Implementasi *Socket Programming* Sebagai Media Sinkronisasi *Database* Terdistribusi dengan Teknik *Multi Master Replication*

Made Pradnyana Ambara[1], Pande Ketut Widiartana[2], Yohanes Priyo Atmojo[3]
Institut Teknologi dan Bisnis STIKOM Bali
e-mail: [1]pradnyana_ambara@yahoo.com, [2]widigent.coder@gmail.com, [3]yohanes@stikom-bali.ac.id

Diajukan: 30 Januari 2020; Direvisi: 4 Juni 2020; Diterima: 17 Juni 2020

Abstrak

Dalam sebuah database terdistribusi, database disimpan tersebar di banyak lokasi yang terpisah namun saling berhubungan satu sama lain. Sinkronisasi data pada database terdistribusi yang heterogenous, dengan sumber data yang secara geografis terletak tersebar di beberapa lokasi yang sangat jauh adalah salah satu masalah dalam penerapannya. Sinkronisasi database yang dilakukan dengan cara export file SQL secara manual di sistem sumber, kemudian dikirimkan melalui email untuk di import di sistem tujuan sangat tidak efektif di mana sangat sering terjadi kesalahan pada sisi user dalam export import file-nya. Penelitian ini bertujuan memberikan solusi atas permasalahan tersebut dengan menerapkan teknik pemrograman socket untuk membangun aplikasi yang berfungsi sebagai media sinkronisasi otomatis pada datatabase terdistribusi. Hasil dari penelitian ini adalah sebuah middleware yang dapat menyinkronkan data pada database terdistribusi menggunakan teknik multi master replication. Teknik ini dapat meningkatkan availability dari database sehingga ketika terjadi kerusakan atau kegagalan pada satu lokasi fisik maka tidak menyebabkan kegagalan keseluruhan sistem pada database. Middleware ini akan membaca setiap perubahan pada database kemudian mengirimkannya melalui komunikasi socket ke aplikasi server. Aplikasi server kemudian akan mem-broadcast perubahan tersebut ke semua client yang terhubung. Metode penelitian yang digunakan adalah metode waterfall model meliputi pengumpulan data, analisa dan desain sistem serta pembuatan dan uji coba.

Kata kunci: *Sinkronisasi database terdistribusi, Socket programming, Multi master replication.*

Abstract

In a distributed database, databases are stored scattered in many separate but interconnected locations. Synchronizing data in a heterogeneous distributed database, with data sources that are geographically located scattered in several locations very far away is one of the problems in its application. Database synchronization which is done by manually exporting SQL files in the source system, then sent by email to be imported in the destination system is very ineffective where very often there are errors on the user side in the export import file. This study aims to provide a solution to these problems by applying socket programming techniques to build applications that function as automatic synchronization media in a distributed database. The result of this research is a middleware that can synchronize data in a distributed database using multi master replication techniques. This technique can increase the availability of a database so that when there is damage or failure at one physical location, it does not cause an overall system failure in the database. This middleware will read every change in the database then send it via socket communication to the server application. The server application will then broadcast the changes to all connected clients. The research method used is the waterfall model method including data collection, analysis and system design as well as manufacturing and testing.

Keywords*: Distributed database synchronization, Socket programming, Multimaster replication.*

1. Pendahuluan

Dalam perkembangannya, semakin besar sebuah perusahaan, maka semakin besar dan semakin kompleks juga data yang diproses sehingga diperlukan sebuah teknologi untuk menangani masalah ini. Pendistribusian *database* merupakan salah satu langkah yang sangat tepat apabila jumlah *client* yang ditangani oleh suatu aplikasi semakin bertambah banyak. Masalah yang dihadapi oleh perusahaan yang

menerapkan sistem *database* terdistribusi yang *heterogenous*, dengan sumber data yang secara geografis terletak tersebar di beberapa lokasi dengan jarak yang sangat jauh adalah bagaimana menyinkronkan data tersebut secara otomatis dan *realtime*. Sinkronisasi antar *database* yang dilakukan dengan cara *export file Structured Query Language* (SQL) secara manual di sistem sumber, yang kemudian dikirimkan melalui *email* untuk di-*import* di sistem tujuan sangat tidak efektif di mana sangat sering terjadi kesalahan pada sisi *user* dalam *export* ataupun *import file*-nya. Kesalahan yang paling sering terjadi adalah lupa meng-*export* data transaksi dari satu tempat untuk di-*import* di tempat lain atau lupa meng-i*mport* data transaksi di satu tempat dari tempat lain, yang menyebabkan data menjadi tidak *valid* sehingga sangat diperlukan *tools* berupa *middleware* yang dapat melakukan sinkronisasi secara otomatis.

Dalam penelitian yang berjudul "Implementasi *Heterogenous Distributed Database System* Oracle Xe 10g dan MySQL" menjelaskan tentang implementasi *heterogenous distributed database* dengan replikasi horizontal menghasilkan sistem informasi yang dapat berjalan dengan baik walaupun salah satu atau lebih server mengalami masalah seperti *down server* atau kerusakan pada basis data. Data yang sedang diproses juga tidak akan hilang, karena sudah dilakukan replikasi data secara horizontal ke *Database Management System* (DBMS) *server* yang sedang aktif. Namun demikian masih diperlukan sinkronisasi manual untuk melakukan *update* data sehingga data pada *server* yang sedang melakukan sinkronisasi memiliki data yang sama dengan *server* lainnya[1]. Pada penelitian yang berjudul "Implementasi *Socket* TCP/IP untuk Mengirim dan Memasukkan *File Text* ke Dalam *Database*" menghasilkan simpulan bahwa menggunakan *socket*, transfer data bisa ditransmisikan antara program aplikasi dan mengirimkan *file* ataupun *input file* dan TCP/IP layak sebagai sarana pengiriman data ke aplikasi lain [2]. Penelitian lain dengan judul "Program *Socket* untuk Mengirim *File* dengan *Visual Basic* pada Sistem Operasi Windows" menghasilkan kesimpulan bahwa aplikasi pada jaringan komputer, transaksinya didasarkan pada konsep *client-server* dengan menggunakan protokol *transport* untuk saling berinteraksi [3].

