W

;/--n

**TUGAS AKHIR – KI141502**

**SINKRONISASI BASIS DATA SQL DENGAN BASIS DATA NOSQL MENGGUNAKAN *DATA ADAPTER* DENGAN PENDEKATAN *QUERY* *DIRECT ACCESS***

**I GUSTI NGURAH ADI WICAKSANA**

**NRP 5113100110**

**Dosen Pembimbing I**

**Ir. Muchammad Husni, M.Kom.**

**Dosen Pembimbing II**

**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

**JURUSAN TEKNIK INFORMATIKA**

**Fakultas Teknologi Informasi**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2017**



**TUGAS AKHIR – KI141502**

**SINKRONISASI BASIS DATA SQL DENGAN BASIS DATA NOSQL MENGGUNAKAN *DATA ADAPTER* DENGAN PENDEKATAN *QUERY DIRECT ACCESS***

**I GUSTI NGURAH ADI WICAKSANA**

**NRP 5113100110**

**Dosen Pembimbing I**

**Ir. Muchammad Husni, M.Kom.**

**Dosen Pembimbing II**

**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

**JURUSAN TEKNIK INFORMATIKA**

**Fakultas Teknologi Informasi**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2017**

***[Halaman ini sengaja dikosongkan]***



**UNDERGRADUATE THESES – KI141502**

**SYNCRONIZATION BETWEEN SQL AND NOSQL DATABASES USING DATA ADAPTER WITH DIRECT ACCESS QUERY APPROACH**

**I GUSTI NGURAH ADI WICAKSANA**

**NRP 5113100110**

**Supervisor I**

**Ir. Muchammad Husni, M.Kom.**

**Supervisor II**

**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**

**DEPARTMENT OF INFORMATICS**

**FACULTY OF INFORMATION TECHNOLOGY**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**SURABAYA 2017**

***[Halaman ini sengaja dikosongkan]***

# LEMBAR PENGESAHAN

**SINKRONISASI BASIS DATA SQL DENGAN BASIS DATA NOSQL MENGGUNAKAN DATA ADAPTER DENGAN PENDEKATAN QUERY DIRECT ACCESS**

**TUGAS AKHIR**

Diajukan Untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Komputer

pada

Rumpun Mata Kuliah Komputasi Berbasis Jaringan

Program Studi S-1 Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Oleh

**I GUSTI NGURAH ADI WICAKSANA**

**NRP : 5113 100 110**

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Ir. Muchammad Husni, M.Kom .....................

NIP: 19600221 198403 1 001 (Pembimbing 1)

# Henning Titi Ciptaningtyas, S.Kom., M.Kom .....................

NIP: 19840708 201012 2 004 (Pembimbing 2)

**SURABAYA**

**MEI, 2017**

***[Halaman ini sengaja dikosongkan]***

**SINKRONISASI BASIS DATA SQL DENGAN BASIS DATA NOSQL MENGGUNAKAN DATA ADAPTER DENGAN PENDEKATAN QUERY DIRECT ACCESS**

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa** | **:** | **I Gusti Ngurah Adi Wicaksana** |
| **NRP** | **:** | **5113100110** |
| **Jurusan** | **:** | **Teknik Informatika FTIF-ITS** |
| **Dosen Pembimbing 1** | **:** | **Ir. Muchammad Husni, M.Kom.** |
| **Dosen Pembimbing 2** | **:** | **Henning Titi Ciptaningtyas, S.Kom., M.Kom.** |

# Abstrak

*Basis data NoSQL, singkatan dari Not SQL, semakin banyak digunakan seiring dengan bertambahnya jumlah aplikasi big data. Kebanyakan sistem masih menggunakan relational databases (RDB), namun seiring dengan bertambahknya jumlah data tiap tahunnya, sistem menangani big data dengan basis data NoSQL untuk menganalisis dan mengakses data dengan lebih cepat.*

*Data dari genom sendiri sangatlah besar sehingga dibutuhkan suatu teknik yang dapat mengurangi waktu komputasi. Hal tersebut dapat tercapai dengan cara mengimplementasikan komputasi paralel terhadap algoritma LCS. Oleh karena itu Tugas Akhir ini mengimplementasikan Algoritma LCS yang diparalelkan.*

*Tugas Akhir ini memiliki tujuan untuk mengimplementasikan mekanisme data adapter dalam sinkronisasi basis data RDB dan NoSQL. Berdasarkan hasil uji coba, waktu eksekusi LCS secara paralel terbukti lebih cepat daripada sekuensial dan panjang karakter yang dapat diproses pada komputasi paralel lebih banyak daripada sekuensial.*

***Kata kunci: RDB, NoSQL, Data Adapter, Sinkronisasi.***

**SYNCHRONIZATION BETWEEN SQL AND NOSQL DATABASE USING DATA ADAPTER WITH DIRECT ACCESS QUERY APPROACH**

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa** | **:** | **I Gusti Ngurah Adi Wicaksana** |
| **NRP** | **:** | **5113100110** |
| **Jurusan** | **:** | **Teknik Informatika FTIF-ITS** |
| **Dosen Pembimbing 1** | **:** | **Ir. Muchammad Husni, M.Kom.** |
| **Dosen Pembimbing 2** | **:** | **Henning Titi Ciptaningtyas, S.Kom., M.Kom.** |

# Abstract

*The process of finding the global alignment of two genomes by comparing the sequence is one of the problems frequently encountered in computational biology. One of the ways to solve this problem is to implement the algorithm Longest common subsequence (LCS) in finding the global alignment of both genomes.*

*Data from the genome itself is large so it takes a technique that can reduce the computation time. This can be achieved by implementing parallel computation of the LCS algorithm. Therefore, this undergraduate thesis implements the parallelization of LCS algorithm.*

*The purpose of this undergraduate thesis is to find the similarity of the two genomes and analyze and compare the performance of the sequential of LCS algorithms and the parallelization of LCS algorithm. LCS algorithm implemented in parallel using OpenMPI library. Based on trial results, the execution time LCS in parallel proved faster than sequential and length of characters that can be processed in parallel computing is longer than sequential.*

***Keywords: LCS, OpenMPI, Dynamic Programming, Genom.***

***[Halaman ini sengaja dikosongkan]***

# KATA PENGANTAR

Alhamdulillahirabbil’alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***“*RANCANG BANGUN KOMPUTASI PARALEL PADA ALGORITMA *LONGEST COMMON SUBSEQUENCE* UNTUK PERBANDINGAN GENOM MENGGUNAKAN *FRAMEWORK* OPENMPI*”*.** 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. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga di Pontianak khususnya Mama, Bapak, dan Ndut yang telah memberikan dukungan moral dan material serta do’a yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
3. Keluarga di Surabaya, Tante Vitri, Om Dodo, Mba Dita, Mas Dimas, Ivan dan Iyuk yang telah memberikan kehangatan keluarga serta support untuk penulis selama kuliah di ITS.
4. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
5. Bapak Hudan Studiawan, S.Kom., M.Kom**.** selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi kepada penulis dalam mengerjakan Tugas Akhir ini.
6. Bapak Dr. Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS.
7. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
8. Teman-teman semangat TA Altea, Angga, Bimo, Dian, Fatih, Fifi, Lubna, Luqman, Oshi, Pinas, dan Puni.
9. Teman-teman administrator laboratorium Arsitektur dan Jaringan Komputer(AJK), Mas Sam, Mas Uyung, Mbak Vivi, Kak Harum, Mas Romen, Mas Dimas, Agus, Pur, Thiar, Uul, Wicak, Zaza, Risma, Nindy, Asbun, Daniel, Fatih, Oink, dan Syukron.
10. Teman-teman angkatan 2012 yang yang telah berbagi ilmu, dan memberi motivasi kepada penulis.
11. 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 2016

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

# DAFTAR KODE SUMBER

***[Halaman ini sengaja dikosongkan]***

# DAFTAR PSEUDOCODE

***[Halaman ini sengaja dikosongkan]***

# BAB I PENDAHULUAN

## Latar Belakang

*Big data* dan *hybrid database* menjadi semakin pupoler seiring dengan berkembangnya servis *cloud computing*. The United Nations Economic Commission for Europe, atau UNECE, memprediksi jika peningkatan jumlah data akan mencapai 350% pada tahun 2019 jika dibandinkan dengan tahun 2015. [1] Basis data NoSQL, singkatan dari *Not Only SQL*, semakin banyak digunakan seiring dengan bertambahnya jumlah aplikasi *big data*.

Kebanyakan sistem dewasa ini masih menggunakan *relational database* (RDB). MySQL adalah salah satu *Database Management System* (DBMS) populer untuk aplikasi berbasis web. Dengan performanya yang terpercaya, keandalan, dan penggunaan yang mudah, MySQL telah menjadi basis data pilihan utama yang digunakan oleh perusahaan dengan profil tinggi seperti Facebook, Twitter, dan Youtube. [2] Tetapi seiring dengan jumlah data yang terus bertambah tiap tahunnya, sistem menangani *big data* dengan basis data NoSQL untuk menganalisis dan mengakses data dengan lebih cepat. NoSQL muncul sebagai akibat dari pertumbuhan eksponensial dari internet dan perkembangan aplikasi web. NoSQL mendukung pengembangan yang gesit, karena NoSQL adalah *schema-less* dan tidak tidak perlu mendefinisikan secara statik bagaimana data harus dimodelkan. [3]

Sistem yang menggunakan basis data NoSQL, ketika mengkueri data tidak menggunakan sintaks yang sama seperti mengambil data pada basisdata SQL karena sintaks *query* yang digunakan berbeda. Dibandingkan merubah kode sumber atau merubah basis data RBD menjadi NoSQL, penulis melakukan riset untuk mengintegrasikan kedua basis data. Aplikasi terkoneksi dengan RDB untuk menangani jumlah data dengan skala kecil dan menengah, server basis data NoSQL sebagai *back-end* sistem untuk menganalisis data dan melakukan sejumlah operasi *read/write*, atau secara periodik melakukan *backup* data dari RDB.

Integrasi basis data dapat mempengaruhi desain sistem yang asli. Dalam sistem, aplikasi memperoleh data dari basis data relasional menggunakan kueri SQL, namun dalam NoSQL tidak dapat diakses dengan menggunakan SQL. Oleh karena itu aplikasi harus merubah desain untuk dapat mengakses basis data RBD dan NoSQL. Mekanisme transformasi data dari RDB ke NoSQL diperlukan ketika mengintegrasikan sistem dengan NoSQL. Selama proses transformasi, aplikasi dituntut untuk menunggu hingga proses sinkronisasi selesai. Proses transformasi ini dapat berlangsung sangat lama karena jumlah data yang diubah dalam skala besar. Hal ini menjadi permasalahan utama untuk beberapa kasus seperti *real-time* atau layanan *non-stop* seperti analisis ilmiah dan aplikasi web.

## Rumusan Masalah

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

1. Bagaimana mengimplementasikan data adapter untuk mensinkronisasikan RDB dan NoSQL?
2. Bagaimana hasil implementasi data adapter dalam sinkronisasi basis data RDB dan NoSQL?

## Batasan Masalah

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

* 1. Banyak node server untuk basis data yang digunakan adalah berjumlah dua komputer.

1. Mekanisme yang digunakan adalah menggunakan konsep data adapter
2. Basis data yang digunakan adalah MySQL sebagai basis data RDB dan Apache HBase sebagai basis data NoSQL.

## Tujuan

Tujuan Tugas Akhir ini adalah mengimplementasikan mekanisme data adapter dalam sinkronisasi basis data RDB dan NoSQL.

## Manfaat

Dengan dibuatnya Tugas Akhir ini yaitu dapat mensinkronisasikan basis data RDB dengan NoSQL dengan mekanisme data adapter.

## 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.

1. Studi literatur

Pada tahap ini dilakukan pencarian informasi dan literatur dari paper dan artikel di internet yang diperlukan untuk tahap implementasi program. Tahap ini diperlukan untuk membantu memahami konsep dan penggunaan komponen-komponen terkait dengan sistem yang akan dibangun, diantaranya data adapter, basis data MySQL dan Apache Hbase.

1. 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*.

1. 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.

1. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian dan pencatatan hasil dari metode yang diimplementasikan. Pengujian akan dilakukan di Laboratorium Arsitektur dan Jaringan Komputer. Parameter yang akan diujikan adalah berapa waktu yang dibutuhkan untuk melakukan sinkronisasi basis data antara RDB dan HBase NoSQL database sehingga dapat diketahui efektifitas implementasi sistem data adapter. Banyak komputer server basis data yang digunakan adalah dua komputer.

1. Penyusunan buku Tugas Akhir

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

## 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.

## Big Data

Setiap hari, manusia menghasilkan 2,5 triliun *byte* data, dan 90% dari semua data yang ada di dunia saat ini muncul pada dua tahun terakhir. Data ini datang dari mana-mana: sensor yang digunakan untuk menangkap informasi perubahan cuaca, sejumlah post di media sosial, gambar digital dan video, catatan transaksi penjualan, dan dan data sinyal GPS dari perangkat bergerak. Jumlah yang sangat besar dan diproduksi secara terus-menerus ini dikenal dengan sebutan *Big Data*. Sejalan dengan peningkatan jumlah data ini, teknik yang ada sekarangpun menjadi semakin kuno. Perlu adanya keterampilan membuat kode yang komprehensif serta penguasaan pengetahuan dan statistik yang baik. [4]

Istilah “*Big Data*” muncul seiring dengan berkembangnya proses industri, personil dan teknologi untuk mendorong bidang baru apa yang tampaknya akan meledak. Perusahaan besar seperti Amazon dan Wal-Mart serta badan lain seperti pemerintahan Amerika Serikat dan NASA menggunakan *big data* untuk memenuhi tujuan bisnis dan strategi mereka. *Big data* dapat memainkan peran bagi perusahaan skala menengah keatas dan organisasi yang dimanfaatkan agar mendapatkan keuntungan.

Tidak ada hal yang baru terkait dengan gagasan *big data*, yang mana telah ada sejak tahun 2001. *Big data* adalah informasi yang dimiliki perusahaan, didapatkan dan diproses dengan teknik baru untuk menghasilkan nilai dengan cara terbaik. [5]

Pada tahun 2001, Gartner’s Doug Laney pertama kalinya memperkenalkan istilah yang dikenal dengan “Tiga V” untuk menggambarkan beberapa karakteristik yang membuat *big data* berbeda dengan pemrosesan data yang lain. ‘Tiga’ V itu ialah *Volume*, *Velocity*, dan *Variety*. [6]

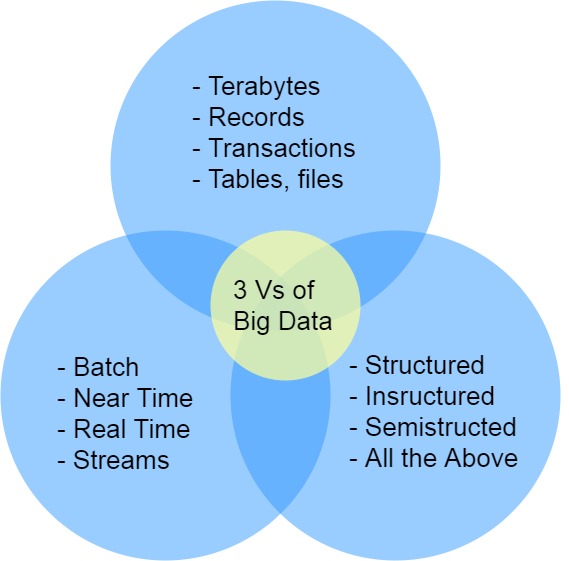
*Volume*, atau kapasitas, mengacu pada jumlah data yang tersedia untuk dianalisis. Namun ketika data terkumpulkan dari pertumbuhan jumlah perangkat, seperti sensor, telepon seluler dan komputer, jumlah data yang terakumulasikan menjadi sangat banyak. Namun sudah ada beberapa teknologi, seperti Hadoop, yang memudahkan beban ini.

*Velocity*, atau kecepatan, mengacu pada kecepatan pengolahan data. Karena data datang dari berbagai sumber, hal itu bisa terkumpulkan dengan cepat. Tantangannya adalah mengumpulkannya secepat dan seefisien mungkin.

*Variety*, atau keberagaman, mengacu pada banyaknya tipe format data. Contohnya seperti tipe data terstruktur, numerik, dokumen, suara, video, *e-mail* dan data transaksi penjualan. Tantangannya adalah membandingkan dan membedakan banyak data sedemikian rupa sehingga menjadi berpola dan dapat dimanfaatkan. [7]

Hal terpenting dari big data adalah potensi untuk meningkatkan efisiensi dalam konteks volume data yang sangat besar, dari tipe data yang berbeda. Jika big data digunakan dengan tepat, suatu instansi dapat memiliki bayangan untuk bisnis mereka, oleh karena itu akan mengarah pada efisiensi di berbagai bidang seperti penjualan, peningkatan produk manufaktur dan sebagainya. *Big data* dapat diterapkan secara efektif pada beberapa area, contohnya adalah sebagai berikut:

* Dalam teknologi informasi digunakan untuk meningkatkan keamanan dan penanganan masalah dengan menganalisis pola dari log yang ada.
* Meningkatkan servis dan produk berdasarkan penggunaan konten media sosial. Dengan mengetahui preferensi pelanggan, perusahaan dapat merubah produknya agar dapat menyebar ke orang yang lebih banyak. [8]



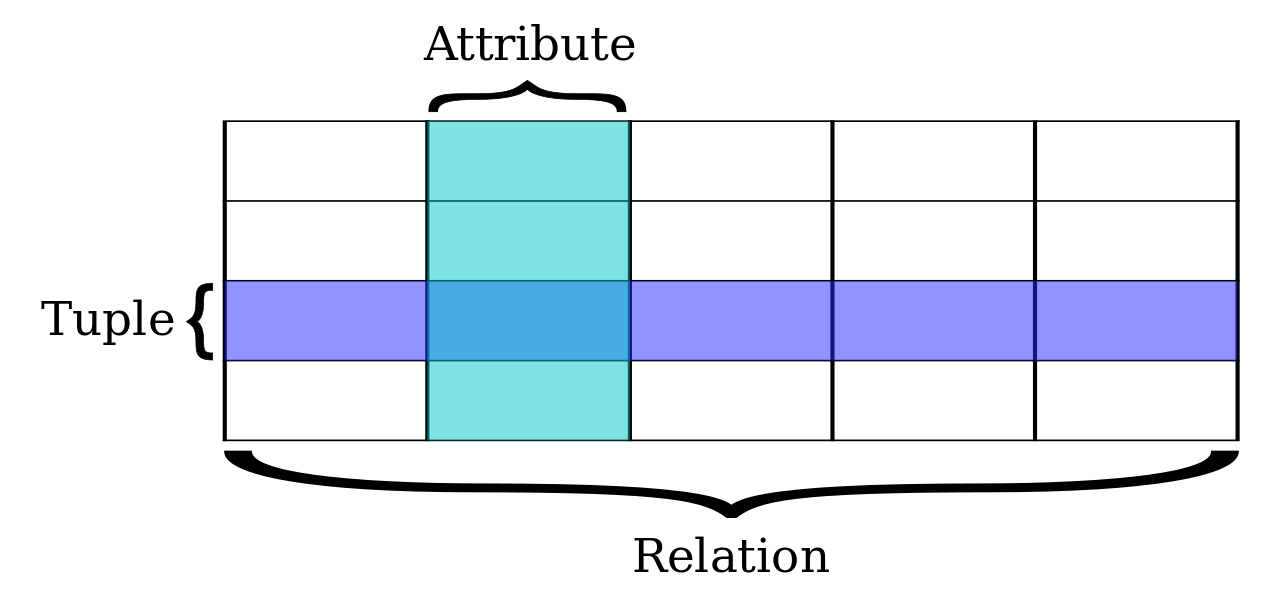
Gambar 2.1 Diagram Tiga ‘V’ Big Data

## 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

Bentuk sebuah basis data, elektronik dan lainnya, harus di rencanakan. Proses dalam merancang basis data adalah aktivitas dalam merepresentasikan kelas, atribut, dan relasi antar basis data. Data adalah fakta. Informasi menggambarkan data. Informasi adalah data dengan konteks yang berarti. [9]

## Basis Data Relasional



Gambar 2.2 Terminologi Basis Data Relasional

*Relational database*, atau disingkat RDB, menggambarkan suatu kumpulan dari banyak relasi. Konsep basis data relasional, bahasa indonesia dari *relational database,* pertama kali diperkenalkan oleh Dr. Codd pada tahun 1970. Sebuah sistem yang mengatur hal ini disebut dengan *Relational Database Management System* (RDBMS). Model relasional terdiri dari beberapa komponen, diantaranya kumpulan objek berelasi, kumpulan operasi yang bekerja pada relasi, dan integritas data untuk.

Sebagian besar *database* yang digunakan pada aplikasi modern saat ini adalah relasional *database*. Basis data relasional adalah model *database* yang menyimpan data pada tabel. Setiap tabel terdiri dari baris (*record*) dan kolom (*field*). Dalam terminologi ilmu komputer, baris sering disebut dengan “*tuples*”, dan kolom dapat disebut dengan “*attribute*”. Sebuah tabel dapat divisualisasikan sebagai sebuah matriks baris dan kolom, dimana setiap persimpangan dari baris dan kolom berisi nilai tertentu. Hal ini adalah relasional selama semua *record* berbagi bidang yang sama. Gambar 2.2 menunjukan terminologi basis data relasional.

