Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

Clustering dan *Merging* Bisnis Proses Model Berdasarkan Graph Database

ARFIAN FIDIANTORO
NRP 5113100179

Dosen Pembimbing
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.
Dwi Sunaryono, S.Kom.,M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2017

(Halaman Ini Sengaja dikosongkan)

TUGAS AKHIR – KI141502

Clustering dan Merging bisnis proses model berdasarkan Graph Database

ARFIAN FIDIANTORO
NRP 5113100179

Dosen Pembimbing
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.
Dwi Sunaryono, S.Kom.,M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

(Halaman Ini Sengaja dikosongkan)

FINAL PROJECT– KI141502

Clustering and Merging on business process model based on Graph Database

ARFIAN FIDIANTORO
NRP 5113100179

Advisor
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc. Ph.D.
Dwi Sunaryono, S.Kom. M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2017

(Halaman Ini Sengaja dikosongkan)

LEMBAR PENGESAHAN

Clustering dan Merging bisnis proses model berdasarkan Graph Database

Tugas Akhir

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

Oleh:
ARFIAN FIDIANTORO
NRP. 5113 100 179

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Dwi Sunaryono, S.Kom. M.Kom.
NIP: 19720528 199702 1 001 ..... (pembimbing 2)

SURABAYA
JANUARI, 2018

(Halaman Ini Sengaja dikosongkan)

CLUSTERING DAN MERGING BISNIS PROSES MODEL BERDASARKAN GRAPH DATABASE

Nama Mahasiswa : Arfian Fidiantoro
NRP : 5113100179
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I :Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc. Ph.D.
Dosen Pembimbing II :Dwi Sunaryono, S.Kom. M.Kom.

ABSTRAK

Bisnis Proses adalah suatu kumpulan pekerjaan yang saling terkait untuk menyelesaikan suatu masalah tertentu. Bisnis process digunakan untuk memberikan panduan cara kerja suatu system atau process, dengan bisnis proses kita dapat mengetahui apakah yang dilakukan dan siapa yang melakukan nya, juga kita dapat menganalisa dengan cepat system tersebut apakah efisien atau tidak, dan dimana kita butuh memperbaiki system tersebut.

Analisis proses bisnis umumnya melibatkan pemetaan proses dan subproses di dalamnya hingga tingkatan aktivitas atau kegiatan. Umum nya kegiatan ini disebut process mining dan membutuhkan sedikit pengetahuan agar dapat melakukannya dan hasil visualisasi dari process ini akan menjadi sedikit sulit untuk dimengerti oleh orang awam.

Dalam tugas akhir ini diusulkan sebuah gagasan baru dalam cara menganalisa bisnis proses yaitu dengan cara yang sama tetapi tool yang berbeda, disini penulis mengusulkan untuk tool Graph Database lebih detailnya menggunakan Neo4J Graph database untuk mengoptimalkan visualisasi dari data bisnis process agar dapat dilihat dan dianalisa dengan mudah dan juga dapat dipahami oleh stakeholder dari system tersebut agar dapat digunakan untuk pengambilan keputusan.

Hasil implementasi dari gagasan ini menunjukan bahwa di Graph database pun dapat membangun bisnis process dari data (csv) dengan baik dan juga dapat menunjukkan beberapa Workflow pattern utama yang dapat digunakan untuk analisa, juga visualnya pun lebih baik dan lebih rapi, juga lebih detail dari setiap aktivitas nya.

Kata kunci: Bisnis Proses, Neo4J, Graph Database, Process Mining, Workflow Pattern.

CLUSTERING AND MERGING BUSINESS PROCESS MODEL BASED ON GRAPH DATABASE

Student Name	: Arfian Fidiantoro
NRP	: 5113100179
Major	: Teknik Informatika FTIf-ITS
Advisor I	: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc. Ph.D.
Advisor II	: Dwi Sunaryono, S.Kom. M.Kom.

ABSTRACT

Business Process is a collection of interrelated work to solve a particular problem. Business process model is used to provide guidance on how a system or process works, with business process model we can know what is done and who do it, also we can analyze quickly whether the system is efficient or not, and where we need to improve the system.

Business process analysis generally involves mapping processes and sub processes in it, up to the level of activity. In general this activity is called process mining and requires little bit of knowledge in order to do so and the visualization results of this process will be a bit difficult to understand by the general person.

In this final project proposed a new idea in how to analyze business process that is in the same way but with a different tool, here the author propose for Graph Database tool, in more details it's using Neo4J Graph database to optimize visualization of business process data so that it can be seen and analyzed easily and can also be understood by stakeholders of the system in order to be used for decision making.

The results of the implementation of this idea show that in Graph database we can build business process model from data (csv) as well, and we can also show some of the main Workflow

pattern that can be used for analysis, and also better and more neatly visual, also more detail from each of its activity.

Keywords: Business process, Neo4J, Graph Database, Process Mining, Workflow Pattern.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kehadirat Allah SWT, karena atas berkat rahmat-Nya, penulis dapat menyelesaikan Tugas Akhir yang berjudul:

“ Clustering dan Merging bisnis proses model berdasarkan Graph Database ”

Pengerjaan Tugas Akhir ini menjadi sebuah sarana untuk penulis mengasah cara penelitian dan memperdalam ilmu yang telah didapatkan selama menempuh pendidikan di kampus perjuangan 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. Bapak, Ibu, adik dan keluarga yang selalu memberikan dukungan penuh untuk menyelesaikan Tugas Akhir ini.
2. Bapak Riyanarto Sarno dan bapak Dwi Sunaryono selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan Tugas Akhir ini.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Segenap dosen rumpun mata kuliah Manajemen Informasi.

6. Rekan-rekan mahasiswa S2 Teknik Informatika yang telah banyak meluangkan waktu dan bersedia membantu penulis dalam memahami pengerjaan Tugas Akhir ini.
7. Teman-teman seperjuangan anak didik Tugas Akhir Prof. Riyanarto Sarno yaitu Renanda Agustiantoro, Andi Putra,Andy,Nezar,Rizal,Rey.
8. Teman-teman Teknik Informatika Angkatan 2013 yang selalu mendukung, menyemangati, membantu dan mendengarkan suka duka selama proses pengerjaan Tugas Akhir.
9. Serta semua pihak yang turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari masih ada kekurangan dalam penyusunan Tugas Akhir ini. Penulis mohon maaf atas kesalahan, kelalaian maupun kekurangan dalam penyusunan Tugas Akhir ini. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan ke depan

Surabaya, Januari 2018
Penulis

Arfian Fidiantoro

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman Ini Sengaja dikosongka

DAFTAR TABEL

(Halaman Ini Sengaja dikosongkan

BAB I
PENDAHULUAN

Bab ini memaparkan garis besar Tugas Akhir yang meliputi latar belakang, tujuan dan manfaat pembuatan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Akhir-akhir ini banyak perusahaan baik kecil maupun besar telah mengadopsi cara standarisasi prosedur dan operasi nya kedalam bisnis proses, dan data bisnis proses ini seringkali mempunyai ukuran yang cukup besar dan sedikit sulit bila digambarkan dengan model database biasa, karena bisnis proses ini mewakili cara kerja operasi perusahaan baik operasi yang besar dan rumit sampai yang kecil dan sederhana.

Maka dari itu perusahaan-perusahaan ini seringkali memiliki manajemen bisnis proses masing-masing, dalam ranah manajemen bisnis proses, mengadopsi cara membangun dan menyimpan bisnis proses secara baik dan benar dapat meningkatkan kualitas bisnis proses yang dimiliki. Penggunaan kembali bisnis proses yang telah ada atau bahkan sepenggal (*fragments*) dari bisnis proses tersebut adalah salah satu cara untuk membuat bisnis proses yang komplit atau bisnis proses *fragments* yang sedikit lebih 'kasar'.

Untuk menggunakan cara ini kita terlebih dahulu harus dapat memisahkan bisnis proses tersebut sesuai dengan kriteria yang di inginkan, di paper ini kami akan melakukan clustering terhadap kumpulan bisnis proses yang ada pada suatu sistem atau perusahaan. Setelah selesai proses *clustering* kami akan me-*merging* beberapa bisnis proses yang telah terpisah tersebut, untuk algoritma *merging* akan digunakan algoritma *Alpha* dan *Alpha++*.

Karena ukuran data yang cukup besar dan bentuk skema dari bisnis proses yang cenderung berat kearah *relationship*

eksperimen ini memilih *graph database* sebagai model database yang dipakai, disini saya akan memakai *Graph* database *Neo4j*, karena *Neo4j* adalah salah satu database yang paling mutakhir dari beberapa *NoSQL database* lainnya.

1.2 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Database hanya menggunakan *Neo4J*.
2. Bahasa *query* yang dipakai adalah *cypher* (teroptimasi untuk *graph pattern matching*).
3. Bentuk data input adalah Bisnis process (*Csv*) yang dijadikan bentuk graph nantinya.
4. *Workflow pattern* yang digunakan adalah *Control Flow Pattern* (*Split And, Split Xor, Join And, Join Xor, Sequence*) dan *Invisible Switch.*

1.3 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Cara menemukan bisnis proses pada bisnis proses *repository*.
2. Menemukan algoritma *Clustering* yang sesuai dan dapat menghasilkan hasil *clustering* sesuai yang diharapkan (*Classification*).
3. Menemukan algoritma *Merging* yang sesuai dan dapat menghasilkan bisnis proses yang baik dari penggabungan bisnis proses *fragments*.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini sebagai berikut:

1. Mampu menerapkan teknik process mining pada *Graph database*.

2. Mampu menerapkan algoritma *Merging* untuk membangun bisnis proses pada *graph database*.
3. Mampu menerapkan cara *clustering* pada bisnis proses yang dibuat di *graph database*.
4. Mampu menyajikan informasi hasil *process mining* dalam antar muka yang jelas dan mudah dipahami.

1.5 Manfaat

Pada akhir dari pengerjaan tugas akhir ini diharapkan hasil dari penelitian tugas akhir ini dapat menjadi rujukan tentang penggunaan algoritma yang cocok untuk membangun bisnis proses dengan cara *merging* dari data bisnis proses yang telah ada dan dapat menjadi dasar untuk penelitian tentang pembuatan bisnis proses pada *graph database* lebih lanjut.

1.6 Metodologi

Tahapan-tahapan yang ditempuh dalam pengerjaan Tugas Akhir ini sebagai berikut:

1. Studi literatur

Studi literatur yang dilakukan dalam pengerjaan penelitian tugas akhir ini adalah mengenai algoritma yang dipakai dalam melakukan *clustering* dan *merging* pada *graph database*. Selain itu, dilakukan juga studi literatur mengenai cara membangun bisnis proses dari sekumpulan data bisnis proses yang telah ada.

2. Analisis dan Perancangan Sistem

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat prototype metode, yang merupakan rancangan dasar dari metode yang akan dibuat. Serta dilakukan

desain suatu metode dan desain proses-proses yang ada serta membuat business process model dalam suatu proses pada salah satu perusahaan.

Kemudian beberapa kebutuhan fungsional dari sistem ini antara lain:

i. Memasukkan Data dari input (*csv)* ke dalam *graph database*.
ii. Membangun *bisnis proses model* dari data diatas.
iii. Membangun *relation* antar data pada bisnis proses model tersebut.
iv. Memperlihatkan bisnis proses model tersebut.
v. Menganalisa bisnis proses model tersebut.

3. Implementasi Sistem

Tahap implementasi ini meliputi implementasi algoritma yang telah dicoba berupa *query* dan bentuk struktur data berupa graph yang telah didukung oleh hasil analisis dan desain pada tahap sebelumnya. Implementasi ini dilakukan dengan menggunakan database *Neo4J*.

4. Uji Coba dan Evaluasi

Pengujian akan dilakukan pada satu database yang telah dimasukkan dataset berupa beberapa bisnis proses kedalam nya terlebih dahulu, lalu pengguna akan mencoba untuk membuat bisnis proses baru, bisnis proses dibuat dari bisnis proses yang telah dimasukkan ke database sebelumnya lalu akan menganalisa bisnis proses tersebut.

Uji coba dan evaluasi dilakukan untuk mengevaluasi jalannya perangkat lunak, mencari kesalahan, menganalisa faktor-faktor yang mempengaruhi kebenaran sistem, dan mengadakan perbaikan jika ada kekurangan.

5. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi sistem, rancangan antarmuka, proses lain yang telah dilakukan, hasil-hasil dan analisa yang telah didapatkan selama pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

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

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

Bab II Dasar Teori
Bab ini membahas teori sebagai penunjang yang berhubungan dengan pokok pembahasan yang menjadi dasar dari pembuatan Tugas Akhir ini.

Bab III 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 IV Implementasi
Bab ini membahas mengenai implementasi dari perancangan perangkat lunak dan implementasi fitur-fitur penunjang perangkat lunak.

Bab V Uji Coba dan Evaluasi
Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (usability) dari

perangkat lunak, serta melakukan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan, serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VI Kesimpulan
Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan perangkat lunak lebih lanjut.

Daftar Pustaka
Merupakan daftar referensi yang digunakan dalam proses membangun dan mengembangkan perangkat lunak dalam Tugas Akhir.

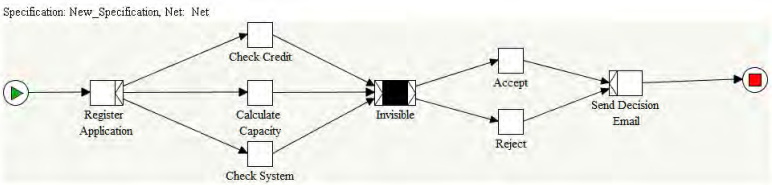
BAB II
TINJAUAN PUSTAKA

Pada bab ini diuraikan mengenai dasar-dasar teori yang digunakan dalam pengerjaan Tugas Akhir dengan tujuan untuk memberikan gambaran secara umum terhadap penelitian yang dikerjakan.

2.1 Model Proses Bisnis

Adalah teknik yang digunakan untuk mengelola, mendokumentasikan dan menggambarkan bagaimana operasi atau system mereka dijalankan atau diimplementasikan kedalam suatu bentuk model [1]. Ide dari teknik ini adalah bahwa teknik untuk memperoleh pemahaman yang lebih baik dari sistem kontrol fisik.
Fungsi dari model proses:
Organisasi menggunakan bisnis proses model (*BP model*) untuk memvisualisasi dokumen, memahami dan meningkatkan efisiensi proses mereka. Bagian dari proses bisnis Manajemen (*BPM*) yaitu pemodelan BP telah digunakan oleh organisasi sebagai alat untuk memetakan sesuatu hal dan menjadi dasar untuk menentukan masa depan dengan berbagai perbaikan [2].
Beberapa fungsi lainnya untuk BP model adalah:

- Untuk menciptakan *visual model* dari proses-proses yang ada
- Meluruskan dan memperbaiki operasi
- Meningkatkan komunikasi antar proses
- Meningkatkan efisiensi operasional
- Memperoleh keunggulan dalam kompetisi

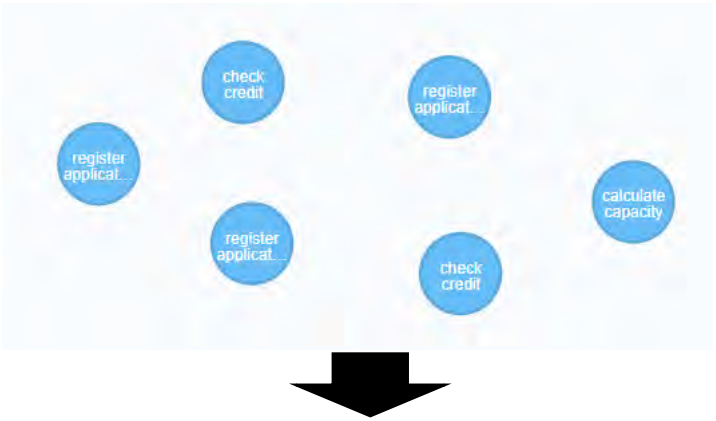


Gambar 2.1 Contoh bisnis proses model

2.2 *Clustering*

Pada teori graph, *clustering* adalah istilah untuk memisahkan atau mengelompokkan data sesuai dengan nilai-nilai similarity antar data, data-data ini seringkali membuat kelompok-kelompok.

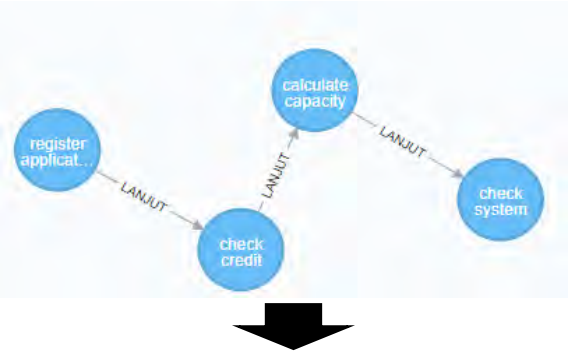
Di dunia nyata seringkali kita mendapat bukti bahwa nodes sering membuat kelompok-kelompok yang berdekatan dengan yang bentuk nya hampir sama dengan dirinya sendiri.

