การเปรียบเทียบประสิทธิภาพของระบบฐานข้อมูล NoSQL ประเภทกราฟบนสภาวะเชิงเดี่ยว

Performance Comparisons of Graph-based NoSQL Systems on Stand-alone Environment

ธันวา ปิ่นทอง

นนทกานต์ คูหาอภิรมย์

นิศารัตน์ ลิ่มอดิศัย

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)

ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

Performance Comparisons of Graph-based NoSQL Systems on Stand-alone Environment

Tanwa Pinthong

Nontakarn Kuhaapirom

Nisarat Limadisai

A SPECIAL PROBLEM SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF SCIENCE (COMPUTER SCIENCE)

DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2017

หัวข้อปัญหาพิเศษ การเปรียบเทียบประสิทธิภาพของระบบฐานข้อมูล NoSQL ประเภทกราฟบนสภาวะเชิงเดี่ยว

Performance Comparisons of Graph-based NoSQL Systems on Stand-alone Environment

ชื่อนักศึกษา นายธันวา ปิ่นทอง รหัสนักศึกษา 57050249

นายนนทกานต์ คูหาอภิรมย์ รหัสนักศึกษา 57050253

นางสาวนิศารัตน์ ลิ่มอดิศัย รหัสนักศึกษา 57050263

ปริญญา วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)

ภาควิชา วิทยาการคอมพิวเตอร์

ปีการศึกษา 2560

อาจารย์ที่ปรึกษา ดร.กุลสวัสดิ์ จิตขจรวานิช

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์) ประจำปีการศึกษา 2560

คณะกรรมการสอบ	ลายมือชื่อ
ผศ.กฤษฎา บุศรา ประธานกรรมการ	
อ.สันธนะ อู่อุดมยิ่ง กรรมการ	
ดร.กุลสวัสดิ์ จิตขจรวานิช กรรมการและอาจารย์ที่ปรึกษา	

หัวข้อปัญหาพิเศษ การเปรียบเทียบประสิทธิภาพของระบบฐานข้อมูล NoSQL ประเภทกราฟบนสภาวะเชิงเดี่ยว

ชื่อนักศึกษา นายธันวา ปิ่นทอง รหัสนักศึกษา 57050249

นายนนทกานต์ คูหาอภิรมย์ รหัสนักศึกษา 57050253

นางสาวนิศารัตน์ ลิ่มอดิศัย รหัสนักศึกษา 57050263

ปริญญา วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)

ภาควิชา วิทยาการคอมพิวเตอร์

คณะ วิทยาศาสตร์

มหาวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.)

ปีการศึกษา 2560

อาจารย์ที่ปรึกษา ดร.กุลสวัสดิ์ จิตขจรวานิช

บทคัดย่อ

ในปัจจุบัน NoSQL ประเภทกราฟได้รับความนิยมสำหรับลักษณะข้อมูลแบบเครือข่ายด้วยเหตุนี้จึงมีการพัฒนาระบบฐานข้อมูลประเภทกราฟอย่างแพร่หลาย อย่างไรก็ตามการเลือกระบบฐานข้อมูลที่เหมาะสมไม่ใช่เรื่องง่าย แต่ละระบบมีคุณสมบัติและลักษณะเฉพาะรวมไปถึงข้อดีข้อเสียที่แตกต่างกัน เราได้เลือกระบบฐานข้อมูล NoSQL ประเภทกราฟตามความนิยมจากเว็บไซต์ต่าง ๆ มาใช้ในการทดลอง เป้าหมายของงานวิจัยนี้คือการทดสอบและเปรียบเทียบประสิทธิภาพของระบบฐานข้อมูล ซึ่งผลของการทดลองสามารถนำมาช่วยตัดสินใจเลือกระบบฐานข้อมูลที่เหมาะสมตามความต้องการของผู้ใช้งานได้ ตัวชี้วัด (evaluation criteria) ที่ใช้ในการวิจัยนี้ประกอบด้วย: 1) เวลาในการคำนวณ 2) การใช้หน่วยความประมวลผลกลาง CPU 3) การใช้หน่วยความจำแรม โดยจะมีการทดสอบกับกลุ่มของชุดข้อมูล 2 ประเภท ได้แก่ข้อมูลที่ไม่ใช่กราฟ และข้อมูลกราฟโดยพิจารณาจากการประมวลผลบน query ประเภทต่าง ๆ

คำสำคัญ : การเปรียบเทียบประสิทธิภาพ ฐานข้อมูล NoSQL ประเภท Graph-based

Title	Performance Comparisons of Graph-based NoSQL Systems on Stand-alone Environment
Student	Mr. Tanwa Pinthong Student ID 57050249
	Mr. Nontakarn Kuhaapirom Student ID 57050253
	Miss Nisarat Limadisai Student ID 57050263
Degree	Bachelor of Science (Computer Science)
Department	Computer Science
Faculty	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2017
Advisor	Dr. Kulsawasd Jitkajornwanich

Abstract

At present, graph-based NoSQLs are becoming popular as network data infrastructure and storage; and as a result, several types of graph databases from diverse vendors are emerged. Out of many options, choosing the right one is not easy. Each system has its own Features and characteristics as well as pros and cons. We selected graph-based nosql systems from different sites according to their reputation in our experiments. The goal of this research is to conduct performance comparison experiments, the result of which could be of helps in making decision as to which is the most suitable system for given requirements. Three evaluation criteria consists of computation time, CPU, memory and disk usage, which are tested on two groups of datasets: non-graph and graph data, based on certain types of queries.

Keywords : Comparison Performance, NoSQL Graph-based

กิตติกรรมประกาศ

ปัญหาพิเศษเรื่อง การเปรียบเทียบประสิทธิภาพของระบบฐานข้อมูล NoSQL ประเภทกราฟบนสภาวะเชิงเดี่ยวสำเร็จลุล่วงไปได้ด้วยดีจากช่วยเหลือและสนับสนุนของบุคคลหลายท่าน คณะผู้จัดทำขอขอบพระคุณผู้มีพระคุณทั้งหลายดังนี้

ขอขอบพระคุณ บิดา มารดา และสมาชิกในครอบครัวทุกคน ผู้ซึ่งอยู่เบื้องหลัง คอยอบรมสั่งสอนและให้กำลังใจตลอดระยะเวลาในการจัดทำปัญหาพิเศษในครั้งนี้

ขอขอบพระคุณ ดร.กุลสวัสดิ์ จิตขจรวานิช อาจารย์ที่ปรึกษาปัญหาพิเศษที่คอยให้คำแนะนำและให้คำปรึกษาแนวทางในการทำงานอย่างใกล้ชิด รวมถึงช่วยแนะนำการแก้ไขปัญหาที่เกิดขึ้นในการจัดทำปัญหาพิเศษ

ขอขอบพระคุณ คณาจารย์ในภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ ซึ่งได้ให้ความรู้ ทางวิชาการและความรู้อันเป็นพื้นฐาน ทำให้คณะผู้จัดทำสามารถที่ดำเนินงานปัญหาพิเศษจนประสบผลสำเร็จ

ท้ายสุดนี้ ขอขอบคุณเพื่อน ๆ ทุกคน ที่ให้คำปรึกษาและเป็นกำลังใจเสมอมา

ธันวา ปิ่นทอง

นนทกานต์ คูหาอภิรมย์

นิศารัตน์ ลิ่มอดิศัย

สารบัญ

หน้า

สารบัญ

หน้า

สารบัญตาราง

ตาราง **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของงานวิจัย

ในการจัดการฐานข้อมูลเรามักใช้ Relational database management systems (RDBMS) เนื่องด้วยความสะดวกและความแพร่หลายในด้านการใช้งาน อย่างไรก็ตาม RDBMS ไม่เหมาะสมกับการจัดการข้อมูลขนาดใหญ่ในปัจจุบัน จึงทำให้เกิดเทคโนโลยีการจัดการข้อมูลขนาดใหญ่ขึ้นมาเรียกว่า NoSQL (Not Only SQL) สามารถรองรับการเก็บข้อมูลที่หลากหลายรองรับการเก็บข้อมูลแบบกระจาย เพื่อจัดการข้อมูลที่เครื่องต่าง ๆ และรองรับการทำงานอย่างรวดเร็ว ในระบบฐานข้อมูลแบบ NoSQL สามารถแบ่งออกเป็น 4 ประเภท [22] คือ 1) Key-value Database ฐานข้อมูลประเภทนี้มีการดึงข้อมูลจาก key ได้โดยตรงและเก็บแบบคู่อันดับ รองรับการจัดเก็บโครงสร้างข้อมูลที่เรียบง่าย จึงทำให้การดึงข้อมูลเร็วกว่าระบบฐานข้อมูลประเภทอื่น ตัวอย่างเช่น Riak Redis และ Amazon Dynamo ตัวอย่างเช่น Riak, Redis และ Amazon Dynamo 2) Document Database ฐานข้อมูลประเภทนี้มีความยืดหยุ่นสูง จึงรองรับโครงสร้างข้อมูลที่ซับซ้อนและการประมวลผลแบบ complex ได้ตัวอย่างเช่น MongoDB CouchDB และ RAVENDB 3) Column-Oriented Database เป็นฐานข้อมูลที่มีการจัดเก็บข้อมูลแบบกลุ่ม Column มีการเก็บข้อมูลคล้าย Row ใน RDBMS แต่มีขนาดใหญ่กว่า ตัวอย่างเช่น HBase Cassandra และ Google BigTable 4) Graph-based ใช้การจัดเก็บข้อมูลที่มีความสัมพันธ์ระหว่างกันแบบกราฟ โดยข้อมูลแต่ละชุดเป็นข้อมูลเชิง Graph และอยู่ในรูปแบบความสัมพันธ์แบบเครือข่าย ตัวอย่างเช่น Neo4j OrientDB และ ArangoDB ซึ่งข้อมูลในแต่ละประเภทต้องเลือกให้เหมาะสมกับระบบฐานข้อมูลที่ใช้งาน

ในปัจจุบันฐานข้อมูล NoSQL ประเภท Graph-based ได้ถูกใช้งานกับแอพพลิเคชั่นที่มีความสัมพันธ์ของข้อมูลแบบโครงข่ายในด้านต่าง ๆ เช่น social networking, scientific paper citation, capital asset clustering และ direction in map [23] โดยทางผู้วิจัยจึงนำระบบฐานข้อมูลประเภทนี้มาปรับใช้ในการวิจัยครั้งนี้ และผู้วิจัยจึงทำการคัดเลือกระบบฐานข้อมูลจากเว็บไซต์จัดอันดับ อาทิ db-engines predictive analytics today และ g2crowd [3], [34], [35] ซึ่งในแต่ละระบบฐานข้อมูลมีทั้งข้อดีและข้อเสียที่แตกต่างกัน จึงทำให้ช่วยในการตัดสินใจใช้ระบบฐานข้อมูล และทำให้ผู้ใช้งานสามารถเลือกใช้ระบบฐานข้อมูลที่มีประสิทธิภาพการทำงานได้เหมาะสมที่สุด และงานวิจัยนี้ได้วัดทั้ง เวลา CPU Memory และ Disk โดยงานวิจัยนี้ได้ใช้ทั้งหมด 4 ระบบฐานข้อมูลดังนี้ 1) Neo4j 2) OrientDB 3) GraphDB 4) ArangoDB

1.2 วัตถุประสงค์ของงานวิจัย

1) เพื่อวัดประสิทธิภาพของ Graph-based Database แต่ละระบบฐานข้อมูลในด้านต่าง ๆ ในรูปแบบกราฟเพื่อนำไปใช้ในการบริหารจัดการข้อมูลเชิงกราฟให้มีประสิทธิภาพมากที่สุด

2) เพื่อใช้เป็นแนวทางในการเลือกระบบฐานข้อมูล Graph-based NoSQL ให้เหมาะสมกับการใช้งานภายใต้สภาวะเชิงเดี่ยว

1.3 ขอบเขตของงานวิจัย

1) เราทดสอบประสิทธิภาพของระบบแบบเชิงเดี่ยวเท่านั้น

2) ข้อมูลที่ใช้การวัดประสิทธิภาพประกอบด้วย 2 รูปแบบ ได้แก่ non-graph และ graph ทั้งนี้ไม่รวมถึงข้อมูลเชิงพื้นที่ภูมิศาสตร์อื่น ๆ

1.4 ประโยชน์ที่คาดว่าจะได้รับจากงานวิจัย

1) ช่วยทำให้เลือกระบบฐานข้อมูล NoSQL ได้อย่างเหมาะสม และสอดคล้องกับรูปแบบการใช้

2) ช่วยวิเคราะห์และแนะนำฟังก์ชันอื่น ๆ ที่จำเป็นทางกราฟขั้นพื้นฐาน (ที่ยังไม่ถูกพัฒนาใน Graph-based NoSQL ปัจจุบัน)

บทที่ 2
หลักการและงานวิจัยที่เกี่ยวข้อง

การวิเคราะห์ประสิทธิภาพการทำงานของระบบฐานข้อมูล NoSQL ในครั้งนี้ ผู้วิจัยได้ศึกษารวบรวมหลักการและทฤษฎีต่าง ๆ ที่เกี่ยวข้องกับการวัดประสิทธิภาพของระบบฐานข้อมูล เพื่อนำมาประยุกต์ใช้ เพื่อเป็นแนวทางการวิจัยในครั้งนี้ โดยแบ่งเป็นหัวข้อต่าง ๆ ดังนี้

2.1 ระบบฐานข้อมูลแบบ NoSQL ที่นำมาใช้และทฤษฎีที่เกี่ยวข้องกับฐานข้อมูลแบบกระจาย

2.1.1 ทฤษฎีของระบบฐานข้อมูลแบบ NoSQL

2.1.2 ประเภทระบบฐานข้อมูลแบบ NoSQL

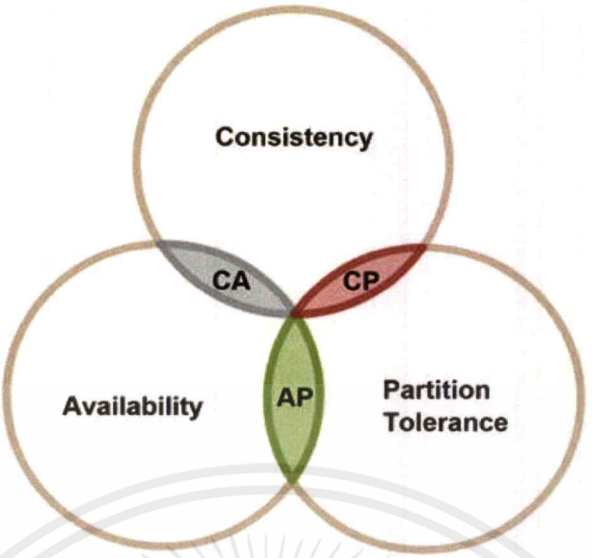
2.2 การทบทวนวรรณกรรมด้านเทคโนโลยี NoSQL

2.1 ระบบฐานข้อมูลแบบ NoSQL ที่นำมาใช้และทฤษฎีที่เกี่ยวข้องกับฐานข้อมูลแบบกระจาย

ระบบฐานข้อมูลแบบ NoSQL (“Not Only SQL”) คือ การเก็บข้อมูลในรูปแบบใหม่ที่ใช้เทคโนโลยี ที่แตกต่างจากการเก็บข้อมูลแบบเชิงสัมพันธ์ (Relational Databases) ซึ่งกำลังได้รับความนิยมมากในปัจจุบัน เพราะสามารถรองรับการใช้งานพร้อมกันได้หลายเครื่อง และรองรับการขยายตัวของเครื่อง (Horizontal scaling) ที่ใช้ในการประมวลผลในรูปแบบกระจายได้ (Distributed Processing)

2.1.1 ทฤษฎีของระบบฐานข้อมูลแบบ NoSQL

1. CAP ย่อมาจากคำว่า Consistency, Availability และ Partition Tolerance [4], [5], [39-40] ซึ่งสามารถรองรับคุณสมบัติเพียง 2 ใน 3 ของคุณสมบัติเท่านั้น (CAP Theorem) และสามารถรองรับการทำระบบแบบกระจายได้ คือ ไม่สามารถเลือกคุณสมบัติทั้งหมดได้ในเวลาเดียวกัน ซึ่งในแต่ละคุณสมบัติมีรายละเอียดดังนี้



รูปที่ 2.1 ทฤษฎี CAP

1.1) Consistency (C) : จะเก็บข้อมูลแบบเดียวกันไว้ในทุก ๆ ช่วงเวลา เมื่อมี user ทำการเรียกใช้งานข้อมูล ระบบจะทำการส่งข้อมูลไปยัง user โดยที่ version ของข้อมูล จะเป็น version ที่อัพเดตล่าสุดเสมอ และจะไม่ยอมให้ข้อมูล version เก่าอยู่ในระบบ user จะต้องทำการอัพเดตข้อมูลจนเสร็จ ระบบจึงสามารถ return ค่าที่เป็น version ล่าสุดออกมาได้ ดังนั้นจึงทำให้ทุกโหนดมีข้อมูลเป็นชุดเดียวกัน

1.2) Availability (A) : ในการเรียกใช้ข้อมูลจาก node ใดก็ตาม จะได้ค่าตอบรับเสมอ โดย การเข้าถึงข้อมูลของโหนดที่อยู่ใน cluster นั้น จึงทำให้ข้อมูลมีแค่สองสถานะ คือ version เก่ากับ version ใหม่ ซึ่งระบบที่มีคุณสมบัติ availability มักจะมีการตั้งค่าเกี่ยวกับ consistency แบบ weak (base) และข้อมูลที่ถูกกระจายเป็นส่วน ๆ อาจจะไม่ใช่ version ล่าสุดเสมอไป เพราะข้อมูลที่เป็น version ล่าสุดอาจเก็บอยู่ในโหนดใดโหนดหนึ่งใน cluster จึงต้องทำการกระจายข้อมูล version ล่าสุดกระจายไปยังโหนดต่าง ๆ ในภายหลัง

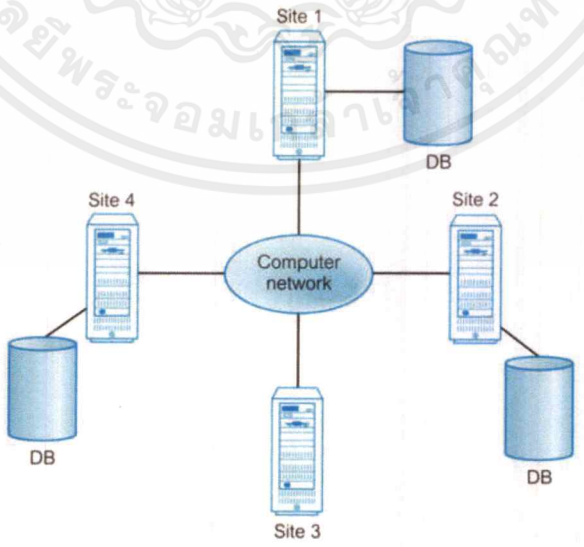
1.3) Partition Tolerance (P) : ระหว่างทำการเชื่อมต่อ แล้วเกิดเส้นทางการเชื่อมต่อระหว่าง node มีความเสียหาย ระบบจะยังคงทำงานต่อไป และ client ยังเข้าถึงได้อยู่ ถ้าระบบสามารถทนต่อสภาพเช่นนี้ได้ ตัวฐานข้อมูลก็จะสามารถทำงานอ่านและเขียนได้ตามปกติ ขณะที่ฐานข้อมูล 2 rack แยกออกจากกัน คุณสมบัติข้อนี้เป็นที่ต้องการระบบฐานข้อมูลแบบกระจายโดยทั่วไป

1.4) CA คือ Consistency และ Availability แต่ไม่ใช้ Partition Tolerance : ระบบจะมีความรวดเร็ว และข้อมูลจะมีความถูกต้อง เมื่อมีการเปลี่ยนแปลงข้อมูล แต่ถ้าระบบเครือข่ายเกิดความขัดข้องจะทำให้ทั้งระบบใช้งานไม่ได้

1.5) CP คือ Consistency และ Partition tolerance แต่ไม่ใช้ Availability : เมื่อมีโหนดใดโหนดหนึ่ง fail ระบบโดยรวมจะทำงานได้อยู่ แต่จะไม่สามารถทำงานอย่างเต็มประสิทธิภาพ เพราะระบบจะหยุดทำงานจนกว่าจะมีการแก้ไขระบบนั้นจนสมบูรณ์ และบาง software ที่ทำการอัพเดท ผู้ใช้งานต้องอัพเดทข้อมูลก่อน จึงสามารถใช้งานได้ตามปกติ

1.6) AP คือ Availability และ Partition tolerance แต่ไม่ใช้ Consistency : เมื่อบางโหนดที่เชื่อมต่อกันอยู่เกิดมีความเสียหาย ผู้ใช้งานบางส่วน จะไม่สามารถทำงานอย่างเต็มประสิทธิภาพ เช่น ระบบ ที่สามารถอ่านและเขียนได้ เมื่อมีบางโหนดเกิดความเสียหาย ระบบนั้นจะทำได้เพียงอ่านข้อมูลได้เพียงอย่างเดียว

2. ระบบฐานข้อมูลแบบกระจาย (Distributed Database) [14] เป็นระบบที่กระจายฐานข้อมูลไปยังตำแหน่งต่าง ๆ โดยจัดเก็บข้อมูลไว้ในแต่ละเครื่อง เพื่อระหว่างการเชื่อมต่อกันในระบบ และใช้เครือข่ายอินเทอร์เน็ต ซึ่งทำให้ระบบมีความพร้อมที่จะทำการกระจายตัวออกไปยังหลาย ๆ เครื่อง อีกทั้งยังสามารถทำงานได้พร้อม ๆ กัน ประกอบไปด้วย กลุ่มของระบบฐานข้อมูลย่อยที่ติดต่อสื่อสารกัน โดยผ่านเครือข่ายสื่อสารในแต่ละระบบฐานข้อมูลแบบย่อย อีกทั้งสามารถทำงานได้ด้วยตนเอง และสามารถให้ user จากแหล่งอื่น ๆ เข้ามาใช้ข้อมูลในฐานข้อมูลนี้ได้



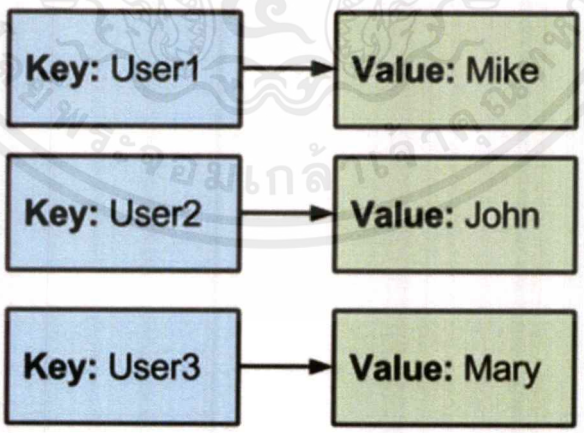
รูปที่ 2.2 ตัวอย่างโครงสร้างของ Distributed DBMS

การประมวลผลแบบกระจาย (Distributed Processing) โดยมีการกระจายหน้าที่การประมวลผล ไปยังเครื่องต่าง ๆ ที่เชื่อมต่อกันเป็นเครือข่ายคอมพิวเตอร์ และนำผลลัพธ์ที่ได้จากการประมวลผลมารวมกัน ซึ่งวิธีนี้ทำให้เพิ่มประสิทธิภาพในการประมวลผลของระบบได้โดยรวม ซึ่งสามารถลดจำนวนข้อมูลที่ส่งผ่านทางเครือข่ายได้ นอกจากนี้ยังเป็นระบบที่ประมวลผลข้อมูล รวมไปถึงประมวลผลทรัพยากรของคอมพิวเตอร์ แบบกระจายอยู่มากกว่าหนึ่งที่ และมีการเชื่อมโยงเป็นเครือข่ายเดียวกัน ในการทำงานต่าง ๆ จะถูกประมวลผลด้วยเครื่องคอมพิวเตอร์หลาย ๆ เครื่อง และประเภทของสื่อกลางสามารถแยกได้เป็น 2 ประเภท คือ แบบมีสาย และแบบไร้สาย ในการประมวลผลแบบกระจายมีหลายชนิด และมีแนวโน้มจะมากขึ้นเรื่อย ๆ

2.1.2 ประเภทระบบฐานข้อมูลแบบ NoSQL

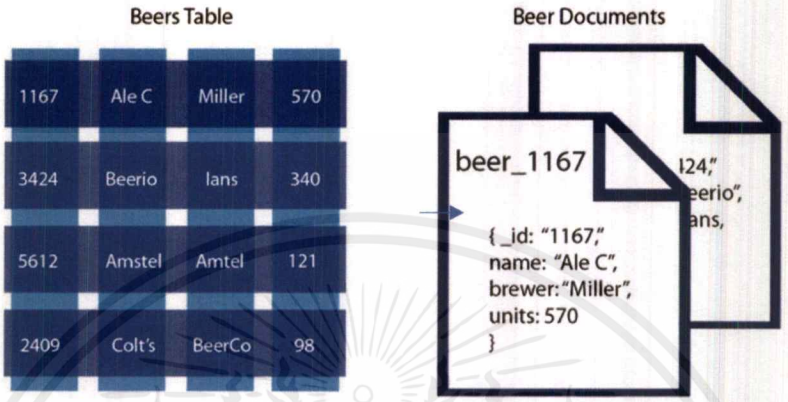
ระบบฐานข้อมูลแบบ NoSQL ในปัจจุบันมีการจัดเก็บข้อมูลหลากหลายชนิด จึงทำให้มีการจัดการฐานข้อมูลหลายแบบ ซึ่งสามารถแบ่งเป็น 4 ประเภท ดังนี้

1) Key-value based คือ ฐานข้อมูลที่เก็บข้อมูลในรูปของคู่อันดับ โดยจะเรียกว่า key และ value ที่มีความสัมพันธ์กันอยู่ ซึ่ง value จะถูกค้นหาโดยใช้ค่า key ตัวอย่างเช่น Redis, Dynamo และ Voldemort เป็นต้น



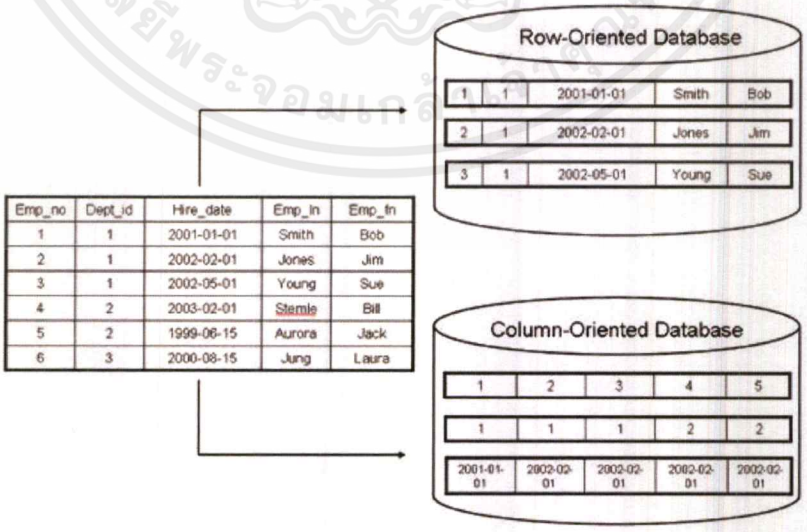
รูปที่ 2.3 การเก็บข้อมูลของ key-value based NoSQL

2) Document-based คือ ฐานข้อมูลที่เก็บข้อมูลในรูปของกลุ่มของเอกสาร (JSON) ซึ่งง่ายต่อการแก้ไขโครงสร้าง โดยแต่ละเอกสารจะมีจำนวน field ที่แตกต่างกัน และเก็บในรูปของตัวอักษร ตัวอย่างเช่น Couch DB และ MongoDB เป็นต้น



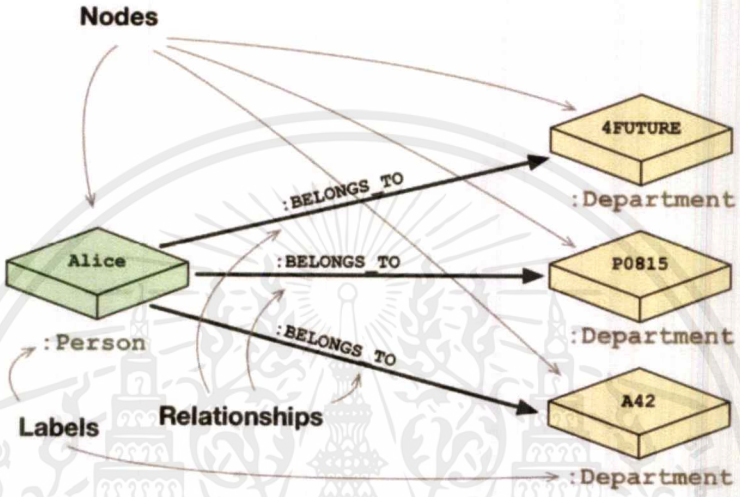
รูปที่ 2.4 การเก็บข้อมูลของ document-based NoSQL

3) Column-based (or wide-column) คือ ฐานข้อมูลที่เก็บข้อมูลในรูปของตาราง โดยจะคล้ายกับฐานข้อมูลเชิงสัมพันธ์ แต่จะเก็บข้อมูลในแบบคอลัมน์แทนแบบแถว ซึ่งจะแบ่งแถวออกเป็นส่วน ๆ ทำให้การค้นหาทำได้เร็วกว่าการค้นหาแบบทั้งหมด ตัวอย่างเช่น HBase, Cassandra และ Big table เป็นต้น



รูปที่ 2.5 การเก็บข้อมูลของ column-based NoSQL

4) Graph-based คือ ฐานข้อมูลที่เก็บข้อมูลในรูปของกราฟ โดยมีความสัมพันธ์กัน ทำให้สามารถ ใช้ทฤษฎีกราฟในการจัดการกับข้อมูลได้ ซึ่งถูกออกแบบมาสำหรับข้อมูลที่ต้องแสดงเป็นกราฟ และต้องมีเส้นเชื่อมไปยังโหนดอื่น เพื่อบอกความสัมพันธ์ เช่น social relations, link ของการขนส่ง ถนน แผนที่ หรือระบบ network ตัวอย่างเช่น Neo4j และ Orient DB เป็นต้น



รูปที่ 2.6 การเก็บข้อมูลของ graph-based NoSQL

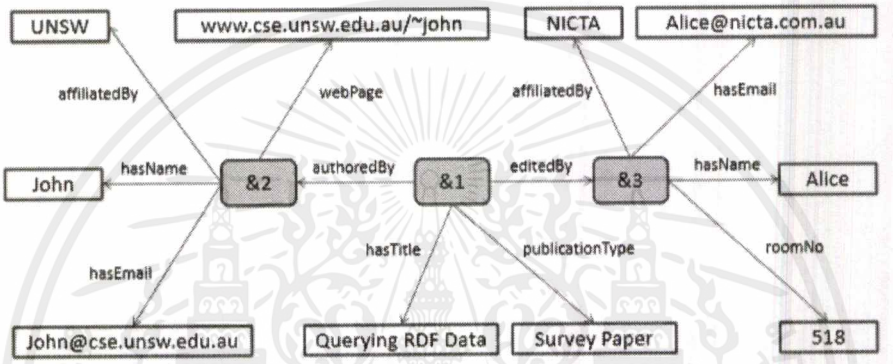
จากรูปที่ 2.6 เป็นโครงสร้างข้อมูลแบบกราฟ ประกอบไปด้วย

- Node คือ ข้อมูลหรือ entity ใด ๆ เช่น ใน Social Network คือ user

- Edge คือ ความสัมพันธ์ระหว่าง entity ซึ่งแสดงอยู่ในรูปของเส้น และมีคุณสมบัติต่าง ๆ อยู่ด้วย รวมทั้งยังมีทิศทาง หรือ direction อีกด้วย เช่น ในการใช้งานด้าน Social Network โดยการใช้ความสัมพันธ์

2.2 การทบทวนวรรณกรรมด้านเทคโนโลยี NoSQL

งานวิจัยที่ 1 จัดทำโดย Sherif Sakr และคณะผู้วิจัย (2009) ได้ทำการวิจัยเรื่อง "Relational Processing of RDF Queries: A Survey" [2] งานวิจัยนี้ได้กล่าวถึงรูปแบบที่ใช้เก็บข้อมูล RDF ที่ได้รับ ความนิยมจากนักวิจัยต่าง ๆ ทั้งในด้านการ queries และการเก็บข้อมูล ซึ่งในงานวิจัยนี้ได้สอนวิธีการจัดการ RDF เพื่อปรับรูปแบบให้มีความยืดหยุ่นและนำไปใช้กับ Semantic Web รูปแบบตัวอย่างของกราฟ RDF ที่ใช้งานกับ Semantic Web แบบทั่วไป ดังรูปที่ 2.7



รูปที่ 2.7 ตัวอย่าง RDF graph

โครงสร้างพื้นฐานของ RDF เป็นแบบ tuple ประกอบไปด้วย 3 ส่วน คือ subject, predicate และ object ซึ่งโครงสร้างการเก็บ RDF แบ่งได้ทั้งหมด 3 ประเภท ได้แก่

1) รูปแบบแนวตั้ง (Vertical) หรือแบบ triple จะเก็บในรูปของตารางที่มีทั้งหมด 3 คอลัมน์ โดยจะแบ่งการเก็บข้อมูลในคอลัมน์ตาม subject, predicate และ object ซึ่งลักษณะของตารางเป็นแนวตั้ง ดังรูปที่ 2.8

Subject	Predicate	Object
Id1	publicationType	Survey Paper
Id1	hasTitle	Querying RDF Data
Id1	authoredBy	Id2
Id2	hasName	John
Id2	affiliatedBy	UNSW
Id2	hasEmail	John@cse.unsw.edu.au
Id2	webPage	www.cse.unsw.edu.au/~john
Id1	editedBy	Id3
Id3	hasName	Alice
Id3	affiliatedBy	NICTA
Id3	hasEmail	Alice@nicta.com.au
Id3	roomNo	518

```
Select T3.Object
From Triples as T1, Triples as T2,
     Triples as T3, Triples as T4
Where
T1.Predicate="publicationType" and
T1.Object="Survey Paper"
and T2.predicate="hasTitle"
and T2.Object="Querying RDF Data"
and T3.Predicate="webPage"
and T1.subject=T2.subject
and T4.subject=T1.subject
and T4.Predicate="authoredBy"
and T4.Object = T3.Subject
```

รูปที่ 2.8 ตัวอย่างตารางแนวตั้ง (Triple)

จากรูปที่ 2.8 ได้ใช้ระบบการเก็บข้อมูลของ RDF โดยทำในรูปแบบ hash ทั้ง subject, predicate และ object ซึ่งในการทำ hash จะสร้างตารางการเก็บสัญลักษณ์เพิ่มขึ้นมา เพื่อทำการเรียกใช้งาน และใช้สัญลักษณ์ในการค้นหาผลลัพธ์ โดยใช้ queries ภาษา SPARQL เพื่อ join ตาราง triple และทำการดำเนินการบนกราฟ ในการหาค่าผ่านความสัมพันธ์บนเส้นทางที่ใช้เชื่อมกันระหว่างกราฟ ซึ่งสามารถเพิ่มประสิทธิภาพในการ queries ได้ และใช้ index ทั้ง 3 คอลัมน์ เพื่อให้มีการเรียงค่าใหม่ และสามารถค้นหาได้รวดเร็วยิ่งขึ้น

