ต้นแบบตัวประมวลคำถามเชิงความหมาย

A PROTOTYPE SEMANTIC QUERY OPTIMIZER

พีรยา มีนาภินันท์

พุทธิพงศ์ ธนูเทพ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ต้นแบบตัวประมวลคำถามเชิงความหมาย
A PROTOTYPE SEMANTIC QUERY OPTIMIZER

โดย

พีรยา มีนาภินันท์

พุทธิพงศ์ ธนูเทพ

อาจารย์ที่ปรึกษา

รศ.ดร.ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ต้นแบบตัวประมวลคำถามเชิงความหมาย

A PROTOTYPE SEMANTIC QUERY OPTIMIZER

ผู้จัดทำ

1. นางสาวพีรยา มีนาภินันท์ รหัสนักศึกษา 44010342

2. นายพุทธิพงศ์ ธนูเทพ รหัสนักศึกษา 44010344

อาจารย์ที่ปรึกษา

(รศ.ดร. ศุภมิตร จิตตะยโศธร)

ต้นแบบตัวประมวลคำถามเชิงความหมาย

นางสาวพีรยา มีนาภินันท์ 44010342
นายพุทธิพงศ์ ธนูเทพ 44010344
รศ.ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

Semantic Query Optimization เป็นกระบวนการที่ใช้ความรู้เชิงความหมาย(semantic knowledge) ของฐานข้อมูลเพื่อใช้ในการแปลงคำสั่งคิวรี่(query)เดิมมาสู่คำสั่งรูปแบบต่างๆ ที่มีประสิทธิภาพมากขึ้น โดยที่ผลลัพธ์ที่ได้ออกมาจากคำสั่งทั้งสองแบบนั้น จะมีค่าเท่ากัน ในปริญญานิพนธ์นี้เป็นการนำเอากฎข้อบังคับ (Integrity Rules) ที่ถูกสร้างขึ้นโดยผู้ดูแลระบบฐานข้อมูล มาช่วยในการหาคำตอบและทำการแปลงรูปแบบสำหรับคำสั่งคิวรี่ของSQLซึ่งเป็นอีกแนวทางหนึ่งในการทดลองสร้างตัวต้นแบบตัวประมวลคำถามเชิงความหมาย โดยได้นำเอาหลักการของเทคนิคต่างๆของ Semantic Query Optimization มาประยุกต์ใช้ นำกฎข้อบังคับมาใช้เป็นความรู้เชิงความหมาย ในการตรวจสอบและแปลงรูปแบบคำสั่ง และได้ทำการทดลองตามทฤษฎีที่ได้ศึกษา และเปรียบเทียบเวลาของคำสั่งคิวรี่เดิมและคิวรี่ที่ถูกแปลงรูปแบบแล้ว

A Prototype Semantic Query Optimizer

Peeraya Meenaphinan

Puttipong Tanutep

Assoc. Prof. Dr. Suphamit chittayasothon Advisor

ABSTRACT

The semantic query optimization is a process that takes advantage of the semantic knowledge for reformulation original query into alternate query that produces the same result and has more efficient. In this paper, we use integrity rules that are manipulated by database administrator to transform user query into equivalent form. This proposes another way for implement the prototype semantic query optimizer that utilizes these techniques to achieve. The implementation and test result by compare the cost are presented.

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ระบบในปัจจุบัน ณ เวลาที่มีผู้ใช้งาน (user) ใช้งานในฐานข้อมูล (database) เป็นจำนวนมากจะทำให้ เกิดการใช้งานที่คับคั่ง บางคำสั่งคิวรี่(Query) ในภาษา SQL ที่ได้รับมาจากผู้ใช้งานนั้น อาจจะขัดแย้งกับเงื่อนไขข้อบังคับ (constraint) ที่กำหนดไว้ในฐานข้อมูล หรือ บางคำสั่งคิวรี่นั้นจะไม่มีผลลัพธ์ที่แสดงออกมา เนื่องจากข้อมูลที่ผู้ใช้ต้องการนั้นไม่มีอยู่ในฐานข้อมูล แต่คำสั่งคิวรี่นี้ก็ยังถูกส่งไปยังระบบจัดการฐานข้อมูล(Database Management System : DBMS) ทำให้ระบบจัดการฐานข้อมูลทำงานโดยเปล่าประโยชน์ เสียเวลาและใช้ทรัพยากรของระบบอย่างไม่คุ้มค่า ซึ่งคำสั่งประเภทนี้ถ้าเราสามารถทำการตรวจสอบเงื่อนไขก่อนที่ส่งไปยังระบบจัดการฐานข้อมูลได้ ทำให้ไม่ต้องเสียเวลาส่งคำสั่งนั้นไปประมวลผล รวมทั้งสามารถส่งกลับไปบอกผู้ใช้งานได้ทันทีว่าไม่พบคำตอบที่ต้องการ ซึ่งจะทำให้ประสิทธิภาพของระบบนั้นดีขึ้น ช่วยลดภาระการทำงานของระบบจัดการฐานข้อมูลได้ในระดับหนึ่ง

1.2 วัตถุประสงค์ของการทำปริญญานิพนธ์

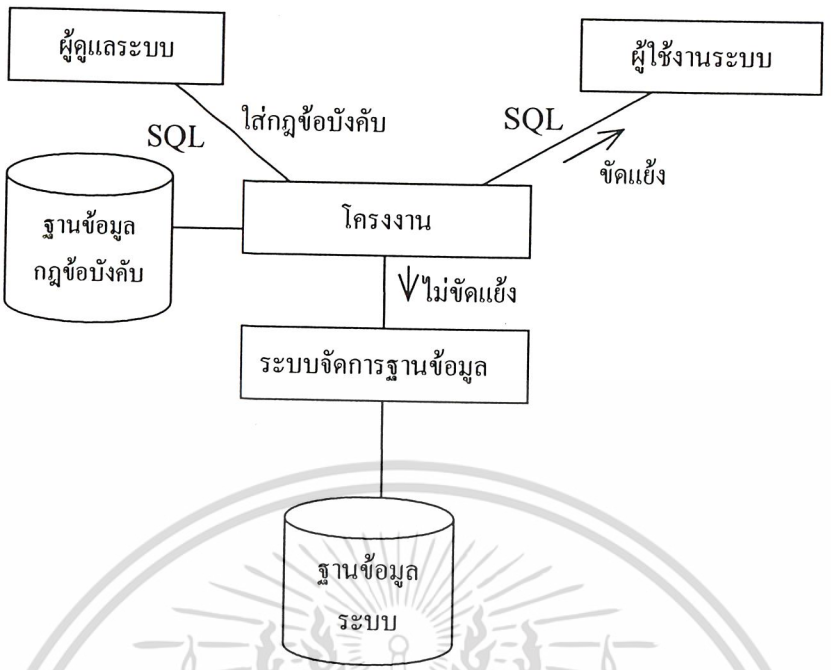
1.2.1 เพื่อลดภาระการทำงานของระบบจัดการฐานข้อมูล โดยทำการตรวจสอบคำสั่งคิวรี่ของผู้ใช้งานที่ระดับ Front-End

1.2.2 เพื่อช่วยลดการจราจร(Traffic) ที่คับคั่งของข้อมูล

1.2.3 เพื่อเพิ่มประสิทธิภาพการทำงานของแอพพลิเคชั่น (Application) โดยทำการลดเวลาการประมวลผลของคำสั่งคิวรี่

1.3 ขอบเขตของการทำปริญญานิพนธ์

ปริญญานิพนธ์นี้เป็นการทำการศึกษาเกี่ยวกับทฤษฎีของ Semantic Query Optimizer และเทคนิคต่างๆ ที่เกี่ยวข้อง ศึกษาและแยกประเภทของเงื่อนไขข้อบังคับ ซึ่งเป็นกฎพื้นฐาน (Simple Rule) ที่เกี่ยวข้องกับเทคนิคนั้น และทำการทดลองเขียนโปรแกรมเพื่อตรวจสอบคำสั่งคิวรี่ที่เข้ามาว่าขัดแย้งกับเงื่อนไขข้อบังคับตามที่กำหนดไว้หรือไม่ และทำการประยุกต์จากทฤษฎีที่ได้ศึกษา มาใช้ในการสร้างแอพพลิเคชั่นเพื่อนำมาแปลงรูปแบบคำสั่งในการคิวรี่ ซึ่งทำให้สามารถเพิ่มประสิทธิภาพในการทำงานโดยทำการเปรียบเทียบเวลาของคำสั่งคิวรี่ที่ได้ทำการแปลงคำสั่งแล้วเทียบกับคำสั่งคิวรี่เดิมที่ผู้ใช้ใส่เข้ามา



รูปที่ I-1 ภาพรวมของโครงงาน

ผู้ดูแลระบบฐานข้อมูล (Database Administrator : DBA) เป็นคนใส่ข้อมูลของกฎข้อบังคับแบบต่างๆ(Integrity Constraint) โดยกฎเหล่านี้สอดคล้องกับข้อมูลที่มีอยู่ในฐานข้อมูล เมื่อผู้ใช้งานมีการส่งคำสั่งคิวรี่เข้ามา แอพพลิเคชั่นนี้จะทำการตรวจสอบคำสั่งคิวรี่ที่เข้ามาก่อนที่จะส่งไปยังฐานข้อมูล โดยจะนำมาเปรียบเทียบกับกฎข้อบังคับที่มีอยู่ในฐานข้อมูลที่ผู้ดูแลระบบฐานข้อมูลใส่เข้ามาโดยถ้าคำสั่งคิวรี่ที่เข้ามาเมื่อตรวจสอบแล้วพบว่าขัดแย้งกับกฎที่มีอยู่ ก็จะทำการส่งกลับไปบอกผู้ใช้งาน เพื่อแจ้งให้รู้ว่าไม่พบคำตอบที่ผู้ใช้งานต้องการอยู่ โดยที่ไม่ต้องส่งไปยังระบบจัดการฐานข้อมูล แต่ในกรณีที่คำสั่งคิวรี่นั้นไม่ขัดแย้งกับกฎ ก็จะทำการแปลงคำสั่งคิวรี่ โดยยึดหลักการของ semantic query optimizer ทั้งนี้การแปลงจะต้องทำให้เพิ่มประสิทธิภาพในการทำงาน การแปลงนี้จะอาศัยกฎข้อบังคับในการใช้เป็นข้อมูลเปรียบเสมือนเป็นความรู้เชิงความหมาย (semantic knowledge)

1.4 วิธีการดำเนินงาน

ปริญญานิพนธ์นี้เริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่างๆ โดยในบทนี้เป็นการกล่าวนำ เพื่อให้เข้าใจถึงภาพรวมของการทำปริญญานิพนธ์ สาเหตุที่มาและประโยชน์ของการสร้างปริญญานิพนธ์ชิ้นนี้ขึ้น ในส่วนของเนื้อหาที่เป็นทฤษฎีเกี่ยวกับ query optimization และ ทฤษฎีของ Semantic Query Optimization รวมทั้งเทคนิคที่ทำการแปลงรูปแบบคำสั่งคิวรี่ มีรายละเอียดอยู่ในบทที่ 2

ในส่วนของการทำแบบจำลองมีการออกแบบโดยนำความรู้มาจากทฤษฎีที่ได้ศึกษามา ขั้นตอนของการทำการออกแบบอยู่ในบทที่ 3 มีการแบ่งประเภทของข้อกำหนดที่ใช้งาน ออกแบบหน้าจอที่ติดต่อ

กับผู้ใช้และผู้ดูแลระบบ ออกแบบฐานข้อมูลในการเก็บข้อกำหนด และออกแบบขั้นตอนการทำงานของแอพพลิเคชั่น โดยจะมีการลงรายละเอียดในบทที่ 4 ซึ่งเป็นการพัฒนาโปรแกรม มีการอธิบายโปรแกรมออกเป็นส่วนย่อย ว่ามีการทำงานอย่างไร

ในบทที่ 5 จะเป็นการทำการทดลองตามทฤษฎีที่ได้ศึกษามา ทำการทดสอบแอพพลิเคชั่น เปรียบเทียบเวลาของคำสั่งที่เข้ามา และคำสั่งที่ถูกทำการแปลงรูปแบบคำสั่งแล้ว แล้วทำการสรุปผลการทดลองจากผลลัพธ์ที่ได้ แนวทางในการพัฒนาเพิ่มเติม

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 Query Optimization

Query Optimization เป็นการเลือกวิธีที่จะทำการดำเนินการกับคำสั่งให้มีประสิทธิภาพ โดยที่เป้าหมายของการทำ Query Optimization คือการเลือกวิธีการที่มีการใช้ทรัพยากรน้อยที่สุด โดยหลักการทั่วไปแล้วจะพยายามที่จะลดเวลาที่ใช้ในการดำเนินการกับคำสั่งคิวรี่นั้นๆ ซึ่ง Query Optimization นั้นแบ่งออกเป็นประเภทได้ดังนี้

2.1.1 ตัวประมวลผลแบบอิงกฎ (Rule-Based Optimizer)

ตัวประมวลผลแบบอิงกฎจะมีการจัดลำดับของกฎตายตัวเรียงตามประสิทธิภาพ เมื่อมีคำสั่งคิวรี่เข้ามาตัวประมวลผลก็จะทำการกำหนดค่าให้แต่ละเส้นทางของการประมวลผลโดยใช้การจัดลำดับของกฎและจะเลือกเส้นทางที่มีค่าน้อยที่สุด กฎตายตัวนี้ขึ้นอยู่กับลักษณะการเขียนคำสั่งของ SQL

ยกตัวอย่างเช่น พิจารณาคำสั่ง บนตาราง PropertyForRent และกำหนดให้มีอินเด็กซ์บนคีย์หลัก (Primary Key) คือคอลัมน์ propertyNo, อินเด็กซ์บนคอลัมน์ rooms และอินเด็กซ์บนคอลัมน์ city

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

ในกรณีนี้ ตัวประมวลผลแบบอิงกฎจะพิจารณาเส้นทางการประมวลผลดังนี้

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

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

การสแกนแบบทั้งตาราง ซึ่งใช้ได้กับทุกคำสั่งของ SQL มีค่าเส้นทางลำดับที่ 15

ถึงแม้ว่าจะมีอินเด็กซ์บนคอลัมน์ propertyNo แต่ว่าคอลัมน์นี้ไม่ได้ปรากฏอยู่ในเงื่อนไข WHERE ดังนั้นจึงไม่ถูกพิจารณาโดยตัวประมวลผลแบบอิงกฎ ด้วยเหตุนี้ ตัวประมวลผลแบบอิงกฎจึงเลือกใช้อินเด็กซ์บนคอลัมน์ 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 Semantic Query Optimization

Semantic Query Optimization เป็นการใช้ความรู้ในเชิงความหมาย เพื่อทำการแปลงคำสั่งคิวรี่ ให้อยู่ในรูปแบบต่างๆ ซึ่งมีประสิทธิภาพในการประมวลผล (execute) ที่ต่างกัน แต่ผลลัพธ์ที่ได้ยังคงมีผลลัพธ์เหมือนกับ คำสั่งคิวรี่ที่ไม่ได้ทำการแปลง ความรู้ในเชิงความหมายนั้นอาจจะได้รับมาจากผู้ใช้งานของระบบโดยเป็นผู้ดูแลระบบ หรือได้รับมาจากระบบโดยมีการประมวลผลมาจากฐานข้อมูล

ในกระบวนการทำการแปลงรูปคำสั่งคิวรี่ (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 กระบวนการทำการแปลงรูปคำสั่งคิวรี่ มีอยู่ 3 รูปแบบหลักๆ คือ

2.2.1.1 การเพิ่มกฎ (Constraint Addition)

เป็นการนำแอททริบิว Attribute ที่เป็นอินเด็กซ์เข้ามาเพิ่มในการคิวรี่ ยกตัวอย่าง เช่น

คำสั่งเดิม:

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

กฎ : entry = 94 ⟶ regno >= 940000

ซึ่ง regno นั้นเป็นแอททริบิวที่เป็นอินเด็กซ์

คำสั่งใหม่ :

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

2.2.1.2 ลบเงื่อนไข (Constraint Deletion)

เป็นการนำเอาเงื่อนไขที่ไม่จำเป็นออกจากคิวรี่ 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)

เป็นการ ไม่ยอมรับคำสั่งนี้ โดยไม่ต้องส่งคำสั่งคิวรี่เข้าไปให้ระบบจัดการดูแลฐานข้อมูลประมวลผล เมื่อคำสั่งคิวรี่นั้นขัดแย้งกับกฎ ยกตัวอย่าง เช่น

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

กฎ : status = 'A' ⟶ advisor = 100

คำสั่งใหม่ : **SELECT** Null

2.2.2 อัลกอริทึมของกระบวนการแปลงรูปแบบคำสั่งคิวรี่ (Query Reformulation Algorithm)

ทำขั้นตอนซ้ำ : 1 ถึง 4 ทุกๆ เงื่อนไข Ci ของคำสั่งคิวรี่รวมทั้ง เงื่อนไขใหม่ที่เพิ่มขึ้น จนกระทั่งทุกๆ 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 สอดคล้องกับเงื่อนไขอื่นๆในคำสั่งคิวรี่หรือ ไม่ ถ้าไม่สอดคล้อง ให้แจ้งขัดแย้งเลย ถ้าสอดคล้อง ก็ทำขั้นตอนที่ 3 ต่อไป

2.2.2.3 ทำการเพิ่มเงื่อนไขเข้าไปในเงื่อนไขของคำสั่ง (Query Condition) เพื่อทำเป็นคำสั่งคิวรี่เงื่อนไข Cn ใหม่

2.2.2.4 ทำการสร้างความสัมพันธ์ที่ขึ้นต่อกัน (Cn,Ci) ที่ซึ่งเงื่อนไขใหม่ (Cn) นั้น ขึ้นอยู่กับเงื่อนไขปัจจุบัน 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 จะได้คำสั่งคิวรี่ดังนี้

```
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 นั้นไม่ใช่อินเด็กซ์ และมีความยาวที่ยาวที่สุด เมื่อเปรียบเทียบกับ 3 แอททริบิวโดยที่ กฎ 1 นั้น ทำให้เกิดความสัมพันธ์ที่ขึ้นต่อกันระหว่าง(dname,dcode) จะเก็บ(hold)ไว้ และเงื่อนไขต่อมาก็ทำซ้ำกันและค่าเงื่อนไขที่มีค่าใช้จ่ายสูงสุดต่อมา คือ dcode = 'ACCT' ซึ่งยังคงถูกเก็บ เพราะมันไม่สอดคล้องกับความสัมพันธ์ที่ขึ้นต่อกันอื่นๆและค่าใช้จ่ายสูงรองลงมา คือ manager = 'A01' จะถูกเก็บไว้เหมือนกัน ดังนั้นคำสั่งคิวรี่จะได้เป็น

```
SELECT *
FROM department
WHERE manager = 'A01' and dcode = 'ACCT"
```

วิธีการแปลงรูปแบบนี้อาจจะง่ายต่อการขยายรูปแบบของคำสั่งคิวรี่ ซึ่งเกี่ยวข้องกับการเชื่อมหลายตาราง ในกรณีนี้คำสั่งคิวรี่จะถูกแบ่งออกเป็น คำสั่งย่อยๆ ซึ่งแต่ละคำสั่งย่อยๆ นี้จะกระทำกับคำสั่งคิวรี่ เพียงความสัมพันธ์ของตารางเดียว แต่ละคำสั่งย่อยจะถูกแปลงโดยเป็นอิสระต่อกัน และคำสั่งย่อยๆ พวกนี้ที่ถูกแปลงแล้วจะถูกรวมกลับมาเป็นคำสั่งคิวรี่เดิม ถึงแม้ว่าแต่ละข้อมูลที่ได้รับกลับมาจากแต่ละตารางจะมีความเกี่ยวข้องกันเนื่องมาจากการเชื่อมโยงกันที่ไม่สามารถเปลี่ยนได้ แต่เวลาที่ใช้นั้นก็ลดลงอย่างเห็นได้ชัด นอกจากนี้ในการที่คำสั่งย่อยนั้นถูกปฏิเสธ ขั้นตอนของการแปลงคำสั่งคิวรี่ นั้นก็จะส่งค่า

ว่างเปล่า(NULL) กลับมาโดยอัตโนมัติ เนื่องจากคำสั่งที่ถูกปฏิเสธนั้นไม่มีข้อมูลที่ตรงกับเงื่อนไขที่กล่าวมา

2.3 เทคนิคของ Semantic Query Optimization

ใน DB2 Universal Database พูดถึงเรื่องการสร้างกระบวนการคิดและประสิทธิภาพของผลลัพธ์ โดยใช้ TPCD และ APB-1 OLAP benchmarks ในการทดลองแสดงให้เห็นว่า Semantic Query Optimization สามารถพัฒนาประสิทธิภาพในการเรียกดูข้อมูล

ระบบฐานข้อมูลแบบรีเลชั่นแนล (Relational Database) กลายเป็นเทคโนโลยีที่มีความพิเศษสำหรับทางด้าน การจัดเก็บ(storing), การจัดการ(handling) และ การเรียกดูข้อมูลต่างๆ หลังจากที่พัฒนาในประสิทธิภาพของการเรียกดูข้อมูล ในระบบเช่นนี้ องค์ประกอบที่เป็นกุญแจหลักในการพัฒนาคือ การนำเข้ามาใช้และการพัฒนาเทคนิคที่ใช้แบบต่างๆ (query optimization techniques) ต่อมามีการพัฒนาเป็น semantic query optimization (SQO) โดยใช้กฎข้อบังคับที่เกี่ยวข้องกับฐานข้อมูล เพื่อพัฒนาประสิทธิภาพของการประเมินค่าเทคนิคในการเรียกดูข้อมูลมีการกล่าวถึงดังนี้

1.) **Join Elimination**

- แบบใช้ข้อบังคับแบบฟอเรนท์คีย์ (using foreign key constraints)

เมื่อมีการกำหนดข้อบังคับแบบฟอเรนท์คีย์ ระหว่างการเชื่อมกันของสองตาราง ระบบจะรับประกันว่าสำหรับแต่ละแถว (row) ในตารางลูก(child table)ที่มีคอลัมน์ที่เป็นฟอเรนท์คีย์ ทั้งหมดที่ไม่มีค่าเป็นนัล(NULL) จะมีแถวในตารางหลัก (parent table) ที่มีค่าตรงกันในคอลัมน์ที่เป็นคีย์หลัก ดังนั้นถ้าจุดประสงค์ของการเชื่อมโยงตารางจึงเป็นเพียงเพื่อเช็คว่า ค่าในฟอเรนท์คีย์คอลัมน์ จะมีปรากฏอยู่ในคอลัมน์ที่เป็นคีย์หลัก ด้วยนั้น เราสามารถตัด การเชื่อมโยงนี้ทิ้งไปได้ (ตัวอย่างคือ สำหรับบางคำสั่งที่เกี่ยวข้องกับการเชื่อมโยงระหว่าง สองตารางสัมพันธ์กันผ่านทาง ข้อบังคับแบบฟอเรนท์คีย์

- แบบตรวจสอบแบบที่เป็นการเชื่อมตารางแบบว่างเปล่า(detection of empty join)

ในบางครั้งผลลัพธ์ของการเชื่อมตารางนั้นไม่มีค่าในฐานข้อมูล ถ้าเราพบผลลัพธ์แบบนี้ เราจะสามารถทำการปรับปรุงคำสั่งดั้งเดิมได้โดยการกำจัดการเชื่อมโยงตารางแบบว่างเปล่า

2.) **Join Introduction**

เป็นวิธีการที่ใช้ข้อได้เปรียบของการเชื่อมโยงตารางที่เพิ่มขึ้นมา ถ้าตารางนั้นมีความสัมพันธ์กับตารางเดิม(วิธีนี้จะน่าสนใจมากขึ้นถ้าแอททริบิวที่นำมาเชื่อมโยงนั้นเป็นอินเด็กซ์)

3.) **Predicate Elimination**

จากกฎข้อบังคับ เราสามารถสรุปได้ว่า บางเพรดดิเคตนั้นเป็นจริงเสมอสำหรับการเรียก ดังนั้นเราสามารถตัดเพรดดิเคตที่ซ้ำซ้อนทิ้งได้ เพื่อเพิ่มประสิทธิภาพให้มากขึ้น ถ้าในเพรดดิเคตนั้นไม่มีอินเด็กซ์อยู่

4.) Predicate Introduction

- แบบใช้อินเด็กซ์ (using index introduction) ถ้าเพรดดิเคตใหม่ที่ได้จากการสรุป จากกฎข้อบังคับและจากบางเพรดดิเคตที่มีอยู่แล้วในคำสั่งคิวรี่ซึ่งมีอินเด็กซ์อยู่ ในเพรดดิเคตนั้น จะทำการนำเพรดดิเคตนี้มาเพิ่มเข้ามาในคำสั่ง query เพื่อทำให้เพิ่มความเร็วของการประเมินผล

- แบบใช้ลดช่วงการค้นหา (scan reduction) คล้ายกับแบบใช้อินเด็กซ์ ในบางครั้งเรานำบางเพรดดิเคตใหม่ลงในคำสั่งคิวรี่ ซึ่งเพรดดิเคตนี้ช่วยทำการลดช่วงในการค้นหาสำหรับบางข้อมูล ดังนั้นจึงเป็นการเพิ่มประสิทธิภาพ

5.) Detecting the Empty Answer Set

ถ้าเพรดดิเคตนั้นมีความขัดแย้งกับกฎข้อบังคับ ผลลัพธ์ของคำสั่งคิวรี่จะไม่มีคำตอบ ดังนั้นถ้าเราทำการตรวจสอบเงื่อนไขนี้ก่อน เราจะสามารถหลีกเลี่ยงการประเมินค่า คำสั่งคิวรี่ที่ไม่มีคำตอบได้

2.4 ส่วนของการสร้าง (Implementation)

2.4.1 การเพิ่มเพรดดิเคต (Predicate Introduction)

แนวคิดของ Predicate Introduction(PI) แยกเป็น 2 วิธี คือ index introduction และ scan reduction แนวคิดของ index introduction เป็นการเพิ่มเพรดดิเคตใหม่ลงในคำสั่งquery โดยในเพรดดิเคตนั้นอินเด็กซ์อยู่บนแอททริบิวนั้น การได้ tuple โดยใช้ index จะมีประสิทธิภาพมากกว่า แต่สมมติฐานนี้ไม่เป็นจริงเสมอไป เนื่องจากว่ากฎโดยทั่วไปนั้นจะได้มาจากการที่ใช้ตัวช่วยการค้นหา(heuristics)หรือคำสั่งต่างๆที่ถูกสร้างขึ้นมา ซึ่งหลังจากนี้จะถูกนำไปประเมินค่าใช้จ่ายจาก plan optimizer ซึ่งเราไม่สามารถได้รับการตอบรับจาก plan generator ได้ดังนั้นเราจึงจำเป็นที่จะต้องกำหนดข้อจำกัดในการใช้ซึ่งจะการันตีได้ว่ามันมีประสิทธิภาพมากขึ้นจริง อย่างที่สองเป็นแนวคิดของการลดช่วงการค้นหา (Scan reduction) แนวคิดนี้คือ เพิ่มเพรดดิเคต เพื่อไปลดจำนวนข้อมูลที่ได้รับจากฐานข้อมูล ตัวอย่างข้างล่างนี้เป็นตัวอย่างการทำ scan reduction

<u>ตัวอย่าง 2</u> S1 เป็นคำสั่งคิวรี่ TPCD database , กฎข้อบังคับ คือ l_receiptdate >= l_shipdate และ predicate ใหม่ คือ l2.l_receiptdate > date ('1998-07-01') และ l1.l_commitdate > date ('1998-07-01')

