# ระบบฐานข้อมูลอนุมานที่ใช้แบบจำลองระดับแนวคิด

# A DEDUCTIVE DATABASE SYSTEMS BASE ON A CONCEPTUAL SCHEMA




นายธนา หงษ์สุวรรณ

MR. THANA HONGSUWAN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2538

ISBN 974-621-412-8

# A DEDUCTIVE DATABASE SYSTEMS BASE ON A CONCEPTUAL SCHEMA

THANA HONGSUWAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

GRADUATE SCHOOL

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1995

ISBN 974-621-412-8

| | |
|---|---|
| **หัวข้อวิทยานิพนธ์** | **ระบบฐานข้อมูลอนุมานที่ใช้แบบจำลองระดับแนวคิด** |
| **นักศึกษา** | **นายธนา หงษ์สุวรรณ** |
| **อาจารย์ผู้ควบคุมวิทยานิพนธ์** | **ผศ.ดร. ศุภมิตร จิตตะยโศธร** |
| **ระดับการศึกษา** | **วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า** |
| **ภาควิชา** | **วิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง** |
| **พ.ศ.** | **2538** |

## บทคัดย่อ


**ในปัจจุบันเทคโนโลยีทางด้านฐานข้อมูลได้พัฒนาไปมาก ทั้งในแง่ของประสิทธิภาพการทำงานที่เร็วขึ้น ทั้งในแง่ของความสามารถที่มากขึ้น ทำให้ได้รับการนำไปใช้อย่างกว้างขวาง โดยโมเดลข้อมูลที่ได้รับความนิยมมากที่สุด คือ โมเดลรีเลชันแนล ซึ่งมีข้อดีที่ใช้งานง่าย สามารถนำไปประยุกต์เข้ากับการเก็บข้อมูลในเชิงธุรกิจได้ดี และมีทฤษฎีทางคณิตศาสตร์รองรับ**

**อย่างไรก็ตาม โมเดลข้อมูลแบบรีเลชันแนลก็ยังมีข้อเสียในแง่ของการแสดงความสัมพันธ์ของข้อมูล เพราะแสดงได้ในแบบของตารางเท่านั้น จึงไม่สามารถแสดงความสัมพันธ์ของข้อมูลในตารางและระหว่างตารางได้อย่างชัดเจน อีกทั้งยังขาดการควบคุมความถูกต้องของข้อมูลที่ดีพอ ด้วยเหตุนี้จึงมีนักวิจัยหลายท่าน ที่พยายามหาโมเดลข้อมูลใหม่ ๆ ที่จะมาทดแทนข้อเสียของโมเดลรีเลชันแนล ซึ่งก็มีแนวทางการวิจัยไปหลายแนวทางด้วยกัน**

**วิทยานิพนธ์ฉบับนี้เป็นอีกแนวทางหนึ่งในการทดลองสร้างฐานข้อมูลที่ใช้โมเดลอื่น ๆ ในการแทนข้อมูล โดยได้นำแบบจำลองข้อมูลระดับแนวคิดไนแอม (NIAM) มาเป็นโมเดลข้อมูล ซึ่งไนแอมเป็นแบบจำลองข้อมูลที่มีรากฐานอยู่บนแต่ละความสัมพันธ์ของข้อมูล จึงสามารถแทนความหมายในแง่ของความสัมพันธ์ได้ดี และมีกฎเกณฑ์ข้อบังคับความถูกต้องของข้อมูลที่ช่วยให้ข้อมูลมีความถูกต้องสูง**

**และเนื่องจากแบบจำลองข้อมูลไนแอม มีความสัมพันธ์ที่ใกล้ชิดกับทฤษฎีทางตรรกศาสตร์ ดังนั้นในวิทยานิพนธ์ฉบับนี้จึงได้ประยุกต์เอาความสามารถทางด้านการอนุมานกฎที่มีอยู่ในทฤษฎีทางตรรกศาสตร์ มาใช้กับฐานข้อมูลที่จะสร้างขึ้น ซึ่งทำให้ฐานข้อมูลนี้มีความสามารถในการอนุมานกฎได้ด้วย ทั้งกฎชนิดเรียกตัวเองและชนิดไม่เรียกตัวเอง โดยเรียกฐานข้อมูลนี้ว่า "ฐานข้อมูลอนุมาน"**

| | |
|---|---|
| **Thesis Title** | A Deductive Database Systems Base On A Conceptual Schema |
| **Student** | Thana Hongsuwan |
| **Thesis Advisor** | Assoc. Prof. Dr. Suphamit Chittayasothon |
| **Level of Study** | Master of Engineering in Electrical Engineering |
| **Department** | Computer Engineering Faculty of Engineering |
| | King Mongkut's Institute of Technology Ladkrabang |
| **Year** | 1995 |

## ABSTRACT

Recently, Database Technology was developed rapidly both performance and ability. Database system is used in many application wildly, and most popular model is relational model. Relational model is easy to use, good applicate with business system and base on mathematics theory.

However, relational model is not good in relation between data, because it represent only in form of table. It can not show relation between table and relation of data on same table precisely. Otherwise, it has not a good integrity constraint . In case that, some researcher try new data modelling to substitute relational model.

This thesis is concerned with the design of database that have another data model, focussing on NIAM conceptual schema. NIAM have only one data structure to represent relation, which are meaningful and easy to understand. Addition, integrity constraint of NIAM help high integrity of data.

NIAM has closely relation with logic theory, thus this thesis has apply inference rule ability to NIAM database system. The NIAM database system with inference capability can inference rule, both recursive rule and non-recursive rule. I has called this database "A Deductive Database Systems Base On A Conceptual Schema"

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ ก็คือ อาจารย์ ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

ขอขอบคุณ คุณภุชงค์ หงษ์สุวรรณ หัวหน้าแผนกสารสนเทศ คุณพิทักษ์ ธรรมวาริน คุณบัณฑิต พัสยา คุณสุธีรา พันธ์ธีรานุรักษ์ เจ้าหน้าที่แผนกสารสนเทศ ที่ให้ความช่วยเหลือในการหาโปรแกรมใช้งานต่าง ๆ และช่วยเหลือในการจัดพิมพ์วิทยานิพนธ์ฉบับนี้ และสุดท้ายก็ขอขอบคุณ คุณนภพล ใจดี คุณธวัชชัย สุทธิทศธรรม และคุณสมชิด ศิริผลหลาย ซึ่งเป็นเพื่อนสนิท ที่คอยถามไถ่และให้กำลังใจอยู่เสมอ

และท้ายสุดนี้ข้าพเจ้าขอขอบคุณ มูลนิธิ C&C และผู้สนับสนุนมูลนิธิทุกท่าน ที่ได้ก่อตั้งมูลนิธิมาเพื่อสนับสนุนการศึกษาโดยไม่คิดผลตอบแทนใด ๆ ซึ่งข้าพเจ้าขอขอบคุณมูลนิธิ C&C ที่ได้มอบทุนสนับสนุนการศึกษาแก่ข้าพเจ้า ซึ่งเป็นส่วนหนึ่งที่ทำให้ข้าพเจ้าประสบผลสำเร็จในการศึกษา ณ วันนี้ได้

ธนา หงษ์สุวรรณ

# สารบัญ

หน้า

# สารบัญตาราง

**ตารางที่** **หน้า**

# สารบัญภาพ

**ตารางที่** **หน้า**

# บทที่ 1
# บทนำ

## 1.1 ความสำคัญและที่มา

ในปัจจุบันพัฒนาการทางด้านวิชาฐานข้อมูลนับว่าได้ก้าวหน้าไปมาก มีนักวิทยาศาสตร์ในสาขานี้เป็นจำนวนมากได้ทำการวิจัยและพัฒนาวิธีเก็บข้อมูลจำนวนมากลงบนหน่วยความจำสำรอง (Secondary Memory) โดยมีแบบจำลองข้อมูลในรูปแบบต่างๆ และมีการจัดเก็บที่เป็นระบบ สามารถลดความซ้ำซ้อน (Redundancy) รักษาความถูกต้อง (Integrity) ลดการสูญหายของข่าวสาร (Information Lost) และลดข้อผิดพลาดจากการแก้ไขข้อมูล (Update Anomolies)

นอกจากพัฒนาการทางด้านการออกแบบฐานข้อมูลแล้ว วิชาการทางด้านระบบจัดการฐานข้อมูล (Database Management System : DBMS) ก็ได้รับการพัฒนาไปอย่างมากเช่นเดียวกัน ระบบจัดการฐานข้อมูลได้รับการพัฒนาให้มีคุณภาพสูง และมีสถาปัตยกรรมที่ใกล้เคียงสถาปัตยกรรมมาตรฐานของ ANSI และ ISO มีการแบ่งแยกเป็นระดับแนวคิด (Conceptual Level) เป็นระดับภายนอก (External Level) และระดับภายใน (Internal Level) ที่เด่นชัด มีความเป็นอิสระของข้อมูล (Data Independance) ทั้งทางตรรกและทางกายภาพ

นอกจากนี้ระบบจัดการฐานข้อมูลในปัจจุบันยังมีความปลอดภัย มีระบบเรียกข้อมูลคืนป้องกันการสูญหาย (Recovery) มีระบบใช้ข้อมูลร่วมกันระหว่างผู้ใช้หลายๆ คน (Multiuser) และมีภาษาจัดการข้อมูลที่มีประสิทธิภาพ อย่างน้อยเทียบเท่ากับพีชคณิตรีเลชัน (Relational Algrebra) และใช้เทคโนโลยีทางด้าน Query Optimization เพื่อให้สามารถจัดการกับข้อมูลขนาดใหญ่ได้ด้วยประสิทธิภาพสูง

เทคโนโลยีทางด้านปัญญาประดิษฐ์ก็ได้พัฒนาไปมากเช่นกัน โดยเฉพาะเทคโนโลยีทางด้านการสร้างฐานความรู้ นักวิจัยในสาขาปัญญาประดิษฐ์จำนวนมากได้พยายามใช้แบบแทนความรู้ (Knowladge Representation) ชนิดต่างๆ มาใช้แทนความรู้ (Domain Knowladge) สำหรับใช้แก้ปัญหาในงานต่างๆ ซึ่งแบบแทนความรู้ที่เป็นที่นิยมมากได้แก่ Semantic Network, Frames ซึ่งเป็นแบบแทนความรู้ชนิดโครงสร้าง (Structured Knowledge Representation) และ Production Rules, Predicate Logic ซึ่งเป็นแบบแทนความรู้ชนิดไม่มีโครงสร้าง สำหรับแสดงความสัมพันธ์ของหน่วยพื้นฐาน (Entity) ในฐานความรู้ .

แบบแทนความรู้ชนิดโครงสร้างมีข้อดีที่สามารถแสดงความสัมพันธ์ของหน่วยพื้นฐานได้อย่างเด่นชัดว่า หน่วยพื้นฐานใดสามารถมีความสัมพันธ์กับหน่วยพื้นฐานอื่นๆ ในรูปแบบใดบ้าง ส่วนแบบแทนความรู้ชนิดไม่มีโครงสร้างนั้น เป็นแบบแทนความรู้ที่ง่ายต่อการประมวลผล และบางชนิดมีทฤษฎีการพิสูจน์ทางคณิตศาสตร์ (Logical Deduction) รองรับทำให้การสรุปความรู้ใหม่โดยอาศัยกฎและความรู้ที่มีอยู่แล้วเป็นไปโดยสะดวก

อย่างไรก็ตามเป็นที่น่าสังเกตุว่า ในวงการปัญญาประดิษฐ์นั้นคำว่า ฐานความรู้ (Knowladge Base) ถูกใช้เรียกแหล่งเก็บความรู้ โดยมิได้คำนึงถึงลักษณะการเก็บเลย ฐานความรู้ส่วนใหญ่จะอยู่ในรูปของโปรแกรมภาษาลิสป์ (Lisp) ซึ่งเหมาะสำหรับใช้สร้าง Semantic Network หรือโปรแกรมภาษาโปร

ลอก (Prolog) ซึ่งแสดงความรู้ในรูปของ พรีดิเขตลอจิก (Predicate Logic) ซึ่งโปรแกรมเหล่านี้จะต้องอยู่ในหน่วยความจำหลักของระบบคอมพิวเตอร์ขณะทำการประมวลผล ทำให้มีข้อจำกัดหลายประการโดยเฉพาะเรื่องขนาดของฐานข้อมูล การจัดระเบียบและการทำดัชนี (Index) เป็นต้น

ระบบทางปัญญาประดิษฐ์ในยุคแรกๆ เก็บข้อมูลในฐานความรู้โดยแทบมิได้อ้างอิงกับ ทฤษฎีทางฐานข้อมูลเลย ต่อมาภายหลังเมื่อระบบปัญญาประดิษฐ์ได้รับการพัฒนายิ่งขึ้น จนถึงขั้นเชิงพาณิชย์ ความจำเป็นในการที่จะนำเอาเทคโนโลยีทางด้านฐานข้อมูล เข้ามาจัดการกับฐานความรู้ขนาดใหญ่ในระบบปัญญาประดิษฐ์ก็เป็นที่ประจักษ์ชัดขึ้น มีนักวิทยาศาสตร์หลายท่านได้พยายามผสมผสานเทคโนโลยีทั้งสองเข้าด้วยกัน เช่น Gallaire H., Minker J. , Kowalski, R.A. ,Raiter, R. เป็นต้น และการผสมผสานดังกล่าวได้ทำให้เกิดสาขาใหม่ขึ้น ซึ่งเป็นสาขาย่อยของทั้งฐานข้อมูลและปัญญาประดิษฐ์ และเรียกสาขานั้นว่า ระบบฐานข้อมูลอนุมาน (Deductive Database)

และจากที่ได้กล่าวมาทั้งหมดนั้น ผู้อ่านคงจะเห็นถึงความไม่เหมาะสมของการแทนความรู้ในระบบปัญญาประดิษฐ์ที่มีฐานความรู้ขนาดใหญ่ และความสามารถของระบบฐานข้อมูลสามารถจะแก้ปัญหาที่จุดนี้ได้ และหากมองในแง่กลับกัน การผสมผสานแนวทางทั้งสองเข้าด้วยกันอาจจะมองว่าเป็นการเพิ่มให้ระบบฐานข้อมูลมีความสามารถในการอนุมาน โดยในโครงงานวิจัยนี้เราจะนำแบบจำลองข้อมูลไนแอม (The NIAM Conceptual Schema) มาใช้ในเป็นแบบในการจำลองข้อมูลซึ่งที่จะได้กล่าวต่อไป

## 1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 ศึกษาความเป็นไปได้ในการนำเอาแบบจำลองข้อมูลไนแอม มาเป็นแบบจำลองข้อมูลระดับแนวคิด และศึกษาลักษณะการแทนข้อมูลลงในฐานข้อมูล โดยอาจจะเป็นทางเลือกหนึ่งที่มาแทนแบบจำลองข้อมูลแบบรีเลชันแนลได้

1.2.1 ศึกษาการสร้างระบบฐานข้อมูลว่าจะต้องมีการออกแบบในส่วนใดบ้าง มีสถาปัตยกรรมเป็นอย่างไร มีการออกแบบเมต้าสกีมาอย่างไร และจะสร้างส่วนตรวจสอบความถูกต้องของข้อมูลได้อย่างไร ซึ่งอาจจะนำความรู้ที่ได้รับนี้ไปประยุกต์ใช้ในงานต่างๆ ได้

1.2.2 เพื่อเป็นการนำเสนอแนวคิด และศึกษาความเป็นไปได้ในการสร้างระบบฐานข้อมูลที่มีความสามารถในการอนุมานกฎได้ และศึกษาวิธีการประมวลผลกฎแบบรีเคอร์ซีฟ ตลอดจนศึกษาวิธีจะแทนกฎลงในฐานข้อมูล ซึ่งด้วยความสามารถในการอนุมานนี้ อาจจะนำฐานข้อมูลนี้ไปใช้เป็นฐานความรู้สำหรับงานทางด้านปัญญาประดิษฐ์ได้

## 1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะสร้างระบบฐานข้อมูลอนุมานขึ้นมา โดยจะสร้างอยู่บนระบบฐานข้อมูลแบบรีเลชันแนล โดยใช้แบบจำลองข้อมูลไนแอมเป็นแบบจำลองข้อมูลในระดับแนวคิด (Conceptual Schema) โดยที่จะออกแบบส่วนติดต่อกับผู้ใช้ให้คล้ายกับภาษาโปรลอก และมีความสามารถในการอนุมานกฎได้ด้วย ในงานวิจัยนี้จะออกแบบสถาปัตยกรรมของระบบทั้งหมด

แต่งานวิจัยนี้จะไม่สนใจในแง่ที่มาของแบบจำลองข้อมูลระดับแนวคิด โดยจะสมมติว่าแบบจำลองข้อมูลถูกจัดเก็บในฐานข้อมูลเรียบร้อยแล้ว งานวิจัยนี้ก็จะนำข้อมูลนั้นมาใช้ได้เลย และนั่นก็หมายความว่าการออกแบบสถาปัตยกรรมของระบบ จะออกแบบทั้งระบบ แต่ในการพัฒนาระบบนั้น เราจะพัฒนาเฉพาะส่วนที่ติดต่อกับผู้ใช้เท่านั้น จะไม่สนใจส่วนที่ติดต่อกับผู้ออกแบบแบบจำลองระดับแนวคิด

นอกจากนั้นในโครงงานนี้ยังถือว่าเป็นโครงการที่ทดลองสร้าง เพื่อศึกษาความเป็นไปได้ในการใช้งาน ดังนั้นจึงมีข้อจำกัดของข้อมูลบางอย่าง เช่น อาจจะมีการจำกัดความยาวของชื่อ อาจจะมีการจำกัดจำนวนข้อมูลที่ไม่สามารถรองรับข้อมูลปริมาณมาก ๆ ได้ แต่ถึงอย่างไรก็ยังเพียงพอต่อการทดสอบอย่างแน่นอน

### 1.4 วิธีการดำเนินงาน

งานวิจัยในโครงงานนี้จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องกับงานวิจัย ซึ่งก็มีเรื่องหลัก ๆ อยู่ 3 เรื่องด้วยกัน คือ แบบจำลองข้อมูลไนแอม ทฤษฎีพื้นฐานทางตรรกศาสตร์ และวิธีการในการจัดการกับกฎในฐานข้อมูลอนุมาน ซึ่งมีรายละเอียดดังในบทที่ 2, 3 และ 4 จากนั้นก็จะนำเอาความรู้ที่ได้ศึกษาทั้งหมดมาออกแบบสถาปัตยกรรมของระบบ ซึ่งมีรายละเอียดในบทที่ 5 และออกแบบเมต้าสกีมา ซึ่งเป็นรูปแบบข้อมูลที่จะใช้เก็บแบบจำลองข้อมูลระดับแนวคิด ซึ่งมีรายละเอียดในบทที่ 6

จากนั้นก็จะเริ่มเข้าสู่ขั้นตอนของการพัฒนาโปรแกรม โดยบทที่ 7 จะกล่าวถึงองค์ประกอบโดยรวมของระบบที่พัฒนาขึ้นมาทั้งหมด และยังอธิบายไปถึงรูปแบบการติดต่อกับผู้ใช้ และการประมวลผลเบื้องต้นก่อนที่จะส่งไปให้กับส่วนประกอบต่าง ๆ จากนั้นจะอธิบายแต่ละส่วน โดยเริ่มจากส่วนนิยามข้อมูลซึ่งจะเป็นการสร้างวิวให้กับชนิดความจริง และส่วนนำเข้าข้อมูล ซึ่งก็คือการนำข้อมูลเข้าและออกจากฐานข้อมูลในรูปของชนิดความจริง โดยมีรายละเอียดในบทที่ 8 สำหรับในบทที่ 9 ก็จะเป็นเรื่องราวของการตรวจสอบความถูกต้องของข้อมูลที่ควบคุมด้วยคอนสแตนต์ต่าง ๆ และบทที่ 10 ก็จะเป็นเรื่องของการจัดการกฎในฐานข้อมูลอนุมาน ซึ่งเป็นส่วนที่สำคัญส่วนหนึ่งของงานวิจัยนี้ โดยจะบอกถึงวิธีการที่ได้คัดเลือกวิธีในการจัดการกับกฎ และรายละเอียดของวิธีการนั้น

สำหรับบทที่ 11 ก็จะเป็นการทดสอบระบบรวมทั้งหมด และบทที่ 12 ซึ่งเป็นบทสุดท้ายก็จะเป็นการสรุปการทำงาน ผลที่ได้รับจากงานวิจัยชิ้นนี้ และแนวทางในการพัฒนางานวิจัยนี้เพิ่มเติม และแนวทางในการนำไปประยุกต์ใช้

### หนังสืออ้างอิง

[1] Codd E.F. (1970) : "*A relational model of data for large shared data banks*", communication ACM 13 ,pp.377-387

[2] Nijssen G.M. (1983) : "*From databases towords knowladge bases*", DBMS - A Technical Comparision, State of Art Report, King P.J. Ed., Pergamon Infotech.

# บทที่ 2
# การโปรแกรมเชิงตรรกเบื้องต้น

เนื่องจากในโครงงานนี้ ได้มีบางส่วนของโครงงานที่นำทฤษฎีทางด้านตรรกศาสตร์ (Symbolic Logic) ไปใช้อีกทั้งจากความจริงที่ว่าทั้งทฤษฎีทางด้านรีเลชันแนล และทฤษฎีทางด้านตรรกศาสตร์นั้นมีความใกล้ชิดกันอยู่มาก และสามารถที่จะพิสูจน์ได้ว่ามีความเหมือนกันภายใต้เงื่อนไขบางประการซึ่ง ดังนั้นในบทนี้จะกล่าวถึงทฤษฎีเบื้องต้นทางด้านตรรกศาสตร์

## 2.1 บทนำ

การโปรแกรมเชิงตรรก (Logic Programming) เป็นสาขาหนึ่งในสาขาวิทยาการคอมพิวเตอร์ ซึ่งตามความเป็นจริงแล้วก็เป็นทฤษฎีที่มีพัฒนาการมาเป็นเวลานาน เพราะการโปรแกรมเชิงตรรกนั้นก็คือรูปแบบหนึ่งของ First Order Predicate Logic (FOPL) ที่เรียกกันว่า Horn Clause นั่นเอง และ FOPL ก็ไม่ไช่อะไรอื่น นอกจากผลจากวิวัฒนาการทางด้าน Symbolic Logic และหากเราต้องการจะย้อนกลับไปดูการเริ่มต้นของการศึกษาในเรื่องของ Symbolic Logic นั้นก็ต้องย้อนกลับไปในสมัยอริสโตเติลนั่นทีเดียว เพราะ FOPL เป็นสาขาหนึ่งของ Symbolic Logic ที่แยกตัวออกมาในช่วง ศตวรรษที่ 20

Symbolic Logic คืออะไร จริงๆ แล้ว หากเราจะศึกษา Symbolic Logic เราจะสามารถทำได้ในหลายๆ มุมมอง เช่น อาจจะมองในแง่ของปรัชญา ในแง่ของคณิตศาสตร์ แต่ในที่นี้เราจะมองในแง่ของการนำไปประยุกต์ใช้งานในการแทนปัญหา เราลองมาดูกันว่า Symbolic Logic สามารถแทนปัญหาได้อย่างไร ลองมาดูตัวอย่างจากประโยคต่อไปนี้

F1 : If it is hot and humid, then it will rain.

F2 : If it is humid, then it is hot.

F3 : It is humid now.

และปัญหาก็คือ Will it rain?

และสมมติให้ประโยค F1, F2 และ F3 เป็นจริง และจากความจริงดังกล่าว เราสามารถใช้สัญญลักษณ์ทางตรรกในการแทนความจริงเหล่านั้นได้ โดยกำหนดให้ P,Q,R แทนความจริงว่า "It is hot", "It is humid" และ "It will rain" ตามลำดับ และใช้เครื่องหมาย ^ ในการแทนความหมายของ and และ -> ในการแทนความหมายของ If...Then (Imply) ดังนั้นจากตัวอย่างข้างต้น เราสามารถนำประโยคปกติมาเขียนใหม่ในรูปของประโยคสัญญลักษณ์ได้ดังนี้

F1 : P ^ Q -> R

F2 : Q -> P

F3 : Q

และเมื่อเราทำการแปลงประโยคภาษาอังกฤษให้อยู่ในรูปแบบของประโยคทางตรรก (Logical Formulas) เราก็จะสามารถที่จะพิสูจน์ได้ว่าหาก F1, F2 และ F3 เป็นจริง แล้ว F4 : R จะเป็นจริงด้วย และในลักษณะเช่นนี้เรากล่าวได้ว่า F4 logically follows from F1, F2 และ F3 นั่นคือจะได้คำตอบว่า ฝนตก ทีนี้เราลองมาดูกันอีกตัวอย่างหนึ่ง สมมติว่าเรามีความจริงต่อไปนี้

F1 : Confucius is a man.

F2 : Every man is mortal.

จากรูปแบบของความจริงข้างต้น จะเห็นได้ว่าเราไม่สามารถแปลงเป็นรูปแบบทางตรรกในลักษณะเช่นเดิมได้ (ไม่มีรูปแบบของ every) ดังนั้นเราจะใช้วิธีการแทนความจริงเหล่านี้ในอีกรูปแบบหนึ่ง และจะเรียกว่าพรีดิเขต (Predicate) โดยกำหนดให้ P(x) และ G(x) แทนความหมายว่า x is a man และ x is mortal และเราจะใช้เครื่องหมาย (∀x) หมายถึงสำหรับทุก x (For all x) ดังนั้นสำหรับความจริงข้างต้น เราจะเขียนในรูปแบบของพรีดิเขตได้เป็น

F1 : P(Confucius)

F2 : (∀x) (P(x) -> Q(x))

และจากความจริง F1 และ F2 นั้นเราสามารถที่จะอนุมานในเชิงตรรก (logically deduce) ได้ว่า F3 : Q(Confucius) หรือหมายความว่า Confucius is mortal.

จากตัวอย่างที่ได้กล่าวมาข้างต้น จะเห็นได้ว่าเราสามารถที่จะสรุป หรืออนุมานความจริงใดๆ ได้จากความจริงอื่นๆ และเราอาจจะมองได้อีกว่า Symbolic Logic คือภาษาหนึ่งที่มีลักษณะพิเศษคือสามารถใช้สัญญลักษณ์ในการแทนและสรุปเหตผลได้ ต่อไปเราจะศึกษา Symbolic Logic ที่มีรูปแบบที่ง่ายที่สุดคือ Propositional Logic

**2.2 Propositional Logic**

ใน Proposition Logic นั้น ขั้นแรกเราจะต้องกำหนดประโยคขึ้นมา และจากนั้นเราจะสนใจว่าประโยคที่กำหนดนั้นมีค่าความจริงเป็นอะไร ซึ่งใน Propositional Logic เรากำหนดว่าค่าความจริงจะต้องเป็น "จริง" หรือ "เท็จ" เท่านั้นจะเป็นอย่างอื่นไม่ได้ และจะเรียกแต่ละประโยคที่กำหนดนี้ว่า Proposition เช่น

P : Snow is white.

Q : Sugar is a hydrocarbon.

R : Smith has a Ph. D. Degree.

ซึ่งเราจะแทนค่าความจริง (Truth Value) ของแต่ละ Proposition ด้วย T (True) และ F (False) และจากประโยคข้างต้น เราจะแทน Proposition แต่ละ Proposition ด้วยสัญญลักษณ์อักษรตัวใหญ่ หรือ

สตริงที่ขึ้นต้นด้วยอักษรตัวใหญ่ ซึ่งในที่นี้ก็คือ P, Q และ R และเราจะเรียกสัญญลักษณ์ดังกล่าวว่า **อะตอม** (atoms)

และจาก Proposition ต่างๆ นั้นเราสามารถที่จะสร้าง Proposition ขึ้นมาใหม่โดยที่ Proposition ที่สร้างขึ้นมาใหม่นี้ จะมีรูปแบบที่ซับซ้อนกว่าเก่าซึ่งจะเรียกว่า Compound Proposition โดยใช้ **ตัวเชื่อมทางตรรก** (logical connective) เช่น จากประโยคที่ว่า

Snowis white **and** sky is clear. และ

**If** john is not at home **then** mary is at home.

ตัวเชื่อมทางตรรกของประโยคข้างต้นก็คือ "and" และ "if ... then" ซึ่งตัวเชื่อมทางตรรก**ทั้งหมด**ที่ใช้งานนั้นก็มีอยู่ 5 ชนิดดังต่อไปนี้ ~(not) ^ (and) v (or) -> (if .. then) และ <-> (if and only if) และใน Propositional Logic เราจะเรียกทั้ง Proposition และ Compound Proposition ว่า WFF (Well-Formed Formula) และในกรณีที่ใน WFF มีตัวเชื่อมมากกว่า 1 ตัวขึ้นไปก็จะต้องมีการจัดลำดับของตัวเชื่อม โดยกำหนดให้ลำดับความสำคัญของตัวเชื่อม (Connective) เรียงจากน้อยไปหามากดังต่อไปนี้

<-> -> ^ v ~

และนั่นก็หมายความว่า WFF ที่ว่า P -> Q ^ ~R v S จะมีความหมายในทางตรรกเทียบเท่ากับ (P -> (Q^((~R)vS))) และเมื่อกำหนดให้ G และ H เป็น WFF ค่าของ ~G, (G ^ H), (G v H), (G -> H), (H <-> G) จะมีได้ดังตารางที่ 2-1

| G | H | ~G | G ^ H | G v H | G -> H | G <-> H |
|---|---|---|---|---|---|---|
| F | F | T | F | F | T | T |
| F | T | T | F | T | F | F |
| T | F | F | F | T | T | F |
| T | T | F | T | T | T | T |

*ตารางที่ 2-1 แสดงค่าความจริงของตัวเชื่อมแบบต่างๆ เมื่อกำหนดให้ G และ H เป็น WFF ที่มีค่าใดๆ*

และจากตารางนี้เองทำให้เราสามารถที่จะหาค่าความจริงของ WFF ใดๆ ได้จากค่าความจริงของทุกๆ อะตอมที่อยู่ใน WFF ได้

### 2.2.1 การแทนค่า Formula ใน Propositional Logic

สมมติว่าเรามีอะตอมอยู่ 2 ตัวคือ P และ Q ซึ่งมีค่าความจริงเป็น T และ F ตามลำดับ และจากตารางที่ 2-1 หากเราแทน P ด้วย G และ Q ด้วย H เราจะได้ว่าค่าความจริงของ (~P), (P ^ Q), (P v Q), (P -> Q), (P <-> Q) คือ F,F,T,F และ F ตามลำดับ ดังนั้นเราสามารถสรุปได้ว่าค่าความจริงของ Formula ใดๆ นั้นสามารถหาได้จากการสรุปจากค่าความจริงของทุกๆ อะตอมที่อยู่ใน Formula นั้น

**ตัวอย่าง** กำหนด Formula

G = (P ^ Q) -> (R <-> (~S))

อะตอมของ Formula ข้างต้นก็คือ P, Q, R และ S ซึ่งในที่นี้กำหนดให้มีค่าความจริงเป็น T,F,T และ T ตามลำดับ และจากค่าความจริงที่เราได้กำหนดให้อะตอม เราสามารถสรุปได้ว่า (P ^ Q) มีค่าความจริงเป็น F และเนื่องจาก (~S) มีค่าเป็น F ดังนั้น (R <-> (~S)) มีค่าเป็น T และจากที่กล่าวมาทั้งหมดจะสามารถสรุปได้ว่า Formula ข้างต้นมีค่าความจริงเป็น T เมื่อกำหนดให้ P,Q,R และ S มีค่าเป็น T,F,T และ T ตามลำดับ

การกำหนดค่าความจริง T,F,T,T ให้กับ P,Q,R,S นี้เราเรียกว่าการแทนค่า (Interpretation) ของ Formula G ดังนั้นสำหรับ P,Q,R และ S ก็จะสามารถมีการแทนค่าได้ทั้งหมด $2^4 = 16$ รูปแบบ ดังแสดงในตารางที่ 2-2

ตารางในลักษณะดังเช่นตารางที่ 2-2 นี้เราจะเรียกว่าตารางค่าความจริง (Truth Table) ซึ่งจะแสดงค่าความจริงที่เป็นไปได้ทั้งหมดของ Formula หากกำหนดค่าความจริงต่างๆ ให้กับแต่ละอะตอมใน Formula

| P | Q | R | S | ~S | (P^Q) | (R<->(~S)) | (P^Q)->(R<->(~S)) |
|---|---|---|---|---|---|---|---|
| T | T | T | T | F | T | F | F |
| T | T | T | F | T | T | T | T |
| T | T | F | T | F | T | T | T |
| T | T | F | F | T | T | F | F |
| T | F | T | T | F | F | F | T |
| T | F | T | F | T | F | T | T |
| T | F | F | T | F | F | T | T |
| T | F | F | F | T | F | F | T |
| F | T | T | T | F | F | F | T |
| F | T | T | F | T | F | T | T |
| F | T | F | T | F | F | T | T |
| F | T | F | F | T | F | F | T |
| F | F | T | T | F | F | F | T |
| F | F | T | F | T | F | T | T |
| F | F | F | T | F | F | T | T |
| F | F | F | F | T | F | F | T |

*ตารางที่ 2-2 แสดงค่าความจริงของ (P ^ Q) -> (R <-> (~S))*

ต่อไปเราลองมาดู Formula G = ((P -> Q) ^ P) -> Q จะเห็นว่าใน Formula นี้มีอะตอมเพียง 2 ตัวคือ P และ Q ดังนั้นการแทนค่าทั้งหมดจะแสดงได้ดังตารางที่ 2-3 และจากตารางจะเห็นได้ว่า Formual G มีค่าเป็นจริงภายใต้การแทนค่าในทุกรูปแบบ ซึ่งเราจะเรียก Formula ประเภทนี้ว่า Tautology

| P | Q | (P->Q) | (P->Q)^P | ((P->Q)^P)->Q |
|---|---|---|---|---|
| T | T | T | T | T |
| T | F | F | F | T |
| F | T | T | F | T |
| F | F | T | F | T |

*ตารางที่ 2-3 ตารางค่าความจริงของ G = ((P -> Q) ^ P) -> Q*

ในลักษณะเช่นเดียวกันเราจะเรียก Formula ที่มีค่าเป็นเท็จทั้งหมดภายใต้การแทนค่าในทุกรูปแบบว่า Contradiction

**2.2.2 รูปแบบทั่วไป (Normal Form) ของ Propostional Logic**

ในการแทนปัญหาและการแก้ปัญหานั้น สิ่งหนึ่งที่จะพบอยู่เสมอคือการแปลงรูปของ Formula หนึ่งไปยังอีก Formula หนึ่ง โดยเฉพาะการแปลงไปสู่รูปแบบที่เรียกว่ารูปแบบทั่วไป (Normal Form) ซึ่งการแปลงนี้สามารถทำได้โดยการแทน Formula ด้วยอีก Formula หนึ่งที่ เท่ากัน (Equivalent) จนกระทั่งได้รูปแบบที่ต้องการ

แต่ปัญหาก็คือการเท่ากันของ Formula มีความหมายว่าอย่างไร เพื่อให้มีความชัดเจนที่จุดนี้ เราจะนิยามว่า Formula F และ G มีความเท่ากัน (Equivalent) ก็ต่อเมื่อค่าความจริงของ G เท่ากับค่าความจริงของ F ภายใต้การแทนค่าเดียวกันเช่น สมมติว่าเรามี Formula P -> G และ ~P v Q เราสามารถแสดงว่า Formula ทั้งสองเท่ากันได้โดยการแทนค่าความจริงดังตารางที่ 2-4 ซึ่งจากตารางเราจะสามารถสรุปได้ว่าทั้ง Formula P -> Q และ ~p v Q นั้นมีความเท่ากัน

| P | Q | (P->Q) | (~PvQ) |
|---|---|---|---|
| T | T | T | T |
| T | F | F | F |
| F | T | T | T |
| F | F | T | T |

*ตารางที่ 2-4 แสดงตารางค่าความจริงของ (P -> Q) และ (~P v Q)*

สำหรับเรื่องของการเท่ากันของ Formula นี้ ก็มีผู้ที่ได้พิสูจน์ไว้แล้วดังแสดงใน ตารางที่ 2-5 โดยกำหนดให้เครื่องหมายสี่เหลี่ยมที่ระบายสีดำแทน Tautology และเครื่องหมายสี่เหลี่ยมที่ระบายสีขาวแทน Contradiction และ G,H เป็น Formula ใดๆ และเพื่อให้ง่ายเราจะเรียกแต่ละข้อว่ากฎ (Laws)

กฎที่แสดงในตารางที่ 2-5 นี้สามารถพิสูจน์ได้โดยการใช้ตารางความจริง กฎที่ 2.3a, 2.3b เรียกว่า Commutative Laws กฎที่ 2.4a, 2.4b เรียกว่า Associative Laws กฎที่ 2.5a,2.5b เรียกว่า Distributive Laws กฎที่ 2.10a, 2.10b เรียกว่า De Morgan's Laws

| | | |
|---|---|---|
| (2.1) | $F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$ | |
| (2.2) | $F \rightarrow G = \sim F \vee G$ | |
| (2.3) | (a) $F \vee G = G \vee F$ | (b) $F \wedge G = G \wedge F$ |
| (2.4) | (a) $(F \vee G) \vee H = F \vee (G \vee H)$ | (b) $(F \wedge G) \wedge H = F \wedge (G \wedge H)$ |
| (2.5) | (a) $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$ | (b) $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$ |
| (2.6) | (a) $F \vee \ = F$ | (b) $F \wedge \ = F$ |
| (2.7) | (a) $F \vee \ =$ | (b) $F \wedge \ =$ |
| (2.8) | (a) $F \vee \sim F =$ | (b) $F \wedge \sim F =$ |
| (2.9) | (a) $\sim (\sim F) = F$ | |
| (2.10) | (a) $\sim (F \vee G) = \sim F \wedge \sim G$ | (b) $\sim (F \wedge G) = \sim F \vee \sim G$ |

*ตารางที่ 2-5 แสดงกฎต่างๆ ที่ใช้ในการแทนค่า*

และเพื่อที่จะสามารถอธิบายถึง Propositional Logic ต่อไปได้ เราจะนิยามคำบางคำดังต่อไปนี้

นิยาม Literal เป็นอะตอมหรือค่าลบ (negation) ของ atom

นิยาม Formula F จะสามารถเรียกว่าเป็น Conjunctive Normal Form ก็ต่อเมื่อ F อยู่ในรูปของ F = F1 ^ F2 ^ ... ^ Fn โดยที่ n >= 1 โดยที่ F1,F2,...,Fn เป็น Disjunctive ของ Literal เช่นหาก F = (P v ~Q v R) ^ (~P v Q) แล้ว F จะเป็น Conjunctive Normal Form

นิยาม Formula F จะสามารถเรียกว่าเป็น Disjunctive Normal Form ก็ต่อเมื่อ F อยู่ในรูปของ F = F1 v F2 v ... v Fn โดยที่ n >= 1 โดยที่ F1,F2,...,Fn เป็น Conjunctive ของ Literal เช่นหาก F = (P ^ ~Q ^ R) v (~P ^ Q) แล้ว F จะเป็น Disjunctive Normal Form

ถึงตรงนี้เราก็จะสามารถกล่าวได้ว่าสำหรับ Formula ใดๆ นั้นเราสามารถที่จะแปลงรูปของ Formula นั้นไปเป็น Normal Form ได้โดยใช้กฎจากตารางที่ 6 เช่น

**ตัวอย่าง** จงแปลง (P v ~Q) -> R ให้อยู่ในรูปของ Disjunctive Normal Form

$$(P \vee \sim Q) \rightarrow R = \sim(P \vee \sim Q) \vee R$$
$$= (\sim P \wedge \sim(\sim Q)) \vee R$$
$$= (\sim P \wedge Q) \vee R$$

**2.2.2.1 Clausal form**

มีรูปแบบหนึ่งของ Conjunctive Normal Form ที่มีการใช้งานในโปรแกรม เชิงตรรกมาก รูปแบบนั้นมีชื่อว่า Clausal Form (Kowalski,1974) ซึ่งจะหมายถึงรูปแบบ Conjunctive Normal Form ของ

A1,A2,...,Am <- B1,...,Bn

ซึ่งจะสามารถเขียนในรูปของ Conjunctive Normal Form ได้เป็น

A1 v ... v Am v ~B1 v ... v ~Bn

ซึ่งจาก Clausal Form ดังกล่าวมีอยู่รูปแบบหนึ่งซึ่งเป็นรากฐานของการโปรแกรมเชิงตรรก รูปแบบนั้นมีชื่อว่า Horn Clauses (Horn, 1951) ซึ่งก็คือ Clausal Form ที่มี m <= 1 นั่นเอง

**2.2.3 Logical Consequnce**

ในชีวิตประจำวัน เรามักจะพบว่าเหตุการณ์อย่างหนึ่งมักจะต้องเกิดหลังอีกเหตุการณ์หนึ่ง และจากความจริงที่ว่า Propositional Logic นั้นสามารถที่จะแทนปัญหาในชีวิตประจำวันได้ได้เช่นกัน จากความจริงที่ได้กล่าวมาทั้ง 2 ประการนี้เองที่จะนำเราไปสู่แนวความคิดของ Logical Consequence แต่ก่อนที่เราจะกล่าวถึง Logical Consequence เรามาดูตัวอย่างต่อไปนี้ก่อน

**ตัวอย่าง** กำหนดให้ราคาหุ้น (stock price) จะมีราคาลดลงเมื่อภาษี (Prime interest rate) เพิ่มขึ้น และประชาชน (People) จะไม่สบายใจ (Unhappy) เมื่อราคาหุ้นลดลง สมมติว่าภาษีเพิ่มขึ้น จงแสดงว่าประชาชนจะไม่สบายใจ

จากโจทย์ เราสามารถเขียนเป็นประโยคได้ดังต่อไปนี้

P = Prime interest rate goes up,

S = Stock prices go down,

U = Most people are unhappy.

และสามารถเขียนได้เป็น

1) If the prime interest rate goes up, stock prices go down.

2) If stock prices go down, most people are unhappy.

3) The prime interest rate goes up.

4) Most People are unhappy.

ซึ่งสามารถเขียนให้อยู่ในรูปของสัญญลักษณ์ได้เป็น

1') P -> S

2') S -> U

3') P

4') U

ซึ่งต่อไปเราก็จะแสดงว่า 4') เป็นจริง เมื่อ 1') ^ 2') ^ 3') เป็นจริง โดยขั้นแรกนั้นเราจะต้องแปลง 1') ^ 2') ^ 3') ให้อยู่ในรูปของ Normal Form เสียก่อน

| | | |
|---|---|---|
| ((P->S) ^ (S->U) ^ P) | = ((~P v S) ^ (~S v U) ^ P) | โดยใช้ 2.2 |
| | = (P ^ (~P v S) ^ (~S v U)) | โดยใช้ 2.3b |
| | = (((P ^ ~P) v (P^S)) ^ (~S v U)) | โดยใช้ 2.5b |
| | = ((f v (P ^ S)) ^ (~S v U) | โดยใช้ 2.8b |
| | = (P ^ S) ^ (~S v U) | โดยใช้ 2.6a |

= (P ^ S ^ ~S) v (P ^ S ^ U) โดยใช้ 2.5b

= (P ^ f) v (P ^ S ^ U) โดยใช้ 2.8b

= f v (P ^ S ^ U) โดยใช้ 2.7b

= P v S v U โดยใช้ 2.6a

และในกรณีเช่นนี้จะเรียก U ว่าเป็น Logical Consequence ของ (P->S), (S->U) และ P

**2.3 First-Order Predicate Logic**

ใน Propositional Logic นั้นเราทราบว่าหน่วยที่เล็กที่สุดของข้อมูล ก็คือ อะตอม และจากอะตอมเราก็สามารถที่จะสร้างเป็น Formula ได้ อย่างไรก็ตามใน Propositional Logic แต่ละอะตอมจะใช้แทนประโยคที่มีค่าความจริงเป็นจริงหรือเท็จเท่านั้น แต่ก็ยังมีเงื่อนไขบางประการที่ไม่สามารถใช้ Propositional Logic เขียนให้อยู่ในรูปแบบของ Formula ได้เช่นจากประโยคต่อไปนี้

P : Every man is mortal.

Q : Confucius is a man.

R : Confucius is mortal.

จากตัวอย่างข้างต้นจะเห็นได้ว่าเราไม่อาจพิสูจน์ได้ว่า R เป็น Logical Consequence ของ P และ Q ใน Propositional Logic ทั้งนี้เพราะโครงสร้างของ P,Q และ R นั้นไม่สามารถจะแทนด้วย Propositional Logic ได้ ในบทนี้เราจะเราจะโครงสร้างการแทนข้อมูลในอีกรูปแบบหนึ่งที่เรียกว่า First Order Predicate Logic (FOPL) ซึ่งมีความเหมาะสมในการแทนความจริงมากกว่า FOPL มีองค์ประกอบอยู่ 3 ส่วนได้แก่ terms, predicates, quantifies

และในเช่นเดียวกันกับ Propositional Logic เราจะเริ่มด้วยการกำหนด อะตอมซึ่งเป็นหน่วยที่เล็กที่สุดของข้อมูล เราลองมาดูตัวอย่างต่อไปนี้

สมมติว่าเราต้องการแทนประโยค "x is greater then 3" โดยใช้ FOPL เราสามารถทำได้โดยกำหนดพรีดิเขต GREATER (x,y) ซึ่งจะมีความหมายว่า x มีค่ามากกว่า y (เราอาจจะมองได้ว่าพรีดิเขตก็คือความสัมพันธ์ (Relation)) ดังนั้นประโยคข้างต้นเราสามารถแทนโดยเขียนว่า GREATER(x,3)

ในลักษณะเช่นเดียวกันเราจะแทนประโยค "x loves y" ด้วยพรีดิเขต LOVE (x,y) ดังนั้นประโยคที่ว่า "John loves mary" ก็จะสามารถแทนได้โดยใช้พรีดิเขต LOVE (john,mary)

นอกจากที่ได้กล่าวมาแล้วนั้น เราสามารถที่จะมีการแทนในลักษณะที่เป็นฟังก์ชันใน FOPL ได้อีกด้วยเช่นเราอาจใช้ plus (x,y) ในการแทนความหมายของ x+y ดังนั้นสำหรับ ประโยคที่ว่า "x+1 greater then x" ก็จะสามารถเขียนให้อยู่ในรูปของพรีดิเขตได้เป็น GREATER (plus (x,1),x)

และสิ่งที่ได้กล่าวมาทั้งหมดไม่ว่าจะเป็น GREATER (x,3) LOVE (john,mary) GREATER(plus (x,1),x) ล้วนจัดเป็นอะตอมใน FOPL ทั้งสิ้นโดยที่ GREATER และ LOVE เป็นเครื่องหมายแสดง พรีดิเขต (Predicate Symbol) ขณะที่ x เป็นตัวแปร และ 3, john, mary เป็นค่าคงที่ สำหรับ plus นั้น จัดเป็น

ฟังก์ชัน ดังนั้นหากจะกล่าวโดยสรุปก็คือใน FOPL จะมีองค์ประกอบอยู่ 4 ชนิดคือค่าคงที่ ตัวแปร ฟังก์ชัน และพรีดิเขต เท่านั้น ที่จะสามารถประกอบขึ้นมาเป็นพรีดิเขตได้

สำหรับฟังก์ชันแล้ว หากจะสังเกตให้ดีจะพบว่า ฟังก์ชันนั้นจะทำหน้าที่ในการแมป (Mapping) ระหว่างค่าคงที่ไปยังค่าคงที่ เช่น ฟังก์ชัน father (john) นั้นก็จะใช้ในการแทนบุคคลอีกคนหนึ่งเช่นกัน และเราเรียก father (John) นี้ว่าเทอม (Term) โดยที่สำหรับคำว่าเทอมใน FOPL แล้วจะหมายถึงค่าคงที่ ตัวแปร หรือฟังก์ชัน เช่น plus (plus (x,1),1), father (father (john)) ล้วนจัดเป็นเทอม

แต่สำหรับกรณีของพรีดิเขตแล้วจะทำหน้าที่ในการแมประหว่างค่าคงที่กับค่าความจริงซึ่งได้แก่ T และ F เช่นหาก GREATER เป็นพรีดิเขต ดังนั้น GREATER(5,3) จะเป็นจริง ขณะที่ GREATER(1,3) จะเป็นเท็จ หลังจากที่เราได้กำหนดความหมายของคำว่าเทอมต่อไปเราจะกำหนดความหมายของอะตอม ใน FOPL โดยกำหนดว่าหาก P เป็นเครื่องหมายพรีดิเขต (Predicate Symbol) และ t1,...,tn เป็นเทอมแล้ว P (t1,...,tn) คืออะตอม

เมื่อเรากำหนดอะตอมเป็นที่เรียบร้อยแล้ว เราก็สามารถใช้ตัวเชื่อมทางตรรก (Logical Connective) ได้เหมือนกับใน propositional Logic เพื่อใช้ในการสร้าง Formula และยิ่งไปกว่านั้นใน FOPL เรายังกำหนดเครื่องหมาย ∃ และ ∀ โดยจะเรียกว่า Existantial และ Universal Quantifier โดยที่ หาก x เป็นตัวแปรแล้ว ∀x จะหมายถึงสำหรับ x ทุกตัว (For all x) ขณะที่ ∃x หมายถึงสำหรับ x บางตัว (there exists an x) ลองดูตัวอย่างต่อไปนี้

สมมติว่าเรามีประโยค

1) Every rational number is a real number.

2) There exists a number that is a prime.

3) For every number x,there exists a number y such that x < y.

กำหนดให้ประโยค "x is a prime number" แทนด้วย P(x), "x is a rational number" แทนด้วย R(x) และ "x is less than y" แทนด้วย LESS(x,y) ดังนั้นสำหรับประโยคทั้ง 3 ข้างต้น เราสามารถเขียนให้อยู่ในรูปของ FOPL ได้ดังนี้

1) (∀x) (Q(x) -> R(x))

2) (∃x) P(x)

3) (∀x) (∃y) LESS(x,y)

แต่จะเรียกทั้งลักษณะการเขียนในลักษณะเช่นนี้ว่า Formula แต่ก่อนที่เราจะกล่าวถึงเรื่องของ Formula ต่อไป เราจะมาทำความรู้จักกับคำ 2 คำคือ free และ bound โดยตัวแปรใดที่อยู่ใน Formula จะถูกเรียกว่า bound ก็ต่อเมื่อเกิดขึ้นอยู่ในช่วงของ Quantifier ของตัวแปรนั้น และจะถูกเรียกว่า free ก็ต่อ เมื่อตัวแปรนั้นไม่เป็นแบบ bound เช่น สมมติว่าเรามี Formula (∀X) P(x,y) ตัวแปร x จะเป็นตัวแปรแบบ bound แต่สำหรับตัวแปร y นั้นจะเป็นแบบ free อย่างไรก็ตามสำหรับตัวแปรหนึ่งใน Formula หนึ่งอาจ

จะเป็นทั้งแบบ bound และ free เลยก็เป็นได้ เช่น $(\forall x)\ P(x,y) \wedge (\forall y)\ Q(y)$ ตัวแปร y จะเป็นทั้งแบบ bound และ free

จากรูปแบบที่ได้กล่าวมาทั้งหมด จะเห็นได้ว่าเราสามารถใช้ FOPL ในการแทนความจริงต่างๆ ในชีวิตประจำวันได้

### 2.3.1 การแทนค่าของ Formula ใน First Order Predicate Logic

ใน Propositional Logic การแทนค่าก็คือการกำหนดค่าความจริงให้กับอะตอม แต่สำหรับ FOPL แล้วเนื่องจากมีตัวแปรที่เกี่ยวข้องมากกว่า ดังนั้นการการกำหนดการแทนใน FOPL เราจะใช้วิธีกำหนด 2 ประการ ประการแรกคือกำหนดโดเมน (Domain) และกำหนดค่าคงที่ ฟังก์ชัน และพรีดิเขต เช่น หากเรามี Formula $(\forall x)\ P(x)$ และ $(\exists x)\ \sim P(x)$

ขั้นแรกเราจะกำหนดให้โดเมนเป็น 1,2 และกำหนดค่าให้ P

| P(1) | P(2) |
|---|---|
| T | F |

จากตัวอย่างที่เรากำหนดค่า จะเห็นได้อย่างชัดเจนว่า $(\forall x)\ P(x)$ นั้นเป็นเท็จในการแทนค่าเพราะ P(x) ไม่เป็นจริงในทั้งสองกรณี แต่ $(\exists x)\ \sim P(x)$ เป็นจริงเพราะมีอย่างน้อย 1 ค่าของการแทนที่ทำให้ $\sim P(x)$ เป็นจริง

จะเห็นได้ว่าเราไม่สามารถใช้เทคนิคของการสร้างตารางค่าความจริงใน FOPL ได้ในการหาว่า Formula ใดเป็น T หรือ F นั้น จะต้องอาศัยการแทนค่าที่ประกอบด้วยการกำหนดขอบเขตของการแทนค่าหรือที่เรียกว่าโดเมน จากนั้นจึงแทนค่าในโดเมนให้กับส่วนของพรีดิเขต และใช้การพิจารณาจึงจะสามารถระบุค่าความจริงได้

## 2.4 การโปรแกรมเชิงตรรก (Logic Programming)

จากที่ได้กล่าวมาแล้วทั้งหมดนั้นเป็นทฤษฎีทางด้าน Logic ซึ่งจะเห็นได้ว่ามีความยุ่งยากซับซ้อนมาก ปัญหาบางปัญหาเมื่อแทนด้วย FOPL แล้วการแก้ปัญหาจะยุ่งยากมาก จนกระทั่งมีการนำคอมพิวเตอร์เข้ามาใช้ในการแก้ปัญหาทาง FOPL ซึ่งทฤษฎีที่นับว่ามีความสำคัญ อันก่อให้เกิดก้าวกระโดดในการศึกษาวิทยาการในสาขานี้ได้แก่ Resolution ซึ่งคิดโดย Robinson [1979] โดยมีขอบเขตอยู่ว่าปัญหาจะต้องอยู่ในรูปของ Horn Clause เท่านั้น

วิธีการ Resolution นี้มีข้อดีหลายประการคือ ง่าย มีความถูกต้องสูง และมีความสมบูรณ์ในตัวเอง ซึ่งวิธี Resolution ซึ่งทฤษฎี Resolution นี่เองที่เป็นรากฐานของภาษาโปรลอก (Prolog) [Roussel 1975] ซึ่งเป็นภาษาที่มีผู้นิยมใช้มากในการโปรแกรมเชิงตรรก

### 2.4.1 Resolution

ทฤษฎี Resolution นี้จริงๆ แล้วก็เป็น กฎการวินิจฉัย (Inference Rule) อันหนึ่งโดยมีข้อดีที่ว่าเหมาะกับการใช้งานโดยคอมพิวเตอร์ ทฤษฎี Resolution มีหลักการดังต่อไปนี้ Clause จำนวน 2 Clause

สามารถที่จะ Resolve กับอีก Clause ได้หากใน Clause หนึ่งมี Literal ที่เป็นบวก (Positive) และอีก Clause หนึ่งมี Literal ที่เป็นลบ (Negative) ที่มาจากพรีดิเขตเดียวกัน และ Argument ของ Clause ทั้งสองนี้สามารถที่จะรวมกัน (Unified) กันได้ ลองมาดูตัวอย่างต่อไปนี้

P(a) v ~Q(b,c) (1)

Q(b,c) v ~R(b,c) (2)

เนื่องจากใน Clause ที่ 1 มี Literal ที่เป็นลบคือ ~Q(b,c) ในขณะที่ Clause ที่ 2 มี Literal ที่มาจากพรีดิเขตเดียวกันหากเป็น Literal ที่เป็นบวกและ Argument ของทั้ง 2 Literal สามารถที่จะรวม (Unified) กันได้ นั่นคือ b ก็จะรวมกับ b และ c ก็จะรวมกับ c ดังนั้นผลรวมของ Clause (1) และ (2) โดยวิธีการ Resolution นี้คือ Clause ที่ (3) ดังข้างล่างนี้ และจะเรียก Clause ที่ (3) นี้ว่า Resolvent

P(a) v ~R(b,c) (3)

### 2.4.2 โปรลอก

จาก FOPL และ Resolution ก็ได้มีนักวิจัยที่ชื่อว่า Kowalski ซึ่งมีส่วนสำคัญต่อการพัฒนาภาษาโปรลอกได้มีความเห็นว่าภาษาทางตรรก (Logic Language) น่าจะสามารถนำมาใช้เป็นภาษาทางโปรแกรมได้ (Progeamming Langiage) และนี่ก็เป็นจุดเริ่มต้นของภาษาโปรลอก

#### 2.4.2.1 โครงสร้างของภาษาโปรลอก

ภาษาโปรลอกเป็นภาษาทางคอมพิวเตอร์ที่ ใช้ในการแก้ปัญหาที่เกี่ยวข้องกับวัตถุ (Objects) และความสัมพันธ์ระหว่างวัตถุนั้น ตัวอย่างเช่นจากประโยคที่ว่า "John owns the book" เราจะกำหนดความสัมพันธ์ชื่อ Ownership (การเป็นเจ้าของ) ระหว่างวัตถุ "john" และ "the book" และนอกจากนั้นความสัมพันธ์นี้ จะต้องเป็นความสัมพันธ์ที่มีลำดับอีกด้วย เพราะ "john" เป็นเจ้าของหนังสือ แต่หนังสือไม่ได้เป็นเจ้าของ "john"

แต่ความสัมพันธ์ที่กล่าวถึงนี้ก็ไม่ได้หมายถึงความสัมพันธ์ระหว่างวัตถุเสมอไป เช่น จากประโยคที่ว่า "The jewel is valuable" นั่นคือความสัมพันธ์ที่จะเกี่ยวข้องกับวัตถุ "jewel" ก็คือ "being valuable" ซึ่งจะเห็นได้ว่าไม่มีอีกวัตถุอีกตัวหนึ่งที่จะสัมพันธ์ด้วย ในภาษาโปรลอกก็จะต้องมีการกำหนดความสัมพันธ์ในลักษณะเช่นนี้ และขึ้นอยู่กับว่าเราจะนำคอมพิวเตอร์ไปใช้งานในรูปแบบใด

อย่างไรก็ตามในโปรลอกนั้นยังมีรูปแบบความสัมพันธ์ระหว่างวัตถุที่สำคัญที่ยังจะต้องกล่าวถึงอยู่อีกรูปแบบหนึ่ง ก่อนที่เราจะเริ่มกล่าวถึงการเขียนโปรแกรม ความสัมพันธ์ในรูปแบบนั้น โดยทั่วไปเรามักจะคุ้นเคยกับมันอยู่แล้ว โดยที่เราอาจจะไม่รู้ตัว นั่นคือการใช้กฎในการอธิบายความสัมพันธ์ระหว่างวัตถุต่างๆ เช่น จากประโยคที่ว่า "Two people are sisters if they are both female and have the same parents" ซึ่งหมายความว่าทั้งสองคนจะเป็น sister กันนั้นเมื่อทั้งสองคนเป็นผู้หญิงและมีพ่อแม่เดียวกัน

เมื่อเราได้กล่าวถึงความสัมพันธ์ในรูปแบบต่างๆ แล้ว ต่อไปเราก็จะกล่าวถึงการโปรแกรมภาษาโปรลอกในคอมพิวเตอร์ ซึ่งจะประกอบด้วยขั้นตอนต่อไปนี้

- กำหนดความจริง (facts) เกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น
- กำหนดกฎ (Rule) เกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น
- ถามคำถาม (Query) ที่ต้องการเกี่ยวกับเกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น

ตัวอย่างเช่น หากเรากำหนดกฎเกี่ยวกับ sister ที่ได้กล่าวไปแล้ว จากนั้นหากเราถามโปรลอกว่า Mary และ Jane เป็น sister กันหรือไม่ โปรลอกจะค้นหาว่าเราได้กำหนดอะไรไว้ในโปรแกรมบ้าง และจากนั้นก็จะตอบว่า yes หรือ no ซึ่งก็จะขึ้นกับว่าความจริงที่เราได้กำหนดไว้ในโปรแกรมนั้น สอดคล้องกับกฎที่เรากำหนดไว้หรือไม่ ดังนั้นหากจะมองให้ดีแล้วจะเห็นได้ว่าภาษาโปรลอกนั้นแท้จริงก็เพียงแต่เป็นที่เก็บความจริงและกฎ และความสามารถในการหาคำตอบจากความจริงและกฎนั้นเท่านั้น

**2.4.2.1.1 ความจริง (facts)**

ในขั้นแรกนี้เราจะกล่าวถึง facts เกี่ยวกับวัตถุต่างๆ สมมติว่าเราต้องการกำหนดในภาษาโปรลอกว่า "John likes Mary" ซึ่งก็จะมีวัตถุ 2 อย่างคือ Mary และ John และจะเรียกความสัมพันธ์นี้ว่า likes และในภาษาโปรลอกเราจะกำหนดความจริงนี้ดังนี้

```
likes (john,mary).
```

ซึ่งจะเห็นว่ามีส่วนที่สำคัญ 3 ส่วนดังนี้

- ชื่อของทุกความสัมพันธ์ (Relation) และวัตถุ (Objects) จะต้องเริ่มต้นด้วยอักษรตัวเล็ก
- จะต้องเขียนความสัมพันธ์ก่อนและคั่นแต่ละวัตถุด้วยเครื่องหมาย "," และใช้เครื่องหมายวงเล็บครอบทุกวัตถุ
- จะต้องใช้เครื่องหมาย "." ปิดท้าย facts

เมื่อเราได้กำหนดความสัมพันธ์ระหว่างวัตถุโดยใช้ facts แล้วต่อไปเราจะเรียกชื่อของความสัมพันธ์ว่าพรีดิเขต (Predicate) และเรียกแต่ละวัตถุว่า Arguments ดังนั้น fact ที่ว่า likes (john,mary) ก็เป็นพรีดิเขตที่มี 2 Argument และจำนวน Argument ในพรีดิเขตจะมีเท่าใดก็ได้ ตั้งแต่ 1 ขึ้นไป และในโปรลอกกลุ่มของ facts จะเรียกว่า database

**2.4.2.1.2 คำถาม (Questions)**

เมื่อเราได้กำหนดความจริงบางประการให้กับโปรลอก เราสามารถจะถามปัญหาบางอย่างได้ ในภาษาโปรลอกคำถามจะมีความคล้ายกับ facts มากเว้นแต่จะมีเครื่องหมายพิเศษเท่านั้น เช่น

```
?- owns(mary,book).
```

จากคำถามดังกล่าวก็จะมีความหมายว่า mary เป็นเจ้าของ book หรือไม่ และเมื่อเราถามคำถามดังกล่าวกับโปรลอก โปรลอกจะค้นหาใน database ของมันว่ามี fact ดังกล่าวกำหนดไว้หรือไม่ หากมีโปรลอกก็จะตอบว่า yes มิฉะนั้นจะตอบว่า no เช่นหากเรามี database ต่อไปนี้

likes (joe,fish).

likes (joe,mary).

likes (mary,book).

likes (john,book).

เมื่อเราใส่ fact ทั้งหมดในโปรลอก และหากเราถามคำถามต่อไปนี้โปรลอกก็จะตอบมาในบรรทัดต่อไปที่เราถามคำถาม

?- likes(joe,money).

no

?- likes(mary,joe).

no

?- likes(mary,book).

yes

? king(john,france).

no

ในตอนต่อไป เราจะกล่าวถึงคำถามในลักษณะที่ว่า ใครบ้างที่ mary ชอบ

**2.4.2.1.3 ตัวแปร (variable)**

จากที่ผ่านมาเราจะสามารถถามได้ในลักษณะที่ว่า "Does John like Mary?" ซึ่งจะได้คำตอบในลักษณะของ yes หรือ no แต่สำหรับคำถามในลักษณะของ "Does John like X" เราไม่ทราบว่า X แทนอะไร แต่สำหรับโปรลอกแล้ว เราไม่เพียงแต่ใช้ชื่อเฉพาะเท่านั้น แต่ยังสามารถใช้ชื่อในลักษณะของ X ในการแทนวัตถุ โดยจะเรียกว่าตัวแปร (Variable) และในโปรลอกจะใช้คำที่ขึ้นต้นด้วยอักษรตัวใหญ่ในการแทนตัวแปร

เมื่อโปรลอกถามปัญหาที่มีตัวแปรอยู่ โปรลอกจะค้นหาผ่าน facts ทั้งหมดที่มีอยู่ซึ่งเช่นคำถามที่ว่า :- likes (john,X) และ database ข้างต้นโปรลอกก็จะพยายามที่จะหา พรีดิเขตที่ชื่อ likes และมี john เป็น argument ตัวแรก จากนั้นก็จะแทน X ด้วย argument ตัวที่ 2 ของพรีดิเขตนั้น ซึ่งจะได้คำตอบว่า

X = book

**2.4.2.1.4 ตัวเชื่อม (Conjunctions)**

สมมติว่าเราถามคำถามที่มีความซับซ้อนขึ้นไปอีก เช่น "Do John and Mary like each other?" วิธีหนึ่งที่เราจะหาคำตอบก็คือขั้นแรกก็จะหาว่า John likes mary. หรือไม่ จากนั้นก็จะถามว่า mary likes John. หรือไม่ ดังนั้นจะเห็นว่าคำถามประกอบด้วย 2 เป้าหมายย่อย (sub goal) ที่โปรลอกจะต้องหา แต่คำถามในลักษณะนี้จะมีการใช้บ่อยมากและสามารถถามได้ในลักษณะต่อไปนี้

?- likes(john,mary), likes(mary,john).

สมมติว่าเรามี Database

likes(mary,food).

likes(mary,wine).

likes(john,wine).

likes(john,mary).

ในที่นี้เครื่องหมาย "," จะใช้ในลักษณะความหมายว่า "และ" (And) และใช้แบ่งเป้าหมายย่อยที่ประกอบกันเป็นคำตอบ เมื่อลำดับของเป้าหมาย (Goal) ที่คั่นด้วย "," ถูกใช้ในโปรลอก โปรลอกจะพยายามหา fact ที่สอดคล้องกับแต่ละเป้าหมายย่อยในคำถาม หากมี fact ที่สอดคล้องกับทุกๆ เป้าหมายย่อย โปรลอกก็จะใช้คำตอบว่า yes มิฉะนั้นก็จะเป็น no ซึ่งในกรณีนี้จะเห็นได้ว่าเป้าหมายที่ 2 เป็นเท็จ ดังนั้นคำตอบจึงเป็น no

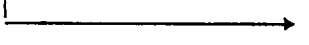
การใช้ตัวเชื่อมและการใช้ตัวแปรสามารถใช้ร่วมกัน เพื่อถามคำถามที่ซับซ้อนกว่านี้ได้ เช่น คำถามที่ว่า "Is there anything that John and Mary both like?" จากคำถามนี้ประกอบด้วยเป้าหมายย่อยดังนี้ ทำการหาว่ามีบางอย่าง X ที่ Mary ชอบและจากนั้นหาว่า John ชอบ X เช่นเดียวกันหรือไม่ สามารถเขียนเป็นภาษาโปรลอกได้ดังนี้

?- likes(mary,X),likes(john,X).

โปรลอกจะพยายามหาคำตอบของเป้าหมายแรกก่อน หากพบเป้าหมายแรกใน Database ก็จะพยายามหาคำตอบของเป้าหมายที่ 2 ต่อไป หากพบว่าเป้าหมายที่ 2 อยู่ใน Database ก็จะได้คำตอบออกมา อย่างไรก็ตามในกรณีของคำถามในลักษณะเช่นนี้ โปรลอกจะสร้าง Marker ขึ้นมาสำหรับแต่ละเป้าหมายย่อย เพราะหากว่าเป้าหมายที่ 2 เกิดเป็นเท็จขึ้นมา โปรลอกก็จะค้นหาเป้าหมายที่ 1 ใน Database ต่อจากเดิมเพื่อหาคำตอบอื่นที่สอดคล้องกับเป้าหมายแรกได้อีก กลไกเช่นนี้เราเรียกว่า backtracking ซึ่งเป็นลักษณะเฉพาะตัวของภาษาโปรลอก และเพื่อให้เข้าใจกลไกในการ backtrack ดีขึ้นลองมาดูการทำงานทีละขั้นตอนดังนี้

จากข้อมูลเดิมที่ผ่านมา และคำถามที่ว่า

?- likes(mary,X),likes(john,X).



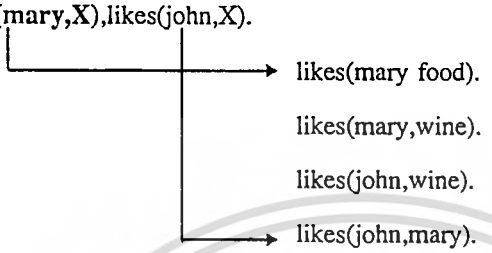
likes(mary food).

likes(mary,wine).

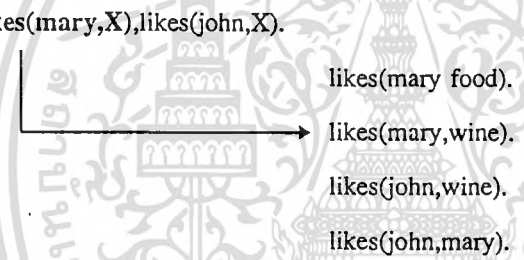
likes(john,wine).

likes(john,mary).

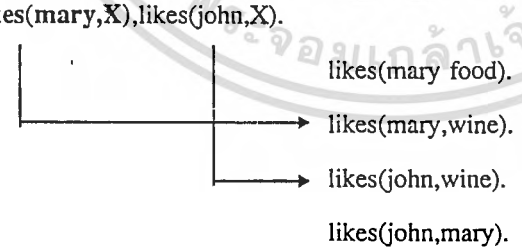
จะเห็นว่าเป้าหมายแรก มี fact ที่สอดคล้องอยู่ในฐานข้อมูลได้แก่ likes(mary,food). ดังนั้น Marker ของเป้าหมายแรกจะอยู่ที่ fact ดังกล่าว และ X ก็จะมีค่าเป็น food จากนั้นจะพยายามหาคำตอบของเป้าหมายที่ 2 คือ likes(john,food). ซึ่งไม่มีใน Database




จากนั้นก็จะทำการ backtrack โดยทิ้งค่าเดิมของ X ไปและทำการหาค่าของ X ใหม่จากเป้าหมายย่อยแรก ต่อจากเดิมที่ Marker ของเป้าหมายแรกชี้ไว้ ซึ่งก็จะได้คำตอบว่า X มีค่าเป็น wine




จากนั้นก็จะหาคำตอบของเป้าหมายที่ 2 ก็คือ likes(john,wine) ซึ่งปรากฏว่ามีใน Database ดังนั้นทั้งสองเป้าหมายก็เป็นจริง และได้คำตอบว่า X=wine




**2.4.2.1.5 กฎ (Rule)**

ทีนี้หากเราต้องการจะสร้างความสัมพันธ์ที่ว่า "John likes all person" หากเราจะสร้างเป็น fact เพื่อเก็บใน Database ก็จะต้องสร้างความสัมพันธ์กับทุกคน ดังนี้

likes(john,david).

like(john,mary).

...

และหากเราเพิ่มคนใดคนหนึ่งเข้าไปใน Database ก็จะต้องสร้างความสัมพันธ์ในลักษณะเช่นนี้ขึ้นมาทุกครั้ง ในลักษณะเช่นนี้จะเห็นได้ว่าเป็นรูปแบบที่ไม่สะดวกต่อการใช้งานอย่างยิ่ง และหากเราแทนด้วยอีกรูปแบบหนึ่งของโปรลอกคือกฎ (Rule) ซึ่งเป็นรูปแบบที่เหมาะกับการเก็บความจริงในลักษณะเช่นนี้กว่า จะได้ดังนี้

```
likes(john,X) :- person(X).
```

โดยที่เครื่องหมาย ":-" จะแทนความหมายของ "if" ดังนั้นในความสัมพันธ์ข้างต้นจะมีความหมายว่า "John likes any object provided it is a person." ซึ่งจะสังเกตุได้ว่า กฎ ก็จะมีความหมายเหมือนกับเป็นการสร้างความสัมพันธ์ขึ้นมาใหม่บนความสัมพันธ์เก่า ซึ่งหากจะมองว่าข้างหลังของเครื่องหมาย ":-" เป็นเป้าหมายและหากเป้าหมายเป็นจริงก็จะได้คำตอบออกมา

และเพื่อให้มีความเข้าใจกับกลไลของการใช้กฎในโปรลอก เราจะลองสมมติข้อมูลและสร้างกฎขึ้นมาบนข้อมูลต่อไปนี้

```
male (albert).
male (edward).
female (alice).
female (victoria).
parents (edward,victoria,albert).
parents (alice,victorai,albert).
```

และมีกฎว่า

```
sister_of(X,Y) :- female(X),
                  parents(X,M,F),
                  parents(Y,M,F).
```

โดยที่ความสัมพันธ์ sister_of นั้นหมายถึงความสัมพันธ์ของ X และ Y ซึ่งหมายความว่า X จะเป็น sister ของ Y เมื่อ X เป็นผู้หญิงและมีพ่อแม่เดียวกัน และเมื่อเราถามว่า :- sister_of(alice,edward). โปรลอกก็จะทำงานตามขั้นตอนต่อไปนี้

- จากคำถามเมื่อค้นหาใน Database ก็จะพบว่า sister_of เป็นกฎก็จะเปรียบเทียบก็จะได้ว่า X=alice และ Y=edward จากนั้นโปรลอกก็จะพยายามที่จะหาคำตอบของเป้าหมายย่อยทั้ง 3 ต่อไป
- ครั้งแรกจะทำการค้นหาว่า female(alice). เป็นจริงหรือไม่ ซึ่งก็จะพบว่าเป็นจริงดังนั้นก็จะทำการค้นหาเป้าหมายที่ 2 ต่อไป
- จาก parents (alice,M,F) ก็จะได้ว่า M=victoria และ F=albert
- จากนั้นก็จะค้นหา fact parents (edward,victoria,albert) ซึ่งปรากฏว่ามีในฐานข้อมูล

จากขั้นตอนทั้งหมดที่ผ่านมาก็จะพบว่าความเป้าหมายย่อยทั้ง 3 เป็นจริง ดังนั้นโปรลอกก็จะตอบกลับมาว่า yes อย่างไรก็ตามในแง่ของกฎแล้วเราสามารถที่สามารถตั้งคำถามในลักษณะของ sister_of (alice,X) ได้อีกด้วย ซึ่งกลไกในการหาคำตอบของกฎนี้ก็จะคล้ายกับกลไกการหาคำตอบแบบธรรมดา รวมทั้งการทำ backtrack ด้วย ซึ่งจะไม่กล่าวในที่นี้ และที่ได้กล่าวมาทั้งหมดนี้ เพื่อเป็นแนวทางและเป็นทฤษฎีพื้นฐานที่จะใช้ในบทต่อไป

**หนังสืออ้างอิง**

[1] John Malpas : "*Prolog : A relational Language And its Application*" ,Prentice-Hall International Editions

[2] W.F. Clocksin, C.S. Mellish, "*Programming in Prolog*", Springer-Verlag Berlin Heidelberg New York Tokyo,1984

# บทที่ 3
# แบบจำลองข้อมูล NIAM

## 3.1 แบบจำลองข้อมูล NIAM (The NIAM Conceptual Schema)

แบบจำลองข้อมูลแบบไนแอมคิดค้นขึ้นโดย Prof. G.M. Nijssen และ E. D. Falkenberg โดยไนแอมเป็นแบบจำลองข้อมูลระดับแนวคิด (Conceptual Model) ซึ่งมีพื้นฐานมาจากภาษาธรรมชาติโครงสร้างลึก (Deep Structured Natural Language) คือภาษาที่มีรูปประโยคเป็น <ประธาน, กริยา, กรรม> เท่านั้น

การสร้างแบบจำลองข้อมูลไนแอมจะมีพื้นฐานอยู่บนการกำหนดตัวอย่างข้อมูล และหลังจากที่ได้ผ่านกระบวนการ (Procedure) ที่ได้กำหนดไว้ เราก็จะได้แผนภาพที่มีความหมายในการแทนแบบจำลองข้อมูลดังกล่าวได้ และเนื่องจากแบบจำลองข้อมูลนี้คิดขึ้นครั้งแรกโดย Nijssen ดังนั้นจึงเรียกแบบจำลองข้อมูลนี้ว่า NIAM (Nijssen's Information Analysis Methodology)

หลังจากนั้นไนแอมก็ได้ถูกนำมาพัฒนาต่อโดย Nijssen และ T.A. Halpin ที่มหาวิทยาลัย Queensland ประเทศออสเตรเลีย จนกระทั่งมีโครงสร้างและรูปแบบการใช้ที่เป็นมาตรฐานมากยิ่งขึ้น และเนื่องจากวิธีการที่พัฒนาขึ้นนี้จะเน้นที่ชนิดความจริง (Fact Type) ดังนั้นอาจจะเรียกไนแอมว่าเป็น Fact-Oriented Modeling และแม้ว่าไนแอมจะมีความสัมพันธ์และมีความคล้ายคลึงกับ ER Model (Entity-Relationship Model) หากแต่ NIAM นั้นจะสามารถทำความเข้าใจได้ง่าย และเป็นธรรมชาติกว่า นอกจากนั้นไนแอมยังช่วยในการออกแบบระบบฐานข้อมูลสัมพันธ์ได้ดีกว่า โดยสามารถที่จะแปลงรูปเป็น 5NF ได้เลย ในขณะที่ ER Model จะแปลงได้แค่ 3 NF เท่านั้น

## 3.2 ขั้นตอนการออกแบบ NIAM

แม้ว่าการออกแบบฐานข้อมูลโดยวิธีไนแอมจะเป็นวิธีการที่ง่าย และมีความเป็นธรรมชาติมากกว่าวิธีอื่น ๆ ก็ตาม แต่หลักการของการออกแบบแบบจำลองข้อมูล ก็ยังนับได้ว่าเป็นงานที่ซับซ้อนอยู่ดี ดังนั้นเพื่อให้สามารถทำความเข้าใจได้ง่าย จึงแบ่งออกการออกแบบออกเป็นงาน 9 ขั้นตอนด้วยกัน ดังนี้

### 3.2.1 ขั้นตอนที่ 1 From Example to elementary facts

ในการออกแบบระบบสารสนเทศนั้น ขั้นตอนแรกที่เราจะต้องทำ ก็คือหาตัวอย่างข้อมูลที่ใช้งานอยู่ในระบบ ซึ่งตัวอย่างข้อมูลที่สำคัญก็คือ รายงาน (Output Report) และแบบฟอร์มที่ใช้ในการรับข้อมูลเข้าสู่ระบบ (Input Form) ซึ่งอาจจะอยู่ในรูปของตาราง ภาพ หรือข้อความใดๆ ก็ตาม สำหรับในตารางที่ 3-1 นั้นเป็นตัวอย่างของข้อมูลของตารางเรียน

| Tutorial group | Time | Room | Student# | Student name |
|---|---|---|---|---|
| A | Mon 3 p.m. | CS.718 | 302156 | Bloggs FB |
| | | | 180064 | Fletcher JB |
| | | | 278155 | Jackson M |
| | | | 334067 | Jones EP |
| .. | .. | .. | .. | .. |
| B1 | Tue 2 p.m. | E-B18 | 266010 | Anderson AB |
| | | | 348112 | Bloggs FB |
| .. | .. | .. | .. | .. |

*ตารางที่ 3-1 รายงานการจองเวลาเรียนของนักศึกษา*

จากนั้นเราก็จะนำข้อมูลในตารางดังกล่าวมาเขียนให้อยู่ในรูปของ "ความจริงพื้นฐาน" หรือ Elementary Facts เช่น จากข้อมูลในบรรทัดแรกเราสามารถเขียนได้เป็น

Tute group A meets at Time Mon 3 p.m.

Tute group A is held in Room CS-718.

Student 302156 belongs to Tute group A.

Student 302156 has Name 'Bloggs FB'.

จะเห็นได้ว่าการเขียนในรูปแบบนี้จะสามารถแสดงความสัมพันธ์ของข้อมูลได้มากกว่าแบบตาราง เช่น นักศึกษาที่มีชื่อและเลขประจำตัวที่อยู่ในแถวเดียวกัน ก็จะหมายถึงคนเดียวกัน อย่างไรก็ตาม แม้ว่าข้อมูลจะเป็นแบบอื่น เช่น อาจจะเป็นแผนภูมิต่างๆ แต่งานในขั้นแรกนี้ก็จะต้องนำมาเขียนเป็นความจริงพื้นฐานทั้งหมด

คำว่า ความจริงพื้นฐาน นั้นอาจจะถูกมองว่าเป็นประโยคที่อยู่ในรูปแบบของ "วัตถุ กระทำ สิ่งต่างๆ" (Particular Object Play Particular Role) เช่น Ann smoke. เราสามารถมองว่า Ann (Object) กระทำ "สูบบุหรี่" (Role) นอกจากนั้นความจริงพื้นฐานยังจะต้องเป็นข้อมูลพื้นฐานที่ไม่สามารถแบ่งแยกได้อีกด้วย

แต่ก่อนที่เราจะกล่าวถึงความจริงพื้นฐานกันต่อไป เราจะกล่าวถึง นิยาม ของคำว่า เอนติตี้ (Entity) เอนติตี้เป็นหน่วยข้อมูลพื้นฐานที่สุดของระบบข้อมูลที่เราให้ความสนใจ เช่น ในระบบข้อมูลของมหาวิทยาลัยนั้น นักศึกษา ภาควิชา อาจารย์ ล้วนจัดเป็นเอนติตี้ และเรียกเซ็ทของเอนติตี้ว่าชนิดเอนติตี้ (Entity Type) ซึ่งมีสมาชิกเป็นตัวอย่างเอนติตี้ (Entity Instance) แต่ละตัวอย่างเอนติตี้จำเป็นต้องมีชื่อเรียก ซึ่งอาจเป็นชื่อ, รหัส อย่างใดอย่างหนึ่ง เช่น เอนติตี้นักศึกษา จะเรียกโดยใช้รหัสนักศึกษาเป็นชื่อเรียก การกำหนดว่าแต่ละชนิดเอนติตี้ควรใช้อะไรเป็นชื่อเรียก ทำได้โดยระบุชนิดเลเบิ้ล (Label Type) สำหรับเอนติตี้นั้น เช่น ชนิดเอนติตี้ Person อาจจะใช้ชนิดเลเบิ้ล Surname เป็นชื่อเรียก

ดังนั้นสำหรับข้อมูลที่ว่า "นักศึกษาเลขประจำตัว 311308 ลงเรียนในวิชา CS112" หากจะเขียนให้อยู่ในรูปของ Elementary Facts สามารถเขียนให้อยู่ในรูปของประโยคภาษาธรรมชาติได้ดังนี้

The STUDENT with student# '311308'

**studies**

the SUBJECT with code 'CS112'.

