TUGAS AKHIR - KI1502

Meningkatkan Efisiensi *Process Discovery* menggunakan *Graph Database* melalui Integrasi Laravel dengan Neo4J

MUHAMMADTAUFIQULSA`DI
NRP 05111640000053

Dosen Pembimbing I
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D

Dosen Pembimbing II
Kelly Rossa Sungkono, S.Kom., M.Kom

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

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – KI1502

Meningkatkan Efisiensi *Process Discovery* menggunakan *Graph Database* melalui Integrasi Laravel dengan Neo4J

MUHAMMAD TAUFIQULSA`DI
NRP 05111640000053

Dosen Pembimbing I
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.

Dosen Pembimbing II
Kelly Rossa Sungkono, S.Kom., M.Kom

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

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI1502

Improving Efficiency of Process Discovery using Graph Database through Integrating Laravel and Neo4J

MUHAMMADTAUFIQULSA`DI
NRP 05111640000053

Supervisor I
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.

Supervisor II
Kelly Rossa Sungkono, S.Kom., M.Kom

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

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

MENINGKATKAN EFISIENSI *PROCESS DISCOVERY* MENGGUNAKAN *GRAPH DATABASE* MELALUI INTEGRASI LARAVEL DENGAN NE04J

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Sistem Studi S-l Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh
MUHAMMAD TAUFIQULSA'DI
NRP. 05111640000053

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RiyanartoSarno, M.Sc., Ph.D.
NIP. 19590803 198601 1 001
(pembimbing 1)

Kelly Rossa Sungkono, S.Kom., M.Kom.
NPP : 1994201912088
(pembimbing2)

SURABAYA
AGUSTUS, 2020

[Halaman ini sengaja dikosongkan]

MENINGKATKAN EFISIENSI *PROCESS DISCOVERY* MENGGUNAKAN *GRAPH DATABASE* MELALUI INTEGRASI LARAVEL DENGAN NEO4J

Nama	: Muhammad Taufiqulsa`di
NRP	: 05111640000053
DosenPembimbing I	: Prof. Drs.Ec. Ir. RiyanartoSarno, M.Sc., Ph.D.
Dosen Pembimbing II	: Kelly Rossa Sungkono, S.Kom.,M.Kom.

Abstrak

Enterprise Resource Planning (ERP) adalah aplikasi yang digunakan untuk mendukung proses bisnis perusahaan. Permasalahan yang muncul pada ERP saat ini adalah analisis proses bisnis secara manual. Process discovery adalah studi untuk membantu perusahaan menganalisis proses bisnis mereka dengan membangun model proses secara otomatis. Permasalahaan terbaru dalam process discovery adalah invisible tasks in non-free choice. Algoritma Alpha$ adalah metode process discovery yang mampu menangani invisible tasks in non-free choice. Alpha$ adalah kombinasi algoritma Alpha++ dan Alpha#, dimana algoritma ini menggambarkan relasi aktivitas secara implisit dalam bentuk kombinasi aktivitas. Relasi implisit tersebut mengakibatkan Alpha$ melakukan pemeriksaan kombinasi tersebut untuk menemukan invisible tasks in non-free choice. Proses pemeriksaan kombinasi pada algoritma Alpha$ menghasilkan waktu kompleksitas tinggi.

Algoritma berbasis graph, yaitu Graph Parallel dan Graph Invisible Task, memanfaatkan graph database dalam process discovery. Graph database menyimpan aktivitas dan relasi antara aktivitas, sehingga algoritma berbasis graph mendeteksi relasi paralel dan invisible task berdasarkan relasi antar aktivitas di graph database. Terdapat beberapa kekurangan dari algoritma berbasis graph yang ada. Pertama, algoritma-

algoritma tersebut belum membangun aturan pendeteksi invisible tasks in non-free choice. Kedua, algoritma-algoritma tersebut tidak terintegrasi dengan ERP, sehingga terdapat proses pengambilan data di ERP dan konversi kedalam bentuk graph database.

Tugas akhir ini mengembangkan algoritma GIN, agloritma process discovery yang mampu menggambarkan invisible tasks in non-free choice secara efisien. Langkah efisiensi pertama adalah pengunaan graph database dengan mengembangkan algoritma berbasis graph yang ada untuk menggambarkan invisible tasks in non-free choice. Langkah efisiensi kedua adalah integrasi ERP dengan graph database sehingga log data dari ERP secara otomatis tersimpan dalam bentuk graph, menghilangkan proses pengambilan data dan konversi pada algoritma graph sebelumnya. Eksperimen dalam tugas akhir ini membandingkan hasil model proses dan kompleksitas waktu antara algoritma GIN dan algoritma Alpha++ dan Alpha$. Berdasarkan hasil model proses, fitness dan presisi algoritma Alpha++ adalah 0,98 dan 0,92. Algoritma Alpha$ dan algoritma GIN mendapat 1, baik fitness dan presisi. Berdasarkan kompleksitas waktu, algoritma GIN lebih baik, dimana Alpha++ dan Alpha$ adalah $O(n^4)$, sedangkan algoritma GIN adalah $O(n^3)$.

Kata kunci: graph database, invisible task, non-free-choice, process discovery

IMPROVING EFFICIENCY OF PROCESS DISCOVERY USING GRAPH DATABASE BY INTEGRATING LARAVEL AND NEO4J

Name	: Muhammad Taufiqulsa`di
NRP	: 05111640000053
Supervisor I	: Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.
Supervisor II	:Kelly Rossa Sungkono, S.Kom.,M.Kom.

Abstract

Enterprise Resource Planning (ERP) is an application used to support company business processes. The problem that arises in ERP today is the analysis of business processes doing manually. Process discovery is a study to help companies analyze their business processes by building process models automatically. The latest problem in the process discovery is invisible tasks in non-free choice. Alpha$ algorithm is a process discovery method that can handle invisible tasks in non-free choice. Alpha$ is a combination of the algorithm Alpha++ and Alpha#, where this algorithm describes the relation of activities implicitly in the form of a combination of activities. The implicit relation causes Alpha$ to check the combination to find invisible tasks in non-free choice. The combination check process on the Alpha$ algorithm produces a high complexity time.

Graph-based algorithms, namely Parallel Graph and Invisible Task Graph, utilize a graph database in process discovery. Graph databases store activities and relationships between activities, so graph-based algorithms detect parallel and invisible task relationships based on relationships between activities in the graph database. There are some disadvantages of existing graph-based algorithms. First, the algorithms have not yet established rules for detecting invisible tasks in non-free choice. Second, the algorithms are not integrated with ERP, so

there is a process of taking data in ERP and converting it into a graph database.

This final project develops the GIN algorithm, a process discovery algorithm that can efficiently describe invisible tasks in non-free choices. The first efficiency step is to use a graph database by developing existing graph-based algorithms to illustrate invisible tasks in non-free choice. The second efficiency step is the integration of ERP with database graph so that log data from ERP is automatically stored in graph form, eliminating the process of data retrieval and conversion in the previous graph algorithm. The experiments in this thesis compare the results of the process model and the time complexity between the GIN algorithm and the Alpha++ and Alpha$ algorithms. Based on the results of the process model, the fitness and the precision of the Alpha++ algorithm are 0.98 and 0.92. Alpha$ algorithm and GIN algorithm get 1, both fitness and precision. Based on time complexity, the GIN algorithm is better, where Alpha ++ and Alpha $ are $O(n^4)$*., while the GIN algorithm is* $O(n^3)$*.*

Keywords: graph database, invisible task, non-free-choice, process discovery

KATA PENGANTAR

Segala puji syukur penulis kepada Allah SWT atas segala nikmat dan karunia-Nya, sehingga tugas akhir berjudul "Meningkatkan Efisiensi *Process Discovery* menggunakan *Graph Database* melalui Integrasi Laravel dengan Neo4J ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan di Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. Terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Orang tua yang selalu memberikan dukungan berupa doa dan nutrisi selama ini.
2. Bapak Riyanarto Sarno dan Ibu Kelly Rossa Sungkono selaku dosen pembimbing yang telah bersedia meluangkan waktu selama proses pengerjaan Tugas Akhir.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang banyak memberikan ilmu dan bimbingan bagi penulis.
4. Seluruh staf dan karyawan FTIK ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Teman-teman 'KBS' dan TC Angkatan 2016 yang selalu mendukung selama proses pengerjaan tugas akhir.
6. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak sekali kekurangan. Dengan kerendahan hati, penulis memohon maaf sebesar-besarnya atas kekuranga tersebut.

Surabaya, Agustus 2020

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SINGKATAN

ERP	:	Enterprise Resource Planning
CSV	:	Comma Separated Value
CQL	:	Cypher Query Language
SQL	:	Structured Query Language
GIN	:	Graph for Invisible Task in Non-free Choice

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pengerjaan Tugas Akhir, serta sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Dengan adanya perkembangan teknologi, persaingan bisnis, dan inovasi, perusahaan atau organsasi dituntut untuk selalu memiliki strategi dalam merebut perhatian pasar dan memenuhi kebutuhan konsumen dengan kualitas yang baik dan waktu yang tepat. Dengan demikian, perusahaan harus melakukan peninjauan berkala terkait aktivitas yang dilakukan untuk mengevaluasi dan melakukan pembaharuan proses supaya tetap dapat bersaing di pasar global [1]. *Process discovery* [2] dilakukan untuk mengevaluasi aktivitas yang telah berjalan. *Process discovery* merupakan suatu tahap untuk menemukan model proses yang berisi urutan aktivitas suatu perusahaan. Model proses tersebut dapat dibentuk secara otomatis dari input berupa *event log* [3]. *Event log* tersebut berfungsi sebagai *audit trail* yang berisi aktivitas-aktivitas yang dieksekusi melalui suatu sistem *Enterprise Resource Planning* (ERP) [4].

Saat ini, *event log* yang ada disimpan dalam suatu *database* ERP. Untuk melakukan evaluasi terhadap proses yang berjalan, beberapa algoritma dapat diterapkan untuk membentuk model proses, seperti Alpha++ [5], Alpha# [6], Alpha$[7], Graph Invisible Task[8], dan terakhir Graph Paralel [9]. Dari seluruh permasalahan yang muncul dalam model proses, permasalahan paling terbaru adalah pembentukan *invisible tasks in non-free choice.*

Alpha$ adalah algoritma *process discovery* pertama yang memperkenalkan permasalahan *invisible tasks in non-free choice* [7]. Alpha$ menggambarkan relasi aktivitas secara implisit dalam bentuk kombinasi aktivitas (tupel). Alpha$ memeriksa semua

tupel untuk menemukan *invisible tasks in non-free choice*. Proses pemeriksaan dapat meningkatkan waktu algoritma Alpha$ sehingga kompleksitasnya menjadi tinggi.

Dikarenakan algoritma yang ada tidak dapat menyimpan relasi aktivitas secara langsung (eksplisit), maka muncul algoritma berbasis *graph*, yaitu Graph Parallel dan Graph Invisible Task, yang memanfaatkan *graph database* dalam *process discovery*. *Graph database* menyimpan aktivitas dan relasi antara aktivitas, sehingga algoritma berbasis *graph* dapat mendeteksi relasi paralel dan *invisible task* berdasarkan relasi antar aktivitas di *graph database*. Terdapat beberapa kekurangan dari algoritma berbasis *graph* tersebut. Pertama, algoritma-algoritma tersebut belum membangun aturan pendeteksi *invisible tasks in non-free choice*. Kedua, algoritma-algoritma tersebut tidak terintegrasi dengan ERP, sehingga terdapat proses pengambilan data di ERP dan konversi kedalam bentuk *graph database*. Proses pengambilan data yang telah disebutkan sebelumnya dimulai dari pengambilan *Event log* yang akan dianalisis. *Event Log* yang ada perlu diekspor terlebih dahulu kedalam file berekstensi *Comma Separated Value* (CSV) yang kemudian diubah menjadi bentuk .mxml atau .xes [2] menggunakan Disco Tools. *Event log* berekstensi .mxml atau .xes tersebut diimpor menggunakan tools seperti ProM Tools [10] untuk dibentuk model proses menggunakan suatu algoritma. Selain ProM Tools, Neo4J juga merupakan tools untuk membentuk model proses menggunakan *graph database* [11]. *Event log* berekstensi .csv yang telah diekspor dari database ERP kemudian diimpor ke Neo4J untuk dibentuk model prosesnya dengan menerapkan algoritma. Hal ini dinilai tidak efisien jika analis atau auditor ingin melakukan pengamatan langsung terhadap model proses yang berjalan.

Berdasarkan uraian permasalahan tersebut, Tugas Akhir ini mengusulkan algoritma dengan nama GIN yang mampu menggambarkan *invisible tasks in non-free choice* secara efisien. Algoritma GIN akan menggunakan *graph database* serta terintegrasi secara langsung dengan sistem ERP untuk

penyimpanan *event log*. Evaluasi algoritma GIN akan terdiri dari dua tahap, yaitu evaluasi hasil model proses dari sisi fitness dan presisi dan evaluasi kompleksitas waktu. Algoritma yang akan dijadikan pembanding adalah Algoritma Alpha++ dan Alpha$.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana metode *process discovery* untuk membangun *invisible tasks in non-free choice*?
2. Bagaimana metode untuk mengintegrasikan penyimpanan *event log* ERP ke dalam *graph database*?
3. Bagaimana hasil evaluasi antara metode *process discovery* yang diusulkan dengan metode yang ada sebelumnya, yaitu Alpha++ dan Alpha$?

1. 3 Batasan Permasalahan

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

1. Data yang digunakan pada Tugas Akhir ini adalah data log pengguna yang berisi CaseId suatu proses, aktivitas yang dijalankan, timestamp, nama user, modul/tab yang diakses dan value suatu aktivitas.
2. Bahasa pemrograman yang mengimplementasikan Tugas Akhir ini adalah PHP, SQL, dan Cypher Query.
3. *Library* yang digunakan untuk mengimplementasikan metode pada Tugas Akhir ini adalah *graphaware.*
4. Framework ERP yang digunakan pada Tugas Akhir ini adalah Laravel.
5. Perangkat lunak yang digunakan pada Tugas Akhir ini adalah Neo4j untuk pembuatan model *graph* dan *Visual Studio Code* untuk menyambungkan Laravel dengan *graph database* Neo4j.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah pembentukan metode penemuan *invisible tasks in non-free choice* serta peningkatan efisiensi *process discovery* dengan menggunakan *graph database* dan integrasi *graph database* dengan sistem ERP.

1.5 Manfaat

Tugas akhir ini diharapkan dapat membantu analis atau auditor untuk mengevaluasi proses yang berjalan melalui penggambaran model proses secara langsung tanpa melakukan proses konversi serta memberikan kontribusi bagi peneliti mengenai metode *process discovery* berbasis *graph* untuk pembentukan *invisible tasks in non-free choice*.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu sebagai berikut:

a. Studi literatur

Pada tahap ini, akan dipelajari beberapa referensi yang diperlukan untuk pengerjaan tugas akhir, yaitu dari referensi jurnal internasional, artikel, buku, dan Tugas Akhir terdahulu.

b. Analisis dan desain perangkat lunak

Pada tahap ini, akan dilakukan analisis terhadap penerapan algoritma yang telah ada lalu dibandingkan dengan algoritma yang penulis terapkan pada studi kasus.

c. Implementasi

Tahap implemetasi meliputi implementasi algoritma untuk mengintegrasikan ERP dengan Neo4J.Implementasi ini dilakukan dengan menggunakan Laravel dan PHP serta *Cypher Query Language.*

d. Penyusunan buku tugas akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini. Pada tahap ini juga disertakan hasil dari implementasi algoritma yang telah dibuat. Sistematika penulisan buku tugas akhir ini secara garis besar antara lain:

1. Pendahuluan
    a. Latar Belakang
    b. Rumusan Masalah
    c. Batasan Masalah
    d. Tujuan
    e. Manfaat
    f. Sistematika Penulisan
2. Tinjauan Pustaka
3. Metodologi Penelitian
4. Implementasi
5. Kesimpulan dan Saran
6. Daftar Pustaka

1.7 Sistematika Penulisan

Buku Tugas Akhir ini disusun dengan sistematika penulisan sebagai berikut:

Bab I Pendahuluan

Bab ini berisi latarbelakang, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan yang menjadi dasar dari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah

Bab ini berisi tentang pembahasan masalah, perencanaan pemecahan masalah, serta metode yang akan diimplementasikan pada bab V.

Bab IV Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan alur, proses dan perancangan antarmuka pada perangkat lunak.

Bab V Implementasi

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

Bab VI Pengujian Dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (usability) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VII Kesimpulan dan Saran

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

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan lampiran yang digunakan untuk menjabarkan hasil pengujian algoritma yang diusulkan.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir.

2.1 Penelitian *Process Discovery*

Tabel 2.1 Penelitian *Process Discovery*

Nama Penelitian	**Deskripsi Penelitian**	**Kelebihan**	**Kekurangan**
Mining process models with non-free-choice Constructs [5]	Paper ini membangun metode *process discovery* bernama Alpha++ yang dapat membentuk relasi *non-free choice*	Dapat membangun relasi *non-free choice*	• Tidak dapat membentuk *invisible taks in-non free choice* • Membentuk relasi aktivitas dari *event log* secara implisit (kombinasi aktivitas) • Tidak terintegrasi dengan sistem ERP
Mining Process Models with Prime Invisible Tasks [6]	Paper ini membangun metode *process discovery* bernama Alpha# yang	Dapat membangun *invisible tasks*	• Tidak dapat membentuk *invisible taks in-non free choice* • Membentuk relasi

	dapat membentuk *invisible tasks*		aktivitas dari *event log* secara implisit (kombinasi aktivitas) • Tidak terintegrasi dengan sistem ERP
Mining Invisible Tasks in Non-free-choice Constructs [7]	Paper ini membangun algoritma Alpha$, dimana algoritma ini mengembangkan Alpha# dan Alpha++	Dapat membentuk *invisible tasks in non-free choice*	• Membentuk relasi aktivitas dari *event log* secara implisit (kombinasi aktivitas) • Tidak terintegrasi dengan sistem ERP
Graph-Based Approach for Modeling and Matching Parallel Business Processes [9]	Paper ini membangun algoritma berbasis *graph* yang dapat mendeteksi relasi paralel di sebuah proses bisnis.	Membentuk relasi aktivitas dari *event log* secara eksplisit (*relationship types* pada *graph database*)	• Tidak dapat membentuk *invisible taks in-non free choice* • Tidak terintegrasi dengan sistem ERP
Graph-Based Algorithms	Paper ini mengajukan sebuah	• Membentuk relasi aktivitas dari	• Tidak dapat membentuk *invisible*

for Discoverin g a Process Model Containing Invisible Tasks [8]	algoritma berbasis *graph* yang dapat mendeteksi *invisible task*.	*event log* secara eksplisit (*relationship types* pada *graph database*) • Dapat membangun *invisible tasks*	*taks in-non free choice* • Tidak terintegrasi dengan sistem ERP

Penelitian – penelitian yang sudah ada saat ini belum melakukan integrasi *process discovery* yang dilakukan dengan sistem ERP itu sendiri serta algoritma – algoritma selain Alpha$ belum dapat menemukan *invisible tasks in non-free choice*. Oleh karena itu, tugas akhir ini mengusulkan metode *process discovery* dengan *graph database* yang terintegrasi dengan sistem ERP dan dapat menggambarkan *invisible tasks in non-free choice*.