```
S1:   select  sum(l1.l_extendedprice * l1.discount) as revenue
      from    tpcd.lineitem as l1, tpcd.lineitem as l2
      where   l1.l_commitdate = l2.l_receiptdate  and
              l2.l_shipdate > date('1998-07-01')  and
```

l2.l_discount between 0.06 – 0.01 and

0.06 + 0.01 and l2.l_quantity < 24;

ในการสร้างแบบของการเพิ่มเพรดดิเคตนั้น ทำการรวมเอา Semantic Query Optimization เทคนิคอื่นๆ เข้ามาด้วยคือ การตรวจหาเซตของคำตอบที่ได้จากการเรียกดูข้อมูล คิวรี่ที่ได้คำตอบเป็นเซตว่าง ดังตัวอย่าง

S2: **select** sum (l_extendedprice * l_discount) as revenue

from tpcd.lineitem

where l_shipdate > date ('1994-01-01') and

l_receiptdate < date('1994-01-01')

ในคำสั่งคิวรี่ของมันเองไม่มีการขัดแย้ง แต่อย่างไรก็ตาม จากกฎที่ว่า l_shipdate <= l_receiptdate ดังนั้นคำสั่งคิวรี่นี้ไม่สามารถให้คำตอบใดๆได้ ดังนั้น กรณีนี้ไม่จำเป็นต้องประเมินค่า(evaluated)

กระบวนการเพิ่มเพรดดิเคต มี 2 ส่วนประกอบหลักคือ มีตัวพิสูจน์ทฤษฎี (theorem prover) และฟิวเตอร์ (filter) ซึ่งอินพุท(input) ที่ส่งให้ theorem prover เป็น เซตของ constraint ทั้งหมดที่ กำหนดสำหรับฐานข้อมูลและเซตของ เพรดดิเคตในคำสั่งquery โดยที่ผลลัพธ์ของมันเป็นเซตของกฎที่ไม่มีการซ้ำซ้อน ซึ่งได้มาจากอินพุทเซตของพวกกฎที่ได้เหล่านี้เป็นเพรดดิเคตใหม่ซึ่งสามารถเพิ่มเข้าไปในคำสั่งซึ่งอยู่ในรูปแบบของ A op B ซึ่ง A และ B เป็นชื่อของแอททริบิว หรือค่าคงที่ โดยที่ op เป็น $>, <, \leq, \geq$ หรืออีกนัยหนึ่งก็คือมีเพียงเพรดดิเคตใหม่บางส่วนที่มีประโยชน์สำหรับการ optimization บทบาทของฟิวเตอร์ คือต้องหาพวกกฎเหล่านี้ ในจุดมุ่งหมายในการออกแบบกฎที่ช่วยในการค้นหา สำหรับฟิวเตอร์คือ ต้องรับประกันได้ว่าเพรดดิเคตใหม่ที่เพิ่มเข้ามาในคำสั่งต้องทำให้ตัวประมวลผลหาเส้นทางการเข้าถึงได้ดีกว่าที่มีอยู่ในคำสั่งคิวรี่เดิม โดยมีกระบวนการดังนี้

ให้ N เป็นเซตของเพรดดิเคตใหม่ที่คำนวณมาจากตัวพิสูจน์ทฤษฎี

1. ถ้า N เป็นเป็นเพรดดิเคตที่ขัดแย้ง แฟลกซ์จะถูกเซตค่าแล้วคำสั่งจะไม่ถูกประเมินค่า
2. กรณีถ้าไม่ขัดแย้ง แต่ละเพรดดิเคต A op B $\in$ N จะถูกเพิ่มเข้าไปในคำสั่ง ถ้า

ก. A หรือ B เป็น join column

ข. - A เป็นค่าคงที่

- B เป็นคอลัมน์หลักของอินเด็กซ์ และ

- ไม่มีอินเด็กซ์ใดๆ บนตารางของ B ที่สามารถใช้ในเส้นทางการเข้าถึง สำหรับคำสั่งคิวรี่เดิม

ในกรณีที่ 1 จะใช้จัดการเกี่ยวกับการขัดแย้งกับกฎในคำสั่งคิวรี่

ในกรณีที่ 2.ก. จะจัดการเกี่ยวกับการทำการลดช่วงซึ่งเป็นส่วนที่สำคัญของกระบวนการ

2.ข. จัดการเกี่ยวกับการนำอินเด็กซ์มาใช้ในคำสั่ง

บทที่ 3
การออกแบบระบบ

3.1 สถาปัตยกรรม

โครงสร้างของระบบแบ่งออกเป็นส่วนของ

3.1.1 ส่วนที่ติดต่อผู้ใช้งาน

เป็นหน้าจอที่ติดต่อกับผู้ใช้งานเพื่อรับข้อมูลจากผู้ใช้และการแสดงผล

3.1.2 ส่วนของกฎข้อบังคับ

เป็นรูปแบบของกฎที่นำไปใช้ในการแปลงรูปแบบ และไว้ตรวจสอบคำสั่งที่เข้ามา

3.1.3 ส่วนของการตรวจสอบ

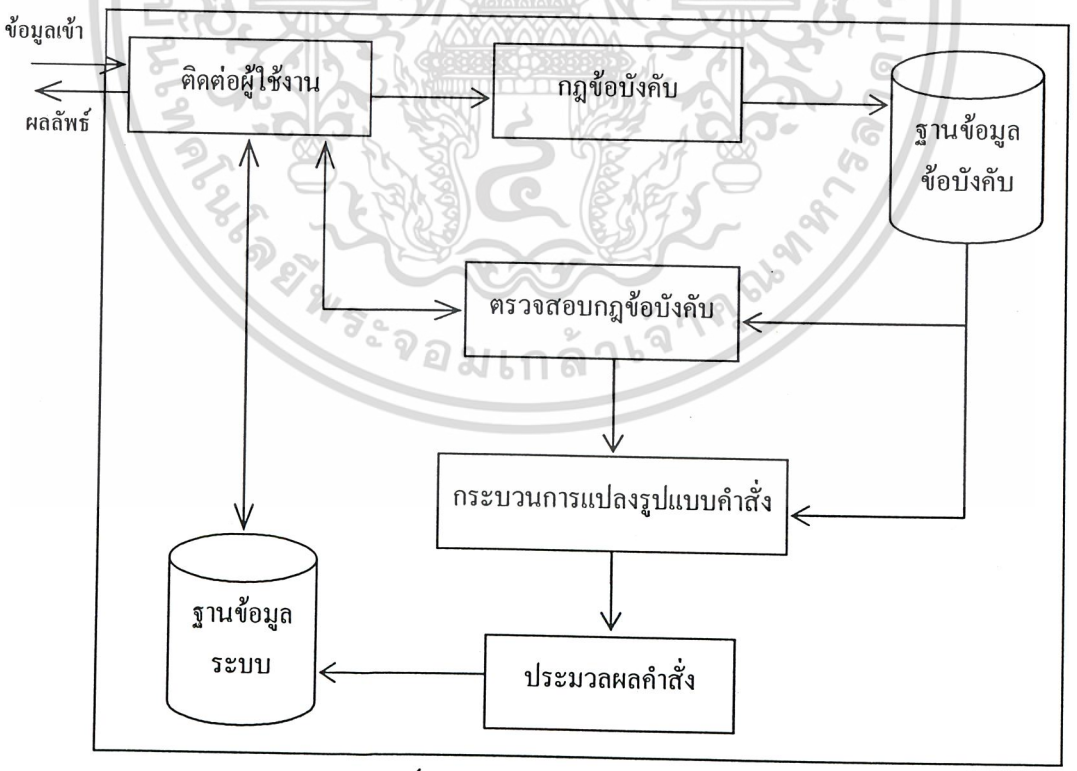
เป็นส่วนของการนำกฎข้อบังคับที่มีอยู่นำมาตรวจสอบว่าขัดแย้งกับกฎข้อบังคับหรือไม่

3.1.4 ส่วนของการแปลงรูปแบบคำสั่ง

เป็นส่วนของการนำกฎที่มีอยู่มาทำการแปลงรูปแบบตามทฤษฎีของ semantic query optimizer

3.1.5 ส่วนของฐานข้อมูล

เป็นส่วนของฐานข้อมูลไว้เก็บกฎข้อบังคับและไว้ทำการติดต่อกับฐานข้อมูล



รูป 3-1 Block diagram

จากรูปบล็อกไดอะแกรม (Block Diagram) ข้อมูลที่รับเข้ามาจะมีสองแบบ คือ คำสั่งของภาษา SQL หรือข้อมูลของข้อกฎบังคับที่ผู้ดูแลระบบใส่เข้ามา ในกรณีที่เป็นข้อมูลของกฎข้อบังคับก็จะถูกกำหนดรูปแบบแล้วส่งไปเก็บยังฐานข้อมูล ในกรณีที่เป็นคำสั่งของภาษา SQL เริ่มแรกจะนำไปตรวจสอบกับกฎข้อบังคับ ถ้าตรวจสอบแล้วพบว่าขัดแย้งกับกฎจะแสดงผลให้กับผู้ใช้งานทราบ แต่ถ้าไม่ขัดแย้งกับกฎ จะเข้าสู่กระบวนการแปลงรูปแบบคำสั่ง โดยกระบวนการนี้ก็นำข้อมูลจากฐานข้อมูลของกฎข้อบังคับมาใช้ เมื่อคำสั่งที่ถูกแปลงเรียบร้อยแล้วจะนำคำสั่งดั้งเดิมที่ผู้ใช้งานใส่ก่อนแปลงรูปแบบและคำสั่งที่ถูกแปลงส่งไปประมวลผล และทำการจับเวลาเพื่อเปรียบเทียบกัน และนำผลลัพธ์มาแสดงออกทางหน้าจอ

3.2 การออกแบบ

3.2.1 การออกแบบส่วนที่ติดต่อกับผู้ใช้งาน

ส่วนของหน้าจอที่ติดต่อกับผู้ใช้งาน (User Interface) แบ่งออกเป็น 2 ส่วน

3.2.1.1 ส่วนของผู้ดูแลระบบ

ผู้ดูแลระบบนั้นมีการใส่รหัสผ่านก่อนการเข้าใช้งานเพื่อป้องกันไม่ให้ผู้ใช้งานทั่วไปสามารถใส่กฎข้อบังคับได้ ออกแบบให้ผู้ใช้งานระบบสะดวกต่อการใส่กฎโดยไม่ต้องพิมพ์คำสั่งรูปแบบเฉพาะ สามารถที่จะเลือกตารางและคอลัมน์ในฐานข้อมูลได้ เพื่อให้ผู้ดูแลระบบไม่จำเป็นที่จะต้องสะกดชื่อตารางและคอลัมน์ได้อย่างถูกต้อง และผู้ดูแลระบบสามารถใช้งานคำสั่งคิวรี่ได้เหมือนกับผู้ใช้งานทั่วไป

3.2.1.2 ส่วนของผู้ใช้งาน

สามารถที่จะทำการใส่คำสั่งคิวรี่ไปในระบบ โดยจะแสดงผลลัพธ์ของคำสั่งที่ถูกใส่เข้ามาและ แสดงเวลาของคำสั่งคิวรี่เดิมเปรียบเทียบกับคำสั่งที่ถูกแปลงรูปแบบแล้ว

หน้าจอแรกจะเป็นการเลือกโหมดการทำงานว่าต้องการจะใช้งานการประมวลผลคำสั่งจะอยู่ในโหมดประมวลผลคำสั่ง รูปที่ 3-2 และรูปที่ 3-3 อยู่ในโหมดกฎข้อบังคับซึ่งโหมดนี้ผู้ดูแลระบบสามารถใช้ได้เพียงผู้เดียว ผู้ใช้งานทั่วไปไม่สามารถใช้งานได้

3.2.2 การออกแบบส่วนของกฎข้อบังคับ

ทำการแยกประเภทของกฎออกเป็น 3 ประเภท คือ

3.2.2.1 เงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว (Attribute Constraint)

เป็นการกำหนดค่าขอบเขตว่าข้อมูลที่ใส่เข้ามาในคอลัมน์ของแอททริบิวนั้น มีค่าเท่าไหร่ได้บ้าง เช่น CID < 30000 หมายความว่า ค่าที่จะใส่เข้ามาในคอลัมน์ของแอททริบิว CID มีค่าน้อยกว่า 30,000

โดยมีรูปแบบที่กำหนดขอบเขตคือ รูปแบบ = , != , > , < , >= , <= และใส่แบบเป็นช่วง เช่น 20 < X < 50 , 60 <= X < 400

3.2.2.2 เงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว (Inter-Attrinute Constraint)

เป็นการกำหนดความสัมพันธ์ของแอททริบิวว่า แอททริบิวสองแอททริบิวนั้นมีความสัมพันธ์กันอย่างไรบ้าง เช่น SALARY > PAYMENT หมายความว่าค่าในคอลัมน์ของแอททริบิว SALARY ต้องมีค่ามากกว่า ค่าในคอลัมน์ของแอททริบิว PAYMENT ในแถวเดียวกัน

3.2.2.3 เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependentcy)

เป็นการกำหนดว่าถ้าแอททริบิวแรกมีค่าตามที่กำหนดไว้แล้ว อีกแอททริบิวหนึ่งก็จะถูกส่งผลให้มีค่าอยู่ในช่วงที่กำหนด เปรียบเสมือนกับกรณี ถ้า-แล้ว (If-Then) ยกตัวอย่างเช่น ADDRESS = 'BANGKOK' -> SALARY = 500000 หมายความว่า ในกรณีที่ ค่าในคอลัมน์ ADDRESS มีค่าเท่ากับ 'BANGKOK' แล้ว ค่าที่อยู่ในคอลัมน์ SALARY จะต้องเท่ากับ 500000 เสมอ แต่ไม่จำเป็นว่า ค่าที่อยู่ในคอลัมน์ SALARY ที่เท่ากับ 500000 จะต้องมีค่าที่อยู่ในคอลัมน์ ADDRESS เท่ากับ 'BANGKOK' อาจเป็นค่าอื่นก็ได้

3.1.3 การออกแบบส่วนของการตรวจสอบ

แบ่งการตรวจสอบออกเป็น 2 ที่

3.1.3.1 การตรวจสอบคำสั่งภาษา SQL ที่เข้ามาของผู้ใช้งานว่าข้อมูลที่ต้องการหานั้นขัดแย้งกับกฎหรือไม่ โดยการตรวจสอบขั้นแรกจะทำโดยตรวจสอบกับเงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว เมื่อไม่ขัดแย้งกับกฎนี้ก็ทำการตรวจสอบกับเงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว และเงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน ถ้าไม่ขัดแย้งก็ทำส่วนอื่นต่อไป แต่ถ้าขัดแย้ง ก็ทำการแจ้งไปบอกผู้ใช้งานให้ทราบ

3.1.3.2 การตรวจสอบการใส่เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependency) ว่าค่าที่ใส่เข้ามานั้นขัดแย้งกับเงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว (Attribute Constraint) หรือไม่

3.1.4 การออกแบบส่วนของการแปลงรูปแบบคำสั่ง

การออกแบบส่วนของการแปลงรูปแบบคำสั่งนั้น จะนำกฎแบบต่างๆ มาใช้ โดยจะนำมาใช้ในการทำเพิ่มเพรดดิเคตที่เป็นดัชนี (Index Introduction) การลดช่วงของการค้นหา (Scan Reduction) การตัดเพรดดิเคตที่ไม่มีค่าอยู่ในข้อมูล หรือตัดเพรดดิเคตที่ซ้ำซ้อน (Predicate Elimination) โดยจะนำค่าเงื่อนไขแต่ละเพรดดิเคตในคำสั่งคิวรี่ มาทำการตรวจสอบทีละเพรดดิเคต โดยนำมาเทียบกับกฎแล้วพิจารณาดูว่าสามารถทำการแปลงรูปด้วยรูปแบบใดได้ ตามทฤษฎี ซึ่งจะลงรายละเอียดในบทต่อไป

3.1.5 การออกแบบส่วนของฐานข้อมูล

การออกแบบฐานข้อมูลในส่วนของกฎข้อบังคับมีการเก็บกฎอยู่สามรูปแบบ คือ เก็บตามเงื่อนไข

ตารางแรก เป็นกรณีของเงื่อนไขแบบกำหนดค่าขอบเขตของแอททริบิว มีสี่คอลัมน์

คอลัมน์แรก เก็บชื่อของแอททริบิวที่ต้องการกำหนดขอบเขต

คอลัมน์สอง เก็บชื่อของตารางที่ต้องการกำหนดขอบเขต

คอลัมน์สาม เก็บเครื่องหมาย กำหนดประเภทของขอบเขต เก็บเป็นค่าจำนวนเต็ม (Integer)

จะเก็บตามกรณีของเครื่องหมาย ตามกรณี

เครื่องหมาย =	เก็บเป็น	0
เครื่องหมาย !=	เก็บเป็น	1
เครื่องหมาย >	เก็บเป็น	2
เครื่องหมาย <	เก็บเป็น	3
เครื่องหมาย >=	เก็บเป็น	4
เครื่องหมาย <=	เก็บเป็น	5
เครื่องหมาย < and <	เก็บเป็น	6
เครื่องหมาย < and <=	เก็บเป็น	7
เครื่องหมาย <= and <=	เก็บเป็น	8
เครื่องหมาย <= and <	เก็บเป็น	9
เครื่องหมาย in	เก็บเป็น	10

คอลัมน์สี่ เก็บค่าขอบเขต เป็นสตริงโดยที่ค่าขอบเขตมีการเก็บแยกเป็นกรณีดังนี้

เครื่องหมาย = , != , < , <= , > , >= จะเก็บเป็นแบบค่าตัวเลขที่รับเข้ามาเลย เช่น

CID < 500 จะเก็บเป็น 500

เครื่องหมาย < and < , < and <= , <= and <= , <= and < จะเก็บเป็น ค่าขอบเขตทางซ้าย/ค่าขอบเขตทางขวา เช่น

500 <= CID < 1000 จะเก็บเป็น 500/1000

กรณีเครื่องหมายเป็น in จะเก็บเป็นรูปแบบ (x,x,x,x) เช่น

CNAME in ('PETER', 'JOHNSON') จะเก็บในรูปแบบ ('PETER', 'JOHNSON')

DISCOUNT in (10 , 20 , 30 , 40) จะเก็บในรูปแบบ (10 , 20 , 30 , 40)

ตารางสอง เป็นกรณีของเงื่อนไขแบบความสัมพันธ์ระหว่างแอททริบิว มี 5 คอลัมน์

คอลัมน์แรก เก็บชื่อของแอททริบิวที่มีความสัมพันธ์กันตัวแรก

คอลัมน์สอง เก็บชื่อของแอททริบิวที่มีความสัมพันธ์กันตัวที่สอง

คอลัมน์สาม เก็บชื่อของตารางที่มีความสัมพันธ์กันตัวแรก

คอลัมน์สี่ เก็บชื่อของตารางที่มีความสัมพันธ์กันตัวที่สอง

คอลัมน์ห้า เก็บเครื่องหมาย โดยจะเก็บเหมือนตารางแรก

เช่น SALARY < PAYMENT จะเก็บเป็น

| SALARY | PAYMENT | CUSTOMER_TBL | CUSTOMER_TBL | 4

ตารางสาม เป็นกรณีของเงื่อนไขแบบความสัมพันธ์ที่ขึ้นต่อกันจะคล้ายกรณีของการเก็บของตารางแรก แต่จะเพิ่มในส่วนของเพรดดิเคต (Predicate) หลังที่ขึ้นต่อส่วนแรกมาเก็บด้วย มี 8 คอลัมน์

คอลัมน์แรก เก็บชื่อของแอททริบิวของเพรดดิเคตแรก

คอลัมน์สอง เก็บชื่อของตารางของเพรดดิเคตแรก

คอลัมน์สาม เก็บเครื่องหมายของเพรดดิเคตแรก

คอลัมน์สี่ เก็บชื่อของค่าของเพรดดิเคตแรก

คอลัมน์ห้า เก็บชื่อของแอททริบิวของเพรดดิเคตสอง

คอลัมน์หก เก็บชื่อของตารางของเพรดดิเคตสอง

คอลัมน์เจ็ด เก็บเครื่องหมายของเพรดดิเคตสอง

คอลัมน์แปด เก็บชื่อของค่าของเพรดดิเคตสอง

เช่น ADDRESS = 'BANGKOK' -> CID BETWEEN 10000 AND 40000

จะเก็บเป็น | ADDRESS | CUSTOMER_TBL | 0 | 'BANGKOK' | CID | 8 |10000/40000 |

ATT	TBL	SIGN	VAL
ADDRESS	CUSTOMER_TBL	1	'USA'
CID	CUSTOMER_TBL	5	50000

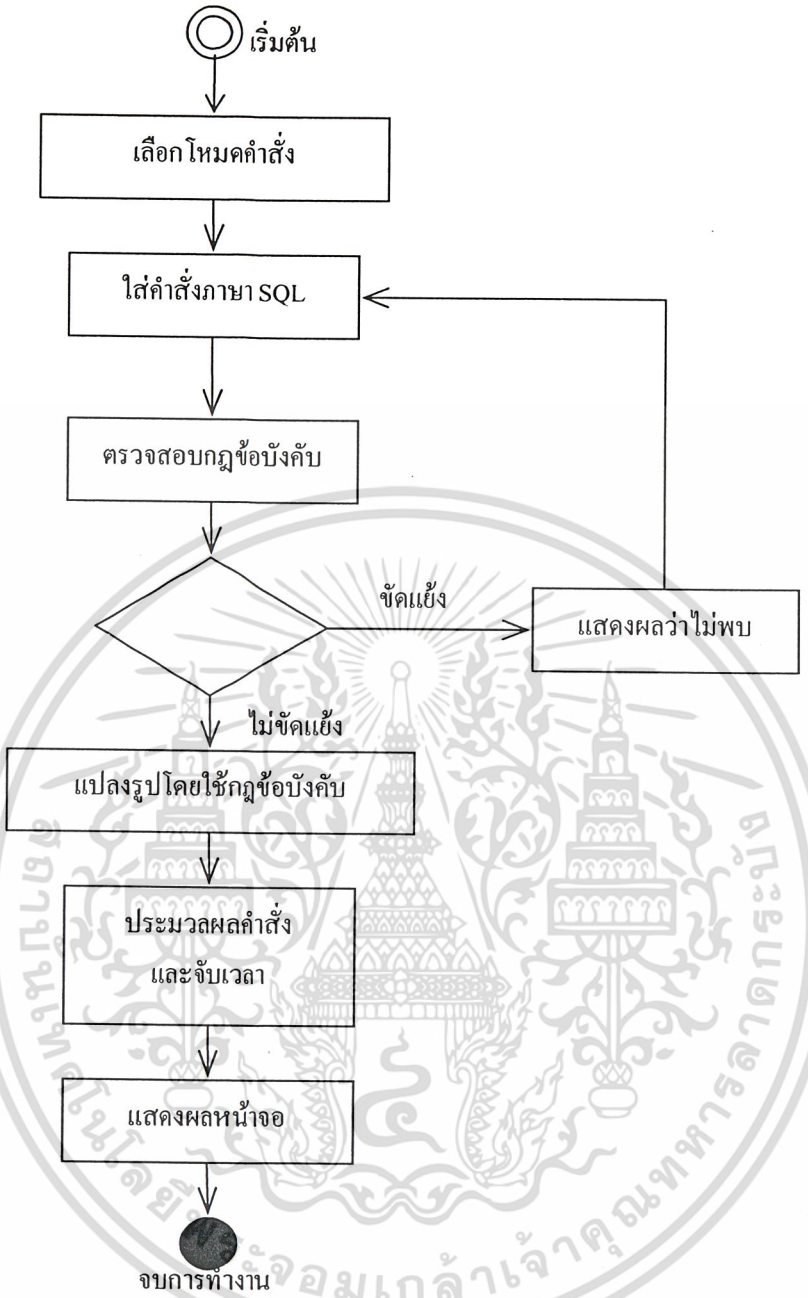
ตาราง 3-1 เงื่อนไขแบบกำหนดค่าขอบเขตของแอททริบิว

ATTL	ATTR	TATTL	TATTR	SIGN
QTY	ONHAND	ORDER_TBL	PRODUCT_TBL	3
CURR_BAL	CREDIT_LIM	CUSTOMER_TBL	CUSTOMER_TBL	3

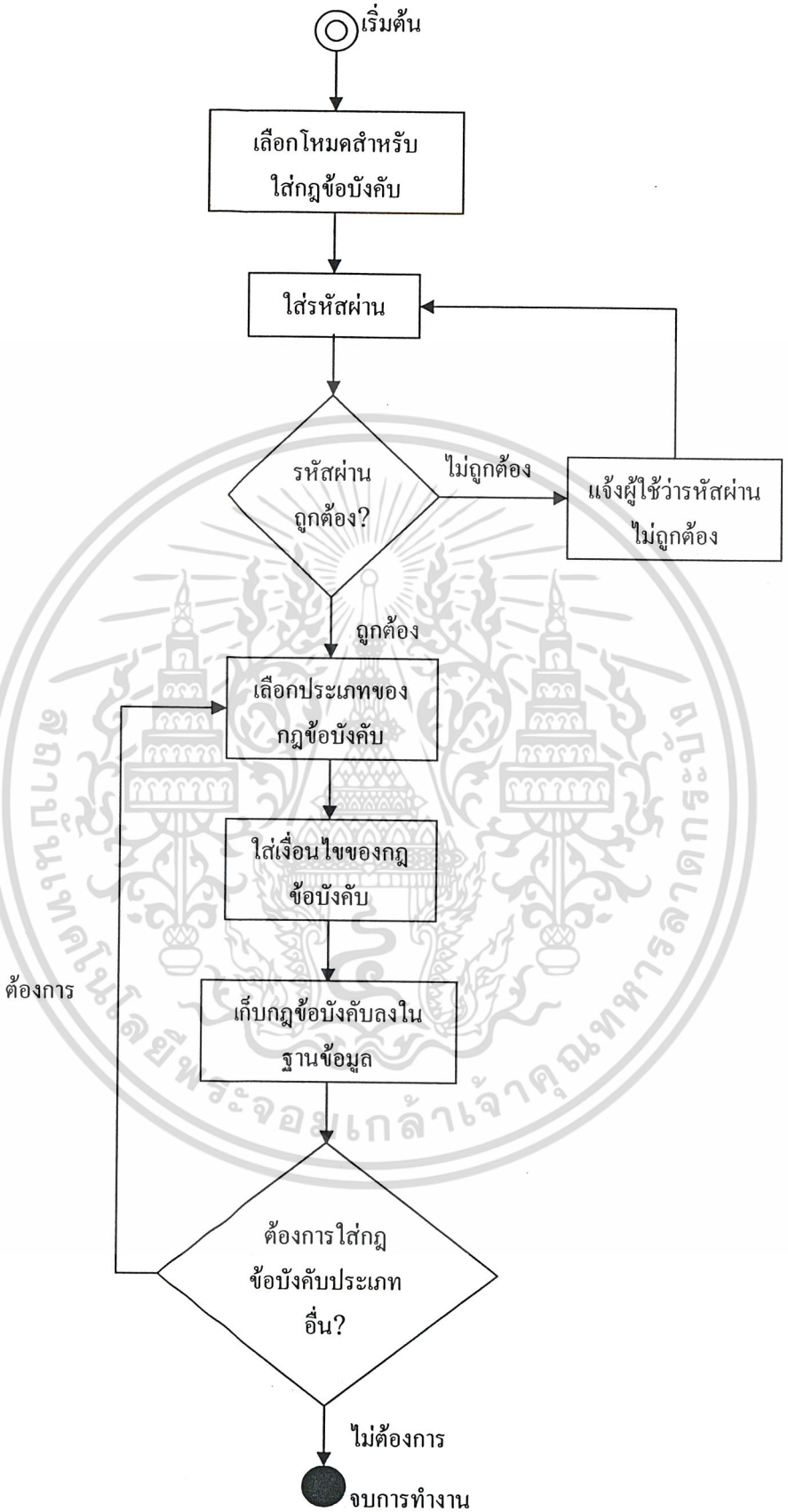
ตาราง 3-2 เงื่อนไขแบบความสัมพันธ์ระหว่างแอททริบิว

ATT1	TB1	SIGN1	VAL1	ATT2	TB2	SIGN2	VAL2
ADDRESS	CUSTOMER_TBL	0	'Bangkok'	CID	CUSTOMER_TBL	8	10000/40000
ADDRESS	CUSTOMER_TBL	10	'Bangkok','Chiangmai'	CREDIT_LIM	CUSTOMER_TBL	0	900000
ADDRESS	CUSTOMER_TBL	0	'Bangkok'	DISCOUNT	ORDER_TBL	4	20

ตาราง 3-3 เงื่อนไขแบบความสัมพันธ์ที่ขึ้นต่อกัน



รูป 3-2 ทิศทางการไหลของข้อมูล(Flow Chart) โหมดประมวลผลคำสั่ง



รูป 3-3 ทิศทางการไหลของข้อมูล(Flow Chart) โหมดกฎข้อบังคับ

บทที่ 4
การสร้างระบบ

ในบทนี้จะเกี่ยวกับการนำเอาทฤษฎีที่ได้ศึกษามาทำการสร้างระบบ โดยแต่ละเทคนิคที่ได้นำมาประยุกต์ใช้จะขึ้นอยู่กับประเภทของเงื่อนไขข้อบังคับตามที่ได้ออกแบบมาแล้วในบทของการออกแบบระบบ ซึ่งในโครงงานนี้จะนำเอาเงื่อนไขข้อบังคับเหล่านี้ มาทำการตรวจสอบและแปลงรูปแบบของคำสั่งคิวรี่ เพื่อให้ได้คำสั่งคิวรีที่มีประสิทธิภาพมากขึ้น โดยอาศัยหลักการของ Semantic Query Optimization

4.1 หลักการทำงานของระบบ

เริ่มต้นคำสั่งคิวรี่ของผู้ใช้จะถูกตรวจสอบโดยกฎข้อบังคับของผู้ดูแลระบบว่าขัดแย้งกับกฎข้อบังคับที่มีอยู่หรือไม่ (โดยวิธีการตรวจสอบจะลงรายละเอียดในหัวข้อ 4.2) ถ้าหากคำสั่งคิวรี่นั้นขัดแย้งกับกฎข้อบังคับ ก็จะทำการแจ้งให้ผู้ใช้ทราบว่าไม่มีคำตอบที่ผู้ใช้ต้องการอยู่ในฐานข้อมูล และจะไม่ส่งคำสั่งคิวรี่นั้นไปให้ระบบจัดการฐานข้อมูลประมวลผล แต่ถ้าหากคำสั่งคิวรี่นั้นไม่ขัดแย้งกับกฎข้อบังคับที่มีอยู่ ก็จะทำการส่งคำสั่งคิวรี่นั้นไปแปลงรูปเพื่อให้ได้คำสั่งคิวรี่ที่มีประสิทธิภาพมากขึ้น ซึ่งเทคนิคการแปลงรูปนั้นจะนำเอาเงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิวและเงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกันมาใช้ (โดยวิธีการแปลงรูปคำสั่งคิวรีของผู้ใช้โดยใช้เงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิวจะลงรายละเอียดในหัวข้อ 4.3 และ แปลงรูปคำสั่งคิวรี่ของผู้ใช้โดยใช้เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกันจะลงรายละเอียดในหัวข้อ 4.4) หลังจากนั้นจึงทำการส่งคำสั่งคิวรี่ของผู้ใช้และคำสั่งคิวรี่ที่ถูกแปลงรูปแล้วไปประมวลผลและทำการจับเวลา เพื่อมาเปรียบเทียบกัน

4.2 วิธีการสร้างฟังก์ชั่นในส่วนของการตรวจสอบในกรณีของค่าขอบเขต

หลักการคือรับค่าของเพรดดิเคตที่ต้องการตรวจสอบเข้ามา นำมาเปรียบเทียบกับกรณีแบบต่างๆ ของกฎของเงื่อนไขในฐานข้อมูลที่มีอยู่ โดยจะดูจากเพรดดิเคตที่มีชื่อของแอททริบิวเดียวกัน จะได้ค่าขอบเขตของแอททริบิวนั้น นำค่าของเพรดดิเคตที่ต้องการตรวจสอบมาตรวจหาโดยจะแบ่งออกเป็นกรณีต่างๆ โดยดูจากเครื่องหมายของเพรดดิเคตที่ต้องการตรวจสอบ มี 7 กรณีคือ = , != , < , <= , > , >= , <= and <= และ in เมื่อเข้าในกรณีต่างๆ เหล่านี้แล้วนำไปพิจารณาเทียบกับเพรดดิเคตที่เป็นกฎโดยจะมีเครื่องหมายอยู่ 11 กรณีคือ = , != , < , <= , > , >= , < and < , < and <= , <= and <= , <= and < และ in

ในกรณีแรก เพรดดิเคตที่เข้ามามีเครื่องหมาย = แบ่งตามเงื่อนไขของกฎ ดูจากเครื่องหมาย

= : จากกฎเครื่องหมาย = หมายความว่าแอททริบิวนี้มีค่า ค่านี้ค่าเดียว ถ้าค่าของเพรดดิเคตที่เข้ามานั้นไม่มีค่าตามนี้ก็จะถูกแจ้งให้ทราบว่าขัดแย้งกับกฎ

!= : จากกฎเครื่องหมาย != หมายความว่าแอททริบิวนี้จะไม่มีค่านี้ ถ้าค่าของเพรดดิเคตที่เข้ามานั้นมีค่านี้ก็จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

> : จากเครื่องหมาย > หมายความว่าแอททริบิวนี้จะมีค่ามากกว่าค่าหนึ่ง ถ้าเพรดดิเคตที่เข้ามานั้นมีค่าน้อยกว่าหรือเท่ากับค่าค่านั้น จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

< : จากเครื่องหมาย < หมายความว่าแอททริบิวนี้จะมีค่าน้อยกว่าค่าหนึ่ง ถ้าเพรดดิเคตที่เข้ามานั้นมีค่ามากกว่าหรือเท่ากับค่าค่านั้น จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

>= :จากเครื่องหมาย >= หมายความว่าแอททริบิวนี้จะมีค่ามากกว่าหรือเท่ากับค่าหนึ่ง ถ้าเพรดดิเคตที่เข้ามานั้นมีค่าน้อยกว่าค่าค่านั้น จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

<= :จากเครื่องหมาย <= หมายความว่าแอททริบิวนี้จะมีค่าน้อยกว่าหรือเท่ากับค่าหนึ่ง ถ้าเพรดดิเคตที่เข้ามานั้นมีค่ามากกว่าค่าค่านั้น จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

< and < : จากเครื่องหมาย < and < หมายความว่าแอททริบิวนี้จะอยู่ในช่วงที่มากกว่าค่าทางซ้ายและน้อยกว่าค่าทางขวา ถ้าเพรดดิเคตที่เข้ามานั้นมีค่าไม่อยู่ในช่วงนั้น จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

< and <= , <= and <= , <= and < : กรณีเช่นนี้ก็พิจารณาคล้ายกับกรณี < and < ว่าค่าเพรดดิเคตที่เข้ามาอยู่ในช่วงหรือไม่

in : จากเครื่องหมายนี้เป็นการกำหนดค่าของแอททริบิวในรูปแบบของเซต ว่ามีค่าใดอยู่บ้าง ถ้าค่าของเพรดดิเคตที่เข้ามานั้นไม่มีอยู่ในที่ระบุไว้ จะแจ้งให้ผู้ใช้งานได้ทราบว่าขัดแย้งกับกฎ

ในกรณีสอง เพรดดิเคตที่เข้ามามีเครื่องหมาย != พิจารณาแต่กรณีที่กำหนดกฎกรณี

= : จากกฎเครื่องหมาย = หมายความว่าแอททริบิวนี้มีค่า ค่านี้ค่าเดียว ถ้าค่าของเพรดดิเคตที่เข้ามานั้นบอกว่าหาค่าที่ไม่มีค่าเท่ากับค่านี้ จะพบว่าถ้าเข้าไปหาในฐานข้อมูลก็จะไม่พบ ดังนั้นจะแจ้งให้ผู้ใช้งานทราบว่าขัดแย้งกับกฎ

ในกรณีสาม เพรดดิเคตที่เข้ามามีเครื่องหมาย > พิจารณาแต่กรณีที่กำหนดกฎกรณี

= , < , <= : จะพิจารณาร่วมกันมาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่ามากกว่า 5 แต่ในกฎระบุว่าต้องมีค่าน้อยกว่าหรือเท่ากับ 4 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่ามากกว่าค่าที่อยู่ในกฎหรือไม่ถ้ามีค่ามากกว่าหรือเท่ากับก็จะขัดแย้งกับกฎ

< and < , < and <= , <= and <= , <= and < : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่ามากกว่าหรือเท่ากับค่าทางด้านขวาของกฎหรือไม่ ถ้ามากกว่าหรือเท่ากับก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

ในกรณีสี่ เพรดดิเคตที่เข้ามามีเครื่องหมาย < พิจารณาแต่กรณีที่กำหนดกฎกรณี

= , > , >= : จะพิจารณาร่วมกันมาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่าน้อยกว่า 5 แต่ในกฎระบุว่าต้องมีค่ามากกว่าหรือเท่ากับ 6 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้ จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่าน้อยกว่าหรือเท่ากับค่าที่อยู่ในกฎหรือไม่ ถ้ามีค่าน้อยกว่าหรือเท่ากับก็จะขัดแย้งกับกฎ

