การสร้างกฎข้อบังคับของฐานข้อมูลโดยการทำเหมืองข้อมูล

นางสาวศิริกาญจนา พิลาบุตร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปีการศึกษา 2551

DATABASE TRIGGER CREATION

WITH DATA MINING

Sirikanjana Pilabutr

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2008

การสร้างกฎข้อบังคับของฐานข้อมูลโดยการทำเหมืองข้อมูล

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(รศ. ดร.นิตยา เกิดประสพ)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
กรรมการ

_____

(อ. ดร.ปรเมศวร์ ห่อแก้ว)
กรรมการ

_____

(ศ. ดร.ไพโรจน์ สัตยธรรม)
รองอธิการบดีฝ่ายวิชาการ

_____

(รศ. น.อ. ดร.วรพจน์ ขำพิศ)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ศิริกาญจนา พิลาบุตร : การสร้างกฎข้อบังคับของฐานข้อมูลโดยการทำเหมืองข้อมูล
(DATABASE TRIGGER CREATION WITH DATA MINING)
อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร.นิตยา เกิดประสพ, 103 หน้า.

ระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database System) เป็นระบบฐานข้อมูลที่มีผู้นิยมใช้กันมาก ในการจัดเก็บข้อมูลลงในฐานข้อมูลให้มีความถูกต้องตามหลักของระบบฐานข้อมูลและความต้องการของแต่ละระบบงานเป็นเรื่องยากหากจะต้องป้องกันความผิดพลาดดังกล่าวโดยการเขียนโปรแกรม การจัดเก็บข้อมูลที่ไม่ถูกต้องลงในฐานข้อมูลจะส่งผลให้การประมวลผลจากข้อมูลดังกล่าวเกิดความผิดพลาด และต้องใช้เวลาในการแก้ไขจัดการข้อมูลให้มีความถูกต้อง ดังนั้นระบบจัดการฐานข้อมูล จึงมีดาต้าเบสทริกเกอร์ (Database Trigger) หรือกฎข้อบังคับของฐานข้อมูลทำหน้าที่ตรวจสอบข้อบังคับของข้อมูลในการประมวลผลคำสั่ง SQL (Structured Query Language) ประเภท DML (Data Manipulation Language) ซึ่งเป็นคำสั่งที่ใช้ในการเปลี่ยนแปลงข้อมูลในฐานข้อมูล ให้มีความถูกต้องตามความต้องการของระบบงาน

ปัจจุบันการสร้างดาต้าเบสทริกเกอร์ให้ตรงตามความต้องการของระบบงานนั้น กระทำได้ด้วยผู้ดูแลจัดการฐานข้อมูลเป็นผู้กำหนดความถูกต้องของข้อมูล โดยพิจารณาจากความต้องการของระบบงาน ซึ่งอาจจะเกิดการผิดพลาดและสิ้นเปลืองเวลาได้หากมีการเปลี่ยนแปลงความต้องการของระบบงานขึ้นใหม่ ดังนั้นในงานวิจัยนี้จึงเสนอแนวคิดในการสร้างดาต้าเบสทริกเกอร์ โดยใช้กฎ (Rules) ที่ได้จากการทำเหมืองข้อมูล (Data mining) มาเป็นตัวสร้างดาต้าเบสทริกเกอร์ขึ้นมาเอง เพื่อเป็นการลดระยะเวลาและเพิ่มความถูกต้องสูงสุดให้กับฐานข้อมูล

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2551 ลายมือชื่ออาจารย์ที่ปรึกษา_____

SIRIKANJANA PILABUTR : DATABASE TRIGGER CREATION WITH DATA MINING. THESIS ADVISOR : ASSOC. PROF. NITTAYA KERDPRASOP, Ph.D., 103 PP.

INDUCTIVE DATABASE/DATABASE TRIGGER/CLASSIFICATION

Database Trigger in Relational Database Management System (RDBMS), is used for checking data integrity in database in case of transact-DML submission. Data manipulation must follow business rules and requirements. DML or Data Manipulation Language is one type of Structured Query Language (SQL) to insert, update and delete data in RDBMS. In this thesis, we propose a novel method to create database triggers semi-automatically from classification rules. Our experiments show an interesting improvement over traditional method of database manipulation. The proposed method can reduce time to create a complete set of database triggers and increase database performance in terms of integrity constraints.

School of Computer Engineering Student's Signature_____

Academic Year 2008 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณ บุคคล และกลุ่มบุคคลต่าง ๆ ที่ได้กรุณาให้คำปรึกษา แนะนำ ช่วยเหลือ อย่างดียิ่ง ทั้งในด้านวิชาการ และด้านการดำเนินงานวิจัย ดังนี้

- รองศาสตราจารย์ ดร. นิตยา เกิดประสพ อาจารย์ที่ปรึกษาวิทยานิพนธ์
- รองศาสตราจารย์ ดร. กิตติศักดิ์ เกิดประสพ
- ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์ และผู้ช่วยศาสตราจารย์ ดร.คะชา ชาญศิลป์ อาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
- คุณกัลญา พับโพธิ์ เลขานุการสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ให้ความช่วยเหลือในการประสานงานด้านเอกสารต่าง ๆ ระหว่างศึกษา
- คุณณัฐพล พันนุรัตน์ คุณภูวดล เข็มพิลา และคุณเกียรติศักดิ์ คิวขุนทด ที่ให้คำปรึกษาและช่วยเหลือด้วยดีมาโดยตลอด
- คุณณัฐภัทร แสนมูลเมือง และคุณศิริพร ชัยประกายวรรณ์ ที่ให้กำลังใจด้วยดีตลอดมา

นอกจากนี้ ขอขอบคุณครู อาจารย์ทั้งในอดีตและปัจจุบันที่ให้ความรู้แก่ผู้วิจัยจนประสบความสำเร็จในชีวิต

ท้ายที่สุด ขอกราบขอบพระคุณบิดา มารดา ที่ให้กำเนิด อุปการะเลี้ยงดูอบรม และส่งเสริมการศึกษาเป็นอย่างดีมาโดยตลอด ทำให้ผู้วิจัยมีความรู้ ความสามารถ มีจิตใจที่เข้มแข็งและช่วยเหลือตัวเองได้จนประสบความสำเร็จ

ศิริกาญจนา พิลาบุตร

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ก
บทคัดย่อภาษาอังกฤษ ..... ข
กิตติกรรมประกาศ ..... ค
สารบัญ ..... ง
สารบัญตาราง ..... ฉ
สารบัญรูป ..... ช
บทที่
1 บทนำ 1
1.1 ความสำคัญและที่มาของปัญหา 1
1.2 วัตถุประสงค์ของการทำวิจัย 4
1.3 ขอบเขตของงานวิจัย 4
1.4 ประโยชน์ที่คาดว่าจะได้รับ 5
2 ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง 6
2.1 ระบบฐานข้อมูลเชิงสัมพันธ์ 6
2.2 ภาษา SQL 6
2.3 การทำเหมืองข้อมูล 8
2.4 อัลกอริทึมการจำแนกประเภทข้อมูล 11
2.5 ดาต้าเบสทริกเกอร์ 16
2.5.1 รูปแบบและข้อจำกัดของดาต้าเบสทริกเกอร์ 17
2.5.2 การนำดาต้าเบสทริกเกอร์มาใช้เพื่อควบคุมความถูกต้องของข้อมูล 19
3 ระเบียบวิธีวิจัย 24
3.1 ขั้นตอนการวิจัย 24
3.2 กระบวนการทำงานของระบบสร้างดาต้าเบสทริกเกอร์ 25
3.3 วิธีการค้นหากฎการจำแนกด้วยโปรแกรม WEKA 28
3.4 อัลกอริทึมการแปลงกฎการจำแนกเป็นดาต้าเบสทริกเกอร์ 34

สารบัญ (ต่อ)

หน้า

3.4.1 การค้นหากฎประเภทการจำแนกข้อมูล 35
3.4.2 อัลกอริทึมการแปลงกฎการจำแนกเป็นดาต้าเบสทริกเกอร์ 39
4 การทดสอบและอภิปรายผล 44
4.1 ข้อมูลที่ใช้ในการทดลอง 44
4.1.1 โรคเบาหวาน 45
4.1.2 การรักษาโรคเบาหวานด้วยอินซูลิน 46
4.1.3 ข้อมูลที่ใช้ในการทดลองสร้างดาต้าเบสทริกเกอร์ โดยการทำเหมืองข้อมูล 50
4.1.4 กฎที่ได้จากการทำเหมืองข้อมูลประเภทการจำแนกข้อมูล 53
4.2 การสร้างดาต้าเบสทริกเกอร์จากกฎการจำแนก 53
4.2.1 การนำดาต้าเบสทริกเกอร์ไปสร้างในฐานข้อมูล 55
4.2.2 การทดสอบดาต้าเบสทริกเกอร์ด้วยการเปลี่ยนแปลงฐานข้อมูล 58
5 บทสรุป 78
5.1 สรุปผลการวิจัย 79
5.2 การประยุกต์งานวิจัย 79
5.3 ปัญหาและข้อเสนอแนะ 80
รายการอ้างอิง 81
ภาคผนวก
ภาคผนวก ก. บทความผลงานวิจัยที่นำเสนอในการประชุมเสนอผลงานวิจัย ระดับบัณฑิตศึกษาแห่งชาติ ครั้งที่ 11 84
ภาคผนวก ข. โปรแกรมแปลงกฎการจำแนกข้อมูลเป็นดาต้าเบสทริกเกอร์ 96
ประวัติผู้เขียน 103

สารบัญตาราง

ตารางที่ **หน้า**

3.1 ตัวอย่างข้อมูลลูกค้าธนาคาร (TBCustomer) 35
3.2 ตัวอย่างข้อมูลการอนุมัติบัตรเครดิต (TBCredit) 36
3.3 ตัวอย่างข้อมูลที่ใช้ในการหากฎประเภทการจำแนก 37
4.1 แสดงความสัมพันธ์ระหว่างดัชนีมวลกายและระดับความอ้วน 46

สารบัญรูป

รูปที่ **หน้า**

1.1 แสดงตัวอย่างดาต้าเบสทริกเกอร์ 2
1.2 แสดง FRAMEWORK ของระบบงาน 3
1.3 แสดงตัวอย่างของ Rule และ Database trigger 4
2.1 ขั้นตอนการทำเหมืองข้อมูล (Data mining) 10
2.2 โครงสร้างสถาปัตยกรรมของ Inductive database 11
2.3 แสดงอัลกอริทึม Simple-rule 12
2.4 แสดงอัลกอริทึม Branch-and-Bound Pruning 15
3.1 ภาพรวมการทำงานของระบบ 26
3.2 ขั้นตอนการทำงานของโปรแกรม 27
3.3 แสดงรูปแบบไฟล์ข้อมูล ARFF (Attribute-Relation File Format) 29
3.4 แสดงหน้าต่างการนำเข้าและเตรียมข้อมูลสำหรับการค้นหากฎการจำแนก 31
3.5 แสดงการเลือกคอลัมน์ในกรณีไม่ต้องการคอลัมน์นั้นในการค้นหากฎ 32
3.6 แสดงหน้าต่างการเลือกอัลกอริทึม J48 เพื่อใช้ค้นหากฎการจำแนก 33
3.7 แสดงหน้าต่างผลลัพธ์ของการค้นหากฎการจำแนกข้อมูล 34
3.8 แสดงรูปแบบคำสั่งการเชื่อมตาราง 36
3.9 แสดงผลการค้นหากฎประเภทการจำแนก 38
3.10 แสดงการตีความหมายของกฎประเภทการจำแนกข้อมูล 38
3.11 แสดงอัลกอริทึมการสร้างดาต้าเบสทริกเกอร์ 40
3.12 แสดงข้อความกฎข้อแรกที่รับเข้ามาในโปรแกรม 41
3.13 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_1 ที่สร้างจากกฎข้อแรก 41
3.14 แสดงข้อความกฎข้อที่สองที่รับเข้ามาในโปรแกรม 42
3.15 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_2 ที่สร้างจากกฎข้อที่สอง 42
3.16 แสดงข้อความกฎที่รับเข้ามาในโปรแกรม 43
3.17 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_3 ที่สร้างจากกฎข้อที่สาม 43
4.1 สูตรการคำนวณดัชนีมวลกาย BMI (Body Mass Index) 46

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.2 แสดงรูปแบบคำสั่งในการเตรียมข้อมูลเพื่อทำเหมืองข้อมูล 51
4.3 แสดงข้อมูลในตารางทั้งหมดที่ใช้ในการทดลอง 52
4.4 แสดงดาต้าเบสทริกเกอร์ที่แปลงจากกฎการจำแนก 54
4.5 แสดงขั้นตอนการสร้างดาต้าเบสทริกเกอร์ในระบบฐานข้อมูล 55
4.6 แสดงการเรียกดูดาต้าเบสทริกเกอร์ทั้งหมดที่ถูกสร้างขึ้นในฐานข้อมูล 56
4.7 แสดงการเรียกดูรายละเอียดของการสร้างดาต้าเบสทริกเกอร์ 57
4.8 แสดงการเรียกดูออบเจ็กต์ที่ถูกอ้างถึงภายในดาต้าเบสทริกเกอร์ 57
4.9 แสดงการเรียกดูคำสั่งภายในดาต้าเบสทริกเกอร์ 58
4.10 แสดงผลการทดสอบคำสั่งที่ 1 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์ 60
4.11 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 1 โดยไม่มีดาต้าเบสทริกเกอร์ 61
4.12 แสดงผลการทดสอบคำสั่งที่ 1 โดยมีการสร้างดาต้าเบสทริกเกอร์ 62
4.13 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 1 โดยมีดาต้าเบสทริกเกอร์ 63
4.14 แสดงผลการทดสอบคำสั่งที่ 2 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์ 65
4.15 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 2 โดยไม่มีดาต้าเบสทริกเกอร์ 66
4.16 แสดงผลการทดสอบคำสั่งที่ 2 โดยมีการสร้างดาต้าเบสทริกเกอร์ 67
4.17 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 2 โดยมีดาต้าเบสทริกเกอร์ 68
4.18 แสดงผลการทดสอบคำสั่งที่ 3 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์ 70
4.19 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 3 โดยไม่มีดาต้าเบสทริกเกอร์ 70
4.20 แสดงผลการทดสอบคำสั่งที่ 3 โดยมีการสร้างดาต้าเบสทริกเกอร์ 72
4.21 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 3 โดยมีดาต้าเบสทริกเกอร์ 72
4.22 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 4 โดยไม่มีดาต้าเบสทริกเกอร์ 74
4.23 แสดงผลการทดสอบคำสั่งที่ 4 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์ 74

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.24 แสดงข้อมูลหลังจากทดสอบคำสั่งที่ 3 ตามเงื่อนไขคำสั่ง
โดยไม่มีดาต้าเบสทริกเกอร์ 75
4.25 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 4 โดยมีดาต้าเบสทริกเกอร์ 76
4.26 แสดงผลการทดสอบคำสั่งที่ 4 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์ 77
4.27 แสดงข้อมูลหลังจากทดสอบคำสั่งที่ 4 ตามเงื่อนไขคำสั่ง
โดยมีดาต้าเบสทริกเกอร์ 77

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหา

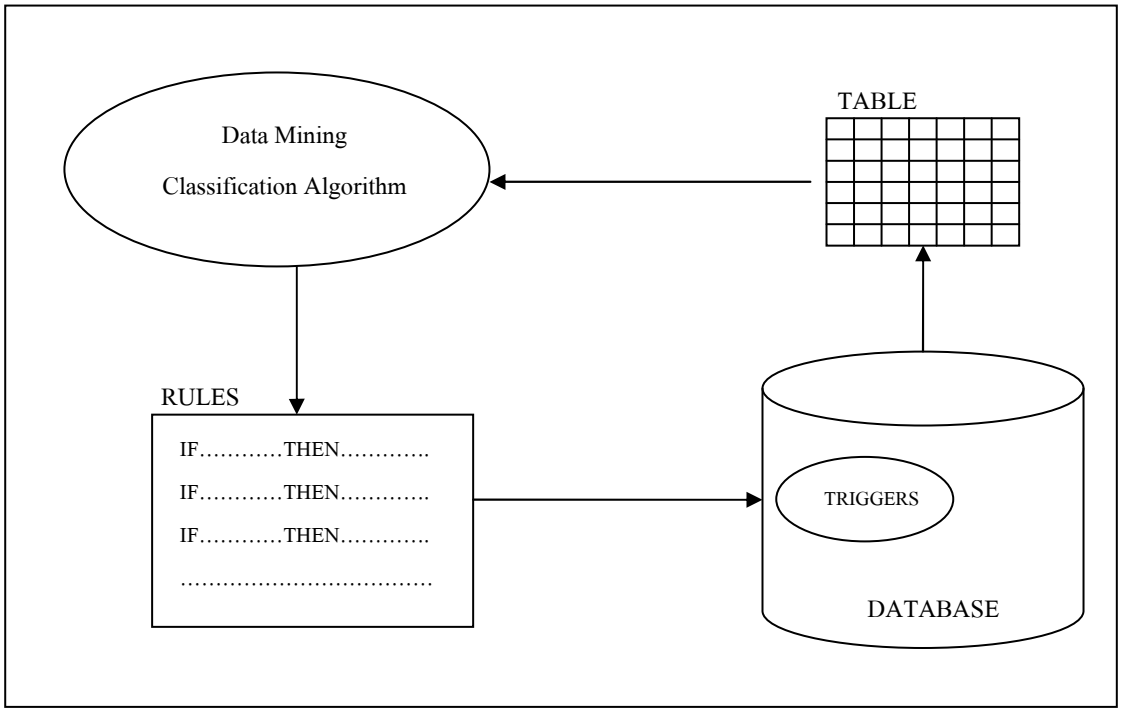
ระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database System) เป็นระบบฐานข้อมูลที่มีผู้นิยมใช้กันมากในปัจจุบัน ระบบฐานข้อมูลเชิงสัมพันธ์เป็นการเก็บข้อมูลอยู่ในรูปแบบตาราง (Table)โดยแต่ละตารางจะมีข้อมูลที่มีความสัมพันธ์กัน ในการจัดเก็บข้อมูลลงในฐานข้อมูลให้มีความถูกต้องตามหลักของระบบฐานข้อมูลและความต้องการของแต่ละระบบงาน (Business rules) สามารถทำได้โดยการเขียนส่วนที่ป้องกันข้อมูลผิดพลาดลงในโปรแกรมประยุกต์ที่มีการติดต่อกับข้อมูลในฐานข้อมูล เพื่อควบคุมการเปลี่ยนแปลงข้อมูลที่จะเกิดขึ้น ซึ่งเป็นเรื่องที่ยุ่งยากในกรณีที่ต้องเขียนโปรแกรมที่มีความซับซ้อน หากมีการจัดเก็บข้อมูลที่ไม่ถูกต้องตามระบบงานลงในฐานข้อมูลจะส่งผลให้การประมวลผลจากฐานข้อมูลดังกล่าวเกิดความผิดพลาด และต้องใช้เวลาในการแก้ไขจัดการข้อมูลให้มีความถูกต้อง (Inmon, 1992a; 1992b) ดังนั้นระบบจัดการฐานข้อมูล (Database Management System : DBMS) จึงมีดาต้าเบสทริกเกอร์ (Database trigger) ซึ่งทำหน้าที่ตรวจสอบกฎข้อบังคับของข้อมูลในการประมวลผลคำสั่ง SQL (Structured Query Language) ประเภท DML (Data Manipulation Language) ให้มีความถูกต้องตามความต้องการของระบบงาน ซึ่งคำสั่ง SQL ประเภท DML ประกอบไปด้วยคำสั่ง INSERT, UPDATE และ DELETE คือการเพิ่ม การแก้ไขและการลบข้อมูล ตามลำดับ คำสั่งเหล่านี้เป็นคำสั่งที่ใช้ในการเปลี่ยนแปลงข้อมูล (Transaction)ในฐานข้อมูล

ตัวอย่างดาต้าเบสทริกเกอร์ดังแสดงในรูปที่ 1.1 เป็นดาต้าเบสทริกเกอร์ที่ใช้ร่วมกับฐานข้อมูลการเก็บข้อมูลการอนุมัติบัตรเครดิต เพื่อควบคุมไม่ให้เกิดข้อผิดพลาดหรือข้อมูลที่ขัดแย้งกันในการเพิ่มและแก้ไขข้อมูลที่เกี่ยวกับผู้ป่วยที่มีความเสี่ยงเป็นโรคเบาหวานและปริมาณการให้อินซูลินเพื่อรักษาระดับน้ำตาลในเลือด โดยจะพิจารณาจากเงื่อนไขในดาต้าเบสทริกเกอร์ที่สร้างไว้ ซึ่งถ้าหากข้อมูลที่ปรับปรุงเข้าไปใหม่มีความขัดแย้งกับดาต้าเบสทริกเกอร์ก็จะไม่สามารถที่จะปรับปรุงข้อมูลดังกล่าวลงไปได้ โดยจะมีข้อความแจ้งข้อผิดพลาดให้ทราบตามที่กำหนดไว้ในการสร้างดาต้าเบสทริกเกอร์

```
CREATE TRIGGER rule_1
ON diabetes
FOR UPDATE, INSERT
AS
   IF (SELECT COUNT(*)
       FROM diabetes
       WHERE (Diabetes_family = 'yes')
                and (BMI <= 24.9)
                and (diabetes <> 'no'))> 0
BEGIN
   ROLLBACK TRAN
   RAISERROR ('diagnose error')
END;
```

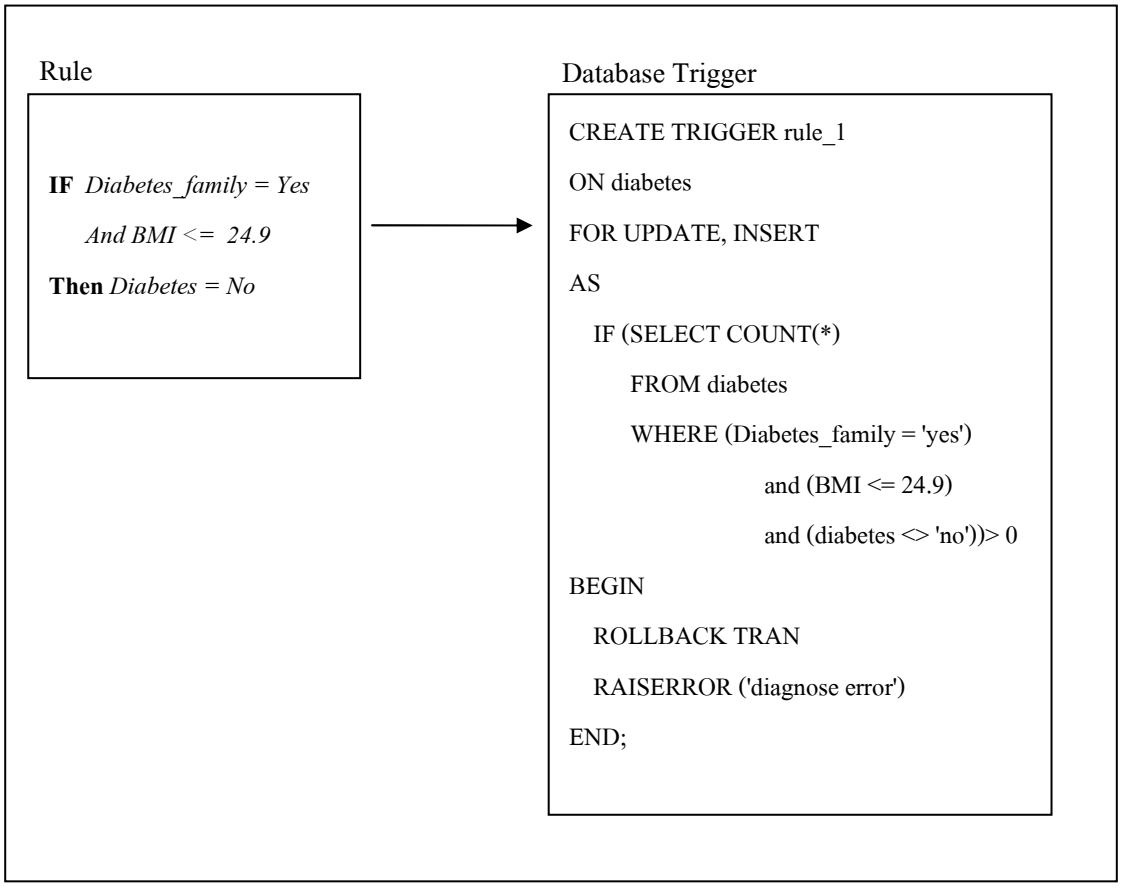
รูปที่ 1.1 แสดงตัวอย่างดาต้าเบสทริกเกอร์

ปัจจุบันในระบบฐานข้อมูลขนาดใหญ่ เช่น Oracle หรือ Microsoft SQL Server การสร้างดาต้าเบสทริกเกอร์เพื่อควบคุมความถูกต้องของข้อมูลให้ตรงตามความต้องการของระบบงานนั้นสามารถกระทำได้ด้วยผู้ดูแลจัดการฐานข้อมูล (Database Administrator : DBA) เป็นผู้กำหนดความถูกต้องของข้อมูล โดยพิจารณาจากความต้องการของระบบงานนั้น ๆ ซึ่งอาจจะเกิดข้อผิดพลาดและทำให้สิ้นเปลืองเวลาได้ถ้าหากมีการเปลี่ยนแปลงความต้องการของระบบงานขึ้นใหม่ ดังนั้นในงานวิจัยนี้จึงได้เสนอแนวคิดในการสร้างดาต้าเบสทริกเกอร์ จากกฎ (Rules) ที่ได้จากการประยุกต์ใช้ความรู้ในการทำเหมืองข้อมูล (Data mining) โดยใช้อัลกอริทึม Classification มาเป็นตัวช่วยสร้างดาต้าเบสทริกเกอร์ขึ้นมาด้วยวิธีกึ่งอัตโนมัติ เพื่อเป็นการลดระยะเวลา เพิ่มประสิทธิภาพและความถูกต้องสูงสุดให้กับฐานข้อมูล



รูปที่ 1.2 แสดง FRAMEWORK ของระบบงาน

จากรูปที่ 1.2 เป็นการแสดง FRAMEWORK ของงานวิจัยนี้ โดยจะเป็นการนำข้อมูลที่เก็บไว้จากฐานข้อมูลจริงผนวกกับเทคโนโลยีการทำเหมืองข้อมูล ซึ่งเป็นเทคโนโลยีที่เริ่มเป็นที่รู้จักกันอย่างแพร่หลาย มาเป็นตัวช่วยในการค้นหากฎข้อบังคับ ที่สามารถนำมาใช้ในกระบวนการสร้างดาต้าเบสทริกเกอร์ การทำเหมืองข้อมูลเป็นการค้นหาความสัมพันธ์และรูปแบบทั้งหมด ที่มีอยู่จริงในฐานข้อมูล ความสัมพันธ์และรูปแบบเหล่านั้นได้ถูกซ่อนไว้ภายในข้อมูลจำนวนมากที่มีอยู่ การทำเหมืองข้อมูลจะทำการสำรวจและวิเคราะห์ข้อมูลให้อยู่ในรูปแบบที่เต็มไปด้วยความหมายและอยู่ในรูปของกฎ โดยความสัมพันธ์นี้จะแสดงให้เห็นถึงความรู้ต่าง ๆ ที่มีประโยชน์ในฐานข้อมูล ซึ่งเป็นความรู้ที่ถูกต้อง และสามารถนำไปใช้ประโยชน์ได้ หรือจะเรียกการกระทำแบบนี้ว่า Knowledge Discovery in Databases หรือเรียกสั้น ๆ ว่า KDD (ในวิทยานิพนธ์เล่มนี้ใช้ชื่อ KDD ในความหมายเดียวกับคำว่า Data mining หรือการทำเหมืองข้อมูล) ในการวิจัยชิ้นนี้เราจะใช้เทคนิคการค้นหากฎการจำแนกประเภทข้อมูล (Classification rules) เพื่อใช้ในการค้นหาประเภทของข้อมูลเพื่อนำมาสร้างเป็นดาต้าเบสทริกเกอร์ โดยกฎที่ได้จากการทำเหมืองข้อมูลลักษณะนี้จะแสดงในรูปแบบที่สามารถเข้าใจได้ง่าย คือ สาเหตุไปสู่ผลลัพธ์ที่เป็นประเภท หรือ IF *condition* Then *specified-class* จากนั้นจะทำการนำกฎที่ได้มาสร้างเป็นดาต้าเบสทริกเกอร์เพื่อไปใช้เป็นตัวควบคุมความถูกต้องของฐานข้อมูลต่อไป ดังมีตัวอย่างการแปลงกฎเป็นดาต้าเบสทริกเกอร์ในรูปที่ 1.3



รูปที่ 1.3 แสดงตัวอย่างของ Rule และ Database Trigger

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อศึกษาการทำงานของดาต้าเบสทริกเกอร์ในระบบฐานข้อมูลเชิงสัมพันธ์

1.2.2 เพื่อศึกษาและประยุกต์การทำเหมืองข้อมูลโดยใช้อัลกอริทึม Classification ที่ใช้ในการสร้างกฎ (Rules)

1.2.3 เพื่อศึกษาการนำกฎไปใช้ในระบบฐานข้อมูลเชิงสัมพันธ์

1.2.4 เพื่อสร้างดาต้าเบสทริกเกอร์จากกฎที่สร้างได้จากการทำเหมืองข้อมูล

1.3 ขอบเขตของงานวิจัย

กฎที่ได้จากการทำเหมืองข้อมูลเป็น Classification rules ที่ใช้เทคนิค Decision-tree induction รันด้วยระบบ WEKA (Witten and Frank, 2005) จาก Classification rules ที่ได้ผู้วิจัยได้พัฒนาโปรแกรมภาษา JAVA เพื่อแปลงจาก Rules เป็นดาต้าเบสทริกเกอร์ ส่วนของการทดสอบดาต้าเบสทริกเกอร์กระทำบน Microsoft SQL Server Version 2000 งานวิจัยนี้เน้นที่ขั้นตอนวิธีการ

สร้างดาต้าเบสทริกเกอร์โดยยังไม่ได้พัฒนาอัลกอริทึมในส่วนของการ Maintain ดาต้าเบสทริกเกอร์ แต่ได้เสนอแนวทางการจัดการไว้ในบทที่ 5 ซึ่งเป็นบทสรุปของวิทยานิพนธ์นี้

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 สามารถนำเทคโนโลยีที่มีประสิทธิภาพสองด้านคือ การสร้างดาต้าเบสทริกเกอร์และการทำเหมืองข้อมูล มาทำงานร่วมกันให้เกิดแนวทางใหม่เพื่อเป็นทางเลือกในการพัฒนาระบบจัดการฐานข้อมูลให้จัดเก็บข้อมูลที่ถูกต้องยิ่งขึ้น

1.4.2 สามารถค้นหาความรู้ที่อยู่ในฐานข้อมูลโดยการทำเหมืองข้อมูล และนำความรู้นั้นมาใช้ประโยชน์ได้

1.4.3 ความรู้ที่ได้จากงานวิจัยนี้ สามารถนำไปใช้เป็นแนวทางในด้านการพัฒนาระบบจัดการฐานข้อมูลให้มีประสิทธิภาพดีขึ้นต่อไปในอนาคต

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

2.1 ระบบฐานข้อมูลเชิงสัมพันธ์

ฐานข้อมูล (Database) จัดเป็นองค์ประกอบหนึ่งที่สำคัญของระบบคอมพิวเตอร์ที่ใช้งานในด้านธุรกิจหรืองานในสาขาอื่น ๆ ที่จำเป็นต้องเก็บข้อมูลที่เกี่ยวข้องกันเป็นจำนวนมาก ซึ่งปัจจุบันนิยมเก็บข้อมูลเหล่านั้นในรูปแบบของระบบฐานข้อมูลเชิงสัมพันธ์ (Relational database system) ความหมายของระบบฐานข้อมูลเชิงสัมพันธ์คือ ข้อมูลที่เกี่ยวข้องกันในเรื่องใดเรื่องหนึ่งที่นำมาจัดเก็บรวบรวมไว้ในที่เดียวกันในลักษณะของตารางหรือเทเบิล (Table) แบบสองมิติซึ่งประกอบด้วยแถว (Row) และคอลัมน์ (Column) เพื่อให้สามารถใช้ประโยชน์จากข้อมูลดังกล่าวได้อย่างสะดวก ไม่ว่าจะเป็นการนำข้อมูลนั้นมาใช้โดยตรงได้แก่การเรียกดูข้อมูล (Retrieval) การจัดการกับข้อมูลต่าง ๆ ที่อยู่ในตาราง การทำรายการเปลี่ยนแปลง (Transaction) หรือการนำข้อมูลนั้นมาใช้ทางอ้อมโดยการนำข้อมูลมาเก็บใน Data warehouse ซึ่งก็คือการนำข้อมูลจากฐานข้อมูลจำนวนมากมายมหาศาลมารวบรวมเก็บไว้ในที่เดียวกันซึ่งข้อมูลที่ถูกนำมาเก็บนี้จะมีข้อมูลและความรู้มากมายซ่อนไว้อยู่ จึงมีประโยชน์สำหรับการตั้งข้อคำถาม (Query) เพื่อสอบถามหรือการวิเคราะห์ (Analysis) ข้อมูลที่เก็บไว้จำนวนมากดังกล่าว เรียกวิธีการนี้ว่าการทำเหมืองข้อมูล (Data mining)

2.2 ภาษา SQL

Elmasri and Navathe (1994) ได้อธิบายว่า ระบบฐานข้อมูลที่นิยมใช้กันทั่วโลกมีด้วยกันหลายระบบ แต่ภาษาคอมพิวเตอร์ที่ใช้ในการจัดการกับฐานข้อมูลจะเป็นมาตรฐานเดียวกัน โดยใช้ภาษา SQL (Structured Query Language) พัฒนาโดยบริษัท IBM

ภาษา SQL เริ่มพัฒนาครั้งแรกในต้นทศวรรษที่ 1970 ที่ San Jose Research Laboratory (ปัจจุบันเปลี่ยนชื่อเป็น Almaden Research Center) โดยมีชื่อในครั้งแรกว่า SEQUEL (Structured English Query Language) (Chamberlin and Boyce,1974) ต่อมาได้เปลี่ยนชื่อเป็น SQL และเป็นต้นแบบภาษา SQL ของผลิตภัณฑ์ด้านฐานข้อมูล เช่น Oracle, DB2, MS-SQL Server, Progress, SyBase, Informic, FoxPro, Access, Paradox, SQLite รวมทั้ง MySQL และโปรแกรมอื่น ๆ อีกมากมาย แสดงให้เห็นถึงความสำคัญของภาษานี้ได้เป็นอย่างดี

ปี ค.ศ.1986 American National Standards Institute (ANSI) ได้กำหนดมาตรฐาน SQL ขึ้นมาเพื่อให้ผลิตภัณฑ์ทั้งหมดเป็นไปตามมาตรฐานเดียวกัน (American National Standard Institute. X3H2 Records, 1978-1995) อย่างไรก็ตามการทำเช่นนี้ทำให้เกิดปัญหาบางประการขึ้น เป็นผลให้มาตรฐาน ANSI มีข้อจำกัดอยู่บ้าง เพราะ SQL มีสองชนิดคือ ชนิดโต้ตอบได้ (Interactive) กับชนิดที่ฝัง (Embedded) อยู่ในโปรแกรม ส่วนใหญ่แล้วทั้งสองชนิดปฏิบัติงานอย่างเดียวกันแต่นำไปใช้ต่างกัน SQL ชนิดโต้ตอบได้ใช้เพื่อปฏิบัติงานกับฐานข้อมูลโดยตรงเพื่อนำเอาผลลัพธ์ไปใช้งาน ส่วน SQL แบบฝังในโปรแกรมประกอบด้วยคำสั่งต่าง ๆ ของ SQL ที่ใส่ในโปรแกรมที่ส่วนมากแล้วเขียนด้วยภาษาอื่น เช่น COBOL, Pascal, C/C++, Visual Basic, Delphi, Java เป็นต้น

ANSI ประกาศมาตรฐาน SQL มาแล้วหลายรุ่น ถ้ามีการประกาศมาตรฐานขึ้นในปีใดก็จะมีเลขปี ค.ศ. ต่อท้าย เช่น ANSI-86, SQL-89, SQL-92 และ SQL-2008 เป็นมาตรฐานล่าสุด

แม้ว่าจะมีผลิตภัณฑ์ฐานข้อมูลออกมามากมายหลายยี่ห้อ แต่ด้วยมาตรฐานภาษา SQL ที่ใช้ร่วมกัน ทำให้มีความสามารถพื้นฐานเหมือนกัน อาจแตกต่างกันได้บ้างเพราะแต่ละผลิตภัณฑ์ก็ล้วนแล้วแต่พยายามสร้างจุดแข็งให้กับผลิตภัณฑ์ของตนเอง จึงเป็นหน้าที่ของผู้ใช้ผลิตภัณฑ์นั้น ที่ต้องศึกษาในส่วนที่แตกต่างเพื่อที่จะได้นำมาใช้งานได้อย่างเต็มประสิทธิภาพ ซึ่งภาษา SQL ตามมาตรฐาน ANSI มีคุณสมบัติดังนี้

1) โครงสร้างของภาษาคล้ายภาษาอังกฤษ สามารถเรียกดูข้อมูลที่ระบุได้ตามความต้องการ เปลี่ยนแปลง เพิ่มเติม และลบข้อมูลออกจากระบบ

2) ใช้ลักษณะเชิงประกาศ (Declarative) เพื่อให้ใช้งานง่าย เพียงระบุความต้องการก็สามารถใช้งานได้แล้ว

3) สามารถประมวลผลข้อมูลเป็นกลุ่มได้

4) ใช้ได้กับผู้ใช้ทุกกลุ่ม ไม่ว่าจะเป็นผู้ดูแลระบบฐานข้อมูล (Database Administrator: DBA), โปรแกรมเมอร์ (Programmer) หรือผู้ใช้ทั่วไป (End user)

คำสั่งของภาษา SQL สามารถจำแนกได้เป็น 5 กลุ่มดังนี้

1) DML (Data Manipulation Language) เป็นคำสั่งที่ใช้ในการจัดการกับข้อมูลต่าง ๆ ใน Table คำสั่งในกลุ่มนี้ได้แก่

- INSERT สำหรับเพิ่มข้อมูลใหม่
- UPDATE สำหรับเปลี่ยนแปลงข้อมูล
- DELETE สำหรับลบข้อมูล

2) DDL (Data Definition Language) เป็นคำสั่งที่ใช้ในการสร้าง เปลี่ยนแปลง หรือลบ Database object ตัวอย่างของ Database object ได้แก่ Table, User, View

- CREATE สำหรับสร้าง Database object
- ALTER สำหรับเปลี่ยนแปลง Database object
- DROP สำหรับลบ Database object

3) DCL (Data Control Language) เป็นคำสั่งสำหรับกำหนดหรือถอนสิทธิ์ (Privileged) สำหรับการทำงานต่าง ๆ ในระบบฐานข้อมูล คำสั่งในกลุ่มนี้ได้แก่

