

**A DATA ANALYTIC MODULE TO EXTEND THE
GRAFANA ANALYTIC FUNCTION**

BY

HAR PUI SEE

A REPORT

SUBMITTED TO

UNIVERSITI TUNKU ABDUL RAHMAN

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

BACHELOR OF COMPUTER SCIENCE(HONS)

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

(KAMPAR CAMPUS)

JUNE 2020

REPORT STATUS DECLARATION FORM

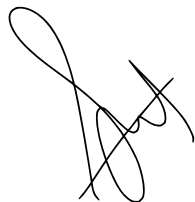
Title: A Data Analytic Module To Extend Grafana Analytic Function

Academic Session: June 2020

I HAR PUI SEE
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

- 1.The dissertation is a property of the Library.
- 2.The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

Address:

20, Jalan Perak 13,
Taman Bandar Baru Selatan,
31900, Kampar, Perak.

Dr. Ooi Boon Yaik

Supervisor's name

Date: 10 September 2020

Date: 10 September 2020

DECLARATION OF ORIGINALITY

I declare that this report entitled “**A DATA ANALYTIC MODULE TO EXTEND THE GRAFANA ANALYTIC FUNCTION**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : _____

Name : _____ HAR PUI SEE

Date : _____ 10 September 2020

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my project supervisor, Dr Ooi Boon Yaik who has taught and guide me throughout the whole development of my final year project. His insight and teaching method played a great influence on guiding me to solve problems.

ABSTRACT

With the staggering growth of IoT data, the IoT industrial has been transformed into big data industrial. There are wide variety of Time Series Databases(TSDB) has been developed to handle the massive amount of time series data which are being optimized in big data fields in terms of data monitoring and analytics. The Grafana is a powerful open sourced analytic and visualization software that associated with vast number of different databases. The data analytic process is important to enable users to gain useful insight from a huge volume of raw data. While there are vast number of different TSDB and RDMS available in the market that possessed different pros and cons. In this project, we benchmarked the database performance between the InfluxDB and the MySQL with Grafana by comparing the data insertion and the query performance of both databases. The benchmarked results will enable the users gain a deeper insight about the performance of InfluxDB and MySQL and thus easier for them to choose a suitable database for their project development used among wide variety of databases available in the market. Next, we will address the limitation of the Grafana as it cannot perform custom query and cannot visualize the data from the custom query. In order to enhance the analytic performance of the Grafana, we will deliver a programme to sit between the visualization tool Grafana and the data sources that function to perform custom query functions from multiple databases and thus deliver to Grafana for further data analytic function work.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT DECLARATION STATUS.....	ii
DECLARATION OF ORIGINALITY.....	iii
ACKNOWLEDGMENTS	iv
ABSTRACT.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
LIST OF SOURCE CODE	x
LIST OF ABBREVIATIONS.....	xi
CHAPTER 1 INTRODUCTION.....	1
1.1 Project Background.....	1
1.2 Problem Statement.....	3
1.3 Project Objectives	4
1.4 Project Scope.....	5
1.5 Impact, Significance and Contribution.....	6
CHAPTER 2 LITERATURE REVIEW.....	7
2.1 InfluxDB.....	7
2.2 MySQL.....	7
2.3 Comparison Between InfluxDB and MySQL.....	8
2.4 Benchmarking Graphite vs InfluxDB for Time Series Data, Metric and Management.....	9
CHAPTER 3 SYSTEM DESIGN.....	13
3.1 Use Case Diagram.....	13
3.2 Use Case Description.....	14
3.3 Activity Diagram.....	16
3.4 Class Diagram.....	17

3.5 Sequence Diagram	17
3.9 Project Timeline.....	18
CHAPTER 4 SYSTEM IMPLEMENTATION.....	19
4.1 Methodologies.....	19
4.2 Tools to Use	22
4.3 Performing Custom Query Function Across Multiple Databases On Grafana.....	23
4.4 Visualizing Data From Custom Query Function On Grafana	31
4.5 Implementation Issues and Challenges	35
CHAPTER 5 EXPERIMENTAL RESULT.....	36
5.1 Develop Database Benchmark Framework Between SQL and NoSQL Databases.....	36
5.2 Generate Visualization Report For Weather Based Retail Sales Data.....	42
5.3 Execution Time Analysis of Extended Grafana Analytic Platform	46
CHAPTER 6 CONCLUSION.....	53
BIBILOGRAPHY	55
APPENDIX A PROJECT POSTER.....	57
APPENDIX B PLAGIARISM CHECK RESULT.....	58
APPENDIX C SUPERVISOR COMMENTS ON ORIGINALITY REPORT...	59
APPENDIX D CHECKLIST FOR FYP1 THESIS SUBMISSION.....	60

LIST OF TABLES

Table Number	Title	Page
Table 2-1	Comparison between InfluxDB and MySQL	8
Table 3-1	Perform custom query function use case description	14
Table 3-2	Visualize data from custom query use case description	15
Table 5-1	Performance comparison result between InfluxDB and MySQL for table CPU	41
Table 5-2	Performance comparison result between InfluxDB and MySQL for table Temperature	41
Table 5-3	Overall results of program performance between Python data visualization program and extended Grafana analytic platform	51

LIST OF FIGURES

Figure Number	Title	Page
Figure 1-1	System Architect of the project	5
Figure 2-1	Comparison of writing throughput between InfluxDB and Graphite	9
Figure 2-2	Comparison of on-disk storage between InfluxDB and Graphite	10
Figure 2-3	Comparison of query throughput between InfluxDB and Graphite	10
Figure 3-1	Use Case Diagram of Extended Grafana Analytic System	13
Figure 3-2	Activity Diagram of Extended Grafana Analytic System	16
Figure 3-3	Class Diagram of Extended Grafana Analytic System	17
Figure 3-4	Sequence Diagram of Extended Grafana Analytic System	17
Figure 3-5	Gantt chart of FYP 1 and FYP 2 Report	18
Figure 4-1	Prototype Based Methodology	19
Figure 4-2	Configure a new datasource to Grafana	23
Figure 4-3	Add a new panel on Grafana	24
Figure 4-4	Query Editor on Grafana	24
Figure 4-5	Query editor in “Toggle Edit Mode” on Grafana	25
Figure 4-6	Custom query request edit on Grafana	25
Figure 4-7	Merged Dataframe result from InfluxDB and MySQL database	30
Figure 4-8	Scatter plot visualization of merged dataframe on Grafana.	34
Figure 4-9	Result dataframe display to Grafana in JSON format	34
Figure 5-1	Insert data using Java Client for InfluxDB.	37
Figure 5-2	Insert data using JDBC for MySQL	37
Figure 5-3	Dataset of table CPU in InfluxDB	38
Figure 5-4	Dataset of table CPU in MySQL	38
Figure 5-5	Query result in InfluxDB	40
Figure 5-6	Query result in MySQL	40

Figure 5-7	Retail sales data stored in MySQL database	42
Figure 5-8	Whether data stored in InfluxDB database	43
Figure 5-9	Scatter plot visualization between Temperature and Sales	44
Figure 5-10	Dual axis visualization of Temperature and Sales	44
Figure 5-11	Histogram visualization of correlation value between Temperature and Sales	45
Figure 5-12	Program execution time is measured between the interval of Influx query input follow by data display to Grafana	47
Figure 5-13	Processing custom query to display Daily Temperature panel on Grafana	47
Figure 5-14	Program execution time is measured between the interval of MySQL query input follow by data display to Grafana	48
Figure 5-15	Processing custom query to display Daily Temperature panel on Grafana.	48
Figure 5-16	Influx data visualization report generated using python matplotlib module	49
Figure 5-17	MySQL data visualization report generated using python matplotlib module.	50
Figure 5-18	Average program execution time of extended Grafana program	50
Figure 5-19	Average program execution time of Python data visualization program.	51

LIST OF SOURCE CODE

Source Code Number	Title	Page
Source Code 4-1	User program load custom query and get processed multi-database result data frame	32
Source Code 4-2	DuplexFetch program initialising the JSON custom query with calling function import from DataPlumbersTee	33
Source Code 4-3	DuplexFetch program processed the custom query request to get result dataframe	33
Source Code 4-4	DataPlumbersTee program initialised custom query database connection	35
Source Code 4-5	DataPlumbersTee program processing custom query and retrieve data across multi-databases.	35
Source Code 4-6	User program processing custom query and get tee object to call the function to display the custom query data to Grafana	37
Source Code 4-7	DuplexFetch program determine the tee variable based on the target database specified in custom query	38
Source Code 4-8	DataPlumbersTeeDebug program display single correlation value of merged dataframe display to Grafana	38
Source Code 4-9	DataPlumbersTeeDebug program display single column MySQL or InfluxDB dataframe to Grafana.	39
Source Code 4-10	DataPlumbersTeeDebug program display merged MySQL with InfluxDB dataframe to Grafana	39
Source Code 5-1	User python program measure the execution time of processing query and display Influx dataframe to Grafana.	46
Source Code 5-2	Python data visualization program process Influx query and plot Influx data visualization report.	49

LIST OF ABBREVIATIONS

IoT	Internet of Things
TSDB	Time Series Database
InfluxDB	Influx Database
OpenTSDB	Open Time Series Database
RAD	Rapid Application Development
RDMS	Relation Database Management System
SQL	Structured Query Language
API	Application Programming Interface

Chapter 1 Project Background

1.1 Project Background

In this data driven age, big data analytics is important to explore the massive volume of the IoT data generated from all connected sensor devices continuously to get better insights of the data for value creation. According to the growth of the application of time series database in big data fields and industrial IoT for data monitoring and analysis, number of the developed time series database is increased and optimized for time series data. However, a vast number of these time series database has possessed differences in terms of algorithm approach, strengths and weaknesses. Thus, an effective database benchmarking tool is crucial for comparing and evaluating the database for time series data.

Internet of Things(IoT) is described as a computer environment that enable massive number of devices connect over the internet physically and thus enabled the collecting and exchanging of data through wireless sensor and actuator networks. Moreover, the IoT paradigm is the integral of numerous technologies like identification and tracking technologies, distributed intelligence, enhanced a communication protocol which provides the communication solution among the objects. As a matter of fact, the staggering growth of IoT data has transformed the industrial data to the industrial big data. (Ranger, 2018)

Time series data is described as a sequence of data points being measured at a periodical interval and stored in a timely order. All the data generated by the IoT sensor devices is collected and stored over time. Time series data has become the new oil as it is importance for analysis purpose to recognise the major component in time series data like trend and novelties. Time series data can be used to understand the part as well as predict the future. However, the rapid growth of the data degrading the efficiency of storing and querying data and thus time series database is developed to optimise the time series data. (Kulkarni, 2018)

Chapter 1 Project Background

Time series database is recognised as storage optimized for time stamped or time series data. It is built for handling the metrics or events of a system and specialised to deal with high traffic data. Time series database is exploded in popularity due to its basic characteristics like high writing throughput, effective query performance and high scalability data storage to enable efficient high volume data processing compared to traditional database management system. Recently, numerous time series database has sprung up as InfluxDB, OpenTSDB, Prometheus and TimescaleDB. Large variety of these time series database has demonstrated different performance in the IoT and industrial development. (Zhou, 2019)

IoT database benchmark is described as database benchmark framework specifically designed for time series database. The benchmark framework is used to evaluate the performance among the wide variety of time series database of desired based on a set of well defined test. Normally, database benchmark is designed to perform main test scenarios like insertion of high volume data and read the data inserted as these basic tests are widespread in many real time systems. Moreover, numerous important features being monitored and measured by the benchmark framework is the writing throughput, reading speed and the size of the data storage is required. An effective database benchmark is important use in different fields especially IoT, industrial and financial fields whenever database need to be chosen for web or application, IoT and Industrial application development.

Grafana is recognised as an open source interactive analytic and monitoring tools that handling the massive amount of data metrics of a specific database follow by building the dashboards and graph for effective data monitoring use. It supports multi-platform and compatible with vast number of data sources and some data sources stores with time series data like InfluxDB, MySQL, Graphite, OpenTSDB and so on. Plus, the rich features of the Grafana like data visualize, dynamic, dashboards, alerting and extensibility of plug-in make the most of the Grafana become a leading options of analytic and monitoring tools to be used among software developers or to be used in a broad range of industries. (Pastil, 2020)

1.2 Problem Statements

- **Incomprehensive database benchmarking between time series database and relational database**

There are vast numbers of database benchmarks available for measuring the performance of different Time Series Database from a wide range of different aspects. However, there are no efficient database benchmark available specifically for measuring the performance between Time Series Database and Relational Database. Instead, there are numerous general comparisons of system properties between relational database and time series database available in the public.

- **Grafana cannot perform customize query function**

Grafana is designed to support both traditional relational database and time series database likes MySQL and InfluxDB. The performance of the Grafana visualization, monitoring and analytic functions are query to the specific database that connected to the Grafana. Thus, only specific functions are performed by the respective database and thus results return to Grafana for data visualization and monitoring.

- **Grafana cannot visualize customized query function as it is unable to perform customize query function**

Due to the unavailability of custom query function of Grafana, users are unable to gain their certain desired insights from the datasets of the database connected to the Grafana as the functionalities of the Grafana is solely depend on the functions of the specific databases used such as InfluxDB or MySQL. Thus, users are unable to visualize some certain customized functions since the Grafana cannot perform the custom query function.

1.3 Project Objectives

- **To evaluate the data insertion and querying performance of Relational Database and Time Series Database**

The objectives of this project is to develop an efficient database benchmark framework that specifically evaluate the data insertion and querying performance of the database that designed with different primary model as MySQL that optimized for storing standard SQL numeric datatypes and InfluxDB, a NoSQL database that optimize for storing the time series metrics, events and real time analytics. Thus, a detailed comparison between MySQL and InfluxDB in term of data insertion and data reading performance are able to be identified effectively.

- **To develop a program that extend the Grafana functionality in data analytic platform**

The second objectives of the project is to deliver a program to address the problems and limitations of the Grafana with InfluxDB and MySQL. The program is designed with the querying function across multiple databases as connected to InfluxDB and MySQL simultaneously. Plus, the program is also enable the users to perform query request from MySQL and InfluxDB separately and thus fetch the combined data results to Grafana for further data visualization and monitoring used. Hence, the Grafana users is able to extract their desired data insights from different databases like InfluxDB and MySQL efficiently.

1.4 Project Scopes

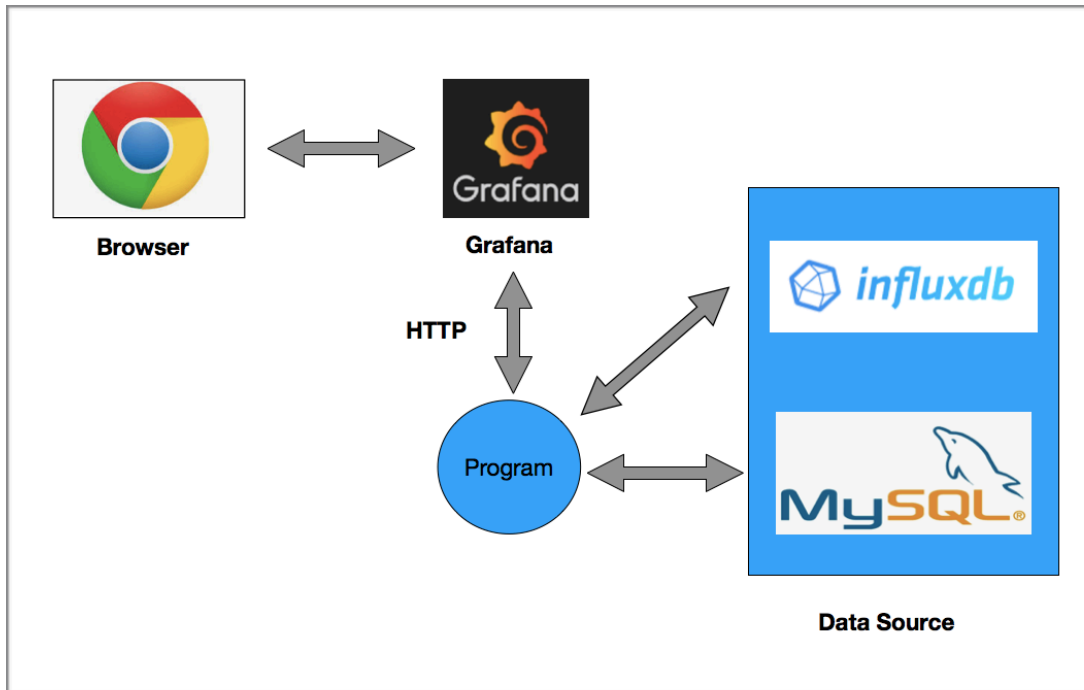


Figure 1-1 System Architect of the project

This project is designed to deliver an efficient and comprehensive database benchmark between the performance of both Relational Database and Time Series Database by using the MySQL and InfluxDB respectively. Instead of benchmarking the databases of same primary model, our project is delivered the benchmarking results of MySQL that designed to optimize for standard SQL numeric datatypes and InfluxDB that optimize for time series data, events and metrics. Moreover, the project scope will not focus on the IoT data visualization, database storage and analytic platform but concentrate solely on developing a program to support and enhance the Grafana, MySQL and InfluxDB in the aspect of data analytic platform by improving the extension functionalities of Grafana.

1.5 Impact, Significance and Contribution

In this project, an effective database benchmark work is being performed to evaluate the performance between the popular database that designed with different primary database model as the InfluxDB is designed for storing time series metrics and events while the MySQL database is designed for the storing standard SQL numeric data. Based on the InfluxDB and MySQL benchmark result, the reader are able to gain a more deeper insights towards the strength and weakness possessed by these two database in term of data insertion and querying performance and thus choose a more suitable database for their project development use more efficiently.

On the other hand, a data analytic module is delivered to improve the analytic functions of Grafana by enable users to perform custom query function across multiple databases like MySQL and InfluxDB databases. The program will redirect the custom query results to Grafana and thus enable users to visualize the data from custom query and to explore their desired insights from the data. In conclude, the program is delivered to benefit society, business or organization to solve their problems and support their business growth by exploring better insights on their business data for a more effective decision making.