Penulis melakukan penelitian ini dengan tujuan untuk memberikan solusi terhadap masalah bagaimana cara untuk sinkronisasi data pada *database* terdistribusi yang *heterogenous* secara otomatis dan *realtime*, dengan mengimplementasikan *socket programming* sebagai media sinkronisasi *database* terdistribusi dengan teknik *multimaster replication* dengan membangun *middleware application*. Penulis memilih metode ini dengan mengembangkan penelitian oleh [1], di mana dengan mengirimkan tiap perubahan data pada *database* melalui komunikasi *socket* tidak diperlukan lagi sinkronisasi secara manual dan dengan didukung penelitian yang dilakukan oleh [2] bahwa komunikasi *socket* TCP/IP layak sebagai sarana pengiriman data. Dengan teknik ini setiap perubahan data pada semua *database client* akan dapat direplika secara *multimaster*, otomatis, dan *realtime*.

2. Metode Penelitian

2.1. *Socket Programming*, *Client,* dan *Server*

Socket adalah sebuah cara untuk berkomunikasi dengan program atau *node* lain menggunakan *file descriptor* [4]. Unix dengan slogannya "*everything is a file*" merupakan tempat di mana *socket* diciptakan, membuat komunikasi dengan program atau *node* menjadi semudah membaca dan menulis *file descriptor*. *Socket client* adalah sebuah penghubung antara dua *host*, yang dapat dibangun dengan tujuh dasar operasi yang meliputi menghubungkan mesin atau perangkat, kemudian mengirim dan menerima data, menutup koneksi, menyiapkan *port*, *listening* data yang masuk, serta menerima koneksi dari mesin berdasarkan sebuah *port* [4]. *Socket Server* adalah *socket* yang akan digunakan untuk menangani koneksi dari *Client* [4]. *Socket server* digunakan pada aplikasi *server* dengan mengimplementasikan *Class Server Socket* yang menyediakan *constructor* untuk memilih IP mana yang akan digunakan, karena server dapat menggunakan *IP Address* lebih dari satu.

2.2. *Database* Terdistribusi dan *Database Trigger*

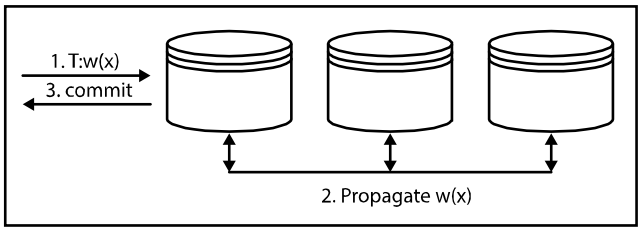
Basis data *(database)* adalah representasi kumpulan fakta yang saling berhubungan disimpan secara bersama sedemikian rupa dan tanpa pengulangan (*redundancy*) yang tidak perlu, untuk memenuhi berbagai kebutuhan [5]. Dengan bantuan perangkat lunak seperti DBMS, data yang tersimpan dalam *database* dapat dikelola dengan mudah dan cepat untuk tujuan penyediaan informasi yang berkualitas. Basis data terdistribusi/*Distributed Database (DDB)* adalah suatu kumpulan berbagai basis data yang secara logika saling berhubungan satu dengan lainnya, yang terdistribusi dalam suatu *network* komputer [2]. Di dalam *database* terdistribusi *file-file database* disimpan dalam beberapa *database* pada komputer-komputer yang secara fisik terpisah satu dengan yang lain. Penyimpanan *database* dapat dilakukan pada DBMS yang sama (*homogenous*) ataupun DBMS yang berbeda (*heterogenous*) namun *database* haruslah memiliki struktur yang sama. *Homogenous Distributed Database* adalah basis data yang menggunakan DBMS sama di setiap simpul. *Heterogenous Distributed Database* yaitu basis data terdistribusi menggunakan DBMS yang

berbeda pada setiap simpul. Dalam penelitian ini digunakan *heterogenous distributed database* di mana *database* pada masing-masing *outlet* menggunakan DBMS MS SQL Server 2008 Express R2, sedangkan pada *database server* digunakan MySQL Versi 14.14 Distribusi 5.7.23. *Trigger* adalah prosedur yang berhubungan dengan *table*, *view*, skema, atau *database* yang dijalankan secara *implicit* pada saat terjadi sebuah *event* [6]. *Trigger* juga dapat diartikan sebagai sebuah *store procedure* yang berjalan secara otomatis saat terjadi modifikasi pada *database*, baik modifikasi penambahan data (*insert)*, perubahan data *(update)*, maupun modifikasi penghapusan data *(delete)*.

2.3. Replikasi

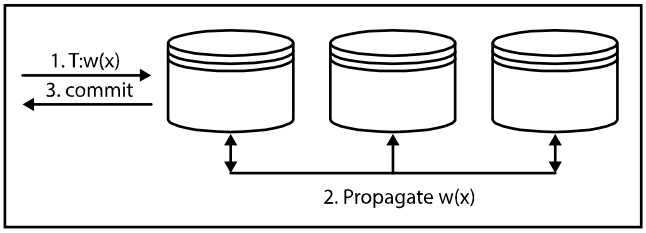
Replikasi adalah suatu teknik untuk melakukan *copy* dan pendistribusian data dan objek-objek *database* dari satu *database* ke *database* lain dan melaksanakan sinkronisasi antara *database* sehingga konsistensi data dapat terjamin [6]. Terdapat dua karakteristik replikasi yaitu:

1. Replikasi *Synchronous*
   Replikasi *Synchronous* bekerja ketika permintaan *update* ke *master database* oleh *slave*, *master* mengunci semua *database* pada tingkat *record*, dan kemudian meng-*update* secara bersamaan.



