แบบจำลองฐานข้อมูลเชิงแนวคิดทีโอโดไนแอมและการแปลงรูปเป็นฐานข้อมูลเชิงวัตถุ

THE TOONIAM CONCEPTUAL SCHEMA MODEL AND A TRANSFORMATION TO AN OBJECT ORIENTED DATABASE SCHEMA

บัณฑิต พัสยา

BUNDIT PASAYA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-376-5

แบบจำลองฐานข้อมูลเชิงแนวคิดทีโอโอไนแอมและการแปลงรูปเป็นฐานข้อมูลเชิงวัตถุ

THE TOONIAM CONCEPTUAL SCHEMA MODEL AND A TRANSFORMATION TO AN OBJECT ORIENTED DATABASE SCHEMA

บัณฑิต พัสยา

BUNDIT PASAYA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-376-5

THE TOONIAM CONCEPTUAL SCHEMA MODEL AND A TRANSFORMATION TO AN OBJECT ORIENTED DATABASE SCHEMA

BUNDIT PASAYA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2001

ISBN 974-648-376-5

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ แบบจำลองฐานข้อมูลเชิงแนวคิดทีโอโอไนแอมและการแปลงรูปเป็นฐานข้อมูลเชิงวัตถุ

THE TOONIAM CONCEPTUAL SCHEMA MODEL AND A TRANSFORMATION TO AN OBJECT ORIENTED DATABASE SCHEMA

ชื่อนักศึกษา นายบัณฑิต พัสยา

รหัสประจำตัว 38621212

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.ดร.ศุภมิตร จิตตะยโศธร

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.บุญธีร์	เครือตราชู	
ดร.วรวัฒน์	ลิ้มโภคา	
ผศ.อภิเนตร	อูนากูล	
รศ.ดร.บุญวัฒน์	อัตชู	
รศ.ดร.ศุภมิตร	จิตตะยโศธร	

วัน/เดือนปี ที่สอบ 26 กรกฎาคม 2544 เวลา 10.00-12.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-404)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 15 **เดือน** ตุลาคม **พ.ศ.** 2544

หัวข้อวิทยานิพนธ์	แบบจำลองฐานข้อมูลเชิงแนวคิดทีโอโอไนแอมและการแปลงรูปเป็นฐานข้อมูลเชิงวัตถุ
นักศึกษา	นาย บัณฑิต พัสยา
รหัสประจำตัว	38621212
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร. ศุภมิตร จิตตะยโศธร

บทคัดย่อ

ไนแอมเป็นแบบจำลองข้อมูลระดับแนวคิดที่มีพื้นฐานมาจากภาษาธรรมชาติ โดยไนแอมเป็นภาษาที่มีรูปประโยคเป็น <ประธาน,กริยา , กรรม> ผู้ใช้งานสามารถนำไนแอมมาออกแบบฐานข้อมูล และแปลงรูปเป็นตารางในรูปของ 5NF ได้เลย แต่ไนแอมก็มีข้อจำกัดหลายประการ คือ ในปัจจุบันไนแอมไม่สามารถรองรับหลักการเชิงวัตถุบางประการและข้อมูลเชิงเวลาได้ ดังนั้นหากต้องการออกแบบฐานข้อมูลเชิงเวลาหรือฐานข้อมูลเชิงวัตถุโดยใช้ไนแอม จำเป็นต้องเพิ่มขีดความสามารถของไนแอมให้สามารถรองรับหลักการทั้งสองได้

วิทยานิพนธ์นี้ได้นำเสนอโมเดลทีโอโอไนแอม ซึ่งเป็นโมเดลสำหรับออกแบบเชิงเวลาโดยเพิ่มสัญลักษณ์ของเวลาและสัญลักษณ์ของหลักการเชิงวัตถุเข้าไปในโมเดลของไนแอม นอกจากนี้ยังได้นำเสนอ อัลกอริธึมในการแปลงรูปจากแบบจำลองระดับแนวคิดทีโอโอไนแอมให้อยู่ในรูปของภาษานิยามเชิงวัตถุ เพื่อนำไปสร้างเป็นฐานข้อมูลเชิงเวลาบนระบบฐานข้อมูลเชิงวัตถุ

Thesis Title	The TOONIAM conceptual schema model and a transformation to an object oriented database schema
Student	Mr. Bundit Pasaya
Student ID.	38621212
Degree	Master of Engineering
Programme	Electrical Engineering
Year	2001
Thesis Advisor	Assoc.Prof. Dr. Suphamit Chitayasothorn

ABSTRACT

NIAM is the conceptual schema model that is based on a natural language. NIAM is composed of subject, verb, and object. NIAM uses a fact-oriented approach for the conceptual modeling. We can use NIAM for designing and translating information table from the first normal form into the fifth normal form. However, NIAM has some disadvantages. At present, NIAM cannot support some features of object oriented concept and temporal information concept. Therefore, we must extend NIAM structure in order to apply NIAM to these concepts.

This thesis proposes a new temporal schema model. We present the Temporal Object Oriented NIAM Schema Model (TOONIAM) for establishing a temporal database schema. We add some symbols for object oriented concept and temporal information into NIAM. In addition, we present an algorithm to translate TOONIAM conceptual schema to Object Definition Language in order to create a temporal database on an object oriented database.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้จะไม่สามารถสำเร็จลงได้เลยถ้าปราศจากความช่วยเหลือของบุคคลดังต่อไปนี้

ขอขอบพระคุณ บิดาและมารดาของผู้จัดทำที่เลี้ยงดูและให้กำลังใจในทุก ๆ เรื่องเสมอมา ไม่ว่าลูกจะอยู่ในสถานการณ์อย่างไร คุณพ่อและคุณแม่ให้ความหวังดีและตั้งใจดีเสมอ

ขอขอบพระคุณ รศ.ดร. ศุภมิตร จิตตะยโศธร ที่ให้แนะนำที่มีค่ายิ่ง สอนแนวคิดในการเรียน การดำรงชีวิตและให้โอกาสกับศิษย์ทุกคนเสมอ

ขอขอบพระคุณ รศ.ดร จเร สุรวัฒน์ปัญญา ที่ให้คำแนะนำและคำปรึกษาในทุก ๆ เรื่องจนวิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์

ขอขอบพระคุณ อ. ธนา หงษ์สุวรรณ ที่คอยชี้แนะเรื่องราวต่าง ๆ สอนทักษะในการเรียนการทำวิจัย ฝึกฝนให้ผู้จัดทำรู้จักเหตุและผลเสมอ

ขอขอบพระคุณ คุณ ฉวีวรรณ มงคลลาภกิจ ที่ให้กำลังใจและช่วยเหลือในหลาย ๆ ด้าน

ขอขอบพระคุณ คณาจารย์ และเจ้าหน้าที่ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ให้กำลังใจและเกื้อกูลในทุก ๆ เรื่อง

ขอขอบคุณ คุณดวงนภา ประเสริฐเมือง ที่ช่วยเหลือในการตรวจสอบวิทยานิพนธ์นี้

บัณฑิต พัสยา

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญตาราง (ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

แต่เดิมระบบฐานข้อมูลที่ใช้งานกันอย่างแพร่หลายคือระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database System : RDBMS)[6][8] ซึ่งใช้งานได้ดี มีความยืดหยุ่นและสะดวกรวดเร็ว ทำให้ผู้ใช้งานพึงพอใจเป็นอย่างยิ่ง โดยระบบฐานข้อมูลดังกล่าวมีโมเดลอยู่ในรูปของตาราง (Table) ซึ่งประกอบด้วย แถว (Tuple) และคอลัมน์ (Column) เมื่อจัดเก็บข้อมูลลงตารางเรียบร้อยแล้ว ผู้ใช้สามารถเรียกค้นข้อมูลที่อยู่ในตารางต่าง ๆ ได้โดยใช้ภาษา SQL (Structured Query Language)[8] ซึ่งเป็นภาษาที่มีรูปแบบที่ง่ายต่อความเข้าใจ นอกจากนี้ยังสามารถนำตารางต่าง ๆ มาวิเคราะห์ร่วมกัน (Join) เพื่อให้ได้คำตอบตามต้องการ แต่ระบบฐานข้อมูลเชิงสัมพันธ์ก็มีขีดจำกัดหลายประการคือ

ประการแรก คือปัญหาเรื่องความซับซ้อนของข้อมูลที่จัดเก็บ (Complex Object) กล่าวคือ ระบบฐานข้อมูลดังกล่าวจะจัดเก็บข้อมูลไว้ในตาราง หากผู้ใช้ต้องการเรียกค้นก็ส่งภาษา SQL มาเรียกค้นก็จะได้ข้อมูลที่ต้องการ แต่การกระทำดังกล่าวนี้จะได้ผลดีกับข้อมูลพื้นฐานที่เป็นตัวเลขและตัวอักษรเท่านั้น แต่หากเป็นข้อมูลที่มีความซับซ้อนเช่น ภาพนิ่ง เสียง หรือ ภาพเคลื่อนไหว หากจัดเก็บอยู่ในระบบฐานข้อมูล เมื่อส่งภาษา SQL มาเพื่อเรียกค้นก็จะได้ข้อมูลออกไปเหมือนกัน แต่ข้อมูลที่ได้ออกไปไม่ได้อยู่ในรูปที่จะทำงานได้เลย จำเป็นต้องผ่านกระบวณการบางอย่างเพื่อให้ข้อมูลนั้นทำงานได้ เช่น หากจัดเก็บข้อมูลวีดีโอ ไว้ในฐานข้อมูลเชิงสัมพันธ์โดยเก็บไว้ในชนิดของข้อมูล BLOB (Binary Large Object) เมื่อส่ง SQL มาเรียกค้น ระบบจะส่งข้อมูลที่เป็นไบนารีให้ ดังนั้นหากผู้ใช้ต้องการที่จะนำข้อมูลดังกล่าวไปแสดงผล ก็ต้องมีความรู้ในเรื่องของการแปลงไบนารีให้อยู่ในรูปของวีดีโอ ดังนั้นผู้ใช้ที่ไม่มีความรู้ทางด้านนี้ก็ไม่สามารถใช้งานข้อมูลนี้ได้เลย

ประการที่สอง คือปัญหาการจัดเก็บข้อมูลเพียงสถานะ(State) เดียว เท่านั้น กล่าวคือ เมื่อจัดเก็บข้อมูลไว้แล้ว หากในภายหลังมีการแก้ไข ก็จะนำข้อมูลเดิมที่มีอยู่มาแก้ไข ทำให้ไม่สามารถทราบได้เลยว่าก่อนการแก้ไข ข้อมูลเคยเป็นค่าใดมาก่อน หรือหากต้องการลบข้อมูลออก ระบบฐานข้อมูลก็จะลบข้อมูลนั้น ๆ ออกให้เลย เปรียบเสมือนว่าข้อมูลนั้นไม่เคยมีอยู่เลย ซึ่งโปรแกรมประยุกต์ต่าง ๆ ที่ทำงานบนระบบฐานข้อมูลในปัจจุบันโดยมากต้องการเรียกค้นสถานะเก่า ๆ ของข้อมูลทั้งสิ้น ดังนั้นหากจะนำระบบฐานข้อมูลเชิงสัมพันธ์เข้ามาจัดการกับข้อมูลที่แปรผันตามเวลานี้ก็ทำได้ เพียงแต่ผู้ใช้งานจำเป็นต้องจัดการเรื่องเวลาของข้อมูลเอง ซึ่งการจัดการนี้ไม่ใช่เรื่องง่าย ๆ

ในปัจจุบันข้อมูลไม่ได้อยู่ในรูปของข้อมูลพื้นฐานอีกแล้ว ข้อมูลที่ใช้ในระบบฐานข้อมูลมีความซับซ้อนมากขึ้น ดังนั้นการใช้ระบบฐานข้อมูลเชิงสัมพันธ์จึงไม่สะดวกอีกต่อไปแล้ว ด้วยเหตุ

นี้จึงได้มีการนำเอาหลักการเชิงวัตถุ (Object Oriented Concept)[6] มารวมกับเทคโนโลยีระบบฐานข้อมูล เพื่อสร้างเป็นฐานข้อมูลเชิงวัตถุ (Object Oriented Database System: OODB) ซึ่งหัวใจหลักของระบบฐานข้อมูลเชิงวัตถุนี้ก็คือ การมีรหัสเฉพาะในแต่ละออบเจ็กต์ (Object Identification : OID) และ การรวมกันระหว่างข้อมูลและการจัดการข้อมูล (Encapsulation) นอกจากนี้ยังคุณสมบัติอื่น ๆ ที่ช่วยให้จัดการกับข้อมูลที่ซับซ้อนได้ อาทิ การถ่ายทอดคุณสมบัติ (Inheritance) โพลีมอร์ฟีซึม (Polymorphism) เป็นต้น ซึ่งคุณสมบัติที่กล่าวมานี้ไม่มีในระบบฐานข้อมูลเชิงสัมพันธ์ และเชื่อกันว่าคุณสมบัติดังกล่าวสามารถทำให้ผู้ใช้งานระบบฐานข้อมูลสามารถจัดการกับข้อมูลในปัจจุบันได้สัมฤทธิ์ผลมากกว่าที่จะใช้ระบบฐานข้อมูลแบบเดิม แต่ระบบฐานข้อมูลเชิงวัตถุนี้ยังไม่สามารถจัดเก็บข้อมูลหลาย ๆ สถานะได้ ดังนั้นหากนำระบบฐานข้อมูลนี้มาใช้กับข้อมูลที่เปลี่ยนแปลงไปตามเวลา ผู้ใช้ก็ต้องจัดการกับเวลาของออบเจ็กต์เอง เช่นเดียวกับระบบฐานข้อมูลเชิงสัมพันธ์

สำหรับข้อมูลที่เปลี่ยนแปลงไปตามเวลานั้น โดยหลักการนั้นคือมีการจัดเก็บเวลาของข้อมูลลงในฐานข้อมูลด้วย โดยผู้ใช้งานไม่จำเป็นต้องจัดการกับเวลาของข้อมูล เมื่อผู้ใช้ต้องการเรียกค้นข้อมูลก็ไม่ต้องจัดการกับเวลาที่เพิ่มเข้าไป ระบบฐานข้อมูลจะจัดการให้ทั้งหมด ระบบฐานข้อมูลดังกล่าวมีชื่อว่า ระบบฐานข้อมูลเชิงเวลา (Temporal Database)[9] ระบบฐานข้อมูลนี้ได้มีการค้นคว้าและทำงานวิจัยมาเป็นเวลายาวนาน แต่ยังไม่มีผลิตภัณฑ์ฐานข้อมูลใดที่มีคุณสมบัติเป็นระบบฐานข้อมูลเชิงเวลาเต็มตัว ดังนั้นหากผู้ใช้งานต้องการที่จะนำผลิตภัณฑ์ที่มีอยู่ไปสร้างเป็นฐานข้อมูลเชิงเวลา ผู้ใช้จำเป็นต้องจัดการกับเวลาเอง

งานวิจัยนี้ได้นำเสนอโมเดลในการออกแบบฐานข้อมูลเชิงเวลาและสร้างเครื่องมือในการออกแบบระบบฐานข้อมูลเชิงเวลา โดยใช้ NIAM เป็นเครื่องมือในการออกแบบ แต่ใน NIAM ที่นำมาใช้นี้ได้มีการเพิ่มส่วนของเวลาเข้าไป นอกจากนี้ยังได้เพิ่มหลักการเชิงวัตถุเข้าไปด้วย เพื่อให้เครื่องมือใหม่นี้สามารถออกแบบฐานข้อมูลเชิงเวลาบนฐานข้อมูลเชิงวัตถุได้ และในงานวิจัยนี้ยังได้สร้างอัลกอริธึมในการแปลงรูปจากแบบจำลองที่สร้างขึ้นให้เป็นภาษานิยามเชิงวัตถุ (Object Definition Language)[1] เพื่อนำไปใช้กับระบบฐานข้อมูลเชิงวัตถุต่อไปอีกด้วย อนึ่งโมเดลใหม่นี้จะมีคุณสมบัติที่ดีกว่าการออกแบบฐานข้อมูลโดยใช้เครื่องมือในการออกแบบฐานข้อมูลเชิงวัตถุอื่นดังนี้