2.2. *Event Log*

Event log, yang berfungsi sebagai input utama dalam Tugas Akhir, merupakan catatan aktivitas yang dieksekusi pada suatu sistem tertentu. *event log* terdiri dari beberapa informasi, yaitu id proses (caseID), aktivitas yang dieksekusi (activity atau event), waktu aktivitas yang dieksekusi (timestamp) dan tambahan informasi lainnya. Proses yang dimaksud disini ialah rangkaian yang terdiri dari dua atau lebih aktvitas yang dilakukan secara beruntun. Pada Tabel 2.2 setelah aktivitas user login ada aktivitas open discount berdasarkan kolom CreatedAt. Dari aktivitas user login sampai open discount bisa dikatakan sebagai proses dikarenakan rangkaian aktivitas tersebut lebih dari satu aktivitas. *Case* disini sebagai proses yang dimulai dari aktivitas pertama sampai aktivitas terakhir dimana aktivitas pertama dan terakhir tersebut akan selalu sama. Contoh dari *case* itu sendiri dapat dilihat pada Tabel 2.2 caseId 44 dimulai dari aktivitas user login sampai user logout begitu juga caseId 55 dan seterusnya. Aktivitas user

login disini sebagai aktivitas pertama suatu *case* dan aktivitas user logout sebagai aktivitas terakhir suatu *case*.

Tabel 2.2 menunjukkan contoh potongan *event log* yang digunakan pada Tugas Akhir ini. Timestamp yang dicatat dalam *event log* ini adalah double-timestamp yaitu CreateAt sebagai start time dan UpdateAt sebagai end time [12]. Kemudian, tambahan informasi lain yang ada pada *event log* ini adalah userId sebagai Id pengguna, CompanyId sebagai Id perusahaan, Value merupakan informasi tambahan yang berkaitan aktivitas, dan Division merupakan modul sistem yang diakses.

Tabel 2.2 Contoh Potongan *Event Log*

case Id	Activity	user Id	CompanyId	Value	Division	CreatedAt	UpdateAt
1	user login	3	3	BEGIN	LOGIN	2/1/2020 4:42	2/1/2020 4:42
1	open discount	3	3	TRUE	CASHIER	2/1/2020 4:52	2/1/2020 4:52
1	set discount end time	3	3	#########	CASHIER	2/1/2020 5:03	2/1/2020 5:03
1	set discount name	3	3	DISKON SPESIAL FEBRUARI	CASHIER	2/1/2020 5:13	2/1/2020 5:13
1	set discount type	3	3	PERCENTAGE	CASHIER	2/1/2020 5:23	2/1/2020 5:23
1	set discount start time	3	3	#########	CASHIER	2/1/2020 5:33	2/1/2020 5:33
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

.	.	.	.	.	.	.	.
44	user login	3	3	BEGIN	LOGIN	2/3/2020 15:19	2/3/2020 15:19
44	open purchasing	3	3	TRUE	PURCHASING	2/3/2020 15:29	2/3/2020 15:29
44	choose supplier	3	3	supplier 1	PURCHASING	2/3/2020 15:39	2/3/2020 15:39
44	choose expedition	3	3	shipping point	PURCHASING	2/3/2020 15:49	2/3/2020 15:49
44	buy items	3	3	items 1:2500, items 2:3000, items 3:3500	PURCHASING	2/3/2020 15:59	2/3/2020 15:59
44	input expedition price	3	3	1000	PURCHASING	2/3/2020 16:09	2/3/2020 16:09
44	purchasing items	3	3	TRUE	PURCHASING	2/3/2020 16:19	2/3/2020 16:19
44	payment with e-facture	3	3	TRUE	PURCHASING	2/3/2020 16:29	2/3/2020 16:29
44	make payment	3	3	TRUE	PURCHASING	2/3/2020 16:40	2/3/2020 16:40
44	look notification	3	3	TRUE	PURCHASING	2/3/2020 16:50	2/3/2020 16:50
44	finish transaction	3	3	TRUE	PURCHASING	2/3/2020 17:00	2/3/2020 17:00
44	logout	3	3	END	LOGOUT	2/3/20	2/3/20

						20 17:10	20 17:10
45	user login	3	3	BEGIN	LOGIN	2/3/20 20 17:20	2/3/20 20 17:20
45	open purchasing	3	3	TRUE	PURCHASING	2/3/20 20 17:30	2/3/20 20 17:30
45	choose supplier	3	3	supplier 1	PURCHASING	2/3/20 20 17:40	2/3/20 20 17:40
45	choose expedition	3	3	shipping point	PURCHASING	2/3/20 20 17:51	2/3/20 20 17:51
45	input expedition price	3	3	1000	PURCHASING	2/3/20 20 18:01	2/3/20 20 18:01
45	buy items	3	3	items 1:2500, items 2:3000	PURCHASING	2/3/20 20 18:11	2/3/20 20 18:11
45	purchasing items	3	3	TRUE	PURCHASING	2/3/20 20 18:21	2/3/20 20 18:21
45	direct payment	3	3	TRUE	PURCHASING	2/3/20 20 18:31	2/3/20 20 18:31
45	make payment	3	3	TRUE	PURCHASING	2/3/20 20 18:41	2/3/20 20 18:41
45	finish transaction	3	3	TRUE	PURCHASING	2/3/20 20 18:51	2/3/20 20 18:51
45	Logout	3	3	END	LOGOUT	2/3/20 20 19:01	2/3/20 20 19:01
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

.	.	.	.	.	.	.	.
120	user login	3	3	BEGIN	LOGIN	2/8/2020 17:45	2/8/2020 17:45
120	open selling	3	3	TRUE	SELLING	2/8/2020 17:55	2/8/2020 17:55
120	choose customer type	3	3	General	SELLING	2/8/2020 18:05	2/8/2020 18:05
120	choose payment type	3	3	Cash	SELLING	2/8/2020 18:16	2/8/2020 18:16
120	choose item	3	3	shampoo sachet clear:2000, shampoo sachet rejoice:2000, shampoo sachet zinc:1500	SELLING	2/8/2020 18:26	2/8/2020 18:26
120	sell item	3	3	TRUE	SELLING	2/8/2020 18:36	2/8/2020 18:36
120	Logout	3	3	END	LOGOUT	2/8/2020 18:46	2/8/2020 18:46

2. 2 Process Discovery

Process discovery adalah teknik menemukan model proses secara otomatis dari *event log* [13]. *Stakeholder* seringkali merasa kesulitan dalam memahami proses bisnis yang sedang berlangsung. Dengan adanya model proses, pemilik perusahaan dapat memahami prosesnya dengan baik, sehingga mampu mengidentifikasi adanya kesalahan [14], [15] dan mengomunikasikannya secara cepat kepada *stakeholder* perusahaan.

Terdapat beberapa algoritma pembentukan model proses secara otomatis yang telah dikembangkan dalam penelitian

sebelumnya. Beberapa algoritma tersebut adalah Alpha++ [5], Alpha# [6], Alpha$[7], Graph Invisible Task[8], dan terakhir Graph Paralel [9]. Terdapat berbagai permasalahan dalam pembuatan model proses, salah satunya dan yang paling terbaru adalah *invisible task in non-free choice*.

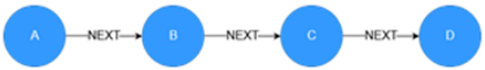
Alpha$ dikembangkan untuk mendeteksi *invisible tasks in non-free choice* [7]. Alpha$ menggunakan tupel yang hanya berisi aktivitas dan memeriksa semua tupel untuk menemukan *invisible tasks in non-free choice*. Proses pemeriksaan dapat meningkatkan waktu algoritma Alpha$ sehingga kompleksitasnya menjadi tinggi.

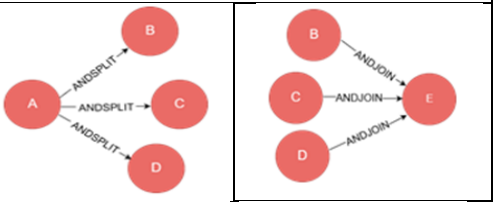
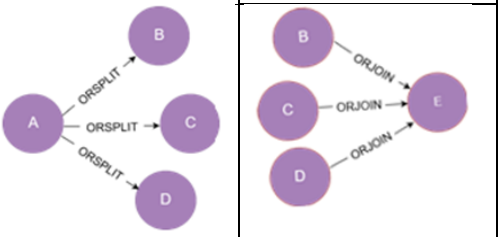
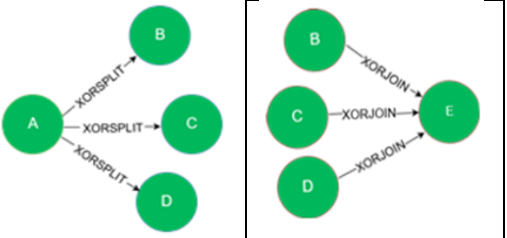
Selain Alpha$, terdapat algoritma berbasis *graph*, yaitu Graph Parallel dan Graph Invisible Task, yang memanfaatkan graph database dalam process discovery. *Graph database* menyimpan aktivitas dan relasi antara aktivitas, sehingga algoritma berbasis *graph* dapat mendeteksi relasi paralel dan *invisible task* berdasarkan relasi antar aktivitas di *graph database*.

2.3 Relasi Model Proses dan *Control-flow Pattern*

Model proses terdiri dari dua jenis relasi, yaitu relasi *sequence* dan relasi paralel. Relasi *sequence* digunakan untuk menggambarkan proses yang berjalan lurus secara berurutan atau sekuensial. Sedangkan relasi paralel digunakan untuk menggambarkan proses yang bercabang. Terdapat tiga macam control-flow pattern [14] untuk merepresentasikan relasi paralel, yaitu AND, OR, dan XOR [15], [16]. Tabel 2.3 menunjukkan contoh relasi dan perbedaan antar control-flow pattern pada graf model.

Tabel 2.3 Contoh Relasi dan *Control-flow Pattern* pada Graf Model Proses

Relasi dan *Control-flow Pattern*	Contoh *case* pada *Event Log*	Representasi Graf Model
Sequence	[A, B, C, D] [A, B, C, D] [A, B, C, D]	

Relasi dan *Control-flow Pattern*	Contoh *case* pada *Event Log*	Representasi Graf Model
AND Split | AND Join	[A, B, C, D, E] [A, B, D, C, E] [A, C, B, D, E] [A, C, D, B, E] [A, D, B, C, E] [A, D, C, B, E]	
OR Split | OR Join	[A, B, C, E] [A, B, D, E] [A, C, B, E] [A, C, D, E] [A, D, B, E] [A, D, C, E]	
XOR Split | XOR Join	[A, B, E] [A, C, E] [A, D, E]	

Pada relasi *sequence*, proses digambarkan secara lurus dan berurutan. Misalnya, setelah mengeksekusi aktivitas A, aktivitas B, C, dan D akan dieksekusi berurutan. Pada AND *pattern*, proses digambarkan secara bercabang di mana cabang tersebut akan dieksekusi semuanya. Pada OR *pattern*, proses digambarkan secara bercabang di mana cabang tersebut akan dipilih lebih dari satu tetapi tidak semua. Pada XOR *pattern*, proses digambarkan secara bercabang di mana cabang tersebut akan dipilih salah satu.

2.4 Neo4J

Neo4J adalah salah satu perangkat *graph database* yang didesain untuk mengoptimasi kecepatan manajemen, penyimpanan, dan perubahan node serta hubungan antar node itu sendiri.

Pada tabel relasional biasanya performa dari operasi *join* menurun secara exponensial bergantung dengan jumlah relasi, tetapi pada Neo4J yang memiliki sifat yang sama dengan *join* berfungsi sebagai navigasi dari satu node ke node yang lain,dimana operasinya bersifat linier [17].

Saat ini, penggunaan Neo4J pun sudah sangat luas. Tidak hanya untuk memodelkan suatu model proses, tetapi juga menganalisis jaringan, perangkat lunak, *routing*, dan manajemen proyek [18]. Dalam Neo4J, informasi yang berisikan aktivitas disimpan dalam *node label.* Sedangkan relasi yang menghubungkan antar *node label* adalah *relationship types.*

Neo4J menggunakan *Cypher Query Language* (CQL) untuk menampilkan informasi-informasi di dalamnya dan menentukan relasi antar *node.* CQL adalah deklarasi *query graph* yang mengizinkan pengguna untuk dapat berinteraksi dengan *graph* database. CQL dikembangkan dan terinspirasi dari berbagai pendekatan untuk mebuat *query* lebih expresif. Banyak kata kunci yang terinspirasi dari *Structured Query Language* (SQL). Selain itu, pendekatan pola ekspresi peminjaman diadaptasi dari SPARQL dan *list* dari semantik diambil dari Haskel dan Python[19].

2. 5 *Invisible Task*

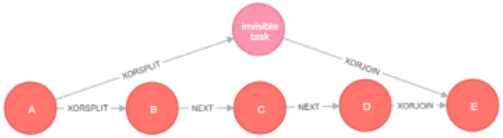
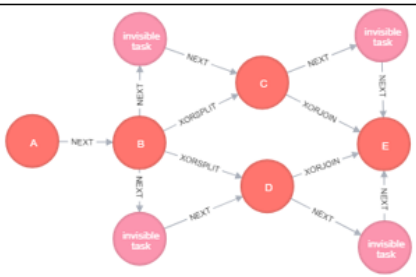
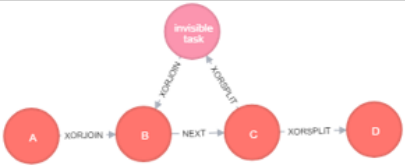
Invisible task adalah aktivitas tambahan yang tidak terlihat di *event log* tetapi ditampilkan dalam model. Kegunaan dari *invisible task* adalah untuk membantu menggambarkan proses secara sebenarnya dalam model proses. Terdapat tiga macam *invisible task prime* yaitu *Skip*, *Redo*, dan *Switch*.

Invisible task tipe *Skip* digunakan untuk menggambarkan aktivitas yang dapat dilewati. Pada Tabel 2.4, yang terdapat pada contoh tipe *skip* ada dua, yaitu [A, B, C, D, E] dan [A, E]. Pada *trace* kedua, aktivitas A tidak melewati aktivitas B, C, dan D melainkan langsung menujuk aktivitas E. Hal ini yang menyebabkan *invisible*

task dengan tipe *skip* perlu digambarkan dari aktivitas A keaktivitas E.

Invisible task tipe *redo* digunakan untuk menggambarkan aktivitas yang dapat diulang. Pada Tabel 2.4, *trace* yang terdapat pada contoh tipe *redo* ada dua, yaitu [A, B, C, D] dan [A, B, C, B, C, D]. Pada *trace* kedua, aktivitas B dan C diulang sebanyak sekali, maka *invisible task* perlu digambarkan diantara aktivitas C dan aktivitas B. *Invisible task* tipe *switch* digunakan untuk perpindahan eksekusi pada beberapa percabangan. Pada Tabel 2.4, *trace* yang terdapat pada contoh tipe *switch* ada dua, yaitu [A, B, D, E] dan [A. C, D, E] yang membuat adanya percabangan atau pilihan antara aktivitas A dengan C, A dengan B, B dengan D, dan C dengan D. Hal ini menyababkan *invisible task* perlu digambarkan di percabangan, baik percabangan dengan tipe split maupun join.

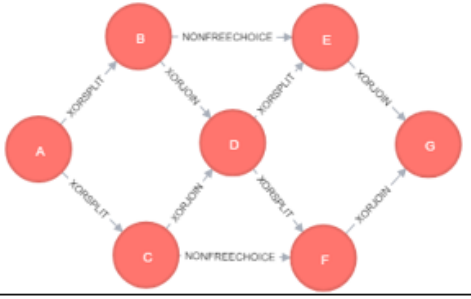
Tabel 2.4 Jenis *Invisible Task*

Condition	Trace	Process Model (Neo4J)
Skip	[A,B,C,D,E] [A,E]	
Switch	[A,B,D,E] [A,C,D,E]	
Redo	[A,B,C,D] [A,B,C,B,C,D]	

2. 6. *Non-Free Choice*

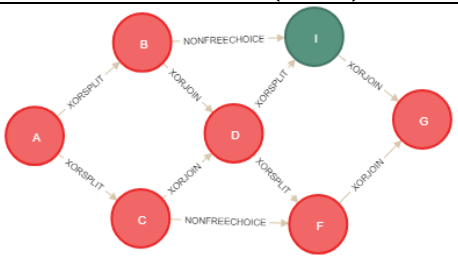
Non-free choice adalah hubungan tidak langsung antara dua *activity* yang saling berhubungan. Hubungan tidak langsung ini terjadi antara aktivitas sebelum relasi pilihan (XOR) dan setelah relasi paralel pilihan (XOR). Algoritma pertama yang memperkenalkan *non-free choice* adalah Algoritma Alpha++. Algoritma ini mengembangkan algoritma Alpha dan Alpha+. Algoritma Alpha merupakan algoritma dasar yang dapat digunakan untuk menganalisa dan menghasilkan *activity* dari *case*. Kekurangan dari algoritma Alpha adalah length *one loop*, *length two loop*, *invisible task*, *duplicate task*, *implicit places*, dan *non-free choice*. Perbaikan atau pengembangan dari algoritma Alpha adalah algoritma Alpha+. Kelebihan algoritma Alpha+ dibandingkan algoritma Alpha antara lain: algoritma Alpha+ dapat melakukan *mining* terhadap *short loops* seperti length one loop yang tidak terdeteksi pada algorithma Alpha. Kemudian Alpha++ dapat menangani *Indirect Dependency* merefleksikan *non-free choice*. Tabel 2.5 menunjukkan contoh hasil *non-free choice.*

Tabel 2.5 Contoh Hasil Penemuan *Non-Free-Choice*

Condition	Trace	Process Model (Neo4J)
Non-free-choice	[A,B,D,E,G] [A,C,D,F,G]	

2. 7 *Invisible Tasks in Non-Free Choice*

Tabel 2.6 Contoh Hasil Penemuan *Invisible Tasks in Non-Free Choice*

Condition	Trace	Process Model(Neo4J)
Invisible Tasks in Non-Free Choice	[A,B,D,G] [A,C,D,F,G]	

Invisible tasks in non-free choice merupakan sebuah kondisi dimana sebuah node *Invisible tasks* memiliki relasi *indirect depedency* terhadap sebuah aktivitas [7]. Hal ini terjadi karena adanya aktivitas yang aktivitas setelahnya dapat dilewati dikarenakan pengguna melakukan aktivitas tertentu sebelum aktivitas tersebut. Tabel 2.6 merupakan contoh dari *Invisible tasks in non-free choice. Invisible task* pada Tabel 2.6 digambarkan melalui lingkaran dengan label I berwarna hijau.