Gambar 1. Replikasi *synchronous*.

2. Replikasi *Asynchronous*
   Replikasi *asynchronous* bekerja saat terjadi sebuah transaksi pada sebuah *client*, *Client* yang me-*request* mengeksekusi 1 per 1 hasil *respons* dari *client* lainya tadi.



Gambar 2. Replikasi *asynchronous*.

2.4. *Multi Master Database Replication*

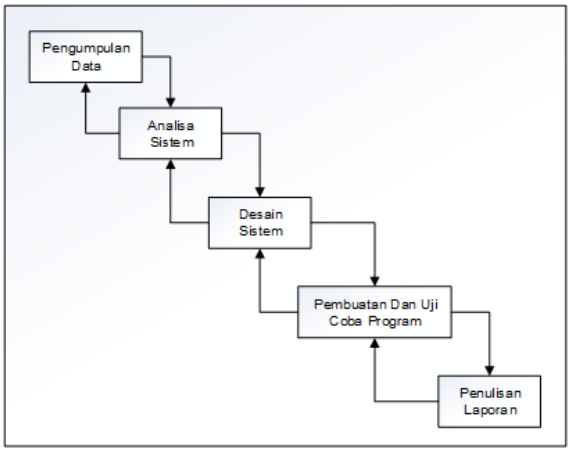
Multi-master database replication memungkinkan seluruh *node* dalam satu klaster menjadi *master database*, sehingga perubahan yang terjadi pada satu *node* akan direplika ke seluruh *node* yang tergabung dalam klaster tersebut [7]. Teknik ini digunakan dalam penelitian karena dapat meningkatkan *availability* dari *database*, di mana *backup database* terdapat di semua *node*, sehingga jika terjadi masalah pada salah satu *node*, maka *database* tersebut dapat diganti/di-*replace* menggunakan *backup* dari *database* yang lain yang masih dalam satu kesatuan *database* terdistribusi tersebut. Dengan teknik ini maka ketersediaan *backup database* menjadi lebih baik.

2.5. JavaFX

JavaFX adalah *platform* klien yang ekspresif untuk menciptakan dan memberikan pengalaman internet yang mendalam di berbagai layar. Aplikasi JavaFX ditulis menggunakan bahasa deklaratif yang diketik secara statis yang disebut *JavaFX Script* yang membuatnya mudah diprogram dalam konteks visual, memungkinkan pengembang untuk membuat GUI yang sangat ekspresif dan intuitif dengan cepat dan mudah [8].

2.6. Alur Kerja Penelitian

Ada beberapa model alur kerja *software* yang umum digunakan, salah satunya adalah model *waterfall*. Model ini melingkupi aktivitas-aktivitas seperti ditunjukkan pada Gambar 3.

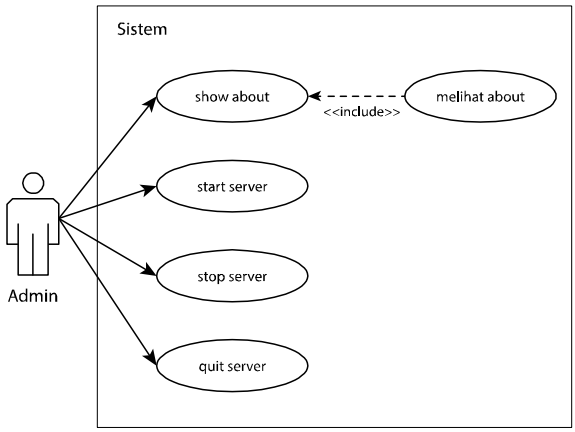


Gambar 3. *Waterfall* model.

Pengumpulan data merupakan proses yang dilakukan untuk memperoleh informasi yang dibutuhkan dalam rangka mencapai tujuan perekayasaan ini, adapun metode pengumpulan data yang digunakan adalah studi pustaka. Analisis Sistem merupakan tahapan menganalisis permasalahan untuk mengetahui dan menentukan batasan-batasan sistem sehingga dapat menentukan cara yang efektif dalam menyelesaikan permasalahan tersebut dan dapat dirancang sebuah aplikasi. Desain sistem pada perekayasaan ini menggunakan UML (*Unified Modeling Language*) yang akan menjelaskan mengenai interaksi pengguna dengan sistem. *Integrated Development Environment* (IDE) yang digunakan dalam penelitian ini adalah Netbean 8 dengan bahasa pemrograman Java dengan JavaFX API dan FXML. Kemudian *database* yang digunakan adalah MS SQL Server dan MySQL server. Pengujian sistem dilakukan untuk mengidentifikasikan dan mengevaluasi permasalahan–permasalahan yang muncul pada saat pengoperasian aplikasi tersebut. Metode pengujian yang digunakan adalah *black box testing*.