โดยที่ STUDENT และ SUBJECT จะเป็นชนิดเอนติตี้ (Entity Type) ส่วน student# และ code เป็นชนิดเลเบิ้ล (Label Type) และ '311308' และ 'CS112' เป็นตัวอย่างข้อมูล (Instant) โดยมี studies เป็นการกระทำ ( Role)

และจากประโยคข้างต้นจะสังเกตุว่าเราจะเขียนชนิดเอนติตี้คู่กับชนิดเลเบิ้ลเสมอ อย่างไรก็ตาม เราอาจเขียนความสัมพันธ์ข้างต้นเสียใหม่ ดังนี้

The SUBJECT with code 'CS112'

**was studied by**

the STUDENT with student# '311308'

ซึ่งจะเห็นได้ว่าการเขียนในทั้งสองรูปแบบสามารถแทนความจริงเดียวกันได้ โดยในประโยคหลังนี้ was studied by เป็น Role โดยที่เราจะเรียกการเขียนประโยคที่ต่างกันนี้ว่า มี "Surface Structure" ที่ต่างกัน หากแต่มี "Deep Strcuture" อันเดียวกัน ดังนั้นหากจะเขียนความสัมพันธ์ดังกล่าวในรูปแบบใดรูปแบบหนึ่ง ก็คงจะเป็นการไม่ถูกต้องนัก และเพื่อให้สื่อถึงความสัมพันธ์เดียวกันทุกครั้ง เราจะเขียนความจริงนี้ใหม่ในรูปแบบต่อไปนี้

Study { [The STUDENT with student# '311308',studies],
[The SUBJECT with code 'CS112',was studied by] }

โดยมี Study เป็นชนิดความจริง

และที่เราได้กล่าวมาทั้งหมดนั้น เป็นการกล่าวถึงการเปลี่ยนจากตัวข้อมูลที่อยู่ในรูปแบบต่างๆ ให้มาอยู่ในรูปแบบของ ความจริงขั้นพื้นฐาน ซึ่งเป็นรูปแบบที่สามารถแสดงความสัมพันธ์ของข้อมูลแต่ละตัวได้ดีกว่า

### 3.2.2 ขั้นตอนที่ 2 Fisrt Draft of concepturl schema diagram

งานหลักในขั้นตอนนี้ก็คือการนำ Elementary Facts มาวาดเป็นแผนภาพ โดยจะสมมติข้อมูลขึ้นมาชุดหนึ่ง ซึ่งเป็นข้อมูลว่าใครขับรถเลขทะเบียนอะไร ดังตารางที่ 3-2

| *Drives* | *Person* | *Car* |
|---|---|---|
| | **Adams B** | **235PZN** |
| | **Jones E** | **235PZN** |
| | **Jones E** | **108AAQ** |

***ตารางที่ 3-2 ตัวอย่างข้อมูลที่แสดงความสัมพันธ์ของรถกับเจ้าของ***

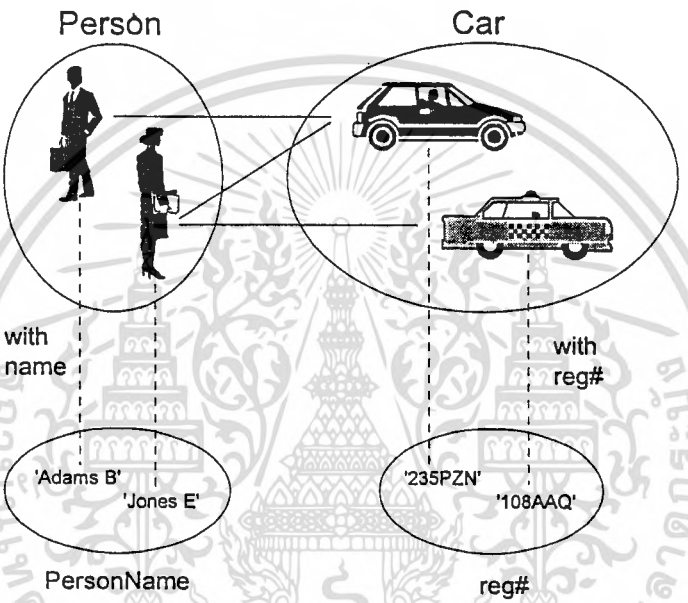
จากข้อมูลข้างต้นเราสามารถนำมาเขียนเป็น Elementary Facts ได้ดังนี้

The Person with Name 'Adams B' drives the Car with reg# '235PZN'.

The Person with Name 'Jones E' drives the Car with reg# '235PZN'.

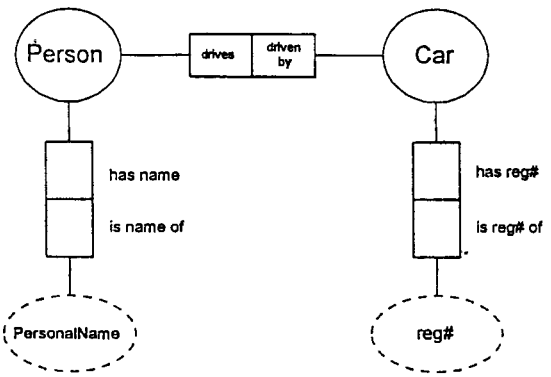
The Person with Name 'Jones E' drives the Car with reg# '108AAQ'.

ซึ่งสามารถแสดงความสัมพันธ์ให้เห็นชัดเจนดังรูปที่ 3-1 และจากความจริงพื้นฐานที่ได้ เราก็จะนำมาเขียนเป็นแผนภาพที่แสดงแนวคิด (Conceptual Schema Diagram) ได้ดังรูปที่ 3-2



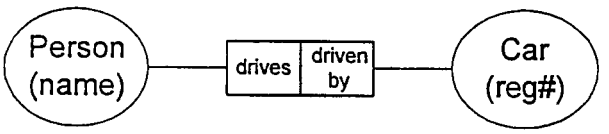

*รูปที่ 3-1 แสดงแผนภาพที่แสดงความสัมพันธ์ของเอนติตี้*

และจากแผนภาพจะเห็นได้ว่าเราได้แทนชนิดเอนติตี้โดยใช้วงรีเส้นทึบ และแทนชนิดเลเบิ้ลด้วยวงรีเส้นประ และเขียน Role ด้วยสี่เหลี่ยม โดยมีเส้นลากจากเอนติตี้ไปยัง Role เพื่อแสดงความสัมพันธ์ และเราจะเรียกความสัมพันธ์ระหว่างชนิดเอนติตี้ทั้งสองโดยผ่าน Role ทั้งสองว่า **ชนิดความจริง** (Fact Type) และเรียกความสัมพันธ์ระหว่างชนิดเอนติตี้และชนิดเลเบิ้ลว่า **ชนิดอ้างอิง** (Reference Type)




*รูปที่ 3-2 แสดงแผนภูมิระดับแนวคิด*

อย่างไรก็ตามจะสังเกตุได้ว่า สมาชิกแต่ละตัวในชนิดเอนติตี้จะมีความสัมพันธ์กับสมาชิกในชนิดอ้างอิงกันแบบ 1:1 เช่นในชนิดเอนติตี้แต่ละ Person จะมีเพียง 1 ชื่อและแต่ละชื่อก็จะหมายถึงคนหนึ่งคน ดังนั้นเราสามารถเขียนแผนภูมิในรูป 3-2 ได้ใหม่ดังในรูปที่ 3-3 โดยไส่ชนิดอ้างอิงไว้ในวงเล็บเพื่อที่จะเขียนได้ง่ายขึ้น โดยสามารถสื่อความหมายได้คงเดิม



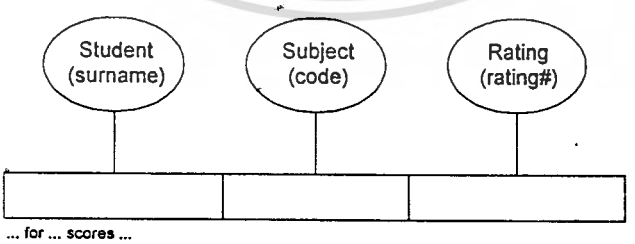

*รูปที่ 3-3 แสดงการเขียนชนิดความจริงแบบย่อ*

สำหรับวิธีการตรวจสอบความถูกต้องของแผนภูมิที่เราวาด สามารถทำได้โดยการไส่ตัวอย่างและจากที่ได้กล่าวมาทั้งหมด คงจะเห็นได้ว่าการเขียนความสัมพันธ์ในรูปแผนภูมิระดับแนวคิดนั้นสามารถมองเห็นความสัมพันธ์ได้ดีกว่า

ลองมาดูกันอีกตัวอย่างโดยสมมติข้อมูลดังในตารางที่ 3-3 ซึ่งสามารถเขียนเป็นความจริงพื้นฐานได้ดังนี้ Student (surname) 'Adams' for Subject (code) 'CS100' scores Rating (rating#) 4. ซึ่งสามารถนำมาเขียนเป็นแผนภูมิระดับแนวคิดได้ดังรูปที่ 3-4 จะเห็นได้ว่าแผนภูมิที่ได้คราวนี้จะมี Role ทั้งหมด 3 Role เพราะข้อมูลทั้ง 3 มีความสัมพันธ์กันทั้งหมด และเราจะเรียกแผนภูมิที่ประกอบด้วย 2 Role ว่า Binary Fact Type และ 3 Role ว่า Ternary Fact Type

| *Student* | *Subject* | *Rating* |
|---|---|---|
| Adams | CS100 | 4 |
| Brown | CS100 | 4 |
| Brown | CS112 | 5 |

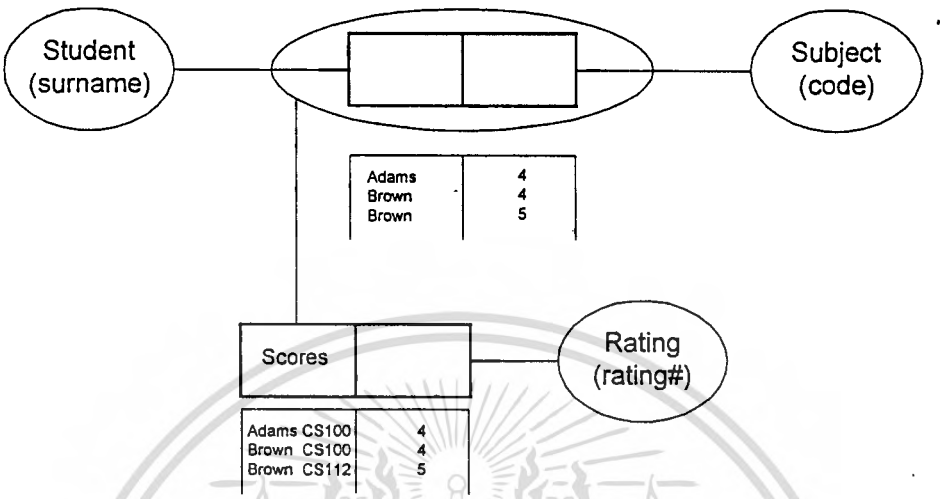
*ตารางที่ 3-3 แสดงตัวอย่างข้อมูลผลการเรียน*




*รูปที่ 3-4 แสดงชนิดความจริงแบบ Ternary*

อย่างไรก็ตามการเขียนชนิดความจริงในลักษณะเช่นนี้ เมื่อพิจารณาในแง่ของความเป็นจริงแล้วก็อาจจะไม่ถูกต้องนัก เพราะชนิดเอนติตี้ Rating ถือได้ว่ามิไช่มีความสัมพันธ์กับเพียงชื่อวิชาหรือตัวนักศึกษาอย่างใดอย่างหนึ่งเท่านั้น แต่มีความสัมพันธ์กับทั้งชนิดเอนติตี้ Student และชนิดเอนติตี้ Subject เช่น นักศึกษาที่มีชื่อ Adams และลงเรียน CS100 เท่านั้น จึงจะมีคะแนนเป็น 4 จะเห็นได้ว่าเป็นความ

สัมพันธ์ร่วมกัน ดังนั้นเราจะเขียนใหม่เพื่อให้สื่อความหมายได้ถูกต้องมากยิ่งขึ้นได้ดังรูปที่ 3-5 และเราเรียกความสัมพันธ์ในลักษณะเช่นนี้ว่า Nesting




*รูปที่ 3-5 แสดงการเขียนความสัมพันธ์แบบ Nesting พร้อมทั้งแสดงตัวอย่างข้อมูล*

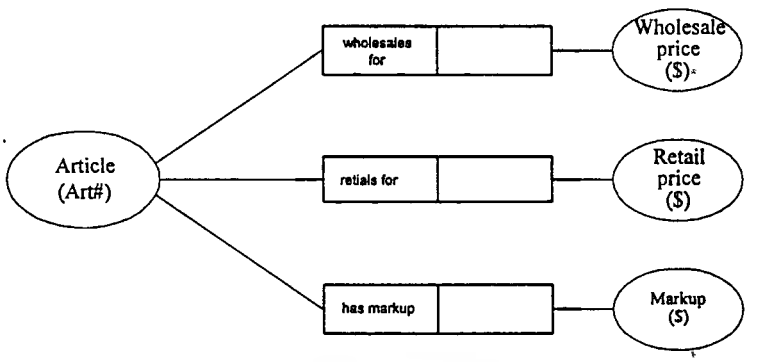
**3.2.3 ขั้นตอนที่ 3 Trim schema and find derived fact type**

ในบางครั้งการกำหนดชนิดเอนติตี้ขึ้นมาเรื่อยๆ ก็อาจจะเกิดความซ้ำซ้อนขึ้นมาได้ ดังนั้นในขั้นตอนนี้เราจะทำการตรวจสอบว่ามีชนิดเอนติตี้ใดบ้างที่มีความซ้ำซ้อนและสามารถจะลดลงได้ เราลองมาดูตัวอย่างในตารางที่ 3-4 ซึ่งเป็นรายการขายสินค้าและการทอนเงิน

| Article | Wholesale Price ($) | Retail Price ($) | Markup ($) |
|---|---|---|---|
| A1 | 50 | 75 | 25 |
| A2 | 80 | 130 | 50 |
| A3 | 50 | 70 | 20 |
| A4 | 100 | 130 | 30 |

*ตารางที่ 3-4 ตัวอย่างข้อมูลการขายสินค้าและทอนเงิน*

จากข้อมูลในตารางจะสามารถเขียนเป็นแผนภูมิระดับแนวคิดได้ดังรูปที่ 3-6 ซึ่งประกอบด้วยชนิดเอนติตี้ 4 ชนิด คือ Article, Wholesale, Retail และ Markup หากเมื่อพิจารณาให้ดีแล้วจะเห็นว่าชนิดเอนติตี้ Wholesale price, Retail price, Markup นั้นสามารถจะรวมเป็นชนิดเอนติตี้เดียวกันได้ เพราะเป็นหน่วยของเงินเหมือนๆ กัน และนอกจากนั้นลองดูข้อมูลบรรทัดแรกในตาราง จะเห็นได้ว่าเงินทอนนั้นเป็นข้อมูลที่สามารถหาได้จากเงินที่จ่ายกับราคาขายตามสูตร Markup = Retail price - Wholesale price ดังนั้นข้อมูลที่เราเก็บจริงนั้นจะมีเพียง Retail price และ Wholesale price เท่านั้น โดยที่ Markup จะเก็บอยู่ในรูปของกฎของความสัมพันธ์แทน




*รูปที่ 3-6 แสดงการเขียนโครงสร้างข้อมูลระดับแนวคิดที่ผิด*

การที่เก็บเช่นนี้จะมีข้อดีคือ สามารถป้องกันความผิดพลาดได้กล่าวคือ หากเราเก็บข้อมูลทั้งสามตัว ก็อาจเป็นไปได้ว่าค่าของเงินทอนของสินค้าตัวหนึ่ง อาจจะไม่เท่ากับค่าเงินที่จ่ายลบด้วยราคาขายก็เป็นได้ ซึ่งจะทำให้เกิดการขัดแย้งกันของข้อมูล ดังนั้นการเก็บข้อมูลในลักษณะดังกล่าวจึงสามารถแก้ปัญหานี้ได้ นอกจากนั้นยังเป็นการประหยัดเนื้อที่ในการเก็บอีกด้วย และเราจะเรียกชนิดความจริงแบบที่เก็บเป็นกฎนี้ว่า ชนิดความจริงแบบหาได้ (Derived Fact Type) และระบุในแผนภาพโดยเขียนเครื่องหมายดอกจันไว้ที่ด้านล่างของชนิดความจริงนั้น ดังนั้นจากแผนภาพในภาพที่ 3-6 เราสามารถนำมาเขียนใหม่ได้ดังภาพที่ 3-7

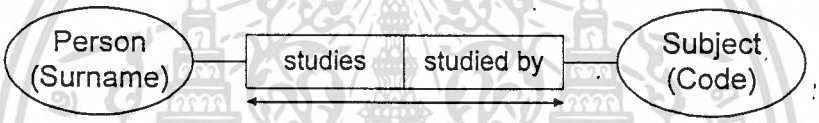

*รูปที่ 3-7 แสดงแผนภาพที่ได้รับการแก้ไขแล้ว*

**3.2.4 ขั้นตอนที่ 4 Uniqueness Constraint**

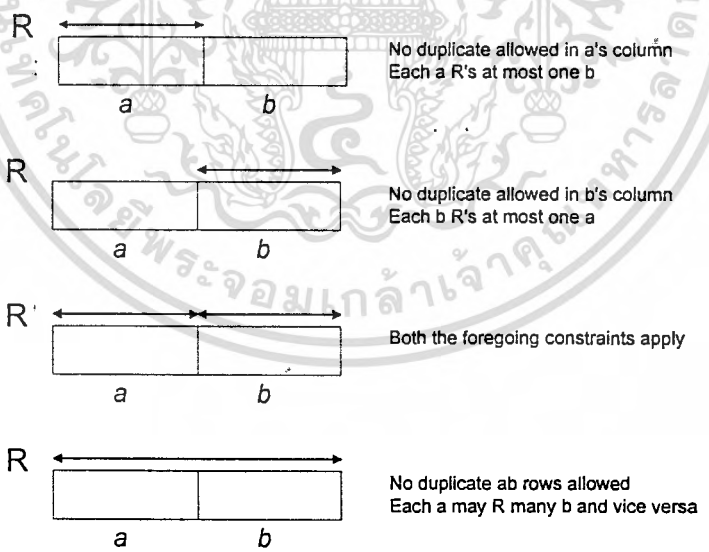
ที่ผ่านมาเราได้ทำการสร้างแผนภาพระดับแนวคิดขึ้น ทั้งในแบบที่เป็นเก็บจริง (store fact type) และแบบหาได้ (derive fact type) สำหรับในขั้นตอนที่เหลือเราจะกล่าวถึงกฎข้อบังคับ (constraint) โดยในขั้นตอนนี้เราจะกล่าวถึง Uniqueness Constriant ซึ่งเป็น Constraint ที่สำคัญมาก ซึ่งจะมีบทบาทมากในภายหลังในขั้นตอนการแปลงจากฐานข้อมูลระดับแนวคิด (Conceptaul Schema) ไปเป็นฐานข้อมูลแบบรีเลชันแนล (Relational Schema)

สำหรับกฎข้อบังคับ ชื่อก็บอกอยู่แล้วโดยมันจะทำหน้าที่ควบคุมความถูกต้องของฐานข้อมูล (static constraints apply to every possible state of database) เช่น อาจจะกำหนดว่า "นักศึกษาแต่ละคนจะสังกัดได้เพียงภาควิชาเดียวเท่านั้น" และโดยเฉพาะ Uniqueness Constraint นั้น ทุกๆ ชนิดความจริง (store) จะต้องมี Uniqueness Constraint บังคับอยู่อย่างน้อย 1 ตัว เพราะในแต่ละชนิดความจริงนั้นจะต้องไม่มีข้อมูล 2 แถวที่ซ้ำกันเกิดขึ้น

สำหรับการเขียน Uniqueness Constrint นั้นจะเขียนโดยใช้ลูกศรที่มี 2 หัว เช่น สำหรับชนิดความจริงของข้อมูลที่ว่า นักศึกษาแต่ละคนจะลงเรียนได้หลายวิชา และแต่ละวิชาก็มีนักศึกษาได้หลายคน จะสามารถเขียน ได้ดังรูปที่ 3-8 ซึ่งจะหมายถึงว่านักศึกษาหนึ่งคนจะลงเรียนในรายวิชาหนึ่งได้เพียงครั้งเดียวเท่านั้น ดังนั้นเมื่อรวมข้อมูลของทั้ง 2 คอลัมน์แล้วจะไม่ซ้ำ (Unique) แต่ในแต่ละคอลัมน์จะมีการซ้ำได้ และเรียกความสัมพันธ์ของชนิดความจริงแบบนี้ว่าแบบ many to many สำหรับความสัมพันธ์ในแบบต่างๆ ของชนิดความจริงแสดงไว้ในรูป 3-9 ซึ่งได้แก่แบบ one to many, many to one, one to one และ many to many ซึ่งจากรูปก็ได้แสดงความหมายไว้อย่างชัดเจนดีอยู่แล้ว จึงไม่กล่าวอะไรมากไปกว่านี้

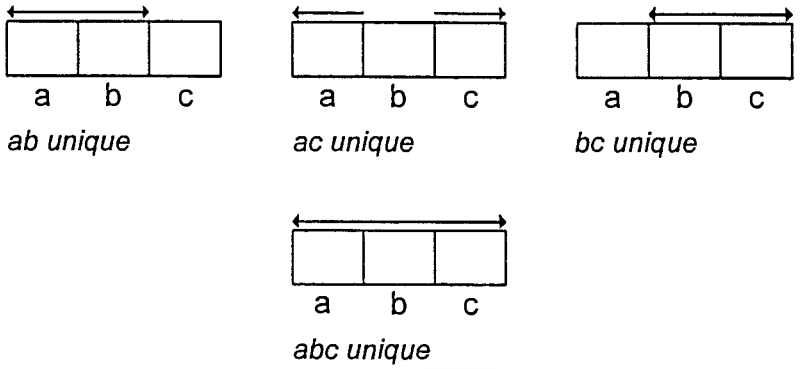



*รูปที่ 3-8 แสดง Uniqueness Constraint ที่ครอบคลุมทุก Role ในชนิดความจริง*




*รูปที่ 3-9 แสดง Uniqueness Constraint ที่เป็นไปได้ทั้งหมด 4 กรณีของชนิดความจริงแบบไบนารี*

อย่างไรก็ตามสิ่งที่กล่าวมาทั้งหมด เราได้กล่าวถึงเฉพาะกับแบบที่เป็น Binary เท่านั้น หากแต่ในความเป็นจริงแล้วยังมีชนิดความจริงที่มี role มากกว่า 2 role และในกรณีดังกล่าว เราก็ใช้หลักที่ว่าสำหรับชนิดความจริงที่มี n role ใดๆ จะสามารถมี role ที่ไม่มี Uniqueness Constriant กำกับได้เพียง 1 role เท่านั้นเช่นชนิดความจริงที่มี 3 role จะสามารถมี Uniqueness Constrint ที่เป็นไปได้ทั้งหมด 4 แบบดังรูปที่

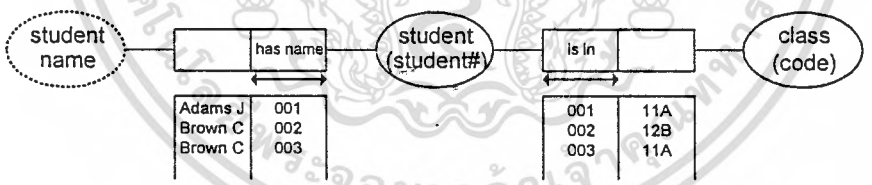



*รูปที่ 3-10 แสดง Uniqueness Constraint ที่เป็นไปได้ 4 แบบของชนิดความจริงแบบเทอนารี*

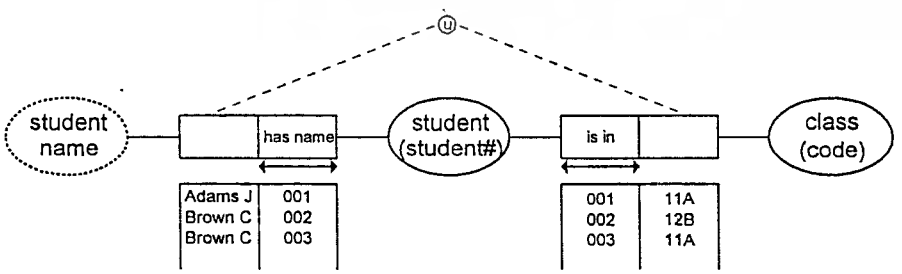
เท่าที่ผ่านมา เราได้กล่าวถึง Uniqueness Constraint ในแบบที่ Role ทุก Role ที่อยู่ภายใต้ Constraint หนึ่ง จะอยู่ในชนิดความจริงเดียวกันเท่านั้น ต่อไปเราลองมาดูข้อมูลในตารางที่ 3-5 จากตารางจะเห็นได้ว่านักศึกษาแต่ละคนจะมีชื่อเพียงชื่อเดียว และนักศึกษาแต่ละคนจะมีชั้นเรียนเพียงชั้นเรียนเดียว จากข้อมูลดังกล่าวจะสามารถเขียนแผนภาพระดับแนวคิดได้ดังรูปที่ 3-11

| *Student :* | *Student#* | *Name* | *Class* |
|---|---|---|---|
| | 001 | Adams J | 11A |
| | 002 | Brown C | 12B |
| | 003 | Brown C | 11A |

*ตารางที่ 3-5 แสดงรายงานห้องเรียนของนักศึกษา*




*รูปที่ 3-11 แสดงแบบจำลองข้อมูลสำหรับตารางที่ 3-5*




*รูปที่ 3-12 แสดงการใช้ Uniqueness Constraint แบบ Inter-fact-type*

อย่างไรก็ตามหากสังเกตุให้ดีจากตาราง จะเห็นได้ว่าผลรวมของชื่อและชั้นเรียนก็จะเป็นข้อมูลที่ไม่ซ้ำเช่นกัน และในกรณีเช่นนี้ NIAM ก็อนุญาติให้สามารถสร้าง Uniqueness Constraint ระหว่างชนิดความจริง 2 ชนิดความจริงได้ ดังรูปที่ 3-12

จากรูปจะเห็นได้ว่าเราจะใช้เครื่องหมาย "u" ล้อมด้วยวงกลมและมีเส้นโยงไปยัง Role ที่อยู่ภายใต้ Uniqueness Constraint นั้น การสร้าง Constraint ระหว่างชนิดความจริงนี้เราเรียกว่า Inter-fact-type และเรียก Constraint ที่อยู่ในชนิดความจริงเดียวกันนี้ว่า Intra-fact-type จากที่ผ่านมาทั้งหมดนี้ เราจะเห็นได้ว่าการตรวจสอบความถูกต้องโดยใช้ตัวอย่างใน NIAM นั้นมีความสำคัญมาก ทั้งนี้เพราะจะทำให้ผู้ออกแบบสังเกตุ ข้อมูลบางอย่างเพิ่มเติมได้จากข้อมูลที่ให้มานี้

### 3.2.5 ขั้นตอนที่ 5 Arity Checks

หากในขั้นตอนที่ผ่านมานั้น เราได้ทำการออกแบบอย่างระมัดระวังแล้ว การทำงานในขั้นตอนนี้นับว่าเป็นสิ่งที่ไม่จำเป็นเลย อย่างไรก็ตาม จากความจริงที่ว่า นักปราชญ์ยังรู้พลั้ง นั้นเป็นข้อเตือนใจได้เป็นอย่างดี เพราะแม้ว่าเราจะเป็นผู้ออกแบบที่ชำนาญเพียงใดก็ตาม ก็ยังอาจเกิดการผิดพลาดขึ้นได้

ความผิดพลาดที่เกิดขึ้นนั้นมีอยู่ 2 ประการคือการออกแบบชนิดความจริงที่สั้นหรือยาวเกินไป ในกรณีของชนิดความจริงที่ยาวเกินไปนั้น ก็จะต้องทำการแยกชนิดความจริงนั้นออกเป็น 2 ชนิดความจริง และสำหรับชนิดความจริงที่สั้นเกินไปนั้น ไม่สามารถที่จะบรรจุข้อมูลได้ครบ (information Loss) ดังนั้นก็จะต้องทำการรวม 2 ชนิดความจริงเป็นชนิดความจริงเดียว

สำหรับการตรวจสอบความถูกต้องนั้น จริงๆ แล้วอาจจะมีวิธีการมากมาย สุดแต่ใครจะใช้วิธีการอย่างไร หากแต่ในที่นี้เราจะกล่าวถึงเพียง 3 วิธี

3.2.5.1 ใช้สามัญสำนึก หรือความรู้พื้นฐานในการพิจารณาตัดสินว่ามีข้อมูลสูญหายเพราะการแยกชนิดความจริงออกมาหรือไม่

3.2.5.2 ใช้กฎของการแยกจากกัน (Splittability Rules) ซึ่งจะกล่าวถึงภายหลัง

3.2.5.3 ใช้วิธีสร้างตัวอย่างขึ้นมาสำหรับชนิดความจริง และทำการแยกชนิดความจริงนั้นออกโดยวิธี Projection (ซึ่งจะกล่าวถึงในภายหลัง) จากนั้นนำมารวมกันใหม่โดยใช้วิธี Natural Join และหากมีตัวอย่างเพิ่มขึ้นมาจากการ Join ก็แสดงว่าชนิดความจริงนั้นเป็นชนิดความจริงที่ไม่สามารถทำการแยกได้

| *Person* | *Degree* | *Subject* | *Rating* |
|---|---|---|---|
| Adams | BSc | CS112 | 7 |
| Adams | Bsc | CS110 | 6 |
| Adams | Bsc | PD102 | 7 |
| Brown | BA | CS112 | 6 |
| Brown | BA | PD102 | 7 |
| Collins | Bsc | CS112 | 7 |

***ตารางที่ 3-6 ผลการเรียนของนักศึกษาในวิชาเรียนวิชาหนึ่ง***

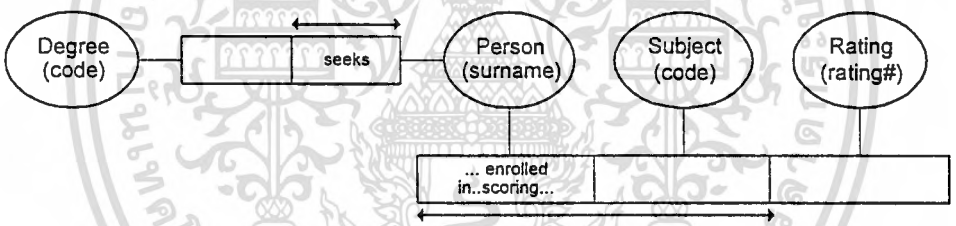
สำหรับวิธีการแรกนั้น อาจจะเป็นวิธีที่ดูไม่ค่อยจะมีหลักการสักเท่าใดนัก แต่จริงๆ แล้วนับได้ว่าเป็นวิธีที่ดีวิธีหนึ่ง เพราะเป็นวิธีที่มีผลสอดคล้องกับขั้นตอนการกำหนด Elementary Facts เพื่อให้การทำงานในวิธีนี้ชัดเจนยิ่งขึ้น เราลองมาดูตัวอย่างต่อไปนี้ เราจะเริ่มจากข้อมูลในตารางที่ 3-6 และจากวิธีการที่ผ่านมา เราสามารถสร้าง Elememtary Facts ได้ดังนี้

Person(surname) 'Adams' seeks Degree(code) 'BSc'.

Person(surname) 'Adams' enrolled in Subject(code) 'CS112'

scoring Rating(Nr) 7.

ในที่นี้คำว่า Seeks หมายถึงสำเร็จตามหลักสูตร และจาก Elementary Facts ดังกล่าว เราสามารถนำมาเขียนเป็นแผนภาพระดับแนวคิดได้ดังรูปที่ 3-13 ทีนี้เราก็จะมาลองดูว่าหากเราต้องการจะแยกชนิดความจริงแบบ Ternary ในภาพออกเป็น 2 ชนิดความจริงจะได้หรือไม่ เราก็จะได้ข้อมูลว่า Adams ได้คะแนนเท่ากับ 7 และ Adams ลงเรียนในวิชา CS112 ซึ่งจะเห็นได้อย่างชัดเจนว่ามีข้อมูลบางส่วนหายไป (Information Loss) นั่นคือข้อมูลที่ว่า Adams ได้คะแนนเท่ากับ 7 ซึ่งไม่ทราบว่าได้คะแนนในวิชาใด ดังนั้นสรุปได้ว่าชนิดความจริงนี้ไม่สามารถจะแยกได้




*รูปที่ 3-13 แสดงแบบจำลองข้อมูลที่ได้จากตารางที่ 3-6*

3.2.5.4 การใช้กฎการแยก (Splittability Rule)

จริงๆ แล้วเราได้กล่าวถึงกฎที่ว่านี้ไปครั้งหนึ่งแล้ว ในขั้นตอนที่ผ่านมา นั่นคือกฎที่ว่าจะต้องมี Role ที่ไม่อยู่ภายใต้ Uniqueness Constraint ได้ไม่เกิน 1 เท่านั้น

3.2.5.5 การตรวจสอบโดยใช้ Projection-Join

ในตอนที่แล้ว เราได้กล่าวถึงเงื่อนไขของการแยกชนิดความจริงโดยใช้ Uniqueness Constraint อย่างไรก็ตามในบางครั้งเงื่อนไขดังกล่าวก็ไม่สามารถใช้งานได้ และนอกจากนั้น ผู้ออกแบบบางคนก็อาจจะต้องการวิธีการที่แน่นอนในการตรวจสอบ ซึ่งเราจะได้กล่าวต่อไป

ต่อไปเราก็จะกล่าวถึงหลักการหรือวิธีการที่แน่นอนที่จะใช้ในการแก้ปัญหานี้ได้ ในวิธีนี้เราจะใช้การทำงาน 2 อย่างคือ Projection และ Joining โดยที่ Projection ก็คือการสร้างตารางใหม่ขึ้นมาจากตารางเดิม 1 ตารางโดยเลือกคอลัมน์มาเพียงบางคอลัมน์ ซึ่งในตารางใหม่นี้ บางครั้งก็จะเกิดการซ้ำกันขึ้น ดังนั้นเราก็จะต้องตัดข้อมูลในบางแถวที่มีการซ้ำกันออกไป สำหรับการ Join ก็คือการรวมตาราง 2 ตารางขึ้นมาเป็นตารางใหม่

| *Person* | *Subject* | *Rating* |
|---|---|---|
| Adams | CS112 | 7 |
| Adams | CS110 | 6 |
| Adams | PD102 | 7 |
| Brown | CS112 | 6 |
| Brown | PD102 | 7 |
| Collins | CS112 | 7 |

*ตารางที่ 3-7 แสดงข้อมูลก่อนการทำ Projection*

ทีนี้เราลองมาดูข้อมูลในตารางที่ 3-7 จากรูปจะเห็นได้ว่าเราทำการ Projection ตารางออกเป็น 2 ตารางดังตารางที่ 3-8 และเมื่อเรานำ 2 ตารางนี้มา Join กันใหม่ก็จะได้ตารางที่ 3-9 ซึ่งจะเห็นได้ว่ามีแถวของข้อมูลเกิดขึ้นใหม่ (แถวที่มีเครื่องหมาย X) ซึ่งข้อมูลเหล่านี้เป็นข้อมูลที่ไม่ได้มีอยู่ในตารางที่เราทำการ Projection มาดังนั้นจะสรุปได้ว่าเราไม่สามารถแยกชนิดความจริงนี้ได้

| *Person* | *Subjects* | *Person* | *Rating* |
|---|---|---|---|
| Adams | CS112 | Adams | 7 |
| Adams | CS110 | Adams | 6 |
| Adams | PD102 | Brown | 6 |
| Brown | CS112 | Brown | 7 |
| Brown | PD102 | Collins | 7 |
| Collins | CS112 | | |

*ตารางที่ 3-8 แสดงข้อมูลที่ผ่านการทำ Projection ออกเป็น 2 ตารางแล้ว*

| | *Person* | *Subject* | *Rating* |
|---|---|---|---|
| *X* | *Adams* | *CS112* | *7* |
| *X* | *Adams* | *CS112* | *6* |
| | *Adams* | *CS110* | *7* |
| | *Adams* | *CS110* | *6* |
| | *Adams* | *PD102* | *7* |
| *X* | *Adams* | *PD102* | *6* |
| | *Brown* | *CS112* | *6* |
| *X* | *Brown* | *CS112* | *7* |
| *X* | *Brown* | *PD102* | *6* |
| | *Brown* | *PD102* | *7* |
| | *Collins* | *CS112* | *7* |

*ตารางที่ 3-9 ตารางที่ได้จากการ Join ตารางเข้าด้วยกัน*

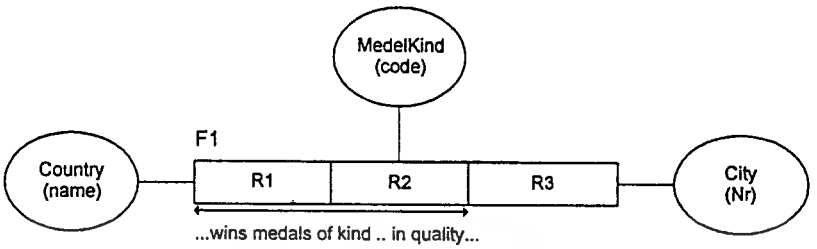
### 3.2.6 ขั้นตอนที่ 6 More Constraints

ในขั้นตอนที่ผ่านมา เราได้กล่าวถึง Constraint ไปเพียง 1 ชนิดเท่านั้นในขั้นตอนนี้เราจะกล่าวถึง Constraint อื่นๆ เพิ่มเติม โดย Constraint ที่จะกล่าวเพิ่มเติมมีดังนี้

3.2.6.1 Entity Type Constraint

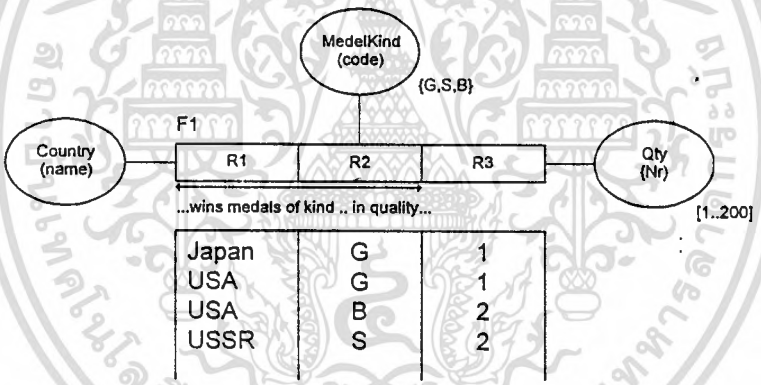
จากรูปที่ 3-14 ชนิดเอนติตี้ Country เป็นเซ็ทของประเทศต่างๆ ที่เป็นไปได้ ในกรณีนี้หากเราต้องการที่จะหาว่ามีสมาชิกใดบ้างที่สามารถอยู่ในเอนติตี้นี้ได้ อาจจะเป็นงานที่ยากเพราะมีการตั้งประเทศ

ขึ้นมาใหม่ๆ อยู่เรื่อยๆ อย่างไรก็ตามสำหรับในบางชนิดเอนติตี้นั้นง่ายมาที่จะบอกถึงสมาชิกของเอนติตี้นั้น เช่น เอนติตี้ของเหรียญที่ใช้ในกีฬาโอลิมปิกก็จะมีเหรียญ ทอง เงิน และบรอนซ์ ซึ่งข้อจำกัดของความเป็นไปได้ในชนิดเอนติตี้นี้เอง ที่เรียกว่า Entity Type Constraint




*รูป 3-14 ชนิดความจริงของผลกีฬาโอลิมปิก*

ซึ่งในกรณีของเอนติตี้เหรียญนี้สามารถเขียนได้ดังในรูปที่ 3-15 จะเห็นได้ว่า เราจะใช้วิธีเขียนเซ็ทของเอนติตี้ไว้ที่ด้านข้างของเอนติตี้ โดยที่ในกรณีที่มีสมาชิกจำกัดก็จะใช้เครื่องหมาย เช่น G,S,B และหากเป็นช่วงเราจะใช้ [ ] เช่น [1..200]




*รูปที่ 3-15 แสดงชนิดความจริงที่มีคอนสแตนต์กำกับในเอนติตี้ MedalKind และ Qty*

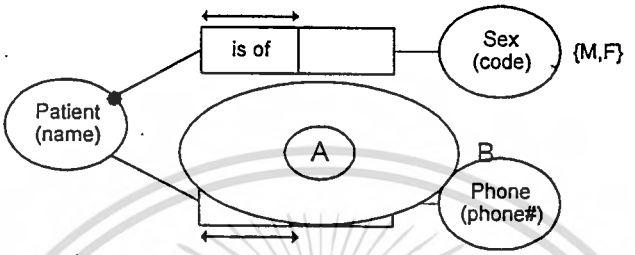
3.2.6.2 Mendatory and optional roles

เพื่อที่จะอธิบายว่าทำไมเราจะต้องมี Constraint นี้ด้วย เราจะพิจารณาจากตัวอย่างข้อมูลในตารางที่ 3-9 จะเห็นได้ว่ามีข้อมูลอยู่ช่องหนึ่งที่เราใส่เครื่องหมาย "?" ไว้เพื่อแสดงว่ายังไม่มีการบันทึกข้อมูลในตารางนี้ เช่น Brown S ที่จริงแล้วอาจจะมีโทรศัพท์ก็เป็นได้ แต่ยังไม่ได้รับการบันทึก

อย่างไรก็ตามหากข้อมูลนั้นเป็นข้อมูลที่สำคัญเช่นข้อมูลของเพศ การที่เราจะไม่บันทึกข้อมูลนั้นลงไปก็อาจจะนับได้ว่าเป็นการไม่ถูกต้องนัก ดังนั้นสำหรับในข้อมูลบางตัวที่ "ต้องบันทึก" นั้น เราจะเรียก Role นั้นว่า Mendatory และใช้เครื่องหมาย "จุด" เพื่อระบุ Mendatory Constraint ดังแสดงในรูปที่ 3-16 ซึ่งมีความหมายว่า Patient จะต้องมีเพศ

3.2.6.3 Subtype

สำหรับเรื่องของ Subtype นี้เราจะใช้แผนภูมิของออยเลอร์ (Euler's Diagram) ในการแสดงความสัมพันธ์ระหว่างเอนติตี้ จากที่เราได้กล่าวมาแล้วว่าเอนติตี้เป็นเซ็ท ทีนี้ลองสมมติว่าเรามีเอนติตี้ Manager ซึ่งเป็นเซ็ทของผู้จัดการทั้งหมด และเอนติตี้ Employee ซึ่งเป็นเซ็ทของพนักงานที่ทำงานอยู่ และจากความจริงที่ว่าผู้จัดการทุกคนย่อมเป็นพนักงานด้วยเช่นกัน และลักษณะความสัมพันธ์ในลักษณะเช่นนี้เราจะเรียกว่า Subtype ซึ่งจะเขียนได้ดังรูปที่ 3-17



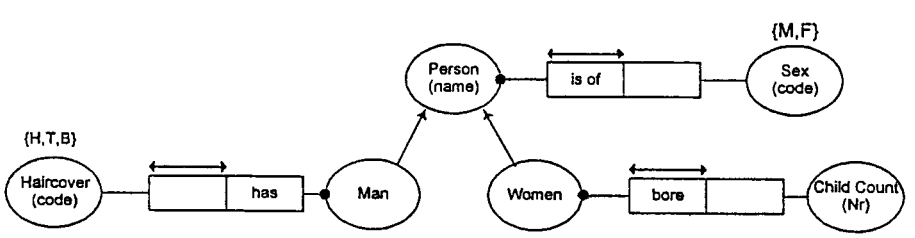

*รูปที่ 3-16 แสดงการใช้ Mendatory Constraints ที่ถูกต้อง*

*รูปที่ 3-17 แผนภูมิออยเลอร์ที่แสดงว่า A เป็นซับเซ็ตของ B*

จากรูปหมายถึงว่าเอนติตี้ B เป็น Subset ของเอนติตี้ A และเรียก A ว่า Supertype เรียก B ว่า Subtype นั่นย่อมหมายถึงว่าสมาชิกทุกตัวใน B ก็ย่อมจะมีคุณสมบัติของเอนติตี้ A ด้วย เราลองมาดูตัวอย่างกันอีกตัวอย่างหนึ่งโดยอาศัยตารางที่ 3-10 จากตารางจะเห็นได้ว่าเฉพาะเพศชายเท่านั้นจึงจะมีการบันทึก Haircover และเฉพาะเพศหญิงเช่นกันที่มีการบันทึกจำนวนบุตร ในลักษณะดังกล่าวนี้หากเราใช้วิธีการเขียนในแบบที่ผ่านมาก็ย่อมได้ และไม่ผิดหากแต่ไม่สามารถแสดงความหมายได้ครบ เพราะ NIAM นั้นเป็นแบบจำลองข้อมูลระดับแนวคิดดังนั้นวิธีการที่ดีกว่าคือการใช้ Subtype นั่นเองซึ่งจะเขียนเป็นแผนภาพได้ดังรูปที่ 3-18

| *Person* | *Sex* | *Haircover* | *NrChildren* |
|---|---|---|---|
| Jones E | F | - | 2 |
| Smith J | M | T | - |
| Blow J | M | B | - |
| Lane L | F | - | 0 |
| Blossom B | F | - | 5 |

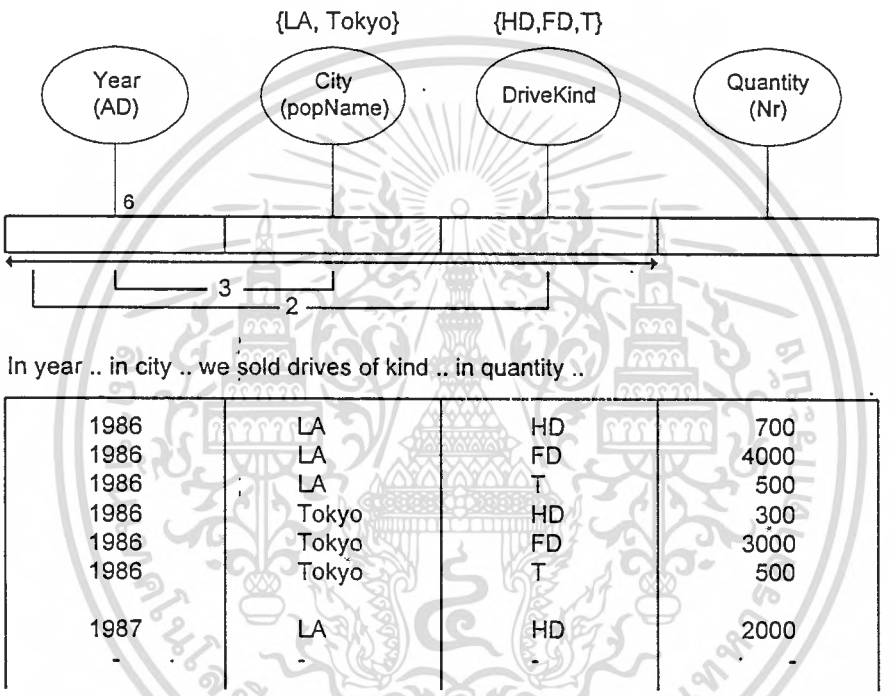
***ตารางที่ 3-10 แสดงข้อมูลตัวอย่างที่แสดงการใช้ Subtype***




***รูปที่ 3-18 แสดงแบบจำลองข้อมูลระดับแนวคิดที่นำเอา Subtype มาใช้งาน***

3.2.6.4 Occurence frequencies

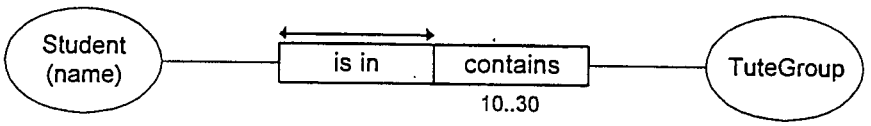
ในตอนนี้เราจะกล่าวถึง Constraint อีกตัวหนึ่งที่จะใช้ในการระบุจำนวนครั้งที่ชนิดเอนติตี้จะเก็บได้สำหรับ Role หนึ่งๆ Constraint นี้มีชื่อเรียกว่า Occurrence frequency constraint ในรูปที่ 3-19 เป็นตัวอย่างของการใช้งาน Occurrence frequency constraint โดยจะเป็นข้อมูลของการขายไดรฟ์ประเภทต่าง ๆ ใน 3 ประเภท คือ HD (Hard Drive), FD (Floppy Drive), T (Tape Drive) ในเมือง LA และ Tokyo ในปีต่าง ๆ ซึ่งจะสังเกตุได้ว่าในแต่ละปีจะต้องมีการบันทึกจำนวน 6 แถวเสมอ โดยเป็นข้อมูลของเมือง LA จำนวน 3 แถว และเมือง Tokyo จำนวน 3 แถว




*รูปที่ 3-19 แสดงตัวอย่างของแบบจำลองข้อมูลที่ใช้ Occurrence Frequenct Constraint*

จากรูปจะเห็นว่ามีการเขียนตัวเลขกำกับไว้ 3 ตัวด้วยกัน โดยเลข 6 จะมีความหมายว่าในแต่ละข้อมูลของเอนติตี้ Year จะต้องมีการบันทึกข้อมูล 6 ครั้งเสมอ และในแต่ละข้อมูลของ Year และ City จะต้องมีการบันทึกข้อมูล 3 ครั้งเสมอ และในแต่ละข้อมูลของ Year และ DriveKind จะต้องมีการบันทึกข้อมูล 2 ครั้งเสมอ การกำกับคอนสแตนต์ไว้เช่นนี้จะเป็นการช่วยกำกับความถูกต้องของการบันทึกข้อมูลที่มีจำนวนครั้งในการบันทึกที่แน่นอนได้

และนอกจากจะระบุจำนวนครั้งการบันทึกข้อมูลที่แน่นอนแล้ว Occurrence Frequency Constraint ยังอนุญาตให้มีการบันทึกเป็นช่วงอีกด้วย ดังตัวอย่างในรูปที่ 3-20 ซึ่งเป็นตัวอย่างของการบันทึกข้อมูลนักเรียนที่อยู่ในกลุ่มกวดวิชาต่าง ๆ โดยสามารถระบุว่าจำนวนนักเรียนที่อยู่ในแต่ละกลุ่มจะต้องมีจำนวนอยู่ระหว่าง 10-30 คน เป็นต้น

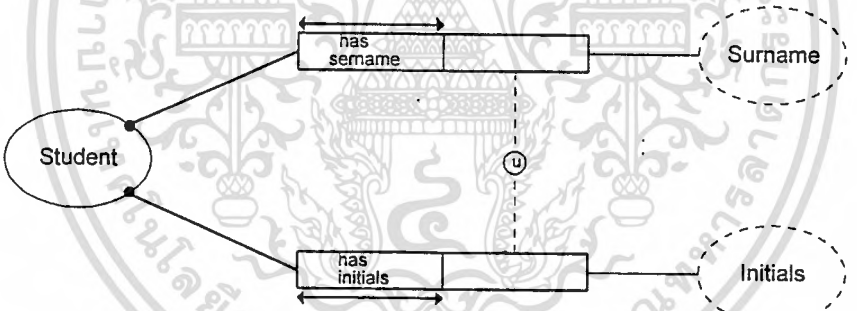



*รูปที่ 3-20 แสดงการใช้ Occurrence Frequency Constraint อีกรูปแบบหนึ่ง*

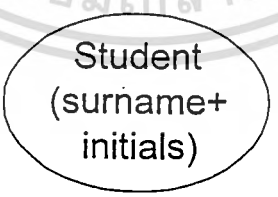
### 3.2.7 ขั้นตอนที่ 7 Emtity Identification Schemes

แบบจำลองข้อมูลไนแอมที่ได้ใช้เป็นตัวอย่างที่ผ่านมาทั้งหมด จะเห็นได้ว่าเป็นแบบจำลองข้อมูลที่มีความสัมพันธ์ระหว่างชนิดเอนติตี้และชนิดเลเบิ้ลเป็นแบบ 1:1 เท่านั้น ทั้งนี้เนื่องจากต้องการให้ง่ายต่อการอธิบายนั่นเอง แต่ในโลกแห่งความเป็นจริงแล้ว การจะอ้างถึงสิ่งใดสิ่งหนึ่งที่อยู่ในชนิดเอนติตี้ใด ๆ บ้างครั้งอาจไม่สามารถอ้างได้โดยใช้ชนิดเลเบิ้ลเดียวได้ เช่น การอ้างชื่อคนก็จะต้องอ้างโดยอาศัยชื่อและนามสกุล เพราะบางคนอาจจะมีชื่อเดียวกันก็ได้ ดังนั้นจึงต้องมีนามสกุลกำกับไว้ด้วย

ดังนั้นความสัมพันธ์ระหว่างชนิดเอนติตี้และชนิดเลเบิ้ลก็ไม่จำเป็นต้องอยู่ในรูปแบบ 1:1 เสมอไป โดยอาจจะอยู่ในรูปแบบใดก็ได้ แต่ทั้งนี้จะต้องมีชนิดเลเบิ้ลกลุ่มหนึ่งที่สามารถระบุถึง (Identify) แต่ละตัวอย่างในชนิดเอนติตี้ได้เสมอ ในรูปที่ 3-21 เป็นตัวอย่างของการใช้ 2 ชนิดเลเบิ้ลในการระบุเอนติตี้ ซึ่งสามารถเขียนแบบย่อได้ในรูปที่ 3-22




*รูปที่ 3-21 การใช้ชนิดเลเบิ้ล 2 ตัวในการระบุถึงแต่ละเอนติตี้ในชนิดเอนติตี้ Student*

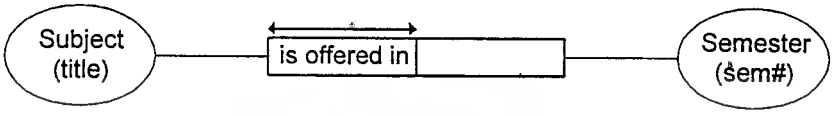



*รูปที่ 3-22 การเขียนความสัมพันธ์แบบย่อระหว่างชนิดเอนติตี้กับชนิดเลเบิ้ลในรูปที่ 3-21*

อย่างไรก็ตามการกำหนดว่าจะให้ชนิดเลเบิ้ลใดเป็นตัวระบุถึงแต่ละเอนติตี้นั้น ก็จะขึ้นอยู่กับขอบเขตของข้อมูลด้วย แบบจำลองข้อมูลที่สามารถใช้กับขอบเขตข้อมูลหนึ่ง ๆ อาจจะใช้ไม่ได้ผลกับขอบเขตข้อมูลที่กว้างกว่าก็ได้ สมมติว่าเรามีข้อมูลที่ว่าในแต่ละภาควิชาในมหาวิทยาลัยจะมีวิชาสอนประจำภาควิชานั้นอยู่ ดังข้อมูลตัวอย่างในตารางที่ 3-11 ซึ่งจากข้อมูลในตารางเราสามารถจะใช้ชื่อวิชาในการระบุถึงแต่ละวิชานั้น และสามารถเขียนแบบจำลองข้อมูลได้ดังรูปที่ 3-23

| *Physics:* | *Subject* | *Semester* |
|---|---|---|
| | **Electronics** | **1** |
| | **Mechanics** | **1** |
| | **Optics** | **2** |

*ตารางที่ 3-11 ตัวอย่างข้อมูลการเรียนในภาควิชา Physics*

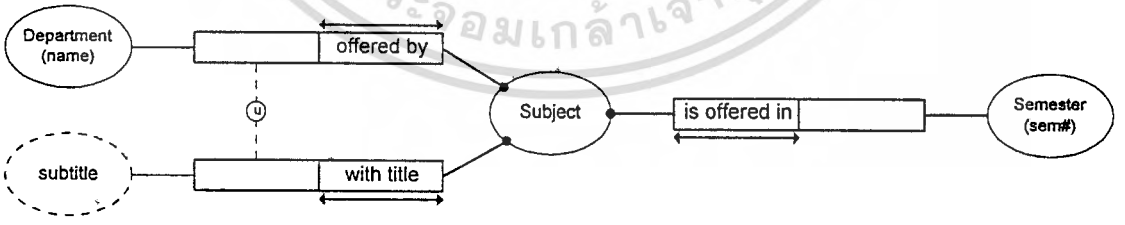



*รูปที่ 3-23 แสดงแบบจำลองข้อมูลที่ได้จากตารางที่ 3-11*

แต่ถ้าขอบเขตของข้อมูลเปลี่ยนไป โดยแทนที่จะมีเฉพาะวิชาเรียนในภาควิชาฟิสิกส์ภาควิชาเดียว ก็มีข้อมูลของภาควิชาคณิตศาสตร์ด้วย ดังตารางที่ 3-12 ซึ่งจะเห็นได้ว่าทั้ง 2 ภาควิชาล้วนแต่มีการสอนวิชา Machanics ด้วยกันทั้งคู่ ซึ่งวิชาทั้งสองนี้แม้จะมีชื่อเดียวกัน แต่ก็ไม่ได้หมายความว่าจะมีเนื้อหาการสอนที่เหมือนกัน ดังนั้นการอ้างถึงแต่ละวิชาโดยใช้ชื่อวิชาจึงไม่สามารถใช้ได้อีกต่อไป ซึ่งหมายความว่าแบบจำลองข้อมูลในรูปที่ 3-23 จะใช้ไม่ได้เมื่อนำมาใช้กับขอบเขตข้อมูลที่กว้างกว่า ดังนั้นเราจึงต้องสร้างแบบจำลองข้อมูลใหม่ภายใต้ขอบเขตข้อมูลใหม่ ซึ่งก็จะได้เป็นแบบจำลองข้อมูลในรูปที่ 3-24

| *Maths* | *Subject* | *Semester* |
|---|---|---|
| | **Algebra** | **2** |
| | **Calculus** | **1** |
| | **Mechanics** | **1** |

*ตารางที่ 3-12 ข้อมูลของภาควิชาคณิตศาสตร์*



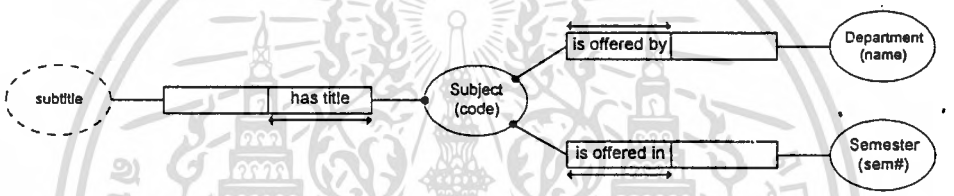

*รูปที่ 3-24 แสดงแบบจำลองข้อมูลที่ออกแบบขึ้นมาใหม่*

แบบจำลองข้อมูลในรูปที่ 3-24 นั้น แม้ว่าจะสามารถแทนข้อมูลในตัวอย่างข้อมูลได้ทั้งหมด และยังสามารถอ้างถึงวิชาเรียนโดยการอ้างภาควิชาที่สอนควบคู่ไปกับการอ้างชื่อวิชาได้ แต่ก็นับว่าเป็นวิธีการที่ไม่สะดวกมากนัก เพราะทุก ๆ ครั้งที่จะอ้างวิชาจะต้องอ้างว่า "วิชาแมคคานิกส์ที่สอนโดยภาควิชาคณิตศาสตร์" จึงจะสามารถถึงวิชาได้อย่างสมบูรณ์ ซึ่งเป็นวิธีการที่ยืดยาดมากไปสักหน่อย ซึ่งในกรณีเช่นนี้ขอแนะนำให้แก้ไขแบบจำลองข้อมูลให้มีรูปแบบที่ง่ายขึ้น โดยการเพิ่มข้อมูลเข้าไปอีกตัวหนึ่ง ซึ่งในที่นี้ก็คือรหัสวิชานั่นเอง โดยมีข้อแม้ว่ารหัสวิชาจะต้องสามารถอ้างถึงทุก ๆ วิชาในชนิดเอนติตี้ได้

ในตารางที่ 3-13 จะเป็นข้อมูลของวิชาทั้งหมดที่ได้เพิ่มรหัสวิชาเข้าไปแล้ว ซึ่งสามารถนำมาเขียนเป็นแบบจำลองข้อมูลได้ดังรูปที่ 3-25 ซึ่งขอให้สังเกตุว่าความสัมพันธ์ของการอ้างวิชาเรียนได้กลับมาอยู่ในรูปแบบ 1:1 อีกครั้ง เพราะในการอ้างชื่อวิชาเราสามารถอ้างแค่ "PH102" ก็สามารถระบุวิชาเรียนได้แล้ว

| *Subject* | *Title* | *Department* | *Semester* |
|---|---|---|---|
| PH101 | Electronics | Physics | 1 |
| PH102 | Machanics | Physics | 1 |
| PH200 | Optics | Physics | 2 |
| MP104 | Algebra | Mathematics | 2 |
| MP210 | Calulus | Mathematics | 1 |
| MA109 | Machanics | Mathematics | 1 |

*ตารางที่ 3-13 แสดงข้อมูลของวิชาทั้งหมดหลังจากที่กำหนดรหัสวิชาแล้ว*

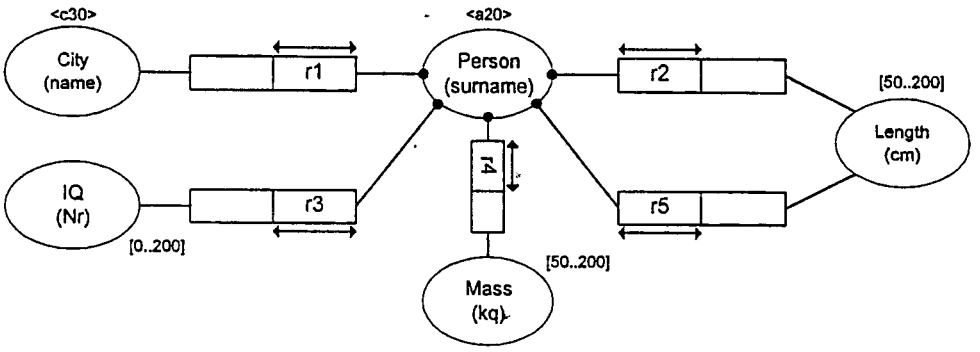



*รูปที่ 3-25 แสดงแบบจำลองข้อมูลที่ได้จากตารางที่ 3-13*

จากแบบจำลองข้อมูลจะเห็นได้ว่าชนิดเอนติตี้ Subject มีวิธีการที่สามารถระบุ (Identify) ถึงแต่ละวิชาได้ 2 วิธีด้วยกัน คือ โดยการใช้รหัสวิชา และโดยการใช้ภาควิชาร่วมกับชื่อวิชาโดยจะเรียกรวมกันว่า Subtitle ซึ่งในกรณีที่ชนิดเอนติตี้ใด ๆ ที่มีการอ้างถึงได้มากกว่า 1 แบบ ก็ให้ระบุถึงแบบอ้างอิงหลัก (Primary Identifier) เอาไว้ด้วย ซึ่งจากรูปที่ 3-25 แบบอ้างอิงหลักก็คือ รหัสวิชานั่นเอง

และก่อนจะจบการทำงานในขั้นตอนที่ 7 ก็จะขอกล่าวถึงรายละเอียดบางประการของชนิดอ้างอิง เนื่องจากชนิดอ้างอิงนั้นเป็นตัวที่ระบุถึงรูปแบบการเก็บข้อมูลของแต่ละชนิดเอนติตี้ ดังนั้นหากจะต้องการให้ชนิดอ้างอิงทำหน้าที่ได้อย่างสมบูรณ์ก็สมควรจะต้องมีการระบุถึงรูปแบบข้อมูลที่จะเก็บด้วย เช่น เป็นตัวเลขหรือตัวอักษร หากเป็นตัวเลขให้ยาวได้กี่หลัก และหากเป็นตัวอักษรจะยาวได้กี่ตัวอักษร เป็นต้น โดยหากต้องการเป็นตัวอักษรไม่เกิน 20 ตัวอาจเขียนกำกับว่า <c20> หรือหากเป็นตัวเลขระหว่าง 0-100 อาจเขียนกำกับว่า [0..100] เป็นต้น

การกำหนดรูปแบบการเก็บข้อมูลตามที่ได้กล่าวไปนั้น นอกจากจะเป็นประโยชน์ในขั้นตอนที่มีการนำเอาแบบจำลองข้อมูลไนแอมไปใช้เก็บข้อมูลจริงหลังจากที่ออกแบบเสร็จแล้ว ก็ยังมีประโยชน์ในการตรวจสอบความถูกต้องของข้อมูลที่จะจัดเก็บในแต่ละชนิดเอนติตี้อีกด้วย เช่น ชนิดเอนติตี้ที่เก็บเฉพาะตัวเลข 0-100 ก็ไม่ควรจะเก็บข้อมูลที่เป็นตัวอักษรหรือตัวเลขที่ไม่ได้อยู่ในช่วงดังกล่าว เป็นต้น ดังนั้นอาจจะถือได้ว่าการกำหนดแบบนี้อาจถือได้ว่าเป็นคอนสแตนต์อย่างหนึ่ง โดยมีชื่อเรียกว่า Lexical Constraint ซึ่งได้แสดงไว้ในรูปที่ 3-26 แล้ว




*รูปที่ 3-26 ตัวอย่างการใช้งาน Lexical Constraint*

3.2.8 ขั้นตอนที่ 8 Further Constraints

ที่ผ่านมาเราได้กล่าวถึงคอนสแตนต์ที่สำคัญและจำเป็นต่อการใช้งานไปหลายคอนสแตนต์แล้ว แต่ก็ยังมีคอนสแตนต์ที่ยังไม่กล่าวถึงอีกหลายคอนสแตนต์เช่นกัน ดังนั้นในขั้นตอนสุดท้ายก่อนที่จะเข้าสู่ขั้นตอนการสรุปและตรวจสอบ ก็จะขอกล่าวถึงคอนสแตนต์ที่เหลือทั้งหมดรวมกันไปเลย ได้แก่ Equality Constraint, Exclusion Constraint และ Subset Constraint

ก่อนจะกล่าวถึงคอนสแตนต์เราก็ต้องกำหนดตัวอย่างข้อมูลก่อน ในตารางที่ 3-14 เป็นข้อมูลของสมาชิกในศูนย์สุขภาพแห่งหนึ่ง จากตารางเราจะเห็นเครื่องหมาย "?" ซึ่งมีความหมายว่าไม่มีการบันทึกข้อมูลในขณะนี้ (อาจจะมีการบันทึกในอนาคต ซึ่งจะต่างกับเครื่องหมาย "-" ซึ่งมีความหมายว่าไม่มีข้อมูล) และนั่นหมายความว่าข้อมูลในสองคอลัมน์ขวาสุดนั้น จะบันทึกหรือไม่ก็ได้

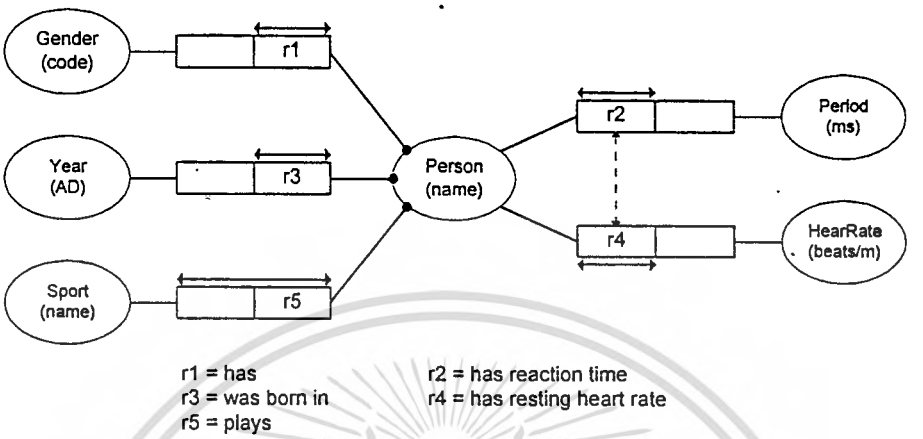
แต่จะสังเกตได้ว่าหากสมาชิกคนใดที่เข้ารับการตรวจสอบ ความเร็วในการโต้ตอบของระบบประสาท ก็จะต้องเข้ารับการตรวจอัตราการเต้นของหัวใจด้วย ซึ่งข้อกำหนดนี้อาจจะเป็นกฎของศูนย์สุขภาพก็ได้ว่า สมาชิกคนใดที่ต้องการตรวจสอบสุขภาพ จะต้องเข้ารับการทดสอบทั้งสองอย่างเสียก่อน เพราะการตรวจสอบเพียงอย่างใดอย่างหนึ่ง ไม่อาจบ่งบอกได้ถึงสุขภาพที่แท้จริงได้ ผลการทดสอบทั้งสองอย่างจะต้องนำมาพิจารณาร่วมกัน จึงสามารถบอกได้ว่าสุขภาพของสมาชิกเป็นอย่างไรบ้าง

| Member | Sex | BirthYr | Sport | Reaction Time (ms) | Resting Heart Rate beats/m) |
|---|---|---|---|---|---|
| Anderson PE | M | 1940 | golf | 250 | 80 |
| Blogge | M | 1940 | golf | ? | ? |
| Fit IM | F | 1960 | aerobics<br>tennis | 250 | 80 |
| Hume PE | F | 1946 | golf | 305 | 93 |
| Jones T | M | 1965 | tennis | ? | ? |

*ตารางที่ 3-14 ข้อมูลของสมาชิกในศูนย์สุขภาพแห่งหนึ่ง*

และเพื่อให้ได้ข้อมูลที่เป็นไปตามข้อกำหนดของศูนย์สุขภาพ ในแบบจำลองข้อมูลจึงควรจะมีการระบุถึงข้อบังคับนี้ โดยจะเรียกกฎข้อบังคับแบบนี้ว่า Equality Constraint ซึ่งจะเขียนโดยใช้เส้นประที่มีลูกศรอยู่ที่หัวและท้าย ดังตัวอย่างในรูปที่ 3-27 ซึ่งหากท่านเคยศึกษาตรรกศาสตร์มาบ้าง ก็จะนึกได้ว่าเครื่องหมายนี้ก็คือเครื่องหมายของ "ก็ต่อเมื่อ" (if and only if) นั่นเอง จากในรูปจะมีความหมายว่าเมื่อมี

การบันทึกข้อมูลลงใน r2 หรือ r4 อันใดอันหนึ่ง จะต้องมีการบันทึกข้อมูลลงในอีก Role หนึ่งด้วย ดังนั้น จำนวนครั้งของการบันทึกข้อมูลลงในชนิดความจริงทั้งสองจึงเท่ากันเสมอ และจึงเรียกคอนสแตนต์นี้ว่า Equality




*รูปที่ 3-27 แสดงการใช้ Equality Constraint*

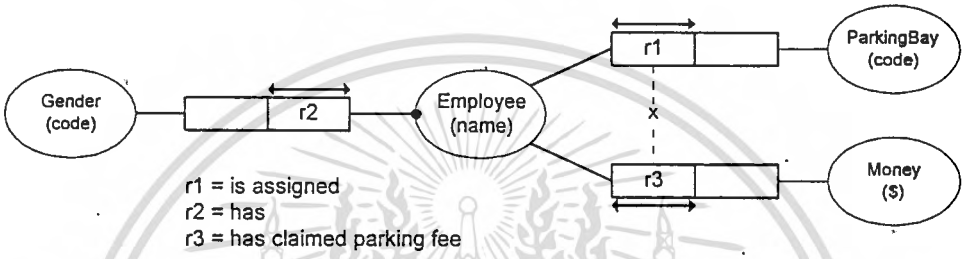
คราวนี้เรามาดูตัวอย่างข้อมูลอีกตัวอย่างหนึ่งในตารางที่ 3-15 ซึ่งเป็นข้อมูลการจอดรถของพนักงานในบริษัทแห่งหนึ่ง การจอดรถในบริษัทนี้จะมีให้จอดได้ 2 แบบคือ อาจจะจอดในที่จอดรถของบริษัท (สำหรับหัวหน้า) หรืออาจจะไปจอดในที่จอดรถข้างนอก ซึ่งจะต้องเสียค่าใช้จ่ายเพิ่มเติม (สำหรับลูกน้อง) ซึ่งพนักงานแต่ละคนจะเลือกได้ในแบบใดแบบหนึ่งเท่านั้น เพราะพนักงานแต่ละคนย่อมจะขับรถมาทำงานได้ครั้งละคันเดียว จึงย่อมต้องการที่จอดแห่งเดียวเท่านั้น

ดังนั้นจึงเห็นได้ว่าข้อมูลในคอลัมน์ขวาสุด 2 คอลัมน์จะไม่มีวันเกิดขึ้นพร้อมกันเด็ดขาด โดยพนักงานที่มีสิทธิในการจอดรถในที่จอดรถของบริษัทก็จะมีการบันทึกตำแหน่งที่จอดเอาไว้ และสำหรับพนักงานที่จอดรถนอกบริษัท ก็จะมีการบันทึกข้อมูลค่าใช้จ่ายต่อเดือน สำหรับคนที่ยังไม่มีรถก็จะยังไม่บันทึกข้อมูลในทั้งสองแบบ ดังนั้นเครื่องหมาย "?" ก็จะมีความหมายว่า อาจจะมีการบันทึกข้อมูลในอนาคต

| Employee | Gender | Parking bay | Parking Claim ($) |
|---|---|---|---|
| Adams B | F | C01 | - |
| Bloggs F | M | - | 200 |
| Collins T | M | B05 | - |
| Dancer S | F | - | 250 |
| Egghead E | M | ? | ? |

*ตารางที่ 3-15 ข้อมูลการจอดรถของพนักงานในบริษัท*

และเมื่อข้อมูลมีลักษณะของการเกิดขึ้นในแบบใดแบบหนึ่งตามตัวอย่างในตารางที่ 3-15 แล้วเมื่อมีการสร้างแบบจำลองข้อมูลที่บันทึกข้อมูลตามตัวอย่าง ก็ควรจะมีการบันทึกกฎข้อบังคับนี้ไว้ด้วย โดยให้ชื่อกฎข้อบังคับนี้ว่า Exclusion Constraint โดยรูปที่ 3-28 จะเป็นแบบจำลองข้อมูลที่สร้างจากตาราง ซึ่งจะเห็นได้ว่ามีการบันทึก Exclusion Constraint ไว้โดยการเขียนเส้นประแล้วมีเครื่องหมาย "x" หรือกากบาทตรงกลาง ซึ่งมีความหมายว่าหากมีการบันทึกข้อมูลพนักงานใน r1 แล้วจะต้องไม่มีการบันทึกข้อมูลพนักงานคนเดียวกันนั้นใน r3 อีก



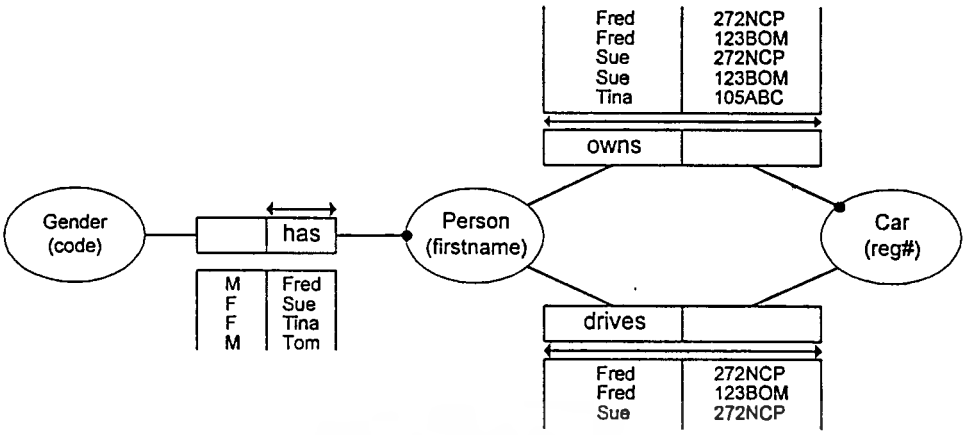

*รูปที่ 3-28 แบบจำลองข้อมูลที่แสดงการใช้ Exclusion Constraint*

เอาละท้ายสุดนี้เราก็มาดูคอนสแตนต์ตัวสุดท้ายกัน โดยจะเริ่มจากข้อมูลในตารางที่ 3-16 ซึ่งเป็นข้อมูลของการเป็นเจ้าของรถและการเป็นผู้ขับรถ โดยรถแต่ละคันจะใช้เลขทะเบียนเป็นชื่อเรียก จากข้อมูลในตารางจะเห็นได้ว่า รถแต่ละคันจะมีเจ้าของได้หลายคน และแต่ละคนก็เป็นเจ้าของรถได้หลายคันเช่นกัน เช่น Fred และ Sue นั้นเป็นคู่สามีภรรยากันดังนั้นจึงเป็นเจ้าของรถร่วมกัน สำหรับข้อมูลการเป็นผู้ขับรถก็เป็นแบบ Many to Many เช่นกัน เพราะแม้ว่า Fred และ Sue จะเป็นเจ้าของรถ 2 คันร่วมกัน แต่ Sue ก็ขับรถเพียงคันเดียวเท่านั้น

จากข้อมูลดังกล่าวจะเห็นได้ว่า ผู้ขับรถนั้นจำเป็นต้องเป็นเจ้าของรถ แต่เจ้าของรถไม่จำเป็นต้องเป็นคนขับ ดังนั้นจะสามารถเขียนแบบจำลองข้อมูลได้ดังรูปที่ 3-29 ซึ่งจะเห็นว่าชนิดความจริงที่บันทึกข้อมูลการขับรถนั้นจะเป็นซับเซ็ตของชนิดความจริงการเป็นเจ้าของ และจะเขียนคอนสแตนต์โดยใช้เส้นประโดยมีลูกศรข้างเดียว โดยชี้ไปทางด้านที่เป็นซูเปอร์เซ็ต โดยเรียกคอนสแตนต์นี้ว่า Subset Constraint

| *Person* | *Sex* | *CarsOwned* | *CarsDriven* |
|---|---|---|---|
| Fred | M | 272NCP; 123BOM | 272NCP; 123BOM |
| Sue | F | 272NCP; 123BOM | 272NCP |
| Tina | F | 105ABC | ? |
| Tom | M | ? | ? |

*ตารางที่ 3-16 ข้อมูลการเป็นเจ้าของรถและข้อมูลการขับรถ*




*รูปที่ 3-29 แสดงแบบจำลองข้อมูลที่ได้จากตารางที่ 3-16*

**3.2.9 ขั้นตอนที่ 9 Final Check**

เมื่อมาถึงขั้นนี้แบบจำลองข้อมูลที่เราได้ออกแบบมาตามขั้นตอนทั้ง 8 ขั้น ก็คงจะอยู่ในรูปแบบที่เกือบจะสมบูรณ์แล้ว สำหรับการทำงานในขั้นตอนสุดท้ายนี้ก็เป็นเพียงการตรวจสอบความผิดพลาดใด ๆ ที่อาจเกิดขึ้น และแก้ไขแบบจำลองข้อมูลให้ถูกต้องต่อไป ซึ่งการตรวจสอบความถูกต้องของแบบจำลองข้อมูลจะประกอบด้วย 3 ขั้นตอนด้วยกัน คือ การตรวจสอบกับตัวอย่างข้อมูล, การตรวจสอบความซ้ำซ้อน และการตรวจสอบขั้นสุดท้าย

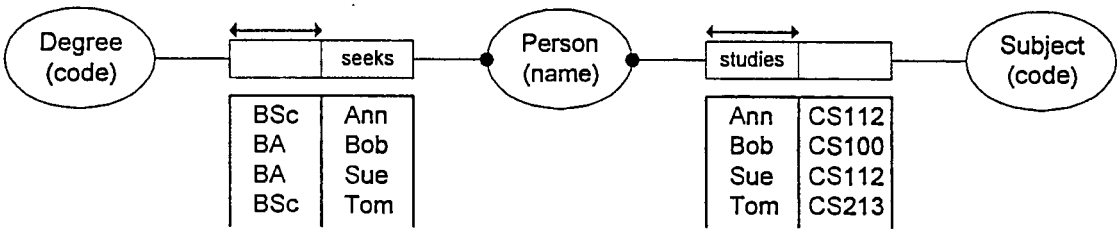
**3.2.9.1 การตรวจสอบกับตัวอย่างข้อมูล**

การทำงานที่ผ่านมาตั้งแต่ขั้นตอนที่ 1 จนถึงขั้นตอนที่ 8 เราได้สร้างแบบจำลองข้อมูล โดยอ้างอิงกับตัวอย่างข้อมูลมาโดยตลอด ดังนั้นก่อนที่จะตรวจสอบความถูกต้องในแง่อื่น ๆ เราควรจะต้องตรวจสอบดูก่อนว่าแบบจำลองข้อมูลที่ได้นั้น ยังใช้กับตัวอย่างข้อมูลได้อยู่หรือไม่ คอนสแตนต์ต่าง ๆ ที่กำหนดขึ้นนั้นมีการขัดแย้งกับข้อมูลหรือไม่ ซึ่งหากแบบจำลองมีการขัดแย้งกับข้อมูล ก็จะต้องแก้ไขแบบจำลองข้อมูลให้ถูกต้องต่อไป

การตรวจสอบในขั้นนี้นั้นเราจะตรวจสอบโดยใช้แผนภูมิที่เรียกว่า Schema-Base Diagram ดังตัวอย่างในรูปที่ 3-30 ซึ่งสร้างจากตัวอย่างข้อมูลในตารางที่ 3-17 ซึ่งจะเห็นได้ว่าเราได้ทดลองใส่ข้อมูลลงในแบบจำลองโดยตรง ซึ่งหากข้อมูลมีจำนวนมากก็ให้ใช้วิธีสุ่มเอาข้อมูลมาทดสอบก็ได้ จากผลการทดสอบในส่วนของชนิดความจริงนั้น ไม่มีอะไรผิดพลาดโดยแบบจำลองสามารถแทนข้อมูลจากตารางได้เป็นอย่างดี

| Person | Subject | Degree |
|---|---|---|
| Ann | CS113 | BSc |
| Ann | CS100 | BSc |
| Bob | ? | BA |
| Sue | CS112 | BA |
| Tom | CS213 | BSc |

*ตารางที่ 3-17 ตัวอย่างข้อมูล*




*รูปที่ 3-30 แบบจำลองข้อมูลที่ได้จากตารางที่ 3-17*

แต่ในส่วนของคอนสแตนต์นั้นจะเห็นได้ว่า Uniqeness Constraint เกิดการขัดแย้งขึ้น เพราะในชนิดความจริงของวิชาที่เรียนนั้น คอนสแตนต์กำหนดไว้ว่าแต่ละคนจะเรียนได้มากที่สุดวิชาเดียวเท่านั้น แต่ในตัวอย่างข้อมูล Ann เรียนถึง 2 วิชา ดังนั้นในส่วนนี้จะต้องแก้ไขแบบจำลอง โดยจะต้องจะต้องขยายคอนสแตนต์ให้ครอบคลุมทั้ง 2 Role (Many to Many)

สำหรับ Uniqueness Constraint ที่ชนิดความจริงสาขาวิชานั้น ก็เกิดการผิดพลาดเช่นกัน เพราะในแบบจำลองกำหนดไว้ว่าข้อมูลในส่วนของสาขาวิชาจะต้องไม่ซ้ำ แต่ปรากฏว่ามีสาขาวิชา BSC และ BA เกิดขึ้นอย่างละ 2 ครั้ง ดังนั้นจะต้องแก้ไขคอนสแตนต์นี้เช่นกัน โดยคอนสแตนต์ที่ถูกต้องจะต้องครอบคลุมทั้ง 2 Role เช่นกัน และท้ายสุดก็ต้องแก้ไข Mendatory Constraint ในฝั่งของวิชาเรียนด้วย เพราะข้อมูลของ Bob นั้นไม่ได้บันทึกไว้ จึงขัดแย้งกับคอนสแตนต์นี้

คงจะเห็นแล้วนะครับว่า การตรวจสอบขั้นต้นโดยวิธีตรวจสอบกับตัวอย่างข้อมูล ก็เป็นวิธีการตรวจสอบแบบง่าย ๆ ที่ให้ผลดี และนอกจากจะตรวจสอบความขัดแย้งที่เกิดขึ้นคอนสแตนต์แล้ว สิ่งหนึ่งที่ต้องไม่ลืมเช่นกัน ก็คือ การเพิ่มคอนสแตนต์เข้าไปหากพบว่ากฎข้อบังคับที่ใส่ให้กับแบบจำลองข้อมูลนั้นอ่อนเกินไป ซึ่งจากจุดนี้จะเห็นว่าการรวบรวมข้อมูลนั้นสำคัญมากจริง ๆ เพราะหากข้อมูลที่เราหามาไม่สมบูรณ์ แบบจำลองข้อมูลที่ได้ก็จะไม่สมบูรณ์เช่นกัน และอาจทำให้ไม่สามารถนำไปใช้กับข้อมูลจริงได้

