Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

IMPLEMENTASI APLIKASI BERBASIS *WEB SERVICE* UNTUK SINKRONISASI BASIS DATA RELASIONAL HETEROGEN

I PUTU GEDE INDRA GUNAWAN
NRP 5113100073

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom.,M.Kom.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom, Ph.D.

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

TUGAS AKHIR – KI141502

IMPLEMENTASI APLIKASI BERBASIS *WEB SERVICE* UNTUK SINKRONISASI BASIS DATA RELASIONAL HETEROGEN

I PUTU GEDE INDRA GUNAWAN
NRP 5113100073

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom.,M.Kom.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom, Ph.D.

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

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION APPLICATION BASED ON WEB SERVICE FOR SYNCHRONIZING RELATIONAL DATABASE HETEROGEN

I PUTU GEDE INDRA GUNAWAN
NRP 5113100073

Supervisor I
Henning Titi Ciptaningtyas, S.Kom.,M.Kom.

Supervisor II
Bagus Jati Santoso, S.Kom, Ph.D.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI APLIKASI BERBASIS *WEB SERVICE* UNTUK SINKRONISASI BASIS DATA RELASIONAL HETEROGEN

TUGAS AKHIR

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

Oleh
I PUTU GEDE INDRA GUNAWAN
NRP : 5113 100 073

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Henning Titi Ciptaningtyas, S.Kom., M.Kom. ..... (Pembimbing 1)
   NIP: 19840708 201012 2 004

2. Bagus Jati Santoso, S.Kom., Ph.D. ..... (Pembimbing 2)
   NIP: -

SURABAYA
JUNI, 2017

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI APLIKASI BERBASIS *WEB SERVICE* UNTUK SINKRONISASI BASIS DATA RELATSIONAL HETEROGEN

Nama Mahasiswa	**: I Putu Gede Indra Gunawan**
NRP	**: 5113100073**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom.,M.Kom.**
Dosen Pembimbing 2	**: Bagus Jati Santoso, S.Kom, Ph.D.**

Abstrak

Dengan perkembangan teknologi yang pesat, semakin banyak perusahaan mulai melakukan manajemen informasi dengan skala enterprise. Hal ini dikarenakan berbagai kemudahan efisiensi yang diberikan, maka dibangunlah berbagai macam sistem informasi yang berbeda satu sama lain untuk menjawab berbagai kebutuhan perusahaan. Maka dari itu permasalahan dari heterogenitas sistem informasi muncul ketika data integrasi dilakukan. Masalah seperti data yang tidak konsistensi akan berdampak buruk bagi suatu perusahaan dalam jangka panjang. Dengan basis data yang heterogen pada berbagai sistem informasi pada suatu perusahaan dapat berdampak langsung pada sistem-sistem tersebut.

Untuk mengatasi permasalahan basis data yang heterogen ini, digunakan metode one-way sinkronisasi basis data dengan web service. Sinkronisasi basis data adalah proses yang kompleks dalam lingkungan basis data heterogen yang memiliki prinsip menjaga keseragaman data dan strukturnya. Sedangkan yang dimaksud dengan one-way sinkronisasi yaitu dimana suatu basis data yang disinkronisasi dengan melakukan duplikat data dari sumber data yaitu master node ke slave node. Selain itu, untuk menangkap perubahan pada basis data digunakan kombinasi dari trigger dan tabel capture log. Setiap

proses sinkronisasi yang dilakukan oleh basis data heterogen ditransmisikan dengan web service REST. Web service berfungsi untuk menangani paket data sinkronisasi dari master node ke queue node dan dari queue node ke slave node yang dikirimkan dengan request post.

Proses sinkronisasi dengan operasi insert, update dan delete pada tipe data integer dan string berhasil bekerja dengan 100% sukses. Namun, untuk operasi insert, update dan delete pada tipe data datetime tidak dapat berjalan dengan baik atau gagal. Beban maksimal yang dapat ditangani oleh web service pada node pusat adalah 100 pengguna secara simultan, dimana rasio request diterima 100% dan request drop adalah 0%. Dengan menggunakan variasi jumlah kolom, variasi jumlah query dan variasi jumlah tabel dilakukan pengujian performa aplikasi. Kemudian, untuk variasi jumlah kolom didapatkan hasil uji coba rata-rata selisih waktu sinkronisasinya yang bernilai rendah. Sehingga dapat disimpulkan variasi jumlah kolom memilki pengaruh yang tidak signifikan terhadap performa aplikasi dilihat dari rata-rata selisih waktu sinkronisasinya yang bernilai rendah. Untuk hasil coba variasi jumlah query didapatkan rata-rata selisih waktu sinkronisasi yang bernilai tinggi. Sehingga dapat disimpulkan variasi jumlah query memilki pengaruh yang signifikan terhadap performa aplikasi dilihat dari rata-rata selisih waktu sinkronisasi dari setiap variasi query yang digunakan bernilai tinggi. Selanjutya, untuk hasil coba variasi jumlah tabel didapatkan rata-rata selisih waktu sinkronisasi yang bernilai tinggi. Sehingga dapat disimpulkan variasi jumlah tabel memilki pengaruh yang signifikan terhadap performa aplikasi dilihat dari rata-rata selisih waktu sinkronisasi dari setiap variasi jumlah tabel yang digunakan bernilai tinggi.

Kata kunci: Web Service, Trigger, One-way sinkronisasi, basis data heterogen

IMPLEMENTATION APPLICATION BASED ON WEB SERVICE FOR SYNCHRONIZING RELATIONAL DATABASE HETEROGEN

Nama Mahasiswa	**: I Putu Gede Indra Gunawan**
NRP	**: 5113100073**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom.,M.Kom.**
Dosen Pembimbing 2	**: Bagus Jati Santoso, S.Kom, Ph.D.**

Abstract

With the rapid development of technology, more and more companies starting information management with the enterprise scale. This is because the efficiency of various facilities provided, then built a wide range of different information systems together to address the needs of the company. Thus the problem of heterogeneity of information systems arise when data integration is performed. Issues such as data consistency will be bad for a company in the long term. With a base of heterogeneous data in various information systems within a company can have a direct impact on those systems.

To overcome the problems of heterogeneous databases, in this thesis proposed method of one-way database synchronization with a web service. Synchronization of databases is a complex process in a heterogeneous database environment that has the principle of preserving the uniformity of data and its structure. While that is a one-way synchronization is where a database that is synchronized by performing duplicate data from the data source to the target. Moreover, to capture changes to the database used a combination of trigger and capture log table. Each synchronization process performed by the base of

heterogeneous data transmitted by a REST web service. The Web service used to handle the synchronization data packets from the master node to the queue node and from the queue node to the slave node that is sent with the post request.

The synchronization process with insert, update and delete operations on integer and string data types works successfully with 100% success. However, for insert operation, update and delete on datetime data types can not run properly or fail. The maximum load that can be handled by the web service on the central node is 100 users simultaneously, where the ratio of requests accepted is 100% and the request drop is 0%. By using variation number of columns, variation number queries and variation number of tables to test the performance of appication. Then, for the variation of the number of columns we get low average of synchronization time difference as trial result. So it can be concluded that the variation number of columns has an insignificant effect on the performance of application seen from its low value for average of synchronization time difference. For the variation of number of queries, we got high value for average of synchronization time difference. So it can be concluded that the variation number of the queries has a significant effect on application performance seen from its high value of the average of synchronization time difference. Then, for the variation of number of tables, we got high value for average of synchronization time difference. So it can be concluded that the variation number of the tables has a significant effect on application performance seen from its high value of the average of synchronization time difference.

Keywords: Web Service, Trigger, One-way synchronization, heterogeneous database

KATA PENGANTAR

Puji syukur penulis panjatkan kehadapan Tuhan Yang Maha Esa, karena dengan rahmat dan karunia-Nya lah penulis dapat menyelesaikan Tugas Akhir yang berjudul **"IMPLEMENTASI APLIKASI BERBASIS *WEB SERVICE* UNTUK SINKRONISASI BASIS DATA RELATSIONAL HETEROGEN".** Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Tuhan Yang Maha Esa.
2. Keluarga di Bali khususnya Ibu dan Bapak yang telah memberikan dukungan moral dan material yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
3. Ibu Henning Titi Ciptaningtyas, S.Kom.,M.Kom. selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
4. Bapak Bagus Jati Santoso, S.Kom, Ph.D. selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi kepada penulis dalam mengerjakan Tugas Akhir ini.
5. Bapak Dr. Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS.

6. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
7. Teman-teman angkatan 2013 yang yang telah berbagi ilmu, dan memberi motivasi kepada penulis.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

DAFTAR PSEUDOCODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kemajuan teknologi saat ini dapat membuat perusahaan ataupun organisasi menyimpan data-data yang dimiliki ke dalam suatu *database offline* maupun *cloud storage*. *Database* yang dimiliki dapat digunakan untuk kebutuhan membuat berbagai aplikasi maupun sistem informasi yang diperlukan perusahaan. Namun, setiap proses pembuatan aplikasi menggunakan menggunakan RDBMS (*Relational Database Management Systems*) yang berbeda-beda dikarenakan *preference* dari developer maupun karena pertimbangan biaya. Hal ini menyebabkan integrasi data menurun atau bahkan tidak ada sama sekali.

Data integrasi merupakan bagian penting dalam mengaplikasikan basis data terdistribusi yang diperoleh dari berbagai sumber yang kemudian disatukan dan diintegrasikan. Untuk menjaga agar data tetap konsisten digunakan metode sinkronisasi. Untuk mengatasi masalah ini digunakan metode *one-way* sinkronisasi data untuk menjaga konsistensi data yang diintegrasikan. Data sinkronisasi merupakan bagian dari replikasi, yang merupakan proses untuk memastikan setiap hasil replikasi memiliki objek dan data yang sama [1].

Dalam penelitian ini, dengan menggunakan tabel *log* berupa tabel *capture log* dan *trigger* maka dimungkinkan untuk melakukan sinkronisasi antara dua RDBMS [2]. Hal ini juga dapat memungkinkan pembuatan salah satu model sinkronisasi basis data yaitu, *one-ways* sinkronisasi. Dimana suatu *database* yang disinkronisasi dapat dilakukan dengan duplikat data dari sumber data ke basis data yang menjadi target [3]. Namun, semua proses tersebut memerlukan aplikasi yang membantu interaksi antara RDBMS yang berbeda. Aplikasi yang berbasis *web service* ini menggunakan t*rigger* untuk merekam informasi yang relevan ke

tabel *capture log*. Diharapkan aplikasi tersebut dapat membantu pengguna dalam proses sinkronisasi basis data.

1.2 Rumusan Masalah

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

1. Bagaimana sinkronisasi data antara RDBMS yang berbeda dengan aplikasi yang dibuat ?
2. Bagaimana ketahanan *web service* pada aplikasi ?
3. Bagaimana pengaruh variasi jumlah kolom terhadap performa aplikasi ?
4. Bagaimana pengaruh variasi jumlah *query* terhadap performa aplikasi ?
5. Bagaimana pengaruh variasi jumlah tabel terhadap performa aplikasi ?

1.3 Batasan Masalah

Batasan dalam Tugas Akhir ini, yaitu:

1. Aplikasi ini menggunakan bahasa pemograman Python.
2. RDMS yang digunakan adalah Oracle dan MySql.
3. Menggunakan *web service* REST dengan *library flask*.
4. Query yang disinkronisasikan oleh aplikasi adalah *insert*, *update* dan *delete*.
5. Tipe data yang disinkronisasikan adalah *string* dan *integer*.

1.4 Tujuan

Tujuan Tugas Akhir ini adalah melakukan sinkronisasi data antara RDBMS yang berbeda serta mengetahui performa dan ketahanannya.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini antara lain:

1. Mempermudah dalam integrasi data pada RDMBS yang berbeda.
2. Mempertahankan konsistensi data yang didistribusikan.

1.6 Metodologi

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

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Penyusunan proposal Tugas Akhir dilaksanakan untuk merumuskan masalah serta melakukan penetapan rancangan dasar dari sistem yang akan dikembangkan dalam pelaksanaan Tugas Akhir ini.

2. Studi literatur
   Pada tahap ini dilakukan pemahaman informasi dan literatur yang diperlukan untuk tahap implementasi program. Tahap ini diperlukan untuk membantu memahami penggunaan komponen-komponen terkait dengan sistem yang akan dibangun, antara lain *web service* dan *two-way* sinkronisasi *database*.

3. Analisis dan perancangan perangkat lunak
   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* sistem, yang merupakan rancangan dasar dari sistem yang

akan dibuat. Serta dilakukan desain fungsi yang akan dibuat yang ditunjukkan melalui *pseudocode*.

4. Implementasi perangkat lunak

   Implementasi perangkat lunak merupakan tahap membangun rancangan program yang telah dibuat. Pada tahap ini akan direalisasikan mengenai rancangan apa saja yang telah didefiniskan pada tahap sebelumnya. Fungsi yang ada pada tahap ini merupakan fungsi hasil implementasi dari tahap analisis dan perancangan perangkat lunak.

5. Pengujian dan evaluasi

   Pada tahap ini dilakukan uji coba pada data yang telah dikumpulkan. Tahap ini digunakan untuk mengevaluasi kinerja program serta mencari masalah yang mungkin timbul saat program dievaluasi serta melakukan perbaikan jika terdapat kesalahan pada program.

6. Penyusunan buku Tugas Akhir

   Pada tahap ini disusun buku yang memuat dokumentasi mengenai perancangan, pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu perumusan masalah, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Perangkat Lunak

Bab ini berisi tentang dasar dari algoritma yang akan diimplementasikan pada Tugas Akhir ini.

Bab IV Implementasi

Bab ini membahas mengenai implementasi dari rancangan yang telah dibuat pada bab sebelumnya.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan mengenai kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari perangkat lunak yang telah dibuat sesuai dengan data yang diujikan.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang telah dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Basis Data

Basis data atau *database* merupakan sebuah koleksi atau kumpulan dari data yang bersifat mekanis, terbagi, terdefinisi secara formal serta terkontrol. Pengontrolan dari sistem *database* tersebut adalah terpusat, yang biasanya dimiliki dan juga dipegang oleh suatu organisasi [4]. Basis data adalah kumpulan data yang saling berelasi. Data sendiri merupakan fakta mengenai obyek, orang dan lain-lain. Data dinyatakan dengan nilai (angka, deretan karakter, atau simbol). Basis data dapat didefinisikan dalam berbagai sudut pandang seperti berikut ini:

1. Himpunan kelompok data yang saling berhubungan yang diorganisasi sedemikian rupa sehingga kelak dapat dimanfaatkan dengan cepat dan mudah
2. Kumpulan data yang saling berhubungan yang disimpan secara bersama sedemikian rupa tanpa pengulangan (*redundancy*) yang tidak perlu, untuk memenuhi kebutuhan
3. Kumpulan file/tabel/arsip yang saling berhubungan yang disimpan dalam media penyimpan elektronis [5].

2.2 Sistem Manajemen Basis Data

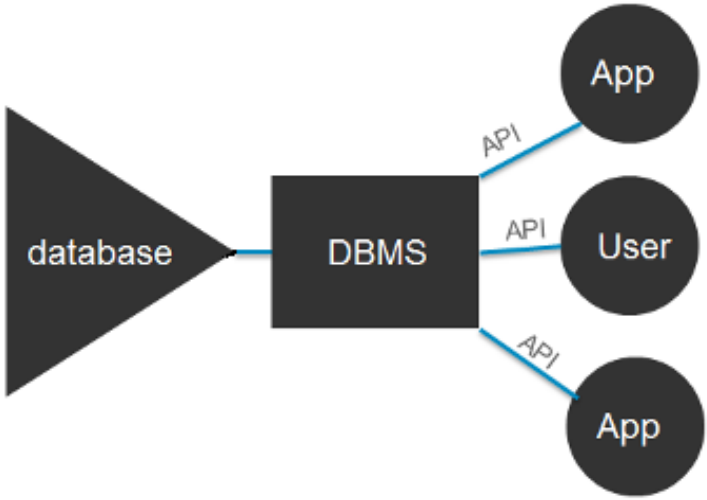
Sistem manajemen basis data atau *Database Managemet System* (DBMS) adalah sistem perangkat lunak untuk membuat dan mengelola basis data. DBMS menyediakan pengguna dan programmer dengan secara sistematis untuk membuat, mengambil, memperbarui dan mengelola data. Sebuah DBMS memungkinkan pengguna untuk membuat, membaca, memperbarui dan menghapus data dalam basis data. DBMS pada dasarnya berfungsi sebagai *interface* antara pengguna dan basis data atau program aplikasi, memastikan bahwa data konsisten, terorganisir dan tetap mudah diakses.

DBMS mengelola tiga hal penting: data, *database engine* yang memungkinkan data dapat diakses, dikunci dan dimodifikasi dan skema basis data, yang mendefinisikan struktur logis basis data ini. Ketiga elemen dasar membantu memberikan *concurrency*, keamanan, integritas data dan prosedur administrasi yang seragam. Tipikal tugas administrasi basis data yang didukung oleh DBMS meliputi manajemen perubahan, kinerja *monitoring* / *tuning* serta *backup* dan pemulihan. Banyak sistem manajemen basis data juga bertanggung jawab untuk *rollbacks* otomatis, *restart* dan *recovery* serta *logging* dan audit kegiatan.

DBMS paling sering digunakan untuk memberikan pandangan data secara terpusat yang dapat diakses oleh beberapa pengguna, dari berbagai lokasi, dengan cara yang terkendali. Sebuah DBMS dapat membatasi data apa yang dapat dilihat penggunanya, serta bagaimana pengguna dapat melihat data, memberikan banyak pandangan dari skema *database* tunggal. Pengguna dan program perangkat lunak bebas dari keharusan untuk memahami dimana data secara fisik terletak atau pada jenis media penyimpanan itu berada karena DBMS menangani semua permintaan.

DBMS dapat menawarkan baik logis dan fisik independensi data. Hal ini berarti DBMS dapat melindungi pengguna dan aplikasi dari perlu tahu dimana data disimpan atau harus khawatir

tentang perubahan struktur fisik data (penyimpanan dan hardware). Selama program menggunakan antarmuka pemrograman aplikasi (API) untuk database yang disediakan oleh DBMS, pengembang tidak perlu memodifikasi program hanya karena perubahan yang telah dibuat ke *database* seperti yang diilustrasikan pada Gambar 2.1. Untuk DBMS relasional (RDBMSs), API ini menggunakan bahasa SQL yaitu, bahasa pemrograman standar untuk mendefinisikan, melindungi dan mengakses data dalam RDBMS. [6]



Gambar 2.1 Illustrasi kerja DBMS [6]

2.2.1 Oracle

Basis data Oracle (Oracle DB) adalah sistem manajemen basis data relasional (RDBMS) dari Oracle Corporation. Awalnya dikembangkan pada tahun 1977 oleh Lawrence Ellison dan pengembang lainnya, Oracle DB adalah salah satu RDBMS yang paling terpercaya dan secara luas digunakan. Sistem ini dibangun di sekitar kerangka *database* relasional di mana objek data dapat langsung diakses oleh pengguna (atau *front end* aplikasi) melalui

bahasa *query* terstruktur (SQL). Oracle merupakan sebuah arsitektur basis data relasional yang memiliki skalabilitas tinggi dan sering digunakan oleh perusahaan-perusahaan global, dimana Oracle dapat digunakan untuk mengelola dan mengolah data di jaringan area luas dan maupun lokal. Oracle *database* memiliki komponen jaringan sendiri untuk memungkinkan komunikasi di seluruh jaringan. Oracle DB juga dikenal sebagai Oracle RDBMS atau hanya Oracle [7].

Sebuah server database Oracle terdiri dari *database* dan setidaknya satu *instance database*. *Database* adalah kumpulan file-file yang terletak pada disk, tempat menyimpan data. Sebuah *instance* adalah seperangkat struktur memori yang mengelola file *database*. *Instance* terdiri dari sebuah *shared memory area* yang disebut *system global area* (SGA) dan satu set *background* proses. Sebuah *instance* bisa eksis secara independen dari file *database* [8].

Oracle digunakan sebagai salah satu DBMS pada tugas akhir ini karena kelebihan Oracle *database* yang disebutkan sebelumnya. Dengan pengguna yang cukup banyak Oracle memiliki komunitas yang sangat besar sehingga memudahkan pengguna dalam mencari referensi dan artikel yang berkaitan dengan pengerjaan tugas akhir.

Kekurangan dari DBMS ini adalah sulitnya mengatur akses dan pembuatan akun dikarenakan banyaknya peran akses. Kekurangan Oracle yang lain adalah Oracle memakai *space storage* yang cukup tinggi dan versi Oracle terbaru yaitu Oracle 12c hanya bisa digunakan atau didukung dengan komputer dengan prosesor 64-bit. Pada pengerjaan ini, digunakan Oracle 12c versi standar dikarenakan adanya fitur *identity* yang menggantikan *sequence* untuk pembuatan *autoincrement* pada kolom *primary key*. .

2.2.2 MySQL

MySQL adalah sistem manajemen basis data Open Source SQL yang paling populer, dikembangkan, didistribusikan, dan didukung oleh Oracle Corporation. Open Source berarti bahwa adalah mungkin bagi siapa saja untuk menggunakan dan memodifikasi perangkat lunak. Siapa saja dapat men-download software MySQL dari internet dan menggunakannya tanpa membayar sepeser pun.

MySQL Server dapat berjalan dengan lancar pada komputer, laptop dan server web, dapat berjalan bersamaan dengan aplikasi lainnya, serta memerlukan sedikit perawatan. MySQL juga dapat ditingkatkan skalabilitasnya hingga membentuk *cluster* dengan memiliki jaringan yang terhubung. MySQL Server pada awalnya dikembangkan untuk menangani basis data berukuran besar dengan cepat dan telah berhasil digunakan dalam lingkungan produksi dalam beberapa tahun terakhir. Dalam pengembangannya sekarang, MySQL Server menawarkan bayak fungsi yang berguna. Dengan kecepatannya, konektivitas, dan keamanan membuat MySQL Server yang sangat cocok untuk mengakses basis data melalui Internet [9].

MySQL digunakan sebagai salah satu DBMS yang digunakan karena popularitasnya yang tinggi dan performa dari MySQL yang bagus. Sedangkan kekurangan MySQL yaitu, MySQL dianggap tidak cocok digunakan pada skala perusahan besar karena diragukan kemampuannya, Selain itu, *technical support* dari MySQL juga dianggap kurang baik karena berhubungan dengan status *open source* yang dimiliki oleh MySQL. Namun, kekurangan ini tidak begitu penting karena pengerjaan tugas akhir ini tidak menggunakan data dengan skala perusahan besar.

2.3 Sinkronisasi Basis Data

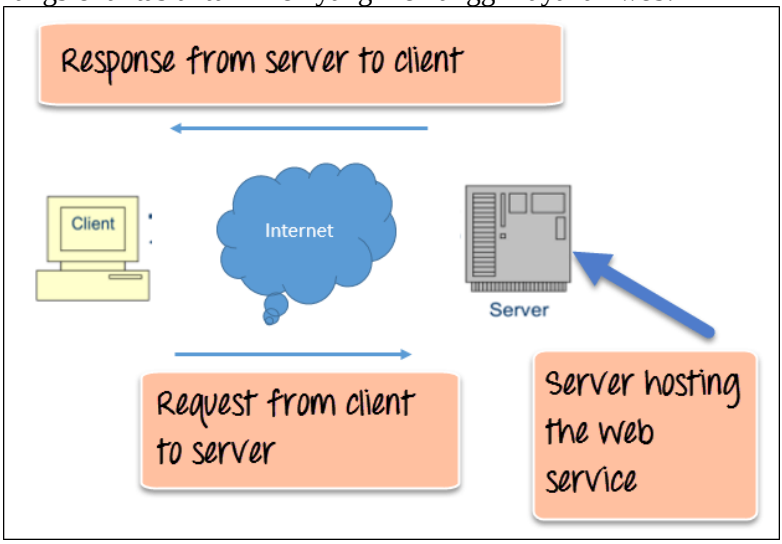
Sinkronisasi basis data adalah proses yang kompleks dalam lingkungan basis data heterogen yang memiliki prinsip menjaga keseragaman data dan strukturnya yang menjadi bagian yang mendasar dari sinkronisasi data [10]. Sedangkan menurut Jue Wang dan Dong-song Zhang, sinkronisasi basis data mengacu ke sistem basis data, ada dua atau lebih basis data, melalui cara-cara teknis untuk menyebarkan data dalam satu basis data ke basis datayang lain dalam rangka menjaga kesatuan dan konsistensi dari semua objek data dalam basis data. Oleh karena itu, basis data berdasarkan sistem, data di dalam satu basis data diubah, maka data dalam basis data lain harus juga terjadi perubahan, berdasarkan kebutuhan aplikasi. Misalnya dalam sistem basis data terdistribusi, sering data yang sama pada basis data dimiliki beberapa *node*. Dalam situasi ini, karena beberapa *node* dapat beroperasi pada data yang sama, sehingga untuk menjaga konsistensi data untuk semua *node* diperlukan sinkronisasi basis data, untuk memastikan sistem dapat beroperasi secara normal [11].