2) รูปแบบคุณสมบัติ (Property) หรือแบบ n-ary รูปแบบนี้จะใช้ในการแก้ไขปัญหาของรูปแบบ triple โดยในรูปแบบนี้เก็บค่าต่างกับแบบ triple ซึ่งจะเก็บทั้งหมด 4 คอลัมน์ ได้แก่ Subject, Predicate, ObjectURI และ ObjectLiteral เพื่อเก็บคำสั่งต่าง ๆ และการอ้างอิงต่าง ๆ ให้มีรูปแบบคล้ายกับ triple และสามารถแปลงให้อยู่ในรูปแบบ URI ดังรูปที่ 2.9 ในรูปตัวอย่างจะไม่เก็บค่าในคอลัมน์ ID

Publication

ID	publicationType	hasTitle	authoredBy	editedBy
Id1	Survey Paper	Querying RDF Data	Id2	Id3

Person

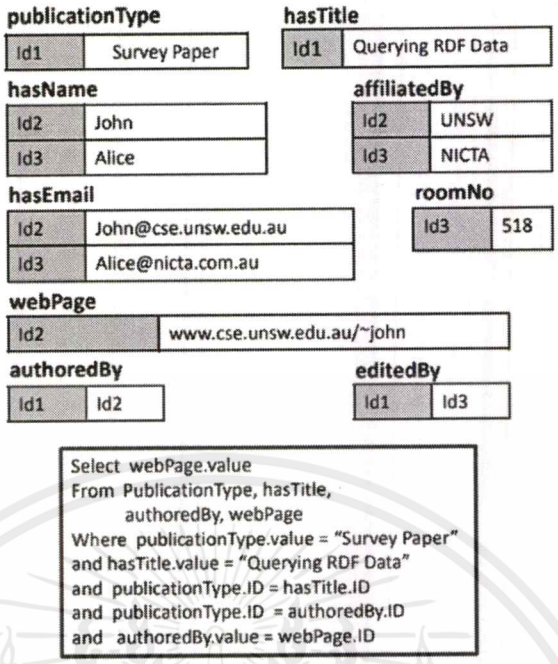
ID	hasName	affiliatedBy	hasEmail	webPage	roomNo
Id2	John	UNSW	John@cse.unsw.edu.au	www.cse.unsw.edu.au/~john	
Id3	Alice	NICTA	Alice@nicta.com.au		518

```
Select Person.webPage
From Person, Publication
Where Publication.publicationType = "Survey Paper"
and Publication.hasTitle = "Querying RDF Data"
and Publication.authoredBy = Person.ID
```

รูปที่ 2.9 ตัวอย่างการเก็บคุณสมบัติในตาราง (property)

โดยเป้าหมายหลักของการใช้วิธีดังกล่าว คือ ลดจำนวนเส้นเชื่อม predicate ในการ join ระหว่างตาราง และลดการใช้หน่วยความจำในการเก็บค่า ซึ่งวิธีดังกล่าวจะเกี่ยวข้องกับการจัดกลุ่มและการแบ่ง partition ในการจัดกลุ่มเก็บข้อมูล โดย predicate เหมือนกันไว้ที่เดียวกัน predicate ที่ค่าต่างกันจะเก็บไว้ในตาราง binary ดังรูปที่ 2.10 และระยะการแบ่ง partition จะสร้างความสมดุลในการจัดเก็บแต่ละ predicate โดยการเก็บค่าสตริงให้เหลือที่ว่างให้น้อยที่สุด ซึ่งในระหว่างการแบ่ง partition อาจมีปัญหาการทับซ้อนกันระหว่างกลุ่มในแต่ละ predicate และการทำวิธีดังกล่าวจะเหมาะกับค่าในตาราง predicate และตาราง binary ที่มีค่าใกล้เคียงกัน จึงทำให้มีประสิทธิภาพมากยิ่งขึ้น

3) รูปแบบแนวนอน (Horizontal) หรือแบบ binary จะเก็บข้อมูลเป็นแต่ละตาราง predicate และมี 2 คอลัมน์ดังรูปที่ 2.10

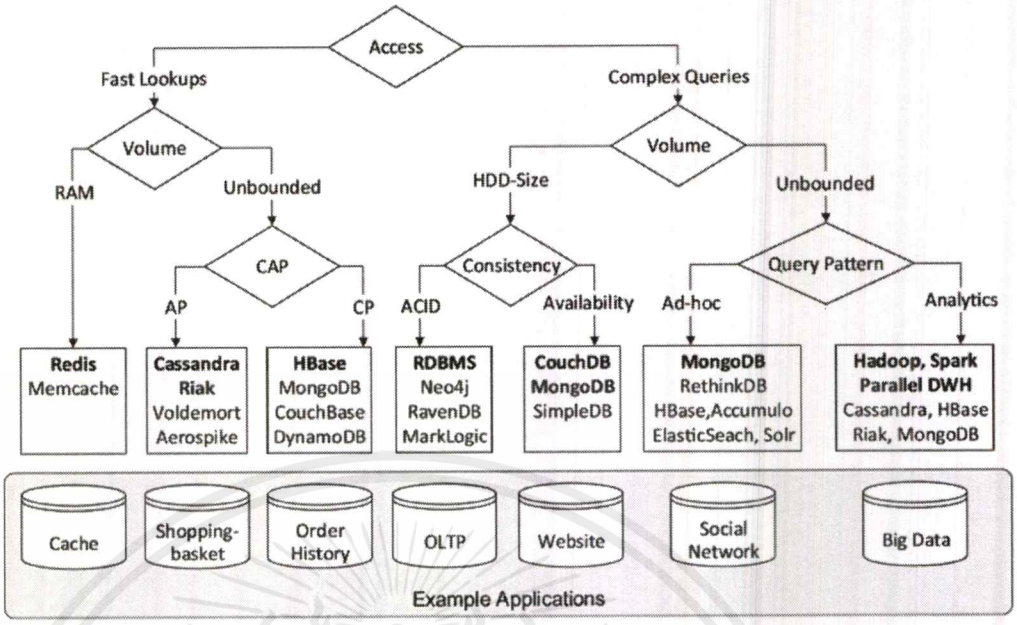


รูปที่ 2.10 ตัวอย่างตารางแนวนอน (Binary)

ในรูปแบบดังกล่าว ตารางจะแบ่งเก็บเป็น 2 ส่วน คือ ส่วนแรกจะเก็บข้อมูล subject และส่วนหลังจะเก็บข้อมูล object โดยการจัดเก็บประเภทนี้จะทำให้มีการค้นหาข้อมูลได้อย่างรวดเร็ว และสามารถทำการ join ได้อย่างรวดเร็ว เนื่องจากข้อมูลในแต่ละตารางจะมีจำนวนไม่มาก ซึ่งในการจัดเก็บข้อมูลวิธีดังกล่าวจะสามารถปรับเปลี่ยนรูปแบบการทำ queries ได้อย่างไม่จำกัด และสามารถทำการ union ได้หลายชั้น แต่ปัญหาที่เกิดขึ้นสำหรับวิธีดังกล่าว คือ เนื่องจากในการค้นหาข้อมูลในแต่ละครั้งจะต้องทำการโหลดข้อมูลลงหน่วยความจำก่อน จึงทำให้การเก็บข้อมูลในรูปแบบนี้จะมีความยากในการอ่านข้อมูลระหว่างการทำ queries

งานวิจัยที่ 2 จัดทำโดย Felix Gessert และคณะผู้วิจัย (2016) ได้ทำการวิจัยเรื่อง "NoSQL database systems: a survey and decision guidance" [10] โดยวิจัยนี้ได้กล่าวถึงการจัดการข้อมูลขนาดใหญ่ด้วยระบบฐานข้อมูลแบบ NoSQL ที่สามารถรองรับปริมาณข้อมูลที่เพิ่มมากขึ้นหลายเท่าตัว และการเรียกใช้งานที่หลากหลาย จึงจำเป็นต้องเลือกประเภทที่จัดเก็บข้อมูลให้เหมาะสมกับแอพพลิเคชัน ซึ่งความแตกต่างของระบบฐานข้อมูลทั้ง 3 แบบได้แก่ key-value store ,document store และ wide-column store รวมถึงต้องนำ CAP theorem และคุณสมบัติ BASE มาพิจารณาในการเลือกระบบฐานข้อมูลให้สอดคล้องกับความต้องการของระบบ

การตัดสินใจเลือกใช้ระบบฐานข้อมูลที่เหมาะสมขึ้นอยู่กับรูปแบบการเข้าถึงของแอพพลิเคชัน โดยระบบโหนดเดียวจะเหมาะสมที่สุดก็ต่อเมื่อหน่วยความจำหลักของเครื่องเครื่องเดียวสามารถเก็บข้อมูลได้ทั้งหมด เช่น Redis , Memcache ถ้าจำนวนข้อมูลมากเกินความจุของ RAM การเลือกจำนวนโหนดแบบ Horizontal จะเหมาะสมกว่า และตัวเลือกที่สำคัญที่สุดคือ การเข้าถึงระบบที่รวดเร็ว (AP) กับความสอดคล้องของข้อมูล (CP) ตามหลัก CAP theorem เช่น Cassandra และ Riak จะเหมาะกับระบบที่ต้องการ AP ในขณะที่ระบบ HBase, MongoDB, DynamoDB จะให้ความถูกต้องและตรงกันของข้อมูลหรือค่า Latency คงที่ (CP) เมื่อทำงานบนไดร์ฟข้อมูลขนาดปานกลาง ระบบฐานข้อมูลแบบ RDBMS ดั้งเดิมหรือฐานข้อมูลกราฟ เช่น Neo4j จะดีที่สุด อย่างไรก็ตาม MongoDB และ CouchDB เหมาะสมกับความพร้อมใช้งานบนระบบแบบกระจาย เช่น แอพพลิเคชันเครือข่ายสังคม สำหรับระบบ HBase และ Cassandra เหมาะสำหรับการทำงานกับข้อมูลขนาดใหญ่ และเมื่อรวมกับ Hadoop จะมีความสามารถมากกว่าการขยายแบบแนวตั้ง (vertical scaling) นอกจากนี้ยังมีรายละเอียดสำหรับการเลือกใช้งานระบบฐานข้อมูล NoSQL ตามต้นไม้ตัดสินใจด้านล่าง ดังรูปที่ 2.11



รูปที่ 2.11 ต้นไม้ตัดสินใจจากงานวิจัย

งานวิจัยนี้กล่าวถึงการเลือกใช้งานระบบฐานข้อมูลให้เหมาะสมกับแอพพลิเคชัน เนื่องจากปัจจุบันมีข้อมูลขนาดใหญ่ขึ้น และการเรียกใช้งานที่สูงขึ้น จึงจำเป็นอย่างยิ่งที่จะต้องเลือกประเภทของระบบฐานข้อมูล เพื่อรองรับข้อมูล และการเรียกใช้งาน ซึ่งส่งผลให้แอพพลิเคชันมีประสิทธิภาพดีที่สุด

งานวิจัยที่ 3 จัดทำโดย Alejandro Corbellini และคณะผู้วิจัย (2016) ได้ทำการวิจัยเรื่อง "Persisting big-data : The NoSQL landscape" [1] โดยการทำงานของฐานข้อมูล NoSQL มีทั้งหมด 4 ประเภท คือ 1. Key-Value 2. Document-oriented 3. Wide-Column databases 4. Graph-oriented โดยจะกล่าวถึง คุณสมบัติในการทำงานขั้นพื้นฐานของแต่ละประเภท โดยมีทั้งหมด 19 ระบบฐานข้อมูล ในงานวิจัยนี้จะกล่าวถึง คุณสมบัติของฐานข้อมูลแบบ NoSQL ทั้งหมด 7 ประเภท คือ (1) Persistence (วิธีการจัดเก็บข้อมูล), (2) Replication (การกระจายการทำงาน), (3) Sharding (การแบ่งข้อมูลออกให้เท่าๆกัน), (4) Consistency (การอ่านและเขียน), (5) Implementation language (ภาษาที่ใช้ในการค้นหาข้อมูล), (6) API (ประเภทของ UI ในการเข้าถึงฐานข้อมูล) และ (7) Query method โดยจะเปรียบเทียบแต่ละประเภทดังนี้

Grouping of NoSQL systems grouped by data layout and CAP properties.

Data layout	AP	CP	AC
Key-Value (Section 4)	Riak, Infinispan, Redis, Voldemort, Hazelcast	Infinispan, Membase/CouchBase, BerkeleyDB, GT.M	Infinispan
Wide Column (Section 5)	Cassandra	HBase, Hypertable	-
Document-oriented (Section 6)	MongoDB, RavenDB , CouchDB, Terrastore	MongoDB	-
Graph-oriented (Section 7)	Neo4J, HypergraphDB, BigData, AllegroGraph, InfoGrid, InfiniteGraph	InfiniteGraph	-

รูปที่ 2.12 ระบบฐานข้อมูลที่นำมาใช้ในงานวิจัย

1) Key-Value จะประกอบไปด้วย 6 ระบบฐานข้อมูล คือ Riak, Infinispan, Hazelcast, Redis, CouchBase และ Voldemort โดยคุณสมบัติ Persistence ของแต่ละระบบฐานข้อมูลจะมีความแตกต่างกัน คุณสมบัติ Replication ระบบฐานข้อมูล Redis จะทำเป็นแบบ Master-Slave และระบบฐานข้อมูล CouchBase จะทำเป็นแบบ vBuckets 1:N ซึ่งระบบฐานข้อมูลอื่น ๆ จะทำแบบ Ring (next N-1) ทั้งหมด คุณสมบัติ Sharding ระบบฐานข้อมูล Redis จะไม่มีคุณสมบัตินี้ และระบบฐานข้อมูล CouchBase จะทำเป็นแบบ vBuckets ซึ่งระบบฐานข้อมูลอื่นจะทำเป็นแบบ Consistent Hashing ทั้งหมด คุณสมบัติ Consistency ระบบฐานข้อมูลที่มีคุณสมบัติเป็น Strong Consistency มีทั้งหมด 3 ระบบฐานข้อมูล คือ Infinispan, Hazelcast และ CouchBase ส่วนระบบฐานข้อมูลอื่น ๆ ที่มีคุณสมบัติเป็น Eventual Consistency ยกเว้นระบบฐานข้อมูล Infinispan ที่มีทั้ง 2 คุณสมบัติ คุณสมบัติ Implementation language ระบบฐานข้อมูล Riak และ CouchBase จะใช้ภาษา Erlang ส่วนระบบฐานข้อมูล Redis และ CouchBase จะใช้ภาษา C ซึ่งระบบฐานข้อมูลอื่นจะใช้ภาษา Java ทั้งหมด คุณสมบัติ API ระบบฐานข้อมูลทั้งหมด สามารถใช้งาน Java API ได้ทั้งหมด และคุณสมบัติ Query Method ระบบฐานข้อมูล Riak Infinispan และ Hazelcast สามารถรองรับการทำ MapReduce ได้

Name	Persistence	Replication	Sharding	Consistency	Implementation language	API	Query method
Riak	Bitcask (log-structured store), LevelDB, In-Memory and Multi-backend (different stores for different keys)	Ring (next $N-1$)	Consistent Hashing	Eventual Consistency	Erlang	PBC (Protocol Buffer Client), HTTP, Java, Erlang, C++, PHP, Ruby, Python	Get, MapReduce, Link Walking
Infinispan	Simple File Storage, BerkeleyDB, JDBM, JDBC	Ring (next $N-1$)	Consistent Hashing	Strong Consistency or Eventual Consistency	Java	HTTP, Java	Get, MapReduce, others
Hazelcast	User-defined MapStore, which can be persistent	Ring (next $N-1$)	Consistent Hashing	Strong Consistency	Java	HTTP, Java, C# and any Memcache client	Get, MapReduce
Redis	Snapshots at specified intervals by default or an Append-only file. Both can be combined	Master-Slave (Slave chains can be formed)	No (in charge of the application)	Eventual Consistency	C	Java, C, C#, Ruby, Perl, Scala	Get (also depends on the value structure)
Membase/ CouchBase	SQLite or CouchDB	vBuckets 1: N Replication	vBuckets	Strong Consistency	C/C++, Erlang	Java, C, C#	Get
Voldemort	BerkeleyDB, In-Memory, MySQL	Ring (next $N-1$)	Consistent Hashing	Eventual Consistency	Java	Java, Python	Get

รูปที่ 2.13 การเปรียบเทียบระหว่างระบบฐานข้อมูล key-value

2) Document-oriented จะประกอบไปด้วย 6 ระบบฐานข้อมูล คือ CouchDB, MongoDB, Terrastore และ RavenDB โดยคุณสมบัติ Persistence ของแต่ละระบบฐานข้อมูลจะมีความแตกต่างกัน คุณสมบัติ Replication ระบบฐานข้อมูล CouchDB จะทำเป็นแบบ Master-Master ซึ่งระบบฐานข้อมูลอื่น ๆ จะทำเป็นแบบ Master-Slave ทั้งหมด คุณสมบัติ Sharding ของแต่ละระบบฐานข้อมูลจะมีความแตกต่างกัน คุณสมบัติ Consistency ระบบฐานข้อมูลทั้งหมดจะมีคุณสมบัติเป็น Eventual Consistency ทั้งหมด คุณสมบัติ Implementation language ของแต่ละระบบฐานข้อมูลจะมีความแตกต่างกัน คุณสมบัติ API ระบบฐานข้อมูลทั้งหมดใช้ HTTP + JSON ทั้งหมด และคุณสมบัติ Query Method ระบบฐานข้อมูล CouchDB Terrastore และ MongoDB สามารถรองรับการทำ MapReduce ได้

Name	Persistence	Replication	Sharding	Consistency	Implementation language	API	Query method
CouchDB	CouchDB Storage Engine (B-Tree)	Master-Master	No, but there are extensions to CouchDB that allow sharding	Eventual Consistency	Erlang	HTTP + JSON, and clients for different languages (including Java)	Views using JavaScript + MapReduce
MongoDB	BSON Objects or GridFS for big files	Replica Sets (sets of Master-Slaves) or simply Master-Slave	By field, which can be any field in a document collection	Strict Consistency by default, but can be relaxed to Eventual Consistency	C++	Mongo Wire Protocol + BSON, HTTP + JSON, and clients for most languages	Queries per field, Cursors and MapReduce
Terrastore	Terrastore storing support	Master-Slave with N replicas in hot-standby	Consistent Hashing	Eventual Consistency	Java	HTTP + JSON, and clients for some languages (Java, Clojure [31], Scala [84])	Conditional queries, queries by range Predicates, MapReduce
RavenDB	Microsoft's ESE (Extensible Storage Engine)	Master-Slave on N-replicas	Allows the user to define a sharding function based on the documents' fields	Eventual Consistency	C#	HTTP + JSON, .Net	LINQ

รูปที่ 2.14 การเปรียบเทียบระหว่างระบบฐานข้อมูล Document-oriented

3) Wide-Column databases จะประกอบไปด้วย 3 ระบบฐานข้อมูลคือ HBase, Hypertable และ Cassandra โดยคุณสมบัติ Persistence ของระบบฐานข้อมูล HBase และ Hypertable จะเป็น HDFS ส่วนระบบฐานข้อมูล Cassandra จะเป็น Proprietary format,

คุณสมบัติ Replication ระบบฐานข้อมูล HBase และ Hypertable จะเป็น HDFS ส่วนระบบฐานข้อมูล Cassandra จะเป็น Ring (next N-1) คุณสมบัติ Sharding ของระบบฐานข้อมูล HBase และ Hypertable จะขึ้นกับจำนวน key ส่วนระบบฐานข้อมูล Cassandra จะเป็น Consistent Hasing คุณสมบัติ Consistency ของระบบฐานข้อมูล HBase และ Hypertable มีคุณสมบัติเป็น Strong Consistency ส่วนระบบฐานข้อมูล Cassandra จะเป็น Eventual Consistency คุณสมบัติ Implementation language ของระบบฐานข้อมูล HBase และ Cassandra จะเป็น Java ส่วนระบบฐานข้อมูล Hypertable จะเป็น C++ คุณสมบัติ API สามารถใช้งาน Thrift ได้ทั้งหมด และคุณสมบัติ Query Method ระบบฐานข้อมูลทั้งหมดสามารถ ทำ MapReduce Pig, Hive และ Hadoop ได้ทั้งหมด

Name	Persistence	Replication	Sharding	Consistency	Implementation language	API	Query method
HBase	HDFS (Hadoop File System)	HDFS replication	By key ranges	Strong Consistency	Java	Java, HTTP + JSON, Avro, Thrift [102]	Hadoop MapReduce, Pig, Hive
Hypertable	HDFS by default (other supports are available)	HDFS replication	By key ranges	Strong Consistency	C++	Thrift	HQL (Hypertable Query Language), Hadoop MapReduce, Hive, Pig
Cassandra	Proprietary format	Ring (next N-1)	Consistent Hashing	Eventual Consistency	Java	Thrift	CQL (Cassandra Query Language), Hadoop MapReduce, Pig, Hive

รูปที่ 2.15 การเปรียบเทียบระหว่างระบบฐานข้อมูล Wide-Column

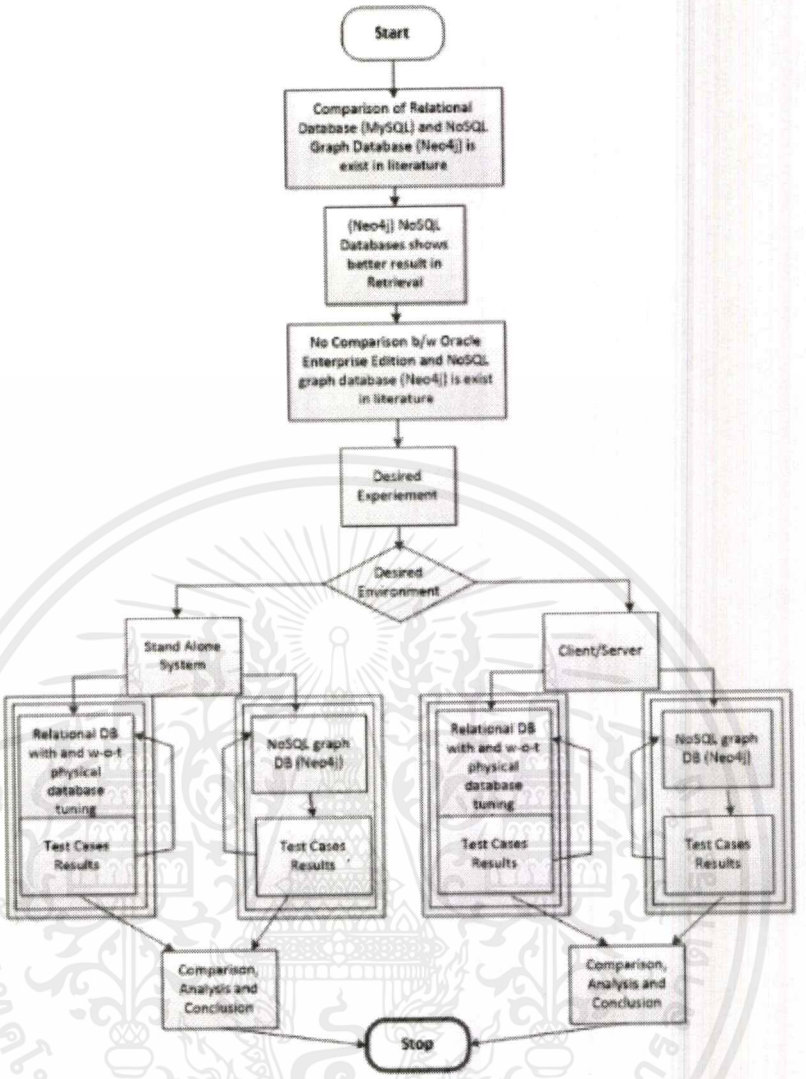
4) Graph-oriented ประกอบไปด้วยระบบฐานข้อมูลทั้งหมด 6 ระบบฐานข้อมูล คือ Neo4j, InfiniteGraph,InfoGrid, HypergraphDB, BigData, AllegroGraph โดยคุณสมบัติ Persistence ของระบบฐานข้อมูล Neo4j, BigData และ AllegroGraph จะเป็น Indexes ซึ่งระบบฐานข้อมูลอื่นจะแตกต่างกันทั้งหมด, คุณสมบัติ Replication ของระบบฐานข้อมูล Neo4j, BigData และ AllegroGraph จะเป็น Master-Slave ส่วนระบบฐานข้อมูล InfoGrid และ HypergraphDB จะเป็น Peer-to-peer, คุณสมบัติ Sharding ของระบบฐานข้อมูล Neo4j, InfoGrid, HepergraphDB และ AllegroGraph จะเป็นแบบ Manual, คุณสมบัติ Consistency ระบบฐานข้อมูลทั้งหมดสามารถเป็นแบบ Eventual Consistency, คุณสมบัติImplementation language ของระบบฐานข้อมูล Neo4j, InfoGrid, HypergraphDB และ BigData, คุณสมบัติ API ระบบฐานข้อมูล Neo4j, InfiniteGraph, InfoGrid, HypergraphDB และ BigData จะเป็น Java API และระบบฐานข้อมูล Neo4j, InfiniteGraph, InfoGrid และ AllegroGraph จะเป็น HTTP + JSON และคุณสมบัติ Query Method ของระบบฐานข้อมูล Neo4j, InfoGrid, HypergraphDB จะรองรับ Java API และระบบ

ฐานข้อมูล Neo4j, HypergraphDB, BigData, AllegroGraph จะเป็น SPARQL ซึ่งจะรองรับการจัดเก็บแบบ RDF เช่นกัน

Name	Persistence	Replication	Sharding	Consistency	Implementation language	API	Query method
Neo4J	Indexes on disk (Apache Lucene by default)	Master-Slave	Manual	Eventual Consistency	Java	Java, HTTP + JSON, bindings in Ruby, Clojure, Python, among others	SPARQL (RDF and OWL), Java API, Gremlin
InfiniteGraph	Objectivity/DB	Synchronous replication of Objectivity/DB	Rule based sharding	Strong Consistency or Eventual Consistency	C++	C++, Java, Python and C#	API for graph traversing and Predicate Queries
InfoGrid	MySQL, HadoopFS, among others	Peer-to-Peer (XPRISO protocol)	Manual	Eventual Consistency	Java	Java, HTTP + JSON	Viewlets, Templates HTML, Java
HypergraphDB	2 tiers: Primitive (Raw Data) and Model (relations + caching + indexes)	Peer-to-Peer (Agent-based)	Manual	Eventual consistency	Java	Java	Java API, Prolog, OWL, RDF via Sesame
BigData	Indexes (B+ trees)	Master-Slave	"Dynamic" sharding (by key-range shards, also called index partitions)	Eventual Consistency	Java	Java and service discovery through JNI	SPARQL, RDFS++
AllegroGraph	Indexes	Master-Slave (Warm Standby)	Manual	Eventual Consistency	Lisp	HTTP + JSON and clients in several languages (included Java)	SPARQL, Prolog, RDFS++, and graph traversal through API

รูปที่ 2.16 การเปรียบเทียบระหว่างระบบฐานข้อมูล Graph-oriented

งานวิจัยที่ 4 จัดทำโดย Wisal Khan และคณะผู้วิจัย (2017) ได้ทำการวิจัยเรื่อง "Predictive Performance Comparsion Analysis of Relational & NoSQL Graph Databases" [36] ที่เกี่ยวข้องกับการวัดประสิทธิภาพของฐานข้อมูลเชิงสัมพันธ์ (Oracle) และฐานข้อมูลเชิงกราฟ (Neo4j) โดยหลายองค์กรได้ใช้ฐานข้อมูลแบบเดิมในการจัดการและวิเคราะห์ข้อมูลที่มีโครงสร้าง ในฐานข้อมูลเชิงสัมพันธ์ต้องใช้ open source เช่น SQL เพื่อจัดการกับข้อมูลที่มีโครงสร้างและรูปแบบของข้อมูล เพื่อประมวลผลข้อมูลได้ตามขีดจำกัดที่กำหนด ในการจัดการชุดข้อมูลขนาดใหญ่ โดยใช้ฐานข้อมูลเชิงสัมพันธ์ องค์กรจำเป็นต้องเพิ่มขีดความสามารถของระบบเช่น RAM, Disk เพื่อจัดการความสมบูรณ์ของคุณสมบัติ ACID (Atomicity, Consistency, Isolation Durability) แต่ในปัจจุบันองค์กรหลายแห่งมีข้อมูลที่ไม่มีโครงสร้างและข้อมูลที่ต้องจัดเก็บเป็นจำนวนมาก จึงนิยมนำฐานข้อมูลเชิงกราฟมาใช้ในการจัดการและประมวลผลข้อมูลขนาดใหญ่ของจำนวนข้อมูลที่ไม่มีโครงสร้างได้ โดยฐานข้อมูล NoSQL แบ่งออกเป็น 4 ประเภท ซึ่งข้อมูลในแต่ละประเภทต้องเลือกให้เหมาะสมกับระบบฐานข้อมูลที่ใช้งาน และงานวิจัยนี้จึงได้นำฐานข้อมูลเชิงสัมพันธ์ (Oracle) และฐานข้อมูลเชิงกราฟ (Neo4j) มาเปรียบเทียบกัน โดยนำชุดข้อมูล Medcare มาทำการปรับตามโครงสร้างตามรูปที่ 2.17 และในการทดสอบครั้งนี้ แสดงให้เห็นว่าทุกครั้งที่ข้อมูลมีการเชื่อมต่อกันมากขึ้น (มีจำนวนรวมกันมาก) และมีขนาดใหญ่ ฐานข้อมูลเชิงกราฟ (Neo4j) จะแสดงประสิทธิภาพที่ดีกว่าฐานข้อมูลเชิงสัมพันธ์ (Oracle)



รูปที่ 2.17 การเปรียบเทียบระหว่างระบบฐานข้อมูล Graph-oriented

สำหรับงานวิจัยครั้งนี้ใช้ทั้งหมด 5 queries โดยแสดงถึงลักษณะระดับความซับซ้อนของแต่ละ queries ในการดำเนินการ ดังตารางที่ 2.1

ตารางที่ 2.1 แสดงตัวอย่างการทดสอบ query

Query#	Oracle 11g Enterprise Edition	Neo4j 3.0.3 Community Edition
1	Select count(*) From Patient_Visit p,Patient_Issuemed i where P.patient_visitno=I.patient_visitno and P.depend_xno=I.depend_xno and P.patient_id=I.patient_id;	MATCH (PATIENT_VISIT)-[r:has_med]->(PATIENT_ISSUEMED) RETURN COUNT(*)
2	Select count(*) From patient p, dependent d Where d.patient_id=p.patient_id;	MATCH (dep:DEPENDENT)<-[:has_dependent]-(pd:PATIENT_DATA) RETURN COUNT(*)
3	Select count(*) from Patient p, Dependent d, Patient_visit pv where d.patient_id=p.patient_id and pv.depend_xno=d.depend_xno and pv.patient_id=d.patient_id;	MATCH (visit:PATIENT_VISIT)<-[:VISITS_ARE]- (dep:DEPENDENT) OPTIONAL MATCH (dep)<-[:has_dependent]-(pd:PATIENT_DATA) Return count(*)
4	select count(*) from patient_visit pv where pv.depend_xno in(select d.depend_xno from dependent d where d.depend_xno=pv.depend_xno) and pv.patient_id in (select p.patient_id from patient p where p.patient_id=pv.patient_id)	MATCH (visit:PATIENT_VISIT)<-[:VISITS_ARE]- (dep:DEPENDENT) OPTIONAL MATCH (dep)<-[:has_dependent]-(pd:PATIENT_DATA) Return count(*)
5	select count(*) from patient_issuemed pi where pi.patient_id in (select p1.patient_id from patient_visit p1 where p1.patient_id=pi.patient_id) and pi.depend_xno in (select p2.depend_xno from patient_visit p2) and pi.patient_visitno in(select p3.patient_visitno from patient_visit p3	MATCH (PATIENT_VISIT)-[r:has_med]->(PATIENT_ISSUEMED) RETURN COUNT(*)

ในการทดสอบจะใช้ queries ทั้งหมด 5 แบบ มาทดสอบกับฐานข้อมูลเชิงสัมพันธ์ (Oracle) และฐานข้อมูลเชิงกราฟ (Neo4j) จากการทดสอบแสดงให้เห็นว่า Neo4j ทำงานได้ดีกับเวลาที่เหมาะสมในการสืบค้นทั้งหมดสำหรับชุดข้อมูล Medcare และเวลาในการค้นหาข้อความแต่ละวินาทีจะแสดงในตารางที่ 2.2

ตารางที่ 2.2 แสดงเวลาในการทำ Query ต่อวินาที

Query#	Oracle 11g	Neo4j 3.0.3
1	4.515 Sec	0.346 Sec
2	0.172 Sec	0.216 Sec
3	3.531 Sec	0.452 Sec
4	3.469 Sec	0.452 Sec
5	10.391 Sec	0.346 Sec

จากนั้นจะนำฐานข้อมูลเชิงสัมพันธ์ (Oracle) มาทดสอบบนระบบ Local และ Server ที่อยู่ในสถานะที่สอดคล้องกัน ดังตารางที่ 2.3

ตารางที่ 2.3 ผลการวิจัยระหว่าง Environment

ORACLE 11g										
Query#	Local System					Server System				
	1st	2nd	3rd	4th	5th	1st	2nd	3rd	4th	5th
1	0.953	0.391	0.422	0.469	9.359	6.093	0.359	5.328	4.907	15.094
2	0.422	0.047	0.094	0.094	9.265	5.828	0.125	4.750	4.672	14.844
3	0.407	0.031	0.109	0.078	9.172	6.375	0.094	4.782	4.844	15.219
4	0.407	0.016	0.109	0.079	9.238	5.781	0.375	5.938	4.391	14.703
5	0.406	0.047	0.094	0.094	9.219	5.781	0.062	4.765	5.047	18.187

จากตารางที่ 2.3 ใช้ queries ทั้งหมด 5 แบบและทดสอบทั้งหมด 5 ครั้งบนระบบ Local และระบบ Server ซึ่งผลลัพธ์ที่ได้คือ ประสิทธิภาพของระบบ Local ทำงานได้ดีกว่าบนระบบ Server ส่วนความเร็วบนระบบ Local ในขณะที่ทำการทดสอบอยู่ที่ 0.557 mb /s และนำไปวัดประสิทธิภาพจากเครื่องมือ Weka (J48) ใช้การตรวจสอบแบบ 10-fold cross-validation ซึ่งได้ผลลัพธ์ คือ ระบบของ Local (Local System) จะทำงานได้ดีกว่า Server (Remote System) โดยได้ค่าความถูกต้อง 96%