Chapter 2 Literature Review

Engine/Tools On Premises Approach

2.1 InfluxDB

InfluxDB is defined as an open source time series database with high scalability optimized for heavy writing and query loads. For instance, user is able to utilize the influxDB to process data at a higher speed. Secondly, the excellent query performance of the influxDB is benefit user with efficient querying with high volume data. It is enabled user to use DevOps to perform data monitoring and metric function and thus increase the efficiency of influxDB. Moreover, the InfluxDB is an integral component of TICK stack. In terms of definition, TICK is referred as Telegraf, InfluxDB, Chronograf and Kapacitor. TICK is an open source projects designed to handle massive amount of time series data interacted with InfluxDB. (Isley and Savage, 2019)

2.2 MySQL

MySQL is an open sourced relational database system that emphasise on data consistency and compliance with a formal database schema in use of Structure Query Language (SQL). Moreover, the quick processing of data, reliability, flexibility and ease of use has made the most of the MySQL recognised as a leading database to be used among the software developers especially for web based application developments. In term of web development platform, MySQL acts as an important component of LAMP stack that consist of Linux operating system, Apache web server, MySQL and object oriented programming languages as either PHP, Python or Pearl. LAMP is recognised as an open source projects designed to be used for the development of websites and web applications. (Rouse, 2013)

2.3 Comparison Between The InfluxDB and MySQL

Name	InfluxDB	MySQL
License	Open Source	Open Source
Predefined Data Type	Numeric Data, Strings	Numeric Data, Strings
Primary Database Model	Time Series DBMS	Relational DBMS
Implementation On Language	JAVA	C and C++
Operating System	Linux OS OS X	Linux OS OS X Windows Solaris
Second Indexes	No	Yes
Supported Programming Language	.Net Java JavaScript Perl PHP Python Ruby Rust Scala	C C++ Java JavaScript Perl PHP Python Ruby
SQL Support	InfluxQL, SQL-like query language	Yes
Data Access APIs	HTTP, Telnet API	PHP API

Table 2-1 Comparison between MySQL and InfluxDB

2.4 Benchmarking Graphite vs InfluxDB for Time Series Data, Metric and Management

This previous database benchmark study was aimed to evaluate the performance metrics between the Graphite and InfluxDB for common time series workloads. InfluxDB is a time series database that built on SQL flavour while the Graphite is not supported by any SQL query language. Both the Graphite and InfluxDB was set to compare across multiple vectors like data ingestion performance, on-disk storage and also the mean query response time.

Besides, all the content and statistics used in this benchmarking scenario is from official benchmarking InfluxDB. In addition, the datasets is used to model the DevOps monitoring and metric use case. The write performance benchmark is carried by loading the 24 hours dataset with 4 worker threads. Furthermore, the query performance of time series database is being tested by aggregating value on random 1 hour, compiled into one minute interval and thus represent a single line in visual context for modelling DevOps use. Moreover, on-disk storage is tested after the dataset is loaded and thus comparing the written value of space consumed by the datasets in aid to get the performance of the TSDB. The lower the disk space used value, the greater the on-disk compression of the TSDB.

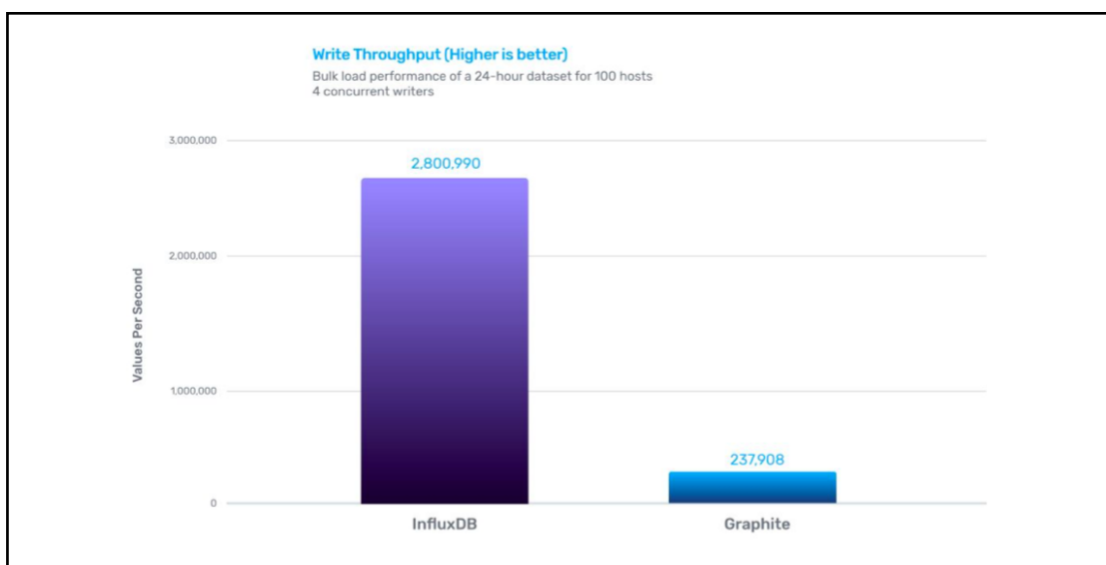


Figure 2-1 Comparison of writing throughput between InfluxDB and Graphite

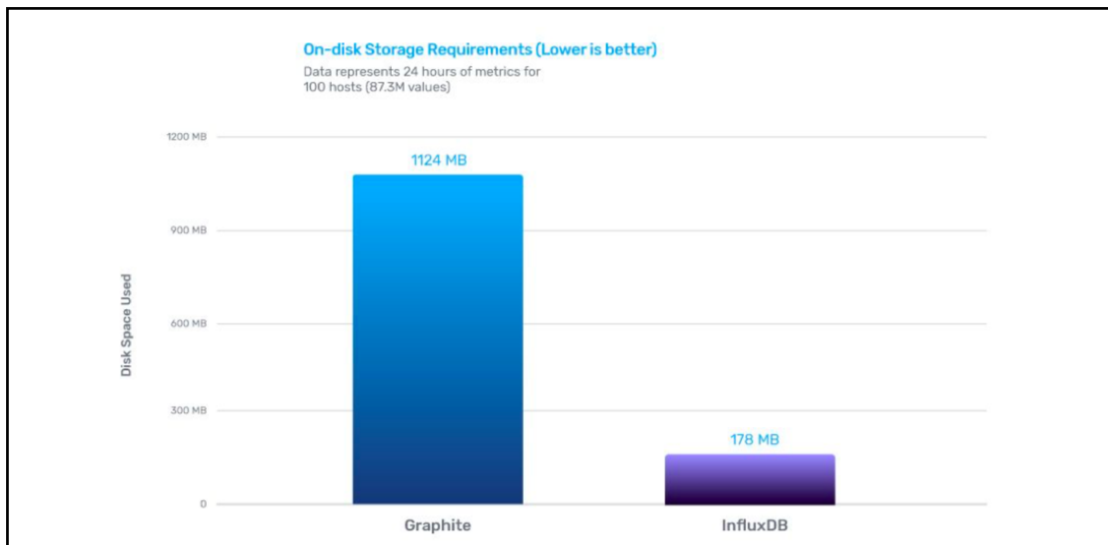


Figure 2-2 Comparison of on-disk storage between InfluxDB and Graphite

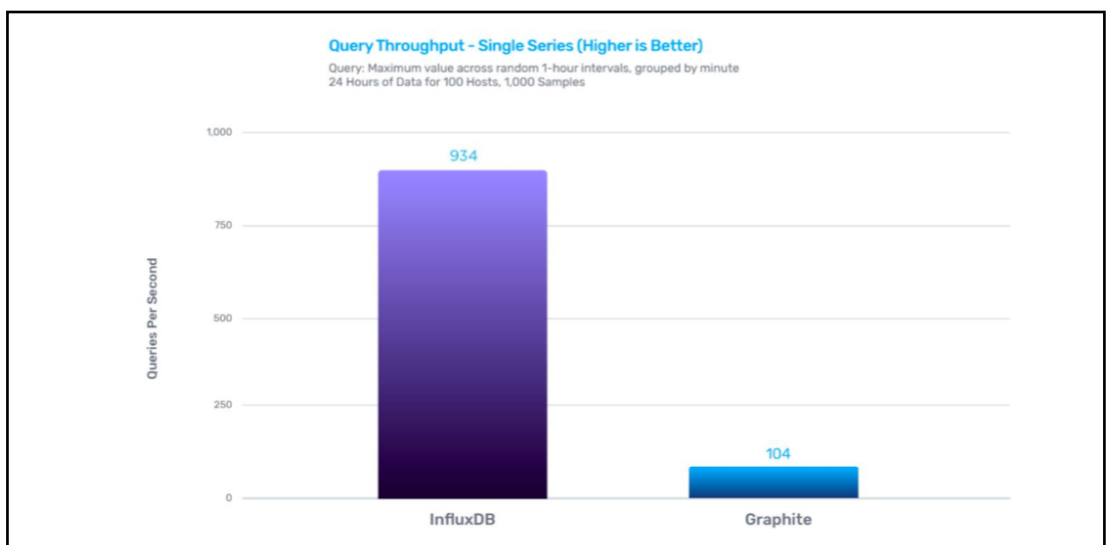


Figure 2-3 Comparison of query throughput between InfluxDB and Graphite

Based on the comparison results shown above, we can conclude that InfluxDB has possessed higher performance in data ingestion as approximately 12x speed faster than Graphite with the same dataset loaded. Plus, the InfluxDB has beaten Graphite by delivering 9x faster query performance. This is because the InfluxDB is built on SQL flavour, a structure query language that introduce higher speed of data retrieval. Thus, the InfluxDB users are able to use InfluxQL to retrieve huge amount of record from InfluxDB efficiently compared to Graphite that is not supported by SQL. Moreover, Graphite is required more space on disk compared to InfluxDB measured on same datasets loaded. Hence, the InfluxDB is proved to be a stronger and more efficient database compared to Graphite to provide better data writing and query performance with lower disk storage required. (Churilo,2019)

Strengths

- Applying DevOps monitoring tool during testing to get better insights of the database performance. Each dataset in this study is used to model a common DevOps to monitor and metric use case such as comparing write performance, query performance and on-disk storage.
- Provide basic performance test to evaluate data writing throughput, query throughput and on-disk storage between InfluxDB and Graphite. Through a series of basic database test, a detailed performance results can be shown to enable user differentiate between these two TSDBs.
- Benchmarking between the popular TSDBs available in the market, Both the InfluxDB and Graphite are recognised as the popular TSDBs widely used by organisations or developer according to their high scalability and efficiency. Thus, a performance benchmark between these TSDBs is important and enable the users to choose easier between them.

Weakness

- This TSDB benchmark work is not comprehensive. In this research, there are only two TSDBs are being benchmarked and three vector performance are being evaluated between these TSDBs. The work is considered not comprehensive to cover the more number of popular TSDBs at once. It is insufficient to solve the difficulties of developers to choose TSDBs among wide variety of TSDBs.
- Data accuracy of each TSDB is remained unchecked. As the data integrity is important to ensure the security of the data in the database and thus, the accuracy and completeness of the database must be ensured to avoid error contained in the particular TSDB. And this may lead to further negative impact to the developers like generate poor data analysis on inaccurate data.

Due to the numerous weaknesses of the benchmark approach has mentioned above, we can conclude that an effective IoT database benchmark work must be developed in order to evaluate the capability of both InfluxDB and MySQL through a series of tests like data insertion test followed by query performance test together with the data integrity check. Thus, an effective and accurate benchmark results is delivered with the detailed data insertion and querying performance between InfluxDB and MySQL in order to address the problems as incomprehensive database benchmark work between TSDB and RDMS available in the market. Moreover, a program will be developed to sit between the Grafana and the data source in order to solve the limitations of Grafana that unable to visualize the data from a custom query by designing the program to query data from multiple databases and thus pass the data to Grafana for further data visualization used.

Chapter 3 System Design

In this chapter, various kinds of system design diagram like use case diagram, activity diagram, class diagram and the communication diagram will be demonstrated to visualize the component of the Grafana data analytic system together with the depiction of how's user able to interact with the Grafana data analytic system through series of designed processes in order to achieve the system requirements that enabling user to perform custom query functions follow by visualize data output with their desired insights from custom query on Grafana for effective and efficient data interpretation.

3.1 Use Case Diagram

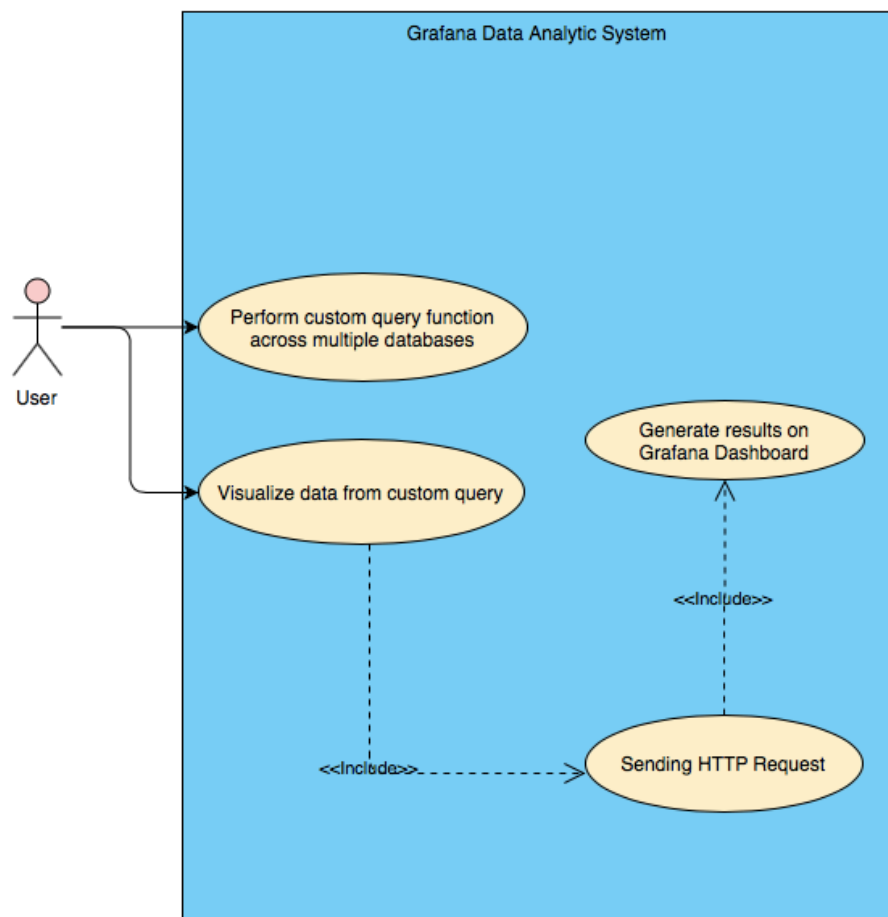


Figure 3-1 Use Case Diagram of Extended Grafana Analytic System

3.2 Use Case Description

Use Case ID	UC001	Version	1.0
Use Case	Perform custom query function across multiple databases.		
Aims	To enable users to perform custom query across multiple database as either retrieve data results from MySQL or InfluxDB or from both MySQL and InfluxDB.		
Actor	User		
Trigger	User send customized query to request for data among multiple databases.		
Precondition	The users must perform a valid custom query.		
Main Flow	<ol style="list-style-type: none"> 1. User designed a valid custom query that can be recognised by the targeted database. 2. User perform the designed query in order to collect the query data across multiple databases. 		
Alternative Flow	1.1 Database errors occurs as wrong query inserted.		
Author	Har Pui See		

Table 3-1 Perform custom query function use case description

Use Case ID	UC002	Version	1.0
Use Case	Visualize data from custom query		
Aims	To enable users to gain better insights on their desired data from custom query.		
Actor	User		
Trigger	User create a dashboard in Grafana and set the panel in view mode.		
Precondition	The user must enter the HTTP request correctly in order to generate result on the dashboard.		
Main Flow	<ol style="list-style-type: none"> 1. User create a dashboard in Grafana. 2. User must enter the correct HTTP request. 3. User must set the correct HTTP request parameters. 4. The system will update the results on the dashboard. 5. User can view the dashboard with desired output. 		
Alternative Flow	<ol style="list-style-type: none"> 2.1 Grafana dashboard panel show request error message. 3.1 Grafana dashboard panel show request error message. 4.1 System show error message. 		
Author	Har Pui See		