- GRANT สำหรับให้สิทธิ์แก่ผู้ใช้
- REVOKE สำหรับถอนสิทธิ์ของผู้ใช้

4) Retrieval Command เป็นคำสั่งที่ใช้ในการเลือกดูข้อมูล สามารถเลือกดูข้อมูลในลักษณะง่าย ๆ ไปจนถึงการเลือกดูข้อมูลจากหลาย Table และเงื่อนไขในการเลือกดูข้อมูลที่ซับซ้อน คำสั่งที่ใช้ในการเรียกดูข้อมูลมีเพียงคำสั่งเดียวคือ

- SELECT

5) Transaction Command เป็นคำสั่งที่ใช้ในการจัดการ Database transaction คำสั่งในกลุ่มนี้ได้แก่

- COMMIT สำหรับยืนยันการทำรายการ
- ROLLBACK สำหรับยกเลิกการทำรายการ

2.3 การทำเหมืองข้อมูล

การทำเหมืองข้อมูลเมื่อถูกผนวกเข้ากับระบบจัดการฐานข้อมูลจะทำให้ได้ระบบ KDDMS: Knowledge and Data Discovery Management System การทำเหมืองข้อมูลเป็นเทคนิคในการกลั่นกรอง วิเคราะห์หรือแม้แต่ค้นหาความรู้ที่ซ่อนอยู่ในข้อมูลที่มีปริมาณมหาศาลเพื่อให้ได้ข้อมูลที่มีประโยชน์ และนำข้อมูลที่มีประโยชน์นี้ใช้เป็นฐานความรู้เพื่อการบริหารงานได้ โดยการทำเหมืองข้อมูลนี้ต้องอาศัยหลักความรู้พื้นฐานของหลายแขนงวิชาอาทิเช่น ความรู้ทางด้านสถิติ (Statistics) ระบบฐานข้อมูล (Database system) การเรียนรู้ของเครื่อง (Machine learning) การรู้จำรูปแบบ (Pattern recognition) และคณิตศาสตร์ (Mathematics) เป็นต้น

ผลลัพธ์จากการทำเหมืองข้อมูลนี้มีอยู่ด้วยกันหลายแบบเช่น รูปแบบ (Pattern) ต้นแบบ (Model) กฎ (Rule) การทำนายหรือคาดการณ์ล่วงหน้า (Prediction) ทั้งนี้ผลลัพธ์ที่ได้ออกมาจะขึ้นอยู่กับกรรมวิธีหรืออัลกอริทึมที่เราใช้ในการทำเหมืองข้อมูล ซึ่งผลลัพธ์หรือความรู้ที่ได้จากการทำเหมืองข้อมูลมีหลายรูปแบบ ได้แก่

1) กฎความสัมพันธ์ (Association rules)

เป็นการแสดงความสัมพันธ์ของเหตุการณ์หรือวัตถุ ที่เกิดขึ้นพร้อมกัน ตัวอย่างของการประยุกต์ใช้กฎความสัมพันธ์ เช่น การวิเคราะห์ข้อมูลการขายสินค้า โดยเก็บข้อมูลจากระบบ ณ จุดขาย (POS: Point-of-sale) หรือร้านค้าออนไลน์ แล้วพิจารณาสินค้าที่ผู้ซื้อมักจะซื้อพร้อมกัน เช่น ถ้าพบว่าคนที่ซื้อเทปกระดาษมักจะซื้อเทปกาวด้วย ร้านค้าก็อาจจะจัดร้านให้สินค้าสองอย่างอยู่ใกล้กันเพื่อเพิ่มยอดขาย หรืออาจจะพบว่าหลังจากคนซื้อหนังสือ ก แล้ว มักจะซื้อหนังสือ ข ด้วย ก็สามารถนำความรู้นี้ไปแนะนำผู้ที่กำลังจะซื้อหนังสือ ก ได้

2) กฎการจำแนกประเภท/ทำนายข้อมูล (Data classification /prediction rules)

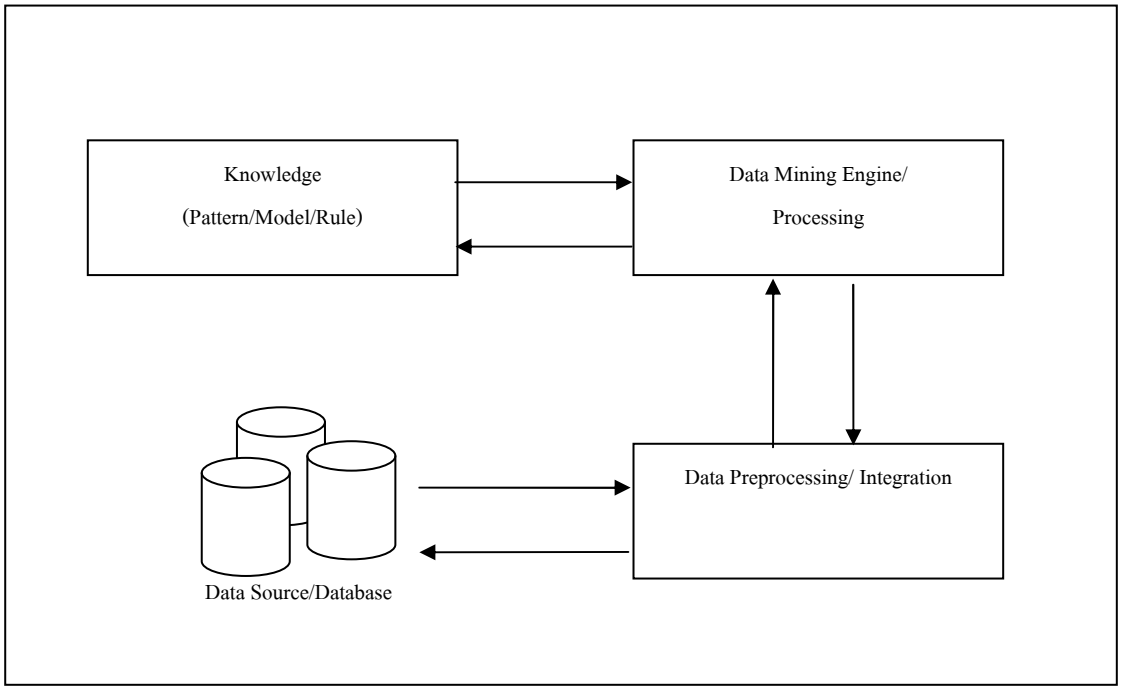
เป็นกฎที่ระบุประเภทของวัตถุหรือคลาสจากคุณสมบัติของวัตถุ เช่น หาความสัมพันธ์ระหว่างผลการตรวจร่างกายต่าง ๆ กับการเกิดโรค โดยใช้ข้อมูลผู้ป่วยและการวินิจฉัยของแพทย์ที่เก็บไว้ เพื่อนำมาช่วยวินิจฉัยโรคของผู้ป่วย หรือการวิจัยทางการแพทย์ ในทางธุรกิจจะใช้เพื่อดูคุณสมบัติของผู้ที่จะก่อหนี้ดีหรือหนี้เสีย เพื่อประกอบการพิจารณาการอนุมัติเงินกู้

3) ลักษณะที่ใช้แบ่งกลุ่มข้อมูล (Data clustering attributes)

เป็นการแสดงลักษณะจำเพาะที่สามารถแบ่งข้อมูลที่มีลักษณะคล้ายกันออกเป็นกลุ่ม เช่น แบ่งกลุ่มผู้ป่วยที่เป็นโรคเดียวกันตามลักษณะอาการ เพื่อนำไปใช้ประโยชน์ในการวิเคราะห์หาสาเหตุของโรค โดยพิจารณาจากผู้ป่วยที่มีอาการคล้ายคลึงกัน

4) จินตทัศน์ (Visualization)

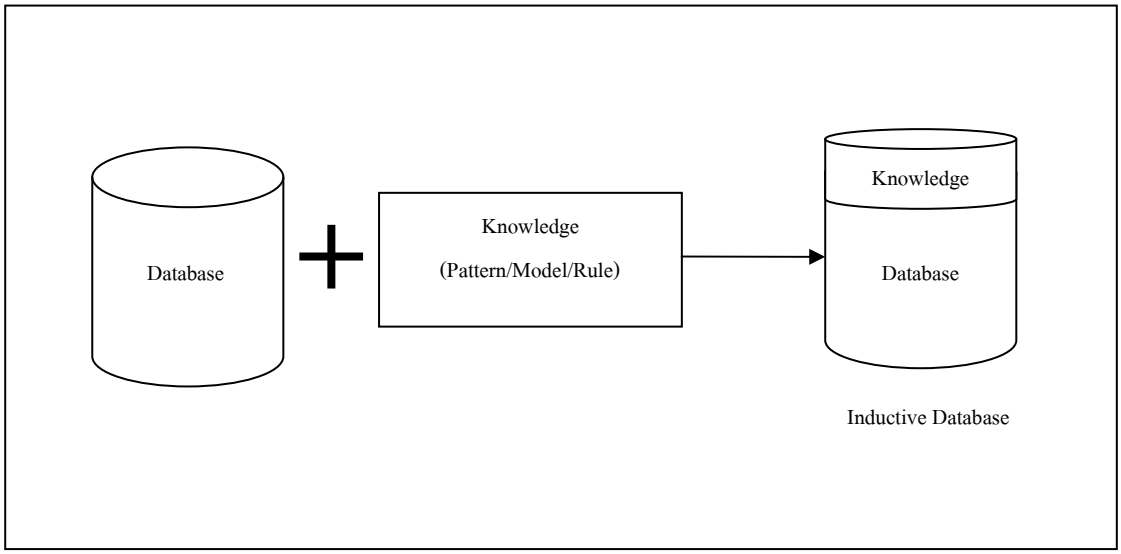
เป็นการสร้างภาพคอมพิวเตอร์กราฟิก (Computer graphic) ที่สามารถนำเสนอข้อมูลมากมายอย่างครบถ้วนแทนการใช้ข้อความนำเสนอ ซึ่งอาจพบข้อมูลที่ซ้อนเร้นเมื่อดูข้อมูลชุดนั้นด้วยจินตทัศน์



รูปที่ 2.1 ขั้นตอนการทำเหมืองข้อมูล (Data Mining)

รูปที่ 2.1 แสดงขั้นตอนการทำเหมืองข้อมูล โดยเริ่มจากนำข้อมูลที่เราต้องการทำเหมืองข้อมูลมาเข้าสู่ขั้นตอน คือการจัดเตรียม (Data preprocessing) กลั่นกรองข้อมูลให้เหมาะสมกับเครื่องมือที่ใช้ทำเหมืองข้อมูล รวมทั้งการทำความเข้าใจปัญหาและทำความเข้าใจข้อมูลที่มีอยู่ ขั้นตอนต่อไปคือขั้นตอนในการสร้างแบบจำลอง เป็นการนำข้อมูลที่เตรียมไว้มาเข้ากระบวนการหรือเครื่องมือในการทำเหมืองข้อมูล (Data mining engine) จากนั้นจะได้ผลลัพธ์ที่ต้องการออกมา โดยผลลัพธ์ที่ได้ออกมานั้นจะขึ้นอยู่กับอัลกอริทึมที่ใช้เป็นเครื่องมือ และนำผลลัพธ์ที่ได้นั้นไปทำการประเมินตรวจสอบว่าตรงกับปัญหาที่เราตั้งไว้ตั้งแต่แรกหรือไม่ ถ้าได้ผลการประเมินเป็นที่น่าพอใจก็จะสามารถนำผลลัพธ์นั้นไปใช้งานต่อไปได้ หากไม่ได้ตรงตามที่ตั้งไว้ ก็สามารถที่จะย้อนกลับไปทำขั้นตอนใดขั้นตอนหนึ่งใหม่ได้ เพื่อให้ได้ความถูกต้องสูงสุด

Imielinski and Mannila (1996) ได้เสนอแนวคิดที่จะนำความรู้ที่ซ่อนอยู่เหล่านั้นมาผูกรวมติดอยู่กับระบบฐานข้อมูล (DBMS: Database Management System) ซึ่งถูกเรียกว่า Inductive Database โดยแสดงโครงสร้างของสถาปัตยกรรมในส่วนของ Inductive Database ไว้ดังรูปที่ 2.2 และต่อมา Imielinski and Virmani (1999) ได้นำแนวคิดของ Inductive Database มาพัฒนาต่อเพื่อใช้ประโยชน์ในการทำเหมืองข้อมูลร่วมกับภาษา SQL โดยได้ออกแบบภาษา MSQL ซึ่งเป็นการรวมคำสั่ง Mining เข้ากับภาษา SQL



รูปที่ 2.2 โครงสร้างสถาปัตยกรรมของ Inductive Database

2.4 อัลกอริทึมการจำแนกประเภทข้อมูล

การจำแนกประเภทข้อมูล (Data classification) เป็นปัญหาพื้นฐานของการเรียนรู้แบบมีผู้สอน (Supervised learning) โดยปัญหาคือการทำนายประเภทหรือคลาสของวัตถุจากคุณสมบัติต่าง ๆ ของวัตถุ ซึ่งการเรียนรู้แบบมีผู้สอนจะสร้างฟังก์ชันเชื่อมโยง ระหว่างคุณสมบัติของวัตถุ กับประเภทของวัตถุจากตัวอย่างสอน แล้วจึงใช้ฟังก์ชันนี้ทำนายประเภทของวัตถุที่ไม่เคยพบ หรืออาจกล่าวได้ว่าเป็นกระบวนการสร้างโมเดล เพื่อแสดงให้เห็นลักษณะที่เป็นความแตกต่างระหว่างคลาสหรือ กลุ่มของข้อมูลได้ ซึ่งโมเดลที่ใช้จำแนกข้อมูลออกเป็นกลุ่มตามที่ได้กำหนดไว้ จะขึ้นอยู่กับการวิเคราะห์เซตของข้อมูลทดลอง (Training data) โดยนำ Training data มาสอนให้ระบบเรียนรู้ว่ามีข้อมูลใดอยู่ในclass เดียวกันบ้าง ตัวอย่างเช่น

กำหนดตัวอย่าง Training Data

$$E = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} \tag{2-1}$$

โดยที่ X_i คือ เวคเตอร์แสดงคุณสมบัติของวัตถุ

$$X_i \in A_1 \times A_2 \times \cdots \times A_m \tag{2-2}$$

โดยที่ A_i เป็นเซตจำกัดระบุคุณสมบัติ และ $Y \in C$ เป็นประเภทหรือคลาสของวัตถุ ซึ่งกำหนดไว้ในเซตจำกัด C ต้องการหาฟังก์ชัน $f(x)$ ซึ่งให้ค่าเวคเตอร์ y จากเวคเตอร์ x ที่กำหนด โดยที่ความผิดพลาดของการทำนายตัวอย่าง Trainning data มีค่าน้อยสุด หรือมีค่า e น้อยที่สุด โดยกำหนดให้

$$e = \sum_{i=1}^{n} t(f(x_i), y_i) \tag{2-3}$$

เมื่อ $t(f(x), y)$ คือ ฟังก์ชันเปรียบเทียบผลการทำนายคลาสของข้อมูล x (นั่นคือ $f(x)$) โดยโมเดลที่ได้จากการจำแนกประเภทข้อมูล เปรียบเทียบกับคลาสที่แท้จริง (นั่นคือค่า y) ของข้อมูล โดยกำหนดค่าที่ได้จากการเปรียบเทียบดังนี้

$$t(a,b) = \begin{cases} 0, \; if \; a = b \\ 1, \; if \; a \neq b \end{cases} \tag{2-4}$$

อัลกอริทึมที่ซับซ้อนน้อยที่สุดของงาน Classification เรียกว่า Simple-rule algorithm (Holte, 1993) เป็นอัลกอริทึมอย่างง่ายที่ใช้ในการสร้างกฎการจำแนกประเภทข้อมูลที่เรียกว่า Simple-rule เนื่องจากอัลกอริทึมนี้จะสร้างกฎในรูปแบบ “IF *condition* Then *specified-class*” โดยจะมีเพียง Attribute เดียวปรากฏใน Condition เช่น IF Blood_Sugar > 120 Then Diabetes = yes ดังแสดงในรูปที่ 2.3

For each attribute,
    For each value of that attribute, make a rule as follows:
        Count how often each class appears.
        Find the most frequent class.
        Make the rule assign that class to this attribute-value.
    Calculate the error rate of the rules.
Choose the rules with the smallest error rate.

รูปที่ 2.3 แสดงอัลกอริทึม Simple-Rule (Holte, 1993)

ในงานวิจัยชิ้นนี้ได้ใช้อัลกอริทึม J48 ซึ่งเป็นอัลกอริทึมในกลุ่มของ Classification ของระบบ WEKA (Witten and Frank, 2005) มาเป็นเครื่องมือในการค้นหากฎการจำแนก ซึ่งจะถูกนำไปใช้ในขั้นตอนของการสร้าง Database triggers อัลกอริทึม J48 นี้ใช้หลักการเดียวกันกับอัลกอริทึม C4.5 (Quinlan, 1992) ซึ่งเป็นอัลกอริทึมที่พัฒนามาจากอัลกอริทึม ID3 (Quinlan, 1986) ที่ใช้สำหรับสร้างต้นไม้เพื่อการตัดสินใจหรือเรียกว่า Decision tree เช่นเดียวกันกับอัลกอริทึม CART (Larose, 2005)

อัลกอริทึม C4.5 นี้ใช้หลักการของ Information gain หรือที่เรียกว่า Entropy reduction มาใช้เพื่อสร้างกิ่ง (node) ของต้นไม้ตัดสินใจ เกณฑ์ที่ใช้ช่วยประกอบการเลือกแอททริบิวต์เพื่อทำหน้าที่เป็นโหนดคือ ทดลองเลือกแต่ละแอททริบิวต์มาทำหน้าที่เป็นโหนดและวัดค่า gain ซึ่งเป็นค่าที่ชี้ว่าแอททริบิวต์นั้นจะช่วยจำแนกคลาสของข้อมูลได้ดีเพียงใด โดยการจำแนกที่ดีที่สุด คือให้ Leaf node ที่เป็นข้อมูลเดียวกันทั้งหมด และค่า gain ที่สูงที่สุดหมายถึง การจำแนกคลาสที่ดีที่สุด

ค่า gain เป็นค่าที่บอกระดับความสามารถของการจำแนกคลาสของแอททริบิวต์ที่ถูกเลือกให้ทำหน้าที่เป็นตัวตรวจสอบเพื่อจัดกลุ่มของข้อมูลในแต่ละ Leaf node เป็นคลาสเดียวกันทั้งหมด หรือมีข้อมูลต่างคลาสปะปนกันมาบ้างเพียงเล็กน้อย สามารถหาค่า gain ได้สมการที่ 2-5

$$gain(X) = \text{info}(T) - \text{info}_x(T) \tag{2-5}$$

โดยกำหนดให้ T แทน เซตของข้อมูลฝึก (Training data)

X แทน แอททริบิวต์ที่ถูกเลือกให้เป็นตัวตรวจสอบเพื่อจัดกลุ่มข้อมูล

$\text{info}(T)$ คือฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อสามารถจำแนกคลาสของข้อมูลได้ มีสูตรในการคำนวณดังสมการที่ 2-6

$$\text{info}(T) = -\sum_{j=1 \text{ to } K}[freq(C_j, T)/|T|] \times \log_2[freq(C_j, T)/|T|] \tag{2-6}$$

โดยกำหนดให้ $|T|$ คือ จำนวนข้อมูลทั้งหมดในเซตของข้อมูลฝึก

$freq(C_j, T)$ คือ ความถี่ที่ข้อมูลใน T ปรากฏเป็นคลาส C_j

$\text{info}_x(T)$ คือฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อการจำแนกคลาสของข้อมูล โดยใช้แอททริบิวต์ X เป็นตัวตรวจสอบเพื่อจำแนกกลุ่มของข้อมูล มีสูตรในการคำนวณดังสมการที่ 2-7

$$\text{info}_x(T) = -\sum_{i=1 \text{ to } K}(|T_i|/|T|) \times \text{info}(T_i) \quad (2\text{-}7)$$

โดยกำหนดให้ i คือ จำนวนค่าที่เป็นไปได้ของแอททริบิวต์ X

$|T_i|$ คือ จำนวนข้อมูลที่มีค่า $X = i$

ผลลัพธ์ที่ได้จากการเรียนรู้ คือ โมเดลจัดประเภทข้อมูล (Classifier model) สามารถแสดงได้ในหลายรูปแบบ เช่น Classification (IF-THEN) rules, Decision tree, Mathematical formula หรือ Neural networks และจะนำข้อมูลอีกส่วนทำหน้าที่ เป็นข้อมูลทดสอบ (Testing data) ข้อมูลทดสอบจะมีโครงสร้างเหมือนข้อมูลฝึก โดยข้อมูลแต่ละรายการจะมีคลาสหรือกลุ่มที่ถูกต้องกำกับอยู่ในขั้นตอนการทดสอบโมเดลกลุ่มที่แท้จริงของข้อมูลที่ใช้ทดสอบนี้จะถูกนำมาเปรียบเทียบกับค่าของกลุ่มที่หามาได้จากโมเดลเพื่อตรวจสอบว่าตรงกันหรือไม่ ถ้าไม่ตรงกันจะเรียกว่าโมเดลมี error ของการจำแนกประเภทข้อมูล โดยเราจะปรับปรุงโมเดลจนกว่าจะได้ค่าความถูกต้องในระดับที่น่าพอใจ หลังจากนั้นเมื่อมีข้อมูลใหม่ที่ไม่ทราบกลุ่มเข้ามา เราจะนำข้อมูลผ่านโมเดล เพื่อให้โมเดลช่วยทำนายกลุ่มของข้อมูลนี้ได้

Mehta, Rissanen, and Agrawal (1995) ได้เสนอแนวคิดในการจัดการ Decision tree ที่เป็นผลลัพธ์จากการทำเหมืองข้อมูลของงาน Classification ซึ่งบ่อยครั้งที่ได้ Decision tree ที่มีขนาดใหญ่หรือได้จำนวนกิ่ง (Node) เป็นจำนวนมากซึ่งทำให้ยากต่อการเข้าใจของมนุษย์ โดยเรียกอัลกอริทึมนี้ว่า MDL (Minimum Description Length) ที่เกี่ยวกับการตัดกิ่งของ Decision tree โดยใช้หลักการในการยึดแอททริบิวต์เป้าหมายเป็นหลักในการทำงาน Classification เพื่อลดค่าความผิดพลาดของงานดังกล่าว โดยผลการทดสอบพบว่าอัลกอริทึม MDL นั้นสามารถตัดกิ่งของ Tree ให้มีขนาดเล็กและมีค่าความถูกต้องของ Tree สูง นอกจากนี้ยังช่วยเพิ่มประสิทธิภาพในด้านความเร็วของการสร้างอีกด้วย

Mehta, Agrawal, and Rissanen (1996) ได้เสนออัลกอริทึม SLIQ ซึ่งเป็นแนวคิดใหม่ในการทำเหมืองข้อมูลของงาน Classification ที่สามารถจัดการกับข้อมูลที่เป็นตัวเลขและข้อมูลที่เป็นข้อความในการสร้าง Decision tree ได้ โดยแนวคิดดังกล่าวถูกพัฒนามาจากอัลกอริทึม MDL ที่ยังยึดหลักการในการสร้าง Decision tree ที่มีความสั้นกระชับและมีค่าความถูกต้องสูง ทำให้ได้อัลกอริทึม SLIQ ที่มีความสามารถในการทำงาน Classification กับข้อมูลที่มีขนาดใหญ่มากและมีจำนวนของชนิดข้อมูล แอททริบิวต์และข้อมูลที่ใช้เรียนรู้ไม่จำกัด ซึ่งเป็นเครื่องมือที่น่าสนใจในการทำเหมืองข้อมูลเป็นอย่างมาก

Garofalakis, Hyun, Rajeev, and Shim (2000) ได้เสนอแนวคิดเกี่ยวกับอัลกอริทึมที่มีประสิทธิภาพต่อการสร้าง Decision tree ด้วย Constraint โดยการสร้าง Decision tree ที่ไม่มีความซับซ้อนและมีจำนวนกิ่งปริมาณไม่มากเกินไป โดยหลักการของการสร้าง Decision tree นี้จะให้ผู้ใช้งานเป็นผู้กำหนดข้อจำกัด (Constraint) ของขนาด Tree หรือค่าความถูกต้อง (Accuracy) ของ Tree ดังกล่าวเองได้ เรียกวิธีการนี้ว่า Branch-and-Bound Pruning Algorithm จากแนวคิดดังกล่าวทำให้การนำ Decision tree ไปใช้งานมีประสิทธิภาพขึ้นในแง่ของความถูกต้อง ลดระยะเวลาในการสร้าง การทำความเข้าใจและการนำไปประยุกต์ใช้กับงานทั่ว ๆ ไปได้ โดยมีอัลกอริทึมแสดงในรูปที่ 2.4 ดังนี้

```
Procedure PruneUsingConst(Node N , integer l , real B ):
    Mark node N
    if B ≤ Bound[ N,l ] return
    for i:= to l do
        if B > Bound[ N,l ]
        Bound[ N,l ] :=B
    If Tree[ N,l ].lowCost >B or Tree[ N,l ].lowCost =C(S)+ 1
        Return
    Else if N is not leaf node and l≥3 {
        For k1:= 1 to l−2 do {
            k2 :=l−k1 −1
            If C_sprit(N)+ 1 + Tree[ N1,k1 ].lowCost + Tree[ N2,k2 ].lowCost ≤ B {
                    B1 := B−(C_split(N)+ 1)− Tree[ N2,k2 ].lowCost
                    B2 := B−(C_split(N)+ 1)− Tree[ N1,k1 ].lowCost
                    PruneUsingConst(N1,k1,B1);
                    PruneUsingConst(N2,k2,B2);
            }
        }
}
```

รูปที่ 2.4 แสดงอัลกอริทึม Branch-and-Bound Pruning (Minos, Dongjoon, and Kyuseok, 2000)

Hulten, Spencer, and Domingos (2001) ได้เสนออัลกอริทึม CVFDT (Concept-adapting Very Fast Decision Tree learner) ซึ่งเป็นอัลกอริทึมที่มีประสิทธิภาพในการสร้าง Decision tree จากข้อมูลที่มีการเปลี่ยนแปลงอย่างต่อเนื่อง โดยอาศัยการเรียนรู้จากข้อมูลที่มีอยู่เดิมร่วมกับข้อมูลที่ป้อนเข้ามาใหม่ เพื่อที่จะทำการเรียนรู้ซ้ำทุกครั้งที่มีข้อมูลใหม่ป้อนเข้ามา ทำให้ได้ผลลัพธ์หรือโมเดลที่ทันสมัยตลอดเวลา

Kai-Uwe and Oliver (2001) ได้เสนอแนวคิดในการใช้คำสั่ง SQL พื้นฐานสำหรับสร้าง Decision tree classification โดยได้มุ่งประเด็นถึงการสร้างและนำ Tree ไปประยุกต์ใช้บนพื้นฐานของการใช้ประโยชน์ในการวิเคราะห์ข้อมูลต่าง ๆ ในเชิงธุรกิจ โดยข้อมูลดังกล่าวจะถูกเก็บไว้ในระบบจัดการฐานข้อมูล

Jin and Agrawal (2003) ได้เสนอแนวคิดในการตัดกิ่งของ Tree กับข้อมูลที่เป็นตัวเลขโดยเรียกว่า NIP (Numerical Interval Pruning) เพื่อลดเวลาในการสร้าง Tree และได้เสนอวิธีลดปริมาณของข้อมูลตัวอย่างเพื่อการเรียนรู้ โดยยังคงมีค่าความถูกต้องสูงเช่นเดิม ซึ่งทั้งสองแนวคิดนี้เป็นวิธีการสร้าง Tree ที่มีประสิทธิภาพ

นอกจากนี้ยังมีอัลกอริทึมอื่น ๆ ที่มุ่งเน้นในการสร้าง Decision tree ที่มีประสิทธิภาพในเรื่องของการลดระยะเวลาในการสร้างและยังคงมีค่าความถูกต้องสูง อาทิเช่น อัลกอริทึม SPRINT (Shafer, Agrawal, and Mehta, 1996), อัลกอริทึม QUEST (Loh and Shih, 1997) และอัลกอริทึม RainForest (Gehrke, Ramakrishnan, and Ganti, 1998)

จากงานวิจัยที่ผ่านมา พบว่ายังไม่มีการนำเทคนิคการทำเหมืองข้อมูล Classification ไปใช้ช่วยในการสร้างดาต้าเบสทริกเกอร์ งานวิจัยชิ้นนี้จึงเป็นงานวิจัยแรกที่ริเริ่มนำแนวความคิดการทำ Classification มาใช้ประโยชน์ในการสร้างดาต้าเบสทริกเกอร์ดังกล่าว

2.5 ดาต้าเบสทริกเกอร์

ดาต้าเบสทริกเกอร์ (Database trigger) หรืออาจเรียกสั้น ๆ ว่าทริกเกอร์ คือ Stored procedure (ใน Microsoft SQL Server) หรือ โปรแกรม PL/SQL (ใน Oracle) ที่ถูกสร้างขึ้นและจัดเก็บในระบบฐานข้อมูลซึ่งจะถูกเรียกให้ทำงานโดยอัตโนมัติเมื่อมีเหตุการณ์หรือ Trigger event เกิดขึ้น

ความหมายของ Trigger event นั้นหมายถึงการเรียกใช้คำสั่ง SQL ประเภท DML (Data Manipulation Language) ซึ่งประกอบไปด้วยคำสั่ง INSERT, UPDATE และ DELETE กระทำกับข้อมูลหรือเทเบิลบนระบบฐานข้อมูล โดยที่เทเบิลดังกล่าวได้มีการสร้างดาต้าเบสทริกเกอร์กำกับไว้ เช่น ถ้ากำหนดให้ทริกเกอร์ทำงานกับเทเบิลเมื่อมีการ INSERT ดังนั้นถ้ามีการใช้คำสั่ง INSERT กับเทเบิลใดทริกเกอร์ก็จะทำงานทันที เราสามารถทำให้ทริกเกอร์หนึ่ง ๆ ทำงานเมื่อมีการปรับปรุง

ข้อมูลมากกว่าหนึ่งแบบ เช่น กำหนดทริกเกอร์หนึ่งให้ทำงานเมื่อเกิดทั้งการ INSERT หรือ UPDATE ข้อมูล นอกจากนี้แต่ละเทเบิลจะสร้างทริกเกอร์กี่ตัวก็ได้ (Multiple trigger) แต่ต้องมีชื่อของทริกเกอร์ต่างกัน

2.5.1 รูปแบบและข้อจำกัดของดาต้าเบสทริกเกอร์

การสร้างทริกเกอร์โดย Transact-SQL (Microsoft, WWW, 2007) ของ Microsoft SQL Server มีรูปแบบดังนี้

```
CREATE TRIGGER trigger_name
ON { table | view }
[WITH ENCRYPTION]
{
   { { FOR | AFTER | INSTEAD OF } {[INSERT] [,] [UPDATE] [,] [DELETE]}
      [WITH APPEND]
      [NOT FOR REPLICATION]
      AS
      [ { IF UPDATE (column)
      [{ AND | OR } UPDATE (column)]
      [...n]
      | IF (COLUMNS_UPDATED ( ) {bitwise_operator} updated_bitmask)
            {comparison_operator} column_bitmask [...n]
      }]
      sql_statement [...n]
   }
}
```

โดยที่ trigger_name คือ ชื่อของทริกเกอร์

table คือ ชื่อของเทเบิลที่ต้องการสร้างทริกเกอร์

sql_statement คือ คำสั่ง Transact-SQL ที่ต้องการให้ทำงานเมื่อทริกเกอร์นั้นถูกเรียกใช้ขึ้นมา ซึ่งอาจมีเงื่อนไขที่ใช้ตรวจสอบว่าจะกระทำอะไรกับเทเบิล คือ DELETE, INSERT หรือ UPDATE ยกเว้นคำสั่ง SELECT

ตัวอย่าง การใช้ทริกเกอร์เพื่อแจ้งข้อความเมื่อมีการใช้คำสั่ง INSERT, UPDATE และ DELETE กับเทเบิลที่ชื่อว่า titles

```
USE pubs
GO
CREATE TRIGGER title_trigger
ON titles
FOR INSERT, UPDATE, DELETE
AS RAISEROR ('Trigger Action', 14, 10)
```

เมื่อสร้างดาต้าเบสทริกเกอร์ตามตัวอย่างแล้วทดสอบการทำงานของทริกเกอร์ โดยเมื่อทำการเพิ่มหรือแก้ไขหรือลบข้อมูลออกจากเทเบิล titles จะได้ข้อความจากทริกเกอร์แสดงออกมาว่า 'Trigger Action' ดังที่กำหนดไว้ในทริกเกอร์

ในระบบฐานข้อมูลเชิงสัมพันธ์ใช้ดาต้าเบสทริกเกอร์ในการควบคุมความถูกต้องของการใช้งานข้อมูล และช่วยลดความผิดพลาดที่อาจเกิดขึ้นได้ ถ้าหากไม่ใช้ดาต้าเบสทริกเกอร์ช่วยในการตรวจสอบความถูกต้องของข้อมูล ก็จะต้องเขียนโปรแกรมควบคุมความถูกต้องในฝั่งของ Program client ซึ่งจะเป็นการไม่สะดวกถ้าหากจำนวนบรรทัดของโปรแกรมที่มีจำนวนมาก ทำให้เกิดความยุ่งยากในการแก้ไขโปรแกรมจำนวนมากถ้ามีการเปลี่ยนแปลงข้อจำกัดบางอย่างเกิดขึ้น ดาต้าเบสทริกเกอร์จึงช่วยให้ความถูกต้องของข้อมูลมีมากขึ้น ยิ่งในกรณีที่ข้อมูลมีความสำคัญมาก ๆ ซึ่งจะเสมือนกับว่ากำลังใช้ดาต้าเบสทริกเกอร์เป็นตัวช่วยในการสร้าง Business rules ขึ้นมาให้กับองค์กรด้วย

ข้อจำกัดในการใช้ดาต้าเบสทริกเกอร์

1) สามารถสร้างทริกเกอร์เพื่อใช้กับการ INSERT, UPDATE หรือ DELETE กับเทเบิลเท่านั้น ไม่สามารถสร้างทริกเกอร์เพื่อใช้กับวิว หรือเทเบิลชั่วคราว (Temporary table) ได้

2) ข้อบังคับหรือคอนสเตรนท์ (Constraint) ที่กำหนดไว้กับเทเบิลจะถูกทำงานก่อน เช่นถ้ามี CHECK คอนสเตรนท์เพื่อตรวจสอบข้อมูล และถ้าการ INSERT นั้นไม่ตรงตามเงื่อนไขของ CHECK ก็จะทำให้เพิ่มข้อมูลในเทเบิลไม่ได้ ซึ่งจะทำให้ทริกเกอร์ของการ INSERT ไม่ทำงานไปด้วย

3) ทริกเกอร์ไม่สามารถถูกเรียกให้ทำงานโดยตรงได้ จะทำงานเมื่อเรียกใช้คำสั่ง INSERT, UPDATE หรือ DELETE เท่านั้น นอกจากนี้จะไม่สามารถรับหรือส่งค่าพารามิเตอร์ได้

4) ทริกเกอร์จะถือว่าเป็นหนึ่งทรานแซ็กชั่น (Transaction) แม้ว่าจะไม่มีการใส่ BEGIN TRANSACTION ไว้ก็ตาม ในทริกเกอร์นี้สามารถจะเรียกคำสั่ง ROLLBACK ได้ แต่ถ้ามีคำสั่ง ROLLBACK จากในทริกเกอร์ ทั้งทรานแซ็กชั่นจะถูก Roll back ไปด้วย แต่ถ้าเรียกจากแบทช์

ที่เรียกทริกเกอร์อีกทีหนึ่ง คำสั่งของแบทช์จะถูก Roll back การใช้คำสั่ง ROLLBACK ในทริกเกอร์ควรทำเมื่อจำเป็นจริง ๆ เพราะจะมีผลต่อประสิทธิภาพของเครื่อง

5) ทริกเกอร์มักนำไปใช้ตรวจสอบความถูกต้องของข้อมูลที่มีการอ้างหรือสัมพันธ์กันระหว่างเทเบิล (Referential integrity) ซึ่งไม่สามารถจะทำได้โดยใช้คอนสเตรนท์ CHECK หรือมักจะนำมาใช้กับกฎเกณฑ์ทางธุรกิจของบริษัทที่ไม่สามารถควบคุมได้โดยการใช้คอนสเตรนท์ได้เช่นกัน เช่น ถ้ากำหนดวงเงินกู้ของสมาชิกแต่ละครั้งไม่เกิน 10,000 บาท ก็อาจใช้คอนสเตรนท์ CHECK ได้ แต่ถ้ากำหนดว่าวงเงินกู้เก่าของทุกรายการรวมทั้งหมดต้องไม่เกินกว่า 50,000 บาทข้อกำหนดนี้จะต้องใช้ทริกเกอร์เข้ามาควบคุมแทน

6) คำสั่ง CREATE TRIGGER จะต้องเป็นคำสั่งแรกเมื่อเขียนคำสั่งในแบทช์

7) ผู้ที่เป็นเจ้าของเทเบิลเท่านั้น จึงจะมีสิทธิ์ในการสร้างทริกเกอร์บนเทเบิลนั้น

8) สามารถสร้างทริกเกอร์ให้กับดาต้าเบสปัจจุบันเท่านั้น แม้ว่าคำสั่งในทริกเกอร์จะอ้างถึงออบเจ็คในดาต้าเบสอื่นได้ก็ตาม

9) คำสั่ง TRUNCATE TABLE จะไม่ทำให้ทริกเกอร์ที่กำหนดไว้ทำงานเมื่อมีการ DELETE

10) คำสั่ง WRITETEXT สำหรับแก้ไขข้อมูลแบบ Text หรือ Image ซึ่งจะไม่ทำให้ทริกเกอร์ทำงาน

2.5.2 การนำดาต้าเบสทริกเกอร์มาใช้เพื่อควบคุมความถูกต้องของข้อมูล

นอกเหนือจากการนำดาต้าเบสทริกเกอร์มาใช้ควบคุมความถูกต้องของข้อมูลที่กระทำต่อเทเบิลโดยตรงแล้ว เรายังอาจนำดาต้าเบสทริกเกอร์มาใช้ควบคุมข้อมูลที่มีการอ้างถึงกันระหว่างเทเบิลหรือที่เรียกว่า Referential integrity ได้เช่นกัน สำหรับตัวอย่างที่จะนำเสนอต่อไปนี้เป็นการนำดาต้าเบสทริกเกอร์มาใช้ร่วมกับการ DELETE และ INSERT เพื่อให้เห็นการทำ Referential integrity ได้ชัดเจนขึ้น

(1) ดาต้าเบสทริกเกอร์กับการลบข้อมูล

ตัวอย่างนี้เป็นการนำดาต้าเบสทริกเกอร์มาใช้ในการตรวจสอบว่าหากลูกหนี้ยังคงมีรายการค้างชำระอยู่ในเทเบิล members_loan จะลบข้อมูลของลูกหนี้คนนั้นในเทเบิล members ออกไปไม่ได้

ขั้นตอนที่ 1 เตรียมเทเบิลและข้อมูล

ก่อนที่จะสร้างดาต้าเบสทริกเกอร์ ทำการสร้างเทเบิลชื่อ members ที่เก็บข้อมูลสมาชิก และ members_loan เก็บข้อมูลการกู้เงินของสมาชิก โดยทั้งสองเทเบิลจะมีความสัมพันธ์กันด้วยคอลัมน์ id

```
USE pubs
GO
CREATE TABLE members (id int, name varchar(80))
GO
INSERT INTO members VALUES(1, 'JOHN')
INSERT INTO members VALUES(2, 'TOM')
GO
CREATE TABLE members_loan (id int, loanamt money)
INSERT INTO members_loan VALUES(1, 30000)
INSERT INTO members_loan VALUES(2, 20000)
GO
```

