การพัฒนาโปรแกรมประยุกต์เชิงเวลาโดยระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล

TEMPORAL APPLICATION DEVELOPMENT USING A NOSQL DATABASE SYSTEM

ษฎาพร พฤกษาชีวะ

ธีรภัทร งามพิเศษศักดิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

การพัฒนาโปรแกรมประยุกต์เชิงเวลาโดยระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล

TEMPORAL APPLICATION DEVELOPMENT USING A NOSQL DATABASE SYETEM

ชฎาพร พฤกษาชีวะ

ธีรภัทร งามพิเศษศักดิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

ปริญญานิพนธ์ปีการศึกษา 2557

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาโปรแกรมประยุกต์เชิงเวลาโดยระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล

TEMPORAL APPLICATION DEVELOPMENT USING A NOSQL DATABASE SYSTEM

ผู้จัดทำ

1. นางสาวชฎาพร พฤกษาชีวะ รหัสนักศึกษา 54010258
2. นายธีรภัทร งามพิเศษศักดิ์ รหัสนักศึกษา 54010638

อาจารย์ที่ปรึกษา

(รศ. ดร. ศุภมิตร จิตตะยโศธร)

การพัฒนาโปรแกรมประยุกต์เชิงเวลาโดยระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล

นางสาวชฎาพร	พฤกษาชีวะ	54010258
นายธีรภัทร	งามพิเศษศักดิ์	54010638
รศ. ดร. สุภมิตร	จิตตะยโศธร	อาจารย์ที่ปรึกษา

ปีการศึกษา 2557

บทคัดย่อ

ในปัจจุบันข้อมูลต่างๆมีความสัมพันธ์ทางเชิงเวลาเพิ่มมากขึ้นไม่ว่าจะเป็นในงานทางด้านธุรกิจ ด้านการท่องเที่ยว ด้านประวัติศาสตร์เป็นต้น ทำให้สามารถสืบค้นข้อมูลที่ต้องการทราบในช่วงเวลาหนึ่งได้ อีกทั้งปัจจุบันนี้ได้มีการนำระบบฐานข้อมูลรูปแบบใหม่ๆที่ไม่ใช่รูปแบบการจัดเก็บข้อมูลเชิงสัมพันธ์(Relational Model) นำใช้ในองค์กรกันมากขึ้น เพื่อให้สอดคล้องกับข้อมูลและรูปแบบการใช้งาน จึงทำให้เกิดแนวคิดการนำระบบฐานข้อมูลเชิงเวลามาประยุกต์ใช้งานบนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล (NoSQL Database System)

โครงงานนี้ได้ทำการพัฒนาโปรแกรมประยุกต์(Application) โดยใช้งานระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ซึ่งถูกออกแบบมาให้สามารถจัดเก็บข้อมูลเชิงเวลาได้ โปรแกรมประยุกต์นี้จะเข้าไปช่วยการทำงานทางฝั่งผู้ใช้งานให้สามารถใช้งานข้อมูลเชิงเวลาได้ง่ายมากขึ้นโดยไม่ต้องใช้คำสั่งที่ยุ่งยากในการส่งคำสั่งในการสอบถาม(Query) ในส่วนของระบบฐานข้อมูลจะเป็นการเก็บข้อมูลเชิงเวลาของเส้นทางการเดินทาง และข้อมูลที่เกี่ยวข้องกับเหตุการณ์และกิจกรรมต่างๆ ที่เกิดบนเส้นทางของการเดินทางนั้น ในแต่ละช่วงเวลาทั้งอดีต ปัจจุบัน และอนาคต ทำให้สามารถช่วยในการวางแผนการเดินทางท่องเที่ยวหรือการเดินทางในชีวิตประจำวันได้ ซึ่งเป็นทางเลือกหนึ่งในการใช้ประโยชน์จากความสามารถของระบบฐานข้อมูลเชิงเวลา

TEMPORAL APPLICATION DEVELOPMENT USING A NOSQL DATABASE SYETEM

Miss Chadaporn	Phruksacheewa	54010258
Mister Theerapat	Ngampisatsak	54010638
Assoc. Prof. Dr. Suphamit	Chittayasothorn	Advisor

Academic Year 2014

ABSTRACT

At present, the Information is more associate with time-variant such as information of business, tourism and history etc. So that users can search information which they need to know at point of time. Moreover, the implementing current database system, not a relational model is widely used in organization in order to correspond with business operation and can be integrated into their business process. Thus, the implementation of temporal database applied on NoSQL Database System concept has been presented.

Our project has developed an application using a NoSQL database system which are designed to store temporal data. This application can help users to be able to manage temporal data without the hassle of sending query command. It makes them quite easy to be understood by the final users. In the part of database system can be store temporal data of travel routes and information related to events or activities which happen on the routes in the past, present and future. Therefore, this can help for planning trip or daily trip. It is an alternative to take advantage of the capabilities of temporal database.

กิตติกรรมประกาศ

ขอขอบคุณ รศ. ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษาโครงงานที่ให้ความรู้ ในเรื่องเนื้อหาการเรียน เรื่องการใช้ชีวิต รวมทั้งการแลกเปลี่ยนความคิดเห็นและแนะนำการทำโครงงานในด้านต่างๆตลอดมาอย่างใกล้ชิดเพื่อให้สามารถปฏิบัติงานได้อย่างลุล่วงตามเป้าหมาย

ขอขอบคุณครอบครัวพฤกษาชีวะและครอบครัวงามพิเศษศักดิ์ ที่ให้ความรักความเข้าใจ การอบรมสั่งสอน เลี้ยงดู ให้โอกาสทางการศึกษา และให้การสนับสนุนในทุกๆ ด้านมาโดยตลอด

ขอขอบคุณรุ่นพี่ รวมทั้งเพื่อนๆ ทุกคนที่มีส่วนร่วมประติดประต่อความรู้ความสามารถ แลกเปลี่ยนความคิดเห็นและแนะนำการทำโครงงานในด้านต่างๆตลอดมา

ขอขอบคุณห้องปฏิบัติงานทางด้านฮาร์ดแวร์ ที่ให้โอกาสในการเรียนรู้และมิตรภาพจากที่แห่งนี้ อีกทั้งยังเป็นสถานที่ทำงานตลอดโครงงานจนสำเร็จลุล่วงไปได้ด้วยดี

ขอขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ได้อบรมสั่งสอนวิชาความรู้ต่างๆ แก่ข้าพเจ้ามาโดยตลอด ทำให้ข้าพเจ้าสามารถนำความรู้เหล่านั้นมาใช้พัฒนาโครงงานนี้จนสำเร็จลุล่วงไปได้ด้วยดี

สุดท้ายนี้ขอขอบพระคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ให้ข้าพเจ้าได้เข้ามาศึกษาหาความรู้ที่นี่ ข้าพเจ้ารู้สึกเป็นเกียรติอย่างยิ่ง คุณความดีใดๆ ที่ปรากฏในโครงงานนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่านมา ณ ที่นี้

ชฎาพร พฤกษาชีวะ
ธีรภัทร งามพิเศษศักดิ์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

รูป หน้า

สารบัญรูป (ต่อ)

รูป หน้า

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

สืบเนื่องจากการทำงานรูปแบบการจัดเก็บข้อมูลเชิงสัมพันธ์(Relational Model)นั้น ในปัจจุบันได้มีการรองรับการใช้งานในส่วนของระบบฐานข้อมูลเชิงเวลา(Temporal Database) เพิ่มเข้ามา อีกทั้งในปัจจุบันนั้นได้มีการนำระบบฐานข้อมูลรูปแบบใหม่ๆที่ไม่ใช่แบบจำลองข้อมูลเชิงสัมพันธ์นำมาใช้ในงานหรือในองค์กรกันมากขึ้น เพื่อให้สอดคล้องกับข้อมูลและรูปแบบการใช้งาน จึงทำให้เกิดแนวคิดการนำระบบฐานข้อมูลเชิงเวลามาประยุกต์ใช้งานบนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล (NoSQL Database System)

โครงงานนี้จัดทำขึ้นเพื่อสร้างโปรแกรมประยุกต์ที่สามารถใช้แปลงคำสั่งการสอบถาม (Query) รูปแบบของภาษาไซเฟอร์(Cypher)ในรูปแบบปกติ ให้เป็นรูปแบบคำสั่งการสอบถามของภาษาไซเฟอร์ที่สามารถใช้งานในระบบฐานข้อมูลเชิงเวลาโดยพัฒนาบนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ซึ่งระบบฐานข้อมูลเชิงเวลานั้นมีความสัมพันธ์ที่ซับซ้อนทั้งรูปแบบการจัดเก็บข้อมูล การค้นคืน และการปรับปรุงข้อมูลที่แตกต่างจากการทำงานของการจัดเก็บข้อมูลทั่วไป ซึ่งผู้ใช้งานจำเป็นต้องมีความรู้เกี่ยวกับการทำงานของระบบฐานข้อมูลเชิงเวลาในระดับหนึ่งเป็นพื้นฐาน จึงจะสามารถใช้งานได้ โปรแกรมประยุกต์นี้จะเข้าไปช่วยการทำงานทางฝั่งผู้ใช้งานให้สามารถใช้งานได้ง่ายมากขึ้นโดยไม่ต้องใช้คำสั่งที่ยุ่งยากในการส่งคำสั่งในการสอบถาม โดยในอีกส่วนหนึ่งของโปรแกรมประยุกต์ที่ได้จัดทำขึ้นเพื่อจำลองรูปแบบการใช้งานระบบฐานข้อมูลเชิงเวลา โดยใช้ในด้านการค้นหาเส้นทางของการเดินทาง และทำการจัดเก็บข้อมูลที่เกี่ยวข้องกับกิจกรรมต่างๆ ที่สามารถเกิดเหตุการณ์บนเส้นทางของการเดินทางนั้น ในแต่ละช่วงเวลาอดีต ปัจจุบัน และอนาคต ทำให้สามารถช่วยวางแผนการเดินทางในการท่องเที่ยวหรือการเดินทางในชีวิตประจำวันได้ อีกทั้งยังเป็นทางเลือกหนึ่งที่ทำให้สามารถใช้รูปแบบฐานข้อมูลเชิงเวลาได้โดยที่ไม่ต้องเสียค่าใช้จ่ายในการซื้อโปรแกรมที่รองรับการทำงานระบบฐานข้อมูลเชิงเวลาที่มีราคาค่อนข้างแพง

1.2 วัตถุประสงค์ของโครงงาน

1) เพื่อศึกษารูปแบบการใช้งานและประโยชน์ของระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล
2) เพื่อศึกษาโครงสร้างข้อมูล กฎข้อบังคับของภาษาและการใช้งานของระบบฐานของมูลทเป็นรูปแบบกราฟ(Graph Database) ซึ่งเป็นรูปแบบหนึ่งของระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล
3) เพื่อพัฒนาการใช้งานระบบฐานข้อมูลเชิงเวลาให้สามารถใช้งานกับระบบจัดการฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล
4) เพื่อลดความซับซ้อนในการใช้งานระบบฐานข้อมูลเชิงเวลาทางฝั่งผู้ใช้งาน
5) เพื่อเป็นโปรแกรมทางเลือกในการใช้งานระบฐานข้อมูลเชิงเวลา

1.3 ขอบเขตของโครงงาน

โครงงานนี้เป็นการศึกษาการทำงานของระบบฐานข้อมูลเชิงเวลา ซึ่งพัฒนาให้สามารถใช้งานได้บนระบบการจัดการฐานข้อมูลที่ไม่ใช้ภาษาแอสคิวแอล โดยจะมุ่งเน้นเฉพาะในส่วนของการทำงานของระบบฐานข้อมูลเชิงเวลาบนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ซึ่งจัดการทางด้านการใช้งานทางฝั่งผู้ใช้ เพื่อลดความยุ่งยากในการส่งคำสั่งการสอบถามในรูปแบบของฐานข้อมูลเชิงเวลา และออกแบบจำลองการใช้งานของฐานข้อมูลเชิงเวลาในรูปแบบของการค้นหาเส้นทางการเดินทางในระบบหรือค้นหากิจกรรมและกิจกรรมที่จะเกิดเหตุการณ์ขึ้นตามช่วงถนนของการเดินทางผ่านทางระบบฐานข้อมูลรูปแบบกราฟผ่านหน้าเว็บแอปพลิเคชัน

1.4 วิธีการดำเนินงาน

1) ศึกษาทฤษฎีของระบบฐานข้อมูลเชิงเวลา (Temporal database)
2) ศึกษาระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล เพื่อนำมาวิเคราะห์ถึงข้อดีข้อเสียใช้งานให้ตรงกับความต้องการของโปรแกรมประยุกต์
3) ศึกษาระบบฐานข้อมูลที่จัดเก็บในรูปแบบกราฟ โครงสร้างภาษา กฎเกณฑ์ของตัวฐานข้อมูลที่เลือก
4) ศึกษาการใช้งานภาษาไซเฟอร์(Cypher) เพื่อทำการออกแบบ รูปแบบของคำสั่งในการสอบถามของทางฝั่งผู้ใช้งาน โดยลดความยุ่งยากในการใช้งานทางของผู้ใช้งาน
5) วิเคราะห์และออกแบบฐานข้อมูลของโปรแกรมประยุกต์ให้สามรถใช้งานในรูปแบบของฐานข้อมูลเชิงเวลา
6) โปรแกรมประยุกต์พัฒนาการใช้งานผ่านเว็บแอปพลิเคชั่น

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ความรู้ความเข้าใจในแนวคิดและหลักการทำงานเกี่ยวกับระบบฐานข้อมูลเชิงเวลา (Temporal Database)
2) *ความรู้และหลักการทำงานเกี่ยวกับระบบฐานข้อมูลที่เป็นรูปแบบกราฟ*
3) สามารถนำความรู้ที่ได้จากกการศึกษาระบบฐานข้อมูลเชิงเวลา(Temporal Database) มาออกแบบระบบฐานข้อมูลเชิงเวลาเพื่อประยุกต์ใช้งานบนระบบจัดการฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอลได้

บทที่ 2

แนวคิดและทฤษฎีที่เกี่ยวข้อง

2.1 ฐานข้อมูลเชิงเวลา (Temporal Database)

ฐานข้อมูลเชิงเวลา คือ ฐานข้อมูลที่เก็บข้อมูลที่มีการเปลี่ยนแปลงตามเวลา(Time-varying) ซึ่งในเวลาต่อมาได้มีการถกเถียงกันถึงในกรณีที่ข้อมูลมีการเปลี่ยนแปลงแต่ไม่ได้ขึ้นอยู่กับเวลา ตัวอย่างเช่น ความสูงของภูเขาที่วัดเมื่อ๘๐ปีก่อกับปัจจุบันที่วัดได้ไม่เท่ากัน หรือความสว่างของดาวที่วัดเมื่อ๘๐ปีก่อกับปัจจุบันที่วัดได้ไม่เท่ากัน ทั้งนี้เกิดจากการที่เทคโนโลยีมีการพัฒนาไปมากการวัดค่าต่างๆจึงสามารถทำได้ละเอียดมากขึ้นค่าที่ได้จึงไม่เท่าเดิม ซึ่งฐานข้อมูลเชิงเวลาก็สามารถเก็บข้อมูลประเภทนี้ได้เช่นกัน ทำให้นิยามเดิมไม่ครอบคลุม จึงเกิดนิยามใหม่ที่ครอบคลุมกว่า คือ ฐานข้อมูลเชิงเวลา หมายถึง ฐานข้อมูลที่สนับสนุนแง่มุมบางแง่มุมของเวลาแต่ไม่นับรวม user-defined time โดย user-defined time คือ Date-time ที่เป็นส่วนหนึ่งของความจริง(fact) เช่น วันเกิด เป็นต้น

2.1.1 ประเภทของเวลาที่ใช้ในระบบฐานข้อมูล

เวลาที่ใช้ในระบบฐานข้อมูล แบ่งออกเป็น 3 ประเภทดังนี้

1) Valid Time
2) Transaction Time
3) User-defined Time

โดยแต่ละประเภทมีคุณสมบัติดังนี้

ตาราง 2.1 คุณสมบัติของเวลาแต่ละประเภท [ที่มา:Managing Temporal Data A Five-Part Series]

Temporal aspect	Valid Time	Transaction Time	User-defined Time
คำจำกัดความ	เวลาที่ Fact นั้นเป็นจริงในความเป็นจริง	เวลาที่ข้อมูลถูกบันทึกลงในฐานข้อมูล	เวลาที่เป็นส่วนหนึ่งของ Fact เช่น วันเกิด
จัดการโดย	ผู้ใช้	ระบบ	ผู้ใช้
สนับสนุนโดย	ระบบ	ระบบ	ผู้ใช้
มีภาษาในการจัดการ	ใช่	ใช่	ไม่

ตาราง 2.2 ฐานข้อมูลที่ใช้รูปแบบ Valid Time ในตาราง [ที่มา:Managing Temporal Data A Five-Part Series]

FDYD_ID	LOT_ID_NUM	PEN_ID	HD_CNT	FROM_DATE	TO_DATE
1	137	1	17	1998-02-07	1998-02-18
1	219	1	43	1998-02-25	1998-03-01
1	219	1	20	1998-03-01	1998-03-14
1	219	2	23	1998-03-01	1998-03-14
1	219	2	43	1998-03-14	9999-12-31
1	374	1	14	1998-02-20	9999-12-31

ตาราง 2.3 ฐานข้อมูลที่ใช้รูปแบบ Transaction Time ในตาราง [ที่มา:Managing Temporal Data A Five-Part Series]

RA_ Hour	RA_ Min	RA_ Sec	Dec-Degree	Dec-Minute	Discoverer	Mag-First	Trans_Start	Trans_Stop
00	00	00	75	30	'A 1248'	12.0	1989-03-12	1992-11-15
00	00	09	75	30	'A 1248'	12.0	1992-11-15	1994-05-18
00	00	09	75	30	'A 1248'	10.5	1994-05-18	1995-07-23
00	00	08	75	30	'A 1248'	10.5	1995-07-23	9999-12-31
05	57	40	00	02	'BU 1190'	6.5	1988-11-08	9999-12-31
04	13	20	50	32	'CHR 15'	15.5	1990-02-09	9999-12-31
01	23	70	-09	55	'HJ 3433'	10.5	1991-03-25	9999-12-31
02	33	10	-09	25	'LDS3402'	10.5	1993-12-19	1996-07-09

2.1.2 ความซ้ำซ้อนของข้อมูลเชิงเวลา

ความซ้ำซ้อนที่เกิดขึ้นในระบบฐานข้อมูลเชิงเวลาจะส่งผลให้เกิดปัญหาในการเพิ่มข้อมูล (Insert) การปรับปรุงข้อมูล (Update) และการลบข้อมูล (Delete) โดยสามารถแบ่งได้เป็น4 ประเภท ดังนี้

1) Sequenced duplicates คือการที่ข้อมูลซ้ำกันในช่วงเวลาหนึ่ง
2) Current duplicates คือการที่ข้อมูลซ้ำกันเมื่อดูจากข้อมูลที่เป็นจริงในช่วงปัจจุบัน
3) Value-equivalent duplicates คือการที่ข้อมูลซ้ำกันแต่เป็นจริงคนละช่วงเวลา
4) Non-sequenced duplicates คือการซ้ำซ้อนโดยที่มีข้อมูลเหมือนกันทุกประการ

ซึ่งแต่ละประเภทก็จะมีการคาบเกี่ยวกันดังนี้

ตาราง 2.4 ความสัมพันธ์ระหว่างความซ้ำซ้อนแต่ละประเภท [ที่มา:Managing Temporal Data A Five-Part Series]