2. 8 Evaluasi Hasil Model Proses

Nilai fitness sebuah *process model* dapat diperoleh dengan menghitung jumlah *case* yang digambarkan dalam *process model* dibagi dengan total *case* dalam *event log*. Sementara nilai presisi sebuah *process model* dapat diperoleh dengan menghitung jumlah *trace* yang digambarkan dalam model dibagi dengan jumlah *trace* yang adadalam *event log*. Rumus untuk menghitung fitness dan presisi dapat dilihat pada Persamaan (1) dan Persamaan (2) masing-masing.

$$fitness = \frac{case\ in\ the\ model}{total\ case\ in\ the\ event\ log} \quad (1)$$

$$precision = \frac{trace\ in\ the\ model}{trace\ in\ the\ event\ log} \quad (2)$$

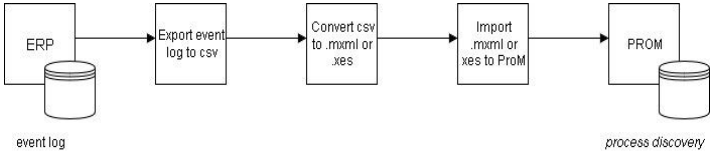
Dimana:

fitness	:	ketepatan model dalam melakukan pencatatan *case*
precision	:	ketepatan model dalam membentuk lintasan/*trace*
case in the model	:	banyaknya *case* dalam model
total case in the event log	:	banyaknya *case* dalam *event log* (data sebenarnya)
trace in the model	:	banyaknya *trace* dalam model
trace in the event log	:	banyaknya *trace* dalam *event log* (data sebenarnya)

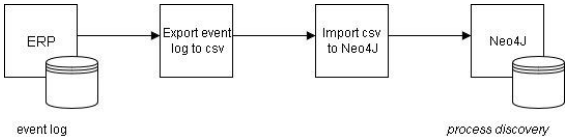
BAB III
METODE PEMECAHAN MASALAH

Pada Tugas Akhir ini, penulis akan membangun algoritma penyimpanan *event log* ke dalam *graph database* untuk *process discovery* secara langsung tanpa adanya proses ekspor *event log* dari database ERP dan impor *event log* ke dalam *tools* lain atau tanpa adanya proses transformasi *event log* ke dalam bentuk .mxml atau .xes. Hal ini akan membuat *process discovery* menjadi lebih efisien dibandingkan metode-metode sebelumnya. Efisien yang dimaksud disini adalah sebuah kondisi yang memaksimalkan hasil dari suatu aksi yang berelasi dengan sumber daya yang digunakan [23] dengan

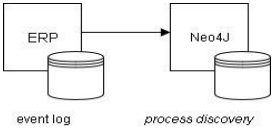
Gambar 3.**1** menunjukkan perbandingan metode sebelumnya dengan metode usulan. Pengerjaan tugas akhir ini dibagi menjadi dua tahapan, yaitu menghubungkan ERP dengan Neo4J dan *process discovery.*



(a) Metode *Process Discovery*saat ini menggunakan ProM Tools [2], [3]



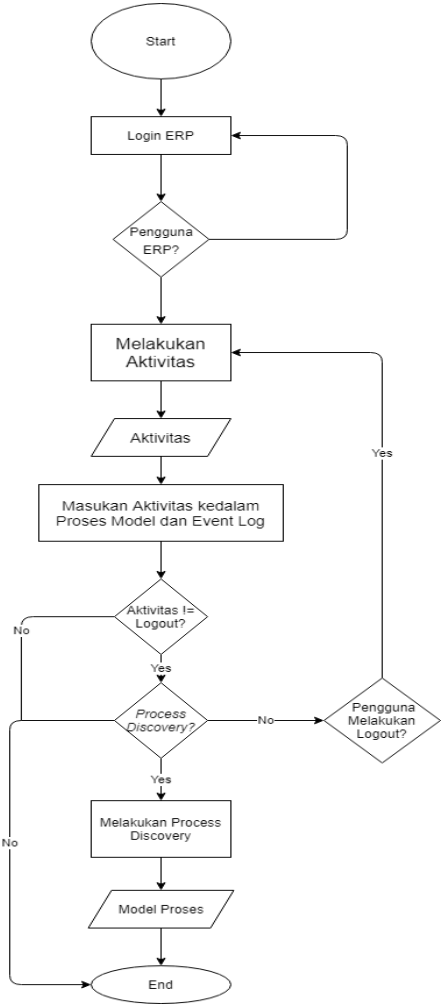
(b) Metode *Process Discovery* saat ini menggunakan Neo4J [8]



(c) Metode Usulan *Process Discovery* dengan Integrasi ERPdengan Neo4J

Gambar 3.1 Perbandingan Metode *Process Discovery*

3.1 Cakupan Permasalahan



Gambar 3.2 Alur Proses ERP sampai *Process Discovery* di dalam Neo4j

Tugas akhir ini memecahkan permasalahan metode-metode *process discovery* yang tidak terintegrasi secara langsung dengan sistem ERP. Permasalahan lain yang diangkat oleh penulis pada tugas akhir ini adalah menemukan sebuah metode baru yang dapat menemukan *invisible tasks in non-free choice* dengan tingkat kompleksitas algoritma yang lebih rendah dibanding dengan algoritma *process discovery* sebelumnya. Gambar 3.2 menjelaskan alur proses ERP sampai dengan *process discovery* didalam Neo4j.

3.2 Metode Integrasi ERP dan *Graph Database*

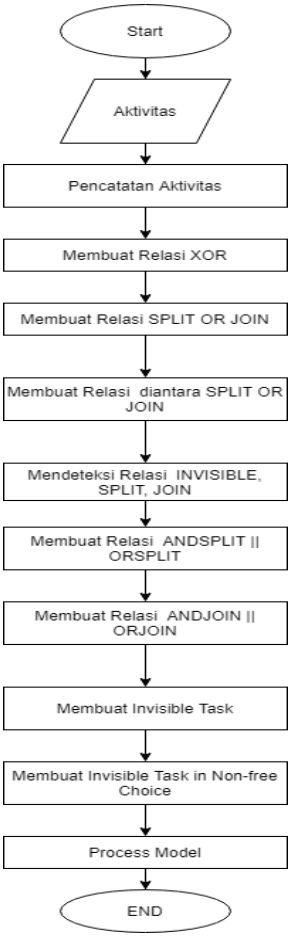
Bagian ini menjelaskan tentang bagaimana cara menghubungkan ERP dengan Neo4J sehingga *event log* yang dihasilkan ERP dapat langsung tersimpan ke dalam database Neo4J. ERP dengan Neo4J dihubungkan menggunakan Laravel dengan mengimplementasikan PHP. Tabel 3.1 menunjukkan script PHP untuk mengoneksikan ERP ke Neo4J melalui Laravel. Langkah pertama adalah mengambil *script* `autoload.php`, lalu mendeklarasikan `ClientBuilder` dari *library* `GraphAware`. Tahap selanjutnya adalah membuat koneksi menggunakan `ClientBuilder`. Laravel dengan Neo4J berhasil terhubung.

Tabel 3.1 Mengoneksikan ERP dengan Neo4J

```
<?php
    require_once 'vendor/autoload.php';
    use GraphAware\Neo4j\Client\ClientBuilder;
    $client = ClientBuilder::create() ->
    addConnection('default','http://neo4j:password@localhost:747
    4')
//Example for HTTP connection configuration (port is optional)
    ->
    addConnection('bolt','bolt://neo4j:password@localhost:7687')
// Example for BOLT connection configuration (port is optional)
    -> build();
```

3.3 Metode *Process Discovery*

Bagian ini menjelaskan tentang bagaiman a alur dari metode *process discovery* yang diajukan. Gambar 3.3 merupakan flowchart dari metode yang diajukan.



Gambar 3.3 Flowchart Algoritma GIN

3.3.1 Pencatatan Aktivitias

Tahap pencatatan *event log* dalam *graph database* dimulai ketika pengguna login ke sistem ERP hingga pengguna logout. Pencatatan dilakukan dengan menggunakan algoritma pada Tabel 3.2. Pertama, algoritma memeriksa log yang telah disimpan di Neo4J. Jika log terakhir ditemukan dalam grafik-database, aktivitas login dimasukkan dalam *case* terakhir + 1. Namun, jika log terakhir tidak ditemukan, maka aktivitas login dicatat sebagai *case* pertama. Koneksi Neo4J dilakukan menggunakan `GraphAwarelibrary` dengan menghubungkan `ClientBuilder` dari `GraphAware`. Pada saat itu, *case* saat ini akan dimasukkan dalam session untuk digunakan pada tahap selanjutnya.

Tabel 3.2 Algoritma Pencatatan aktivitas login

Keterangan	**Rincian Keterangan**

Algoritma	lastLogCase=DB::table(‘event_logs’)-
	>latest(‘case’)->first();
	if lastLogCase is empty then
	  set caseId to 1
	else
	  set caseId to lastLogCase+1
	  caseId++
	endif
	set caseId, activity, userId, companyId, value, division, and timestamp
	create event log
	Session([‘case’=>caseId])
	Store the event log into graphDatabase
	caseData <- MERGE activity and value
	modelBusiness <- MERGE activity and value

Luaran	**Node with activity as attribute on model process and cases**

Dimana:

lastLogCase	:	*case* terakhir pada database SQL
caseId	:	ID dari *case* pada tabel SQL
activity	:	aktivitas yang dilakukan pengguna
userId	:	ID dari pengguna pada database SQL
companyId	:	ID dari perusahaan pada database SQL
value	:	nilai atau hasil dari suatu aktivitas
division	:	modul yang diakses oleh pengguna
timestamp	:	waktu pada saat melakukan aktivitas
caseData	:	jalur aktivitas yang dilakukan oleh pengguna dalam satu *case*
modelBusiness	:	kumpulan jalur aktivitas yang dilakukan dalam satu perusahaan

Tabel 3.3 digunakan untuk membuat hubungan antar aktivitas pada *event log*. Pada dasarnya, chyper dari Neo4j digunakan untuk menghubungkan semua node ke dalam linked-list. Neo4J akan membaca semua aktivitas dalam label aktivitas, mengurutkannya berdasarkan waktu aktivitas dilakukan, dan menyimpan aktivitas ke dalam array.

Tabel 3.3 Algoritma GIN untuk Membuat Relasi *Sequence*

Keterangan	**Rincian Keterangan**

Algoritma	for i=0 until i=count(activity-1):
	a, b = all nodes of caseActivity
	if (activity[i]).caseId=(activity[i+1]).caseIdand
	(activity[i]).name=a.nameand
	(activity[i+1]).name=b.name:
	create relation(activity [a]-[r:NEXT]->activity[b])
	endif
	endfor

Luaran	**A graph model with sequential relationships**

Dimana:

caseId : ID dari *case* suatu aktivitas

activity : aktivitas yang dilakukan pengguna
Activities : sekumpulan aktivitas yang dilakukan pengguna
name : nama dari suatu aktivitas

Langkah selanjutnya adalah penemuan control-flow patterns, yaitu AND, OR, dan XOR seperti yang dapat dilihat pada Table 2.3 Control-flow pattern digunakan untuk menentukan jenis relasi paralel. Relasi paralel dapat dibagi menjadi tiga jenis, yaitu XOR, AND, dan OR.

3.3.2 Membuat Relasi XOR

Terdapat dua jenis relasi XOR yaitu relasi XORSPLIT dan relasi XORJOIN. Relasi XORSPLIT terbentuk ketika sebuah aktivitas hanya bisa memilih 1 dari kumpulan *trace* setelah aktivitas tersebut dimana aktivitas pertama pada setiap *trace* tidak saling terhubung satu dengan lainnya. Penemuan relasi XORSPLIT dapat dilihat pada Tabel 3.4.

Tabel 3.4 Algoritma GIN untuk Membuat Relasi XORSPLIT

Keterangan Tipe Relasi	**Rincian Keterangan**

XORSPLIT	for activityNowin activities:
	if nextActivityNow = activityX and count(nextActivitesNow) > 1 and (prevActivitiesX) = 1:
	merge((activityNow)-[:XORSPLIT]->(activityX))
	delete((activityNow)-[:NEXT]->(activityX))
	endif
	endfor

Luaran	**XORSPLIT relation**

Dimana:
NextActivityNow : aktivitas setelah aktivitas saat ini
NextActivitiesNow : kumpulan aktivitas setelah aktivitas saat ini

PrevActivitiesX : kumpulan aktivitas sebelum aktivitasX
Activities : kumpulan aktivitas didalam model proses

Setelah penemuan relasi XORSPLIT, tahap selanjutnya adalah menemukan relasi XORJOIN. Relasi XORJOIN terbentuk ketika sebuah aktivitas merupakan satu-satunya aktivitas terakhir dari kumpulan *trace* sebelum aktivitas tersebut dimana aktivitas terakhir pada setiap *trace* tidak saling terhubung satu dengan lainnya. Penemuan relasi XORJOIN dapat dilihat pada Tabel 3.5.

Tabel 3.5 Algoritma GIN untuk Membuat Relasi XORJOIN

Keterangan Tipe Relasi	**Rincian Keterangan**

XORJOIN	for activityNowin activities:
	if prevActivityNow = activityX and count(prevActivitesNow) > 1 and (nextActivitiesX) = 1:
	create(activityX)-[:XORJOIN]->(activityNow)
	delete(activityX)-[:NEXT]->(activityNow)
	endif
	endfor

Luaran	**XORJOIN relation**

Dimana:
PrevActivityNow : aktivitas sebelum aktivitas saat ini
PrevActivitiesNow : kumpulan aktivitas sebelum aktivitas saat ini
ActivityNow : aktivitas saat ini
NextActivitiesX : kumpulan aktivitas setelah aktivitas X
Activities : sekumpulan aktivitas yang dilakukan pengguna

3.3.3 Membuat Relasi SPLIT OR JOIN

Relasi SPLIT OR JOIN digunakan untuk menemukan relasi SPLIT dan JOIN yang ada sekaligus menghapus semua

relasi yang tidak diperlukan didalam SPLIT dan JOIN. Penemuan relasi SPLIT OR JOIN dapat dilihat pada Tabel 3.6.

Tabel 3.6 Algoritma GIN untuk Membuat Relasi SPLIT OR JOIN

Keterangan Tipe Relasi	**Rincian Keterangan**

Membuat relasi SPLIT OR JOIN	for a in activities:
	if (a)-[:NEXT]->(n) AND size(a)-->() AND >= 2 AND size((n)<--()) >=2 :
	create(a)-[:SPLIT_OR_JOIN]->(n)
	endif
	endfor

Luaran	**SPLIT OR JOIN relation**

Dimana:
Activities : kumpulan aktivitas didalam model proses

3.3.5 Menghapus Relasi diantara SPLIT OR JOIN

Penghapusan relasi untuk aktivitias setelah relasi split maupun sebelum relasi join ini dilakukan untuk memudahkan *chyper* dalam mencari relasi paralel AND , OR, dan *invisible task*. Penghapusan relasi diantara relasi SPLIT OR JOIN dapat dilihat pada Tabel 3.7.

Tabel 3.7 Algoritma GIN untuk Menghapus Relasi diantara SPLIT OR JOIN

Keterangan Tipe Relasi	**Rincian Keterangan**

Menghapus SPLIT OR JOIN	for a in activities:
	if (a)-[:NEXT]->(n) AND exist ()-[:SPLIT_OR_JOIN]->(a)AND exist ()-[:SPLIT_OR_JOIN]->(n) :
	delete (a)-[:NEXT]->(n)
	else (a)-[:NEXT]->(n)
	endif

	endfor
Luaran	**relasi diantara SPLIT OR JOIN relation terhapus**

Dimana:
activities : kumpulan aktivitas didalam model proses

3.3.6 Mendeteksi Relasi INVISIBLE, SPLIT, JOIN

Tahap selanjutnya adalah pendeteksian relasi SPLIT, JOIN, dan INVISIBLE. Relasi JOIN adalah relasi yang terbentuk dikarenakan suatu aktivitas memiliki banyak aktivitas setelahnya pada model proses , relasi akan terbentuk dari aktivitas tersebut ke semua aktivitas setelahnya. Sementara relasi JOIN adalah relasi yang terbentuk dikarenakan suatu aktivitas memiliki banyak aktivitas sebelumnya pada model proses, relasi akan terbentuk dari aktivitas tersebut ke semua aktivitas sebelumnya.