**3.2.9.2 การตรวจสอบความซ้ำซ้อน**

ในขั้นตอนการออกแบบที่ผ่านมานั้น เป้าหมายเด่นชัดอย่างหนึ่งที่เรายึดถือมาตลอด ก็คือ การลดความซ้ำซ้อนในการจัดเก็บข้อมูล การลดความซ้ำซ้อนนั้นนอกจากจะช่วยให้สิ้นเปลืองเนื้อที่ในการจัดเก็บน้อยแล้ว ก็ยังช่วยลดปัญหาในเรื่องของการอัปเดตข้อมูล (Update Anomalies) ได้อีกด้วย ความซ้ำซ้อนที่เกิดขึ้นนี้ สามารถแบ่งได้ 2 แบบด้วยกัน คือ Stored Redundancy ซึ่งหมายถึงมีข้อมูลเดียวกัน เก็บอยู่มากกว่า 1 ที่ขึ้นไป ซึ่งหากมีการออกแบบที่ดีพอแล้ว ความซ้ำซ้อนแบบนี้จะไม่เกิดขึ้นเลย

โดยทั่วไปการเกิด Stored Redundancy มักจะเกิดขึ้นได้ หากมีการกำหนด Elecmentary Facts ไม่ถูกต้อง คงจำกันได้ว่าในขั้นตอนที่ 5 (Arity Checking) ก็ได้มีการตรวจสอบ Elecmentary Facts ไปครั้งหนึ่งแล้ว ซึ่งวิธีการหนึ่งที่เราใช้ทดสอบก็คือ การตรวจสอบความซ้ำซ้อนนั่นเอง ความซ้ำซ้อนนั้นบางครั้งอาจจะเกิดขึ้นในรูปแบบที่ไม่ชัดเจนนัก เช่น มีชนิดความจริงอยู่ 2 ตัวโดยชนิดความจริงแรกจะบันทึกข้อมูลที่ว่า คนงาน (Emp#) จะต้องมีแผนกที่ทำงาน (Dept) ชนิดความจริงนี้จะเป็นแบบไบนารี สำ

หรับอีกชนิดความจริงหนึ่งจะบันทึกข้อมูลว่า คนงาน (Emp#) แผนก (Dept) จะต้องมีวุฒิการศึกษา โดยชนิดความจริงนี้เป็นเทอนารี ซึ่งจะเห็นได้ว่ามีการบันทึกข้อมูลซ้ำกัน แต่บางครั้งอาจจะมองข้ามไปได้ ดังนั้นจะต้องพิจารณาให้ดี

คราวนี้เรามาดูความซ้ำซ้อนแบบที่ 2 คือ Derived Redundancy ซึ่งความหมายของความซ้ำซ้อนแบบนี้ก็คือ มีการบันทึกความจริง 2 ความจริง โดยที่ความจริงหนึ่งสามารถจะหาได้จากอีกความจริงหนึ่ง ตัวอย่างของความซ้ำซ้อนแบบนี้ เราได้เคยกล่าวไว้แล้วครั้งหนึ่ง ในกรณีของการบันทึกการทอนเงินในตัวอย่างที่ผ่านมาตอนต้นบท ทั้งที่เงินทอนนั้นสามารถจะหาได้จากสูตร เงินต้น-ราคาสินค้า ดังนั้นการบันทึกความจริงของเงินทอนก็จะถือว่าเป็นการซ้ำซ้อนแบบหนึ่งด้วยเช่นกัน ดังนั้นจะต้องคฃตรวจสอบได้ดี แต่โดยทั่วไปแล้วการออกแบบที่ดีก็จะช่วยให้ความจำเป็นในการตรวจสอบความซ้ำซ้อนลดลงได้

3.2.9.3 การตรวจสอบความสมบูรณ์

อันที่จริงการตรวจสอบขั้นสุดท้ายนี้ ก็ไม่ได้เป็นอะไรมากไปกว่าการตรวจสอบว่ายังมีการหลงลืมอะไรเล็ก ๆ น้อย ๆ บ้างหรือเปล่าเท่านั้นเอง ดังนั้นการทำงานในขั้นตอนนี้จึงไม่มีหลักเกณฑ์ตายตัวอะไร การตรวจสอบก็เป็นงานทั่ว ๆ ไป เช่น ตรวจสอบชื่อของชนิดเอนติตี้ ชนิดความจริง ชนิดเลเบิ้ล ชื่อของคอนสแตนต์ต่าง ๆ ตรวจสอบว่ามีการตกหล่นไปหรือไม่ หากตกหล่นก็เติมให้ครบถ้วน เมื่อไม่มีอะไรบกพร่องแล้ว ก็คงจะถือได้ว่าแบบจำลองข้อมูลของคุณ สมบูรณ์แล้วครับ

## 3.3 การแปลงจาก Conceptual Schema ไปเป็น Relational Schema

ในขั้นตอนก่อนหน้านี้ เราได้ศึกษาถึงวิธีการออกแบบแบบจำลองข้อมูลระดับแนวคิดมาแล้ว ซึ่งแม้ว่าจะเป็นแบบจำลองข้อมูลที่สมบูรณ์ สามารถรองรับข้อมูลได้ถูกต้องโดยมีการควบคุมความผิดพลาดได้เป็นอย่างดี แต่เป็นที่น่าเสียใจว่าในปัจจุบันยังไม่มีระบบฐานข้อมูลใดที่สามารถใช้งานกับแบบจำลองข้อมูลไนเอมได้ หากแต่เป็นระบบฐานข้อมูลที่เป็นแบบรีเลชันแนล

ระบบฐานข้อมูลแบบรีเลชันแนลได้รับการพัฒนาจนมีประสิทธิภาพสูง และมีภาษา SQL (Structured Query Language) ซึ่งเป็นภาษาหลักของแบบจำลองข้อมูล (Data Model) ที่เป็นรีเลชันแนล และได้นำไปใช้งานอย่างกว้างขวางในวงการต่างๆ ซึ่งเมื่อไม่นานมานี้ภาษา SQL ก็ได้รับการรับรองให้เป็นภาษามาตรฐาน ทำให้มีการใช้งานภาษานี้ทั้งบนเครื่องเมนเฟรม มินิคอมพิวเตอร์ ไมโครคอมพิวเตอร์ ในระบบปฏิบัติการ (Operating System) ต่างๆ และนอกจากที่ได้กล่าวมาแล้วนั้น SQL ยังสามารถที่จะเชื่อมต่อกับภาษาต่างๆเช่น COBOL หรือ C ได้อีกด้วย ทำให้การใช้งานคล่องตัวยิ่งขึ้น

ดังนั้นต่อไปเราก็จะนำเสนออัลกอริทึมที่ใช้ในการแปลงจาก Conceptual Schema ไปเป็น Relational Schema เพื่อจะนำเอาแบบจำลองข้อมูลแบบรีเลชันแนลไปใช้งานต่อไป

### 3.3.1 แบบจำลองข้อมูลแบบรีเลชันแนล

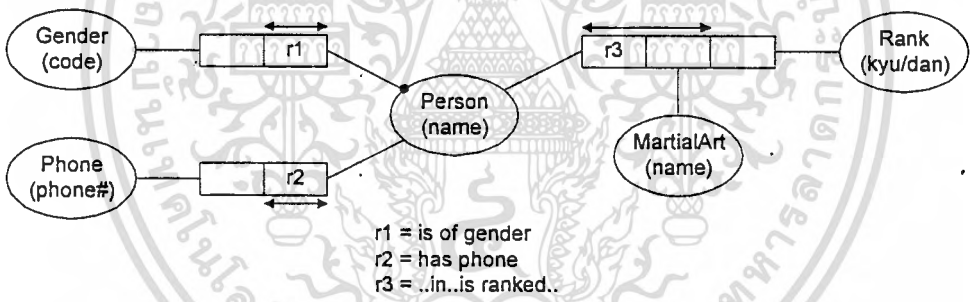
แบบจำลองข้อมูลแบบรีเลชันแนล มีโครงสร้างข้อมูลที่สำคัญคือตารางซึ่งใช้ในการแสดงความสัมพันธ์ และในแต่ละตารางจะประกอบด้วยเซ็ตของ tuple ซึ่งเรามักจะเรียกว่า rows หรือ records และหากเราดูตารางให้ดีจะเห็นว่าเราได้แบ่งตามแนวนอนเป็น rows แบ่งตามแนวตั้งเป็นคอลัมน์ โดยที่ทั้ง

ตารางและคอลัมน์นั้นจะเป็นชื่อ ในขณะที่ rows นั้นไม่ใช่ และหากเรานำทั้ง row และคอลัมน์มาอินเตอร์เซ็ค (Intersection) เราก็จะได้ฟิลด์ (Field) ซึ่งจะเป็นที่เก็บข้อมูล 1 ข้อมูลซึ่งอาจจะเป็น NULL ได้ และสำหรับข้อมูลในแต่ละฟิลด์ในตารางจะต้องเป็นชนิดเดียวกัน

เพื่อที่จะแสดงรูปแบบของโครงสร้างข้อมูลแบบรีเลชันแนล เราจะลองมาดูข้อมูลในตารางที่ 3-18 ซึ่งเราจะใช้เป็นตัวอย่างในการอธิบาย ซึ่งข้อมูลในตารางนั้นเป็นข้อมูลของสมาชิกในสโมสรแห่งหนึ่ง ซึ่งจะสังเกตได้ว่ามีข้อมูลที่เป็น NULL อยู่หลายตัวซึ่งจะหมายถึงว่าไม่ได้รับการบันทึก เช่น สมาชิกบางคนอาจจะไม่มีเบอร์โทรศัพท์ และจากข้อมูลดังกล่าว เราสามารถหา Conceptual Schema ได้ดังรูปที่ 3-31

| *Member* | *Gender* | *Phone* | *Art* | *Rank* |
|---|---|---|---|---|
| Adams B | M | 2052777 | judo | 3dan |
| | | | karatedo | 2kyu |
| Adams S | F | 2052777 | judo | 2kyu |
| Brown C | F | 3579001 | ? | ? |
| Collins T | M | ? | aikido | 2dan |
| | | | judo | 2dan |
| Dancer A | F | ? | ? | ? |

*ตารางที่ 3-18 ข้อมูลสมาชิกของสโมสรแห่งหนึ่ง*




*รูปที่ 3-31 แบบจำลองข้อมูลที่ได้จากตารางที่ 3-18*

| *Member:* | *Name* | *Gender* | *Phone* |
|---|---|---|---|
| | Adams B | M | 2052777 |
| | Adams S | F | 2052777 |
| | Brown C | F | 3579001 |
| | Collins T | M | ? |
| | Dancer A | F | ? |
| *Ranks:* | *Person* | *Art* | *Rank* |
| | Adams B | judo | 3dan |
| | Adams B | karatedo | 2kyu |
| | Adams S | judo | 2kyu |
| | Collins T | aikido | 2dan |
| | Collins T | judo | 2dan |

*ตารางที่ 3-19 แสดงตารางที่ได้จากแบบจำลองข้อมูล*

แบบจำลองข้อมูลดังกล่าวประกอบด้วย 3 ชนิดความจริง ดังนั้นในฐานข้อมูลในระดับแนวคิด (Conceptual Database) เราก็จะได้ 3 ตารางความจริง (fact table) โดยที่แต่ละ row ก็จะเก็บแต่ละ Elementary Facts แต่สำหรับในฐานข้อมูลแบบรีเลชันแนลแล้ว เราจะรวมชนิดความจริงที่เป็น Binary เข้าด้วยกันดังนั้น เราจะได้ตารางขึ้นมา 2 ตารางดังตารางที่ 3-19

**3.3.2 Optimal Normal Form Algorithm**

ตารางที่ได้จากการแปลงแบบจำลองข้อมูลที่เราได้กล่าวมาแล้ว เป็นตารางที่แปลงมาอย่างถูกต้อง จึงสามารถเก็บข้อมูลได้อย่างครบถ้วนและไม่ซ้ำซ้อน จากนั้นเราจะนำตารางที่ได้นี้ไปสร้างบนฐานข้อมูลแบบรีเลชันแนล กำหนดรูปแบบของตาราง และการเขียนรูทีนเพื่อจัดการกับคอนสแตนต์ต่าง ๆ แต่การแปลงโดยไม่มีกฎเกณฑ์ก็อาจทำให้เกิดข้อผิดพลาดได้ง่าย ดังนั้นต่อไปเราก็จะแนะนำวิธีการที่เป็นระบบในการแปลงแบบจำลองข้อมูลในแบบแนวคิดไปเป็นแบบรีเลชันแนล

สำหรับแต่ละ Conceptual Schema นั้นเมื่อแปลงเป็น Relational Schema นั้นอาจจะแปลงได้หลายรูปแบบ แต่สำหรับอัลกอริทึม ONF นั้นมีจุดประสงค์เพื่อสร้างรูปแบบที่ง่าย ประหยัด และมีประสิทธิภาพ โดยมีหลักการต่อไปนี้ ประการแรกคือเพื่อให้ได้โครงสร้างข้อมูลที่ง่าย ไม่มีแอทริบิวต์ใดซ้ำ ประการที่สองคือป้องกันการผิดพลาดอันเนื่องมาจากการอัพเดต (Update Anomolies) ข้อมูลไม่มีการซ้ำซ้อน ประการที่ 3 เพื่อให้ได้ประสิทธิภาพ ดังนั้นจะต้องได้จำนวนตารางที่น้อยด้วย เพราะการลดจำนวนตารางก็เท่ากับการลดเวลาที่ใช้ในการเข้าถึงข้อมูลและยังทำให้การทำ Query สามารถทำได้ง่ายขึ้นอีกด้วย

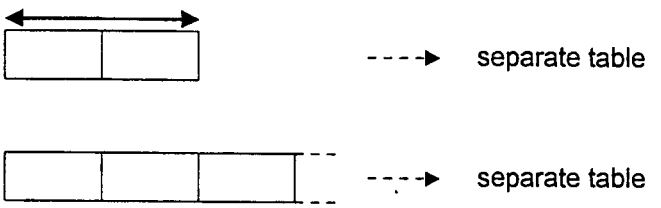
การแปลงเป็นตารางนี้ หากจะเทียบกับวิธี Normalization ซึ่งเป็นวิธีการสร้างตารางแล้ว วิธีการสร้างแบบจำลองข้อมูลและแปลงเป็นตารางที่กล่าวถึงไปแล้วจะง่ายกว่า เพราะวิธีการ Normalization จะเริ่มจาก First Normal Form และสิ้นสุดที่ Fifth Normal Form ในขณะที่ ONF สามารถที่จะสร้าง Fifth Normal Form ได้เลย และสำหรับ ONF Algorithm ก็มีดังต่อไปนี้

1. สำหรับแต่ละชนิดความจริงที่ไม่ได้มี simply key ให้สร้างเป็นตารางแยกออกมา โดยเลือกคีย์ที่สั้นที่สุดของชนิดความจริงนั้นเป็นคีย์หลัก

2. รวมชนิดความจริงที่มี simple key ที่เชื่อมกับ object type เดียวกันเป็นตารางเดียวกัน และให้ object type นั้นเป็นคีย์หลัก

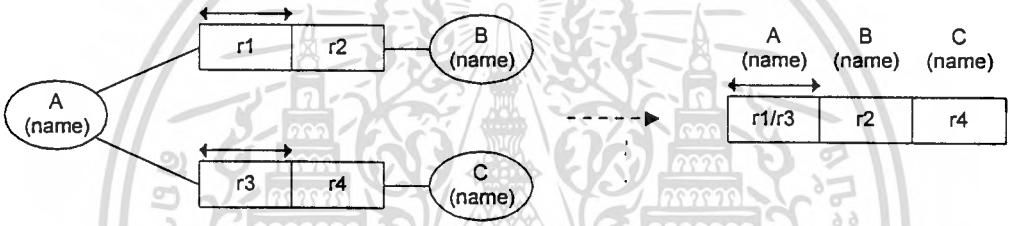
3. สำหรับชนิดความจริงที่เหลือให้สร้างเป็นตารางแยกต่างหาก

เพื่อที่จะเข้าใจอัลกอริทึม ONF เราจะลองมาดูตัวอย่างกัน โดยจะเริ่มจากกำหนดให้ทุกๆ ชนิดเอนติตี้มีความสัมพันธ์กับชนิดอ้างอิงเป็นแบบ 1:1 และไม่มีการใช้สับไทป์ เราจะเริ่มที่ขั้นตอนที่ 1 ซึ่งกล่าวไว้ว่าชนิดความจริงที่ไม่มี simply key จะต้องสร้างตารางขึ้นมารองรับต่างหาก ถ้าแบบจำลองข้อมูลของเรามีความถูกต้องการที่จะเกิดตามเงื่อนไขที่ว่าไม่มี Simple Key นั้นก็ต่อเมื่อเป็นชนิดความจริงแบบไบนารีที่มีความสัมพันธ์กันแบบ many to many หรือเป็นชนิดความจริงที่มี Role มากกว่า 2 เช่นจากรูปที่ 3-32 รูปบนเป็นไบนารีแบบ many to many ส่วนข้างล่างเป็นชนิดความจริงที่มี role มากกว่า 2 ในลักษณะดังรูปนี้จะต้องแยกออกเป็นตารางเดี่ยวออกมา




*รูปที่ 3-32 ตัวอย่างของชนิดความจริงที่ต้องแยกเป็นตารางเดี่ยวออกมา*

สำหรับเงื่อนไขที่ 2 นั้นกล่าวว่าหากมีชนิดเอนติตี้ใดที่มี ชนิดความจริงที่เป็น simply key เชื่อมอยู่ เราจะรวมชนิดความจริงเป็นหนึ่งตาราง และให้ชนิดเอนติตี้นั้นเป็นคีย์หลัก เช่นจากรูปที่ 3-33 ชนิดเอนติตี้ A มี 2 ชนิดความจริงที่มี simply key เชื่อมอยู่ดังนั้นเราจะรวม 2 ชนิดความจริงนี้เป็นหนึ่งตาราง ดังรูป




*รูปที่ 3-33 แสดงการรวม 2 ชนิดความจริงที่มี simple key เดียวกัน*

และสำหรับเงื่อนไขที่ 3 นั้นกล่าวว่าชนิดความจริงที่เหลือให้สร้างเป็นตารางต่างหาก ซึ่งก็หมายความว่าชนิดความจริงใด ที่ไม่ได้มีการแมปไปในขั้นตอนก่อนหน้านี้ ก็ให้สร้างเป็นชนิดความจริงต่อหนึ่งตารางเลย ซึ่งตรงนี้ก็จะสะท้อนกลับไปถึงขั้นตอนการออกแบบ เพราะในกรณีเดียวกันในบางครั้งก็อาจจะแมปได้ปริมาณตารางที่มาก แต่บางครั้งก็จะได้ปริมาณตารางที่น้อย ขึ้นอยู่กับการออกแบบ ดังนั้นหากคุณต้องการสร้างแบบจำลองข้อมูลเพื่อใช้ในฐานข้อมูลแบบรีเลชันแนล ก็จะต้องคำนึงถึงข้อนี้ด้วย

หลังจากที่ออกแบบและสร้างตารางแล้ว เนื่องจากระบบฐานข้อมูลรีเลชันแนลทั่วไป ไม่มีความสามารถในการควบคุมตามกฎข้อบังคับที่ได้ตั้งเอาไว้ ดังนั้นหากคุณต้องการบังคับข้อมูลให้เป็นไปตามที่กำหนดเอาไว้ คุณก็จะต้องเขียนโปรแกรมเพื่อควบคุมกฎข้อบังคับเอง

**หนังสืออ้างอิง**

[1] Nijssen G.M. (1977) : "*On the Gross Arthitecture for the Next Generation Database Management Systems*", Information Processing 77, Gilchrist B.Ed., IFIP, North-Holland Publishing Company, pp. 327-335

[2] G.M. Nijssen, T.A. Halpin : "*Conceptual Schema and Relational Database Design : A fact oriented approach*",Prentice Hall 1989

# บทที่ 4

# ระบบฐานข้อมูลอนุมาน

## 4.1 บทนำ

ดังที่ได้กล่าวมาแล้วว่า ระบบฐานข้อมูลอนุมานได้รับการพัฒนาโดยมีพื้นฐานมาจากสาขาหลัก 2 สาขาด้วยกันคือ การโปรแกรมเชิงตรรก (Logic Programming) และระบบฐานข้อมูลแบบรีเลชันแนล (Relational Database System) สำหรับสาเหตุที่เรียกฐานข้อมูลชนิดนี้ว่า "อนุมาน" (Deductive) นั้นก็เนื่องจากระบบฐานข้อมูลชนิดนี้ มีความสามารถในการอนุมานข้อมูลใหม่ได้จากข้อมูลที่มีอยู่แล้วได้ โดยข้อมูลจะเก็บอยู่ในรูปของความจริง (Facts) และกฎ (Rule) (ระบบฐานข้อมูลอนุมานนี้อาจถูกเรียกในชื่ออื่นได้ เช่น Logic Database [Gallaire and Minker 1978], Deductive Relational Databases [Minker 1982] และ Virtual Relational Database [Debanham and McGrath 1983] ซึ่งความสามารถในการอนุมานนั้นก็เป็นผลมาจากพัฒนาการทางด้านปัญญาประดิษฐ์นั่นเอง

ทฤษฎีทางด้านฐานข้อมูลอนุมานสามารถที่จะมองได้ว่า เป็นส่วนที่เพิ่มเติมขึ้นมาจากทฤษฎีของระบบฐานข้อมูลแบบรีเลชันแนล โดยมองว่าฐานข้อมูลแบบรีเลชันแนลจะเก็บเฉพาะความจริง (Collection of Facts) ในขณะที่ระบบฐานข้อมูลอนุมานจะเก็บข้อมูลทั้งในรูปของความจริงและในรูปของกฎ โดยจะแทนความจริงและกฎด้วย FOPL (First Order Predicate Logic) และจากการใช้ FOPL มาใช้นี่เองทำให้ฐานข้อมูลนี้ไม่เพียงแต่จะใช้เก็บข้อมูลเท่านั้น แต่จะเก็บทั้งโปรแกรม คำถาม วิว และกฎข้อบังคับต่างๆ ได้

แรงบันดาลใจอีกประการหนึ่งของการพัฒนาระบบฐานข้อมูลอนุมาน ก็มาจากความจริงที่ว่า FOPL นั้นสามารถใช้งานในลักษณะของโปรแกรมได้ ซึ่งทำให้เป็นที่สนใจมากในระยะหลังนี้ จนกระทั่งได้ผลลัพธ์ออกมาเป็นภาษาที่สามารถใช้งานได้นั่นก็คือภาษาโปรลอก (PROgramming in LOGic) ภาษาโปรลอกเป็นภาษาที่มีลักษณะที่ง่ายและมีประสิทธิภาพสูง สามารถนำไปใช้งานได้ในหลายๆ ด้านรวมทั้งด้านปัญญาประดิษฐ์และระบบฐานข้อมูลด้วย

## 4.2 การโปรแกรมเชิงตรรก

ระบบของการโปรแกรมเชิงตรรกส่วนใหญ่ที่มีในปัจจุบันนี้ก็คือ ภาษาโปรลอกนั่นเอง และเพื่อที่จะสามารถเข้าใจได้ดีขึ้น เราก็จะยกตัวอย่างง่าย ๆ ของภาษาโปรลอกเพื่อที่จะประกอบการอธิบาย

```
grandparent(x,y) :- parent(x,z),parent(z,y)
parent(x,y) :- mother(x,y)
parent(x,y) :- father(x,y)
ancestor(x,y) :- parent(z,y),ancestor(x,z)
ancestor(x,y) :- parent(x,y)
```

father(fred,mary)

father(george,james)

father(john,fred)

mother(sue,mary)

mother(jane,sue)

mother(liz,fred)

mother(sue,james)

รูปแบบของโปรแกรมภาษาโปรลอกนั้นยังไม่มีการกำหนดเป็นมาตรฐาน แต่ภาษาโปรลอกที่ใช้งานกันอยู่ในปัจจุบัน ก็จะมีรูปแบบที่ใกล้เคียงกับที่ยกตัวอย่างข้างต้น กล่าวคือโปรแกรมจะประกอบด้วยลำดับของ Clause (Collection of Clauses) เช่น ancestor(x,y) :- parent(x,y) โครงสร้างของ Clause เช่น ancestor(x,y) เรียกว่าอะตอม แต่ละอะตอมจะมีสัญลักษณ์พรีดิเขตตามด้วย Argument ดังนั้น grantparent, parent, ancestor, ... จึงล้วนเป็นพรีดิเขต โดยที่ argument นั้นอาจจะเป็นตัวแปร เช่น x, y, z หรือค่าคงที่ เช่น mary, john ผู้เขียนโปรแกรมสามารถที่จะกำหนดชื่อของพรีดิเขต, ค่าคงที่และตัวแปรต่างๆ ได้อย่างอิสระ

แต่ละ Clause ของโปรลอกเป็น Formula ของ FOPL ที่ Quantifier ของตัวแปรทุกตัวต้องเป็น Universal และเครื่องหมาย :- ก็มีความหมายเท่ากับ -> หรือ ถ้า..แล้วของ FOPL และเครื่องหมาย "," ที่คั่นอยู่ระหว่างอะตอมก็มีความหมายว่า "และ" นอกจากนั้นจะเรียกพรีดิเขตที่อยู่ทางด้านซ้ายของเครื่องหมาย :- ซึ่งจะต้องมีเพียงพรีดิเขตเดียวเท่านั้นว่า Head และเรียกส่วนที่เหลือของ Clause ที่อยู่ทางด้านขวาของเครื่องหมาย :- ว่า Body และจะเห็นได้ว่าในบางกรณีใน Clause ก็ไม่มี Body และเราจะเรียก Clause ที่ไม่มี Body ว่าความจริง (Fact) และเรียก Clause ที่มี Body ว่ากฎ (Rule) ดังนั้นเราอาจจะมองได้ว่าโปรแกรมภาษาโปรลอกก็คือลำดับของความจริงและกฎนั่นเอง

ในการนำโปรแกรมภาษาโปรลอกมาใช้งาน จะต้องมีการสร้างคำถาม (Query) หรือ Goal เช่น หากต้องการหาว่าใครเป็นพ่อของ fred ก็จะต้องใช้คำถาม father (X,fred) ซึ่งก็จะได้คำตอบเป็น X=john หรือสำหรับคำถาม mother(sue,X) ก็คือคำถามที่ต้องการหาว่าใครบ้างที่เป็นลูกของ sue ซึ่งในที่นี้คำตอบก็จะเป็น mary และ james ซึ่งปกติแล้วคำตอบจะออกมาทีละคำตอบ แต่ก็สามารถที่จะสร้างให้มีการให้คำตอบออกมาในรูปของเซ็ตได้

สำหรับ Argument ของภาษาโปรลอกนั้นจะเห็นได้ว่าจะมีลักษณะพิเศษคือสามารถเป็นได้ทั้งอินพุตและเอาต์พุต โดยที่หาก Argument นั้นเป็นค่าคงที่ Argument นั้นจะเป็นอินพุต และหาก Argument นั้นเป็นตัวแปร Argument นั้นจะเป็นเอาต์พุต ซึ่งจะเห็นได้ว่าจะสะดวกในการเขียนโปรแกรมมาก อย่างไรก็ตามหากคำถามมีลักษณะดังเช่น parent(fred,mary) ซึ่งจะเห็นได้ว่าไม่มี Argument ใดเป็น เอาต์พุต ในคำถามลักษณะเช่นนี้โปรลอกจะให้คำตอบออกมาเป็น yes หรือ no แทน ซึ่งสำหรับคำถามข้างต้นคำตอบก็คือ yes

ทีนี้เราก็จะมาดูกันต่อว่าโปรลอกมีวิธีการหาคำตอบมาให้เราได้อย่างไร ลองมาดูคำถามที่ว่า grandparent(john,X) สำหรับคำตอบของคำถามนี้ โปรลอกจะมีลักษณะเป็น grandparent(john,X) โดยที่ X นั้นจะถูกแทนด้วยค่าของคำตอบที่ได้จากการอนุมานของโปรแกรม ซึ่งในที่นี้ X จะมีค่าเป็น mary และนั่นหมายความว่าคำตอบที่ได้จากการอนุมานก็คือ grandparent(john,mary) และสำหรับคำถามนี้คำตอบได้มาจากการอนุมานของ grandparent และกฎที่ว่า parent(x,y) :- father(x,y) และจากความจริงของ father (john,fred) และ father(fred,mary)

สำหรับรายละเอียดว่าการอนุมานมีหลักการอย่างไรนั้น เราจะไม่กล่าวในที่นี้ อย่างไรก็ตามเราสามารถกล่าวได้ว่าทฤษฎีทางด้านการโปรแกรมเชิงตรรกนั้นมีพื้นฐานมาจากทฤษฎีทางด้าน Automatic Theorem Proving ซึ่งเป็นสาขาหนึ่งของปัญญาประดิษฐ์ โดยเราอาจจะมองได้ว่าคำถามที่เราต้องการคำตอบก็คือทฤษฎี (Theorem) ที่ต้องการพิสูจน์ และคำตอบที่เราต้องการหานั้นก็คือการพิสูจน์ว่าทฤษฎีเป็นจริง โดยที่แต่ละขั้นตอนในการหาคำตอบก็คือขั้นตอนในการพิสูจน์ทฤษฎีนั่นเอง แต่ละขั้นตอนของการอนุมานนั้นจะใช้วิธีการที่เรียกว่า Resolution และขั้นตอนหลักของ Resolution เรียกว่า Unification

ภาษาโปรลอกเป็นภาษาที่ง่ายต่อการเรียนรู้ ง่ายต่อการเขียน และง่ายต่อการแก้ไข โปรลอกเป็นภาษาที่มีประสิทธิภาพ และนอกจากนั้นโปรลอกยังได้รับเลือกจากญี่ปุ่นให้เป็นภาษาสำหรับการพัฒนาระบบคอมพิวเตอร์ยุคที่ 5 (Japan's fifth generation computer system project) อีกด้วย

### 4.3 ระบบฐานข้อมูลอนุมาน

ระบบฐานข้อมูลอนุมานนั้น เมื่อมองในมุมของแนวคิดแล้ว ก็ไม่ไช่อะไรอื่นนอกจากโปรแกรมภาษาโปรลอกนั่นเอง คือเป็นลำดับของความจริงและกฎ (Facts and Rules) แต่ในทางปฏิบัติแล้วก็ยังมีความแตกต่างบางประการระหว่างภาษาโปรลอกและระบบฐานข้อมูลอนุมาน กล่าวคือโปรแกรมภาษาโปรลอกมักจะประกอบด้วยกฎเป็นจำนวนมากขณะที่ใช้ความจริงเป็นจำนวนน้อย แต่สำหรับระบบฐานข้อมูลอนุมานนั้นจะเป็นตรงข้าม กล่าวคือจะประกอบด้วยความจริงเป็นจำนวนมาก ในขณะที่มีกฎเป็นจำนวนที่ไม่มากนัก และนั่นก็หมายความว่าเป็นไปไม่ได้ที่จะโหลดข้อมูลทั้งหมดลงในหน่วยความจำเพื่อประมวลผล ตัวโปรแกรมภาษาโปรลอกจะต้องมีขนาดเล็กเพียงพอที่จะโหลดลงในหน่วยความจำหลักได้เพื่อที่จะง่ายต่อการอ้างถึงโดย Interpreter

สำหรับระบบฐานข้อมูลอนุมานจะต้องเก็บความจริงในหน่วยความจำสำรอง เช่น ดิสก์ และโหลดเฉพาะข้อมูลที่จะใช้ในการหาคำตอบเท่านั้น ลงในหน่วยความจำหลักดังนั้นระบบฐานข้อมูลอนุมานจึงต้องมีความสามารถในการจัดการกับไฟล์อีกด้วย

เราได้กล่าวว่าระบบฐานข้อมูลแบบรีเลชันแนลเป็นกรณีพิเศษของระบบฐานข้อมูลอนุมาน เพื่อที่จะแสดงว่าคำกล่าวนั้นเป็นจริง เราจะอธิบายโดยใช้ตัวอย่าง Supplier-Part จากหนังสือ An Introduction to Database Systems [Date 1981] ฐานข้อมูลนี้มีความสัมพันธ์อยู่ 3 ชนิดได้แก่ S เป็นความสัมพันธ์ของ Supplier, P เป็นความสัมพันธ์ของ Part และ SP เป็นความสัมพันธ์ระหว่าง Supplier และ Part และเพื่อให้ชัดเจน เราจะเขียนความสัมพันธ์ในรูปแบบต่อไปนี้

S(sno,sname,status,city)

P(pno,pname,colour,weight,city)

SP(sno,pno,qty)

ในแต่ละความสัมพันธ์สามารถแทนด้วยตารางที่มีข้อมูล อย่างไรก็ตาม เราจะเขียนโดยใช้รูปแบบของภาษาโปรลอก ดังต่อไปนี้

| | |
|---|---|
| S(s1,smith,20,london) | SP(s1,p1) |
| S(s2,jones,10,paris) | SP(s1,p2) |
| S(s3,blake,30,paris) | SP(s1,p3) |
| S(s4,clark,20,london) | SP(s1,p4) |
| S(s5,adams,30,athens) | SP(s1,p5) |
| | SP(s1,p6) |
| P(p1,nut,red,12,london) | SP(s2,p1) |
| P(p2,bolt,green,17,paris) | SP(s2,p2) |
| P(p3,screw,blue,17,rome) | SP(s3,p2) |
| P(p4,screw,red,14,london) | SP(s4,p2) |
| P(p5,cam,blue,12,paris) | SP(s4,p4) |
| P(p6,cog,red,19,london) | SP(s4,p5) |

แต่ละ Tuple ของความสัมพันธ์จะเขียนในรูปแบบที่คล้ายกับความจริง (fact) ในภาษาโปรลอก ซึ่งในความเป็นจริงแล้วหากจะตัดความแตกต่างเล็กน้อยออกไป เราอาจกล่าวได้ว่าทั้ง Tuple และ fact เป็นสิ่งเดียวกัน และนั่นก็หมายความว่าเราอาจจะเรียกฐานข้อมูลแบบรีเลชันแนลว่าเป็นกรณีพิเศษของฐานข้อมูลอนุมานก็ย่อมได้ เพราะฐานข้อมูลอนุมานนั้นจะมีส่วนของกฎอีกด้วย ซึ่งส่วนของกฎนี่เองที่ทำให้สามารถใช้งานในลักษณะของภาษาได้อีกด้วย

ที่เราได้กล่าวผ่านมาทั้งหมด เราได้ใช้ความสัมพันธ์ในลักษณะของตารางในการแทนความจริง (facts) หรือความรู้ (Knowledge) ซึ่งการแทนในลักษณะดังกล่าวนั้น ยังมีข้อบกพร่องหรือปัญหาอยู่บางประการ ซึ่งต่อไปเราก็จะมาพิจารณาปัญหานั้น

### 4.4 การใช้ฐานข้อมูลแบบรีเลชันแนล (Relational Database) เป็นแบบแทนความรู้

แบบจำลองข้อมูลที่นิยมในระดับตรรก (logical Data Model) มีอยู่ 3 แบบ คือแบบรีเลชันแนล (Relational Model) แบบไฮราคี (Hierachical Model) และแบบโครงข่าย (Network Model) ซึ่งแบบที่กำลังเป็นที่นิยมและถูกสร้างขึ้นมาโดยทฤษฎีทางคณิตศาสตร์รองรับอย่างชัดเจน และมีความสัมพันธ์อย่างใกล้ชิดกับ First Order Predicate Logic (FOPL) คือแบบรีเลชันแนล นักวิทยาศาสตร์เป็นจำนวนมากที่เป็นผู้บุกเบิกงานทางด้านความสัมพันธ์ระหว่างงานทั้งสองสาขานี้

ในจำนวนนี้ Reiter ถึงกับนิยามฐานข้อมูลแบบรีเลชันแนลขึ้นใหม่ตามกฎเกณฑ์ทาง FOPL โดยจะมองคำถาม (Query) แต่ละคำถามที่ให้แก่ระบบฐานข้อมูล ถ้ามองทางทฤษฎีฐานข้อมูล จะมองเป็นโจทย์ที่ระบบจัดการฐานข้อมูลจะต้องค้นหาและดึงข้อมูลออกมาเป็นคำตอบ แต่ถ้ามองทางทฤษฎี FOPL จะมองคำถามเป็นทฤษฎี (Theorem) ที่จะต้องพิสูจน์ว่าจริงหรือไม่ จากความจริง (fact) ที่เก็บอยู่ในฐานข้อมูล ผู้ที่พิสูจน์ว่าทฤษฎีทั้งสองแนวทางนี้ต่างให้ผลลัพธ์ที่เหมือนกันภายใต้เงื่อนไขบางประการคือ ศาสตราจารย์ Kowalski แห่ง Imperial Collage London

อย่างไรก็ดีการใช้แบบจำลองข้อมูลรีเลชันแนลเป็นแบบแทนความรู้ในระบบปัญญาประดิษฐ์โดยตรงยังมีอุปสรรคอยู่บางประการ ซึ่งแม้แต่ในบทความของ Kowalski เองก็ยังไม่ได้ให้คำตอบที่จะเป็นแนวทางในการแก้ไขอุปสรรคดังกล่าว

เราจะมาพิจารณาอุปสรรคดังกล่าว สมมติว่าเรามีฐานข้อมูลซึ่งจะเก็บข้อมูลเกี่ยวกับนักศึกษาในมหาวิทยาลัยแห่งหนึ่ง โดยจะเก็บเลขประจำตัว ชื่อ ที่อยู่ หมายเลขโทรศัพท์ เพศ และชั้นปีที่ศึกษาอยู่ และให้แอทริบิวต์ (Attribute) ทุกตัวมีความสัมพันธ์อย่างเต็มที่ (Fully Functional Dependency) กับเลขประจำตัว (ID) เราจะได้โครงสร้างฐานข้อมูลแบบรีเลชันแนลที่อยู่ใน Fifth Normal Form [8] สำหรับเก็บข้อมูลเหล่านี้ดังรูปที่ 4-1 โดยมี ID เป็นคีย์หลัก (Primary Key)

สำหรับผู้ที่ใช้แบบจำลองรีเลชันแนล เป็นแบบแทนความรู้โดยตรงจะแทน n-ary รีเลชันด้วย n-ary พรีดิเขต ดังเช่นรีเลชันที่ชื่อ Student ในรูปที่ 4-1 ก็สามารถแทนด้วย n-ary พรีดิเขตชื่อ Student เช่นกัน โดยที่แต่ละแอทริบิวต์ของรีเลชันจะสัมพันธ์กับแต่ละอาร์กิวเมนต์ (Argument) ของพรีดิเขต

| Student_ID | Student_Name | Address | Telephone | Sex | Class |
|---|---|---|---|---|---|

*รูปที่ 4-1 ตัวอย่างการใช้ n-ary พรีดิเขตแทน n-ary รีเลชัน*

ถ้าฐานข้อมูลอนุมานของเราเก็บเฉพาะข้อเท็จจริง (Fact) คือเก็บเฉพาะส่วนที่เรียกว่า Extensional Database การแทน n-ary พรีดิเขตโดยตรงเช่นนี้จะไม่มีปัญหาอะไร แต่เมื่อไรก็ตามที่เราต้องการเก็บข้อมูลในรูปของ Production Rules หรือ Horn Clause ใน Intensional Database แล้วจะเกิดปัญหาขึ้นได้ ตัวอย่างเช่น ถ้าเราต้องการแสดงกฎที่ว่า นักศึกษาชั้นปีที่ 3 ทุกคนลงทะเบียนเรียนวิชา CS314 เราจะใช้ n-ary พรีดิเขตตามรูปที่ 4-1 มาแสดงกฎได้ดังนี้

Student (x,__,__,__,__,'year3') ---> Stu_subj(x,'CS314')

จะเห็นได้อย่างชัดเจนว่าการแทนความรู้ในลักษณะนี้ ไม่ใช่ทางเลือกที่ดีนักเพราะในแต่ละรีเลชันสามารถที่จะเก็บข้อเท็จจริงได้มากกว่า 1 อย่าง ข้อเท็จจริงที่ไม่เกี่ยวกับกฎจะถูกละไว้ แต่เนื่องจากแต่ละพรีดิเขตถูกกำหนดโดยรีเลชัน ดังนั้นผู้ใช้ต้องรู้จำนวนและตำแหน่งของทุกแอทริบิวต์ในแต่ละรีเลชัน ถึงแม้จะไม่เกี่ยวกับความจริงที่เป็นเงื่อนไขของกฎก็ตาม ผู้ใช้การแทนความรู้ในลักษณะนี้จะเสียความเป็นอิสระของข้อมูลในทางตรรกไปด้วยเนื่องจากพรีดิเขตผูกอยู่กับรีเลชัน จากปัญหาข้างต้นนี้ จึงได้นำเสนอการใช้แบบจำลองข้อมูลไนแอม มาเป็นแบบแทนความรู้ซึ่งสามารถที่จะแก้ปัญหานี้ได้โดยตรง

**4.5 แบบจำลองข้อมูลไนแอม (The NIAM Conceptual Schema Model)**

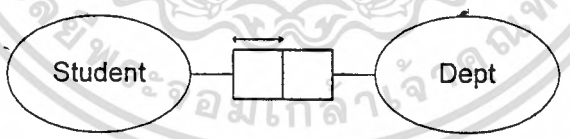
แบบจำลองข้อมูลไนแอมถูกคิดค้นขึ้นโดย Prof. G.M. Nijssen โดยไนแอมเป็นแบบจำลองข้อมูลระดับแนวคิด (Conceptual Model) ซึ่งมีพื้นฐานมาจากภาษาธรรมชาติแบบโครงสร้างลึก (Deep Structured Natural Language) คือภาษาที่มีรูปประโยคเป็น <ประธาน, กิริยา, กรรม> เท่านั้น

ไนแอมมีส่วนประกอบพื้นฐานดังต่อไปนี้

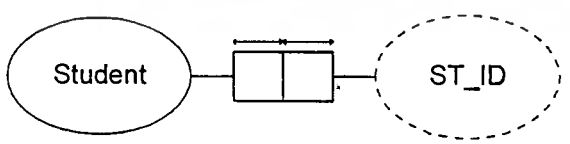
1. ชนิดเอนติตี้ (Entity Type)
2. ชนิดเลเบิ้ล (Label Type)
3. ชนิดความจริง (Fact Type)
4. ชนิดอ้างอิง (Reference Type)
5. ข้อจำกัดเพื่อความถูกต้อง (Integrity Constraint)

ส่วนประกอบพื้นฐานทั้ง 5 แสดงไว้ดังตัวอย่างในรูปที่ 4-2 โดยเอนติตี้เป็นหน่วยพื้นฐานที่สุดของระบบข้อมูล เช่น ในมหาวิยาลัย นักศึกษา อาจารย์ ฯลฯ ล้วนจัดเป็นเอนติตี้ ชนิดเอนติตี้เป็นเซ็ต (Set) ซึ่งมีสมาชิกเป็นตัวอย่างเอนติตี้ (Entity Instance) แต่ละตัวอย่างเอนติตี้จำเป็นที่จะต้องมีชื่อเรียกซึ่งอาจเป็นชื่อ รหัส อย่างใดอย่างหนึ่ง การกำหนดว่าแต่ละชนิดเอนติตี้ควรใช้อะไรเป็นชื่อเรียกทำได้โดยระบุชนิดเลเบิ้ล สำหรับเอนติตี้นั้น

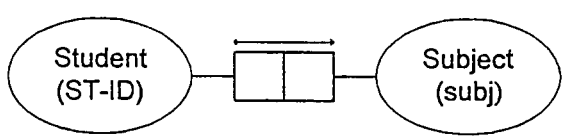

A : An entity and label type in NIAM




B : A one - to - many Fact Type



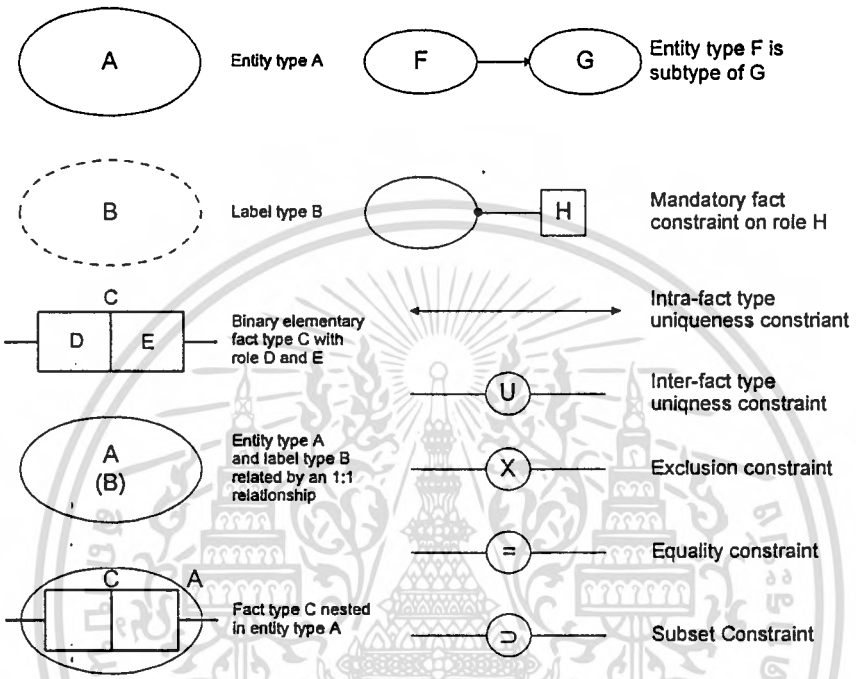

C : A one - to - one reference type



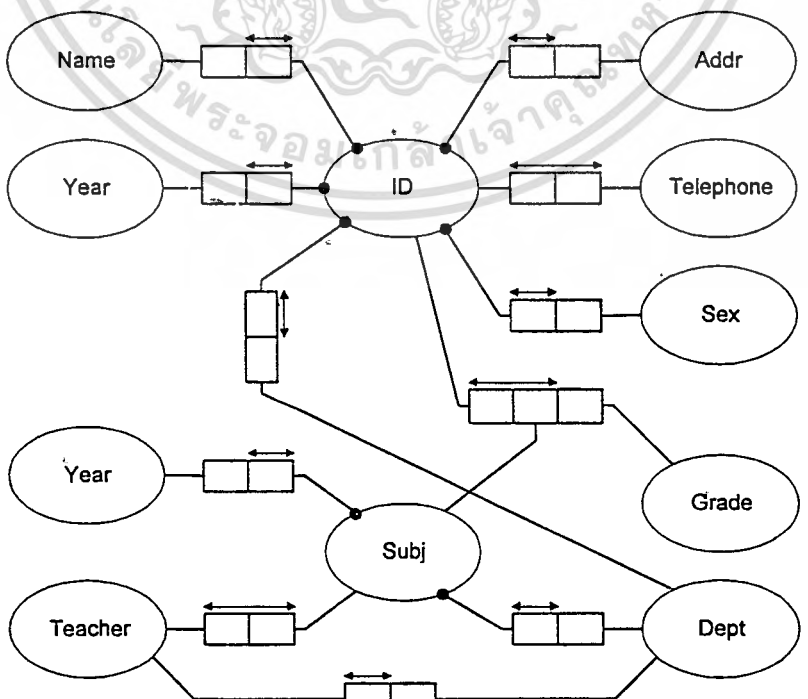

D : A many to many Fact Type

***รูปที่ 4-2 ตัวอย่างชนิดเอนติตี้, ชนิดเลเบิ้ล, ชนิดความจริงและชนิดอ้างอิง***

ตัวอย่างเช่นรูปที่ 4-2C ชนิดเอนติตี้ Student ใช้ชนิดเลเบิ้ล St_id หรือเลขประจำตัวเป็นชื่อเรียก ความสัมพันธ์ระหว่างชนิดเอนติตี้กับชนิดเลเบิ้ลเรียกว่า "ชนิดอ้างอิง" ส่วนความสัมพันธ์ระหว่างชนิดเอนติตี้กับชนิดเอนติตี้ด้วยกันเรียกว่า "ชนิดความจริง" รูปที่ 4-2C แสดงชนิดอ้างอิงแบบ 1:1 ซึ่งมีความหมายว่า นักศึกษาแต่ละคนจะมีเลขประจำตัวได้ 1 เลขเท่านั้น




*รูปที่ 4-3 สัญญลักษณ์ที่ใช้ในแบบจำลองข้อมูลไนแอม*




*รูปที่ 4-4 ตัวอย่างการใช้ไนแอมจำลองข้อมูล*

รูปที่ 4-2B แสดงชนิดความจริงชนิด 1:n ซึ่งมีความหมายว่านักศึกษาแต่ละคนสังกัดได้เพียง 1 ภาควิชาแต่ใน 1 ภาควิชาสามารถมีนักศึกษาได้หลายคน รูปที่ 4-2D แสดงชนิดความจริงแบบ m:n ซึ่งมีความหมายว่านักศึกษาแต่ละคนสามารถเรียนได้หลายวิชา และแต่ละวิชาก็มีนักศึกษาได้หลายคุนสัญญลักษณ์ของไนแอมแสดงได้ในรูปที่ 4-3 และตัวอย่างของการใช้แบบจำลองข้อมูลไนแอม แสดงไว้ในดังรูปที่ 4-4

**4.5.1 การแทนชนิดความจริง (Fact Type) ด้วยพรีดิเขต**

จากรูปที่ 4 จะเห็นได้ว่ามีชนิดความจริงอยู่ 11 ชนิดคือ

F1 (ID#, Year)
F2 (ID#, Name)
F3 (ID#, Address)
F4 (ID#, Sex)
F5 (ID#, Telephone)
F6 (ID#, Dept)
F7 (ID#, Subj#, Grade)
F8 (Subj#, Dept)
F9 (Teacher, Dept)
F10 (Subj#, Teacher)
F11 (Subj#, Subj_Name)

แต่ละชนิดความจริงเป็นความจริงพื้นฐานซึ่งไม่สามารถแบ่งย่อยลงไปได้อีกแล้ว จึงเป็นการเหมาะสมมากที่จะใช้เป็นแบบแทนความรู้สำหรับแทนพรีดิเขต นอกจากนั้นแบบจำลองข้อมูล NIAM ยังมีลักษณะเป็นโครงสร้าง สามารถแสดงความสัมพันธ์ของแต่ละชนิดเอนติตี้ และแสดงความสัมพันธ์ระหว่างชนิดเอนติตี้หลักและชนิดเอนติตี้ย่อย (Subtype Hierachy) นั่นคือสามารถแสดงความสัมพันธ์ได้เช่นเดียวกับ Semantic Network นอกจากนั้น NIAM ยังสามารถแปรรูปมาเป็นรีเลชันที่เป็น Fifth Normal Form ได้เลย ในขณะที่ Semantic Network ไม่มีคุณสมบัติเช่นนั้น

และจากที่ได้กล่าวมาทั้งหมด คงจะทำให้ผู้อ่านได้เข้าใจถึงความสัมพันธ์ระหว่างระบบฐานข้อมูลรีเลชันแนล และระบบฐานข้อมูลอนุมาน และคงจะได้เข้าใจว่าเหตุใดเมื่อเราต้องการสร้างระบบฐานข้อมูลอนุมานขึ้นมา เราจึงไม่นำแบบจำลองข้อมูลในแบบรีเลชันแนลมาใช้งานเป็นแบบแทนความรู้ แต่กลับไปนำเอาแบบจำลองข้อมูลมาใช้งานแทน สำหรับในบทต่อ ๆ ไปนั้นท่านผู้อ่านก็จะได้เห็นประโยชน์ของการนำเอาแบบจำลองข้อมูลไนแอมไปใช้เป็นแบบแทนความรู้ได้มากขึ้น และได้เห็นถึงความเหมาะสมของการใช้แบบจำลองข้อมูลไนแอมเป็นแบบแทนความรู้ในฐานข้อมูลอนุมาน

# บทที่ 5

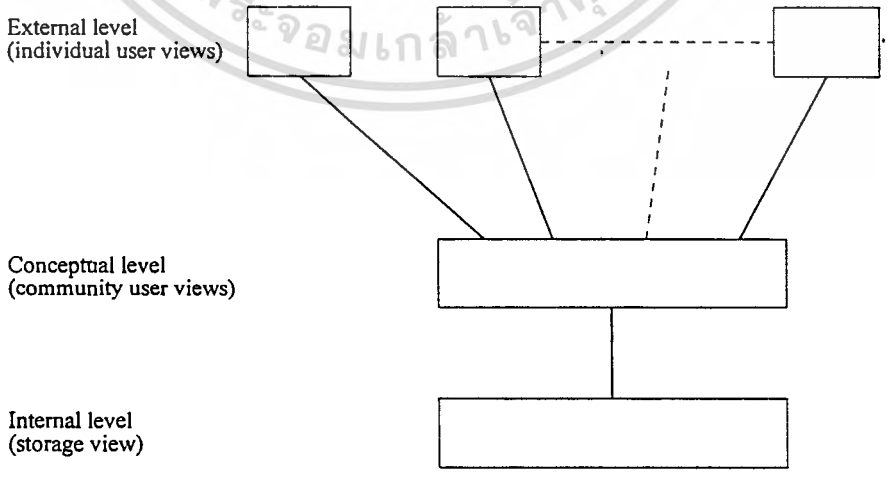
# สถาปัตยกรรมของระบบ

ในบทที่ผ่านมาเราได้ศึกษาถึงทฤษฎีต่าง ๆ ที่จำเป็นต่อการเรียนรู้และทำความเข้าใจกับงานวิจัยชิ้นนี้ไปหมดแล้ว เราได้ทราบแล้วว่าแบบจำลองข้อมูลไนแอมมีข้อที่ดีกว่าแบบจำลองข้อมูลแบบรีเลชันแนลอย่างไร เราได้รู้จักกับระบบฐานข้อมูลอนุมาน และได้ทราบถึงข้อดีของความสามารถในการอนุมานกฎในฐานข้อมูลอนุมาน และสำหรับในบทนี้และในบทต่อ ๆ ไป จะเป็นส่วนของงานวิจัยที่ได้ทำไว้ โดยจะแยกอธิบายเป็นส่วน ๆ ไป แต่สำหรับในบทนี้เราจะกล่าวถึงสถาปัตยกรรมโดยรวมของระบบ

การออกแบบสถาปัตยกรรมของระบบ นับเป็นสิ่งที่จำเป็นในการสร้างอะไรขึ้นมาสักอย่างหนึ่ง ไม่ว่าจะเป็นการสร้างบ้าน สร้างสิ่งประดิษฐ์ต่าง ๆ และในการทำโครงงานก็เช่นกัน การออกแบบสถาปัตยกรรมที่ดี ก็จะเปรียบเสมือนกับการมีแผนผังการทำงานที่ดี ทำให้เราสามารถแบ่งการทำงานออกเป็นส่วน ๆ เพื่อทำตามลำดับก่อนหลังได้อย่างถูกต้อง ซึ่งจะทำให้งานสำเร็จตามเป้าหมายได้อย่างรวดเร็ว

## 5.1 สถาปัตยกรรมมาตรฐานของ ANSI และ ISO

การออกแบบสถาปัตยกรรมรวมของระบบฐานข้อมูลอนุมานของเรา เราจะเริ่มต้นด้วยการศึกษาสถาปัตยกรรมต่าง ๆ ที่เกี่ยวข้อง โดยจะเริ่มจากสถาปัตยกรรมมาตรฐานของระบบฐานข้อมูลทั่วไป สถาปัตยกรรมในรูปที่ 5-1 นี้เป็นสถาปัตยกรรมฐานข้อมูลมาตรฐานของ ANSI และ ISO ซึ่งเป็นสถาปัตยกรรมที่ได้รับการยอมรับและนำไปใช้อย่างกว้างขวาง และเนื่องจากระบบฐานข้อมูลอนุมานของเราก็เป็นฐานข้อมูลแบบหนึ่ง ดังนั้นเราก็จะนำสถาปัตยกรรมนี้มาใช้เป็นพื้นฐานของการออกแบบด้วยเช่นกัน

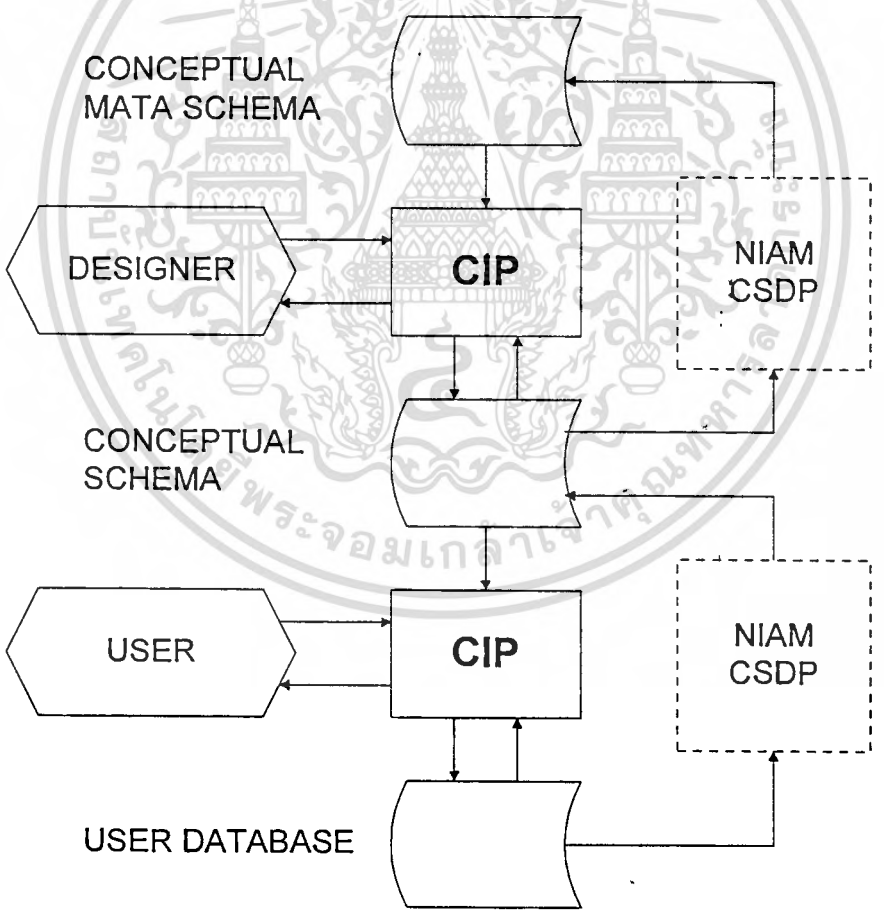



*รูปที่ 5-1 แสดงสถาปัตยกรรมมาตรฐานของ ANSI และ ISO*

สถาปัตยกรรมในรูปที่ 5-1 นี้จะแบ่งระดับของรูปแบบข้อมูลและการเข้าถึงข้อมูลออกเป็น 3 ระดับด้วยกัน คือ ระดับภายนอก (External) ซึ่งจะเก็บข้อมูลการมองเห็นและเข้าถึงข้อมูลของผู้ใช้แต่ละคนไว้ ระดับแนวคิด (Conceptual) จะเก็บรูปแบบข้อมูลที่แทนด้วยแบบจำลองข้อมูล และระดับภายใน (Internal) ซึ่งจะเก็บรูปแบบการเก็บข้อมูลภายในฐานข้อมูลนั้น ๆ

**5.2 สถาปัตยกรรม CIP**

สถาปัตยกรรมอีกแบบหนึ่งที่เราจะใช้เป็นพื้นฐานในการออกแบบสถาปัตยกรรมของระบบก็คือ สถาปัตยกรรม CIP (Conceptual Information Processor) โดยในสถาปัตยกรรมนี้จะเน้นที่การออกแบบแบบจำลองข้อมูล โดยแบ่งระดับของการควบคุมข้อมูลออกเป็น 3 ระดับ (ดูรูปที่ 5-2) โดยข้อมูลในระดับบนสุดจะเรียกว่า Conceptual Meta Schema ซึ่งเป็นข้อมูลข้อกำหนดในการสร้าง Conceptual Schema โดยหากว่าเราใช้แบบจำลองข้อมูลในแอมเป็น Conceptual Schema แล้ว ข้อมูลที่เก็บอยู่ใน Conceptual Meta Schema ก็คือ รูปแบบและข้อบังคับในการสร้างแบบจำลองข้อมูลในแอมนั่นเอง




*รูปที่ 5-2 สถาปัตยกรรมแบบ CIP*

ข้อมูลที่เก็บอยู่ใน Conceptual Meta Schema จะถูกใช้โดย CIP (ตัวบน) โดย CIP จะเป็นส่วนที่ทำหน้าที่ติดต่อกับผู้ออกแบบ Conceptual Schema ผู้ออกแบบจะออกแบบแบบจำลองข้อมูลในระดับแนว

คิดไปเรื่อย ๆ ภายใต้ข้อจำกัดของ Conceptual Meta Schema โดยแบบจำลองข้อมูลที่ออกแบบแล้วก็จะเก็บไว้ที่ Meta Database เพื่อใช้ในการควบคุมข้อมูลจริงต่อไป

สำหรับการทำงานของ CIP (ตัวล่าง) ก็เช่นเดียวกัน โดย CIP จะทำหน้าที่ในการติดต่อกับผู้ใช้ฐานข้อมูล โดยข้อมูลที่รับมาจากผู้ใช้นั้นจะได้รับการควบคุมจาก Conceptual Schema ซึ่งทำหน้าที่ควบคุมรูปแบบและควบคุมความถูกต้องของข้อมูล โดยหากข้อมูลไม่มีความขัดแย้งกับ Conceptual Schema ก็จะนำไปจัดเก็บลงใน User Database ต่อไป

สำหรับที่เห็นเป็นเส้นประนั้นเป็นเพียงการแสดงให้เห็นว่าการออกแบบ Conceptual Schema นั้นจะใช้ข้อมูลตัวอย่างจาก User Database มาผ่านกระบวนการออกแบบ (CSDP) จำนวน 9 ขั้นตอนตามที่ได้กล่าวไปแล้ว และ Conceptual Meta Schema ก็จะได้มาจากการนำเอาข้อมูลตัวอย่างจาก Conceptual Schema มาผ่านขั้นตอนการออกแบบ 9 ขั้นตอน แล้วจึงได้ผลลัพธ์มาเป็น Conceptual Meta Schema

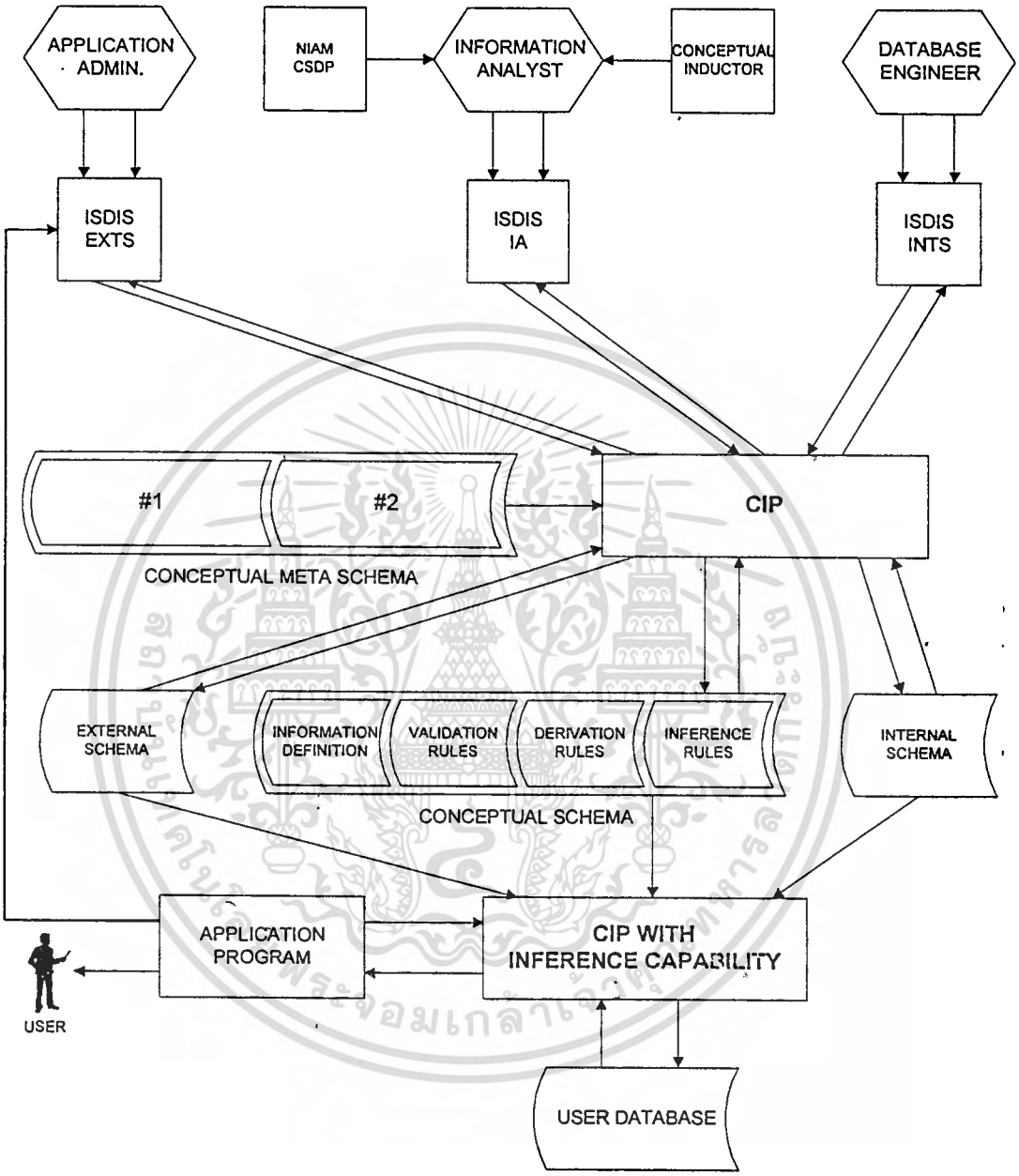
### 5.3 สถาปัตยกรรมของระบบ

หลังจากที่เราได้ศึกษาสถาปัตยกรรมที่เกี่ยวข้องทั้งสองนั้น เราก็ได้แนวคิดที่ได้มาทำการออกแบบสถาปัตยกรรมของระบบของเราต่อไป ซึ่งผลที่ได้จากการออกแบบการทำงานของระบบก็ได้ผลลัพธ์ออกมาดังรูปที่ 5-3 องค์ประกอบต่าง ๆ ในรูปนั้น แม้ว่าจะค่อนข้างซับซ้อนแต่เมื่อพิจารณาให้ดีแล้ว ก็จะเห็นได้ว่ามีรากฐานมาจากสถาปัตยกรรม ANSI, ISO และ CIP อย่างชัดเจน ซึ่งต่อไปก็จะขออธิบายถึงรายละเอียดส่วนต่าง ๆ ของสถาปัตยกรรมที่เราสร้างขึ้น

ในแผนภาพนั้นสัญญลักษณ์รูปหกเหลี่ยมจะแทนกระบวนการที่ผู้ใช้เป็นผู้ปฏิบัติ และสี่เหลี่ยมจะแทนการทำงานของระบบ ซึ่งโดยทั่วไปจะอยู่ในรูปของโปรแกรม ในส่วนบนสุดของแผนภาพนั้นจะเห็นว่ามีกระบวนการอยู่ 3 กระบวนการด้วยกัน คือ กระบวนการที่ทำโดย Application Administrator กระบวนการนี้เป็นการบวนการออกแบบส่วนควบคุมการติดต่อระหว่างผู้ใช้กับ Conceptual Schema โดยเป็นตัวกำหนดว่าจะใช้ผู้ใช้คนใด หรือแอปพลิเคชันใด จะเข้ามาใช้งานข้อมูลในส่วนต่าง ๆ ของ Conceptual Schema ได้บ้าง การทำงานส่วนนี้จะมี ISDIS (Information System Design and Implementation Systems) ในส่วน External เป็นผู้ดูแล

กระบวนการที่ทำโดย Information Analyst นั้น จะเป็นกระบวนการออกแบบข้อมูลในส่วนของ Conceptual Schema ซึ่งจะทำหน้าที่ควบคุมรูปแบบการเก็บข้อมูลในฐานข้อมูลของผู้ใช้ ผู้ที่ทำหน้าที่นี้จะใช้ขั้นตอน CSDP ที่ได้กล่าวไว้ในบทที่ 3 เป็นเครื่องมือที่ช่วยในการออกแบบ และใช้ความรู้พื้นฐานเกี่ยวกับการอนุมานกฎ (Conceptual Induction) เข้ามาช่วยในการออกแบบด้วย การทำงานส่วนนี้จะรองรับโดย ISDIS ในส่วน Information Analysis

สำหรับกระบวนการที่ทำโดย Database Engineer นั้นจะเป็นการออกแบบการจัดเก็บข้อมูลของ Conceptual Schema ลงในแหล่งเก็บข้อมูล โดยอาจจะเป็นฐานข้อมูล หรืออาจจะเป็นเพียงไฟล์ข้อมูลก็ได้ การทำงานในส่วนนี้จะรองรับโดย ISDIS ในส่วน Internal




CIP = CONCEPTUAL INFORMATION PROCESSOR
#1 = CONCEPTUAL SCHEMA FOR STRUCTURE VALIDATION DERIVATION SECTION
#2 = CONCEPTUAL SCHEMA FOR INFERENCE SECTION
ISDIS = INFORMATION SYSTEMS DESIGN AND IMPLEMENTATION SYSTEMS

*รูปที่ 5-3 แสดงสถาปัตยกรรมของระบบฐานข้อมูลแบบอนุมาน*

การทำงานของ ISDIS ทั้ง 3 ส่วนจะรองรับโดย CIP (Conceptual Information Processor) โดย CIP จะทำหน้าที่ควบคุมรูปแบบและความถูกต้องของข้อมูล Conceptual Schema ซึ่งถูกระบุโดยรูปแบบและข้อบังคับที่กำหนดไว้ใน Conceptual Meta Schema (ต่อไปจะเรียกว่า Metaschema) ข้อมูลที่เก็บไว้

ใน Metaschema นั้นจะมีอยู่ 2 ส่วนด้วยกัน โดยส่วนแรกจะเป็นรูปแบบข้อมูลและข้อบังคับต่าง ๆ สำหรับข้อมูลทั่ว ๆ ไป และส่วนที่ 2 จะเป็นส่วนที่ระบุถึงการเก็บข้อมูลและข้อบังคับสำหรับข้อมูลที่มีลักษณะเป็น กฎ ซึ่งข้อมูลที่ไม่ขัดแย้งกับ Metaschema ก็จะถูกส่งไปเก็บใน Conceptual Schema ต่อไป

ข้อมูลที่เก็บใน Conceptual Schema นั้นจะแบ่งออกเป็น 4 ส่วนด้วยกัน โดยส่วนแรกจะระบุถึงรูปแบบข้อมูลที่จัดเก็บและความสัมพันธ์ของข้อมูลนั้น (Relation Definition) ส่วนที่สองจะระบุถึงกฎข้อบังคับต่าง ๆ (Validation Rules) ส่วนที่สามจะระบุที่จะเก็บในลักษณะของกฎ คือ ไม่จำเป็นต้องเก็บข้อมูลนั้นไว้จริง ๆ เช่น สำหรับข้อมูลอายุนั้น หากรู้วันเกิดก็ไม่จำเป็นต้องเก็บข้อมูลอายุ เพราะใช้วิธีคำนวณเอาได้ ข้อมูลในลักษณะนี้เราจะเรียกว่าข้อมูลแบบหาได้ (Derived) สำหรับข้อมูลส่วนสุดท้ายก็จะเป็นข้อมูลที่เป็นกฎในลักษณะที่เป็น Inference Rules ซึ่งเป็นกฎในเชิงตรรก ซึ่งจะได้กล่าวต่อไป

ในส่วนของ CIP ที่ด้านล่างนั้น จะทำหน้าที่ติดต่อกับผู้ใช้หรือติดต่อกับโปรแกรมแอปพลิเคชัน โดยจะทำหน้าที่รับข้อมูล ลบข้อมูล ค้นหาข้อมูลตามเงื่อนไขที่กำหนด นอกจากนั้นก็ยังทำหน้าที่ควบคุมความถูกต้องของข้อมูลอีกด้วย โดยการทำงานทุก ๆ อย่างจะอยู่ภายใต้การควบคุมของข้อมูลที่ถูกจัดเก็บไว้ใน Conceptual Schema และข้อมูลจะเก็บไว้ในฐานข้อมูลผู้ใช้ (User Database)

**5.4 รายละเอียดของงานวิจัย**

จากที่ได้กล่าวไปทั้งหมดนั้นเป็นสถาปัตยกรรมรวมทั้งระบบ ซึ่งจะเห็นได้อย่างชัดเจนว่าแบ่งการทำงานออกเป็น 2 ส่วนอย่างชัดเจน คือ ส่วนที่เกี่ยวข้องกับการออกแบบ Conceptual Schema ซึ่งก็คือ CIP ตัวบนแล้วส่วนประกอบต่าง ๆ ที่อยู่เหนือ CIP ตัวบนทั้งหมด และส่วนที่เกี่ยวข้องกับการนำ Conceptual Schema มาใช้งาน ซึ่งก็ได้แก่ CIP ตัวล่าง โดยมี Conceptual Schema เป็นตัวเชื่อมการทำงานของทั้ง 2 ส่วน ดังนั้นจะเห็นได้ว่าการทำงานของทั้งสองส่วนนั้น จะมีการทำงานที่เป็นอิสระต่อกัน

สำหรับในงานวิจัยนี้จะเป็นการสร้าง CIP ตัวล่างขึ้นมา เพื่อจัดการกับข้อมูลที่จะเข้าและออกจากฐานข้อมูล รวมทั้งประมวลผลคำถามที่เป็นทั้งคำถามทั่วไป และคำถามในแบบที่เป็นกฎเกณฑ์ทางตรรกด้วย โดยจะถือว่าข้อมูลในส่วน Conceptual Schema นั้นถูกจัดเตรียมไว้เรียบร้อยในฐานข้อมูล Conceptual Schema แล้ว ดังนั้นความเกี่ยวข้องที่เกี่ยวกับ Conceptual Schema นั้นก็เป็นเพียงการนำข้อมูลที่เก็บอยู่มาใช้งานเท่านั้น ดังนั้นต่อไปเราจะกล่าวถึงรายละเอียดของ CIP ซึ่งในโครงงานนี้จะถือว่าเป็นตัวประมวลผลของระบบฐานข้อมูลแบบอนุมาน

โปรแกรมทั้งหมดที่เราจะสร้างขึ้นเพื่อทำหน้าที่เป็น CIP นี้ จะสร้างซ้อนอยู่บนระบบฐานข้อมูลแบบรีเลชันแนลอีกที นั่นคือแทนที่ CIP จะจัดการกับข้อมูลในฐานข้อมูลผู้ใช้เอง ก็จะส่งผ่านไปยังฐานข้อมูลแบบรีเลชันแนลให้จัดการกับข้อมูลให้ โดยการติดต่อกับระบบฐานข้อมูลแบบรีเลชันแนลทั้งหมดจะกระทำโดยผ่านทางภาษา SQL และระบบฐานข้อมูลที่เราเลือกใช้ก็คือ ORACLE เพราะมีความสามารถสูงและทำงานบนเครื่องพีซีได้ดี

สำหรับองค์ประกอบของ CIP (Conceptual Information Processor) จะมีดังต่อไปนี้

**5.4.1 ส่วนนิยามข้อมูล ( INFORMATION DEFINITION )**

เนื่องจากระบบงานของเราจะสร้างครอบอยู่บนระบบฐานข้อมูลแบบรีเลชันแนลอีกที ดังนั้น ข้อมูลต่าง ๆ ที่เก็บอยู่ใน Conceptual Schema และ User Database ก็จะมีรูปแบบที่เป็นตาราง ซึ่งประกอบด้วยแถวและคอลัมน์ไปด้วย และการทำงานในส่วนนี้ก็จะเป็นการแปลงข้อมูลที่อยู่ในรูปของความสัมพันธ์แบบตาราง ให้กลายมาเป็นข้อมูลที่มีความสัมพันธ์ในลักษณะของชนิดความจริง ซึ่งจะเป็นความสัมพันธ์พื้นฐานที่ใช้ในระบบงานของเรา

### 5.4.2. ส่วนจัดการข้อมูล ( INFORMATION MANIPULATION )

ในการติดต่อกับระบบฐานข้อมูลอนุมานนั้น จะมีรูปแบบที่ไม่เหมือนกับการใช้งานระบบฐานข้อมูลแบบรีเลชันแนล เพราะจากที่ได้กล่าวมาแล้วว่าระบบฐานข้อมูลอนุมาน จะมีรูปแบบที่ใกล้เคียงกับภาษาโปรลอก ดังนั้นการติดต่อกับฐานข้อมูลอนุมานที่เราสร้างขึ้น จึงได้ออกแบบให้มีการติดต่อที่คล้ายกับภาษาโปรลอกไปด้วย โดยข้อมูลจะเข้าและออกโดยผ่านทางคำสั่ง Assert และ Remove เท่านั้น

ทั้งนี้เพราะข้อมูลที่เก็บอยู่ในรูปแบบชนิดความจริงนั้น จะเป็นความจริงพื้นฐานที่ไม่สามารถแบ่งลงไปได้อีกแล้ว ดังนั้นข้อมูลจะมีเพียง 2 สถานะเท่านั้น คือ มีชนิดความจริงนั้นอยู่หรือไม่มีชนิดความจริงนั้น ดังนั้นคำสั่งที่นำข้อมูลเข้าและออกจึงกำหนดให้มี 2 คำสั่งเท่านั้นโดยคำสั่ง Assert จะมีความหมายของการนำชนิดความจริงเข้าไปเก็บในฐานข้อมูล และ Remove จะมีความหมายถึงการนำชนิดความจริงออกจากฐานข้อมูล

นอกจากนั้นการทำงานในส่วนนี้ ยังครอบคลุมไปถึงการตรวจสอบความถูกต้อง (Constraint Checking) ด้วย เพราะในการนำข้อมูลเข้าหรือออกแต่ละครั้งนั้น อาจจะทำให้เกิดความขัดแย้งตามกฎข้อบังคับที่กำหนดไว้ใน Conceptual Schema ได้ ซึ่ง CIP จะต้องคอยรักษาสถานะความถูกต้องของฐานข้อมูลไว้เสมอ ดังนั้นทุก ๆ ครั้งที่มีการนำข้อมูลเข้าหรือออกจากระบบ ก็จำเป็นจะต้องมีการตรวจสอบความถูกต้องของข้อมูลด้วย

### 5.4.3. ส่วนประมวลผลคำถาม (QUERY PROCESSOR)

การทำงานในส่วนนี้อาจจะถือได้ว่าเป็นการทำงานที่สำคัญที่สุดในฐานข้อมูลอนุมาน ข้อมูลที่อยู่ในฐานข้อมูลคงไม่มีประโยชน์อะไร หากไม่สามารถประมวลผลในรูปของคำถามได้ โดยคำถามที่ระบบฐานข้อมูลอนุมานของเราสามารถรับได้นั้น จะมีอยู่ 3 รูปแบบด้วยกัน โดยรูปแบบที่ 1 จะเป็นคำถามธรรมดาที่สามารถดึงออกจากข้อมูลที่เก็บอยู่ในฐานข้อมูลโดยตรงได้ รูปแบบที่ 2 จะเป็นคำถามที่สามารถดึงออกจากฐานข้อมูลโดยการประมวลผลผ่านกฎแบบง่าย ๆ ที่ได้กำหนดไว้ สำหรับรูปแบบที่ 3 นั้นเป็นคำถามที่ต้องประมวลผลโดยอาศัยการประมวลผลทางตรรกก่อน จึงจะสามารถตอบคำถามได้ ซึ่งรูปแบบคำถามแบบนี้เอง ที่นับได้ว่าเป็นที่มาของ "ระบบฐานข้อมูลอนุมาน" เพราะก่อนจะตอบคำถาม จะต้องอนุมานคำตอบจากกฎที่กำหนดไว้เสียก่อนนั่นเอง

ซึ่งรายละเอียดการทำงานของส่วนต่าง ๆ จะได้กล่าวถึงในบทต่อ ๆ ไป

# บทที่ 6

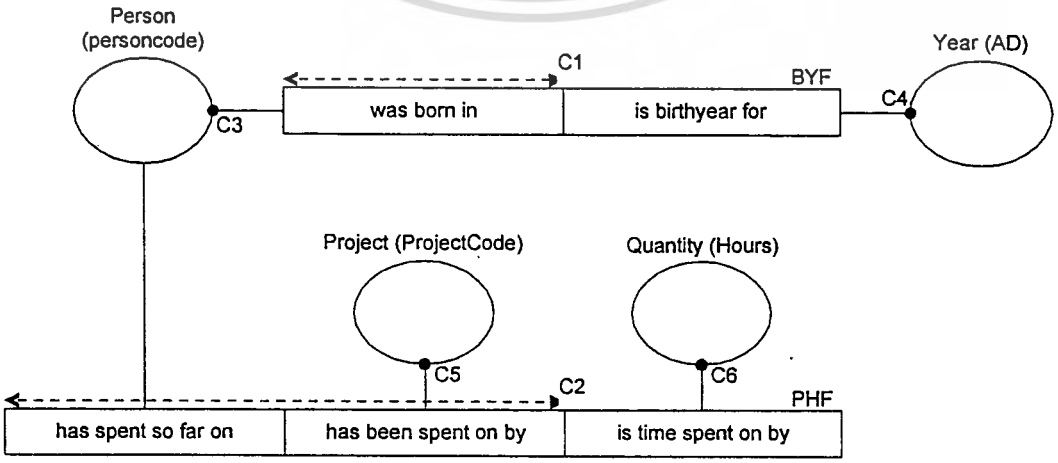
# การออกแบบ Conceptual Meta Schema

ในบทที่ 5 เราได้กล่าวถึงสถาปัตยกรรมของระบบไปแล้ว และจากแผนภูมิของแบบจำลองข้อมูลก็จะเห็นได้ว่าจุดที่ถือว่าเป็นหัวใจของสถาปัตยกรรมก็คือ Conceptual Meta Schema ซึ่งเป็นแหล่งที่เก็บข้อมูลที่ระบุถึงรูปแบบการเก็บข้อมูลของ Conceptual Schema ซึ่งเป็นข้อมูลที่เราจำเป็นต้องใช้ในงานของเรา ดังนั้นงานขั้นต่อไปของเราก็คือ การออกแบบ Conceptual Meta Schema เพื่อจะใช้เก็บโครงสร้างของ Conceptual Schema ต่อไป

## 6.1 วิธีการออกแบบ Conceptual Meta Schema

ในการออกแบบ Cónceptual Meta Schema นั้น เราก็จะใช้หลักการ 9 ขั้นตอนที่ได้กล่าวไว้ในบทที่ 3 ช่วยในการออกแบบเช่นกัน แต่เนื่องจากขั้นตอนในการออกแบบที่ค่อนข้างซับซ้อน อีกทั้งข้อมูลที่นำมาใช้ในการออกแบบก็ค่อนข้างมาก จึงไม่สามารถแสดงรายละเอียดในการออกแบบได้ครบถ้วน เพราะจะทำให้เนื้อหาในบทนี้ยาวมากเกินไป และที่สำคัญก็คือการออกแบบ Conceptual Meta Schema ไม่ได้เป็นเป้าหมายหลักของวิทยานิพนธ์ฉบับนี้ เป็นแต่เพียงส่วนประกอบเท่านั้น ดังนั้นจะขอถึงกล่าววิธีการออกแบบอย่างคร่าว ๆ เท่านั้น