Sinkronisasi yang melibatkan sinkronisasi item dalam satu arah, dari sumber data ke basis data sasaran. Ini berarti bahwa ketika kita membuat, menghapus atau mengubah item di basis data sumber, hal ini juga akan tercermin dalam basis data sasaran. Di sisi lain, perubahan pada item diperkenalkan ke basis data target tidak akan direplikasi ke basis data sumber. Selain itu, sinkronisasi satu arah dapat didefinisikan antara lebih dari dua *node*. Tergantung pada jenis sinkronisasi satu arah, jumlah sumber atau target *node* dapat bervariasi dan tergantung terhadap pilihan kita [3]. Jenis sinkronisasi ini digunakan karena *feasibility* dari jenis sinkronisasi ini. Banyaknya referensi, jurnal dan arsitektur yang mendukung menjadi pertimbangan digunakannya sinkronisasi ini. *One-way* sinkronisasi memiliki kelebihan tingkat konsistensi data yang tinggi, akurasi yang tinggi dan performa yang melebihi *two-way* sinkronisasi dikarenakan perubahan yang dilakukan hanya pada *node* sumber saja. Kekurangan dari jenis sinkronisasi ini yaitu *node* yang menjadi tujuan sinkronisasi dari *node* sumber tidak bisa

dilakukan perubahan. Teknik *one-way* sinkronisasi ini dijadikan acuan cara sinkronisasi pada aplikasi.

2.4 *Web Service*

Web service menyediakan sarana standar interoperasi antara aplikasi perangkat lunak yang berjalan pada berbagai jenis platform dan kerangka kerja [12]. *Web service* adalah modul *software* yang dirancang untuk melakukan satu set tertentu dari tugas. *Web service* dapat dicari melalui jaringan dan juga dapat dipanggil. Ketika dipanggil *web service* akan mampu menyediakan fungsionalitas untuk klien yang memanggil layanan web.



Gambar 2.2 Illustrasi cara kerja web service [13]

Pada Gambar 2.2 di atas menunjukkan pandangan yang sangat sederhana tentang bagaimana web service bekerja. Klien akan memanggil serangkaian layanan atau fungsi dari *web service* melalui *requests* ke server yang menjadi *host web service* [13].

REST merupakan singkatan dari *Representational State Transfer*. REST digunakan untuk membangun web service yang ringan, mudah dirawat, dan tingkat skalabilitas tinggi. REST bisa

juga disebut dengan gaya arsitektur, dan pendekatan komunikasi yang sering digunakan dalam pengembangan web service. Penggunaan REST sering lebih disukai daripada SOAP (*Simple Object Access Protocol*) karena REST tidak memanfaatkan banyak bandwidth, yang membuatnya lebih cocok untuk digunakan melalui internet. Pendekatan SOAP membutuhkan penulisan atau menggunakan program server yang disediakan (untuk melayani data) dan program klien (untuk *request* data). Arsitektur REST yang terpisah dan bobot komunikasi yang lebih ringan antara produsen dan konsumen, membuat REST populer untuk membuat API berbasis *cloud*, seperti yang disediakan oleh Amazon, Microsoft, dan Google. Ketika *web service* menggunakan arsitektur REST, maka *web service* itu disebut RESTful API (*Application Programming Interfaces*).

REST, yang biasanya menggunakan protokol HTTP (Hypertext Transfer Protocol). REST memiliki beberapa constraint pada arsitekturnya yaitu:

1. Konsumen dari produsen yang terpisah
2. Adanya interaksi yang bersifat stateless
3. Mampu memanfaatkan cache
4. Memanfaatkan sistem berlapis
5. Memanfaatkan uniform interfaces(*Put, get, post, delete*)

REST sering digunakan dalam aplikasi *mobile*, situs jejaring sosial dan proses bisnis otomatis. REST menekankan bahwa interaksi antara klien dan *servic*e ditingkatkan dengan memiliki membatasi jumlah operasi. Fleksibilitas disediakan oleh menugaskan sumber daya dengan Universal Resource Identifier(URI) yang unik. Dikarenakan setiap kata kerja memiliki arti tertentu (GET, POST, PUT dan DELETE), REST dapat menghindari ambiguitas.

REST memiliki beberapa kekurangan. REST tidak bisa langsung untuk menghasilkan klien dari metadata server-side yang dihasilkan. SOAP mampu mendukung ini dengan *Web Service Description Language* (WSDL). REST memiliki beberapa

kelebihan yaitu, RESTful web service mudah dibuat karena tersedianya alat-alat yang gratis maupun yang harganya murah, REST memiliki skalabilitas yang tinggi sedangkan SOAP memiliki skalabilitas yang sangat rendah, REST menggunakan format pesan yang lebih pendek dibandingkan dengan SOAP yang menggunakan XML untuk semua pesan sehingga otomatis membuat REST lebih cepat daripada SOAP, REST mampu mendukung cloud karena tingkat skalabilitasnya yang tinggi [14]. REST digunakan sebagai *web service* pada aplikasi karena kecepatan dan performanya yang tinggi sehingga mampu lebih cepat dalam mentransmisi data yang akan disinkronisasikan ke *node* tujuan. Jenis *web service* REST ini dijadikan sebagai lapisan aplikasi yang menerima *request* dari *node* lain dengan kata lain transportasi data pada aplikasi menggunkan *web service* REST.

2.5 *Trigger*

Trigger adalah objek basis data yang berhubungan dengan tabel. *Trigger* aktif hanya saat suatu event terjadi terhadap tabel dimana trigger tersebut dipasang. Beberapa kegunaan untuk *trigger* adalah melakukan pemeriksaan dari nilai-nilai yang akan dimasukkan ke dalam tabel atau untuk melakukan perhitungan pada nilai-nilai yang terlibat dalam *update* [15]. *Trigger* digunakan sebagai komponen yang menangkap setiap perubahan yang ada pada tabel tersebut. Hal ini dikarenakan *trigger* merupakan *event-driven* komponen yaitu komponen yang mampu menangkap perubahan secara cepat saat peristiwa tertentu yang didefinisikan berlangsung. Kecepatan dan mudahnya trigger didefinisikan menjadi alasan utama digunakannya *trigger* untuk menangkap perubahan.

2.6 Python

Python adalah bahasa pemrograman tingkat tinggi yang didesain untuk mudah dibaca dan mudah dimplementasi. Bahasa python bersifat *open source*, dimana kita dapat mengunduh dan menggunakannya secara gratis. Python dapat dijalankan pada beberapa sistem operasi seperti Mac, Windows dan Linux. Python biasanya digunakan untuk membuat program seperti aplikasi web dan konten web yang dinamis. Selain itu, python juga didukung oleh aplikasi modeling 2D dan 3D seperti Blender dan Autodesk Maya yang memungkinkan penggunanya membuat *plug-in* dan *custom extension* [16].

Bahasa pemograman python digunakan karena bahasa python merupakan bahasa pemograman tingkat tinggi yang mudah dipahami. Selain itu, python memiliki banyak *library* yang bersifat *open source* sehingga memudahkan proses pembuatan aplikasi. Bahasa python digunakan untuk membuat semua komponen aplikasi, yaitu *capture* komponen, *queue* komponen dan *receiver* komponen.

2.6.1 *Library* MySQLdb

MySQLdb adalah *library* python yang berperan sebagai *interface* untuk mengkoneksikan basis data MySQL dengan python. *Library* ini mengimplementasikan Python Database API v2.0 dan dibangun di atas MySQL C API [17]. *Library* ini digunakan karena *library* ini memiliki dokumentasi yang jelas dan lengkap. Selain itu, pengguna dari *library* ini lumayan banyak sehingga memudahkan dalam mencari tutorial. *Library* ini digunakan sebagai library yang mengakses basis data MySQL, untuk membuat *capture* komponen, *queue* komponen dan *receiver* komponen pada aplikasi.

2.6.2 *Library* cx_Oracle

Library cx_Oracle adalah *library* python merupakan sebuah modul untuk mengakses basis data Oracle dari python dengan spesifikasi *python database* API [18]. *Library* ini digunakan untuk pembuatan aplikasi karena *library* ini bersifat *open source* dan merupakan *library* yang dibuat langsung oleh Oracle. *Library* ini digunakan sebagai *library* yang mengakses basis data oracle, digunakan pada *capture* komponen dan *receiver* komponen.

2.6.3 *Library* flask

Flask adalah *library* python yang merupakan mikro *framework* untuk pembuatan aplikasi *web*. Flask termasuk ke dalam kategori mikro *framework* karena memiliki sedikit ketergantungan dengan *library* lainnya [19]. Flask digunakan karena *library* ini mudah dijalankan, ringan dan memiliki dokumentasi yang lengkap. Flask digunakan sebagai *library* yang membuat *web service* pada *queue* dan *slave node*.

2.6.4 *Library* Requests

Requests adalah *library* python yang menangani HTTP, didesain sedemikian hingga mudah dipahami manusia. *Library* ini dapat digunakan untuk mengirimkan *request* HTTP dari python [20]. *Library* ini digunakan karena dokumentasi yang lengkap dan juga memiliki fitur melakukan *json encode* pada data yang dikirimkan. *Library* ini digunakan sebagai *library* yang mengirimkan *query* dari *master node* ke *queue node* maupun dari *queue node* ke *slave node*.

2.6.5 *Library* Locust

Locust adalah *library* python yang merupakan *user load testing tools* yang mudah digunakan. *Library* ini ditujukan untuk pengujian beban pada situs web atau sistem lainnya dan mengetahui seberapa banyak pengguna yang bisa ditangani sistem secara bersamaan [21]. *Library* digunakan karena dokumentasinya yang lengkap dan mudah dipahami. *Library* ini digunakan sebagai tool yang membantu uji coba ketahanan *web service* aplikasi.

BAB III
ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

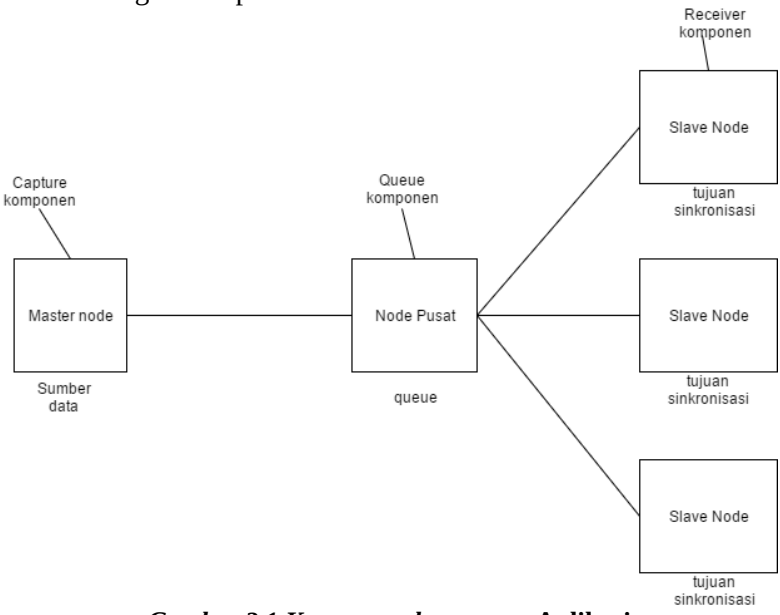
Pada bab ini akan dijelaskan perancangan perangkat lunak yang dibuat. Perancangan akan dibagi menjadi dua tahapan utama, yaitu perancangan arsitektur program dan peracancangan alur proses utama program. Pada bab ini juga akan dijelaskan mengenai gambaran umum setiap proses utama program dalam diagram alir beserta penjelasannya.

3.1 Penjelasan Umum Arsitektur Aplikasi

Pada subbab ini menjelaskan secara umum mengenai arsitektur aplikasi. Seperti yang dijelaskan sebelumnya, tujuan aplikasi ini menutupi heterogenitas dari DBMS yang digunakan. Hal ini menyebabkan heterogenitas dari DBMS yang digunakan tidak menghambat proses sinkronisasi data. Selain itu dengan menggunakan web service sebagai transport layer seperti yang disampaikan oleh Yubiao Wang, Junhao Wen, Weitao Fang dan Xiru Rao performa sinkronisasi menjadi lebih cepat [2].

Akses basis data dikelola oleh DBMS maka dari itu dengan memanfaatkan DB API yang dimiliki setiap DBMS kita bisa mengkoneksikan aplikasi dengan akses basis data yang telah diberikan oleh DBMS. Untuk melakukan koneksi aplikasi dan DBMS ini diperlukan *credential* basis data tersebut. *Credential* ini dapat berupa *username, password*, nama basis data dan alamat basis data biasanya dalam bentuk IP *address*. Untuk masalah *credential* ini MySQL dan Oracle memiliki beberapa perbedaan. Perbedaan tersebut adalah dalam Oracle *credential* nama basis data yang dimaksud adalah nama *service* pada host yang menangani Oracle, nama *service* ini biasanya dapat ditemukan pada file tnsnames.ora yang berada pada lokasi instalasi. Sedangkan pada MySQL *credential* nama basis data merupakan nama basis data sesungguhnya yang ditampilkan pada DBMS MySQL. Dengan

akses dari DB API ini aplikasi mampu mengelola proses sinkronisasi, termasuk alur data keluar maupun masuk basis data dan juga menangkap perubahan pada basis data melalui komponen penangkap perubahan. Selain komponen *capture* aplikasi yang dibuat penulis memiliki komponen lain yaitu komponen *queue* dan komponen yang menangani atau menerima JSON pada *node* tujuan disebut dengan komponen *receiver*.

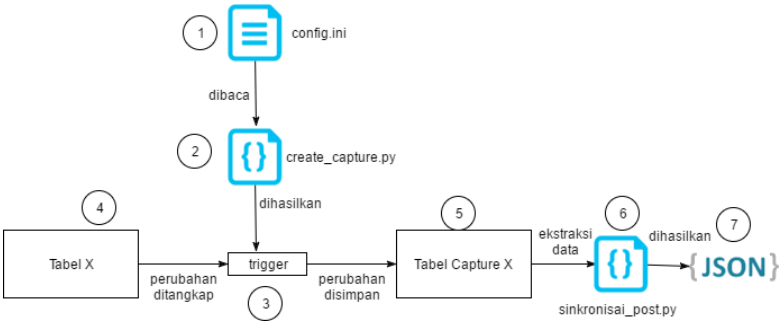


***Gambar 3.1 Komponen-komponen* Aplikasi**

Pada Gambar 3.1 digambarkan letak dan posisi komponen-komponen aplikasi secara umum. Dapat dilihat pada gambar di atas, lokasi *capture* komponen berada pada *master node* atau sumber data karena *capture* komponen berfungsi untuk menangkap perubahan pada sumber data yang kemudian dikirimkan ke queue komponen pada *node* pusat untuk nantinya didistribusikan ke *slave node* tujuan yang diterima oleh *receiver* komponen pada *slave node*. Penjelasan lebih rinci akan dijelaskan pada subbab-subbab selanjutnya.

3.1.1 *Capture* Komponen

Capture komponen merupakan salah satu komponen pada aplikasi yang dibuat. *Capture* komponen ini terletak pada *master node*. *Capture* komponen berfungsi untuk menangkap perubahan pada sumber data atau *master node*. Dapat dilihat pada Gambar 3.2 komponen ini terdiri dari dua sub-komponen utama yaitu, *trigger* dan tabel *capture*. *Trigger* berfungsi untuk menangkap perubahan pada tabel tempat *trigger* itu terpasang sedangkan tabel capture berfungsi untuk menampung semua perubahan yang ditangkap oleh *trigger*. Perubahan yang dimaksud di sini adalah aksi INSERT, UPDATE dan DELETE pada basis data.



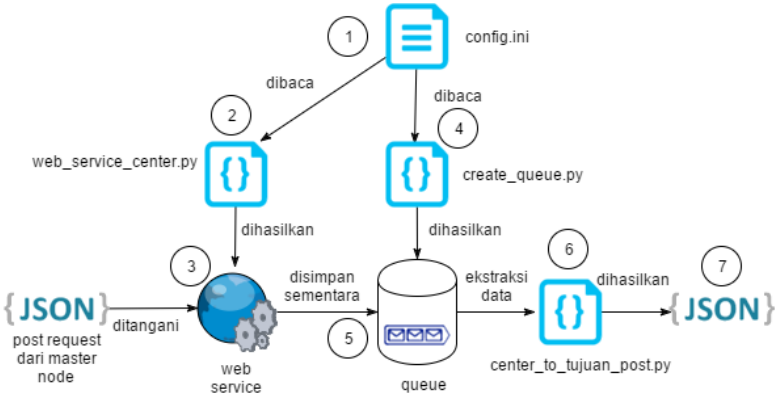
Gambar 3.2 Sub-komponen dari capture komponen

Sub-komponen *trigger* dihasilkan dari kodingan create_capture.py pada nomor 1. Kode ini membaca file *config* pada nomor 1, pada aplikasi untuk mendapatkan *credential* basis data kemudian membuat *trigger* pada tabel mana saja yang tertera pada file *config* yang ditunjukkan pada nomor 3. Selanjutnya, trigger-triger yang dihasilkan siap untuk menangkap perubahan yang terjadi pada nomor 4. Setelah perubahan tertampung di tabel *capture* pada nomor 5, file kodingan sinkronisasi_post.py melakukan ekstraksi

data pada tabel *capture* dan menghasilkan JSON dari data ektraksi yang ditunjukkan pada nomor 6. Selanjutnya, file JSON yang ditunjukkan pada nomor 7, dikirimkan melalui *request* ke *node* pusat secara periodik.

3.1.2 *Queue* Komponen

Queue komponen merupakan komponen aplikasi yang terletak pada *node* pusat. Tugas *queue* komponen adalah mengolah JSON yang diterima dari hasil *post request master node* dan menampungnya dalam *queue*. Queue komponen terdiri dari dua komponen utama yaitu *web service* dan tabel *queue*.



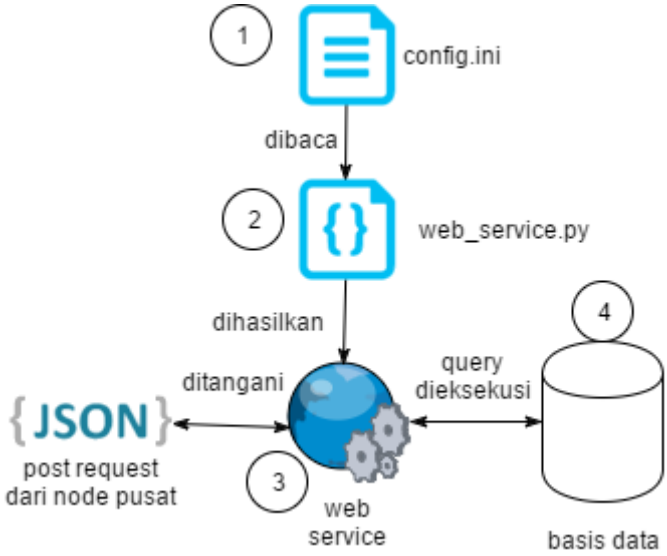
Gambar 3.3 Sub-komponen dari queue komponen

Pada Gambar 3.3 dapat kita perhatikan, sub-komponen *web service* pada nomor 3 bertugas untuk menangani *post request* dari *master node*. *Web service* ini dihasilkan dari pembacaan file konfigurasi pada nomor 1 oleh web_service_center.py pada nomor 2. Sub komponen pada nomor 5 ini dihasilkan dari pembacaan file konfigurasi pada nomor 1 oleh create_queue.py pada nomor 2. Sedangkan sub-komponen *queue* yang ditunjukkan pada nomor 5, berbentuk tabel pada *node* pusat yang menampung *query*, alamat tujuan, alamat sumber data, nama tabel tujuan. Data-data yang ditampung hanya bersifat sementara karena setelah data-data

tersebut diekstraksi yang ditunjukkan nomor 6 dan 7. Jika data tersebut berhasil dikirimkan ke *slave node* tujuan maka data tersebut akan dihapus dari *queue*. Data-data ini hanya akan dikirimkan jika *slave node* tujuan hidup, jika *slave node* tujuan tidak hidup maka data tersebut disimpan sampai *slave node* terdeteksi oleh *node* pusat.

3.1.3 *Receiver* Komponen

Receiver komponen merupakan komponen aplikasi yang terletak pada *slave node*. Komponen ini hanya terdiri dari satu sub-komponen utama yaitu *web service* pada nomor 3, yang menangani post request dari *node* pusat. *Web service* ini dihasilkan dari pembacaan file konfigurasi pada nomor 1 oleh web_service.py pada nomor 2.



Gambar 3.4 Sub-komponen dari receiver komponen

Web service ini berfungsi untuk mengolah *request* yang berfomat JSON dan mendapatkan perintah *query* dari request tersebut. Selanjutnya *query* tersebut dieksekusi pada basis data local yang ditunjukkan pada nomor 4. Jika hasil eksekusi data sukses maka *web service* akan memberikan respon sukses kepada *node* pusat seperti yang ditunjukkan Gambar 3.4.

3.2 Perancangan Alur Proses Program

Tahap perancangan alur proses program yaitu proses *capture* perubahan, proses penyimpanan pada *queue* dan proses penerimaan oleh *receiver.*

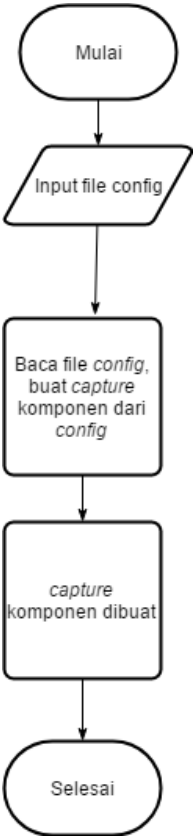
3.2.1 Proses *Capture* Perubahan

Proses *capture* perubahan adalah proses penangkapan perubahan pada table-tabel yang menjadi tujuan sinkronisasi oleh *capture* komponen. Pada Gambar 3.5 diilustrasikan diagram alir dari pembuatan *capture* komponen. Dari diagram alir di atas, proses yang dilakukan adalah pembacaan file *config* dan pembuatan *capture* komponen berupa *trigger* dan tabel *capture*. Fungsi *capture* komponen sebelumnya sudah dijelaskan pada 3.1.1.

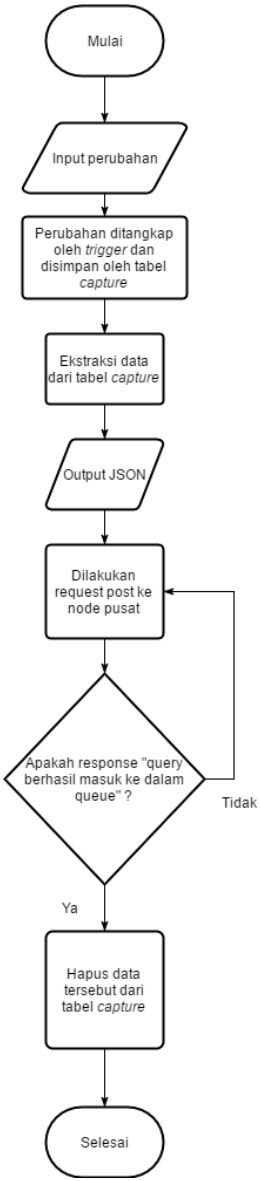
Pada Gambar 3.6 digambarkan alur proses penangkapan perubahan. Perubahan yang dimaksud di sini adalah aksi INSERT, UPDATE dan DELETE pada basis data. Perubahan tersebut ditangkap oleh trigger lalu disimpan ke dalam tabel *capture*. Kemudian proses berikutnya yaitu ekstraksi data dari tabel *capture* ke dalam format JSON. Data yang berbentuk JSON ini dikirimkan ke *node* pusat melalui *post request.* Jika respon dari *node* pusat berupa pesan yang berisi "*query* berhasil masuk dalam queue" maka data tersebut dihapus dari tabel *capture*. Namun, jika respon "*query* gagal masuk ke dalam queue" maka proses *request* diulang kembali. Proses ekstraksi data sampai pengiriman data ke *node* pusat ini dilakukan secara periodik.

