Using Golang for implementation of a concurrent and distributed realtime processing system.

CHAI YING HUA

SESSION 2017/2018

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
SEPTEMBER 2017

Using Golang for implementation of a concurrent and distributed realtime processing system.

BY

CHAI YING HUA

SESSION 2017/2018

THIS PROJECT REPORT IS PREPARED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
IN PARTIAL FULFILLMENT
FOR
BACHELOR OF COMPUTER SCIENCE (HONS)
WITH SPECIALIZATION IN
SOFTWARE ENGINEERING
FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

FEBRUARY 2018

Declaration

I hereby declare that the work in this thesis have been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Name: *Chai Ying Hua*
Student ID: 1141328508
Faculty of Computing & Informatics
Multimedia University
Date: 7^{th} February 2018

Acknowledgements

The success and outcome of this project require tons of guidance and assistance from many people, and I am blessed and appreciate to have got this all along the completion of my project. My project would not be complete smoothly without their helping hands.

First and foremost, I would like to express sincere gratitude to my project supervisor, Mr Wan Ruslan Yusoff of Faculty of Computing Informatics at Multimedia University Cyberjaya for your unfailing support and assistance on my project. The door to Ruslan office and his mailbox was always open whenever I faced any impediment and trouble about my project or writing. He consistently and patiently steered me in the right direction whenever he thought I needed it. Also, he is eager to share his expertise and industrial experience in computing fields and provide encouragement and motivation on my project.

I owe gratitude to my parents for providing chances and opportunity for me to study in Multimedia University. They are caring, and concern about my academic and regularly provide support and attention on cultivating me to get myself prepare for an upcoming challenge.

Last but not least, I place on record, my sense of gratitude to my friend and classmate, who directly and indirectly unceasing encouragement and provide guidance till the completion of my project.

Abbreviations and Acronyms

IBM	International Business Machines
GCP	Google Cloud Platform
AWS	Amazon Web Services
ICT	Information and Communication Technology
AMD	Advanced Micro Devices
GCC	GNU Compiler Collection
GCCGO	Golang GNU Compiler Collection
LEO	Longitudinal Education Outcomes
NSPL	National Statistic Postcode Lookup
OORDBMS	Object-Oriented Relational Database Management System
MMU	Multimedia University
FYP	Final Year Project
IDE	Integrated Development Environment
UK	United Kingdom
CTF	Capture The Flag
MVCC	Multi-Version Concurrency Control
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
OO	Object-Oriented
POC	Proof Of Concept
OS	Operating System
CSV	Comma Separate Values
GDB	GNU Project Debugger
GNU	GNU's Not Unix!
UNIX	Uniplexed Information and Computing Services
SQL	Structured Query Language
WIP	Work In Progress

Contents

List of Tables

List of Figures

Listing

Management Summary

The project focuses on a utilized concurrent programming language concepts and their expressive power on data processing with concurrent computing.

The research draws attention on implementation and utilization of Go and Rust programming language on data processing cycle with PostgreSQL database as data storage. These languages' paradigm, characteristic and focus are used in data preparation, data processing and data storage.

Big datasets are obtained from secondary sources with data collection and verify with data validation to inspect the quality and logical weakness in data contents. The raw datasets in CSV format will be backup and import into PostgreSQL database with data transformation. The defects discovered such as inconsistency, incorrect and duplication in large datasets are eliminated with data encoding and data cleaning. Ultimately, the unnormalized and unorganized data will be migrated into normalized table in new storage to establish excellent relational database management system freed from anomalies.

Several concurrent programming language based programs are developed to support data processing activities such as data transformation, data cleaning and data migration. The processing execution's performance of program developed from different concurrent programming language and programming style will be compared and discussed in detail.

PL/pgSQL scripts will be developed to create database entity's data structure, objects, schemas and perform data migration within PostgreSQL database. The lightweight scripts will execute multiple written query simultaneously to perform database creation, manipulation and control efficiently.

The project successfully prove concurrent programming has better performance and throughput on data processing compare to sequential programming. Data duplication, data inconsistencies and data incompleteness had successfully eliminated to establish high data quality. The capabilities and limitation of concurrent programming features on data processing are demonstrated and further discussed.

Chapter 1

Introduction

1.1 Introduction

In a globalization and modernization era, the volume and variety of big data continue to increase at an exponential rate. Cloud computing environment such as IBM, Microsoft Azure, GCP and Amazon AWS possess great shifts in modern ICT and robust architecture to perform large-scale and complex computing service for enterprise applications.[1] Chip makers AMD, IBM, Intel, and Sun rapidly building chips with energy-efficient multiple processing cores that improve overall performance by handling more work in parallel for server, desktops and laptops. [2] The performance and availability of system required to increase dramatically with the inclusion of multi-threading and multi-processing.

Software development activities are consistently working on improving efforts in development and deployment activities by solving issues, challenges and problem regarding concurrent and distributed computing. With the advent of client/server focus; massive cluster and networking technologies, the

advancement of technology reveal problem and constraints on linguistic issues to the developer.[3] Availability of inexpensive hardware allow developer to exploit various possibilities in the construction of distributed system and multi-processors that were previously economically infeasible. [4]

Software application today is inherently and expanded into concurrent and distributed computing with real-time applications. [5]. However, majority of systems language not designed with concurrent and parallelization in mind and software users and load of request gradually increase.

Google created a new concurrent programming language, known as Go to rewrite their large production system to solve compile time and string processing by inventing a language that design for efficiency, simplicity and quick compilation without dependency checking. [6] At the same time, Mozilla Research invents a system concurrent programming language, known as Rust that emphasize security, safety and control with performance.

Go is free and open source programming language created by Google at 2007 and announce on 2009 [7] with two compiler implementation, GC and GCCGO. [8] The language were designed for high-speed compilation, support for concurrency and communication, and efficient or latency-free garbage collection. It is C-like and statically typed language that compiles into single binary with go compiler to reduce compile time. Go allow developer to model problems with a random order of events, optimize data operations, and utilize parallel processing of machines and network with concurrency programming. [9]

In this paper, we are going to focus on utilizing concurrent programming concepts of RUST and Go language in data processing activities. We will develop programs with Go and Rust to carry out retrieving, transform, cleaning,

parsing and migration operations in data processing cycle. This paper attempts to expose important concepts of these languages and conduct a comparison for the use of self-study material and propose an evaluation scheme.

1.1.1 Project Brief Description

We will use the Go and Rust programming language to process a combination of static data to represents a real time, concurrent processing system. For this project, we will covering the utilization of concurrent languages' elements and key concepts in entire data processing cycle. The cycle consists of collection of data sources (inputs), implementation of data processors (program filters/codes), and manipulation of data storage.

The Go and Rust programming language based application process mash-up of three unambiguous, free and informed consent dataset in stream. These application are developed to demonstrate the capabilities of concurrent features on data processing activities. These program attempts to transform specific structure of data into required format of data storage. In addition, the application capable to detect and correct inaccurate records found in datasets and import into PostgreSQL database, an object-oriented relational database management system (OORDBMS).

PL/pgSQL, a procedural language supported by PostgreSQL OORDMS is used to write data definition language (DDL) and data manipulation language (DML) to create objects, schemas and structure of database. The query are written into files and composed as scripts to be executed automatically to perform database creation, manipulation and control efficiently.

The performance execution of program developed from different programming language and programming style will be recorded, compared and discussed in detail. Further conclusions and inference can be drawn from execution results to identify the expressive power and concepts of these language.

1.1.2 Project Objectives

The objectives of this project are:

1. To learn and understand about Go and RUST programming language concepts and their concurrent processing features.
2. To explore different techniques on data processing, concurrent and distributed programming for big data.
3. To conduct performance comparison between Go and Rust language implementation in data processing with concurrent programming.
4. To conduct a comparison on Go and Rust concurrent programming language concepts in retrieving big data with different techniques.
5. To implement the handling of big data with PostgreSQL, an object-oriented relational database management system (OORDBMS).

1.1.3 Project Motivations

During my involvement and participation of industrial training in JobStreet.com (A SEEK ASIA Company), my colleague often discuss about Golang implementation in worker thread with session on server side scripting to handle concurrent request and reduce web server loads. In Tech Talk Thursday with Grab Singapore organised in MMU Cyberjaya in January, the speaker mentioned the companies use Go language as tool to build their backend on handling request. Indirectly, the discussion and seminar by technical professionals stimulate my curiosity on capabilities and usage of golang.

In my process of exploration, I had attended several Golang meetups and learning sections in Kuala Lumpur. I am impressed the new language helps company saving cost on building servers and running well in small hardware specs. Other than that, I had discovered various notable company and sites start migrated their essential services and critical component from other languages to Go. Within several years, Google's Go language has gone from being an unfamiliar language to well-known promising tools or significant source for a big technology company to develop fast-moving new projects.

As Go soared to a new height in Tiobe programming language popularity, it has inspired me to gather more information and knowledge regarding the capabilities of the language. After viewing online articles and journals, I had discover this concurrency-friendly programming language may be the future of development, and it stimulates my passion and excitement for learning the language.

Simultaneously, I notice this project was published as FYP title in this semester. Without any hesitation, I am exhilarated to pursuit and register this

project in my final academic year in order unveil the capabilities of golang. It will be enjoyable and great to learn this language throughout the project.

1.2 Project Scope

1.2.1 Phase 1 Scope of Work

1. Research project interest and raise question in different categories of data repositories.
2. Setup boot partition for Ubuntu 16.04 LTS operating system with Window 10.
3. Install Go language compiler and RUST language compiler on PC.
4. Install Eclipse for Parallel Application IDE.
5. Install Goclipse and RUST GUI into Eclipse IDE.
6. Install Terminator application into Ubuntu; it is an application that produces multiple terminals in a single window so that developer can perform various task in a single environment.
7. Install Synaptic Package Manager that enable upgrade and remove software package a user-friendly way without dealing with dependencies issues.
8. Set up PostgreSQL into PC for big data handling.

1.2.2 Project Deliverables for Phase 1

1. Acquire free, consent and big UK's basic company data published by Companies House in data.gov.uk that containing basic company data of live companies on the register for data processing.
2. Acquire institution subject data published by UK Higher Education site and create a mashup in a project which works with two sets of data and process them to provide output.
3. Acquire postcode data for UK location as the linker of basic company data with institution subject data.
4. Develop a proof of concepts and understanding on concurrent and program with Go language.
5. Write Go code for sequential and concurrent programs which able to process raw CSV data and PostgreSQL database.
6. Conduct comparison on sequential and concurrent programming with Go programming language on retrieving 300 rows of data.

1.2.3 Phase 2 Scope of Work

1. Perform data encoding to convert dirty data into consistent and valid format.
2. Perform database normalization to eliminate data redundancy and improve data integrity.
3. Write a Go programming language based sequential and concurrent program as ORM tool to export data from raw CSV file and PostgreSQL database convert into object model.

4. Write a Rust programming language based sequential and concurrent program as ORM tool to export data from raw CSV file and PostgreSQL database convert into object model.
5. Perform data cleaning to eliminate missing data and standardize the fields in consistent format.
6. Develop Go programming language based data cleaning parser to clean company raw datasets and import into PostgreSQL database.
7. Perform database tuning to optimize database's performance on handling extra workloads and increase client's connection limit.
8. Perform query tuning to increase query execution performance on data processing.
9. Develop several Go programming language based concurrent program as data migration tool to transfer data from legacy storage into new storage within PostgreSQL database.
10. Write several PL/pgSQL scripts to import raw data from legacy storage into normalized table within PostgreSQL database.
11. Perform data verification to verify the consistency and accuracy of database records after the data migration is complete.
12. Perform distributed programming to process data through multiple nodes.

1.2.4 Project Deliverables for Phase 2

1. A dirty raw CSV datasets shall be encoded and consistent format.
2. A Go programming language based and a Rust programming based ORM tools capable to retrieve 4 millions row of data from CSV datasets and

PostgreSQL database in sequential and concurrent manner.

3. Duplication, missing, corruption and inconsistency of data shall be eliminated with data cleaning.
4. The database shall capable to handle extra workloads and allow more clients to establish concurrent connection on perform transaction simultaneously.
5. Conduct performance comparison of sequential and concurrent programming with Go programming language on retrieving 4 millions row of data.
6. Conduct performance comparison of sequential and concurrent programming with Rust programming language on retrieving 4 millions row of data.
7. Conduct performance comparison between Go and Rust programming language on data retrieval.
8. A Go programming language based data migration tool capable to migrate 4 millions of data from legacy storage to normalized table within PostgreSQL database.
9. Several PL/pgSQL scripts capable to perform table creation, data manipulation and migrate data from legacy storage into normalized table within PostgreSQL database.
10. Ensure the migrated data are consistent and accurate.

Chapter 2

Literature Review

2.1 Sequencial Programming vs Concurrent Programming

Sequential programming involves process execution one after another [10] and have no linguistic design construct for concurrent computations. [11] The processes will only run after other is successful and executed chronologically in predetermined manner. [12] However, it's difficult to implement complex interaction and handle problems in parallel and concurrent environments with single-threaded. [13]

Concurrency had cause major turning point force in software development for developing concurrent software in order to exploit greater efficiency and performance optimization by fully utilize multiple core. To leverage the full power of hardware resource in software industry, concurrency and clouds will be the things every developer requires to deal with future software development

and it is essential for both concurrent and distributed system. [14] Future generation computing system likely being developed by concurrent programming on multiprocessors. [15]

2.2 Concurrent Programming

Concurrent programming is form of computing where two or more threads cooperate to achieve common goals, inter-process communication and synchronization without require multi-processors. [16] Implementing concurrency into system requires imperative and functional language which allow programmer to take in control of concurrency by specifying step-by-step changes to variables and data structures in manipulation of data. [17] Therefore, concurrent programming language possess the ability to enable express concurrent computation easily by making synchronization requirements achievable and facilitate parallelism. Moreover, concurrent programming language possess programming notation, package and techniques for expressing potential parallelism and solving resulting synchronization and computer system communication problems. [4]

2.3 Distributed Programming

Concurrency and distributed programming often discuss together on implementing for a wide application of computer platforms from mobile devices to distributed servers. Distribute programming is form of computing where various source of parallelism running program on multiple machines simultaneously. It allow a distributed server make efficient use of network resources to communicate and coordinate in order to provide closer service for clients. [18] Concurrent programming is used to implement distributed process for real-time applications operate by microcomputer networks which possess distributed storage. The concurrent program is implemented into distributed server or storage in order to execute sequential processes simultaneously. Concurrent Pascal is possible to satisfy the efficiency, reliability and consistency of distributed storage. [19]

2.4 PostgreSQL

PostgreSQL is general object-oriented relational database management system that first possesses MVCC feature before Oracle. It is an open source object oriented relational database management system (OORDBMS) created by University of California [20] and currently maintained by the PostgreSQL Global Development Group with companies and contributors. PostgreSQL supports various concurrent programming language such as C, C++ and Java, etc and guarantees data consistency while performing concurrency transaction. [21] Other than that, PostgreSQL store multiple version of records in the database by keeping the latest version of tuple and garbage collects old records no longer required. [22]

The database is implemented with TelegraphCQ data flow system for processing continuous queries in data streaming environment. Research has found the open source database system possess extensibility feature and reusable component to improve adaptivity and concurrent read-write. [23] Ultimately, PostgreSQL is used to optimize pipeline on handle runtime update request for conventional data warehouse to process data analysis concurrent queries efficiently. The database system offers a modern feature to support adaptive query processing and maximize work sharing during execution. [24]

The advantage of PostgreSQL are listed as follow:

1. **Multi version concurrency control (MVCC).** The database system allows client to perform concurrent request and transaction to data and enforcing data consistency. [25] It provided support for concurrency model and designed for high volume environments with serializable transaction

isolation level to prevent dirty reads and better than row-level locking provided by several enterprise database systems such as MySQL. [26]

2. **Process-based.** PostgreSQL server is process-based and not threaded-based which increase robustness and stabilization during querying data compare to other database systems for this project. This can be explained by the difference between multiprocessing and multi-threading. A single thread die kills whole multi threaded environment dies but single process terminate will not affect other process running.

3. **Support Ubuntu OS.** PostgreSQL provides lifetime support for Ubuntu version. The database system repositories such as core database server (postgresql-9.5), client libraries and binaries (postgresql-client-9.5) and other additional modules (postgresql-contrib-9.5) are supported and consistent with various Linux distribution. [27]
4. **Security.** PostgreSQL make data processing more safety compare to direct retrieval with CSV because it is not open for modification by normal user.

2.5 Go language

Go's principle focus on simplicity, orthogonal, succinct and safe to provide its expressiveness to support efficient large scale programming, faster compilation speed and utilized multi-core hardware. [28] In the past, Go had been used to implement high-performance, scalable radio access system to evaluate its suitability and language functionality. [29]

The language had also utilized to assess text data processing in information system and mentioned Go is promising featuring native support for distributed applications. [30] Other than that, Go's concurrency primitives is used to implement an artificial intelligence and graph theory based sliding-puzzle game for Unix terminals. The language concepts and package are supportive to developed real-time notification delivery architecture with its what s. [31]

2.6 Rust language

Rust is a new and multi-paradigm programming language developed by Mozilla Research. [32] Earlier projects were using the Rust programming language to built several higher level abstractions on GPU kernels. They show how Rust advanced features enable to support both system-level concept and high-level operators on GPU computing. [33] Small model of RUST called Patina was experimented and study for claiming the language memory is safety without garbage collection by identify whether there are leaks during deallocating memory and ensure data initialized correctly on the runtime memory. [34]

2.7 Comparison of concurrent programming language concepts

Experimental design and demonstration are often conducted by the previous researcher to compare concurrent programming languages concepts with debugging existing system and writing correct new programs. [35] Structure embedding concepts in several concurrent programming languages has been examined by demonstrating mapping to a parallel composition to test its expressive power of these languages through results. [36]

Moreover, a general method is developed by previous research for comparing concurrent programming languages based on categories of language embeddings to obtain separation results. The programming language's properties affect the concept and performance of concurrent programming language. As an example, even though CSP and Actors possess common characteristic with

non-compositional observable equivalence and interference free but CSP contains composition with hiding while Actors don't. [37]

In addition, expressive power of concurrent programming languages often compared by previous research to investigate how synchronization and logical control construction affect the efficiency of resulting word from three computational model. [38] Several conventional techniques and concurrent programming structures were analyze for implementing objects related to critical sections with concurrent programming languages. [39] Furthermore, previous researchers had proposed classification frameworks to study relevant elements of architecture description languages by present definition for comparing language components, connectors and configurations. [40]

Surveys is conducted on a preference of design and language features on 13 concurrent languages and found available architectural supports profoundly influence the language's style. The results indicate the concurrent feature of programming language will influence the intended use and application of the language. [41] In addition, previous research is conducted to compare implicit and explicit parallel programming with SISAL and SR to evaluate for programmability and performance. [42] Detailed performance measurements are presented with the comparison of various parallel architecture and measured with Beowulf-class parallel architecture. [43]

2.8 Comparison of Go and Rust language

Go and RUST has start to gain popularity among the trends. [44] Rust and Go are also some of the developers most loved programming language. [45] The

Rust and Go programming languages are new programming languages for implementing concurrent and distributed based system. [46]

Go and RUST are both new concurrent programming language create after the year 2000. Go had become language of the year in Tiobe programming language ranking in 2009 and 2016. [47] Simultaneously, Rust won first place in most love programming language in Stack Overflow survey 2016 and 2017. [48]

Both concurrency programming languages support functional and imperative procedural paradigms. [49] [50]. Go is a CSP-based language provide rich support concurrency with goroutines and channel [51] but Rust is an actor model language focus on memory safety over performance. [52] Go and Rust often used to be compared with current software industry in concurrent computing implementation. [53]

Figure 2.5 shows characteristic and paradigm of Go and RUST programming language. All the language characteristic below will be discussed in the following subsection.

Language	**Go**	**Rust**
Categories	Communicating Sequence Process (CSP), High-level	Actor Model, Low-level
Focus	Simplicity, Concurrency, Efficiency	Memory Safety, Concurrency, Security
Intended Use	Application, games, web, server-side	Application, System
Imperative	Yes	Yes
Multi-paradigm	Yes	Yes
Object-oriented	Yes	No
Functional	Yes	Yes
Procedural	Yes	Yes
Generic	No	Yes
Reflective	Yes	No
Event-driven	Yes	No
Failsafe I/O	Yes (unless result explicitly ignored)	Yes (unless result explicitly ignored)

FIGURE 2.1: Comparison of Go and Rust language characteristic

2.8.1 Comparison of language categories and focus

Go is a high-level language focus on simplicity, reliability and efficiency. The language is designed with communicating sequential process (CSP) to express concurrency based on message passing channels. The processes and messages communicate via goroutine and gochannel within a shared memory. [54] The language is intended to use for building web application programming interface (API) or networking application such as TCP or HTTP server to handle request.

Go possess simple syntax, garbage collector and runtime which allow developer to increase code readability and implement concurrency easier. However, Go is lack of language extensibility which leads to a limitation on implement manual memory management. [55]

Rust is a low-level language focus on memory safety, security and fault tolerance. The language designed with actor model concurrent programming language that use “actors” as fundamental agent on message passing. The actor takes input, send output after performing functions. [56] The processes and message communicate point-to-point via actors in a consistent state. The language intended use for system programmings such as building game engines, driver and embedded devices.

Rust doesn't possess garbage collection and runtime which promote extensibility and deterministic on implement memory management. [57] However, Rust has much inherent complexity of syntax and semantics and has a high learning curve for a developer.

2.8.2 Similarities of Go and Rust language

The similarities of both languages are discussed as follow:

1. **Imperative.** Go and Rust are imperative programming paradigm where a value can be assigned into a variable to perform operation on information located in memory. Moreover, these languages allow declaration of a variable to store the results in memory for later use, affect the global state of a variable.
2. **Functional.** Go and Rust language can be written with mathematical functions to express control flow by combining function calls. The function avoid changing global state of variable.
3. **Procedural.** Go and Rust language can be written into statement structured and divided into function. The function known as procedure takes input processes it and produces output.
4. **Multi-paradigm.** Go and Rust language are support various programming paradigm and provide developer to use suitable programming style to develop a program to achieve project objectives.
5. **Failsafe I/O and callbacks.** Go and Rust language compiler warn error or throw an exception if the system calls fail. Go language throw errors if developer doesn't use the declare function or variable and Rust language does not compile if found any dangling pointers.

2.8.3 Difference between Go and Rust language

The difference between both languages are discussed as follow:

1. **Object-oriented.** Go language support object-oriented programming with struct and interface. However, Rust is not an object-oriented language result of the idiomatic language and its appearance in an OO language. [58]
2. **Generic.** Go language is lack of generic where the compiler doesn't allow declared a function or variable written in to-be-specified-later types await to be instantiated when needed for a specific purpose. However, Rust is possible to specify generalized function and avoid codes rewriting.
3. **Reflective.** Go language possess the ability to observe and modify type, object, function execution on runtime by import "reflect". However, Rust doesn't have reflection.
4. **Event-Driven.** Go is a high-level language enable write application respond to demand and expectation from mobile devices, multicore architectures and cloud computing environments. However, Rust is a low-level language prevent the flow of program interrupt by an event from user actions to enforce security and safety.

2.9 Ubuntu 16.04.03 LTS 64-bit OS

Ubuntu OS is an open source operating system with Linux distribution system and based on Debian architecture which provides long-term support (LTS) on security and fixes. [59] The advantage of Ubuntu operating system are described below:

1. **Free and customizable.** The openness of using Ubuntu OS offers a wide range of choices for the programmer to conduct development activities with Linux terminal. The APT packaging system allows developer to manage software and programming languages package efficient compared to Window operating system. The OS provides freedom in customization for a developer to catered different sets of need with source access and root permission to meet project requirements.

2. **Security.** The system files are owned by root in Ubuntu OS and not accessible by casual user, malware and third party software without root privilege. [60] As the operating system is maintained and contributed by vast amount of developer and programmer due to its open source and environment, the bugs are fixed efficiently with regular updates and provide less vulnerability for the attacker to exploit the system. [61] The key factors underline within Ubuntu security provide sufficient statement to prove Ubuntu is more secure than Window or Mac OS on this project.

3. **Consistent.** Ubuntu OS provide excellent consistent from front-end (UIUX) to back end. The user interface and user experience of Ubuntu operating system increase usability and efficiency in development,

maintenance and deployment activities in the different version.

4. **Stable and Reliable.** UNIX preceded and outshine MS-DOS kernel with hardware abstraction, security model, resource management and various services that ran as background processes. [62] Ubuntu promotes multitasking and multi-user which is suitable and ideal for this project to conduct concurrent and distributed processing activities with PostgreSQL. Last but not least, MS-DOS is an image loader system that preload memory addresses without memory or resource management quickly leads to BSOD and data corruption during data processing.

2.10 Debugging tools

Debugging could be painful for a software engineer to monitor and identify the performance of applications running in concurrent and distributed on sophisticated operating systems like Ubuntu.

Debugging with printf() for program bring many disadvantages and limitation during concurrency programming. The function could consume much memory in the multi-threaded environment because it's not lightweight and thread safety. [63] Moreover, it is not an efficient way to identify problems occurs related to memory allocation or interruption.

Therefore, debugger is used in this project to understand event or consequence happens in a running software system without consuming the enormous amount of memory. Simultaneously, it helps developer to save times on finding coding and logic errors in source codes. [64]

2.10.1 GDB Debugger

GDB is a build in GNU debugger for UNIX systems to debug programs to obtain information of root cause that cause the program to fail. [65] GDB allows set breakpoints and watchpoints on certain functions and print values during the program execution with terminal interface. Unfortunately, GDB possess limitation on finding bugs cause by memory leakage and compile errors.

2.11 Eclipse for Parallel Application Developers Oxygen Release (4.7.0) IDE.

Eclipse is an integrated development environment create and maintain by Eclipse Open Source Project teams. The Eclipse Oxygen release possess better functionality and performance for a developer to manage, build and deploy software system. The advantage of Eclipse IDE are listed as follows:

1. **Auto Completion.** The openness of using Ubuntu OS offers a wide range of choices for the programmer to conduct development activities with Linux terminal. The APT packaging system allows developer to manage software and programming languages package efficient compared to Window operating system. The OS provides freedom in customisation for a developer to catered different sets of need with source access and root permission to meet project requirements.

2. **Integrated Environment.** The system files are owned by root in Ubuntu OS and not accessible by casual user, malware and third party

software without root privilege. [60] As the operating system is maintained and contributed by vast amount of developer and programmer due to its open source and environment, the bugs are fixed efficiently with regular updates and provide less vulnerability for the attacker to exploit the system. [61] The key factors underline within Ubuntu security provide sufficient statement to prove Ubuntu is more secure than Window or Mac OS on this project.

3. **Debugger.** Ubuntu OS provide excellent consistent from front-end (UIUX) to backend. The user interface and user experience of Ubuntu operating system increase usability and efficiency in development, maintenance and deployment activities in the different version.

4. **Plugins.** UNIX preceded and outshine MS-DOS kernel with hardware abstraction, security model, resource management and various services that ran as background processes. [62] Ubuntu promotes multitasking and multi-user which is suitable and ideal for this project to conduct concurrent and distributed processing activities with PostgreSQL. Last but not least, MS-DOS is an image loader system that preload memory addresses without memory or resource management quickly leads to BSOD and data corruption during data processing.

2.12 Chapter Summary

The finding for literature review is concurrent programming language possess specific built-in notation, package and functions to build parallel and distributed

application. PostgreSQL is suitable for this project because it possesses MVCC that able handle concurrent request with good adaptivity and accuracy. Golang and Rust are concurrent programming language support multi-paradigm programming with multiprocessing and multithreading. Go language focused on simplicity while Rust language focuses on security. Both programming languages invented with different model and concepts for a different purpose.

Concurrent language is often compared and evaluated with configuration, categories and architecture to obtain performance and expressive power. The language's feature is essential to prove the performance of specific concurrent language. Debugging tools play a main role on observing processes and threads activities during the development and debugging activity to ensure program's execution is observed and error are discovered.

Chapter 3

Project Design

3.1 Phase 1

3.1.1 Introduction

The primary focus of Phase 1 is implement prototype to prove theoretical concepts of the domain to research in this project. Requirements are listed as follow:

1. To acquire free large data set for big data processing.
2. To ensure data set acquired from the website are free, consent and clean with Devil Advocation Test.
3. A program will be implemented in RUST and Go programming language as a proof-of-concept (POC) that CSV raw data is capable of importing into PostgreSQL database.

4. A program will be implemented with Go programming language as POC that PostgreSQL database transaction can be sequential and concurrent.
5. A program will be implemented with Go programming language as POC that reading CSV files can be sequential and concurrent.
6. To ease the debugging and troubleshooting on concurrent and distributed development environment, LTTng tracing network and Eclipse Trace Compass will be installed to obtain a reading and outputs traces via Common Trace File (CTF) binary format.

3.1.2 Data Collection

The project is required to work with large data sets to utilize infrastructure and processing power of GO and RUST concurrent programming language. Data collection is conducted to identify of company recruitment preferences on higher education graduates of different subjects in the UK with basic company and LEO datasets. Data collected is required to be clean and able to solve interesting problem or question.