< and < , < and <= , <= and <= , <= and < : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่าน้อยกว่าหรือเท่ากับค่าทางด้านซ้ายของกฎหรือไม่ ถ้าน้อยกว่าหรือเท่ากับก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

ในกรณีห้า เพรดดิเคตที่เข้ามามีเครื่องหมาย >= พิจารณาแต่กรณีที่กำหนดกฎ

< : จะพิจารณาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่ามากกว่าหรือเท่ากับ 5 แต่ในกฎระบุว่าต้องมีค่าน้อยกว่า 4 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่ามากกว่าหรือเท่ากับค่าที่อยู่ในกฎหรือไม่ ถ้ามีค่ามากกว่าหรือเท่ากับก็จะขัดแย้งกับกฎ

= , <= : จะพิจารณาร่วมกันมาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่ามากกว่าหรือเท่ากับ 5 แต่ในกฎระบุว่าต้องมีค่าน้อยกว่าหรือเท่ากับ 4 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่ามากกว่าค่าที่อยู่ในกฎหรือไม่ถ้ามีค่ามากกว่าก็จะขัดแย้งกับกฎ

< and < , <= and < : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่ามากกว่าหรือเท่ากับค่าทางด้านขวาของกฎหรือไม่ ถ้ามากกว่าหรือเท่ากับก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

< and <= , <= and <= : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่ามากกว่าค่าทางด้านขวาของกฎหรือไม่ ถ้ามากกว่าก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

ในกรณีหก เพรดดิเคตที่เข้ามามีเครื่องหมาย <= พิจารณาแต่กรณีที่กำหนดกฎกรณี

> : จะมาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่าน้อยกว่าหรือเท่ากับ 5 แต่ในกฎระบุว่าต้องมีค่ามากกว่า 6 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้ จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่าน้อยกว่าหรือเท่ากับค่าที่อยู่ในกฎหรือไม่ ถ้ามีค่าน้อยกว่าหรือเท่ากับก็จะขัดแย้งกับกฎ

= , , >= : จะพิจารณาร่วมกันมาตรวจสอบดูว่ามีค่าขัดแย้งหรือไม่ เช่น ถ้าเพรดดิเคตที่เข้ามากำหนดว่ามีค่าน้อยกว่าหรือเท่ากับ 5 แต่ในกฎระบุว่าต้องมีค่ามากกว่าหรือเท่ากับ 6 ดังนั้นจะเห็นว่าเกิดการขัดแย้งกัน ในกรณีนี้ จะนำค่าของเพรดดิเคตที่เข้ามา มาดูว่ามีค่าน้อยกว่าค่าที่อยู่ในกฎหรือไม่ ถ้ามีค่าน้อยกว่าก็จะขัดแย้งกับกฎ

< and < , < and <= : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่าน้อยกว่าหรือเท่ากับค่าทางด้านซ้ายของกฎหรือไม่ ถ้าน้อยกว่าหรือเท่ากับก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

<= and <= , <= and < : พิจารณาดูว่าค่าเพรดดิเคตนั้นมีค่าน้อยกว่าค่าทางด้านซ้ายของกฎหรือไม่ ถ้าน้อยกว่าก็ทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้ง

ในกรณีเจ็ด เพรดดิเคตที่เข้ามามีเครื่องหมาย <= and <= พิจารณาแต่กรณีที่กำหนดกฎกรณี

= : พิจารณาดูว่าช่วงที่เข้ามานั้นมีค่าในกฎอยู่ในช่วงนั้นหรือไม่ ถ้าไม่มีเมื่อไปหาในฐานข้อมูลก็จะไม่พบ ดังนั้นจะแจ้งให้ผู้ใช้งานทราบได้ทันทีว่าขัดแย้งกับกฎ

> : พิจารณาดูว่าค่าทางด้านขวาของเพรดดิเคตมีค่าน้อยกว่าหรือเท่ากับ ค่าที่กำหนดหรือไม่ ถ้าน้อยกว่าหรือเท่ากับก็จะแจ้งให้ผู้ใช้งานทราบได้ทันทีว่าขัดแย้งกับกฎ เช่น ในกฎ A > 10 แล้ว เพรดดิเคตที่เข้ามาเป็น 5 <= A <= 15 กรณีนี้ 15 > 10 ก็จะไม่ขัดแย้งเพราะ A สามารถเป็น ค่า 11,12,13,14,15,16,17 ได้ แต่เพรดดิเคตที่เข้ามาต้องการในช่วง ถึง 15 เท่านั้นก็ไม่ขัดแย้งอะไรกับกฎ แต่ถ้า เพรดดิเคตที่เข้ามาอยู่ในรูปแบบ 5 <= A <= 10 กรณีนี้จะพบว่าช่วงที่ต้องการหาไม่เกิน 10 แต่ค่า A มีตั้งแต่ 11 เป็นต้นไปจะพบว่าไม่มีค่า A ที่ต้องการอยู่ในฐานข้อมูล ดังนั้นจะแจ้งให้ผู้ใช้งานทราบได้ทันทีว่าขัดแย้งกับกฎ

< : พิจารณาดูว่าค่าทางด้านซ้ายของเพรดดิเคตมีค่ามากกว่าหรือเท่ากับ ค่าที่กำหนดหรือไม่ ถ้ามากกว่าหรือเท่ากับก็จะแจ้งให้ผู้ใช้งานทราบได้ทันทีว่าขัดแย้งกับกฎ เช่น ในกฎ A < 10 แล้ว เพรดดิเคตที่เข้ามาเป็น 5 <= A <= 15 กรณีนี้ 5 < 10 ก็จะไม่ขัดแย้งเพราะ A สามารถเป็น ค่า 9,8,7 ได้ แต่เพรดดิเคตที่เข้ามาต้องการในช่วง ไม่น้อยกว่า 5 เท่านั้นก็ไม่ขัดแย้งอะไรกับกฎ แต่ถ้า เพรดดิเคตที่เข้ามาอยู่ในรูปแบบ 10 <= A <= 15 กรณีนี้จะพบว่าช่วงที่ต้องการหาไม่น้อยกว่า 10 แต่ค่า A มีตั้งแต่ 9 เป็นต้นไปจะพบว่าไม่มีค่า A ที่ต้องการอยู่ในฐานข้อมูล ดังนั้นจะแจ้งให้ผู้ใช้งานทราบได้ทันทีว่าขัดแย้งกับกฎ

< and < : พิจารณาดูว่าค่าในช่วงของเพรดดิเคตนี้อยู่ในช่วงที่กำหนดไว้หรือไม่ ถ้าไม่อยู่ก็จะทำการแจ้งไปบอกผู้ใช้งานว่าไม่พบ ตัวอย่างเช่น กฎมีอยู่ว่า 10 < B < 20 แล้วเพรดดิเคตที่เข้ามาจะไม่ขัดแย้งถ้าเพรดดิเคตอยู่ในช่วง เช่น 9 < B < 15 , 16 < B < 22 , 16 < B < 18 , 9 < B < 22 กรณีพวกนี้จะไม่ขัดแย้งจะสรุปได้ว่าพิจารณาว่า กรณีเหล่านี้จะไม่ขัดแย้งกับเงื่อนไขเมื่ออยู่ในกรณีใดกรณีหนึ่งดังนี้

ถ้าค่าทางด้านซ้ายของเพรดดิเคตมีค่าน้อยกว่าค่าทางด้านขวาของกฎ และ ค่าทางด้านซ้ายของเพรดดิเคตมีค่ามากกว่าค่าทางด้านซ้ายของกฎ

ถ้าค่าทางด้านขวาของเพรดดิเคตมีค่าน้อยกว่าค่าทางด้านขวาของกฎ และค่าทางด้านขวาของกฎมีค่ามากกว่าค่าทางด้านขวาของเพรดดิเคต

ถ้าค่าทางด้านขวาของเพรดดิเคตนั้นมีค่ามากกว่าหรือเท่ากับค่าทางด้านขวาของกฎ และค่าทางด้านซ้ายของกฎมีค่าน้อยกว่าค่าทางด้านซ้ายของกฎ

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

ในกรณีแปด เพรดดิเคตที่เข้ามามีเครื่องหมาย in พิจารณาแต่กรณีที่กำหนดกฎกรณี

= : ถ้าค่าในเซตของเพรดดิเคตที่เข้ามามีค่าใดค่าหนึ่งในนั้นเท่ากับที่กฎกำหนดอยู่ก็จะไม่ขัดแย้ง เช่น กฎบอกว่า A = 'HOME' แล้วเพรดดิเคตที่เข้ามาเป็น A in ('TEST', 'HOME') กรณีนี้ จะไม่ขัดแย้ง แต่ถ้า เพรดดิเคตเข้ามาเป็น A in ('TEST','TEMP') กรณีนี้ในกฎบอกว่า A จะเท่ากับ HOME แต่ที่ต้องการหาก็จะไม่พบในฐานข้อมูลดังนั้นจะทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้งกับกฎ

in : พิจารณาเหมือนกับกรณีเท่ากับ แต่พิจารณาได้หลายค่ามากขึ้นโดยถ้ามีค่าใดค่าหนึ่งของเพรดดิเคตอยู่ในค่าใน in ตัวใดตัวหนึ่งแล้วก็จะไม่ขัดแย้งกับกฎ ก็จะไปทำการหาค่าได้ทันที แต่ถ้าไม่มีค่าใดอยู่ในกฎเลยก็จะทำการแจ้งไปบอกผู้ใช้งานว่าขัดแย้งกับกฎ

4.3 วิธีการแปลงคำสั่งคิวรี่ของผู้ใช้โดยใช้เงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว

4.3.1. กรณีที่คำสั่งคิวรี่ของผู้ใช้ ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้

จากทฤษฎีการตรวจหาเงื่อนไขที่ไม่เป็นที่พอใจ (Detection of Unsatisfiable Conditions) ซึ่งจะทำการตรวจสอบเงื่อนไขของประโยค where ในคำสั่งคิวรี่ ว่ามีการขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้หรือไม่ ถ้าขัดแย้งก็จะทำการแจ้งผู้ใช้ให้ทราบว่าไม่มีคำตอบที่ต้องการอยู่ในฐานข้อมูล ซึ่งทำให้ไม่ต้องส่งคำสั่งคิวรี่ไปประมวลผลโดยเปล่าประโยชน์

ตัวอย่าง

ให้ คำสั่งคิวรี่ของผู้ใช้ คือ $A > 100$ and $B < 50$

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น $A \leq B$

ในกรณีนี้ จะเห็นว่า ในฐานข้อมูล A มีค่ามากกว่าหรือเท่ากับ B แต่คำสั่งคิวรี่ของผู้ใช้ที่เข้ามาบอกว่าถ้า A มีค่ามากกว่า 100 และ B จะมีค่าน้อยกว่า 50 ซึ่งค่าของ B จะต้องน้อยกว่าค่าของ A แน่นอน ดังนั้นคำสั่งคิวรี่ของผู้ใช้จึงขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้ โดยในฐานข้อมูลจะไม่มีคำตอบที่ใช้ต้องการอยู่ จะทำการแจ้งผู้ใช้ว่าไม่มีผลลัพธ์ที่ผู้ใช้ต้องการอยู่ในฐานข้อมูล โดยไม่ทำการส่งคำสั่งคิวรี่ไปให้ระบบจัดการฐานข้อมูลประมวลผล

4.3.2 กรณีที่คำสั่งคิวรี่ของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้

จากทฤษฎี Index introduction เป็นการเพิ่มเงื่อนไขใหม่ที่มาจากกฎข้อบังคับเข้าไปในคำสั่งคิวรี่ของผู้ใช้ โดยเงื่อนไขใหม่นั้นจะต้องมีแอททริบิวที่เป็น อินเด็กซ์อยู่ด้วย โดยคำสั่งคิวรี่ของผู้ใช้ใหม่นี้จะช่วยให้การค้นหาข้อมูลรวดเร็วยิ่งขึ้น

ตัวอย่าง

ให้ Query ของผู้ใช้ คือ $B \leq 100$

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น $A \leq B$

ในกรณีนี้คำสั่งคิวรี่ของผู้ใช้ที่เข้ามา บอกว่า B มีค่าน้อยกว่าหรือเท่ากับ 100 ส่วนในฐานข้อมูล ค่าของ A มีค่าน้อยกว่าหรือเท่ากับ B ทำให้ทราบว่า A จะมีค่าน้อยกว่าหรือเท่ากับ 100 ด้วย ถ้าหากแอททริบิว A เป็น อินเด็กซ์ซึ่งจากทฤษฎี index introduction เราสามารถเพิ่มเงื่อนไข $A \leq 100$ เข้าไปในคำสั่งคิวรี่ของผู้ใช้ได้

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ $B \leq 100$ and $A \leq 100$

4.4 วิธีการแปลงคำสั่งคิวรีของผู้ใช้โดยใช้เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน

4.4.1 กรณีที่คำสั่งคิวรี่ของผู้ใช้ ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้

จากทฤษฎี Detection of Unsatisfiable Conditions ซึ่งจะทำการตรวจสอบเงื่อนไขของประโยค where ว่ามีการขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้หรือไม่ ถ้าขัดแย้งก็จะทำการแจ้งผู้ใช้ให้

ทราบว่าไม่มีคำตอบที่ต้องการอยู่ในฐานข้อมูล ซึ่งทำให้ไม่ต้องส่งคำสั่งคิวรี่ไปประมวลผลโดยเปล่าประโยชน์

ตัวอย่าง

ให้ query ของผู้ใช้ คือ A > 5 and B = 5

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 $\longrightarrow$ B = 3

ในกรณีนี้ จะเห็นว่า ในฐานข้อมูล ถ้า A มีค่ามากกว่า 3 แล้ว B จะมีค่าเท่ากับ 3 เท่านั้น แต่คำสั่งคิวรี่ของผู้ใช้ที่เข้ามา บอกว่าถ้า A มีค่ามากกว่า 5 แล้ว B จะมีค่าเท่ากับ 5 ซึ่งค่า B เท่ากับ 5 นั้นไม่มีอยู่ในฐานข้อมูล จะทำการแจ้งผู้ใช้ว่าไม่มีผลลัพธ์ที่ผู้ใช้ต้องการอยู่ในฐานข้อมูล โดยไม่ทำการส่งคำสั่งคิวรี่ไปให้ระบบจัดการฐานข้อมูลประมวลผล

4.4.2 กรณีที่คำสั่งคิวรี่ของผู้ใช้ ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้

จะแบ่งออกเป็น 2 กรณี

4.4.2.1 เงื่อนไขของคำสั่งคิวรี่ของผู้ใช้ซ้ำซ้อนกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้

จากทฤษฎี Predicate Elimination ซึ่งจะสามารถกำจัดเงื่อนไขในคำสั่งคิวรี่ของผู้ใช้ที่ซ้ำซ้อนกับในฐานข้อมูลได้ โดยเงื่อนไขนั้นไม่มีแอททริบิวที่เป็นอินเด็กซ์อยู่เพื่อพัฒนาประสิทธิภาพของระบบ

ตัวอย่าง1

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 5 and B = 5

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 $\longrightarrow$ B = 5

ในกรณีนี้ จะเห็นว่า ในฐานข้อมูล ถ้า A มีค่ามากกว่า 3 แล้ว B จะมีค่าเท่ากับ 5 เท่านั้น และในคำสั่งคิวรี่ของผู้ใช้ที่เข้ามา บอกว่าถ้า A มีค่ามากกว่า 5 แล้ว B จะมีค่าเท่ากับ 5 เช่นกัน ซึ่งจะเห็นได้ว่า B = 5 เป็นเงื่อนไขของคำสั่งคิวรี่ของผู้ใช้มีความซ้ำซ้อนกับในฐานข้อมูล ซึ่งถ้า B ไม่ได้เป็น อินเด็กซ์จะสามารถกำจัดเงื่อนไข B = 5 ออกจากคำสั่งคิวรี่ของผู้ใช้ได้ แต่ถ้า B เป็น อินเด็กซ์ก็จะไม่กำจัดเงื่อนไขดังกล่าวออกไปได้

ซึ่งจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 5 (กรณีนี้ B ไม่ได้เป็น index)

ตัวอย่าง2

ให้ คำสั่งคิวรี่ ของผู้ใช้ คือ A > 5 and B > 3

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 $\longrightarrow$ B > 5

ในกรณีนี้จะเห็นว่าเงื่อนไข B > 3 ของคำสั่งคิวรี่ของผู้ใช้ สามารถกำจัดได้ (กรณีนี้ B ไม่ได้เป็นอินเด็กซ์) เพราะเงื่อนไขในฐานข้อมูล เมื่อ A>3 แล้ว B>5 ก็จะหมายความว่า เมื่อ A>5 แล้ว B>5 ด้วย ดังนั้น เงื่อนไข B>3 ของคำสั่งคิวรี่ของผู้ใช้จึงไม่จำเป็นต้องมี เพราะค่าในฐานข้อมูล B จะมีค่ามากกว่า 3 อยู่แล้ว

ตัวอย่าง3

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 5 and B > 5

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 → B > 3

ในกรณีนี้จะเห็นว่าเงื่อนไข B > 5 ของคำสั่งคิวรี่ของผู้ใช้ ไม่สามารถกำจัดได้ เพราะเงื่อนไขในฐานข้อมูล เมื่อ A>3 แล้ว B >3 ก็จะ หมายความว่า เมื่อ A>5 แล้ว B >3 ด้วย ซึ่ง B มีโอกาสที่จะเป็น 4 ,5, . . .ได้ ซึ่งในคำสั่งคิวรี่ของผู้ใช้ต้องการค่า B ที่มากกว่า 5 ขึ้นไปเท่านั้น ถ้าหากเรากำจัดเงื่อนไข B > 5 ออกไปจากคำสั่งคิวรี่ของผู้ใช้แล้ว ก็อาจจะมีโอกาสที่ B จะเท่ากับ 4 ได้ ซึ่งทำให้ได้ผลลัพธ์ที่ผู้ใช้ไม่ต้องการด้วย

4.4.2.2 เงื่อนไขของคำสั่งคิวรี่ของผู้ใช้ไม่ซ้ำซ้อนกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด

4.4.2.2.1 จากทฤษฎี scan reduction ซึ่งเป็นวิธีที่เพิ่มเงื่อนไขใหม่ที่มาจากกฎข้อบังคับเข้าไปในคำสั่งคิวรี่ของผู้ใช้ โดยเงื่อนไขใหม่ที่เพิ่มเข้าไปนี้จะช่วยลดช่วงของการค้นหาคำตอบ ทำให้การประมวลผลทำได้รวดเร็วยิ่งขึ้น

ตัวอย่างที่1

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3 and B > 5

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 → B < 9

ในกรณีนี้จะเห็นว่าเงื่อนไข B < 9 จะเป็นเงื่อนไขใหม่ที่เพิ่มเข้าไปในคำสั่งคิวรี่ของผู้ใช้ ซึ่งจะช่วยให้ช่วงในการค้นหาค่าของ B แคบลง

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 3 and B > 5 and B < 9

ตัวอย่างที่2

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3 and B < 10

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 → B > 3

ในกรณีนี้จะเห็นว่าเงื่อนไข B > 3 จะเป็นเงื่อนไขใหม่ที่เพิ่มเข้าไปในคำสั่งคิวรี่ของผู้ใช้ ซึ่งจะช่วยให้ช่วงในการค้นหาค่าของ B แคบลง

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 3 and B <10 and B >3

ตัวอย่างที่3

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3 and B < 10

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 → 7< B <15

ในกรณีนี้จะเห็นว่าเงื่อนไข B > 7 จะเป็นเงื่อนไขใหม่ที่เพิ่มเข้าไปในคำสั่งคิวรี่ของผู้ใช้ ซึ่งจะช่วยให้ช่วงในการค้นหาค่าของ B แคบลง

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 3 and B <10 and B >7

ตัวอย่างที่4

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3 and B > 10

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 → 7< B <15

ในกรณีนี้จะเห็นว่าเงื่อนไข B < 15 จะเป็นเงื่อนไขใหม่ที่เพิ่มเข้าไปในคำสั่งคิวรี่ขอผู้ใช้ ซึ่งจะช่วยให้ช่วงในการค้นหาค่าของ B แคบลง

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 3 and B >10 and B <15

ตัวอย่างที่5

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3 and B < 10

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 $\rightarrow$ 1< B <5

ในกรณีนี้จะเห็นว่าเงื่อนไขที่ควรจะถูกเพิ่มเข้าไปในคำสั่งคิวรี่ของผู้ใช้แทนที่เงื่อนไข B < 10 ควรจะมี 2 เงื่อนไข คือ B > 1 และ เงื่อนไข B < 5 ซึ่งจะช่วยให้ช่วงในการค้นหาค่าของ B แคบลง

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A > 3 and B >1 and B <5

4.4.2.2.2 จากทฤษฎี index introduction เป็นการเพิ่มเงื่อนไขใหม่ที่มาจากกฎข้อบังคับเข้าไปในคำสั่งคิวรี่ของผู้ใช้ โดยเงื่อนไขใหม่นั้นจะต้องมีแอททริบิวที่เป็น อินเด็กซ์อยู่ด้วย โดยคำสั่งคิวรี่ของ ผู้ใช้ใหม่นี้จะช่วยให้การค้นหาข้อมูลรวดเร็วยิ่งขึ้น

ตัวอย่าง

ให้คำสั่งคิวรี่ของผู้ใช้ คือ A > 3

กฎข้อบังคับที่ผู้ดูแลระบบกำหนด เป็น A>3 $\rightarrow$ B < 9

โดยที่แอททริบิว B เป็น อินเด็กซ์ซึ่งจากทฤษฎี index introduction เราสามารถเพิ่มเงื่อนไข B < 9 เข้าไปในคำสั่งคิวรี่ของผู้ใช้ได้

โดยจะได้คำสั่งคิวรี่ของผู้ใช้ใหม่ คือ A>3 and B < 9

บทที่ 5

ผลการทดลอง

ในการทดลองนี้ใช้ระบบจัดการฐานข้อมูล MS SQL Server 2000 โดยที่ SQL Server query optimizer โดยปกติตัว Optimizer มีการกำหนดเป็นแบบ cost-based optimizer ในการทดลองนี้จะทำการทดลองกับระบบจัดการฐานข้อมูล MS SQL Server 2000 ทั้งในแบบ Heuristic Optimization ซึ่งเป็นวิธีการแบบ rules-based และ แบบ Cost-Based Optimization

5.1 กฎข้อบังคับที่กำหนด

5.1.1 เงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว (Attribute Constraint)

CID <= 50000

CREDIT_LIM <= 1000000

500 <= CURR_BAL <= 600000

UNITPRICE > 2

ONHAND >=1

3 <= REORDER_QTY <= 300

SALARY <= 200000

EID <= 350

OID <= 30000

DISCOUNT IN (5, 10, 20, 25, 30, 40, 50, 60)

ADDRESS != 'USA'

QTY > 0

5.1.2 เงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว (Inter-Attribute Constraint)

UNITPRICE < REORDER_PTY

QTY <= ONHAND

CURR_BAL < CREDIT_LIM

ENAME != CNAME

5.1.3 เงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน (Functional Dependency Constraint)

เพรดดิเคต 1		*เพรดดิเคต 2*
ADDRESS = 'BANGKOK'	⟶	10000 <= CID <= 40000
ADDRESS = 'BANGKOK'	⟶	DISCOUNT >= 20
DISCOUNT > 50	⟶	EID = 298
PID <= 50	⟶	REORDER_QTY > 10

UNITPRICE < 50	⟶	QTY > 12
SALARY = 80000	⟶	EID > 260
ADDRESS IN ('BANGKOK','CHIANGMAI')	⟶	CREDIT_LIM = 900000
CNAME = 'HARRY'	⟶	80000 <= CURR_BAL <= 300000
CID < 70	⟶	ADDRESS != 'YALA'
2000 < ONHAND < 3000	⟶	200 < PID < 300
CREDIT_LIM <= 500000	⟶	CURR_BAL > 10000
UNITPRICE >= 500	⟶	PID >= 100
100 <= OID < 150	⟶	DISCOUNT = 30

5.2 ผลการทดสอบโดยใช้โปรแกรม Query Analyzer

โดยใน Query Analyzer นั้นใช้คำสั่ง

Set Statistics time on

Set statistics io on

Select ...

Set Statistics time off

Set statistics io off ในการดูค่าของเวลาและค่า Cost ที่ใช้ในการทำคำสั่งนั้น

ผลการทดสอบการจับเวลาของคำสั่งคิวรี่ของผู้ใช้เปรียบเทียบกับคำสั่งคิวรี่ที่ถูกแปลงรูปแล้วโดยใช้โปรแกรม Query Analyzer ซึ่งได้แบ่งรูปแบบของกรณีที่ทำการทดสอบไว้ 4 กรณี คือ

5.2.1 กรณีที่คำสั่งคิวรี่ของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนดไว้ และสามารถนำเอากฎข้อบังคับมาทำการลดช่วงการค้นหาได้ โดยใช้ทฤษฎี Scan reduction

ตัวอย่างที่ 1

คำสั่งคิวรี่ของผู้ใช้ :