Table 3-2 Visualize data from custom query use case description

3.3 Activity Diagram

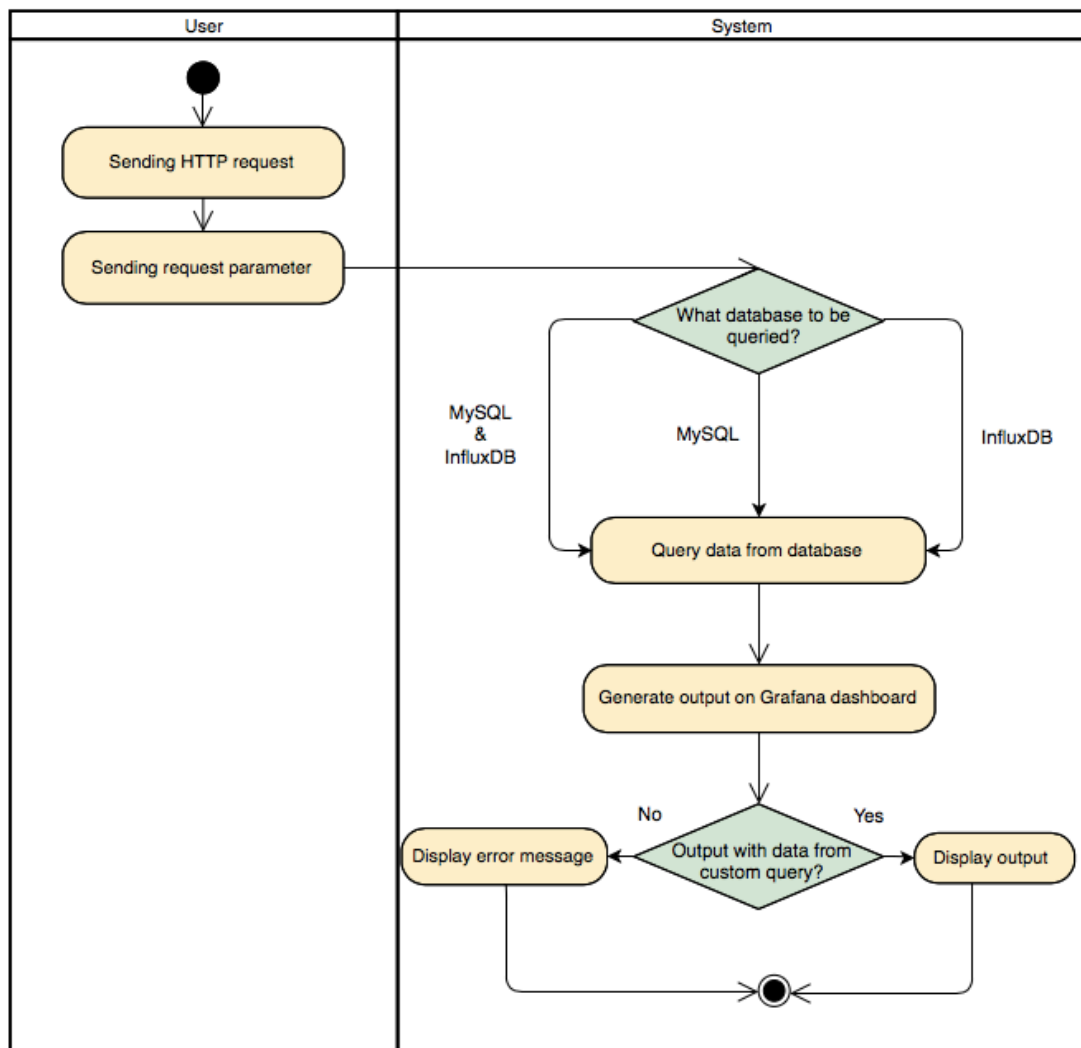


Figure 3-2 Activity Diagram of Extended Grafana Analytic System

3.4 Class Diagram

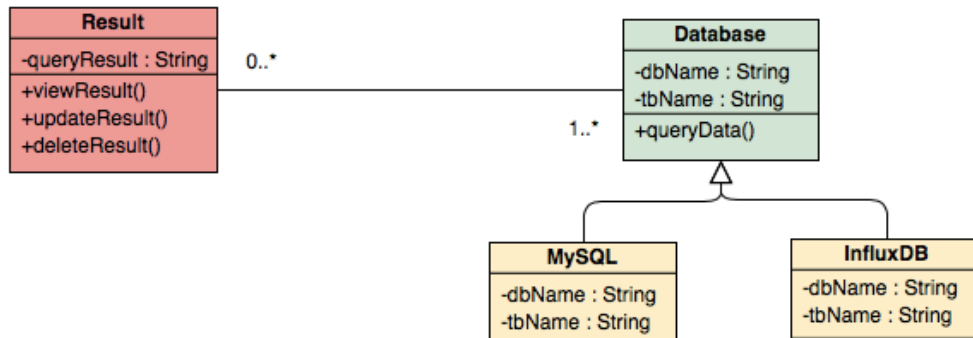


Figure 3-3 Class Diagram of Extended Grafana Analytic System

3.5 Sequence Diagram

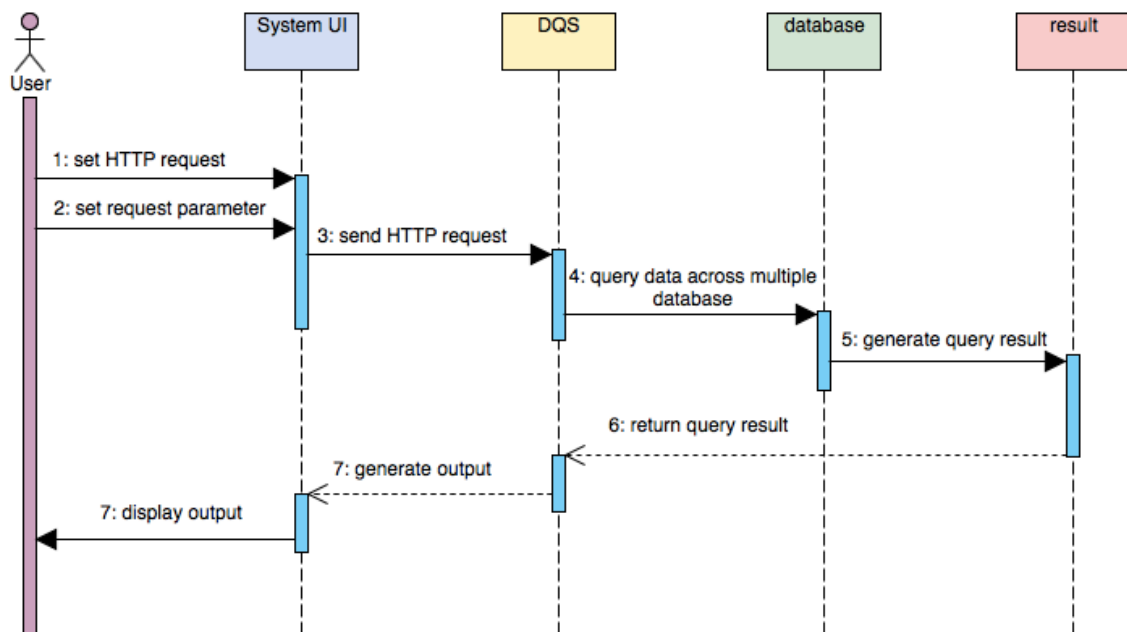


Figure 3-4 Sequence Diagram of Extended Grafana Analytic System

3.6 Project Timeline

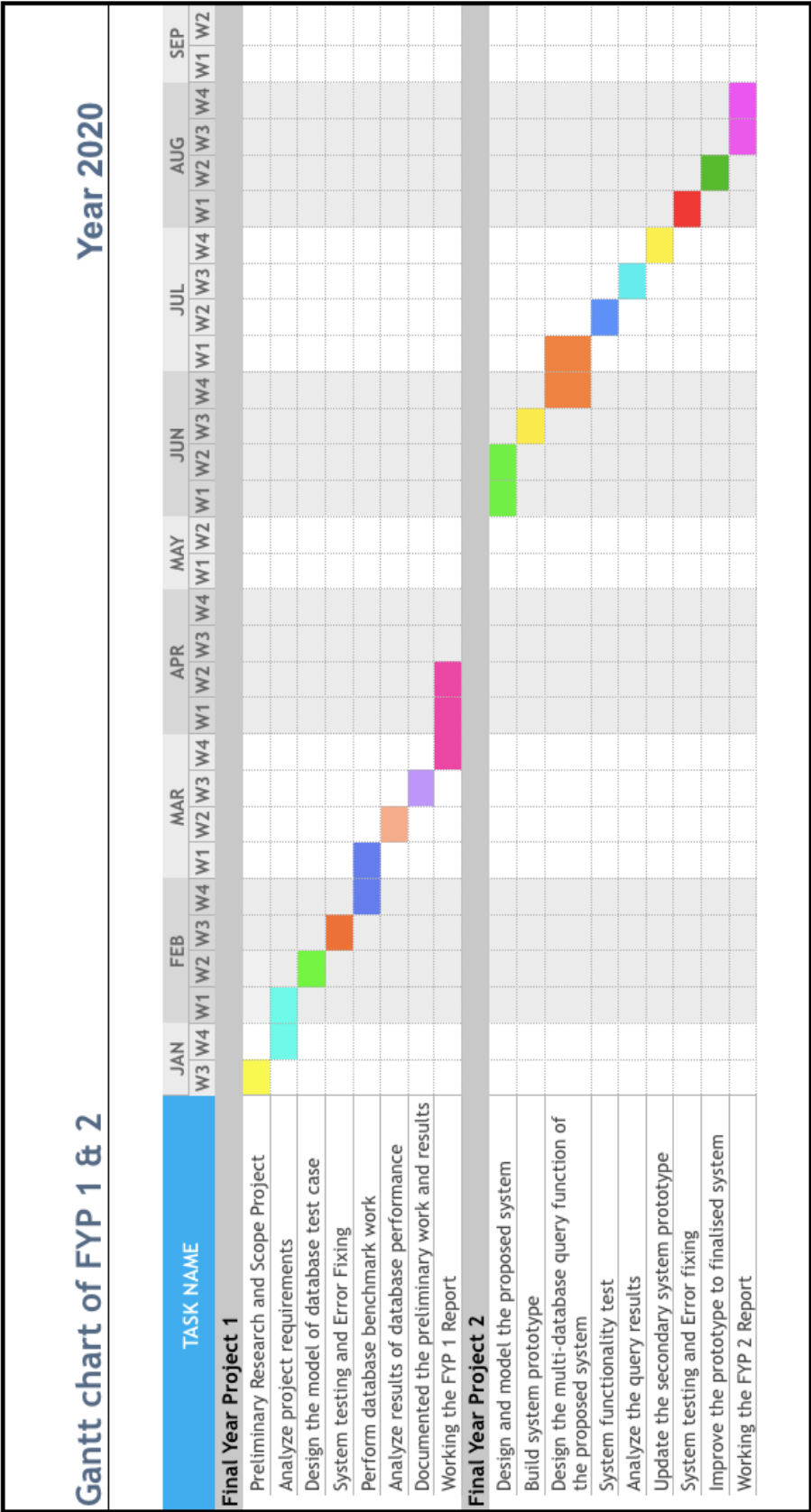


Figure 3-5 Gantt chart of FYP 1 and 2 Report

Chapter 4 System Implementation

4.1 Methodologies

The Rapid Application Development(RAD) is a kind of agile development that optimized for rapid prototyping and iterative delivery. In this project, the program is being developed based on emphasised on prototype in aid to deliver the products with shorten delivery times. (Powell-Morse,2016).

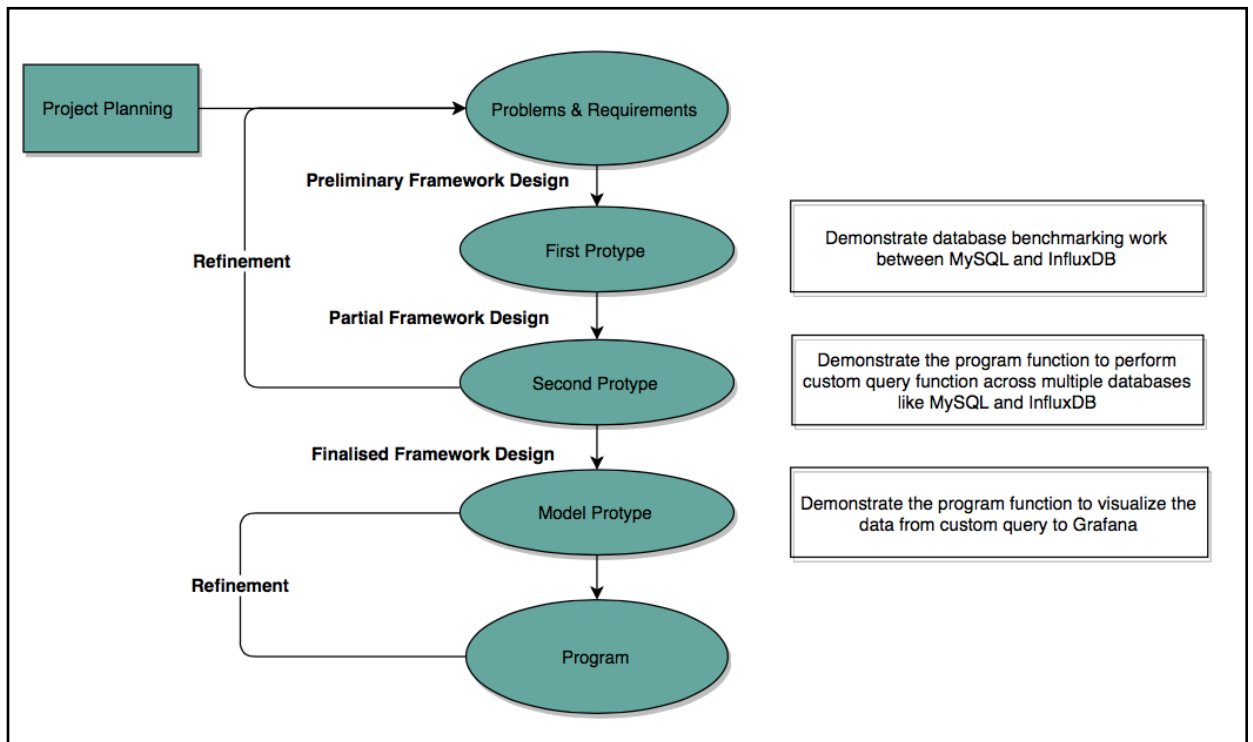


Figure 4-1 Prototype Based Methodology

In this prototyping based project, various number of prototype has been designed to be used throughout the whole development cycle. There are concurrent analysis, design and implementation steps keep repeated to generate few version of system prototype until a prototype that satisfied all the requirements is being developed. Different version of prototype has been developed throughout the whole development cycle.

In the beginning of the project, all the project requirements are being well defined as the problem statement and motivation of the project and follow by finalizing the project scope and main objective of the project in order to improve the analytic platform of the Grafana by addressing the limitation and weakness between Grafana , InfluxDB and MySQL databases.

The second phase of the project is to analyze all project related scopes and objectives in order to formalize and formulate the project requirements. Thus, literature review has been done by gaining more understanding of current existing research or approaches for addressing the incomprehensive benchmarking problem between MySQL and InfluxDB together with the Grafana analytic platform related problems and thus evaluate numerous available approaches and implement the system based on a suitable approach is chosen.

Next, the first prototype is built to demonstrate the benchmarking work between the MySQL and InfluxDB in order to possess the strengths and weakness of both databases in term of data insertion and query performance with the designed test cases. Thus, further test the performance of both databases with the Grafana visualization functionalities. Based on this first prototyping testing , if new user requirements introduced and thus refinement on the initial project problems and requirements is needed.

Moreover, second prototype is built to further implement on the first prototype to demonstrate and verify the interaction between MySQL, InfluxDB and Grafana in term of data query and data visualization performance and thus design a program to address the problems and limitations of data analytic functions of Grafana by enabling the user to perform the custom query functions to access and retrieve data from both MySQL and InfluxDB respectively. Follow by the further implement on the second prototype, the model prototype is then delivered to demonstrate and test the function of visualize the data from custom query on Grafana for effective data interpretation.

Furthermore, the prototype that performing partial functions of the programs will be delivered to the users for collecting feedbacks and further improve and refine the program until the developed system has completely satisfy all user requirements efficiently as the system prototype will be re-analyzed, re-designed and re-implemented in order to solve all project stated problems and thus to deliver a finalised program that has been delivered.

Lastly, the finalised prototype will be implemented by performing functionalities test in order to ensure the program to function properly and able to gain accurate results effectively without any unwanted errors occurs before the finalised program is being delivered to the users.

4.2 Tools to Use

In our project, we have use the following varies of tools to develop the data analytic module to extend the Grafana data analytic function as:

❖ Development Tools

- Linux CentOS - A free community enterprise operating system that support computer platform that function to support development and production servers.
- Linux Ubuntu - An open sourced operating system with Linux distribution based on Debian used for developing the database benchmark work.
- Visual Paradigm - An online free UML diagramming tools that support effective and efficient modelling of system design processes.

❖ Visualization Tools

- Grafana - A free data analytic platform with feature-rich dashboards and data sources plugin that support effective and efficient data visualization and monitoring functions.

❖ Data Sources

- MySQL - A relational database management system that mainly used for data warehousing and web database based on SQL and it is supported by Grafana.
- InfluxDB - A time series database well known for dealing with high volume and velocity of time series data and it is supported by Grafana.

❖ Programming Language

- Python - A general purpose programming language that designed with extensive support of libraries that provide rich set of functions that support wide range of different project development use.

❖ Remote Computing Tools

- MobaXterm - A free remote computing tools that support wide range of server connection like SSH, Telnet, FTP, VNC and so on.

4.3 Performing Custom Query Function Across Multiple Databases On Grafana

Grafana that provided feature-rich dashboards support and wide range of data sources plugin support has made the most of Grafana to become a powerful piece of data analytic tools that being widely employed among data scientists. In the event of generating a data visualization report on Grafana, a dashboard with one or a set of panels are being created to visualize the data being extracted by processing specific query functions with connected to a corresponding data source. However, before processing the query, it must have at least one datasource being added to Grafana for further data visualization used. Each panel has come with a query editor to enable user edit the query request to visualize specific data on Grafana panel.

Creation of data visualization report on Grafana can be generalized as following steps:

- Add a datasource to Grafana
- Create a new dashboard
- Add a new panel in dashboard for each visual representation of data
- Write at least one query to extract the specific data visualization display on panel