ขั้นตอนที่ 2 สร้างดาต้าเบสทริกเกอร์สำหรับ DELETE

สำหรับการทำงานของดาต้าเบสทริกเกอร์ del_member มีวัตถุประสงค์คือไม่ให้มีการ DELETE ข้อมูลจากเทเบิล members ถ้าสมาชิกคนนั้นยังมีหนี้อยู่ใน members_loan ถ้าดูคำสั่งในช่วง BEGIN…END ถ้าพบว่าสมาชิกคนใดยังมีหนี้อยู่จะแสดงข้อความเตือนจาก RAISERROR และให้ ROLLBACK TRANSACTION (หมายถึงให้ย้อนกลับการ DELETE นั่นคือจะได้ข้อมูลที่ DELETE ไปแล้วกลับคืนมา)

```
USE pubs
GO
CREATE TRIGGER del_members
    ON members FOR DELETE
AS
    IF (SELECT COUNT (*)
    FROM members_loan, deleted
    WHERE members_loan.id = deleted.id) > 0
BEGIN
    RAISERROR ('Member still has outstanding loan', 16, 10)
    ROLLBACK TRANSACTION
END
```

deleted หรือ inserted เป็นเทเบิลแบบ logical table ที่มีโครงสร้างเหมือนกับตารางหรือเทเบิลที่ดาต้าเบสทริกเกอร์นั้นทำงานอยู่ ทำหน้าที่เก็บข้อมูลเก่าก่อนลบหรือเก็บข้อมูลใหม่จากการเพิ่มรายการ

ขั้นตอนที่ 3 ทดสอบการทำงานของดาต้าเบสทริกเกอร์

เมื่อลบข้อมูลจาก members ที่มี id = 1 สิ่งที่เราพบคือ เครื่องคอมพิวเตอร์แสดงข้อความที่ได้กำหนดไว้ที่ RAISERROR การที่ผลลัพธ์เป็นเช่นนี้เพราะว่า id = 1 ยังมีรายการที่เป็นหนี้อยู่ในเทเบิล members_loan และสิ่งที่จะได้ต่อจากนั้นคือ ถ้าเราไปเรียกดูข้อมูลทั้งหมดของ members จะพบว่ายังอยู่ครบ 2 แถว (เพราะให้ทำ ROLLBACK TRANSACTION) โดยใช้คำสั่ง SQL ในการลบข้อมูลดังนี้

```
DELETE FROM members
WHERE id =1
```

(2) ดาต้าเบสทริกเกอร์กับการใส่ข้อมูล

สำหรับดาต้าเบสทริกเกอร์ ins_member_loan เป็นตัวอย่างดาต้าเบสทริกเกอร์ที่จะทำงานเมื่อมีการ INSERT ข้อมูลที่ members_loan ซึ่งดาต้าเบสทริกเกอร์จะกันไม่ให้เพิ่มข้อมูลการกู้เงินในเทเบิล members_loan ถ้าสมาชิกคนนั้นมีวงเงินการกู้เก่าและใหม่รวมกันเกิน 25,000 บาท

```
USE pubs
GO
CREATE TRIGGER ins_member_loan
ON members_loan FOR INSERT
AS
    IF (SELECT SUM (members_loan), loanamt
      FROM members_loan, inserted
      WHERE members_loan.id = inserted.id) > 25000
BEGIN
      RAISERROR (‘Member has over limit loan ’, 10, 1)
      ROLLBACK TRANSACTION
END
```

จากนั้นทำการทดสอบโดยทดลอง INSERT ข้อมูลเข้าไปในเทเบิล members_loan จะพบว่าไม่สามารถเพิ่มรายการได้โดยจะแสดงข้อความตามที่กำหนดไว้ใน RAISERROR เนื่องจากเมื่อรวมจำนวนเงินกู้ทั้งหมดและเงินกู้ที่กำลัง INSERT นั้น จะมีค่ารวมทั้งสิ้นเกินกว่า 25,000 บาท นอกจากนี้เมื่อตรวจสอบข้อมูลจาก members_loan ก็ยังคงมีรายการเท่าเดิม โดยใช้คำสั่ง SQL ในการเพิ่มข้อมูลดังนี้

```
INSERT INTO members_loan VALUES(1, 2000)
```

(3) การแก้ไขดาต้าเบสทริกเกอร์

สำหรับดาต้าเบสทริกเกอร์ที่สร้างไว้สามารถเปลี่ยนแปลงได้ด้วยคำสั่ง ALTER TRIGGER ด้วยรูปแบบดังนี้

รูปแบบ

```
ALTER TRIGGER trigger_name
ON table
FOR { [ DELETE] [,] [INSERT] [,] [UPDATE] }
    AS
sql_statement […n]
```

ตัวอย่าง เปลี่ยนเงื่อนไขของทริกเกอร์ ins_members_loan จากวงเงิน 25,000 เป็น 70,000 ดังคำสั่งข้างล่างนี้

```
USE pubs
GO table
ALTER TRIGGER ins_members_loan
ON members_loan FOR INSERT
AS
    IF (SELECT SUM (members_loan, loanamt)
FROM members_loan, inserted
WHERE members_loan.id = inserted.id) > 70000
BEGIN
    RAISERROR (‘Member has over limit loan ’, 16, 10)
    ROLLBACK TRANSACTION
END
```

(4) การลบดาต้าเบสทริกเกอร์

การลบดาต้าเบสทริกเกอร์ที่ไม่ต้องการใช้อีกต่อไปออกจากดาต้าเบส ทำได้โดยใช้คำสั่ง DROP TRIGGER โดยมีรูปแบบดังนี้

รูปแบบ

DROP TRIGGER {trigger} [,…n]

USE pubs

GO

DROP TRIGGER ins_members_loan

บทที่ 3
ระเบียบวิธีวิจัย

งานวิจัยนี้ มีจุดมุ่งหมายที่จะพัฒนากฎที่ได้จากการทำเหมืองข้อมูลเพื่อนำมาใช้ในการสร้างกฎข้อบังคับการเปลี่ยนแปลงฐานข้อมูลหรือที่รู้จักกันในชื่อว่าดาต้าเบสทริกเกอร์ (Database trigger) โดยโปรแกรมที่พัฒนาขึ้นจะใช้ภาษา JAVA ในการพัฒนา และนำเอาเทคโนโลยีทางด้านเหมืองข้อมูลที่เริ่มเป็นที่รู้จักกันแพร่หลายนี้มาประยุกต์ใช้ในการค้นหารูปแบบของข้อมูล เพื่อนำมาสร้างเป็นกฎข้อบังคับ โดยอัลกอริทึมที่ใช้ในการค้นหารูปแบบของข้อมูลเพื่อนำมาใช้ในการสร้างเป็นกฎข้อบังคับนี้คืออัลกอริทึม C4.5 ซึ่งเป็นอัลกอริทึมที่ใช้ในการทำเหมืองข้อมูลประเภทการจำแนก โดยมุ่งเน้นให้อัลกอริทึมสามารถสร้างกฎการจำแนกที่ได้ให้อยู่ในรูปอย่างง่ายต่อการนำไปใช้ในการสร้างกฎข้อบังคับของข้อมูล รายละเอียดในเนื้อหาบทนี้ประกอบด้วยขั้นตอนการวิจัย ปรากฏอยู่ในหัวข้อ 3.1 กระบวนการทำงานของระบบสร้างดาต้าเบสทริกเกอร์ปรากฏอยู่ในหัวข้อ 3.2 วิธีการค้นหากฎการจำแนกด้วยโปรแกรม WEKA ปรากฏอยู่ในหัวข้อที่ 3.3 และอัลกอริทึมการแปลงกฎการจำแนกเป็นดาต้าเบสทริกเกอร์เพื่อใช้ประโยชน์ในการควบคุมความถูกต้องของข้อมูลในฐานข้อมูล ปรากฏอยู่ในหัวข้อ 3.4

3.1 ขั้นตอนการวิจัย

การค้นคว้าวิจัยแบ่งออกเป็น ขั้นตอนดังนี้

1. ศึกษาและรวบรวมงานวิจัยที่เกี่ยวข้อง
2. ศึกษาการทำงานของอัลกอริทึมการจำแนกประเภทข้อมูล และศึกษาลักษณะของข้อมูลที่ใช้ในการค้นหากฎการจำแนกประเภทข้อมูล และรูปแบบของกฎการจำแนกประเภทข้อมูลที่ได้จากการค้นหา จากนั้นศึกษาโปรแกรมระบบ WEKA ซึ่งเป็นโปรแกรมสำเร็จรูป ที่มีการเปิดเผยซอร์ส โค้ด (Open-source environment) ซึ่งเป็นโปรแกรมสำหรับการทำเหมืองข้อมูลที่ถูกพัฒนาโดยทีมงานของมหาวิทยาลัย Waikato ประเทศนิวซีแลนด์ (Witten and Frank, 2005) ซึ่งใช้เป็นโปรแกรมสำหรับค้นหากฎการจำแนกประเภทข้อมูลจากข้อมูลที่ใช้ทดสอบในงานวิจัยครั้งนี้
3. ศึกษาเกี่ยวกับลักษณะการทำงานของดาต้าเบสทริกเกอร์ในระบบฐานข้อมูลเชิงสัมพันธ์ ข้อจำกัดและการใช้ประโยชน์ รวมทั้งออกแบบดาต้าเบสทริกเกอร์ที่จะนำมาใช้ในการควบคุมความถูกต้องของข้อมูล

4. ศึกษาและทำการเลือกอัลกอริทึมการจำแนกประเภทข้อมูลโดยเลือกใช้อัลกอริทึม J48 ซึ่งเป็นอัลกอริทึมที่ใช้หลักการเดียวกันกับอัลกอริทึม C4.5 โดยกฎหรือผลลัพธ์ที่ได้จากการทำเหมืองข้อมูลการจำแนกประเภทข้อมูลนี้มีรูปแบบที่เข้าใจง่ายและเหมาะสมต่อการนำไปสร้างเป็นดาต้าเบสทริกเกอร์

5. ออกแบบอัลกอริทึมการแปลงกฎการจำแนกประเภทข้อมูลเป็นดาต้าเบสทริกเกอร์ และพัฒนาโปรแกรมในรูปแบบของภาษา JAVA

6. ออกแบบฐานข้อมูลและสร้างฐานข้อมูลเพื่อใช้ในการทดสอบการสร้างกฎการจำแนกประเภทข้อมูลและทดสอบการทำงานของดาต้าเบสทริกเกอร์

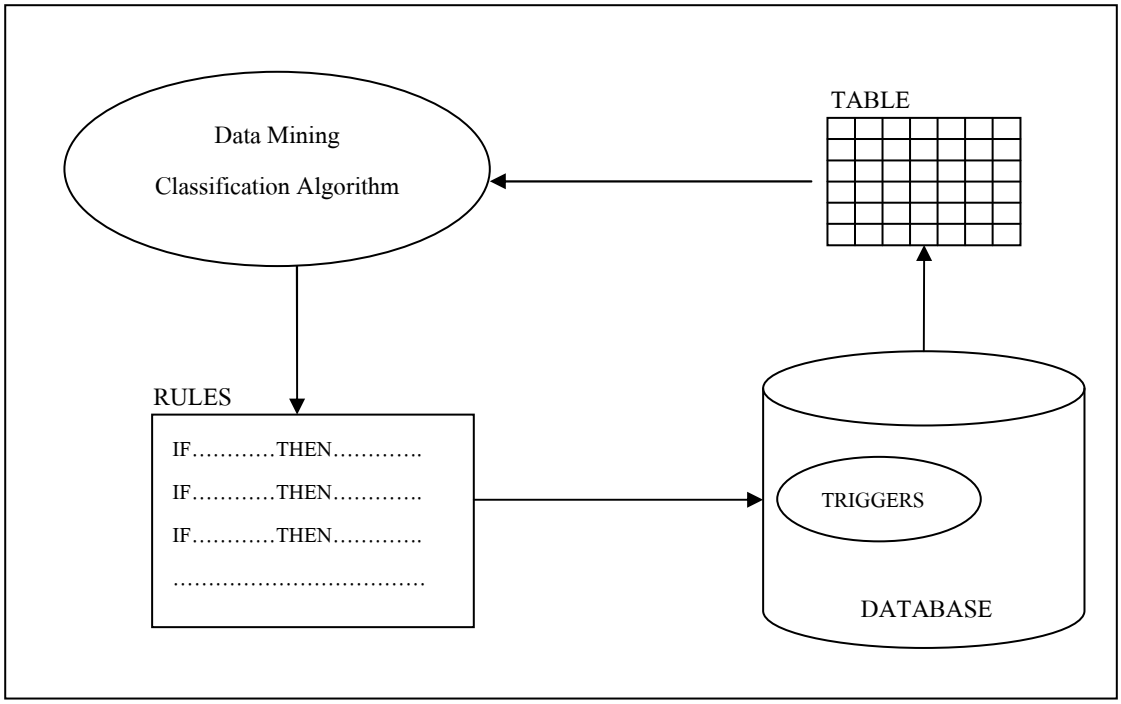
7. ทำการทดสอบอัลกอริทึมกับชุดข้อมูลที่ออกแบบไว้ โดยทดสอบกับระบบฐานข้อมูล Microsoft SQL Server 2000 โดยคอมพิวเตอร์ที่ใช้ในการทดสอบเป็นคอมพิวเตอร์ CPU Intel Celeron M processor 440 ความเร็ว 1.86 GHz หน่วยความจำหลัก 512 MB ฮาร์ดดิสก์ความจุ 80 GB

8. บันทึกผล และเสนอแนะรูปแบบของข้อมูลที่สามารถใช้ในการสร้างกฎข้อบังคับข้อมูล รวมทั้งข้อมูลที่จะนำมาใช้ในการค้นหากฎการจำแนกประเภทข้อมูลเพื่อนำมาใช้สร้างเป็นดาต้าเบสทริกเกอร์ เพื่อให้ได้ประสิทธิภาพมากที่สุด

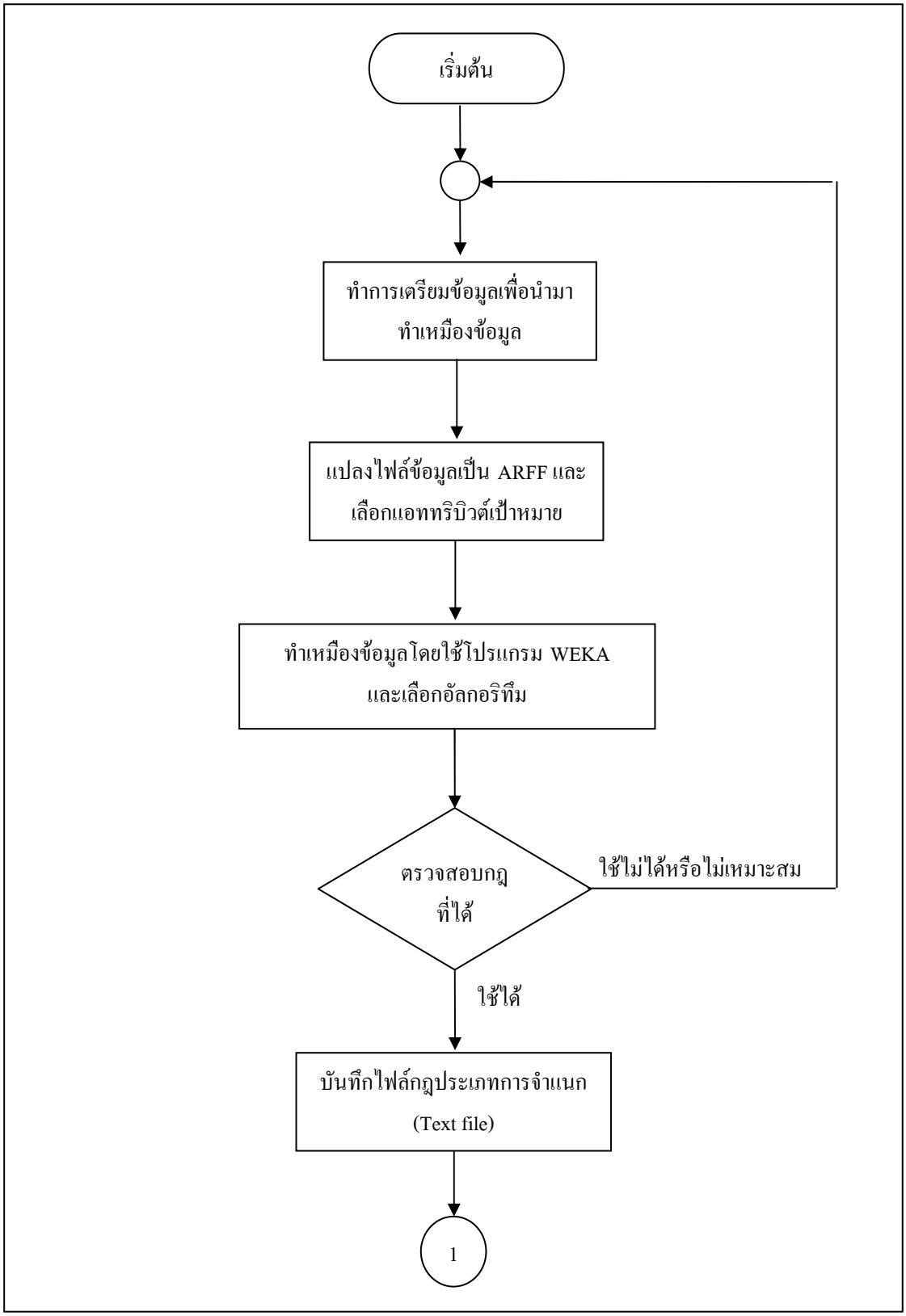
3.2 กระบวนการทำงานของระบบสร้างดาต้าเบสทริกเกอร์

ในกระบวนการทำงานของระบบแบ่งการทำงานออกเป็น 2 ส่วนหลักด้วยกันคือ กระบวนการค้นหากฎประเภทการจำแนกข้อมูล และกระบวนการแปลงกฎประเภทการจำแนกให้เป็นดาต้าเบสทริกเกอร์ โดยในส่วนของกระบวนการค้นหากฎประเภทการจำแนกข้อมูลนี้จะเริ่มตั้งแต่การเตรียมข้อมูลที่ถูกเก็บไว้ในระบบฐานข้อมูลเชิงสัมพันธ์ ซึ่งตามปกติแล้วการเก็บข้อมูลในระบบฐานข้อมูลเชิงสัมพันธ์จะเก็บในรูปแบบของตารางสองมิติ โดยที่แต่ละตารางจะมีความสัมพันธ์เกี่ยวข้องซึ่งกันและกัน แต่ในกระบวนการทำเหมืองข้อมูล ข้อมูลที่จะนำมาทำเหมืองข้อมูลนี้จะต้องเป็นข้อมูลในลักษณะเพียงตารางเดียว โดยจะทำการคัดเลือกเฉพาะข้อมูลที่มีประโยชน์และมีความน่าสนใจหรืออาจจะเป็นข้อมูลที่ปรากฏอยู่ในฐานข้อมูลบ่อย ๆ ที่ทำให้เกิดรูปแบบเท่านั้นจึงไม่จำเป็นที่จะต้องนำข้อมูลที่เก็บในระบบฐานข้อมูลเชิงสัมพันธ์ทุกตารางและทุกแอททริบิวต์มาทำเหมืองข้อมูลทั้งหมด จากนั้นนำข้อมูลที่เตรียมได้ไปทำเหมืองข้อมูลเพื่อค้นหากฎประเภทการจำแนก โดยจะเลือกใช้อัลกอริทึมที่อยู่ในกลุ่มของ Classification ที่มีรูปแบบที่เหมาะสมกับการนำไปแปลงเป็นดาต้าเบสทริกเกอร์ ซึ่งอยู่ในรูปแบบลักษณะคอลัมน์เหตุไปสู่คอลัมน์ผล (IF-THEN) ซึ่งกฎประเภทการจำแนกดังกล่าวจะถูกเก็บไว้ในลักษณะของเท็กซ์ไฟล์ (*.txt) ส่วนกระบวนการแปลงกฎประเภทการจำแนกให้เป็นดาต้าเบสทริกเกอร์นั้น จะใช้กฎ

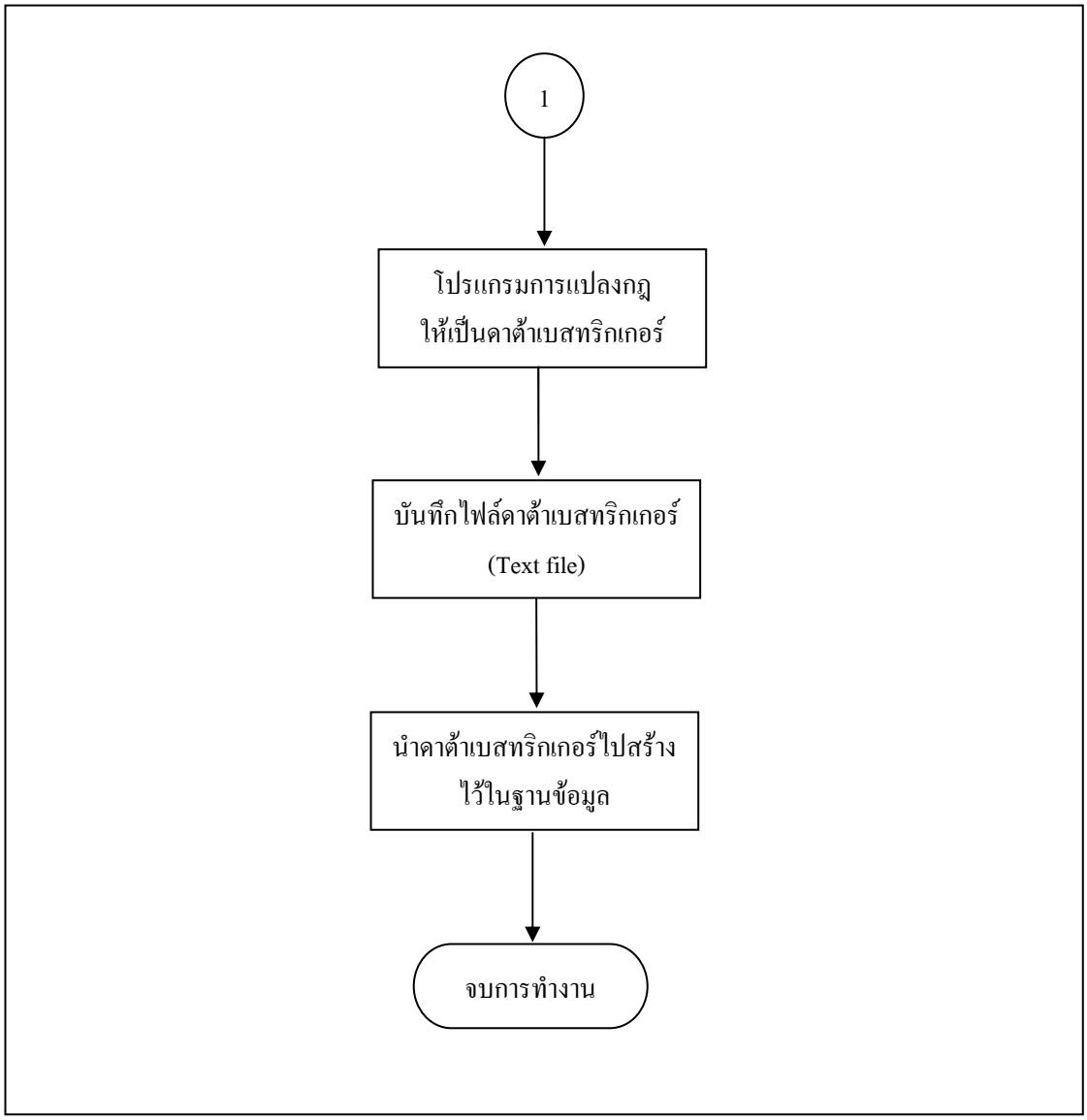
ประเภทการจำแนกที่เป็นเท็กซ์ไฟล์นำมาเป็นอินพุทป้อนเข้าสู่โปรแกรมที่ถูกสร้างขึ้นเพื่อแปลงกฎการจำแนกดังกล่าวให้เป็นดาต้าเบสทริกเกอร์ โดยผลลัพธ์ที่เป็นดาต้าเบสทริกเกอร์นี้จะอยู่ในลักษณะของเท็กซ์ไฟล์เช่นเดียวกันกับอินพุท โดยภาพรวมการทำงานของโปรแกรมสามารถแสดงได้ดังรูปที่ 3.1 และมีขั้นตอนการทำงานของโปรแกรม แสดงได้ดังรูปที่ 3.2



รูปที่ 3.1 ภาพรวมการทำงานของระบบ



รูปที่ 3.2 ขั้นตอนการทำงานของโปรแกรม



รูปที่ 3.2 ขั้นตอนการทำงานของโปรแกรม (ต่อ)

3.3 วิธีการค้นหากฎการจำแนกด้วยโปรแกรม WEKA

โปรแกรมที่ใช้ในการค้นหากฎการจำแนกเพื่อใช้สร้างกฎข้อบังคับความถูกต้องของข้อมูลในฐานข้อมูลหรือดาต้าเบสทริกเกอร์คือ โปรแกรม WEKA (Witten and Frank, 2005) ในการนำข้อมูลมาค้นหากฎการจำแนกจากโปรแกรม WEKA นี้ จะต้องมีการแปลงรูปแบบข้อมูลให้อยู่ในรูปแบบที่โปรแกรมกำหนดไว้ ซึ่งอยู่ในรูปแบบของ Flat file เป็นไฟล์ที่มีลักษณะเป็นข้อความ (Text file) เรียกว่า ARFF (Attribute-Relation File Format) ซึ่งประกอบด้วยข้อมูลสองส่วน คือส่วนอธิบายข้อมูลและส่วนที่เป็นข้อมูล ดังตัวอย่างในรูปที่ 3.3

```
@relation customer_credit

@attribute problematic_area      {yes, no}
@attribute married               {yes, no}
@attribute employed_client       {yes, no}
@attribute credit_approved       {yes, no}

@data
no, yes, yes, yes
no, no, yes, yes
yes, no, yes, yes
no, yes, no, yes
no, yes, no, yes
yes, no, yes, yes
yes, no, no, no
yes, no, no, no
```

รูปที่ 3.3 แสดงรูปแบบไฟล์ข้อมูล ARFF (Attribute-Relation File Format)

ในการสร้างไฟล์ข้อมูลสำหรับใช้งานร่วมกับโปรแกรม WEKA นี้ จะต้องประกอบไปด้วยส่วนประกอบทั้งหมด 3 ส่วน ด้วยกันดังนี้

ส่วนที่ 1 ส่วนของชื่อไฟล์

@relation customer_credit

ส่วนที่ 2 เป็นส่วนการประกาศชื่อและชนิดแอททริบิวต์ของข้อมูล ซึ่งชนิดข้อมูลนี้สามารถแบ่งอย่างกว้าง ๆ ออกเป็น 2 ชนิดด้วยกัน คือ ข้อมูลชนิดตัวเลข (Numeric) และข้อมูลแบบแจกแจง (Categorical) ซึ่งข้อมูลแบบแจกแจงจะต้องมีการบอกค่าที่เป็นไปได้ทั้งหมดในแอททริบิวต์นั้น

@attribute problematic_area {yes, no}

@attribute married {yes, no}

@attribute employed_client {yes, no}

@attribute credit_approved {yes, no}

ส่วนที่ 3 เป็นส่วนของข้อมูล ซึ่งแต่ละแอททริบิวต์ในข้อมูลจะคั่นด้วยเครื่องหมายจุลภาค ‘,’ และข้อมูลหนึ่งแถวจะหมายถึง หนึ่งทูพเพิล หรือ หนึ่งเรคคอร์ด

@data

no, yes, yes, yes

no, no, yes, yes

yes, no, yes, yes

no, yes, no, yes

no, yes, no, yes

yes, no, yes, yes

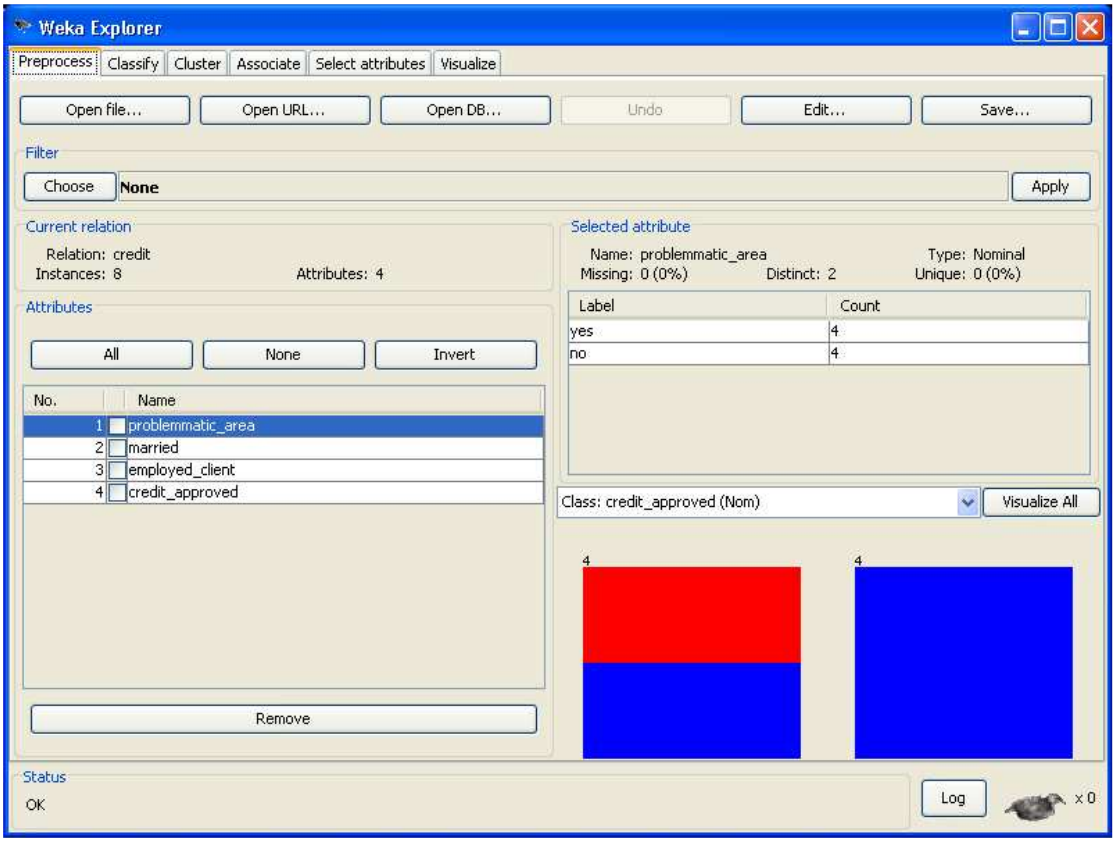
yes, no, no, no

yes, no, no, no

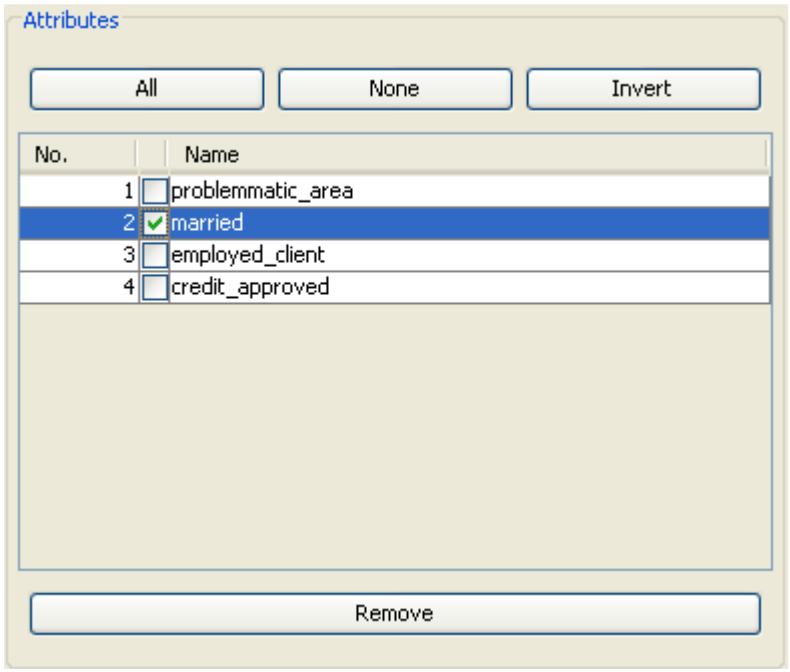
ขั้นตอนในการค้นหากฎการจำแนกจากข้อมูลด้วยโปรแกรม WEKA ประกอบด้วยขั้นตอนทั้งหมด 2 ขั้นตอน ดังนี้

1. การเตรียมข้อมูลก่อนทำการค้นหากฎ

ทำการเปิดไฟล์ข้อมูลที่อยู่ในรูปแบบของ ARFF เพื่อนำข้อมูลที่ต้องการมาทำการค้นหากฎ (รูปที่ 3.4) ในส่วนนี้ เราสามารถทำการตัดคอลัมน์ที่ไม่ต้องการออกจากการค้นหากฎ โดยคลิกเมาส์เพื่อทำการเช็คเครื่องหมายถูกยังบริเวณ Check box ของคอลัมน์นั้น ๆ จากนั้นทำการเลือกที่ปุ่ม Remove (รูปที่ 3.5) เพื่อทำการตัดคอลัมน์นั้นออกจากการค้นหากฎ



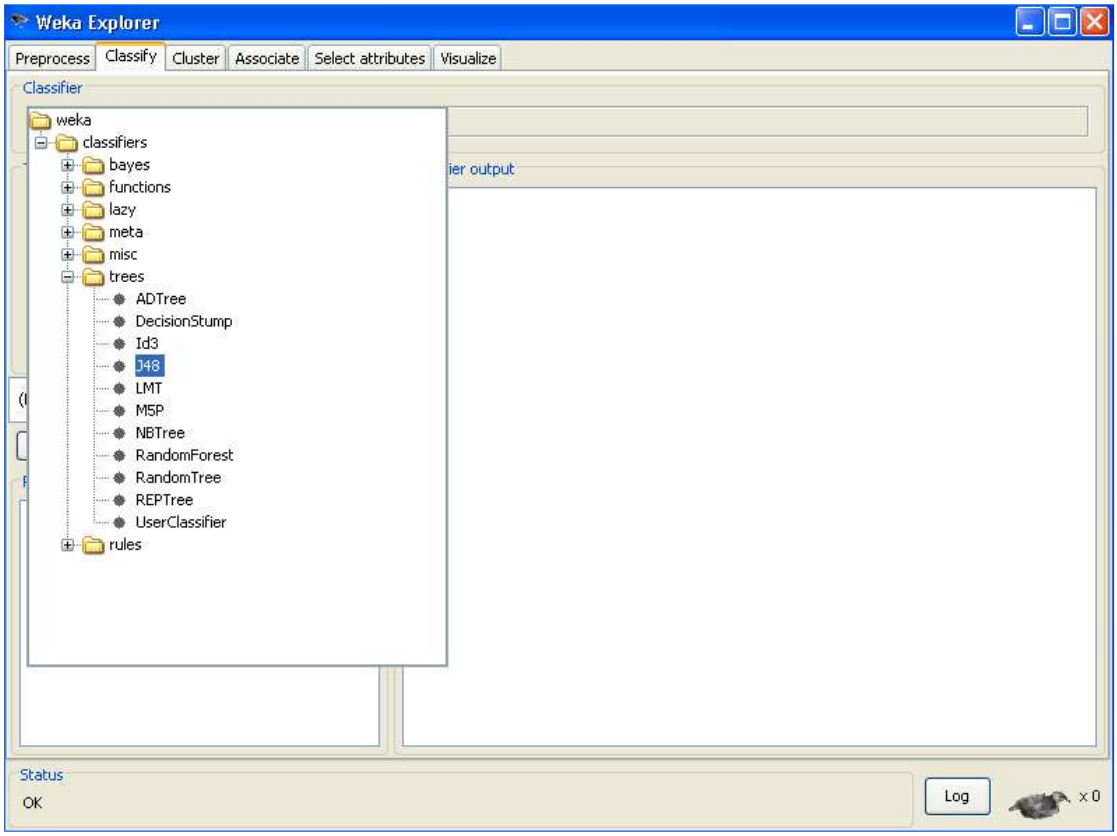
รูปที่ 3.4 แสดงหน้าต่างการนำเข้าและเตรียมข้อมูลสำหรับการค้นหากฎการจำแนก



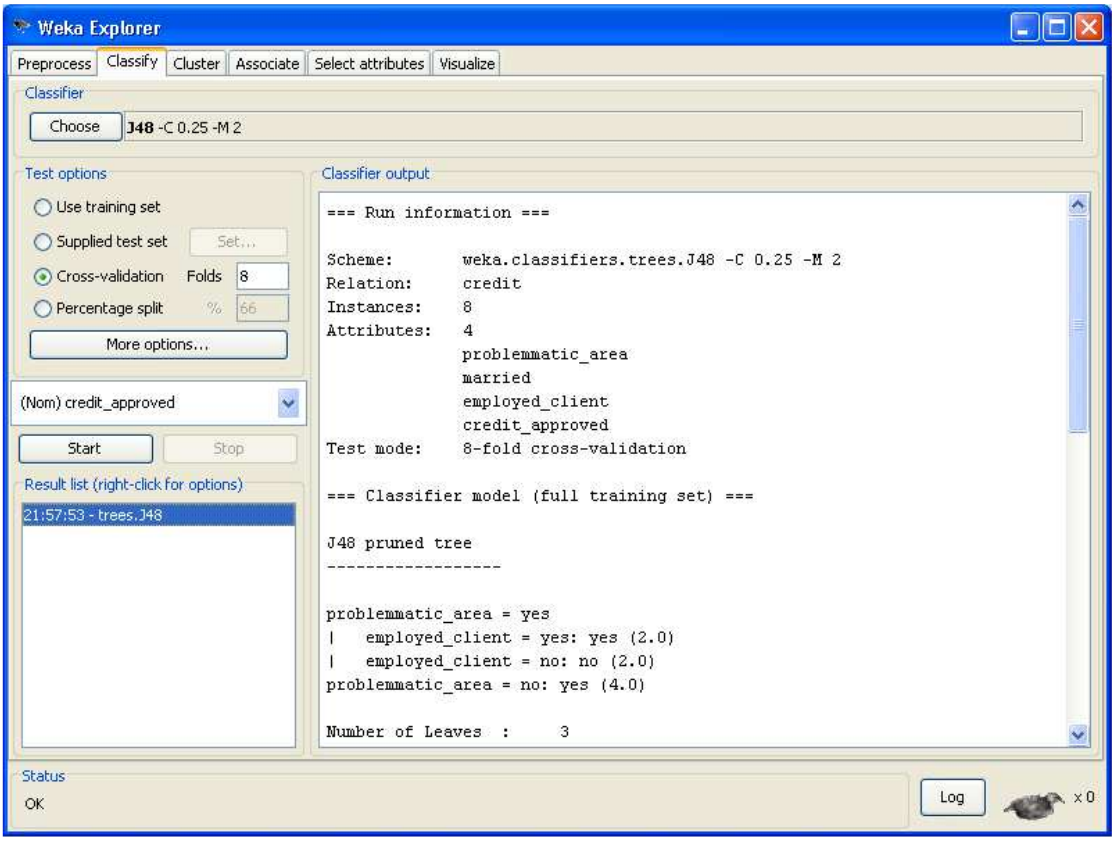
รูปที่ 3.5 แสดงการเลือกคอลัมน์ในกรณีไม่ต้องการคอลัมน์นั้นในการค้นหากฎ

