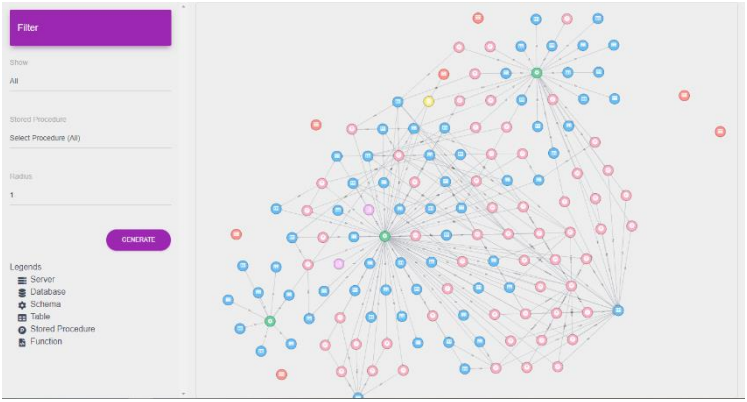


Gambar 6. 17 Hasil Pengujian V02

Aplikasi berhasil menampilkan form untuk *filtering*. Pada form *filtering* terdapat tiga pilihan yaitu pilihan untuk relasi yang ingin ditampilkan, stored procedure yang ingin ditampilkan, serta jumlah hops yang ingin ditampilkan.

6.2.2.10 Pembahasan Hasil Pengujian V03

Kasus pengujian V03 adalah untuk menguji apakah aplikasi dapat menampilkan *graph* sesuai dengan data. Hasil pengujian dapat dilihat pada Gambar 6.18 berikut ini.



Gambar 6. 18 Hasil Pengujian V03

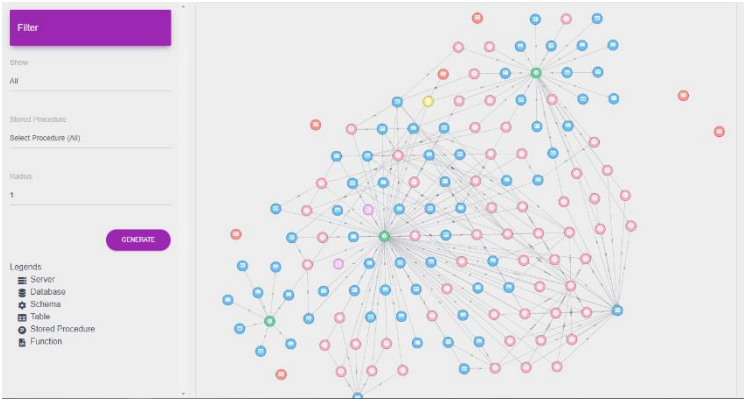
Data JSON yang digunakan sebagai masukan adalah data JSON yang ditunjukkan pada Kode 6.1. Data JSON tersebut mengandung semua data *node* dan relasi yang sudah tersimpan pada neo4j. Pada Gambar 6.18 menunjukkan bahwa aplikasi berhasil menampilkan *graph* yang sesuai dengan data masukan yang digunakan. Hal ini menandakan pengujian V03 berhasil dilakukan.

6.2.2.11 Pembahasan Hasil Pengujian V04

Kasus pengujian V04 adalah untuk menguji apakah aplikasi dapat menampilkan *graph* sesuai filter yang dipilih pengguna. Pada pengujian ini dilakukan sembilan skenario yang berbeda.

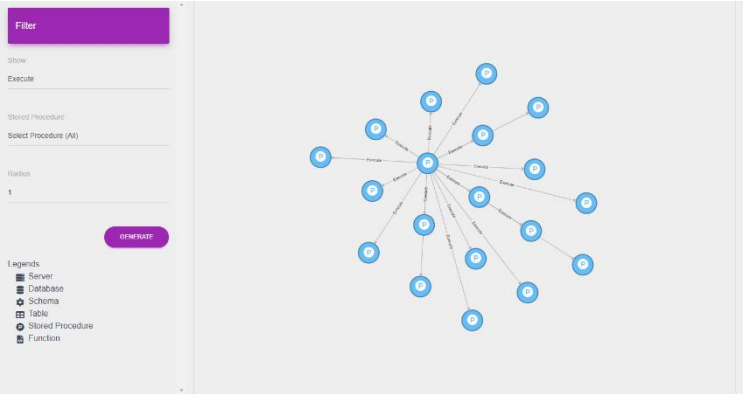
Setiap skenario merepresentasikan *filter* yang berbeda. Hasil pengujian dapat dilihat pada gambar 6.19 – 6.27 berikut ini.

Skenario yang pertama merupakan skenario filter dengan memilih semua relationship dan semua stored procedure. Skenario ini mengharapkan tampilan *graph* untuk semua *node* dan relasinya. Filter yang digunakan dan hasil *graph* untuk skenario ini dapat dilihat pada Gambar 6.19. Hasil yang didapatkan sesuai dengan hasil yang diharapkan. Hal ini menunjukkan skenario pertama untuk pengujian V04 berhasil dilakukan.



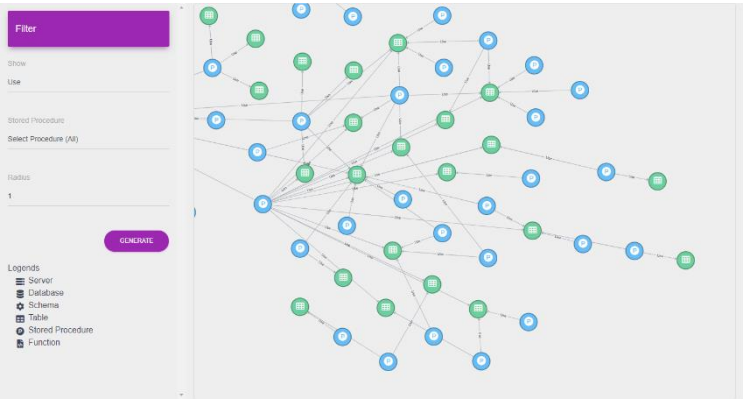
Gambar 6. 19 Hasil Skenario Pertama Pengujian V04

Skenario yang kedua merupakan skenario dengan filter relasi execute untuk semua stored procedure yang ada. Hasil yang diharapkan dari skenario ini adalah aplikasi dapat menampilkan *graph* untuk semua *node* stored procedure dengan hanya relasi execute. Filter yang digunakan dan hasil *graph* dapat dilihat pada Gambar 6.20. Hasil yang didapatkan sesuai dengan hasil yang diharapkan skenario sehingga skenario kedua untuk pengujian V04 berhasil dilakukan.