Relasi INVISIBLE disini dibentuk untuk menandai relasi yang tergolong relasi JOIN dan relasi SPLIT yang nantinya akan dimasukkan *invisible task* di relasi tersebut. Semua node yang memiliki relasi SPLIT ke node lain akan ditandai sebagai splitNode dan disimpan didalam variabel splitNodes. Sementara semua node yang memiliki relasi JOIN dari node lain akan ditandai sebagai joinNode dan disimpan didalam variabel joinNodes. SplitNodes dan joinNodes akan digunakan untuk menemukan relasi ANDSPLIT, ORSPLIT, ANDJOIN, dan ORJOIN. Pembuatan relasi INVISIBLE, SPLIT, dan JOIN dapat dilihat pada Tabel 3.8.

Tabel 3.8 Algoritma GIN untuk Mendeteksi Relasi INVISIBLE, SPLIT, JOIN

Keterangan Tipe Relasi	**Rincian Keterangan**

Membuat relasi INVISIBLE, SPLIT,	let splitNodes = array[]
	let joinNodes = array[]
	for a in activities:
	if (a)-[:SPLIT_OR_JOIN]->(n) AND size((a)--

JOIN	>())> 1 AND size((n)<--()) > 1 :
	create (a)-[:INVISIBLE]->(n)
	elseif (a)-[:SPLIT_OR_JOIN]->(n) AND size((a)-->()) > 1 :
	create (a)-[:SPLIT]->(n)
	splitNodes.append(a)
	elseif (a)-[:SPLIT_OR_JOIN]->(n) AND size((n)<--()) > 1 :
	create (a)-[:JOIN]->(n)
	joinNodes.append(n)
	endif
	endfor

Luaran	**INVISIBLE, SPLIT, and JOIN Relation**

Dimana:
Activities : kumpulan aktivitas didalam model proses

3.3.7 Membuat Relasi ANDSPLIT || ORSPLIT

Relasi ANDSPLIT terbentuk jika jumlah node aktivitas setelah splitNode sama dengan jumlah node aktivitas pada salah satu *case*, yang dimulai dari splitNode sampai salah satu node aktivitas setelah splitNode. Jika jumlah node aktivitas yang ditemukan pada *case* tersebut kurang dari jumlah node aktivitas setelah splitNode maka relasi yang terbentuk adalah ORSPLIT. Pembuatan relasi ANDSPLIT dan ORSPLIT dapat dilihat pada Tabel 3.9.

Tabel 3.9 Algoritma GIN untuk Membuat Relasi ANDSPLIT || ORSPLIT

Keterangan Tipe Relasi	Rincian Keterangan
ANDSPLIT || ORSPLIT	for splitNode in splitNodes: nextNodes=Chyper(MATCH (n:ModelBusiness{activity:splitNode})-[:SPLIT]->(a:ModelBusiness) RETURN a)

	logData = EventLog::where([['company_id', Auth::user()->company_id],['activity', splitNode]])->first()
	id = logData->case
	logData = EventLog::where([['company_id', Auth::user()->company_id],['case', id]])->orderBy('created_at')->get()
	counter = 0
	count = false
	foreach logData as data
	if count && !in_array(data->activity,nextNodes))
	break
	endif
	if in_array($data->activity,nextNodes)
	count = true
	counter+=1
	endif
	endforeach
	if count(nextNodes) = count(counter):
	merge((splitNode)-[:ANDSPLIT] ->(nextNodes))
	delete((nextNodes)-[:NEXT]->())
	endif
	if count(nextNodes) <count(counter):
	merge((splitNode)-[:ORSPLIT] ->(nexNodes))
	delete((nextNodes)-[:NEXT]->())
	endif
	endfor

Luaran	**ANDSPLIT or ORSPLIT Relation**

Dimana:

SplitNodes	:	kumpulan node yang memiliki banyak relasi NEXT ke node setelahnya
nextNode	:	node setelah relasi split
nextNodes	:	sekumpulan node setelah relasi split
counter	:	jumlah nextNode yang membentuk sequence

3.3.8 Membuat Relasi ANDJOIN || ORJOIN

Relasi ANDJOIN terbentuk jika jumlah node aktivitas sebelum joinNode dan memiliki relasi JOIN sama dengan jumlah node aktivitas pada salah satu *case*, yang dimulai dari salah satu node aktivitas sebelum joinNode tersebut sampai joinNode. Jika jumlah node aktivitas yang ditemukan pada *case* tersebut kurang dari jumlah node aktivitas sebelum joinNode maka relasi yang terbentuk adalah ORJOIN. Pembuatan relasi ANDJOIN dan ORJOIN dapat dilihat pada Tabel 3.10.

Tabel 3.10 Algoritma GIN untuk Membuat Relasi ANDJOIN || ORJOIN

Keterangan Tipe Relasi	Rincian Keterangan
ANDJOIN || ORJOIN	for joinNode in joinNodes: prevNodes =Chyper(MATCH (n:ModelBusiness{activity: splitNode})-[:JOIN]- >(a:ModelBusiness) RETURN n) logData = EventLog::where([['company_id', A uth::user()- >company_id],['activity', splitNode]])->first() id = logData->case logData = EventLog::where([['company_id', Auth::user()->company_id],['case', id]])- >orderBy('created_at')->get() counter = 0

	count = false
	foreach logData as data
	if count && !in_array(data->activity,prevNodes))
	break
	endif
	if in_array($data->activity,prevNodes)
	count = true
	counter+=1
	endif
	endforeach
	if count(prevNodes) = count(counter):
	delete((prevNodes)-[:NEXT]->())
	merge((prevNodes)-[:ANDJOIN]->(:joinNode))
	endif
	if count(prevNodes) <count(counter):
	delete((prevNodes)-[:NEXT]->())
	merge((prevNodes)-[:ORJOIN]->(:joinNode))
	endif
	endfor

Luaran	**ANDJOIN or ORJOIN Relation**

Dimana:

JoinNodes : kumpulan node yang memiliki relasi join ke nodetersebut

prevNode : node yang memiliki relasi join ke joinNode

prevNodes : sekumpulan node yang memiliki relasi join kejoinNode

counter : jumlah prevNodes yang membentuk sequence

Perbedaan metode yang diusulkan dengan metode sebelumnya yang menggunakan *graph database* Neo4J untuk membentuk

model proses adalah cara pengambilan kumpulan aktivitas dan pembuatan relasi paralel OR dan AND. Metode sebelumnya mengambil kumpulan aktivitas dari csv dengan melakukan operasi *chyper* didalam Neo4J yang tergambar di Gambar 3.4.

```
LOAD CSV with headers FROM "file:///*.csv"
  AS line
Merge (:Activity {CaseId:line.Case_ID,
Name:line.Activity,
StartTime:line.Start_Timestamp,
EndTime:line.End_Timestamp,
Actor:line.Actor })
LOAD CSV with headers FROM "file:///*.csv"
  AS line
Merge (:CaseActivity {Name:line.Activity })
//Discovery Sequence
match (c:Activity)
with COLLECT(c) AS Caselist
unwind range(0,Size(Caselist) - 2) as idx
with Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
where s1.CaseId = s2.CaseId AND s1.Name = a.Name AND s2.Name = b.Name
merge (a)-[:NEXT {relation:"NEXT"}]->(b)
```

Gambar 3.4 Cypher Pengambilan Data

Sementara metode saat ini melakukan pencatatan dan pembuatan model setiap kali rangkaian aktivitas selesai menggunakan algoritma pada tabel 3.3. Dari segi pembuatan relasi paralel OR dan AND metode sebelumnya melakukan operasi *chyper* didalam Gambar 3.5 dan Gambar 3.6.

```
match (n)-[:NEXT]->(a)
where size((n)-->()) > 1 and size((a)-->()) >= 2
create (n)-[:ANDSPLIT {relation:"AND Split"}]->(a)
return distinct 'ANDSPLIT', n.Activity, a.Activity
match (n)-[:NEXT]->(a)
where size((n)-->()) >= 2 and size((a)<--()) = 2
create (n)-[:ANDJOIN {relation:"AND Join"}]->(a)
return distinct 'ANDJOIN', n.Activity, a.Activity
```

Gambar 3.5 CypherPenemuan AND di Neo4J

```
match (n)-[:NEXT]->(a)
where size((n)-->()) > 1 and size((a)-->()) >= 2
create (n)-[:ORSPLIT {relation:"OR Split"}]->(a)
return distinct 'ORSPLIT', n.Activity, a.Activity
match (n)-[:NEXT]->(a)
where size((n)-->()) >= 2 and size((a)<--()) = 3
create (n)-[:ORJOIN {relation:"OR Join"}]->(a)
return distinct 'ORJOIN', n.Activity, a.Activity
```

Gambar 3.6 Cypher Penemuan OR di Neo4J

Sementara metode saat ini membandingkan jumlah node aktivitas setelah relasi SPLIT dengan jumlah node akitivitas pada salah satu *case* yang memiliki node aktivitas splitNode,seperti yang dijelaskan pada Tabel 3.9 untuk menentukan relasi yang terbentu AND SPLIT atau OR SPLIT. Sementara untuk menentukan relasi yang terbentuk AND JOIN atau OR JOIN, algoritma GIN membandingkan jumlah node aktivitas sebelum relasi JOIN dengan jumlah node aktivitas pada salah satu *case* yang memiliki node aktivitas joinNode, seperti yang dijelaskan pada Tabel 3.10

3.3.9 Membuat Invisible Task

Setelah membentuk semua relasi paralel selanjutnya adalah memasukan semua *invisible task* didalam relasi INVISIBLE. *Invisible task* akan membentuk 2 relasi. Relasi pertama yaitu relasi yang tertuju pada *invisible task* yang berasal dari suatu node dan memiliki jenis relasi yang sama dengan relasi yang terbentuk dari node tersebut ke node yang lain. Relasi kedua yaitu relasi yang berasal dari *invisible task* ke suatu joinNode dan memiliki jenis relasi yang sama dengan relasi yang tertuju ke joinNode tersebut. Pembuatan *invisible task* dapat dilihat pada Tabel 3.11.

Tabel 3.11 Algoritma GIN untuk Membuat *Invisible Task* didalam Relasi Paralel

Keterangan Tipe Relasi	**Rincian Keterangan**

Membuat INVISIBLE TASK didalam Relasi Paralel	for a in activities:
	if (a)-[:ANDSPLIT]->() and (a)-[:INVISIBLE]->(x):
	create (INVISIBLETASK)
	create (a)-[:ANDSPLIT]->(INVISIBLETASK)
	create (x)-[:ANDJOIN]->(INVISIBLETASK)
	elseif (a)-[:ORSPLIT]->() and (a)-[:INVISIBLE]->(x):
	create (INVISIBLETASK)
	create (a)-[:ORSPLIT]->(INVISIBLETASK)
	create (x)-[:ORJOIN]->(INVISIBLETASK)
	elseif (a)-[:XORSPLIT]->() and (a)-[:INVISIBLE]->(x):
	create (INVISIBLETASK)
	create (a)-[:XORSPLIT]->(INVISIBLETASK)
	create (x)-[:XORJOIN]->(INVISIBLETASK)
	elseif ()-[:ORJOIN]->(a) and (x)-[:INVISIBLE]->(a):
	create (INVISIBLETASK)
	create (a)-[:ORSPLIT]->(INVISIBLETASK)
	create (x)-[:ORJOIN]->(INVISIBLETASK)
	endif
	endfor

Luaran	**Create Node as INVISIBLE TASK**

Dimana:

nextActivityNow : aktivitas setelah aktivitas saat ini

nextActivitiesNow : kumpulan aktivitas setelah aktivitas saat ini

nextFromNextActivitiesNow	:	kumpulan aktivitas setelah nextActivityNow saat ini
activityNow	:	aktivitas saat ini
splitNodes	:	kumpulan node yang memiliki banyak relasi NEXT ke node setelahnya
prevActivityNow	:	aktivitas sebelum aktivitas saat ini
prevActivitiesNow	:	kumpulan aktivitas sebelum aktivitas saat ini

3.3.10 Invisible Task in Non-free Choice

Langkah terakhir adalah penemuan *invisible tasks in non-free choice*. Relasi *non-free choice* menghubungkan suatu aktivitas dalam satu relasi pilihan denganaktivitas lain dalam relasi pilihan berikutnya. Relasi ini menunjukkan bahwa aktivitas relasi pilihan berikutnya tidak dapat dipilih secara bebas dan dipengaruhi oleh aktivitas relasi pilihan sebelumnya.Tabel 3.12 adalah contoh relasi *non-free choice* menggunakan Neo4J. Seperti yang digambarkan dalam Tabel 3.12, simpul F dan simpul E tidak dapat dipilih secara bebas meskipun hubungan antara D ke F dan D ke E adalah XORSPLIT, sebagai gantinya pilihan simpul F hanya dapat diambil jika aktivitas sebelumnya yang diambil adalah simpul C dan pilihan simpul E hanya dapat diambil jika aktivitas sebelumnya yang diambil adalah simpul B. Relasi tersebut digambarkan oleh relasi NONFREECHOICE. Menggunakan algoritma pada Tabel 3.13 algoritma GIN menemukan beberapa node dengan hubungan keluar XORJOIN, beberapa node dengan hubungan XORSPLIT, mendeteksi invisible task, dan terakhir mencocokkannya dengan beberapa node di dalam label aktivitas yang memiliki nama yang sama.

Tabel 3.12 Contoh *Non-free Choice*

Kondisi	Traces
Non free-choice	[A, B, D, E, G] [A, C, D, F, G]
Graph Process Model	

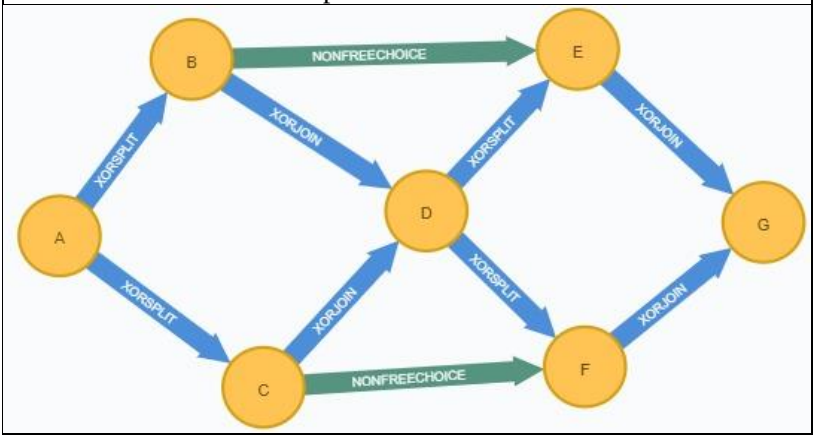


Table 3.13 Algoritma GIN dalam menemukan *Invisible Task in Non-free Choice*

Keterangan Tipe Relasi	**Rincian Keterangan**

Invisible Tasks in Non-free Choice	for i = 0 until *count_variants_activities* do
	n = nodeAsact[i]
	if not((n)-[:XORSPLIT]->() and ()-[:XORJOIN]->(n)):
	continue to loop
	endif
	path = 0
	for a,b in (a)-[xorjoin]->(n)-[xorsplit]->(b):
	if b == INVISIBLE TASK:
	b = (b)-->(x)
	return x

	endif
	for j = 0 until *count_event-1* do
	if (a:case[j])--->(n:case[j])-->(x:
	case[j]) and x != b:
	path++
	if path > 0 then
	break
	endif
	endif
	if path == 1:
	(a)-[:NON_FREE_CHOICE]->(b)
	endif
	endfor
	endfor
	endfor

Luaran	**Relasi Non Free Choice**

Dimana:

n : node aktivitas pada *graph database*

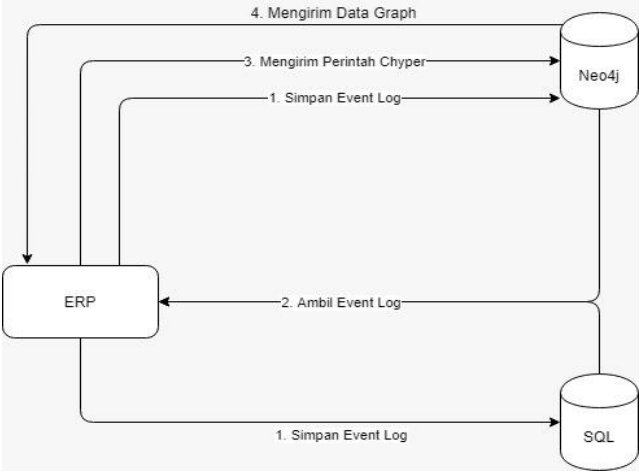
a, b : node aktivitas lain pada *graph database*

path : semua lintasan dari a ke b dan melewati node aktivitas n pada *case* ke j

BAB IV
ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tentang kueri Cypher yang akan digunakan untuk pemodelan proses bisnis menggunakan Neo4J.

4.1 Deskripsi Umum Sistem



Gambar 4.1 Arsitektur Sistem

Gambar 4.1 merupakan arsitektur sistem yang diajukan, dari gambar tersebut dapat kita lihat aplikasi ERP merupakan pusat pengolahan data *event log*. *Event log* yang dicatat diteruskan ke Neo4j dan SQL untuk disimpan dan diolah nantinya untuk membentuk model dan melakukan *process discovery* . *Process discovery* dilakukan menggunakan integrasi ERP dan Neo4j. *Process discovery* yang diajukan menggunakan algoritma GIN dengan alur pada Gambar 3.3.

4.2 Kebutuhan Fungsional Sistem

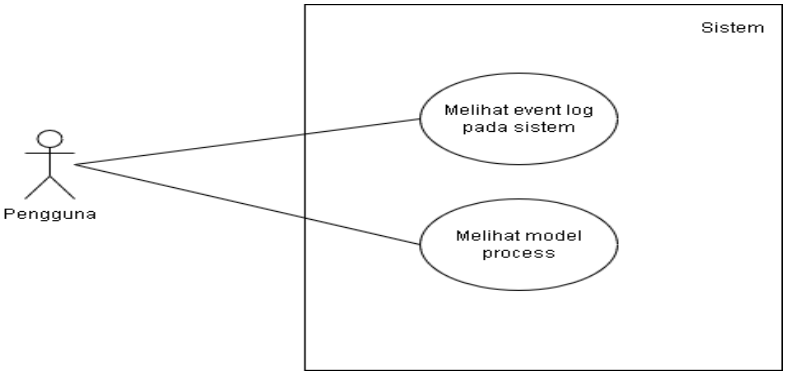
Kebutuhan fungsional berisi kebutuhan utama yang harus dipenuhi oleh sistem agar dapat bekerja dengan baik. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan oleh sistem, bagaimana reaksi terhadap masukan, dan hal-hal yang harus dilakukan sistem pada situasi khusus. Daftar kebutuhan fungsional dapat dilihat pada Tabel 4.1.