2.7. *Use Case Diagram*

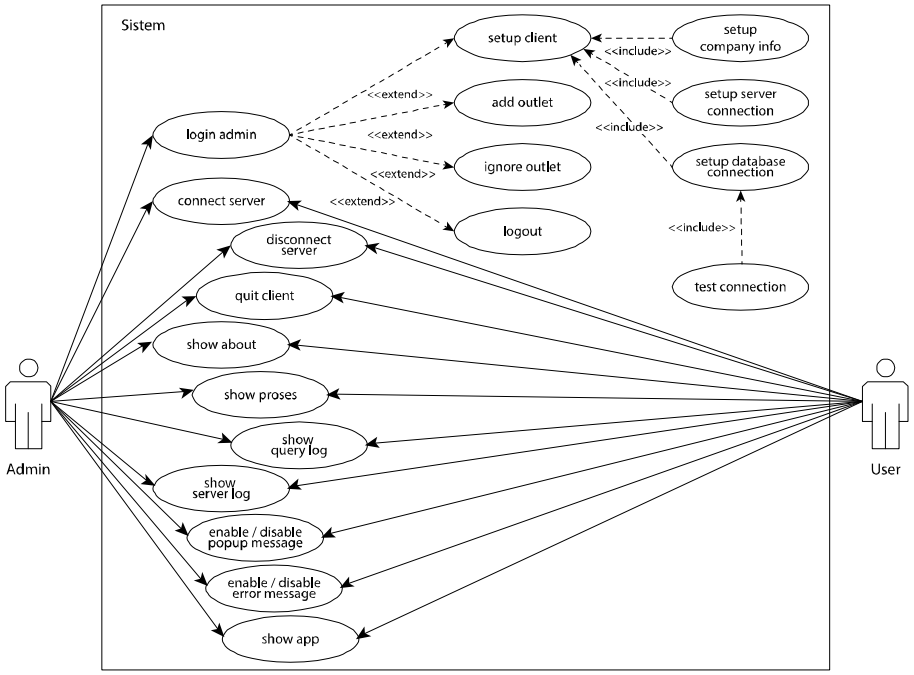
Dalam *use case* terdapat aktivitas apa saja yang dapat dilakukan oleh *admin* pada aplikasi *server*. Pada aplikasi *server* terdapat lima aktivitas yang dilakukan oleh *admin* antara lain *start server, stop server*, *quit server*, dan *show about*. Dalam *show about user* dapat melihat informasi aplikasi *server*. Gambar 4 menunjukkan *use case diagram* pada *server*.



Gambar 4. *Use case diagram server*.

Pada *client* terdapat enam belas aktivitas yang dilakukan oleh dua aktor dengan pembagian sebagai berikut. Admin dapat melakukan *login* sebagai *admin* kemudian dilanjutkan dengan aktivitas *setup client*, *add outlet*, *ignore outlet*, dan *logout*. Pada aktivitas *setup client*, *admin* dapat melakukan aktivitas *setup*

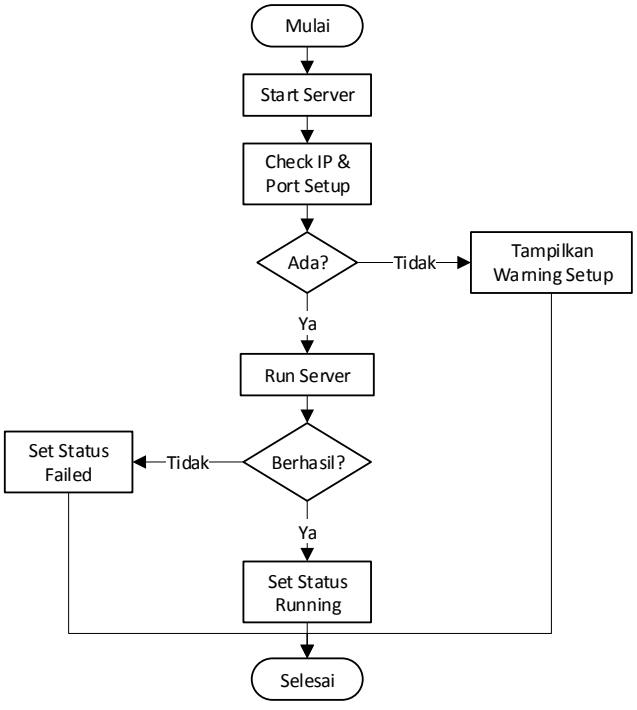
company info, *setup server connection*, *setup database connection*, dan *test connection*. Aktivitas yang dikakukan oleh *client* juga dapat dilakukan oleh *admin*. *Use case diagram* pada *client* ditunjukkan pada Gambar 5.



Gambar 5. *Use case diagram client*.

2.8. *Flowchart Start Server* pada Aplikasi *Server*

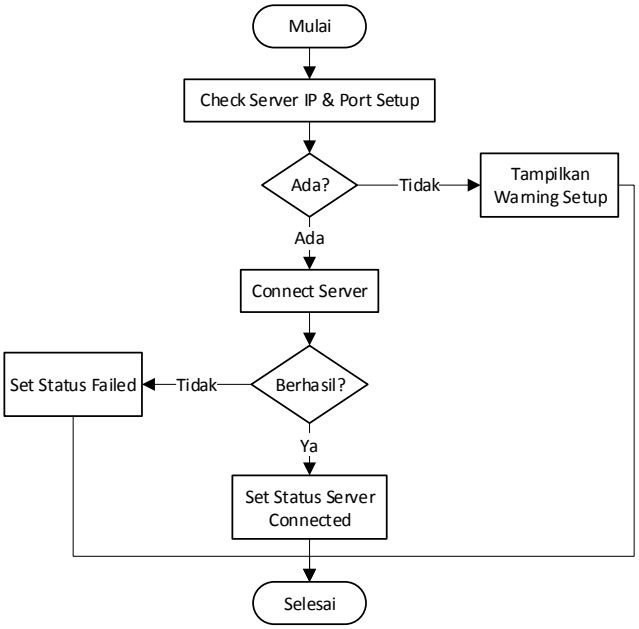
Sebelum aplikasi *client* dapat terkoneksi ke aplikasi server, aplikasi server harus membuka sebuah *Socket server* dengan mengimplementasikan *Class Server Socket* dengan menggunakan *port* dan IP yang telah disimpan dalam setup server. *Flowchart start server* pada aplikasi *server* dapat dilihat pada Gambar 6.



Gambar 6. *Flowchart start server* pada aplikasi server.