1. Class Diagram ของ UML (Unified Modeling Language) ไม่สามารถบอกได้ว่าส่วนใดเป็น Embedded Class และส่วนใดเป็น Class ที่ต้องสร้างใหม่

2. Class Diagram ไม่ได้นำเสนอ Function Dependency (FD) ซึ่งอาจทำให้แอตทริบิวต์ซ้ำซ้อนกันได้

3. Class Diagram ไม่สามารถรองรับหลักการเชิงเวลาได้

ในงานวิจัยนี้จะนำเสนอหลักการใหม่ในการออกแบบฐานข้อมูลที่สามารถแก้ปัญหาทั้งสามประการของ UML ได้

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ในการออกแบบฐานข้อมูลเชิงเวลานั้น ยังไม่มีการออกแบบใด ๆ ที่เป็นมาตรฐาน ดังนั้นในงานวิจัยนี้จึงได้นำการออกแบบฐานข้อมูล NIAM ที่ได้รับความเชื่อถือ และใช้กันอย่างแพร่หลายมาพัฒนาให้สามารถใช้งานกับข้อมูลที่แปรผันตามเวลาได้ นอกจากนี้ยังได้นำหลักการเชิงวัตถุผสานเข้าไปในเครื่องมือใหม่นี้ด้วย โดยเครื่องมือใหม่นี้จะสร้างรองรับข้อมูลที่ซับซ้อนได้ ซึ่งจะเป็นประโยชน์อย่างมากสำหรับผู้ที่จะออกแบบฐานข้อมูลเชิงเวลาบนฐานข้อมูลเชิงวัตถุ

1.3 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

1. หลักการเชิงวัตถุ
2. เครื่องมือออกแบบฐานข้อมูล NIAM
3. ฐานข้อมูลเชิงเวลา

1.4 ขอบเขตการวิจัย

งานวิจัยนี้นำเสนอหลักการและสร้างเครื่องมือใหม่ โดยเพิ่มสิ่งที่จำเป็นเข้าไปในโครงสร้างของ NIAM เพื่อออกแบบฐานข้อมูลเชิงเวลา ผู้วิจัยมิได้นำไปใช้งานกับระบบฐานข้อมูลเชิงเวลาจริง ๆ เนื่องจากในปัจจุบันระบบฐานข้อมูลเชิงเวลายังไม่มีจำหน่ายเลย ดังนั้นการทดสอบเครื่องมือดังกล่าวนั้นคือ การนำภาษานิยามเชิงวัตถุที่ได้จากการแปลงแบบจำลองไปทำงานบนระบบฐานข้อมูลเชิงวัตถุ เพื่อสร้างเป็นฐานข้อมูลเชิงเวลาเท่านั้น โดยการจัดการกับเวลาของข้อมูลนั้นจะมีเมธอดที่เขียนเก็บไว้ในออบเจ็กต์เชิงเวลาพื้นฐาน เมื่อต้องการให้ออบเจ็กต์ใดมีการจัดเก็บข้อมูลหลาย ๆ สถานะก็ให้ถ่ายทอดคุณสมบัติมาจากออบเจ็กต์เชิงเวลาพื้นฐานนั้น ๆ

1.5 ขั้นตอนของการศึกษา

1. ศึกษาระบบฐานข้อมูลเชิงวัตถุ
2. ศึกษาระบบฐานข้อมูลเชิงเวลา
3. ศึกษาการออกแบบฐานข้อมูล NIAM
4. สร้างโมเดลในการออกแบบฐานข้อมูลเชิงเวลา
5. สร้างเครื่องมือในการออกแบบฐานข้อมูลเชิงเวลา โดยพัฒนา NIAM ให้สามารถรองรับข้อมูลเชิงเวลาได้ และนำหลักการเชิงวัตถุมาใช้งานร่วมกันได้
6. สร้างอัลกอริธึมในการแปลงแบบจำลองที่สร้างขึ้นให้เป็นภาษานิยามเชิงวัตถุ

บทที่ 2
แบบจำลองข้อมูล NIAM

2.1 แบบจำลองข้อมูล NIAM (The NIAM Conceptual Schema)

แบบจำลองข้อมูลแบบไนแอม[4] คิดค้นขึ้นโดย 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

2.2 ขั้นตอนการออกแบบ NIAM

แม้ว่าการออกแบบฐานข้อมูลโดยวิธีไนแอมจะเป็นวิธีการที่ง่าย และมีความเป็นธรรมชาติมากกว่าวิธีอื่น ๆ ก็ตาม แต่หลักการของการออกแบบแบบจำลองข้อมูล ก็ยังนับได้ว่าเป็นงานที่ซับซ้อนอยู่ดี ดังนั้นเพื่อให้สามารถทำความเข้าใจได้ง่าย จึงแบ่งออกการออกแบบออกเป็นงาน 9 ขั้นตอนด้วยกัน ดังนี้

2.2.1 ขั้นตอนที่ 1 From Example to elementary facts

ในการออกแบบระบบสารสนเทศนั้น ขั้นตอนแรกที่เราจะต้องทำ ก็คือหาตัวอย่างข้อมูลที่ใช้งานอยู่ในระบบ ซึ่งตัวอย่างข้อมูลที่สำคัญก็คือ รายงาน (Output Report) และแบบฟอร์มที่ใช้ใน

การรับข้อมูลเข้าสู่ระบบ (Input Form) ซึ่งอาจจะอยู่ในรูปของตาราง ภาพ หรือข้อความใดๆ ก็ตาม สำหรับในตารางที่ 2.1 นั้นเป็นตัวอย่างของข้อมูลของตารางเรียน

ตารางที่ 2.1 รายงานการจองเวลาเรียนของนักศึกษา

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
..	..	..	..	..

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 อย่างไรก็ตามแม้ว่าข้อมูลจะเป็นแบบอื่น เช่น อาจจะเป็นแผนภูมิต่างๆ แต่งานในขั้นแรกนี้ก็จะต้องนำมาเขียนเป็นความจริงพื้นฐานทั้งหมด

คำว่า ความจริงพื้นฐาน นั้นอาจจะถูกมองว่าเป็นประโยคที่อยู่ในรูปแบบของ "วัตถุ กระทำสิ่งต่างๆ" (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 และเรียกเซ็ทของเอนติตี้ว่า ชนิดเอนติตี้ (Entity Type) ซึ่งมีสมาชิกเป็นตัวอย่างเอนติตี้ (Entity Instance) แต่ละตัวอย่างเอนติตี้จำเป็นต้องมีชื่อเรียกซึ่งอาจเป็นชื่อ, 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

2.2.2 ขั้นตอนที่ 2 First Draft of conceptual schema diagram

งานหลักในขั้นตอนนี้ก็คือการนำ Elementary Facts มาวาดเป็นแผนภาพ โดยจะสมมติข้อมูลขึ้นมาชุดหนึ่ง ซึ่งเป็นข้อมูลว่าใครขับรถเลขทะเบียนอะไร ดังตารางที่ 2.2

ตารางที่ 2.2 ตัวอย่างข้อมูลที่แสดงความสัมพันธ์ของรถกับเจ้าของ

Drives	Person	Car
	Adams B	235PZN
	Jones E	235PZN
	Jones E	108AAQ

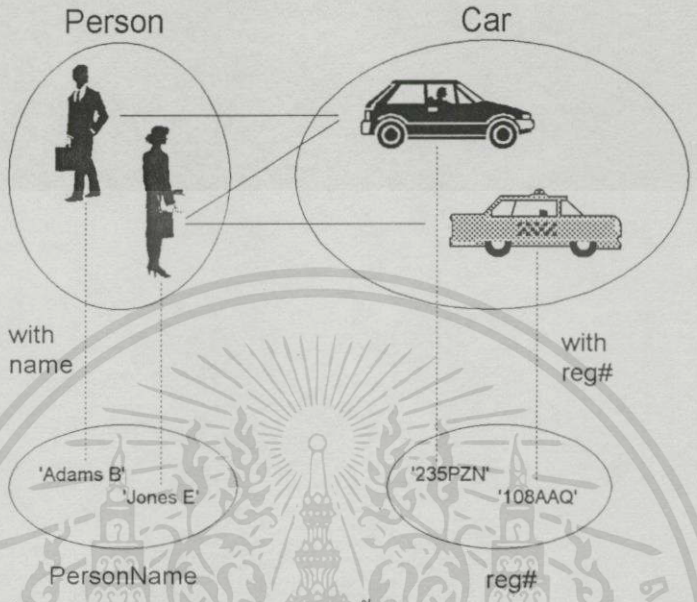
จากข้อมูลข้างต้นเราสามารถนำมาเขียนเป็น Elementary Facts ได้ดังนี้

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

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

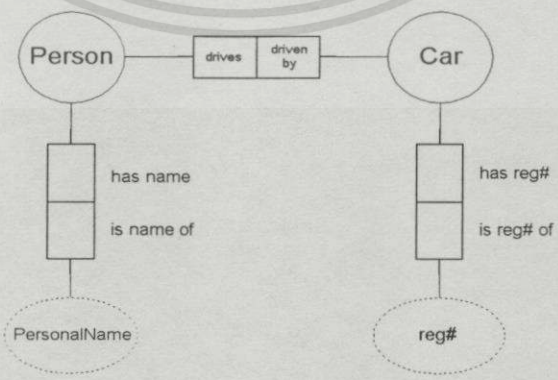
The Person with Name 'Jones E' drives the Car with reg# '108AAQ'.

ซึ่งสามารถแสดงความสัมพันธ์ให้เห็นชัดเจนดังรูปที่ 2.1 และจากความจริงพื้นฐานที่ได้ เราก็จะนำมาเขียนเป็นแผนภาพที่แสดงแนวคิด (Conceptual Schema Diagram) ได้ดังรูปที่ 2.2



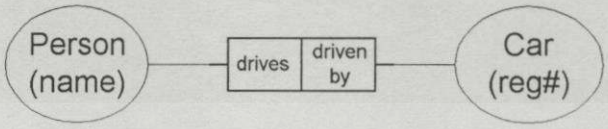
รูปที่ 2.1 แผนภาพที่แสดงความสัมพันธ์ของเอนติตี้

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



รูปที่ 2.2 แผนภูมิระดับแนวคิด

อย่างไรก็ตามจะสังเกตุได้ว่า สมาชิกแต่ละตัวในชนิดเอนติตี้จะมีความสัมพันธ์กับสมาชิกในชนิดอ้างอิงกันแบบ 1:1 เช่นในชนิดเอนติตี้แต่ละ Person จะมีเพียง 1 ชื่อและแต่ละชื่อก็จะหมายถึงคนหนึ่งคน ดังนั้นเราสามารถเขียนแผนภูมิในรูปที่ 2.2 ได้ใหม่ดังในรูปที่ 2.3 โดยไส่ชนิดอ้างอิงไว้ในวงเล็บเพื่อที่จะเขียนได้ง่ายขึ้น โดยสามารถสื่อความหมายได้คงเดิม



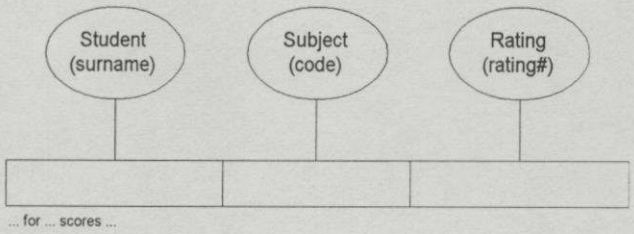
รูปที่ 2.3 การเขียนชนิดความจริงแบบย่อ

สำหรับวิธีการตรวจสอบความถูกต้องของแผนภูมิที่เราวาด สามารถทำได้โดยการไส่ตัวอย่าง และจากที่ได้กล่าวมาทั้งหมด คงจะเห็นได้ว่าการเขียนความสัมพันธ์ในรูปแผนภูมิระดับแนวคิดนั้นสามารถมองเห็นความสัมพันธ์ได้ดีกว่า

ลองมาดูกันอีกตัวอย่างโดยสมมติข้อมูลดังในตารางที่ 2.3 ซึ่งสามารถเขียนเป็นความจริงพื้นฐานได้ดังนี้ Student (surname) 'Adams' for Subject (code) 'CS100' scores Rating (rating#) 4. ซึ่งสามารถนำมาเขียนเป็นแผนภูมิระดับแนวคิดได้ดังรูปที่ 2.4 จะเห็นได้ว่าแผนภูมิที่ได้คราวนี้จะมี Role ทั้งหมด 3 Role เพราะข้อมูลทั้ง 3 มีความสัมพันธ์กันทั้งหมด และเราจะเรียกแผนภูมิที่ประกอบด้วย 2 Role ว่า Binary Fact Type และ 3 Role ว่า Ternary Fact Type

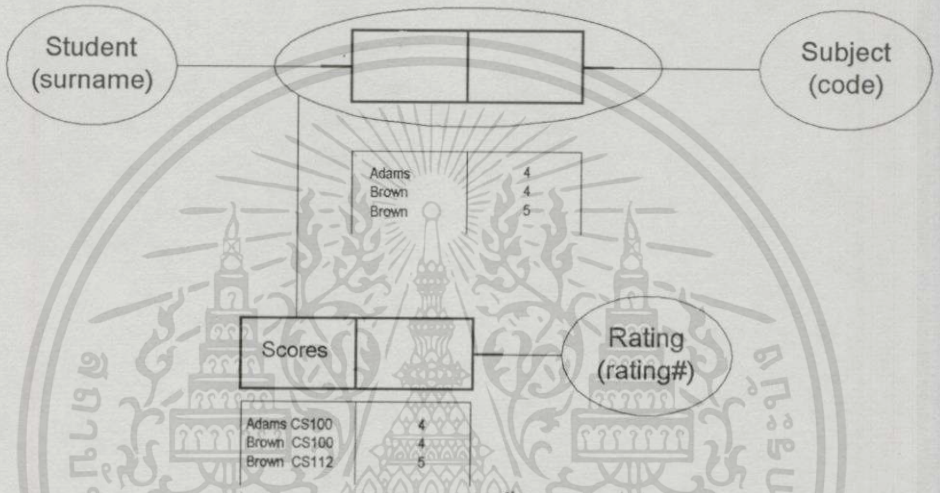
ตารางที่ 2.3 ตัวอย่างข้อมูลผลการเรียน

Student	Subject	Rating
Adams	CS100	4
Brown	CS100	4
Brown	CS112	5



รูปที่ 2.4 ชนิดความจริงแบบ Ternary

อย่างไรก็ตามการเขียนชนิดความจริงในลักษณะเช่นนี้ เมื่อพิจารณาในแง่ของความเป็นจริงแล้วก็อาจจะไม่ถูกต้องนัก เพราะชนิดเอนติตี้ Rating ถือได้ว่ามิใช่มีความสัมพันธ์กับเพียงชื่อวิชาหรือตัวนักศึกษาอย่างใดอย่างหนึ่งเท่านั้น แต่มีความสัมพันธ์กับทั้งชนิดเอนติตี้ Student และชนิดเอนติตี้ Subject เช่น นักศึกษาที่มีชื่อ Adams และลงเรียน CS100 เท่านั้น จึงจะมีคะแนนเป็น 4 จะเห็นได้ว่าเป็นความสัมพันธ์ร่วมกัน ดังนั้นเราจะเขียนใหม่เพื่อให้สื่อความหมายได้ถูกต้องมากยิ่งขึ้นได้ดังรูปที่ 2.5 และเราเรียกความสัมพันธ์ในลักษณะเช่นนี้ว่า Nesting