Tabel juga sering terdapat *primary key*, yang menyediakan tanda pengenal unik untuk setiap baris dalam tabel. Kunci ini bisa ditunjuk ke kolom, atau dapat terdiri dari beberapa kolom yang bersama-sama membentuk kombinasi unik dari beberapa nilai. Disisi lain, *primary key* menyediakan cara yang efisien untuk pengindeksan dan dapat digunakan untuk berbagi nilai antar tabel dalam database. Sebagai contoh, nilai *primary key* dari satu tabel dapat digunakan pada baris pada tabel lainnya. Nilai yang disisipkan ini tabel lain ini disebut dengan *foreign key*

Cara yang digunakan untuk mengakses data pada basis data relasional adalah dengan menggunakan kueri SQL (*Structured Query Language).* Kueri SQL dapat digunakan untuk membuat, memodifikasi, dan menghapus tabel, serta memilih, *insert*, dan menghapus data dari tabel yang ada. [10]

## Basis Data NoSQL

Basis data NoSQL, kependekan dari *Not Only SQL*, adalah sebuah pendekatan untuk manajemen data dan desain *database* yang berguna untuk kumpulan distribusi data yang sangat besar. NoSQL, meliputi berbagai teknologi dan arsitektur, berusaha untuk memecahkan masalah skalabilitas dan permasalahan performa big data pada kinerja database relasional. NoSQL sangat berguna ketika suatu perusahaan perlu untuk mengakses dan menganalisis data yang tidak terstruktur dalam jumlah yang sangat besar atau data yang disimpan dari jarak jauh pada beberapa server virtual. [11]

NoSQL mencakup berbagai teknologi basis data yang berbeda dan dikembangkan untuk menangani permintaan dalam membangan aplikasi modern:

* Pengembang bekerja dengan aplikasi yang membuat volume baru yang sangat besar, dengan cepat melakukan perubahan tipe data (terstruktur, tidak terstruktur, semi terstruktur dan polimorfik)
* Lama pengerjaan siklus pengembangan *waterfall* adalah dua sampai delapan belas bulan. Dijaman sekarang, tim bekerja dengan secepat-cepatnya, mengiterasi dan mem-*push* kode hampir setiap minggu bahkan setiap hari.
* Aplikasi yang pernah dibuat untuk khalayak tertentu, kini dibuat menjadi servis yang harus selalu menyala, dapat diakses dari berbagai perangkat dalam skala global oleh jutaan pengguna.
* Perusahaan kini beralih ke *scale-out architecture* menggunakan perangkat lunak *open source*, server komoditas dan komputasi awan ketimbang monolotik server dan infrastruktur penyimpanan.

Basis data relasional tidak dirancang untuk mengatasi permasalahan dengan tantangan skala dan kegesitan yang dihadapi aplikasi modern. Basis data relasional juga tidak dibangun untuk mendapatkan keuntungan dari penyimpanan komoditas dan kemampuan pemrosesan yang tersedia saat ini. [12]

Basis data NoSQL adalah teknologi penyimpanan data yang lebih bervariasi sehingga sulit untuk membuat karakteristik mereka menjadi general. NoSQL memberikan performa dan skalabilitas yang lebih baik jika dibandingkan dengan basis data relasional. [13] Ada beberapa tipe basis data NoSQL, yaitu:

* 1. *Document*

*Document* dikenal dengan memasangkan tiap key-nya dengan struktur data yang kompleks. *Document* dapat terdiri dari banyak pasangan *key-value* yang berbeda, atau pasangan *key-array*, atau bahkan dokumen bersarang (*nested document*)

* 1. *Graph*

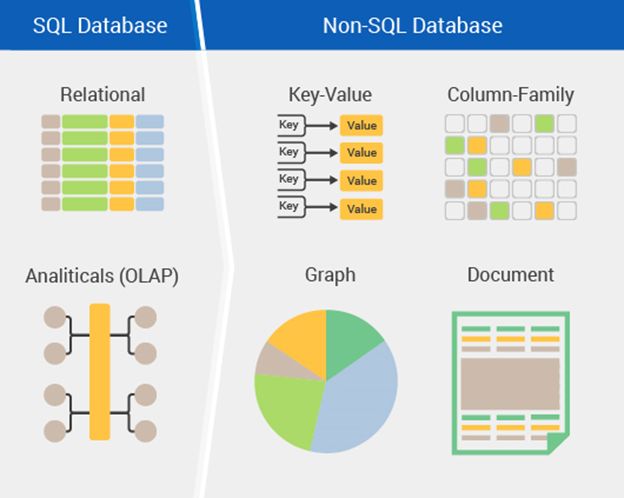
Digunakan untuk menyimpan informasi tentang jaringan seperti koneksi sosial (social connection). Yang termasuk graph diantaranya Neo4J dan HyperGraphDB.

* 1. *Key-value*

*Key-value* adalah basis data NoSQL paling sederhana. Setiap item didalam basis data disimpan dengan atribut nama (atau ‘kunci’), bersama dengan nilainya. Contoh dari key-value adalah Riak dan Voldemort. Beberapa penyimpanan key-value, seperti Redis, memungkinkan setiap nilai memiliki tipe data, seperti integer, yang dapat menambahkan fungsionalitas.

* 1. *Wide-column*

Tipe *wide-column*, seperti Cassandra dan HBase, adalah mengoptimasi kueri dengan kumpulan data yang besar, dan menyimpan data kolum secara bersama, termasuk barisnya.



**Document**

**Column-wide**

**Graph**

**Key-value**

Gambar 2.3 Tipe Basis Data NoSQL

## Data Adapater

Data adapter sangat termodulasi, yang mana terletak diantara aplikasi dengan basis data. Data adapter bertanggung jawab sebagai untuk melakukan query dari aplikasi dan transformasi data antar basis data dalam waktu. Sistem menyajikan antarmuka parsing kueri SQL untuk mengakses RDB dan NoSQL.

Data adapter menawarkan mekanisme untuk mengontrol proses transformasi basis data dan memperbolehkan aplikasi untuk melakukan kueri meskipun data (tabel) yang dimaksud sedang transformasi atau tidak. Setelah data ditransformasi, data adapter menyediakan mekanisme penambalan untuk mensinkronisasikan ketidak konsistenan data pada tabel. Data adapter menawarkan *non-stoping service* selama proses transformasi berlangsung.

Terdapat tiga fitur utama pada data adapter. Yang pertama adalah data adapter menyediakan antarmuka SQL untuk basis data RDB dan NoSQL. Di dalamnya terdiri atas Apache Phoenix sebagai penerjemah SQL untuk terhubung dengan HBase, dan MySQL JDBC driver untuk menghubungkan dengan RDB yaitu MySQL. Dengan ini aplikasi tidak perlu mengubah kueri untuk menangani kueri NoSQL.

Berikutnya adalah *Database Converter* yang digunakan untuk menangani transformasi basis data. *Database Converter* merubah data dari MySQL ke HBase menggunakan Apache Sqoop dan Apache Phoenix. Mekanisme ini mensinkronisasi data setelah transformasi selesai dilakukan dengan penambalan blok kueri.

Fitur terakhir adalah *Query Approach* atau pendekatan kueri. Pendekatan kueri yang digunakan adalah *direct access (DA mode)*. Pendekatan ini memperbolehkan aplikasi untuk melakukan kueri secara langsung ke basis data tanpa harus menunggu proses transformasi selesai dilakukan atau mengambil data pada server lain. [14]

Pada pengerjaan Tugas Akhir ini, data adapter menjadi komponen utama untuk melakukan sinkronisasi antara basis data SQL dan NoSQL. Tipe transformasi yang digunakan adalah dimana pada kedua basis data, SQL dan NoSQL, memiliki salinan data yang sama, dan arah transformasi adalah dari basis data SQL ke NoSQL.

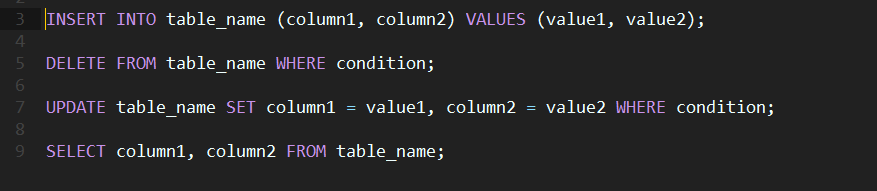
## MySQL

MySQL adalah aplikasi manajemen basis data, terjemahan dari *Database Management System* (DBMS), *open-source* yang paling terkenal di seluruh dunia. MySQL termasuk kedalam basis data relasional. Dengan performa yang terbukti, kehandalan dan penggunaanya yang mudah, membuat MySQL menjadi basis data dengan pilihan terdepan untuk aplikasi bebrbasis web. MySQL telah digunakan oleh beberapa situs bersar dunia seperti Facebook, Twitter, Youtube, Yahoo dan masih banyak lagi. Oracle menjadi penggerak inovasi MySQL, menyajikan kapabilitas baru untuk memperkuat web, *cloud*, *mobile* dan *embedded application* dimasa mendatang. [15]

Sebagai basis data yang relasional, MySQL menyimpan data dalam tabel yang terpisah dibandingkan dengan menyimpannya kedalam satu tabel. Model logika, dengan objek seperti basis data, tabel, *view*, baris dan kolom, menawarkan lingkungan pemrograman yang fleksibel. Basis data relasional memiliki pengaturan dasar diantara antara tabel yang berbeda seperti *one-to-one*, *one-to-many*, unik, diperlukan atau opsional, dan ‘pointer’ di antara tabel yang berbeda. Basis data dengan pengaturan seperti ini dirancang agar aplikasi tidak mengakses data yang tidak konsisten, duplikat, orphan, *out-of-date*, atau data yang hilang. [15]

MySQL memiliki beberapa terminologi dasar. MySQL membuat, mengkonfigurasikan dan berkomunikasi dengan basis data. Sebuah basis data adalah kumpulan data yang teratur. Tabel terdiri dari beberapa *records* (biasa disebut dengan baris), dan record mengandung *field* (biasa disebut dengan kolom). [16] Untuk mengakses dan mengubah data yang tersimpan, MySQL menggunakan sintaks *Structured Query Language* (SQL) sebagai standar bahasanya. Selain MySQL, ada beberapa aplikasi DBMS yang menggunakan sintaks SQL sebagai bahasa standar untuk memanipulasi data didalamnya, seperti SQL Server, MS Access, Sybase, Informix, dan Postgres. Beberapa sintaks sederhana dapat dilihat pada Gambar 2.1. Tabel 2. 1 adalah contoh gambaran sederhana bentuk tabel pada basis data MySQL.

Dalam pengerjaan Tugas Akhir ini, MySQL berperan sebagai basis data relasional. Basis data MySQL menyimpan data asli sistem, dimana pada percobaan penelitian ini, basis data yang ditambahkan menggunakan basis data NoSQL yaitu Apache Hbase. Kedua basis data dinsinkronisasi menggunakan data adapter.



Gambar 2.4 Contoh Sintaks SQL

Tabel 2. 1. Contoh tabel pada basis data MySQL

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Field 1* | *Field 2* | *Field 3* |
| *Field Names:* | **Nama** | **Umur** | **Warna Favorit** |
| *Record 1* | Bruce Callow | 13 | Tidak punya |
| *Record 2* | Frank Wright | 37 | Merah |
| *Record 3* | Seymour Hawthorne | 82 | Hitam, Putih |

## Apache HBase

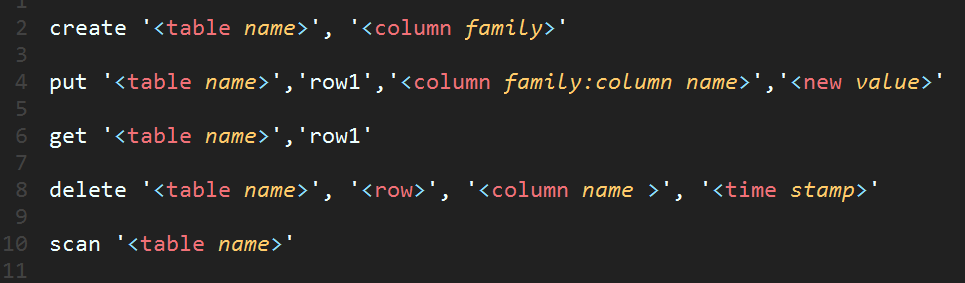
Apache Hbase dibuat pada tahun 2007 di Powerset dan awalnya merupakan bagian dari Hadoop. Sejak saat itu, proyek ini menjadi proyek tingkat atas dibawah Apache Software Foundation. HBase tersedia dibawah lisensi Apache Software License, versi 2.0. [17]

Apache HBase adalah salah satu basis data NoSQL yang *open source* dimana menyajikan akses *real-time read/write* untuk sebuah basis data yang besar. HBase berjalan dalam sebauh file sistem yang bernama Hadoop. Skala penggunaan HBase adalah untuk menangani data set yang sangat besar dengan berjuta-juta baris dan kolom, dan basis data ini dengan mudah mengkombinasikan sumber data yang menggunakan struktur dan skema dengan variasi yang berbeda. HBase terintegrasi dengan Hadoop dan disamping itu bekerja dengan baik pada *data* *engine* YARN.

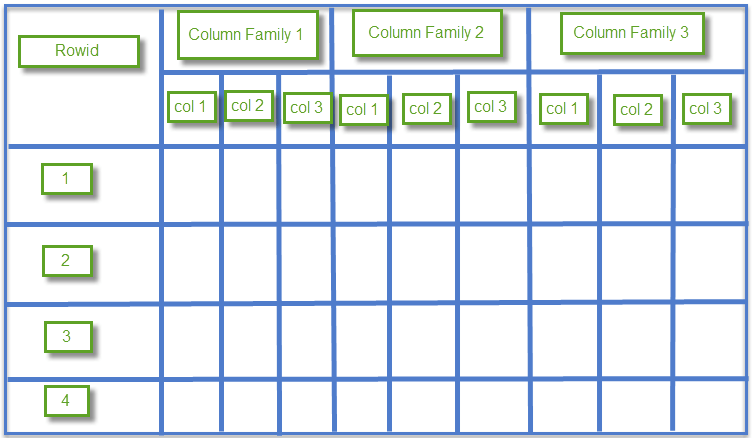
Apache HBase memberikan beberapa fitur seperti secara acak dan *real-time* mengakses data pada Hadoop. HBase dibuat untuk menampung tabel yang berukuran sangat besar, menjadikannya pilihan yang tepat untuk menyimpan *multi-structure* atau data yang jarang. Pengguna dapat melakukan kueri HBase dalam waktu tertentu, untuk melakukan “*flashback*” *query*. Karakteristik ini membuat HBase menjadi pilihan yang tepat untuk menyimpan data *semi-structured* seperti data log dan memberikan data yang sangat cepat untuk pengguna atau aplikasi yag terintegrasi dengan HBase. [18]

HBase adalah basis data yang berorientasi kolum dan data disimpan dalam tabel. Tabel diurutkan berdasarkakn *row id*. HBase memiliki *row id* yang merupakan kumpulan dari beberapa *column family* yang ditampilkan dalam bentuk tabel. *Column family* yang ditampilkan dalam skema adalah pasangan *key-value*. Jika dilihat secara rinci, setiap *column family* memiliki beberapa kolom. Nilai kolom ini disimpan dalam memori disk. Setiap sel dalam tabel memiliki data meta sendiri, seperti stempel waktu dan data lainnya. [19] Skema penyimpan di Hbase dapat dilihat pada Gambar 2.5. HBase memiliki perintah kueri yang berbeda dengan MySQL. Perintah kueri ini dapat dijalankan didalam HBase Shell. Gambar 2.5 menunjukan bentuk dasar perintah pada HBase. [20]

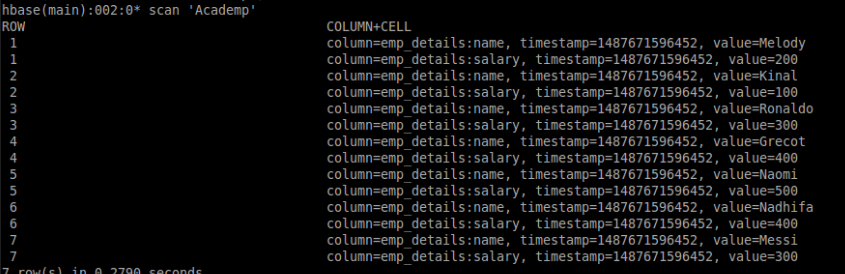
Pada pengerjaan tugas akhir ini, Apache HBase digunakan sebagai basis data NoSQL. Apache HBase disinkronisasikan dengan basis data SQL, dalam tugas akhir ini menggunakan MySQL, melalui data adapter. Contoh bentuk data yang disimpan pada Apache HBase jika dilihat melalui terminal ditunjukan pada Gambar 2.6.



Gambar 2.5 Perintah Dasar pada HBase Shell



Gambar 2.6 Skema penyimpanan pada Apache Hbase



Gambar 2.7 Contoh tampilan data yang disimpan dalam Hbase jika diakses melalui terminal

## Apache Phoenix

Banyak perkakas kecerdasan bisnis dan perkakas analisis data memiliki kemampuan yang kurang untuk bekerja dengan basis data HBase secara langsung. Apache Phoenix dapat membuat Anda untuk berinteraksi dengan HBase menggunakan SQL. Dengan menggunakan ODBC *drivers*, Anda dapat menghubungkan aplikasi ODBC dengan HBase melalui Apache Phoenix. [21] Beberapa perusahaan besar telah menggunakan Apache Phoenix seperti salesforce.com, Bloomberg, Sogou, Hortonworks, Alibaba.com, CertusNet, ebay.com, NGDATA, eHarmony dan PubMatic. [22]

Apache Phoenix adalah lapisan SQL yang efisien untuk Apache HBase. Apache Phoenix menambahkan SQL ke HBase, aplikasi *big data* yang terdistribusi dan *scalable* dan berjalan di atas file sistem Hadoop. Phoenix, bertujuan untuk mempermudah mengakses basis data HBase dengan mendukung sintaks SQL dan memperbolehkan *input* dan *output* dengan menggunakan JDBC API standar dibandingkan dengan menggunakan *HBase’s Java client APIs*. Hal ini membuat kita dapat melakukan operasi CRUD dan DDL seperti membuat tabel, memasukan data baru, dan kueri data. SQL dan JDBC mengurangi jumlah kode yang harus dituliskan oleh pengguna, dapat melakukan optimasi performa yang transparan untuk pengguna, dan membukakan perkakas lain yang ada untuk menggunakan dan mengintegrasikannya.

Lebih mendalam, Phoenix mengambil kueri SQL, meng-*compile*-nya menjadi rangkaian pemanggilan API HBase, dan mendorongnya kedalam kluster untuk eksekusi pararel. Phoenix secara otomatis membuat *metadata* yang menyediakan tipe akses untuk data yang disimpan ke tabel HBase. Phoenix secara langsung menggunakan HBase API, bersama dengan *coprocessor* dan penyaringan secara *custom*, yang menghasilkan performa seper seribu detik untuk kueri data kecil, atau beberapa detik untuk 10 juta baris. [23]

Apache Phoenix menyediakan bahasa kueri SQL yang sudah dikenal dan jauh lebih mudah dibandingkan dengan bahasa shell HBase atau HBase Java API. Manfaat utamanya adalah menjalankan semua kueri secara pararel si semua server wilayah. Akibatnya, HBase bekerja lebih cepat dan lebih efisien. Lapisan ini kompatibel dengan driver JDBC sehingga migrasi dari sistem basis data relasional biasa tidak memerlukan perubahan kode utama. Phoenix memerlukan beberapa perubahan sintaks, beberapa sintaks yang mengalami perubahan dapat dilihat pada Tabel 2.1.

Dalam pengerjaan tugas akhir ini, Apache Phoenix digunakan pada data adapter sebagai data *converter* untuk merubah data dari MySQL ke HBase dengan menjalankan sintaks kueri MySQL. Selain itu, Apache Phoenix juga digunakan oleh aplikasi untuk mengambil data dari HBase melalui data adapter.