2.9. *Flowchart* Koneksi ke *Server* pada Aplikasi *Client*

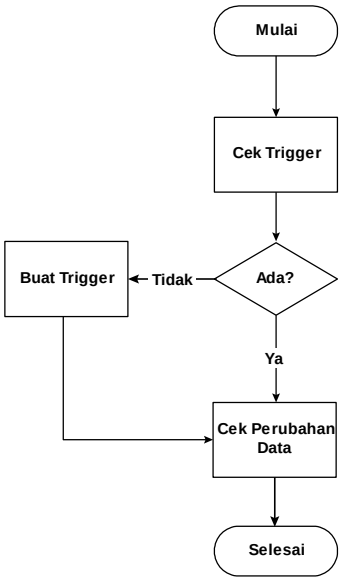
Jika aplikasi *server* telah siap untuk menerima koneksi maka aplikasi *client* siap untuk dikoneksikan ke aplikasi *server*. *Flowchart* koneksi *client* ke *server* dapat dilihat pada Gambar 7.



Gambar 7. *Flowchart* koneksi ke *server* pada aplikasi *client*.

2.10. *Flowchart* Pengecekan *Trigger* pada Aplikasi *Client*

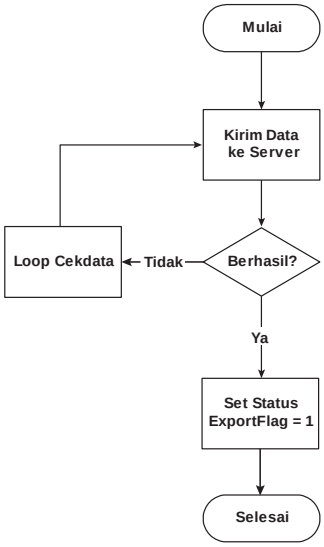
Pada saat aplikasi *client start*, aplikasi akan memeriksa apakah *trigger* sudah ada pada *database client* yang akan disinkronkan. Jika *trigger* tidak ada, maka *trigger* akan ditambahkan ke *database*. *Trigger* ini berfungsi membuat *querystring* dari setiap perubahan pada *database*, baik karena operasi *insert*, *update*, maupun *delete* seperti terlihat pada Gambar 8.



Gambar 8. *Flowchart trigger check*

2.11. *Flowchart* Pengiriman Data pada Aplikasi *Client*

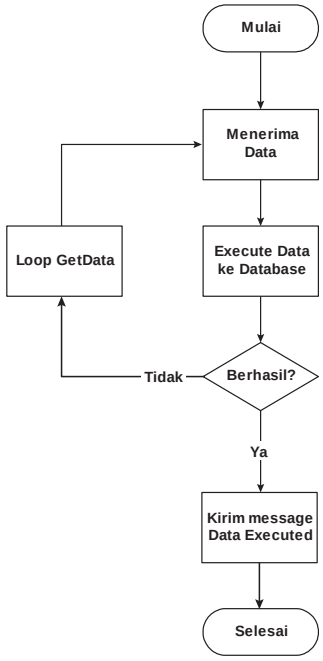
Pada saat aplikasi *client* berjalan, aplikasi akan mengecek apakah ada perubahan pada *database* dengan membaca tabel *CloudQuery* yang diisi data oleh *trigger*. Jika terjadi perubahan data pada *database*, maka tabel *CloudQuery* akan terisi *record* dengan *ExportFlag* status 0. Jika pengiriman data ini ke *server* berhasil, maka aplikasi akan mengubah status dari *record* tersebut seperti Gambar 9.



Gambar 9. *Flowchart* pengiriman data ke *server*.

2.12. *Flowchart* Penerimaan Data pada Aplikasi *Client*

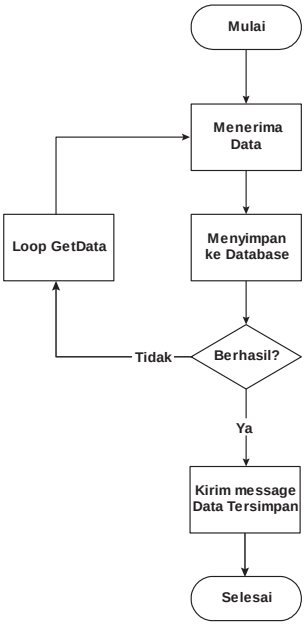
Aplikasi *client* saat menerima data dari *server* akan meng-*execute* data tersebut ke *database*. Jika data yang berupa SQL tersebut berhasil di-*execute*, maka aplikasi akan mengirimkan *message* ke *server* bahwa data berhasil di-*execute*.



Gambar 10. *Flowchart* penerimaan data pada *client*.

2.13. *Flowchart* Penerimaan Data pada Aplikasi *Server*

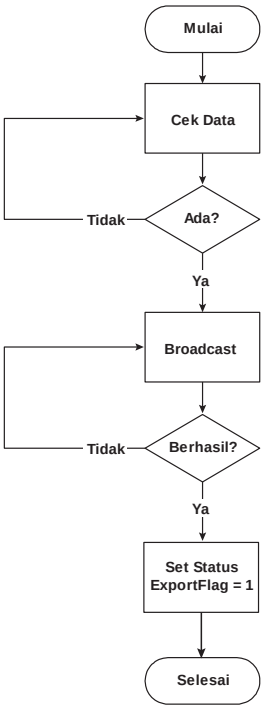
Pada aplikasi *server*, saat menerima data dari *client*, *server* akan menyimpan data tersebut ke dalam *database*. Data disimpan dengan status 0 sebanyak *client* yang ada kecuali *client* pengirim data. Jika penyimpanan data berhasil, aplikasi akan mengirimkan *message* ke *client* pengirim bahwa data sudah berhasil dieksekusi. Data ini kemudian akan dibaca untuk dikirimkan ke setiap *client* kecuali *client* yang mengirim data tersebut.