3.2.2 Proses Penyimpanan pada Queue

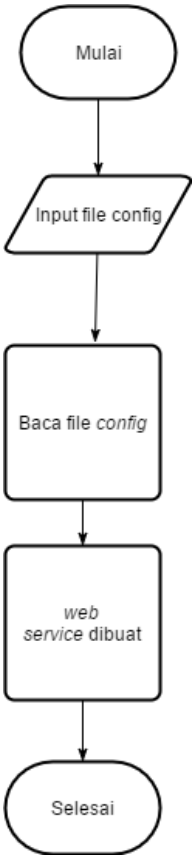
Proses penyimpan pada queue merupakan proses penerimaan request dari *master node* yang kemudian JSON dari request tersebut dilakukan diseminasi menjadi data-data yang dapat disimpan pada queue. Proses penyimpanan queue ini dimulai dengan pembuatan queue pada basis data dan *web service* yang menangani *request* dari *master node* di *node* pusat.



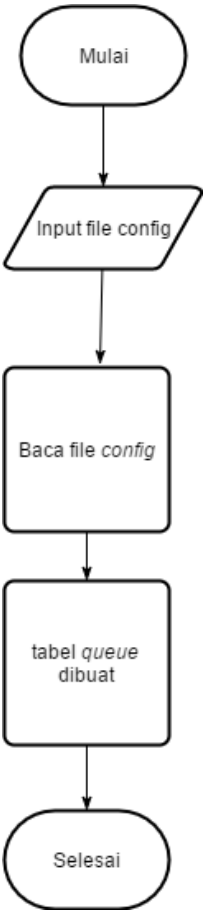
Gambar 3.5 Diagram alir pembuatan capture komponen



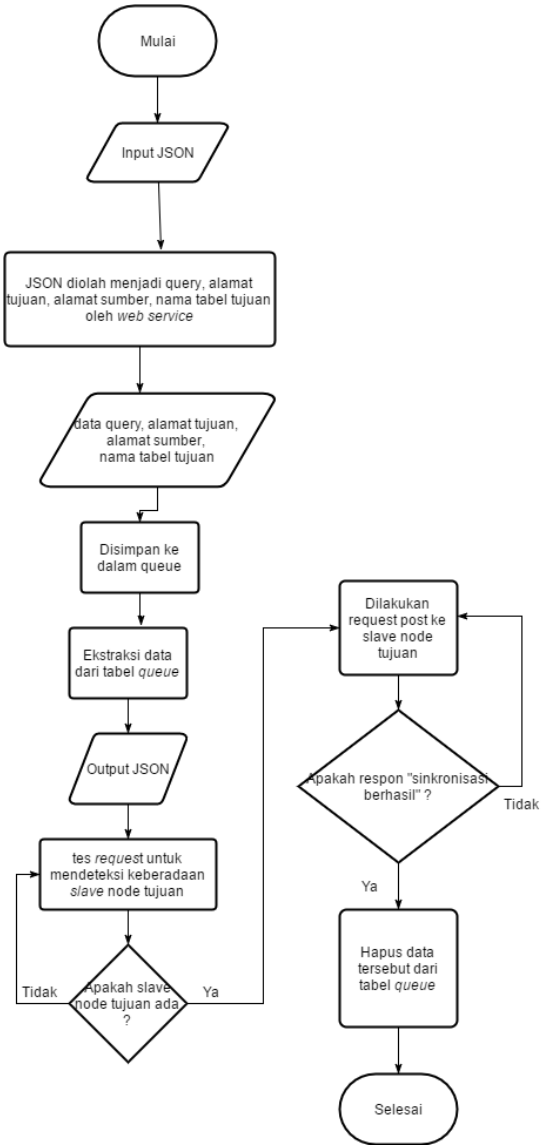
Gambar 3.6 Diagram alir proses penangkapan perubahan



Gambar 3.7 Diagram alir pembuatan web service di node pusat



Gambar 3.8 Diagram alir pembuatan tabel queue



Gambar 3.9 Diagram alir proses penyimpanan, ekstraksi queue dan pengiriman ke slave node

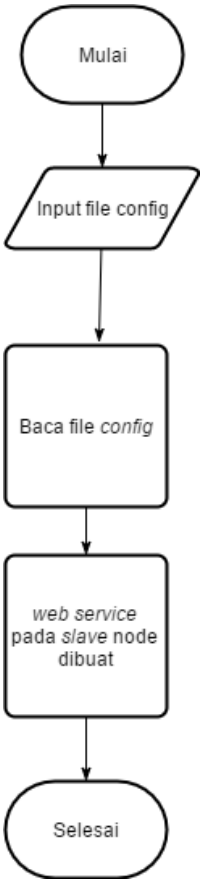
Pada Gambar 3.7 digambarkan proses pembacaan file *config* lalu pembuatan web service yang menangani post request dari *master node* dan pada Gambar 3.8 digambarkan proses pembuatan tabel queue pada basis data *node* pusat dari file *config*.

Pada Gambar 3.9 diilustrasikan dari proses penyimpanan data ke *queue* sampai pengiriman data ke *slave node* tujuan. Proses penyimpanan data ke queue dimulai dari input JSON yang ditangani *web service* pada *node* pusat. Data dari file JSON ini kemudian diolah dan dipecah menjadi data *query*, alamat tujuan, alamat sumber dan nama tabel tujuan yang kemudian akan dimasukkan ke dalam *queue*. Setelah itu, *node* pusat mengirimkan *request* ke *slave node* tujuan untuk mengetahui apakah *node* tujuan tersebut ada atau tidak. Proses pengetesan keberadaan slave *node* tujuan terus diulang sampai *node* tersebut terdeteksi oleh *node* pusat.

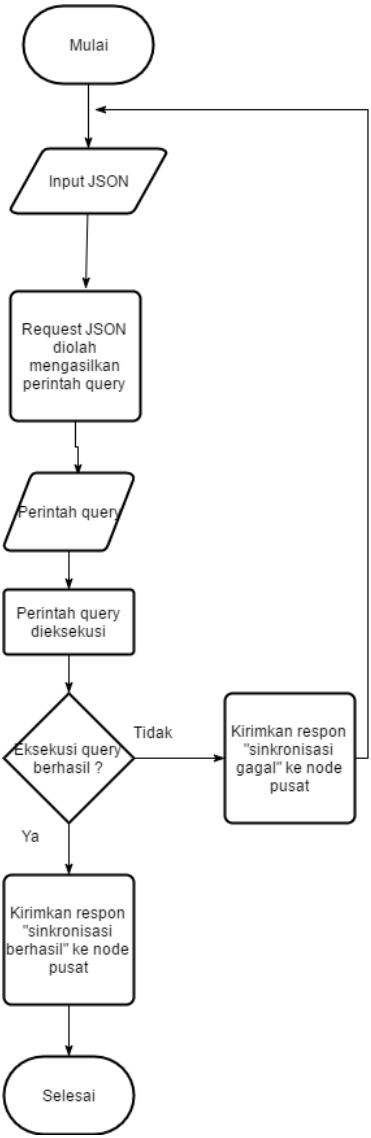
Jika *node* tersebut ada maka proses ekstraksi data dari *queue* dilakukan. Kemudian data hasil ekstraksi tersebut dikirimkan ke *slave node* tujuan, jika respon sukses didapat maka data tersebut dihapus pada queue. Jika respon dari slave *node* berupa kegagalan maka proses pengiriman request diulang. Proses ekstraksi data sampai pengiriman request ke *node* tujuan ini dilakukan secara periodik.

3.2.3 Proses Penerimaan oleh *Receiver*

Proses penerimaan oleh receiver ini adalah proses paling sederhana pada aplikasi. Proses ini hanya terjadi pada *slave node* tujuan. Proses ini dimulai dengan terjadinya pembuatan *web service* pada *slave node* tujuan. Pertama, input file *config* dibaca oleh kode kemudian dibuat web service dari file *config* tersebut yang ditunjukkan Gambar 3.10.



Gambar 3.10 Diagram alir proses pembuatan web service pada slave node



Gambar 3.11 Diagram alir penerimaan oleh web service pada slave node

Pada Gambar 3.11 ditunjukkan alur proses penerimaan data dari *node* pusat oleh *slave node*. Inputan berupa JSON yang diterima oleh *web service* diolah untuk mendapatkan perintah *query* yang ada. Kemudian perintah *query* tersebut deksekusi ke basis data lokal, jika eksekusi berhasil maka dikirimkan respon "sinkronisasi berhasil" oleh *web service* ke *node* pusat. Jika eksekusi gagal maka dikirimkan "sinkronisasi gagal" oleh web service pada *slave node* ke *node* pusat sehingga *node* pusat mengirimkan lagi post request JSON tersebut.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan yang akan digunakan untuk melakukan proses implementasi adalah MySQL dalam paket instalasi aplikasi XAMPP, Oracle 12c versi standar dan Python 2.7.13 versi 64-bit yang dipasang pada sistem operasi *Windows 10 64-bit* dan eksekusi program dijalankan melalui *command prompt Windows*.

4.2 Pemasangan *Library* Mysqldb, Cx_Oracle, Requests dan Flask pada Interpreter Python

Pada bagian ini dijelaskan mengenai cara pemasangan *libray* python yaitu mysqldb, cx_oracle, requests dan flask karena instalasi dasar pada python tidak memiliki *library* yang disebutkan. Langkah-langkah pemasangan *library* pada python adalah sebagai berikut:

1. Pastikan pada komputer yang digunakan telah terpasang python 2.7 versi 64-bit.
2. Jika belum, unduh aplikasi pada halaman https://www.python.org/download/releases/2.7/ lalu lakukan instalasi dari installer yang diunduh.
3. Jika python 2.7 telah terpasang maka lakukan konfigurasi pada *environment variable* pada sistem operasi windows dengan memasukkan "C:\Python27;C:\Python27\Scripts" pada *system variable*.

4. Untuk melakukan instalasi *library* mysqldb, buka *command prompt* dan ketikkan perintah “pip install mysqlclient”.
5. Untuk melakukan instalasi *library* cx_oracle, buka *command prompt* dan ketikkan perintah “pip install cx_oracle”.
6. Untuk melakukan instalasi *library requests*, buka *command prompt* dan ketikkan perintah “pip install requests”.
7. Untuk melakukan instalasi *library* flask, buka *command prompt* dan ketikkan perintah “pip install flask”
8. Untuk melakukan instalasi *library* locust, buka *command prompt* dan ketikkan perintah “pip install locustio”

4.3 Implementasi Pembacaan File Konfigurasi

Dalam pembuatan *capture* komponen diperlukan pembacaan file konfigurasi untuk mendapatkan *credential* akses pada basis data. Fungsi pembacaan file konfigurasi ditunjukkan pada Pseudocode 4.1. Implementasi dari Pseudocode 4.1 ditunjukkan pada Kode Sumber 1.

```
1  input config; initialize a empty dictionary
2  set options as section options from config
3  for length of options
4      assign section and options to empty dictionary
5  return dictionary
```

Pseudocode 4.1 Pembacaan file konfigurasi

Dengan fungsi pembacaan file konfigurasi tersebut kita mendapatkan *dictionary* yang berisi *credential* yang dapat digunakan mengakses basis data pada *host*.

4.4 Implementasi *Capture* Komponen

Untuk membuat komponen utama *capture* komponen yaitu *trigger* dan tabel *capture* ditunjukkan pada Pseudocode 4.2. Implementasi Pseudocode 4.2 dari ditunjukkan pada Kode Sumber 2.

```
1  set ip from ip database in config
2  set db from database name in config
3  set username from username in config
4  set password from password in config
5  set list of tabel from all table for synchronizing in config
6  set jenis from jenis in config
7  If  jenis = oracle
8      connect to oracle using cx_oracle with credential ip, db,
       username and password
9      for index = all member in list of table
10       create trigger in the table[index]
11       create copy table[index]
12       add new column action to copy table, now copy table can be
         called capture table
13     end for
14 else if jenis = mysql
15   connect to mysql using mysqldb with credential ip, db, username
and password
16     for index = all member in list of table
17       create trigger in the table[index]
18       create copy table[index]
19       add new column action to copy table, now copy table can be
         called capture table
20     end for
```

21	**end if**

Pseudocode 4.2 Fungsi untuk membuat capture komponen

Proses pada Pseudocode 4.2 dijelaskan sebagai berikut :

1. Pada baris 1 sampai 6 merupakan pengalokasian *credential* dari konfigurasi ke variabel ip, db, *username*, *password* dan *list of table*.
2. Baris 7 merupakan operasi percabangan jika variabel jenis berisi string "oracle" maka *librar*y yang digunakan untuk terkoneksi ke basis data adalah cx_oracle. Sebaliknya, jika variabel jenis berisi string "mysql" maka proses yang berjalan adalah proses 14, *librar*y yang digunakan untuk terkoneksi ke basis data adalah mysqldb.
3. Baris 10 merupakan proses untuk membuat *trigger* dengan berdasarkan nama tabel pada perulangan baris 9. *Trigger* ini menyimpan semua perubahan yang ada pada tabel *capture* yang dibuat pada baris 11 dan 12.
4. Baris 16 sampai 20 merupakan proses dari baris yang sama dengan baris 9 sampai 13.

4.5 Implementasi Sinkronisasi *Post Requests*

Pada bagian ini berisi *pseudocode* dari proses sinkronisasi *post request* yang sering digunakan pada aplikasi untuk mengirimkan perintah *query* ke *web service*. Hasil dari post request ini berupa respon dari web service yang digunakan menentukan tindakan yang diambil selanjutnya. Implementasi dari Pseudocode 4.3 ditunjukkan pada Kode Sumber 3.

1	**set** ip from ip database in config
2	**set** db from database name in config
3	**set** username from username in config
4	**set** password from password in config

```
5   set list of tabel from all table for synchronizing in config
6   set jenis from jenis in config
7   set list tujuan from address destination node in config minus list not
    detected node
8   if  jenis = oracle
9       connect to oracle using cx_oracle with credential ip, db,
        username and password
10      for index = all member in list of table
11          read data in capture table and store in data capture variable
12          for index = all member in data capture
13              If  action in data capture[index] = insert
14                  Set query = insert sql command in data capture[index]
                    for index = all member of list tujuan
15                      set payload = {list tujuan[index], name of
                        table[index], query}
16                      send request to destination
17                      If there are responses success insert to queue
18                          delete data capture[index]
19                      End if
20                  end for
21              Else if  action in data capture[index] = update
22                  Set query = update sql command in data
                    capture[index]
                    for index = all member of list tujuan
23                      set payload = {list tujuan[index], name of
                        table[index], query}
24                      send request to destination
25                      If there are responses success insert to queue
26                          delete data capture[index]
27                      End if
28                  end for
29              Else if  action in data capture[index] = insert
30                  Set query = sql command in data capture[index]
```

```
                    for index = all member of list tujuan
31                      set payload = {list tujuan[index], name of
                        table[index], query}
32                      send request to destination
33                      If there are responses success insert to queue
34                          delete data capture[index]
35                      End if
36                  end for
37              End if
38          end for
39      end for
40  else if jenis = mysql
41      connect to mysql using mysqldb with credential ip, db, username
    and password
42      for index = all member in list of table
43          read data in capture table and store in data capture variable
44              for index = all member in data capture
45              If  action in data capture[index] = insert
46                  Set query = insert sql command in data capture[index]
                    for index = all member of list tujuan
47                      set payload = {list tujuan[index], name of
                        table[index], query}
48                      send request to destination
49                      If there are responses success insert to queue
50                          delete data capture[index]
51                      End if
52                  end for
53              Else if  action in data capture[index] = update
54                  Set query = update sql command in data
                    capture[index]
```

```
                    for index = all member of list tujuan
55                      set payload = {list tujuan[index], name of
                        table[index], query}
56                      send request to destination
57                      If there are responses success insert to queue
58                          delete data capture[index]
59                      End if
60                  end for
61              If  action in data capture[index] = insert
62                  Set query = sql command in data capture[index]
                    for index = all member of list tujuan
63                      set payload = {list tujuan[index], name of
                        table[index], query}
64                      send request to destination
65                      If there are responses success insert to queue
66                          delete data capture[index]
67                      End if
68                  end for
69              End if
70          end for
71      end for
72  End if
```

Pseudocode 4.3 Sinkronisasi Post Request

Proses pada Pseudocode 4.3 dijelaskan sebagai berikut :

1. Pada baris 1 sampai 7 merupakan proses untuk mengisi variabel-varabel tersebut dari data-data konfigurasi pada file konfigurasi.
2. Pada baris 8 dan baris 40 merupakan proses percabangan yang ditentukan dari nilai jenis basis data yang digunakan dari file konfigurasi.
3. Baris 11 merupakan proses ekstraksi data dari tabel *capture* yang disimpan ke variabel data *capture* yang diulang sebayak tabel yang menjadi target sinkronisasi.
4. Baris 10 sampai baris 39 merupakan proses pengolahan ekstraksi data *capture*. Pada baris 13, 21 dan 29 memisahkan data pada data *capture* dari *action* yang dilakukan apakah *insert, update* dan *delete*. Kemudian baris 16 yang melakukan *packaging payload* dan proses yang mengirimkan *request* ke alamat tujuan. Jika respon yang didapatkan berupa data berhasil masuk ke dalam *queue* maka data *capture* pada index tersebut dihapus seperti pada baris 17 sampai 19.
5. Baris 42 sampai 71 merupakan proses yang sama dengan baris 10 sampai baris 39.

4.6 Implementasi *Test Post Request*

Pada bagian ini berisi *pseudocode test post request* yang merupakan fungsi untuk melakukan test request ke *node* tujuan untuk menentukan keberadaan *slave node* tujuan. Kemudian list *node* tujuan yang tidak dideteksi disimpan untuk dapakai di sinkronisasi *post request* sebagai alamat tujuan *node* yang tidak dikirimi request. Implementasi dari Pseudocode 4.4 ditunjukkan padaKode Sumber 4.

1	**set** list tujuan from all address destination *node* for synchronizing in config

```
2   set copy tujuan from copy of list tujuan
3       for index = range all member in list tujuan
4           Payload = {dummy}
5           send request to list tujuan[index]
6           If get response
7                   Delete copy tujuan[index]
8           End if
9     end for
10  return copy tujuan
```

Pseudocode 4.4 Test Post Request

Penjelasan pada Pseudocode 4.4 adalah sebagai berikut :

1. Pada baris 1 dan 2 merupakan inisialisasi variable yang digunakan pada fungsi ini.
2. Pada baris 4 sampai 9 merupakan proses pengecekkan *node* tujuan, jika *node* tujuan terdeteksi maka alamat *node* tujuan tersebut akan dihapus dari list copy pada baris ke 7 yang diulang sebanyak jumlah *node* tujuan pada file konfigurasi.
3. Pada baris ke 10 list tersebut menjani nilai return dari fungsi ini.

4.7 Implementasi *Web Service* pada Slave Node

Pada bagian ini berisi *pseudocode web service* yang merupakan fungsi logika web service pada *slave node*. Fungsi ini berguna untuk menangani *post request* dari *node* pusat. Implementasi dari Pseudocode 4.5 ditunjukkan pada Kode Sumber 5.

```
1   Initialize  content from request message received
2   Set query from query syntax in content
3   Set table from name table syntax in content
4   If  name table exist in local database
```

5	Do *query* in database local
6	**If** *query* success
7	**return** success response
8	**else**
9	**Return** failed message
10	**End if**
11	**else if** table name not exist
12	**return** table not exist message
13	**End if**

Pseudocode 4.5 Web service pada Slave Node

Pada Pseudocode 4.5 dapat dijelaskan sebagai berikut :

1. Baris 1, 2 dan 3 merupakan inisialisasi variable yang digunakan.
2. Pada baris 4 jika nama tabel pada basis data dan konten cocok maka lakukan maka masuk pe percabangan tersebut.
3. Pada baris 5 sampai 9 dilakukan jika nama tabel ada. Query pada konten dieksekusi pada basis data lokal. Jika hasilnya sukses maka kirimkan respon sukses. Jika eksekusi gagal maka kirimkan respon gagal.
4. Pada baris 11, jika tabel tidak ada maka dikirimkan respon tabel tidak ada.

4.8 Implementasi *Queue* pada Node Pusat

Untuk pembuatan *queue* dilakukan pada *node* pusat sebagai penyimpanan data sementara sebelum dikirimkan ke *node* tujuan. Proses pembuatan *queue* terdapat pada Pseudocode 4.6. Implementasi dari Pseudocode 4.6 ditunjukkan pada Kode Sumber 6.

```
1  set ip from ip database in config
2  set db from database name in config
3  set username from username in config
4  set password from password in config
5  set list of tabel from all table for synchronizing in config
6      connect to mysql using mysqldb with credential ip, db, username
       and password
7          create table queue
8  end if
```

Pseudocode 4.6 Pembuatan Queue

Perlu diketahui penulis menggunakan MySQL sebagai basis data yang digunakan sebagai *queue* karena basis data ini bersifat *open source.* Implementasi dari Pseudocode 4.7 ditunjukkan pada Kode Sumber 7.

```
1  set ip from ip database in config
2  set db from database name in config
3  set username from username in config
4  set password from password in config
5  set list of tabel from all table for synchronizing in config
6  set jenis from jenis in config
7  set list tujuan from address destination node in config minus list not
   detected node
8  connect to mysql using mysqldb with credential ip, db, username
   and password
9  data_queue = readed data in queue
10 for index = all member in data_queue only form detected node
11     query =data_queue[index][1] //to get query data only from queue
12     set payload = {query, name of table[index]}
13     send request to destination
```

14	**If** there are responses
15	delete data_queue[index]
16	**End if**
17	**End fo**r

Pseudocode 4.7 Distribusi dari Queue ke Slave Node

. Proses pada Pseudocode 4.7Pseudocode 4.3 dijelaskan sebagai berikut:

1. Pada baris 1 sampai 7 merupakan proses untuk mengisi variabel-varabel tersebut dari data-data konfigurasi pada file konfigurasi.
2. Pada baris 8 merupakan proses melakukan koneksi dengan *credential* ke DBMS MySQL.
3. Baris 9 merupakan proses penyaimpanan data dari queue ke variabel data_queue.
4. Baris 10 sampai baris 16 merupakan proses pengolahan data_queue. Kemudian baris 12 yang melakukan *packaging payload* dan proses yang mengirimkan request ke alamat tujuan.
5. Baris 14, jika respon *query* berhasil disinkronisasi di *node* tujuan atau slave maka data pada queue dengan index tersebut dihapus dari tabel queue seperti yang ditunjukkan pada baris 15.

Untuk merima data dari *master node*, pada queue diperlukan sebuah web service yang mengatur *request* data dari *master node* yang ditunjukkan pada Pseudocode 4.8. Implementasi dari Pseudocode 4.8 ditunjukkan pada Kode Sumber 8.

1	**Initialize** content from request message received
2	**Set** *query* from *query* syntax in content
3	**Set** tabel from name table syntax in content

4	**Set** tujuan from destination address in content
5	**Set** sumber ftom source address in content
6	Insert *query*, table, tujuan and sumber in database queue
7	**if** insert queue success
8	**return** success response
9	**else**
10	**return** failed message

Pseudocode 4.8 Web Service Queue

Pada Pseudocode 4.8 dapat dijelaskan sebagai berikut:

1. Baris 1, 2, 3, 4 dan 5 merupakan inisialisasi variabel yang digunakan.
2. Pada baris 6 data-data pada variabel *query*, tabel, tujuan dan sumber dimasukkan ke queue..
3. Pada baris 7 sampai 10 merupakan percabangan, jika data-data pada konten berhasil dimasukkan ke dalam queue maka dikirimkan respon sukses ke *master node*. Jika proses memasukkan data ke dalam queue gagal maka dikirimkan respon gagal ke *master node*.

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji performa serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji performa aplikasi. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

Tabel 5.1 Spesifikasi Perangkat Keras dan Lunak Komputer untuk Uji Coba

No	Nama Komputer	Prosesor	Tipe Sistem Operasi	Peran
1	Dekstop-0L5QMAV 192.168.43.234	Intel® Core™ i7-4720HQ CPU @ 2.60GHz	Windows 10 64-bit	*Master Node*
2	Bhisma 192.168.43.110	Intel® Core™ i3-3240 CPU @ 3.40GHz	Windows 8.1 64-bit	*Queue* / *Node* Pusat
3	User 192.168.43.176	Intel® Core™ i5-3317U CPU @ 1.70 GHz	Windows 8.1 64-bit	*Slave Node* (Oracle)
4	Regulus 192.168.43.179	Intel® Core™ 2 Duo CPU T5870 @ 2.00 GHz	Windows 7 64-bit	*Slave Node* (MySQL)

5.2 *Dataset* Uji Coba

Pada tugas akhir ini, digunakan basis data dokumen kedutaan sebagai *dataset* yang dimodifikasi sesuai skenario pengujian.

.

5.3 Skenario dan Evaluasi Pengujian

Skenario uji coba dilakukan dengan beberapa tahap uji coba yaitu uji coba fungsionalitas dan uji coba peforma. Penjelasan secara rinci skenario uji coba dijelaskan pada sub bab berikut ini.