Tabel 2.1 Beberapa Perbedaan Sintaks Antara Apache Phoenix dengan MySQL

|  |  |
| --- | --- |
| **Phoenix** | **MySQL** |
| **UPSERT** INTO t (col) VALUES (val); | **INSERT** INTO t (col) VALUES (val); |
| **UPSERT** INTO t (PK, col1) **SELECT** PK, ‘val1’ FROM t WHERE col2 = ‘val2’; | **UPDATE** t SET col1 = ‘val1’ WHERE col2 = ‘val2’; |

## Python

Python dikembangkan oleh Guido van Rossum di akhir tahun 80-an di *National Research Institute for Mathematics and Computer Science Netherlands*. Python berasal dari beberapa bahasa pemrogramman lain seperti ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix Shell dan bahasa pemrograman lainnya. Seperti Perl, Python berada dibawah GNU *General Public License* (GPL). [24]

Python adalah salah satu bahasa pemrograman tingkat tinggi yang *interpreted* dan *object-oriented* dengan semantik yang dinamis. Dibuat di dalam struktur data, dikombinasikan dengan tipe dan ikatan yang dinamis, membuatnya menjadi atraktif untuk mengembangkan aplikasi secara sepat. Bahasa python adalah bahasa pemrograman yang sederhana, mudah untuk dipelajari dan karena itu dapat mengurangi biaya perawatan. Python mendukung penggunaan modul dan paket, yang mendorong modularitas dan penggunaan kode kembali. Python *intepreter* dan *library* standar yang tersedia dapat digunakan secara gratis dan dapat didistribusikan dengan bebas. Banyak programmer yang suka dengan bahasa Python karena dapat meningkatkan produktivitas seperti tidak perlu adanya langkah kompilasi, dan siklus edit-tes-debug menjadi sangat cepat. [25]

## Flask

Flask adalah sebuah kerangka kerja berbasis Python yang dipelopori oleh Armin Ronacer. Flask berada dibawah lisensi BSD. Flask menyediakan alat, beberapa library, dan teknologi yang membantu dalam membuat aplikasi web. Aplikasi web ini bisa terdiri dari beberapa halaman, blog, wiki atau e-commerce.

Flask termasuk kedalam kategori micro-framework. Micro-framework adalah kerangka kerja biasa dengan sedikit atau tanpa ketergantungan dengan libraries eksternal. Hal ini menimbulkan pro dan kontra. Kelebihannya adalah framework menjadi ringan, memiliki ketergantungan yang sedikit untuk mengupdate dan mengamati bug keamanan. [26]

Untuk dapat menggunakan Flask, yang perlu dilakukan adalah menginstallnya dengan menggunakan pip. Pip akan secara otomatis melakukan pemasangan berdasarkan versi Python yang ada di komputer. Secara sederhana, contoh penggunaan Flask dapat dilihat pada Gambar 2.3. [27]

Pada pengerjaan tugas akhir ini, Flask dipasang di sistem data adapter pada bagian DB Adapter. DB Adapter berfungsi untuk melakukan komunikasi dengan aplikasi dengan menerima beberapa permintaan seperti perubahan data, pengambilan data dan memasukan data baru. Flask menghasilkan antar muka dengan bentuk data JSON yang digunakan oleh aplikas untuk mengakses data di basis data. Selain itu, pada Flask juga dipasang pada DB Converter sebagai antar muka untuk berkomunikasi dengan DB Adapter.



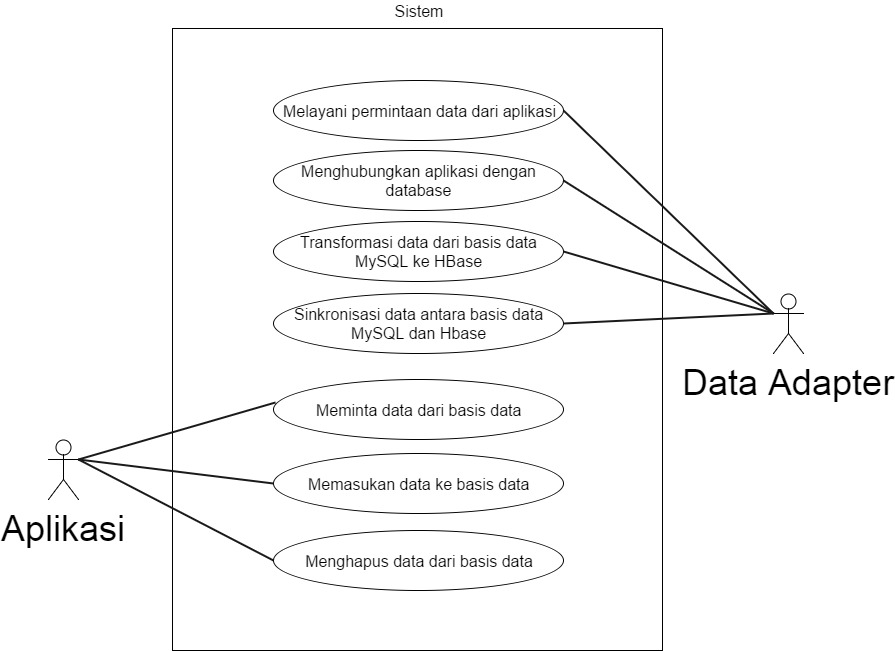
Gambar 2.8 Contoh Program Sederhana Menggunakan Flask

# BAB III ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

Pada bab ini akan dijelasakan mengenai analisis dan perancangan sistem. Perancangan di bagi menjadi perancangan arsitektur sistem dan perancangan proses utama sistem menggunakan data adapter.

## Kasus Pengguna

Terdapat dua aktor dalam diagram kasus pengguna yaitu aplikasi dan data adapter. Pada sistem, aplikasi memiliki tiga aktifitas dan data adapter memiliki 4 aktifitas yang di gambarkan pada Gambar 3.1.



Gambar 3.1 Diagram Kasus Pengguna

Diagram kasus pengguna pada Gambar 3.1 dijelaskan secara rinci pada tabel Tabel 3.1.

Tabel 3.1 Penjelasan Diagram Kasus Pengguna

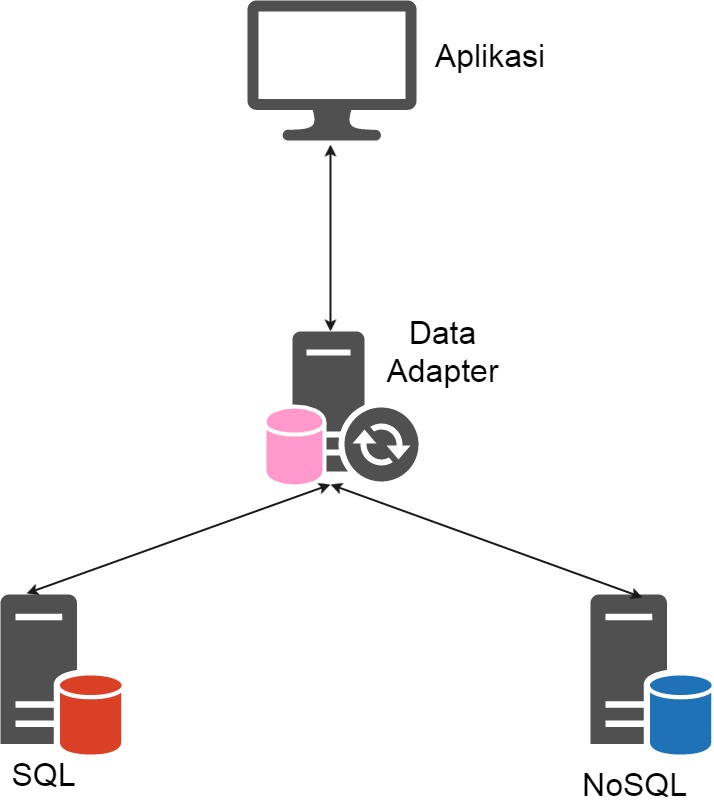
| **Kode** | **Nama Kasus Penggunaan** | **Aktor** | **Deskripsi** |
| --- | --- | --- | --- |
| UC01 | Melayani permintaan dari aplikasi | Data Adapter | Melayani permintaan data dari aplikasi, kemudian data adapter akan meneruskannya ke basis data |
| UC02 | Menghubungkan aplikasi dengan basis data | Data Adapter | Data adapter meneruskan permintaan aplikasi ke basis data. Data adapter menentukan dari basis data mana permintaan akan dijalankan. |
| UC03 | Mentransformasi data dari basis data MySQL ke HBase | Data Adapter | Data adapter bertugas melakukan transformasi data dari MySQL ke HBase. |
| UC04 | Melakukan sinkronisasi data antara basis data MySQL dan HBase | Data Adapter | Ketika aplikasi dijalankan oleh pengguna, data adapter akan melakukan proses sinkronisasi berdasarkan perubahan yang terjadi di basis data MySQL. |
| UC05 | Meminta data ke basis data | Aplikasi | Pengguna melalui aplikasi meminta data ke basis data. |
| UC06 | Memasukan data ke basis data | Aplikasi | Pengguna melalui aplikasi memasukan data baru ke basis data. |
| UC07 | Menghapus data dari basis data | Aplikasi | Pengguna melalui aplikasi menghapus data pada basis data. |

## Arsitektur Sistem

Sub-bab ini akan membahas mengenai analisis kebutuhan dan desain dari sistem yang akan diimplementasikan.

### Desain Umum Sistem

Sistem yang dibangun adalah untuk mensinkronisasi dua database SQL dan No SQL. Untuk basis data tipe SQL, Database Management System (DBMS) yang digunakan adalah MySQL. Sedangkan untuk basis data tipe NoSQL, DBMS yang digunakan adalah Apache HBase. Jumlan total server yang digunakan pada penelitian ini adalah berjumlah tiga server, satu server untuk MySQL, satu server untuk Apache HBase dan satu server untuk Data Adapter. Secara umum, visualisasi arsitektur yang digunakan sistem data adapter dapat dilihat pada Gambar 3.2.

Data adapter adalah penghubung antara aplikasi dengan basis data. Data adapter berperan untuk menerima permintaan dari aplikasi dan melakukan transformasi data dari MySQL ke Apache Hbase. Komponen utama dari sistem data adapter ini terdiri dari empat, yaitu basis data relasional, basis data NoSQL, DB Adapter dan DB Converter. DB Adapter berfungsi untuk menerima permintaan dari aplikasi seperti insert data, pembaruan data, menghapus data dan mengambil data. Untuk dapat melakukan perubahan dan pengaksesan data, data adapter menyediakan *interface* yang dapat diakses oleh aplikasi yang terhubung. *Interface* yang disediakan oleh data adapter adalah dalam bentuk *Application Programming Interface* (API). *Interface*, atau antar muka, ini menghasilkan bentuk tipe data JSON yang dibuat dengan menggunakan Flask. Flask adalah kerangka kerja mikro yang berjalan menggunakan bahasa pemrograman Python. Antar muka ini dapat diakses melalui port 5000 oleh aplikasi. Dengan antar muka ini, aplikasi dapat melakukan perubahan dan pengaksesan data pada basis data.

Gambar 3.2 Desain Sistem Data Adapter Secara Umum

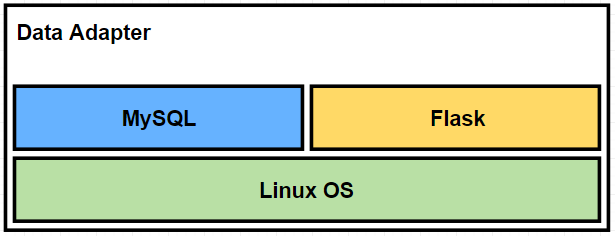
Sementara itu, DB Converter bertanggung jawab dalam transformasi data dan pelaporan hasil transformasi yang dicatat di basis data. Transformasi data dilakukan dari basis data RDB ke basis data NoSQL. Untuk menerima permintaan dari DB Adapter, DB Converter berjalan pada port 5001. Proses transformasi ini dilakukan dengan bantuan Apache Phoenix. Pada DB Converter ini juga dipasang sebuah basis data menggunakan MySQL untuk menyimpan log sinkronisasi. Komponen utama dari data adapter ini dibangun dengan menggunakan bahasa pemrograman Python.

### Desain Data Adapter

Data adapter memiliki peran paling penting di dalam sistem. Setiap permintaan yang dilakukan oleh aplikasi akan diolah terlebih dahulu di data adapter, yang kemudian akan diteruskan ke basis data. Ada beberapa contoh antarmuka yang digunakan oleh aplikasi, dalam pengerjaan tugas akhir ini, antarmuka yang digunakan adalah API dalam bentuk JSON. Diagram arsitektur data adapter dapat dilihat pada

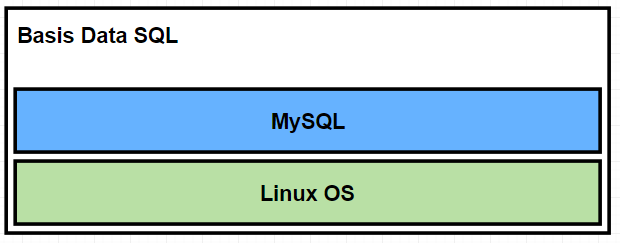
Data adapter terdiri dari dua komponen utama, yaitu DB Adapter dan DB Converter. DB Adapter berfungsi untuk menerima semua permintaan dari aplikasi seperti pengajuan kueri, pengambilan data, pembaruan data, dan penghapusan data. Untuk dapat melakukan hal ini, data adapter menyediakan antar muka dalam bentuk API. API menghasilkan keluaran data denga tipe data JSON. Semua komunikasi data yang dilakukan oleh aplikasi adalah melalui antar muka ini. Flask adalah kerangka kerja mikro yang digunakan untuk membuat fitur antar muka.

Sementara, DB Converter berfungsi untuk melakukan transformasi data dari basis data SQL ke basis data NoSQL. Dalam hal ini, transformasi yang dilakukan adalah dari MySQL ke Apache HBase. Secara *default*, permintaan perubahan data dari aplikasi akan diarahkan ke MySQL, sedangkan untuk permintaan pengaksesan data, permintaan dari aplikasi akan di arahkan ke HBase. Proses transformasi data ini dilakukan dengan bantuan Apache Phoenix. Apache Phoenix memungkinkan untuk menerjemahkan kueri SQL, mengkompilasinya menjadi beberapa rangkaian perintah HBase, kemudian mengeksekusinya ke HBase. Python menjadi bahasa pemrograman yang digunakan untuk mengimplementasikan semua proses yang ada pada data adapter. Gambar 3.3 menampilkan diagram arsitektur data adapter.



Gambar 3.3 Desain Arsitektur Data Adapter

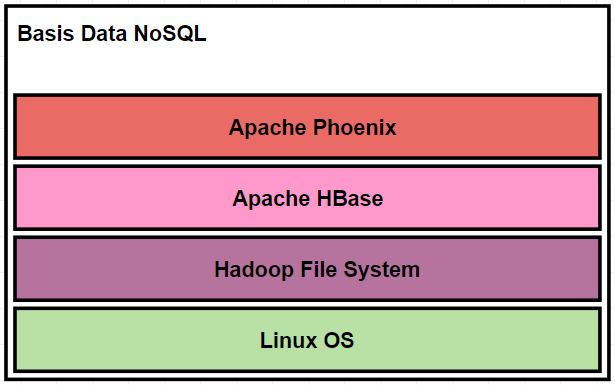
### Desain Basis Data SQL

Basis data SQL digunakan untuk menyimpan semua data aplikasi yang bersifat relasional. Dalam perancangan dan pengembangan suatu aplikasi dituntut kemampuan aplikasi untuk dapat diakses oleh banyak pengguna. Banyak pengembang yang memisahkan server aplikasi dengan server basis data atas dasar kebutuhan ini. Pada pengerjaan tugas akhir ini, satu server yang terpisah dari server aplikasi digunakan sebagai basis data SQL. Aplikasi yang DBMS digunakan adalah MySQL. Agar dapat digunakan aplikasi, perlu dibuatkan akun pengguna yang memiliki kewenangan untuk melakukan perubahan data pada server basis data. Semua permintaan dari aplikasi yang berkaitan dengan perubahan data, seperti *insert, update* dan *delete,* dan pengambilan data dapat dieksekusi di MySQL. Diagram arsiterkur basis data SQL tertera pada Gambar 3.4.

Gambar 3.4 Diagram Arsitektur Basis Data SQL

### Desain Basis Data NoSQL

Basis Data NoSQL digunakan untuk menyimpan data dengan format yang berbeda dengan basis data NoSQL. Basis data ini adalah basis data tambahan karena jumlah data yang semakin bertambah jumlahnya. Aplikasi yang digunakan sebagai basis data NoSQL adalah Apache HBase. Untuk dapat menggunakan Apache HBase, terlebih dahulu harus memasang Apache Hadoop, sistem berkas milik Apache yang khusus menangani *big data*, pada server karena Apache Hbase berjalan diatas Apache Hadoop.

Pemasangan basis data NoSQL dilakukan pada satu server lain yang terpisah dari aplikasi dan basis data SQL. HBase akan menerima hasil transformasi data dari MySQL yang dikontrol melalui data adapter. Proses transformasi ini membutuhkan bantuan dari Apache Phoenix sebagai penerjemah kueri SQL. Desain arsitektur basis data NoSQL dapat dilihat pada Gambar 3.5.

Gambar 3.5 Diagram Arsitektur Basis Data NoSQL

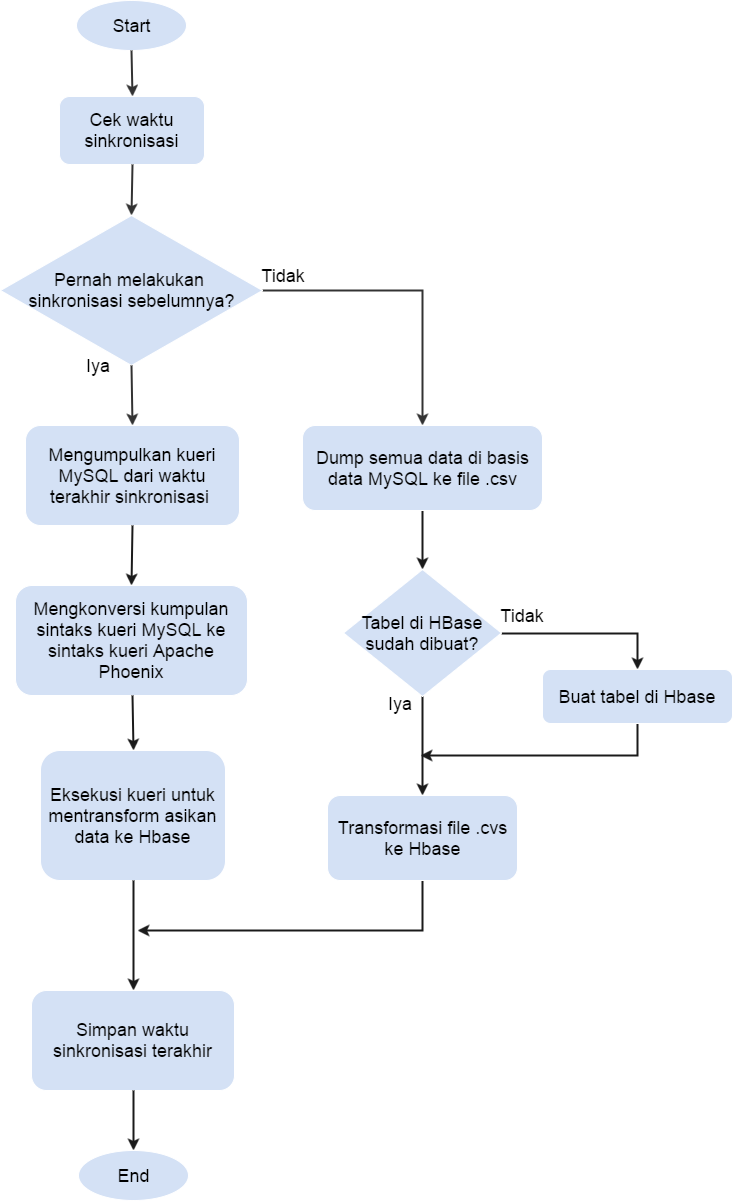
## Proses Transformasi dan Sinkronisasi

Proses transformasi dilakukan secara manual ketika user mengeksekusi aplikasi sinkronisasi. Hal pertama yang dilakukan ketika melakukan sinkronisasi adalah mengecek di basis data log sinkronisasi, apakah pernah melakukan sinkronisasi sebelumnya atau belum. Jika belum maka proses yang dilakukan adalah proses inisialisasi.

Proses inisialisasi adalah proses menyalin semua data di basis data SQL ke basis data NoSQL. Proses ini diawali dengan menge-*dump* atau mengekspor semua data pada tabel di MySQL menjadi berkas berformat .csv. Kemudian sistem akan membuat tabel baru yang sama dengan tabel yang ada pada basis data di MySQL. Setelah selesai, proses selanjutnya adalah melakukan transformasi data ke HBase. Proses transformasi data ini ditangani dengan menggunakan bantuan Apache Phoenix.

Jika pernah melakukan sinkronisasi sebelumnya, maka inisialisasi tidak akan dilakukan. Proses yang akan dilakukan adalah sistem akan mengumpulkan semua kueri dari log pada basis data MySQL. Kueri yang diambil adalah kueri-kueri yang melakukan perubahan pada basis data yaitu perintah *insert*, *update* dan *delete*. Semua log kemudian disimpan didalam satu file .sql. Dengan bantuan Apache phoenix, semua kueri pada berkas akan ditranslasikan ke HBase satu satu, sehingga semua perubahan data yang terjadi pada basis data SQL akan terjadi juga di basis data NoSQL.

Jika proses sinkronisasi berhasil, sistem akan mencatat log dan status sinkronisasi tersebut ke basis data log sinkronisasi. Basis data yang digunakan untuk mencatat log ini menggunakan MySQL dan terpisah dari server basis data SQL, server basis data NoSQL maupun server aplikasi. Diagram alur proses sinkronisasi dapat dilihat pada Gambar 3.6.