The characteristic of free, consent and licensed data sets acquired from UK government website provider (data.gov.uk) are as follow:

No	**Name of Datasets**	**Column**	**Rows**	**Size**
1.	Longitudinal Educations Outcomes (LEO)	21	32706	1.8 GB
2.	Basic Company Profile (Company)	55	3595702	667.5 MB
3.	National Statistics Postcode Lookup (NSPL)	35	1754882	4.2 MB

Table 3.1: Result of Golang programming on process CSV raw data

The file format of all large dataset obtained are Comma Separated Values (CSV) format which the information is organized with one record as one line and each field is separated by comma (,). CSV format is used for data processing in this project because it is human readable and simple to be parse. It can be handle using PostgreSQL database and retrievable by programs.

3.1.2.1 Longitudinal Education Outcomes (LEO) dataset

The data set focus on employment and earnings outcome of Bachelor's Degree graduate in Great Britain after five years. It contains information about students include personal characteristics, education or qualification achieved, employment and income earnings. The data dictionary of longitudinal education outcome is created and placed in Appendix I.1.1.

3.1.2.2 Basic Company dataset

The data set possesses up-to-date basic companies information on UK register. It contains company names, annual returns filing dates, location details, account and basic information about mortgage and business changes. The data dictionary of basic company dataset is created and placed in Appendix I.1.2.

3.1.2.3 National Statistics Postcode Lookup (NSPL) dataset

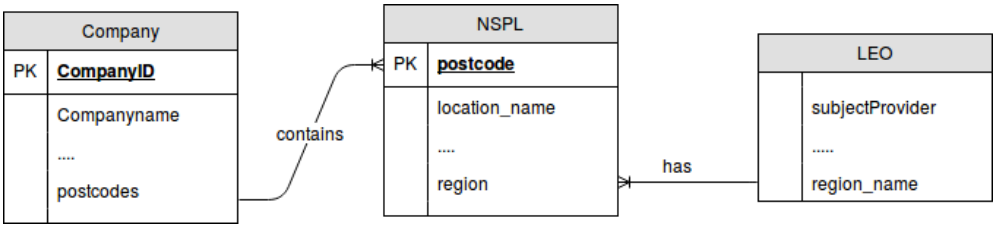


FIGURE 3.1: Entity Relationship Diagram

As postcode data for every location on earth is unique. Company data sets possess **postcode** field in the business address, but LEO dataset do not have the **postcode** field which leads to difficulty of defining a relationship between

these two datasets. Figure above show NSPL dataset serves as a linker to map **region** column from LEO data to link with **postcode** column found in company datasets.

The data set possesses current postcode for the United Kingdom. It contains information relates postcode number, location, country name, parliamentary constitution, electoral and other geographical details. The data dictionary of National Statistics Postcode Lookup (NSPL) dataset is created and placed in Appendix I.1.3.

3.1.3 Data Validation

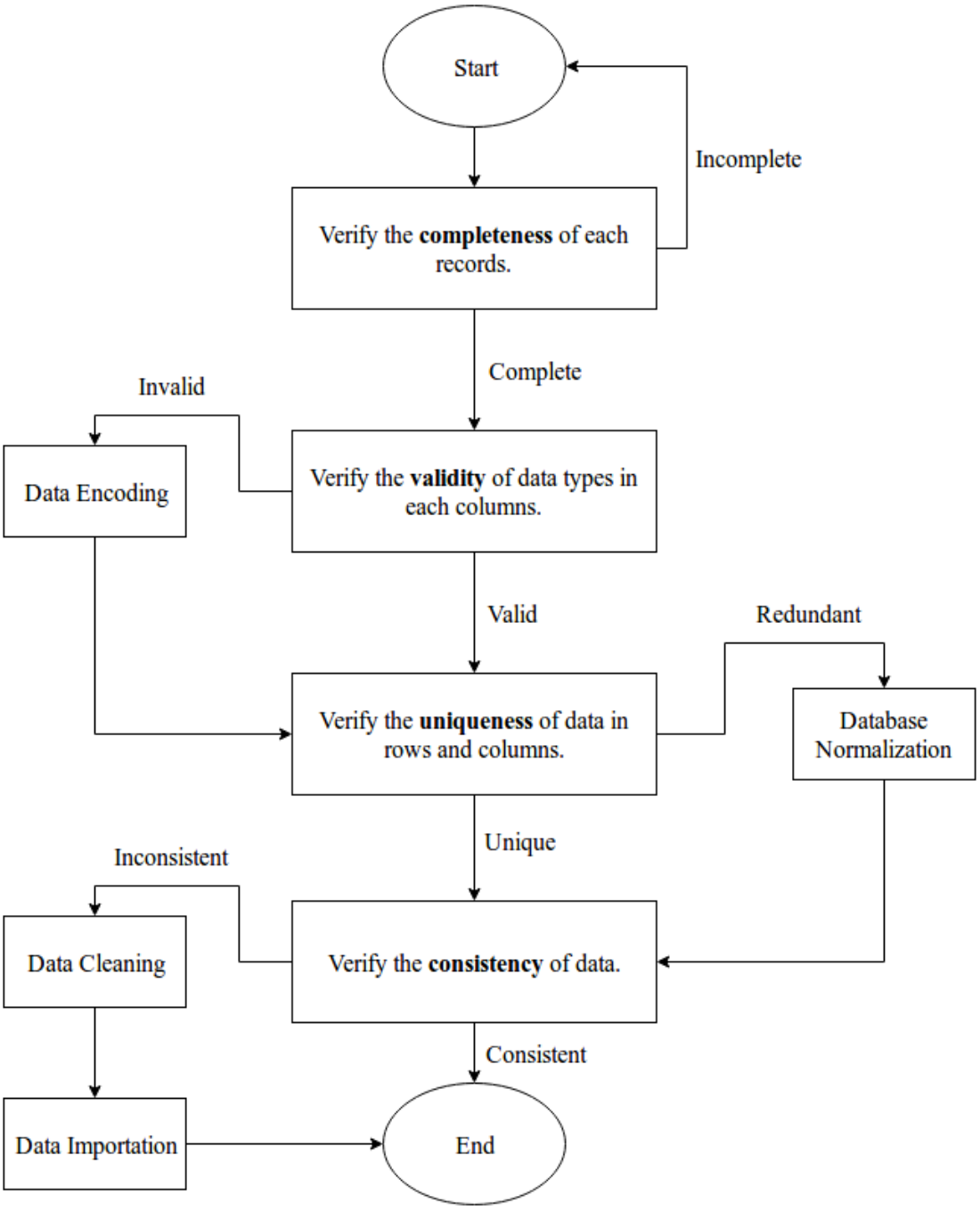


FIGURE 3.2: Data Validation Procedure Flowchart

Data validation is conducted to inspect the quality dimension of data sources acquire in Data Collection (Section 3.1.2) to prevent corruption, inconsistency and conflicts during importing, using and processing. It is performed to ensure the data acquired are clean and in excellent quality.

The important steps taken on validation of data are shown in Figure 3.5. The **completeness** of datasets will be examine to assures the characteristic of data fulfill Comma Separate Values (CSV) standard and requirement. The common test performed during data completeness check are using aggregate functions such as max, min or counts. [66]

Furthermore, the **validity** of data types in each columns are measured to prevent incompatible data types during Data Importation, Object Relational Mapping (ORM) and Data Migration. The types of data stored in each columns of obtained datasets shall be identify to describe suitable data type for Database Definition Language (DDL) during database table creation. As an example, the alphanumeric and text field are usually defined as VARCHAR and field contains only number will be declared as INTEGER.

In addition, the **uniqueness** of records will be verify to discover wasteful and duplication of data. The data redundancy indicates same piece of data are exist in multiple place. [67] This condition will results in waste of space, data inconsistency and violates data integrity. If the duplication of data is discovered, database normalization will be performed to eliminate the duplication of records.

Last but not least, the **consistency** of data will be analyze to ensure datasets obtained are conform to specific standards and meet requirements. The data consistency check shall be performed during data preparation to inspect

discover missing, corrupted or invalid data in record. The conformity and consistency of data in specific column should be handled in wariness to prevent affect the outcomes and efficiency of data processing. If the data is found inconsistent, Data Cleaning and Data Importation will be conducted to fix the defects discovered in the datasets.

3.1.4 Performance Benchmarking

To conduct a comparison between Go and RUST language, benchmarking plays an important role to achieve fairness in compare performance and expressive power of language.

The component that are benchmarked are listed below:

1. **SQL Queries run on program.** Go and Rust program execute the same amount of database retrieval query to achieve the fairness of comparison.

2. **Table configurations.** The space of table of this project should be same for Go and Rust program to test the performance.

3. **Hardware configurations.** Both Go and Rust program are required to run on same hardware configuration to achieve fairness of comparison on performance.

3.1.5 Database Retrieval Program

3.1.5.1 Phase 1 System Context Diagram

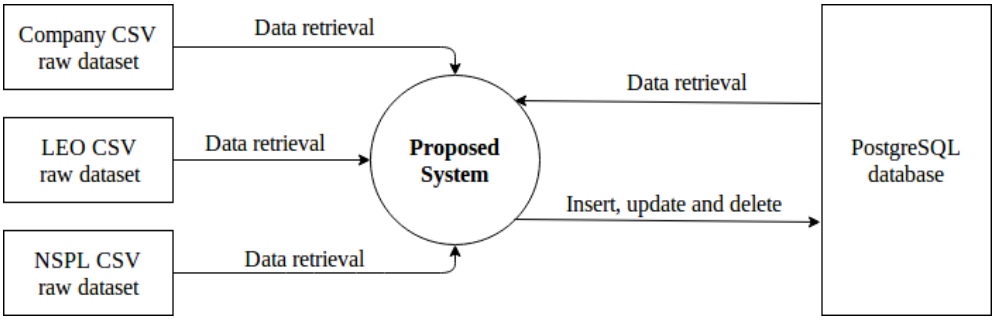


FIGURE 3.3: Phase 1 System Context Diagram

System context diagram provide high level view that defines relationship between proposed system with external entities. The proposed system is written in Go and Rust programming language with sequential and concurrent computing. The system shall process raw dataset stores in different nodes and dataset stores in PostgreSQL database. Moreover, the system should process data from raw CSV dataset and PostgreSQL database in sequential and concurrent manner.

3.1.5.2 Phase 1 Block Diagram

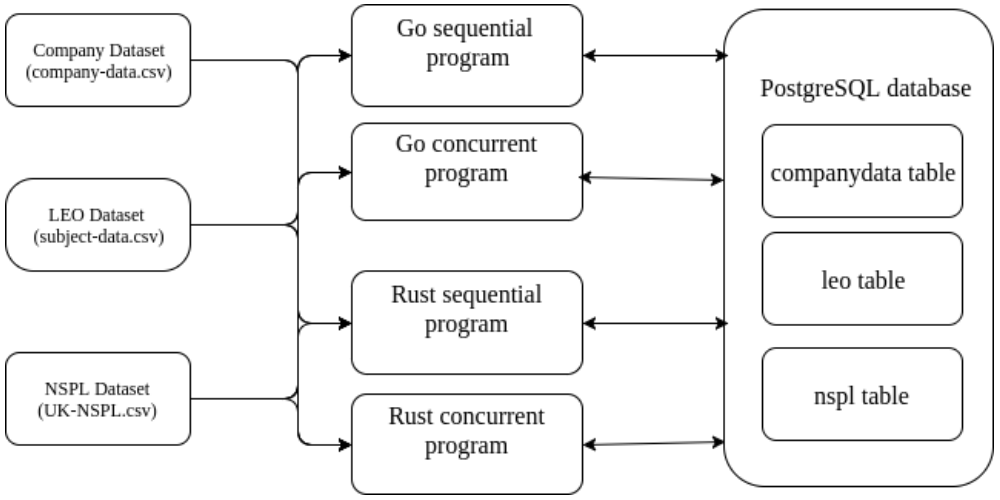


FIGURE 3.4: Phase 1 Block Diagram

The block diagram provides a high-level overview of importation CSV into PostgreSQL with Go and Rust program. The large dataset store is store in different nodes with CSV format. Data stores in PostgreSQL database and raw CSV data at different nodes will be processed by Go and Rust program with sequential and concurrent manner. The database table is created with query in the terminal before Go and Rust program is executed.

3.1.6 PostgreSQL Database Retrieval with Go and Rust program

3.1.6.1 Phase 1 Sequential Program Flowchart

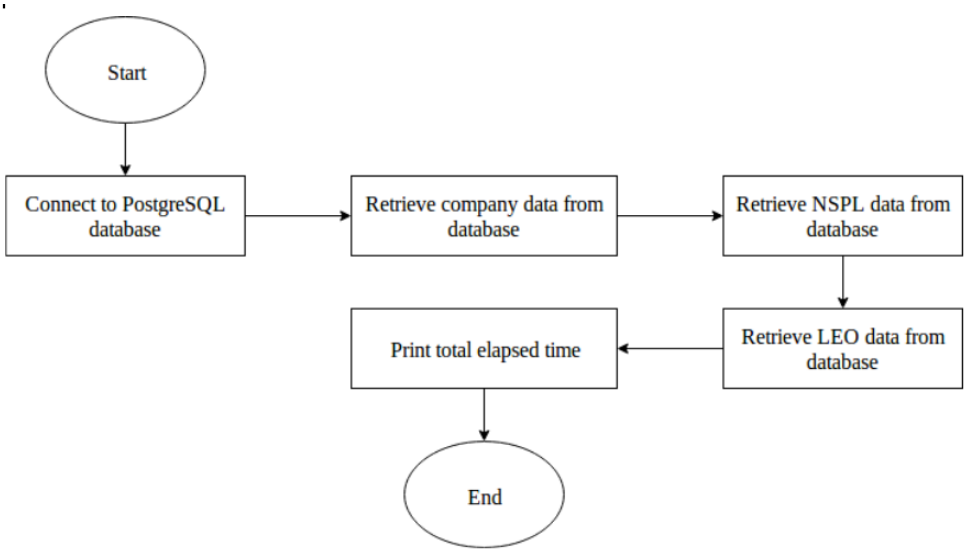


FIGURE 3.5: Phase 1 Sequential program flowchart

The flowchart provides a high-level view of concurrent manner during data retrieval in PostgreSQL with Go and Rust program. The program first establishes connection with PostgreSQL database with a connection string. Afterwards, it will retrieve a different set of data from various database table concurrently. The total elapsed time for entire program execution will be print.

3.1.6.2 Phase 1 Concurrent Program Flowchart

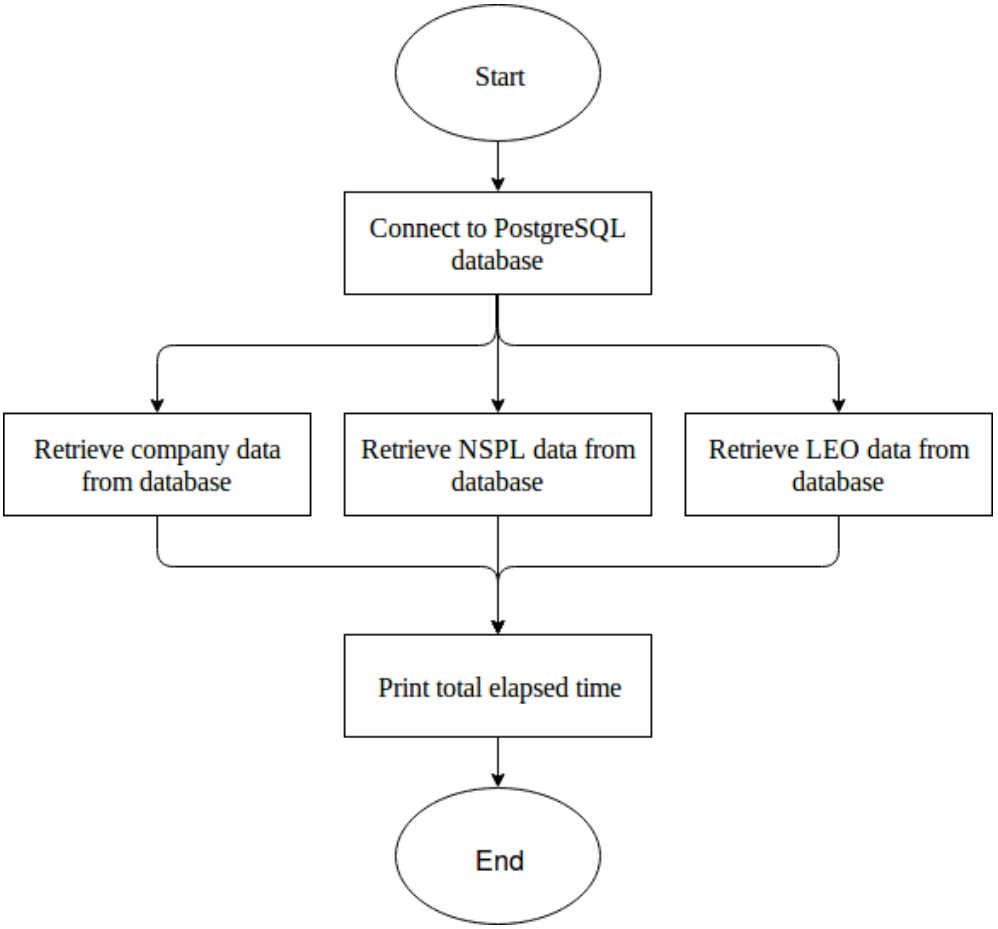


FIGURE 3.6: Phase 1 Concurrent program flowchart

The flowchart provides a high-level view on concurrent manner during data retrieval in PostgreSQL with Go and Rust program. The program first establish connection with PostgreSQL database with connection string. Afterwards, it will retrieve different set of data from different database table in concurrent manner. The total elapsed time for entire program execution will be print.

3.1.7 Raw CSV Data Retrieval with Go and Rust program

3.1.7.1 Phase 1 Sequential Program Flowchart

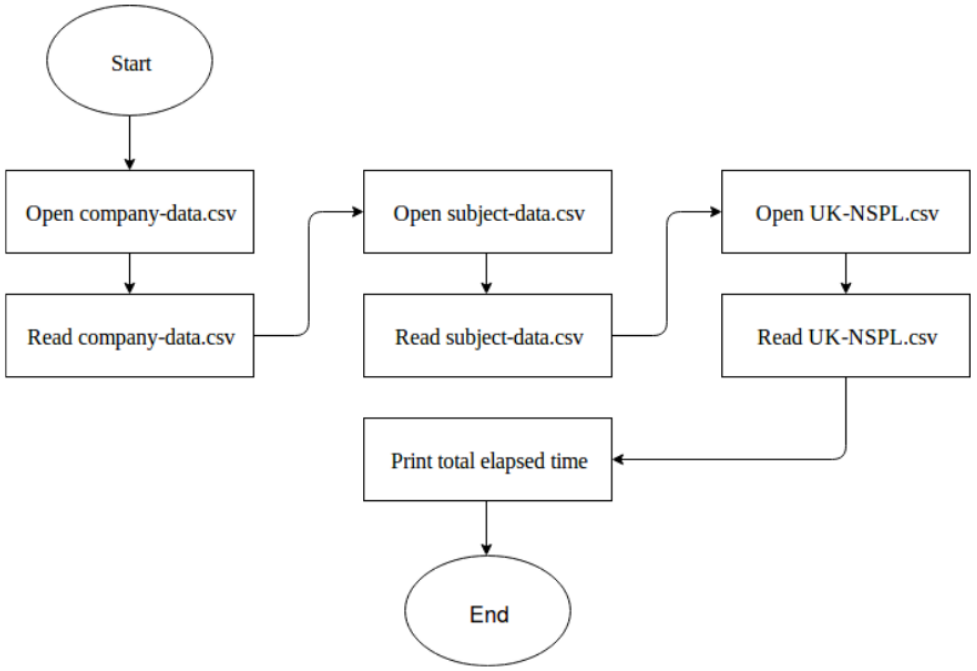


FIGURE 3.7: Phase 1 Sequential program flowchart

The flowchart provides a high-level view on sequential manner on reading CSV file with Go and Rust program. The program will open csv file and read containing data concurrently. The total elapsed time for entire program execution will be print.

3.1.7.2 Phase 1 Concurrent Program Flowchart

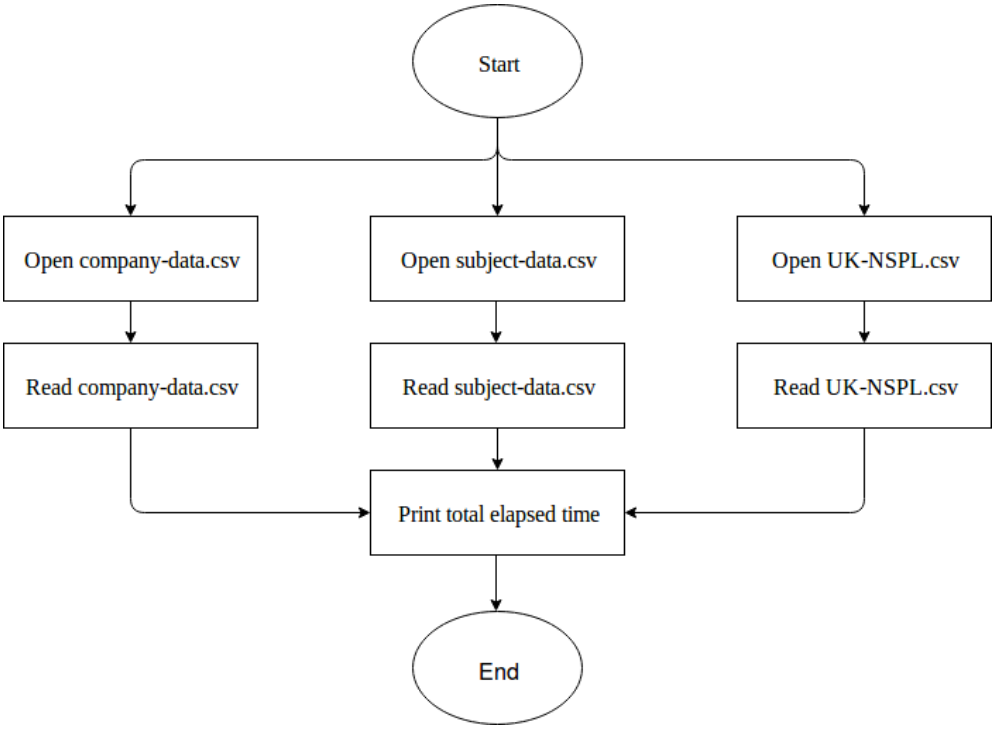


FIGURE 3.8: Phase 1 Concurrent program flowchart

The flowchart provides a high-level view on concurrent manner on reading CSV file with Go and Rust program. The program will open csv file and read containing data in particular order of sequence. The total elapsed time for entire program execution will be print.

3.1.8 Proof of Concept in Phase 1

3.1.8.1 Phase 1 Deployment Diagram

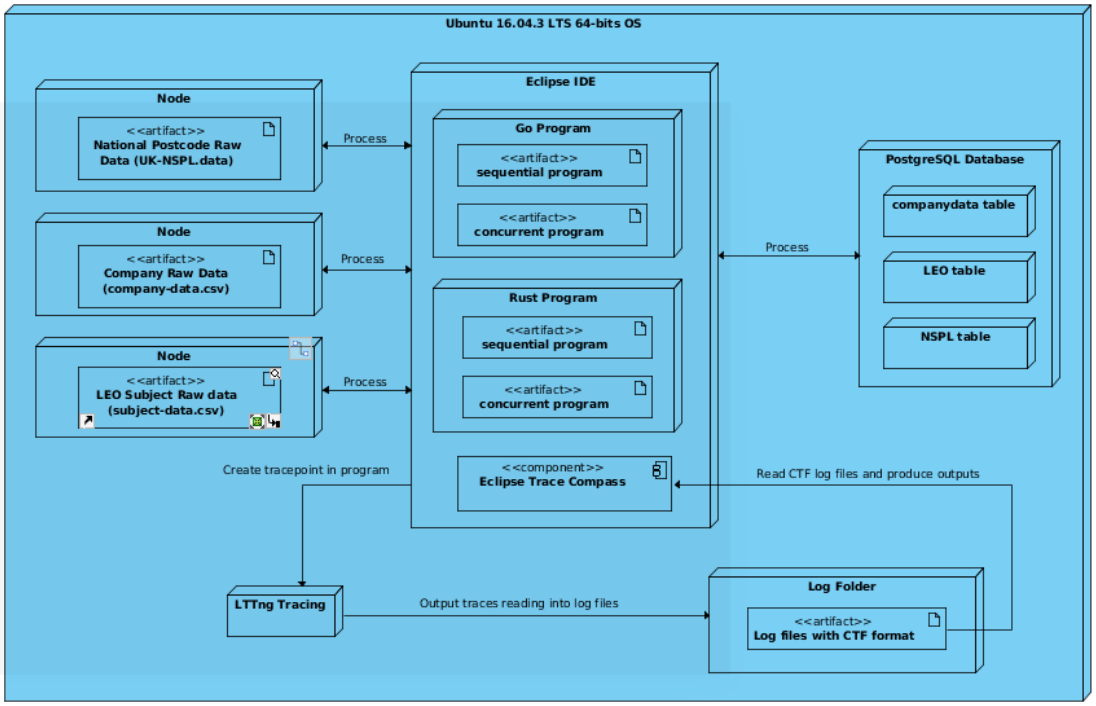


FIGURE 3.9: Phase 1 Deployment Diagram

The deployment diagram describes the proof of concept of phase 1 in specification level and overall architecture of the project. Three database table is created in PostgreSQL database prepare to be processed. Simultaneously, three large data sets are stored in different nodes await to be process or retrieved. The Go and Rust program are written in sequentially and concurrently to process data from CSV file or PostgreSQL database system.

3.2 Phase 2

3.2.1 Introduction

Figure below shows Data Processing Cycle to provide an overview of activities carried out to process big data with the utilization of concurrent programming language and Structure Query Language (SQL).

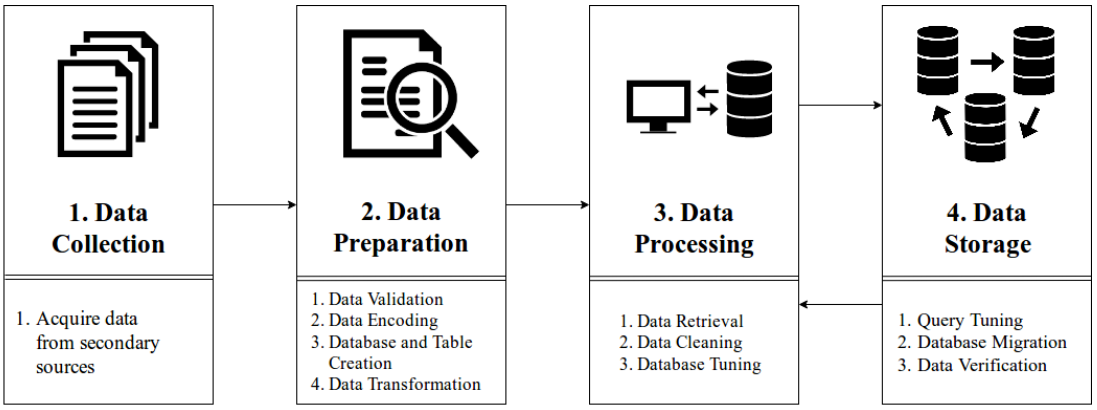


FIGURE 3.10: Data Process Cycle

In Phase 2, we have established an extensive understanding on concurrent language characteristic by utilized the languages' feature on each activity in data processing cycle. The requirement as listed as follow:

1. Data encoding will be conducted with stream editor to convert dirty data into consistent format.
2. Data transformation will be conducted to extracted data from CSV file and import into PostgreSQL database for data handling.
3. Database normalization will be perform to eliminate data redundancy and improve data integrity.

4. The structure of database schema and object (user and tables) will be created with scripts written in Data Definition Language (DDL) of PL/pgSQL (Procecural Language/PostgreSQL).
5. A **sequential** and **concurrent** program will be implemented with Go programming language as an Object Relational Mapping (O/R mapping tool) to convert raw data from CSV data sources into object model, the performance execution will be recorded and compared.
6. A **sequential** and **concurrent** program will be implemented with Go programming language as ORM tool to convert data retrieve from PostgreSQL database into object model, the performance execution will be recorded and compared.
7. A **sequential** and **concurrent** program will be implemented with Rust programming language as ORM tool to convert raw data from CSV data sources into object model, the performance execution will be recorded and compared.
8. A **sequential** and **concurrent** program will be implemented with Rust programming language as ORM tool to convert data retrieve from PostgreSQL database into object model, the performance execution will be recorded and compared.
9. Data cleaning will be performed on CSV raw data to eliminate missing records and standardize the fields in common format.
10. Database tuning will be conducted to configure PostgreSQL database's environment for performance optimization on processing large-scale data and handling workloads.
11. A **sequential** and **concurrent** program will be implemented with Go

programming language as data importation tool to export Company raw data from CSV data sources and import into PostgreSQL database.