ในการออกแบบ Conceptual Meta Schema เราจะเริ่มด้วยการกำหนดตัวอย่างแบบจำลองข้อมูลไนแอมแบบง่าย ๆ ที่ไม่ซับซ้อนมากนัก โดยจะมีเอนติตี้เพียง 3-4 เอนติตี้เท่านั้น โดยกำหนดให้แต่ละเอนติตี้มีชนิดอ้างอิงเพียงตัวเดียวและมีความสัมพันธ์กับชนิดอ้างอิงแบบ 1:1 เท่านั้น และกำหนดให้ชนิดความจริงมีความสัมพันธ์ทั้งแบบ one-to-one และ many-to-many ในแบบไบนารีและเทอนารี โดยมีคอนสแตนต์แค่ Uniqueness Constraint และ Mendatory Constraint เท่านั้น ดังตัวอย่างในรูปที่ 6-1



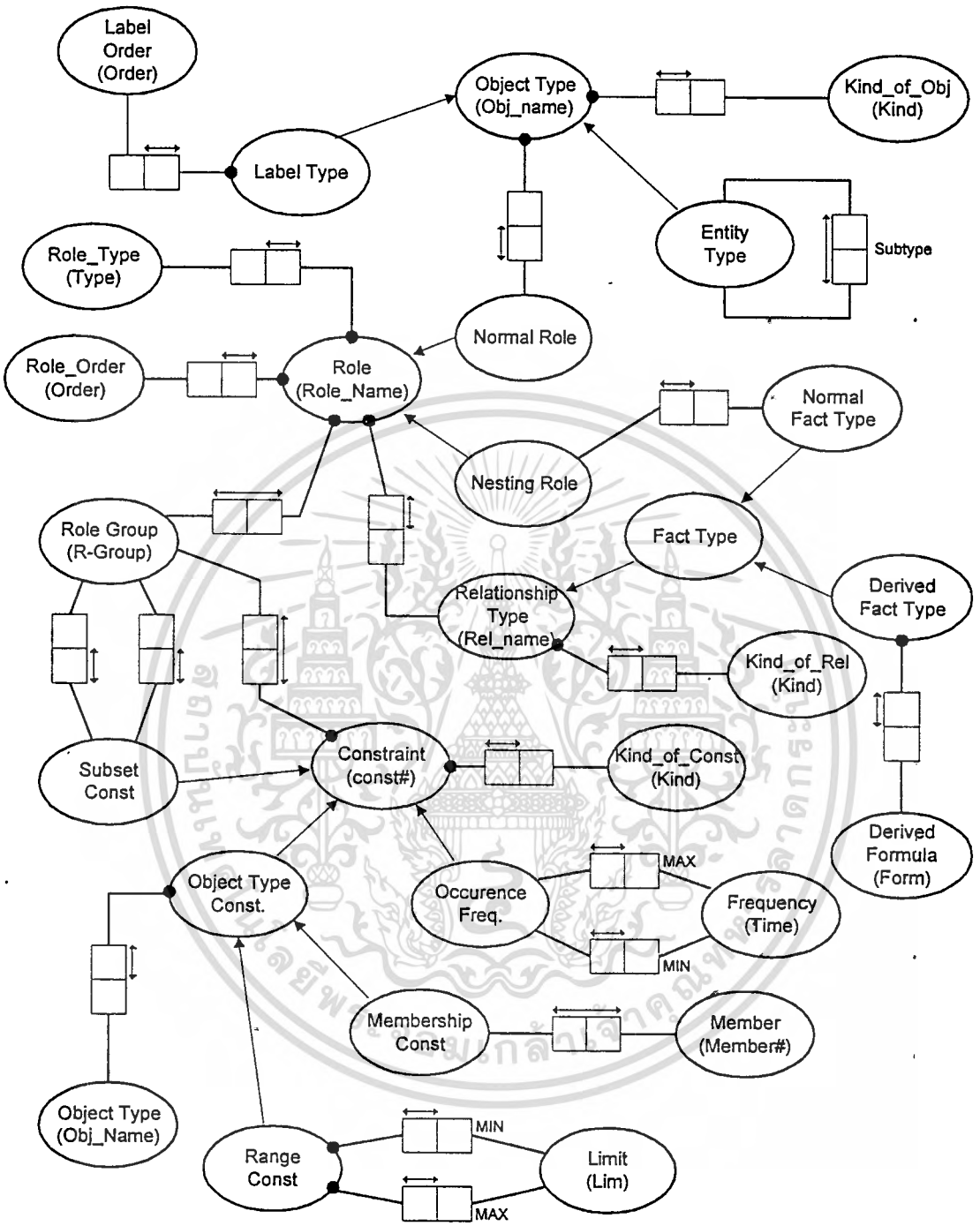

*รูปที่ 6-1 ตัวอย่างแบบจำลองข้อมูลที่ใช้ในการสร้าง Conceptual Meta Schema*

จากตัวอย่างข้อมูลในรูปที่ 6-1 เราก็นำมาเขียนในรูปของข้อความตามรูปที่ 6-2 ซึ่งอาจกล่าวได้ว่าเป็น Conceptual Schema ที่อยู่ในรูปแบบข้อความ สำหรับรูปแบบข้อความที่นำมาใช้งานนี้ ไม่ได้หมายความว่าจำเป็นต้องใช้ในรูปแบบเช่นนี้เสมอไป โดยอาจจะใช้รูปแบบอื่นก็ได้เช่นกัน ซึ่งเราจะถือว่าข้อมูลนี้เป็น Elementary Fact และนำข้อมูลนี้มาเขียนแบบจำลองข้อมูลเพื่อเก็บข้อมูลตัวอย่าง

```
ENTITY TYPE NAME IS "PERSON", LABEL TYPE IS "PERSONCODE"
        "YEAR"          "AD"
        "PROJECT"               "PROJECTCODE"
        "QUANTITY"              "HOURS"

FACT TYPE NAME IS "BYF"
        ROLE NAME IS "WAS BORN IN", REFERRING TO "PERSON"
        ROLE NAME IS "IS BIRTHYEAR FOR", REFERING TO YEAR"

FACT TYPE NAME IS "PHF"
        ROLE NAME IS "HAS SPENT SO FAR ON", REFERING TO "PERSON"
        ROLE NAME IS "HAS BEEN SPENT ON BY", REFERING TO "PROJECT"
        ROLE NAME IS "IS TIME SPENT ON BY", REFERING TO "QUANTITY"

CONSTRAINT C1:
        ROLE "WAS BORN IN" IS UNIQUE

CONSTRAINT C2:
        ROLE "HAS SPENT SO FAR ON", "HAS BEEN SPENT ON BY" IS UNIQUE

CONSTRAINT C3:
        ROLE "WAS BORN IN" IS MENDATORY

CONSTRAINT C4:
        ROLE "IS BIRTHYEAR FOR" IS MENDATORY

CONSTRAINT C5:
        ROLE "HAS BEEN SPENT ON BY" IS MENDATORY

CONSTRAINT C6:
        ROLE "IS TIME SPENT ON BY" IS MENDATORY
```

*รูปที่ 6-2 แสดง Conceptual Schema ในรูปของข้อความ*

และจาก Conceptual Schema ที่เราได้มาเราก็จะนำไปสร้างเป็น Metaschema ขั้นต้นและจาก Metaschema ที่ได้ เราก็จะเริ่มเพิ่มความซับซ้อนของแบบจำลองข้อมูลเข้าไปเรื่อย ๆ โดยเพิ่มส่วนต่างๆ ของแบบจำลองข้อมูล เช่น สับไทป์ คอนสแตนต์ และปรับปรุง Metaschema ของเราให้สามารถรองรับแบบจำลองข้อมูลได้ ซึ่งการทำงานดังกล่าวนับเป็นการทำงานที่ต้องอาศัยความอดทนเป็นอย่างมาก เพราะต้องสร้างข้อมูลทดลองที่มีความซับซ้อนมากขึ้นเรื่อย ๆ ซึ่งก็หมายถึงความยาวที่มากขึ้นด้วย ซึ่งในบางครั้งข้อมูลที่เพิ่มเข้ามานี้ก็ได้ทำให้แบบจำลองที่เราสร้างมาตั้งแต่ต้น ต้องเปลี่ยนแปลงรูปแบบไปอย่างสิ้นเชิงเลยก็มี และผลสุดท้ายเราก็ได้ Conceptual Meta Schema ที่เราค่อนข้างแน่ใจว่าจะสามารถรองรับแบบจำลองข้อมูลได้ทุกรูปแบบ ดังแสดงในรูปที่ 6-3




*รูปที่ 6-3 แสดง Conceptual Meta Schema ที่ออกแบบเสร็จแล้ว*

## 6.2 รายละเอียดของเอนติตี้ต่าง ๆ

จากแบบจำลองข้อมูล Conceptual Meta Schema ในรูปที่ 6.3 จะเห็นว่ามีเอนติตี้ต่าง ๆ อยู่มากถึง 28 เอนติตี้ ซึ่งต่อไปเราจะอธิบายว่าแต่ละเอนติตี้มีหน้าที่สำหรับเก็บข้อมูลอะไรบ้าง และมีความสัมพันธ์กันอย่างไร โดยรายละเอียดของแต่ละเอนติตี้มีดังต่อไปนี้

| | |
|---|---|
| Object Type | ออบเจ็กต่าง ๆ ในแบบจำลองข้อมูล จะใช้ Obj_Name ในการอ้างถึง |
| Label Type | ชนิดเลเบิ้ลซึ่งเป็นสับเซตของอ็อบเจ็คไทป์ ใช้ Obj_Name ในการอ้างถึง |
| Entity Type | ชนิดเอนติตี้ซึ่งเป็นสับเซตของอ็อบเจ็คไทป์ ใช้ Obj_Name ในการอ้างถึง |
| Kind_Of_Obj | ประเภทของอ็อบเจ็คของไนแอม (Label or Entity) ใช้ Kind ในการอ้างถึง |
| Label_Order | ลำดับของชนิดเลเบิ้ล ใช้ Order ในการอ้างถึง |
| Role | โรล ใช้ Role_Name ในการอ้างถึง |
| Role Type | ชนิดของโรล (แบบปกติ, แบบเนสติ้ง) ใช้ Type ในการอ้างถึง |
| Role Order | ลำดับของโรล ในแต่ละชนิดความจริง หรือ ชนิดอ้างอิง ใช้ Order ในการอ้างถึง |
| Normal Role | โรลแบบปกติ เป็นสับเซตของ Role ใช้ Role_Name ในการอ้างถึง |
| Role Group | กลุ่มของโรล ใช้กับโรลที่มีความสัมพันธ์ในแบบกลุ่ม ใช้ R_Group ในการอ้างถึง |
| Nesting Role | โรลแบบเนสติ้ง เป็นสับเซตของ Role ใช้ Role_Name ในการอ้างถึง |
| Relationship Type | ประเภทความสัมพันธ์ ซึ่งมีอยู่ 2 ประเภทคือ ชนิดความจริง และ ชนิดอ้างอิง ใช้ Rel_Name ในการอ้างถึง |
| Fact Type | ชนิดความจริง เป็นสับเซตของ Relationship Type มี 2 ประเภทคือ แบบปกติ (Normal) และแบบหาได้ (Derived) ใช้ Rel_Name ในการอ้างถึง |
| Normal Fact Type | แฟกท์ไทป์แบบปกติ เป็นสับเซตของ Fact Type ซึ่งใช้แทนความสัมพันธ์ที่เก็บในฐานข้อมูลจริง ในรูปของตาราง ใช้ Rel_Name ในการอ้างถึง |
| Derived Fact Type | ชนิดความจริงแบบหาได้ เป็นสับเซตของ Fact Type ใช้แทนความสัมพันธ์ที่มาจากการคำนวณค่าข้อมูลของความสัมพันธ์อื่น ใช้ Rel_Name ในการอ้างถึง |
| Derived Formula | สูตรที่คำนวณค่าข้อมูลในความสัมพันธ์แบบหาได้ เก็บได้ทั้งสูตรการคำนวณและโปรแกรมภาษา SQL ที่ใช้ในการสร้างวิว ใช้ Form ในการอ้างถึง |
| Kind_Of_Rel | ประเภทของรีเลชัน ได้แก่ ชนิดอ้างอิง (REF) ชนิดความจริงแบบหาได้ (F_DERIVED) ชนิดความจริงแบบปกติ (F_NORMAL) ใช้ Kind ในการอ้างถึง |
| Constraint | ข้อจำกัดหรือคอนสเตรนท์ ใช้ Const# ในการอ้างถึง |
| Kind_Of_Const | ประเภทของคอนสเตรนท์ ได้แก่ Uniqueness, Mendatory, Exclusion, Subset, Equality, Membership, Range, Occurrence Frequency ใช้ Kind ในการอ้างถึง |
| Subset Const | คอนสเตรนท์แบบ Subset เป็นสับเซตของ Constraint เนื่องจากเป็นประเภทหนึ่งของคอนสเตรนท์ จึงใช้การอ้างถึงเช่นเดียวกันคือ Const# |
| Object Type Const | คอนสเตรนท์แบบ Object Type เป็นสับเซตของ Constraint เนื่องจากเป็นประเภทหนึ่งของคอนสเตรนท์ที่เกี่ยวกับอ็อบเจ็ค จึงใช้การอ้างถึงเช่นเดียวกันคือ Const# |
| Occurrence Freq | คอนสเตรนท์แบบ Occurrence Frequency เป็นสับเซตของ Constraint เนื่องจากเป็นประเภทหนึ่งของคอนสเตรนท์ จึงใช้การอ้างถึงเช่นเดียวกันคือ Const# |
| Frequency | ค่าความถี่ที่กำหนดในคอนสเตรนท์แบบ Occurrence Frequency มีทั้งค่าความถี่เริ่ม |

| | |
|---|---|
| | ต้น และความถี่สุดท้าย ใช้ Time ในการอ้างถึง |
| Membership Const | คอนสเตรนท์แบบ Membership เป็นสับเซตของคอนสเตรนท์แบบ เนื่องจากเป็นประเภทหนึ่งของคอนสเตรนท์ จึงใช้ Const# การอ้างถึง |
| Member | สมาชิกที่กำหนดอยู่ในคอนสเตรนท์แบบเมมเบอร์ชิพใช้ MEMBER# ในการอ้างถึง |
| Range Const | คอนสเตรนท์แบบเรนจ์ ซึ่งเป็นสับเซตของคอนสเตรนท์แบบ Object Type เนื่องจากเป็นประเภทหนึ่งของคอนสเตรนท์ จึงใช้ Const# ในการอ้างถึง |
| Limit | เซตของค่าจำกัดที่กำหนดอยู่ในคอนสเตรนท์แบบเรนจ์ ค่าจำกัดนี้มีได้ทั้งค่า จำกัดบน และค่าจำกัดล่าง ใช้ Lim ในการอ้างถึง |

สำหรับความสัมพันธ์ระหว่างชนิดเอนติตี้ใน Conceptual Meta Schema จะมีความสัมพันธ์ทั้งหมด 2 แบบด้วยกัน คือ ความสัมพันธ์ในแบบสับเซต และ ความสัมพันธ์ในแบบชนิดความจริง โดยมีรายละเอียดของความสัมพันธ์แต่ละแบบดังต่อไปนี้

### 6.2.1 ความสัมพันธ์แบบสับเซต

ความสัมพันธ์ในแบบสับเซต สามารถแบ่งย่อยออกเป็นกลุ่ม ๆ ได้อีก 4 กลุ่ม คือ

1. เอนติตี้ Object Type แบ่งเป็น 2 ประเภท ได้แก่ Label Type และ Entity Type เอนติตี้ทั้งสองจึงเป็น Subtype ของ Object Type

2. เอนติตี้ Role แบ่งเป็น 2 ประเภท ได้แก่ Normal Role และ Nesting Role เอนติตี้ทั้งสองเป็น Subtype ของ Role

3. เอนติตี้ Relationship Type แบ่งเป็น 2 ประเภท ได้แก่ Fact Type และ Reference Type โดยในส่วนของเอนติตี้ Fact Type จะปรากฏในรูป Metaschema แต่สำหรับ Reference Type จะอยู่ในส่วนแมปปิ้งอินฟอร์เมชั่น (Mapping Information) ซึ่งจะกล่าวต่อไป โดยในเอนติตี้ Fact Type จะแบ่งประเภทออกเป็น Subtype ย่อยลงไปอีกคือ Normal Fact Type และ Derived Fact Type

4. เอนติตี้ Constraint แบ่งเป็น 8 ประเภท หากแต่มีเพียง 4 ประเภทที่ปรากฏใน Metaschema เนื่องจากคอนสเตรนท์ตัวอื่น ๆ ไม่มีข้อมูลบ่งพิเศษ จะมีความสัมพันธ์ก็แต่เพียงกับ Role Group เท่านั้น ตัวอย่างคอนสเตรนท์เหล่านี้ เช่น Uniqueness, Mendatory, Exclusion เป็นต้น จากรูปจะมีสับไทป์ 3 สับไทป์ อันได้แก่ Subset, Object Type และ Occurence Frequency โดยที่เอนติตี้ Object Type ยังแบ่งประเภทย่อยออกเป็นอีก 2 สับไทป์ ซึ่งได้แก่ Membership และ Range

### 6.2.2 ความสัมพันธ์แบบชนิดความจริง

ความสัมพันธ์ในแบบชนิดความจริง สามารถแบ่งอธิบายตามกลุ่มสับเซตทั้ง 4 กลุ่ม ดังต่อไปนี้

6.2.2.1. กลุ่ม Object Type

- มีความสัมพันธ์กับ Normal Role เนื่องจากโรลแบบปกติ (ไม่ใช่โรลแบบเนสติ้ง) ต้องต่ออยู่กับ Object Type ตัวใดตัวหนึ่งเสมอ จะอยู่ลอย ๆ ไม่ได้

- มีความสัมพันธ์กับ Kind_Of_Obj เพื่อชี้บ่งว่า Object Type ใด ๆ เป็นประเภทชนิดเอนติตี้ หรือ ชนิดเลเบิ้ล เรียกเอนติตี้ไทป์อย่าง Kind_Of_Obj นี้ว่า "ตัวบ่งชนิด" หรือ Category

- สับไทป์ Entity Type มีความสัมพันธ์กับตัวเอง กล่าวคือ ความสัมพันธ์ดังกล่าวนี้เกิดขึ้นในกรณีที่มีการวาดสับไทป์ในไนแอม ชนิดความจริงนี้จะเก็บข้อมูลของชื่อเอนติตี้ไทป์ที่เป็นซูเปอร์ไทป์และชื่อของเอนติตี้ที่เป็นสับไทป์

- สับไทป์ Label Type มีความสัมพันธ์กับ Label Order เพื่อใช้บ่งว่าชนิดเลเบิ้ลตัวใด ๆ มีลำดับเท่าไรในการอ้างถึงของเอนติตี้ที่เกี่ยวข้องด้วย เกิดในกรณีที่มีการวาดชนิดเอนติตี้ที่มีการอ้างถึง (reference) ชนิดเลเบิ้ลมากกว่าหนึ่ง

6.2.2.2. กลุ่ม Role

- มีความสัมพันธ์กับชนิดเอนติตี้ Role Type เพื่อเป็นตัวบ่งชนิดว่าโรลตัวใด ๆ เป็นโรลประเภทโรลปกติ หรือ โรลแบบเนสติ้ง (หมายถึงโรลที่ต่ออยู่กับเนสติ้ง)

- มีความสัมพันธ์กับ Role Order เพื่อเก็บหมายเลขกำหนดลำดับของโรลภายในชนิดความจริงหรือชนิดอ้างอิง

- มีความสัมพันธ์กับ Role Group เนื่องจากจากการกำหนดคอนสเตรนท์จะกระทำบนโรลใน Relationship Type (ชนิดความจริงและชนิดอ้างอิง) ทั้งในลักษณะของการใช้โรลมากกว่า 1 โรล หรือการผนวกโรลเข้าไว้เป็นกลุ่ม เช่น การกำหนดคอนสเตรนท์แบบ Inter-fact type Uniqueness Constraint ซึ่งต้องอาศัยกลุ่มโรล 2 กลุ่ม โดยแต่ละกลุ่มอาจมีเพียง 1 หรือมากกว่า 1 โรลก็ได้ ความสัมพันธ์อันนี้จึงหมายถึง การกำหนดว่าโรลใดเป็นสมาชิกในกลุ่มโรลใด (หากไม่มีการกำหนดคอนสเตรนท์บนโรล โรลดังกล่าวก็ไม่จำเป็นต้องมีกลุ่ม)

- มีความสัมพันธ์กับ Relationship Type เนื่องจากในแต่ละ Relationship Type ประกอบด้วยโรลต่าง ๆ ดังนั้นทุกโรลที่ปรากฏในไนแอมจะต้องมีการบันทึกที่อยู่ในความสัมพันธ์ใดความสัมพันธ์หนึ่งเสมอ

- สับไทป์ Normal Role มีความสัมพันธ์กับ Object Type เพื่อเป็นการบันทึกว่าโรลปกติใดต่ออยู่กับอ็อบเจ็คใด

- สับไทป์ Nesting Role มีความสัมพันธ์กับชนิดเอนติตี้ Normal Fact Type เนื่องจากโรลแบบเนสติ้งต้องต่ออยู่กับเนสติ้ง ซึ่งที่จริงแล้วโดยตัวเนสติ้งเอง ก็มีฐานะเป็นชนิดความจริงปกติตัวหนึ่ง ดังนั้นจึงต้องบันทึกโรลเนสติ้งคู่ไปกับชนิดความจริงปกติที่เป็นเนสติ้ง

**6.3 รายละเอียดของกลุ่ม Relationship Type**

สำหรับรายละเอียดใน Conceptual Meta Schema มีรายละเอียดดังต่อไปนี้

- มีความสัมพันธ์กับ Role ในลักษณะที่บอกว่าโรลชื่อใด อยู่ภายในรีเลชันใด

- มีตัวบ่งชนิดคือ Kind_Of_Rel ซึ่งจะบ่งชนิดของ Relationship Type ว่าเป็นชนิดความจริงหรือชนิดอ้างอิง หากเป็นชนิดความจริงจะเป็นประเภทหาได้หรือประเภทปกติ

- สับไทป์ Derived Fact Type ซึ่งเป็นสับไทป์ของ Fact Type อีกทีหนึ่ง มีความสัมพันธ์กับ Derived Formula ซึ่งเป็นสูตรที่ใช้ในการหาค่าข้อมูลของชนิดความจริงแบบหาได้ โดยจะเก็บเป็นภาษา SQL หรือ เก็บในลักษณะสูตรคำนวณก็ได้

**6.4 รายละเอียดของกลุ่ม Constraint**

สำหรับรายละเอียดที่เกี่ยวข้องกับ Constraint จะมีรายละเอียดดังต่อไปนี้

- มีตัวบ่งชนิดคือ Kind_Of_Const เพื่อบ่งว่าคอนสเตรนท์ในเป็นคอนสเตรนท์ชนิดใด

- มีความสัมพันธ์กับ Role Group เนื่องจากคอนสเตรนท์เป็นข้อจำกัดที่กระทำอยู่บนโรล ซึ่งเป็นสมาชิกอยู่ในกลุ่มโรล คอนสเตรนท์จึงต้องสัมพันธ์กับกลุ่มโรล แต่ก็ยังมีคอนสเตรนท์บางประเภทที่ไม่ต้องยึดอยู่บนโรล ได้แก่ คอนสเตรนท์ในกลุ่มของ Object Type Constraint ที่ยึดอยู่กับอ็อบเจ็คในไนแอมแทน

- สับไทป์ Object Type Const เป็นคอนสเตรนท์ประเภทที่ยึดอยู่กับอ็อบเจ็คเป็นหลัก จึงมีความสัมพันธ์กับ Object_Type แทนที่จะสัมพันธ์กับ Role Group เช่น คอนสเตรนท์ตัวอื่น ๆ

- สับไทป์ Occurrence Frequency เป็นคอนสเตรนท์ที่ต้องมีการกำหนดค่าความถี่บนและค่าความถี่ล่างคือ Frequency

- สับไทป์ Membership Constraint เป็นคอนสเตรนท์ที่ต้องมีการประกาศสมาชิกที่จะใช้ จึงมีความสัมพันธ์กับ Member ซึ่งเก็บสมาชิกที่ใช้ในคอนสเตรนท์แบบเมมเบอร์ชิพ

- สับไทป์ Range Constraint เป็นคอนสเตรนท์ที่ต้องมีการกำหนดค่าลิมิตสูงต่ำ จึงมีความสัมพันธ์กับ Limit ที่เก็บค่าลิมิต 2 ค่า คือ ค่าลิมิตบนและลิมิตล่าง

- สับไทป์ Subset Constrint เป็นคอนสเตรนท์ที่ต้องเก็บกลุ่มโรล ที่เป็นกลุ่มโรลที่มีสมาชิกน้อยกว่าอีกกลุ่มโรลหนึ่ง เรียกกลุ่มโรลนี้ว่า กลุ่มโรลที่เป็นสับเซต จากรูป Metaschema จะสังเกตเห็นว่ามีความสัมพันธ์อยู่กับ Role Group ถึง 2 ความสัมพันธ์ โดยที่ความสัมพันธ์หนึ่งเก็บในลักษณะชนิดความจริงปกติ ได้แก่ ความสัมพันธ์ที่ใช้บอกกลุ่มโรลที่เป็นสับเซต กับอีกหนึ่งความสัมพันธ์ที่เป็นชนิดความจริงแบบหาได้ มีสมาชิกคือกลุ่มโรลที่เป็นซูเปอร์เซต เนื่องด้วยเป็นความสัมพันธ์ที่เป็นชนิดความจริงแบบหาได้นี้ จึงต้องมีการกำหนดสูตรที่ใช้ในการหาค่าข้อมูล โดยกรณีนี้จะเป็นภาษา SQL ที่ใช้สร้างวิวในฐานข้อมูล มีความว่า

```
CREATE VIEW SUPERSET_CONST ( CONST# , ROLE_GROUP ) AS
SELECT   T1.CONST# , T1.ROLE_GROUP
FROM     CONST_ROLE T1 , SUBSET_CONST T2
WHERE    T1.CONST# = T2.CONST#
             AND  T1.ROLE_GROUP <> T2.ROLE_GROUP ;
```

ซึ่งหมายถึงให้หาค่าข้อมูลจากการเชื่อม (join) ระหว่าง 2 ตาราง คือตาราง Const_Role ที่เก็บความสัมพันธ์ระหว่าง Constraint และ Role Group และตาราง Subset_Const ที่เก็บความสัมพันธ์ระหว่าง Constraint และ Role Group ที่เป็นสับเซต โดยมีเงื่อนไขว่าค่าข้อมูลที่จะอยู่ในความสัมพันธ์นี้จะต้องอยู่ในตาราง Const_Role แต่ไม่อยู่ใน Subset_Const

### 6.5 การแปลง Conceptual Meta Schema เป็นตารางบนฐานข้อมูลแบบสัมพันธ์

จากแบบจำลองข้อมูล Conceptual Meta Schema ที่ได้อธิบายในหัวข้อที่ผ่านมานั้น ในขั้นตอนต่อไปเราก็จะนำไปแปลงไปเป็นตารางบนฐานข้อมูลแบบสัมพันธ์ ซึ่งขั้นตอนการแปลงนั้นเราก็ได้กระทำตามขั้นตอนที่ได้กล่าวถึงในบทที่ 3 ซึ่งก็ได้ผลลัพธ์ออกมาเป็นตาราง 15 ตาราง กับวิวอีก 1 วิว ซึ่งจากตารางและวิวที่ได้นี้ เราก็จะนำไปสร้างเป็นตารางและวิวในฐานข้อมูล ORACLE เพื่อใช้เก็บข้อมูลจริง ๆ ของ Conceptual Schema ต่อไป ซึ่งรายละเอียดของตารางและวิวมีดังต่อไปนี้

(1) TABLE OBJECT_TYPE

| OBJECT_NAME | KIND_OF_OBJECT |
|---|---|
| CHAR(20) | CHAR(10) |

(2) TABLE LABEL_TYPE

| OBJECT_NAME | TYPE_NAME | LABEL_ORDER |
|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(5) |

(3) TABLE SUBTYPE

| SUB_NAME | SUPER_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(4) TABLE ROLE

| ROLE_NAME | ROLE_TYPE | RELATION_NAME | ROLE_ORDER |
|---|---|---|---|
| CHAR(20) | CHAR(10) | CHAR(10) | CHAR(1) |

(5) TABLE OBJECT_ROLE

| ROLE_NAME | OBJECT_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(6) TABLE NESTING

| ROLE_NAME | RELATION_NAME |
|---|---|
| CHAR(20) | CHAR(10) |

(7) TABLE RELATION_TYPE

| RELATION_NAME | KIND_OF_RELATION |
|---|---|
| CHAR(10) | CHAR(10) |

(8) TABLE ROLE_GROUP

| ROLE_NAME | ROLE_GROUP |
|---|---|
| CHAR(20) | CHAR(10) |

(9) TABLE CONSTRAINT

| CONST# | KIND_OF_CONST | CONST_NOTE |
|---|---|---|
| CHAR(10) | CHAR(10) | CHAR(1) |

(10) TABLE CONST_ROLE

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

(11) TABLE OBJECT_CONST

| CONST# | OBJECT_NAME |
|---|---|
| CHAR(10) | CHAR(20) |

(12) TABLE RANGE_CONST

| CONST# | MIN_NUMBER | MAX_NUMBER |
|---|---|---|
| CHAR(10) | CHAR(5) | CHAR(5) |

(13) TABLE MEMBERSHIP_CONST

| CONST# | MEMBER |
|---|---|
| CHAR(10) | CHAR(10) |

(14) TABLE SUBSET_CONST

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

(15) VIEW SUPERSET_CONST

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

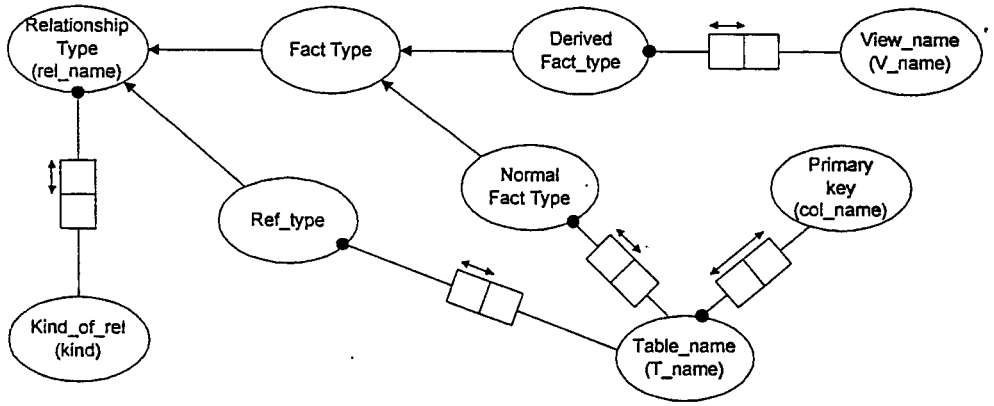
```
CREATE VIEW SUPERSET_CONST AS
    SELECT  T1.CONST# , T1.ROLE_GROUP
    FROM  CONST_ROLE T1 , SUBSET_CONST T2
    WHERE  T1.CONST# = T2.CONST#
    AND  T2.ROLE_GROUP <> T1.ROLE_GROUP ;
```

(16) TABLE FREQ_CONST

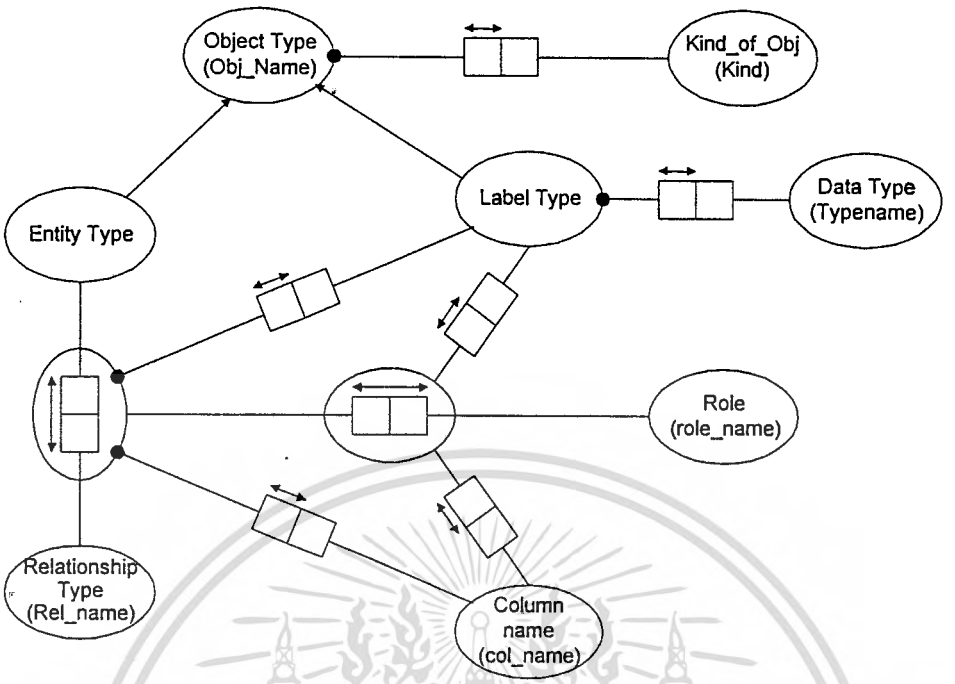
| CONST# | UPPER_LIMIT | LOWER_LIMIT |
|---|---|---|
| CHAR(10) | CHAR(5) | CHAR(5) |

**6.6 แมปปิ้งอินฟอร์เมชัน**

และนอกจากตารางที่เก็บข้อมูล Conceptual Schema แล้ว ในขั้นตอนการแปลงข้อมูลจาก Conceptual Schema ให้เป็นตารางในฐานข้อมูลรีเลชันแนล จะต้องมีข้อมูลส่วนหนึ่งที่จำเป็นต้องใช้งานในระหว่างขั้นตอนการแปลง และผลลัพธ์ที่ได้จากขั้นตอนการแปลงก็จำเป็นต้องมีที่เก็บเช่นกัน เช่น ชื่อวิว ชื่อตาราง เป็นต้น โดยข้อมูลที่ได้กล่าวไปทั้งหมดข้างต้นนี้ เราจะเรียกว่า แมปปิ้งอินฟอร์เมชั่น (Mapping information)




*รูปที่ 6-4 Mapping Information ในส่วนเก็บชื่อของตารางรีเลชั่นที่ได้จากกระบวนการแมปปิ้ง*




*รูปที่ 6-5 . Mapping Information ในส่วนเก็บการอ้างอิงของเอนติตี้*

แมปปิ้งอินฟอร์เมชันประกอบด้วย 2 ส่วน คือ 1. ส่วนเก็บการอ้างอิงของเอนติตี้ในแบบจำลองไนแอม (ดูรูปที่ 6-5) และ 2. ส่วนเก็บชื่อของตารางรีเลชั่นที่ได้จากกระบวนการแมปปิ้ง (ดูรูปที่ 6-4) ซึ่งกระบวนการที่ออกแบบโครงสร้างข้อมูลสำหรับเก็บแมปปิ้งอินฟอร์เมชัน ก็จะใช้ขั้นตอนในแบบเดียวกับที่ออกแบบ Conceptual Schema นั่นเอง

**6.6.1** เอนติตี้ไทป์ที่เพิ่มเข้ามาในส่วนแมปปิ้งอินฟอร์เมชั่น

1. Column Name ใช้ในการเก็บชื่อคอลัมน์ของรีเลชั่นที่ได้จากกระบวนการแมปปิ้ง มีความสัมพันธ์กับ Entity Type, Relationship Type, Role และ Label Type ในลักษณะของเนสติ้ง เป็นความสัมพันธ์ที่ต้องเกิดขึ้นทุกครั้งที่มีการสร้างไนแอมเพื่อแมปปิ้งเป็นรีเลชั่น

2. Data Type ใช้ในการเก็บชนิดข้อมูลของแต่ละแอตตริบิวท์ที่อยู่ในรีเลชั่นที่ได้จากกระบวนการแมปปิ้ง ชนิดข้อมูลดังกล่าวนี้ผู้ใช้งานระบบ (นักออกแบบ) จะต้องกำหนดไว้ตั้งแต่เริ่มวาดแบบจำลอง มิฉะนั้นจะไม่สามารถสร้างรีเลชั่นได้ ชนิดข้อมูลนี้จะมีความสัมพันธ์กับชนิดเลเบิ้ล Label Type เนื่องจากชนิดเลเบิ้ลเป็นการอ้างถึงของเอนติตี้ในระบบจึงถือเป็นคุณสมบัติที่เอนติตี้ต่าง ๆ พึงจะมี

3. Table Name ใช้ในการเก็บชื่อของรีเลชั่นที่ได้จากกระบวนการแมปปิ้ง มีความสัมพันธ์กับ Reference Type และ Normal Fact Type เนื่องจากในกระบวนการแมปปิ้งจะอาศัย Relationship Typeทั้งสองเป็นหลักในการสร้างความสัมพันธ์ที่เป็นรีเลชั่น นอกจากนี้ยังมีความสัมพันธ์กับ Primary Key ซึ่งจะกล่าวต่อไป

4. Primary Key ใช้ในการเก็บชื่อแอตตริบิวท์ที่เป็นไพรมารี่คีย์ของรีเลชั่นที่ได้ เพื่อบ่งบอกคุณสมบัติของรีเลชั่นนั้น ๆ

5. View Name ใช้ในการเก็บชื่อวิวที่ได้จากกระบวนการสร้างวิวจากซนิดความจริงแบบหาได้ของไน่แอม

**6.6.2 แมปปิ้งอินฟอร์เมชั่นในลักษณะตารางรีเลชันบนฐานข้อมูลรีเลชันแนล**

จากแบบจำลองข้อมูลของแมปปิ้งอินฟอร์เมชัน เราก็ได้นำไปแปลงเป็นตารางเพื่อจะเก็บในฐานข้อมูลแบบรีเลชันแนลต่อไป ซึ่งก็ได้ผลลัพธ์ออกมา ดังต่อไปนี้

(1) TABLE UNIQ_REF

| ENTITY_NAME | RELATION_NAME | ROLE_NAME | LABEL_NAME | COL_NAME |
|---|---|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) |

(2) TABLE NONUNIQ_REF

| ENTITY_NAME | RELATION_NAME | LABEL_NAME | COL_NAME |
|---|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) |

(3) TABLE NORMAL_FACTTYPE

| RELATION_NAME | TABLE_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(4) TABLE REFERENCE

| RELATION_NAME | TABLE_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(5) TABLE PRIMARY_KEY

| TABLE_NAME | COL_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(6) TABLE DERIVED_FACTTYPE

| RELATION_NAME | VIEW_NAME | DERIVED_DEF |
|---|---|---|
| CHAR(10) | CHAR(20) | CHAR(100) |

**6.7 ข้อมูลรายละเอียดของแอตตริบิวท์ต่าง ๆ ในตารางเมต้าสกีมม่าและแมปปิ้งอินฟอร์เมชั่น**

จากข้อมูลต่าง ๆ ที่ได้กล่าวไปทั้งหมด ไม่ว่าจะเป็นข้อมูลในส่วนของ Conceptual Schema หรือส่วน Mapping Information ทั้งในแง่ของแบบจำลองข้อมูลและในแง่ของตาราง ซึ่งในตอนสุดท้ายนี้ก็ได้รวบรวมเอารายละเอียดข้อมูลทั้งหมดไว้แล้วดังตารางข้างล่างนี้

| ชื่อแอตตริบิวท์ | จำนวนอักขระ | ความหมาย |
|---|---|---|
| OBJECT_NAME | 20 | เป็นตัวเก็บชื่อของ Entity หรือ Label |
| KIND_OF_OBJECT | 10 | เก็บชนิดของ Object_Name ("LABEL", "ENTITY") |
| TYPE_NAME | 20 | ชี้ให้เห็นว่า Object_Name นี้ เมื่อแมปเป็นตารางแล้วจะเก็บข้อมูลเป็นอย่างไร เช่น C10 จำนวนอักขระ 10 ตัวอักษร, N10 ค่าตัวเลขขนาด 10 หลัก เป็นต้น |
| LABEL_ORDER | 5 | เป็นตัวกำหนดลำดับค่าตัวเลขให้กับ Label ต่อ 1 Entity (1,2,3,4,5,...) |
| SUB_NAME | 20 | ชื่อ Subtype ของ Entity |
| SUPER_NAME | 20 | ชื่อ Supertype ของ Entity |
| ROLE_NAME | 20 | ชื่อของ Role |
| ROLE_TYPE | 10 | ชนิดของ Role ("NORMAL","NESTING") |
| ROLE_ORDER | 1 | กำหนดลำดับของ Role (1,2,3,4,...,9) |
| RELATION_NAME | 10 | ชื่อของความสัมพันธ์ (จะเก็บชื่อของชนิดความจริง และ ชื่อของชนิดอ้างอิง) |
| KIND_OF RELATION | 10 | ชนิดของความสัมพันธ์ ( "F_NORMAL", " F_DERIVED","REF") |
| ROLE_GROUP | 10 | กลุ่มของ Constraint |
| CONST# | 10 | ID ของ Constraint (C1,C2,...,Cn) |
| KIND_OF_CONST# | 10 | ชนิดของ Constraint โดยจะมีอยู่ 7 ชนิดดังนี้ (UNIQUE, EXCLUSIV, EQUAL, SUBSET, RANGE, FREQ, MEMBER, MAND) |
| CONST_NOTE | 1 | เป็น Flag แสดงว่าจะให้มีการเช็คConstraints หรือไม่ ( N ไม่เช็ค, Y เช็ค ) |
| MIN_NUMBER | 5 | ความถี่ล่างที่ใช้ในการกำหนด Occurence frequency เก็บเป็นค่าจำนวนเต็ม |
| MAX_NUMBER | 5 | ความถี่บนที่ใช้ในการกำหนด Occurence frequency เก็บค่าจำนวนเต็ม |
| MEMBER | 10 | สมาชิกที่กำหนดใน Membership constraint เก็บเป็นสตริงขนาดไม่เกินที่กำหนด |

| | | |
|---|---|---|
| UPPER_LIMIT | 5 | เป็นค่าตัวเลขที่มากที่สุดที่เก็บได้ในแต่ละ Object |
| LOWER_LIMIT | 5 | เป็นค่าตัวเลขที่น้อยที่สุดที่เก็บได้ในแต่ละ Object |
| ENTITY_NAME | 20 | ชื่อเอนติตี้ |
| LABEL_NAME | 20 | ชื่อเลเบิ้ล |
| COL_NAME | 20 | ชื่อคอลัมน์ |
| TABLE_NAME | 20 | ชื่อตาราง |
| VIEW_NAME | 20 | ชื่อวิว |
| DERIVED_DEF | 100 | สูตรที่ใช้ในการ Derived |

# บทที่ 7

# การพัฒนาระบบฐานข้อมูลอนุมาน

ในบทที่ผ่านมาเราได้กล่าวถึงทฤษฎีต่าง ๆ ที่จำเป็นต้องทราบในการทำวิทยานิพนธ์ฉบับนี้ ได้กล่าวถึงสถาปัตยกรรมของระบบทั้งหมดที่เราได้สร้างขึ้น และได้กล่าวถึงการออกแบบเมต้าสกีมาไปแล้ว ซึ่งงานทั้งหมดที่ผ่านมานั้นเป็นพื้นฐานที่สำคัญต่อการพัฒนาต่อไป และจากบทนี้เป็นต้นไปจะเป็นรายละเอียดของการพัฒนาโปรแกรม และรายละเอียดการทำงานของส่วนประกอบแต่ละส่วนอย่างละเอียด

สำหรับในบทนี้จะกล่าวถึงการพัฒนาระบบโดยรวมทั้งหมด ซึ่งประกอบด้วยส่วนติดต่อกับผู้ใช้ ส่วนนิยามข้อมูล ส่วนจัดการข้อมูล ส่วนตรวจสอบความถูกต้องของข้อมูล และส่วนประมวลผลคำถาม โดยจะเน้นไปที่การส่งผ่านการทำงานระหว่างส่วนประกอบต่าง ๆ ซึ่งจะทำให้ทราบถึงรูปแบบการทำงานโดยรวมทั้งหมดของโครงงานนี้ นอกจากนั้นในบทนี้ยังจะกล่าวถึงรายละเอียดของการพัฒนาส่วนติดต่อกับผู้ใช้โดยละเอียดอีกด้วย โดยส่วนติดต่อกับผู้ใช้จะรับคำสั่งจากผู้ใช้ และจัดการประมวลผลเบื้องต้นให้ จากนั้นจึงส่งให้กับส่วนต่าง ๆ ทำงานตามหน้าที่ต่อไป

## 7.1 อุปกรณ์ที่ใช้ในการพัฒนาระบบ

ในการพัฒนาระบบนั้นจำเป็นต้องมีเครื่องมือสำหรับช่วยในการพัฒนา ซึ่งประกอบด้วยเครื่องคอมพิวเตอร์ และโปรแกรมช่วยพัฒนาระบบต่าง ๆ ซึ่งมีรายละเอียดดังต่อไปนี้

- เครื่องคอมพิวเตอร์

เครื่องคอมพิวเตอร์ที่ใช้ในโครงงานนี้เป็นไมโครคอมพิวเตอร์ IBM PC คอมแพทิเบิลโดยมีหน่วยประมวลผลกลาง (CPU : Central Processing Unit) ขนาด 32 บิต 80486 DX2-66 หน่วยความจำ 8 เมกะไบต์ ฮาร์ดดิสก์ขนาด 213 เมกะไบต์ ฟลอปปี้ดิสก์ขนาด 1.44 และ 1.2 เมกะไบต์ อย่างละ 1 ตัว จอภาพแบบ Super VGA (1024 X 768) 256 สี แป้นพิมพ์ 101 คีย์

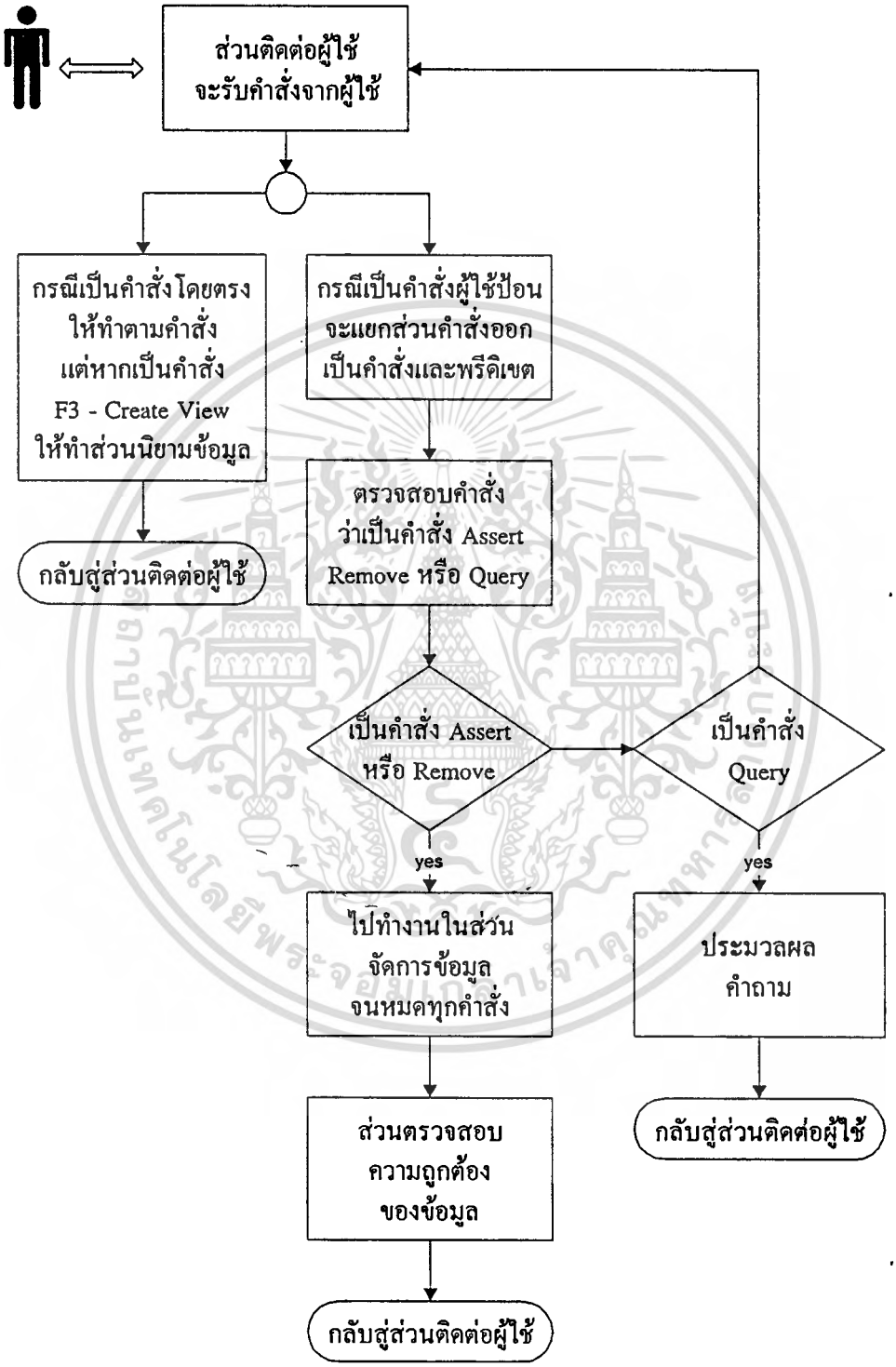
- ซอฟต์แวร์ที่ใช้ในการพัฒนาระบบ

ซอฟต์แวร์ที่ใช้ในการพัฒนาระบบประกอบด้วยโปรแกรมระบบปฏิบัติการ MS-DOS 6.2 และโปรแกรมไมโครซอฟต์วินโดวส์ ทำหน้าที่เป็นระบบปฏิบัติการหลัก โดยโปรแกรม MS-DOS 6.2 ใช้ในการพัฒนาโปรแกรม และไมโครซอฟต์วินโดวส์ใช้ในการเขียนวิทยานิพนธ์ สำหรับซอฟต์แวร์ฐานข้อมูลในโครงงานนี้จะใช้ ORACLE 6.0 และใช้โปรแกรม Microsoft C 8.0 ในการพัฒนาระบบทั้งหมด สำหรับโปรแกรมที่ใช้ในการเขียนวิทยานิพนธ์ประกอบด้วย Microsoft Word 6.0a และโปรแกรม VISIO 3.0

## 7.2 ส่วนประกอบต่าง ๆ ของระบบทั้งหมด

จากที่เราได้กล่าวในตอนต้นแล้วว่าโปรแกรมระบบฐานข้อมูลอนุมานนี้ มีส่วนประกอบทั้งหมดอยู่ 5 ส่วนด้วยกัน คือ ส่วนติดต่อกับผู้ใช้ ส่วนนิยามข้อมูล ส่วนจัดการข้อมูล ส่วนตรวจสอบความถูกต้อง

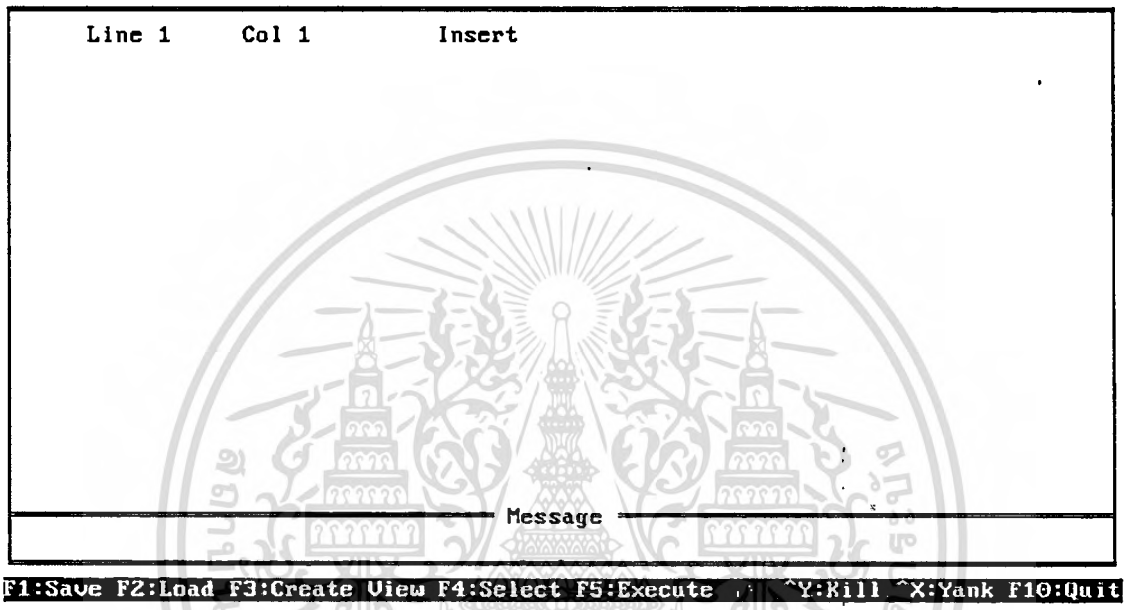
ของข้อมูล และส่วนประมวลผลคำถาม โดยมีส่วนติดต่อกับผู้ใช้ทำหน้าที่เป็นแกนกลางของระบบ โดยจะติดต่อกับผู้ใช้เพื่อรับคำสั่ง เพื่อจะส่งต่อให้กับส่วนอื่นต่อไป โดยมีรูปแบบการทำงานดังรูปที่ 7-1




*รูปที่ 7-1 แสดงการทำงานโดยรวมทั้งหมดของระบบฐานข้อมูลอนุมาน*

**7.3 การออกแบบส่วนติดต่อกับผู้ใช้**

การออกแบบส่วนติดต่อกับผู้ใช้นี้ ในทางปฏิบัติแล้วถือได้ว่าเป็นส่วนที่สำคัญส่วนหนึ่ง เพราะโปรแกรมจะใช้งานง่ายหรือยาก ก็ขึ้นอยู่กับการออกแบบส่วนติดต่อกับผู้ใช้ทั้งสิ้น แต่สำหรับในโครงงานนี้ เนื่องจากเป็นโครงงานที่เน้นหนักไปที่งานวิจัย ที่ทดสอบความเป็นไปได้ของการสร้างระบบฐานข้อมูลอนุมาน ดังนั้นเราจะไม่ออกแบบส่วนติดต่อกับผู้ใช้ให้มีความสามารถมากนัก โดยจะสร้างเป็นเอดิเตอร์แบบง่าย ๆ ที่มีรูปแบบการแสดงผลตามรูปที่ 7-2




*รูปที่ 7-2 แสดงรูปแบบการแสดงผลของส่วนติดต่อกับผู้ใช้*

จากรูปที่ 7-2 จะเห็นได้ว่าในจอภาพจะแบ่งออกเป็น 2 ส่วนด้วยกัน คือ ส่วนเนื้อที่ที่ใช้ในการแก้ไขข้อมูล และส่วนแสดงข้อความ โดยพื้นที่ส่วนแรกจะเป็นพื้นที่ทำงานหลัก ที่อนุญาติให้ผู้ใช้ป้อนข้อมูลเข้าไปได้ สำหรับส่วนแสดงข้อความนั้นจะเอาไว้สำหรับแสดงข้อความสั้น ๆ ที่เอาไว้ให้โปรแกรมสามารถโต้ตอบกับผู้ใช้งานได้ โดยมีเนื้อที่ใช้งานให้เพียงบรรทัดเดียว ที่ด้านบนของจอภาพจะมีข้อความบอกตำแหน่งของการแก้ไข โดยจะบอกเป็น Line และ Col เหมือน ๆ กับเอดิเตอร์ทั่วไป และที่บรรทัดล่างสุดจะมีข้อความบอกฟังก์ชันคีย์หลัก ๆ ที่ใช้กับโปรแกรมนี้ สำหรับฟังก์ชันคีย์ทั้งหมดที่ใช้ได้นั้นได้แสดงไว้ในตารางที่ 7-1 แล้ว

จากฟังก์ชันคีย์ในตารางที่ 1 ก็จะเห็นได้ว่าโปรแกรมนี้มีความสามารถในการแก้ไขข้อมูลอยู่พอสมควรเลยทีเดียว ก็เรียกว่าช่วยให้สามารถแก้ไขข้อมูลได้อย่างไม่อึดอัดใจมากนัก แต่ที่นับว่าเป็นข้อจำกัดอย่างหนึ่งของโปรแกรมนี้ คือ โปรแกรมนี้จะมีเนื้อที่รองรับข้อมูลได้เพียง 20 กิโลไบต์เท่านั้น ซึ่งแม้ว่าจะไม่มากมายอะไร แต่ก็นับว่าเพียงพอต่อการใช้งานในระดับทดลองใช้ สำหรับการใช้งานผู้ใช้ก็เพียงแต่ป้อนข้อมูลเมต้าสกีมาเข้าไปในตาราง จากนั้นก็กดฟังก์ชันคีย์ F3 เพื่อสร้างวิวที่จะใช้แทนชนิดความจริง จากนั้นก็ป้อนคำสั่งเข้าไป และกด F5 เพื่อทำงาน

| ฟังก์ชันคีย์ | การทำงาน | ฟังก์ชันคีย์ | การทำงาน |
|---|---|---|---|
| F1 | เซฟข้อมูล | UP | เลื่อนเคอร์เซอร์ขึ้น |
| F2 | โหลดข้อมูล | DOWN | เลื่อนเคอร์เซอร์ลง |
| F3 | สร้างวิวให้กับแฟ็กไทป์ | LEFT | เลื่อนเคอร์เซอร์ทางซ้าย |
| F4 | ขอดูข้อมูลจากตาราง | RIGHT | เลื่อนเคอร์เซอร์ทางขวา |
| F5 | ทำงานตามคำสั่ง | PAGE UP | เลื่อนขึ้น 1 หน้า |
| F10 | ออกจากโปรแกรม | PAGE DOWN | เลื่อนลง 1 หน้า |
| HOME | ไปที่ต้นไฟล์ | BACK SPACE | ลบอักษรก่อนหน้า |
| END | ไปที่ท้ายไฟล์ | CTRL-Y | ลบบรรทัด |
| DEL | ลบตัวอักษร 1 ตัว | CTRL-X | แทรกบรรทัด |

*ตารางที่ 7-1 แสดงฟังก์ชันคีย์ที่ใช้ได้กับโปรแกรมฐานข้อมูลอนุมาน*

สำหรับคำสั่งที่สามารถใช้กับโปรแกรมนี้ได้ ก็จะมีอยู่เพียง 3 คำสั่งด้วยกัน คือ คำสั่ง Assert, Remove และ Query ซึ่งสาเหตุที่มีเพียง 3 คำสั่งก็เนื่องจากว่าแบบจำลองข้อมูลไนแอมมองข้อมูลในรูปแบบเดียวเท่านั้น คือ มองในรูปแบบของชนิดความจริง และการกระทำที่สามารถทำได้กับชนิดความจริงก็มีเพียง 3 แบบเท่านั้น คือ การนำข้อมูลเข้าสู่ฐานข้อมูล (Assert) การลบข้อมูลออกจากฐานข้อมูล (Remove) และการเรียกดูข้อมูลจากฐานข้อมูล (Query)

การทำงานของโปรแกรมจะมีลักษณะการทำงานเป็นแบบแบตช์ กล่าวคือเมื่อสร้างหรือแก้ไขจนสมบูรณ์แล้ว จะต้องกดคีย์ F5-Execute โปรแกรมจึงจะนำคำสั่งที่พิมพ์เอาไว้ไปทำงาน โดยหากเป็นกรณีที่มีการนำข้อมูลเข้าหรือนำข้อมูลออกจากฐานข้อมูล โดยใช้คำสั่ง Assert หรือ Remove เราจะอนุญาตให้มีการประมวลผลเป็นทรานเซกชันได้ โดยการป้อนคำสั่งทีละหลายคำสั่งได้ แต่สำหรับคำสั่งประเภท Query แล้วจะต้องป้อนทีละคำสั่งเท่านั้น นอกจากนั้นการป้อนคำสั่งจะไม่สามารถป้อนคำสั่ง Query ปนกับคำสั่งอื่น ๆ ได้ และเมื่อโปรแกรมรับข้อมูลคำสั่งทั้งหมดมาแล้ว ก็จะจัดการแยกส่วนประกอบของคำสั่งออกเป็น ประเภทของคำสั่ง ชื่อเพรดิเคต และอากิวเมนต์ต่าง ๆ โดยจัดเก็บในตัวแปรชื่อ predicate โดยมีโครงสร้างดังต่อไปนี้

```
struct {
        char    type;
        char    pred_name[20];
        int     arg_num;
        char    argument[20][20];
        int     end_flag;
} predicate[20];
```

ตัวแปร predicate จะเป็นตัวแปรแบบ struct สามารถเก็บคำสั่งได้ทั้งหมด 20 คำสั่ง ซึ่งนับว่าพอเพียงกับการใช้งานทั่วไป แต่ถ้าหากไม่พอต่อการใช้งานก็สามารถแก้ไขโปรแกรมเพื่อเพิ่มเนื้อที่ได้ ภายใน struct predicate จะสามารถเก็บข้อมูลประเภทของคำสั่ง (type) โดยเก็บเป็นชื่อย่อ ('A','R','Q') เก็บชื่อเพรดิเขต (pred_name) จำนวนอากิวเมนต์ (arg_num) อากิวเมนต์ทั้งหมด (argument) ซึ่งสามารถเก็บได้ทั้งหมด 20 ตัวด้วยกัน โดยมีตัวแปร end_flag เป็นตัวระบุจุดสิ้นสุดของทรานเซกชัน

เมื่อโปรแกรมแยกคำสั่งออกมาทั้งหมดแล้ว ก็จะส่งให้แต่ละส่วนจัดการ โดยหากเป็นคำสั่ง Assert ก็จะส่งไปยังโปรแกรมย่อยที่ทำหน้าที่นำเข้าสู่ฐานข้อมูล และหากเป็นคำสั่ง Remove ก็จะส่งให้โปรแกรมย่อยที่ทำหน้าที่ลบข้อมูล และหากเป็นคำสั่ง Query ก็จะส่งไปยังโปรแกรมย่อยที่ทำหน้าที่ประมวลผลคำถาม และกระทำเช่นนี้จนครบทุกคำสั่ง เมื่อครบทุกคำสั่งแล้วหากเป็นคำสั่งแบบ Assert หรือ Remove ประเภทใดประเภทหนึ่ง ก็จะเรียกโปรแกรมย่อยตรวจสอบคอนสแตนต์ด้วย และหากผลการทำงานของการตรวจสอบคอนสแตนต์พบว่าขัดแย้ง ก็จะสั่งยกเลิกทรานเซกชันที่ได้ทำไปก่อนหน้านี้

สำหรับรูปแบบของคำสั่งที่สามารถใช้ได้นั้น จะต้องเริ่มต้นคำสั่งด้วยคำสั่ง Assert, Remove หรือ Query เท่านั้น จะเป็นคำสั่งอื่นนอกเหนือจากนี้ไม่ได้ ถัดจากคำสั่งก็จะต้องเป็นชื่อของชนิดความจริง และตามด้วยวงเล็บภายในวงเล็บจะเป็นลำดับของข้อมูล ตามด้วยวงเล็บปิดและปิดท้ายด้วยเครื่องหมาย ";" (Semi-Colon) อีกที ดังตัวอย่างต่อไปนี้

```
Assert car_owner('John','A1090');
Remove subject_mark('Mary','Database','A');
Query love('Mary','John');
Query Love('Mary',X);
```

จะเห็นได้ว่าหากข้อมูลเป็นตัวอักษรจะต้องใส่เครื่องหมายคำพูดกำกับไว้ทุกครั้ง และหากไม่ใส่เครื่องหมายคำพูดก็จะถือว่าเป็นตัวเลข โดยข้อมูลที่จะใช้กับคำสั่ง Assert และ Remove นั้นจะต้องเป็นข้อมูลที่เป็นค่าคงที่เสมอ แต่สำหรับคำสั่ง Query แล้วจะสามารถใช้ข้อมูลได้ทั้งในรูปค่าคงที่และตัวแปร โดยจะสังเกตว่าตัวแปรจะไม่มีเครื่องหมายคำพูดและจะต้องใช้เป็นอักษรตัวใหญ่เสมอ รายละเอียดการทำงานของฟังก์ชันคีย์ที่น่าสนใจ มีดังต่อไปนี้

- คำสั่ง Create View คำสั่งนี้เป็นคำสั่งที่ใช้ในการสร้างวิวให้กับแบบจำลองข้อมูล โดยจะมองว่าแต่ละวิวจะเป็น 1 ชนิดความจริง คำสั่งนี้จะใช้เพียงครั้งเดียวเท่านั้น คือ เมื่อป้อนข้อมูลในส่วนเมต้าสกีมาเสร็จเรียบร้อยแล้วเท่านั้น และเมื่อมีการเปลี่ยนแปลงเมต้าสกีมาจึงจะเรียกใช้คำสั่งนี้อีกที เมื่อเรียกใช้คำสั่งนี้โปรแกรมหลักก็จะเรียกโปรแกรมที่ใช้สร้างวิวขึ้นมาทำงาน โดยมีรายละเอียดในบทที่ 8

- คำสั่ง Select เป็นคำสั่งที่เรียกดูข้อมูลในตาราง โดยเมื่อเรียกใช้ฟังก์ชันนี้แล้วโปรแกรมจะถามชื่อตาราง และจำนวนคอลัมน์ในตาราง จากนั้นโปรแกรมก็จะสร้างคำสั่ง SQL และเรียกใช้คำสั่ง SQL แบบไดนามิก เพื่อดึงข้อมูลจากฐานข้อมูลมาแสดงผลที่จอภาพ โดยโปรแกรมจะสร้างวินโดวส์พิเศษที่ทำหน้าที่แสดงผลข้อมูลจากตารางโดยเฉพาะ ฟังก์ชันนี้เอาไว้ก็เพื่อเรียกดูผลจากการทำงานได้ง่าย ๆ โดยไม่ต้องออกจากโปรแกรมนั่นเอง

- คำสั่ง Execute คำสั่งนี้เป็นคำสั่งที่สั่งให้โปรแกรมนำคำสั่งที่ป้อนไว้ในเอดิเตอร์ไปทำงาน โดยการทำงานจะเริ่มจากการแยกส่วนประกอบของคำสั่งออกเป็นส่วนย่อย ๆ พร้อมทั้งตรวจสอบว่ารูปแบบของคำสั่งที่ป้อนเข้ามานั้นถูกต้องหรือไม่ ถ้าไม่ถูกก็จะฟ้องว่าเกิดข้อผิดพลาดเกิดขึ้น และยกเลิกการทำงานทั้งหมด โดยกลับไปรอให้ผู้ใช้ป้อนคำสั่งเข้ามาใหม่ แต่หากคำสั่งที่ป้อนเข้ามาถูกต้องตามรูปแบบที่กำหนดเอาไว้ทั้งหมด ก็จะตรวจสอบว่าเป็นคำสั่งอะไร ตรวจสอบว่าเป็นคำสั่ง Assert, Remove หรือ Query และทำงานตามที่ได้กล่าวไว้ข้างต้นต่อไป

สำหรับรายละเอียดการทำงานของส่วนต่าง ๆ จะได้กล่าวถึงในบทต่อ ๆ ไป

# บทที่ 8
# การออกแบบในส่วนนำข้อมูลเข้า

ในส่วนของการนำข้อมูลเข้านี้ จะประกอบด้วยส่วนหลัก ๆ สองส่วนด้วยกัน คือ ส่วนนิยามข้อมูล (Information Definition) และส่วนจัดการข้อมูล (Information Manipulation) โดยจะเริ่มการออกแบบในส่วนนิยามข้อมูลก่อน ดังนี้

## 8.1 ส่วนนิยามข้อมูล (Information Definition)

การทำงานในส่วนนี้ ที่จริงเป็นการทำงานส่วนหนึ่งในขั้นตอนการแม็ปจากแบบจำลองข้อมูลมาเป็นตาราง แต่เนื่องจากโครงงานนี้มิได้ครอบคลุมไปถึงการออกแบบและการแม็ปข้อมูลเป็นตาราง ดังนั้นจะถือว่าในตารางที่เก็บข้อมูลมีข้อมูลในระดับเมตาอยู่แล้ว และมีการสร้างตารางเรียบร้อยแล้วด้วย แต่เนื่องจากการประมวลผลการทำงานในรูปแบบของไนแอม จำเป็นต้องประมวลผลกับชนิดความจริง ไม่ใช่การประมวลผลในระดับตาราง ดังนั้นจึงใช้วิธีการสร้างวิวซ้อนขึ้นมาจากตารางอีกที โดยวิวที่สร้างขึ้นมานี้จะสอดคล้องกับชนิดความจริงที่เก็บอยู่ในข้อมูลระดับเมตาทุกประการ

สำหรับการทำงานโดยย่อของการสร้างวิวซึ่งถือเป็นงานหลักของส่วนนิยามข้อมูล ก็จะเริ่มต้นจากการอ่านข้อมูลชนิดความจริงขึ้นมาทีละข้อมูล โดยจะอ่านจากตาราง NORMAL_FACTTYPE ซึ่งเป็นตารางที่ประกอบด้วย 2 แอตทริบิวต์ คือ Relation_Name และ Table_Name โดยข้อมูลชนิดความจริงจะเก็บอยู่ใน Relation_Name นั่นเอง สำหรับเหตุผลที่เลือกอ่านข้อมูลชนิดความจริงจากตารางนี้ ก็เพราะข้อมูลในตารางนี้จะไม่มีการซ้ำกันของชนิดความจริง นอกจากนั้นยังได้ข้อมูลที่เป็นชื่อของตารางที่เก็บชนิดความจริงนั้นด้วย

เมื่อได้ชื่อของชนิดความจริงแล้ว งานในขั้นต่อไปก็คือการค้นหาข้อมูลที่เป็นองค์ประกอบของชนิดความจริงนั้น ซึ่งเป็นที่ทราบกันอยู่แล้วว่าชนิดความจริงประกอบด้วยโรล ดังนั้นเราก็จะค้นหาโรลของชนิดความจริงนั้น โดยค้นหาจากตารางที่ชื่อ ROLE โดยจะค้นหาตามลำดับของโรล ที่ระบุโดยแอตทริบิวต์ Role_Order ซึ่งจะได้ข้อมูลมา 2 ข้อมูล คือ Role_Name และ Role_Type โดย Role_Name ก็คือชื่อของโรลนั้น ๆ และสำหรับ Role_Type ก็จะเก็บข้อมูลที่ระบุว่า Role นั้น ๆ เป็น Role ประเภท Normal หรือประเภท Nesting

ในกรณีที่เป็น Nesting ก็จะเข้าสู่ขั้นตอนการทำงานพิเศษ โดยจะอ่านข้อมูล Role_Name ทุกตัวของ Relation_Name ที่ผูกติดอยู่กับโรลที่เป็น Nesting นั้น โดยจะต้องค้นหาชื่อของ Relation_Name ของ Role_Name ในตาราง NESTING จากนั้นจึงนำ Relation_Name มาค้นหาชื่อ Role_Name และทำเช่นนี้ไปจนกว่าจะครบทุก Role_Name ที่อยู่ใน Relation นั้น ซึ่งข้อมูล Role_Name ทุกตัวไม่ว่าจะเป็นแบบปกติหรือแบบ Nesting จะถูกเก็บเข้าไปไว้ในลิงก์ลิสต์ที่ได้กำหนดโครงสร้างไว้ดังนี้

```
struct clmn {
        char    role[20];
        char    entity[20];
        int     no_relation;
        struct {
                        char    relation_name[20];
                        char    relation_type[10];
                }relation[10];
        char    label_name[10][20];
        char    column_name[10][20];
        char    label_order[10];
        char    unique_id[10];
        struct  clmn    *next;
        };

typedef struct clmn Column;
```

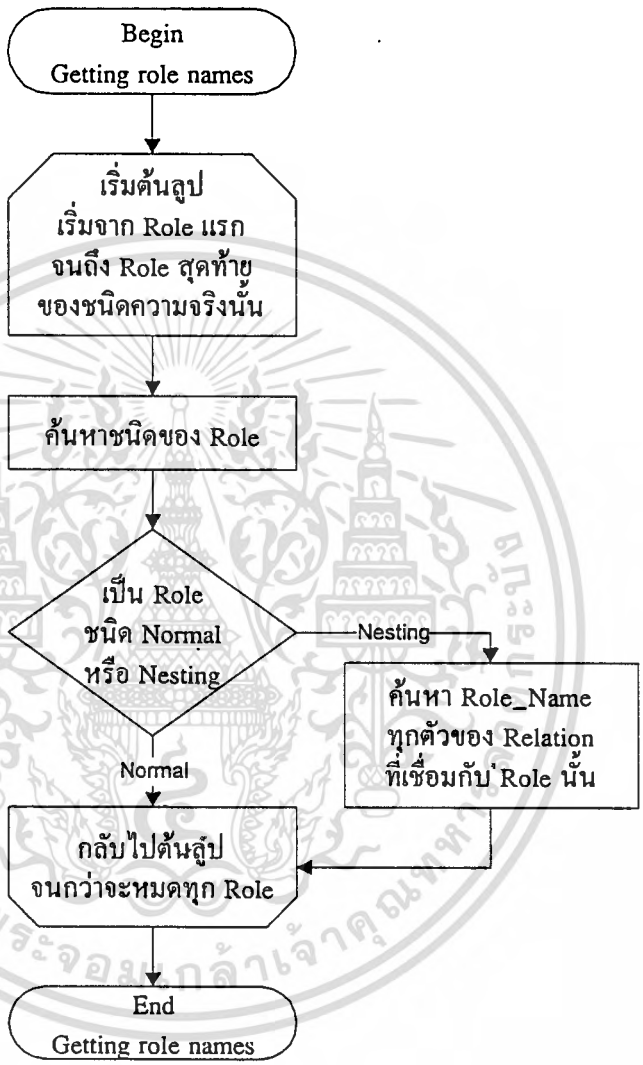
เมื่อเสร็จสิ้นการทำงานในขั้นตอนข้างต้นทั้งหมด เราก็จะมี Role_Name ทั้งหมดที่อยู่ในแต่ละชนิดความจริง ซึ่งจะดำเนินการสร้างวิวต่อไป การสร้างวิวจะเริ่มจากการหาชนิดเอนติตี้ของแต่ละ Role โดยเริ่มจากตาราง Object_Role ที่ประกอบด้วย 2 แอตทริบิวต์ คือ Role_Name และ Object_Name (ซึ่ง Object_Name ในกรณีนี้จะเป็นได้เพียงชนิดเอนติตี้เท่านั้น) จากนั้นก็จะตรวจสอบว่าชนิดเอนติตี้ที่ได้นี้เป็นซับไทป์ของชนิดเอนติตี้อื่นหรือไม่ หากเป็นซับไทป์ของชนิดเอนติตี้อื่น ก็จะต้องค้นหาขึ้นไปถึงซูเปอร์ไทป์ลำดับสูงสุด เพราะซูเปอร์ไทป์ลำดับสูงสุดเท่านั้นที่จะมีข้อมูลเก็บอยู่ในตารางจริง ๆ

ต่อจากนั้นก็จะนำ Role_Name, Relation_Name และ Entity ที่ได้ไปค้นหาข้อมูลในตาราง UNIQ_REF ซึ่งก็จะได้ชื่อของชนิดเลเบิลและชื่อของคอลัมน์ในตารางออกมา ซึ่งก็จะนำชนิดเลเบิลไปหาข้อมูลในตาราง LABEL_TYPE อีกทีหนึ่ง เพื่อตรวจสอบลำดับของชนิดเลเบิ้ลนั้น ในกรณีที่หนึ่งชนิดเอนติตี้มีมากกว่า 1 ชนิดเลเบิล ซึ่งหมายถึงว่าชนิดเลเบิลทั้งหมดนั้นจะทำหน้าที่เป็น Unique Identifier ร่วมกัน และท้ายสุดก็จะตรวจสอบชื่อคอลัมน์ที่ทำหน้าที่เป็นไพรมารีคีย์ โดยค้นหาจากตาราง Primary_Key จากนั้นก็นำข้อมูลทั้งหมดมาเก็บเข้าในลิงก์ลิสต์ ที่ได้กล่าวไปแล้วข้างต้น และทำเช่นนี้จนครบทุกโรลที่อยู่ในชนิดความจริงเดียวกัน

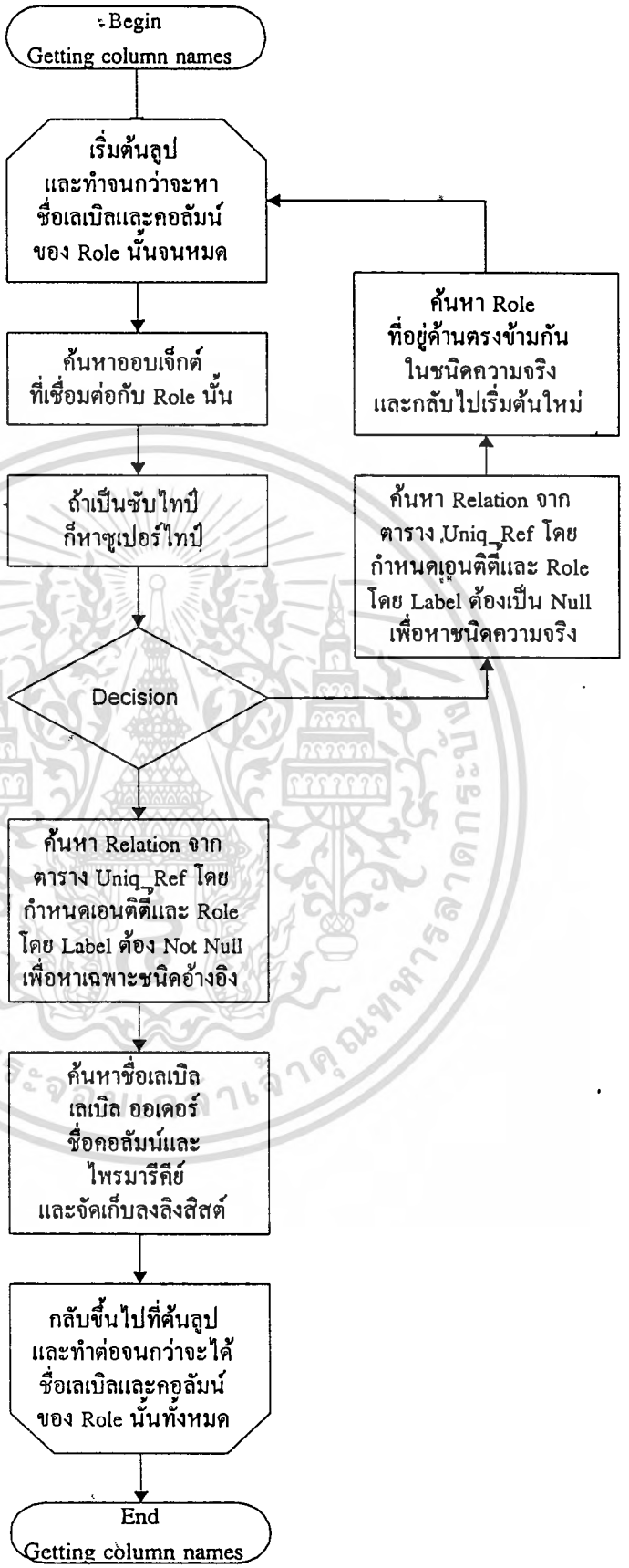
และเมื่อถึงจุดนี้ก็หมายความว่า เราได้ข้อมูลที่ต้องการทั้งหมดแล้ว และจะดำเนินการสร้างวิว โดยจะเริ่มจากค้นหาข้อมูลในลิงก์ลิสต์ และนำข้อมูลนั้นพักเก็บลงในตารางชื่อ VIEW_COLUMN ซึ่งเป็นตารางชั่วคราวที่สร้างขึ้นมาใหม่ จากนั้นจึงนำข้อมูลในตารางนี้ไปสร้างเป็นวิวอีกครั้ง ก็เป็นอันสิ้นสุดการทำงานในขั้นตอนนี้ และการทำงานทั้งหมดได้แสดงไว้ในโฟลว์ชาร์ตในรูปที่ 8-1, 8-2 และ 8-3 แล้ว




*รูปที่ 8-1 โฟลว์ชาร์ตของขั้นตอนทั้งหมด*




*รูปที่ 8-2 โฟลวชาร์ตรายละเอียดในส่วน Getting Role Names*



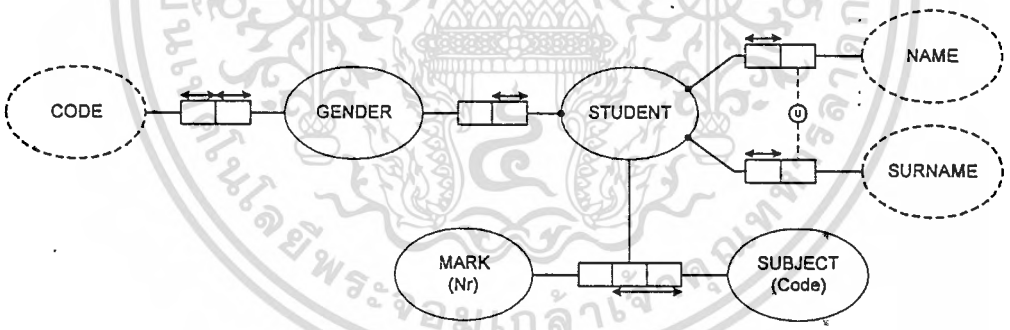

***รูปที่ 8-3 โฟลวชาร์ตในส่วน Getting Column Names***

อย่างไรก็ตาม ยังมีรูปแบบของชนิดเอนติตี้อีกรูปแบบหนึ่ง คือ แบบที่ชนิดความจริงมีชนิดเลเบิลที่เป็นแบบลำดับชั้น ซึ่งจะมีวิธีดำเนินการที่แยกต่างหากออกไป โดยจะมีวิธีทำงานดังต่อไปนี้ เริ่มจากการค้นหาข้อมูลจากตาราง REFERENCE ซึ่งจะได้ชื่อของ Relation_Name และชื่อของตาราง ต่อจากนั้นก็จะค้นหาข้อมูล Role_Name จากตาราง ROLE เช่นเดียวกับการทำงานในแบบแรก แต่แทนที่จะเป็น Role ที่ติดอยู่กับชนิดเอนติตี้ ก็จะเป็น Role ที่ติดอยู่กับชนิดความจริงแทน และจากนั้นก็นำข้อมูลทั้งหมดไปสร้างวิวเช่นเดียวกับในแบบแรก

**8.2 การทดสอบการทำงานของโปรแกรม**

และเพื่อจะทดสอบว่าโปรแกรมที่เราสร้างขึ้นนั้น สามารถทำงานได้อย่างถูกต้อง เราจะใช้วิธีทดสอบโดยการสร้างแบบทดสอบขึ้นมาเพื่อทดสอบในกรณีต่าง ๆ โดยแบบทดสอบที่จะสร้างขึ้นนี้ก็จะมีอยู่ 2 กรณีด้วยกัน คือ