	Sequenced	Current	Value-equivalent	Nonsequenced
Sequenced	✓		✓	
Current	✓	✓	✓	
Value-equivalent			✓	
Nonsequenced	✓		✓	✓

2.1.3 ตารางฐานข้อมูลเชิงเวลาประเภท Valid time state table

Valid time state table คือ ตารางที่เพิ่ม valid time เข้ามา เพื่อระบุว่าข้อมูลนั้นเป็นจริงในช่วงระยะเวลาใด

ตาราง 2.5 บันทึกประวัติวัวในฟาร์ม [ที่มา:Managing Temporal Data A Five-Part Series]

FDYD_ID	LOT_ID_NUM	PEN_ID	HD_CNT	FROM_DATE	TO_DATE
1	137	1	17	1998-02-07	1998-02-18
1	219	1	43	1998-02-25	1998-03-01
1	219	1	20	1998-03-01	1998-03-14
1	219	2	23	1998-03-01	1998-03-14
1	219	2	43	1998-03-14	9999-12-31
1	374	1	14	1998-02-20	9999-12-31

ตารางนี้นำเสนอประวัติของวัว ว่าเคยอยู่ในคอกใด ช่วงเวลาใดบ้าง โดยบันทึกข้อมูลดังนี้ FDYD_ID คือรหัสลานให้อาหารของวัว, LOT_ID_NUM คือเลขระบุฝูงของวัว, PEN_ID คือรหัสคอกวัว, HD_CNT คือจำนวนวัว และ FROM_DATE กับ TO_DATE แทนช่วงเวลาที่ข้อมูลนั้นเป็นจริง โดยหาก TO_DATE เป็น 9999-12-31 หมายถึงข้อมูลนั้นยังเป็นจริงมาจนถึงปัจจุบัน จากตารางข้างต้นจะใช้ในการอธิบายการทำงานรูปแบบต่างของระบบฐานข้อมูลเชิงเวลา

2.1.4 Temporal Query

การเก็บข้อมูลในรูปแบบระบบฐานข้อมูลเชิงเวลาจะช่วยให้การสอบถาม(Query) สามารถทำได้หลากหลายมากยิ่งขึ้นและช่วยในการตอบโจทย์บางข้อที่ไม่สามารถตอบได้หากใช้ระบบฐานข้อมูลธรรมดา ซึ่งในที่นี้จะเรียกการสอบถามที่ใช้กับระบบฐานข้อมูลเชิงเวลาว่า "Temporal Query" โดยสามารถแบ่งการสอบถามออกเป็นประเภทได้ดังนี้

1) Current Query คือการทำการสอบถาม เพื่อหาข้อมูลที่เป็นจริงอยู่ในปัจจุบัน ตัวอย่างคำถามประเภทนี้ (อ้างอิงจากตาราง2.5) เช่น "ในปัจจุบันวัวฝูง 219 ในลานให้อาหาร 1 อยู่ในคอกใดบ้างจำนวนเท่าใด" จะเห็นได้ว่าหากเป็นการสอบถาม ระบบฐานข้อมูลทั่วไปจะใช้คำสั่ง SQL ดังนี้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม [ที่มา:Managing Temporal Data A Five-Part Series]

```
SELECT PEN_ID, HD_CNT
FROM LOT_LOC
WHERE FDYD_ID = 1 AND LOT_ID_NUM = 219
```

แต่ระบบฐานข้อมูลเชิงเวลาจะต้องระบุช่วงเวลาปัจจุบันลงไปด้วยดังนี้

```
SELECT PEN_ID, HD_CNT
FROM LOT_LOC
WHERE FDYD_ID = 1 AND LOT_ID_NUM = 219 AND TO_DATE =
DATE '9999-12-31'
```

ซึ่งผลลัพธ์ของการสอบถามจะได้ผลดังนี้

ตาราง 2.6 ผลลัพธ์ของการ current query [ที่มา:Managing Temporal Data A Five-Part Series]

PEN_ID	HD_CNT
2	43

2) Sequence Query คือการสอบถาม ถึงข้อมูลต่างๆที่เปลี่ยนแปลงไปในอดีตจนถึงปัจจุบันเพื่อต้องการทราบประวัติการเปลี่ยนแปลงของข้อมูลตัว อย่างคำถามเช่น "วัวฝูง 219 เคยอยู่ในคอกใดบ้างและคอกละกี่ตัว" คำถามลักษณะนี้จะไม่สามารถตอบได้หากใช้ระบบฐานข้อมูลปกติ สามารถเขียน SQL เพื่อตอบคำถามดังกล่าวได้ดังนี้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม [ที่มา:Managing Temporal Data A Five-Part Series]

```
SELECT PEN_ID, HD_CNT,
FROM_DATE, TO_DATE FROM LOT_LOC
WHERE FDYD _D = 1 AND LOT_ID_NUM = 219
```

การไม่ระบุเวลาในระบบฐานข้อมูลเชิงเวลา จะหมายถึงการนับรวมช่วงเวลาทั้งหมดจากอดีตจนถึงปัจจุบันซึ่งจะให้ผลลัพธ์ดังนี้

ตาราง 2.7 ผลลัพธ์ของการ *Sequence Query* [ที่มา:*Managing Temporal Data A Five-Part Series*]

PEN_ID	HD_CNT	FROM_DATE	TO_DATE
1	43	1998-02-25	1998-03-01
1	20	1998-03-01	1998-03-14
2	23	1998-03-01	1998-03-14
2	43	1998-03-14	9999-12-31

3) Non-Sequence Query คือการสอบถามประเภทนี้จะไม่สนใจเรื่องลำดับช่วงเวลา ตัวอย่างคำถามเช่น "มีวัวฝูงใดบ้างเคยอยู่คอกเดียวกันแต่คนละเวลา" การจะตอบคำถามนี้จำเป็นจะต้องใช้ระบบฐานข้อมูลเชิงเวลาเท่านั้นเนื่องจากเป็นการถามถึงข้อมูลในอดีตที่ปัจจุบันมีการเปลี่ยนแปลงไปแล้ว สามารถเขียน SQL เพื่อตอบคำถามดังกล่าวได้ดังนี้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม[ที่มา:Managing Temporal Data A Five-Part Series]

```
SELECT L1.LOT_ID NUM, L2.LOT_ID NUM, L1.PEN_ID
FROM LOT LOC AS L1, LOT LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
      AND L1.FDYD_ID = L2.FDYD_ID
      AND L1.PEN_ID = L2.PEN_ID
```

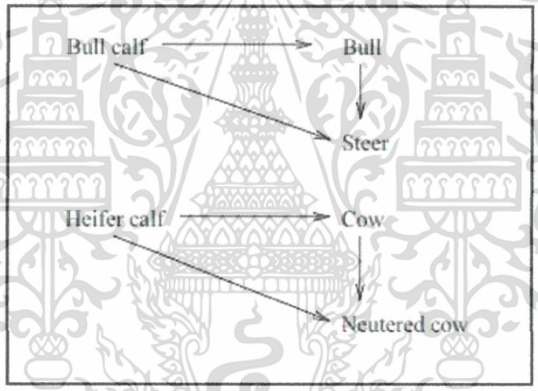
ซึ่งจะให้ผลลัพธ์ดังนี้

ตาราง 2.7 ผลลัพธ์ของการ Non-Sequence Query [ที่มา:Managing Temporal Data A Five-Part Series]

L1	L2	PEN_ID
137	219	1
137	219	1
137	374	1
219	374	1
219	374	1

2.1.5 Modifications

การเพิ่มข้อมูล (Insert) การปรับปรุงข้อมูล (Update) และการลบข้อมูล (Delete) สำหรับระบบฐานข้อมูลเชิงเวลานั้นจะแตกต่างกับระบบฐานข้อมูลปกติโดยจะอธิบายโดยใช้ข้อมูลเหล่านี้



รูป 2.1 สถานะของวัว

(ที่มา: Managing Temporal Data A Five-Part Series)

ตาราง 2.8 สถานะของฝูงวัว [ที่มา:Managing Temporal Data A Five-Part Series]

LOT_ID_NUM	GNDR_CODE	FROM_DATE	TO_DATE
101	C	1998-01-01	1998-03-23
101	S	1998-03-23	9999-12-31
234	C	1998-02-17	9999-12-31
799	S	1998-03-12	9999-12-31

การทำการ Insert Update และ Delete สามารถแบ่งได้เป็น 3 ประเภท คือ

1) Current modifications คือการ modifications กับข้อมูลที่เป็นปัจจุบัน
2) Sequence modifications คือการ modifications กับข้อมูลโดยมีการกำหนดช่วงเวลาเริ่มต้นและสิ้นสุด
3) Non-Sequence modifications คือการ modifications กับข้อมูลโดยไม่สนใจเวลา

2.1.5.1 Current modifications

1) Current Insert คือการเพิ่มข้อมูลที่เป็นจริงในปัจจุบันตัวอย่างเช่นต้องการเพิ่มข้อมูลวัวฝูง 433 สถานะ h เป็นจริงในปัจจุบันเขียนคำสั่ง SQL ได้ดังนี้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม[ที่มา:Managing Temporal Data A Five-Part Series]

```
INSERT INTO LOT
VALUES (433, 'h', CURRENT DATE, DATE '9999-12-31')
```

2) Current Delete คือการลบข้อมูลที่เป็นจริงในปัจจุบัน เช่นต้องการ Delete LOT 234 ในปัจจุบัน จะไม่สามารถใช้การ Delete ปกติได้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม [ที่มา:Managing Temporal Data A Five-Part Series]

```
DELETE FROM LOT

WHERE LOT_ID_NUM = 234
```

เนื่องจากต้องการลบเฉพาะข้อมูลส่วนที่เป็นจริงในปัจจุบัน จึงต้องมีการใช้ Update เพิ่มเข้ามาโดยจะใช้ตัวอย่างของตารางสถานะของฝูงวัวแบบที่ 2 ในการอธิบายดังนี้

ตาราง 2.9 สถานะของฝูงวัวแบบที่ 2 [ที่มา:Managing Temporal Data A Five-Part Series]

LOT_ID_NUM	GNDR_CODE	FROM_DATE	TO_DATE
101	C	1998-01-01	1998-03-23
101	S	1998-03-23	9999-12-31
234	C	1998-02-17	1998-10-17
234	S	1998-10-17	9999-12-31
799	S	1998-03-12	9999-12-31

จากตาราง 2.9 สมมติว่าวันนี้เป็นวันที่ 1998-07-29 และต้องการ Delete ฝูง 234 จะต้องแก้ไขให้ข้อมูลสิ้นสุดที่วันที่ 1998-07-29 และอนาคตต้องไม่มีข้อมูลนั้นอยู่แล้วได้ดังนี้

ตัวอย่าง คำสั่งSQL ที่ใช้ในการสอบถาม [ที่มา:Managing Temporal Data A Five-Part Series]

```
UPDATE LOT
SET TO_DATE = CURRENT_DATE
WHERE LOT_ID_NUM = 234
AND TO_DATE >= CURRENT_DATE
AND FROM_DATE < CURRENT_DATE

DELETE FROM LOT
WHERE LOT_ID_NUM = 234
AND FROM_DATE > CURRENT_DATE
```

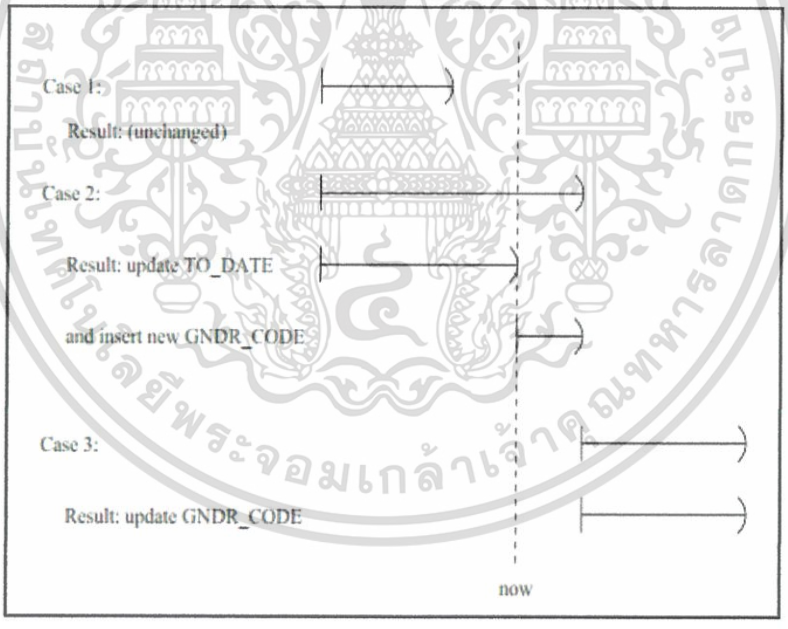
ผลลัพธ์จากการทำ Current Delete ได้ดังนี้

ตาราง 2.10 สถานะของฝูงวัว 2 หลังจากการทำ Current Delete [ที่มา:Managing Temporal Data A Five-Part Series]

LOT_ID_NUM	GNDR_CODE	FROM_DATE	TO_DATE
101	C	1998-01-01	1998-03-23
101	S	1998-03-23	9999-12-31
234	C	1998-02-17	1998-07-29
799	C	1998-03-12	9999-12-31

จะเห็นได้ว่ามีการ Update ฝูง 234 Calf ให้ TO_DATE เป็นวันที่ 1998-07-29(current date) และลบข้อมูล ฝูง 234 Steer ออกไปเพราะเป็นข้อมูลที่บันทึกไว้ล่วงหน้า

3) Current Update คือการ Update ข้อมูลที่เป็นจริงในปัจจุบัน ซึ่งสามารถแบ่งข้อมูลที่จะ update ได้เป็น 3 กรณีดังนี้



รูป 2.2 กรณีต่างๆของการทำ Current Update
(ที่มา: Managing Temporal Data A Five-Part Series)

กรณีที่ 1 ปรับปรุงข้อมูลที่เป็นอดีต

ผลลัพธ์ : จะต้องไม่เกิดการเปลี่ยนแปลงใดๆ

กรณีที่ 2 ปรับปรุงข้อมูลที่ยังเป็นจริงอยู่ในเวลาปัจจุบัน

ผลลัพธ์ : มีการ Update TO_DATE ของข้อมูลเดิมเป็นเวลาปัจจุบัน และ Insert ข้อมูลใหม่ที่มี FROM_DATEเป็นเวลาปัจจุบัน

กรณีที่ 3 ปรับปรุงข้อมูลที่จะเป็นจริงในอนาคต

ผลลัพธ์ : สามารถ Update ได้ทันที

การ Update จึงจะต้องมีการตรวจสอบทั้ง 3 กรณี ตัวอย่างเช่น หากต้องการ Update ข้อมูลฝูง 709 ให้มี GNDR_CODE เป็น "s" จะต้องใช้คำสั่ง SQL ดังนี้

```
INSERT INTO LOT
SELECT LOT_ID_NUM, 's', CURRENT_DATE, TO_DATE

FROM LOT
WHERE LOT_ID_NUM = 799
AND FROM_DATE <= CURRENT_DATE
AND TO_DATE > CURRENT_DATE
UPDATE LOT
SET TO_DATE = CURRENT_DATE
WHERE LOT_ID_NUM = 799
AND GNDR_CODE <> 's'
AND FROM_DATE < CURRENT_DATE
AND TO_DATE > CURRENT_DATE
UPDATE LOT
SET GNDR_CODE = 's'
WHERE LOT_ID_NUM = 799
AND FROM_DATE >= CURRENT_DATE
```

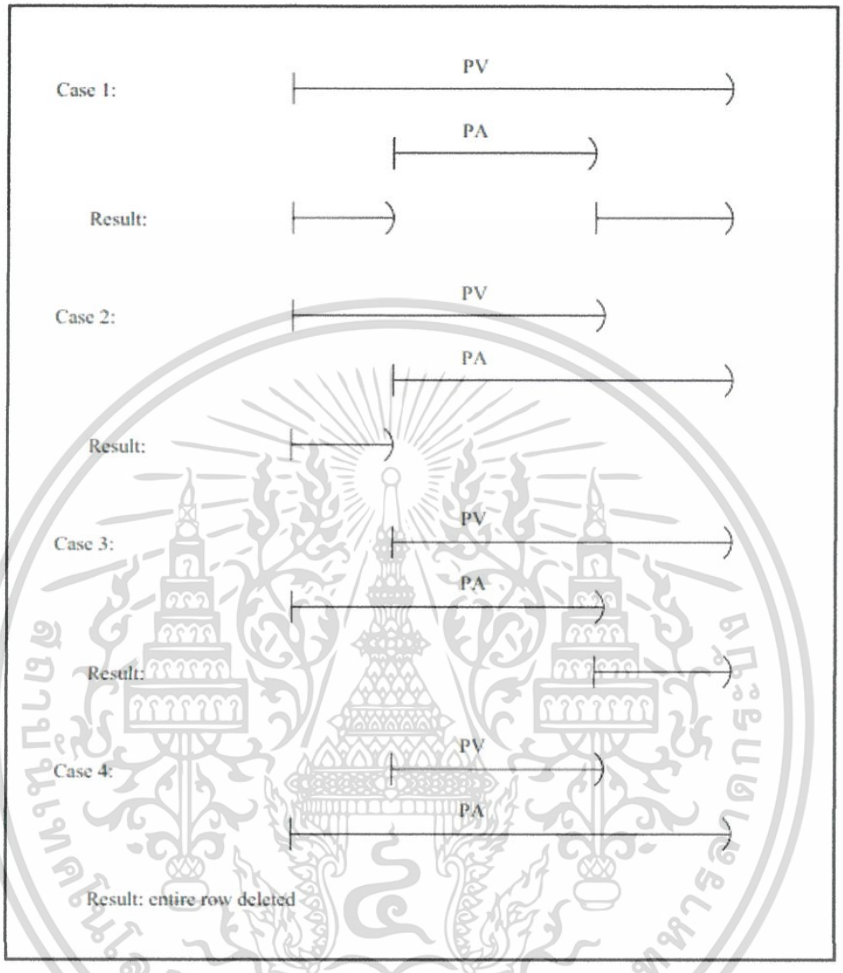
จากทั้งหมดจะเห็นได้ว่าการทำ Current Modifications ไม่มีการยุ่งเกี่ยวกับข้อมูลในอดีตเลย

2.1.5.2 Sequenced Modifications

1) Sequence Insert คือการ Insert โดยมีการกำหนดช่วงเวลาที่ข้อมูลนั้น valid ตั้งแต่เมื่อไหร่ถึงเมื่อไหร่ ตัวอย่างเช่น

```
INSERT INTO LOT
VALUES (426, 'h', DATE '1998-03-26', DATE '1998-04-14')
```

2) Sequence Delete คือการลบข้อมูลโดยมีการเจาะจงช่วงเวลาที่ต้องการจะลบ โดยการ Delete ลักษณะนี้ทำให้ต้องพิจารณาลักษณะของข้อมูลทำก่อนลบ ซึ่งแบ่งได้เป็น 4 กรณี



รูป 2.3 กรณีของการทำ Sequence delete แบบต่างๆ
(ที่มา: *Managing Temporal Data A Five-Part Series*)