2. การค้นหากฎการจำแนกจากข้อมูลที่ต้องการ

ในการค้นหากฎการจำแนกด้วยโปรแกรม WEKA นี้ จะต้องทำการเลือกอัลกอริทึมที่ใช้ในการค้นหากฎ โดยการเลือกที่ปุ่ม Choose (รูปที่ 3.6) ซึ่งในการค้นหากฎการจำแนกในงานวิจัยนี้จะใช้อัลกอริทึม J48 ซึ่งเป็นอัลกอริทึมในกลุ่ม Classification หรือ Classifiers และในกลุ่มย่อย trees เมื่อทำการกำหนดค่าต่าง ๆ เรียบร้อยแล้วจึงคลิกที่ปุ่ม Start เพื่อทำการประมวลผลข้อมูลเพื่อค้นหากฎการจำแนกข้อมูลนั้นออกมา แสดงได้ดังรูปที่ 3.7



รูปที่ 3.6 แสดงหน้าต่างการเลือกอัลกอริทึม J48 เพื่อใช้ค้นหากฎการจำแนก



รูปที่ 3.7 แสดงหน้าต่างผลลัพธ์ของการค้นหากฎการจำแนกข้อมูล

กฎการจำแนกที่ได้ดังรูปที่ 3.7 จะถูกนำไปแปลงให้เป็นดาต้าเบสทริกเกอร์ โดยจะได้อธิบายรายละเอียดในหัวข้อที่ 3.4 ในการทดสอบดาต้าเบสทริกเกอร์ที่ได้จากการแปลงกฎประเภทการจำแนกข้อมูลแล้ว จะนำไปทำการทดสอบกับระบบฐานข้อมูล Microsoft SQL Server 2000 ในการเปลี่ยนแปลง แก้ไขและเพิ่มข้อมูลลงไปในระบบฐานข้อมูล

3.4 อัลกอริทึมการแปลงกฎการจำแนกเป็นดาต้าเบสทริกเกอร์

เนื้อหาในส่วนนี้กล่าวถึงการนำแนวคิดต่าง ๆ และอัลกอริทึมที่ได้ออกแบบไว้ มาพัฒนาเพื่อสร้างกฎข้อบังคับหรือดาต้าเบสทริกเกอร์ โดยเป็นการนำเอาความรู้เดิมที่มีอยู่แล้วมาประยุกต์ใช้เข้าด้วยกัน นั่นคือ อัลกอริทึมประเภทการจำแนก (Classification) ซึ่งเป็นอัลกอริทึมหนึ่งในการทำเหมืองข้อมูลมาประยุกต์ใช้ให้ทำงานร่วมกันกับระบบฐานข้อมูลเชิงสัมพันธ์ เพื่อเพิ่มประสิทธิภาพด้านความถูกต้องอย่างสูงสุดให้กับระบบฐานข้อมูล

3.4.1 การค้นหากฎประเภทการจำแนกข้อมูล

ในการเก็บข้อมูลลงในฐานข้อมูลนั้นมีความเป็นไปได้ที่ข้อมูลที่ถูกเก็บลงไปจะอยู่ในลักษณะที่ซ้ำ ๆ กันหรือคล้ายคลึงกัน ทำให้ข้อมูลที่ถูกเก็บไว้เกิดรูปแบบ (Pattern) งานวิจัยชิ้นนี้จึงได้นำความรู้ดังกล่าวมาประยุกต์ใช้ เพื่อให้ข้อมูลที่ถูกเก็บสะสมไว้เป็นจำนวนมากเกิดรูปแบบที่ถูกต้อง หรืออาจกล่าวได้อีกอย่างหนึ่งคือเพิ่มความถูกต้องให้กับฐานข้อมูลด้วยข้อมูลที่ถูกเก็บอยู่เอง ซึ่งการค้นหากฎประเภทการจำแนกข้อมูลนี้ ผลลัพธ์ที่ได้ออกมาจะอยู่ในรูปแบบของคอลัมน์เหตุไปสู่คอลัมน์ผล ตามรูปแบบของการค้นหาด้วยอัลกอริทึมประเภทการจำแนก ดังนี้

If column_A1 = ‘value_A1’ Then column_Z = ‘value_Z’

จากที่กล่าวนำไว้ในขั้นต้นว่าระบบฐานข้อมูลเชิงสัมพันธ์นั้นจะเก็บข้อมูลในลักษณะตารางสองมิติซึ่งแต่ละตารางจะมีความสัมพันธ์ซึ่งกันและกัน ดังนั้นก่อนที่จะนำข้อมูลมาทำเหมืองข้อมูลได้จึงต้องทำการจัดเตรียมข้อมูล (Data preparation) ซึ่งตารางที่ 3.1 และ 3.2 เป็นลักษณะของตารางข้อมูลที่ยังไม่ได้ผ่านขั้นตอนการจัดเตรียมข้อมูล ซึ่งแสดงรายละเอียดข้อมูลลูกค้าธนาคารและข้อมูลการอนุมัติบัตรเครดิต ตามลำดับดังนี้

ตารางที่ 3.1 ตัวอย่างข้อมูลลูกค้าธนาคาร (TBCustomer)

Customer_ID	Name	Problematic_area	Married	Employed_clients
A01	AA	No	Yes	Yes
A02	BB	No	No	Yes
A03	CC	Yes	No	Yes
A04	DD	No	Yes	No
A05	EE	No	Yes	No
A06	FF	Yes	No	Yes
A07	GG	Yes	No	No
A08	HH	Yes	No	No

ตารางที่ 3.2 ตัวอย่างข้อมูลการอนุมัติบัตรเครดิต (TBCredit)

Customer_ID	Date	Credit_Approved
A01	01/04/2008	Yes
A02	03/04/2008	Yes
A03	03/04/2008	Yes
A04	19/05/2008	Yes
A05	02/06/2008	Yes
A06	04/06/2008	Yes
A07	07/06/2008	No
A08	13/06/2008	No

จากข้อมูลทั้งสองตารางดังกล่าว นำมารวมกันเป็นหนึ่งตารางเพื่อรูปแบบที่เหมาะสมกับการทำเหมืองข้อมูล โดยใช้คำสั่ง JOIN ON สำหรับการเชื่อมตารางเข้าด้วยกัน ซึ่งเป็นคำสั่งพื้นฐานของภาษา SQL (Microsoft SQL Server Ver.2000) โดยมีรูปแบบคำสั่งดังรูปที่ 3.8

```
SELECT Problematic_area, Married, Employed_clients, Client_approved
INTO     Customer_Credit
FROM    TBCustomer t1 JOIN TBCredit t2
ON         t1.Customer_ID = t2.Customer_ID
```

รูปที่ 3.8 แสดงรูปแบบคำสั่งการเชื่อมตาราง

พร้อมกันนี้คัดเอาไว้เฉพาะคอลัมน์ที่มีประโยชน์และเหมาะสมที่จะนำมาหารูปแบบนั้นคือคอลัมน์ที่มีข้อมูลซ้ำ ๆ กันหรือเกิดขึ้นหลาย ๆ ครั้งในแต่ละเรคคอร์ด ดังจะได้ตารางที่มีโครงสร้างข้อมูลใหม่ดังนี้

Customer_Credit(Problematic_area, Married, Employed_clients, Credit_approved)

จากกฎประเภทการจำแนกข้อมูลที่ได้จากการทำเหมืองข้อมูล คอลัมน์ที่เป็นเหตุอาจจะมีมากว่าหนึ่งคอลัมน์ และเนื่องจากกฎประเภทการจำแนกจะมีการเชื่อมค่าของแต่ละคอลัมน์ด้วย (AND) ซึ่งจะสอดคล้องมีความสัมพันธ์กันกับการเปลี่ยนแปลงข้อมูลในฐานข้อมูลที่อาจจะทำการเปลี่ยนแปลงหรือแก้ไขข้อมูลครั้งละจำนวนมากกว่า 1 คอลัมน์ เช่นถ้ามีข้อมูลดังแสดงในตารางที่ 3.3 ซึ่งเป็นข้อมูลตัวอย่าง ประกอบไปด้วยจำนวนคอลัมน์ทั้งหมด 4 คอลัมน์ด้วยกัน คือ ข้อมูลที่อยู่อาศัยที่มีปัญหา (Problematic area), ข้อมูลสถานะการสมรส (Married), ข้อมูลการมีงานทำ (Employed clients) และข้อมูลการอนุมัติบัตรเครดิต (Credit Approved) โดยความหมายของข้อมูลในแต่ละคอลัมน์คือ ใช่ (yes) และ ไม่ใช่ (no)

ตารางที่ 3.3 ตัวอย่างข้อมูลที่ใช้ในการหากฎประเภทการจำแนก

Problematic_area	**Married**	**Employed_clients**	**Credit_Approved**
No	Yes	Yes	Yes
No	No	Yes	Yes
Yes	No	Yes	Yes
No	Yes	No	Yes
No	Yes	No	Yes
Yes	No	Yes	Yes
Yes	No	No	No
Yes	No	No	No

จากข้อมูลข้างต้นเมื่อนำมาค้นหากฎประเภทการจำแนกโดยใช้โปรแกรมระบบ WEKA อัลกอริทึม J48 ซึ่งเป็นอัลกอริทึมในกลุ่ม Classification จะได้ผลลัพธ์ในลักษณะของ Decision tree ดังรูปที่ 3.9 ซึ่ง Tree ที่ได้สามารถแปลความให้อยู่ในลักษณะของกฎได้ โดยตัวเลขที่อยู่ในวงเล็บที่ปรากฏอยู่ในกฎแสดงถึงจำนวนข้อมูลที่ถูกจำแนกเป็นแต่ละประเภทต่าง ๆ

```
problematic_area = yes
    |   employed_client = yes: yes (2.0)
    |   employed_client = no: no (2.0)
problematic_area = no: yes (4.0)
```

รูปที่ 3.9 แสดงผลการค้นหากฎประเภทการจำแนก

เมื่อได้กฎประเภทการจำแนกข้อมูลตามที่ต้องการแล้วจะทำการจัดเก็บกฎประเภทการจำแนกข้อมูลในรูปที่ 3.9 ดังกล่าวลงในเท็กซ์ไฟล์ เพื่อนำไปแปลงเป็นกฎควบคุมความถูกต้องของฐานข้อมูลหรือดาต้าเบสทริกเกอร์

เพื่อความเหมาะสมและง่ายต่อการเข้าใจในการอ่านกฎจึงมีการจัดรูปแบบให้อยู่ในลักษณะดังนี้

If column_A1 = ‘value_A1’ AND column_A2 = ‘value_A2’ AND…….

Then column_Z = ‘value_Z’

ดังนั้นเมื่อวิเคราะห์กฎประเภทการจำแนกข้อมูลที่ได้จากการทำเหมืองข้อมูลในรูปที่ 3.9 สามารถนำเอาคอลัมน์ในส่วนของเหตุมาเชื่อมกันโดยใช้ตัวเชื่อม AND โดยจะได้กฎทั้งหมด 3 ข้อ ดังรูปที่ 3.10 ดังนี้

1. IF problematic_area = yes AND employed_clients = yes THEN credit_approved = yes
2. IF problematic_area = yes AND employed_clients = no THEN credit_approved = no
3. IF problematic_area = no THEN credit_approved = yes

รูปที่ 3.10 แสดงการตีความหมายของกฎประเภทการจำแนกข้อมูล

3.4.2 อัลกอริทึมการแปลงกฎประเภทการจำแนกเป็นดาต้าเบสทริกเกอร์

ดาต้าเบสทริกเกอร์เป็นกฎข้อบังคับของการเปลี่ยนแปลงข้อมูลในฐานข้อมูล ซึ่งทำหน้าที่ตรวจสอบกฎข้อบังคับของข้อมูลในการประมวลผลคำสั่ง SQL (Structured Query Language) ประเภท DML (Data Manipulation Language) ให้มีความถูกต้องตามความต้องการของระบบงาน ซึ่งคำสั่งSQL ประเภท DML ประกอบไปด้วยคำสั่ง INSERT, UPDATE และ DELETE คือการเพิ่ม การแก้ไขและการลบข้อมูล ตามลำดับ คำสั่งเหล่านี้เป็นคำสั่งที่ใช้ในการเปลี่ยนแปลงข้อมูล (Transaction)ในฐานข้อมูล ในหัวข้อนี้จะนำเสนออัลกอริทึมการแปลงกฎประเภทการจำแนกให้เป็นดาต้าเบสทริกเกอร์ ซึ่งอินพุทของอัลกอริทึมดังกล่าวนี้จะเป็นกฎในลักษณะของเท็กซ์ไฟล์ ซึ่งเป็นผลลัพธ์ที่ได้จากการจำแนกข้อมูลที่ได้จากโปรแกรมระบบ WEKA ดังรูปที่ 3.9 เข้ามาในโปรแกรมเพื่อแปลงให้อยู่ในรูปแบบของดาต้าเบสทริกเกอร์ ดังอัลกอริทึมที่ปรากฏอยู่ในรูปที่ 3.11 และจะได้ผลลัพธ์ออกมาเป็นดาต้าเบสทริกเกอร์ในรูปแบบของเท็กซ์ไฟล์

(1) อ่านข้อความกฎเข้ามาทีละบรรทัด
(2) ถ้าไม่เริ่มต้นด้วยเครื่องหมาย |
    (2.1) ถ้ามีเครื่องหมาย :
        (2.1.1) แบ่งข้อความออกเป็นสองส่วนคือ ข้อความก่อนและหลังเครื่องหมาย : โดยแบ่งเก็บไว้ในตัวแปร
        (2.1.2) ทำการแปลงข้อความในส่วนหลังเครื่องหมาย :
        (2.1.3) เชื่อมข้อความทั้งหมดเข้าด้วยกันโดยเก็บไว้ในตัวแปร
        (2.1.4) เพิ่มค่าตัวแปรนับจำนวนกฎ
    (2.2) ถ้าไม่มีเครื่องหมาย :
        เก็บข้อความกฎไว้ในตัวแปร
(3) ถ้าเริ่มต้นด้วยเครื่องหมาย |
    (3.1) หาตำแหน่งของเครื่องหมาย | ตัวสุดท้าย โดยเลือกส่วนท้ายของข้อความมาเก็บไว้ในตัวแปร
    (3.2) ถ้ามีเครื่องหมาย :
        (3.2.1) แบ่งข้อความออกเป็นสองส่วนคือ ข้อความก่อนและหลังเครื่องหมาย : โดยแบ่งเก็บไว้ในตัวแปร
        (3.2.2) ทำการแปลงข้อความในส่วนหลังเครื่องหมาย :
        (3.2.3) เชื่อมข้อความทั้งหมดเข้าด้วยกันโดยเก็บไว้ในตัวแปร
        (3.2.4) เพิ่มค่าตัวแปรนับจำนวนกฎ
    (3.3) ถ้าไม่มีเครื่องหมาย :
        เก็บข้อความกฎไว้ในตัวแปร

ขั้นตอนการแสดงผลลัพธ์

(1) วนลูปจำนวนรอบเท่ากับตัวแปรนับจำนวนกฎ
    (1.1) จัดรูปแบบข้อความภาษา SQL ในกรณีที่ข้อมูลเป็นชนิดของข้อความ
    (1.2) สร้างข้อความการสร้างดาต้าเบสทริกเกอร์โดยเก็บไว้ในตัวแปร โดยแทรกข้อความกฎที่ได้จากผลข้างต้น
(2) เขียนข้อความผลลัพธ์ที่ได้ลงบนไฟล์

รูปที่ 3.11 แสดงอัลกอริทึมการสร้างดาต้าเบสทริกเกอร์

ตัวอย่างที่ 1 ให้ข้อความกฎข้อแรกที่รับเข้ามามีลักษณะดังนี้

```
problematic_area = yes
    |   employed_client = yes: yes (2.0)
```

รูปที่ 3.12 แสดงข้อความกฎข้อแรกที่รับเข้ามาในโปรแกรม

เมื่อนำเข้าโปรแกรมจะทำให้ได้ผลลัพธ์ดังรูปที่ 3.13 โดยในส่วนของข้อความกฎจะถูกแปลงและปรากฏอยู่ในส่วนของ Where clause ในคำสั่ง SQL ในประโยคของคำสั่งการสร้างดาต้าเบสทริกเกอร์ จากข้อความกฎในส่วนที่เป็นเครื่องหมาย | จะถูกแทนด้วยคำว่า and ซึ่งเป็นคำที่เชื่อมประโยคเงื่อนไขในการจำแนกประเภทข้อมูล และเครื่องหมาย : จะใช้เป็นส่วนที่ใช้จำแนกประเภทข้อมูล โดยจากตัวอย่างในรูปที่ 3.12 มีเงื่อนไขคือ ถ้าลูกค้ามีพื้นที่อยู่อาศัยอยู่ในเขตที่มีปัญหา problematic_area = yes และลูกค้ามีงานทำ employed_client = yes จะสามารถจำแนกประเภทได้ว่าลูกค้าคนนี้จะได้รับการอนุมัติบัตรเครดิต credit_approved = yes

```
CREATE TRIGGER rule_1
ON Customer_Credit
FOR UPDATE, INSERT
AS
   IF (SELECT COUNT(*)
       FROM Customer_credit
       WHERE (problematic_area = 'yes')
                   and (employed_client = 'yes')
                   and (credit_approved  <> 'yes'))> 0
BEGIN
   ROLLBACK TRAN
   RAISERROR ('Credit Approved Error', 16, 10)
END;
```

รูปที่ 3.13 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_1 ที่สร้างจากกฎข้อแรก

ตัวอย่างที่ 2 ให้ข้อความกฎข้อที่สองที่รับเข้ามามีลักษณะดังนี้

```
problematic_area = yes
        |   employed_client = no: no (2.0)
```

รูปที่ 3.14 แสดงข้อความกฎข้อที่สองที่รับเข้ามาในโปรแกรม

เมื่อนำเข้าโปรแกรมจะทำให้ได้ผลลัพธ์ดังรูปที่ 3.15 โดยในส่วนของข้อความกฎจะถูกแปลงและปรากฏอยู่ในส่วนของ Where clause ในคำสั่ง SQL ในประโยคของคำสั่งการสร้างดาต้าเบสทริกเกอร์ โดยจากตัวอย่างในรูปที่ 3.14 มีเงื่อนไขคือ ถ้าลูกค้ามีพื้นที่อยู่อาศัยอยู่ในเขตที่มีปัญหา problematic_area = yes และลูกค้าไม่มีงานทำ employed_client = no จะสามารถจำแนกประเภทได้ว่าลูกค้าคนนี้จะไม่ได้รับการอนุมัติบัตรเครดิต credit_approved = no

```
CREATE TRIGGER rule_2
ON Customer_Credit
FOR UPDATE, INSERT
AS
   IF (SELECT COUNT(*)
        FROM Customer_credit
        WHERE (problematic_area = 'yes')
                    and (employed_client = 'no')
                    and (credit_approved  <> 'no'))> 0
BEGIN
   ROLLBACK TRAN
   RAISERROR ('Credit Approved Error')
END;
```

รูปที่ 3.15 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_2 ที่สร้างจากกฎข้อที่สอง

ตัวอย่างที่ 3 ให้ข้อความกฎข้อที่สามที่รับเข้ามามีลักษณะดังนี้

```
problematic_area = no: yes (4.0)
```

รูปที่ 3.16 แสดงข้อความกฎที่รับเข้ามาในโปรแกรม

เมื่อนำเข้าโปรแกรมจะทำให้ได้ผลลัพธ์ดังรูปที่ 3.17 โดยในส่วนของข้อความกฎจะถูกแปลงและปรากฏอยู่ในส่วนของ Where clause ในคำสั่ง SQL ในประโยคของคำสั่งการสร้างดาต้าเบสทริกเกอร์ โดยในตัวอย่างนี้จะไม่มีเครื่องหมาย | ที่ใช้ในการเชื่อมเงื่อนไข เพราะเนื่องจากมีเพียงหนึ่งเงื่อนไขในการจำแนกประเภทข้อมูลที่แสดงถึงเงื่อนไขการอนุมัติบัตรเครดิตที่เป็น ‘yes’ ในกรณีที่ลูกค้าไม่มีพื้นที่อยู่อาศัยในเขตที่มีปัญหา problematic_area = no

```
CREATE TRIGGER rule_3
ON Customer_Credit
FOR UPDATE, INSERT
AS
   IF (SELECT COUNT(*)
       FROM Customer_credit
       WHERE (problematic_area = ‘no’)
                  and (credit_approved  <> ‘yes’))> 0
BEGIN
   ROLLBACK TRAN
   RAISERROR ('Credit Approved Error')
END;
```

รูปที่ 3.17 แสดงดาต้าเบสทริกเกอร์ชื่อ rule_3 ที่สร้างจากกฎข้อที่สาม

บทที่ 4
การทดสอบและอภิปรายผล

ในการทดสอบผลของการสร้างกฎบังคับของฐานข้อมูลหรือดาต้าเบสทริกเกอร์ที่ได้จากการทำเหมืองข้อมูลด้วยอัลกอริทึมประเภทการจำแนกข้อมูล (Classification) จะแบ่งการทดสอบออกเป็นสองส่วนหลัก ๆ ด้วยกัน คือส่วนของกระบวนการค้นหากฎการจำแนกข้อมูล และกระบวนการแปลงกฎการจำแนกให้เป็นดาต้าเบสทริกเกอร์ เพื่อนำไปทดลองใช้ในการสร้างดาต้าเบสทริกเกอร์จริงในฐานข้อมูลและนำไปใช้งาน โดยในส่วนแรกจะเป็นการค้นหากฎการจำแนกข้อมูลโดยใช้โปรแกรม WEKA จากนั้นนำกฎที่ได้มาสร้างเป็นดาต้าเบสทริกเกอร์เพื่อตรวจสอบความถูกต้องในกรณีที่มีการเปลี่ยนแปลงข้อมูลในฐานข้อมูลซึ่งเป็นส่วนที่สองของการทดสอบผล ในการทดสอบใช้ Microsoft SQL Server 2000 มาเป็นระบบจัดการฐานข้อมูล โดยข้อมูลที่ใช้ในการทดสอบเป็นฐานข้อมูลของผู้ป่วยที่มีความเสี่ยงเป็นโรคเบาหวานและปริมาณการให้อินซูลินเพื่อรักษาระดับน้ำตาลในเลือด คอมพิวเตอร์ที่ใช้ในการทดสอบเป็นคอมพิวเตอร์ CPU Intel Celeron M Processor 440 ความเร็ว 1.86 GHz หน่วยความจำหลัก 512 MB ฮาร์ดดิสก์ความจุ 80 GB โดยรายละเอียดของข้อมูลโรคเบาหวาน ปรากฏรายละเอียดในหัวข้อ 4.1 ผลของกระบวนการค้นหากฎการจำแนกข้อมูล ปรากฏรายละเอียดในหัวข้อ 4.2 ผลของกระบวนการแปลงกฎการจำแนกเป็นดาต้าเบสทริกเกอร์โดยใช้โปรแกรมที่สร้างขึ้น ปรากฏรายละเอียดในหัวข้อ 4.3 ผลการสร้างดาต้าเบสทริกเกอร์ในระบบจัดการฐานข้อมูล Microsoft SQL Server 2000 และการนำไปใช้งานจริง ปรากฏรายละเอียดในหัวข้อ 4.4 และหัวข้อ 4.5 เป็นการอภิปรายสรุป

4.1 ข้อมูลที่ใช้ในการทดลอง

เนื้อหาในส่วนนี้กล่าวถึงรายละเอียดข้อมูลโรคเบาหวาน ข้อมูลผู้ป่วยที่มีภาวะเสี่ยงต่อการเป็นโรคเบาหวานรวมถึงการรักษาโรคเบาหวานหรือการรักษาระดับน้ำตาลในเลือดด้วยการให้อินซูลินปริมาณที่เพียงพอ ซึ่งในงานวิจัยชิ้นนี้ได้ใช้ข้อมูลโรคเบาหวานนี้เป็นข้อมูลในการทดลองงานวิจัย

4.1.1 โรคเบาหวาน

เบาหวานคือ ภาวะที่ร่างกายมีระดับน้ำตาลในเลือดสูงกว่าปกติ เกิดขึ้นเมื่อตับอ่อนไม่สามารถผลิตอินซูลินได้เพียงพอหรือสร้างไม่ได้เลย หรือสร้างอินซูลินได้แต่ออกฤทธิ์ไม่ดีเท่าที่ควร ทำให้การเผาผลาญอาหารประเภทแป้งและน้ำตาลน้อยลงทำให้ร่างกายไม่สามารนำน้ำตาลมาใช้ได้อย่างเหมาะสม ซึ่งจะทำให้น้ำตาลค้างในเลือดมาก จึงล้นออกมาในปัสสาวะเกิดอาการที่เรียกว่าเบาหวาน (สมาคมโรคเบาหวานแห่งประเทศไทยในพระราชูปถัมภ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี, www, 2007) โรคแทรกซ้อนของผู้ที่เป็นโรคเบาหวานเนื่องจากภาวะน้ำตาลที่สูงในเลือดเป็นเวลานาน ๆ จะทำให้อวัยวะต่าง ๆ ของร่างกายเกิดโรคแทรกซ้อนได้ ซึ่งโรคเบาหวานมีด้วยกัน 2 ชนิดคือ

(1) เบาหวานชนิดต้องพึ่งอินซูลิน

เบาหวานชนิดนี้ส่วนใหญ่พบในเด็กหรือวัยรุ่น และคนรูปร่างผอม ต้องรักษาด้วยการฉีดอินซูลินไปจนตลอดชีวิต

(2) เบาหวานชนิดไม่ต้องพึ่งอินซูลิน

เบาหวานชนิดนี้พบมากกว่าชนิดแรก และมักพบในผู้ใหญ่อายุมากกว่า 40 ปี ผู้ที่เป็นเบาหวานชนิดนี้มักอ้วนและอยู่ในสภาวะโรคอ้วน สามารถรักษาด้วยการควบคุมอาหารร่วมกันกับการรับประทานยาลดปริมาณน้ำตาล แต่บางครั้งหากรับประทานยาไม่ได้ผลก็จำเป็นต้องฉีดอินซูลินเหมือนกัน

ปัจจัยที่ทำให้เกิดโรคเบาหวานมีจากหลายสาเหตุ เช่น กรรมพันธุ์ ความอ้วนซึ่งทำให้อินซูลินออกฤทธิ์ได้น้อยลง การใช้ยาขับปัสสาวะ การใช้ยาสเตียรอยด์ซึ่งทำให้ระดับน้ำตาลในเลือดสูง การเป็นโรคของตับอ่อนหรือการตั้งครรภ์เป็นต้น

ในงานวิจัยชิ้นนี้ได้เลือกใช้ปัจจัยการเกิดโรคเบาหวานโดยกรรมพันธุ์และความอ้วนมาใช้เป็นข้อมูลในการทดลอง เนื่องจากการที่มีบรรพบุรุษหรือญาติที่ใกล้ชิดเป็นโรคเบาหวานก็จะทำให้ผู้ป่วยอยู่ในภาวะเสี่ยงที่จะเป็นโรคเบาหวานด้วย หรือหากผู้ป่วยเป็นโรคอ้วนก็จะทำให้อยู่ในภาวะเสี่ยงเช่นเดียวกัน เนื่องจากเมื่อน้ำหนักตัวสูงขึ้น จะส่งผลทำให้ความดันโลหิตสูงขึ้นตามด้วย และยังเป็นปัจจัยเสี่ยงของโรคหัวใจและเพิ่มโอกาสของภาวะระดับโคเลสเตอรอลในเลือดสูง รวมทั้งโรคเบาหวาน โดยโรคอ้วนจะพิจารณาจากการวัดปริมาณไขมันในร่างกายซึ่งจำเป็นต้องใช้เครื่องมือในการวัด จึงใช้ดัชนีมวลกาย BMI (Body Mass Index) มาวัด ซึ่งค่าที่ได้มีความแม่นยำพอสมควร และมีความสัมพันธ์กับปริมาณไขมันในร่างกาย โดยดัชนีมวลกายมีสูตรการคำนวณ (วิทยาลัยวิทยาศาสตร์และเทคโนโลยีการกีฬา มหาวิทยาลัยมหิดล, www, 2008) ดังรูปที่ 4.1

$$BMI = \frac{Weight(Kg)}{Height^2(m)^2}$$

รูปที่ 4.1 สูตรการคำนวณดัชนีมวลกาย BMI (Body Mass Index)

ค่าดัชนีมวลกายสามารถคำนวณได้โดยใช้น้ำหนักตัวเป็นกิโลกรัมหารด้วยส่วนสูงเป็นเมตรยกกำลังสอง จากนั้นนำค่าที่ได้มาเปรียบเทียบกับค่าในตารางที่ 4.1 เพื่อพิจารณาระดับความอ้วน (วิทยาลัยวิทยาศาสตร์และเทคโนโลยีการกีฬา มหาวิทยาลัยมหิดล, www, 2008)

ตารางที่ 4.1 แสดงความสัมพันธ์ระหว่างดัชนีมวลกายและระดับความอ้วน

กลุ่ม	ดัชนีมวลกาย (BMI)	ผลลัพธ์
น้ำหนักปกติ	18.5 – 24.9	เป็นค่าที่ดี และควรพยายามควบคุมน้ำหนักไม่ให้เพิ่มขึ้น
น้ำหนักเกิน	25 – 29.9	ควรพยายามควบคุมน้ำหนักไม่ให้เพิ่มขึ้นและควรลดน้ำหนักลง
อ้วน	ตั้งแต่ 30 ขึ้นไป	ควรต้องลดน้ำหนัก โดยลดลงอย่างช้า ๆ โดยอาศัยการควบคุมอาหารและออกกำลังกายอย่างเหมาะสมภายใต้คำแนะนำของแพทย์

4.1.2 การรักษาโรคเบาหวานด้วยอินซูลิน

อินซูลินเป็นฮอร์โมนที่สร้างจากตับอ่อน ออกฤทธิ์โดยการนำน้ำตาลจากเลือดเข้าไปในเซลล์ของร่างกายเพื่อใช้เป็นพลังงาน ในผู้ป่วยที่ขาดอินซูลินหรืออินซูลินไม่สามารถออกฤทธิ์ได้ตามปกติหรือที่เรียกว่า ดื้อต่ออินซูลิน ทำให้เซลล์ไม่สามารถนำน้ำตาลไปใช้จึงทำให้ระดับน้ำตาลในเลือดสูงจึงเกิดโรคเบาหวาน

อินซูลินเป็นยาที่จำเป็นในการรักษาโรคเบาหวานชนิดที่หนึ่งทุกราย นอกจากนั้นยังจำเป็นสำหรับผู้ป่วยโรคเบาหวานชนิดที่สองที่ไม่ตอบสนองต่อยาเม็ด ซึ่งการใช้อินซูลินเหมาะสำหรับผู้ป่วยเบาหวานซึ่งอาการเกิดเร็วและน้ำหนักลดลง ผู้ป่วยเบาหวานที่ควรใช้อินซูลินในการรักษาได้แก่

- เบาหวานชนิดที่หนึ่งทุกราย
- เบาหวานขณะตั้งครรภ์
- เบาหวานที่มีโรคตับ หรือไต
- เบาหวานชนิดที่สองที่ใช้ยาเม็ดลดน้ำตาลไม่ได้ผล
- เบาหวานที่มีภาวะเครียด เช่น การติดเชื้อ การผ่าตัด เป็นต้น

อินซูลินแบ่งตามระยะเวลาการออกฤทธิ์ได้เป็น 4 ชนิด (สมาคมโรคเบาหวานแห่งประเทศไทยในพระราชูปถัมภ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี, www, 2007) ดังนี้

1) ออกฤทธิ์เร็ว (Short acting insulin)

เป็นอินซูลินน้ำใส ได้แก่ Regular insulin (Actrapid, Humalin-R) เริ่มออกฤทธิ์ 15-60 นาทีหลังฉีด ยาออกฤทธิ์สูงสุด 2-4 ชั่วโมงหลังฉีด และอยู่ได้นาน 4-6 ชั่วโมงหลังฉีด

2) ออกฤทธิ์ปานกลาง (Intermediate acting insulin)

เป็นอินซูลินชนิดน้ำขุ่นแบ่งออกเป็นสองชนิดคือ NPH (Neutral Protamine Hagedorn) insulin หรืออาจเรียกว่า Isophane insulin ใช้สาร Protamine ทำให้อินซูลินออกฤทธิ์ยาวขึ้นได้แก่ Humalin-N เริ่มออกฤทธิ์ 1-4 ชั่วโมงหลังฉีด ออกฤทธิ์สูงสุด 8-10 ชั่วโมง และยาอยู่ได้นาน 12-20 ชั่วโมง ใช้ฉีดใต้ผิวหนังได้อย่างเดียว ส่วนชนิดที่สองคือ Lente insulin ใช้ Zinc ทำให้ยาออกฤทธิ์นานขึ้น เริ่มออกฤทธิ์ 2-4 ชั่วโมง หลังฉีดออกฤทธิ์สูงสุด 8-12 ชั่วโมง และยาอยู่ได้นาน 12-20 ชั่วโมง

3) ออกฤทธิ์ระยะยาว (Long acting insulin)

ได้แก่ Ultralente insulin ออกฤทธิ์นานสุด เริ่มออกฤทธิ์ 3-5 ชั่วโมงหลังฉีด ออกฤทธิ์สูงสุด 10-16 ชั่วโมง และยาอยู่ได้นาน 18-24 ชั่วโมง

4) อินซูลินผสม (Insulin mixtures)

เป็นการผสมอินซูลินออกฤทธิ์เร็วกับอินซูลินออกฤทธิ์ปานกลางโดยมาก ผสมอัตราส่วน 30:70

สำหรับข้อบ่งใช้อินซูลินนั้นจะขึ้นอยู่กับชนิดของผู้ป่วยโรคเบาหวานแต่ละชนิด (คณะเภสัชศาสตร์ มหาวิทยาลัยศิลปากร, www, 2008) ดังนี้

ผู้ป่วยเบาหวานชนิดที่หนึ่ง ผู้ป่วยเบาหวานชนิดที่สองที่ไม่สามารถควบคุมระดับน้ำตาลในเลือดได้ด้วยการปรับเปลี่ยนพฤติกรรมและการใช้ยาลดระดับน้ำตาลในเลือดชนิดรับประทาน ผู้ป่วยเบาหวานขณะตั้งครรภ์ ผู้ที่เป็นเบาหวานหลังการตัดตับอ่อน ผู้ป่วยเบาหวานที่มีภาวะ Ketoacidosis และการควบคุมน้ำตาลในผู้ป่วยเบาหวานขณะผ่าตัด ปกติคนทั่วไปหลั่งอินซูลิน 18-40 ยูนิตต่อวันหรือชั่วโมงละ 0.5-1 ยูนิต แต่เมื่อรับประทานอาหาร อินซูลินจะหลั่งเพิ่มขึ้นเป็น 6 ยูนิตต่อชั่วโมง กรณีคนอ้วนจะมีการหลั่งเพิ่มขึ้น 4 เท่า หรือมากกว่าโดยอินซูลินที่ถูกหลั่งออกมาจะเข้า portal circulation ซึ่งจะถูกทำลายไปประมาณครึ่งหนึ่งก่อนเข้าสู่กระแสเลือด

การหลั่งอินซูลินตามธรรมชาติมีลักษณะดังนี้ เมื่อเริ่มต้นมื้ออาหารอินซูลินจะถูกหลั่งออกมาจากตับอ่อนในปริมาณมากอย่างรวดเร็ว ซึ่งเสร็จภายใน 10 นาที ระยะนี้ถือเป็นช่วงเริ่มต้นของการหลั่งอินซูลิน จากนั้นตับอ่อนจะหลั่งอินซูลินต่อไปในปริมาณที่ต่ำลง แต่ก็ยังสูงกว่าเวลาอดอาหาร หรือช่วงที่ไม่ได้รับประทานอาหาร โดยการหลั่งช่วงนี้คงอยู่ประมาณ 2-3 ชั่วโมง ก่อนที่ปริมาณการหลั่งจะลดลงมาสู่ระดับต่ำ ๆ (0.5 ยูนิตต่อชั่วโมง) ซึ่งถือเป็นปริมาณพื้นฐานของการหลั่งอินซูลิน

การให้อินซูลินในผู้ป่วยเบาหวานจะพยายามเลียนแบบการหลั่งตามธรรมชาติให้มากที่สุด ดังนี้

1) ผู้ป่วยเบาหวานชนิดที่หนึ่ง มีความบกพร่องในการหลั่งอินซูลินจากตับอ่อนจึงต้องอาศัยการให้อินซูลินจากภายนอกทดแทนเข้าไป ขนาดอินซูลินเริ่มต้นคือ 0.6-0.75 ยูนิตต่อกิโลกรัมต่อวัน แต่เนื่องจากผู้ป่วยมักมีภาวะดื้ออินซูลินด้วย ในช่วงสัปดาห์แรกของการใช้ยาจึงอาจต้องใช้ยาถึง 1 ยูนิตต่อกิโลกรัมต่อวัน การให้อินซูลินควรเริ่มจากการฉีดวันละสองครั้ง โดยฉีดก่อนอาหารเช้าและก่อนอาหารเย็น ขนาดที่ฉีดตอนเช้าปริมาณ 2/3 ของขนาดยาทั้งหมด และขนาดที่ฉีดตอนเย็นประมาณ 1/3 ของยาทั้งหมด สัดส่วนของยาฉีดตอนเช้ามี Intermediate : Rapid/short acting ประมาณ 70:30 และสัดส่วนยาตอนเย็นประมาณ 50:50

ผู้ป่วยควรได้รับอินซูลินทดแทนทั้งช่วงเริ่มต้นการหลั่งอินซูลินและช่วงการหลั่งปกติ ในการทดแทนช่วงเริ่มต้น ผู้ป่วยควรได้รับอินซูลินที่ออกฤทธิ์เร็วเช่น Rapid/regular insulin โดยต้องฉีดประมาณ 30 นาทีก่อนอาหาร หรือให้ Rapid-acting insulin analog ซึ่งสามารถให้ก่อนอาหารเพียง 15 นาทีหรือก่อนอาหารทันที และอาจช่วยสามารถควบคุมระดับน้ำตาลในเลือดได้ดีกว่า Regular insulin

สำหรับทดแทนช่วงการหลั่งปกติของอินซูลินหรือ Basal insulin ในปัจจุบันแนะนำให้ใช้ Insulin glargine ซึ่งไม่มี Peak effect และมีระดับยาในเลือดค่อนข้างคงที่กว่า NPH แทนการใช้ NPH เช่นในอดีต ในทางปฏิบัติมักให้ Insulin glargine วันละ 1 ครั้งก่อนนอน ซึ่งโดยปกติผู้ป่วยมักต้องการทดแทน Basal insulin ปริมาณครึ่งหนึ่งของปริมาณความต้องการอินซูลิน