8.2.1 แบบทดสอบที่เป็นแบบจำลองข้อมูลไนแอมอย่างง่าย ที่ประกอบด้วยชนิดความจริงแบบไบนารีและแบบเทอนารีและ n-ary โดยมีชนิดอ้างอิงของชนิดความจริงเหล่านั้นเป็นทั้งแบบที่มีชนิดเลเบิลเพียงตัวเดียว และแบบที่มีชนิดเลเบิลหลายตัว ในกรณีที่เป็นชนิดเลเบิลแบบหลายตัวนั้นก็จะมีทั้งแบบที่มี Unique Identifier เพียงตัวเดียว และแบบที่มี Unique Identifier ประกอบกันหลายตัว ซึ่งรูปแบบของไนแอมที่เราจะใช้ทดสอบดูได้จากรูปที่ 8-4



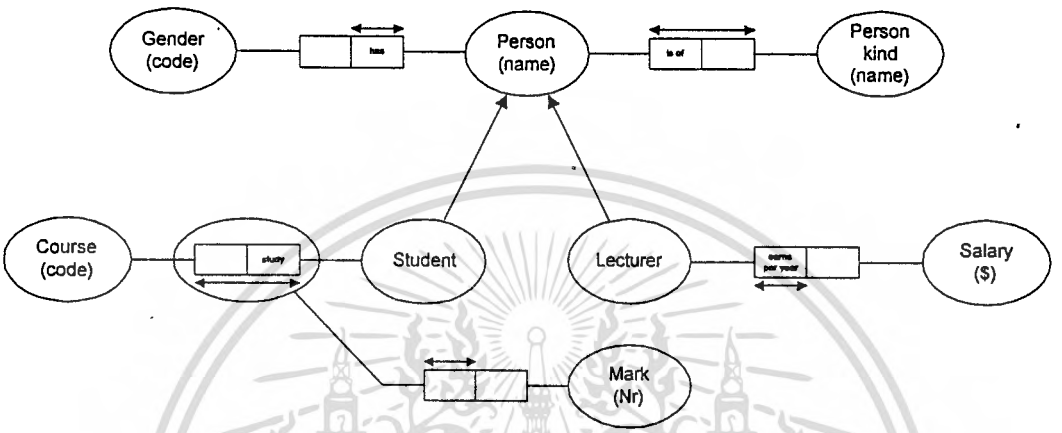

*รูปที่ 8-4 ตัวอย่างของแบบจำลองข้อมูลไนแอมที่ใช้ในการทดสอบ*

แบบจำลองข้อมูลที่ใช้ในการทดสอบ จะสามารถแมปเป็นตารางได้ 2 ตารางด้วยกัน คือ ตาราง STUDENT ประกอบด้วยคอลัมน์ STU_NAME (C20), STU_SERNAME(C20) และ GENDER (C1) โดยค่าในวงเล็บหมายถึงความยาวของแต่ละคอลัมน์ สำหรับตารางที่ 2 คือ ตาราง SUBJECT_MARK ซึ่งประกอบด้วยคอลัมน์ STU_NAME (C20), STU_SERNAME(C20), SUBJECT(C5) และ MARK (C2) จากนั้นก็จะนำเอาแบบจำลองข้อมูลไปเก็บในเมต้าสกีมาที่อยู่ในรูปของรีเลชันแนล แต่เนื่องจากข้อมูลในส่วนเมต้าสกีมามีจำนวนค่อนข้างมาก ดังนั้นจึงได้นำไปไว้ในภาคผนวก ง. แทน

จากนั้นก็เรียกโปรแกรมส่วนนิยามข้อมูลขึ้นมาทำงาน และจากผลการทำงานของโปรแกรม เราก็จะได้วิวเกิดขึ้นมา 2 วิว โดยแต่ละวิวจะแทนแต่ละชนิดความจริง โดยวิวแรกจะประกอบด้วยคอลัมน์ NAME (C20), SURNAME (C20) และ CODE (C1) สำหรับวิวที่ 2 จะประกอบด้วยคอลัมน์ NAME

(C20), SURNAME (C20), SUBJ_CODE (C5) และ NR (C2) ซึ่งจะเห็นได้ว่าผลการทดลองที่ได้นี้ถูกต้องตามต้องการ

8.2.2 แบบทดสอบที่เป็นแบบจำลองข้อมูลไนแอม ที่ประกอบด้วยซับไทป์และซูเปอร์ไทป์ นอกจากนั้นยังรวมไปถึงชนิดความจริงที่เป็นแบบ Nesting ด้วย ซึ่งรูปแบบของไนแอมที่จะใช้ทดสอบดูได้จากรูปที่ 8-5




*รูปที่ 8-5 ตัวอย่างของแบบจำลองข้อมูลไนแอมที่ใช้ในการทดสอบ*

แบบจำลองข้อมูลที่ใช้ในการทดสอบ จะสามารถแมปเป็นตารางได้ 4 ตารางด้วยกัน คือ (1) ตาราง PERSON ประกอบด้วยคอลัมน์ PERSON (C20) และ GENDER (C1) (2) ตาราง PERSON_ID ประกอบด้วยคอลัมน์ PERSON (C20) และ NAME (C5) (3) ตาราง ST_COURSE ประกอบด้วยคอลัมน์ STUDENT (C20), COURSE (C20) และ MARK (C5) และ (4) ตาราง LECTURER ประกอบด้วยคอลัมน์ LECTURER (C20) และ SALARY (N10) จากนั้นก็จะนำเอาแบบจำลองข้อมูลไปเก็บในเมต้าสกีมาที่อยู่ในรูปของรีเลชันแนล แต่เนื่องจากข้อมูลในส่วนเมต้าสกีมามีจำนวนค่อนข้างมาก ดังนั้นจึงได้นำไปไว้ในภาคผนวก ง. แทน

จากนั้นก็เรียกโปรแกรมส่วนนิยามข้อมูลขึ้นมาทำงาน และจากผลการทำงานของโปรแกรม เราก็จะได้วิวเกิดขึ้นมา 5 วิว โดยแต่ละวิวจะแทนแต่ละชนิดความจริง โดยวิวที่ 1 จะประกอบด้วยคอลัมน์ PERSON (C20) และ GENDER (C1) วิวที่ 2 ประกอบด้วยคอลัมน์ PERSON (C20) และ NAME (C5) วิวที่ 3 ประกอบด้วยคอลัมน์ STUDENT (C20) และ COURSE (C20) วิวที่ 4 ประกอบด้วยคอลัมน์ STUDENT (C20), COURSE (C20) และ MARK (C5) และวิวที่ 5 ประกอบด้วยคอลัมน์ LECTURER (C20) และ SALARY (N10) ซึ่งจะเห็นได้ว่าผลการสร้างวิวที่จะใช้แทนชนิดความจริงนี้ ถูกต้องตามต้องการทุกประการ

### 8.3 ส่วนจัดการข้อมูล (Information Manipulation)

การทำงานในส่วนนี้ ถือได้ว่าเป็นการทำงานหลักส่วนหนึ่งของระบบฐานข้อมูลอนุมาน เพราะเป็นส่วนที่ติดต่อกับผู้ใช้งานระบบฐานข้อมูลโดยตรง โดยส่วนจัดการข้อมูลนี้จะทำหน้าที่รับข้อมูลเข้าสู่ระบบ และลบข้อมูลออกจากระบบ โดยการนำข้อมูลเข้าเราจะใช้คำสั่งว่า Assert และการนำข้อมูลออกเรา

จะใช้คำว่า Remove ซึ่งสาเหตุที่เราไม่นำศัพท์ทางรีเลชันแนลอย่าง Insert, Update และ Delete มาใช้ก็เนื่องจากว่าการทำงานของระบบฐานข้อมูลอนุมานนั้น จะอ้างอิงกับชนิดความจริงเป็นหลัก ซึ่งแต่ละชนิดความจริงก็จะถือว่าเป็นหน่วยข้อมูลที่เล็กที่สุดอยู่แล้ว ดังนั้นสถานะของชนิดความจริงจึงมีเพียง 2 ประการ คือ มีหรือไม่มีชนิดความจริงนั้นเท่านั้น ดังนั้นการกระทำกับแต่ละชนิดความจริงจึงมี 2 คำสั่งไปด้วย และเพื่อให้แตกต่างกับฐานข้อมูลแบบรีเลชันแนล เราจึงกำหนดคำสั่งให้ต่างกันด้วย

สำหรับรูปแบบของชนิดความจริงนั้น เราจะขอยืมรูปแบบคำสั่งของภาษาโปรล็อกมาใช้ ทั้งนี้เนื่องจากแบบจำลองข้อมูลไนแอม ไม่ได้มีการกำหนดรูปแบบที่เป็นลักษณะของข้อความ (Text) เอาไว้ และจากความจริงที่ว่าชนิดความจริงมีความใกล้เคียงกับพรีดิเขตของโปรล็อกอยู่แล้ว ดังนั้นในการแทนชนิดความจริงเราจึงยืมรูปแบบของแฟกซ์ในภาษาโปรล็อกมาใช้ เช่น ชนิดความจริงที่ว่า นักศึกษาทุกคนต้องบันทึกเพศ ก็จะเขียนในรูปแบบของพรีดิเขตได้เป็น student_sex('John','M') เป็นต้น การทำงานในส่วนจัดการข้อมูลจะแบ่งออกเป็น 2 ส่วนด้วยกัน คือ ส่วนที่จัดการกับคำสั่ง Assert และส่วนที่จัดการกับคำสั่ง Remove

การทำงานของส่วนจัดการข้อมูลจะเริ่มจากการรับข้อมูลเข้ามา ซึ่งข้อมูลที่รับเข้ามานี้จะอยู่ในรูปของเร็กคอร์ดที่มีโครงสร้างข้อมูลดังนี้

```
struct {
        char    cmd_type;
        char    pred_name[20];
        int     arg_num;
        char    argument[10][20];
        int     end_flag;
    } predicate[20];
```

โครงสร้างข้อมูลนี้จะสามารถเก็บคำสั่งได้ทั้งหมด 20 คำสั่งด้วยกัน เพราะมีโครงสร้างในลักษณะของอาเรย์ การส่งคำสั่งเข้ามาจะสามารถส่งเข้ามาทีละคำสั่งหรือส่งมาทีละหลายคำสั่งก็ได้ ขึ้นอยู่กับลักษณะการใช้งานของชนิดความจริงนั้น โดยมีตัวแปร end_flag ทำหน้าที่ระบุจุดจบในแต่ละชุดของพรีดิเขต สำหรับสาเหตุที่ต้องทำให้สามารถรับได้ทีละคำสั่งหรือหลายคำสั่งเช่นนี้ ก็เนื่องจากการทำงานในส่วนจัดการข้อมูลนี้จะเกี่ยวข้องกับการตรวจสอบข้อบังคับ (Constraint Handling) ด้วย

ในการตรวจสอบข้อบังคับของคอนสแตนต์ต่าง ๆ นั้น เราจะใช้วิธีตรวจสอบทุกครั้งที่เสร็จสิ้นแต่ละคำสั่งไม่ได้ เพราะการ Assert หรือ Remove แต่ละครั้งอาจจะไม่ได้การทำงานที่สมบูรณ์ก็ได้ ตัวอย่างที่เห็นได้ชัดก็คือ กรณีที่มีการบังคับใช้ Occurrence Constraint โดยอาจจะกำหนดค่าเป็น 3 ดังนั้นชนิดความจริงที่บังคับโดยคอนสแตนต์นี้จะต้องนับได้ 3 เสมอ และนั่นก็หมายความว่าการ Assert หรือ Remove ก็จะต้องทำครั้งละ 3 ด้วย การ Assert ชนิดความจริง 1 ครั้งและตรวจสอบคอนสแตนต์เลย จึงไม่สามารถใช้กับกรณีนี้ได้

และด้วยเหตุดังกล่าว เราจึงอนุญาตให้การ Assert หรือ Remove สามารถทำงานในลักษณะของ Transection ได้ โดยการทำ Assert หรือ Remove จะทำต่อเนื่องไปเรื่อย ๆ จนกว่าจะหมดคำสั่งในลิงก์ลิสต์ เช่น ในกรณีของคอนสแตนต์ข้างต้นนั้น ลิงก์ลิสต์ที่ส่งมาก็จะต้องส่งมาครั้งละ 3 เช่นกัน และทั้งหมดที่กล่าวมานั้นก็หมายความว่าการตรวจสอบคอนสแตนต์แต่ละครั้ง จะตรวจสอบเมื่อหมดคำสั่งที่ส่งมาในลิงก์ลิสต์แต่ละครั้งเท่านั้น

### 8.3.1 ส่วนนำข้อมูลเข้าในฐานข้อมูล (Assert)

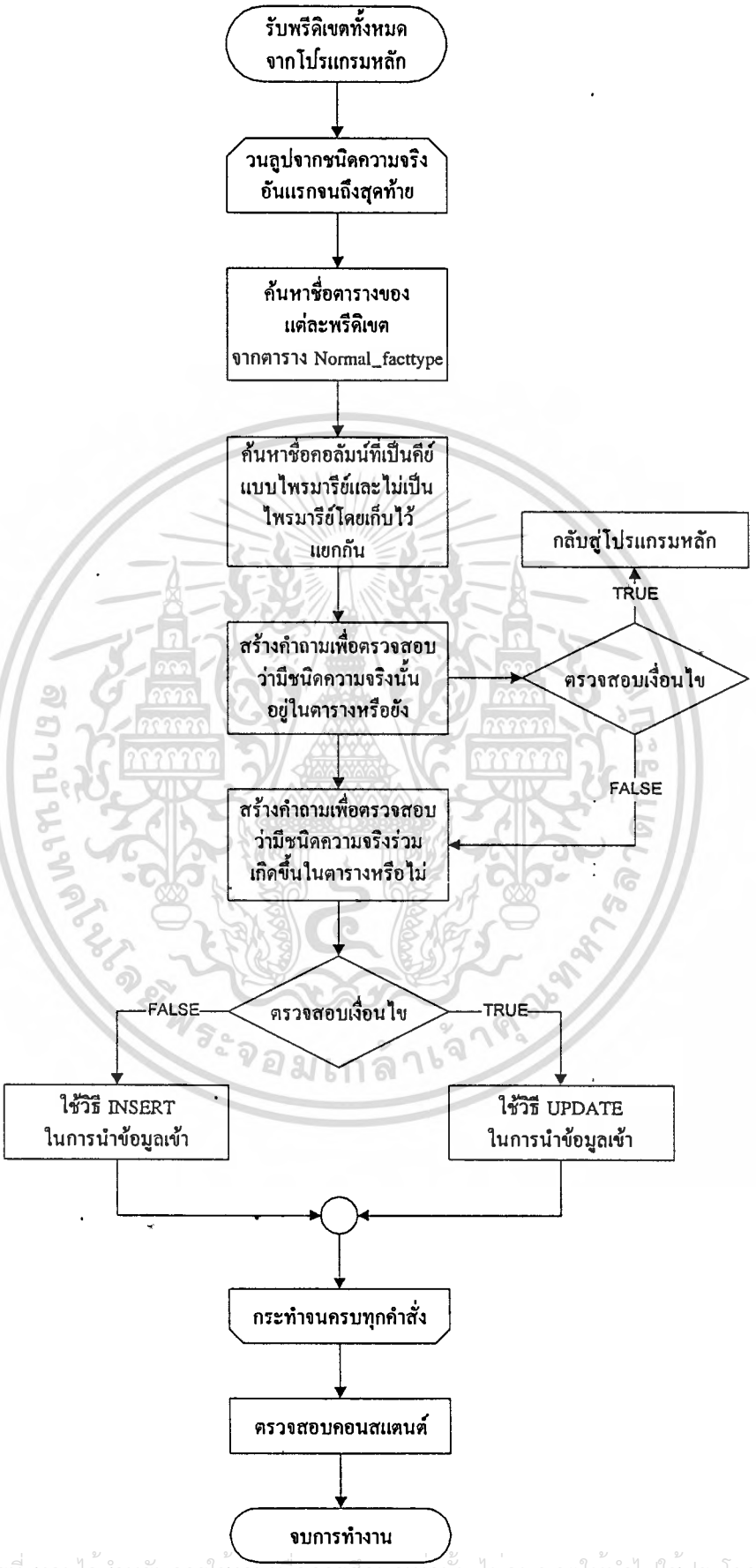
การทำงานในส่วนนำเข้าข้อมูลจะเริ่มต้นด้วย การนำเอาชื่อของชนิดความจริงมาค้นหาชื่อของตารางที่เก็บชนิดความจริงนั้นอยู่จากตาราง NORMAL_FACTTYPE จากนั้นก็กำหนดเคอร์เซอร์ขึ้นมาเพื่อค้นหาชื่อของคอลัมน์และลำดับของคอลัมน์ เฉพาะที่เป็นไพรมารีคีย์ของตารางที่ได้ค้นหาไว้ก่อนหน้านี้ และเมื่อค้นหาคอลัมน์ที่เป็นไพรมารีคีย์แล้วก็ค้นหาคอลัมน์ที่ไม่ได้เป็นไพรมารีคีย์มาด้วย แต่จะเก็บเอาไว้แยกกัน โดยจะค้นหาได้จากตาราง VIEW_COLUMN

เมื่อได้ทั้งชื่อตารางและชื่อของคอลัมน์มาหมดแล้ว ก็จะสร้างคิวรี่เพื่อตรวจสอบว่ามีชนิดความจริงที่มีค่าตรงกันกับชนิดความจริงที่ต้องการ Assert ในตารางหรือไม่ โดยใช้คิวรี่แบบ Dynamic SQL ซึ่งหากพบว่ามีชนิดความจริงอยู่ก็ย่อมจะหมายความว่าเกิดการซ้ำซ้อนขึ้น ซึ่งย่อมจะขัดแย้งกับหลักการของไนแอม ดังนั้นเราจะไม่ Assert ชนิดความจริงนั้นเข้าไป โดยฟ้องข้อผิดพลาดว่าเกิดการขัดแย้งในฐานข้อมูล โดยอาจจะถือว่าเป็นกรณีหนึ่งของคอนสเตนต์แบบ Uniqueness ก็ได้

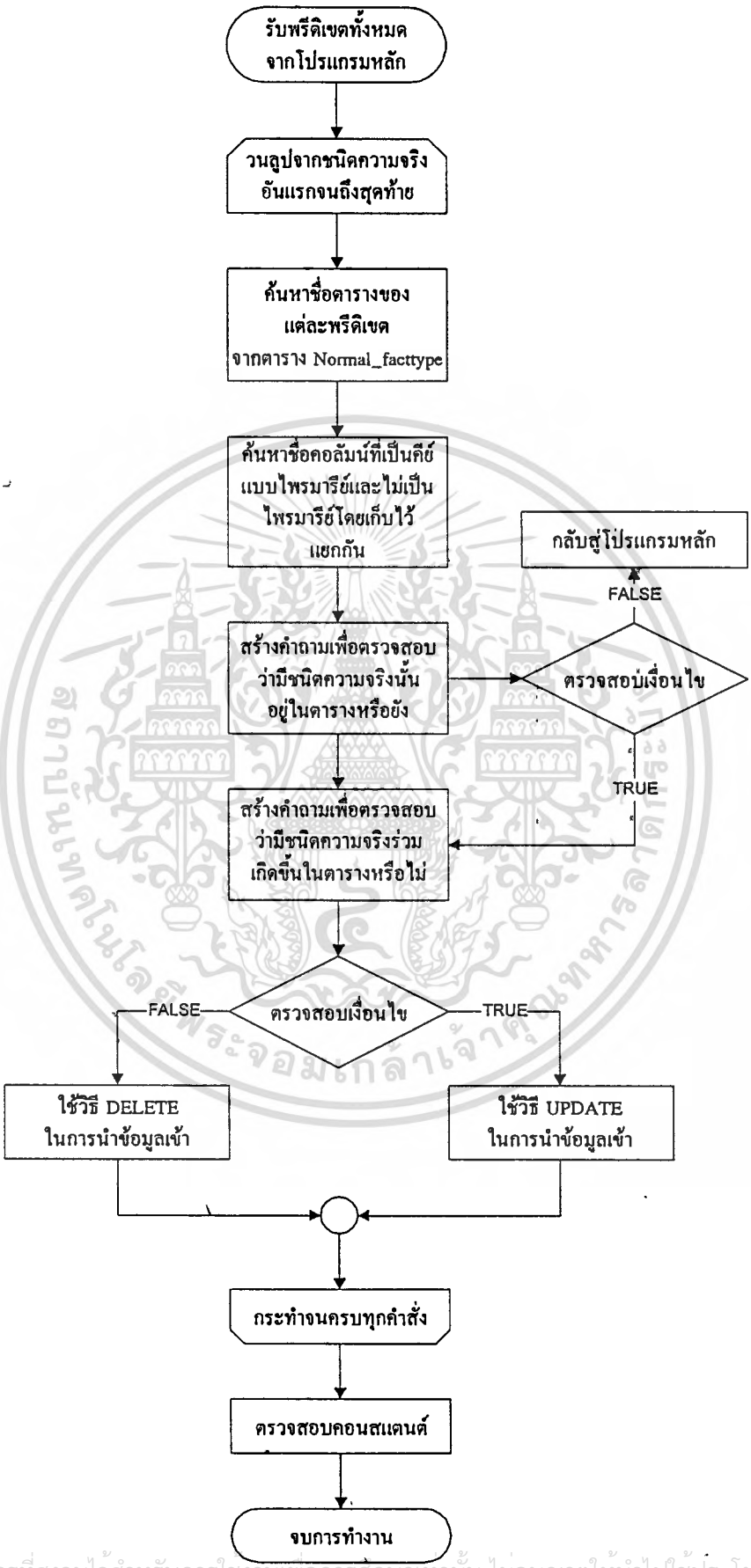
เมื่อไม่พบว่าเกิดการซ้ำขึ้น เราก็สามารถจะ Assert ชนิดความจริงนั้นลงในฐานข้อมูลได้ แต่ก่อนอื่นจะต้องตรวจสอบเสียก่อนว่ามีชนิดความจริงร่วมเกิดขึ้นหรือยัง คำว่าชนิดความจริงร่วมนั้นหมายความถึงชนิดความจริงอื่น ๆ ที่มีไพรมารีคีย์ร่วมกันกับชนิดความจริงนี้ เพราะหากมีชนิดความจริงร่วมเกิดขึ้นแล้ว ก็ย่อมจะหมายความว่าในตารางที่เก็บข้อมูลจริง ๆ มีแถวของข้อมูลที่มีไพรมารีย์คีย์เกิดขึ้นแล้ว ดังนั้นการ Assert ก็จะต้องใช้รูปแบบของการ Update และหากไม่มีชนิดความจริงร่วมก็จะใช้รูปแบบของการ Insert แทน ซึ่งการทำงานทั้งหมดสามารถเขียนเป็นโฟลว์ชาร์ตได้ดังรูปที่ 8-6

### 8.3.2 ส่วนนำข้อมูลออกจากฐานข้อมูล (Remove)

การทำงานในส่วนนำข้อมูลออกจากฐานข้อมูลจะเริ่มด้วย การนำเอาชื่อของชนิดความจริงมาค้นหาชื่อของตารางที่เก็บชนิดความจริงนั้นอยู่จากตาราง NORMAL_FACTTYPE จากนั้นก็กำหนดเคอร์เซอร์ขึ้นมาเพื่อค้นหาชื่อของคอลัมน์และลำดับของคอลัมน์ เฉพาะที่เป็นไพรมารีคีย์ของตารางที่ได้ค้นหาไว้ก่อนหน้านี้ และเมื่อค้นหาคอลัมน์ที่เป็นไพรมารีคีย์แล้วก็ค้นหาคอลัมน์ที่ไม่ได้เป็นไพรมารีคีย์มาด้วย แต่จะเก็บเอาไว้แยกกัน




*รูปที่ 8-6 แสดงโฟลวชาร์ตการทำงานของส่วนนำข้อมูลเข้า (ASSERT)*



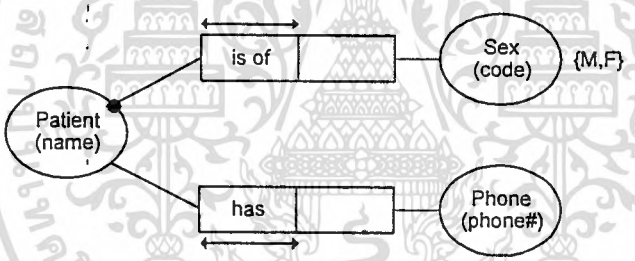

*รูปที่ 8-7 แสดงโฟลว์ชาร์ตของส่วนนำข้อมูลออก (REMOVE)*

เมื่อได้ทั้งชื่อตารางและชื่อของคอลัมน์มาหมดแล้ว ก็จะสร้างคิวรี่เพื่อตรวจสอบว่ามีชนิดความจริงที่มีค่าตรงกันกับชนิดความจริงที่ต้องการ Remove ในตารางหรือไม่ โดยใช้คิวรี่แบบ Dynamic SQL ซึ่งหากพบว่ามีชนิดความจริงอยู่ก็ย่อมจะหมายความว่าการ Remove ข้อมูลสามารถทำได้ แต่หากไม่มีชนิดความจริงนั้นอยู่ก็ย่อมจะไม่สามารถ Remove ได้ โดยฟ้องข้อผิดพลาดว่าเกิดการขัดแย้งในฐานข้อมูล

จากนั้นก็จะตรวจสอบว่ามีชนิดความจริงร่วมเกิดขึ้นในตารางหรือไม่ เช่นเดียวกับในการทำงานในส่วนนำข้อมูลเข้า โดยที่หากมีชนิดความจริงร่วมเกิดขึ้นแล้ว ก็ย่อมจะหมายความว่าในตารางที่เก็บข้อมูลมีชนิดความจริงอื่น ๆ อยู่ด้วย ดังนั้นการ Remove ก็จะต้องใช้รูปแบบของการ Update และหากไม่มีชนิดความจริงร่วมก็จะใช้รูปแบบของการ Delete แทน โดยการทำงานทั้งหมดสามารถเขียนเป็นโฟลว์ชาร์ตได้ดังรูปที่ 8-7

## 8.4 การทดสอบส่วนจัดการกับข้อมูล

หลังจากที่เราได้สร้างโปรแกรมขึ้นมาแล้ว เราก็จะทดสอบว่าโปรแกรมที่เราสร้างขึ้นนั้น สามารถทำงานได้อย่างถูกต้อง โดยจะใช้วิธีทดสอบโดยการสร้างแบบทดสอบขึ้นมาเพื่อทดสอบในกรณีต่าง ๆ โดยจะสร้างแบบจำลองข้อมูลแบบง่าย ๆ ขึ้นมา ดังรูปที่8-8




*รูปที่ 8-8 แบบจำลองข้อมูลที่ใช้ในการทดสอบส่วนจัดการข้อมูล*

จากแบบจำลองข้อมูลในรูปที่ 8-8 เราจะแปลงแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 8-8 ซึ่งจะได้ตารางเพียงตารางเดียวตั้งชื่อว่า PATIENT โดยจะมี 3 คอลัมน์ คือ Patient , Sex และ Phone จากนั้นเราก็จะเรียกโปรแกรมในส่วนนิยามข้อมูล เพื่อสร้างวิวให้กับชนิดความจริงทั้งสองชนิดความจริง

จากนั้นจะเริ่มทำการทดสอบโดยสมมติคำสั่งขึ้นมา โดยคำสั่งจะมีทั้งคำสั่งทั้งชนิดคำสั่งเดี่ยวและคำสั่งแบบกลุ่ม (Transection) และจะต้องมีทั้งคำสั่ง Assert และ Remove โดยคำสั่งทั้งหมดเราจะให้มีคำสั่งทั้งในแบบที่ถูกต้องและคำสั่งที่ผิดพลาด โดยที่หลังจากการทดสอบแต่ละครั้ง เราจะใช้โปรแกรม SQL*Plus เข้าไปตรวจสอบข้อมูลว่าถูกต้องตามที่ควรจะเป็นหรือไม่ ซึ่งหลังจากการทดสอบทุกครั้งก็พบว่าข้อมูลมีความถูกต้องตามต้องการ

# บทที่ 9
# ส่วนตรวจสอบความถูกต้องของข้อมูล

## 9.1 บทนำ

จากบทที่ผ่านมาเราได้กล่าวถึงการทำงานในส่วนนิยามข้อมูลและส่วนจัดการข้อมูลไปแล้ว สำหรับในบทนี้จะกล่าวถึงการทำงานในส่วนตรวจสอบความถูกต้องของข้อมูล (Constraint Checking) ตามวิธีการของไนแอมที่ได้กล่าวไว้ในบทที่ 3 การทำงานของส่วนตรวจสอบความถูกต้องของข้อมูลนี้จะแบ่งออกเป็น 2 ส่วนด้วยกัน คือ ส่วนเตรียมการ และ ส่วนตรวจสอบ โดยส่วนเตรียมการจะเป็นโปรแกรมย่อยที่เรียกใช้ในขั้นตอนการนิยามข้อมูล และส่วนตรวจสอบจะเรียกใช้ในขั้นตอนการจัดการข้อมูล

สำหรับสาเหตุที่มีการแยกการทำงานของส่วนตรวจสอบความถูกต้องของข้อมูลออกเป็น ส่วนเตรียมการและส่วนตรวจสอบนั้น ก็เนื่องจากหลังจากที่ได้กำหนดเมตาสกีมาเรียบร้อยแล้ว คอนสแตนต์ต่าง ๆ จะไม่มีการเปลี่ยนแปลงใด ๆ อีก ซึ่งนั่นก็หมายความว่ารูปแบบของการตรวจสอบคอนสแตนต์นั้น ๆ ก็จะไม่มีการเปลี่ยนแปลงไปด้วย จะมีส่วนที่เปลี่ยนแปลงก็เพียงแค่ข้อมูลเท่านั้นที่จะเปลี่ยนไป

ดังนั้นหากเราย้ายงานในส่วนที่ไม่เกี่ยวข้องกับข้อมูลมาไว้ในส่วนนิยามข้อมูล ก็จะทำให้มีการทำงานในขั้นตอนนั้น ๆ เพียงครั้งเดียวเท่านั้น เพราะหากเรานำงานทั้งหมดไปไว้ในส่วนจัดการข้อมูล การทำงานจะต้องเกิดขึ้นทุกครั้งที่มีการเปลี่ยนแปลงของข้อมูลที่เกี่ยวข้องกับคอนสแตนต์นั้น ๆ ซึ่งจะเห็นได้ว่าการทำงานในรูปแบบแรกประหยัดเวลาได้มากกว่า

เป็นที่น่าสังเกตว่าการทำงานทั้งหมดของส่วนตรวจสอบความถูกต้องของข้อมูลนี้ ไม่มีส่วนใดส่วนหนึ่งที่ทำงานเป็นเอกเทศเลย การทำงานในส่วนต่าง ๆ ล้วนแล้วแต่ถูกเรียกใช้งานโดยส่วนการทำงานอื่นทั้งสิ้น และเนื่องจากคอนสแตรนต์ที่ใช้ในโครงงานนี้ทั้งหมดมีอยู่ด้วยกัน 8 คอนสแตนต์ ดังนั้นการทำงานทั้งหมดของส่วนตรวจสอบความถูกต้องจึงสามารถแบ่งออกเป็น 16 ส่วนด้วยกัน

การอธิบายการทำงานของส่วนตรวจสอบความถูกต้องของข้อมูล นี้จะอธิบายไปทีละคอนสแตนต์ โดยในแต่ละคอนสแตนต์จะเริ่มด้วยการอธิบายลักษณะของคอนสแตนต์นั้น ๆ อย่างคร่าว ๆ จากนั้นจึงอธิบายในส่วนการทำงานของส่วนเตรียมการ การทำงานของส่วนตรวจสอบ และการทดสอบการทำงานของโปรแกรม ก็ขอให้ตกลงกันตามนี้นะครับ

การทำงานของส่วนเตรียมการนั้น จะเริ่มจากการตรวจสอบว่าในเมตาสกีมามีคอนสแตนต์อะไรบ้าง จากนั้นก็จะไล่การทำงานไปทุก ๆ คอนสแตนต์ โดยที่หากคอนสแตนต์นั้นเป็นชนิดใด ก็จะเรียกส่วนเตรียมการของคอนสแตนต์นั้นมาทำงาน ซึ่งก็จะได้ผลการทำงานเป็นข้อมูลที่จะใช้ในส่วนของการตรวจสอบต่อไป แต่สำหรับการทำงานของส่วนตรวจสอบนั้นจะต่างออกไป โดยส่วนตรวจสอบจะตรวจสอบจากข้อมูลที่ป้อนเข้ามาเท่านั้นว่าเกี่ยวข้องกับคอนสแตนต์ใดบ้าง จากนั้นก็เรียกคอนสแตนต์นั้น ๆ มาทำงานโดยอาศัยข้อมูลที่ได้เตรียมไว้ในขั้นตอนเตรียมการ และได้ผลลัพธ์เป็นอย่างไรก็จะส่งผ่านกลับไปยังส่วนจัดการข้อมูลต่อไป

ในบรรดาคอนสแตนด์ทั้งหมดนั้น หากจะมองในแง่ของรูปแบบการตรวจสอบ ก็อาจจะแบ่งการตรวจสอบคอนสแตนต์ออกได้เป็น 2 แบบด้วยกัน คือ คอนสแตนต์ที่สามารถที่จะทำการตรวจสอบได้ทันทีที่มีการป้อนข้อมูลเข้ามาแต่ละครั้ง ตัวอย่างของคอนสแตนต์เหล่านี้ได้แก่ Uniqueness, Mendatory สำหรับคอนสแตนต์อีกแบบหนึ่งนั้นจะไม่สามารถตรวจสอบได้ทันทีที่ป้อนข้อมูลแต่ละครั้ง แต่จะต้องรอจนกว่าผู้ใช้จะป้อนเข้ามาหมดทั้งชุด (Transection) เสียก่อนจึงจะสามารถตรวจสอบได้

ตัวอย่างของคอนสแตนต์เหล่านี้ได้แก่ Equality, Exclusion เป็นต้น ดังนั้นเพื่อความสะดวกต่อการทำงาน เราจะกำหนดให้มีการตรวจสอบคอนสแตนต์หลังจากที่มีการป้อนข้อมูลครบชุดแล้วเท่านั้น ซึ่งการทำงานเช่นนี้จะไม่ส่งผลให้การตรวจสอบคอนสแตนต์ในแบบแรกเกิดความผิดพลาดขึ้น

## 9.2 Uniqueness Constraints

จากที่ได้กล่าวไว้แล้วในบทที่ 3 ว่าคอนสแตนต์แบบ Uniqueness นั้นจะใช้ในกรณีที่ต้องการให้ข้อมูลที่ต้องการมีความเป็นหนึ่งเดียว คือ ไม่มีการซ้ำกันของข้อมูลใน Role นั้น (หรือกลุ่มของ Role) นอกจากนั้นคอนสแตนต์แบบ Uniqueness ยังสามารถแบ่งออกได้เป็น 2 ประเภท คือ แบบที่อยู่ในชนิดความจริงเดียวกัน (Inter-Facttype) และแบบที่อยู่คนละชนิดความจริงกัน (Intra-Facttype) ดังนั้นโปรแกรมที่เขียนก็จะต้องตรวจสอบทั้ง 2 ส่วนด้วย

### 9.2.1 ส่วนการเตรียมการ

การทำงานในส่วนเตรียมการจะเริ่มจากการกำหนด CURSOR เพื่อค้นหาข้อมูลในฐานข้อมูลที่มีคอนสเตรนท์แบบ Equality ทีละตัว โดยค้นหาจากตาราง CONSTRAINT เมื่อได้เลขคอนสเตรนท์มาแล้วก็ค้นหาข้อมูลจากตาราง CONST_ROLE เพื่อหาว่ามีกลุ่มของโรลใดที่ต่ออยู่กับคอนสเตรนท์ตัวนั้น หากมีกลุ่มโรลที่ต่อกับคอนสแตนต์ตัวนั้นมากกว่า 1 กลุ่ม ก็จะมีความหมายอยู่ 2 กรณี คือ เป็นคอนสแตนต์ Uniqueness แบบ Intra-Facttype หรือเป็นคอนสแตนต์ Uniqness ที่เชื่อมต่อกับชนิดอ้างอิง

หากพบว่ามีกลุ่มโรล 2 กลุ่ม ก็จะเริ่มทำงานกับกลุ่มโรลแรกก่อน โดยจะเริ่มจากการค้นหาข้อมูลที่เป็นชื่อโรลจากแต่ละกลุ่มโรล โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE_GROUP จากนั้นก็นำโรลที่ได้มาเป็นเงื่อนไข เพื่อหาว่าโรลนั้น ๆ อยู่บนความสัมพันธ์ใด โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE จากนั้นก็ค้นหาชนิดของความสัมพันธ์จากตาราง RELATION_TYPE ก็เป็นอันว่าเสร็จสิ้นการทำงานสำหรับกลุ่มโรลแรก จากนั้นก็ทำเช่นเดียวกันกับกลุ่มโรลที่สอง

เมื่อได้ชนิดความสัมพันธ์ของกลุ่มโรลทั้งสองแล้ว ก็จะดูว่าเป็นชนิดอ้างอิงหรือไม่ โดยหากเป็นชนิดอ้างอิงก็หมายความว่าเป็นคอนสแตนน์ของชนิดอ้างอิง ก็จะสร้างคิวรี่และจบการทำงาน แต่หากเป็นชนิดความจริงแล้ว ก็จะค้นหาชื่อโรลและชื่อเลเบิ้ลทั้งหมดที่อยู่ในชนิดความจริงทั้งสอง และสร้างวิวขึ้นมาโดยจะเป็นการ Join ของตารางที่บรรจุชนิดความจริงทั้งสอง โดยมีคอลัมน์ในวิวเป็นสมาชิกของกลุ่มโรลทั้งสอง และสร้างคิวรีเพื่อทดสอบโดยจะเป็นคำสั่ง SELECT COUNT(*) สมาชิกของวิวทั้งหมด ซึ่งหากไม่มีข้อมูลซ้ำ ผลลัพธ์ของคิวรีก็จะมีค่าเป็น 1 เสมอ

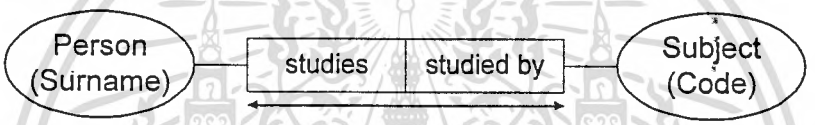
แต่ในกรณีที่มีกลุ่มโรลเพียงกลุ่มเดียวก็หมายความว่าเป็นคอนสแตนต์ Uniqueness แบบ Inter-Facttype ซึ่งก็จะสร้างวิวเช่นกัน แต่เป็นวิวของชนิดความจริงเดียว โดยมีสมาชิกเป็นเลเบิ้ลทั้งหมดของ

คอนสแตนต์นั้น ๆ จากนั้นก็จะสร้างคิวรีที่เป็นคำสั่ง SELECT COUNT(*) สมาชิกของวิวนั้นทั้งหมด ซึ่งหากไม่มีข้อมูลซ้ำ ผลการทำงานก็จะมีค่าเป็น 1 เสมอ

### 9.2.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำเลขของคอนสเตรนท์ที่ส่งมา มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนเตรียมการ ว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคภาษา SQL ที่เป็นลักษณะของ dynamic SQL และถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีการข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้นๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไปจนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.2.3 การทดสอบการทำงานของโปรแกรม




*รูปที่ 9-1 แบบจำลองข้อมูลที่ใช้ในการทดสอบคอนสแตนต์ Uniqueness*

จากแบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนล เพื่อเก็บลงในฐานข้อมูลในส่วนของเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-1 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวนสองคอลัมน์ด้วยกัน คือ Surname และ Code

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Uniqueness สามารถผ่านไปด้วยดี

## 9.3 Range Constraint

คอนสแตนต์นี้เป็นหนึ่งในคอนสแตนต์ Entity Type ซึ่งก็คือ คอนสแตนต์ที่ผูกอยู่กับเอนติตี้ใดเอนติตี้หนึ่งเท่านั้น โดยความหมายของคอนสแตนต์นี้แล้วก็คือ คอนสแตนต์ที่ระบุถึงขอบเขตที่เป็นไปได้ของข้อมูลที่อยู่ในเอนติตี้นั้น ๆ เช่น เอนติตี้คะแนนสอบก็อาจจะกำหนดว่ามีคะแนนระหว่าง 0 ถึง 100 เป็นต้น การกำหนดขอบเขตในคอนสแตนต์นี้โดยทั่วไปจะเป็นข้อมูลในลักษณะของตัวเลข ดังนั้นในวิทยานิพนธ์นี้ก็จะจำกัดข้อมูลไว้ว่าต้องเป็นตัวเลขเท่านั้น

### 9.3.1 ส่วนเตรียมการ

การทำงานของส่วนเตรียมการนี้ จะเริ่มจากการกำหนด CURSOR โดยใช้ Embeded SQL เพื่อค้นหาข้อมูลในฐานข้อมูลที่มีคอนสเตรนท์แบบ Range ทีละตัว โดยจะค้นหาจากตาราง CONSTRAINT

เมื่อได้เลขคอนสเตรนท์มาแล้วก็นำมาใช้เป็นเงื่อนไขในการค้นหาชนิดเลเบิ้ล โดยอาศัยข้อมูลจากตาราง OBJECT_CONST หลังจากนั้นก็ตรวจสอบข้อมูลที่อยู่ในตาราง UNIQ_REF ว่าเลเบิ้ลที่ใช้อ้างเอนทิตี้เป็นคอลัมน์จากการแปลงหรือไม่

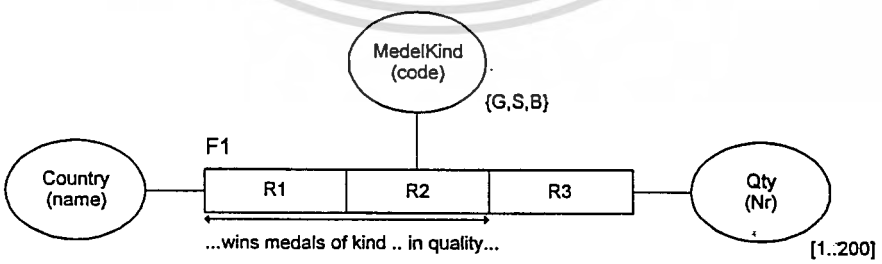
ถ้าเป็นก็จะกำหนด CURSOR เพื่อค้นหาข้อมูลที่เป็น Role ที่เกี่ยวข้องกับเอนทิตี้นั้น เมื่อได้ชื่อ Role แล้ว ก็จะนำมาเป็นเงื่อนไขในการค้นหา Relation จากตาราง ROLE จากนั้นก็จะใช้ Relation เป็นเงื่อนไขในการหาชื่อตารางจากตาราง NORMAL_FACTTYPE จากนั้นก็นำเอา ชื่อเอนทิตี้ ชื่อเลเบิ้ล และชื่อโรล มาหาชื่อคอลัมน์จากตาราง UNIQ_REF และหาข้อมูลที่เป็นตัวกำหนดคอนสเตรนท์แบบ Range จากตาราง RANGE_CONST จากนั้นนำข้อมูลที่จำเป็นมาสร้างคิวรี่ลงในตัวแปร query

และถ้าคิวรี่ที่สร้างได้มีความยาวของตัวอักษรมากกว่า 240 ตัวก็จะแบ่งคิวรี่ออกเป็น 2 ส่วนลงในตัวแปรที่ in_query และ in_query2 แล้ว Insert ลงในตาราง CONST_QUERY ในคอลัมน์ CONST# ซึ่งเป็นเลขคอนสเตรนท์ QUERY1 เป็นส่วนของคิวรี่ส่วนที่ 1 และ QUERY2 เป็นส่วนของคิวรี่ส่วนที่ 2 แต่ถ้าคิวรี่ที่สร้างได้ยาวไม่เกิน 240 ตัวก็ไม่ต้องแบ่งคิวรี่ออก และตัวแปร in_query2 ก็ไม่จำเป็นต้องใช้ Insert ลงตาราง CONST_QUERY ก็ไม่ต้องใส่ข้อมูลในคอลัมน์ QUERY2

### 9.3.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำเลขของคอนสเตรนท์ที่ส่งมา มาค้นหาคิวรีที่ได้สร้างแล้วจากส่วนเตรียมการ ว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคภาษา SQL ที่เป็นลักษณะของ dynamic SQL และถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีการข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้นๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไปจนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.3.3 การตรวจสอบการทำงานของโปรแกรม




***รูปที่ 9-2 แบบจำลองข้อมูลที่ใช้ในการทดสอบคอนสแตนต์ Range และ Membership***

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 โดยแบบจำลองข้อมูลนี้เราจะใช้ในการทดสอบคอนสแตนต์ Range และ Membership ไปพร้อม ๆ กัน ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก

ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-2 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวนสามคอลัมน์ด้วยกัน คือ Country, MedelKind และ Qty

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Range สามารถผ่านไปด้วยดี

### 9.4 Membership Constraint

คอนสแตนต์นี้เป็นหนึ่งในคอนสแตนต์ Entity Type เช่นเดียวกัน โดยความหมายแล้วคอนสแตนต์นี้จะหมายถึง คอนสแตนต์ที่ทำหน้าที่ระบุสมาชิกที่เป็นไปได้ของเอนติตี้ เช่น เอนติตี้ของเพศก็อาจจะระบุว่าจะต้องเป็นเพศหญิงหรือชาย (M or F) เท่านั้น ดังนั้นจะใส่ค่าอื่น ๆ ลงในเอนติตี้นี้ไม่ได้เป็นต้น โดยทั่วไปการกำหนดสมาชิกมักจะมีการกำหนดเป็นสตริงก์ ดังนั้นในวิทยานิพนธ์นี้จะจำกัดการใช้งานไว้เฉพาะสตริงก์เท่านั้น

#### 9.4.1 ส่วนเตรียมการ

การทำงานของส่วนเตรียมการนี้ จะเริ่มจากการกำหนด CURSOR โดยใช้ Embeded SQL เพื่อค้นหาข้อมูลในฐานข้อมูลที่มีคอนสเตรนท์แบบ Membership ทีละตัว โดยจะค้นหาจากตาราง CONSTRAINT เมื่อได้เลขคอนสเตรนท์มาแล้วก็นำมาใช้เป็นเงื่อนไขในการค้นหาชนิดเลเบิ้ล โดยอาศัยข้อมูลจากตาราง OBJECT_CONST หลังจากนั้นก็ตรวจสอบข้อมูลที่อยู่ในตาราง UNIQ_REF ว่าเลเบิ้ลที่ใช้อ้างเอนทิตี้หรือไม่ ซึ่งถ้าเป็นก็จะกำหนด CURSOR เพื่อค้นหาข้อมูลที่เป็น Role ที่เกี่ยวข้องกับเอนทิตี้นั้น เมื่อได้ชื่อ Role แล้ว ก็จะนำมาเป็นเงื่อนไขในการค้นหา Relation จากตาราง ROLE จากนั้นก็จะใช้ Relation เป็นเงื่อนไขในการหาชื่อตาราง จากตาราง NORMAL_FACTTYPE จากนั้นก็นำเอา ชื่อเอนทิตี้ ชื่อเลเบิ้ล และชื่อโรล มาหาชื่อคอลัมน์ โดยอาศัยข้อมูลจากตาราง UNIQ_REF และหาข้อมูลที่เป็นตัวกำหนดคอนสเตรนท์แบบ Membership จากตาราง RANGE_CONST จากนั้นนำข้อมูลที่จำเป็นมาสร้างคิวรี่ลงในตัวแปร query

และถ้าคิวรี่ที่สร้างได้มีความยาวของตัวอักษรมากกว่า 240 ตัวก็จะแบ่งคิวรี่ออกเป็น 2 ส่วนลงในตัวแปรที่ in_query และ in_query2 แล้ว Insert ลงในตาราง CONST_QUERY ในคอลัมน์ CONST# ซึ่งเป็นเลขคอนสเตรนท์ QUERY1 เป็นส่วนของคิวรี่ส่วนที่ 1 และ QUERY2 เป็นส่วนของคิวรี่ส่วนที่ 2 แต่ถ้าคิวรี่ที่สร้างได้ยาวไม่เกิน 240 ตัวก็ไม่ต้องแบ่งคิวรี่ออก และตัวแปร in_query2 ก็ไม่จำเป็นต้องใช้ Insert ลงตาราง CONST_QUERY ก็ไม่ต้องใส่ข้อมูลในคอลัมน์ QUERY2

#### 9.4.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบนี้จะเริ่มต้นด้วยการนำค่าที่ส่งมาซึ่งเป็นเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนเตรียมการว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่พบคิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคของภาษา

SQL ที่เป็นลักษณะของ dynamic SQL ถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้นๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไปจนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

**9.4.3 การทดสอบการทำงานของโปรแกรม**

การทดสอบการทำงานในคอนสแตนต์นี้จะใช้แบบจำลองจากรูปที่ 9-2 ซึ่งเป็นข้อมูลเดียวกับที่ใช้ในคอนสแตนต์ Range ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-2 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวนสามคอลัมน์ด้วยกัน คือ Country, MedelKind และ Qty

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Membership สามารถผ่านไปด้วยดี

**9.5 Mendatory Constraint**

คอนสแตนต์นี้เป็นคอนสแตนต์ที่ผูกอยู่กับ Role เพียง Role เดียวเท่านั้น โดยคอนสแตนต์นี้มีความหมายว่าให้มีการบังคับข้อมูลในเอนติตี้ฝั่งตรงข้าม เช่น หากมีชนิดความจริงที่เชื่อมต่อกับเอนติตี้ Person และ Sex โดยมีการระบุคอนสแตนต์ Mendatory ที่ฝั่งของ Person ก็จะมีความหมายว่าให้บังคับว่าจะต้องมีการบันทึกข้อมูล Sex สำหรับทุก ๆ เอนติตี้เสมอ คอนสแตนต์นี้จัดว่าเป็นคอนสแตนต์ที่มีการใช้งานมากในแบบจำลองข้อมูลไนแอม

**9.4.1 ส่วนเตรียมการ**

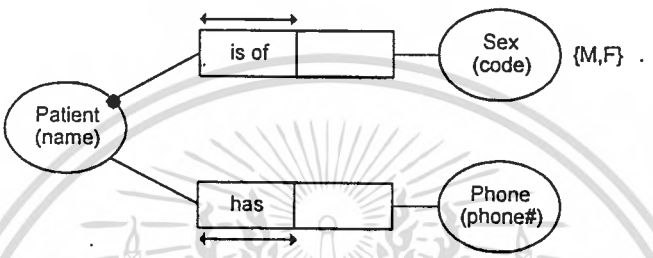
การทำงานของส่วนเตรียมการนี้จะเริ่มจากการกำหนด CURSOR เพื่อค้นหาเลขคอนสเตรนท์ที่มี KIND_OF_CONST เป็น MENDATORY จากนั้นก็จะค้นหาข้อมูลที่เป็น ROLE_GROUP ที่ต่ออยู่กับคอนสเตรนท์ตัวนั้นจากตาราง CONST_ROLE จากนั้นก็กำหนด CURSOR เพื่อค้นหาข้อมูลที่เป็น ROLE_NAME โดยอาศัย ROLE_GROUP ที่ได้เป็นเงื่อนไขจากตาราง ROLE_GROUP จากนั้นก็นำรีเลชันที่ได้มาเป็นเงื่อนไขในการหาโรลที่อยู่ในรีเลชันนั้นจากตาราง ROLE

เมื่อได้ชื่อชนิดความจริงก็จะหาว่าอยู่ในตารางใด โดยค้นหาจากตาราง NORMAL_FACTTYPE จากนั้นก็จะเริ่มสร้างคิวรี่ที่ 1 โดยจะเป็นคำสั่ง SELECT COUNT(*) คอลัมน์ทั้งหมดโดยมีเงื่อนไขว่าให้คอลัมน์ใดคอลัมน์หนึ่งเป็น NULL จากนั้นก็จะไปสร้างคิวรี่ที่ 2 ซึ่งจะเป็นคิวรี่ที่ใช้ในกรณีที่คอนสแตนต์หนึ่งครอบคลุมการทำงานในหลาย ๆ ชนิดความจริง ดังนั้นก็จะต้องสร้างวิวที่เป็น UNION ของข้อมูลในทุก ๆ ตาราง โดยจะ SELECT จากตารางที่ชนิดความจริงนั้นอยู่ จากนั้นก็จะตรวจสอบโดยวิธีนับคอลัมน์ที่เป็น NULL เช่นเดียวกัน เสร็จแล้วก็ Insert คิวรีทั้งสองลงในตาราง

### 9.4.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนเตรียมการว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข และถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีการข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไป จนกระทั่งไม่มีคิวรี่ใดอีกแล้วก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.4.3 การทดสอบการทำงานของโปรแกรม




*รูปที่ 9-3 แบบจำลองข้อมูลที่ใช้ทดสอบคอนสแตนต์ Mendatory*

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-3 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวนสามคอลัมน์ด้วยกัน คือ Patient, Sex และ Phone

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Mendatory สามารถผ่านไปด้วยดี

## 9.6 Occurrence Frequency

คอนสแตนต์นี้เป็นคอนสแตนต์ที่ใช้ระบุจำนวนครั้งในการบันทึกใน Role ใด ๆ หรือกลุ่มของ Role ใด ๆ เช่น สมมติว่ามีชนิดความจริงของคะแนนของนักศึกษา เราอาจจะกำหนดการบังคับว่าให้มีการบันทึกคะแนนปีการศึกษาละ 2 ครั้ง ในกรณีเช่นนี้จะต้องบังคับโดยคอนสแตนต์ Occurrence ซึ่งดู ๆ ไปอาจจะรู้สึกว่าคอนสแตนต์นี้ไม่จำเป็นต้องใช้ หรือไม่สำคัญมากนักก็ตาม แต่ถึงอย่างไรก็ช่วยให้ข้อมูลที่บันทึกในฐานข้อมูลมีความถูกต้องมากขึ้นได้

### 9.6.1 ส่วนเตรียมการ

การทำงานของส่วนเตรียมการนี้ จะเริ่มจากการกำหนด CURSOR เพื่อค้นหาเลขคอนสเตรนท์ที่มี KIND_OF_CONST เป็น OCCURENCY จากนั้นก็จะค้นหาข้อมูลที่เป็น ROLE_GROUP ที่ต่ออยู่กับคอนสเตรนท์ตัวนั้นจากตาราง CONST_ROLE จากนั้นก็กำหนด CURSOR เพื่อค้นหาข้อมูลที่เป็น

ROLE_NAME โดยอาศัย ROLE_GROUP ที่ได้เป็นเงื่อนไขจากตาราง ROLE_GROUP จากนั้นก็นำรีเลชันที่ได้มาเป็นเงื่อนไขในการหาโรลที่อยู่ในรีเลชันนั้นจากตาราง ROLE

จากนั้นก็นำโรลที่ได้มาเป็นเงื่อนไขในการหาเอนทิตี้ที่ต่ออยู่กับโรลนั้นจาก OBJECT_ROLE นำโรลและเอนทิตี้ที่ได้มาหาคอลัมน์จาก ENCOLUMN หาชื่อตารางจาก NORMAL_FACTTYPE ในระหว่างที่หาข้อมูลที่สำคัญ ก็นำมาต่อกันในตัวแปร query ที่เป็นตัวแปรที่เก็บคิวรี่ เมื่อไม่พบโรลที่ต่ออยู่กับคอนสเตรนท์แล้วก็หาข้อมูลที่เป็นขอบเขตบนและขอบเขตล่างของคอนสเตรนท์ตัวนั้น จากตาราง FREQ_CONST และนำมาต่อกับ query

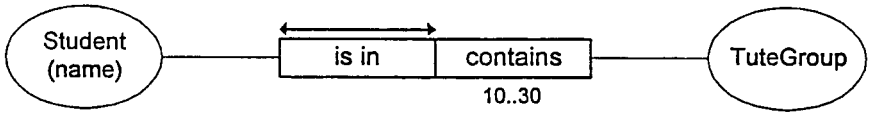
และถ้าคิวรี่ที่สร้างได้มีความยาวของตัวอักษรมากกว่า 240 ตัวก็จะแบ่งคิวรี่ออกเป็น 2 ส่วนลงในตัวแปรที่ in_query และ in_query2 แล้ว Insert ลงในตาราง CONST_QUERY ในคอลัมน์ CONST# ซึ่งเป็นเลขคอนสเตรนท์ QUERY1 เป็นส่วนของคิวรี่ส่วนที่ 1 และ QUERY2 เป็นส่วนของคิวรี่ส่วนที่ 2 แต่ถ้าคิวรี่ที่สร้างได้ยาวไม่เกิน 240 ตัวก็ไม่ต้องแบ่งคิวรี่ออก และตัวแปร in_query2 ก็ไม่จำเป็นต้องใช้ Insert ลงตาราง CONST_QUERY ก็ไม่ต้องใส่ข้อมูลในคอลัมน์ QUERY2 และหาคอนสเตรนท์ตัวต่อไปและทำงานเหมือนที่กล่าวมา จนกระทั่งไม่มีคอนสเตรนท์แบบนี้อีกก็จะออกจากโปรแกรม โดยที่ถ้าไม่เกิดข้อผิดพลาดใดๆเกิดขึ้น ก็จะส่งค่า 0 กลับ แต่ถ้าเกิดข้อผิดพลาดขึ้น ก็จะส่งค่า -1 กลับ

### 9.6.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำค่าที่ส่งมาซึ่งเป็นเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนเตรียมการว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคของภาษา SQL ที่เป็นลักษณะของ dynamic SQL ถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้นๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไปจนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.6.3 การทดสอบการทำงานของโปรแกรม

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-4 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวนสองคอลัมน์ด้วยกัน คือ Student และ TuteGroup




*รูปที่ 9-4 แบบจำลองข้อมูลที่ใช้ทดสอบคอนสแตนต์ Occurrence*

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Occurency สามารถผ่านไปด้วยดี

**9.7 Equality Constraint**

คอนสแตนต์นี้เป็นหนึ่งในคอนสแตนต์ที่เกี่ยวกับเซ็ต โดยคอนสแตนต์นี้จะมีความหมายว่าในระหว่าง Role ที่อยู่ต่างชนิดความจริงกัน หรือกลุ่มของ Role ที่อยู่ต่างชนิดความจริงกัน หากมีการบันทึกข้อมูลใน Role หรือกลุ่มของ Role ในชนิดความจริงหนึ่ง จะต้องมีการบันทึกข้อมูลในอีกชนิดความจริงด้วย จะบันทึกลงในชนิดความจริงอันหนึ่งอันใดไม่ได้ เช่น สมมติว่ามีชนิดความจริงของการเรียนวิชาคอมพิวเตอร์ โดยตั้งกฎไว้ว่าหากจะเรียนวิชานี้จะต้องบันทึกชั่วโมงเรียนในวิชาปฏิบัติการคอมพิวเตอร์ด้วย จะบันทึกลงในชนิดความจริงใดชนิดความจริงหนึ่งเพียงชนิดความจริงเดียวไม่ได้ กฎข้อบังคับในลักษณะนี้จะต้องใช้คอนสแตนต์แบบ Equality ในการบังคับ

**9.7.1 ส่วนเตรียมการ**

การทำงานในส่วนเตรียมการจะเริ่มจากการกำหนด CURSOR เพื่อค้นหาข้อมูลในฐานข้อมูลที่มีคอนสเตรนท์แบบ Equality ทีละตัว โดยค้นหาจากตาราง CONSTRAINT เมื่อได้เลขคอนสเตรนท์มาแล้วก็ค้นหาข้อมูลจากตาราง CONST_ROLE เพื่อหาว่ามีกลุ่มของโรลใดที่ต่ออยู่กับคอนสเตรนท์ตัวนั้น เมื่อได้ชื่อกลุ่มของโรลแล้ว ก็ค้นหาข้อมูลที่เป็นชื่อโรลจากแต่ละกลุ่มโรล โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE_GROUP จากนั้นก็นำโรลที่ได้มาเป็นเงื่อนไข เพื่อหาว่าโรลนั้น ๆ อยู่บนชนิดความจริงใด โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE

เมื่อได้ชื่อชนิดความจริงก็จะหาว่าชนิดความจริงนั้นอยู่ในตารางใด โดยอาศัยข้อมูลที่อยู่ในตาราง NORMAL_FACTTYPE แล้วนำตารางทั้งหมดที่ได้มาเปรียบเทียบว่าเป็นตารางเดียวกันหรือไม่ ถ้าไม่ใช่ก็แสดงว่า จากขั้นตอนการแปลงนั้นจะได้ชนิดความจริงดังกล่าวอยู่คนละตาราง ก็จะนำกลุ่มโรลที่ค้นหามาแล้ว ซึ่งได้เก็บไว้ในตัวแปรชื่อ group และ rgroup โดยจะนำค่าที่อยู่ในตัวแปร group มาเป็นเงื่อนไขในการหาโรลที่อยู่ในกลุ่มโรลที่อยู่ในตัวแปร group โดยใช้ข้อมูลจากตาราง ROLE_GROUP เมื่อได้โรลมาก็จะต้องกำหนด CURSOR เพื่อหาว่ามีเอนติตี้ใดต่ออยู่กับโรลนั้นบ้าง โดยอาศัยข้อมูลที่อยู่ในตาราง OBJECT_ROLE

จากนั้นก็นำข้อมูลที่เป็นชื่อโรลและชื่อของเอนติตี้มาค้นหาข้อมูลที่เป็นชื่อคอลัมน์และชื่อเลเบิ้ลจากตาราง UNIQ_REF เมื่อได้ชื่อคอลัมน์ก็จะนำไปรวมเข้ากับตัวแปร query ซึ่งเป็นตัวแปรที่เก็บคิวรี่ที่สร้าง จากนั้นก็จะต้องหาชื่อคอลัมน์ของอีกตารางหนึ่ง โดยการเขียนคิวรี่ที่มีเงื่อนไขว่า จะต้องมีชนิดเอนติตี้ชนิดเดียวกับที่หาได้ รวมทั้งชนิดเลเบิ้ล และต้องเป็นโรลที่อยู่ในกลุ่มโรลที่มีค่าเหมือนกับค่าที่อยู่ในตัวแปร rgroup

เมื่อได้แล้วก็นำมาต่อเข้าในตัวแปร query จนกระทั่งไม่มีชื่อคอลัมน์ที่ต่ออยู่กับโรลนั้นแล้ว ก็จะค้นหาโรลต่อไปที่อยู่ในกลุ่มโรลที่อ้างด้วยตัวแปร group และดำเนินการเช่นเดียวกัน จนกระทั่งไม่มีโรล

ใดที่อยู่ในกลุ่มโรลและไม่ได้อ้างถึงอีกแล้ว ก็จะได้คิวรี่ที่ใช้ในการตรวจสอบคอนสเตรนท์ Equality ก็จะใส่ข้อมูลที่ได้ลงในตาราง CONST_QUERY โดยจะตรวจสอบว่าคิวรี่ที่สร้างมีความยาวเกินกว่าความยาวสูงสุดที่แต่ละคอลัมน์ในออราเคิลจะรับได้ ถ้าเกินก็จะทำการแบ่งคิวรี่ออกเป็น 2 ส่วน และ Insert ข้อมูลลงในคอลัมน์ Query1 และ Query2 ตามลำดับ

แต่ถ้าพบว่าตารางที่ได้นั้นเป็นคนละตารางกันก็จะสร้างวิวที่เป็นการนำเอาคอลัมน์ที่อยู่ในแต่ละชนิดความจริงมาเป็นคอลัมน์ในวิว โดยที่ 1 ชนิดความจริงเป็น 1 วิว โดยเก็บชื่อวิวและเลขคอนสเตรนท์ที่เกี่ยวข้องกับวิวนั้น ลงในตาราง CONST_VIEW จากนั้น ก็จะตรวจสอบว่าในตารางดังกล่าวมีคอลัมน์ใดเป็นไพรมารีคีย์ ก็ไม่ต้องตรวจสอบข้อมูลในคอลัมน์นี้ เนื่องจากเป็นไพรมารีย์คีย์และวิวก็สร้างมาจากตารางเดียวกัน ก็จะต้องมีค่าเท่ากัน จากนั้นก็หาคอลัมน์ที่ไม่ได้เป็นไพรมารีคีย์และชนิดเลเบิ้ล และชนิดเอนติตี้จากวิวใดวิวหนึ่ง โดยอาศัยข้อมูลที่ได้จากตาราง UNIQ_REF

เมื่อได้คอลัมน์มาแล้วก็จะหาคอลัมน์จากอีกวิวหนึ่ง โดยมีเงื่อนไขว่าจะต้องเป็นชนิดเอนทิตี้ ชนิดเลเบิ้ล และ ชื่อโรลเดียวกัน และนำคอลัมน์ที่ได้มาต่อเข้าในคิวรี่ที่อยู่ในตัวแปร query จนกระทั่ง ไม่มีโรลและคอลัมน์ที่เกี่ยวข้องกับคอนสเตรนต์ตัวนั้นอีก ก็จะได้คิวรี่ออกมา จากนั้นก็ Insert ลงในตาราง CONST_QUERY โดยจะมีการตรวจสอบความยาวของคิวรี่เช่นเดียวกัน หลังจากนั้นก็จะค้นหาคอนสเตรนท์ที่เป็นชนิด Equality ตัวต่อไปจนกระทั่งไม่พบคอนสเตรนท์ที่เป็นแบบ Equality อีกแล้ว

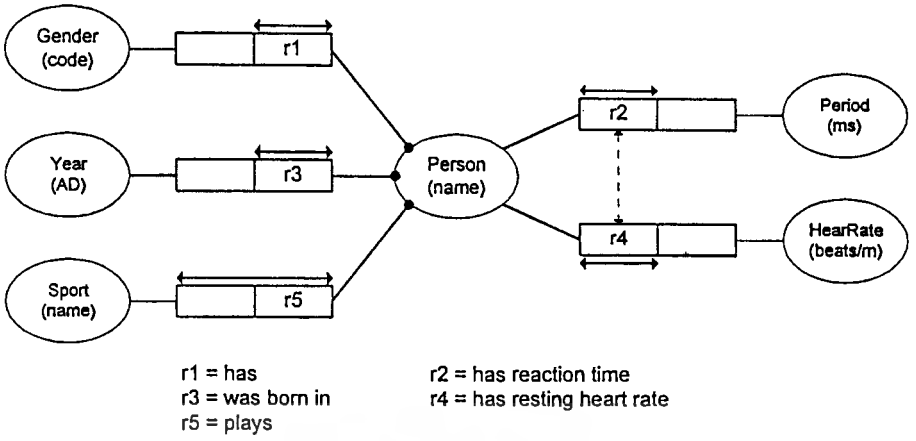
ในการสร้างคิวรี่ ถ้าไม่มีข้อผิดพลาดใดๆเกิดขึ้น ก็จะส่งค่า 0 กลับไปยังฟังก์ชันที่เรียกใช้งาน แต่ถ้าพบว่าเกิดข้อผิดพลาดขึ้น ก็จะส่งค่า -1 กลับ

### 9.7.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำค่าที่ส่งมาซึ่งเป็นเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนสร้างคิวรี่ ว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคของภาษา SQL ที่เป็นลักษณะของ dynamic SQL ถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้นๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไปจนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.7.3 การทดสอบการทำงานของโปรแกรม

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-5 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวน 5 คอลัมน์ด้วยกัน คือ Person, Gender, Year, Sport, Period และ HearRate




*รูปที่ 9-5 แบบจำลองข้อมูลที่ใช้ในการทดสอบคอนสแตนต์ Equality*

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Equality สามารถผ่านไปด้วยดี

## 9.8 Exclusion Constraint

คอนสแตนต์นี้เป็นอีกหนึ่งในคอนสแตนต์ที่เกี่ยวกับเซ็ต โดยคอนสแตนต์นี้มีความหมายตรงข้ามกับคอนสแตนต์ Equality โดยมีความหมายว่า ในระหว่าง Role ที่อยู่ต่างชนิดความจริงกัน หรือกลุ่มของ Role ที่อยู่ต่างชนิดความจริงกัน หากมีการบันทึกข้อมูลใน Role หรือกลุ่มของ Role ในชนิดความจริงหนึ่ง จะต้องไม่บันทึกข้อมูลในอีกชนิดความจริงหนึ่ง

### 9.8.1 ส่วนเตรียมการ

การทำงานในส่วนเตรียมการจะเริ่มจากการกำหนด CURSOR เพื่อค้นหาข้อมูลในฐานข้อมูลที่มีคอนสเตรนท์แบบ Exclusion ทีละตัว โดยค้นหาจากตาราง CONSTRAINT เมื่อได้เลขคอนสเตรนท์มาแล้วก็ค้นหาข้อมูลจากตาราง CONST_ROLE เพื่อค้นหาว่ามีกลุ่มโรลใดที่ต่ออยู่กับคอนสเตรนท์ตัวนั้น เมื่อได้ชื่อโรลกรุปแล้ว ก็ค้นหาข้อมูลที่เป็นชื่อโรลจากแต่ละกลุ่มโรล โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE_GROUP จากนั้นก็นำโรลที่ได้มาเป็นเงื่อนไข เพื่อหาว่าโรลนั้น ๆ อยู่บนชนิดความจริงใด โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE

เมื่อได้ชื่อชนิดความจริงก็จะหาว่าชนิดความจริงนั้นอยู่ในตารางใด โดยอาศัยข้อมูลที่อยู่ในตาราง NORMAL_FACTTYPE แล้วนำตารางทั้งหมดที่ได้ดังกล่าวมาเปรียบเทียบว่าเป็นตารางเดียวกันหรือไม่ ถ้าไม่ใช่ก็แสดงว่าจากขั้นตอนการแปลงนั้นจะได้ชนิดความจริงดังกล่าวอยู่คนละตาราง ก็จะนำกลุ่มโรลที่ได้ค้นหามาแล้ว ซึ่งได้เก็บไว้ในตัวแปรชื่อ group และ rgroup โดยจะนำค่าที่อยู่ในตัวแปร group มาเป็นเงื่อนไขในการหาโรลที่อยู่ในโรลกรุปที่เหมือนกับที่อยู่ในตัวแปร group โดยอาศัยข้อมูลจากตาราง ROLE_GROUP

เมื่อได้โรลมาก็จะต้องกำหนด CURSOR เพื่อหาว่ามีเอนติตี้ใดต่ออยู่กับโรลนั้นบ้าง โดยอาศัยข้อมูลที่อยู่ในตาราง OBJECT_ROLE จากนั้นก็นำข้อมูลที่เป็นชื่อโรลและชื่อของเอนติตี้มาค้นหาข้อมูลที่เป็นชื่อคอลัมน์และชื่อเลเบิ้ลจากตาราง UNIQ_REF เมื่อได้ชื่อคอลัมน์ก็จะนำรวมกับตัวแปร query จากนั้นก็จะต้องหาชื่อคอลัมน์ของอีกตารางหนึ่ง โดยการเขียนคิวรี่ที่มีเงื่อนไขว่า จะต้องมีชนิดเอนติตี้ชนิดเดียวกับที่หาได้ รวมทั้งชนิดเลเบิ้ล และต้องเป็นโรลที่อยู่ในกลุ่มโรลที่มีค่าเหมือนกับค่าที่อยู่ในตัวแปร rgroup

เมื่อได้แล้วก็นำมาต่อเข้าในตัวแปร query จนกระทั่งไม่มีชื่อคอลัมน์ที่ต่ออยู่กับโรลนั้นแล้ว ก็จะค้นหาโรลต่อไปที่อยู่ในกลุ่มโรลที่อ้างด้วยตัวแปร group และดำเนินการเช่นเดียวกัน จนกระทั่งไม่มีโรลใดที่อยู่ในกลุ่มโรลและไม่ได้อ้างถึงอีกแล้ว ก็จะได้คิวรี่ที่ใช้ในการตรวจสอบคอนสเตรนท์แบบ Exclusion ก็จะใส่ข้อมูลที่ได้ลงในตาราง CONST_QUERY โดยจะตรวจสอบว่าคิวรี่ที่สร้างมีความยาวเกินกว่าความยาวสูงสุดที่แต่ละคอลัมน์ในออราเคิลจะรับได้ ถ้าเกินก็จะแบ่งคิวรี่ออกเป็น 2 ส่วน และใส่ (insert) ข้อมูลลงในคอลัมน์ Query1 และ Query2 ตามลำดับ

แต่ถ้าพบว่า ตารางที่ได้นั้นเป็นคนละตารางกันก็จะสร้างวิว (view) ที่เป็นการนำเอาคอลัมน์ที่อยู่ในแต่ละชนิดความจริงมาเป็นคอลัมน์ในวิว โดย 1 ชนิดความจริงเป็น 1 วิว โดยจะเก็บชื่อวิวและเลขคอนสเตรนท์ที่เกี่ยวข้องกับวิวนั้น ลงในตาราง CONST_VIEW จากนั้น ก็จะตรวจสอบว่าในตารางดังกล่าวมีคอลัมน์ใดเป็นไพรมารีคีย์ ก็ไม่ต้องตรวจสอบข้อมูลในคอลัมน์นี้ เนื่องจากเป็นไพรมารีคีย์และวิวก็สร้างมาจากตารางเดียวกัน ก็จะต้องมีค่าเท่ากัน จากนั้นก็หาคอลัมน์ที่ไม่ได้เป็นไพรมารีคีย์และชนิดเลเบิ้ลและชนิดเอนทิตี้จากวิวใดวิวหนึ่ง โดยอาศัยข้อมูลที่ได้จากตาราง UNIQ_REF

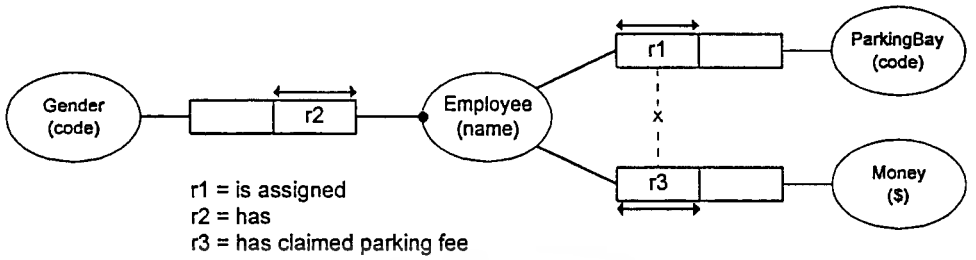
เมื่อได้คอลัมน์มาแล้วก็จะหาคอลัมน์จากอีกวิวหนึ่ง โดยมีเงื่อนไขว่า จะต้องเป็นชนิดเอนติตี้ชนิดเลเบิ้ล และ ชื่อโรลเดียวกัน และนำคอลัมน์ที่ได้มาต่อเข้าในคิวรี่ที่อยู่ในตัวแปร query จนกระทั่ง ไม่มีโรลและคอลัมน์ที่เกี่ยวข้องกับคอนสเตรนท์ตัวนั้นอีก ก็จะได้คิวรี่ออกมา จากนั้นก็ Insert ลงในตาราง CONST_QUERY โดยจะมีการตรวจสอบความยาวของคิวรี่เช่นเดียวกัน หลังจากนั้นก็จะค้นหาคอนสเตรนท์ที่เป็นชนิด Exclusion ตัวต่อไปจนกระทั่งไม่พบคอนสเตรนท์ที่เป็นแบบ Exclusion อีกแล้ว

ในการสร้างคิวรี่ ถ้าไม่มีข้อผิดพลาดใดๆเกิดขึ้น ก็จะส่งค่า 0 กลับไปยังฟังก์ชันที่เรียกใช้งาน แต่ถ้าพบว่าเกิดข้อผิดพลาดขึ้น ก็จะส่งค่า -1 กลับ

### 9.8.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบเริ่มต้นด้วยการนำค่าที่ส่งมาซึ่งเป็นเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างแล้วจากส่วนสร้างคิวรี่ว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคของภาษา SQL ที่เป็นลักษณะของ dynamic SQL ถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้น ๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไป จนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.8.3 การทดสอบการทำงานของโปรแกรม




*รูปที่ 9-6 แบบจำลองข้อมูลที่ใช้ในการทดสอบคอนสแตนต์ Exclusion*

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-6 ซึ่งจะได้ตารางเพียงตารางเดียว ที่มีคอลัมน์จำนวน 4 คอลัมน์ด้วยกัน คือ Employee, Gender, Parking_Bay, Money

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Exclusion สามารถผ่านไปด้วยดี

## 9.9 Subset Constraint

คอนสแตนต์นี้เป็นคอนสแตนต์สุดท้ายที่เกี่ยวกับเซ็ต โดยคอนสแตนต์นี้มีความหมายว่า ในระหว่าง Role ที่อยู่ต่างชนิดความจริงกัน หรือกลุ่มของ Role ที่อยู่ต่างชนิดความจริงกัน หากมีการบันทึกข้อมูลใน Role หรือกลุ่มของ Role ในชนิดความจริงที่เป็นซับเซ็ต จะต้องมีการบันทึกข้อมูลในอีกชนิดความจริงที่เป็นซูเปอร์เซ็ตด้วย

### 9.9.1 ส่วนเตรียมการ

ในส่วนเตรียมการจะเริ่มจากการกำหนด CURSOR เพื่อค้นหาข้อมูลที่มีคอนสเตรนต์แบบ Subset ทีละตัว โดยค้นหาจากตาราง CONSTRAINT เมื่อได้เลขคอนสเตรนท์มาแล้วก็ค้นหาข้อมูลจากตาราง CONST_ROLE เพื่อค้นหาว่ามีกลุ่มโรลใดที่ต่ออยู่กับคอนสเตรนท์ตัวนั้น เมื่อได้ชื่อกลุ่มโรลแล้วก็ค้นหาข้อมูลที่เป็นชื่อโรลจากแต่ละกลุ่มโรล โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE_GROUP จากนั้นก็นำโรลที่ได้มาเป็นเงื่อนไขเพื่อหาว่าโรลนั้น ๆ อยู่บนชนิดความจริงใด โดยอาศัยข้อมูลที่อยู่ในตาราง ROLE

เมื่อได้ชื่อชนิดความจริงก็จะหาว่า ชนิดความจริงนั้นอยู่ในตารางใด โดยอาศัยข้อมูลที่อยู่ในตาราง NORMAL_FACTTYPE แล้วนำตารางทั้งหมดที่ได้ดังกล่าวมาเปรียบเทียบว่า เป็นตารางเดียวกันหรือไม่ ถ้าไม่ใช่ ก็แสดงว่าจากขั้นตอนการแปลงนั้นจะได้ชนิดความจริงดังกล่าวอยู่คนละตาราง ก็จะนำกลุ่มโรลที่ได้ค้นหามาแล้วเก็บไว้ในตัวแปร group และ rgroup โดยจะนำค่าที่อยู่ในตัวแปร group มาเป็นเงื่อนไขในการหาโรลที่อยู่ในโรลกรุปที่เหมือนกันที่อยู่ในตัวแปร group โดยอาศัยข้อมูลจากตาราง ROLE_GROUP

เมื่อได้โรลมาก็จะต้องกำหนด CURSOR เพื่อหาว่ามีเอนติตี้ใดต่ออยู่กับโรลนั้นบ้าง โดยอาศัยข้อมูลที่อยู่ในตาราง OBJECT_ROLE จากนั้นก็นำข้อมูลที่เป็นชื่อโรลและชื่อของเอนทิตี้มาค้นหาข้อมูลที่เป็นชื่อคอลัมน์และชื่อเลเบิ้ลจากตาราง UNIQ_REF เมื่อได้ชื่อคอลัมน์ก็จะนำไปต่อเข้ากับตัวแปร query ซึ่งเป็นตัวแปรที่เป็นคิวรี่ที่สร้าง จากนั้นก็จะต้องหาชื่อคอลัมน์ของอีกตาราง โดยการเขียนคิวรี่ที่มีเงื่อนไขว่า จะต้องมีชนิดเอนทิตี้เดียวกับที่หาได้ รวมทั้งชนิดเลเบิ้ล และต้องเป็นโรลที่อยู่ในกลุ่มโรลที่มีค่าเหมือนกับค่าที่อยู่ในตัวแปร rgroup

เมื่อได้แล้วก็นำมาต่อเข้าในตัวแปร query จนกระทั่งไม่มีชื่อคอลัมน์ที่ต่ออยู่กับโรลนั้นแล้ว ก็จะค้นหาโรลต่อไปที่อยู่ในกลุ่มโรลที่อ้างด้วยตัวแปร group และดำเนินการเช่นเดียวกัน จนกระทั่งไม่มีโรลใดที่อยู่ในกลุ่มโรลและไม่ได้อ้างถึงอีกแล้ว ก็จะได้คิวรี่ที่ใช้ในการตรวจสอบคอนสเตรนท์แบบสับเซต ก็จะใส่ข้อมูลที่ได้ลงในตาราง CONST_QUERY โดยจะตรวจสอบว่าคิวรี่ที่สร้างมีความยาวเกินกว่าความยาวสูงสุดที่แต่ละคอลัมน์ในออราเคิลจะรับได้ ถ้าเกินก็จะแบ่งคิวรี่ออกเป็น 2 ส่วน และใส่ Insert ข้อมูลลงในคอลัมน์ Query1 และ Query2 ตามลำดับ

แต่ถ้าพบว่าตารางที่ได้นั้นเป็นคนละตารางกันก็จะสร้างวิวที่เป็นการนำเอาคอลัมน์ที่อยู่ในแต่ละชนิดความจริงมาเป็นคอลัมน์ในวิวโดย 1 ชนิดความจริงเป็น 1 วิว โดยจะเก็บชื่อวิวและเลขคอนสเตรนท์ที่เกี่ยวข้องกับวิวนั้น ลงในตาราง CONST_VIEW จากนั้น ก็จะตรวจสอบว่าในตารางดังกล่าวมีคอลัมน์ใดเป็นไพรมารีคีย์ ก็ไม่ต้องตรวจสอบข้อมูลในคอลัมน์นี้ เนื่องจากเป็นไพรมารีคีย์และวิวก็สร้างมาจากตารางเดียวกัน ก็จะต้องมีค่าเท่ากัน จากนั้นก็หาคอลัมน์ที่ไม่ได้เป็นไพรมารีคีย์และชนิดเลเบิ้ล และชนิด เอนทิตี้จากวิวใดวิวหนึ่ง โดยอาศัยข้อมูลที่ได้จากตาราง UNIQ_REF

เมื่อได้คอลัมน์มาแล้วก็จะหาคอลัมน์จากอีกวิวหนึ่ง โดยมีเงื่อนไขว่าจะต้องเป็นชนิดเอนทิตี้ ชนิดเลเบิ้ล และชื่อโรลเดียวกัน และนำคอลัมน์ที่ได้มาต่อเข้าในคิวรี่ที่อยู่ในตัวแปร query จนกระทั่งไม่มีโรลและคอลัมน์ที่เกี่ยวข้องกับคอนสเตรนท์ตัวนั้นอีก ก็จะได้คิวรี่ออกมา จากนั้นก็ Iinsert ลงในตาราง CONST_QUERY โดยจะมีการตรวจสอบความยาวของคิวรี่เช่นกัน หลังจากนั้นก็จะค้นหาคอนสเตรนท์ที่เป็นชนิดสับเซตตัวต่อไปจนกระทั่งไม่พบคอนสเตรนท์ที่เป็นแบบสับเซตอีกแล้ว

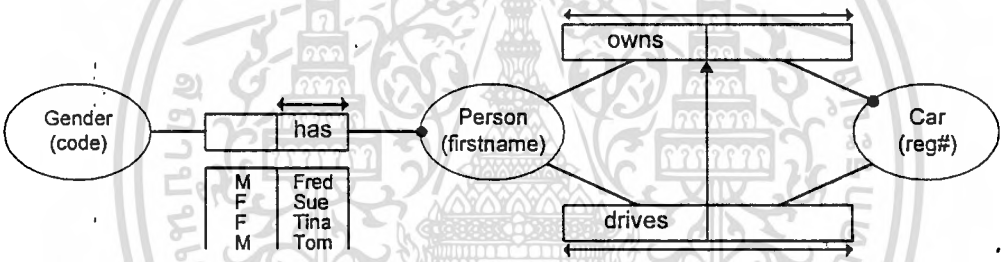
ในการสร้างคิวรี่ ถ้าไม่มีข้อผิดพลาดใดๆเกิดขึ้น ก็จะส่งค่า 0 กลับไปยังฟังก์ชันที่เรียกใช้งาน แต่ถ้าพบว่าเกิดข้อผิดพลาดขึ้น ก็จะส่งค่า -1 กลับ