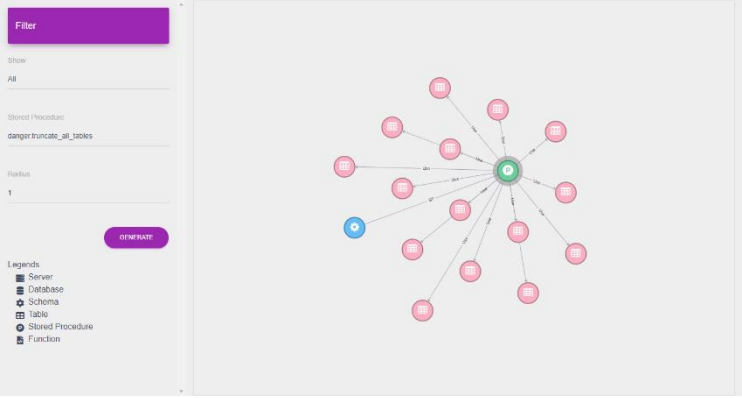
Gambar 6. 20 Hasil Skenario Kedua Pengujian V04

Skenario ketiga adalah menampilkan *node* dan relasi use. Filter show yang dipilih adalah use untuk semua stored procedure. Hasil yang diharapkan dari skenario ini adalah aplikasi dapat menampilkan *graph* untuk semua *node* yang berhubungan dengan relasi use. Hasil *graph* dapat dilihat pada Gambar 6.21. Hasil yang didapatkan sesuai dengan hasil yang diharapkan. Hal ini menunjukkan bahwa skenario ketiga untuk pengujian V04 berhasil dilakukan.



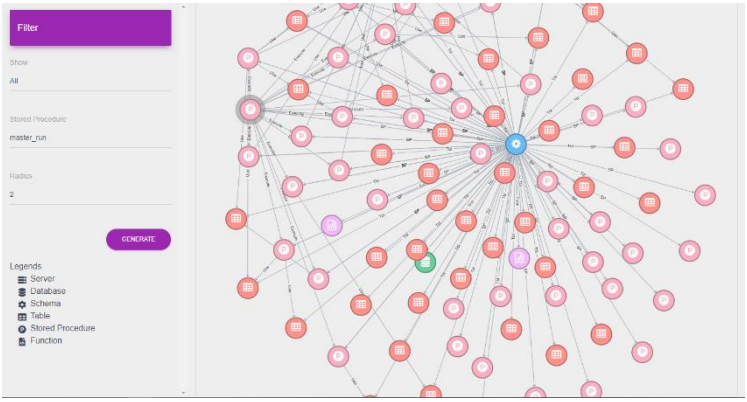
Gambar 6. 21 Hasil Skenario Ketiga Pengujian V04

Skenario keempat merupakan skenario dengan filter show yang dipilih adalah all, filter stored procedure yang dipilih adalah stored procedure yang bernama danger.truncate_all_tables, serta radius yang dipilih adalah 1. Hasil yang diharapkan dari skenario ini adalah tampilan *graph* untuk *node* stored procedure danger.truncate_all_tables serta objek database lain yang terkait dengan radius relasi sejauh satu relasi. Hasil *graph* dapat dilihat pada Gambar 6.22. Gambar 6.22 menujukkan hasil yang sesuai dengan yang diharapkan pada pengujian ini. Hal ini menandakan skenario keempat pengujian V04 berhasil dilakukan.



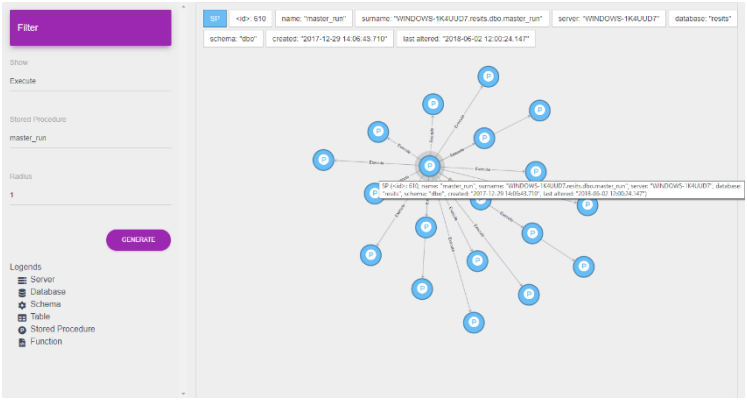
Gambar 6. 22 Hasil Skenario Keempat Pengujian V04

Skenario kelima merupakan skenario dengan filter show yang dipilih adalah all, stored procedure yang dipilih adalah master_run dan radius yang dipilih adalah 2. Dari skenario ini hasil yang diharapkan adalah aplikasi dapat menampilkan *graph* dengan *node* berlabel SP dengan nama master_run serta *node* lainya dalam dua hops (lompatan) relasi. Hasil yang didapatkan dalam pengujian ditunjukan dengan Gambar 6.23 berikut. Gambar menunjukan hasil yang sesuai dengan harapan. Hal ini menandakan bahwa skenario untuk pengujian V04 berhasil dilakukan.



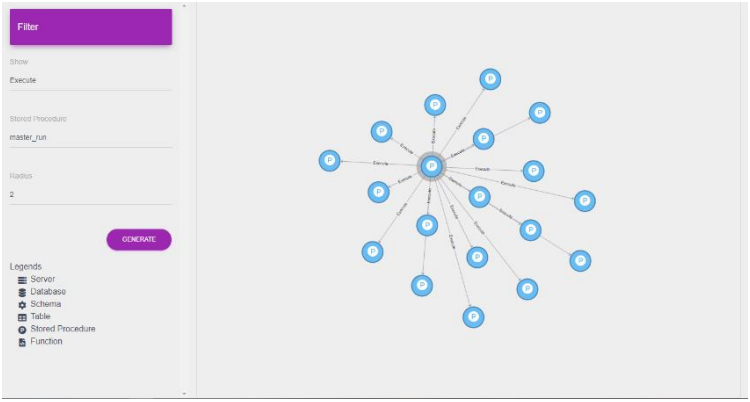
Gambar 6. 23 Hasil Skenario Kelima Pengujian V04

Skenario keenam adalah skenario dengan filter show yang dipilih adalah execute, stored procedure yang dipilih adalah master_run, dan radius / hops adalah 1. Hasil yang diharapkan dari skenario ini adalah tampilan *graph* dengan *node* master_run dan *node* yang berhubungan execute dengan radius satu hops. Hasil skenario keenam dapat dilihat pada Gambar 6.24. Hasil pengujian sesuai dengan hasil yang diharapkan, sehingga skenario ini berhasil dilakukan.



Gambar 6. 24 Hasil Skenario Keenam Pengujian V04

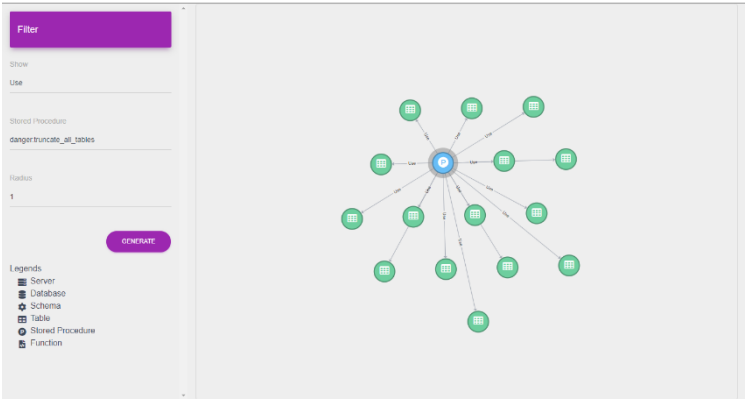
Skenario ketujuh adalah skenario dengan filter sama dengan skenario keenam. Yang berbeda hanyalah hops yang dipilih pada skenario ini adalah 2. Hasil yang diharapkan dari skenario ini adalah tampilan *graph* dengan *node* master_run dan *node* lain yang berelasi execute dalam radius 2 hops (lompatan). Hasil pengujian dapat dilihat pada Gambar 6.25.



