ระบบฐานข้อมูลแบบกระจายที่มีการค้นหาข้อมูลเชิงความหมาย

DISTRIBUTED DATABASE SYSTEM WITH SEMANTIC QUERY OPTIMIZER

ณชิรัตน์ ราชบุรี

NACHIRAT RACHBUREE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

ภาควิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2559

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

ระบบฐานข้อมูลแบบกระจายที่มีการค้นหาข้อมูลเชิงความหมาย

DISTRIBUTED DATABASE SYSTEM WITH SEMANTIC QUERY OPTIMIZER

นชิรัตน์ ราชบุรี

NACHIRAT RACHBUREE

เลขหมู่.....
เลขทะเบียน.....75112
วัน,เดือน,ปี.....19 ต.ค. 2550

b.....11833749
i.....

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2550

DISTRIBUTED DATABASE SYSTEM WITH SEMANTIC QUERY OPTIMIZER

NACHIRAT RACHBUREE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2007

COPYRIGHT 2007

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ ระบบฐานข้อมูลแบบกระจายที่มีการค้นหาข้อมูลเชิงความหมาย

Distributed Database System with Semantic Query Optimizer

นักศึกษา นายนชิรัตน์ ราชบุรี

รหัสประจำตัว 45061206

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร.ศุภมิตร จิตตะยโศธร

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วรวัฒน์	ลิ้มโภคา	
ดร.ชุติเมษฎ์	ศรีนิลทา	
รศ.ดร.นุชรี	เปรมชัยสวัสดิ์	
รศ.สมศักดิ์	มิตะถา	
รศ.ดร.ศุภมิตร	จิตตะยโศธร	

วัน / เดือน/ ปี ที่สอบ 28 พฤษภาคม 2550 เวลา 09.00-11.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-403)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.จารวัตร เจริญสุข)

คณบดีบัณฑิตวิทยาลัย

วันที่.....๒๔.....**เดือน**...กรกฎาคม....**พ.ศ.**๒๕๕๐.....

หัวข้อวิทยานิพนธ์	ระบบฐานข้อมูลแบบกระจายที่มีการค้นหาข้อมูลเชิงความหมาย
นักศึกษา	นายนชิรัตน์ ราชบุรี
รหัสประจำตัว	45061206
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2550
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร. ศุภมิตร จิตตะยโศธร

บทคัดย่อ

ในระบบจัดการฐานข้อมูลแบบกระจาย ที่มีเข้าถึงผ่านโปรโตคอลเกทเวย์ ผู้ใช้งานต้องระวังถึงคุณสมบัติที่แตกต่างกันระหว่างระบบจัดการฐานข้อมูลของแต่ละโลคอลไซต์ ระบบปรับปรุงคิวรีในฐานข้อมูลแบบกระจาย โดยปกติแล้ว จะขึ้นอยู่กับการเก็บสถิติในโลคอลไซต์ บทความนี้จะอธิบายถึง ระบบจัดการฐานข้อมูลแบบกระจาย ซึ่งรองรับ โกลบอลสกีมา แฟรกเมนเทชั่นสกีมา และ ออโลเคชั่น สกีมา ด้วยระบบตัดสินใจค้นหาข้อมูลเชิงความหมาย ในระบบนี้จะทำงานโดยรองรับการจัดการข้อมูลในระดับของโกลบอล ที่มีการผสมผสานกันในระดับ แฟรกเมนเทชั่น คุณสมบัติของระบบตัดสินใจค้นหาข้อมูลเชิงความหมายนี้ ถูกพิสูจน์ให้เห็นว่าสามารถใช้งานได้ดี และลดเวลาในการประมวลผลของคิวรีในระดับโกลบอล

Thesis Title	Distributed Database System with Semantic Query Optimizer
Student	Mr. Nachirat Rachburee
Student ID.	45061206
Degree	Master of Engineering
Program	Computer engineering
Year	2007
Thesis Advisor	Assoc. Prof. Dr. Suphamit Chittayasothorn

ABSTRACT

At present, Distributed Database System provide access to local databases via gateway protocols. Users have to be aware of product-specific features of local database management systems. Query optimization in Distributed Database System are normally based on statistics collected at local sites. This paper describes a Distributed Database System that supports global schemas, fragmentation schemas and allocation schemas together with the semantic query optimization capability. The system also supports data manipulations on global databases with mixed fragmentation. The semantic query optimization feature proved to be very useful and helps reduce response time of global queries.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดีเนื่องจากกำลังใจและพระคุณอันหาที่สุดมิได้ จากคุณพ่อคุณแม่ ข้าพเจ้าขอสำนึกในพระคุณนี้อย่างเป็นที่สุด

วิทยานิพนธ์นี้จะไม่สำเร็จลุล่วงหากปราศจากแรงผลักดัน และคำแนะนำที่มีประโยชน์ของ รศ.ดร. ศุภมิตร จิตตะยโศธร ผู้ควบคุมวิทยานิพนธ์ ข้าพเจ้าขอกราบขอบพระคุณเป็นอย่างสูง

ข้าพเจ้าขอกราบเท้า คุณครูและอาจารย์ทุกท่านตั้งแต่เล็กจนเติบใหญ่ ที่ได้มอบวิชาความรู้ให้แก่ข้าพเจ้า รวมทั้งคำสั่งสอนและอบรมให้ข้าพเจ้าเป็นคนดี ข้าพเจ้าขอกราบขอบพระคุณเป็นอย่างสูง

ข้าพเจ้าขอขอบพระคุณ ภาควิชาวิศวกรรมคอมพิวเตอร์ ที่ได้สนับสนุนเครื่องมือ ตลอดจนข้อมูล และหนังสือต่างๆ ที่ใช้ในการทำวิจัย ข้าพเจ้าขอกราบขอบพระคุณเป็นอย่างสูง

ข้าพเจ้าขอขอบคุณสำหรับกำลังใจ คำแนะนำ และประสบการณ์ที่ดีจากพี่ ๆ และเพื่อน ๆ นักศึกษา ป.โททุกท่าน ห้าปีที่ลาดกระบังข้าพเจ้าจะไม่มีวันลืม

สุดท้ายนี้คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับผู้มีพระคุณทุกท่าน หากวิทยานิพนธ์ฉบับนี้มีข้อผิดพลาดประการใดข้าพเจ้าขอน้อมรับไว้เพียงผู้เดียว

นชิรัตน์ ราชบุรี

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัญหาของการค้นหาข้อมูลในระบบฐานข้อมูลแบบกระจาย เนื่องมาจากการค้นหาข้อมูลแบบกระจายผ่านระบบเครือข่าย ในแต่ละไซต์มีระบบจัดการฐานข้อมูลที่แตกต่างกันและมีระบบปฏิบัติการต่างกัน ทำให้มีปัญหาในการค้นหาข้อมูล เนื่องจาก การเชื่อมต่อระหว่างฐานข้อมูลและ คำสั่งที่แตกต่างกัน นอกจากนี้ ข้อมูลในแต่ละไซต์ยังมีการเก็บข้อมูลไว้ต่างกัน ทั้งในระดับรีเลชั่นและคอลัมน์

การใช้งานฐานข้อมูลแบบ กระจาย ที่มีข้อมูลกระจายอยู่ในแต่ละไซต์ และไซต์เหล่านั้นมีระบบจัดการฐานข้อมูลและฮาร์ดแวร์ที่แตกต่างกัน ซึ่งการเรียกค้นข้อมูลในแต่ละไซต์ที่อยู่ต่างสถานที่กันจะทำให้เกิดประสิทธิภาพในการเรียกค้นข้อมูลลดลงและยังเกิดค่าใช้จ่ายในการส่งข้อมูลผ่านระบบเครือข่ายที่มากขึ้น ซึ่งบางครั้งคำสั่งคิวรีที่ส่งออกไปยังไซต์ต่างๆ นั้น อาจจะไม่มีผลลัพธ์แสดงออกมา เนื่องจากไม่มีข้อมูลที่ผู้ใช้ต้องการค้นหาอยู่ในฐานข้อมูล แต่คำสั่งคิวรีก็ยังถูกส่งออกไปยังไซต์ต่างๆ ทำให้ระบบจัดการฐานข้อมูลทำงานโดยเสียเวลาในการทำงานของตัวประมวลผลและทรัพยากรของระบบและเพิ่มข้อมูลในการส่งข้อมูลผ่านระบบเครือข่ายอย่างสูญเปล่า ซึ่งคำสั่งคิวรีเหล่านี้ สามารถตรวจสอบเงื่อนไขที่กำหนดไว้ก่อนที่จะส่งไปยังแต่ละไซต์ได้ ทำให้ไม่เสียเวลาประมวลผล และบอกกับผู้ใช้งานได้ทันทีว่าไม่พบข้อมูลที่ต้องการ ซึ่งก็มีอยู่หลายวิธีการที่นำมาใช้ในการแปลงรูปแบบคำสั่งคิวรีให้ได้ประสิทธิภาพ ในฐานข้อมูลแบบกระจาย

1.2 จุดมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เพื่อศึกษาวิธีการลดเงื่อนไขหรือคุณลักษณะที่ไม่จำเป็นออกจากคำสั่งคิวรีโดยใช้วิธีการค้นหาข้อมูลเชิงความหมาย (semantic query optimization)
2. เพื่อศึกษาวิธีการใช้งานการค้นหาข้อมูลเชิงความหมาย บนฐานข้อมูลแบบกระจาย
3. เพื่อลดภาระการทำงานของระบบจัดการฐานข้อมูล โดยทำการตรวจสอบคำสั่งคิวรีของผู้ใช้งานที่ระดับ Front-End
4. เพื่อเพิ่มประสิทธิภาพการทำงานของแอพพลิเคชั่น (Application) โดยทำการลดเวลาการประมวลผลของคำสั่งคิวรี

1.3 แนวคิดที่ใช้ในงานวิจัย

ในงานวิจัยนี้นำเสนอการออฟติไมซ์โดยใช้หลักการของ semantic เข้ามาร่วมกับ การใช้สถิติเพื่อช่วยในการตัดสินใจ และ ช่วยการจัดการในฐานข้อมูลแบบกระจาย เมื่อมีการเรียก หรือจัดการข้อมูลในระดับโกลบอล

วัตถุประสงค์หลักของระบบคือต้องการความสมบูรณ์ของฐานข้อมูลที่ซ่อนการแยกข้อมูลทางกายภาพ การแยกข้อมูลทางลอจิกและคุณสมบัติเฉพาะของระบบจัดการฐานข้อมูลไว้จากผู้ใช้งาน โดยระบบต้องมีความสามารถในการตัดสินใจค้นหาข้อมูลเชิงความหมาย ซึ่งคิวรีในระดับโกลบอลจะต้องสามารถแยกคิวรีแล้วส่งออกไปยังแฟรกเมนท์และไซต์ ดังนั้นเวลาตอบรับของคิวรีในระดับโกลบอลก็จะลดน้อยลง เป็นไปได้ว่า แฟรกเมนท์ในที่นี้เป็นการผสมผสานกันของแฟรกเมนท์และระบบต้องสามารถจัดการ การอัพเดทซึ่งอาจจะมีการเคลื่อนย้ายข้อมูลในระดับกายภาพโดยที่อยู่ต่างไซต์กัน

1.4 การเปรียบเทียบระหว่างวิธีการที่นำเสนอกับวิธีที่มีอยู่เดิม

หลักการเดิมนั้น จะใช้วิธีการออฟติไมซ์โดยใช้วิธีการค้นหาข้อมูลเชิงความหมายเพียงอย่างเดียว หรือออฟติไมซ์ด้วยการเก็บสถิติ แต่เพียงอย่างเดียว แต่ในระบบตัดสินใจค้นหาข้อมูลเชิงความหมายนี้จะรวมทั้งสองวิธีเข้าด้วยกัน และรวมถึงการจัดการข้อมูลในระดับแฟรกเมนท์และระดับออโลเคชั่นในฐานข้อมูลแบบกระจายด้วย

1.5 ขอบเขตการวิจัย

1. ศึกษาการปรับปรุงคำสั่งคิวรีโดยวิธีการค้นหาข้อมูลเชิงความหมายบนฐานข้อมูลแบบกระจาย
2. ศึกษาการจัดการข้อมูลในระดับแฟรกเมนท์และระดับออโลเคชั่นบนฐานข้อมูลแบบกระจาย

1.6 เนื้อหาของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 6 บทคือ

บทที่ 1 กล่าวถึงความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ ขอบเขตของการวิจัย และขั้นตอนการศึกษา

บทที่ 2 กล่าวถึงการค้นหาข้อมูลเชิงความหมายและงานวิจัยที่เกี่ยวข้อง

บทที่ 3 กล่าวถึงการออกแบบระบบตัดสินใจค้นหาข้อมูลเชิงความหมาย

บทที่ 4 การทำงานบนฐานข้อมูลแบบกระจาย

บทที่ 5 เป็นการทดลองและ ผลการทดลอง

บทที่ 6 เป็นบทสรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
การค้นหาข้อมูลเชิงความหมายและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะอธิบายพื้นฐานของการค้นหาข้อมูลเชิงความหมายและงานวิจัยที่เกี่ยวข้อง โดยเนื้อหาประกอบด้วยการปรับปรุงคำสั่งคิวรี การค้นหาข้อมูลเชิงความหมาย ความหมายของฐานข้อมูลแบบกระจาย รูปแบบในการทำแฟรกเมนท์เทชั่นและงานวิจัยที่เกี่ยวกับการค้นหาข้อมูลเชิงความหมาย

2.1 การปรับปรุงคำสั่งคิวรี

Query Optimization เป็นการเลือกวิธีที่จะทำการดำเนินการกับคำสั่งให้มีประสิทธิภาพ โดยที่เป้าหมายของการทำ Query Optimization คือการเลือกวิธีการที่มีการใช้ทรัพยากรน้อยที่สุด โดยหลักการทั่วไปแล้วจะพยายามที่จะลดเวลาที่ใช้ในการดำเนินการกับคำสั่งคิวรีนั้นๆ ซึ่ง Query Optimization นั้นแบ่งออกเป็นประเภทได้ดังนี้

2.1.1 ตัวประมวลผลแบบอิงกฎ (Rule-Based Optimizer)

ตัวประมวลผลแบบอิงกฎจะมีการจัดลำดับของกฎตายตัวเรียงตามประสิทธิภาพ เมื่อมีคำสั่งคิวรีเข้ามาตัวประมวลผลก็จะทำการกำหนดค่าให้แต่ละเส้นทางของการประมวลผลโดยใช้การจัดลำดับของกฎและจะเลือกเส้นทางที่มีค่าน้อยที่สุด กฎตายตัวนี้ขึ้นอยู่กับลักษณะการเขียนคำสั่งของ SQL

ยกตัวอย่างเช่น พิจารณาคำสั่ง บนตาราง PropertyForRent และกำหนดให้มีอินเดกซ์บนคีย์หลัก(Primary Key) คือคอลัมน์ propertyNo, อินเดกซ์บนคอลัมน์ rooms และอินเดกซ์บนคอลัมน์ city

```
SELECT propertyNo
FROM PropertForRent
WHERE rooms > 7 and city = 'London';
```

ในกรณีนี้ ตัวประมวลผลแบบอิงกฎจะพิจารณาเส้นทางการประมวลผลดังนี้

เส้นทางการเข้าถึงแบบคอลัมน์เดี่ยวใช้อินเดกซ์วิ่งบนคอลัมน์ city จากเงื่อนไข WHERE (city = 'London') เส้นทางนี้มีค่าลำดับที่ได้ = 9 จากการประเมินผลแบบอิงกฎ

การค้นหาแบบไม่มีขอบเขตโดยใช้อินเดกซ์บนคอลัมน์ rooms จากเงื่อนไข WHERE (rooms > 7) เส้นทางนี้มีค่าลำดับที่ได้ = 11 จากการประเมินผลแบบอิงกฎ

การสแกนแบบทั้งตาราง ซึ่งใช้ได้กับทุกคำสั่งของ SQL มีค่าเส้นทางลำดับที่ 15

ถึงแม้ว่าจะมีอินเดกซ์บนคอลัมน์ propertyNo แต่ว่าคอลัมน์นี้ไม่ได้ปรากฏอยู่ในเงื่อนไข WHERE ดังนั้นจึงไม่ถูกพิจารณาโดยตัวประมวลผลแบบอิงกฎ ด้วยเหตุนี้ ตัวประมวลผลแบบอิงกฎจึงเลือกใช้อินเดกซ์บนคอลัมน์ city ซึ่งมีค่าน้อยที่สุดในการกำหนดค่าจากลำดับ

2.1.2 ตัวประมวลผลแบบอิงสถิติ (Cost-Based Optimizer)

ตัวประมวลผลแบบอิงสถิติถูกสร้างมาใหม่ให้ดีกว่าตัวประมวลผลแบบอิงกฎ มีการเก็บสถิติล่วงหน้า เพื่อหาเส้นทางการประมวลผลที่ดีที่สุดสำหรับแต่ละคำสั่งย่อย ตัวประมวลผลแบบอิงสถิติใช้ข้อมูลทางสถิติที่เก็บเป็นฐานข้อมูล เช่น ขนาดของตาราง, จำนวนของแถว (row), การกำหนดคีย์ (key), และอื่นๆ ซึ่งค่อนข้างจะดีกว่า การใช้กฎบังคับโดยการเก็บข้อมูลทางสถิตินั้นอาจจะเป็นหน้าที่ของผู้ใช้ แต่ถ้าผู้ใช้ไม่ได้สร้างข้อมูลทางสถิติไว้ ตัวประมวลผลแบบอิงสถิติก็จะไปใช้แบบอิงกฎแทน

2.1.3 ตัวประมวลคำถามเชิงความหมาย (Semantic Query Optimizer)

ตัวประมวลคำถามเชิงความหมาย ใช้ความรู้เชิงความหมายที่มีอยู่ นำมาแปลงคำสั่งที่เริ่มต้นให้อยู่ในรูปแบบต่างๆ ที่มีผลลัพธ์เหมือนเดิมแต่มีประสิทธิภาพมากขึ้น แล้วตัวประมวลคำถามเชิงความหมายจะเลือกรูปแบบคำสั่งที่ดีที่สุดนำไปประมวลผล

2.2 การค้นหาข้อมูลเชิงความหมาย

Semantic Query Optimization เป็นการใช้ความรู้ในเชิงความหมาย เพื่อทำการแปลงคำสั่งคิวรี ให้อยู่ในรูปแบบต่างๆ ซึ่งมีประสิทธิภาพในการประมวลผล (execute) ที่ต่างกัน แต่ผลลัพธ์ที่ได้ยังคงมีผลลัพธ์เหมือนกับ คำสั่งคิวรีที่ไม่ได้ทำการแปลง ความรู้ในเชิงความหมายนั้นอาจจะได้รับมาจากผู้ใช้งานของระบบโดยเป็นผู้ดูแลระบบ หรือได้รับมาจากระบบโดยมีการประมวลผลมาจากฐานข้อมูล

ในกระบวนการทำการแปลงรูปคำสั่งคิวรี (Query Reformulation) นั้น จำเป็นที่ต้องใช้ความรู้ในเชิงความหมาย ซึ่งประกอบด้วยกฎพื้นฐาน ซึ่งเปรียบเสมือนกฎข้อบังคับที่ผู้ดูแลระบบกำหนดให้

ยกตัวอย่าง เช่น

Department (dcode, dname, project, manager)

กฎพื้นฐาน :

1) dcode = 'ACCT' -> dname = 'Accounting'
2) dname = 'Accounting' -> dcode = 'ACCT'
3) dcode = 'PRSN' -> 1 <= project <=3

2.2.1 กระบวนการทำการแปลงรูปคำสั่งคิวรี มีอยู่ 3 รูปแบบหลักๆ คือ

2.2.1.1 การเพิ่มกฎ (Constraint Addition)

เป็นการนำแอททริบิว Attribute ที่เป็นอินเดกซ์เข้ามาเพิ่มในการคิวรี ยกตัวอย่าง เช่น

คำสั่งเดิม:
```
SELECT *
FROM student
WHERE entry = 94;
```

กฎ : entry = 94 -> regno >= 940000

ซึ่ง regno นั้นเป็นแอททริบิวที่เป็นอินเดกซ์

คำสั่งใหม่ :
```
SELECT *
FROM student
WHERE entry = 94 and regno >= 940000;
```

2.2.1.2 ลบเงื่อนไข (Constraint Deletion)

เป็นการนำเอาเงื่อนไขที่ไม่จำเป็นออกจากคิวรี ยกตัวอย่าง เช่น

คำสั่งเดิม:
```
SELECT *
FROM student
WHERE advisor = 100 and status = 'A'
```

กฎ : status = 'A'-> advisor = 100

คำสั่งใหม่ :
```
SELECT *
FROM student
WHERE status = 'A'
```

2.2.1.3 เงื่อนไขขัดแย้ง (Query Refutation)

เป็นการไม่ยอมรับคำสั่งนี้ โดยไม่ต้องส่งคำสั่งคิวรีเข้าไปให้ระบบจัดการดูแลฐานข้อมูลประมวลผล เมื่อคำสั่งคิวรีนั้นขัดแย้งกับกฎ ยกตัวอย่าง เช่น

คำสั่งเดิม:
```
SELECT *
FROM student
WHERE advisor = 105 and status = 'A'
```

กฎ : status = 'A'-> advisor = 100

คำสั่งใหม่ : **SELECT** Null

2.2.2 อัลกอริทึมของกระบวนการแปลงรูปแบบคำสั่งคิวรี (Query Reformulation Algorithm)

ทำขั้นตอนซ้ำ : 1 ถึง 4 ทุกๆ เงื่อนไข Ci ของคำสั่งคิวรีรวมทั้ง เงื่อนไขใหม่ที่เพิ่มขึ้น จนกระทั่ง ทุกๆ กฎตรวจสอบ

2.2.2.1 ทำการจับคู่เงื่อนไข Ci กับตัวก่อนหน้า(antecedent) A ของแต่ละ rule A -> B (A คือตัวก่อนหน้า ส่วน B คือตัวที่ตามมา)

2.2.2.2 ในการจับคู่นั้น ตรวจสอบดูว่าตัวที่ตามมา B สอดคล้องกับเงื่อนไขอื่นๆ ในคำสั่งคิวรีหรือ ไม่ ถ้าไม่สอดคล้อง ให้แจ้งขัดแย้งเลย ถ้าสอดคล้อง ก็ทำขั้นตอนที่ 3 ต่อไป

2.2.2.3 ทำการเพิ่มเงื่อนไขเข้าไปในเงื่อนไขของคำสั่ง (Query Condition) เพื่อทำเป็นคำสั่งคิวรี เงื่อนไข Cn ใหม่

2.2.2.4 ทำการสร้างความสัมพันธ์ที่ขึ้นต่อกัน (Cn,Ci) ที่ซึ่งเงื่อนไขใหม่ (Cn) นั้นขึ้นอยู่กับเงื่อนไขปัจจุบัน Ci ซึ่งจับคู่เหมาะกับกฎ

2.2.2.5 เอาค่าใช้จ่ายของแอททริบิว (Attribute Cost) มาสำหรับแต่ละเงื่อนไข (เช่น ชนิด(Type) ,ความยาว(length),หรือ ดัชนี(index))

2.2.2.6 ทำตามเงื่อนไขทั้งหมด : หาค่าใช้จ่าย (cost) ที่สูงที่สุดของเงื่อนไขChigh จากลิสต์(list) ของเงื่อนไข

ถ้ามีการขึ้นต่อกันของ (Chigh,Ci) ให้กำจัด Chigh ทิ้ง และแทนที่ทุกความสัมพันธ์ที่ขึ้นต่อกันที่เป็น(Cx,Chigh) ด้วย (Cx,Ci) เช่น ตัวอย่างการใช้วิธีการนี้ โดยใช้ตาราง Department และกฎดังนี้

กฎ :

1) dcode = 'ACCT'-> dname = 'Accounting'

2) dname = 'Accounting'-> dcode = 'ACCT'

3) dcode = 'PRSN' -> 1 <= project <=3

คำสั่งเดิม: **SELECT** *

FROM department

WHERE dname = 'Accounting' **and** manager = 'A01"

หลังจากทำค่าเริ่มต้นการทำซ้ำ แล้ว จะได้คำสั่งคิวรีดังนี้

SELECT *

from department

FROM dname = 'Accounting' **and** manager = 'A01'

WHERE dcode = 'ACCT"