### 9.9.2 ส่วนตรวจสอบ

การทำงานในส่วนตรวจสอบจะเริ่มต้นด้วยการนำค่าที่ส่งมาซึ่งเป็นเลขของคอนสเตรนท์ มาค้นหาคิวรี่ที่ได้สร้างจากส่วนเตรียมการ ว่าคอนสเตรนท์นั้นมีคิวรี่ใดบ้าง ถ้าไม่ได้คิวรี่ก็จะส่งค่า 0 กลับ แต่ถ้าพบคิวรี่ก็จะอ่านคิวรี่นั้นลงในตัวแปร query แล้วนำตัวแปรนี้มาเป็นเงื่อนไข ในประโยคของภาษาSQL ที่เป็นลักษณะของ dynamic SQL ถ้าปฏิบัติตามคิวรี่แล้วพบว่ามีข้อมูลที่ตรงตามคิวรี่ ซึ่งเป็นข้อมูลที่ขัดแย้งกับข้อกำหนดของคอนสเตรนท์นั้น ๆ ก็จะส่งค่า -1 กลับ แต่ถ้าไม่พบข้อมูลก็จะค้นหาคิวรี่ตัวต่อไป จนกระทั่งไม่มีคิวรี่ใดอีกแล้วที่เกี่ยวข้องกับคอนสเตรนท์ตัวนี้ก็จะส่งค่า 0 กลับยังฟังก์ชันที่เรียก

### 9.9.3 ส่วนตรวจสอบการทำงานของโปรแกรม

แบบจำลองข้อมูลที่เราใช้ทดสอบนี้ได้มาจากตัวอย่างในบทที่ 3 ซึ่งจากแบบจำลองข้อมูลนี้ เราสามารถแปลงให้อยู่ในรูปของรีเลชันแนลเพื่อเก็บลงในตารางเมตาสกีมา ซึ่งมีรายละเอียดดังตารางในภาคผนวก ง. และจากนั้นเราจะสร้างตารางที่แมปจากแบบจำลองในรูป 9-7 ซึ่งจะได้ตารางจำนวน 2 ตาราง (เฉพาะที่เกี่ยวข้องกับคอนสแตนต์) โดยตาราง OWN มี 2 คอลัมน์ คือ Person และ Car และตาราง DRIVE มี 2 คอลัมน์ คือ Person และ Car เช่นเดียวกัน




*รูปที่ 9-7 แบบจำลองข้อมูลที่ใช้ในการทดสอบคอนสแตนต์ Subset*

ซึ่งเมื่อถึงตรงนี้เราก็จะรันโปรแกรมในส่วนเตรียมการ เพื่อสร้างคิวรี่สำหรับตรวจสอบ และจากนั้นเราได้ทดลองป้อนข้อมูลหลาย ๆ แบบโดยตั้งใจทั้งให้ผ่านและไม่ผ่านการตรวจสอบคอนสแตนต์ และทดลองรันโปรแกรมในส่วนตรวจสอบ ซึ่งจากผลการทดสอบที่ได้ก็ถูกต้องตามที่คาดไว้ ดังนั้นจะถือว่าการทดสอบโปรแกรมตรวจสอบคอนสแตนต์ Subset สามารถผ่านไปด้วยดี

# บทที่ 10
# การจัดการกับกฎในฐานข้อมูลอนุมาน

## 10.1 การจัดการกับกฎในฐานข้อมูลอนุมาน

ในบทก่อน ๆ ที่ผ่านมา เราได้กล่าวถึงการใช้แบบจำลองข้อมูลไนแอมเป็นแบบแสดงความรู้ และวิธีที่จะแทนแบบจำลองข้อมูลไนแอมลงในฐานข้อมูลแบบรีเลชันแนล ซึ่งประกอบด้วย การนิยามข้อมูล การจัดการข้อมูล และการตรวจสอบความถูกต้อง แต่องค์ประกอบที่สำคัญอีกประการหนึ่งในระบบฐานข้อมูลอนุมาน คือ การจัดการกับกฎซึ่งถือว่าเป็นหัวใจของส่วนประมวลผลคำถามนั้น เรายังไม่ได้กล่าวถึงเลย ดังนั้นในบทนี้เราก็จะกล่าวถึงกลไกที่จะใช้กับกฎเพื่อใช้กับฐานข้อมูลอนุมานที่ใช้แบบจำลองข้อมูลไนแอมเป็นฐานข้อมูลอนุมาน

## 10.2 การจัดการกับกฎชนิดไม่รีเคอร์ซีฟ

สมมติว่าเรามีฐานข้อมูลที่เก็บความสัมพันธ์ของวิชาบังคับ เช่น Imm_Prereq ('Data Base', 'Data Structure') หมายถึงความสัมพันธ์ที่ว่าการลงทะเบียนเรียนวิชา 'Data Base' จะต้องผ่านวิชา 'Data Structure' มาก่อน แต่สมมติว่าถ้าเราต้องการเก็บความสัมพันธ์ว่าผู้ที่เรียนวิชา 'Data Base' จะต้องผ่านวิชาอะไรมาบ้าง (ทั้งนี้เพราะว่าวิชา 'Data Structure' ก็อาจจะมีวิชาบังคับที่จะต้องผ่านเสียก่อนได้เหมือนกัน) ซึ่งจะเห็นว่ากรณีเช่นนี้เราไม่จำเป็นต้องเก็บความสัมพันธ์นี้ลงในฐานข้อมูลจริงๆ แต่เราจะเก็บเป็นกฎของความสัมพันธ์นั้นแทนดังนี้ (เขียนในรูปแบบของภาษาโปรลอก)

```
Prereq(X,Y) :- Prereq(X,Z),Imm_Prereq(Z,Y)
```

ความสัมพันธ์ Imm_Prereq จะอยู่ในส่วนของ Extensional Database (EDB) ซึ่งหมายถึงความสัมพันธ์ที่เก็บลงในฐานข้อมูลจริง ส่วน Prereq จะอยู่ในส่วนของ Intensional Database (IDB) ซึ่งหมายถึงความสัมพันธ์ที่ไม่ต้องเก็บจริงในฐานข้อมูล แต่จะอนุมานได้จากกฎแห่งความสัมพันธ์นั้นแทน ภาษาที่ใช้ในระบบฐานข้อมูลในปัจจุบันนั้น ยังไม่มีกลไกในการจัดการกับกฎได้โดยตรง ในขณะที่ภาษาทางด้าน Logic เช่น PROLOG มีกลไกที่เรียกว่า resolution ที่จะใช้ในการจัดการกับกฎต่างๆ ได้ แม้กระนั้นเราก็ยังสามารถที่จะประยุกต์ใช้ความสามารถบางอย่าง ของระบบฐานข้อมูลเพื่อใช้ในการจัดการกับกฎดังกล่าวนี้ได้ ความสามารถดังกล่าวของฐานข้อมูลก็คือความสามารถในการสร้างวิว (View) ซึ่งฐานข้อมูลโดยทั่วไปความสามารถอันนี้อยู่แล้ว

เช่นสมมติว่ามีกฎ

```
P1(X,Y),P2(Y,Z) ---> P3(X,Z)
```

เราสามารถที่จะใช้คำสั่งภาษา SQL สร้างวิวที่มีความเท่าเทียมกับกฎข้างต้นได้ดังนี้

```
CREATE VIEW P3(X,Z) AS
```

```
SELECT X,Z
FROM P1,P2
WHERE P1.Y = P2.Y ;
```

แต่ในกรณีที่คำตอบที่เราต้องการประกอบด้วยกฎหลาย ๆ กฎ ดังนี้

```
Major_In (X,'AI') <-- Take_Subj (X,'CS345').
Major_In (X,'Database') <-- Take_Subj (X,'CS314').
Major_In (X,'Statistics') <-- Take_Subj (X,'MS312'),Take_Subj (X,'MS390')
```

ถ้าเราใช้พรีดิเขตแทนชนิดความจริงของแบบจำลองข้อมูล ชนิดความจริง 'Major_In หมายถึงนักศึกษา X จะเรียนในสาขาหลักสาขาใดสาขาหนึ่งได้เมื่อนักศึกษา X ได้ลงเรียนในวิชาที่กำหนด เช่น ถ้า X จะเป็นนักศึกษาในสาขาหลัก 'AI' เมื่อ X ลงเรียนในวิชา 'CS345' ซึ่งกฎทั้งสามนี้ก็จะมีความหมายว่าให้หานักศึกษาที่อยู่ในสาขาหลัก 'AI','Database' และ 'Statistics'

โดยทั่วไปคำตอบที่เราต้องการจะหาได้จากเซ็ทของกฎทั้งสาม ก็คือผลลัพธ์ของแต่ละกฎที่จะถูกนำมายูเนียน (Union) กัน ซึ่งในกรณีเช่นนี้เราไม่สามารถใช้ View เข้าช่วยได้ ทั้งนี้เพราะในระบบฐานข้อมูลส่วนมากไม่มีความสามารถในการทำยูเนียนใน View แต่ในปัจจุบันระบบฐานข้อมูลบางตัว เช่น Oracle ได้สร้างความสามารถนี้ไว้แล้ว อย่างไรก็ตามในกรณีเช่นนี้เราจะใช้วิธีการอีกแบบหนึ่งโดยการสร้างตารางที่ชื่อ Major_In ขึ้นมาใหม่ ซึ่งตารางที่สร้างขึ้นนี้จะถูกใช้งานเพียงชั่วคราวเพื่อช่วยในการรวมคำตอบที่ได้จากกฎทั้งสาม เมื่อเราสร้างตารางดังกล่าวเรียบร้อยแล้ว เราก็จะทำการ Insert ข้อมูลที่ได้จากกฎทั้งสามซึ่งสามารถเขียนเป็นคำสั่ง SQL ได้ดังนี้

```
INSERT INTO Major_In
        SELECT STUDENT,'AI'
        FROM Take_Subj
        WHERE SUBJECT = 'CS345';
INSERT INTO Major_In
        SELECT STUDENT,'Database'
        FROM Take_Subj
        WHERE SUBJECT = 'CS314' ;
INSERT INTO Major_In
        SELECT STUDENT,'Statistics'
        FROM Take_Subj FT1,Take_Subj FT2
        WHERE FT1.STUDENT = FT2.STUDENT AND
                FT1.SUBJECT = 'MS312' AND
                FT2.SUBJECT = 'MS390' ;
```

เมื่อมาถึงจุดนี้เราก็ได้ทราบแนวทางที่จัดการกับกฎ โดยที่แต่ละพรีดิเขตแทนชนิดความจริงที่กำหนดไว้ในแบบจำลองข้อมูลไนแอม อย่างไรก็ตามกฎที่สามารถจัดการโดยใช้เทคนิคข้างต้นนั้นได้ ก็จะต้องมีข้อแม้อยู่ว่าจะต้องเป็นกฎที่ไม่เป็นรีเคอร์ซีฟ ซึ่งในขั้นตอนต่อไปเราก็จะแสดงให้เห็นเพิ่มเติมว่าสำหรับกฎที่อยู่ในรูปแบบที่เป็นรีเคอร์ซีฟนั้น เรามีวิธีจัดการอย่างไร

## 10.3 กฎชนิดรีเคอร์ซีฟ

นักวิทยาศาสตร์ทางด้านฐานข้อมูล ต่างทราบกันดีถึงการที่จะประมวลผลคำถามทางลอจิก (Logic Query) โดยผ่านทางวิว (View) ซึ่งเป็นกลไกหนึ่งที่มีอยู่ในฐานข้อมูลแบบรีเลชันแนลทั่วไป แต่คำถามเหล่านั้นมีขอบเขตอยู่ว่าจะต้องไม่เป็นคำถามแบบรีเคอร์ซีฟ (Non-Recursive Query) ทั้งนี้เพราะถ้าเราทำการประมวลผลคำถามแบบรีเคอร์ซีพในลักษณะเช่นเดียวกับคำถามที่ไม่เป็นรีเคอร์ซีพ จะทำให้การประมวลผลเกิดวงรอบการทำงานที่เป็นอนันต์ ซึ่งแม้กระทั่งทุกวันนี้ ปัญหาทางของการแก้ไขคำถามแบบรีเคอร์ซีฟก็ยังมีผู้วิจัยอยู่

เมื่อหลายปีก่อน ได้มีนักวิทยาศาสตร์กลุ่มหนึ่งซึ่งประกอบด้วย Chang, Shapiro และ McKay, Henschen และ Naqvi ได้บุกเบิกงานทางด้านการจัดการกับกฎแบบรีเคอร์ซีฟ ทำให้งานทางด้านนี้ได้พัฒนาไปมาก แต่งานที่ได้พัฒนาไปนั้นส่วนใหญ่แล้วจะอยู่ในรูปของพรีดิเขตลอจิก อย่างไรก็ตามอย่างที่ได้กล่าวไปแล้วว่า พื้นฐานของทฤษฎีทางรีเลชันแนลและพรีดิเขตลอจิก มีความสัมพันธ์กันอย่างใกล้ชิดภายใต้เงื่อนไขบางประการ ดังนั้นเราจึงสามารถนำงานดังกล่าวมาประยุกต์ใช้ได้

ก่อนที่เราจะกล่าวถึงการแก้ปัญหาของคำถามแบบรีเคอร์ซีพ เราจะมาดูรูปแบบของคำถามประเภทนี้ว่ามีกี่แบบ และในแต่ละแบบมีลักษณะอย่างไรบ้าง

### 10.3.1 กฎชนิดลิเนียร์รีเคอร์ซีฟ (Linear Recursive Rule)

สมมติว่าเรามีพรีดิเขตต่อไปนี้

(1) Imm_Prereq ('Data Base','Data Structure').

(2) Imm_Prereq ('Data Structure','Programming Language').

(3) Prereq (X,Y) :- Prereq(X,Z),Imm_Prereq(Z,Y).

จะสังเกตเห็นได้ว่าในข้อที่ (1)-(2) จะเป็น EDB (Extensional Database) ซึ่งเป็นชนิดความจริงที่ถูกเก็บในฐานข้อมูล และข้อที่ (3) เป็นการกำหนด IDB (Intensional Database) ซึ่งเป็นข้อมูลที่ไม่ต้องเก็บลงในฐานข้อมูลจริง และสามารถหามาได้ในภายหลัง กฎแบบรีเคอร์ซีฟนั้นสามารถเขียนขึ้นในรูปอย่างง่ายให้อยู่ในรูปทั่วไปได้ดังนี้

p(t) :- ...,p(t'),...

เช่น Prereq (X,Y) :- Prereq(X,Z),Imm_Prereq(Z,Y).

ซึ่งกฎที่อยู่ในรูปแบบดังกล่าวเราจะเรียกได้อีกชื่อว่า "Linear Recursive" ซึ่งจะมี ลักษณะที่ว่าชนิดความจริงที่เป็น Head ของกฎนั้นไปปรากฏที่ Body ของกฎนั้นด้วย

**10.3.2 กฎชนิดมิวช่วลรีเคอร์ซีฟ**

ต่อไปเราลองพิจารณากฎต่อไปนี้

p (X,Y) :- b1 (X,Z),q (Z,Y).

q (X,Y) :- p (X,Z),p2 (Z,Y).

จะเห็นได้ว่าไม่มีกฎใดเลยที่เป็นรีเคอร์ซีฟตามข้อกำหนดข้างต้น แต่เมื่อพิจารณาให้ดีแล้วจะพบว่าทั้งชนิดความจริง p และ q ต่างก็เป็นรีเคอร์ซีฟของซึ่งกันและกัน ถึงตรงนี้เราก็จะหาข้อกำหนดสำหรับรีเคอร์ซีฟแบบที่เกี่ยวข้องกับกฎหลายๆ กฎ เรากล่าวว่า p derive q (p --> q) ถ้ามีชนิดความจริง p ปรากฏอยู่ใน body ของกฎที่มี q เป็น Head และทั้ง p และ q จะถูกเรียกว่าเป็น "Mutually Recursive" ของซึ่งกันและกันเมื่อ p --> q และ q --> p

**10.3.3 การจัดการกับกฎแบบรีเคอร์ซีฟ**

จากที่ได้กล่าวมาข้างต้นว่าคำถามแบบรีเคอร์ซีพไม่สามารถแก้ได้โดยวิธีทั่วไปได้ ดังนั้นก็น่าจะมีวิธีที่จะสามารถจัดการกับมันได้โดยเฉพาะ ซึ่งวิธีในการจัดการกฎแบบรีเคอร์ซีฟนี้ จากการศึกษาจากบทความเปรียบเทียบวิธีการจัดการกับกฎแบบรีเคอร์ซีฟ An Amateur's Introduction to Recursive Query Processing Strategies ดังต่อไปนี้

10.3.3.1 การเปรียบเทียบวิธีการจัดการกับกฎแบบรีเคอร์ซีฟ

จากการศึกษาบทความเปรียบเทียบเรื่อง An Amature's Introduction to Recursive Query Processing Strategies ซึ่งวิจัยโดย Francois Bancihon และ Raghu Ramakrishnan [1986] โดยบทความนี้จะเปรียบเทียบวิธีการต่างๆ ที่ใช้ในการประมวลผลคำถามที่มีลักษณะเป็นกฎชนิดรีเคอร์ซีฟ (Recursive Query) ในระบบฐานข้อมูลแบบรีเลชันแนล (Relational Database Systems) โดยที่คำถามนั้นจะต้องมีลักษณะเป็น Horn Clause และไม่มี Function Symbol

โดยที่งานทางด้านนี้ได้เพิ่งถูกบุกเบิก เมื่อประมาณ 10 ปีที่ผ่านมานี่เองโดยนักวิจัย เช่น Chang, Shapiro และ Mckey, Henschen และ Naqvi จนได้ผลลัพธ์เป็นอัลกอริทึมออกมามากมาย แต่ปัญหาก็คือไม่ทราบว่าจะเลือกใช้อัลกอริทึมใดที่เหมาะสมกับงานที่ต้องการ ในบทความการเปรียบเทียบนี้จะใช้หลักการเปรียบเทียบในหลาย ๆ แง่มุม แต่หลักการที่เป็นหลักใหญ่ ๆ และเป็นหลักการที่เราสนใจได้แก่

(1) ใช้วิธีการ Interpretation หรือ Compilation วิธีการใดจะเรียกว่าเป็นคอมไพล์ก็ต่อเมื่อวิธีการนั้นมีการแบ่งขั้นตอนการทำงานออกเป็น 2 ขั้นตอนคือขั้นตอนสร้างคำถาม (Compile Query) และขั้นตอนนำคำถามมาทำงาน (Execute Query) ซึ่งในขบวนการสร้างคำถามนั้น จะต้องอ้างถึงแต่เฉพาะข้อมูลในส่วนที่เป็น IDB เท่านั้น และจะอ้างถึงข้อมูลในส่วนที่เป็น EDB เมื่อมีการนำคำถามมาทำงาน. วิธีการใดที่ไม่เป็นไปตามนี้ ถือได้ว่าเป็นการ Interpretation

(2) มีการทำงานในแบบ Recursion หรือ Iteration การทำงานในแบบรีเคอร์ชันนั้นเป็นการทำงานที่มีประสิทธิภาพกว่า เพราะการที่เป็นรีเคอร์ชันทำให้ไม่มีขอบเขตของการทำซ้ำ ขณะที่ Iteration จะต้องมีขอบเขต

(3) มีการทำงานแบบ Top-Down หรือ Bottom-Up การทำงานทั้งสองแบบนี้เป็นการมองว่าการหาคำตอบนี้เหมือนกับการตรวจสอบไวยากรณ์ (Parsing) ในตัวแปลภาษา โดยที่จะมองแต่ละพรีดิเขตใน EDB เป็นสัญญลักษณ์สิ้นสุด (Terminal Symbol) และพรีดิเขตใน IDB เป็นสัญญลักษณ์ไม่สิ้นสุด (Non-Terminal Symbol) และการค้นหาคำตอบก็คือการพยายามทำให้คำถามซึ่งเป็น Non-Terminal Symbol กลายเป็น Terminal Symbol ทั้งหมด อย่างไรก็ตามในกรณีของคำถามที่เป็นรีเคอร์ซีฟนี้จะทำให้เกิดวงรอบที่ไม่สิ้นสุด ดังนั้นก็จะต้องมีเงื่อนไขสิ้นสุด (Termination Condition) คือเมื่อ Terminal Symbol ที่สร้างขึ้น Evaluate ได้เซ็ทว่างก็ถือเป็นการสิ้นสุด

และความแตกต่างระหว่าง Top-Down และ Bottom-Up ก็คือ Top_down จะเริ่มจากคำถาม (Non-Termial Symbol) และ Bottom-Up จะเริ่มจากคำตอบ (Terminal Symbol) ซึ่งวิธี Top-Down จะมีประสิทธิภาพมากกว่า เพราะว่าเรารู้ว่าคำถามอะไรที่เราต้องการจะหาคำตอบ แต่วิธีการจะซับซ้อนกว่าขณะที่ Bottom Up จะง่ายกว่าแต่ไม่มีประสิทธิภาพเพราะจะมีผลลัพธ์จำนวนมากที่ไม่ได้ใช้งาน

ซึ่งจากบทความดังกล่าวได้ทำการเปรียบเทียบวิธีการต่างๆ จำนวน 10 วิธีดังตารางที่ 10-1 และสำหรับวิธีการที่เราเลือกใช้นั้น ที่แน่นอนคือต้องเป็นแบบ Iterative เพราะภาษา SQL ที่เราจะทำการเชื่อมต่อด้วยนั้น (Interface) ไม่มีความสามารถในการ Recursive และควรจะเป็น Top-Down ซึ่งเป็นวิธีการที่มีประสิทธิภาพในการหาคำถามมากกว่า ซึ่งวิธีการที่มีคุณสมบัติ 2 ประการก็มีอยู่ 2 วิธี แต่ที่เราเลือกใช้วิธีการของ Henschen-Naqvi นั้นก็เพราะใช้วิธีการคอมไพล์ ซึ่งมีการทำงานแยกกันระหว่างการสร้างคำถามและการหาคำตอบซึ่งจะทำให้การพัฒนาเป็นไปได้ง่ายกว่า

| Method | Top-Down Bottom-Up | Compiled Interpreted | Iterative Recursive |
|---|---|---|---|
| Naive Evaluation | Bottom-Up | Compiled | Iterative |
| Semi-Naive Evaluation | Bottom-Up | Compiled | Iterative |
| Query/Subquery | Top-Down | Interpreted | Iterative |
| Query/Subquery | Top-Down | Interpreted | Recursive |
| APEX | Mixed | Compiled | Recursive |
| Prolog | Top-Down | Interpreted | Recursive |
| Henschen-Naqvi | Top-Down | Compiled | Iterative |
| Aho-Ullman | Bottom-Up | Compiled | Iterative |
| Kifer-Lozinskii | Bottom-Up | Compiled | Iterative |
| Counting | Bottom-Up | Compiled | Iterative |
| Magic Sets | Bottom-Up | Compiled | Iterative |

*ตารางที่ 10-1 ผลลัพธ์ของการเปรียบเทียบวิธีการแก้ปัญหากฎแบบรีเคอร์ซีฟ*

### 10.4 วิธีการของ Henschen และ Naqvi

เมื่อเราเลือกใช้วิธีการของ Henchen และ Naqvi จากนั้นเราจะนำวิธีการของนักวิจัยทั้งสองท่านมาประยุกต์ใช้กับปัญหาการแทนชนิดความจริงด้วยพรีดิเขตของเรา ซึ่งวิธีการดังกล่าวนั้นจะทำการแปลงกฎชนิดรีเคอร์ซีฟให้เป็นลำดับของกฎชนิดไม่รีเคอร์ซีฟ และจากนั้นเราก็จะแทนลำดับของกฎเหล่านี้ด้วยภาษาฐานข้อมูล SQL เพื่อให้ชัดเจนเรามาลองพิจารณากฎ R1 และ R2 ต่อไปนี้

R1 : Prereq(X,Y) <-- Imm_Prereg(X,Y).

R2 : Prereg(X,Y) <-- Prereg(X,Z),Imm_Prereg(Z,Y).

Query : Prereq(X,Y) ?

ซึ่งความสัมพันธ์ Prereq(X,Y) หมายถึงนักศึกษาผู้ที่จะลงในวิชา X จะต้องลงในรายวิชา Y อะไรมาบ้างและความสัมพันธ์ Imm_Prereq(X,Y) หมายถึงนักศึกษาที่จะลงวิชา X จะต้องลงในรายวิชา Y ในเทอมที่ผ่านมา ในการหาคำตอบของคำถามและกฎข้างต้นก็คือเซ็ทของกฎที่หามาได้จากกฎ R1 และ R2 ซึ่งกฎใหม่ที่ได้จากกฎทั้งสองนั้นจะไม่เป็นแบบรีเคอร์ซีฟ ซึ่งกฎใหม่ดังกล่าวนี้ได้แสดงไว้ข้างล่างนี้แล้ว

Imm_Prereq(X,Y);

Imm_Prereq(X,Y1),Imm_Prereq(Y1,Y);

Imm_Prereq(X,Y1),Imm_Prereq(Y1,Y2),Imm_Prereq(Y2,Y);

...

กฎที่ได้แสดงไว้ข้างต้นนี้ กฎแรกได้มาจาก R1 และกฎต่อมาหามาได้จากการแทนกฎ R1 ลงในกฎ R2 และต่อจากนั้นก็จะเป็นการแทนกฎ R1 ลงในกฎที่ได้จากกฎที่ 2 และแทนลงเช่นนี้ จะเห็นได้ว่ากฎที่สร้างขึ้นใหม่นี้มีจำนวนที่ไม่จำกัด ดังนั้นเราจะต้องหาเงื่อนไขที่จะทำให้การสร้างกฎนี้มีการสิ้นสุด เรารู้ว่ากฎแต่ละกฎที่สร้างขึ้นจะได้จากจำนวนครั้งที่เราทำการแทนกฎ R1 และ R2 ซึ่งจะเหมือนกับการวงรอบทำรีเคอร์ซีฟแต่ละรอบนั่นเอง

และเงื่อนไขที่จะทำให้การสร้างกฎนี้สิ้นสุดก็คือ เมื่อไรก็ตามที่เราหาคำตอบที่ได้จากกฎที่เราสร้างขึ้นใหม่แล้วได้ผลลัพธ์เป็นเซ็ทว่าง (Empty Set) ก็จะถือว่าเป็นเงื่อนไขที่สิ้นสุดการสร้างกฎใหม่ และคำตอบของกฎที่เป็นรีเคอร์ซีฟ ก็คือยูเนียนของผลลัพธ์ที่ได้จากกฎที่สร้างขึ้นก่อนที่จะถึงเงื่อนไขที่สิ้นสุดนั่นเอง

ต่อไปเราก็จะกล่าวถึงวิธีของ Henschen-Naqvi โดยละเอียด จากที่ได้กล่าวไปแล้วข้างต้น

ข้อกำหนด ฐานข้อมูลอนุมานประกอบด้วยสองส่วนโดยส่วนแรกคือ EDB (Extensional database) ซึ่งก็คือข้อมูลที่เป็น Ground Clause ซึ่งก็จะสอดคล้องกับข้อมูลในแต่ละ tuple ที่ถูกเก็บอยู่ในรูปของความสัมพันธ์ในฐานข้อมูลและส่วนที่สองคือ IDB (Intensional Database) ซึ่งก็คือเซ็ตของ Horn Clause นั่นเอง

ข้อกำหนด พรีดิเขตใดๆ ที่เกิดขึ้นใน EDB จะเรียกว่าพรีดิเขต EDB และพรีดิเขตใดๆ ที่เกิดขึ้นใน IDB จะเรียกว่าพรีดิเขต IDB ซึ่งวิธีการของ Henschen-Naqvi จะแทนแต่ละรีเลชันด้วยพรีดิเขต เช่นถ้าเรามีความสัมพันธ์ P ซึ่งมี Tuple เป็น <a1,...,an) เราสามารถแทนด้วยพรีดิเขตได้เป็น P(a1,...,an) และแทน clause ของ IDB นี้ในรูปของ Connected Graphs (CG)

ข้อกำหนด CG คือความสัมพันธ์ (N,E,S,P) โดยที่

(1) N เป็นเซ็ตของ Node โดยที่แต่ละ Node จะแทนแต่ละ Literal ที่เกิดขึ้นใน IDB

(2) E คือเซ็ตของ Edge โดยสมมติว่า e เป็น Edge ที่เชื่อมระหว่าง L1 และ ~L2 และ e จะอยู่ใน เซ็ต E ก็ต่อเมื่อ L1 และ L2 สามารถรวมกันได้ (Unificable)

(3) S เป็นเซ็ตของการแทน (Substitution) โดยสมมติว่า s เป็นการแทน และ s จะอยู่ใน S ก็ต่อ เมื่อ s เป็น m.g.u. (most general unifier) ของ Literal L1 และ L2 บน Edge E

(4) P เป็นเซ็ตของส่วนของ Clause โดยสมมติให้ p เป็นส่วนของ Clause และ p จะอยู่ในเซ็ต P ก็ต่อเมื่อมันมี Literal บางตัวที่อยู่ใน Clause ของ IDB

วิธีการของ Henschen และ Naqvi จะแทนกฎทั้งหมดด้วย CG และโดยอาศัยวิธีการของ Sickel (1976) เราจะสามารถตรวจพบวงรอบขึ้นใน CG นั้นในกรณีที่มี Clause ที่เป็นรีเคอร์ซีฟเกิดขึ้นและเราจะเรียกวงรอบนี้ว่า Potential Recursive Loop (PRL) ซึ่งรูปแบบโดยทั่วไปของ PRL แสดงในรูปที่ 10-1

ต่อไปเราก็จะกล่าวถึงข้อกำหนดต่างๆ ใน PRL ดังต่อไปนี้

ข้อกำหนด clause ของ PRL ที่เชื่อมกับคำถามจะเรียกว่า Start Clause ซึ่งในรูปที่ 10-1 ก็คือ clause D0' ~D1 ~E1

ข้อกำหนด Literal ใน Start Clause ที่เชื่อมกับคำถามจะเรียกว่า Start Literal

ข้อกำหนด LIteral ภายใน PRL ที่เชื่อมกับ Start Literal เรียกว่า End Literal

ข้อกำหนด Clause ที่มี End Literal อยู่ภายในเรียกว่า End Clause ซึ่งจากตัวอย่างในรูปที่ 10-1 ก็คือ clause Dn' ~Dn+1 ~En+1

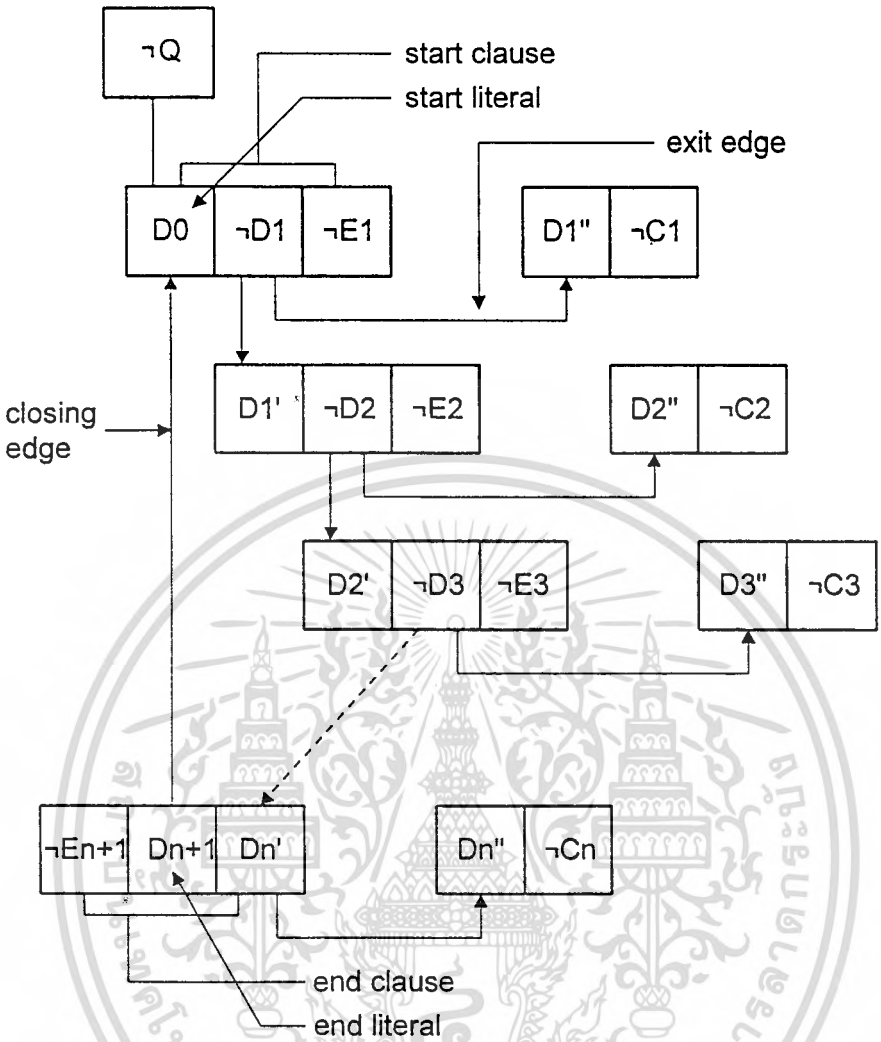
ข้อกำหนด Literal ~Di และ Di' ที่แสดงไว้ในรูปที่ 10-1 เรียกว่า Cycle Literals

ข้อกำหนด Closing Edge ของ PRL คือ Edge ที่เชื่อมต่อกับ End Literal และ Start Literal

ข้อกำหนด Edge ที่ต่อกับ Cycle Literal และทำให้ออกจาก PRL ได้เรียกว่า Exit Edge

ข้อกำหนด Augmented Loop Residue หาได้จากการหาผลลัพธ์ของ Start Clause ในภาพด้วย การแทน D1' ~D2 ~E2 ลงใน Start Clause และหาผลลัพธ์ของ Clause ที่ได้ใหม่นี้โดยการแทน clause D2' ~D3 ~E3 และทำเช่นนี้จนถึง End Clause

ข้อกำหนด Augmented Exit Expression คือการหาผลลัพธ์ของ Start Clause โดยการแทน D1" ~C1 หรือโดยการแทน D1' ~D2 ~E2 ลงใน Start Clause แล้วจึงแทน clause D2" ~C2 ลงใน ผลลัพธ์ที่ได้นี้




*รูปที่ 10-1 รูปแบบทั่วไปของ Potential Recursive Loop (PRL)*

เพื่อให้เข้าใจยิ่งขึ้น จะขอยกตัวอย่างประกอบดังนี้ สมมติว่า M,F,P เป็น EDB และ S,T เป็น IDB และเป็น Mutually Recursive ซึ่งกันและกัน พรีดิเขตเหล่านี้จะแทนความหมายของ Mother, Father และ Parent และ S กับ T เป็นแต่ละชนิดของ Ancestor

R1 : S (X1,Z1) <-- M (X1,Y1),T(Y1,Z1).

R2 : T (Y1,Z1) <-- S (Y1,W1),P(W1,Z1).

R3 : T (Y1,Z1) <-- F(Y1,Z1).

ซึ่งกฎดังกล่าวสามารถเขียนให้อยู่ในรูปของ Clausal Form ได้ดังนี้

C1 : ~M(X1,Y1), ~T(Y1,Z1),S(X1,Z1).

C2 : ~S(Y1,W1), ~P(W1,Z1),T(Y1,Z1).

C3 : ~F(Y1,Z1),T(Y1,Z1).

Query : S(X,'a').

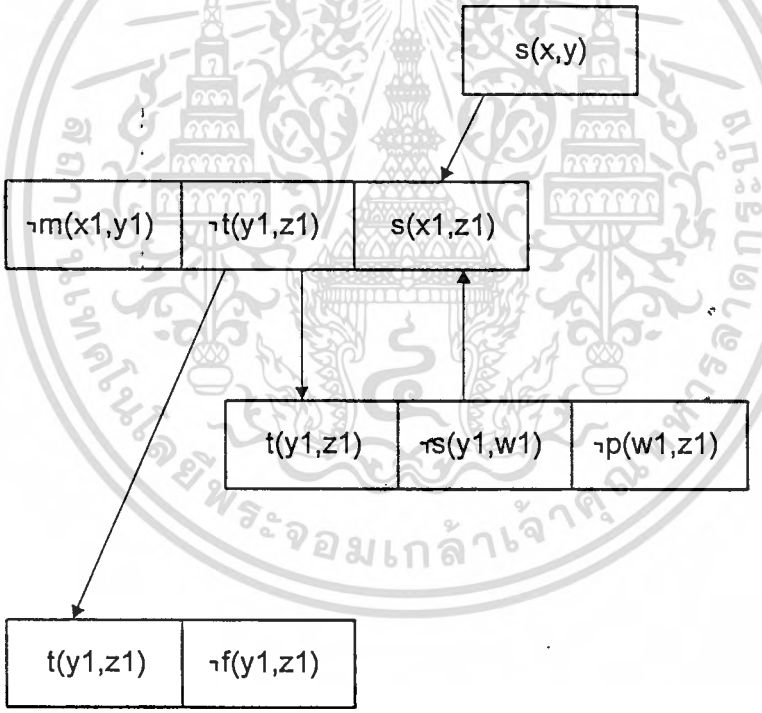
จากกฎข้างต้นนั้นเราสามารถนำมาเขียนเป็น PRL ได้ดังรูปที่ 10-2 และจากข้อกำหนด Start Clause คือ C1 และ Start Literal คือ S(X1,Z1) และ Cycle Literal ~T(Y1,Z1) ของ Clause C1 และ T (Y1,Z1) ของ Clause C2 สามารถรวมกันได้ (Unification) และเชื่อมกันโดย Edge e2 และเนื่องจาก S (X1,Z1) เป็น Start Literal ~S(Y1,W1) เชื่อมกับ End Literal ดังนั้น C2 ก็จะเป็น End Clause โดยมี Edge e3 ออกจาก Cycle Literal ~T(Y1,Z1) ไปยัง Exit Edge

และจากที่กล่าวมาเราจะได้ Augmented Loop Residue ของ PRL ข้างต้นคือ

S(X1,Z1), ~M(X1,Y1), ~S(Y1,W1), ~P(W1,Z1).

และจะได้ Augmented Exit Expression ดังนี้

S(X1,Z1), ~M(X1,Y1), ~F(Y1,Z1).




*รูปที่ 10-2 ตัวอย่างของ Potantial Recursive Loop*

เมื่อถึงจุดนี้จาก PRL เราก็จะได้ Augmented Loop Residue และ Augmented Exit Expression และลำดับของกฎที่ไม่เป็นรีเคอร์ซีฟที่จะนำมาใช้หาคำตอบของกฎที่เป็นรีเคอร์ซีฟ ซึ่งจะสร้างได้จากการแทน Augmented Exit Expression ลงใน Augmented Loop Residue จากนั้นก็ทำการ Evaluate กฎใหม่ที่เราสร้างขึ้นและถ้าผลลัพธ์จากการ Evaluate กฎใหม่ที่ได้นี้ยังไม่ให้ผลลัพธ์เป็นเซ็ทว่าง เราก็จะนำ Augmented Exit Expression แทนลงในกฎใหม่นี้อีกและทำเช่นนี้จนกระทั่งได้ผลลัพธ์ของการ Evaluate เป็นเซ็ทว่างและคำตอบของกฎที่เป็นรีเคอร์ซีฟดังกล่าวก็คือยูเนียนของกฎที่เราสร้างขึ้นนั่นเอง

แต่เนื่องจากภาษา SQL ไม่มีความสามารถในการยูเนียนดังที่ได้กล่าวมาก่อนหน้านี้ ดังนั้นเราก็จะใช้วิธีการเช่นเดียวกับกรณีที่มีกฎหลาย ๆ กฎ กล่าวคือจะใช้วิธีการสร้างตารางขึ้นมาเพื่อเก็บผลลัพธ์จากทั้งหมดที่ได้กล่าวมาแล้วสามารถนำมาเขียนเป็นอัลกอริธึมได้ดังนี้

**Algorithm**

[1] For each recursively defined deducible fact type, construct a PRL. The predicate representing the fact type is the query which indicate the starting clause.

[2] Resolving the PRL from the starting clause by using the Henschen-Naqvi method to obtain the augmented loop residue and augmented exit expression.

[3] Derive a corresponding relation for each non-recursive rule which defined the deducible fact type by using the mapping from Horn-clause rules to derive relations.

[4] Derive a corresponding relation for the augmented exit expression by using the mapping from Horn-clause rules to derive relations.

[5] Set MAIN := the augmented loop residue.

Set SUB := the augmented exit expression.

[6] Repeat until TERMINATE:

[6.1] Set RULE := MAIN.

[6.2] Replace the predicate representing the deducible fact type in condition part of RULE by the condition of SUB.

[6.3] Derive a corresponding relation for RULE by using the mapping from Horn-clause rules to derived relations.

If the derived relation is an empty set,

then set TERMINATE := True;

Else set SUB := RULE; TERMINATE := False.

[7] The corresponding relation of the deducible fact type is union of all the derived relations ontained from [2] to [6].

จากตัวอย่างที่ผ่านมา M,F และ T เป็น EDB และ S,T เป็น IDB นั้น
จะได้ Augmented Loop Residue คือ

S(X1,Z1), ~M(X1,Y1), ~S(Y1,W1), ~P(W1,Z1).

และจะได้ Augmented Exit Expression ดังนี้

S(X1,Z1), ~M(X1,Y1), ~F(Y1,Z1).

ซึ่ง Clause ทั้งสองนี้สามารถเขียนในรูปของ Logic ได้ดังนี้

S(X1,Z1) <- M(X1,Y1),S(Y1,W1),P(W1,Z1)

S(X1,Z1) <- M(X1,Y1),F(Y1,Z1)

และจาก Clause ทั้งสองนั้น จะสามารถสร้างกฎขึ้นมาใหม่ตามวิธีการของ Henschen และ Naqvi ได้ดังนี้

S(X1,Z1) <- M(X1,Y1),F(Y1,Z1)

S(X1,Z1) <- M(X1,Y1),M(Y1,D1),F(Y1,Z1),P(W1,Z1)

S(X1,Z1)<-M(X1,Y1),M(Y1,D1),M(D1,D2),F(D2,D3),P(D3,W1),P(W1,Z1)

...

และจากที่ได้กล่าวมาแล้วว่าระบบฐานข้อมูลโดยทั่วๆ ไปไม่สนับสนุนความสามารถของการสร้าง Union บนวิว และจากความจริงที่ว่าเราไม่อาจที่จะทราบได้ถึงจำนวนกฎที่จะถูกสร้างขึ้น ดังนั้นเราจะใช้วิธีการสร้างความสัมพันธ์ขึ้นมาเป็นตารางถาวร และจากนั้นในแต่ละครั้งของการสร้างกฎ เราก็จะแปลงกฎที่สร้างขึ้นนี้ไปเป็นภาษา SQL เพื่อ INSERT ผลของแต่ละกฎลงในตาราง ซึ่งจากกฎข้างต้นก็จะนำมาสร้าง SQL ได้ดังต่อไปนี้

```
INSERT TO S
SELECT M.NAME1,F.NAME2
FROM M,F
WHERE M.NAME2 = F.NAME1

INSERT TO S
SELECT FT1.NAME1,P.NAME2
FROM M FT1,M FT2,F,P
WHERE FT1.NAME2 = FT2.NAME1 AND
        FT2.NAME2 = F.NAME1 AND
        F.NAME2 = P.NAME1

INSERT INTO S
SELECT FT1.NAME1,FT5.NAME2
FROM M FT1,M FT2,M FT3,F,P FT4, P FT5
WHERE FT1.NAME2 = FT2.NAME1 AND
        FT2.NAME2 = FT3.NAME1 AND
        FT3.NAME2 = F.NAME1 AND
        F.NAME2 = FT4.NAME1 AND
        FT4.NAME2 = FT5.NAME1
```

และในกรณีที่ระบบจัดการฐานข้อมูลมีความสามารถในการทำ Union บนวิวแล้ว การสร้างตารางขึ้นมารองรับนี้ก็ไม่จำเป็น ซึ่งในกรณีเช่นนี้เราจะเก็บกฎที่สร้างขึ้นนี้อยู่ในรูปของข้อกำหนดของวิวแทน จากที่ผ่านมาเราได้แสดงการจัดการกับกฎแบบรีเคอร์ซีพซึ่งเป็นแบบ Mutually Recursive ไปแล้ว สำหรับขั้นตอนต่อไปเราจะแสดงการจัดการกับกฎแบบ Linear Recursive โดยใช้ตัวอย่างเรื่องของวิชาบังคับก่อนการลงวิชาต่างๆ ที่ได้กล่าวมาแล้วครั้งหนึ่ง ซึ่งมีดังต่อไปนี้

R1 : Prereq(X,Y) <-- Imm_Prereg(X,Y).

R2 : Prereg(X,Y) <-- Prereg(X,Z),Imm_Prereg(Z,Y).

Query : Prereq('Database',Y) ?

และสมมติว่าข้อมูลมีดังต่อไปนี้

Imm_Prereq('Database','Data Structure')

Imm_Prereq('Data Structure','Programming Language')

โดยที่ Prereq(X,Y) เป็นกฎชนิดรีเคอร์ซีพที่หมายถึงว่า ถ้าจะลงเรียนในวิชา X จะต้องลงทะเบียนเรียนในรายวิชา Y อะไรมาบ้างและ Imm_Prereq(X,Y) เป็นชนิดความจริงที่มีความหมายว่าการลงทะเบียนเรียนในวิชา X จะต้องผ่านในวิชา Y มาก่อน ในกรณีเช่นนี้ทั้ง Start Clause และ End Clause จะเป็น Clause เดียวกันคือ R2 และจากการสร้าง PRL เราจะสามารถหา Augmented Loop Residue ได้เป็น

Prereg(X,Y), ~Imm_Prereg(X,Z), ~Prereq(Z,Y)

และจะได้ Augmented Exit Expression เป็น

Prereq(X,Y), ~Imm_Prereq(X,Z), ~Imm_Prereq(Z,Y)

คำตอบแรกจะได้จากการแทน Augmented Exit Expression ลงใน Augmented Loop Residue และคำตอบต่อไปจะได้จากการแทน Augmented Exit Expression ลงในกฎที่ได้จากขั้นตอนที่ผ่านมาจากนั้นก็ทำการ Evaluate ผลลัพธ์ที่ได้จนกว่าผลลัพธ์ที่ Evaluate จะเป็นเซ็ทว่างซึ่งกฎที่สร้างขึ้นนี้จะมีดังต่อไปนี้

Prereq(X,Y) <-- Imm_Prereq(X,Y)

Prereq(X,Y) <-- Imm_Prereq(X,Z),Imm_Prereq(Z,Y)

Prereq(X,Y) <-- Imm_Prereq(X,Z),Imm_Prereq(Z,D1),Imm_Prereq(D1,Y)

...

และเช่นเดียวกับตัวอย่างที่ผ่านมา โดยการสร้างเป็นภาษา SQL เพื่อหาคำตอบจากกฎข้างต้นซึ่งจะไม่แสดงตัวอย่างในกรณีนี้ และเมื่อทำการ Evaluate กฎแรกจะได้คำตอบเป็น 'Data Structure' และกฎ

ที่สองได้คำตอบเป็น 'Programming Language' และผลลัพธ์จากกฎที่สามจะได้เป็นเซ็ทว่างซึ่งหมายถึงการสิ้นสุดการทำงาน ดังนั้นสำหรับคำถามข้างต้นที่ว่านักศึกษาที่จะลงเรียนในวิชา 'Database' จะต้องลงเรียนในวิชาอะไรมาบ้าง ซึ่งคำตอบก็คือ 'Data Structure' และ 'Programming Language'

## 10.5 การพัฒนาโปรแกรมการจัดการกฎบนฐานข้อมูลอนุมาน

เมื่อเราทราบถึงหลักการทำงานของการจัดการกับกฎแล้ว ต่อไปเราก็จะกล่าวถึงส่วนของการพัฒนาโปรแกรมบ้าง แต่ก่อนอื่นขอชี้แจงในเรื่องของที่มาของกฎเสียก่อน ในขั้นตอนการออกแบบและสร้างแบบจำลองข้อมูลตามสถาปัตยกรรมที่ได้กล่าวไว้ในบทที่ 5 เราจะถือว่าการออกแบบกฎควรจะออกแบบพร้อม ๆ กับการออกแบบแบบจำลองข้อมูล โดยผู้ออกแบบฐานข้อมูล เพราะในการใช้งานในระดับแอปพลิเคชันนั้น ไม่ควรจะให้ผู้ใช้ต้องทำหน้าที่ออกแบบกฎเสียเอง ดังนั้นในโปรแกรมนี้จึงไม่อนุญาตให้ใส่กฎลงในขณะที่ประมวลผลโปรแกรมคำถามได้ แต่จะสร้างเป็นโปรแกรมแยกออกมาต่างหากเพื่อใช้งานในลักษณะของเครื่องมือเท่านั้น

และด้วยรูปแบบการทำงานดังกล่าว เราจึงแบ่งการทำงานที่เกี่ยวข้องกับกฎออกเป็น 2 ส่วนด้วยกัน คือ ส่วนเตรียมการและส่วนตอบคำถาม โดยส่วนเตรียมการจะทำหน้าที่เริ่มตั้งแต่รับกฎจากผู้ออกแบบคอนเซปชวลสกีมา ซึ่งจะอยู่ในรูปของข้อความธรรมดา (Text) จากนั้นก็จะนำข้อความดังกล่าวมาประมวลผลเบื้องต้น โดยเริ่มตั้งแต่การนำเอากฎมาสร้างในลักษณะของลิงก์ลิสต์ และตรวจสอบประเภทของกฎ และจัดเก็บลงในฐานข้อมูล สำหรับส่วนตอบคำถามก็จะเริ่มด้วยการนำข้อมูลจากตารางมาสร้างเป็นลิงก์ลิสก์และตรวจสอบชนิดของกฎ จากนั้นก็ประมวลผลตามประเภทของกฎต่อไป

### 10.5.1 ส่วนเตรียมการและประมวลผลเบื้องต้น

โปรแกรมที่ทำหน้าที่จัดการกฎบนฐานข้อมูลนั้น ถือได้ว่าเป็นส่วนหนึ่งของส่วนประมวลผลคำถาม (Query Processing) โดยคำถามที่สามารถใช้ได้กับระบบฐานข้อมูลอนุมานนั้น จะแบ่งออกเป็น 2 อย่างคือ แบบที่เป็นชนิดความจริง (Fact) และแบบที่เป็นกฎ (Rule) โดยส่วนที่เป็นชนิดความจริงนั้นจะเป็นเพียงการค้นหาข้อมูลในฐานข้อมูลธรรมดาเท่านั้น ดังนั้นการทำงานจึงไม่ซับซ้อนเท่ากับการทำงานของส่วนประมวลผลคำถามที่เป็นแบบกฎ หรือหากจะกล่าวว่าการจัดการกฎถือเป็นองค์ประกอบหลักของส่วนประมวลผลคำถามก็ไม่ผิดนัก

อย่างไรก็ตามในการเตรียมการนั้น เราจะเตรียมการเฉพาะกับคำถามที่เป็นแบบกฎเท่านั้น สำหรับคำถามแบบชนิดความจริงนั้น สามารถค้นหาคำตอบได้เลยโดยไม่ต้องเตรียมการล่วงหน้า ขั้นตอนการทำงานของส่วนเตรียมการจะเริ่มจากการอ่านข้อมูลของกฎทั้งหมด มาจัดเก็บในหน่วยความจำ โดยจะจัดเก็บในโครงสร้างของลิงก์ลิสต์ โดยจะมีโครงสร้างของลิงก์ลิสต์ดังต่อไปนี้

```
typedef struct PREDICATE {
                        char p_name[15];
                        char parameter[10][20];
                        unsigned char param_flag[10];
```

```
        unsigned char n_edge_flag;
        struct {
                struct PREDICATE *rule;
                unsigned char flag;
        }s_rule[10];
        struct PREDICATE *n_pred;
};
```

โดยมีรายละเอียดของแต่ละส่วนดังนี้

char p_name[15] : เก็บชื่อของพรีดิเขต ซึ่งมีความยาวสูงสุด 14 ตัว

char parameter[10][20] : เก็บอาร์กิวเมนต์ของพรีดิเขต ซึ่งมีความยาวสูงสุด 19 ตัว และจำนวนอาร์กิวเมนต์สูงสุด 10 อาร์กิวเมนต์

unsigned char param_flag[10] : แฟลกแสดงสถานะของอาร์กิวเมนต์

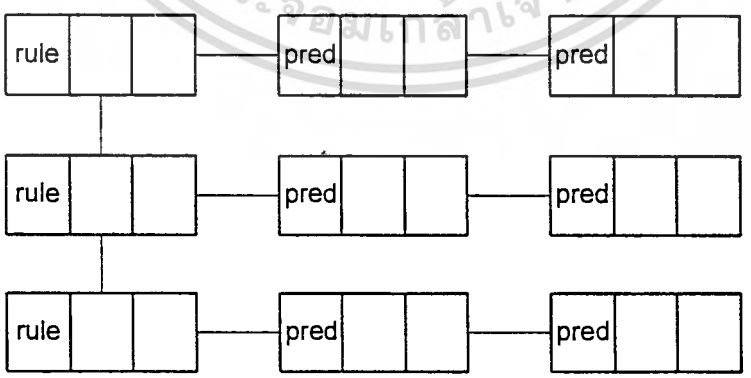
unsigned char n_edge_flag : แฟลกแสดงสถานะของพรีดิเขต

```
struct  {
        struct PREDICATE *rule;
        unsigned char flag;
    } s_rule[10]
```

: พอยเตอร์ชี้ไปยังกฎซึ่งมีได้สูงสุด 10 กฎ และ แฟลกแสดงสถานะของพรีดิเขตนั้น

struct PREDICATE *n_pred : พอยเตอร์ชี้ไปยังกฎถัดไป

จากรายละเอียดของโครงสร้างข้อมูลนี้จะได้รูปแบบของอินพุท ที่รับมาเป็นดังนี้




*รูปที่ 10-3 แสดงลักษณะของอินพุท*

จากนั้นจึงตรวจสอบว่ากฎทั้งหมดที่สร้างรับเข้ามานั้น กฎใดเป็นกฎแบบเรียกตัวเอง และกฎใดเป็นกฎแบบไม่เรียกตัวเอง ซึ่งสามารถตรวจสอบโดยการสร้าง CG (Connected Graph) ขึ้นมา โดยมีอัลกอริทึมดังต่อไปนี้

1. นำกฎที่อยู่ในลิงก์ลิสต์มาทีละกฎ

2. นำชื่อพรีดิเขตที่เป็นสมาชิกของแต่ละกฎมาทีละพรีดิเขต

3. นำชื่อพรีดิเขตที่อ่านมาไปทำการเปรียบเทียบกับชื่อของกฎทุกกฎที่มีอยู่ในลิสท์ ถ้ามีชื่อเหมือนกัน ให้พอยเตอร์ชี้ไปยังกฎนั้น

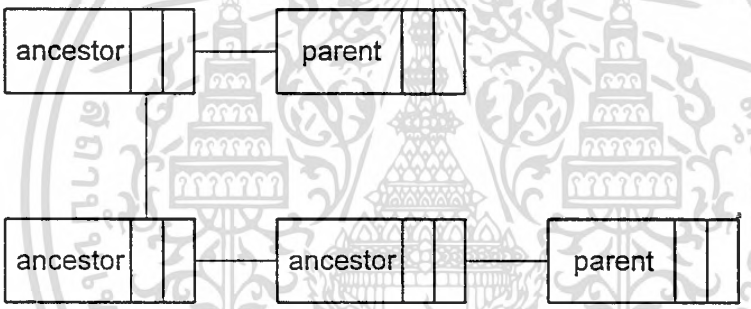
4. เมื่อตรวจสอบครบหมดทุกกฎให้กลับไปทำข้อ 2 ใหม่จนกว่าจะหมด

5. เปลี่ยนเป็นกฎถัดไป แล้วกลับไปทำข้อ 1 ใหม่จนกว่าจะหมด

ตัวอย่าง จากอัลกอริธึมข้างต้นจะสามารถสร้างซีจีของ

```
ancestor(X,Y) <- parent(X,Y).
ancestor(X,Y) <- ancestor(X,Z),parent(Z,Y).
```

ได้เป็น




*รูปที่ 10-4 แสดงอินพุทจากตัวอย่าง*

เมื่อตรวจสอบกฎทั้งหมดแล้ว ก็จัดเก็บลงในฐานข้อมูลโดยให้ระบุว่ากฎใดเป็นแบบเรียกตัวเอง (S) และกฎใดไม่เป็นแบบเรียกตัวเอง (N) โดยหากเป็นกฎแบบไม่เรียกตัวเองก็จะสร้างวิวที่แทนกฎแบบไม่เรียกตัวเองนั้นให้ด้วย และหากเป็นกรณีของกฎแบบเรียกตัวเอง แม้ว่าจะสร้าง Augmented Loop Residue และ Augmented Exit Expression เอาไว้ก่อนได้ แต่สำหรับโครงงานนี้เราจะสร้างในช่วงของการตรวจสอบแทน เพราะจะได้ง่ายต่อการส่งผ่านข้อมูล

การทำงานทั้งหมดข้างต้นจะถือว่าเป็นการทำงานส่วนหนึ่งในส่วนนิยามข้อมูล ซึ่งหมายความว่าการกำหนดกฎจะต้องกำหนดเอาไว้ก่อน จะมากำหนดหลังจากที่สร้างในภายหลังไม่ได้ ดังนั้นโปรแกรมในส่วนของการเตรียมการนี้จะแยกออกเป็นโปรแกรมต่างหาก จะไม่รวมกับส่วนของโปรแกรมหลัก ในส่วนของโปรแกรมหลังจะเป็นการค้นหาตามกฎที่สร้างไว้เท่านั้น โดยตารางที่ใช้จัดเก็บข้อมูลมีโครงสร้างดังต่อไปนี้

ตาราง VIEW_COLUMN

| view_name | column_name | column_type | column_order | column_width | prikey_id | deduc_id |
|---|---|---|---|---|---|---|
| char(30) | char(30) | char(30) | number | number | char(1) | char(1) |

ตาราง PREDICATE

| argument | arg_order | head_number | body_number |
|---|---|---|---|
| char(20) | number | number | number |

ตาราง RULE

| head | body | head_number | body_number |
|---|---|---|---|
| char(20) | char(20) | number | number |

10.5.2 การสร้างวิวให้กับกฎชนิดไม่เรียกตัวเอง

จากที่ได้กล่าวมาข้างต้น จะเห็นว่าขั้นตอนหนึ่งที่สำคัญมากในส่วนของการประมวลผลเบื้องต้น ก็คือ การสร้างวิวให้กับกฎชนิดไม่เรียกตัวเอง ซึ่งกฎชนิดไม่เรียกตัวเองนี้ แน่นอนว่าจะต้องประกอบด้วยกฎเพียงกฎเดียวเท่านั้น โดยในกฎนั้นจะมีพรีดิเขตในกฎเป็นจำนวนเท่าใดก็ได้ โดยคำว่าพรีดิเขตในที่นี้ก็หมายถึงชนิดความจริงนั่นเอง และเพื่อให้การอธิบายสามารถเข้าใจได้ง่าย ก็จะขอยกตัวอย่างประกอบก็แล้วกันนะครับ โดยสมมติว่ามีกฎชนิดไม่เรียกตัวเองอยู่กฎหนึ่ง ดังนี้

P3(X,Z) <- P1(X,Y),P2(Y,Z)

ในการสร้างวิวนั้นเราจะเริ่มจากการนำชื่อของพรีดิเขตแต่ละพรีดิเขต ไปค้นหาชื่อของชนิดความจริงที่สัมพันธ์กับพรีดิเขตนั้น ซึ่งก็จะเป็นการตรวจสอบด้วยว่ามีชนิดความจริงเหล่านั้นอยู่จริง เพราะการสร้างวิวจะไม่สามารถสร้างได้เลย หากไม่มีชนิดความจริงที่ตรงกับแต่ละพรีดิเขตในฐานข้อมูล จากนั้นก็จะสร้างคำสั่ง SELECT จากชื่อของโรลที่ประกอบเป็นชนิดความจริงนั้น สมมติว่า P1 มีชื่อโรลเป็น P1 (R1,R2) และ P2 มีชื่อโรลเป็น P2(R2,R3) ก็จะสร้างคำสั่ง SELECT เป็น

SELECT R1,R3

ต่อจากนั้นก็จะสร้างอนุประโยค FROM ให้กับคำสั่ง SELECT โดยใช้ชื่อของชนิดความจริงมาประกอบในอนุประโยค FROM ซึ่งจะได้เป็น

SELECT R1,R3
FROM P1,P2

และท้ายสุดก็จะสร้างอนุประโยค WHERE โดยค้นหาชื่อของโรลที่เหมือนกันในกฎ ซึ่งโดยทั่วไปแล้วในกฎแต่ละกฎจะต้องมีอากิวเมนต์ที่ร่วมกันเสมอ เช่น จากกฎข้างต้นก็จะเห็นว่ามีอากิวเมนต์ Y

ร่วมกัน ซึ่งจากอากิวเมนต์นี้เราก็จะนำมาเทียบหาชื่อโรลที่จะใช้เป็นเงื่อนไข ซึ่งก็จะได้เป็นโรล R2 นั่นเอง ดังนั้นจะสามารถสร้างอนุประโยค WHERE เพิ่มได้เป็น

```
SELECT R1,R3
FROM P1,P2
WHERE P1.R2 = P2.R2
```

และจะได้คำสั่งที่ใช้ในการสร้างวิวเป็น

```
CREATE VIEW P3(R1,R3) AS
SELECT R1,R3
FROM P1,P2
WHERE P1.R2 = P2.R2;
```

### 10.5.3 ส่วนประมวลผลคำถาม

การประมวลผลคำถามจะแบ่งออกเป็น 2 แบบ คือ การประมวลผลคำถามที่เป็นชนิดความจริง และการประมวลผลคำถามที่เป็นแบบกฎ โดยการประมวลผลคำถามที่เป็นแบบชนิดความจริงจะเริ่มจากการรับเพรดิเขตเข้ามาจากนั้นจะแยกส่วนประกอบเพรดิเขตออกเป็น ชื่อความสัมพันธ์ และชื่อของอากิวเมนต์ต่าง ๆ จากนั้นก็จะตรวจสอบว่าความสัมพันธ์นั้นเป็นชนิดความจริงหรือไม่ โดยค้นหาจากตาราง NORMAL_FACTTYPE หากพบว่าเป็นชนิดความจริงก็จะทำงานต่อ โดยนำส่วนอากิวเมนต์มาตรวจสอบว่าเป็นค่าคงที่หรือเป็นตัวแปร โดยหากเป็นค่าคงที่ก็จะค้นหาว่ามีชนิดความจริงนั้นอยู่ในฐานข้อมูลหรือไม่ หากมีก็จะได้คำตอบว่า TRUE แต่หากไม่มีก็จะได้คำตอบว่า FALSE แต่ถ้าเป็นกรณีของตัวแปรก็จะสร้างคิวรี่เพื่อค้นหาข้อมูลทั้งหมดที่สอดคล้องกับตัวแปรนั้น และแสดงผลออกทางจอภาพ

การประมวลผลคำถามที่เป็นกฎนั้นจะแบ่งออกเป็น 2 อย่าง คือ กฎแบบเรียกตัวเองและกฎแบบไม่เรียกตัวเอง โดยมีการทำงานเริ่มจากการอ่านกฎที่ได้เตรียมการไว้แล้วเข้ามาในระบบ จากนั้นจะนำกฎเหล่านั้นมาแยกพารามิเตอร์ต่าง ๆ ออก ได้แก่ ชื่อเพรดิเขต และเทอมต่าง ๆ โดยจัดเก็บลงในลิงก์ลิงค์ ซึ่งมีโครงสร้างข้อมูลตามที่ได้กล่าวไปข้างต้น จากนั้นก็แยกการดำเนินงานออกเป็น 2 แบบ คือ ในแบบกฎแบบไม่เรียกตัวเองและกฎแบบเรียกตัวเอง

### 10.5.4 การประมวลผลกฎแบบไม่เรียกตัวเอง

การทำงานในส่วนการประมวลผลคำถามจะเริ่มจากการรับคำถามมาจากผู้ใช้ จากนั้นจะนำคำถามนั้นไปค้นหาในตาราง VIEW_COLUMN ซึ่งจะเก็บรายละเอียดของคำถามทั้งในแบบเรียกตัวเอง และแบบไม่เรียกตัวเอง และหากพบว่าเป็นคำถามแบบไม่เรียกตัวเอง ก็จะสร้างคิวรีเพื่อค้นหาคำตอบจากวิวที่ได้สร้างเอาไว้ในขั้นตอนเตรียมการ โดยการทำงานจะเริ่มจากการค้นหาองค์ประกอบของวิวที่ได้สร้างไว้แล้วจากขั้นตอนในส่วนเตรียมการในตาราง VIEW_COLUMN

จากนั้นก็จะตรวจสอบคำถามว่าเป็นแบบค่าคงที่หรือเป็นแบบตัวแปร โดยหากเป็นค่าคงที่ก็จะนำมาใช้เป็นเงื่อนไขในอนุประโยค WHERE และเพื่อให้เข้าใจ เรามาลองยกตัวอย่างกัน สมมติว่ามีกฎและคำถามดังนี้

```
ancester(X,Y) <- parent(X,Z),parent(Z,Y).
ancester(X,Y).
ancester(X,'john').
ancester('mary','john');
```

ในกรณีของคำถามแรกจะเห็นได้ว่าไม่มีค่าคงที่อยู่เลย ดังนั้นการประมวลผลคำถามในแบบแรกจึงง่ายมาก เราเพียงแต่สร้างประโยค SELECT จากวิวตามธรรมดาก็ใช้ได้แล้ว โดยจะได้เป็น

```
SELECT R1,R3
FROM ANCESTOR;
```

ทั้งนี้เพราะ Ancester เป็นวิวอยู่แล้วจึงสามารถ SELECT ได้โดยตรง แต่หากมีเงื่อนไขตามแบบคำถามที่ 2 เราก็จะเพิ่มอนุประโยค WHERE เข้าไป ซึ่งจะได้เป็น

```
SELECT R1,R3
FROM ANCESTOR
WHERE R3 = 'john';
```

ในกรณีขอคำถามในแบบที่ 3 การทำงานก็จะเป็นเช่นเดิม โดยจะนำเอาค่าคงที่ทั้งสองค่ามาใช้เป็นเงื่อนไขในอนุประโยค WHERE ดังนั้นสามารถสร้างคำถามได้เป็น

```
SELECT R1,R3
FROM ANCESTOR
WHERE R1 = 'mary' AND R3 = 'john';
```

แต่การแสดงผลจะไม่แสดงผลเป็นข้อมูลที่เก็บในชนิดความจริงเหมือนกับคำถามสองแบบแรก แต่จะแสดงผลในรูปของ TRUE และ FALSE โดยตรวจสอบว่าหากมีข้อมูลส่งกลับมาก็จะหมายความว่ามีข้อมูลนั้นอยู่จริง ก็จะแสดงผลเป็นค่า TRUE และหากไม่มีข้อมูลส่งกลับมาก็จะแสดงผลเป็น FALSE อนึ่ง รูปแบบการประมวลผลคำถามแบบไม่เรียกตัวเองนี้ จะมีรูปแบบที่เหมือนกับการประมวลผลคำถามแบบชนิดความจริงเช่นกัน เพราะในทางฐานข้อมูลแล้ว คำถามแบบชนิดความจริงกับวิวนี้มีส่วนคล้ายกันมาก จนสามารถใช้วิธีประมวลร่วมกันได้ ดังนั้นเราจะข้ามส่วนประมวลผลชนิดความจริงไปเลย ขอให้เข้าใจตามนี้นะครับ

### 10.5.5 การประมวลผลกฎแบบเรียกตัวเอง

ในส่วนของโปรแกรมที่จัดการทำงานส่วนที่เกี่ยวกับกฎแบบเรียกตัวเองนี้ รายละเอียดค่อนข้างมาก จึงจะอธิบายในรูปของโปรแกรม โดยจะแบ่งออกเป็น 3 โปรแกรมด้วยกัน คือ knwldg.pc, query.c, residue.c โดยมีรายละเอียดการทำงานของแต่ละไฟล์ดังต่อไปนี้

#### 10.5.5.1 ไฟล์ knwldg.pc

เป็นไฟล์หลักในการทำงานของส่วนนี้ จะรับกฎและคิวรีมาจากโปรแกรมที่เรียกใช้โปรแกรมนี้ และนำกฎและคิวรีที่รับมาทำกระทำการ โดยมีการทำงานโดยสรุปดังต่อไปนี้

ฟังก์ชัน main จะรับอาร์กิวเมนต์มาจากโปรแกรมที่เรียกใช้ ซึ่งประกอบด้วยแอดเดรสที่ชี้ไปยังหน่วยความจำที่ใช้เก็บกฎและคิวรีในรูปของตัวอักษร ดังนั้นจะต้องเปลี่ยนให้อยู่ในรูปตัวเลขก่อน แล้วนำค่าแอดเดรสเก็บไว้ในตัวแปร rule และ query จากนั้นจะตรวจสอบว่ากฎนี้เป็นกฎแบบเรียกตัวเองหรือไม่ โดยการเรียกฟังก์ชัน is_recursive ถ้าเป็นก็จะทำตามขั้นตอนของการจัดการกฎแบบเรียกตัวเอง โดยเริ่มจากการสร้าง augmented loop residue และ augmented exit expression จากการเรียกใช้ฟังก์ชัน make_res เก็บไว้ในตัวแปร res และ express ตามลำดับ จากนั้นจะสร้างกฎแบบไม่เรียกตัวเองขึ้นมาทีละกฎโดยใช้ฟังก์ชัน mix และฟังก์ชัน make_query จะสร้างคิวรีแบบเอสคิวแอลพร้อมทั้งนำคิวรีนี้ไปรันแล้ว Insert ผลลัพท์ลงในตารางที่สร้างขึ้น ทำวนลูปไปเรื่อย ๆ จนไม่สามารถดึงข้อมูลมาได้อีก ก็จะออกจากโปรแกรม พร้อมทั้งเรียกฟังก์ชัน logout เพื่อออกจากการติดต่อกับฐานข้อมูล

ฟังก์ชัน column_info มีหน้าที่ในการหาข้อมูลของคอลัมน์จากตารางที่กำหนดมา โดยรับอาร์กิวเมนต์เป็นชื่อตาราง และลำดับของคอลัมน์ที่ต้องการรายละเอียด โดยดูรายละเอียดจากตาราง view_column จะได้ชื่อ ชนิด และความกว้างในกรณีที่เป็นตัวอักษรของคอลัมน์ และคืนค่ารายละเอียดนี้ในตัวแปร c_name,c_type และ width ตามลำดับ

ฟังก์ชัน run_create จะรับคิวรีที่สั่งให้สร้างตารางมารัน ฟังก์ชัน run_insert จะรับคิวรีที่ฟังก์ชัน make_query สร้างขึ้น มาทำการรัน ฟังก์ชัน login จะทำหน้าที่ในการ login เข้าสู่ระบบฐานข้อมูล และฟังก์ชัน logout จะทำการ logout ออกจากระบบฐานข้อมูลเมื่อจบโปรแกรม สำหรับฟังก์ชัน is_recursive จะตรวจสอบว่าชื่อตารางที่เป็นอาร์กิวเมนต์ส่งมานี้ มีชนิดเป็นแบบเรียกตัวเองหรือไม่ โดยดูค่าจากตาราง view_column แล้วคืนค่ากลับเป็นจริงหรือเท็จ

#### 10.5.5.2 ไฟล์ query.c

ทำหน้าที่ในการสร้าง SQL Query โดยรับอาร์กิวเมนต์เป็นลิงค์ลิสท์ของกฎ คิวรีที่เป็นคำถามที่ต้องการถาม เมื่อเข้าสู่โปรแกรมจะสร้างสตริงค์ที่จะนำมาสร้างเป็นคิวรี เสร็จแล้วจะนับจำนวนตัวแปรที่มีอยู่ในคำถาม นับจำนวนพรีดิเขตในส่วนตัวของกฎ เมื่อทราบจำนวนพรีดิเขตแล้วจะสร้างชื่อตารางให้กับพรีดิเขตเหล่านี้ โดยเรียกใช้ฟังก์ชัน get_name นำมาใส่ไว้ในอาเรย์ table_name

หลังจากได้ชื่อตารางเสร็จแล้วก็จะสร้างคิวรีในช่วงของการ select โดยจะดูว่าต้องใช้คอลัมน์ไหนจากตารางไหนบ้าง โดยการพิจารณาชื่ออาร์กิวเมนต์ของพรีดิเขตที่เป็นส่วนหัวของกฎกับส่วนตัว ถ้าเป็น

ตัวแปรเดียวกันก็จะใช้คอลัมภ์จากตารางนั้น ซึ่งจะทำการเรียกฟังก์ชัน update_select เพื่อทำหน้าที่ในการเชื่อมต่อสตริงค์ วนลูปทำจนครบหมดทุกตัวแปรในส่วนหัวของกฎ ก็จะได้ส่วนของ select เพื่อตรวจสอบว่าเป็นการเรียกฟังก์ชันนี้ครั้งแรกหรือไม่ ถ้าเป็นก็จะสร้างตารางขึ้นมาเก็บผลลัพท์ โดยการเรียกฟังก์ชัน mk_creat_query เพื่อสร้างคิวรีที่จะสร้างตารางและ run_create เพื่อนำคิวรีที่สร้างขึ้นมารันเพื่อสร้างตารางขึ้นมา

ต่อไปจะสร้างคิวรีในส่วนของ from โดยวนลูปไล่ส่วนตัวของกฎไปทีละพรีดิเขต พร้อมกับนำชื่อที่เก็บไว้ในอาเรย์ table_name[] โดยการเรียกใช้ฟังก์ชัน update_from ได้คิวรีในส่วนของ from และหลังจากได้ส่วนของ from แล้วจะสร้างส่วน where ซึ่งประกอบด้วย 2 ส่วนคือ ส่วนของค่าคงที่ที่อยู่ในคำถาม โดยตรวจสอบจากฟังก์ชัน find_parameter_const ได้ ตำแหน่งของค่าคงที่เก็บไว้ในตัวแปร n เรียกฟังก์ชัน match_pred_npara เพื่อตรวจสอบหาว่าค่าคงที่นี้อยู่ในคอลัมน์และตารางไหน เติมเครื่องหมาย " ' " เข้าไปถ้าค่าคงที่นี้เป็นชนิดตัวอักษรโดยเรียกใช้ฟังก์ชัน insert_quote เพื่อสร้างชื่อตารางจากฟังก์ชัน make_name เมื่อครบแล้วก็จะเชื่อมต่อสตริงค์โดยเรียกใช้ฟังก์ชัน update_where วนลูปไปจนกระทั่งไม่พบค่าคงที่ อีกส่วนหนึ่งคือส่วนของตัวแปรที่เชื่อมโยงกันภายในกฎ โดยวนไล่ทีละพรีดิเขต ในแต่ละพรีดิเขตจะดูแต่ละอาร์กิวเมนต์ถ้าเป็นค่าคงที่ก็จะทำการเชื่อมต่อค่าคงที่ ถ้าเป็นตัวแปรก็จะดูว่าตัวแปรนี้มีอยู่อีกหรือไม่โดยการเรียกใช้ฟังก์ชัน match_var ถ้าพบก็จะเพิ่มเติมเข้าไปในคำสั่ง where ทำไปเรื่อย ๆ จนครบหมดทุกพรีดิเขต หลังจากได้ครบทั้ง 3 ส่วนแล้ว ก็จะนำคำสั่งทั้งหมดมาต่อกันเป็นคิวรี เรียกฟังก์ชัน run_insert ถ้าสามารถดึงข้อมูลมาได้ก็จะคืนค่ากลับว่าจริง มิฉะนั้นแล้วก็จะคืนค่าเป็นเท็จ

ฟังก์ชัน get_name จะสร้างชื่อตารางขึ้นมาโดยเรียงหมายเลขไปเรื่อย ๆ และฟังก์ชัน update_select จะเชื่อมต่อคำในช่วงของคำสั่ง select โดยรับอาร์กิวเมนต์ temp เป็นพอยท์เตอร์ชี้ไปยังสตริงค์ที่ต้องการเชื่อม first จะเป็นตัวบอกว่าเป็นการเรียกใช้ครั้งแรก ซึ่งจะไม่มีการเติมเครื่องหมาย "," เข้าไป หลังจากนั้นจะนำชื่อตารางมาต่อจากตัวแปร table_name ตามด้วยเครื่องหมาย "." เรียกฟังก์ชัน column_info เพื่อนำชื่อของคอลัมน์มาจากฐานข้อมูล โดยบอกลำดับของคอลัมน์ที่ตัวแปร num_column ได้ผลลัพธ์มาเก็บไว้ที่ตัวแปร t_into->attr_name นำชื่อที่ได้มาเชื่อมต่อ จะได้เป็นคำสั่งที่สมบูรณ์

ฟังก์ชัน update_where จะทำหน้าที่ในการเชื่อมต่อคำสั่ง where รับอาร์กิวเมนต์เป็นพอยเตอร์ชี้ไปยังสตริงค์ที่ต้องการเชื่อมต่อคือ temp ตัวแปร first จะบอกให้ทราบว่าเป็นการเรียกใช้ครั้งแรกก็จะไม่ทำการเชื่อมเครื่องหมาย "," นำ table_name ซึ่งเป็นชื่อของตารางมาเชื่อม ตามด้วยชื่อเสมือนของตาราง

ฟังก์ชัน mactch_var จะหาตัวแปรที่ใช้ร่วมกันในกฎ โดยรับค่าพอยเตอร์ชี้ไปยังตำแหน่งปัจจุบันของเพรดิเคตในตัวแปร current ตำแหน่งของตัวแปรใน count ทำการเปรียบเทียบค่าโดยเริ่มตั้งแต่ตัวแปรถัดไปในพรีดิเขตนี้ วนลูปไล่ไปทีละตัวแปรในแต่ละพรีดิเขตจนครบทุกพรีดิเขต ถ้าพบก็จะคืนค่าพอยท์เตอร์ชี้ไปยังพรีดิเขตและลำดับของตัวแปรที่พบ และคืนค่ากลับเป็นจริง ถ้าไม่พบก็จะคืนค่ากลับเป็นเท็จ

ฟังก์ชัน find_parameter_const มีหน้าที่ในการหาค่าคงที่ที่อยู่ในพรีดิเขต โดยวนลูปไปจนครบทุกตัว ถ้าพบก็จะคืนค่ากลับเป็นลำดับของตัวที่เจอ สำหรับฟังก์ชัน mk_creat_query จะสร้างคิวรีที่ทำหน้าที่ในการ create ตารางที่ใช้เก็บผลลัพท์ รับอาร์กิวเมนต์เป็นพอยท์เตอร์ชี้ไปยังอาเรย์ที่ใช้เก็บข้อมูล

ของตาราง ชื่อตารางที่จะสร้าง เมื่อเข้าสู่โปรแกรมจะเชื่อมสตริงค์ของคำสั่ง หลังจากนั้นจะนำข้อมูลของตารางมาทำการสร้างเป็นรายละเอียดของตารางที่จะสร้าง วนลูปไปเรื่อย ๆ จนครบทุกคอลัมน์

10.5.5.3 ไฟล์ Residue.c

ประกอบด้วยฟังก์ชันต่าง ๆ ดังนี้

ฟังก์ชัน genstr(char *string) ทำหน้าที่สร้างตัวแปรประเภทสตริงที่มีรูปแบบดังนี้ @n โดย n คือตัวเลขจำนวนเต็ม ในการเรียกใช้ฟังก์ชันนี้จะสร้างสตริงที่ไม่ซ้ำค่ากันออกมาแต่ตัวเลขจำนวนเต็มที่ตามหลัง @ จะมีลำดับก่อนหลังตามการเรียกใช้

ฟังก์ชัน upgrade(struct PREDICATE *res) ทำหน้าที่ในการแก้ตัวแปรทุกตัวในกฎ ให้อยู่ในรูปแบบ @n โดย n คือตัวเลขจำนวนเต็ม ซึ่งทำได้โดยการเรียกใช้ฟังก์ชัน genstr อีกทีหนึ่ง ในฟังก์ชันนี้มีจุดที่น่าสังเกตคือ ตัวแปรในภาษาโปรลอกนั้นจะต้องขึ้นต้นด้วยตัวอักษรภาษาอังกฤษตัวพิมพ์ใหญ่ ( 'A' - 'Z' ) ที่สำคัญ คือหากเราแก้ตัวแปร 'A' เป็น @1 แล้ว ทุกที่ในกฎนั้นที่มีตัวแปร 'A' จะต้องแก้เป็น @1 ทั้งหมดด้วย

ฟังก์ชัน dupval(struct PREDICATE *source,struct PREDICATE *dest) ทำหน้าที่เก็บค่าทุกค่าใน struct PREDICATE *dest ให้เหมือนกับ struct PREDICATE *source ทั้งหมด ซึ่งเป็นการ COPY ค่าในระดับ struct PREDICATE 1 ตัว

ฟังก์ชัน construct(struct PREDICATE *source,struct PREDICATE **dest, struct PREDICATE **end) เป็นการ COPY ค่าในระดับลิงค์ลิสต์ จึงมีการเรียกใช้ฟังก์ชัน dupval อีกต่อหนึ่ง โดยมีต้นฉบับคือ source ส่วนลิงค์ลิสต์ผลลัพธ์คือ dest โดยมี end เป็นตัวชี้ struct PREDICATE ตัวสุดท้ายในลิงค์ลิสต์ dest ซึ่งลิงค์ลิสต์ในที่นี้คือ กฎ 1 กฎ ที่มี struct PREDICATE ตัวแรกเป็นส่วนหัวของกฎ นอกนั้นในลิงค์ลิสต์จะเป็นส่วนหางของกฎทั้งหมด

ฟังก์ชัน instead(struct PREDICATE *source,struct PREDICATE *dest) ทำหน้าที่ในการแทนค่าตัวแปรใน dest ให้เหมือนกับใน source หรืออาจกล่าวว่าเป็นส่วนหนึ่งของขั้นตอน unification ก็ว่าได้ โดย source ก็คือตัวชี้ส่วนหางตัวหนึ่งในกฎใดกฎหนึ่ง ส่วน dest ก็คือตัวชี้กฎที่ตรงกับส่วนหางที่ source ชี้กล่าวคือเป็นกฎที่จะต้องถูกเรียกใช้หรือถูก unification นั่นเอง

ในขั้นตอนการทำเราจะใช้อาเรย์ 2 มิติขนาด 10*20 จำนวน 2 ชุดคือ old และ new โดย 10 คือจำนวนอาร์กิวเมนต์สูงสุดใน struct PREDICATE 1 ตัว 20 คือความยาวสูงสุดของชื่ออาร์กิวเมนต์หน่วยเป็น ตัวอักษรและเราจะใช้ new เก็บอาร์กิวเมนต์ของ source ส่วน old จะใช้เก็บอาร์กิวเมนต์ของส่วนหัวของกฎใน dest หลังจากนั้นเราจะเช็คว่าในส่วนหางของ dest มีตัวแปรใดบ้างที่ตรงกับ old และ ตรงกับ old ตัวที่เท่าไร สมมุติว่าตรงกับ old ตัวที่ i เราจะแก้ให้ ตัวแปรตัวนั้นมีค่าเท่ากับ new ตัวที่ i ทำเช่นนี้กับส่วนหางทุกตัวใน dest