Gambar 6. 25 Hasil Skenario Ketujuh Pengujian V04

Pada gambar diatas ditunjukkan *graph* yang menampilkan relasi execute yang sama dengan hasil pada skenario sebelumnya. Hal ini terjadi karena tidak ada relasi execute pada jarak dua hops (lompatan). Hasil ini sesuai dengan hasil yang diinginkan, sehingga skenario ini berhasil dilakukan.

Skenario kedelapan adalah skenario dengan filter show yang dipilih adalah use, stored procedure yang dipilih dange.truncate_all_table, dengan 1 hops. Hasil yang diharapkan adalah tampilan *graph* dengan *node* master_run serta *node* lain yang berelasi use dengan jarak satu hops (lompatan). Hasil pengujian dapat dilihat pada Gambar 6.26. Hasil yang didapatkan sesuai dengan hasil yang diharapkan. Hal ini menandakan bahwa skenario kedelapan untuk pengujian V04 berhasil dilakukan.



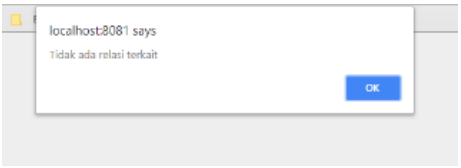
Gambar 6. 26 Hasil Skenario Kedelapan Pengujian V04

Skenario kesembilan hamper sama dengan skenario kedelepan. Yang membedakan hanyalah hops yang dipakai adalah 2. Hasil yang diharapkan adalah tampilan *graph* dengan *node* master_run serta *node* lain yang berelasi use dengan jarak dua hops (lompatan). Hasil pengujian dapat dilihat pada Gambar 6.27. Hasil yang didapatkan sesuai dengan yang diharapkan. Hal ini menandakan bahwa skenario kesembilan pengujian V04 berhasil dilakukan.

Gambar 6. 27 Hasil Skenario Kesembilan Pengujian V04

6.2.2.12 Pembahasan Hasil Pengujian V05

Kasus pengujian V05 adalah untuk menguji apakah aplikasi dapat menampilkan pesan peringatan ketika terjadi error. Hasil yang diharapkan adalah munculnya pesan peringatan ketika tidak ada *graph* yang ditampilkan. Hasil pengujian dapat dilihat pada Gambar 6.28 berikut. Hasil yang didapatkan sesuai dengan hasil yang diharapkan sebelumnya. Hal ini menunjukan bahwan pengujian V05 berhasil dilakukan.



Gambar 6. 28 Hasil Pengujian V05

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dibahas mengenai kesimpulan dari pengerjaan Tugas Akhir dan saran yang dapat diberikan untuk pengembangan yang lebih baik.

7.1. Kesimpulan

Berdasarkan dengan pengerjaan tugas akhir dengan judul "Rancang Bangun Aplikasi Stored Procedure Dependency Tool untuk Database SQL Server berbasis Graph Neo4J" yang telah dilakukan dapat disimpulkan beberapa hal sebagai berikut :

1. Pembuatan aplikasi Stored Procedure Dependency Tool telah berhasil dibuat dalam tugas akhir ini.
2. Objek Database berhasil didapatkan dengan cara melakukan query yang terdapat pada komponen DB Object Getter. Hasil yang didapatkan dari query ini sebagai masukan untuk pembuatan graph data pada neo4j. Jumlah objek database yang didapatkan adalah 14 server, 1 database, 3 schema, 64 tabel, 59 *stored procedure*, dan 2 *function*.
3. Informasi stored procedure didapatkan dengan cara melakukan query yang terdapat pada komponen translator. Hasil dari query ini sebagai masukan untuk proses parsing. Informasi stored procedure yang didapat adalah infomasi server, database, schema, *sql definition*, *created date* dan *last altered date*.
4. Hubungan Stored Procedure dengan objek database lainya dapat diidentifikasi melalui proses parsing yang dilakukan oleh komponen SP Parser. Hasil parsing digunakan sebagai masukan untuk pembuatan relasi antara SP dengan objek database lain pada graph data. Hal ini juga didukung dengan pengujian aplikasi untuk komponen SP Parser.
5. Aplikasi stored procedure dependency tool dapat memvisualisasikan informasi hubungan *database*,

table, *schema*, *function*, dan *stored procedure* dengan memanfaatkan library neo4jd3.js. Hal ini didukung dengan pengujian komponen visualizer yang telah dijelaskan pada bab sebelumnya.

6. Rata-rata waktu ekseskusi untuk proses *extraction* adalah 18.94 detik sedangkan untuk proses *parsing* selama 6.36 detik.
7. Hasil pengujian dari aplikasi menujukkan hasil yang sangat baik dari sisi fungsional maupun dari sisi performa. Pada pengujian performa, dua proses yang diuji menujukkan perfoma yang baik. Pada pengujian fungsional terdapat dua komponen yang diuji dengan 13 kasus pengujian. Setiap kasus pengujian berhasil dilakukan oleh aplikasi.

7.2. Saran

Berdasarkan hasil pengerjaan tugas akhir ini, maka saran yang dapat penulis berikan untuk penelitian selanjutnya antara lain:

1. Parser yang dibuat untuk menganalisis hubungan stored procedure dengan objek database lainya hanya bisa menganalisis hubungan dari query from, join, merge, truncate, insert, update, dan execute. Untuk pengembangan selanjutnya, diharapkan dapat menganalisis dari query yang lain.
2. Parser yang dibuat belum bisa menganalisis hubungan stored procedure dengan objek database lain yang berasal dari server yang berbeda atau linked server. Pada pengembangan selanjutnya disarankan dapat mengenalisis hubungan dengan objek database dari linked server.
3. Aplikasi ini perlu dikembangkan fitur CDC (Change Data Capture) sehingga ketika ada perubahan data, tidak harus mengulang semua proses dari awal hanya mencakup data-data yang berubah saja.

4. Perlu dilakukan pembuatan API (Application Programming Interface) apabila aplikasi ini akan diintegrasikan dengan lingkungan atau aplikasi eksternal.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] M. Kajko-Mattsson, "A Survey of Documentation Practice within Corrective Maintenance," *Empir. Softw. Eng.*, vol. 10, no. 1, pp. 31–55, 2005.

[2] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 1990.

[3] "Stack Overflow Developer Survey 2016 Results." [Online]. Available: https://insights.stackoverflow.com/survey/2016. [Accessed: 21-Dec-2017].

[4] Kementerian Pemuda dan Olahraga, *Roadmap Pengembangan Teknologi Informasi Komunikasi 2015-2020 Kementerian Pemuda dan Olahraga*. 2015.

[5] W. R. King, "Knowledge Management and Organizational Learning," vol. 4, pp. 3–13, 2009.

[6] I. L. Drabikova, K. Matiasko, and A. Lieskovsky, "Database Object Dependency Tool," pp. 1022–1027, 2016.

[7] H. I. Robbani, "Rancang Bangun Aplikasi Visualisasi Database SQL Server dengan Dynamic Management View berbasis Graph Neo4J untuk Memetakan Relasi Implisit pada Database," 2017.