ทั้งหมดต่อวัน หากผู้ป่วยได้รับ NPH เป็น Basal insulin อยู่แล้ว แนะนำให้ใช้ขนาดเริ่มต้นของ Insulin glargine ประมาณร้อยละ 80 ของ NPH แล้วจึงปรับขนาดตามการตอบสนองของผู้ป่วยอีกครั้ง

ในผู้ป่วยที่เป็นเบาหวานตั้งแต่เด็ก มักเริ่มฉีดอินซูลินในขนาด 0.6-0.9 ยูนิตต่อกิโลกรัมต่อวัน เมื่อเข้าสู่วัยหนุ่มสาวจะต้องการอินซูลินในขนาดสูงขึ้นมาก ๆ เนื่องจากมีความต้องการพลังงานเพิ่มขึ้น การเจริญเติบโตและการเปลี่ยนแปลงของระดับฮอร์โมนในร่างกาย อาจต้องเพิ่มขนาดอินซูลินเป็น 1.5 ยูนิตต่อกิโลกรัมต่อวัน และเมื่อเข้าสู่วัยผู้ใหญ่ความต้องการอินซูลินจะลดลง ควรได้รับอินซูลินในขนาดที่ต่ำกว่า 1 ยูนิตต่อกิโลกรัมต่อวัน ทั้งนี้ต้องขึ้นอยู่กับความสามารถในการควบคุมระดับน้ำตาลของแต่ละบุคคลด้วย

2) ผู้ป่วยเบาหวานชนิดที่สอง โดยทั่วไปการรักษามักเริ่มต้นด้วยการให้ยาลดระดับน้ำตาลในเลือดเพียงตัวเดียวก่อน หากไม่สามารถควบคุมระดับน้ำตาลได้ อาจพิจารณาเพิ่มยารับประทานอีกชนิดหนึ่ง แต่หากใช้ยา 2 ชนิดร่วมกันแล้วยังไม่ได้ผลที่พอใจ ควรเริ่มใช้อินซูลินร่วมด้วย ขนาดเริ่มต้นในการใช้อินซูลินคือ 10-20 ยูนิตต่อวัน และอาจเพิ่มขนาดยาได้ 5-10 ยูนิตต่อสัปดาห์ จนกระทั่งระดับน้ำตาล 120 มิลลิกรัมต่อเดซิลิตร การใช้ควรเป็นอินซูลินชนิดออกฤทธิ์นาน เช่น NPH, Ultralente หรือ Glargine ในขนาด 0.1-0.4 ยูนิตต่อน้ำหนักตัวเป็นกิโลกรัม ฉีดเข้าใต้ผิวหนังก่อนนอน ร่วมกับการให้ยาชนิดรับประทานช่วงกลางวัน

สำหรับผู้ป่วยเบาหวานชนิดที่สองที่ยังคงมีการหลั่งอินซูลินจากตับอ่อนอยู่บ้าง และใช้อินซูลินเป็นยาเดียวในการรักษา เลือก Regimens ได้ หลายรูปแบบ โดยอาจฉีดอย่างน้อยวันละ 3 ครั้งก่อนอาหาร หรือเป็น Prandial insulin 2 ครั้งก่อนอาหารเช้าและก่อนอาหารเย็น และให้ NPH ร่วมด้วยตอนเช้า สำหรับผู้ที่ไม่อยากฉีดยาบ่อย อาจฉีดอย่างน้อยวันละ 2 ครั้ง โดยใช้อินซูลินสูตรผสมระหว่าง Short acting หรือ Rapid acting ร่วมกับ Intermediate acting

โดยทั่วไปในผู้ป่วยเบาหวานชนิดที่สองนิยมฉีดอินซูลินวันละ 2 ครั้ง โดยฉีดก่อนอาหารเช้าและอาหารเย็น ขนาดที่ต้องฉีดตอนเช้าประมาณ 2/3 ของขนาดยาทั้งหมด และขนาดที่ฉีดตอนเย็นประมาณ 1/3 ของยาทั้งหมด สัดส่วนของยาฉีดตอนเช้ามี Intermediate: Rapid/short acting ประมาณ 70:30 และสัดส่วนตอนเย็นประมาณ 70:30 หรือ 50:50 โดยขนาดที่กล่าวมาเป็นขนาดยาเริ่มต้น ต้องปรับขนาดยาตามการตอบสนองของผู้ป่วยแต่ละรายต่อไป

หลังการรักษาประมาณ 15-20 ปี ผู้ป่วยเบาหวานชนิดที่สองส่วนมากจะมีการหลั่งอินซูลินน้อยหรือไม่มีการหลั่งเลยเนื่องจากการเสื่อมของเบต้าเซลล์ ในกรณีนี้ผู้ป่วยจำเป็นต้องได้รับอินซูลินทดแทนเข้าไปเหมือนผู้ป่วยเบาหวานชนิดที่หนึ่ง และเนื่องจากผู้ป่วยมักมีภาวะอ้วนและดื้อต่ออินซูลินด้วย จึงอาจมักต้องได้รับอินซูลินขนาดสูงถึง 80-100 ยูนิตต่อวันหรือมากกว่านี้ถึงจะควบคุมระดับน้ำตาลในเลือดให้อยู่ในระดับที่น่าพอใจได้

การใช้อินซูลินร่วมกับยาเม็ดลดระดับน้ำตาลในเลือดใช้ในกรณีที่การรักษาวิธีใดวิธีหนึ่งไม่สามารถควบคุมระดับน้ำตาลได้ดีพอ เช่น การรับประทานยาได้ผลไม่เพียงพอ หรือในผู้ป่วยเบาหวานชนิดที่สองที่ฉีดอินซูลินขนาดสูง (100 ยูนิตต่อวัน หรือ 1.5 ยูนิตต่อกิโลกรัมต่อวัน) แล้วยังควบคุมระดับน้ำตาลไม่ได้ เนื่องจากเกิดภาวะดื้ออินซูลินสูง การเพิ่มยาเม็ดลดระดับน้ำตาลในเลือดอาจทำให้ผู้ป่วยควบคุมระดับน้ำตาลได้ดีขึ้น

สำหรับปริมาณอินซูลินที่จะลดระดับน้ำตาลในเลือดแต่ละคนนั้น ไม่สามารถกำหนดปริมาณอินซูลินเพื่อลดระดับน้ำตาลที่แน่นอนได้ เนื่องจากมีปัจจัยหลายอย่างที่มีผลต่อการดูดซึมอินซูลิน เช่น ตำแหน่งที่ฉีด ซึ่งพบว่าหน้าท้องเป็นตำแหน่งที่ดีที่สุด อาหาร การออกกำลังกาย ชนิด/ปริมาตร และความเข้มข้นของอินซูลินที่ใช้ หรือความลึกในการฉีด เป็นต้น

4.1.3 ข้อมูลที่ใช้ในการทดลองสร้างดาต้าเบสทริกเกอร์โดยการทำเหมืองข้อมูล

ในงานวิจัยชิ้นนี้ได้ศึกษาและทดลองวิธีการเรียนรู้ข้อมูลจากฐานข้อมูลเชิงสัมพันธ์ร่วมกับคำสั่ง SQL ในการสร้างดาต้าเบสทริกเกอร์ โดยได้ใช้ฐานข้อมูลประวัติของผู้ป่วยที่มีความเสี่ยงเป็นโรคเบาหวานและการให้ปริมาณอินซูลินเพื่อรักษาระดับน้ำตาลในเลือดที่มีโครงสร้างตารางต่าง ๆ ดังนี้ (แอททริบิวต์ที่ขีดเส้นใต้แสดงถึงแอททริบิวต์ที่เป็น Primary key)

Patient (Patient _ID, Name, Sex, Age, Occupation)

Diabetes_family (Diabetes_family_code, Relationship, Diabetes_family)

Symptom (Patient_ID, Date, Temperature, Weight, Height, BMI,
Blood_Pressure_Upper, Blood_Pressure_Low,
Diabetes_family_Code, Blood_Sugar, Insulin_level, Diabetes)

ในการค้นหากฎการจำแนกข้อมูล จำเป็นต้องทำการเตรียมข้อมูลเพื่อให้ข้อมูลดังกล่าวมีรูปแบบที่เหมาะสมกับการทำเหมืองข้อมูล โดยใช้คำสั่ง JOIN ON สำหรับการเชื่อมตารางเข้าด้วยกัน ซึ่งเป็นคำสั่งพื้นฐานของภาษา SQL (Microsoft SQL Server 2000) และคัดเลือกเฉพาะแอททริบิวต์ที่คาดว่ามีประโยชน์ในการค้นหากฎ โดยมีรูปแบบคำสั่งดังรูปที่ 4.2

```
SELECT  P.Patient_ID, P.Name, P.Sex, P.Age, S.Temperature, S.Blood_Pressure_Upper,
        S.Blood_Pressure_Low, D.Diabetes_family, S.Weight, S.Height, S.BMI,
        S.Blood_Sugar, S.Insulin_level, S.Diabetes
INTO    DIABETES
FROM    SYMPTOM S
JOIN    DIABETES_FAMILY D
ON      S.Diabetes_family_code = D.Diabetes_family_code
JOIN    PATIENT P
ON      P.Patient_ID = S.Patient_ID
```

รูปที่ 4.2 แสดงรูปแบบคำสั่งในการเตรียมข้อมูลเพื่อทำเหมืองข้อมูล

ข้อมูลของทั้ง 3 ตารางคือตาราง Patient, Diabetes_family และตาราง Symptom รวมทั้งข้อมูลของตาราง Diabetes ที่ได้จากการรวมตารางหรือการ Join แสดงไว้ดังรูปที่ 4.3 และมีโครงสร้างของตาราง Diabetes ที่มีจำนวนแอททริบิวต์ทั้งหมด 14 แอททริบิวต์ดังนี้

Diabetes (Patient_ID, Name, Sex, Age, Temperature, Blood_Pressure_Upper,
Blood_Pressure_Low, Diabetes_family, Weight, Height, BMI,
Blood_Sugar, Insulin_level, Diabetes)

Patient

Patient_ID	Name	Sex	Age	Occupation
P001	Adams	Male	40	Teacher
P002	Blake	Male	45	Engineer
P003	King	Male	43	Lecturer
P004	Miller	Male	55	Doctor
P005	Minny	Female	30	Nurse
……	……	……	…….	……

Diabetes_family

Diabetes_family_code	Relationship	Diabetes_family
0	None	NO
1	Father	YES
2	Mother	YES
……	……	……

⋈

Symptom

Patient_ID	Date	Temp.	Weight	Height	BMI	Pressure_Upper	Pressure_Low	Diabetes_family_Code	Blood_Sugar	Insulin_Level	Diabetes
P001	2/3/2007	37.4	85	1.7	29.4	130	80	0	150	80	YES
P002	3/3/2007	37.6	88	1.78	27.8	128	90	2	155	85	YES
P003	4/3/2007	38.6	90	1.7	31.1	170	90	2	128	20	YES
P004	5/3/2007	37.5	79	1.73	26.4	160	90	1	128	20	YES
P005	6/3/2007	37.7	69	1.6	27.0	135	80	0	119	0	NO
….	….	….	…	…	….	….	…	….	….	…	…

↓

Diabetes

Patient_ID	Name	Sex	Age	Temperature	Blood_Pressure_Upper	Blood_Pressure_Lower	Weight	Height	BMI	Blood Sugar	Diabetes_family	Diabetes	Insulin (U/d)
P001	Adams	Male	40	37.4	130	80	85	1.7	29.4	150	No	YES	80
P002	Blake	Male	45	37.6	128	90	88	1.78	27.8	155	Yes	YES	85
P003	King	Male	43	38.6	170	90	90	1.7	31.1	128	Yes	YES	20
P004	Miller	Male	55	37.5	160	90	79	1.73	26.4	128	Yes	YES	20
P005	Minny	Female	30	37.7	135	80	69	1.6	27.0	119	No	NO	0
……	……	……	….	……	…….	…….	….	…..	…..	……	….	…..	…….

รูปที่ 4.3 แสดงข้อมูลในตารางทั้งหมดที่ใช้ในการทดลอง

4.1.4 กฎที่ได้จากการทำเหมืองข้อมูลประเภทการจำแนกข้อมูล

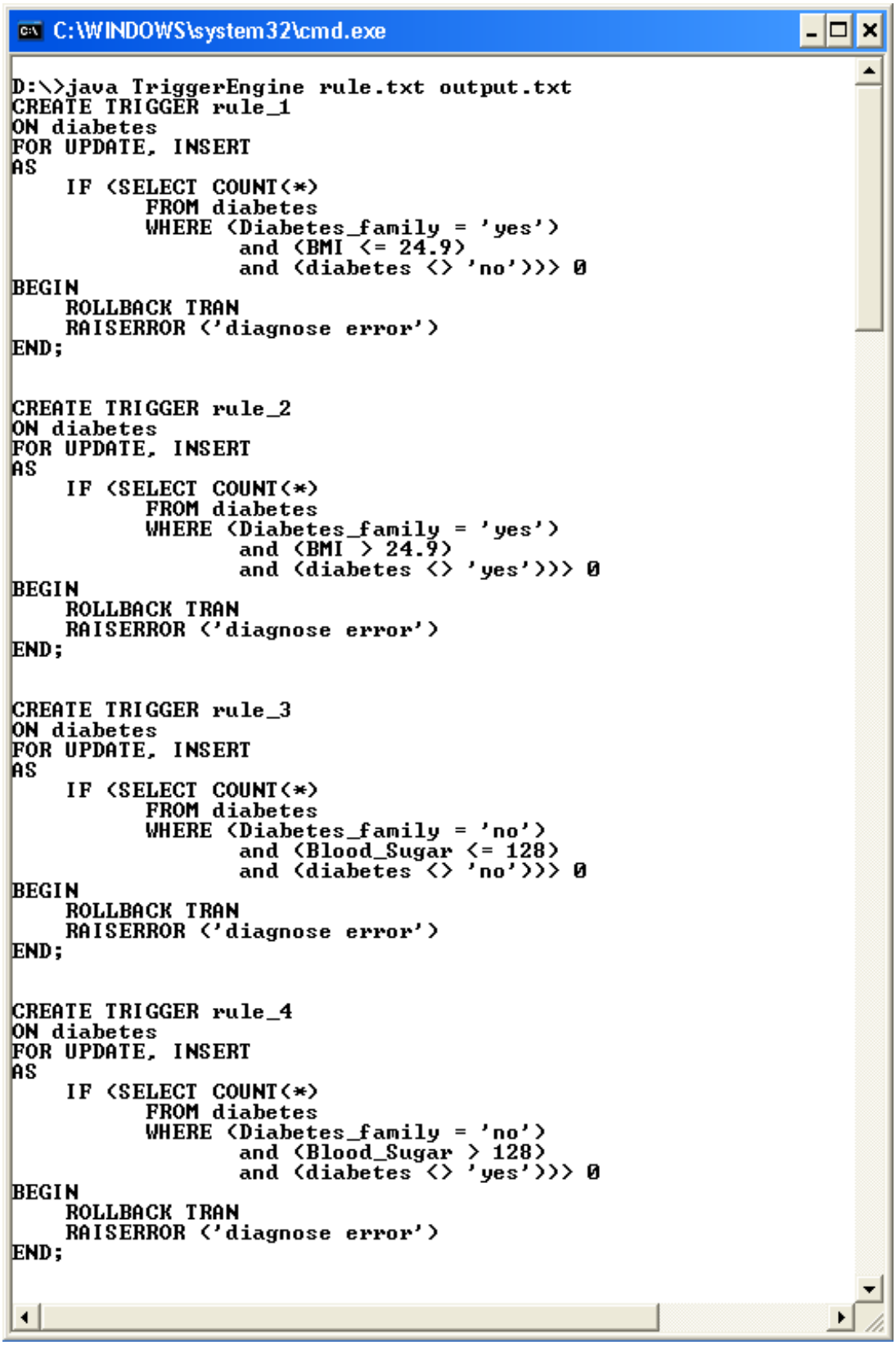
ในการทดลองได้ใช้ Training Data ตามรูปที่ 4.3 จำนวน 20 เรคคอร์ดผนวกกับการใช้ความรู้พื้นฐานทางด้านการแพทย์ในการให้ปริมาณอินซูลินแก่ผู้ป่วยโรคเบาหวาน และนำข้อมูลชุดดังกล่าวมาทำเหมืองข้อมูลโดยใช้อัลกอริทึม J48 ทำให้ได้ Induced trigger rules กฎที่ 1– 4 ดังนี้

Induced Trigger Rule:

1. If Diabetes_family = yes and BMI <= 24.9 Then Diabetes = no
2. If Diabetes_family = yes and BMI > 24.9 Then Diabetes = yes
3. If Diabetes_family = no and blood_sugar <= 128 Then Diabetes = no
4. If Diabetes_family = no and blood_sugar > 128 Then Diabetes = yes

4.2 การสร้างดาต้าเบสทริกเกอร์จากกฎการจำแนก

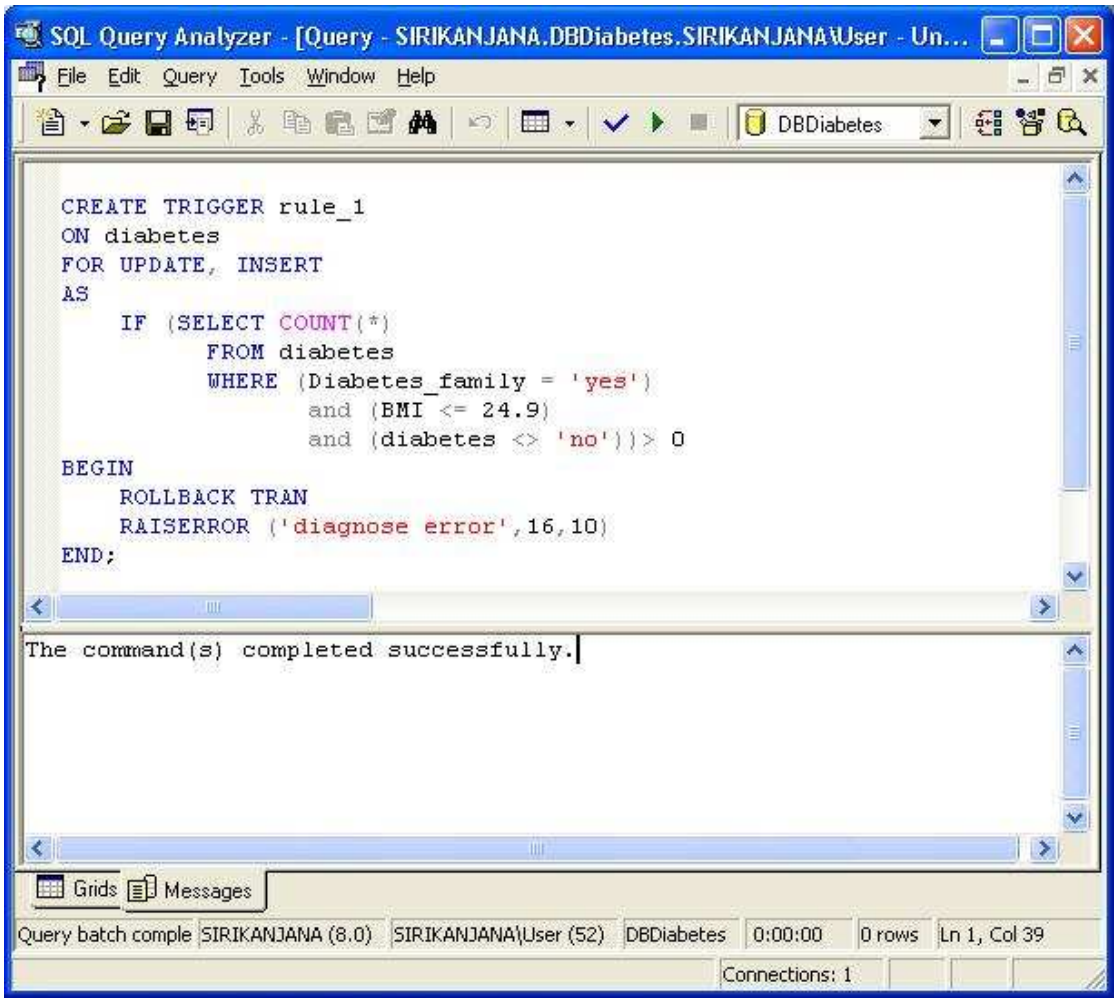
จาก Induced trigger rules ที่ได้ ดังปรากฏในกฎที่ 1-4 ด้านบน สามารถนำมาแปลงเป็นดาต้าเบสทริกเกอร์ได้ โดยใช้โปรแกรมการสร้างดาต้าเบสทริกเกอร์จากกฎที่เขียนขึ้นด้วยโปรแกรมภาษา JAVA ซึ่งแสดงรายละเอียดของโปรแกรมในภาคผนวก ข และมีวิธีรายละเอียดของทริกเกอร์ดังรูปที่ 4.4



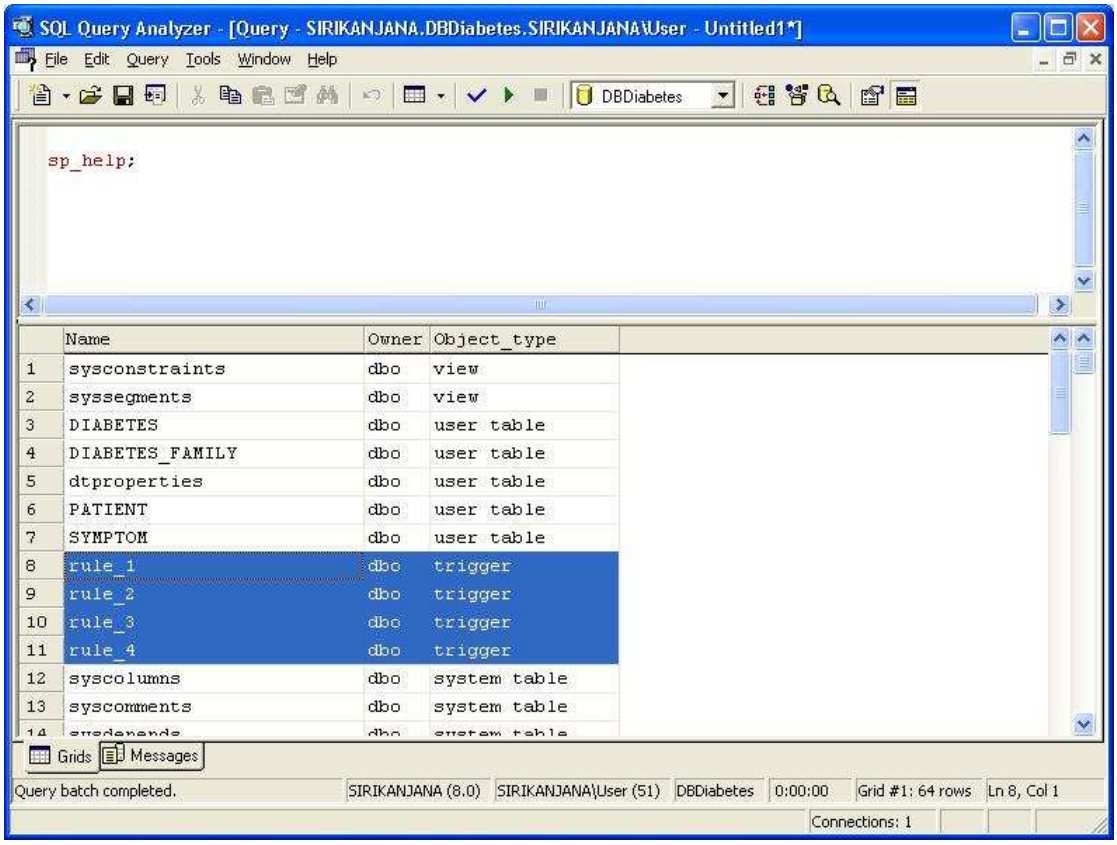
รูปที่ 4.4 แสดงดาต้าเบสทริกเกอร์ที่แปลงจากกฎการจำแนก

4.2.1 การนำดาต้าเบสทริกเกอร์ไปสร้างในฐานข้อมูล

จากขั้นตอนการแปลงกฎประเภทการจำแนกเป็นดาต้าเบสทริกเกอร์ เมื่อได้รูปแบบของดาต้าเบสทริกเกอร์แล้ว จากนั้นนำรูปแบบของดาต้าเบสทริกเกอร์ทั้งหมดที่ได้ไปสร้างไว้ในระบบจัดการฐานข้อมูล Microsoft SQL Server 2000 เพื่อเป็นการตรวจสอบความถูกต้องของการเปลี่ยนแปลงข้อมูลเมื่อมีการใช้คำสั่ง SQL ประเภท DML คือ การเพิ่ม การแก้ไขและการลบข้อมูล ซึ่งมีขั้นตอนการสร้างดาต้าเบสทริกเกอร์ในฐานข้อมูลดังปรากฏในรูปที่ 4.5 และแสดงการเรียกดูออบเจ็กต์รวมทั้งดาต้าเบสทริกเกอร์ทั้งหมดในฐานข้อมูลดังรูปที่ 4.6

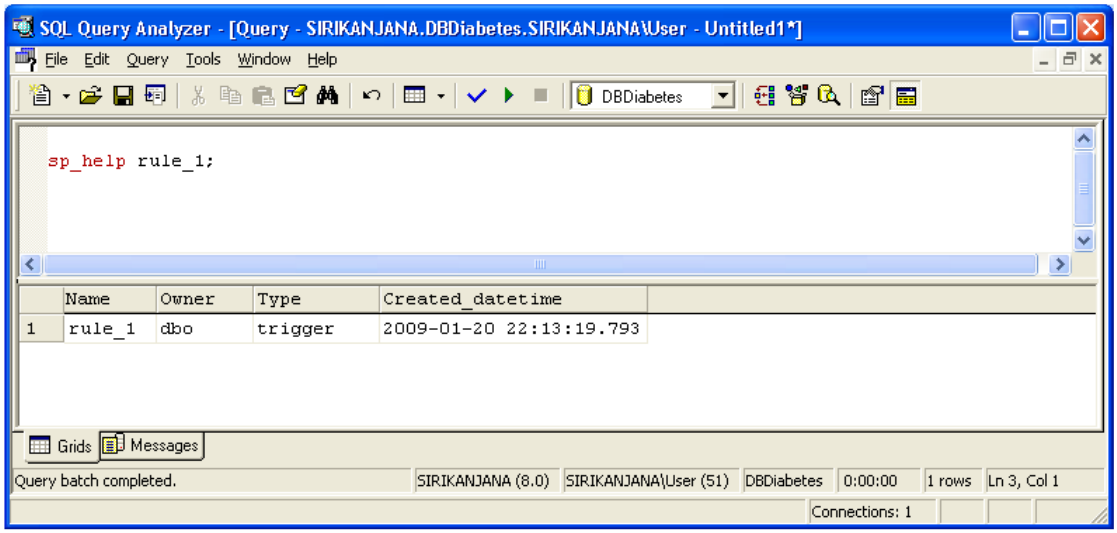


รูปที่ 4.5 แสดงขั้นตอนการสร้างดาต้าเบสทริกเกอร์ในระบบฐานข้อมูล

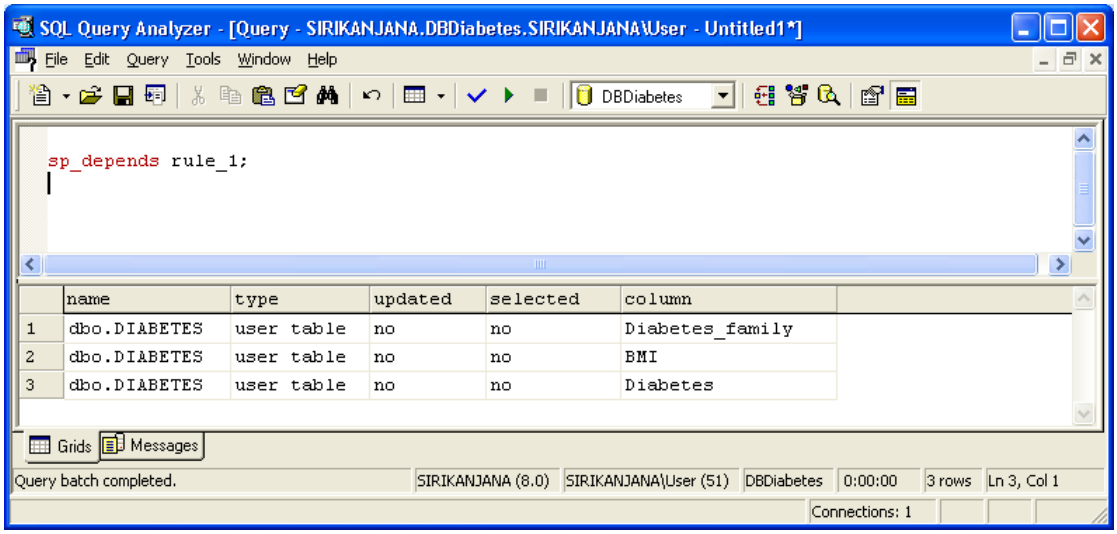


รูปที่ 4.6 แสดงการเรียกดูดาต้าเบสทริกเกอร์ทั้งหมดที่ถูกสร้างขึ้นในฐานข้อมูล

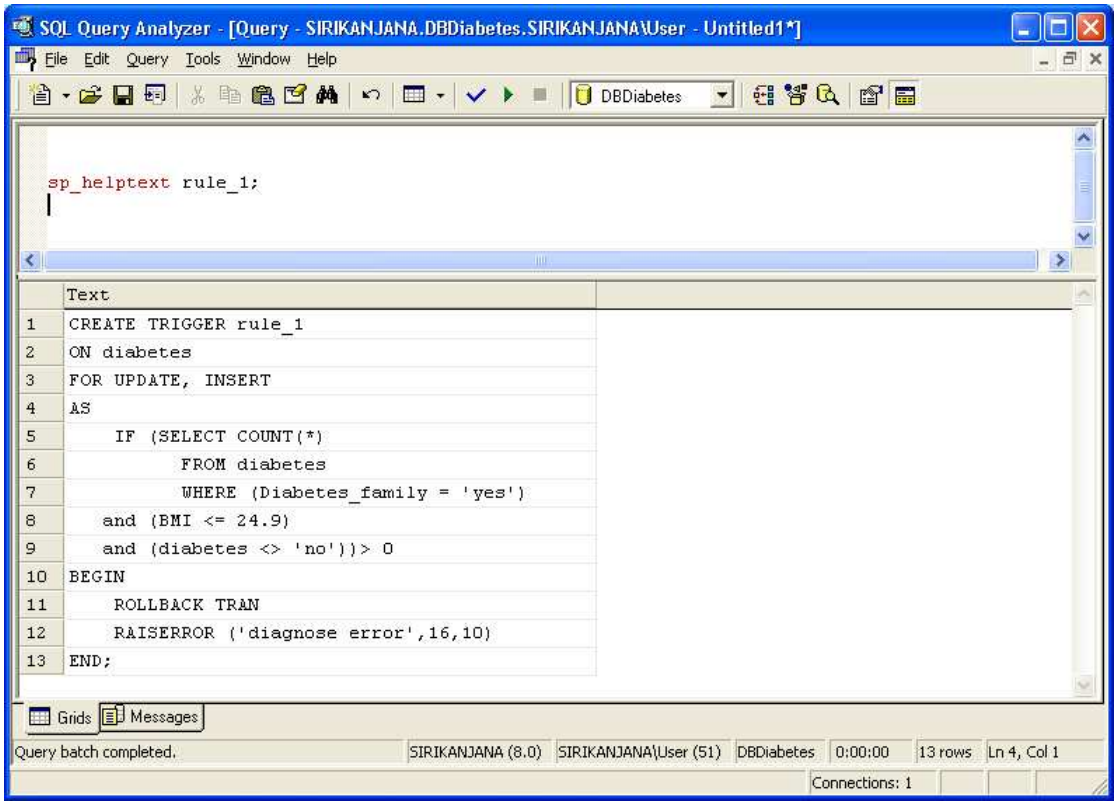
Microsoft SQL Server 2000 ได้เตรียม Stored Procedure ที่ชื่อ sp_help, sp_depend และ sp_helptext เพื่อช่วยแสดงรายละเอียดต่าง ๆ ของดาต้าเบสทริกเกอร์ เช่น คำสั่งภายในหรือสิ่งที่ดาต้าเบสทริกเกอร์อ้างถึง ผู้สร้างดาต้าเบสทริกเกอร์รวมทั้งเวลาที่สร้างดาต้าเบสทริกเกอร์ เป็นต้น โดยมีการเรียกใช้งานดังรูปที่ 4.7 – 4.9 ดังนี้



รูปที่ 4.7 แสดงการเรียกดูรายละเอียดของการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.8 แสดงการเรียกดูออบเจ็กต์ที่ถูกอ้างถึงภายในดาต้าเบสทริกเกอร์



รูปที่ 4.9 แสดงการเรียกดูคำสั่งภายในดาต้าเบสทริกเกอร์

4.2.2 การทดสอบดาต้าเบสทริกเกอร์ด้วยการเปลี่ยนแปลงฐานข้อมูล

ในหัวข้อนี้เป็นการทดสอบความถูกต้องของการเปลี่ยนแปลงฐานข้อมูลด้วยคำสั่ง SQL ประเภท DML โดยใช้คำสั่ง Insert และ Update ซึ่งก็คือการเพิ่ม และการแก้ไขข้อมูล ตามลำดับ โดยในหนึ่งคำสั่งที่ใช้ทดสอบ จะแบ่งทำการทดสอบเป็นสองส่วนคือ ส่วนแรกจะเป็นการทดสอบความถูกต้องของการเปลี่ยนแปลงฐานข้อมูลโดยไม่ได้สร้างดาต้าเบสทริกเกอร์ และส่วนที่สองจะเป็นการทดสอบความถูกต้องของฐานข้อมูลโดยการสร้างดาต้าเบสทริกเกอร์ ซึ่งทดลองกับ Training data จากเดิมจำนวน 20 เรคคอร์ด โดยมีคำสั่งที่ใช้ทดสอบดังต่อไปนี้

คำสั่งที่ 1 คำสั่งในการเพิ่มข้อมูลเข้าไปในตารางที่ชื่อว่า Diabetes

```
INSERT INTO Diabetes (Patient_ID, Name, Sex, Age, Temperature,
                      Blood_Pressure_Upper, Blood_Pressure_Low,
                      Diabetes_family, Weight, Height, BMI, Blood_Sugar,
                      Diabetes, Insulin_level)
VALUES ('P021', 'Amitta', 'Female', 33, 37.6, 130, 80, 'no', 72, 1.62,
        27.4, 130, 'no', 0)
```

<u>ทดสอบครั้งที่ 1</u> ทดสอบโดยไม่มีการสร้างดาต้าเบสทริกเกอร์

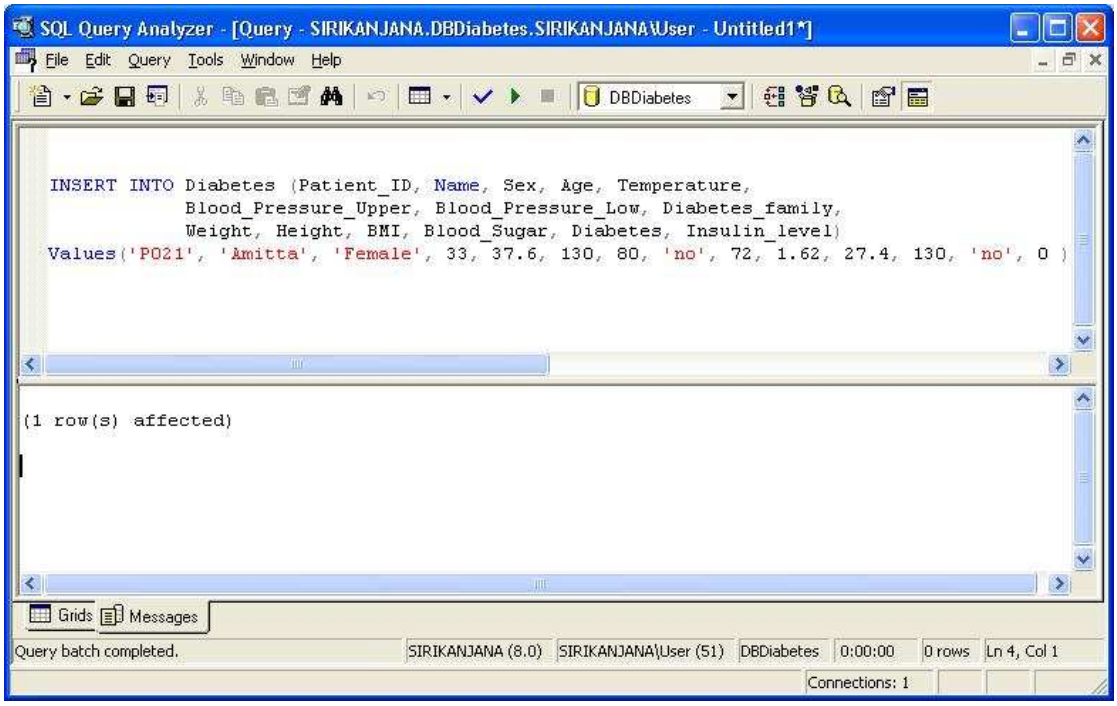
จากผลการทดสอบพบว่า ระบบฐานข้อมูลยอมให้เพิ่มข้อมูลดังกล่าวเข้าไปในฐานข้อมูลได้ จากเดิมมีข้อมูลอยู่ทั้งหมด 20 เรคคอร์ด ซึ่งข้อมูลที่ทำการเพิ่มเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกันเอง ข้อมูลดังกล่าวได้แก่