ซึ่งกฎ 1 และ 2 จาก 3 กฎถูกนำมาใช้ในกระบวนการแปลงรูปแบบที่ขั้นตอนที่ 6 เงื่อนไขที่มีค่าใช้จ่ายสูงสุด คือเงื่อนไขของ dname = 'Accounting' เพราะแอททริบิว dname นั้นไม่ใช่อินเดกซ์ และมี ความยาวที่ยาวที่สุด เมื่อเปรียบเทียบกับ 3 แอททริบิวโดยที่ กฎ 1 นั้น ทำให้เกิดความสัมพันธ์ที่ขึ้นต่อกันระหว่าง(dname,dcode) จะเก็บ(hold)ไว้ และเงื่อนไขต่อมาก็ทำซ้ำกันและค่าเงื่อนไขที่มีค่าใช้จ่ายสูงสุดต่อมา คือ dcode = 'ACCT' ซึ่งยังคงถูกเก็บ เพราะมันไม่สอดคล้องกับความสัมพันธ์ที่ขึ้นต่อกันอื่นๆและค่าใช้จ่ายสูงรองลงมา คือ manager = 'A01' จะถูกเก็บไว้เหมือนกัน ดังนั้นคำสั่งคิวรีจะได้เป็น

SELECT *

FROM department

WHERE manager = 'A01' **and** dcode = 'ACCT"

วิธีการแปลงรูปแบบนี้อาจจะง่ายต่อการขยายรูปแบบของคำสั่งคิวรี ซึ่งเกี่ยวข้องกับการเชื่อมหลายตาราง ในกรณีนี้คำสั่งคิวรีจะถูกแบ่งออกเป็น คำสั่งย่อยๆ ซึ่งแต่ละคำสั่งย่อยๆ นี้จะกระทำกับคำสั่ง คิวรี เพียงความสัมพันธ์ของตารางเดียว แต่ละคำสั่งย่อยจะถูกแปลงโดยเป็นอิสระต่อกัน และคำสั่งย่อยๆ พวกนี้ที่ถูกแปลงแล้วจะถูกรวมกลับมาเป็นคำสั่งคิวรีเดิม ถึงแม้ว่าแต่ละข้อมูลที่ได้รับกลับมาจากแต่ละตารางจะมีความเกี่ยวข้องกัน เนื่องมาจากการเชื่อมโยงกันที่ไม่สามารถเปลี่ยนได้ แต่เวลาที่ใช้นั้นก็ลดลงอย่างเห็นได้ชัด นอกจากนี้ในการที่คำสั่งย่อยนั้นถูกปฏิเสธ ขั้นตอนของการแปลงคำสั่งคิวรี นั้นก็จะส่งค่า ว่างเปล่า(NULL) กลับมาโดยอัตโนมัติ เนื่องจากคำสั่งที่ถูกปฏิเสธนั้นไม่มีข้อมูลที่ตรงกับเงื่อนไขที่กล่าวมา

Semantic Query Optimization เป็นการใช้ความรู้ในเชิงความหมาย มาทำการแปลงคำสั่งคิวรีให้อยู่รูปแบบใหม่ที่มีประสิทธิภาพมากขึ้น ซึ่งความรู้ในเชิงความหมายที่นำมาใช้นั้นอาจจะมาจากผู้ใช้งานของระบบ หรืออาจจะมาจากระบบเองโดยการเก็บข้อมูลจากระบบ ซึ่งเทคนิคนี้ถูกนำมาใช้สร้างต้นแบบในหลายกรณีด้วยกัน ซึ่งในกรณีเหล่านี้เป็นการใช้ semantic query optimization บนระบบฐานข้อมูลแบบระบบรวมศูนย์กลาง และในบางกรณีก็ใช้บนระบบฐานข้อมูลแบบกระจาย

เทคนิคในการพัฒนา Semantic Query Optimization โดยใช้กฎข้อบังคับที่เกี่ยวข้อง เพื่อพัฒนาประสิทธิภาพในการเรียกดูข้อมูลดังนี้

1. Site Constraints

กรณีที่ ในแต่ละไซต์ของระบบการจัดการฐานข้อมูลแบบกระจายมีคุณสมบัติบางประการที่แตกต่างกัน เราสามารถนำเอา คุณสมบัติเหล่านั้น มากำหนดเป็น Integrity Constraint เฉพาะสำหรับแต่ละไซต์ เพื่อให้เป็นประโยชน์ในการทำ Query Optimization ได้ แอททริบิวที่ถูกกำหนดให้เป็น Site Constraint นั้นสามาถเป็นได้ทั้งชนิด ข้อมูลตัวอักษร ซึ่งค่าที่เป็นไปได้เมื่อมีการใช้คำสั่งคิวรีก็คือ กลุ่มของตัวอักษรและถ้าหากแอททริบิวเป็นชนิดตัวเลข ค่าที่เป็นไปได้ ก็คือค่า สูงสุดและต่ำสุดในช่วงของคุณสมบัติในแต่ละไซต์

2. Functional Dependencies

ความสัมพันธ์ที่ขึ้นต่อกัน เมื่อ constraint ที่กำหนดขึ้นเพื่อสร้างกฎที่แสดงถึงความสัมพันธ์ที่ขึ้นต่อกัน โดยนำเอากฎที่ได้ไปใช้ในการลดรูปหรือแปลงคำสั่งคิวรีให้ได้คำสั่งคิวรีที่ให้ผลลัพธ์ออกมาเหมือนกัน แต่ใช้เวลาประมวลผลน้อยกว่า หรือไม่ต้องส่งคำสั่งคิวรีนั้นไปประมวลผลเลย หรือบางกรณีอาจจะได้กฎที่ ได้มาจากการ derive กฎสองกฎ [4], [5], [10], [11]

3. Join Constraint

1.) Join Elimination

- แบบใช้ข้อบังคับแบบฟอเรนท์คีย์ (using foreign key constraints)

เมื่อมีการกำหนดข้อบังคับแบบฟอเรนท์คีย์ ระหว่างการเชื่อมกันของสองตาราง ระบบจะรับประกันว่าสำหรับแต่ละแถว (row) ในตารางลูก(child table) ที่มีคอลัมน์ที่เป็นฟอเรนท์คีย์ทั้งหมดที่ไม่มีค่าเป็นนัล(NULL) จะมีแถวในตารางหลัก (parent table) ที่มีค่าตรงกันในคอลัมน์ที่เป็นคีย์หลัก ดังนั้นถ้าจุดประสงค์ของการเชื่อมโยงตารางจึงเป็นเพียงเพื่อเช็คว่า ค่าในฟอเรนท์คีย์คอลัมน์ จะมีปรากฏอยู่ในคอลัมน์ที่เป็นคีย์หลักด้วยนั้น เราสามารถตัด การเชื่อมโยงนี้ทิ้งไปได้ (ตัวอย่างคือ สำหรับบางคำสั่งที่เกี่ยวข้องกับการเชื่อมโยงระหว่าง สองตารางสัมพันธ์กันผ่านทางข้อบังคับแบบฟอเรนท์คีย์

- แบบตรวจสอบแบบที่เป็นการเชื่อมตารางแบบว่างเปล่า(detection of empty join)

ในบางครั้งผลลัพธ์ของการเชื่อมตารางนั้นไม่มีค่าในฐานข้อมูล ถ้าเราพบผลลัพธ์แบบนี้ เราจะสามารถทำการปรับปรุงคำสั่งดั้งเดิมได้โดยการกำจัดการเชื่อมโยงตารางแบบว่างเปล่า

2.) Join Introduction

เป็นวิธีการที่ใช้ข้อได้เปรียบของการเชื่อมโยงตารางที่เพิ่มขึ้นมา ถ้าตารางนั้นมีความสัมพันธ์กับตารางเดิม(วิธีนี้จะน่าสนใจมากขึ้นถ้าแอททริบิวที่นำมาเชื่อมโยงนั้นเป็นอินเดกซ์)

3.) Predicate Elimination

จากกฎข้อบังคับ เราสามารถสรุปได้ว่า บางเพรดดิเคตนั้นเป็นจริงเสมอสำหรับการเรียก ดังนั้นเราสามารถตัดเพรดดิเคตที่ซ้ำซ้อนทิ้งได้ เพื่อเพิ่มประสิทธิภาพให้มากขึ้น ถ้าในเพรดดิเคตนั้นไม่มีอินเดกซ์อยู่

4.) Predicate Introduction

- แบบใช้อินเดกซ์ (using index introduction) ถ้าเพรดดิเคตใหม่ที่ได้จากการสรุป จากกฎข้อบังคับและจากบางเพรดดิเคตที่มีอยู่แล้วในคำสั่งคิวรีซึ่งมีอินเดกซ์อยู่ ในเพรดดิเคตนั้น จะทำการนำเพรดดิเคตนี้มาเพิ่มเข้ามาในคำสั่ง query เพื่อทำให้เพิ่มความเร็วของการประเมินผล

- แบบใช้ลดช่วงการค้นหา (scan reduction) คล้ายกับแบบใช้อินเดกซ์ ในบางครั้งเรานำบางเพรดดิเคตใหม่ลงในคำสั่งคิวรี ซึ่งเพรดดิเคตนี้ช่วยทำการลดช่วงในการค้นหาสำหรับบางข้อมูล ดังนั้นจึงเป็นการเพิ่มประสิทธิภาพ

5.) Detecting the Empty Answer Set

ถ้าเพรดดิเคตนั้นมีความขัดแย้งกับกฎข้อบังคับ ผลลัพธ์ของคำสั่งคิวรีจะไม่มีคำตอบ ดังนั้น ถ้าเราทำการตรวจสอบเงื่อนไขนี้ก่อน เราจะสามารถหลีกเลี่ยงการประเมินค่า คำสั่งคิวรีที่ไม่มีคำตอบได้ [3], [5], [6], [7], [8], [9]

2.3 ความหมายของฐานข้อมูลแบบกระจาย

ระบบฐานข้อมูลแบบกระจาย คือ การเก็บรวบรวมข้อมูลที่มีความสัมพันธ์ในทางตรรกะเข้าไว้เป็นระบบเดียวกัน แต่จะกระจายข้อมูลเก็บตามสถานที่ (site) ต่างๆ เชื่อมต่อกันอยู่บนระบบเครือข่ายคอมพิวเตอร์ แต่ละสถานที่สามารถประมวลผลได้อย่างเป็นอิสระ และมีการทำโปรแกรมประยุกต์แบบท้องถิ่น(Local Application) นอกจากนั้นจำเป็นต้องมีการทำงานร่วมกับสถานที่อื่นอย่างน้อย 1 โปรแกรมประยุกต์แบบรวม (Global Application)

หลักในการพิจารณาว่าเป็นระบบฐานแบบกระจายหรือไม่มีดังนี้

1. **Autonomous site** คือ แต่ละสถานที่สามารถดูแลจัดการฐานข้อมูลได้ด้วยตนเอง ซึ่งคำว่า Autonomous site จะต้องประกอบไปด้วย Autonomous Computer, Autonomous DBMS, Autonomous Database

2. **Interconnected via communication networks** คือ แต่ละสถานที่ต้องมีการเชื่อมต่อผ่านทางเครือข่ายคอมพิวเตอร์

3. **Local application** แต่ละสถานที่ต้องมีโปรแกรมประยุกต์ที่เรียกใช้ข้อมูลของสถานที่ที่โปรแกรมทำงานอยู่เท่านั้น

4. **Global application** แต่ละสถานที่ต้องมีโปรแกรมประยุกต์ที่มีการเรียกใช้ข้อมูลที่เก็บไว้ต่างสถานที่กัน

ปัจจัยสำคัญที่ใช้ตัดสินว่าระบบใดๆ นั้นควรพัฒนาให้เป็นระบบฐานข้อมูลแบบกระจาย หรือว่าระบบฐานข้อมูลแบบรวมดีนั้น ขึ้นอยู่กับปริมาณการใช้งานของโปรแกรมประยุกต์แบบท้องถิ่นยิ่งปริมาณการใช้งานมากยิ่งเหมาะสำหรับระบบฐานข้อมูลแบบกระจาย

โดยหลักการของระบบฐานข้อมูลแบบกระจายในมุมมองของผู้ใช้งานนั้น ผู้ใช้งานไม่ควรจะทราบว่าระบบที่ใช้งานอยู่เป็นระบบฐานข้อมูลแบบกระจาย ซึ่ง C.J. Date ได้กำหนดเป็นกฎ 12 ข้อดังต่อไปนี้

1. Local autonomy

คือการที่ แต่ละสถานที่ของฐานข้อมูลนั้นสามารถดูแลจัดการได้โดยผู้ดูแลแต่ละสถานที่เอง ไม่ต้องมีศูนย์กลางการควบคุมซึ่งทำให้ผู้ดูแลแต่ละเซิร์ฟเวอร์นั้นสามารถดูแล ปรับปรุง แก้ไขให้สามารถใช้งานได้อย่างมีประสิทธิภาพ

2. No reliance on a central site

นั่นคือไม่ต้องมีศูนย์กลางการควบคุมดูแลฐานข้อมูล ทุกสถานที่ของฐานข้อมูลนั้นมีความสำคัญเท่ากันซึ่งการทำ Query optimization, Transaction management, Naming service ไม่มีเซิร์ฟเวอร์ใดที่ทำหน้าที่เป็นศูนย์กลาง ทุกๆเซิร์ฟเวอร์จะต้องทำของแต่ละเซิร์ฟเวอร์เอง

3. Continuous operation

การทำการกระจายฐานข้อมูลไปในหลายๆ สถานที่จะช่วยเพิ่มการทำงานที่ต่อเนื่องของระบบ ซึ่งหากมีส่วนใดส่วนหนึ่งไม่สามารถทำงานได้ ระบบส่วนใหญ่ยังทำงานต่อไปได้ การทำให้ระบบทำงานได้อย่างต่อเนื่องนั้น สามารถทำใช้หลายวิธีร่วมกันได้ เช่น การทำ Replication เพื่อให้มีข้อมูลมากกว่าหนึ่งชุดอยู่ในระบบ หากเซิร์ฟเวอร์ที่เก็บข้อมูลชุดหนึ่งเสียหาย ก็ยังมีข้อมูลอีกชุดที่สามารถเรียกดูได้ ซึ่งการทำงานของ Replication จะกล่าวถึงต่อไปในภายหลัง การมีฐานข้อมูลมากกว่าหนึ่งที่ ก็เป็นการช่วยให้งานสามารถดำเนินการต่อเนื่องได้ โดยมีโอกาสน้อยมากที่ทุกๆ ฐานข้อมูล จะไม่สามารถใช้งานพร้อมกันได้หมด ต่างจากฐานข้อมูลแบบมีศูนย์กลางเพียงที่เดียว ถ้าเครื่องเซิร์ฟเวอร์มีปัญหาอาจจะส่งผลให้ต้องหยุดการให้บริการฐานข้อมูลไปด้วย

4. Location independence

คือ การที่ผู้ใช้งานสามารถเรียกดูข้อมูลที่ต้องการได้โดยไม่จำเป็นต้องทราบว่าข้อมูลนั้นถูกเก็บไว้ที่ ณ ฐานข้อมูลที่ใด รวมทั้งการย้ายสถานที่เก็บข้อมูลนั้นผู้ใช้งานไม่จำเป็นต้องเปลี่ยนแปลงการเรียกใช้งานด้วย

5. Fragmentation independence

Fragmentation independence คือ การที่ผู้ใช้งานไม่จำเป็นต้องทราบว่าข้อมูลที่เห็นนั้นในระดับกายภาพแล้วเก็บไว้ที่เดียวกันหรือมีการแบ่งแยกข้อมูลไปเก็บไว้ในหลายๆที่ รวมทั้งไม่จำเป็นต้องทราบว่าแต่ละส่วนของตารางที่ได้แบ่งแยกนั้นเก็บอยู่ที่ใดบ้าง ตัวอย่างเช่น ข้อมูลของพนักงานบริษัทซึ่งข้อมูลของพนังงานทั่วไปถูกเก็บไว้ในเซิร์ฟเวอร์ของสำนักงานในประเทศไทย

แต่ในขณะที่พนักงานที่มีตำแหน่งเป็นผู้บริหารจะถูกเก็บอยู่ในเซิร์ฟเวอร์ของบริษัทแม่ที่อยู่ในต่างประเทศ เป็นต้น โดยรายละเอียดของการทำ Fragmentation จะอธิบายเพิ่มเติมต่อไป

6. Replication independence

Replication คือการที่มีข้อมูลมากกว่าหนึ่งชุดถูกเก็บไว้ในระบบ มีประโยชน์คือ สามารถกระจายการเข้าถึงของข้อมูล ทำให้ผู้ใช้งานเข้าถึงข้อมูลได้หลายทาง รวมทั้งยังเป็นการทำให้ระบบทำงานได้อย่างต่อเนื่องคือ หากเซิร์ฟเวอร์หนึ่งมีปัญหาอาจจะยังมีข้อมูลเดียวกันอยู่อีกที่หนึ่ง ทำให้การดำเนินงานของระบบไม่สะดุด ส่วน Replication independence นั้น คือการที่ผู้ใช้งานไม่จำเป็นต้องรู้ว่าข้อมูลที่ใช้งานอยู่นั้นถูกคัดลอกไปไว้ที่ใดบ้าง การปรับปรุง เปลี่ยนแปลง แก้ไขนั้นสามารถทำได้เสมือนว่าข้อมูลนั้นมีอยู่เพียงชุดเดียวในระบบ

7. Distributed query processing

มีการวิเคราะห์การค้นหาแนวทางการได้มาซึ่งข้อมูลที่ดีที่สุด โดยในฐานข้อมูลแบบกระจายนั้นจำเป็นที่จะต้องคำนึงถึงการประมวลข้อมูลที่ได้ผลลัพธ์ตามต้องการ เนื่องจากต้องมีการส่งข้อมูลข้ามระบบเครือข่าย และถ้าข้อมูลที่ส่งนั้นเป็นข้อมูลที่มีขนาดใหญ่กว่าที่อยู่ที่ฝั่งรับ จะเป็นการทำให้ใช้เวลาในการส่งมากกว่าการส่งข้อมูลฝั่งน้อยกว่าไปประมวลผลอีกฝั่งหนึ่ง

8. Distributed transaction management (update processing)

มีการจัดการ Transaction ที่มีการทำงานที่ยุ่งเกี่ยวกับฐานข้อมูลหลายๆสถานที่ต้องมีการทำงานอย่างสมบูรณ์คือ ถ้าสามารถทำงานได้ทุกที่ที่มีการแก้ไขต้องได้รับการแก้ไขอย่างถูกต้องหมด แต่ถ้ามีฐานข้อมูลที่ใดไม่สามารถทำงานได้อย่างถูกต้อง จำเป็นต้องมีการ Roll Back กลับเพื่อให้เสมือนว่าการแก้ไขข้อมูลไม่เคยเกิดขึ้น เป็นการรักษาความถูกต้องของข้อมูล

9. Hardware independence

โดยปกติแล้ว DBMS ที่ติดตั้งนั้นต้องไม่ขึ้นอยู่กับอุปกรณ์ฮาร์ดแวร์ การเลือกฮาร์ดแวร์สำหรับเครื่องเซิร์ฟเวอร์นั้นจะคำนึงถึงปริมาณข้อมูลและผู้ใช้งานเป็นหลัก เพราะยิ่งมีผู้ใช้งานพร้อมกันเป็นจำนวนมากเครื่องที่ให้บริการก็จะเป็นที่จะต้องมีประสิทธิภาพสูงพอที่จะรองรับได้ไม่เช่นนั้นอาจจะเกิดปัญหาในการบริการได้

10. Operating system independence

คือการเป็นอิสระจากระบบปฏิบัติการ ซึ่งในปัจจุบันนั้นผู้ผลิตซอฟต์แวร์ฐานข้อมูลนั้นส่วนใหญ่จะพัฒนาออกมารองรับให้สามารถใช้งานได้ในหลายๆ ระบบปฏิบัติการ และการทำฐานข้อมูลแบบกระจายนั้นควรจะไม่ขึ้นอยู่กับระบบปฏิบัติการแบบใดแบบหนึ่งเท่านั้น ควรสามารถทำการเชื่อมต่อกันได้แม้ว่าจะเป็นคนละระบบปฏิบัติการรวมถึงมีฮาร์ดแวร์ที่แตกต่างกันด้วย

11. Network independence

Network independence เป็นการที่ระบบไม่จำเป็นต้องขึ้นอยู่กับลักษณะของการเชื่อมต่อระหว่างกัน ขอเพียงสามารถส่งข้อมูลระหว่างฐานข้อมูลที่อยู่คนละที่กันได้ก็เพียงพอ ไม่จำเป็นต้องมีการเชื่อมต่อที่เหมือนกัน แต่อย่างไรก็ตามควรมีระบบเน็ตเวิร์คที่รองรับปริมาณการเชื่อมต่อได้ไม่น้อยเกินไปเพื่อป้องกันไม่ให้เกิดปัญหาในการทำงาน

12. DBMS independence

คือการทำฐานข้อมูลแบบกระจายนั้นไม่ได้จำกัดอยู่เพียงการทำกระจายบนซอฟต์แวร์ฐานข้อมูลที่ผลิตมาจากผู้ผลิตรายเดียวกัน ควรจะสนับสนุนการทำการกระจายฐานข้อมูลทั้งแบบบนซอฟต์แวร์ฐานข้อมูลเดียวกัน (Homogeneous) หรือเป็นแบบใช้หลายๆซอฟต์แวร์ฐานข้อมูลร่วมกัน (Heterogeneous)

2.3.1 ประเภทของ Distributed Database System

จะสามารถแบ่งออกได้เป็น 2 ประเภท ด้วยกันคือ Homogeneous distributed database system และ Heterogeneous distributed database system

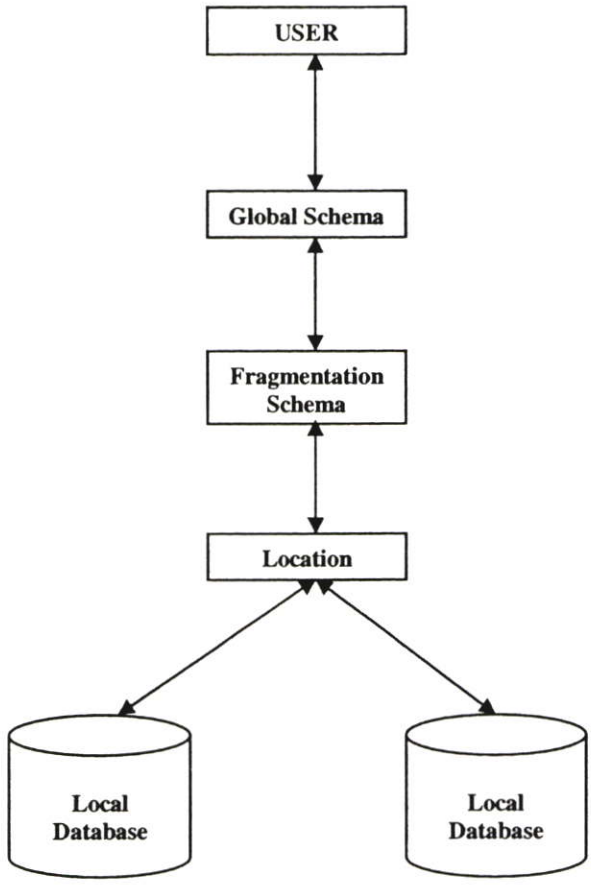
Homogeneous distributed database system ทุกสถานที่จะใช้ซอฟต์แวร์ระบบจัดการฐานข้อมูล (DBMS) ชนิดเดียวกัน ซึ่งอาจจะไม่ได้อยู่บนระบบปฏิบัติการเดียวกันก็ได้

Heterogeneous distributed database system มีอย่างน้อย 1 สถานที่ ที่ใช้ซอฟต์แวร์ระบบจัดการฐานข้อมูล (DBMS) ต่างจากสถานที่อื่น ซึ่งหลักในการสร้าง heterogeneous distributed database system ต้องอาศัยมาตรฐานที่เรียกว่า gateway protocols ในที่นี้ gateway protocols จะหมายถึง API (Application Programming Interface) ซึ่งทำหน้าที่เชื่อมต่อระหว่าง DBMS และโปรแกรมต่าง ๆ เช่น ODBCและ JDBC เป็นต้น