***Gambar 3.6 Diagram Alur Proses Sinkronisasi***

# BAB IV IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi sistem data adapter berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada yaitu basis data SQL, basis data NoSQL dan data adapter.

## Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan komputer dengan spesifikasi Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz dengan memory 5.8 GB di Laboratorium Arsitektur dan Jaringan Komputer, Jurusan Teknik Informatika ITS. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut :

* Sistem Operasi Linux Ubuntu 14.04 LTS
* Editor teks Sublime Text 2
* Python 2.7.6 untuk bahasa pemrograman utama
* Flask 0.12.1 untuk kerangka kerja pemrograman
* Virtual Environment 15.1.0 untuk lingkungan kerja pemrograman
* Apache Hadoop 2.7.3 untuk sistem berkas Apache HBase
* Apache HBase 1.2.4 untuk basis data NoSQL
* Apache Phoenix 4.10.0 untuk adapter antara MySQL dan Apache HBase
* MySQL versi 14.14 untuk basis data SQL
* Git 1.9.1 untuk pengelolaan versi program
* Postman 4.10.7 untuk uji coba API program
* Hedidi SQL 9.3.0.4984
* Microsoft Word 2013

## Rincian Implementasi Server Basis Data

Basis data digunakan untuk menyimpan semua data yang digunakan aplikasi. Pada pengerjaan Tugas Akhir ini, basis data yang akan disinkronisasikan adalah basis data SQL dan NoSQL. Sistem manajemen basis data SQL yang digunakan adalah MySQL Server dan untuk NoSQL menggunakan Apache HBase. Kedua basis data ini dipasang pada server yang terpisah. Pada sub bab ini akan dijelaskan secara rinci mengenai implementasi masing-masing basis data.

### Instalasi Server Basis Data SQL

Pada pengerjaan tugas akhir ini hanya menggunakan satu server untuk Basis data SQL. Perlu dilakukan konfigurasi lebih lanjut setelah melakukan pemasangan MySQL pada server agar data adapter dapat terhubung dengan server.

Alamat IP server diatur sesuai dengan subnet di Laboratorium Arsitektur dan Jaringan agar server data adapter dan server basis data NoSQL dapat saling terhubung dan bekomunikasi. Server yang digunakan untuk basis data SQL menggunakan alamat IP 10.151.36.22. Konfigurasi basis data agar berjalan dengan menggunakan IP yang tertera dengan mengganti variabel *bind-address*.

Secara default, MySQL hanya akan mencatat error log, loh yang berfungsi untuk mencatat kegagalan yang terjadi saat MySQL beroperasi. Pada pengerjaan Tugas Akhir ini, MySQL perlu mencatat log kueri apa saja yang dieksekusi karena data adapter akan membaca log ini untuk melakukan sinkronisasi dengan basis data HBase. Oleh karena itu perlu ditambahkan baris konfigurasi agar MySQL mencatat log query yang dieksekusi ke MySQL. Untuk memperbolehkannya, sunting berkas /etc/mysql/my.cnf dan tambahkan baris *log = /var/log/mysql/log\_query.log*. Kode Sumber 4.1 adalah rincian konfigurasi tambahan pada berkas my.cnf.

Secara default, server MySQL hanya bisa diakses oleh localhost saja. Oleh karena itu perlu ditambahkan konfigurasi agar data adapter dapat mengakses server MySQL. Dalam pengerjaan tugas akhir ini, karena tingkat keamanan saat ini tidak terlalu dibutuhkan, maka perintah untuk menambahkan akses ke server diperbolehkan untuk semua alamat IP. Perintah ini dijalankan seperti dengan mengeksekusi kueri di MySQL. Terlebih dahulu harus masuk sebagai user root. Langkah-langkah menambahkan akses user lebih detailnya adalah sebagai berikut:

* 1. Masuk ke MySQL dengan menggunakan username dan password default. Jalankan *perintah mysql –u root –p*, kemudian tekan enter dua kali.
  2. Buat pengguna baru dari semua alamat dengan nama ‘wicak’ dan password ‘w’ dengan menjalankan perintah *CREATE USER ‘wicak’@’%’ IDENTIFIED BY ‘w’*;
  3. Berikan akses kepada pengguna baru yang kita buat agar pengguna yang dimaksud dapat mengakses basis data dari alamat IP lain. Jalankan perintah *GRANT ALL PRIVILEGES ON \*.\* TO ‘wicak’@’%’ IDENTIFIED BY ‘w’*;
  4. Muat ulang konfigurasi dengan menjalankan perintah *FLUSH PRIVILEGES*;

Kode Sumber 4.1 Konfigurasi MySQL pada Berkas my.cnf

|  |  |
| --- | --- |
| 1  2 | bind-address = 10.151.36.129  log = /var/log/mysql/log\_query.log |

### Instalasi Server Basis Data NoSQL

Basis data yang dipasang sebagai basis data NoSQL adalah Apache HBase. HBase dipasang pada satu server yang terpisah dengan server basis data MySQL. Untuk memasang HBase, terlebih dahulu harus melakukan instalasi Hadoop karena HBase berjalan diatas sistem berkas Hadoop.

#### Instalasi Hadoop

Pada bagian ini akan dipaparkan tahapan pemasangan sistem berkas Hadoop. Langkah-langkahnya adalah sebagai berikut :

Pastikan pada server yang digunakan telah terpasang java. Kemudian install OpenJDK dengan mengetikan perintah *apt-get install default-jdk*.

Kemudian buat pengguna untuk Hadoop. Namun terlebih dahulu buat group dengan dengan nama hadoop. Ketikan *sudo addgroup hadoop*.

Tambahkan pengguna baru pada group hadoop dengan menjalankan perintah *sudo adduser --ingroup hadoop hduser*.

Tambahkan pengguna hduser ke akses sudo dengan perintah *sudo adduser hduser sudo*.

Hadoop memerlukan SSH untuk memanajemen node-nya secara remote dan lokal. Lakukan instalasi ssh dengan menjalankan perintah *sudo apt-get install ssh*.

Hadoop menggunakan SSH (untuk mengakses node-nya) biasanya mengharuskan penggunanya untuk memasukan kata sandi. Namun, persyaratan ini bisa ditiadakan dengan membuat dan mengatur sertifikat SSH dengan menjalankan perintah *ssh-keygen –t rsa –P “.*

Selanjutnya adalah menambahkan kunci baru ke daftar kunci yang berwenang sehingga Hadoop dapat menggunakan ssh tanpa meminta password. Ketikan perintah *cat $HOME/ .ssh/id\_rsa.pub >> $HOME/ .ssh/authorized\_keys*.

Unduh berkas instalasi Hadoop dengan perintah *wget http://mirrors.sonic.net/apache/hadoop/common/stable/hadoop-2.7.3.tar.gz.*

Untuk mengekstrak berkas Hadoop jalankan tar xvzf hadoop-2.7.3.tar.gz.

Kemudian pindahkan semua berkas instalasi ke direktori */usr/local/hadoop*.

Selanjutnya adalah melakukan konfigurasi pada Hadoop. Yang pertama adalah melakukan konfigurasi pada berkas bashrc dengan menjalankan perintah *vim ~/.bashrc*. Konfigurasi yang ditambahkan pada berkas .bashrc dapat dilihat pada Kode Sumber 4.1

Tambahkan variabel JAVA\_HOME kedalam berkas hadoop-env.sh agar ketika Hadoop dijalankan maka nilai JAVA\_HOME tersedia. Ketikkan perintah *vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh* untuk merubahnya. Isi konfigurasi variable ini dapat dilihat pada Kode Sumber 4.3.

Berkas */usr/local/hadoop/etc/hadoop/core-site.xm*l berisi konfigurasi yang digunakan Hadoop ketika mengawali proses. Buka dan sunting berkas tersebut dengan perintah *vim /usr/local/hadoop/etc/hadoop/core-site.xml*. Isi konfigurasi dapat dilihat pada Kode Sumber 4.4.

Salin dan *rename* */usr/local/hadoop/etc/hadoop/mapred-site.xml.template* menjadi *mapred-site.xml*. Kemudian tambahkan konfigurasi seperti pada Kode Sumber 4.5.

*Berkas /usr/local/hadoop/etc/hadoop/hdfs-site.xml* diatur untuk setiap *host* kluster yang digunakan. Ubah *berkas hdfs-site.xml* menjadi seperti pada Kode Sumber 4.6. Sebelum menyuntingnya, buat dua direktori baru yang akan terdiri *namenode* dan *datanode* Hadoop. Buat direktori dengan menjalankan perintah *mkdir –p /usr/local/hadoop-store/hdfs/namenode* dan *mkdir –p /usr/local/hadoop-store/hdfs/datanode.*

Sistem berkas Hadoop perlu di format untuk dapat digunakan. Jalankan perintah *hadoop namenode –format*.

Hadoop telah siap digunakan. Masuk ke direktori */usr/local/hadoop/sbin*. Kemudian gunakan *start-all.sh* untuk memulai Hadoop. Tampilan Web UI Hadoop dapat dilihat dengan mengakses *http://localhost:50070* pada *browser*.

Kode Sumber 4.2 Konfigurasi pada Berkas .bashrc

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | # HADOOP VARIABLES START  export JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64  export HADOOP\_INSTALL=/usr/local/hadoop  export PATH=$PATH:$HADOOP\_INSTALL/bin  export PATH=$PATH:$HADOOP\_INSTALL/sbin  export HADOOP\_MAPRED\_HOME=$HADOOP\_INSTALL  export HADOOP\_COMMON\_HOME=$HADOOP\_INSTALL  export HADOOP\_HDFS\_HOME=$HADOOP\_INSTALL  export YARN\_HOME=$HADOOP\_INSTALL  export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_INSTALL/lib/native  export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_INSTALL/lib" |

Kode Sumber 4.3 Variable JAVA\_HOME pada hadoop-env.sh

|  |  |
| --- | --- |
| 1  2 | export JAVA\_HOME=${JAVA\_HOME}  export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64 |

Kode Sumber 4.4 Konfigurasi pada Berkas core-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <configuration>  <property>  <name>hadoop.tmp.dir</name>  <value>/app/hadoop/tmp</value>  <description>A base for other temporary directories.</description>  </property>  <property>  <name>fs.default.name</name>  <value>hdfs://localhost:54310</value>  <description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.  </description>  </property>  </configuration> |

Kode Sumber 4.5 Konfigurasi pada Berkas mapred-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <configuration>  <property>  <name>mapred.job.tracker</name>  <value>localhost:54311</value>  <description>The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.  </description>  </property>  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  </configuration> |

Kode Sumber 4.6 Konfigurasi pada Berkas hdfs-site.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  <description>Default block replication. The actual number of replications can be specified when the file is created. The default is used if replication is not specified in create time.  </description>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>file:/usr/local/hadoop\_store/hdfs/namenode</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:/usr/local/hadoop\_store/hdfs/datanode</value>  </property>  </configuration> |

#### Instalasi Apache HBase

Setelah melakukan instalasi sistem berkas Hadoop, dapat dilakukan instalasi Apache HBase. Apache HBase yang digunakna adalah versi 1.2.4. Pada sub bab ini akan dijelaskan mengenai cara pemasangan Apache HBase.

* + - 1. Unduh paket Apache Hbase dengan menjalankan perintah *wget https://archive.apache.org/dist/hbase/1.2.4/hbase-1.2.4-bin.tar.gz*
      2. Unzip berkas dengan menjalankan tar –xvf hbase-1.2.4-bin.tar.gz.
      3. Buat direktori hbase dengan menjalankan perintah *mkdir /usr/lib/hbase.* Pindahkan folder hasi ekstrasi kedalam folder ini.
      4. Atur path java yang ada pada server pada berkas hbase-env.sh. Ubah dengan menjalankan perintah vim /usr/lib/hbase/hbase-1.2.4/conf/hbase-env.sh dan tambahkan konfigurasi seperti pada
      5. Tambahkan juga path HBASE\_HOME pada berkas .bashrc. Tambahkan konfigurasi seperti pada dengan menjalan perintah *vim ~/.bashrc*.
      6. Selanjutnya adalah mengubah berkas konfigurasi pada HBase pada berkas hbase-site.xml untuk mengatur direktori tempat HBase akan menyimpan data. Menunjukan isi konfigurasi *hbase-site.xml*
      7. Untuk memulai HBase masuk ke direktori */usr/lib/hbase/hbase-1.2.4/bin* kemudian jalankan start-hbase.sh.
      8. Untuk mengecek apakah HBase telah berjalan, jalankan perintah *hbase shell.*

## Rincian Implementasi Sistem Data Adapter

Komponen utama pada sistem data adapter yaitu DB Adapter dan DB Converter. Sistem data adapter dibangun dengan menggunakan bahasa pemrograman Python dan kerengka kerja pemrograman Flask. Selain itu, digunakan beberapa modul tambahan yang dipasang secara terpisah dengan menggunakan pip pada terminal. Pip adalah sebuah perkakas untuk melakukan pemasangan modul-modul paket Python. Pada sub bab ini akan dijelaskan secara rinci mengenai implementasi data adapter.

### Instalasi Paket

Seperti yang dijelaskan sebelumnya, pemasangan modul-modul Python ini dilakukan dengan menggunakan pip. Cara kerja pip ini adalah mengunduh modul untuk sistem yang dibutuhkan, kemudian dipasang pada server. Semua modul yang dibutuhkan sistem disimpan dalam berkas requirement\_interface.txt. Untuk menjalankan pemasangan modul di dalam file tersebut dapat dijalankan dengan menggunakan perintah pip install –r requirement\_interface.txt. Semua daftar modul di dalam berkas tersebut akan dipasang di server. Modul-modul tambahan yang dipasang yaitu :

* Flask 0.12.1
* Appdirs 1.4.3
* Click 6.7
* Itsdangerous 0.24
* Jinja2 2.9.6
* MarkupSafe 1.0
* MySQL-python 1.2.5
* Packaging 16.8
* Protobuf 3.2.0
* Pyparsing 2.2.0
* Six 1.10.0
* Werkzeug 0.12.1

### Implementasi DB Adapter

DB Adapter adalah komponen pada sistem data adapter yang bertugas sebagai penghubung antara aplikasi dengan basis data. DB Adapter menerima kueri, mengeksekusi kueri dan mendapatkan hasilnya dari basis data melalui DB Converter. DB Adapter menyediakan antar muka yang dapat diakses oleh aplikasi dimana antarmuka ini dapat menerima permintaan yang dilakukan oleh aplikasi seperti pengajuan kueri, pengambilan data, pembaruan data, dan penghapusan data. Semua permintaan ini diterima oleh DB Adapter dengan menggunakan API yang dibuat dengan menggunakan Flask. Flask yang digunakan adalah versi 0.12.1. Antar muka ini menghasilkan keluaran data dalam bentuk JSON.

Untuk melakukan permintaan atau proses data, DB Adapter menyediakan antar muka yang diakses oleh aplikasi dengan melakukan permintaan ke rute yang disediakan. Dalam pengerjaan Tugas Akhir ini, beberapa rute yang disediakan beserta dengan rincian penjelasannya dapat dilihat pada Tabel 4.1. Rute ini diimplementasikan dalam sebuah berkas python.

Tabel 4.1 Implementasi Rute Pada DB Adapter

| **No** | **Rute** | **Masukan** | **Luaran** | **Proses** |
| --- | --- | --- | --- | --- |
| 1 | GET /sinkron | - | status, message | Melakukan sinkronisasi basis data MySQL ke HBase |
| 2 | POST /insert\_routes | data route | status, message | Memasukan data baru ke tabel routes |
| 3 | POST /delete\_routes\_by\_id | id\_route | status, message | Menghapus satu baris data data pada tabel routes |
| 4 | POST /update\_routes\_by\_id | id\_route | status, message | Melakukan perbaruan data pada tabel routes |
| 5 | GET /select\_all\_routes | - | host, database, rows, flight\_routes | Mengambil semua data pada tabel routes |
| 6 | GET /select\_route\_by\_id/<id> | id\_route | host, database, route | Mengambil satu baris data pada tabel route berdasarkan id |
| 7 | POST /insert\_airline | data\_airline | Status, message | Memasukan satu barus data baru ke tabel airline |
| 8 | POST /delete\_airline\_by\_id | id\_airline | Status, message | Menghapus data pada tabel airline berdasarkan id airline |
| 9 | POST /update\_airline\_by\_id | data\_airline | Status, message | Memperbarui data pada tabel airline |
| 10 | GET /select\_all\_airline | - | Host, database, rows, airlines | Mengambil semua data pada tabel airline |
| 11 | GET /select\_airline\_by\_id/<id> | Id\_airline | Host, database, airline | Mengambil satu baris data pada tabel airline |
| 12 | POST /insert\_airport | Data\_airport | Status, message | Memasukan satu baris data baru ke tabel airport |
| 13 | POST /delete\_airport\_by\_id | Id\_airport | Status, message | Menghapus satu barus data pada tabel airport |
| 14 | POST /update\_airport\_by\_id | data\_airport | Status, message | Memperbarui data pada tabel airport |
| 15 | GET /select\_all\_airport | - | Host, database, rows, airport | Mengambil semua data pada tabel airport |
| 16 | GET /select\_airport\_by\_id/<id> | id\_airport | Host, database, airport | Mengambil satu baris data pada tabel airport berdasarkan id |
| 17 | GET /select\_routes\_country/<id> | id\_route | Host, database, route | Mengambil detail dari tabel route dan tabel airline |
| 18 | GET /select\_all\_number\_routes\_by\_airline | - | Host, rows, database, routes | Mengambil jumlah rute dari setiap airline |
| 19 | GET /select\_all\_number\_routes\_more\_than/<number> | number | Host, database, routes, rows | Mengambil data airline yang memiliki rute lebih dari nomor yang ditentukan |
| 20 | GET /select\_all\_airline\_in\_routes | - | Host, database, airlines, rows | Mengambil semua airline yang ada pada tabel rute |

Untuk melakukan pengolahan data, terdapat *controller* di tiap-tiap di rute yang berfungsi untuk memilih pada basis data mana kueri akan dieksekusi. Jika kueri bersifat merubah data pada basis data, maka eksekusi kueri diarahkan ke basis data SQL, dalam penelitian ini yaitu MySQL. Pseudocode kontroler untuk proses perubahan data ditunjukan pada Pseudocode 4.1. Sedangkan untuk kueri yang memiliki sifat mengambil data, maka eksekusi kueri diarahkan ke basis data NoSQL, dalam hal ini adalah HBase. Pseudocode 4.2 menunjukan proses eksekusi kueri untuk pengambilan data.

|  |  |
| --- | --- |
| 1  2  3  4 | **post** data from application  connect to MySQL DB  **excecute** query  **return** result |

Pseudocode 4.1 Eksekusi Kueri untuk Proses Perubahan Data

Penjelasan Pseudocode 4.1 adalah sebagai berikut :

Mengambil data *post* dari aplikasi

Jika aksi yang dilakukan aplikasi adalah perubahan pada basis data, seperti menambahkan, menghapus dan menyunting, maka DB Adapter terhubung basis data MySQL.

Proses menjalankan sintaks kueri

Mengembalikan hasil eksekusi kueri

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **get** status synchroniztion  **if** syncrhonization process still working  connect to MySQL DB  **else**  connect to HBase  result = **excecute** query  **return** result |

Pseudocode 4.2 Eksekusi Kueri untuk Proses Pengambilan Data

Secara default, proses pengambilan data akan diarahkan ke HBase karena untuk proses pembacaan data pada HBase menghabiskan waktu yang lebih singkat dibandingkan dengan basis data relasional. Berikut adalah penjelasan Pseudocode 4.2 :

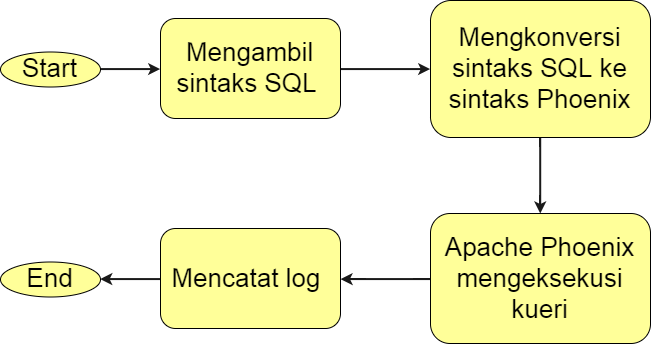
1. Pengeksekusian dimulai dari mengecek status apakah sedang berlangsung proses sinkronisasi atau tidak
2. Cek jika proses sinkronisasi sedang berlangsung, maka proses pengambilan data diarahkan ke basis data MySQL. Jika tidak maka diarahkan ke HBase.
3. Proses menjalankan sintaks kueri.
4. Mengembalikan hasil kueri.

### Implementasi DB Converter

DB Converter adalah komponen lain data adapter yang berfungsi untuk mencatat log dan status sinkronisasi serta melakukan transformasi data dari basis data SQL ke basis data NoSQL. Komponen yang terpasang pada DB Converter ini diantaranya Flask dan Apache Phoenix.