[8] D. Sam, "SQL Server: Search And Find Stored Procedure," 2016. [Online]. Available: http://www.mytecbits.com/microsoft/sql-server/search-find-stored-procedure. [Accessed: 28-Dec-2017].

[9] A. V Aho, M. S. Lam, and J. D. Ullman, "Second Edition Ravi Sethi," *Read. MA*, vol. 71, no. 6, p. 1038, 2007.

[10] J. Celko, *Graph Databases*. 2014.

[11] Gartner, "The Competitive Dynamics of the Consumer Web: Five Graphs Deliver a Sustainable Advantage." [Online]. Available: https://www.gartner.com/doc/2081316/competitive-

dynamics-consumer-web-graphs. [Accessed: 22-Dec-2017].

[12] Microsoft Corporation, "-- (Comment) (Transact-SQL)," 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/comment-transact-sql?view=sql-server-2017. [Accessed: 10-Jul-2018].

[13] Microsoft Corporation, "Slash Star (Block Comment) (Transact-SQL)," 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/slash-star-comment-transact-sql?view=sql-server-2017.

[14] Microsoft Corporation, "Transact-SQL Syntax Conventions (Transact-SQL)," 2018. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transact-sql-syntax-conventions-transact-sql?view=sql-server-2017. [Accessed: 07-Oct-2018].

[15] Microsoft Corporation, "Microsoft SQL Server 2012 Transact-SQL DML Reference," 2012.

LAMPIRAN A

```
-- =============================================
-- Author:      w sakti
-- Create date: 211217
-
- Description: redundansi data publikasi jurnal dari its dw
-- =============================================
ALTER PROCEDURE [dbo].[insert_ppm_penelitian] AS

declare @judul varchar(8000)
declare @tahun int
declare @kode_pegawai int
declare @jumlah_data int
declare @id int
declare @cursor CURSOR
declare @abstrak varchar(8000)
declare @pengarang varchar(5000)

set @cursor = cursor for
select
gsj.judul as judul,
gsj.tahun as tahun,
gsj.dosen_id as kode_pegawai,
gsj.abstraksi as abstrak,
gsj.pengarang as pengarang
FROM [its-dw].[ppm].[penelitian] gsj
WHERE gsj.dosen_id IS NOT NULL AND gsj.judul IS NOT NULL AND gsj.tahun IS NOT NULL

open @cursor
fetch next from @cursor into @judul,@tahun,@kode_pegawai,@abstrak,@pengarang

while @@FETCH_STATUS = 0
begin

    print 'judul :'+@judul
    print 'kode_pegawai :'+cast(@kode_pegawai as varchar)
    print 'tahun : '+cast(@tahun as varchar)
    print 'abstrak : '+@abstrak
    print 'pengarang: '+@pengarang
```

```
38.
39.     select @jumlah_data=count(*) from resits.dbo.tran_publikasi_dosen_tetap
40.     WHERE kode_kegiatan_publikasi='PT' and
41.     REPLACE(lower(cast(LEFT(CAST(@judul as varchar(255)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast(LEFT(CAST(judul as varchar(255)),200) as varchar(255))), ' ', '')
42.     and tahun=@tahun
43.     and kode_pegawai=@kode_pegawai
44.
45.     print 'jumlah_data: '+cast(@jumlah_data as varchar)

46.
47.     if @jumlah_data=0
48.     begin
49.         print 'insert data'
50.         insert into resits.dbo.tran_publikasi_dosen_tetap (judul, tahun, kode_pegawai, kode_kegiatan_publikasi, abstraksi, pengarang, created_at,flag)
51.         values (@judul, @tahun, @kode_pegawai,'PT', @abstrak, @pengarang, CURRENT_TIMESTAMP,'23')
52.     end
53.     if @jumlah_data>0
54.     begin
55.         print 'update data'
56.         select @id=kode_publikasi from resits.dbo.tran_publikasi_dosen_tetap
57.         WHERE kode_kegiatan_publikasi='PT' and
58.         REPLACE(lower(cast(LEFT(CAST(@judul as varchar(255)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast(LEFT(CAST(judul as varchar(255)),200) as varchar(255))), ' ', '')
59.         and tahun=@tahun
60.         and kode_pegawai=@kode_pegawai
61.
62.         update resits.dbo.tran_publikasi_dosen_tetap
63.         set judul=@judul
64.         ,tahun=@tahun
65.         ,kode_pegawai=@kode_pegawai
66.         ,abstraksi=@abstrak
67.         ,pengarang=@pengarang
68.         ,updated_at=CURRENT_TIMESTAMP
69.         where kode_publikasi=@id
```

```
    end

    fetch next from @cursor into @judul,@tahun,@kode_pegawai,@abstrak,@pengarang
end

close @cursor
deallocate @cursor
```

Kode A. 1 Stored Procedure untuk Pengujian S01 (1)

```
ALTER PROCEDURE [dbo].[generate_jumlah_scopus]
AS
BEGIN

    SET NOCOUNT ON;

    TRUNCATE TABLE jumlah_scopus;
    INSERT INTO jumlah_scopus(kode,nip,nama,jurnal_scopus,seminar_scopus,jumlah)
    SELECT *,(Jurnal_scopus+(0.25*Seminar_scopus)) as Jumlah from(SELECT
        kode,
        nip_baru,
        nama,
        (select count(*) from [its-dw].[ppm].[scopus_jurnal] where dosen_id=kode) as Jurnal_scopus,
        (select count(*) from [its-dw].[ppm].[scopus_seminar] where dosen_id=kode) as Seminar_scopus
    FROM [resits].[dbo].tmst_pegawai) as Hasil

END
```