2.3.2 Distributed Database Architecture

สถาปัตยกรรมของฐานข้อมูลแบบกระจายที่แบ่งเป็นระดับต่างๆ แสดงดังรูปนี้ ในทางปฏิบัติแล้วไม่จำเป็นที่จะต้องทำได้ถึงระดับสูงสุดซึ่งก็คือ Global schema แล้วจึงจะถือว่าเป็นระบบฐานข้อมูลแบบกระจาย อาจจะทำได้เพียงในระดับ Allocation schema หรือ Fragmentation schema ก็ได้ และเนื่องจาก Data Model ที่ใช้ในระบบฐานข้อมูลแบบกระจายอาจเป็นได้ทั้ง Relational Model หรือ Object Model เห็นได้จากรูปแสดงสถาปัตยกรรมของระบบฐานข้อมูลแบบกระจาย ซึ่งใน 3 ระดับบนแสดงให้เห็นว่าไม่ขึ้นอยู่กับ Data Model ที่ใช้

ต่อไปนี้จะเป็นการอธิบายความสามารถของสถาปัตยกรรมในแต่ละระดับ



รูปที่ 2.1 แสดงสถาปัตยกรรมระดับต่างๆ ของฐานข้อมูลแบบกระจาย

2.3.2.1 Global schema

เป็นส่วนที่อยู่บนสุดของสถาปัตยกรรมการอ้างถึงข้อมูลของระบบฐานข้อมูลแบบกระจาย ซึ่ง Global schema จะอ้างถึงข้อมูลที่สถานที่ (site) ต่างๆ เสมือนกับว่าข้อมูลทั้งหมดไม่มีการกระจายเลย ซึ่งจะเหมือนกับการอ้างถึงข้อมูลในฐานข้อมูลแบบรวม (Centralize database) และ relation ที่ถูกอ้างถึงในสถาปัตยกรรมระดับนี้ จะเรียกว่า Global relation

2.3.2.2 Fragmentation schema

สถาปัตยกรรมในระดับนี้เป็นการเข้าถึงข้อมูลโดย อ้างถึงบางส่วนของ Global relation ที่ถูกแบ่งออกเป็นส่วนๆ โดยจะเรียกแต่ละส่วนว่า fragment ซึ่งวิธีการแบ่ง Global relation ออกเป็น fragment นี้ สามารถแบ่งได้หลายวิธี เช่น แบ่งตามแนวนอน (Horizontal Fragmentation) คือแยกทัปเปิล (tuples) ออกเป็นกลุ่มๆ หรือแบ่งตามแนวตั้ง (Vertical Fragmentation) คือแยกแอททริบิว (Attributes) ออกเป็นกลุ่มๆ และ fragment ที่แบ่งออกมานี้ สามารถที่จะนำกลับมารวมเข้าเป็นความสัมพันธ์โดยรวมได้ดังเดิม ด้วยการทำ UNION และ JOIN ตามลำดับ อย่างไรก็ตามการทำ

Fragmentation นี้ สามารถที่จะทำทั้งสองวิธีผสมกัน (Mixed Fragmentation) ได้ ทั้งนี้จะต้องสามารถนำกลับมารวมเป็นความสัมพันธ์โดยรวมได้ดังเดิม

2.3.2.3 Allocation schema

สถาปัตยกรรมในระดับนี้จะกล่าวถึง Fragment ที่แบ่งมาจาก Global relation นั้น ว่าจะถูกกำหนดสถานที่ (site) ที่จะเก็บข้อมูล fragment ไว้ที่ไหน ซึ่งในแต่ละ fragment นี้ อาจจะมีสถานที่ (site) ในเก็บข้อมูลมากกว่าหนึ่งแห่งได้ ทั้งนี้ขึ้นอยู่กับการตัดสินใจของผู้ออกแบบระบบฐานข้อมูลว่าจะยอมให้มีการซ้ำซ้อนของข้อมูลมากน้อยเพียงใด และสอดคล้องกับการใช้งานหรือไม่

2.3.2.4 Local mapping schema

ในสถาปัตยกรรมทั้ง 3 ระดับด้านบนที่ได้กล่าวไปแล้วนั้น ฐานข้อมูลแบบกระจายในแต่ละสถานที่ (site) จะไม่ขึ้นอยู่กับ Data Model ที่ใช้ เพราะฉะนั้นแล้วในสถาปัตยกรรมในระดับนี้เองจะเป็นระดับที่มีการทำให้ DBMS ในแต่ละสถานที่ (site) สามารถเข้าใจ Data Model ที่ใช้ในสถาปัตยกรรมระดับที่สูงขึ้นไปได้

2.4 Distribution Transparency

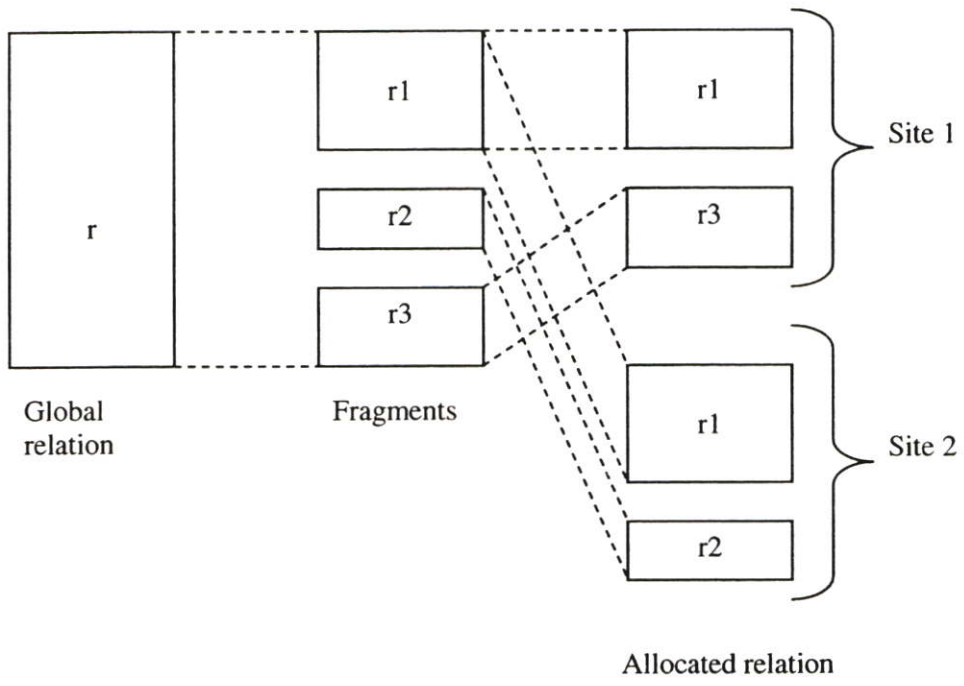
จากสถาปัตยกรรมของระบบฐานข้อมูลแบบกระจายในระดับต่างๆ ทำให้สามารถกำหนดระดับ Transparency ได้ดังต่อไปนี้

1. Fragmentation transparency เป็นระดับ transparency สูงสุด ซึ่งในระดับนี้ ผู้ใช้งานหรือ โปรแกรมเมอร์ จะเห็นในระดับ Global relation คือ ไม่ทราบว่าแบ่งเป็นกี่ fragment แต่ละ fragment ถูกเก็บไว้ที่ไหน รวมทั้งมีการทำซ้ำที่ไหนบ้าง

2. Location transparency เป็นระดับที่ ผู้ใช้งาน หรือ โปรแกรมเมอร์ จะเห็นในระดับ fragment ทำงานบน fragment แต่จะไม่ทราบว่า fragment นั้นถูกเก็บไว้ที่สถานที่ (site) ไหน

3. Local mapping transparency เป็นระดับที่ผู้ใช้งาน หรือโปรแกรมเมอร์ จะทราบว่าเป็น fragment ที่ใช้งานถูกเก็บไว้ที่สถานที่ (site) ไหน มีการทำซ้ำที่ไหนบ้าง แต่จะไม่ทราบว่า ที่สถานที่ (site) นั้น ใช้ DBMS ยี่ห้อไหน Data model แบบใด

4. No transparency เป็นระดับที่ผู้ใช้งานหรือ โปรแกรมเมอร์ จำเป็นต้องทราบว่าในสถานที่ (site) นั้นใช้ DBMS ยี่ห้อไหน Data model แบบใด



รูปที่ 2.2 แสดงการพิจารณาระดับสถาปัตยกรรมและระดับ transparency

2.5 Distributed Data Storage

ในการจัดเก็บข้อมูลในระบบฐานข้อมูลแบบกระจายมีอยู่หลายวิธีด้วยกันคือ

1. Fragmentation เป็นการแบ่ง relation ออกเป็นหลาย ๆ ส่วน และจัดเก็บแต่ละส่วนไว้ต่างสถานที่กัน

2. Replication เป็นการทำสำเนาของ relationไว้หลาย ๆ สำเนา และแต่ละสำเนาจะถูกเก็บไว้ต่างสถานที่กัน

นอกจากนั้น เรายังสามารถทำ Fragmentation ร่วมกับ Replication ได้ คือ สามารถทำการแบ่ง Relation ออกเป็นหลาย ๆ ส่วน และแต่ละส่วนก็จะมีการจัดทำสำเนาไว้ด้วย

2.6 รูปแบบในการทำ Fragmentation

การแบ่งความสัมพันธ์โดยรวม (Global relation) ออกเป็น fragment นี้ สามารถแบ่งออกได้ 2 วิธีหลักๆ ด้วยกันคือ แบ่งตามแนวนอน (Horizontal Fragmentation) คือแยกทัปเปิล (tuples) ออกเป็นกลุ่มๆ หรือแบ่งตามแนวตั้ง (Vertical Fragmentation) คือแยกแอททริบิว (Attributes)

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

ออกเป็นกลุ่มๆ ซึ่งในการแบ่งความสัมพันธ์โดยรวม (Global relation) ออกเป็น fragment ต้องเป็นไปตามกฎเกณฑ์ดังนี้

1. Completeness condition คือ ทุกข้อมูลของความสัมพันธ์โดยรวม (Global relation) ต้องพบได้ใน fragment ที่ได้แบ่งไว้ด้วย เช่น จะต้องไม่เกิดกรณีที่มีข้อมูลอยู่ในความสัมพันธ์โดยรวมแต่ไม่อยู่ใน fragment ใดๆ

2. Reconstruction condition คือ ต้องสามารถสร้างความสัมพันธ์โดยรวม (Global relation) ขึ้นมาได้จาก fragment ที่ได้แบ่งเอาไว้ เพราะว่าในระบบฐานข้อมูลแบบกระจายนั้น แต่ละสถานที่ (site) จะเก็บ fragment ที่ได้แบ่งเอาไว้ ส่วนความสัมพันธ์โดยรวม (Global relation) จะต้องถูกสร้างขึ้นใหม่เมื่อมีความจำเป็นจาก fragment ในแต่ละสถานที่ (site)

3. Disjointness condition คือ แต่ละ fragment ต้องแยกออกจากกัน สำหรับเงื่อนไขนี้ถือเป็นหลักการสำหรับกรณี การแบ่งส่วนย่อยตามแนวนอน (Horizontal Fragmentation) ขณะที่กรณีการแบ่งส่วนย่อยตามแนวตั้ง (Vertical Fragmentation) จะยอมให้มีการละเมิดเงื่อนไขในข้อนี้ได้

สามารถอธิบายรายละเอียดเพิ่มเติมในการแบ่งส่วนย่อย (Fragmentation) ประเภทต่างๆ ได้ดังนี้

2.6.1 การแบ่งตามแนวนอน (Horizontal Fragmentation)

การแบ่งตามแนวนอนคือ การแยกทัปเปิล (tuples) ออกเป็นกลุ่มๆ ซึ่งแต่ละกลุ่มสามารถแบ่งได้ตามลักษณะทางภูมิศาสตร์ อย่างเช่น ภาค จังหวัด ประเทศ เป็นต้น ซึ่งในการทำ Fragmentation ตามแนวนอนนี้สามารถทำได้โดยการใช้คำสั่งของภาษา Relation algebra คือใช้ select บน Global relation ตัวอย่างเช่น กำหนดให้ Global relation คือ

SUPPLIER (SNUM, NAME, CITY)

สามารถทำการ Fragmentation ตามแนวนอนได้โดยการ select ข้อมูลใน SUPPLIER ตามค่า แอททริบิวของ CITY ซึ่งก็คือค่า "SF" สำหรับ SUPPLIER1 และ "LA" สำหรับ SUPPLIER2 ได้ดังนี้

SUPPLIER1 = $\mathbf{SL}_{\text{CITY= "SF"}}$ SUPPIER

เป็นข้อมูลของ SUPPLIER เฉพาะในเมือง San Francisco

SUPPLIER2 = $\mathbf{SL}_{\text{CITY= "LA"}}$ SUPPIER

เป็นข้อมูลของ SUPPLIER เฉพาะในเมือง Los Angeles

75112

กรณี Completeness condition จะเป็นจริงได้ถ้าค่าแอททริบิวของ CITY มีแค่ "SF" และ "LA" เท่านั้น

กรณี Reconstruction condition ในการสร้าง SUPPLIER Global relation ขึ้นมาใหม่ สามารถทำได้โดยการใช้ UNION operation ดังนี้

SUPPLIER = SUPPLIER1 **UN** SUPPLIER2

กรณี Disjointness condition จะเห็นได้ว่าข้อมูลจะถูกแยกตาม CITY คือ San Francisco และ Los Angeles เท่านั้น ต้องไม่มีทัปเปิล (tuple) ไหนที่ค่า CITY เป็นทั้ง San Francisco และ Los Angeles

2.6.2 การแบ่งตามการสืบทอดแนวนอน (Derived Horizontal Fragmentation)

มีบางกรณีที่การแบ่งตามแนวนอน (Horizontal Fragmentation) ไม่สามารถที่จะแบ่งตามแอททริบิวของตัวเองได้ จำเป็นต้องสืบทอดมาจาก Relation อื่น ยกตัวอย่างเช่นมี Relation ดังนี้

SUPPLY (SNUM, PNUM, DEPTNUM, QUAN)

Relation นี้จะต้องทำการแบ่ง fragment ตามค่า CITY ของ SUPPLIER relation อย่างไรก็ตาม CITY ไม่ได้เป็นแอททริบิวของ SUPPLY relation แต่เป็นแอททริบิวของ SUPPLIER relation ดังนั้นเราจำเป็นต้องใช้ SEMI-JOIN operation ในการบอกว่าทัปเปิล (tuple) ไหนใน SUPPLY ที่ตรงกับ SUPPLIER ที่ได้แบ่งตาม CITY จะทำให้ Derived fragmentation ของ SUPPLY สามารถกำหนดได้ดังนี้

SUPPLY1 = SUPPLY **SJ** $_{\text{SNUM = SNUM}}$ SUPPLIER1

SUPPLY2 = SUPPLY **SJ** $_{\text{SNUM = SNUM}}$ SUPPLIER2

ซึ่ง SNUM เป็นค่าของ supplier number ถูกใช้เป็นค่าในการทำ SEMI-JOIN ระหว่าง SUPPLY relation กับ SUPPLIER relation

กรณี Completeness condition มีความจำเป็นที่ต้องไม่มีค่า SNUM ค่าใดที่มีอยู่ใน SUPPLY relation แต่ไม่ได้อยู่ใน SUPPLIER relation จึงจะตรงกับเงื่อนไขนี้

กรณี Reconstruction condition การสร้าง Global relation SUPPLY ขึ้นมาใหม่สามารถใช้ UNION operation ได้เหมือนกับการแบ่งตามแนวนอนของ SUPPLIER relation

กรณี Disjointness condition คือ จะต้องไม่มีทัปเปิลใดใน SUPPLY relation ที่ตรงกับ 2 fragment ที่ต่างกัน

2.6.3 การแบ่งตามแนวตั้ง (Vertical Fragmentation)

คือการแบ่งแอททริบิว (Attributes) ของ Global relation ออกเป็นกลุ่มๆ อธิบายได้ง่ายๆ ก็คือการแบ่งตามคอลัมน์นั่นเอง มักจะขึ้นอยู่กับ Application ว่าต้องการใช้แอททริบิวหรือคอลัมน์ไหนบ้าง แต่ละ fragment จะมาจากการใช้ Projection operation เลือกเอาคอลัมน์ที่ต้องการ นำมาจัดกลุ่ม ตัวอย่างสามารถพิจารณาจาก Global relation ดังนี้

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

การแบ่ง fragment ตามแนวตั้งของ relation นี้ คือ

$$EMP1 = \mathbf{PJ}_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$$

$$EMP2 = \mathbf{PJ}_{EMPNUM,\ NAME,\ SAL,\ TAX}\ EMP$$

กรณี Completeness condition คือค่าของแอททริบิวหรือคอลัมน์ของ Global relation จะต้องมีอยู่อย่างน้อยใน 1 fragment ที่ได้แบ่งเอาไว้

กรณี Reconstruction condition การสร้าง Global relation EMP ขึ้นมาใหม่จะใช้การ JOIN ระหว่าง fragment ที่ได้แบ่งเอาไว้ ซึ่งตัวอย่างนี้จะใช้ค่า EMPNUM เป็น key ในการ JOIN

$$EMP = EMP1\ \mathbf{JN}_{EMPNUM\ =\ EMPNUM}\ \mathbf{PJ}_{EMPNUM,\ SAL,\ TAX}\ EMP2$$

จะเห็นได้ว่าเมื่อทำการ JOIN กันแล้ว มีค่าแอททริบิวหรือว่าคอลัมน์ที่ซ้ำกันซึ่งก็คือ EMPNUM และ NAME สามารถกำจัดค่าที่ซ้ำออกไปโดยใช้ Projection operation จัดกลุ่มเอาเฉพาะค่าที่ต้องการเท่านั้น

กรณี Disjointness condition เพราะว่าในจำเป็นที่ต้องมี key ที่อยู่ในทุกๆ fragment เพื่อใช้ในการสร้าง Global relation ขึ้นมาใหม่ เพราะฉะนั้นในการทำ Vertical Fragmentation จึงยอมให้มีการละเมิดกฎในข้อนี้ได้

2.6.4 การแบ่งแบบผสม (Mixed Fragmentation)

นอกจากการทำ Fragmentation แบบต่างๆ ที่ได้กล่าวมาแล้ว ยังสามารถนำการทำ Fragmentation แบบต่างๆ มาผสมกันได้ ตัวอย่างเช่นมี Global relation ดังนี้

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

สามารถทำ Fragmentation แบบผสมได้โดย การแบ่งตามแนวตั้ง (Vertical fragmentation) ตามด้วยแบ่งตามแนวนอน (Horizontal Fragmentation) โดยใช้ค่า DEPTNUM เป็นตัวแบ่ง จะได้

$$EMP1 = \mathbf{SL}_{DEPTNUM \quad 10} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

$$EMP2 = \mathbf{SL}_{10 < DEPTNUM \quad 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

$$EMP3 = \mathbf{SL}_{DEPTNUM > 20} \mathbf{PJ}_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

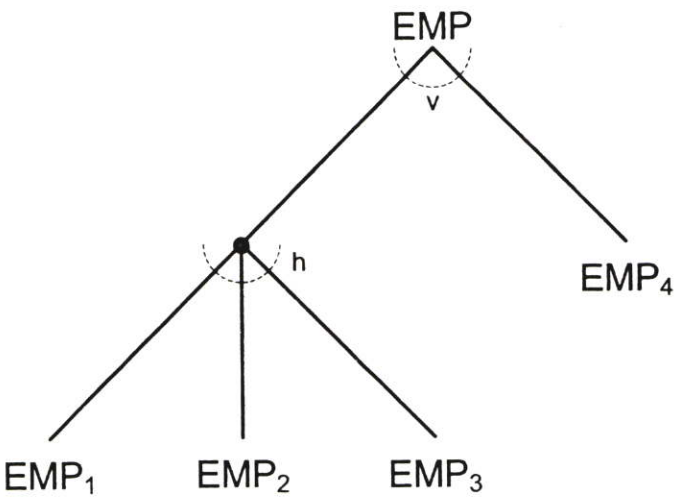
$$EMP4 = \mathbf{PJ}_{EMPNUM, NAME, SAL, TAX} EMP$$

กรณี Completeness condition ใช้การพิจารณาตามหลักของการแบ่งตามแนวนอน (Horizontal fragmentation) และแบ่งตามแนวตั้ง (Vertical fragmentation) ประกอบกัน

กรณี Reconstruction condition สำหรับการสร้าง Global relation ขึ้นใหม่สามารถทำได้โดยย้อนกลับขึ้นไปหา Global relation ตัวอย่างนี้จะทำการ UNION fragment ที่แบ่งตามแนวนอนก่อนแล้วจึงทำการ JOIN fragment ที่ทำการแบ่งตามแนวตั้ง แสดงเป็นลำดับคำสั่งได้ดังนี้

$$EMP = \mathbf{UN}(EMP1, EMP2, EMP3) \mathbf{JN}_{EMPNUM = EMPNUM} \mathbf{PJ}_{EMPNUM, SAL, TAX} EMP4$$

กรณี Disjointness condition ก็พิจารณาตามหลักการของของการแบ่งตามแนวนอน (Horizontal fragmentation) และแบ่งตามแนวตั้ง (Vertical fragmentation) ประกอบกัน การทำ Fragmentation แบบผสมสามารถแสดงได้เป็น Tree ซึ่งจะถูกเรียกว่า Fragmentation tree จากตัวอย่างแสดง Tree ได้ดังนี้



รูปที่ 2.3 แสดง tree ที่เกิดจากการทำ fragmentation แบบผสม

2.7 Data Replication

การที่จะบอกว่า relation r หรือ fragment ของ relation r มีการทำ replication ก็ต่อเมื่อมีการทำสำเนาของ relation r หรือ fragment นี้ เก็บไว้ตามสถานที่ต่างๆ โดยที่ไม่จำเป็นจะต้องเก็บไว้ในทุกๆ สถานที่ แต่สำหรับกรณีที่มีการทำสำเนาไว้ที่ทุกสถานที่ จะเรียกว่า full replication

ซึ่งการทำ Replication นี้มีทั้งข้อดีและข้อเสีย สามารถอธิบายได้ดังต่อไปนี้

ข้อดี

Availability ถ้าสถานที่กำลังทำงานอยู่กับ relation r เกิดหยุดการทำงานลง ระบบสามารถที่จะทำงานต่อไปได้โดยไปดึงข้อมูลที่สถานที่อื่น โดยไม่ต้องคำนึงถึงสถานที่ ที่หยุดทำงานไป

Increased parallelism ในกรณีที่การดำเนินการกับข้อมูลส่วนใหญ่ของ relation r เป็นการอ่านข้อมูลจาก relation เมื่อเรามีข้อมูลของ relation r อยู่ในหลาย ๆ สถานที่ เราก็สามารถที่จะทำการดึงข้อมูลมาพร้อมกันได้ อีกทั้งจากการที่มีสำเนาของ relation อยู่หลาย ๆ สถานที่ โอกาสที่จะพบข้อมูลในสถานที่ ที่ต้องการจะทำ transaction ก็มีโอกาสมากกว่า ดังนั้นด้วยวิธีการนี้จะช่วยลดปริมาณข้อมูลที่จะส่งผ่านระหว่างสถานที่ได้

ข้อเสีย

Increased overhead on update ด้วยวิธีการ replication นี้ สำเนาข้อมูลของ relation r ในแต่ละสถานที่จะต้องเหมือนกันทุก ๆ สถานที่ ไม่เช่นนั้นแล้วการประมวลผลกับ relation r อาจจะเกิดความผิดพลาดขึ้นได้ ดังนั้นเมื่อไรก็ตามที่ relation r ถูกแก้ไข ระบบจะต้องทำการแก้ไขข้อมูล relation r ให้ครบทุกสถานที่ ที่มี relation r อยู่ ยกตัวอย่างเช่น ในระบบธนาคารซึ่งมีการทำสำเนา

ของข้อมูลบัญชีไว้หลาย ๆ สถานที่ เมื่อมีการปรับปรุงยอดบัญชี ก็จะต้องทำการปรับยอดบัญชีให้เท่ากันในทุก ๆ สถานที่

โดยทั่วไปแล้วการทำสำเนาข้อมูลจะช่วยเพิ่มประสิทธิภาพในการอ่านข้อมูล และช่วยเพิ่มความสะดวกในการสืบค้นข้อมูลของ transaction แบบอ่านอย่างเดียว อย่างไรก็ตาม transaction แบบที่มีการปรับปรุงข้อมูลจะต้องมีการดำเนินการที่เพิ่มมากขึ้น ซึ่งการควบคุมการปรับปรุงข้อมูลพร้อมๆ กันในหลาย ๆ สถานที่ค่อนข้างจะยุ่งยากกว่าระบบฐานข้อมูลแบบรวม