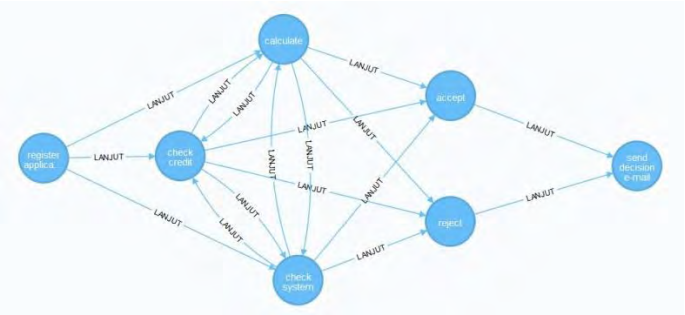


Gambar 2.2 Contoh Clustering (Cluster sesuai CaseId)

2.3 *Merging*

Merging adalah istilah untuk menggabungkan beberapa data menjadi satu data. Di tugas ini artinya proses ini menggabungkan beberapa bisnis proses atau bisnis proses fragment menjadi satu bisnis proses yang utuh.





Gambar 2.3 Contoh Merging dari hasil clustering

2.4 NoSql Database

NoSql (dasar nya merujuk pada *"Non-SQL","Non-relation*al", atau "bukan hanya SQL") database memberikan mekanisme untuk menyimpan dan mengambil data yang dimodelkan selain dengan relasi berbentuk table yang digunakan pada *relation*al *database*.

Mulai menjadi populer di awal abad ke-21, dipicu oleh kebutuhan perusahaan-perusahaan web seperti Facebook, Google, dan Amazon.com. *NoSql database* sering digunakan dalam big data dan aplikasi web *real-time*. Motivasi untuk pendekatan ini meliputi: design yang simple, *"horizontal" scaling* yang lebih simple ke *cluster-cluster* beberapa mesin (komputer) (yang menjadi problem dalam database *relation*al), dan juga *finer control* untuk ketersediaan. Banyak penyimpanan NoSql yang tidak mengutamakan *consistency* tetapi lebih mengutamakan ke ketersediaan, kecepatan, dan juga toleransi partisi.

2.5 *Big Data*

Big data adalah istilah untuk data set yang sangat besar atau terlalu kompleks yang dimana software pengolah data tradisional tidak dapat menanganinya. Tantangan-tantangan dalam menangani big data termasuk:

- *Capture* (menangkap data)

- *Storage* (menyimpan data)
- *Analysis* data
- *Data curation*
- *Search* (pencarian data)
- *Sharing* data
- Visualisasi data
- *Query* data
- Meng-*update* data
- Dan privasi dari suatu informasi

Istilah big data seringkali merujuk kepada penggunaan analisa prediksi, analisa tingkah laku user, atau metode analisa data lebih lanjut lainnya yang mengambil nilai dari data, dan memindahkannya ke *dataset* dengan ukuran tertentu.

2.6 *Graph Database*

Graph database atau yang disebut *graph-oriented database* adalah salah satu tipe dari *NoSql Database* yang menggunakan teori graph untuk menyimpan, memapkan, dan mengolah relasi-relasi antar data. *Graph database* pada dasar nya adalah koleksi dari nodes dan *edges*.[3] Setiap node mewakili 1 *entity* atau data(seperti orang) dan setiap *edge* mewakili relasi atau koneksi antara 2 nodes. Setiap node di *graph database* memiliki *unique identifier*, *edge* yang masuk dan keluar, dan properti yang memiliki nilai. Setiap *edge* memiliki *unique identifier*, *start point* dan *end point*, juga properti yang memiliki nilai.

Pemikiran utama dari *graph database* adalah "*If you can whiteboard it, you can graph it*". *Graph database* cocok untuk digunakan menganalisa relasi-relasi, karena itu sangat banyak keinginan untuk menggunakan graph database untuk memining data dari *social media*.

Tidak seperti database *system* lainnya, pada *graph database* pemikiran dan prioritas utama adalah relasi dari setiap data [4].

Keunggulan dari *Graph Database*:

- Performa

- Untuk penanganan relasi data yang intensif, *graph database* meningkatkan kinerja dengan beberapa kali lipat lebih baik. Pada database tradisional, *Query* tentang relasi akan menjadi jauh lebih lambat seiring dengan jumlah dan kedalaman relasi antar data meningkat. Tapi sebaliknya, kinerja *graph database* tetap sama bahkan ketika data anda tumbuh sepanjang tahun [4].

- Fleksibel
  - Pada *graph database*, tim IT dan tim Arsitek Data akan bergerak lebih cepat sehingga menyamai seperti kecepatan bisnis, ini dikarenakan struktur dan skema model graph di database akan fleksibel untuk dipakai pada aplikasi dan industri yang selalu berubah-ubah. Daripada memodelkan ulang *domain* baru, tim database mampu untuk menambahkan perubahan ke struktur grafik yang telah ada tanpa membahayakan fungsi yang telah berjalan dan menggunakan waktu untuk tugas yang lebih efisien [4].
- Agility
  - *Graph database* sangat sesuai dengan trend saat ini yaitu tangkas, praktis, dan pengembangan berbasis uji coba. Itulah persyaratan standar pada metode pengembangan saat ini, maka dari itu hal ini memungkinkan *graph database* Anda berkembang dengan *requirements* dari berbagai aplikasi yang pernah ada. *Graph database* yang modern dilengkapi dan dikhususkan untuk

pengembangan tanpa bergesekan dengan fungsi lain dan pemeliharaan yang cukup mudah [4].

Kredit dari konsep dibelakang untuk meng-graph kan database sering diberikan pada matematikawan abad ke 18 “Leonhard Euler” [3].

2.7 *Neo4J*

Neo4J graph database, yang dikembangkan oleh *Neo Technology*, adalah salah satu *NoSql graph database* yang paling populer saat ini. Neo4j adalah salah satu *NoSql database* yang paling maju dan mutakhir, ia juga terdokumentasi dengan baik dan memberikan banyak tool, yang meliputi aplikasi visualisasi berbasis *Eclipse* yaitu *NeoEclipse* [5].

Gambar 2.4 Lambang Neo4J Database

2.8 *Cypher*

Cypher atau CQL *(Cypher Query Language)* adalah salah satu bahasa yang dipakai oleh database Neo4j, bahasa ini dibuat dan dioptimasikan oleh tim dari *Neo4j database* untuk *graph pattern matching* dan *query Relationship* [5].

```
CREATE ( <node-name>:<label-name>
  {
    <Property1-name>:<Property1-Value>
    ........
    <Propertyn-name>:<Propertyn-Value>
  }
)
```

Gambar 2.5 Contoh query Cypher

2.9 *Linked List*

Dalam ilmu computer, *linked list* adalah koleksi elemen data, yang disebut sebagai *nodes*, setiap *nodes* nya memiliki *pointer* untuk menunjuk ke *nodes* selanjutnya. Struktur datanya terdiri dari kelompok node yang merepresentasikan sebuah *sequence*.

2.10 *Inductive Miner*

Inductive miner adalah hasil improvisasi dari *alpha* dan *heuristic miner*, ini adalah salah satu metode *miner* yang menjamin kelengkapan (*soundness*) dari proses model yang dihasilkan. *Inductive miner* biasanya tidak dapat menghasilkan *process model* berbentuk *Petrinets* dengan baik, tetapi hasil yang dikeluarkan biasanya berbentuk *process tree*.
Tahapan metode ini adalah:

- Cari Potongan yang paling dominan (1 macam) pada *event logs*
- Temukan *Operator* atau *Pattern* pada potongan tersebut
- Lanjutkan pada sisa *event logs* yang belum diproses

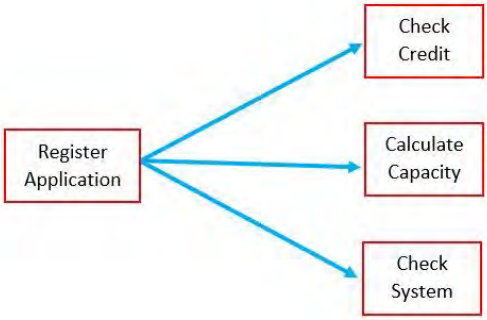
Inductive miner akan melakukan 3 hal diatas berulang-ulang sampai seluruh *event logs* selesai diproses [6].

2.11 *Workflow Pattern*

Workflow pattern adalah bentuk khusus dari *design pattern* seperti yang didefinisikan pada masing-masing bidang rekayasa perangkat lunak atau rekayasa bisnis proses [7]. Terdapat cukup banyak *workflow pattern*, salah satunya adalah *control flow pattern* yang biasanya dibagi dalam 6 kategori [8]. :

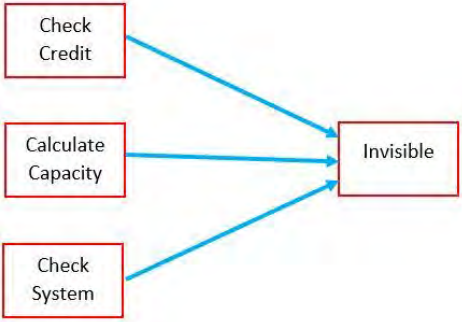
- *Basic Patterns*
    - *Sequence*
    - *Parallel Split (AND Split)*
        - *Parallel Split* atau *AND split* adalah *workflow pattern* dimana proses terpisah menjadi 2

atau lebih cabang dan semua cabang tersebut dijalankan secara bersamaan.[9]



Gambar 2.6 Contoh pattern AND Split

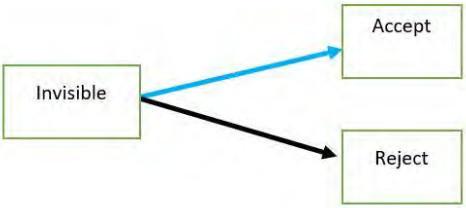
- *Synchronization (AND Join)*
    - Penggabungan 2 atau lebih cabang menjadi satu cabang , proses akan dilanjutkan saat semua proses di semua cabang yang bergabung telah diselesaikan.[9]



Gambar 2.7 Contoh pattern AND Join

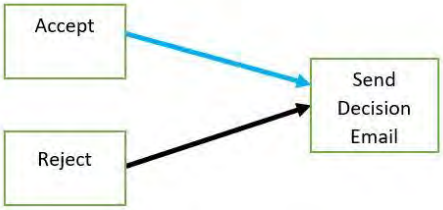
- *Exclusive Choice (XOR Split)*
    - Adalah *Workflow Pattern* dimana proses terpisah menjadi 2 atau lebih cabang, proses akan langsung dilakukan saat terjadi

percabangan dan hanya 1 cabang yang akan dilakukan.[9]



Gambar 2.8 Contoh pattern XOR Split

- *Simple Merge (XOR Join)*
    - Penggabungan 2 atau lebih cabang menjadi satu cabang. Di *join* ini tidak perlu menunggu semua cabang selesai karena hanya satu cabang yang menjalankan proses jadi bila ada proses yang masuk maka langsung dilanjutkan alur proses tersebut.[9]



Gambar 2.9 Contoh pattern XOR Join

- *Advanced Branching and Synchronization*
    - *Multi Choice (OR Split)*
    - *Structured Synchronizing Merge (OR Join)*
- *Structural Pattern*
- *Multiple Instance Pattern*
- *State Based Pattern*
- *Cancellation Patterns*

BAB III
ANALISIS DAN PERANCANGAN

Bab ini akan membahas mengenai analisis dan langkah-langkah pembuatan BP Model di graph database.

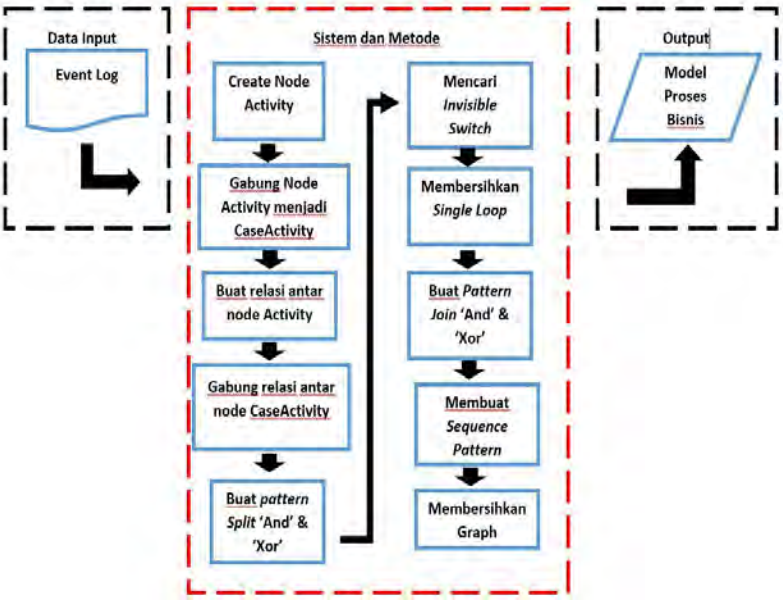
3.1 Tools yang dipakai

Perkembangan teknologi menyebabkan banyak organisasi baik ingin maupun tidak ingin untuk mengubah cara bekerja dan evaluasi mereka. Salah satu cara untuk memastikan cara bekerja dan proses yang dikerjakan berjalan dengan baik adalah dengan membuat SOP (*Standard operating procedure*) dan salah satu cara yang sering digunakan untuk meng-evaluasi SOP adalah dengan membuat Bisnis proses model (*BP Model*).

Metode dan *query* yang dibangun kali ini bertujuan untuk membantu analis untuk meng-evaluasi *BP Model* secara baik dan tepat. Pada *BP model* yang dibangun kali ini akan berbentuk graph yang bentuk akhir nya akan hampir sama dengan *BP Model* sesuai *event logs* yang di masukkan. *Query* dan metode yang dibangun ini diharapkan mampu memudahkan analis dalam membangun *BP Model* yang sesuai dengan *Event Log* nya.

Penulis Menggunakan Database berbentuk *Graph Database* dan juga *Tools Neo4J* untuk membangun *BP models*. *Neo4J* digunakan karena termasuk salah satu yang paling terdepan dalam Database NoSql juga di neo4j kita tidak perlu untuk membangun Graph satu persatu, hanya butuh beberapa arahan dan *query* agar graph dapat dibuat dengan baik. Penulis juga memakai bahasa *Cypher Query Language* yang telah dioptimasi untuk *Neo4J* dan juga *Classification* dan *searching details* dari data dalam database.

3.2 Perancangan Metode dan *Query*



Gambar 3.1 Flow diagram Metode yang dipakai.

3.2.1 Deskripsi dan gambaran Umum dari Metode dan *Query*

Tugas akhir yang akan dikembangkan adalah berupa metode dan *query* yang bisa dipakai untuk membangun *BP Model* secara otomatis. *BP model* dibangun dari input *event logs* atau pun gambar SOP bila tidak mempunyai *event logs*.

Beberapa langkah Utama dalam metode yang digunakan untuk input event logs, adalah:

- *Load* dan buat node activity dari *event logs*.
- *Load* dan gabung node CaseActivity dari *event logs*.
- Buat Hubungan (*Relationship*) sementara antar node Activity menggunakan metode *Linked list*.

- Gabung Hubungan (*Relationship*) sementara antar node CaseActivity menggunakan metode *Linked list*.
- Membuat *WorkflowPattern split 'AND'*.
- Membuat *WorkflowPattern split 'XOR'*.
- Mencari *Invisible Switch* (*Optional*)
- Membersihkan Single Loop
- Membuat *WorkflowPattern 'ANDJoin'*
- Membuat *WorkflowPattern 'XORJoin'*
- Membuat *Sequence Pattern*.
- Membersihkan graph bila dirasa perlu (*Parallelism*).
- Memeriksa hasil dari *query* diatas dan *BP model* yang dihasilkan.

Selama penjelasan Metode dan *query, event logs* yang dipakai adalah 'Artificial-Loan.csv'

3.2.2 Deskripsi detail dari setiap Langkah

3.2.2.1 Load dan buat node activity dari event logs.

Pada langkah pertama kita akan mengupload event logs dari computer kita menuju ke Database neo4j dengan menggunakan *query* dibawah sekaligus membuat nodes Activity yang berisi seluruh data dari event logs tersebut
Dengan *query* dan *pseudo code*:

```
Using periodic commit
LOAD CSV with headers FROM "file:///Artificial-LoanProcess.csv"
  AS line
Merge (:Activity
{CaseId:line.case,Name:line.event,EndTime:line.completeTime })
```

Gambar 3.2 *Query load* dan *create* node Activity

```
Pseudo code:
Sampai seluruh data terproses
        Load event logs Artificial-Loan Process
        Data dari event logs menjadi tipe Line
        Buat Node Activity dari hasil Proses load dengan atribut
        CaseId,Nama,WaktuSelesai
End For
```

Gambar 3.3 *Pseudo code load* dan *create* node Activity

Penjelasan:

Pertama *Load event logs*, lalu ubah data yang diupload menjadi bentuk *line*. Setelahnya buat node Activity dengan atribut *CaseId, Name,* dan *Complete Time*. Atribut ini didapat dari data yang telah diupload dan dalam bentuk *line*. Langkah ini akan diulang sebanyak 5000 kali atau sampai semua data telah diupload dan diproses.