Diabetes_family = no, Blood_sugar = 130, Diabetes = No

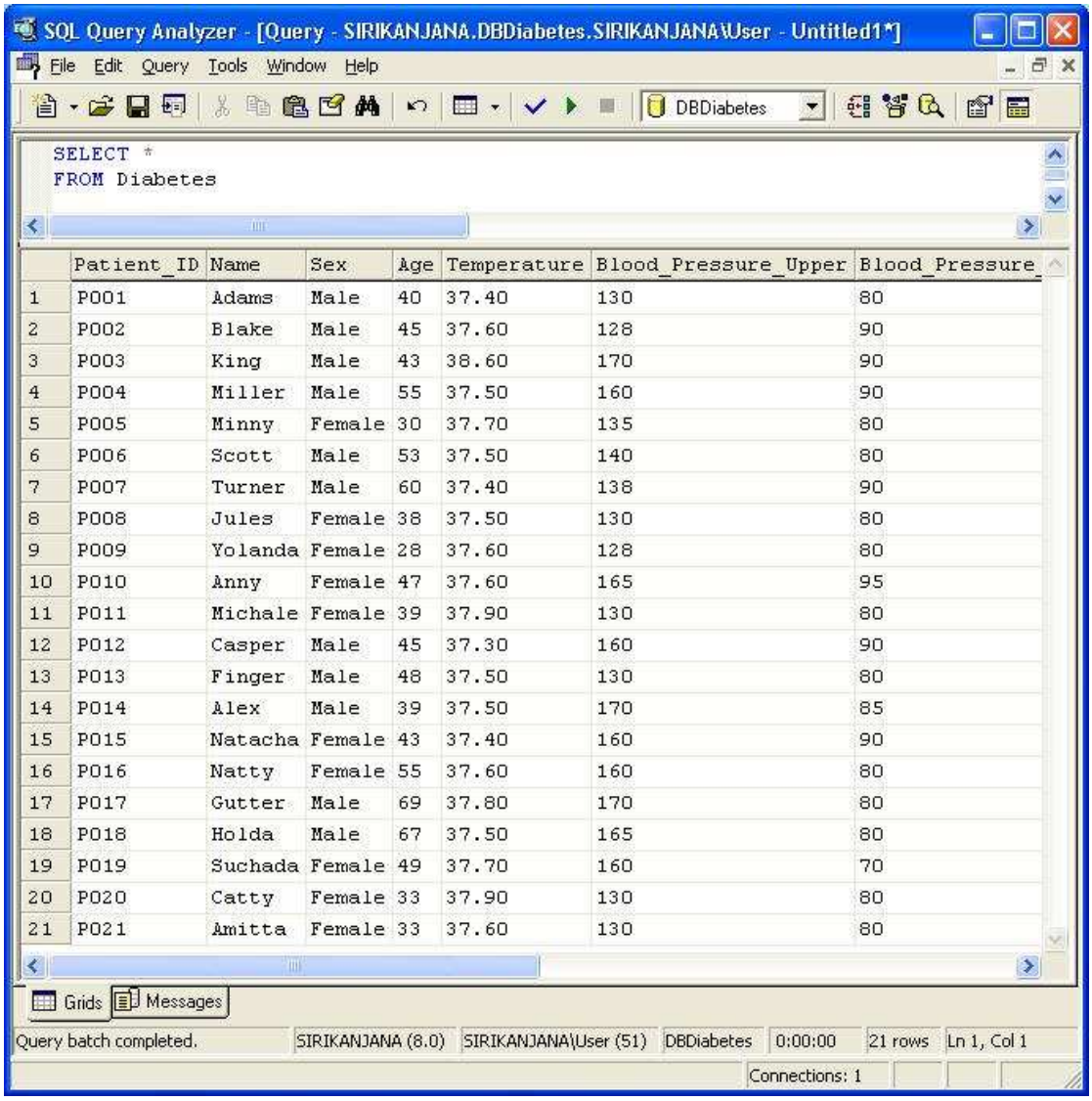
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่าเท่ากับ 130 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับเงื่อนไขคือ

Diabetes_family = no, Blood_sugar > 128, Diabetes = Yes

โดยเงื่อนไขของข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 128 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ดังนั้นถ้าหากมีการเก็บข้อมูลที่ไม่ถูกต้องลงไปในฐานข้อมูล จะส่งผลให้การประมวลผลมีความผิดพลาดเกิดขึ้นได้ รูปที่ 4.10 แสดงผลการทำงานของคำสั่งทดสอบว่าข้อมูลที่ผิดพลาดดังกล่าวสามารถเพิ่มเข้าไปในฐานข้อมูลได้ และรูปที่ 4.11 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากที่ทำการเพิ่มข้อมูลชุดใหม่เข้าไปพบว่ามีข้อมูลทั้งหมด 21 เรคคอร์ด



รูปที่ 4.10 แสดงผลการทดสอบคำสั่งที่ 1 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์



	Patient_ID	Name	Sex	Age	Temperature	Blood_Pressure_Upper	Blood_Pressure_
1	P001	Adams	Male	40	37.40	130	80
2	P002	Blake	Male	45	37.60	128	90
3	P003	King	Male	43	38.60	170	90
4	P004	Miller	Male	55	37.50	160	90
5	P005	Minny	Female	30	37.70	135	80
6	P006	Scott	Male	53	37.50	140	80
7	P007	Turner	Male	60	37.40	138	90
8	P008	Jules	Female	38	37.50	130	80
9	P009	Yolanda	Female	28	37.60	128	80
10	P010	Anny	Female	47	37.60	165	95
11	P011	Michale	Female	39	37.90	130	80
12	P012	Casper	Male	45	37.30	160	90
13	P013	Finger	Male	48	37.50	130	80
14	P014	Alex	Male	39	37.50	170	85
15	P015	Natacha	Female	43	37.40	160	90
16	P016	Natty	Female	55	37.60	160	80
17	P017	Gutter	Male	69	37.80	170	80
18	P018	Holda	Male	67	37.50	165	80
19	P019	Suchada	Female	49	37.70	160	70
20	P020	Catty	Female	33	37.90	130	80
21	P021	Amitta	Female	33	37.60	130	80

รูปที่ 4.11 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 1 โดยไม่มีดาต้าเบสทริกเกอร์

<u>ทดสอบครั้งที่ 2</u> ทดสอบโดยมีการสร้างดาต้าเบสทริกเกอร์

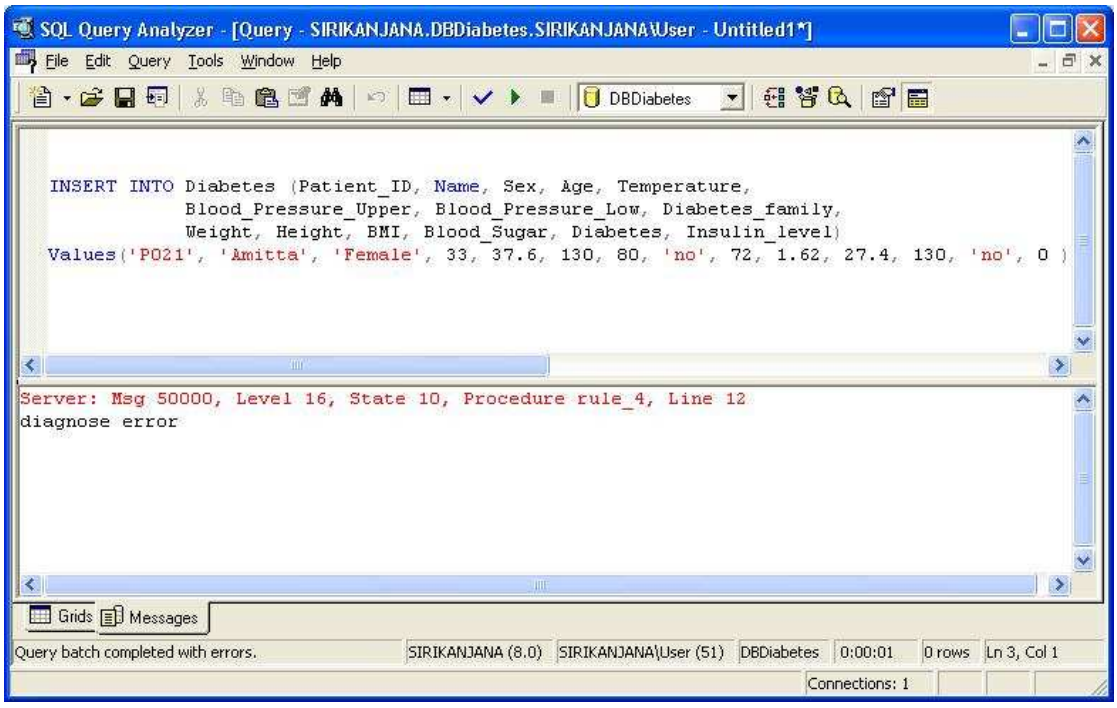
จากผลการทดสอบพบว่า ข้อมูลที่เพิ่มเข้าไปใหม่มีความขัดแย้งกับ Induced Trigger Rule จึงทำให้ไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้สำเร็จ เนื่องจากขัดแย้งกับดาต้าเบสทริกเกอร์ ซึ่งข้อมูลที่ทำการเพิ่มเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกับดาต้าเบสทริกเกอร์ดังนี้

Diabetes_family = no, Blood_sugar = 130, Diabetes = No

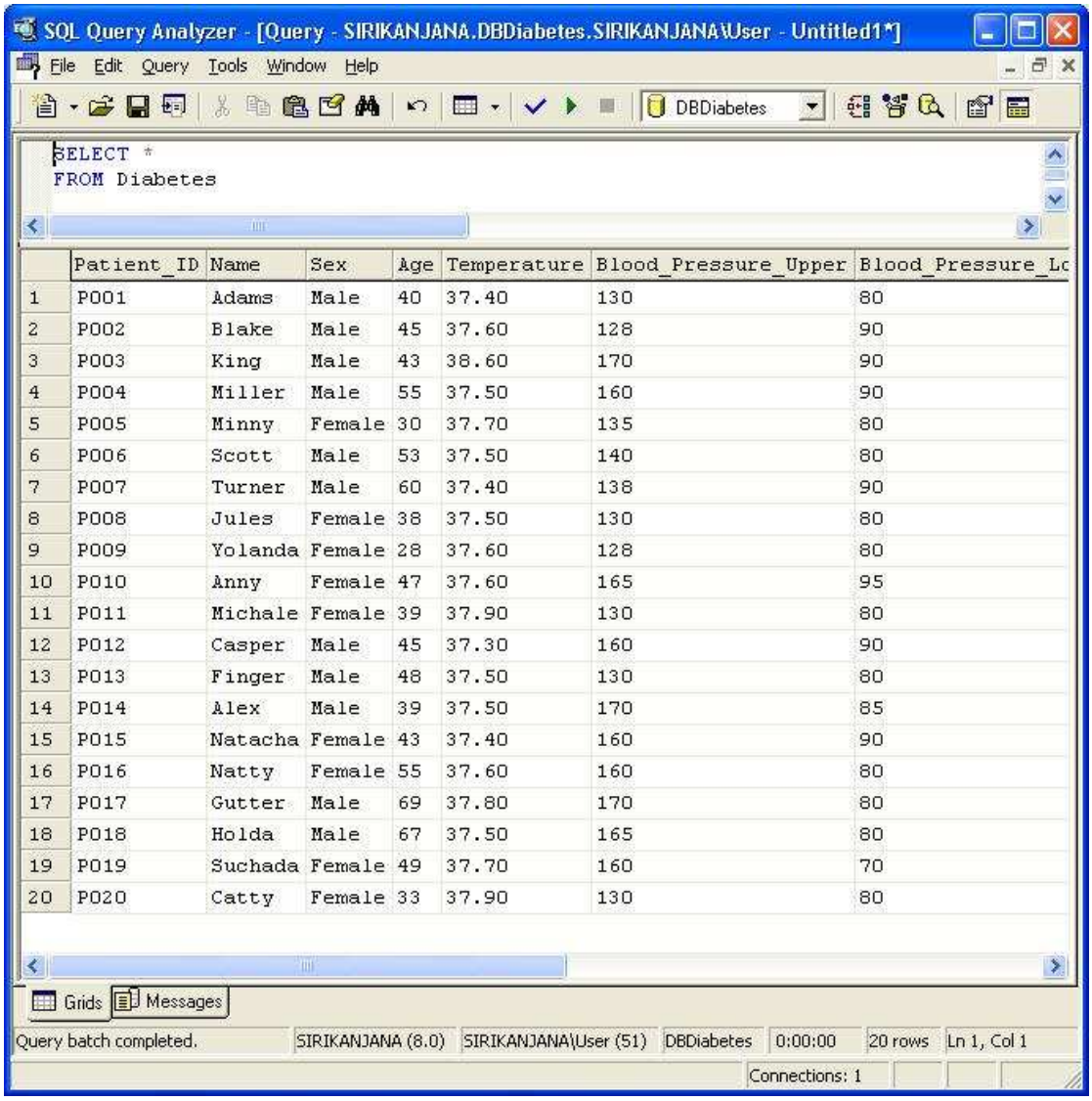
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่าเท่ากับ 130 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับดาต้าเบสทริกเกอร์ที่ชื่อว่า Rule_4 คือ

IF Diabetes_family = no and Blood_sugar > 128 Then Diabetes = Yes

โดยดาต้าเบสทริกเกอร์นี้มีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 128 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ซึ่งเมื่อทำการทดสอบคำสั่งในการเพิ่มข้อมูลดังกล่าวจะมีผลการทำงานแจ้งให้ทราบว่าไม่สามารถเพิ่มข้อมูลลงไปในฐานข้อมูลได้ โดยข้อความดังกล่าวเป็นผลจากการทำงานของดาต้าเบสทริกเกอร์ที่สร้างไว้ในฐานข้อมูล ดังแสดงในรูปที่ 4.12 และข้อมูลทั้งหมดในฐานข้อมูลหลังจากที่ทำการเพิ่มข้อมูลชุดใหม่เข้าไปพบว่ามีจำนวนข้อมูลเท่าเดิมคือ 20 เรคคอร์ด เนื่องจากไม่สามารถเพิ่มข้อมูลเข้าไปในฐานข้อมูลได้ แสดงในรูปที่ 4.13



รูปที่ 4.12 แสดงผลการทดสอบคำสั่งที่ 1 โดยมีการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.13 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 1 โดยมีดาต้าเบสทริกเกอร์

คำสั่งที่ 2 คำสั่งในการเพิ่มข้อมูลเข้าไปในตารางที่ชื่อว่า Diabetes

```
INSERT INTO  Diabetes (Patient_ID, Name, Sex, Age, Temperature,
                       Blood_Pressure_Upper, Blood_Pressure_Low,
                       Diabetes_family, Weight, Height, BMI, Blood_Sugar,
                       Diabetes, Insulin_level)
VALUES ('P022', 'John', 'Male', 31, 37.6, 165, 95, 'yes', 68, 1.55,
        28.33, 132, 'no', 0)
```

<u>ทดสอบครั้งที่ 1</u> ทดสอบโดยไม่มีการสร้างดาต้าเบสทริกเกอร์

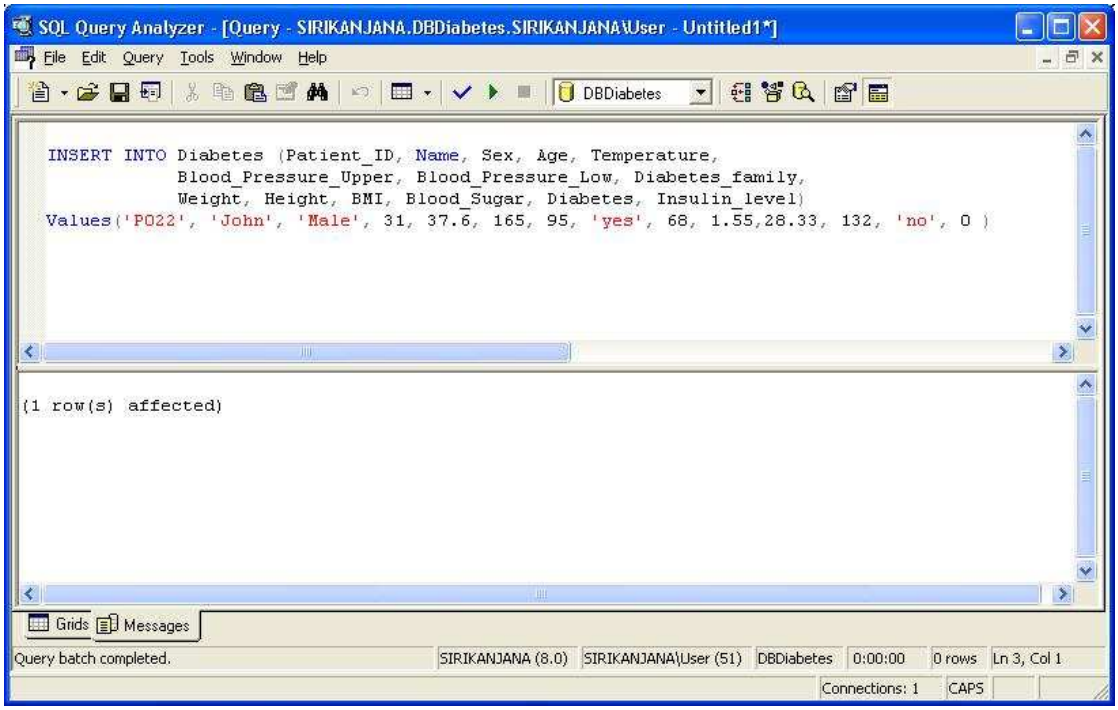
จากผลการทดสอบพบว่า ระบบฐานข้อมูลยอมให้เพิ่มข้อมูลดังกล่าวเข้าไปในฐานข้อมูลได้ จากเดิมมีข้อมูลอยู่ทั้งหมด 21 เรคคอร์ด (จากการทดสอบคำสั่งที่ 1 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์) ซึ่งข้อมูลที่ทำการเพิ่มเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกันเอง ข้อมูลดังกล่าวได้แก่

Diabetes_family = yes, BMI = 28.33, Diabetes = No

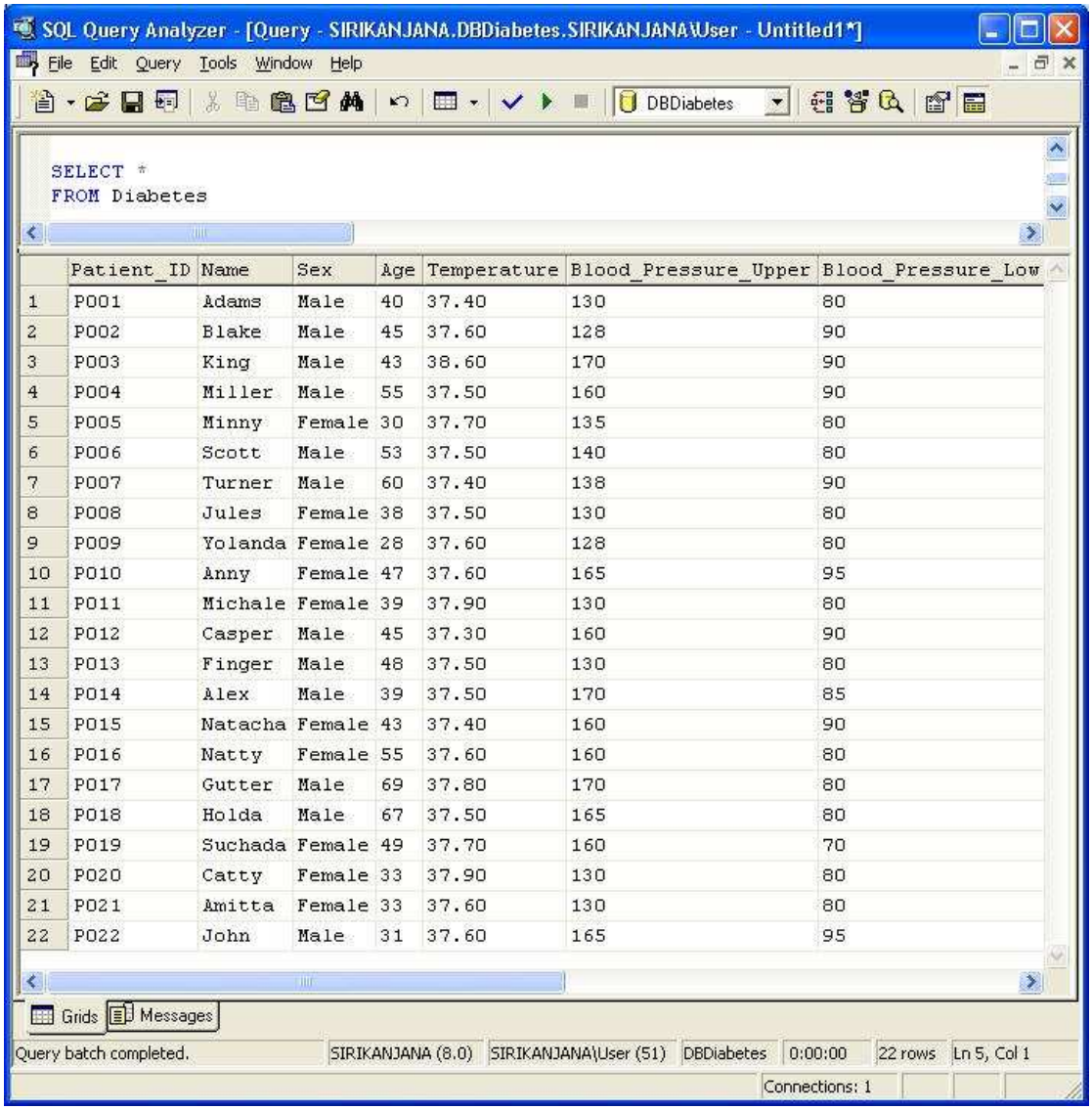
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและมีค่าดัชนีมวลกายเท่ากับ 28.33 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับเงื่อนไขคือ

Diabetes_family = yes, BMI > 24.9, Diabetes = Yes

โดยเงื่อนไขของข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและมีค่าดัชนีมวลกายมากกว่า 24.9 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ถ้าหากมีการเก็บข้อมูลที่ไม่ถูกต้องลงไป จะส่งผลให้การประมวลผลมีความผิดพลาดเกิดขึ้นได้ รูปที่ 4.14 แสดงผลการทำงานของคำสั่งทดสอบว่าข้อมูลที่ผิดพลาดดังกล่าวสามารถเพิ่มเข้าไปในฐานข้อมูลได้ และรูปที่ 4.15 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากที่ทำการเพิ่มข้อมูลชุดใหม่เข้าไปพบว่ามีข้อมูลทั้งหมด 22 เรคคอร์ด



รูปที่ 4.14 แสดงผลการทดสอบคำสั่งที่ 2 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์



	Patient_ID	Name	Sex	Age	Temperature	Blood_Pressure_Upper	Blood_Pressure_Low
1	P001	Adams	Male	40	37.40	130	80
2	P002	Blake	Male	45	37.60	128	90
3	P003	King	Male	43	38.60	170	90
4	P004	Miller	Male	55	37.50	160	90
5	P005	Minny	Female	30	37.70	135	80
6	P006	Scott	Male	53	37.50	140	80
7	P007	Turner	Male	60	37.40	138	90
8	P008	Jules	Female	38	37.50	130	80
9	P009	Yolanda	Female	28	37.60	128	80
10	P010	Anny	Female	47	37.60	165	95
11	P011	Michale	Female	39	37.90	130	80
12	P012	Casper	Male	45	37.30	160	90
13	P013	Finger	Male	48	37.50	130	80
14	P014	Alex	Male	39	37.50	170	85
15	P015	Natacha	Female	43	37.40	160	90
16	P016	Natty	Female	55	37.60	160	80
17	P017	Gutter	Male	69	37.80	170	80
18	P018	Holda	Male	67	37.50	165	80
19	P019	Suchada	Female	49	37.70	160	70
20	P020	Catty	Female	33	37.90	130	80
21	P021	Amitta	Female	33	37.60	130	80
22	P022	John	Male	31	37.60	165	95

รูปที่ 4.15 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 2 โดยไม่มีดาต้าเบสทริกเกอร์

ทดสอบครั้งที่ 2 ทดสอบโดยมีการสร้างดาต้าเบสทริกเกอร์

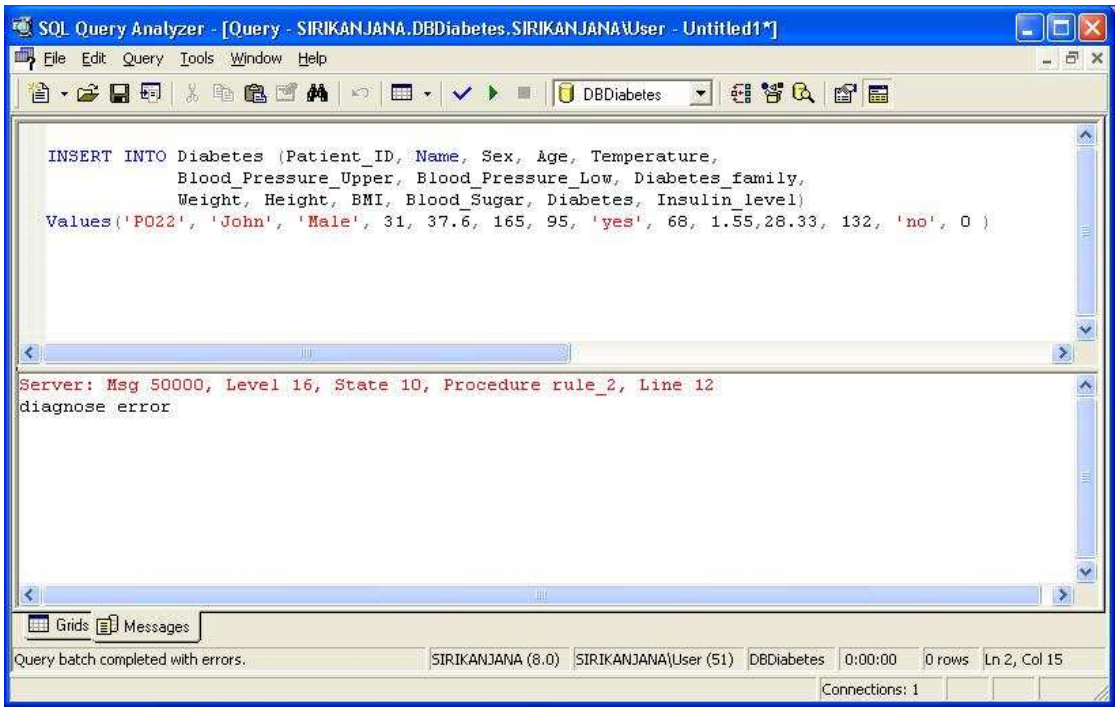
จากผลการทดสอบพบว่า ข้อมูลที่เพิ่มเข้าไปใหม่มีความขัดแย้งกับ Induced Trigger Rule จึงทำให้ไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้สำเร็จ เนื่องจากขัดแย้งกับดาต้าเบสทริกเกอร์ ซึ่งข้อมูลที่ทำการเพิ่มเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกับดาต้าเบสทริกเกอร์ดังนี้

Diabetes_family = yes, BMI = 28.33, Diabetes = No

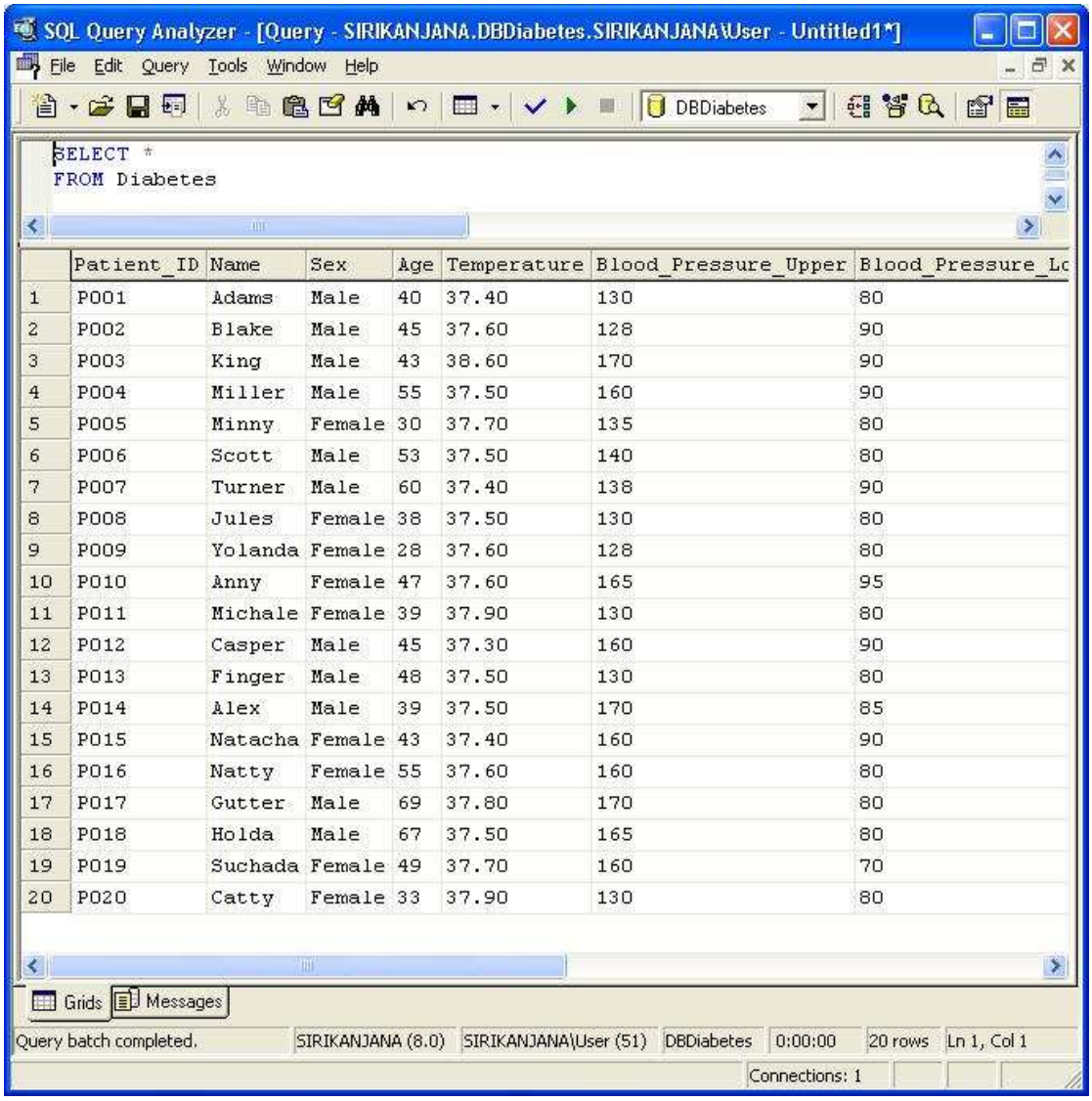
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและมีค่าดัชนีมวลกายเท่ากับ 28.33 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับคำสั่งดาต้าเบสทริกเกอร์ที่ชื่อว่า Rule_2 คือ

If Diabetes_family = yes and BMI > 24.9 Then Diabetes = yes

โดยเงื่อนไขของข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและมีค่าดัชนีมวลกายมากกว่า 24.9 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ซึ่งเมื่อทำการทดสอบคำสั่งในการเพิ่มข้อมูลดังกล่าวจะมีผลการทำงานแจ้งให้ทราบว่าไม่สามารถเพิ่มข้อมูลลงไปในฐานข้อมูลได้ โดยข้อความดังกล่าวเป็นผลจากการทำงานของดาต้าเบสทริกเกอร์ที่สร้างไว้ในฐานข้อมูล ดังแสดงในรูปที่ 4.16 และข้อมูลทั้งหมดในฐานข้อมูลหลังจากที่ทำการเพิ่มข้อมูลชุดใหม่เข้าไปพบว่ามีจำนวนข้อมูลเท่าเดิมคือ 20 เรคคอร์ด เนื่องจากไม่สามารถเพิ่มข้อมูลเข้าไปในฐานข้อมูลได้ แสดงในรูปที่ 4.17



รูปที่ 4.16 แสดงผลการทดสอบคำสั่งที่ 2 โดยมีการสร้างดาต้าเบสทริกเกอร์



	Patient_ID	Name	Sex	Age	Temperature	Blood_Pressure_Upper	Blood_Pressure_Lo
1	P001	Adams	Male	40	37.40	130	80
2	P002	Blake	Male	45	37.60	128	90
3	P003	King	Male	43	38.60	170	90
4	P004	Miller	Male	55	37.50	160	90
5	P005	Minny	Female	30	37.70	135	80
6	P006	Scott	Male	53	37.50	140	80
7	P007	Turner	Male	60	37.40	138	90
8	P008	Jules	Female	38	37.50	130	80
9	P009	Yolanda	Female	28	37.60	128	80
10	P010	Anny	Female	47	37.60	165	95
11	P011	Michale	Female	39	37.90	130	80
12	P012	Casper	Male	45	37.30	160	90
13	P013	Finger	Male	48	37.50	130	80
14	P014	Alex	Male	39	37.50	170	85
15	P015	Natacha	Female	43	37.40	160	90
16	P016	Natty	Female	55	37.60	160	80
17	P017	Gutter	Male	69	37.80	170	80
18	P018	Holda	Male	67	37.50	165	80
19	P019	Suchada	Female	49	37.70	160	70
20	P020	Catty	Female	33	37.90	130	80

รูปที่ 4.17 แสดงข้อมูลทั้งหมดในฐานข้อมูลหลังจากทดลองคำสั่งที่ 2 โดยมีดาต้าเบสทริกเกอร์

คำสั่งที่ 3 คำสั่งในการปรับปรุงข้อมูลเข้าไปในตารางที่ชื่อว่า Diabetes

```
UPDATE    Diabetes
SET       Diabetes = 'no'
WHERE     (Blood_sugar > 130) and (Diabetes_family = 'no')
```

ทดสอบครั้งที่ 1 ทดสอบโดยไม่มีการสร้างดาต้าเบสทริกเกอร์

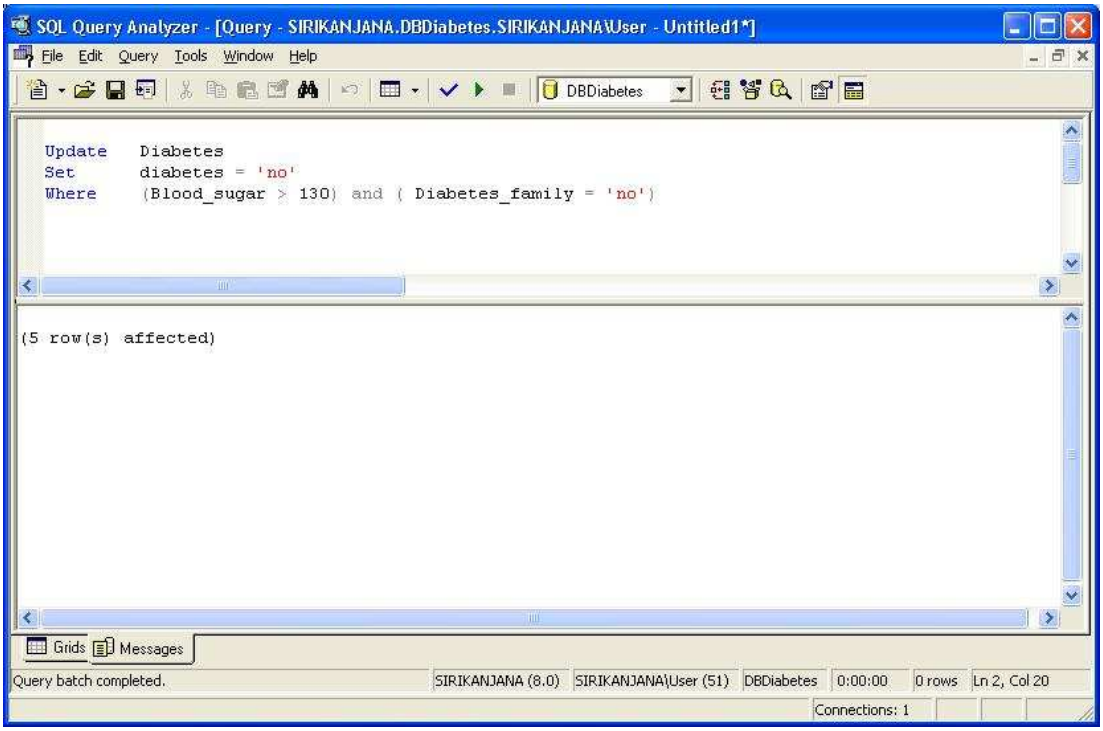
จากผลการทดสอบพบว่า ระบบฐานข้อมูลยอมให้ปรับปรุงแก้ไขข้อมูลตามคำสั่งดังกล่าวในฐานข้อมูลได้ โดยข้อมูลที่มีเงื่อนไขตรงกับคำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าวมีอยู่ทั้งหมด 5 เรคคอร์ด ดังนี้คือ Patient_ID เท่ากับ P001, P010, P012, P015, P018 และถูกปรับปรุงแก้ไขข้อมูลใหม่ตามคำสั่งดังกล่าวทั้งหมด ทำให้ข้อมูลที่ทำการปรับปรุงแก้ไขเข้าไปใหม่นั้นเป็นข้อมูลที่มีความขัดแย้งกันเอง ข้อมูลดังกล่าวได้แก่

Diabetes_family = no, Blood_sugar > 130, Diabetes = No

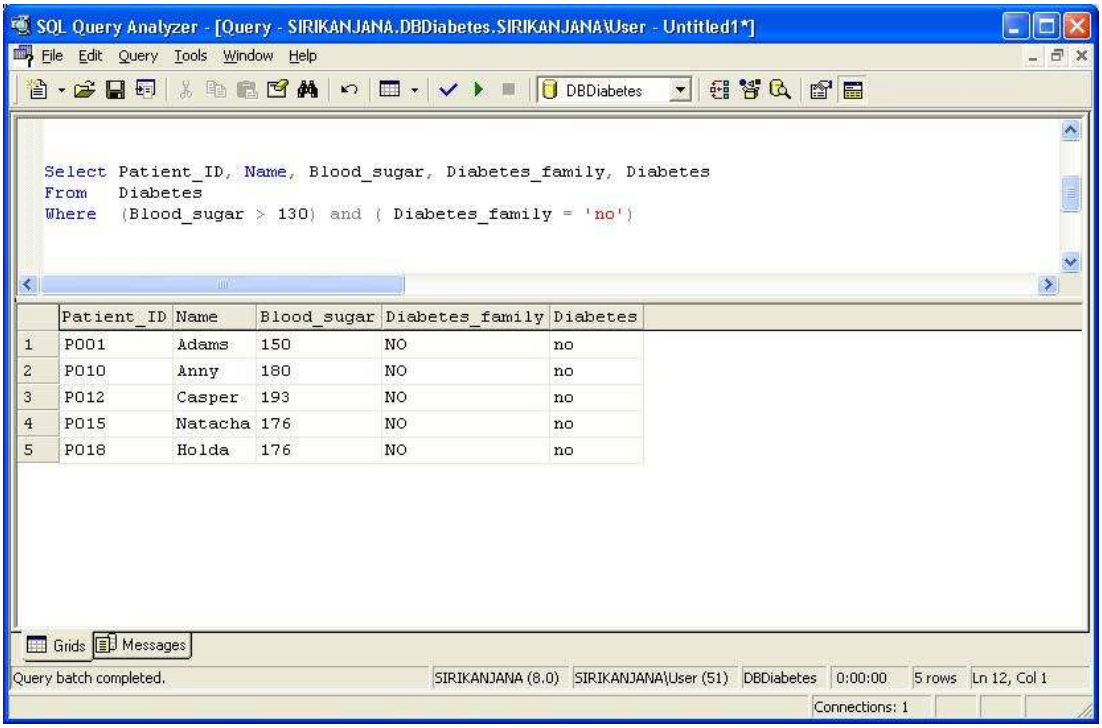
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 130 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับเงื่อนไขคือ

Diabetes_family = no, Blood_sugar > 128, Diabetes = Yes

โดยเงื่อนไขของข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 128 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ดังนั้นถ้าหากมีการเปลี่ยนแปลงข้อมูลที่ไม่ถูกต้องลงไปในฐานข้อมูล จะส่งผลให้การประมวลผลมีความผิดพลาดเกิดขึ้นได้ รูปที่ 4.18 แสดงผลการทำงานของคำสั่งทดสอบว่าสามารถแก้ไขข้อมูลในฐานข้อมูลได้ และรูปที่ 4.19 แสดงข้อมูลทั้งหมดในฐานข้อมูลจำนวน 5 เรคคอร์ด ที่ถูกปรับปรุงแก้ไขข้อมูลตามคำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าว