2.7.1 ชนิดของการทำ Replication

2.7.1.1 Synchronous Replication คือการที่ข้อมูลที่มีการคัดลอกไว้หลายชุดและกระจายไปอยู่ตามสถานที่ต่างๆนั้น จะเป็นข้อมูลที่มีการแก้ไขหรือเปลี่ยนแปลงพร้อมกันเสมอ คือ เมื่อมีการแก้ไขเปลี่ยนแปลงข้อมูล ณ ที่ใดๆ ต้องมีการแก้ไขข้อมูลชุดอื่นๆ ด้วย โดยถ้ามีข้อมูลชุดใดชุดหนึ่งไม่สามารถแก้ไขได้ ก็จะถือว่าการแก้ไขไม่สามารถทำได้ การทำ Synchronous Replication นั้นเพื่อเป็นการรักษา Integrity Constraints ซึ่งจะทำให้ไม่ว่าเราดึงข้อมูลมาจากข้อมูลชุดใดๆ จะได้ข้อมูลที่เหมือนกันและข้อมูลที่ใหม่ที่สุดเสมอ แต่อย่างไรก็ตามการทำ Replication แบบนี้นั้น จำเป็นที่จะต้องมีระบบเครือข่ายคอมพิวเตอร์ที่เชื่อถือได้อย่างมาก เพราะว่าการทำแบบนี้เมื่อมีการเพิ่ม หรือแก้ไขข้อมูลจำเป็นที่จะต้องแก้ไขข้อมูลทุกชุดเท่าที่มีอยู่ ซึ่งเป็นการเพิ่มภาระหน้าที่ของการทำงานแต่ละงานและทำให้การ COMMIT ช้าลงไปอีกด้วย โดยการทำ Synchronous Replication นั้นจำเป็นที่จะต้องมีการใช้งาน 2-Phrase Commit Protocols (2PC) เข้ามาช่วยในการที่จะ Commit Transaction เพื่อให้สามารถแก้ไขข้อมูลทุกๆที่พร้อมกัน และการทำ Synchronous Replication นั้นถ้าหากมีเซิร์ฟเวอร์ใดที่เก็บข้อมูลชุดนี้อยู่ด้วย ไม่สามารถทำงานได้นั้น การแก้ไขข้อมูลนั้นไม่สามารถทำได้เลยซึ่งอาจจะเป็นปัญหาได้ ดังนั้นจึงต้องระมัดระวังในการวางแผนการทำ Replication

2.7.1.2 Asynchronous Replication เป็นการทำ Replication ที่ข้อมูลชุดหลักนั้นมีเพียงชุดเดียว ส่วนข้อมูลที่คัดลอกไปนั้น ทำเพื่อเพิ่มช่องทางการเข้าถึงข้อมูล โดยข้อมูลทั้งสองชุดนั้นอาจไม่เหมือนกันตลอดเวลา เนื่องจากสถานที่ที่เก็บข้อมูลไว้สามารถทำการ commit โดยอิสระ การทำ Replication แบบนี้เหมาะกับกรณีข้อมูลไม่ต้องเปลี่ยนแปลงบ่อยนัก แบ่งตามการทำงานได้ 3 ลักษณะด้วยกัน คือ

1. PUSH คือ สถานที่หลัก (primary site) จะส่ง transaction ไปยังสถานที่ที่เกี่ยวข้อง (remote site) โดยที่สถานที่ที่เกี่ยวข้องไม่ต้องทำการร้องขอ อาจจะส่งเป็นข้อมูลที่มีการเปลี่ยนแปลงแล้วก็ได้

2. PULL คือ สถานที่ที่เกี่ยวข้อง (remote site) จะต้องทำการร้องขอการเปลี่ยนแปลงจากสถานที่หลัก (primary site) ถ้าไม่มีการร้องขอสถานที่หลักก็ไม่ทำการส่งข้อมูลที่มีการเปลี่ยนแปลง

3. SNAPSHOT คือ สถานที่หลัก (primary site) จะส่งการเปลี่ยนแปลงไปยังสถานที่ที่เกี่ยวข้อง (remote site) ตามช่วงเวลาที่ได้กำหนดไว้

2.8 Distributed Transaction

ในการทำงานของ application ของระบบฐานข้อมูลแบบรวมจะมีการทำ transaction เป็นส่วนหนึ่งอยู่ด้วยเสมอ เช่นกันในระบบฐานข้อมูลแบบกระจาย เนื่องจากมีการทำงานของ application ดังนั้นจึงมี transaction เข้ามาเกี่ยวข้องด้วยเช่นกัน ในระบบฐานข้อมูลแบบกระจายนี้จะมีการทำ transaction อยู่ 2 ประเภทด้วยกัน คือ

Local transaction จะดำเนินการกับข้อมูลในฐานข้อมูลของตัวเองเท่านั้น

Global transaction จะทำการเข้าถึงข้อมูลและปรับปรุงข้อมูลในหลาย ๆ ฐานข้อมูลตามสถานที่ต่างๆ

สำหรับในกรณีของ global transaction การรักษาคุณสมบัติ ACID (Atomicity, Consistency, Isolation, Durability) ของการทำ transaction แบบกระจายจะมีการดำเนินการที่ซับซ้อนมากขึ้น เนื่องจากแต่ละสถานที่จะมีส่วนร่วมในการทำ transaction ซึ่งความล้มเหลวที่เกิดขึ้นในสถานที่เหล่านี้ หรือความล้มเหลวจากการสื่อสารกันระหว่างสถานที่ อาจทำให้การทำ transaction ผิดพลาดได้

2.8.1 โครงสร้างของระบบ

ในแต่ละสถานที่จะมีตัวจัดการ transaction (local transaction manager) เป็นของตนเอง ซึ่งทำหน้าที่ควบคุมให้การทำ transactionให้มีคุณสมบัติ ACIDโดยที่ตัวจัดการ transaction แต่ละตัวก็จะร่วมกันในการทำ global transaction โดยหลักการแล้ว เมื่อทำการเริ่มต้นการทำงานของ transaction ณ สถานที่ไหน ตัวจัดการ transaction ของสถานที่นั้นจะทำหน้าที่เป็นตัวประสานงาน transaction (transaction coordinator) ส่วนสถานที่อื่นจะเป็นผู้เกี่ยวข้อง (participant) อธิบายการทำงานเพิ่มเติมได้ ดังนี้

2.8.1.1 ตัวจัดการ transaction (transaction manager) จะคอยควบคุมดูแลการดำเนินการของ transaction ต่างๆ ที่มีการเข้าถึงข้อมูลในสถานที่ของตนเอง แต่ละ transaction เป็นได้ทั้ง local transaction หรือเป็นส่วนหนึ่งของ global transaction ก็ได้ อธิบายหน้าที่การทำงานเพิ่มเติมได้ดังนี้

2.8.1.1.1 การจัดการกับ log เพื่อจุดประสงค์ในการฟื้นคืนสภาพข้อมูล (recovery)

2.8.1.1.2 มีส่วนร่วมในการทำ concurrency control กับสถานที่อื่นๆ เพื่อร่วมทำ concurrent execution ของ transaction ในสถานที่ของตนเอง

2.8.1.2 ตัวประสานงาน transaction (transaction coordinator) จะคอยประสานการทำงานของ transaction ต่าง ๆ ทั้ง local transaction และ global transaction อธิบายหน้าที่การทำงานเพิ่มเติมได้ดังนี้

2.8.1.2.3 เริ่มต้นการทำงานของ transaction ที่เกี่ยวข้อง

2.8.1.2.3 แบ่ง transaction ออกเป็น transaction ย่อย และส่งไปยังสถานที่ ที่เหมาะสม

2.8.1.2.3 ประสานการดำเนินการของ transaction ว่า ผลของการทำ transaction นั้น ทำสำเร็จในทุก ๆ สถานที่ หรือยกเลิกในทุก ๆ สถานที่หรือไม่ คือ ดูแลการทำ Two-phase commit

2.8.2 Commit Protocols

เพื่อให้การทำ transaction บนระบบฐานข้อมูลแบบกระจายมีคุณสมบัติ Atomicity คือ transaction T ต้อง commit ในทุก ๆ สถานที่ หรือยกเลิกการทำ transaction ในทุก ๆ สถานที่ transaction coordinator ของ transaction T ต้องมีการทำ commit protocol ซึ่ง commit protocol ที่มีการใช้งานกันอย่างแพร่หลายก็คือ two-phase commit protocol (2PC) และ three-phase commit protocol(3PC) ซึ่งจะช่วยหลีกเลี่ยงข้อเสียของ 2PC แต่การดำเนินการจะมีความซับซ้อนมากกว่า สำหรับในงานนี้จะพูดถึงเฉพาะในกรณีของ two-phase commit protocol(2PC) เท่านั้น

Two-Phase Commit (2PC)

ให้ T เป็น transaction เริ่มต้นที่สถานที่ S_i และให้ ตัวประสานงาน transaction ที่สถานที่ S_i เป็น C_i สามารถอธิบายการทำงานเป็น 2 เฟส ดังนี้

Phase 1: Obtaining a decision

ฝั่ง Transaction coordinator

ตัวประสานงาน transaction Ci จะถาม participants รอบๆ ว่าพร้อมที่จะ commit รึยังโดย

- C_i ทำการเพิ่มเรคอร์ด <prepare T> ลงไปใน log ไฟล์ บน stable storage
- จากนั้น C_i จะส่งสัญญาณ prepare T ไปให้กับทุก ๆ สถานที่ ที่ประมวลผล transaction T

ฝั่ง Participant

เมื่อสถานที่อื่น ๆ ได้รับสัญญาณ ตัวจัดการ transaction จะต้องตอบกลับไปหาตัวประสานงาน transaction ว่าจะทำการ commit transaction ได้หรือไม่

- ถ้าไม่ได้จะเพิ่มเรคอร์ด <no T> ลงไปใน log ไฟล์และส่งสัญญาณ abort T กลับไปที่ C_i
- ถ้าได้จะเพิ่มเรคอร์ด <ready T> ลงไปใน log ไฟล์ และตัวจัดการ transaction จะส่งสัญญาณ ready T กลับไปที่ C_i

Phase 2: Recording the decision

ฝั่ง Transaction coordinator

- C_i จะ commit transaction T ก็ต่อเมื่อได้รับสัญญาณ ready T จากทุก ๆ สถานที่ ที่มีส่วนร่วมในการทำ transaction กรณีนอกเหนือจากนี้ transaction T จะต้องถูกยกเลิก และจะต้องทำการเพิ่มเรคอร์ด <commit T> หรือ <abort T> ลงไปใน log ไฟล์ ถ้า transaction T สามารถ commit ได้ หรือ ทำการยกเลิก transaction
- ต่อมา C_i ก็จะส่งสัญญาณ commit T หรือ abort T ไปให้กับทุกๆ สถานที่ ที่ร่วมกันทำ transaction

ฝั่ง Participant

- เมื่อแต่ละสถานที่ได้รับสัญญาณ commit T หรือ abort T ที่ถูกส่งมาจาก Ci ก็จะทำการบันทึกเรคอร์ดนี้ลงไปใน log ไฟล์

จะเห็นว่าสัญญาณ ready T มีความสำคัญมาก เนื่องจากสถานที่ต่าง ๆ ที่ทำ transaction T จะต้องส่งสัญญาณ ready T กลับไปให้ตัวประสานงาน transaction เพื่อบอกกับตัวประสานงาน transaction ว่าพร้อมที่จะ commit แล้ว ซึ่งสัญญาณนี้จะมีผลต่อการทำ commit หรือ abort ของ transaction จากลักษณะดังกล่าวอาจเป็นไปได้ว่าเมื่อสถานที่หนึ่งได้ทำการส่งสัญญาณ ready T ไปให้ตัวประสานงาน transaction แล้ว อาจเกิดความล้มเหลวของสถานที่ได้ ซึ่งจากโปรโตคอล 2PC เมื่อตัวประสานงาน transaction ได้รับสัญญาณ ready T หรือ abort T ครบ ก็จะส่งสัญญาณ commit T หรือ abort T ไปให้กับ สถานที่ ที่ร่วมทำ transaction เท่านั้น โดยไม่ได้สนใจว่าสถานที่ต่าง ๆ เหล่านั้นสามารถ commit หรือ abort ตามสัญญาณที่ส่งไปได้หรือไม่ ดังนั้นในการดำเนินการโปรโตคอล 2PC จะต้องมีการส่งสัญญาณ acknowledge T กลับมาที่ตัวประสานงาน transaction เพื่อเป็นการบอกว่าได้ดำเนินการ ในระยะที่สองเสร็จเรียบร้อยแล้ว เมื่อตัวประสานงาน transaction ได้รับ acknowledge T ครบจากทุก ๆ สถานที่แล้ว ก็จะเพิ่มเรคอร์ด <complete T> เข้าไปใน log ไฟล์ของ C_i

2.9 เปรียบเทียบระหว่าง Distributed Database System กับ Centralized Database System

ในการเปรียบเทียบระหว่างระบบฐานข้อมูลแบบกระจายกับระบบฐานข้อมูลแบบรวม จะยึดเอาความสามารถของฐานข้อมูลแบบรวมเป็นหลัก แล้วพิจารณาว่าฐานข้อมูลแบบกระจายมีความสามารถแค่ไหน ซึ่งหัวข้อที่นำมาพิจารณาประกอบไปด้วย Centralized control, Data independence Reduction of redundancy, Complex physical structure for efficient access, Integrity, Recovery, Concurrency control, Privacy และ Security

Centralized control คือ ความสามารถในการควบคุมจัดการข้อมูลโดยรวมทั้งหมด ซึ่งหน้าที่นี้เป็นของ ผู้ดูแลระบบฐานข้อมูล database administrator (DBA)

ในระบบฐานข้อมูลแบบกระจาย แนวความคิดนี้จะไม่ได้ให้ความสำคัญมากนัก โดยทั่วไปแล้ว ในระบบฐานข้อมูลแบบกระจายจะมีระดับการควบคุมอยู่ โดยที่ผู้ดูแลระบบฐานข้อมูลโดยรวม (Global database administrator) มีหน้าที่ในการดูแลฐานข้อมูลทั้งหมด ส่วนผู้ดูแลระบบฐานข้อมูลระดับท้องถิ่น (Local database administrator) มีหน้าที่ในการดูแลระบบฐานข้อมูล ณ สถานที่ต่างๆ ซึ่งจุดนี้จะต้องเน้นว่าผู้ดูแลระบบส่วนท้องถิ่นต้องมีความสำคัญสูงกว่าผู้ดูแลระบบโดยรวม

Data independence หมายถึง การเปลี่ยนแปลงที่อยู่ของข้อมูลจะต้องไม่ส่งผลกระทบกับโปรแกรม

ในระบบฐานข้อมูลแบบกระจาย Data independence ก็มีความสำคัญเช่นกัน และได้มีข้อกำหนดใหม่ขึ้นมา ซึ่งเรียกว่า Distribution transparency ซึ่งหมายถึง โปรแกรมสามารถถูกเขียนขึ้นมาเหมือนกับว่าไม่ใช่ฐานข้อมูลแบบกระจาย ดังนั้นการเคลื่อนย้ายข้อมูลจะไม่ส่งผลกับความถูกต้องของโปรแกรม อย่างไรก็อาจส่งผลกับความเร็วในการประมวลผลโปรแกรม

Reduction of redundancy ในระบบฐานข้อมูลแบบรวม ความซ้ำซ้อนของข้อมูลจะถูกจำกัดด้วยเหตุผล 2 กรณี ด้วยกันคือ กรณีความไม่สอดคล้องกันของชุดสำเนาของข้อมูล ซึ่งสามารถหลีกเลี่ยงโดยการทำสำเนาเพียงแค่ 1 ชุดเท่านั้น ส่วนอีกกรณีนั้นเพื่อการประหยัดเนื้อที่ที่ใช้เก็บข้อมูล

ในระบบฐานข้อมูลแบบกระจาย ความซ้ำซ้อนเป็นสิ่งที่ต้องการ จากเหตุผลที่ว่า เพื่อทำให้โปรแกรมสามารถเข้าถึงข้อมูลได้เลยเมื่อต้องการ โดยการทำสำเนาของข้อมูลไว้ในทุกๆ สถานที่ และถ้ามีกรณีที่สถานที่ที่เก็บข้อมูลไว้อยู่ไม่สามารถทำงานได้ การทำงานของโปรแกรมก็ยังคงดำเนินต่อไปได้เพราะว่าได้มีการทำซ้ำข้อมูลเอาไว้แล้ว แต่การทำซ้ำของข้อมูลก็มีข้อเสียเช่นกันคือ มีเมื่อมีการเปลี่ยนแปลงของข้อมูลที่สถานที่ใด ต้องตามเปลี่ยนแปลงข้อมูล ที่สถานที่อื่นๆ ด้วย

Complex physical structure and efficient access ในฐานข้อมูลแบบกระจายนั้น เมื่อมีการต้องการประมวลผลข้อมูลที่ซับซ้อนซึ่งมีขนาดใหญ่และอยู่คนละที่กันนั้น เช่นการเรียกดูข้อมูลที่เกิดจากการรวมกันของข้อมูลซึ่งต้องมีการวนรอบในการทำงาน และถ้าการวนรอบในการทำงานเหล่านี้มีข้อมูลอยู่ต่างสถานที่กัน จะทำให้ต้องมีการเชื่อมต่อเพื่อส่งข้อมูลไปตรวจสอบไปมา ทำให้มีปัญหาในเรื่องเวลาของการประมวลผล ซึ่งจะต่างจากฐานข้อมูลที่อยู่ที่เดียวซึ่งจะเป็นการส่งข้อมูลภายในซึ่งมีความเร็วมากกว่าการส่งข้อมูลข้ามระบบเครือข่ายมาก

Integrity, recovery, and concurrency control ความถูกต้องของข้อมูล การฟื้นฟูข้อมูล และการควบคุมการสภาวะการทำงาน ปัญหาเหล่านี้เกิดมาจากการทำ transaction ซึ่งจะต้องดูแลให้มีคุณสมบัติ Atomic unit

ในระบบฐานข้อมูลแบบกระจาย ในการดูแลให้การทำ transaction มีคุณลักษณะ Atomic unit กรณีของความถูกต้องของข้อมูล (Integrity) การควบคุมสภาวะการทำงาน (concurrency control) รวมทั้งการฟื้นฟูข้อมูล (recovery) จะมีความยุ่งยากมากกว่าในระบบฐานข้อมูลแบบรวม เพราะจำเป็นที่จะต้องทำการดูแลควบคุมในหลายๆ สถานที่

Privacy and security ในระบบฐานข้อมูลแบบรวม ผู้แลระบบฐานข้อมูลจะมีสิทธิในการควบคุมจัดการระบบฐานข้อมูลได้ทั้งหมด ทำให้มั่นใจได้ว่าเฉพาะผู้ที่ได้รับอนุญาตเท่านั้นที่สามารถเข้าถึงข้อมูลได้ อย่างไรก็ตามหากเป็นระบบฐานข้อมูลแบบรวมที่ไม่มีการควบคุมที่ดีพออาจจะไม่ปลอดภัยจากผู้ไม่ประสงค์ดี

ส่วนในระบบฐานข้อมูลแบบกระจาย สำหรับผู้ดูแลระบบฐานระดับท้องถิ่นจะประสบปัญหาเดียวกับกรณีของระบบฐานข้อมูลแบบรวม อย่างไรก็ตามข้อกำหนดที่พิเศษเพิ่มเติมของระบบฐานข้อมูลแบบกระจาย ข้อแรกคือ ผู้ดูแลระบบฐานข้อมูลระดับท้องถิ่นจะมีสิทธิเหนือกว่าผู้ดูแลระบบฐานข้อมูลโดยรวมในการควบคุมจัดการฐานข้อมูล ณ สถานที่นั้นๆ ข้อสองก็คือ ปัญหาเรื่องความปลอดภัย เพราะว่าระบบฐานข้อมูลแบบกระจายมีการเชื่อมต่อผ่านเครือข่ายคอมพิวเตอร์ซึ่งอาจเป็นจุดอ่อนในเรื่องความปลอดภัยของข้อมูล

2.10 งานวิจัยที่เกี่ยวข้อง

ปัจจุบันการวิจัยเรื่องการค้นหาข้อมูลเชิงความหมาย ได้มีการพัฒนาอย่างต่อเนื่อง บนฐานข้อมูลแบบต่างๆ โดยนักวิจัยได้พัฒนาการค้นหาข้อมูลเชิงความหมายใช้วิธีต่างๆ ดังนี้

B.G.T. Lowden และ J. Robinson, K.Y. Lim, [5] Semantic query optimization ได้ใช้ semantic knowledge เพื่อแปลงสภาพ query ให้เปลี่ยนเป็นรูปแบบที่สามารถทำงานได้มีประสิทธิภาพมากขึ้น แต่ยังคงให้ผลลัพธ์คงเดิมเช่นเดียวกับ query ต้นฉบับ โดย semantic knowledge นี้สามารถใช้ได้โดยผู้ใช้หรือระบบงานโดยตรง โดยบทความนี้ ได้อธิบาย ARDOR

semantic query optimizer โดยใช้ความสามารถทางด้าน automatic rule derivation ซึ่งสามารถแสดงให้เห็นถึงการลดลงของเวลาการทำงาน

John Cardiff [11] ได้นำเสนอการออกแบบ semantic query optimizer สำหรับระบบฐานข้อมูล heterogeneous database management system โดยยึดพื้นฐานจากการขยายขอบข่ายของการติดตั้งในระบบส่วนกลาง ซึ่งจะเน้นหนักในการค้นหาให้คำสั่ง query ที่ "ใกล้เคียงประสิทธิภาพสูงสุด" ที่จะสามารถแปลงรูปแบบของคำสั่งได้อย่างรวดเร็ว โดยที่ semantic ของระบบปฏิบัติการได้ถูกนำเสนอในรูปของ relational database และ algorithm ซึ่งกำหนดให้ใช้งานกับฐานข้อมูลเพื่อทำหน้าที่ในการแปลงสภาพ query

Chun-Nan Hsu และ Craig A. Knoblock [10] วิธี Semantic Query Optimization (SQD) ในการเพิ่มประสิทธิภาพ query ให้สูงสุด บนระบบฐานข้อมูล heterogeneous multidatbase systems วิธีการดังกล่าวนี้ทำให้มี global optimization สำหรับแบบแผน query พร้อมๆ กับ local optimization สำหรับ subquery ซึ่งเรียกข้อมูลมาจากฐานข้อมูลต้นฉบับแต่ละแห่ง ส่วนสำคัญของ local optimization algorithm ของบทความนี้คือการที่พิสูจน์เงื่อนไขที่จำเป็นและสำคัญเพียงพอเพื่อลดปริมาณความเชื่อมโยงที่ไม่สำคัญใน conjunctive query ของ arbitrary join topology ออกไป วิธีการนี้ช่วยให้การเพิ่มประสิทธิภาพ ใช้เงื่อนไขความสัมพันธ์ได้อย่างกว้างขวางมากขึ้น อันทำให้เกิดขอบข่ายของ optimization ที่กว้างขวางขึ้นกว่างานเดิมๆ ใน SQO นอกจากนั้น local optimization algorithm ยังประกอบด้วยโครงสร้างข้อมูลใหม่ ที่เรียกว่า AND-OR implication graphs

2.11 สรุป

การค้นหาข้อมูล ซึ่งเป็นการเรียกหาข้อมูลที่ไซต์ของตัวเองโดยไม่ทราบว่า ข้อมูลที่ต้องการนั้น เก็บอยู่ที่ไหนบ้าง ซึ่งการเรียกข้อมูลที่ต้องการนั่นอาจจะมาจากหลายไซต์และหลายๆ รีเลชั่น บางครั้งข้อมูลที่ต้องการค้นหานั้นอาจจะไม่มีอยู่ในฐานข้อมูล ถ้าหากต้องส่งคำสั่งคิวรีออกไปในไซต์ต่างๆ เพื่อทำการค้นหาข้อมูลอาจจะเป็นการเพิ่มข้อมูลออกไปในระบบเครือข่าย ทั้งที่ข้อมูลนั้นไม่มีอยู่ในฐานข้อมูล ดังนั้น ระบบออฟติไมเซอร์นี้ จึงเป็นตัวช่วยในการคัดกรองคำสั่งคิวรีเหล่านี้เมื่อเข้ากับเงื่อนไขก็ทำการปรับปรุง หรือ ยกเลิกคำสั่งนั้นได้ทันที ซึ่งจะช่วยประหยัดเวลา และทรัพยากรในระบบเครือข่ายได้