3.2.2.2 Load dan gabung node CaseActivity dari event logs.

Pada Langkah kedua ini hampir sama apa yang akan dilakukan dengan pada langkah pertama, langkah ini juga bertujuan untuk mengupload data event logs ke database.
Dengan *Query* dan *Pseudo code*:

```
Using periodic commit
LOAD CSV with headers FROM "file:///Artificial-
LoanProcess.csv"
  AS line
Merge (:CaseActivity {Name:line.event })
```

Gambar 3.4 Query load dan create node CaseActivity

Pseudo code:
Sampai seluruh data terproses
    Load event logs Artificial-Loan Process
    Data dari event logs menjadi tipe *Line*
    Merge Node CaseActivity dari hasil Proses load dengan atribut nama
End For

Gambar 3.5 *Pseudo code* load dan create node CaseActivity

Penjelasan:

Hampir sama dengan langkah 1, pertama *Load event logs*, ubah data yang diupload menjadi bentuk *line*. Setelahnya *merge*/buat node CaseActivity dengan atribut Nama. Langkah ini akan diulang sebanyak 5000 kali atau sampai semua data telah diupload dan diproses.

3.2.2.3 Buat Hubungan (*Relationship*) antar node Activity menggunakan metode Linked *list*.

Langkah selanjutnya kami membangun *relation* atau hubungan antar node Activity, hubungan atau *relation* ini akan digunakan untuk menemukan detail dari graph yang akan coba di bangun.
Dengan *Query* dan *pseudo code*:

```
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
WHERE s1.CaseId = s2.CaseId
MERGE (s1)-[:LANJUT]->(s2)
```

Gambar 3.6 Query relasi node Activity

Pseudo code:
Ambil Node Activity
Kumpulkan sebagai *list* dengan nama A1
Ambil jumlah range (*max* A1) – 2 sebagai variable U2
Buat 2 variable berisi *list* A1,
    satu variable dengan index U2 sebagai A3 dan
    yang lain (U2+1) sebagai A4
Cocokkan Atribut CaseId pada A3 dan A4 agar sama
Buat relation A3 -> A4

Gambar 3.7 *Pseudo code* relasi node Activity

Penjelasan:

Pada langkah ini kita akan membangun relasi antar Node Activity. Pertama ambil node activity, kumpulkan hasil menjadi bentuk *list*, buka *list* dan ambil *Max* dari *list* tersebut, lalu buat variable yang berisi range antara 0 – (*Max*-2) beri nama U2. Lalu buat 2 variable lagi kali ini berisi *list* pertama yang mempunyai range U2 dan U2+1, berikan where untuk mencocokan Node yang sesuai setelah mendapatkan node yang sesuai bangun relasi Antara node tersebut.

3.2.2.4 Gabung Hubungan (*Relationship*) sementara antar node CaseActivity menggunakan metode Linked *list*.

Langkah ke 4 hampir sama dengan langkah ke 3 tetapi yang dibuat *relation* antar nodes CaseActivity, dengan *query* yang hamper sama tetapi menggunakan 2 tipe nodes di *query* ini. Langkah ini juga berfungsi untuk membangun *relation* antar nodes dari data event logs yang telah dimasukkan sebelumnya.
Dengan *Query* dan *pseudo code*:

```
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)-[:LANJUT]->(b)
```

Gambar 3.8 Query relasi node CaseActivity

Pseudo code:
Ambil Node Activity
Kumpulkan sebagai *list* dengan nama A1
Ambil jumlah range *max* A1 – 2 sebagai variable U2
Buat 2 variable berisi *list* A1 ,
satu variable dengan index U2 sebagai A3 dan
yang lain (U2+1) sebagai A4
Ambil 2 node CaseActivity dengan nama B1 dan B2
Cocokkan
Atribut CaseId pada A3 dan A4 ,
atribut nama pada A3 dan B1,
A4 dan B2 Agar Sama
Buat relation B1 -> B2

Gambar 3.9 *Pseudo code* relasi node CaseActivity

Penjelasan :

Penjelasan langkah ke 4 hampir sama dengan langkah ke 3, bedanya apabila pada langkah ke 3 relasi tersebut permanen maka di langkah ke 4 relasi untuk CaseActivity hanyalah sementara yang nantinya akan digantikan dengan relasi yang lebih detail.

Pertama ambil node activity, kumpulkan hasil menjadi bentuk *list*, buka *list* dan ambil *Max* dari *list* tersebut, lalu buat variable yang berisi range antara 0 – (*Max*-2) beri nama U2. Lalu buat 2 variable lagi kali ini berisi *list* pertama yang mempunyai range U2 dan U2+1, setelahnya ambil 2 node CaseActivity,

berikan where untuk mencocokan Node Activity dalam *list* dan node CaseActivity setelah mendapatkan node CaseActivity yang sesuai bangun relasi Antara node tersebut.

3.2.2.5 Membuat WorkflowPattern split 'AND'.

Pada langkah ini kita akan membuat salah satu Workflow Pattern yang umum digunakan saat membuat process model yaitu 'Parallel Split' atau lebih umum disebut 'AND'.
Dengan *Query* dan *Pseudo code*:

```
MATCH (b:CaseActivity)<-[p:LANJUT]-(a:CaseActivity)-
[q:LANJUT]->(c:CaseActivity)
WHERE (b)-[:LANJUT]->(c) And (c)-[:LANJUT]->(b)And Not (c)-
[:LANJUT]->(a) And Not (b)-[:LANJUT]->(a)
DELETE p,q
MERGE (b)<-[:AND2]-(a)
```

Gambar 3.10 Query WorkflowPattern AndSplit

```
Pseudo code :
Match struktur relasi  Node CaseActivity b <- a -> c
Cocokkan dimana
        ada parallelism Antara b-c dan
        tidak ada parallelism antara a-b dan a-c
Hapus relasi a dengan b
Buat relasi And baru untuk a-b
```

Gambar 3.11 *Pseudo code* WorkflowPattern AndSplit

Penjelasan :

Pada langkah ke 5 kita akan membuat Workflow Pattern yaitu 'AND' Split , pattern ini dipakai saat proses membagi alur nya menjadi 2 proses tetapi 2 proses tersebut dilakukan bersamaan.

Ambil relasi Antara node CaseActivity yang memiliki 2 outbound relationship atau struktur B<-A->C, cocokan relasi tersebut dimana ada parallelism diantara B-C dan tidak ada

parallelism Antara A-B dan A-C, lalu hapus relasi dari A-B dan bangun relasi AND Antara node A-B.

3.2.2.6 Membuat WorkflowPattern split 'XOR'.

Pada langkah ini kita akam membuat Workflow Pattern lagi yaitu 'Exclusive Choice' atau lebih umum dikenal dengan 'XOR' Split. Pattern ini juga umum digunakan pada process model, karena sangat sering dipakai di process-process secara real.
Dengan *Query* dan *Pseudo code*:

```
MATCH (b:CaseActivity)<-[p:LANJUT]-(a:CaseActivity)-
[q:LANJUT]->(c:CaseActivity)
where Not (b)-[:LANJUT]->(c) And Not (c)-[:LANJUT]->(b) and
Not (a)-[:AND2]->(b)
Delete p,q
Merge (b)<-[:XOR]-(a)
```

Gambar 3.12 Query WorkflowPattern XorSplit

```
Pseudo code :
Match struktur relasi  Node CaseActivity b <- a -> c
Cocokkan dimana
        tidak ada parallelism Antara b-c dan
        tidak ada relasi And antara a-b
Hapus relasi a dengan b
Buat relasi Xor baru untuk a-b
```

Gambar 3.13 *Pseudo code* WorkflowPattern XorSplit

Penjelasan:

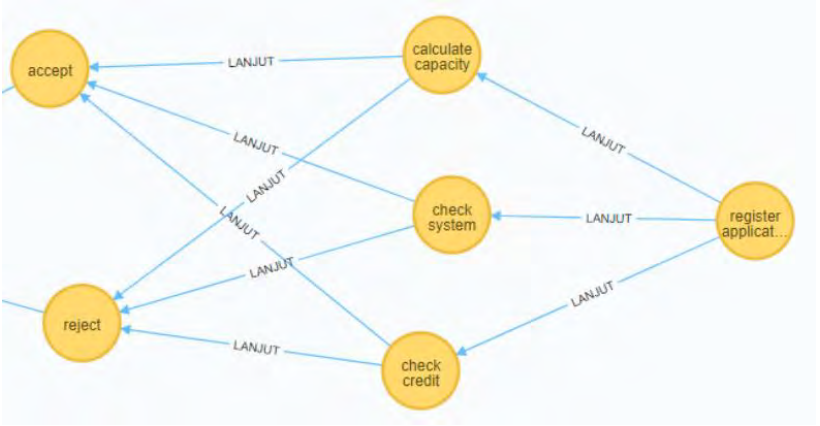
Pada langkah ini kita akan membuat salah satu Workflow pattern yaitu 'XOR' Split, pattern ini dilakukan saat proses membagi alur nya menjadi 2 proses dan hanya melakukan salah satu proses tersebut.

Ambil relasi Antara node CaseActivity yang memiliki 2 outbound relationship atau struktur B<-A->C, cocokan relasi

tersebut dimana tidak ada parallelism diantara B-C dan tidak ada relasi AND Antara A-B, lalu hapus relasi dari A-B dan bangun relasi XOR Antara node A-B.

3.2.2.7 Menemukan dan membuat Invisible Switch (Optional).

Langkah selanjutnya adalah menemukan apakah pada proses model ini terdapat / butuh invisible node, salah satu macam dari invisible node adalah Invisible Switch, dimana invisible ini terdapat pada saat node melakukan split process dan langkah selanjutnya melakukan split process lagi, dengan contoh:



Gambar 3.14 Contoh tampilan Invisible Switch

Penjelasan:

Pada “Register Application” flow dari process melakukan split “And” dengan tujuan node “Calculate Capacity”, “Check System”, “Check Credit”. Dan ketiga node tersebut langsung melakukan split “Xor” tanpa melakukan sequence *relation* terlebih dahulu, maka terjadi lah Invisible Switch.

Dengan *Query* dan *pseudo code* (‘And’ dan ‘Xor’) :

```
Create (s:CaseActivity{Name:'Invisible'})
```

Gambar 3.15 Query create Invisible node

```
MATCH (s:CaseActivity)
WHERE s.Name='Invisible'
With Collect (s) as Invi
Unwind Invi as invis1
Match (a:CaseActivity)-[b:AND]->(c:CaseActivity)-[d:XOR]-
>(e:CaseActivity)
Merge (c)-[:LANJUT]->(invis1)
```

Gambar 3.16 Query Invisible Switch (And-Xor) part 1

```
MATCH (s:CaseActivity)
WHERE s.Name='Invisible'
With Collect (s) as Invi
Unwind Invi as invis1
Match (a:CaseActivity)-[b:AND]->(c:CaseActivity)-[d:XOR]-
>(e:CaseActivity)
Delete d
Merge (invis1)-[:XOR]->(e)
```

Gambar 3.17 Query Invisible Switch (And-Xor) part 2

Pseudo code:
Match node CaseActivity dengan atribut nama Invisible
Kumpulkan menjadi *list*
Buka *list* sebagai A1
Match struktur relasi Node CaseActivity A-and->B-xor->C
Buat relasi Lanjut antara B dan A1

Match node CaseActivity dengan atribut nama Invisible
Kumpulkan menjadi *list*
Buka *list* sebagai A1
Match struktur relasi Node CaseActivity A-and->B-xor->C
Delete relasi xor B-C
Buat relasi Xor antara A1 dan C

Gambar 3.18 *Pseudo code* Invisible Switch (And-Xor)

Dengan *Query* dan *pseudo code* ('Xor' dan 'Xor'):

Create (s:CaseActivity{Name:'Invisible'})

Gambar 3.19 Query create Invisible node

```
MATCH (s:CaseActivity)
WHERE s.Name='Invisible'
With Collect (s) as Invi
Unwind Invi as invis1
Match (a:CaseActivity)-[b:XOR]->(c:CaseActivity)-[d:XOR]-
>(e:CaseActivity)
Merge (c)-[:LANJUT]->(invis1)
```

Gambar 3.20 Query Invisible Switch (Xor-Xor) part 1

```
MATCH (s: CaseActivity)
WHERE s.Name='Invisible'
With Collect (s) as Invi
Unwind Invi as invis1
Match (a:CaseActivity)-[b:XOR]->(c:CaseActivity)-[d:XOR]-
>(e:CaseActivity)
Delete d
Merge (invis1)-[:XOR2]->(e)
```

Gambar 3.21 Query Invisible Switch (Xor-Xor) part 2

Pseudo code:
Match node CaseActivity dengan atribut nama Invisible
Kumpulkan menjadi *list*
Buka *list* sebagai A1
Match struktur relasi Node CaseActivity A-xor->B-xor->C
Buat relasi Lanjut antara B dan A1
Match node CaseActivity dengan atribut nama Invisible
Kumpulkan menjadi *list*
Buka *list* sebagai A1
Match struktur relasi Node CaseActivity A-xor->B-xor->C
Delete relasi xor B-C
Buat relasi Xor antara A1 dan C

Gambar 3.22 Pseudo code Invisible Switch (Xor-Xor)

Penjelasan:

Pada *query* ini terdapat 2 *query* tergantung dengan situasi yang ada pada process model, setiap *query* tersebut dibagi menjadi 2 bagian, tetapi sebelumnya kita harus men-*create* node Invisible terlebih dahulu. Pada bagian pertama seperti contoh pada gambar 3.2 pada node "Calculate Capacity","Check System","Check Credit" (yang selanjutnya akan disebut node awal) akan dibuatkan *relationship* dengan node Invisible dan bagian ke dua akan dibuatkan *relationship* Antara node Invisible dengan node "Accept", "Reject" (yang selanjutnya akan disebut sebagai node akhir).

Setiap *query* memiliki 2 bagian tetapi hampir sama bagian tersebut hanya berbeda di bagian akhir, Bagian pertama dimulai dengan mengambil node CaseActivity dengan atribut nama 'Invisible' , kumpulkan menjadi *list* dan buka sehingga menjadi variable yang dapat dipakai. Lalu ambil relasi dari node CaseActivity dengan struktur A-And->B-Xor->C (untuk tipe pertama) atau A-Xor->B-Xor->C (untuk tipe kedua) lalu buat relasi lanjut antara variable yang dibuat dengan B.

Untuk bagian kedua langkah yang dilakukan sama dengan bagian pertama sampai bagian ambil relasi dari node CaseActivity dengan struktur A-And->B-Xor->C (untuk tipe pertama) atau A-Xor->B-Xor->C (untuk tipe kedua) setelahnya hapus relasi Xor antara B-C dan buat relasi Xor antara variable yang telah dibuat dengan node C.

Untuk *query* ke 2 semua sama Cuma dibagian struktur yang dicari yang berbeda seperti yang telah dijelaskan yaitu ubah A-And->B-Xor->C menjadi A-Xor->B-Xor->C.

3.2.2.8 Membersihkan Single Loop (Optional)

Pada langkah ini eksperimen akan membersihkan semua single Loop yang ada pada process model untuk mempersiapkan process model sebelum membuat pattern Join
Dengan *Query* dan *Pseudo code*:

```
MATCH (a:CaseActivity)<-[p:LANJUT]-(a:CaseActivity)
delete p
```

Gambar 3.23 Query Clear Single Loop

```
Pseudo code :
Match struktur relasi Node CaseActivity A->A
Hapus relasi A->A
```

Gambar 3.24 *Pseudo code* Clear Single Loop

Penjelasan:
Pada match kita akan mencocokan struktur Single Loop yaitu relasi Node dengan dirinya sendiri, setelah mendapatkan hasil dari match dengan struktur A->A maka relasi tersebut akan dihapus.

3.2.2.9 Membuat WorkflowPattern split 'ANDJoin'.

Di langkah selanjutnya eksperimen ini akan mencoba membuat And Join pattern, yaitu saat 2 atau lebih cabang process flow yang berjalan bersama bergabung menjadi 1 cabang.
Dengan *Query* dan *Pseudo code*:

```
Match (a:CaseActivity)-[r:LANJUT]->(n:CaseActivity)
Where size((n)<--()) > 1 and (size((a)-->()) > 1)And Not (n)-
[:LANJUT]->(a)
Delete r
Merge(a)-[:ANDJoin]->(n)
```

Gambar 3.25 Query WorkflowPattern AndJoin

```
Pseudo code :
Match struktur relasi Node CaseActivity A->B
Cocokkan dimana
        (relasi yang masuk ke B >1) dan
        (relasi yang keluar dari A > 1) dan
        tidak ada parallelism Antara A dan B
Hapus relasi A->B
Buat relasi AndJoin Antara A-B
```

Gambar 3.26 *Pseudo code* WorkflowPattern AndJoin

Penjelasan:

Pada langkah ini eksperimen akan membangun *And Join pattern* yaitu dimana 2 cabang akan menjadi 1, hampir sama dengan XorJoin pattern hanya ada beberapa kondisi yang membedakannya.

Pertama Ambil relasi dari node CaseActivity dengan struktur A->B, lalu gunakan where dengan kondisi Inbound relasi dari B > 1, Outbound relasi dari A > 1, dan tidak ada parallelism antara node A-B. setelah mendapat hasil dari where maka hapus relasi A-B dan bangun relasi AndJoin antara node A-B.