กำหนดให้ PV คือ ช่วงเวลาที่ Fact เป็นจริงอยู่ในฐานข้อมูล

PA คือ ช่วงเวลาที่ต้องการปฏิบัติการ

กรณีที่ 1 PV ครอบคลุม PA

ผลของการลบข้อมูลจะเกิดการ update ข้อมูลเดิม โดยให้ TO_DATE เท่ากับ FROM_DATE ของ PA และinsert ข้อมูลอีก 1 แถว โดยให้ FROM_DATE เท่ากับ TO_DATE ของ PA และ TO_DATE เท่ากับ TO_DATE ของ PV เดิม

กรณีที่ 2 PA เกิดขึ้นหลัง PV และสิ้นสุดหลัง PV

ผลการลบข้อมูลจะเป็น update ข้อมูลเดิมโดย TO_DATE เท่ากับ FROM_DATE ของPA

กรณีที่ 3 PA เกิดก่อน PV และสิ้นสุดก่อน PV

ผลการลบข้อมูลจะเป็น update ข้อมูลเดิมโดย FROM_DATE เท่ากับ TO_DATE ของ PA

กรณีที่ 4 PA เกิดขึ้นก่อนและสิ้นสุดหลัง PV

ผลของการลบข้อมูลจะเป็นการลบแถวข้อมูลของ PV ออกจากฐานข้อมูลทั้งหมด

ตัวอย่างเช่น ถ้าต้องการลบฝูง 234 ในช่วงเวลาตั้งแต่วันที่ 1998-10-01 ถึง 1998-10-22 จะต้องทำให้ครอบคลุมทั้ง 4 กรณี โดยใช้คำสั่ง SQL ดังนี้ [ที่มา: Managing Temporal Data A Five-Part Series]

กรณีที่ 1

```
INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, DATE '1998-10-22', TO_
DATE
FROM LOT
WHERE LOT ID NUM = 234
AND FROM DATE <= DATE '1998-10-01'
AND TO DATE > DATE '1998-10-22'
```

กรณีที่ 2

```
UPDATE LOT
SET TO_DATE = DATE '1998-10-01'
WHERE LOT_ID_NUM = 234
AND FROM_DATE < DATE '1998-10-01'
AND TO_DATE >= DATE '1998-10-01'
```

กรณีที่ 3

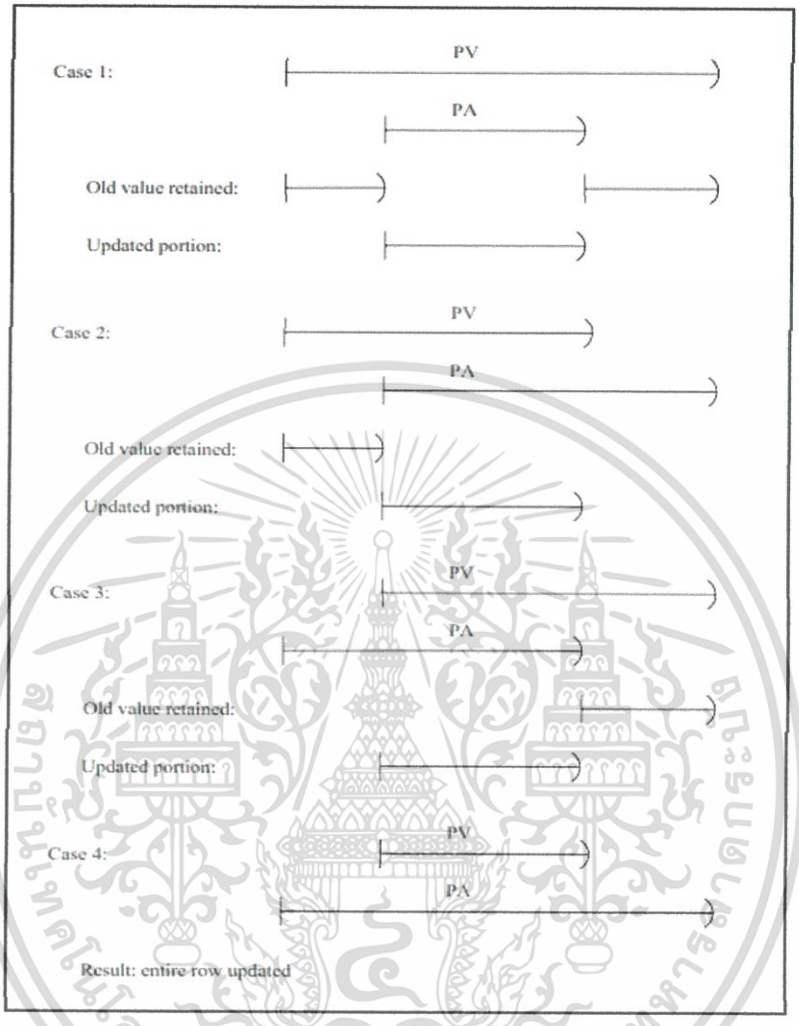
```
UPDATE LOT
SET FROM_DATE = DATE '1998-10-22'
WHERE LOT_ID_NUM = 234
AND FROM_DATE < DATE '1998-10-22'
AND TO_DATE >= DATE '1998-10-22'
```

กรณีที่ 4

```
DELETE FROM LOT
WHERE LOT_ID_NUM = 234
AND FROM_DATE >= DATE '1998-10-01'
AND TO_DATE <= DATE '1998-10-22'
```

จะเห็นว่าการทำ Sequence Delete จะต้องเขียนคำสั่ง SQL ถึง 4 คำสั่ง เพื่อให้เกิดการ DELETE ที่ถูกต้อง ซึ่งมีความซับซ้อนอย่างมากเมื่อเทียบกับการ DELETE ข้อมูลในระบบฐานข้อมูลปกติ

3) Sequence Update คือการ Update ข้อมูลเฉพาะที่เป็นจริงในช่วงเวลาที่กำหนด ซึ่งแบ่งได้เป็น 4 กรณี ดังนี้



รูป 2.4 กรณีของการทำ Sequence Update แบบต่างๆ

(ที่มา: Managing Temporal Data A Five-Part Series)

กรณีที่ 1 PV ครอบคลุม PA

ผลของการ Update จะเป็นการ Insert 2 แถวและ Update 1 แถว ดังนี้

Insert ข้อมูลแถวแรกมีลักษณะเหมือนข้อมูลเดิมแต่ให้ FROM_DATE มีค่าเท่ากับ FROM_DATE ของ PV และ TO_DATE มีค่าเท่ากับ FROM_DATE ของ PA

Insert ข้อมูลแถวที่สองมีลักษณะเหมือนข้อมูลเดิมแต่ให้ FROM_DATE มีค่าเท่ากับ TO_DATE ของ PA และ TO_DATE มีค่าเท่ากับ TO_DATE ของ PV

Update PV ให้ FROM_DATE ของ PV มีค่าเท่ากับ FROM_DATE ของ PA และ TO_DATE มีค่าเท่ากับ TO_DATE ของ PA และค่าของข้อมูลเท่ากับ PA

กรณีที่ 2 PA ครอบคลุมส่วนหลังของ PV

ผลของการ Update จะเป็นการ Insert 1แถว และ Update ข้อมูล 1แถว ดังนี้

Insert 1แถวโดยมีค่าข้อมูลเท่ากับ PV แต่ให้ FROM_DATE มีค่าเท่ากับ FROM_DATE ของ PV และ TO_DATE มีค่าเท่ากับ FROM_DATE ของ PA

Update PV ให้ FROM_DATE ของ PV มีค่าเท่ากับ FROM_DATE ของ PA และ TO_DATE มีค่าเท่ากับ TO_DATE ของ PA และค่าของข้อมูลเท่ากับ PA

กรณีที่ 3 PA ครอบคลุมครึ่งหน้าของ PV

ผลของการ Update จะเป็นการ Insert 1แถว และ Update ข้อมูล 1แถว ดังนี้

Insert 1 แถวโดยมีค่าข้อมูลเท่ากับ PA และให้ FROM_DATE มีค่าเท่ากับ FROM_DATE ของ PV และ TO_DATE มีค่าเท่ากับ TO_DATE ของ PA

Update PV ให้ FROM_DATE ของ PV มีค่าเท่ากับ TO_DATE ของ PA

กรณีที่ 4 PAครอบคลุม PV ทั้งหมด

ผลของการ Update จะเป็นการ Updete ข้อมูล PV ให้เท่ากับ PA แต่ FROM_DATE และ TO_DATE คงเดิม

ตัวอย่าง เช่นต้องการ Update ข้อมูลวัวฝูง 799 ให้มีสถานะเป็น "Steer" ในช่วงวันที่ 1998-03-01 ถึง 1998-04-01 สามารถใช้คำสั่ง SQL ได้ดังนี้ [ที่มา: Managing Temporal Data A Five-Part Series]

```
INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, FROM_DATE, DATE '1998-03-
01'
FROM LOT
WHERE LOT_ID_NUM = 799
AND FROM_DATE < DATE '1998-03-01'
AND TO_DATE > DATE '1998-03-01'

INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, DATE '1998-04-01',
TO_DATE
FROM LOT
WHERE LOT_ID_NUM = 799
AND FROM_DATE < DATE '1998-04-01'
AND TO_DATE > DATE '1998-04-01'

UPDATE LOT
SET GNDR_CODE = 's'
WHERE LOT_ID_NUM = 799
AND FROM_DATE < DATE '1998-04-01'
AND TO_DATE > DATE '1998-03-01'

UPDATE LOT
SET FROM_DATE = DATE '1998-03-01'
WHERE LOT_ID_NUM = 799
AND FROM_DATE < DATE '1998-03-01'
AND TO_DATE > DATE '1998-03-01'

UPDATE LOT
SET TO_DATE = DATE '1998-04-01'
WHERE LOT_ID_NUM = 799
AND FROM_DATE < DATE '1998-04-01'
AND TO_DATE > DATE '1998-04-01'
```

จะเห็นได้ว่าการทำ Sequence Update ต้องเขียนคำสั่ง SQLถึง 5 คำสั่ง เพื่อให้เกิดการ Update ข้อมูลที่ถูกต้อง ซึ่งแต่ละคำสั่งจะต้องมีการตรวจสอบ FROM_DATE และ TO_DATE ให้ถูกต้อง

2.1.5.3 Non-Sequenced Modifications

Non-Sequenced Modifications คือการ Modifications ที่กระทำกับข้อมูลที่ตรงกับเงื่อนไขเวลาที่กำหนด แต่เงื่อนไขเวลานั้นจะต้องไม่เป็นการเจาะจงเวลาเริ่ม หรือ สิ้นสุด *ตัวอย่างเช่น ต้องการลบข้อมูลฝูง 234 แถวที่มีช่วงระยะเวลานานเกินกว่า 3 เดือน ใช้คำสั่ง SQL ได้ดังนี้* [ที่มา: Managing Temporal Data A Five-Part Series]

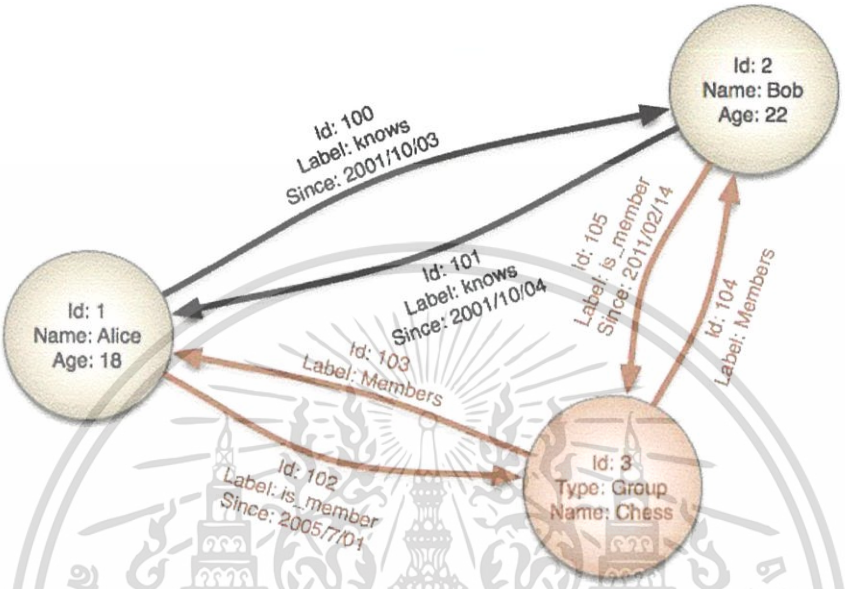
```
DELETE FROM LOT
WHERE LOT_ID_NUM = 234
AND (TO_DATE - FROM_DATE MONTH) > INTERVAL '3' MONTH
```

Non-Sequenced Modifications นั้นใช้คำสั่ง SQL เรียบง่ายไม่ยุ่งยากเท่า Current Modifications และ Sequenced Modifications แต่ Non-Sequenced Modifications จะมีโอกาสใช้จริงได้ยากกว่า

จากที่กล่าวมาข้างต้นจะเห็นได้ว่า การ Query และ Modifications บนระบบฐานข้อมูลเชิงเวลาจะมีความยุ่งยากกกว่าบนระบบฐานข้อมูลปกติมาก แต่ก็แลกมากับความสามารถในการตอบคำถามที่ระบบฐานข้อมูลปกติไม่สามารถตอบได้ด้วยเช่นกัน

2.2 Graph Database

คือระบบฐานข้อมูล ที่เก็บข้อมูลในรูปแบบโครงสร้างของกราฟ จะใช้ส่วนประกอบ 3 ส่วนในการนำเสนอข้อมูลดังรูปได้แก่



รูป 2.5 ระบบฐานข้อมูลรูปแบบกราฟ

(ที่มา: Neo4j-manual-2.1.7)

1) โหนด (Node) ทำหน้าที่นำเสนอเอนทิตี (Entity) เช่น คน สัตว์ สิ่งของ หรือ สิ่งที่ต้องการติดตามข้อมูล

2) สมบัติ (Property) คือข้อมูลที่มีความเกี่ยวข้องกับโหนด เช่น โหนด คือ Alice สมบัติของโหนดนี้อาจจะเป็นอายุของ Alice เป็นต้น

3) เส้นเชื่อม (Edge) คือเส้นเชื่อมระหว่างโนด 2 โนดทำหน้าที่นำเสนอความสัมพันธ์ (Relationship) ระหว่างทั้ง 2 โนด ข้อมูลที่สำคัญของระบบฐานข้อมูลรูปแบบกราฟมักจะอยู่บนส่วนนี้

2.2.1 Neo4j

Neo4j คือระบบฐานข้อมูลรูปแบบกราฟที่ไม่ใช้ภาษาเอสคิวแอล (NoSQL) แบรนด์หนึ่ง จุดเด่นของ Neo4j มีดังต่อไปนี้

1) มีคุณสมบัติ ACID

2) การเพิ่มข้อมูลสามารถทำได้ง่ายผ่านการเพิ่ม Node และ Relationships

3) ค้นหาข้อมูลได้รวดเร็วโดยใช้วิธีการ "Traversals"

4) มีภาษาเฉพาะ (Cypher) ทำให้ง่ายต่อการใช้งานระบบฐานข้อมูลรูปแบบกราฟ

2.2.1.1 ACID properties ของ Neo4j

1) Atomicity คือถ้ามีส่วนไหนของ Transactions ที่ล้มเหลฐานข้อมูลจะไม่มีการเปลี่ยนแปลง

2) Consistency คือ Transactions ใดๆ จะมีผลกับฐานข้อมูลทันที

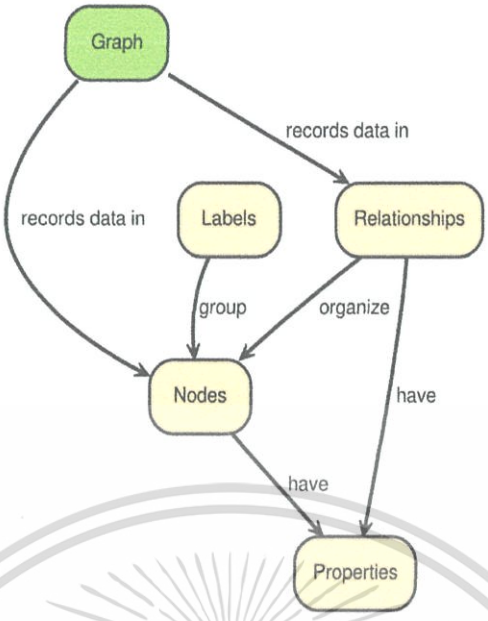
3) Isolation คือระหว่าง Transaction ข้อมูลที่เกี่ยวข้อง (มีการ Modified) จะไม่สามารถเข้าถึงได้จาก Transaction อื่น(Isolation levelขั้น Read – Committed)

4) Durability คือDBMS สามารถกู้คืนผลลัพธ์จาก Committed Transaction ได้ตลอด

2.2.1.2 รูปแบบโครงสร้างของระบบฐานข้อมูล Neo4j

1) เป็นระบบฐานข้อมูลที่เก็บข้อมูลในรูปแบบของโหนด และ Relationships

"A Graph —records data in⟶ Nodes —which have⟶ Properties"



รูป 2.6 กราฟอธิบายความสัมพันธ์ระหว่างกราฟ โหนด และ Relationships

(ที่มา: Neo4j-manual-2.1.7)

กราฟที่ง่ายที่สุด คือกราฟที่มีเพียงแค่โหนดเดียว และ มี Property เป็นชื่อของโหนด

2) Relationships เป็นตัวจัดระเบียบกราฟ

"Nodes —are organized by⟶ Relationships —which also have⟶ Properties"

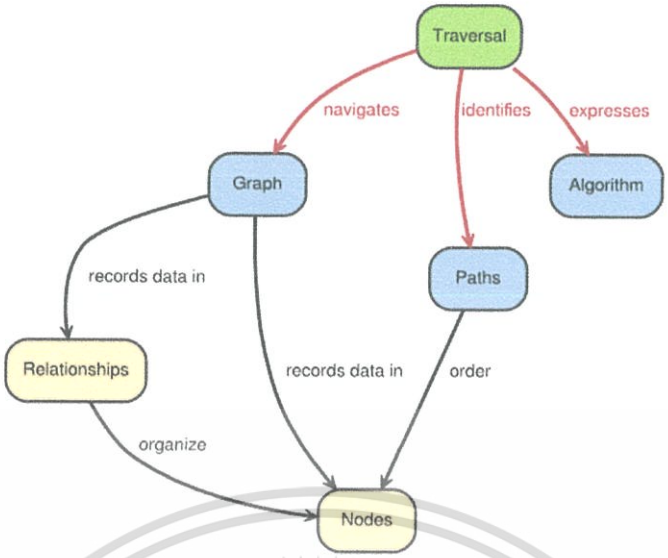
3) Label ทำหน้าที่จับกลุ่มโหนดแยกเป็นเซต (Set)

"Nodes —are grouped by⟶ Labels —into⟶ Sets"

Label นั้นยังสามารถเอาไปใช้เป็นเงื่อนไขในการใช้คำสั่ง Query ได้อีกด้วย

4) การ Query กราฟจะทำโดยใช้วิธี Traversal

"A Traversal —navigates⟶ a Graph; it —identifies⟶ Paths —which order⟶ Nodes"

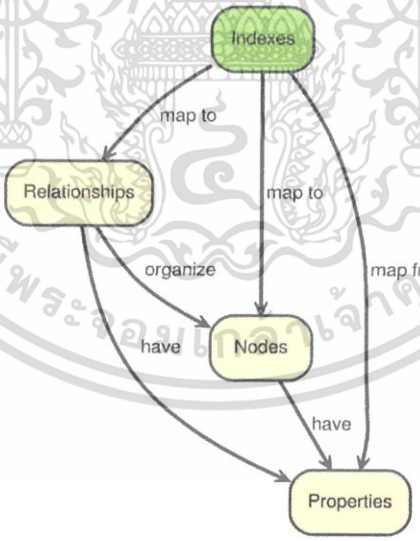


รูป 2.7 กราฟอธิบายความสัมพันธ์ระหว่าง Traversal กับ กราฟ

(ที่มา: Neo4j-manual-2.1.7)