Gambar 11. *Flowchart* penerimaan data pada *server*.

2.14. *Flowchart Broadcast* Data pada Aplikasi *Server*

Aplikasi *server* secara terus menerus mengecek data pada antrean di *database server*. Jika terdapat data dengan status 0, maka data tersebut akan di-*broadcast* ke semua *client* yang sedang *online*. Jika *broadcast* berhasil, maka status data tersebut di ubah menjadi 1 untuk masing-masing *client* yang berhasil meng-*execute* SQL data tersebut ke *database*-nya.



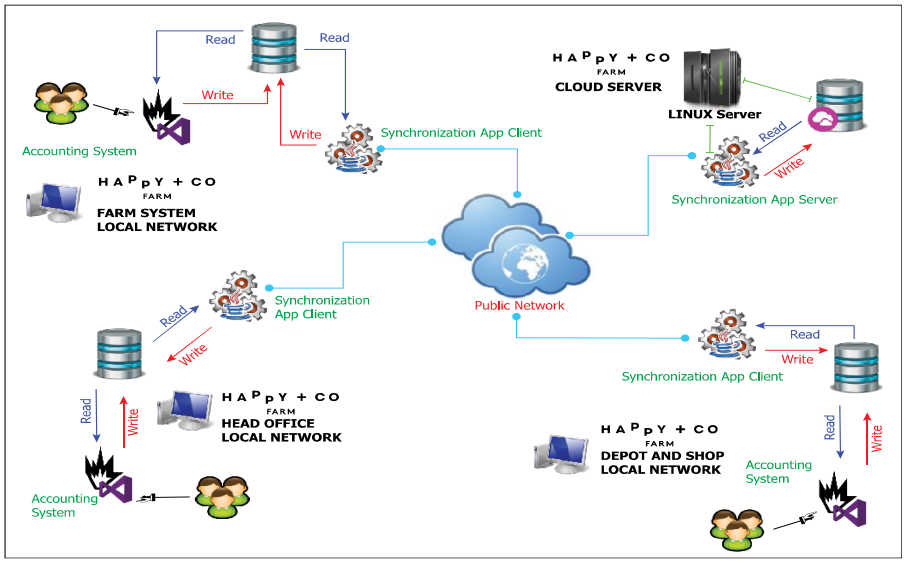
Gambar 12. *Flowchart broadcast* data pada *server*.

3. Hasil dan Pembahasan

3.1. Gambaran Umum Sistem

Aplikasi *middleware* yang dihasilkan dalam penelitian ini berfungsi sebagai media sinkronisasi *database* terdistribusi dengan teknik *multi master replication*. Aplikasi ini berfungsi sebagai *middleware*

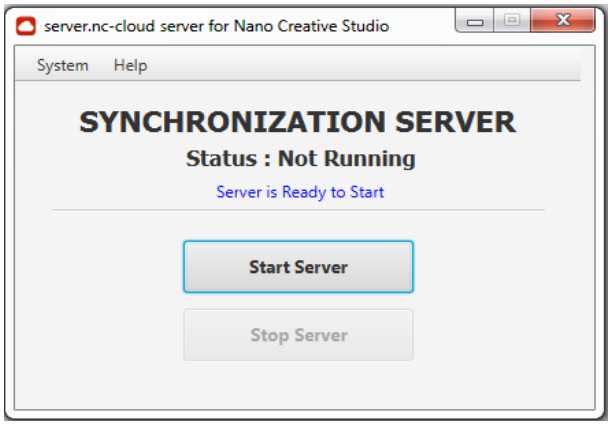
yang akan memeriksa apakah terjadi perubahan pada *database* yang diidentifikasi dari perubahan tabel *CloudQuery* yang berisikan *querystring* dari setiap transaksi yang dilakukan oleh *user* pada aplikasi yang terkoneksi ke *database* tersebut dan kemudian mengirim *querystring* tersebut ke aplikasi *server* untuk kemudian di-*broadcast* ke semua cabang. Gambaran umum sistem ditunjukkan pada Gambar 13.



Gambar 13. Gambaran umum sistem.

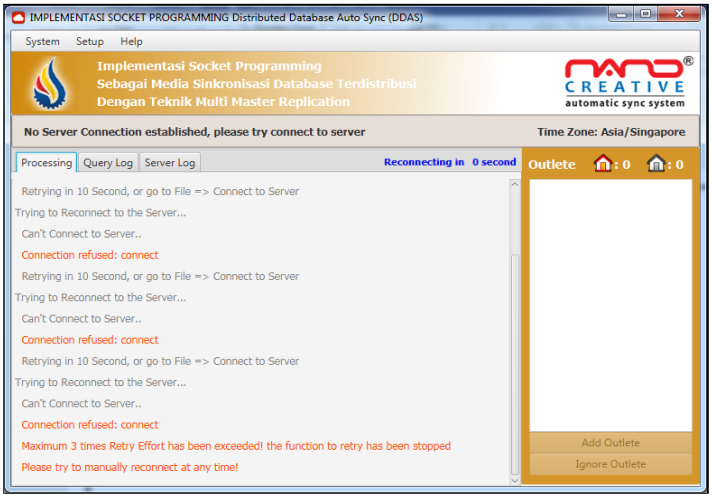
3.2. Implementasi Sistem

Aplikasi ditulis dalam bahasa Java dengan *design* antar mukanya menggunakan JavaFX. Aplikasi terdiri dari aplikasi *server* dan aplikasi *client*. Aplikasi *server* di-*install* di komputer *server* yang dapat dikelola oleh *admin*. Aplikasi ini berfungsi untuk menerima koneksi dari semua *client* dan menyimpan data yang masuk ke *database* dengan *status ExportFlag* = 0. Aplikasi *server* secara terus menerus mengecek data yang *status ExportFlag*-nya = 0 untuk kemudian di-*broadcast* ke semua *client* kecuali *client* pengirim data. Jika *server* menerima *message* bahwa data yang dikirimkan ke *client* telah berhasil di-*execute* oleh *client*, maka aplikasi *server* akan mengubah *status ExportFlag* dari data tersebut menjadi 1. Tampilan aplikasi *server* dapat dilihat pada Gambar 14.