รูปที่ 2.5 การเขียนความสัมพันธ์แบบ Nesting พร้อมทั้งแสดงตัวอย่างข้อมูล

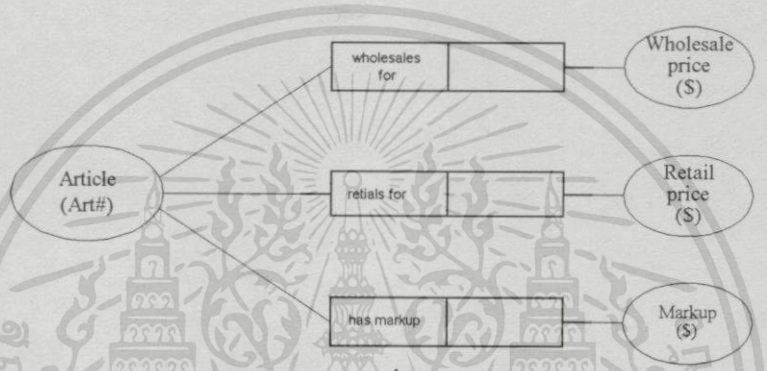
2.2.3 ขั้นตอนที่ 3 Trim schema and find derived fact type

ในบางครั้งการกำหนดชนิดเอนติตี้ขึ้นมาเรื่อยๆ ก็อาจจะเกิดความซ้ำซ้อนขึ้นมาได้ ดังนั้นในขั้นตอนนี้เราจะทำการตรวจสอบว่ามีชนิดเอนติตี้ใดบ้างที่มีความซ้ำซ้อนและสามารถจะลดลงได้ เราลองมาดูตัวอย่างในตารางที่ 2.4 ซึ่งเป็นรายการขายสินค้าและการทอนเงิน

ตารางที่ 2.4 ตัวอย่างข้อมูลการขายสินค้าและทอนเงิน

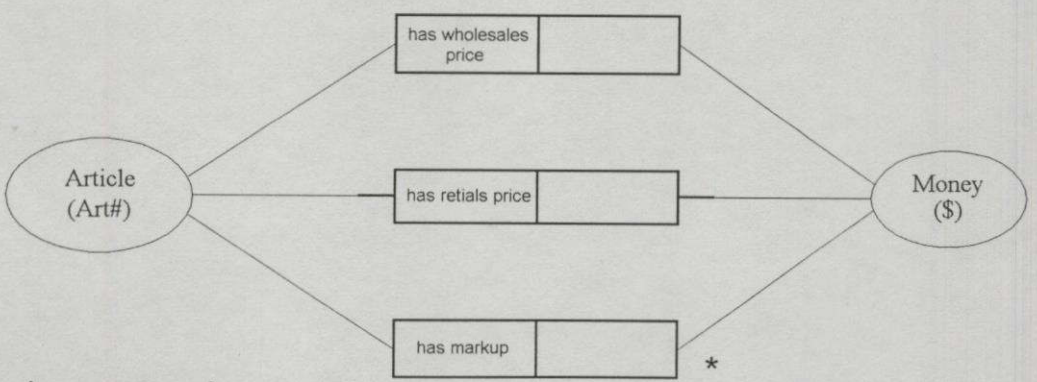
Article	Wholesale Price ($)	Retail Price ($)	Markup ($)
A1	50	75	25
A2	80	130	50
A3	50	70	20
A4	100	130	30

จากข้อมูลในตารางจะสามารถเขียนเป็นแผนภูมิระดับแนวคิดได้ดังรูปที่ 2.6 ซึ่งประกอบด้วยชนิดเอนติตี้ 4 ชนิด คือ Article, Wholesale, Retail และ Markup หากเมื่อพิจารณาให้ดีแล้วจะเห็นว่า ชนิดเอนติตี้ 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 2.6 การเขียนโครงสร้างข้อมูลระดับแนวคิดที่ผิด

การที่เก็บเช่นนี้จะมีข้อดีคือ สามารถป้องกันความผิดพลาดได้กล่าวคือ หากเราเก็บข้อมูลทั้งสามตัว ก็อาจเป็นไปได้ว่าค่าของเงินทอนของสินค้าตัวหนึ่ง อาจจะไม่เท่ากับค่าเงินที่จ่ายลบด้วยราคาขายก็เป็นได้ ซึ่งจะทำให้เกิดการขัดแย้งกันของข้อมูล ดังนั้นการเก็บข้อมูลในลักษณะดังกล่าวจึงสามารถแก้ปัญหานี้ได้ นอกจากนั้นยังเป็นการประหยัดเนื้อที่ในการเก็บอีกด้วย และเราจะเรียกชนิดความจริงแบบที่เก็บเป็นกฎนี้ว่า ชนิดความจริงแบบหาได้ (Derived Fact Type) และระบุในแผนภาพโดยเขียนเครื่องหมายดอกจันไว้ที่ด้านล่างของชนิดความจริงนั้น ดังนั้นจากแผนภาพในภาพที่ 2.6 เราสามารถนำมาเขียนใหม่ได้ดังภาพที่ 2.7



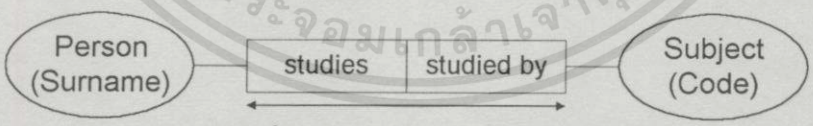
รูปที่ 2.7 แผนภาพที่ได้รับการแก้ไขแล้ว

2.2.4 ขั้นตอนที่ 4 Uniqueness Constraint

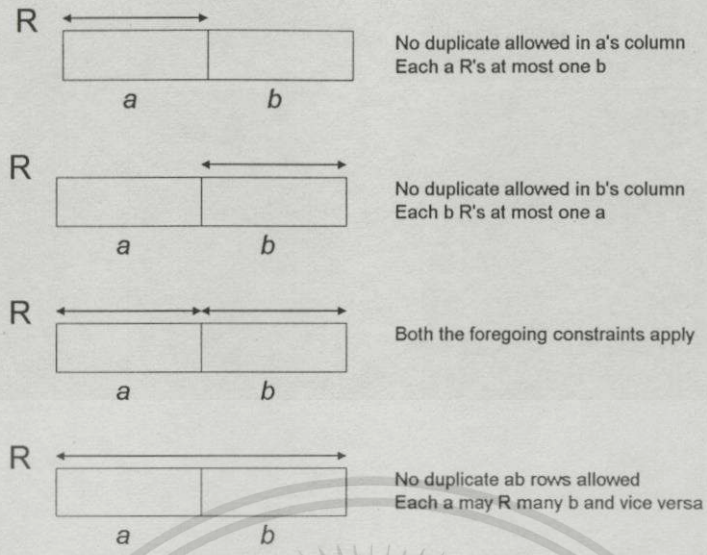
ที่ผ่านมาเราได้ทำการสร้างแผนภาพระดับแนวคิดขึ้น ทั้งในแบบที่เป็นเก็บจริง (store fact type) และแบบหาได้ (derive fact type) สำหรับในขั้นตอนที่เหลือเราจะกล่าวถึงกฎข้อบังคับ (constraint) โดยในขั้นตอนนี้เราจะกล่าวถึง Uniqueness Constriant ซึ่งเป็น Constraint ที่สำคัญมาก ซึ่งจะมีบทบาทมากในภายหลังในขั้นตอนการแปลงจากฐานข้อมูลระดับแนวคิด (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 2.8 ซึ่งจะหมายถึงว่านักศึกษาหนึ่งคนจะลงเรียนในรายวิชาหนึ่งได้เพียงครั้งเดียวเท่านั้น ดังนั้นเมื่อรวมข้อมูลของทั้ง 2 คอลัมน์แล้วจะไม่ซ้ำ (Unique) แต่ในแต่ละคอลัมน์จะมีการซ้ำได้ และเรียกความสัมพันธ์ของชนิดความจริงแบบนี้ว่าแบบ many to many สำหรับความสัมพันธ์ในแบบต่างๆ ของชนิดความจริงแสดงไว้ในรูป 2.9 ซึ่งได้แก่แบบ one to many, many to one, one to one และ many to many ซึ่งจากรูปก็ได้แสดงความหมายไว้อย่างชัดเจนดีอยู่แล้ว จึงไม่กล่าวอะไรมากไปกว่านี้

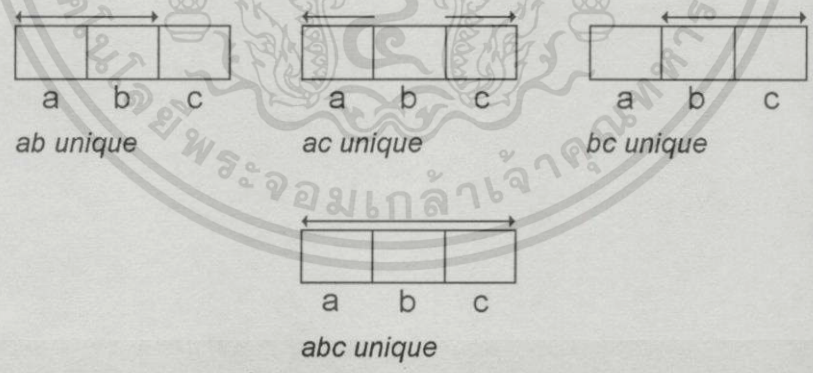


รูปที่ 2.8 Uniqueness Constraint ที่ครอบคลุมทุก Role ในชนิดความจริง



รูปที่ 2.9 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 2.10

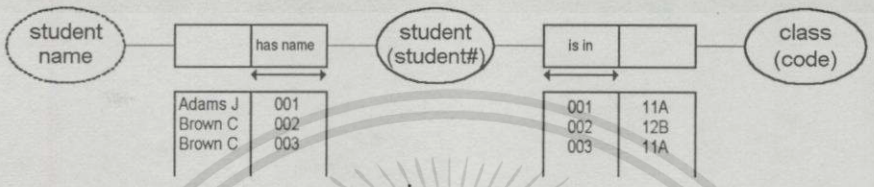


รูปที่ 2.10 Uniqueness Constraint ที่เป็นไปได้ 4 แบบของชนิดความจริงแบบเทอนารี

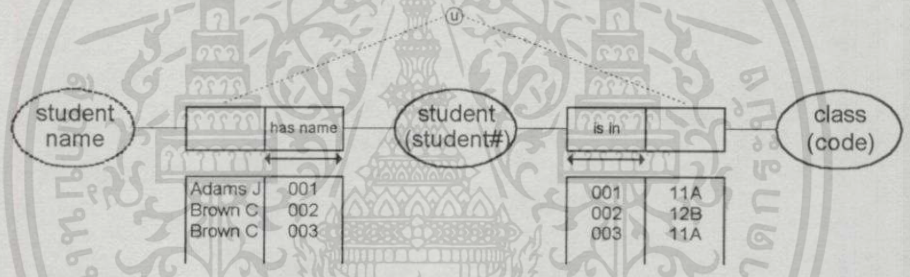
เท่าที่ผ่านมา เราได้กล่าวถึง Uniqueness Constraint ในแบบที่ Role ทุก Role ที่อยู่ภายใต้ Constraint หนึ่ง จะอยู่ในชนิดความจริงเดียวกันเท่านั้น ต่อไปเราลองมาดูข้อมูลในตารางที่ 2.5 จากตารางจะเห็นได้ว่านักศึกษาแต่ละคนจะมีชื่อเพียงชื่อเดียว และนักศึกษาแต่ละคนจะมีชั้นเรียนเพียงชั้นเรียนเดียว จากข้อมูลดังกล่าวจะสามารถเขียนแผนภาพระดับแนวคิดได้ดังรูปที่ 2.11

ตารางที่ 2.5 รายงานห้องเรียนของนักศึกษา

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



รูปที่ 2.11 แบบจำลองข้อมูลสำหรับตารางที่ 2.5



รูปที่ 2.12 การใช้ Uniqueness Constraint แบบ Inter-fact-type

อย่างไรก็ตามหากสังเกตุให้ดีจากตาราง จะเห็นได้ว่าผลรวมของชื่อและชั้นเรียนก็จะเป็นข้อมูลที่ไม่ซ้ำเช่นกัน และในกรณีเช่นนี้ NIAM ก็อนุญาตให้สามารถสร้าง Uniqueness Constraint ระหว่างชนิดความจริง 2 ชนิดความจริงได้ ดังรูปที่ 2.12

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

2.2.5 ขั้นตอนที่ 5 Arity Checks

หากในขั้นตอนที่ผ่านมานั้น เราได้ทำการออกแบบอย่างระมัดระวังแล้ว การทำงานในขั้นตอนนี้ นับว่าเป็นสิ่งที่ไม่จำเป็นเลย อย่างไรก็ตาม จากความจริงที่ว่า นักปราชญ์ยังรู้พลั้ง นั้นเป็นข้อ

เตือนใจได้เป็นอย่างดี เพราะแม้ว่าเราจะเป็นผู้ออกแบบที่ชำนาญเพียงใดก็ตาม ก็ยังอาจเกิดการผิดพลาดขึ้นได้

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

สำหรับการตรวจสอบความถูกต้องนั้น จริงๆ แล้วอาจจะมีวิธีการมากมาย สุดแต่ใครจะใช้วิธีการอย่างไร หากแต่ในที่นี้เราจะกล่าวถึงเพียง 3 วิธี

2.2.5.1 ใช้สามัญสำนึก หรือความรู้พื้นฐานในการพิจารณาตัดสินว่ามีข้อมูลสูญหายเพราะการแยกชนิดความจริงออกมาหรือไม่

2.2.5.2 ใช้กฎของการแยกจากกัน (Splittability Rules) ซึ่งจะกล่าวถึงภายหลัง

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

ตารางที่ 2.6 ผลการเรียนของนักศึกษาในวิชาเรียนวิชาหนึ่ง

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

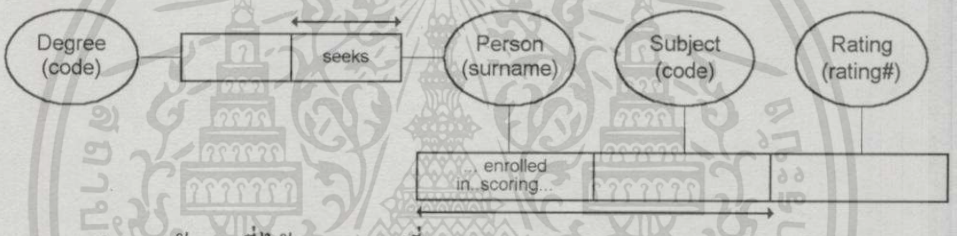
สำหรับวิธีการแรกนั้น อาจจะเป็นวิธีที่ดูไม่ค่อยจะมีหลักการสักเท่าใดนัก แต่จริงๆ แล้วนับได้ว่าเป็นวิธีที่ดีวิธีหนึ่ง เพราะเป็นวิธีที่มีผลสอดคล้องกับขั้นตอนการกำหนด Elementary Facts เพื่อให้การทำงานในวิธีนี้ชัดเจนยิ่งขึ้น เราลองมาดูตัวอย่างต่อไปนี้ เราจะเริ่มจากข้อมูลในตารางที่ 2.6 และจากวิธีการที่ผ่านมา 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 2.13 ทีนี้เราก็จะมาลองดูว่าหากเราต้องการจะแยกชนิดความจริงแบบ Ternary ในภาพออกเป็น 2 ชนิดความจริงจะได้หรือไม่ เราก็จะได้ข้อมูลว่า Adams ได้คะแนนเท่ากับ 7 และ Adams ลงเรียนในวิชา CS112 ซึ่งจะเห็นได้อย่างชัดเจนว่ามีข้อมูลบางส่วนหายไป (Information Loss) นั่นคือข้อมูลที่ว่า Adams ได้คะแนนเท่ากับ 7 ซึ่งไม่ทราบว่าได้คะแนนในวิชาใด ดังนั้นสรุปได้ว่าชนิดความจริงนี้ไม่สามารถจะแยกได้