5) Index ใช้สำหรับการ Look-up เพื่อหาโหนดหรือ Relationships

"An Index —maps from⟶ Properties —to either⟶ Nodes or Relationships"

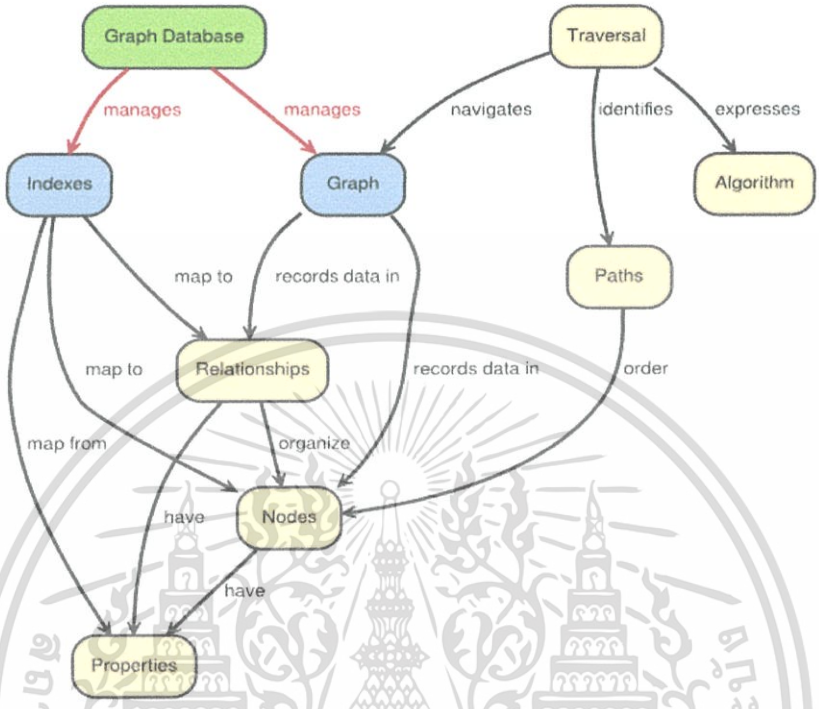


รูป 2.8 กราฟอธิบายความสัมพันธ์ระหว่าง Index โหนด Relationship และ Property

(ที่มา: Neo4j-manual-2.1.7)

6) Neo4j เป็นระบบฐานข้อมูลรูปแบบกราฟ

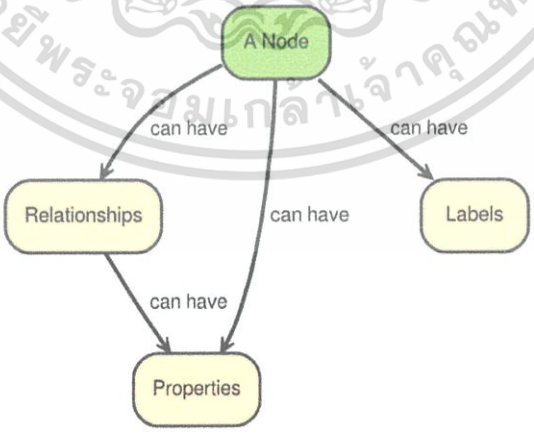
"A Graph Database —manages a⟶ Graph and —also manages related⟶ Indexes"



รูป 2.9กราฟอธิบายภาพรวมโครงสร้างของ Neo4j (ที่มา: Neo4j-manual-2.1.7)

2.2.1.3 ส่วนประกอบของ Neo4j

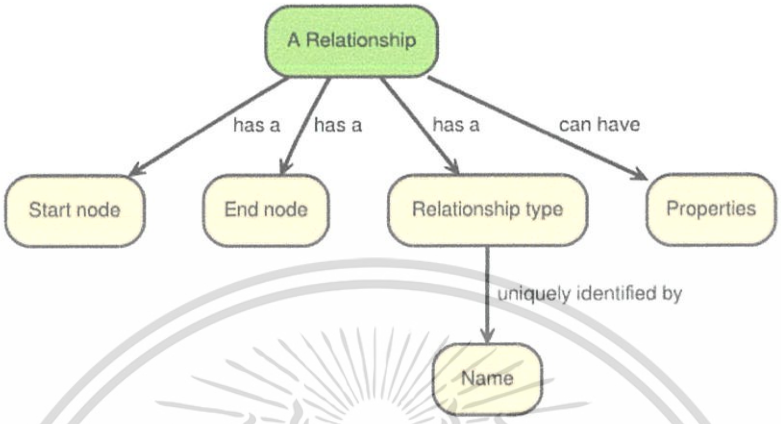
1) โหนด (node)



รูป 2.10 กราฟอธิบายคุณสมบัติของโหนด (ที่มา: Neo4j-manual-2.1.7)

โหนด(node) ส่วนพื้นฐานในการสร้างระบบฐานข้อมูลแบบกราฟประกอบด้วย โหนดและ Relationships ทั้ง 2 อย่างนี้สามารถมี Property ของตนเองได้

2) Relationship

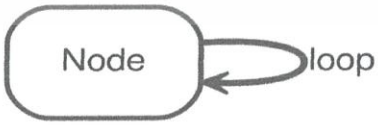


รูป 2.11 กราฟอธิบายคุณสมบัติของ Relationship (ที่มา: Neo4j-manual-2.1.7)

Relationship จะทำหน้าที่เชื่อมระหว่าง 2 โหนดเข้าด้วยกัน โดยโหนดหนึ่งเป็นโหนดเริ่มต้น (Start Node) และ โหนดหนึ่งเป็นโหนดสิ้นสุด (End Node) โดยดูได้จากเครื่องหมายหัวลูกศรดังนี้

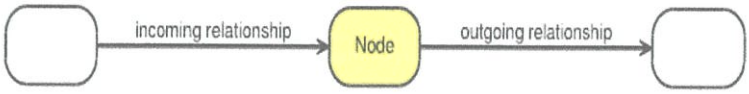
รูป 2.12 ลักษณะของ Relationship (ที่มา: Neo4j-manual-2.1.7)

หรือหากโหนดเริ่มต้นและโหนดสิ้นสุดเป็นโหนดเดียวกันจะได้กราฟดังนี้



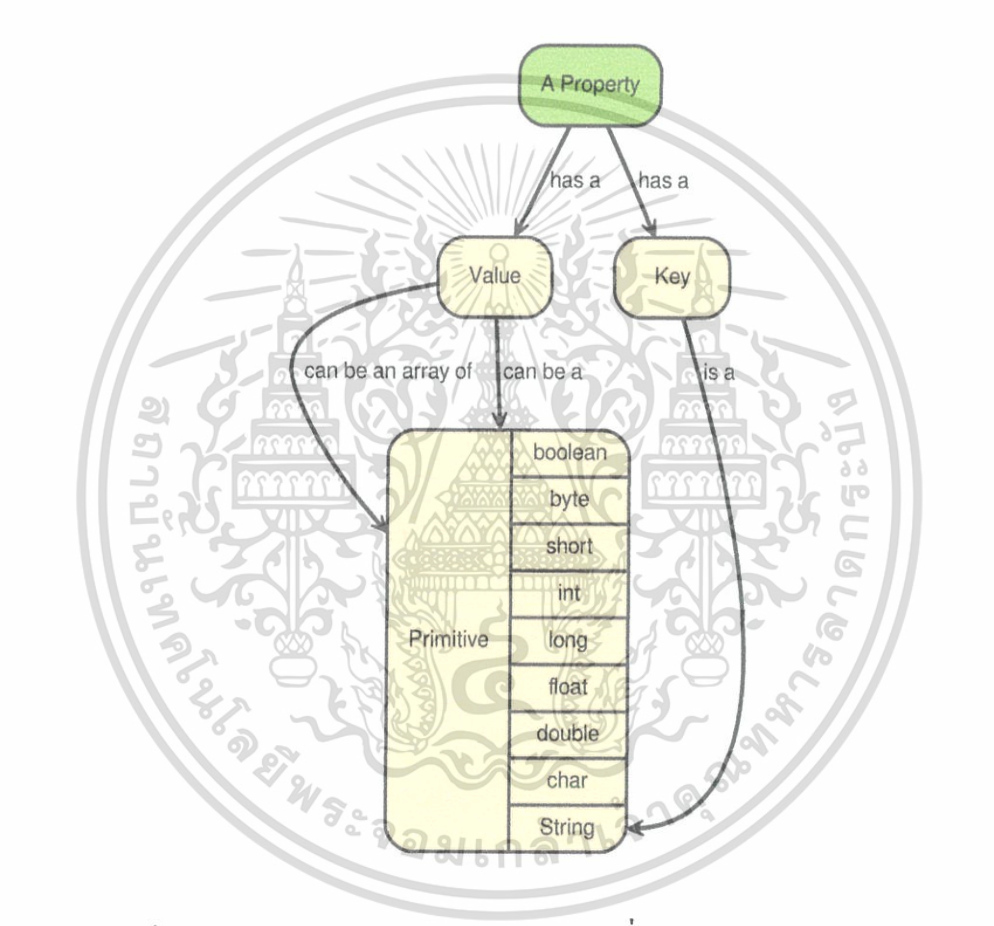
รูป 2.13 Relationship ที่เชื่อม Node เดียว (ที่มา: Neo4j-manual-2.1.7)

หากใช้โหนดเป็นหลัก relationship จะแบ่งได้เป็น 2ประเภทคือ relationship ขาเข้า และ relationship ขาออกดังนี้



รูป 2.13 Relationship ขาเข้า และขาออก (ที่มา: Neo4j-manual-2.1.7)

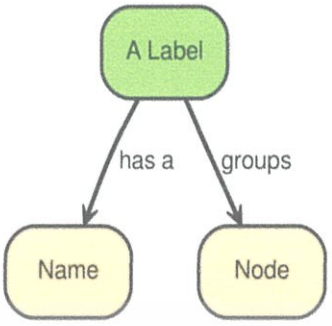
3) Property



รูป 2.14 กราฟอธิบายคุณสมบัติของ Property (ที่มา: Neo4j-manual-2.1.7)

Property ทั้งโหนด และ *Relationship* สามารถมี *Property* ได้ *Property* ประกอบด้วย 2 ส่วนคือ Value และ Key ที่ผูกกัน โดย Key จะต้องเป็น String และ Value ใช้ Primitive

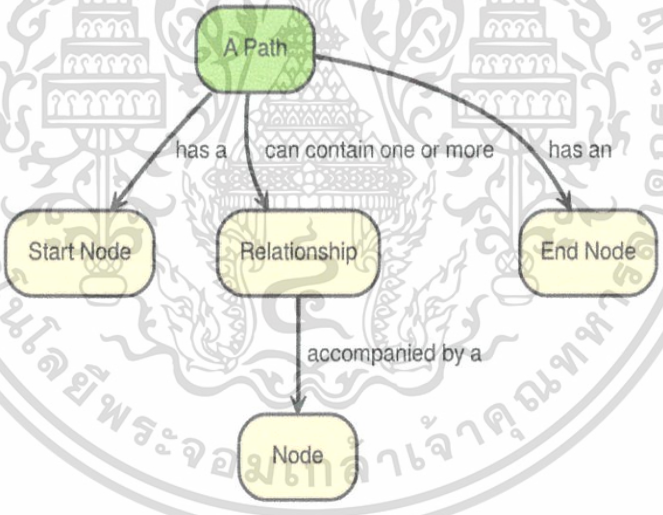
4) Label



รูป 2.15 กราฟอธิบายคุณสมบัติของ Label (ที่มา: Neo4j-manual-2.1.7)

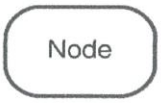
Label เป็นชื่อที่ใช้ในการกำหนดให้โหนด โดยโหนดที่มี Label เหมือนกันจะถือว่าอยู่ใน set เดียวกัน

5) Path

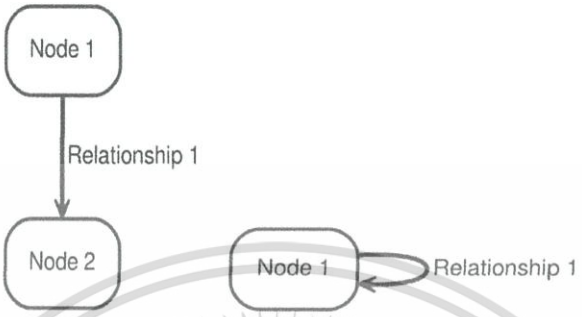


รูป 2.16 กราฟอธิบายคุณสมบัติของ Path (ที่มา: Neo4j-manual-2.1.7)

Path คือ โหนดอย่างน้อย 1 โหนดที่เชื่อมต่อกับ Relationships ส่วนประกอบของ path มีดังนี้



รูป 2.17 ตัวอย่าง Path ความยาว 0 (ที่มา: Neo4j-manual-2.1.7)



รูป 2.18 ตัวอย่าง Path ที่มีความยาวเป็น 1 (ที่มา: Neo4j-manual-2.1.7)

2.2.1.2 ไซเฟอร์ (Cypher)

ไซเฟอร์เป็นภาษาที่ Neo4j ใช้ในการ query ถูกดัดแปลงมาจากภาษา SQL ให้เหมาะสมกับการใช้งานบนระบบฐานข้อมูลประเภทกราฟ ด้วยเหตุนี้เองทำให้ผู้ที่มีความรู้ในภาษา SQL จะสามารถใช้งาน ภาษาไซเฟอร์ได้ไม่ยาก

คำสั่งต่างๆของภาษาไซเฟอร์ จะให้ผลคล้ายๆกับคำสั่งในภาษา SQL ตัวอย่างการเปรียบเทียบระหว่างภาษาไซเฟอร์ กับภาษา SQL มีดังนี้

1) คำสั่ง Start

ใน SQL ผลลัพธ์ที่ต้องการหา นั้นจะใช้ SELECT ในการบอกไปยังที่ซึ่งจะได้มาซึ่งผลลัพธ์ ส่วนใน Cypher นั้นจะ ใช้ START ในการกำหนดจุดเริ่มต้นของกราฟที่ต้องการค้นหาในการนำไปประมวลผล

ตัวอย่าง การใช้ SELECT ใน SQL [ที่มา: Neo4j-manual-2.1.7]

```
SELECT *
FROM "Person"
WHERE name = 'Anakin' ;
```

ตัวอย่าง การใช้ Start ใน Cypher

```
START person=node: Person (name = 'Anakin')
RETURN person ;
```

2) สั่ง Match

คำสั่ง Match ใช้ระบุในการค้นหาของระหว่าง Sub-graph มีลักษณะคล้ายกับ Join ใน SQL เช่น a ⟶b relationship คือ Inner join ระหว่าง node a และ b ตัวอย่างเช่น ต้องการค้นหา *email addresses* ของ คนที่ชื่อ"*Anakin*" ซึ่งมีความสัมพันธ์ แบบ One to Many

ตัวอย่าง การใช้ Join ใน SQL

```
START person=node:Person(name = 'Anakin')
MATCH person-[:email]->email
RETURN email
```

ตัวอย่าง การใช้ Match ใน Cypher

```
START person=node:Person(name = 'Anakin')
MATCH person-[:email]->email
RETURN email
```

3) คำสั่ง Where

คำสั่ง Where สำหรับ Cypher นั้นใช้เพื่อกรองผลที่ต้องการออกมา โดยหลักการในการใช้จะเหมือนกับใน SQL

ตัวอย่าง การใช้ Where ใน SQL

```
SELECT *
FROM "Person"
WHERE "Person".age > 35 AND "Person".hair = 'blonde'
```

ตัวอย่าง การใช้ Where ใน Cypher

```
START person=node:Person('name: *')
WHERE person.age > 35 AND person.hair = 'blonde'
RETURN person
```

4) คำสั่ง Return

ลักษณะการทำงานของ Return ใน Cypher จะคล้ายกับ Select ใน SQL แต่จะมีจุดที่แตกต่างกันคือ ค่าที่อยู่หลัง Return นั้นจะถูกนำมาใช้ในการจัดกลุ่มในลักษณะ คล้ายกับคำสั่ง Group by ของ SQL ด้วย

ตัวอย่าง การใช้ Select และ Group by ใน SQL

```
SELECT  "Person".name, count(*)
   FROM  "Person"
   GROUP BY  "Person".name
   ORDER BY  "Person".name
```

ตัวอย่าง การใช้ Return ใน Cypher

```
START  person = n START  person = node:Person('name:
*')
   RETURN person.name, count(*)
   ORDER  BY  person.name  SELECT     "Person".name,
count(*)
  ode:Person('name: *')
  RETURN person.name, count(*)
  ORDER BY person.name SELECT  "Person".name, count(*)
```

2.3 อุปกรณ์ที่ใช้ในการ แปลงภาษา

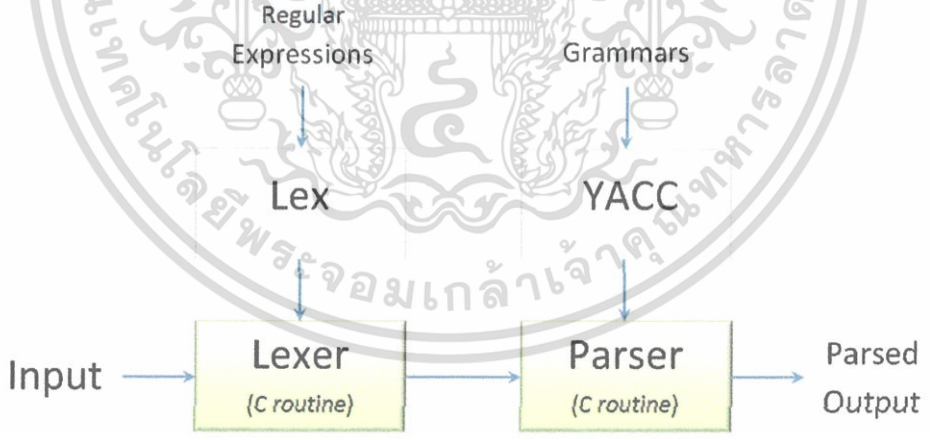
2.3.1 LEX

A Lexical Analyzer Generator (LEX) เป็นโปรแกรมสำหรับสร้างโค้ดภาษา C ที่ซึ่งทำหน้าที่เป็น Scanner หรือ Lexical analyzer ในคอมไพเลอร์ Scanner นี้จะทำหน้าที่เป็นตัวเปลี่ยน Input ทั้งหมดที่เป็นสตริงให้กลายเป็น Token หรือหน่วยพื้นฐานที่สุดสำหรับที่เราสนใจและจะนำไปตรวจสอบไวยากรณ์หรือ Patternอีกทีที่ตัวParser

ตัวอย่างเช่น สตริงที่รับเข้ามาเป็น 512 + 456 ซึ่งเป็นตัวอักษร 7 ตัวด้วยกัน แต่สำหรับที่เราสนใจแล้ว 512 นั้นเป็น Token เดียวกันนั่นคือเป็นตัวเลขการแปลงสตริงทั้งหมดเป็น Token คือหน้าที่ของ lexer ที่สร้างขึ้นโดย LEX ปัจจุบัน ได้มีการนำ LEX ไปพัฒนา จนเกิดเป็น LEX ที่รองรับภาษาอื่นได้ เช่น JFLEX ที่รองรับภาษา JAVA เป็นต้น