Kode A. 2 Stored Procedure untuk Pengujian S01 (2)

```
ALTER PROCEDURE [dbo].[mapping_fakultas_jurusan_recent]

AS


            MERGE dbo.mapping_fakultas_jurusan AS t
            USING
            (
            select  pf.id as fakultas_id_itsdw,
                    pj.id as jurusan_id_itsdw,
```

```
10.                     rf.kode as fakultas_id_resits,
11.                     rj.kode as jurusan_id_resits,
12.                     --
    sf.fak_id as fakultas_id_silacak,
13.                     --sj.jur_id as jurusan_id_silacak,
14.                     lf.FAKULTAS_ID as fakultas_id_simpel
    ,
15.                     lj.JURUSAN_ID as jurusan_id_simpel
16.
17.             from    [its-dw].akademik.jurusan pj
18.
19.             join [its-
    dw].akademik.fakultas pf ON pj.fakultas_id = pf.id
20.             join [resits].dbo.tmst_fakultas_baru rf ON p
    f.kode_fakultas_lama = rf.kode
21.             join [resits].dbo.tmst_jurusan_baru rj ON rj
    .nama_indonesia = pj.nama COLLATE DATABASE_DEFAULT
22.             --
    join [SILACAK].[pelacakan].dbo.fakultas sf ON sf.fak_sin
    gkatan = pf.singkatan COLLATE DATABASE_DEFAULT
23.             --
    join [SILACAK].[pelacakan].dbo.jurusan sj ON sj.jur_nama
    _indonesia = pj.nama COLLATE DATABASE_DEFAULT
24.             join openquery([simpel],'select * from simpe
    l.dbo.fakultas') lf ON lf.FAKULTAS_NAMA_SINGKATAN = pf.s
    ingkatan COLLATE DATABASE_DEFAULT
25.             join openquery([simpel],'select * from simpe
    l.dbo.jurusan') lj on lj.JURUSAN_NAMA = pj.nama COLLATE
    DATABASE_DEFAULT
26.
27.
28.             ) AS s
29.
30.             ON t.fakultas_id_itsdw = s.fakultas_id_itsdw
     AND t.jurusan_id_itsdw = s.jurusan_id_itsdw
31.             WHEN NOT MATCHED BY TARGET
32.                     -
    - THEN INSERT(fakultas_id_itsdw, fakultas_id_resits, fak
    ultas_id_silacak, fakultas_id_simpel,
33.                     -
    - jurusan_id_itsdw, jurusan_id_resits, jurusan_id_silaca
    k, jurusan_id_simpel)
34.                     THEN INSERT(fakultas_id_itsdw, fakulta
    s_id_resits, fakultas_id_simpel, fakultas_id_silacak,
```

```
                    jurusan_id_itsdw, jurusan_id_resits, j
urusan_id_simpel, jurusan_id_silacak)
                    -
- VALUES(s.fakultas_id_itsdw, s.fakultas_id_resits, s.fa
kultas_id_silacak, s.fakultas_id_simpel,
                    -
- s.jurusan_id_itsdw, s.jurusan_id_resits, s.jurusan_id_
silacak, s.jurusan_id_simpel)
                    VALUES(s.fakultas_id_itsdw, s.fakultas
_id_resits, s.fakultas_id_simpel, NULL,
                    s.jurusan_id_itsdw, s.jurusan_id_resit
s, s.jurusan_id_simpel, NULL)
            WHEN MATCHED
                    THEN UPDATE SET   t.fakultas_id_itsdw
= s.fakultas_id_itsdw,
                                      t.fakultas_id_resits
 = s.fakultas_id_resits,
                                      --
t.fakultas_id_silacak = s.fakultas_id_silacak,
                                      t.fakultas_id_simpel
 = s.fakultas_id_simpel,
                                      t.jurusan_id_itsdw =
 s.jurusan_id_itsdw,
                                      t.jurusan_id_resits
= s.jurusan_id_resits,
                                      --
t.jurusan_id_silacak = s.jurusan_id_silacak,
                                      t.jurusan_id_simpel
= s.jurusan_id_simpel;
```

Kode A. 3 Stored Procedure untuk Pengujian S02 (1)

```
ALTER PROCEDURE [dbo].[mapping_dosen_recent]
AS


            MERGE dbo.mapping_dosen AS t
            USING
            (
            select
                pdi.id as dosen_id_itsdw,
                pdi.nip_kepegawaian as nip_dosen,
                pr.kode as kode_pegawai_resits,
                psp.peneliti_id as peneliti_id_simpel
                -- ds.peg_id as pegawai_id_silacak
```

```
14.             from    [its-
    dw].kepegawaian.profil_dosen pdi
15.             left join [resits].dbo.tmst_pegawai pr ON pr
    .kode = pdi.id
16.             left join openquery([simpel],'select * from
    simpel.dbo.peneliti') psp ON psp.peneliti_nip = pdi.nip_
    kepegawaian
17.
18.             ) AS s
19.
20.             ON t.dosen_id_itsdw = s.dosen_id_itsdw
21.             WHEN NOT MATCHED BY TARGET
22.                     --
    THEN INSERT(dosen_id_itsdw, pegawai_id_silacak, peneliti
    _id_simpel, kode_pegawai_resits, nip_dosen)
23.                     THEN INSERT(dosen_id_itsdw, peneliti_i
    d_simpel, kode_pegawai_resits, nip_dosen)
24.                     --
    VALUES(s.dosen_id_itsdw, s.pegawai_id_silacak, s.penelit
    i_id_simpel, s.kode_pegawai_resits, s.nip_dosen)
25.                     VALUES(s.dosen_id_itsdw, s.peneliti_id
    _simpel, s.kode_pegawai_resits, s.nip_dosen)
26.             WHEN MATCHED
27.                     THEN UPDATE SET
28.                                         t.dosen_id_itsdw = s
    .dosen_id_itsdw,
29.                                         --
    t.pegawai_id_silacak = s.pegawai_id_silacak,
30.                                         t.peneliti_id_simpel
     = s.peneliti_id_simpel,
31.                                         t.nip_dosen = s.nip_
    dosen,
32.                                         t.kode_pegawai_resit
    s = s.kode_pegawai_resits;
```

Kode A. 4 Stored Procedure untuk Pengujian S02 (2) dan S03 (1)

```
1.  ALTER PROC [dbo].[insert_jurusan_itsdw] AS
2.  MERGE dbo.tmst_jurusan_baru AS t
3.          USING
4.          (
5.          SELECT
6.              jur_id,
7.              fakultas_id,
8.              log_audit,
```

```
            nama_indonesia,
            nama_inggris,
            dosen_kepala
        FROM [its-dw].[ppm].[jurusan]
        where jur_id BETWEEN '1' and '38'
        ) AS s
            ON t.kode = s.jur_id
                WHEN NOT MATCHED BY TARGET
                THEN INSERT(kode, kode_fakultas, kode_log_audit, nama_indonesia, nama_inggris, kode_dosen_kepala, created_at, flag) VALUES
            (
                s.jur_id, s.fakultas_id, s.log_audit, s.nama_indonesia, s.nama_inggris, s.dosen_kepala, CURRENT_TIMESTAMP, '3'
            )
                WHEN MATCHED
                THEN
                UPDATE SET
                                    t.kode = s.jur_id,
                                    t.kode_fakultas = s.fakultas_id,
                                    t.kode_log_audit = s.log_audit,
                                    t.nama_indonesia = s.nama_indonesia,
                                    t.nama_inggris = s.nama_inggris,
                                    t.kode_dosen_kepala =s.dosen_kepala,
                                    t.updated_at = CURRENT_TIMESTAMP,
                                    t.flag = '3';
```