รูปที่ 2.13 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.6

2.2.5.4 การใช้กฎการแยก (Splittability Rule)

จริงๆ แล้วเราได้กล่าวถึงกฎที่ว่านี้ไปครั้งหนึ่งแล้ว ในขั้นตอนที่ผ่านมา นั่นคือกฎที่ว่าจะต้องมี Role ที่ไม่อยู่ภายใต้ Uniqueness Constraint ได้ไม่เกิน 1 เท่านั้น

2.2.5.5 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

ตารางที่ 2.7 ข้อมูลก่อนการทำ Projection

Person	Subject	Rating
Adams	CS112	7
Adams	CS110	6
Adams	PD102	7
Brown	CS112	6
Brown	PD102	7
Collins	CS112	7

ทีนี้เราลองมาดูข้อมูลในตารางที่ 2.7 จากรูปจะเห็นได้ว่าเราทำการ Projection ตารางออกเป็น 2 ตารางดังตารางที่ 2.8 และเมื่อเรานำ 2 ตารางนี้มา Join กันใหม่ก็จะได้ตารางที่ 2.9 ซึ่งจะเห็นได้ว่ามีแถวของข้อมูลเกิดขึ้นใหม่ (แถวที่มีเครื่องหมาย X) ซึ่งข้อมูลเหล่านี้เป็นข้อมูลที่ไม่ได้มีอยู่ในตารางที่เราทำการ Projection มาดังนั้นจะสรุปได้ว่าเราไม่สามารถแยกชนิดความจริงนี้ได้

ตารางที่ 2.8 ข้อมูลที่ผ่านการทำ Projection ออกเป็น 2 ตารางแล้ว

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 2.9 ที่ได้จากการ Join ตารางเข้าด้วยกัน

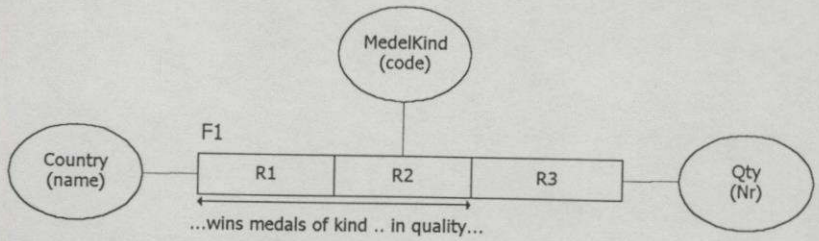
	Person	Subject	Rating
X	Adams	CS112	7
X	Adams	CS112	6
	Adams	CS110	7
	Adams	CS110	6
	Adams	PD102	7
X	Adams	PD102	6
	Brown	CS112	6
X	Brown	CS112	7
X	Brown	PD102	6
	Brown	PD102	7
	Collins	CS112	7

2.2.6 ขั้นตอนที่ 6 More Constraints

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

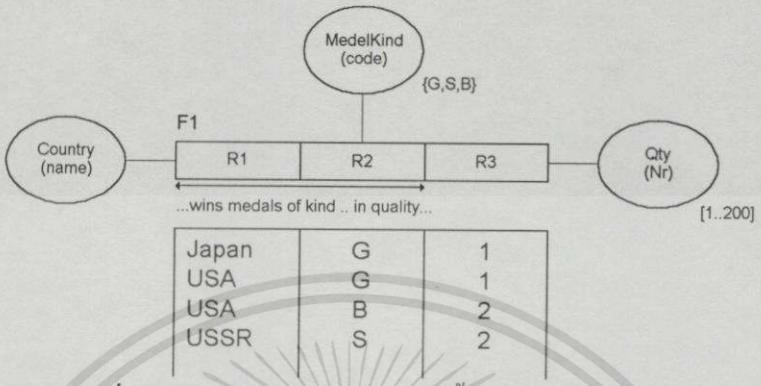
2.2.6.1 Entity Type Constraint

จากรูปที่ 2.14 ชนิดเอนติตี้ Country เป็นเซ็ทของประเทศต่างๆ ที่เป็นไปได้ ในกรณีนี้หากเราต้องการที่จะหาว่ามีสมาชิกใดบ้างที่สามารถอยู่ในเอนติตี้นี้ได้ อาจจะเป็นงานที่ยากเพราะมีการตั้งประเทศขึ้นมาใหม่ๆ อยู่เรื่อยๆ อย่างไรก็ตามสำหรับในบางชนิดเอนติตี้นั้นง่ายมาที่จะบอกถึงสมาชิกของเอนติตี้นั้น เช่น เอนติตี้ของเหรียญที่ใช้ในกีฬาโอลิมปิกก็จะมีเหรียญ ทอง เงิน และบรอนซ์ ซึ่งข้อจำกัดของความเป็นไปได้ในชนิดเอนติตี้นี้เอง ที่เรียกว่า Entity Type Constraint



รูปที่ 2.14 ชนิดความจริงของผลกีฬาโอลิมปิก

ซึ่งในกรณีของเอนติตี้เหรียญนี้สามารถเขียนได้ดังในรูปที่ 2.15 จะเห็นได้ว่า เราจะใช้วิธีเขียนเซ็ทของเอนติตี้ไว้ที่ด้านข้างของเอนติตี้ โดยที่ในกรณีที่มีสมาชิกจำกัดก็จะใช้เครื่องหมาย เช่น G,S,B และหากเป็นช่วงเราจะใช้ [ ] เช่น [1..200]



รูปที่ 2.15 ชนิดความจริงที่มีคอนสแตนต์กำกับในเอนติตี้ MedalKind และ Qty

2.2.6.2 Mendatory and optional roles

เพื่อที่จะอธิบายว่าทำไมเราจะต้องมี Constraint นี้ด้วย เราจะพิจารณาจากตัวอย่างข้อมูลในตารางที่ 2.10 จะเห็นได้ว่ามีข้อมูลอยู่ช่องหนึ่งที่เราใส่เครื่องหมาย "?" ไว้เพื่อแสดงว่ายังไม่มีการบันทึกข้อมูลในตารางนี้ เช่น Brown S ที่จริงแล้วอาจจะมีโทรศัพท์ก็เป็นได้ แต่ยังไม่ได้รับการบันทึก

ตารางที่ 2.10 ตารางหมายเลขโทรศัพท์ของคนไข้

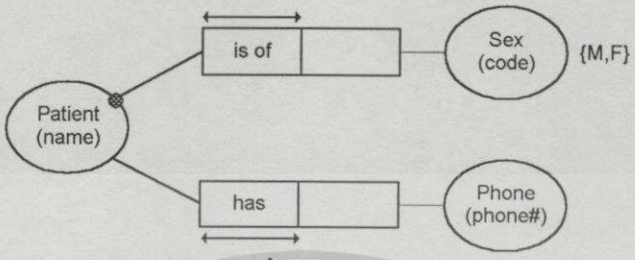
Patient	Sex	Phone
Adams C	F	2057642
Brown S	F	?
Collins T	M	8853020

อย่างไรก็ตามหากข้อมูลนั้นเป็นข้อมูลที่สำคัญเช่นข้อมูลของเพศ การที่เราจะไม่บันทึกข้อมูลนั้นลงไปก็อาจจะนับได้ว่าเป็นการไม่ถูกต้องนัก ดังนั้นสำหรับในข้อมูลบางตัวที่ "ต้องบันทึก" นั้น เราจะเรียก Role นั้นว่า Mendatory และใช้เครื่องหมาย "จุด" เพื่อระบุ Mendatory Constraint ดังแสดงในรูปที่ 2.16 ซึ่งมีความหมายว่า Patient จะต้องมีเพศ

2.2.6.3 Subtype

สำหรับเรื่องของ Subtype นี้เราจะใช้แผนภูมิของออยเลอร์ (Euler's Diagram) ในการแสดงความสัมพันธ์ระหว่างเอนติตี้ จากที่เราได้กล่าวมาแล้วว่าเอนติตี้เป็นเซ็ท ทีนี้ลองสมมติว่าเรามีเอนติ

ตี้ Manager ซึ่งเป็นเซ็ทของผู้จัดการทั้งหมด และเอนติตี้ Employee ซึ่งเป็นเซ็ทของพนักงานที่ทำงานอยู่ และจากความจริงที่ว่าผู้จัดการทุกคนย่อมเป็นพนักงานด้วยเช่นกัน และลักษณะความสัมพันธ์ในลักษณะเช่นนี้เราจะเรียกว่า Subtype ซึ่งจะเขียนได้ดังรูปที่ 2.17



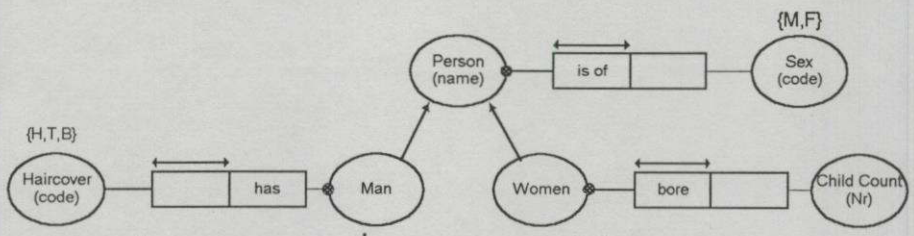
รูปที่ 2.16 การใช้ Mendatory Constraints ที่ถูกต้อง

รูปที่ 2.17 แผนภูมิออยเลอร์ที่แสดงว่า A เป็นซับเซ็ตของ B

จากรูปหมายถึงว่าเอนติตี้ B เป็น Subset ของเอนติตี้ A และเรียก A ว่า Supertype เรียก B ว่า Subtype นั่นย่อมหมายถึงว่าสมาชิกทุกตัวใน B ก็ย่อมจะมีคุณสมบัติของเอนติตี้ A ด้วย เราลองมาดูตัวอย่างกันอีกตัวอย่างหนึ่งโดยอาศัยตารางที่ 2.11 จากตารางจะเห็นได้ว่าเฉพาะเพศชายเท่านั้นจึงจะมีการบันทึก Haircover และเฉพาะเพศหญิงเช่นกันที่มีการบันทึกจำนวนบุตร ในลักษณะดังกล่าวนี้หากเราใช้วิธีการเขียนในแบบที่ผ่านมาก็ย่อมได้ และไม่ผิดหากแต่ไม่สามารถแสดงความหมายได้ครบ เพราะ NIAM นั้นเป็นแบบจำลองข้อมูลระดับแนวคิดดังนั้นวิธีการที่ดีกว่าคือการใช้ Subtype นั่นเองซึ่งจะเขียนเป็นแผนภาพได้ดังรูปที่ 2.18

ตารางที่ 2.11 ข้อมูลตัวอย่างที่แสดงการใช้ Subtype

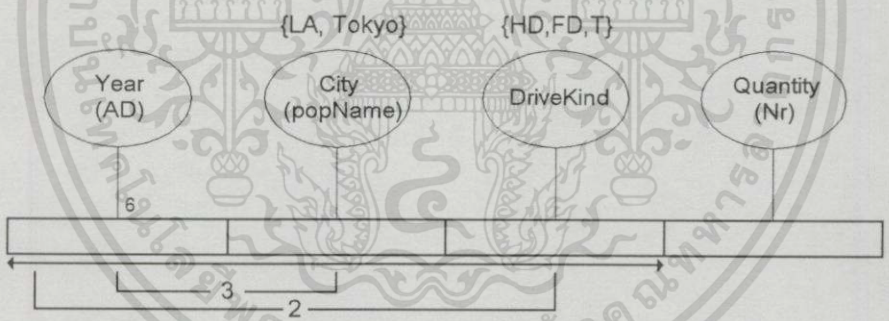
Person	Sex	Haircover	NrChildren
Jones E	F	-	2
Smith J	M	T	-
Blow J	M	B	-
Lane L	F	-	0
Blossom B	F	-	5



รูปที่ 2.18 แบบจำลองข้อมูลระดับแนวคิดที่นำเอา Subtype มาใช้งาน

2.2.6.4 Occurrence frequencies

ในตอนนี้เราจะกล่าวถึง Constraint อีกตัวหนึ่งที่จะใช้ในการระบุจำนวนครั้งที่ชนิดเอนติตี้จะเก็บได้สำหรับ Role หนึ่งๆ Constraint นี้มีชื่อเรียกว่า Occurrence frequency constraint ในรูปที่ 2.19 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



In year .. in city .. we sold drives of kind .. in quantity ..