5.3.1 Skenario Uji Coba Fungsionalitas

Uji coba dilakukan dengan menguji operasi-operasi *query* dan komponen utama aplikasi. Operasi sinkronisasi yang diuji adalah operasi *insert*, *update* dan *delete*. Selain itu, pengujian ini dilakukan dengan menggunakan tipe data *string*, *integer* dan *datetime*. Pengujian dilakukan dengan menguji apakah operasi *insert*, *update* dan *delete* dapat berjalan dengan baik pada aplikasi. Hal ini ditunjukkan dengan jumlah operasi yang setara antara *master node* dengan *slave node*. Pada Tabel 5.2 ditampilkan detail uji coba fungsionalitas secara ringkas dan jelas.

Tabel 5.2 Ringkasan Detail Uji Coba Fungsionalitas

No.	Tipe Data	Jenis Operasi	Jumlah Operasi	Hasil yang Diharapkan
1	Integer	Insert	500	Jumlah operasi *insert* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
		Update	500	Jumlah operasi *update* pada *master node* setara dengan
			1000	

No.	Tipe Data	Jenis Operasi	Jumlah Operasi	Hasil yang Diharapkan
				jumlah operasi pada *slave node*
		Delete	500	Jumlah operasi *delete* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
2	String	Insert	500	Jumlah operasi *insert* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
		Update	500	Jumlah operasi *update* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
		Delete	500	Jumlah operasi *delete* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
3	Datetime	Insert	500	Jumlah operasi *insert* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
		Update	500	Jumlah operasi *update* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	
		Delete	500	Jumlah operasi *delete* pada *master node* setara dengan jumlah operasi pada *slave node*
			1000	

5.3.2 Evaluasi Uji Coba Fungsionalitas

Setelah dilakukan uji coba fungsionalitas sesuai dengan skenario maka diperoleh hasil pada pada Tabel 5.3, Tabel 5.4 dan Tabel 5.5. Dari hasil uji coba pada Tabel 5.3 didapatkan hasil dimana proses sinkronisasi dengan operasi insert, update dan delete pada tipe data *string* berjalan dengan baik. Hal in dilihat dari operasi pada master node sudah setara dengan jumlah operasi pada *slave node* Oracle dan *slave node* MySQL.

Tabel 5.3 Hasil Uji Coba Fungsionalitas Tipe Data String

No.	Jenis Operasi	Jumlah Operasi pada Master Node	Jumlah Operasi pada Slave Node (Oracle)	Jumlah Operasi pada Slave Node (MySQL)	Hasil Uji Coba
1	Insert	500	500	500	Sukses
		1000	1000	1000	Sukses
2	Update	500	500	500	Sukses
		1000	1000	1000	Sukses
3	Delete	500	500	500	Sukses
		1000	1000	1000	Sukses

Untuk operasi hasil uji coba fungsional dengan tipe data *integer* dapat dilihat pada Tabel 5.4. Pada Tabel 5.4 ditunjukkan bahwa operasi-operasi sinkronisasi pada tipe data *integer* seperti *insert*, *update* dan *delete* sudah dapat berjalan dengan baik. Hal ini dikarenakan jumlah operasi pada *master node* sudah setara dengan jumlah operasi pada *slave node* Oracle dan *slave node* MySQL.

Tabel 5.4 Hasil Uji Coba Fungsionalitas Tipe Data Integer

No.	Jenis Operasi	Jumlah Operasi pada Master Node	Jumlah Operasi pada Slave Node (Oracle)	Jumlah Operasi pada Slave Node (MySQL)	Hasil Uji Coba
1	Insert	500	500	500	Sukses
		1000	1000	1000	Sukses
2	Update	500	500	500	Sukses
		1000	1000	1000	Sukses
3	Delete	500	500	500	Sukses
		1000	1000	1000	Sukses

Pada Tabel 5.5 ditunjukkan hasil uji coba fungsionalitas sinkronisasi tipe data *datetime* dengan menggunakan operasi *insert*, *update* dan *delete*. Dapat dilihat bahwa operasi *insert*, *update* dan *delete* pada tipe data *datetime* adalah gagal. Hal ini dikarenakan format *datetime* pada Oracle dan MySQL tidak sama. Dimana format *datetime* Oracle adalah DD-MON-YY sedangkan format *datetime* MySQL adalah YYYY-MM-DD HH:MI:SS. Untuk menyamakan format *datetime* ini digunakan fungsi *to_date* pada Oracle. Namun, fungsi *to_date* ini setelah dikirimkan dari *queue node* ke *slave node* terjadi kegagalan eksekusi pada *slave node* Oracle. Hal ini dikarenakan fungsi *to_date* dibaca sebagai string bukan sebagai fungsi oleh Oracle.

Tabel 5.5 Hasil Uji Coba Fungsionalitas Tipe Data Datetime

No.	Jenis Operasi	Jumlah Operasi pada Master Node	Jumlah Operasi pada Slave Node (Oracle)	Jumlah Operasi pada Slave Node (MySQL)	Hasil Uji Coba
1	Insert	500	0	500	Gagal

No.	Jenis Operasi	Jumlah Operasi pada Master Node	Jumlah Operasi pada Slave Node (Oracle)	Jumlah Operasi pada Slave Node (MySQL)	Hasil Uji Coba
		1000	0	1000	Gagal
2	Update	500	0	500	Gagal
		1000	0	1000	Gagal
3	Delete	500	0	500	Gagal
		1000	0	1000	Gagal

5.3.3 Skenario Uji Coba Ketahanan

Uji coba dilakukan dengan menguji ketahanan atau *endurance* dari *web service*. *Web service* yang dimaksud adalah *web service* yang berada pada node pusat. *Web service* pada *queue* bertugas untuk menangani *request post* yang dikirimkan untuk selanjutnya dimasukkan ke dalam *queue*. Pengujian dilakukan dengan membuat sejumlah pengguna melakukan koneksi ke *web service* pada node pusat dan mengirimkan *request post* secara serentak yang dibantu dengan *tool locustio*. Setelah itu, data hasil *monitoring request post* dicatat berapa jumlah *request* yang berhasil ditangani dan berapa yang gagal ditangani oleh *web service*. Adapun jumlah total *request* yang dilakukan uji coba adalah 5000, 7500 dan 10000 *request*. Untuk jumlah pengguna yang melakukan koneksi secara simultan adalah 50, 100, 150, 200 dan 250 pengguna.

5.3.4 Evaluasi Uji Coba Ketahanan

Setelah dilakukan uji coba sesuai dengan skenario didapatkan hasil uji coba pada Tabel 5.6, Tabel 5.7 dan Tabel 5.8. Setiap hasil uji coba merupakan hasil dari rata-rata lima kali uji coba. Rasio *request* diterima adalah nilai rasio yang didapatkan dari nilai *request* diterima dibagi total *request* lalu dikalikan 100%. Rasio *request drop* adalah nilai rasio yang didapatkan dari nilai *request drop* dibagi total *request* lalu dikalikan 100%.

Pada Tabel 5.6 ditunjukkan data hasil uji coba ketahanan dengan total request berjumlah 5000 *request*. Hasil uji coba pada Tabel 5.6 menunjukkan dengan jumlah pengguna 50 dan 100 rasio *request* yang diterima bernilai 100%. Hal ini menunjukkan semua *request* berhasil ditangani oleh *web service*. Kemudian, untuk jumlah pengguna 150 mulai terjadi *request* yang drop sejumlah 53.436 %. Dengan jumlah *request* yang diterima bernilai 46.564 %. Untuk jumlah pengguna 200 rasio *request* yang drop meningkat menjadi 74.988%. Dengan jumlah *request* yang diterima menurun dengan nilai 25.012 %. Pada jumlah pengguna 250 rasio *request* yang drop meningkat menjadi 85.784%. Dengan jumlah *request* yang diterima menurun dengan nilai 14.216 %.

Tabel 5.6 Hasil Uji Coba Ketahanan dengan 5000 Request

No	Jumlah Pengguna	Request Diterima	Request Drop	Rasio Diterima (%)	Rasio Drop (%)
1	50	5000	0	100	0
2	100	5000	0	100	0
3	150	2328	2672	46.564	53.436
4	200	1251	3749	25.012	74.988
5	250	711	4289	14.216	85.784

Pada Tabel 5.7 ditunjukkan data hasil uji coba ketahanan dengan total request berjumlah 7500 *request*. Hasil uji coba pada Tabel 5.7 menunjukkan dengan jumlah pengguna 50 dan 100 rasio *request* yang diterima bernilai 100%. Hal ini menunjukkan semua *request* berhasil ditangani oleh *web service*. Kemudian, untuk jumlah pengguna 150 mulai terjadi *request* yang drop sejumlah 53.781 %. Dengan jumlah *request* yang diterima bernilai 46.219 %. Untuk jumlah pengguna 200 rasio *request* yang drop meningkat menjadi 80.299%. Dengan jumlah *request* yang diterima menurun dengan nilai 19.701 %. Pada jumlah pengguna 250 rasio *request* yang drop meningkat menjadi 82.472%. Dengan jumlah *request* yang diterima menurun dengan nilai 17.528 %.

Tabel 5.7 Hasil Uji Coba Ketahanan dengan 7500 Request

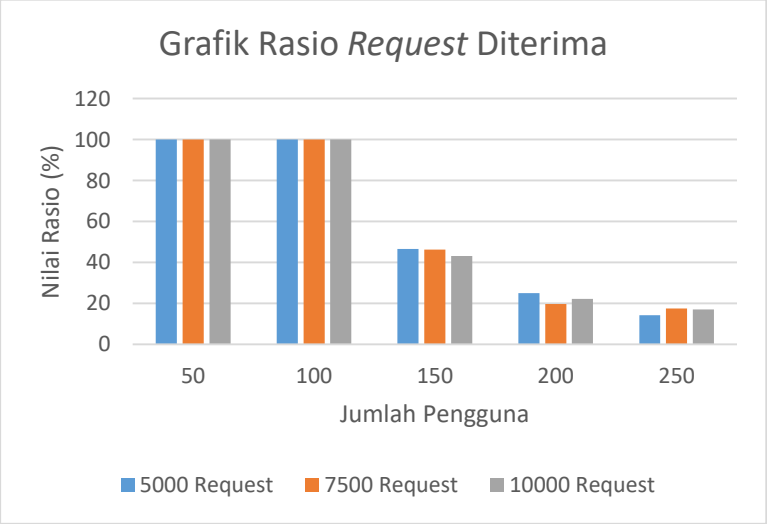
No	Jumlah Pengguna	Request Diterima	Request Drop	Rasio Diterima (%)	Rasio Drop (%)
1	50	7500	0	100	0
2	100	7500	0	100	0
3	150	3466	4034	46.219	53.781
4	200	1478	6022	19.701	80.299
5	250	1315	6185	17.528	82.472

Pada Tabel 5.8 ditunjukkan data hasil uji coba ketahanan dengan total request berjumlah 1000 *request*. Hasil uji coba pada Tabel 5.8 menunjukkan dengan jumlah pengguna 50 dan 100 rasio *request* yang diterima bernilai 100% dan rasio drop bernilai 0 %. Hal ini menunjukkan semua *request* berhasil ditangani oleh *web service*. Kemudian, untuk jumlah pengguna 150 mulai terjadi *request* yang drop sejumlah 56.886 %. Dengan jumlah *request* yang diterima bernilai 43.114 %. Untuk jumlah pengguna 200 rasio *request* yang drop meningkat menjadi 77.868%. Dengan jumlah

request yang diterima menurun dengan nilai 22.132 %. Pada jumlah pengguna 250 rasio *request* yang drop meningkat menjadi 82.966%. Dengan jumlah *request* yang diterima menurun dengan nilai 17.034 %.

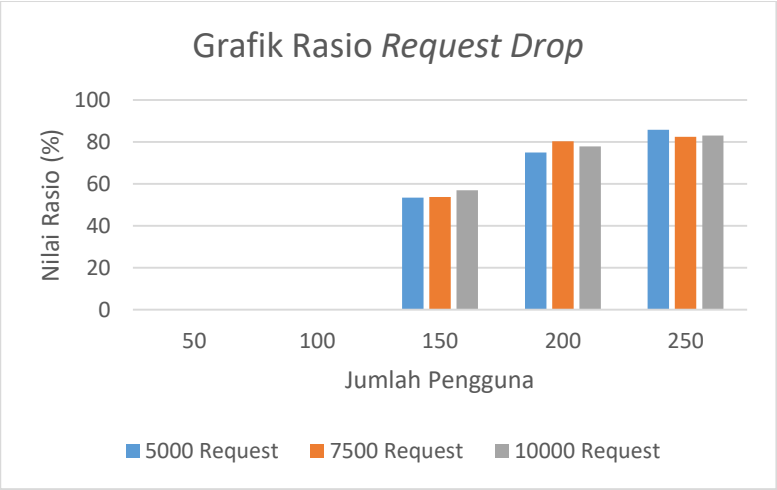
Tabel 5.8 Hasil Uji Coba Ketahanan dengan 10000 Request

No	Jumlah Pengguna	Request Diterima	Request Drop	Rasio Diterima (%)	Rasio Drop (%)
1	50	10000	0	100	0
2	100	10000	0	100	0
3	150	4311	5689	43.114	56.886
4	200	2213	7787	22.132	77.868
5	250	1703	8297	17.034	82.966



Gambar 5.1 Grafik Rasio Request Diterima

Pada grafik Gambar 5.1 diilustrasikan perbandingan rasio *request* yang diterima *web service*. Grafik ini membandingkan nilai rasio *request* denga total *request* 5000, 7500 dan 10000 *request*. Dapat diperhatikan dengan jumlah pengguna 50, nilai rasio diterima bernilai 100 % untuk total *request* 5000, 7500 dan 10000. Kemudian, dengan jumlah pengguna 100, nilai rasio diterima masih bernilai 100 % untuk total *request* 5000, 7500 dan 10000. Namun, pada jumlah pengguna 150, 200 dan 250 terjadi penurunan nilai rasio diterima yang menunjukkan pada ketiga jumlah pengguna ini web service mulai tidak mampu menangani semua *request*.



Gambar 5.2 Grafik Rasio Request Drop

Sebaliknya pada Gambar 5.2, nilai rasio drop bernilai 0% pada pengguna 50 dan 100. Lalu terjadi peningkatan rasio *request* yang *drop* pada pengguna berjumlah 150, 200 dan 250. Dari perbandingan rasio pada Gambar 5.1 dan Gambar 5.2 dapat diperhatikan jumlah pengguna maksimal dimana request 100 % diterima dan 0% *request drop* adalah saat pengguna berjumlah 100.

Hal ini menunjukkan beban maksimal yang dapat ditangani oleh *web service* pada node pusat adalah 100 pengguna secara simultan.

5.3.5 Skenario Uji Coba Performa

Uji coba ini dilakukan untuk menguji bagaimana performa aplikasi yang telah diimplementasikan. Uji coba akan didasarkan pada beberapa skenario untuk menguji performa aplikasi.

Skenario pengujian terdiri dari enam skenario pengujian yaitu:

1. Skenario efek variasi kolom terhadap performa kecepatan sinkronisasi aplikasi. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Semua *query* yang digunakan adalah *insert query*. Dengan dua *node* tujuan.
    a. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah *integer* serta dua *node* tujuan.
    b. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah *string* serta dua *node* tujuan.
    c. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.
2. Skenario efek variasi jumlah *query* terhadap performa kecepatan sinkronisasi aplikasi. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Semua *query* yang digunakan adalah *insert query*. Dengan dua *node* tujuan.
    a. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *integer* serta dua *node* tujuan.
    b. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *string* serta dua *node* tujuan.

   c. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.
3. Skenario efek variasi jumlah tabel terhadap performa kecepatan sinkronisasi aplikasi. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Semua *query* yang digunakan adalah *insert query*. Dengan dua *node* tujuan.
   a. Dengan 5 kolom dan 500 *insert query* pada setiap tabel serta semua data dalam tabel adalah *integer* serta dua *node* tujuan.
   b. Dengan 5 kolom dan 500 *insert query* pada setiap tabel serta semua data dalam tabel adalah *string* serta dua *node* tujuan.
   c. Dengan 5 kolom dan 500 *insert query* pada setiap tabel serta semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.
4. Skenario efek variasi kolom terhadap performa kecepatan sinkronisasi aplikasi. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Semua *query* yang digunakan adalah *update query*.Dengan dua *node* tujuan.
   a. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah *integer* serta dua *node* tujuan.
   b. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah *string* serta dua *node* tujuan.
   c. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.
5. Skenario efek variasi jumlah *query* terhadap performa kecepatan sinkronisasi aplikasi. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Semua *query* yang digunakan adalah *update query*. Dengan dua *node* tujuan.

a. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *integer* serta dua *node* tujuan.
b. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *string* serta dua *node* tujuan.
c. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.

6. Skenario efek variasi jumlah tabel terhadap performa kecepatan sinkronisasi aplikasi. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Semua *query* yang digunakan adalah *update query*. Dengan dua *node* tujuan.
   a. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua data dalam tabel adalah *integer* serta dua *node* tujuan.
   b. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua data dalam tabel adalah *string* serta dua *node* tujuan.
   c. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan.

Tabel 5.9 Ringkasan Detail Skenario Uji Coba Performa

No	Nama Skenario	Jenis Query	Tipe Data	Tujuan
1	Skenario 1a	Insert	Integer	Efek variasi kolom terhadap performa
2	Skenario 1b	Insert	String	Efek variasi kolom terhadap performa
3	Skenario 1c	Insert	String dan Integer	Efek variasi kolom terhadap performa

No	Nama Skenario	Jenis Query	Tipe Data	Tujuan
4	Skenario 2a	Insert	Integer	Efek variasi jumlah *query* terhadap performa
5	Skenario 2b	Insert	String	Efek variasi jumlah *query* terhadap performa
6	Skenario 2c	Insert	String dan Integer	Efek variasi jumlah *query* terhadap performa
7	Skenario 3a	Insert	Integer	Efek variasi jumlah tabel terhadap performa
8	Skenario 3b	Insert	String	Efek variasi jumlah tabel terhadap performa
9	Skenario 3c	Insert	String dan Integer	Efek variasi jumlah tabel terhadap performa
10	Skenario 4a	Update	Integer	Efek variasi kolom terhadap performa
11	Skenario 4b	Update	String	Efek variasi kolom terhadap performa
12	Skenario 4c	Update	String dan Integer	Efek variasi kolom terhadap performa
13	Skenario 5a	Update	Integer	Efek variasi jumlah *query* terhadap performa
14	Skenario 5b	Update	String	Efek variasi jumlah *query* terhadap performa

No	Nama Skenario	Jenis Query	Tipe Data	Tujuan
15	Skenario 5c	Update	String dan Integer	Efek variasi jumlah *query* terhadap performa
16	Skenario 6a	Update	Integer	Efek variasi jumlah tabel terhadap performa
17	Skenario 6b	Update	String	Efek variasi jumlah tabel terhadap performa
18	Skenario 6c	Update	String dan Integer	Efek variasi jumlah tabel terhadap performa

Pada Tabel 5.9 ditampilkan detail skenario uji coba secara ringkas dan jelas.

5.3.6 Skenario Uji Coba Performa 1

Skenario uji coba performa 1 adalah pengujian dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Waktu yang dihitung adalah waktu dimana semua *query* berhasil dikirimkan dari *master node* ke queue *node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *insert*.

5.3.6.1 Skenario Uji Coba Performa 1a

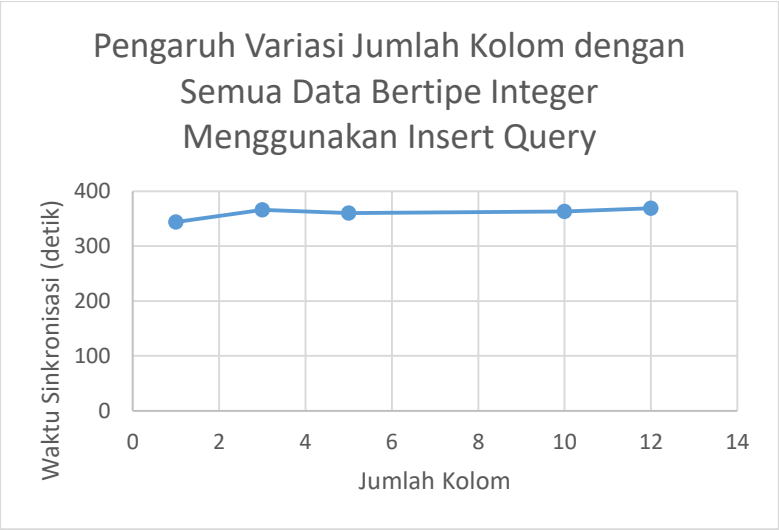
Skenario uji coba performa 1a adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang

digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.6.2 Evaluasi Uji Coba Performa 1a

Setelah dilakukan uji coba seperti skenario uji coba performa 1a didapatkan hasil waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan pada grafik Gambar 5.3 dan Tabel 5.10. Pada grafik yang ditunjukkan pada Gambar 5.3 jika diamati dapat diperhatikan perubahan yang terjadi pada setiap titik tidak ada perubahan yang tajam sehingga menimbulkan garis tren konstan pada grafik. Meskipun variasi jumlah kolom juga mempengaruhi perubahan waktu sinkronisasi tetapi dari analisa pada grafik pengaruhnya masih rendah. Berdasarkan kolom rata-rata selisih pada Tabel 5.10. Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi.

Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba bernilai rendah, selain itu rata-rata dari rata-rata selisih hanya bernilai 3.106 detik yang menunjukkan uji coba skenario performa 1a ini tidak memiliki pengaruh yang besar pada performa aplikasi dilihat dari nilai rata-rata selisih setiap variasi uji coba skenario 1a.



Gambar 5.3 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 1a

Pada Tabel 5.10 dapat diperhatikan pada kolom rata-rata selisih, rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai 10.974 detik. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai -2.781 detik, nilai minus ini berarti terjadi peningkatan perfoma pada aplikasi dan waktu sinkronisasi menurun pada uji coba 5 kolom. Rata-rata selisih uji coba 5 kolom dengan 10 kolom bernilai 1.208 detik, nilai rata-rata selisih yang rendah ini dikarenakan rendahnya perbedaan antara kedua variasi uji coba ini. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai 3.021 detik, terjadi peningkatan pada waktu sinkronisasi pada 12 kolom meskipun nilainya tergolong rendah.

Tabel 5.10 Hasil Uji Coba Performa Skenario 1a

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	344.062	-
2	3	366.009	10.974
3	5	360.447	-2.781
4	10	362.864	1.208
5	12	368.906	3.021
Rata-rata			3.106

5.3.6.3 Skenario Uji Coba Performa 1b

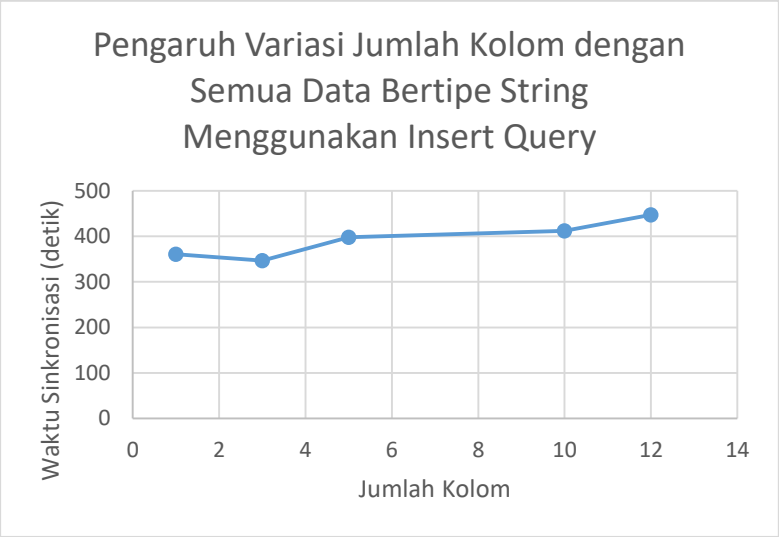
Skenario uji coba performa 1b adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.6.4 Evaluasi Uji Coba Performa Performa 1b

Setelah dilakukan uji coba seperti pada skenario 1b didapatkan hasil waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba berupa grafik dan tabel yang ditunjukkan pada Gambar 5.4 dan Tabel 5.11. Pada grafik Gambar 5.4 dapat diamati lebih meningkatnya perubahan waktu sinkronisasi dibandingkan waktu sinkronisasi pada uji coba performa 1a.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata

untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap variasi uji coba pada Tabel 5.11 tidak terlalu besar, selain itu rata-rata dari rata-rata selisih hanya bernilai 10.801 detik yang menunjukkan uji coba skenario performa 1b ini tidak memiliki pengaruh yang signifikan pada performa aplikasi. Namun jika dibandingkan dengan skenario 1a, skenario performa 1b memiliki rata-rata selisih waktu sinkronisasi yang lebih tinggi dari skenario performa 1a. Hal ini dikarenakan semua data yang disinkronisasikan bertipe *string* yang juga memiliki ukuran data lebih tinggi setiap baris dan kolomnya dibandingkan *integer* pada skenario performa 1a.