Kode A. 5 Stored Procedure untuk Pengujian S03 (2)

```
-- =============================================
-- Author:      w sakti
-- Create date: 291217
-- Description: re generate mapping temporary dosen
-- =============================================
ALTER PROCEDURE [dbo].[generate_mapping_temp_dosen]
    -
- Add the parameters for the stored procedure here
AS
BEGIN
```

```
10.     -
    - SET NOCOUNT ON added to prevent extra result sets from
11.     -- interfering with SELECT statements.
12.     SET NOCOUNT ON;
13.
14.     -- Insert statements for procedure here
15.     TRUNCATE TABLE mapping_temp_dosen;
16. INSERT INTO mapping_temp_dosen
17. SELECT tp.kode_fakultas,
18. tp.kode_jurusan,
19. tp.nama,
20. tp.gelar_akademik_depan,
21. tp.gelar_akademik_belakang,
22. NULL AS skor,
23. NULL kualifikasi,
24. tj.nama_inggris,
25. (SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
    n_tetap
26.                 WHERE kode_publikasi IN
27.                 (SELECT kode
28.                 FROM [resits].[dbo].mapping_publikasi_dose
    n
29.                 WHERE kode_pegawai = tp.kode) AND kode_keg
    iatan_publikasi='JL') AS journals,
30. (SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
    n_tetap
31.                 WHERE kode_publikasi IN
32.                 (SELECT kode
33.                 FROM [resits].[dbo].mapping_publikasi_dose
    n
34.                 WHERE kode_pegawai = tp.kode) AND kode_keg
    iatan_publikasi='SM') AS conferences,
35. (SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
    n_tetap
36.                 WHERE kode_publikasi IN
37.                 (SELECT kode
38.                 FROM [resits].[dbo].mapping_publikasi_dose
    n
39.                 WHERE kode_pegawai = tp.kode) AND kode_keg
    iatan_publikasi='BK') AS books,
40. tp.kode,
41. (SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
    n_tetap
```

```
                WHERE kode_publikasi IN
                (SELECT kode
                FROM [resits].[dbo].mapping_publikasi_dose
n
                WHERE kode_pegawai = tp.kode) AND (kode_ke
giatan_publikasi='TA' OR kode_kegiatan_publikasi='DS'))
AS thesis,
(SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
n_tetap
                WHERE kode_publikasi IN
                (SELECT kode
                FROM [resits].[dbo].mapping_publikasi_dose
n
                WHERE kode_pegawai = tp.kode) AND kode_keg
iatan_publikasi='PN') AS paten,
(SELECT COUNT(*) FROM [resits].[dbo].tran_publikasi_dose
n_tetap
                WHERE kode_publikasi IN
                (SELECT kode
                FROM [resits].[dbo].mapping_publikasi_dose
n
                WHERE kode_pegawai = tp.kode) AND kode_keg
iatan_publikasi='PT') AS research,
NULL AS id_node,
tp.nip_baru,
tf.nama_inggris,
tp.email,
NULL AS best_term
FROM [resits].[dbo].tmst_pegawai tp
LEFT JOIN [resits].[dbo].tmst_jurusan_baru tj ON tp.kode
_jurusan=tj.kode
LEFT JOIN [resits].[dbo].tmst_fakultas_baru tf ON tf.kod
e=tp.kode_fakultas
WHERE tp.kode_jurusan IS NOT NULL;
END
```

Kode A. 6 Stored Procedure untuk Pengujian S04 (1)

```
-- =============================================
-- Author:      w sakti
-- Create date: 291217
-- Description: re generate temporary latest publikasi
-- =============================================
ALTER PROCEDURE [dbo].[tran_temp_publikasi_recent]
    -
- Add the parameters for the stored procedure here
```

```
8.
9.  AS
10. BEGIN
11.     -
    - SET NOCOUNT ON added to prevent extra result sets from

12.     -- interfering with SELECT statements.
13.     SET NOCOUNT ON;
14.
15.     -- Insert statements for procedure here
16. TRUNCATE TABLE tran_temp_publikasi;
17. INSERT INTO tran_temp_publikasi
18. SELECT tp.gelar_akademik_depan, tp.nama, tp.kode, tp.gel
    ar_akademik_belakang, tj.nama_indonesia, tp.kode_jurusan
    ,
19. pub.tahun, pub.pengarang, pub.judul, pub.kode_publikasi,
     tp.kode_fakultas, tp.nip_baru, pub.kode_kegiatan_publik
    asi
20. FROM (
21.     SELECT f.*
22.     FROM (
23.       SELECT kode_pegawai, max(kode_publikasi) AS kode_p
    ublikasi
24.       FROM [resits].[dbo].tran_publikasi_dosen_tetap GRO
    UP BY kode_pegawai
25.     ) AS x
26.     INNER JOIN [resits].[dbo].tran_publikasi_dosen_tetap
     AS f ON f.kode_pegawai = x.kode_pegawai AND f.kode_publ
    ikasi = x.kode_publikasi
27.  ) pub
28. LEFT JOIN [resits].[dbo].tmst_pegawai tp ON pub.kode_peg
    awai=tp.kode
29. LEFT JOIN [resits].[dbo].tmst_jurusan_baru tj ON tj.kode
    =tp.kode_jurusan
30. WHERE tp.nama IS NOT NULL AND tp.kode_jurusan IS NOT NUL
    L
31. ORDER BY tahun DESC;
32.
33.
34. truncate table tran_temp_publikasi_distinct;
35. truncate table tran_temp_publikasi_mapping;
36.
37. insert into tran_temp_publikasi_distinct(tahun,judul,tip
    e)
```

```
38. select distinct periode_penelitian,judul_penelitian,tipe

39. from tran_temp_publikasi
40.
41.
42. insert into tran_temp_publikasi_mapping
43. select a.id,b.kode_publikasi
44. from tran_temp_publikasi_distinct a
45. join tran_temp_publikasi b
46. on a.tahun = b.periode_penelitian
47. and a.judul = b.judul_penelitian
48. and a.tipe = b.tipe
49.
50. END
```

Kode A. 7 Stored Procedure untuk Pengujian S04 (2)

```
1.  ALTER PROCEDURE [dbo].[insert_publikasi_itsdw_buku] AS
2.
3.  declare @judul varchar(2000)
4.  declare @tahun int
5.  declare @kode_pegawai int
6.  declare @jumlah_data int
7.  declare @id int
8.  declare @cursor CURSOR
9.  declare @abstrak varchar(5000)
10. declare @keyword varchar(5000)
11. declare @penerbit varchar(500)
12.
13. set @cursor = cursor for
14. select
15. pjid.judul as judul,
16. pjid.tahun_terbit as tahun,
17. pjid.dosen_id as kode_pegawai,
18. pjid.abstrak as abstrak,
19. pjid.kata_kunci as keyword,
20. pjid.penerbit as penerbit
21. FROM [its-dw].[kepegawaian].[buku] pjid
22. WHERE pjid.dosen_id IS NOT NULL AND pjid.judul IS NOT NU
    LL
23.
24. open @cursor
25. fetch next from @cursor into @judul,@tahun,@kode_pegawai
    ,@abstrak,@keyword,@penerbit
26.
27. while @@FETCH_STATUS = 0
```

```
begin

    print 'judul :'+@judul
    print 'kode_pegawai :'+cast(@kode_pegawai as varchar
)
    print 'tahun : '+cast(@tahun as varchar)
    print 'abstrak : '+@abstrak
    print 'keyword: '+@keyword
    print 'penerbit: '+@penerbit

    select @jumlah_data=count(*) from resits.dbo.tran_pu
blikasi_dosen_tetap
    WHERE kode_kegiatan_publikasi='BK' and
    REPLACE(lower(cast(LEFT(CAST(@judul as varchar(255))
,200) as varchar(255))), ' ', '')=REPLACE(lower(cast(LEF
T(CAST(judul as varchar(255)),200) as varchar(255))), '
', '')
    and tahun=@tahun
    and kode_pegawai=@kode_pegawai

    print 'jumlah_data: '+cast(@jumlah_data as varchar)


    if @jumlah_data=0
    begin
        print 'insert data'
        insert into resits.dbo.tran_publikasi_dosen_teta
p (judul, tahun, kode_pegawai, kode_kegiatan_publikasi,
abstraksi, kata_kunci, published_in, created_at, flag)
        values (@judul, @tahun, @kode_pegawai,'BK', @abs
trak, @keyword,@penerbit, CURRENT_TIMESTAMP, '25')
    end
    if @jumlah_data>0
    begin
        print 'update data'
        select @id=kode_publikasi from resits.dbo.tran_p
ublikasi_dosen_tetap
        WHERE kode_kegiatan_publikasi='BK' and
        REPLACE(lower(cast(LEFT(CAST(@judul as varchar(2
55)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast
(LEFT(CAST(judul as varchar(255)),200) as varchar(255)))
, ' ', '')
        and tahun=@tahun
        and kode_pegawai=@kode_pegawai
```

```
59.
60.         update resits.dbo.tran_publikasi_dosen_tetap
61.         set judul=@judul
62.         ,tahun=@tahun
63.         ,kode_pegawai=@kode_pegawai
64.         ,abstraksi=@abstrak
65.         ,kata_kunci=@keyword
66.         ,published_in=@penerbit
67.         ,updated_at=CURRENT_TIMESTAMP
68.         where kode_publikasi=@id
69.     end
70.
71.     fetch next from @cursor into @judul,@tahun,@kode_peg
    awai,@abstrak,@keyword,@penerbit
72. end
73.
74. close @cursor
75. deallocate @cursor
```