Flask digunakan untuk membuat antar muka yang berfungsi untuk menerima permintaan proses sinkronisasi dari DB Adapter. Apache Phoenix berfungsi sebagai lapisan layer yang bekerja diatas HBase yang dapat menerjemahkan SQL dan mengeksekusinye ke HBase. Terdapat basis data yang menggunakan MySQL untuk mencatat log dan status setiap kali menjalankan proses sinkronisasi.

Ketika DB Adapter menerima permintaan sinkronisasi, DB Adapter kemudian meneruskan permintaan tersebut dengan meminta DB Converter untuk melakukan sinkronisasi. DB Converter mengambil sintaks SQL dari log kueri pada server MySQL. Kueri ini kemudian dikonversi menjadi sintaks Apache Phoenix. Kemudian Apache Phoenix akan menjalankan kueri tersebut ke Apache HBase. Data di HBase akan berubah sesuai dengan kueri yang dijalankan. Setelah proses transformasi selesai, sistem mencatat log pada basis data log sinkronisasi. Gambar 4.1 menampilkan diagram alur proses sinkronisasi pada DB Converter. Kode sumber menjelaskan alur pemrosesan permintaan sinkronisasi dari aplikasi.



Gambar 4.1 Diagram Alur DB Converter

Proses sinkronisasi basis data SQL ke NoSQL dari MySQL ke Apache HBase ditunjukan pada Pseudocode 4.3. Penjelasan secara rinci dari Pseudocode 4.3 adalah sebagai berikut:

1. Proses sinkronisasi diawali dengan mengambil data-data server dari berkas konfigurasi. Data-data ini diambil dari berkas *configuration.ini*.
2. Untuk menandakan jika proses sinkronisasi sedang berjalan, simpan inisialisasi log sinkronisasi pada tabel log sinkronisasi dengan status 0.
3. Cek apakah pada tabel log sinkronisasi terdapat data atau tidak. Jika kosong, maka sinkronisasi belum pernah dilakukan dan sistem akan melakukan proses inisialisasi. Proses inisialisasi adalah membuat tabel baru pada hbase dan menampung semua data pada mysql kedalam sebuah berkas *mysql\_table.csv*. Kemudian berkas *mysql\_table.csv* akan di import ke basis data Apache HBase.
4. Jika terdapat log, maka proses sinkronisasi sudah pernah dijalankan sebelumnya. Maka langkah selanjutnya adalah melakukan proses *patching* atau penambalan. *Patching* diawali dengan mengambil semua sintaks kueri pada berkas log di server basis data MySQL. Kemudian sintkas-sintaks kueri tersebut akan di konvert menjadi sintaks Apache Phoenix dan di simpan kedalam satu berkas *list\_all\_query.sql*. Selanjutnya *list\_all\_query.sql* akan di eksekusi pada basis data Apache HBase.
5. Jika proses telah berakhir, maka *update* status pada tabel log sinkronisasi untuk menandakan jika proses sinkronisasi sedang tidak berlangsung.
6. Catat waktu yang dihabiskan untuk melakukan proses sinkronisasi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | **get** mysql\_db\_conf\_data  **get** hbase\_db\_conf\_data  **get** sync\_log\_db\_conf\_data  **get** ssh\_access\_data  **get** last\_sync\_log\_data  **set** initial\_sync\_log  **set** start\_time  query\_count = 0  do\_patching = false  **if** (last\_sync\_log\_data is empty)  dump mysql\_table to mysql\_table.csv  **execute** create\_table.sql with apache phoenix  **if** (create\_table is success)  **execute** mysql\_table.csv to hbase with apache phoenix  **else**  ssh to mysql\_db\_server  get log\_query\_file  **for** line **in** log\_query\_file  **get** query\_time  **if** (query\_time **>** last\_sync\_log\_data)  get query\_syntax  query\_tmp = convert query\_syntax to apache phoenix syntax  append query\_tmp to list\_all\_query.sql  do\_patching = true  **if** (query\_count **is** 0)  update sync\_log  **if** (do\_patching **is** true)  **execute** list\_all\_query.sql with apache phoenix  update sync\_log  duration = time\_now – start\_time |

Pseudocode 4.3 Proses Sinkronisasi

Pada proses patching, terdapat proses untuk melakukan perubahan sintaks kueri MySQL menjadi sintaks kueri Apache Phoenix karena terdapat beberapa kueri yang memiliki perbedaan struktur sintkas SQL. Secara garis besar, kueri *delete* dan *select* pada MySQL memiliki struktur yang sama dengan Apache Phoenix, namun Apache Phoenix menggunakan sintaks *upsert* untuk mengeksekusi perintah *insert* dan *update* pada MySQL. Oleh karena itu diperlukan fungsi yang digunakan untuk melakukan konversi tersebut. Pseudocode 4.4 menunjukan konverter sintaks kueri *insert*. Untuk konverter sintkas kueri *update* dapat dilihat pada Pseudocode 4.5.

|  |  |
| --- | --- |
| 1  2 | **get** insert\_str  **replace** ‘INSERT’ **to** ‘UPSERT’ in insert\_str |

Pseudocode 4.4 Konverter Sintaks Kueri Insert MySQL ke Sintaks Kueri Apache Phoenix

Penjelasan secara rinci Pseudocode 4.4 adalah sebagai berikut :

1. Ambil sintaks kueri MySQL
2. Ganti string ‘*INSERT’* menjadi ‘*UPSERT’*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **get** update\_str  table\_name = select string between ‘UPDATE and ‘SET  list\_table = get list primary key from file configuration  **if** table\_name in list\_table  **get** table\_primary\_key  **get** index\_table  **else**  **return** table not in list  kolom\_value\_list = select string between ‘SET and ‘WHERE  **for** i **in** kolom\_value\_list  split string i into kolom and value  append kolom to kolom\_list  append value to value\_list  where\_clause = select string after ‘WHERE’  final\_string = **merge** string ‘UPSERT’, table\_name, table\_primary\_key, kolom\_list, value\_list, and where\_clause to apache phoenix update syntax  **return** final\_string |

Pseudocode 4.5 Konverter Sintaks Kueri Update ke Sintaks Kueri Apache Phoenix

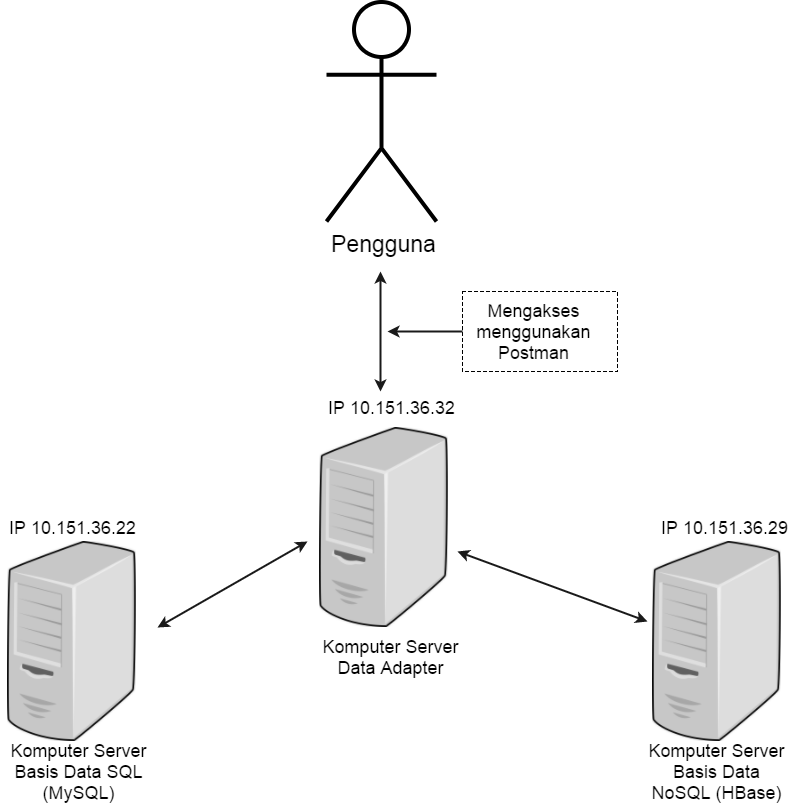
Untuk perubahan kueri update, dibutuhkan elemen tambahan yaitu *primary key*. *Primary key* didapat dengan mengambilnya pada berkas konfigurasi. Penjelasan secara rinci Pseudocode 4.5 adalah sebagai berikut :

1. Pertama adalah mengambil string update MySQL.
2. Dari struktur update MySQL, nama tabel didapatkan dengan mengambil kata yang ada di antara ‘*UPDATE’* dan ‘*SET’*.
3. Selanjutnya adalah mengecek apakah nama tabel tersebut ada di list\_table yang ada di basis data. Jika tabel terdapat dalam list maka ambil *primary key* tabel. Jika tidak, maka kembalikan nilai tabel tidak terdapat dalam list.
4. Mengambil nilai kolom dan value yang terletak di antara string ‘*SET’* dan ‘*WHERE’*. Dari *string* ini, kumpulkan kolom dan nilai menjadi masing-masing ke variabel *kolom \_list* dan *value*\_*list*.
5. Mengambil string yang diikuti setelah *string* ‘*WHERE’* dan simpan ke variabel *where\_clause* untuk mendapatkan klause where atau kondisi dimana sintaks akan dieksekusi.
6. Menggabungkan semua variabel *table\_name, primary\_key, kolom\_list, value\_list,* dan *where\_clause* menjadi satu baris sintaks *upsert*.

# 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 kebenaran dan uji kinerja serta analisa setiap pengujian.

## Lingkungan Uji Coba



Gambar 5.1 Desain Arsitektur Uji Coba Sistem

Lingkungan untuk pengujian menggunakan tiga buah server yang terdiri dari server basis data SQL, server basis data NoSQL dan server data adapter serta satu buah komputer sebagai komputer penguji. Proses pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer gedung Teknik Informatika ITS. Gambar 5.1 menunjukan desain arsitektur untuk proses uji coba yang digunakan. Spesifikasi perangkat lunak dan perangkat keras masing-masing komputer dijelaskan sebagai berikut:

* Server basis data SQL (Alamat IP: 10.151.36.22)
  + Perangkat Keras
    - * Komputer Fisik
      * Processor Intel(R) Core(TM) i3-530 CPU @ 2.39GHz (4 CPUs)
      * RAM 1.8 GB
      * Hardisk 52.9 GB
  + Perangkat Lunak
    - * Ubuntu Desktop 14.04 LTS
      * MySQL server
* Server basis data NoSQL (Alamat IP: 10.151.36.29)
  + Perangkat Keras
    - * Komputer Fisik
      * Processor Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz (4 CPUs)
      * RAM 5.8 GB
      * Hardisk 84 GB
  + Perangkat Lunak
    - * Ubuntu Desktop 14.04 LTS
      * Apache Hadoop 2.7.3
      * Apache HBase 1.2.4
      * Apache Phoenix 4.10.0
      * Flask 0.12.1
      * Python 2.7.6
* Server data adapter (Alamat IP: 10.151.36.32)
  + Perangkat Keras
    - * Komputer Fisik
      * Processor Intel(R) Core(TM) 2 Duo-E4600 CPU @ 2.40GHz (2 CPUs)
      * RAM 1.9 GB
      * Hardisk 196.7 GB
  + Perangkat Lunak
    - * Ubuntu Desktop 16.04 LTS
      * MySQL server
      * Flask 0.12.1
      * Python 2.7.6
* Komputer Penguji
  + Perangkat Keras
    - * Komputer Fisik
      * Processor Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz (4 CPUs)
      * RAM 4 GB
      * Hardisk 500 GB
  + Perangkat Lunak
    - * Windows 8.1 Pro
      * Postman 4.10.7
      * Microsoft Word 2013

## Dataset Uji Coba

Pada tugas akhir ini, penulis menggunakan data set dari [28] yang berisi data-data tentang penerbangan diseluruh dunia. Data-data tersebut terdiri dari tiga tabel yaitu :

1. Tabel Routes
   * + - Baris : 67663 baris
       - Kolom :

id\_route (integer)

airline (varchar)

id\_airline (integer)

src\_airport (varchar)

id\_src\_airport (integer)

dst\_airport (varchar)

id\_dst\_airport (integer)

codeshare (varchar)

stop\_val (integer)

equipment (varchar)

log\_date (date)

1. Tabel Airline
   * Baris : 6161
   * Kolom :

id\_airline (integer)

name (varchar)

alias (varchar)

iata (varchar)

icao (varchar)

callsign (varchar)

country (varchar)

active\_stat (varchar)

1. Tabel Airport
   * Baris : 7184
   * Kolom :

airport\_id (integer)

name\_airport (varchar)

city (varchar)

country (varchar)

iata (varchar)

icao (varchar)

latitude (varchar)

longitude (varchar)

altitude (varchar)

timezone (varchar)

dst (varchar)

tz\_db (varchar)

type\_airport (varchar)

source (varchar)

Data set tersebut disimpan dalam berkas dengan format .csv. Jumlah baris dan tabel tersebut tidak sepenuhnya digunakan tetapi dirubah sesuai dengan uji coba yang akan dilakukan.

## Skenario Uji Coba

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

### Skenario Uji Coba Fungsionalitas

Uji coba dilakukan dengan mengakses semua antar muka yang ada dan menguji fungsi sinkronisasi dari data adapter. Pengujian dilakukan oleh pengguna dengan menggunakan bantuan aplikasi Postman. Proses pengujian ini berguna untuk menguji apakah operasi-operasi perubahan dasar pada data dapat dilakukan dan memberikan hasil yang diharapkan. Operasi dasar dari data adapter dapat dikategorikan kedalam tiga fitur, yaitu sebagai berikut :

* **Inisialisasi Basis Data**

Inisialisasi basis data akan membuat tabel baru pada basis data NoSQL yaitu HBase. Jika tabel pada HBase telah siap, maka sistem akan mengekspor data pada basis data SQL ke dalam sebuah berkas dengan format .csv. Data ini kemudian dieksekusi ke basis data NoSQL. Pengujian ini akan memastikan proses ini berjalan ketika proses sinkronisasi pada basis data SQL dan NoSQL belum pernah dilakukan sebelumnya, atau dalam kata lain sistem untuk pertama kalinya menjalankan proses sinkronisasi.

* **Transformasi Basis Data**

Proses transformasi berlangsung ketika terdapat perubahan data pada basis data SQL. Perubahan terjadi dari adanya kueri dari aplikasi seperti *insert*, *update* dan *delete*. Jika sistem sebelumnya sudah pernah melakukan transformasi, maka proses inisialisasi tidak akan berlangsung. Sistem akan mengambil daftar kueri yang di esekusi pada basis data SQL, kemudian dijalankan ke basis data NoSQL sehingga data pada kedua basis data adalah sama.

* **Antar Muka Data Adapter**

Antar muka data adapter menyediakan rute yang dapat diakses oleh aplikasi untuk melakukan suatu proses kepada basis data. Fitur ini juga bertugas untuk memilih dari basis data mana permintaan aplikasi akan di proses. Pada pengujian ini diharapkan setiap permintaan dari aplikasi yang berupa perubahan data, yaitu kueri *insert*, *update* dan *delete*, akan diarahkan ke basis data SQL dan untuk permintaan pengambilan data akan diarahkan ke basis data NoSQL. Tabel 5.1 menunjukan rancangan setiap aksi pengujian dan hasil atau nilai yang diharapkan.

Selain melakukan uji coba untuk mengetahui berhasil atau tidaknya antar muka berjalan, dilakukan pencatatan waktu respon yang diperlukan sistem sampai proses berakhir kedalam tabel. Waktu respon dicatat sebanyak lima kali percobaan, kemudian diambil nilai rata-ratanya.

Tabel 5.1 Aksi dan Hasil Harapan Setiap Fungsi Antar Muka

| **No** | **Pengendali** | **Uji Coba** | **Hasil Harapan** |
| --- | --- | --- | --- |
| 1 | /sinkron | Melakukan sinkronisasi basis data | Status sinkronisasi berhasil, log sinkronisasi tersimpan. Isi data pada basis data SQL dan NoSQL sama |
| 2 | /insert\_routes | Menambahkan satu baris baru ke tabel routes | Satu baris rute pada tabel routes bertambah |
| 3 | /delete\_routes\_by\_id | Menghapus satu baris data pada tabel routes | Satu baris rute pada tabel route terhapus |
| 4 | /update\_routes\_by\_id | Menyunting data pada tabel routes | Data pada tabel routes diperbarui nilainya |
| 5 | /select\_all\_routes | Mengambil semua baris data pada tabel routes | Semua data pada tabel routes terambil |
| 6 | /select\_route\_by\_id/<id> | Mengambil satu baris data pada tabel routes | Satu baris yang dipilih diambil dari tabel routes |
| 7 | /insert\_airline | Menambahkan satu baris baru ke tabel airline | Satu baris data berhasil ditambahkan pada tabel airline |
| 8 | /delete\_airline\_by\_id | Menghapus satu baris data pada tabel airline | Satu baris data berhasil dihapus dari tabel airline |
| 9 | /update\_airline\_by\_id | Menyunting data pada tabel airline | Data pada tabel airline berhasil diperbarui |
| 10 | /select\_all\_airline | Mengambil semua baris data pada tabel airline | Semua data pada tabel airline terambil |
| 11 | /select\_airline\_by\_id/<id> | Mengambil satu baris data pada tabenl airline | Satu baris yang dipilih berhasil diambil dari tabel airline |
| 12 | /insert\_airport | Menambahkan satu baris data baru pada tabel airport | Satu baris data airport baru ditambahkan pada tabel airport |
| 13 | /delete\_airport\_by\_id | Menghapus satu baris data pada tabel airport | Satu baris data airport berhasil dihapus dari tabel airport |
| 14 | /update\_airport\_by\_id | Menyunting data pada tabel airport | Data pada tabel airport berhasil diperbarui |
| 15 | /select\_all\_airport | Mengambil semua data pada tabel airport | Semua data pada tabel airport berhasil diambil |
| 16 | /select\_airport\_by\_id/<id> | Mengambil satu baris data pada tabel airport | Satu baris yang dipilih berhasil diambil dari tabel airport |
| 17 | /select\_routes\_country/<id\_route> | Mengambil detail nama airline dan negara pada rute | Satu baris yang dipilih berhasil diambil dari tabel routes dan airline |
| 18 | /select\_all\_number\_routes\_by\_airline | Mengambil jumlah rute yang ditempuh oleh airline | Semua data jumlah rute berhasil diambil dari tabel route dan airline |
| 19 | /select\_all\_number\_routes\_more\_than/<number> | Mengambil airline yang memiliki jumlah rute lebih dari jumlah yang dimaksud | Semua data jumlah rute yang melebihi berhasil diambil dari tabel route |
| 20 | /select\_all\_airline\_in\_routes | Mengambil data airline yang melakukan penerbangan | Semua data jumlah rute tiap airline berhasil diambil dari tabel route |
| 21 | /select\_all\_airline\_source | Mengambil detail airline dan asal penerbangan yang tercatat pada rute | Semua data airline dan asal penerbangan dari gabungan tabel route, airline dan airport |

### Skenario Uji Coba Kapasitas dan Performa

Pada pengujian ini dilakukan untuk menguji kemampuan tiap-tiap server yang ada pada sistem adapter. Parameter yang menjadi acuan pengukuran adalah dengan menghitung penggunaan CPU, *memory* dan waktu respon selama proses inisialisasi atau transformasi. Hasil uji coba ini disimpan dalam tabel dan ditampilkan dalam bentuk grafik sehingga dapat terlihat dengan jelas perubahan dari setiap skenario yang dijalankan. Uji coba menggunakan arsitektur yang sama seperti yang ditampilkan pada Gambar 5.1.

Komputer server diuji secara bertahap, mulai dari proses transformasi maupun *patching* kueri. Kumpulan data yag didapat diolah untuk mendapatkan parameter yang diinginkan. Pengujian dilakukan dengan beberapa skenario sebagai berikut:

**Perbandingan Jumlah Baris pada Tabel**

Pengujian ini dilakukan untuk mengetahui berapa jumlah sumber daya yang digunakan dan waktu respon selama proses transformasi data dari basis data MySQL ke HBase saat proses inisialisasi berlangsung dengan perbandingkan jumlah baris. Perbandingan jumlah baris yang digunakan pada tabel untuk setiap percobaan adalah sebagai berikut :

* Airplane : 6.000, 12.000, 18.000, 24.000, 30.000
* Airport : 7.000, 14.000, 21.000, 28.000, 35.000
* Route : 60.000, 120.000, 180.000, 240.000, 300.000

**Perbandingan Jumlah Tabel pada Basis Data**

Pengujian ini dilakukan untuk mengetahui berapa jumlah sumber daya yang digunakan dan waktu respon sistem data adapter selama proses inisialisasi data dari basis data MySQL ke HBase berlangsung dengan perbandingan perbedaan jumlah tabel. Jumlah baris *default* yang digunakan pada setiap tabel adalah sesuai dengan jumlah baris yang didapat pada [28]. Perbandingan jumlah tabel yang digunakan untuk setiap percobaan adalah 3, 6, 9, 12, dan 15 tabel.