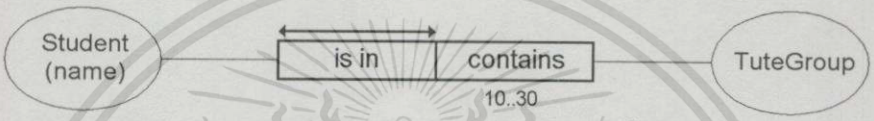
1986	LA	HD	700
1986	LA	FD	4000
1986	LA	T	500
1986	Tokyo	HD	300
1986	Tokyo	FD	3000
1986	Tokyo	T	500
1987	LA	HD	2000
-	-	-	-

รูปที่ 2.19 ตัวอย่างของแบบจำลองข้อมูลที่ใช้ 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

นอกจากจะระบุจำนวนครั้งการบันทึกข้อมูลที่แน่นอนแล้ว Occurrence Frequency Constraint ยังอนุญาตให้มีการบันทึกเป็นช่วงอีกด้วย ดังตัวอย่างในรูปที่ 2.20 ซึ่งเป็นตัวอย่างของการบันทึกข้อมูลนักเรียนที่อยู่ในกลุ่มกวดวิชาต่าง ๆ โดยสามารถระบุว่าจำนวนนักเรียนที่อยู่ในแต่ละกลุ่มจะต้องมีจำนวนอยู่ระหว่าง 10-30 คน เป็นต้น

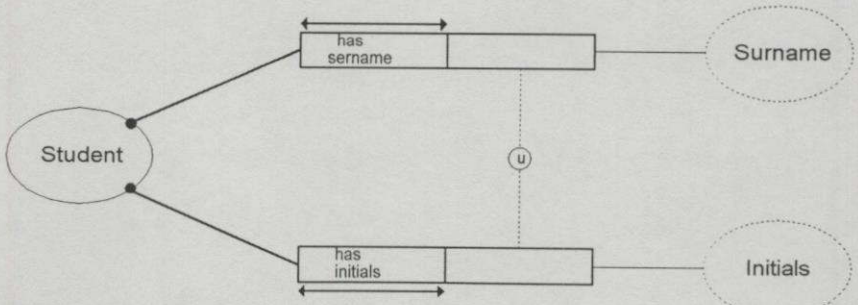


รูปที่ 2.20 การใช้ Occurrence Frequency Constraint อีกรูปแบบหนึ่ง

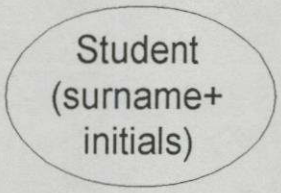
2.2.7 ขั้นตอนที่ 7 Entity Identification Schemes

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

ดังนั้นความสัมพันธ์ระหว่างชนิดเอนติตี้และชนิดเลเบิ้ลก็ไม่จำเป็นต้องอยู่ในรูปแบบ 1:1 เสมอไป โดยอาจจะอยู่ในรูปแบบใดก็ได้ แต่ทั้งนี้จะต้องมีชนิดเลเบิ้ลกลุ่มหนึ่งที่สามารถระบุถึง (Identify) แต่ละตัวอย่างในชนิดเอนติตี้ได้เสมอ ในรูปที่ 2.21 เป็นตัวอย่างของการใช้ 2 ชนิดเลเบิ้ลในการระบุเอนติตี้ ซึ่งสามารถเขียนแบบย่อได้ในรูปที่ 2.22



รูปที่ 2.21 การใช้ชนิดเลเบิ้ล 2 ตัวในการระบุถึงแต่ละเอนติตี้ในชนิดเอนติตี้ Student

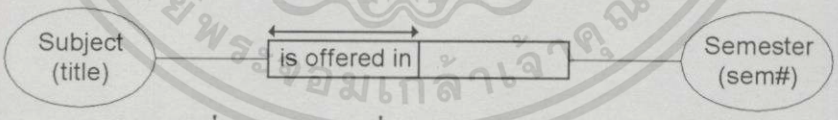


รูปที่ 2.22 การเขียนความสัมพันธ์แบบย่อระหว่างชนิดเอนติตี้กับชนิดเลเบิ้ลในรูปที่ 2.21

อย่างไรก็ตามการกำหนดว่าจะให้ชนิดเลเบิ้ลใดเป็นตัวระบุถึงแต่ละเอนติตี้นั้น ก็จะขึ้นอยู่กับขอบเขตของข้อมูลด้วย แบบจำลองข้อมูลที่สามารถใช้กับขอบเขตข้อมูลหนึ่ง ๆ อาจจะใช้ไม่ได้ผลกับขอบเขตข้อมูลที่กว้างกว่าก็ได้ สมมติว่าเรามีข้อมูลที่ว่าในแต่ละภาควิชาในมหาวิทยาลัยจะมีวิชาสอนประจำภาควิชานั้นอยู่ ดังข้อมูลตัวอย่างในตารางที่ 2.12 ซึ่งจากข้อมูลในตารางเราสามารถจะใช้ชื่อวิชาในการระบุถึงแต่ละวิชานั้น และสามารถเขียนแบบจำลองข้อมูลได้ดังรูปที่ 2.23

ตารางที่ 2.12 ตัวอย่างข้อมูลการเรียนในภาควิชา Physics

Physics:	Subject	Semester
	Electronics	1
	Mechanics	1
	Optics	2

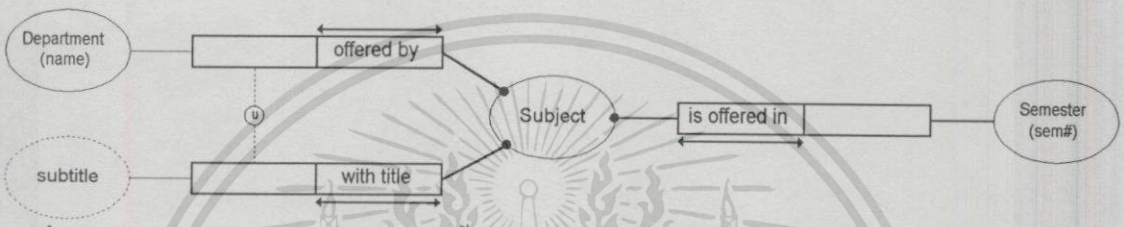


รูปที่ 2.23 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.12

แต่ถ้าขอบเขตของข้อมูลเปลี่ยนไป โดยแทนที่จะมีเฉพาะวิชาเรียนในภาควิชาฟิสิกส์ภาควิชาเดียว ก็มีข้อมูลของภาควิชาคณิตศาสตร์ด้วย ดังตารางที่ 2.13 ซึ่งจะเห็นได้ว่าทั้ง 2 ภาควิชาล้วนแต่มีการสอนวิชา Machanics ด้วยกันทั้งคู่ ซึ่งวิชาทั้งสองนี้แม้จะมีชื่อเดียวกัน แต่ก็ไม่ได้หมายความว่าจะมีเนื้อหาการสอนที่เหมือนกัน ดังนั้นการอ้างถึงแต่ละวิชาโดยใช้ชื่อวิชาจึงไม่สามารถใช้ได้อีกต่อไป ซึ่งหมายความว่าแบบจำลองข้อมูลในรูปที่ 2.23 จะใช้ไม่ได้เมื่อนำมาใช้กับขอบเขตข้อมูลที่กว้างกว่า ดังนั้นเราจึงต้องสร้างแบบจำลองข้อมูลใหม่ภายใต้ขอบเขตข้อมูลใหม่ ซึ่งก็จะได้เป็นแบบจำลองข้อมูลในรูปที่ 2.24

ตารางที่ 2.13 ข้อมูลของภาควิชาคณิตศาสตร์

Maths	Subject	Semester
	Algebra	2
	Calculus	1
	Mechanics	1



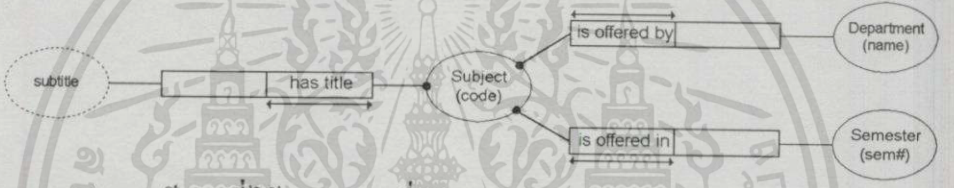
รูปที่ 2.24 แบบจำลองข้อมูลที่ออกแบบขึ้นมาใหม่

แบบจำลองข้อมูลในรูปที่ 2.24 นั้น แม้ว่าจะสามารถแทนข้อมูลในตัวอย่างข้อมูลได้ทั้งหมด และยังสามารถอ้างถึงวิชาเรียนโดยการอ้างภาควิชาที่สอนควบคู่ไปกับการอ้างชื่อวิชาได้ แต่ก็นับว่าเป็นวิธีการที่ไม่สะดวกมากนัก เพราะทุก ๆ ครั้งที่จะอ้างวิชาจะต้องอ้างว่า "วิชาแมคคานิกส์ที่สอนโดยภาควิชาคณิตศาสตร์" จึงจะสามารถถึงวิชาได้อย่างสมบูรณ์ ซึ่งเป็นวิธีการที่ยืดยาดมากไปสักหน่อย ซึ่งในกรณีเช่นนี้ ขอแนะนำให้แก้ไขแบบจำลองข้อมูลให้มีรูปแบบที่ง่ายขึ้น โดยการเพิ่มข้อมูลเข้าไปอีกตัวหนึ่ง ซึ่งในที่นี้ก็คือรหัสวิชานั่นเอง โดยมีข้อแม้ว่ารหัสวิชาจะต้องสามารถอ้างถึงทุก ๆ วิชาในชนิดเอนติตี้ได้

ในตารางที่ 2.14 จะเป็นข้อมูลของวิชาทั้งหมดที่ได้เพิ่มรหัสวิชาเข้าไปแล้ว ซึ่งสามารถนำมาเขียนเป็นแบบจำลองข้อมูลได้ดังรูปที่ 2.25 ซึ่งขอให้สังเกตุว่าความสัมพันธ์ของการอ้างวิชาเรียนได้กลับมาอยู่ในรูปแบบ 1:1 อีกครั้ง เพราะในการอ้างชื่อวิชาเราสามารถอ้างแค่ "PH102" ก็สามารถระบุวิชาเรียนได้แล้ว

ตารางที่ 2.14 แสดงข้อมูลของวิชาทั้งหมดหลังจากที่กำหนดรหัสวิชาแล้ว

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



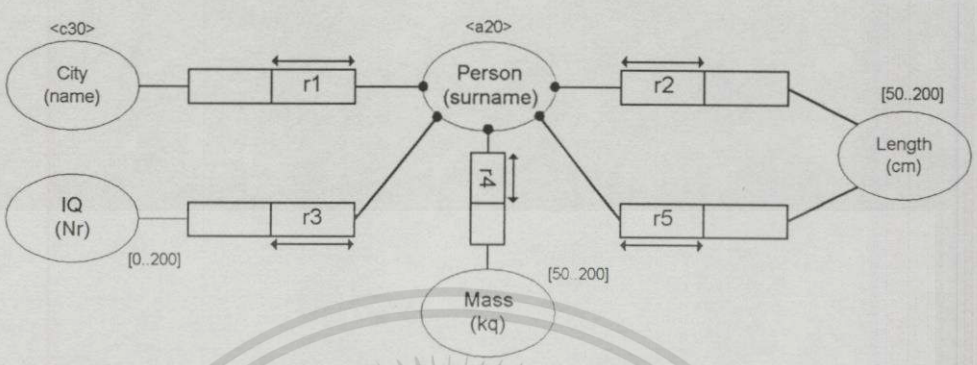
รูปที่ 2.25 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.14

จากแบบจำลองข้อมูลจะเห็นได้ว่าชนิดเอนติตี้ Subject มีวิธีการที่สามารถระบุ (Identify) ถึงแต่ละวิชาได้ 2 วิธีด้วยกัน คือ โดยการใช้รหัสวิชา และโดยการใช้ภาควิชาร่วมกับชื่อวิชาโดยจะเรียกรวมกันว่า Subtitle ซึ่งในกรณีที่ชนิดเอนติตี้ใด ๆ ที่มีการอ้างถึงได้มากกว่า 1 แบบ ก็ให้ระบุถึงแบบอ้างอิงหลัก (Primary Identifier) เอาไว้ด้วย ซึ่งจากรูปที่ 2.25 แบบอ้างอิงหลักก็คือ รหัสวิชานั่นเอง

ก่อนจะจบการทำงานในขั้นตอนที่ 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 ก็ไม่ควรจะเก็บข้อมูลที่เป็นตัวอักษรหรือตัวเลขที่ไม่ได้อยู่ใน

ช่วงดังกล่าว เป็นต้น ดังนั้นอาจจะถือได้ว่าการกำหนดแบบนี้อาจถือได้ว่าเป็นคอนสแตนต์อย่างหนึ่ง โดยมีชื่อเรียกว่า Lexical Constraint ซึ่งได้แสดงไว้ในรูปที่ 2.26 แล้ว



รูปที่ 2.26 ตัวอย่างการใช้งาน Lexical Constraint

2.2.8 ขั้นตอนที่ 8 Further Constraints

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

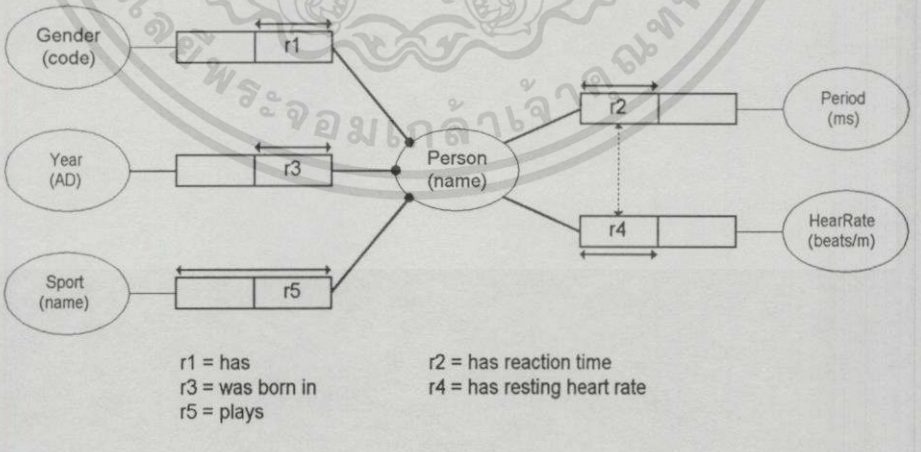
ก่อนจะกล่าวถึงคอนสแตนต์เราก็ต้องกำหนดตัวอย่างข้อมูลก่อน ในตารางที่ 2.15 เป็นข้อมูลของสมาชิกในศูนย์สุขภาพแห่งหนึ่ง จากตารางเราจะเห็นเครื่องหมาย "?" ซึ่งมีความหมายว่าไม่มีการบันทึกข้อมูลในขณะนี้ (อาจจะมีการบันทึกในอนาคต ซึ่งจะต่างกับเครื่องหมาย "-" ซึ่งมีความหมายว่าไม่มีข้อมูล) และนั่นหมายความว่าข้อมูลในสองคอลัมน์ขวาสุดนั้น จะบันทึกหรือไม่ก็ได้

แต่จะสังเกตได้ว่าหากสมาชิกคนใดที่เข้ารับการตรวจสอบ ความเร็วในการโต้ตอบของระบบประสาท ก็จะต้องเข้ารับการตรวจอัตราการเต้นของหัวใจด้วย ซึ่งข้อกำหนดนี้อาจจะเป็นกฎของศูนย์สุขภาพก็ได้ว่า สมาชิกคนใดที่ต้องการตรวจสอบสุขภาพ จะต้องเข้ารับการทดสอบทั้งสองอย่างเสียก่อน เพราะการตรวจสอบเพียงอย่างใดอย่างหนึ่ง ไม่อาจบ่งบอกได้ถึงสุขภาพที่แท้จริงได้ ผลการทดสอบทั้งสองอย่างจะต้องนำมาพิจารณาร่วมกัน จึงสามารถบอกได้ว่าสุขภาพของสมาชิกเป็นอย่างไรบ้าง

ตารางที่ 2.15 ข้อมูลของสมาชิกในศูนย์สุขภาพแห่งหนึ่ง

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 tennis	250	80
Hume PE	F	1946	golf	305	93
Jones T	M	1965	tennis	?	?