12. Query tuning will be conducted to increase query execution performance on data processing.
13. Several **concurrent** program will be implemented with Go programming language as data migration tool to transfer company and NSPL data from legacy storage into normalized table within PostgreSQL database.
14. Data Manipulation Language (DML) scripts will be written with PL/pgSQL to transfer raw data from legacy storage into normalized table within PostgreSQL database.
15. Data verification will be conducted with UNIX command line to check the accuracy and consistency of database records after the data migration is complete.

3.3 Data Encoding

3.3.1 Phase 2 Architecture Diagram

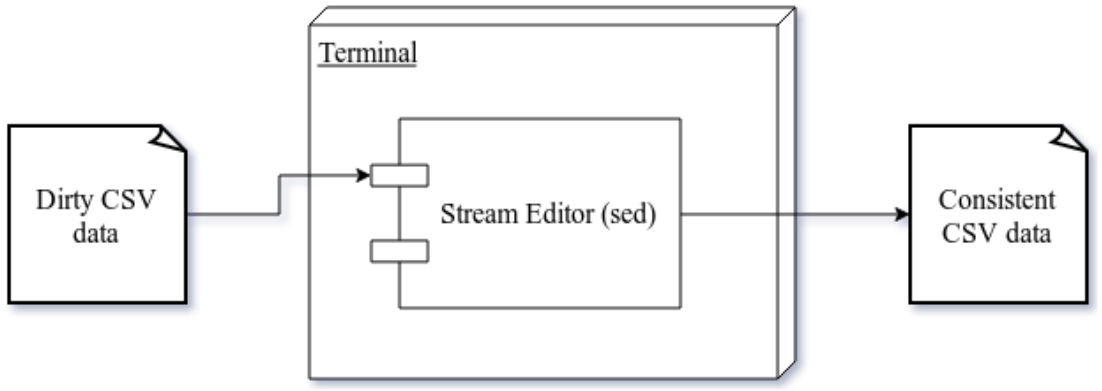


FIGURE 3.11: Data Encoding Architecture Diagram

Data encoding is a conversion of records or fields into specialized format for efficient transformation, importation and migration. [68] Figure 3.14 shows an architecture diagram that describe a high-level view of data encoding flow. The sed stream editor provide powerful feature to perform editing operations coming from a file to remove inconsistency data. [69]

The stream editor allow developer to make editing decisions by calling the commands on terminal. It consumes the dirty raw data as input file and perform text substitution line-by-line based on the text patterns of regular expressions provided in the commands. Ultimately, the encoded file will be output and store into the same directory.

3.4 Data Transformation

3.4.1 Phase 2 Architectural Diagram

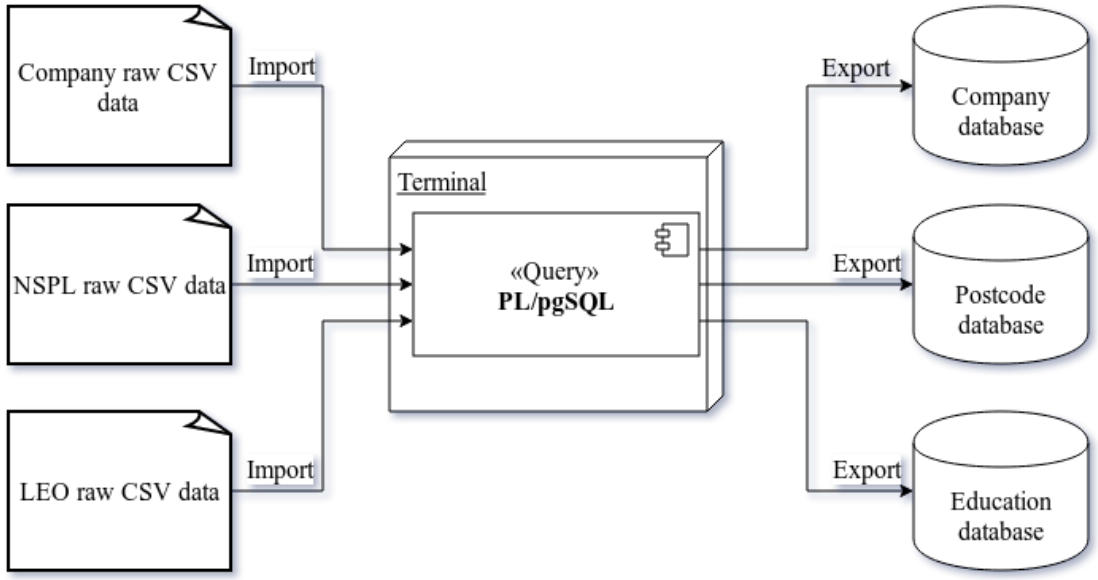


FIGURE 3.12: Data Transformation Architectural Diagram

Data transformation is the process of converting one format to another by extracting from source application into data warehouse. [70]

Figure 3.15 shows the architectural diagram of data transformation process in this project. After the data inconsistency is eliminated with data encoding (performed in Section 3.3), the data in CSV format is extracted and import into PostgreSQL database with PL/pgSQL commands in terminal environment.

3.5 Data Retrieval

3.5.1 Phase 2 Deployment Diagram

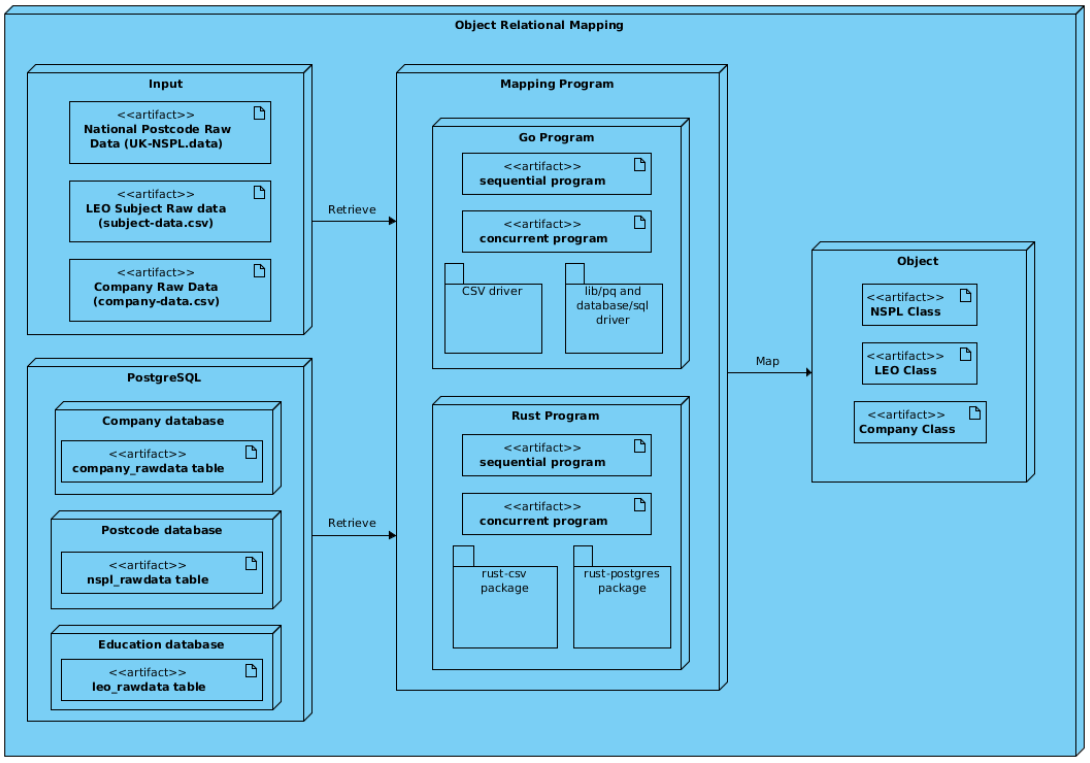


FIGURE 3.13: Data Retrieval with ORM Deployment Diagram

Object-Relational Mapping (ORM) is a technique to manipulate data from database with object-oriented paradigm. The data retrieve from CSV data sources and PostgreSQL database will be convert into object model to ease the manipulation of data in discipline manner. [71] The approach increase usability, flexibility and improve data handling for Data Cleaning and Data Migration.

Figure 3.16 shows the ORM deployment diagram that provide graphic representation of mapping between object and data with mapping program

written in Go and Rust programming language. In this project, we will construct our own ORM tools tool for data retrieval from CSV file and PostgreSQL database with the assistance of CSV package driver, PostgreSQL driver and built-in SQL library from respective language.

All the rows of data will be retrieved from PostgreSQL database and CSV file with Go and Rust's ORM to conduct performance comparison between sequential and concurrent execution and concurrent programming languages' expressive power. The results will be recorded and compared.

3.6 Data Cleaning

3.6.1 Introduction

Data cleaning is the action of detecting and removing missing, incomplete and data redundancy within database. [72] The inconsistencies and incorrect records will be detected in the datasets obtained from the secondary sources because we have lack of control over the data quality.

Data redundancy occurs within a data storage when same piece of data exists in two separate places or two different fields within a single database. Database without normalization will cause updation, deletion and insertion anomalies. The table below is used to understand these the impact of these anomalies on causing data inconsistencies.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

FIGURE 3.14: Student table without normalization

1. **Insertion anomaly.** If student don't enroll any subject and **subject** is a mandatory field, the records cannot be insert into the database without the presence of other attributes or columns.
2. **Deletion anomaly.** If specific student willing to drop a subject, the entire records are forced to delete. As a result, certain attributes or part of the records are lost due to deletion of specific attributes without awareness which leads to missing data.
3. **Updation anomaly.** To update student address in the table, the entire **address** column are required to be updated. If the duplicate records in the database are partially updated, it will leads to data inconsistency.

Therefore, **database normalization** and **data standardization** is conducted to improve the quality and reliability of the datasets.

3.6.2 Database Normalization

3.6.2.1 Introduction

Data normalization is conducted to eliminate data redundancy and improve data integrity. The mentioned method is an approach to remove all the data anomalies and recover the database into consistent state. [73] Normalization is a multi-step approach and require rules to organize data into tabular forms and define relationships among them. The normalization rules and description are listed as follow:

1. **First Normal Form (1NF).** The rule required to eliminate repeating groups, identify primary key and discover **partial dependencies** or **transitive dependencies** among column by determine the determinant of the records.
2. **Second Normal Form (2NF).** The rule required to create new table with primary key assigned for **partial dependencies** elimination.
3. **Third Normal Form (3NF).** The rule required to create new table with primary key assigned for transitive dependencies elimination.

Relational database design is conducted to define entities, attributes, relationships and keys to fulfill normalization rules on eliminating data redundancy. The information contains in raw data are divided and separated into specific table and establish relationship among them to form an organized database. Ultimately, naming conventions and standards are used to form table to increase the usability and maintainability of database.

3.6.2.2 Phase 2 Normalized Company Entity Relationship Diagram

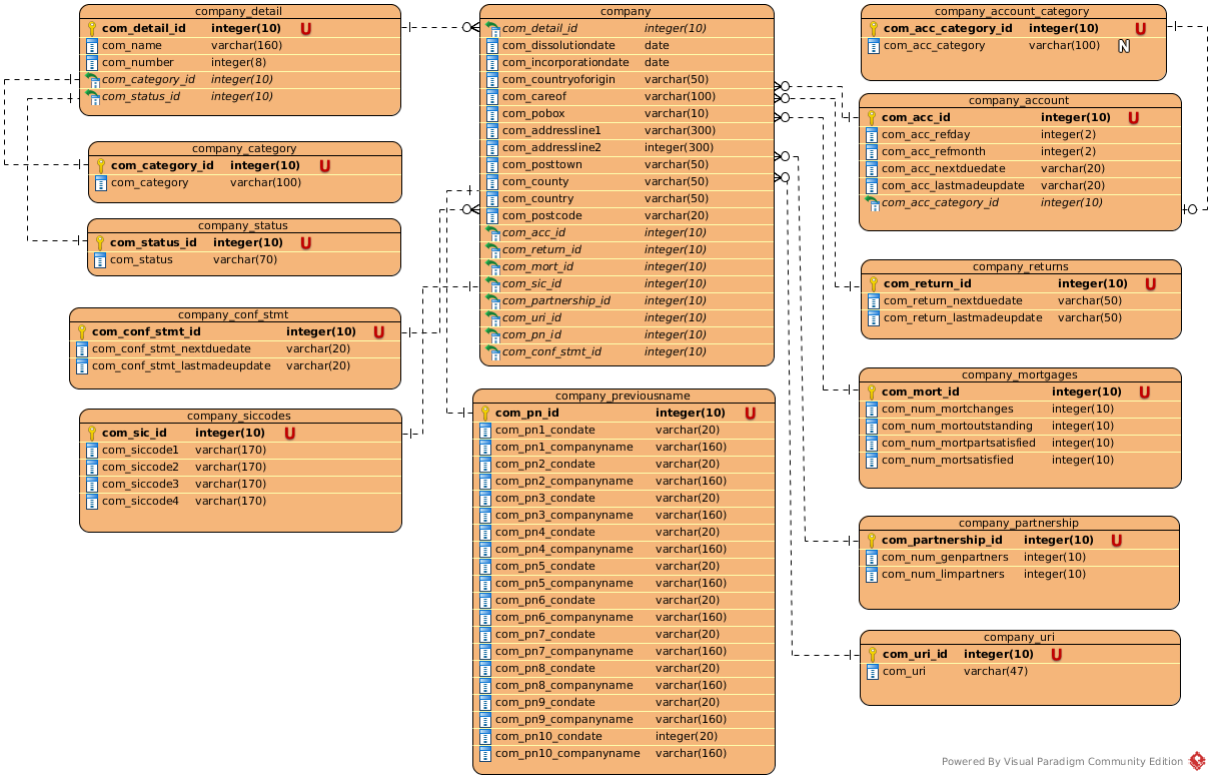


FIGURE 3.15: Company Normalized Database Design

The figure above shows Company's entity relationship diagram (ERD) to provide a graphical representation of normalized database design that display the relationships of entity stored in a database.

3.6.2.3 Phase 2 Normalized Postcode Entity Relationship Diagram

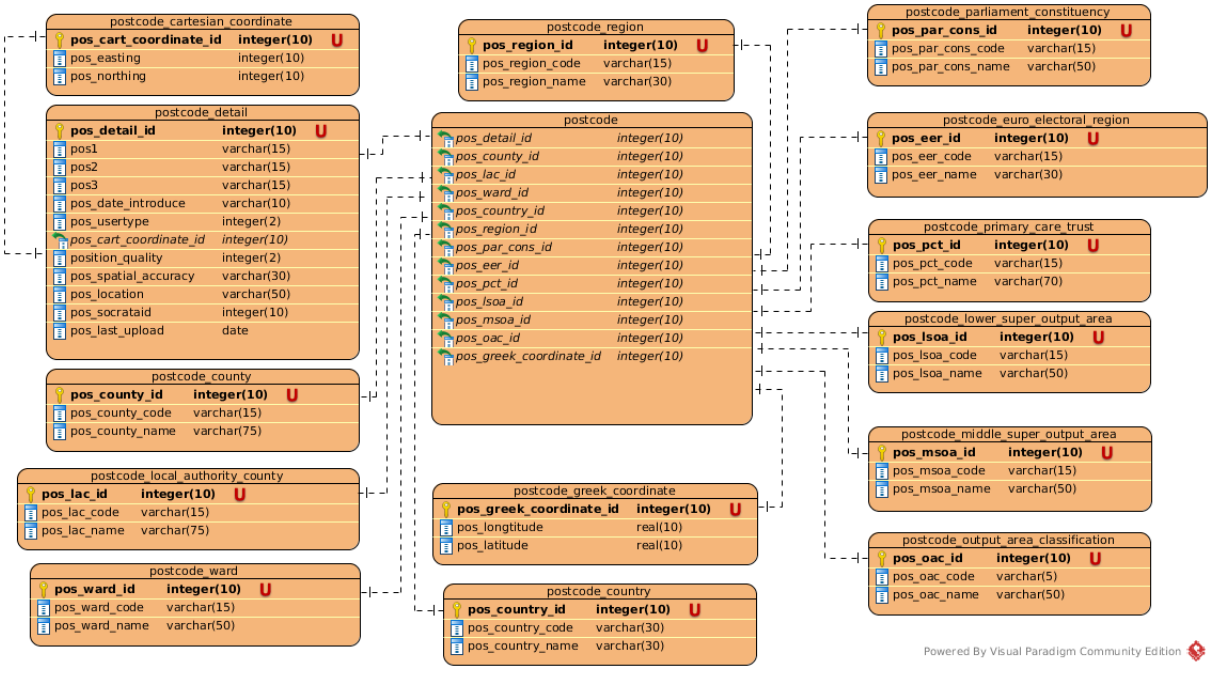


FIGURE 3.16: Postcode Normalized Database Design

The figure above shows Postcode's entity relationship diagram (ERD) to provide a graphical representation of normalized database design that display the relationships of entity stored in a database.

3.6.2.4 Phase 2 Normalized Education Entity Relationship Diagram

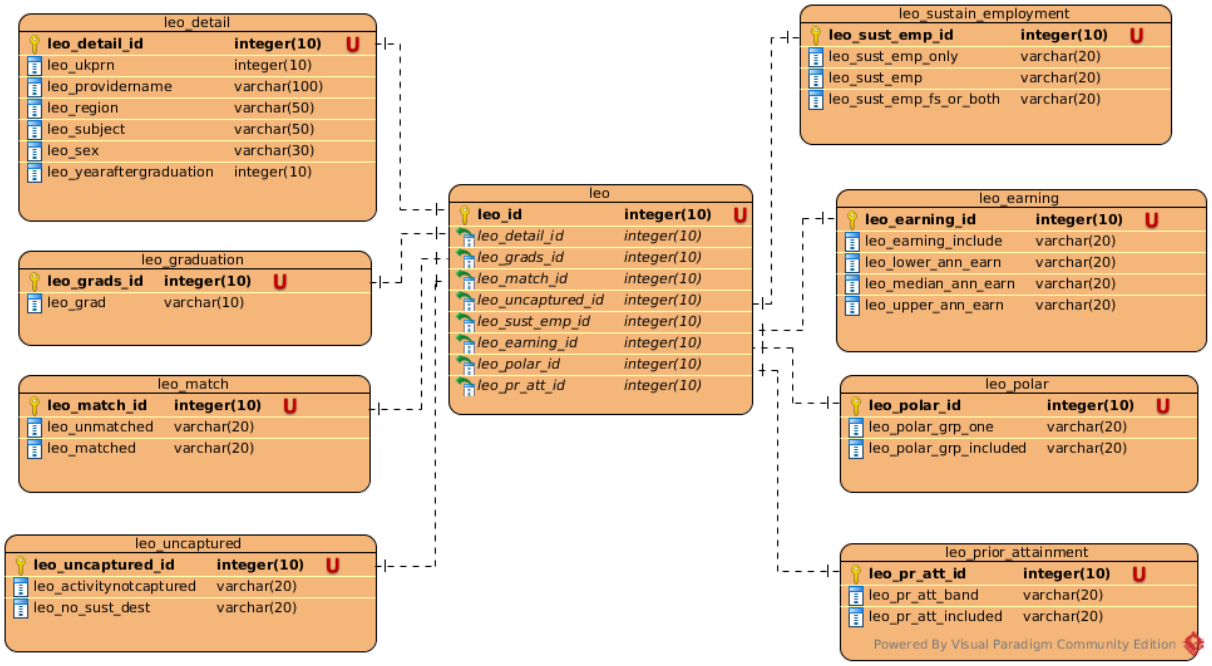


FIGURE 3.17: Education Normalized Database Design

The figure above shows Education's entity relationship diagram (ERD) to provide a graphical representation of normalized database design that display the relationships of entity stored in a database.

3.6.3 Data Cleaning Parser

3.6.3.1 Phase 2 Company Data Cleaning Parser Deployment Diagram

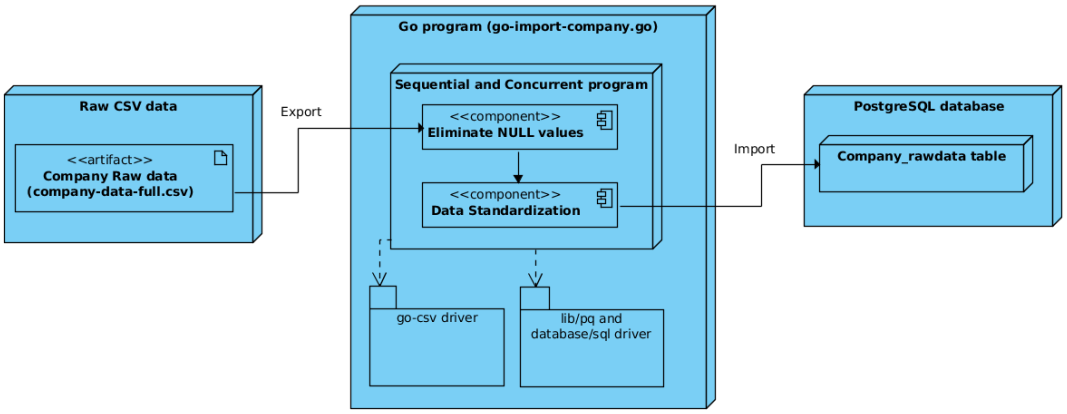


FIGURE 3.18: Company Data Cleaning Parser Deployment Diagram

The figure 3.18 shows the deployment diagram of company data cleaning parser.

The cleaning parser is written with Go program language that consume encoded company raw data (performed in Section 3.3) as input and make execution decisions to eliminate NULL values and perform data standardization to repair missing and incorrect data. Afterwards, the cleaned data will be stored into PostgreSQL database await to be processed. The program work similarly as ORM (mentioned in Section 3.5) by utilizing go-csv driver to retrieve data from CSV files and lib/pq or database/sql driver to establish connection and perform transaction with the PostgreSQL database.

3.7 Database Tuning

3.7.1 Phase 2 Database Tuning Flowchart

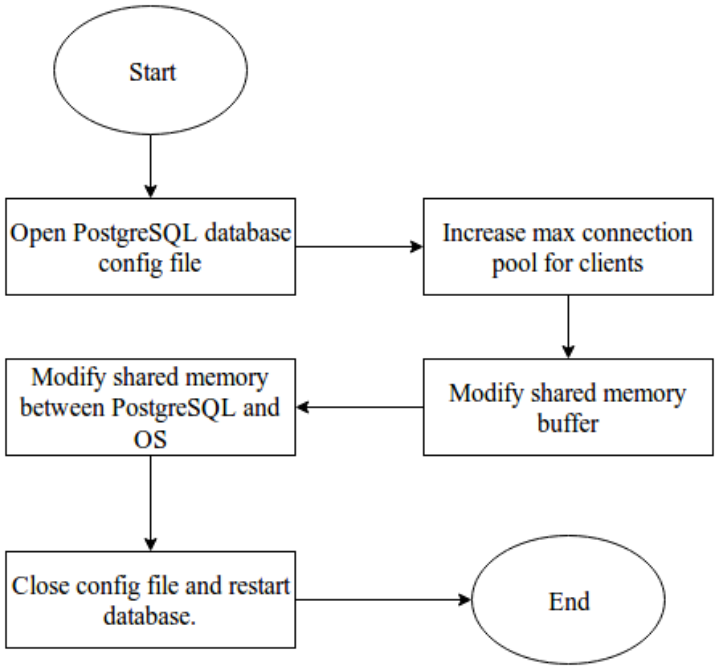


FIGURE 3.19: Database Tuning Flowchart

Database tuning is a process of configure PostgreSQL database's environment to optimize performance by increase throughput and decrease response time. The approach required to open PostgreSQL database configuration file with root access in Linux Operating System environment. The configuration made and reason to perform are describe as follow:

1. **Max Connection.** The number max connection of PostgreSQL database is modified to allow more *Goroutines* from Go program to establish database connection concurrently and perform parallelize transaction.

This modification helps increase performance on Data Cleaning and Data Migration in this project. If the connection pool is not modified, the database system will display FATAL error and terminate the process immediately.

2. **Shared Buffer.** The parameter of shared memory buffer shall be modified as 25 percents of memory in our systems. Increase the amount of memory PostgreSQL database uses for shared memory buffers allow the database to handle extra workloads.
3. **Shared Memory.** The maximum size of shared memory segment shall be modified to allow *Goroutines* or *threads* access to PostgreSQL database simultaneously for better data passing and avoid redundant copies. This configuration parameter determine dedicated memory for PostgreSQL to caching data and increase the space for threads to communicate with the database. The parameter shall be modify with Bytes(B).

Ultimately, restart of PostgreSQL database is required to update the changes and modifications.

3.8 Data Migration

Data migration is the process of transferring data within storage system for database migration. [74] Data migration is extremely challenging as we need to take care of performance issues, data integrity, data consistency and prevent data corruption. The data should be protect carefully and prevent missing during the migration process.

3.8.1 Phase 2 Data Migration Deployment Diagram

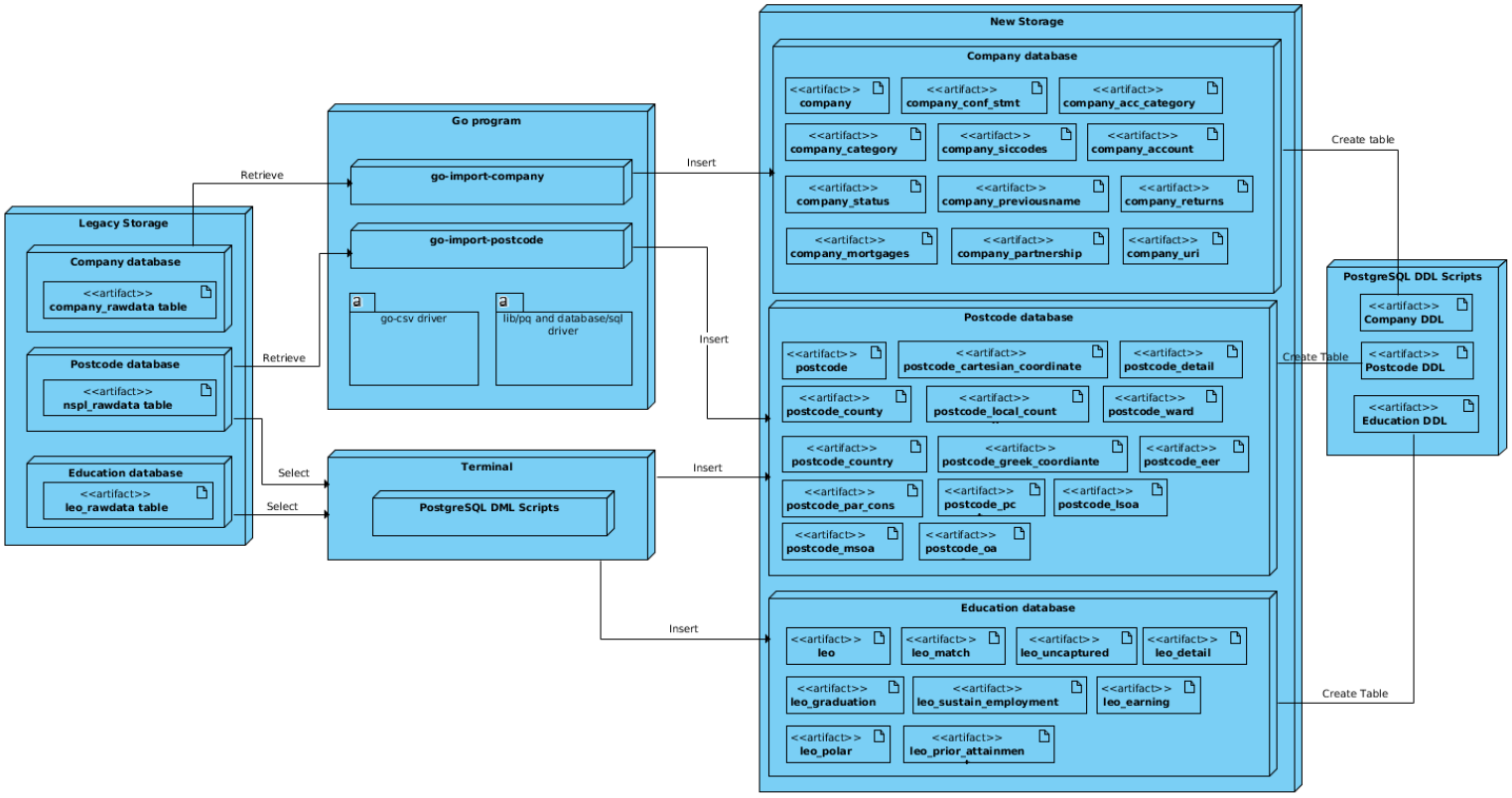


FIGURE 3.20: Data Migration Deployment Diagram

Figure 3.20 shows the deployment diagram of data migration process.

The normalized table in all database are created with PL/pgSQL DDL scripts. Once the creation of table is successful, the data migration of education database is performed with PL/pgSQL DML script running in terminal environment. The mentioned database is migrated with script because it only contains 30000+ rows and its lightweight to be process with queries.