Tabel 4.1 Daftar Kebutuhan Fungsional Perangkat Lunak

Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
F-01	Melihat *event log* pada sistem	Pengguna dapat melihat data *event log* yang tercatat secara otomatis pada sistem
F-02	Melihat model proses	Pengguna dapat melihat model proses yang dibuat secara otomatis oleh sistem

4.3 Kasus Pengunaan Sistem

Kasus penggunaan secara umum akan digambarkan oleh salah satu model UML, yaitu diagram kasus penggunaan. Rincian kasus penggunaan berisi spesifikasi kasus penggunaan, diagram aktivitas, dan diagram urutan untuk masing-masing kasus penggunaan. Diagram kasus penggunaan dapat dilihat pada Gambar 4.2.



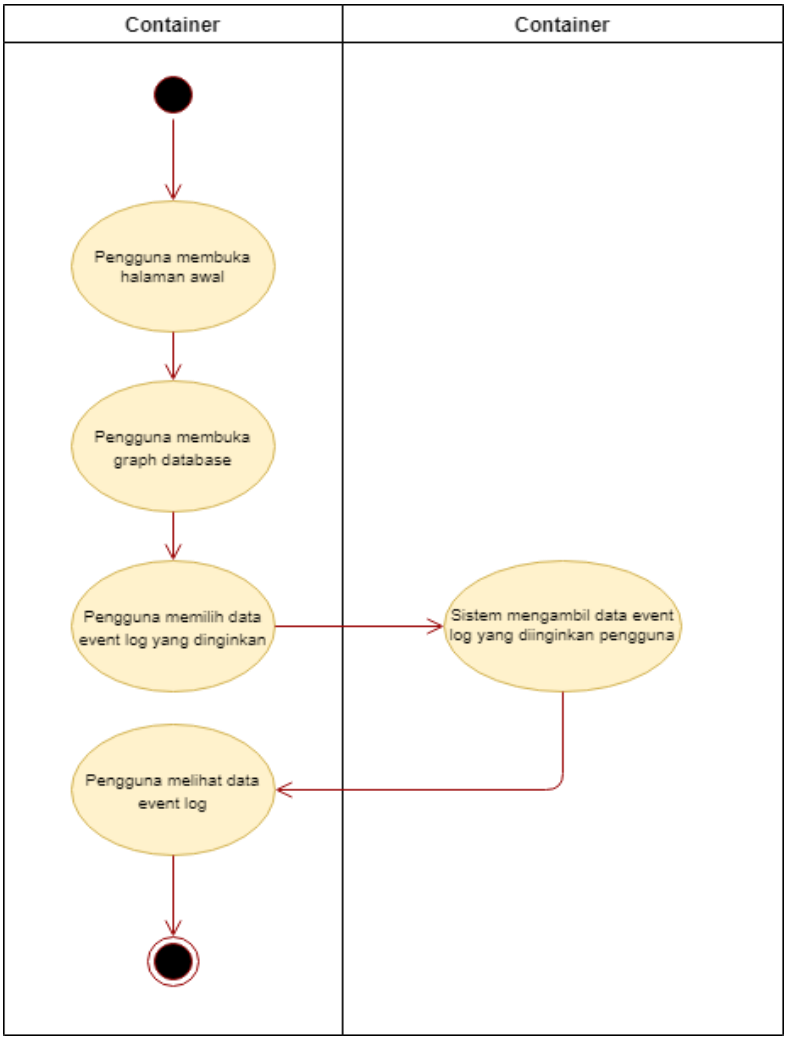
Gambar 4.2 Diagram Kasus Penggunaan

4.3.1 Deskripsi Kasus Pengunaan F-01

Kasus Pengunaan F-01 merupakan kasus dimana pengguna ingin melihat *event log* yang dicatat oleh sistem. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.2 dan diagram aktivitas dari kasus pengunaan ini bisa dilihat pada Gambar 3.4.

Tabel 4.2 Spesifikasi Kasus Penggunaan F-01

Nama	Melihat *event log* pada sistem
Kode	F-01
Deskripsi	Pengguna Melihat data *event log* pada sistem
Aktor	Pengguna
Kondisi Awal	Pengguna melihat halaman awal
Kondisi Akhir	Pengguna melihat data *event log*
Aliran	
Kejadian Normal	1. Pengguna membuka halaman awal. 2. Pengguna membuka *graph database*. 3. Pengguna memilih data *event log* yang diinginkan 4. Sistem mengambil data *event log* yang diinginkan pengguna 5. Pengguna melihat data *event log*



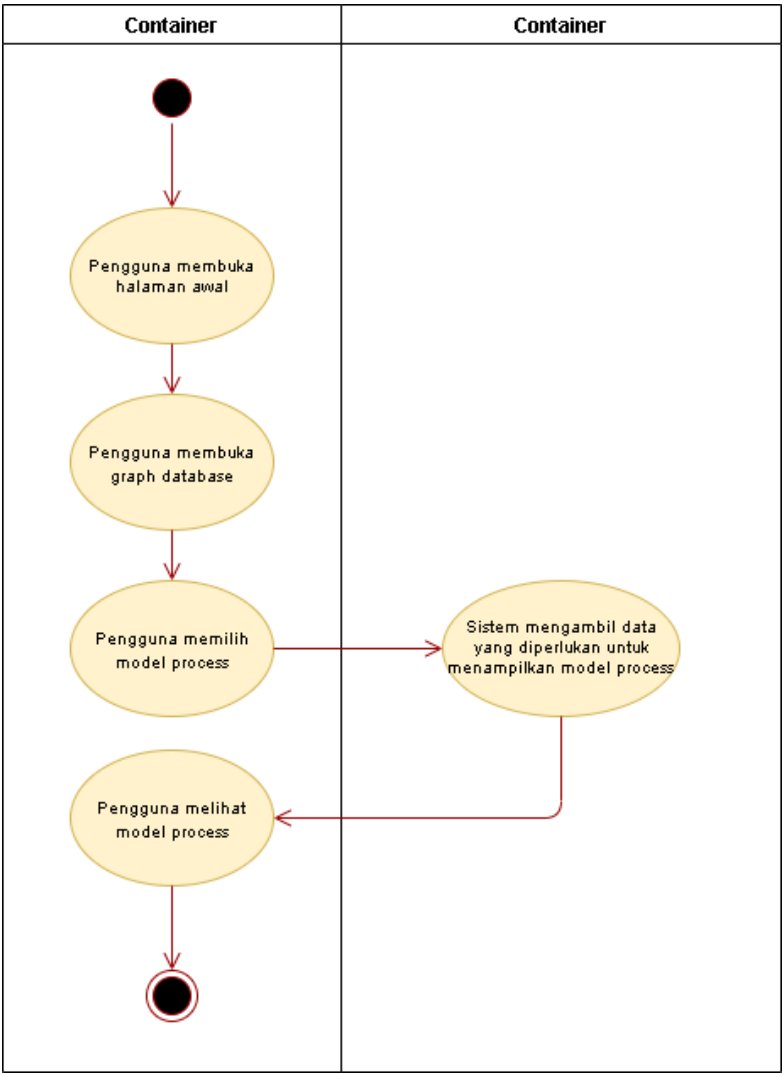
Gambar 4.3 Diagram Aktivitas Kasus Pengunaan F-01

4.3.2 Deskripsi Kasus Pengunaan F-02

Kasus Pengunaan F-02 merupakan kasus dimana pengguna ingin melihat model proses yang dibuat secara otomatis oleh sistem. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.3 dan diagram aktivitas dari kasus pengunaan ini bisa dilihat pada Gambar 3.5.

Tabel 4.3 Spesifikasi Kasus Penggunaan F-02

Nama	Melihat model proses
Kode	F-02
Deskripsi	Pengguna Melihat model proses pada sistem
Aktor	Pengguna
Kondisi Awal	Pengguna melihat halaman awal
Kondisi Akhir	Pengguna melihat model proses
Aliran	
Kejadian Normal	1. Pengguna membuka halaman awal. 2. Pengguna membuka *graph database*. 3. Pengguna memilih model proses 4. Sistem mengambil data yang diperlukan untuk menampilkan model proses 5. Pengguna melihat model proses



Gambar 4.4 Diagram Aktivitas Kasus Pengunaan F-02

4.4 Perancangan Sistem

Perancangan Sistem Integrasi ini didasarkan pada kebutuhan fungsional dan kasus pengunaan yang telah dijabarkan pada sub bab 4.2 dan 4.3. Integrasi ERP dengan Neo4j membuat pengguna cukup menggunakan antarmuka yang sudah disediakan oleh Neo4j.

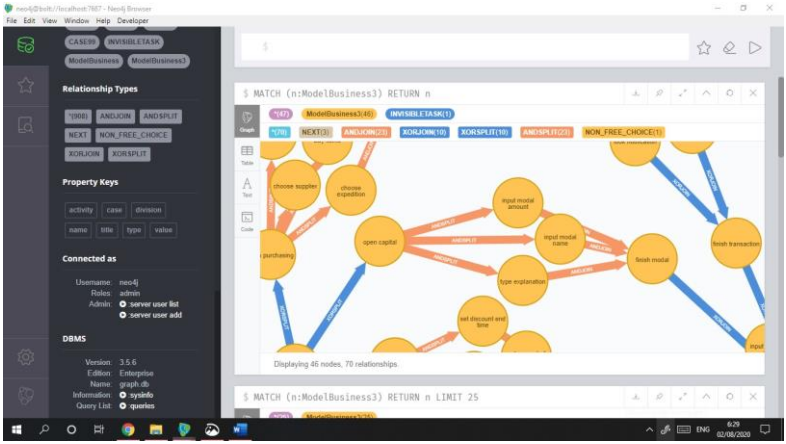
4.4.1 Antarmuka Melihat *Event Log*

Antarmuka ini digunakan untuk memenuhi kebutuhan pengunaan F-01. Pada antarmuka ini, pengguna akan memilih *case* pada *event log* dengan menekan salah satu *Node Labels* dengan awalan CASE pada panel *Database Information* dan sistem akan mengambil data *event log* yang dinginkan pengguna. Relasi antar aktivitas yang terbentuk pada setiap *case* didapatkan dari serangkaian aktivitas yang dilakukan beberapa pengguna dalam satu perusahaan pada saat menggunakan ERP. Gambar antarmuka dapat dilihat pada Gambar 4.5.

Gambar 4.5 Antarmuka melihat *Event Log*

4.4.2 Antarmuka Melihat Model Proses

Antarmuka ini digunakan untuk memenuhi kebutuhan pengunaan F-02. Pada antarmuka ini, pengguna akan memilih *process model* pada *event log* dengan menekan salah satu *Node Labels* dengan awalan ModelBusiness dan diakhiri dengan ID perusahaan ERP pada panel *Database Information*. Sistem akan mengambil data yang diperlukan untuk menampilkan *process model* yang dinginkan pengguna. Relasi antar aktivitas yang Terbentuk pada *process model* didapatkan dari kumpulan *trace* pada *event log* yang diolah menggunakan algoritma GIN. Gambar antarmuka dapat dilihat pada Gambar 4.6.



Gambar 4.6 Antarmuka melihat Model Proses

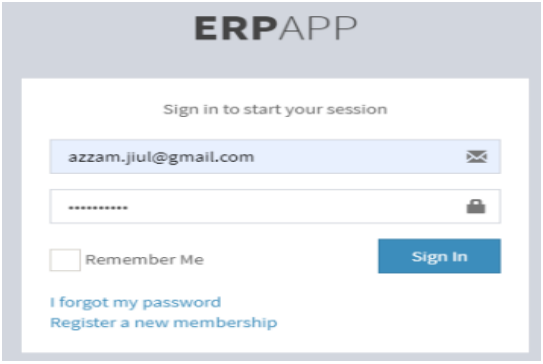
BAB V
IMPLEMENTASI

Bab ini membahas tentang kueri Cypher yang akan digunakan untuk pemodelan proses bisnis menggunakan Neo4J.

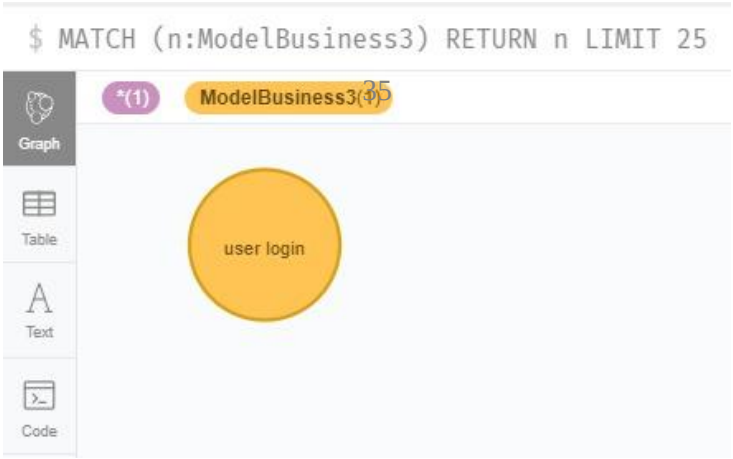
5.1 Proses Pencatatan *Log* dalam *Graph Database*

5.1.1 *Login*

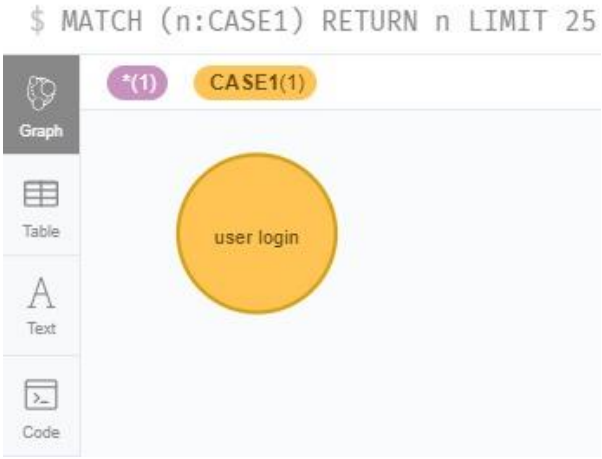
Pencatatan log dalam *graph database* dimulai dari ketika pengguna melakukan login pada sistem ERP. Gambar 5.1 menunjukkan *user interface* ERP saat melakukan *login.* Sedangkan Gambar 5.2 dan Gambar 5.3 menunjukkan pencatatan aktivitas *login* dalam *graph database* sebagai *case* dan model bisnis. *Case* digunakan untuk menyimpan model per *case.* Sedangkan model bisnis digunakan untuk menyimpan model secara utuh. Pencatatan tersebut dilakukan menggunakan algoritma pada Tabel 3.2. Pertama, sistem akan melakukan pengecekan *log* yang telah tersimpan dalam Neo4J. Jika ditemukan *log* terakhir pada *graph database*, maka aktivitas *login* tersebut dimasukkan dalam *case* terakhir + 1. Namun, jika *log* terakhir tidak ditemukan, maka aktivitas *login* tersebut dicatat sebagai *case* pertama. Adapun pengoneksian Neo4J dilakukan dengan menggunakan *library graphware* dengan menghubungkan ClientBuilder dari *graphware* tersebut. Pada saat itu juga, *case* yang sekarang akan dimasukkan ke dalam *session* untuk digunakan di tahap-tahap selanjutnya.



Gambar 5.1 *User Interface Login*



Gambar 5.2 Pencatatan Aktivitas *Login* sebagai *Case*

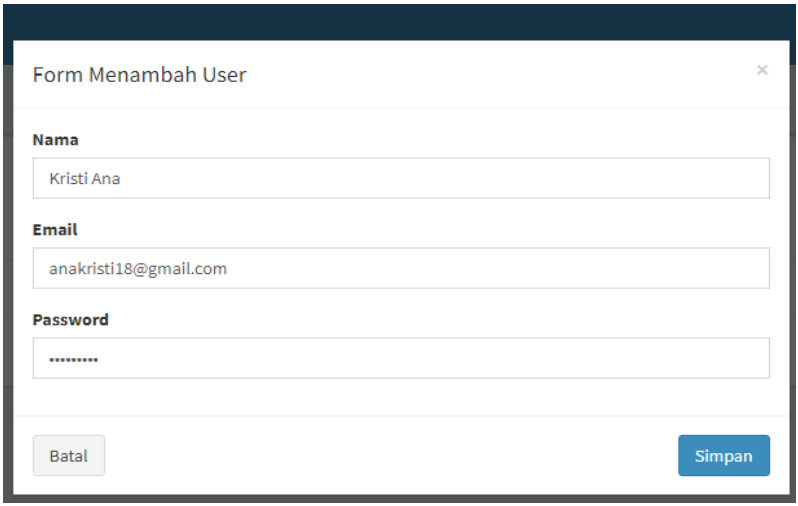


Gambar 5.3 Pencatatan Aktivitas *Login* sebagai Model Bisnis

Case digunakan untuk menyimpan model per *case.* Sedangkan model bisnis digunakan untuk menyimpan model secara utuh. Pencatatan tersebut dilakukan menggunakan algoritma pada Tabel 3.2.

5.2.2 Penambahan *User*

Setelah aktivitas *login,* aktivitas selanjutnya yang dicatat adalah tambah *user.* Aktivitas penambahan *user* dicatat sejak pengguna menekan tombol "Tambah *User*" dengan nama aktivitas "Open User Form" dengan nilai *TRUE* dan *timestamp* sesuai dengan saat pengguna menekan tombol "Tambah *User"*. Saat *user* memasukkan nama, *email*, dan *password*, *event log* akan dikirimkan ke *route* yang mengarah ke *controller* menggunakan *event on submit.* Adapun *user interface* penambahan *user* terlihat pada Gambar 5.4 dan algoritma untuk penambahan *user* terlihat pada Tabel 5.1.