จากงานวิจัยทั้งหมดที่กล่าวมาในข้างต้น สามารถสรุปได้ว่างานวิจัยที่ 1 จะกล่าวถึงโครงสร้างปัญหา ที่เกิดขึ้นในการจัดเก็บข้อมูล และวิธีการเพิ่มประสิทธิภาพของ RDF ที่นำไปใช้งานกับ Semantic Web งานวิจัยที่ 2, 3 และ 4 ได้กล่าวถึงคุณสมบัติพื้นฐานของระบบฐานข้อมูล NoSQL และ CAP theorem โดยงานวิจัยที่ 2 จะเน้นถึงการเลือกระบบฐานข้อมูลให้เหมาะสมกับความต้องการของระบบ แต่ภายในเนื้อหาของงานวิจัยนั้นไม่ได้กล่าวถึงระบบฐานข้อมูลประเภท Graph-based NoSQL แต่ในขณะที่งานวิจัยที่ 3 จะได้กล่าวถึงคุณสมบัติต่าง ๆ ของระบบฐานข้อมูล NoSQL ทั้ง 4 ประเภท ภายในเนื้อหาของงานวิจัยได้มีการยกตัวอย่างของระบบฐานข้อมูลที่ได้รับความนิยมในแต่ละประเภท และงานวิจัยที่ 4 ได้กล่าวถึงการเปรียบเทียบระหว่างระบบฐานข้อมูล Neo4j ซึ่งอยู่ในประเภท Graph-based NoSQL กับระบบฐานข้อมูลเชิงสัมพันธ์ (Oracle) ภายในเนื้อหาได้มีการเปรียบเทียบการดำเนินการรูปแบบต่าง ๆ

คณะผู้วิจัยได้ทำการศึกษาวิจัยต่างประเทศทั้งหมด 4 ฉบับ และได้เล็งเห็นข้อเสนอแนะเพิ่มเติมสำหรับงานวิจัยทั้ง 4 ฉบับ ดังนี้

- ในงานวิจัยที่ 1 ได้มีข้อเสนอแนะว่า ควรเพิ่มเติมตัวอย่างของระบบฐานข้อมูลที่รองรับการทำงาน RDF ซึ่งควรมีการพูดถึงการใช้งาน RDF กับข้อมูลลักษณะอื่น ๆ เช่น ข้อมูลเชิงกราฟ และควรมีการเปรียบเทียบรูปแบบการเก็บข้อมูลระหว่าง RDF กับ RDBMS เพื่อให้มีความเข้าใจมากยิ่งขึ้น

- ในงานวิจัยที่ 2 ได้มีข้อเสนอแนะว่า งานวิจัยนี้ควรกล่าวถึง ฐานข้อมูล NoSQL ประเภท Graph-based ซึ่งควรมีการกล่าวถึง CA ใน CAP Theorem เพื่อทำให้รายละเอียดในงานวิจัยมีความสมบูรณ์มากยิ่งขึ้น

- ในงานวิจัยที่ 2 และ 3 ได้มีข้อเสนอแนะว่า ควรเพิ่มเติมเกี่ยวกับการเก็บข้อมูลเชิงกราฟ การจัดการ Index, การวิเคราะห์ข้อมูล และเครื่องมือที่ใช้ในการเก็บข้อมูล

- ในงานวิจัยที่ 4 ได้มีข้อเสนอแนะว่า ควรเพิ่มเติมรายละเอียดส่วนของการจัดการ Index เพื่อให้เห็นถึงประสิทธิภาพที่เพิ่มมากยิ่งขึ้นของระบบฐานข้อมูลที่นำมาทำการวิจัย

บทที่ 3
วิธีการดำเนินการวิจัย

ในบทนี้จะกล่าวถึงการวิเคราะห์เชิงปริมาณของระบบฐานข้อมูล NoSQL ประเภทกราฟ โดยใช้ระบบฐานข้อมูลบนระบบปฏิบัติการ Windows (Windows 10) คุณสมบัติของเครื่องที่ใช้งาน CPU i7-6700k, RAM 16GB และ SSD 120 GB โดยนำแต่ละระบบฐานข้อมูลนำมาเปรียบเทียบกัน ดังนี้

3.1 ข้อมูลที่ใช้ในการวัดประสิทธิภาพ

3.1.1 Non-Graph data

3.1.2 Graph data

3.2 ระบบฐานข้อมูลที่ใช้ในการวิจัย

3.2.1 ระบบฐานข้อมูล Neo4j (3.3.5) Free Edition

3.2.2 ระบบฐานข้อมูล OrientDB (3.0.0) Free Edition

3.2.3 ระบบฐานข้อมูล GraphDB (GraphDB 8) Free Edition

3.2.4 ระบบฐานข้อมูล ArangoDB (3.2.9) Free Edition

3.3 การวัดประสิทธิภาพของโปรแกรม

3.3.1 การเปรียบเทียบคำศัพท์ระหว่าง RDBMS กับ Graph-based databases

3.3.2 การเปรียบเทียบคุณสมบัติพื้นฐานของ Graph-based NoSQL ที่นำมาใช้กับ RDBMS

3.3.3 ตัวชี้วัดประสิทธิภาพการทำงาน

3.3.4 Queries ที่นำมาใช้ในการทดลองกับข้อมูลประเภท non-graph

3.3.5 Queries ที่นำมาใช้ในการทดลองกับข้อมูลประเภท graph

3.1 ข้อมูลที่ใช้ในการวัดประสิทธิภาพ

3.1.1 Non-Graph data

Non-Graph data หมายถึง ข้อมูลโดยทั่วไป โดยที่ entities และลักษณะเชิงบรรยาย (attribute) ไม่มีความสัมพันธ์ระหว่างกัน

BibNumber	ItemBarcode	ItemType	Collection	CallNumber	CheckoutDateTime
1842225	10035249209	acbk	namys	MYSTERY ELKINS1999	5/23/2005 15:20
1928264	10037335444	jcbk	ncpic	E TABACK	12/14/2005 17:56
1982511	10039952527	jcvhs	ncvidnf	VHS J796.2 KNOW_YO 2000	8/11/2005 13:52
2026467	10040985615	accd	nacd	CD 782.421642 Y71T	10/19/2005 19:47
2174698	10047696215	jcbk	ncpic	E KROSOCZ	12/29/2005 15:42
1602768	10028318730	jcbk	ncpic	E BLACK	10/8/2005 14:15
2285195	10053424767	accd	cacd	CD 782.42166 F19R	9/30/2005 10:16
2245955	10048392665	jcbk	ncnf	J949.73 Or77S 2004	12/5/2005 17:03
770918	10044828100	jcbk	ncpic	E HILL	7/22/2005 15:17
2288252	10053488788	jcdvd	ncdvd	DVD J HOW DO	5/5/2005 11:18
2129883	10044896768	acdvd	nadvd	DVD GOOD TH	6/8/2005 17:44
1100696	10039424485	acbk	nypb	FIC COONEY	5/27/2005 16:54
2285204	10053420229	accd	nacd	CD 782.421649 B7391B	5/19/2005 17:30
2122199	10042855428	acvhs	navid	VHS CHEYENN	4/15/2005 13:56

รูปที่ 3.1 ข้อมูลที่นำมาใช้ในการทดลอง non-graph

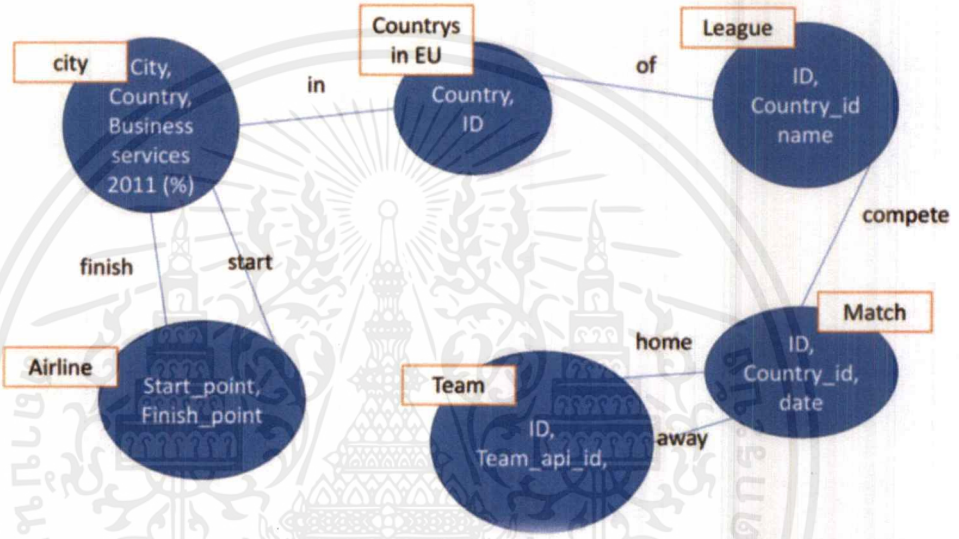
ข้อมูลที่ใช้แบบ Non-graph data โดยในงานวิจัยนี้ได้ใช้ชุดข้อมูลการบันทึกการตรวจสอบรายการทั้งหมดจากหอสมุดสาธารณะในเมือง Seattle โดยชุดข้อมูลเริ่มเก็บข้อมูลตั้งแต่ 2005-2017 ซึ่งจัดเก็บข้อมูลการยืม-คืน หนังสือในหอสมุดแห่งนี้โดยขนาดที่ใช้มี 4 ขนาด คือ 1 GB, 3 GB, 5 GB และ 7 GB ตามลำดับ ดังรูปที่ 3.1 ซึ่งข้อมูลของแต่ละคอลัมน์จะแสดงถึงการคืนหนังสือ ดังนี้ หมายเลขการยืม, รหัสบาร์โค้ดของหนังสือ, ประเภทของอุปกรณ์ที่ยืม, รายละเอียดของอุปกรณ์ และวันที่คืนหนังสือ

3.1.2 Graph data

ทฤษฎีกราฟ เป็นแบบจำลองทางคณิตศาสตร์ ซึ่งใช้จำลองในการแก้ปัญหาโดยเขียนแผนภาพ ที่ประกอบด้วยโหนด (Node) หรือจุด (Vertex) และเส้นเชื่อม (Edge) ซึ่งในปัจจุบันมีการนำทฤษฎีกราฟมาประยุกต์ใช้ในศาสตร์สาขาต่าง ๆ เช่น วิทยาศาสตร์ สังคมศึกษา เศรษฐศาสตร์ พันธุศาสตร์ วิศวกรรมศาสตร์เป็นต้น กราฟ (G) ประกอบไปด้วยเซตจำนวน 2 เซต คือ

Customer	Total Cost ex VAT	Travel Class	Ticket Single or Return	Travel Date	Journey Start Point	Journey Finish Point	Air Carrier
Department for Transport	£81.52	ECONOMY	Return	10/2/2011	GLASGOW	LONDON	FLYBE
Department for Transport	£217.14	ECONOMY	Return	29/3/2011	LONDON	MARRAKECH	EASYJET
Department for Transport	£7,969.20	BUSINESS	Return	7/4/2011	LONDON	SINGAPORE	BRITISH AIRWAYS
Department for Transport	£272.82	BUSINESS	Return	7/4/2011	LONDON	EDINBURGH	BRITISH AIRWAYS
Department for Transport	£7,969.20	BUSINESS	Return	7/4/2011	LONDON	SINGAPORE	BRITISH AIRWAYS
Department for Transport	£387.60	ECONOMY	Return	8/4/2011	LONDON	MILAN - LINATE	ALITALIA
Department for Transport	£206.03	ECONOMY	Return	9/4/2011	SOUTHAMPTON	MANCHESTER	FLYBE
Department for Transport	£278.80	ECONOMY	Return	10/4/2011	LONDON	MANCHESTER	BRITISH AIRWAYS
Department for Transport	£278.80	ECONOMY	Return	10/4/2011	LONDON	MANCHESTER	BRITISH AIRWAYS
Department for Transport	£214.68	ECONOMY	Return	12/4/2011	LONDON	DUBLIN	AER LINGUS
Department for Transport	£190.43	ECONOMY	Return	12/4/2011	STANSTED	EDINBURGH	EASYJET

รูปที่ 3.2 ข้อมูลที่นำมาใช้ในการทดลอง graph



รูปที่ 3.3 ความสัมพันธ์ของ graph data

จากรูปที่ 3.3 แสดงถึงข้อมูลที่ใช้สำหรับ Graph data ข้อมูลไฟล์ทบินระหว่างเมืองในประเทศของทวีปยุโรป ซึ่งประกอบไปด้วย คลาสของตั๋วที่นั่ง, วันที่บิน,เมืองต้นทาง-ปลายทาง และชื่อของสายการบิน ข้อมูลธุรกิจในแต่ละประเทศของทวีปยุโรป การผลิตของโรงงาน, เหมือง, ธุรกิจเกี่ยวกับการบริการ, จำนวนประเภทงาน และทักษะการทำงานของประชากร ข้อมูลทีมฟุตบอลในแต่ละลีคของทวีปยุโรป ประกอบไปด้วย แมทช์, ทีมฟุตบอล, ชื่อลีค และข้อมูลในการแข่งขันในแต่ละแมทช์ที่มีขนาดทั้งหมด 1.5 GB จากรูปที่ 3.2 ซึ่งข้อมูลของแต่ละคอลัมน์จะแสดงถึงรายละเอียดการบินในแต่ละไฟล์ท ดังนี้ ชื่อของแผนกที่บริการ, ราคา, ระดับที่นั่ง, ประเภทของตั๋วการบิน, วันที่เดินทาง, จุดเริ่มต้น การเดินทาง, จุดปลายทางการเดินทาง, ชื่อสายการบิน

3.2 ระบบฐานข้อมูลที่ใช้ในการวิจัย

3.2.1 ระบบฐานข้อมูล Neo4j (3.3.5) [6-8], [23-24]

รูปที่ 3.4 โลโก้ของระบบฐานข้อมูล Neo4j

เป็นระบบฐานข้อมูลที่สร้างแบบจำลองของกราฟขึ้นมาโดยใช้โหนดและความสัมพันธ์ในการอธิบายความสัมพันธ์เหล่านั้น ซึ่งฐานข้อมูลแบบกราฟสามารถทำการ query ให้เข้าใจได้ง่ายจากภาษาที่ทางระบบฐานข้อมูล (Cypher) ได้พัฒนาขึ้นใหม่ เพื่อตอบสนองกับปัญหาในการ query ที่ซับซ้อนของความสัมพันธ์ที่มีอย่างหลากหลาย

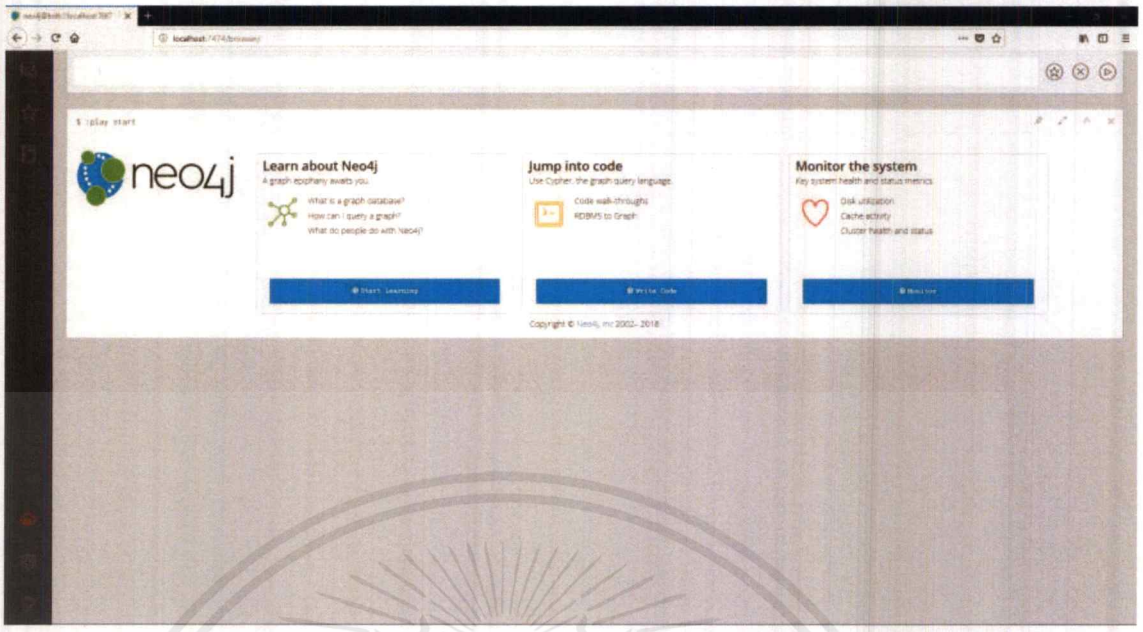
องค์ประกอบของกราฟจะประกอบด้วย โหนด ความสัมพันธ์ คุณสมบัติ และป้ายชื่อกำกับโหนด (label) โหนดมีคุณสมบัติคล้ายกับเอกสารที่เก็บอยู่ในรูปแบบของคู่อันดับแบบ key-value เป็นแบบสตริงหรือ แบบอาเรย์ใน Java ซึ่งโหนดสามารถมีป้ายกำกับชื่ออย่างน้อยหนึ่งป้ายจะช่วยในการบ่งบอกถึงข้อมูลที่อยู่ในโหนดนั้น ความสัมพันธ์ที่เชื่อมต่อกันในแต่ละโหนดของโครงสร้างกราฟโดยที่ความสัมพันธ์จะมีทิศทางและมีป้ายชื่อแบบเดียวกับโหนดที่อยู่ระหว่างโหนดเริ่มต้นกับโหนดปลาย ภายในโหนดความสัมพันธ์สามารถมีคุณสมบัติได้เพื่อเก็บข้อมูลที่มีระหว่างโหนดที่เชื่อมต่ออยู่

โดยการจัดเก็บข้อมูลของฐานข้อมูลประเภทนี้จะเก็บแบบ label กับ node ซึ่งรูปแบบของ label จะคล้ายกับ table และ node คล้ายกับ row ในฐานข้อมูลแบบ RDBMS จึงทำให้ node สามารถมีข้อมูลซ้ำกันหลาย ๆ node ได้ โดยจะแบ่งแยกออกจากกันด้วย label ที่กำหนดให้เพื่อบอกถึงกลุ่มของแต่ละ node ซึ่งแต่ละ node สามารถเชื่อมความสัมพันธ์กับ node อื่นได้ด้วยการระบุเงื่อนไขในการ query ที่คล้ายกับการเชื่อมความสัมพันธ์ใน RDBMS แต่จะไม่มีเงื่อนไขของ foreign key โดยสามารถทำการเปลี่ยนแปลงหรือลบข้อมูลภายใน node ออกได้ทั้งที่ยังมีความสัมพันธ์อยู่

คุณลักษณะสำคัญของ Neo4j การสร้างความสัมพันธ์ระหว่างโหนดจะไม่มีเงื่อนไข constraint และในการค้นหาข้อมูลที่มีความซับซ้อนกัน โดยค้นหาโหนดที่มีความสัมพันธ์ทั้งความลึกและความกว้างของโครงสร้างกราฟจะมีผลการแสดงตามความสัมพันธ์ ความสัมพันธ์ทั้งหมดของ Neo4j มีความสำคัญและทำให้มีความความเร็วการในทำงาน ในการสร้างความสัมพันธ์ใหม่สามารถทำได้ในภายหลังจากการสร้างโหนดได้ หน่วยความจำที่มีขนาดไม่ใหญ่มากและสามารถรองรับการเพิ่มจำนวนเครื่อง เพื่อเพิ่มประสิทธิภาพ สามารถทำงานได้ใน Java Virtual Machine และมีการสนับสนุนคุณสมบัติ ACID โดยมีการทำงานของระบบฐานข้อมูลดังต่อไปนี้

รูปที่ 3.5 หน้าต่างที่ใช้ในการเปิด service

จากรูปที่ 3.5 วิธีการ Start service สามารถทำได้โดยเข้าไปยัง path ของ Neo4j โดยใช้คำสั่งผ่าน Command Prompt >> \neo4j-community-3.3.5-windows\neo4j-community-3.3.5\bin และใช้คำสั่ง Neo4j start เพื่อเปิด Service ของ Neo4j



รูปที่ 3.6 หน้าต่างในการรับคำสั่งของระบบฐานข้อมูล

จากรูปที่ 3.6 แสดงหน้าต่างการใช้งานของระบบฐานข้อมูล โดยที่ URL ใส่ค่า localhost:7474

อัลกอริทึมของระบบฐานข้อมูล Neo4j

Sokratis Kartelias ของทาง Neo4j ได้จัดทำ search engine optimization (SEO) ขึ้นเพื่อจัดเก็บข้อมูลสำหรับการจัดการกับ Metadata โดยใช้ในการจัดการหมวดหมู่ข้อมูลสินค้าของ adidas และค้นหาข้อมูลได้ทุกแพลตฟอร์ม เพื่อเพิ่มความสามารถในการทำงานแบบเรียลไทม์ได้อย่างมีประสิทธิภาพ โดยการกำหนดรูปแบบของ Metadata ทั้ง 3 โดเมน เพื่อใช้ในการค้นหาความสัมพันธ์ที่หลากหลายมากขึ้น

Shutl ได้สร้างแพลตฟอร์มขึ้นมาเพื่อรองรับการเติบโตของข้อมูลภายในเวลาอันรวดเร็วจากกรณีของ ebay จึงได้เพิ่ม Service-Oriented Architecture (SOA) เพื่อแก้ไขปัญหาด้านประสิทธิภาพและเพิ่มความยืดหยุ่น และเพิ่มฟังก์ชันในการทำงานของภาษา Cypher ให้สามารถรองรับกับ library ของภาษา Ruby เพื่อเพิ่มกำลังในการค้นหาข้อมูลให้มีประสิทธิภาพมากยิ่งขึ้น

Features ของ Neo4j

ฐานข้อมูลแบบกราฟสามารถเขียน query ให้เข้าใจได้อย่างง่าย รองรับการใช้ Index ของ Apache Lucence สนับสนุนกฎทุกข้อของ ACID (Atomicity, Consistency, Isolation and Durability) ใช้หน่วยเก็บข้อมูลกราฟ Native โดยใช้ Native GPE (Graph Processing Engine) และให้ REST API เข้าถึงได้โดยใช้ภาษา Java ,Spring, Scala และอื่น ๆ ซึ่ง CQL สามารถ query ได้เหมือนภาษา SQL อีกทั้งสามารถใช้กับ UI จาก Neo4j Data Browser ในการจัดการรูปแบบของข้อมูลได้ และรองรับ API ของ JAVA ในการใช้งาน Java applications

3.2.2 ระบบฐานข้อมูล OrientDB (3.0.0) [19-21], [29]

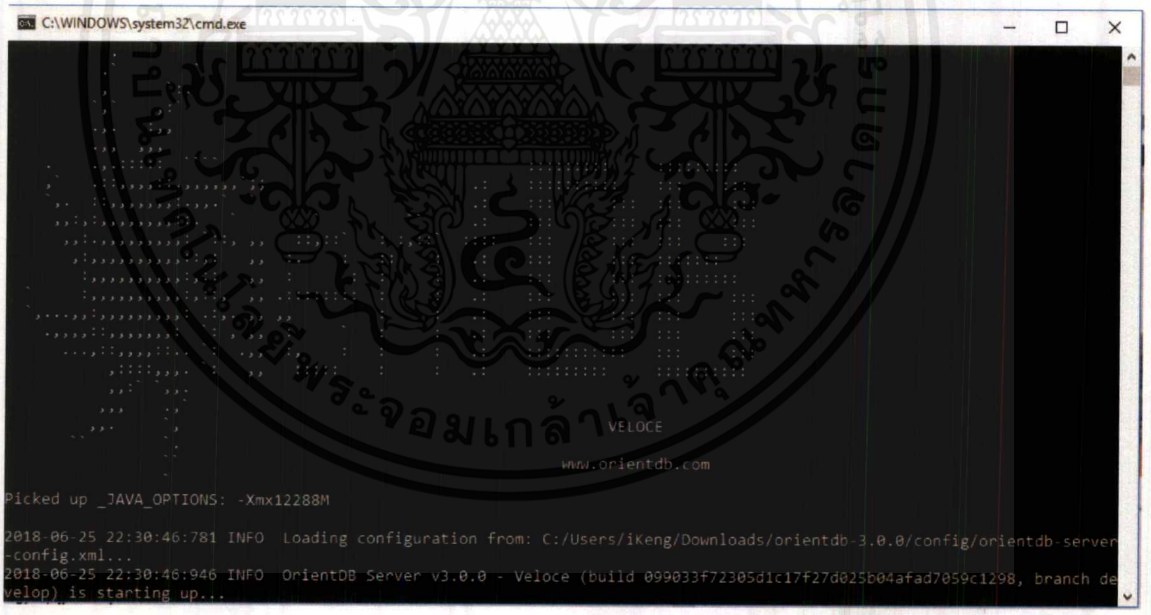
รูปที่ 3.7 โลโก้ของระบบฐานข้อมูล OrientDB

OrientDB เป็นฐานข้อมูลตัวใหม่ที่เป็น NoSQL ซึ่งเป็น open source โดยเกิดมาพร้อมกับคุณสมบัติที่ดีที่สุดของทุก ๆ สิ่ง โดยตัว Orient DB นั้นถูกเขียนขึ้นด้วยภาษาจาวาทั้งหมดและตัวระบบฐานข้อมูลนั้นมีการทำงานของระบบฐานข้อมูลมีความรวดเร็วเป็นอย่างมาก โดยการทำงานสามารถรองรับได้ในทุก platform ที่สนับสนุนเทคโนโลยีของจาวาตั้งแต่เวอร์ชัน 5 ขึ้นไปจึงทำให้ตัว Orient DB มันสามารถจัดเก็บเรคอร์ดได้สูงสุดถึง 200,000 เรคอร์ดภายใน เวลา 5 วินาทีบนฮาร์ดแวร์ทั่วไป จะมี การเชื่อมต่อโดยตรงระหว่างกราฟกับเรคอร์ดนั้น ๆ และเป็น Document Database แต่มีคุณสมบัติที่ดีที่สุดของ DBMS อื่น ๆ เช่น ความสัมพันธ์ได้รับการจัดการเป็น Graph Databases

โดยที่คุณสามารถท่องกราฟไปในทุก ๆ ส่วนหรือในแต่ละส่วนของต้นไม้และกราฟของเรคอร์ดนั้นในเวลาเพียงหลักมิลลิวินาทีเท่านั้น ตัว OrientDB จะสนับสนุนตัว schema-less, schema-full และ schema-mixed modes โดยมีระบบรักษาความปลอดภัยของข้อมูลบนผู้ใช้ที่แข็งแรงและมีการสนับสนุนภาษา SQL ระหว่างการใส่ข้อมูลลงในตาราง

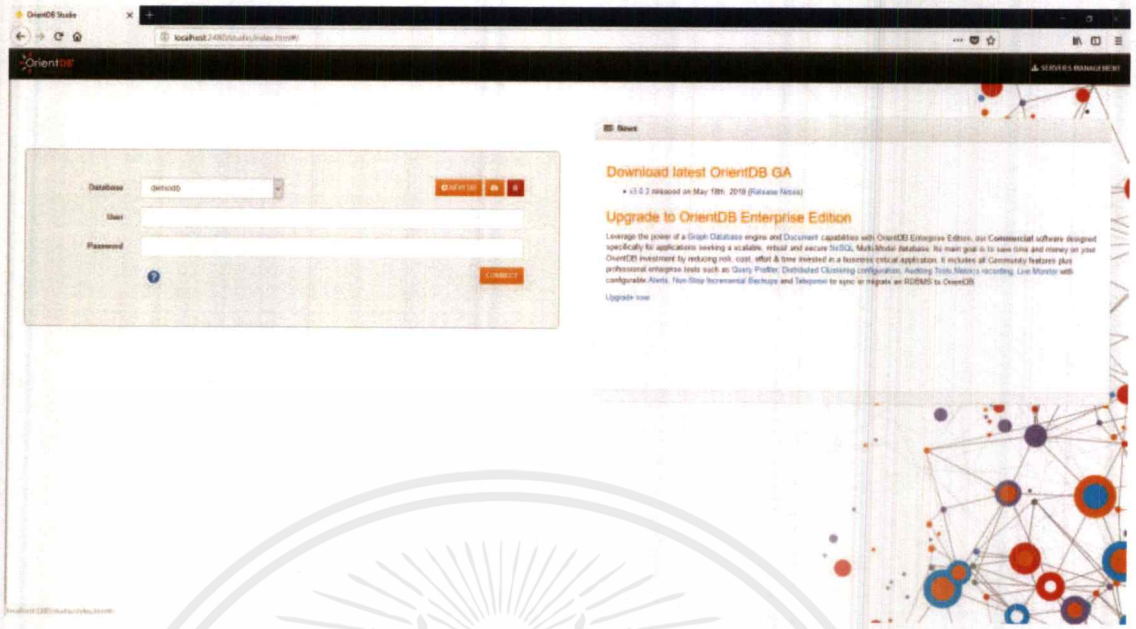
ปัญหาที่พบบ่อยในการทำงาน คือ ปัญหาที่ฐานข้อมูลที่เป็นคอขวดของการใช้งานมากที่สุด OrientDB มีขนาดของข้อมูลที่ไหลออกมาบนเครื่องจักรที่เป็นแบบระบบเดี่ยว แล้วหนึ่งในเซิร์ฟเวอร์นั้นสามารถสร้างการทำงานได้ถึง 125 เซิร์ฟเวอร์ที่กำลังทำงานกับ MySQL กระบวนการดำเนินการของเครื่องจักรนั้นสามารถทำงานได้ ในระบบประมวลผลแบบกระจายแล้วยังสนับสนุนและรองรับเรคอร์ดได้ถึง 9,223,372,036 billions ของเรคอร์ดแล้วมีความจุมากที่สุด 19,807,040,628,566,084 terabytes ของข้อมูลที่อยู่กระจัดกระจายกันบนหลาย ๆ ฮาร์ดดิสก์ในหลาย ๆ โหนด

OrientDB ทำงานได้รวดเร็ว เนื่องจากระบบฐานข้อมูลมีการสืบทอดคุณสมบัติและแนวคิดของฐานข้อมูล เชิงวัตถุ Graph DBMS และ NoSQL นอกจากนี้ยังมีการใช้โครงสร้าง B^+- tree ซึ่งมีความคล้ายคลึงกับแบบ B-Tree ซึ่งแต่ละโหนดจะต้องมีค่าคีย์อย่างน้อย ½ ของค่าคีย์ทั้งหมดที่จะมีได้เต็มที่ นอกจากนี้ leaf node ทุกตัวยังเชื่อมโยงค่าคีย์ไปตามลำดับ ซึ่งจะทำให้สามารถเข้าถึงตัวข้อมูลด้วยวิธีการแบบ Sequential ได้รวดเร็วยิ่งขึ้น และยังมีความเร็วในการกู้คืนข้อมูลและยังเก็บเส้นทางของโหนดที่ยังคงมีอยู่ไว้โดยมีการทำงานของระบบฐานข้อมูลดังต่อไปนี้



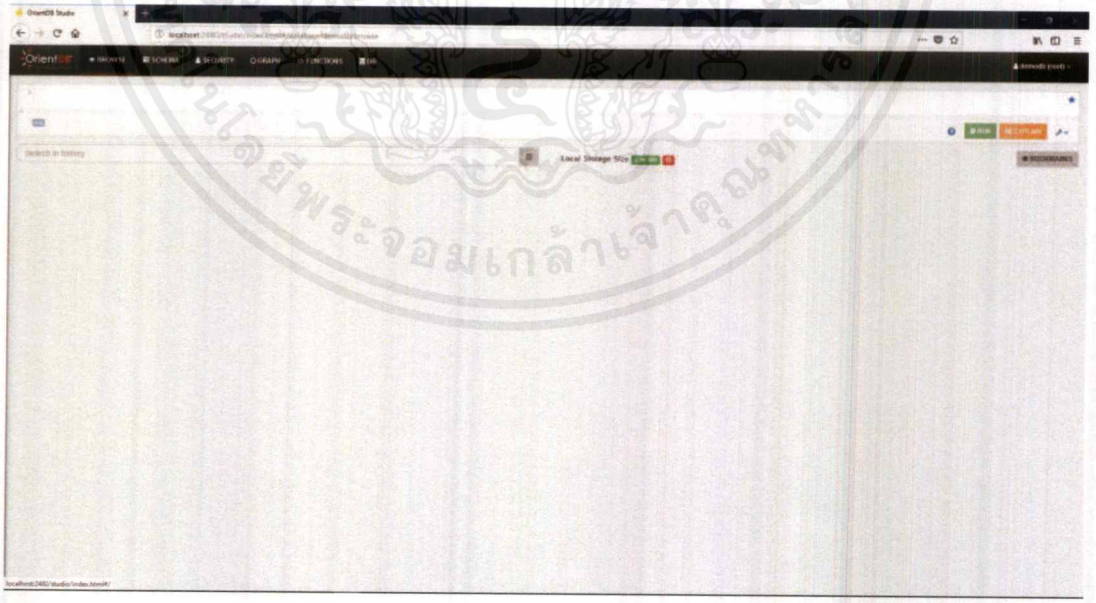
รูปที่ 3.8 แสดงหน้าต่างที่ใช้ในการเปิด Service ของระบบฐานข้อมูล

จากรูปที่ 3.8 ทำการเลือกไฟล์ Service.sh ในการเปิดการทำงานของ service (local host) โดยการรันบน Terminal ดังรูปที่ 3.9



รูปที่ 3.9 หน้าต่างในเข้าสู่ระบบของระบบฐานข้อมูล

จากรูปที่ 3.9 หน้าต่างเข้าสู่ระบบของระบบฐานข้อมูลในการใช้งาน User และ Password จะใส่เหมือนกับตอนเปิดเซิร์ฟเวอร์บน Terminal ดังรูป 3.8



รูปที่ 3.10 แสดงหน้าต่างในการรับคำสั่งของระบบฐานข้อมูล

จากรูปที่ 3.10 แสดงหน้าต่างการใช้งานของระบบฐานข้อมูล โดยที่ URL ใส่ค่า localhost:2480

อัลกอริทึมของระบบฐานข้อมูล OrientDB

OrientDB เป็นฐานข้อมูลที่สามารถสนับสนุนข้อมูลหลายรูปแบบ ทั้งแบบ graph document key-values และรูปแบบ object โดยการจัดการความสัมพันธ์ของฐานข้อมูลทำกระทำการระหว่าง record ซึ่งกลไกที่ใช้ในการจัดการ index ได้ใช้ b-tree และ extendible hashing (ซึ่งจะเรียกว่า hash index) โดยโครงสร้างของฐานข้อมูลถูกสร้างขึ้นมาในรูปแบบ Multi-master แบบ Zero-Configuration ที่สามารถรองรับระบบ cloud ในการประมวลผลแบบกระจายได้สมบูรณ์มากยิ่งขึ้น

Features ของ OrientDB

เป็นระบบการจัดการฐานข้อมูล NoSQL แบบ Open Source ที่มีความคล้ายคลึงกับระบบฐานข้อมูล RDBMS เป็นอย่างมาก ทั้งการ query และการจัดการข้อมูล ซึ่ง OrientDB เป็นระบบการจัดการฐานข้อมูล multi-model ที่มีความยืดหยุ่นมากกว่าแบบ RDBMS จึงสามารถรองรับการทำงานแบบ graph และแบบ relational ได้ทั้ง 2 รูปแบบ และสามารถรองรับ index ได้หลายรูปแบบ เช่น Hash index, Full-text index, Spatial index, B-Tree index เป็นต้น ซึ่งในด้านการประยุกต์ใช้งานก็สามารถรองรับการทำงานของ API Java Script, API Gremlin, API HTTP methods เป็นต้น

3.2.3 ระบบฐานข้อมูล GraphDB (GarphDB8) [12], [25-26], [30-33]