เพื่อให้ได้ข้อมูลที่เป็นไปตามข้อกำหนดของศูนย์สุขภาพ ในแบบจำลองข้อมูลจึงควรจะมีการระบุถึงข้อบังคับนี้ โดยจะเรียกกฎข้อบังคับแบบนี้ว่า Equality Constraint ซึ่งจะเขียนโดยใช้เส้นประที่มีลูกศรอยู่ที่หัวและท้าย ดังตัวอย่างในรูปที่ 2.27 ซึ่งหากท่านเคยศึกษาตรรกศาสตร์มาบ้าง ก็จะนึกได้ว่าเครื่องหมายนี้ก็คือเครื่องหมายของ "ก็ต่อเมื่อ" (if and only if) นั่นเอง จากในรูปจะมีความหมายว่าเมื่อมีการบันทึกข้อมูลลงใน r2 หรือ r4 อันใดอันหนึ่ง จะต้องมีการบันทึกข้อมูลลงในอีก Role หนึ่งด้วย ดังนั้นจำนวนครั้งของการบันทึกข้อมูลลงในชนิดความจริงทั้งสองจึงเท่ากันเสมอ และจึงเรียกคอนสแตนต์นี้ว่า Equality



รูปที่ 2.27 การใช้ Equality Constraint

คราวนี้เรามาดูตัวอย่างข้อมูลอีกตัวอย่างหนึ่งในตารางที่ 2.16 ซึ่งเป็นข้อมูลการจอดรถของพนักงานในบริษัทแห่งหนึ่ง การจอดรถในบริษัทนี้จะมีให้จอดได้ 2 แบบคือ อาจจะจอดในที่จอดรถ

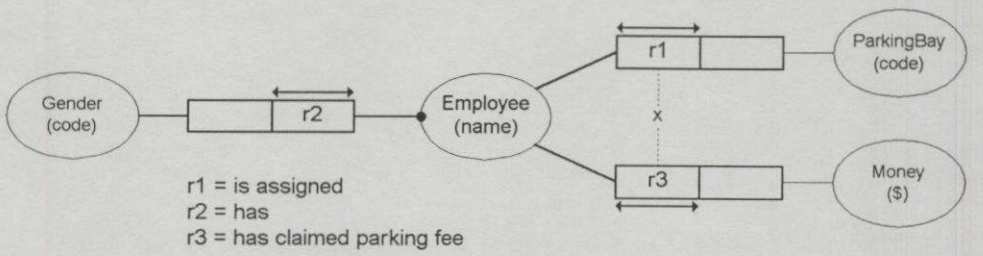
ของบริษัท (สำหรับหัวหน้า) หรืออาจจะไปจอดในที่จอดรถข้างนอก ซึ่งจะต้องเสียค่าใช้จ่ายเพิ่มเติม (สำหรับลูกน้อง) ซึ่งพนักงานแต่ละคนจะเลือกได้ในแบบใดแบบหนึ่งเท่านั้น เพราะพนักงานแต่ละคนย่อมจะขับรถมาทำงานได้ครั้งละคันเดียว จึงย่อมต้องการที่จอดแห่งเดียวเท่านั้น

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

ตารางที่ 2.16 ข้อมูลการจอดรถของพนักงานในบริษัท

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

เมื่อข้อมูลมีลักษณะของการเกิดขึ้นในแบบใดแบบหนึ่งตามตัวอย่างในตารางที่ 2.16 แล้ว เมื่อมีการสร้างแบบจำลองข้อมูลที่บันทึกข้อมูลตามตัวอย่าง ก็ควรจะมีการบันทึกกฎข้อบังคับนี้ไว้ด้วย โดยให้ชื่อกฎข้อบังคับนี้ว่า Exclusion Constraint โดยรูปที่ 2.28 จะเป็นแบบจำลองข้อมูลที่สร้างจากตาราง ซึ่งจะเห็นได้ว่ามีการบันทึก Exclusion Constraint ไว้โดยการเขียนเส้นประแล้วมีเครื่องหมาย "x" หรือกากบาทตรงกลาง ซึ่งมีความหมายว่าหากมีการบันทึกข้อมูลพนักงานใน r1 แล้วจะต้องไม่มีการบันทึกข้อมูลพนักงานคนเดียวกันนั้นใน r3 อีก



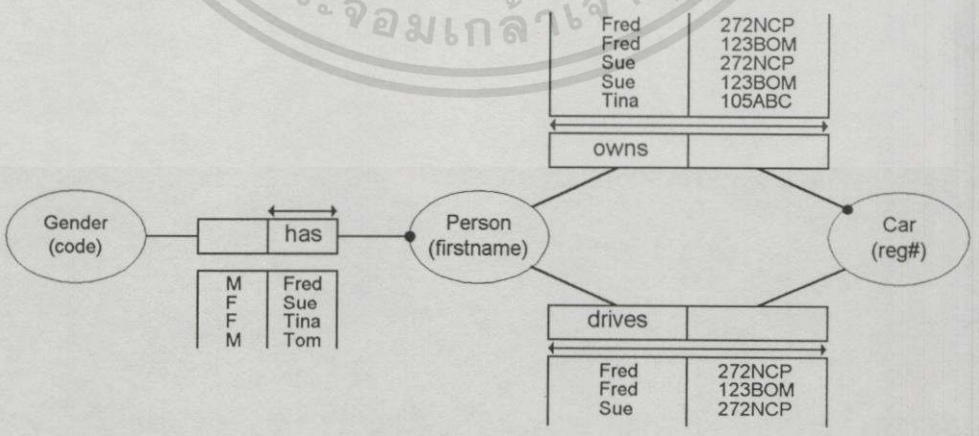
รูปที่ 2.28 แบบจำลองข้อมูลที่แสดงการใช้ Exclusion Constraint

เอาละท้ายสุดนี้เราก็มาดูคอนสแตนต์ตัวสุดท้ายกัน โดยจะเริ่มจากข้อมูลในตารางที่ 2.17 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 2.29 ซึ่งจะเห็นว่าชนิดความจริงที่บันทึกข้อมูลการขับรถนั้นจะเป็นซับเซ็ตของชนิดความจริงการเป็นเจ้าของ และจะเขียนคอนสแตนต์โดยใช้เส้นประโดยมีลูกศรข้างเดียว โดยชี้ไปทางด้านที่เป็นซูเปอร์เซ็ต โดยเรียกคอนสแตนต์นี้ว่า Subset Constraint

ตารางที่ 2.17 ข้อมูลการเป็นเจ้าของรถและข้อมูลการขับรถ

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



รูปที่ 2.29 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.17

2.2.9 ขั้นตอนที่ 9 Final Check

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

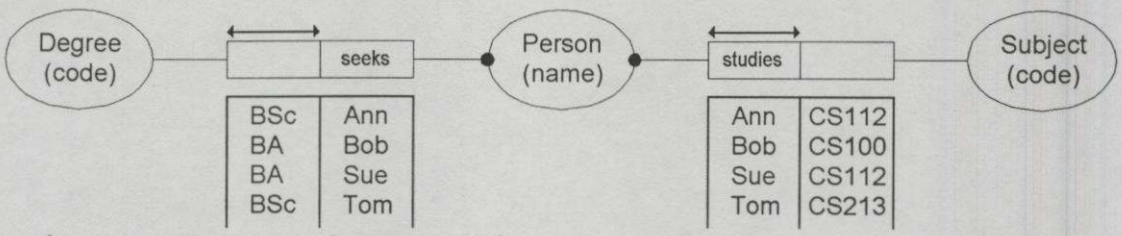
2.2.9.1 การตรวจสอบกับตัวอย่างข้อมูล

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

การตรวจสอบในขั้นนี้นั้นเราจะตรวจสอบโดยใช้แผนภูมิที่เรียกว่า Schema-Base Diagram ดังตัวอย่างในรูปที่ 2.30 ซึ่งสร้างจากตัวอย่างข้อมูลในตารางที่ 2.18 ซึ่งจะเห็นได้ว่าเราได้ทดลองใส่ข้อมูลลงในแบบจำลองโดยตรง ซึ่งหากข้อมูลมีจำนวนมากก็ให้ใช้วิธีสุ่มเอาข้อมูลมาทดสอบก็ได้ จากผลการทดสอบในส่วนของชนิดความจริงนั้น ไม่มีอะไรผิดพลาดโดยแบบจำลองสามารถแทนข้อมูลจากตารางได้เป็นอย่างดี

ตารางที่ 2.18 ตัวอย่างข้อมูล

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



รูปที่ 2.30 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.18

แต่ในส่วนของคอนสแตนต์นั้นจะเห็นได้ว่า Uniqeness Constraint เกิดการขัดแย้งขึ้น เพราะในชนิดความจริงของวิชาที่เรียนนั้น คอนสแตนต์กำหนดไว้ว่าแต่ละคนจะเรียนได้มากที่สุดวิชาเดียวเท่านั้น แต่ในตัวอย่างข้อมูล Ann เรียนถึง 2 วิชา ดังนั้นในส่วนนี้จะต้องแก้ไขแบบจำลองโดยจะต้องจะต้องขยายคอนสแตนต์ให้ครอบคลุมทั้ง 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

2.2.9.2 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 สำหรับอีกชนิดความจริงหนึ่งจะบันทึกข้อมูลว่า 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

2.2.9.3 การตรวจสอบความสมบูรณ์

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

2.3 การแปลงจาก Conceptual Schema ไปเป็น Relational Schema

ในขั้นตอนก่อนหน้านี้ เราได้ศึกษาถึงวิธีการออกแบบแบบจำลองข้อมูลระดับแนวคิดมาแล้ว ซึ่งแม้ว่าจะเป็นแบบจำลองข้อมูลที่สมบูรณ์ สามารถรองรับข้อมูลได้ถูกต้องโดยมีการควบคุมความผิดพลาดได้เป็นอย่างดี แต่เป็นที่น่าเสียใจว่าในปัจจุบันยังไม่มีระบบฐานข้อมูลใดที่สามารถใช้งานกับแบบจำลองข้อมูลไนแอมได้ 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

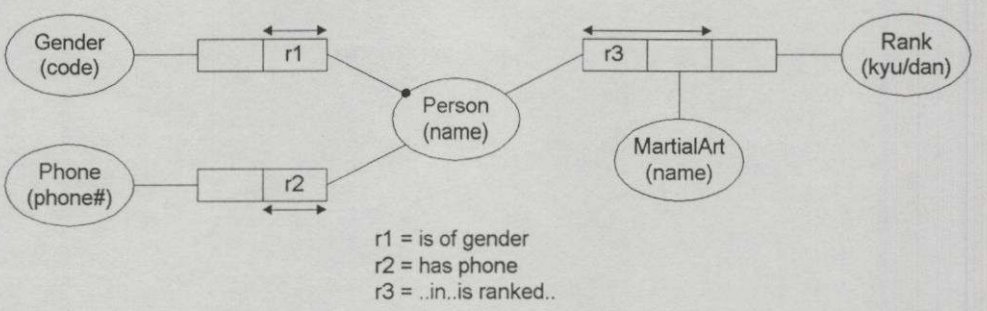
2.3.1 แบบจำลองข้อมูลแบบรีเลชันแนล

แบบจำลองข้อมูลแบบรีเลชันแนล มีโครงสร้างข้อมูลที่สำคัญคือตารางซึ่งใช้ในการแสดงความสัมพันธ์ และในแต่ละตารางจะประกอบด้วยเซ็ตของ tuple ซึ่งเรามักจะเรียกว่า rows หรือ records และหากเราดูตารางให้ดีจะเห็นว่าเราได้แบ่งตามแนวนอนเป็น rows แบ่งตามแนวตั้งเป็นคอลัมน์ โดยที่ทั้งตารางและคอลัมน์นั้นจะเป็นชื่อ ในขณะที่ rows นั้นไม่ใช่ และหากเรานำทั้ง row และคอลัมน์มาอินเตอร์เซ็ค (Intersection) เราก็จะได้ฟิลด์ (Field) ซึ่งจะเป็นที่เก็บข้อมูล 1 ข้อมูลซึ่งอาจจะเป็น NULL ได้และสำหรับข้อมูลในแต่ละฟิลด์ในตารางจะต้องเป็นชนิดเดียวกัน

เพื่อที่จะแสดงรูปแบบของโครงสร้างข้อมูลแบบรีเลชันแนล เราจะลองมาดูข้อมูลในตารางที่ 2.19 ซึ่งเราจะใช้เป็นตัวอย่างในการอธิบาย ซึ่งข้อมูลในตารางนั้นเป็นข้อมูลของสมาชิกในสโมสรแห่งหนึ่ง ซึ่งจะสังเกตได้ว่ามีข้อมูลที่เป็น NULL อยู่หลายตัวซึ่งจะหมายถึงว่าไม่ได้รับการบันทึก เช่น สมาชิกบางคนอาจจะไม่มีเบอร์โทรศัพท์ และจากข้อมูลดังกล่าว เราสามารถหา Conceptual Schema ได้ดังรูปที่ 2.31

ตารางที่ 2.19 ข้อมูลสมาชิกของสโมสรแห่งหนึ่ง

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	?	?	?



รูปที่ 2.31 แบบจำลองข้อมูลที่ได้จากตารางที่ 2.19

ตารางที่ 2.20 ตารางที่ได้จากแบบจำลองข้อมูล

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

แบบจำลองข้อมูลดังกล่าวประกอบด้วย 3 ชนิดความจริง ดังนั้นในฐานข้อมูลในระดับแนวคิด (Conceptual Database) เราก็จะได้ 3 ตารางความจริง (fact table) โดยที่แต่ละ row ก็จะเก็บแต่ละ Elementary Facts แต่สำหรับในฐานข้อมูลแบบรีเลชันแนลแล้ว เราจะรวมชนิดความจริงที่เป็น Binary เข้าด้วยกันดังนั้น เราจะได้ตารางขึ้นมา 2 ตารางดังตารางที่ 2.20

2.3.2 Optimal Normal Form Algorithm

ตารางที่ได้จากการแปลงแบบจำลองข้อมูลที่เราได้กล่าวมาแล้ว เป็นตารางที่แปลงมาอย่างถูกต้อง จึงสามารถเก็บข้อมูลได้อย่างครบถ้วนและไม่ซ้ำซ้อน จากนั้นเราจะนำตารางที่ได้นี้ไปสร้างบนฐานข้อมูลแบบรีเลชันแนล กำหนดรูปแบบของตาราง และการเขียนรูทีนเพื่อจัดการกับคอนส

แตนต์ต่าง 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

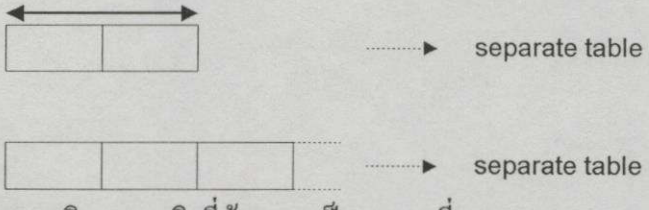
การแปลงเป็นตารางนี้ หากจะเทียบกับวิธี Normalization ซึ่งเป็นวิธีการสร้างตารางแล้ว วิธีการสร้างแบบจำลองข้อมูลและแปลงเป็นตารางที่กล่าวถึงไปแล้วจะง่ายกว่า เพราะวิธีการ Normalization จะเริ่มจาก First Normal Form และสิ้นสุดที่ Fifth Normal Form ในขณะที่ ONF สามารถที่จะสร้าง Fifth Normal Form ได้เลย และสำหรับ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

1. สำหรับแต่ละชนิดความจริงที่ไม่ได้มี simply key ให้สร้างเป็นตารางแยกออกมา โดยเลือกคีย์ที่สั้นที่สุดของชนิดความจริงนั้นเป็นคีย์หลัก

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

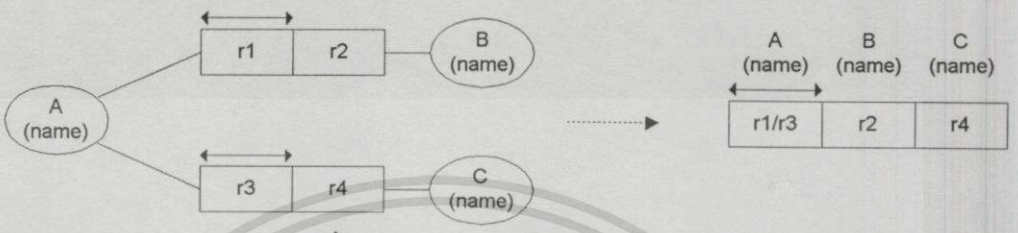
3. สำหรับชนิดความจริงที่เหลือให้สร้างเป็นตารางแยกต่างหาก

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 2.32 รูปบนเป็นไบนารีแบบ many to many ส่วนข้างล่างเป็นชนิดความจริงที่มี role มากกว่า 2 ในลักษณะดังรูปนี้จะต้องแยกออกเป็นตารางเดี่ยวออกมา