ฟังก์ชัน recur(struct PREDICATE **res,struct PREDICATE *head, struct PREDCATE ** pre_end) เป็นฟังก์ชันที่เขียนขึ้นเพื่อสร้าง Augmented Loop Residue โดยเราจะจัดการในส่วนของกฎที่อยู่ใน PRL เท่านั้น โดยฟังก์ชันนี้ถูกเขียนแบบเรียกตนเองด้วยเช่นกันจึงอาจจะเข้าใจยากสักหน่อย แต่

ขั้นตอนหลักๆ คือ การพยายามค้นหาเส้นทางที่อยู่ใน PRL ให้พบแล้วแทนส่วนหางที่พบว่าเป็นเส้นทางที่อยู่ใน PRL ด้วยส่วนหางของกฎที่ตรงกันนั้นอีกที ทำเช่นนี้ไปเรื่อยๆ จนพบว่าเส้นทางที่เราค้นหานั้นกลับมาถึง struct PREDICATE *head แล้ว นั่นคือผลลัพธ์จะเก็บที่ res สำหรับ pre_end นั้นเป็นประโยชน์ในช่วงที่กลับมาจากการเรียกตนเองของฟังก์ชันนี้แล้วเพราะ pre_end->n_pred จะต้องถูกปรับให้ชี้ที่ตัวที่ถูกต้อง เพื่อให้ลิงค์ลิสต์ไม่ขาดตอน

ฟังก์ชัน non_recur(struct PREDICATE **res,char *name,struct PREDICATE **pre_end) ฟังก์ชันนี้เป็นฟังก์ชันที่เรียกโปรแกรมส่วนที่จัดการกฎแบบไม่เรียกตนเองมาทำงาน

ฟังก์ชัน exex(struct PREDICATE **res,struct PREDICATE **pre_end) ฟังก์ชันนี้ทำหน้าที่ในการสร้าง Augmented Exit Expression มีหลักการทำงานคล้ายกับฟังก์ชัน recur หากแต่ว่าเราจะต้องเช็คการออกจาก PRL ว่าเส้นทางไหนกันแน่ที่ทำให้ออกจาก PRL ได้เร็วที่สุด

ฟังก์ชัน printres(struct PREDICATE *name) ทำหน้าที่ในการพิมพ์ค่าในลิงค์ลิสต์ name ออกทางหน้าจอทั้งหมดเพื่อประโยชน์ในการแก้ไขโปรแกรมในภายหลัง

ฟังก์ชัน enum BOOL mkres(struct PREDICATE *graph,struct PREDICATE **res, struct PREDICATE **express,char *name,enum BOOL *flag1) เป็นฟังก์ชันที่จะเรียกใช้ฟังก์ชัน recur และ non_recur เพื่อใช้ในการสร้าง Augmented Loop Residue และ เรียกใช้ฟังก์ชัน exex เพื่อสร้าง Augmented Exit Expression โดยมี

*graph เป็นอินพุทซึ่งในที่นี้ก็คือ CG

**res เป็นเอาท์พุทก็คือ Augmented Loop Residue

**express เป็นเอาท์พุทคือ Augmented Exit Expression

*name เป็นอินพุทที่ซึ่งก็คือ ชื่อ Query

*flag1 เป็นเอาท์พุทที่จะบอกว่า Augmented Exit Expression ในที่นี้เป็นกฎธรรมดา ๆ ที่มีอยู่ในฐานความรู้อยู่แล้วหรือไม่ กล่าวคือหาก flag1=TRUE หมายความว่า Augmented Exit Expression เป็นกฎที่มีอยู่แล้วในฐานความรู้ แต่หาก flag1=FALSE หมายความว่า เราจะต้องสร้างมันขึ้นมาเองด้วยการเรียกใช้ฟังก์ชัน exex ค่าที่ return จากฟังก์ชันนี้จะเป็นการบ่งชี้ว่า Query ที่ส่งมานั้นมีอยู่ในฐานความรู้หรือไม่ และที่สำคัญคือ ต้องใช้กฎแบบเรียกตนเองหรือไม่ เพราะหากไม่ใช้กฎแบบเรียกตนเอง ก็ไม่มีความจำเป็นจะต้องสร้าง Augmented Loop Residue และ Augmented Exit Expression กล่าวคือมันจะเป็นตัวบอกได้ว่าฟังก์ชันนี้ประสบความสำเร็จหรือไม่ในการสร้าง Augmented Loop Residue และ Augmented Exit Expression

ฟังก์ชัน devariable(struct PREDICATE *res) ทำหน้าที่ในการแปลงตัวแปรจากรูปแบบ @n เป็นรูปแบบปกติคือขึ้นต้นด้วยตัวอักษรภาษาอังกฤษตัวพิมพ์ใหญ่ ในฟังก์ชันนี้เราทำได้ง่าย ๆ ด้วยการเปลี่ยนจาก @ เป็น A เท่านั้นก็เรียบร้อย เพราะฉะนั้นจะเห็นว่าฟังก์ชันนี้ทำหน้าที่ตรงข้ามกับฟังก์ชัน genstr ดังนั้น อาจมีผู้สงสัยว่าแล้วทำไมเราต้อง เปลี่ยนรูปแบบไปมาให้มากเรื่องด้วย ? เหตุที่เราทำเช่นนี้ก็เพราะว่า ในกฎต่างๆกันสามารถใช้ชื่อตัวแปรซ้ำกันได้ ทั้งที่ความจริงแล้วจะถูกมองเหมือนเป็นตัวแปร

คนละตัวกันก็ตามที ในการสร้าง Augmented Loop Residue และ Augmented Exit Expression นั้นมีการผสมกันระหว่างหลายกฎเพื่อให้ได้ผลลัพธ์มา

เพราะฉะนั้น เป็นไปได้ที่จะมีตัวแปรที่มีชื่อซ้ำกันมาอยู่ร่วมกันใน Augmented Loop Residue หรือ Augmented Exit Expression ซึ่งมันจะผิด เราจึงจำเป็นต้องสร้างรูปแบบหนึ่งขึ้นมาเพื่อสร้างความแตกต่างระหว่างตัวแปรที่มีชื่อซ้ำกัน และที่เราต้องใช้รูปแบบแปลกๆ ไปก่อนแล้วค่อยแปลงกลับเป็นแบบเดิมภายหลังก็เพื่อความสะดวกรวดเร็วและทำให้โปรแกรมแยกแยะได้ว่าตัวแปรใดที่เราสามารถแยกแยะได้แล้วว่าจะไม่ซ้ำกับตัวแปรอื่นๆอีก เราก็ไม่จำเป็นจะต้องแปลงมันอีก มิฉะนั้นจะเกิดการแปลงซ้ำซ้อนขึ้นได้ อันจะทำให้เสียเวลาเปล่าๆ

ฟังก์ชัน mix(struct PREDICATE *res,struct PREDICATE *express, struct PREDICATE ** result,enum BOOL flag) ทำหน้าที่ในการผสม Augmented Loop Residue และ Augmented Exit Expression เป็นกฎแบบไม่เรียกตนเองกฎหนึ่งซึ่งสามารถนำไปใช้ดึงข้อมูลจากฐานความรู้ได้ โดย

*res คือ อินพุท ซึ่งเป็น Augmented Loop Residue

*express คือ อินพุท ซึ่งเป็น Augmented Exit Expression

**result คือ เอาท์พุทที่ได้จากการผสม Augmented Loop Residue กับ Augmented Exit Expression

flag คือ อินพุท หรือ แฟลกที่เป็นตัวบอกว่า Query ที่เป็นอินพุทของระบบนี้มีชื่อตรงกับกฎแบบไม่เรียกตนเองและกฎแบบเรียกตนเองทั้งสองอย่างพร้อมกันหรือไม่ หาก flag=TRUE หมายความว่าในการหาคำตอบของกฎแบบเรียกตนเองนี้ครั้งแรกสามารถใช้กฎแบบไม่เรียกตนเองได้เลยโดยไม่ต้องคำนึงถึงการผสมกันระหว่าง Augmented Loop Residue และ Augmented Exit Expression

อย่างไรก็ตามในขั้นตอนการผสมกันระหว่าง Augmented Loop Residue และ Augmented Exit Expression แต่ละครั้งของการเรียกใช้ฟังก์ชันนี้จะได้ผลลัพธ์ที่ไม่เหมือนเดิม โดยยิ่งเรียกผลลัพธ์ก็จะยิ่งมีความยาวมากขึ้น ซึ่งเป็นไปตามวิธีของ Henschen และ Naqvi ทุกประการ เราจะเช็คได้ว่า เป็นการเรียกใช้ฟังก์ชันนี้ครั้งแรกหรือไม่ ด้วยการเช็คที่ result ว่าเท่ากับ NULL หรือไม่ ซึ่งหากเท่าหมายความว่า เป็นการเรียกใช้ฟังก์ชันนี้ครั้งแรก ถ้าพิจารณาให้ดีจะพบว่าหาก result ไม่เท่ากับ NULL เราจะนำไปต่อกับ Augmented Loop Residue เพื่อสร้าง result ตัวใหม่ที่ยาวขึ้นโดยไม่เสียเวลาด้วย

ฟังก์ชัน freepred(struct PREDICATE *graph) เป็นฟังก์ชันที่ใช้ในการคืนหน่วยความจำให้กับระบบ โดยจะคืนหน่วยความจำในส่วนของกฎ โดยกระทำการในระดับ 1 กฎ ฟังก์ชันนี้ถูกเขียนแบบเรียกตนเองเพื่อให้สั้นเข้าไว้ และจะถูกเรียกใช้จากฟังฌ์ชัน freegr อีกที

ฟังก์ชัน freegr(struct PREDICATE *graph) เป็นฟังก์ชันที่ใช้ในการคืนหน่วยความจำให้กับระบบ โดยจะกระทำการในระดับ CG และมีการเรียกใช้ฟังก์ชัน freepred อีกต่อหนึ่ง เหตุที่ฟังก์ชันนี้ถูกเขียนแบบเรียกตนเองก็เพื่อให้โปรแกรมสั้นนั่นเอง

# บทที่ 11
# การทดสอบระบบฐานข้อมูลอนุมาน

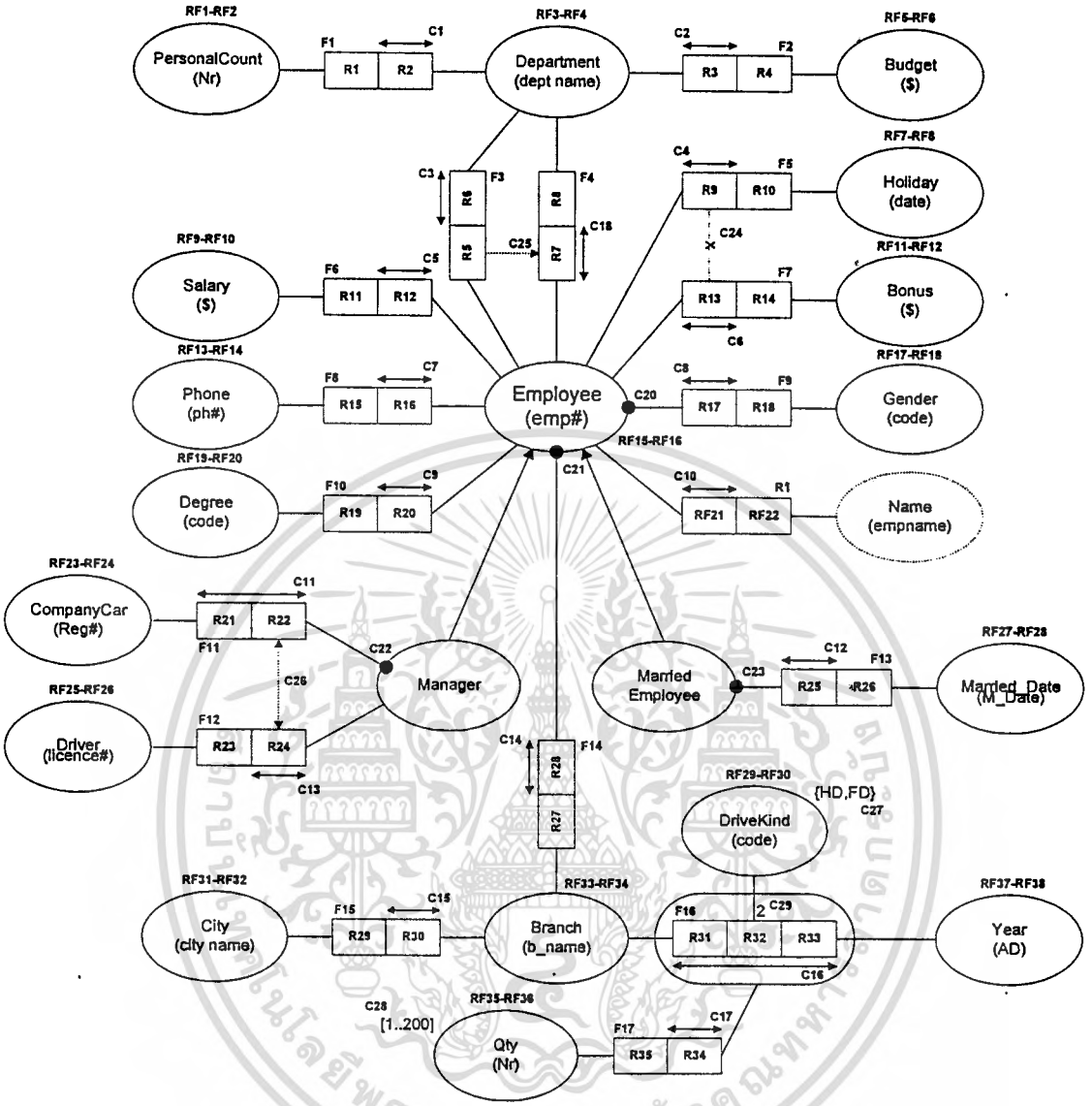
ในบทที่ผ่านมาเราได้กล่าวถึงทฤษฎีต่าง ๆ ที่จำเป็นต้องทราบในการทำวิทยานิพนธ์ฉบับนี้ ได้กล่าวถึงสถาปัตยกรรมของระบบทั้งหมดที่เราได้สร้างขึ้น ได้กล่าวถึงการออกแบบเมต้าสกีมา และได้กล่าวถึงรายละเอียดในส่วนต่าง ๆ ของระบบทั้งหมด นอกจากนั้นเรายังได้กล่าวถึงการทดสอบในแต่ละส่วนย่อยไปหมดทุกส่วนอีกด้วย

## 11.1 แนวทางการทดสอบ

สำหรับเนื้อหาในบทนี้ก็จะเป็นการทดสอบการทำงานของซอฟต์แวร์ขั้นสุดท้าย โดยจะเริ่มจากการกำหนดตัวอย่างข้อมูลในรูปของแบบจำลองข้อมูลไนแอม จากนั้นก็แปลงให้อยู่ในรูปของตารางเพื่อจัดเก็บลงในเมต้าสกีมา ซึ่งแบบจำลองข้อมูลนี้เราได้ออกแบบให้ครอบคลุมองค์ประกอบทุก ๆ ส่วนที่เราต้องการทดสอบ เช่น จะต้องมีการใช้งานในลักษณะของสับไทป์ ต้องมีการใช้งานในลักษณะของ Nesting และต้องมีการใช้คอนสแตนต์ครบทุกคอนสแตนต์ นอกจากนั้นเราก็ยังกำหนดกฎที่จะใช้ทดสอบกฎ โดยให้มีทั้งกฎชนิดเรียกตัวเองและไม่เรียกตัวเองอีกด้วย ซึ่งผลจากการออกแบบก็ได้ตามรูปที่ 11-1 อนึ่ง แบบจำลองข้อมูลในรูปที่ 11-1 นั้นเป็นแบบจำลองที่สร้างขึ้นมาโดยมุ่งหวังจะใช้ทดสอบเป็นสำคัญ ดังนั้นลักษณะการเก็บข้อมูลก็อาจจะดูไม่สมเหตุผลในบางประการได้ ขอให้เข้าใจตามนี้นะครับ

แบบจำลองข้อมูลในรูปที่ 11-1 เป็นข้อมูลส่วนหนึ่งของบริษัทแห่งหนึ่ง โดยเป็นข้อมูลรายละเอียดของพนักงานทุกคนในบริษัท โดยพนักงานแต่ละคนจะระบุโดยใช้เลขประจำตัวพนักงาน ในการบันทึกข้อมูลพนักงานนั้น สามารถบันทึกข้อมูลชื่อ เพศ ระดับการศึกษา เบอร์โทรศัพท์ เงินเดือน แผนกที่ทำงาน และสาขาที่ทำงาน โดยมีข้อแม้ว่าพนักงานทุกคนจะได้ต้องได้รับการบันทึกข้อมูลเพศ ชื่อแผนกและสาขาที่ทำงาน สำหรับพนักงานที่แต่งงานแล้ว จะได้รับการบันทึกข้อมูลปีที่แต่งงานเพิ่มเติมด้วย และสำหรับพนักงานในระดับผู้จัดการขึ้นไป จะได้รับรถประจำตำแหน่งโดยจะต้องบันทึกว่าผู้จัดการคนไหนได้รับรถเลขทะเบียนอะไร และจะต้องมีการบันทึกเลขทะเบียนใบขับขี่ไว้ควบคู่กันอีกด้วย

ในแต่ละปีบริษัทอนุญาติให้พนักงานลาพักร้อนได้ 7 วันโดยจะต้องมีการบันทึกวันที่พนักงานแต่ละคนได้ไปพักร้อนเอาไว้ และสำหรับพนักงานที่ไม่ไปพักร้อนทางบริษัทมีนโยบายให้โบนัสเป็นรางวัลอีกด้วย สำหรับรายละเอียดของแผนกนั้นจะสามารถบันทึกข้อมูลงบประมาณประจำปี บันทึกจำนวนพนักงานประจำแผนก และบันทึกรหัสประจำตัวของหัวหน้าแผนก โดยการระบุถึงแผนกแต่ละแผนกสามารถทำได้โดยระบุชื่อแผนก นอกจากนั้นแล้วก็ยังให้มีการบันทึกที่ตั้งของสาขาแต่ละแห่ง รวมทั้งมีการบันทึกยอดขายของสาขาต่าง ๆ โดยผลิตภัณฑ์ที่มีการผลิตจำหน่าย คือ ฮาร์ดดิสก์ (HD) และฟลอปปี้ดิสก์ (Floppy Disk) ในแต่ละปีอีกด้วย โดยในแต่ละปีจะต้องบันทึกยอดขายทั้งฮาร์ดดิสก์และฟลอปปี้ดิสก์เสมอ




*.รูปที่ 11-1 แสดงแบบจำลองข้อมูลที่ใช้ในการทดสอบ*

และจากแบบจำลองข้อมูลในรูปที่ 11-1 เราก็จะแปลงข้อมูลให้อยู่ในรูปของตาราง เพื่อจัดเก็บในฐานข้อมูลโดยจะมีรายละเอียดในภาคผนวก ง. จากนั้นเราก็จะแมปเป็นตาราง ซึ่งก็จะได้ตารางทั้งหมดตามรายการต่อไปนี้

ตาราง Employee

Emp# เลขประจำตัวพนักงาน

EmployeeName ชื่อพนักงาน

Gender เพศของพนักงาน

Degree ระดับการศึกษาของพนักงาน

Phone หมายเลขโทรศัพท์ของพนักงาน

Salary เงินเดือนของพนักงาน

Department แผนกของพนักงาน

Branch ชื่อสาขา

Bonus โบนัสพนักงาน

Holiday วันพักร้อนของพนักงาน

ตาราง ManagerCar

Emp# เลขประจำตัวของพนักงาน

Reg# เลขทะเบียนรถ

ตาราง Manager

Emp# เลขประจำตัวของพนักงาน

Licence# หมายเลขทะเบียนใบขับขี่

ตาราง MarriedEmployee

Emp# เลขประจำตัวของพนักงาน

Year ปีที่แต่งงาน

ตาราง Department

DeptName ชื่อแผนก

HeadofDept ชื่อหัวหน้าแผนก

PersonalCount จำนวนพนักงานในแผนก

Budget งบประมาณของแผนกประจำปี

ตาราง Branch

BranchName ชื่อสาขา

Locate เมืองที่สาขานั้นอยู่

ตาราง Sell

BranchName ชื่อสาขา

DriveKind ชนิดไดรฟ์ที่ผลิต

Year ปีที่ผลิต

Qty จำนวนที่ผลิต

และนอกจากการกำหนดแบบจำลองขข้อมูลขึ้นมาแล้ว เรายังจะต้องกำหนดกฎขึ้นมาทดสอบความสามารถของการประมวลผลคำถามอีกด้วย โดยจะกำหนดขึ้นมาทั้งกฎชนิดเรียกตัวเองและแบบไม่เรียกตัวเอง โดยจะกำหนดขึ้นมาอย่างละ 1 กฎ โดยกฎแบบไม่เรียกตัวเองจะเป็นกฎที่จะหาว่าพนักงานแต่ละคนอาศัยอยู่ในเมืองใด โดยจะอนุมานได้จากชนิดความจริง F14 ซึ่งเป็นชนิดความจริงที่เก็บข้อมูลว่า

พนักงานคนใดทำงานที่สาขาใด และชนิดความจริง F15 ซึ่งเป็นชนิดความจริงที่เก็บข้อมูลว่าสาขาใดตั้งอยู่ในเมืองอะไร โดยสามารถเขียนเป็นกฎได้ดังนี้

live(X,Z) -> F14(X,Y),F15(Y,Z)

สำหรับกฎแบบเรียกตัวเองนั้นจะเป็นกฎของผู้บังคับบัญชา โดยจะสมมติว่าในบริษัทนี้มีการแบ่งแผนกเป็นลำดับชั้นได้ เช่น แผนกบัญชีอาจจะเป็นแผนกย่อยของแผนกการเงินได้ ดังนั้นพนักงานที่อยู่ในแผนกบัญชีก็จะต้องเป็นลูกน้องของหัวหน้าแผนกการเงินด้วย ดังนั้นสำหรับคำถามที่ว่าใครบ้างที่เป็นผู้บังคับบัญชาของพนักงาน 'A' จึงเป็นคำถามตามแบบเรียกตัวเอง ซึ่งจะต้องประมวลผลตามกฎแบบเรียกตัวเอง แต่ก่อนอื่นจะต้องสร้างความสัมพันธ์ของผู้บังคับบัญชาระดับเดียวขึ้นมาก่อน โดยจะสร้างจากชนิดความจริง F3 และ F4 โดยสามารถเขียนความสัมพันธ์และกฎได้ดังนี้

```
CREATE VIEW DIRECT_RULER (RULER, BOY)
AS SELECT T1.EMP#, T2.EMP#
FROM F3 T1, F4 T2, F19 T3
WHERE T1.EMP# IN (SELECT EMP# FROM F3)
AND T1.DEPT_NAME = T2.DEPT_NAME;

ruler(X,Y) -> direct_ruler(X,Y);
ruler(X,Y) -> ruler(X,Z), direct_ruler(Z,Y);
```

**11.2 การทดสอบส่วนนิยามข้อมูล**

จากตารางที่เราได้สร้างไว้และจากแบบจำลองข้อมูลที่เราได้นำไปจัดเก็บในเมต้าสกีมาเป็นที่เรียบร้อยแล้ว เราก็จะทดสอบการทำงานส่วนแรกของโปรแกรม ซึ่งก็คือ ส่วนนิยามข้อมูล โดยการเรียกโปรแกรม DDS (Deductive Database System) และกด F5 เพื่อเรียกโปรแกรมในส่วนนิยามข้อมูลมาทำงาน ซึ่งโปรแกรมในส่วนนิยามข้อมูลนี้จะทำงาน 3 ส่วนด้วยกัน ส่วนแรกก็คือการสร้างวิวให้กับชนิดความจริงแต่ละชนิดความจริง สำหรับส่วนที่สองจะเป็นการเรียกโปรแกรมตรวจสอบความถูกต้องของข้อมูลส่วนเตรียมการมาทำงาน และสุดท้ายก็จะเรียกโปรแกรมประมวลผลคำถามส่วนเตรียมการมาทำงาน และผลจากการทำงานนี้ก็จะได้ข้อมูลกลับมา 3 ส่วน คือ วิวของชนิดความจริง คิวรีที่ใช้สำหรับตรวจสอบคอนสแตนต์แต่ละคอนสแตนต์ และข้อมูลของกฎชนิดต่าง ๆ ที่เก็บไว้ในตาราง และหากเป็นกฎชนิดไม่รีเคอร์ซีฟ ก็จะสร้างวิวของกฎนั้นให้ด้วย และผลการทำงานก็มีดังต่อไปนี้

11.2.1 วิวของชนิดความจริง

จากแบบจำลองข้อมูลในภาพที่ 11.1 เมื่อนำมาสร้างเป็นวิวที่ใช้แทนชนิดความจริง จะได้วิวทั้งหมด 17 วิว ดังต่อไปนี้

วิว F1 (DeptPersonCount)

```
CREATE VIEW F1(DEPT_NAME, PERS_NR)
AS SELECT DEPT_NAME, PERSONAL_COUNT
FROM DEPARTMENT;
```

วิว F2 (DeptBudget)

```
CREATE VIEW F2 (DEPT_NAME, BUDGET$)
AS SELECT DEPT_NAME, BUDGET
FROM DEPARTMENT;
```

วิว F3 (DeptHead)

```
CREATE VIEW F3 (EMP#, DEPT_NAME)
AS SELECT EMP#, DEPT_NAME
FROM DEPARTMENT;
```

วิว F4 (EmpDepartment)

```
CREATE VIEW F4 (EMP#, DEPT_NAME)
AS SELECT EMP#, DEPARTMENT
FROM EMPLOYEE;
```

วิว F5 (EmpHoliday)

```
CREATE VIEW F5 (EMP#, H_DATE)
AS SELECT EMP#, HOLIDAY
FROM EMPLOYEE;
```

วิว F6 (EmpSalary)

```
CREATE VIEW F6 (EMP#, SALARY$)
AS SELECT EMP#,SALARY
FROM EMPLOYEE;
```

วิว F7 (EmpBobus)

```
CREATE VIEW F6 (EMP#, BONUS$)
AS SELECT EMP#, BONUS
FROM EMPLOYEE;
```

วิว F8 (EmpPhone)

```
CREATE VIEW F8 (EMP#,PHONE#)
AS SELECT EMP#,PHONE
FROM EMPLOYEE;
```

วิว F9 (EmpGender)

```
CREATE VIEW F9 (EMP#, G_CODE)
AS SELECT EMP#, GENDER
FROM EMPLOYEE;
```

วิว F10 (EmpDegree)

```
CREATE VIEW F10 (EMP#, D_CODE)
AS SELECT EMP#,DEGREE
FROM EMPLOYEE;
```

วิว F11 (ManagerCar)

```
CREATE VIEW F11 (EMP#, REG#)
AS SELECT EMP#, REG#
FROM MANAGE_CAR;
```

วิว F12 (Manager#)

```
CREATE VIEW F12 (EMP#, LICENCE#)
AS SELECT EMP#, LICENCE#
FROM MANAGER;
```

วิว F13 (MarriageEmployee)

```
CREATE VIEW F13 (EMP#, Y_AD)
AS SELECT EMP#, YEAR
FROM MARRIED_EMPLOYEE;
```

วิว F14 (EmpBranch)

```
CREATE VIEW F14 (EMP#, B_NAME)
AS SELECT EMP#,BRANCH
FROM EMPLOYEE;
```

วิว F15 (Branch)

```
CREATE VIEW F15 (B_NAME, CITY)
AS SELECT BRANCH_NAME,LOCATE
FROM BRANCH
```

วิว F16 (SellInfo)

```
CREATE VIEW F16 (B_NAME, DR_CODE, Y_AD)
AS SELECT BRANCH, DRIVE_KIND, YEAR
FROM SELL;
```

วิว F17 (SellQty)

```
CREATE VIEW F17 (B_NAME, DR_CODE, Y_AD, QTY_NR)
AS SELECT BRANCH, DRIVE_KIND, YEAR, QTY
FROM SELL;
```

11.2.2 คิวรีที่ใช้ตรวจสอบคอนสแตนต์

หลังจากที่ได้สร้างวิวเสร็จเรียบร้อยแล้ว งานในขั้นต่อไปที่โปรแกรม DDS จะทำก็คือ การเรียกส่วนเตรียมการของส่วนตรวจสอบคอนสแตนต์มาทำงาน ซึ่งจะได้ผลลัพธ์เป็นคิวรีที่ใช้สำหรับตรวจสอบคอนสแตนต์ จากแบบจำลองข้อมูลในรูปที่ 11-1 ที่เราใช้ในการทดสอบ ซึ่งมีคอนสแตนต์อยู่ทั้งหมด 28 คอนสแตนต์โดยแบ่งเป็นคอนสแตนต์แบบ Uniqueness จำนวน 18 คอนสแตนต์ คอนสแตนต์แบบ Mendatory จำนวน 5 คอนสแตนต์ และคอนสแตนต์ Range, Membership, Occurrence Frequency, Equality, Exclusion, Subset อย่างละ 1 คอนสแตนต์ ดังนั้นคิวรีที่ได้จากขั้นตอนนี้จะค่อนข้างมากและแต่ละคิวรีก็จะมีความยาวพอสมควร แต่ละคิวรีจะมีรายละเอียดดังต่อไปนี้

คิวรี่สำหรับคอนสแตนต์ C1

```
SELECT COUNT (DEPT_NAME)
FROM F1;
SELECT COUNT (DISTINCT DEPT_NAME)
FROM F1;
```

คิวรี่สำหรับคอนสแตนต์ C2

```
SELECT COUNT (DEPT_NAME)
FROM F2;
SELECT COUNT (DISTINCT DEPT_NAME)
FROM F2;
```

คิวรี่สำหรับคอนสแตนต์ C3

```
SELECT COUNT (DEPT_NAME)
FROM F3;
SELECT COUNT (DISTINCT DEPT_NAME)
FROM F3;
```

คิวรี่สำหรับคอนสแตนต์ C4

```
SELECT COUNT (EMP#)
FROM F5;
SELECT COUNT (DISTINCT EMP#)
FROM F5;
```

คิวรี่สำหรับคอนสแตนต์ C5

```
SELECT COUNT (EMP#)
FROM F6;
SELECT COUNT (DISTINCT EMP#)
FROM F6;
```

คิวรี่สำหรับคอนสแตนต์ C6

```
SELECT COUNT (EMP#)
FROM F7;
SELECT COUNT (DISTINCT EMP#)
FROM F7;
```

คิวรี่สำหรับคอนสแตนต์ C7

```
SELECT COUNT (EMP#)
FROM F8;
SELECT COUNT (DISTINCT EMP#)
FROM F8;
```

คิวรี่สำหรับคอนสแตนต์ C8

```
SELECT COUNT (EMP#)
FROM F9;
SELECT COUNT (DISTINCT EMP#)
FROM F9;
```

คิวรี่สำหรับคอนสแตนต์ C9

```
SELECT COUNT (EMP#)
FROM F10;
SELECT COUNT (DISTINCT EMP#)
FROM F10;
```

คิวรี่สำหรับคอนสแตนต์ C11

```
SELECT COUNT (EMP#,REG#)
FROM F11;
SELECT COUNT (DISTINCT EMP#,REG#)
FROM F11;
```

คิวรี่สำหรับคอนสแตนต์ C12

```
SELECT COUNT (EMP#)
FROM F12;
SELECT COUNT (DISTINCT EMP#)
FROM F12;
```

คิวรี่สำหรับคอนสแตนต์ C13

```
SELECT COUNT (EMP#)
FROM F13;
SELECT COUNT (DISTINCT EMP#)
FROM F13;
```

คิวรี่สำหรับคอนสแตนต์ C14

```
SELECT COUNT (EMP#)
FROM F14;
SELECT COUNT (DISTINCT EMP#)
FROM F14;
```

คิวรี่สำหรับคอนสแตนต์ C15

```
SELECT COUNT (EMP#)
FROM F15;
SELECT COUNT (DISTINCT EMP#)
FROM F15;
```

คิวรี่สำหรับคอนสแตนต์ C16

```
SELECT COUNT (B_NAME, DR_CODE, Y_AD)
FROM F16;
SELECT COUNT (DISTINCT B_NAME, DR_CODE, Y_AD)
FROM F16;
```

คิวรี่สำหรับคอนสแตนต์ C17

```
SELECT COUNT (B_NAME, DR_CODE, Y_AD)
FROM F18;
SELECT COUNT (DISTINCT B_NAME, DR_CODE, Y_AD)
FROM F18;
```

คิวรี่สำหรับคอนสแตนต์ C18

```
SELECT COUNT (EMP#)
FROM F4;
SELECT COUNT (DISTINCT B_NAME, DR_CODE, Y_AD)
FROM F4;
```

คิวรี่สำหรับคอนสแตนต์ C20

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE EMP# IS NOT NULL
AND GENDER IS NULL;
```

คิวรี่สำหรับคอนสแตนต์ C21

```
SELECT COUNT(*)
FROM EMPLOYEE
WHERE EMP# IS NOT NULL
AND BRANCH IS NULL;
```

คิวรี่สำหรับคอนสแตนต์ C22

```
SELECT COUNT(*)
FROM MANAGE_CAR
WHERE EMP# IS NOT NULL
AND REG# IS NULL;
```

คิวรี่สำหรับคอนสแตนต์ C23

```
SELECT COUNT(*)
FROM MARRIED_EMP
WHERE EMP# IS NOT NULL
AND YEAR IS NULL;
```

คิวรี่สำหรับคอนสแตนต์ C24

```
SELECT COUNT(*)
FROM EMPLOYEE T1
WHERE EXISTS (SELECT * FROM EMPLOYEE T2
                WHERE T1.EMP# = T2.EMP#);
```

คิวรี่สำหรับคอนสแตนต์ C25

```
SELECT COUNT(*)
FROM DEPARTMENT T1
WHERE NOT EXISTS (SELECT * FROM EMPLOYEE T2
                    WHERE T1.HEAD_OF_DEPT = T2.EMP#);
```

คิวรี่สำหรับคอนสแตนต์ C26

```
SELECT COUNT(*)
FROM MANAGE_CAR T1
WHERE NOT EXISTS (SELECT * FROM MANAGER T2
                    WHERE T1.EMP# = T2.EMP#);
SELECT COUNT(*)
FROM MANAGER T1
WHERE NOT EXISTS (SELECT * FROM MANAGE_CAR T2
                    WHERE T1.EMP# = T2.EMP#);
```

คิวรี่สำหรับคอนสแตนต์ C27

```
SELECT COUNT(*)
FROM SELL
WHERE DRIVE_KIND NOT IN ('HD','FD');
```

**คิวรี่สำหรับคอนสแตนต์ C28**

```
SELECT COUNT(*)
FROM SELL
WHERE QTY > 200 OR QTY < 1;
```

**คิวรี่สำหรับคอนสแตนต์ C29**

```
SELECT COUNT(*)
FROM SELL
GROUP BY DRIVE_KIND;
```

11.2.3 ผลลัพธ์ของการประมวลผลคำถามเบื้องต้น

จากที่ได้กล่าวมาแล้วในบทที่ 10 ว่าการประมวลคำถามเบื้องต้น จะนำกฎที่รับเข้ามาจะนำไปแยกเก็บในตาราง 3 ตาราง คือ ตาราง VIEW_COLUMN, PREDICATE และ RULE เพื่อให้ส่วนประมวลผลอ่านมาทำงานได้โดยภายหลัง โดยจากกฎทั้งสองข้างต้นจะได้ผลลัพธ์ที่สามารถจัดเก็บลงในตารางได้ดังนี้

ตาราง VIEW _COLUMN

| V_NAME | C_NAME | C_TYPE | C_ORDER | C_WIDTH | P_ID | D_ID |
|---|---|---|---|---|---|---|
| ruler | RULER | CHAR | 1 | 20 | N | R |
| ruler | BOY | CHAR | 2 | 20 | N | R |
| direct_ruler | RULER | CHAR | 1 | 20 | N | N |
| direct_ruler | BOY | CHAR | 2 | 20 | N | N |
| live | EMP# | CHAR | 1 | 10 | N | N |
| live | CITY | CHAR | 2 | 20 | N | N |

ตาราง RULE

| HEAD | BODY | HEAD_NUMBER | BODY_NUMBER |
|---|---|---|---|
| ruler | direct_ruler | 1 | 1 |
| ruler | direct_ruler | 2 | 1 |
| ruler | direct_ruler | 2 | 2 |

ตาราง PREDICATE

| ARGUMENT | ARG_ORDER | HEAD_NUMBER | BODY_NUMBER |
|---|---|---|---|
| X | 1 | 1 | 0 |
| Y | 2 | 1 | 0 |
| X | 1 | 1 | 1 |
| Y | 2 | 1 | 1 |
| X | 1 | 2 | 0 |
| Z | 2 | 2 | 0 |
| X | 1 | 2 | 1 |
| Y | 2 | 2 | 1 |
| Y | 1 | 2 | 2 |
| Z | 2 | 2 | 2 |

11.2.4 วิวที่ได้จากกฎชนิดไม่เรียกตัวเอง

จากที่ได้กล่าวไปแล้วในบทก่อนหน้านี้ข้างต้นว่ากฎชนิดที่ไม่เรียกตัวเองนั้น สามารถแทนด้วยการสร้างวิวได้ และจากตัวอย่างข้อมูลซึ่งมีกฎชนิดไม่เรียกตัวเองอยู่ 1 กฎ ซึ่งเมื่อผ่านการทำงานในส่วนการประมวลผลคำถามเบื้องต้นแล้ว ก็จะได้วิวขึ้นมา 1 วิว โดยวิวนี้จะแทนความสัมพันธ์ของพนักงานกับเมืองที่พนักงานนั้นอยู่ โดยอนุมานมาจากความสัมพันธ์ของพนักงานกับสาขา และความสัมพันธ์ของสาขากับเมือง โดยสามารถเขียนเป็นคำสั่ง SQL ในการสร้างวิวได้ดังนี้

```
CREATE VIEW LIVE(EMP#, CITY)
AS SELECT EMP#, CITY
FROM F14, F15
WHERE F14.B_NAME = F15.B_NAME;
```

**11.3 การทดสอบส่วนจัดการข้อมูล**

เราจะทดสอบส่วนจัดการข้อมูลโดยการสร้างชุดทดสอบขึ้นมาดังต่อไปนี้ ซึ่งประกอบด้วยคำสั่ง Assert และ Remove ชนิดความจริงต่าง ๆ โดยตั้งใจให้การทดสอบผิดพลาดบ้างถูกต้องบ้างและดูผลการทำงาน ซึ่งจากชุดทดสอบที่เราสร้างขึ้น จะได้ผลลัพธ์การทำงานดังนี้

```
ASSERT F9 ('001','M');
ASSERT RF1 ('001','JOHN');
ASSERT F6 ('001','326-9068');
ASSERT F4 ('001','RESEARCH');
ผลลัพธ์ : Rejected : C21 Violated
```

เพราะ C21 เป็นคอนสแตนต์แบบ Mendatory ดังนั้นการบันทึกข้อมูลของพนักงานแต่ละคนจะต้องมีการบันทึก ชื่อแผนก เพศ และชื่อสาขาด้วยเสมอ

ASSERT F9 ('001','M');

ASSERT RF1 ('001','JOHN');

ASSERT F6 ('001','326-9068');

ASSERT F4 ('001','RESEARCH');

ASSERT F14 ('001','BANGKOK');

ผลลัพธ์ : Accepted

เพราะมีการใส่ข้อมูลที่จำเป็นครบทุกข้อมูลแล้ว

ASSERT F11 ('001','CA-2475');

ผลลัพธ์ : Rejected : C26 Violated

เพราะในการบันทึกเลขทะเบียนรถของพนักงานระดับผู้จัดการนั้น จะต้องมีการบันทึกเลขใบขับขี่ควบคู่ด้วยเสมอ

ASSERT F11 ('001','CA-2475');

ASSERT F13 ('001','123-4567');

ผลลัพธ์ : Accepted

เพราะในการบันทึกเลขทะเบียนรถของพนักงานระดับผู้จัดการนั้น จะต้องมีการบันทึกชื่อคนขับควบคู่ด้วยเสมอ

ASSERT F5('001','10-5-94');

ASSERT F7('001','2500');

ผลลัพธ์ : Rejected : C24 Violated

เพราะการบันทึกวันพักร้อนและโบนัสจะบันทึกพร้อมกันไม่ได้ เพราะถ้าได้วันพักร้อนก็ย่อมจะไม่มีโบนัส และถ้าได้โบนัสก็จะต้องไม่มีวันพักร้อน

ASSERT F6('001','745-2111');

ผลลัพธ์ : Rejected : C7 Violated

เพราะพนักงานแต่ละคนจะอนุญาตให้มีการบันทึกเบอร์โทรศัพท์ได้เพียงครั้งเดียวเท่านั้น แต่จะอนุญาตให้พนักงานหลายคนมีเบอร์โทรศัพท์เดียวกันได้

ASSERT F3 ('002','RESEARCH');

ผลลัพธ์ : Rejected : Violated C25

เพราะการบันทึกชื่อของหัวหน้าแผนกนั้น จะต้องมีการบันทึกชื่อนั้นลงในพนักงานของแผนกก่อน เพราะถ้าไม่เป็นพนักงานของแผนกนั้นแล้ว จะเป็นหัวหน้าแผนกได้อย่างไร

ASSERT F16('BANGNA', 'HD', '1992');
ASSERT F17('BANGNA', 'HD', '1992', '300');

ผลลัพธ์ : Rejected : C28 Violated
C29 Violated

เพราะคอนสแตนต์ C28 บังคับไว้ว่าปริมาณการขายในแต่ละปีจะไม่เกิน 200 และจะต้องไม่ต่ำกว่า 1 นั่นหมายความว่าจะต้องขายได้ทุกปี และคอนสแตนต์ก็บังคับไว้อีกว่าจะต้องมีการบันทึกไดรฟ์ทั้ง 2 ชนิดด้วยเสมอ

ASSERT F16('BANGNA', 'HD', '1992');
ASSERT F17('BANGNA', 'HD', '1992', '100');
ASSERT F17('BANGNA', 'FD', '1992', '200');

ผลลัพธ์ : Accepted
เพราะมีการบันทึกการขายครบทั้ง 2 อย่างในแต่ละปี

**11.4 การทดสอบส่วนประมวลผลคำถาม**

เราจะทดสอบส่วนประมวลผลคำถามเราจะสร้างคำถามที่มีทั้งคำถามแบบชนิดความจริง คำถามแบบเรียกตัวเอง และคำถามแบบไม่เรียกตัวเอง แต่ก่อนอื่นจะต้องบันทึกข้อมูลพื้นฐานลงไปก่อน เพราะหากไม่มีข้อมูลอยู่จริงแล้ว การถามคำถามก็ย่อมจะไม่ได้คำตอบใด ๆ ซึ่งข้อมูลพื้นฐานก็มีดังต่อไปนี้ (สมมติว่าข้อมูลพื้นฐานอื่น ๆ บันทึกไว้เรียบร้อยแล้วเช่นกัน)

ASSERT F14 ('001', 'BANKNA');
ASSERT F14 ('002', 'PATTAYA');
ASSERT F15 ('BANGNA', 'BANGKOK');
ASSERT F15 ('PATTAYA', 'CHONBURI');
ASSERT F4 ('001', 'ACCOUNTING');
ASSERT F4 ('002', 'ACCOUNTING');
ASSERT F4 ('001', 'FINANCE');
ASSERT F4 ('003', 'FINANCE');
ASSERT F4 ('003', 'ADMIN');

ASSERT F4 ('004', 'ADMIN');

ASSERT F3 ('001', 'ACCOUNTING');

ASSERT F3 ('003', 'FINANCE');

ASSERT F3 ('004', 'ADMIN');

ซึ่งจากข้อมูลข้างต้นจะทำให้ได้ข้อมูลในวิว DIRECT_RULER จะได้เป็น

direct_ruler ('001', '002');

direct_ruler ('003', '001');

direct_ruler ('004', '003');

และเมื่อถามคำถามโดยถามว่าใครบ้างที่เป็นผู้บังคับบัญชาของพนักงานหมายเลข 002 ก็จะมีการสร้าง Augmented Loop Residue และ Augmented Exit Expression ได้เป็น

ruler(X,Y), ~direct_ruler(X,Z), ~ruler(Z,Y)

ruler(X,Y), ~direct_ruler(X,Z), ~direct_ruler(Z,Y)

ซึ่งในระหว่างขั้นตอนการค้นหาคำตอบจะสร้างคิวรี่ภาษา SQL ออกมาได้เป็น

```
INSERT INTO RULER
SELECT RULER, BOY
FROM DIRECT_RULER
WHERE BOY = '002';                    ซึ่งจะได้คำตอบเป็น 001

INSERT INTO RULER
SELECT T1.RULER, T2.BOY
FROM DIRECT_RULER T1, DIRECT_RULER T2
WHERE T1.BOY = T2.RULER
AND T2.BOY = '002';                   ซึ่งจะได้คำตอบเป็น 003

INSERT INTO RULER
SELECT T1.RULER, T3.BOY
FROM DIRECT_RULER T1, DIRECT_RULER T2, DIRECT_RULER T3
WHERE T1.BOY = T2.RULER AND T2.BOY = T3.RULER
AND T3.BOY = '002';                   ซึ่งจะได้คำตอบเป็น 004
```

```
INSERT INTO RULER
SELECT T1.RULER, T4.BOY
FROM DIRECT_RULER T1, DIRECT_RULER T2,
        DIRECT_RULER T3, DIRECT_RULER T4
WHERE T1.BOY = T2.RULER AND T2.BOY = T3.RULER AND T3.BOY = T4.RULER
AND T4.BOY = '002';                    ซึ่งจะได้คำตอบเป็นเซ็ตว่าง
```

ดังนั้นจะได้ผลลัพธ์ของคำถามออกมาเป็น '001', '003' และ '004'

**11.5 สรุปผลการทำงาน**

จากการทดสอบต่าง ๆ ที่ได้กำหนดขึ้น จะเห็นได้ว่าโปรแกรม DDS สามารถทำงานตามเป้าหมายได้อย่างน่าพอใจ ไม่ว่าจะเป็นส่วนนิยามข้อมูล ส่วนจัดการข้อมูล ส่วนประมวลผลคำถาม ทั้งที่เป็นคำถามชนิดเรียกตัวเองและแบบไม่เรียกตัวเอง ซึ่งการทำงานทั้งหมดก็เป็นไปตามทฤษฎีที่ได้กล่าวไว้ทั้งหมด ซึ่งหมายความว่าระบบ DDS ได้พัฒนาเสร็จสิ้นสมบูรณ์แล้ว

# บทที่ 12
# สรุปผลและแนวทางการวิจัย

## 12.1 สรุปผลงานวิจัย

ในฐานข้อมูลแบบรีเลชันแนลที่มีการใช้งานกันอย่างกว้างขวางในปัจจุบันนี้ แม้ว่าจะมีลักษณะโมเดลข้อมูลที่ง่ายต่อการทำความเข้าใจ และง่ายต่อการใช้งาน โดยมีลักษณะเป็นตารางที่มีการแบ่งเป็นแถวและคอลัมน์เท่านั้น รูปแบบโมเดลที่ง่ายนี้เป็นเสมือนกับดาบสองคม คือ แม้ว่าจะเปิดโอกาสให้มีการนำไปประยุกต์ใช้แบบไม่มีขีดจำกัด แต่ก็เปิดโอกาสให้ผิดพลาดได้ง่ายเช่นกัน การเก็บข้อมูลในลักษณะตารางนั้นสามารถเก็บได้แม้ว่าจะไม่มีความสัมพันธ์ของข้อมูลใด ๆ เลย หรือจะกล่าวว่าสามารถเก็บได้ทุกรูปแบบก็คงจะไม่ผิดนัก

ด้วยเหตุดังกล่าวจึงมีความพยายามในการหาเครื่องมือที่เข้ามาช่วยให้ความถูกต้องของข้อมูลสูงขึ้น เริ่มตั้งแต่การใส่กฎเกณฑ์ที่ใช้ตรวจสอบความถูกต้องของข้อมูลเข้าไปในระบบจัดการฐานข้อมูล เช่น การสร้างไพรมารีคีย์และฟอเรนคีย์ และการสร้างวิธีการออกแบบฐานข้อมูลด้วยวิธีการต่าง ๆ เช่น นอร์มอลไลเซชั่น และวิธีการ Entity-Relationship ซึ่งได้พัฒนาขึ้นมาเป็นซอฟต์แวร์ที่ช่วยในการออกแบบฐานข้อมูล ที่มีจุดมุ่งหมายที่จะช่วยให้ข้อมูลมีความถูกต้องมากที่สุด แต่นั่นก็นับเป็นการแก้ปัญหาที่ปลายเหตุเท่านั้น

และนอกจากโมเดลข้อมูลแบบรีเลชันแนลจะมีความเสี่ยงต่อความผิดพลาดมากแล้ว จุดด้อยอีกประการหนึ่งของโมเดลข้อมูลรีเลชันแนล คือ ความสามารถในการแสดงความสัมพันธ์ของข้อมูล เพราะในโมเดลรีเลชันแนลที่มีรูปแบบเป็นเพียงตารางนั้น ไม่อาจแสดงความสัมพันธ์ของข้อมูลภายในตารางได้อย่างแจ่มชัดมากนัก และยิ่งไม่อาจแสดงความสัมพันธ์ในระหว่างตารางได้เลย แม้ว่าจะมีความพยายามในการแทนข้อมูลในรูปแบบอื่น ๆ ในขั้นตอนการออกแบบ เช่น การแทนข้อมูลด้วยโมเดล ER หรือ NIAM ก่อนที่จะแมปเป็นโมเดลรีเลชันแนล หรือแม้จะมีการสร้างระบบที่สามารถโยงเส้นความสัมพันธ์ระหว่างตารางในซอฟต์แวร์ใหม่ ๆ ก็ยังไม่สามารถแก้ปัญหาได้หมด

แนวทางหนึ่งที่นับเป็นการแก้ปัญหาที่ต้นเหตุก็คือ การหาโมเดลข้อมูลใหม่ที่สามารถแสดงความสัมพันธ์ได้อย่างแจ่มชัด และสามารถเข้าใจได้ง่าย ซึ่งโมเดลข้อมูลไนแอมนับเป็นโมเดลข้อมูลหนึ่งที่มีคุณสมบัติดังกล่าว โดยมีพื้นฐานอยู่บนชนิดความจริง ซึ่งเป็นรูปแบบข้อมูลที่เล็กที่สุดที่แทนความหมายได้ นอกจากนั้นแบบจำลองข้อมูลไนแอมยังมีกลไกในการตรวจสอบความถูกต้องของข้อมูล จึงทำให้สามารถควบคุมข้อมูลได้สูงขึ้น และไนแอมยังสามารถแปลงไปเป็นโมเดลรีเลชันแนลที่มีความถูกต้องในระดับ 5 NF ได้อีกด้วย

วัตถุประสงค์ในการทำงานวิจัยนี้ ก็เพื่อจะศึกษาหาความเป็นไปได้ ในการสร้างฐานข้อมูลที่มีโมเดลข้อมูลแบบอื่นเป็น Conceptual Schema โดยได้ทดสอบสร้างระบบฐานข้อมูลอนุมาน โดยมีการนำเอา

แบบจำลองข้อมูลไนแอม มาใช้เป็นแบบจำลองข้อมูลในระดับแนวคิด (Conceptual Schema) ของฐานข้อมูล และยังได้เพิ่มความสามารถทางด้านการอนุมานกฎเข้าไปในระบบฐานข้อมูลด้วย

ซอฟต์แวร์ฐานข้อมูลที่สร้างขึ้นนี้จะวิ่งอยู่บนระบบจัดการฐานข้อมูล ORACLE เพื่ออาศัยความสามารถของ ORACLE ในการจัดการข้อมูลระดับล่าง ทั้งนี้ก็โดยอาศัยคุณสมบัติของ NIAM ในการแปลงไปเป็นโมเดลรีเลชันแนลได้นั้นเอง ซอฟต์แวร์ที่สร้างขึ้นนี้สามารถทำงานได้ดีพอสมควร โดยมีความสามารถของฐานข้อมูลโดยครบถ้วน สามารถนำข้อมูลในรูปของชนิดความจริงเข้าไปเก็บในฐานข้อมูล และลบข้อมูลในรูปของชนิดความจริงแบบฐานข้อมูลออกจากฐานข้อมูลได้

นอกจากนั้นยังสามารถเรียกค้นข้อมูลแบบง่าย ๆ ด้วยรูปแบบการค้นที่คล้ายกับภาษาโปรลอก โดยสามารถเรียกค้นข้อมูลได้ทั้งแบบชนิดความจริง ทั้งแบบกฎชนิดไม่เรียกตัวเอง และกฎชนิดเรียกตัวเองได้ นอกจากนั้นยังมีกลไกในการรักษาความถูกต้องของข้อมูลที่สูงระดับหนึ่งอีกด้วย นับได้ว่าโปรแกรมที่สร้างขึ้นนี้ ทำงานได้ตามเป้าหมายที่ตั้งไว้ได้อย่างน่าพอใจ

อย่างไรก็ตาม ซอฟต์แวร์ที่สร้างขึ้นนี้ยังมีข้อบกพร่องในแง่ของความเร็วในการประมวลผล ทั้งนี้เนื่องจากสร้างขึ้นในลักษณะที่ซ้อนทับระบบฐานข้อมูลแบบรีเลชันแนล ซึ่งจะต้องเสียเวลาส่วนหนึ่งในการแปลงความสัมพันธ์ของข้อมูลในลักษณะไนแอมไปเป็นความสัมพันธ์ข้อมูลในแบบรีเลชันแนล และการใช้ไนแอมเป็นแบบแทนข้อมูล ก็ยังทำให้การแมปเกิดเป็นตารางจำนวนมาก ซึ่งก็ยิ่งทำให้การประมวลผลช้ายิ่งขึ้นไปอีก แต่หากมีการนำเอาไปใช้งานจริงที่มีระบบจัดการกับข้อมูลระดับล่างที่มีประสิทธิภาพมากเพียงพอแล้ว ก็คาดว่าปัญหานี้คงจะลงลงไปได้มาก

นอกจากนั้นงานวิจัยนี้เพียงแต่มุ่งเน้นในเรื่องของการแทนข้อมูลในลักษณะของไนแอมเท่านั้น ไม่ได้มุ่งไปที่ภาษาที่ใช้ในการเรียกค้นข้อมูล ทำให้มีข้อจำกัดในการใช้งานค่อนข้างมาก ดังนั้นหากจะมีการนำไปใช้งานจริง ก็ควรจะต้องพัฒนาภาษาเรียกค้นที่มีประสิทธิภาพมากกว่านี้ ในฐานข้อมูลรีเลชันแนลนั้นมีภาษาเรียกค้น SQL ที่มีประสิทธิภาพสูงโดยมีคณิตศาสตร์สาขา Relational Algrebra เป็นพื้นฐาน แต่เนื่องจากไนแอมนั้นมีรากฐานอยู่บน Predicate Logic ดังนั้นภาษาเรียกค้นก็น่าจะมีลักษณะที่คล้ายกับภาษาโปรลอกมากกว่า

สรุปได้ว่างานวิจัยที่ได้พัฒนาขึ้นนี้ สามารถจัดเก็บข้อมูลในลักษณะของแบบจำลองข้อมูลไนแอมได้อย่างสมบูรณ์ สามารถนำข้อมูลเข้าและลบข้อมูลจากฐานข้อมูลได้ตามแนวคิดของไนแอม สามารถตรวจสอบความถูกต้องจามรูปแบบของไนแอม และสามารถอนุมานกฎได้ทั้งชนิดเรียกตัวเอง และไม่เรียกตัวเอง ได้อย่างสมบูรณ์

## 12.2 แนวทางการพัฒนาต่อ

โครงงานวิจัยนี้แม้ว่าจะมีความสมบูรณ์ในตัวเอง แต่เมื่อเทียบกับสถาปัตยกรรมของระบบที่ได้ออกแบบไว้ในบทที่ 5 ก็ยังนับว่ายังพัฒนาไม่ครบถ้วนทั้งระบบ สำหรับแนวทางการพัฒนาต่อนั้นมีอยู่หลายแนวทางด้วยกัน ได้แก่ การพัฒนาระบบฐานข้อมูลอนุมานนี้ขึ้นมาใหม่ทั้งระบบ โดยให้มีการจัดการข้อมูลภายในเป็นของตัวเอง โดยไม่ต้องพึ่งพา ORACLE ซึ่งจะทำให้ประสิทธิภาพโดยรวมของระบบดีขึ้นอย่างมาก แต่ก็นับว่าเป็นงานที่หนักไม่น้อยเลยทีเดียว เพราะเป็นการพัฒนาระบบฐานข้อมูลขึ้นมาใหม่

ทั้งระบบ ซึ่งจะต้องมีระบบรักษาความปลอดภัยต่าง ๆ ให้ได้เท่ากับซอฟต์แวร์ฐานข้อมูลแบบรีเลชันแนลมิฉะนั้นระบบจะไม่มีเสถียรภาพมากพอที่จะทำงานได้อย่างปลอดภัย นอกจากนั้นยังจะต้องออกแบบรูปแบบการจัดเก็บข้อมูลระดับล่างที่มีประสิทธิภาพ สามารถรองรับข้อมูลในลักษณะชนิดความจริงได้เป็นอย่างดีอีกด้วย

แนวทางการพัฒนาต่อแนวทางที่สอง คือ การพัฒนาภาษาเรียกค้นสำหรับฐานข้อมูลที่ใช้ในแอมเป็นแบบจำลองข้อมูลระดับแนวคิดโดยเฉพาะ โดยอาจจะมีรูปแบบที่คล้ายกับภาษาโปรลอกก็ได้ แต่ที่สำคัญภาษาที่ใช้จะต้องมีความเป็นไดนามิกเพียงพอ เพราะในโครงงานนี้มีการสร้างระบบภาษาที่มีขีดจำกัดมาก ดังจะเห็นได้ว่าการจะสร้างคิวรีอะไรขึ้นมาสักอย่าง จะต้องมีการเตรียมการอย่างถาวร คือ ต้องมีการสร้างวิวเอาไว้รองรับ มีการเตรียมข้อมูลเบื้องต้นมากมาย ต่างกับภาษาเรียกค้น SQL ที่สามารถใช้งานได้อย่างอิสระมาก นอกจากนั้นภาษาเรียกค้นที่สร้างขึ้นนี้ควรจะครอบคลุมทั้งส่วน DDL (Data Definition Language) และ DML (Data Manipulation Language) ด้วย

แนวทางการพัฒนาอีกแนวทางหนึ่ง คือ การสร้างส่วนออกแบบข้อมูลขึ้นมาเชื่อมต่อกับระบบฐานข้อมูล ซึ่งจะทำให้ผู้ใช้สามารถใช้งานไนแอมได้อย่างครบวงจร คือ เพียงแต่ออกแบบข้อมูลในแอมในลักษณะของภาพกราฟิก โปรแกรมก็จะนำข้อมูลไปจัดเก็บลงในเมต้าสกีมา พร้อมทั้งแมปเป็นตารางให้ด้วยในตัว โดยผู้ใช้ไม่จำเป็นต้องป้อนข้อมูลเข้าในเมต้าสกีมาเอง แต่ก็เป็นที่น่าสังเกตว่าการเชื่อมต่อนี้สามารถทำได้ 2 แนวทาง คือ เชื่อมต่อกับระบบปัจจุบันซึ่งจะมีการทำงานภายในเป็นแบบรีเลชันแนล หรือจะเชื่อมต่อหลังจากที่ได้มีการปรับปรุงให้สามารถจัดการข้อมูลภายในได้เอง ซึ่งการทำงานภายในจะเป็นแบบไนแอมอย่างสมบูรณ์

### 12.3 แนวทางการประยุกต์ใช้

โปรแกรมระบบฐานข้อมูลอนุมานนี้ เมื่อพัฒนาเสร็จสิ้นแล้วสามารถนำไปประยุกต์ใช้กับงานได้หลาย ๆ ด้าน แต่ที่นับว่าเหมาะสมต่อการนำไปประยุกต์ใช้มากที่สุด ก็คือ งานทางด้านปัญญาประดิษฐ์ เพราะรูปแบบข้อมูลของฐานข้อมูลนี้มีลักษณะที่ใกล้เคียงกับภาษาโปรลอก นอกจากนั้นฐานข้อมูลอนุมานนี้ยังสามารถวินิจฉัยกฎได้ จึงเหมาะอย่างยิ่งที่จะนำไปใช้เป็นฐานความรู้สำหรับระบบผู้เชี่ยวชาญ เพราะนอกจากจะสามารถเก็บความรู้ได้อย่างน้อยเทียบเท่ากับโปรแกรมโปรลอกทั่วไปแล้ว ยังมีข้อได้เปรียบที่สามารถเก็บข้อมูลในลักษณะแฟ็กซ์ได้มากอย่างไม่มีขีดจำกัดอีกด้วย

แนวทางการประยุกต์ใช้ที่สอง คือ การพัฒนาให้สามารถเชื่อมต่อกับภาษาโปรลอกได้โดยตรง โดยกำหนดว่าภาษาโปรลอกที่สร้างขึ้นนี้จะมีแฟ็กซ์และรูลเก็บได้ 2 ที่ คือ สามารถกำหนดให้เก็บในฐานข้อมูล และกำหนดให้เก็บในหน่วยความจำ ซึ่งภาษาโปรลอกที่สร้างขึ้นนี้จะสามารถนำไปใช้กับงานปัญญาประดิษฐ์ได้มากมายหลายด้านด้วยกัน โดยจะมีข้อได้เปรียบโปรแกรมโปรลอกอื่น ๆ คือ สามารถเก็บข้อมูลที่เป็นแฟ็กซ์อย่างไม่มีข้อจำกัด นอกจากนั้นยังอาจพัฒนาต่อให้มีความสามารถมากขึ้น โดยอาจจะมีลักษณะเป็นเปลือกระบบผู้เชี่ยวชาญ (Expert System Shell) ก็ยังได้

และแนวทางสุดท้ายก็คือ นำไปประยุกต์ใช้กับงานทั่ว ๆ ไป เช่นนำไปประยุกต์ใช้กับงานทางด้าน GIS (Geographical Information System) ซึ่งนอกจากจะทำหน้าที่ได้เหมือนกับฐานข้อมูลทั่วไปแล้ว ก็ยังจะช่วยให้ผู้พัฒนาสามารถประหยัดเวลาในการพัฒนาระบบได้ เพราะฐานข้อมูลสามารถช่วยประมวลผลงานในลักษณะที่เป็นกฎได้

# บรรณานุกรม


[1] Tsichritzis D.C. and Klug A. (1978) : "*The ANSI/X3/SPARC DBMS Frame Work : Report of the Study Group on Data Base Management Systems*", "Information System" ,3 1978.

[2] Griethuysen, J.J. (1982) : "*Concept and Terminology for the Conceptual Schema and the Information Base*", ISO Technical Report ISO/TC97/SC5/WG3.

[3] Gallaire. H and Minker. J (1978) : "*(Editors) Logic and Data Bases*" Plenum Press, New York. 1978.

[4] Kowalski, R.A. (1978) : Logic for Data Description, "*Logic and Data Bases*",Gallaire. H and Minker. J. Eds., Plenum Press, New York. 1978, pp. 77-103.

[5] Kowalski, R.A. (1984) : "*Logic as a Database Language*", Research Report Doc 82/85, Revised May 1984, Department of Computing, Imperial Collage of Science and Technology, University of London.

[6] Reiter,R. (1978) : "*Deductive question-answering on relational database*", Logic and Databases, Gallaire. H and Minker. J. Eds., Plenum Press, New York. 1978

[7] Reiter R. (1984) : "*Towords a logical reconstruction of relational database theory, On Conceptual Modelling*", Brodie, M.L., Mylopoulos J., Schmdt J.W. Eds, Springer-Varlag, New York

[8] Date C.J. (1986) : "*An introduction to Database System Vol. 1, 4 Edition.* ", Addison-Wesley, Massachusatts, 1986.

[9] Codd E.F. (1970) : "*A relational model of data for large shared data banks*", communication ACM 13 ,pp.377-387

[10] Nijssen G.M. (1977) : "*On the Gross Arthitecture for the Next Generation Database Management Systems*", Information Processing 77, Gilchrist B.Ed., IFIP, North-Holland Publishing Company, pp. 327-335

[11] Date C.J. (1981) : "*Referential Integrity*", Proceeding of the Seventh International Conference on Very Large Data Bases, Cannes, September 1981, pp. 2-12

[12] Verheijn G.M.A. and Van bekkum J. (1982) :"*NIAM - An Information Analysis Method*", Information Systems Design Methodologies : A Comparative Review, T.W. olle H.G. Sol and A.A. Verrijin-Stuart, Eds. North-Holland.

[13] Halpin T. (1986) : "*CS112 Introduction to Information System*", Department of Computer Science, University of Queensland.

[14] Nijssen G.M. (1983) : "*From databases towords knowladge bases*", DBMS - A Technical Comparision, State of Art Report, King P.J. Ed., Pergamon Infotech.

[15] Nijssen G.M. (1985) : "*On experience with large scale teaching and use of fact based conceptual schemas in industry and university*", in Proceeding of IFIP Conference Data Base Semantics 1, edited by Meersman R. and Steel T., North-Holland Publishing Comp.

[16] Robinson J.A. (1965) : "*A machine-oriented logic based on the resolusion principle*", J. ACM. 12,1 ,pp. 23-41

[17] Chang (1981) : "*On the Evaluation of Queries Containing Derived Relations in Relational Databases*", Advance in Data Base Theory Vol.1, Gallaire. H and Minker. J. and Nicolas J.M., Plenum Press, New York 1981, pp. 235-260.

[18] Shapiro and McKay (1980) : "*Inference with Recursive Rules*", S. Shapiro and D. McKay, Proc. 1st Annual National Conference on Artificial Intelligent, 1980

[19] Henschen and Naqvi (1984) : *"On Compiling Queries in Recursive First-Order Data Bases*", L. Henschen and S. Naqvi, JACM Vol. 31 1984, pp. 47-85

[20] Sickel S. : "*A search technique for cluase interconnectivity graphs*", IEEE Trans. Comput. C-25,8" (Aug 1976), pp 823-834

[22] G.M. McGrath : "*The Transition to Fifth Generation Technology : Conceptual Schema Implementation*",The Australian Computer Journal Vol. 19,No. 1 Fabruary 1987

[23] John Malpas : "*Prolog : A relational Language And its Application*" ,Prentice-Hall International Editions

[24] W.F. Clocksin, C.S. Mellish, "*Programming in Prolog*", Springer-Verlag Berlin Heidelberg New York Tokyo,1984

[25] Suphamit Chittayasothorn : "*A Knowlegde Base System Based On NIAM Conceptual Schema Model*" University of Queenslands, Australia

[26] Herve Gallaire, Jack Minker, Jean-Marie Nicolas : "*Logic and Database : A Deductive Approach*", Computing Survays, Vol. 16, No.2, June 1984

[27] C.M.R. Leung, G.M. Nijssen : "*From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema*", the 10th Australian Computer Science Conference,February 1987

[28] Jack Minger (Editor) : "*Foundation of Deductive Database and Logic Programming*", Morgan Kaufmann Publishers,Inc ,Los Altos California

[29] G.M. Nijssen, T.A. Halpin : "*Conceptual Schema and Relational Database Design : A fact oriented approach*",Prentice Hall 1989

# ภาคผนวก ก

# ระบบจัดการฐานข้อมูลแบบสัมพันธ์ออราเคิล

# (ORACLE RELATIONAL DATA BASE MANAGEMENT SYSTEM)

โดยทั่วไประบบจัดการฐานข้อมูลแบบสัมพันธ์ (RDBMS : Relational Database Management System) เป็นตัวเชื่อมต่อระหว่างหน่วยเก็บข้อมูลแบบกายภาพกับข้อมูลทางตรรกะ ในทางปฏิบัติจะแบ่งออกเป็นกลุ่มๆ ตามลักษณะการกระทำกับข้อมูลบนเครื่องมือ (Tools) โดยสามารถใช้เครื่องมือเหล่านี้ในการทำงานดังต่อไปนี้

- ให้นิยามของฐานข้อมูล (Define a database)
- สอบถามข้อมูลจากฐานข้อมูล (Query the database)
- เพิ่มเติม แก้ไข้ และลบข้อมูล (Add,edit and delete data)
- เปลี่ยนแปลงโครงสร้างของฐานข้อมูล (Modify the structure of the database)
- รักษาความปลอดภัยแก่ข้อมูลจากการใช้งาน (Secure data from public access)
- ติดต่อสื่อสารระหว่างระบบเครือข่าย (Communicate within networks)
- ส่งข้อมูลออก และรับข้อมูลเข้า (Export and import data)

เนื่องจากระบบจัดการฐานข้อมูลแบบสัมพันธ์อนุญาตให้ผู้ใช้ควบคุมข้อมูลได้เต็มที่ ทำให้มีผลิตภัณฑ์รองรับการจัดการฐานข้อมูลแบบสัมพันธ์เกิดขึ้นมากมาย ซึ่งความสามารถเหล่านี้ ทำให้ผู้ใช้สามารถเลือกทำงานตามที่ต้องการได้

ในปัจจุบันระบบจัดการฐานข้อมูลแบบสัมพันธ์ ต้องการที่จะให้ผู้ใช้มากกว่า 1 คนใช้งานกับฐานข้อมูลเดียวกันโดยมีความปลอดภัยเท่าที่ควร ดังนั้นโปรแกรมประยุกต์เกี่ยวกับฐานข้อมูลได้มีการพัฒนามาเป็นระบบที่มีผู้ใช้หลายคน (Multiuser) ออราเคิลจะมีเครื่องมือช่วยหลายๆ แบบเพื่อจัดการข้อมูลและการพัฒนาโปรแกรมประยุกต์ สำหรับเวอร์ชัน 6.0 ออราเคิลได้แบ่งกลุ่มของโปรแกรมกำหนดหน้าที่ออกเป็นผลิตภัณฑ์ ซึ่งมีหลายทางที่จะเข้าไป และใช้ระบบตามที่ต้องการ

**1. การจัดการข้อมูลของออราเคิล (ORACLE)**

ในออราเคิลข้อมูลทั้งหมดจะถูกเก็บและแสดงในรูปของตาราง (Table) ในแต่ละตารางจะประกอบไปด้วยคอลัมน์ (Columns) หรือบางทีเรียกว่า ฟิลด์ (fields) และแถว (Rows) แต่ละแถวข้อมูลจะเรียกว่า 1 เรคอร์ด จากตารางผู้ใช้อาจจะเลือกสับเซตของแถวหรือคอลัมน์ แต่ผลลัพธ์จะแสดงออกมาบนหน้าจอ หรือถูกพิมพ์ออกมาเป็นรูปแบบของตารางด้วย

วิว (View) ได้มาจากตารางซึ่งผู้ใช้สามารถสร้างเพื่อจุดประสงค์สำหรับการแสดงผล ถึงแม้ว่า วิวจะมองดูเหมือนกับว่าเป็นตารางจริง ๆ แต่ในฐานข้อมูล วิวจะเก็บในลักษณะนิยามเท่านั้น ด้วยเหตุผลนี้ วิวก็จะถูกอ้างว่าเป็นตารางเสมือน (Virtual Tables) โดยตารางที่ถูกวิวถ่ายทอดเรียกว่า ตารางพื้นฐาน (Base Tables) วิวอาจจะเกิดจากการรวมกันของ 2 ตารางพื้นฐานหรือเป็นซับเซตของตารางพื้นฐาน ผู้ใช้สามารถใช้วิวเพื่อที่จะเข้าถึงตารางได้ และเพื่อทำให้งานที่มีความยุ่งยากให้มีความง่ายขึ้น

เนื่องจาก ORACLE เป็นระบบจัดการฐานข้อมูลแบบสัมพันธ์ ผู้ใช้จึงสามารถที่จะเชื่อมโยงข้อมูลที่ถูกเก็บไว้ในหลาย ๆ ตารางเพื่อเพิ่มประสิทธิภาพในการทำงานและเพื่อการหลีกเลี่ยงการทำซ้ำ การเลือก (Selection) เป็นกระบวนการเพื่อสร้างตารางใหม่ซึ่งประกอบไปด้วยกลุ่มของแถวจากตารางใดๆ ซึ่งตรงกับเกณฑ์ที่ต้องการ โปรเจ็คชั่น (Projection) เป็นกระบวนการเพื่อสร้างตารางจากกลุ่มของคอลัมน์จากตารางใดๆ ซึ่งตรงกับเกณฑ์ที่ต้องการ การจอยน์ (Join) จะสร้างตารางใหม่ซึ่งเป็นการเชื่อมกันของแถวทั้งหมดใน 2 ตาราง โดยไม่รวมแถวที่มีข้อมูลเหมือนกัน

**2 การเข้าถึงข้อมูลของออราเคิล**

ส่วนแก่น(Core) ของออราเคิลคือ เอสคิวแอล (SQL:Structured Query Language) เป็นภาษาซึ่งผู้ใช้สามารถติดต่อกับออราเคิล เอสคิวแอลประกอบไปด้วยกลุ่มคำพื้นฐานของภาษาอังกฤษ เช่น Select และ Create เป็นต้น ซึ่งผู้ใช้สามารถใช้คำสั่งโครงสร้างเพื่อที่จะเข้าถึงและจัดการข้อมูลที่ถูกเก็บไว้ในฐานของมูลแบบสัมพันธ์

ออราเคิลใช้เอสคิวแอลเพื่อที่จะเข้าถึง (Access), เปลี่ยนแปลง (Modify) และแสดงผล(Display) ข้อมูล ออราเคิลยังสามารถใช้เครื่องมือ (Tools) อื่นในการกระทำกับข้อมูลได้เช่นกัน

**3 เหตุผลของการเลือกออราเคิล**

ออราเคิลมีมาหลายปีในวงการระบบฐานข้อมูลแบบสัมพันธ์ ถึงแม้จะไม่มีความเป็นฟังก์ชัน และระบบที่ใช้กับไมโครคอมพิวเตอร์ เหมือนกับ dBASE แต่ออราเคิลมีความสามารถในการใช้งานแบบผู้ใช้หลายคน การเข้าถึงการควบคุมและ เอสคิวแอลคอมเพททิบิลิตี้ (SQL-Compatibility) ก็ถูกเพิ่มเข้ามา นอกจากนี้ออราเคิลยังมีข้อดีสำหรับการใช้งาน คือ

3.1 ออราเคิลให้ความปลอดภัยแก่ผู้ใช้

ออราเคิลมีหลายลักษณะเพื่อที่จะให้ความแน่นอนในความถูกต้องของฐานข้อมูล เมื่อมีการขัดจังหวะ (Interruption) เกิดขึ้นในขณะที่ทำงานอยู่ การกลับคืนสู่ระดับเดิม (Rollback) สามารถที่จะเริ่มให้ฐานข้อมูลชี้ไปยังตำแหน่งที่ก่อนที่จะเกิดความผิดพลาด

ออราเคิลมีฟังก์ชันมากมายสำหรับการรักษาความปลอดภัยของข้อมูล คำสั่งแกรนท์ (Grant) และรีโว้ค (Revoke) จะจำกัดการเข้าถึงของข้อมูลลงไปถึงระดับแถวและคอลัมน์ วิว ก็เป็นรูปแบบหนึ่งสำหรับการจำกัดการเข้าถึงตารางพื้นฐานในระบบฐานข้อมูล ซึ่งผู้ใช้จะพบว่ามีหลายทางที่จะควบคุมการเข้าถึงฐานข้อมูลในออราเคิล

3.2 ออราเคิลสามารถดำเนินงานได้อย่างเหมาะสม

ออราเคิล ได้ถูกปรับปรุงอย่างต่อเนื่อง เพื่อที่จะให้มีความสามารถดำเนินงานได้อย่างเหมาะสมบนฐานข้อมูลที่มีขนาดใหญ่ เนื่องจากระบบฐานข้อมูลแบบสัมพันธ์มีข้อเสียในเรื่องของความล่าช้าในการเข้าถึงข้อมูลเป็นอย่างมาก ออราเคิลได้พัฒนาตัวเองอย่างตัวเนื่อง ซึ่งผลก็คือชุดคำสั่งประจำสามารถคำนวณหาทางที่ดีที่สุดที่จะไปยังข้อมูลอย่างรวดเร็วโดยอัตโนมัติ เทคนิคในการจัดกลุ่มของออราเคิล เพื่อที่จะเก็บข้อมูลลงบนดิสก์ก็เป็นอีกคุณสมบัติหนึ่ง และยังมีฟังก์ชันเพิ่มเติมช่วยควบคุมการติดตามฐานข้อมูลที่มีความซับซ้อนด้วย

3.3 ออราเคิลสนับสนุนการพัฒนาโปรแกรมประยุกต์

SQL*Forms เป็นเครื่องมือที่ให้ผู้ใช้ใช้งานได้ง่าย (User Friendly) อย่างดียิ่ง ในการสร้างฟอร์มผู้ใช้สามารถเริ่มต้นด้วยการใช้ฟอร์มที่มีอยู่แล้วหรือฟังก์ในการจัดการหน้าจอเพื่อสร้างรายละเอียดบนหน้าจอ สำหรับการเข้าถึง และการแก้ไข้ ตารางหลาย ๆ ตาราง และเพื่อควบคุมและป้อนข้อมูลเข้าไป ใน SQL*Forms ออราเคิลได้จัดให้มีหน่วยควบคุมที่เรียกว่าทริกเกอร์ (Trigger) โดยมีผลในการทำงานของผู้ใช้บนฟิลด์ก่อน ระหว่าง และหลังการป้อนข้อมูลทริกเกอร์เหล่านี้สามารถปฏิบัติตามคำสั่งเอสคิวแอลและคำสั่งใน SQL*Forms หรือโปรแกรมย่อยภายนอกของภาษาเชิงกระบวนคำสั่ง(Procedural Language) SQL*Forms เป็นเครื่องมือของภาษายุคที่ 4 (4GL) ซึ่งจะปรับปรุงตามความต้องการของผู้ใช้ได้อย่างดี

3.4 ออราเคิลใช้ชุดคำสั่งเอสคิวแอล

ออราเคิลใช้ชุดคำสั่งเอสคิวแอล ซึ่งใกล้เคียงกับมาตรฐาน ANSI และมีความเข้ากันได้กับ DB2 ของ IBM และ DS/SQL ผู้ใช้จะได้รับประโยชน์ของการปรับปรุงอย่างมากของออราเคิล เพื่อให้เข้าใกล้มาตรฐานเอสคิวแอล ออราเคิลได้เพิ่มคำสั่งรูปแบบของการรายงานเพื่อที่จะขยายความสามารถในการรายงานผลของผลลัพธ์ และเพื่อที่จะให้มีการแสดงผลการรายงานอย่างต่อเนื่อง นอกจากนี้ออราเคิลได้เพิ่มฟังก์ชันทางสถิติ (Statistical), ทางคณิตศาสตร์ (Arithmetic), ชุดอักขระ (String) และวันที่เวลา (Date/Time) ด้วย

# ภาคผนวก ข

# โปรสตาร์ซี (Pro*c)

ระบบจัดการฐานข้อมูลออราเคิล (ORACLE Relational Database Management System) ได้สร้างโปรแกรมที่ใช้ในการเปลี่ยนประโยคคำสั่งของเอสคิวแอล ที่ประกอบอยู่ในโปรแกรมภาษาซี ซึ่งจะนำหน้าด้วย "EXEC SQL" ให้เป็นประโยคคำสั่งในภาษาซี ซึ่งสามารถจัดการฐานข้อมูลของออราเคิลได้ และเรียกวิธีนี้ว่า "การพรีคอมไพล์" (Precompiler) และจากนั้นจึงนำผลที่ได้ มาคอมไพล์หรือลิงก์ โดยใช้คอมไพเลอร์ของภาษาสูงที่ใช้ เพื่อให้ได้โปรแกรมที่สามารถปฏิบัติการได้ ซึ่งโปรแกรมที่ใช้ในการพรีคอมไพล์นี้คือ "โปรสตาร์ซีทลู" (Pro*c Tool)

ดังนั้น เราสามารถสรุปขั้นตอนในการเขียนโปรแกรมโปรสตาร์ซี ได้ดังนี้

1. เขียนโปรแกรมโปรสตาร์ซี
2. พรีคอมไพล์ โดยใช้โปรสตาร์ซีทลู
3. คอมไพล์แฟ้มข้อมูลที่ได้จากการคอมไพล์
4. ลิงค์แฟ้มข้อมูลที่เกี่ยวข้องกัน
5. รันโปรแกรม

**1. ประโยคคำสั่งของเอสคิวแอลในการปฏิบัติการและการกำหนด**

ประโยคคำสั่งของเอสคิวแอลที่ใช้ในโปรแกรมโปรสตาร์ซี สามารถแบ่งได้ 2 ประเภทคือ

1. ประโยคคำสั่งปฏิบัติการ คือคำสั่งเกี่ยวกับการจัดการข้อมูล (Data Manipulation Language), การกำหนดข้อมูล (Data Definition Language) และการควบคุมข้อมูล (Data Control Language) ซึ่งการปฏิบัติการของประโยคคำสั่งเหล่านี้จะมีผลต่อส่วนการติดต่อข้อมูลของเอสคิวแอล (SQL Communication Area) หรือเรียกอีกอย่างว่าเอสคิวแอลซีเอ (SQLCA) โดยจะมีการแสดงรหัสสถานะที่ได้จากผลของประโยคคำสั่งการปฏิบัติการที่ส่วนนี้ นอกจากนี้ ยังมีผลต่อการเริ่มต้น หรือการสิ้นสุดของส่วนการเก็บการเรียงลำดับของประโยคคำสั่งเอสคิวแอล (Logical Unit of Work)

2. ประโยคคำสั่งการกำหนด คือประโยคคำสั่งที่จะไม่มีการแสดงรหัสสถานะของการปฏิบัติที่ส่วนการติดต่อข้อมูลของเอสคิวแอล และส่วนการเก็บการเรียงลำดับของประโยคคำสั่งของเอสคิวแอล

**2. ส่วนประกอบในโปรแกรมโปรสตาร์ซี (Parts of the Pro*C Program)**

โปรแกรมโปรสตาร์ซี สามารถแบ่งได้เป็น 2 ส่วน คือ

1. ส่วนโครงสร้างการประยุกต์ (Application Prologue) คือส่วนกำหนดตัวแปรที่ใช้ในการติดต่อกับฐานข้อมูล, การเตรียมพร้อม และการเริ่มต้นเพื่อที่จะติดต่อกับฐานข้อมูล

2. ส่วนตัวโปรแกรมประยุกต์ (Application body) คือ ส่วนของโปรแกรมที่ประโยคคำสั่งต่าง ๆ ของเอสคิวแอล ในการจัดการฐานข้อมูลที่ปรากฏอยู่

**3. ส่วนโครงสร้างการประยุกต์**

สามารถแบ่งออกได้เป็น 3 ส่วน คือ

1. ส่วนกำหนดตัวแปร (DECLARE section) คือ ส่วนที่ใช้กำหนดตัวแปรและชนิดของตัวแปรในภาษาซีที่จะใช้ในการติดต่อกับฐานข้อมูล มีรูปแบบในการใช้โดยจะเริ่มต้นด้วยประโยค

EXEC SQL BEGIN DECLARE SECTION;

และจบด้วยประโยค

EXEC SQL END DECLARE SECTION;

ระหว่างประโยคทั้ง 2 จะเป็นการกำหนดตัวแปรและชนิดของตัวแปร ซึ่งตัวแปรที่กำหนดในส่วนนี้สามารถแบ่งได้เป็น ตัวแปรหลัก (Host Vaiable) และตัวแปรบ่งชี้ (Indicator Variable) ตัวแปรหลักเป็นตัวแปรที่เก็บค่าข้อมูลต่าง ๆ ที่ใช้ในการติดต่อกับฐานข้อมูล ส่วนตัวแปรบ่งชี้จะใช้ในการเพิ่มข้อมูลในคอลัมน์ที่ไม่ทราบค่า (NULL) และสามารถแสดงให้ทราบว่าข้อมูลที่ตัวแปรหลักได้รับจากฐานข้อมูลมีลักษณะอย่างไร เช่น ไม่ทราบค่า หรือ มีการตัดข้อมูลบางส่วนทิ้ง เป็นต้น

สำหรับชนิดของตัวแปรหลักนั้น จะเหมือนกับชนิดของตัวแปรในภาษาซี แต่ที่เพิ่มขึ้นมาคือประเภท วาร์ชาร์ (VARCHAR) ซึ่งมีลักษณะเป็นแบบเรคอร์ด (Record)

สำหรับข้อผิดพลาดที่เกิดขึ้นในส่วนการกำหนดตัวแปรนี้ ในกรณีที่ผู้ใช้มิได้กำหนดตัวแปรที่ใช้ในโปรแกรม แต่มีการเรียกใช้จะมีการแสดงข้อความผิดพลาด คือ

Undeclared host variable <a> at line <b> in file <c>

2. INCLUDE SQLCA จะมีรูปแบบการใช้คือ

EXEC SQL INCLUDE SQLCA;

ประโยคคำสั่งนี้ จะเป็นการกำหนดให้โปรสตาร์ซี รวมส่วนของการติดต่อข้อมูลกับเอสคิวแอลเข้าไว้ในโปรแกรม โดยขณะที่พรีคอมไพล์ ออราเคิลจะทำหน้าที่เปลี่ยนหรือแทนที่ตัวแปรหลักในโปรแกรมด้วยตัวแปรที่ได้จากแฟ้มข้อมูลที่นำมารวม และหน้าที่สำคัญของการติดต่อข้อมูล

กับเอสคิวแอล อีกอย่างหนึ่ง นอกเหนือจากการติดต่อกับออราเคิล ก็คือ การแสดงข้อผิดพลาดและข้อระวังต่างๆ (Warning) ที่เกิดขึ้นในการปฏิบัติคำสั่งของเอสคิวแอล

- sqlca.sqlcode : ค่ามากกว่า 0 จะแสดงถึงการกระทำคำสั่ง
  ค่าเท่ากับ 0 แสดงว่าทำคำสั่งได้สมบูรณ์
  ค่าน้อยกว่า 0 เกิดการผิดพลาดขึ้น

- sqlca.sqlwarn : จะประกอบด้วยอาร์เรย์ (array) ของแฟลกส์ (flags) 8 ตัว ซึ่งแต่ละตัวก็จะแสดงถึงลักษณะของข้อระวังที่แตกต่างกันออกไป นอกจากจะสามารถเรียกใช้เอสคิวแอลได้แล้ว ยังสามารถใช้คำสั่งเฉพาะหรือติดต่อกับออราเคิลได้โดยตรง โดยใช้คำสั่ง

EXEC SQL INCLUDE ORACA;

3. การติดต่อกับออราเคิล (Connecting to ORACLE) มีรูปแบบการใช้คือ

EXEC SQL CONNECT (:oracleid> IDENTIFIED BY <:oraclepasswd>

หรือ

EXEC SQL CONNECT <:oracleid>

โดยที่ oracleid อยู่ใน <:oracleid>/<:oraclepasswd> จะเป็นส่วนที่ต้องใช้เพื่อให้โปรแกรมสามารถติดต่อกับระบบจัดการฐานข้อมูลแบบสัมพันธ์ของออราเคิลได้ ซึ่งจะทำให้สามารถเข้าถึงฐานข้อมูลของออราเคิลได้

4. ตัวโปรแกรม (Application Body)

เป็นส่วนที่ภาษาหลัก (Host) และภาษา Embeded รวมกันอยู่ ลักษณะโดยทั่วไปของโปรแกรม คือ

- ภาษาหลัก จะเป็นตัวจัดการเกี่ยวกับการแสดงผล (display) และรูปแบบการใช้งานต่าง ๆ ของโปรแกรม เช่น เมนู เป็นต้น

- ภาษา Embeded จะทำงานในด้านการจัดการเกี่ยวกับข้อมูล รวมทั้งการเรียกใช้คำสั่งของเอสคิวแอลและออราเคิลด้วย ซึ่งการเรียกใช้นั้น จะต้องมี "EXEC SQL" นำหน้าก่อนเสมอ

5. การถามตอบกับเอมเบดด์ (Query)

เป็นส่วนหนึ่งในตัวโปรแกรม (Application Body) ซึ่งจะใช้ในการเรียกข้อมูลมาใช้หรือเก็บข้อมูลต่าง ๆ

สำหรับคำสั่งที่ใช้ในการสอบถาม (Query) จะประกอบด้วย

- SELECT - INTO

- FROM - WHERE
- CONNECT - UNION
- INTERSECT - MINUS
- GROUP BY - HAVING
- ORDER BY

สำหรับตัวแปรที่ใช้ในการสอบถามนั้น มาจาก 2 ที่ คือ จากตารางในภาษาเอมเบดด์ และจากตัวแปรในภาษาหลัก ซึ่งตัวแปรในภาษาหลักที่ใช้ในการสอบถามจะต้องมีเครื่องหมาย ":" (colon) นำหน้าชื่อตัวแปรเสมอ

ลักษณะการสอบถาม มี 2 แบบ คือ

5.1.การสอบถามที่ให้ผลลัพธ์ออกมาเพียง 1 แถว (Query which return SINGLE ROW only) เป็นการสอบถามที่จะต้องอ้างกับค่าที่มีเพียง 1 แถวในตารางเท่านั้น (Unique index) ซึ่งถ้าให้ค่ามากกว่า 1 แถวจะแสดงข้อผิดพลาดออกมา

5.2. การสอบถามที่ให้ผลลัพธ์มากกว่า 1 แถว (Query which return MULTIPLE ROWS) การสอบถามลักษณะนี้มักจะใช้กับการเรียกข้อมูลที่มีเป็นกลุ่มในตาราง ซึ่งเมื่อกระทำการสอบถามนี้แล้ว เอสคิวแอลจะให้ผลลัพธ์ทั้งหมดออกมาในครั้งเดียว ดังนั้นการใช้การสอบถามแบบนี้จึงจำเป็นต้องเตรียมเนื้อที่ส่วนหนึ่งในออราเคิลหรือเอสคิวแอล เพื่อที่จะใช้ในการเก็บผลลัพธ์นั้นไว้ แล้วจึงเรียกออกมาใช้ตามที่ต้องการ ซึ่งพื้นที่นั้นเรียกว่า เคอร์เซอร์ (Cursor)

เคอร์เซอร์มีลักษณะการใช้ ดังนี้คือ

- การประกาศเคอร์เซอร์ (DECLARE CURSOR) เพื่อกำหนดพื้นที่ , ชื่อ และการสอบถามที่ต้องการ

รูปแบบ : EXEC SQL DECLARE <cursorname> CURSOR FOR [Query];

- การเปิดเคอร์เซอร์ (OPEN CURSOR) เพื่อเปิดให้สามารถเรียกใช้เคอร์เซอร์ได้

รูปแบบ : EXEC SQL OPEN <cursorname>;

- การเฟทช์ (FETCH) ให้เคอร์เซอร์แสดงผลลัพธ์ตัวต่อไป

รูปแบบ : EXEC SQL FETCH <cursorname> INTO <HostVar>;

- การปิดเคอร์เซอร์ (CLOSE CURSOR) เป็นการยกเลิกเคอร์เซอร์ที่ระบุออกไป

รูปแบบ : EXEC SQL CLOSE <cursorname>;

- ตำแหน่งของเคอร์เซอร์ปัจจุบัน (CURRENT OF CURSOR) ใช้ในการอ้างถึงตำแหน่งแถวปัจจุบันในเคอร์เซอร์ที่มีการเฟทช์ข้อมูล เพื่อนำมาใช้ในการแก้ไขหรือลบข้อมูล ตำแหน่งของเคอร์เซอร์ปัจจุบันสามารถใช้แทนด้วย โรว์ไอดี (ROWID)

6. คอมมิท และ โรลแบค (Commit and Rollback)

ในการทำงานของโปรแกรม คำสั่งที่เป็นภาษาเอสคิวแอลแต่ละคำสั่ง จะถูกออราเคิลมองเป็นส่วนย่อย (logical unit of work) ซึ่งในแต่ละส่วนนี้ จะถูกประมวลผลเป็นลำดับขั้นไปจนจบ หรืออาจมีการถูกยกเลิกกลางคันก็ได้ สำหรับการยกเลิก unit of work นั้นเกิดได้จาก 2 กรณี คือ

1. ผู้ใช้ (user) ยกเลิกเอง

2. ระบบ (system) ไม่สามารถทำงานต่อไปได้ เช่น เกิดเดดล็อก (Deadlock) ขึ้นและการจบของ unit of work มี 2 แบบ คือ

- การคอมมิทเวิร์ก (Commit work) เป็นการจบ unit of work โดยให้เก็บการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นไว้ในฐานข้อมูล มีรูปแบบการใช้คือ