Gambar 5.4 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 1b

Tabel 5.11 Hasil Uji Coba Performa Skenario 1b

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	360.989	-
2	3	346.694	-7.148
3	5	397.787	25.547
4	10	412.200	7.207
5	12	447.399	17.600
Rata-rata			10.801

Pada Tabel 5.11 dapat diperhatikan pada kolom rata-rata selisih, rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai -7.148 detik, nilai minus ini berarti terjadi peningkatan perfoma pada aplikasi dan waktu sinkronisasi menurun pada uji coba 3 kolom. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai 25.547 detik, terjadi peningkatan pada waktu sinkronisasi pada 5 kolom. Rata-rata selisih uji coba 5 kolom dengan 10 kolom bernilai 7.207 detik. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai 17.600 detik, terjadi peningkatan pada waktu sinkronisasi pada 12 kolom meskipun nilainya tergolong rendah.

5.3.6.5 Skenario Uji Coba Performa 1c

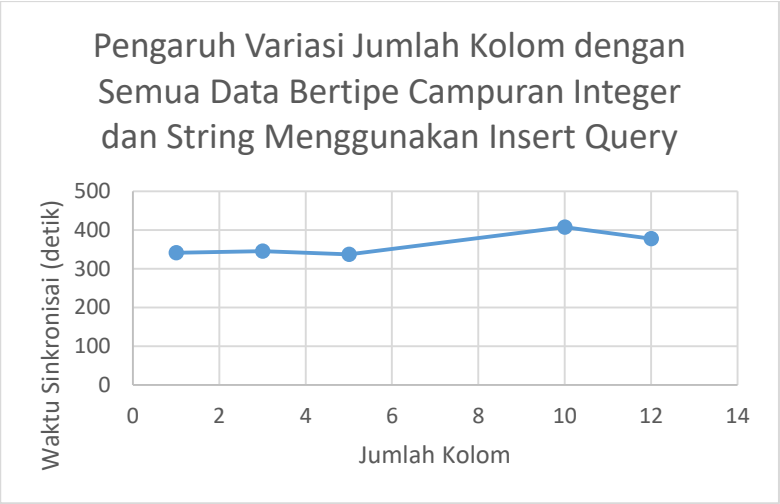
Skenario uji coba performa 1c adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *insert query* dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node*

dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.6.6 Evaluasi Uji Coba Performa 1c

Setelah dilakukan uji coba seperti pada skenario 1c didapatkan hasil waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang disajikan pada grafik Gambar 5.5 dan Tabel 5.12. Pada grafik Gambar 5.5 dapat diamati perubahan waktu sinkronisasi. Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.12 tidak terlalu signifikan, selain itu rata-rata dari rata-rata selisih hanya bernilai 4.574 detik yang menunjukkan uji coba skenario 1b ini tidak memiliki pengaruh yang signifikan pada performa aplikasi.

Rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai 2.419 detik. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai -4.344 detik, nilai minus ini berarti terjadi sedikit peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 5 kolom. Rata-rata selisih uji coba 5 kolom dengan 10 kolom bernilai 35.287 detik, terjadi peningkatan pada waktu sinkronisasi pada 10 kolom dengan nilai tergolong tinggi. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai -15.066 detik, nilai minus ini berarti terjadi peningkatan perfoma pada aplikasi dan waktu sinkronisasi menurun pada uji coba 12 kolom.



Gambar 5.5 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 1c

Tabel 5.12 Hasil Uji Coba Performa Skenario 1c

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	340.679	-
2	3	345.517	2.419
3	5	336.828	-4.344
4	10	407.403	35.287
5	12	377.272	-15.066
Rata-rata			4.574

5.3.7 Skenario Uji Coba Performa 2

Skenario uji coba performa 2 adalah pengujian dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Waktu yang dihitung adalah waktu dimana semua *query* berhasil dikirimkan dari *master node* ke queue *node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *insert*.

5.3.7.1 Skenario Uji Coba Performa 2a

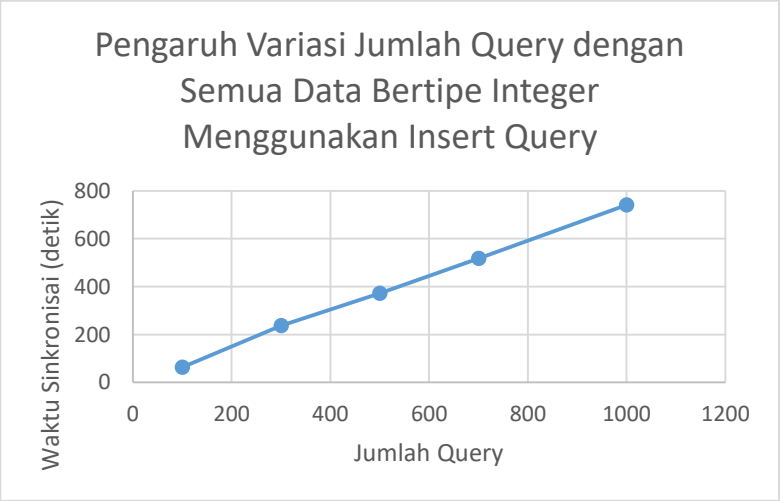
Skenario uji coba performa 2a adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.7.2 Evaluasi Uji Coba Performa 2a

Setelah dilakukan uji coba seperti pada skenario 2a didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.6 dan pada Tabel 5.13. Pada grafik Gambar 5.6 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 63.501 detik. Untuk titik kedua dimana jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar 237.235 detik. Untuk titik ketiga dimana jumlah *query* adalah 500, waktu sinkronisasi diperlukan sekitar 371.818 detik. Untuk titik

keempat dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 517.267 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 741.003 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.13, rata-rata dari rata-rata selisih bernilai 84.688 detik yang menunjukkan uji coba skenario 2a ini memiliki pengaruh yang signifikan pada performa aplikasi. Meningkatnya jumlah *query* pada masing-masing tabel menyebabkan meningkat pula beban kerja *queue* untuk pendistribusian *query*. Hasilnya distribusi *query* pun menjadi lebih lama sehingga waktu sinkronisasi terus terjadi peningkatan seperti pada grafik.



Gambar 5.6 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 2a

Tabel 5.13 Hasil Uji Coba Performa Skenario 2a

No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata-rata (deik)
1	100	63.501	-
2	300	237.235	86.867
3	500	371.818	67.291
4	700	517.267	72.724
5	1000	741.003	111.868
Rata-rata			84.688

5.3.7.3 Skenario Uji Coba Performa 2b

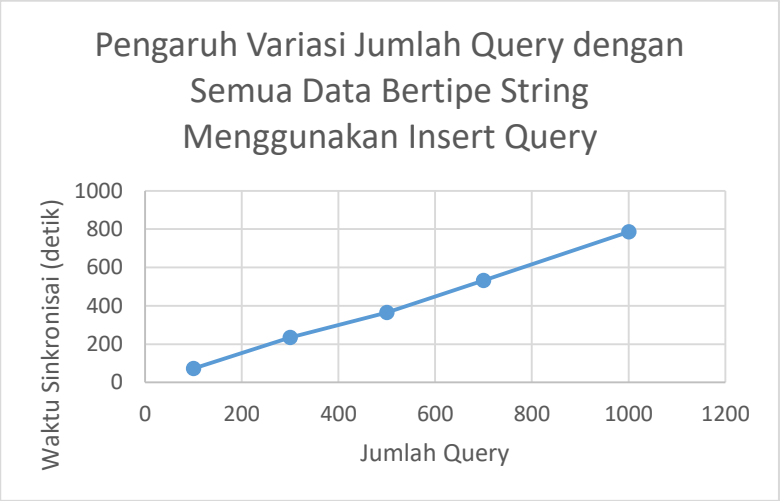
Skenario uji coba performa 2b adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.7.4 Evaluasi Uji Coba Performa 2b

Setelah dilakukan uji coba seperti pada skenario 2b didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.7 dan pada Tabel 5.14. Pada grafik Gambar 5.7 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 72.827 detik. Untuk titik kedua dimana jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar 233.635 detik. Untuk titik ketiga dimana jumlah *query* adalah 500,

waktu sinkronisasi diperlukan sekitar 364.151 detik. Untuk titik keempat dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 531.364 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 786.269 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan rata-rata selisih pada setiap uji coba pada Tabel 5.14 peningkatan waktu sinkronisasi pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 89.180 detik yang menunjukkan uji coba skenario 2b ini memiliki pengaruh yang signifikan pada performa aplikasi. Variasi jumlah *query* ini meningkatkan kerja *queue* sehingga distribusi menjadi bertambah lama setiap terjadi peningkatan jumlah *query*.



Gambar 5.7 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 2b

Tabel 5.14 Hasil Uji Coba Performa Skenario 2b

No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	100	72.827	-
2	300	233.635	80.404
3	500	364.151	65.258
4	700	531.364	83.606
5	1000	786.269	127.453
Rata-rata			89.180

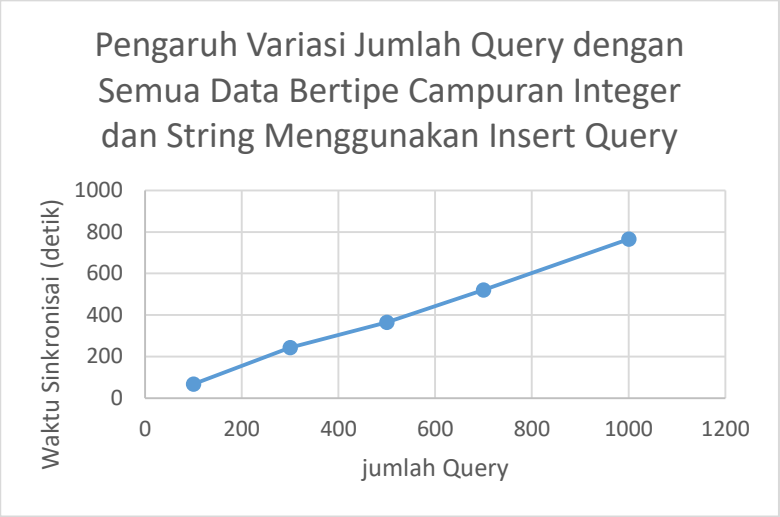
5.3.7.5 Skenario Uji Coba Performa 2c

Skenario uji coba performa 2c adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah campuran *integer* dan *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.7.6 Evaluasi Uji Coba 2c

Setelah dilakukan uji coba seperti pada skenario 2c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.8 dan pada Tabel 5.15. Pada grafik Gambar 5.8 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 67.869 detik. Untuk titik kedua dimana

jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar 243.127 detik.



Gambar 5.8 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 2c

Tabel 5.15 Hasil Uji Coba Performa Skenario 2c

No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	100	67.869	-
2	300	243.127	87.629
3	500	364.953	60.913
4	700	520.909	77.978
5	1000	766.170	122.630
Rata-rata			87.288

Untuk titik ketiga dimana jumlah *query* adalah 500, waktu sinkronisasi diperlukan sekitar 364.953 detik. Untuk titik keempat

dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 520.909 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 766.170 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.15, rata-rata dari rata-rata selisih bernilai 87.288 detik yang menunjukkan uji coba skenario 2c ini memiliki pengaruh yang signifikan pada performa aplikasi.

Peningkatan pada waktu sinkronisasi yang membuat grafik terus meningkat ini disebabkan oleh variasi jumlah *query* yang mempengaruhi kerja *queue*. Hal ini karena setiap terjadi peningkatan jumlah *query* maka *queue* pun bertambah banyak sehingga *query* yang didistribusikan pun meningkat sehingga waktu sinkronisasi pun juga ikut meningkat.

5.3.8 Skenario Uji Coba Performa 3

Skenario uji coba performa 3 adalah pengujian dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Waktu yang dihitung adalah waktu dimana semua *query* berhasil dikirimkan dari *master node* ke queue *node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *insert*.

5.3.8.1 Skenario Uji Coba Performa 3a

Skenario uji coba performa 3a adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500

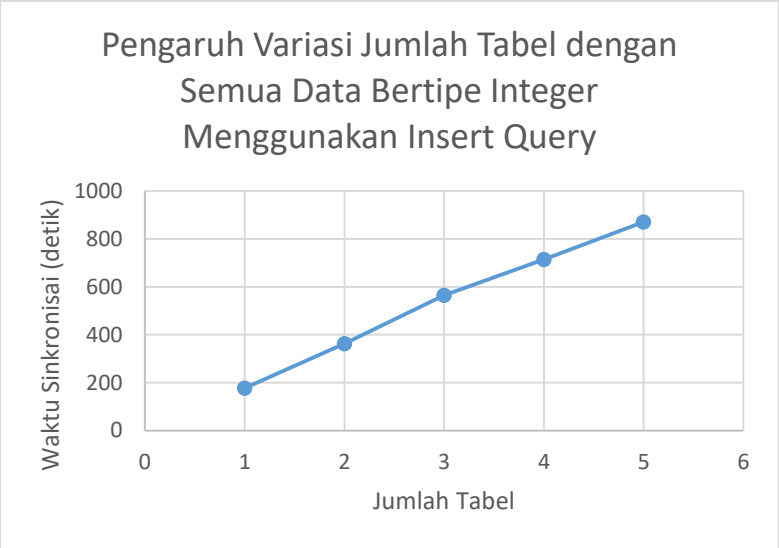
insert query pada setiap tabel serta semua tipe data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.8.2 Evaluasi Uji Coba Performa 3a

Setelah dilakukan uji coba seperti pada skenario 3a didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.9 dan pada Tabel 5.16. Pada grafik Gambar 5.9 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 177.893 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 362.961 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi diperlukan sekitar 564.911 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 870.498 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 860.207 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan rata-rata selisih pada setiap variasi uji coba pada Tabel 5.16, itu rata-rata dari rata-rata selisih bernilai 86.576 detik yang menunjukkan uji coba skenario 3a ini memiliki pengaruh yang signifikan pada performa aplikasi.

Sehingga efek variasi jumlah tabel dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat. Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.



Gambar 5.9 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 3a

Tabel 5.16 Hasil Uji Coba Performa Skenario 3a

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	177.893	-
2	2	362.961	92.534
3	3	564.911	100.975
4	4	713.665	74.377
5	5	870.498	78.417
Rata-rata			86.576

5.3.8.3 Skenario Uji Coba Performa 3b

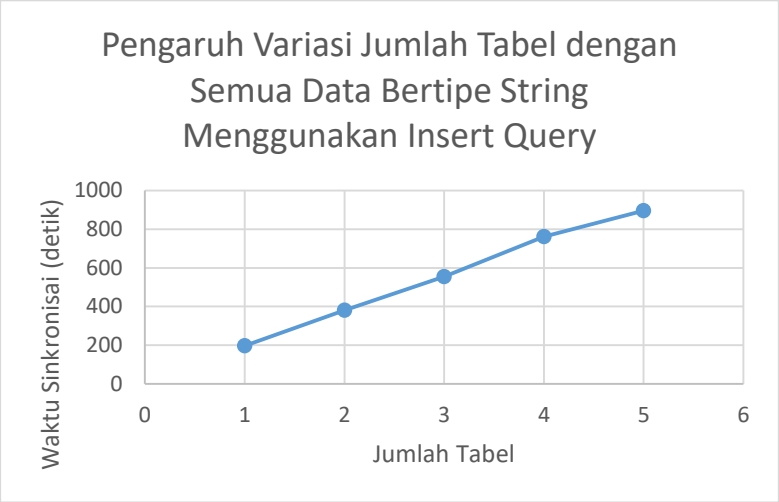
Skenario uji coba performa 3b adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500 *insert query* pada setiap tabel serta semua tipe data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.8.4 Evaluasi Uji Coba Performa 3b

Setelah dilakukan uji coba seperti pada skenario 3b didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.10 dan pada Tabel 5.17. Pada grafik Gambar 5.10 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 197.969 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 381.619 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi diperlukan sekitar 554.211 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 761.478 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 895.437 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan rata-rata selisih pada setiap variasi uji coba pada Tabel 5.17 memiliki nilai yang tergolong tinggi, rata-rata dari rata-rata selisih bernilai 87.183 detik yang menunjukkan uji coba skenario 3b ini memiliki pengaruh yang signifikan pada performa aplikasi.

Sehingga efek variasi tabel pada dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat. Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.



Gambar 5.10 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 3b

Tabel 5.17 Hasil Uji Coba Performa Skenario 3b

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	197.969	-
2	2	381.619	91.825
3	3	554.211	86.296

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
4	4	761.478	103.634
5	5	895.437	66.979
Rata-rata			87.183

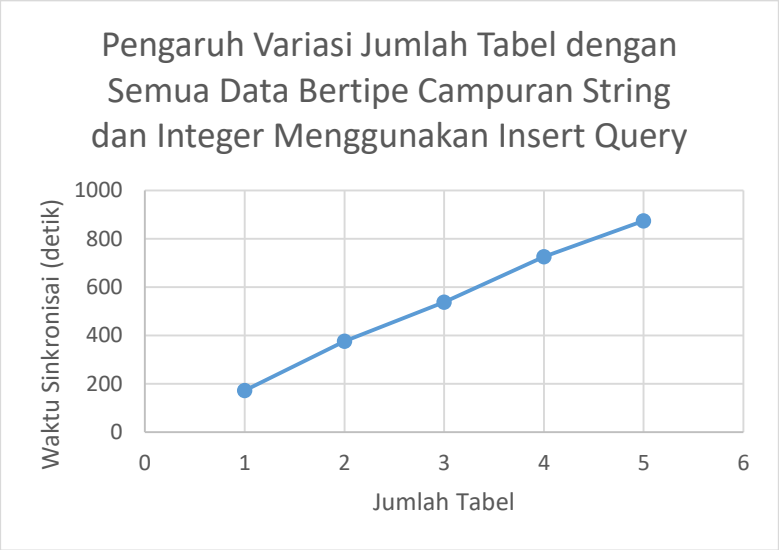
5.3.8.5 Skenario Uji Coba Performa 3c

Skenario uji coba performa 3c adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500 *insert query* pada setiap tabel serta semua data dalam tabel adalah campuran *integer* dan *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.8.6 Evaluasi Uji Coba Performa 3c

Setelah dilakukan uji coba seperti pada skenario 3c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.10 dan pada Tabel 5.17. Pada grafik Gambar 5.10 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 171.420 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 375.712 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi diperlukan sekitar 537.199 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 725.550 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 874.214 detik. Dapat diamati

pada grafik Tabel 5.16 setiap meningkatnya variasi jumlah tabel maka meningkat pula waktu sinkronisasinya.



Gambar 5.11 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 3c

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.18, rata-rata dari rata-rata selisih bernilai 87.849 detik yang menunjukkan uji coba skenario 3c ini memiliki pengaruh yang signifikan pada performa aplikasi. Sehingga efek variasi jumlah tabel pada dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat.

Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.

Tabel 5.18 Hasil Uji Coba Performa Skenario 3c

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	171.420	-
2	2	375.712	102.146
3	3	537.199	80.743
4	4	725.550	94.176
5	5	874.214	74.332
Rata-rata			87.849

5.3.9 Skenario Uji Coba Performa 4

Skenario uji coba performa 4 adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Waktu yang dihitung adalah waktu dimana semua *query* berhasil dikirimkan dari *master node* ke queue *node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *update*.

5.3.9.1 Skenario Uji Coba Performa 4a

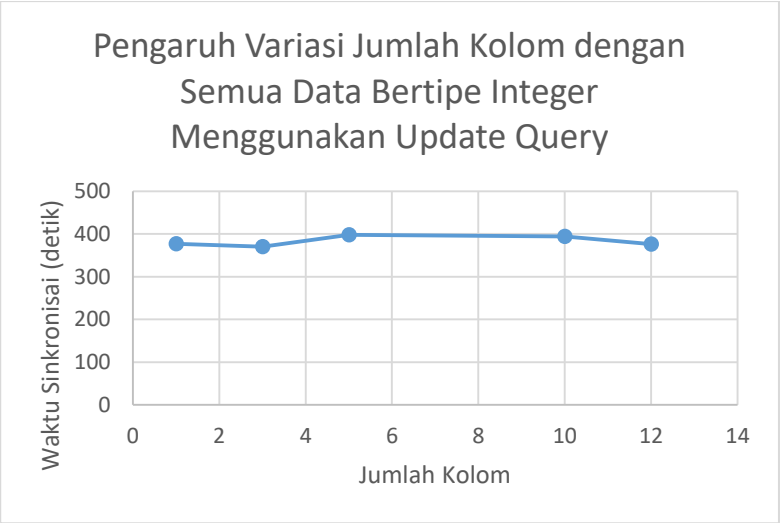
Skenario uji coba performa 4a adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan

waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.9.2 Evaluasi Uji Coba Performa 4a

Setelah dilakukan uji coba seperti pada skenario 3c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.12 dan pada Tabel 5.19. Dapat dilihat pada grafik Gambar 5.12 terjadi peningkatan dan penurunan waktu sinkronisasi di beberapa titik. Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.19 tidak terlalu signifikan, selain itu rata-rata dari rata-rata selisih hanya bernilai -0.081 detik yang menunjukkan uji coba skenario 4a ini tidak memiliki pengaruh yang signifikan pada performa aplikasi. Nilai minus pada -0.507 dimaksudkan aplikasi mengalami peningkatan performa dan penurunan pada waktu sinkronisasi.

Pada Tabel 5.19 dapat diperhatikan pada kolom rata-rata selisih, rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai -3.442 detik, nilai minus ini berarti terjadi sedikit peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 3 kolom. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai 14.067 detik, terjadi peningkatan pada waktu sinkronisasi pada 5 kolom meskipun nilainya tergolong rendah. Rata-rata selisih uji coba 5 kolom dengan 10 kolom bernilai -2.032 detik, terjadi sedikit peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 3 kolom. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai -8.917 detik, terjadi sedikit peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 12 kolom.



Gambar 5.12 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 4a

Tabel 5.19 Hasil Uji Coba Performa Skenario 4a

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	376.912	-
2	3	370.029	-3.442
3	5	398.162	14.067
4	10	394.098	-2.032
5	12	376.264	-8.917
Rata-rata			-0.081

5.3.9.3 Skenario Uji Coba Performa 4b

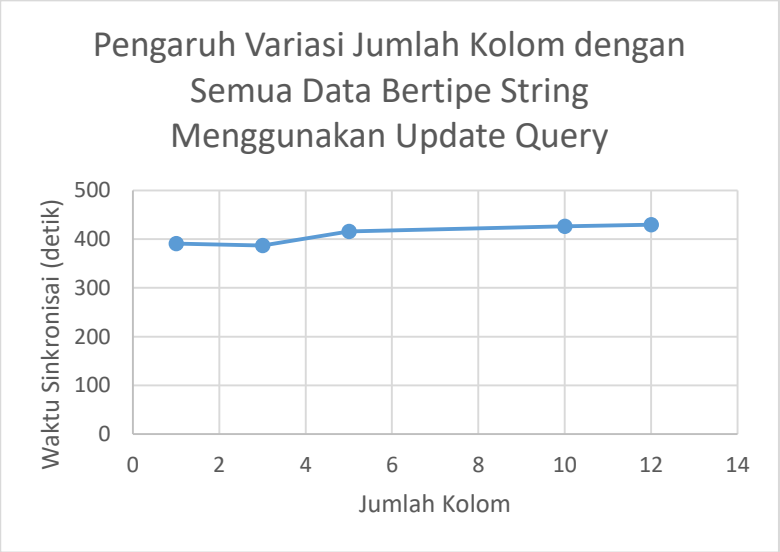
Skenario uji coba performa 4b adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.9.4 Evaluasi Uji Coba Performa 4b

Setelah dilakukan uji coba seperti pada skenario 3c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari tiga uji coba yang ditunjukkan oleh grafik Gambar 5.13 dan pada Tabel 5.20. Dapat dilihat pada grafik Gambar 5.13 terjadi peningkatan dan penurunan waktu sinkronisasi di beberapa titik. Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.20 tidak terlalu signifikan, selain itu rata-rata dari rata-rata selisih hanya bernilai 4.766 detik yang menunjukkan uji coba skenario 4b ini tidak memiliki pengaruh yang signifikan pada performa aplikasi.