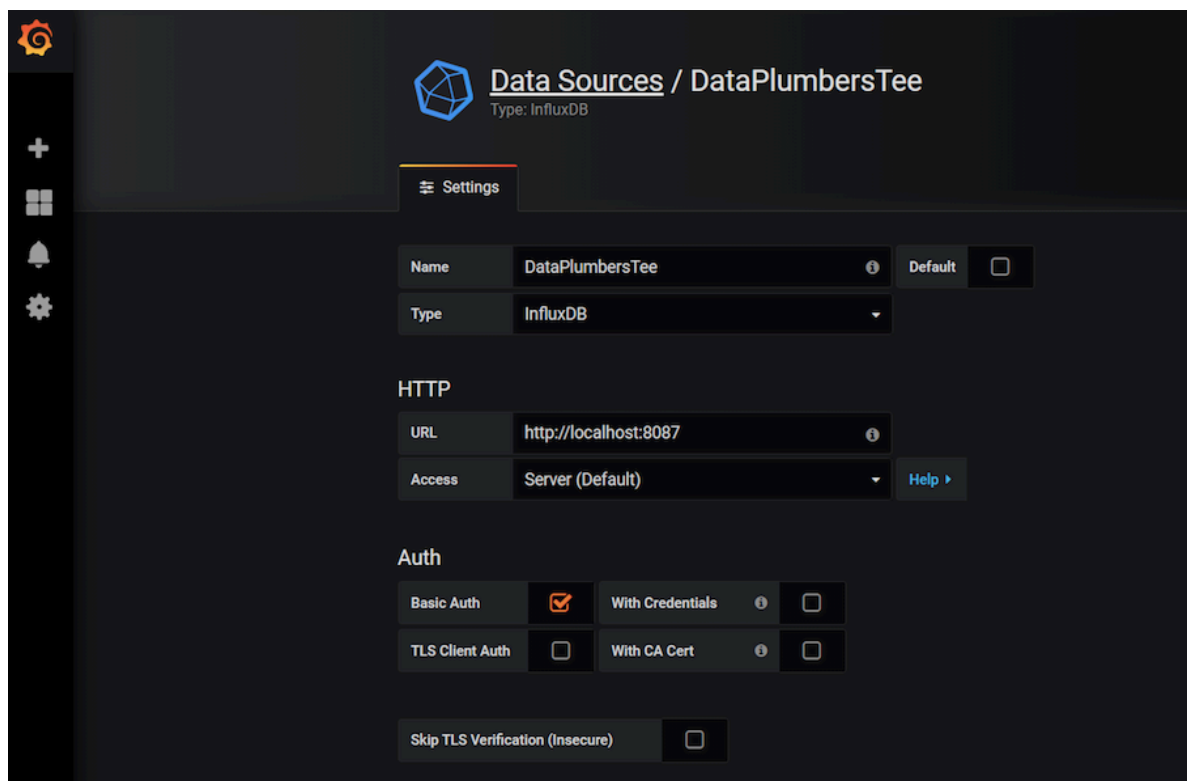


Figure 4-2 Configure a new datasource to Grafana

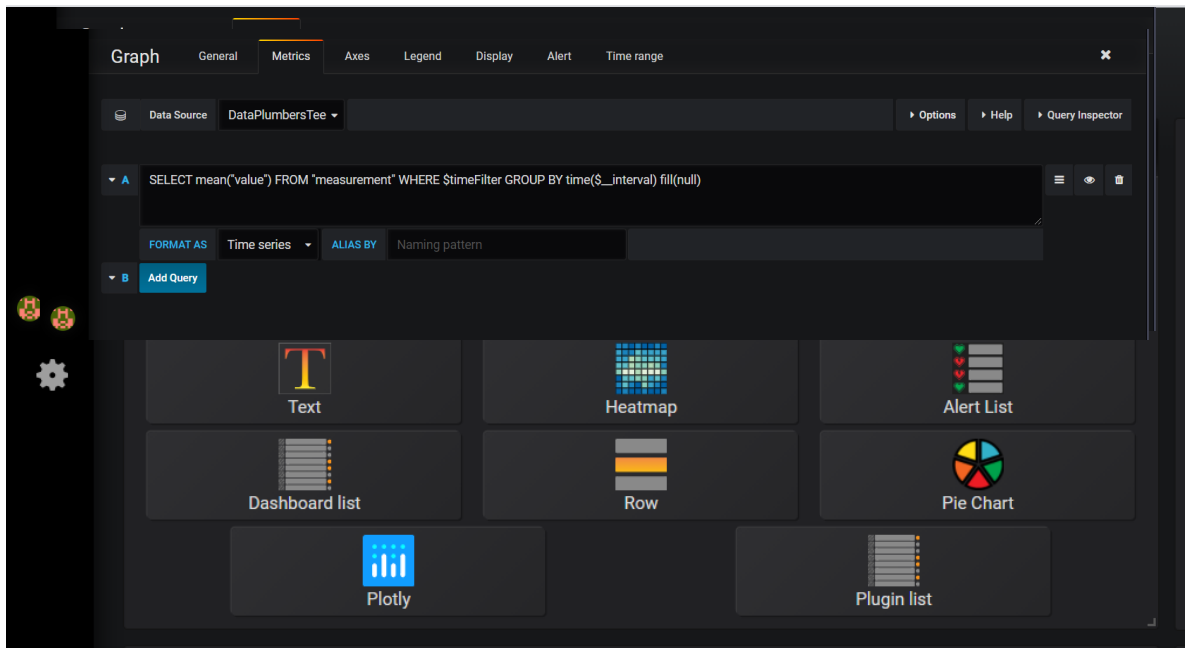


Figure 4-3 Add a new panel on Grafana

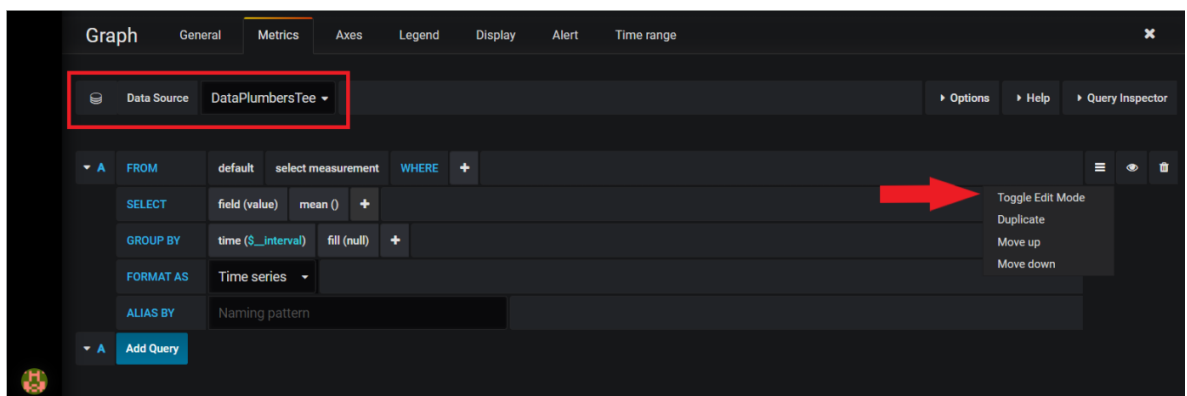


Figure 4-4 Query Editor on Grafana

Writing query is a key process to extract the specific data visualization display on the panel. Its basically a process to design a query to enable Grafana panel communicate with the specific connected datasource to fetch data for further visualization on Grafana used. According different selected datasources, the query editor will initially provide auto-completion, suggestion of metric names, or variable that subject to availability. Alternatively, users are able to further edit the auto completion query in a more flexible way by switching the initial query editor to a “Toggle Edit Mode”.

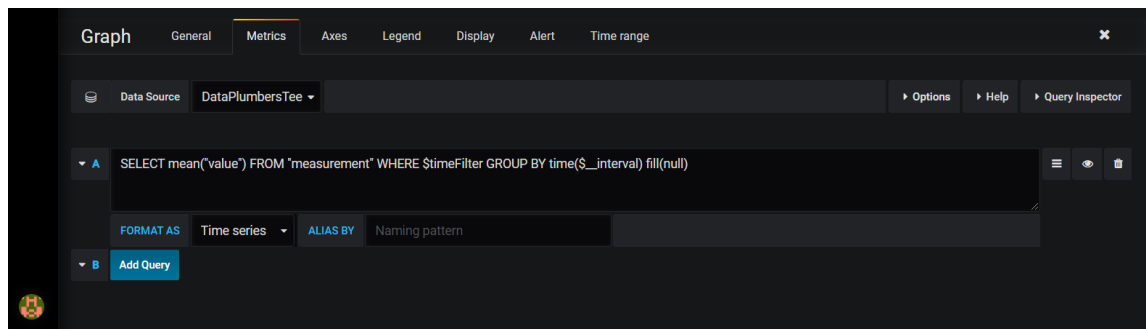


Figure 4-5 Query editor in “Toggle Edit Mode” on Grafana

In this “Toggle Edit Mode”, users are able to further edit the query by ignoring those initially provided auto-completion and suggestion on data query request which relatively more convenient for users wish to specify a custom query instead on depending on the data source matching available query pattern.

In order to address the limitations of Grafana analytic functions that unable user to perform custom query, we have developed a program seated between Grafana and multiple datasources like MySQL and InfluxDB with function aid to enable users performing custom query functions that valid to access and retrieve data across multiple databases at the same time. Moreover, users are granted with the extended functions to further process the retrieved data results like merging both correlated data results in one single output or return single correlation value between different data results collected across multiple databases as visualization results can be depend on the custom query request input by users in the Grafana query editor.

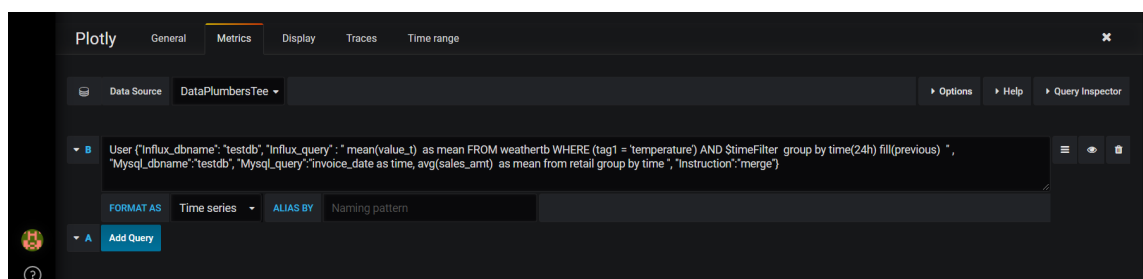


Figure 4-6 Custom query request edit on Grafana

Based on the figure above, it shows a panel editing work with a custom query request that connected to an initial configured datasource named DataPlumbersTee. In order to process the custom query, a module named DataPlumbersTee.jar will be

executed using the same port number as the DataPlumbersTee datasource for the redirecting of the input custom query from Grafana.

As from the custom query design, “User” is the name of the python program that design to be executed to call all specific functions that mainly import from DuplexFetch python program correspondingly to process the requested custom query. Next, the custom query is designed in JSON format for ease and efficient data parsing and translation between browser and server. Moreover, the custom query in JSON format is designed to store both MySQL and InfluxDB database name and query respectively together with data processing instructions.

With the intention of delivering a clean, readable and manageable program, we have developed the program in modular fashion that possessed high reusability of code. The simple and clean design of the User program is a sample program to depict how Data Scientist can perform the custom query function efficiently and effectively by making use of our developed DataPlumbersTee and DuplexFetch program through execution of only a few program lines through python importing modules feature.

```
import sys
import json
from urllib2 import unquote
from DuplexFetch import DuplexFetch

#load incoming query in JSON
unquoted_query = unquote(sys.argv[1]).replace("+", " ")
res = json.loads(unquoted_query)

fetch = DuplexFetch(res)
tee = fetch.get_tee()

#get query result dataframe
output_df = fetch.get_resultFrame()

#display result dataframe to grafana
tee.ps_display_dataframe_to_grafana(output_df)
```

Source Code 4-1 User program load custom query in JSON and get processed multi-database result data frame

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# =====
# Created By : Pui See
# Created Date: Mon August 3
# =====

import sys
import json
import pandas as pd
from urllib2 import unquote
from DataPlumbersTee import DataPlumbersTee

class DuplexFetch:

    def __init__(self, incoming_query):
        self.influx_dbname = incoming_query["Influx_dbname"]
        self.mysql_dbname = incoming_query["Mysql_dbname"]
        self.influx_tee = DataPlumbersTee("localhost",8086,"admin","admin", self.influx_dbname)
        self.mysql_tee = DataPlumbersTee("localhost",3306,"root","puisee123",self.mysql_dbname)
        self.influx_query = incoming_query["Influx_query"]
        self.mysql_query = incoming_query["Mysql_query"]
        self.instruction = incoming_query["Instruction"]
```

Source Code 4-2 DuplexFetch program initialising the JSON custom query with calling function import from DataPlumbersTee

```
#Get result dataframe display to grafana
def get_resultFrame(self):
    if str(self.instruction).strip() != "none":
        self.influx_df = self.influx_tee.ps_get_dataframe_from_influx(sys.argv[1])
        self.mysql_df = self.mysql_tee.ps_get_dataframe_from_mysql(sys.argv[1])

        if str(self.influx_query).strip() != "none" and str(self.mysql_query).strip() != "none":
            if(str(self.instruction).strip()) == "merge":
                self.mysql_df.index.name = 'time'
                self.influx_df.index.name = 'time'
                self.merged_df = self.influx_df.merge(self.mysql_df, on='time')
                self.output_df = self.merged_df
                return self.output_df

            if(str(self.instruction).strip()) == "corr":
                self.mysql_df.index.name = 'time'
                self.influx_df.index.name = 'time'
                self.merged_df = self.influx_df.merge(self.mysql_df, on='time')
                self.corr_value = self.merged_df["mean_x"].corr(self.merged_df["mean_y"])
                self.corr = {'col1': str(self.corr_value)}
                self.corr_df = pd.DataFrame(self.corr, index =[0])
                return self.corr_df

        elif str(self.instruction).strip() == "none":
            if str(self.mysql_query).strip() != "none" and str(self.influx_query).strip() == "none":
                self.mysql_df = self.mysql_tee.ps_get_dataframe_from_mysql(sys.argv[1])
                self.mysql_df.index.name = 'time'
                self.output_df = self.mysql_df
                return self.output_df

            elif str(self.mysql_query).strip() == "none" and str(self.influx_query).strip() != "none":
                self.influx_df = self.influx_tee.ps_get_dataframe_from_influx(sys.argv[1])
                self.influx_df.index.name = 'time'
                self.output_df = self.influx_df
                return self.output_df
```

Source Code 4-3 DuplexFetch program processed the custom query request to get result dataframe

Based on the source codes above, the DuplexFetch program is developed to initialised JSON custom query passed from User program once the related DuplexFetch module function is being called. From the DuplexFetch `__init__` method, all related query component like target InfluxDB, MySQL databases name, query and instruction will be initialised for further processing use. However, the `self.influx_tee`, and `self.mysql_tee` variable that require for processing multiple databases querying functions is being initialised from calling the function named `ps_get_dataframe_from_influx()` and `ps_get_dataframe_from_mysql()` respectively imported from DataPlumbersTee module.

Next, `get-resultFrame()` function is developed to recognised the custom query conditions as either to query across multiple databases, query from MySQL database or just query data from InfluxDB. As for the custom query request to get data from both InfluxDB and MySQL, the instruction query is also subject to recognise as either “merge” for merging multiple databases result in one or “corr” for return single correlation value between multiple databases result. In end of the function, only a single dataframe will be returned for further process to be displayed on Grafana. Furthermore, in order to execute the `get_resultFrame()` function both `self.influx_df` and `self.mysql_df` is initialised by calling functions import from DataPlumbersTee module which is intentionally designed to get the specific databases query result and return in dataframe format for further data processing or visualisation used.


```
class DataPlumbersTee:

    timezone = 'Asia/Kuala_Lumpur'

    def __init__(self, host, port, username, password, database):
        self.host = host
        self.port = port
        self.username = username
        self.password = password
        self.database = database
```

Source Code 4-4 DataPlumbersTee program initialised custom query database connection

```
def ps_get_dataframe_from_influx(self, incoming_query):
    self.client = DataFrameClient(str(self.host), self.port, str(self.username), str(self.password), str(self.database))
    #self.client = DataFrameClient(host='localhost', port=8086, username='admin', password='admin', database=str(self.database))
    #remove additional queries, we only process Influxdb query
    self.unquoted_query = unquote(incoming_query.strip()).replace("+", " ")
    self.query_list = json.loads(self.unquoted_query)
    self.query = "SELECT " + self.query_list["Influx_query"]
    self.results = self.client.query(self.query, chunked=False)
    self.client.close()
    #get tablename
    self.table = self.results.keys()[0]
    #get aggregation method name e.g. last / mean
    self.aggregation_method = self.results.values()[0].keys()[0]
    #Grab the dataframe from the dictlist of the query
    self.df = self.results[self.results.keys()[0]]
    return self.df

def ps_get_dataframe_from_mysql(self, incoming_query):
    #self.con=mysql.connector.connect(str(self.host),self.port,str(self.username),str(self.password), str(self.database))
    self.con=mysql.connector.connect(host=str(self.host),port=self.port,user=str(self.username),password=str(self.password), database=str(self.database))
    self.unquoted_query = unquote(incoming_query.strip()).replace("+", " ")
    self.query_list = json.loads(self.unquoted_query)
    self.query = "SELECT " + self.query_list["Mysql_query"]
    self.cur = self.con.cursor(dictionary=True)
    self.cur.execute(self.query)
    self.results = self.cur.fetchall()
    self.con.close()
    for r in self.results:
        r = r
    #get tablename , r = self.results stored in <dict> type
    self.table = r.keys()[0]
    #get aggregation method name
    self.aggregation_method = r.keys()[0]
    #get dataframe from MySQL query result
    self.df = pd.DataFrame(self.results)
    #formalise the DataFrame format
    self.df = self.df.set_index('time')
    self.df.index = self.df.index.tz_localize('UTC').tz_convert('UTC')
    self.df.index.name = ' '
    return self.df
```

Source Code 4-5 DataPlumbersTee program processing custom query and retrieve data across multi-databases

Firstly, the `DataPlumberTee __init__` method is designed to initialised all custom query related variable that required for further data querying or visualization used. On the other hand, both of the `ps_get_dataframe_from_influx` method and `ps_get_dataframe_from_mysql` method will process the custom query and load it in JSON follow by processing the specific database query to retrieved specific data results accordingly.

In the event of processing the custom query function that shown on the figure 4-12 with title Custom query request edit on Grafana, “User” program is executed to process the custom query that is designed to retrieve average temperature value from InfluxDB and average sales from MySQL databases together with the instruction “merge” to join both InfluxDB and MySQL query results together. Instead of directly process the dataframe results for visualization, the merged dataframe result is able to be print and displayed on the executed `DataPlumbersTee.jar` server connected fronted as below:

```
User
Merged Temperature and Sales Dataframe:
```

time	averageTemperature	averageSales
2011-01-04 00:00:00+00:00	3.0	19.057143
2011-01-05 00:00:00+00:00	6.0	31.517727
2011-02-01 00:00:00+00:00	3.0	18.888947
2011-02-17 00:00:00+00:00	7.0	15.775000
2011-02-22 00:00:00+00:00	4.0	24.301818
2011-03-01 00:00:00+00:00	4.0	16.881000
2011-03-28 00:00:00+00:00	8.0	15.197377
2011-04-01 00:00:00+00:00	13.0	21.339487
2011-04-15 00:00:00+00:00	11.0	23.463478
2011-05-01 00:00:00+00:00	13.0	19.368000
2011-06-01 00:00:00+00:00	13.0	21.407813
2011-06-22 00:00:00+00:00	14.0	31.745000
2011-06-27 00:00:00+00:00	24.0	17.757917
2011-07-01 00:00:00+00:00	5.0	21.592836
2011-08-01 00:00:00+00:00	19.0	24.230555
2011-09-01 00:00:00+00:00	14.0	44.630435
2011-09-22 00:00:00+00:00	16.0	15.964706
2011-09-29 00:00:00+00:00	18.0	21.140000
2011-10-02 00:00:00+00:00	19.0	21.679455
2011-10-18 00:00:00+00:00	10.0	44.612857
2011-11-01 00:00:00+00:00	12.0	25.177069
2011-12-01 00:00:00+00:00	9.0	21.017273
2011-12-09 00:00:00+00:00	6.0	26.155745

Figure 4-7 Merged Dataframe result from InfluxDB and MySQL database

4.4 Visualizing Data From Custom Query Function on Grafana

As a matter of fact, there is a strong relationship between both data querying and data visualization as query is basically a vital component in data analytic platform to communicate with data sources to get the user desired data results successfully and thus only able to generate and display the data visualization report on Grafana. Hence, the development of the program functions to visualize data from custom query is built based on the valid program function that able to preform custom query function and get the related accurate data results.

```
import sys
import json
from urllib2 import unquote
from DuplexFetch import DuplexFetch

#load incoming query in JSON
unquoted_query = unquote(sys.argv[1]).replace("+", " ")
res = json.loads(unquoted_query)

fetch = DuplexFetch(res)
tee = fetch.get_tee()

#get query result dataframe
output_df = fetch.getResultFrame()

#display result dataframe to grafana
tee.ps_display_dataframe_to_grafana(output_df)
```

Source Code 4-6 User program processing custom query and get tee object to call the function to display the custom query data to Grafana

From User program, the tee object can be obtained by calling get_tee function for further execution of ps_display_dataframe_to_grafana function import from DuplexFetch module.