Afterwards, the postcode and company database are migrated from legacy storage to new storage with the execution of scripts and Go program as shown in Figure 3.23. Both company and postcode data are migrated with Go program because it contains more than 4 millions rows in total and its difficult to handle with queries. The unique data is extracted from legacy storage and stored into the normalized table in new storage.

The migration program is written with Go programming language with the inclusion of database/sql driver to establish connection and lib/pq driver to perform transaction with PostgreSQL database. All the migration process does not modify the source data in legacy storage to serve as backup for in case of emergence. In addition, the changes of migration can be easily tracked for verification purposes. Ultimately, the migration duration is recorded and measured.

Chapter 4

Implementation Methodology

4.1 Software Engineering Methodology

Software engineering life cycle (SDLC) is a well structured and iterative sequence of stages in to deliver quality research which meet or exceed project scope. It involves five major activities in this project which are: :

- **Communication.** Student initiate the request to supervisor for apply specific project title offered in this semester. Requirement gathering is conducted in order to discuss the expectation of project and understand the critical factors to achieve project scope or objective. The process required mass amount of communication and collaboration between student and supervisor to ensure requirement are fully understood.
- **Planning.** Project management plan is define and prepare with Gantt Chart to manage project execution by considering risk assessment, resources estimation, time and task management. The tools and

techniques to be used requires to be understand in detail and comprehensive manner to achieve solid understand on whole project execution.

- **Construction.** The creation of project documentation and program through a combination of verification, coding, writing, debugging and testing. The complexity of project are required to be minimize and reduce with the use of standards. The program is construct based on requirement designed in software design phase to ensure the outcomes meet project objectives.
- **Testing.** The project outcomes and deliveries are required to update for supervisor and hand-in to the institution. Documentation and outcomes are required to conform with requirement specification and meet project requirements to ensure the project is doing right.

4.1.1 Prototyping Model Method

The software prototyping method is build prototypes with limited functionality as preliminary design to represent an approximation of concept. The prototype is implemented as proof of concepts for project objectives and reviewed by supervisor to enhance the prototype.

Prototyping helps strengthen understanding the requirement of project through communication and negotiation. The characteristic and basic features of program are demonstrate to collect feedback for enhancement and improvement. This method helps improve familiarity and early determination of requirement specification before development process to reduce chances of fail in the project. Time and project resources can be estimated throughout the process to conduct task and time management in order to deliver the final product.

4.2 Agile Software Methodology

The process decision framework used by this project is Agile Methodology. The mentioned methodology simplified process decisions around incremental and iterative solution delivery, rapid deliver features and update in order to satisfy requirement for weekly project updates. Agile methodology provide flexibility for the project progress respond to change and modification from FYP weekly meeting.

Agile software development describes set of principles for product and technology development under which requirements and solutions evolve through the collaborative effort of self-organizing management. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change according to feedback provide by supervisor. The SDLC or paradigm involved in agile methodology in this project is Kanban.

4.2.1 Kanban

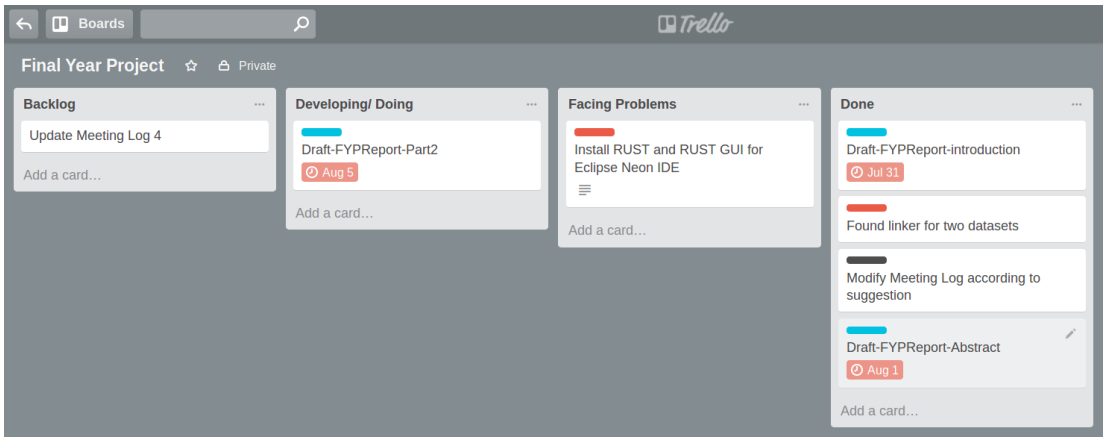


FIGURE 4.1: Kanban board

Kanban provide visual information of workflow by using sticky notes on a whiteboard to create a "picture" of our work. The board allow visualize the project development process or work flows within process and it helps ease the communicate status but also give and receive context for the work. Trello is used in this project as online Kanban board to manage the task in this methodology.

There are an amount of work-in-progress (WIP) on each simple phased process to prevents overproduction and reveals bottlenecks dynamically to aware several roles whether are in bottlenecks. As an example, if the software pipelines are Backlog, Developing, Facing Problems and Done. There are WIP limits on each phased to increase the inspection and create awareness in order to facilitate adaptation based on the work loads.

When a new requirement or changes requested, the task is insert into the backlog. The priority of the task are influenced by time constraint and importance. Afterwards, the task will be move into "developing" to began

construction of documentation or codes. Once the task is encountered difficulty and problem, it will move to “”facing problem”. Alternatively, the task will move to “done” once the task is completed and ready to submit or show to supervisor during meeting.

The Kanban events required to developed immediately and unknown incident may interrupt the progress depends on project feedback and requirement needs. A new high priority fix or changes may requested and it will break off the current project flow. Kanban allow the project respond to change efficiently and provide continuous update on progress to supervisor in order to submit quality works at end of project phase.

4.2.2 Methodology for this Project

In this project, we will be developing Go and Rust program for conduct concurrent and distributed programming. To achieve the required tasks, rapid communication and modification is conducted to improve quality of program and satisfy project objectives. Prototyping method and Kanban will be use in this project.

4.3 Project Infrastructure

4.3.1 List of Hardware Resources

1. **64-bit Personal Computer.** This machine is used for research and development activities of this project. The details are tabulated and shown below:

Processor	8x Intel ® Core (TM) i7-6700HQ CPU @ 2.60 Hz
GPU	NVIDIA GEFORCE GTX960M GDDR4
Memory (RAM)	16330MB, approximately 16GB

FIGURE 4.2: Personal Computer Hardware table

4.3.2 List of Software Resources

1. **Linux Ubuntu 16.04.3 LTS 64-bit.** The community driven and open source operating system is used to conduct concurrent and distributed computing with Go and Rust compiler installed. The details are discussed in Chapter 3.2.1.
2. **Golang language compiler 1.8.3.** The linux amd64 gccgo compiler build Go source code into binary executable with "go build" and run the go program with "go run". It is use to compile and run Go files this project.
3. **Rust language compiler 1.20.0.** The linux amd64 rustc compiler compile Go source code into executable with "rustc". It is use to compile Rust files in this project.

4. **PostgreSQL database 9.5.8.** The open source database management system is use for data handling and data storage for this project. The details are discussed in Chapter 3.2.3.
5. **Eclipse for Parallel Application Developers Oxygen Release (4.7.0) IDE.** The open source IDE provide perspective feature and integrated debugger to ease the coding and development activities for this project. The details are discussed in Chapter 3.2.2.
6. **Goclipse Plugin for Eclipse IDE.** The plugin provide debugging functionality, content assist, auto code indentation, open definition and integrated compiler for Go language on Eclipse IDE.
7. **RustDT Plugin for Eclipse IDE.** The plugin provide syntax highlighting, error reporting, outline support, auto code indentation, debugging functionality and integrated compiler for Rust language on Eclipse IDE.
8. **TeXstudio 2.10.8.** The software provide writing environment for create LaTeX document with numerous feature such as syntax-highlighting, reference checking with bibtex and various assistant. It is use for creating documentation for this project.

9. **Visual Paradigm 14.1 free edition for non-commercial use.** The software is a free Unified Modelling Language Computer-Aided Software Engineering tool support 13 UML diagram types for software design and modelling. It is use to draw diagrams for this project.

4.3.3 Other Project Resources

1. **Synaptic Package Manager.** The software system is a graphical package management program of APT libraries and provide same features as apt-get command. It provide great assist and help on managing software package dependencies. It is installed with *"sudo apt-get install synaptic"* in terminal.
2. **Terminator.** Terminator provide multiple tabs, safe quit, UTF-8 encoding, automatic logging to ease the development activities for developer. The system is required to update source list with *"sudo apt-get update"* and run *"sudo apt-get install terminator"* to install the repository.

4.3.4 Infrastructure Setup and Installation

The required hardware and software resources are listed and discussed in Chapter 4.2.1 and Chapter 4.2.2.

4.3.4.1 Go language compiler installation

1. Ensure Golang go1.8.3.linux-amd64.tar.gz is downloaded using wget in terminal.
2. Ensure downloaded file is extract, move and rename Golang directory.
3. Ensure Golang's compiler export to system path.
4. Ensure Goroot and Gopath is set.
5. Ensure path to user profile .bashrc file is append.
6. Ensure Go executable and Go version installation is success.
7. Ensure Go libraries such as gocode, golint, guru, goimports, gorename and godef into Gopath directory are installed.
8. Ensure Godef Gometalinter is downloaded and executed.

The full installation steps for Go language compiler is found in Appendix A.1.

4.3.4.2 RUST language compiler installation

1. Install Rust toolchain with command line.
2. Export rust executable to system path.
3. Install Racer, Rustfmt, Rainicorn.
4. Ensure all the required Rust executables are installed.

The full installation steps for RUST language compiler is found in Appendix A.2.

4.3.4.3 Eclipse IDE installation

1. Ensure Java is installed before start download Eclipse.
2. Run *"sudo apt-get update"* and *"sudo apt-get upgrade"* before start download.
3. Make eclipse-workspace folder as default storage for better management.

The installation details for Eclipse IDE is found in Appendix A.3.

4.3.4.4 GoClipse plugin for Eclipse IDE installation

1. Install Goclipse plugin with Eclipse marketplace.
2. Ensure Goclipse preferences and setting are correct.

The full installation steps for Goclipse plugin on Eclipse IDE is found in Appendix A.4.

4.3.4.5 RustDT plugin for Eclipse IDE installation

1. Install RustDT plugin with Eclipse marketplace.
2. Ensure RustDT preferences and setting are correct.

The full installation steps for RustDT plugin on Eclipse IDE is found in Appendix A.5.

4.3.4.6 PostgreSQL database installation and setup

1. Install postgreSQL in command line.
2. Ensure database for FYP1 is created.
3. Create new user for database.
4. Ensure database connection is established with user access.

The full installation steps for PostgreSQL database is found in Appendix A.6.

Chapter 5

Implementation Plan

5.1 Project Task Identification

5.1.1 Identification of Critical Success Factors

Critical success factors are a key requirement which is necessary and essential to be identified to achieve the project objectives in this project. The requirement for our design objectives are listed below:

1. **Determine a suitable operating system.** The operating system should be reliable, secure and appropriate for data processing, concurrent and distributed computing activities. If the selected operating system does not meet requirements, a new operating system has to be considered.

2. **Acquire free public data set for big data processing.** Large data set is required for data processing with concurrent and distributed computing to make use of concurrent programming language's package

and architecture. If the data set obtains not clean and useful, data cleansing and data deduplication have to be conducted.

3. **Selection of database management system (DBMS).** The database-management system for this project should support for operating system, concurrent programming language and project activities. If the selected DBMS does not compatible and suitable, a new DBMS capability has to be considered.

4. **Installation and setup DBMS for big data handling.** The selected database-management system should be installed and running on the operating system for data storing and data handling. The database system allows developer to conduct development activities for manage concurrency control for update and retrieval in this project.

5. **Selection of Go and RUST concurrent programming language for comparison.** There are many types of concurrent programming language for system development. The selected language for this project is RUST and Go. This programming language architecture, packages and capabilities should be considered to conduct performance comparison.

6. **Coding of "Import CSV into database" with Go program.** The program is required to write with Go language to read CSV and upload into PostgreSQL database. This task is conduct for data definition and data preparation before data processing is performed.

7. **Coding of "Import CSV into database" with RUST program.** The program is required to write with Go language in order to read CSV and upload into PostgreSQL database. This task is conduct for data definition and data preparation before data processing is performed.

8. **Conduct minor comparison on sequential and concurrent programming with Go and RUST language on PostgreSQL database transaction.** The sequential and concurrent program is required to write with Go and RUST language in order to conduct a comparison of execution time for database retrieval on PostgreSQL.

9. **Conduct minor comparison on sequential and concurrent programming with Go and RUST language on reading CSV files.** The sequential and concurrent program is required to write with Go and RUST language to conduct a comparison of execution time on reading CSV files.

10. **Installation of LTTng tracing network on user application or Linux kernel to produce outcomes into log files.** The open source software tracing toolkits enable the developer to create a tracepoint in Linux kernel or user applications to obtain process reading and create output into log files as Common Trace Format (CTF). This task has to be completed to improve troubleshooting and debugging process.

11. **Install Eclipse Trace Compass to extract and read Common Trace Format information from log files.** The open source Eclipse IDE plugin read CTF files and produce useful graphical and tabulated information from traces. This task has to be completed to improve debugging process and analyse process behaviour.

5.1.2 Project Tasks for FYP Phase 1

1. Installation of Ubuntu 16.04 LTS 64-bit operating system.
2. Acquire free public data set for big data processing.
3. Installation of Eclipse Parallel Application IDE Parallel Oxygen version.
4. Selection of Go and RUST concurrent programming language for comparison.
5. Installation of Go language compiler and Goclipse plugin for Eclipse IDE.
6. Installation of RUST language compiler and RustDT plugin for Eclipse IDE.
7. Selection of PostgreSQL object-oriented relational database management system (OORDBMS).
8. Installation and setup PostgreSQL database system intro PC for data handling.
9. Golang programming for import CSV files into PostgreSQL database.
10. Sequential and concurrent programming with Golang on PostgreSQL database retrieval.
11. Sequential and concurrent programming with Golang on reading CSV files.
12. Big data checking, cleaning and preparation with Devil Advocate.
13. Installation of LTTng tracing network on user application or linux kernel.
14. Install Eclipse Trace Compass to extract and read Common Trace Format information from log files.

5.1.3 Gantt Chart for Phase 1

	Comm	Task Name	Start	Finish	Duration
1		**Final Year Project Phase 1**	**07/13/17**	**09/18/17**	**68d**
2		**Project Identification**	**07/13/17**	**07/16/17**	**4d**
3		Project Objective	07/13/17	07/14/17	2d
4	**CS**	Select Go and Rust as language comparison	07/14/17	07/15/17	2d
5		Project Deliverables	07/15/17	07/16/17	2d
6		**Write Project Documentation Chapter 1**	**07/16/17**	**07/20/17**	**5d**
7	**CS**	**Acquire Free Large Dataset**	**07/20/17**	**07/23/17**	**4d**
8	**CS**	Acquire Basic Company Dataset	07/20/17	07/21/17	2d
9	**CS**	Acquire Education dataset	07/21/17	07/22/17	2d
10	**CS**	Acquire Postcode dataset	07/22/17	07/23/17	2d
11		**Write Project Documentation Chapter 2**	07/24/17	07/28/17	5d
12		**Project Resource Installation**	**07/28/17**	**08/03/17**	**7d**
13		Install Eclipse Oxygen IDE	07/28/17	07/29/17	2d
14		Install Go compiler and Goclipse plugins	07/29/17	07/31/17	3d
15		Install Rust compiler and RUSTDT plugin	07/31/17	08/02/17	3d
16		Install PostgreSQL database	08/02/17	08/03/17	2d
17		**Write Project Documentation Chapter 4**	**08/04/17**	**08/09/17**	**6d**
18		**Write Project Documentation Chapter 5**	**08/09/17**	**08/14/17**	**6d**
19	**CS**	**Data Checking, cleaning and preparation with Devil Advocate Test**	**08/15/17**	**08/17/17**	**3d**
20		**Database table and user creation**	**08/18/17**	**08/20/17**	**3d**
21	**CS**	**Import CSV into PostgreSQL database**	**08/20/17**	**08/22/17**	**3d**
22	**CS**	**Go Programming on PostgreSQL database**	**08/22/17**	**08/28/17**	**7d**
23	**CS**	Write sequential program	08/22/17	08/25/17	4d
24	**CS**	Write concurrent program	08/25/17	08/28/17	4d
25	**CS**	**Go Programming on read CSV files**	**08/28/17**	**09/03/17**	**7d**
26	**CS**	Write sequential program	08/28/17	08/31/17	4d
27	**CS**	Write concurrent program	08/31/17	09/03/17	4d
28		**Write Project Documentation Chapter 3**	**09/03/17**	**09/06/17**	**4d**
29	**CS**	**Install LTTng Tracing Network**	**09/06/17**	**09/07/17**	**2d**
30	**CS**	**Install Eclipse Trace Compass**	**09/07/17**	**09/08/17**	**2d**
31	**CS**	**Conduct performance comparison**	**09/08/17**	**09/11/17**	**4d**
32		**Finalize Documentation**	**09/11/17**	**09/18/17**	**8d**

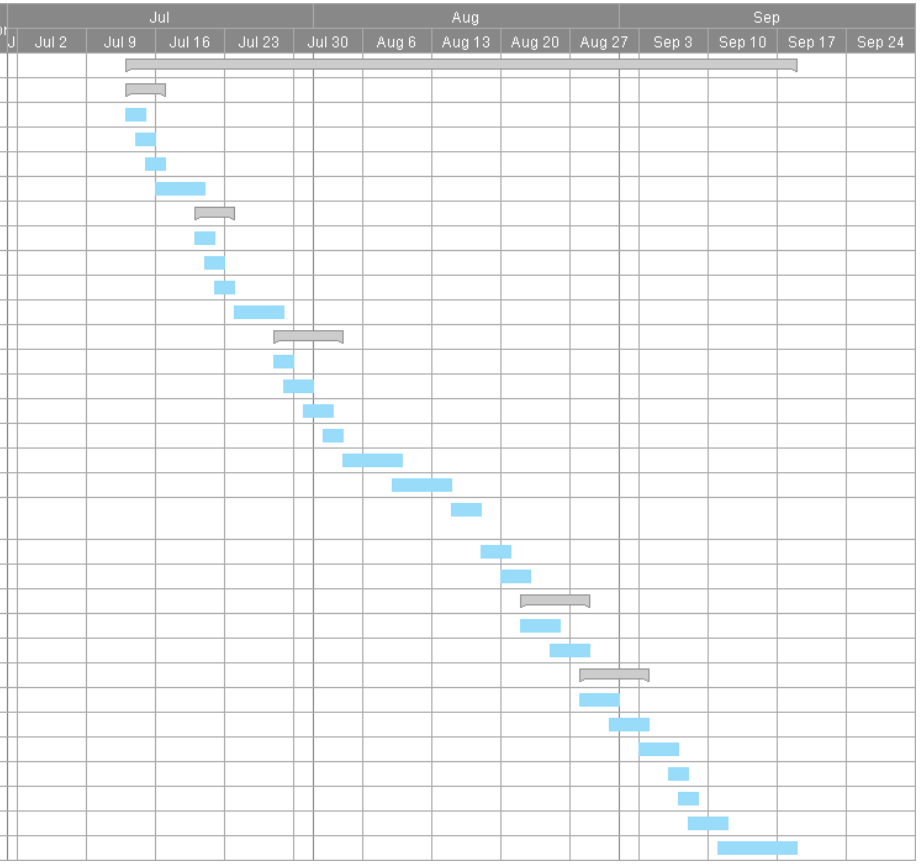
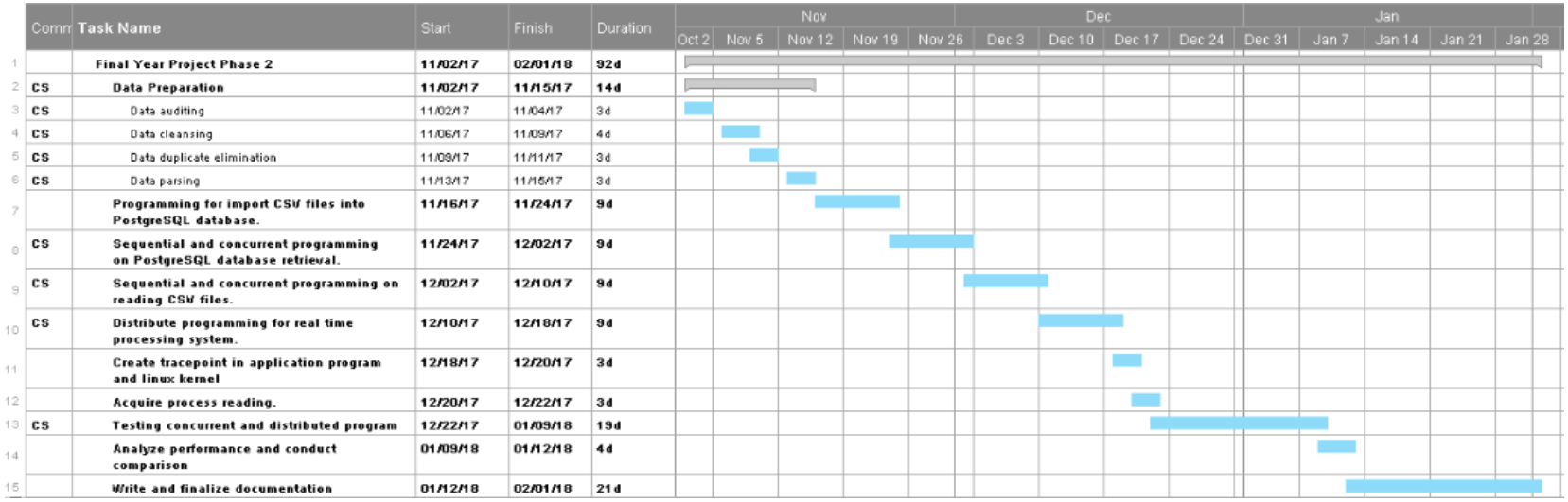


FIGURE 5.1: Gantt Chart for Phase 1

5.1.4 Project Tasks for FYP Phase 2

1. Data encoding.
2. Data transformation.
3. Data parsing.
4. Data cleansing.
5. Data normalization.
6. Database tuning.
7. Query tuning.
8. Data migration.
9. Sequential and concurrent programming with Go and RUST on PostgreSQL database retrieval.
10. Sequential and concurrent programming with Go and RUST on reading CSV files.

5.1.5 Gantt Chart for Phase 2



	Comn	Task Name	Start	Finish	Duration
1		**Final Year Project Phase 2**	**11/02/17**	**02/01/18**	**92d**
2	**CS**	**Data Preparation**	**11/02/17**	**11/15/17**	**14d**
3	**CS**	Data auditing	11/02/17	11/04/17	3d
4	**CS**	Data cleansing	11/06/17	11/09/17	4d
5	**CS**	Data duplicate elimination	11/08/17	11/11/17	3d
6	**CS**	Data parsing	11/13/17	11/15/17	3d
7		**Programming for import CSV files into PostgreSQL database.**	**11/16/17**	**11/24/17**	**9d**
8	**CS**	**Sequential and concurrent programming on PostgreSQL database retrieval.**	**11/24/17**	**12/02/17**	**9d**
9	**CS**	**Sequential and concurrent programming on reading CSV files.**	**12/02/17**	**12/10/17**	**9d**
10	**CS**	**Distribute programming for real time processing system.**	**12/10/17**	**12/18/17**	**9d**
11		**Create tracepoint in application program and linux kernel**	**12/18/17**	**12/20/17**	**3d**
12		**Acquire process reading.**	**12/20/17**	**12/22/17**	**3d**
13	**CS**	**Testing concurrent and distributed program**	**12/22/17**	**01/09/18**	**19d**
14		**Analyze performance and conduct comparison**	**01/09/18**	**01/12/18**	**4d**
15		**Write and finalize documentation**	**01/12/18**	**02/01/18**	**21d**

FIGURE 5.2: Gantt Chart for Phase 2

5.1.6 Milestone Deliverables

The milestone deliverables are:

1. Go program for data parsing, object relational mapping and data migration.
2. RUST program for data parsing and object relational mapping.
3. PL/pgSQL's DDL, DML and DCL scripts for database creation, manipulation and migration control.
4. A report based of this project.

5.2 Planned Execution Activities

5.2.1 Phase 1

1. **Data Validation.** The Data Validation is conducted to ensure obtained raw CSV data set is clean and useful. The expected result of this test is the number of commas in the record should not exceed the number of columns in a database. In addition, the data content itself should be unique and suitable for storing in the database. More information is provided in Appendix B.1.
2. **Golang programming for import CSV files into PostgreSQL database.** The Golang programming for import CSV raw data into PostgreSQL is to ensure Go language is capable of processing raw CSV data and PostgreSQL database. The expected result for this program

should read 100 rows of data from raw CSV file and insert into PostgreSQL database. More information is provided in Appendix C.

3. **Sequential and concurrent programming with Golang on PostgreSQL database retrieval.** The Go program should retrieve 300 rows of data from three tables (each table 100 rows) in PostgreSQL database sequentially and concurrently. The expected result for this program is concurrent processing should have better performance than sequential. More information is provided in Appendix D.
4. **Sequential and concurrent programming with Golang on reading CSV files.** The Go program should retrieve 100 rows of data from raw CSV file sequentially and concurrently. The expected result for this program is concurrent processing should have better performance than sequential. More information is provided in Appendix E.

5.2.2 Phase 2

1. **Data encoding.** This activity is a deliverable of Phase 2 in this project. It is conducted to ensure that the dirty and corrupted datasets are converted into consistent format so that it will be safe to used for Object Relational Mapping, Data Transformation and Data Parsing.
2. **Development of PL/pgSQL scripts for data transformation.** This activity is a deliverable of Phase 2 in this project. It is conducted to extract data in CSV format from raw datasets and import into PostgreSQL database.
3. **Development of Go and Rust Object Relational Mapping (ORM) program for data retrieval.** This activity is a deliverable of Phase 2 in this project. The data from CSV file and PostgreSQL database are retrieved and map into object model. Go and Rust program should

retrieve 4 millions row of data from raw CSV file and PostgreSQL database in sequential and concurrent manner. The execution duration of each program are tabulated and recorded for comparison purposes.

4. **Development of PL/pgSQL DDL scripts for normalized entity creation.** This activity is a deliverable of Phase 2 in this project. It can eliminate redundancy and data anomalies to improve data integrity. Database design is performed to define table and establish relationship between entity to create a relational database schema. Moreover, normalized table will be created correctly with PL/pgSQL's DDL scripts based on the Entity Relationship Diagram shown in Section 3.
5. **Development of Go data parser program.** This activity is a deliverable of Phase 2 in this project. The missing fields will be eliminated and data standardization is conducted to promote conformity and usability of data.
6. **Database tuning.** This activity is a deliverable of Phase 2 in this project. It is performed to configure PostgreSQL's database environment and setting to increase performance on data processing.
7. **Query tuning.** This activity is conducted to optimize database transaction performance and avoid poor performance execution on data processing activities.
8. **Development of PL/pgSQL's DML, DCL scripts and Go concurrent program for database migration.** This activity is a deliverable of Phase 2 in this project. The data that are transformed and cleaned will be import into normalized table. These data are migrated from legacy storage into new storage within PostgreSQL database.

Chapter 6

Results and Findings

6.1 Phase 1

1. **Data validation.** This activity has been successfully achieved. It has been found the method can detect unmatched numbers of commas, unsuitable data types during data importation from CSV to PostgreSQL database and identify the uniqueness of rows and columns in data. Results and detailed information is provided in Appendix B.2 to B.4.
2. **Golang programming for import CSV files into PostgreSQL database.** This activity has been successfully achieved. The program is capable to read 100 rows of data from three datasets and import into PostgreSQL database. Results and detailed information is provided in Appendix H.

3. **Sequential and concurrent programming with Golang on PostgreSQL database retrieval.** This activity has been successfully achieved. The program is capable to prove concurrent processing is faster than sequential in data retrieval with PostgreSQL database. Results and detailed information is provided in Appendix G.
4. **Sequential and concurrent programming with Golang on reading CSV files.** This activity has been successfully achieved. The program is capable to prove concurrent processing is faster than sequential in reading CSV data. Results and detailed information is provided in Appendix F.

6.2 Phase 2

1. **Data encoding.** This activity has been successfully achieved. The dirty and corrupted CSV raw datasets can be converted into consistent format with stream editor.
2. **Development of PL/pgSQL scripts for data transformation.** This activity has been successfully achieved. The developed scripts is capable to extract data from CSV format from raw datasets and import into PostgreSQL database.
3. **Development of Go and Rust Object Relational Mapping (ORM) program for data retrieval.** This activity has been successfully achieved. The Go and Rust program developed is capable to retrieved data from CSV file and PostgreSQL database and map into object model in sequential and concurrent manner. Other than that, the

activity proves concurrent processing is faster than sequential in data retrieval with PostgreSQL database and reading CSV data.