```
Select *
From customer_tbl c, order_tbl o
Where c.cid=o.cid and c.address = 'bangkok' and o.discount < 30
```

กฎข้อบังคับ :

```
address = 'bangkok'  ->  discount >=20
```

คำสั่งคิวรี่ที่ถูกแปลงรูป:

```
select *
From customer_tbl c,order_tbl o
Where c.cid=o.cid and c.address = 'bangkok' and o.discount < 30
and  o.discount >=20
```

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-1-1 จะเป็นแบบ Rules-based Optimization และตาราง 5-1-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	1472	1430	1395	1439	1443
คำสั่งคิวรี่ที่ถูกแปลงรูป	793	783	783	788	802

ตารางที่ 5-1-1 ผลการทดลอง Scan Reduction

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	834	829	820	802	810
คำสั่งคิวรี่ที่ถูกแปลงรูป	743	775	737	750	756

ตารางที่ 5-1-2 ผลการทดลอง Scan Reduction

ตัวอย่างที่2

คำสั่งคิวรี่ของผู้ใช้ :

```
select *
From product_tbl p,order_tbl o
Where  o.pid=p.pid and p.unitprice<50 and o.qty <30
```

กฎข้อบังคับ : unitprice < 50 -> qty >12

คำสั่งคิวรี่ที่ถูกแปลงรูป:

```
select *
From product_tbl p,order_tbl o
Where  o.pid=p.pid and p.unitprice<50 and o.qty <30
and  o.qty > 12
```

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-2-1 จะเป็นแบบ Rules-based Optimization และตาราง 5-2-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	28	29	30	32	20
คำสั่งคิวรี่ที่ถูกแปลงรูป	20	21	26	19	18

ตารางที่ 5-2-1 ผลการทดลอง Scan Reduction2

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	31	31	32	33	32
คำสั่งคิวรี่ที่ถูกแปลงรูป	23	19	27	28	28

ตารางที่ 5-2-2 ผลการทดลอง Scan Reduction2

ตัวอย่างที่3

คำสั่งคิวรี่ของผู้ใช้ :

```
SELECT *
FROM customer_tbl
where credit_lim <= 500000  and curr_bal < 50000
```

กฎข้อบังคับ : credit_lim <= 500000 -> curr_bal > 10000

คำสั่งคิวรี่ที่ถูกแปลงรูป:

```
SELECT *
FROM customer_tbl
where credit_lim <= 500000  and curr_bal < 50000
and curr_bal > 10000
```

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดยตาราง 5-3-1 จะเป็นแบบ Rules-based Optimization และตาราง 5-3-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	165	157	170	155	160
คำสั่งคิวรี่ที่ถูกแปลงรูป	162	156	164	143	155

ตารางที่ 5-3-1 ผลการทดลอง Scan Reduction 3

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	176	188	176	142	175
คำสั่งคิวรี่ที่ถูกแปลงรูป	153	158	144	136	141

ตารางที่ 5-3-2 ผลการทดลอง Scan Reduction 3

5.2.2 กรณีที่คำสั่งคิวรี่ของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด โดยเพิ่มเงื่อนไขใหม่ที่มาจากกฎข้อบังคับเข้าไปใน คำสั่งคิวรี่ของผู้ใช้ โดยเงื่อนไขใหม่นั้นจะต้องมีแอททริบิวที่เป็นอินเด็กซ์อยู่ด้วย โดยใช้ทฤษฎี index introduction

ตัวอย่างที่1

คำสั่งคิวรี่ของผู้ใช้ :

```
select *
From employee_tbl e, order_tbl o
Where o.eid = e.eid  and o.discount >50
```

กฎข้อบังคับ : discount > 50 -> eid = 298
โดยที่ eid เป็นอินเด็กซ์

คำสั่งคิวรี่ที่ถูกแปลงรูป:

```
select *
From employee_tbl e, order_tbl o
Where o.eid = e.eid  and o.discount >50
and  e.eid = 298
```

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-4-1 จะเป็นแบบ Rules-based Optimization และตาราง 5-4-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	369	340	343	367	342
คำสั่งคิวรี่ที่ถูกแปลงรูป	282	284	305	289	301

ตารางที่ 5-4-1 ผลการทดลอง index introduction

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	228	241	230	234	230
คำสั่งคิวรี่ที่ถูกแปลงรูป	208	205	209	207	208

ตารางที่ 5-4-2 ผลการทดลอง index introduction

ตัวอย่างที่2

คำสั่งคิวรี่ของผู้ใช้ :

```
select *
From customer_tbl
Where address = 'bangkok'
```

กฎข้อบังคับ : address = 'bangkok'-> 10000<=cid <= 40000
โดยที่ cid เป็นอินเด็กซ์

คำสั่งคิวรี่ที่ถูกแปลงรูป: select *

From customer_tbl

Where address = 'bangkok'

and cid >=10000 and cid <=40000

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-5-1 จะเป็นแบบ Rules-based Optimization และตาราง 5-5-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	1185	1130	1193	1158	1168
คำสั่งคิวรี่ที่ถูกแปลงรูป	1140	1032	1051	1014	1077

ตารางที่ 5-5-1 ผลการทดลอง index introduction 2

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	1056	952	959	985	949
คำสั่งคิวรี่ที่ถูกแปลงรูป	855	822	836	844	859

ตารางที่ 5-5-2 ผลการทดลอง index introduction 2

5.2.3. กรณีที่คำสั่งคิวรี่ของผู้ใช้ไม่ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด โดยเงื่อนไขของคำสั่งคิวรี่ของผู้ใช้ซ้ำซ้อนกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด ซึ่งจะใช้ทฤษฎี Predicate Elimination เพื่อกำจัดเงื่อนไขที่ซ้ำซ้อนออกไปจากคำสั่งคิวรี่ของผู้ใช้

ตัวอย่าง

คำสั่งคิวรี่ของผู้ใช้ : select *

From product_tbl p, order_tbl o

Where o.pid = p.pid and unitprice < 50

and qty >10

กฎข้อบังคับ : unitprice < 50 -> qty >12

คำสั่งคิวรี่ที่ถูกแปลงรูป: select *

From product_tbl p, order_tbl o

Where o.pid = p.pid and unitprice < 50

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-6-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-6-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	101	112	93	110	115
คำสั่งคิวรี่ที่ถูกแปลงรูป	76	73	71	77	87

ตารางที่ 5-6-1 ผลการทดลอง Predicate Elimination

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	92	94	101	97	104
คำสั่งคิวรี่ที่ถูกแปลงรูป	86	89	88	88	86

ตารางที่ 5-6-2 ผลการทดลอง Predicate Elimination

5.2.4. กรณีที่คำสั่งคิวรี่ของผู้ใช้ขัดแย้งกับกฎข้อบังคับที่ผู้ดูแลระบบกำหนด จะไม่ส่งคำสั่งคิวรี่นั้นไปประมวลผล โดยใช้หลักการ Detection of Unsatisfiable Conditions

ตัวอย่าง1

คำสั่งคิวรี่ของผู้ใช้ :
```
select *
From customer_tbl
Where cid <50 and  address = 'yala'
```

กฎข้อบังคับ : cid <70 -> address != 'yala'

ในกรณีนี้คำสั่งคิวรี่ของผู้ใช้ขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรี่นั้นไปประมวลผล

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-7-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-7-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	3	2	1	4	1
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-7-1 ผลการทดลอง Detection of Unsatisfiable Conditions

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	1	1	1	1	1
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-7-2 ผลการทดลอง Detection of Unsatisfiable Conditions

ตัวอย่าง2

คำสั่งคิวรี่ของผู้ใช้ :

```
select *
From customer_tbl
Where curr_bal >10000 and  credit_lim < 5000
```

กฎข้อบังคับ : curr_bal < credit_lim

ในกรณีนี้คำสั่งคิวรี่ของผู้ใช้ขัดแย้งกับกฎข้อบังคับ จะไม่ส่งคำสั่งคิวรี่นั้นไปประมวลผล

ผลของการจับเวลาเมื่อส่งคำสั่งคิวรี่ทั้งสองไปประมวลผลโดยใช้โปรแกรม Query Analyzer โดย ตาราง 5-8-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-8-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	20	20	24	19	21
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-8-1 ผลการทดลอง Detection of Unsatisfiable Conditions2

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	23	19	19	24	20
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-8-2 ผลการทดลอง Detection of Unsatisfiable Conditions2

5.3 ผลการทดลองของโปรแกรม

5.3.1 ทดสอบกรณีที่เป็นการพบเงื่อนไขที่ขัดแย้งกับกฎความสัมพันธ์ที่ขึ้นต่อกัน คือ ไม่มีข้อมูลที่ต้องการอยู่ในฐานข้อมูล

คำสั่งที่รับเข้ามา (User query) :

```
SELECT *
FROM customer_tbl
WHERE address = 'bangkok' and cid = 3000
```

กฎที่เกี่ยวข้อง : ADDRESS = 'BANGKOK' ⟶ 10000 <= CID <= 40000

คำสั่งที่แปลงรูปแบบแล้ว (Transform query) :