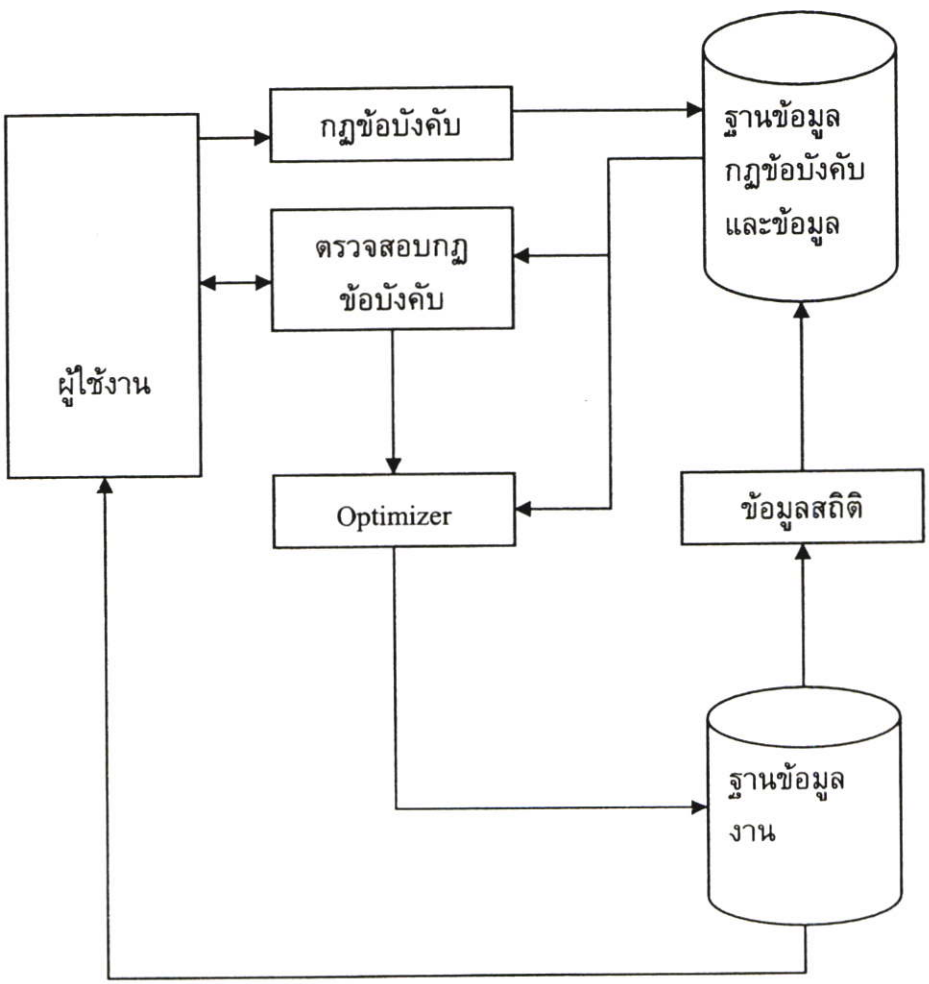
นอกจากนี้การจัดการข้อมูลของฐานข้อมูลแบบกระจายในระดับแฟรกเมนท์และระดับออโลเคชั่น จะเกิดความซับซ้อนเมื่อมีการใช้คำสั่งคิวรีในระดับโกลบอล ซึ่งจะต้องมีการแตกคำสั่งคิวรีออกเป็นหลายๆ คำสั่งเพื่อส่งไปยังแต่ละแฟรกเมนท์

บทที่ 3
การออกแบบระบบตัดสินใจค้นหาข้อมูลเชิงความหมาย

ในบทนี้จะกล่าวถึงหลักการและวิธีการดำเนินงานวิจัย ของงานวิจัยที่ผู้วิจัยนำเสนอ ซึ่งในส่วนนี้จะนำเสนอถึงวิธีการในการดำเนินงานวิจัย เป็นต้น

3.1 ขั้นตอนการทำงานของระบบ

ในบทความนี้นำเสนอการออฟติไมซ์โดยใช้หลักการของ semantic เข้ามาร่วมกับ การใช้สถิติเพื่อช่วยในการตัดสินใจ และ ช่วยการจัดการในฐานข้อมูลแบบกระจาย เมื่อมีการเรียก หรือจัดการข้อมูลในระดับโกลบอล



รูปที่ 3.1 การทำงานของระบบค้นหาข้อมูลเชิงความหมาย

การทำงานของระบบ จะมีการรับข้อมูลของกฎข้อบังคับที่ผู้ดูแลระบบใส่เข้าไปในฐานข้อมูลข้อบังคับและมีการเก็บข้อมูลสถิติไว้ในฐานข้อมูลด้วย เมื่อผู้ใช้ระบบส่งคำสั่งคิวรีเข้ามา คำสั่งคิวรีจะถูกนำไปตรวจสอบกับกฎข้อบังคับจากฐานข้อมูลกฎข้อบังคับ ถ้าพบว่าขัดแย้งกับกฎข้อบังคับก็จะแจ้งให้ผู้ใช้ระบบทราบ แต่ถ้าไม่ขัดแย้งกับกฎ ก็จะเข้าไปในส่วนของ Optimizer เพื่อทำการปรับปรุงรูปแบบคำสั่ง โดยนำกฎข้อบังคับจากจากฐานข้อมูลกฎข้อบังคับมาใช้ เมื่อแปลงคำสั่งแล้วจะนำคำสั่งคิวรีที่ได้มาทำการปรับปรุงโดยพิจารณาจากสถิติที่เก็บข้อมูลไว้ก่อนแล้ว จากนั้น นำคำสั่งคิวรีที่ได้ ไปทำการประมวลผลและจับเวลาเพื่อเปรียบเทียบเวลาที่ใช้ในการประมวลผล และแสดงผลลัพธ์ที่ได้

3.1.1 Semantic query optimization

Semantic query optimization เป็นโปรแกรมประยุกต์การใช้ความรู้ด้าน semantic มาใช้แปลงรูป คิวรีอย่างง่ายเป็นคิวรีที่มีประสิทธิภาพสูงกว่าคิวรีต้นแบบ คลังความรู้ semantic อาจจะได้มาจากโปรแกรมที่ยูสเซอร์สร้างขึ้นมาหรือจากข้อมูลของระบบ ในบทความนี้ใช้หลักการเหล่านี้ในการปรับปรุงคิวรีของระบบ

3.1.1.1. Site Constraint

ในกรณีที่แต่ละโลคอลไซต์มีเอกลักษณ์ที่แตกต่างกัน เราสามารถนำเอาเอกลักษณ์เหล่านั้นมาสร้างเป็นกฎข้อบังคับ เพื่อกำหนดลักษณะเฉพาะของแต่ละโลคอลไซต์นั้นๆ ตัวอย่างเช่น การค้นหานักศึกษาที่ศึกษาอยู่ใน วิทยาเขตสายช่างอุตสาหกรรม ซึ่งไม่ควรจะต้องไปค้นหาข้อมูลในทุกวิทยาเขต แต่ไปค้นหาข้อมูลนักศึกษา เฉพาะ วิทยาเขตที่เปิดการเรียนการสอนสายช่างอุตสาหกรรมเท่านั้น

3.1.1.2. Functional Dependencies

ความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependencies) สามารถช่วยลดคิวรีที่มีความซับซ้อนลงให้กลายเป็นคิวรีที่มีความง่ายขึ้นได้ เงื่อนไขที่ไม่จำเป็นทำให้คิวรีซับซ้อนและต้องการการลดรูปคิวรีจาก optimizer ของระบบมากขึ้น ความสัมพันธ์ที่ขึ้นต่อกันนี้จะช่วยในการค้นหาเส้นทางที่ดีกว่าจาก เงื่อนไขที่อ้างถึงโดยใช้อินเดกซ์ และแทนที่ ด้วยเงื่อนไขอื่นที่อ้างอิงไปยังอินเดกซ์ ในกรณีนี้เงื่อนไขที่อ้างอิงโดยใช้อินเดกซ์ถูกใช้งานแทนที่เงื่อนไขที่ไม่ได้อ้างอิงอินเดกซ์

3.1.1.3. Join Constraint

1. Join Elimination

- แบบใช้ข้อบังคับแบบฟอเรนท์คีย์ (using foreign key constraints)

เมื่อมีการกำหนดข้อบังคับแบบฟอเรนท์คีย์ ระหว่างการเชื่อมกันของสองตาราง ระบบจะรับประกันว่าสำหรับแต่ละแถว (row) ในตารางลูก (child table) ที่มีคอลัมน์ที่เป็นฟอเรนท์คีย์ ทั้งหมดที่ไม่มีค่าเป็นนัล(NULL) จะมีแถวในตารางหลัก (parent table) ที่มีค่าตรงกันในคอลัมน์

ที่เป็นคีย์หลัก ดังนั้นถ้าจุดประสงค์ของการเชื่อมโยงตารางจึงเป็นเพียงเพื่อเช็คว่า ค่าในฟอเรนท์คีย์คอลัมน์ จะมีปรากฏอยู่ในคอลัมน์ที่เป็นคีย์หลัก ด้วยนั้น เราสามารถตัด การเชื่อมโยงนี้ทิ้งไปได้ (ตัวอย่างคือ สำหรับบางคำสั่งที่เกี่ยวข้องกับการเชื่อมโยงระหว่าง สองตารางสัมพันธ์กันผ่านทางข้อบังคับแบบฟอเรนท์คีย์)

- แบบตรวจสอบแบบที่เป็นการเชื่อมตารางแบบว่างเปล่า (detection of empty join)

ในบางครั้งผลลัพธ์ของการเชื่อมตารางนั้นไม่มีค่าในฐานข้อมูล ถ้าเราพบผลลัพธ์แบบนี้ เราจะสามารถทำการปรับปรุงคำสั่งดั้งเดิมได้โดยการกำจัดการเชื่อมโยงตารางแบบว่างเปล่า

2. Join Introduction

เป็นวิธีการที่ใช้ข้อได้เปรียบของการเชื่อมโยงตารางที่เพิ่มขึ้นมา ถ้าตารางนั้นมีความ สัมพันธ์กับตารางเดิม(วิธีนี้จะน่าสนใจมากขึ้นถ้าแอททริบิวที่นำมาเชื่อมโยงนั้นเป็นอินเดกซ์)

3. Predicate Elimination

จากกฎข้อบังคับ เราสามารถสรุปได้ว่า บางเพรดดิเคตนั้นเป็นจริงเสมอสำหรับการเรียก ดังนั้นเราสามารถตัดเพรดดิเคตที่ซ้ำซ้อนทิ้งได้ เพื่อเพิ่มประสิทธิภาพให้มากขึ้น ถ้าใน เพรดดิเคตนั้นไม่มีอินเดกซ์อยู่

4. Predicate Introduction

- แบบใช้อินเดกซ์ (using index introduction) ถ้าเพรดดิเคตใหม่ที่ได้จากการสรุปจากกฎข้อบังคับและจากบางเพรดดิเคตที่มีอยู่แล้วในคำสั่งคิวรีซึ่งมีอินเดกซ์อยู่ ในเพรดดิเคตนั้น จะทำการนำเพรดดิเคตนี้มาเพิ่มเข้ามาในคำสั่งคิวรี เพื่อทำให้เพิ่มความเร็วของการประเมินผล

- แบบใช้ลดช่วงการค้นหา (scan reduction) คล้ายกับแบบใช้อินเดกซ์ ในบางครั้งเรานำบางเพรดดิเคตใหม่ลงในคำสั่งคิวรี ซึ่งเพรดดิเคตนี้ช่วยทำการลดช่วงในการค้นหาสำหรับบางข้อมูล ดังนั้นจึงเป็นการเพิ่มประสิทธิภาพ

5. Detecting the Empty Answer Set

ถ้าเพรดดิเคตนั้นมีความขัดแย้งกับกฎข้อบังคับ ผลลัพธ์ของคำสั่งคิวรีจะไม่มีคำตอบ ดังนั้นถ้าเราทำการตรวจสอบเงื่อนไขนี้ก่อน เราจะสามารถหลีกเลี่ยงการประเมินค่า คำสั่งคิวรีที่ไม่มีคำตอบได้

3.1.1.4. Global Statistics

ในการประมวลผลแบบอิงสถิตินั้น ตัวประมวลผลจะตัดสินใจว่าแผนในการดำเนินการใดมีประสิทธิภาพมากที่สุด โดยจะพิจารณาจากเส้นทางที่เป็นไปได้ และปัจจัยด้านข้อมูลที่เกี่ยวข้องกับสถิติ โดยปกติแล้วจุดมุ่งหมายในการประมวลผลแบบอิงสถิตินั้นคือ เพื่อให้มีผลลัพธ์ ที่ดีที่สุด หรือให้มีการใช้ทรัพยากรที่จำเป็นน้อยที่สุด สถิติในฐานข้อมูลนั้นมีทั้งขนาดของตาราง จำนวน

ของค่าที่ต่างกันในแต่ละแอททริบิว จำนวนบล็อคและอินเดกซ์ นี่เป็นการจัดเก็บสถิติแบบเดิม สำหรับ ระบบช่วยตัดสินใจแบบ cost-based ซึ่งจะช่วยระบบในการตัดสินใจที่ระดับ โกลบอล และแยกคิวรีในระดับโกลบอลออกเป็นคิวรีสำหรับแต่ละแฟรกเมนท์ และคิวรีในระดับแฟรกแมนท์จะถูกปรับปรุงให้ดีขึ้นโดยระบบจัดการฐานข้อมูลที่ไซต์นั้นๆ

กฎข้อบังคับ

แบ่งออกเป็น 4 ประเภท ดังนี้

1. Site Integrity Constraint กำหนด ค่าของข้อมูลเฉพาะแต่ละไซต์ เช่น ไซต์ A PROGRAM = "ช่างอุตสาหกรรม" ไซต์ B PROGRAM = "เกษตรกรรม"

2. ขอบเขตของข้อมูล

เป็นการกำหนดขอบเขตของข้อมูลที่ถูกต้องสำหรับเขตข้อมูลนั้น เช่น Age > 18 นั่นคือ ค่าที่จะนำเข้าไปใน Age ได้นั้นต้องมีค่ามากกว่า 18 โดยมีตัวดำเนินการคือ = , != , < , > ,<= , >= และช่วงของข้อมูล เช่น 18 >= Age < 35

3. ความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependency)

หากแอททริบิวแรกมีค่าตามที่กำหนดไว้แล้ว อีกแอททริบิวหนึ่งก็จะถูกส่งผลให้มีค่าอยู่ในช่วงที่กำหนด อย่างเช่น ถ้า x> 5000 -> y < 50 หมายความว่า ในกรณีที่ ค่า x มีค่ามากกว่า 5000 แล้ว ค่า y จะต้องมีค่าน้อยกว่า 50

4. ความสัมพันธ์ระหว่างแอททริบิว

เป็นการกำหนดความสัมพันธ์ระหว่างแอททริบิว เช่น x < y หมายความว่า ค่า x จะต้องมีค่าน้อยกว่า y เสมอ ในแถวเดียวกัน

3.2 การตรวจสอบเงื่อนไข

3.2.1 การตรวจสอบเงื่อนไขจากฐานข้อมูลกฎข้อบังคับ

เมื่อได้รับค่าของเงื่อนไขที่ต้องการตรวจสอบมาแล้ว จะนำมาเปรียบเทียบกับกับกฎข้อบังคับที่กำหนดไว้ โดยตรวจสอบจากแอททริบิว ซึ่งแบ่งออกเปนกรณีต่างๆ ตามตัวดำเนินการดังนี้

1. เครื่องหมาย =

= : ถ้าเพลทดิเคท ที่เข้ามาไม่มีค่าตามที่กำหนดเอาไว้ หมายความว่า ขัดแย้งกับกฎ

!= : ถ้าเพลทดิเคท ที่เข้ามามีค่าเท่ากับ ค่าที่กำหนดไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

< : ถ้าเพลทดิเคทที่เข้ามามีค่า มากกว่าหรือเท่ากับ ค่าที่กำหนดไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

> : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับ ค่าที่กำหนดไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

<= : ถ้าเพลทดิเคทที่เข้ามามีค่ามากกว่า ค่าที่กำหนดไว้ในกฎ หมายความว่า ขัดแยงกับกฎ

>= : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่า ค่าที่กำหนดไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < , <= x < , < x <= , <= x <= : ถ้าเพลทดิเคทที่ข้ามามีค่าไม่อยู่ในช่วงที่กำหนดไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

In : ถ้าเพลทดิเคทที่เข้ามาไม่มีค่าตามที่ระบุไว้ในกฎ หมายความว่า ขัดแย้งกับกฎ

2. เครื่องหมาย !=

= : ถ้าเพลทดิเคทที่เข้ามามีค่าเท่ากับ ค่าที่กำหนดไว้ในกฎ หมายความว่าขัดแย้งกับกฎ

3. เครื่องหมาย <

= ,> , >= : ถ้า เพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < , < = x < , <= x < , <= x <= : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับ ค่าด้านซ้ายของกฎ หมายความว่า ขัดแย้งกับกฎ

4. เครื่องหมาย >

= ,> , >= : ถ้า เพลทดิเคทที่เข้ามามีค่ามากกว่าหรือเท่ากับค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < , < = x < , <= x < , <= x <= : ถ้าเพลทดิเคทที่เข้ามามีค่ามากกว่าหรือเท่ากับ ค่าด้านขวาของกฎ หมายความว่า ขัดแย้งกับกฎ

5. เครื่องหมาย <=

> : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับค่าที่อยู่ในกฎ หมายความว่าขัดแย้งกับกฎ

= , >= : ถ้าเพลทดิเคทที่เข้ามา มีค่าน้อยกว่าค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < , < x <= : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับค่าที่อยู่ทางด้านซ้ายของกฎ หมายความว่า ขัดแย้งกับกฎ

<= x < , <= x <= : ถ้าเพลทดิเคทที่เข้ามามีค่าน้อยกว่าค่าที่อยู่ทางซ้ายของกฎ หมายความว่า ขัดแย้งกับกฎ

6. เครื่องหมาย >=

> : ถ้าเพลทดิเคทที่เข้ามามีค่ามากกว่าหรือเท่ากับค่าที่อยู่ในกฎ หมายความว่าขัดแย้งกับกฎ

= , >= : ถ้าเพลทดิเคทที่เข้ามา มีค่ามากกว่าค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < , < x <= : ถ้าเพลทดิเคทที่เข้ามามีค่ามากกว่าหรือเท่ากับค่าที่อยู่ทางด้านขวาของกฎ หมายความว่า ขัดแย้งกับกฎ

<= x < , <= x <= : ถ้าเพลทดิเคทที่เข้ามามีค่ามากกว่าค่าที่อยู่ทางขวาของกฎ หมายความว่า ขัดแย้งกับกฎ

7. เครื่องหมาย <= x <=

= : ถ้าเพลทดิเคทที่เข้ามามีค่าไม่อยู่ในช่วงของค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< : ถ้าค่าทางซ้ายของเพลทดิเคทที่เข้ามามีค่ามากกว่าหรือเท่ากับ ค่าที่อยู่ในกฎ หมายความว่าขัดแย้งกับกฎ

> : ถ้าค่าทางขวาของเพลทดิเคทที่เข้ามามีค่าน้อยกว่าหรือเท่ากับ ค่าที่อยู่ในกฎ หมายความว่า ขัดแย้งกับกฎ

< x < : ถ้าค่าของเพลทดิเคทที่เข้ามาไม่อยู่ในช่วงข้อมูลของค่าที่อยู่ในกฎ หมายความว่าขัดแย้งกับกฎ กรณีดังต่อไปนี้ปืนกรณีที่ไม่ขัดแย้งกับกฎข้อบังคับ

- ถ้าค่าทางด้านซ้ายของเพรดดิเคตมีค่าน้อยกว่าค่าทางด้านขวาของกฎ และ ค่าทางด้านซ้ายของเพรดดิเคตมีค่ามากกว่าค่าทางด้านซ้ายของกฎ

- ถ้าค่าทางด้านขวาของเพรดดิเคตมีค่าน้อยกว่าค่าทางด้านขวาของกฎ และค่าทางด้านขวาของกฎมีค่ามากกว่าค่าทางด้านขวาของเพรดดิเคต

- ถ้าค่าทางด้านขวาของเพรดดิเคตนั้นมีค่ามากกว่าหรือเท่ากับค่าทางด้านขวาของกฎ และค่าทางด้านซ้ายของกฎมีค่าน้อยกว่าค่าทางด้านซ้ายของกฎ

< and <= , <= and < , <= and <= : จะพิจารณาเช่นเดียวกับกรณี < and < แต่ต้องพิจารณาตรงเครื่องหมายเท่ากับด้วย

8. เครื่องหมาย in

= : ถ้าค่าในเพลทดิเคทที่เข้ามา มีค่าใดค่าหนึ่ง เท่ากับค่าที่อยู่ในกฎ หมายความว่าไม่ขัดแย้งกับกฎ แต่ถ้าทุกค่าในเพลทดิเคทที่เข้ามาไม่มีค่าใดเท่ากับค่าที่อยู่ในกฎเลย หมายความว่าขัดแย้งกับกฎ

In : ถ้าค่าในเพลทดิเคทที่เข้ามา มีค่าใดค่าหนึ่ง เท่ากับค่าที่อยู่ในกฎ หมายความว่าไม่ขัดแย้งกับกฎ แต่ถ้าทุกค่าในเพลทดิเคทที่เข้ามาไม่มีค่าใดเท่ากับค่าที่อยู่ในกฎเลย หมายความว่าขัดแย้งกับกฎ

3.2.2 การตรวจสอบเงื่อนไขจาก ความสัมพันธ์ระหว่างแอททริบิว

1. เมื่อตรวจสอบพบความขัดแย้งกับกฎข้อบังคับ

เมื่อได้รับคำสั่งคิวรีเข้ามา และทำการตรวจสอบเงื่อนไขในอนุประโยค where ว่ามีความขัดแย้งกับกฎที่กำหนดไว้หรือไม่ ซึ่งหากพบว่ามีความขัดแย้งกับกฎ ก็จะแสดงผลให้ผู้ใช้ทราบว่า ไม่มีข้อมูลที่ต้องการในฐานข้อมูล โดยไม่ทำการส่งคำสั่งคิวรีไปยังฐานข้อมูลงาน

ตัวอย่างที่ 1

ในกรณีที่เงื่อนไขใน where clause เป็นดังนี้ X > 100 and Y <40

กฎข้อบังคับที่กำหนดไว้คือ X < Y

ในกรณีจะเห็นว่า ค่า X จะมีค่า มากกว่า ค่า Y แต่เงื่อนไขใน where ให้ X มีค่ามากกว่า 100 และ Y มีค่าน้อยกว่า 40 นั่นแสดงว่า ค่า X จะมากกว่า Y ดังนั้น เงื่อนไขที่เข้ามาจึงขัดแย้งกับกฎข้อบังคับที่กำหนดไว้ แสดงว่าไม่มีข้อมูลในฐานข้อมูล แสดงผลให้ผู้ใช้ทราบ โดยไม่ส่งคำสั่งคิวรีไปประมวลผลที่ฐานข้อมูล

2. เมื่อตรวจสอบเงื่อนไขแล้วไม่พบความขัดแย้งกับกฎข้อบังคับ

จากทฤษฎี index introduction ซึ่งเป็นการเพิ่มเงื่อนไขใหม่เข้าไปในคำสั่งคิวรี โดยเงื่อนไขที่เพิ่มเข้าไปนั้นจะต้องมี แอททริบิวที่เป็น index อยู่ด้วย ซึ่งจะช่วยให้ค้นหาข้อมูลได้เร็วขึ้น

ตัวอย่าง

ในกรณีที่ เงื่อนไข ใน where clause เป็นดังนี้ Y < 50

กฎข้อบังคับที่กำหนดไว้คือ X < Y

ในกรณีนี้ คำสั่งคิวรีของผู้ใช้ที่เข้ามา ค่า Y มีค่า น้อยกว่า 50 แต่กฎข้อบังคับ กำหนดไว้ว่า X มีค่าน้อยกว่า Y ดังนั้น ค่า X จะมีค่า น้อยกว่า 100 ด้วย ในแถวเดียวกัน หากแอททริบิว X เป็น index เราสามารถเพิ่มเงื่อนไข X < 50 เข้าไปในคำสั่งคิวรีได้ ดังนี้