4. **Development of PL/pgSQL DDL scripts for normalized entity creation.** This activity has been successfully achieved. The database design is capable to define table and establish relationship between entity. In addition, the PL/pgSQL's DDL scripts developed is able to create database entity based on the database design correctly.
5. **Development of Go data parser program.** This activity has been successfully achieved. The developed Go program is capable to eliminate NULL values and standardize the records of specific columns to promote conformity and usability of data.
6. **Database Tuning.** This activity has been successfully achieved. The number of database maximum connections, amount of shared buffer utilized and maximum of shared memory segments are configured to increase performance and transaction efficiency of concurrent program.
7. **Query Tuning.** This activity has been successfully achieved. The weaknesses of query are identified and poor performance query execution are avoided on data processing activities.
8. **Development of PL/pgSQL's DML, DCL scripts and Go concurrent program for database migration.** This activity has been successfully achieved. The PL/pgSQL's DML, DCL scripts is capable to retrieve unique data from legacy storage and insert into normalize table. In addition, the scripts and Go program are capable to migrate more than 4 millions row of data into normalized table without causing missing of records.

Chapter 7

Comparison Discussion and Recommendations

7.1 Problems Encountered & Overcoming Them

7.1.1 Acquisition of free large datasets for data processing

The problem encountered during data gathering of this project is difficulty on finding suitable free big data from websites. It is a challenge to find problem and raise question by going into data details. It took huge amount of time to understand the focus of project and gather desired data for problem solving.

With the help of supervisor, I had successfully obtained suitable datasets for this project. He provides guidance and helping hand to clear my doubts and

confusion by suggests several website and introduce various data repositories during the meeting.

7.1.2 Goclipse plugin compile error

Eclipse IDE could not compile and build my Go files, this is because the IDE couldn't find GOROOT in usr/local/go. The development activities cannot proceed and face impediment on executing critical success factors. The cause of the problem is Golang compiler executable doesn't possess a copy in usr/local/go, which caused Eclipse fail to compile Go file because couldn't file the compiler.

The problem is resolved with help of supervisors, he guides me to execute Linux command line to resolve the problem during FYP meeting. Moreover, he helps identify the root cause of problem with Google Hangout in the midnights.

7.1.3 Unclear and doubts on writing documentation

The problem encountered during writing documentation is unclear about the purpose and objectives of each section which leads to messy and poor content deliveries in writing. A certain standard and requirement should be achieved in writing the FYP document.

The problem is resolved with the help of supervisor as he patiently guide us to arrange the content layout of document and writing citation with references.

7.1.4 Difficulty on understand concurrent programming concepts

The problem encountered during coding process is to understand concurrent concepts. It took an enormous amount of time to implement the ideas of Goroutine and Go channel into the program to achieve concurrency with Go programming language. This is because I do not possess the experiences and knowledge to build a concurrent program.

The problem is resolved with the help of official documentation and StackOverflow websites which provide clear explanation and enlightenment for me to understand the concepts and semantics of languages.

7.1.5 Difficulty on develop PG/pgSQL scripts on data migration.

The problem encountered during development process is writing PG/pgSQL scripts to perform data migration. The DML query requires to use *insert with select* query to retrieve primary key from each entity and insert as foreign key into specific table. In addition, the query is shall possess high throughput on data processing while maintaing the data consistency and validity during the migration process. The mentioned difficulty has caused impediment on development progress and stuck for a week.

The problem is posted on Stackoverflow forum as database question and it was discussed by various database expert with high reputation in the community.

Ultimately, the issue is resolved with suggested answer provided and the query with JOIN works on my project.

7.1.6 Difficulty on perform database tuning.

The problem encountered during database tuning are listed as follow:

1. **Understand the risk of modification.** Modified number of maximum concurrent connection, parameter of shared memory buffer and maximum size of shared memory segment utilized by PostgreSQL database could result in database corruption and data loss. The PostgreSQL will running inconsistently and caused freezing or termination of any process if one of these value are not configure correctly.
2. **Limitation of knowledge on database system configuration.** Database tuning is an advanced techniques and incredibly difficult task perform by database administrator in mid-sized and large company to configure the database environment for situational usage. The process require deep understanding on hardware memory resources and database concepts to prevent the error on resource management between system and database.
3. **Performance bottleneck of programming langugae with database.** Each programming language utilized threads and stack differently. The maximum number of OS stacks and OS threads allow the language to utilized shall be carefully inspected and measure to prevent crash during the runtime. As an example, Go programming language only

allow 1 GB of stack utilized on 64-bit system which indicates it only allow 10,000 threads (1,000,000 goroutines)to be assigned in each execution.

The understanding is established and discovered from Go official documentation and PostgreSQL 9.5 documentation to resolve this problem. The information provided in the documentation is clear and easy to be learnt as it helps resolve all the problem mentioned.

7.1.7 Distributed Programming

The project objectives had been reduce to concurrent programming on data processing instead of distributed programming.

It is possible to perform distributed programming on data processing activities in this project. However, the development required extra duration on experimentation, design and testing. Based on my current understanding and knowledge on the subject, it will require extra 3 months to develop distributed programming based program.

7.2 Execution time performance comparisons

7.2.1 Performance comparisons of Golang process PostgreSQL database

Table G.2 in Appendix G shows the total elapsed time for Go sequential program and Go concurrent program to retrieve 300 rows of data from

PostgreSQL. Real refers to actual elapsed time for the program; user and sys refer to CPU time used by process. Fmt.Println() of data retrieval is removed during the execution to obtain accurate results on program performance. It is found that concurrent programming is faster than sequential programming on process data from PostgreSQL database.

However, the amount of CPU time spent in the kernel within the process (sys) by concurrent program is higher than sequential program. This is probably caused by utilization of hardware resources when goroutine and gochannel are communicating in the process.

7.2.2 Performance comparisons of Golang process raw CSV files

Table F.1 in Appendix F shows the total elapsed time for Go sequential program and Go concurrent program to retrieve 300 rows from raw CSV data. Real refers to actual elapsed time for the program; user and sys refer to CPU time used by process. Fmt.Println() of data retrieval is removed during the execution to obtain accurate results on program performance. It is found that concurrent programming is faster than sequential programming on process data from raw CSV files.

The optimization to implementations in textitencoding/csv has improved [75] and fixed slow reading problems in Go version 1.8. The build-in CSV library works blazingly well with bufio.Reader() to split the commas during the read CSV files process.

Chapter 8

Conclusions

8.1 Conclusions

In phase 1, we have review many concepts and addressed the details of concurrent programming language concepts.

The project objectives for Phase 1 are:

1. To learn and understand about Go and RUST programming language concepts and their concurrent processing features.
2. To conduct a comparison on Go programming language concepts in processing big data with different techniques.
3. To implement the handling of big data with PostgreSQL, an object-oriented relational database management system (OORDBMS)

What we have achieved on Phase 1:

1. We reviewed different concepts and characteristics of concurrent programming language.
2. We established the fundamentals of concurrent programming knowledge and possess confident advance to the next phase of development.
3. We established a development platform for concurrency programming.
4. We demonstrated the capability of concurrent programming language, which is provide better performance and throughput on data processing compare to sequential programming with results.

The project objectives for Phase 2 are:

1. To learn and understand the importance of data processing activities in data process cycle.
2. To understand the limitation of concurrent programming language and PostgreSQL database resource utilization.
3. To perform text substitution with data encoding on eliminate incompatible data type on data source.
4. To implement Go and Rust concurrent programming features on data processing activities.
5. To perform database cleansing on eliminate defects and error found in big data.
6. To develop Go and Rust program as ORM tool on data retrieval from CSV file and PostgreSQL database and map into object model.
7. To conduct database and query tuning to optimize performance on data processing execution.

8. To produce PL/pgSQL's DDL, DML and DCL scripts for database entity creation and database migration.
9. To develop a Go programming based data migration system to transfer data from legacy storage into new storage within PostgreSQL database.

What we have achieved on Phase 2:

1. We understand the purpose and benefits of each data processing activities in data process cycle.
2. We established understanding of limitation of concurrent programming language and PostgreSQL database resource utilization to prevent crashes in system execution.
3. We demonstrated the capabilities of data encoding on text substitution of raw CSV files with regular expression as input command.
4. We demonstrated strength and limitation of Go and Rust concurrent programming features on data retrieval through execution times and language structure.
5. We have implemented Go program to eliminate NULL values in every single row and perform data standardization to increase usability of data.
6. We have conducted database normalization to eliminate data redundancy, resolve anomalies and improve data integrity.
7. We have implemented Go and Rust program as ORM tool to retrieve 4 millions of data from CSV file and PostgreSQL database in sequential and concurrent manner.
8. We have conducted database and query tuning to optimize data processing performance and allow more threads to establish connection with database concurrently.

9. We have developed PL/pgSQL's DDL, DML and DCL scripts to create database table, establish relationship between entity and perform database migration for 30,000 rows of data.
10. We have developed a Go program to migrate 4 millions rows of data from legacy storage to new storage within PostgreSQL database without violates data consistency, validity and consistency.

8.2 Lessons Learned

1. **Data science knowledge.** Data science is being use as competitive weapon and it transform the way how companies operate with information. It is a totally new knowledge and experience for me as Software Engineering student to learn and explore.
2. **Concurrent programming concepts.** Concurrent concepts is difficult to be understand and never thought in subject syllabus. Learning the art of concurrent programming for building applications in this project provide satisfaction and motivation to fulfill my desire to build a real-time system.
3. **Consistent update with FYP Supervisor.** FYP supervisor ensure the project is on track and doing right. It is essential to make available time for consultation and rapidly update the progress for supervisor via email to enhance the work quality. Moreover, FYP supervisor review my work ensure the time and resource is not waste on doing the wrong task.
4. **Ubuntu Operating System.** The project allow me to learn Linux Bash commands through practice. The Ubuntu operating system is found not

difficult to be learned and it is more safety, reliable and consistent to conduct development activities due to its lightweight.

5. **PostgreSQL database.** The project allow me to learn the basics of PostgreSQL database configuration and developed PL/pgSQL scripts through development activities. It is enjoyable and joyful to learn the world's most advanced open source database and establish deeper understanding on database's feature. Other than that, the feeling of accomplishment emerged in my mind as I possess the flexibility to manipulate database settings and communicate with data source through query.
6. **Database normalization.** The project allow me to learn and implement the normalization rules to perform excellent data management with good database designs. My supervisor patiently guides the important procedure to perform database normalization and data migration during FYP meeting and constantly provide example to perform data cleaning.

8.3 Recommendations for Future Work

8.3.1 Phase 1

1. **GORM for CRUD on data processing.** GORM is an Object-relational mapping (ORM) library for Golang that converting data from incompatible files types into struct or interface. For instance, this project does not use GORM to import data and possess poor readability, error handling and maintainability in program. It is recommend to import data with GORM package because it supports auto migration,

associations with database and every features are tested.

2. **Benchmark on language performance comparison.** Although this project possess well-defined of benchmarking on database table spacing, hardware configuration and amount of query execution on data retrieval to conduct language performance comparison. These benchmarks are insufficient to determine the accurateness of programming language performance. This is because the CPU usage might be running on other processes or program while conducting the performance test. It is recommend to unified number of processes running in background and programming style for performance comparison between different concurrent programming languages.
3. **Data quality.** Although this project use data validation to identify raw dataset quality. The method is insufficient to ensure data obtained is valid, complete and accurate to be processed. It is recommend to use several scripting language such as Python and Perl to identify internal data consistency and validity.

8.3.2 Phase 2

1. **Company database normalized design.** Although the company datasets is normalized correctly, there are several transitive functional dependencies found in the table and required to be divide with 3NF (Third Normal Form) rules. The database still possess insert, delete and update anomalies on **company** tables and require extra efforts on reduce the complexity of tables.

2. **Data types for date attributes.** Although the data transformation, data parsing and data migration of company datasets are conducted successfully. The date values are declared as *VARCHAR* in PostgreSQL database and declared as *String* in Go and Rust program to reduce errors on date format conversion. The declaration increase difficulty on sorting and does not comply to unambiguous input format (ISO 8601). It is recommend to use *date* data types to store date values for better data analysis and processing results.
3. **Code structure of data cleaning parser.** Although the data cleaning parser is able to eliminate NULL values in every single rows and provide standardization support on each field, the program use more than 40 if-else statement within a loops and it reduces the performance on program execution. It is recommend to use better control flow statement to reduce the effort on data checking and resources utilization for the program.

Bibliography

[1] Ibrahim Abaker Targio Hashem et al. *The rise of big data on cloud computing: Review and open research issues*, 2014. URL `https://www.acm.org/publications/authors/reference-formatting`. Retrieved on 28/07/2017.

[2] David Geer. *Chip Makers Turn to Multicore Processors*, 2015. URL `http://ieeexplore.ieee.org/document/1430623/?part=1`. Retrieved on 28/07/2017.

[3] Bob Pike. *Google Tech Talk*, 2005. URL `http://9p.io/sources/contrib/ericvh/go-plan9/doc/go_talk-20091030.pdf`. Retrieved on 28/07/2017.

[4] Schneider F. B. Andrew G. R. Concept and notation of concurrent programming. *Computing Surveys*, pages 1–2, 1983. doi: http://babel.ls.fi.upm.es/teaching/concurrencia/material/concepts_and_notations.pdf. Retrieve on 04/08/2017.

[5] M. Ben-Ari. *Principle of Concurrent Programming*. Pearson 2nd Edition, 2005. ISBN 9780321312839.

[6] Kurt Guntheroth. *Why did Google develop Go?*, 2017. URL `https://www.quora.com/Why-did-Google-develop-Go/`. Retrieved on 29/07/2017.

[7] golang.org. The go programming language, 1999. URL `https://golang.org/`. Retrieved on 29/07/2017.

[8] GCC Organization. Ada, go and objective-c++ are not default languages, 2011. URL `https://gcc.gnu.org/install/configure.html`. Retrieved on 29/07/2017.

[9] Uwe R.Zimmer Benjamin J.L. Wang. College of engineering and computer sciences the australian national university. *Pure Concurrent Programming*, 2017. URL `http://ieeexplore.ieee.org/abstract/document/7965126/?part=1`. Retrieved on 29/07/2017.

[10] Britannica. Control structures, 2017. URL https://www.britannica.com/technology/computer-programming-language/Control-structures#ref849883. Retrieved on 05/08/2017.

[11] Joe Armstrong. Sequential vs concurrent programming languages. *Programming Erlang 2nd Edition*, 2013. doi: https://www.safaribooksonline.com/library/view/programming-erlang-2nd/9781941222454/f_0018.html.

[12] Brian Harvey and Matthew Wright. Sequential programming. *Simply Scheme: Introducing Computer Science*, 1999. doi: https://www.safaribooksonline.com/library/view/programming-erlang-2nd/9781941222454/f_0018.html.

[13] Herb Sutter. Will concurrency be the next revolution in software development?, 2005. URL http://www.drdobbs.com/the-concurrency-revolution/184401916. Retrieved on 05/08/2017.

[14] Jan Stenberg. Concurrent and distributed programming in the future, 2017. URL https://www.infoq.com/news/2017/03/distributed-programming-qcon. Retrieved on 06/08/2017.

[15] Gul Agha. Concurrent object-oriented programming. *Magazine Communications of the ACM*, pages 125–141, 1990. doi: 10.1145/83880.84528. URL http://dl.acm.org/citation.cfm?id=84528. Retrieved on 06/08/2017.

[16] Theodore Norvell. What is concurrent programming? pages 1–2, 2009. URL http://www.engr.mun.ca/~theo/Courses/cp/pub/cp0.pdf. Retrieved on 06/08/2017.

[17] Herb Sutter and James Larus. Software and concurrency revolution. *Queue – Multiprocessors*, pages 59–60, 2005. doi: 10.1145/1095408.1095421. URL http://dl.acm.org/citation.cfm?id=1095421. Retrieved on 06/08/2017.

[18] Tribaud. Top programming language to learn in 2017, 2017. URL https://www.codingame.com/blog/top-programming-languages-to-learn-in-2017. Retrieved on 07/08/2017.

[19] Horning J.J. Distributed processes: A concurrent programming concept. *Communication of the ACM*, 1978. doi: 10.1145/359642.359651. URL http://dl.acm.org/citation.cfm?id=359651. Retrieved on 07/08/2017.

[20] PostgreSQL Global Development Group. What is postgresql? *PostgreSQL 9.5.9 Documentation: Preface.*, 2017. URL `https://www.postgresql.org/docs/9.5/static/intro-whatis.html`. Retrieved on 08/09/2017.

[21] What is postgresql?, 2017. URL `http://www.postgresqltutorial.com/what-is-postgresql/`. Retrieved on 18/08/2017.

[22] Dibyendu Majumdar. A quick survey of multiversion concurrency algorithms. *MVCC Survey*, 2006. URL `forge.ow2.org/docman/view.php/237/132/mvcc-survey.pdf`. Retrieved on 19/08/2017.

[23] Sirish et al. Telegraphcq: continuous dataflow processing. *SIGMOD '03 Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, page 668, 2003. doi: 10.1145/872757.872857. URL `http://dl.acm.org/citation.cfm?id=872857`. Retrieved on 19/08/2017.

[24] George et al. Predictable performance and high query concurrency for data analytics. *The VLDB Journal*, pages 227–248, 2011. doi: 10.1007/s00778-011-0221-2. URL `http://delivery.acm.org.proxyvlib.mmu.edu.my/10.1145/1970000/1969355/778_2011_Article_221.pdf?ip=203.106.62.29&id=1969355&acc=ACTIVE%20SERVICE&key=69AF3716A20387ED%2EE854CB4DB8D6D408%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=801067487&CFTOKEN=72015032&__acm__=1503554298_5a4d19e623542c1086bd72577837f01a#URLTOKEN#`. Retrieved on 19/08/2017.

[25] PostgreSQL Global Development Group. Concurrency control. *Introduction*, 2017. URL `https://www.postgresql.org/docs/9.5/static/mvcc-intro.html`. Retrieved on 10/09/2017.

[26] PostgreSQL Global Development Group. Postgresql concurrency with mvcc. *How MVCC Works*, 2017. URL `https://devcenter.heroku.com/articles/postgresql-concurrency`. Retrieved on 10/09/2017.

[27] PostgreSQL Global Development Group. Linux downloads. *PostgreSQL Support Documentation*, 2017. URL `https://www.postgresql.org/download/linux/ubuntu/`. Retrieved on 10/09/2017.

[28] Rob Pike. Expressiveness of go, 2010. URL `http://www.intercapedine.net/documenti/ExpressivenessOfGo.pdf`. Retrieved on 07/08/2017.

[29] Forsby Filip and Persson Martin. Evaluation of golang for high performance scalable radio access systems, 2015. URL `http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A873124&dswid=-8907#sthash.gj7rKTc5.dpbs`. Retrieved on 07/08/2017.

[30] Slavomir Polak and Tomas Pitner. Text processing performance in go language. pages 149–152, 2014. URL `http://www.cssi-morava.cz/new/doc/IT2014/sbornik.pdf#page=149`. Retrieved on 08/08/2017.

[31] Pravenda Singh. Implementing an intelligent version of the classical sliding-puzzle game for unix terminals using golang's concurrency primitives. 2015. URL `https://arxiv.org/pdf/1503.08345.pdf`. Retrieved on 08/08/2017.

[32] Hoare. The rust programming language, 2013. URL `http://www.rust-lang.org/`. Retrieved on 08/08/2017.

[33] Eric Holk et al. Gpu programming in rust: Implementing high-level abstractions in a systems-level language. *Indiana University*, 2013. doi: 10.1109/IPDPSW.2013.173. URL `http://ieeexplore.ieee.org/abstract/document/6650903`. Retrieved on 08/08/2017.

[34] Eric Reed. Patina: A formalization of the rust programming language. university of washington. 2015. URL `https://www.cs.washington.edu/tr/2015/03/UW-CSE-15-03-02.pdf`. Retrieved on 08/08/2017.

[35] Sebastian Nanz et al. Design of an empirical study for comparing the usability of concurrent programming languages. *Information of Software Technology*, 55(7):1304–1315, 2013. URL `http://www.sciencedirect.com/science/article/pii/S0950584912001802`. Retrieved on 09/08/2017.

[36] Ehud Shapiro. Embeddings among concurrent programming languages (preliminary version). *Lecture Notes in Computer Science*, 630, 2006. URL `https://link.springer.com/chapter/10.1007%2FBFb0084811?LI=true`. Retrieved on 09/08/2017.

[37] Ehud Shapiro. Separating concurrent languages with categories of language embeddings. 2006. URL `https://pdfs.semanticscholar.org/7d2a/9a3954922741472f5ff06d2c1dafb258420e.pdf`. Retrieved on 09/08/2017.

[38] Ehud Shapiro. The family of concurrency programming languages. *ACM Computing Surveys (CSUR)*, 21(3):413–510, 1989. URL `http://dl.acm.org/citation.cfm?id=72555`. Retrieved on 10/08/2017.

[39] Maurice Herlihy. A methodology for implementing highly concurrent data objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(5), 1993. doi: 10.1145/161468.161469. URL `http://dl.acm.org/citation.cfm?id=161469`. Retrieved on 13/08/2017.

[40] Nenad Medvidovic and Richard N. Taylor. A framework for classifying and comparing architecture description languages. *ACM SIGSOFT Software Engineering Notes Homepage*, 22(6):60–76, 1997. doi: 10.1145/267896.267903. URL `http://dl.acm.org/citation.cfm?id=267903`. Retrieved on 13/08/2017.

[41] Stotts P.D. A comparative survey of concurrent programming languages. *ACM SIGPLAN*, 17:50–61, 1982. doi: 10.1109/2.73. URL `http://research.cs.queensu.ca/home/cordy/cisc860/Biblio/drb/CE/stotts82.pdf`. Retrieved on 15/08/2017.

[42] Vincent W.F. A comparison of implicit and explicit parallel programming. *Journal of Parallel and Distributed Computing*, 34(1):50–65, 1996. URL `http://www.sciencedirect.com/science/article/pii/S0743731596900453`. Retrieved on 15/08/2017.

[43] H.W. Loidl. Comparing parallel functional languages: Programming and performance. *Higher-Order and Symbolic Computation*, 16(3):203–251, 2003. doi: 10.1023/A:1025641323400. URL `http://dl.acm.org/citation.cfm?id=940872`. Retrieved on 15/08/2017.

[44] Simon Marlow. Distributed programming. *Parallel and Concurrent Programming in Haskell*, 2013. URL `https://www.safaribooksonline.com/library/view/parallel-and-concurrent/9781449335939/ch14.html`. Retrieved on 08/08/2017.

[45] Stackoverflow. Ii. most loved, dreaded, and wanted., developer survey results, 2016. URL `https://insights.stackoverflow.com/survey/2016`. Retrieved on 08/08/2017.

[46] Ty. Rust vs go adventures in error handling, 2017. URL `https://insights.stackoverflow.com/survey/2016`. Retrieved on 08/08/2017.

[47] Tiobe Software BV. The go programming language. *TIOBE Index*, 2017. URL `https://www.tiobe.com/tiobe-index/go/`. Retrieved on 11/09/2017.

[48] Stackoverflow. Most love programming language. *Developer Survey Results 2017*, 2017. URL `https://insights.stackoverflow.com/survey/2017`. Retrieved on 11/09/2017.

[49] golang.org. The go programming language, 2017. URL `https://golang.org/doc/`. Retrieved on 08/08/2017.

[50] rustlang.org. The rust programming language, 2017. URL `https://www.rust-lang.org/en-US/`. Retrieved on 08/08/2017.

[51] Caleb Doxsey. *Concurrency*. An Introduction to Programming in Go. 2017. URL `https://www.golang-book.com/books/intro/10`. Retrieved on 08/08/2017.

[52] Chua Yong Wen. Appreciating rust's memory safety guarantees, 2017. URL `https://blog.gds-gov.tech/appreciating-rust-memory-safety-438301fee097`. Retrieved on 09/08/2017.

[53] Hackernews. Rust vs go, 2017. URL `https://news.ycombinator.com/item?id=13430108`. Retrieved on 09/08/2017.

[54] Arild Nilsen. Communication sequential process (csp). *An alternative to the actor model*, 2017. URL `https://arild.github.io/csp-presentation/`. Retrieved on 11/09/2017.

[55] Will Yager. The problem. *Why Go is no good*, 2017. URL `http://yager.io/programming/go.html`. Retrieved on 11/09/2017.

[56] Techopedia. Actor model. *Programming Tools*, 2017. URL `https://www.techopedia.com/definition/25150/actor-model`. Retrieved on 11/09/2017.

[57] Ticki. Why should i use rust? *The RUST programming language*, 2016. URL `https://www.reddit.com/r/rust/comments/4l44z3/why_should_i_use_rust/`. Retrieved on 11/09/2017.

[58] Rust lang organization. How do i map object-oriented concepts to rust? *Design Patterns*, 2017. URL `https://www.rust-lang.org/en-US/faq.html#how-do-i-map-object-oriented-concepts-to-rust`. Retrieved on 11/09/2017.

[59] Simon Hoare. What is the difference between unix, linux and ubuntu? *Ask Ubuntu Forum*, 2012. URL `https://askubuntu.com/questions/183723/whats-the-difference-between-unix-linux-and-ubuntu`. Retrieved on 08/09/2017.

[60] Invert. Why is ubuntu is more secure than windows or mac os x? *Ask Ubuntu Forum*, 2010. URL `https://askubuntu.com/questions/1069/why-is-ubuntu-more-secure-than-windows-or-mac-os-x`. Retrieved on 08/09/2017.

[61] Katherine Noyes. Why linux is more secure than windows? *Linux Line*, 2017. URL `https://www.pcworld.com/article/202452/why_linux_is_more_secure_than_windows.html`. Retrieved on 08/09/2017.

[62] James McInnes. What are key differences between unix and ms-dos? *Programming language comparisons*, 2015. URL `https://www.quora.com/What-are-the-key-differences-between-Unix-and-MS-DOS`. Retrieved on 08/09/2017.

[63] Spehro Pefhany. Why is printf() bad for debugging embedded systems? *Electrical Engineering Stack Exchange*, 2014. URL `https://electronics.stackexchange.com/questions/105283/why-is-printf-bad-for-debugging-embedded-systems`. Retrieved on 11/09/2017.

[64] The LTTng project. What is tracing? *Trace Compass Documentation*, 2017. URL `http://lttng.org/docs/v2.9/#doc-what-is-tracing`. Retrieved on 11/09/2017.

[65] Tutorialpoint. What is gnu debugger? *How GDB debugs?*, 2017. URL `https://www.tutorialspoint.com/gnu_debugger/what_is_gdb.htm`. Retrieved on 11/09/2017.

[66] Tutorialspoint. Data profile validation. *ETL Testing - Data Completeness*, 2017. URL `https://www.tutorialspoint.com/etl_testing/etl_testing_data_completeness.htm`. Retrieved on 03/02/2018.

[67] Techopedia. Data redundancy. *Unified Communication*, 2018. URL `https://www.techopedia.com/definition/18707/data-redundancy`. Retrieved on 03/02/2018.

[68] Margaret Rouse. Encoding and decoding. *Programming*, 2005. URL `http://searchnetworking.techtarget.com/definition/encoding-and-decoding`. Retrieved on 03/02/2018.

[69] Justin Eillingwood. The basics of using the sed stream editor to manipulate text in linux. *Linux Basics and Commands*, 2013. URL `https://www.digitalocean.com/community/tutorials/the-basics-of-using-the-sed-stream-editor-to-manipulate-text-in-linux`. Retrieved on 03/02/2018.

[70] MuleSoft. Data transformation. *SQA Resources*, 2018. URL `https://www.mulesoft.com/resources/esb/data-transformation`. Retrieved on 03/02/2018.

[71] Satis. Introduction. *What is an ORM and where can I learn about it*, 2009. URL `https://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-more-about-it`. Retrieved on 03/02/2018.

[72] Experian. What is data cleansing. *Data Cleansing*, 2014. URL `https://www.edq.com/uk/glossary/data-cleansing/`. Retrieved on 03/02/2018.

[73] Vikrant Oberoi. What is data redundancy in a dbms? what is a simple explanation? *Database management software of Quora*, 2017. URL `https://www.quora.com/What-is-data-redundancy-in-a-DBMS-What-is-a-simple-explanation`. Retrieved on 04/02/2018.

[74] Margaret Rouse. Definition. *data migration*, 2017. URL `http://searchstorage.techtarget.com/definition/data-migration`. Retrieved on 04/02/2018.

[75] Golang organization. Performance. *Go 1.8 Release Notes*, 2017. URL `https://golang.org/doc/go1.8#performance`. Retrieved on 18/09/2017.