Pada Tabel 5.20 dapat diperhatikan pada kolom rata-rata selisih, rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai -2.291 detik, nilai minus ini berarti terjadi sedikit peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 3 kolom. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai 14.645 detik, terjadi peningkatan pada waktu sinkronisasi pada 5 kolom meskipun nilainya tergolong rendah. Rata-rata selisih uji coba 5 kolom dengan 10 kolom bernilai 5.233

detik. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai 1.477 detik.



Gambar 5.13 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 4b

Tabel 5.20 Hasil Uji Coba Performa Skenario 4b

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	391.237	-
2	3	386.655	-2.291
3	5	415.946	14.645
4	10	426.412	5.233
5	12	429.367	1.477
Rata-rata			4.766

5.3.9.5 Skenario Uji Coba Performa 4c

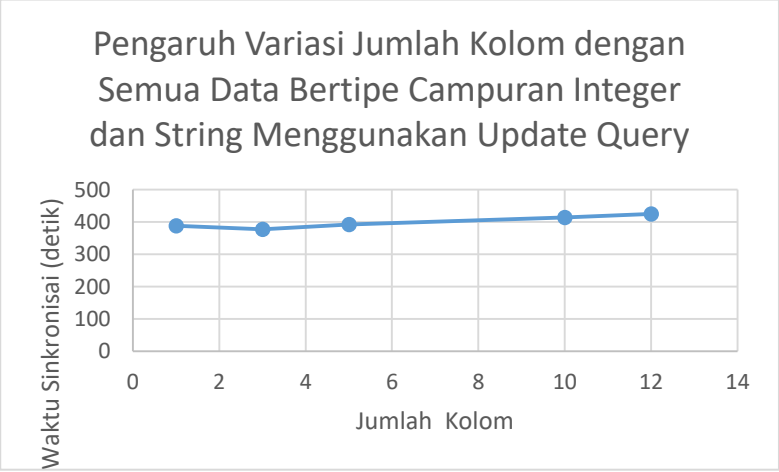
Skenario uji coba performa 4c adalah pengujian aplikasi dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah kolom. Variasi kolom yang digunakan adalah 1 kolom, 3 kolom, 5 kolom, 10 kolom, 12 kolom. Dengan dua tabel yang masing-masing berisi 500 *update query* dan semua data dalam tabel adalah campuran *string* dan *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.9.6 Evaluasi Uji Coba Performa 4c

Setelah dilakukan uji coba seperti pada skenario 4c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.14 dan pada Tabel 5.21. Dapat dilihat pada grafik Gambar 5.14 terjadi peningkatan dan penurunan waktu sinkronisasi di beberapa titik. Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.21 tidak terlalu signifikan, selain itu rata-rata dari rata-rata selisih hanya bernilai 4.594 detik yang menunjukkan uji coba skenario 4c ini tidak memiliki pengaruh yang signifikan pada performa aplikasi.

Pada Tabel 5.21 dapat diperhatikan pada kolom rata-rata selisih, rata-rata selisih uji coba 1 kolom dengan 3 kolom bernilai -5.655 detik, nilai minus ini berarti terjadi peningkatan perfoma pada aplikasi dan waktu sinkronisasi sedikit menurun pada uji coba 3 kolom. Rata-rata selisih uji coba 3 kolom dengan 5 kolom bernilai 7.219 detik, terjadi peningkatan pada waktu sinkronisasi pada 5 kolom meskipun nilainya tergolong rendah. Rata-rata

selisih uji coba 5 kolom dengan 10 kolom bernilai 11.135 detik, terjadi peningkatan pada waktu sinkronisasi pada 10 kolom meskipun nilainya tergolong rendah. Rata-rata selisih uji coba 10 kolom dengan 12 kolom bernilai 5.675 detik, terjadi peningkatan pada waktu sinkronisasi pada 12 kolom meskipun nilainya tergolong rendah.



Gambar 5.14 Grafik Pengaruh Jumlah Kolom terhadap Waktu Sinkronisasi pada Skenario 4c

Tabel 5.21 Hasil Uji Coba Skenario Performa 4c

No.	Jumlah Kolom	Waktu Sinkronisasi (detik)	Rata-rata Selish (detik)
1	1	388.643	-
2	3	377.333	-5.655
3	5	391.772	7.219
4	10	414.043	11.135
5	12	425.394	5.675
Rata-rata			4.594

5.3.10 Skenario Uji Coba Performa 5

Skenario uji coba performa 5 adalah pengujian dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Waktu yang dihitung adalah waktu dimana semua *query* berhasil dikirimkan dari *master node* ke *queue node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *update*.

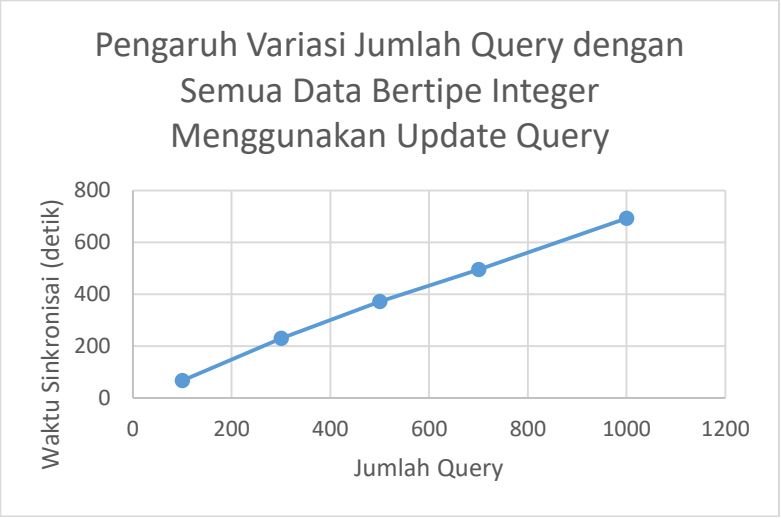
5.3.10.1 Skenario Uji Coba Performa 5a

Skenario uji coba 5a adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.10.2 Evaluasi Uji Coba Performa 5a

Setelah dilakukan uji coba performa seperti pada skenario 5a didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.15 dan pada Tabel 5.22. Pada grafik Gambar 5.15 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 67.002 detik. Untuk titik kedua dimana jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar

230.200 detik. Untuk titik ketiga dimana jumlah *query* adalah 500, waktu sinkronisasi diperlukan sekitar 371.212 detik. Untuk titik keempat dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 495.286 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 691.698 detik.



Gambar 5.15 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 5a

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap variasi uji coba pada Tabel 5.22 peningkatan rata-rata selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 78.087 detik yang menunjukkan uji coba skenario 5a ini memiliki pengaruh yang signifikan pada performa aplikasi. Meningkatnya jumlah *query* pada masing-masing tabel menyebabkan meningkat pula beban

kerja *queue* untuk pendistribusian *query*. Hasilnya distribusi *query* pun menjadi lebih lama sehingga waktu sinkronisasi terus terjadi peningkatan seperti pada grafik.

Tabel 5.22 Hasil Uji Coba Performa Skenario 5a

No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	100	67.002	-
2	300	230.200	81.599
3	500	371.212	70.506
4	700	495.286	62.037
5	1000	691.698	98.206
Rata-rata			78.087

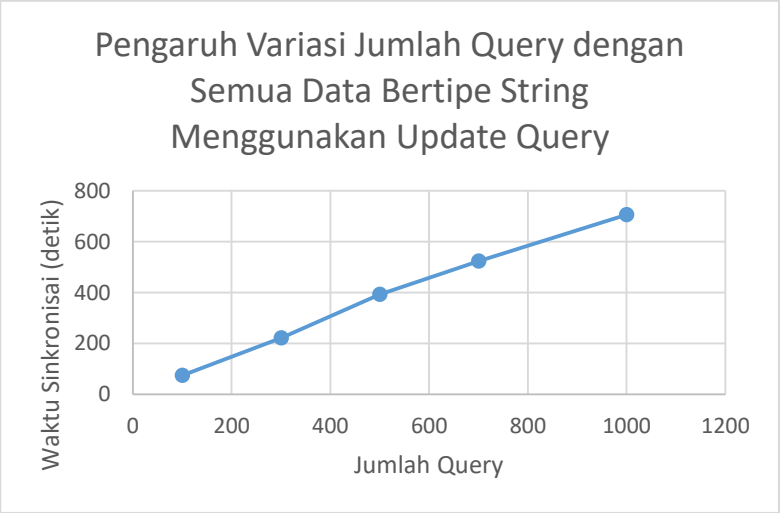
5.3.10.3 Skenario Uji Coba Performa 5b

Skenario uji coba performa 5b adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.10.4 Evaluasi Uji Coba Performa 5b

Setelah dilakukan uji coba seperti pada skenario 2b didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.16 dan pada Tabel 5.23. Pada grafik Gambar 5.16 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada

titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 74.440 detik. Untuk titik kedua dimana jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar 221.885 detik. Untuk titik ketiga dimana jumlah *query* adalah 500, waktu sinkronisasi diperlukan sekitar 392.786 detik. Untuk titik keempat dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 523.467 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 706.823 detik.



Gambar 5.16 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 5b

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.23 peningkatan rata-rata selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 79.048 detik yang menunjukkan uji

coba skenario 5b ini memiliki pengaruh yang signifikan pada performa aplikasi. Peningkatan yang signifikan pada setiap titik waktu sinkronisasi disebabkan oleh variasi jumlah *query*. Variasi jumlah *query* ini meningkatkan kerja *queue* seihingga distribusi menjadi bertambah lama setiap terjadi peningkatan *query*.

Tabel 5.23 Hasil Uji Coba Performa Skenario 5b

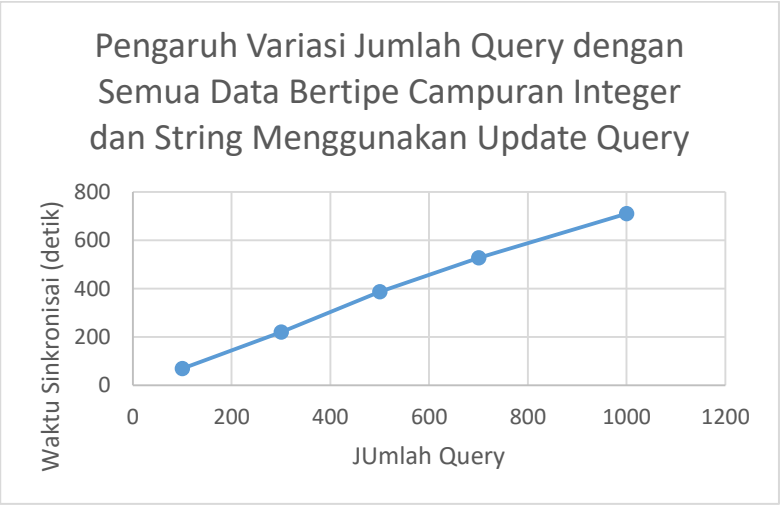
No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata - rata Selisih (detik)
1	100	74.440	-
2	300	221.885	73.722
3	500	392.786	85.450
4	700	523.467	65.341
5	1000	706.823	91.678
Rata-rata			79.048

5.3.10.5 Skenario Uji Coba Performa 5c

Skenario uji coba performa 5c adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah *query*. Variasi jumlah *query* yang digunakan adalah 100 *query*, 300 *query*, 500 *query*, 700 *query*, 1000 *query*. Dengan dua tabel yang masing-masing berisi 5 kolom dan semua data dalam tabel adalah campuran *integer* dan *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.10.6 Evaluasi Uji Coba Performa 5c

Setelah dilakukan uji coba seperti pada skenario 2c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.17 dan pada Tabel 5.24. Pada grafik Gambar 5.17 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah *query* adalah 100, waktu sinkronisasi diperlukan hanya sekitar 69.000 detik. Untuk titik kedua dimana jumlah *query* adalah 300, waktu sinkronisasi diperlukan sekitar 220.642 detik. Untuk titik ketiga dimana jumlah *query* adalah 500, waktu sinkronisasi diperlukan sekitar 386.856 detik. Untuk titik keempat dimana jumlah *query* adalah 700, waktu sinkronisasi diperlukan sekitar 526.742 detik. Untuk titik kelima dimana jumlah *query* adalah 1000, waktu sinkronisasi diperlukan sekitar 709.859 detik.



Gambar 5.17 Grafik Pengaruh Jumlah Query terhadap Waktu Sinkronisasi pada Skenario 5c

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.24 peningkatan rata-rata selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 80.107 detik yang menunjukkan uji coba skenario 5c ini memiliki pengaruh yang signifikan pada performa aplikasi.

Peningkatan pada waktu sinkronisasi yang membuat grafik terus meningkat ini disebabkan oleh variasi *query* yang mempengaruhi kerja *queue*. Hal ini karena setiap terjadi pengingkatan jumlah *query* maka *queue* pun bertambah banyak sehingga *query* yang didistribusikan pun meningkat sehingga waktu sinkronisasi pun juga ikut meningkat.

Tabel 5.24 Hasil Uji Coba Performa Skenario 5c

No.	Jumlah Query	Waktu Sinkronisasi (detik)	Rata-rata Selish (detik)
1	100	69.000	-
2	300	220.642	75.821
3	500	386.856	83.107
4	700	526.742	69.943
5	1000	709.859	91.558
Rata-rata			80.107

5.3.11 Skenario Uji Coba Performa 6

Skenario uji coba 6 adalah pengujian dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Waktu yang dihitung adalah

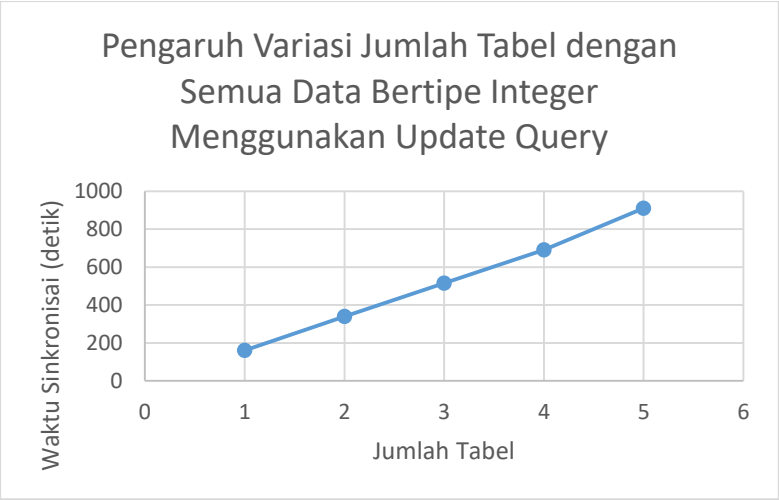
waktu dimana semua *query* berhasil dikirimkan dari *master node* ke queue *node*, kemudian didistribusikan ke *node* tujuan sampai *query* berhasil dieksekusi pada dua *node* tujuan. Untuk skenario ini *query* yang digunakan adalah *update*.

5.3.11.1 Skenario Uji Coba Performa 6a

Skenario uji coba performa 6a adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua tipe data dalam tabel adalah *integer* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.11.2 Evaluasi Uji Coba Performa 6a

Setelah dilakukan uji coba seperti pada skenario 3a didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.18 dan pada Tabel 5.25. Pada grafik Gambar 5.18 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 159.884 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 338.995 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi diperlukan sekitar 515.307 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 690.875 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 909.489 detik.



Gambar 5.18 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 6a

Tabel 5.25 Hasil Uji Coba Performa Skenario 6a

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	159.884	-
2	2	338.995	89.555
3	3	515.307	88.156
4	4	690.875	87.784
5	5	909.489	109.307
Rata-rata			93.701

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.25 peningkatan rata-rata

selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 93.701 detik yang menunjukkan uji coba skenario 6a ini memiliki pengaruh yang signifikan pada performa aplikasi.

Efek variasi tabel pada dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat. Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.

5.3.11.3 Skenario Uji Coba Performa 6b

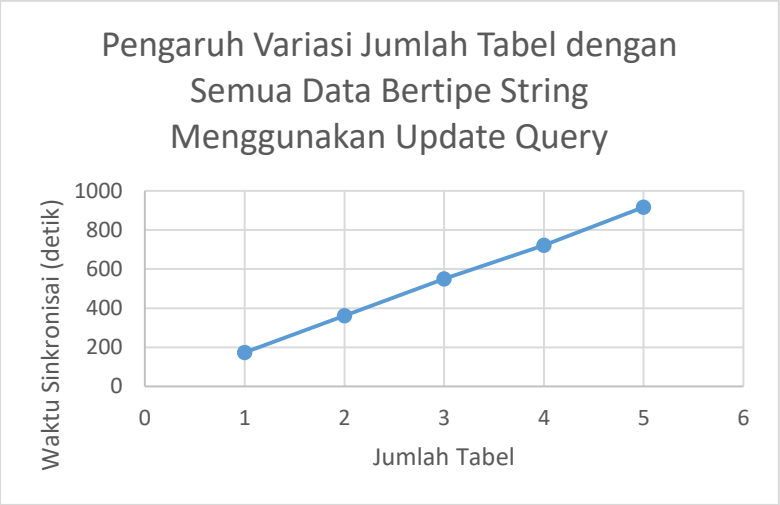
Skenario uji coba performa 6b adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua tipe data dalam tabel adalah *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.11.4 Evaluasi Uji Coba Performa 6b

Setelah dilakukan uji coba seperti pada skenario 3b didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.19 dan pada Tabel 5.26. Pada grafik Gambar 5.19 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 173.062 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 362.140 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi

diperlukan sekitar 550.507 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 722.308 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 916.154 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.26 peningkatan rata-rata selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 92.886 detik yang menunjukkan uji coba skenario 6b ini memiliki pengaruh yang signifikan pada performa aplikasi.



Gambar 5.19 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 6b

Tabel 5.26 Hasil Uji Coba Performa Skenario 6b

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	173.062	-
2	2	362.140	94.539
3	3	550.507	94.183
4	4	722.308	85.900
5	5	916.154	96.923
Rata-rata			92.886

Efek variasi tabel dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat. Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.

5.3.11.5 Skenario Uji Coba Performa 6c

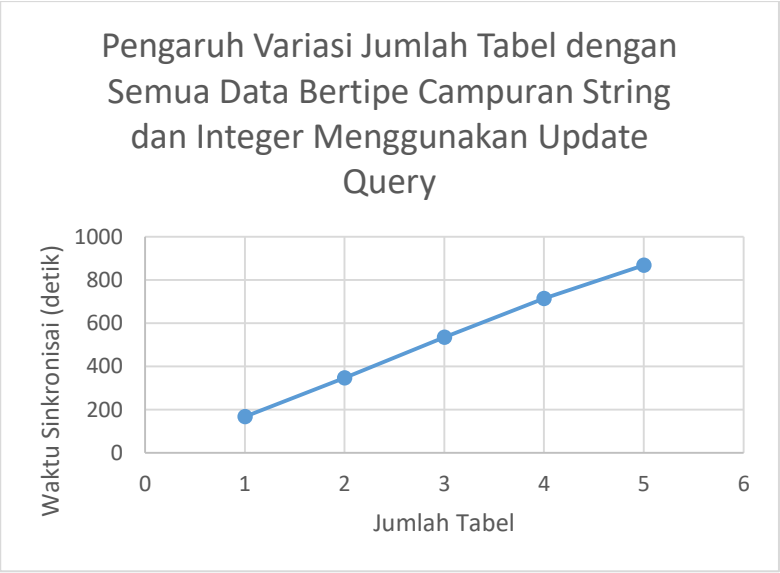
Skenario uji coba performa 6c adalah dengan melakukan perhitungan waktu performa pada aplikasi yang dipengaruhi oleh variasi jumlah tabel. Variasi jumlah tabel yang digunakan adalah 1 tabel, 2 tabel, 3 tabel, 4 tabel, 5 tabel. Dengan 5 kolom dan 500 *update query* pada setiap tabel serta semua data dalam tabel adalah campuran *integer* dan *string* serta dua *node* tujuan. Pencatatan yang dilakukan pada skenario ini adalah pencatatan waktu sampai sinkronisasi antara *master node* dan dua *node* tujuan atau *slave node* selesai dengan data yang sudah disebutkan.

5.3.11.6 Evaluasi Uji Coba Performa 6c

Setelah dilakukan uji coba seperti pada skenario 3c didapatkan hasil berupa waktu sinkronisasi yang setiap nilainya merupakan rata-rata dari lima uji coba yang ditunjukkan oleh grafik Gambar 5.20 dan pada Tabel 5.27. Pada grafik Gambar 5.20 dapat diamati perubahan waktu sinkronisasi terus meningkat. Pada titik pertama dimana jumlah tabel adalah 1, waktu sinkronisasi diperlukan sekitar 167.789 detik. Untuk titik kedua dimana jumlah tabel adalah 2, waktu sinkronisasi diperlukan sekitar 347.073 detik. Untuk titik ketiga dimana jumlah tabel adalah 3, waktu sinkronisasi diperlukan sekitar 534.790 detik. Untuk titik keempat dimana jumlah tabel adalah 4, waktu sinkronisasi diperlukan sekitar 713.822 detik. Untuk titik kelima dimana jumlah tabel adalah 5, waktu sinkronisasi diperlukan sekitar 868.830 detik.

Rata-rata selisih waktu sinkronisasi adalah rata-rata nilai hasil dari selisih antara dua waktu sinkronisasi yang dirata-rata untuk mengetahui seberapa besar pengaruh suatu variasi uji coba terhadap performa aplikasi. Dapat diperhatikan kolom rata-rata selisih pada setiap uji coba pada Tabel 5.27 peningkatan rata-rata selisih pada setiap uji coba sangat signifikan, selain itu rata-rata dari rata-rata selisih bernilai 87.630 detik yang menunjukkan uji coba skenario 6c ini memiliki pengaruh yang signifikan pada performa aplikasi.

Peningkatan waktu sinkronisasi pada grafik cukup signifikan, efek variasi tabel pada dapat mempengaruhi performa secara signifikan. Hal ini dikarenakan jika ada peningkatan tabel maka ada pula peningkatan *capture* tabel. Sehingga proses ekstraksi data pada *master node* juga meningkat. Demikian juga pada *queue node*, jumlah *queue* pun akan meningkat sehingga distribusi *query* akan bertambah lama.



Gambar 5.20 Grafik Pengaruh Jumlah Tabel terhadap Waktu Sinkronisasi pada Skenario 6c

Tabel 5.27 Hasil Uji Coba Performa Skenario 6c

No.	Jumlah Tabel	Waktu Sinkronisasi (detik)	Rata-rata Selisih (detik)
1	1	167.789	-
2	2	347.073	89.642
3	3	534.790	93.858
4	4	713.822	89.516
5	5	868.830	77.504
Rata-rata			87.630

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan pada bab sebelumnya, yaitu Bab Uji Coba dan Evaluasi. Bab ini juga digunakan sebagai jawaban dari rumusan masalah yang dikemukakan pada Bab Pendahuluan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan dapat diambil beberapa kesimpulan sebagai berikut:

1. Proses sinkronisasi dengan operasi *insert*, *update* dan *delete* pada tipe data *integer* dan *string* berhasil bekerja dengan 100% sukses. Namun, untuk operasi *insert*, *update* dan *delete* pada tipe data *datetime* tidak dapat berjalan dengan baik atau gagal.
2. Beban maksimal yang dapat ditangani oleh *web service* pada node pusat adalah 100 pengguna secara simultan, dimana rasio *request* diterima 100% dan *request drop* adalah 0%.
3. Variasi jumlah kolom memiliki pengaruh yang tidak signifikan terhadap performa aplikasi, dilihat dari nilai rata-rata dari rata-rata selisih waktu sinkronisasi yang rendah. Ditunjukkan pada skenario 1a, 1b, 1c, 4a, 4b, 4c yang memiliki nilai rata-rata dari rata-rata selisih waktu sinkronisasi secara berurutan adalah 3.106 detik, 10.801 detik, 4.574 detik, -0.081 detik, 4.766 detik, 4.594 detik.
4. Variasi jumlah *query* dapat mempengaruhi waktu sinkronisasi aplikasi, dengan melihat hasil evaluasi pada skenario 2a, 2b, 2c, 5a, 5b, 5c memiliki nilai rata-rata dari rata-rata selisih waktu sinkronisasi secara berurutan adalah 84.688 detik, 89.180 detik, 87.288 detik, 78.087

detik, 79.048 detik, 80.107 detik. Maka dari itu, dapat disimpulkan variasi jumlah *query* memiliki pengaruh yang signifikan terhadap performa aplikasi, dilihat dari nilai rata-rata dari rata-rata selisih waktu sinkronisasi yang tinggi.