3.2.2.10 Membuat WorkflowPattern split 'XORJoin'.

Pada langkah ini eksperimen ini akan membangun Xor Join pattern, yaitu saat process flow memecah menjadi 2 atau lebih cabang, tetapi hanya 1 cabang yang diproses selanjutnya, saat 2 atau lebih cabang ini bergabung maka disitulah terjadi Xor Join pattern.
Dengan *Query* dan *Pseudo code*:

```
match (a:CaseActivity)-[r:LANJUT]->(n:CaseActivity)
where size((n)<--())>1 and (size((a)<--()) = 1)
delete r
Merge(a)-[:XORJoin]->(n)
```

Gambar 3.27 Query WorkflowPattern XorJoin

```
Pseudo code :
Match struktur relasi Node CaseActivity A->B
Cocokkan dimana
        (relasi yang masuk ke B >1) dan
        (relasi yang masuk ke A = 1)
Hapus relasi A->B
Buat relasi XorJoin Antara A-B
```

Gambar 3.28 Pseudo code WorkflowPattern XorJoin

Penjelasan :

Ambil relasi node CaseActivity dengan struktur A->B, gunakan where dengan kondisi inbound relasi dari node B > 1 dan Inbound relasi dari A = 1, setelah mendapatkan hasil dari where maka selanjutnya hapus relasi A->B dan bangun relasi XorJoin antara node A-B.

3.2.2.11 Membuat relasi *Sequence* pada node CaseActivity.

Pada langkah ini eksperimen akan membuat relation permanen yaitu sequence relation dari process flow yang dimiliki.
Dengan *Query* dan *Pseudo code*:

```
MATCH (b:CaseActivity)<-[p:LANJUT]-(a:CaseActivity)
Where size ((a)-->()) = 1
Delete p
Merge (b)<-[:Next]-(a)
```

Gambar 3.29 Query Sequence Pattern

```
Pseudo code:
Match Struktur relasi Node CaseActivity A->B
Cocokkan dimana Relasi yang keluar dari A = 1
Hapus relasi A->B
Buat relasi Next pada A-B
```

Gambar 3.30 Pseudo code Sequence Pattern

Penjelasan:

Sequence pattern mungkin adalah salah satu pattern yang paling umum pada process model, tapi terkadang malah tidak ada sama sekali seperti pada event logs kali ini.

Pertama ambil relasi node CaseActivity dengan struktur A->B , gunakan where dengan kondisi Outbound relasi dari node A=1 untuk mendapatkan struktur sequence pattern yang sesuai, lalu hapus relasi A->B , dan bangun relasi Next antara node A-B.

3.2.2.12 Membersihkan graph bila dirasa perlu.

Pada langkah ini akan dilakukan pembersihan graph dari relationship yang tidak diperlukan seperti *parallelism* yang telah digantikan dengan And Split pattern.
Dengan *Query* dan *Pseudo code* (Parallelism):

```
MATCH (b:CaseActivity)<-[p:LANJUT]-(a:CaseActivity)
Where (b)-[:LANJUT]->(a) And (a)-[:LANJUT]->(b)
Delete p
```

Gambar 3.31 Query pembersihan parallelism

```
Pseudo code:
Match struktur relasi Node CaseActivity A->B
Cocokkan dimana ada parallelism antara A-B
Hapus relasi A->B
```

Gambar 3.32 Pseudo code pembersihan parallelism

Penjelasan:

Pada *query* ini akan dicari relationship yang menjadi parallelism yaitu saat node awal menuju node akhir, dan node akhir menuju node awal. Setelah mendapatkan hasil dari *query* where maka pada *query* delete eksperimen ini akan menghapus semua relationship pada hasil *query* where tersebut.

3.2.2.13 Memeriksa hasil dari *query* diatas dan proses model yang dihasilkan.

Di langkah ini kita akan melakukan cek hasil dari keseluruhan process model yang telah kita buat.
Dengan *Query* dan *Pseudo code*:

```
MATCH (n:CaseActivity) RETURN n
```

Gambar 3.33 Query Hasil proses model

```
Pseudo code:
Match node CaseActivity
Tampilkan hasilnya dengan return
```

Gambar 3.34 Pseudo code Hasil proses model

Penjelasan:

Pada *query* ini match akan memanggil node CaseActivity yaitu node utama dari process model yang kita punya, lalu dari program neo4j akan menampilkan pula relationship yang dipunyai node tersebut secara otomatis. Return akan mengembalikan hasil dari match dan menampilkan hasilnya pada layar.

(Halaman Ini Sengaja Dkosongkan)

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan perangkat lunak. Di dalamnya mencakup proses penerapan dan pengimplementasian algoritma yang telah dibahas sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi dari tugas akhir dijelaskan pada Tabel 4.1.

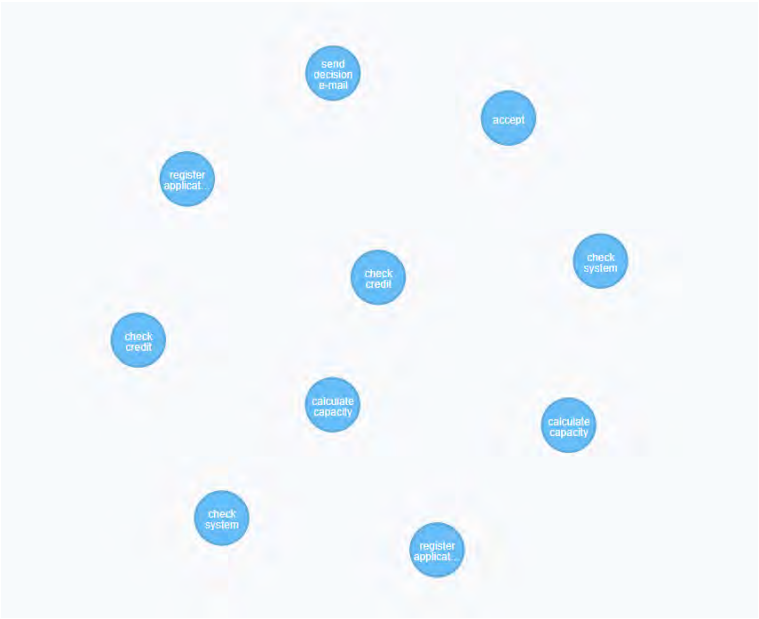
Tabel 4-1 Lingkungan Implementasi Perangkat Lunak

Perangkat Keras	Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz (4 CPUs), ~2.7GHz Memory: 8 GB
Perangkat Lunak	Sistem Operasi: Windows 10 Home Single Language 64-bit Perangkat Pengembang: Neo4J,ProM

4.2 Implementasi Metode

Implementasi terdapat dua macam yakni implementasi kasus penggunaan dan antarmuka, untuk antarmuka dilakukan screenshot terhadap hasil-hasil *query* dan antarmuka program Neo4J dan Prom. Screen shoot yang diambil dianggap mewakili dari keseluruhan yang mana terdapat pada **Lampiran 1.**

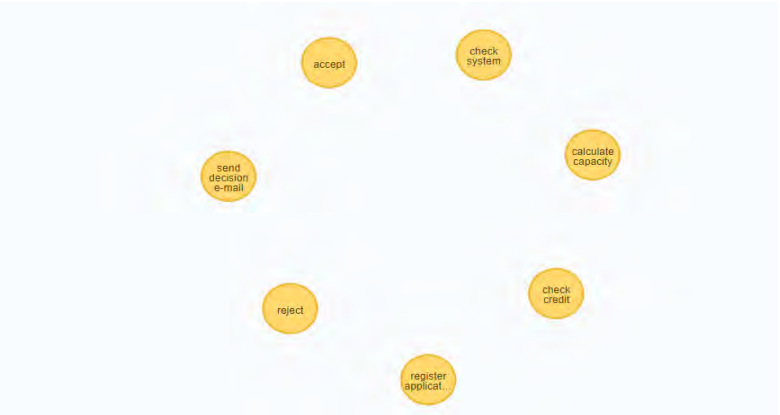
4.2.1 Implementasi Hasil *query* Load and *Create* Node Activity



Gambar 4.1 Tampilan hasil query Node activity

Langkah pertama pada metode yang dijalankan adalah dengan memasukkan data ke dalam database dan membuat node Activity terlebih dahulu, node Activity adalah representasi setiap data yang dimasukkan ke graph database. Gambar 4.1 adalah sebagian hasil dari node Activity yang dibuat dan menjadi contoh visualisasi dari data yang masuk ke graph database.

4.2.2 Implementasi Hasil *query* Load and *Create* node CaseActivity



Gambar 4.2 Tampilan hasil *query* Node CaseActivity

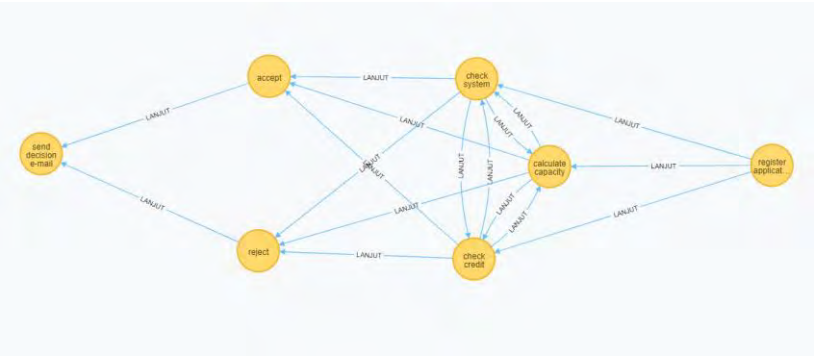
Pada langkah selanjutnya kita akan menggabungkan node Activity yang tadi dibuat menjadi node-node CaseActivity, pada Gambar 4.2 dapat dilihat sebagian node CaseActivity, setiap node CaseActivity merupakan Aktifitas/event yang ada pada seluruh data, jadi setiap aktifitas menghasilkan 1 node CaseActivity walaupun berbeda CaseId atau Waktu, selama aktifitas yang dilakukan sama maka akan menjadi 1 node CaseActivity.

4.2.3 Implementasi hasil *Query relation* antar nodes Activity dengan metode linked *list*

Gambar 4.3 Tampilan hasil Query relation pada nodes Activity

Selanjutnya adalah membuat relasi antar node Activity, pembuatan relasi ini menggunakan metode *linked list*, jadi kita akan mengambil semua node Activity dan akan mengubahnya menjadi *list*, lalu akan dibuat relasi sesuai dengan CaseId yang sama (Clustering), contoh hasil dari pembuatan relasi dapat dilihat pada gambar 4.3, pada gambar tersebut hanya sebagian relasi yang dapat diperlihatkan, tapi pada dasarnya semua node Activity minimal mempunyai 1 relasi dengan node Activity lainnya.

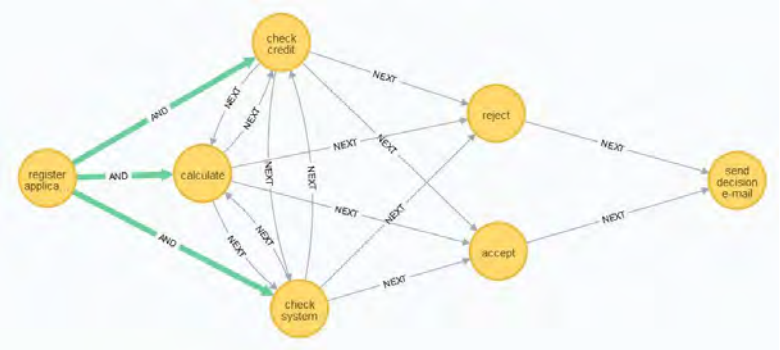
4.2.4 Implementasi hasil *Query merging relation* antar nodes CaseActivity dengan metode linked *list*



Gambar 4.4 Tampilan hasil Query relation antar node CaseActivity

Pada tahap selanjutnya kita akan membuat dasar dari graph proses model yang akan dibuat yaitu adalah relasi antar node CaseActivity, sama dengan pembuatan relasi node Activity, pembuatan relasi dengan menggunakan metode linked *list*, bedanya kita akan menggabungkan relasi-relasi yang melewati aktifitas yang sama menjadi 1 alur, pada gambar 4.4 kita bisa mengetahui semua alur yang dapat dilewati proses, gambar 4.4 juga menampilkan semua relasi antar node CaseActivity yang digabungkan.

4.2.5 Implementasi Tampilan *query* workflow pattern 'AND'

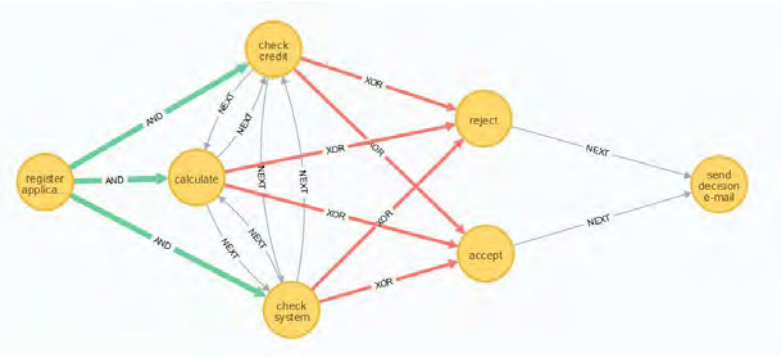


Gambar 4.5 Tampilan hasil Query WorkflowPattern 'AND'

Setelah semua node CaseActivity mempunyai relasi maka kita akan mencari dan bila ada membuat *control flow pattern,* yang pertama kita cari adalah *split pattern*, pada langkah ini kita akan mencari *And Split* yaitu saat alur dari proses terbagi menjadi 2 atau lebih dan semua cabang alur dijalankan, seperti pada gambar 4.4 salah satu node dimana alur terpisah menjadi 2 atau lebih adalah pada node 'register application', setelah dilakukan pencarian dengan metode yang telah dibahas pada bab 3 , ditemukan pada node 'register application' melakukan *And Split,* maka akan kita buat relasi *And* antara node 'register application' dengan node 'check system', 'calculate capacity', dan 'check

system'. Hasil dari *query* ini dapat dilihat pada gambar 4.5 dimana node 'register application' mempunyai relasi *And*.

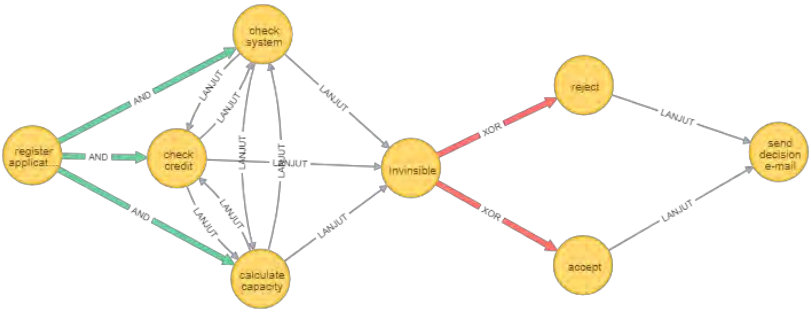
4.2.6 Implementasi Tampilan *Query* workflow pattern 'XOR'



Gambar 4.6 Tampilan hasil Query workflow pattern 'XOR'

Selanjutnya kita mencari *split pattern* lainnya yaitu *Xor Split,* split ini terjadi dimana alur terbagi menjadi 2 atau lebih tapi hanya 1 cabang yang dijalankan. Setelah dilakukan *query* untuk mencari *pattern Xor Split* ditemukan percabangan / *split* pada node 'check system', 'calculate capacity', dan 'check system' melakukan *Xor Split* dengan node 'accept' dan 'reject', hasil dari *query* dapat dilihat pada gambar 4.6 , panah merah mewakili *pattern Xor*.

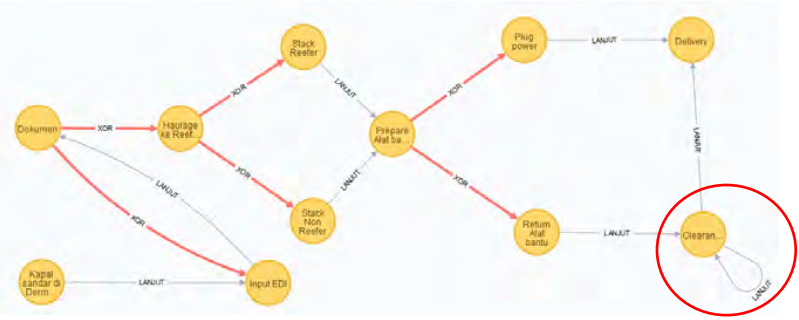
4.2.7 Implementasi Tampilan *Query* pembentukan node Invisible pada kasus Invisible Switch (Optional)

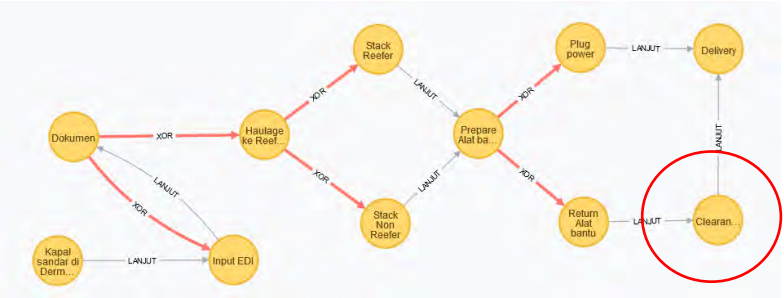


Gambar 4.7 Tampilan hasil graph setelah dilakukan query untuk menemukan Invisible Switch

Pada langkah selanjutnya akan dicari salah satu *workflow pattern* yaitu *Invisible* salah satunya yaitu *Invisible Switch*, dimana saat node yang menjadi target relasi *Split* langsung melakukan *Split* lagi baik dengan *pattern* yang sama maupun berbeda. Pada kondisi ini kita harus memasukkan data atau node kedalam *Invisible switch* agar tidak merusak proses atau alur. Pada gambar 4.7 kita dapat melihat hasil *query* invisible dimana yang sebelumnya node 'check system' adalah target *split* tetapi melakukan *split* lagi menuju node 'accept' dan 'reject', maka sebaiknya dimasukkan node 'Invisible' antara node 'check system' dan node 'accept' dan 'reject'.