Appendices

Appendix A

Infrastructure Setup and Installation

A.1 Linux command for Go compiler installation

```
1
2  ============================================================
3  (1) DOWNLOAD GOLANG go1.8.3.linux-amd64.tar.gz
4  AT URL https://golang.org/dl/ USING wget IN TERMINAL
5  ============================================================
6
7  yinghua@yinghua-NL8C:~/Downloads/temp$ wget -c https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar
       .gz
8  ...
9  go1.8.3.linux-amd64 100%[====================>]  85.86M  5.93MB/s    in 14s
10 yinghua@yinghua-NL8C:~/Downloads/temp$
11
12 ============================================================
13 (2) EXTRACT DOWNLOADED SOURCE
14 ============================================================
15 yinghua@yinghua-NL8C:~/Downloads/temp$ tar -xzvf go1.8.3.linux-amd64.tar.gz
16 ....
17 yinghua@yinghua-NL8C:~/Downloads/temp$
18
19 ============================================================
20 (3) MOVE AND RENAME GOLANG DIRECTORY
21 ============================================================
22 yinghua@yinghua-NL8C:~/Downloads/temp$ mkdir -p ~/Desktop/apps/golang1.8.3
23 yinghua@yinghua-NL8C:~/Downloads/temp$ mv go ~/apps/golang1.8.3
24 yinghua@yinghua-NL8C:~/Downloads/temp$
25
26 ============================================================
27 (4) CHECK GOLANG DIRECTORY
28 ============================================================
29 yinghua@yinghua-NL8C:~/Downloads/temp$ cd ~/Desktop/apps/
30 yinghua@yinghua-NL8C:~/Desktop/apps$ ls -l
31 total 24
32 drwxr-xr-x  8 yinghua yinghua 4096 Sep 11 03:03 eclipse-oxygen
33 drwxrwxr-x  4 yinghua yinghua 4096 Sep  7 23:19 eclipse-workspace
34 drwxr-xr-x 11 yinghua yinghua 4096 May 25 02:16 golang1.8.3
35
36 ============================================================
37 (5) GO INTO GOLANG INSTALLED DIRECTORY
38 ============================================================
39 yinghua@yinghua-NL8C:~/Desktop/apps$ cd golang1.8.3/
40 yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$ ls -l
41 total 160
42 drwxr-xr-x  2 yinghua yinghua  4096 May 25 02:15 api
43 -rw-r--r--  1 yinghua yinghua 33243 May 25 02:15 AUTHORS
44 drwxr-xr-x  2 yinghua yinghua  4096 May 25 02:16 bin
45 drwxr-xr-x  4 yinghua yinghua  4096 May 25 02:16 blog
```

```
-rw-r--r--  1 yinghua yinghua  1366 May 25 02:15 CONTRIBUTING.md
....
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$

=========================================================
(5.1) CHECK GOLANG EXECUTABLES
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$ ls -al bin
total 28120
drwxr-xr-x  2 yinghua yinghua     4096 May 25 02:16 .
drwxr-xr-x 11 yinghua yinghua     4096 May 25 02:16 ..
-rwxr-xr-x  1 yinghua yinghua 10073055 May 25 02:16 go
-rwxr-xr-x  1 yinghua yinghua 15226597 May 25 02:16 godoc
-rwxr-xr-x  1 yinghua yinghua  3481554 May 25 02:16 gofmt
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$


=========================================================
(5.2) CHECK GOLANG LIBRARIES
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$ ls -al lib
total 12
drwxr-xr-x  3 yinghua yinghua 4096 May 25 02:15 .
drwxr-xr-x 11 yinghua yinghua 4096 May 25 02:16 ..
drwxr-xr-x  2 yinghua yinghua 4096 May 25 02:15 time
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$

=========================================================
(5.3) CHECK GOLANG PACKAGES
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$ ls -al pkg
total 28
drwxr-xr-x  7 yinghua yinghua 4096 May 25 02:16 .
drwxr-xr-x 11 yinghua yinghua 4096 May 25 02:16 ..
drwxr-xr-x  2 yinghua yinghua 4096 May 25 02:15 include
drwxr-xr-x 30 yinghua yinghua 4096 May 25 02:16 linux_amd64
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$
....

=========================================================
(6) SET PATH TO GOLANG BINARY EXECUTABLES AND EXPORT PATH
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$ cd bin
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ pwd
/home/yinghua/Desktop/apps/golang1.8.3/bin
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ export PATH=/home/yinghua/Desktop/apps/golang1.8.3/bin:
     $PATH
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$

=========================================================
(6.1) CHECK ADDED GOLANG PATH
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ echo $PATH
/home/yinghua/Desktop/apps/golang1.8.3/bin: <=== PATH ADDED
/home/yinghua/.cargo/bin:
/home/yinghua/bin:
/home/yinghua/.local/bin:
/usr/local/sbin:
....
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$


=========================================================
(6.2) SET GOROOT AND GOPATH
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ mkdir ~/Desktop/apps/gopath
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ ls -al ~/Desktop/apps
total 24
drwxr-xr-x  8 yinghua yinghua 4096 Sep 11 03:03 eclipse-oxygen
drwxrwxr-x  4 yinghua yinghua 4096 Sep  7 23:19 eclipse-workspace
drwxr-xr-x 11 yinghua yinghua 4096 May 25 02:16 golang1.8.3
drwxrwxr-x  5 yinghua yinghua 4096 Sep  7 23:05 gopath

yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ export GOROOT=/home/yinghua/Desktop/apps/golang1.8.3
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ export GOROOT=/home/yinghua/Desktop/apps/gopath
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ export PATH=$GOPATH/bin:$PATH

=========================================================
(6.3) CHECK GOROOT AND GOPATH
=========================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ echo $GOROOT
/home/yinghua/Desktop/apps/golang1.8.3
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ echo $GOPATH
/home/yinghua/Desktop/apps/gopath
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$

=========================================================
(6.4) APPLY SYSTEM UPDATES
=========================================================
```

```
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ sudo updatedb
[sudo] password for yinghua:
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ sudo ldconfig
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ sudo depmod
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$

============================================================
(7) APPEND PATH TO USER PROFILE .bashrc FILE
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ nano ~/.bashrc

# ======= ADDED BY CYH INTO ~/.bashrc ============
# added by CYH for Golang1.8.3 Compiler
export GOROOT=/home/yinghua/Desktop/apps/golang1.8.3
export GOPATH=/home/yinghua/Desktop/apps/gopath
export PATH=$GOROOT/bin:$GOPATH/bin:$PATH

============================================================
(8) CHECK GO EXECUTABLE AND GO VERSION
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ which go
/home/yinghua/Desktop/apps/golang1.8.3/bin/go
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ go version
go version go1.8.3 linux/amd64
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$

============================================================
(9) TEST GO EXECUTABLE
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ go help
Go is a tool for managing Go source code.
.....

============================================================
(10) GO TO GOPATH DIRECTORY TO INSTALL TOOLS
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3/bin$ cd ..
yinghua@yinghua-NL8C:~/Desktop/apps/golang1.8.3$  cd ..
yinghua@yinghua-NL8C:~/Desktop/apps$  cd gopath/
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$  ls -l
total 0
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$

============================================================
(11) DOWNLOAD GO PACKAGE TOOLS (EXECUTABLES)
============================================================
Use git to download go libraries (gocode, golint, guru, goimports, gorename, godef)

yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get github.com/nsf/gocode
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get github.com/golang/lint/golint
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get golang.org/x/tools/cmd/guru
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get golang.org/x/tools/cmd/goimports
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get golang.org/x/tools/cmd/gorename

============================================================
(11.1) DOWNLOAD GODEF GOMETALINTER
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get github.com/rogpeppe/godef
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ go get -u gopkg.in/alecthomas/gometalinter.v1

============================================================
(11.2) EXECUTE GOMETALINTER
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ cd bin
yinghua@yinghua-NL8C:~/Desktop/apps/gopath/bin$ gometalinter.v1 --install
.....
gocyclo
goimports
interfacer
safesql
unparam
wruslan@dell-ub1604-64b:~/apps/gopath/bin$

============================================================
(11.3) CHECK INSTALLED PACKAGES (LIBRARIES)
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/gopath$ ls -al bin
total 154644
drwxrwxr-x 2 yinghua yinghua     4096 Sep  7 23:09 .
drwxrwxr-x 5 yinghua yinghua     4096 Sep  7 23:05 ..
-rwxrwxr-x 1 yinghua yinghua  7521174 Sep  7 23:09 gas
-rwxrwxr-x 1 yinghua yinghua 10521898 Sep  7 23:05 gocode <=== FOR ECLIPSE IDE
-rwxrwxr-x 1 yinghua yinghua  3015835 Sep  7 23:09 goconst
-rwxrwxr-x 1 yinghua yinghua  2453860 Sep  7 23:09 gocyclo
-rwxrwxr-x 1 yinghua yinghua  5503061 Sep  7 23:06 godef <=== FOR ECLIPSE IDE
-rwxrwxr-x 1 yinghua yinghua  4898036 Sep  7 23:09 goimports
-rwxrwxr-x 1 yinghua yinghua  8309030 Sep  7 23:05 guru   <=== FOR ECLIPSE IDE
-rwxrwxr-x 1 yinghua yinghua  2494881 Sep  7 23:09 ineffassign
.....
```

```
223
224 ============================================================
225 END
226 ============================================================
```

Listing A.1: Linux command for Golang compiler installation

A.2 Linux command for Rust compiler installation

```
1
2  ============================================================
3  (1) INSTALL COMMANDLINE Rust toolchain
4  ============================================================
5  yinghua@yinghua-NL8C:~/Desktop/apps/rust$ curl https://sh.rustup.rs -sSf | sh
6
7  Welcome to Rust!
8
9  This will download and install the official compiler for the Rust programming
10 language, and its package manager, Cargo.
11
12 It will add the cargo, rustc, rustup and other commands to Cargo's bin
13 directory, located at:
14
15 /home/yinghua/.cargo/bin
16
17 This path will then be added to your PATH environment variable by modifying the
18 profile file located at:
19
20 /home/yinghua/.profile
21
22 You can uninstall at any time with rustup self uninstall and these changes will
23 be reverted.
24
25 Current installation options:
26
27 default host triple: i686-unknown-linux-gnu
28 default toolchain: stable
29 modify PATH variable: yes
30
31 1) Proceed with installation (default)
32 2) Customize installation
33 3) Cancel installation
34
35 info: syncing channel updates for 'stable-i686-unknown-linux-gnu'
36 156.7 KiB / 156.7 KiB (100 %) 126.1 KiB/s ETA:   0 s
37 info: downloading component 'rustc'
38 38.9 MiB /  38.9 MiB (100 %) 505.6 KiB/s ETA:   0 s
39 ......
40
41 stable installed - rustc 1.17.0 (56124baa9 2017-04-24)
42
43
44 Rust is installed now. Great!
45
46 To get started you need Cargo's bin directory in your PATH environment
47 variable. Next time you log in this will be done automatically.
48
49 To configure your current shell run source $HOME/.cargo/env
50 yinghua@yinghua-NL8C:~/Desktop/apps/rust$
51
52 ============================================================
53 (2) EXPORT RUST EXECUTABLE TO PATH
54 ============================================================
55
56 yinghua@yinghua-NL8C:~$ cd ~/Desktop/apps/rust/
57 yinghua@yinghua-NL8C:~/Desktop/apps/rust$ rustc --version
58 rustc 1.20.0 (f3d6973f4 2017-08-27)
59 yinghua@yinghua-NL8C:~/Desktop/apps/rust$ sudo updatedb
60 [sudo] password for yinghua:
61 yinghua@yinghua-NL8C:~/Desktop/apps/rust$ locate bin/rustc
62 /home/yinghua/.cargo/bin/rustc
63 /home/yinghua/.rustup/toolchains/stable-x86_64-unknown-linux-gnu/bin/rustc
64 /usr/bin/rustc
65 yinghua@yinghua-NL8C:~/Desktop/apps/rust$ export PATH=$PATH:$HOME/.cargo/bin
66 yinghua@yinghua-NL8C:~/Desktop/apps/rust$ rustup component add rust-src
67 info: downloading component 'rust-src'
68 30.4 MiB /  30.4 MiB (100 %) 371.2 KiB/s ETA:   0 s
```

```
info: installing component 'rust-src'

============================================================
(3) INSTALL RACER
============================================================
yinghua@yinghua-NL8C:~$ cargo install racer
Updating registry `https://github.com/rust-lang/crates.io-index`
.....
Finished release [optimized + debuginfo] target(s) in 928.10 secs
Installing /home/yinghua/.cargo/bin/racer
yinghua@yinghua-NL8C:~$

============================================================
(4) INSTALL RUSTFMT
============================================================
yinghua@yinghua-NL8C:~$ cargo install rustfmt
Updating registry `https://github.com/rust-lang/crates.io-index`
....
Finished release [optimized] target(s) in 786.15 secs
Installing /home/yinghua/.cargo/bin/cargo-fmt
Installing /home/yinghua/.cargo/bin/rustfmt
yinghua@yinghua-NL8C:~$

============================================================
(5) INSTALL RAINICORN
============================================================
yinghua@yinghua-NL8C:~$ cargo install --git https://github.com/RustDT/Rainicorn --tag version_1.x
The program 'cargo' is currently not installed. You can install it by typing:
sudo apt install cargo
yinghua@yinghua-NL8C:~$ export PATH=$PATH:$HOME/.cargo/bin
yinghua@yinghua-NL8C:~$ which cargo
/home/yinghua/.cargo/bin/cargo

yinghua@yinghua-NL8C:~$ cargo install --git https://github.com/RustDT/Rainicorn --tag version_1.x
Updating git repository `https://github.com/RustDT/Rainicorn`
Installing rainicorn v1.3.0 (https://github.com/RustDT/Rainicorn?tag=version_1.x#365f819b)
Updating registry `https://github.com/rust-lang/crates.io-index`
.....
Finished release [optimized] target(s) in 527.77 secs
Installing /home/yinghua/.cargo/bin/parse_describe
yinghua@yinghua-NL8C:~$

============================================================
(6) CHECK RUST EXECUTABLES (11 NOS.)
============================================================
yinghua@yinghua-NL8C:~/Desktop/apps/rust$ which cargo
/home/yinghua/.cargo/bin/cargo
yinghua@yinghua-NL8C:~/Desktop/apps/rust$ rustc --version
rustc 1.20.0 (f3d6973f4 2017-08-27)
yinghua@yinghua-NL8C:~/Desktop/apps/rust$ which rustc
/home/yinghua/.cargo/bin/rustc
yinghua@yinghua-NL8C:~/Desktop/apps/rust$ ls -al /home/yinghua/.cargo/bin/
total 145404
drwxrwxr-x 2 yinghua yinghua     4096 Sep  7 22:39 .
drwxrwxr-x 5 yinghua yinghua     4096 Sep  7 22:36 ..
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 cargo
-rwxrwxr-x 1 yinghua yinghua  4126864 Sep  7 22:39 cargo-fmt
-rwxrwxr-x 1 yinghua yinghua  3828768 Sep  7 22:38 parse_describe
-rwxrwxr-x 1 yinghua yinghua 46240312 Sep  7 22:34 racer
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rls
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rustc
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rustdoc
-rwxrwxr-x 1 yinghua yinghua  8291104 Sep  7 22:39 rustfmt
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rust-gdb
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rust-lldb
-rwxr-xr-x 7 yinghua yinghua 12340104 Sep  7 22:19 rustup
yinghua@yinghua-NL8C:~/Desktop/apps/rust$

============================================================
END
============================================================
```

Listing A.2: Linux command for Rust compiler installation

A.3 Eclipse IDE installation

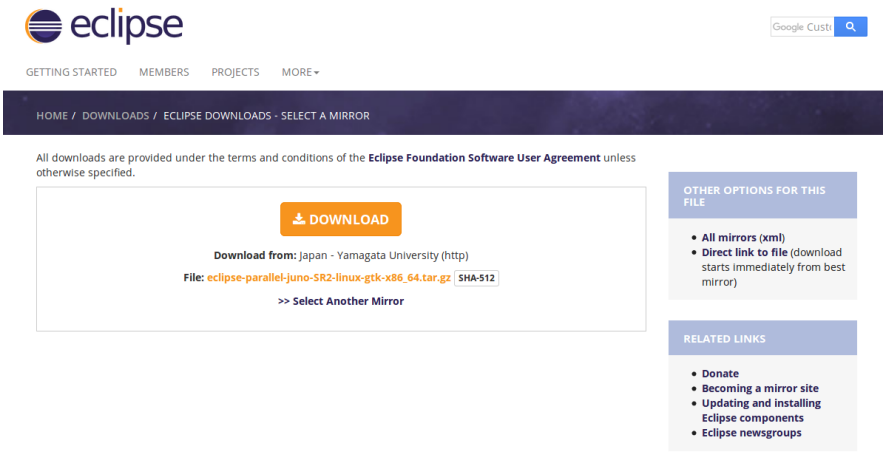


FIGURE A.1: Eclipse Oxygen Download Official Website

Ensure the Eclipse IDE version selected is compatible with 64-bit Ubuntu Operating System.

A.4 GoClipse plugin for Eclipse IDE installation

A.4.1 Eclipse Marketplace

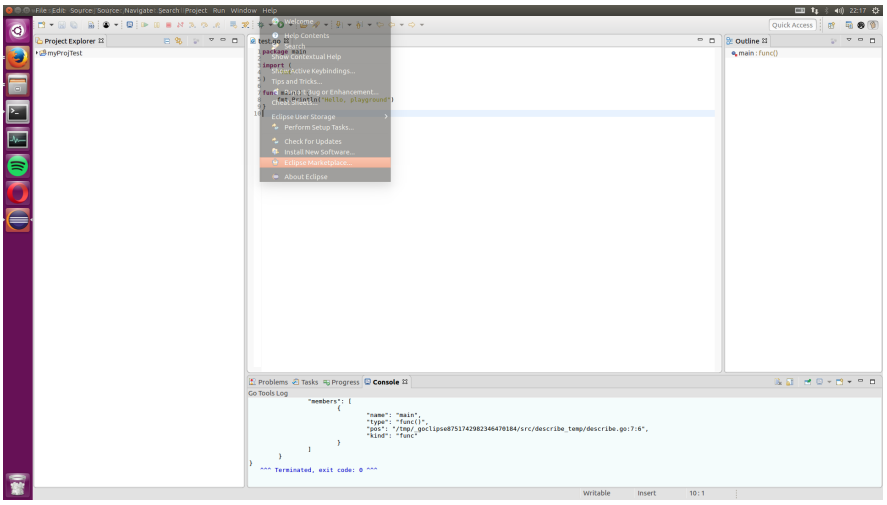


FIGURE A.2: Eclipse IDE Marketplace

Open Eclipse Marketplace from Help and select Eclipse Marketplace to search for GoClipse plugin.

A.4.2 Search Marketplace

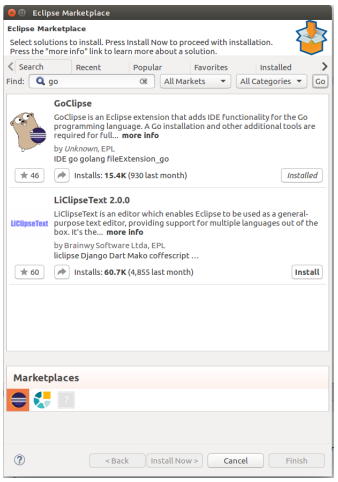


FIGURE A.3: Search Eclipse IDE Marketplace

Type "Go" in search bar and press Go button to search for available plugin. Press install now to proceed with installation.

A.4.3 Open Perspective

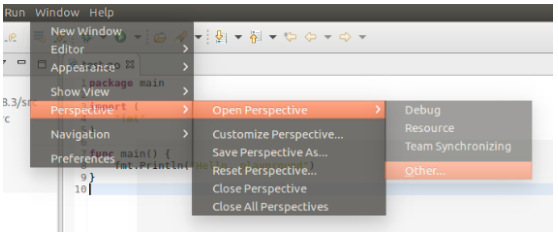


FIGURE A.4: Open Perspective

After the installation is done and success, open Eclipse Perspective by select Window, Perspective, Open Perspective and choose Other.

A.4.4 Choose Perspective

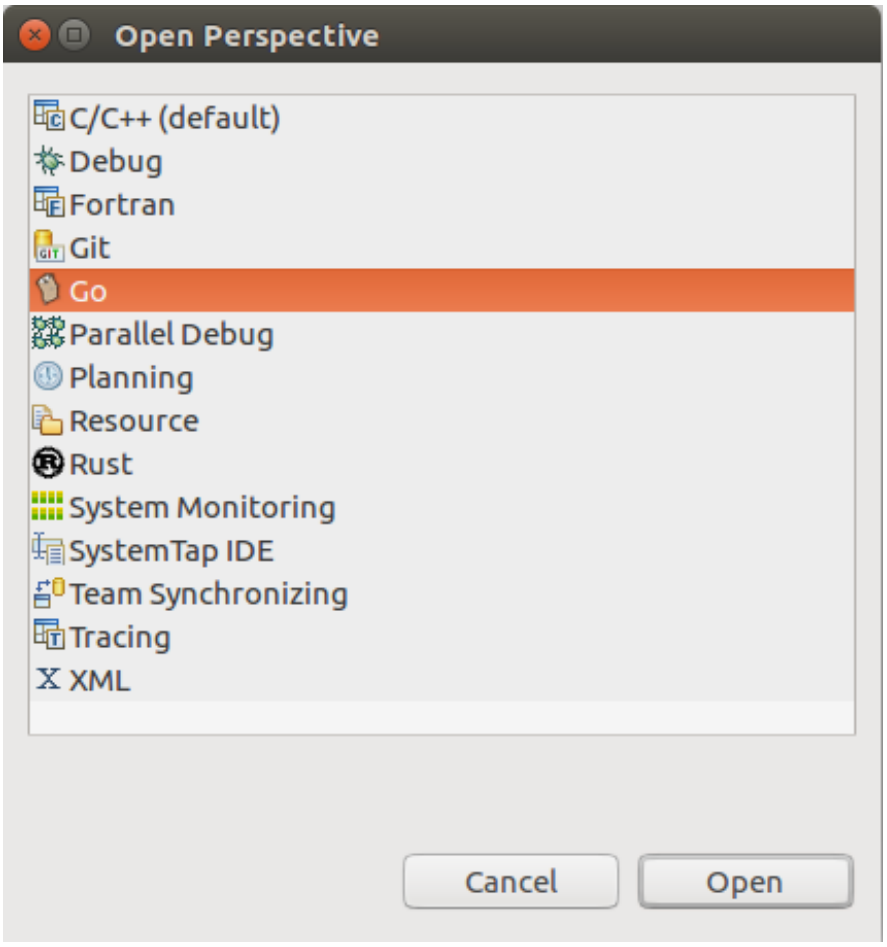


FIGURE A.5: Choose Go Perspective

Choose Go Perspective and press Enter.

A.4.5 Set Go compiler and GOPATH

goclipse-setting.png goclipse-setting.png

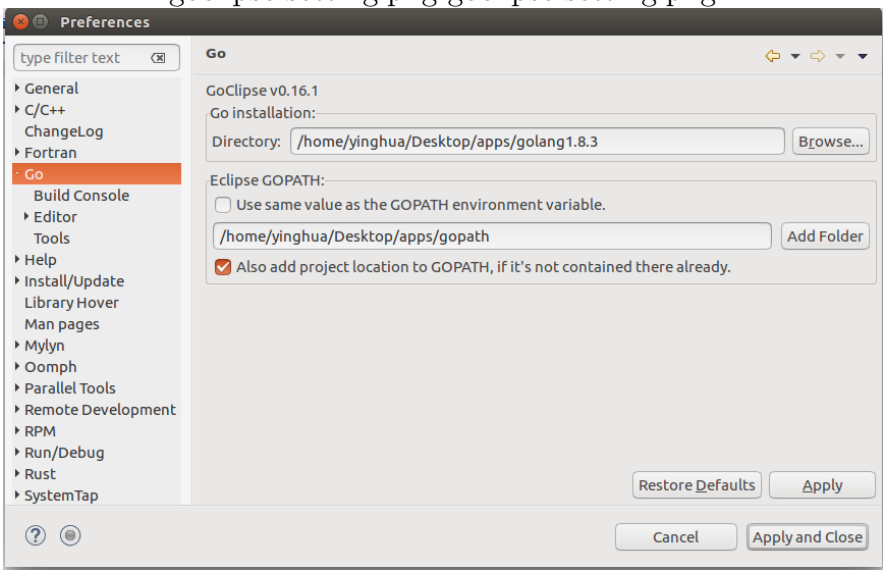


FIGURE A.6: Set Go compiler and GOPATH

Set Go compiler and GOPATH into Goclipse plugins.

A.4.6 Set GOCODE, GURU, GODEF and GOFMT path

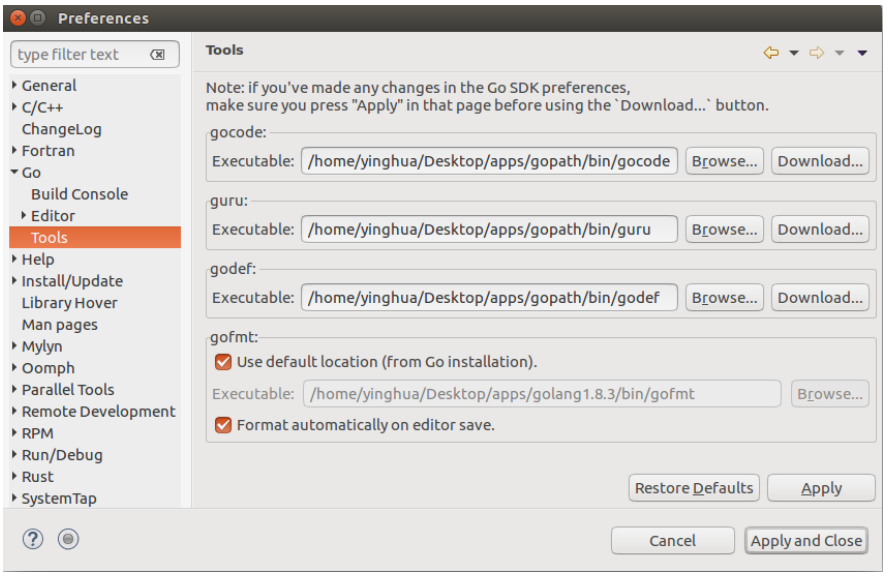


FIGURE A.7: Set GOCODE, GURU, GODEF and GOFMT path

Set GOCODE, GURU, GODEF and GOFMT executable path into Goclipse plugins and press "Apply and Close" to complete the setup process.

A.4.7 Test Go compilation in Eclipse IDE

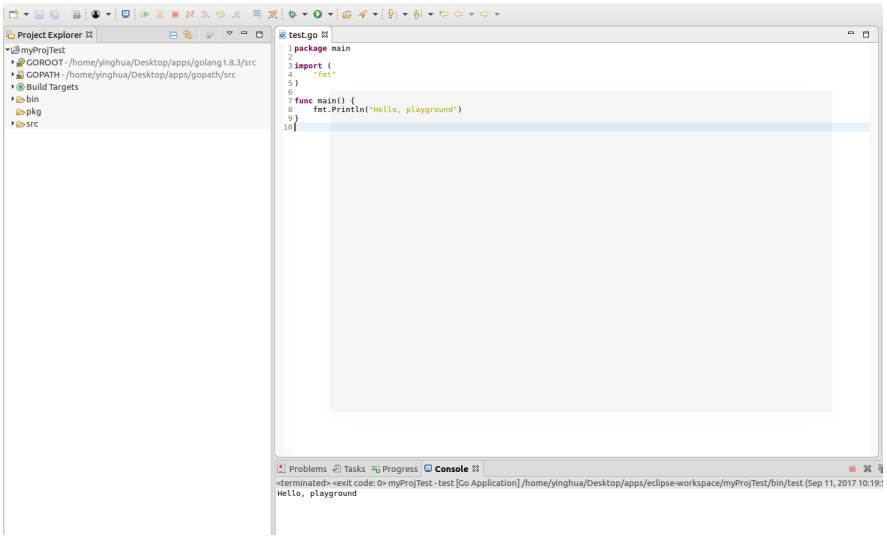


FIGURE A.8: Test Go compilation in Eclipse IDE

Test Go compilation with simple Hello Playground program, the setup process is successful if the Go program is compile and run correctly.

A.5 RustDT plugin for Eclipse IDE installation

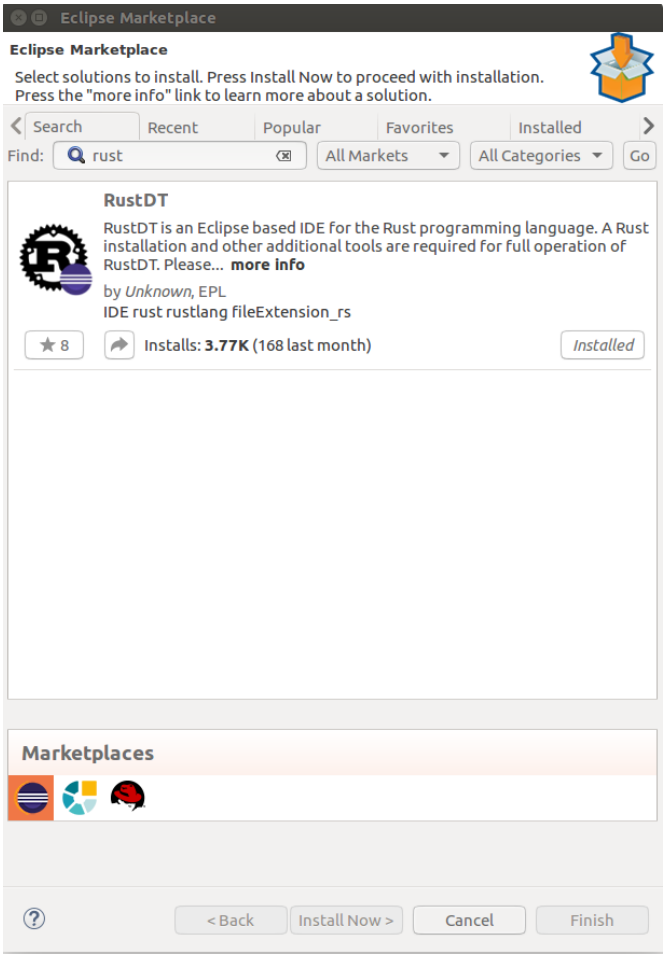


FIGURE A.9: Test Go compilation in Eclipse IDE

Open Eclipse Marketplace similar to step in Appendix A.4.1 to A.4.7. Search the marketplace by type "Rust" in search bar and press Go button to search for tools. Press install now to proceed with installation. The setup process is similar with Goclipse installation process, once the installation and setup is done. The program will compile and run successfully.