รูปที่ 3.11 โลโก้ของระบบฐานข้อมูล GraphDB

เป็นฐานข้อมูลการสร้างแบบจำลองของระบบแต่ละชนิดข้อมูลและการค้นหาข้อมูลภายในระบบ ซึ่งฐานข้อมูลจะเป็นชุดของคลาสที่มีออบเจ็กต์จะแบ่งออกเป็น 3 ประเภท คลาสแบบง่าย คลาสแบบลิงค์ และคลาสแบบเส้นทาง โมเดลของฐานข้อมูลจะเป็นออบเจ็กต์ภายในโหนดจะมีข้อมูลประจำตัวของออบเจ็กต์นั้น และสามารถมีประเภทข้อมูลที่เป็นจำนวนเต็มหรือสตริงหรือชนิดของออบเจ็กต์จะเป็นการอ้างอิงออบเจ็กต์อื่น ดังนั้นโครงสร้างของ tuple จะมีออบเจ็กต์อยู่ ในแต่ละออบเจ็กต์จะเป็นโหนดของฐานข้อมูลกราฟออบเจ็กต์ ที่เป็นคลาสลิงค์จะอ้างอิงถึงออบเจ็กต์เป้าหมายสองโหนด ออบเจ็กต์คลาสเส้นทางจะประกอบด้วย รายการอ้างอิงไปยังโหนด และโหนดปลายทางของกราฟ

โมเดลของฐานข้อมูลใช้แบบ RDF (Resource Description Framework) ใช้แทนการอธิบายความหมายของข้อมูลโมเดลจะประกอบด้วย ลิงค์ (เส้นเชื่อมความสัมพันธ์) และโครงสร้างของโมเดลจะประกอบด้วย Subject, Predicate, Object การใช้งานฐานข้อมูลกราฟ โดยใช้ภาษา SPARQL ในการ query ข้อมูลสำหรับข้อมูลกราฟ RDF ซึ่งจะใช้คำสั่ง Select ซึ่งส่งผลให้เกิดผลลัพธ์แบบตาราง Construct สร้างกราฟ RDF ใหม่ตามผลการค้นหา ASK ซึ่งส่งกลับ “Yes” หากข้อความค้นหามีทางออกมิฉะนั้น “No” Describe ซึ่งส่งกลับข้อมูลกราฟ RDF เกี่ยวกับทรัพยากร มีประโยชน์เมื่อผู้ใช้งานสามารถ query โดยไม่ทราบโครงสร้างของข้อมูล RDF ในแหล่งข้อมูล Insert ซึ่งสามารถใส่ข้อมูล 3 ส่วนลงในกราฟได้ และ Delete ซึ่งลบโหนดที่ไม่ต้องการออกจากกราฟ

คุณสมบัติหลักของระบบฐานข้อมูล GraphDB มีดังนี้

1. การรักษาความสามารถของ index ที่สร้างไว้กับข้อมูลที่เชื่อมต่อกันในระบบฐานข้อมูล

2. ระบบฐานข้อมูล GraphDB สามารถรองรับการทำงานของข้อมูลเชิงภูมิศาสตร์

3. ค่าที่เก็บอยู่ในระบบฐานข้อมูลจะมีความเป็นอิสระต่อกัน

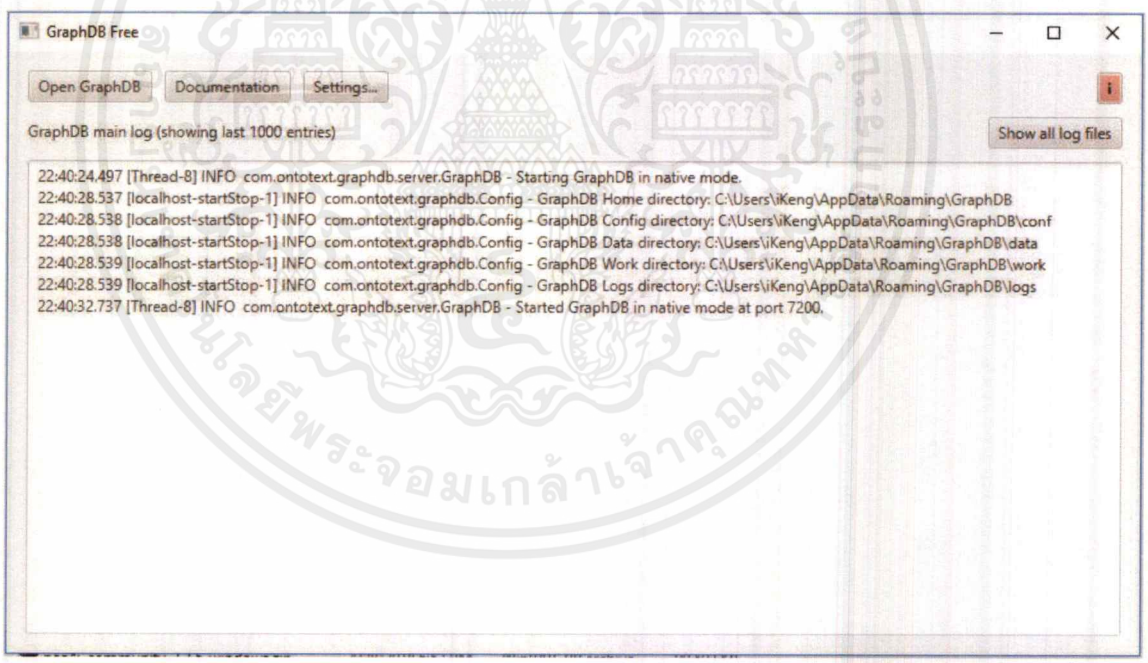
4. มีการกำหนดเอนทิตี้ในการเชื่อมต่อกัน ดังนี้

    4.1 รายการของฟิลด์และคุณสมบัติที่มีค่าจะถูกเชื่อมต่อกัน

    4.2 รายการภาษาที่ใช้สำหรับการเชื่อมต่อ สามารถเป็นได้ทุกภาษา

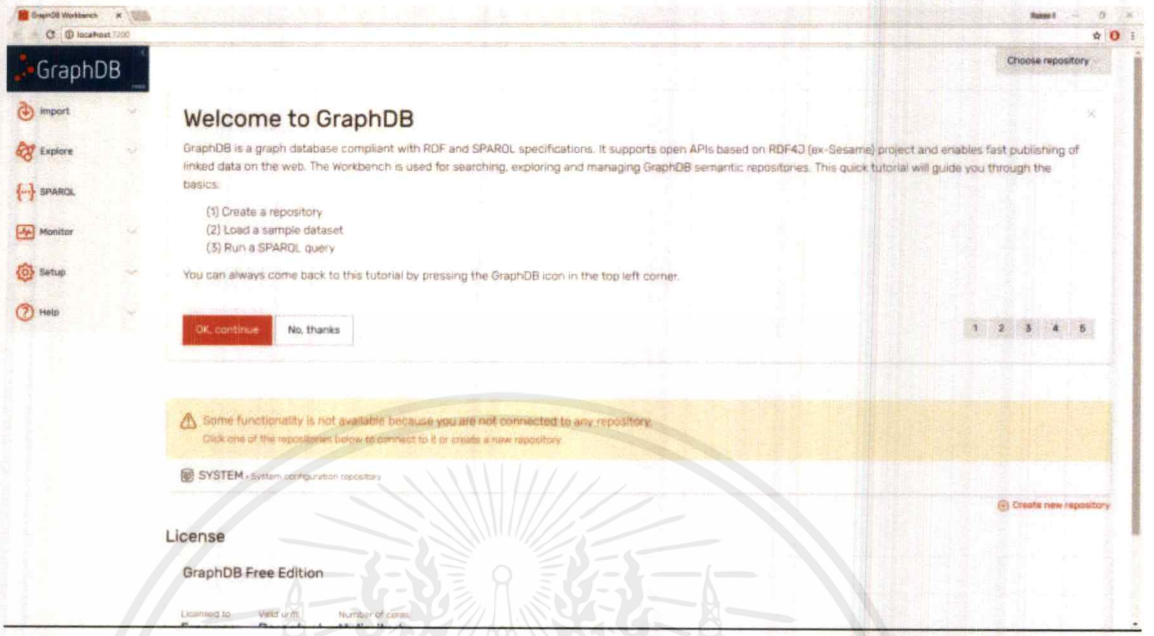
    4.3 สามารถเพิ่มเติมคุณสมบัติและค่าในคุณสมบัติได้

โดยมีการทำงานของระบบฐานข้อมูลดังต่อไปนี้



รูปที่ 3.12 แสดงหน้าต่างที่ใช้ในการเปิด Service ของระบบฐานข้อมูล

จากรูปที่ 3.12 แสดงถึงหน้าต่างรับคำสั่งของระบบฐานข้อมูล และเมื่อต้องการสั่งคำสั่งกับระบบฐานข้อมูลต้องกดไปที่ SPARQL ดังรูปที่ 3.13



รูปที่ 3.13 แสดงหน้าต่างในการรับคำสั่งของระบบฐานข้อมูล

จากรูปที่ 3.13 แสดงหน้าต่างการใช้งานของระบบฐานข้อมูล โดยที่ URL ใส่ค่า localhost:7200

อัลกอริทึมของระบบฐานข้อมูล GraphDB

GraphDB ได้ใช้รูปแบบ Semantic Tagging ในการจัดการรูปแบบความสัมพันธ์ความสอดคล้องกันของกราฟในรูปแบบ RDF และได้ใช้ Concept Extraction Service (CES) เป็นส่วนเสริมในการประมวลผลโดยใช้ tagging ในการอ้างอิงการเชื่อมต่อของกราฟ ซึ่งภายในฐานข้อมูลรองรับการทำ Migration Service เพื่อเพิ่มประสทธิภาพในการจัดการข้อมูลให้มีประสิทธิภาพโดยที่ไม่ต้องใช้งานทรัพยากรที่สูงมาก

Feature ของ GraphDB

เป็นฐานข้อมูลกราฟที่มีประสิทธิภาพสูงและมีประสิทธิภาพพร้อมด้วย RDF และ SPARQL ซึ่งในคู่มือการสอนจะอธิบายถึงคุณลักษณะต่างๆของ GraphDB รวมถึงหัวข้อต่าง ๆ เช่นการตั้งค่าพื้นที่เก็บข้อมูลการโหลดและการทำงานกับข้อมูลการปรับแต่งประสิทธิภาพการปรับขนาดเป็นต้น

3.2.4 ระบบฐานข้อมูล ArangoDB (3.2.9) [13],[28]

รูปที่ 3.14 โลโก้ของระบบฐานข้อมูล ArangoDB

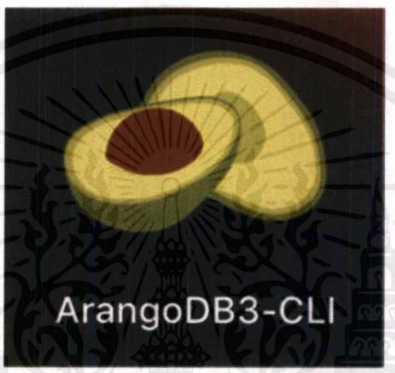
ArangoDB เป็นฐานข้อมูลแบบ Open-source หลายรูปแบบและมีความยืดหยุ่นเป็นฐานข้อมูลกราฟสำหรับการค้นหาข้อความบนกราฟที่เกี่ยวข้องกับเส้นทางที่ไม่เคยรู้มาก่อน เช่น การค้นหาเส้นทางที่สั้นที่สุดระหว่างโหนด หรือการหาเส้นทางทั้งหมดที่ตรงกับรูปแบบที่กำหนด รูปแบบของการเก็บข้อมูลของ ArangoDB แบ่งเป็น 2 ประเภท คือ Key-value และ Document store การทำการค้นหาของ ArangoDB จะใช้ภาษา ArangoDB query language (AQL) เพื่อการดึงและการแก้ไขข้อมูลที่เก็บอยู่ในฐานข้อมูล โดยกระบวนการ query ของ ArangoDB แอพพลิเคชันของ client จะส่งคำสั่ง AQL ไปยังเซิร์ฟเวอร์ของฐานข้อมูล ArangoDB จะทำการรวบรวมชุดของผลลัพธ์ ตัวระบบฐานข้อมูลจะทำการวิเคราะห์การ query และรวบรวมผลส่งกลับไปยัง client ถ้าการ query ไม่ถูกต้องหรือไม่สามารถดำเนินการได้ เซิร์ฟเวอร์จะส่งคืนข้อผิดพลาดไปยัง client โดยภาษา AQL จะคล้ายกับ SQL ตัวภาษา AQL จะสนับสนุนการอ่านและการแก้ไขข้อมูล แต่ไม่สนับสนุนการดำเนินการข้อมูลที่กำหนด เช่น การสร้างและการวางแผนหรือการสร้าง index เป็นภาษาที่เน้นการทำ data manipulation language (DML) จะไม่เน้นการทำ Data definition language (DDL) หรือ Data control language (DCL)

คุณสมบัติหลักของระบบฐานข้อมูล ArangoDB มีดังนี้

1. การสร้างแบบจำลองข้อมูลแบบยืดหยุ่น : การจำลองข้อมูลเป็นชุดค่าให้เหมาะกับความสัมพันธ์
2. ใช้ ArangoDB เป็น Application Server และ Field Server ได้

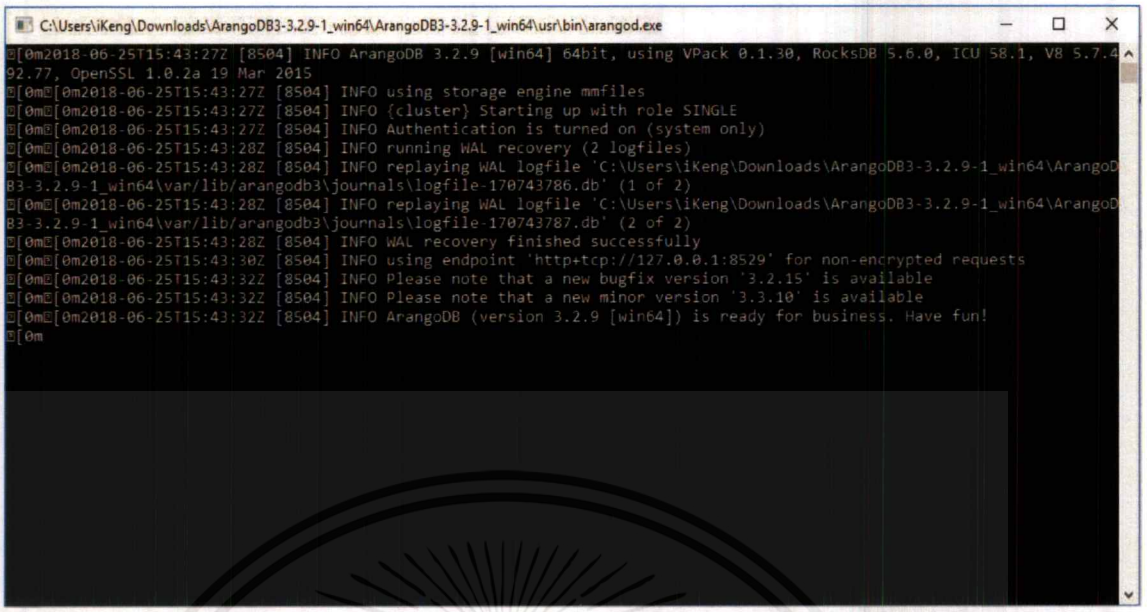
3. การทำ query ที่ละหลายชุดที่มีข้อมูลสอดคล้องกันหรือการแยกการ query สำหรับตัวที่เลือกได้
4. สามารถทำ Replication และ Sharding แบบ Master-salve กระจายไปยังเซิร์ฟเวอร์อื่น
5. เป็น Open-source ของ Apache License 2.0

โดยการทำงานของระบบฐานข้อมูลดังต่อไปนี้



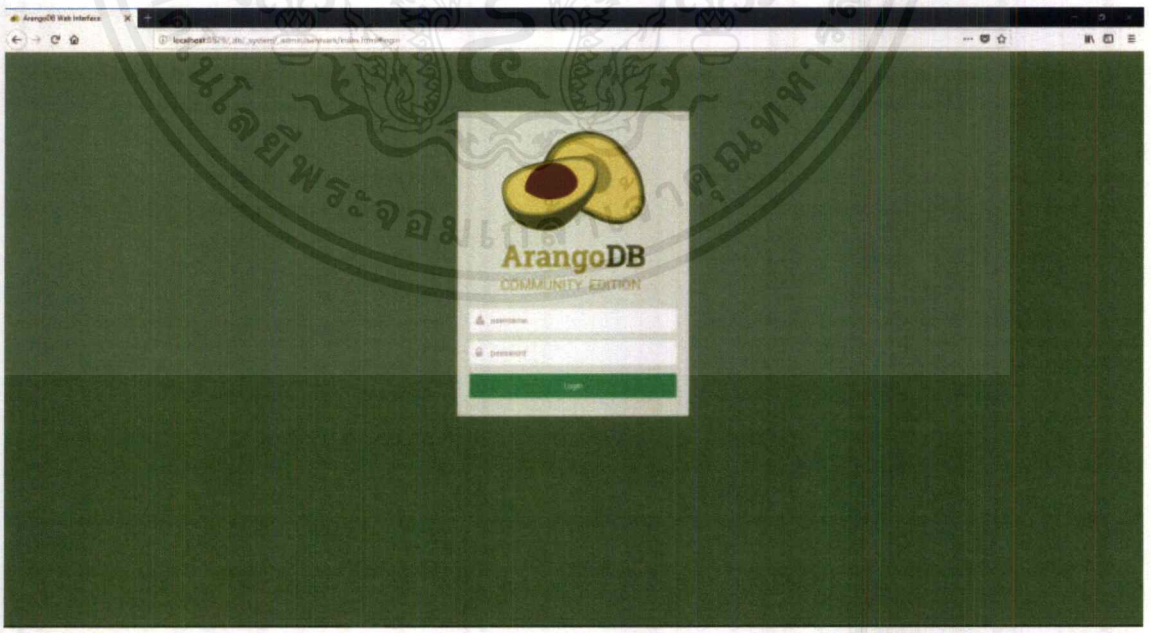
รูปที่ 3.15 แสดงไอคอนในการเปิดระบบฐานข้อมูล

จากรูปที่ 3.15 จะแสดงไอคอนของระบบฐานข้อมูลเพื่อทำการเปิดเซิร์ฟเวอร์ของฐานข้อมูลดังรูปต่อไปนี้



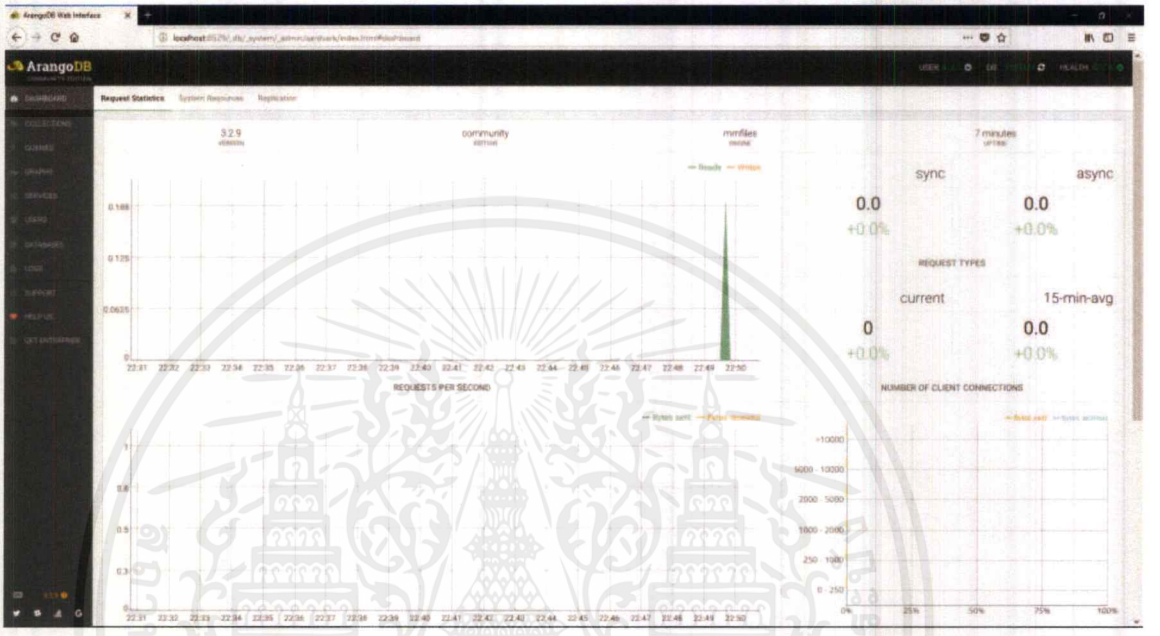
รูปที่ 3.16 แสดงหน้าต่างที่ใช้ในการเปิด Service ของระบบฐานข้อมูล

จากรูปที่ 3.16 ทำการเลือกไฟล์ arangod.exe ในการเปิดการทำงานของ service (local host) arangod.exe จะอยู่ใน directory >> \ArangoDB3-3.2.9-1_win64\ArangoDB3-3.2.9-1_win64\ usr\bin



รูปที่ 3.17 แสดงหน้าต่างเลือกฐานข้อมูล

จากรูปที่ 3.17 แสดงหน้าเข้าสู่ระบบให้ทำการใส่ username และ password ที่ทำการสมัคร โดยค่า Default คือ username ใส่ค่า root และ password ไม่ต้องใส่ค่าใด ๆ จะกด Login เพื่อทำการเข้าสู่ระบบดังรูปต่อไปนี้



รูปที่ 3.18 แสดงหน้าต่างในการรับคำสั่งของระบบฐานข้อมูล

จากรูปที่ 3.18 แสดงหน้าต่างการใช้งานของระบบฐานข้อมูล โดยที่ URL ใส่ค่า localhost:8529

อัลกอริทึมของระบบฐานข้อมูล ArangoDB

ArangoDB เป็นฐานข้อมูลที่มีหลายรูปแบบ (Native multi-model) สามารถรองรับการจัดเก็บประเภท key/value , graphs หรือ documents โดยรองรับการทำงานทั้งหมดในภาษา AQL การเก็บข้อมูลเป็นแบบลำดับชั้นภายในฐานข้อมูลได้ใช้ MMFiles engine เป็นเครื่องมือการนำไฟล์เข้าสู่ระบบฐานข้อมูล ซึ่ง MMFiles engine นั้นได้เพิ่มประสิทธิภาพการทำงานโดย index และโครงสร้างของ index ได้ใช้รูปแบบเดียวกับ Lucene ประกอบไปด้วย 1. Term dictionary ใช้ในการจัดเก็บและค้นหาข้อมูล 2. Segment metadata ใช้ในการจัดการคุณสมบัติ และข้อมูลต่าง ๆ

3. Tombstones ใช้จัดเก็บข้อมูลที่ถูกลบทิ้ง และเก็บข้อมูลเพื่อการกู้คืนข้อมูลได้ในภายหลัง

4. Columnstore ใช้ในการเข้าถึงและจัดเก็บข้อมูลในรูปแบบคอลัมน์

Features ของ ArangoDB

มีประโยชน์ในแง่ของประสิทธิภาพของคลัสเตอร์ ในคลัสเตอร์ที่มี CPU เสมือน 640 ตัวและพบว่าสามารถรองรับการเขียนเอกสาร JSON จำนวน 1.1 ล้านฉบับได้เทียบเท่า 1 GB ต่อวินาที และยังสามารถใช้บนระบบปฏิบัติการ Mesosphere Datacenter (DCOS)

3.3 การวัดประสิทธิภาพของโปรแกรม

3.3.1 การเปรียบเทียบคำศัพท์ระหว่าง RDBMS กับ Graph-based databases

การเปรียบเทียบคำศัพท์ของฐานข้อมูลระหว่าง RDBMS กับ NoSQL ประเภทกราฟทั้งหมด 4 ระบบฐานข้อมูล คือ Neo4j, OrientDB, GraphDB และ ArangoDB โดยแต่ละแถวจะมีคุณสมบัติที่แตกต่างกันดังตารางที่ 3.1

ตารางที่ 3.1 การเปรียบเทียบคำศัพท์ระหว่าง RDBMS กับ Graph-based databases

RDBMS & OrientDB	Neo4j	GraphDB	ArangoDB
Table	Label	Class	Collection
Row (tuple)	Node	Subject	Document
Column (attribute)	Properties	Properties	Attribute
Tuple value	Value	Object	Value
Key-value (Primary Key)	-	URI	Primary Key
Indexes	Indexes	Indexes	Indexes

3.3.2 การเปรียบเทียบคุณสมบัติพื้นฐานของ Graph-based NoSQL ที่นำมาใช้กับ RDBMS

การเปรียบเทียบคุณสมบัติพื้นฐานของฐานข้อมูลระหว่าง RDBMS กับ NoSQL ประเภทกราฟ ทั้งหมด 4 ระบบฐานข้อมูล คือ Neo4j, OrientDB, GraphDB และ ArangoDB โดยแต่ละแถวจะมีคุณสมบัติที่แตกต่างกัน ดังตารางที่ 3.2 [9], [15-18]

ตารางที่ 3.2 แสดงการเปรียบเทียบคุณสมบัติพื้นฐาน

Name	RDBMS	Neo4j	OrientDB	GraphDB	ArangoDB
Primary database model	Relational DBMS	Graph DBMS	Document store Graph DBMS Key-value store	Graph DBMS RDF store	Document store Graph DBMS Key-value store
Additional database models	Key-value store Document store	Key-value store	Document store	Column store	Key-value store Document store
Developer	Oracle	Neo4j, Inc.	OrientDB LTD; CallidusCloud	ontotext	ArangoDB GmbH / triagens GmbH
Implementation language	C and C++	Java , Scala	Java	Java	C, C++, Script
Query langyage	SQL	CQL	SQL	Sparql	AQL
Sever operating systems	FreeBSD Linux OS X Solaris Windows	Linux OS X Solaris Windows	All OS with a Java JDK (>= JDK 6)	All OS with a Java VM Linux OS X Windows	Linux OS X Raspbian Solaris Windows

Name	RDBMS	Neo4j	OrientDB	GraphDB	ArangoDB
Database-schema	Yes	Schema-free	Schema-free	schema-free and OWL/RDFS-schema support	Schema-free
APIs and other access methods	ADO.NET JDBC ODBC	Cypher query language Java API Neo4j-OGM RESTful HTTP API Spring Data Neo4j TinkerPop 3	Java API RESTful HTTP/JSON API TinkerPop technology stack with Blueprints, Gremlin , pipes	GeoSPARQL Java API RDF4J API RIO Sail API Sesame REST HTTP Protocol SPARQL 1.1	HTTP API JSON style queries
Supported programming Languages	C# C++ Java JavaScript (Node.js) PHP Python Etc.	.Net Java JavaScript (Node.js) PHP Python Etc.	.NET C# Java JavaScript (Node.js) PHP Python Etc.	.Net C# Java JavaScript (Node.js) PHP Python Scala	C# Java JavaScript (Node.js) PHP Python

3.3.3 ตัวชี้วัดประสิทธิภาพการทำงาน

การวัดประสิทธิภาพของระบบฐานข้อมูล ตัวชี้วัดที่ทำมาใช้ในการทดลองจะขึ้นอยู่กับการตอบสนองในระหว่างการประมวลผล โดยจะเก็บผลของการใช้งานทรัพยากรจากการบันทึกหน้าจอ ซึ่งใช้โปรแกรม OBS Studio ที่ช่วยในการบันทึก เพื่อนำไปหาค่าเฉลี่ยของการใช้งานทรัพยากรมีดังนี้

1) วัดหน่วยความจำที่ใช้ระหว่างทำการประมวลผล เนื่องจากการใช้งานหน่วยความจำทำให้ประสิทธิภาพของระบบฐานข้อมูลอื่นที่กำลังทำงานช้าลง

2) วัดเวลาในการทำงานที่ผ่านไป และการใช้ CPU ของระบบฐานข้อมูลแต่ละระบบฐานข้อมูล เนื่องจากเวลา และ CPU ที่ใช้งานจะบอกถึงประสิทธิภาพของระบบฐานข้อมูลในระหว่างการประมวลผล

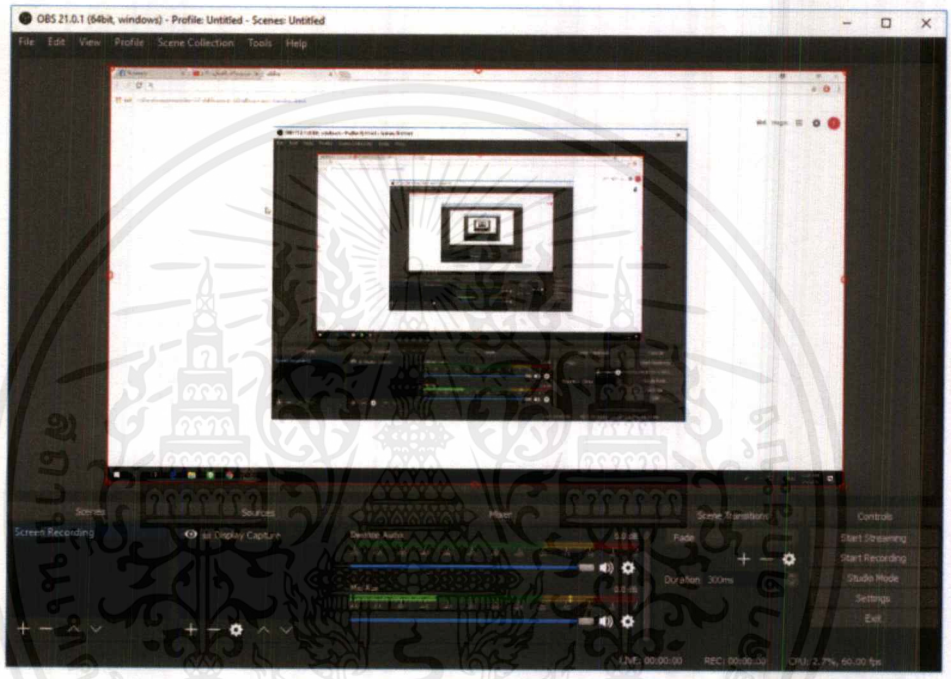
3) วัดการใช้งานทรัพยากรของระบบฐานข้อมูลอื่นที่ทำงานระหว่างการประมวลผล เนื่องจากการประมวลผลของฐานข้อมูล อาจใช้งานทรัพยากรมากขึ้นจากการใช้งานระบบฐานข้อมูลอื่นช่วยในการประมวลผล

โปรแกรม OBS Studio (21.0.1)

รูปที่ 3.19 แสดงไอคอนในการเปิดระบบฐานข้อมูล

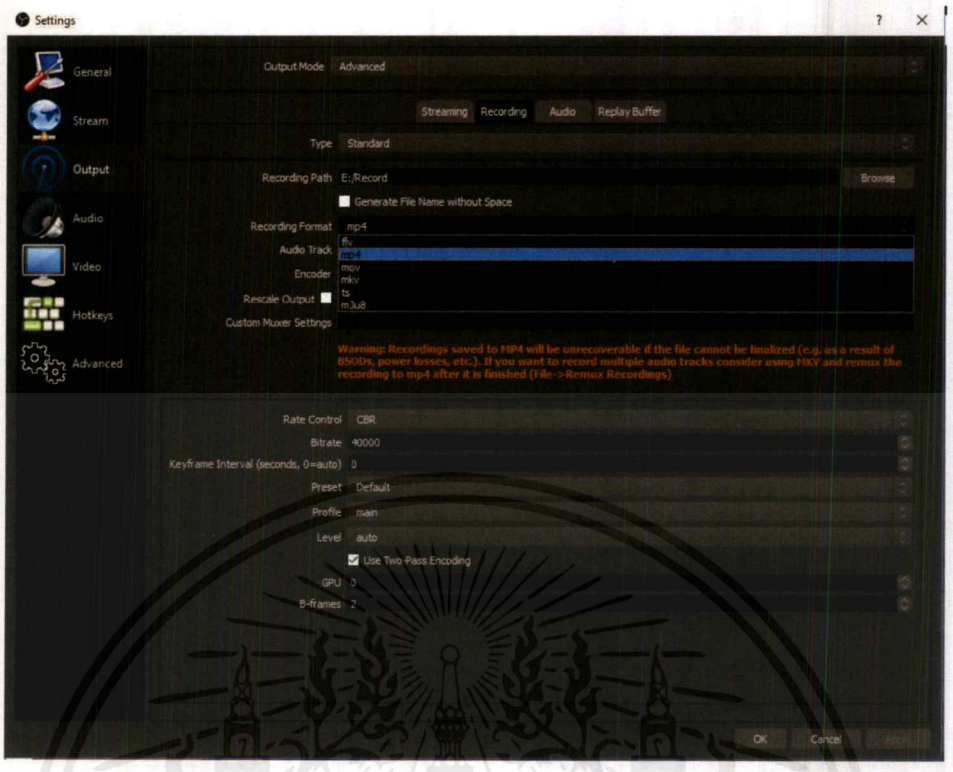
ระบบฐานข้อมูล OBS Studio หรือ Open Broadcaster Software Studio คือ ระบบฐานข้อมูลสำหรับอัดวีดีโอหน้าจอคอมพิวเตอร์ ซึ่งอาจจะเป็นการเล่นเกมส์คอมพิวเตอร์ หรือสาธิตการใช้งานซอฟต์แวร์ ระบบฐานข้อมูล หรือ แอพพลิเคชั่น เพื่อทำการถ่ายทอดสดผ่าน YouTube Live (บริการถ่ายทอดสดของยูทูป) หรือแม้แต่ Facebook Live (บริการถ่ายทอดสดของเฟสบุ๊ค)

อีกทั้งยังสามารถรวมไฟล์ภาพและไฟล์เสียงพากย์สด จากกล้องเว็บแคมเข้าไปเป็นส่วนหนึ่งของการถ่ายทอดสด ซึ่งทางผู้พัฒนา OBS Studio ได้พัฒนาระบบฐานข้อมูลเวอร์ชั่นใหม่ที่พัฒนามาจาก OBS Classic เวอร์ชั่นดั่งเดิมเพื่อให้รองรับการใช้งานได้แบบ Multi-platform ใช้งานได้บนหลายแพลตฟอร์ม ไม่ว่าจะเป็น บนระบบปฏิบัติการ Windows Mac OS และ Linux โดยมีสามารถส่งออกไฟล์ MP4 หรือ ไฟล์ FLV เพื่อที่จะเอาไปอัปบนโซเชียล



รูปที่ 3.20 แสดงหน้าต่าง UI

จากรูปที่ 3.20 เป็นหน้าต่าง UI หลักสำหรับใช้ในการสั่งการเลือกบันทึกหรือการถ่ายทอดสดหน้าจอ



รูปที่ 3.21 แสดงการเลือกรูปแบบนามสกุลไฟล์ Output

จากรูปที่ 3.21 แสดงถึงหน้าต่างการตั้งค่าสำหรับไฟล์หลังการบันทึกหรือถ่ายทอดสดหน้าจอเสร็จ ซึ่งจะสามารถเลือกนามสกุลไฟล์ได้หลายแบบ และสามารถเลือกตำแหน่งในการเก็บไฟล์ได้

3.3.4 Queries ที่นำมาใช้ในการทดลองกับข้อมูลประเภท non-graph

การดำเนินการที่ใช้สำหรับข้อมูล Non-graph ทั้งหมด 3 แบบ คือ 1) การเรียกข้อมูลประเภทของหนังสือที่ไม่ซ้ำกัน 2) การเรียกข้อมูลประเภทของหนังสือที่ไม่ซ้ำกัน (เรียงลำดับจากน้อยไปมาก) 3) การเรียกข้อมูลประเภทหนังสือชื่อ “jcbk” ตามตารางที่ 3.2