2.3.2 YACC

Yet Another Compiler Compile (YACC) เป็นโปรแกรมสำหรับสร้างโค้ดภาษา C ที่ทำหน้าที่เป็น Parser หรือ Syntax Analyzer ที่เป็นตัวตรวจสอบไวยากรณ์ของ Token ทั้งหมดว่าเป็นไปตามรูปแบบที่ถูกต้องหรือไม่ เช่น expression, statement, declaration, block, procedure เป็นต้น ซึ่งรายการของกฎต่าง ๆ ที่ทำให้ Parser เข้าใจความสัมพันธ์เหล่านี้เรียกว่า Grammar



รูป 2.19 หลักการทำงาน ของ LEX และ YACC

(ที่มา: http://parasitebass.blogspot.com/2012/06/in-complete-guide-for-lex-yacc.html)

บทที่ 3

การออกแบบ และพัฒนา

3.1 หลักการและเหตุผล

1) จากการ Query และ Modification บนระบบฐานข้อมูลเชิงเวลามีความซับซ้อน เป็นเหตุให้ใช้งานยาก ทั้งๆ ที่ระบบฐานข้อมูลเชิงเวลามีประสิทธิภาพในการตอบโจทย์ที่ ระบบฐานข้อมูลปกติไม่สามารถตอบได้ จึงเกิดแนวคิดในการหาวิธีให้ใช้งานได้ง่ายขึ้น โดยให้ผู้ใช้งาน Query และ Modification โดยใช้คำสั่งปกติ แล้วให้ application เป็นตัวจัดการแปลงคำสั่งเหล่านั้นให้ใช้บนระบบฐานข้อมูลเชิงเวลาได้

2) ระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ในปัจจุบันยังไม่มีการรองรับการใช้งานระบบฐานข้อมูลเชิงเวลา จึงจำเป็นต้องทดสอบว่า ระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล สามารถ ใช้งานเป็นระบบฐานข้อมูลเชิงเวลา ได้หรือไม่ และมีข้อแตกต่างกับ ระบบฐานข้อมูลเชิงเวลา ที่อยู่บนระบบฐานข้อมูลที่ใช้ภาษาเอสคิวแอล อย่างไร

3) เหตุผลที่เลือกใช้ Neo4j เพราะเป็นระบบฐานข้อมูลรูปแบบกราฟ ที่แปลกใหม่และใช้ภาษาไซเฟอร์ที่มีพื้นฐานมาจากภาษาเอสคิวแอล ทำให้ง่ายต่อการเรียนรู้และเปรียบเทียบ

3.2 การออกแบบ

จากหลักการและเหตุผลที่กล่าวข้างต้นการออกแบบโปรแกรมประยุกต์จึงแบ่งออกเป็น 2 ส่วน ดังนี้

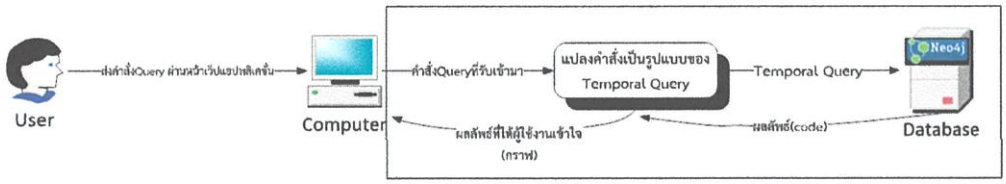
3.2.1ส่วนระบบฐานข้อมูลเชิงเวลา

คือการออกแบบการเก็บข้อมูลบนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ให้อยู่ในรูปแบบระบบฐานข้อมูลเชิงเวลา

3.2.2 ส่วนระบบแปลงคำสั่ง

คือการออกแบบส่วนที่ทำหน้าที่แปลงคำสั่งภาษาไซเฟอร์ ให้สามารถใช้งานกับระบบฐานข้อมูลเชิงเวลาได้

โดยทั้ง 2ส่วนนี้จะรวมกันเป็นโปรแกรมประยุกต์ในลักษณะดังรูป



รูป 3.1 ภาพรวมของโปรแกรมประยุกต์

3.3 การพัฒนา

3.3.1 ส่วนระบบฐานข้อมูลเชิงเวลา

3.3.1.1 ตรวจสอบคุณสมบัติของระบบฐานข้อมูล Neo4j

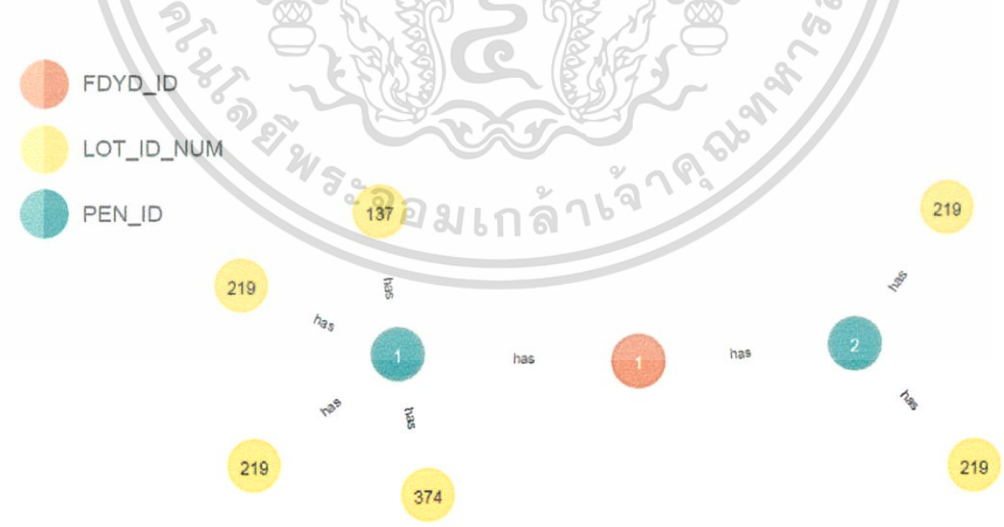
โดยการทดสอบแปลงข้อมูลจากตาราง 2.5 และ 2.10 ซึ่งเป็นตารางเก็บข้อมูลเชิงเวลา ให้อยู่ในรูปแบบกราฟ โดยมีการแปลงดังนี้

แต่ละ Attribute ใน 1 record ที่เป็น primary key แปลงเป็น 1 node และเชื่อมต่อกันด้วย relationship ข้อมูลในส่วนอื่นๆที่เหลือและเวลา แปลงเป็น property ใส่ไว้บน relationship ที่ชี้ไปยัง node ที่เป็นหน่วยย่อยที่สุด ภาษาไซเฟอร์เพื่อบันทึกกราฟลงระบบฐานข้อมูล Neo4j และผลลัพธ์ที่ได้เป็นดังนี้

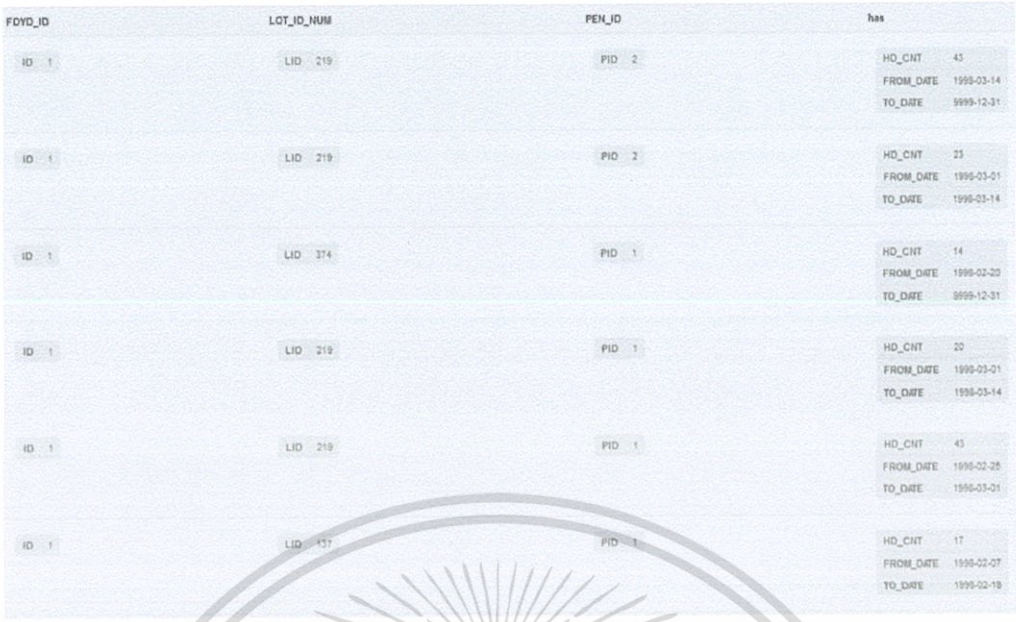
```
CREATE (n:FDYD_ID { ID:"1" });
MATCH (n:FDYD_ID)
WHERE n.ID = "1"
CREATE (m:LOT_ID_NUM { LID:'137' })
CREATE (o:LOT_ID_NUM { LID:'219' })
CREATE (p:LOT_ID_NUM { LID:'374' })
CREATE (n)-[:has]->(m)
CREATE (n)-[:has]->(o)
CREATE (n)-[:has]->(p);
```

```
MATCH (n:LOT_ID_NUM)
where (n.LID = '137')
CREATE (p1:PEN_ID { PID:'1' })
CREATE (p2:PEN_ID { PID:'2' })
CREATE (n)-[:in]->(p1)
CREATE (n)-[:in]->(p2);

MATCH (n:LOT_ID_NUM)
where (n.LID = '219')
CREATE (p1:PEN_ID { PID:'1' })
CREATE (p2:PEN_ID { PID:'2' })
CREATE (n)-[:in]->(p1)
CREATE (n)-[:in]->(p2);

MATCH (n:LOT_ID_NUM)
where (n.LID = '374')
CREATE (p1:PEN_ID { PID:'1' })
CREATE (p2:PEN_ID { PID:'2' })
CREATE (n)-[:in]->(p1)
CREATE (n)-[:in]->(p2);

MATCH (n:LOT_ID_NUM{LID:'137'}),(m:PEN_ID{PID:'1'})
where (n)-->(m)
CREATE (o:HD_CNT {N:'17', FROM_DATE:'1998-02-07',
TO_DATE:'1998-02-18' })
CREATE (m)-[:Number_of_head]->(o);

MATCH (n:LOT_ID_NUM{LID:'219'}),(m:PEN_ID{PID:'1'})
where (n)-->(m)
CREATE (o:HD_CNT {N:'43', FROM_DATE:'1998-02-25',
TO_DATE:'1998-03-01' })
CREATE (m)-[:Number_of_head]->(o);
```

```
MATCH (n:LOT_ID_NUM{LID:'219'}),(m:PEN_ID{PID:'1'})
where (n)-->(m)
CREATE (o:HD_CNT {N:'20', FROM_DATE:'1998-03-01',
TO_DATE:'1998-03-14' })
CREATE (m)-[:Number_of_head]->(o);

MATCH (n:LOT_ID_NUM{LID:'219'}),(m:PEN_ID{PID:'2'})
where (n)-->(m)
CREATE (o:HD_CNT {N:'23', FROM_DATE:'1998-03-01',
TO_DATE:'1998-03-14' })
CREATE (p:HD_CNT {N:'43', FROM_DATE:'1998-03-14',
TO_DATE:'9999-12-31' })
CREATE (m)-[:Number_of_head]->(o)
CREATE (m)-[:Number_of_head]->(p);

MATCH (n:LOT_ID_NUM{LID:'374'}),(m:PEN_ID{PID:'1'})
where (n)-->(m)
CREATE (o:HD_CNT {N:'14', FROM_DATE:'1998-02-20',
TO_DATE:'9999-12-31' })
CREATE (m)-[:Number_of_head]->(o);
```



รูป 3.2 กราฟนำเสนอข้อมูลจากตาราง 2.5



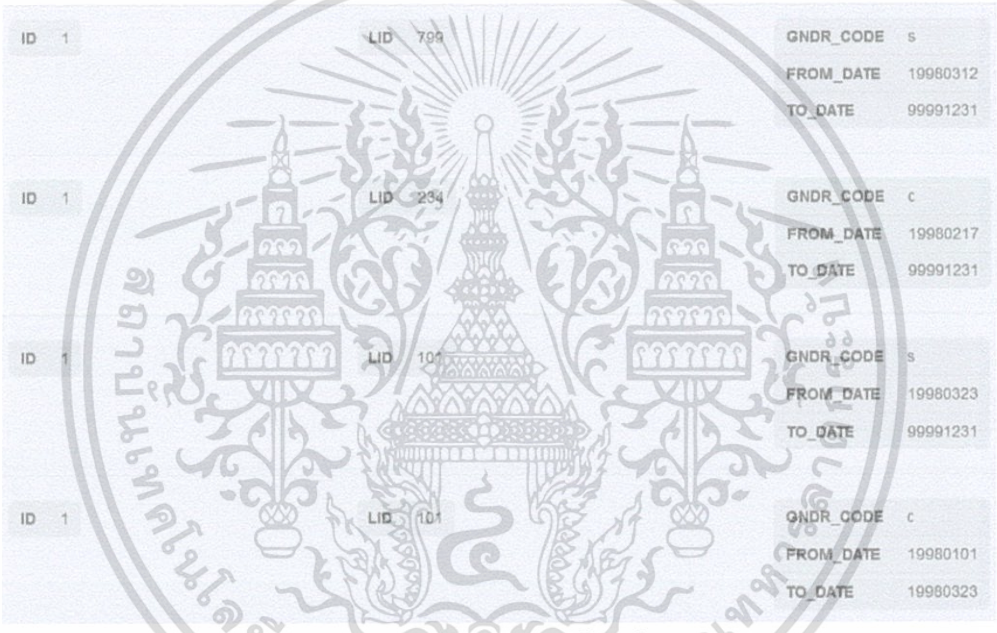
รูป 3.3 กราฟจากรูป 3.1 แสดงผลในรูปแบบตาราง

```
CREATE (n:FDYD_ID { ID:"1" });

MATCH (p:FDYD_ID)
WHERE p.ID = "1"
CREATE (a:LOT_ID_NUM { LID:'101' })
CREATE (b:LOT_ID_NUM { LID:'101' })
CREATE (c:LOT_ID_NUM { LID:'234' })
CREATE (d:LOT_ID_NUM { LID:'799' })
CREATE (p)-[:has{GNDR_CODE:'c', FROM_DATE:'19980101',
TO_DATE:'19980323'}]->(a)
CREATE (p)-[:has{GNDR_CODE:'s', FROM_DATE:'19980323',
TO_DATE:'99991231'}]->(b)
CREATE (p)-[:has{GNDR_CODE:'c', FROM_DATE:'19980217',
TO_DATE:'99991231'}]->(c)
CREATE (p)-[:has{GNDR_CODE:'b', FROM_DATE:'19980312',
TO_DATE:'99991231'}]->(d);
```

```
MATCH (p:FDYD_ID)
WHERE p.ID = "1"
CREATE (a:LOT_ID_NUM { LID:'433' })
CREATE (p)-[:has{GNDR_CODE:'h', FROM_DATE:'19980729',
TO_DATE:'99991231'}]->(a);


MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "101" and has.TO_DATE = '99991231'
SET    has.TO_DATE = '19980729';


MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799"
SET has.GNDR_CODE = 'c';


MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980729' and
has.TO_DATE >= '19980729'
CREATE (L2:LOT_ID_NUM { LID:'799' })
CREATE (a)-[:has{GNDR_CODE:'s', FROM_DATE:'19980729',
TO_DATE:'99991231'}]->(L2)
SET has.TO_DATE = '19980729';
```

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE >= '19980729'
SET has.GNDR_CODE = 's';
MATCH (p:FDYD_ID)
WHERE p.ID = "1"
CREATE (a:LOT_ID_NUM { LID:'426' })
CREATE (p)-[:has{GNDR_CODE:'h', FROM_DATE:'19980326',
TO_DATE:'19980414'}]->(a);
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980301' and
has.TO_DATE > '19980301'
CREATE (L2:LOT_ID_NUM { LID:'799' })
CREATE (a)-[:has{GNDR_CODE:has.GNDR_CODE ,
FROM_DATE:has.FROM_DATE, TO_DATE:'19980301'}]->(L2);
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980401' and
has.TO_DATE > '19980401'
CREATE (L2:LOT_ID_NUM { LID:'799' })
CREATE (a)-[:has{GNDR_CODE:has.GNDR_CODE ,
FROM_DATE:'19980401', TO_DATE:has.TO_DATE}]->(L2);
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980401' and
has.TO_DATE > '19980301'
SET has.GNDR_CODE = 's' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980301' and
has.TO_DATE > '19980301'
SET has.FROM_DATE = '19980301' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE < '19980401' and
has.TO_DATE > '19980401'
SET has.TO_DATE = '19980401' ;
```

รูป 3.4 กราฟนำเสนอข้อมูลจากตาราง 2.10

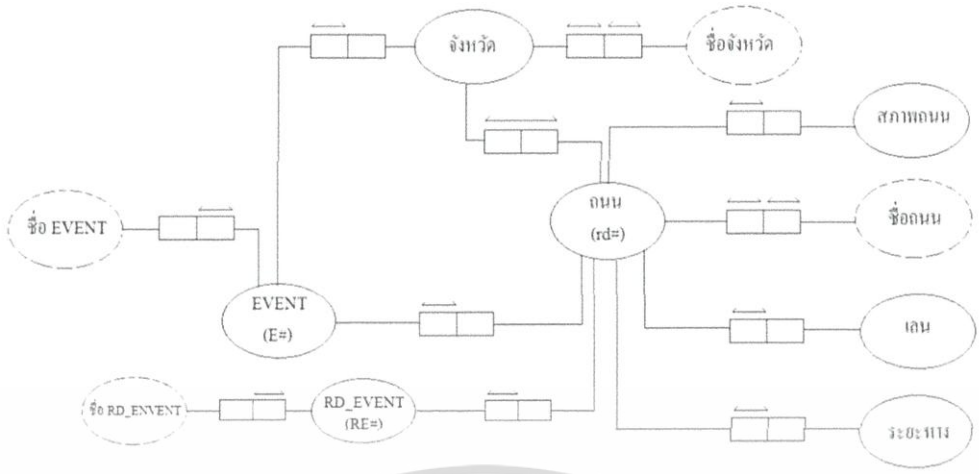


รูป 3.5 กราฟจากรูป 3.3 แสดงผลในรูปแบบตาราง

จากนั้น ทดสอบการทำงานของระบบฐานข้อมูลเชิงเวลาโดยใช้การลองทำ Query รูปแบบต่างๆ ซึ่งพบว่า ระบบฐานข้อมูล Neo4j สามารถใช้เป็นระบบฐานข้อมูลเชิงเวลาได้

3.3.2 เก็บข้อมูลจริงของ เส้นทาง, เทศกาล และอื่นๆ เขียนความสัมพันธ์ในรูป NIAM

เพื่อให้เข้าใจง่ายดังนี้



รูป 3.6 NIAM ของข้อมูลที่จะใช้ทำแผนที่

คำอธิบายรูป 3.6

ถนน คือ ช่วงของถนน มีจุดเริ่มและสิ้นสุด มีรายละเอียดจำนวนเลน สภาพถนน ชื่อถนน และระยะทาง

จังหวัด คือ "จังหวัด" มีชื่อจังหวัดระบุไว้

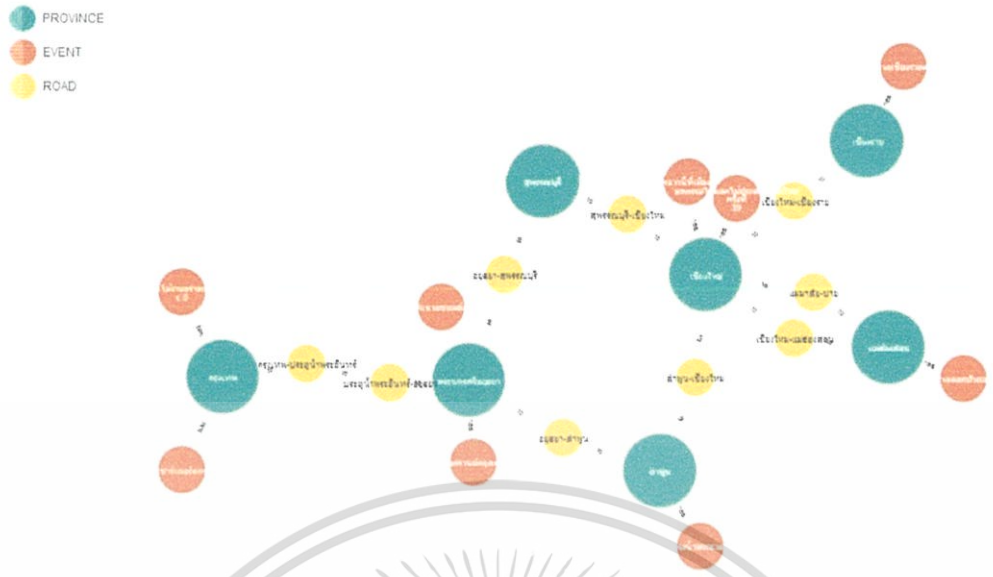
EVENT คือ เหตุการณ์กิจกรรมต่างๆ ที่เกิดขึ้นมี valid time ระบุไว้ เกิดขึ้นได้ทั้งบนจังหวัด และ ถนน

RD EVENT คือ เหตุการณ์ที่เกิดบนถนน เช่น ปิดถนน มี valid time ระบุไว้ เกิดขึ้นได้บนถนนเท่านั้น

ทำการแมพ NIAM เป็นกราฟ โดยใช้หลักการดังนี้

1) ข้อมูลที่เป็น Entity type แปลงเป็น node
2) ข้อมูลที่เป็น Value type แปลงเป็น property ไปเกาะกับ node ที่มีความสัมพันธ์กัน
3) ความสัมพันธ์ระหว่าง Entity แปลงเป็น relationship ระหว่าง node
4) ข้อมูลเวลาของ entity ตัวไหน ให้แปลงเป็น เป็น property ไปเกาะกับ node ที่ได้จาก entity นั้น

ทำตามขั้นตอนจะได้กราฟดังรูป



รูป 3.7 กราฟที่ได้จากการแปลง NIAM

การ Insert ข้อมูลนี้ไปยังระบบฐานข้อมูล Neo4j โดยใช้ภาษาไซเฟอร์ เขียนได้ดังนี้หลังจากนั้น