รูปที่ 4.18 แสดงผลการทดสอบคำสั่งที่ 3 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.19 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 3 โดยไม่มีดาต้าเบสทริกเกอร์

ทดสอบครั้งที่ 2 ทดสอบโดยมีการสร้างดาต้าเบสทริกเกอร์

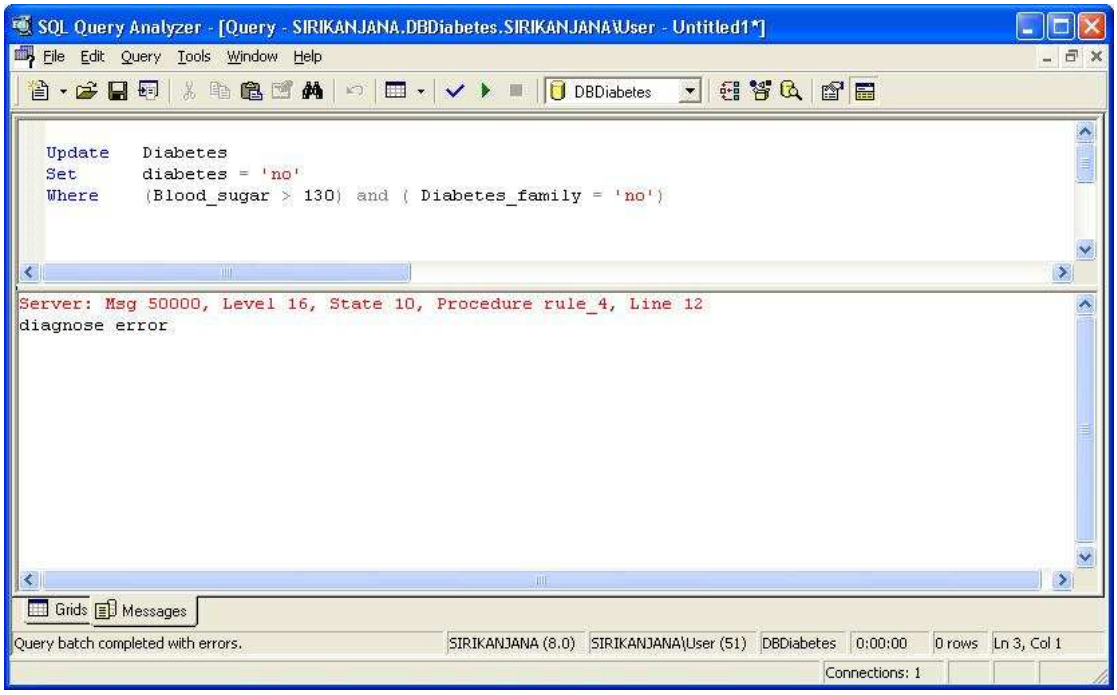
จากผลการทดสอบพบว่า คำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าวมีความขัดแย้งกับ Induced Trigger Rule จึงทำให้ไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้สำเร็จ เนื่องจากขัดแย้งกับดาต้าเบสทริกเกอร์ ซึ่งข้อมูลที่ทำการปรับปรุงแก้ไขเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกับดาต้าเบสทริกเกอร์ดังนี้

Diabetes_family = no, Blood_sugar = 130, Diabetes = No

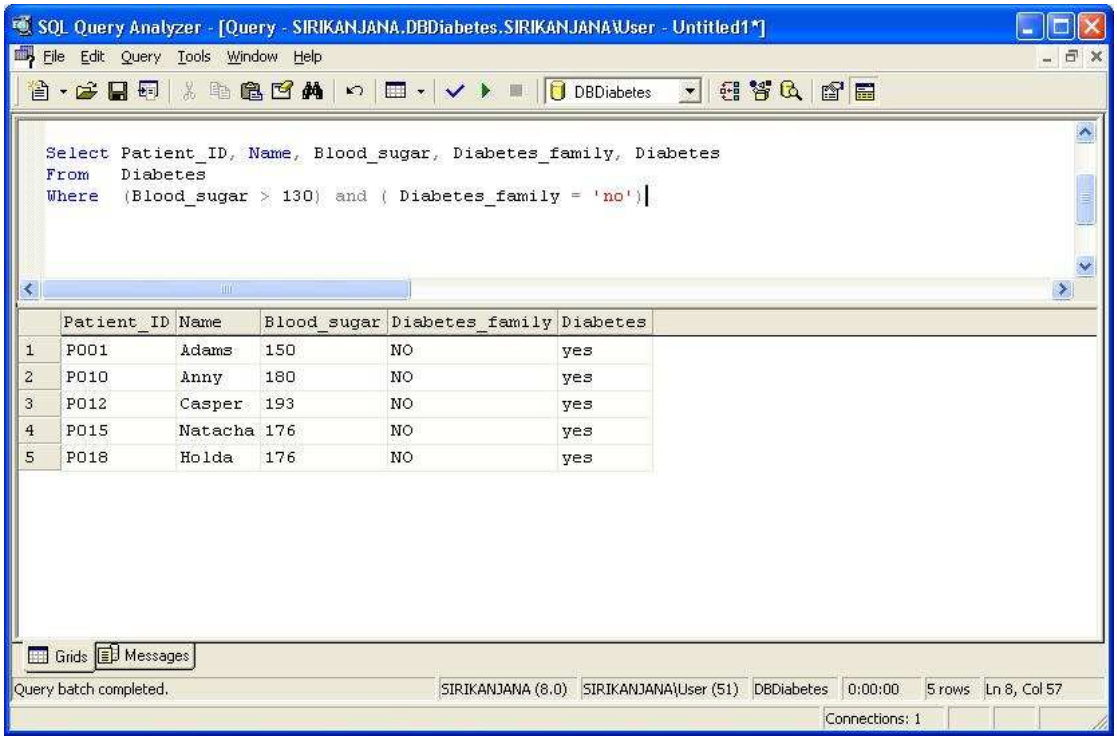
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่าเท่ากับ 130 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับคำสั่งดาต้าเบสทริกเกอร์ที่ชื่อว่า Rule_4 คือ

IF Diabetes_family = no and Blood_sugar > 128 Then Diabetes = Yes

โดยดาต้าเบสทริกเกอร์นี้มีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 128 จะส่งผลทำให้มีผลวินิจฉัยเป็นโรคเบาหวาน ซึ่งเมื่อทำการทดสอบคำสั่งในการปรับปรุงแก้ไขข้อมูลดังกล่าว จะมีผลการทำงานแจ้งให้ทราบว่าไม่สามารถปรับปรุงแก้ไขข้อมูลลงไปในฐานข้อมูลได้ โดยข้อความชุดดังกล่าวเป็นผลจากการทำงานของดาต้าเบสทริกเกอร์ที่สร้างไว้ในฐานข้อมูล ดังแสดงในรูปที่ 4.20 และข้อมูลทั้งหมดที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ไม่สามารถปรับปรุงแก้ไขข้อมูลได้จำนวน 5 เรคคอร์ด จากข้อมูลทั้งหมดจำนวน 20 เรคคอร์ด แสดงในรูปที่ 4.21



รูปที่ 4.20 แสดงผลการทดสอบคำสั่งที่ 3 โดยมีการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.21 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 3 โดยมีดาต้าเบสทริกเกอร์

คำสั่งที่ 4 คำสั่งในการปรับปรุงข้อมูลเข้าไปในตารางที่ชื่อว่า Diabetes

```
UPDATE    Diabetes
SET       Diabetes = 'yes'
WHERE     (Blood_sugar > 110) and (Diabetes_family = 'no')
```

<u>ทดสอบครั้งที่ 1</u> ทดสอบโดยไม่มีการสร้างดาต้าเบสทริกเกอร์

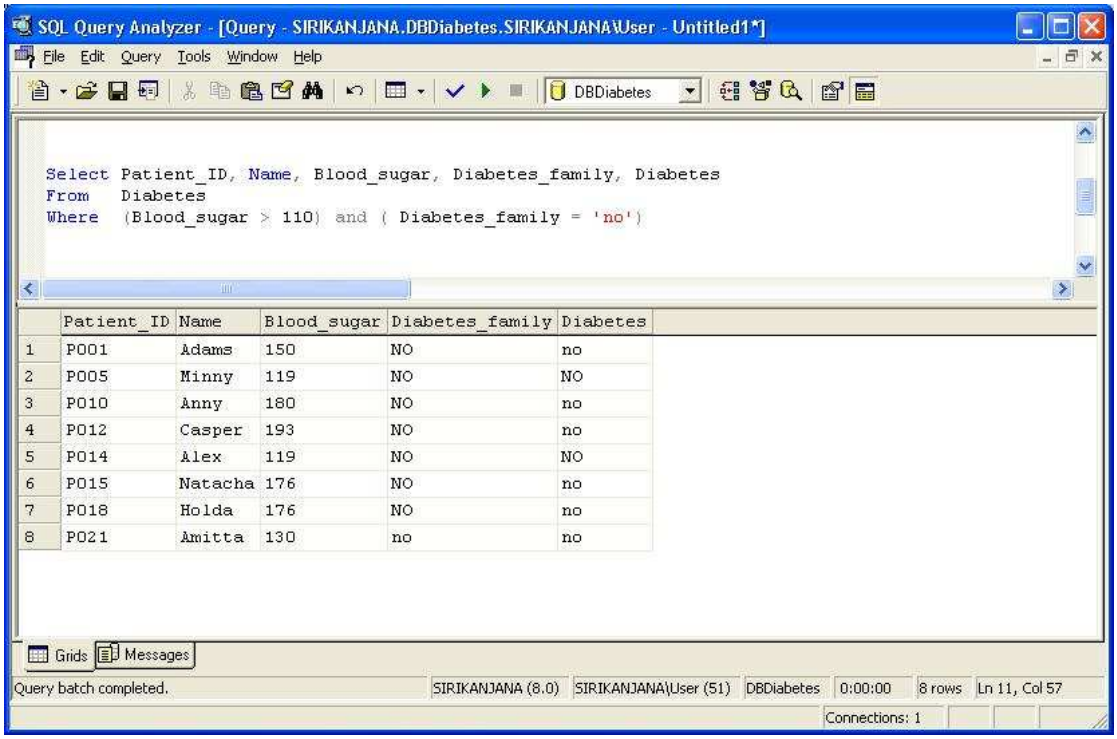
จากผลการทดสอบพบว่า ระบบฐานข้อมูลยอมให้ปรับปรุงแก้ไขข้อมูลตามคำสั่งดังกล่าวในฐานข้อมูลได้ โดยข้อมูลที่มีเงื่อนไขตรงกับคำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าวมีอยู่ทั้งหมด 8 เรคคอร์ด ดังแสดงในรูปที่ 4.22 (จากข้อมูลในการทดสอบคำสั่งที่ 3 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์) ข้อมูลชุดดังกล่าวถูกปรับปรุงแก้ไขข้อมูลใหม่ตามคำสั่งทดสอบทั้งหมด ทำให้ข้อมูลที่ทำการปรับปรุงแก้ไขเข้าไปใหม่นั้นเป็นข้อมูลที่มีความขัดแย้งกันเอง ข้อมูลดังกล่าวได้แก่

Diabetes_family = yes, Blood_sugar > 110, Diabetes = no

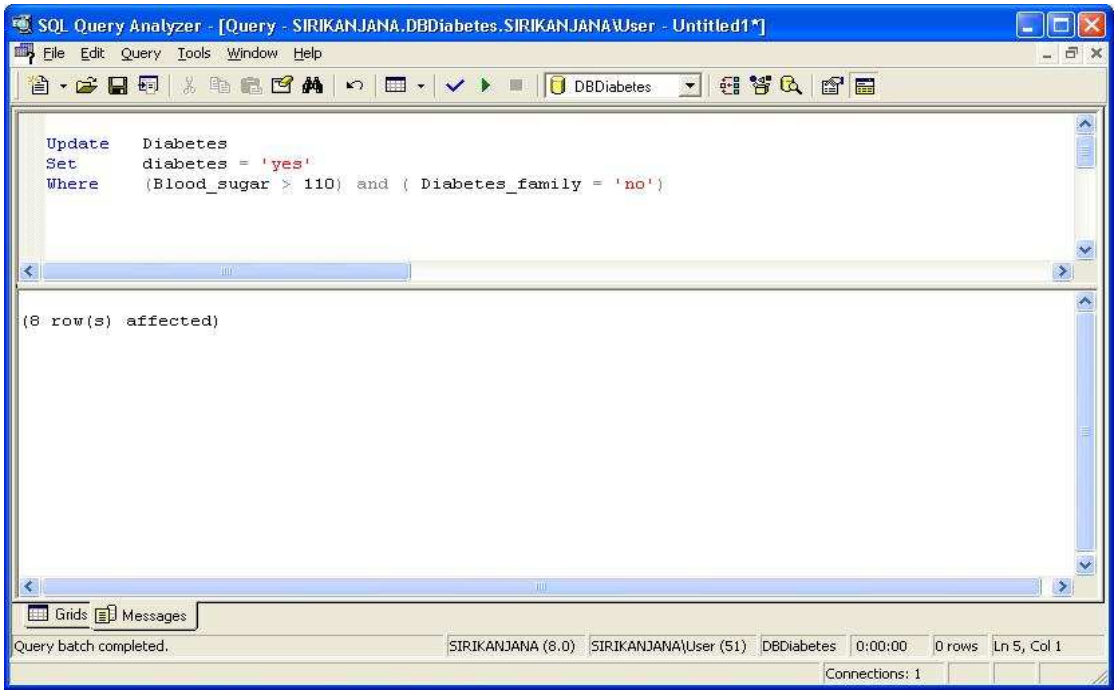
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 110 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับเงื่อนไขคือ

Diabetes_family = no, Blood_sugar <= 128, Diabetes = no

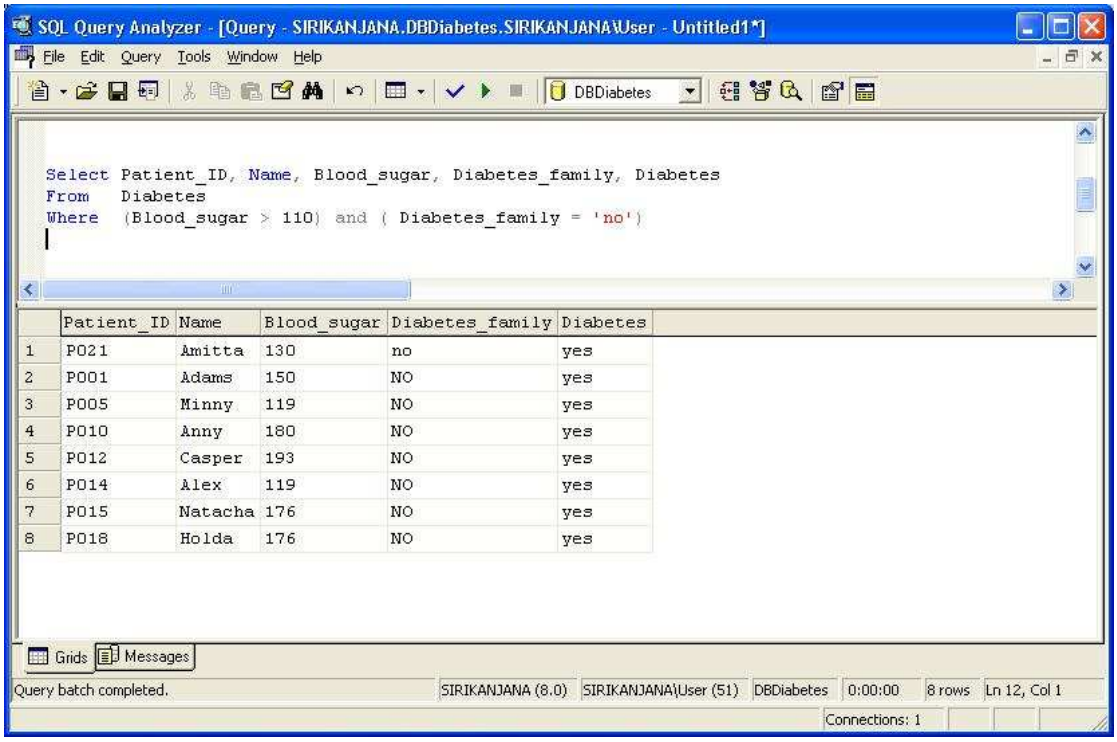
โดยเงื่อนไขของข้อมูลดังกล่าวมีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่าน้อยกว่าหรือเท่ากับ 128 จะส่งผลทำให้มีผลวินิจฉัยไม่เป็นโรคเบาหวาน ดังนั้นถ้าหากมีการเปลี่ยนแปลงข้อมูลที่ไม่ถูกต้องลงไปในฐานข้อมูล จะส่งผลให้การประมวลผลมีความผิดพลาดเกิดขึ้นได้ รูปที่ 4.23 แสดงผลการทำงานของคำสั่งทดสอบว่าสามารถแก้ไขข้อมูลในฐานข้อมูลได้ และรูปที่ 4.24 แสดงข้อมูลทั้งหมดในฐานข้อมูลจำนวน 8 เรคคอร์ด หลังจากทำการทดสอบคำสั่งปรับปรุงเปลี่ยนแปลงข้อมูล



รูปที่ 4.22 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 4 โดยไม่มีดาต้าเบสทริกเกอร์



รูปที่ 4.23 แสดงผลการทดสอบคำสั่งที่ 4 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.24 แสดงข้อมูลหลังจากทดสอบคำสั่งที่ 3 ตามเงื่อนไขคำสั่ง โดยไม่มีดาต้าเบสทริกเกอร์

ทดสอบครั้งที่ 2 ทดสอบโดยมีการสร้างดาต้าเบสทริกเกอร์

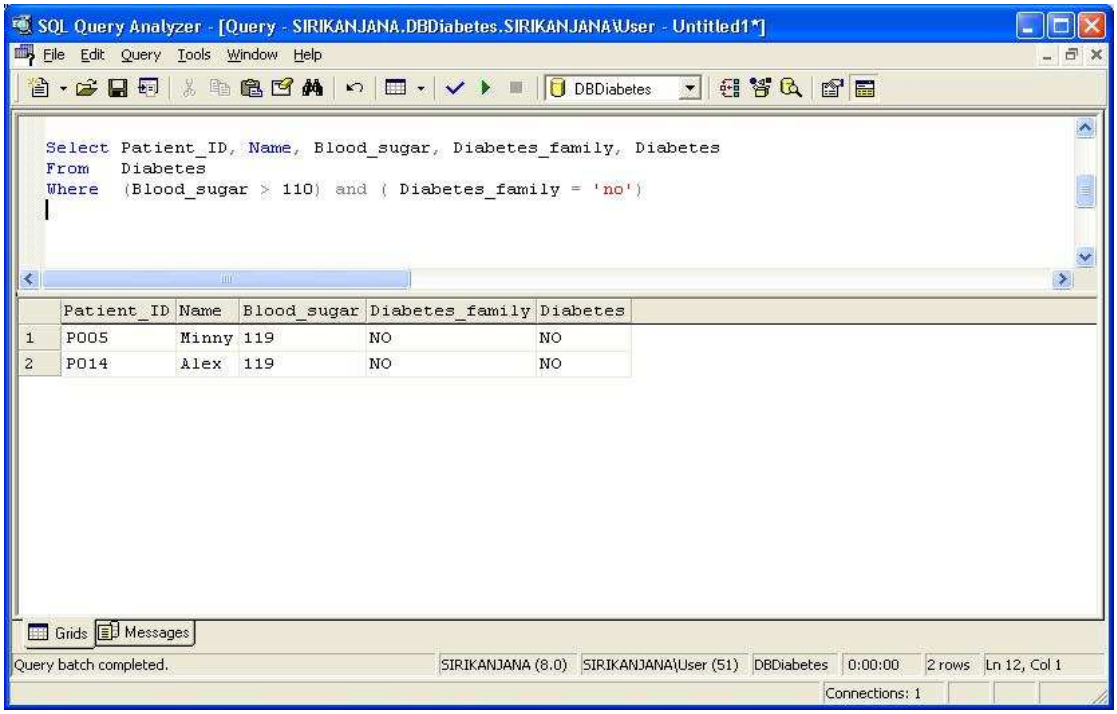
จากผลการทดสอบพบว่า คำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าวมีความขัดแย้งกับ Induced Trigger Rule จึงทำให้ไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้สำเร็จ เนื่องจากขัดแย้งกับดาต้าเบสทริกเกอร์ โดยข้อมูลที่มีเงื่อนไขตรงกับคำสั่งปรับปรุงแก้ไขข้อมูลดังกล่าวมีอยู่ทั้งหมด 2 เรคคอร์ด ดังแสดงในรูปที่ 4.25 (จากข้อมูลการทดสอบคำสั่งที่ 3 โดยมีการสร้างดาต้าเบสทริกเกอร์) ซึ่งข้อมูลที่ทำการปรับปรุงแก้ไขเข้าไปใหม่นั้น มีข้อมูลบางแอททริบิวต์ที่เป็นข้อมูลที่มีความขัดแย้งกับดาต้าเบสทริกเกอร์ดังนี้

Diabetes_family = yes, Blood_sugar > 110, Diabetes = no

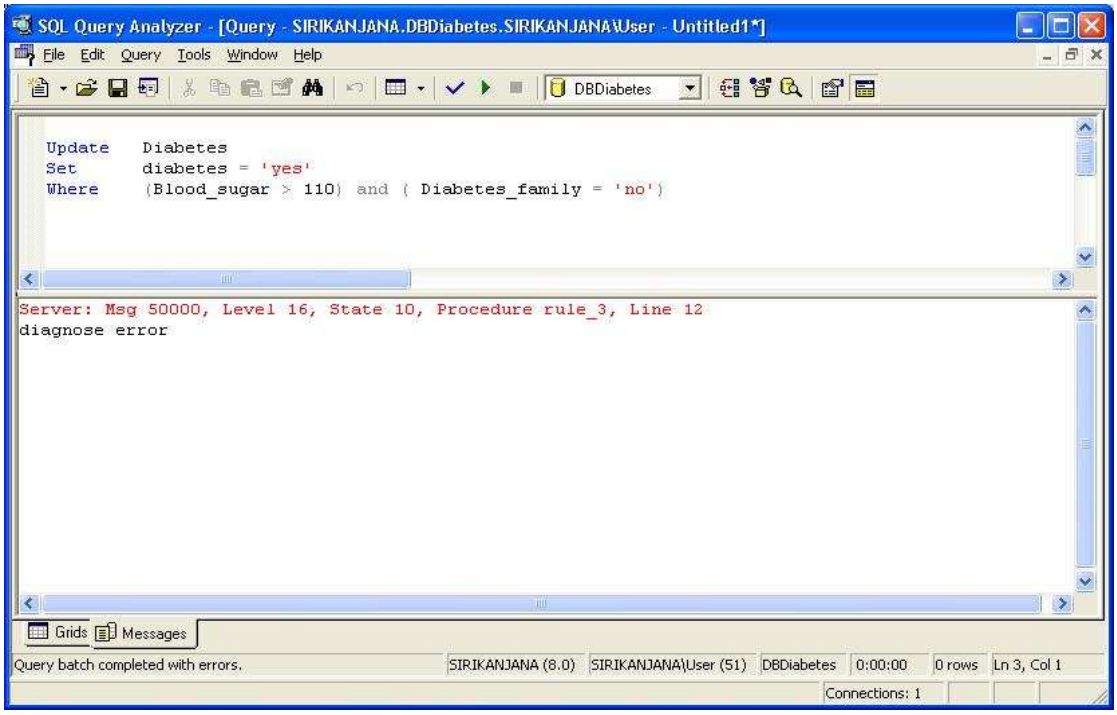
โดยข้อมูลดังกล่าวมีความหมายคือ ถ้ามีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่ามากกว่า 110 และมีผลวินิจฉัยไม่เป็นโรคเบาหวาน ซึ่งข้อมูลชุดนี้ขัดแย้งกับคำสั่งดาต้าเบสทริกเกอร์ที่ชื่อว่า Rule_3 คือ

IF Diabetes_family = no and Blood_sugar <= 128 Then Diabetes = no

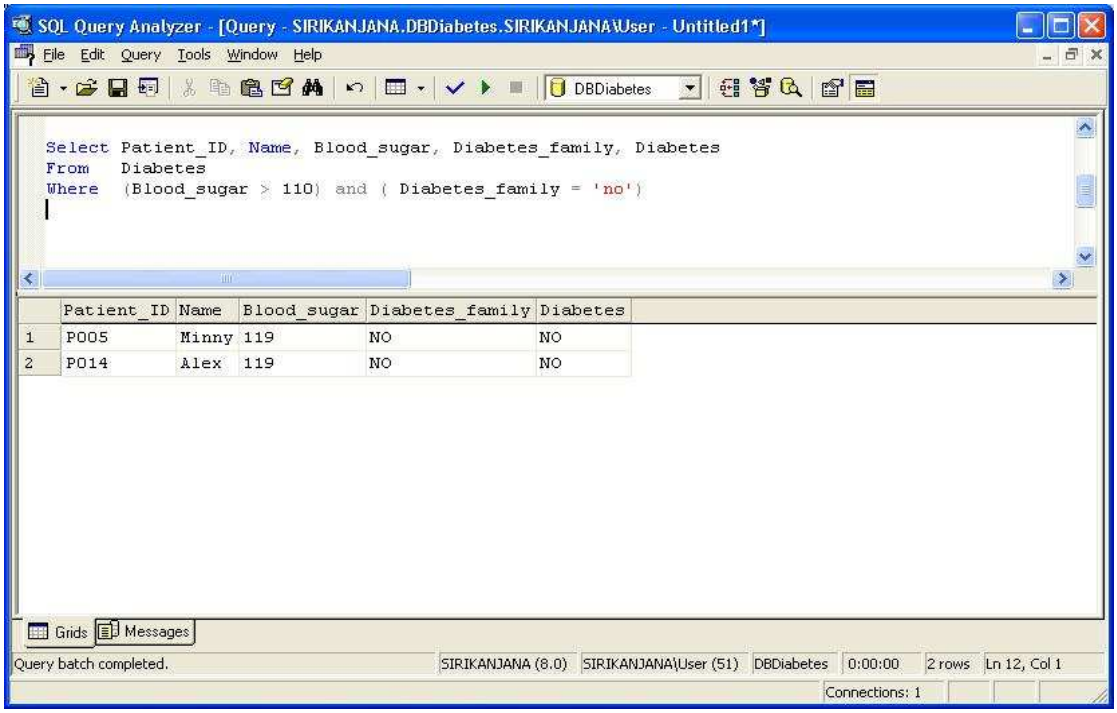
โดยดาต้าเบสทริกเกอร์นี้มีความหมายคือ ถ้าไม่มีญาติพี่น้องเป็นโรคเบาหวานและระดับน้ำตาลในเลือดมีค่าน้อยกว่าหรือเท่ากับ 128 จะส่งผลทำให้มีผลวินิจฉัยไม่เป็นโรคเบาหวาน และเมื่อทดสอบคำสั่งดังกล่าวจะมีผลการทำงานแจ้งให้ทราบว่าไม่สามารถปรับปรุงแก้ไขข้อมูลในฐานข้อมูลได้ โดยข้อความดังกล่าวเป็นผลจากการทำงานของดาต้าเบสทริกเกอร์ที่สร้างไว้ในฐานข้อมูล ดังแสดงในรูปที่ 4.26 และข้อมูลทั้งหมดที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ไม่สามารถปรับปรุงแก้ไขข้อมูลได้จำนวน 2 เรคคอร์ด จากข้อมูลทั้งหมดจำนวน 20 เรคคอร์ด แสดงในรูปที่ 4.27



รูปที่ 4.25 แสดงข้อมูลที่มีเงื่อนไขตรงกับคำสั่งทดสอบที่ 4 โดยมีดาต้าเบสทริกเกอร์



รูปที่ 4.26 แสดงผลการทดสอบคำสั่งที่ 4 โดยไม่มีการสร้างดาต้าเบสทริกเกอร์



รูปที่ 4.27 แสดงข้อมูลหลังจากทดสอบคำสั่งที่ 4 ตามเงื่อนไขคำสั่ง โดยมีดาต้าเบสทริกเกอร์

บทที่ 5
บทสรุป

ระบบจัดการฐานข้อมูลมีดาต้าเบสทริกเกอร์เป็นตัวควบคุมความถูกต้องของข้อมูลที่ทำการจัดเก็บเข้าไป ให้ตรงตามหลักของระบบฐานข้อมูลและความต้องการของแต่ละระบบงานนั้น ๆ ซึ่งถ้าหากฐานข้อมูลเก็บข้อมูลที่มีความผิดพลาดเข้าไปปริมาณมาก ๆ ก็จะส่งผลทำให้การวิเคราะห์หรือประมวลผลจากข้อมูลชุดดังกล่าวเกิดความผิดพลาดตามไปด้วย ดังนั้นจึงควรมีการตรวจสอบความถูกต้องของข้อมูลก่อนทำการจัดเก็บ หรือแก้ไขลงไปในฐานข้อมูล เพื่อให้ระบบจัดการฐานข้อมูลเก็บข้อมูลที่ถูกต้องและทำงานได้เต็มประสิทธิภาพมากที่สุด

งานวิจัยชิ้นนี้ มีจุดมุ่งหมายที่จะสร้างดาต้าเบสทริกเกอร์จากกฎ (Rules) ที่เรียนรู้จากข้อมูลในฐานข้อมูลเพื่อนำดาต้าเบสทริกเกอร์ที่ได้ไปสร้างในฐานข้อมูลเพื่อควบคุมความถูกต้องของข้อมูลที่ถูกจัดเก็บ ซึ่งในงานวิจัยชิ้นนี้จะทำการค้นหากฎประเภทการจำแนกข้อมูลในฐานข้อมูล โดยการนำเทคโนโลยีทางด้านการทำเหมืองข้อมูล โดยใช้อัลกอริทึม Classification มาใช้ในการค้นหากฎดังกล่าว หลังจากนั้นจึงนำกฎที่ได้มาสร้างเป็นดาต้าเบสทริกเกอร์ เพื่อเป็นการลดระยะเวลา เพิ่มประสิทธิภาพและความถูกต้องสูงสุดให้กับฐานข้อมูล

ขั้นตอนในการพัฒนางานวิจัยชิ้นนี้ มีการพัฒนาโปรแกรมเพื่อสร้างดาต้าเบสทริกเกอร์จากกฎการจำแนกข้อมูลขึ้นมา เพื่อช่วยในการสร้างดาต้าเบสทริกเกอร์ โปรแกรมดังกล่าวเป็นเพียงโปรแกรมต้นแบบเพื่อใช้ในขั้นตอนการทดสอบผลเท่านั้น จึงยังไม่มีความสมบูรณ์เพียงพอที่จะนำไปใช้ในระดับเชิงพาณิชย์ได้

ขั้นตอนการค้นหากฎการจำแนกและทดสอบประสิทธิภาพของดาต้าเบสทริกเกอร์ จะทำการทดสอบกับข้อมูลผู้ป่วยที่มีภาวะเสี่ยงต่อการเป็นโรคเบาหวาน โดยทำการค้นหากฎด้วยโปรแกรม WEKA (Witten and Frank, 2005) โดยใช้อัลกอริทึม J48 หลังจากนั้นจะนำกฎที่ได้มาสร้างเป็นดาต้าเบสทริกเกอร์ด้วยโปรแกรมที่ผู้วิจัยพัฒนาขึ้น และทำการทดสอบการทำงานของดาต้าเบสทริกเกอร์ ซึ่งจะทดสอบกับระบบจัดการฐานข้อมูล Microsoft SQL Server 2000 โดยการเปรียบเทียบการทำงานของคำสั่งเพิ่มหรือปรับปรุงแก้ไขข้อมูล ในกรณีที่ไม่มีการสร้างและมีการสร้างดาต้าเบสทริกเกอร์

5.1 สรุปผลการวิจัย

ผลการทดสอบสรุปได้ดังนี้

1) ในการนำดาต้าเบสทริกเกอร์ที่สร้างได้จากกฎประเภทการจำแนกข้อมูล มาสร้างไว้ในระบบฐานข้อมูล พบว่ารูปแบบของดาต้าเบสทริกเกอร์ดังกล่าวถูกต้องและสามารถใช้งานในการตรวจสอบความถูกต้องของข้อมูลได้จริง และมีประโยชน์ที่ไม่เพียงแต่เป็นการสร้างความถูกต้องสูงสุดกับฐานข้อมูลเท่านั้น แต่ยังเป็นการป้องกันความผิดพลาดในกรณีที่ผู้ใช้หรือ User ทำการปรับปรุงเปลี่ยนแปลงฐานข้อมูล เพื่อไม่ให้ข้อมูลที่บันทึกเข้าไปขัดแย้งกันเอง เมื่อข้อมูลที่บันทึกเข้าไปมีปริมาณมากขึ้นก็จะใช้ข้อมูลชุดใหม่ ๆ นั้นเป็นความรู้พื้นฐานเพื่อที่จะทำให้เกิดรูปแบบหรือกฎใหม่ ๆ และนำกฎที่ได้ขึ้นมาใหม่นั้นไปสร้างดาต้าเบสทริกเกอร์ต่อไป

2) การทดสอบการเปลี่ยนแปลงของข้อมูลในระบบฐานข้อมูล ในกรณีที่ไม่มีการสร้างดาต้าเบสทริกเกอร์ พบว่าข้อมูลที่ใช้ทดสอบนั้นสามารถจัดเก็บและปรับปรุงแก้ไขได้ทั้งหมดโดยไม่มีการตรวจสอบความถูกต้อง ทำให้ข้อมูลที่ทำการปรับปรุงแก้ไขเข้าไปใหม่นั้นอาจเป็นข้อมูลที่มีความขัดแย้งกันเอง จึงอาจส่งผลให้การประมวลผลมีความผิดพลาดเกิดขึ้นได้

3) การทดสอบการเปลี่ยนแปลงของข้อมูลในระบบฐานข้อมูล ในกรณีที่มีการสร้างดาต้าเบสทริกเกอร์ พบว่าข้อมูลที่ใช้ทดสอบนั้นไม่สามารถจัดเก็บและปรับปรุงแก้ไขได้ เนื่องจากถูกตรวจสอบความถูกต้องของข้อมูลด้วยดาต้าเบสทริกเกอร์ ส่งผลให้ข้อมูลที่ผิดพลาดและมีความขัดแย้งไม่สามารถถูกจัดเก็บลงไปในฐานข้อมูล ทำให้ในฐานข้อมูลมีเฉพาะข้อมูลที่มีความถูกต้องเท่านั้น

5.2 การประยุกต์งานวิจัย

งานวิจัยชิ้นนี้เป็นการนำเทคโนโลยีสองด้านที่มีอยู่แล้วในปัจจุบัน ได้แก่การทำเหมืองข้อมูล และการควบคุมความถูกต้องของฐานข้อมูลด้วยดาต้าเบสทริกเกอร์ ผู้วิจัยได้นำเทคโนโลยีที่รู้จักกันอย่างแพร่หลายมาทำงานร่วมกันจนเกิดเป็นแนวทางใหม่ในการพัฒนาการเพิ่มประสิทธิภาพด้านการจัดเก็บข้อมูลที่ถูกต้องของระบบจัดการฐานข้อมูล เพื่อให้มีประสิทธิภาพในการทำงานมากยิ่งขึ้น งานวิจัยชิ้นนี้จึงเป็นแนวทางอีกแนวทางหนึ่งในการคิดค้นวิธีที่จะสร้างดาต้าเบสทริกเกอร์โดยการนำกฎการจำแนกที่ได้จากการทำเหมืองข้อมูลมาใช้ประโยชน์ เพื่อนำไปใช้ในการสร้างดาต้าเบสทริกเกอร์ ซึ่งงานวิจัยชิ้นนี้ยังคงเป็นเพียงแนวทางเบื้องต้น ดังนั้นจึงสามารถทำการวิจัยเพื่อพัฒนาต่อไปให้มีประสิทธิภาพมากยิ่งขึ้น โดยเฉพาะอย่างยิ่งในประเด็นของการจัดการกับ Induced trigger rules เมื่อข้อมูลในฐานข้อมูลมีการเปลี่ยนแปลง จึงน่าจะเป็นประโยชน์สำหรับนักวิจัยในอนาคตเพื่อเป็นจุดเริ่มต้นในการพัฒนางานวิจัยลักษณะนี้ต่อไป และเมื่อมีการพัฒนาต่อไปที่ดีขึ้นแล้วอาจนำเอาความรู้ใหม่นี้มาประยุกต์รวมเข้ากับระบบจัดการฐานข้อมูลต่อไปในอนาคตได้

5.3 ปัญหาและข้อเสนอแนะ

1) การทดสอบประสิทธิภาพการทำงานของดาต้าเบสทริกเกอร์ในครั้งนี้ ยังคงเป็นเพียงการใช้คำสั่งทดสอบการเปลี่ยนแปลงแก้ไขกับชุดข้อมูลเพียงหนึ่งตาราง ดังนั้นจึงต้องสร้างตารางดังกล่าวขึ้นมาเพื่อใช้งานจริง ควบคู่กับตารางที่จัดเก็บข้อมูลที่มีอยู่เดิมทั้งหมด

2) การสร้างดาต้าเบสทริกเกอร์เข้าไปในฐานข้อมูลในปริมาณมาก ๆ ซึ่งสร้างได้จากกฎการจำแนกที่ได้จากข้อมูลที่ถูกจัดเก็บเข้าไปเรื่อย ๆ อาจส่งผลทำให้มีดาต้าเบสทริกเกอร์จำนวนมากเกินความจำเป็น ดังนั้นจึงควรที่จะจำกัดหรือลดปริมาณดาต้าเบสทริกเกอร์ที่ไม่ค่อยได้ใช้งานออกไป โดยการเก็บสถิติของการใช้งานดาต้าเบสทริกเกอร์แต่ละตัว

3) ในกรณีที่มีการเพิ่มข้อมูลใหม่ในฐานข้อมูล เมื่อตรวจสอบโดยผู้เชี่ยวชาญแล้วพบว่าข้อมูลนั้นถูกต้อง แต่ข้อมูลมีเงื่อนไขขัดแย้งกับดาต้าเบสทริกเกอร์ที่สร้างไว้แล้วก่อนหน้านั้น ควรจะมีการอนุญาตให้ข้อมูลดังกล่าวเพิ่มเข้ามาในฐานข้อมูลได้ เพื่อให้ข้อมูลนั้นเป็นข้อมูลความรู้ใหม่ในการทำเหมืองข้อมูล โดยการยกเลิกดาต้าเบสทริกเกอร์ที่มีอยู่เดิมชั่วคราวเพื่อให้ข้อมูลที่ขัดแย้งสามารถเพิ่มเข้าไปได้ จากนั้นนำข้อมูลในฐานข้อมูลทั้งหมดมาทำเหมืองข้อมูลประเภทการจำแนกให้ได้ผลลัพธ์เป็นกฎใหม่ ๆ ออกมา เพื่อนำไปใช้สร้างดาต้าเบสทริกเกอร์ ที่สามารถนำไปใช้ประโยชน์เสมือนว่าเป็นตัวช่วยสร้าง Business rules ที่ทันสมัยอยู่เสมอ

รายการอ้างอิง

คณะเภสัชศาสตร์ มหาวิทยาลัยศิลปากร (2008). **ประสิทธิภาพของการลดระดับน้ำตาลในเบาหวาน** [ออนไลน์]. ได้จาก: http://www.pharm.su.ac.th/thai/Organizations/DIS/showQAnswer.asp?qNo=226

วิทยาลัยวิทยาศาสตร์และเทคโนโลยีการกีฬา มหาวิทยาลัยมหิดล (2008). **ค่าดัชนีมวลกาย** [ออนไลน์]. ได้จาก: http://www.ss.mahidol.ac.th/thai/Bmi.htm

สมาคมโรคเบาหวานแห่งประเทศไทยในพระราชูปถัมภ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี (2007). **ความรู้ทั่วไปเรื่องเบาหวาน** [ออนไลน์]. ได้จาก: http://www.diabassocthai.org/news_detail.php?news_id=10

American National Standard Institute. (2009). **X3H2 Records 1978-1995** [On-line]. Available: http://special.lib.umm.edu/findaid/xml/cbi00168.xml

Chamberlin, D., and Boyce R. (1974). SEQUEL: A structured English query language. **Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control**, pp. 249-264.