4.2.8 Pembersihan *Single Loop Pattern*

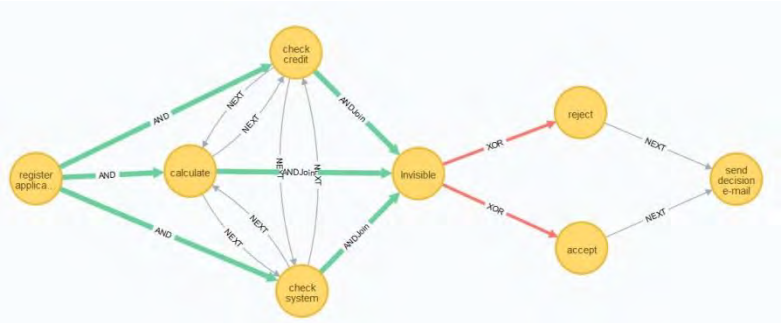




Gambar 4.8 Tampilan hasil pembersihan Single Loop Pattern

Pada tahap ini kita akan membersihkan *single loop pattern* yaitu dimana proses melakukan aktifitas secara berulang , biasanya dibarengi dengan penentuan keputusan saat aktifitas tersebut dijalankan, pada gambar 4.8 diperlihatkan perubahan graph dengan *single loop pattern*. Pada gambar 4.8 data input berubah dengan gambar lain karena data input yang pertama tidak memiliki *single loop pattern*.

4.2.9 Implementasi Tampilan *Query WorkflowPattern 'AND-Join'*

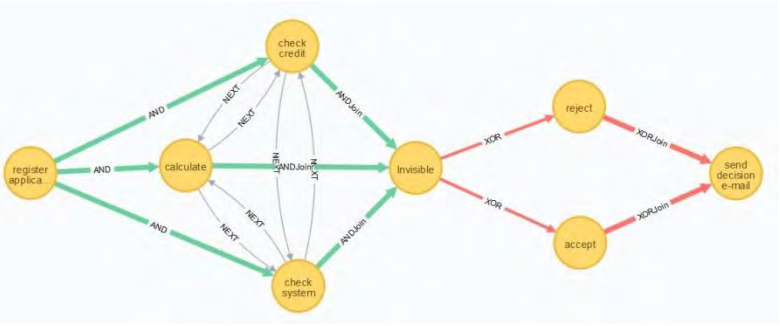


Gambar 4.9 Tampilan hasil Query Workflow Pattern 'AND- Join'

Setelah menghilangkan *single loop pattern* maka akan dicari dan dibuat *control flow pattern Join,* dan yang akan dilakukan pada

langkah ini adalah mencari dan membuat *And join pattern*, yaitu saat 2 atau lebih cabang yang melakukan *And Split* bergabung menjadi 1 cabang. Seperti pada gambar 4.9 dimana setelah melakukan *And Split* akan bergabung pada node 'Invisible'

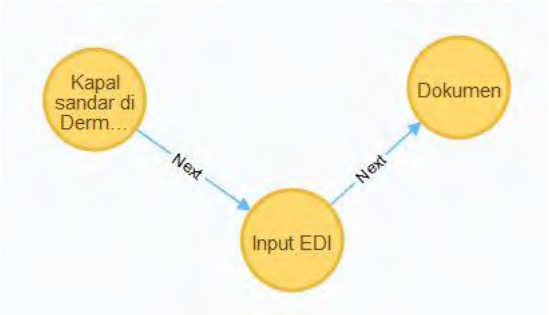
4.2.10 Implementasi Tampilan *Query WorkflowPattern 'XOR-Join'*



Gambar 4.10 Tampilan hasil Query Workflow Pattern 'XOR-Join'

Langkah selanjutnya adalah mencari dan membuat *Xor Join pattern*, yaitu saat 2 atau lebih cabang bergabung menjadi 1 cabang, tetapi berbeda dengan *And Join* dimana kita harus menunggu proses dari semua cabang , pada penggabungan ini dilakukan saat satu proses sampai. Contoh nya pada gambar 4.10 dimana node 'accept' dan 'reject' bergabung menjadi 1 pada node 'Send decision email', node 'send decision email' akan berjalan saat salah satu dari kedua node sebelumnya selesai dijalankan.

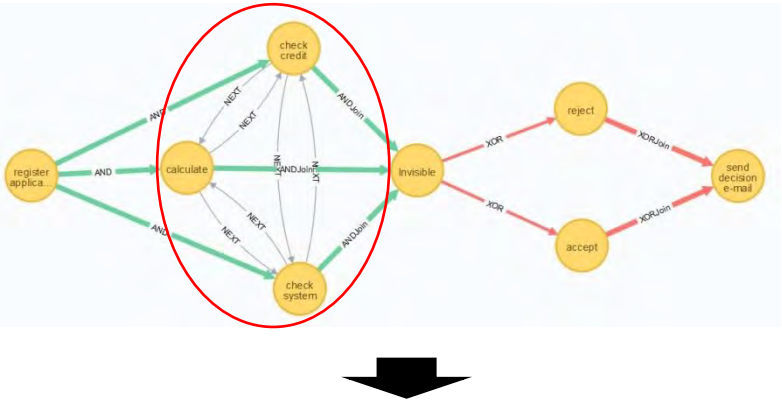
4.2.11 Implementasi Tampilan hasil *Query Sequence* pada node CaseActivity

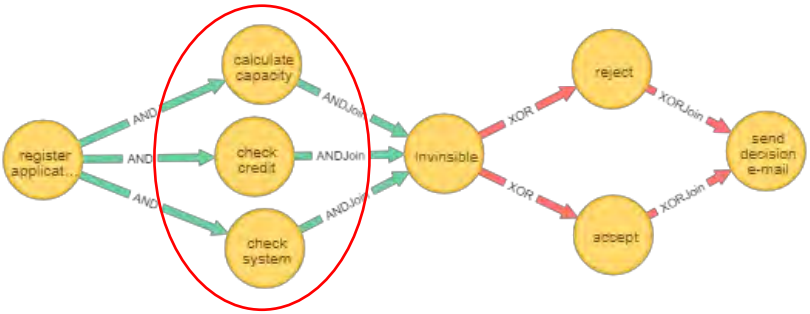


Gambar 4.11 Tampilan hasil Query Sequence pada node CaseActivity

Selanjutnya adalah salah satu pattern yang paling umum pada proses model yaitu *sequence pattern*, yaitu saat satu proses berjalan tanpa memecah proses menjadi beberapa cabang, jadi hanya berjalan lurus. Seperti pada gambar 4.11 mulai node awal berjalan terus sampai node akhir. Pada node 4.11 data input yang digunakan sama seperti gambar 4.8 ini dikarenakan pada data input yang dipakai oleh implementasi lainnya tidak memiliki *sequence pattern*.

4.2.12 Implementasi Tampilan *Query* Pembersihan Graph (Optional)

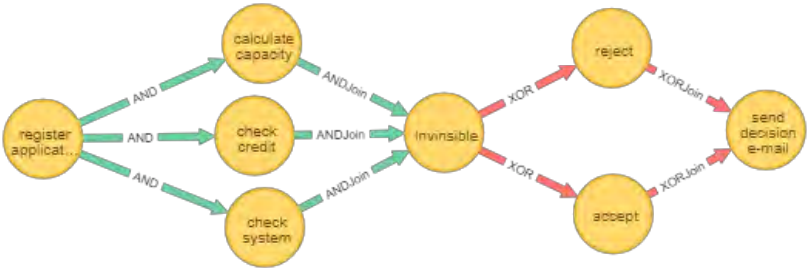




Gambar 4.12 Tampilan hasil graph setelah dilakukan query pembersihan parallelism

Langkah selanjutnya adalah merapikan graph salah satunya dengan cara menghilangkan *parallelism* yang telah digantikan oleh *And Split,* seperti pada gambar 4.12 sebenarnya secara fungsional tidak akan berpengaruh tetapi bila secara visual akan lebih memudahkan analis atau orang umum mengetahui alur kerja proses yang dibuat pada graph tersebut.

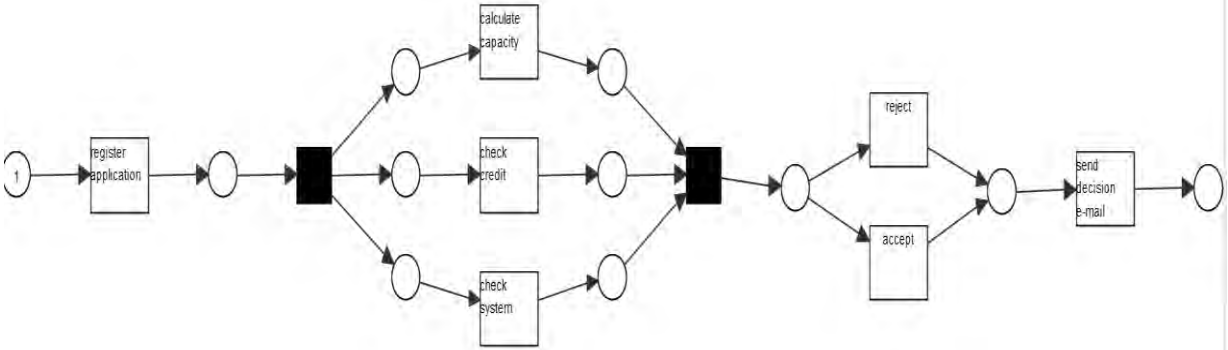
4.2.13 Implementasi Tampilan hasil Graph Final



Gambar 4.13 Tampilan graph Final hasil Neo4J

Gambar 4.13 adalah hasil akhir dari graph yang telah kita buat dengan metode yang di usulkan, terlihat seluruh node CaseActivity telah berelasi dengan node CaseActivity lainnya.

4.2.14 Implementasi Tampilan Inductive miner pada ProM



Gambar 4.13 Tampilan graph Final hasil Inductive Miner pada program ProM

Pada langkah terakhir kita akan membuat proses model dalam bentuk graph tetapi melalui program Prom6 dengan algoritma yang sama yaitu *Inductive Miner* dan dalam bentuk *petriNets.*

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan dijelaskan mengenai rangkaian uji coba dan evaluasi yang dilakukan. Proses pengujian dilakukan menggunakan metode yang telah dijelaskan sebelumnya, akan ada 2 kasus event logs yang akan diujicoba yaitu "Artificial-LoanProcess" dan juga "EventLogPelabuhan".

5.1 Lingkungan Uji Coba

Lingkungan pelaksanaan uji coba meliputi perangkat keras dan perangkat lunak yang akan digunakan pada sistem ini. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam rangka uji coba perangkat lunak ini dicantumkan pada Tabel 5.1 dan Tabel 5.2.

Tabel 5-1 Lingkungan Ujicoba Perangkat Lunak (kasus 1)

Perangkat Keras	Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz (4 CPUs), ~2.7GHz Memory: 8 GB
Perangkat Lunak	Sistem Operasi: Windows 10 Home Single Language 64-bit Perangkat Pengembang: Neo4J,ProM

Tabel 5-2 Lingkungan Ujicoba Perangkat Lunak (kasus 2)

Perangkat Keras	Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz (4 CPUs), ~2.7GHz Memory: 8 GB
Perangkat Lunak	Sistem Operasi: Windows 10 Home Single Language 64-bit Perangkat Pengembang: Neo4J,ProM

5.2 Data Studi Kasus

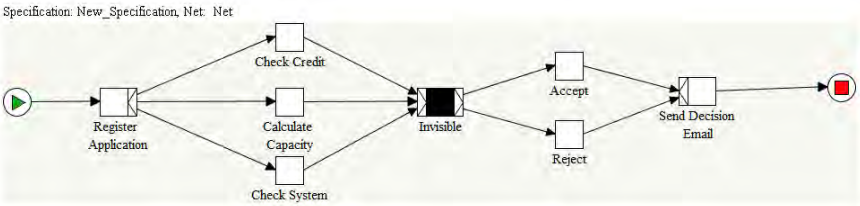
5.2.1 Data Studi kasus Peminjaman uang pada bank (Artificial-LoanProcess)

Data event log "Artificial-LoanProcess" memiliki 100 Case dan 7 aktifitas, potongan dari event log "Artificial-LoanProcess" dapat dilihat pada tabel 5.3 dan process model dari event log ini dapat dilihat pada Gambar 5.1.

Tabel 5-3 Potongan Event Log Artificial Loan Process

case	event	completeTime
trace 0	register application	4/16/2013 10:08
trace 0	check credit	4/16/2013 10:16
trace 0	calculate capacity	4/16/2013 10:16
trace 0	check system	4/16/2013 10:20
trace 0	accept	4/16/2013 10:21
trace 0	send decision e-mail	4/16/2013 10:26
trace 1	register application	4/16/2013 10:10
trace 1	check credit	4/16/2013 10:16
trace 1	calculate capacity	4/16/2013 10:16
trace 1	check system	4/16/2013 10:20
trace 1	accept	4/16/2013 10:24
trace 1	send decision e-mail	4/16/2013 10:29
trace 2	register application	4/16/2013 10:15
trace 2	check credit	4/16/2013 10:22
trace 2	calculate capacity	4/16/2013 10:23
trace 2	check system	4/16/2013 10:29
trace 2	accept	4/16/2013 10:37
trace 2	send decision e-mail	4/16/2013 10:44

trace 97	register application	4/16/2013 18:17
trace 97	calculate capacity	4/16/2013 18:23
trace 97	check system	4/16/2013 18:27
trace 97	check credit	4/16/2013 18:33
trace 97	reject	4/16/2013 18:43
trace 97	send decision e-mail	4/16/2013 18:51
trace 98	register application	4/16/2013 18:22
trace 98	calculate capacity	4/16/2013 18:29
trace 98	check system	4/16/2013 18:34
trace 98	check credit	4/16/2013 18:37
trace 98	reject	4/16/2013 18:43
trace 98	send decision e-mail	4/16/2013 18:50
trace 99	register application	4/16/2013 18:30
trace 99	calculate capacity	4/16/2013 18:39
trace 99	check credit	4/16/2013 18:48
trace 99	reject	4/16/2013 18:50
trace 99	send decision e-mail	4/16/2013 18:55



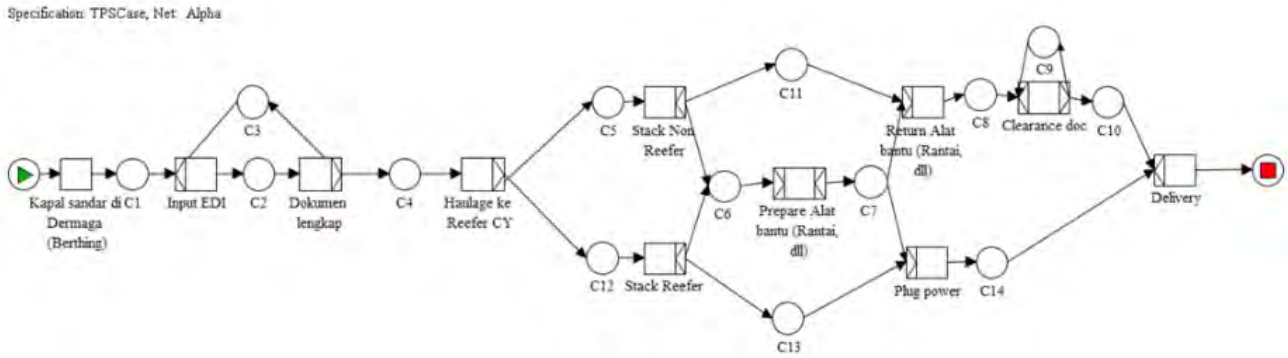
Gambar 5.1 Process model event log "Artificial Loan Process"

5.2.2 Data Studi kasus Dwelling time Pelabuhan (EventLogPelabuhan)

Data event log "EventLogPelabuhan" memiliki 40 Case dan 11 aktifitas, potongan dari event log "EventLogPelabuhan" dapat dilihat pada tabel 5.4 dan process model dari event log ini dapat dilihat pada Gambar 5.2.