```
SELECT *
FROM customer_tbl
WHERE address = 'bangkok' and cid = 3000
```

ผลที่ได้ : ตาราง 5-9-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-9-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	30	10	20	20	10
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-9-1 แสดงผลการทดลองของคำสั่ง

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	21	30	10	10	20
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-9-2 แสดงผลการทดลองของคำสั่ง

จำนวนแถวที่ได้รับ = 0

สรุปผล : ในกรณีนี้ เมื่อไม่พบข้อมูลอยู่ในฐานข้อมูล เนื่องจากทราบจากกฎ ก็จะไม่ต้องส่งคำสั่งลงไปประมวลผลที่ระบบจัดการฐานข้อมูล ทำให้ผลลัพธ์เป็นศูนย์

ในคำสั่งที่เข้ามานี้ได้ขัดแย้งกับกฎข้อกำหนดที่ว่า ถ้า Address มีค่าเท่ากับ “Bangkok” นั้นจะมีค่า Cid อยู่ในช่วง 10000 ถึง 40000 เท่านั้น ถ้าต้องการ Cid เท่ากับ 3000 นั้นจะไม่มีอยู่ในฐานข้อมูล โดยในขั้นแรกจะทำการรับข้อมูลคำสั่งเข้ามาแล้วตรวจสอบกับฟังก์ชั่นตรวจสอบของค่าขอบเขตแอททริบิวก่อน ดังรูปที่ 5-2 โดยในกรณีนี้จะมีกฎที่เกี่ยวข้องคือ Cid < 50000 จะมีการนำค่า 3000 และ 50000 ไปเปรียบเทียบกัน เมื่อพบว่าไม่ขัดแย้งกับกฎแล้ว ก็เข้าสู่กระบวนการแปลง โดยใช้กฎข้อบังคับของเงื่อนไขประเภทความสัมพันธ์ที่ขึ้นต่อกัน และเงื่อนไขประเภทความสัมพันธ์ระหว่างแอททริบิว ในคำสั่งที่รับเข้ามานี้มีกฎมีการนำมาตรวจสอบคือ

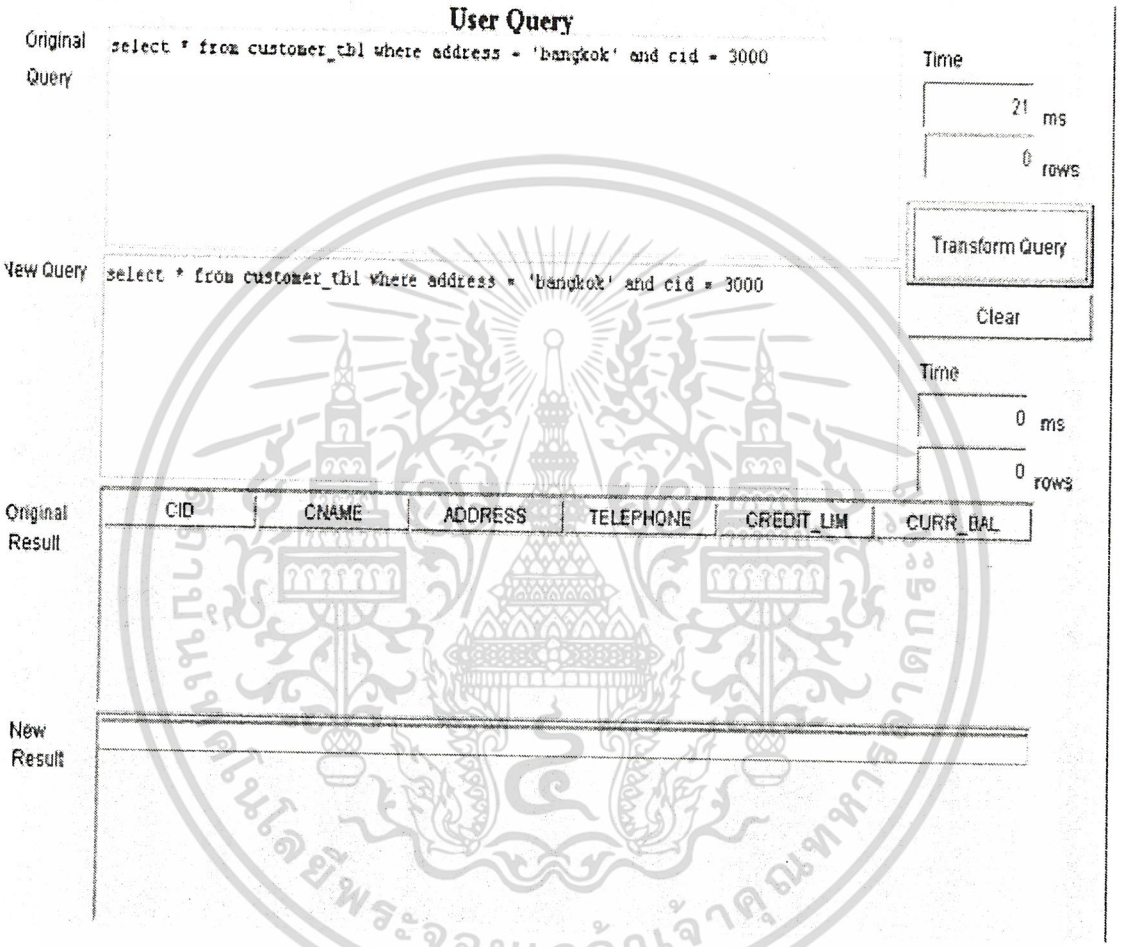
ADDRESS = ‘BANGKOK’ ⟶ 10000 <= CID <= 40000

ADDRESS = ‘BANGKOK’ ⟶ DISCOUNT >= 20

ADDRESS IN (‘BANGKOK’, ‘CHIANGMAI’) ⟶ CREDIT_LIM = 900000

โดยขั้นแรกนั้นทำการตรวจสอบว่าคำสั่งที่รับเข้ามานั้นมีเพรดดิเคตที่สอดคล้องกับกฎหรือไม่ WHERE address = ‘bangkok’ and cid = 3000 จะพบว่ามีกฎอยู่ 3 กฎ ที่ถูกนำมาใช้ ซึ่งจะเริ่มตรวจสอบที่กฎแรก เมื่อ กฎแรกมีเพรดดิเคตแรก ADDRESS = ‘BANGKOK’ ครอบคลุมเพรดดิเคตที่เข้ามา คือ address = ‘bangkok’ แล้วเราจะมาพิจารณาใช้กฎนั้น โดยนำเอา เพรดดิเคตหลัง 10000 <= CID <= 40000 มาพิจารณา ทำการวนลูปเพื่อตรวจสอบเงื่อนไขทั้งหมดของคำสั่งที่รับเข้ามา ตรวจสอบดูว่ามีเพรดดิเคตไหนมีค่าแอททริบิวเดียวกันกับ เพรดดิเคตหลังของกฎ หากพบก็จะทำการตรวจสอบดูว่าจะทำการลดช่วงโดยวิธีการ Scan Reduction หรือไม่ แต่ถ้าไม่พบก็จะมาพิจารณาว่าเพรดดิเคตนั้นมีแอททริบิวเป็นอินเด็กซ์ (Index Key) หรือไม่ถ้าเป็นก็จะทำการใส่เพรดดิเคตที่เป็นอินเด็กซ์เข้าไป จากที่ทดลองนั้นพบเพรดดิเคตที่

สอดคล้องกับเพรดดิเคตหลังของกฎ 10000 <= CID <= 40000 คือ cid = 3000 นำเพรดดิเคตนี้ไปตรวจเช็คว่ามีค่าของเงื่อนไขที่ต้องการนั้นขัดแย้งหรือไม่ จะพบว่าขัดแย้งเนื่องจากกฎบอกว่าถ้า address เท่ากับ Bangkok แล้ว cid จะต้องมากกว่า 10000 แล้วคำสั่งที่ต้องการ cid = 3000 มีค่าไม่ตรงที่กำหนดจึงขัดแย้ง ก็ทำการเซตค่าแฟลกซ์รีเจ็กไปบอกกับโปรแกรมหลัก เพื่อให้ไม่ต้องทำการส่งคำสั่งนี้ไปประมวลผลที่ระบบจัดการฐานข้อมูล



รูปที่ 5-1 แสดงหน้าจอผลลัพธ์โปรแกรมตามคำสั่งที่เข้ามา

```
select * from customer_tbl where address = 'bangkok' and cid = 3000

After pass function findatt- below -loop =2
address   0  'bangkok'   customer_tbl
cid   0  3000   customer_tbl
----- check attribute -----
attindb : ADDRESS
 table : CUSTOMER_TBL
 sign : 1
 value : 'USA'                                              - end -
case0
attindb : CID
 table : CUSTOMER_TBL
 sign : 5
 value : 50000                                              - end -
case0
3000 < 50000----------------------------
******** new query *********
att = address| sign 0| value 'bangkok'| tbl customer_tbl
att = cid| sign 0| value 3000| tbl customer_tbl
 loop = count = 2
when check predicate match

--  query -- Attribute + address+ sign = 0 value = 'bangkok' table = customer_tbl
--database-- Attribute + ADDRESS+ sign = 0 value = 'Bangkok' table = CUSTOMER_TBL
case '='

value predicate = 0
-0 match
-1 cover range
-2 not use

---------start check B when A match A->B -----------
false:

 In fn predicate2  check value of rule and value of query

in query     : sign  0 | att= address | table = customer_tbl | value = 'bangkok'

in rule of fd: sign  8 | att= CID | table = CUSTOMER_TBL | value = 10000/40000

 In fn predicate2  check value of rule and value of query

in query     : sign  0 | att= cid | table = customer_tbl | value = 3000

in rule of fd: sign  8 | att= CID | table = CUSTOMER_TBL | value = 10000/40000
case0

---------- end check B  A->B ------------
reject = true -
```

รูปที่5-2 เอาท์พุทที่แสดงออกค่าที่หน้าจอขณะทดสอบ

5.3.2 ทดสอบกรณีที่เป็นการพบเงื่อนไขที่มีการเพิ่มเพรดิเคตที่เป็นอินเด็กซ์เข้ามาและตัดเพรดิเคตที่ทำงานซ้ำซ้อน

คำสั่งที่รับเข้ามา (User query) :

SELECT *

FROM customer_tbl c, order_tbl o

WHERE c.cid = o.cid and c.address = 'bangkok' and o.discount > 10

กฎที่เกี่ยวข้อง : ADDRESS = 'BANGKOK' ⟶ 10000 <= CID <= 40000

ADDRESS = 'BANGKOK' ⟶ DISCOUNT >= 20

คำสั่งที่แปลงรูปแบบแล้ว (Transform query) :

```
SELECT *
FROM customer_tbl c, order_tbl o
WHERE c.cid = o.cid and c.address = 'bangkok' and c.cid between 10000 and 40000
```

ผลที่ได้ : ตาราง 5-10-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-10-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	841	891	821	841	852
คำสั่งคิวรี่ที่ถูกแปลงรูป	691	671	701	661	670

ตารางที่ 5-10-1 แสดงผลการทดลองของคำสั่ง

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	1122	1372	1072	1232	1272
คำสั่งคิวรี่ที่ถูกแปลงรูป	961	1242	971	1071	981

ตารางที่ 5-10-2 แสดงผลการทดลองของคำสั่ง

จำนวนแถวที่ได้รับ = 20832

สรุปผล : ในกรณีนี้เป็นการตัดเพรดดิเคตที่ซ้ำซ้อนออก และทำการเพิ่มเพรดดิเคตที่มีแอททริบิวเป็นอินเด็กซ์เข้าไปจะทำให้เร็วขึ้น

โดยขั้นแรกนั้นทำการตรวจเช็คว่าคำสั่งที่รับเข้ามานั้นมีเพรดดิเคตที่สอดคล้องกับกฎหรือไม่ WHERE address = 'bangkok' and o.discount > 10จะพบว่ามีกฎอยู่ 3 กฎ ที่ถูกนำมาใช้

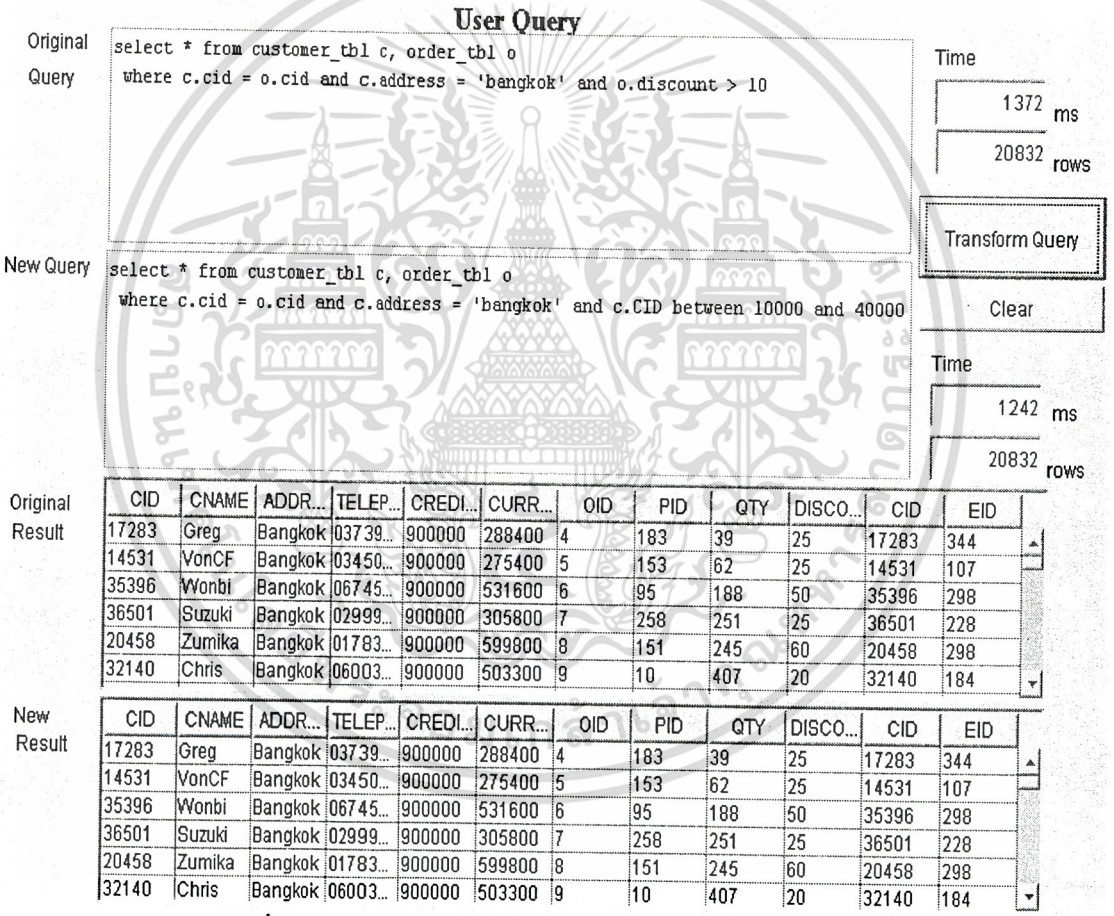
ADDRESS = BANGKOK' ⟶ 10000 <= CID <= 40000

ADDRESS = 'BANGKOK' ⟶ DISCOUNT >= 20

ADDRESS IN ('BANGKOK','CHIANGMAI') ⟶ CREDIT_LIM = 900000

ซึ่งจะเริ่มตรวจสอบที่กฎแรก เมื่อกฎแรกมีเพรดดิเคตแรก ADDRESS = 'BANGKOK' ครอบคลุมเพรดดิเคตที่เข้ามา คือ address = 'bangkok' แล้วเราจะมาพิจารณาใช้กฎนั้น โดยนำเอา เพรดดิเคตหลัง 10000 <= CID <= 40000 มาพิจารณา ทำการวนลูปเพื่อตรวจสอบเงื่อนไขทั้งหมดของคำสั่งที่รับเข้ามา ตรวจเช็คดูว่ามีเพรดดิเคตไหน มีค่าแอททริบิวเดียวกันกับเพรดดิเคตหลังของกฎ หากพบก็จะทำการตรวจสอบดูว่าจะทำการลดช่วงโดยวิธีการ Scan Reduction หรือไม่แต่ถ้าไม่พบก็จะมาพิจารณาว่าเพรดดิเคตนั้นมีแอททริบิวเป็นอินเด็กซ์(Index Key) หรือไม่ถ้าเป็นก็จะทำการใส่เพรดดิเคตที่เป็นอินเด็กซ์เข้าไป จากที่ทดลองนั้นไม่มีเพรดดิเคตที่สอดคล้องกับเพรดดิเคตหลังของกฎ 10000 <= CID <= 40000 จึงนำเพรดดิเคตนี้ไปตรวจเช็คว่ามีแอททริบิวที่เป็นอินเด็กซ์หรือไม่ ในกรณีนี้ CID เป็นอินเด็กซ์ จึงทำการเพิ่มเพรดดิเคตนี้เข้าไปในคำสั่งคิวรี่และทำการพิจารณากฎต่อไป คือ ADDRESS = 'BANGKOK' ⟶ DISCOUNT >= 20 เพรดดิเคตแรกของกฎนั้นครอบคลุมในคำสั่งคิวรี่ที่เข้ามาดังนั้นมาพิจารณาหาว่า เพรดดิเคตใดในคำสั่งที่รับเข้ามา เกี่ยวข้องกับ DISCOUNT ก็จะพบ o.discount > 10 ของคำสั่งคิวรี่ ดังนั้นก็นำค่าทั้งสองมา

ตรวจสอบกันว่าขัดแย้งหรือไม่ จะพบว่า สามารถตัดเพรดดิเคตของคำสั่งคิวรี่ออกได้เนื่องจากว่าในกฎมีการระบุไว้แล้วว่า discount จะมากกว่าหรือเท่ากับ 20 ในกรณีที่ address เท่ากับ Bangkok มาพิจารณาที่กฎต่อไป ADDRESS IN ('BANGKOK','CHIANGMAI') ⟶ CREDIT_LIM = 900000 ไม่พบว่า มีเพรดดิเคตที่เกี่ยวของคำสั่งที่เกี่ยวข้องกับ credit_lim พอหมดกฎของเพรดดิเคตนี้ของคำสั่งก็ไปดูยังเพรดดิเคตต่อไป ซึ่งเพรดดิเคตที่เป็น o.discount > 10 ถูกตัดไปตามหลักของ Predicate Eliminate ส่วน cid ถูกเพิ่มเข้ามาในคำสั่ง มาพิจารณาที่ cid มีกฎที่ใช้คือ CID < 70 ⟶ ADDRESS != 'YALA' พบว่าไม่มีเพรดดิเคตไหนในคำสั่งถูกครอบคลุมด้วยเงื่อนไข CID < 70 ดังนั้นไม่ต้องพิจารณาต่อ เมื่อหมดการวนทุกเพรดดิเคตของคำสั่งแล้วก็จะเสร็จการแปลงคำสั่ง และแสดงเวลาของ คำสั่งคิวรี่ที่แปลงแล้วกับที่ยังไม่ถูกแปลง จะได้ผลลัพธ์ดังรูป 5-3



รูปที่ 5-3 แสดงหน้าจอผลลัพธ์โปรแกรมตามคำสั่งที่เข้ามา

5.3.3 ทดสอบกรณีที่เป็นการพบการขัดแย้งกับเงื่อนไขประเภทกำหนดค่าขอบเขตของแอททริบิว คือ ไม่มีข้อมูลที่ต้องการอยู่ในฐานข้อมูลเพราะแอททริบิวนั้นมีการกำหนดค่าขอบเขตเอาไว้แล้ว

คำสั่งที่รับเข้ามา (User query) :

```
SELECT *
FROM employee_tbl
WHERE salary > 250000
```

กฎที่เกี่ยวข้อง : SALARY <= 200000

คำสั่งที่แปลงรูปแบบแล้ว (Transform query) :

```
SELECT *
FROM employee_tbl
WHERE salary > 250000
```

ผลที่ได้ : ตาราง 5-11-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-11-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	30	10	10	20	10
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-11-1 แสดงผลการทดลองของคำสั่ง

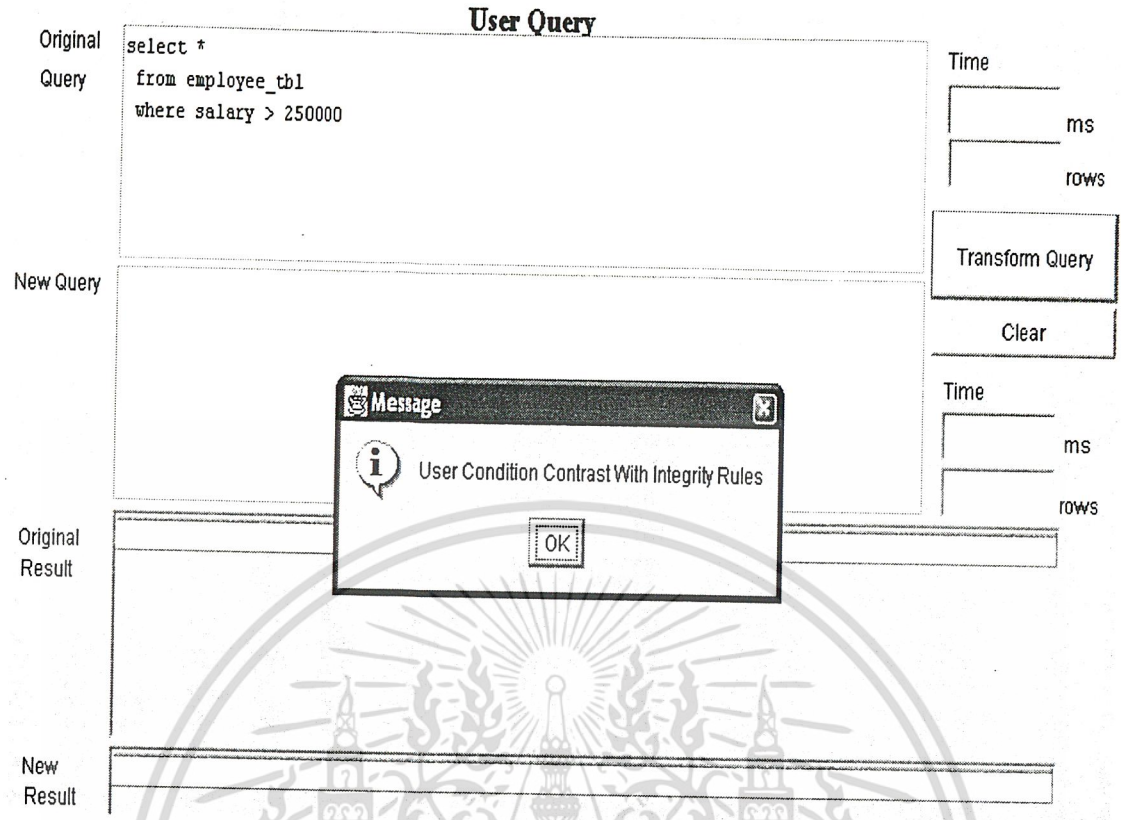
Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	20	10	10	10	20
คำสั่งคิวรี่ที่ถูกแปลงรูป	0	0	0	0	0

ตารางที่ 5-11-2 แสดงผลการทดลองของคำสั่ง

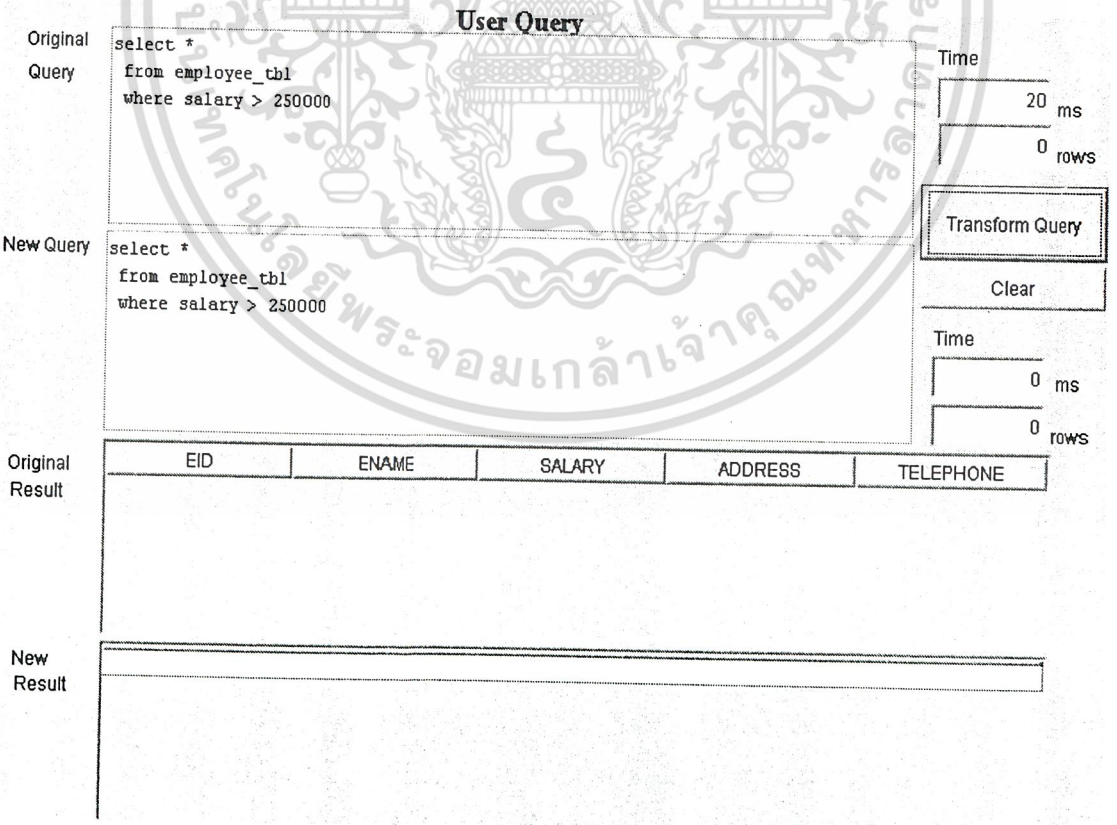
จำนวนแถวที่ได้รับ = 0

สรุปผล : ในกรณีนี้ เมื่อไม่พบข้อมูลอยู่ในฐานข้อมูล เนื่องจากทราบจากกฎประเภทกำหนดค่าขอบเขตของแอททริบิว ก็จะไม่ต้องส่งคำสั่งลงไปประมวลผลที่ระบบจัดการฐานข้อมูล ทำให้ผลลัพธ์เป็นศูนย์

ในกรณีนี้ทำเกิดในกรณีที่ตรวจสอบเจอตั้งแต่ตอนตรวจเงื่อนไขที่ขัดแย้งกับกฎประเภทกำหนดค่าขอบเขตของแอททริบิว โดยกฎมีอยู่ว่า SALARY <= 200000 แต่คำสั่งที่ใส่เข้ามาต้องการหาข้อมูลที่มีค่า SALARY มากกว่า 250000 เมื่อนำมาเปรียบเทียบกันแล้วจะพบว่ามันขัดแย้ง จึงทำการส่งไปบอกผู้ใช้งานว่าหาข้อมูลไม่พบโดยที่ไม่ต้องส่งคำสั่งไปประมวลผล



รูปที่ 5-4 แสดงไดอะล็อกบล็อกแสดงให้ทราบว่าขัดแย้ง



รูปที่ 5-5 แสดงหน้าจอผลลัพธ์โปรแกรมตามคำสั่งที่เข้ามา

5.2.4 ทดสอบกรณีที่เป็นการพบเงื่อนไขที่ใช้กฎความสัมพันธ์ที่ขึ้นต่อกันมาทำการลดช่วงการค้นหา

คำสั่งที่รับเข้ามา (User query) :

```
SELECT *
FROM product_tbl p, order_tbl o
WHERE p.pid=o.pid and p.unitprice < 50 and o.qty < 30
```

กฎที่เกี่ยวข้อง : UNITPRICE < 50 ⟶ QTY > 12

คำสั่งที่แปลงรูปแบบแล้ว (Transform query) :