Gambar 5.4 *Interface* Penambahan *User*

Tabel 5.1 Algoritma Pencatatan *Log* Penambahan *User*

Pencatatan ketika menekan Tombol Tambah Pengguna

let event_log = {}
$(document).on('click', '#btn-add-user', function ()
{
event_log = {}
let timestamp = getTimeNow();
event_log.open_user_form = {
activity:'open user form',
value: 'TRUE',
timestamp: timestamp
}
})

Pencatatan ketika mengisi Input Nama
$(document).on('input', '#user-name', function () { let value=this.value

```
    let timestamp = getTimeNow();
    event_log.user_name = {
     activity:'set user name',
     value: value,
     timestamp: timestamp
      }
     })
```

Pencatatan ketika mengisi Input Email

```
$(document).on('input', '#user-email', function () {
  let value=this.value
  let timestamp = getTimeNow()
  event_log.user_email = {
    activity:'set user email',
    value: value,
    timestamp: timestamp
  }})
```

Pencatatan ketika mengisi Input Password

```
$(document).on('input', '#user-password', function ()
{
let value=this.value
let timestamp = getTimeNow()

event_log.user_password = {
     activity:'set user password',
     value: value,
     timestamp: timestamp
}
})
```

Pencatatan ketika menekan tombol Simpan

```
$(document).on('submit', '#form-add-user', function()
{
    let timestamp = getTimeNow();
    event_log.close_user_form = {
```

```
    activity:'close user form',
    value: 'TRUE',
    timestamp: timestamp
    }

    appendToForm(event_log)
})
```

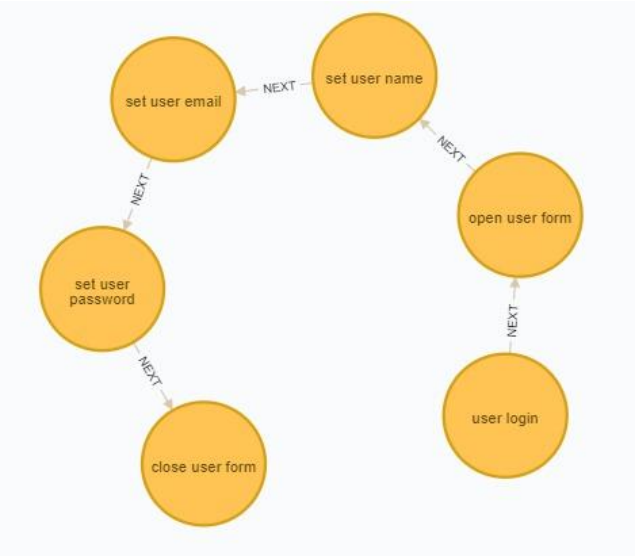
5.1.3 Pengambilan Data dari Sisi *Controller*

Pada sisi *controller, case* dari *session, user id, company id,* dan aktivitas terakhir pada *database* akan diambil. Aktivitas yang sudah dilakukan sebelumnya (seperti tambah *user*) akan dimasukkan ke dalam *database sql.* Selanjutnya, perulangan akan dilakukan untuk memasukkan serangkaian aktivitas ke dalam Neo4J (*case* dan model bisnis). Algoritma tersebut ditunjukkan oleh Tabel 5.2. Adapun hasil *log* yang tercatat pada *database* Neo4J ditunjukkan oleh Gambar 5.5.

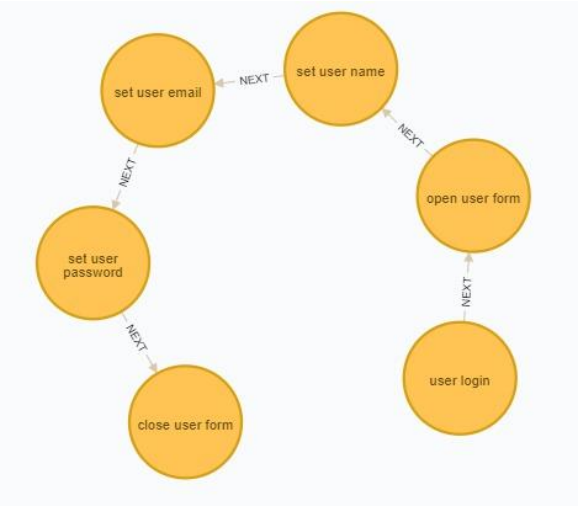
Tabel 5.2 Algoritma Pengambilan Data dari Sisi *Controller*

```
Input:eventLogs,caseId,userId,companyId,lastActivity
foreach eventLogs as eventLog:

   insertIntoSQL(eventLog)

   MERGE(n:caseId{activity:'eventLog-
>activity',value: 'eventLog->value' })

   Let          ModelBusinessWithCompanyId          =
nodeslabel(companyId)

   MERGE(n:ModelBusinessWithCompanyId{activity:
'eventLog->activity',value: 'eventLog->value'})
  MATCH(a:caseId),(b:caseId)
  WHERE a.activity <> b.activity AND a.activity =
  lastActivity->activity AND b.activity = eventLogs-
  >activityMERGE (a)-[r:NEXT]->(b)
```

```
  MATCH(a:ModelBusinessWithCompanyId),(b:ModelBusines
  sWithCompanyId)WHERE a.activity <> b.activity AND
  a.activity = lastActivity->activity AND b.activity
  = eventLogs->activityMERGE (a)-[r:NEXT]->(b)

  lastActivity = activity

endforeach
```



(a) Pencatatan *log* per *case*



(b) Pencatatan *log* dalam Model Bisnis

Gambar 5.5 Hasil Pencatatan *Log* pada *database* Neo4J (a) Pencatatan *log* per *case* dan (b) Pencatatan *log* dalam Model Bisnis

5.1.4 *Logout*

Setelah *User* selesai menggunakan aplikasi ERP dan menekan tombol logout, maka akan dilakukan pencatatan menggunakan algoritma Tabel 5.3 dengan cara:

1. Mengambil aktivitas terakhir pada database MySQL
2. Memasukan aktivitas ‘user logout’ kedalam database MySQL
3. Membuat node ‘user logout’ pada *case*.
4. Membuat node ‘user logout’ pada model.
5. Menyambungkan aktivitas terakhir sebelum logout

Dengan aktivitas ‘user logout’ menggunakan relasi NEXT. Adapun model hasil pencatatan aktivitas logout dapat dilihat pada **Error! Reference source not found.**. Dari gambar tersebut dapat dilihat bahwa aktivitas terakhir pada database MySQL yaitu aktivitas ‘close user form’ memiliki relasi NEXT ke aktivitas ‘user logout’.ketika *user* akan melakukan login lagi di aplikasi

ERP maka *user* akan masuk pada *case* yang baru seperti yang dijelaskan sebelumnya.

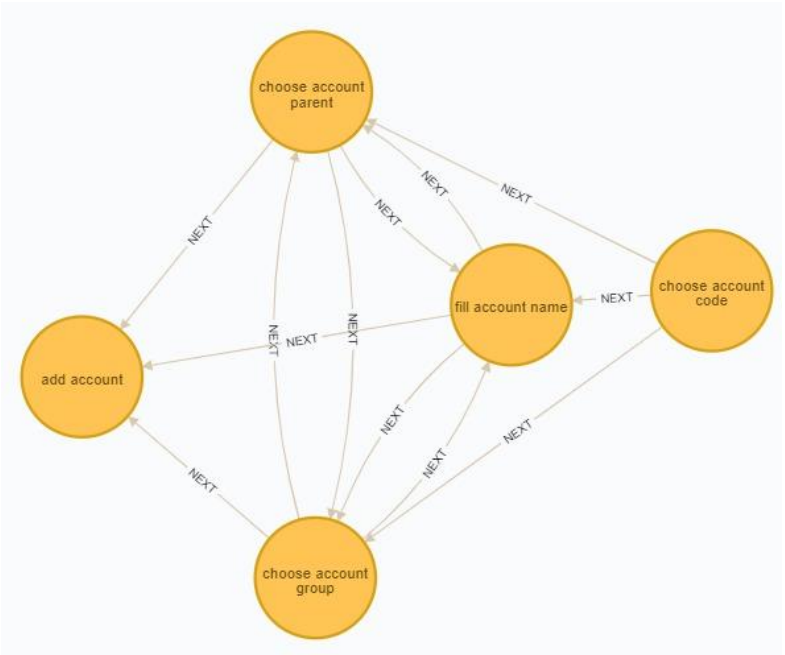
Tabel 5.3 Algoritma Pencatatan Aktivitas Logout

Input	userData,lastActivity,user LogoutActivity,caseId
Insert IntoSQL (userLogout) MERGE (n:caseId { activity : 'user logout', value: 'END' }) MATCH(a:caseId),(b:caseId) WHERE a.activity <> b.activity AND a.activity = lastActivity->activity AND b.activity = 'user logout'MERGE (a)-[r:NEXT]->(b) MATCH(a:ModelBusinessWithCompanyId),(b:ModelBusinessWithCompanyId)WHERE a.activity <> b.activity AND a.activity = lastActivity->activity AND b.activity = 'user logout'MERGE (a)-[r:NEXT]->(b)	
Output	create and end one case

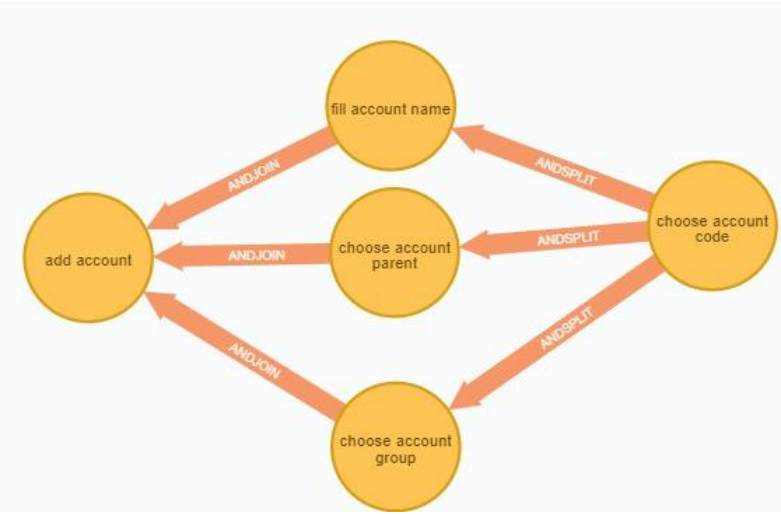
5.2 Proses Pembuatan Model

Dalam pembuatan model, indeks dibuat pada model bisnis menggunakan aktivitas yang terkandung di dalamnya. Setelah itu, XORSPLIT dan XORJOIN dicari terlebih dahulu, lalu mencari relasi ANDSPLIT, ORSPLIT, ANDJOIN, dan ORJOIN. Relasi ANDSPLIT dan ANDJOIN dibentuk jika banyaknya lintasan yang ada dalam relasi tersebut sama dengan banyak *node* dari SPLIT ke JOIN. Sebaliknya, ORSPLIT dan ORJOIN dibentuk ketika banyaknya lintasan yang ada dalam relasi tersebut lebih sedikit dari banyaknya *node* dari SPLIT ke JOIN. Kemudian, *invisible task* untuk relasi yang mengandung SPLIT dan JOIN. Jika satu relasi memiliki SPLIT dan JOIN secara bersamaan, maka *invisible task* disisipkan. Adapun algoritma detail untuk proses pembuatan model dapat dilihat pada Lampiran A.

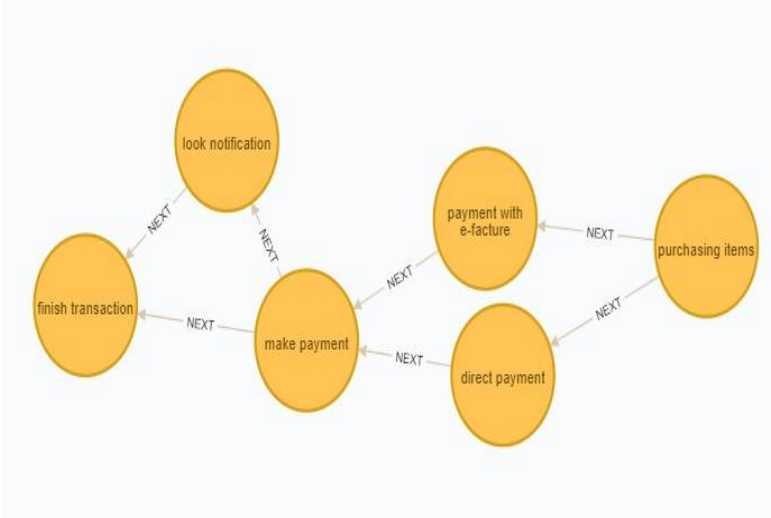
Sedangkan hasil model graf yang terbentuk dapat dilihat pada Gambar dibawah.



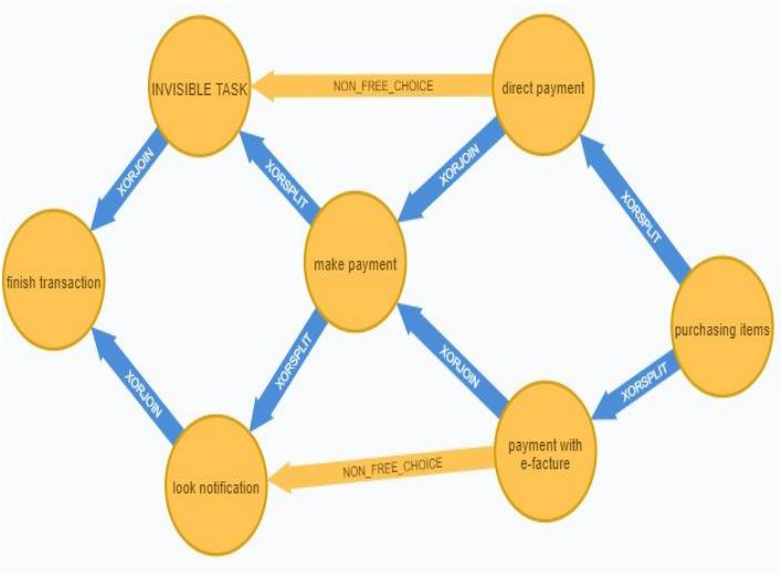
Gambar 5.6 Proces Pengisian Form Tambah Akun



Gambar 5.7 Proces Pengisian Form Tambah Akun(Penemuan Relasi AND)



Gambar 5.8 Proces Pengecekan Jenis Pembayaran Barang



Gambar 5.9 Proces Pengecekan Jenis Pembayaran Barang(Penemuan Relasi, XOR, *Invisible Task*, dan *non-free choice*)

Gambar 5.6 sampai Gambar 5.9 merupakan bagian dari model proses. Aktivitas ditandai dengan *node* berwarna orange,Model proses yang dibentuk memiliki relasi AND, XOR, *non-free choice*. Relasi NEXT dilambangkan sebagai panah hitam, relasi AND dilambangkan sebagai panah tebal merah, relasi XOR dilambangkan sebagai panah tebal biru, dan *non-free choice* dilambangkan sebagai panah tebal orange. Gambar 5.6 dan Gambar 5.8 merupakan model proses yang belum melalui tahap *process discovery* untuk menemukan semua relasi paralel sementara Gambar 5.7 dan Gambar 5.9 merupakan model proses setelah melalui tahap *process discovery* untuk menemukan semua relasi paralel. Pada Gambar 5.7 terbentuk relasi AND SPLIT dan AND JOIN , hal ini menunjukan bahwa setiap kali pengguna melakukan aktivitas ‘choose account code’ maka pengguna selalu melakukan aktivitas setelahnya yaitu ‘fiil acount name’, ’choose

account', dan 'choose account parent' sampai aktivitas 'add account'. Sementara pada Gambar 5.9 terbentuk satu aktivitas *invisible task* yaitu 'INVISIBLE TASK' serta relasi XOR SPLIT, XOR JOIN,dan *non-free choice*, relasi XOR SPLIT yang terbentuk menunjukan bahwa setiap kali pengguna melakukan aktivitas 'purchasing items' maka pengguna hanya bisa melakukan aktivitas 'direct payment' atau 'payment with e-facture' dan tidak melakukan keduanya pada proses yang sama. Relasi XOR JOIN yang terbentuk menunjukan bahwa setiap kali pengguna selesai melakukan aktivitas 'look notification' satu-satunya kegiatan yang dilakukan selanjutnya adalah 'finish transaction'. Relasi *non-free choice* yang terbentuk menunjukan bahwa setelah pengguna melakukan aktivitas 'payment with e-facture' aktivitas yang sudah pasti dilakukan pengguna setelah aktivitas 'make payment' adalah aktivitas 'look notification'.Relasi *non-free choice* yang terbentuk dari Algoritma GIN dapat berelasi dengan *invisible task*. Dengan demikian, algoritma ini dapat mendeteksi invisible task di dalam non-free choice (ditunjukkan pada Gambar 5.9) menggunakan *graph database*. Kasus *invisible tasks in non-free choice* ini dapat ditemukan menggunakan algoritma GIN dengan cara mencari rangkaian aktivitas yang selalu sama untuk setiap *trace*. Jika terdapat *invisible task* pada rangkaian aktivitas di model, maka aktivitas *invisible task* dapat digantikan dengan aktivitas setelah *invisible task*. Jika rangkaian aktivitas tersebut selalu sama untuk setiap *trace*, maka buat relasi *non-free choice* dari aktivitas awal ke *invisible task*.

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini membahas tentang uji dan evaluasi dari metode yang diusulkan.

6.1 Lingkungan Uji Coba

Uji coba sistem pada pengerjaan tugas akhir ini dilakukan pada lingkungan dan kakas bantu sebagai berikut :

- Prosesor Ryzen 2700U
- *RAM 8GB*
- Jenis Laptop
- Sistem Operasi Windows 10

6.2 Data Studi Kasus

Data Studi Kasus yang dilakukan pada tahapan ini menggunakan 1078 aktivitas (120 *case* dan 86 *trace*) dimana beberapa rangkaian aktivitas dapat membentuk *invisible task in non-free choice*.

6.3 Pengujian Fungsionalitas

Pengujian pada sistem ini mengacu pada pengujian Blackbox untuk menguji apakah fungsionalitas sistem telah berjalan sebagaimana mestinya. Pengujian mengacu pada setiap fitur yang telah diimplementasikan. Berdasarkan perancangan antarmuka di bab IV, maka pengujian fitur dibagi menjadi dua yaitu fitur awal melihat *event log* dan fitur akhir melihat model proses.