```
#get_tee for either mysql_df / influx_df / merged_df
def get_tee(self):
    if str(self.instruction).strip() != "none":
        if str(self.influx_query).strip() != "none" and str(self.mysql_query).strip() != "none":
            self.tee = self.influx_tee
            return self.tee

        elif str(self.instruction).strip() == "none":
            if str(self.mysql_query).strip() != "none" and str(self.influx_query).strip() == "none":
                self.tee = self.mysql_tee
                return self.tee

            elif str(self.mysql_query).strip() == "none" and str(self.influx_query).strip() != "none":
                self.tee = self.influx_tee
                return self.tee
```

Source Code 4-7 DuplexFetch program determine the tee variable based on the target database specified in custom query

```
def ps_display_dataframe_to_grafana(self, out_dataframe, row_column1 = 0, row_column2 = 1, table = None):
    #display corr_df / mysql_df / influx_df with one index and one column value
    if len(out_dataframe.columns) == 1:
        if (table is None):
            table = self.table
        if (str(out_dataframe.index.name).strip() != "time"):
            out_dataframe.index.name = "index"

        json_values_list = []
        # display corr_df without time column as index
        if (str(out_dataframe.index.name).strip() != "time"):
            for index, row in out_dataframe.iterrows():
                json_value_point = []
                json_value_point.append(index)
                json_value_point.append(row[row_column1])
                json_values_list.append(json_value_point)

        json_columns_array = []
        json_columns_array.append(str(out_dataframe.index.name))
        json_columns_array.append(str(out_dataframe.columns[0]))

        json_series = {}
        json_series["name"] = table
        json_series["columns"] = json_columns_array
        json_series["values"] = json_values_list

        json_series_array = []
        json_series_array.append(json_series)

        json_statement = {}
        json_statement["series"] = json_series_array
        json_statement["statement_id"] = row_column1

        json_statement_array = []
        json_statement_array.append(json_statement)
        json_result = {}
        json_result["results"] = json_statement_array

        print (json.dumps(json_result).replace("'", '"').replace("NaN", "null"))
```

Source Code 4-8 DataPlumbersTeeDebug program display single correlation value of merged dataframe display to Grafana

```
# display mysql_df or influx_df with time column as index
elif (str(out_dataframe.index.name).strip() == "time"):
    for index, row in out_dataframe.iterrows():
        json_value_point = []
        json_value_point.append(pd.Timestamp(index, tz = self.timezone).value/1000000)
        json_value_point.append(row[row_column1])
        json_values_list.append(json_value_point)

    json_columns_array = []
    json_columns_array.append(str(out_dataframe.index.name))
    json_columns_array.append(str(out_dataframe.columns[0]))

    json_series = {}
    json_series["name"] = table
    json_series["columns"] = json_columns_array
    json_series["values"] = json_values_list

    json_series_array = []
    json_series_array.append(json_series)

    json_statement = {}
    json_statement["series"] = json_series_array
    json_statement["statement_id"] = row_column1

    json_statement_array = []
    json_statement_array.append(json_statement)
    json_result = {}
    json_result["results"] = json_statement_array

    print (json.dumps(json_result).replace("'", '').replace("NaN", "null"))
```

Source Code 4-9 DataPlumbersTeeDebug program display single column MySQL or InfluxDB dataframe to Grafana

```
else:
    if (table is None):
        table = self.table

    json_values_list = []

    #in single output_df (iterate its stored element)
    for index, row in out_dataframe.iterrows():
        json_value_point = []
        #get 1st column of df "always time column"
        json_value_point.append(pd.Timestamp(index, tz = self.timezone).value/1000000)
        #2nd column, temperature
        json_value_point.append(row[row_column1])
        #3rd column, sales
        json_value_point.append(row[row_column2])
        json_values_list.append(json_value_point)

    json_columns_array = []
    json_columns_array.append("time")
    json_columns_array.append("temperature")
    json_columns_array.append("sales")

    json_series = {}
    json_series["name"] = table
    json_series["columns"] = json_columns_array
    json_series["values"] = json_values_list

    json_series_array = []
    json_series_array.append(json_series)

    json_statement = {}
    json_statement["series"] = json_series_array
    json_statement["statement_id"] = row_column1
    #add
    json_statement["statement_id2"] = row_column2

    json_statement_array = []
    json_statement_array.append(json_statement)
    json_result = {}
    json_result["results"] = json_statement_array

    print (json.dumps(json_result).replace("'", '').replace("NaN", "null"))
```

Source Code 4-10 DataPlumbersTeeDebug program display merged MySQL with InfluxDB dataframe to Grafana

With the second highlights of our project that enabling users to visualize the data from custom query, our program is developed to visualize the result dataframe collected from performing the custom query functions regarding different conditions like either to display the single correlation value of merged dataframe, Influx dataframe, MySQL dataframe or merged dataframe of both InfluxDB and MySQL. In this specific visualization function event, the deliverable of the custom query function is act as a determinant.



Figure 4-8 Scatter plot visualization of merged dataframe on Grafana

```
User
{"results": [{"series": [{"values": [[1294099200000, 3.0, 19.057143143245153], [1294185600000, 6.0, 31.5177273533561], [1296518400000, 3.0, 18.88894734466285], [1296518400000, 7.0, 15.77500095367432], [1298332800000, 4.0, 24.30181814323772], [1298937600000, 4.0, 16.881000022888184], [1301270400000, 8.0, 15.197376900031918], [1301270400000, 13.0, 21.339487198071602], [1302825600000, 11.0, 23.463478046914805], [1304208000000, 13.0, 19.36799980044365], [1306886400000, 13.0, 21.40781255811], [1306886400000, 14.0, 31.7449999503194], [1309132800000, 24.0, 17.7579167286555], [1309470400000, 5.0, 21.592035767945246], [1312156800000, 19.0, 24.23055548138], [1312156800000, 14.0, 44.630434782608695], [1316649600000, 16.0, 15.9647050318643], [1317254400000, 10.0, 21.139999937503895], [1317513600000, 19.0, 21.67945457], [1318896000000, 10.0, 44.61285718282864], [1320105600000, 12.0, 25.1770689596706], [1322697600000, 9.0, 21.017272689125754], [1323388800000, 6.0, 26.1557447], [1323388800000, 10.0, 21.017272689125754], [1323388800000, 12.0, 25.1770689596706], [1323388800000, 14.0, 21.017272689125754], [1323388800000, 16.0, 21.017272689125754], [1323388800000, 18.0, 21.017272689125754], [1323388800000, 20.0, 21.017272689125754], [1323388800000, 22.0, 21.017272689125754], [1323388800000, 24.0, 21.017272689125754], [1323388800000, 26.0, 21.017272689125754], [1323388800000, 28.0, 21.017272689125754], [1323388800000, 30.0, 21.017272689125754], [1323388800000, 32.0, 21.017272689125754], [1323388800000, 34.0, 21.017272689125754], [1323388800000, 36.0, 21.017272689125754], [1323388800000, 38.0, 21.017272689125754], [1323388800000, 40.0, 21.017272689125754], [1323388800000, 42.0, 21.017272689125754], [1323388800000, 44.0, 21.017272689125754], [1323388800000, 46.0, 21.017272689125754], [1323388800000, 48.0, 21.017272689125754], [1323388800000, 50.0, 21.017272689125754], [1323388800000, 52.0, 21.017272689125754], [1323388800000, 54.0, 21.017272689125754], [1323388800000, 56.0, 21.017272689125754], [1323388800000, 58.0, 21.017272689125754], [1323388800000, 60.0, 21.017272689125754], [1323388800000, 62.0, 21.017272689125754], [1323388800000, 64.0, 21.017272689125754], [1323388800000, 66.0, 21.017272689125754], [1323388800000, 68.0, 21.017272689125754], [1323388800000, 70.0, 21.017272689125754], [1323388800000, 72.0, 21.017272689125754], [1323388800000, 74.0, 21.017272689125754], [1323388800000, 76.0, 21.017272689125754], [1323388800000, 78.0, 21.017272689125754], [1323388800000, 80.0, 21.017272689125754], [1323388800000, 82.0, 21.017272689125754], [1323388800000, 84.0, 21.017272689125754], [1323388800000, 86.0, 21.017272689125754], [1323388800000, 88.0, 21.017272689125754], [1323388800000, 90.0, 21.017272689125754], [1323388800000, 92.0, 21.017272689125754], [1323388800000, 94.0, 21.017272689125754], [1323388800000, 96.0, 21.017272689125754], [1323388800000, 98.0, 21.017272689125754], [1323388800000, 100.0, 21.017272689125754], [1323388800000, 102.0, 21.017272689125754], [1323388800000, 104.0, 21.017272689125754], [1323388800000, 106.0, 21.017272689125754], [1323388800000, 108.0, 21.017272689125754], [1323388800000, 110.0, 21.017272689125754], [1323388800000, 112.0, 21.017272689125754], [1323388800000, 114.0, 21.017272689125754], [1323388800000, 116.0, 21.017272689125754], [1323388800000, 118.0, 21.017272689125754], [1323388800000, 120.0, 21.017272689125754], [1323388800000, 122.0, 21.017272689125754], [1323388800000, 124.0, 21.017272689125754], [1323388800000, 126.0, 21.017272689125754], [1323388800000, 128.0, 21.017272689125754], [1323388800000, 130.0, 21.017272689125754], [1323388800000, 132.0, 21.017272689125754], [1323388800000, 134.0, 21.017272689125754], [1323388800000, 136.0, 21.017272689125754], [1323388800000, 138.0, 21.017272689125754], [1323388800000, 140.0, 21.017272689125754], [1323388800000, 142.0, 21.017272689125754], [1323388800000, 144.0, 21.017272689125754], [1323388800000, 146.0, 21.017272689125754], [1323388800000, 148.0, 21.017272689125754], [1323388800000, 150.0, 21.017272689125754], [1323388800000, 152.0, 21.017272689125754], [1323388800000, 154.0, 21.017272689125754], [1323388800000, 156.0, 21.017272689125754], [1323388800000, 158.0, 21.017272689125754], [1323388800000, 160.0, 21.017272689125754], [1323388800000, 162.0, 21.017272689125754], [1323388800000, 164.0, 21.017272689125754], [1323388800000, 166.0, 21.017272689125754], [1323388800000, 168.0, 21.017272689125754], [1323388800000, 170.0, 21.017272689125754], [1323388800000, 172.0, 21.017272689125754], [1323388800000, 174.0, 21.017272689125754], [1323388800000, 176.0, 21.017272689125754], [1323388800000, 178.0, 21.017272689125754], [1323388800000, 180.0, 21.017272689125754], [1323388800000, 182.0, 21.017272689125754], [1323388800000, 184.0, 21.017272689125754], [1323388800000, 186.0, 21.017272689125754], [1323388800000, 188.0, 21.017272689125754], [1323388800000, 190.0, 21.017272689125754], [1323388800000, 192.0, 21.017272689125754], [1323388800000, 194.0, 21.017272689125754], [1323388800000, 196.0, 21.017272689125754], [1323388800000, 198.0, 21.017272689125754], [1323388800000, 200.0, 21.017272689125754], [1323388800000, 202.0, 21.017272689125754], [1323388800000, 204.0, 21.017272689125754], [1323388800000, 206.0, 21.017272689125754], [1323388800000, 208.0, 21.017272689125754], [1323388800000, 210.0, 21.017272689125754], [1323388800000, 212.0, 21.017272689125754], [1323388800000, 214.0, 21.017272689125754], [1323388800000, 216.0, 21.017272689125754], [1323388800000, 218.0, 21.017272689125754], [1323388800000, 220.0, 21.017272689125754], [1323388800000, 222.0, 21.017272689125754], [1323388800000, 224.0, 21.017272689125754], [1323388800000, 226.0, 21.017272689125754], [1323388800000, 228.0, 21.017272689125754], [1323388800000, 230.0, 21.017272689125754], [1323388800000, 232.0, 21.017272689125754], [1323388800000, 234.0, 21.017272689125754], [1323388800000, 236.0, 21.017272689125754], [1323388800000, 238.0, 21.017272689125754], [1323388800000, 240.0, 21.017272689125754], [1323388800000, 242.0, 21.017272689125754], [1323388800000, 244.0, 21.017272689125754], [1323388800000, 246.0, 21.017272689125754], [1323388800000, 248.0, 21.017272689125754], [1323388800000, 250.0, 21.017272689125754], [1323388800000, 252.0, 21.017272689125754], [1323388800000, 254.0, 21.017272689125754], [1323388800000, 256.0, 21.017272689125754], [1323388800000, 258.0, 21.017272689125754], [1323388800000, 260.0, 21.017272689125754], [1323388800000, 262.0, 21.017272689125754], [1323388800000, 264.0, 21.017272689125754], [1323388800000, 266.0, 21.017272689125754], [1323388800000, 268.0, 21.017272689125754], [1323388800000, 270.0, 21.017272689125754], [1323388800000, 272.0, 21.017272689125754], [1323388800000, 274.0, 21.017272689125754], [1323388800000, 276.0, 21.017272689125754], [1323388800000, 278.0, 21.017272689125754], [1323388800000, 280.0, 21.017272689125754], [1323388800000, 282.0, 21.017272689125754], [1323388800000, 284.0, 21.017272689125754], [1323388800000, 286.0, 21.017272689125754], [1323388800000, 288.0, 21.017272689125754], [1323388800000, 290.0, 21.017272689125754], [1323388800000, 292.0, 21.017272689125754], [1323388800000, 294.0, 21.017272689125754], [1323388800000, 296.0, 21.017272689125754], [1323388800000, 298.0, 21.017272689125754], [1323388800000, 300.0, 21.017272689125754], [1323388800000, 302.0, 21.017272689125754], [1323388800000, 304.0, 21.017272689125754], [1323388800000, 306.0, 21.017272689125754], [1323388800000, 308.0, 21.017272689125754], [1323388800000, 310.0, 21.017272689125754], [1323388800000, 312.0, 21.017272689125754], [1323388800000, 314.0, 21.017272689125754], [1323388800000, 316.0, 21.017272689125754], [1323388800000, 318.0, 21.017272689125754], [1323388800000, 320.0, 21.017272689125754], [1323388800000, 322.0, 21.017272689125754], [1323388800000, 324.0, 21.017272689125754], [1323388800000, 326.0, 21.017272689125754], [1323388800000, 328.0, 21.017272689125754], [1323388800000, 330.0, 21.017272689125754], [1323388800000, 332.0, 21.017272689125754], [1323388800000, 334.0, 21.017272689125754], [1323388800000, 336.0, 21.017272689125754], [1323388800000, 338.0, 21.017272689125754], [1323388800000, 340.0, 21.017272689125754], [1323388800000, 342.0, 21.017272689125754], [1323388800000, 344.0, 21.017272689125754], [1323388800000, 346.0, 21.017272689125754], [1323388800000, 348.0, 21.017272689125754], [1323388800000, 350.0, 21.017272689125754], [1323388800000, 352.0, 21.017272689125754], [1323388800000, 354.0, 21.017272689125754], [1323388800000, 356.0, 21.017272689125754], [1323388800000, 358.0, 21.017272689125754], [1323388800000, 360.0, 21.017272689125754], [1323388800000, 362.0, 21.017272689125754], [1323388800000, 364.0, 21.017272689125754], [1323388800000, 366.0, 21.017272689125754], [1323388800000, 368.0, 21.017272689125754], [1323388800000, 370.0, 21.017272689125754], [1323388800000, 372.0, 21.017272689125754], [1323388800000, 374.0, 21.017272689125754], [1323388800000, 376.0, 21.017272689125754], [1323388800000, 378.0, 21.017272689125754], [1323388800000, 380.0, 21.017272689125754], [1323388800000, 382.0, 21.017272689125754], [1323388800000, 384.0, 21.017272689125754], [1323388800000, 386.0, 21.017272689125754], [1323388800000, 388.0, 21.017272689125754], [1323388800000, 390.0, 21.017272689125754], [1323388800000, 392.0, 21.017272689125754], [1323388800000, 394.0, 21.017272689125754], [1323388800000, 396.0, 21.017272689125754], [1323388800000, 398.0, 21.017272689125754], [1323388800000, 400.0, 21.017272689125754], [1323388800000, 402.0, 21.017272689125754], [1323388800000, 404.0, 21.017272689125754], [1323388800000, 406.0, 21.017272689125754], [1323388800000, 408.0, 21.017272689125754], [1323388800000, 410.0, 21.017272689125754], [1323388800000, 412.0, 21.017272689125754], [1323388800000, 414.0, 21.017272689125754], [1323388800000, 416.0, 21.017272689125754], [1323388800000, 418.0, 21.017272689125754], [1323388800000, 420.0, 21.017272689125754], [1323388800000, 422.0, 21.017272689125754], [1323388800000, 424.0, 21.017272689125754], [1323388800000, 426.0, 21.017272689125754], [1323388800000, 428.0, 21.017272689125754], [1323388800000, 430.0, 21.017272689125754], [1323388800000, 432.0, 21.017272689125754], [1323388800000, 434.0, 21.017272689125754], [1323388800000, 436.0, 21.017272689125754], [1323388800000, 438.0, 21.017272689125754], [1323388800000, 440.0, 21.017272689125754], [1323388800000, 442.0, 21.017272689125754], [1323388800000, 444.0, 21.017272689125754], [1323388800000, 446.0, 21.017272689125754], [1323388800000, 448.0, 21.017272689125754], [1323388800000, 450.0, 21.017272689125754], [1323388800000, 452.0, 21.017272689125754], [1323388800000, 454.0, 21.017272689125754], [1323388800000, 456.0, 21.017272689125754], [1323388800000, 458.0, 21.017272689125754], [1323388800000, 460.0, 21.017272689125754], [1323388800000, 462.0, 21.017272689125754], [1323388800000, 464.0, 21.017272689125754], [1323388800000, 466.0, 21.017272689125754], [1323388800000, 468.0, 21.017272689125754], [1323388800000, 470.0, 21.017272689125754], [1323388800000, 472.0, 21.017272689125754], [1323388800000, 474.0, 21.017272689125754], [1323388800000, 476.0, 21.017272689125754], [1323388800000, 478.0, 21.017272689125754], [1323388800000, 480.0, 21.017272689125754], [1323388800000, 482.0, 21.017272689125754], [1323388800000, 484.0, 21.017272689125754], [1323388800000, 486.0, 21.017272689125754], [1323388800000, 488.0, 21.017272689125754], [1323388800000, 490.0, 21.017272689125754], [1323388800000, 492.0, 21.017272689125754], [1323388800000, 494.0, 21.017272689125754], [1323388800000, 496.0, 21.017272689125754], [1323388800000, 498.0, 21.017272689125754], [1323388800000, 500.0, 21.017272689125754], [1323388800000, 502.0, 21.017272689125754], [1323388800000, 504.0, 21.017272689125754], [1323388800000, 506.0, 21.017272689125754], [1323388800000, 508.0, 21.017272689125754], [1323388800000, 510.0, 21.017272689125754], [1323388800000, 512.0, 21.017272689125754], [1323388800000, 514.0, 21.017272689125754], [1323388800000, 516.0, 21.017272689125754], [1323388800000, 518.0, 21.017272689125754], [1323388800000, 520.0, 21.017272689125754], [1323388800000, 522.0, 21.017272689125754], [1323388800000, 524.0, 21.017272689125754], [1323388800000, 526.0, 21.017272689125754], [13233888
```

4.5 Implementation Issues and Challenges

In order to deliver an effective database benchmark work between MySQL and InfluxDB together with develop a data analytic module to extend the analytic functionalities of Grafana, there are numerous issues and challenges has been possessed throughout the whole implementation of the project like

- **Difficulties in design the model of the benchmark workload between different datasources**

Due to the MySQL and InfluxDB possessing differentiation in database type, scalability and support different query language. Thus, in order to evaluate the database efficiently, a comprehensive data writing and querying test case must be performed by using specific query design that fit to both MySQL and InfluxDB respectively.

- **Benchmarking the performance of different datasources is time consumed.**

In order to gain a reliable and comparable benchmark results between MySQL and InfluxDB, the designed benchmark tests are needed to be run multiple times to minimize the impacts and side effects to the output results, Plus, the quality of the output results will be verified by calculate the average value of the collected measurements. Thus, the project time management must be handled well in order to deliver the benchmark work more efficiently.

- **Design program to perform custom query function and integrating with multiple datasources is challenging**

In the event of develop program to perform custom query functions across multiple databases, the query design and data standardising has become the pre-requisite for further program development to delivered effective and accurate analysis work. In fact, both MySQL and InfluxDB are compatible with different query pattern. Hence, the design of the custom query statement is a challenging work to ensure the query is effectively to fetch all related data from both MySQL and InfluxDB and thus follow by data processing work to standardize different format of dataframe collected from multiple datasource accordingly.

Chapter 5 Experimental Result

5.1 Develop Database Benchmark Framework between SQL and NoSQL Databases

In this chapter, we described the ideas and the work done to benchmark the performance of the NoSQL and SQL databases like InfluxDB and MySQL with the numerous designed test cases to evaluate the performance of both databases in term of data insertion and data querying respectively. Besides, the different test methods of the databases will be further discussed together with the performance metric used and the overall test results of the database benchmark in this section.

5.1.1 Data Insertion

By testing the data insertion of the databases, we have used Java programming language to build a program to access, store and retrieve data from InfluxDB and MySQL respectively. In order to ensure the effectiveness and accuracy of the database benchmark, both databases have been tested with same design model like table design and amount of data tested in the databases. Firstly, the InfluxDB and the MySQL has designed with a database that included 2 tables named as CPU and Temperature. The CPU table was designed to store 3 attributes like host, region and value while Temperature table is stored 2 attributes like machine and value. Moreover, each database will be tested with the same 10000 rows of data for table CPU and Temperature respectively. Next, the JAVA JDBC API is used to store and access the MySQL while the java client is used to stored and retrieved data from InfluxDB with the use Eclipse software act as a medium.

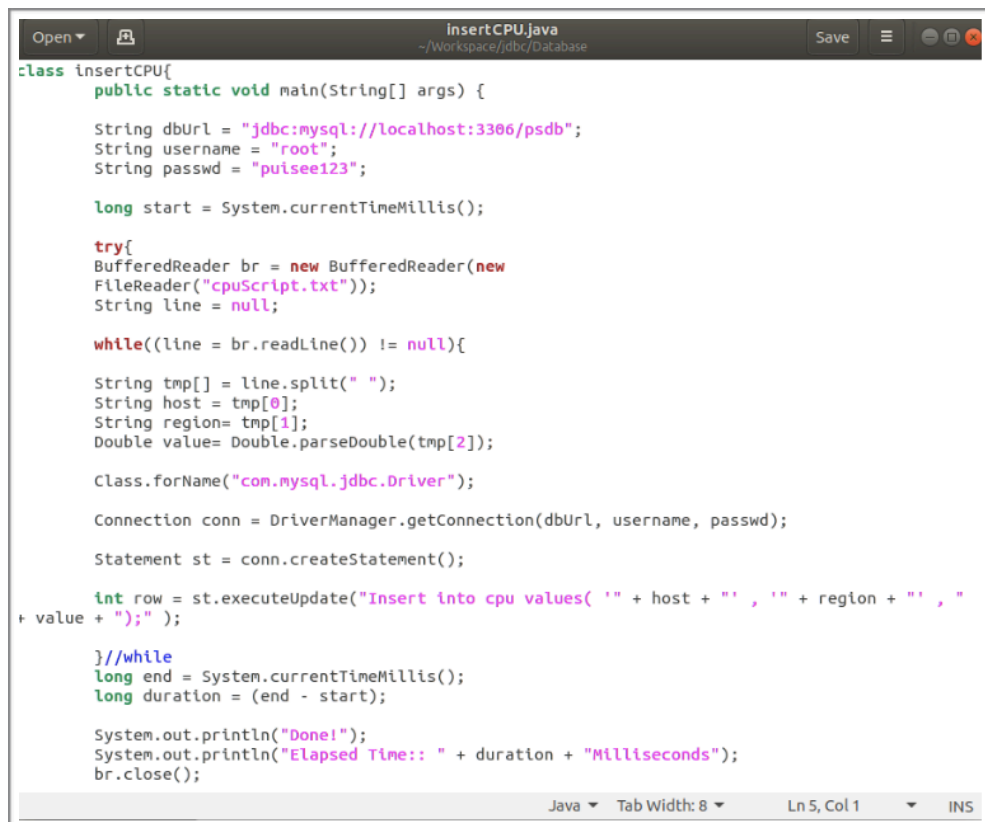