```
CREATE (n:PROVINCE { NAME:"กรุงเทพ" });
MATCH (n:PROVINCE)
WHERE n.NAME = "กรุงเทพ"
CREATE (p1:ROAD { NAME:"no.347" , LANES:4 ,
CONDITIONS:"good" , DISTANCE:76})
CREATE (p2:ROAD { NAME:"no.340" , LANES:4 ,
CONDITIONS:"good" , DISTANCE:100})
CREATE (n)-[:has]->(p1)
CREATE (n)-[:has]->(p2);

MATCH (n:ROAD)
WHERE n.NAME = "no.347"
CREATE (p1:PROVINCE { NAME:"อยุธยา"})
CREATE (n)-[:has]->(p1);
```

```
MATCH (n:ROAD)
WHERE n.NAME = "no.340"
CREATE (p1:PROVINCE { NAME:"สุพรรณบุรี"})
CREATE (n)-[:has]->(p1);

MATCH (n:PROVINCE)
WHERE n.NAME = "อยุธยา"
CREATE (p1:ROAD { NAME:"no.1" , LANES:4 ,
CONDITIONS:"good" , DISTANCE:623})
CREATE (n)-[:has]->(p1);
MATCH (n:PROVINCE)
WHERE n.NAME = "สุพรรณบุรี"
CREATE (p1:ROAD { NAME:"no.1" , LANES:4 ,
CONDITIONS:"good" , DISTANCE:796})
CREATE (n)-[:has]->(p1);

MATCH (n:ROAD)
WHERE n.NAME = "no.1" and n.DISTANCE = 623
CREATE (p1:PROVINCE { NAME:"เชียงใหม่"})
CREATE (n)-[:has]->(p1);

MATCH (n:ROAD),(n2:PROVINCE)
WHERE n.NAME = "no.1" and n.DISTANCE = 796 and
n2.NAME = "เชียงใหม่"
CREATE (n)-[:has]->(n2);

MATCH (n:PROVINCE)
WHERE n.NAME = "เชียงใหม่"
CREATE (p1:EVENT { ENAME:"สงกรานต์"})
CREATE (n)-[:has{FROM_DATE:'20150411',
TO_DATE:'20150415'}]->(p1);

MATCH (n:PROVINCE)
WHERE n.NAME = "สุพรรณบุรี"
CREATE (p1:EVENT { ENAME:"ตรุษจีนสุพรรณบุรี"})
CREATE (n)-[:has{FROM_DATE:'20150219',
TO_DATE:'20150221'}]->(p1);
```

```
MATCH (n:PROVINCE)
WHERE n.NAME = "อยุธยา"
CREATE (p1:EVENT { ENAME:"สงกรานต์กรุงเก่า"})
CREATE (n)-[:has{FROM_DATE:'20150413',
TO_DATE:'20150415'}]->(p1);

MATCH (n:PROVINCE)
WHERE n.NAME = "อยุธยา"
CREATE (p1:EVENT { ENAME:"วันนายขนมต้ม"})
CREATE (n)-[:has{FROM_DATE:'20150317',
TO_DATE:'20150317'}]->(p1);
```

ทดสอบระบบฐานข้อมูลเชิงเวลาที่ได้โดยการทดสอบ Query ต่างๆ

3.3.2 ระบบแปลงคำสั่ง Query

1) กำหนด input/output ของระบบ โดยให้ input เป็น ภาษาไซเฟอร์ที่ใช้ตามปกติ และให้ output เป็นภาษาไซเฟอร์ที่ใช้งานกับระบบฐานข้อมูลเชิงเวลาได้

2) ออกแบบระบบให้เหมาะสมกับ input/output ดังต่อไปนี้

```
//Current insert (Input)
     MATCH (p:AAA)
     WHERE p.BBB = "CCC"
     CREATE (a:DDD)
     CREATE (p)-[:has]->(a);
//Current insert (Output)
     MATCH (p:AAA)
     WHERE p.BBB = "CCC"
     CREATE (a:DDD)
     CREATE (p)-[:has{FROM_DATE:"current date",
                      TO_DATE:'99991231'}]->(a);
```

```
//Current delete (Input)
     MATCH  (a)-[has:AAA]->(L:BBB)
     WHERE  L.CCC = "DDD"
     DELETE has , L ;

//Current delete (Output)
     MATCH  (a)-[has:AAA]->(L:BBB)
     WHERE  L.CCC = "DDD" and has.TO_DATE = '99991231'
     SET      has.TO_DATE = 'current date';

//Current update (Input)
     MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "DDD" and has.FROM_DATE <= 'current
date'     and has.TO_DATE > 'today'
     SET has.EEE = 'FFF' ;

//Current update (output)
MATCH (a)-[has:AAA]->(L:BBB)
WHERE L.CCC = "DDD" and has.FROM_DATE <= 'current
     date' and has.TO_DATE > 'current date'
CREATE (L2:BBB { CCC:'DDD' })
CREATE (a)-[:has{EEE:'FFF', FROM_DATE:'current date',
TO_DATE:'99991231'}]->(L2)
     SET has.TO_DATE = 'current date'; //case 1,2
     MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "DDD" and has.FROM_DATE >= 'current
date'
     SET has.EEE = 'FFF'; //case 3
```

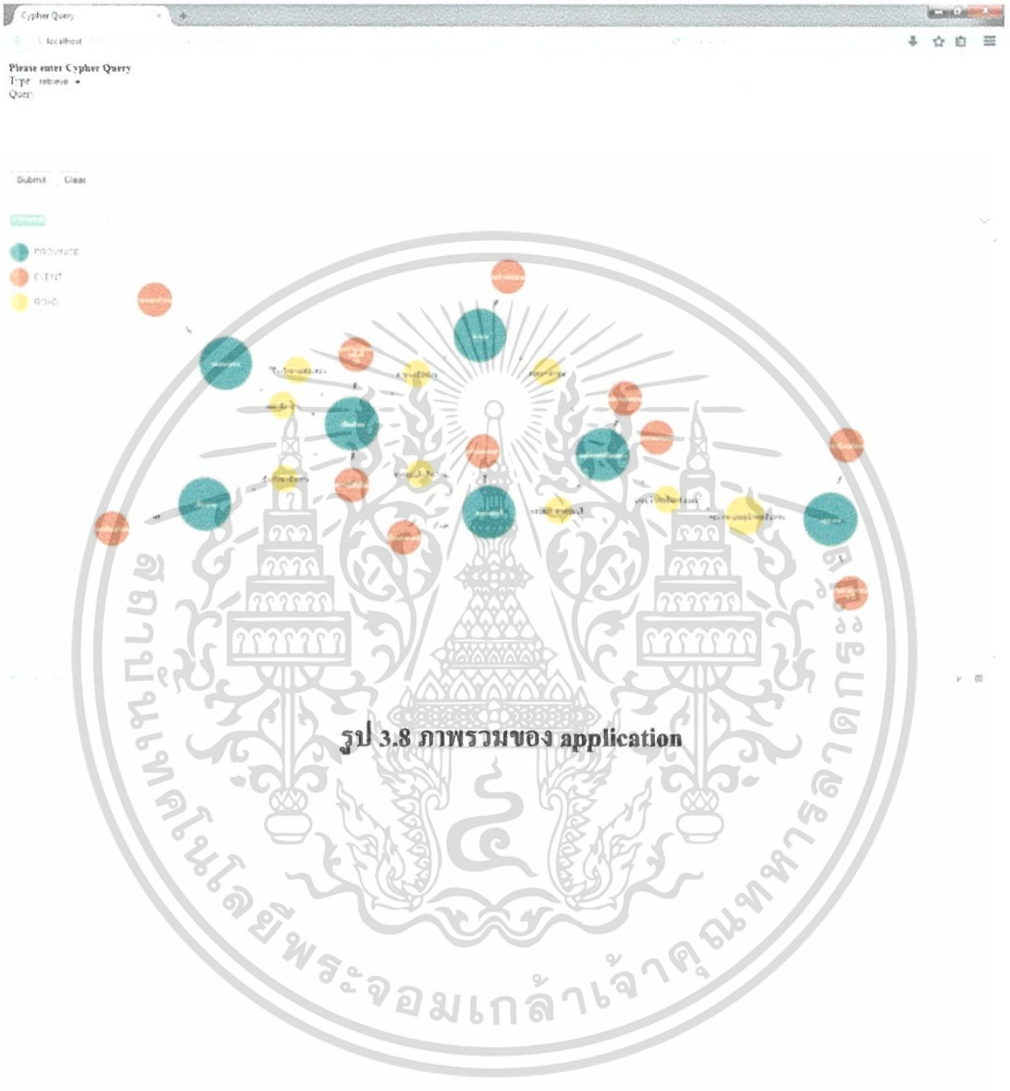
```
//Sequenced insert(Input)

MATCH (p:AAA)
WHERE p.BBB = "CCC"
CREATE (a:DDD)
CREATE (p)-[:has{FROM_DATE:'from date', TO_DATE:'to
date'}]->(a);

//Sequenced insert(Output)

MATCH (p:AAA)
WHERE p.BBB = "CCC"
CREATE (a:DDD)
CREATE (p)-[:has{FROM_DATE:'from date', TO_DATE:'to
date'}]->(a);

//Sequenced insert(Input)


MATCH (p:AAA)
WHERE p.BBB = "CCC"
CREATE (a:DDD)
CREATE (p)-[:has{FROM_DATE:'from date', TO_DATE:'to
date'}]->(a);

//Sequenced insert (Output)


MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE <= 'from
date' and has.TO_DATE > 'to date'
     CREATE (L2:BBB { CCC:'EEE' })
     CREATE (a)-[:has{ FROM_DATE:'to date',
TO_DATE:has.TO_DATE}]->(L2);
     MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE < 'from
date' and has.TO_DATE >= 'from date'
     SET has.TO_DATE = from date' ;
      MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE < 'to date'
and has.TO_DATE >= 'to date'
     SET has.FROM_DATE = 'to date' ;
      MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE >= 'to
date' and has.TO_DATE <= 'from date
     delete has, L;
```

```
//Sequenced delete (Input)
MATCH (a)-[has:AAA]->(L:BBB)
WHERE L.CCC = "EEE" and has.FROM_DATE >= 'from date'
          and has.TO_DATE <= 'to date'
DELETE has , L ;

//Sequenced delete (Output)
MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE <= 'from
date' and has.TO_DATE > 'to date'
     CREATE (L2:BBB { CCC:'EEE' })
     CREATE (a)-[:has{ FROM_DATE:'to date',
TO_DATE:has.TO_DATE}]->(L2);
     MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE < 'from
date' and has.TO_DATE >= 'from date'
     SET has.TO_DATE = from date' ;
      MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE < 'to date'
and has.TO_DATE >= 'to date'
     SET has.FROM_DATE = 'to date' ;
      MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE >= 'to
date' and has.TO_DATE <= 'from date
     delete has, L;
```

```
//Sequenced update (Input)

MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE >= 'from
date' and      has.TO_DATE <= 'to date'
     SET has.DDD = 'G' ;


//Sequenced update (Output)

MATCH (a)-[has:AAA]->(L:BBB)
     WHERE L.CCC = "EEE" and has.FROM_DATE < 'from
date' and has.TO_DATE > 'from date'
     CREATE (L2:BBB { CCC:'EEE' })
     CREATE (a)-[:has{DDD:has.DDD,
FROM_DATE:has.FROM_DATE, TO_DATE:'from date'}]->(L2);
     MATCH (a)-[has:AAA]->(L:BBB)
      WHERE L.CCC = "EEE" and has.FROM_DATE < 'to
date' and has.TO_DATE > 'to date'
      CREATE (L2:BBB { CCC:'EEE' })
     CREATE (a)-[:has{DDD:has.DDD , FROM_DATE:'to
date', TO_DATE:has.TO_DATE}]->(L2);
     MATCH (a)-[has:AAA]->(L:BBB)
      WHERE L.CCC = "EEE" and has.FROM_DATE < 'to
date' and has.TO_DATE > 'from date'
     SET has.DDD = 'G' ;
     MATCH (a)-[has:AAA]->(L:BBB)
      WHERE L.CCC = "EEE" and has.FROM_DATE < 'from
date' and has.TO_DATE > 'from date'
     SET has.FROM_DATE = 'from date' ;
     MATCH (a)-[has:AAA]->(L:BBB)
      WHERE L.CCC = "EEE" and has.FROM_DATE < 'to
date' and has.TO_DATE > 'to date'
     SET has.TO_DATE = 'to date' ;
```

ส่วนระบบแปลงคำสั่ง Query นี้จะมีหลักการทำงาน คือใช้การ Detect String โดยเมื่อรับข้อมูลตรงตามที่กำหนดจะทำการแปลงคำสั่ง โดยพัฒนาขึ้นมาในลักษณะ web-base(.jsp)

3.3.2 เมื่อทั้ง 2 ส่วนเรียบร้อยก็นำมารวมกัน เป็นโปรแกรมเดียวได้หน้าตาดังรูป



รูป 3.8 ภาพรวมของ application

บทที่ 4

ผลการทดลอง

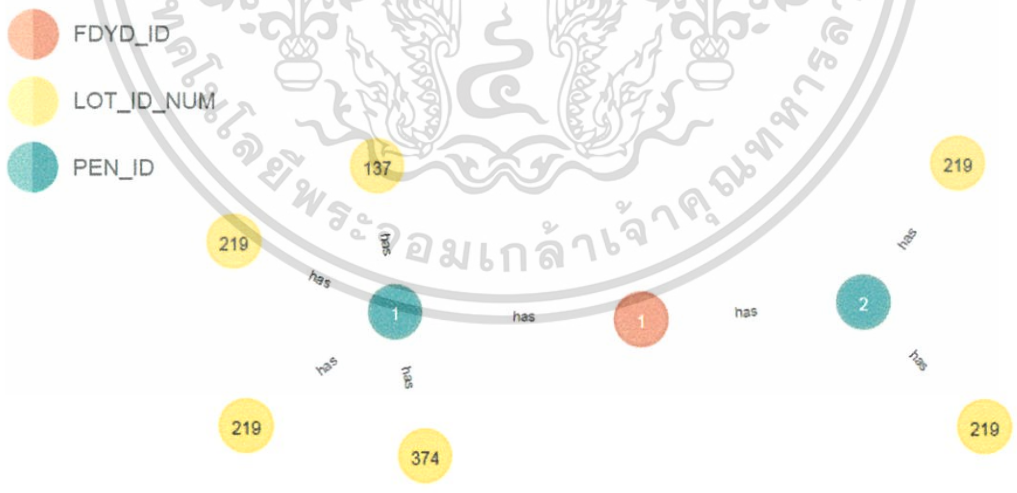
4.1 การใช้ระบบฐานข้อมูลเชิงเวลาบนระบบฐานข้อมูลที่ไม่ใช้เอสคิวแอล

ทดสอบโดยการสร้างกราฟโดยอ้างอิงจากตารางที่มีการเก็บข้อมูลเชิงเวลาดังนี้

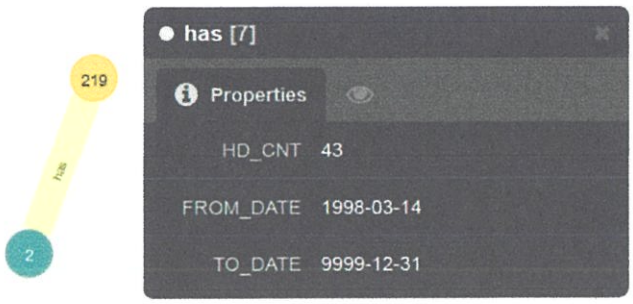
ตาราง 4.1 The LOT_LOC table [ที่มา: Managing Temporal Data A Five-Part Series]

FDYD_ID	LOT_ID_NUM	PEN_ID	HD_CNT	FROM_DATE	TO_DATE
1	137	1	17	1998-02-07	1998-02-18
1	219	1	43	1998-02-25	1998-03-01
1	219	1	20	1998-03-01	1998-03-14
1	219	2	23	1998-03-01	1998-03-14
1	219	2	43	1998-03-14	9999-12-31
1	374	1	14	1998-02-20	9999-12-31

ได้กราฟดังรูป



รูป 4.1 กราฟที่เก็บข้อมูลเชิงเวลา



รูป 4.2 การดูข้อมูลของ Relationship

จะเห็นได้ว่าทั้งตาราง 4.1 และ รูป 4.1 (กราฟ) นำเสนอข้อมูลเดียวกัน โดยเปรียบเทียบกันได้ดังนี้

ตาราง จะแสดงรายละเอียดของข้อมูลได้ดีกว่า

กราฟ จะแสดงข้อมูลเป็นภาพรวม ข้อมูลที่เห็นไม่ค่อยละเอียดจำเป็นต้องเจาะจงดูข้อมูลเป็นจุดๆ

4.1.1 การตอบคำถามของระบบฐานข้อมูลเชิงเวลา

1) ตอนนี้วัวที่มาจากฝูง 219 ในลาน 1 อยู่ในคอกใดบ้างและเป็นจำนวนเท่าใด *[ที่มา: Managing Temporal Data A Five-Part Series]*

หาคำตอบโดยใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (f:FDYD_ID)-->(PEN_ID:PEN_ID)-[has:has]-
>(l:LOT_ID_NUM)
     WHERE (f.ID = '1') and (l.LID = '219') and
(has.TO_DATE='9999-12-31')
     RETURN PEN_ID, has ;
```