Gambar 14. Tampilan aplikasi *server*.

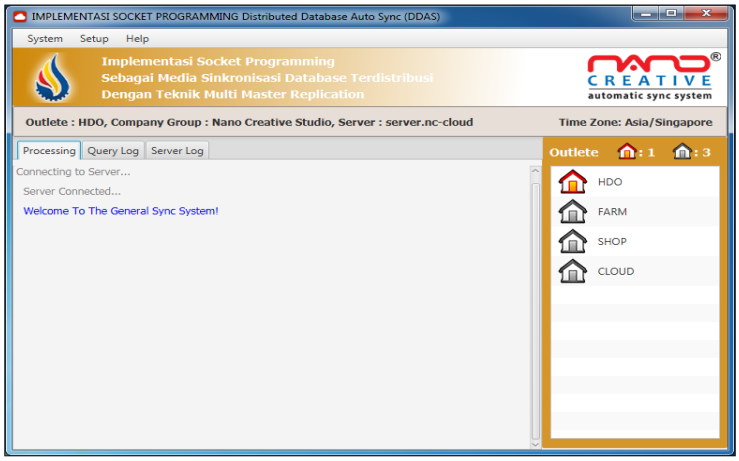
Aplikasi *client* di-*install* di masing-masing *computer outlet*. Aplikasi ini berfungsi untuk mengecek perubahan yang terjadi pada *database* aplikasi dengan bantuan *trigger*. Saat aplikasi dijalankan, aplikasi *client* akan mencoba untuk terkoneksi ke *server*. Jika gagal, aplikasi *client* akan melakukan mencoba secara otomatis sebanyak tiga kali berselang 10 detik. Jika koneksi ke *server* tetap gagal, maka ditampilkan pesan untuk menyarankan koneksi secara manual melalui *menu connect server* seperti terlihat pada Gambar 15.



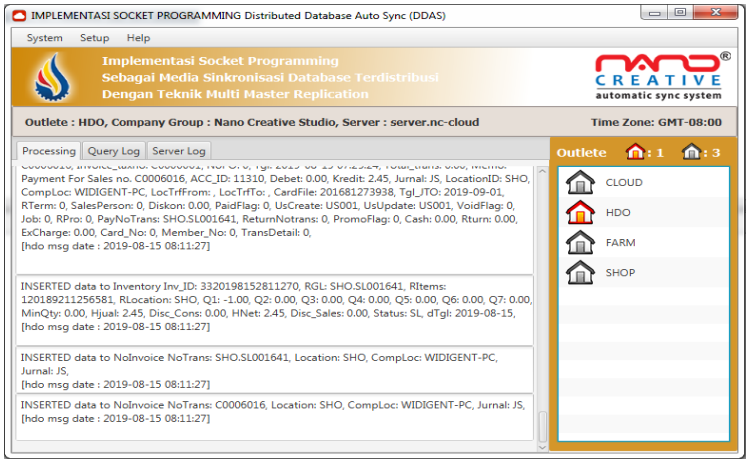
Gambar 15. Tampilan aplikasi *client* gagal terkoneksi ke *server*.

Kegagalan koneksi ke *server* dapat terjadi karena aplikasi server belum di-*start* sehingga tidak ada *socket server* yang terbuka atau dapat juga terjadi karena koneksi jaringan terputus atau *down*.

Jika aplikasi *server* telah terkoneksi ke *server* seperti terlihat pada Gambar 16, dan terdapat antrean data di *server* maka aplikasi *client* akan menerima semua data yang masih mengantre di *database server*. Kemudian *client* akan mengecek apakah terjadi perubahan data pada tabel *CloudQuery*. Jika terdapat perubahan data maka aplikasi akan mengirimkan data tersebut ke *server* dan mengubah *export flag* menjadi 1 jika berhasil. Tampilan aplikasi *client* pada saat sinkronisasi data dapat dilihat pada Gambar 17.

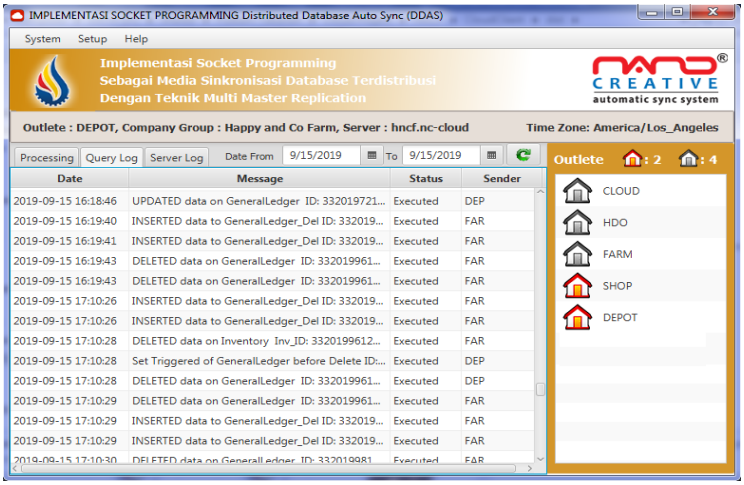


Gambar 16. Tampilan aplikasi *client* setelah berhasil terhubung dengan *server*.



Gambar 17. Tampilan aplikasi *client* saat sinkronisasi data.