Y < 50 and X < 50

3.2.3 การตรวจสอบเงื่อนไขจากความสัมพันธ์ที่ขึ้นต่อกัน

1. เมื่อตรวจสอบแล้วพบความขัดแย้งกับกฎข้อบังคับ

เมื่อได้รับคำสั่งคิวรีเข้ามา และทำการตรวจสอบเงื่อนไขในอนุประโยค where ว่ามีความขัดแย้งกับกฎที่กำหนดไว้หรือไม่ ซึ่งหากพบว่ามีความขัดแย้งกับกฎ ก็จะแสดงผลให้ผู้ใช้ทราบว่า ไม่มีข้อมูลที่ต้องการในฐานข้อมูล โดยไม่ทำการส่งคำสั่งคิวรีไปยังฐานข้อมูลงาน

ตัวอย่าง

ในกรณีที่ เงื่อนไข ใน where clause เป็นดังนี้ X >10 and Y = 50

กฎข้อบังคับที่กำหนดไว้คือ X > 5 -> Y > 100

ในกรณีนี้จากกฎข้อบังคับ ถ้า X > 5 แล้ว Y จะมีค่ามากกว่า 100 แต่คำสั่งคิวรีของผู้ใช้มีเงื่อนไขว่า X > 10 และ Y = 50 ซึ่งไม่มีในฐานข้อมูลเพราะขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรีไปประมวลผล

2. เมื่อตรวจสอบแล้วไม่พบความขัดแย้งกับกฎข้อคับ

2.1 เมื่อเงื่อนไขของเพลดดิเคตที่เข้ามามีความซ้ำซ้อน กับกฎข้อบังคับ

จากเทคนิคการแปลงคำสั่งคิวรี สามารถตัดเงื่อนไขที่ซ้ำซ้อนกับกฎข้อบังคับ ถ้าหากแอททริบิวนั้นไม่ได้เป็น index

ตัวอย่าง

ในกรณีที่ เงื่อนไข ใน where clause เป็นดังนี้ X > 10 and Y > 50

กฎข้อบังคับที่กำหนดไว้คือ X > 5 -> Y > 100

ในกรณีนี้จากกฎข้อบังคับ ถ้า X มีค่ามากกว่า 5 แล้ว Y จะมีค่ามากกว่า 100 ส่วนคำสั่งคิวรีของผู้ใช้มีเงื่อนไขว่า ค่า X มีค่ามากกว่า 10 และ ค่า Y มีค่ามากกว่า 50 ซึ่งจากกฎ เมื่อ X > 5 ค่า Y มีค่ามากกว่า 100 ดังนั้น เงื่อนไข Y > 50 ไม่จำเป็น เพราะ ในฐานข้อมูล ค่า Y มีค่ามากกว่า 50 อยู่แล้ว

2.2 เมื่อเงื่อนไขของเพลดดิเคตที่เข้ามาไม่มีความซ้ำซ้อน กับกฎข้อบังคับ

- การลดช่วงการค้นหาข้อมูล โดยเพิ่มเงื่อนไขจากกฎข้อบังคับเข้าไปใน คำสั่งคิวรีของผู้ใช้

ตัวอย่าง

ในกรณีที่ เงื่อนไข ใน where clause เป็นดังนี้ X > 10 and Y > 50

กฎข้อบังคับที่กำหนดไว้คือ X > 10 -> Y < 100

ในกรณีนี้จะเห็นว่า จากกฎข้อบังคับ ถ้า X มีค่ามากกว่า 10 แล้ว Y มีค่าน้อยกว่า 100 ซึ่งในคำสั่งคิวรีของผู้ใช้ X > 10 ดังนั้น จึงสามารถเพิ่มเงื่อนไขเข้าไปในคำสั่งคิวรีได้ เป็นดังนี้ X > 10 and Y > 50 and Y < 100

- การเพิ่มเงื่อนไขเข้าไปใน คำสั่งคิวรีของผู้ใช้ โดยเงื่อนไขใหม่นั้นมี แอททริบิวที่เป็น index

ในกรณีที่ เงื่อนไข ใน where clause เป็นดังนี้ X > 5

กฎข้อบังคับที่กำหนดไว้คือ X > 5 -> Y > 50

ในกรณีนี้ แอททริบิว Y เป็น index เมื่อคิวรีจากผู้ใช้ มีเงื่อนไข ตรงกับ กฎ ก็สามารถเพิ่มเงื่อนไข โดยแอททริบิวที่เป็น index ได้ เป็น X > 5 and Y > 50

3.3 การปรับปรุงคำสั่งคิวรีโดยการใช้ข้อมูลทางสถิติ

3.3.1 การปรับปรุงคิวรีโดยใช้ หลักการ Driving Table

driving table จะเป็นตารางแรกที่ถูกใช้ จากนั้น rows ของตารางที่สองจะถูกรวมเข้าไปกับเซทผลลัพธ์ (result set) ของตารางแรก ซึ่ง driving table ไม่จำเป็นที่จะต้องเป็นตารางที่มีจำนวน rows ที่น้อยที่สุดเสมอไป โดยที่เราต้องทำการดูค่า rows return โดยเปรียบเทียบจากเงื่อนไขใน WHERE clause ด้วย ซึ่งหมายความว่า driving table ควรเป็นตารางที่ return rows น้อยที่สุดเมื่อดูเทียบจาก WHERE clause ด้วย ซึ่ง จะนำมากำหนดให้เป็น driving tableใน FROM

3.3.2 การกำหนดลำดับการ join ตารางโดยใช้ ORDERED Hint

ใน cost-based optimizer นั้น ORDERED hint จะทำตารางถูก join ตามลำดับที่ระบุไว้ใน from clause ซึ่งตารางแรกใน from clause จะถูกใช้เป็น driving table

เมื่อมีการ join หลายๆตารางเช่น7 ตารางหรือมากกว่านั้นขึ้นไป ซึ่งใช้เวลาในการส่ง SQL นานมาก เนื่องจากว่า จะต้องคำนวณหาลำดับของการ join ที่เป็นไปได้ทั้งหมด เช่นเมื่อมีการ join ตาราง 9 ตารางเข้าด้วยกันใน query แล้วจะทำการหาลำดับของการ join ที่เป็นไปได้ทั้งหมดซึ่งก็คือ 9! หรือ 362880 ทางที่เป็นไปได้ซึ่งจะเป็นการเสียเวลามากในการที่จะมาหาลำดับการ join ที่ดีที่สุดจากวิธีการทั้งหมดที่เป็นไปได้ ซึ่งหากเรารู้ลำดับของการ join ที่ดีที่สุดอยู่แล้ว เราสามารถใช้ ORDERED hint เขียนเพิ่มไปใน query เพื่อลดเวลาลงไปได้

3.3.3 การใช้งาน USE_CONCAT Hint เพื่อให้มีการทำ Union All

กรณีที่ USE_CONCAT hint จะถูกมองข้ามไป คือกรณีที่ ไม่มี OR condition หลายๆ condition ในwhere clause ซึ่งหากเราตรวจสอบได้ว่ามีการการทำ OR condition หลาย condition อนุประโยค WHERE แล้ว และทุก condition สามารถถูก access ผ่าน index ได้ เราสามารถใช้ USE_CONCAT hint เพื่อให้มีการทำแผนการประมวลผล union all ได้

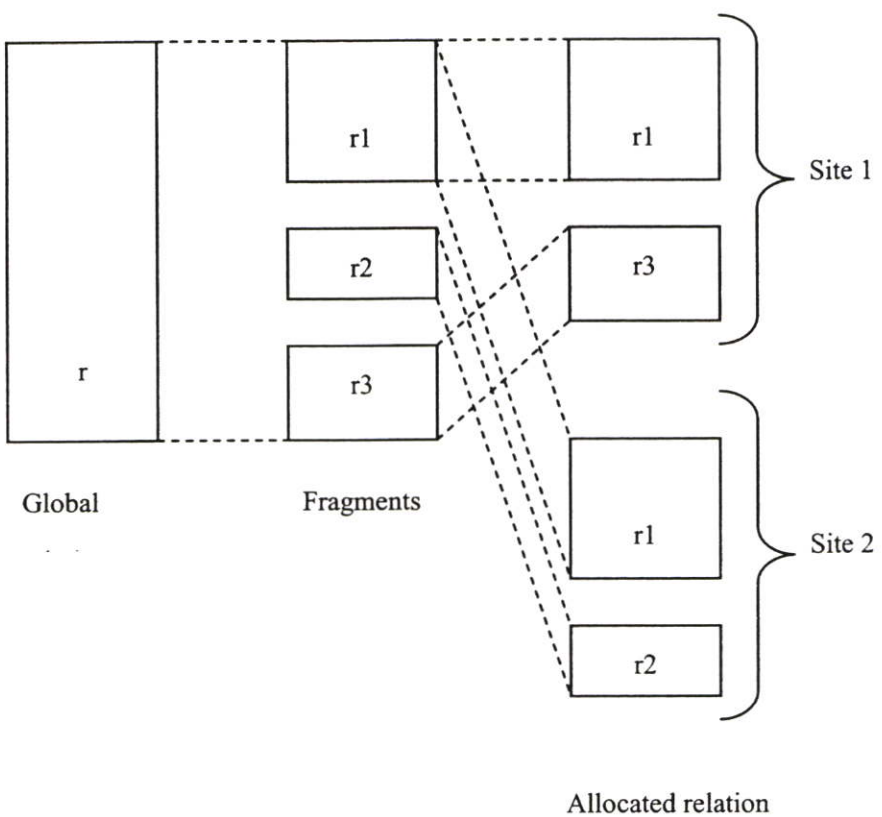
3.3.4 การใช้งาน Driving SITE Hint

เป็นการกำหนดให้ query ประมวลผลที่ไซต์ที่กำหนด ซึ่งสามารถใช้ได้กับทั้ง rule-based และ cost-based optimization

บทที่ 4
การทำงานบนฐานข้อมูลแบบกระจาย

ในบทนี้จะกล่าวถึงหลักการและวิธีการดำเนินงานวิจัย ของงานวิจัยที่ผู้วิจัยนำเสนอ ซึ่งในส่วนนี้จะนำเสนอถึงวิธีการในการดำเนินงานวิจัย เป็นต้น

4.1 สถาปัตยกรรมข้อมูลแบบกระจาย (Distributed Data Architecture)



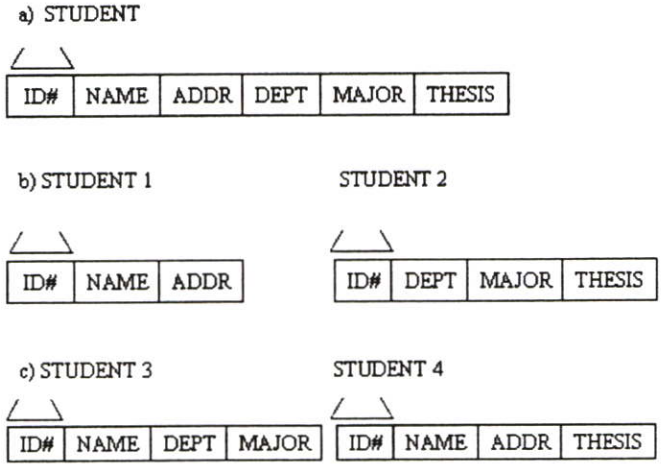
รูปที่ 4.1 ตัวอย่าง fragmentation และ allocation

วัตถุประสงค์หลักของระบบคือต้องการความสมบูรณ์ของฐานข้อมูลที่ซ่อนการแยกข้อมูลทางกายภาพ การแยกข้อมูลทางลอจิกและคุณสมบัติเฉพาะของระบบจัดการฐานข้อมูลไว้จากผู้ใช้งาน โดยระบบต้องมีความสามารถในการตัดสินใจค้นหาข้อมูลเชิงความหมาย ซึ่งคิวรีในระดับ

โกลบอลจะต้องสามารถแยกคิวรีแล้วส่งออกไปยังแฟรกเมนท์และไซต์ ดังนั้นเวลาตอบรับของคิวรีในระดับโกลบอลก็จะลดน้อยลง เป็นไปได้ว่า แฟรกเมนท์ในที่นี้เป็นการผสมผสานกันของแฟรกเมนท์และระบบต้องสามารถจัดการ การอัพเดทซึ่งอาจจะมีการเคลื่อนย้ายข้อมูลในระดับกายภาพโดยที่อยู่ต่างไซต์กัน และนี่คือประเด็นสำคัญของงานนี้

ในระดับแฟรกเมนท์สามารถถูกสำเนาไปยังไซต์ต่างๆ ได้ ซึ่งในแต่ละไซต์อาจจะมีระบบจัดการฐานข้อมูลที่แตกต่างกัน ถ้าหากการอัพเดทข้อมูล มีการเปลี่ยนแปลงข้อมูลในแฟรกเมนท์สำเนาในไซต์ต่างๆ ก็จะต้องมีการเปลี่ยนแปลงตามไปด้วย ทรานแซกชั่นในการซิงโครไนซ์ ต้องมีการใช้โปรโตคอล Two – phase commit จากรูปที่ 4.1 แสดงให้เห็นของหลักการของโกลบอลรีเลชั่น แฟรกเมนท์และออลโลเคชั่น ในรีเลชั่น r เป็น แฟรกเมนท์แนวนอน ซึ่งใน แต่ละแฟรกเมนท์ถูกเก็บไว้ในไซต์ที่ต่างกัน

4.2 การทำงานบน Mixed Fragmentation



รูปที่ 4.2 Mixed Fragmentation.

กลยุทธ์การอิมพลีเมนท์ของระบบมิกซ์แฟรกเมนท์ คือเมื่อมีการอัพเดทรีเลชั่นในระดับโกลบอลด้วยมิกซ์แฟรกเมนท์เทชั่น การอัพเดทข้อมูลง่ายๆในระดับโกลบอลอาจจะถูกแปลงเป็นหลายๆ คำสั่ง เช่น คำสั่ง select , insert และ delete ในหลากหลายไซต์ ในรูปที่ 4.2a แสดงให้เห็นความสัมพันธ์ในระดับโกลบอล โครงสร้างข้อมูลของ STUDENT ซึ่งกำหนดให้มีความสัมพันธ์ในแนวนอน 2 ส่วน ส่วนแรกประกอบด้วยข้อมูลนักศึกษา ซึ่งอยู่ในภาควิชาตั้งแต่ 1 ถึง 10 และส่วนที่สอง ประกอบด้วย ข้อมูลนักศึกษาซึ่งอยู่ในภาควิชาตั้งแต่ 11 ถึง 20

ในรูปที่ 4.2b แสดงให้เห็นการแบ่งส่วนในแนวนอน และถูกแบ่งข้อมูลในแนวตั้งด้วยโครงสร้างข้อมูลดัง STUDENT 1และ STUDENT2 ซึ่งการแบ่งข้อมูลแบบนี้แตกต่างจากการแบ่งข้อมูลในแนวตั้งด้วยโครงสร้างข้อมูลดัง STUDENT3 และ STUDENT4 ในรูปที่ 4.2c และการอัพเดทข้อมูลง่ายๆ ในระดับโกลบอลเพียงคำสั่งเดียว ในตาราง STUDENT อาจก่อให้เกิดคำสั่งย่อยหลายๆ คำสั่งเช่น select update และ delete ดังจะแสดงคำสั่ง SQL ให้เห็นดังตัวอย่างข้างล่างนี้ โดยกำหนดให้นักศึกษาที่มีรหัส ID#12345 ต้องการที่จะย้ายจากภาควิชา 2 ไปยังภาควิชา 11 ในระดับของส่วนกลางจะใช้คำสั่งที่ง่ายๆ ดังนี้

```
UPDATE STUDENT
SET DEPT=11
WHERE ID#=12345;
```

ก่อนที่นักศึกษารหัส 12345 จะถูกย้าย ข้อมูลถูกเก็บไว้ในแนวตั้งส่วนแรก และเมื่อข้อมูลเปลี่ยนแปลงจะต้องถูกย้ายไปเก็บไว้ในข้อมูลในแนวตั้งส่วนที่สองและเป็นไปได้ว่าอาจจะอยู่ต่างไซต์กัน ดังนั้น จึงจำเป็นที่จะต้องใช้คำสั่ง SELECT ในการดึงข้อมูล นักศึกษารหัส 12345 จากตาราง STUDENT1 และ STUDENT2 ผลลัพธ์ที่ได้จะต้องนำมาเพิ่มเข้าไปในตาราง STUDENT3 และ STUDENT4 โดยใช้คำสั่ง INSERT จากนั้นใช้คำสั่ง DELETE ลบข้อมูลเดิมใน STUDENT1 และ STUDENT2 สิ้นสุดกระบวนการในระดับ fragmentation คำสั่ง SQL ที่ใช้งานแสดงดังตัวอย่างข้างล่างนี้

```
SELECT ID#, NAME, ADDR INTO VAR1, VAR2, VAR3
FROM STUDENT1
WHERE ID#=12345;
SELECT DEPT, MAJOR, THESIS INTO VAR4, VAR5, VAR6
FROM STUDENT2
WHERE ID#=12345;
INSERT INTO STUDENT3 VALUE (VAR1, VAR2, VAR4, VAR5);
INSERT INTO STUDENT4 VALUE (VAR1, VAR2, VAR3, VAR6);
DELETE FROM STUDENT1
WHERE ID#=12345;
DELETE FROM STUDENT2
WHERE ID#=12345;
```

ในกรณีของระดับกายภาพต้องมองในเรื่องของการจองพื้นที่ของในแต่ละไซต์ที่จะทำการ replicate ซึ่งในทุกไซต์ที่มีการเปลี่ยนแปลงจะต้องอ้างอิงไปถึง ใน STUDENT1 จองพื้นที่ไว้ใน SERVER 3 ใน STUDENT2 จองพื้นที่ไว้ใน SERVER4 ใน STUDENT3 จองพื้นที่ไว้ใน SERVER1 และ SERVER2 และใน SERVER4 ใน STUDENT3 จองพื้นที่ไว้ใน SERVER1 และ SERVER2 เช่นกัน ดังนั้น จะมีการกระจายทรานแซกชั่นออกเป็นดังนี้

```
SELECT ID#, NAME, ADDR INTO VAR1, VAR2, VAR3
FROM STUDENT1 AT SERVER3
WHERE ID#=12345;
SELECT DEPT, MAJOR, THESIS INTO VAR4, VAR5, VAR6
FROM STUDENT2 AT SERVER4
WHERE ID#=12345;
INSERT INTO STUDENT3 AT SERVER1 VALUE (VAR1, VAR2, VAR4, VAR5);
INSERT INTO STUDENT4 AT SERVER1 VALUE (VAR1, VAR2, VAR3, VAR6);
INSERT INTO STUDENT3 AT SERVER2 VALUE (VAR1, VAR2, VAR4, VAR5);
INSERT INTO STUDENT4 AT SERVER2 VALUE (VAR1, VAR2, VAR3, VAR6);
DELETE FROM STUDENT1 AT SERVER3
WHERE ID#=12345;
DELETE FROM STUDENT2 AT SERVER4
WHERE ID#=12345;
```

คำสั่ง SQL เหล่านี้รวมกันเป็นทรานแซกชั่นเดียวกัน ซึ่งจะมีส่วนที่ช่วยจัดการทรานแซกชั่นเหล่านี้โดยอยู่ภายใต้สภาวะแวดล้อมของระบบจัดการฐานข้อมูลแบบกระจาย ไม่เพียงเฉพาะแต่ การอัพเดทในระดับของส่วนกลางเท่านั้นที่นำไปสู่ความซับซ้อนในระดับที่ต่ำกว่า แต่บางครั้งคำสั่ง SELECT ในระดับส่วนกลางอาจจะแยกได้เป็นคำสั่ง SQL หลายๆ คำสั่งในระดับ fragmentation และมากกว่านั้นในระดับ allocation

ในตัวอย่างนี้ กำหนดให้ค้นหาข้อมูลนักศึกษา โดยค้นหาข้อมูลจาก รหัสนักศึกษา

```
SELECT *
FROM STUDENT
WHERE ID#=12345;
```

ในระดับ fragmentation แยกคำสั่ง ได้เป็นดังนี้

```
SELECT T1.ID#, NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT1 T1, STUDENT2 T2
WHERE T1.ID#=T2.ID# AND T1.ID#=12345;
If not found then
SELECT T1.ID#, T1.NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT3 T3, STUDENT4 T4
WHERE T3.ID#=T4.ID# AND T3.ID#=12345;
```

ในระดับ Allocation คิวรีที่ถูกส่งไปยังแต่ละไซต์ ในกรณีของการ replicate มันจะถูกส่งไปยังไซต์ที่อยู่ใกล้ที่สุด ที่มีตารางที่ต้องการอยู่ ในที่สุดแล้ว คำสั่ง SQL ในระดับ fragmentation จะถูกแปลงออกมาได้ดังนี้

```
SELECT T1.ID#, NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT1 AT SERVER3 T1, STUDENT2 AT SERVER4 T2
WHERE T1.ID#=T2.ID# AND T1.ID#=12345;
If not found then
SELECT T1.ID#, T1.NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT3 AT SERVER1 T3, STUDENT4 AT SERVER1 T4
WHERE T3.ID#=T4.ID# AND T3.ID#=12345;
```

4.3 สรุป

ในบทความนี้นำเสนอการออฟติไมซ์โดยใช้หลักการของ semantic เข้ามาร่วมกับ การใช้สถิติเพื่อช่วยในการตัดสินใจ และ ช่วยการจัดการในฐานข้อมูลแบบกระจาย เมื่อมีการเรียก หรือจัดการข้อมูลในระดับโกลบอล

บทที่ 5

ผลการทดลอง

ในบทนี้จะกล่าวถึงการนำการค้นหาข้อมูลเชิงความหมายมาใช้ในระบบฐานข้อมูลแบบกระจาย และทำการเปรียบเทียบประสิทธิภาพในการค้นหาข้อมูลเชิงความหมายบนฐานข้อมูลแบบกระจายเมื่อไม่มีตัวช่วยปรับปรุงคำสั่ง คิวรี และเมื่อมีการค้นหาข้อมูลเชิงความหมายโดยมีการปรับปรุงคำสั่งคิวรี โดยในการเปรียบเทียบจะใช้ค่า response time

5.1 การเตรียมข้อมูลสำหรับการทดลอง

การทดลองนี้จะใช้ระบบจัดการฐานข้อมูล oracle 10g มาสร้างเป็นระบบจัดการฐานข้อมูลแบบกระจายโดยติดตั้งในแบบ Enterprise Edition เพื่อทำให้ฐานข้อมูลที่สร้างขึ้นเป็นฐานข้อมูลแบบกระจาย โดยติดตั้งลงบนเครื่องแม่ข่ายจำนวน 4 เครื่อง เพื่อให้เห็นภาพการทำงานของฐานข้อมูลแบบกระจาย โดยใช้ข้อมูลการทดลองเป็นข้อมูลนักศึกษาจำนวน 15,000 คน

ในปัจจุบันนี้ oracle ได้สนับสนุนการแนวความคิดฐานข้อมูลแบบกระจายในระดับหนึ่งแต่ยังไม่ครอบคลุมจนถึงในระดับของโกลบอลสกีมาและยังมีข้อจำกัดในการ insert update และ delete จึงได้อาศัยกระบวนการโดยในส่วนของ Location mapping transparency นั้นได้ใช้ Database link เข้ามาช่วยจัดการ ส่วนกรณีของ location transparency ใช้ synonym เข้ามาช่วยเพื่อที่ผู้ใช้ไม่ต้องทราบว่าเรียกใช้ข้อมูลจากที่ใด

สำหรับในส่วนของ fragmentation transparency สามารถสร้าง view เข้ามาครอบข้อมูลที่ถูกเก็บอยู่ในที่ต่างๆ แทนการใช้งานตารางโดยตรง เพราะ ใน Oracle นั้นยังไม่รองรับการทำ Fragmentation โดยตรง โดยยังมีข้อจำกัดหลายๆอย่าง เช่น มีข้อจำกัดที่ว่า Integrity constraint จะไม่สามารถใช้ foreign key อ้างถึง primary key ของตารางที่อยู่ต่างที่กันได้