6.3.1 Pengujian Fitur Melihat *Event Log*

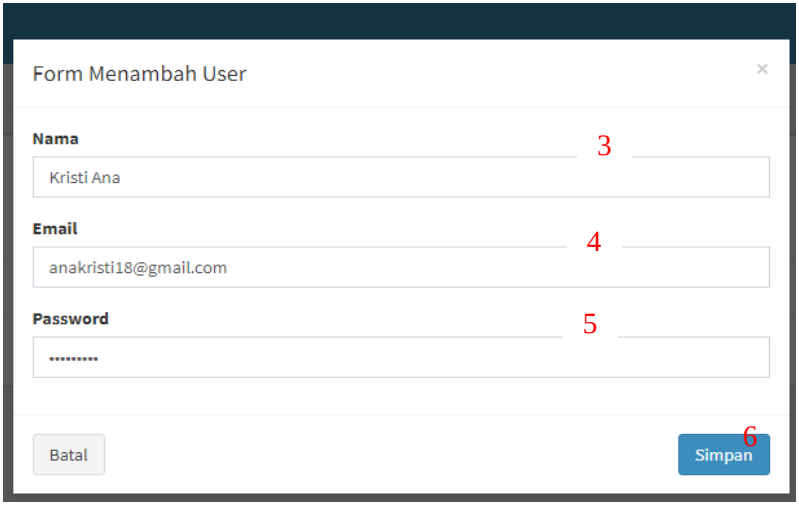
Pengujian fitur ini dilakukan dengan menggunakan salah satu modul pada aplikasi ERP. Rincian pengujian fitur ini dapat

dilihat pada Tabel 6.1. Sedangkan tampilan antarmuka yang menunjukkan keberhasilan pengujian dilihat pada Gambar 6.1 dan Gambar 6.2.

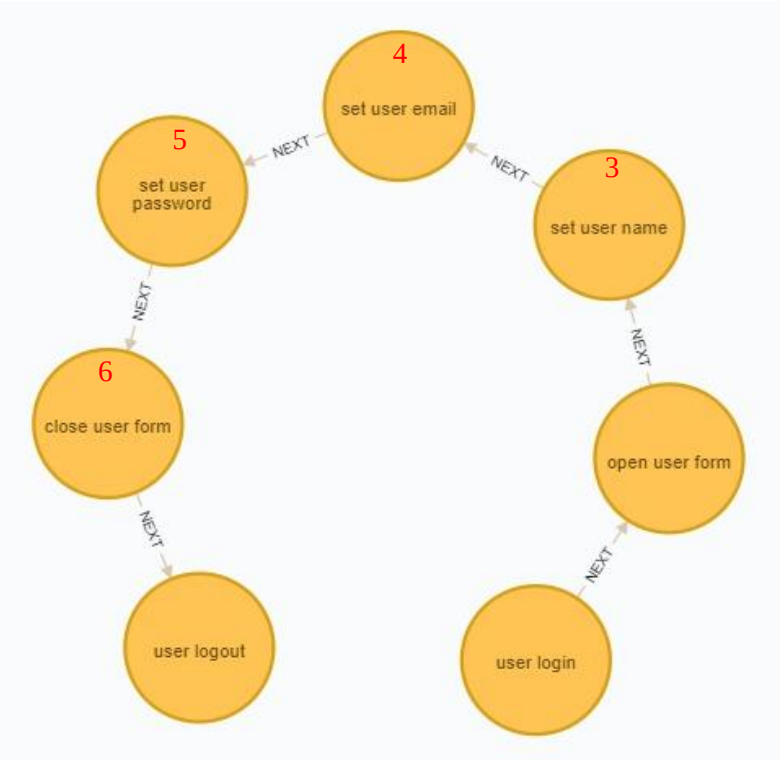
Tabel 6.1 Pengujian Fitur Melihat *Event Log*

Nama	Pengujian Melihat *Event Log*
Kode	UF-01
Referensi Kasus Penggunaan	F-01
Tujuan Pengujian	Menguji fitur melihat *event log*
Skenario	Pengguna ingin melihat *event log* terbaru dan sistem menyediakan *event log* tersebut.
Kondisi Awal	Sistem menampilkan halaman awal.
Data Uji	Data uji menggunakan masukan dari form tambah pengguna
Langkah Pengujian	1. Pengguna masuk ke aplikasi ERP 2. Pengguna membuka halaman pengguna 3. Pengguna menambahkan pengguna baru 4. Pengguna Logout dari aplikasi ERP 5. Pengguna melihat *graph database*
Hasil Yang Diharapkan	Sistem mampu memasukan rangkaian kegiatan tambah pengguna di *graph database*

Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan *event log* yang diinginkan pengguna



Gambar 6.1 Form Menambah *User*



Gambar 6.2 *Event Log* pada *graph database*

Gambar 6.1 dan Gambar 6.2 menunjukkan bahwa sistem berhasil dalam pengujian F-01, terbukti dengan apa yang dilakukan pengguna pada Gambar 6.1 dengan urutan aktivitas ditunjukan dengan angka berwarna merah sesuai dengan urutan rangkaian aktivitas dengan angka berwarna merah di dalam *graph database* yang ditunjukan Gambar 6.2. Maksud dari penomoran tersebut dapat kita contohkan sebagai berikut, pada Gambar 6.2 dimana aktivitas set user name berada pada urutan ketiga dari rangkaian aktivitas pada *graph database*. Hal ini ditandai dengan angka 3 berwarna merah, sama dengan urutan rangkaian aktivitas yang dilakukan pengguna pada saat mengisi form tambah *user* dengan urutan 3 ditandai dengan angka 3 berwarna merah pada

Gambar 6.1.

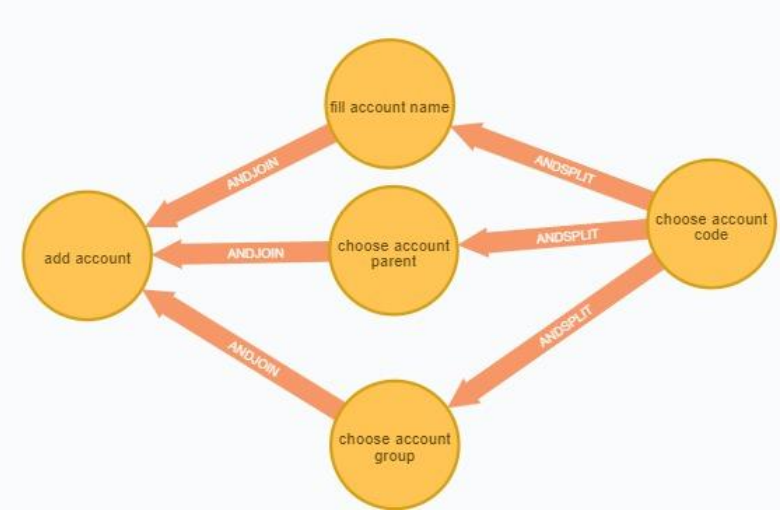
6.3.2 Pengujian Fitur Melihat Model Proses

Pengujian fitur ini dilakukan dengan melihat model proses yang terbentuk pada *graph database*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.2. Sedangkan tampilan antarmuka yang menunjukkan keberhasilan pengujian dapat dilihat pada Gambar 6.3.

Tabel 6.2 Pengujian Fitur Melihat Model Proses

Nama	Pengujian Melihat Model Proses
Kode	UF-02
Referensi Kasus Penggunaan	F-02
Tujuan Pengujian	Menguji fitur melihat model proses
Skenario	Pengguna ingin melihat model proses yang dibentuk sistem.
Kondisi Awal	Sistem menampilkan halaman awal.
Data Uji	Data yang berisi semua proses yang bisa dilakukan pengguna didalam aplikasi ERP Ketika menambah akun
Langkah Pengujian	1. Pengguna membuka *graph database* 2. Pengguna memilih model proses 3. Pengguna melihat model proses
Hasil Yang Diharapkan	Sistem mampu menampilkan

	model proses yang tepat
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan model proses



Gambar 6.3 Model Proses pada *Graph Database*

6.4 Pengujian Hasil dan Algoritma

Kualitas *process discovery* dapat diuji melalui 4 ha lyaitu fitness, presisi, generalisasi dan kompleksitas [24]. Hasil dari metode yang diusulkan diuji dengan membandingkan kualitas *process discovery* algoritma GIN berdasarkan nilai fitness, presisi, dan kompleksitas algoritma dengan hasil yang diperoleh oleh algoritma sebelumnya dalam menangani *invisible task* dan *non-free choice*, masing-masing Algoritma Alpha$ dan Alpha++. Nilai fitness diperoleh dengan menghitung jumlah *case* yang digambarkan dalam model dibagi dengan total *case* dalam *event log*. Sementara nilai presisi diperoleh dengan menghitung jumlah *trace* yang digambarkan dalam model dibagi dengan jumlah *trace*

yang ada dalam *event log*.

Dalam membentuk model proses, algoritma Alpha ++ memperoleh hasil fitness dan presisi masing-masing 0,98 dan 0,92. Hal ini dikarenakan algoritma Alpha++ hanya dapat menemukan *non-free choice* tanpa *invisible task* dalam model. Sementara itu, Algoritma Alpha$ dan Algoritma GIN memperoleh hasil fitness dan presisi 1. Tabel 6.3 menunjukan nilai *case* dan *trace* pada *event log* dan model proses untuk masing-masing algoritma *process discovery*.

Tabel 6.3 Nilai *Case* dan *Trace* setiap Algoritma

	Alpha+	Alpha#	GIN
Case in the model	131	133	133
Case in the event log	133	133	133
Trace in the model	93	101	101
Trace in the event log	101	101	101

Selain membandingkan hasil fitness dan presisi, kompleksitas algoritma dari masing-masing algoritma juga dibandingkan. Kompleksitas algoritma GIN dapat dihitung dimulai dari proses penyimpanan dan pembuatan relasi *sequence* antara aktivitas. Proses tersebut memiliki kompleksitas $O(n^2)$. Setelah relasi *sequence* terbentuk, tahap selanjutnya adalah membuat *invisible task* dengan tingkat kompleksitas $O(n^2)$ dan membuat relasi paralel dengan tingkat kompleksitas $O(n^3)$. Terakhir, penemuan relasi *non-free choice* dengan tingkat kompleksitas $O(n^3)$. Sehingga total kompleksitas yang dimiliki algoritma GIN adalah $O(n^3)$.

Perbedaan kompleksitas pada algoritma GIN dengan algoritma yang telah dijelaskan sebelumnya bisa menunjukan tingkat efisiensi yang dapat diukur menggunakan *Big-O notation* [21]. Nilai n yang ditunjukan merupakan aktivitas yang terdapat didalam *event log*. Jika menggunakan algoritma GIN sistem hanya perlu melakukan perulangan paling banyak n^3 sementara jika menggunakan algoritma sebelumnya yaitu Alpha$ sistem perlu melakukan perulangan sebanyak n^4.

Keuntungan dari *graph database* itu sendiri dibanding *tuples* adalah kemampuan untuk menyimpan serangkaian aktivitas beserta relasinya. Jadi Algoritma GIN bisa melakukan pengecekan pada relasi antar aktivitas untuk menemukan *invisible task* dalam *non-free choice*. Tugas Akhir ini juga membandingkan Algoritma GIN dan Algoritma Alpha$ berdasarkan kualitas masing-masing model. Pengukuran kualitas yang digunakan adalah fitness, precision, dan kompleksitas algoritma. Hasil dari percobaan menunjukan bahwa fitness dan precision dari algoritma Alpha$ dan algoritma GIN bernilai satu. Sementara, algoritma GIN mendapatkan tingkat kompleksitas yang lebih rendah $O(n^3)$ jika dibandingkan dengan algoritma Alpha++ dan Alpha$ yang memiliki kompleksitas $O(n^4)$ [25].

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

7.1 Kesimpulan

Dari hasil pengamatan selama proses perancangan dan implementasi, dapat diambil kesimpulan sebagai berikut :

1. *Invisible tasks in non-free choice* dapat ditemukan menggunakan algoritma GIN dengan cara mencari rangkaian aktivitas yang selalu sama untuk setiap *trace*. Jika terdapat *invisible task* pada rangkaian aktivitas di model, maka aktivitas *invisible task* dapat digantikan dengan aktivitas setelah *invisible task*. Jika rangkaian aktivitas tersebut selalu sama untuk setiap *trace*, maka buat relasi *non-free choice* dari aktivitas awal ke *invisible task.*
2. Integrasi ERP dengan Neo4j dapat dilakukan dengan menggunakan *library* yang disediakan oleh Neo4j bernama `GraphAware`. *Library* GraphAware merupakan *library official* Neo4j untuk sistem yang dibangun menggunakan bahasa pemrograman PHP, pada kasus ini sistem ERP yang dibangun menggunakan PHP sebagai Bahasa pemrograman-nya.
3. Tugas akhir ini membandingkan Algoritma GIN dengan Algoritma Alpha$ dan Alpha++ berdasarkan 3 hal yang menentukan kualitas *process discovery* yaitu fitness, presisi, dan kompleksitas algoritma. Eksperimen menunjukkan bahwa fitness dan presisi Algoritma Alpha++ adalah masing-masing 0,98 dan 0,92. Kemudian, fitness dan presisiAlgoritma Alpha$ dan Algoritma GIN adalah 1. Namun demikian, Algoritma GIN mendapatkan kompleksitas yang lebih baik dibandingkan dengan

Algoritma Alpha++ dan Alpha$. Kompleksitas Algoritma Alpha++ dan Alpha$ adalah $O(n^4)$, sedangkan kompleksitas Algoritma GIN adalah $O(n^3)$.

7.2 Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan sebagai berikut:

1. Fitur memasukkan *event log* lampau dapat dikembangkan dengan mengimplementasikan hadoop atau database NOSQL untuk menjawab permasalahan *bigdata*.
2. Sistem dapat dikembangkan untuk mendeteksi anomali proses secara langsung.

DAFTAR PUSTAKA

[1] C. S. Wahyuni, N. Y. Setiawan, and I. Aknuranda, "Pemodelan dan Evaluasi Proses Bisnis Berdasarkan Hasil Ekstraksi Event Log denganMenerapkan Process Mining pada Divisi Produksi PT. Kutai Timber Indonesia Kota Probolinggo," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 2, no. 9, pp. 3087–3094, 2018.

[2] W. M. P. Van Der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Heidelberg: Springer, 2011.

[3] W. M. P. Van Der Aalst, T. Weijters, and L. Maruster, "Workflow Mining : Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[4] R. Sarno, R. D. Dewandono, T. Ahmad, M. F. Naufal, and F. Sinaga, "Developing a workflow management system for enterprise resource planning," *IAENG International Journal of Computer Science*, vol. 72, no. 3, pp. 412–421, 2015.

[5] L.Wen, W.M.P Van Der Aalst, J. Wang,J. Sun," Mining process models with non-free-choice constructs," *Data Mining and Knowledge Discovery* , vol. 15, no.2, pp. 145–180, 2007. https://doi.org/10.1007/s10618-007-0065-y

[6] W. M. P. Van Der Aalst, A. J. M. M. Weijters, and L. Maruster, "Workflow Mining : Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[7] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining Invisible Tasks in Non-free-choice Constructs", in *Lecture Notes in Computer Science, Springer International Publishing*, pp. 109–125, 2016

[8] R. Sarno, K. R. Sungkono & R. Johanes & D. Sunaryono,

"Graph-Based Algorithms for Discovering a Process Model Containing Invisible Tasks," *International Journal of Intelligent Engineering and Systems*, vol. 5, no. 2, pp. 85-94, 2019.

[9] R. Sarno, K. R. Sungkono, and R. Septiarakhman, "Graph-Based Approach for Modeling and Matching Parallel Business Processes," *International Information Institute (Tokyo) Information*, vol. 21, no. 5, pp. 1603--1614, 2018.

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

[11] Y. A. Effendi and R. Sarno, "Discovering Process Model From Event Logs by Considering Overlapping Rules," *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Yogyakarta, 2017, pp. 1-6, doi: 10.1109/EECSI.2017.8239193.

[12] S. Huda, R. Sarno, and T. Ahmad, "Increasing accuracy of process-based fraud detection using a behavior model," *International Journal of Software Engineering and its Applications*, vol. 10, no. 5, pp. 175–188, 2016.

[13] D. Rahmawati, R. Sarno, C. Fatichah, and D. Sunaryono, "Fraud Detection on Event Log of Bank Financial Credit Business Process using Hidden Markov Model Algorithm," *3rd International Conference on Science in Information Technology (ICSITech)*, pp. 35–40, 2017.

[14] N. Russell, A. H. M. Hofstede, W. M. P. Van Der Aalst, and N. Mulyar, "Workflow Control-Flow Patterns: A Revised View," BPMcenter. Org, 2006.

[15] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks," in *4th Information Systems International Conference 2017 (ISICO 2017)*, 2018, vol. 124, pp. 134–141.

[16] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *International Review on Computers and Software*

(IRECOS), vol. 11, no. 4, p. 290, 2016.

[17] "The Neo4J Operations Manual v3.5," *https://neo4j.com/docs/operations-manual/3.5/introduction/*. [Online]. Available: https://neo4j.com/docs/operations-manual/3.5/introduction/. [Accessed: 06-Aug-2019].

[18] H. Darmawan, "Pemodelan Proses Bisnis Interlibrary Loan Studi Kasus Perpustakaan ITS," Institut Teknologi Sepuluh Nopember Surabaya, 2018.

[19] "The Neo4J Cypher Manual v3.5," *https://neo4j.com/docs/cypher-manual/current/introduction/#cypher-introduction*. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/introduction/#cypher-introduction. [Accessed: 06-Aug-2019].

[20] Y.A. Effendi and R. Sarno, "Moddeling Parallel Business Process Using Modified Time-Based Alpha Miner," *International Journal of Innovative Computing, Information and Control*, vol. 14, no. 5, pp. 1565-1582, 2018.

[21] Hermawan and R.Sarno, "A More Efficient Deterministic Algorithm In Process Model Discovery", *International Journal of Innovative Computing, Information and Control*, vol. 14, no. 3, pp 971-995, 2018.

[22] S. Gayathri Devi, K. Selvam and S. P. Rajagopalan, "An abstract to calculate big o factors of time and space complexity of machine code," *International Conference on Sustainable Energy and Intelligent Systems (SEISCON 2011)*, Chennai, 2011, pp. 844-847, doi: 10.1049/cp.2011.0483

[23] Mihaiu, Diana Marieta, Opreana, Alin, Cristescu, and Marian Pompiliu, "Efficiency, Effectiveness and Performance of the Public Sector," *Journal for Economic Forecasting, Institute for Economic Forecasting*, vol. 1, no. 4, pp. 132-147, 2010.