```

42
43 influxDB.write(Point.measurement("cpu")
44     .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
45     .addField("host", host)
46     .addField("region", region)
47     .addField("value", value)
48     .build());
49
50 influxDB.write(Point.measurement("temperature")
51     .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
52     .addField("machine", machine)
53     .addField("value", value)
54     .build());
55
56     } //while
57     long end = System.currentTimeMillis();
58     long duration = (end - start);
59
60     influxDB.write(Point.measurement("InsertTempDuration")
61         .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS)
62         .addField("ID", 8)
63         .addField("duration", duration)
64         .build());
65     System.out.println("Done!");
66     System.out.println("Elapsed Time:: " + duration + "Milliseconds");
67     br.close();
68 } catch (Exception e) {
69
70     System.out.println(e.getMessage());
71 }
72
73

```

Figure 5-1 Insert data using Java Client for InfluxDB.



```

class InsertCPU{
    public static void main(String[] args) {

        String dbUrl = "jdbc:mysql://localhost:3306/psdb";
        String username = "root";
        String passwd = "putsee123";

        long start = System.currentTimeMillis();

        try{
            BufferedReader br = new BufferedReader(new
            FileReader("cpuScript.txt"));
            String line = null;

            while((line = br.readLine()) != null){

                String tmp[] = line.split(" ");
                String host = tmp[0];
                String region= tmp[1];
                Double value= Double.parseDouble(tmp[2]);

                Class.forName("com.mysql.jdbc.Driver");

                Connection conn = DriverManager.getConnection(dbUrl, username, passwd);

                Statement st = conn.createStatement();

                int row = st.executeUpdate("Insert into cpu values( ' " + host + "' , ' " + region + "' , "
+ value + " );");

            } //while
            long end = System.currentTimeMillis();
            long duration = (end - start);

            System.out.println("Done!");
            System.out.println("Elapsed Time:: " + duration + "Milliseconds");
            br.close();
        }
    }
}

```

Figure 5-2 Insert data using JDBC for MySQL

```

root@pulsee: ~
File Edit View Search Terminal Help
1584815769091000000 serverA us_west 90
1584815769098000000 serverA us_west 17
1584815769108000000 serverA us_west 96
1584815769115000000 serverA us_west 49
1584815769122000000 serverA us_west 68
1584815769130000000 serverA us_west 9
1584815769137000000 serverA us_west 22
1584815769144000000 serverA us_west 12
1584815769150000000 serverA us_west 39
1584815769157000000 serverA us_west 21
1584815769164000000 serverA us_west 81
1584815769171000000 serverA us_west 75
1584815769178000000 serverA us_west 17
1584815769185000000 serverA us_west 76
1584815769193000000 serverA us_west 98
1584815769199000000 serverA us_west 58
1584815769206000000 serverA us_west 56
1584815769213000000 serverA us_west 19
1584815769220000000 serverA us_west 48
1584815769226000000 serverA us_west 5
1584815769233000000 serverA us_west 7
1584815769239000000 serverA us_west 22
1584815769246000000 serverA us_west 70
1584815769253000000 serverA us_west 23
1584815769260000000 serverA us_west 40
1584815769267000000 serverA us_west 50
1584815769274000000 serverA us_west 2
1584815769280000000 serverA us_west 13
1584815769287000000 serverA us_west 84
1584815769294000000 serverA us_west 97

```

Figure 5-3 Dataset of table CPU in InfluxDB

```

pulsee@pulsee: ~
File Edit View Search Terminal Help
| serverA | us_west | 47.00 |
| serverA | us_west | 58.00 |
| serverA | us_west | 78.00 |
| serverA | us_west | 93.00 |
| serverA | us_west | 85.00 |
| serverA | us_west | 96.00 |
| serverA | us_west | 28.00 |
| serverA | us_west | 44.00 |
| serverA | us_west | 34.00 |
| serverA | us_west | 78.00 |
| serverA | us_west | 0.00 |
| serverA | us_west | 68.00 |
| serverA | us_west | 67.00 |
| serverA | us_west | 86.00 |
| serverA | us_west | 54.00 |
| serverA | us_west | 77.00 |
| serverA | us_west | 43.00 |
| serverA | us_west | 31.00 |
| serverA | us_west | 94.00 |
| serverA | us_west | 37.00 |
| serverA | us_west | 50.00 |
| serverA | us_west | 18.00 |
+-----+-----+
10000 rows in set (0.04 sec)

mysql>

```

Figure 5-4 Dataset of table CPU in MySQL

5.1.2 Querying of Data

Numerous query types are designed to test the InfluxDB and MySQL that storing with same dataset. Both InfluxDB and MySQL also support multiple query types and thus we evaluated the query performance of InfluxDB and MySQL by using different query types. The query types we used as

- **Select All Query**

The select all query is designed to retrieved all available data from the specific tables with the syntax, i.e., `select * from <table>`.


- **Average Aggregation Query**

The average aggregation query is designed to retrieved the average value of an attributes from a specific tables with syntax, i.e., `select avg(column_name) from <table>`.

- **Minimum Aggregation Subquery**

The minimum aggregation subquery is designed to retrieve the minimum value returned from the nested inner query with syntax, i.e., `select min(column_name) from (select min(column_name) from <table>)`.

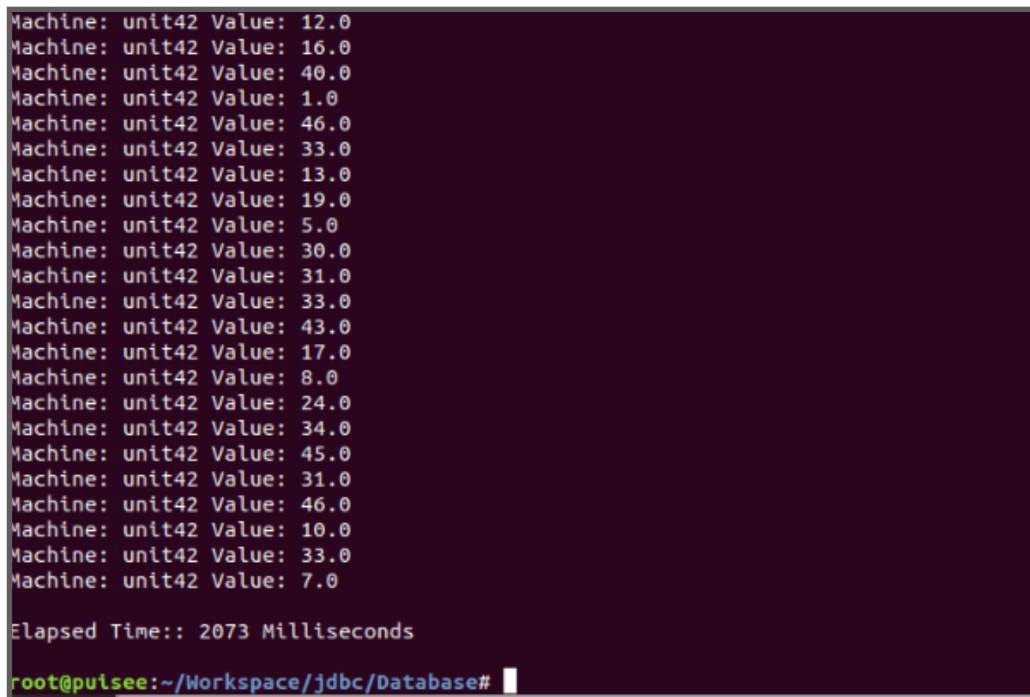
Chapter 5 Experimental Result



```
<terminated> Application01 [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (31 Mar 2020, 4:09:14 pm)
19.0], [2020-03-31T07:55:07.242Z, unit42, 48.0], [2020-03-31T07:55:07.272Z, unit42, 22.0], [2020-03-
31T07:55:07.296Z, unit42, 42.0], [2020-03-31T07:55:07.308Z, unit42, 6.0], [2020-03-31T07:55:07.321Z, unit42,
16.0], [2020-03-31T07:55:07.336Z, unit42, 11.0], [2020-03-31T07:55:07.362Z, unit42, 20.0], [2020-03-
31T07:55:07.375Z, unit42, 40.0], [2020-03-31T07:55:07.382Z, unit42, 0.0], [2020-03-31T07:55:07.393Z, unit42,
43.0], [2020-03-31T07:55:07.406Z, unit42, 16.0], [2020-03-31T07:55:07.415Z, unit42, 37.0], [2020-03-
31T07:55:07.428Z, unit42, 1.0], [2020-03-31T07:55:07.438Z, unit42, 41.0], [2020-03-31T07:55:07.445Z, unit42,
20.0], [2020-03-31T07:55:07.457Z, unit42, 40.0], [2020-03-31T07:55:07.471Z, unit42, 19.0], [2020-03-
31T07:55:07.476Z, unit42, 18.0], [2020-03-31T07:55:07.501Z, unit42, 5.0], [2020-03-31T07:55:07.523Z, unit42,
37.0], [2020-03-31T07:55:07.53Z, unit42, 15.0], [2020-03-31T07:55:07.537Z, unit42, 10.0], [2020-03-
31T07:55:07.543Z, unit42, 12.0], [2020-03-31T07:55:07.555Z, unit42, 16.0], [2020-03-31T07:55:07.571Z, unit42,
40.0], [2020-03-31T07:55:07.584Z, unit42, 1.0], [2020-03-31T07:55:07.605Z, unit42, 46.0], [2020-03-
31T07:55:07.619Z, unit42, 33.0], [2020-03-31T07:55:07.628Z, unit42, 13.0], [2020-03-31T07:55:07.643Z, unit42,
19.0], [2020-03-31T07:55:07.655Z, unit42, 5.0], [2020-03-31T07:55:07.664Z, unit42, 30.0], [2020-03-
31T07:55:07.678Z, unit42, 31.0], [2020-03-31T07:55:07.691Z, unit42, 33.0], [2020-03-31T07:55:07.699Z, unit42,
43.0], [2020-03-31T07:55:07.71Z, unit42, 17.0], [2020-03-31T07:55:07.722Z, unit42, 8.0], [2020-03-
31T07:55:08.093Z, unit42, 24.0], [2020-03-31T07:55:08.102Z, unit42, 34.0], [2020-03-31T07:55:08.111Z, unit42,
45.0], [2020-03-31T07:55:08.123Z, unit42, 31.0], [2020-03-31T07:55:08.13Z, unit42, 46.0], [2020-03-
31T07:55:08.136Z, unit42, 10.0], [2020-03-31T07:55:08.142Z, unit42, 33.0], [2020-03-31T07:55:08.148Z, unit42,
7.0]]], error=null]], error=null]

Elapsed Time: 852 Milliseconds
```

Figure 5-5 Query result in InfluxDB



```
Machine: unit42 Value: 12.0
Machine: unit42 Value: 16.0
Machine: unit42 Value: 40.0
Machine: unit42 Value: 1.0
Machine: unit42 Value: 46.0
Machine: unit42 Value: 33.0
Machine: unit42 Value: 13.0
Machine: unit42 Value: 19.0
Machine: unit42 Value: 5.0
Machine: unit42 Value: 30.0
Machine: unit42 Value: 31.0
Machine: unit42 Value: 33.0
Machine: unit42 Value: 43.0
Machine: unit42 Value: 17.0
Machine: unit42 Value: 8.0
Machine: unit42 Value: 24.0
Machine: unit42 Value: 34.0
Machine: unit42 Value: 45.0
Machine: unit42 Value: 31.0
Machine: unit42 Value: 46.0
Machine: unit42 Value: 10.0
Machine: unit42 Value: 33.0
Machine: unit42 Value: 7.0

Elapsed Time:: 2073 Milliseconds

root@pulsee:~/Workspace/jdbc/Database#
```

Figure 5-6 Query result in MySQL

5.1.3 Performance Metric

InfluxDB and MySQL database performance is being evaluated by comparing the performance of each database in response to each different operations type. We use average cost time metric as a measurements to benchmark the database performance by comparing the elapsed time measured in milliseconds of complete data insertion into the database or the query request sending to the database follow by a full results are returned. The performance of InfluxDB and MySQL will be measured in average cost time of 30 attempts of each operation tested in order to get a more accurate database benchmark result.

5.1.4 Overall Database Benchmark Result

In this database benchmark work, the result shows that the InfluxDB is outperform the MySQL by delivering greater data insertion and querying performance.

	Average Performance Measure in Milliseconds	
	InfluxDB	MySQL
Data Insertion	394515	826230
Select All Query	288	2328
Average Aggregation Query	31	139
Minimum Aggregation Subquery	43	101

Table 5-1 Performance result between InfluxDB and MySQL for table CPU

	Average Performance Measure in Milliseconds	
	InfluxDB	MySQL
Data Insertion	286993	631872
Select All Query	249	2141
Average Aggregation Query	38	155
Minimum Aggregation Subquery	54	81

Table 5-2 Performance result between InfluxDB and MySQL for table Temperature

5.2 Generate Visualization Report for Weather Based Retail Sales Data

With purpose of testing and ensuring the functions and performances of our developed program is able to extend Grafana analytic function by enable users to perform custom query functions and generate analytic report on Grafana to visualize the data from custom query, we have created a real world scenario test case as Weather Based Sales Forecasting to conduct the functionalities testing of the program.

The weather based retail sales forecasting test case is considered as an effective real world scenario test case to further interpret the functionality and possessed the importance of our program. In today business industry, there is an advanced increase in the employment of the hybrid databases which embrace the strengths of both SQL and NoSQL database like high scalability, efficiently data query performance enable data analyst to execute transactional (OLTP) and analytical (OLAP) workloads parallelly benefit for easy data interpretation and visualization. In our test case, same period of weather datasets and retail sales datasets will be stored in both InfluxDB and MySQL databases respectively and ready for further performing of custom query functions.