```
SELECT *
FROM product_tbl p, order_tbl o
WHERE p.pid=o.pid and p.unitprice < 50 and o.qty < 30 and o.qty > 12
```

ผลที่ได้ : ตาราง 5-12-1 จะเป็นแบบ Rules-Based Optimization และตาราง 5-12-2 เป็นแบบ Cost-Based Optimization

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	40	31	30	30	40
คำสั่งคิวรี่ที่ถูกแปลงรูป	20	10	20	10	30

ตารางที่ 5-12-1 แสดงผลการทดลองของคำสั่ง

Query	ครั้งที่ 1 (ms)	ครั้งที่ 2 (ms)	ครั้งที่ 3 (ms)	ครั้งที่ 4 (ms)	ครั้งที่ 5 (ms)
คำสั่งคิวรี่ของผู้ใช้	61	30	30	40	71
คำสั่งคิวรี่ที่ถูกแปลงรูป	30	20	20	30	50

ตารางที่ 5-12-2 แสดงผลการทดลองของคำสั่ง

จำนวนแถวที่ได้รับ = 12

สรุปผล : ในกรณีนี้เป็นการใช้วิธีการเพิ่มเพรดิเคตเข้ามาเพื่อทำการลดช่วงการค้นหา ทำให้ขอบเขตของการค้นหาลดน้อยลง

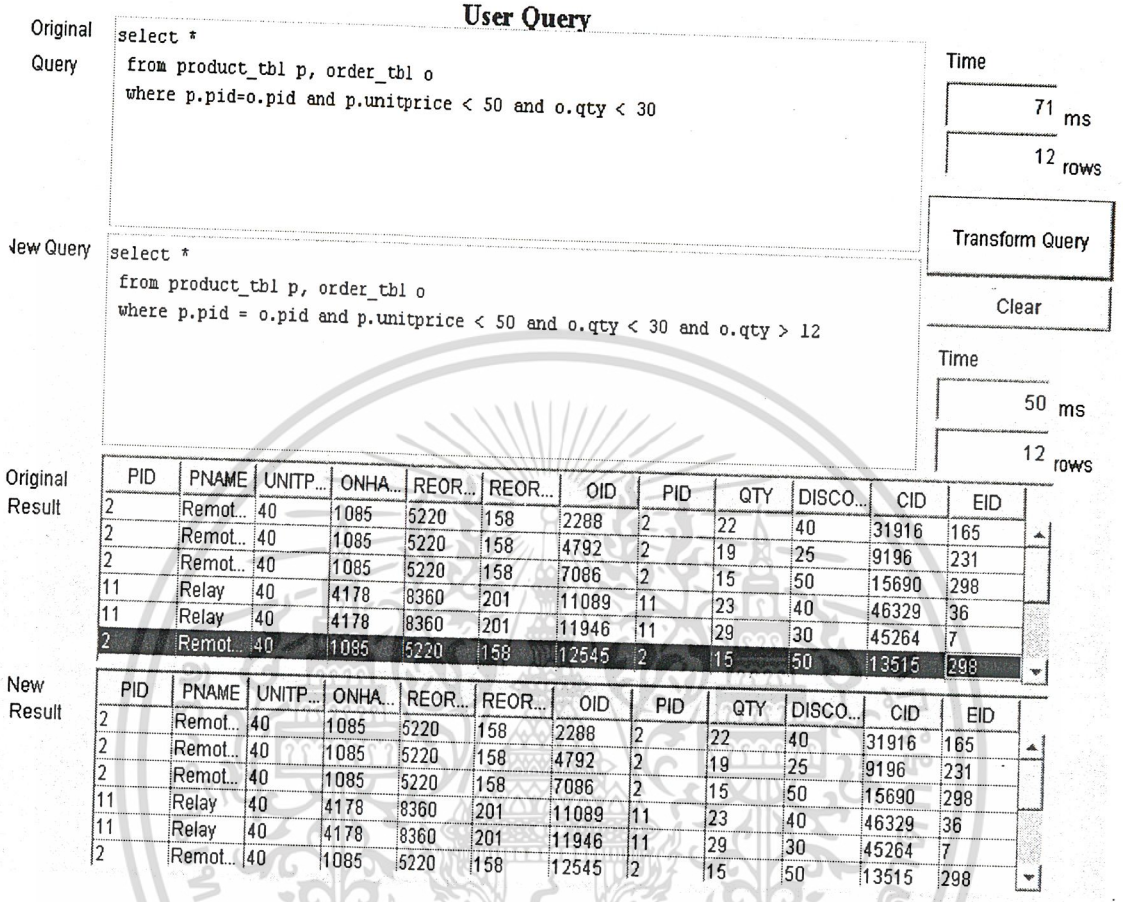
โดยเริ่มจากการตรวจสอบที่เพรดิเคตแรกของคำสั่งที่เข้ามา คือ p.pid = o.oid พบว่าไม่มีเงื่อนไข PID <= 50 ⟶ REORDER_QTY > 10 ไม่มีค่าสอดคล้องกัน ค่าในกฎไม่ครอบคลุมกับค่าในคำสั่ง ดูที่เพรดิเคตต่อไปของคำสั่งที่เข้ามาคือ p.unitprice < 50 จะได้กฎที่นำมาตรวจสอบคือ

UNITPRICE >= 500 ⟶ PID >= 100

UNITPRICE < 50 ⟶ QTY > 12

จากกฎเห็นว่ากฎแรกนั้นไม่ครอบคลุมเพรดิเคตของคำสั่งที่เข้ามาใดๆ แต่กฎที่สองครอบคลุมกับเพรดิเคตของคำสั่งที่สอง ดังนั้นมาพิจารณาที่เพรดิเคตตัวหลังของกฎ คือ QTY > 12 นำมาหาว่ามีเพรดิเคตไหนในคำสั่งที่สอดคล้องกันก็พบว่ามีเพรดิเคต o.qty < 30 จะทำการตรวจสอบว่ามีค่าขัดแย้งหรือไม่ ใน

กรณีนี้ไม่ขัดแย้งและพบว่า เข้ากรณีการเพิ่มเพรดิเคตที่ทำการลดช่วงเนื่องจากมีเพรดิเคตที่มีชื่อแอททริบิวนั้นมีอยู่แล้วในคำสั่งที่เข้ามา ทำให้เพิ่ม QTY > 12 เข้าไปในคำสั่งที่เข้ามา จะมีค่าผลลัพธ์ดังรูปที่ 5-6



รูปที่ 5-6 แสดงหน้าจอผลลัพธ์โปรแกรมตามคำสั่งที่เข้ามา

บทที่ 6
สรุปผลการทดลอง

6.1 สรุปผลการทดลอง

จากที่ทำการทดลองใช้หลักการของเทคนิคหลักๆ อยู่ 3 แบบ คือ การเพิ่มเพรดดิเคตเข้าไปในคำสั่ง (Predicate Introduction), เทคนิคการตัดเพรดดิเคต (Predicate Eliminate) เป็นการตัดเพรดดิเคตที่ซ้ำซ้อนกับกฎออกจากคำสั่งคิวรี่ของผู้ใช้งาน และเทคนิคการตรวจหาเงื่อนไขที่มีคุณสมบัติไม่เหมาะสม (Detection of Unsatifiable Condition) การตรวจสอบคำสั่งคิวรี่ที่เข้ามาว่ามีเงื่อนไขขัดแย้งกับกฎหรือไม่ โดยถ้าเกิดการขัดแย้งขึ้นจะไม่ส่งคำสั่งคิวรี่ไปประมวลผลที่ระบบจัดการฐานข้อมูล ทำให้ลดภาระการทำงานของระบบจัดการฐานข้อมูล โดยที่ Predicate Introduction นั้นมีการเพิ่มเพรดดิเคตเข้าไปสองแบบ คือแบบ Index Introduction เป็นการเพิ่มเพรดดิเคตที่มีอินเด็กซ์อยู่เข้าไปในคำสั่งคิวรี่และแบบ Scan Reduction คือการเพิ่มเพรดดิเคตเข้าไปเพื่อทำการลดขอบเขตของการค้นหาคำตอบของคำสั่งคิวรี่

ซึ่งจากผลการทดลองนั้นนำค่าข้อมูลมาทำการเปรียบเทียบเวลาของการประมวลผลพบว่าเทคนิคแบบ Introduction มีประสิทธิภาพในการแปลงคำสั่ง ได้ดีที่สุด เมื่อดูจากความแตกต่างของเวลาระหว่างคำสั่งที่ยังไม่ถูกแปลงและคำสั่งที่ถูกแปลงแล้ว เทคนิคที่มีความแตกต่างรองลงมาคือวิธีการแบบ Scan Reduction และ Detection of Unsatisfiable Condition จะเห็นได้ชัดเจนถึงความแตกต่างของเวลาในกรณีที่คำสั่งคิวรี่นั้นมีการค้นหาคำตอบในฐานข้อมูลที่มีข้อมูลจำนวนมาก และแบบ Predicate Elimination นั้น มีความแตกต่างของเวลาไม่ค่อยชัดเจน

6.2 ปัญหาที่พบ

การนำเทคนิคแบบ Predicate Elimination มาใช้ในการแปลงนั้นพบว่าค่าความแตกต่างของเวลามีความไม่ชัดเจน

บางกรณีค่าของคำสั่งที่ถูกแปลงนั้น มีค่าเวลาที่มากกว่าคำสั่งที่ไม่ถูกแปลง ซึ่งอาจเกิดจากสภาพแวดล้อมของระบบในขณะนั้น

6.3 ข้อเสนอแนะ

ควรจะมีการเพิ่มขนาดของข้อมูลให้มากขึ้น เพื่อให้เห็นถึงความแตกต่างของเวลาที่ออกมาได้อย่างชัดเจน

6.4 แนวทางการพัฒนาต่อ

6.4.1 พัฒนาโปรแกรมให้รองรับคำสั่งประเภท having, group by, order by

6.4.2 การทำ คำสั่งย่อย subquery เพื่อให้โปรแกรมทำงานได้อย่างมีประสิทธิภาพ

6.4.3 เพิ่มความสามารถในด้านการแก้ไขกฎข้อบังคับที่มีอยู่ ลบกฎข้อบังคับ

6.4.4 เพิ่มฟังก์ชั่นการตรวจเช็คคำสั่งที่เข้ามาว่าขัดแย้งกันเองหรือไม่

ภาคผนวก

ภาคผนวก ก

SQL Server 2000

ระบบฐานข้อมูลถือเป็นหัวใจสำคัญของการทำแอพพลิเคชั่นทั่วไป เนื่องจากเป็นแหล่งที่เก็บรวบรวมข้อมูล ซึ่งจะถูกแปรเปลี่ยนให้สามารถนำไปวิเคราะห์ต่อการใช้งานได้ต่อไป โดย SQL Server 2000 เป็นโปรแกรมจัดการฐานข้อมูลระดับเซิร์ฟเวอร์ที่มีขีดความสามารถในการรองรับข้อมูลขนาดใหญ่ ช่วยให้การบริหารจัดการฐานข้อมูลมีประสิทธิภาพ รวดเร็ว สามารถตอบสนองต่อความต้องการขององค์กรขนาดใหญ่ได้ดี SQL Server 2000 แบ่งออกได้หลายเวอร์ชันตามลักษณะการใช้งานดังนี้

- **Personal Edition** เป็นเวอร์ชันที่ออกแบบมาเพื่อใช้กับงานฐานข้อมูลที่มีขนาดเล็ก โดยสามารถใช้งานได้เฉพาะภายในเครื่องโลคอล(local)เท่านั้น หรืออาจกล่าวได้ว่าเป็นแบบ Stand Alone ส่วนใหญ่นิยมนำมาใช้ในการทดสอบการเขียนแอพพลิเคชั่น

- **Standard Edition** เป็นเวอร์ชันมาตรฐานที่ใช้สำหรับฐานข้อมูลทั่วไป สามารถรองรับการใช้งานของเครื่องเซิร์ฟเวอร์ที่มีขนาดโพรเซสเซอร์ได้สูงสุด 4 CPU หน่วยความจำสูงสุด 2 GB

- **Enterprise Edition** เป็นเวอร์ชันที่ขยายขีดความสามารถจากเวอร์ชันมาตรฐาน เพื่อให้สามารถรองรับการทำงานได้กับเครื่องเซิร์ฟเวอร์ที่มีขนาดโปรเซสเซอร์สูงสุดถึง 32 CPU หน่วยความจำสูงสุด 64 GB นิยมนำมาใช้งานกับองค์กรที่มีขนาดใหญ่และมีสาขามากมาย เพราะมีระบบสนับสนุนการทำงานมากมาย เช่น การทำ Data Mining, Data Warehouse เป็นต้น โดย SQL Server 2000 มีวิธีการจัดการระบบฐานข้อมูล ทั้งการสร้างตาราง เปลี่ยนแปลง หรือ ทำลาย รวมถึงการกำหนดออปชั่นต่างๆ ของฐานข้อมูล โดยตารางจะเป็นส่วนสำคัญหลักของระบบฐานข้อมูล ดังนั้นจะอธิบายถึงส่วนประกอบต่างๆของตาราง ประเภทของข้อมูลที่จัดเก็บ ตลอดจน การใช้ภาษา SQL เพื่อทำคำสั่งคิวรี่ข้อมูลที่อยู่ในตาราง

ประโยคคำสั่ง SELECT

เป็นคำสั่งสำหรับเรียกดูข้อมูลจากตาราง หรือ แสดงเรคคอร์ดจากตารางต่างๆ ที่ต้องการ มีรูปแบบประโยคคำสั่งการใช้งาน 2 กรณีคือ

1. การเรียกดูข้อมูลแบบไม่มีเงื่อนไข เป็นการแสดงข้อมูลทุกแถวที่อยู่ในแต่ละตารางมีรูปแบบประโยคคือ

SELECT *column_name1, column_name2,.....* **FROM** *table_name*

เมื่อคีย์เวิร์ด SELECT จะตามด้วยชื่อคอลัมน์ที่ต้องการแสดงซึ่งจะคั่นด้วยเครื่องหมายจุลภาคไปเรื่อยๆ ซึ่งใช้ระบุคอลัมน์ที่จะถูกคืนค่ากลับมาจากการคิวรี่ และ FROM ตามด้วยชื่อตารางที่ต้องการ เช่น SELECT [Flight number], Date, [available seats] FROM [Flight Instance] เป็นการคืนค่าในคอลัมน์ Flight number, Date และ available seats สำหรับทุกแถว จากตาราง Flight Instance ซึ่งผลที่ได้จะแสดงดังตารางที่ ก-1 จะเห็นได้ว่าถ้าต้องการแสดงคอลัมน์ทั้งหมดภายในตารางสำหรับทุกๆ แถวจะใช้เครื่องหมาย (*) เป็นการย่อพิเศษสำหรับระบุคอลัมน์ทั้งหมดได้ ดังนั้นประโยคคำสั่งนี้มีรูปแบบดังนี้

SELECT * FROM *table_name*

ยกตัวอย่างเช่น SELECT * FROM [Flight Instance] เป็นการเรียกดูคอลัมน์ทั้งหมดจากตาราง Flight Instance ซึ่งได้ผลดังตารางที่ ก-2

Flight number	Date	available seats
TG401	16/3/2005	100
TG402	16/3/2005	50
TG601	17/3/2005	25
TG603	17/3/2005	20

ตารางที่ ก-1 ผลการคิวรี่เมื่อระบุคอลัมน์ที่จะเรียกดูโดยใช้ประโยคคำสั่ง SELECT

Flight number	Date	Plane number	available seats
TG401	16/3/2005	777	100
TG402	16/3/2005	333	50
TG601	17/3/2005	773	25
TG603	17/3/2005	773	20

ตารางที่ ก-2 ผลการคิวรี่เมื่อต้องการเรียกดูคอลัมน์ทั้งหมดโดยใช้ประโยคคำสั่ง SELECT

2. การเรียกดูข้อมูลแบบมีเงื่อนไข เป็นการจำกัดข้อมูลที่จะแสดงออกมาเพื่อให้ได้เฉพาะข้อมูลที่อยู่ในเงื่อนไขที่ต้องการออกมา โดยการใช้งานจะใช้ร่วมกับคำสั่ง WHERE มีรูปแบบประโยคคือ

SELECT *column_name1, column_name2,.....* **FROM** *table_name* **WHERE** *condition*

เมื่อ WHERE *condition* ใช้เป็นเงื่อนไขในการแสดงข้อมูล ยกตัวอย่างเช่น SELECT [Flight number], Date, [available seats] FROM [Flight Instance] WHERE [available seats] > 50 จะคืนค่าในคอลัมน์ Flight number, Date และ available seats จากตาราง Flight Instanceโดยมีเงื่อนไขว่าค่าในคอลัมน์ available seats มีค่ามากกว่า 50 เท่านั้น ผลที่ได้แสดงดังตาราง ก-3

Flight number	Date	available seats
TG401	16/3/2005	100
TG402	16/3/2005	50

ตารางที่ ก-3 ผลการคิวรี่เมื่อเรียกดูข้อมูลแบบมีเงื่อนไขโดยใช้ประโยคคำสั่ง SELECT

2.1 ประโยคย่อย FROM

มักใช้ร่วมกับประโยคคำสั่ง SELECT และ ประโยคคำสั่ง DELETE เราจะใช้ประโยคย่อย FROM เพื่อแสดงถึงการเรียกดูว่ามาจากตารางฐานข้อมูลตัวไหน หรือว่าต้องการจะลบข้อมูลจากตารางไหน ยกตัวอย่างเช่น DELETE FROM Flight Instance WHERE ([Flight number]='TG401')

(DAY([Date]) = 16) AND (MONTH([Date]) =3) AND (YEAR([Date])=2005) AND ([available seats]=50) AND ([Plane number]=777) เป็นการลบแถวในตาราง Flight Instance เมื่อค่า Flight number = TG401, Date = 16/3/2005, Plane number = 777, available seats = 50

2.2 ประโยคย่อย WHERE

ในการใช้ประโยคย่อย WHERE นั้น สามารถระบุกลุ่มย่อยของแถวที่จะถูกคืนค่ากลับมาได้ดังตัวอย่างข้างต้น ดังนั้นรูปแบบของประโยคย่อย WHERE คือ

WHERE <*colume*> <*operator*> <*value*>

จะเห็นว่าประโยคย่อย WHERE จะใช้ร่วมกับโอเปอร์เรเตอร์ (operator) ใน SQL ซึ่งใน SQL Server มีโอเปอร์เรเตอร์ในการเปรียบเทียบทั้งหมดดังแสดงในตารางที่ ก-4

โอเปอร์เรเตอร์	ความหมาย
=	เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าหรือเท่ากับ
<=	น้อยกว่าหรือเท่ากับ
<>	ไม่เท่ากับ

ตารางที่ ก-4 โอเปอร์เรเตอร์ใน SQL