5. Variasi jumlah tabel dapat mempengaruhi waktu sinkronisasi aplikasi dengan melihat hasil evaluasi pada skenario 3a, 3b, 3c, 6a, 6b, 6c memiliki nilai rata-rata dari rata-rata selisih waktu sinkronisasi secara berurutan adalah 86.576 detik, 87.183 detik, 87.849 detik, 93.701 detik, 92.886 detik, 87.630 detik. Maka dari itu variasi jumlah tabel memiliki pengaruh yang signifikan terhadap performa aplikasi, dilihat dari nilai rata-rata dari rata-rata selisih waktu sinkronisasi yang tinggi.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Untuk transportasi data disarankan dapat dioptimalisasi agar distribusi dari *master node* ke *queue* maupuan *queue* ke *node* tujuan menjadi lebih cepat.
2. Disarankan dilakukan optimalisasi pada *queue*.

DAFTAR PUSTAKA

[1] R. Gudakesa, I. M. Sukarsa dan I. G. M. A. Sasmita, "TWO-WAYS DATABASE SYNCHRONIZATION IN," *Journal of Theoretical and Applied Information Technology,* vol. 65, no. 3, pp. 854-859, 2014.

[2] J. W. F. R. Yubiao Wang, "Research on Incremental Heterogeneous database Synchronization," *International Conference on Computational Intelligence and Communication Networks,* pp. 1415-1419, 2015.

[3] Code Two, "One-way synchronization," Code Two, [Online]. Available: https://www.codetwo.com/userguide/exchange-folder-sync/one-way-sync.htm. [Diakses 15 Maret 2017].

[4] G. C. Everest, Database Management: Objectives, System Functions, and Administration, New York: McGraw-Hill Companies, 1986.

[5] Kusrini, Strategi Perancangan dan Pengelolaan Basis Data, Yogyakarta: Andi Offset, 2007.

[6] M. Rouse, "database management system (DBMS)," techtarget.com, Januari 2015. [Online]. Available: http://searchsqlserver.techtarget.com/definition/database-management-system. [Diakses 15 April 2017].

[7] Technopedia, "Oracle Database (Oracle DB)," Technopedia, 2017. [Online]. Available: https://www.techopedia.com/definition/8711/oracle-database. [Diakses 17 April 2017].

[8] Oracle, "Introduction to Oracle Database," Oracle, 2017. [Online]. Available: https://docs.oracle.com/database/122/CNCPT/introduction-to-oracle-database.htm#CNCPT001. [Diakses 17 April 2017].

[9] MySQL, “What is MySQL?,” Oracle Corporation, 2017. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html. [Diakses 17 April 2017].

[10] M. I. Hossain dan M. M. Ali, “SQL query based data synchronization in heterogeneous database environment,” *Computer Communication and Informatics (ICCCI),* no. 10, pp. 1-5, 2012.

[11] D.-s. Z. Jue Wang, “Research and Design of Distributed Database Synchronization System Based on,” *International Conference on Intelligent Computation Technology and Automation,* no. 8, pp. 685-688, 2015.

[12] Oracle, “What Are Web Services?,” Oracle, Januari 2013. [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html. [Diakses 3 November 2016].

[13] guru99.com, “Web service architecture & Introduction,” guru99.com, 2017. [Online]. Available: http://www.guru99.com/web-service-architecture.html. [Diakses 17 April 2017].

[14] M. Rouse, “REST (representational state transfer),” TechTarget, Desember 2014. [Online]. Available: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer. [Diakses 17 April 2017].

[15] MySql, “Using Triggers,” MySql, [Online]. Available: https://dev.mysql.com/doc/refman/5.6/en/triggers.html. [Diakses 15 Maret 2017].

[16] techterms, “Python,” techterms, 15 Juni 2010. [Online]. Available: https://techterms.com/definition/python. [Diakses 12 Mei 2017].

[17] A. Dustman, “MySQLdb User's Guide,” sourceforge, 18 December 2005. [Online]. Available: http://mysql-

python.sourceforge.net/MySQLdb.html. [Diakses 10 Mei 2017].

[18] A. Tuininga, “Welcome to cx_Oracle’s documentation!,” oracle, 2015. [Online]. Available: https://cx-oracle.readthedocs.io/en/latest/. [Diakses 15 April 2017].

[19] K. Das, “Introduction to Flask,” pymbook, 2015. [Online]. Available: http://pymbook.readthedocs.io/en/latest/flask.html. [Diakses 17 April 2017].

[20] K. Reitz, “Requests: HTTP for Humans,” python-requests.org, 2017. [Online]. Available: http://docs.python-requests.org/en/master/. [Diakses 8 April 2017].

[21] J. Heyman, C. Byström, J. Hamrén dan H. Heyman, “What is Locust?,” locust.io, 2017. [Online]. Available: http://docs.locust.io/en/latest/what-is-locust.html. [Diakses 21 Juni 2017].

[Halaman ini sengaja dikosongkan]

LAMPIRAN A
KODE SUMBER

Pada lampiran berikut dijelaskan mengenai kode sumber yang digunakan. Kode sumber menggunakan Bahasa Pemrograman Python. Berikut kode sumber yang dicantumkan:

1. Kode Sumber 1 Fungsi Pembacaan File Konfigurasi merupakan bagian dari kode sumber lain yang memiliki fungsi pembacaan file konfigurasi.
2. Kode Sumber 2 Fungsi Membuat Capture Komponen merupakan kode sumber untuk membuat *capture* komponen.

3. Kode Sumber 3 Sinkronisasi Post Request merupakan kode sumber untuk melakukan ekstraksi data, pengolahan data menjadi JSON dikirimkan ke queue.

4. Kode Sumber 4 Fungsi Test Post Request merupakan kode untuk melakukan test request ke *node* tujuan atau slave untuk mengetahui *node* itu hidup tatau mati. Kode sumber ini merupakan bagian dari Kode Sumber 7.

5. Kode Sumber 5 Web Service pada Slave Node merupakan kode sumber untuk membuat komponen receiver pada *node* tujuan atau slave *node*.

6. Kode Sumber 6 Pembuatan Queue merupakan kode sumber untuk queue pada *node* pusat.

7. Kode Sumber 7 Distribusi dari Queue ke Slave Node merupakan kode sumber untuk mendistribusikan data dari queue ke *node* tujuan atau slave *node*.

8. Kode Sumber 8 Web Service pada Queue merupakan kode sumber untuk membuat komponen receiver pada *node* pusat untuk menerima data dari *master node*.

Kode Sumber 1 Fungsi Pembacaan File Konfigurasi

```
1.   Config.read("config.ini")
2.   def ConfigSectionMap(section):
3.       dict1 = {}
4.       options = Config.options(section)
5.       for option in options:
6.           try:
7.               dict1[option] = Config.get(section, option)
8.               if dict1[option] == -1:
9.                   print ("skip: %s" % option)
10.          except:
11.              print("exception on %s!" % option)
12.              dict1[option] = None
13.      return dict1
```

Kode Sumber 2 Fungsi Membuat Capture Komponen

```
1.   __author__ = 'Indra Gunawan'
2.   import ConfigParser
```

```
3.  import MySQLdb
4.  import cx_Oracle
5.
6.  Config = ConfigParser.ConfigParser()
7.
8.  Config.read("config.ini")
9.  def ConfigSectionMap(section):
10.     dict1 = {}
11.     options = Config.options(section)
12.     for option in options:
13.         try:
14.             dict1[option] = Config.get(section, option)
15.             if dict1[option] == -1:
16.                 print ("skip: %s" % option)
17.         except:
18.             print("exception on %s!" % option)
19.             dict1[option] = None
20.     return dict1
21.
22. ip = ConfigSectionMap("database")['ip']
23. db = ConfigSectionMap("database")['db']
24. username = ConfigSectionMap("database")['username']
25. password = ConfigSectionMap("database")['password']
26.
27. #mendapatkan list tabel dari config
28. list_nama_tabel_old = [e.strip() for e in ConfigSectionMap("tabel")['nama'].split(',')]
29. # print type(list_nama_tabel_old)
30. # print list_nama_tabel_old
```

```
31.
32. #membuat list nama komponen yg dibutuhkan dalam list
33. list_nama_tabel_capture = []
34. list_trigger_name_af_ins=[]
35. list_trigger_name_af_upd = []
36. list_trigger_name_af_del = []
37. for index_nama in range(len(list_nama_tabel_old)):
38.     #print "do something"
39.     list_nama_tabel_capture.append(list_nama_tabel_old[index_nama]+"_capture")
40.     list_trigger_name_af_ins.append(list_nama_tabel_old[index_nama] + "_trigger_af_ins")
41.     list_trigger_name_af_upd.append(list_nama_tabel_old[index_nama] + "_trigger_af_upd")
42.     list_trigger_name_af_del.append(list_nama_tabel_old[index_nama] + "_trigger_af_del")
43.
44. # print list_nama_tabel_capture
45. # print list_trigger_name_af_ins
46. # print list_trigger_name_af_upd
47. # print list_trigger_name_af_del
48.
49. jenis = ConfigSectionMap("database")['jenis']
50. if jenis == "oracle":
51.     con = cx_Oracle.connect(username+'/'+password+'@'+ip+'/'+db+'')
52.     # print con.version
53.
54.     cursor_ora = con.cursor()
55.     for index_ora in range(len(list_nama_tabel_old)):
56.         print list_nama_tabel_old[index_ora]
57.         #Cara Describe tabel
```

```
58.         describe_ora = "select column_name, data_type, data_length, nullable from all_tab_columns where
    table_name ='%s'" %list_nama_tabel_old[index_ora].upper()
59.         # print sql3
60.
61.         try:
62.             cursor_ora.execute(describe_ora)
63.             print "describe table berhasil"
64.
65.
66.         except:
67.             print "Error: unable to describe"
68.
69.         result = cursor_ora.fetchall()
70.         print result
71.         simpan_kolom = []
72.         simpan_value_new = []
73.         simpan_value_old = []
74.
75.         for index2_ora in range(len(result)):
76.             simpan_kolom.append(result[index2_ora][0])
77.             simpan_value_new.append(":new." + result[index2_ora][0])
78.             simpan_value_old.append(":old." + result[index2_ora][0])
79.         # kolom dan value untuk trigger
80.         kolomToString = ', '.join(simpan_kolom)
81.         valueToStringNew = ', '.join(simpan_value_new)
82.         valueToStringOld = ', '.join(simpan_value_old)
83.         print kolomToString
84.         print valueToStringNew
```

```
85.         print valueToStringOld
86.
87.         # create table capture
88.         create_capture = "CREATE TABLE "+list_nama_tabel_capture[index_ora].upper()+" AS (SELECT * FROM
    "+list_nama_tabel_old[index_ora].upper()+" )"
89.         try:
90.             cursor_ora.execute(create_capture)
91.             print "capture table berhasil dibuat"
92.
93.         except:
94.             print "Error: unable to create capture"
95.
96.         # alter table capture
97.         tambah_kolom_action = "ALTER TABLE " + list_nama_tabel_capture[index_ora] + " ADD ACTION VARCHAR2(20)"
98.         try:
99.             cursor_ora.execute(tambah_kolom_action)
100.            print "alter table berhasil dibuat"
101.
102.        except:
103.            print "Error: unable to alter"
104.
105.        tambah_kolom_id = "ALTER TABLE " + list_nama_tabel_capture[index_ora] + " ADD ID_CAPTURE NUMBER(10)
    GENERATED ALWAYS AS IDENTITY"
106.        print tambah_kolom_id
107.        try:
108.            cursor_ora.execute(tambah_kolom_id)
109.            print "alter table berhasil dibuat"
110.
```

```
111.         except:
112.             print "Error: unable to alter"
113.
114.
115.         update_insert_data_exist = "UPDATE " + list_nama_tabel_capture[index_ora] + " SET action='insert'"
116.
117.         try:
118.             cursor_ora.execute(update_insert_data_exist)
119.             con.commit()
120.             print "update kolom action dari insert data yg sudah ada"
121.
122.         except:
123.             print "Error: unable to update or no data to update"
124.
125.         # trigger capture waktu insert
126.         trigger_insert = "CREATE TRIGGER " + list_trigger_name_af_ins[index_ora] + " AFTER INSERT ON " +
     list_nama_tabel_old[index_ora].upper() + " FOR EACH ROW BEGIN INSERT INTO " +
     list_nama_tabel_capture[index_ora].upper() + " (" + kolomToString + ",ACTION) VALUES (" + valueToStringNew +
     ",'insert');END;"
127.
128.         # trigger capture waktu update
129.         trigger_update = "CREATE TRIGGER " + list_trigger_name_af_upd[index_ora] + " AFTER UPDATE ON " +
     list_nama_tabel_old[index_ora].upper() + " FOR EACH ROW BEGIN INSERT INTO " +
     list_nama_tabel_capture[index_ora].upper() + " (" + kolomToString + ",ACTION) VALUES (" + valueToStringNew +
     ",'update');END;"
130.
131.         # trigger capture waktu delete
```

```
132.        trigger_delete = "CREATE TRIGGER " + list_trigger_name_af_del[index_ora] + " AFTER DELETE ON " +
    list_nama_tabel_old[index_ora].upper() + " FOR EACH ROW BEGIN INSERT INTO " +
    list_nama_tabel_capture[index_ora].upper() + " (" + kolomToString + ",ACTION) VALUES (" + valueToStringOld +
    ",'delete');END;"
133.
134.        try:
135.
136.            cursor_ora.execute(trigger_insert)
137.            print "trigger insert berhasil dibuat"
138.
139.
140.        except:
141.            print "Error: unable to create trigger or trigger already exist"
142.
143.        try:
144.
145.            cursor_ora.execute(trigger_update)
146.            print "trigger update berhasil dibuat"
147.
148.        except:
149.            print "Error: unable to create trigger or trigger already exist"
150.
151.        try:
152.
153.            cursor_ora.execute(trigger_delete)
154.            print "trigger delete berhasil dibuat"
155.
156.        except:
```

```
157.             print "Error: unable to create trigger or trigger already exist"
158.
159.     cursor_ora.close()
160.     con.close()
161.
162. elif jenis == "mysql":
163.     db = MySQLdb.connect(ip, username, password, db)
164.     cursor = db.cursor()
165.
166.     for index in range(len(list_nama_tabel_old)):
167.         print list_nama_tabel_old[index]
168.         sql2 = "DESCRIBE " + list_nama_tabel_old[index]
169.         try:
170.             cursor.execute(sql2)
171.             print "describe table berhasil"
172.
173.         except:
174.             print "Error: unable to describe"
175.
176.         result = cursor.fetchall()
177.         simpan_kolom = []
178.         simpan_value_new = []
179.         simpan_value_old = []
180.
181.         for index2 in range(len(result)):
182.             simpan_kolom.append(result[index2][0])
183.             simpan_value_new.append("new." + result[index2][0])
184.             simpan_value_old.append("old." + result[index2][0])
```

```
185.         # kolom dan value untuk trigger
186.         kolomToString = ', '.join(simpan_kolom)
187.         valueToStringNew = ', '.join(simpan_value_new)
188.         valueToStringOld = ', '.join(simpan_value_old)
189.         print kolomToString
190.         print valueToStringNew
191.         print valueToStringOld
192.
193.         # create table capture
194.         create_capture = "CREATE TABLE " + list_nama_tabel_capture[index] + " SELECT * FROM " +
    list_nama_tabel_old[
195.             index] + " "
196.         try:
197.             cursor.execute(create_capture)
198.             print "capture table berhasil dibuat"
199.
200.         except:
201.             print "Error: unable to create capture"
202.
203.         # create table capture
204.         tambah_kolom_action = "ALTER TABLE " + list_nama_tabel_capture[index] + " ADD action TEXT"
205.         try:
206.             cursor.execute(tambah_kolom_action)
207.             print "alter table berhasil dibuat"
208.
209.         except:
210.             print "Error: unable to alter"
211.
```

```
212.        tambah_kolom_id = "ALTER TABLE " + list_nama_tabel_capture[index] + " ADD id_capture INT PRIMARY KEY
    AUTO_INCREMENT;"
213.        try:
214.            cursor.execute(tambah_kolom_id)
215.            print "alter table berhasil dibuat"
216.
217.        except:
218.            print "Error: unable to alter"
219.
220.        update_insert_data_exist = "UPDATE " + list_nama_tabel_capture[index] + " SET action='insert' WHERE 1 "
221.
222.        try:
223.            cursor.execute(update_insert_data_exist)
224.            db.commit()
225.            print "update kolom action dari insert data yg sudah ada"
226.
227.        except:
228.            print "Error: unable to update or no data to update"
229.
230.        # trigger capture waktu insert
231.        trigger_insert = "CREATE TRIGGER " + list_trigger_name_af_ins[index] + " AFTER INSERT ON " + \
232.                         list_nama_tabel_old[index] + " FOR EACH ROW BEGIN INSERT INTO " +
    list_nama_tabel_capture[
233.                             index] + " (" + kolomToString + ",action) VALUES (" + valueToStringNew +
    ",'insert');END;"
234.
235.        # trigger capture waktu update
236.        trigger_update = "CREATE TRIGGER " + list_trigger_name_af_upd[index] + " AFTER UPDATE ON " + \
```

```
237.                         list_nama_tabel_old[index] + " FOR EACH ROW BEGIN INSERT INTO " +
    list_nama_tabel_capture[
238.                             index] + " (" + kolomToString + ",action) VALUES (" + valueToStringNew +
    ",'update');END;"
239.
240.         # trigger capture waktu delete
241.         trigger_delete = "CREATE TRIGGER " + list_trigger_name_af_del[index] + " AFTER DELETE ON " + \
242.                         list_nama_tabel_old[index] + " FOR EACH ROW BEGIN INSERT INTO " +
    list_nama_tabel_capture[
243.                             index] + " (" + kolomToString + ",action) VALUES (" + valueToStringOld +
    ",'delete');END;"
244.
245.         try:
246.
247.             cursor.execute(trigger_insert)
248.             print "trigger insert berhasil dibuat"
249.
250.
251.         except:
252.             print "Error: unable to create trigger or trigger already exist"
253.
254.         try:
255.
256.             cursor.execute(trigger_update)
257.             print "trigger update berhasil dibuat"
258.
259.         except:
260.             print "Error: unable to create trigger or trigger already exist"
```

```
261.
262.        try:
263.
264.            cursor.execute(trigger_delete)
265.            print "trigger delete berhasil dibuat"
266.
267.        except:
268.            print "Error: unable to create trigger or trigger already exist"
269.
270.    db.close()
```