```
mysql> select * from retail;
```

invoice_no	product_code	product_name	ordered_qty	invoice_date	unit_price	sales_amt	customer_id	location
536365	85123A	WHITE HANGING HEART	6	2010-12-01 00:00:00	2.55	15.3	17850	United Kin
536365	71053	WHITE METAL LANTERN	6	2010-12-01 00:00:00	3.39	20.34	17850	United Kin
536365	84406B	CREAM CUPID HEARTS C	8	2010-12-01 00:00:00	2.75	22	17850	United Kin
536365	84029G	KNITTED UNION FLAG H	6	2010-12-01 00:00:00	3.39	20.34	17850	United Kin
536365	84029E	RED WOOLLY HOTTIE WH	6	2010-12-01 00:00:00	3.39	20.34	17850	United Kin
536365	22752	SET 7 BABUSHKA NESTI	2	2010-12-01 00:00:00	7.65	15.3	17850	United Kin
536365	21730	GLASS STAR FROSTED T	6	2010-12-01 00:00:00	4.25	25.5	17850	United Kin
536366	22633	HAND WARMER UNION JA	6	2010-12-01 00:00:00	1.85	11.1	17850	United Kin
536366	22632	HAND WARMER RED POLK	6	2010-12-01 00:00:00	1.85	11.1	17850	United Kin
536367	84879	ASSORTED COLOUR BIRD	32	2010-12-01 00:00:00	1.69	54.08	13047	United Kin
536367	22745	POPPY'S PLAYHOUSE BE	6	2010-12-01 00:00:00	2.1	12.6	13047	United Kin
536367	22748	POPPY'S PLAYHOUSE KI	6	2010-12-01 00:00:00	2.1	12.6	13047	United Kin
536367	22749	FELTCRAFT PRINCESS C	8	2010-12-01 00:00:00	3.75	30	13047	United Kin
536367	22310	IVORY KNITTED MUG CO	6	2010-12-01 00:00:00	1.65	9.9	13047	United Kin
536367	84969	BOX OF 6 ASSORTED CO	6	2010-12-01 00:00:00	4.25	25.5	13047	United Kin
536367	22623	BOX OF VINTAGE JIGSA	3	2010-12-01 00:00:00	4.95	14.85	13047	United Kin
536367	22622	BOX OF VINTAGE ALPHA	2	2010-12-01 00:00:00	9.95	19.9	13047	United Kin
536367	21754	HOME BUILDING BLOCK	3	2010-12-01 00:00:00	5.95	17.85	13047	United Kin
536367	21755	LOVE BUILDING BLOCK	3	2010-12-01 00:00:00	5.95	17.85	13047	United Kin
536367	21777	RECIPE BOX WITH META	4	2010-12-01 00:00:00	7.95	31.8	13047	United Kin
536367	48187	DOORMAT NEW ENGLAND	4	2010-12-01 00:00:00	7.95	31.8	13047	United Kin
536368	22960	JAM MAKING SET WITH	6	2010-12-01 00:00:00	4.25	25.5	13047	United Kin
536368	22913	RED COAT RACK PARIS	3	2010-12-01 00:00:00	4.95	14.85	13047	United Kin
536368	22912	YELLOW COAT RACK PAR	3	2010-12-01 00:00:00	4.95	14.85	13047	United Kin
536368	22914	BLUE COAT RACK PARIS	3	2010-12-01 00:00:00	4.95	14.85	13047	United Kin
536369	21756	BATH BUILDING BLOCK	3	2010-12-01 00:00:00	5.95	17.85	13047	United Kin

Figure 5-7 Retail sales data stored in MySQL database

```
> select * from weathertb
name: weathertb
time
----
1291161600000000000 temperature humidity visibility windspeed condition 0 89 0 7 26
1291248000000000000 temperature humidity visibility windspeed condition 0 94 -2 3 14
1291334400000000000 temperature humidity visibility windspeed condition 0.2 92 -4 5 11
1291420800000000000 temperature humidity visibility windspeed condition 1 90 3 9 13
1291507200000000000 temperature humidity visibility windspeed condition 1 91 2 9 6
1291593600000000000 temperature humidity visibility windspeed condition 0.2 98 -3 1 2
1291680000000000000 temperature humidity visibility windspeed condition 1 91 -2 9 3
1291766400000000000 temperature humidity visibility windspeed condition 1 90 0 8 14
1291852800000000000 temperature humidity visibility windspeed condition 1 77 1 10 16
1291939200000000000 temperature humidity visibility windspeed condition 1 85 4 10 13
1292025600000000000 temperature humidity visibility windspeed condition 1 83 7 10 11
1292112000000000000 temperature humidity visibility windspeed condition 1 90 3 9 8
1292198400000000000 temperature humidity visibility windspeed condition 0.2 91 2 9 3
1292284800000000000 temperature humidity visibility windspeed condition 1 81 1 9 8
1292371200000000000 temperature humidity visibility windspeed condition 0.6 90 2 7 11
1292457600000000000 temperature humidity visibility windspeed condition 1 88 3 9 19
1292544000000000000 temperature humidity visibility windspeed condition 1 88 -3 10 14
1292630400000000000 temperature humidity visibility windspeed condition 1 94 -3 8 6
1292716800000000000 temperature humidity visibility windspeed condition 0 87 -2 10 11
1292803200000000000 temperature humidity visibility windspeed condition 1 89 -4 7 3
1292889600000000000 temperature humidity visibility windspeed condition 1 96 1 6 6
1292976000000000000 temperature humidity visibility windspeed condition 0.6 94 1 5 8
1293062400000000000 temperature humidity visibility windspeed condition 1 82 0 10 21
1293148800000000000 temperature humidity visibility windspeed condition 1 82 1 10 19
1293235200000000000 temperature humidity visibility windspeed condition 1 85 -1 10 5
1293321600000000000 temperature humidity visibility windspeed condition 1 79 2 10 11
1293408000000000000 temperature humidity visibility windspeed condition 1 79 2 10 11
1293494400000000000 temperature humidity visibility windspeed condition 1 93 3 4 8
```

Figure 5-8 Weather data stored in InfluxDB

Data analytic technologies is employed by tremendous numbers of company to optimize their business revenue by analyse and gain certain insights from their business data in order to do informed business decision based on the history sales data. According to researcher's study, the weather is considered as a factor impact on sales. Moreover, after the custom query across both MySQL and InfluxDB is being processed by data scientist to collect the data results, thus the retrieved weather based retail sales data is then able to be displayed on Grafana for efficient data visualisation.

With the intention of testing the multi-database custom query function together visualize function of the custom query data, we have generated numerous different panel on Grafana to visually interpret the correlation between weather and sales data by employ the use of scatter plot visualizations, dual axis graph visualization together with histogram graph visualizations to depict the correlated weather based retail sales data.

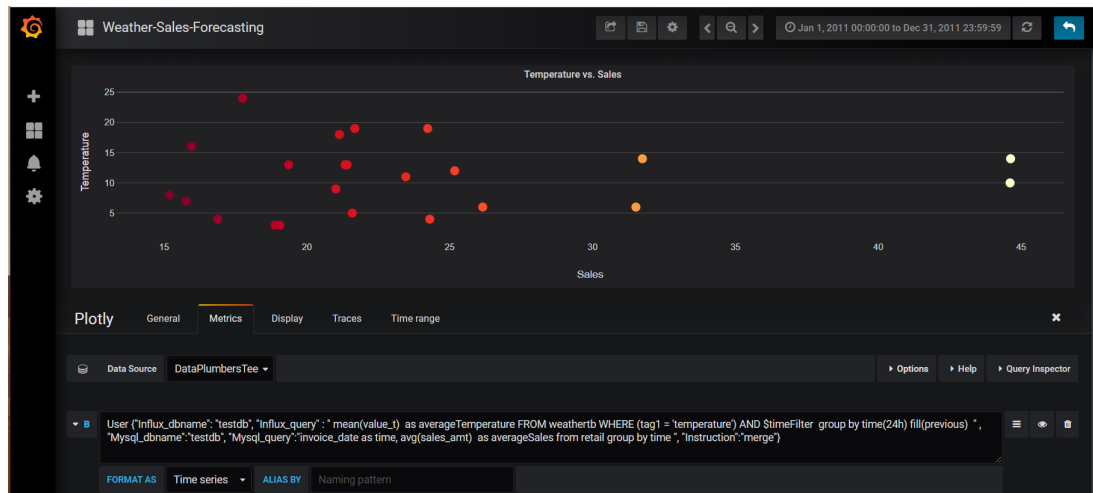


Figure 5-9 Scatter plot visualization between Temperature and Sales

Based on the figures, it shows our program is able to process the custom query function to retrieve average temperatures data and average sales data from both InfluxDB and MySQL respectively and thus display the custom query results to Grafana in scatter plot fashion. Plus, the scatter plot is able to display the correlation between temperature data and sales data within a specified time interval.

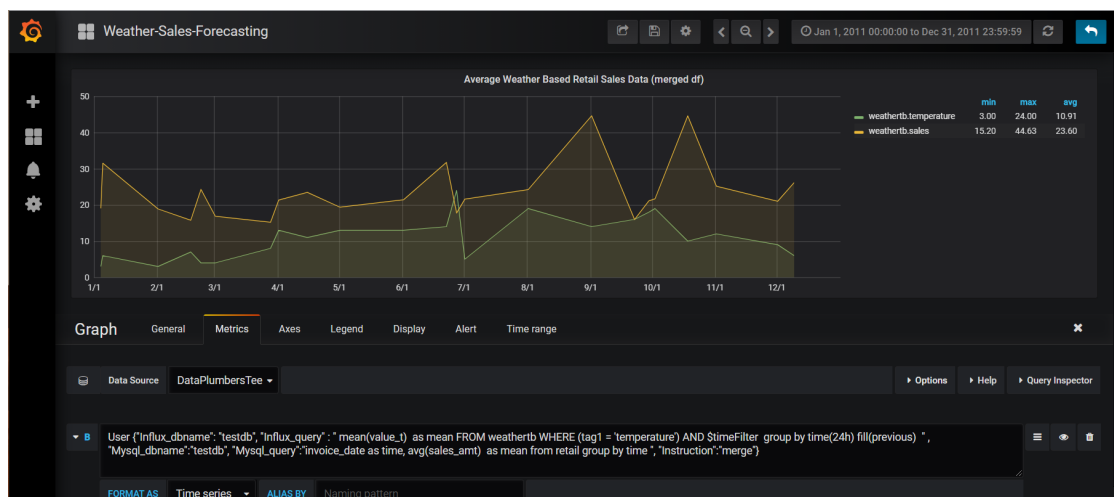


Figure 5-10 Dual axis visualization of Temperature and Sales

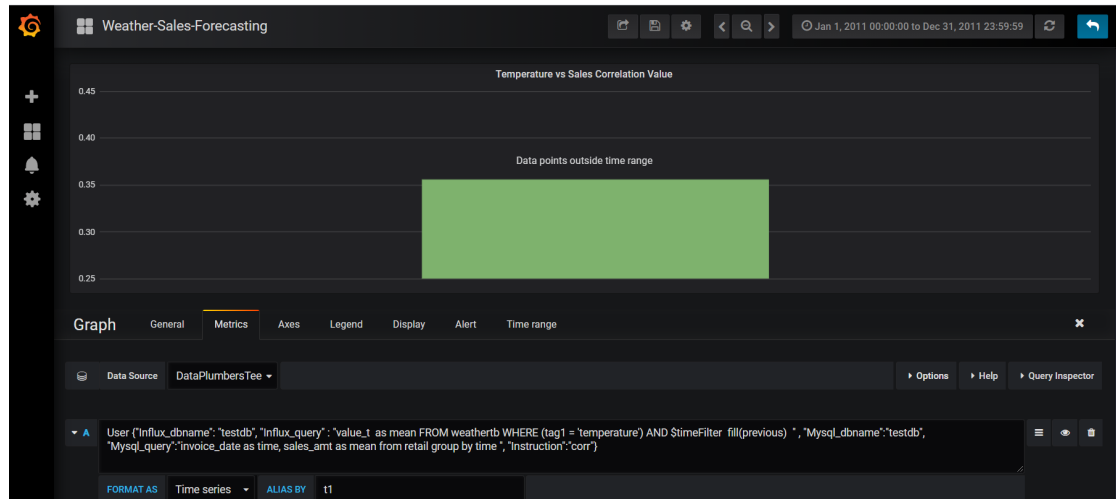


Figure 5-11 Histogram visualization of correlation value between Temperature and Sales

To summarise all of the test case results, our developed program is able to achieve all our expected requirements in order to extend the Grafana analytic functions in term of query performance together with extended custom query visualization function that enable data scientist to interpret and generate analytic visualization report by performing customized query across multiple databases more efficiently and effectively.

5.3 Execution Time Analysis of Extended Grafana AnalyticPlatform

As a matter of fact, the program execution time analysis is often consider as a mandatory steps to evaluate the performance of the developed software program component. Instead of conducting execution time analysis of single extended Grafana analytic platform, we have further explore and compare the performance between both extended Grafana analytic platform and simple python visualization program under scenario performing same data query functions to retrieve and display the data in same type of visualization report. The execution performance of both programs are generally being measured from program initiation at presentation of some input and function call follow by a termination whenever the expected output of the program is being delivered.

```
import sys
import json
import time
from urllib2 import unquote
from DuplexFetch import DuplexFetch

start = time.time()

#load incoming query in JSON
unquoted_query = unquote(sys.argv[1]).replace("+", " ")
res = json.loads(unquoted_query)

fetch = DuplexFetch(res)
tee = fetch.get_tee()

#get query result dataframe
output_df = fetch.get_resultFrame()

#display result dataframe to grafana
tee.ps_display_dataframe_to_grafana(output_df)

end = time.time()

print("Elapsed Time_in_seconds: ")
print(end - start)

#Append test result to file
file1 = open("GrafanaInfluxDF.txt", "a")#append mode
file1.write(str(end-start)+"\n")
file1.close()
```

Source Code 5-1 User python program measure the execution time of processing query and display Influx dataframe to Grafana

For ease of data storing and retrieving for later used, all attempts of the program execution times is written into a text file and being loaded and stored in MySQL database correspondingly.

Chapter 5 Experimental Result

```
, [14], [131371200000, 16], [131379840000, 19], [131388480000, 21], [131397120000, 18], [131405760000, 16], [131414400000, 18], [131423040000, 18], [131431680000, 16], [131440320000, 16], [131448960000, 16], [131457600000, 13], [131466240000, 14], [131474880000, 14], [131483520000, 14], [131492160000, 20], [131500800000, 20], [131509440000, 17], [131518080000, 16], [131526720000, 17], [131535360000, 14], [131544000000, 17], [131552640000, 19], [131561280000, 20], [131569920000, 18], [131578560000, 19], [131587200000, 17], [131595840000, 16], [131604480000, 13], [131613120000, 18], [131621760000, 14], [131630400000, 13], [131639040000, 14], [131647680000, 18], [131656320000, 16], [131664960000, 16], [131673600000, 14], [131682240000, 13], [131690880000, 17], [131699520000, 20], [131708160000, 17], [131716800000, 17], [131725440000, 18], [131734080000, 18], [131742720000, 18], [131751360000, 19], [131760000000, 21], [131768640000, 17], [131777280000, 19], [131785920000, 13], [131794560000, 12], [131803200000, 11], [131811840000, 18], [131820480000, 18], [131829120000, 18], [131837760000, 17], [131846400000, 14], [131855040000, 11], [131863680000, 8], [131872320000, 9], [131880960000, 12], [131889600000, 10], [131898240000, 9], [131906880000, 7], [131915520000, 10], [131924160000, 10], [131932800000, 14], [131941440000, 12], [131950080000, 13], [131958720000, 11], [131967360000, 12], [131976000000, 12], [131984640000, 14], [131993280000, 16], [132001920000, 13], [132010560000, 12], [132019200000, 13], [132027840000, 17], [132036480000, 14], [132045120000, 12], [132053760000, 10], [132062400000, 12], [132071040000, 10], [132079680000, 13], [132088320000, 11], [132096960000, 9], [132105600000, 12], [132114240000, 11], [132122880000, 9], [132131520000, 8], [132140160000, 6], [132148800000, 12], [132157440000, 12], [132166080000, 7], [132174720000, 7], [132183360000, 6], [1321920000, 11], [132200640000, 7], [132209280000, 11], [132217920000, 11], [132226560000, 10], [132235200000, 10], [132243840000, 4], [132252480000, 11], [132261120000, 9], [132269760000, 9], [132278400000, 6], [132287040000, 10], [132295680000, 7], [132304320000, 4], [132312960000, 4], [132321600000, 7], [132330240000, 10], [132338880000, 6], [132347520000, 7], [132356160000, 6], [132364800000, 6], [132373440000, 7], [132382080000, 6], [132390720000, 7], [132399360000, 2], [132408000000, 3], [132416640000, 1], [132425280000, 2], [132433920000, 7], [132442560000, 9], [132451200000, 9], [132459840000, 8], [132468480000, 6], [132477120000, 7], [132485760000, 11], [132494400000, 8], [132503040000, 9], [132511680000, 8], [132520320000, 7], [132528960000, 12]], "name": "weatherbt", "columns": [{"time", "value_t"}], "statement_id": 0}}

Elapsed Time in seconds:
0.219964981079
```

Figure 5-12 Program execution time is measured between the interval of Influx query input follow by data display to Grafana



Figure 5-13 Processing custom query to display Daily Temperature panel on Grafana

Chapter 5 Experimental Result

In a similar fashion as measure the execution time of processing Influx query and display the data to Grafana, all the attempts of different execution time is being append to a GrafanaMySQL text file and being stored to MySQL databases.