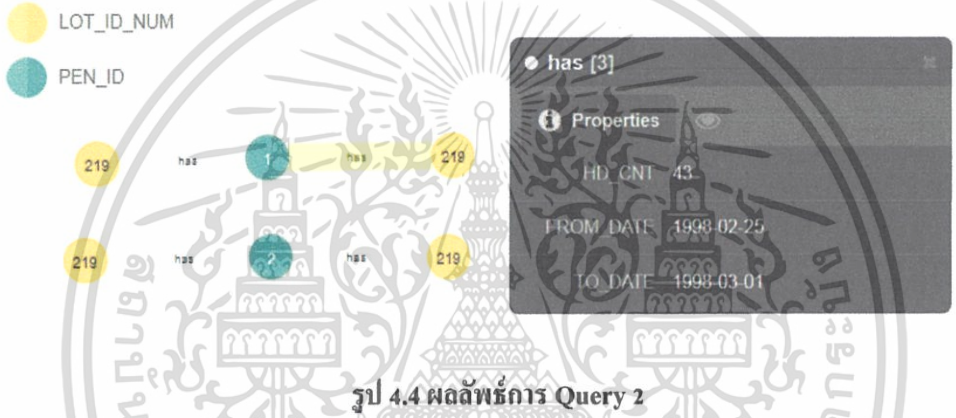
ผลที่ได้

รูป 4.3 ผลลัพธ์การ Query 1

2) วัวจากฝูง 219 ในลาน 1 เคยอยู่ในคอกใดบ้าง [ที่มา: Managing Temporal Data A Five-Part Series] หาคำตอบโดยใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (f:FDYD_ID)-->(PEN_ID:PEN_ID)-[has:has]-
>(l:LOT_ID_NUM)
WHERE (f.ID = '1') and (l.LID = '219')
RETURN PEN_ID,has
ORDER BY has.FROM_DATE;
```

ผลที่ได้

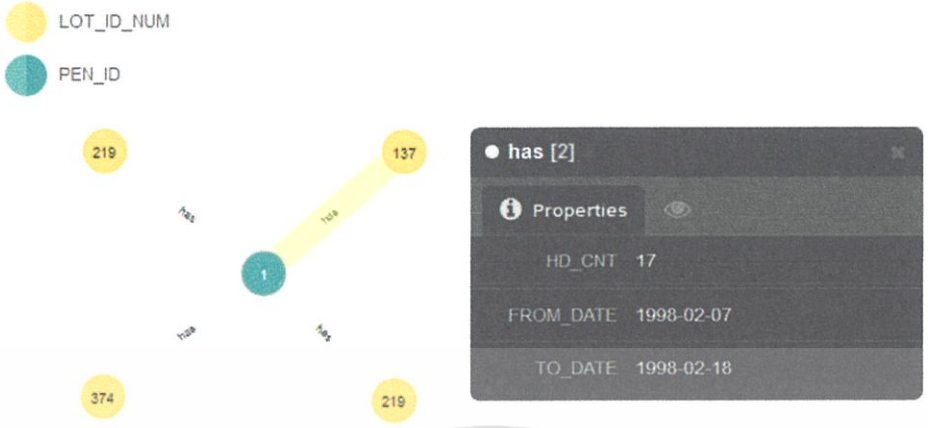


รูป 4.4 ผลลัพธ์การ Query 2

3) วัวฝูงไหนที่เคยอยู่คอกเดียวกันแต่คนละเวลา [ที่มา: Managing Temporal Data A Five-Part Series]
หาคำตอบโดยใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (f:FDYD_ID)-->(PEN_ID:PEN_ID)-[has:has]-
>(l:LOT_ID_NUM)
MATCH (f2:FDYD_ID)-->(PEN_ID2:PEN_ID)-[has2:has]-
>(l2:LOT_ID_NUM)
WHERE (f.ID = f2.ID) and (l.LID < l2.LID) and
(PEN_ID.PID = PEN_ID2.PID)
RETURN l,l2,PEN_ID
```

ผลที่ได้



รูป 4.5 ผลลัพธ์การ Query 3

4) วัวฝูงไหนที่เคยอยู่ในคอกเดียวกันในช่วงเวลาเดียวกันด้วย [ที่มา: Managing Temporal Data A Five-Part Series]

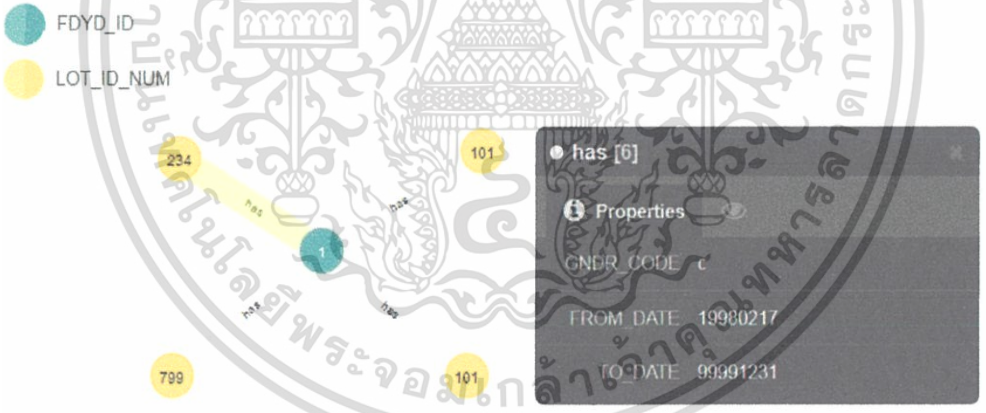
หาคำตอบโดยใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (f:FDYD_ID)-->(PEN_ID:PEN_ID)-[has:has]-
>(l:LOT_ID_NUM)

MATCH (f2:FDYD_ID)-->(PEN_ID2:PEN_ID)-[has2:has]-
>(l2:LOT_ID_NUM)

WHERE (f.ID = f2.ID) and (l.LID < l2.LID) and
(PEN_ID.PID = PEN_ID2.PID) and ((CASE WHEN
has.FROM_DATE > has2.FROM_DATE

THEN has.FROM_DATE ELSE l2.FROM_DATE END)<

(CASE WHEN has.TO_DATE > has2.TO_DATE

THEN has2.TO_DATE ELSE has.TO_DATE END))

RETURN l,l2,PEN_ID,

CASE WHEN has.FROM_DATE > has2.FROM_DATE

THEN has.FROM_DATE ELSE l2.FROM_DATE END as
FROM_DATE,     CASE WHEN has.TO_DATE > has2.TO_DATE

THEN has2.TO_DATE ELSE has.TO_DATE END as TO_DATE;
```

ผลที่ได้

รูป 4.6 ผลลัพธ์การ Query 4

4.1.2 การ modifications



รูป 4.7 กราฟจากตาราง LOT_LOC ใช้ในการทดสอบ Modifications

(ที่มา: Managing Temporal Data A Five-Part Series)

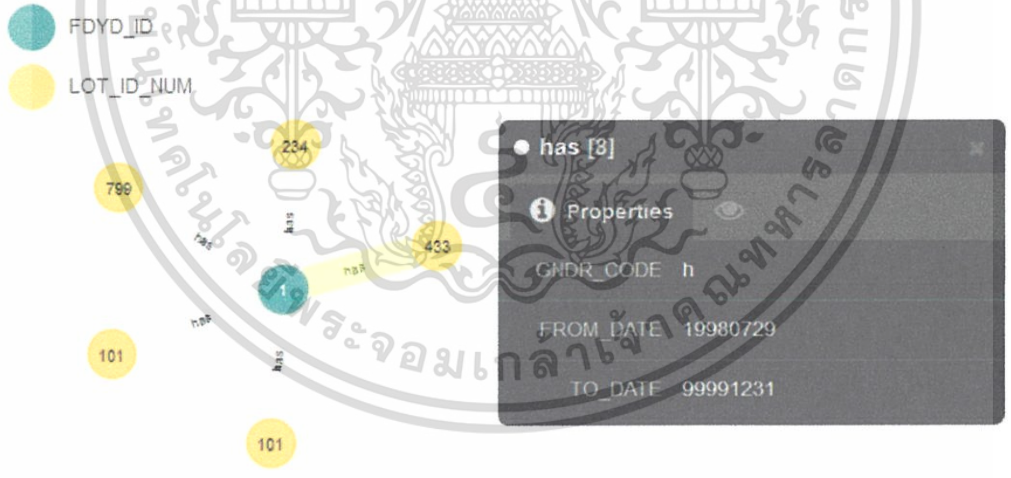
1) Current Insert

ต้องการเพิ่มข้อมูลของลูกวัวตัวเมีย (Heifer calf) ฝูงใหม่(433) [ที่มา: Managing Temporal Data A Five-Part Series]

ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (p:FDYD_ID)
      WHERE p.ID = "1"
      CREATE (a:LOT_ID_NUM { LID:'433' })
      CREATE (p)-[:has{GNDR_CODE:'h',
FROM_DATE:'19980729',
                  TO_DATE:'99991231'}]->(a);
      // สมมติว่าวันนี้เป็นวันที่ 29/07/1998
```

ผลที่ได้



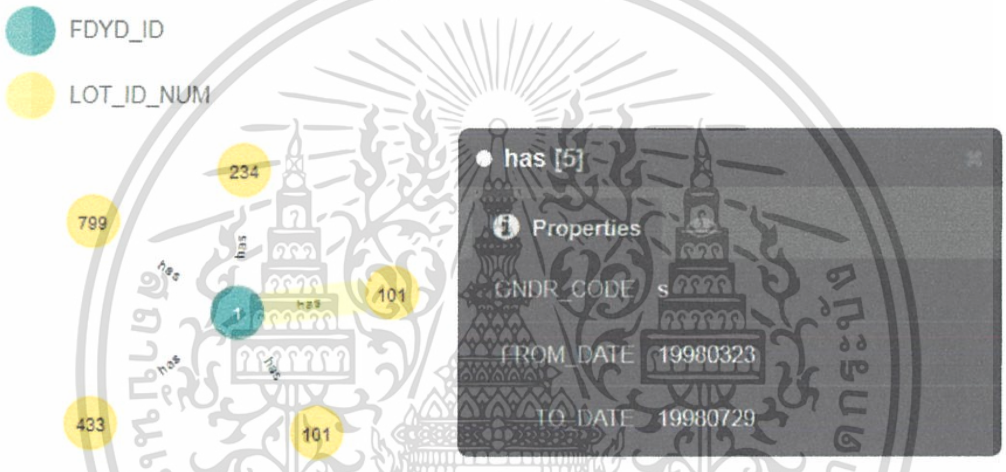
รูป 4.8 ผลลัพธ์จากการทำ Current Insert

2) Current Delete

วัวฝูง 101 จะย้ายออกไป [ที่มา: Managing Temporal Data A Five-Part Series] ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH   (a)-[has:has]->(L:LOT_ID_NUM)
WHERE   L.LID = "101" and has.TO_DATE = '99991231'
SET       has.TO_DATE = '19980729';
```

ผลที่ได้



รูป 4.9 ผลลัพธ์จากการทำ Current Delete

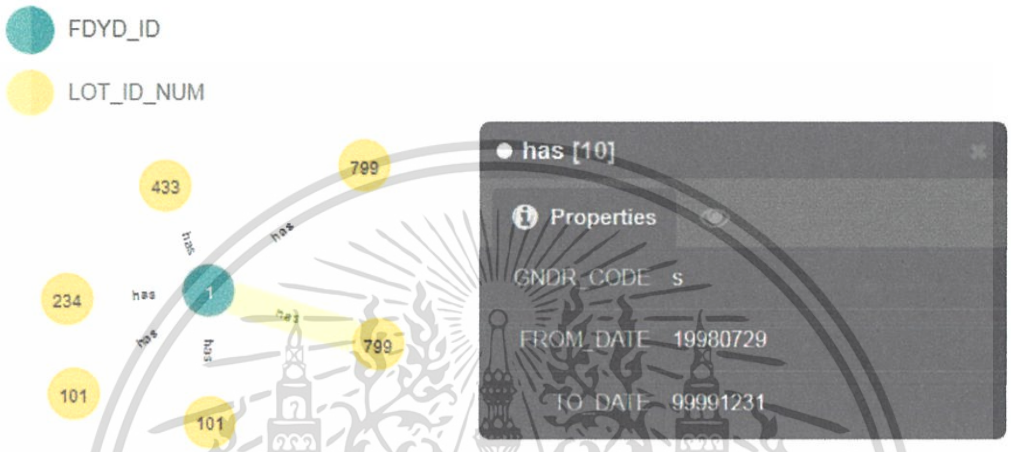
3) Current Update

ต้องการ Update ข้อมูลของวัวฝูง 799 จาก Cow เป็น Steer [ที่มา: Managing Temporal Data A Five-Part Series] ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE <= '19980729'
and has.TO_DATE > '19980729'
CREATE (L2:LOT_ID_NUM { LID:'799' })
CREATE (a)-[:has{GNDR_CODE:'s',
FROM_DATE:'19980729', TO_DATE:'99991231'}]->(L2)
SET has.TO_DATE = '19980729'; //case 1,2
```

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "799" and has.FROM_DATE >= '19980729'
SET has.GNDR_CODE = 's'; //case 3
```

ผลที่ได้



รูป 4.10 ผลลัพธ์จากการทำ Current Update

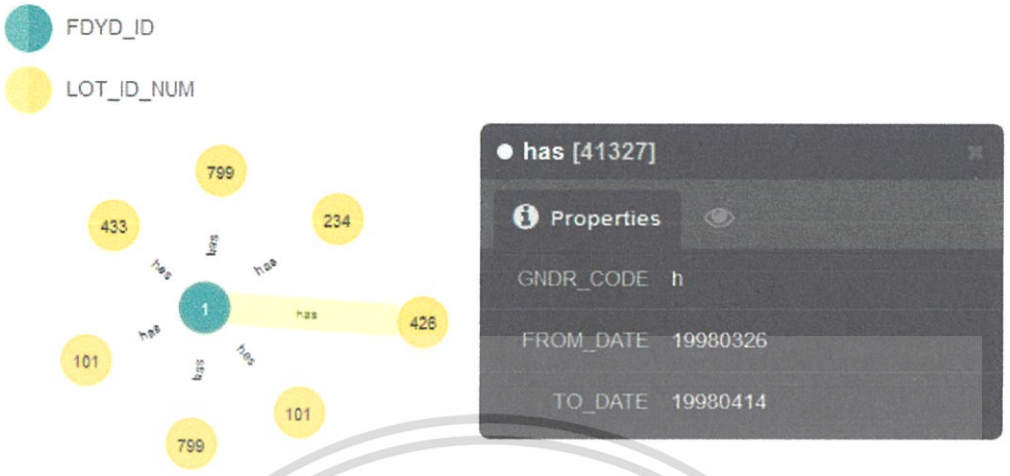
4) Sequenced Insert

ต้องการเพิ่มข้อมูลของลูกวัวตัวเมียฝูง 426 ที่จะมาอยู่ตั้งแต่วันที่ 26/3/1998 ถึงวันที่ 14/4/1998 [ที่มา: Managing Temporal Data A Five-Part Series]

ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (p:FDYD_ID)
     WHERE p.ID = "1"
     CREATE (a:LOT_ID_NUM { LID:'426' })
     CREATE (p)-[:has{GNDR_CODE:'h',
FROM_DATE:'19980326',
TO_DATE:'19980414'}]->(a);
```

ผลที่ได้



รูป 4.11 ผลลัพธ์จากการทำ Sequenced Insert

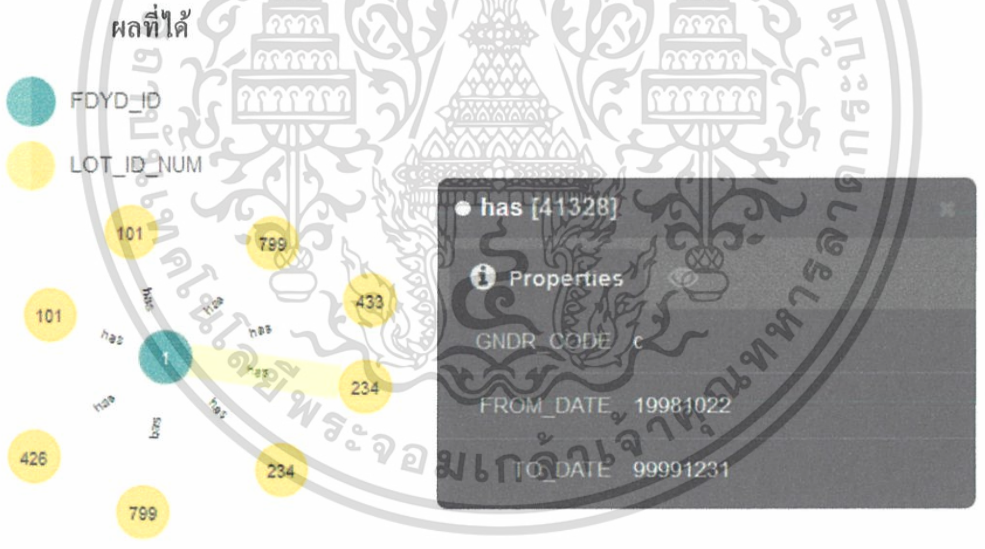
5) Sequenced Delete

ต้องการลบข้อมูลของฝูง 234 ตั้งแต่วันที่ 1/10/1998 ถึงวันที่ 22/10/1998 [ที่มา: Managing Temporal Data A Five-Part Series]

ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "234" and has.FROM_DATE <=
'19981001' and has.TO_DATE > '19981022'
     CREATE (L2:LOT_ID_NUM { LID:'234' })
     CREATE (a)-[:has{GNDR_CODE:has.GNDR_CODE ,
FROM_DATE:'19981022', TO_DATE:has.TO_DATE}]->(L2);
```

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "234" and has.FROM_DATE <
'19981001' and has.TO_DATE >= '19981001'
SET has.TO_DATE = '19981001' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "234" and has.FROM_DATE <
'19981022' and has.TO_DATE >= '19981022'
SET has.FROM_DATE = '19981022' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "234" and has.FROM_DATE >=
'19981001' and has.TO_DATE <= '19981022'
delete has, L;
```