De Raedt, L. (2002). A perspective on inductive database. **SIGKDD Explorations,** Vol.4, No.2, pp. 69-77.

Elmasri, R., and Navathe, S. B. (1994). **Fundamentals of Database System,** The California: Benjamin/Cummings Publishing Company, pp. 137-222.

Garofalakis, M., Hyun, D., Rastogi, R., and Shim, K. (2000). Efficient algorithms for constructing decision trees with constraint. **Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.** pp. 335-339.

Gehrke, J., Ramakrishnan, R., and Ganti, V. (1998). RainForest – A framework for fast decision tree construction of large datasets. **Proceedings of the 24th VLDB Conference.** New York, USA. pp. 127-162.

Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., and Zhuge, Y. (1995). The Stanford data warehousing project. **Proceedings of IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing**, Vol.18, No.2, pp. 41-48.

Han, J., Fu, Y., Wang, W., Koperski, K., and Zaiane, O. (1996). DMQL: A data mining query language for relational databases. **Proceedings of SIGMOD DMKD Workshop**, Montreal, Canada. pp. 27-33.

Holte, R.C. (1993). Very simple classification rules perform well on most commonly used datasets. **Machine Learning**, Vol. 11, pp. 63-90.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. **Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.** pp. 97-106.

Imielinski, T., and Virmani, A. (1999). MSQL: A query language for database mining. **Data Mining and Knowledge Discovery**, Vol. 3, pp. 373–408.

Imielinski, T., and Mannila, H. (1996). A database perspective on knowledge discovery. **Communications of the ACM,** Vol. 39, pp. 58–64.

Inmon, W.H. (1992a). Building the data bridge: The ten critical success factors of building a data warehouse. **Database Programming & Design.** Vol.5, No.11, pp. 68-69.

Inmon, W.H. (1992b). EIS and the data warehouse: A simple approach to building an effective foundation for EIS. **Database Programming & Design,** Vol.5, No.11, pp. 70-73.

Jin, R., and Agrawal, G. (2003). Efficient decision tree construction on streaming data. **Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery.** pp. 571-576.

Larose, Daniel T. (2005). **Discovering Knowledge in Data: An Introduction to Data Mining.** New Jersey: John Wiley & Sons, Inc.

Loh, W., and Shih, Y. (1997). Split selection Methods for classification trees. **Statistica Sinica**. Vol.7, No.4, pp. 815-840.

Mehta, M., Agrawal, R., and Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. **Proceedings of the 5th International Conference on Extending Database Technology.** pp. 18-32.

Mehta, M., Rissanen, J., and Agrawal, R. (1995). MDL-based decision tree pruning. **Proceedings of KDD-95**. Montreal, Canada. pp. 216-221.

Meo, R., Psaila, G., and Ceri, S. (1998). An extension to SQL for mining association rules. **Data Mining and Knowledge Discovery,** Vol. 2, pp. 195-224.

Microsoft Developer Network: MSDN. (2007). Create trigger [On-line]. Available: http://msdn.microsoft.com/en-us/library/aa258254(SQL.80).aspx

Quinlan, J. (1986). Induction of decision trees. **Machine Learning**, Vol.1, No.1, pp. 81-106.

Quinlan, J. (1992). **C4.5: Programs for Machine Learning**, San Francisco: Morgan Kaufmann.

Sattler, K., and Dunemann, O. (2001). SQL Database primitives for decision tree classifiers. **Proceedings of the Tenth International Conference on Information and Knowledge Management**, pp. 379-386.

Shafer, J., Agrawal, R., and Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. **Proceedings of VLDB Conference**. pp. 544-555.

Witten, I.H., and Frank, E. (2005). **Data Mining: Practical Machine Learning Tools and Techniques,** 2nd edition. San Francisco: Morgan Kaufmann.

ภาคผนวก ก

บทความผลงานวิจัยที่นำเสนอในการประชุมเสนอผลงานวิจัย
ระดับบัณฑิตศึกษาแห่งชาติ ครั้งที่ 11 วันที่ 17-18 ธันวาคม 2551
มหาวิทยาลัยราชภัฏวไลยอลงกรณ์ ในพระบรมราชูปถัมภ์ จ.ปทุมธานี

การสร้างกฎข้อบังคับการเปลี่ยนแปลงฐานข้อมูลโดยวิธีการทำเหมืองข้อมูล

ศิริกาญจนา พิลาบุตร, รศ. ดร.นิตยา เกิดประสพ และ รศ. ดร.กิตติศักดิ์ เกิดประสพ

สาขาวิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี อ.เมือง จ.นครราชสีมา 30000

บทคัดย่อ

ระบบจัดการฐานข้อมูลเชิงสัมพันธ์มีดาต้าเบสทริกเกอร์ (Database Trigger) ทำหน้าที่ตรวจสอบข้อบังคับของข้อมูลในการประมวลผลคำสั่ง SQL ประเภท DML (Data Manipulation Language) ซึ่งเป็นคำสั่งที่ใช้ในการเปลี่ยนแปลงข้อมูลในฐานข้อมูล ให้มีความถูกต้องตามความต้องการของระบบงาน ในงานวิจัยนี้เสนอแนวคิดในการสร้างดาต้าเบสทริกเกอร์ โดยใช้กฎที่ได้จากการทำเหมืองข้อมูลประเภทการจำแนก มาเป็นตัวสร้างดาต้าเบสทริกเกอร์ด้วยวิธีกึ่งอัตโนมัติ เพื่อเป็นการลดระยะเวลาในการสร้างดาต้าเบสทริกเกอร์ขึ้นมาเองโดยผู้จัดการฐานข้อมูลและเพิ่มความถูกต้องสูงสุดให้กับฐานข้อมูลเมื่อมีเหตุการณ์เปลี่ยนแปลงข้อมูล

คำสำคัญ: ฐานข้อมูลเชิงอุปนัย, ดาต้าเบสทริกเกอร์, การจำแนก, การทำเหมืองข้อมูล

บทนำ

ในปัจจุบันระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database Management System : RDBMS) เป็นระบบฐานข้อมูลที่มีผู้นิยมใช้กันมาก ในการจัดเก็บข้อมูลลงในฐานข้อมูลให้มีความถูกต้องตามหลักของระบบฐานข้อมูลและความต้องการของแต่ละระบบงาน (Business Rule) เป็นเรื่องยากหากจะต้องป้องกันความผิดพลาดดังกล่าวโดยการเขียนโปรแกรม การจัดเก็บข้อมูลที่ไม่ถูกต้องตามระบบงานลงในฐานข้อมูลจะส่งผลให้การประมวลผลจากข้อมูลดังกล่าวเกิดความผิดพลาด และต้องใช้เวลาในการแก้ไขจัดการข้อมูลให้มีความถูกต้อง (Elmasri and Navathe, 1994; Inmon, 1992a; 1992b) เพื่อเป็นการลดความผิดพลาดดังกล่าว ระบบจัดการฐานข้อมูลจึงมีดาต้าเบสทริกเกอร์ (Database Trigger) ซึ่งทำหน้าที่ตรวจสอบข้อบังคับของข้อมูลในการประมวลผลคำสั่ง SQL ประเภท DML ซึ่งเป็นคำสั่งที่ใช้ในการเปลี่ยนแปลงข้อมูลในฐานข้อมูล ให้มีความถูกต้องตามความต้องการของระบบงาน

ปัจจุบันการสร้างดาต้าเบสทริกเกอร์ให้ตรงตามความต้องการของระบบงานนั้น กระทำได้ด้วยผู้ดูแลจัดการฐานข้อมูลเป็นผู้กำหนดความถูกต้องของข้อมูล โดยพิจารณาจากความต้องการของระบบงาน ซึ่งอาจจะเกิดการผิดพลาดและสิ้นเปลืองเวลาได้หากมีการเปลี่ยนแปลงความต้องการของระบบงานขึ้นใหม่ ดังนั้นในงานวิจัยนี้จึงเสนอแนวคิดในการสร้างดาต้าเบสทริกเกอร์ โดยใช้ผลลัพธ์ที่ได้จากการทำเหมืองข้อมูลด้วยอัลกอริทึม Classification ซึ่งก็คือกฎ (Rule) มาเป็นตัวสร้างดาต้าเบสทริกเกอร์ เพื่อนำไปใช้ให้เกิดความถูกต้องสูงสุดของการเก็บข้อมูล การนำความสามารถด้านการทำเหมืองข้อมูลมาผนวกเข้ากับฐานข้อมูลทำให้เกิดเป็นระบบฐานข้อมูลรูปแบบใหม่เรียกว่าฐานข้อมูลเชิงอุปนัย (Inductive Database) (Imielinski and Mannila, 1996)

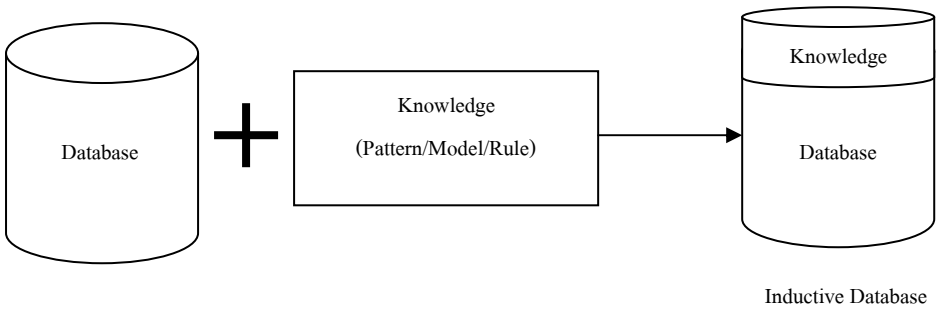
วัตถุประสงค์

1. เพื่อศึกษาการทำงานของดาต้าเบสทริกเกอร์ในระบบฐานข้อมูลเชิงสัมพันธ์
2. เพื่อศึกษาและประยุกต์การทำเหมืองข้อมูลโดยใช้อัลกอริทึม Classification ที่ใช้ในการสร้างกฎ (rule)
3. เพื่อศึกษาการนำกฎไปใช้ในระบบฐานข้อมูลเชิงสัมพันธ์
4. เพื่อสร้างดาต้าเบสทริกเกอร์จากกฎที่สร้างได้จากการทำเหมืองข้อมูล

วิธีการวิจัย

1. Inductive Database and Database Trigger

Inductive Databases เป็นฐานข้อมูลที่ไม่เพียงแต่เก็บข้อมูลในตารางต่าง ๆ เพียงอย่างเดียวเท่านั้น แต่ยังเก็บข้อมูลในส่วนของ concept, Pattern, Rule หรือเรียกอีกอย่างหนึ่งว่า model ผูกติดรวมไว้กับฐานข้อมูลเดิม (Hammer และคณะ, 1995; Luc De Raedt, 2002) เพื่อใช้เป็นประโยชน์ในการค้นหาข้อมูลหรือตอบข้อคำถาม (query) จากฐานข้อมูลที่เก็บไว้เป็นจำนวนมาก หรือช่วยในการทำเหมืองข้อมูล (Data mining) เพื่อให้ได้ประโยชน์และความรู้จากข้อมูลที่เก็บไว้สูงสุด ดังแสดงในภาพที่ 1



ภาพที่ 1 แสดงโครงสร้างสถาปัตยกรรมของฐานข้อมูลเชิงอุปนัย

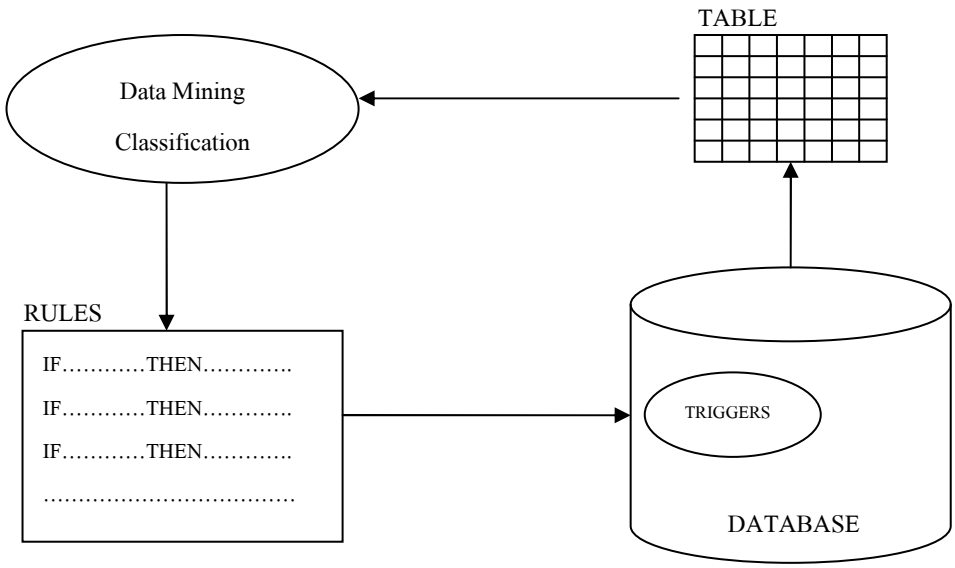
Han และคณะ (1996) นำแนวคิดของ Inductive Database มาพัฒนาต่อเพื่อใช้ประโยชน์ในการทำเหมืองข้อมูลร่วมกับภาษา SQL ซึ่งภาษา SQL นั้นเป็นภาษาที่มีความนิยมสูงในการนำมาใช้เข้าถึงข้อมูลกับระบบฐานข้อมูลเชิงสัมพันธ์ โดยปรากฏแนวคิดนี้ในงานวิจัยจำนวนมาก (Imielinski and Virmani, 1999; Meo และคณะ, 1998)

ดาต้าเบสทริกเกอร์เป็น Stored Procedure ชนิดพิเศษที่จะทำงานเมื่อมีการประมวลผลคำสั่งSQL ประเภท DML ซึ่งก็คือเกิดการเพิ่ม แก้ไข หรือลบข้อมูลในตารางขึ้น ซึ่งถ้าการทำงานของทริกเกอร์ไม่ผ่านขึ้นมา ก็จะทำให้การเพิ่ม แก้ไข หรือลบข้อมูลที่มำให้เกิดทริกเกอร์นั้นไม่ผ่านด้วย และจะมีข้อผิดพลาดส่งออกมาให้ทราบถึงข้อมูลที่ไม่ถูกต้อง เพื่อดำเนินการแก้ไขต่อไป โดยดาต้าเบสทริกเกอร์มีรูปแบบคำสั่งดังนี้

รูปแบบดาต้าเบสทริกเกอร์

```
CREATE TRIGGER Trigger_name
ON Table_name
FOR {INSERT, UPDATE, DELETE}
AS
BEGIN
        Transact-SQL
END
```

ในงานวิจัยชิ้นนี้ได้ใช้แนวคิดของ Inductive Database โดยใช้ในส่วนของ Knowledge ในลักษณะ IF condition THEN specified-class ซึ่งก็คือกฎ (Rule) ซึ่งเป็นผลที่ได้จากการทำเหมืองข้อมูลโดยใช้อัลกอริทึมในกลุ่ม Classification ของระบบ WEKA (Witten and Frank, 2005) มาเป็นเครื่องมือช่วยในการสร้าง Database Trigger ในระบบฐานข้อมูลเชิงสัมพันธ์โดยมีขอบข่ายงาน (Framework) ของระบบดังภาพที่ 2



ภาพที่ 2 แสดงขอบข่ายงานของระบบช่วยสร้างดาต้าเบสทริกเกอร์

2. อัลกอริทึม Classification

Data Classification เป็นปัญหาพื้นฐานของการเรียนรู้แบบมีผู้สอน (supervised learning) โดยปัญหาคือการทำนายประเภทของวัตถุจากคุณสมบัติต่าง ๆ ของวัตถุ ซึ่งการเรียนรู้แบบมีผู้สอนจะสร้างฟังก์ชั่นเชื่อมโยง ระหว่างคุณสมบัติของวัตถุ กับประเภทของวัตถุจากตัวอย่างสอน แล้วจึงใช้ฟังก์ชันนี้ทำนายประเภทของวัตถุที่ไม่เคยพบ หรืออาจกล่าวได้ว่าเป็นกระบวนการสร้างโมเดลจัดการข้อมูลให้อยู่ในกลุ่มที่กำหนดมาให้ เพื่อแสดงให้เห็นความแตกต่างระหว่าง class หรือ กลุ่มของข้อมูลได้ และเพื่อทำนายว่าข้อมูลนี้ ควรจัดอยู่ใน classใด ซึ่งโมเดลที่ใช้จำแนกข้อมูลออกเป็นกลุ่มตามที่ได้กำหนดไว้ จะขึ้นอยู่กับการวิเคราะห์เซตของข้อมูลทดลอง (Training data) โดยนำ Training data มาสอนให้ระบบเรียนรู้ว่ามีข้อมูลใดอยู่ในclass เดียวกันบ้าง

อัลกอริทึมแรกของงาน Classification เรียกว่า simple-rule algorithm เป็นอัลกอริทึมอย่างง่ายที่ใช้ในการสร้าง classification rules ที่เรียกว่า simple-rule เนื่องจากอัลกอริทึมนี้จะสร้าง classification rule ในรูปแบบ “IF condition Then specified-class” โดยจะมีเพียง attribute เดียวปรากฏใน condition เช่น IF Blood_Sugar > 120 Then Diabetes = yes ดังแสดงในภาพที่ 3

For each attribute,

For each value of that attribute, make a rule as follows:

Count how often each class appears.

Find the most frequent class.

Make the rule assign that class to this attribute-value.

Calculate the error rate of the rules.

Choose the rules with the smallest error rate.

ภาพที่ 3 แสดงอัลกอริทึม Simple-Rule

ในงานวิจัยชิ้นนี้ได้ใช้อัลกอริทึม J48 ซึ่งเป็นอัลกอริทึมในกลุ่มของ Classification ของระบบ WEKA มาเป็นเครื่องมือในการสร้าง Database Trigger ซึ่งอัลกอริทึม J48 นี้ใช้หลักการเดียวกันกับอัลกอริทึม C4.5 ซึ่งเป็นอัลกอริทึมที่พัฒนามาจากอัลกอริทึม ID3 (Quinlan, 1992) ที่ใช้สำหรับสร้างต้นไม้เพื่อการตัดสินใจหรือเรียกว่า decision tree เช่นเดียวกันกับอัลกอริทึม CART (Larose, 2005)

อัลกอริทึม C4.5 นี้ใช้หลักการของ Information gain หรือที่เรียกว่า entropy reduction มาใช้เพื่อจำแนกกิ่ง (node) ของต้นไม้ (tree) เกณฑ์ที่ใช้ช่วยประกอบการเลือก attribute คือ ทดลองเลือกแต่ละ attribute มาทำหน้าที่เป็น root node และวัดค่า gain ซึ่งเป็นค่าที่ชี้ว่า attribute นั้นจะช่วยจำแนกคลาสของข้อมูลได้ดีเพียงใด โดยการจำแนกที่ดีที่สุด คือให้ leaf node ที่เป็นข้อมูลเดียวกันทั้งหมด และค่า gain ที่สูงที่สุดหมายถึง การจำแนกคลาสที่ดีที่สุด

ค่า gain เป็นค่าที่บอกระดับความสามารถของการจำแนกคลาสของ attribute ที่ถูกเลือกให้ทำหน้าที่เป็นตัวตรวจสอบเพื่อจัดกลุ่มของข้อมูลในแต่ละ leaf node เป็นคลาสเดียวกันทั้งหมด หรือมีข้อมูลต่างคลาสปะปนกันมาบ้างเพียงเล็กน้อย สามารถหาค่า gain ได้ดังนี้

$$gain(X) = \text{info}(T) - \text{info}_x(T)$$

โดยกำหนดให้

T แทนเซตของข้อมูลฝึก (training data)

X แทน attribute ที่ถูกเลือกให้เป็นตัวตรวจสอบเพื่อจัดกลุ่มข้อมูล

$\text{info}(T)$ คือฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อสามารถจำแนกคลาสของข้อมูลได้

$$\text{info}(T) = -\sum_{j=1 \text{ to } K}[freq(C_j, T)/|T|] \times \log_2[freq(C_j, T)/|T|]$$

โดยกำหนดให้

$|T|$ คือจำนวนข้อมูลทั้งหมดในเซตของข้อมูลฝึก

$freq(C_j, T)$ คือความถี่ที่ข้อมูลใน T ปรากฏเป็นคลาส C_j

$\text{info}_x(T)$ คือฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อการจำแนกคลาสของข้อมูล โดยใช้ attribute X เป็นตัวตรวจสอบเพื่อจำแนกกลุ่มของข้อมูล

$$\text{info}_x(T) = -\sum_{i=1 \text{ to } K}(|T_i|/|T|) \times \text{info}(T_i)$$

โดยกำหนดให้

i คือจำนวนค่าที่เป็นไปได้ของ attribute X

$|T_i|$ คือจำนวนข้อมูลที่มีค่า $X = i$

ผลลัพธ์ที่ได้จากการเรียนรู้ คือ โมเดลจำแนกประเภทข้อมูล (classifier model) สามารถแทนได้ในหลายรูปแบบ เช่น Classification (IF-THEN) rules, Decision Tree, Mathematical formula หรือ Neural networks และจะนำข้อมูลส่วนที่เหลือจาก training data เป็นข้อมูลที่ใช้ทดสอบ (Testing data) ซึ่งเป็นกลุ่มที่แท้จริงของข้อมูลที่ใช้ทดสอบนี้จะถูกนำมาเปรียบเทียบกับกลุ่มที่หามาได้จากโมเดลเพื่อทดสอบความถูกต้อง โดยเราจะปรับปรุงโมเดลจนกว่าจะได้ค่าความถูกต้องในระดับที่น่าพอใจ หลังจากนั้นเมื่อมีข้อมูลใหม่เข้ามา เราจะนำข้อมูลผ่านโมเดล โดยโมเดลจะสามารถทำนายกลุ่มของข้อมูลนี้ได้

ผลการวิจัย

ในงานวิจัยชิ้นนี้ได้ศึกษาและทดลองวิธีการเรียนรู้ข้อมูลจากฐานข้อมูลเชิงสัมพันธ์ร่วมกับคำสั่ง SQL ในการสร้าง Database Trigger โดยได้ใช้ฐานข้อมูลประวัติของผู้ป่วยที่มีความเสี่ยงเป็นโรคเบาหวานและการให้ปริมาณอินซูลินเพื่อรักษาระดับน้ำตาลในเลือดที่มีโครงสร้างดังนี้ (attribute ที่ขีดเส้นใต้แสดงถึง attribute ที่เป็น primary key)

Diabetes (Patient_ID, Name, Sex, Age, Temperature, Blood_Pressure_Upper, Blood_Pressure_Low, Diabetes_family, Weight, Height, BMI, Blood_Sugar, Diabetes, Insulin_level)

ในการทดลองได้ใช้ Training Data จำนวน 20 Records ผนวกกับการใช้ความรู้พื้นฐานทางด้านการแพทย์ในการให้ปริมาณอินซูลินแก่ผู้ป่วยโรคเบาหวาน ทำให้ได้ Trigger Rule กฎที่ 1- 5 และนำข้อมูลชุดดังกล่าวมาทำเหมืองข้อมูลโดยใช้อัลกอริทึม J48 ทำให้ได้ Induced Trigger Rule กฎที่ 6 - 7 ดังนี้

Trigger Rule:

1. If age<11 and blood_sugar>120	Then 10<insulin_level<50
2. If 11<age<20 and blood_sugar>120	Then 50<insulin_level<100
3. If age>20 and 120<blood_sugar<150	Then 10<insulin_level<50
4. If age>20 and 150<blood_sugar<180	Then 50<insulin_level<90
5. If age>20 and blood_sugar>180	Then 90<insulin_level<120

Induced Trigger Rule:

6. If Diabetes_family=yes and BMI>24.9 Then Diabetes=yes

7. If Diabetes_family=no and blood_sugar>128 Then Diabetes=yes

จาก Induced Trigger Rule ที่ได้นำมาสร้างเป็น Database Trigger ได้ดังนี้

กฎที่ 6 If Diabetes_family=yes and BMI>24.9 Then Diabetes=yes

```
CREATE TRIGGER rule_6
ON diabetes
FOR UPDATE, INSERT
AS
IF (SELECT COUNT(*)
FROM diabetes
WHERE (diabetes_family = ‘yes’) and (BMI > 24.9)
and (diabetes <> ‘yes’) ) > 0
BEGIN
   ROLLBACK TRAN
   RAISERROR (‘diagnose error’)
END
```

กฎที่ 7 If Diabetes_family=yes and blood_sugar>128 Then Diabetes=yes

```
CREATE TRIGGER rule_7
ON diabetes
FOR UPDATE, INSERT
```

```
AS
IF (SELECT COUNT(*)
FROM diabetes
WHERE (diabetes_family = 'no') and (blood_sugar > 128)
and (diabetes <> 'yes') ) > 0
BEGIN
   ROLLBACK TRAN
   RAISERROR ('diagnose error')
END
```

เมื่อนำ Induced Trigger Rule ที่ได้ไปใช้ทดสอบเพื่อตรวจสอบความถูกต้องของการเปลี่ยนแปลงฐานข้อมูล ซึ่งทดลองกับ Training Data จากเดิมจำนวน 20 records โดยทำการเพิ่มข้อมูลเข้าไปดังนี้

```
INSERT INTO Diabetes (Patient_ID, Name, Sex, Age, Temperature,
Blood_Pressure_Upper, Blood_Pressure_Low, Diabetes_family, Weight, Height,
BMI, Blood_Sugar, Diabetes, Insulin_level)
Values(P021, Amitta, Female, 33, 37.6, 130, 80, no, 72, 1.62, 27.4, 130, no, 0)
```

โดยข้อมูลที่เพิ่มเข้าไปใหม่มีความขัดแย้งกับ Induced Trigger Rule จึงทำให้ไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้สำเร็จ เนื่องจากขัดแย้งกับกฎที่ 7 คือ If Diabetes_family = no and blood_sugar > 128 Then Diabetes = yes

จากการทดสอบความถูกต้องของ Induced Trigger Rule โดยการนำไปใช้งาน เพื่อทดสอบความถูกต้องของ Test Data พบว่ามีประโยชน์ในการตรวจสอบความถูกต้องของข้อมูลที่เปลี่ยนแปลงเข้าไปใหม่โดยที่จะไม่ยอมให้ทำการเปลี่ยนแปลงข้อมูลที่ขัดแย้งกับ Induced Trigger Rule โดยข้อมูลที่ขัดแย้งนี้อาจเกิดขึ้นในขณะที่ User ป้อนข้อมูลเข้าไปและข้อมูลที่ต้องการเปลี่ยนแปลงมี attribute จำนวนมาก

สรุปและอภิปรายผลการวิจัย

การศึกษาครั้งนี้พบว่าการสร้างดาต้าเบสทริกเกอร์จากกฎที่ได้จากการทำเหมืองข้อมูลนั้น มีประโยชน์ที่ไม่เพียงแต่เป็นการสร้างความถูกต้องสูงสุดกับฐานข้อมูลเท่านั้น แต่ยังเป็นการป้องกันความผิดพลาดในกรณีที่ผู้ใช้หรือ User ทำการปรับปรุงเปลี่ยนแปลงฐานข้อมูล เพื่อไม่ให้ข้อมูลที่บันทึกเข้าไปขัดแย้งกันเอง เมื่อข้อมูลที่บันทึกเข้าไปมีปริมาณมากขึ้นก็จะใช้ข้อมูลชุดใหม่ ๆ นั้นเป็นความรู้พื้นฐานเพื่อที่จะทำให้เกิดรูปแบบหรือกฎใหม่ ๆ และนำกฎที่ได้ขึ้นมาใหม่นั้นไปสร้างดาต้าเบสทริกเกอร์ต่อไป

กิตติกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนจากมหาวิทยาลัยเทคโนโลยีสุรนารีผ่านหน่วยวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้

บรรณานุกรม

De Raedt, L.(2002). A perspective on inductive database. *SIGKDD Explorations*, 4(2), 69-77.

Elmasri, R. & Navathe, S. B. (1994). *Fundamentals of Database System*, The Benjamin/Cummings Publishing Company, California.

Hammer, J., Garcia-Molina, H., Widom, J., Labio, W. & Zhuge, Y. (1995). The Stanford Data Warehousing Project. *Proceedings of IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing,* 18(2), 41-48.

Han, J., Fu, Y., Wang, W., Koperski, K. & Zaiane, O. (1996). DMQL: A Data Mining Query Language for Relational Databases. *Proceedings of SIGMOD DMKD Workshop.* Montreal, Canada.

Imielinski, T. & Virmani, A. (1999). MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery,* 3, 373–408.

Imielinski, T. & Mannila, H. (1996). A Database Perspective on Knowledge Discovery. *Communications of the ACM,* 39, 58–64.

Inmon, W.H. (1992a). Building the data bridge: the ten critical success factors of building a data warehouse. *Database Programming & Design.* 5(11), 68-69.

Inmon, W.H. (1992b). EIS and the data warehouse: a simple approach to building an effective foundation for EIS. *Database Programming & Design,* 5(11), 70-73.

Larose, Daniel T. (2005). *Discovering knowledge in data : an introduction to data mining*. New Jersey: John Wiley & Sons, Inc..

Meo, R., Psaila, G. & Ceri, S. (1998). An Extension to SQL for Mining Association Rules. *Data Mining and Knowledge Discovery*, 2, 195-224.

Quinlan, J. (1992). C4.5:*Programs for Machine Learning,* San Francisco, Morgan Kaufmann.

Witten, I.H. & Frank, E.(2005). Data Mining: *Practical Machine Learning Tools and Techniques,* 2nd edition. San Francisco: Morgan Kaufmann.

ภาคผนวก ข

โปรแกรมแปลงกฎการจำแนกข้อมูลเป็นดาต้าเบสทริกเกอร์

```
// โปรแกรมแปลงดาต้าเบสทริกเกอร์จากกฎ
// โดยศิริกาญจนา  พิลาบุตร, 30 พ.ย. 2551
import java.io.*;

public class TriggerEngine
{
public static void main(String[] args) //throws FileNotFoundException, IOException
{
// ถ้าไม่ได้ใส่ไฟล์ input และ output ให้แสดงวิธีการใช้งาน
if(args.length != 2){
System.out.println("Usage:");
System.out.println("java TriggerEngine <input file> <out file>");
System.exit(0);
}

// สร้างไฟล์ตัวแปรชื่อ input
File input = new File( args[0] );
// ตรวจสอบว่ามีไฟล์นี้อยู่หรือไม่ ถ้าไม่มีให้ออกจากโปรแกรม
if ( !input.exists() ) // if name not exists, output warning
{
System.out.printf( "%s %s", args[0], "does not exist." );
System.exit(1);
}

FileInputStream in;
try{
// สร้าง FileInputStream จาก FileInputStream จาก input ให้อยู่ในตัวแปรชื่อ in เพื่ออ่านไฟล์
in = new FileInputStream(input);
```

```
// หาขนาดของไฟล์
        int size = (int)input.length();
        // จองพื้นที่การเก็บข้อมูล
        byte[] txt = new byte[size];

        try{
           in.read(txt);
        }catch(IOException e){
           System.out.println( e );
        }
        // เปลี่ยนจาก byte เป็น string
        String input_string = new String(txt);

        // แบ่ง input_string ออกจากกันด้วยการขึ้นบรรทัดให้ แล้วเก็บเป็น array ของสตริง
        String[] results = input_string.split( "\\s*\n\\s*" ); // split on new line

        // ใช้เก็บจำนวนของ rule
        int rule_count =0;
        int index;

        // จองพื้นที่ของ rule
        String[] rules = new String[10];

        // กำหนด rule เป็นสตริงว่าง
        String rule="";

        // วนลูปเพื่ออ่านข้อมูลทีละบรรทัด โดยแต่ละบรรทัดเก็บไว้ในตัวแปรสตริง
         for ( String string : results ){
           // ถ้าไม่เริ่มต้นด้วย | ให้ทำต่อ
```

```
if(!string.startsWith("|")){
    // หาเครื่องหมาย : ซึ่งเป็นการจบ rule
    // โดยให้ index = -1 เมื่อไม่เจอ
    index = string.indexOf(":");

    if( index > 0){
       // เมื่อเจอ
       // แบ่งสตริงออกเป็นสองส่วน ก่อนและหลังเครื่องหมาย
       String before_str = string.substring(0, index);
       String after_str = string.substring(index+1);

       String after;
       // ถ้าสตริงส่วนหลังมีคำว่า YES
       if(after_str.indexOf("YES")>0){
          after = "diabetes <> Yes";
       }else{
          after  = "diabetes <> No";
       }

       // เชื่อมสตริงทั้งหมดเข้าด้วยกัน แล้วเก็บไว้ในตัวแปร rule
       rule = "(" + before_str.trim() + ") and (" + after.trim() + ")";

       // เก็บ rule ให้ใน rule ที่ index เท่ากับ rule_count
       rules[rule_count] = rule;
       // เพิ่มค่า rule_count อีก 1
       rule_count++;
    }else{
       // rule
       rule = "(" + string.trim() + ") ";
    }
```

```
                // เมื่อเจอ เครื่องหมาย |
                // หาตำแหน่งของเครื่องหมาย | ตัวสุดท้าย
                index = string.lastIndexOf("|");
                // เลือกส่วนท้ายของสตริงมาเก็บไว้ใน s
                String s = string.substring(index+1);

               // หาเครื่องหมาย : ซึ่งเป็นการจบ rule
                // โดยให้ index = -1 เมื่อไม่เจอ
                index = s.indexOf(":");
                if( index > 0){
                   // เมื่อเจอ
                   // แบ่งสตริงออกเป็นสองส่วน ก่อนและหลังเครื่องหมาย
                   String before_s = s.substring(0, index);
                   String after_s = s.substring(index+1);

                   String after;
                   // ถ้าสตริงส่วนหลังมีคำว่า YES
                   if(after_s.indexOf("YES")>0){
                      after = "diabetes <> Yes";
                   }else{
                      after  = "diabetes <> No";
                   }
                   // เชื่อมสตริงทั้งหมดเข้าด้วยกันรวมทั้ง rule ด้วย แล้วไว้ในตัวแปร rules
                   rules[rule_count] = rule + "\r\n\t\t and (" + before_s.trim() + ")\r\n\t\t and (" +
after.trim() + ")";
                   // เพิ่มค่า rule_count อีก 1
                   rule_count++;
                }else{
                   // ต่อข้อความเข้าไปใน rule อีก
                   rule = rule + "\r\n\t\t and (" + s.trim() + ")";
```

```
        }
      }
}
// เตรียมแสดงผลออก output
String r, sql;
// เปิดไฟล์เพื่อเขียนผลลัพธ์ลงไป
File output = new File( args[1] );
FileWriter out;

try{
   out = new FileWriter(output);
   // วนลูปทีละ rule
   for(int j=0;j<rule_count;j++){
      r = rules[j];
      // เปลี่ยนรูปแบบของข้อความ
      r = r.replace( "Yes", "'yes'" );
      r = r.replace( "No", "'no'" );

      // สร้างข้อความ trigger เก็บไว้ในตัวแปร sql
      sql = "CREATE TRIGGER rule_" + (j+1) + "\r\n";
      sql += "ON diabetes\r\n";
      sql += "FOR UPDATE, INSERT\r\n";
      sql += "AS\r\n";
      sql += "    IF (SELECT COUNT(*)\r\n";
      sql += "        FROM diabetes\r\n";
      // แทรกข้อความที่ได้ไว้ใน where
      sql += "        WHERE " + r +")> 0\r\n";
      sql += "BEGIN\r\n";
      sql += "    ROLLBACK TRAN\r\n";
      sql += "    RAISERROR ('diagnose error')\r\n";
```

```
                // เขียนลงไฟล์
                out.write(sql);
                System.out.println(sql);
            }
            // ปิดไฟล์
            out.close();
        }catch(IOException e){
            System.out.println( e );
            System.exit(0);
        }
    }catch(FileNotFoundException e){
        System.out.println( e );
        System.exit(0);
    }
  }
}
```

ประวัติผู้เขียน

นางสาวศิริกาญจนา พิลาบุตร เกิดเมื่อวันที่ 31 ธันวาคม พ.ศ. 2521 ที่ อำเภอเมือง จังหวัดนครราชสีมา เริ่มการศึกษาระดับประถมศึกษาปีที่ 1-6 ที่โรงเรียนเทศบาล 3 (ยมราชสามัคคี) จังหวัดนครราชสีมา ระดับมัธยมศึกษาปีที่ 1-3 ที่โรงเรียนมารีย์วิทยา จังหวัดนครราชสีมา ระดับมัธยมศึกษาปีที่ 4-5 ที่โรงเรียนบุญวัฒนา จังหวัดนครราชสีมา และสอบเทียบได้รับวุฒิการศึกษาระดับมัธยมศึกษาปีที่ 6 ในปีการศึกษา 2538 จากนั้นเข้าศึกษาในระดับประกาศนียบัตรวิชาชีพชั้นสูงหลักสูตร 4 ปี ที่สาขาวิชาเทคนิคคอมพิวเตอร์ ภาควิชาอิเล็กทรอนิกส์ สถาบันเทคโนโลยีราชมงคล วิทยาเขตภาคตะวันออกเฉียงเหนือ จังหวัดนครราชสีมา และสำเร็จการศึกษาเมื่อปี พ.ศ. 2542 จากนั้นได้เข้าศึกษาต่อในระดับปริญญาตรี สาขาวิศวกรรมไฟฟ้า สาขาย่อยวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีมหานคร และสำเร็จการศึกษาเมื่อปี พ.ศ. 2544 ระหว่างการศึกษาได้เป็นผู้ช่วยอาจารย์สอนปฏิบัติในรายวิชา C Programming Language ภายหลังจบปริญญาตรีได้เข้าปฏิบัติงานที่การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย อำเภอบางกรวย จังหวัดนนทบุรี จากนั้นเมื่อปี พ.ศ. 2548 เข้าปฏิบัติงานที่วิทยาลัยนครราชสีมา และได้เข้าศึกษาต่อระดับปริญญาโท สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2550

ในระหว่างการศึกษาได้รับความอนุเคราะห์อย่างดียิ่งจากคณาจารย์ในสาขาวิชา โดยได้รับความไว้วางใจให้เป็นผู้สอนปฏิบัติการรายวิชา Database System และ Computer Programming Laboratory โดยมีผลงานตีพิมพ์จำนวน 1 เรื่อง มีรายละเอียดปรากฏในภาคผนวก ก.