Tabel 5-4 Potongan Event Log pelabuhan

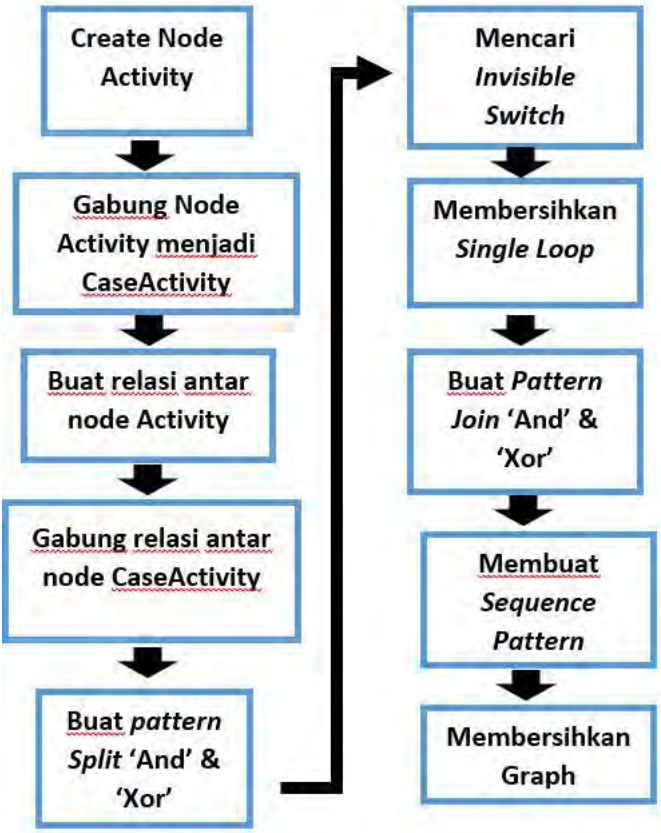
CaseID	Activity	StartTime	EndTime	Resources	ProductType	Cost
PP1	Kapal sandar di Dermaga (Berthing)	21/08/2017 10:00	21/08/2017 11:00	Wharf Supervisor		
PP1	Input EDI	21/08/2017 11:10	21/08/2017 11:30	Wharf Admin		
PP1	Dokumen lengkap	21/08/2017 11:35	21/08/2017 11:55	Wharf Admin		
PP1	Haulage ke Reefer CY	21/08/2017 12:00	21/08/2017 17:00	Trailer Driver	Durable Goods	
PP1	Stack Non Reefer	21/08/2017 17:30	21/08/2017 20:30	RTG Operator		
PP1	Prepare Alat bantu (Rantai, dll)	21/08/2017 20:35	21/08/2017 22:35	Gear Store		
PP1	Return Alat bantu (Rantai, dll)	21/08/2017 22:40	22/08/2017 00:40	Yard Supervisor		
PP1	Clearance doc	22/08/2017 00:45	22/08/2017 01:30	Customer Service		
PP1	Delivery	22/08/2017 01:35	22/08/2017 05:35	Shipper		
PP2	Kapal sandar di Dermaga (Berthing)	21/08/2017 10:00	21/08/2017 11:00	Wharf Supervisor		
PP2	Input EDI	21/08/2017 11:10	21/08/2017 11:30	Wharf Admin		
PP2	Dokumen lengkap	21/08/2017 11:35	21/08/2017 11:55	Wharf Admin		
PP2	Haulage ke Reefer CY	21/08/2017 12:00	21/08/2017 17:00	Trailer Driver	Durable Goods	
PP2	Stack Non Reefer	21/08/2017 17:30	21/08/2017 20:30	RTG Operator		
PP2	Prepare Alat bantu (Rantai, dll)	21/08/2017 20:35	21/08/2017 22:35	Gear Store		
PP2	Return Alat bantu (Rantai, dll)	21/08/2017 22:40	22/08/2017 00:40	Yard Supervisor		
PP2	Clearance doc	22/08/2017 00:45	22/08/2017 01:30	Customer Service		
PP2	Delivery	22/08/2017 01:35	22/08/2017 05:35	Shipper		
PP49	Kapal sandar di Dermaga (Berthing)	23/08/2017 13:15	23/08/2017 13:175	Wharf Supervisor		
PP49	Input EDI	23/08/2017 13:30	23/08/2017 13:200	Wharf Admin		
PP49	Dokumen lengkap	23/08/2017 13:55	23/08/2017 14:18	Wharf Admin		
PP49	Haulage ke Reefer CY	23/08/2017 14:23	24/08/2017 00:23	Trailer Driver		
PP49	Stack Reefer	24/08/2017 00:28	24/08/2017 02:28	RTG Operator	Durable Non Goods	
PP49	Prepare Alat bantu (Rantai, dll)	24/08/2017 02:33	24/08/2017 04:90	Gear Store		
PP49	Plug power	24/08/2017 04:95	24/08/2017 04:100	Reefer Operator		
PP49	Delivery	24/08/2017 04:105	24/08/2017 07:48	Shipper		
PP50	Kapal sandar di Dermaga (Berthing)	23/08/2017 06:53	23/08/2017 07:10	Wharf Supervisor		
PP50	Input EDI	23/08/2017 07:15	23/08/2017 07:25	Wharf Admin		
PP50	Dokumen lengkap	23/08/2017 07:30	23/08/2017 07:50	Wharf Admin		
PP50	Haulage ke Reefer CY	23/08/2017 07:55	23/08/2017 12:58	Trailer Driver		
PP50	Stack Reefer	23/08/2017 13:03	23/08/2017 16:03	RTG Operator	Durable Non Goods	
PP50	Prepare Alat bantu (Rantai, dll)	23/08/2017 16:08	23/08/2017 18:65	Gear Store		
PP50	Plug power	23/08/2017 18:70	23/08/2017 18:75	Reefer Operator		
PP50	Delivery	23/08/2017 18:80	23/08/2017 21:23	Shipper		



Gambar 5.2 Process model event log "EventLogPelabuhan"

5.3 Pengujian Metode pada Studi kasus

Pengujian dilakukan untuk menguji metode yang telah dijelaskan pada bab 4, akan dilakukan setiap langkah nya beserta hasil yang akan ditunjukkan dengan metode gambar setelah itu akan dicoba pula dari input yang sama dibangun process model melalui program Prom 6 dengan Algoritma *Inductive Miner.*



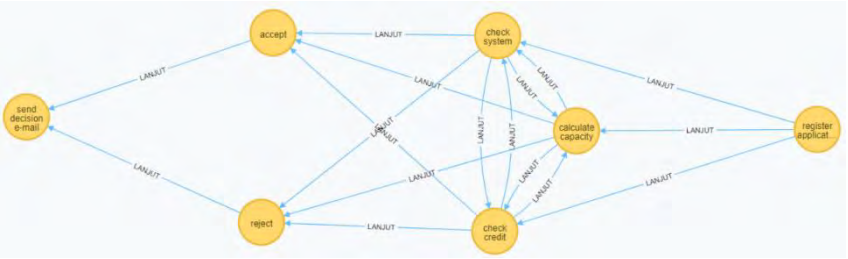
Gambar 5.3 Flow Diagram metode yang dipakai

5.3.1 Pengujian metode pada studi kasus Peminjaman Bank

- *Load event log* "Artificial Loan Process", lalu *Create* Node Activity dan CaseActivity

Gambar 5.4 Node Activity dan Case Activity kasus 1

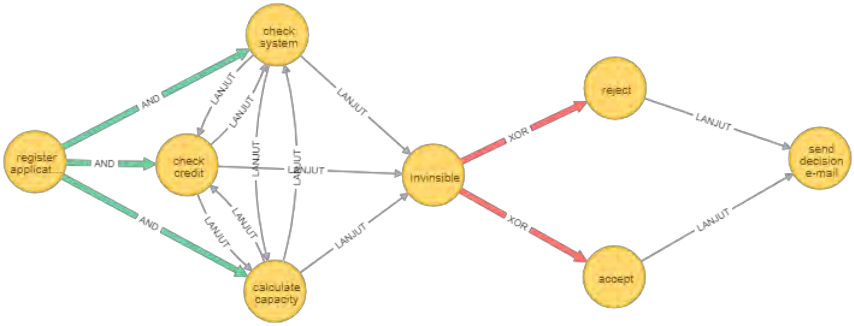
- Buat relasi antar node Activity dan Case Activity menggunakan metode *Linked list*



Gambar 5.5 Node Activity dan CaseActivity dengan relasi

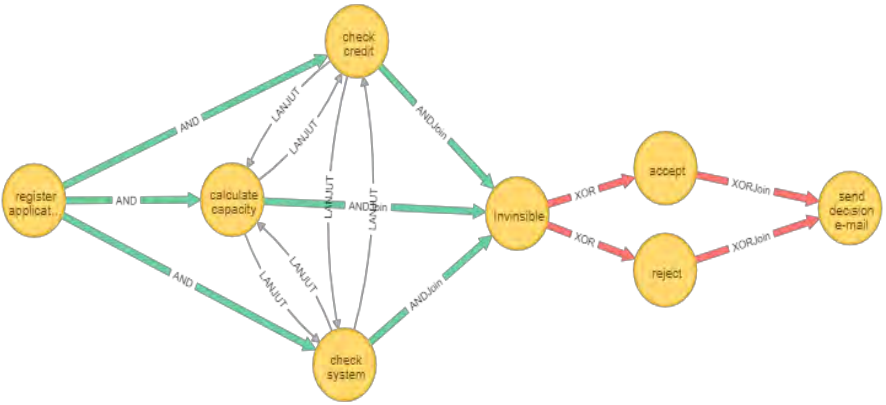
Metode linked *list* adalah mengambil Node dan diubah menjadi bentuk *list* , lalu membuat range *list* yang akan dipakai, disini kita akan membuat 2 range yaitu A=0–(x-1),(x adalah jumlah node yang ada pada *list*) dan yang kedua Antara B=1–x. Dengan begini akan terbuat 2 buah *list* yang nantinya akan dapat di linked dengan relasi Antara A dan B dan hasilnya bisa diamati pada gambar 5.5.

- Buat pattern Split And dan Split Xor
- Membuat dan mencari Invisible Switch (Optional, tidak pasti ada pada setiap proses model)



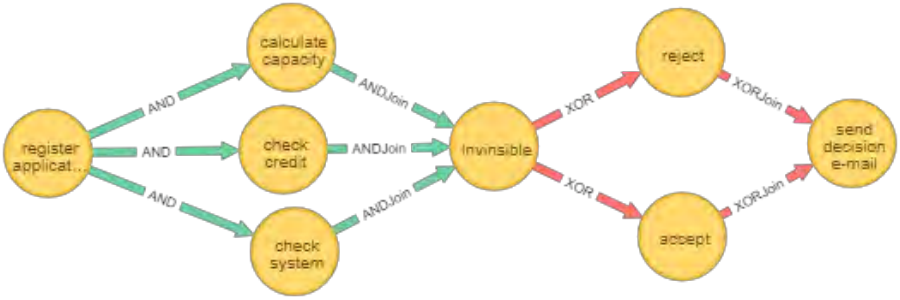
Gambar 5.6 Proses model setelah query Split Xor dan And, lalu query Invisible Switch

- Membersihkan dan membuang single Loop pada process model (Optional, tidak pasti ada pada setiap proses model)
- Membuat pattern Join And dan Join Xor



Gambar 5.7 Proses model setelah query Join And dan Join Xor

- Membuat pattern Sequence
- Membersihkan Graph bila dirasa perlu (yang dibersihkan pada data ini adalah Parallelism)

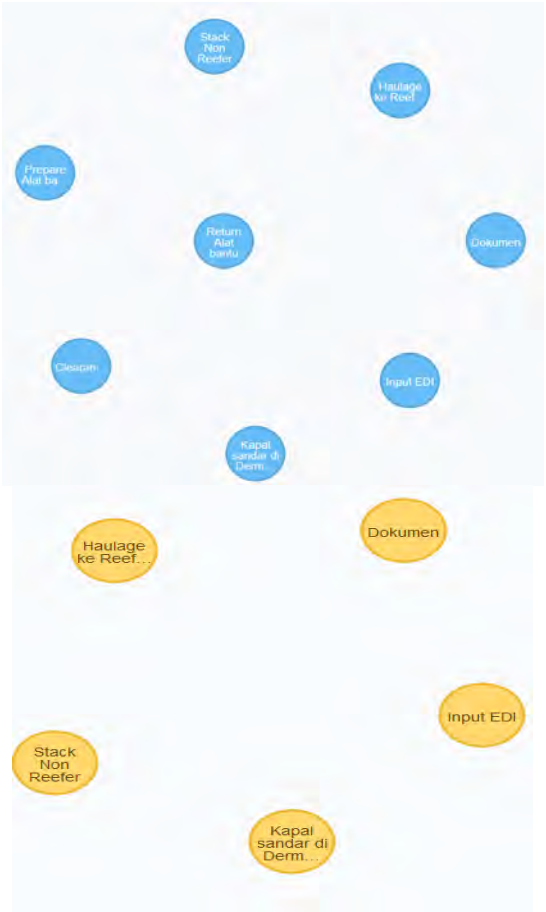


Gambar 5.8 Proses model setelah query pembersihan (Parallelism)

Proses model pada gambar 5.8 adalah hasil akhir dari proses model pada kasus 1 yaitu peminjaman uang pada bank.

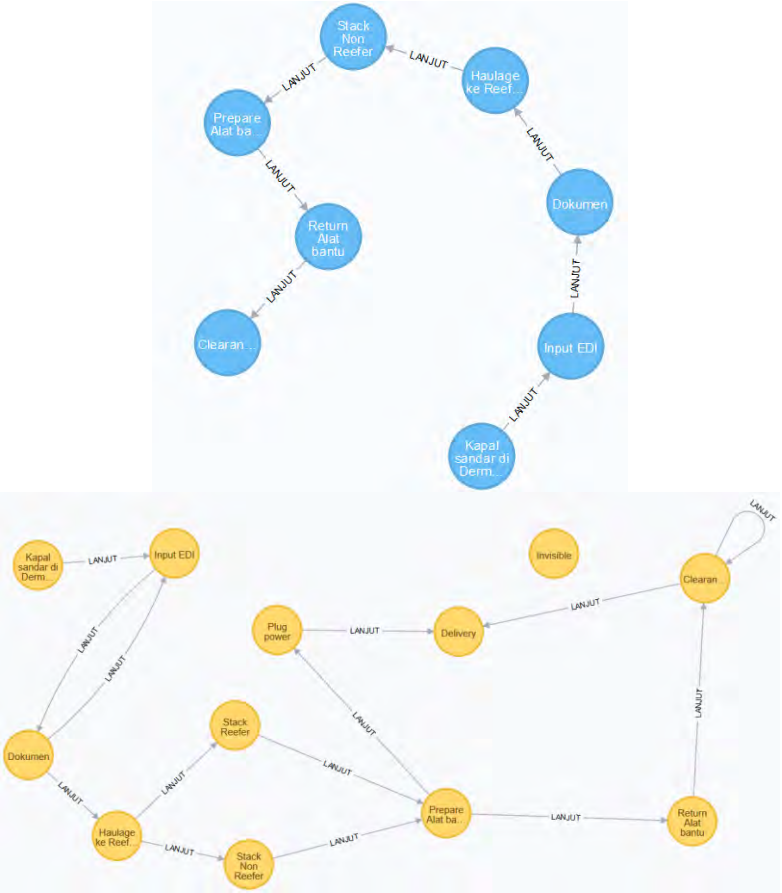
5.3.2 Pengujian metode pada studi kasus Pelabuhan

- Load event log "EventLogPelabuhan", lalu *Create* Node Activity dan CaseActivity



Gambar 5.9 Node Activity dan Case Activity kasus 2

- Buat relasi antar node Activity dan Case Activity menggunakan metode Linked *list*



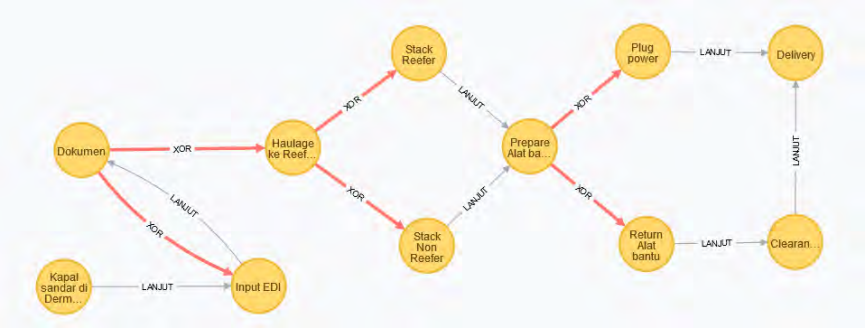
Gambar 5.10 Node Activity dan CaseActivity kasus 2 dengan relasi

Metode linked *list* adalah mengambil Node dan diubah menjadi bentuk *list* , lalu membuat range *list* yang akan dipakai, disini kita akan membuat 2 range yaitu A=0–(x-1),(x adalah jumlah node yang ada pada *list*) dan yang kedua Antara B=1–x. Dengan begini akan terbuat 2 buah *list* yang nantinya akan dapat di linked dengan relasi Antara A dan B dan hasilnya bisa diamati pada gambar 5.10.