```
10560000, 10.2], [132010560000, 15.0], [132010560000, 15.0], [132010560000, 15.0], [132010560000, 18.72], [132010560000, 9.36], [132010560000, 6.96], [132010560000, 6.96], [132010560000, 6.96], [132010560000, 19.8], [132010560000, 19.9], [132010560000, 25.2], [132010560000, 11.4], [132010560000, 101.88], [132010560000, 25.2], [132010560000, 15.0], [132010560000, 19.8], [132010560000, 0.28], [132010560000, 17.7], [132010560000, 30.0], [132010560000, 38.16], [132010560000, 24.96], [132010560000, 19.8], [132010560000, 5], [132010560000, 15.0], [132010560000, 15.0], [132010560000, 29.7], [132010560000, 30.0], [132010560000, 10.2], [132010560000, 17.0], [132010560000, 12.7], [132010560000, 17.0], [132010560000, 19.5], [132010560000, 23.4], [132010560000, 15.0], [132010560000, 19.8], [132010560000, 15.0], [132010560000, 42.5], [132010560000, 163.8], [132010560000, 42.5], [132010560000, 62.4], [132010560000, 62.4], [132010560000, 20.8], [132010560000, 20.8], [132010560000, 62.4], [132010560000, 62.4], [132010560000, 20.8], [132010560000, 15.0], [132010560000, 17.0], [132010560000, 9.36], [132010560000, 10.5], [132010560000, 17.0], [132010560000, 19.8], [132010560000, 2], [132010560000, 6.96], [132269760000, 39.6], [132269760000, 17.85], [132269760000, 15.0], [132269760000, 17.34], [132269760000, 16.6], [132269760000, 16.6], [132269760000, 20.4], [132269760000, 20.4], [132338880000, 16.5], [132338880000, 12.48], [132338880000, 17.4], [132338880000, 58.0], [132338880000, 66.6], [132338880000, 51.84], [132338880000, 88.8], [132338880000, 4.68], [132338880000, 4.56], [132338880000, 23.4], [132338880000, 15.0], [132338880000, 20.4], [132338880000, 27.04], [132338880000, 20.4], [132338880000, 17.0], [132338880000, 17.85], [132338880000, 30.0], [132338880000, 15.0], [132338880000, 8.5], [132338880000, 10.08], [132338880000, 10.5], [132338880000, 15.0], [132338880000, 10.2], [132338880000, 8], [132338880000, 15.0], [132338880000, 11.4], [132338880000, 23.4], [132338880000, 23.6], [132338880000, 30.0], [132338880000, 2], [132338880000, 70.8], [132338880000, 23.4], [132338880000, 19.8], [132338880000, 15.0], [132338880000, 15.0], [132338880000, 15.0], [132338880000, 12.6], [132338880000, 16.6], [132338880000, 16.6], [132338880000, 14.85]], "name": "sales", "columns": ["time", "sales"]}, {"status_id": 0}]}
```

Elapsed Time_in_seconds:
0.553485155106

Response Completed

Figure 5-14 Program execution time is measured between the interval of MySQL query input follow by data display to Grafana



Figure 5-15 Processing custom query to display Retail Sales panel on Grafana

In the event of conduct the program execution time analysis of python data visualization program, it practice the In a similar fashion to extended Grafana analytic platform as capturing the program execution time of processing the MySQL query follow by generation of a visualization report and storing the elapsed time in text file and thus loaded in MySQL database for further data retrieving. However, this python

Chapter 5 Experimental Result

data visualization program has possessed differences from extended Grafana analytic platform as instead of display visualization report on Grafana, it was able to generate a data visualization by import matplotlib module and export the plot in PNG format.

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

start = time.time()

client = DataframeClient("localhost", 8086, "admin", "admin", "testdb")
query = "SELECT mean(value_t) as mean FROM weatherb WHERE (tag1 = 'temperature') AND time >= 1293811200000ms and time <= 1325347199000ms group by time(24h) fill(previous)"
results = client.query(query, chunked=False)
client.close()

df = results[results.keys()[0]]
df.index.name = 'time'

date = list(df.index)
mean = df['mean']

plt.figure(figsize=(15.0, 10.0))
plt.plot(date, mean)
plt.xlabel('Date')
plt.ylabel('Average Temperature')
plt.title('Average Temperature Report')

plt.savefig('Average_Temperature_Report.png')

end = time.time()

#Append test result to file
file1 = open("InfluxDF.txt", "a") #append mode
file1.write(str(end-start)+"\n")
file1.close()
```

Source Code 5-2 Python data visualization program process Influx query and plot
Influx data visualization report

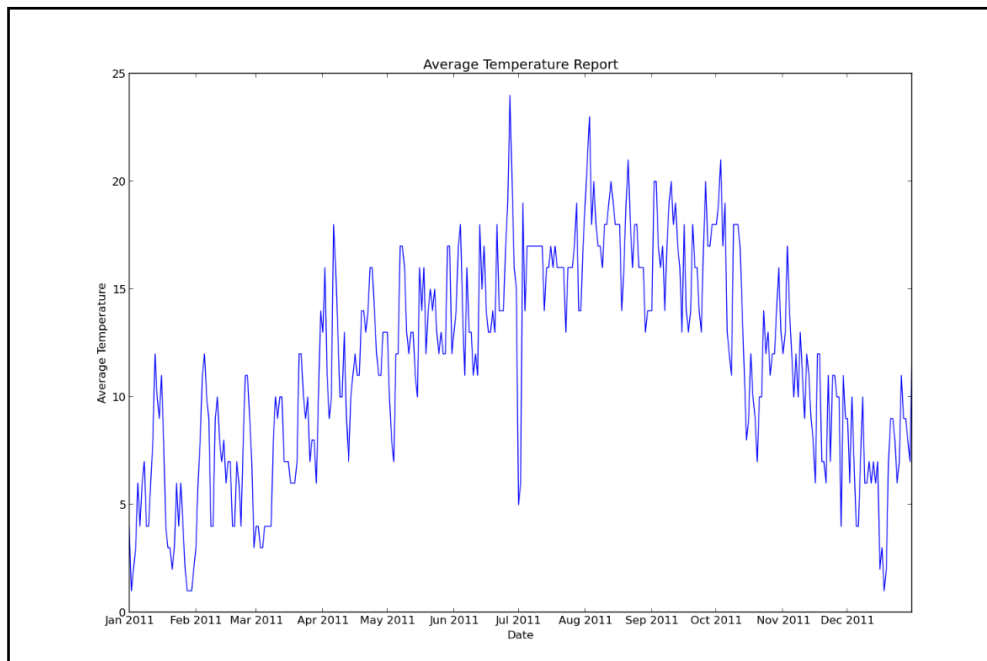


Figure 5-16 Influx data visualization report generated using python matplotlib module

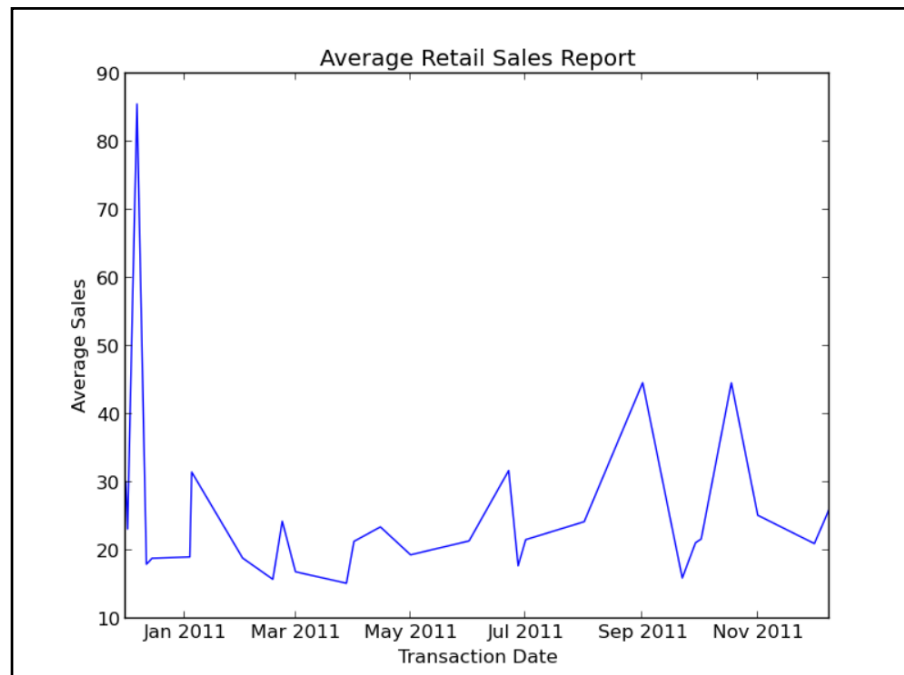


Figure 5-17 MySQL data visualization report generated using python matplotlib module

In a similar fashion, the Average Retail Sales report above is being plotted by using python matplotlib module, program execution time will be recorded and stored in MySQL databases for further performance analysis use.

The comparison performance of both Python data visualization program and Extended Grafana data analytic platform are being measured and compare in average case:

```
mysql> select avg(time_in_seconds) from GrafanaInfluxDF;
+-----+
| avg(time_in_seconds) |
+-----+
| 0.1709929923216502 |
+-----+
1 row in set (0.01 sec)
```

Figure 5-18 Average program execution time of extended Grafana program

```
mysql> select avg(time_in_seconds) from InfluxDF;
+-----+
| avg(time_in_seconds) |
+-----+
| 0.39754542509714763 |
+-----+
1 row in set (0.00 sec)
```

Figure 5-19 Average program execution time of Python data visualization program

	Average Program Execution Time (in seconds)	
	Python Data Visualization Program with Matplotlib	Extended Grafana Analytic Module with DataPlumbersTee
Influx Datafame	0.397	0.171
MySQL Dataframe	0.252	0.279

Table 5-3 Overall results of program performance between Python data visualization program and extended Grafana analytic platform

In overall speaking, the extended Grafana analytic platform is outperform the Python data visualization program with matplotlib as it possessed lower program execution time in context of processing same input query and generate same type visualization report. Lower program execution time is recognised as the program execute the task in a reasonable or faster speed with lower program energy consumption. In contrast, if the execution of the program is running way too slow it will cost higher program execution time that lead to higher program energy consumption to execute the specific task. Apart from execution time, the user satisfaction, program error rates together with program energy consumption is also the important metric to evaluate and manage the quality of the software program.

In addition, the implementation of the program execution timing analysis between both Python data visualization program with Matplotlib and extended Grafana analytic platform is not only deliver the program performance measured results in execution time but it also depict the vast different between both program in

performance of generating data visualization report. In term of efficiency and effectiveness, the higher flexibility of the extended Grafana analytic platform is enable users to generate the data visualization that are subject change with time range controls feature provided on Grafana query editor. The visualization generated to display on extended Grafana platform can be updated whenever users edit on the query time interval controls. Conversely, in generate data visualization report using Python Matplotlib users is required to specify the data to plot with titles, labels together with the plotting types and thus only the data visualization report will be shown or saved in PNG format.

In view of the data analytic market, the extended Grafana analytic platform is able to gain competitive advantage with its extended analytic function that able to process custom query function and visualize the data from custom query on wide range support of different visualization panel. For instances, the data scientists are able to generate the visualization report once and submit for its client to further interpret or update the report with flexible time range controls. Consequently, this could lead to the boasting of the effectiveness and efficiency of the Grafana analytic platform function to deliver important data insights to company for further decision making that could devote to optimise the business revenue.

Chapter 6 Conclusion

IoT database benchmark is crucial for individuals, companies or organisations to investigate the different database solutions available today. In this project, the benchmark work is developed to evaluate the performance between the NoSQL and SQL database like InfluxDB and MySQL through the series of data insertion and query performance tests. The delivered benchmark work was able to solve the incomprehensive benchmark work between NoSQL and SQL database available in the market. As the NoSQL and SQL database will possess different performance towards the big data with high volume and velocity of data. Therefore, the comprehensive benchmark results between InfluxDB and MySQL in this project will enable users to gain a better understanding towards the differentiations, pros and cons of both databases.

Grafana is a powerful visualization tools that associated with wide range of databases as including NoSQL and SQL databases. However, there is a limitations possessed by the Grafana that troubles the users unable to perform the custom query functions to extract desired group of data from the specific databases. Moreover, the users also unable to visualize the data from the custom query as the Grafana failed to performed the custom query functions. Thus, the purpose of this project is to delivered a programs to address the failed custom query functions of Grafana by enable the users to use the program to perform customized query data from different data sources connected to the program and thus pass all the returning rows to the Grafana for further data visualization used. This enable the users to visualize their desired data collected across multiple databases for gaining a better insights from their customized query data.

Furthermore, the novelty of our program to enable users perform custom query on Grafana as our program are able to processing the query to fetch data from heterogenous data sources like MySQL and InfluxDB. Moreover, the custom query design is functions to communicate with both SQL and NoSQL databases. In today markets, the enterprise data is pouring in various scenarios like customer relationship

management system, web application and business databases system. All of these scenarios possessed the importance of our program developed to extend the Grafana analytic work that enable users to perform custom query to extract data from heterogenous data sources. In addition, our program is developed to handle the integration of data extracted from both MySQL and InfluxDB follow by data processing and standardizing in order to deliver an effective and accurately analytic visualization report on Grafana. Plus, with extended Grafana function users are able to optimize the data analytic on Grafana. In a business analytic scenario there are various visualization reports are required to interpret the huge business data and thus deliver certain insights for business decision making that could lead to optimize the business revenue. Due to high data traffic flow in the business, data scientist may require to generate visualization report according periodic data change over time. In this case, our program is developed to enable users to perform such efficient extended function on a mature visualization tools Grafana that provide high flexibility to users as enable the users to visualize the data with time range controls. This features could save data scientist from generate visualization report more frequently due to slightly change in time range request by their client.

In conclude, the program was able to support multiple data sources and compatible with the visualization tools Grafana to perform data analytics functions. In future, the program can be improve by designing to support more different databases and thus optimise the use of the programs with the Grafana data analytic platform.

BIBLIOGRAPHY

Bogaards, L. 2019, *An Introduction to Graphite - DZone DevOps*. Available from: < <https://dzone.com/articles/an-introduction-to-graphite> >. [1 Aug 2019].

Churilo, C. 2019, *InfluxDB vs. Graphite for Time Series Data & Metrics Benchmark* | InfluxData, InfluxData. Available from: < <https://www.influxdata.com/blog/influxdb-outperforms-graphite-in-time-series-data-metrics-benchmark/> >.[1 Aug.2019].

CI, T. 2010, *OpenTSDB - A Distributed, Scalable Monitoring System*. Available from:< <http://opentsdb.net/overview.html> >.[1 Aug 2019].

Isley, N. and Savage, R. 2019, *InfluxDB Open Source Time Series Database* | InfluxDB | InfluxData. Available from:<<https://www.influxdata.com/products/influxdb-overview/>>.[31 Jul 2019].

Kulkarni, A. 2018, *What the heck is time-series data (and why do I need a time-series database)?*. Available from:< <https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563/> >.[1 Aug 2019].

Pagano Dritto, G. 2019, *An Overview on Elasticsearch and its usage*. Available from:<<https://towardsdatascience.com/an-overview-on-elasticsearch-and-its-usage-e26df1d1d24a> >.[5 Aug 2019].

Powell-Morse, A. 2016, *What Is Rapid Application Development (RAD) and How Do You Use It?*, Airbrake Blog. Available from:< <https://airbrake.io/blog/sdlc/rapid-application-development> >.[3 Aug 2019].

Prometheus.io. 2014, *Overview* | Prometheus. Available from:< <https://prometheus.io/docs/introduction/overview/> >.[31 Jul 2019].

Bibliography

Ranger, S. 2018, *What is the IoT? Everything you need to know about the Internet of Things right now* | ZDNet. Available from:< <https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/> >.

[31 Jul 2019].

Rouse, M. (2013). *What is MySQL? - Definition from WhatIs.com*. [online] SearchOracle. Available from:<<https://searchoracle.techtarget.com/definition/MySQL>>.[18 Feb. 2020].

Zhou, Z. 2019, *Key Concepts and Features of Time Series Databases*, Alibaba Cloud Community. Available from:< https://www.alibabacloud.com/blog/key-concepts-and-features-of-time-series-databases_594734 >.[1 Aug 2019].

Appendix A Poster

FYP Poster

A Data Analytic Module
to Extend
Grafana Data Analytic Function

Developed by Har Pui See Supervised by Dr Ooi Boon Yaik

INTRODUCTION

- In this data driven age, an analytic platform is crucial to visualize and analyze the massive volume of data.
- Grafana is a leading visualization and analytic solutions that support a wide range of data sources and rich set of graphing options.

PROBLEM STATEMENTS

- Incomprehensive database benchmarking between time series database and relational database
- Grafana cannot perform customize query function.
- Grafana cannot visualise customized query function as it is unable to perform customize query function.

Proposed System

Visualization Tool

Grafana

HTTP

Program

Data Source

influxdb

MySQL

- A data analytic module was developed to address the limitation of Grafana that failed to perform custom query functions by enhance the analytic functions of Grafana.
- A program is designed to seat between datasources and Grafana, it enable users to perform custom query functions across multiple databases.
- Next, the program will delivered the query result to Grafana for visualization purpose.

Proposed System

CONTRIBUTION

The functionalities of the Grafana anaytic platform will be enhanced via this analytic module by

- Enable users to perform multi-database query functions and visualize the data from custom query for getting better insights from the data.
- In future, the data analytic module can be further improved to support more different data sources for efficient visualization of custom query data.

CONCLUSION


- The interactive analytic solution of Grafana is crucial for high data driven companies or organizations.
- Thus the Grafana must be further improve to achieve higher flexibility and comprehensive to become an effective analytic solutions that able to support a wider range of data sources efficiently.

UTAR


UNIVERSITI TERAJUAN ANJALIS BANGSA

BCS (HONS) Computer Science
Faculty of Information and Communication
Technology (Perak Campus), UTAR

Appendix B Plagiarism Check Result

 feedback studio

Har Pui See | A Data Analytic Module To Extend Grafana Analytic Function



Chapter 1 Project Background

1.1 Project Background

In this data driven age, big data analytics is important to explore the massive volume of the IoT data generated from all connected sensor devices continuously to get better insights of the data for value creation. According to the growth of the application of time series database in big data fields and industrial IoT for data monitoring and analysis, number of the developed time series database is increased and optimized for time series data. However, a vast number of these time series database has possessed differences in terms of algorithm approach, strengths and weaknesses. Thus, an effective database benchmarking tool is crucial for comparing and evaluating the database for time series data.

Internet of Things(IoT) is described as a computer environment that enable massive number of devices connect over the internet physically and thus enabled the collecting and exchanging of data through wireless sensor and actuator

Match Overview

1%

1	mafiadoc.com Internet Source	<1% >
2	Submitted to Universiti ... Student Paper	<1% >
3	Submitted to QA Learn... Student Paper	<1% >
4	www.iist.ik Internet Source	<1% >
5	www.foxinfosoft.com Internet Source	<1% >
6	Sandin, Marianne, Joha... Publication	<1% >
7	iwmptripura.in Internet Source	<1% >

A Data Analytic Module To Extend Grafana Analytic Function

ORIGINALITY REPORT			
1%	1%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	mafiadoc.com Internet Source		<1%
2	Submitted to Universiti Teknologi MARA Student Paper		<1%
3	Submitted to QA Learning Student Paper		<1%
4	www.iist.lk Internet Source		<1%
5	www.foxinfosoft.com Internet Source		<1%
6	Sandin, Marianne, Johan Teleman, Johan Malmström, and Fredrik Levander. "Data processing methods and quality control strategies for label-free LC-MS protein quantification", Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics, 2013. Publication		<1%
7	iwmptripura.in Internet Source		<1%

Appendix C Supervisor's Comments on Originality Report

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



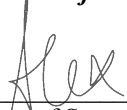
FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY

Full Name(s) of Candidate(s)	HAR PUI SEE
ID Number(s)	17ACB05558
Programme / Course	CS
Title of Final Year Project	A Data Analytic Module To Extend Grafana Analytic Function

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u>1</u> % Similarity by source Internet Sources: <u>1</u> % Publications: <u>0</u> % Student Papers: <u>0</u> %	
Number of individual sources listed of more than 3% similarity: <u>No</u>	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.



Signature of Supervisor

Name : Dr. Ooi Boon Yaik

Date : 10-Sep-2020

Signature of Co-Supervisor

Name: _____

Date : _____

Appendix D Checklist for FYP 1 Thesis Submission



UNIVERSITI TUNKU ABDUL RAHMAN


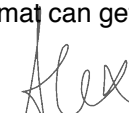
**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP1 THESIS SUBMISSION

Student Id	17ACB05558
Student Name	HAR PUI SEE
Supervisor Name	DR. OOI BOON YAIK

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
/	Title Page
/	Signed form of the Declaration of Originality
/	Abstract
/	Table of Contents
/	List of Figures (if applicable)
/	List of Tables (if applicable)
/	List of Symbols (if applicable)
/	List of Abbreviations (if applicable)
/	Chapters / Content
/	Bibliography (or References)
/	All references in bibliography are cited in the thesis, especially in the chapter of literature review
/	Appendices (if applicable)
/	Poster
/	Signed Turnitin Report (Plagiarism Check Result – Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.  (Signature of Student) Date: 10 Sep 2020	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction.  (Signature of Supervisor) Date: 10 Sep 2020
--	---