Kode A. 8 Stored Procedure untuk Pengujian S05 (1)

```
1.  -- =============================================
2.  -- Author:      w sakti
3.  -- Create date: 231217
4.  -- Description: redundansi data seminar dari itsdw
5.  -- =============================================
6.  ALTER PROCEDURE [dbo].[insert_publikasi_itsdw_seminar] A
    S
7.  RETURN
8.
9.  --Tidak Di Pakai
10. declare @judul varchar(2000)
11. declare @tahun int
12. declare @kode_pegawai int
13. declare @jumlah_data int
14. declare @id int
15. declare @cursor CURSOR
16. declare @abstrak varchar(5000)
17. declare @keyword varchar(5000)
18.
19. set @cursor = cursor for
20. select
21. smid.judul as judul,
22. smid.tahun as tahun,
23. smid.dosen_id as kode_pegawai,
24. smid.abstrak as abstrak,
```

```
smid.keyword as keyword
FROM [its-dw].[kepegawaian].[seminar] smid
WHERE smid.dosen_id IS NOT NULL AND smid.judul IS NOT NULL AND smid.tahun IS NOT NULL

open @cursor
fetch next from @cursor into @judul,@tahun,@kode_pegawai,@abstrak,@keyword

while @@FETCH_STATUS = 0
begin

    print 'judul :'+@judul
    print 'kode_pegawai :'+cast(@kode_pegawai as varchar)
    print 'tahun : '+cast(@tahun as varchar)
    print 'abstrak : '+@abstrak
    print 'keyword: '+@keyword

    select @jumlah_data=count(*) from resits.dbo.tran_publikasi_dosen_tetap
    WHERE kode_kegiatan_publikasi='SM' and
    REPLACE(lower(cast(LEFT(CAST(@judul as varchar(255)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast(LEFT(CAST(judul as varchar(255)),200) as varchar(255))), ' ', '')
    and tahun=@tahun
    and kode_pegawai=@kode_pegawai

    print 'jumlah_data: '+cast(@jumlah_data as varchar)


    if @jumlah_data=0
    begin
        print 'insert data'
        insert into resits.dbo.tran_publikasi_dosen_tetap (judul, tahun, kode_pegawai, kode_kegiatan_publikasi, kata_kunci, abstraksi, created_at, flag)
        values (@judul, @tahun, @kode_pegawai,'SM', @keyword, @abstrak, CURRENT_TIMESTAMP, '24')
    end
    if @jumlah_data>0
    begin
        print 'update data'
```

```
58.          select @id=kode_publikasi from resits.dbo.tran_p
     ublikasi_dosen_tetap
59.          WHERE kode_kegiatan_publikasi='SM' and
60.          REPLACE(lower(cast(LEFT(CAST(@judul as varchar(2
     55)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast
     (LEFT(CAST(judul as varchar(255)),200) as varchar(255)))
     , ' ', '')
61.          and tahun=@tahun
62.          and kode_pegawai=@kode_pegawai
63.
64.          update resits.dbo.tran_publikasi_dosen_tetap
65.          set judul=@judul
66.          ,tahun=@tahun
67.          ,kode_pegawai=@kode_pegawai
68.          ,kata_kunci=@keyword
69.          ,abstraksi=@abstrak
70.          ,updated_at=CURRENT_TIMESTAMP
71.          where kode_publikasi=@id
72.      end
73.
74.      fetch next from @cursor into @judul,@tahun,@kode_peg
     awai,@abstrak,@keyword
75.  end
76.
77.  close @cursor
78.  deallocate @cursor
```

Kode A. 9 Stored Procedure untuk Pengujian S05 (2) dan S06 (1)

```
1.   -- =============================================
2.   -- Author:      w sakti
3.   -- Create date: 211217
4.   -
     - Description: redundansi data publikasi jurnal dari its
     dw
5.   -- =============================================
6.   ALTER PROCEDURE [dbo].[insert_publikasi_itsdw_jurnal] AS

7.   RETURN
8.
9.   --Tidak DI pakai
10.  declare @judul varchar(2000)
11.  declare @tahun int
12.  declare @kode_pegawai int
13.  declare @jumlah_data int
14.  declare @id int
```

```
15. declare @cursor CURSOR
16. declare @abstrak varchar(5000)
17. declare @keyword varchar(5000)
18. declare @penerbit varchar(500)
19. declare @no_doi varchar(2500)
20. declare @nama_jurnal varchar(500)
21. declare @url varchar(5000)
22.
23. set @cursor = cursor for
24. select
25. pjid.judul as judul,
26. pjid.tahun as tahun,
27. pjid.dosen_id as kode_pegawai,
28. pjid.abstrak as abstrak,
29. pjid.keyword as keyword,
30. pjid.penerbit as penerbit,
31. pjid.no_doi as no_doi,
32. pjid.nama_jurnal as nama_jurnal,
33. pjid.namafile as url
34. FROM [its-dw].[kepegawaian].[jurnal] pjid
35. WHERE pjid.dosen_id IS NOT NULL AND pjid.judul IS NOT NU
    LL AND pjid.tahun IS NOT NULL
36.
37. open @cursor
38. fetch next from @cursor into @judul,@tahun,@kode_pegawai
    ,@abstrak,@keyword,@penerbit,@no_doi,@nama_jurnal,@url
39.
40. while @@FETCH_STATUS = 0
41. begin
42.
43.     print 'judul :'+@judul
44.     print 'kode_pegawai :'+cast(@kode_pegawai as varchar
    )
45.     print 'tahun : '+cast(@tahun as varchar)
46.     print 'abstrak : '+@abstrak
47.     print 'keyword: '+@keyword
48.     print 'penerbit: '+@penerbit
49.     print 'no_doi: '+@no_doi
50.     print 'nama_jurnal: '+@nama_jurnal
51.     print 'url: '+@url
52.
53.     select @jumlah_data=count(*) from resits.dbo.tran_pu
    blikasi_dosen_tetap
54.     WHERE kode_kegiatan_publikasi='JL' and
```

```
55.     REPLACE(lower(cast(LEFT(CAST(@judul as varchar(255))
    ,200) as varchar(255))), ' ', '')=REPLACE(lower(cast(LEF
    T(CAST(judul as varchar(255)),200) as varchar(255))), '
    ', '')
56.     and tahun=@tahun
57.     and kode_pegawai=@kode_pegawai
58.
59.     print 'jumlah_data: '+cast(@jumlah_data as varchar)

60.
61.     if @jumlah_data=0
62.     begin
63.         print 'insert data'
64.         insert into resits.dbo.tran_publikasi_dosen_teta
    p (judul, tahun, kode_pegawai, kode_kegiatan_publikasi,
    abstraksi, kata_kunci, published_in, no_doi, type_jurnal
    , created_at, url,flag)
65.         values (@judul, @tahun, @kode_pegawai,'JL', @abs
    trak, @keyword,@penerbit, @no_doi, @nama_jurnal, CURRENT
    _TIMESTAMP, @url, '21')
66.     end
67.     if @jumlah_data>0
68.     begin
69.         print 'update data'
70.         select @id=kode_publikasi from resits.dbo.tran_p
    ublikasi_dosen_tetap
71.         WHERE kode_kegiatan_publikasi='JL' and
72.         REPLACE(lower(cast(LEFT(CAST(@judul as varchar(2
    55)),200) as varchar(255))), ' ', '')=REPLACE(lower(cast
    (LEFT(CAST(judul as varchar(255)),200) as varchar(255)))
    , ' ', '')
73.         and tahun=@tahun
74.         and kode_pegawai=@kode_pegawai
75.
76.         update resits.dbo.tran_publikasi_dosen_tetap
77.         set judul=@judul
78.         ,tahun=@tahun
79.         ,kode_pegawai=@kode_pegawai
80.         ,abstraksi=@abstrak
81.         ,kata_kunci=@keyword
82.         ,published_in=@penerbit
83.         ,no_doi=@no_doi
84.         ,type_jurnal=@nama_jurnal
85.         ,updated_at=CURRENT_TIMESTAMP
86.         ,url=@url
```

```
87.         where kode_publikasi=@id
88.     end
89.
90.     fetch next from @cursor into @judul,@tahun,@kode_pegawai,@abstrak,@keyword,@penerbit,@no_doi,@nama_jurnal,@url
91. end
92.
93. close @cursor
94. deallocate @cursor
```