**Perbandingan Jumlah Kueri *Insert, Update* dan *Delete***

Pengujian ini bertujuan untuk mengetahui seberapa sumberdaya dan waktu yang dihabiskan sistem untuk mengeksekusi tiga operasi kueri yang berbeda yaitu *insert, update* dan *delete* dari MySQL ke HBase pada proses tranformasi data. Pada setiap operasi kueri, perbandingan jumlah kueri yang digunakan adalah dimulai dari 1.000, 2.000, 3.000, 4.000 dan 5.000 kueri.

## Hasil Uji Coba dan Evaluasi

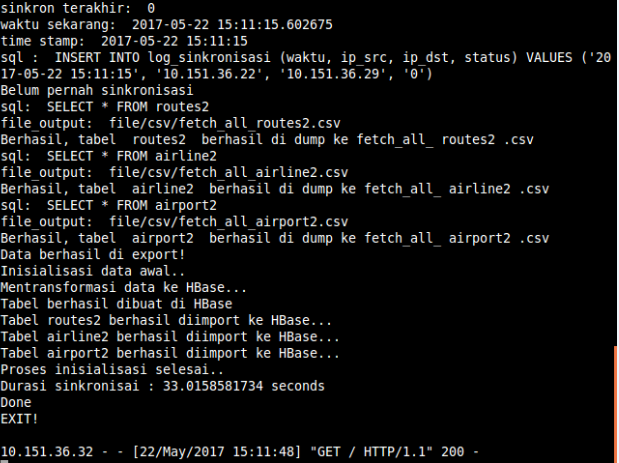
Pada subbab ini akan dijelaskan mengenai hasil pengujian yang dilakukan sesuai dengan skenario pengujian yang telah ditentukan. Pengujian yang dilakukan adalah uji coba fungsionalitas dan uji coba kapasitas dan performa.

### Hasil Uji Coba Fungsionalitas

Uji coba funcgsionalitas dilakukan oleh pengguna menggunakan bantuan aplikasi Postman. Waktu respon didapat dari perhitungan ketika memulai proses dan mengakhiri proses pada kode program dan dilihat melalui terminal server.

* **Inisialisasi Basis Data**

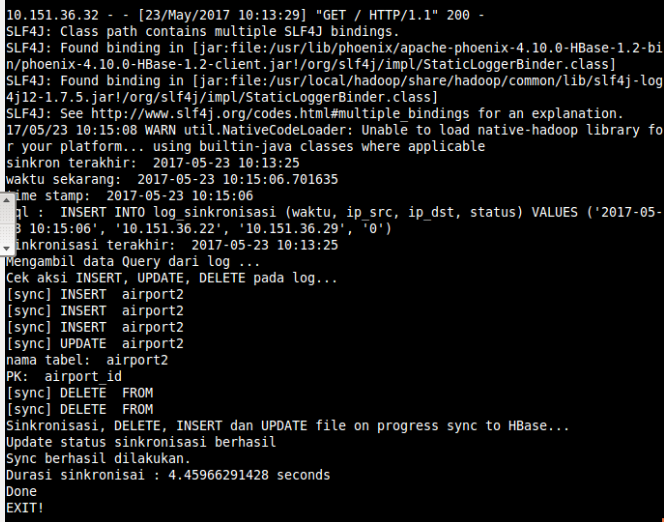
Pengujian ini dilakukan dengan melakukan transformasi data untuk pertama kali. Untuk mengujinya, data pada tabel log sinkronisasi di kosongkan dan tabel yang ada di basis data HBase dikosongkan. Kemudian isi data dari kedua basis data dicek apakah sudah sama atau belum Dari pengujian yang dilakukan, proses transformasi berjalan dengan sukses. Log proses selama proses sinkronisasi akan ditampilkan pada terminal. Gambar 5.2 menunjukan hasil sinkronisasi pada terminal.



Gambar 5.2 Tampilan pada Terminal Ketika Proses Inisialisasi Berlangsung

* **Transformasi Basis Data (*Pathcing*)**

Pengujian ini dilakukan jika sebelumnya proses inisialisasi sudah pernah dilakukan, karena sistem terlebih dahulu akan mengecek dari tabel log untuk memastikan jika proses sinkronisasi sudah pernah dilakukan. Pengujian dilakukan dengan menjalankan beberapa perintah kueri seperti *insert*, *update* dan *delete*. Kemudian dilakukan pengecekan apakah sistem berhasil membaca log kueri dari basis data SQL dan mengeksekusinya ke basis data HBase. Setelah proses transformasi ini, isi data pada kedua basis data di cek untuk memastikan jika kedua basis data memiliki data yang sama. Dari pengujian yang telah dilakukan, proses transformasi berjalan dengan sukses. Gambar 5.3 menampilkan log pada terminal selama proses transformasi berlangsung.



Gambar 5.3 Tampilan pada Terminal Ketika Proses Transformasi Berlangsung

* **Antar Muka Data Adapter**

Pengujian dilakukan dengan mengakses rute yang telah disediakan sistem dan disesuaikan dengan hasil harapan yang didapatkan. Daftar antar muka yang diuji berdasarkan alamat rute pengendali ditunjukan pada Tabel 5.2.

Tabel 5.2 Hasil Uji Coba Fungsionalitas

| **No** | **Pengendali** | **Uji Coba** | **Hasil** |
| --- | --- | --- | --- |
| 1 | /sinkron | Melakukan sinkronisasi basis data | Sukses |
| 2 | /insert\_routes | Menambahkan satu baris baru ke tabel routes | Sukses |
| 3 | /delete\_routes\_by\_id | Menghapus satu baris data pada tabel routes | Sukses |
| 4 | /update\_routes\_by\_id | Menyunting data pada tabel routes | Sukses |
| 5 | /select\_all\_routes | Mengambil semua baris data pada tabel routes | Sukses |
| 6 | /select\_route\_by\_id/<id> | Mengambil satu baris data pada tabel routes | Sukses |
| 7 | /insert\_airline | Menambahkan satu baris baru ke tabel airline | Sukses |
| 8 | /delete\_airline\_by\_id | Menghapus satu baris data pada tabel airline | Sukses |
| 9 | /update\_airline\_by\_id | Menyunting data pada tabel airline | Sukses |
| 10 | /select\_all\_airline | Mengambil semua baris data pada tabel airline | Sukses |
| 11 | /select\_airline\_by\_id/<id> | Mengambil satu baris data pada tabel airline | Sukses |
| 12 | /insert\_airport | Menambahkan satu baris data baru pada tabel airport | Sukses |
| 13 | /delete\_airport\_by\_id | Menghapus satu baris data pada tabel airport | Sukses |
| 14 | /update\_airport\_by\_id | Menyunting data pada tabel airport | Sukses |
| 15 | /select\_all\_airport | Mengambil semua data pada tabel airport | Sukses |
| 16 | /select\_airport\_by\_id/<id> | Mengambil satu baris data pada tabel airport | Sukses |
| 17 | /select\_routes\_country/<id\_route> | Mengambil satu baris data rute beserta detail negara airline | Sukses |
| 18 | /select\_all\_number\_routes\_by\_airline | Mengambil semua data airline dan jumlah rute | Sukses |
| 19 | /select\_all\_number\_routes\_more\_than/<number> | Mengambil semua jumlah penerbangan yang lebih dari pilihan | Sukses |
| 20 | /select\_all\_airline\_in\_routes | Mengambil semua data airline yang ada pada route | Sukses |
| 21 | /select\_all\_airline\_source | Mengambil semua data airline dan asal penerbangan | Sukses |

Dari skenario uji coba yang diberikan pada Tabel 5.1, hasil uji coba antar muka sistem menunjukan 100% berhasil dilaksanakan. Rute yang ada berhasil diakses dan memberikan output sesuai dengan yang diharapkan, dan kondisi data pada kedua basis data ketika di sinkronisasi kembali adalah sama.

Pencatatan waktu respon dibagi menjadi tiga kategori tabel yang berbeda. Kategori pertama adalah mencatat waktu untuk rute yang melakukan kueri perubahan pada basis data. Rute yang termasuk dalam kategori ini adalah semua rute yang menjalankan kueri *insert*, *update* dan *delete*. Data adapter mengarahkan kueri ke basis data MySQL untuk mengeksekusi kueri pada kategori ini. Tabel 5.3 menunjukan waktu respon dari rute yang melakukan perubahan data. Data yang digunakan adalah data dari [28] tanpa adanya perubahan jumlah baris

Tabel 5.3 Hasil Waktu Respons Uji Coba Fungsionalitas untuk Perubahan Data

| **No** | **Nama Interfaces** | **Waktu Respons (ms)** | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **Rata-rata** |
| 1 | POST /insert\_routes | 313 | 59 | 75 | 49 | 58 | 110,8 |
| 2 | POST /delete\_routes\_by\_id | 259 | 232 | 256 | 257 | 271 | 255 |
| 3 | POST /update\_routes\_by\_id | 323 | 241 | 266 | 255 | 257 | 268,4 |
| 4 | POST /insert\_airline | 55 | 122 | 58 | 67 | 63 | 73 |
| 5 | POST /delete\_airline\_by\_id | 81 | 100 | 62 | 65 | 71 | 75,8 |
| 6 | POST /update\_airline\_by\_id | 81 | 78 | 81 | 80 | 72 | 78,4 |
| 7 | POST /insert\_airport | 88 | 69 | 54 | 33 | 60 | 60,8 |
| 8 | POST /delete\_airport\_by\_id | 90 | 91 | 87 | 143 | 345 | 151,2 |
| 9 | POST /update\_airport\_by\_id | 85 | 86 | 107 | 134 | 92 | 100,8 |

Kategori yan kedua adalah pengambilan data pada basis data SQL. Pada kategori ini, semua kueri yang berkaitan dengan pengambilan data, atau kueri *select*, akan diarahkan ke MySQL. Hal ini terjadi ketika proses sinkronisasi sedang berlangsung. Tabel 5.4 menunjukan catatan waktu respon untuk pengambilan data pada basis data MySQL.

Dan kategori yang terakhir adalah pengambilan basis data pada basis data NoSQL. Proses pengambilan data ini terjadi jika proses sinkronisasi sedang tidak berlangsung. Kueri select akan diarahkan ke HBase dan hasil uji coba dicatat pada Tabel 5.5.

Tabel 5.4 Waktu Respon Uji Fungsionalitas Proses Pengambilan Data pada Basis Data MySQL

| **No. Antarmuka** | **Nama Interfaces** | **Waktu Respons (s)** | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **Rata-rata** |
| 1 | GET /select\_all\_routes | 701 | 708 | 698 | 686 | 728 | 704,2 |
| 2 | GET /select\_route\_by\_id/<id> | 53 | 56 | 57 | 51 | 52 | 53,8 |
| 3 | GET /select\_all\_airline | 89 | 53 | 56 | 54 | 67 | 63,8 |
| 4 | GET /select\_airline\_by\_id/<id> | 18 | 28 | 30 | 12 | 22 | 22 |
| 5 | GET /select\_all\_airport | 126 | 129 | 169 | 129 | 139 | 138,4 |
| 6 | GET /select\_airport\_by\_id/<id> | 20 | 22 | 15 | 17 | 52 | 25,2 |
| 7 | GET /select\_routes\_country/<id> | 345 | 343 | 336 | 331 | 323 | 335,6 |
| 8 | GET /select\_all\_number\_routes\_by\_airline | 21340 | 21338 | 21308 | 21308 | 21340 | 21326,8 |
| 9 | GET /select\_all\_number\_routes\_more\_than/<number> | 46 | 49 | 50 | 66 | 55 | 53,2 |
| 10 | GET /select\_all\_airline\_in\_routes | 49 | 45 | 49 | 64 | 54 | 52,2 |
| 11 | GET /select\_all\_airline\_source | TLE | TLE | TLE | TLE | TLE | TLE |

Tabel 5.5 Waktu Respon Uji Fungsionalitas Proses Pengambilan Data pada Basis Data HBase

| **No. Antarmuka** | **Nama Interfaces** | **Waktu Respons (ms)** | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **Rata-rata** |
| 1 | GET /select\_all\_routes | 13873 | 13784 | 13711 | 13762 | 13772 | 13780,4 |
| 2 | GET /select\_route\_by\_id/<id> | 52 | 46 | 52 | 50 | 50 | 50 |
| 3 | GET /select\_all\_airline | 1148 | 1006 | 1004 | 0998 | 0998 | 1030,8 |
| 4 | GET /select\_airline\_by\_id/<id> | 84 | 94 | 47 | 50 | 45 | 64 |
| 5 | GET /select\_all\_airport | 2031 | 1990 | 2058 | 2024 | 2018 | 2024,2 |
| 6 | GET /select\_airport\_by\_id/<id> | 99 | 72 | 60 | 57 | 49 | 67,4 |
| 7 | GET /select\_routes\_country/<id> | 228 | 207 | 180 | 158 | 181 | 190,8 |
| 8 | GET /select\_all\_number\_routes\_by\_airline | 480 | 455 | 451 | 449 | 444 | 455,8 |
| 9 | GET /select\_all\_number\_routes\_more\_than | 208 | 237 | 216 | 229 | 222 | 222,4 |
| 10 | GET /select\_all\_airline\_in\_routes | 211 | 207 | 216 | 212 | 215 | 212,2 |
| 11 | GET /select\_all\_airline\_source | 3372 | 3367 | 3222 | 3302 | 3321 | 3316,8 |

Gambar 5.4 Grafik Perbandingan Waktu Respon Rata-rata Basis Data SQL (MySQL) dengan NoSQL (HBase)

Dari hasil uji coba yang dilakukan, terdapat beberapa kueri dengan perbedaan waktu respon yang signifikan seperti pada nomor antar muka 1, 8, dan 11. Pada nomor antar muka 11, waktu respon untuk pengambilan data pada MySQL melebihi dari batas waktu yang ditentukan atau TLE (*time limit exceeded*). Waktu respon untuk pengambilan data ke basis data SQL lebih cepat pada nomor antar muka 1, 3, 4, 5, 6, 9, dan 10. Waktu respon untuk pengambilan data pada NoSQL lebih cepat pada nomor antar muka 2, 7, 8, dan 11.

### Hasil Uji Coba Kapasitas dan Performa

Seperti yang dijelaskan pada sub bab sebelumnya, terdapat lima skenario yang digunakan untuk melakukan uji coba kapasitas dan performa. Hasil pengujian disimpan kedalam tabel Hasil uji coba yang telah dilakukan dijelaskan secara rinci pada sub bab berikut:

#### Perbandingan Jumlah Baris dalam Tabel

Pengujian ini dilakukan dengan menggunakan kumpulan data yang memiliki jumlah baris yang berbeda. Pengujian diawali dengan baris dengan jumlah 6.000 pada tabel Airplane, 7.000 pada tabel Airport dan 60.000 pada tabel Route. Jumlah baris bertambah sesuai dengan skenario. Waktu respon dicatat mulai dari awal proses sampai dengan proses inisialisasi telah berakhir. Tabel 5.6 menunjukan hasil sumber daya yang digunakan dan waktu respon saat proses inisialisasi data dari MySQL ke HBase.

Dari hasil pengujian, waktu respon yang dibutuhkan untuk melakukan inisialisasi meningkat secara konsisten, mulai dari 30,848 detik untuk skenario pertama dan naik bertahap sampai mencapai 56,185 pada skenario terakhir. Grafik peningkatan waktu respon ini dapat dilihat pada Gambar 5.3. Perubahan penggunaan memori pada RAM menunjukan angka perubahan yang tidak terlalu signifikan dengan persentase tertinggi sebesar 58,91%.

Tabel 5.6 Hasil Uji Kapasitas dan Performa pada Komputer Data Adapter (DB Converter) dengan Perbandingan Jumlah Baris

| **No** | **Jumlah Baris** | **Memori Terpakai** | **Processor yang terpakai** | **Waktu Transformasi (s)** |
| --- | --- | --- | --- | --- |
| 1 | Airplane : 6000  Airpot : 7000  Route : 60000 | 55,08% | 1: 58,0%  2: 58,1%  3: 35,3%  4: 56,4% | 30,848 |
| 2 | Airplane : 12000  Airpot : 14000  Route : 120000 | 57,12% | 1: 69,7%  2: 71,3%  3: 55,1%  4: 49,0% | 36,755 |
| 3 | Airplane : 18000  Airpot : 21000  Route : 180000 | 56,93% | 1: 71,4%  2: 59,2%  3: 69,0%  4: 74,0% | 43,355 |
| 4 | Airplane : 24000  Airpot : 28000  Route : 240000 | 55,20% | 1: 96,8%  2: 94,7%  3: 91,0%  4: 91,1% | 49,794 |
| 5 | Airplane : 30000  Airpot : 35000  Route : 300000 | 58,91% | 1: 65,6%  2: 80,5%  3: 71,4%  4: 64,1% | 56,185 |

Gambar 5.5 Grafik Waktu Transformasi Berdasarkan Perbandingan Jumlah Baris

#### Perbandingan Jumlah Tabel pada Basis Data

Pengujian dilakukan dengan menggunakan kumpulan data yang memiliki jumlah tabel berbeda. Tabel yang digunakan untuk setiap kali pengujian adalah berbeda, sesuai dengan skenario pada sub bab sebelumnya. Penambahan jumlah tabel dilakukan dengan jumlah baris yang sama. Waktu respon dicatat mulai dari awal proses sampai dengan proses inisialisasi telah berakhir. Tabel 5.7 menunjukan hasil sumber daya yang digunakan dan waktu respon saat proses inisialisasi data dari MySQL ke HBase.

Dari hasil pengujian, waktu respon yang dibutuhkan untuk melakukan inisialisasi adalah meningkat secara konsisten, namun memiliki selisih yang lebih besar jika dibandingan pengujian dengan perbedaan jumlah baris. Waktu transformasi tercepat adalah 31,982 detik pada jumlah tabel tiga dan waktu transformasi terlama adalah 139,741 detik. Grafik peningkatan waktu respon ini dapat dilihat pada Gambar 5.6. Penggunaan RAM terbanyak terjadi pada jumlah tabel 15 dengan kondisi terpakai sejumlah 60,77%.

Tabel 5.7 Hasil Uji Kapasitas dan Performa pada Komputer Data Adapter (DB Converter) dengan Perbandingan Jumlah Tabel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Jumlah Tabel** | **Memori (MB)** | **Processor yang terpakai** | **Waktu transformasi (s)** |
| 1 | 3 | 52,26% | 1: 80,0%  2: 77,1%  3: 56,1%  4: 64,9% | 31,982 |
| 2 | 6 | 52,78% | 1: 55,7%  2: 67,8%  3: 70,1%  4: 67,1% | 58,028 |
| 3 | 9 | 54,44% | 1: 70,3%  2: 63,4%  3: 60,3%  4: 61,8% | 84,482 |
| 4 | 12 | 57,48% | 1: 61,9%  2: 63,6%  3: 52,9%  4: 53,6% | 112,031 |
| 5 | 15 | 60,77% | 1: 40,4%  2: 44,4%  3: 50,3%  4: 30,9% | 139,741 |

Gambar 5.6 Grafik Waktu Transformasi Berdasarkan Perbandingan Jumlah Tabel

#### Perbandingan Jumlah Kueri Insert, Update dan Delete

Pengujian yang pertama pada sub bab ini adalah dengan mengeksekusi sejumlah kueri *insert* pada sistem. Selanjutnya, sistem akan melakukan proses *patching* dengan mentransformasikan semua kueri yang dieksekusi pada basis data SQL ke basis data NoSQL. Untuk transformasi sejumlah kueri insert, hasil uji coba ditunjukan pada Tabel 5.8.

Tabel 5.8 Hasil Uji Kapasitas dan Performa pada Komputer Data Adapter (DB Converter) dengan Perbandingan Jumlah Kueri Insert

| **No** | **Jumlah Kueri *Insert*** | **Memori (MB)** | **Processor yang terpakai** | **Waktu transformasi (s)** |
| --- | --- | --- | --- | --- |
| 1 | 1000 | 43,84% | 1: 64,1%  2: 85,8%  3: 28,1%  4: 53,6% | 7,913 |
| 2 | 2000 | 52,28% | 1: 63,4%  2: 54,0%  3: 60,3%  4: 68,6% | 10,630 |
| 3 | 3000 | 53,51% | 1: 52,1%  2: 59,6%  3: 38,8%  4: 35,5% | 13,021 |
| 4 | 4000 | 54,07% | 1: 52,3%  2: 54,2%  3: 40,3%  4: 51,0% | 16,190 |
| 5 | 5000 | 53,10% | 1: 61,3%  2: 54,4%  3: 54,6%  4: 41,0% | 18,909 |

Hal yang sama dilakukan untuk proses transformasi sejumlah kueri *update*. Hasil uji coba transformasi sejumlah kueri update dapat dilihat pada Tabel 5.9.