ตารางที่ 3.3 แสดงคำสั่งในการดำเนินการสำหรับข้อมูล non-graph

Query	ความหมาย	Neo4j	OrientDB	GraphDB	ArangoDB
1	เป็น Query ที่หา attribute ItemType	MATCH (n) RETURN distinct n.ItemType	SELECT DISTINCT(ItemType) FROM book	SELECT DISTINCT ?ItemType WHERE { ?row a mydata:Row ; mydata:ItemType ?ItemType . }	FOR doc IN test RETURN Distinct doc.ItemType
2	เป็น Query ที่หา attribute ItemType โดยจัดเรียงจากน้อยไปมาก	MATCH (n) RETURN distinct n.ItemType ORDER BY n.ItemType	SELECT DISTINCT(ItemType) FROM book ORDER BY ItemType ASC	SELECT DISTINCT ?ItemType WHERE { ?row a mydata:Row ; mydata:ItemType ?ItemType . } ORDER by ASC (?ItemType)	FOR doc IN test SORT doc.ItemType ASC RETURN Distinct doc.ItemType
3	เป็น Query ที่หา attribute ItemType โดยแสดงผลแค่ jcbk	MATCH (n) WHERE n.ItemType = 'jcbk' RETURN n.ItemType	SELECT ItemType FROM book WHERE ItemType LIKE 'jcbk'	SELECT ?ItemType WHERE { ?row a mydata:Row ; mydata:ItemType "jcbk" ; mydata:ItemType ?ItemType . }	FOR doc in test FILTER doc.ItemType == 'jcbk' RETURN doc.ItemType

จากตารางที่ 3.3 ได้แสดงถึงความแตกต่างของแต่ละคุณสมบัติภายในระบบฐานข้อมูลที่สามารถรองรับการทำงานในด้านต่าง ๆ ได้ ซึ่งภาษาที่ใช้ในการ query จะแตกต่างกันทั้งหมด

โดยที่ทางผู้พัฒนาระบบฐานข้อมูลได้ออกแบบภาษาขึ้นมาเพื่อการใช้งานของแต่ละระบบฐานข้อมูล ซึ่งจะเป็นภาษาเฉพาะระบบฐานข้อมูลเท่านั้น ซึ่งแต่ละระบบฐานข้อมูลสามารถใช้ภาษาหลักที่ใช้สร้างในการ query ได้ เช่น Neo4j สามารถใช้ภาษา Java ในการ query ได้และ ArangoDB สามารถใช้ภาษา C ในการ query ได้ เป็นต้น

3.3.5 Queries ที่นำมาใช้ในการทดลองกับข้อมูลประเภท graph

ในงานวิจัยนี้ได้ทำการเปรียบเทียบ query ในการดำเนินการทางกราฟที่ใช้งานทั้งหมดทั้งหมด 3 แบบ คือ การค้นหาแบบแนวลึก (Depth-First Search : DFS) การค้นหาแบบแนวกว้าง (Breadth-First Search : BFS) และเส้นทางที่สั้นที่สุด (Shortest Path) เพื่อแสดงให้เห็นถึงความต่างในการค้นหาของประสิทธิภาพของแต่ละระบบฐานข้อมูลตารางที่ 3.4

ตารางที่ 3.4 แสดงคำสั่งในการดำเนินการสำหรับข้อมูล graph

Query	Neo4j	OrientDB	GraphDB	ArangoDB
1 (Depth)	MATCH (a:Europe)-[*1..20]-(b:Team) WHERE b.team_short_name ='MUN' AND a.Country = 'england' RETURN a,b LIMIT 10000	SELECT * FROM (TRAVERSE OUT FROM Europe WHILE Country = 'england')	-	-
2 (Breadth)	MATCH (b:Team) WHERE b.team_short_name ='MUN' CALL apoc.path.expandConfig (b,{maxLevel:10}) YIELD path WITH b,RELATIONSHIPS(path) as r, LAST(NODES(path)) as a WHERE a:Europe AND a.Country ='england' RETURN b,a LIMIT 300	SELECT * FROM (TRAVERSE OUT FROM Europe WHILE Country = 'england' STRATEGY BREADTH_FIRST)	-	-

3 (Shortest)	MATCH p=shortestPath((a:Europe) -[*1..20]-(b:Team)) WHERE a.Country ='england'AND b.team_short_name = 'MUN' RETURN a,b,length(p) LIMIT 50000	SELECT shortestPath(#54:2, #40:36,OUT)	-	-

จากตารางที่ 3.4 ได้แสดงถึง query ทั้งหมด 3 แบบที่แตกต่างกันโดยที่ query ที่ 1 คือ การท่องโหนดแบบ DFS โดยเริ่มจากโหนด Europe ที่ประเทศ England ไปยังโหนดปลายทางของข้อมูลทีม MUN (Manchester United) ซึ่งจะบอกถึงรายละเอียดของทีมและรายละเอียดของแมตช์ที่แข่งขัน ส่วน query ที่ 2 คือการท่องโหนดแบบ BFS โดยเริ่มจากโหนด Europe ที่ประเทศ England ไปยังโหนดปลายทางของข้อมูลทีม MUN (Manchester United) ซึ่งจะบอกถึงรายละเอียดของทีมและรายละเอียดของแมตช์ที่แข่งขัน และ query ที่ 3 คือการท่องโหนดแบบ Shortest Path โดยเริ่มจากโหนด Europe ที่ประเทศ England ไปยังโหนดปลายทางของข้อมูลทีม MUN (Manchester United) ซึ่งจะบอกถึงรายละเอียดของเส้นทางที่ทำให้เป็น Shortest Path

ส่วนระบบฐานข้อมูล GraphDB ที่ไม่มี syntax เพราะว่าไม่สามารถทำได้ เนื่องจาก GraphDB ใช้การค้นหาแบบ Full-text search และ RDF search จึงไม่สามารถนำมาเปรียบเทียบได้ และ ArangoDB ไม่สามารถเขียน query เพื่อให้วนสร้างเส้นเชื่อมแบบระบบฐานข้อมูล Neo4j ได้ ซึ่งสามารถสร้างเส้นเชื่อมได้แบบจับคู่ทีละโหนดเท่านั้น

บทที่ 4
ผลการวิจัย

ในบทนี้จะกล่าวถึงผลการดำเนินงานวิจัย ผู้วิจัยได้นำระบบฐานข้อมูลทั้ง 4 ระบบฐานข้อมูล มาทำการ query โดยจะใช้ข้อมูลแบบ graph ขนาด 1.5 GB และ non-graph ทั้งหมด 4 ขนาด คือ 1 GB, 3 GB, 5 GB และ 7 GB มาใช้ในการทดลอง ซึ่งได้ผลลัพธ์ของการวิจัย ดังนี้

4.1 ผลการวิจัยของข้อมูล non-graph

4.1.1 ผลการนำข้อมูลเข้าระบบฐานข้อมูล non-graph

4.1.2 ผลการทดลองของการ query สำหรับข้อมูล non-graph

4.1.3 ผลการเปรียบเทียบการใช้งานทรัพยากรของข้อมูลแบบ non-graph

4.2 ผลการวิจัยของข้อมูล graph

4.2.1 การเปรียบเทียบการนำข้อมูลเข้าในระบบฐานข้อมูล

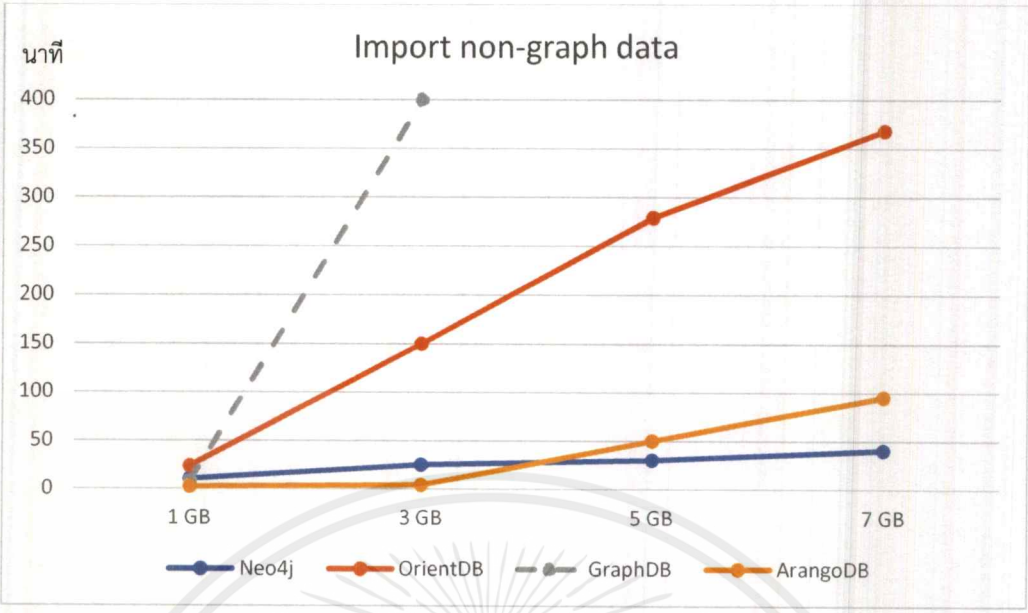
4.2.2 ผลการทดลองของการ query สำหรับข้อมูล graph

4.2.3 ผลการเปรียบเทียบการใช้งานทรัพยากรของข้อมูลแบบ graph

4.1 ผลการวิจัยของข้อมูล non-graph

4.1.1 ผลการนำข้อมูลเข้าระบบฐานข้อมูล non-graph

โดยการทดลองจะมีการนำข้อมูลเข้าระบบฐานข้อมูลที่ใช้ทั้งหมด 4 ระบบฐานข้อมูล ดังรูปที่ 4.1



รูปที่ 4.1 กราฟแสดงการนำข้อมูลเข้าระบบฐานข้อมูล

จากรูปที่ 4.1 แสดงถึงการนำข้อมูลเข้าโดยแกน x คือขนาดของข้อมูล และแกน y คือ เวลาที่ใช้ในการนำเข้าข้อมูล ซึ่งแสดงให้เห็นการเปลี่ยนแปลงขนาดของข้อมูลที่เปลี่ยนไปตามเวลาที่ใช้ โดยใช้เวลาในการนำเข้าข้อมูลต่างกัน

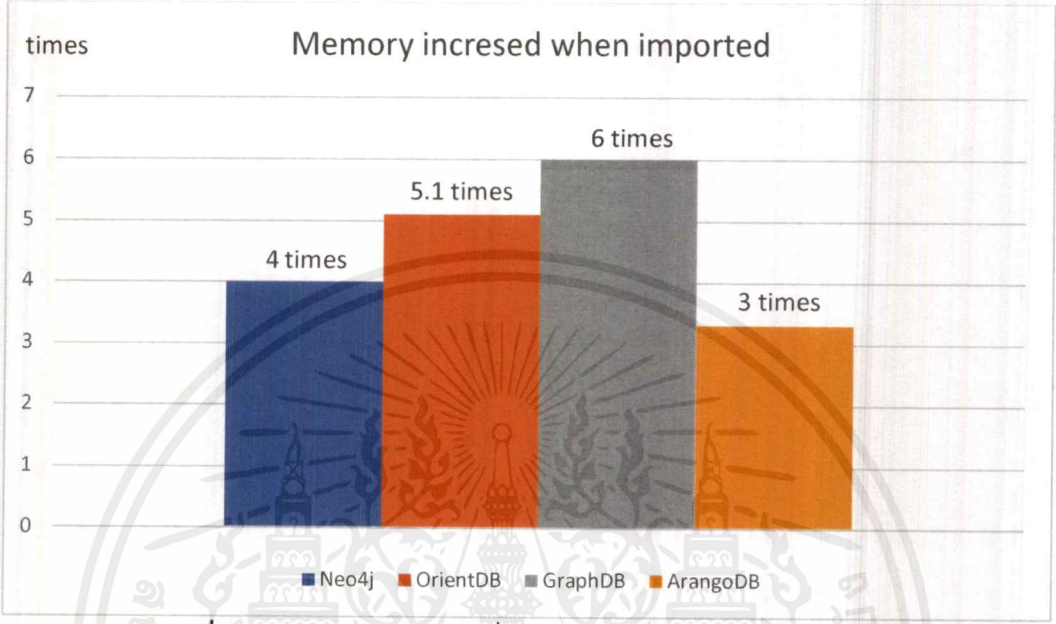
ระบบฐานข้อมูล Neo4j ใช้เวลาค่อนข้างคงที่ตลอดการนำเข้าข้อมูลเมื่อเทียบกับขนาดข้อมูลที่เพิ่มขึ้น โดยใช้เวลาสำหรับนำเข้าข้อมูล 1 GB, 3 GB, 5 GB และ 7 GB โดยใช้เวลา 5 นาที 50 วินาที, 17 นาที 30 วินาที, 29 นาที 17 วินาที และ 40 นาที 58 วินาที ตามลำดับ

ระบบฐานข้อมูล OrientDB จะทำการนำข้อมูลเข้าผ่านระบบฐานข้อมูล Neo4j โดยการนำข้อมูลส่งให้กับระบบฐานข้อมูล OrientDB ซึ่งใช้เวลาค่อนข้างคงที่ตลอดการนำเข้าข้อมูลเมื่อเทียบกับขนาดข้อมูลที่เพิ่มขึ้น โดยใช้เวลาสำหรับนำเข้าข้อมูล 1 GB, 3 GB, 5 GB และ7 GB โดยใช้เวลา 1 ชั่วโมง 18 นาที, 2 ชั่วโมง 30 นาที, 4 ชั่วโมง 39 นาที และ 6 ชั่วโมง 8 นาที ตามลำดับ

ระบบฐานข้อมูล GraphDB สามารถนำเข้าข้อมูลขนาดได้เพียงขนาด 1 GB ใช้เวลาในการนำเข้าข้อมูล 6 นาที 41 วินาที สำหรับข้อมูลที่มีขนาดมากกว่า 1 GB ขึ้นไปทางผู้วิจัยได้สรุปว่าทรัพยากรที่ใช้ในการทำวิจัยครั้งนี้ไม่เพียงพอต่อการนำเข้าข้อมูล และระยะเวลาในการนำข้อมูลมากกว่า 1 GB ใช้เวลามากกว่า 6 ชั่วโมงทางผู้วิจัยไม่สามารถรอผลการทดลองได้

ระบบฐานข้อมูล ArangoDB ใช้เวลาในการนำเข้าข้อมูลมากขึ้นตามขนาดข้อมูลที่มีขนาดใหญ่ขึ้นเมื่อเทียบกับขนาดข้อมูลใช้เวลาสำหรับนำเข้าข้อมูล 1 GB, 3 GB, 5 GB และ7 GB โดยใช้

เวลา 2 นาที, 6 นาที 22 วินาที, 1 ชั่วโมง 30 นาที และ 4 ชั่วโมง 29 นาที ตามลำดับ โดยการนำข้อมูลเข้าระบบฐานข้อมูลจะมีการเติบโตของข้อมูลจะแสดงผล ดังรูปที่ 4.2



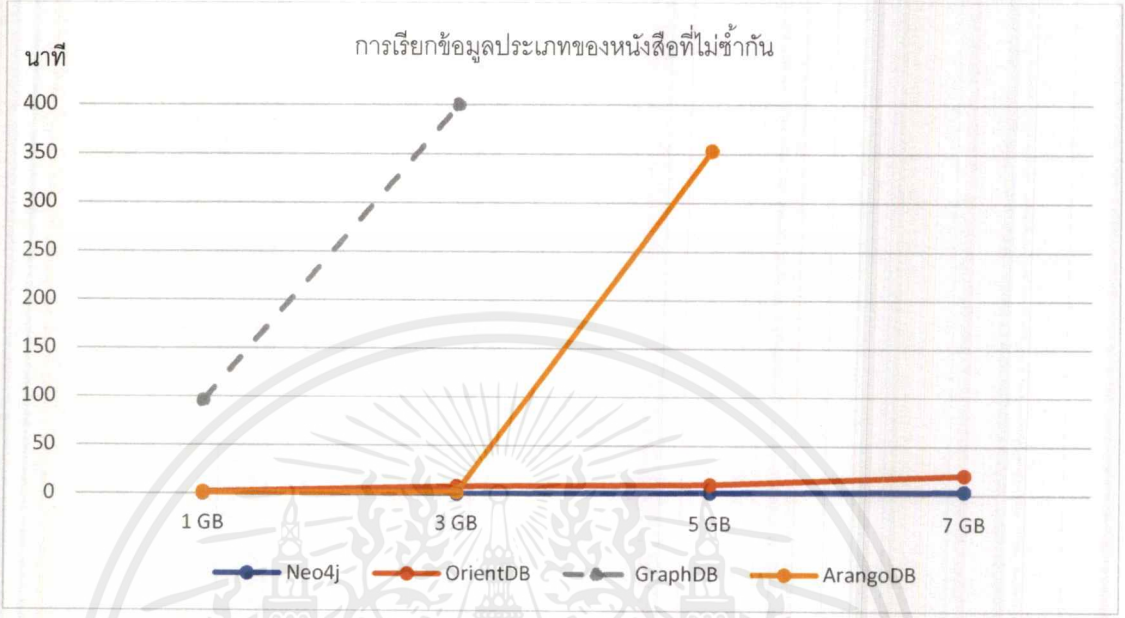
รูปที่ 4.2 การเติบโตของข้อมูลที่หลังนำข้อมูลเข้าระบบฐานข้อมูลแล้ว

จากรูปที่ 4.2 แสดงให้เห็นถึงการเติบโตของข้อมูล non-graph ภายในระบบฐานข้อมูลโดยระบบฐานข้อมูลที่มีการเติบโตน้อยที่สุด คือ ระบบฐานข้อมูล ArangoDB, Neo4j, OrientDB และ GraphDB มีอัตราการเติมโต 3.3, 4, 5.1, 6 เท่า ตามลำดับ

4.1.2 ผลการทดลองของการ query สำหรับข้อมูล non-graph

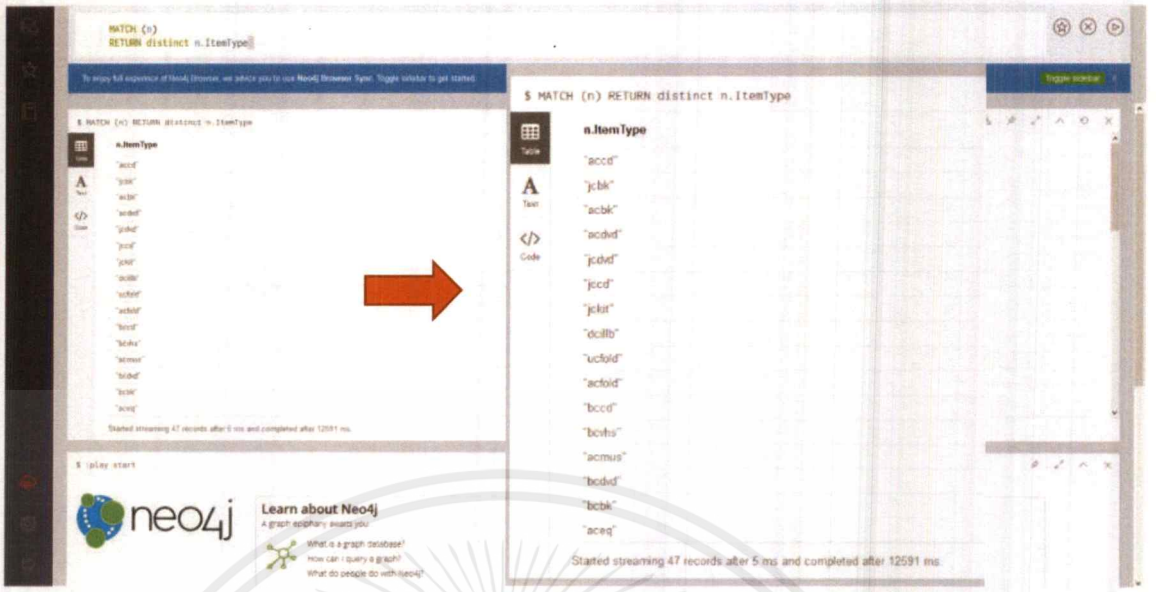
การทดสอบจะทำการทดลองทั้งหมด 3 แบบ คือ 1) การเรียกข้อมูลประเภทของหนังสือที่ไม่ซ้ำกัน 2) การเรียกข้อมูลประเภทของหนังสือที่ไม่ซ้ำกัน (เรียงลำดับจากน้อยไปมาก) 3) การเรียกข้อมูลประเภทหนังสือชื่อ “jcbk”

1) การเรียกข้อมูลประเภทของหนังสือที่ไม่ซ้ำกัน

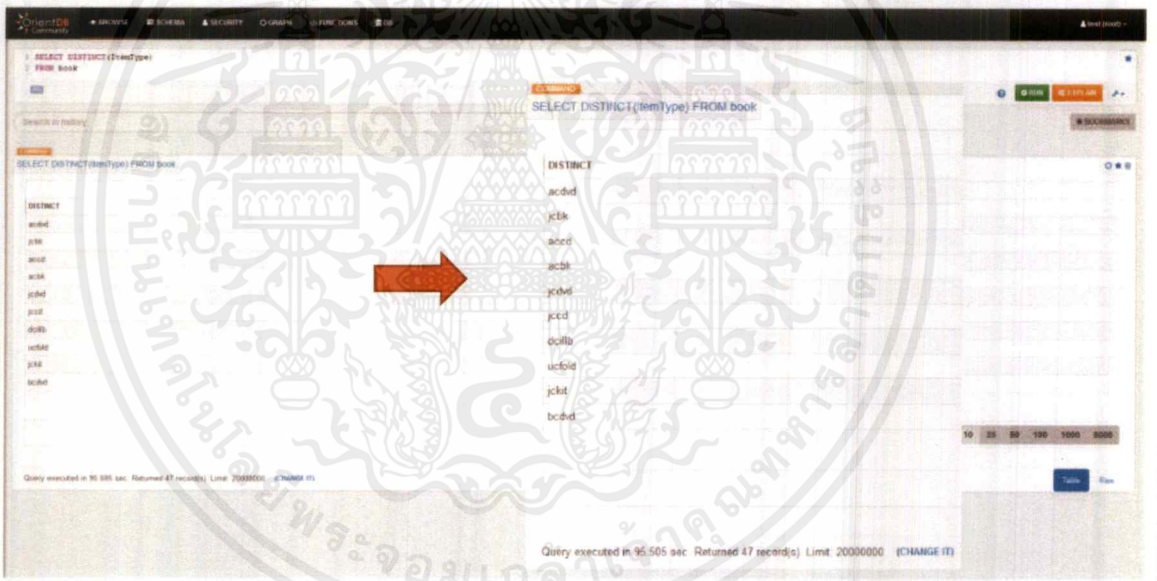


รูปที่ 4.3 กราฟแสดงผลการ query แบบที่ 1

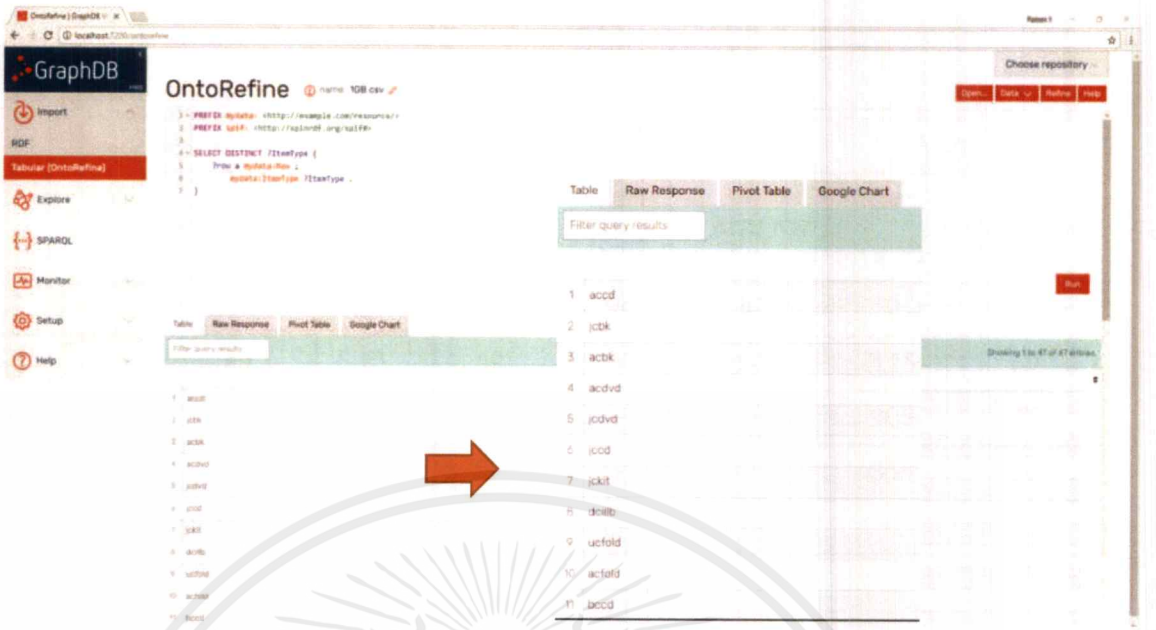
จากรูปที่ 4.3 แสดงถึงที่ได้จากการรัน query แบบที่ 1 โดยแกน x คือขนาดของข้อมูล และแกน y คือเวลาที่ใช้ในการประมวลผล (นาที) query จะสังเกตได้ว่าความเร็วในการ query ข้อมูลจะใช้เวลาเพิ่มขึ้นตามขนาดของทุก NoSQL ข้อมูลขนาด 1 GB, 3 GB Neo4j, OrientDB และ ArangoDB ใช้เวลาไล่เลี่ยกัน ในขณะที่ GraphDB ใช้เวลาในการประมวลผลเป็นเวลานาน และขนาดข้อมูลตั้งแต่ 5 GB และ 7 GB Neo4j, OrientDB ยังต้องใช้เวลามากขึ้นเล็กน้อย แต่ ArangoDB ใช้เวลาในการประมวลผลแบบ exponential ส่วน GraphDB ตั้งแต่ข้อมูล 3 GB ขึ้นไปไม่สามารถรันออกได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการ query ข้อมูล และใช้เวลามากกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ โดยแสดงผลการ query แบบที่ 1 ในแต่ละระบบฐานข้อมูลดังรูปที่ 4.4



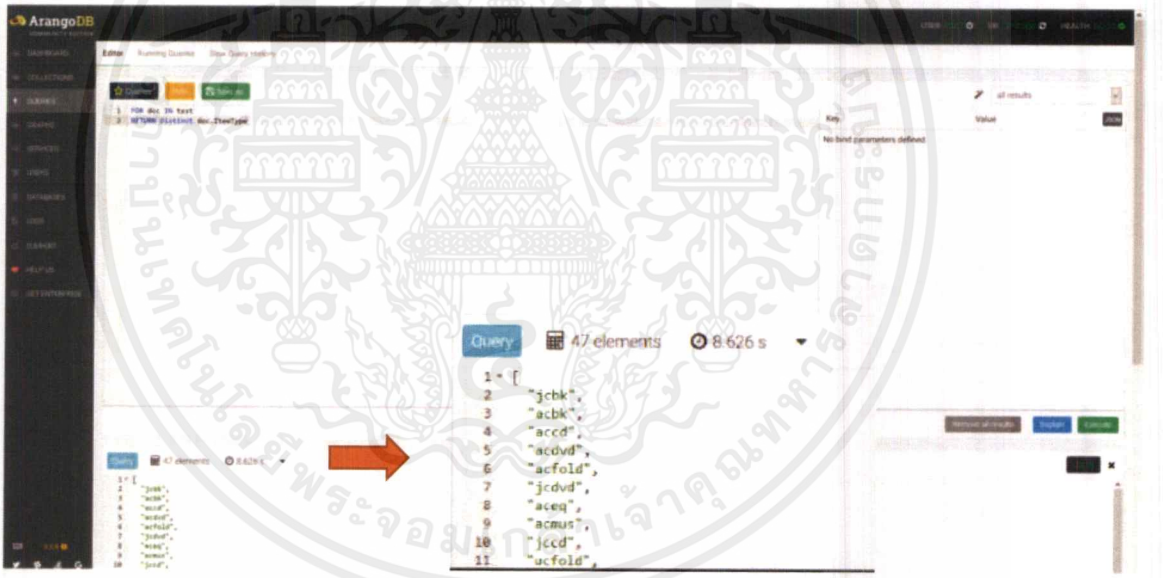
(A) แสดงผลการ query แบบที่ 1 ของ Neo4j



(B) แสดงผลการ query แบบที่ 1 ของ OrientDB



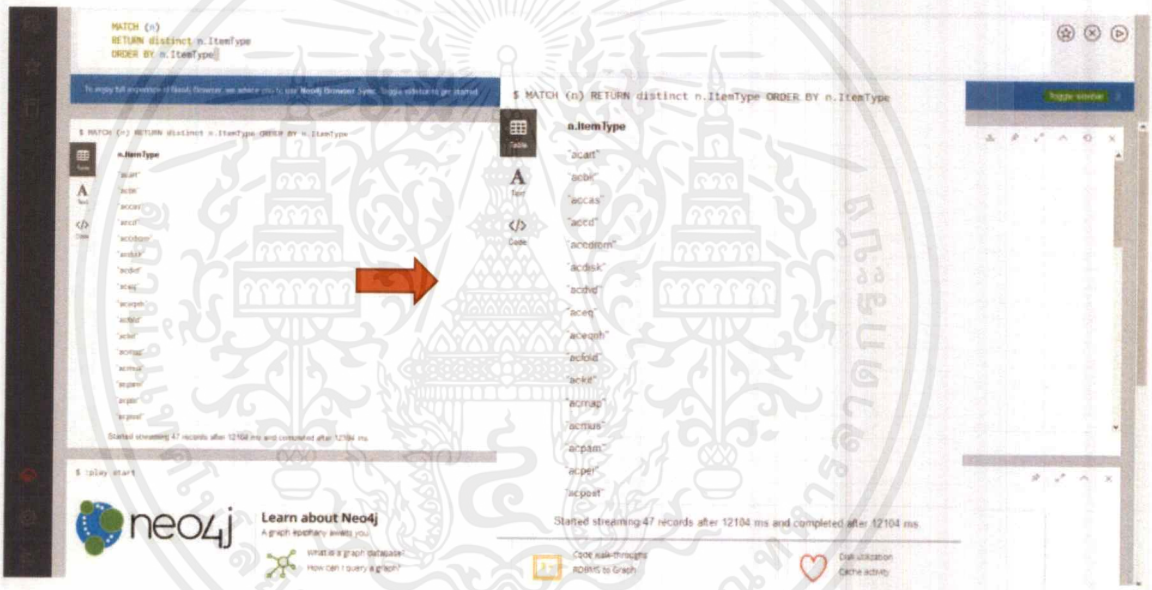
(C) แสดงผลการ query แบบที่ 1 ของ GraphDB



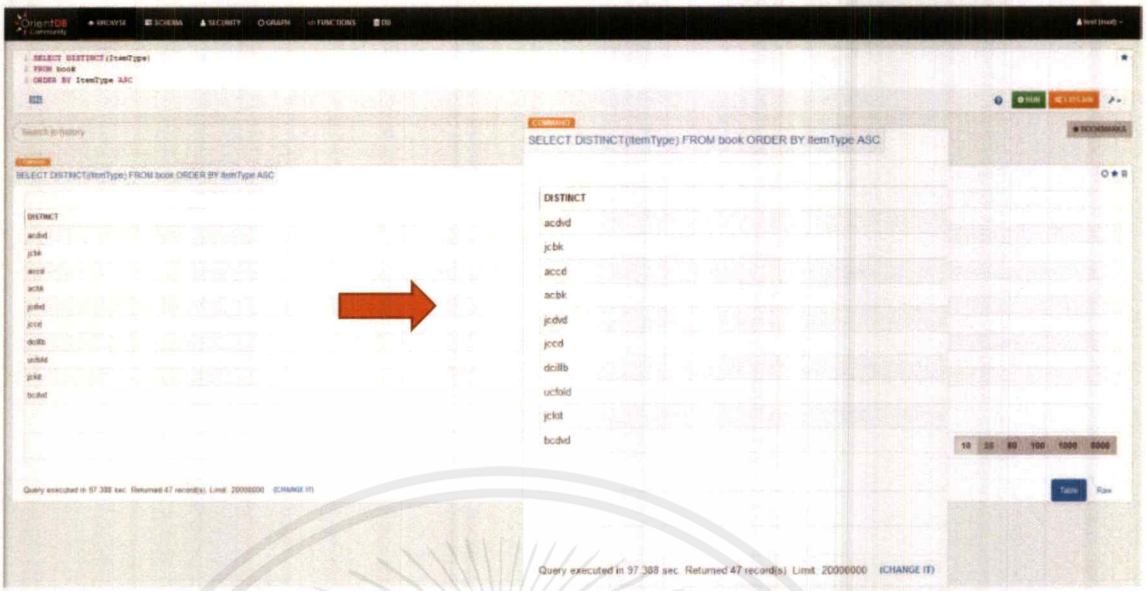
(D) แสดงผลการ query แบบที่ 1 ของ ArangoDB

รูปที่ 4.4 แสดงตัวอย่างผลที่ได้จากการ query แบบที่ 1 บน Neo4j (A), OrientDB (B), GraphDB (C) และ ArangoDB (D)

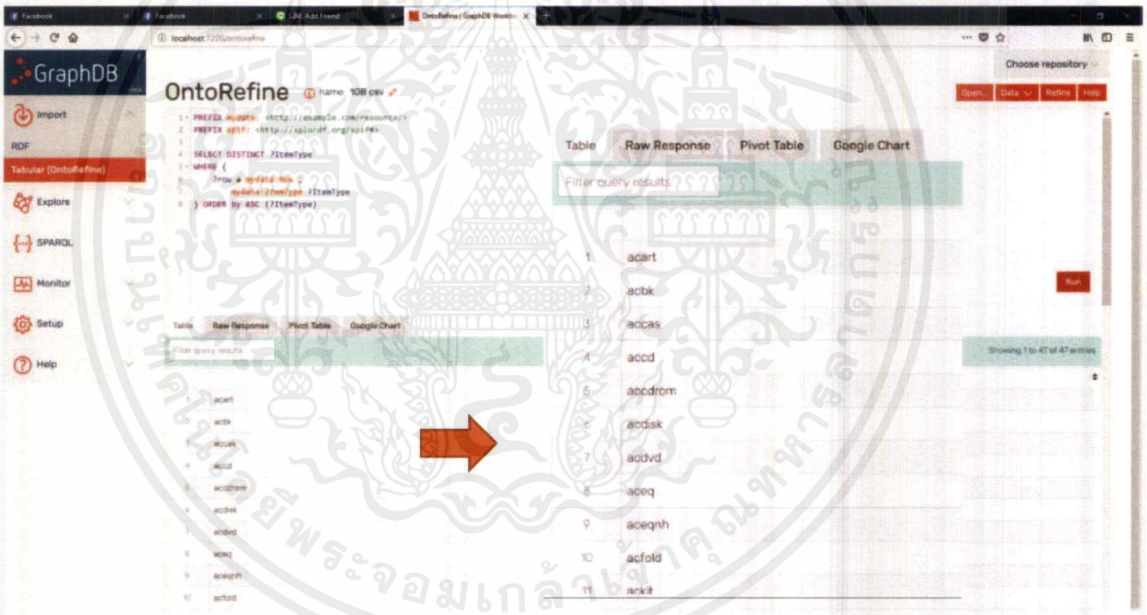
จากรูปที่ 4.5 แสดงถึงที่ได้จากการรัน query แบบที่ 2 โดยแกน x คือขนาดของข้อมูล และแกน y คือเวลาที่ใช้ในการประมวลผล (นาที) query จะสังเกตได้ว่าความเร็วในการ query ข้อมูลจะใช้เวลาเพิ่มขึ้นตามขนาดของทุก NoSQL ข้อมูลขนาด 1 GB, 3 GB Neo4j, OrientDB และ ArangoDB ใช้เวลาไล่เลี่ยกัน ในขณะที่ GraphDB ใช้เวลาในการประมวลผลเป็นเวลานาน และขนาดข้อมูลตั้งแต่ 5 GB และ 7 GB Neo4j, OrientDB ยังต้องใช้เวลามากขึ้นเล็กน้อย แต่ ArangoDB ใช้เวลาเพิ่มขึ้น เมื่อเทียบกับขนาดของข้อมูล ส่วนGraphDB ตั้งแต่ข้อมูล 3 GB ขึ้นไปไม่สามารถรันออกได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการ query ข้อมูล และใช้เวลามากกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ โดยแสดงผลการ query แบบที่ 2 ในแต่ละระบบฐานข้อมูล ดังรูปที่ 4.6