A.6 Linux command for PostgreSQL database installation

```
1
2  ==========================================
3  Step 1 - Install postgreSQL in command line
4  ==========================================
5
6  yinghua@yinghua-NL8C:~$ sudo apt-get update
7  yinghua@yinghua-NL8C:~$ sudo apt-get install postgresql postgresql-contrib
8  [sudo] password for yinghua:
9
10 ==========================================
11 Step 2 - Create database for FYP1
12 ==========================================
13 postgres=# create database fyp1;
14 CREATE DATABASE
15
16 postgres=# \l
17 List of databases
18 Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
19 ----------+----------+----------+-------------+-------------+-----------------------
20 fyp1      | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
21 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
22 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
23 |         |          |             |             | postgres=CTc/postgres
24 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
25 |          |          |             |             | postgres=CTc/postgres
26 (4 rows)
27
28 ==========================================
29 Step 3 - Initial login with postgres user into psql
30 ==========================================
31 yinghua@yinghua-NL8C:~$ sudo -i -u postgres psql
32 psql (9.5.7)
33 Type "help" for help.
34
35 ======================================================
36 Step 4 - Add myself as new user for PostgreSQL with Superuser access
37 ======================================================
38 yinghua@yinghua-NL8C:~/Documents/FYP/Postcode-data/uk-postcodes-master$ sudo -i -u postgres psql fyp1
39 [sudo] password for yinghua:
40 psql (9.5.7)
41 Type "help" for help.
42
43 postgres@yinghua-NL8C:~$ createuser -P -s -e yinghua
44 Enter password for new role:
45 Enter it again:
46 CREATE ROLE yinghua PASSWORD 'md5eec308d944ffa817c37ee6230b0c98eb' SUPERUSER CREATEDB CREATEROLE INHERIT
       LOGIN;
47
48 ======================================================
49 Step 5 - List all the user in PostgreSQL
50 ======================================================
51 postgres=# \du
52 List of roles
53 Role name |                         Attributes                         | Member of
54 -----------+------------------------------------------------------------+-----------
55 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
56 yinghua   | Superuser, Create role, Create DB
57
58
59 ==========================================
60 Step 6 - Connect FYP1 Database
61 ==========================================
62 postgres=# \c fyp1
63 You are now connected to database "fyp1" as user "postgres".
64
65 ================================================
66 Step 7 - Check whether there are tables in FYP1 database
67 ================================================
68 fyp1=# \dt
69 No relations found.
```

Listing A.3: Linux command for PostgreSQL database installation

Install PostgreSQL database with command line using APT package. After the installation is success, create new user for new database in PostgreSQL.

Appendix B

Data Validation

B.1 Introduction

The devil advocation test is conducted to ensure obtained raw CSV data are clean and useful. The test is conduced to ensure:-

1. The number of commas in each records should match number of columns in database.
2. Raw data from CSV should match the column's data type in database for data importation and preparation.
3. Review and check uniqueness of data in each columns and row.

B.2 Match number of commas with database columns

```
1
2  ===================
3  1. connect to database
4  ===================
5
6  yinghua@yinghua:~$ psql fyp1;
7  psql (9.5.8)
8  Type "help" for help.
9
10 ==================================
11 2. create table without one column
12 ==================================
13
14 fyp1=# create table subject_test ( ukprn int not null, providername varchar(100) not null, region varchar
       (100) not null, subject varchar(50) not null, sex varchar(30) not null, yearaftergraduation varchar
       (30) not null, grads varchar(10) null default null, unmatched varchar(20) null default null, matched
       varchar(20) null default null, activityNotCaptured varchar(20) default null, nosustdest varchar(20)
       null default null, sustemponly varchar(20) null default null, sustemp varchar(20) null default null,
       sustempfsorboth varchar(20) null default null, earningsinclude varchar(20) null default null,
       lowerannearn varchar(20) null default null, medianannearn varchar(20) null default null, upperannearn
       varchar(20) null default null, polargrpone varchar(20) null default null, polargrponeincluded varchar
       (20) null default null, prattband varchar(20) null default null);
15
16 ========================================================================
17 3. Terminal return error complains extra data after last expected column
18 ========================================================================
19
20 fyp1=# \copy subject_test from 'institution-subject-data.csv' with header csv;
21
22 ERROR:  extra data after last expected column
23
24 CONTEXT:  COPY subject_test, line 2: "10000291,Anglia Ruskin University,East,Agriculture & related subjects,
       Female,1,30,x,x,x,x,x,x,20,9..."
```

Listing B.1: Match number of commas with database columns

In this section, PostgreSQL query is executed on a terminal to check the number of commas match the number of columns possesses in the table. We will purposely remove one column during table creation and try to import all rows of data into PostgreSQL database.

The terminal will return an error and complains data could not insert into the table because a column is expected during importation process.

```
1
2  ========================================================================
3  4. Add new column into tables and import data successfully
4  ========================================================================
5
6  fyp1=# alter table subject_test add column prattincluded varchar(20) null default null;
7  fyp1=# \copy subject_test from 'institution-subject-data.csv' with header csv;
8  COPY 32706
```

Listing B.2: Identify correctness of data types

Ultimately, the CSV raw data will import successfully only if the count of commas match the counts of columns in table.

B.3 Identify correctness and suitability of data types

```
1
2  ======================
3  Step 1. connect to database
4  ======================
5
6  yinghua@yinghua:~$ psql fyp1;
7  psql (9.5.8)
8  Type "help" for help.
9
10 ==================================
11 Step 2. create companydata table
12 ==================================
13
14 fyp1=# create table companydata ( CompanyName varchar(160) null default null, CompanyNumber varchar(8) not
       null primary key, CareOf varchar(100) null, POBOX varchar(10) null, AddressLine1 varchar(300) null,
       AddressLine2 varchar(300) null, PostTown varchar(50) null, County varchar(50) null, Country varchar
       (50) null, PostCode varchar(20) null, CompanyCategory varchar(100) not null, CompanyStatus varchar(70)
        not null, CountryOfOrigin varchar(50) not null, DissolutionDate date null default null,
       IncorporationDate date null default null, AccountingRefDay int null, AccountingRefMonth int null
       default 0, Account_NextDueDate date null default null, Account_LastMadeUpdate date null default null,
       AccountCategory varchar(30) null, Return_NextDueDate date null default null, Return_LastMadeUpDate
       date null default null, NumMortCharges int null, NumMortOutstanding int null, NumMortPartSatisfied int
        null, NumMortSatisfied int null, SICCode1 varchar(170) null, SICCode2 varchar(170) null, SICCode3
       varchar(170) null, SICCode4 varchar(170) null, NumGenPartners int not null, NumLimPartners int not
       null, URI varchar(47) not null, pn1_CONDate date null default null, pn1_CompanyName varchar(160) null,
        pn2_CONDate date null default null, pn2_CompanyName varchar(160) null, pn3_CONDate date null default
       null, pn3_CompanyName varchar(160) null, pn4_CONDate date null default null, pn4_CompanyName varchar
       (160) null, pn5_CONDate date null default null, pn5_CompanyName varchar(160) null, pn6_CONDate date
       null default null, pn6_CompanyName varchar(160) null, pn7_CONDate date null default null,
       pn7_CompanyName varchar(160) null, pn8_CONDate date null default null, pn8_CompanyName varchar(160)
       null, pn9_CONDate date null default null, pn9_CompanyName varchar(160) null, pn10_CONDate date null
       default null, pn10_CompanyName varchar(160) null, ConfStmtNextDueDate date null default null,
       ConfStmtLastMadeUpDate date null default null);
15 CREATE TABLE
16
17 ========================================
18 Step 3 - Import data into companydata table
19 ========================================
20 fyp1=# \copy companydata from 'Basic-Company-Data-Full.csv' with header csv;
21
22 ============================================================================
23 Step 4 - Terminal return error because double quotes are not allow to insert into date datatypes.
24 ============================================================================
25 ERROR:  invalid input syntax for type date: ""
26 CONTEXT:  COPY companydata, line 2, column dissolutiondate: ""
```

Listing B.3: Identify correctness of data types

In this section, PostgreSQL query is executed on a terminal to check the suitability and correctness of data types during data importation from CSV files to PostgreSQL database.

The terminal will return an error and because double quotes are not allow to insert into "date" datatypes. It is caused by the NULL values in company CSV raw data is generated with double quotes and unable to insert them into "date" data types.

```
1
2  ====================================
3  Step 5 - Remove null value with double quotes for data insertion on DATE DATATYPE
4  ====================================
5  yinghua@yinghua-NL8C:~/Documents/FYP/Basic-Company-Data$ sed 's/""//g' Basic-Company-Data-Full.csv > Full.
       csv
6
7  ====================================
8  Step 6 - Import data into companydata table
9  ====================================
10 fyp1=# \copy companydata from 'Full.csv' with header csv;
11 COPY 4077979
```

Listing B.4: Remove null values with double quotes in CSV raw data

As the meaning of null values with double quotes and without quotes are the same. To resolve this problem, *seq* command is required produce new files by remove null values with double quotes stores in each columns. The CSV raw data will import successfully if every columns of data match table's column data types.

B.4 Identify row and column uniqueness in each raw data

Data redundancy and duplication is an inevitable phenomenon found in million of data obtained from on-line sources. Unintentional duplication of records created from data warehouse are hardly avoided. Therefore, the uniqueness of data has to be check in every row and columns for conduct data de-duplication in Phase 2.

B.4.1 Identify row uniqueness

```
======================
Step 1. connect to database
======================

yinghua@yinghua:~$ psql fyp1;
psql (9.5.8)
Type "help" for help.

fyp1#Data redundancy and duplication is an inevitable phenomenon found in million of data obtained from on-
     line sources. Unintentional duplication of records created from the data warehouse 's hard to be
     avoided. Therefore, the uniqueness of data has to be check in every row and columns for conduct data
     de-duplication in Phase 2.

=====================================================
Step 2 - Verify duplicates row in company data tables
=====================================================
fyp1=# select (companydata.*)::text, count(*) from companydata group by companydata.* having count(*) > 1;

 companydata | count
-------------+-------
(0 rows)

=====================================================
Step 3 - Verify duplicates row in subject data tables
=====================================================
fyp1=# select (leo.*)::text, count(*) from leo group by leo.* having count(*) > 1;
  leo    | count
---------+-------
(0 rows)

=====================================================
Step 4 - Verify duplicates row in LEO data tables
=====================================================
fyp1=# select (leo.*)::text, count(*) from leo group by leo.* having count(*) > 1;
  leo    | count
---------+-------
(0 rows)

=====================================================
Step 5: Verify duplicates row in NSPL data table
=====================================================
fyp1=# select (nspl.*)::text, count(*) from nspl group by nspl.* having count(*) > 1;
 nspl | count
------+-------
(0 rows)
```

Listing B.5: Identify row uniqueness

In this section, PostgreSQL query is executed on a terminal to identify duplicates row found in every table. The result shows that there is no row duplication occurs between rows.

B.4.2 Identify column uniqueness

```
======================
Step 1. connect to database
======================

yinghua@yinghua:~$ psql fyp1;
psql (9.5.8)
Type "help" for help.

==============================
Step 2. List structure of table
==============================

fyp1=# \d+ leo

Table "public.leo"
Column              |          Type          |            Modifiers            | Storage  |
--------------------+------------------------+---------------------------------+----------+
ukprn               | integer                | not null                        | plain    |
providername        | character varying(100) | not null                        | extended |
region              | character varying(100) | not null                        | extended |
subject             | character varying(50)  | not null                        | extended |
sex                 | character varying(30)  | not null                        | extended |
yearaftergraduation | character varying(30)  | not null                        | extended |
grads               | character varying(10)  | default NULL::character varying | extended |
unmatched           | character varying(20)  | default NULL::character varying | extended |

(more columns are not shown.....)

============================================================
Step 3. Check duplication of data in selected columns
============================================================

fyp1=# select ukprn, providername, region, count(*) from leo group by ukprn, providername, region having
      count(*) > 1;

============================================================
Step 4. The duplication of columns with rows are return
============================================================

ukprn   |                    providername                      |          region           | count
----------+------------------------------------------------------+---------------------------+-------
10007775 | Queen Mary University of London                      | London                    |   207
10007792 | The University of Exeter                             | South West                |   207
10003324 | The Institute of Cancer Research                     | London                    |   207
10007784 | University College London                            | London                    |   207
10003957 | Liverpool John Moores University                     | North West                |   207
10000886 | The University of Brighton                           | South East                |   207
10007816 | The Royal Central School of Speech and Drama         | London                    |   207
10002681 | Glasgow School of Art                                | Scotland                  |   207
10005545 | Royal Agricultural University                        | South West                |   207
10037449 | University of St Mark and St John                    | South West                |   207
10007144 | The University of East London                        | London                    |   207
10007161 | Teesside University                                  | North East                |   207
10007713 | York St John University                              | Yorkshire and the Humber  |   207
10003863 | Leeds Trinity University                             | Yorkshire and the Humber  |   207

(more duplication data found in columns are not shown......)
```

Listing B.6: Identify column uniqueness

In this section, PostgreSQL query is executed on a terminal to identify duplicates data found in specific columns. The result shows the count of duplication data found in selected columns and lists out in tabular form. This method is proved to be able to identify data duplication occurs within a column.

Appendix C

Golang programming for import CSV into PostgreSQL database

C.1 Introduction

The Go Programming Language possess package csv to reads and write comma-separated values (CSV) files. The package will automatically ignore whitespace, blank lines and delimits commas to read data. In addition, the language also contains a driver to perform CRUD transaction on PostgreSQL database.

The program below imports 100 rows of company data, LEO data and NSPL data from CSV files to PostgreSQL database. Five columns of data are selected from each file to import into this program as proof of concept in this project. The tables will be created in PostgreSQL database before the program is executed.

C.1.1 LEO table for data importation

```
1
2  -- File: fyp1-leo.sql
3  -- Author: Chai Ying Hua
4  -- Database: psql (PostgreSQL) 9.5.8
5
6  -- ====================================
7  -- CHANGES IN V1.1(Sun Aug 27. 2017)
8  --      Create leo table for phase 1 to import data
9  -- ====================================
10
11 create table go_subject  (
12         ukprn int not null,
13         providername varchar(100) not null,
14         region varchar(100) not null,
15         subject varchar(50) not null,
16         sex varchar(30) not null
17 );
```

Listing C.1: PostgreSQL query for LEO table creation.

C.1.2 NSPL table for data importation

```
1
2  -- File: fyp1-nspl.sql
3  -- Author: Chai Ying Hua
4  -- Database: psql (PostgreSQL) 9.5.8
5
6  -- ====================================
7  -- CHANGES IN V1.1(Mon Sep 4. 2017)
8  --      Create nspl table for phase 1 to import data
9  -- ====================================
10
11 create table go_nspl (
12         postcode1 varchar(15) not null,
13         postcode2 varchar(15) not null primary key,
14         date_introduce varchar(10) not null,
15         usertype int not null,
16         position_quality int not null
17 )
```

Listing C.2: PostgreSQL query for NSPL table creation.

C.1.3 LEO table for data importation

```
1
2  -- File: fyp1-company.sql
3  -- Author: Chai Ying Hua
4  -- Database: psql (PostgreSQL) 9.5.8
5
6  -- ========================================================================
7  -- CHANGES IN V1.1(Sun Aug 27. 2017)
8  --      Create companydata table for phase 1 to import data
9  -- ========================================================================
10
11 create table go_company (
12         CompanyName varchar(160) null default null,
13         CompanyNumber varchar(8) not null primary key,
14         CompanyCategory varchar(100) not null,
15         CompanyStatus varchar(70) not null
16         CountryOfOrigin varchar(50) not null
17 );
```

Listing C.3: PostgreSQL query for Company table creation.

C.1.4 Source code of Go program

```
1
2  package main
3
4  import (
5          "bufio"
6          "database/sql"
7          "encoding/csv"
8          "fmt"
9          "io"
10         "os"
11         "strconv"
12
13         _ "github.com/lib/pq"
14 )
15
16 const (
17         DB_USER                       = "yinghua"
18         DB_PASSWORD                   = "123"
19         DB_NAME                       = "fyp1"
20         COMPANY_FILE_DIRECTORY string = "/home/yinghua/Documents/FYP-data/company-data/company-data-full.csv
       "
21         LEO_FILE_DIRECTORY     string = "/home/yinghua/Documents/FYP-data/subject-data/institution-subject-
       data.csv"
22         NSPL_FILE_DIRECTORY    string = "/home/yinghua/Documents/FYP-data/postcode-data/UK-NSPL.csv"
23 )
24
25 type CompanyData struct {
26         name     string
27         number   string
28         category string
29         status   string
30         country  string
31 }
32
33 type LEOData struct {
34         ukprn   int
35         name    string
36         region  string
37         subject string
38         sex     string
39 }
40
41 type NSPLData struct {
42         postcode1      string
43         postcode2      string
44         date_introduce string
45         usertype       int
46         pos_quality    int
47 }
48
49 var db *sql.DB
50
51 //====================================================
52 //function to check error and print error messages
53 //====================================================
54 func checkErr(err error, message string) {
55         if err != nil {
56                 panic(message + " err: " + err.Error())
57         }
58 }
59
60 //====================================================
61 // initialize connection to database
62 //====================================================
63 func initDB() {
64
65         dbInfo := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=disable",
66         DB_USER, DB_PASSWORD, DB_NAME)
67         psqldb, err := sql.Open("postgres", dbInfo)
68         checkErr(err, "psql open")
69         db = psqldb
70
71 }
72
73 //====================================================
74 // Import company data
75 //====================================================
76 func importCompanyData() {
77
78         var sStmt string = "insert into go_company values ($1, $2, $3, $4, $5)"
79
80         stmt, err := db.Prepare(sStmt)
81         checkErr(err, "Prepare Stmt")
82
83         // Open CSV files
84         csvFile, err := os.Open(COMPANY_FILE_DIRECTORY)
```

```
85          checkErr(err, "Open CSV")
86
87          defer csvFile.Close()
88
89          // Create a new reader.
90          reader := csv.NewReader(bufio.NewReader(csvFile))
91
92          for i := 0; i <= 100; i++ {
93                  record, err := reader.Read()
94
95                  // skipped the first line
96                  if i == 0 {
97                          continue
98                  }
99
100                 // Stop at EOF.
101                 if err == io.EOF {
102                         break
103                 }
104
105                 company := CompanyData{
106                         name:     record[0],
107                         number:   record[1],
108                         category: record[10],
109                         status:   record[11],
110                         country:  record[12],
111                 }
112
113                 stmt.Exec(company.name, company.number, company.category, company.status, company.country)
114                 checkErr(err, "Company Data importation")
115         }
116 }
117
118 //=================================================
119 // Import LEO data
120 //=================================================
121 func importSubjectData() {
122
123         var sStmt string = "insert into go_subject values ($1, $2, $3, $4, $5)"
124
125         stmt, err := db.Prepare(sStmt)
126         checkErr(err, "Prepare Subject Stmt")
127
128         csvFile, err := os.Open(LEO_FILE_DIRECTORY)
129         checkErr(err, "Open LEO CSV")
130
131         defer csvFile.Close()
132
133         // Create a new reader.
134         reader := csv.NewReader(bufio.NewReader(csvFile))
135
136         for i := 0; i <= 100; i++ {
137                 record, err := reader.Read()
138
139                 // skipped the first line
140                 if i == 0 {
141                         continue
142                 }
143
144                 // Stop at EOF.
145                 if err == io.EOF {
146                         break
147                 }
148
149                 integer, err := strconv.Atoi(record[0])
150                 checkErr(err, "Convert UKRPN to Integer")
151
152                 subject := LEOData{
153                         ukprn:   integer,
154                         name:    record[1],
155                         region:  record[2],
156                         subject: record[3],
157                         sex:     record[4],
158                 }
159
160                         stmt.Exec(subject.ukprn, subject.name, subject.region, subject.subject, subject.sex)
161                         checkErr(err, "Subject Data importation")
162                 }
163 }
164
165 //=================================================
166 // Import NSPL data
167 //=================================================
168 func importNSPLData() {
169
170         var sStmt string = "insert into go_nspl values ($1, $2, $3, $4, $5)"
171
172         stmt, err := db.Prepare(sStmt)
173         checkErr(err, "Prepare Postcode Stmt")
```

```
174
175         csvFile, err := os.Open(NSPL_FILE_DIRECTORY)
176         checkErr(err, "Open Postcode CSV")
177
178         defer csvFile.Close()
179
180         // Create a new reader.
181         reader := csv.NewReader(bufio.NewReader(csvFile))
182
183         for i := 0; i <= 100; i++ {
184                 record, err := reader.Read()
185
186                 // skipped the first line
187                 if i == 0 {
188                 continue
189                 }
190
191                 // Stop at EOF.
192                 if err == io.EOF {
193                 break
194                 }
195
196                 userInt, err := strconv.Atoi(record[4])
197                 checkErr(err, "Convert Usertype to Integer")
198
199                 posInt, err := strconv.Atoi(record[7])
200                 checkErr(err, "Convert Usertype to Integer")
201
202                 postcode := NSPLData {
203                 postcode1:      record[0],
204                 postcode2:      record[1],
205                 date_introduce: record[3],
206                 usertype:       userInt,
207                 pos_quality:    posInt,
208                 }
209
210                 stmt.Exec(postcode.postcode1, postcode.postcode2, postcode.date_introduce, postcode.usertype
      , postcode.pos_quality)
211                 checkErr(err, "Postcode Data importation")
212         }
213 }
214
215 func main() {
216
217         initDB()
218         importCompanyData()
219         importSubjectData()
220         importNSPLData()
221
222 }
223
224 /**
225
226 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build import-csv-psql.go
227 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run import-csv-psql.go
228
229 real    0m3.647s
230 user    0m0.328s
231 sys     0m0.096s
232 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$
233
234 **/
```

Listing C.4: Source code of Go program

Appendix D

Sequential and concurrent programming with Golang on PostgreSQL database retrieval.

D.1 Golang Sequential Program Source Code

```
1
2  package main
3
4          import (
5          "database/sql"
6          "fmt"
7          "time"
8
9          _ "github.com/lib/pq"
10 )
11
12 const (
13         DB_USER     = "yinghua"
14         DB_PASSWORD = "123"
15         DB_NAME     = "fyp1"
16 )
17
18 var db *sql.DB
19
20 //=====================================================
21 //function to check error and print error messages
22 //=====================================================
23 func checkErr(err error, message string) {
24         if err != nil {
25                 panic(message + " err: " + err.Error())
26         }
27 }
28
29 //=====================================================
30 // initialize connection with database
31 //=====================================================
32 func initDB() {
33
34         dbInfo := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=disable",
35         DB_USER, DB_PASSWORD, DB_NAME)
36         psqldb, err := sql.Open("postgres", dbInfo)
37         checkErr(err, "Initialize database")
38         db = psqldb
39
40 }
41
42 //=====================================================
43 // retrieve data from company table in postgres
44 //=====================================================
45 func retrieveCompanyData() {
```

```
46
47          fmt.Println("Start retrieve company data from database ... ")
48          start := time.Now()
49
50          time.Sleep(time.Second * 2)
51
52          rows, err := db.Query("SELECT c.companyname, c.companynumber, c.companycategory, c.companystatus, c.
        countryoforigin FROM companydata AS c ORDER BY c.companynumber limit 100;")
53          checkErr(err, "Query Company DB rows")
54
55          var (
56                  companyname     string
57                  companynumber   string
58                  companycategory string
59                  companystatus   string
60                  countryoforigin string
61          )
62
63          for rows.Next() {
64                  err = rows.Scan(&companyname, &companynumber, &companycategory, &companystatus, &
        countryoforigin)
65                  checkErr(err, "Read company data rows")
66                  //fmt.Printf("%8v %3v %6v %6v %6v\n", companyname, companynumber, companycategory,
        companystatus, countryoforigin)
67          }
68
69          fmt.Println("Data retrieval of company data SUCCESS! ")
70          fmt.Printf("%.8fs elapsed\n\n", time.Since(start).Seconds())
71
72  }
73
74  //=====================================================
75  // retrieve data from postcode table in postgres
76  //=====================================================
77  func retrievePostcodeData() {
78
79          fmt.Println("Start retrieve postcode data from database ... ")
80          start := time.Now()
81
82          time.Sleep(time.Second * 2)
83
84          rows, err := db.Query("SELECT postcode1, postcode2, date_introduce, usertype, position_quality FROM
        go_nspl LIMIT 50")
85          checkErr(err, "Query Postcode DB rows")
86
87          var (
88                  postcode1        string
89                  postcode2        string
90                  date_introduce   string
91                  usertype         int
92                  position_quality int
93          )
94
95          for rows.Next() {
96                  err = rows.Scan(&postcode1, &postcode2, &date_introduce, &usertype, &position_quality)
97                  checkErr(err, "Read postcode data rows")
98                  //fmt.Printf("%6v %8v %6v %6v %6v\n", postcode1, postcode2, date_introduce, usertype,
        position_quality)
99          }
100
101         fmt.Print("Data retrieval of postcode data SUCCESS! ")
102         fmt.Printf("%.8fs elapsed\n\n", time.Since(start).Seconds())
103
104 }
105
106 //=====================================================
107 // retrieve data from subject table in postgres
108 //=====================================================
109 func retrieveSubjectData() {
110
111         fmt.Println("Start retrieve LEO data from database ... ")
112         start := time.Now()
113
114         time.Sleep(time.Second * 2)
115
116         rows, err := db.Query("SELECT ukprn, providername, region, subject, sex FROM go_subject LIMIT 50")
117         checkErr(err, "Query subject DB rows")
118
119         var (
120                 ukprn   int
121                 name    string
122                 region  string
123                 subject string
124                 sex     string
125         )
126
127         for rows.Next() {
128                 err = rows.Scan(&ukprn, &name, &region, &subject, &sex)
129                 checkErr(err, "Read subject data rows")
```

```
130                 //fmt.Printf("%6v %8v %6v %6v %6v\n", ukprn, name, region, subject, sex)
131         }
132
133         fmt.Print("Data retrieval of subject data SUCCESS! ")
134         fmt.Printf(" %.8fs elapsed\n\n", time.Since(start).Seconds())
135
136 }
137
138 //=====================================================
139 // Main function
140 //=====================================================
141 func main() {
142
143         // get the time before execution
144         start := time.Now()
145
146         initDB()
147         retrieveCompanyData()
148         retrievePostcodeData()
149         retrieveSubjectData()
150
151         // print the time after execution
152         fmt.Printf("Total execution %.5fs elapsed\n", time.Since(start).Seconds())
153
154 }
155
156 /**
157
158 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build sequential-psql.go
159 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run sequential-psql.go
160 Start retrieve company data from database ...
161 Data retrieval of company data SUCCESS!
162 2.00721985s elapsed
163
164 Start retrieve postcode data from database ...
165 Data retrieval of postcode data SUCCESS!
166 2.00144933s elapsed
167
168 Start retrieve LEO data from database ...
169 Data retrieval of subject data SUCCESS!
170 2.00131415s elapsed
171
172 Total execution 6.01005s elapsed
173
174 real    0m6.252s
175 user    0m0.272s
176 sys             0m0.032s
177
178
179 **/
```