Tabel 5.9 Hasil Uji Kapasitas dan Performa pada Komputer Data Adapter (DB Converter) dengan Perbandingan Jumlah Kueri Update

| **No** | **Jumlah Kueri *Update*** | **Memori (MB)** | **Processor yang terpakai** | **Waktu transformasi (s)** |
| --- | --- | --- | --- | --- |
| 1 | 1000 | 54,81% | 1: 21,2%  2: 24,8%  3: 17,4%  4: 21,4% | 41,304 |
| 2 | 2000 | 53,97% | 1: 18,2%  2: 14,4%  3: 11,8%  4: 18,1% | 75,050 |
| 3 | 3000 | 55,53% | 1: 19,0%  2: 23,1%  3: 17,6%  4: 24,1% | 107,772 |
| 4 | 4000 | 56,26% | 1: 30,3%  2: 18,7%  3: 18,2%  4: 29,2% | 135,771 |
| 5 | 5000 | 54,42% | 1: 13,1%  2: 19,2%  3: 14,5%  4: 13,3% | 162,906 |

Operasi terakhir yang diuji adalah melakukan transformasi untuk sejumlah kueri *delete*. Hasil uji coba kapasitas dan performa untuk perbandingan jumlah kueri *delete* dapat dilihat pada Tabel 5.10.

Tabel 5.10 Hasil Uji Kapasitas dan Performa pada Komputer Data Adapter (DB Converter) dengan Perbandingan Jumlah Kueri Delete

| **No** | **Jumlah Kueri *Delete*** | **Memori (MB)** | **Processor yang terpakai** | **Waktu transformasi (s)** |
| --- | --- | --- | --- | --- |
| 1 | 1000 | 48,21% | 1: 9,9%  2: 13,2%  3: 3,3%  4: 29,0% | 8,229 |
| 2 | 2000 | 53,32% | 1: 50,6%  2: 40,0%  3: 54,9%  4: 68,0% | 11,423 |
| 3 | 3000 | 52,78% | 1: 51,0%  2: 51,0%  3: 59,1%  4: 36,4% | 15,029 |
| 4 | 4000 | 55,24% | 1: 40,8%  2: 39,2%  3: 43,9%  4: 44,4% | 18,520 |
| 5 | 5000 | 54,03% | 1: 54,3%  2: 41,2%  3: 66,2%  4: 68,4% | 22,250 |

Dari hasil percobaan, waktu yang dibutuhkan untuk proses transformasi pada operasi insert dan delete memiliki nilai yang hampir sama. Namun untuk proses transformasi operasi update memiliki waktu yang lebih lama. Grafik perbandingan masing-masing operasi untuk setiap perbandingan jumlah kueri dapat dilihat pada Gambar 5.7.

Gambar 5.7 Perbandingan Waktu Transformasi pada Transformasi Kueri Insert, Update dan Delete

### Evaluasi

Berdasarkan hasil uji coba, sistem dapat berjalan sesuai dengan skenario pada sub bab 5.3. Dari pengujian fungsionalitas, semua fungsi dapat berjalan sesuai dengan harapan. Namun, untuk proses pengambilan data pada MySQL, terdapat satu antar muka dengan waktu respon yang melebih batas. MySQL tidak cukup tangguh untuk menangani kueri yang membutuhkan pengambilan data ke lebih dari satu tabel pada tabel yang memiliki jumlah data yang banyak. Disisi lain, HBase mampu menangani semua permintaan pemrosesan kueri dari setiap antar muka yang diujicobakan.

Dari sisi kapasitas dan performa, untuk semua skenario yang diujikan, jumlah persentase sumber daya memori RAM yang digunakan cenderung konstan berkisar pada angka 40%-60%. Sumber daya processor yang digunakan menghasilkan angka yang bervariasi. Rata-rata persentase processor yang digunakan untuk proses inisialisasi adalah sekitar 60%. Proses transformasi kueri mengabiskan rata-rata sumber daya processor lebih sedikit yaitu 53,3% untuk kueri *insert*, 19,05% untuk kueri *update* dan 42,9% untuk kueri *delete*. Proses transformasi kueri *update* memiliki angka persentase paling kecil, berkebalikan dengan waktu transfomasinya yang paling lama.

# 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.

## Kesimpulan

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

1. Data adapter dapat digunakan sebagai sistem untuk mensinkronisasi antara basis data SQL dan basis data NoSQL. Dari hasil pengujian, proses sinkronisasi berhasil bekerja 100% sukses dan kedua basis data memiliki data yang sama. Dengan menggunakan pendekatan kueri *direct access*, aplikasi tetap dapat mengakses basis data saat proses sinkronisasi sedang berlangsung.
2. Berdasarkan hasil pengujian, untuk setiap peningkatan jumlah data atau kueri, persentase penggunaan memori RAM tidak mengalami peningkatan yang signifikan. Hal ini menunjukan jika sistem data adapter tidak membutuhkan kapasitas RAM yang terlalu besar. Sedangkan untuk persentase processor, saat proses sinkronisasi sistem mengambil sumber daya yang cukup besar pada server basis data NoSQL.
3. Basis data NoSQL yaitu Apache HBase, memiliki kemampuan yang lebih baik jika dibandingkan dengan basis data relasional, yaitu MySQL, dalam hal melakukan kueri analisis pada banyak tabel dengan jumlah baris yang banyak. Hal ini ditunjukan dengan waktu respon yang dibutuhkan untuk mengambil data pada antar muka nomor 8 dan 11 pada Gambar 5.4 menunjukan waktu respon HBase lebih cepat dibandingkan dengan MySQL, bahkan pada nomor 11 respon MySQL melewati batas waktu. Hal ini menunjukan jika NoSQL layak digunakan untuk menangani permasalahan *big data*.

## Saran

Saran yang diberikan untuk pengembangan lebih lanjut terhadap sistem ini adalah sebagai berikut:

1. Mekanisme data adapter perlu dikembangkan lagi dengan melakukan perbandingan dengan tipe basis data NoSQL lainnya.

# DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | G. Audin, “No Jitter,” UBM, 2017. [Online]. Available: http://www.nojitter.com/post/240170228/the-network-impact-of-big-data. [Diakses 03 April 2017]. |
| [2] | Oracle, “Oracle,” Oracle, [Online]. Available: https://www.oracle.com/mysql/index.html. [Diakses 13 April 2017]. |
| [3] | COUCHBASE, “COUCHBASE,” COUCHBASE , 2017. [Online]. Available: https://www.couchbase.com/nosql-resources/why-nosql. [Diakses 13 April 2017]. |
| [4] | R. S. Samiddha Mukherjee, “Big Data – Concepts, Applications, Challenges,” *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 5, no. 2, p. 66, 2016. |
| [5] | S. Matteson, “TechRepublic,” CBS Interactive., 2017. [Online]. Available: http://www.techrepublic.com/blog/big-data-analytics/big-data-basic-concepts-and-benefits-explained/. [Diakses 2017 Mei 10]. |
| [6] | J. Ellingwood, “Digital Ocean,” DigitalOcean™ Inc., 28 September 2016. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-big-data-concepts-and-terminology. [Diakses 10 Mei 2017]. |
| [7] | SAS Institute Inc., “SAS US,” SAS Institute Inc., 2017. [Online]. Available: https://www.sas.com/en\_us/insights/big-data/what-is-big-data.html. [Diakses 12 Mei 2017]. |
| [8] | P. K. Pankaj Sareen, “NOSQL DATABASE AND ITS COMPARISON WITH SQL DATABASE,” *International Journal of Computer Science & Communication Networks,* vol. 5, pp. 293-298. |
| [9] | G. C. Everest, Database Management: Objectives, System Functions, and Administration, New York: McGraw-Hill Companies, 1986. |
| [10] | P. ". D. D. Christensson, “Techterms.com,” Sharpened Production , 16 July 2016. [Online]. Available: https://techterms.com/definition/relational\_database.. [Diakses 28 March 2017]. |
| [11] | M. Rouse, “TechTarget.com,” TechTarget, 10 2011. [Online]. Available: http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL. [Diakses 20 10 2016]. |
| [12] | mongoDB, “mongoDB,” MongoDB, Inc., [Online]. Available: https://www.mongodb.com/nosql-explained. [Diakses 20 10 2016]. |
| [13] | B. M. Sasaki, “neo4j,” Neo Technology, Inc., 2017. [Online]. Available: https://neo4j.com/blog/why-nosql-databases/. [Diakses 12 Mei 2017]. |
| [14] | J. Z. C.-H. L. S.-C. C. C.-H. H. C. M.-F. J. Y.-C. C. Ying-Ti Liao, “Data adapter for querying and transformation between SQL and NoSQL database,” *Elsevier,* vol. 65, pp. 111-121, 2016. |
| [15] | MySQL, “MySQL,” Oracle Corporation, [Online]. Available: https://www.mysql.com/about/. [Diakses 20 10 2016]. |
| [16] | P. Lutus, “arachnoid.com,” 2012. [Online]. Available: https://arachnoid.com/MySQL/. [Diakses 28 Marh 2017]. |
| [17] | INTELLIPAAT.COM, “INTELLIPAAT.COM,” 2017. [Online]. Available: https://intellipaat.com/tutorial/hbase-tutorial/introduction/. [Diakses 20 04 2017]. |
| [18] | H. Inc., “HORTONWORKS,” Hortonworks Inc., 2016. [Online]. Available: http://hortonworks.com/apache/hbase/. [Diakses 20 11 2016]. |
| [19] | “Guru99,” 2017. [Online]. Available: http://www.guru99.com/hbase-architecture-data-flow-usecases.html. [Diakses 03 March 2017]. |
| [20] | Apache HBase Team, “Apache HBase Reference Guide,” The Apache Software Foundation, 11 April 2017. [Online]. Available: http://hbase.apache.org/book.html#\_read\_hbase\_shell\_commands\_from\_a\_command\_file. [Diakses 20 May 2017]. |
| [21] | “HORTONWORKS,” Hortonworks Inc, [Online]. Available: https://hortonworks.com/hadoop-tutorial/bi-apache-phoenix-odbc/. [Diakses 6 April 2017]. |
| [22] | Apache Software Foundation, “Apache Phoenix,” Apache Software Foundation, [Online]. Available: https://phoenix.apache.org/who\_is\_using.html. [Diakses 6 April 2017]. |
| [23] | S. Srungarapu, “Cloudera Engineering Blog,” Cloudera, Inc, 6 Mei 2015. [Online]. Available: https://blog.cloudera.com/blog/2015/05/apache-phoenix-joins-cloudera-labs/. [Diakses 6 Maret 2017]. |
| [24] | T. Point, “Tutorialspoint.com,” Tutorials Point (I) Pvt. Ltd., 2017. [Online]. Available: https://www.tutorialspoint.com/python/python\_overview.htm. [Diakses 14 April 2017]. |
| [25] | Python, “Python.org,” Python Software Foundation, 2017. [Online]. Available: https://www.python.org/. [Diakses 14 April 2017]. |
| [26] | K. Das, “Pym,” pymbook, 2015. [Online]. Available: http://pymbook.readthedocs.io/en/latest/flask.html. [Diakses 2 Mei 2017]. |
| [27] | A. Ronacher, “Flask,” Armin Ronacher and contributors, 2015. [Online]. Available: http://flask.pocoo.org/. [Diakses 2 Mei 2017]. |
| [28] | Contentshare, “OpenFlights.org,” Contentshare, 2017. [Online]. Available: https://openflights.org/data.html. [Diakses 20 04 2017]. |

# LAMPIRAN A KODE SUMBER

Kode Sumber 7.1 Kode Sumber Sinkronisasi Utama

|  |
| --- |
| **import** **paramiko**  **import** **string**  **import** **webbrowser**  **import** **time**, **os**, **re**  **from** **datetime** **import** datetime  **import** **MySQLdb**  **import** **csv**, **sys**  **import** **subprocess**  **import** **conv\_phoenix**  **import** **c\_db**  **from** **ConfigParser** **import** SafeConfigParser  sync\_time\_init=**0**  allow\_q = ["insert", "delete", "update"]  **def** **getLastSync**(data):  host = data['host']  username = data['username']  password = data['password']  db\_name = data['db\_name']  db = MySQLdb.connect(host=host, user=username, passwd=password, db=db\_name)  cursor = db.cursor()  sql = "SELECT \* FROM log\_sinkronisasi ORDER BY waktu DESC LIMIT 1"  **try** :  cursor.execute(sql)  row = cursor.fetchone()  get\_date = row[**1**]  **return** get\_date  **except** :  **print** "Error, unable to fetch Last Sync data"  **return** **0**  db.close()  **def** **dumpMysqlTableToCsv**(db\_data, tabel\_name):  data = db\_data  host = data['host']  username = data['username']  password = data['password']  db\_name = data['db\_name']  # dump dari MySQL ke csv  db = MySQLdb.connect(host=host, user=username, passwd=password, db=db\_name)  cursor = db.cursor()  sql = "SELECT \* FROM {0}".format(tabel\_name)  file\_output = "file/csv/fetch\_all\_{0}.csv".format(tabel\_name)  **print** "sql: ", sql  **print** "file\_output: ", file\_output  **try** :  cursor.execute(sql)  result = cursor.fetchall()  c = csv.writer(open(file\_output, "wb"))  **for** row **in** result:  c.writerow(row)  **return** **1**  **except** :  **print** "Error, unable to fetch data from MySQL (",host,")"  **return** **0**  **finally**:  db.close()  **def** **initiateDBtoHBase**(data\_db):  get\_table = c\_db.getTableToSync()  list\_table = get\_table['table'].replace(" ", "").split(',')  **for** table **in** list\_table:  d = dumpMysqlTableToCsv(data\_db, table)  **if** (d) :  **print** "Berhasil, tabel ", table," berhasil di dump ke fetch\_all\_",table,".csv"  **return** list\_table  **def** **insertNewSync**(sync\_db, mysql\_db, hbase\_db):  # DB Log sinkronisasi  host = sync\_db['host']  username = sync\_db['username']  password = sync\_db['password']  db\_name = sync\_db['db\_name']  # DB MySQL  host\_mysql = mysql\_db['host']  # DB HBase  host\_hbase = hbase\_db['host']  ts = time.time()  timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  **print** 'time stamp: ', timestamp  db = MySQLdb.connect(host=host, user=username, passwd=password, db=db\_name)  cursor = db.cursor()  sql = "INSERT INTO log\_sinkronisasi (waktu, ip\_src, ip\_dst, status) VALUES ('{0}', '{1}', '{2}', '{3}')".format(timestamp,host\_mysql,host\_hbase,**0**)  **print** 'sql : ', sql  **try** :  cursor.execute(sql)  db.commit()  **if** cursor.lastrowid :  **return** cursor.lastrowid  **else** :  **return** **0**  **except** :  db.rollback()  **print** "Error, unable to record current sync time!"  **return** **0**  **finally** :  db.close  **def** **updateStatusSinkronisasi**(sync\_db, id\_log, status):  # DB Log sinkronisasi  host = sync\_db['host']  username = sync\_db['username']  password = sync\_db['password']  db\_name = sync\_db['db\_name']  db = MySQLdb.connect(host=host, user=username, passwd=password, db=db\_name)  cursor = db.cursor()  sql = "UPDATE log\_sinkronisasi SET status='{0}' WHERE id\_log={1}".format(status, id\_log)  **try** :  cursor.execute(sql)  db.commit()  **return** **1**  **except** :  db.rollback()  **print** "Error, unable to update log status!"  **return** **0**  **finally** :  db.close  **def** **isDateFormat**(input):  **try**:  time.strptime(input, '%y%m%d')  **return** True  **except** **ValueError**:  **return** False  **def** **isTimeFormat**(input):  **try**:  time.strptime(input, '%H:%M:%S')  **return** True  **except** **ValueError**:  **return** False  **def** **convertDateTimeFormat**(input):  result = datetime.strptime(input, '%y%m%d %H:%M:%S')  **return** result  **def** **remoteServerMysqlDb**():  data = c\_db.getSshAccess()  host = data['host']  username = data['username']  password = data['password']  ssh = paramiko.SSHClient()  ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  ssh.connect(host, username=username, password=password)  **return** ssh  **def** **mainGetFile**() :  ftp = ssh.open\_sftp()  ftp.get('/var/log/mysql/log\_query.log', 'file/get\_log\_query.log')  ftp.close()  # webbrowser.open\_new(os.getcwd()+'/get\_log\_query.log')  **print** ("FILE GOT!")  **def** **initiateTransformMysqlToHbase**(mysql\_db, hbase\_db, sync\_db):  **print** "Belum pernah sinkronisasi"  init = initiateDBtoHBase(mysql\_db)  **if** (init) :  **print** "Data berhasil di export!"  **print** "Inisialisasi data awal.."  **print** "Mentransformasi data ke HBase..."  create\_tabel = "psql.py file/flight\_create\_2.sql"  **try** :  result\_create\_tabel = subprocess.check\_output([create\_tabel], shell=True)  **if** (result\_create\_tabel):  **print** "Tabel berhasil dibuat di HBase"  **for** table **in** init:  hbase\_table = table.upper()  phoenix\_cmd = "psql.py -t {0} file/csv/fetch\_all\_{1}.csv".format(hbase\_table, table)  **try** :  result\_phoenix = subprocess.check\_output([phoenix\_cmd], shell=True)  **if** (result\_phoenix):  **print** "Tabel {0} berhasil diimport ke HBase...".format(table)  **else** :  **print** "Tabel {0} gagal diimport ke HBase..".format(table)  **except** **Exception** **as** e:  **print**(e)  **else**:  **print** "Gagal membuat tabel di HBase.. "    **except** **Exception** **as** e:  **print**(e)  update\_status\_sync = updateStatusSinkronisasi(sync\_db, sync\_time\_init, **1**)  **if** (update\_status\_sync) :  **print** 'Proses inisialisasi selesai..'  **if** \_\_name\_\_ == '\_\_main\_\_':  data\_mysql = c\_db.getConfMysqlDb()  data\_hbase = c\_db.getConfHbaseDb()  data\_sync\_db = c\_db.getConfSyncLogDb()  data\_ssh = c\_db.getSshAccess()  ssh = paramiko.SSHClient()  ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  ssh.connect(data\_ssh['host'], username=data\_ssh['username'], password=data\_ssh['password'])    # print "1: get file log"  # print "2: Syncronize MySQL to HBase"  # c = input('Select an option :')    **try** :  act = int(sys.argv[**1**])  **if** (act==**1**):  c = **1**  **elif** (act==**2**):  c = **2**  **else** :  c = -**1**  **print** "Parameter salah ..."  **print** ">> 1: get file log"  **print** ">> 2: Syncronize MySQL to HBase"  **except** :  **print** "Sintaks salah!"  c = -**1**  **if** c==**1** :  mainGetFile()  **if** c==**2** :  start\_time = time.time()  last\_sync = getLastSync(data\_sync\_db)  present = datetime.now()  **print** 'sinkron terakhir: ',last\_sync  **print** 'waktu sekarang: ', present  sync\_time\_init = insertNewSync(data\_sync\_db, data\_mysql, data\_hbase)  **if** (**not** last\_sync) :  initiateTransformMysqlToHbase(data\_mysql, data\_hbase, data\_sync\_db)  **elif** (last\_sync) :  **print** "Sinkronisasi terakhir: ", last\_sync  # print last\_sync > present  do\_patching = False  # Mengambil data DELETE dari remote server, kemudian di simpan di file local  **print** "Mengambil data Query dari log ..."  ftp = ssh.open\_sftp()  file = ftp.file('/var/log/mysql/log\_query.log', 'r')  f\_list\_query = open('file/list\_all\_query.sql', 'w')  date\_tmp = ''  count\_kueri = **0**  **print** "Cek aksi INSERT, UPDATE, DELETE pada log..."  **for** line **in** file :  kueri\_line = line  get\_waktu = kueri\_line.split()    tgl = str(get\_waktu[**0**])  **if** (len(get\_waktu)>**1**):  wkt = str(get\_waktu[**1**])  **if** (isDateFormat(tgl)) :  **if** (isTimeFormat(wkt)) :  date\_time\_join = tgl + ' ' + wkt  date\_tmp = convertDateTimeFormat(date\_time\_join)  **if** (date\_tmp > last\_sync) :  **if** 'Query' **in** line :  **for** a\_q **in** allow\_q :  q = re.split(r'\t+', kueri\_line)  **if** (a\_q **in** q[**2**]) **or** (a\_q.upper() **in** q[**2**]) :  kueri = q[**2**].replace("**\n**", "")  split\_kueri = kueri.split()  count\_kueri+=**1**  # Cek jika sintaks adalah UPDATE  **if** (split\_kueri[**0**].upper()=="INSERT"):  **print** "[sync] INSERT ", split\_kueri[**2**]  new\_kueri = kueri.replace("INSERT", "UPSERT").replace("insert", "UPSERT") + ";**\n**";  **elif** (split\_kueri[**0**].upper()=="UPDATE"):  **print** "[sync] UPDATE ", split\_kueri[**1**]  new\_kueri = conv\_phoenix.update\_to\_upsert(kueri)  **if** (new\_kueri==**0**) :  **print** "[err] Table not int List! "  **elif** (split\_kueri[**0**].upper()=="DELETE") :  **print** "[sync] DELETE ", split\_kueri[**1**]  new\_kueri = kueri.replace("INSERT", "UPSERT").replace("insert", "UPSERT") + ";**\n**";    **if** (new\_kueri!=**0**):  f\_list\_query.write(new\_kueri)  do\_patching = True  **if** (count\_kueri ==**0**) :  **print** "Tidak ada data yang transformasikan!"  updateStatusSinkronisasi(data\_sync\_db, sync\_time\_init, **1**)  f\_list\_query.close()  ftp.close()  # Menjalankan DELETE, INSERT, dan UPDATE  **if** (do\_patching):  **print** "Sinkronisasi, DELETE, INSERT dan UPDATE file on progress sync to HBase..."  phoenix\_cmd = "psql.py file/list\_all\_query.sql"  res\_phoenix =''  **try** :  res\_phoenix = subprocess.check\_output([phoenix\_cmd], shell=True)  **except** **Exception** **as** e:  **print**(e)  **if** (res\_phoenix) :  insert\_time = updateStatusSinkronisasi(data\_sync\_db, sync\_time\_init, **1**)  **if** (insert\_time) :  **print** 'Update status sinkronisasi berhasil'  **print** 'Sync berhasil dilakukan.'  duration = time.time() - start\_time  **print** "Durasi sinkronisai : %s seconds" % (duration)  **print** "Done"  **if** c==**3** :  **print** "Key terlarang!"  getconf = c\_db.getConfMysqlDb()  **print** getconf  **else** :  **print** "EXIT!";  ssh.close() |