รูปที่ 2.32 ตัวอย่างของชนิดความจริงที่ต้องแยกเป็นตารางเดี่ยวออกมา

สำหรับเงื่อนไขที่ 2 นั้นกล่าวว่าหากมีชนิดเอนติตี้ใดที่มี ชนิดความจริงที่เป็น simply key เชื่อมอยู่ เราจะรวมชนิดความจริงเป็นหนึ่งตาราง และให้ชนิดเอนติตี้นั้นเป็นคีย์หลัก เช่นจากรูปที่ 2.33 ชนิดเอนติตี้ A มี 2 ชนิดความจริงที่มี simply key เชื่อมอยู่ดังนั้นเราจะรวม 2 ชนิดความจริงนี้เป็นหนึ่งตารางดังรูป



รูปที่ 2.33 การรวม 2 ชนิดความจริงที่มี simple key เดียวกัน

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

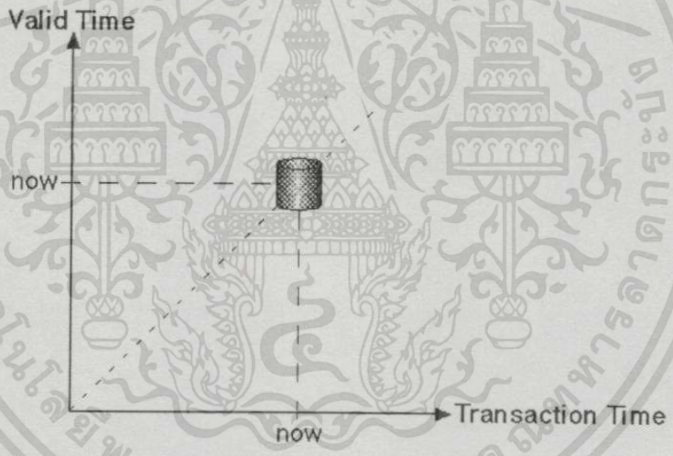
หลังจากที่ออกแบบและสร้างตารางแล้ว เนื่องจากระบบฐานข้อมูลรีเลชันแนลทั่วไป ไม่มีความสามารถในการควบคุมตามกฎข้อบังคับที่ได้ตั้งเอาไว้ ดังนั้นหากคุณต้องการบังคับข้อมูลให้เป็นไปตามที่กำหนดเอาไว้ คุณก็จะต้องเขียนโปรแกรมเพื่อควบคุมกฎข้อบังคับเอง

บทที่ 3

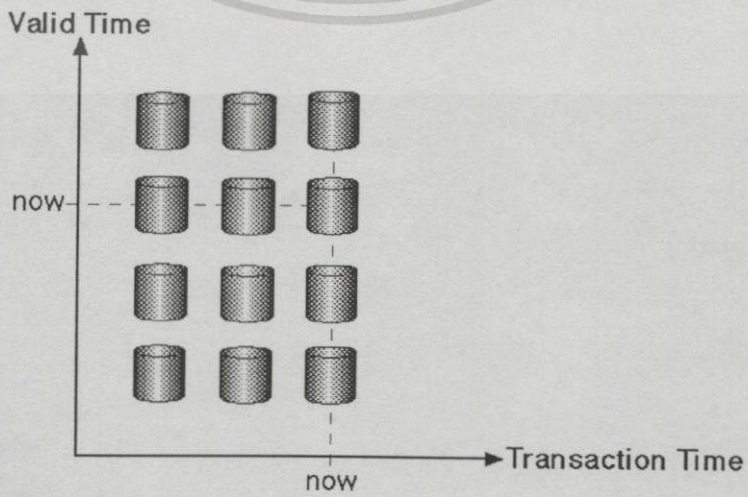
ฐานข้อมูลเชิงเวลา

3.1 ฐานข้อมูลเชิงเวลา (Temporal Database)

ระบบฐานข้อมูลที่ใช้งานกันอยู่ทั่วไปมักจะเป็นระบบฐานข้อมูลเชิงสัมพันธ์ ซึ่งมีการจัดเก็บข้อมูลเพียงสถานะเดียวเท่านั้น เมื่อมีการแก้ไขข้อมูล ข้อมูลเก่าจะถูกลบออกจากฐานข้อมูล และแทนข้อมูลใหม่ลงไปแทน แม้ว่าระบบฐานข้อมูลเดิมนี้จะรองรับโปรแกรมประยุกต์บางโปรแกรมได้ดี แต่ก็ยังไม่เพียงพอที่จะใช้งานกับ โปรแกรมประยุกต์ที่ต้องการเรียกค้นข้อมูลหลาย ๆ สถานะ อาทิ โปรแกรมทางการแพทย์ โปรแกรมทางด้านธุรกิจ ดังนั้นเราจึงต้องการระบบฐานข้อมูลที่สามารถเรียกค้นข้อมูลที่แปรผันตามเวลาได้ ซึ่งก็คือระบบฐานข้อมูลที่เก็บข้อมูลหลาย ๆ สถานะ หรือเรียกว่า ระบบฐานข้อมูลเชิงเวลา (Temporal Database)[1][11] นั่นเอง ดังรูปที่ 3.1 และ รูปที่ 3.2



รูปที่ 3.1 การจัดเก็บข้อมูลของระบบฐานข้อมูลปกติ



รูปที่ 3.2 การจัดเก็บข้อมูลของระบบฐานข้อมูลเชิงเวลา

จากรูปที่ 3.1 และรูปที่ 3.2 เราสามารถเห็นความแตกต่างของระบบฐานข้อมูลทั้งสองได้ ข้อมูลที่เก็บอยู่ใน Temporal Database นั้นแตกต่างจากข้อมูลที่ถูกเก็บอยู่ในฐานข้อมูลปกติ ตรงที่ระบบฐานข้อมูลเชิงเวลาเก็บช่วงเวลาไว้กับข้อมูล (Timestamp)[9] เพื่อแสดงว่าข้อมูลถูกเก็บในฐานข้อมูลเมื่อใดหรือข้อมูลเป็นจริงในช่วงเวลาใด โดยเวลา[11][12]ที่เก็บใน Temporal Database มี 3 แบบคือ

3.1.1 Valid Time คือ เวลาที่ข้อมูลเป็นจริง ซึ่งจะแบ่งแยกย่อยออกเป็น

- ValidTimeStart คือเวลาเริ่มต้นที่ข้อมูลนั้นเป็นจริง
- ValidTimeEnd คือเวลาสุดท้ายที่ข้อมูลนั้นเป็นจริง

3.1.2 Transaction Time คือ เวลาที่ข้อมูลถูกเก็บใน Database โดยสามารถแบ่งออกเป็น

- Transaction Time Start คือเวลาที่เริ่มจัดเก็บข้อมูลลงในฐานข้อมูล
- Transaction Time End คือเวลาที่แก้ไขหรือลบข้อมูล

3.1.3 User Defined Time คือ เวลาที่ผู้ใช้งานสร้างขึ้นเพื่อใช้งาน ซึ่งผู้ใช้งานต้องจัดการและเรียกค้นเอง ระบบฐานข้อมูลจะไม่จัดการให้อย่างอัตโนมัติ หรือกล่าวอีกนัยหนึ่งได้ว่า เวลาชนิดนี้จะเป็นเวลาที่เป็นจริงเสมอ เช่น วันเกิด วันเข้าทำงาน และวันที่เกษียร เป็นต้น

สำหรับตัวอย่างของ Valid Time และ Transaction Time นั้นคือ ถ้านาย บัณฑิต พัสยา เป็นนักศึกษาตั้งแต่วันที่ 1 มิถุนายน ค.ศ. 1994 ถึงวันที่ 30 พฤษภาคม ค.ศ. 1998 แต่ข้อมูลนี้ถูกป้อนเข้าฐานข้อมูลในวันที่ 20 มิถุนายน ค.ศ. 1994 โดยที่ข้อมูลนี้ถูกลบออกจากฐานข้อมูลในวันที่ 25 มิถุนายน ค.ศ. 1998 เราสามารถแจกแจงเวลาได้ดังนี้

- ValidTimeStart คือ วันที่ 1 มิถุนายน ค.ศ. 1994
- ValidTimeEnd คือ วันที่ 30 พฤษภาคม ค.ศ. 1998
- Transaction Time Start คือ วันที่ 20 มิถุนายน ค.ศ. 1994
- Transaction Time End คือ วันที่ 25 มิถุนายน ค.ศ. 1998

สังเกตว่าช่วงเวลา 3 ช่วงนี้ไม่จำเป็นต้องเป็นช่วงเวลาเดียวกัน ตัวอย่างเช่น สมมติว่ามี Temporal Database ที่เก็บข้อมูลของศตวรรษที่ 18 Valid Time ของข้อมูลคือช่วงเวลาใดๆระหว่างปี 1700 และ 1799 ส่วน Transaction Time จะเป็นเวลาที่เราเก็บข้อมูลเข้าไปในฐานข้อมูล ถ้า ValidTimeStart เป็นอดีตแต่ข้อมูลที่ใส่เข้าไปยังไม่ทราบ ValidTimeEnd เราจึงกำหนดให้ ValidTimeEnd เป็น NOW และให้ ValidTimeEnd เป็น NULL เมื่อ ValidTimeStart เป็นอนาคต

$$T_E := \text{NOW}, \text{if } T_S \leq \text{NOW}$$

$$T_E := \text{NULL}, \text{if } T_S > \text{NOW}$$

จะเห็นได้ว่า ช่วงเวลาวาลิดไทม์และช่วงเวลาทรานส์แอกชันไทม์ ไม่จำเป็นต้องเป็นช่วงเวลาเดียวกันสำหรับข้อมูลตัวเดียวกัน ตัวอย่างเช่น ฐานข้อมูลเก็บชื่อของพนักงานคนหนึ่งไว้ตั้งแต่ปี 2540หากพนักงานคนนี้มีการเปลี่ยนแปลงชื่อในปี 2543 แต่มาตรวจพบว่าชื่อมีการเปลี่ยนแปลง

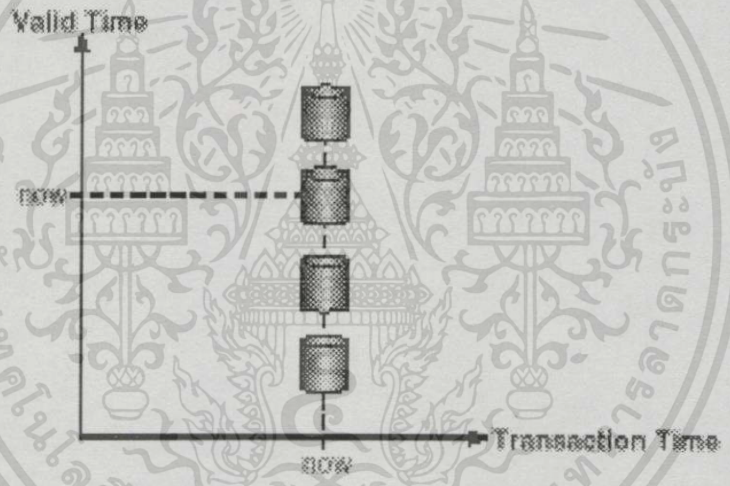
ในปี 2544 จะทำให้วาลิดไทม์ กับ ทรานส์แอกชันไทม์ เป็น 2543 และ 2544 ตามลำดับ ซึ่งมีค่าไม่เท่ากัน

3.2 ประเภทของฐานข้อมูลเชิงเวลา

ช่วงเวลาวาลิดไทม์ และทรานส์แอกชันไทม์ จะทำให้เกิดฐานข้อมูลในหลายประเภท [11] ดังนี้

3.2.1 ฐานข้อมูลวาลิดไทม์ (Valid-Time Database)

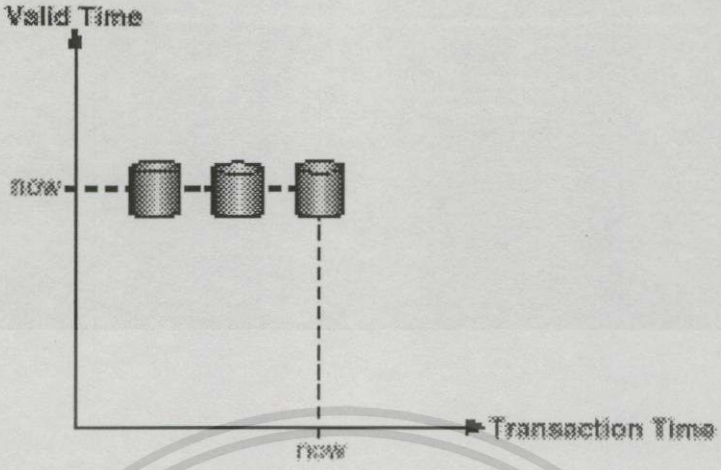
ฐานข้อมูลประวัติ (Historical Database) จะเก็บเฉพาะวาลิดไทม์ ซึ่งจะเก็บข้อมูลในอดีตกับปัจจุบัน แสดงดังรูปที่ 3.3



รูปที่ 3.3 ฐานข้อมูลประวัติ

3.2.2 ฐานข้อมูลทรานส์แอกชันไทม์ (Transaction-Time Database)

ฐานข้อมูลย้อนกลับ (Rollback Database) ซึ่งจะเก็บเฉพาะทรานส์แอกชันไทม์ แสดงดังรูปที่ 3.4

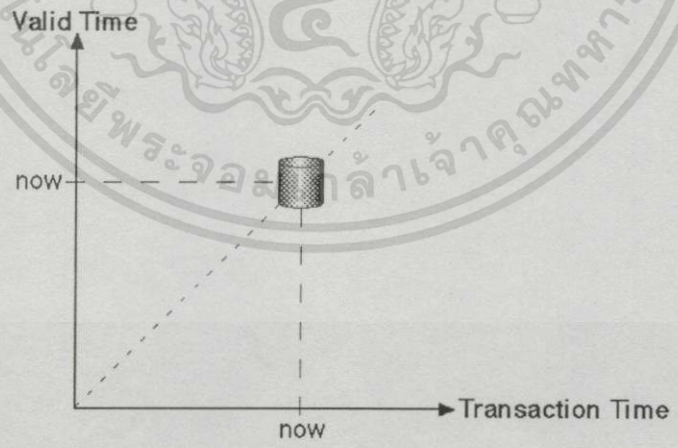


รูปที่ 3.4 ฐานข้อมูลย้อนกลับ

3.2.3 ฐานข้อมูลไบเทมโพรอล (Bitemporal Database)

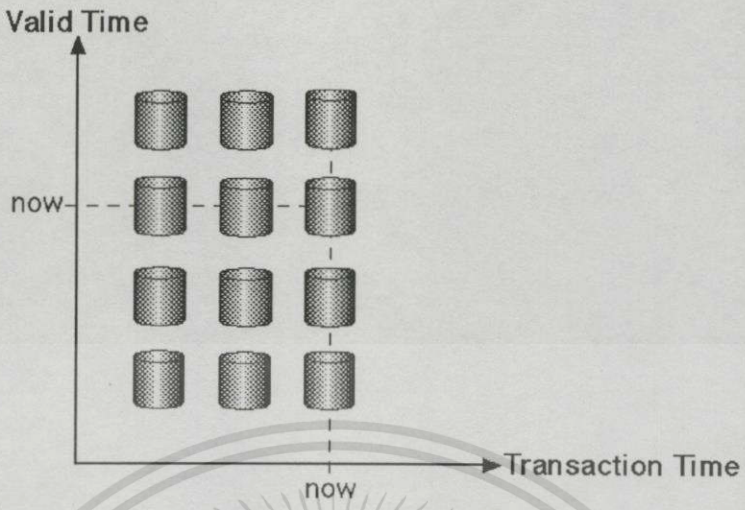
เก็บทั้งวาลิดไทม์และทรานส์แอกชันไทม์ จะทำให้เก็บข้อมูลในอดีต อนาคต และปัจจุบันได้