Listing D.1: Golang Sequential Program Source Code

D.1.1 Golang Concurrent Program Source Code

```
1
2  package main
3
4  import (
5          "database/sql"
6          "fmt"
7          "time"
8
9          _ "github.com/lib/pq"
10 )
11
12 //=====================================================
13 // database information
14 //=====================================================
15 const (
16         DB_USER     = "yinghua"
17         DB_PASSWORD = "123"
18         DB_NAME     = "fyp1"
19 )
20
21 var (
22         db          *sql.DB
23         numChannels int = 3
24 )
25
26 //=====================================================
27 // function to check error and print error messages
28 //=====================================================
29 func checkErr(err error, message string) {
30         if err != nil {
31                 panic(message + " err: " + err.Error())
32         }
33 }
34
35 //=====================================================
36 // initialize connection with database
37 //=====================================================
38 func initDB() {
39
40         dbInfo := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=disable",
41         DB_USER, DB_PASSWORD, DB_NAME)
42         psqldb, err := sql.Open("postgres", dbInfo)
43         checkErr(err, "Initialize database")
44         db = psqldb
45
46 }
47
48 //=====================================================
49 // retrieve company data store in postgres database
50 //=====================================================
51 func retrieveCompanyData(ch_company chan string) {
52
53         fmt.Println("Start retrieve company data from database ... ")
54         start := time.Now()
55
56         time.Sleep(time.Second * 2)
57
58         rows, err := db.Query("SELECT c.companyname, c.companynumber, c.companycategory, c.companystatus, c.
       countryoforigin FROM companydata AS c ORDER BY c.companynumber limit 100;")
59         checkErr(err, "Query Company DB rows")
60
61         var (
62                 companyname     string
63                 companynumber   string
64                 companycategory string
65                 companystatus   string
66                 countryoforigin string
67         )
68
69         for rows.Next() {
70                 err = rows.Scan(&companyname, &companynumber, &companycategory, &companystatus, &
       countryoforigin)
71                 checkErr(err, "Read company data rows")
72                 //fmt.Printf("%8v %3v %6v %6v %6v\n", companyname, companynumber, companycategory,
       companystatus, countryoforigin)
73         }
74
75         fmt.Printf("%.8fs elapsed\n", time.Since(start).Seconds())
76         ch_company <- "Retrieval of company data success. \n"
77 }
78
79 //=====================================================
80 // retrieve postcode data store in postgres database
81 //=====================================================
82 func retrievePostcodeData(ch_postcode chan string) {
83
```

```
84          fmt.Println("Start retrieve postcode data from database ... ")
85          start := time.Now()
86
87          time.Sleep(time.Second * 2)
88
89          rows, err := db.Query("SELECT postcode1, postcode2, date_introduce, usertype, position_quality FROM
        go_nspl LIMIT 50")
90          checkErr(err, "Query Postcode DB rows")
91
92          var (
93                  postcode1        string
94                  postcode2        string
95                  date_introduce   string
96                  usertype         int
97                  position_quality int
98          )
99
100         for rows.Next() {
101                 err = rows.Scan(&postcode1, &postcode2, &date_introduce, &usertype, &position_quality)
102                 checkErr(err, "Read postcode data rows")
103                 //fmt.Printf("%6v %8v %6v %6v %6v\n", postcode1, postcode2, date_introduce, usertype,
        position_quality)
104         }
105
106         fmt.Printf("%.8fs elapsed\n", time.Since(start).Seconds())
107                 ch_postcode <- "Retrieval of postcode success. \n"
108         }
109
110         //=====================================================
111         // retrieve subject data store in postgres database
112         //=====================================================
113         func retrieveSubjectData(ch_subject chan string) {
114
115                 fmt.Println("Start retrieve LEO data from database ... ")
116                 start := time.Now()
117
118                 time.Sleep(time.Second * 2)
119
120                 rows, err := db.Query("SELECT ukprn, providername, region, subject, sex FROM go_subject
        LIMIT 50")
121                 checkErr(err, "Query subject DB rows")
122
123                 var (
124                         ukprn   int
125                         name    string
126                         region  string
127                         subject string
128                         sex     string
129                 )
130
131                 for rows.Next() {
132                         err = rows.Scan(&ukprn, &name, &region, &subject, &sex)
133                         checkErr(err, "Read subject data rows")
134                         //fmt.Printf("%6v %8v %6v %6v %6v\n", ukprn, name, region, subject, sex)
135                 }
136
137                 fmt.Printf("%.8fs elapsed\n", time.Since(start).Seconds())
138                         ch_subject <- "Retrieval of subject data success. \n"
139                 }
140
141                 // select function
142                 func goSelect(ch_company, ch_subject, ch_postcode chan string) {
143
144                 for i := 0; i < numChannels; i++ {
145
146                         select {
147                         case msg1 := <-ch_postcode:
148                                 fmt.Println(msg1)
149                         case msg2 := <-ch_company:
150                                 fmt.Println(msg2)
151                         case msg3 := <-ch_subject:
152                                 fmt.Println(msg3)
153
154                 }
155
156         }
157 }
158
159 //=====================================================
160 // Main function
161 //=====================================================
162 func main() {
163
164         // make three channel for three functions
165         ch_company := make(chan string)
166         ch_subject := make(chan string)
167         ch_postcode := make(chan string)
168
169         // get the time before execution
```

```
170         start := time.Now()
171
172         initDB()
173
174         //go routines
175         go retrieveCompanyData(ch_company)
176         go retrieveSubjectData(ch_subject)
177         go retrievePostcodeData(ch_postcode)
178
179         goSelect(ch_company, ch_subject, ch_postcode)
180
181         // obtain the time after execution
182         fmt.Printf("Total execution %.5fs elapsed\n", time.Since(start).Seconds())
183
184 }
185
186 /**
187
188 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build concurrent-psql.go
189 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run concurrent-psql.go
190 Start retrieve postcode data from database ...
191 Start retrieve company data from database ...
192 Start retrieve LEO data from database ...
193 2.00615007s elapsed
194 Retrieval of subject data success.
195
196 2.00661550s elapsed
197 Retrieval of postcode success.
198
199 2.00745319s elapsed
200 Retrieval of company data success.
201
202 Total execution 2.00754s elapsed
203
204 real    0m2.268s
205 user    0m0.244s
206 sys             0m0.076s
207
208
209
210 **/
211
212 )
```

Listing D.2: Golang Concurrent Program Source Code

Appendix E

Sequential and concurrent programming with Golang on reading CSV file

E.1 Golang Sequential Program Source Code

```
1
2  package main
3
4  import (
5          "bufio"
6          "database/sql"
7          "encoding/csv"
8          "fmt"
9          "io"
10         "os"
11         "time"
12
13         _ "github.com/lib/pq"
14 )
15
16 const (
17         DB_USER                      = "yinghua"
18         DB_PASSWORD                  = "123"
19         DB_NAME                      = "fyp1"
20         COMPANY_FILE_DIRECTORY string = "/home/yinghua/Documents/FYP-data/company-data/company-data-full.csv
       "
21         LEO_FILE_DIRECTORY     string = "/home/yinghua/Documents/FYP-data/subject-data/institution-subject-
       data.csv"
22         NSPL_FILE_DIRECTORY    string = "/home/yinghua/Documents/FYP-data/postcode-data/UK-NSPL.csv"
23 )
24
25 var db *sql.DB
26
27 // function to check error and print error messages
28 func checkErr(err error, message string) {
29         if err != nil {
30                 panic(message + " err: " + err.Error())
31         }
32 }
33
34 func read_CompanyCSV() {
35
36         fmt.Println("Start reading 100 row Company CSV data")
37
38         time.Sleep(time.Second * 2)
39
40         csvFile, err := os.Open(COMPANY_FILE_DIRECTORY)
41         checkErr(err, "Open CSV")
42
43         defer csvFile.Close()
```

```
44
45          // Create a new reader.
46          reader := csv.NewReader(bufio.NewReader(csvFile))
47
48          for i := 0; i <= 100; i++ {
49                  _, err := reader.Read()
50
51                  // skipped the first line
52                  if i == 0 {
53                          continue
54                  }
55
56                  // Stop at EOF.
57                  if err == io.EOF {
58                          break
59                  }
60
61          }
62
63          fmt.Println("Finish reading Company CSV data")
64
65  }
66
67  func read_LEOCSV() {
68
69          fmt.Println("Start reading 100 row LEO CSV data")
70
71          time.Sleep(time.Second * 2)
72
73          csvFile, err := os.Open(LEO_FILE_DIRECTORY)
74          checkErr(err, "Open LEO CSV")
75
76          defer csvFile.Close()
77
78          // Create a new reader.
79          reader := csv.NewReader(bufio.NewReader(csvFile))
80
81          for i := 0; i <= 100; i++ {
82                  _, err := reader.Read()
83
84                  // skipped the first line
85                  if i == 0 {
86                          continue
87                  }
88
89                  // Stop at EOF.
90                  if err == io.EOF {
91                          break
92                  }
93          }
94
95          fmt.Println("Finish readying LEO CSV data")
96
97  }
98
99  func read_NSPLCSV() {
100
101         fmt.Println("Start reading 100 row NSPL CSV data")
102
103         time.Sleep(time.Second * 2)
104
105         csvFile, err := os.Open(NSPL_FILE_DIRECTORY)
106         checkErr(err, "Open Postcode CSV")
107
108         defer csvFile.Close()
109
110         // Create a new reader.
111         reader := csv.NewReader(bufio.NewReader(csvFile))
112
113         for i := 0; i <= 100; i++ {
114                 _, err := reader.Read()
115
116         // skipped the first line
117                 if i == 0 {
118                         continue
119                 }
120
121         // Stop at EOF.
122                 if err == io.EOF {
123                         break
124                 }
125         }
126
127         fmt.Println("Finish readying LEO CSV data")
128
129 }
130
131 func main() {
132
```

```
133         // get the time before execution
134         start := time.Now()
135 
136         read_CompanyCSV()
137         read_LEOCSV()
138         read_NSPLCSV()
139 
140         // obtain the time after execution
141         fmt.Printf("Total execution %.5fs elapsed\n", time.Since(start).Seconds())
142 
143 }
144 
145 /**
146 
147 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build sequential-read-csv.go
148 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run sequential-read-csv.
        go
149 Start reading 100 row Company CSV data
150 Finish reading Company CSV data
151 Start reading 100 row LEO CSV data
152 Finish readying LEO CSV data
153 Start reading 100 row NSPL CSV data
154 Finish readying LEO CSV data
155 Total execution 6.00823s elapsed
156 
157 real    0m6.285s
158 user    0m0.316s
159 sys             0m0.056s
160 **/
```

Listing E.1: Golang Sequential Program Source Code

E.1.1 Golang Concurrent Program Source Code

```
1 
2  package main
3 
4  import (
5          "bufio"
6          "database/sql"
7          "encoding/csv"
8          "fmt"
9          "io"
10         "os"
11         "time"
12 
13         _ "github.com/lib/pq"
14 )
15 
16 const (
17         DB_USER                      = "yinghua"
18         DB_PASSWORD                  = "123"
19         DB_NAME                      = "fyp1"
20         COMPANY_FILE_DIRECTORY string = "/home/yinghua/Documents/FYP-data/company-data/company-data-full.csv
       "
21         LEO_FILE_DIRECTORY     string = "/home/yinghua/Documents/FYP-data/subject-data/institution-subject-
       data.csv"
22         NSPL_FILE_DIRECTORY    string = "/home/yinghua/Documents/FYP-data/postcode-data/UK-NSPL.csv"
23 )
24 
25 var (
26         db          *sql.DB
27         numChannels int = 3
28 )
29 
30 // function to check error and print error messages
31 func checkErr(err error, message string) {
32         if err != nil {
33                 panic(message + " err: " + err.Error())
34         }
35 }
36 
37 func read_CompanyCSV(ch_company chan string) {
38 
39         fmt.Println("Start reading 100 row Company CSV data")
40 
41         time.Sleep(time.Second * 2)
42 
43         csvFile, err := os.Open(COMPANY_FILE_DIRECTORY)
44         checkErr(err, "Open CSV")
45 
```

```
46          defer csvFile.Close()
47
48          // Create a new reader.
49          reader := csv.NewReader(bufio.NewReader(csvFile))
50
51          for i := 0; i <= 100; i++ {
52                  _, err := reader.Read()
53
54                  // skipped the first line
55                  if i == 0 {
56                          continue
57                  }
58
59                  // Stop at EOF.
60                  if err == io.EOF {
61                          break
62                  }
63          }
64
65          ch_company <- "Finish readying LEO CSV data"
66
67  }
68
69  func read_LEOCSV(ch_leo chan string) {
70
71          fmt.Println("Start reading 100 row LEO CSV data")
72
73          time.Sleep(time.Second * 2)
74
75          csvFile, err := os.Open(LEO_FILE_DIRECTORY)
76          checkErr(err, "Open LEO CSV")
77
78          defer csvFile.Close()
79
80          // Create a new reader.
81          reader := csv.NewReader(bufio.NewReader(csvFile))
82
83          for i := 0; i <= 100; i++ {
84                  _, err := reader.Read()
85
86                  // skipped the first line
87                  if i == 0 {
88                          continue
89                  }
90
91                  // Stop at EOF.
92                  if err == io.EOF {
93                          break
94                  }
95          }
96
97          ch_leo <- "Finish reading LEO CSV data"
98
99  }
100
101 func read_NSPLCSV(ch_nspl chan string) {
102
103         fmt.Println("Start reading 100 row NSPL CSV data")
104
105         time.Sleep(time.Second * 2)
106
107         csvFile, err := os.Open(NSPL_FILE_DIRECTORY)
108         checkErr(err, "Open Postcode CSV")
109
110         defer csvFile.Close()
111
112         // Create a new reader.
113         reader := csv.NewReader(bufio.NewReader(csvFile))
114
115         for i := 0; i <= 100; i++ {
116                 _, err := reader.Read()
117
118                 // skipped the first line
119                 if i == 0 {
120                         continue
121                 }
122
123                 // Stop at EOF.
124                 if err == io.EOF {
125                         break
126                 }
127         }
128
129         ch_nspl <- "Finish reading NSPL CSV data"
130 }
131
132 // select function
133 func goSelect(ch_company, ch_leo, ch_nspl chan string) {
134
```

```
135         for i := 0; i < numChannels; i++ {
136
137                 select {
138                         case msg1 := <-ch_leo:
139                                 fmt.Println(msg1)
140                         case msg2 := <-ch_company:
141                         fmt.Println(msg2)
142                                 case msg3 := <-ch_nspl:
143                         fmt.Println(msg3)
144
145                 }
146
147         }
148 }
149
150 func main() {
151
152         // make three channel for three functions
153         ch_company := make(chan string)
154         ch_leo := make(chan string)
155         ch_nspl := make(chan string)
156
157         // get the time before execution
158         start := time.Now()
159
160         go read_CompanyCSV(ch_company)
161         go read_LEOCSV(ch_leo)
162         go read_NSPLCSV(ch_nspl)
163
164         goSelect(ch_company, ch_leo, ch_nspl)
165
166         // obtain the time after execution
167         fmt.Printf("Total execution %.5fs elapsed\n", time.Since(start).Seconds())
168
169 }
170
171 /**
172
173 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build concurrent-read-csv.go
174 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run concurrent-read-csv.
      go
175 Start reading 100 row NSPL CSV data
176 Start reading 100 row Company CSV data
177 Start reading 100 row LEO CSV data
178 Finish reading LEO CSV data
179 Finish reading NSPL CSV data
180 Finish readying LEO CSV data
181 Total execution 2.00376s elapsed
182
183 real    0m2.243s
184 user    0m0.264s
185 sys             0m0.044s
186
187 **/
```

Listing E.2: Golang Concurrent Program Source Code

Appendix F

Result of Sequential and concurrent programming with Golang on process CSV

F.1 Linux command for Go program execution

```
1  ================================================================
2  Step 1 - Build sequential-read-csv.go
3  ================================================================
4  yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build sequential-read-csv.go
5
6  ================================================================
7  Step 2 - Execute sequential-read-csv.go program
8  ================================================================
9  yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run sequential-read-csv.
       go
10 Start reading 100 row Company CSV data
11 Finish reading Company CSV data
12 Start reading 100 row LEO CSV data
13 Finish readying LEO CSV data
14 Start reading 100 row NSPL CSV data
15 Finish readying LEO CSV data
16 Total execution 6.00823s elapsed
17
18 real    0m6.285s
19 user    0m0.316s
20 sys     0m0.056s
21
22 ================================================================
23 Step 3 - Build concurrent-read-csv.go
24 ================================================================
25 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build concurrent-read-csv.go
26
27 ================================================================
28 Step 4 - Execute concurrent-read-csv.go program
29 ================================================================
30 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run concurrent-read-csv.
       go
31 Start reading 100 row NSPL CSV data
32 Start reading 100 row Company CSV data
33 Start reading 100 row LEO CSV data
34 Finish reading LEO CSV data
35 Finish reading NSPL CSV data
36 Finish readying LEO CSV data
37 Total execution 2.00376s elapsed
38
39 real    0m2.243s
40 user    0m0.264s
41 sys     0m0.044s
```

Listing F.1: Linux command for Go program execution

F.2 Result of Golang programming on process CSV

Elapsed Time	**sequential-read-csv.go**	**concurrent-read-csv.go**
real	6.285s	2.243s
user	0.316s	0.264s
sys	0.056s	0.044s

Table F.1: Result of Golang programming on process CSV raw data

Appendix G

Result of Sequential and concurrent programming with Golang on process PostgreSQL database.

G.1 Linux command for Go program execution

```
1
2  ================================================================
3  Step 1 - Build sequential-psql.go
4  ================================================================
5  yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build sequential-psql.go
6
7  ================================================================
8  Step 2 - Execute sequential-psql.go program
9  ================================================================
10 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run sequential-psql.go
11 Start retrieve company data from database ...
12 Data retrieval of company data SUCCESS!
13 2.00721985s elapsed
14
15 Start retrieve postcode data from database ...
16 Data retrieval of postcode data SUCCESS!
17 2.00144933s elapsed
18
19 Start retrieve LEO data from database ...
20 Data retrieval of subject data SUCCESS!
21 2.00131415s elapsed
22
23 Total execution 6.01005s elapsed
24
25 real    0m6.252s
26 user    0m0.272s
27 sys     0m0.032s
28
29 ================================================================
30 Step 3 - Build concurrent-psql.go
31 ================================================================
32 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build concurrent-psql.go
33
34 ================================================================
35 Step 4 - Execute concurrent-psql.go program
36 ================================================================
37 yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run concurrent-psql.go
38 Start retrieve postcode data from database ...
39 Start retrieve company data from database ...
40 Start retrieve LEO data from database ...
```

```
41 2.00615007s elapsed
42 Retrieval of subject data success.
43
44 2.00661550s elapsed
45 Retrieval of postcode success.
46
47 2.00745319s elapsed
48 Retrieval of company data success.
49
50 Total execution 2.00754s elapsed
51
52 real    0m2.268s
53 user    0m0.244s
54 sys     0m0.076s
```

Listing G.1: Linux command for Go program execution

G.2 Result of Golang programming on process PostgreSQL database

Elapsed Time	**sequential-psql.go**	**concurrent-psql.go**
real	6.252s	2.268s
user	0.272s	0.244s
sys	0.032s	0.076s

Table G.1: Result of Golang programming on PostgreSQL database

Appendix H

Result of import data from CSV file to PostgreSQL database with Golang

H.1 Linux command for import data

```
1  ====================================
2  Step 1 - Connect to FYP1 database
3  ====================================
4
5  yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ psql fyp1;
6  psql (9.5.8)
7  Type "help" for help.
8
9  fyp1=#
10
11 ====================================
12 Step 2 - Check number of tables
13 ====================================
14 fyp1=# \d
15 List of relations
16 Schema |    Name     | Type  |  Owner
17 --------+-------------+-------+---------
18 public | companydata | table | yinghua
19 public | leo         | table | yinghua
20 public | nspl        | table | yinghua
21 (3 rows)
22
23 ====================================================================
24 Step 3 - Create go_company table ready for importation
25 ====================================================================
26 fyp1=# create table go_company (companyname varchar(160) null default null, companynumber varchar(8) not
       null primary key, companycategory varchar(100) not null, companystatus varchar(70) not null,
       countryoforigin varchar(50) not null );
27 CREATE TABLE
28
29 ====================================================================
30 Step 4 - Create go_subject table ready for importation
31 ====================================================================
32 fyp1=# create table go_subject (ukprn int not null, providername varchar(100) not null, region varchar(100)
       not null, subject varchar(50) not null, sex varchar(30) not null );
33 CREATE TABLE
34
35 ====================================================================
36 Step 5 - Create go_nspl table ready for importation
37 ====================================================================
38 fyp1=# create table go_nspl (postcode1 varchar(15) not null, postcode2 varchar(15) not null primary key,
       date_introduce varchar(10) not null,usertype int not null, position_quality int not null);
39
40 ====================================================================
41 Step 6 - Check number of data in each respective table
```

```
================================================================
fyp1=# \d
List of relations
Schema |    Name     | Type  |  Owner
--------+-------------+-------+---------
public | companydata | table | yinghua
public | go_company  | table | yinghua
public | go_nspl     | table | yinghua
public | go_subject  | table | yinghua
public | leo         | table | yinghua
public | nspl        | table | yinghua
(6 rows)

fyp1=# select count(*) from go_company;
count
-------
0
(1 row)

fyp1=# select count(*) from go_nspl;
count
-------
0
(1 row)

fyp1=# select count(*) from go_subject;
count
-------
0
(1 row)

================================================================
Step 7 - List all the Go files
================================================================
yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ ls -l
total 33084
-rwxrwxr-x 1 yinghua yinghua 4903560 Sep 16 23:10 concurrent-psql
-rw-rw-r-- 1 yinghua yinghua    5487 Sep 17 23:25 concurrent-psql.go
-rwxrwxr-x 1 yinghua yinghua 4724204 Sep 16 23:13 concurrent-read-csv
-rw-rw-r-- 1 yinghua yinghua    3571 Sep 16 23:13 concurrent-read-csv.go
-rwxrwxr-x 1 yinghua yinghua 4858407 Sep 17 23:01 import-csv-psql
-rw-rw-r-- 1 yinghua yinghua    5146 Sep 17 23:02 import-csv-psql.go
-rwxrwxr-x 1 yinghua yinghua 4895323 Sep 16 23:09 sequential-psql
-rw-rw-r-- 1 yinghua yinghua    4728 Sep 17 23:20 sequential-psql.go
-rwxrwxr-x 1 yinghua yinghua 4720029 Sep 16 23:12 sequential-read-csv
-rw-rw-r-- 1 yinghua yinghua    3002 Sep 16 23:12 sequential-read-csv.go

================================================================================
Step 8 - Build and run import-csv-psql.go to import data from CSV to PostgreSQL
================================================================================
yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ go build import-csv-psql.go
yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ time go run import-csv-psql.go

real    0m3.622s
user    0m0.312s
sys     0m0.088s

================================================================================
Step 9 - Connect to database and verified whether the importation is success
================================================================================
yinghua@yinghua:~/Desktop/apps/eclipse-workspace/FYP1/src/postgres-process$ psql fyp1;
psql (9.5.8)
Type "help" for help.

fyp1=# select count(*) from go_company;
count
-------
100
(1 row)

fyp1=# select count(*) from go_nspl;
count
-------
100
(1 row)

fyp1=# select count(*) from go_subject;
count
-------
100
(1 row)
```

Listing H.1: Linux command for import data

Appendix I

Data Collection

I.1 Data Dictionary of Raw Datasets

I.1.1 Phase 1 Longitudinal Education Outcomes (LEO) Data Dictionary

Longitudinal Education Outcomes Data Dictionary

Data	Data Type	NULL	Description
UKPRN	int	NOT NULL	UK Provider Reference Number.
providerName	varchar(100)	NOT NULL	University name that provide the subject
Region	varchar(50)	NOT NULL	UK Region
subject	varchar(50)	NOT NULL	Subject studied.
sex	varchar(30)	NOT NULL	Sex of graduate.
yearsAfterGraduation	int	NOT NULL	Number of years after graduation.
grads	int	NULL DEFAULT 0	Number of graduates included in calculations.
unmatched	varchar(20)	NULL DEFAULT NULL	Percentage of graduates that have been classed as unmatched.
matched	varchar(20)	NULL DEFAULT NULL	Number of graduates that have been classed as matched.
activityNotCaptured	varchar(20)	NULL DEFAULT NULL	Percentage of matched graduates whose activity could not be captured.
noSustDest	varchar(20)	NULL DEFAULT NULL	Percentage of matched graduates with an unsustained destination.
sustEmpOnly	varchar(20)	NULL DEFAULT NULL	Percentage of graduates with a record or sustained employment only.
sustEmp	varchar(20)	NULL DEFAULT NULL	Percentage of graduates with a record or sustained employment (these graduates may or may not have a further study record in addition to a sustained employment record).
sustEmpFSorBoth	varchar(20)	NULL DEFAULT NULL	Percentage of graduates with a record or sustained employment, a record of further study, or both.
earningsInclude	varchar(20)	NULL DEFAULT NULL	Number of matched graduates included in earnings calculations.
lowerAnnEarn	varchar(20)	NULL DEFAULT NULL	Annualised earnings lower quartile.
medianAnnEarn	varchar(20)	NULL DEFAULT NULL	Median annualised earnings.
upperAnnEarn	varchar(20)	NULL DEFAULT NULL	Annualised earnings upper quartile.
POLARGrpOne	varchar(20)	NULL DEFAULT NULL	Percentage of graduates in POLAR group 1 (of those eligible to be included in POLAR calculations).
POLARGrpOneIncluded	varchar(20)	NULL DEFAULT NULL	Percentage of graduates included in POLAR calculations .
prAttBand	varchar(20)	NULL DEFAULT NULL	Prior attainment band.
prAttIncluded	varchar(20)	NULL DEFAULT NULL	Percentage of graduates included in prior attainment calculations.

FIGURE I.1: Phase 1 Longitudinal Education Outcomes (LEO) Data Dictionary

I.1.2 Phase 1 Company Data Dictionary

Basic Company Data Dictionary

	Data Type	NULL	Description
CompanyName	VARCHAR(160)	NULL DEFAULT NULL	
CompanyNumber	VARCHAR(8)	NOT NULL (PK)	Company number
CareOf	VARCHAR(100)	NULL	Registered Office Address Care Of
POBox	VARCHAR(10)	NULL	Registered Office Address PO BOX
AddressLine1 (House number and street)	VARCHAR(300)	NULL	Registered Office Address Line 1
AddressLine2 (Area)	VARCHAR(300)	NULL	Registered Office Address Line 2
PostTown	VARCHAR(50)	NULL	Registered Office Address Post Town
County	VARCHAR(50)	NULL	Registered Office Address County
Country	VARCHAR(50)	NULL	Registered Office Address Country
PostCode	VARCHAR(20)	NULL	Registered Office Address Postcode
CompanyCategory	VARCHAR(100)	NOT NULL	Registered Office Address Company category
CompanyStatus	VARCHAR(70)	NOT NULL	Registered Office Address Company Status
CountryofOrigin	VARCHAR(50)	NOT NULL	Registered Office Address Country of Origin
DissolutionDate	DATE	NULL	Registered Office Address Dissolution date
IncorporationDate	DATE	NULL	Registered Office Address Incorporation date
AccountingRefDay	INT	NULL DEFAULT 0	Accounting references day
AccountingRefMonth	INT	NULL DEFAULT 0	Accounting Reference months
Account_NextDueDate	DATE	NULL DEFAULT NULL	Account's next due date
Account_LastMadeUpDate	DATE	NULL DEFAULT NULL	Account's last made up date
AccountCategory	VARCHAR(30)	NULL	Account category
Return_NextDueDate	DATE	NULL DEFAULT NULL	Return next due date
Return_LastMadeUpDate	DATE	NULL DEFAULT NULL	Return last made up date
NumMortCharges	INT	NOT NULL	Number of Mortgages charges
NumMortOutstanding	INT	NOT NULL	Number of Mortgages outstanding
NumMortPartSatisfied	INT	NOT NULL	Number of Mortgages Partial satisfied
NumMortSatisfied	INT	NOT NULL	Number of Mortgages satisfied
SICCode1	VARCHAR(170)	NULL	SIC Codes 1
SICCode2	VARCHAR(170)	NULL	SIC Codes 2
SICCode3	VARCHAR(170)	NULL	SIC Codes 3
SICCode4	VARCHAR(170)	NULL	SIC Codes 4
NumGenPartners	INT	NOT NULL	Number of general partners
NumLimPartners	INT	NOT NULL	Number of limited partners
URI	VARCHAR(47)	NOT NULL	URI
pn_CONDate	DATE	NULL DEFAULT NULL	Previous change of name date (occurs max 10)
pn_CompanyName	VARCHAR(160)	NULL DEFAULT NULL	Previous company name

FIGURE I.2: Phase 1 Company Data Dictionary

I.1.3 Phase 1 National Statistics Postcode Lookup (NSPL) Data Dictionary

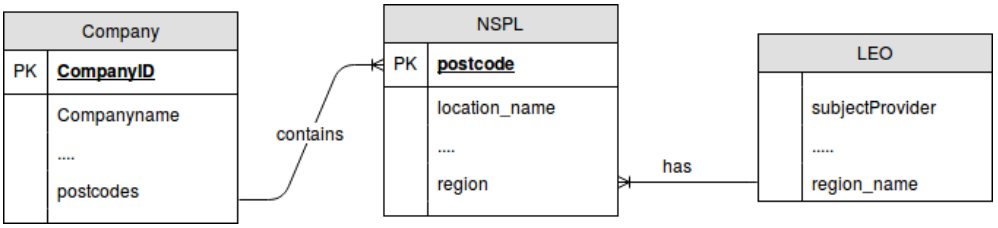


FIGURE I.3: Phase 1 National Statistics Postcode Lookup (NSPL) Data Dictionary