Kode Sumber 7.2 Fungsi Perhitungan LCS Paralel

|  |
| --- |
| 1. **for** (k = 1; k <= numberOfProcess / alpa; k++) { 2. **if** (process == 0) { 3. **int** rowStart = ((alpa \* (k - 1) \* lengthOfX) / numberOfProcess); 4. **int** rowEndPrevProcess = ((alpa \* (k - 2) \* lengthOfX) / numberOfProcess); 5. **int** rowEnd = (alpa \* k \* lengthOfX) / numberOfProcess; 7. **if** (rowStart != 0) rowStart = rowStart + 1; 9. **if** (k != 1 && rowEnd == 0 && rowStart == 0) { 10. rowStart = rowStart + 1; 11. } 12. **if** (k != 1 && rowEndPrevProcess == 0 && rowStart == 0) { 13. rowStart = rowStart + 1; 14. } 16. **int** coloumnStart = (process) \* lengthOfY / numberOfProcess; 17. **if** (coloumnStart != 0) coloumnStart = coloumnStart + 1; 19. **int** coloumnEnd = (process + 1) \* lengthOfY / numberOfProcess; 20. **for** (rowCalculate = rowStart; rowCalculate <= rowEnd; rowCalculate++) { 21. numberOf*LCS*Character[rowCalculate].resize(sizeForY); 22. **for** (coloumnCalculate = coloumnStart; coloumnCalculate <= coloumnEnd; coloumnCalculate++) { 24. **if** (rowCalculate == 0 || coloumnCalculate == 0) { 25. numberOf*LCS*Character[rowCalculate][coloumnCalculate] = 0; 26. } **else** **if** (X[rowCalculate - 1] == Y[coloumnCalculate - 1]) { 27. numberOf*LCS*Character[rowCalculate][coloumnCalculate] =  numberOf*LCS*Character[rowCalculate - 1][coloumnCalculate - 1] + 1; 28. } **else** { 29. numberOf*LCS*Character[rowCalculate][coloumnCalculate] =  max(numberOf*LCS*Character[rowCalculate -1][coloumnCalculate],  numberOf*LCS*Character[rowCalculate][coloumnCalculate - 1]); 30. } 31. } 33. } 34. dest = process + 1; 35. **int** rowSendProcess0, coloumnSendProcess0; 36. **for** (rowSendProcess0 = rowStart; rowSendProcess0 <= rowEnd; rowSendProcess0++) { 37. coloumnSendProcess0 = (dest \* lengthOfY / numberOfProcess); 39. MPI\_Send( & numberOf*LCS*Character[rowSendProcess0][coloumnSendProcess0], 1, MPI\_INT, dest,  0, MPI\_COMM\_WORLD); 40. } 42. } **else** **if** (process > 0) { 43. **int** finish = 0; 44. **int** source = process - 1; 45. dest = process + 1; 46. **int** coloumnReceive, rowReceive, coloumnSend, rowSend; 47. **int** finishReceived = 0; 49. **int** firstColoumn = ((process) \* lengthOfY / numberOfProcess); 50. **int** rowStart = ((alpa \* (k - 1) \* lengthOfX) / numberOfProcess); 51. **int** rowEndPrevProcess = ((alpa \* (k - 2) \* lengthOfX) / numberOfProcess); 53. **int** coloumnStart = (process) \* lengthOfY / numberOfProcess; 54. **int** coloumnEnd = (process + 1) \* lengthOfY / numberOfProcess; 56. **int** rowEnd = (alpa \* k \* lengthOfX) / numberOfProcess; 57. **int** coloumnPreviousProcess = (process - 1) \* lengthOfY / numberOfProcess; 58. **if** (rowStart != 0) { 59. rowStart = rowStart + 1; 60. } 61. **if** (k != 1 && rowEnd == 0 && rowStart == 0) rowStart = rowStart + 1; 62. **if** (k != 1 && rowStart == 0 && rowEndPrevProcess == 0) rowStart = rowStart + 1; 64. **if** (coloumnStart != 0) { 65. coloumnStart = coloumnStart + 1; 66. } 68. **int** colStartForProcess1; 69. colStartForProcess1 = (1 \* lengthOfY / numberOfProcess); 70. **if** ((process \* lengthOfY / numberOfProcess) != colStartForProcess1) { 71. coloumnReceive = firstColoumn - coloumnPreviousProcess; 72. } **else** { 73. coloumnReceive = coloumnStart - 1; 74. } 76. **for** (rowReceive = rowStart; rowReceive <= rowEnd; rowReceive++) { 77. numberOf*LCS*Character[rowReceive].resize(sizeForY); 78. MPI\_Recv( & numberOf*LCS*Character[rowReceive][coloumnReceive], 1, MPI\_INT, source, 0,  MPI\_COMM\_WORLD, & status); 79. numberOf*LCS*Character[rowReceive][coloumnReceive - coloumnReceive] =  numberOf*LCS*Character[rowReceive][coloumnReceive]; 80. finishReceived = 1; 81. } 82. **if** (finishReceived == 1) { 83. **for** (rowCalculate = rowStart; rowCalculate <= rowEnd; rowCalculate++) { 84. numberOf*LCS*Character[rowCalculate].resize(sizeForY); 85. **for** (coloumnCalculate = coloumnStart; coloumnCalculate <= coloumnEnd;  coloumnCalculate++) { 86. **if** (rowCalculate == 0 || coloumnCalculate == 0) { 87. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] = 0; 88. } **else** **if** (X[rowCalculate - 1] == Y[coloumnCalculate - 1]) { 89. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] =  numberOf*LCS*Character[rowCalculate - 1][coloumnCalculate - 1 - firstColoumn] + 1; 90. } **else** { 91. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] =  max(numberOf*LCS*Character[rowCalculate - 1][coloumnCalculate - firstColoumn],  numberOf*LCS*Character[rowCalculate][coloumnCalculate - 1 - firstColoumn]); 92. } 93. rowIndex = rowCalculate; 94. coloumnIndex = coloumnCalculate - firstColoumn; 95. } 97. }; 98. coloumnSend = coloumnEnd - firstColoumn; 99. **if** (process != numberOfProcess - 1) { 100. **for** (rowSend = rowStart; rowSend <= rowEnd; rowSend++) { 101. **http://www.fileformat.info/info/unicode/font/arial_unicode_ms/u21AA.png**                    MPI\_Send( & numberOf*LCS*Character[rowSend][coloumnSend], 1, MPI\_INT, dest, 0,  MPI\_COMM\_WORLD); 102. } 103. } 104. } 105. } 106. finishedCalculating = 1; |

Kode Sumber 7.3 Fungsi Mencetak LCS

|  |
| --- |
| 1. **if** (finishedCalculating == 1) { 2. **if** (process == numberOfProcess - 1) { 3. **int** destBacktrack, coloumnStartBacktrack, coloumnEndBacktrack; 4. point valueForBacktrack; 5. **int** firstColoumn = ((process)\*lengthOfY / numberOfProcess); 7. destBacktrack = process - 1; 9. coloumnStartBacktrack =   (((process)\*lengthOfY / numberOfProcess) + 1) - firstColoumn; 10. coloumnEndBacktrack =   ((process + 1) \* lengthOfY / numberOfProcess) - firstColoumn; 11. **int** index = numberOf*LCS*Character[rowIndex][coloumnIndex]; 12. **int** initialIndex*LCS* = index - 1; 13. **int** lengthOf*LCS* = index; 15. vector<**char**> character*LCS*; 16. character*LCS*.resize(index); 18. **int** rowBacktrack = rowIndex, coloumnBacktrack = coloumnIndex; 19. **int** originalColoumnIndexY = coloumnBacktrack + firstColoumn; 20. **while** (coloumnBacktrack >= coloumnStartBacktrack  && coloumnBacktrack <= coloumnEndBacktrack) { 21. **if** (X[rowBacktrack - 1] == Y[originalColoumnIndexY - 1]) { 22. character*LCS*[index - 1] = X[rowBacktrack - 1]; 23. originalColoumnIndexY--; 24. rowBacktrack--; 25. coloumnBacktrack--; 26. index--; 27. } 28. **else** **if** (numberOf*LCS*Character[rowBacktrack - 1][coloumnBacktrack] > numberOf*LCS*Character[rowBacktrack][coloumnBacktrack - 1]) { 29. rowBacktrack--; 30. } 31. **else** { 32. coloumnBacktrack--; 33. originalColoumnIndexY--; 34. } 35. } 36. valueForBacktrack.rowForBacktrack = rowBacktrack; 37. valueForBacktrack.coloumnForBacktrack = coloumnBacktrack; 38. valueForBacktrack.index*LCS*ForBacktrack = index; 39. valueForBacktrack.originalColoumnBacktrack = originalColoumnIndexY; 40. **int** endIndex*LCS* = index; 42. MPI\_Send(&valueForBacktrack, 4, MPI\_INT, destBacktrack, 3, MPI\_COMM\_WORLD); 44. MPI\_Send(&initialIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 45. MPI\_Send(&endIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 47. MPI\_Send(&lengthOf*LCS*, 1, MPI\_INT, 0, 5, MPI\_COMM\_WORLD); 48. **for** (**int** sendCharacter = endIndex*LCS*; sendCharacter <= initialIndex*LCS*;  sendCharacter++) { 49. MPI\_Send(&character*LCS*[sendCharacter], 1, MPI\_CHAR, 0, 4, MPI\_COMM\_WORLD); 50. } 51. } 52. **if** (process < numberOfProcess - 1) { 54. **int** sourceBacktrackProcess, destBacktrackProcess; 55. **int** coloumnStartBacktrack, coloumnEndBacktrack; 56. **int** firstColoumn = ((process)\*lengthOfY / numberOfProcess); 57. point valueForBacktrack; 59. sourceBacktrackProcess = process + 1; 60. destBacktrackProcess = process - 1; 62. MPI\_Recv(&valueForBacktrack, 4, MPI\_INT, sourceBacktrackProcess, 3,  MPI\_COMM\_WORLD, &status); 63. **if** (process != 0) { 64. coloumnStartBacktrack =   (((process)\*lengthOfY / numberOfProcess) + 1) - firstColoumn; 65. coloumnEndBacktrack =   ((process + 1) \* lengthOfY / numberOfProcess) - firstColoumn; 66. **int** rowBacktrack, coloumnBacktrack, indexOf*LCS*; 67. **int** initialIndex*LCS*, endIndex*LCS*, originalColoumnIndexY; 68. rowBacktrack = valueForBacktrack.rowForBacktrack; 69. coloumnBacktrack = coloumnEndBacktrack; 70. indexOf*LCS* = valueForBacktrack.index*LCS*ForBacktrack; 71. originalColoumnIndexY = valueForBacktrack.originalColoumnBacktrack; 73. vector<**char**> character*LCS*; 74. character*LCS*.resize(indexOf*LCS*); 75. initialIndex*LCS* = indexOf*LCS* - 1; 76. **int** backtrack = 0; 77. **while** (coloumnBacktrack >= coloumnStartBacktrack &&  coloumnBacktrack <= coloumnEndBacktrack) { 78. **if** (indexOf*LCS* <= 0) 79. **break**; 80. **if** (X[rowBacktrack - 1] == Y[originalColoumnIndexY - 1]) { 81. character*LCS*[indexOf*LCS* - 1] = X[rowBacktrack - 1]; 82. rowBacktrack--; 83. coloumnBacktrack--; 84. indexOf*LCS*--; 85. originalColoumnIndexY--; 86. } 87. **else** **if** (numberOf*LCS*Character[rowBacktrack - 1][coloumnBacktrack] >  numberOf*LCS*Character[rowBacktrack][coloumnBacktrack - 1]) { 88. rowBacktrack--; 89. } 90. **else** { 91. coloumnBacktrack--; 92. originalColoumnIndexY--; 93. } 94. valueForBacktrack.rowForBacktrack = rowBacktrack; 95. valueForBacktrack.coloumnForBacktrack = coloumnBacktrack; 96. valueForBacktrack.index*LCS*ForBacktrack = indexOf*LCS*; 97. valueForBacktrack.originalColoumnBacktrack = originalColoumnIndexY; 98. endIndex*LCS* = indexOf*LCS*; 99. backtrack = 1; 100. } 101. MPI\_Send(&valueForBacktrack, 4, MPI\_INT, destBacktrackProcess, 3,  MPI\_COMM\_WORLD); 103. **char** flag[] = { '\0' }; 104. **int** flag\_notcount = 1; 105. **if** (backtrack == 1) { 106. MPI\_Send(&initialIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 107. MPI\_Send(&endIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 108. **for** (**int** sendCharacter = endIndex*LCS*; sendCharacter <= initialIndex*LCS*;  sendCharacter++) { 109. MPI\_Send(&character*LCS*[sendCharacter], 1, MPI\_CHAR, 0, 4,  MPI\_COMM\_WORLD); 110. } 111. } 112. **if** (backtrack == 0) { 113. initialIndex*LCS* = -1; 114. endIndex*LCS* = -1; 115. MPI\_Send(&initialIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 116. MPI\_Send(&endIndex*LCS*, 1, MPI\_INT, 0, 4, MPI\_COMM\_WORLD); 117. } 118. } 119. **if** (process == 0) { 120. **int** initialIndex*LCS*, endIndex*LCS*, lengthOf*LCS*; 121. **int** coloumnBacktrack, rowBacktrack; 123. MPI\_Recv(&lengthOf*LCS*, 1, MPI\_INT, numberOfProcess - 1,  5, MPI\_COMM\_WORLD, &status); 124. vector<**char**> characterOf*LCS*; 125. characterOf*LCS*.resize(lengthOf*LCS*); 127. characterOf*LCS*[lengthOf*LCS*] = '\0'; 128. **int** finishReceived*LCS* = 0, finishBacktrackAndReceived = 0,  finishBacktrackProcess0 = 0; 129. **int** firstColoumn = ((process)\*lengthOfY / numberOfProcess); 130. coloumnStartBacktrack =   (((process)\*lengthOfY / numberOfProcess)) - firstColoumn; 131. coloumnEndBacktrack =   ((process + 1) \* lengthOfY / numberOfProcess) - firstColoumn; 133. **int** index*LCS*, originalColoumnIndexY; 135. rowBacktrack = valueForBacktrack.rowForBacktrack; 136. coloumnBacktrack = coloumnEndBacktrack; 137. index*LCS* = valueForBacktrack.index*LCS*ForBacktrack; 138. originalColoumnIndexY = valueForBacktrack.originalColoumnBacktrack; 139. **while** (coloumnBacktrack >= coloumnStartBacktrack &&  coloumnBacktrack <= coloumnEndBacktrack) { 140. **if** (rowBacktrack - 1 < 0 || coloumnBacktrack - 1 < 0 ||  originalColoumnIndexY - 1 < 0 || index*LCS* <= 0) 141. **break**; 142. **else** **if** (X[rowBacktrack - 1] == Y[originalColoumnIndexY - 1]) { 143. characterOf*LCS*[index*LCS* - 1] = X[rowBacktrack - 1]; 144. rowBacktrack--; 145. coloumnBacktrack--; 146. index*LCS*--; 147. originalColoumnIndexY--; 148. } 149. **else** **if** (numberOf*LCS*Character[rowBacktrack - 1][coloumnBacktrack] >  numberOf*LCS*Character[rowBacktrack][coloumnBacktrack - 1]) { 150. rowBacktrack--; 151. } 152. **else** { 153. coloumnBacktrack--; 154. originalColoumnIndexY--; 155. } 156. } 157. finishBacktrackProcess0 = 1; 159. **if** (finishBacktrackProcess0 == 1) { 160. **for** (**int** fromProcess = numberOfProcess - 1;  fromProcess >= 1; fromProcess--) { 161. MPI\_Recv(&initialIndex*LCS*, 1, MPI\_INT, fromProcess, 4,  MPI\_COMM\_WORLD, &status); 162. MPI\_Recv(&endIndex*LCS*, 1, MPI\_INT, fromProcess, 4,  MPI\_COMM\_WORLD,  &status); 163. **if** (initialIndex*LCS* == -1 && endIndex*LCS* == -1) { 164. } 165. **else** { 166. **for** (**int** received*LCS* = endIndex*LCS*;  received*LCS* <= initialIndex*LCS*; received*LCS*++) { 167. MPI\_Recv(&characterOf*LCS*[received*LCS*], 1, MPI\_INT,  fromProcess, 4, MPI\_COMM\_WORLD, &status); 168. } 169. } 171. finishReceived*LCS* = 1; 172. } 173. } 175. finishBacktrackAndReceived = 1; 176. } 177. } 178. } |

Kode Sumber 7.4 Pemilihan Indeks Perhitungan

|  |
| --- |
| 1. **int** rowStart = ((alpa \* (k - 1) \* lengthOfX) / numberOfProcess); 2. **int** rowEndPrevProcess = ((alpa \* (k - 2) \* lengthOfX) / numberOfProcess); 3. **int** rowEnd = (alpa \* k \* lengthOfX) / numberOfProcess; 5. **if** (rowStart != 0) 6. rowStart = rowStart + 1; 8. **if** (k != 1 && rowEnd == 0 && rowStart == 0) { 9. rowStart = rowStart + 1; 10. } 11. **if** (k != 1 && rowEndPrevProcess == 0 && rowStart == 0) { 12. rowStart = rowStart + 1; 13. } 15. **int** coloumnStart = (process)\*lengthOfY / numberOfProcess; 16. **if** (coloumnStart != 0) 17. coloumnStart = coloumnStart + 1; 19. **int** coloumnEnd = (process + 1) \* lengthOfY / numberOfProcess; |

Kode Sumber 7.5 Pemilihan Indeks Kolom Untuk Menerima Data

|  |
| --- |
| 1. **int** colStartForProcess1; 2. colStartForProcess1 = (1 \* lengthOfY / numberOfProcess); 3. **if** ((process \* lengthOfY / numberOfProcess) != colStartForProcess1) { 4. coloumnReceive = firstColoumn - coloumnPreviousProcess; 5. } 6. **else** { 7. coloumnReceive = coloumnStart - 1; 8. } |

Kode Sumber 7.6 Pengambilkan Karakter LCS

|  |
| --- |
| 1. **if** (X[rowBacktrack - 1] == Y[originalColoumnIndexY - 1]) { 2. character*LCS*[indexOf*LCS* - 1] = X[rowBacktrack - 1]; 3. rowBacktrack--; 4. coloumnBacktrack--; 5. indexOf*LCS*--; 6. originalColoumnIndexY--; 7. } 8. **else** **if** (numberOf*LCS*Character[rowBacktrack - 1][coloumnBacktrack] >  numberOf*LCS*Character[rowBacktrack][coloumnBacktrack - 1]) { 9. rowBacktrack--; 10. } 11. **else** { 12. coloumnBacktrack--; 13. originalColoumnIndexY--; 14. } |

Kode Sumber 7.7 Perhitungan Nilai Indeks LCS

|  |
| --- |
| 1. **if** (rowCalculate == 0 || coloumnCalculate == 0) { 2. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] = 0; 3. } 4. **else** **if** (X[rowCalculate - 1] == Y[coloumnCalculate - 1]) { 5. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] =  numberOf*LCS*Character[rowCalculate - 1][coloumnCalculate - 1 - firstColoumn] + 1; 6. } 7. **else** { 8. numberOf*LCS*Character[rowCalculate][coloumnCalculate - firstColoumn] =  max(numberOf*LCS*Character[rowCalculate - 1][coloumnCalculate - firstColoumn],  numberOf*LCS*Character[rowCalculate][coloumnCalculate - 1 - firstColoumn]); 9. } |

***[Halaman ini sengaja dikosongkan]***

BIODATA PENULIS

I Gusti Ngurah Adi Wicaksana, lahir pada 29 September 1995 di Denpasar. Penulis merupakan seorang mahasiswa yang tengah menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain mendengarkan dan bermain musik serta bermain futsal. Pernah menjadi asisten dosen di mata kulian Sistem Digital dan Kecerdasan Buatan, serta asisten praktikum pada mata kuliah Sistem Operasi dan Jaringan Komputer. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2 dan Staff Ahli Departemen Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-3.

Kritik dan saran sangat diharapkan guna peningkatan kualitas dan penulisan selanjutnya. Untuk itu, silahkan kirim kritik dan saran ke : [adiwicaksana29@gmail.com](mailto:adiwicaksana29@gmail.com).