Kode A. 10 Stored Procedure untuk Pengujian S06 (2)

```
1.  -- =============================================
2.  -- Author:      `w sakti
3.  -- Create date: 291217
4.  -- Description: jalankan semua sp update
5.  -- =============================================
6.  ALTER PROCEDURE [dbo].[master_run]
7.      -
    - Add the parameters for the stored procedure here
8.  AS
9.  BEGIN
10.     -
    - SET NOCOUNT ON added to prevent extra result sets from

11.     -- interfering with SELECT statements.
12.     SET NOCOUNT ON;
13.
14.     -- Insert statements for procedure here
15.     EXEC insert_fakultas_itsdw;
16.     EXEC insert_jurusan_itsdw;
17.     --EXEC insert_pegawai_dosen;
18.     EXEC insert_ppm_pegawai_dosen;
19.     EXEC insert_laboratorium_ppm;
20.
21.     --EXEC insert_publikasi_itsdw_buku;
22.     --EXEC insert_publikasi_itsdw_jurnal;
23.     EXEC insert_ppm_buku;
24.     EXEC insert_ppm_penelitian;
25.     EXEC insert_publikasi_itsdw_jurnal_gs;
26.     EXEC insert_publikasi_itsdw_jurnal_scopus;
27.     EXEC insert_publikasi_itsdw_paten;
28.     EXEC insert_ppm_pengabdian_masyarakat;
29.     --EXEC insert_publikasi_itsdw_penelitian;
```

```
30.     --EXEC insert_publikasi_itsdw_seminar;
31.     EXEC insert_publikasi_itsdw_seminar_gs;
32.     EXEC insert_publikasi_itsdw_seminar_scopus;
33.
34.     EXEC distinct_publikasi_dosen_tetap
35.
36.     EXEC generate_jumlah_scopus;
37.     EXEC tran_temp_publikasi_recent;
38.     EXEC generate_mapping_temp_dosen;
39.     EXEC generate_mapping_publikasi_dosen;
40.     EXEC update_laboratorium_fakjur_itsdw;
41.     EXEC mapping_dosen_recent;
42.     EXEC generate_mapping_publikasi_dosen;
43.     EXEC generate_mapping_temp_dosen;
44.
45.     --EXEC mapping_fakultas_jurusan_recent;
46.     --EXEC mapping_laboratorium_recent;
47.     --EXEC mapping_dosen_laboratorium_recent;
48. END
```

Kode A. 11 Stored Procedure untuk Pengujian S07 (1)

```
1.  ALTER PROCEDURE [dbo].[sp_helpdiagrams]
2.      (
3.          @diagramname sysname = NULL,
4.          @owner_id int = NULL
5.      )
6.      WITH EXECUTE AS N'dbo'
7.      AS
8.      BEGIN
9.          DECLARE @user sysname
10.         DECLARE @dboLogin bit
11.         EXECUTE AS CALLER;
12.             SET @user = USER_NAME();
13.             SET @dboLogin = CONVERT(bit,IS_MEMBER('db_ow
    ner'));
14.         REVERT;
15.         SELECT
16.             [Database] = DB_NAME(),
17.             [Name] = name,
18.             [ID] = diagram_id,
19.             [Owner] = USER_NAME(principal_id),
20.             [OwnerID] = principal_id
21.         FROM
22.             sysdiagrams
23.         WHERE
```

```
24.             (@dboLogin = 1 OR USER_NAME(principal_id) =
    @user) AND
25.             (@diagramname IS NULL OR name = @diagramname
    ) AND
26.             (@owner_id IS NULL OR principal_id = @owner_
    id)
27.         ORDER BY
28.             4, 5, 1
29.     END
```

Kode A. 12 Stored Procedure untuk Pengujian S07 (2)

BIODATA PENULIS

Penulis memiliki nama lengkap Aprilia Rizki Rahmawati, lahir di Lumajang, 20 April 1997. Merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh Pendidikan formal yaitu; SDN Pasirian 01, SMPN 1 Lumajang, dan SMAN 2 Lumajang.

Pada tahun 2014, penulis melanjutkan Pendidikan tinggi di Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember (ITS) dan terdaftar sebagai mahasiswa dengan NRP 05211440000071. Selama menjadi mahasiswa, penulis mengikuti berbagai kegiatan kemahasiswaan seperti beberapa kepanitian di tingkat jurusan hingga ITS. Selain itu juga penulis aktif sebagai staff Departemen Kewirausahaan HMSI 2015/2016 serta menjadi Sekretaris Departemen Kewirausahaan HMSI 2016/2017. Disamping aktif dalam kegiatan kemahasiswaan, penulis juga aktif menjadi asisten dosen matakuliah PSDP (Perencanaan Sumberdaya Perusahaan) dan DBD (Desain Basis Data).

Pada tahun keempat, penulis memiliki ketertarikan pada bidang basis data sehingga mengambil bidang minat Akuisisi Data dan Diseminasi Informasi (ADDI). Penulis dapat dihubungi melalui *email* di aprilia.rizki.r@gmail.com