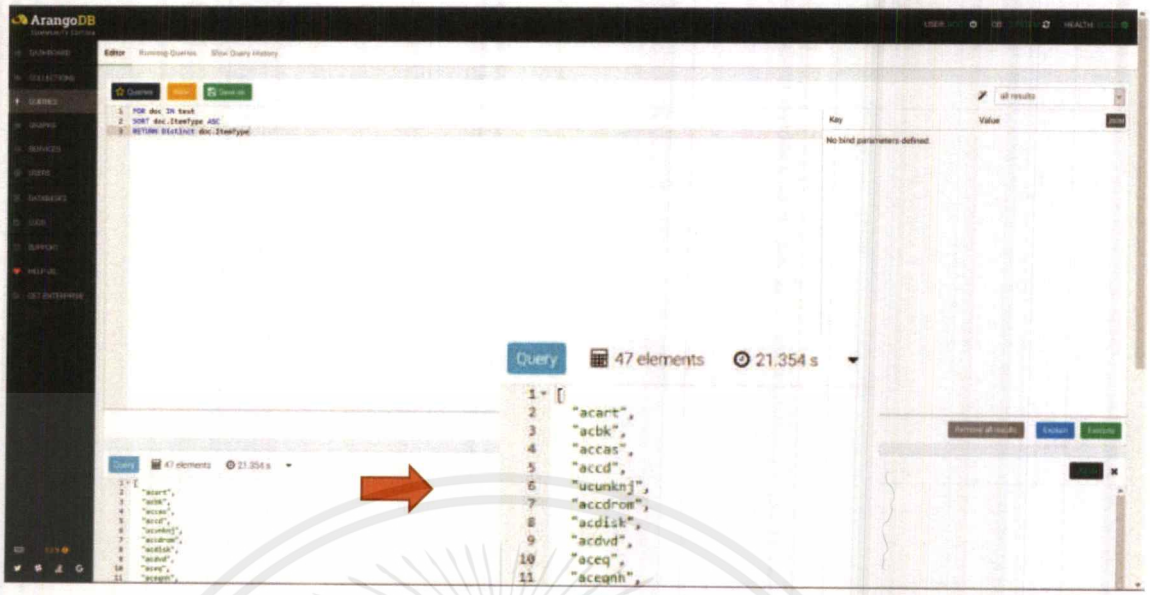
(A) แสดงผลการ query แบบที่ 2 ของ Neo4j



(B) แสดงผลการ query แบบที่ 2 ของ OrientDB



(C) แสดงผลการ query แบบที่ 2 ของ GraphDB



(D) แสดงผลการ query แบบที่ 2 ของ ArangoDB

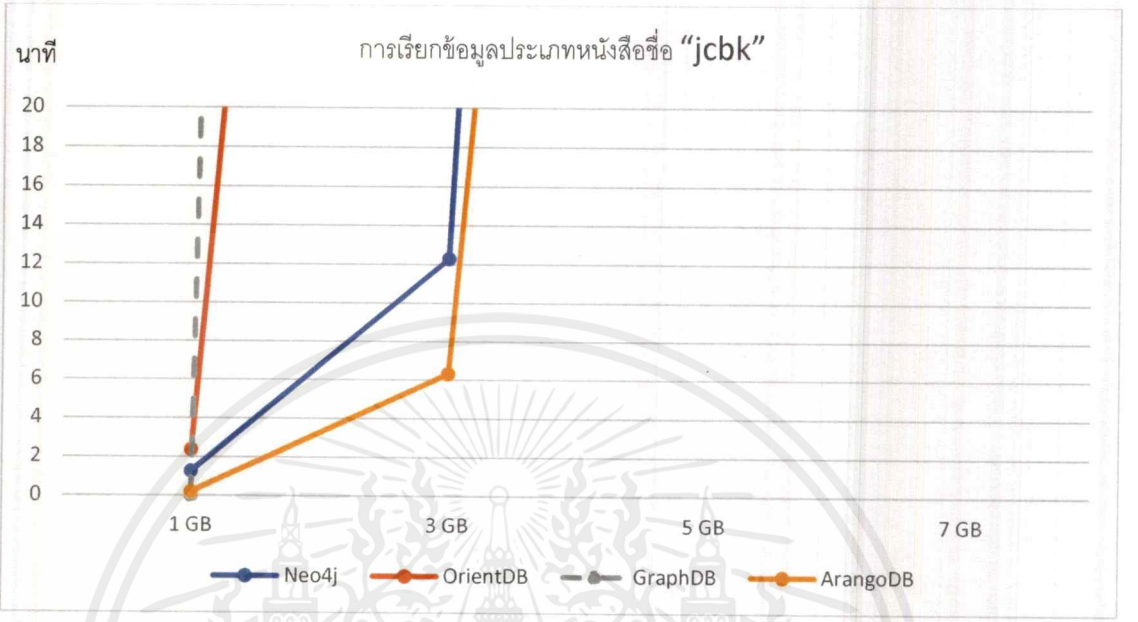
รูปที่ 4.6 แสดงตัวอย่างผลที่ได้จากการ query แบบที่ 2 บน Neo4j (A), OrientDB (B), GraphDB (C) และ ArangoDB (D)

ตารางที่ 4.2 ตารางแสดงผลของ query แบบที่ 2

Q2	Neo4j		OrientDB		GraphDB		ArangoDB	
	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row
1 GB	0.21	47	1.76	47	60.91	47	0.35	47
3 GB	1.08	69	5.18	69	-	-	2.13	69
5 GB	2.31	73	9.28	73	-	-	244	73
7 GB	3.58	89	20	90	-	-	353	90
หมายเหตุ : - คือ ทรัพยากรไม่เพียงพอ จึงไม่สามารถทำการทดลอง query ได้								

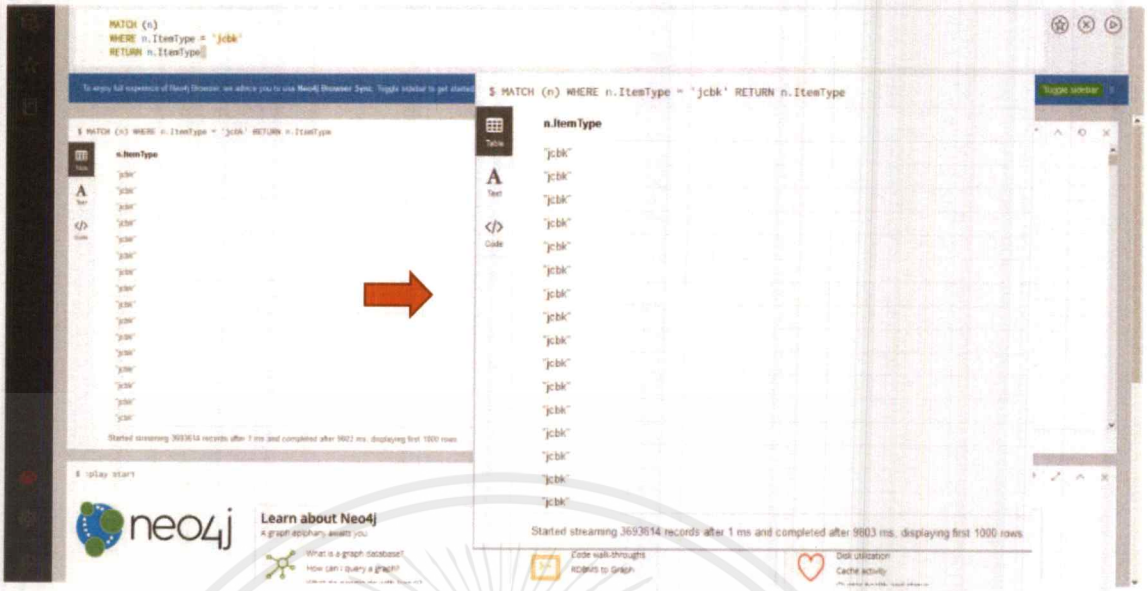
จากตารางที่ 4.2 แสดงถึงเวลาและจำนวน row ของ query แบบที่ 2 โดยมีขนาดข้อมูล 1 GB, 3 GB, 5 GB และ 7 GB ตามลำดับ

3) การเรียกข้อมูลประเภทหนังสือชื่อ “jcbk”

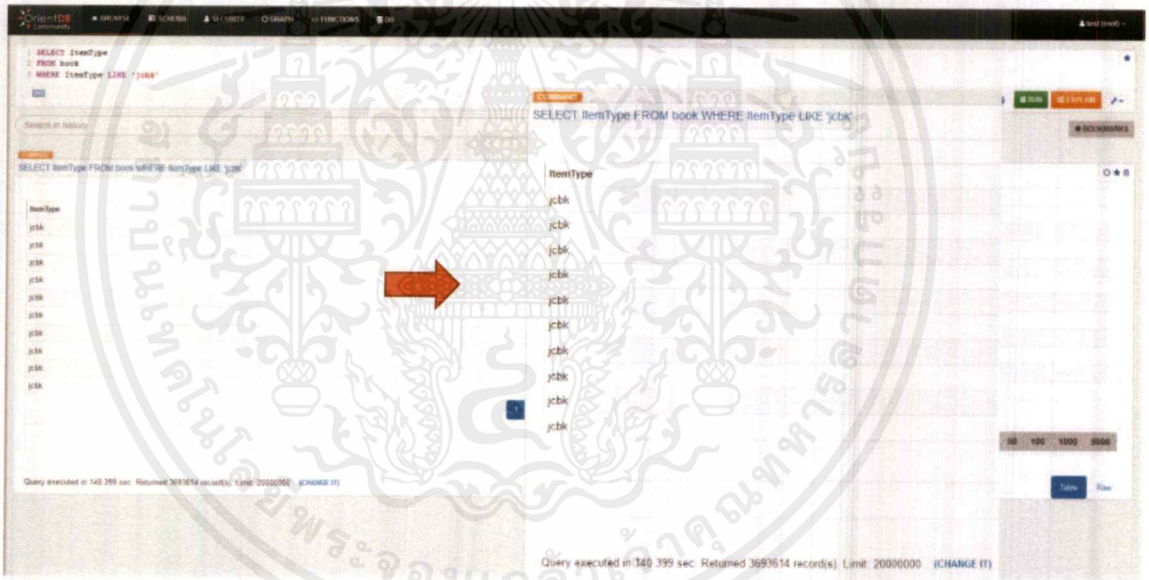


รูปที่ 4.7 กราฟแสดงผลการ query แบบที่ 3

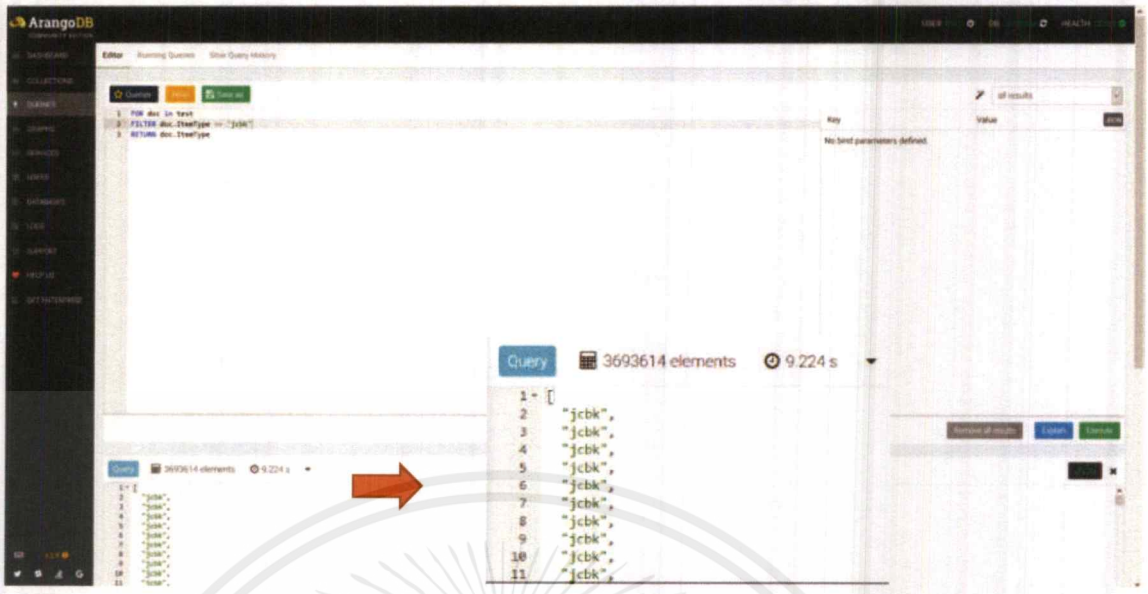
จากรูปที่ 4.7 แสดงถึงที่ได้จากการรัน query แบบที่ 3 โดยแกน x คือขนาดของข้อมูล และแกน y คือเวลาที่ใช้ในการประมวลผล (นาที) query จะสังเกตได้ว่าความเร็วในการ query ข้อมูลจะใช้เวลาเพิ่มขึ้นตามขนาดของทุก NoSQL ขนาด 1 GB, 3 GB Neo4j และ ArangoDB ใช้เวลาไล่เลี่ยกัน ส่วนขนาดข้อมูลตั้งแต่ 5 GB และ 7 GB ไม่สามารถรันออกได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการ query ข้อมูล และใช้เวลามากกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ ส่วน OrientDB ตั้งแต่ข้อมูล 3 GB ขึ้นไปไม่สามารถรันออกได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการ query ข้อมูล และใช้เวลามากกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ และ GraphDB ไม่สามารถนำข้อมูลเข้าระบบฐานข้อมูลได้ เพราะทรัพยากรทางผู้วิจัยที่ใช้ในการทำวิจัย ไม่เพียงพอต่อการ query ข้อมูล และใช้เวลามากกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ โดยแสดงผลการ query แบบที่ 3 ในแต่ละระบบฐานข้อมูล ดังรูปที่ 4.8



(A) แสดงผลการ query แบบที่ 3 ของ Neo4j



(B) แสดงผลการ query แบบที่ 3 ของ OrientDB



(C) แสดงผลการ query แบบที่ 3 ของ ArangoDB

รูปที่ 4.8 แสดงตัวอย่างผลที่ได้จากการ query แบบที่ 3 บน Neo4j (A), OrientDB (B) และ ArangoDB (C)

ตารางที่ 4.3 ตารางแสดงผลของ query แบบที่ 3

Q3	Neo4j		OrientDB		GraphDB		ArangoDB	
	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row	เวลา (นาที)	จำนวน row
1 GB	1.25	3693614	2.35	3693614	∞	-	0.18	3693614
3 GB	12.28	9726007	∞	-	-	-	6.3	9726007
5 GB	∞	-	∞	-	-	-	∞	-
7 GB	∞	-	∞	-	-	-	∞	-
หมายเหตุ : ∞ คือ การประมวลผลใช้เวลามากเกินกว่าที่ผู้วิจัยกำหนด (6 ชม.) - คือ ทรัพยากรไม่เพียงพอ จึงไม่สามารถทำการทดลอง query ได้								

จากตารางที่ 4.3 แสดงถึงเวลาและจำนวน row ของ query แบบที่ 3 โดยมีขนาดข้อมูล 1 GB, 3 GB, 5 GB และ 7 GB ตามลำดับ

4.1.3 ผลการเปรียบเทียบการใช้งานทรัพยากรของข้อมูลแบบ non-graph

จากวิธีการดำเนินการวิจัย แสดงถึงการใช้งานของทรัพยากรโดยเฉลี่ยของแต่ละระบบฐานข้อมูล โดยการใช้งานทรัพยากรของระบบฐานข้อมูล Neo4j ดังตารางที่ 4.4

ตารางที่ 4.4 การใช้งานทรัพยากรของระบบฐานข้อมูล Neo4j

Neo4j		1 GB			3 GB			5 GB			7 GB		
		CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)
Import	Service	25	7425	35	30	7300	25	33	8250	35	32	8600	45
	Firefox	0	3320	0	0	3100	0	0	2875	0	0	3175	0
Query 1	Service	11	10235	60	9	10500	85	13	10625	95	12	9875	105
	Firefox	0	2615	0	0	2800	0	0	2425	0	0	2915	0
Query 2	Service	15	9150	55	13	9235	75	17	8725	65	16	8560	90
	Firefox	0	3500	0	0	3125	0	0	3340	0	0	3270	0
Query 3	Service	13	8950	70	14	9725	75	-	-	-	-	-	-
	Firefox	0	3255	0	0	3355	0	-	-	-	-	-	-

จากตารางที่ 4.4 แสดงการใช้งานทรัพยากรของระบบฐานข้อมูล Neo4j ที่เรียกใช้งานในระหว่างการทำงาน โดยค่าเฉลี่ยในการใช้งานทรัพยากรจะเปรียบเทียบกับเวลาที่ใช้ในการทำงานทั้งหมด ซึ่งจะแบ่งออกเป็น 4 ขนาด คือ 1 GB, 3 GB ,5 GB และ 7 GB โดยจะใช้งานทรัพยากรจากส่วนต่าง ๆ ที่แตกต่างกัน โดยจะแบ่งออกตามขนาดของข้อมูล ดังนี้

1) การทำงานของขนาดข้อมูล 1 GB

- การนำข้อมูลเข้าใช้ระบบฐานข้อมูล Service (Neo4j) CPU 25% Memory 7425 MB Disk 35 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3320 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 11% Memory 10235 MB Disk 60 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2615 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 15% Memory 9150 MB Disk 55 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3500 MB

- query แบบที่ 3 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 13% Memory 8950 MB Disk 14 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3255 MB

2) การทำงานของขนาดข้อมูล 3 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 30% Memory 7300 MB Disk 25 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3100 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 9% Memory 10500 MB Disk 85 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2800 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 13% Memory 9235 MB Disk 75 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3125 MB

- query 3 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 14% Memory 9725 MB Disk 75 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3325 MB

3) การทำงานของขนาดข้อมูล 5 GB

- การนำข้อมูลเข้า ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 33% Memory 8250 MB Disk 35 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2875 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 13% Memory 10625 MB Disk 95 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2425 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 17% Memory 8725 MB Disk 65 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3340 MB

- query 3 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 17% Memory 9525 MB Disk 80 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2795 MB

4) การทำงานของขนาดข้อมูล 7 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 32% Memory 8600 MB Disk 45 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3175 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 12% Memory 9875 MB Disk 105 MB และระบบฐานข้อมูล Firefox ใช้ Memory 2915 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 16% Memory 8560 MB Disk 90 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3270 MB

- query 3 ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 15% Memory 9525 MB Disk 105 MB และระบบฐานข้อมูล Firefox ใช้ Memory 3235 MB

โดยการใช้งานทรัพยากรของระบบฐานข้อมูล OrientDB ดังตารางที่ 4.2

ตารางที่ 4.5 การใช้งานทรัพยากรของระบบฐานข้อมูล OrientDB

OrientDB		1 GB			3 GB			5 GB			7 GB		
		CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)
Import	Service Neo4j	22	4530	35	18	4580	35	19	4625	35	20	4560	35
	WCP	18	5720	30	19	5930	35	20	5735	35	18	5685	45
Query 1	WCP	15	6850	45	16	6950	50	17	6635	55	15	6775	60
	Firefox	1	535	0	1	540	0	1	520	0	1	545	0
Query 2	WCP	16	6400	45	19	6650	50	17	6595	45	18	6810	70
	Firefox	1	220	0	0	230	0	1	245	0	1	265	0
Query 3	WCP	15	8000	50	-	-	-	-	-	-	-	-	-
	Firefox	1	450	0	-	-	-	-	-	-	-	-	-

จากตารางที่ 4.5 แสดงการใช้งานทรัพยากรของระบบฐานข้อมูล OrientDB ที่เรียกใช้งานในระหว่างการทำงาน โดยค่าเฉลี่ยในการใช้งานทรัพยากรจะเปรียบเทียบกับเวลาที่ใช้ในการทำงาน

ทั้งหมด ซึ่งจะแบ่งออกเป็น 4 ขนาดของข้อมูลคือ 1 GB, 3 GB ,5 GB และ 7 GB โดยจะใช้งานทรัพยากรจากส่วนต่าง ๆ ที่แตกต่างกัน โดยจะแบ่งออกตามขนาดของข้อมูล ดังนี้

1) การทำงานของขนาดข้อมูล 1 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 22% Memory 4350 MB Disk 35 MB และระบบฐานข้อมูล WCP ใช้ CPU 18% Memory 6850 MB Disk 30 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล WCP CPU 15% Memory 6850 MB Disk 45 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 535 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล WCP CPU 16% Memory 6400 MB Disk 45 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 220 MB

- query แบบที่ 3 ใช้ระบบฐานข้อมูล WCP CPU 15% Memory 8000 MB Disk 50 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 450 MB

2) การทำงานของขนาดข้อมูล 3 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 18% Memory 4850 MB Disk 35 MB และระบบฐานข้อมูล WCP ใช้ CPU 19% Memory 5930 MB Disk 35 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล WCP CPU 16% Memory 6950 MB Disk 50 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 540 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล WCP CPU 19% Memory 6650 MB Disk 50 MB และระบบฐานข้อมูล Firefox ใช้ Memory 230 MB

3) การทำงานของขนาดข้อมูล 5 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 19% Memory 4625 MB Disk 35 MB และระบบฐานข้อมูล WCP ใช้ CPU 20% Memory 5735 MB Disk 35 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล WCP CPU 17% Memory 6635 MB Disk 55 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 520 MB

- query 2 ใช้ระบบฐานข้อมูล WCP CPU 17% Memory 6595 MB Disk 45 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 245 MB

4) การทำงานของขนาดข้อมูล 7 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 20% Memory 4560 MB Disk 35 MB และระบบฐานข้อมูล WCP ใช้ CPU 18% Memory 5685 MB Disk 45 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล WCP CPU 15% Memory 6775 MB Disk 60 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 545 MB

- query 2 ใช้ระบบฐานข้อมูล WCP CPU 18% Memory 6810 MB Disk 70 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 265 MB

โดยการใช้งานทรัพยากรของระบบฐานข้อมูล GraphDB ดังตารางที่ 4.6

ตารางที่ 4.6 การใช้งานทรัพยากรของระบบฐานข้อมูล GraphDB

GraphDB		1 GB			3 GB			5 GB			7 GB		
		CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)
Import	Service	50	9800	75	-	-	-	-	-	-	-	-	-
	Firefox	1	80	0	-	-	-	-	-	-	-	-	-
Query 1	Service	80	12875	70	-	-	-	-	-	-	-	-	-
	Firefox	2	220	0	-	-	-	-	-	-	-	-	-
Query 2	Service	83	13050	150	-	-	-	-	-	-	-	-	-
	Firefox	3	280	0	-	-	-	-	-	-	-	-	-
Query 3	Service	87	13250	120	-	-	-	-	-	-	-	-	-
	Firefox	3	295	0	-	-	-	-	-	-	-	-	-

จากตารางที่ 4.6 แสดงการใช้งานทรัพยากรของระบบฐานข้อมูล GraphDB ที่เรียกใช้งานในระหว่างการทำงาน โดยค่าเฉลี่ยในการใช้งานทรัพยากรจะเปรียบเทียบกับเวลาที่ใช้ในการทำงานทั้งหมด ขนาดของข้อมูลคือ 1 GB โดยจะใช้งานทรัพยากรจากส่วนต่าง ๆ ที่แตกต่างกันดังนี้

1) การทำงานของขนาดข้อมูล 1 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (GraphDB) CPU 50% Memory 9800 MB Disk 75 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 80 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (GraphDB) CPU 80% Memory 12875 MB Disk 70 MB ระบบฐานข้อมูล Firefox ใช้ CPU 2% Memory 220 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (GraphDB) CPU 83% Memory 13050 MB Disk 150 MB และระบบฐานข้อมูล Firefox ใช้ CPU 3% Memory 280 MB

โดยการใช้งานทรัพยากรของระบบฐานข้อมูล ArangoDB ดังตารางที่ 4.7

ตารางที่ 4.7 การใช้งานทรัพยากรของระบบฐานข้อมูล ArangoDB

ArangoDB		1 GB			3 GB			5 GB			7 GB		
		CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)
Import	Service	12	7775	125	13	8750	150	14	8200	100	14	8500	150
	Firefox	0	400	25	0	350	25	0	375	35	0	350	45
Query 1	Service	13	9000	100	13	8895	100	13	9000	125	13	9575	175
	Firefox	0	375	45	0	365	35	0	375	50	0	350	25
Query 2	Service	13	9150	110	14	8350	145	14	9100	145	12	8250	140
	Firefox	0	400	35	0	550	55	0	350	45	0	500	40
Query 3	Service	13	8450	150	13	91175	100	12	9075	125	14	8250	135
	Firefox	0	400	20	0	375	50	0	400	35	0	450	30

จากตารางที่ 4.7 แสดงการใช้งานทรัพยากรของระบบฐานข้อมูล ArangoDB ที่เรียกใช้งานในระหว่างการทำงาน โดยค่าเฉลี่ยในการใช้งานทรัพยากรจะเปรียบเทียบกับเวลาที่ใช้ในการทำงานทั้งหมด ซึ่งจะแบ่งออกเป็น 4 ขนาดของข้อมูลคือ 1 GB, 3 GB, 5 GB และ 7 GB โดยจะใช้งานทรัพยากรจากส่วนต่าง ๆ ที่แตกต่างกัน โดยจะแบ่งออกตามขนาดของข้อมูล ดังนี้

1) การทำงานของขนาดข้อมูล 1 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 12% Memory 7775 MB Disk 125 MB และระบบฐานข้อมูล Firefox ใช้ Memory 400 MB Disk 25 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 9000 MB Disk 100 MB และระบบฐานข้อมูล Firefox ใช้ Memory 375 MB Disk 45 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 9150 MB Disk 110 MB และระบบฐานข้อมูล Firefox ใช้ Memory 400 MB Disk 35 MB

- query แบบที่ 3 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 8450 MB Disk 150 MB และระบบฐานข้อมูล Firefox ใช้ Memory 400 MB Disk 20 MB

2) การทำงานของขนาดข้อมูล 3 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 8750 MB Disk 150 MB และระบบฐานข้อมูล Firefox ใช้ Memory 350 MB Disk 25 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 8895 MB Disk 100 MB และระบบฐานข้อมูล Firefox ใช้ Memory 365 MB Disk 35 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 14% Memory 8350 MB Disk 145 MB และระบบฐานข้อมูล Firefox ใช้ Memory 550 MB Disk 55 MB

- query แบบที่ 3 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 9175 MB Disk 100 MB และระบบฐานข้อมูล Firefox ใช้ Memory 375 MB Disk 50 MB

3) การทำงานของขนาดข้อมูล 5 GB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 14% Memory 8200 MB Disk 100 MB และระบบฐานข้อมูล Firefox ใช้ Memory 375 MB Disk 35 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 9000 MB Disk 125 MB และระบบฐานข้อมูล Firefox ใช้ Memory 375 MB Disk 50 MB

- query แบบที่ 2 ใช้ระบบฐานข้อมูล Service(ArangoDB) CPU 14% Memory 9100 MB Disk 145 MB และระบบฐานข้อมูล Firefox ใช้ Memory 350 MB Disk 45 MB

- query แบบที่ 3 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 12% Memory 9075 MB Disk 125 MB และระบบฐานข้อมูล Firefox ใช้ Memory 400 MB Disk 35 MB

4) การทำงานของขนาดข้อมูล 7 GB

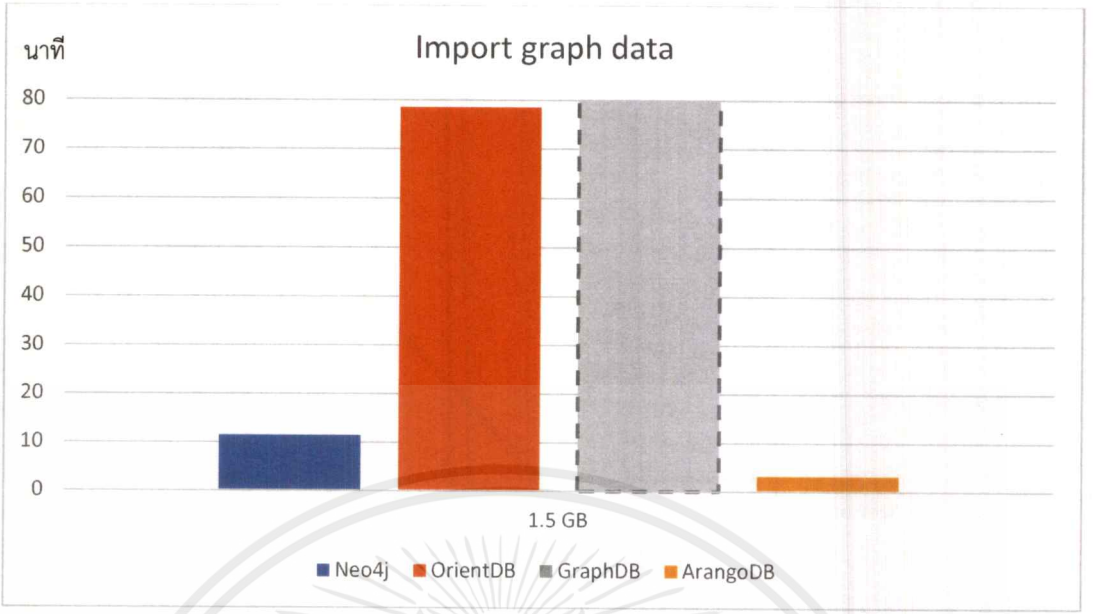
- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 14% Memory 8500 MB Disk 150 MB และระบบฐานข้อมูล Firefox ใช้ Memory 350 MB Disk 45 MB

- query แบบที่ 1 ใช้ระบบฐานข้อมูล Service (ArangoDB) CPU 13% Memory 9575 MB Disk 175 MB และระบบฐานข้อมูล Firefox ใช้ Memory 350 MB Disk 25 MB

4.2 ผลการวิจัยของข้อมูล graph

4.2.1 การเปรียบเทียบการนำข้อมูลเข้าในระบบฐานข้อมูล

โดยการทดลองจะมีการนำข้อมูลเข้าระบบฐานข้อมูลที่ใช้ทั้งหมด 4 ระบบฐานข้อมูล ดังรูปที่ 4.9

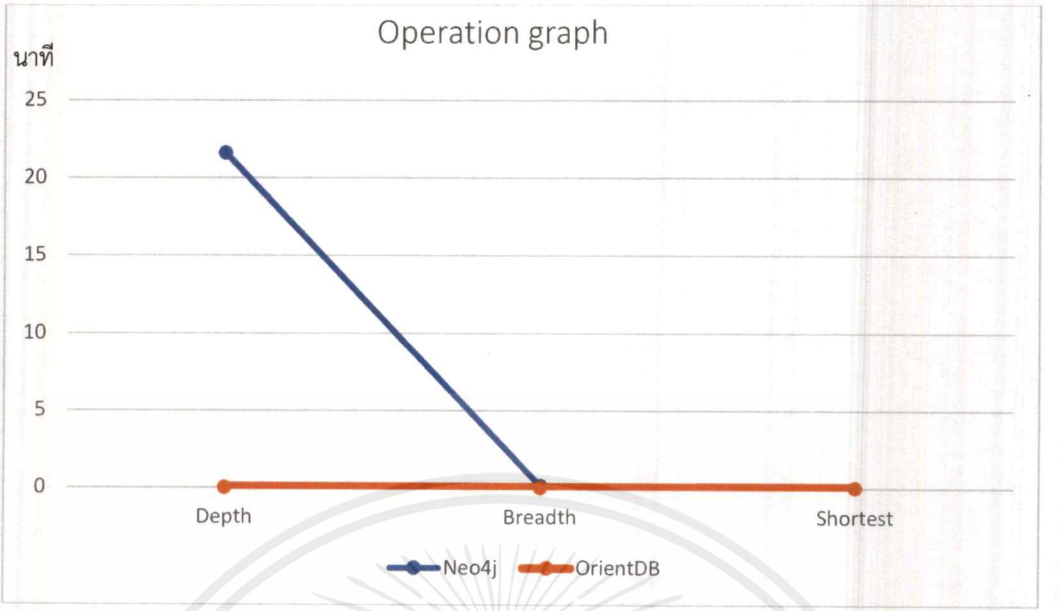


รูปที่ 4.9 กราฟแสดงเวลาในการนำข้อมูลเข้าแต่ละระบบฐานข้อมูล

จากรูปที่ 4.9 แสดงการเปลี่ยนแปลงขนาดของข้อมูลที่เพิ่มขึ้นเมื่อเทียบกับขนาดของเวลาที่ใช้ โดยระบบฐานข้อมูล Neo4j ใช้เวลาค่อนข้างคงที่เมื่อเทียบกับขนาดข้อมูลที่ต่างกัน สำหรับระบบฐานข้อมูล OrientDB ขนาดของเวลาที่ใช้ขึ้นอยู่กับขนาดของข้อมูล สำหรับระบบฐานข้อมูล GraphDB ตั้งแต่ข้อมูลขนาด 3 GB ขึ้นไปจะใช้เวลาในการทำงานนานเกินกว่า 6 ชั่วโมง ทางผู้วิจัยจึงไม่สามารถรอผลการทดลองได้ และระบบฐานข้อมูล ArangoDB ขนาดข้อมูล 1 GB และ 3 GB ใช้เวลาน้อยเมื่อเทียบกับขนาดข้อมูล ส่วนขนาด 5 GB และ 7 GB ใช้เวลานานขึ้นเมื่อเทียบกับข้อมูลที่มีขนาดที่เพิ่มขึ้น