[24] J. C. A. M. Buijs, B. F. Van Dongen, and W. M. P Van Der Aalast, "Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity," *International Journal of Cooperative Information Systems*, vol. 23, no. 1, p. 1440001, 2014.

[25] R. Sarno and K. R. Sungkono, "A survey of graph-based algorithms for discovering business processes," *International Journal of Advances in Intelligent Informatics*, vol. 5, no. 2, pp. 137 - 149, 2019.

LAMPIRAN A

```
ini_set('max_execution_time', 12000);
    ini_set('memory_limit', '-1');
    $client = ClientBuilder::create()
    ->addConnection('GraphTest',
'bolt://neo4j:123@localhost:7687')
    ->build();
    $query_create_index_ModelBusiness = "CREATE INDEX
ON :ModelBusiness".strval(Auth::user()-
>company_id)."(activity)";
    $client->run($query_create_index_ModelBusiness);
    $query_check_xorSplit = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:NEXT]->(a:ModelBusiness".strval(Auth::user()-
>company_id).")
                WHERE size((n)-->()) > 1 and (size((a)<--
()) = 1)
                MERGE (n)-[x:XORSPLIT]->(a)
                DELETE r";
    $client->run($query_check_xorSplit);
    $query_check_xorJoin = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:NEXT]->(a:ModelBusiness".strval(Auth::user()-
>company_id).")
                WHERE size((n)-->()) = 1 and (size((a)<--
()) > 1 )
                MERGE (n)-[x:XORJOIN]->(a)
                DELETE r";
```

```
    $client->run($query_check_xorJoin);


    $query_check_joinNode = "MATCH (a:ModelBusiness".strval(Auth::user()->company_id).")-[r:NEXT]->(n:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE size((a)-->()) >= 2 AND size((n)<--()) >= 2 AND NOT EXISTS((n)-->(a))
    MERGE (a)-[x:SPLIT_OR_JOIN]->(n)
    DELETE r";
    $client->run($query_check_joinNode);


    $query_delete_nextMiddleNode = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[r:NEXT]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE EXISTS(()-[:SPLIT_OR_JOIN]->(n)) AND EXISTS(()-[:SPLIT_OR_JOIN]->(a))
    DELETE r";
    $records = $client->run($query_delete_nextMiddleNode);


    $query_check_invisibleTask = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[r:SPLIT_OR_JOIN]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE size((n)-->()) > 1 AND size((a)<--()) > 1
    MERGE (n)-[nr1:INVISIBLE]->(a)
```

```
    DELETE r";
    $client->run($query_check_invisibleTask);


    $query_check_splitNode = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:SPLIT_OR_JOIN]->(a:ModelBusiness".strval(Auth::user()-
>company_id).")
                WHERE size((n)-->()) > 1 AND NOT
EXISTS((a)-->(n))
                MERGE (n)-[x:SPLIT]->(a)
                DELETE r";
    $client->run($query_check_splitNode);


    $query_check_joinNode = "MATCH
(a:ModelBusiness".strval(Auth::user()->company_id).")-
[r:SPLIT_OR_JOIN]->(n:ModelBusiness".strval(Auth::user()-
>company_id).")
    WHERE size((n)<--()) > 1 AND NOT EXISTS((n)-->(a))
    MERGE (a)-[x:JOIN]->(n)
    DELETE r";
    $client->run($query_check_joinNode);



    $query_check_invisibleTask = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:JOIN|XORJOIN]->(a:ModelBusiness".strval(Auth::user()-
>company_id).")
```

```
    WHERE EXISTS((n)-[:XORSPLIT]->())
    MERGE (n)-[nr1:INVISIBLE]->(a)
    DELETE r";
    $client->run($query_check_invisibleTask);


    $query = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE (n)-[:SPLIT]->()
    return n";
    $result = $client->run($query);
    $splitNodes = array();



    foreach ($result->getRecords() as $record) {
      array_push($splitNodes, $record->get('n')->value('activity'));
    }
    $query = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE ()-[:JOIN]->(n)
    return n";
    $result = $client->run($query);
    $joinNodes = array();
```

```
    foreach ($result->getRecords() as $record) {
      array_push($joinNodes, $record->get('n')->value('activity'));
    }

    $noRelations = array();
    foreach ($splitNodes as $splitNode) {
      error_log("masuk  di splitnode: ".$splitNode);

      $query = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id)." {activity:'".$splitNode."'})-[:SPLIT]->(a:ModelBusiness".strval(Auth::user()->company_id).") RETURN a";
      $result = $client->run($query);
      $records = $result->getRecords();
      $nextNodes = array();
      foreach ($result->getRecords() as $record) {
        array_push($nextNodes, $record->get('a')->value('activity'));
      }
      $log_data = EventLog::where([['company_id', Auth::user()->company_id],['activity', $splitNode]])->first();
      $id =  $log_data->case;
      $log_data = EventLog::where([['company_id', Auth::user()->company_id],['case', $id]])-
```

```
>orderBy('created_at')->get();



        $counter = 0;
        $count = false;
        foreach($log_data as $data){
          if($count && !in_array($data->activity,$nextNodes)){
            break;
          }

          if(in_array($data->activity,$nextNodes)){
            $count = true;
            $counter+=1;
          }
        }

        if (!$count) {
          array_push($noRelations,[$splitNode, null]);
          continue;
        }

        if (count($nextNodes) == $counter) {
          error_log("buat relasi AND");
          $query_check_splitNode = "MATCH
```

```
(n:ModelBusiness".strval(Auth::user()->company_id)."{activity:'".$splitNode."'})-[r:SPLIT]->(a:ModelBusiness".strval(Auth::user()->company_id).")
                        MERGE (n)-[x:ANDSPLIT]->(a)
                        DELETE r";
          $client->run($query_check_splitNode);
        }elseif (count($nextNodes) > $counter) {
          error_log("buat relasi OR");
          $query_check_splitNode = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id)."{activity:'".$splitNode."'})-[r:SPLIT]->(a:ModelBusiness".strval(Auth::user()->company_id).")
                        MERGE (n)-[x:OR_SPLIT]->(a)
                        DELETE r";
          $client->run($query_check_splitNode);
        }
      }
      foreach ($joinNodes as $joinNode) {
        error_log("masuk  di joinNode: ".$joinNode);

        $query = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[:JOIN]->(a:ModelBusiness".strval(Auth::user()->company_id)." {activity:'".$joinNode."'}) RETURN n";
        $result = $client->run($query);
        $records = $result->getRecords();
        $prevNodes = array();
```

```
        foreach ($result->getRecords() as $record) {
          array_push($prevNodes, $record->get('n')->value('activity'));
        }
        $log_data = EventLog::where([['company_id', Auth::user()->company_id],['activity', $joinNode]])->first();
        $id =  $log_data->case;
        $log_data = EventLog::where([['company_id', Auth::user()->company_id],['case', $id]])->orderBy('created_at')->get();



        $counter = 0;
        $count = false;
        foreach($log_data as $data){
          if($count && !in_array($data->activity,$prevNodes)){
            break;
          }

          if(in_array($data->activity,$prevNodes)){
            $count = true;
            $counter+=1;
          }
        }
```

```
        if (!$count) {
            array_push($noRelations,[$joinNode, null]);
            continue;
        }

        if (count($prevNodes) == $counter) {
            error_log("buat relasi AND");
            $query_check_joinNode = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:JOIN]->(a:ModelBusiness".strval(Auth::user()-
>company_id)." {activity:'".$joinNode."'})
                        MERGE (n)-[x:ANDJOIN]->(a)
                        DELETE r";
            $client->run($query_check_joinNode);
        }elseif (count($prevNodes) > $counter) {
            error_log("buat relasi OR");
            $query_check_joinNode = "MATCH
(n:ModelBusiness".strval(Auth::user()->company_id).")-
[r:JOIN]->(a:ModelBusiness".strval(Auth::user()-
>company_id)." {activity:'".$joinNode."'})
                        MERGE (n)-[x:OR_JOIN]->(a)
                        DELETE r";
            $client->run($query_check_joinNode);
        }
    }
    error_log("masukan invisible node diantara AND
```

```
SPLIT");
    $query_check_invisibleTask = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[:ANDSPLIT]->(),(n:ModelBusiness".strval(Auth::user()->company_id).")-[r:INVISIBLE]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    CREATE (x:ModelBusiness".strval(Auth::user()->company_id)." {activity:'INVISIBLE TASK'})
    MERGE (n)-[nr1:ANDSPLIT]->(x)
    MERGE (x)-[nr2:JOIN]->(a)
    DELETE r";
    $client->run($query_check_invisibleTask);



    error_log("masukan invisible node diantara XOR SPLIT");
    $query_check_invisibleTask = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[:XORSPLIT]->(),(n:ModelBusiness".strval(Auth::user()->company_id).")-[r:INVISIBLE]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    CREATE (x:ModelBusiness".strval(Auth::user()->company_id)." {activity:'INVISIBLE TASK'})
    MERGE (n)-[nr1:XORSPLIT]->(x)
    MERGE (x)-[nr2:NEXT]->(a)
    DELETE r";
    $client->run($query_check_invisibleTask);
```

```
    $query_check_xorJoin = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[r:NEXT]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE size((n)-->()) = 1 and (size((a)<--()) > 1 )
    MERGE (n)-[x:XORJOIN]->(a)
    DELETE r";
    $client->run($query_check_xorJoin);


    error_log("masukan invisible node diantara SPLIT");
    $query_check_invisibleTask = "MATCH (n:ModelBusiness".strval(Auth::user()->company_id).")-[:SPLIT]->(),(n:ModelBusiness".strval(Auth::user()->company_id).")-[r:INVISIBLE]->(a:ModelBusiness".strval(Auth::user()->company_id).")
    CREATE (x:ModelBusiness".strval(Auth::user()->company_id)." {activity:'INVISIBLE TASK'})
    MERGE (n)-[nr1:SPLIT]->(x)
    MERGE (x)-[nr2:JOIN]->(a)
    DELETE r";
    $client->run($query_check_invisibleTask);


    error_log("masukan invisible node diantara JOIN");
    $query_check_invisibleTask = "MATCH ()-[:JOIN]->(n:ModelBusiness".strval(Auth::user()->company_id)."),(a:ModelBusiness".strval(Auth::user()->company_id).")-[r:INVISIBLE]-
```

```
>(n:ModelBusiness".strval(Auth::user()->company_id).")
    CREATE (x:ModelBusiness".strval(Auth::user()->company_id)." {activity:'INVISIBLE TASK'})
    MERGE (a)-[nr1:SPLIT]->(x)
    MERGE (x)-[nr2:JOIN]->(n)
    DELETE r";
    $client->run($query_check_invisibleTask);

    $query = "MATCH (middle:ModelBusiness".strval(Auth::user()->company_id).")
    WHERE EXISTS((middle)-[:XORSPLIT]->(:ModelBusiness".strval(Auth::user()->company_id).")) AND EXISTS((:ModelBusiness".strval(Auth::user()->company_id).")-[:XORJOIN]->(middle))
    RETURN middle";

    $middleNodeChyper = $client->run($query);
    $middleNode = array();
    foreach ($middleNodeChyper->getRecords() as $middleNode) {
      $middleNode = $middleNode->get('middle')->value('activity');

      $query = "MATCH
```

```
(n:ModelBusiness".strval(Auth::user()->company_id.")-
[:XORJOIN]->(middle:ModelBusiness".strval(Auth::user()-
>company_id).")
        WHERE middle.activity='$middleNode'
        RETURN n";


        $beforeMiddleNodeChyper = $client->run($query);


        $nodesBefore = array();
        foreach ($beforeMiddleNodeChyper->getRecords() as
$beforeMiddleNode){
          $beforeMiddleNode = $beforeMiddleNode-
>get('n')->value('activity');
          array_push($nodesBefore,$beforeMiddleNode);
        }


        $query = "MATCH
(middle:ModelBusiness".strval(Auth::user()-
>company_id).")-[:XORSPLIT]-
>(n:ModelBusiness".strval(Auth::user()->company_id).")
        WHERE middle.activity='$middleNode'
        RETURN n";


        $afterMiddleNodeChyper = $client->run($query);


        $nodesAfter = array();
        foreach ($afterMiddleNodeChyper->getRecords() as
```

```
$afterMiddleNode){
            $afterMiddleNode = $afterMiddleNode->get('n')-
>value('activity');
            array_push($nodesAfter,$afterMiddleNode);
        }
        foreach ($nodesBefore as $nodeBefore) {
            $path = array();
            foreach ($nodesAfter as $nodeAfter) {
              if ($nodeAfter == 'INVISIBLE TASK') {
                $query = "MATCH
(middle:ModelBusiness".strval(Auth::user()-
>company_id).")--
>(invisible:ModelBusiness".strval(Auth::user()-
>company_id).")--
>(after:ModelBusiness".strval(Auth::user()-
>company_id).")
                WHERE middle.activity = '$middleNode' AND
invisible.activity = '$nodeAfter'
                RETURN after";


                $exist = $client->run($query);
                $nodeAfter = $exist->getRecords()[0]-
>get('after')->value('activity');
                $arrayOfData =
EventLog::where([ ['company_id', Auth::user()-
>company_id],['activity', $nodeBefore] ])->get();
                foreach ($arrayOfData as $data) {
                  $query = "MATCH (before:CASE$data->case)-
```

```
->(after:CASE$data->case)
                WHERE before.activity = '$middleNode' AND
after.activity = '$nodeAfter'
                RETURN after";
                $exist = $client->run($query);
                if (count($exist->getRecords())) {
                  $data = 'INVISIBLE TASK';
                  if (in_array($data, $path)) {
                    continue;
                  }
                  array_push($path,'INVISIBLE TASK');
                }
              }
              continue;
            }


            $arrayOfData = EventLog::where([ ['company_id',
Auth::user()->company_id],['activity',  $nodeBefore]  ])-
>get();
            foreach ($arrayOfData as $data) {
              $query = "MATCH (before:CASE$data->case)--
>(middle:CASE$data->case)-->(after:CASE$data->case)
              WHERE before.activity = '$nodeBefore' AND
middle.activity = '$middleNode' AND after.activity =
'$nodeAfter'
              RETURN after";
```

```
                $exist = $client->run($query);
                if (count($exist->getRecords()) ) {
                  $data = $exist->getRecords()[0]->get('after')->value('activity');

                  if (in_array($data, $path)) {
                    continue;
                  }
                  array_push($path,$data);
                }
              }
            }

            if (count($path) == 1) {
              //echo "non free choice pada $nodeBefore -> $middleNode ->".json_encode($path[0])."<br><br>";
              $query_check_nonFreeChoice = "MATCH (a:ModelBusiness".strval(Auth::user()->company_id).",(b:ModelBusiness".strval(Auth::user()->company_id).")
              WHERE a.activity <> b.activity AND a.activity = '$nodeBefore' AND b.activity = '".$path[0]."'
              MERGE (a)-[r:NON_FREE_CHOICE]->(b)";
              $client->run($query_check_nonFreeChoice);
            }else{
              dd($path);
```

```
            }
          }



      }

      // dd(array($nodesBefore,$nodesAfter));
      foreach ($noRelations as &$noRelation) {
        $noRelation = implode(' & ', $noRelation);
      }

      $noRelations = implode(',', $noRelations);

      return                    view('dashboard.graph_display',
compact('noRelations'));
```

[Halaman ini sengaja dikosongkan]

DAFTAR ISTILAH

Event Log
: Rekaman data transaksi dari ERP.

Process Discovery
: Tahapan dalam *process mining* untuk menemukan model proses berdasarkan event log.

Process Mining
: Metode penambangan proses berdasarkan *event log*.

Non-free Choice
: Suatu pilihan tidak bebas yang bergantung dengan aktivitas sebelumnya.

Invisible Task
: Suatu aktivitas tidak terlihat yang dapat terjadi karena adanya aktivitas yang tidak dieksekusi dan / atau aktivitas berulang.

ProM
: Suatu alat yang digunakan untuk *process mining.*

Neo4J
: Suatu alat untuk mengeksekusi model proses berbasis graf.

Case
: Suatu ID yang menandakan berjalannya proses dari awal sampai akhir.

Event
: Suatu aktivitas dalam *event log.*

Timestamp
: Menandakan kapan aktivitas tersebut dilaksanakan.

Resource
: Menandakan siapa yang mengeksekusi aktivitas tersebut.

AND
: Suatu relasi yang meneksekusi seluruh opsi aktivitas yang diberikan.

OR
: Suatu relasi yang mengeksekusi lebih dari satu opsi aktivitas yang diberikan, namun tidak semua.

XOR
: Suatu relasi yang mengeksekusi hanya satu opsi aktivitas dari keseluruhan opsi yang diberikan.

Sequence
: Suatu relasi yang mengeksekusi aktivitas secara berurutan.

Control-flow pattern
: Menunjukkan beberapa tipe aliran kontrol aktivitas, yaitu paralel dan sekuensial.

Node
: Gambaran aktivitas pada model graf.

DAFTAR INDEKS

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Nama	:	Muhammad Taufiqulsa`di
Tempat, tanggal lahir	:	Palu, 25 Oktober 1998
Jenis Kelamin	:	Laki - laki
Agama	:	Islam
Status	:	Belum Menikah

Alamat Asal	:	Perumahan Dosen Untad Blok C5 No 10 RT/RW 003/008, Tondo, Mantikulore, Palu, Sulawesi Tengah
Alamat Surabaya	:	Hans Regency15 A,Jl. Taman Manyar Tirtoyoso Utr No.15, Klampis Ngasem, Sukolilo, Surabaya City, East Java 60117
Telepon	:	08133505846
Email	:	taufiq1689@@gmail.com

PENDIDIKAN FORMAL

2016 – sekarang	: Mahasiswa S1 Informatika ITS
2013 – 2016	: SMA Negeri Model TerpaduMadani
2010 – 2013	: SMP Negeri Model Terpadu Madani
2004 – 2010	: SDN Inti Tondo

AKADEMIS

Kuliah	:	Departemen Teknik Informatika – Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya
Angkatan	:	2016
Semester	:	8 (Delapan)