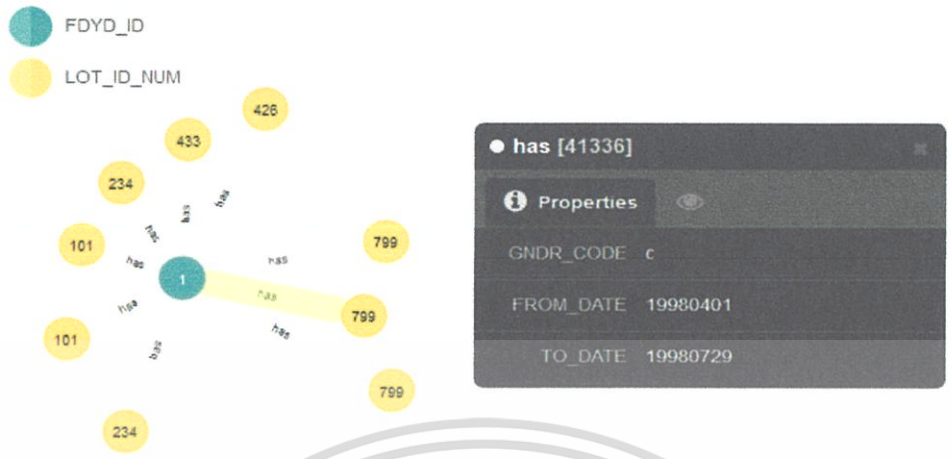
รูป 4.12 ผลลัพธ์จากการทำ Sequenced Delete

6) Sequenced update

ต้องการ update GNDR_CODE ของวัวฝูง 799 ในช่วงวันที่1/3/1998 ถึงวันที่ 1/4/1998 เป็น 's' [ที่มา: Managing Temporal Data A Five-Part Series] ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "799" and has.FROM_DATE <
'19980301' and has.TO_DATE > '19980301'
     CREATE (L2:LOT_ID_NUM { LID:'799' })
     CREATE (a)-[:has{GNDR_CODE:has.GNDR_CODE ,
FROM_DATE:has.FROM_DATE, TO_DATE:'19980301'}]-
>(L2);
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "799" and has.FROM_DATE <
'19980401' and has.TO_DATE > '19980401'
     CREATE (L2:LOT_ID_NUM { LID:'799' })
     CREATE (a)-[:has{GNDR_CODE:has.GNDR_CODE ,
FROM_DATE:'19980401', TO_DATE:has.TO_DATE}]->(L2);
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "799" and has.FROM_DATE <
'19980401' and has.TO_DATE > '19980301'
     SET has.GNDR_CODE = 's' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "799" and has.FROM_DATE <
'19980301' and has.TO_DATE > '19980301'
     SET has.FROM_DATE = '19980301' ;
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
     WHERE L.LID = "799" and has.FROM_DATE <
'19980401' and has.TO_DATE > '19980401'
     SET has.TO_DATE = '19980401' ;
```

ผลที่ได้



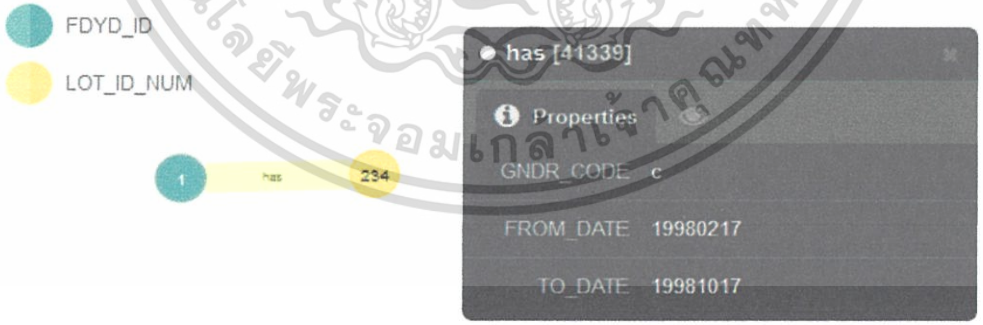
รูป 4.13 ผลลัพธ์จากการทำ Sequenced Update

7) **Non-Sequenced Delete**

ต้องการลบข้อมูลของวัวล็อต 234 ที่มีช่วงของข้อมูลนานเกิน 3 เดือนใช้คำสั่งภาษาไซเฟอร์ดังนี้ ([ที่มา: Managing Temporal Data A Five-Part Series]

```
MATCH (a)-[has:has]->(L:LOT_ID_NUM)
WHERE L.LID = "234" and has.TO_DATE-has.FROM_DATE > 90
delete has, L;
```

ผลที่ได้



รูป 4.14 กราฟส่วนที่ถูก Non-Sequenced Delete

จากการทดลองจะเห็นได้ว่าระบบฐานข้อมูลรูปแบบกราฟ สามารถใช้เป็นระบบฐานข้อมูลเชิงเวลาได้

4.2 ผลการทดลอง Application

แบ่งออกเป็น 2 ส่วนคือ

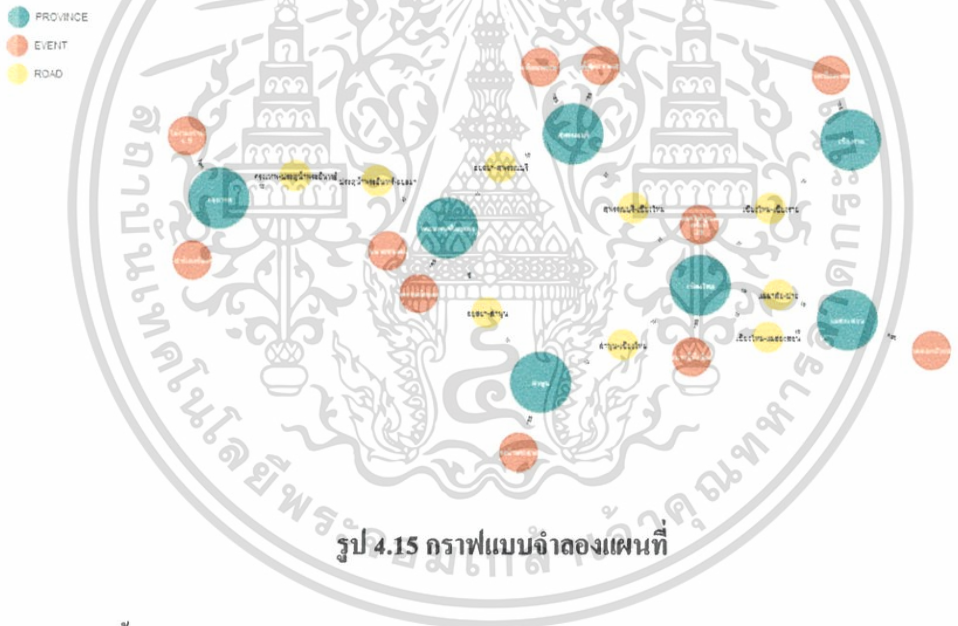
1) ผลการทดลองส่วนแปลงคำสั่ง
2) ผลการทดลองส่วนระบบฐานข้อมูล

4.2.1 การทดลองส่วนแปลงคำสั่ง

จากการทดลองโดยการใช้คำสั่งพื้นฐาน Insert, Update, Delete ตามลักษณะที่อ้างอิงจากเอกสาร Managing Temporal Data A Five-Part Series พบว่าแปลงเป็นคำสั่งสำหรับใช้งานกับระบบฐานข้อมูลเชิงเวลาได้ถูกต้อง แต่หากใส่คำสั่งไม่ตรงรูปแบบการแปลงจะผิดพลาดและทำให้เกิด Error ขึ้น

4.2.1 ผลการทดลองส่วนระบบฐานข้อมูล

ลักษณะแบบจำลองที่ทำมีลักษณะดังนี้



รูป 4.15 กราฟแบบจำลองแผนที่

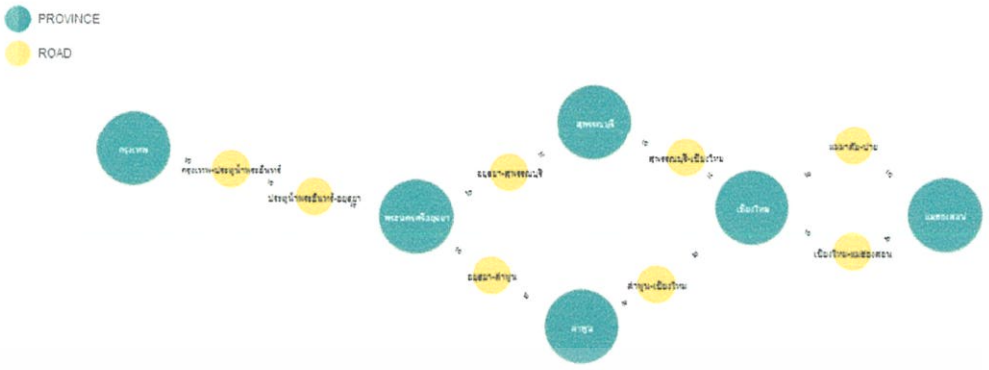
แบบจำลองนี้สามารถใช้ตอบคำถามได้หลายแบบตัวอย่างเช่น

1) แสดงเส้นทางจากกรุงเทพไปแม่ฮ่องสอน

ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH (a:PROVINCE{NAME:"กรุงเทพ"}) -[:to*]->(r)-[:to*]-
>(b:PROVINCE{NAME:"แม่ฮ่องสอน"})
RETURN *;
```

ผลที่ได้



รูป 4.16 กราฟเส้นทางจากกรุงเทพไปแม่ฮ่องสอน

2) แสดง EVENT ทั้งหมดที่เกิดในเส้นทางจากกรุงเทพไปแม่ฮ่องสอนตามช่วงเวลาที่กำหนด

ใช้คำสั่งภาษาไซเฟอร์ดังนี้

```
MATCH ()-[h:has]->(e:EVENT),(a:PROVINCE{NAME:"กรุงเทพ"}) -
[:to*]->(r)-[:to*]->(b:PROVINCE{NAME:"แม่ฮ่องสอน"})

WHERE (r)-[h]->(e) and h.FROM_DATE>20151201 and
h.TO_DATE<20151231

RETURN DISTINCT r, e;
```

ผลที่ได้

รูป 4.17 EVENT ทั้งหมดที่เกิดในเส้นทางจากกรุงเทพไปแม่ฮ่องสอนในช่วงเดือนธันวาคม

จากการตอบคำถามข้างต้นสามารถต่อยอดโดยการเพิ่มเงื่อนไขเข้าไปเพื่อตอบคำถามเช่น ทางใดเส้นทางสั้นที่สุดหรือแม้กระทั่งทางใดมีผ่านเทศกาลเยอะในช่วงวันเวลาที่กำหนด เมื่อนำทั้ง 2 ส่วนมารวมกันพบว่าการทำงานตามเงื่อนไขที่กำหนด สามารถทำงานได้เป็นปกติ

4.3 ภาพรวม ปัญหา และข้อเสนอแนะ

1) การแปลงคำสั่งยังจำกัดอยู่ที่การ Modification เนื่องจากใช้การอ่านสายข้อความ String ในการ Detect คำ หากเปลี่ยนมาใช้ LEX กับ YACC ในการ Detect ไวยกรณ์จะช่วยให้การใส่ Input มีความยืดหยุ่นยิ่งขึ้น

2) การแปลง NIAM เป็น กราฟยังไม่มีหลักการที่ชัดเจน อาจจะมีรูปแบบอื่นที่ดีกว่านี้

3) ระบบฐานข้อมูลเชิงเวลาสามารถพัฒนาให้ใช้งานง่ายขึ้นได้ควรส่งเสริมให้มีการใช้แพร่หลาย

4) ระบบฐานข้อมูลรูปแบบกราฟ มีข้อดีที่แตกต่างจาก ตารางคือสามารถแสดงความสัมพันธ์แบบ Hierarchy ได้แบบเข้าใจง่าย

บทที่ 5

วิเคราะห์ และ สรุปผล

5.1 บทสรุป

ระบบฐานข้อมูลเชิงเวลา เป็นระบบฐานข้อมูลที่มีประสิทธิภาพในการตอบคำถามสูง เนื่องจากเป็นข้อมูลเชิงเวลาจึงทำให้สามารถแสดงข้อมูล ได้ทั้งข้อมูลในอดีต ปัจจุบันและ อนาคต ถ้ามีพื้นที่ในการเก็บข้อมูลมากพอ แต่ก็มีข้อเสียตรงที่การใช้งานคำสั่ง Query และ Modifications มีความซับซ้อนต่างจากการ Query และ Modifications บนระบบฐานข้อมูลปกติมากโดยเฉพาะ Modifications ที่หากมีข้อผิดพลาดจะทำให้ข้อมูลไม่ถูกต้องได้

ในปัจจุบันระบบฐานข้อมูลที่ใช้ภาษาเอสคิวแอล เริ่มมีการรองรับการใช้งาน ระบบฐานข้อมูลเชิงเวลา โดยมีการใช้คำสั่งเฉพาะเพื่อช่วยลดความซับซ้อนในการใช้งานลง และยังคงมีการพัฒนาอย่างต่อเนื่อง แต่ในระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล ยังไม่มีการรองรับระบบฐานข้อมูลเชิงเวลา คณะผู้จัดทำจึงถือโอกาสนี้ในการทดลองออกแบบการจัดเก็บข้อมูลเชิงเวลา บนระบบฐานข้อมูลที่ไม่ใช้ภาษาเอสคิวแอล โดยระบบฐานข้อมูลที่เลือกมาใช้งานคือ Neo4j ซึ่งเป็นระบบฐานข้อมูลรูปแบบกราฟที่เริ่มมีการนำมาใช้งานในปัจจุบันมากขึ้น

จากการทดลองพบว่าสามารถใช้งานระบบฐานข้อมูลเชิงเวลา บนระบบฐานข้อมูล Neo4j ได้ แต่การ Modifications จะมีความซับซ้อนมากตามที่ได้กล่าวไปข้างต้น ซึ่งเป็นภาระแก่ผู้ใช้งานที่จะต้องคอยตรวจสอบคำสั่งไม่ให้มีข้อผิดพลาด คณะผู้จัดจึงได้พัฒนาโปรแกรมส่วนที่ใช้สำหรับแปลงคำสั่งขึ้นมา ซึ่งผลการทดลองพบว่าช่วยให้การใช้งานระบบฐานข้อมูลเชิงเวลาได้ง่ายขึ้น แต่ทั้งนี้รูปแบบคำสั่งที่ใช้จะต้องตรงตามเงื่อนไขที่ได้กำหนดไว้

ในส่วนของโปรแกรมประยุกต์ที่คณะผู้จัดทำทำขึ้นนั้นเป็นการจำลองการใช้งานระบบฐานข้อมูลเชิงเวลาบนระบบฐานข้อมูล Neo4j โดยเลือกใช้ข้อมูลสถานที่ เส้นทาง และเหตุการณ์ต่างๆ จากข้อมูลจริง เพื่อเป็นการแสดงให้เห็นถึงการทำงานของระบบฐานข้อมูลเชิงเวลา ควบคู่กับจุดเด่นของระบบฐานข้อมูลรูปแบบกราฟที่สามารถค้นหาข้อมูลที่มีลักษณะเป็น Hierarchy ได้ดี โดยจากผลการทดลองพบว่ารูปแบบการเก็บข้อมูลที่ใช้สามารถตอบคำถามประเภท Temporal Query ได้

5.2 แนวทางในการพัฒนาต่อ

1) โปรแกรมในส่วนของการแปลงคำสั่งยังมีข้อจำกัดอยู่มาก คือต้องมีลำดับคำสั่งตามเงื่อนไขที่กำหนดหากสามารถพัฒนาให้สามารถยืดหยุ่นกว่านี้ได้จะช่วยให้การใช้งานง่ายขึ้นมาก

2) *การออกแบบการเก็บข้อมูลในระบบฐานข้อมูลเชิงเวลา บนระบบฐานข้อมูลรูปแบบกราฟ* ยังไม่มีหลักการที่แน่นอน สามารถพัฒนาต่อ เพื่อให้ง่ายต่อการออกแบบได้

3) ทดลองนำระบบฐานข้อมูลเชิงเวลา ไปประยุกต์ใช้กับระบบบฐานข้อมูลรูปแบบอื่นๆ

บรรณานุกรม

Richard T. Snodgrass. 1998. Managing Temporal Data A Five-Part Series. A TIMECENTER Technical Report.

Ian Robinson, Jim Webber and Emil Eifrem. 2013. Graph Databases. O'Reilly Media, Inc.

Abraham Silberschatz, Henry F. Korth and S. Sudarshan. 2006. Database System Concept Fifth Edition. Singapore : McGraw-Hill Education.

Rik Van Bruggen. 2014. Learning Neo4j. Packt Publishing Ltd.

Onofrio Panzarino. 2014. Learning Cypher. Packt Publishing Ltd.

Justin J. Miller. 2013. Graph Database Applications and Concepts with Neo4j. Georgia Southern University.

The Neo4j Team. 2015. The Neo4j Manual v2.1.7. [Online]. Available: http://neo4j.com/docs/2.1.7/index.html.

Neubauer Peter. 2010. Graph Databases, NOSQL and Neo4j. [Online]. Available: http://www.infoq.com/articles/graph-nosql-neo4j.

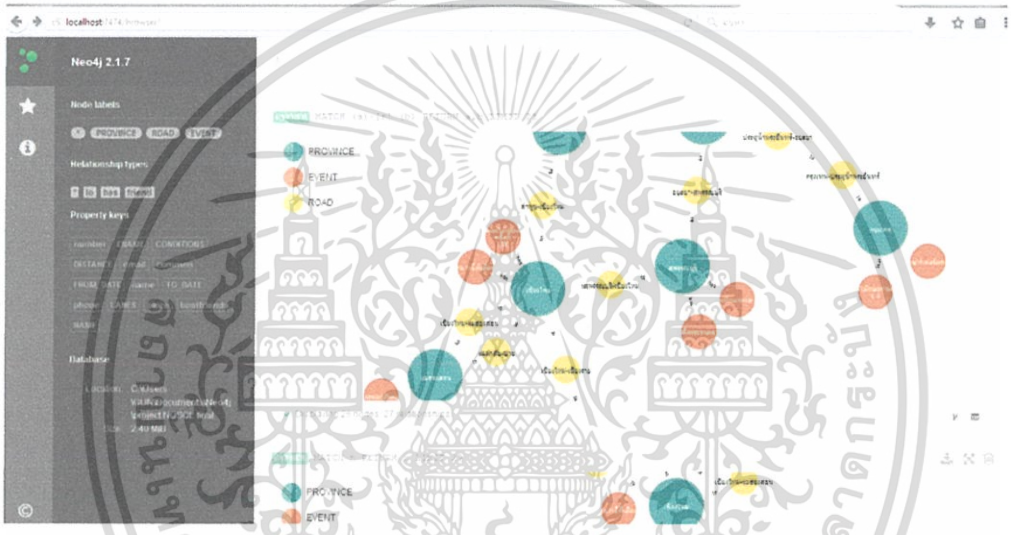
Wikipedia. 2015. NoSQL. [Online]. Available: http://en.wikipedia.org/wiki/NoSQL

ภาคผนวก

ก. การเชื่อมต่อกับ Neo4j Graph Database

การเชื่อมต่อกับ Neo4j Graph Database นั้น สามารถเชื่อมต่อได้โดยสร้างการเชื่อมต่อไปยัง port 7474

โดยการเปิดเบราเซอร์เชื่อมต่อไปยัง localhost:7474 ซึ่งจะมี interface แสดงผลปรากฏขึ้นมา สามารถใช้ interface นี้ในการติดต่อกับ Neo4j Graph Database ได้



รูป ก. interface ของ Neo4j

ในหน้า Interface นี้จะมีฟังก์ชันการใช้งานอยู่ทางซ้ายของหน้าจอ interface สามารถใช้แทนการใส่คำสั่ง Query ได้