```
EXEC SQL COMMIT WORK [RELEASE];
```

โดยที่ทางเลือก RELEASE จะเป็นการคืนเนื้อที่หน่วยความจำทั้งหมดและออกจากระบบ ซึ่งจะเป็นการจบ unit of work สุดท้าย

- โรลแบคเวิร์ก (Rollback Work) เป็นการจบ unit of work เช่นกัน แต่จะยกเลิกการแก้ไขข้อมูลทั้งหมด จะใช้ในกรณีที่เกิดการผิดพลาดในการทำงานของโปรแกรม มีรูปแบบการใช้ลักษณะเดียวกับคอมมิท คือ

```
EXEC SQL ROLLBACK WORK [RELEASE]
```

7. การแสดงความผิดพลาด (Error and Warning)

หน้าที่สำคัญอีกประการหนึ่งของส่วนติดต่อข้อมูลของเอสคิวแอล คือ เป็นส่วนที่จะกระทำเกี่ยวกับการแสดงความผิดพลาดของโปรแกรม หรือในส่วนต่างๆของเอสคิวแอล โดยที่ลักษณะของส่วนการติดต่อข้อมูลของเอสคิวแอล

8. การตรวจข้อผิดพลาด (Error Detection)

ในการตรวจสอบข้อผิดพลาด จะใช้คำสั่ง WHENEVER ซึ่งจะตรวจสอบสถานะของส่วนการติดต่อข้อมูลของเอสคิวแอลทุก ๆ ครั้งที่กระทำคำสั่งเอสคิวแอล มีรูปแบบการใช้ดังนี้

```
EXEC SQL WHENEVER [SQLERROR]  [STOP]
                  [SQLWARNING] [CONTINUE]
                  [NOT FOUND]  [GOTO label]
```

โดยที่

- SQL ERROR : จะถูก set เมื่อ sqlca.sqlcode เป็นลบ
- SQL WARNING : จะถูก set เมื่อ sqlca.sqlwarn[0] = "w"

- NOT FOUND : จะถูก set เมื่อ sqlca.sqlcode = +1403 (no row found)
- STOP : หยุดการทำงานของโปรแกรม และ โรลแบค
- CONTINUE : ทำงานต่อไป ไม่ว่า sqlca จะเป็นอย่างไร
- GOTO label : ข้ามไปทำงานที่ label

# ภาคผนวก ค

# ตารางในฐานข้อมูล

สำหรับตารางต่าง ๆ ของแบบจำลองการเก็บข้อมูล ไนแอม (NIAM) ของ เมต้าสกีมา (META SCHEMA) และแมปปิ้งอินฟอร์เมชั่น (Mapping Information) ได้แสดงในส่วนสถาปัตยกรรมของฐานความรู้ ส่วนตารางความสัมพันธ์ของเมต้าสกีมา และ แมปปิ้งอินฟอร์เมชั่น มีลักษณะและรายละเอียดดังนี้

**ส่วนเมต้าสกีมา**□

(1) TABLE OBJECT_TYPE

| OBJECT_NAME | KIND_OF_OBJECT |
|---|---|
| CHAR(20) | CHAR(10) |

(2) TABLE LABEL_TYPE

| OBJECT_NAME | TYPE_NAME | LABEL_ORDER |
|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(5) |

(3) TABLE SUBTYPE

| SUB_NAME | SUPER_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(4) TABLE ROLE

| ROLE_NAME | ROLE_TYPE | RELATION_NAME | ROLE_ORDER |
|---|---|---|---|
| CHAR(20) | CHAR(10) | CHAR(10) | CHAR(1) |

(5) TABLE OBJECT_ROLE

| ROLE_NAME | OBJECT_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(6) TABLE NESTING

| ROLE_NAME | RELATION_NAME |
|---|---|
| CHAR(20) | CHAR(10) |

(7) TABLE RELATION_TYPE

| RELATION_NAME | KIND_OF_RELATION |
|---|---|
| CHAR(10) | CHAR(10) |

(8) TABLE ROLE_GROUP

| ROLE_NAME | ROLE_GROUP |
|---|---|
| CHAR(20) | CHAR(10) |

(9) TABLE CONSTRAINT

| CONST# | KIND_OF_CONST | CONST_NOTE |
|---|---|---|
| CHAR(10) | CHAR(10) | CHAR(1) |

(10) TABLE CONST_ROLE

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

(11) TABLE OBJECT_CONST

| CONST# | OBJECT_NAME |
|---|---|
| CHAR(10) | CHAR(20) |

(12) TABLE RANGE_CONST

| CONST# | MIN_NUMBER | MAX_NUMBER |
|---|---|---|
| CHAR(10) | CHAR(5) | CHAR(5) |

(13) TABLE MEMBERSHIP_CONST

| CONST# | MEMBER |
|---|---|
| CHAR(10) | CHAR(10) |

(14) TABLE SUBSET_CONST

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

(15) VIEW SUPERSET_CONST

| CONST# | ROLE_GROUP |
|---|---|
| CHAR(10) | CHAR(10) |

```
CREATE VIEW SUPERSET_CONST AS
        SELECT  T1.CONST# , T1.ROLE_GROUP
        FROM  CONST_ROLE T1 , SUBSET_CONST T2
        WHERE  T1.CONST# = T2.CONST#
        AND  T2.ROLE_GROUP <> T1.ROLE_GROUP ;
```

(16) TABLE FREQ_CONST

| CONST# | UPPER_LIMIT | LOWER_LIMIT |
|---|---|---|
| CHAR(10) | CHAR(5) | CHAR(5) |

**ส่วนแมปปิ้งอินฟอร์เมชั่น**□

(1) TABLE UNIQ_REF

| ENTITY_NAME | RELATION_NAME | ROLE_NAME | LABEL_NAME | COL_NAME |
|---|---|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) |

(2) TABLE NONUNIQ_REF

| ENTITY_NAME | RELATION_NAME | LABEL_NAME | COL_NAME |
|---|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(20) | CHAR(20) |

(3) TABLE NORMAL_FACTTYPE

| RELATION_NAME | TABLE_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(4) TABLE REFERENCE

| RELATION_NAME | TABLE_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(5) TABLE PRIMARY_KEY

| TABLE_NAME | COL_NAME |
|---|---|
| CHAR(20) | CHAR(20) |

(6) TABLE DERIVED_FACTTYPE

| RELATION_NAME | VIEW_NAME | DERIVED_DEF |
|---|---|---|
| CHAR(10) | CHAR(20) | CHAR(100) |

**ข้อมูลรายละเอียดของแอตตริบิวท์ต่าง ๆ ในตารางเมต้าสกีมม่าและแมปปิ้งอินฟอร์เมชั่น□□**

| ชื่อแอตตริบิวท์ | จำนวนอักขระ | ความหมาย |
|---|---|---|
| OBJECT_NAME | 20 | เป็นตัวเก็บชื่อของ Entity หรือ Label |
| KIND_OF_OBJECT | 10 | เก็บชนิดของ Object_Name ("LABEL", "ENTITY") |
| TYPE_NAME | 20 | ชี้ให้เห็นว่า Object_Name นี้ เมื่อแมปเป็นตารางแล้ว จะเก็บข้อมูลเป็นอย่างไร เช่น C10 จำนวนอักขระ 10 ตัวอักษร, N10 ค่าตัวเลขขนาด 10 หลัก เป็นต้น |
| LABEL_ORDER | 5 | เป็นตัวกำหนดลำดับค่าตัวเลขให้กับ Label ต่อ 1 Entity (1,2,3,4,5,...) |
| SUB_NAME | 20 | ชื่อ Subtype ของ Entity |
| SUPER_NAME | 20 | ชื่อ Supertype ของ Entity |
| ROLE_NAME | 20 | ชื่อของ Role |
| ROLE_TYPE | 10 | ชนิดของ Role ("NORMAL","NESTING") |
| ROLE_ORDER | 1 | กำหนดลำดับของ Role (1,2,3,4,....,9) |
| RELATION_NAME | 10 | ชื่อของความสัมพันธ์ (จะเก็บชื่อของชนิดความจริง และ ชื่อของชนิดอ้างอิง) |
| KIND_OF RELATION | 10 | ชนิดของความสัมพันธ์ ( "F_NORMAL", " F_DERIVED","REF") |
| ROLE_GROUP | 10 | กลุ่มของ Constraint |

| | | |
|---|---|---|
| CONST# | 10 | ID ของ Constraint (C1,C2,...,Cn) |
| KIND_OF_CONST# | 10 | ชนิดของ Constraint โดยจะมีอยู่ 7 ชนิดดังนี้ (UNIQUE, EXCLUSIV, EQUAL, SUBSET, RANGE, FREQ, MEMBER, MAND) |
| CONST_NOTE | 1 | เป็น Flag แสดงว่าจะให้มีการเช็คConstraints หรือไม่ ( N ไม่เช็ค, Y เช็ค ) |
| MIN_NUMBER | 5 | ความถี่ล่างที่ใช้ในการกำหนด Occurence frequency เก็บเป็นค่าจำนวนเต็ม |
| MAX_NUMBER | 5 | ความถี่บนที่ใช้ในการกำหนด Occurence frequency เก็บค่าจำนวนเต็ม |
| MEMBER | 10 | สมาชิกที่กำหนดใน Membership constraint เก็บเป็นสตริงขนาดไม่เกินที่กำหนด |
| UPPER_LIMIT | 5 | เป็นค่าตัวเลขที่มากที่สุดที่เก็บได้ในแต่ละ Object |
| LOWER_LIMIT | 5 | เป็นค่าตัวเลขที่น้อยที่สุดที่เก็บได้ในแต่ละ Object |
| ENTITY_NAME | 20 | ชื่อเอนติตี้ |
| LABEL_NAME | 20 | ชื่อเลเบิ้ล |
| COL_NAME | 20 | ชื่อคอลัมน์ |
| TABLE_NAME | 20 | ชื่อตาราง |
| VIEW_NAME | 20 | ชื่อวิว |
| DERIVED_DEF | 100 | สูตรที่ใช้ในการ Derived |

และยังได้มีการสร้างตารางและวิวต่าง ๆ เพิ่มเพื่อใช้ในงานส่วนนี้ดังต่อไปนี้

1. วิวของแฟคไทป์ต่าง ๆ ซึ่งจะใช้เป็นเพรดิเคตในฐานความรู้

2. ตาราง "VIEW_COLUMN" ใช้เก็บรายละเอียดต่าง ๆ ของวิวของแฟคไทป์

สำหรับวิวในข้อ 1 และตารางในข้อ 2 จะสร้างในขั้นตอน Information Definition

3. ตาราง "CONST_QUERY" ใช้เก็บ query สำหรับการตรวจสอบคอนสเตรนท์ ในส่วนตรวจสอบ ตารางนี้จะสร้างในส่วนเตรียมการของขั้นตอน Constraint Handling

4. วิวที่สร้างขึ้นเพื่อความสะดวกในการตรวจสอบคอนสเตรนท์ ประกอบด้วยการจอยน์ตารางในเมต้าสกีมา ดังนี้

4.1 วิว CONST_KIND_RG เกิดจากการ join ตาราง CONST_ROLE และตาราง CONSTRAINT

4.2 วิว RNAME_RELATION_RG เกิดจากการจอยตาราง NESTING และตาราง ROLE_GROUP

4.3 วิว RNAME_OBJECT_RG เกิดจากการจอยตาราง OBJECT_ROLE และ ROLE_GROUP

4.4 วิว TT1 จากการจอยวิว CONST_KIND_RG และ วิว RNAME_RELATION_RG

4.5 วิว TT2 เกิดจากการจอยวิว CONST_KIND_RG และวิว RNAME_OBJECT_RG

5. วิวของแฟคไทป์ที่เกิดจากการจอยน์แฟคไทป์ทั้งหมด ที่จะต้องตรวจสอบภายในคอนสเตรนท์หนึ่ง ๆ ซึ่งชื่อวิวจะขึ้นต้นด้วย "T_C" และตามด้วยหมายเลขของคอนสเตรนท์นั้น ๆ

สำหรับโครงสร้างของวิวและตารางข้างต้นเป็นดังนี้

(1) ตาราง VIEW_COLUMN

| VIEW_NAME | COLUMN_NAME | COLUMN_ORDER | PRI_KEY_ID | DEDUCIBLE_ID |
|---|---|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(2) | CHAR(1) | CHAR(1) |

(2) ตาราง CONST_QUERY

| CONST# | QUERY1 | QUERY2 | NOTE |
|---|---|---|---|
| CHAR(10) | CHAR(240) | CHAR(240) | CHAR(2) |

(3) วิว CONST_KIND_RG

| CONST# | KIND | RG# |
|---|---|---|
| CHAR(10) | CHAR(10) | CHAR(10) |

(4) วิว RNAME_RELATION_RG

| ROLE_NAME | RELATION_NAME | ROLE_GROUP |
|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(10) |

(5) วิว RNAME_OBJECT_RG

| ROLE_NAME | OBJECT_NAME | ROLE_GROUP |
|---|---|---|
| CHAR(20) | CHAR(20) | CHAR(10) |

(6) วิว TT1

| CONST# | KIND_OF_CONST | ROLE_GROUP | ROLE_NAME | RELATION_NAME |
|---|---|---|---|---|
| CHAR(10) | CHAR(10) | CHAR(10) | CHAR(20) | CHAR(20) |

(7) วิว TT2

| CONST# | KIND_OF_CONST | ROLE_GROUP | ROLE_NAME | OBJECT_NAME |
|---|---|---|---|---|
| CHAR(10) | CHAR(10) | CHAR(10) | CHAR(20) | CHAR(20) |

หมายเหตุ□ สำหรับโครงสร้างของวิวของแฟคไทป์ในข้อ 1 และข้อ 5 มีลักษณะไม่แน่นอนขึ้นอยู่กับลักษณะของแอตทิบิวต์ในแฟคไทป์นั้น ๆ

ข้อมูลรายละเอียดของแอททริบิวต์ต่าง ๆ ในตารางข้างต้น (เฉพาะส่วนเพิ่มเติมจากส่วนเมต้าสกีมา)

| ชื่อแอททริบิวต์ | จำนวนอักขระ | ความหมาย |
|---|---|---|
| VIEW_NAME | 20 | ชื่อชนิดความจริงในฐานข้อมูล |
| COLUMN_NAME | 20 | ชื่อคอลัมน์ในตารางฐานข้อมูลผู้ใช้ |
| COLUMN_ORDER | 2 | ลำดับของคอลัมน์ในฐานข้อมูลจำลองที่เป็นตัวเดียวกับ COLUMN_NAME ในฐานข้อมูลผู้ใช้ |
| PRI_KEY_ID | 1 | ดัชนีที่ระบุว่าคอลัมน์นั้นเป็นคีย์หลัก (primary key) ของชนิดความจริงนั้นหรือไม่ (Y ใช่,N ไม่ใช่) |
| DEDUCIBLE_ID | 1 | ดัชนีที่ระบุว่าเพรดิเคตนั้นได้จากการอนุมานแบบใด (F,N,S,R) |
| CONST# | 10 | ID ของคอนสเตรนต์เช่นเดียวกับในเมต้าสกีม่า |
| QUERY1 | 240 | ที่เก็บคำสั่งที่ใช้ตรวจสอบคอนสเตรนท์ในส่วนตรวจสอบคำสั่งที่ 1 |
| QUERY2 | 240 | เช่นเดียวกับ QUERY1 แต่เป็นคำสั่งที่ 2 |
| NOTE | 2 | ระบุว่าคอนสเตรนท์นั้น ๆ ต้องตรวจสอบหรือไม่ (Y ตรวจสอบได้ , N ตรวจสอบไม่ได้) |

# ภาคผนวก ง
# ข้อมูลที่ใช้ในการทดสอบ

สำหรับภาคผนวก ง. นี้จะรวบรวมเอาข้อมูลทั้งหมดที่ใช้ในการทดสอบในวิทยานิพนธ์นี้เอาไว้ โดยจะเริ่มตั้งแต่การทดสอบส่วนนิยามข้อมูลซึ่งมีข้อมูลอยู่ด้วยกัน 2 ชุดด้วยกัน จากนั้นก็จะเป็นข้อมูลทดสอบในส่วนจัดการข้อมูลซึ่งมีข้อมูลอยู่เพียงชุดเดียว แต่สำหรับข้อมูลที่ใช้ในการทดสอบคอนสแตนต์จะมีค่อนข้างมาก คือ มีถึง 7 ชุดด้วยกัน และสุดท้ายก็จะเป็นข้อมูลที่ใช้ในการทดสอบระบบทั้งหมด โดยข้อมูลทั้งหมดจะมีเฉพาะข้อมูลที่เพียงพอต่อการทดสอบเท่านั้น ข้อมูลที่ไม่จำเป็นต่อการทดสอบนั้นก็จะไม่มี ยกเว้นข้อมูลทดสอบระบบรวมเท่านั้นที่จะมีข้อมูลครบทุกตาราง ก็ขอให้เข้าใจตามนี้นะครับ

**1. ข้อมูลส่วนนิยามข้อมูล**

ในส่วนนิยามข้อมูลนี้จะมีข้อมูลที่ใช้ทดสอบอยู่ 2 ชุดด้วยกัน ซึ่งก็คือข้อมูลจากแบบจำลองข้อมูลไนแอมในรูปที่ 8-4 และในรูปที่ 8-5 โดยข้อมูลเหล่านี้เป็นข้อมูลที่ได้แปลงให้อยู่ในรูปแบบที่สามารถเก็บในเมต้าสกีม่าที่อยู่ในรูปของตาราง ดังนั้นข้อมูลจึงค่อนข้างมาก แม้แบบจำลองจะไม่ซับซ้อนมากก็ตาม ข้อมูลชุดแรกจะอยู่ในไฟล์ PIC84.SQL ซึ่งมีรายละเอียดดังต่อไปนี้

```
DROP TABLE STUDENT;
DROP TABLE SUBJECT_MARK;

CREATE TABLE STUDENT
        (STU_NAME      CHAR(20),
         STU_SURNAME   CHAR(20),
         GENDER                CHAR(1));

CREATE TABLE SUBJECT_MARK
        (STU_NAME      CHAR(20),
         STU_SURNAME   CHAR(20),
         SUBJECT       CHAR(5),
         MARK          CHAR(2));

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('STUDENT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SURNAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SUBJECT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('SUBJ_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('MARK','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('NR','LABEL');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('SURNAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('SUBJ_CODE','C5','1');
INSERT INTO LABEL_TYPE VALUES ('NR','C2','1');
```

```
DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('STU_N','NORMAL','STU_NAME','1');
INSERT INTO ROLE VALUES ('NAME','NORMAL','STU_NAME','2');
INSERT INTO ROLE VALUES ('STU_S','NORMAL','STU_SER','1');
INSERT INTO ROLE VALUES ('SURNAME','NORMAL','STU_SER','2');
INSERT INTO ROLE VALUES ('STU_G','NORMAL','STU_GEN','1');
INSERT INTO ROLE VALUES ('GENDER','NORMAL','STU_GEN','2');
INSERT INTO ROLE VALUES ('GEND','NORMAL','GEN_CODE','1');
INSERT INTO ROLE VALUES ('CODE','NORMAL','GEN_CODE','2');
INSERT INTO ROLE VALUES ('STU_M','NORMAL','STU_SUBJ','1');
INSERT INTO ROLE VALUES ('SUBJECT','NORMAL','STU_SUBJ','2');
INSERT INTO ROLE VALUES ('MARK','NORMAL','STU_SUBJ','3');
INSERT INTO ROLE VALUES ('SUBJ','NORMAL','SUBJ_CODE','1');
INSERT INTO ROLE VALUES ('S_CODE','NORMAL','SUBJ_CODE','2');
INSERT INTO ROLE VALUES ('MARKS','NORMAL','MARK_NR','1');
INSERT INTO ROLE VALUES ('NR','NORMAL','MARK_NR','2');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('STU_N','STUDENT');
INSERT INTO OBJECT_ROLE VALUES ('NAME','NAME');
INSERT INTO OBJECT_ROLE VALUES ('STU_S','STUDENT');
INSERT INTO OBJECT_ROLE VALUES ('SURNAME','SURNAME');
INSERT INTO OBJECT_ROLE VALUES ('STU_G','STUDENT');
INSERT INTO OBJECT_ROLE VALUES ('GENDER','GENDER');
INSERT INTO OBJECT_ROLE VALUES ('GEND','GENDER');
INSERT INTO OBJECT_ROLE VALUES ('CODE','CODE');
INSERT INTO OBJECT_ROLE VALUES ('STU_M','STUDENT');
INSERT INTO OBJECT_ROLE VALUES ('SUBJECT','SUBJECT');
INSERT INTO OBJECT_ROLE VALUES ('MARK','MARK');
INSERT INTO OBJECT_ROLE VALUES ('SUBJ','SUBJECT');
INSERT INTO OBJECT_ROLE VALUES ('S_CODE','SUBJ_CODE');
INSERT INTO OBJECT_ROLE VALUES ('MARKS','MARK');
INSERT INTO OBJECT_ROLE VALUES ('NR','NR');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('STU_NAME','REF');
INSERT INTO RELATION_TYPE VALUES ('STU_SER','REF');
INSERT INTO RELATION_TYPE VALUES ('STU_GEN','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('GEN_CODE','REF');
INSERT INTO RELATION_TYPE VALUES ('STU_SUBJ','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('SUBJ_CODE','REF');
INSERT INTO RELATION_TYPE VALUES ('MARK_NR','REF');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('GENDER','GEN_CODE','GENDER','CODE','GENDER');
INSERT INTO UNIQ_REF VALUES
('STUDENT','STU_NAME','STU_G','NAME','STU_NAME');
INSERT INTO UNIQ_REF VALUES
('STUDENT','STU_SER','STU_G','SURNAME','STU_SURNAME');
INSERT INTO UNIQ_REF VALUES
('STUDENT','STU_NAME','STU_M','NAME','STU_NAME');
INSERT INTO UNIQ_REF VALUES
('STUDENT','STU_SER','STU_M','SURNAME','STU_SURNAME');
INSERT INTO UNIQ_REF VALUES
('SUBJECT','SUBJ_CODE','SUBJECT','SUBJ_CODE','SUBJECT');
INSERT INTO UNIQ_REF VALUES ('MARK','MARK_NR','MARK','NR','MARK');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('STU_GEN','STUDENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('STU_SUBJ','SUBJECT_MARK');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('STU_NAME','STUDENT');
INSERT INTO REFERENCE VALUES ('STU_SER','STUDENT');
```

```
INSERT INTO REFERENCE VALUES ('GEN_CODE','STUDENT');
INSERT INTO REFERENCE VALUES ('SUBJ_CODE','SUBJECT_MARK');
INSERT INTO REFERENCE VALUES ('MARK_NR','SUBJECT_MARK');

DELETE FROM PRIMARY_KEY;
INSERT INTO PRIMARY_KEY VALUES ('STUDENT','STU_NAME');
INSERT INTO PRIMARY_KEY VALUES ('STUDENT','STU_SURNAME');
INSERT INTO PRIMARY_KEY VALUES ('SUBJECT_MARK','STU_NAME');
INSERT INTO PRIMARY_KEY VALUES ('SUBJECT_MARK','STU_SURNAME');

COMMIT WORK
```

สำหรับข้อมูลชุดที่สองก็จะมีรายละเอียดดังต่อไปนี้

```
DROP TABLE PERSON;
DROP TABLE PERSON_ID;
DROP TABLE ST_COURSE
DROP TABLE LECTURER;

CREATE TABLE PERSON
        (P_NAME                 CHAR(20),
         G_CODE                 CHAR(1));

CREATE TABLE PERSON_ID
        (P_NAME                 CHAR(20),
         KIND_NAME      CHAR(5));

CREATE TABLE ST_COURSE
        (STUDENT        CHAR(20),
         COURSE                 CHAR(20);
         MARK           CHAR(5));

CREATE TABLE LECTURER
        (LECTURER       CHAR(20),
         SALARY                 NUMBER);

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('PERSON','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('P_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('G_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('PERSON_KIND','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('KIND_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('STUDENT','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('LECTURER','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('SALARY','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('S_NR','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('COURSE','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('C_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('MARK','ENTIRY');
INSERT INTO OBJECT_TYPE VALUES ('NR','LABEL');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('P_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('G_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('KIND_NAME','C5','1');
INSERT INTO LABEL_TYPE VALUES ('COURSE','C20','1');
INSERT INTO LABEL_TYPE VALUES ('NR','C5','1');
INSERT INTO LABEL_TYPE VALUES ('S_NR','N5','1');

DELETE FROM SUBTYPE;
INSERT INTO SUBTYPE VALUES ('STUDENT','PERSON');
INSERT INTO SUBTYPE VALUES ('LECTURER','PERSON');
```

```
DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('PERSON','NORMAL','PERSON_NAME','1');
INSERT INTO ROLE VALUES ('P_NAME','NORMAL','PERSON_NAME','2');
INSERT INTO ROLE VALUES ('PER_KIND','NORMAL','PERSON_KIND','1');
INSERT INTO ROLE VALUES ('KIND_NAME','NORMAL','PERSON_KIND','2');
INSERT INTO ROLE VALUES ('KIND','NORMAL','KIND_NAME','1');
INSERT INTO ROLE VALUES ('K_NAME','NORMAL','KIND_NAME','2');
INSERT INTO ROLE VALUES ('PER_GEND','NORMAL','PERSON_GEND','1');
INSERT INTO ROLE VALUES ('GEND_NAME','NORMAL','PERSON_GEND','2');
INSERT INTO ROLE VALUES ('GEND','NORMAL','GENDER','1');
INSERT INTO ROLE VALUES ('G_NAME','NORMAL','GENDER','2');
INSERT INTO ROLE VALUES ('STUDENT','NORMAL','STU_COURSE','1');
INSERT INTO ROLE VALUES ('COURSE','NORMAL','STU_COURSE','2');
INSERT INTO ROLE VALUES ('COUR_NAME','NORMAL','COURSE','1');
INSERT INTO ROLE VALUES ('C_NAME','NORMAL','COURSE','2');
INSERT INTO ROLE VALUES ('LECTURER','NORMAL','LECTURE_SAL','1');
INSERT INTO ROLE VALUES ('SALARY','NORMAL','LECTURE_SAL','2');
INSERT INTO ROLE VALUES ('SAL_NR','NORMAL','SALARY','1');
INSERT INTO ROLE VALUES ('S_NR','NORMAL','SALARY','2');
INSERT INTO ROLE VALUES ('ST_COUR','NESTING','ST_CL_MARK','1');
INSERT INTO ROLE VALUES ('MARK','NORMAL','ST_CL_MARK','2');
INSERT INTO ROLE VALUES ('MARK_NR','NORMAL','MARK','1');
INSERT INTO ROLE VALUES ('M_NR','NORMAL','MARK','2');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('PERSON','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('P_NAME','LABEL');
INSERT INTO OBJECT_ROLE VALUES ('PER_KIND','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('KIND_NAME','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('KIND','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('K_NAME','LABEL');
INSERT INTO OBJECT_ROLE VALUES ('PER_GEND','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('GEND_NAME','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('GEND','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('G_NAME','LABEL');
INSERT INTO OBJECT_ROLE VALUES ('STUDENT','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('COURSE','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('COUR_NAME','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('C_NAME','LABEL');
INSERT INTO OBJECT_ROLE VALUES ('LECTURER','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('SALARY','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('SAL_NR','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('S_NR','LABEL');
INSERT INTO OBJECT_ROLE VALUES ('ST_COUR','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('MARK','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('MARK_NR','ENTITY');
INSERT INTO OBJECT_ROLE VALUES ('M_NR','LABEL');

DELETE FROM NESTING;
INSERT INTO NESTING VALUES ('ST_COUR','STU_COURSE');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('PERSON_NAME','REF');
INSERT INTO RELATION_TYPE VALUES ('PERSON_KIND','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('KIND_NAME','REF');
INSERT INTO RELATION_TYPE VALUES ('PERSON_GEND','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('GENDER','REF');
INSERT INTO RELATION_TYPE VALUES ('STU_COURSE','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('COURSE','REF');
INSERT INTO RELATION_TYPE VALUES ('LECTURE_SAL','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('SALARY','REF');
INSERT INTO RELATION_TYPE VALUES ('ST_CL_MARK','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('MARK','REF');
```

```
DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PERSON','PERSON_NAME','PER_KIND','
P_NAME','P_NAME');
INSERT INTO UNIQ_REF VALUES
('PERSON_KIND','PERSON_KIND','KIND_NAME','KIND_NAME','KIND_NAME');
INSERT INTO UNIQ_REF VALUES ('PERSON','PERSON_NAME','PER_GEND','
P_NAME','P_NAME');
INSERT INTO UNIQ_REF VALUES ('GENDER','GENDER','GEND_NAME','G_CODE','G_CODE');
INSERT INTO UNIQ_REF VALUES ('STUDENT','PERSON_NAME','STUDENT','
P_NAME','P_NAME');
INSERT INTO UNIQ_REF VALUES ('COURSE','COURSE','COURSE','COURSE','COURSE');
INSERT INTO UNIQ_REF VALUES
('LECTURER','PERSON_NAME','LECTURER','P_NAME','P_NAME');
INSERT INTO UNIQ_REF VALUES ('SALARY','SALARY','SALARY','S_NR','SALARY');
INSERT INTO UNIQ_REF VALUES ('ST_COURSE','ST_COURSE','ST_COUR',NULL,NULL);
INSERT INTO UNIQ_REF VALUES ('MARK','MARK','MARK','NR','MARK');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('PERSON_KIND','PERSON_ID');
INSERT INTO NORMAL_FACTTYPE VALUES ('PERSON_GEND','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('STU_COURSE','ST_COURSE');
INSERT INTO NORMAL_FACTTYPE VALUES ('ST_CL_MARK','ST_COURSE');
INSERT INTO NORMAL_FACTTYPE VALUES ('LECTURE_SAL','LECTURER');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('PERSON_NAME','PERSON');
INSERT INTO REFERENCE VALUES ('KIND_NAME','PERSON_ID');
INSERT INTO REFERENCE VALUES ('GENDER','PERSON');
INSERT INTO REFERENCE VALUES ('COURSE','ST_COURSE');
INSERT INTO REFERENCE VALUES ('SALARY','LECTURER');
INSERT INTO REFERENCE VALUES ('MARK','ST_COURSE');

DELETE FROM PRIMARY_KEY;
INSERT INTO PRIMARY_KEY VALUES ('PERSON','P_NAME');
INSERT INTO PRIMARY_KEY VALUES ('PERSON_ID','P_NAME');
INSERT INTO PRIMARY_KEY VALUES ('ST_COURSE','STUDENT');
INSERT INTO PRIMARY_KEY VALUES ('ST_COURSE','COURSE');
INSERT INTO PRIMARY_KEY VALUES ('LECTURER','LECTURER');

COMMIT WORK;
```

2. ข้อมูลส่วนนิยามข้อมูล

ในส่วนจัดการข้อมูลนี้จะมีข้อมูลที่ใช้ทดสอบอยู่เพียงชุดเดียว ซึ่งก็คือข้อมูลจากแบบจำลองข้อมูลไนแอมในรูปที่ 8-7 โดยข้อมูลเหล่านี้เป็นข้อมูลที่ได้แปลงให้อยู่ในรูปแบบที่สามารถเก็บในเมต้าสกีม่าที่อยู่ในรูปของตาราง ข้อมูลชุดแรกจะอยู่ในไฟล์ PIC87.SQL ซึ่งมีรายละเอียดดังต่อไปนี้

```
DROP TABLE PATIENT;

CREATE TABLE PATIENT
                (PA_NAME        CHAR(20),
                 PHONE          CHAR(10),
                 GENDER                 CHAR(1));

CREATE VIEW PA_PHONE (PA_NAME, PHONE)
            AS SELECT PA_NAME,PHONE FROM PATIENT;
CREATE VIEW PA_GENDER (PA_NAME, GENDER)
            AS SELECT PA_NAME,PHONE FROM PATIENT;
```

```
DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('PA_PHONE','PATIENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('PA_GENDER','PATIENT');

DELETE FROM VIEW_COLUMN;
INSERT INTO VIEW_COLUMN VALUES ('PA_PHONE','PA_NAME','1','Y','F');
INSERT INTO VIEW_COLUMN VALUES ('PA_PHONE','PHONE','2','N','F');
INSERT INTO VIEW_COLUMN VALUES ('PA_GENDER','PA_NAME','1','Y','F');
INSERT INTO VIEW_COLUMN VALUES ('PA_GENDER','GENDER','2','N','F');

COMMIT WORK
```

3. ส่วนตรวจสอบคอนสแตนต์

ในส่วนนิยามข้อมูลนี้จะมีข้อมูลที่ใช้ทดสอบอยู่ 7 ชุดด้วยกัน ซึ่งก็คือข้อมูลจากแบบจำลองข้อมูลไนแอมในรูปที่ 9-1 ถึงรูปที่ 9-7 โดยข้อมูลเหล่านี้เป็นข้อมูลที่ได้แปลงให้อยู่ในรูปที่สามารถเก็บในเมต้าสกีม่าที่อยู่ในรูปของตาราง ดังนั้นข้อมูลจึงค่อนข้างมาก แม้แบบจำลองจะไม่ซับซ้อนมากก็ตาม

3.1 ข้อมูลของ Uniqueness Constraint

ข้อมูลนี้จะเป็นข้อมูลที่ได้แปลงมาจากรูปที่ 9-1 โดยมีละเอียดของมูลดังนี้

```
DROP TABLE STUDY;
CREATE TABLE STUDY
        (PERSON                 CHAR(20),
         SUBJECT        CHAR(10));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','UNIQ','Y');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');
INSERT INTO CONST_ROLE VALUES ('C1','RG2');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('STUDIES','RG1');
INSERT INTO ROLE_GROUP ('STUDIED_BY','RG2');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('PERSON','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('SURNAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SUBJECT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('S_CODE','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PERSON','STUDY','STUDIES','SURNAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('SUBJECT','STUDY','STUDIED_BY','S_CODE','SUBJECT');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('SURNAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('S_CODE','C10','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('STUDIES','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('STUDIED_BY','SUBJECT');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('STUDIES','NORMAL','STUDY','1');
INSERT INTO ROLE VALUES ('STUDIED_BY','NORMAL','STUDY','2');

DELETE FROM NORMAL_FACTTYPE;
```

```
INSERT INTO NORMAL_FACTTYPE VALUES ('STUDY','STUDY');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('P_REF','STUDY');
INSERT INTO REFERENCE VALUES ('S_REF','STUDY');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('STUDY','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('S_REF','REF');

COMMIT WORK;
```

3.2 ข้อมูลของ Range Constraint และ Membership Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-2 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE MEDEL;
CREATE TABLE MEDEL
        (COUNTRY        CHAR(20),
         MEDEL_KIND     CHAR(1),
         QTY            NUMBER);

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','RANGE','Y');
INSERT INTO CONSTRAINT VALUES ('C2','MEMBER','Y');

DELETE FROM RANGE_CONST;
INSERT INTO RANGE_CONST VALUES ('C1','1','200');

DELETE FROM MEMBERSHIP_CONST;
INSERT INTO MEMBERSHIP_CONST VALUES ('C2','G');
INSERT INTO MEMBERSHIP_CONST VALUES ('C2','S');
INSERT INTO MEMBERSHIP_CONST VALUES ('C2','B');

DELETE FROM OBJECT_CONST;
INSERT INTO OBJECT_CONST VALUES ('C1','NR');
INSERT INTO OBJECT_CONST VALUES ('C2','M_CODE');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('COUNTRY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('C_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('MEDEL_KIND','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('M_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('QTY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('NR','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('COUNTRY','F1','R1','C_NAME','COUNTRY');
INSERT INTO UNIQ_REF VALUES ('MEDEL_KIND','F1','R2','M_CODE','MEDEL_KIND');
INSERT INTO UNIQ_REF VALUES ('QTY','F1','R3','NR','QTY');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('C_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('M_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('NR','N5','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','COUNTRY');
INSERT INTO OBJECT_ROLE VALUES ('R2','MEDEL_KIND');
INSERT INTO OBJECT_ROLE VALUES ('R3','QTY');

DELETE FROM ROLE;
```

```
INSERT INTO ROLE VALUES ('R1','NORMAL','F1','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','F1','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','F1','3');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('F1','MEDEL');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('C_REF','MEDEL');
INSERT INTO REFERENCE VALUES ('M_REF','MEDEL');
INSERT INTO REFERENCE VALUES ('Q_REF','MEDEL');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('F1','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('C_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('M_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('Q_REF','REF');

COMMIT WORK;
```

3.3 ข้อมูลของ Mendatory Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-3 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE PATIENT;
CREATE TABLE PATIENT
        (PATIENT        CHAR(20),
         SEX            CHAR(1),
         PHONE          CHAR(10));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','MENDATORY','Y');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('R1','RG1');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('PATIENT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('P_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SEX','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('S_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('PHONE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('P_NR','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PATIENT','P_SEX','R1','P_NAME','PATIENT');
INSERT INTO UNIQ_REF VALUES ('SEX','P_SEX','R2','S_CODE','SEX');
INSERT INTO UNIQ_REF VALUES ('PATIENT','P_PHONE','R3','P_NAME','PATIENT');
INSERT INTO UNIQ_REF VALUES ('PHONE','P_PHONE','R4','P_NR','PHONE');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('P_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('S_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('P_NR','C10','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','PATIENT');
INSERT INTO OBJECT_ROLE VALUES ('R2','SEX');
INSERT INTO OBJECT_ROLE VALUES ('R3','PATIENT');
INSERT INTO OBJECT_ROLE VALUES ('R4','PHONE');
```

```
DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('R1','NORMAL','P_SEX','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','P_SEX','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','P_PHONE','1');
INSERT INTO ROLE VALUES ('R4','NORMAL','P_PHONE','2');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('P_SEX','PATIENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_PHONE','PATIENT');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('P_REF','PATIENT');
INSERT INTO REFERENCE VALUES ('S_REF','PATIENT');
INSERT INTO REFERENCE VALUES ('PH_REF','PATIENT');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('P_SEX','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_PHONE,'F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('S_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('PH_REF','REF');

COMMIT WORK;
```

3.4 ข้อมูลของ Occurrency Frequency Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-4 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE TUTE;
CREATE TABLE TUTE
        (STUDENT        CHAR(20),
         GROUP          CHAR(10));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','OCCURRENCY','Y');

DELETE FROM FREQ_CONST;
INSERT INTO FREQ_CONST VALUES ('C1','10','30');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('CONTAINS','RG1');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('STUDENT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('S_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('TUTE_GROUP','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('GROUP_NO','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('STUDENT','TUTE','IS_IN','S_NAME','STUDENT');
INSERT INTO UNIQ_REF VALUES
('TUTE_GROUP','TUTE','CONTAINTS','GROUP_NO','GROUP');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('S_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('GROUP_NO','C10','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('IS_IN','STUDENT');
```

```
INSERT INTO OBJECT_ROLE VALUES ('CONTAINS','TUTE_GROUP');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('IS_IN','NORMAL','TUTE','1');
INSERT INTO ROLE VALUES ('CONTAINS','NORMAL','TUTE','2');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('TUTE','TUTE');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('S_REF','TUTE');
INSERT INTO REFERENCE VALUES ('T_REF','TUTE');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('TUTE','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('S_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('T_REF','REF');

COMMIT WORK;
```

3.5 ข้อมูลของ Equality Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-5 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE PERSON;
DROP TABLE SPORT;

CREATE TABLE PERSON
        (PERSON                 CHAR(20),
         YEAR           CHAR(5),
         GENDER                 CHAR(1),
         PERIOD         CHAR(5),
         HEAR_RATE      CHAR(5));

CREATE TABLE SPORT
        (PERSON                 CHAR(20),
         SPORT          CHAR(20));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','EQUALITY','Y');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');
INSERT INTO CONST_ROLE VALUES ('C1','RG2');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('R2','RG1');
INSERT INTO ROLE_GROUP ('R4','RG2');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('PERSON','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('P_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('G_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('YEAR','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('AD','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SPORT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('S_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('PERIOD','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('MS','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('HEAR_RATE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('BEATS','LABEL');
```

```
DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PERSON','P_GENDER','R1','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('GENDER','P_GENDER','R2','G_CODE','GENDER');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_PERIOD','R3','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('PERIOD','P_PERIOD','R4','MS','PERIOD');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_YEAR','R5','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('YEAR','P_YEAR','R6','AD','YEAR');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_HEAR','R7','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('HEAR_RATE','P_HEAR','R8','BEATS','HEAR_RATE');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_SPORT','R9','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('SPORT','P_SPORT','R10','S_NAME','SPORT');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('P_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('G_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('S_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('BEATS','C5','1');
INSERT INTO LABEL_TYPE VALUES ('AD','C5','1');
INSERT INTO LABEL_TYPE VALUES ('MS','C5','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R2','GENDER');
INSERT INTO OBJECT_ROLE VALUES ('R3','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R4','PERIOD');
INSERT INTO OBJECT_ROLE VALUES ('R5','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R6','YEAR');
INSERT INTO OBJECT_ROLE VALUES ('R7','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R8','HEAR_RATE');
INSERT INTO OBJECT_ROLE VALUES ('R9','SPORT');
INSERT INTO OBJECT_ROLE VALUES ('R10','SPORT');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('R1','NORMAL','P_GENDER','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','P_GENDER','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','P_PERIOD','1');
INSERT INTO ROLE VALUES ('R4','NORMAL','P_PERIOD','2');
INSERT INTO ROLE VALUES ('R5','NORMAL','P_YEAR','1');
INSERT INTO ROLE VALUES ('R6','NORMAL','P_YEAR','2');
INSERT INTO ROLE VALUES ('R7','NORMAL','P_HEAR','1');
INSERT INTO ROLE VALUES ('R8','NORMAL','P_HEAR','2');
INSERT INTO ROLE VALUES ('R9','NORMAL','P_SPORT','1');
INSERT INTO ROLE VALUES ('R10','NORMAL','P_SPORT','2');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('P_GENDER','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_PERIOD','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_YEAR','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_HEAR','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_SPORT','SPORT');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('P_REF','PERSON');
INSERT INTO REFERENCE VALUES ('G_REF','PERSON');
INSERT INTO REFERENCE VALUES ('PR_REF','PERSON');
INSERT INTO REFERENCE VALUES ('Y_REF','PERSON');
INSERT INTO REFERENCE VALUES ('H_REF','PERSON');
INSERT INTO REFERENCE VALUES ('S_REF','SPORT');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('P_GENDER','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_PERIOD','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_YEAR','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_HEAR','F_NORMAL');
```

```
INSERT INTO RELATION_TYPE VALUES ('P_SPORT','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('G_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('PR_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('Y_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('H_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('S_REF','REF');

COMMIT WORK;
```

3.6 ข้อมูลของ Equality Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-6 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE EMPLOYEE;

CREATE TABLE EMPLOYEE
        (EMPLOYEE       CHAR(20),
         GENDER                 CHAR(1),
         PARKING_BAY    CHAR(10),
         MONEY          CHAR(10));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','EXCLUSION','Y');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');
INSERT INTO CONST_ROLE VALUES ('C1','RG2');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('R1','RG1');
INSERT INTO ROLE_GROUP ('R5','RG2');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('EMPLOYEE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('E_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('G_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('PARKING_BAY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('P_CODE','LABEL');
INSERT INTO QBJECT_TYPE VALUES ('MONEY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('M_NR','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','E_PARKING','R1','E_NAME','EMPLOYEE');
INSERT INTO UNIQ_REF VALUES
('PARKING_BAY','E_PARKING','R2','P_CODE','PARKING_BAY');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','E_GENDER','R3','E_NAME','EMPLOYEE');
INSERT INTO UNIQ_REF VALUES ('GENDER','E_GENDER','R4','G_CODE','GENDER');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','E_MONEY','R5','E_NAME','EMPLOYEE');
INSERT INTO UNIQ_REF VALUES ('MONEY','E_MONEY','R6','M_NR','MONEY');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('E_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('G_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('P_NAME','C10','1');
INSERT INTO LABEL_TYPE VALUES ('M_NR','C10','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R2','PARKING_BAY');
INSERT INTO OBJECT_ROLE VALUES ('R3','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R4','GENDER');
```

```
INSERT INTO OBJECT_ROLE VALUES ('R5','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R6','MONEY');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('R1','NORMAL','E_PARKING','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','E_PARKING','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','E_GENDER','1');
INSERT INTO ROLE VALUES ('R4','NORMAL','E_GENDER','2');
INSERT INTO ROLE VALUES ('R5','NORMAL','E_MONEY','1');
INSERT INTO ROLE VALUES ('R6','NORMAL','E_MONEY','2');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('E_PARKING','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('E_GENDER','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('E_MONEY','EMPLOYEE');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('E_REF','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('P_REF','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('G_REF','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('M_REF','EMPLOYEE');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('E_PARKING','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('E_GENDER','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('E_MONEY','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('E_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('P_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('G_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('M_REF','REF');

COMMIT WORK;
```

3.7 ข้อมูลของ Equality Constraint

ข้อมูลเป็นข้อมูลที่แปลงมาจากรูปที่ 9-7 โดยมีรายละเอียดดังต่อไปนี้

```
DROP TABLE PERSON;
DROP TABLE P_OWN_CAR;
DROP TABLE P_DRIVE_CAR;

CREATE TABLE PERSON
        ((PERSON        CHAR(20),
          GENDER        CHAR(1));

CREATE TABLE P_OWN_CAR
        ((PERSON        CHAR(20),
          CAR           CHAR(10));

CREATE TABLE P_DRIVE_CAR
        ((PERSON        CHAR(20),
          CAR           CHAR(10));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','SUBSET','Y');

DELETE FROM SUBSET_CONST;
INSERT INTO SUBSET_CONST VALUES ('C1','RG2');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');
INSERT INTO CONST_ROLE VALUES ('C1','RG2');
```

```
DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP ('R3','RG1');
INSERT INTO ROLE_GROUP ('R5','RG2');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('PERSON','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('P_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('G_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('CAR','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('REG#','LABEL');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PERSON','P_GENDER','R1','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('GENDER','P_GENDER','R2','G_CODE','GENDER');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_OWN_CAR','R3','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('CAR','P_OWN_CAR','R4','REG#','CAR');
INSERT INTO UNIQ_REF VALUES ('PERSON','P_DRV_CAR','R5','P_NAME','PERSON');
INSERT INTO UNIQ_REF VALUES ('CAR','P_DRV_CAR','R6','REG#','CAR');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('P_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('G_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('REG#','C10','1');

DELETE          FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R2','GENDER');
INSERT INTO OBJECT_ROLE VALUES ('R3','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R4','CAR');
INSERT INTO OBJECT_ROLE VALUES ('R5','PERSON');
INSERT INTO OBJECT_ROLE VALUES ('R6','CAR');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('R1','NORMAL','P_GENDER','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','P_GENDER','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','P_OWN_CAR','1');
INSERT INTO ROLE VALUES ('R4','NORMAL','P_OWN_CAR','2');
INSERT INTO ROLE VALUES ('R5','NORMAL','P_DRIVE_CAR','1');
INSERT INTO ROLE VALUES ('R6','NORMAL','P_DRIVE_CAR','2');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('P_GENDER','PERSON');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_OWN_CAR','P_OWN_CAR');
INSERT INTO NORMAL_FACTTYPE VALUES ('P_DRV_CAR','P_DRIVE_CAR');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('P_REF','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('G_REF','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('C_REF','EMPLOYEE');

DELETE FROM RELATION_TYPE;
INSERT INTO RELATION_TYPE VALUES ('P_GENDER','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_OWN_CAR','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_DRV_CAR','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('P_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('G_REF','REF');
INSERT INTO RELATION_TYPE VALUES ('C_REF','REF');

COMMIT WORK;
```

**4. ข้อมูลที่ใช้ทดสอบระบบทั้งหมด**

ในส่วนข้อมูลที่ใช้ทดสอบระบบทั้งหมดจะมีข้อมูลที่ใช้ทดสอบชุดใหญ่เพียงชุดเดียว ซึ่งก็คือข้อมูลจากแบบจำลองข้อมูลในแอมในรูปที่ 11-1 โดยข้อมูลเหล่านี้เป็นข้อมูลที่ได้แปลงให้อยู่ในรูปแบบที่สามารถเก็บในเมต้าสกีม่าที่อยู่ในรูปของตาราง ดังนั้นข้อมูลจึงค่อนข้างมาก โดยข้อมูลจะเก็บไว้ในไฟล์ PIC11.SQL

```
DROP TABLE EMPLOYEE;
DROP TABLE MANAGE_CAR;
DROP TABLE MANAGER;
DROP TABLE MARRIED_EMP;
DROP TABLE DEPARTMENT;
DROP TABLE BRANCH;
DROP TABLE SELL;

CREATE TABLE EMPLOYEE
                (EMP#           CHAR(10),
                 EMP_NAME       CHAR(20),
                 GENDER                 CHAR(1),
                 DEGREE                 CHAR(10),
                 PHONE          CHAR(10),
                 SALARY                 CHAR(10),
                 DEPARTMENT     CHAR(20),
                 BRANCH                 CHAR(20),
                 BONUS          CHAR(10),
                 HOLIDAY        CHAR(10));

CREATE TABLE MANAGE_CAR
                (EMP#           CHAR(10),
                 REG#           CHAR(10));

CREATE TABLE MANAGER
                (EMP#           CHAR(10),
                 LICENCE#       CHAR(10));

CREATE TABLE MARRIED_EMP
                (EMP#           CHAR(10),
                 YEAR           CHAR(5));

CREATE TABLE DEPARTMENT
                (DEPT_NAME      CHAR(20),
                 HEAD_OF_DEPT CHAR(10),
                 PERS_COUNT     CHAR(5),
                 BUDGET                 CHAR(10));

CREATE TABLE BRANCH
                (BRANCH_NAMECHAR(20),
                 LOCATE                 CHAR(20));

CREATE TABLE SELL
                (BRANCH_NAMECHAR(20),
                 DRIVE_KIND     CHAR(2),
                 YEAR           CHAR(5),
                 QTY            CHAR(5));

DELETE FROM CONSTRAINT;
INSERT INTO CONSTRAINT VALUES ('C1','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C2','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C3','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C4','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C5','UNIQUE','Y');
```

```
INSERT INTO CONSTRAINT VALUES ('C6','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C7','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C8','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C9','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C10','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C11','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C12','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C13','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C14','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C15','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C16','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C17','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C18','UNIQUE','Y');
INSERT INTO CONSTRAINT VALUES ('C19','MENDATORRY','Y');
INSERT INTO CONSTRAINT VALUES ('C20','MENDATORRY','Y');
INSERT INTO CONSTRAINT VALUES ('C21','MENDATORRY','Y');
INSERT INTO CONSTRAINT VALUES ('C22','MENDATORRY','Y');
INSERT INTO CONSTRAINT VALUES ('C23','MENDATORRY','Y');
INSERT INTO CONSTRAINT VALUES ('C24','EXCLUSION','Y');
INSERT INTO CONSTRAINT VALUES ('C25','SUBSET','Y');
INSERT INTO CONSTRAINT VALUES ('C26','EQUALITY','Y');
INSERT INTO CONSTRAINT VALUES ('C27','MEMBER','Y');
INSERT INTO CONSTRAINT VALUES ('C28','RANGE','Y');
INSERT INTO CONSTRAINT VALUES ('C29','OCCURRENCY','Y');

DELETE FROM ROLE_GROUP;
INSERT INTO ROLE_GROUP VALUES ('R2','RG1');
INSERT INTO ROLE_GROUP VALUES ('R3','RG2');
INSERT INTO ROLE_GROUP VALUES ('R6','RG3');
INSERT INTO ROLE_GROUP VALUES ('R7','RG4');
INSERT INTO ROLE_GROUP VALUES ('R9','RG5');
INSERT INTO ROLE_GROUP VALUES ('R12','RG6');
INSERT INTO ROLE_GROUP VALUES ('R13','RG7');
INSERT INTO ROLE_GROUP VALUES ('R16','RG8');
INSERT INTO ROLE_GROUP VALUES ('R17','RG9');
INSERT INTO ROLE_GROUP VALUES ('R20','RG10');
INSERT INTO ROLE_GROUP VALUES ('R22','RG11');
INSERT INTO ROLE_GROUP VALUES ('R21','RG12');
INSERT INTO ROLE_GROUP VALUES ('R24','RG13');
INSERT INTO ROLE_GROUP VALUES ('R25','RG14');
INSERT INTO ROLE_GROUP VALUES ('R28','RG15');
INSERT INTO ROLE_GROUP VALUES ('R30','RG16');
INSERT INTO ROLE_GROUP VALUES ('R31','RG17');
INSERT INTO ROLE_GROUP VALUES ('R32','RG18');
INSERT INTO ROLE_GROUP VALUES ('R33','RG19');
INSERT INTO ROLE_GROUP VALUES ('R34','RG20');
INSERT INTO ROLE_GROUP VALUES ('R5','RG21');
INSERT INTO ROLE_GROUP VALUES ('R13','RG22');

DELETE FROM CONST_ROLE;
INSERT INTO CONST_ROLE VALUES ('C1','RG1');
INSERT INTO CONST_ROLE VALUES ('C2','RG2');
INSERT INTO CONST_ROLE VALUES ('C3','RG3');
INSERT INTO CONST_ROLE VALUES ('C4','RG5');
INSERT INTO CONST_ROLE VALUES ('C5','RG6');
INSERT INTO CONST_ROLE VALUES ('C6','RG7');
INSERT INTO CONST_ROLE VALUES ('C7','RG8');
INSERT INTO CONST_ROLE VALUES ('C8','RG9');
INSERT INTO CONST_ROLE VALUES ('C9','RG10');
INSERT INTO CONST_ROLE VALUES ('C11','RG11');
INSERT INTO CONST_ROLE VALUES ('C11','RG12');
INSERT INTO CONST_ROLE VALUES ('C12','RG14');
INSERT INTO CONST_ROLE VALUES ('C13','RG13');
INSERT INTO CONST_ROLE VALUES ('C14','RG15');
```

```
INSERT INTO CONST_ROLE VALUES ('C15','RG16');
INSERT INTO CONST_ROLE VALUES ('C16','RG17');
INSERT INTO CONST_ROLE VALUES ('C16','RG18');
INSERT INTO CONST_ROLE VALUES ('C16','RG19');
INSERT INTO CONST_ROLE VALUES ('C17','RG20');
INSERT INTO CONST_ROLE VALUES ('C18','RG4');
INSERT INTO CONST_ROLE VALUES ('C19','RG4');
INSERT INTO CONST_ROLE VALUES ('C20','RG9');
INSERT INTO CONST_ROLE VALUES ('C21','RG15');
INSERT INTO CONST_ROLE VALUES ('C22','RG11');
INSERT INTO CONST_ROLE VALUES ('C23','RG14');
INSERT INTO CONST_ROLE VALUES ('C24','RG5');
INSERT INTO CONST_ROLE VALUES ('C24','RG22');
INSERT INTO CONST_ROLE VALUES ('C25','RG21');
INSERT INTO CONST_ROLE VALUES ('C25','RG4');
INSERT INTO CONST_ROLE VALUES ('C26','RG11');
INSERT INTO CONST_ROLE VALUES ('C26','RG13');
INSERT INTO CONST_ROLE VALUES ('C29','RG18');

DELETE FROM OBJECT_CONST;
INSERT INTO OBJECT_CONST VALUES ('C27','DRIVE_KIND');
INSERT INTO OBJECT_CONST VALUES ('C28','QTY');

DELETE FROM RANGE_CONST;
INSERT INTO RANGE_CONST VALUES ('C28','1','200');

DELETE FROM MEMBERSHIP_CONST;
INSERT INTO MEMBERSHIP_CONST VALUES ('C27','HD');
INSERT INTO MEMBERSHIP_CONST VALUES ('C27','FD');

DELETE FROM FREQ_CONST;
INSERT INTO FREQ_CONST VALUES ('C29','2','2');

DELETE FROM SUBSET_CONST;
INSERT INTO SUBSET_CONST VALUES ('C25','RG21');

DELETE FROM OBJECT_TYPE;
INSERT INTO OBJECT_TYPE VALUES ('EMPLOYEE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('EMP#','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('EMP_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('GENDER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('G_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('DEGREE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('D_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('PHONE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('PHONE#','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('SALARY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('SAL$','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('DEPARTMENT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('DEPT_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('BRANCH','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('B_NAME','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('BONUS','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('BONUS$','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('HOLIDAY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('H_DATE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('MANAGER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('MARRIED_EMP','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('COMP_CAR','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('REG#','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('DRIVER','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('LICENCE#','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('YEAR','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('Y_AD','LABEL');
```

```
INSERT INTO OBJECT_TYPE VALUES ('PERS_COUNT','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('PERS_NR','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('BUDGET','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('BUDGET$','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('LOCATE','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('CITY','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('DRIVE_KIND','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('DR_CODE','LABEL');
INSERT INTO OBJECT_TYPE VALUES ('QTY','ENTITY');
INSERT INTO OBJECT_TYPE VALUES ('QTY_NR','LABEL');

DELETE FROM LABEL_TYPE;
INSERT INTO LABEL_TYPE VALUES ('EMP#','C10','1');
INSERT INTO LABEL_TYPE VALUES ('EMP_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('G_CODE','C1','1');
INSERT INTO LABEL_TYPE VALUES ('D_CODE','C10','1');
INSERT INTO LABEL_TYPE VALUES ('PHONE#','C10','1');
INSERT INTO LABEL_TYPE VALUES ('SAL$','C10','1');
INSERT INTO LABEL_TYPE VALUES ('DEPT_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('B_NAME','C20','1');
INSERT INTO LABEL_TYPE VALUES ('BONUS$','C10','1');
INSERT INTO LABEL_TYPE VALUES ('H_DATE','C10','1');
INSERT INTO LABEL_TYPE VALUES ('REG#','C10','1');
INSERT INTO LABEL_TYPE VALUES ('LICENCE#','C10','1');
INSERT INTO LABEL_TYPE VALUES ('Y_AD','C5','1');
INSERT INTO LABEL_TYPE VALUES ('PERS_NR','C5','1');
INSERT INTO LABEL_TYPE VALUES ('BUDGET$','C10','1');
INSERT INTO LABEL_TYPE VALUES ('CITY','C20','1');
INSERT INTO LABEL_TYPE VALUES ('DR_CODE','C2','1');
INSERT INTO LABEL_TYPE VALUES ('QTY_NR','C5','1');

DELETE FROM SUBTYPE;
INSERT INTO SUBTYPE VALUES ('MANAGER','EMPLOYEE');
INSERT INTO SUBTYPE VALUES ('MARRIED_EMP','EMPLOYEE');

DELETE FROM ROLE;
INSERT INTO ROLE VALUES ('R1','NORMAL','F1','1');
INSERT INTO ROLE VALUES ('R2','NORMAL','F1','2');
INSERT INTO ROLE VALUES ('R3','NORMAL','F2','1');
INSERT INTO ROLE VALUES ('R4','NORMAL','F2','2');
INSERT INTO ROLE VALUES ('R5','NORMAL','F3','1');
INSERT INTO ROLE VALUES ('R6','NORMAL','F3','2');
INSERT INTO ROLE VALUES ('R7','NORMAL','F4','1');
INSERT INTO ROLE VALUES ('R8','NORMAL','F4','2');
INSERT INTO ROLE VALUES ('R9','NORMAL','F5','1');
INSERT INTO ROLE VALUES ('R10','NORMAL','F5','2');
INSERT INTO ROLE VALUES ('R11','NORMAL','F6','1');
INSERT INTO ROLE VALUES ('R12','NORMAL','F6','2');
INSERT INTO ROLE VALUES ('R13','NORMAL','F7','1');
INSERT INTO ROLE VALUES ('R14','NORMAL','F7','2');
INSERT INTO ROLE VALUES ('R15','NORMAL','F8','1');
INSERT INTO ROLE VALUES ('R16','NORMAL','F8','2');
INSERT INTO ROLE VALUES ('R17','NORMAL','F9','1');
INSERT INTO ROLE VALUES ('R18','NORMAL','F9','2');
INSERT INTO ROLE VALUES ('R19','NORMAL','F10','1');
INSERT INTO ROLE VALUES ('R20','NORMAL','F10','2');
INSERT INTO ROLE VALUES ('R21','NORMAL','F11','1');
INSERT INTO ROLE VALUES ('R22','NORMAL','F11','2');
INSERT INTO ROLE VALUES ('R23','NORMAL','F12','1');
INSERT INTO ROLE VALUES ('R24','NORMAL','F12','2');
INSERT INTO ROLE VALUES ('R25','NORMAL','F13','1');
INSERT INTO ROLE VALUES ('R26','NORMAL','F13','2');
INSERT INTO ROLE VALUES ('R27','NORMAL','F14','1');
INSERT INTO ROLE VALUES ('R28','NORMAL','F14','2');
```

```
INSERT INTO ROLE VALUES ('R29','NORMAL','F15','1');
INSERT INTO ROLE VALUES ('R30','NORMAL','F15','2');
INSERT INTO ROLE VALUES ('R31','NORMAL','F16','1');
INSERT INTO ROLE VALUES ('R32','NORMAL','F16','2');
INSERT INTO ROLE VALUES ('R33','NORMAL','F16','3');
INSERT INTO ROLE VALUES ('R34','NESTING','F17','1');
INSERT INTO ROLE VALUES ('R35','NORMAL','F17','2');

INSERT INTO ROLE VALUES ('RF1','NORMAL','RT1','1');
INSERT INTO ROLE VALUES ('RF2','NORMAL','RT1','1');
INSERT INTO ROLE VALUES ('RF3','NORMAL','RT2','1');
INSERT INTO ROLE VALUES ('RF4','NORMAL','RT2','1');
INSERT INTO ROLE VALUES ('RF5','NORMAL','RT3','1');
INSERT INTO ROLE VALUES ('RF6','NORMAL','RT3','1');
INSERT INTO ROLE VALUES ('RF7','NORMAL','RT4','1');
INSERT INTO ROLE VALUES ('RF8','NORMAL','RT4','1');
INSERT INTO ROLE VALUES ('RF9','NORMAL','RT5','1');
INSERT INTO ROLE VALUES ('RF10','NORMAL','RT5','1');
INSERT INTO ROLE VALUES ('RF11','NORMAL','RT6','1');
INSERT INTO ROLE VALUES ('RF12','NORMAL','RT6','1');
INSERT INTO ROLE VALUES ('RF13','NORMAL','RT7','1');
INSERT INTO ROLE VALUES ('RF14','NORMAL','RT7','1');
INSERT INTO ROLE VALUES ('RF15','NORMAL','RT8','1');
INSERT INTO ROLE VALUES ('RF16','NORMAL','RT8','1');
INSERT INTO ROLE VALUES ('RF17','NORMAL','RT9','1');
INSERT INTO ROLE VALUES ('RF18','NORMAL','RT9','1');
INSERT INTO ROLE VALUES ('RF19','NORMAL','RT10','1');
INSERT INTO ROLE VALUES ('RF20','NORMAL','RT10','1');
INSERT INTO ROLE VALUES ('RF21','NORMAL','RT11','1');
INSERT INTO ROLE VALUES ('RF22','NORMAL','RT11','1');
INSERT INTO ROLE VALUES ('RF23','NORMAL','RT12','1');
INSERT INTO ROLE VALUES ('RF24','NORMAL','RT12','1');
INSERT INTO ROLE VALUES ('RF25','NORMAL','RT13','1');
INSERT INTO ROLE VALUES ('RF26','NORMAL','RT13','1');
INSERT INTO ROLE VALUES ('RF27','NORMAL','RT14','1');
INSERT INTO ROLE VALUES ('RF28','NORMAL','RT14','1');
INSERT INTO ROLE VALUES ('RF29','NORMAL','RT15','1');
INSERT INTO ROLE VALUES ('RF30','NORMAL','RT15','1');
INSERT INTO ROLE VALUES ('RF31','NORMAL','RT16','1');
INSERT INTO ROLE VALUES ('RF32','NORMAL','RT16','1');
INSERT INTO ROLE VALUES ('RF33','NORMAL','RT17','1');
INSERT INTO ROLE VALUES ('RF34','NORMAL','RT17','1');
INSERT INTO ROLE VALUES ('RF35','NORMAL','RT18','1');
INSERT INTO ROLE VALUES ('RF36','NORMAL','RT18','1');

DELETE FROM OBJECT_ROLE;
INSERT INTO OBJECT_ROLE VALUES ('R1','PERS_COUNT');
INSERT INTO OBJECT_ROLE VALUES ('R2','DEPARTMENT');
INSERT INTO OBJECT_ROLE VALUES ('R3','DEPARTMENT');
INSERT INTO OBJECT_ROLE VALUES ('R4','BUDGET');
INSERT INTO OBJECT_ROLE VALUES ('R5','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R6','DEPARTMENT');
INSERT INTO OBJECT_ROLE VALUES ('R7','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R8','DEPARTMENT');
INSERT INTO OBJECT_ROLE VALUES ('R9','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R10','HOLIDAY');
INSERT INTO OBJECT_ROLE VALUES ('R11','SALARY');
INSERT INTO OBJECT_ROLE VALUES ('R12','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R13','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R14','BONUS');
INSERT INTO OBJECT_ROLE VALUES ('R15','PHONE');
INSERT INTO OBJECT_ROLE VALUES ('R16','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R17','EMPLOYEE');
INSERT INTO OBJECT_ROLE VALUES ('R18','GENDER');
```

```
INSERT INTO RELATION_TYPE VALUES ('F6','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F7','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F8','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F9','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F10','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F11','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F12','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F13','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F14','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F15','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F16','F_NORMAL');
INSERT INTO RELATION_TYPE VALUES ('F17','F_NORMAL');

INSERT INTO RELATION_TYPE VALUES ('RT1','REF');
INSERT INTO RELATION_TYPE VALUES ('RT2','REF');
INSERT INTO RELATION_TYPE VALUES ('RT3','REF');
INSERT INTO RELATION_TYPE VALUES ('RT4','REF');
INSERT INTO RELATION_TYPE VALUES ('RT5','REF');
INSERT INTO RELATION_TYPE VALUES ('RT6','REF');
INSERT INTO RELATION_TYPE VALUES ('RT7','REF');
INSERT INTO RELATION_TYPE VALUES ('RT8','REF');
INSERT INTO RELATION_TYPE VALUES ('RT9','REF');
INSERT INTO RELATION_TYPE VALUES ('RT10','REF');
INSERT INTO RELATION_TYPE VALUES ('RT11','REF');
INSERT INTO RELATION_TYPE VALUES ('RT12','REF');
INSERT INTO RELATION_TYPE VALUES ('RT13','REF');
INSERT INTO RELATION_TYPE VALUES ('RT14','REF');
INSERT INTO RELATION_TYPE VALUES ('RT15','REF');
INSERT INTO RELATION_TYPE VALUES ('RT16','REF');
INSERT INTO RELATION_TYPE VALUES ('RT17','REF');

DELETE FROM UNIQ_REF;
INSERT INTO UNIQ_REF VALUES ('PERS_COUNT','F1','R1','PERS_NR','PERS_COUNT');
INSERT INTO UNIQ_REF VALUES ('DEPARTMENT','F1','R2','DEPT_NAME','DEPT_NAME');
INSERT INTO UNIQ_REF VALUES ('DEPARTMENT','F2','R3','DEPT_NAME','DEPT_NAME');
INSERT INTO UNIQ_REF VALUES ('BUDGET','F2','R4','BUDGET$','BUDGET');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F3','R5','EMP#','HEAD_OF_DEPT');
INSERT INTO UNIQ_REF VALUES ('DEPARTMENT','F3','R6','DEPT_NAME','DEPT_NAME');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F4','R7','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('DEPARTMENT','F4','R8','DEPT_NAME','DEPT_NAME');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F5','R9','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('HOLIDAY','F5','R10','H_DATE','HOLIDAY');
INSERT INTO UNIQ_REF VALUES ('SALARY','F6','R11','SALARY$','SALARY');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F6','R12','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F7','R13','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('BONUS','F7','R14','BONUS$','BONUS');
INSERT INTO UNIQ_REF VALUES ('PHONE','F8','R15','PHONE#','PHONE');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F8','R16','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F9','R17','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('GENDER','F9','R18','G_CODE','GENDER');
INSERT INTO UNIQ_REF VALUES ('DEGREE','F10','R19','D_CODE','DEGREE');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F10','R20','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('COMP_CAR','F11','R21','REG#','REG#');
INSERT INTO UNIQ_REF VALUES ('MANAGER','F11','R22','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('DRIVER','F12','R23','LICENCE#','LICENCE#');
INSERT INTO UNIQ_REF VALUES ('MANAGER','F12','R24','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('MARRIED_EMP','F13','R25','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('YEAR','F13','R26','Y_AD','YEAR');
INSERT INTO UNIQ_REF VALUES ('BRANCH','F14','R27','B_NAME','BRANCH_NAME');
INSERT INTO UNIQ_REF VALUES ('EMPLOYEE','F14','R28','EMP#','EMP#');
INSERT INTO UNIQ_REF VALUES ('LOCATE','F15','R29','CITY_NAME','LOCATE');
INSERT INTO UNIQ_REF VALUES ('BRANCH','F15','R30','B_NAME','BRANCH_NAME');
INSERT INTO UNIQ_REF VALUES ('BRANCH','F16','R31','B_NAME','BRANCH_NAME');
INSERT INTO UNIQ_REF VALUES ('DRIVE_KIND','F16','R32','DR_CODE','DRIVE_KIND');
```

```
INSERT INTO UNIQ_REF VALUES ('YEAR','F16','R33','Y_AD','YEAR');
INSERT INTO UNIQ_REF VALUES ('F16','F17','R34',NULL,NULL);
INSERT INTO UNIQ_REF VALUES ('QTY','F17','R35','QTY_NR','QTY');

DELETE FROM NORMAL_FACTTYPE;
INSERT INTO NORMAL_FACTTYPE VALUES ('F1','DEPARTMENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('F2','DEPARTMENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('F3','DEPARTMENT');
INSERT INTO NORMAL_FACTTYPE VALUES ('F4','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F5','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F6','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F7','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F8','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F9','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F10','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F11','MANAGE_CAR');
INSERT INTO NORMAL_FACTTYPE VALUES ('F12','MANAGER');
INSERT INTO NORMAL_FACTTYPE VALUES ('F13','MARRIED_EMP');
INSERT INTO NORMAL_FACTTYPE VALUES ('F14','EMPLOYEE');
INSERT INTO NORMAL_FACTTYPE VALUES ('F15','BRANCH');
INSERT INTO NORMAL_FACTTYPE VALUES ('F16','SELL');
INSERT INTO NORMAL_FACTTYPE VALUES ('F17','SELL');

DELETE FROM REFERENCE;
INSERT INTO REFERENCE VALUES ('RT1','DEPARTMENT');
INSERT INTO REFERENCE VALUES ('RT2','DEPARTMENT');
INSERT INTO REFERENCE VALUES ('RT3','DEPARTMENT');
INSERT INTO REFERENCE VALUES ('RT4','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT5','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT6','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT7','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT8','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT9','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT10','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT11','EMPLOYEE');
INSERT INTO REFERENCE VALUES ('RT12','MANAGE_CAR');
INSERT INTO REFERENCE VALUES ('RT13','MANAGER');
INSERT INTO REFERENCE VALUES ('RT14','MARRIED_EMP');
INSERT INTO REFERENCE VALUES ('RT15','SELL');
INSERT INTO REFERENCE VALUES ('RT16','BRANCH');
INSERT INTO REFERENCE VALUES ('RT17','BRANCH');
INSERT INTO REFERENCE VALUES ('RT18','SELL');

DELETE FROM PRIMARY_KEY;
INSERT INTO PRIMARY_KEY VALUES ('EMPLOYEE','EMP#');
INSERT INTO PRIMARY_KEY VALUES ('MANAGE_CAR','EMP#');
INSERT INTO PRIMARY_KEY VALUES ('MANAGER','EMP#');
INSERT INTO PRIMARY_KEY VALUES ('MARRIED_EMP','EMP#');
INSERT INTO PRIMARY_KEY VALUES ('DEPARTMENT','DEPT_NAME');
INSERT INTO PRIMARY_KEY VALUES ('BRANCH','BRANCE_NAME');
INSERT INTO PRIMARY_KEY VALUES ('SELL','BRANCE_NAME');
INSERT INTO PRIMARY_KEY VALUES ('SELL','DRIVE_KIND');
INSERT INTO PRIMARY_KEY VALUES ('SELL','QTY');

COMMIT WORK;
```

# ประวัติผู้เขียน

| | |
|---|---|
| ชื่อผู้เขียน | นายธนา หงษ์สุวรรณ |
| วันเดือนปีเกิด | วันที่ 16 ตุลาคม พ.ศ. 2509 |
| สถานที่เกิด | จังหวัดกรุงเทพมหานคร |
| บิดาชื่อ | นายชวน หงษ์สุวรรณ |
| มารดาชื่อ | นางอรพรรณ หงษ์สุวรรณ |
| วุฒิการศึกษาระดับปริญญาตรี | วิศวกรรมศาสตร์บัณฑิต<br>สาขาวิชาวิศวกรรมคอมพิวเตอร์<br>ภาควิชาวิศวกรรมคอมพิวเตอร์<br>สถาบันเทคโนโลยีพระจอมเกล้า<br>เจ้าคุณทหารลาดกระบัง<br>ปีการศึกษา 2530 |
| ผลงานการศึกษาระดับปริญญาตรี | เป็นผู้พัฒนาระบบฐานข้อมูลวิทยานิพนธ์<br>ซึ่งได้นำแสดงในงานพระจอมเกล้า<br>ลาดกระบังนิทรรศน์ 30 ซึ่งปัจจุบัน<br>ยังใช้งานอยู่ที่ห้องสมุดคณะวิศวกรรมศาสตร์<br>และมีวิทยานิพนธ์เรื่อง การรู้จำตัวอักษร<br>ภาษาไทยในระบบไมโครคอมพิวเตอร์ |
| ผลงานทางวิชาการที่ได้รับการตีพิมพ์ | การจัดการกฎชนิดรีเดอร์ซีฟในฐานความรู้<br>ขนาดใหญ่มาก ในการประชุมทางวิชาการ<br>สถิติประยุกต์ ครั้งที่ 8 ของสถาบันบัณฑิต<br>พัฒนบริหารศาสตร์ 24-25 พฤษภาคม 2533 |
| ประสบการณ์การทำงาน | - ผู้ช่วยวิจัยภาควิชาวิศวกรรมคอมพิวเตอร์<br>คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยี<br>พระจอมเกล้าเจ้าคุณทหารลวดกระบัง<br>ปี 2531-2532<br>- หัวหน้าแผนกสารสนเทศ คณะวิศวกรรม<br>ศาสตร์ สถาบันเทคโนโลยีพ่ระจอมเกล้า<br>ประเทศไทย เมื่อปี 2532 ถึง 2535<br>- บรรณาธิการนิตยสารคอมพิวเตอร์<br>ปี 2535-2537 |