2.3 ประโยคย่อย ORDER BY

เป็นประโยคองค์ประกอบที่เป็นตัวเลือกของประโยคคำสั่ง SELECT ซึ่งยอมให้ระบุลำดับของแถวที่จะถูกเรียกขึ้นมาได้ โดยสามารถระบุคอลัมน์หลายคอลัมน์และเรียกดูแถวจากลำดับมากไปหาน้อยหรือน้อยไปหามากได้ รูปแบบที่ง่ายที่สุดของประโยคย่อย ORDER BY คือใส่ชื่อคอลัมน์ที่จะใช้ในการเรียงลำดับแถวที่จะถูกเรียกดูจากคิวรี่เพียงคอลัมน์เดียว ยกตัวอย่างเช่น SELECT * FROM [Flight schedule] ORDER BY [Depart time] เป็นการคืนค่าทุกคอลัมน์จากตาราง Flight schedule โดยแสดงผลเรียงลำดับตามค่าเวลาในคอลัมน์ Depart time น้อยไปมาก ผลที่ได้ดังตารางที่ ก-5

Flight number	From airportcode	To airportcode	Depart time	Arrival time
TG402	SIN	BKK	08.20	09:40
TG401	BKK	SIN	19:15	22:35
TG409	BKK	SIN	15:15	18:35

ตารางที่ ก-5 ผลการคิวรี่เมื่อต้องการเรียกดูคอลัมน์โดยใช้ร่วมกับประโยคย่อย ORDER BY

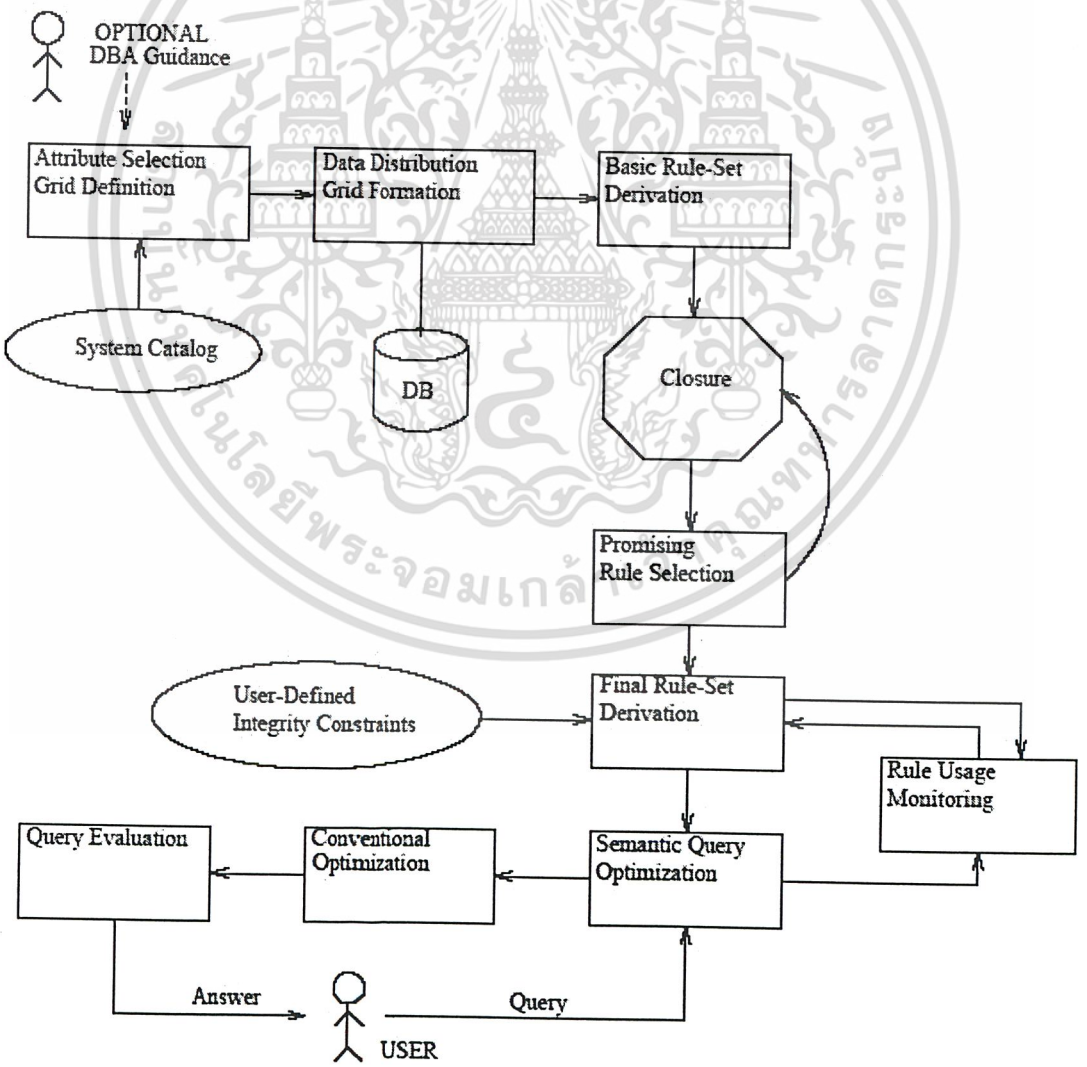
ภาคผนวก ข

การเรียนรู้กฎการแปลงสำหรับ Semantic Query Optimization ด้วยวิธี Data-Driven

Semantic Query Optimization ใช้กฎสำหรับแปลงรูปคำสั่งคิวรี่ เช่น กฎข้อบังคับเชิงความหมาย (semantic integrity constraints) ใน query optimization โดย คำสั่งคิวรี่ของผู้ใช้จะถูกแปลงให้อยู่ในรูปคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกัน(Semantically Equivalent Queries) ในรูปแบบต่างๆ ซึ่งให้ผลลัพธ์เหมือนกับคำสั่งคิวรี่ของผู้ใช้แต่มีประสิทธิภาพในการประมวลผลมากกว่า

ความสำเร็จของ Semantic Query Optimization ขึ้นอยู่กับประสิทธิภาพของกฎที่ใช้ในการแปลงรูปคำสั่งคิวรี่ ที่ซึ่งสามารถลดเวลาที่ใช้ในการประมวลผลของคำสั่งคิวรี่ที่มีขนาดใหญ่ โดยเซ็ตของ กฎที่ใช้ในการแปลงรูปคำสั่งคิวรี่ จะรวมถึงกฎข้อบังคับที่ผู้ใช้เป็นคนกำหนดเองด้วย

ในเรื่องนี้ เราจะใช้วิธี data-driven เพื่อเรียนรู้กฎที่ใช้ในการแปลงรูปคำสั่งคิวรี่ซึ่ง วิธีนี้จะเป็นประโยชน์สำหรับ Semantic Query Optimization โดยเราจะใช้ตารางแสดงการกระจายของข้อมูลที่ซึ่งง่ายต่อการเรียนรู้กฎที่ใช้ในการแปลงรูปคำสั่งคิวรี่



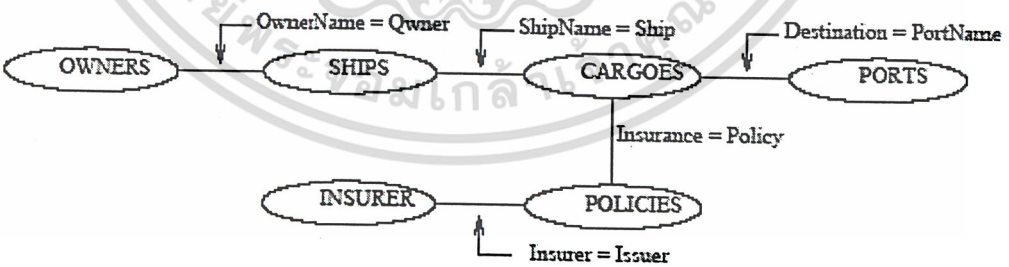
รูปที่ ข-1 แสดงถึงภาพรวมของวิธี data-driven

เริ่มต้นเซ็ตของแอททริบิวจะถูกเลือกสำหรับการสร้างตารางการกระจายตัวของข้อมูลโดยแอททริบิวเหล่านั้น อาจจะถูกกำหนดโดยผู้ดูแลระบบ หรือ ถูกเลือกโดยอัตโนมัติ และ เซ็ต ของกฎพื้นฐานจะถูกดึงมาจากตารางและส่งไปให้ closure สำหรับการสร้างกฎเพิ่ม โดยผลลัพธ์ จะได้เซ็ตของกฎที่ถูกคัดเลือกโดย rule selection ซึ่งเป็นกระบวนการคัดเลือกกฎ และ เซ็ตของ กฎที่ถูกคัดเลือกจะถูกใช้โดย semantic query optimizer เพื่อแปลงคำสั่งคิวรี่ของผู้ใช้ให้อยู่ในรูปของคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกัน ที่ซึ่งใช้เวลาในการประมวลคำสั่งที่น้อยกว่าคำสั่งคิวรี่ก่อนที่จะถูกแปลงรูป

Relation	Attributes	Attribute for Primary Index (Clustered)	Attributes for Secondary Indices (Unclustered)	Relation Size (in tuples)
SHIPS	ShipName, Owner, ShipType, Draft, DeadWt, Capacity, Registry	ShipName	ShipType, Owner, DeadWt	20,000
PORTS	PortName, Country, Depth FacilityType	PortName	---	1,000
CARGOES	Ship, Destination, Shipper CargoType, Quantity, DollarValue, Insurance	Ship	Destination, Insurance	25,000
OWNERS	OwnerName, Location, Assets, Business	OwnerName	Business	1,000
POLICIES	Policy, Issuer, Coverage	Policy	Issuer	25,000
INSURERS	Insurer, InsCountry, Capitalization	Insurer	---	500

ตารางที่ ข-1 : ฐานข้อมูลของการเรือ(shipping database)

ซึ่งตารางนี้จะใช้อ้างอิงเนื้อหาในเรื่องนี้



รูปที่ ข-2 แสดงถึงความสัมพันธ์ของการเชื่อมต่อกันของตาราง

การเรียนรู้วิธีของการกระจายตัวของข้อมูล (Data Distribution-Based learning)

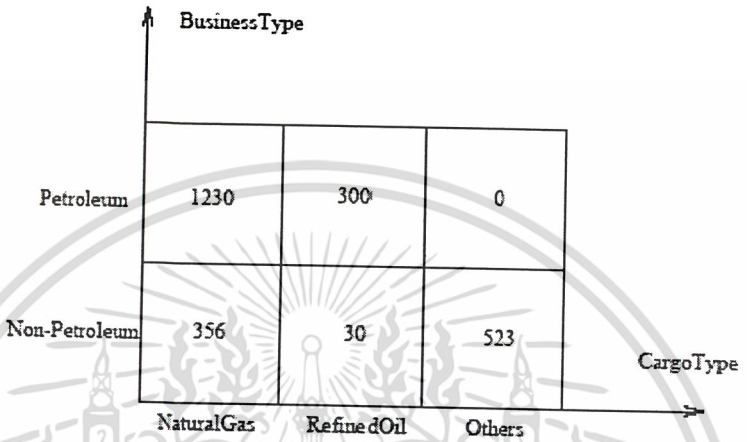
การกระจายตัวของข้อมูล คือ ฟังก์ชันที่จับคู่ระหว่างแอททริบิว ซึ่งในแต่ละเซลล์ของตารางจะแสดงถึงจำนวนแถวที่สอดคล้องกับเงื่อนไข

ตารางจะถูกกำหนดโดย เซ็ตของจุดโคออดิเนท ซึ่งเป็น แอททริบิว ใน database

ตัวอย่าง พิจารณารูปที่ 3 กับเซ็ตของจุดโคออดิเนท คือ BusinessType และ CargoType โดย

Range-sets สำหรับโดออดิเนทแรก อาจจะเป็น (Petroleum ,Non-Petroleum) ส่วนRange-sets สำหรับโดออดิเนทที่สอง อาจจะเป็น (NaturalGas , RefinedOil , Others)

ส่วนเซลล์ ของตารางนี้คือ (Petroleum ,NaturalGas),(Petroleum , RefinedOil), (Petroleum, Others), (Non-Petroleum, NaturalGas) ,(Non-Petroleum ,RefinedOil), (Non-Petroleum, Others)



รูปที่ ข-3 แสดงถึงตารางที่มีแอททริบิว BusinessType และ CargoType เป็นโดออดิเนท

การสร้างกฎใหม่โดย Closure

ในหัวข้อนี้ จะแสดงถึงการสร้างกฎใหม่จากกฎที่มีอยู่

สมมติว่าเรามีกฎอยู่ 2 กฎ คือ

IC1: A1 ⟶ C1

IC2: A2 ⟶ C2

กำหนดให้ช่วงของค่าเป็นดังนี้ $R100 = (0..100)$, $R150 = (100..150)$, $R200 = (150..200)$, $R250 = (200..250)$, $R300 = (250..300)$, $R350 = (300..350)$, $R\infty = (350.. \infty)$

เทคนิคที่1 : จาก IC1 และ IC2 ถ้า C1 ⟶A2 คือ เงื่อนไขข้อบังคับที่มีอยู่แล้ว สามารถสรุป A1 ⟶C2 เข้าไปยัง เซ็ตของกฎได้

ตัวอย่าง : ให้ กฎเป็นดังนี้

(DeadWt $\in R100$) ⟶ (ShipType = 'Barge') ,

(ShipType = 'Barge') ⟶ (CargoType $\in$ { 'Grain', 'Metal'}), and

(CargoType $\in$ { 'Grain', 'Metal'}) ⟶ (DollarValue < 5000,000),

เราสามารถเพิ่มสรุปข้างล่างเข้าไปยังเซ็ตของกฎได้

(DeadWt $\in R100$) ⟶ (DollarValue < 5000,000)

เทคนิคที่2 : จาก IC1 และ IC2 ถ้า A1 และ A2 อยู่บน แอททริบิว เดียวกัน และ C1 และ C2 ก็อยู่บน แอททริบิว เดียวกัน แล้ว สามารถเพิ่ม

Union(A1,A2) ⟶ Union(C1,C2) เข้าไปในเซ็ตของกฎได้

ตัวอย่าง : ให้ กฎเป็นดังนี้

(DeadWt∈*R*200) ⟶ (ShipType = 'SuperTanker'),

(DeadWt ∈*R*250) ⟶ (ShipType = 'SuperTanker'),

(DeadWt∈ *R*300) ⟶ (ShipType = 'SuperTanker'),

(DeadWt ∈*R*∞)⟶ (ShipType = 'SuperTanker'),

เราสามารถเพิ่ม rule ข้างล่างเข้าไปยังเซ็ตของกฎได้

(DeadWt >= 200) ⟶ (ShipType = 'SuperTanker')

เทคนิคที่3 จาก IC1 และ IC2 สามารถเพิ่ม

Intersection(A1,A2) ⟶ Intersection(C1,C2)

ตัวอย่าง: ให้ กฎเป็นดังนี้

(DollarValue>3000)⟶ (Issuer∈{ 'Lloyds', 'Issuerx', 'Issuery'}),

(ShipType= 'SuperTanker') ⟶ (Issuer ∈{ 'Lloyds'})

เราสามารถเพิ่ม rule ข้างล่างเข้าไปยังเซ็ตของกฎได้

(DollarValue > 3000) AND (ShipType= 'SuperTanker') ⟶ (Issuer ∈{'Lloyds'})

เทคนิคที่4 จาก IC1 และ IC2 เพิ่ม B ⟶ D ไปยังเซ็ตของกฎ เมื่อ

B = Not(C1) และ D = Not(A1)

ตัวอย่าง :ให้ กฎเป็นดังนี้

(DeadWt > 150) ⟶ (ShipType = 'SuperTanker')

เราสามารถเพิ่ม rule ข้างล่างเข้าไปยังเซ็ตของกฎได้

Not(ShipType = 'SuperTanker')⟶ Not(DeadWt > 150)

สมมติ โดเมนของ แอททริบิว ShipType คือ {'Barge' , 'Tanker', 'Supertaker'}

Not(ShipType = 'SuperTanker') = (ShipType ∈{'Barge', 'Tanker'})

และ Not(DeadWt > 150) = (DeadWt $\leq$ 150)

ดังนั้น จะได้กฎใหม่ ดังนี้

(ShipType∈ {'Barge', 'Tanker'})⟶ (DeadWt $\leq$ 150).

Rule #	Antecedent	Implication	Consequent
R_1	*(DeadWt > 350)*	->	*(FacilityType = 'OffShore')*
R_2	*(DeadWt > 300)*	->	*(Business = 'Leasing')*
R_3	--	--	*(Coverage ≤ DollarValue)*
R_4	--	--	*(Quantity ≤ Capacity)*
R_5	*(CargoType ε {'Natural Gas', 'Refined'})* and *(DollarValue > 500)*	->	*(Facilitytype = 'General')*
R_6	*(DeadWt > 150)*	->	*(ShipType = 'SuperTanker')*
R_7	*(DollarValue > 3000) and (ShipType = 'SuperTanker')*	->	*(Issuer = 'Lloyds')*
R_8	*(Business = 'Petroleum')*	->	*(CargoType ε {'Natural Gas', 'Refined', 'Oil'})*

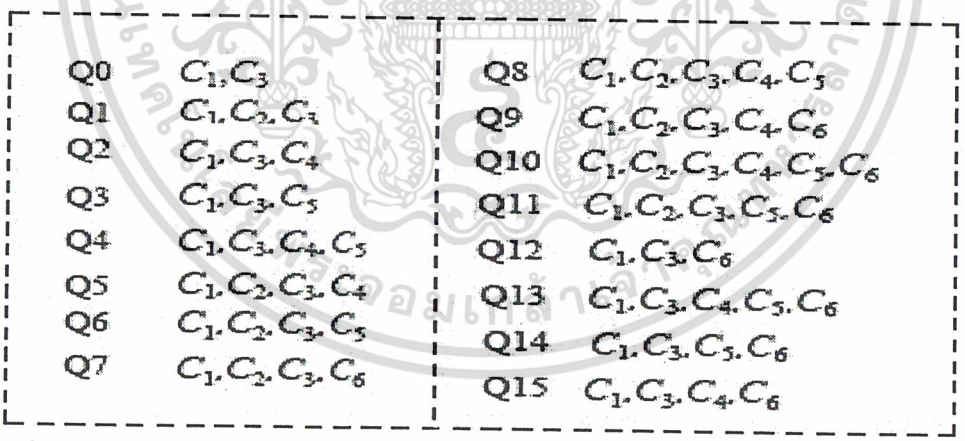
ตารางที่ ข-2 แสดงถึงกฎที่ถูกใช้ในการแปลงคำสั่งคิวรี่โดยอ้างอิงจากฐานข้อมูลของการเรือ

การใช้กฎข้อบังคับสำหรับการแปลง query ใน Semantic Query Optimization

พิจารณาคำสั่งคิวรี่ของผู้ใช้ นี้

Qo : (DeadWt > 400) and (DollarValue > 4000) : (?Destination)

ขั้นแรก คือ สร้างพื้นที่สำหรับเก็บคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกันในรูปแบบต่างๆ ซึ่งถูกทำโดยการประยุกต์กฎ R1 ถึง R8 กับ Qo ซึ่งอันที่จริงแล้ว มีเพียงกฎ R1,R2,R6 และ R7 เท่านั้นที่สามารถเปลี่ยนคำสั่งคิวรี่ของผู้ใช้ไปเป็นคำสั่งคิวรี่ Q1 ถึง Q15 ซึ่งเป็นคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกับคำสั่งคิวรี่ของผู้ใช้ที่ก่อนถูกแปลงรูป



รูปที่ ข-4 แสดงถึงการสร้างคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกับคำสั่งคิวรี่ของผู้ใช้ที่ก่อนถูกแปลงรูป

จาก รูปที่ 4 แสดงถึง การสร้างคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกับคำสั่งคิวรี่ของผู้ใช้ที่ก่อนถูกแปลงรูปซึ่งจะได้ทั้งหมด 15 คำสั่งคิวรี่โดยเงื่อนไข C1 ถึง C6 เป็นดังนี้

C1: (DeadWt > 400)

C2: (ShipType = 'SuperTanker')

C3: (DollarValue > 4000)

C4: (FacilityType = 'OffShore')

C5: (Business = 'Leasing')

C6: (Insurer = 'Lloyds')

ซึ่ง C1 – C6 เอามาจากกฎ R1 , R2 , R6 , R7

โดย การสร้างคำสั่งคิวรี่ที่ให้ผลลัพธ์เหมือนกับคำสั่งคิวรี่ของผู้ใช้ที่ก่อนถูกแปลงรูปใน รูปที่4 อาศัยกฎพื้นฐาน ที่เรียกว่า การเพิ่มเงื่อนไขเข้าไปในคำสั่งคิวรี่ของผู้ใช้(Clause Introduction(CI)) และ การกำจัดเงื่อนไขในคำสั่งคิวรี่ของผู้ใช้(Clause Elimination (CE))

สมมติว่า A,B เป็นเงื่อนไขที่แตกต่างกัน และ (A⟶ B) เป็น กฎ

(CI): (A) and (A⟶ B) ≡ (A) and (B) and (A⟶ B)

(CE): (A) and (B) and (A⟶ B) ≡ (A) and (A ⟶ B)