การสร้าง View ที่ทำหน้าที่รวมข้อมูลจากที่ต่างๆ มาไว้ด้วยกันนั้นทำโดยการใช้คำสั่งในการรวมข้อมูลจากภาษา SQL ที่ใช้เรียกดูข้อมูล โดยจะใช้การ Join ของสองตารางเมื่อมีการแบ่งข้อมูลตามแนวคอลัมน์ (Vertical Fragmentation) และใช้การ Union ข้อมูลเมื่อมีการแบ่งข้อมูลในลักษณะของการแบ่งแถว (Horizontal Fragmentation) ซึ่งเมื่อทำการรวมข้อมูลจากที่ต่างๆโดยใช้ View นั้นจะเกิดปัญหาตามมาคือ View ที่สร้างขึ้นจากการ Join หรือ Union กันของข้อมูลนั้นไม่สามารถ Insert Update และ Delete ข้อมูลได้ โดยปัญหาดังกล่าว สามารถแก้ได้โดยการใช้งาน Trigger ตัวหนึ่งคือ Instead of Trigger

Instead of Trigger เป็น Trigger ประเภทหนึ่ง ซึ่งใช้ภาษา PL/SQL ในการเขียนคำสั่งในการทำงานขึ้นมา สำหรับ Instead of trigger นี้จะช่วยให้สามารถทำการ Insert, Update, delete ผ่าน View ที่ไม่สามารถทำการ Update ข้อมูลได้ อย่างเช่น view ที่เกิดจาก Union หรือ จากการ Join ในบางกรณี อธิบายการทำงานได้คือ จะมีการตรวจสอบว่ามีคำสั่ง Insert, Update, delete ผ่าน View หรือไม่ ถ้ามีก็เข้าไปทำตามคำสั่งที่เขียนไว้ใน Trigger body ซึ่งก็คือ เราต้องทำการเขียนคำสั่ง Insert, Update, Delete ตารางที่ต้องการด้วยตัวเอง

5.2 กฎข้อบังคับที่กำหนด

5.2.1 เงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว (Attribute Constraint)

CAMPUS < 100

TOTAL_CREDIT <= 88

500 <= CURR_BAL <= 600000

CREDIT > 77

LOWPRO < 1.5

1.5 < HIGHPRO < 2.0

CLASSSTATUS = ('N','W','A','D','X','C')

5.2.2 เงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว (Inter-Attribute Constraint)

RECIEVEAMOUNT < LIMITAMOUNT

BALANCE <= AMOUNT

ENROLLSEAT < TOTALSEAT

EXAMTIMEFROM < EXAMTIMETO

5.2.3 เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependency Constraint)

เพรดดิเคต 1	⟶	*เพรดดิเคต 2*
CAMPUS = 70	⟶	ID# > 7000
FAC = '03'	⟶	LEVEL > 3
MAJOR = 'IE'	⟶	CAMPUS < 50
FAC = '05'	⟶	FID = 10
GRADESTATUS = 'W'	⟶	STUDENTSTATUS < 40
GPAX < 2.0	⟶	GRADEPRO = L

MAJOR = 'BU' ⟶ 200< PREGRAMID < 500

GRADEPRO = 'L' ⟶ CREDITATTEMP < 9

GRADEPRO='H' ⟶ CREDITATTEMP < 12

5.2.4 เงื่อนไขประเภทลักษณะเฉพาะ Site Constraint

GROUP = 'ช่างอุตสาหกรรม' ⟶ SITE = 'S2'

GROUP = 'บริหารธุรกิจ' ⟶ SITE = 'S4'

5.3 ผลการทดลอง

ผลการทดลองได้ดำเนินการทดลองออฟติไมซ์โดยใช้ semantic ได้ผลการทดลอง โดยเวลาที่ใช้ในการประมวลผลมีเวลาที่ได้น้อยกว่าเวลาที่ส่งออกไปโดยไม่ใช้ออฟติไมซ์

1. กรณีที่คำสั่งคิวรีของผู้ใช้ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด จะไม่ส่งคำสั่งคิวรีนั้นไปประมวลผล โดยใช้หลักการ Detection of unsatisfiable conditions

ตัวอย่างที่ 1

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where campus = 70 and ID# < 50000
```

กฎข้อบังคับ : campus = 70 -> ID# > 70000

ในกรณีนี้คำสั่งคิวรีขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรีนั้นไปประมวลผล

ตารางที่ 5.1 Response time จากหลักการ Detection of unsatisfiable conditions 1

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.453	0.531	0.336	0.843	0.734
คำสั่งคิวรีที่ถูกแปลง	0	0	0	0	0

ในคำสั่งที่เข้ามานี้ได้ขัดแย้งกับกฎข้อกำหนดที่ว่า ถ้า campus มีค่าเท่ากับ 70 นั้นจะมีค่า ID# มากกว่า 70000 เท่านั้น ถ้าต้องการ ID# น้อยก่า 50000 นั้นจะไม่มีอยู่ในฐานข้อมูล โดยในขั้นแรกจะทำการรับข้อมูลคำสั่งเข้ามาแล้วตรวจสอบกับฟังก์ชั่นตรวจสอบของค่าขอบเขตแอททริบิวก่อน โดยในกรณีนี้จะมีกฎที่เกี่ยวข้องคือ campus < 100 จะมีการนำค่า 100 และ 70 ไปเปรียบเทียบกัน เมื่อพบว่าไม่ขัดแย้งกับกฎแล้ว ก็เข้าสู่กระบวนการแปลง โดยใช้กฎข้อบังคับของเงื่อนไข

ประเภทความสัมพันธ์ที่ขึ้นต่อกัน และเงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว ในคำสั่งที่รับเข้ามานี้มีกฎมีการนำมาตรวจสอบคือ

CAMPUS = 70 $\longrightarrow$ ID# > 7000

โดยขั้นแรกนั้นทำการตรวจสอบว่าคำสั่งที่รับเข้ามานั้นมีเพรดดิเคตที่สอดคล้องกับกฎหรือไม่ WHERE campus = 70 and ID# < 50000 จะพบว่ามีกฎอยู่ 1 กฎ ที่ถูกนำมาใช้ ซึ่งจะเริ่มตรวจสอบที่กฎแรก เมื่อ กฎแรกมีเพรดดิเคตแรก CAMPUS = 70 ครอบคลุมเพรดดิเคตที่เข้ามา คือ address = 70 แล้วเราจะมาพิจารณาใช้กฎนั้น โดยนำเอา เพรดดิเคตหลัง ID# > 70000 มาพิจารณาทำการวนลูปเพื่อตรวจสอบเงื่อนไขทั้งหมดของคำสั่งที่รับเข้ามา ตรวจสอบดูว่ามีเพรดดิเคต ไหนมีค่าแอททริบิวเดียวกันกับ เพรดดิเคตหลังของกฎ หากพบก็จะทำการตรวจสอบดูว่าจะทำการลดช่วงโดยวิธีการ Scan Reduction หรือไม่ แต่ถ้าไม่พบก็จะมาพิจารณาว่าเพรดดิเคตนั้นมีแอททริบิวเป็นอินเดกซ์ (Index Key) หรือไม่ถ้าเป็นก็จะทำการใส่เพรดดิเคตที่เป็นอินเดกซ์เข้าไป จากที่ทดลองนั้นพบเพรดดิเคตที่สอดคล้องกับเพรดดิเคตหลังของกฎ ID# > 70000 คือ ID# < 50000 นำเพรดดิเคตนี้ไปตรวจเช็คว่ามีค่าของเงื่อนไขที่ต้องการนั้นขัดแย้งหรือไม่ จะพบว่าขัดแย้งเนื่องจากกฎบอกว่าถ้า campus = 70 แล้ว ID# ต้องมีค่ามากกว่า 70000 แล้วคำสั่งที่ต้องการ ID# < 50000 มีค่าไม่ตรงที่กำหนดจึงขัดแย้งก็ให้บอกกับโปรแกรมหลัก เพื่อไม่ต้องทำการส่งคำสั่งนี้ไปประมวลผลที่ระบบจัดการฐานข้อมูล

ตัวอย่างที่ 2

คำสั่งคิวรีของผู้ใช้ :

```
select *
From classexam
Where examtimefrom = 17.00 and
examtimeto = 13.00
```

กฎข้อบังคับ : examtimefrom < examtimeto

ในกรณีนี้คำสั่งคิวรีขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรีนั้นไปประมวลผล

ตารางที่ 5.2 Response time จากหลักการ Detection of unsatisfiable conditions 2

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.365	0.683	0.445	0.591	0.702
คำสั่งคิวรีที่ถูกแปลง	0	0	0	0	0

ตัวอย่างที่ 3

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where total_credit = 100
```

กฎข้อบังคับ : total_credit < = 88

ในกรณีนี้คำสั่งคิวรีขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรีนั้นไปประมวลผล

ตารางที่ 5.3 Response time จากหลักการ Detection of unsatisfiable conditions 3

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.432	0.256	0.435	0.154	0.354
คำสั่งคิวรีที่ถูกแปลง	0	0	0	0	0

ในกรณีนี้ เมื่อไม่พบข้อมูลอยู่ในฐานข้อมูล เนื่องจากทราบจากกฎประเภทกำหนดค่าขอบเขตของแอททริบิว ก็จะไม่ต้องส่งคำสั่งลงไปประมวลผลที่ระบบจัดการฐานข้อมูล

ในกรณีนี้ทำเกิดในกรณีที่ตรวจสอบเจอตั้งแต่การตรวจเงื่อนไขที่ขัดแย้งกับกฎประเภทกำหนดค่าขอบเขตของแอททริบิว โดยกฎมีอยู่ว่า total_credit <= 88 แต่คำสั่งที่ใส่เข้ามาต้องการหาข้อมูลที่มีค่า total_credit เท่ากับ 100 เมื่อนำมาเปรียบเทียบกันแล้วจะพบว่าเกิดความขัดแย้ง จึงทำการส่งไปบอกผู้ใช้งานว่าหาข้อมูลไม่พบโดยที่ไม่ต้องส่งคำสั่งไปประมวลผล

2. กรณีที่คำสั่งคิวรีของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด โดยเงื่อนไขของคำสั่งคิวรีของผู้ใช้ ซ้ำซ้อนกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด ซึ่งจะใช้ทฤษฎี Predicate Elimination เพื่อกำจัดเงื่อนไขที่ซ้ำซ้อนออกไปจากคำสั่งคิวรี

ตัวอย่างที่ 1

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where gradstatus = 'W' and studentstatus  < 60
```

กฎข้อบังคับ : gradestatus = 'W' ⟶ studentstatus < 40

คำสั่งคิวรีที่ถูกแปลง :

```
select *
From student
Where gradestatus = 'W'
```

ตารางที่ 5.4 Response time จากหลักการ Predicate Elimination 1

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.367	0.289	0.325	0.198	0.337
คำสั่งคิวรีที่ถูกแปลง	0.211	0.163	0.249	0.132	0.170

ตัวอย่างที่ 2

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where fac = '03' and level > 1
```

กฎข้อบังคับ : fac = '03' ⟶ level > 3

คำสั่งคิวรีที่ถูกแปลง :

```
select *
From student
Where fac = '03'
```

ตารางที่ 5.5 Response time จากหลักการ Predicate Elimination 2

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.312	0.296	0.314	0.298	0.308
คำสั่งคิวรีที่ถูกแปลง	0.125	0.212	0.302	0.128	0.149

ในกรณีนี้เป็นการตัดเพรดดิเคตที่ซ้ำซ้อนออก และทำการเพิ่มเพรดดิเคตที่มีแอททริบิวเป็นอินเดกซ์เข้าไปจะทำให้เร็วขึ้น

โดยขั้นแรกนั้นทำการตรวจว่าคำสั่งที่รับเข้ามานั้นมีเพรดดิเคตที่สอดคล้องกับกฎหรือไม่ เช่น Where fac = '03' and level > 1 จะพบว่ามีกฎอยู่ 1 กฎ ที่ถูกนำมาใช้

FAC = '03' ⟶ LEVEL > 3

ซึ่งจะเริ่มตรวจสอบเพรดดิเคตแรก FAC = '03' ครอบคลุมเพรดดิเคตที่เข้ามา คือ FAC = '03' แล้วเราจะมาพิจารณาใช้กฎนั้น โดยนำเอา เพรดดิเคตหลัง LEVEL > 3 มาพิจารณา ทำการวนลูปเพื่อตรวจสอบเงื่อนไขทั้งหมดของคำสั่งที่รับเข้ามา ตรวจเช็คดูว่ามีเพรดดิเคตไหน มีค่าแอททริบิวเดียวกันกับเพรดดิเคตหลังของกฎ หากพบก็จะทำการตรวจสอบดูว่าจะทำการลดช่วงโดยวิธีการ Scan Reduction หรือไม่แต่ถ้าไม่พบก็จะมาพิจารณาว่าเพรดดิเคตนั้นมีแอททริบิวเป็นอินเดกซ์ (Index Key) หรือไม่ ถ้าเป็นก็จะทำการใส่เพรดดิเคตที่เป็นอินเดกซ์เข้าไป มาพิจารณาว่า เพรดดิเคต

ใดในคำสั่งที่รับเข้ามา เกี่ยวข้องกับ level ก็จะพบ level >1 ของคำสั่งคิวรี ดังนั้นก็นำค่าทั้งสองมาตรวจสอบกันว่าขัดแย้งหรือไม่ จะพบว่า สามารถตัดเพรดดิเคตของคำสั่งคิวรี level > 1 ออกได้ เนื่องจากว่าในกฎมีการระบุไว้แล้วว่า LEVEL จะมากกว่า 3 ในกรณีที่ FAC เท่ากับ '03'

3. กรณีที่คำสั่งคิวรีผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับ ที่ผู้ดูแลระบบกำหนด โดยเพิ่มเงื่อนไขใหม่ที่มาจากกฎข้อบังคับ เข้าไปในคำสั่งคิวรีของผู้ใช้ โดยเงื่อนไขใหม่นั้นจะต้องมีแอททริบิวเป็นอินเดกซ์อยู่ด้วย โดยใช้ทฤษฎี index introduction

ตัวอย่างที่ 1

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where fac = '05'
```

กฎข้อบังคับ : fac = '05' ⟶ fid =10

โดยที่ fid เป็นอินเดกซ์

คำสั่งคิวรีที่ถูกแปลง :

```
select *
From student
Where fac = '05' and fid = 10
```

ตารางที่ 5.6 Response time จากหลักการ index introduction 1

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.343	0.321	0.315	0.344	0.356
คำสั่งคิวรีที่ถูกแปลง	0.141	0.140	0.145	0.153	0.162

ตัวอย่างที่ 2

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where major = 'IE'
```

กฎข้อบังคับ : major ='IE' ⟶ campus < 50

โดยที่ campus เป็นอินเดกซ์

คำสั่งคิวรีที่ถูกแปลง : select *

From student

Where major ='IE' and campus < 50

ตารางที่ 5.7 Response time จากหลักการ index introduction 2

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.378	0.432	0.353	0.389	0.418
คำสั่งคิวรีที่ถูกแปลง	0.137	0.154	0.142	0.166	0.142

ในกรณีนี้เป็นการตัดเพรดดิเคตที่ซ้ำซ้อนออก และทำการเพิ่มเพรดดิเคตที่มีแอททริบิวเป็นอินเด็กซ์เข้าไปจะทำให้เร็วขึ้น

โดยขั้นแรกนั้นทำการตรวจว่าคำสั่งที่รับเข้ามานั้นมีเพรดดิเคตที่สอดคล้องกับกฎหรือไม่ เช่น Where major = 'IE' จะพบว่ามีกฎอยู่ 1 กฎ ที่ถูกนำมาใช้

major ='IE' ⟶ campus < 50

แล้วจะเริ่ม ทำการวนลูปเพื่อตรวจสอบเงื่อนไขทั้งหมดของคำสั่งที่รับเข้ามา ตรวจเช็คดูว่ามีเพรดดิเคตไหน มีค่าแอททริบิวเดียวกันกับเพรดดิเคตหลังของกฎ หากพบก็จะทำการตรวจสอบดูว่าจะทำการลดช่วงโดยวิธีการ Scan Reduction หรือไม่แต่ถ้าไม่พบก็จะมาพิจารณาว่าเพรดดิเคตนั้นมีแอททริบิวเป็นอินเดกซ์(Index Key) หรือไม่ถ้าเป็นก็จะทำการใส่เพรดดิเคตที่เป็นอินเด็กซ์เข้าไป จากที่ทดลองนั้นไม่มีเพรดดิเคตที่สอดคล้องกับเพรดดิเคตหลังของกฎ campus < 50 จึงนำเพรดดิเคตนี้ไปตรวจเช็คว่ามีแอททริบิวที่เป็นอินเด็กซ์หรือไม่ ในกรณีนี้ campus เป็นอินเดกซ์ จึงทำการเพิ่มเพรดดิเคตนี้เข้าไปในคำสั่งคิวรี

4. กรณีคำสั่งคิวรีของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้ และสามารถนำกฎข้อบังคับมาทำการลดช่วงการค้นหาได้ โดยใช้ทฤษฎี Scan reduction

ตัวอย่างที่ 1

คำสั่งคิวรีของผู้ใช้ : select *

From student

Where major ='IE' and campus > 30

กฎข้อบังคับ : major ='IE' ⟶ campus < 50

คำสั่งคิวรีที่ถูกแปลง :

```
select *
From student
Where major ='IE' and campus >30
and campus < 50
```

ตารางที่ 5.8 Response time จากหลักการ Scan reduction 1

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.296	0.297	0.313	0.297	0.298
คำสั่งคิวรีที่ถูกแปลง	0.241	0.288	0.281	0.282	0.281

ตัวอย่างที่ 2

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where major ='BU' and programid > 300
```

กฎข้อบังคับ : major = 'BU' $\longrightarrow$ 200< programid < 500

คำสั่งคิวรีที่ถูกแปลง :

```
select *
From student
Where major ='BU' and programid >300
and programid < 500
```

ตารางที่ 5.9 Response time จากหลักการ Scan reduction 2

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.389	0.348	0.372	0.359	0.377
คำสั่งคิวรีที่ถูกแปลง	0.321	0.316	0.343	0.324	0.352

ในกรณีนี้เป็นการใช้วิธีการเพิ่มเพรดดิเคตเข้ามาเพื่อทำการลดช่วงการค้นหา ทำให้ขอบเขตของการค้นหาลดน้อยลง

โดยเริ่มจากการตรวจสอบที่เพรดดิเคตแรกของคำสั่งที่เข้ามา คือ major = 'BU' จะได้กฎที่นำมาตรวจสอบคือ

major = 'BU' $\longrightarrow$ 200< programid < 500

ดังนั้นมาพิจารณาที่เพรดดิเคตตัวหลังของกฎ คือ 200< programid < 500 นำมาหาว่ามีเพรดดิเคต ไหนในคำสั่งที่สอดคล้องกันก็พบว่ามีเพรดดิเคต programid >300 จะทำการตรวจสอบว่ามีค่าขัดแย้งหรือไม่ ในกรณีนี้ไม่ขัดแย้งและพบว่า เข้ากรณีการเพิ่มเพรดดิเคตที่ทำการลดช่วง เนื่องจากมีเพรดดิเคตที่มีชื่อแอททริบิวนั้นมีอยู่แล้วในคำสั่งที่เข้ามา ทำให้เพิ่ม programid < 500 เข้าไปในคำสั่งที่เข้ามา

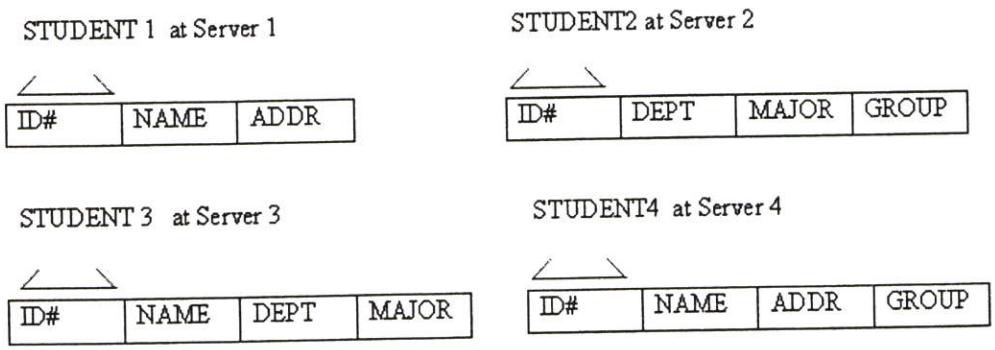
5. กรณีคำสั่งคิวรีของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้ และสามารถนำกฎข้อบังคับมากำหนดให้ค้นหาเฉพาะไซต์ที่มีข้อมูลอยู่จริง โดยใช้เงื่อนไขแบบ Site constraint

ตัวอย่างที่ 1

คำสั่งคิวรีของผู้ใช้ :

```
select *
From student
Where  group = 'ช่างอุตสาหกรรม'
```

กฎข้อบังคับ : group = 'ช่างอุตสาหกรรม' ⟶ site ='S2'



รูปที่ 5.1 แสดงโครงสร้าง Mixed Fragmentation

เมื่อเข้าสู่กระบวนการของระบบฐานข้อมูลแบบกระจายซึ่งมีโครงสร้างดังรูปที่ 5.1 ปกติเมื่อสั่งให้ส่งคำสั่งคิวรี ก็จะถูกแยกออกเป็นสองคำสั่งประกอบด้วย 2 view ดังนี้