Hasil eksekusi *query* yang dikirimkan oleh *server* dapat dilihat pada *tab query log* pada aplikasi *client*. Terdapat kolom tanggal, *message*, status, dan *sender*. Kolom tanggal menampilkan data tanggal dan waktu *query* dieksekusi, kolom *message* menampilkan transaksi yang dieksekusi dan *detail* datanya. Kolom status menampilkan status dari *query* tersebut telah dieksekusi dan kolom *sender* menampilkan data *outlet* yang mengirim *query* tersebut. *Log query* ini dapat dilihat pada Gambar 18.



Gambar 18. Tampilan aplikasi *client* saat sinkronisasi data.

3.3. Hasil Pengujian

Pengujian yang dilakukan meliputi *respons system* saat *client* mencoba terhubung dengan *server*, *respons system* jika *client* telah terkoneksi dengan *server*, dan *respons system* saat terjadi perubahan pada *database client*. Pengujian dilakukan oleh 3 orang *tester* yang memiliki kemampuan dibidang pemrograman *database* yang memfokuskan pada fungsional kinerja sistem. Hasil pengujian dengan metode *black box testing*, bahwa sistem yang dirancang 100% sesuai dengan apa yang diharapkan. Skenario pengujian dapat dilihat pada Tabel 1 dan hasil kuesioner *check list* pengujian pada Tabel 2.

Tabel 1. Skenario pengujian *black box testing*

No	Skenario Pengujian	Masukan	Hasil yang diharapkan	Hasil Pengujian
1	Mengetahui respons sistem jika *client* mencoba terhubung dengan *server*	*User* dan *admin* memilih menu *connect to server*	Sistem menampilkan informasi *server* terhubung dan menampilkan daftar *outlet* sesuai pada Gambar 16	☑ **Sesuai** Tidak Sesuai
2	Dst…5 skenario pengujian fungsional sistem. Jumlah skenario pengujian ini disesuaikan pada ruang lingkup sistem yang dirancang.			

Tabel 2. Hasil kuesioner pengujian *black box testing*.

No	Programmer	SP1		SP2		SP3		SP4		SP5		Hasil
		S	TS	S	TS	S	TS	S	TS	S	TS	
1	P1	1	0	1	0	1	0	1	0	1	0	S = 5, TS =0
2	P2	1	0	1	0	1	0	1	0	1	0	S = 5, TS =0
3	P3	1	0	1	0	1	0	1	0	1	0	S = 5, TS =0
HP1 = (Jumlah S / Jumlah SP) x 100 % = (5 /5) x 100 %												100%
HP2 = (Jumlah S / Jumlah SP) x 100 % = (5 /5) x 100 %												100%
HP3 = (Jumlah S / Jumlah SP) x 100 % = (5 /5) x 100 %												100%
Hasil Pengujian												100% Sesuai

Keterangan :

- SP : Skenario Pengujian
- S/TS : Sesuai / Tidak Sesuai
- P : *Programmer*
- HP : Hasil *Programmer*

4. Kesimpulan

Dari hasil penelitian, perancangan, dan implementasi yang telah dilakukan, dapat ditarik beberapa kesimpulan bahwa dihasilkan sebuah aplikasi *middleware* yang bekerja dengan baik sesuai fungsinya sebagai media sinkronisasi *database* yang dapat melakukan sinkronisasi data pada *database* terdistribusi yang *heterogenous* melalui komunikasi *socket.* Sinkronisasi ini berjalan secara otomatis setelah aplikasi server dan aplikasi *client* terhubung dan dapat menyinkronkan data antara DBMS *MsSQL Server* dan DBMS *MySQL Server*.

Penulis menyadari banyak kekurangan dalam penelitian ini sehingga perlu dilakukan pengembangan dan penyempurnaan lebih lanjut. Oleh karena itu penulis ingin memberikan saran yaitu sistem ini perlu dikembangkan dengan menambahkan fitur enkripsi pada data yang terkirim melalui komunikasi jaringan untuk tujuan keamanan data.

Daftar Pustaka

[1] V. A. Abrar and M. D. R. Wahyudi, "Implementasi Heterogenous Distributed Database System Oracle Xe 10g dan MySQL Rekam Medis Poliklinik UIN Sunan Kalijaga," *Citec J.*, vol. 4, no. 1, 2017.

[2] I. F. Anshori, "Implementasi Socket Tcp/Ip Untuk Mengirim Dan Memasukan File Text Kedalam Database," *J. Responsive*, vol. 1, no. 1, 2019.

[3] H. Listiyono, "Program Socket Untuk Mengirim file Dengan Visual Basic Pada Sistem Operasi WIndows," *J. Responsive*, vol. 1, no. 1, 2019.

[4] K. Agus, *"Mengenal Socket" in Pemrograman Jaringan Dengan Java*. Yogyakarta, Indonesia, 2011.

[5] S. Arifin, B. Antonius, and A. Abdul, "'Pembangunan Sistem Informasi Presensi Perkuliahan Menggunakan Basis Data Terdistribusi Dengan Metode Replikasi-Asynchronus,'" *Itsmart*, vol. 4, no. 2, 2015.

[6] D. A. Hidayat, "'Rancang Bangun Aplikasi Point of Sale (POS) Berbasis Web Dengan Pemanfaatan Trigger Pada Distribution Store CV.NMRQ,'" *J. Sist. dan Teknol. Inf.*, vol. 2, no. 2, 2014.

[7] M. Data, G. Ramadhan, and K. Amron, ", 'Analisis Availabilitas dan Reliabbilitas Multi-Master Database Server dengan State Snapshot transfers (SST) Jenis Rsync pada MariaDB Galera Cluster,'" *J. Teknol. Inf. dan Ilmu Komput.*, vol. 4, no. 1, 2017.

[8] P. Lawrence and M. Praveen, ""Introduction to JavaFX" in Beginning JavaFX," *Springer Sci. Media*, 2010.