4.2.2 ผลการทดลองของการ query สำหรับข้อมูล graph

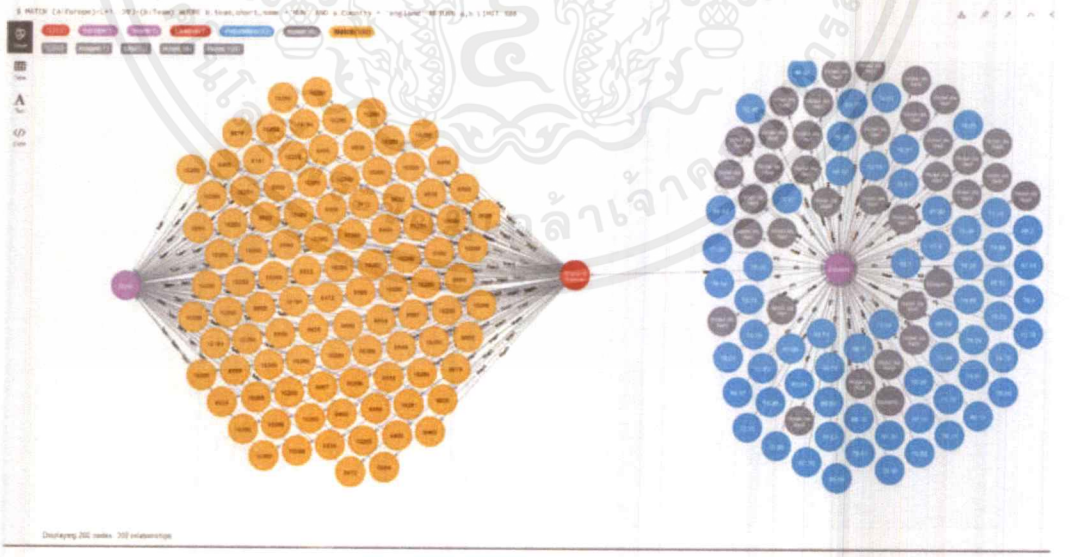
โดยการทดสอบจะทำการทดลองทั้งหมด 3 แบบ คือ การ query ในการดำเนินการทางกราฟที่ใช้งาน ทั้งหมด 3 แบบ คือ 1) การค้นหาแบบแนวลึก (Depth-First Search : DFS) 2) การค้นหาแบบแนวกว้าง (Breadth-First Search : BFS) 3) เส้นทางที่สั้นที่สุด (Shortest Path) ของแต่ละระบบฐานข้อมูล ดังรูปที่ 4.10



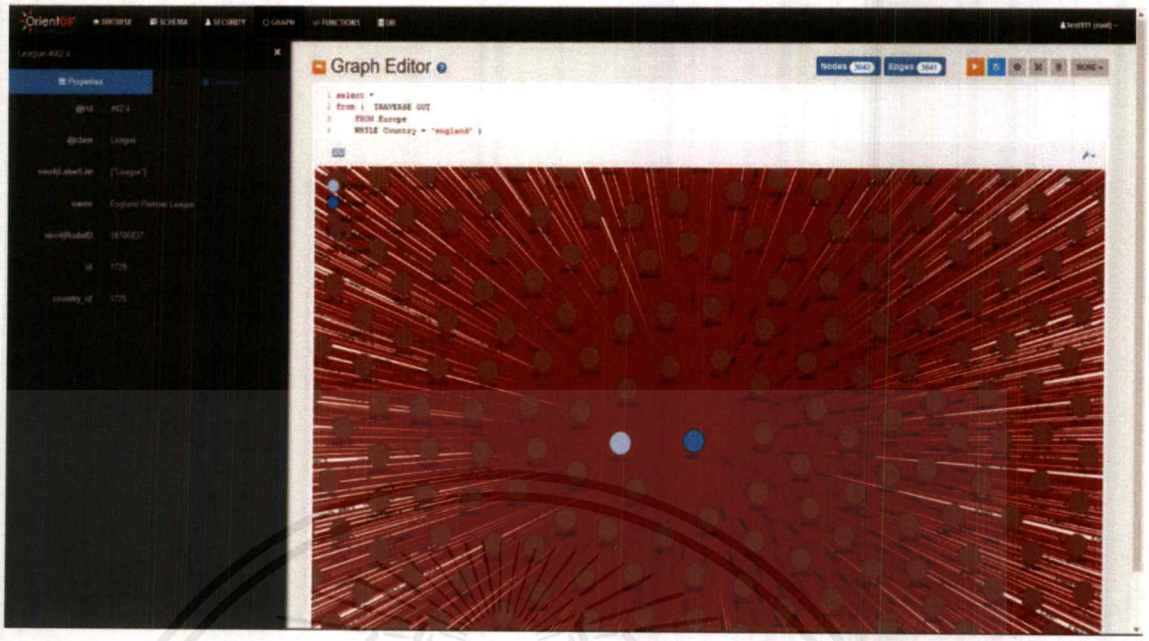
รูปที่ 4.10 กราฟแสดงผลการ query แบบ graph

จากรูปที่ 4.10 แสดงเวลาการ query แบบต่าง ๆ ของแต่ละระบบฐานข้อมูล ซึ่งผลของการทดลอง ดังต่อไปนี้

1) การค้นหาแบบแนวลึก (Depth-First Search : DFS)



(A) แสดงผลการ query แบบแนวลึก (Depth-First Search : DFS) ของ Neo4j

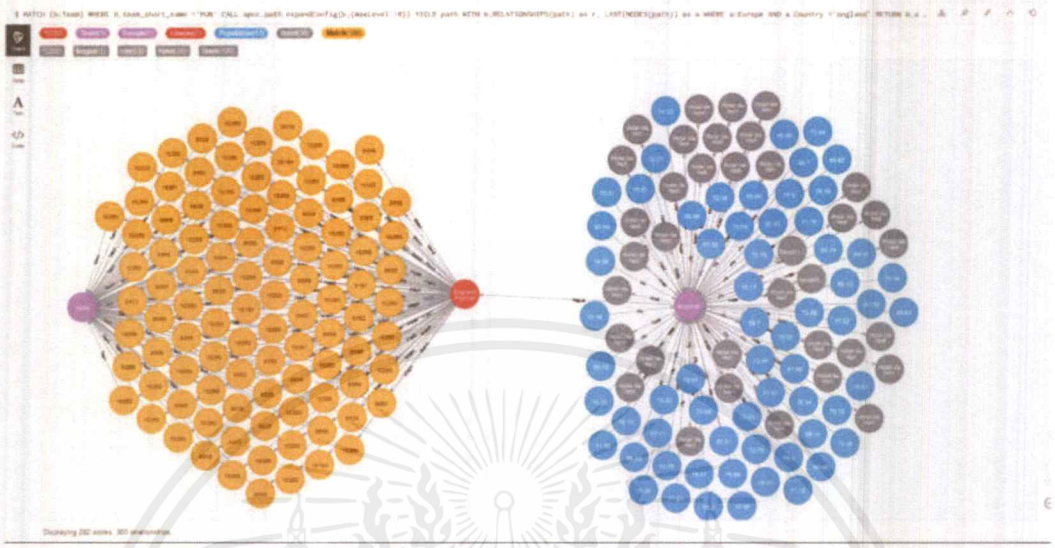


(B) แสดงผลการ query แบบแนวลึก (Depth-First Search : DFS) ของ OrientDB

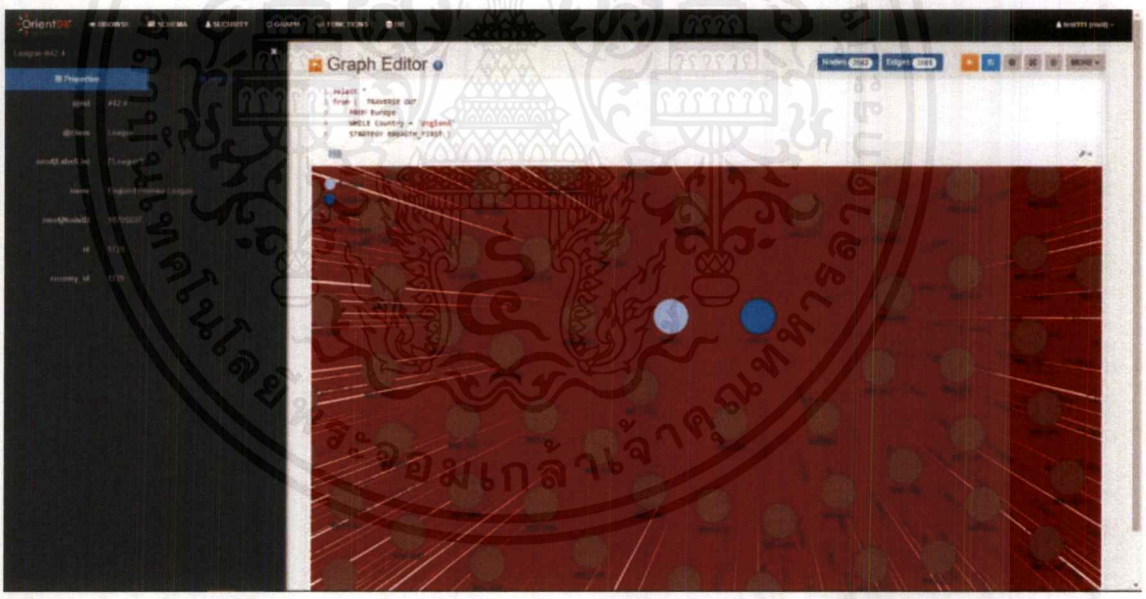
รูปที่ 4.11 แสดงผลที่ได้จากการ query แบบแนวลึก (Depth-First Search :DFS) ของ Neo4j (A) และ OrientDB (B)

จากรูปที่ 4.11 แสดงผลที่ได้จากการ query แบบแนวลึก (Depth-First Search :DFS) ของ Neo4j ใช้เวลาในการรัน 21.58 นาที และผลที่ได้จากการ query แบบแนวลึก (Depth-First Search :DFS) ของ OrientDB ใช้เวลาในการรัน 0.001 นาที ส่วน GraphDB เป็นการดำเนินการแบบ RDF Search ทางผู้วิจัยจึงตัดสินว่าไม่สามารถนำมาเปรียบเทียบกันได้และไม่สามารถนำข้อมูลเข้าได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการทดลอง ส่วน ArangoDB ไม่สามารถเขียน query เพื่อให้วนสร้างเส้นเชื่อมแบบระบบฐานข้อมูล Neo4j ได้ ซึ่งสามารถสร้างเส้นเชื่อมได้แบบจับคู่ทีละโหนดเท่านั้น

2) การค้นหาแบบแนวกว้าง (Breadth-First Search : BFS)



(A) แสดงผลการ query แบบแนวกว้าง (Breadth-First Search : BFS) ของ Neo4j

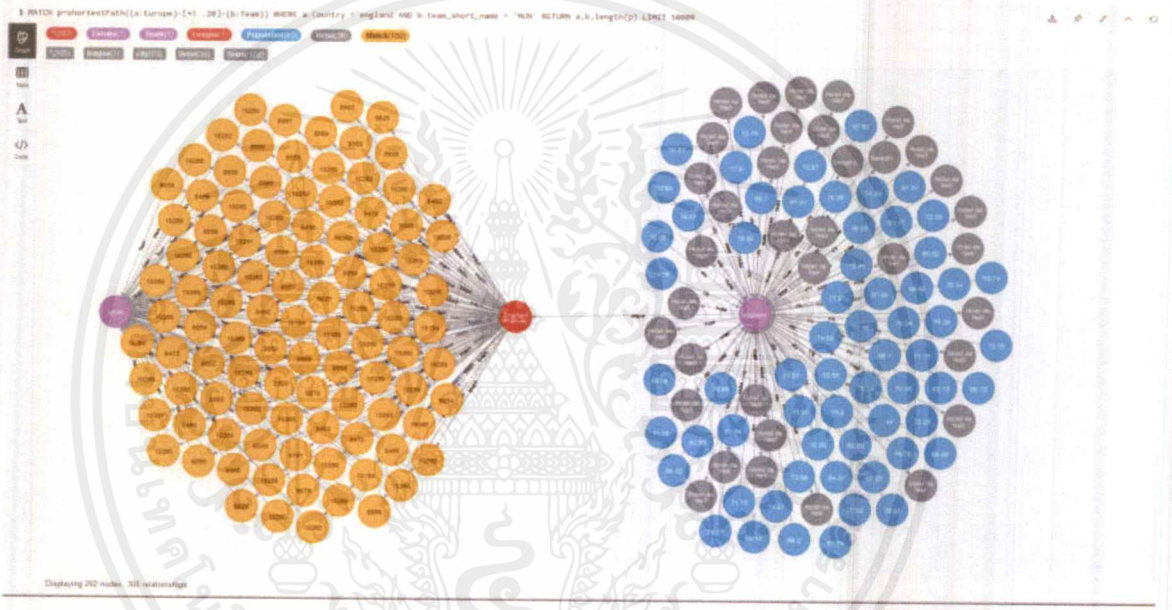


(B) แสดงผลการ query แบบแนวกว้าง (Breadth-First Search : BFS) ของ OrientDB

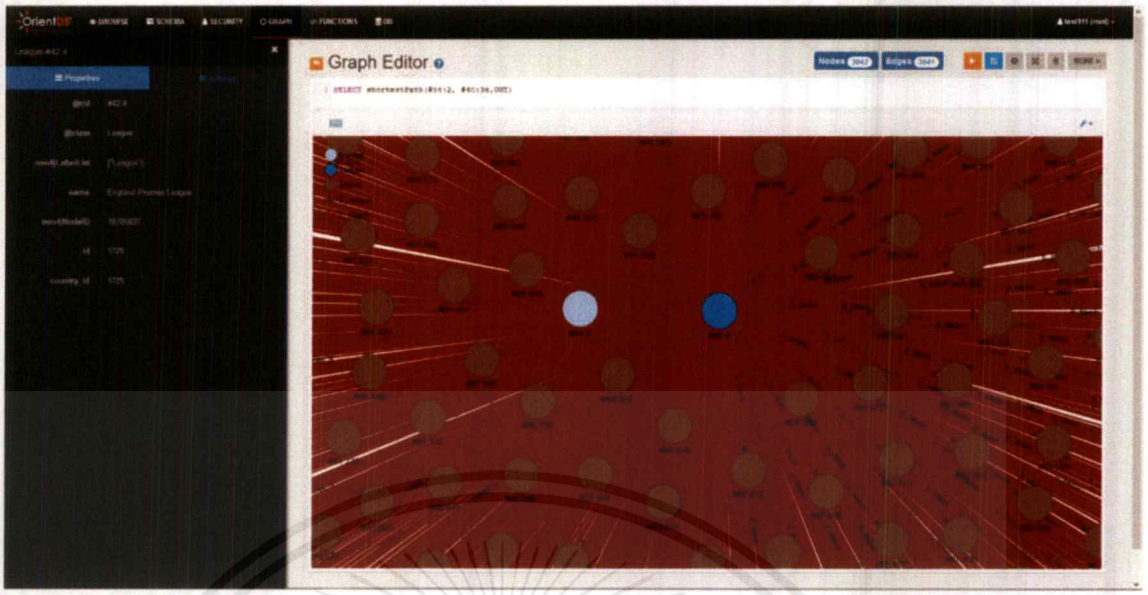
รูปที่ 4.12 แสดงผลที่ได้จากการ query แบบแนวกว้าง (Breadth-First Search : BFS) ของ Neo4j (A) และ OrientDB (B)

จากรูปที่ 4.12 แสดงผลที่ได้จากการ query แบบแนวกว้าง (Breadth-First Search : BFS) ของ Neo4j ใช้เวลาในการรัน 0.15 นาที และผลที่ได้จากการ query แบบแนวกว้าง (Breadth-First Search : BFS) ของ OrientDB ใช้เวลาในการรัน 0.001 นาที ส่วน GraphDB เป็นการดำเนินการแบบ RDF Search ทางผู้วิจัยจึงตัดสินว่าไม่สามารถนำมาเปรียบเทียบกันได้และไม่สามารถนำข้อมูลเข้าได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการทดลอง ส่วน ArangoDB ไม่สามารถเขียน query เพื่อให้วนสร้างเส้นเชื่อมแบบระบบฐานข้อมูล Neo4j ได้ ซึ่งสามารถสร้างเส้นเชื่อมได้แบบจับคู่ทีละโหนดเท่านั้น

3) เส้นทางที่สั้นที่สุด (Shortest Path)



(A) แสดงผลการ query แบบเส้นทางที่สั้นที่สุด (Shortest Path) ของ Neo4j



(B) แสดงผลการ query แบบเส้นทางที่สั้นที่สุด (Shortest Path) ของ OrientDB

รูปที่ 4.13 แสดงผลที่ได้จากการ query แบบเส้นทางที่สั้นที่สุด (Shortest Path) ของ Neo4j (A) และ OrientDB (B)

จากรูปที่ 4.13 แสดงผลที่ได้จากการ query แบบเส้นทางที่สั้นที่สุด (Shortest Path)) ของ Neo4j ใช้เวลาในการรัน 0.02 นาที และผลที่ได้จากการ query แบบเส้นทางที่สั้นที่สุด (Shortest Path) ของ OrientDB ใช้เวลาในการรัน 0.02 นาที ส่วน GraphDB เป็นการดำเนินการแบบ RDF Search ทางผู้วิจัยจึงตัดสินว่าไม่สามารถนำมาเปรียบเทียบกันได้และไม่สามารถนำข้อมูลเข้าได้ เนื่องจากทรัพยากรของทางผู้วิจัยไม่เพียงพอต่อการทดลอง ส่วน ArangoDB ไม่สามารถเขียน query เพื่อให้วนสร้างเส้นเชื่อมแบบระบบฐานข้อมูล Neo4j ได้ ซึ่งสามารถสร้างเส้นเชื่อมได้แบบจับคู่ทีละโหนดเท่านั้น

4.2.3 ผลการเปรียบเทียบการใช้งานทรัพยากรของข้อมูลแบบ graph

จากการทดลองวิจัย ได้แสดงถึงการใช้งานของทรัพยากรโดยเฉลี่ยของแต่ละระบบฐานข้อมูล ดังตารางที่ 4.8

ตารางที่ 4.8 การใช้ทรัพยากรของแต่ละระบบฐานข้อมูล

1.5GB		Neo4j				OrientDB			GraphDB				ArangoDB		
		CPU (%)	Memory (MB)	Disk (MB)		CPU (%)	Memory (MB)	Disk (MB)	CPU (%)	Memory (MB)	Disk (MB)		CPU (%)	Memory (MB)	Disk (MB)
Import	Service	37	12930	20	WCP	13	5615	35	-	-	-	Service	-	-	-
	Firefox	0	820	0	Service (Neo4j)	12	1120	25	-	-	-	Firefox	-	-	-
Query 1 (Depth)	Service	12	1320	0	WCP	4	820	0	-	-	-	Service	-	-	-
	Firefox	1	650	0	Firefox	7	760	0	-	-	-	Firefox	-	-	-
Query 2 (Breadth)	Service	12	2700	0	WCP	5	880	0	-	-	-	Service	-	-	-
	Firefox	2	800	0	Firefox	6	825	0	-	-	-	Firefox	-	-	-
Query 3 (Shortest)	Service	14	2730	0	WCP	4	890	0	-	-	-	Service	-	-	-
	Firefox	1	820	0	Firefox	5	850	0	-	-	-	Firefox	-	-	-

จากตารางที่ 4.8 แสดงการใช้งานทรัพยากรของแต่ละระบบฐานข้อมูลที่เรียกใช้งานระบบฐานข้อมูลอื่น ๆ ในระหว่างการทำงาน โดยค่าเฉลี่ยในการใช้งานทรัพยากรจะเปรียบเทียบกับเวลาที่ใช้ในการทำงานทั้งหมด ซึ่งระบบฐานข้อมูลทั้งหมดจะใช้งานทรัพยากรจากส่วนต่าง ๆ ที่แตกต่างกัน ดังนี้

1) ระบบฐานข้อมูล Neo4j

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 37% Memory 12930 MB Disk 20 MB และระบบฐานข้อมูล Firefox ใช้ Memory 820 MB

- query แบบที่ 1(Depth) ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 12% Memory 1320 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 650 MB

- query แบบที่ 2 (Breadth) ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 12% Memory 2700 MB และระบบฐานข้อมูล Firefox ใช้ CPU 2% Memory 800 MB

- query แบบที่ 3 (Shortest) ใช้ระบบฐานข้อมูล Service (Neo4j) CPU 14% Memory 2730 MB และระบบฐานข้อมูล Firefox ใช้ CPU 1% Memory 820 MB

2) ระบบฐานข้อมูล OrientDB

- การนำข้อมูลเข้าระบบฐานข้อมูล ใช้ระบบฐานข้อมูล WCP 13% Memory 5615 MB Disk 35% และระบบฐานข้อมูล Service (Neo4j) CPU 12% Memory 1120 MB Disk 25 MB

- query แบบที่ 1 (Depth) ใช้ระบบฐานข้อมูล WCP CPU 4% Memory 820 MB และระบบฐานข้อมูล Firefox ใช้ CPU 7% Memory 760 MB

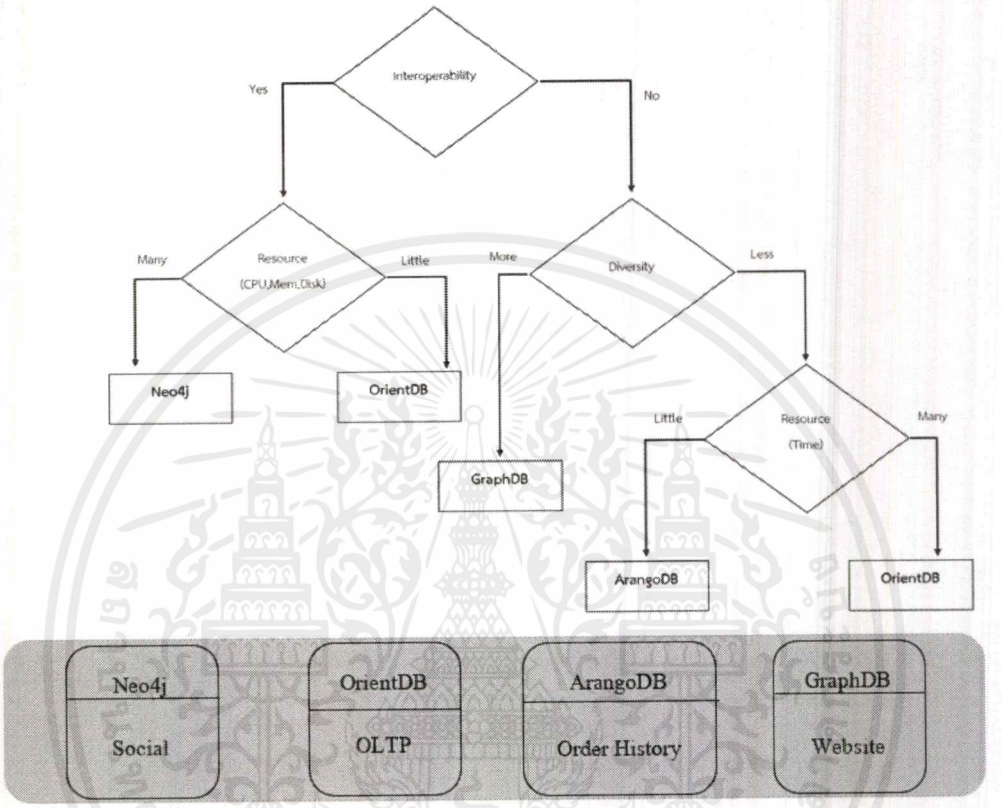
- query แบบที่ 2 (Breadth) ใช้ระบบฐานข้อมูล WCP CPU 5% Memory 880 MB และระบบฐานข้อมูล Firefox ใช้ CPU 6% Memory 825 MB

- query แบบที่ 3 (Shortest) ใช้ระบบฐานข้อมูล WCP CPU 4% Memory 890 MB และระบบฐานข้อมูล Firefox ใช้ CPU 5% Memory 850 MB

3) ระบบฐานข้อมูล GraphDB เป็นการดำเนินการแบบ RDF Search ทางผู้วิจัยจึงตัดสินว่าไม่สามารถนำมาเปรียบเทียบกันได้

4) ระบบฐานข้อมูล ArangoDB ไม่สามารถเขียน query เพื่อทำการสร้างเส้นเชื่อมวน แบบระบบฐานข้อมูล Neo4j ได้ ซึ่งสามารถสร้างเส้นเชื่อมได้แบบทีละคู่เท่านั้น

จากผลการทดลองที่ได้ทำให้ผู้วิจัยได้สรุปทางเลือกระบบฐานข้อมูล NoSQL โดยนำเสนอในรูปแบบต้นไม้ตัดสินใจดังรูปที่ 4.14



รูปที่ 4.14 ต้นไม้ตัดสินสำหรับการแนะนำระบบฐานข้อมูล

จากรูปที่ 4.14 เป็นทางเลือกสำหรับการนำไปใช้งาน ซึ่งสำหรับการเลือกใช้งานจะเลือกจากเงื่อนไขแรก คือ Interoperability จะประเมินผลจากการทำงานรวมกันของฐานข้อมูลกับซอฟแวร์อื่นที่ทำงานร่วมกัน รวมไปถึงการเข้าถึงข้อมูลสนับสนุนการทำงานหรือตัวอย่างการใช้งาน โดยเงื่อนไขในลำดับต่อมา คือ Resource (CPU, Mem, Disk) ได้ประเมินผลจากการทดลองในระหว่างการประมวลผลได้ใช้งานทรัพยากร CPU, Memory และ Disk โดยเงื่อนไขในลำดับต่อมา คือ Diversity ได้ประเมินจากคุณสมบัติต่าง ๆ ของ data model ที่ฐานข้อมูลรองรับได้ และโดยเงื่อนไขในลำดับสุดท้าย คือ ระยะเวลาที่ใช้ในการประมวลผลที่แตกต่างกันในรูปแบบ non-graph

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการทดลอง

ปัญหาพิเศษนี้มีจุดมุ่งหมายเพื่อศึกษาการใช้งานและวัดประสิทธิภาพระบบฐานข้อมูล NoSQL ประเภท Graph-based เพื่อใช้เป็นแนวทางสำหรับผู้ใช้งานในการนำไปประยุกต์ใช้กับข้อมูลขนาดใหญ่ประเภทกราฟ

ในปัญหาพิเศษนี้ เราได้ทำการทดลองเป็นแบบสภาวะเชิงเดี่ยว (stand-alone environment) ดังนั้นในการนำไปใช้งานจริงกับการเชื่อมต่อแบบเซิร์ฟเวอร์ อาจมีความคลาดเคลื่อน นอกจากนี้คุณสมบัติของอุปกรณ์/ฮาร์ดแวร์ (specification) ที่นำมาใช้ส่งผลกระทบโดยตรงต่อการวัดประสิทธิภาพของระบบ เราได้ทำการทดสอบการทำงานของระบบฐานข้อมูล NoSQL ประเภท Graph-based กับข้อมูลประเภท non-graph และ graph ตามขนาดที่แตกต่างกัน (1 - 7 GB) จากผลการทดลองพบว่าการใช้ทรัพยากรภาพรวมของข้อมูลประเภท non-graph ใช้ทรัพยากรมากกว่าข้อมูลประเภท graph ของแต่ละระบบฐานข้อมูลโดยมีการใช้ทรัพยากรดังนี้ (จากมากไปน้อย) GraphDB > Neo4j > ArangoDB > OrientDB และสามารถสรุปได้ว่า

1. สำหรับข้อมูลประเภท non-graph: ในการเปรียบเทียบระยะเวลาที่ใช้ในการนำข้อมูลขนาด 1-3 GB ระบบฐานข้อมูล ArangoDB ใช้เวลาน้อยที่สุด และข้อมูลขนาด 4-7 GB ระบบฐานข้อมูล Neo4j ใช้เวลาน้อยที่สุด ซึ่งหลังจากการนำข้อมูลเข้าระบบ ขนาดของข้อมูลได้เติบโตขึ้น พบว่าระบบฐานข้อมูล ArangoDB มีการเติบโตของข้อมูลที่เพิ่มขึ้นน้อยที่สุด ในขณะที่ระบบฐานข้อมูล GraphDB มีอัตราการเติบโตของข้อมูลเพิ่มขึ้นมากที่สุด ในส่วนของการ query ข้อมูลของแต่ละ query พบว่า query ที่ 1 กับ 2 ระบบฐานข้อมูล Neo4j ใช้เวลาน้อยที่สุด และ query ที่ 3 ระบบฐานข้อมูล ArangoDB ใช้เวลาน้อยที่สุด อย่างไรก็ตาม โดยภาพรวม ระบบฐานข้อมูล Neo4j ใช้เวลาในการทำงานแปรผันตามกับขนาดของข้อมูลใกล้เคียงกันมากที่สุด

2. สำหรับข้อมูลประเภท graph: จากผลการทดลองกับข้อมูลขนาดเท่ากันคือ 1.5 GB โดยในการเปรียบเทียบระยะเวลาในการนำข้อมูล ระบบฐานข้อมูล ArangoDB ใช้เวลาในการทำงานน้อยที่สุด แต่ระยะเวลาในการสร้างเส้นเชื่อมระหว่างโหนด ระบบฐานข้อมูล Neo4j ทำงานได้เร็วที่สุด การนำเข้าข้อมูลของระบบฐานข้อมูล OrientDB สามารถใช้ Feature ของระบบฐานข้อมูล Neo4j ได้ เราพบว่าระบบฐานข้อมูล OrientDB สามารถนำเข้าข้อมูลและสร้างเส้นเชื่อมได้เร็วกว่า ระบบฐานข้อมูล Neo4j นอกจากนี้ เรายังสังเกตเห็นว่าในการสร้างเส้นเชื่อมของระบบฐานข้อมูล ArangoDB นั่นไม่สามารถเขียน query เพื่อให้วนสร้างเส้นเชื่อมแบบอัตโนมัติดังเช่นระบบฐานข้อมูล

Neo4j การสร้างเส้นเชื่อมโหนดจำเป็นต้องทำแบบเชื่อมต่อทีละคู่ ในเรื่องการประมวลผลหลังจากนำเข้าข้อมูล ระบบฐานข้อมูล OrientDB ใช้เวลาในการทำงานน้อยที่สุด ในขณะที่ระบบฐานข้อมูล Neo4j ใช้เวลามากที่สุด ทั้งนี้อาจเป็นไปได้ว่าระบบฐานข้อมูล มีการแสดงผลเป็นกราฟจึงทำให้ใช้เวลาในการทำงานมากกว่า สำหรับระบบฐานข้อมูล GraphDB ซึ่งมีรูปแบบการเก็บข้อมูลเป็นแบบ RDF Search ทางผู้วิจัยจึงไม่สามารถวิเคราะห์และตัดสิน ประสิทธิภาพในเรื่องดังกล่าวได้อย่างชัดเจน นอกจากนี้เรายังพบว่าระบบฐานข้อมูล GraphDB นั้นใช้งานยาก ขาดคู่มือและเอกสารที่เพียงพอในการทดลองของเรา การนำเข้าข้อมูลขนาด 1.5 GB ไม่สามารถทำได้

5.2 ปัญหาที่เกิดขึ้น

จากการทดลองทางผู้วิจัยได้เห็นถึงปัญหาที่เกิดขึ้น ดังนี้

1) ในการติดตั้งระบบฐานข้อมูล Neo4j จำเป็นต้องปรับการตั้งค่า เพื่อรองรับการนำเข้าของข้อมูล โดยเพิ่มทรัพยากร Memory ในไฟล์ config ให้มีค่าสูงที่สุด

2) การนำเข้าข้อมูลใน OrientDB แม้ว่าจะมีการปรับการตั้งค่า เพื่อรองรับการนำเข้าของข้อมูลก่อนแล้ว โดยเพิ่มทรัพยากร Memory ในไฟล์ Java ก็ยังไม่สามารถนำข้อมูลลง OrientDB ได้ จึงจำเป็นต้องนำข้อมูลเข้าผ่านระบบฐานข้อมูล Neo4j แทน

3) พบปัญหาการนำข้อมูลลงในระบบฐานข้อมูล GraphDB ที่มีขนาดมากกว่า 1 GB ขึ้นไป จึงทำให้ผู้วิจัยไม่สามารถทำการทดลองกับขนาดข้อมูลอื่น ๆ ได้ และระบบฐานข้อมูล GraphDB ใช้ทรัพยากรในการประมวลผลมากที่สุด

4) ระบบฐานข้อมูล ArangoDB มีปัญหาในเรื่อง การสร้างความสัมพันธ์แบบซับซ้อน จึงทำให้ไม่สามารถท่องกราฟ และการทดลองกราฟทั้ง 3 แบบได้

5.3 ข้อเสนอแนะ

เนื่องจากผลการวิจัย สรุปได้ว่าระบบฐานข้อมูลทั้ง 4 ประเภท ใช้ทรัพยากรของ RAM, Disk และ CPU ทางผู้วิจัยจึงเล็งเห็นว่า การใช้งานระบบฐานข้อมูลทั้ง 4 ประเภทนี้ ควรใช้งานบนระบบ Server ที่ทรัพยากรสามารถรองรับการประมวลผลของข้อมูลได้ และสิ่งสำคัญต่อประสิทธิภาพของระบบฐานข้อมูล คือ RAM, Disk และ CPU ซึ่งการรองรับฟังก์ชันของระบบฐานข้อมูลของ Neo4j และ OrientDB สามารถรองรับการทำงานได้ใกล้เคียงกับ RDBMS โดยในการทำงานได้ขึ้นอยู่กับความซับซ้อนของข้อมูล และรูปแบบการดำเนินการของการค้นหาข้อมูล

เอกสารอ้างอิง

[1] Alejandro Corbellini, C. M. (2017). **Persisting big-data: The NoSQL landscape.** Information Systems, 1-23.

[2] Al-Naymat, S. S. (2009). **Relational Processing of RDF Queries: A Survey**. SIGMOD Record, 23-28.

[3] **Best Graph Databases Software.** (n. d.) . Retrieved from g2crowd: https://www.g2crowd.com/categories/graph-databases

[4] Bozan, O. B. (2017, Feburary 26). **Retrieved from Distributed Computing in Microservices: CAP Theorem:** https: / / blog. kloia. com/ distributed- computing- in- microservices- cap-theorem-253c16017a99

[5] Bryce Merkl Sasaki, A. G. (2015, September 4). **Graph Databases for Beginners: ACID vs. BASE Explained** . Retrieved from https: / / neo4j. com/ blog/ acid- vs- base- consistency- models-explained/

[6] Bryce Merkl Sasaki, A. G. (2015, August 7). **Graph Databases for Beginners: The Basics of Data Modeling.** Retrieved from https://neo4j.com/blog/data-modeling-basics/

[7] Bryce Merkl Sasaki, A. G. (2015, August 28). **Graph Databases for Beginners: Why We Need NoSQL Databases.** Retrieved from https://neo4j.com/blog/data-modeling-basics/

[8] Chao, J. (2016, August 1). **Graph Databases for Beginners: Graph Search Algorithm Basics.** Retrieved from https://neo4j.com/blog/graph-search-algorithm-basics/

[9] **DB-Engines Ranking of Graph DBMS.** (n.d.). Retrieved from DB-Engines: https://db-engines.com/en/ranking/graph+dbms

[10] Felix Gessert, W. W. (2017). **NoSQL database systems: a survey and decision guidance.** Comput Sci Res Dev , 353-365.

[11] Hunger, M. (2014, February 17). **Combining depth- and breadth-first traversals in a single cypher query .** Retrieved from https://stackoverflow.com/questions/21695484/combining-depth-and-breadth-first-traversals-in-a-single-cypher-query

[12] Ian Robinson, J. W. (2015). **Graph databases.USA**: O'Reilly Media.