```
View S2
SELECT T1.ID#, T1.NAME, T1.ADDR, T2.DEPT, T2.MAJOR, T2.GROUP
FROM STUDENT1 T1, STUDENT2 T2
WHERE T1.ID#=T2.ID# and  T2.GROUP = 'ช่างอุตสาหกรรม';
Union
```

View S4

SELECT T3.ID#, T3.NAME, T3.DEPT, T3.MAJOR, T4.ADDR, T4.GROUP

FROM STUDENT1 T3, STUDENT2 T4

WHERE T3.ID#=T4.ID# and T4.GROUP = 'ช่างอุตสาหกรรม';

คำสั่งคิวรีที่ถูกแปลง :

select *

From S2

Where group = 'ช่างอุตสาหกรรม'

ตารางที่ 5.10 Response time จากกฎข้อบังคับ site constraint

	ครั้งที่ 1 (s)	ครั้งที่ 2 (s)	ครั้งที่ 3 (s)	ครั้งที่ 4 (s)	ครั้งที่ 5 (s)
คำสั่งคิวรีเดิม	0.489	0.472	0.503	0.497	0.434
คำสั่งคิวรีที่ถูกแปลง	0.295	0.285	0.311	0.318	0.281

ในกรณีนี้เป็นการตัดเซิร์ฟเวอร์ที่ไม่มีข้อมูลตามที่คำสั่งต้องการอยู่ออกไป โดยขั้นตอนแรกนั้นตรวจสอบกฎข้อบังคับ site constraint ที่มีอยู่ พบว่ามี 1 กฎ ที่ถูกนำมาใช้

group = 'ช่างอุตสาหกรรม' ⟶ site ='S2'

เมื่อพบกฎ และทราบลักษณะเฉพาะก็จะทำการแปลงคำสั่ง จาก ตารางเดิมเป็น ตารางที่มีข้อมูล ตามที่กฎกำหนด จาก เพลทดิเคต site = 'S2' จึงทำการเปลี่ยนชื่อตารางจาก student ไปเป็นชื่อ view S2 ซึ่งจะค้นหาข้อมูลเฉพาะไซต์ S2 เท่านั้น

View S2

SELECT T1.ID#, T1.NAME, T1.ADDR, T2.DEPT, T2.MAJOR, T2.GROUP

FROM STUDENT1 T1, STUDENT2 T2

WHERE T1.ID#=T2.ID# and T2.GROUP = 'ช่างอุตสาหกรรม';

บทที่ 6
สรุปผลการวิจัย และข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอระบบฐานข้อมูลแบบกระจายที่มีการตัดสินใจค้นหาข้อมูลเชิงความหมาย เนื่องจากระบบฐานข้อมูลแบบกระจายมีข้อมูลกระจายอยู่ในแต่ละไซต์ และไซต์เหล่านั้นมีระบบจัดการฐานข้อมูลและฮาร์ดแวร์ที่แตกต่างกัน ซึ่งการเรียกค้นข้อมูลในแต่ละไซต์ที่อยู่ต่างสถานที่กันจะทำให้เกิดประสิทธิภาพในการเรียกค้นข้อมูลลดลงและยังเกิดค่าใช้จ่ายในการส่งข้อมูลผ่านระบบเครือข่ายที่มากขึ้น ซึ่งบางครั้งคำสั่งคิวรีที่ส่งออกไปยังไซต์ต่างๆ นั้น อาจจะไม่มีผลลัพธ์แสดงออกมา เนื่องจากไม่มีข้อมูลที่ผู้ใช้ต้องการค้นหาอยู่ในฐานข้อมูล /

นอกจากนี้วิธีการค้นหาข้อมูลเชิงความหมายที่ใช้วิธีการสร้างกฎข้อบังคับ ในระดับต่างๆ ของฐานข้อมูลแบบกระจาย เพื่อการปรับปรุงคำสั่งคิวรีที่เข้าไปจัดการข้อมูลในระดับแฟรกเมนท์และระดับออโลเคชั่นบนฐานข้อมูลแบบกระจายด้วย เพื่อเพิ่มประสิทธิภาพในการจัดการข้อมูลบนฐานข้อมูลแบบกระจาย จากการทดลองพบว่าระบบฐานข้อมูลแบบกระจายที่มีการค้นหาข้อมูลเชิงความหมาย สามารถเพิ่มประสิทธิภาพในการจัดการข้อมูลได้ดี ในระดับโกลบอล

6.2 ข้อเสนอแนะ

1. จากการทดลองพบว่าบางครั้งผลการทดลองที่ได้ ก็อาจจะขึ้นอยู่ ข้อมูลที่ใช้ในการทดลองและสภาพแวดล้อมอื่นๆ

2. จากการทดลองพบว่าการสร้างฐานข้อมูลแบบกระจายบนระบบปฏิบัติการฐานข้อมูลสามารถทำได้ แต่การ update ,insert และ delete ยังต้องมีการจัดการเพิ่มเติมในบางส่วน

เอกสารอ้างอิง

[1] Abraham Silberschatz, Henry F. Korth and S. Sudarshan : "Database System Concepts, Fourth Edition.", McGraw-Hill, 2002

[2] I. K. Ibrahim, V. Dignum, W. Winiwarter, E. Weippl, "Logic Based Approach to Semantic Query Transformation for Knowledge Management Applications" Proc. of the International Conference on Knowledge Management, Berlin, Springer-Verlag, 2002.

[3] P. Godfrey, J. Grant, J. Gryz, J. Minker, "Integrity Constraints: Semantics and Applications", Logics for Databases and Information Systems,1998, pp 265-306

[4] S.T. Shenoy and Z.M. Ozsoyoglu, "Design and implementation of semantic query optimizer", IEEE Transactions on Knowledge and Data Eng., 1989, pp 344-361.

[5] B.G.T. Lowden, J. Robinson, K.Y. Lim, "A Semantic Query Optimiser Using Automatic Rule Derivation", Proc. WITS '95, 5th International Workshop on Information Technologies and Systems, 1995, pp 68-76.

[6] Wei Sun and Clement T. Yu, "Semantic Query Optimization for Tree and Chain Queries", IEEE Transactions on Knowledge and Data Eng. 1994, pp 136-151

[7] Shashi Shekhar, Babak Hamidzadeh, Ashim Kohli, Mark Coyle, "Learning Transformation Rules for Semantic Query Optimization : A Data-Driven Approach", IEEE Transactions on Knowledge and Data Eng. 1993, pp 950-964

[8] Jerome Robinson, Barry G. T. Lowden, "Semantic Query Optimisation and Rule Graphs", Proceedings of the 5th KRDB workshop, 1998: 14.1-14.10

[9] M. F. Van Bommel, "Semantic Query Optimization for ODMG-93 Databases" , Proceedings of the 1999 International Symposium on Database Engineering & Applications, 1999, pp 16

[10] Chun-Nan Hsu , Craig A. Knoblock, "Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems", IEEE Transactions on Knowledge and Data Eng, 2000 , pp 959- 978

[11] John Cardiff , "Semantic Query Optimization in Heterogeneous DBMSs", Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994 ,pp 273-282

[12] Prabhu Ram, William Perrizo, "Multidatabase Global Query Optimization", Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1995, pp 253 - 262

ภาคผนวก

The 10th World Multi-Conference on Systemics, Cybernetics and Informatics

July 16-19, 2006 ~ Orlando, Florida, USA

Jointly with
The 12th International Conference on
Information Systems Analysis and Synthesis: ISAS 2006

PROCEEDINGS

Volume I

Edited by

Nagib Callaos
William Lesso
Wanju Bo
Aleksandras Rutkauskas

Organized by
International Institute of Informatics and Systemics
Member of the International Federation of Systems Research (IFSR)

Global Semantic Query Optimizer in Multidatabase Systems

Nachirat RACHBURI
Department of Computer Engineering, King Mongkut's Institute of Technology
Ladkrabang, Bangkok 10520, Thailand

and

Suphamit CHITTAYASOTHORN
Department of Computer Engineering, Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

ABSTRACT

At present, multidatabase systems provide access to local databases via gateway protocols. Users have to be aware of product-specific features of local database management systems. Query optimization in multidatabase systems are normally based on statistics collected at local sites. This paper describes a multidatabase system that supports global schemas, fragmentation schemas and allocation schemas together with the semantic query optimization capability. The system also supports data manipulations on global databases with mixed fragmentation. The semantic query optimization feature proved to be very useful and helps reduce response time of global queries.

Keywords: multidatabase, semantic query optimization, global schema, fragmentation, allocation.

1. INTRODUCTION

Semantic query optimization is a research topic that applies application-specific rules to the query optimization process for more efficient searches of queries result. The rules can be related to both physical and logical database structures. Physical-level rules are about access mechanisms which are available on certain attributes of a relation. Queries on attributes without such access mechanisms can be transformed into ones that refer to attributes with the mechanisms using data dependencies between attributes. Logical-level rules are integrity constraints which are commonly used as validation rules. Data manipulation activities are validated against these rules and the ones which fail to pass the validation will be rejected thus guarantee the integrity of the database. The semantic query optimizer takes advantage of the fact that the database must follows integrity rules by checking incoming queries against the rules and rejects queries that are out of bound. The semantic query optimization topic is even more interesting when the database systems are distributed multidatabase systems. Site-specific integrity rules must be known to other sites so that subqueries of a global query need not be sent to some sites whose integrity rules are outside of the scope of the query [1].

This work presents the design and implementation of a global semantic query optimizer in a multidatabase system. The operating environment is a multi-campus university system comprises eight universities and thirty nine campuses. The campuses have their own specialties and expertise and thus offer different variety of degrees. Each campus has a local computer center for processing of local applications. The central administration of this university system issues global-level queries to the global semantic optimizer which in turn, decides the appropriate sites that subqueries should be sent to. This global optimizer utilizes integrity rules and database statistics periodically collected from each site. The subqueries sent to the sites will further be locally optimized by the local DBMS at each site. The test result shows significant improvement of the response times [12].

The system architecture comprises the global schema which is the top most level logical structure of the system. Users see the entire multidatabase system as a centralized system with complete transparency. Global-level requests are translated to fragment-level requests and finally to the local-level requests. Semantic query optimization is done at the global level. However, if permitted by the DBA, users who wish to access the database at fragment and location levels may choose to do so. Local applications run undisturbed on local servers.

2. SEMANTIC QUERY OPTIMIZATION

Semantic query optimization is the application of semantic knowledge for the transformation of simple queries to higher performance ones. The semantic knowledge may be from the user's application knowledge or from the system's own information. Our system employs several following techniques for the optimization of global queries.

2.1) Site Constraints

In the case that some local database sites have their own special characteristics, we can exploit these characteristics by creating integrity constraints describing the characteristics. For example, a search for students that belong to the college of engineering should not go to all database servers but should be directed to the server that has the tables for engineering students instead.

2.2) Functional Dependencies

Functional dependencies can help reduce complex queries into simple ones. Unnecessary conditions can make a query too complex and requires much optimization effort by the underlying cost-based optimizer. Functional dependency is also used to find a better access path from conditions that refer to non-indexed conditions which determine other conditions which are indexed. In this case, the indexed conditions are used instead of the non indexed ones [4], [5], [10], [11].

2.3) Join Constraints

Join constraints include both join elimination and join introduction. Join elimination is for tables that have referential integrity enforcement. In the case that there is a condition which checks the existence of the corresponding primary key value on the main parent table and the foreign key attribute is not null, it is always true that a corresponding primary key exists and the checks can be eliminated. On the other hands, join introduction could be an advantage if the joined attributes are indexed and also refer to other indexed attributes [2], [3], [6], [7], [8], [9].

2.4) Global Statistics

Statistics on database tables include the size of the tables, number of distinct values for each attribute, blocking factors and indexes. These traditional statistics are for cost-based optimization. It helps the system make decision at the global level and break a global query into several smaller queries for each fragment. These

fragment-level queries will further be optimized locally by the local DBMS at each site.

3. Multidatabase Data Architecture

The main purpose of the system is to have a complete global database that hides all fragmentation, allocation and specific DBMS implementation details from the users. The system must have the semantic optimization capability so that a global query can be decomposed into fragment and site dependent queries in an intelligent manner thus improve the global query response time. It is also possible that the fragmentation is a mixed fragmentation and the system must be able to handle he updates that require physical movement of the data over the sites. In this case distributed transaction processing is the issue.

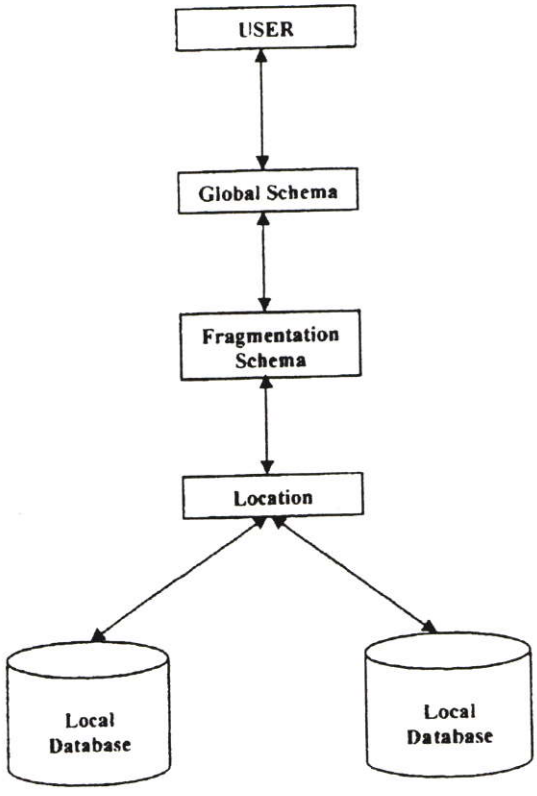


Figure. 1 The Multidatabase Architecture.

Figure.1 shows the simple but challenging multidatabase data architecture. The global schema describes the entire logical data structures of the database. The relational database model is employed here. This global schema can be logically decomposed into fragments. In practice, database tables can be fragmented by rows according to some classifications such as geographical areas, departments or some kinds of categories. This is technically called horizontal fragmentation. Database tables can also be fragmented by columns. This is normally the case of application requirements. Different applications may require different attributes. This kind of fragmentation is called vertical fragmentation. It is possible that the table is first fragmented horizontally and then further fragmented vertically. This is the case of the mixed fragmentation.

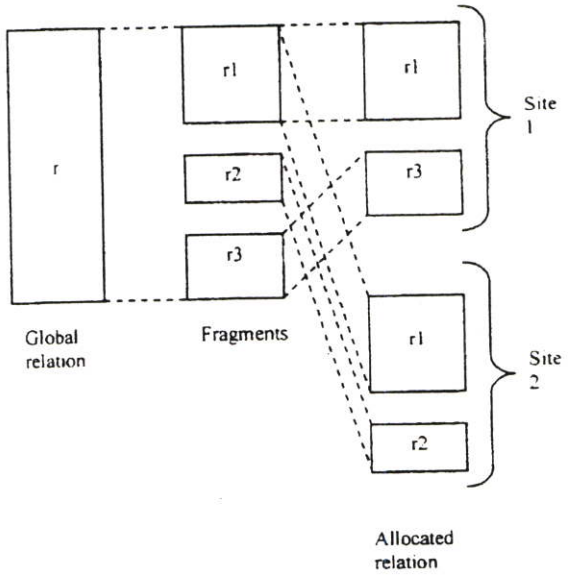


Figure.2 A sample fragmentation and allocation.

Fragments can be physically replicated to different sites. Each site may use different DBMS. If an update takes place on a fragment, all its replicated copies must be updated in a synchronous manner. This synchronous distributed transaction requires at least the two-phase commit protocol. Figure.2 shows the concepts of global relation, fragments and allocations. The relation r is horizontally fragmented in to three fragments two of which are allocated at each site.

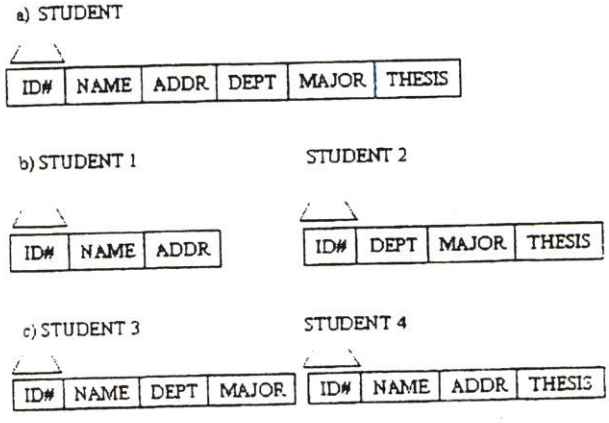


Figure.3 A Mixed Fragmentation.

4. Operations on Mixed Fragmentation

A challenging task in the implementation of mixed fragmentation is the update of a global relation with mixed fragmentation. A simple update statement at the global level could be translated into several select, insert and delete statements at multiple sites. Figure.3a shows a global relational schema STUDENT which is assumed to have two horizontal relation fragments. The first fragment contains student records for students whose department number is between 1 and 10. The second one contains student records whose department number is between 11 and 20.

Figure. 3b shows the first horizontal fragment with further vertical fragments STUDENT1 and STUDENT2. These fragmentations are different from the vertical fragmentations STUDENT3 and STUDENT4 shown in Figure 3c. Simple updates made to the global relation STUDENT may become a collection of select, insert and delete operations an illustrated example in SQL is shown below. Suppose a student whose ID#=12345 would like to move from DEPT 2 to DEPT 11. The global update request is a very simple SQL statement as follow:

```
UPDATE STUDENT
SET DEPT=11
WHERE ID#=12345;
```

Since the student is in DEPT 2 before the update, his database record is in the first horizontal fragment. This record has to be moved to the second fragment and possibly a different database server. So, a SELECT is required to retrieve the student's record from the relations of STUDENT1 and STUDENT2. The result will then be inserted into the relations of STUDENT3 and STUDENT4. Delete operations on the relations of STUDENT1 and STUDENT2 will complete the fragment-level transaction. The SQL statements of the transaction are shown as follow:

```
SELECT ID#, NAME, ADDR INTO VAR1, VAR2, VAR3
FROM STUDENT1
WHERE ID#=12345;

SELECT DEPT, MAJOR, THESIS INTO VAR4, VAR5, VAR6
FROM STUDENT2
WHERE ID#=12345;

INSERT INTO STUDENT3 VALUE (VAR1, VAR2, VAR4,
VAR5);

INSERT INTO STUDENT4 VALUE (VAR1, VAR2, VAR3,
VAR6);

DELETE FROM STUDENT1
WHERE ID#=12345;

DELETE FROM STUDENT2
WHERE ID#=12345;
```

In the more physical case of the transaction that sees the actual allocation on each location which could be replicated, each and every site involved must be referred to. The fragment STUDENT1 is allocated at SERVER3. The fragment STUDENT2 is allocated at SERVER4. The fragment STUDENT3 is allocated at SERVER1 and SERVER2. The fragment STUDENT4 is also allocated at SERVER1 and SERVER2. The distributed transaction is as detail as follow:

```
SELECT ID#, NAME, ADDR INTO VAR1, VAR2, VAR3
FROM STUDENT1 AT SERVER3
WHERE ID#=12345;

SELECT DEPT, MAJOR, THESIS INTO VAR4, VAR5, VAR6
FROM STUDENT2 AT SERVER4
WHERE ID#=12345;

INSERT INTO STUDENT3 AT SERVER1 VALUE (VAR1,
VAR2, VAR4, VAR5);

INSERT INTO STUDENT4 AT SERVER1 VALUE (VAR1,
VAR2, VAR3, VAR6);

INSERT INTO STUDENT3 AT SERVER2 VALUE (VAR1,
VAR2, VAR4, VAR5);

INSERT INTO STUDENT4 AT SERVER2 VALUE (VAR1,
VAR2, VAR3, VAR6);

DELETE FROM STUDENT1 AT SERVER3
WHERE ID#=12345;

DELETE FROM STUDENT2 AT SERVER4
WHERE ID#=12345;
```

Since all these SQL statements belong to the same transaction, a distributed transaction manager is required to maintain transaction atomicity in the multidatabase environment.

Not only the global update statements that could lead to more complex operations at lower levels, a simple SQL SELECT statement at the global level may lead to several SQL statements at the fragmentation level and even more at the allocation level.

For example, consider a simple search for a student record by the primary key ID#.

```
SELECT *
FROM STUDENT
WHERE ID#=12345;
```

At the fragment level this could lead to

```
SELECT T1.ID#, NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT1 T1, STUDENT2 T2
WHERE T1.ID#=T2.ID# AND T1.ID#=12345;
If not found then
SELECT T1.ID#, T1.NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT3 T3, STUDENT4 T4
WHERE T3.ID#=T4.ID# AND T3.ID#=12345;
```

At the allocation level, the query refers to sites. In the case of replications, it will refer to the nearest site that has the require table. The above query is finally translated to the following one.

```
SELECT T1.ID#, NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT1 AT SERVER3 T1, STUDENT2 AT
SERVER4 T2
WHERE T1.ID#=T2.ID# AND T1.ID#=12345;
If not found then
SELECT T1.ID#, T1.NAME, ADDR, DEPT, MAJOR, THESIS
FROM STUDENT3 AT SERVER1 T3, STUDENT4 AT
SERVER1 T4
WHERE T3.ID#=T4.ID# AND T3.ID#=12345;
```

5. Some Semantic Query Optimization Experimental Results

Some experiments have been conducted to test the performance of the global queries. A query is tested with and without the semantic optimizer on a multidatabase system. The test results show significant improvement of response times.

5.1) Detection of Unsatisfiable Conditions

This is the case where the input query is against an integrity rule. In this case, the global optimizer will not send the request to any database servers.

For example, the query is

```
SELECT *
FROM STUDENT
WHERE CAMPUS =70 AND ID# < 50000.
```

The integrity rule: If CAMPUS = 70 THEN ID# > 7000

In this case the query is against the integrity rule and there will be no rows returned.

	1 (s)	2 (s)	3 (s)	4 (s)	5 (s)
Original Query	0.453	0.531	0.336	0.843	0.734
Modified Query	0	0	0	0	0

5.2) Predicate Elimination

This the case where there are too many predicates in the search condition of a query. Some extra predicates can be eliminated.

For example, the query is

SELECT *
FROM STUDENT
WHERE FAC='03' AND LEVEL > 1;
The integrity rule: IF FAC='03' THEN LEVEL > 3

The query is transformed into

SELECT *
FROM STUDENT
WHERE FAC = '03';

The predicate LEVEL > 1 is eliminated because the fact that FAC='03' implies that LEVEL must be > 3. Test results show the improvement of the response time.

	1 (s)	2 (s)	3 (s)	4 (s)	5 (s)
Original Query	03.12	0.296	0.314	0.298	0.308
Modified Query	0.125	0.212	0.302	0.128	0.149

5.3) Index Introduction

This case is the case where a new condition is introduced because the condition involves a predicate which is indexed. The query optimizer can exploit the index on the new search condition and improve the response time.

For example, the query is

SELECT *
FROM STUDENT
WHERE FAC = '05';

The integrity rule: IF FAC='05' THEN FID=10 and FID is indexed.

The query is transformed into

SELECT *
FROM STUDENT
WHERE FAC='05' AND FID=10.

This query takes advantage of the index on FID to improve the query response time. The test result shows a significant improvement

	1 (s)	2 (s)	3 (s)	4 (s)	5 (s)
Original Query	0.343	0.321	0.315	0.344	0.356
Modified Query	0.141	0.140	0.145	0.153	0.162

5.4) Scan Reduction

This is the case where integrity rules can be used to reduce the scan intervals.

For example, the query is

SELECT *
FROM STUDENT
WHERE MAJOR ='IE' AND CAMPUS > 30;

Integrity Rule: IF MAJOR ='IE' THEN CAMPUS < 50

The query is transformed into

SELECT *
FROM STUDENT
WHERE MAJOR ='IE' AND CAMPUS > 30
AND CAMPUS < 50;

The result show slight improvement of the response time. This is due to the fact that the scan range is reduced.

	1 (s)	2 (s)	3 (s)	4 (s)	5 (s)
Original Query	0.296	0.297	0.313	0.297	0.298
Modified Query	0.241	0.288	0.281	0.282	0.281

6. Conclusions

This paper presents a multidatabase system that supports semantic query optimization at the global level. Data architecture of the system and illustrated examples of query requests on global, fragment and allocation are given. Update operation that leads to distributed transaction that requires at least the 2-phase commit protocol is also illustrated.

7. References

[1] Abraham Silberschatz, Henry F. Korth and S. Sudarshan "**Database System Concepts, Fourth Edition.**", McGraw-Hill, 2002

[2] I. K. Ibrahim, V. Dignum, W. Winiwarter, E. Weippl, "Logic Based Approach to Semantic Query Transformation for Knowledge Management Applications" **Proc. of the International Conference on Knowledge Management**, Berlin, Springer-Verlag, 2002

[3] P. Godfrey, J. Grant, J. Gryz, J. Minker, "Integrity Constraints, Semantics and Applications", **Logics for Databases and Information Systems**, 1998, pp 265-306

[4] S.T. Shenoy and Z.M. Ozsoyoglu, "Design and implementation of semantic query optimizer", **IEEE Transactions on Knowledge and Data Eng.**, 1989, pp 344-361.

[5] B.G.T. Lowden, J. Robinson, K.Y. Lim, "A Semantic Query Optimiser Using Automatic Rule Derivation", **Proc. WITS '95, 5th International Workshop on Information Technologies and Systems**, 1995, pp 68-76.

[6] Wei Sun and Clement T. Yu, "Semantic Query Optimization for Tree and Chain Queries", **IEEE Transactions on Knowledge and Data Eng**, 1994, pp 136-151

[7] Shashi Shekhar, Babak Hamidzadeh, Ashim Kohli, Mark Coyle, "Learning Transformation Rules for Semantic Query Optimization : A Data-Driven Approach", **IEEE Transactions on Knowledge and Data Eng**, 1993, pp 950-964

[8] Jerome Robinson, Barry G. T. Lowden, "Semantic Query Optimisation and Rule Graphs", **Proceedings of the 5th KRDB workshop**, 1998: 14.1-14.10

[9] M. F. Van Bommel, "Semantic Query Optimization for ODMG-93 Databases" , **Proceedings of the 1999 International Symposium on Database Engineering & Applications**, 1999, pp 16

[10] Chun-Nan Hsu , Craig A. Knoblock, "Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems", **IEEE Transactions on Knowledge and Data Eng**, 2000 , pp 959- 978

[11] John Cardiff , "Semantic Query Optimization in Heterogeneous DBMSs", **Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences**, 1994 ,pp 273-282

[12] Prabhu Ram, William Per&o, "Multidatabase Global Query Optimization", **Proceedings of the 28th Annual Hawaii International Conference on System Sciences**, 1995, pp 253 - 262

ประวัติผู้เขียน

ชื่อ-สกุล	นายนชิรัตน์ ราชบุรี
วันเดือนปีเกิด	วันที่ 5 มกราคม พ.ศ. 2520 ณ จังหวัดกำแพงเพชร
ที่อยู่	110 ม. 3 ต.ลานกระบือ อ.ลานกระบือ จ.กำแพงเพชร 62170
ประวัติการศึกษา	
พ.ศ. 2542	สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ศูนย์กลางสถาบันเทคโนโลยีราชมงคล
พ.ศ. 2545	เข้าศึกษาต่อในระดับวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ประสบการณ์ทำงาน	
พ.ศ. 2542-2549	เข้ารับราชการ สำนักเทคโนโลยีสารสนเทศ ศูนย์กลางสถาบันเทคโนโลยีราชมงคล
พ.ศ. 2549-2550	เข้ารับราชการตำแหน่งอาจารย์ประจำภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีราชมงคล ธัญบุรี