- Buat pattern Split And dan Split Xor
- Membuat dan mencari Invisible Switch (Optional, tidak pasti ada pada setiap proses model)

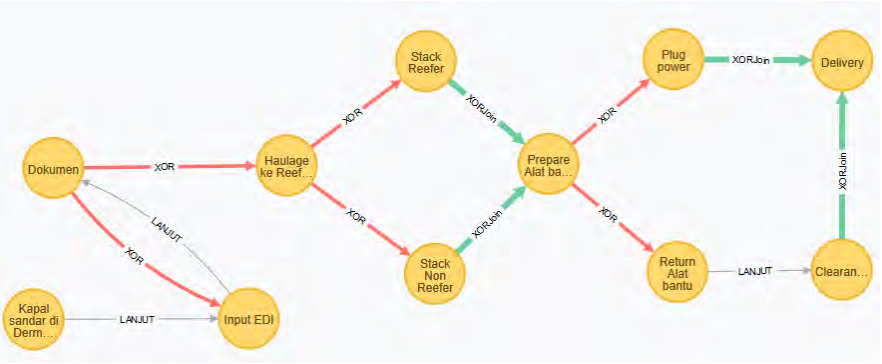
Gambar 5.11 Proses model setelah query Split Xor dan And

- Membersihkan dan membuang single Loop pada process model (Optional, tidak pasti ada pada setiap proses model)



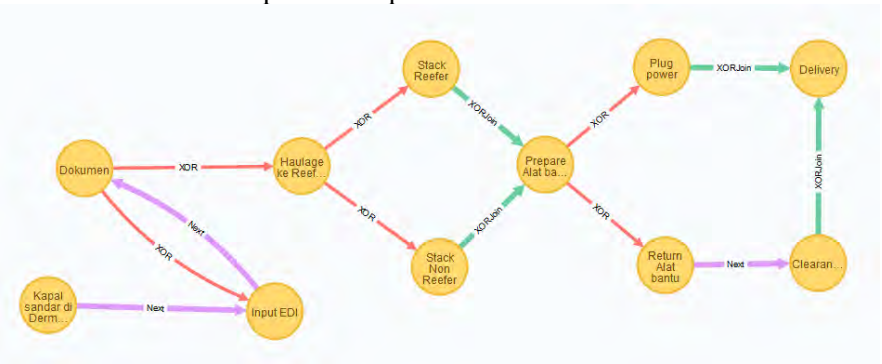
Gambar 5.12 Proses model setelah membuang single Loop

- Membuat pattern Join And dan Join Xor



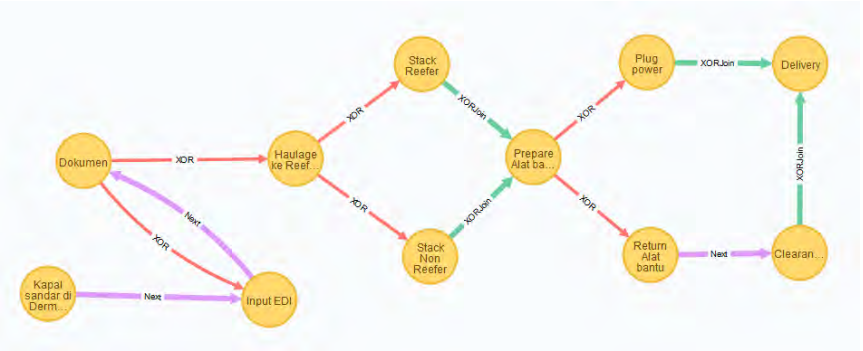
Gambar 5.13 Proses model setelah query Join And dan Join Xor

- Membuat pattern Sequence



Gambar 5.14 Proses model setelah query sequence pattern

- Membersihkan Graph bila dirasa perlu

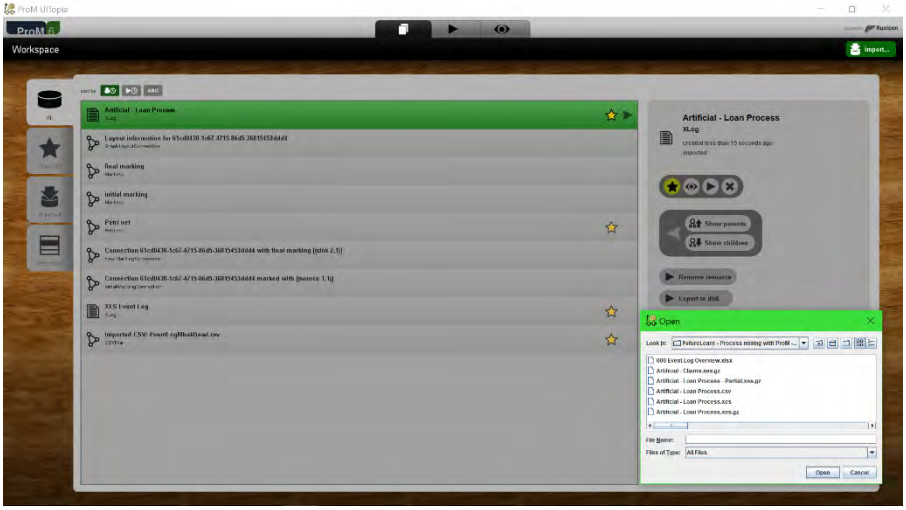


Gambar 5.15 Proses model final untuk kasus 2

Proses model pada gambar 5.15 adalah hasil akhir dari metode yang dilakukan pada kasus ke 2 (pelabuhan).

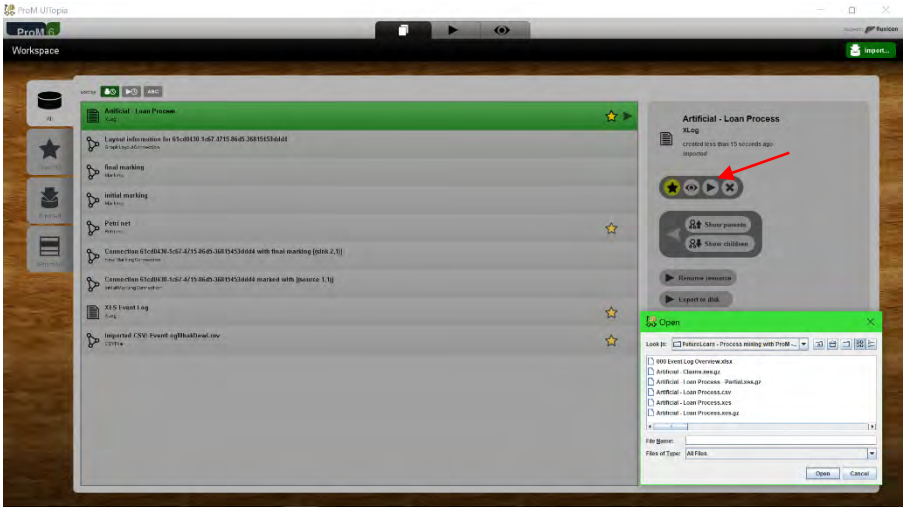
5.3.3 Pembuatan proses model melalui Prom pada Studi Kasus 1

- Buka program Prom 6.
- Import event log bisa dalam bentuk csv atau xes.gz ke dalam Prom 6.



Gambar 5.16 Window Prom6 dan window import file

- Pilih event log tersebut lalu use resource untuk diolah.

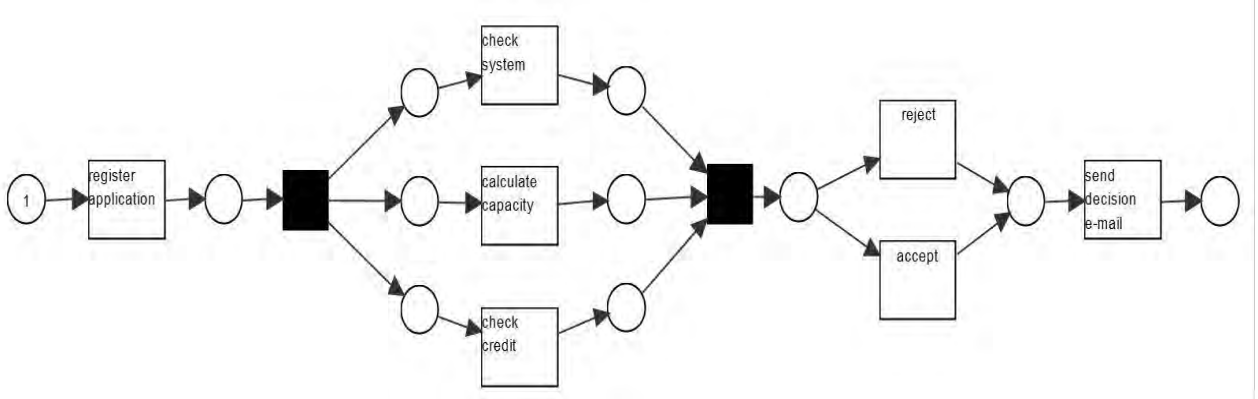


Gambar 5.17 Window Prom6 dan anak panah menuju tombol use resource

- Pilih '*mine petri net with inductive miner'* pada *Actions menu*

Gambar 5.18 Window action dalam prom 6

- Klik Start lalu pilih next bila ingin default.
- Hasil akan disajikan dalam bentuk petrinet oleh Prom 6

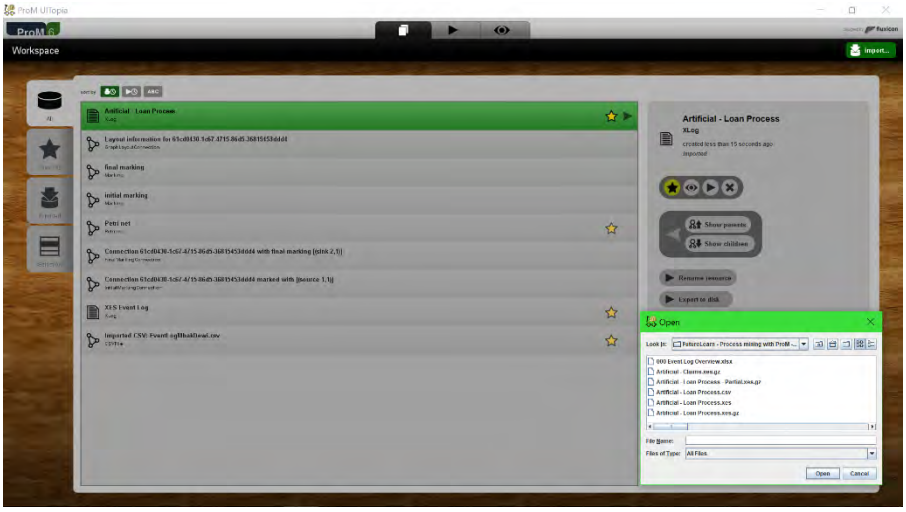


Gambar 5.19 Process model hasil Inductive Miner pada Prom 6

Hasil yang didapat berbentuk gambar (png / jpeg) sehingga sulit untuk diolah lebih lanjut.

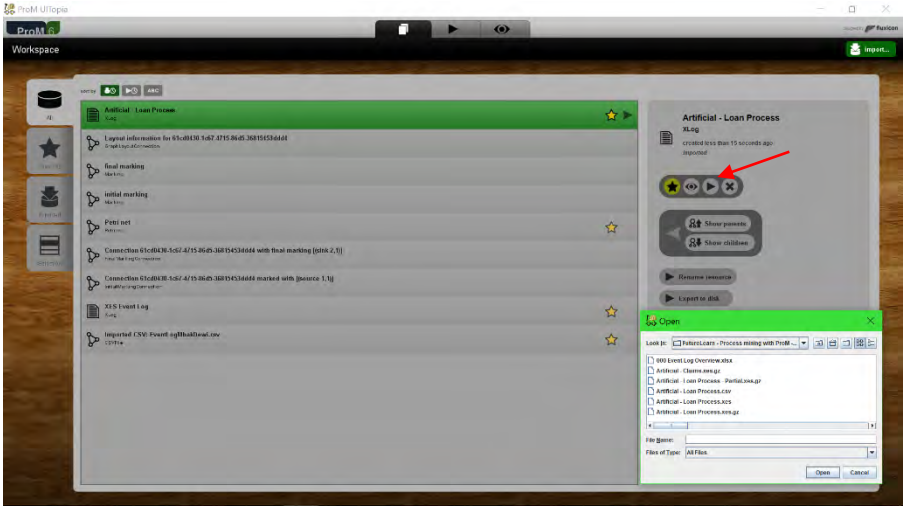
5.3.4 Pembuatan proses model melalui Prom pada Studi Kasus 2

- Buka program Prom 6
- Import event log bisa dalam bentuk csv atau xes.gz ke dalam Prom 6



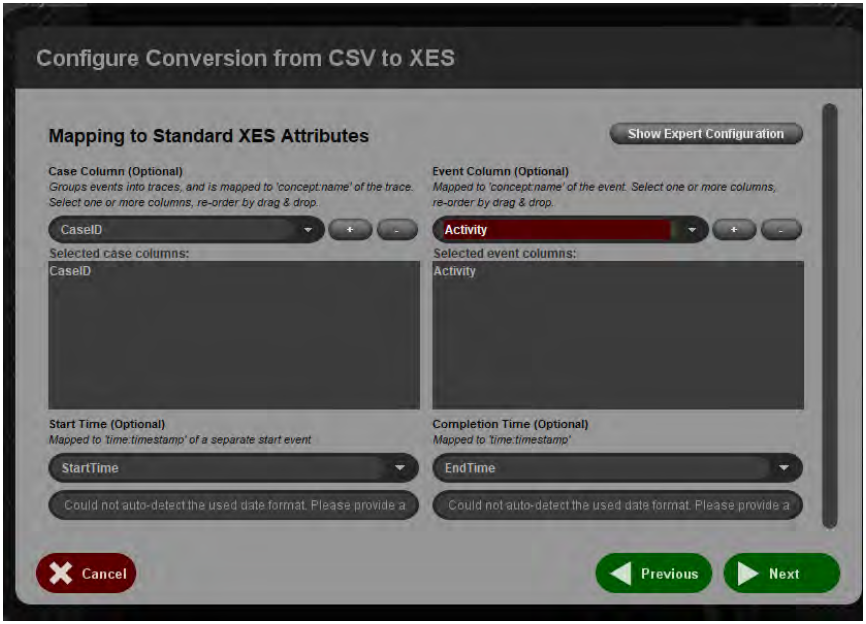
Gambar 5.20 Window Prom6 dan window import file

- Pilih event log tersebut lalu use resource untuk diolah



Gambar 5.21 Window Prom6 dan anak panah menuju tombol use resource

- Pilih Convert csv to Xes
- Pada parser menu bila dirasa benar pilih next
- Pada mapping menu pilih Case id untuk Case Column dan Activity untuk Event Column, lalu pilih next, lalu pilih finish.

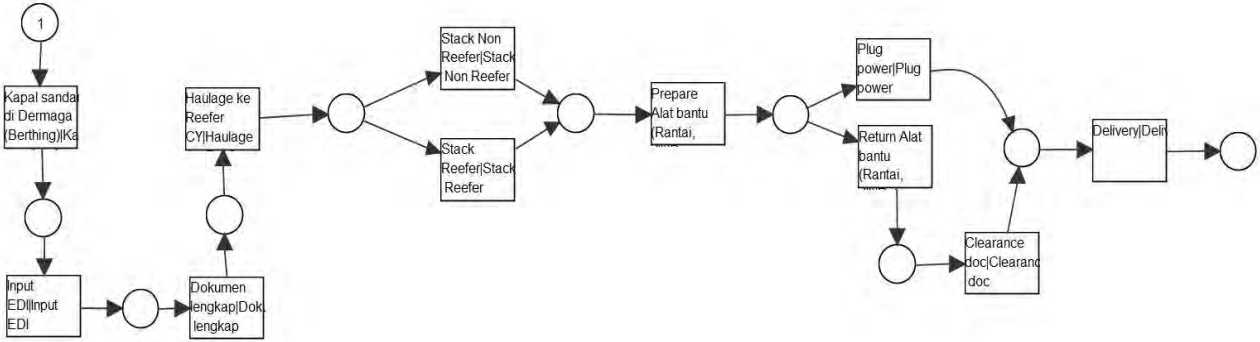


Gambar 5.22 Window mapping menu

Maka hasilnya akan menjadi bentuk xes dari file csv yang kita input sebelumnya, di prom 6 kebanyakan tipe file harus diubah ke bentuk xes atau xes.gz agar dapat diolah.