[13] Introduction. (n.d.). Retrieved from ArangoDB: https://docs.arangodb.com/3.1/AQL/

[14] **Introduction to Distributed Database Management Systems** . (2014, May 17). Retrieved from http://sungsoo.github.io/2014/05/17/distributed-database-management-system-introduction.html

[15] IT, s. (n.d.). **System Properties Comparison ArangoDB vs. MySQL**. Retrieved from db-engines: https://db-engines.com/en/system/MySQL;ArangoDB

[16] IT, s. (n.d.). **System Properties Comparison GraphDB vs. MySQL**. Retrieved from db-engines: https://db-engines.com/en/system/MySQL;graphdb

[17] IT, s. (n.d.). **System Properties Comparison MySQL vs. Neo4j**. Retrieved from db-engines: System Properties Comparison MySQL vs. Neo4j

[18] IT, s. (n.d.). **System Properties Comparison MySQL vs. OrientDB**. Retrieved from db-engines: https://db-engines.com/en/system/MySQL;orientdb

[19] Longgang Xiang (2016, 14 July). **A MongoDB-Based Management of Planar Spatial Data with a Flattened R-Tree.** ISPRS Int. J. Geo-Inf. 2016, 5, 119.

[20] luigidellaquila. (2017). **OrientDB is the most versatile DBMS supporting Graph**. Retrieved from https://github.com/orientechnologies/orientdb

[21] Mainetti, I. (2016) . **Retrieve all paths from a node.** Retrieved from https://stackoverflow.com/questions/37207492/retrieve-all-paths-from-a-node?rq=1

[22] OrientDB Manual. (2017). London: Unit 702, Salisbury Hourse, London Wall.

[23] Puisungnoen, S. (2014). **ฐานข้อมูลรูปแบบต่างๆ สำหรับจัดการ** Big Data Retrieved from http://www.somkiat.cc/big-data-database-model/

[24] Retrieved from https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_WP_Retail_Innovation_EN_US.pdf?_ga=2.263453772.1979217833.1529937703-581337064.1517290624

[25] Retrieved from https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_WP_Recommendations_EN_BUS.pdf?_ga=2.263861196.1979217833.1529937703-581337064.1517290624

[26] Retrieved from http://graphdb.ontotext.com/free/introduction-to-semantic-web.html:

[27] Retrieved from http://graphdb.ontotext.com/documentation/free/devhub/map.html:

[28] Retrieved from ArangoDB: https://www.arangodb.com/why-arangodb/cluster/

[29] Retrieved from ArangoDB: (Graph Course for Freshers: The Shortest_Path to first graph skills)

[30] Retrieved from OrientDB: https://orientdb.com/multi-model-database/

[31] Retrieved from GraphDB: https://www.w3.org/standards/semanticweb/

[32] Retrieved from GraphDB: https://ontotext.com/technology-solutions/semantic-tagging/

[33] Retrieved from GraphDB: https://ontotext.com/services/semantic-data-modelling-and-etl/

[34] Retrieved from GraphDB: https://ontotext.com/migration-service-processes/

[35] **Top 5 Considerations When Evaluating NoSQL Databases**. (2018, February). Retrieved from https://webassets.mongodb.com/_com_assets/collateral/10gen_Top_5_NoSQL_Considerations.pdf?_ga=2.50243686.1782174566.1520855471-1516443113.152085 5471

[36] **Top Graph Databases.** (n. d.) . Retrieved from predictiveanalyticstoday: https://www.predictiveanalyticstoday.com/top-graph-databases/

[37] Wisal Khan, E. a. (2017). **Predictive Performance Comparison Analysis of. International Journal of Advanced Computer Science and Applications**, 523-530.

[38] X.Y. Chen (2016, October). **Efficient Historical Query in HBase for Spatio-Temporal Decision Support**. INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL ISSN 1841-9836, 11(5):613-630.

[39] Lior Messinger (2013, Feb 17). **the CAP Theorem** . Retrieved from https://dzone.com/articles/better-explaining-cap-theorem

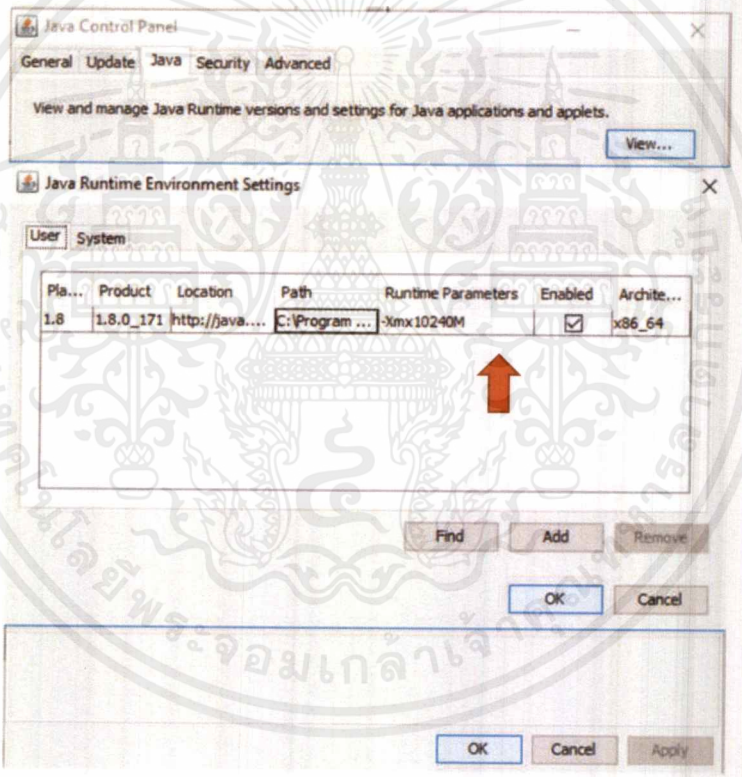
[40] Ravindra Prasad (2017, Aug 26). **CAP Theorem simplified.** Retrieved from https://medium.com/@ravindraprasad/cap-theorem-simplified-28499a67eab4

ภาคผนวก

ภาคผนวก ก
การตั้งค่าของระบบฐานข้อมูล

ก.1 การตั้งค่า Java

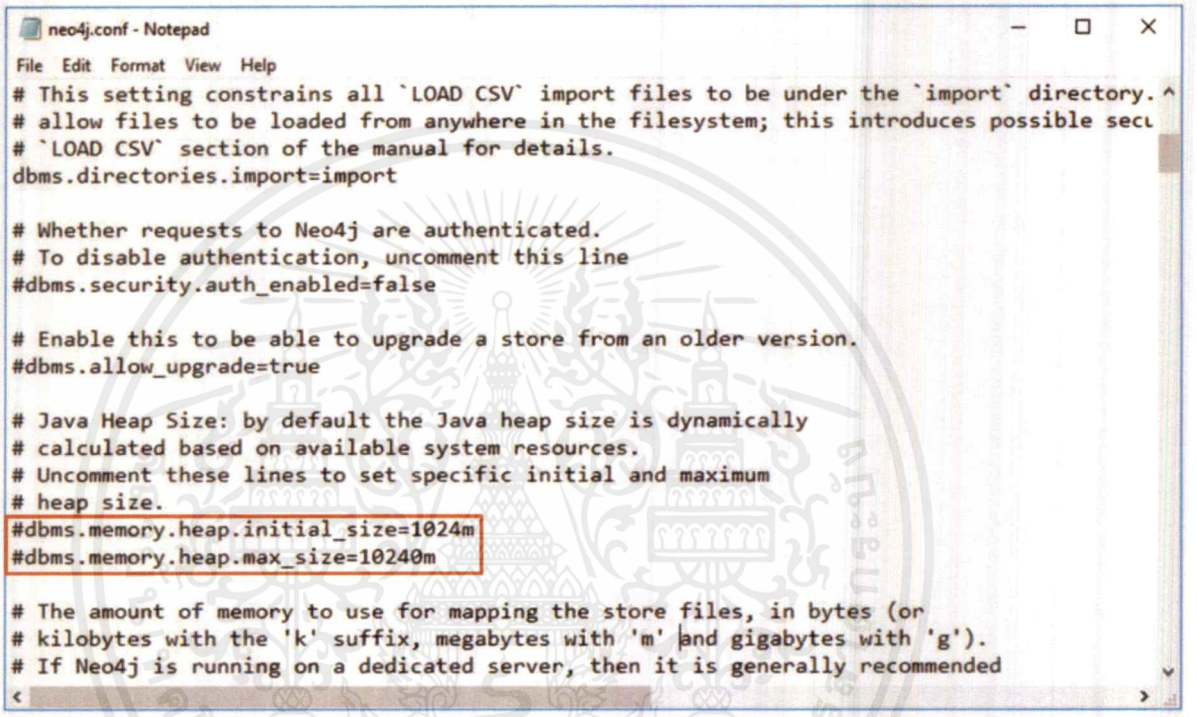
เริ่มต้นจากการเปิดขนาดของการใช้งาน Memory ในระบบฐานข้อมูล Java โดยทำการเพิ่มภายใน Control Panel > All Control Panel Items > Java > เลือก Java (View) และเพิ่มคำสั่ง -Xmx ตามด้วยตัวเลขทรัพยากรตาม RAM ของเครื่องที่ใช้งาน ซึ่งการเพิ่มการใช้งาน Memory จะส่งผลกับทุกระบบฐานข้อมูลที่ใช้ Java เป็น Services



รูปที่ ก.1 หน้าต่างการปรับค่าภายใน Java

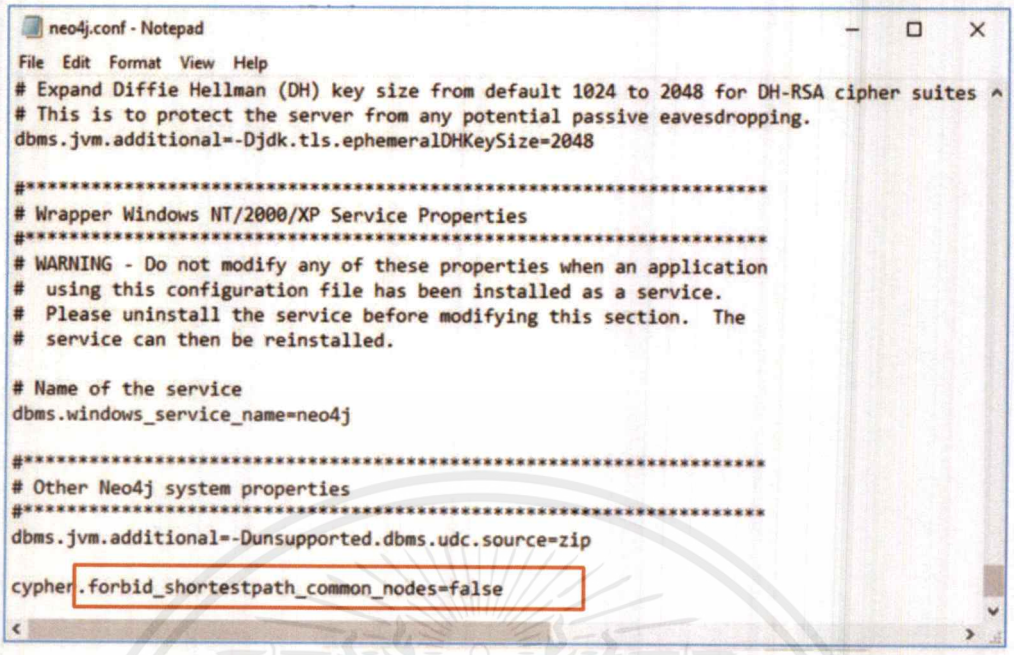
ก.2 การตั้งค่าของระบบฐานข้อมูล Neo4j

1. เปิดไฟล์ neo4j.conf เพื่อทำการแก้ไขการตั้งค่าต่าง ๆ ภายในโฟลเดอร์ neoj4-community-3.3.5 > conf > neo4j.conf ทำการเพิ่มตัวเลขทรัพยากรตาม RAM ของเครื่องที่ใช้งาน



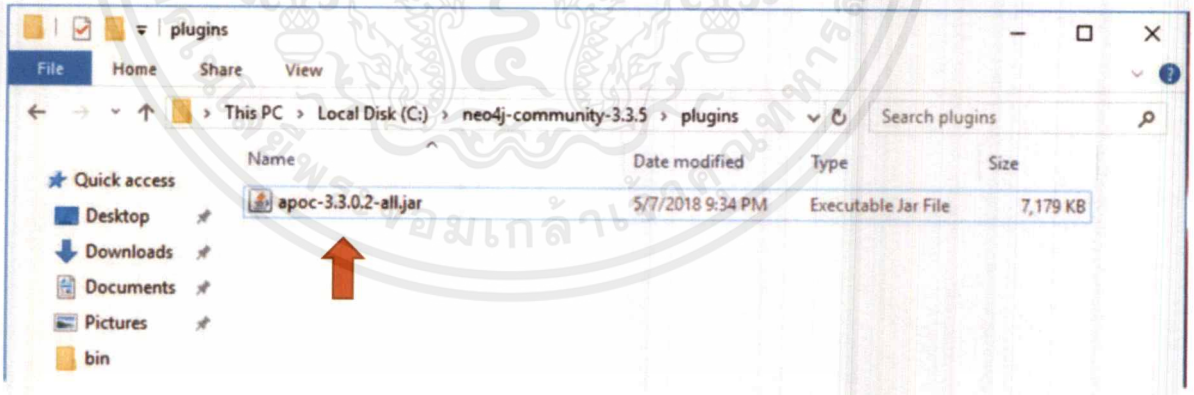
รูปที่ ก.2 หน้าต่างการตั้งค่าไฟล์ neo4j.conf ภายในระบบฐานข้อมูล OrientDB

2. จากนั้นคำการเพิ่มคำสั่ง cypher.forbid_shortestpath_common_nodes=false ลงไปภายในไฟล์ conf เพื่อเปิดการทำงานของฟังก์ชัน Shortest path



รูปที่ ก.3 หน้าต่างการเพิ่มคำสั่งภายในระบบฐานข้อมูล Neo4j

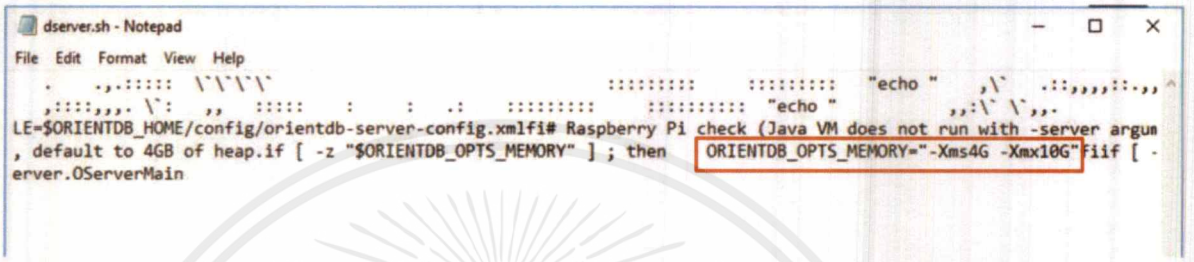
3. ทำการโหลด API เสริมของระบบฐานข้อมูลเพื่อทำการเปิดการทำงานของฟังก์ชัน Breath-First-Search ลงในโฟลเดอร์ neo4j-community-3.3.5 > plugins



รูปที่ ก.4 หน้าต่างการเพิ่ม API ลงในระบบฐานข้อมูล Neo4j

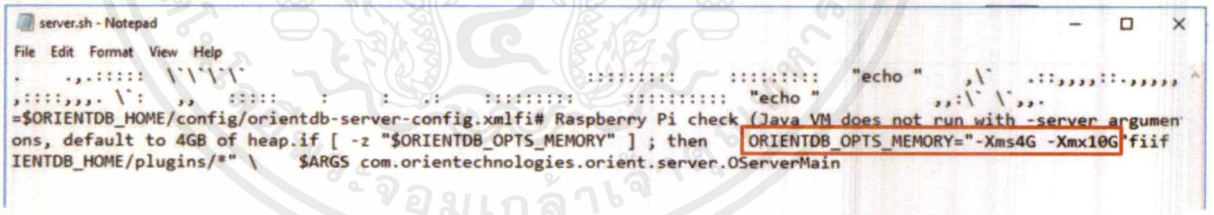
ก.2 การตั้งค่าของระบบฐานข้อมูล OrientDB

1. การเพิ่มการใช้งาน Memory ภายในระบบฐานข้อมูล OrientDB โดยจะเพิ่มจากไฟล์ dserver.sh ในโฟลเดอร์ Orientdb-3.0.1 > bin และทำการแก้ไขตัวเลขทรัพยากรตาม RAM ของเครื่องที่ใช้งาน



รูปที่ ก.5 หน้าต่างการตั้งค่าไฟล์ dserver ภายในระบบฐานข้อมูล OrientDB

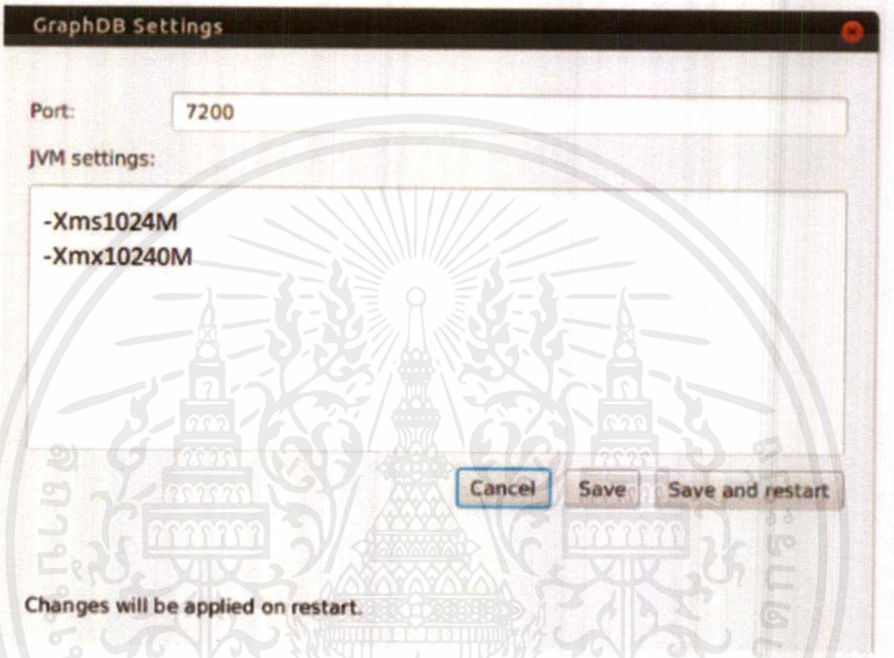
2. การเพิ่มการใช้งาน Memory ภายในระบบฐานข้อมูล OrientDB โดยจะเพิ่มจากไฟล์ server.sh ในโฟลเดอร์ Orientdb-3.0.1 > bin และทำการแก้ไขตัวเลขทรัพยากรตาม RAM ของเครื่องที่ใช้งาน โดยจะตั้งค่าให้ตัวเลขตรงกันทั้ง 2 ไฟล์



รูปที่ ก.6 หน้าต่างการตั้งค่าไฟล์ server ภายในระบบฐานข้อมูล OrientDB

ก.3 การตั้งค่าของระบบฐานข้อมูล GraphDB

1. การเพิ่มการใช้งาน Memory ภายในระบบฐานข้อมูล GraphDB ทำโดยการเปิดระบบฐานข้อมูลขึ้นแล้ว คลิ๊กไปที่ Setting... เพื่อทำการเปิด และทำการเพิ่มตัวเลขทรัพยากรตาม RAM ของเครื่องที่ใช้งาน



รูปที่ ก.7 หน้าต่างการตั้งค่าภายในระบบฐานข้อมูล GraphDB

ภาคผนวก ข
การทดลองอื่น ๆ

ข.1 การทดลองฟังก์ชันของแต่ละระบบฐานข้อมูล

การทดลองฟังก์ชันภายในของแต่ละระบบฐานข้อมูล โดยการทดลองจะฟังก์ชันดังนี้ BETWEEN, LIKE, และ IN ซึ่งจะเป็นการทดลองและเปรียบเทียบ syntax กับระบบฐานข้อมูลอื่น ๆ ที่ใช้ในการทำวิจัยนี้

ข.2 การทดลองฟังก์ชันของระบบฐานข้อมูล Neo4j

1. การทดลองฟังก์ชัน BETWEEN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง " ค้นหาหมวดหมู่หนังสือที่มีชื่อระหว่าง accd ไปจนถึง jcbk จำนวน 10000 ค่าแรก " โดยใช้เวลาใน การประมวลผล 3 วินาที

รูปที่ ข.1 หน้าแสดงผลการทำงานฟังก์ชัน BETWEEN ของระบบฐานข้อมูล Neo4j

2. การทดลองฟังก์ชัน LIKE โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง " ค้นหาหมวดหมู่หนังสือที่มีชื่อขึ้นต้นด้วยตัว j จำนวน 10000 ค่าแรก " โดยใช้เวลาในการประมวลผล 2 วินาที

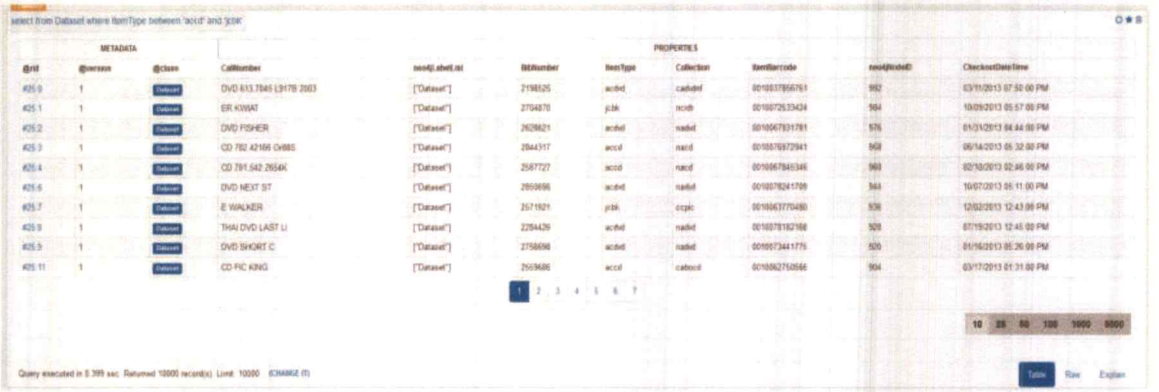
รูปที่ ข.2 หน้าแสดงผลการทำงานฟังก์ชัน LIKE ของระบบฐานข้อมูล Neo4j

3. การทดลองฟังก์ชัน IN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง " ค้นหาหมวดหมู่หนังสือที่มีชื่อ accd กับ jcbk จำนวน 10000 ค่าแรก " โดยใช้เวลาในการประมวลผล 3 วินาที

รูปที่ ข.3 หน้าแสดงผลการทำงานฟังก์ชัน IN ของระบบฐานข้อมูล Neo4j

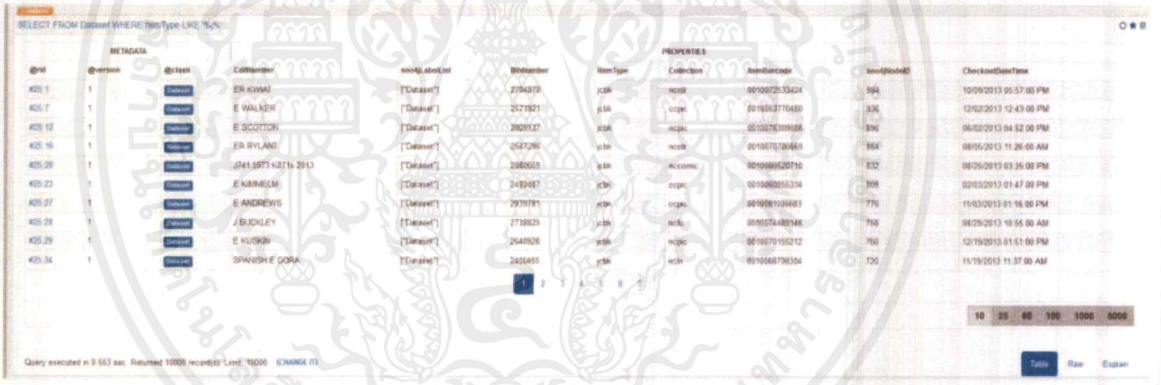
ข.3 การทดลองฟังก์ชันของระบบฐานข้อมูล OrientDB

1. การทดลองฟังก์ชัน BETWEEN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง " ค้นหาหมวดหมู่หนังสือที่มีชื่อระหว่าง accd ไปจนถึง jcbk จำนวน 10000 ค่าแรก " โดยใช้เวลาในการประมวลผล 24 วินาที



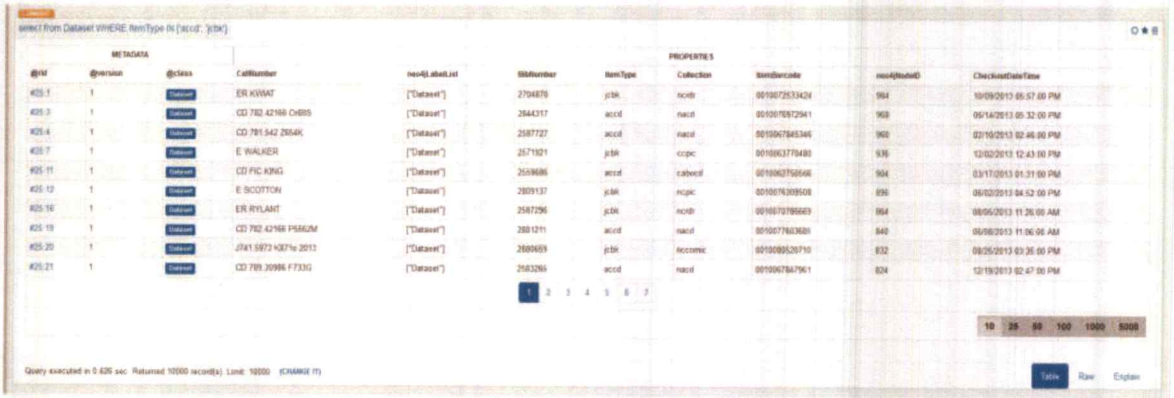
รูปที่ ข.4 หน้าแสดงผลการทำงานฟังก์ชัน BETWEEN ของ OrientDB

2. การทดลองฟังก์ชัน LIKE โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง “ ค้นหาหมวดหมู่หนังสือที่มีชื่อขึ้นต้นด้วยตัว j จำนวน 10000 ค่าแรก “ โดยใช้เวลาในการประมวลผล 25 วินาที



รูปที่ ข.5 หน้าแสดงผลการทำงานฟังก์ชัน LIKE ของ OrientDB

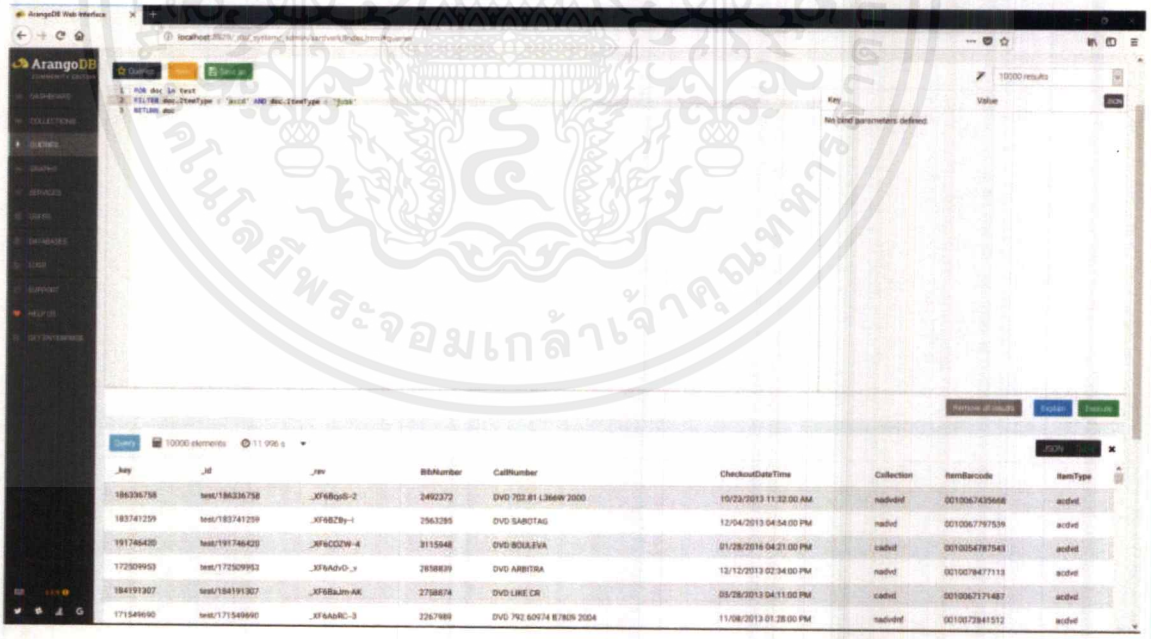
3. การทดลองฟังก์ชัน IN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง “ ค้นหาหมวดหมู่หนังสือที่มีชื่อ accd กับ jcbk จำนวน 10000 ค่าแรก “ โดยใช้เวลาในการประมวลผล 28 วินาที



รูปที่ ข.6 หน้าแสดงผลการทำงานฟังก์ชัน IN ของ OrientDB

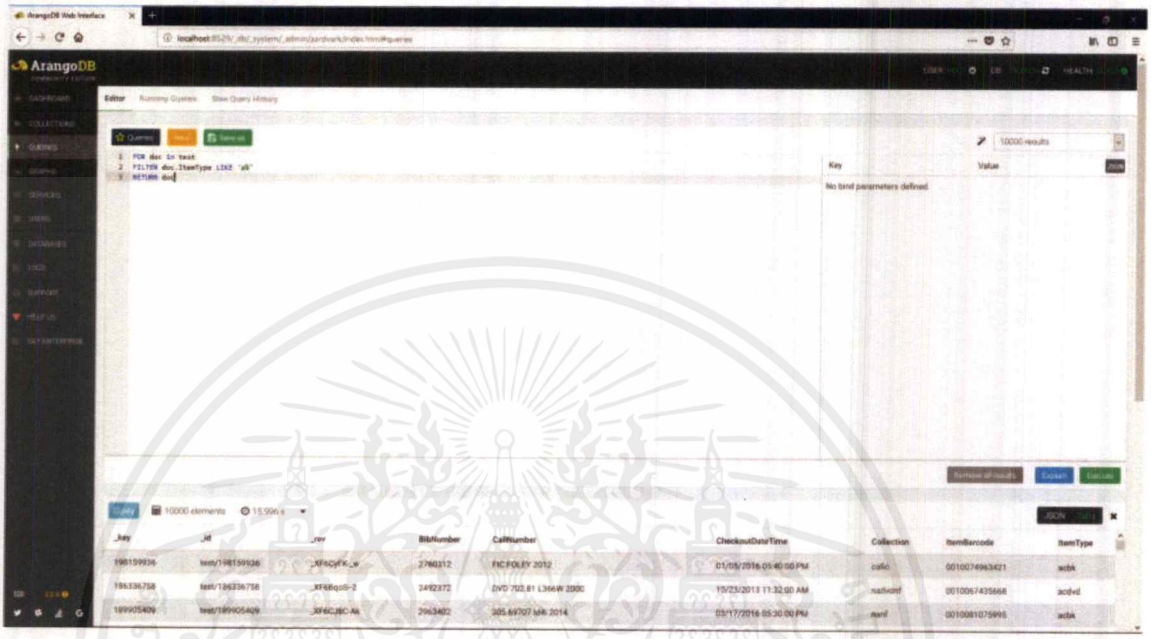
ข.4 การทดลองฟังก์ชันของระบบฐานข้อมูล ArangoDB

1. การทดลองฟังก์ชัน BETWEEN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง " ค้นหาหมวดหมู่หนังสือที่มีชื่อระหว่าง accd ไปจนถึง jcbk จำนวน 10000 ค่าแรก " โดยใช้เวลาในการประมวลผล 16 วินาที



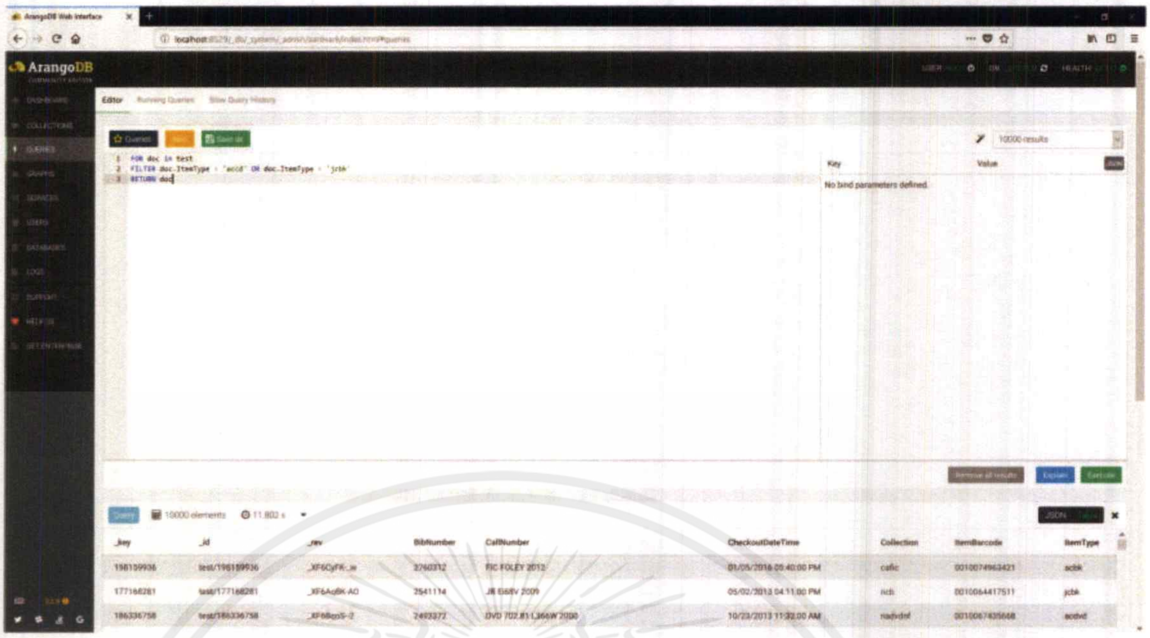
รูปที่ ข.7 หน้าแสดงผลการทำงานฟังก์ชัน BETWEEN ของ ArangoDB

2. การทดลองฟังก์ชัน LIKE โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง “ ค้นหาหมวดหมู่หนังสือที่มีชื่อขึ้นต้นด้วยตัว j จำนวน 10000 ค่าแรก “ โดยใช้เวลาในการประมวลผล 19 วินาที



รูปที่ ข.8 หน้าแสดงผลการทำงานฟังก์ชัน LIKE ของ ArangoDB

3. การทดลองฟังก์ชัน IN โดยคำสั่งที่ใช้ในการค้นหาจะหมายถึง “ ค้นหาหมวดหมู่หนังสือที่มีชื่อ accd กับ jcbk จำนวน 10000 ค่าแรก ” โดยใช้เวลาในการประมวลผล 15 วินาที



รูปที่ ข.9 หน้าแสดงผลการทำงานฟังก์ชัน IN ของ ArangoDB