แต่ละคำสั่งคิวรี่ที่ถูกแปลงรูป Q0 จนถึง Q15 จะถูก optimizer สร้างแผนคำสั่งคิวรี่(query plan) และประเมินค่า cost ของแต่ละแผนคำสั่งคิวรี่โดยดูจำนวนของคำสั่งของ CPU และ page fetches

Query#	0	1	2	3	4	5	6	7
Total Cost	605.7	157.6	605.7	127.8	127.8	157.6	127.0	157.6
Query#	8	9	10	11	12	3	14	15
Total Cost	127.0	157.6	127.0	127.0	227.6	127.8	127.8	227.6

ตารางที่ ข-3 แสดงถึงค่า cost ของแต่ละแผนคำสั่งคิวรี่

ข้อมูลที่แสดงใน ตารางที่ 3 พบว่าแผนคำสั่งคิวรี่ของผู้ใช้ที่ก่อนถูกแปลง(Q0) ที่ปราศจาก Semantic Query Optimization มีค่า cost เป็น 605 units แต่ Semantic Query Optimization สามารถให้แผนคำสั่งคิวรี่กับค่า cost ในการประมวลผลที่น้อยกว่า

ภาคผนวก ค

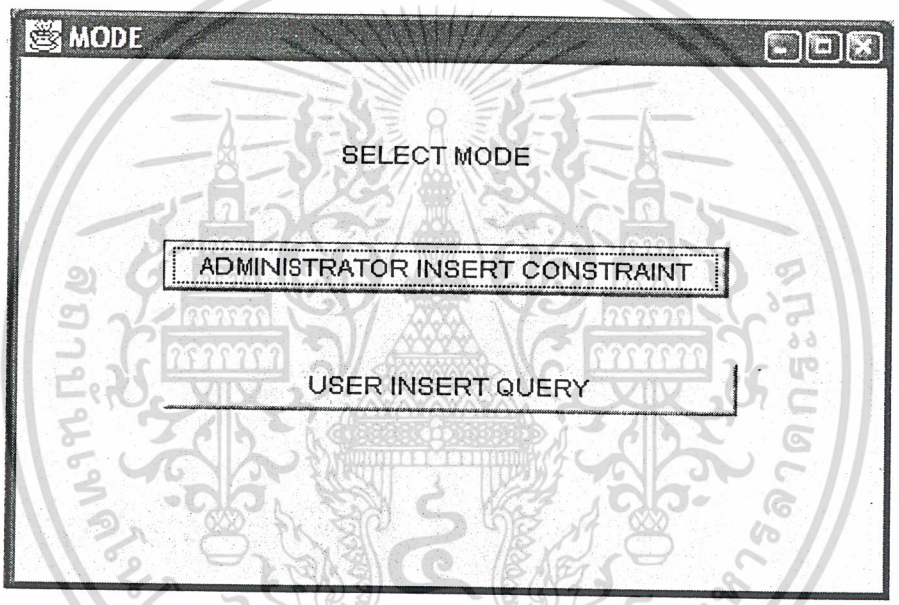
คู่มือการใช้งาน

ผู้ใช้งานระบบจะมีอยู่ 2 ประเภท คือ

1. Administrator หรือ ผู้ดูแลระบบ
2. ผู้ใช้งานทั่วไป

โดยมีโหมดการทำงานให้ผู้ใช้เลือกอยู่ 2 โหมดด้วยกัน คือ

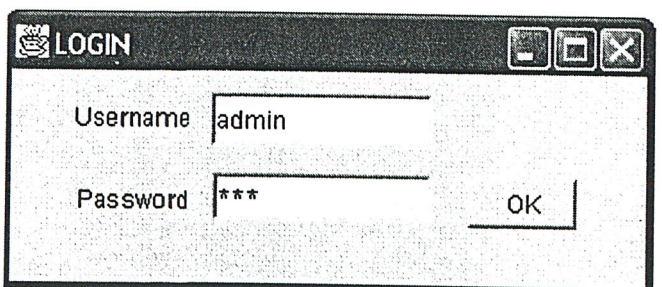
1. insert constraint สำหรับผู้ดูแลระบบ เพื่อใส่กฎข้อบังคับต่างๆ
2. insert query สำหรับ ผู้ใช้งานทั่วไป เพื่อหาคำตอบที่ผู้ใช้ต้องการ



รูปที่ ค-1 หน้าจอของการเลือกโหมดการทำงาน

ในกรณีที่ผู้ดูแลระบบเลือกโหมด insert constraint

จะมีหน้าจอให้ผู้ดูแลระบบใส่ชื่อและรหัสผ่านเพื่อพิสูจน์ว่าตนเองเป็นผู้ดูแลระบบ โดยผู้ใช้ทั่วไปจะไม่สามารถกำหนดเงื่อนไขข้อบังคับต่างๆได้ (โดยชื่อและรหัสผ่านจะอยู่ในฐานข้อมูลที่ชื่อว่า SQODBC ในตาราง LOGIN)



รูป ค-2 หน้าจอสำหรับใส่ชื่อและรหัสผ่านสำหรับผู้ดูแลระบบ

หลังจากนั้น จะมีหน้าจอให้เลือกประเภทของกฎข้อบังคับ ซึ่งมีอยู่ด้วยกัน 3 ประเภท คือ

1. Attribute Constraint

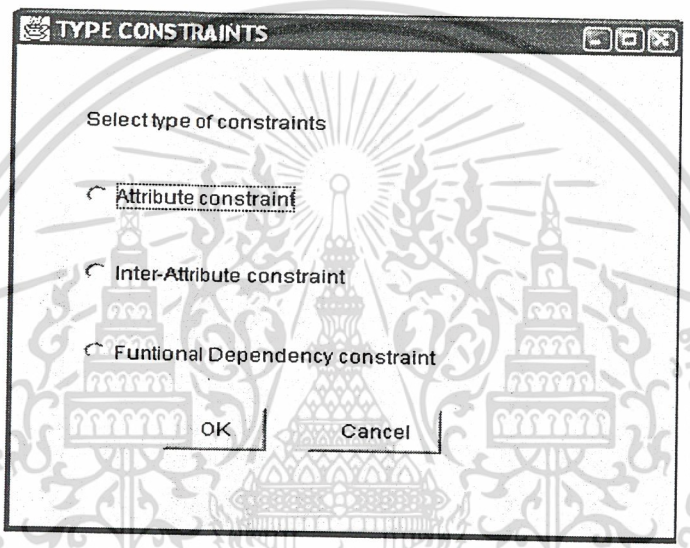
   เป็นเงื่อนไขสำหรับกำหนดขอบเขตของแอททริบิว

2. Inter-Attribute Constraint

   เป็นเงื่อนไขที่เกี่ยวกับความสัมพันธ์ระหว่างแอททริบิว

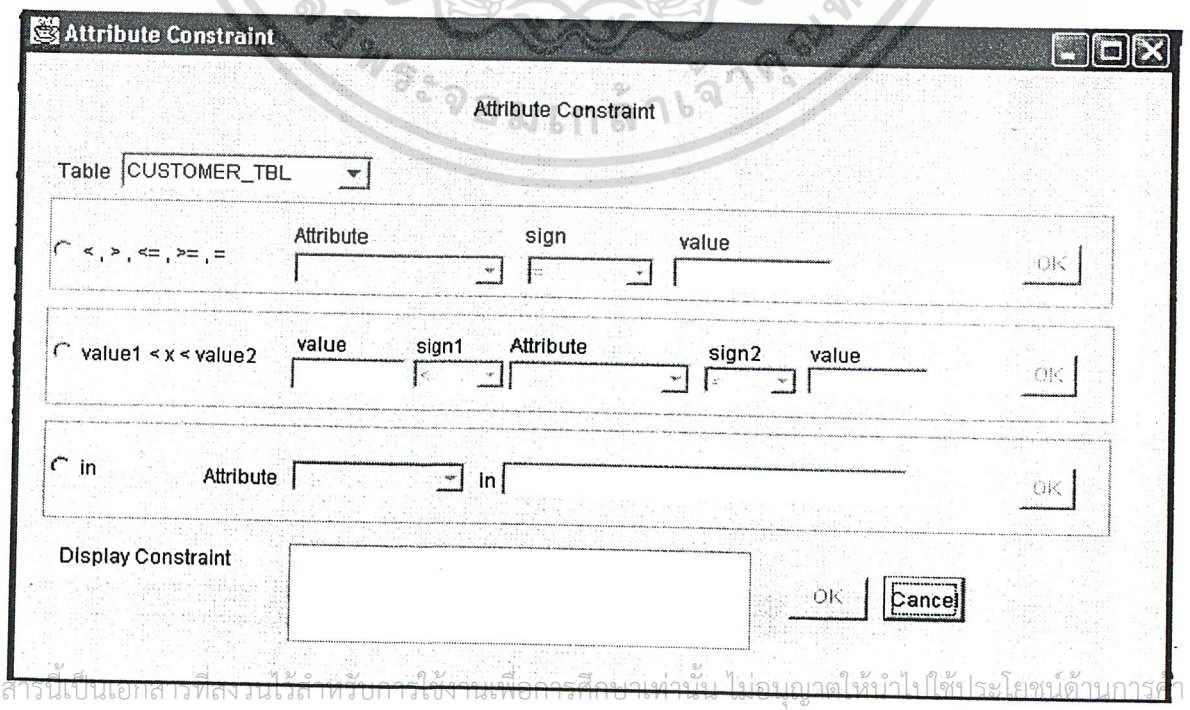
3. Functional Dependency Constraint

   เป็นเงื่อนไขที่เกี่ยวกับความสัมพันธ์ที่ขึ้นต่อกันของสองแอททริบิว



รูปที่ ค-3 ประเภทของ Constraint

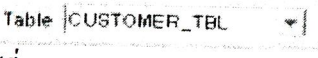
กรณีที่ผู้ดูแลระบบเลือก Attribute Constraint



รูปที่ ค-4 หน้าจอสำหรับใส่ขอบเขตเงื่อนไขของแอททริบิว

จากรูป

ขั้นแรก คือ เลือกตารางที่มีแอททริบิวที่ต้องการ



รูปที่ ค-5 แสดงตารางของแอททริบิว

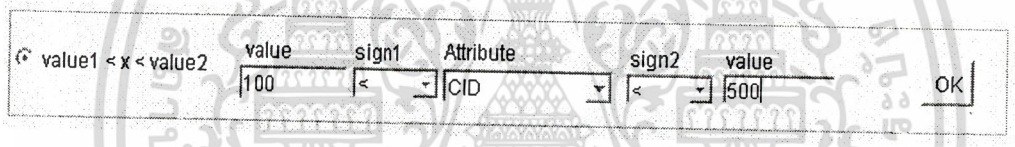
โดยการใส่ขอบเขตเงื่อนไขของแอททริบิว จะมีอยู่ 3 รูปแบบด้วยกัน คือ

1. กำหนดขอบเขตโดยใช้ เครื่องหมาย $=, \neq, >, <, \geq, \leq$

รูปที่ ค-6 แสดงเงื่อนไขขอบเขตของแอททริบิว

เช่น จากรูป เป็นการกำหนดเงื่อนไข CID > 23523

2. กำหนดขอบเขตของแอททริบิวเป็นช่วงของค่าที่กำหนด



รูปที่ ค-7 แสดงเงื่อนไขขอบเขตของแอททริบิวเป็นช่วง

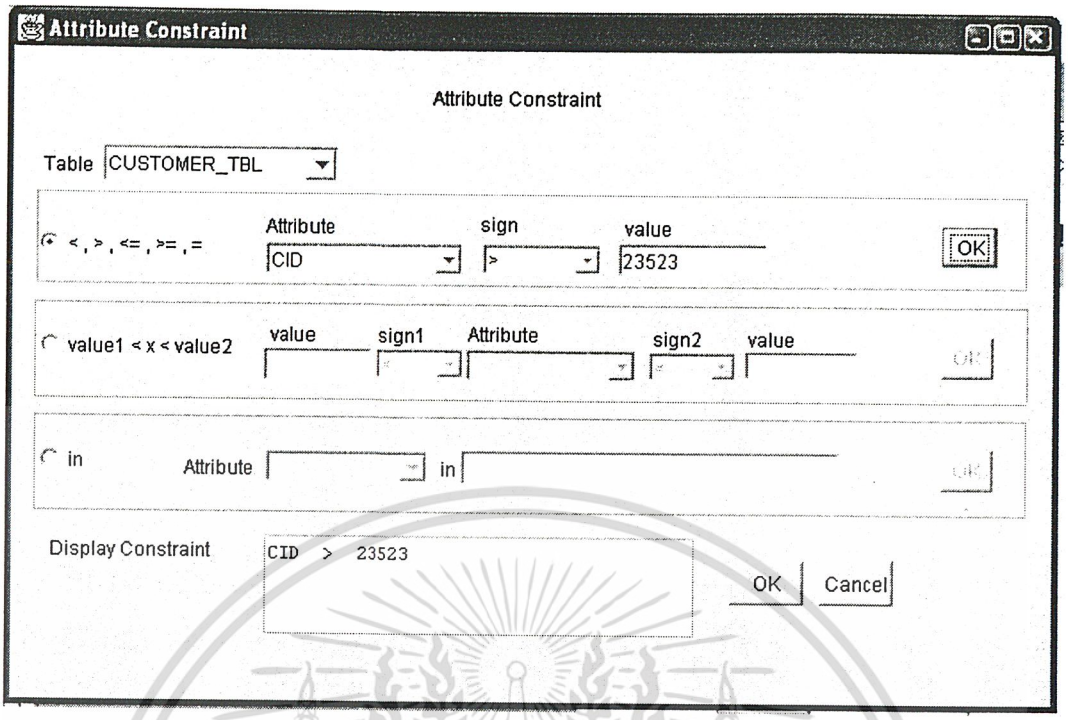
เช่น จากรูป เป็นการกำหนดเงื่อนไข 100 < CID < 500

3. กำหนดให้ขอบเขตของแอททริบิวเป็นสมาชิกภายในค่าที่กำหนด

รูปที่ ค-8 แสดงเงื่อนไขขอบเขตของแอททริบิวในลักษณะสมาชิก

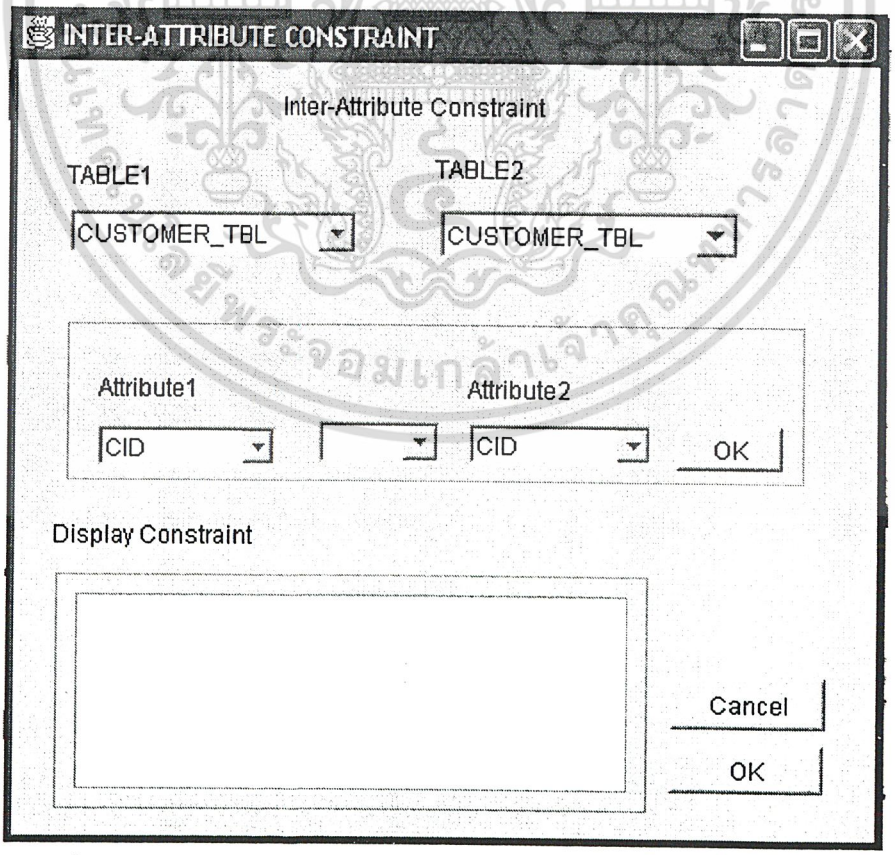
เช่น จากรูป เป็นการกำหนดเงื่อนไขให้ CID เป็นสมาชิกของ set {100,200,300}

หลังจากกำหนดขอบเขตของแอททริบิวเสร็จ กดปุ่ม OK แล้วเงื่อนไขดังกล่าว ก็จะปรากฏอยู่ที่ช่อง Display Constraint ดังรูป ค-9 หลังจากนั้น ถ้าผู้ดูแลระบบกดปุ่ม OK ก็จะเก็บกฎข้อบังคับนั้นลงฐานข้อมูล แต่ถ้าผู้ดูแลระบบกดปุ่ม Cancel ก็จะลบกฎข้อบังคับนั้น



รูปที่ ค-9 แสดงเงื่อนไขสำหรับกำหนดขอบเขตของแอททริบิว

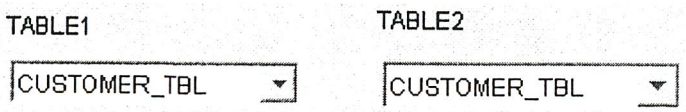
กรณีที่ผู้ดูแลระบบเลือก Inter-Attribute Constraint



รูปที่ ค-10 หน้าจอสำหรับใส่เงื่อนไขที่เกี่ยวกับความสัมพันธ์ระหว่างแอททริบิว

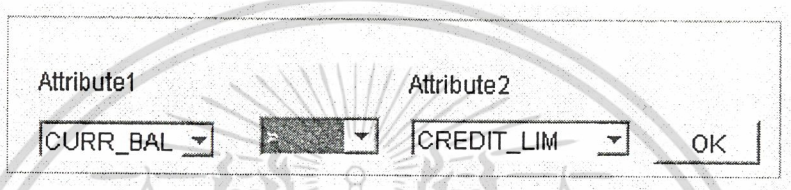
จากรูป

ขั้นแรก คือ เลือกตารางสำหรับแอททริบิวที่ต้องการจะเปรียบเทียบความสัมพันธ์



รูปที่ ค-11 แสดงตารางของแอททริบิว

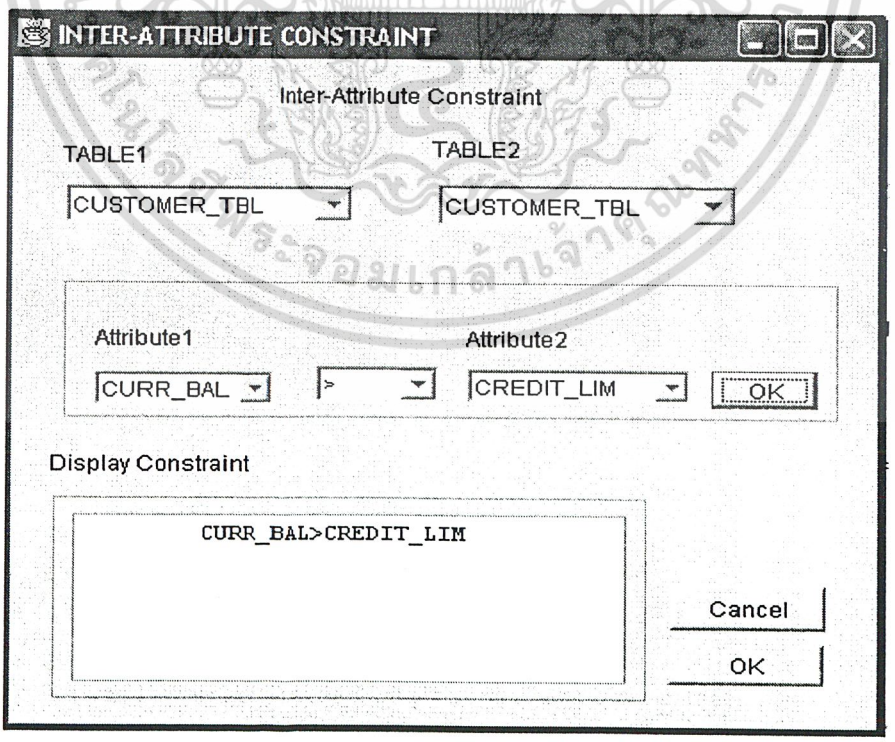
หลังจากนั้น เลือกแอททริบิวและ เครื่องหมาย ที่ต้องการแสดงความสัมพันธ์



รูปที่ ค-12 แสดงเงื่อนไขความสัมพันธ์ระหว่างแอททริบิว

เช่น จากรูป เป็นการกำหนดเงื่อนไขว่า CURR_BAL > CREDIT_LIM ซึ่งจะต่างจาก attribute constraint ตรงที่ เป็นการเปรียบเทียบกันระหว่างแอททริบิว ไม่ใช่เปรียบเทียบกับ value

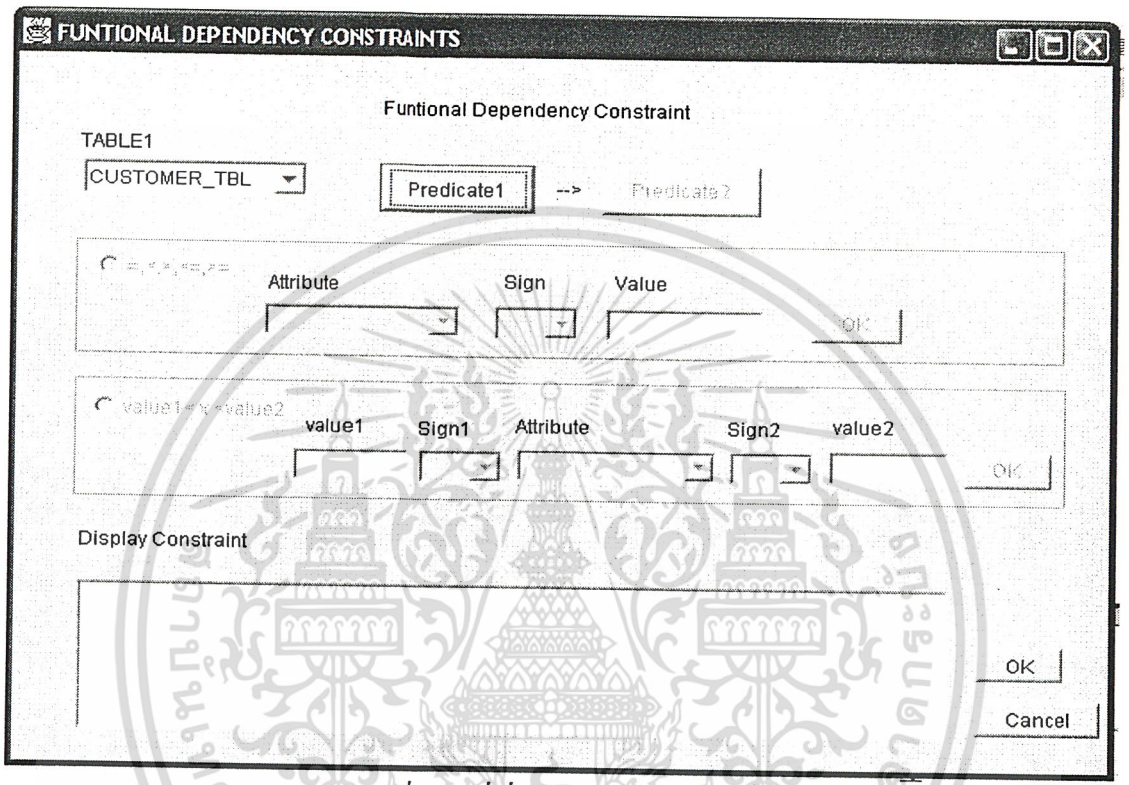
หลังจากกำหนดเงื่อนไขเสร็จ กดปุ่ม OK แล้วเงื่อนไขดังกล่าว ก็จะปรากฏอยู่ที่ช่อง Display Constraint ดังรูป



รูปที่ ค-13 แสดงเงื่อนไขที่เกี่ยวกับความสัมพันธ์ระหว่างแอททริบิว

หลังจากนั้น ถ้าผู้ดูแลระบบกดปุ่ม OK ก็จะเก็บกฎข้อบังคับนั้นลงฐานข้อมูล แต่ถ้าผู้ดูแลระบบกดปุ่ม Cancel ก็จะลบกฎข้อบังคับนั้น

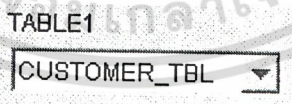
กรณีที่ผู้ดูแลระบบเลือก Functional Dependency Constraint



รูป ค-14 หน้าจอสำหรับใส่เงื่อนไขที่เกี่ยวกับความสัมพันธ์ที่ขึ้นต่อกันของ แอททริบิว

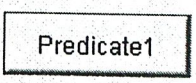
จากรูป

ขั้นแรก คือ เลือกตารางสำหรับแอททริบิว ตัวแรก



รูป ค-15 แสดงตารางของเพรดดิเคตแรก

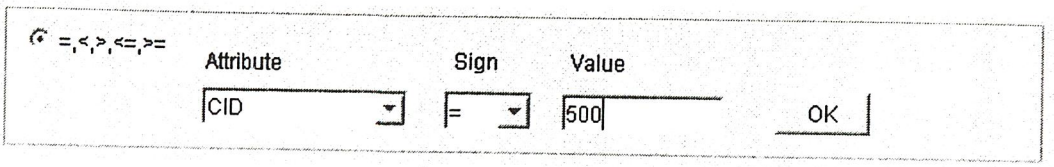
หลังจากนั้นกดปุ่ม predicate1 เพื่อใส่เงื่อนไขให้กับแอททริบิวตัวแรก



รูป ค-16 แสดงปุ่มกดเพรดดิเคตแรก

โดยเงื่อนไขที่จะกำหนดให้กับแอททริบิว จะมีอยู่ 2 ประเภท คือ

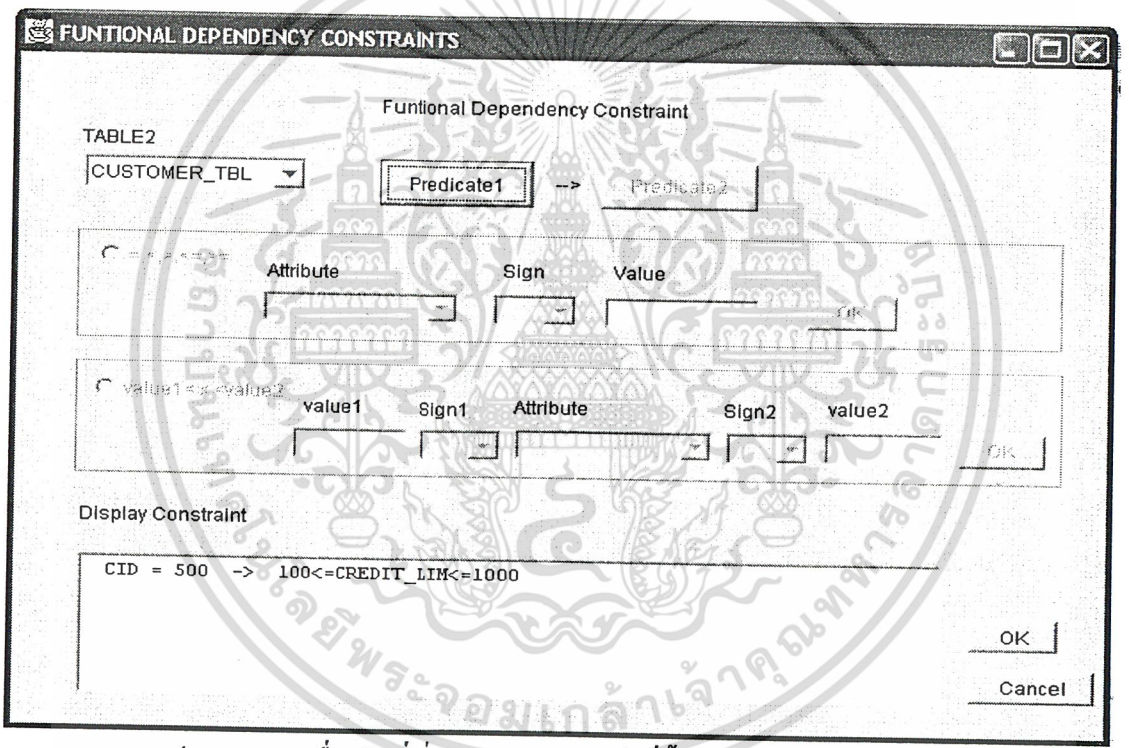
1. กำหนดขอบเขตของแอททริบิว โดยใช้ เครื่องหมาย $=, \neq, >, <, \geq, \leq$
2. กำหนดขอบเขตของแอททริบิว เป็นช่วงของค่าที่กำหนด



รูป ค-17 แสดงเงื่อนไขกำหนดขอบเขตของแอททริบิว

จากรูป เป็นการกำหนดเงื่อนไข CID = 500 เมื่อกดปุ่ม OK แล้ว ปุ่ม predicate2 จะ enable ส่วนปุ่ม predicate1 จะ disable ไป

หลังจากนั้น ทำการเลือกตารางสำหรับแอททริบิว ตัวที่สอง และกำหนดเงื่อนไขให้กับแอททริบิวที่สองในลักษณะเดียวกับแอททริบิวตัวแรก แล้วกดปุ่ม OK เงื่อนไขที่กำหนดก็จะปรากฏในช่อง Display Constraint ดังรูป



รูป ค-18 แสดงเงื่อนไขที่เกี่ยวกับความสัมพันธ์ที่ขึ้นต่อกันของสองแอททริบิว

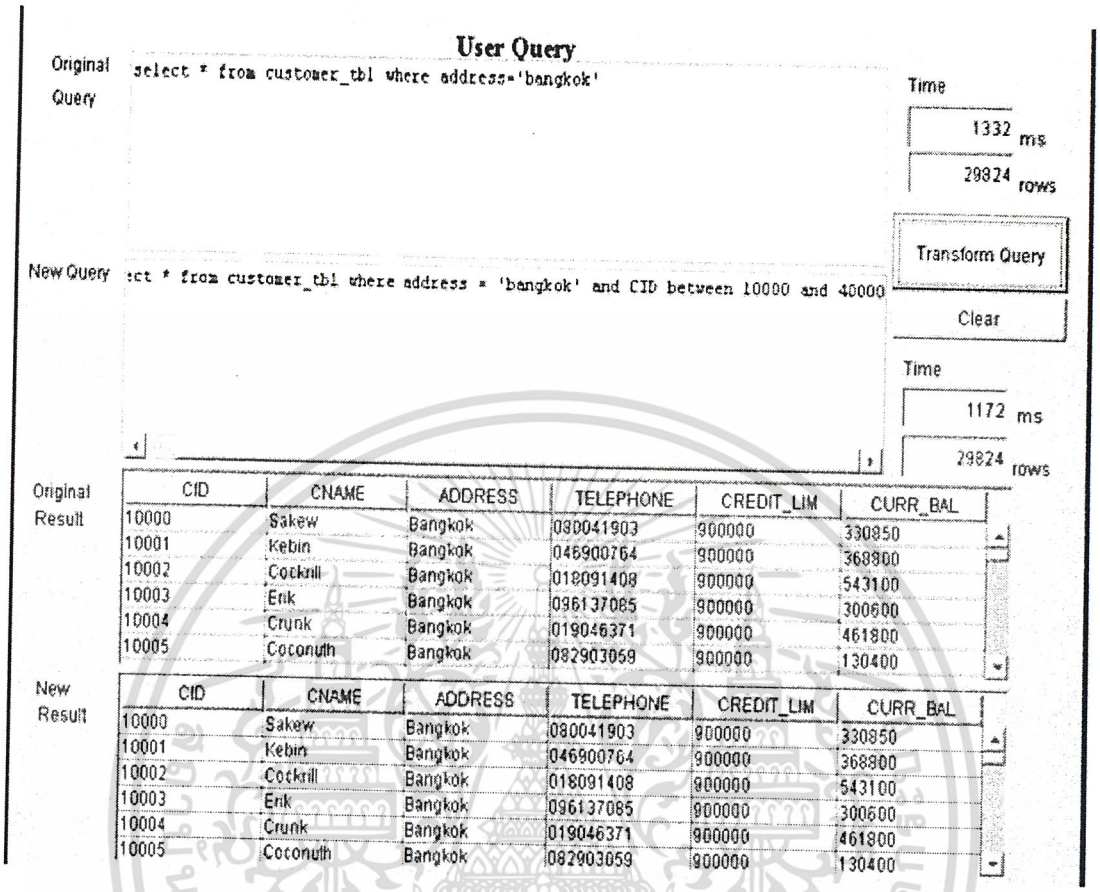
จากรูปเป็นการกำหนดเงื่อนไขว่า

$$CID = 500 \rightarrow 100 \leq CREDIT\_LIM \leq 1000$$

หมายความว่า ถ้า CID มีค่าเท่ากับ 500 แล้ว CREDIT_LIM จะมีค่ามากกว่าหรือเท่ากับ 100 แต่มีค่าไม่เกิน 1000

หลังจากนั้น ถ้าผู้ดูแลระบบกดปุ่ม OK ก็จะเก็บกฎข้อบังคับนั้นลงฐานข้อมูล แต่ถ้าผู้ดูแลระบบกดปุ่ม Cancel ก็จะลบกฎข้อบังคับนั้น

ในกรณีที่ผู้ใช้เลือกโหมด insert query



รูป ค-19 หน้าจอสำหรับให้ผู้ใช้ใส่คำสั่งคิวรี่

จะมีหน้าจอให้ผู้ใช้ใส่คำสั่งคิวรี่โดยใช้ภาษา SQL ลงในช่อง Original Query หลังจากนั้น เมื่อผู้ใช้กดปุ่ม Transform Query ระบบก็จะทำการแปลงคำสั่งคิวรี่ของผู้ใช้ และแสดงคำสั่งที่ถูกแปลงแล้วในช่อง New Query และทำการส่งคำสั่งทั้งสองไปหาคำตอบ แล้วแสดงผลลัพธ์ในช่อง Result โดยผลลัพธ์ของคำสั่งคิวรี่ของผู้ใช้จะแสดงในช่อง Original Result ส่วนผลลัพธ์ของคำสั่งคิวรี่ที่ถูกแปลงรูปแล้วจะแสดงในช่อง New Result พร้อมทั้งแสดงเวลาที่ใช้และจำนวนแถวของผลลัพธ์ออกมาด้วย

หมายเหตุ : เมื่อมีการเชื่อมตาราง (join) จะต้องใส่ label ของตารางนั้นๆ ไว้ข้างหน้าของแอททริบิวทุกๆ ตัว เช่น select * from customer_tbl c , order_tbl o

where c.cid = o.cid and c.address = 'bangkok' and o.discount > 10

ในกรณีนี้ เป็นการเชื่อมตารางระหว่างตาราง customer กับตาราง order สังเกตว่าจะต้องใส่ label c และ o ไว้ข้างหน้าแอททริบิวทุกๆ แอททริบิวที่มีอยู่ในเงื่อนไข

บรรณานุกรม

[1] Chun-Nan Hsu and Craig A Knoblock, Learning Database Abstractions For Query Reformulation*. In Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, 1993

[2] Qi Cheng, Jarek Gryz, Fred Koo, Cliff Leung, Linqi Liu, Xiaoyan Qian, and Berni Schiefer, Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database. In Proceedings of the 25th VLDB, Edinburgh, Scotland, September 1999

[3] Jarek Gryz, Linqi Liu, Xiaoyan Qian, Semantic Query Optimization Techniques in IBM DB2: Initial Results. Technical Report CS-1999-01, York University, February 19, 1999

[4] Jerome Robinson, Barry G. T. Lowden, Semantic Query Optimisation and Rule Graphs. KRDB 1998: 14.1-14.10

[5] Shekhar S., Hamidzadeh B., Kohli A., and Coyle M, Learning transformation rules for semantic query optimization: a data-driven approach. IEEE Transactions on Knowledge and Data Engineering 5, 6 (December 1993), 950--964.

[6] 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.

[7] Koppelaars , Business Rules, Classification & Implementation. EOUG'94 Conference proceedings, Volume 3, paper 169, 1,1994.

[8] P. Godfrey, J. Grant, J. Gryz, J. Minker, Integrity Constraints: Semantics and Applications (1998), Logics for Databases and Information Systems,1998

[9] 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.

[10] S.T. Shenoy and Z.M. Ozsoyoglu, Design and implementation of semantic query optimiser, IEEE Transactions on Knowledge and Data Eng., 1(3), 1989, 344-361.

[11] วรัชญ์ กิจชระภูมิ, การใช้งาน Microsoft SQL Server2000 Step by Step, สำนักพิมพ์สามย่าน.COM, 2544

[12] ThomasM. Comnolly, Database system: a practical approach to design, implementation and management, Carolyn E.Begg.-3rd ed.

[13] ผศ.ดร. วรรณวิภา ติตถะสิริ , คู่มือเรียน SQL ด้วยตนเอง, บริษัท โปรวิชั่น จำกัด, 2544