- Pilih resource yang telah didapat dari hasil convert, lalu klik use resource
- Pilih '*mine petri net with inductive miner'* pada *Actions menu*
- Klik Start

- Hasil akan disajikan dalam bentuk petrinet oleh Prom 6

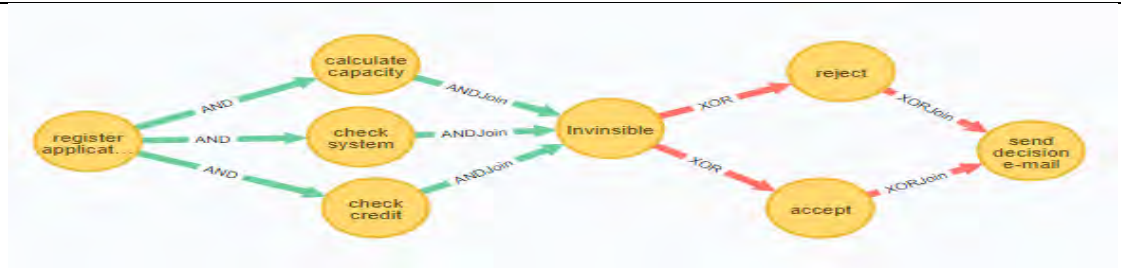
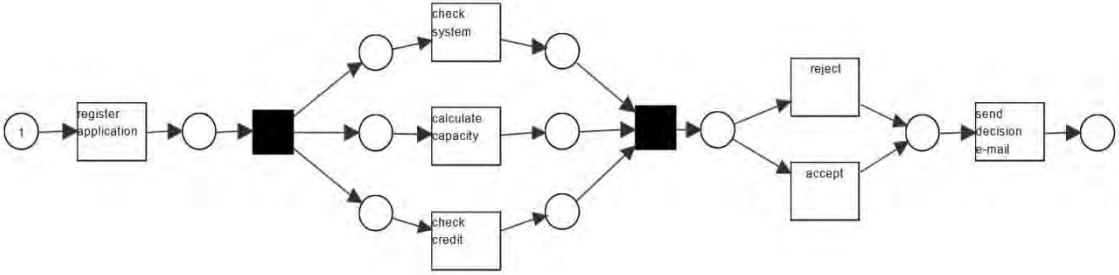


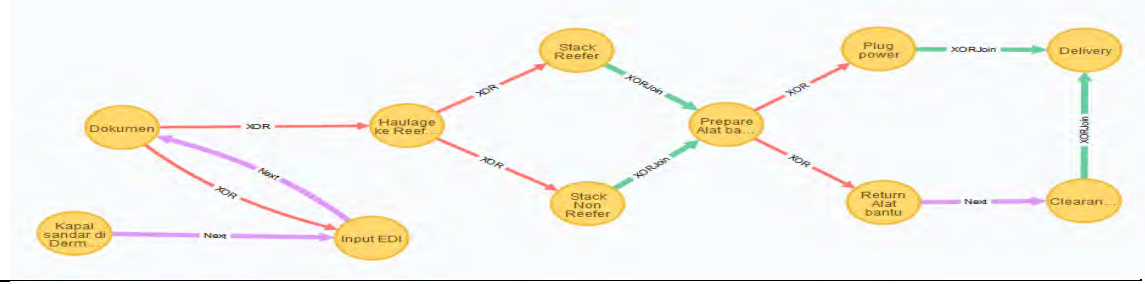
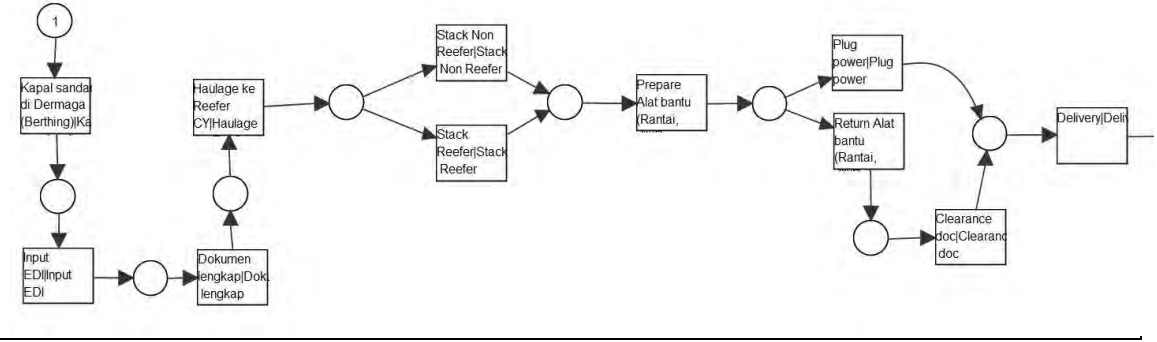
Gambar 5.23 Process model hasil Inductive Miner pada Prom 6

Hasil yang didapat berbentuk gambar (png / jpeg) sehingga sulit untuk diolah lebih lanjut.

5.4 Pembandingan hasil proses model Graph Database dan Prom

Tabel 5-5 Perbandingan hasil akhir Neo4J dan Prom 6

Arftificial Loan Process	
Neo4J	
Prom 6	

Eventlog Pelabuhan	
Neo4 J	
Prom 6	

5.4.1 Pembandingan struktur process model dari graph database(Neo4J) dan Prom

Tabel 5-6 Perbandingan struktur hasil proses model Neo4J dan Prom 6 kasus Artificial Loan process

Structure	Neo4J	Prom
Jumlah Node	8	9
Jumlah Relation	10	11
Functionality	Neo4J	Prom
Split 'And'	3	3
Split 'Xor'	2	2
Join 'And'	3	3
Join 'Xor'	2	2
Invisible Switch	1	1

Tabel 5-7 Pembandingan struktur hasil proses model Neo4J dan Prom 6 kasus Eventlog Pelabuhan

`	Neo4J	Prom
Jumlah Node	11	11
Jumlah Relation	13	12
Functionality	Neo4J	Prom
Split 'And'	0	0
Split 'Xor'	6	4
Join 'And'	0	0
Join 'Xor'	4	4

5.4.2 Hasil Uji Presisi

Selanjutnya akan dilakukan uji presisi Antara proses model yang dibangun pada Neo4j dengan dasar proses model yang dibangun di Prom.

Dengan Rumus: $\frac{\Delta N}{\Delta P}$

Keterangan:

ΔN : jumlah dari *Node* atau *relation* yang ada pada proses model Neo4J.

ΔP : jumlah dari seluruh *Node* atau *relation* yang digambarkan di proses model Prom6.

Tabel 5-8 Hasil Uji Presisi

Kasus 1	Neo4J	Prom6	Presisi
Node	8	9	0.88
Relation	10	11	0.90
Kasus 2	Neo4J	Prom6	Presisi
Node	11	11	1
Relation	13	12	1.08

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari tujuan pembuatan perangkat lunak dan hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, terdapat pula saran yang ditujukan untuk pengembangan perangkat lunak lebih lanjut.

6.1. Kesimpulan

Dalam proses pengerjaan tugas akhir mulai dari tahap analisis, desain, implementasi, hingga pengujian didapatkan kesimpulan sebagai berikut:

1. Metode dan *Query* yang diusulkan dapat membangun proses model dalam bentuk graph dengan cukup baik.
2. Dalam hasil uji pembandingan dengan Program Prom6 dapat dikatakan struktur proses model yang dihasilkan pada graph database hampir sama dengan sedikit perbedaan.
3. Perbedaan yang didapatkan karena beberapa hal :
    a. Salah satu invisible node pada kasus 1 masih belum dapat diketahui syaratnya.
    b. Algoritma mining yang dipakai adalah Inductive miner yang tidak dapat mendiscover Loop baik single loop maupun loop lain.
4. Hasil proses model yang didapatkan pada Neo4J masih dapat diolah dan dilihat detailnya, sehingga dapat digunakan lebih lanjut.

6.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang, berdasarkan pada hasil perancangan, implementasi dan uji coba yang telah dilakukan.

1. Dapat dibandingkan dengan Algoritma mining lain seperti Alpha dan Alpha ++.
2. Algoritma yang jadi referensi untuk pembuatan metode adalah Linked *list* dan Inductive miner, mungkin dapat dicoba dengan algoritma lain untuk membangun metode tersebut..
3. Agar mencoba Metode dengan lebih banyak test data dan lebih complex.
4. Membangun lebih banyak *query* untuk peningkatan performa dan fungsionalitas.

.

DAFTAR PUSTAKA

[1] J. L Whitten, Metode desain dan analysis sistem. Jogjakarta: Andi, 2017.

[2] "Guide to Business Process Modeling | Smartsheet", Smartsheet, 2017. [Online]. Available: https://www.smartsheet.com/beginners-guide-business-processmodeling. [Accessed: 14- Dec- 2017].

[3] "What is graph database? - Definition from WhatIs.com", WhatIs.com, 2017. [Online]. Available: http://whatis.techtarget.com/definition/graph-database. [Accessed: 21- Dec- 2017].

[4] "Why Graph Databases? - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2017. [Online]. Available: https://neo4j.com/why-graph-databases/. [Accessed: 14- Dec- 2017].

[5] W. Fan, X. Wang and Y. Wu, "Incremental graph pattern matching", ACM Transactions on Database Systems, vol. 38, no. 3, pp. 1-47, 2013.

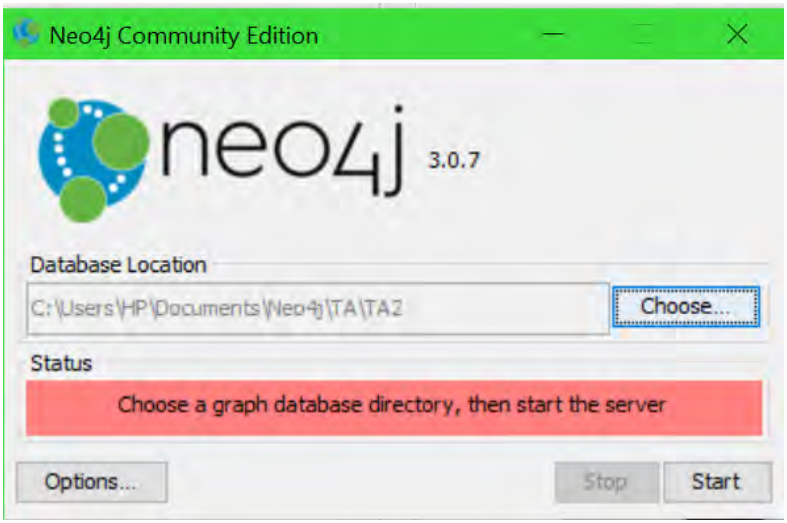
[6] J. Buijs, "Inductive miner - Introduction to Process Mining with ProM - Eindhoven University of Technology", FutureLearn, 2017. [Online]. Available: https://www.futurelearn.com/courses/process-mining/0/steps/15642. [Accessed: 14- Dec- 2017].

[7] "Workflow pattern", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Workflow_pattern. [Accessed: 21- Dec- 2017].

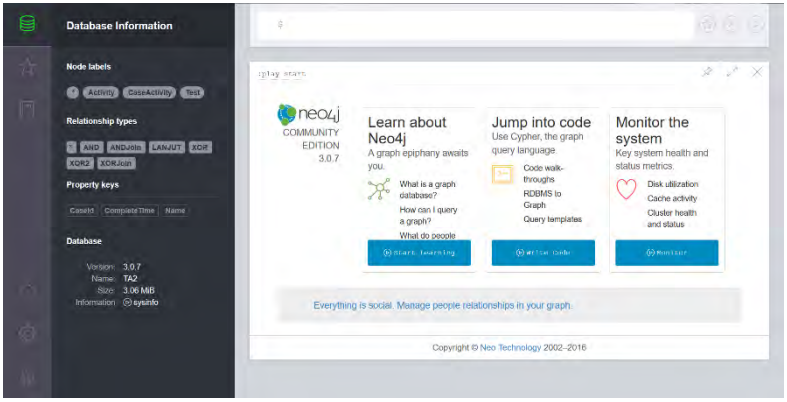
[8] N. Russell, Workflow control-flow patterns : a revised view. BPMcenter.org, 2006.

[9] "Workflow Patterns, Part II: Control and Flow of Task Execution", Ehrscience.com, 2017. [Online]. Available: http://ehrscience.com/2013/08/26/workflow-patterns-part-ii-control-and-flow-of-task-execution/. [Accessed: 21- Dec- 2017].

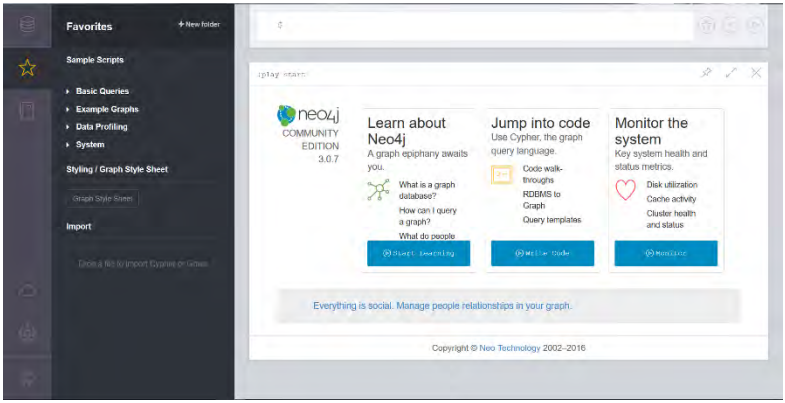
LAMPIRAN



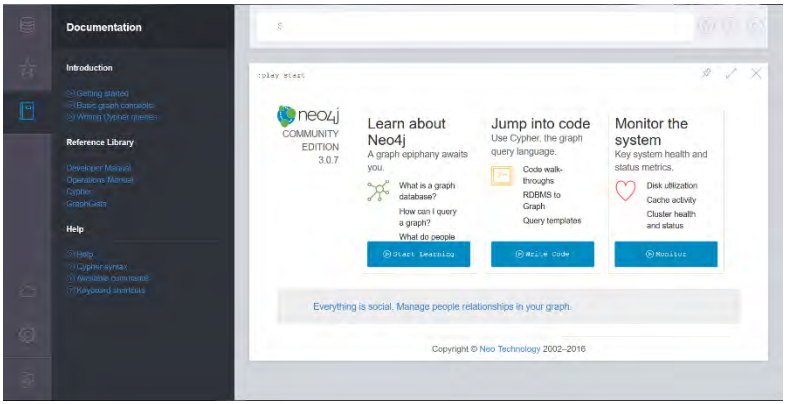
Tampilan awal Neo4J



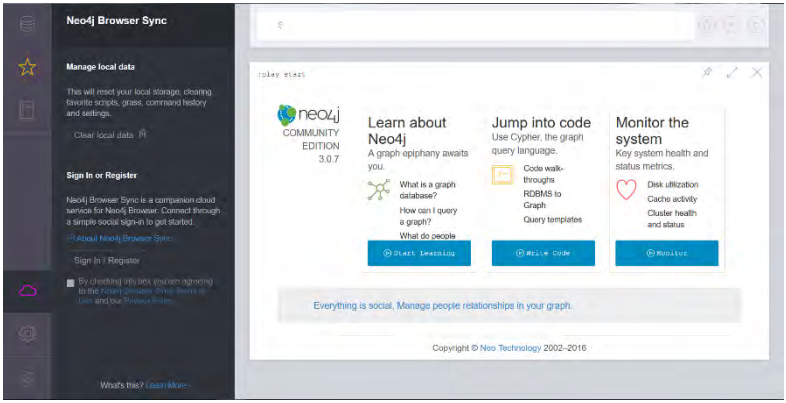
Browser neo4j Database information



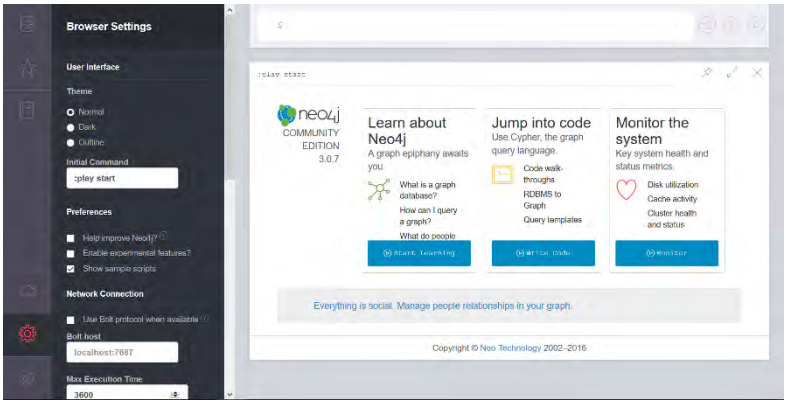
Browser neo4j Favorites



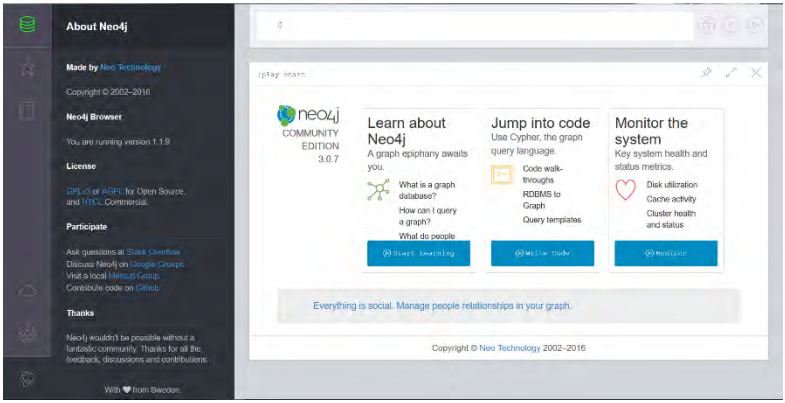
Browser neo4j Help dan Documentation



Browser Neo4J Sync



Neo4J Browser Setting



Browser Neo4J About

Hasil *query* relasi node activity melengkapi gambar 4.3

BIODATA PENULIS

Penulis dilahirkan di Madiun, 5 Juli 1995, merupakan anak pertama dari 2 bersaudara. Penulis telah menempuh pendidikan formal yaitu TK Ariska (1999-2001), SD Negeri 1 Sedati Sidoarjo(2001-2007), SMP Negeri 4 Waru Sidoarjo (2007-2010), SMA Hang Tuah 2 Sidoarjo (2010-2013), dan mahasiswa S1 Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya rumpun mata kuliah Manajemen Informasi (2013-2017). Selama menjadi mahasiswa . Penulis yang memiliki hobi bermain game online ,travelling, membaca. merupakan mahasiswa yang aktif dalam organisasi diantaranya, HMTC (staff hublu 2014-2015), Panitia Schematics (Staff keamanan dan perizinan 2013-2014, 2014-2015). Penulis dapat dihubungi melalui surel arfianfd@gmail.com, dan id *line* arfianfd13.