Kode Sumber 3 Sinkronisasi Post Request

```
1.  __author__ = 'Indra Gunawan'
2.  import requests
3.  import ConfigParser
4.  import MySQLdb
5.  import cx_Oracle
6.  import threading
7.
8.  Config = ConfigParser.ConfigParser()
9.
10. Config.read("config.ini")
11. def ConfigSectionMap(section):
12.     dict1 = {}
13.     options = Config.options(section)
14.     for option in options:
```

```
        try:
            dict1[option] = Config.get(section, option)
            if dict1[option] == -1:
                print ("skip: %s" % option)
        except:
            print("exception on %s!" % option)
            dict1[option] = None
    return dict1

def tuple_without(original_tuple, element_to_remove):
    lst = list(original_tuple)
    # lst.remove(element_to_remove)
    del lst[element_to_remove]
    return tuple(lst)

# ip = ConfigSectionMap("database")['ip']
# db = ConfigSectionMap("database")['db']
# username = ConfigSectionMap("database")['username']
# password = ConfigSectionMap("database")['password']

# list_other_IP_nodes = [e.strip() for e in ConfigSectionMap("other-nodes")['ip'].split(',')]
# print type(list_other_IP_nodes)
# print list_other_IP_nodes
#
# list_nama_tabel_old = [e.strip() for e in ConfigSectionMap("tabel")['nama'].split(',')]
# list_capture = [e.strip() for e in ConfigSectionMap("tabel_capture")['nama'].split(',')]
# list_primary = [e.strip() for e in ConfigSectionMap("tabel")['kolom_primary'].split(',')]
# print type(list_capture)
```

```
# print list_capture
def sinkronisasi():
    ip = ConfigSectionMap("database")['ip']
    db = ConfigSectionMap("database")['db']
    username = ConfigSectionMap("database")['username']
    password = ConfigSectionMap("database")['password']
    list_other_IP_nodes = [e.strip() for e in ConfigSectionMap("other-nodes")['ip'].split(',')]
    center_node = ConfigSectionMap("center-node")['ip']
    own_ip = ConfigSectionMap("host")['ip']
    print type(list_other_IP_nodes)
    print list_other_IP_nodes

    list_nama_tabel_old = [e.strip() for e in ConfigSectionMap("tabel")['nama'].split(',')]
    list_capture = [e.strip() for e in ConfigSectionMap("tabel_capture")['nama'].split(',')]
    list_primary = [e.strip() for e in ConfigSectionMap("tabel")['kolom_primary'].split(',')]
    jenis = ConfigSectionMap("database")['jenis']
    if jenis == "mysql":
        db = MySQLdb.connect(ip, username, password, db)
        cursor = db.cursor()
        for index_tabel_capture in range(len(list_capture)):
            print "Tabel " + list_capture[index_tabel_capture]
            sql = "SELECT * FROM "+list_capture[index_tabel_capture]+" LIMIT 20"

            try:
                cursor.execute(sql)
            except:
                print "unable to syncronize karena tabel capture tidak ditemukan"
                break
```

```
71.
72.             data_capture = cursor.fetchall()
73.
74.             # print data_capture
75.
76.             nama_kolom = [i[0] for i in cursor.description]
77.             index_action = nama_kolom.index('action')
78.             index_id = nama_kolom.index(list_primary[index_tabel_capture])
79.             index_id_capture = nama_kolom.index('id_capture')
80.             # print index_id_capture
81.
82.
83.             kolom_insert = list(nama_kolom)
84.             del kolom_insert[index_id_capture]
85.             del kolom_insert[index_action]
86.             del kolom_insert[index_id]
87.
88.             kolomToString = ', '.join(kolom_insert)
89.
90.             kolom_update = list(nama_kolom)
91.             del kolom_update[index_id_capture]
92.             del kolom_update[index_action]
93.             del kolom_update[index_id]
94.
95.
96.             print kolom_update
97.
98.             print kolom_insert
```

```
99.             # print kolomToString
100.
101.
102.            list_data_insert = []
103.
104.            for index in range(len(data_capture)):
105.
106.                if data_capture[index][index_action]=='insert':
107.                    # print 'insert captured'
108.                    # print data_capture[index]
109.                    data = tuple_without(data_capture[index], index_id_capture)
110.                    data1 = tuple_without(data,index_action)
111.                    data2 = tuple_without(data1, index_id)
112.                    # print data
113.                    copy_data2 = list(data2)
114.                    list_insert_value = []
115.
116.                    for index3 in range(len(copy_data2)):
117.                        if str(type(copy_data2[index3])) == "<type 'long'>":
118.                            list_insert_value.append(int(copy_data2[index3]))
119.                        else:
120.                            list_insert_value.append(copy_data2[index3])
121.
122.
123.                    insert_sql_to_send = "INSERT INTO "+list_nama_tabel_old[index_tabel_capture]+"
    ("+kolomToString+") VALUES %r" % (tuple(list_insert_value),)
124.                    insert_sql_to_send.replace("None", "NULL")
125.                    print insert_sql_to_send
```

```
126.
127.                     delete_data_capture = "DELETE FROM " + list_capture[index_tabel_capture] + " WHERE id_capture
     = " + repr(int(data_capture[index][index_id_capture])) + ""
128.                     print delete_data_capture
129.
130.                     count=0
131.                     for index_tujuan1 in range(len(list_other_IP_nodes)):
132.                         payload = {'sumber':own_ip,'tujuan': list_other_IP_nodes[index_tujuan1], 'data':
     insert_sql_to_send, 'nama_tabel':list_nama_tabel_old[index_tabel_capture]}
133.                         try:
134.                             r = requests.post("http://"+center_node+"/queue", json=payload)
135.                             print r.json()['pesan']
136.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
137.                                 count += 1
138.                         except:
139.                             print "center node "+center_node+" tidak ditemukan"
140.                         # print(r.text)
141.
142.                     print count
143.                     if len(list_other_IP_nodes)==count:
144.                         try:
145.                             cursor.execute(delete_data_capture)
146.                             db.commit()
147.
148.                         except:
149.                             db.rollback()
150.
151.                 elif data_capture[index][index_action] == 'update':
```

```
152.                     data = tuple_without(data_capture[index], index_id_capture)
153.                     data1 = tuple_without(data, index_action)
154.                     data2 = tuple_without(data1, index_id)
155.                     to_send = []
156.
157.                     for index2 in range(len(data2)):
158.                         if str(type(data2[index2])) == "<type 'long'>":
159.                             # print "flag"
160.                             # test.append(data2[index2])
161.                             to_send.append(kolom_update[index2] + "=" + repr(int(data2[index2])))
162.                         else:
163.                             to_send.append(kolom_update[index2]+"="+repr(data2[index2]))
164.
165.
166.                     value = ",".join(to_send)
167.
168.                     update_to_send = "UPDATE "+list_nama_tabel_old[index_tabel_capture]+" SET "+value+" WHERE
    "+list_primary[index_tabel_capture]+" = "+repr(int(data_capture[index][index_id]))+""
169.                     # update_to_send.replace("None", "NULL")
170.                     print update_to_send
171.
172.                     delete_data_capture = "DELETE FROM " + list_capture[index_tabel_capture] + " WHERE id_capture
    = " + repr(int(data_capture[index][index_id_capture])) + ""
173.                     print delete_data_capture
174.
175.                     count = 0
176.                     for index_tujuan2 in range(len(list_other_IP_nodes)):
```

```
177.                         payload = {'sumber': own_ip, 'tujuan': list_other_IP_nodes[index_tujuan2],'data':
    update_to_send, 'nama_tabel': list_nama_tabel_old[index_tabel_capture]}
178.                         try:
179.                             r = requests.post("http://" + center_node + "/queue", json=payload)
180.                             print r.json()['pesan']
181.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
182.                                 count += 1
183.                         except:
184.                             print "node "+list_other_IP_nodes[index_tujuan2]+" tidak ditemukan"
185.                         # print(r.text)
186.
187.                     print count
188.                     if len(list_other_IP_nodes) == count:
189.                         try:
190.                             cursor.execute(delete_data_capture)
191.                             db.commit()
192.
193.                         except:
194.                             db.rollback()
195.
196.                 elif data_capture[index][index_action] == 'delete':
197.
198.
199.                     delete_to_send = "DELETE FROM "+list_nama_tabel_old[index_tabel_capture]+" WHERE
    "+list_primary[index_tabel_capture]+" = "+repr(int(data_capture[index][index_id]))+""
200.                     print delete_to_send
201.
```

```
202.                     delete_data_capture = "DELETE FROM " + list_capture[index_tabel_capture] + " WHERE id_capture
    = " + repr(int(data_capture[index][index_id_capture])) + ""
203.                     print delete_data_capture
204.
205.                     count = 0
206.                     for index_tujuan3 in range(len(list_other_IP_nodes)):
207.                         payload = {'sumber': own_ip, 'tujuan': list_other_IP_nodes[index_tujuan3],'data':
    delete_to_send, 'nama_tabel': list_nama_tabel_old[index_tabel_capture]}
208.                         try:
209.                             r = requests.post("http://" + center_node + "/queue", json=payload)
210.                             print r.json()['pesan']
211.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
212.                                 count += 1
213.                         except:
214.                             print "node "+list_other_IP_nodes[index_tujuan3]+" tidak ditemukan"
215.                         # print(r.text)
216.
217.                     print count
218.                     if len(list_other_IP_nodes) == count:
219.                         try:
220.                             cursor.execute(delete_data_capture)
221.                             db.commit()
222.
223.                         except:
224.                             db.rollback()
225.
226.         db.close()
227.     elif jenis == "oracle":
```

```
228.         db = cx_Oracle.connect(username+'/'+password+'@'+ip+'/'+db+'')
229.         cursor = db.cursor()
230.
231.         for index_tabel_capture in range(len(list_capture)):
232.             print "Tabel "+list_capture[index_tabel_capture]
233.             sql = "SELECT * FROM "+list_capture[index_tabel_capture]+""
234.
235.             try:
236.                 cursor.execute(sql)
237.             except:
238.                 print "unable to syncronize karena tabel capture tidak ditemukan"
239.                 break
240.
241.             data_capture = cursor.fetchall()
242.             print data_capture
243.             nama_kolom = [i[0] for i in cursor.description]
244.             print nama_kolom
245.
246.             index_action = nama_kolom.index('action'.upper())
247.             index_id = nama_kolom.index(list_primary[index_tabel_capture].upper())
248.             index_id_capture = nama_kolom.index('id_capture'.upper())
249.             #print index_id_capture
250.
251.
252.             kolom_insert = list(nama_kolom)
253.             del kolom_insert[index_id_capture]
254.             del kolom_insert[index_action]
255.             del kolom_insert[index_id]
```

```
256.
257.            kolomToString = ', '.join(kolom_insert)
258.
259.            kolom_update = list(nama_kolom)
260.            del kolom_update[index_id_capture]
261.            del kolom_update[index_action]
262.            del kolom_update[index_id]
263.
264.            print kolom_update
265.
266.            print kolom_insert
267.
268.            list_data_insert = []
269.
270.            for index in range(len(data_capture)):
271.
272.                if data_capture[index][index_action] == 'insert':
273.                    # print 'insert captured'
274.                    # print data_capture[index]
275.                    data = tuple_without(data_capture[index], index_id_capture)
276.                    data1 = tuple_without(data, index_action)
277.                    data2 = tuple_without(data1, index_id)
278.                    # print data
279.                    copy_data2 = list(data2)
280.                    list_insert_value = []
281.
282.                    for index3 in range(len(copy_data2)):
283.                        if str(type(copy_data2[index3])) == "<type 'long'>":
```

```
284.                             list_insert_value.append(int(copy_data2[index3]))
285.                         else:
286.                             list_insert_value.append(copy_data2[index3])
287.
288.                     # print list_insert_value
289.
290.
291.                     insert_sql_to_send = "INSERT INTO " + list_nama_tabel_old[
292.                         index_tabel_capture] + " (" + kolomToString + ") VALUES %r" % (tuple(list_insert_value),)
293.                     insert_sql_to_send.replace("None", "NULL")
294.                     print insert_sql_to_send
295.
296.                     delete_data_capture = "DELETE FROM " + list_capture[
297.                         index_tabel_capture] + " WHERE id_capture = " + repr(
298.                         int(data_capture[index][index_id_capture])) + ""
299.                     print delete_data_capture
300.
301.                     count = 0
302.                     for index_tujuan1 in range(len(list_other_IP_nodes)):
303.                         payload = {'sumber': own_ip, 'tujuan': list_other_IP_nodes[index_tujuan1],
304.                                    'data': insert_sql_to_send, 'nama_tabel':
    list_nama_tabel_old[index_tabel_capture]}
305.                         try:
306.                             r = requests.post("http://"+center_node+"/queue", json=payload)
307.                             print r.json()['pesan']
308.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
309.                                 count += 1
310.                         except:
```

```
311.                             print "node "+list_other_IP_nodes[index_tujuan1]+" tidak ditemukan"
312.                         # print(r.text)
313.
314.                     print count
315.                     if len(list_other_IP_nodes) == count:
316.                         try:
317.                             cursor.execute(delete_data_capture)
318.                             db.commit()
319.
320.                         except:
321.                             db.rollback()
322.
323.                 elif data_capture[index][index_action] == 'update':
324.                     data = tuple_without(data_capture[index], index_id_capture)
325.                     data1 = tuple_without(data, index_action)
326.                     data2 = tuple_without(data1, index_id)
327.                     to_send = []
328.
329.                     for index2 in range(len(data2)):
330.                         if str(type(data2[index2])) == "<type 'long'>":
331.                             # print "flag"
332.                             # test.append(data2[index2])
333.                             to_send.append(kolom_update[index2] + "=" + repr(int(data2[index2])))
334.                         else:
335.                             to_send.append(kolom_update[index2] + "=" + repr(data2[index2]))
336.
337.                     value = ",".join(to_send)
338.
```

```
339.                     update_to_send = "UPDATE " + list_nama_tabel_old[index_tabel_capture] + " SET " + value + "
    WHERE " + \
340.                                      list_primary[index_tabel_capture] + " = " + repr(
341.                         int(data_capture[index][index_id])) + ""
342.                     print update_to_send
343.
344.                     delete_data_capture = "DELETE FROM " + list_capture[
345.                         index_tabel_capture] + " WHERE id_capture = " + repr(
346.                         int(data_capture[index][index_id_capture])) + ""
347.                     print delete_data_capture
348.
349.                     count = 0
350.                     for index_tujuan2 in range(len(list_other_IP_nodes)):
351.                         payload = {'sumber': own_ip, 'tujuan': list_other_IP_nodes[index_tujuan2],
352.                                    'data': update_to_send, 'nama_tabel':
    list_nama_tabel_old[index_tabel_capture]}
353.                         try:
354.                             r = requests.post("http://"+center_node+"/queue", json=payload)
355.                             print r.json()['pesan']
356.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
357.                                 count += 1
358.                         except:
359.                             print "node "+list_other_IP_nodes[index_tujuan2]+" tidak ditemukan"
360.                         # print(r.text)
361.
362.                     print count
363.                     if len(list_other_IP_nodes) == count:
364.                         try:
```

```
365.                             cursor.execute(delete_data_capture)
366.                             db.commit()
367.
368.                         except:
369.                             db.rollback()
370.
371.                 elif data_capture[index][index_action] == 'delete':
372.
373.                     delete_to_send = "DELETE FROM " + list_nama_tabel_old[index_tabel_capture] + " WHERE " +
    list_primary[
374.                         index_tabel_capture] + " = " + repr(int(data_capture[index][index_id])) + ""
375.                     print delete_to_send
376.
377.                     delete_data_capture = "DELETE FROM " + list_capture[
378.                         index_tabel_capture] + " WHERE id_capture = " + repr(
379.                         int(data_capture[index][index_id_capture])) + ""
380.                     print delete_data_capture
381.
382.                     count = 0
383.                     for index_tujuan3 in range(len(list_other_IP_nodes)):
384.                         payload = {'sumber': own_ip, 'tujuan': list_other_IP_nodes[index_tujuan3],
385.                                    'data': delete_to_send, 'nama_tabel':
    list_nama_tabel_old[index_tabel_capture]}
386.                         try:
387.                             r = requests.post("http://"+center_node+"/queue", json=payload)
388.                             print r.json()['pesan']
389.                             if r.json()['pesan'] == "query berhasil masuk ke dalam queue":
390.                                 count += 1
```

```
391.                         except:
392.                             print "node "+list_other_IP_nodes[index_tujuan3]+" tidak ditemukan"
393.                         # print(r.text)
394.
395.                     print count
396.                     if len(list_other_IP_nodes) == count:
397.                         try:
398.                             cursor.execute(delete_data_capture)
399.                             db.commit()
400.
401.                         except:
402.                             db.rollback()
403.
404.
405.
406.         cursor.close()
407.         db.close()
408.     waktu_sinkronisasi = int(ConfigSectionMap("timer_sinkronisasi")['time'])
409.     threading.Timer(waktu_sinkronisasi, sinkronisasi).start()
410.
411. sinkronisasi()
412.
413. # payload = {'test': 'value1', 'key2': 'value2'}
414. # r = requests.post("http://localhost:5000/test", data=payload)
415. # print(r.text)
```

Kode Sumber 4 Fungsi Test Post Request

```
list_other_IP_nodes = [e.strip() for e in ConfigSectionMap("other-nodes")['ip'].split(',')]
def cek_tujuan_ada(list_other_IP_nodes):
    list_tujuan = list_other_IP_nodes
    copy_list_tujuan = list(list_tujuan)
    for index in range(len(list_tujuan)):
        # print list_tujuan[index]
        payload = {'dummy': "test"}
        try:
            r = requests.post("http://" + list_tujuan[index] + "/test_request", timeout=0.05, json=payload)
            if r.json()['pesan'] == "node siap digunakan":
                del copy_list_tujuan[index]
        except:
            print "service node tidak ditemukan"
        # print copy_list_tujuan
    # print copy_list_tujuan
    return copy_list_tujuan
```

Kode Sumber 5 Web Service pada Slave Node

```
__author__ = 'Indra Gunawan'
from flask import Flask, jsonify
from flask import request
from flask import abort
from flask import json
import ConfigParser
import MySQLdb
import cx_Oracle
```

```
9.
10. Config = ConfigParser.ConfigParser()
11.
12. Config.read("config.ini")
13. def ConfigSectionMap(section):
14.     dict1 = {}
15.     options = Config.options(section)
16.     for option in options:
17.         try:
18.             dict1[option] = Config.get(section, option)
19.             if dict1[option] == -1:
20.                 print ("skip: %s" % option)
21.         except:
22.             print("exception on %s!" % option)
23.             dict1[option] = None
24.     return dict1
25.
26. own_ip = ConfigSectionMap("host")['ip']
27. own_port = int(ConfigSectionMap("host")['port'])
28.
29. ip_db = ConfigSectionMap("database")['ip']
30. db = ConfigSectionMap("database")['db']
31. username = ConfigSectionMap("database")['username']
32. password = ConfigSectionMap("database")['password']
33. jenis = ConfigSectionMap("database")['jenis']
34. if jenis == "mysql":
35.     db = MySQLdb.connect(ip_db, username, password, db)
36.     cursor = db.cursor()
```

```
37.  elif jenis=="oracle":
38.      db = cx_Oracle.connect(username + '/' + password + '@' + ip_db + '/' + db + '')
39.      cursor = db.cursor()
40.
41.
42.  def checkTabelExist(tabel):
43.      if jenis =="mysql":
44.          checkTabelSql = "SELECT * FROM "+tabel+" LIMIT 1"
45.      elif jenis =="oracle":
46.          checkTabelSql = "SELECT * FROM " + tabel + " WHERE ROWNUM = 1"
47.
48.      try:
49.          cursor.execute(checkTabelSql)
50.          data_capture = cursor.fetchall()
51.          print data_capture
52.          return "tabel ada"
53.      except:
54.          return "tabel tidak ada"
55.
56.  def lakukan_query(query):
57.      # query_dari_node_lain = query
58.
59.      try:
60.          cursor.execute(query)
61.          db.commit()
62.          return "ok"
63.      except:
64.          db.rollback()
```

```
65.         return "gagal"
66.
67.
68. app = Flask(__name__)
69.
70. @app.route("/test_request", methods=["POST"])
71. def test_req():
72.     return jsonify({"pesan": "node siap digunakan"})
73.
74.
75. @app.route("/receiver", methods=["POST"])
76. def test():
77.     content = request.json
78.     query = content['data']
79.     tabel = content['nama_tabel']
80.     print tabel
81.     hasil_check_tabel = checkTabelExist(tabel)
82.     if hasil_check_tabel=="tabel ada":
83.         hasil_query = lakukan_query(query)
84.         if hasil_query=="ok":
85.             return jsonify({"pesan": "sinkronisasi berhasil"})
86.         else:
87.             return jsonify({"pesan": "sinkronisasi gagal","hasil_query":hasil_query,})
88.
89.     elif hasil_check_tabel=="tabel tidak ada":
90.         return jsonify({"pesan":"Tidak ada tabel dengan nama tersebut"})
91.
92.
```

```
93.  if __name__ == '__main__':
94.      app.run(host=own_ip, port=own_port, debug=True,threaded=True)
95.
```

Kode Sumber 6 Pembuatan Queue

```
1.    __author__ = 'Indra Gunawan'
2.   import ConfigParser
3.   import MySQLdb
4.
5.
6.   Config = ConfigParser.ConfigParser()
7.
8.   Config.read("config.ini")
9.   def ConfigSectionMap(section):
10.      dict1 = {}
11.      options = Config.options(section)
12.      for option in options:
13.          try:
14.              dict1[option] = Config.get(section, option)
15.              if dict1[option] == -1:
16.                  print ("skip: %s" % option)
17.          except:
18.              print("exception on %s!" % option)
19.              dict1[option] = None
20.      return dict1
21.
```

```
22.  ip = ConfigSectionMap("database")['ip']
23.  db = ConfigSectionMap("database")['db']
24.  username = ConfigSectionMap("database")['username']
25.  password = ConfigSectionMap("database")['password']
26.
27.  jenis = ConfigSectionMap("database")['jenis']
28.  db = MySQLdb.connect(ip, username, password, db)
29.  cursor = db.cursor()
30.
31.  sql_check_tabel = "DROP TABLE IF EXISTS QUEUE"
32.
33.  try:
34.      cursor.execute(sql_check_tabel)
35.  except:
36.      print "drop table error. silahkan cek config"
37.
38.  sql = """CREATE TABLE QUEUE (
39.           id INT PRIMARY KEY AUTO_INCREMENT,
40.           query TEXT,
41.           nama_tabel TEXT,
42.           sumber TEXT,
43.           tujuan TEXT )"""
44.
45.
46.  try:
47.      cursor.execute(sql)
48.      print "tabel queue berhasil dibuat"
49.  except:
```

```
50.      print "create table error, mungkiin table sudah ada"
51.
52.  db.close()
```

Kode Sumber 7 Distribusi dari Queue ke Slave Node

```
1.   __author__ = 'Indra Gunawan'
2.   import requests
3.   import ConfigParser
4.   import MySQLdb
5.   import cx_Oracle
6.   import threading
7.
8.   Config = ConfigParser.ConfigParser()
9.
10.  Config.read("config.ini")
11.  def ConfigSectionMap(section):
12.      dict1 = {}
13.      options = Config.options(section)
14.      for option in options:
15.          try:
16.              dict1[option] = Config.get(section, option)
17.              if dict1[option] == -1:
18.                  print ("skip: %s" % option)
19.          except:
```

```
20.             print("exception on %s!" % option)
21.             dict1[option] = None
22.     return dict1
23.
24. def tuple_without(original_tuple, element_to_remove):
25.     lst = list(original_tuple)
26.     # lst.remove(element_to_remove)
27.     del lst[element_to_remove]
28.     return tuple(lst)
29.
30. def cek_tujuan_ada(list_other_IP_nodes):
31.     list_tujuan = list_other_IP_nodes
32.     copy_list_tujuan = list(list_tujuan)
33.     for index in range(len(list_tujuan)):
34.         # print list_tujuan[index]
35.         payload = {'dummy': "test"}
36.         try:
37.             r = requests.post("http://" + list_tujuan[index] + "/test_request", timeout=0.05, json=payload)
38.             if r.json()['pesan'] == "node siap digunakan":
39.                 del copy_list_tujuan[index]
40.         except:
41.             print "service node tidak ditemukan"
42.         # print copy_list_tujuan
43.     # print copy_list_tujuan
44.     return copy_list_tujuan
45.
46. def sinkronisasi_dari_pusat():
47.     ip = ConfigSectionMap("database")['ip']
```

```
48.     db = ConfigSectionMap("database")['db']
49.     username = ConfigSectionMap("database")['username']
50.     password = ConfigSectionMap("database")['password']
51.     list_other_IP_nodes = [e.strip() for e in ConfigSectionMap("other-nodes")['ip'].split(',')]
52.     center_node = ConfigSectionMap("center-node")['ip']
53.     own_ip = ConfigSectionMap("host")['ip']
54.     # print type(list_other_IP_nodes)
55.     print list_other_IP_nodes
56.
57.     # list_nama_tabel_old = [e.strip() for e in ConfigSectionMap("tabel")['nama'].split(',')]
58.     # list_capture = [e.strip() for e in ConfigSectionMap("tabel_capture")['nama'].split(',')]
59.     # list_primary = [e.strip() for e in ConfigSectionMap("tabel")['kolom_primary'].split(',')]
60.     jenis = ConfigSectionMap("database")['jenis']
61.
62.     db = MySQLdb.connect(ip, username, password, db)
63.     cursor = db.cursor()
64.
65.     list_node_terdeteksi = cek_tujuan_ada(list_other_IP_nodes)
66.     # print list_node_terdeteksi
67.
68.     if len(list_node_terdeteksi) == 0:
69.         sql_not_in = "SELECT * FROM QUEUE LIMIT 10"
70.     else:
71.         sql_not_in = "SELECT * FROM QUEUE WHERE tujuan NOT IN %r" % (tuple(list_node_terdeteksi),)
72.
73.
74.     try:
75.         cursor.execute(sql_not_in)
```

```
76.     except:
77.         print "unable to extract queue karena queue tidak ada"
78.
79.
80.     data_queue = cursor.fetchall()
81.
82.     print data_queue
83.
84.     nama_kolom = [i[0] for i in cursor.description]
85.
86.     print nama_kolom
87.
88.     for index in range(len(data_queue)):
89.
90.         print data_queue[index]
91.         payload = {'data': data_queue[index][1],'nama_tabel': data_queue[index][2]}
92.         print payload
93.
94.         try:
95.             r = requests.post("http://"+data_queue[index][4]+"/receiver", timeout=0.5,json=payload)
96.             print r.json()['pesan']
97.             if r.json()['pesan'] == "sinkronisasi berhasil":
98.                 delete_data_queue = "DELETE FROM QUEUE WHERE ID = " + repr(int(data_queue[index][0])) + ""
99.                 print delete_data_queue
100.                try:
101.                    cursor.execute(delete_data_queue)
102.                    db.commit()
103.                    print "data queue berhasil dihapus"
```

```
104.
105.                 except:
106.                     db.rollback()
107.                     print "data queue gagal dihapus"
108.             else:
109.                 print "node tujuan tidak ditemukan atau sinkronisasi gagal"
110.         except:
111.             print "node " +data_queue[index][4]+ " tidak ditemukan"
112.     db.close()
113.
114.     waktu_sinkronisasi = int(ConfigSectionMap("timer_sinkronisasi")['time'])
115.     threading.Timer(waktu_sinkronisasi, sinkronisasi_dari_pusat).start()
116.
117. sinkronisasi_dari_pusat()
```

Kode Sumber 8 Web Service pada Queue

```
1.  __author__ = 'Indra Gunawan'
2.  from flask import Flask, jsonify
3.  from flask import request
4.  from flask import abort
```

```
from flask import json
import ConfigParser
import MySQLdb
import cx_Oracle

Config = ConfigParser.ConfigParser()

Config.read("config.ini")
def ConfigSectionMap(section):
    dict1 = {}
    options = Config.options(section)
    for option in options:
        try:
            dict1[option] = Config.get(section, option)
            if dict1[option] == -1:
                print ("skip: %s" % option)
        except:
            print("exception on %s!" % option)
            dict1[option] = None
    return dict1

own_ip = ConfigSectionMap("host")['ip']
own_port = int(ConfigSectionMap("host")['port'])

ip_db = ConfigSectionMap("database")['ip']
db = ConfigSectionMap("database")['db']
username = ConfigSectionMap("database")['username']
password = ConfigSectionMap("database")['password']
```

```
33. jenis = ConfigSectionMap("database")['jenis']
34. if jenis == "mysql":
35.     db = MySQLdb.connect(ip_db, username, password, db)
36.     cursor = db.cursor()
37. elif jenis=="oracle":
38.     db = cx_Oracle.connect(username + '/' + password + '@' + ip_db + '/' + db + '')
39.     cursor = db.cursor()
40.
41. def insert_to_queue(val_query, val_nama, val_sumber, val_tujuan):
42.
43.     sql = "INSERT INTO QUEUE (QUERY, NAMA_TABEL, SUMBER, TUJUAN) VALUES (%r,'%s','%s','%s')" %
    (str(val_query),val_nama,val_sumber,val_tujuan)
44.     print sql
45.     try:
46.         cursor.execute(sql)
47.         db.commit()
48.         print "berhasil"
49.         return "berhasil insert queue"
50.     except:
51.         db.rollback
52.         return "gagal"
53.
54.
55. app = Flask(__name__)
56.
57. @app.route("/queue", methods=["POST"])
58. def test():
59.     content = request.json
```

```
60.      query = content['data']
61.      tabel = content['nama_tabel']
62.      sumber = content['sumber']
63.      tujuan = content['tujuan']
64.
65.      hasil_queue = insert_to_queue(query,tabel,sumber,tujuan)
66.      print query
67.      print hasil_queue
68.
69.      if hasil_queue=="berhasil insert queue":
70.          return jsonify({"pesan": "query berhasil masuk ke dalam queue"})
71.
72.      elif hasil_queue=="gagal":
73.          return jsonify({"pesan":"query gagal masuk ke dalam queue"})
74.
75.
76.  if __name__ == '__main__':
77.      app.run(host=own_ip, port=own_port, debug=True)
```

BIODATA PENULIS

I Putu Gede Indra Gunawan, lahir pada tanggal 19 Juli 1995 di Denpasar. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain membaca novel, bermain *game*, dan membaca komik. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Riset dan Teknologi Himpunan Mahasiswa Teknik Computer-Informatika dan Staff Departemen Komunikasi dan Informasi Tim Pembina Kerohanian Hindu ITS pada tahun ke-2, serta Staff Ahli Departemen Komunikasi dan Informasi Tim Pembina Kerohanian Hindu ITS pada tahun ke-3.

Kritik dan saran sangat diharapkan guna peningkatan kualitas dan penulisan selanjutnya. Untuk itu, silahkan kirim kritik dan saran ke : **i.pt.gd.indra.gunawan@gmail.com**