ในระบบจัดการฐานข้อมูลเชิงพาณิชย์จะเก็บเฉพาะข้อมูลที่เป็นจริงในปัจจุบันเท่านั้น ฐานข้อมูลชนิดนี้จะเรียกว่าฐานข้อมูลสแนปช็อท (Snapshot Database) โดยฐานข้อมูลสแน็ปช็อทในแบบของวาลิดไทม์และทรานส์แอกชันไทม์ แสดงดังรูปที่ 3.5



รูปที่ 3.5 ฐานข้อมูลสแน็ปช็อท

และฐานข้อมูลแบบไบเทมโพรอล อาจจะเป็นได้ทั้งตารางสแน็ปช็อท คือเก็บตารางวาลิดไทม์ คือเก็บว่าข้อมูลเป็นจริงเมื่อใด โดยมีลักษณะฐานข้อมูล หรือเก็บตารางทรานส์แอกชันไทม์ คือ เก็บว่าข้อมูลถูกบันทึกลงในฐานข้อมูลเมื่อใด หรืออาจจะเป็นตารางไบเทมโพรอล ที่เก็บทั้งวาลิดไทม์และทรานแซคชันไทม์ก็ได้ โดยมีลักษณะของฐานข้อมูลดังรูปที่ 3.6



รูปที่ 3.6 ฐานข้อมูลไบเทมโพรอล

3.3 ความซ้ำซ้อน (Duplicates)

ตารางที่ 3.1 ตารางวาลิดไทม์ของข้อมูลเด็กในโรงพยาบาลแห่งหนึ่ง

Name	Status	FROM_DATE	TO_DATE
Kenneth Robert	Serious	1997-11-19	1997-11-21
Alexis May	Serious	1997-11-19	1997-11-27
Natalie Sue	Serious	1997-11-19	1997-11-25
Kelsey Ann	Serious	1997-11-19	1997-11-26
Joel Steven	Critical	1997-11-19	1997-11-20
Joel Steven	Serious	1997-11-20	1997-11-26
Kenneth Robert	Fair	1997-11-21	1998-01-03
Alexis May	Fair	1997-11-27	1998-01-11
Alexis May	Fair	1997-12-02	9999-12-31
Alexis May	Fair	1997-12-02	9999-12-31

ตารางที่ 3.1 เป็นข้อมูลของเด็กในโรงพยาบาลแห่งหนึ่งที่มีอาการต่างๆ กัน ในช่วงเวลาต่างๆ โดยมีคอลัมน์ From_date และ To_date เป็นช่วงเวลา (Timestamp) ที่ข้อมูลในแถวนั้นๆ มีค่าวาลิด ซึ่งถ้าค่า To_date เป็น "9999-12-31" จะหมายถึงข้อมูลในแถวนั้นมีค่าวาลิดมาจนถึงปัจจุบัน

จากตารางเชิงเวลานี้ จะพบว่าข้อมูลมีความซ้ำซ้อนอยู่ ซึ่งเราสามารถจำแนกความซ้ำซ้อนออกได้เป็น 4 ประเภท ดังนี้

3.3.1 ความซ้ำซ้อนแบบค่าเท่ากัน (Value Equivalent Duplicate)

เป็นความซ้ำซ้อนที่แถวนั้นมีค่าของคอลัมน์ทุกคอลัมน์เท่ากันหมด ยกเว้นคอลัมน์ช่วงเวลา (Timestamp Column) ดังเช่น 3 แถวสุดท้ายของตาราง มีความซ้ำซ้อนแบบค่าเท่ากัน เพราะ 2 คอลัมน์แรกมีค่าเท่ากัน

3.3.2 ความซ้ำซ้อนแบบนอนซีเควนต์ (Nonsequenced Duplicate)

เป็นความซ้ำซ้อนที่ทุกคอลัมน์รวมทั้งคอลัมน์ช่วงเวลา (Timestamp Column) มีค่าเท่ากันหมด ดังเช่น 2 แถวสุดท้ายของตารางมีความซ้ำซ้อนแบบนอนซีเควนต์นี้

3.3.3 ความซ้ำซ้อนแบบซีเควนต์ (Sequenced Duplicate)

เป็นความซ้ำซ้อนที่เกิดขึ้นในบางช่วงของเวลา คือ มีค่าซ้ำซ้อนกันในช่วงเวลาใดเวลาหนึ่ง ดังเช่น 3 แถวสุดท้ายของตาราง เป็นความซ้ำซ้อนแบบซีเควนต์ ในช่วง “1997-11-27” ถึง “1998-01-11” เป็นต้น

3.3.4 ความซ้ำซ้อนแบบปัจจุบัน (Current Duplicate)

เป็นความซ้ำซ้อนที่เกิดขึ้น เมื่อเทียบกับ ณ เวลาปัจจุบัน เช่น สมมติว่าวันนี้มีค่าวันที่เป็น “1998-01-06” จะทำให้ 3 แถวสุดท้ายของตารางมีความซ้ำซ้อนแบบปัจจุบันเกิดขึ้น แต่ถ้าวันนี้มีค่าวันที่เป็น “1998-01-12” จะทำให้ 2 แถวสุดท้ายมีความซ้ำซ้อนกันเท่านั้น

ตารางที่ 3.2 จะแสดงให้เห็นถึงความสัมพันธ์ของความซ้ำซ้อนแบบต่างๆ อาทิเช่น หากเป็นความซ้ำซ้อนแบบนอนซีเควนต์แล้ว จะเป็นความซ้ำซ้อนแบบซีเควนต์ และความซ้ำซ้อนแบบปัจจุบันด้วย เป็นต้น

ตารางที่ 3.2 ความสัมพันธ์ของความซ้ำซ้อนแบบต่างๆ

ความซ้ำซ้อน	ซีเควนต์	ปัจจุบัน	ค่าเท่ากัน	นอนซีเควนต์
ซีเควนต์	✓		✓	
ปัจจุบัน	✓	✓	✓	
ค่าเท่ากัน			✓	
นอนซีเควนต์	✓		✓	✓

3.4 การป้องกันความซ้ำซ้อน

เมื่อนำเวลามาเพิ่มไว้ในตารางโดยการเพิ่มคอลัมภ์เข้าไปนั้น ทำให้เกิดความซ้ำซ้อนขึ้นในตาราง การป้องกันความซ้ำซ้อนสามารถทำได้ตอนสร้างตารางโดยจำแนกตามประเภทของความซ้ำซ้อนได้ ดังนี้

3.4.1 การป้องกันความซ้ำซ้อนแบบค่าเท่ากัน

จากตารางที่แล้ว เราสามารถเขียนภาษา SQL ในการสร้างตารางที่ป้องกันความซ้ำซ้อนแบบค่าเท่ากัน ได้ดังนี้

```
CREATE TABLE NICUStatue (
        Name            CHAR (15),
        Status          CHAR (8),
        From_date       DATE,
        To_date         DATE,
        UNIQUE ( Name, Status ) );
```

ซึ่งเป็นการบังคับว่าคอลัมน์ทุกคอลัมน์ ยกเว้นคอลัมน์ที่เป็น Timestamp คอลัมน์ต้องเป็นเอกลักษณ์ (Unique)

3.4.2 การป้องกันความซ้ำซ้อนแบบนอนซีเควนซ์

```
CREATE TABLE NICUStatue (
        . . .
        UNIQUE ( Name, Status, From_date, To_date ) );
```

ซึ่งเป็นการบังคับความถูกต้องให้ทุกคอลัมน์เป็นเอกลักษณ์

3.4.3 การป้องกันความซ้ำซ้อนแบบปัจจุบัน

```
CREATE TABLE NICUStatus (
        . . .
CHECK (NOT EXISTS (SELECT N1.SSN
                   FROM NICUStatus AS N1
                   WHERE 1 < (SELECT COUNT(Name)
                     FROM NICUStatus AS N2
                     WHERE N1.Name = N2.Name AND N1.Status = N2.Status
```

```
                AND N1.from_date <= CURRENT_DATE
                AND CURRENT_DATE < N1.to_date
                AND N2.from_date <= CURRENT_DATE
                AND CURRENT_DATE < N2.to_date )))
```

ซึ่งลักษณะของการสร้างตารางจะบังคับความถูกต้องว่าค่าข้อมูลในปัจจุบันนั้นมีค่าเดียว โดยจะไม่ซ้ำซ้อนกับข้อมูล ณ เวลาปัจจุบัน แต่ข้อมูลอาจมีการซ้ำซ้อนกันได้ในอนาคตแม้ตารางจะไม่มีการเปลี่ยนแปลง แต่หากทราบว่าแอพพลิเคชันไม่มีการจัดเก็บข้อมูลที่เป็นอนาคต เราสามารถบังคับเอกลักษณ์ของตาราง ได้โดย

```
CREATE TABLE NICUStatue (
    . . .
    UNIQUE ( Name, Status, To_date ) );
```

เนื่องจากข้อมูลในปัจจุบันจะมีค่า to_dateเดียวกัน คือ "9999-12-31"

3.4.4 การป้องกันความซ้ำซ้อนแบบซีเควนต์

```
CREATE TABLE NICUStatus (
    . . .
    CHECK (NOT EXISTS (SELECT N1.Name
        FROM NICUStatus AS N1
        WHERE 1 < (SELECT COUNT (NAME)
        FROM NICUStatus AS N2
        WHERE N1.Name = N2.Name AND N1.Status = N2.Status
        AND N1.from_date < N2.to_date AND N2.from_date <
N1.to_date ))))
```

การบังคับความซ้ำซ้อนของข้อมูลแบบซีเควนซ์ จะคล้ายกับการป้องกันความซ้ำซ้อนแบบปัจจุบัน แต่แตกต่างกันตรงเงื่อนไขในการตรวจสอบเวลาเริ่มต้นกับเวลาสิ้นสุด โดยจะตรวจสอบค่าของข้อมูลที่ช่วงเวลามีการซ้อนทับกัน

3.5 การค้นหาข้อมูลของตารางวาลิดไทม์

ในที่นี้จะยกตัวอย่างตาราง LOT_LOC ซึ่งเป็นตารางที่เก็บการเปลี่ยนแปลงของจำนวนวัวในแต่ละล็อทที่อยู่ในแต่ละคอกของแต่ละฟาร์ม ดังแสดงในตารางที่ 3.3

ตารางที่ 3.3 ตาราง LOT_LOC

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

ตารางนี้เป็นตารางวาลิดไทม์ ซึ่งบันทึกข้อมูลที่มีค่าเป็นจริงและสถานะตามแต่ละช่วงเวลา ตัวอย่างเช่น มีวัวจำนวน 17 ตัวอยู่ในคอกที่ 1 11 วัน แล้วย้ายออกเมื่อวันที่ 1998-02-18 , มีวัวจำนวน 23 ตัวจากล็อท 219 ย้ายจากคอกที่ 1 ไปคอกที่ 2 วันที่ 1998-03-01 แล้วย้ายวัว 20 ตัวที่เหลือไปคอกที่ 2 วันที่ 1998-03-14 และยังคงอยู่ในคอกนั้นจนถึงปัจจุบัน เป็นต้น

3.5.1 การตอบคำถามเชิงเวลาโดยใช้การ Projection และ Selection

3.5.1.1 การค้นหาข้อมูลโดยปกติทั่วๆ ไป (Coventional Query ; Non-temporal Query) คือการค้นหาคำถามที่ไม่คำนึงถึงเวลา เช่น "มีวัวจำนวนกี่ตัวในแต่ละคอกที่มาจากล็อทที่ 219 ฟาร์มที่ 1" จะต้องใช้ภาษา SQL ดังนี้

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

3.5.1.2 การค้นหาข้อมูลแบบปัจจุบัน (Current Query) คือ การค้นหาคำถามที่เกี่ยวข้องกับ ณ เวลาขณะหนึ่ง เช่น "มีวัวจำนวนกี่ตัว ในแต่ละคอกในปัจจุบันนี้ที่มาจากล็อทที่ 219 ฟาร์มที่ 1" จะต้องใช้ภาษา SQL ดังนี้

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

ซึ่งจะได้ผลลัพธ์ดังตารางที่ 3.4

ตารางที่ 3.4 คำตอบของตัวอย่างการตอบคำถามแบบปัจจุบัน

PEN_ID	HD_CNT
2	43

3.5.1.3 การค้นหาข้อมูลแบบซีเควนซ์ (Sequenced Query) คือ การค้นหาคำถามที่เกี่ยวข้องกับช่วงของเวลาช่วงหนึ่ง เช่น "จงแสดงประวัติตั้งแต่อดีตจนถึงปัจจุบันของจำนวนวัวในแต่ละคอกที่มาจากล็อทที่ 219 ฟาร์มที่ 1" จะต้องใช้ภาษา SQL ดังนี้

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

จะได้ผลลัพธ์ดังตารางที่ 3.5

ตารางที่ 3.5 คำตอบของตัวอย่างการตอบคำถามแบบซีเควนซ์

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

3.5.1.4 การค้นหาข้อมูลแบบนอนซีเควนต์ (Nonsequenced Query) คือ การค้นหาคำถามที่ไม่คำนึงถึงช่วงเวลา ซึ่งจะมีลักษณะคล้ายกับการค้นหาข้อมูลโดยปกติทั่วๆ ไป (Nontemporal Query) เช่น "ณ เวลาใดๆ มีวัวกี่ตัวในแต่ละคอกที่มาจากล็อทที่ 219 ในฟาร์มที่ 1" จะต้องใช้ภาษา SQL ดังนี้

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

จะได้ผลลัพธ์ดังตารางที่ 3.6

ตารางที่ 3.6 คำตอบของตัวอย่างการตอบคำถามแบบนอนซีเควนซ์

PEN_ID	HD_CNT
1	43
1	20
2	23
2	43

3.5.2 การค้นหาข้อมูลโดยใช้การจอย (Join) กันของตาราง

3.5.2.1 ค้นหาข้อมูลแบบปกติทั่ว ๆ ไปโดยใช้การจอย (Conventional Join Query) พิจารณาคำถามที่ว่า "ในแต่ละคอกมีวัวล็อทใดบ้างที่อยู่ด้วยกัน" จะต้องใช้ SQL ดังต่อไปนี้

```
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
    AND L1.FDYD_ID = L2.FDYD_ID
    AND L1.PEN_ID = L2.PEN_ID
```

3.5.2.2 ค้นหาข้อมูลแบบปัจจุบัน โดยใช้การจอย (Current Join Query) พิจารณาคำถามที่ว่า "ในแต่ละคอกในปัจจุบันมีวัวล็อทใดบ้างที่อยู่ด้วยกัน" จะต้องใช้ SQL ดังต่อไปนี้

```
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
    AND L1.FDYD_ID = L2.FDYD_ID
    AND L1.PEN_ID = L2.PEN_ID
    AND L1.TO_DATE = '9999-12-31'
    AND L2.TO_DATE = '9999-12-31'
```

3.5.2.3 ค้นหาข้อมูลแบบนอนซีเควนต์โดยใช้การจอย (Nonsequenced Join Query) พิจารณาคำถามที่ว่า "ณ เวลาใดๆ ในแต่ละคอกมีวัวล็อทใดบ้างที่อยู่ด้วยกัน" จะต้องใช้ SQL ดังต่อไปนี้

```
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
    AND L1.FDYD_ID = L2.FDYD_ID
    AND L1.PEN_ID = L2.PEN_ID
```

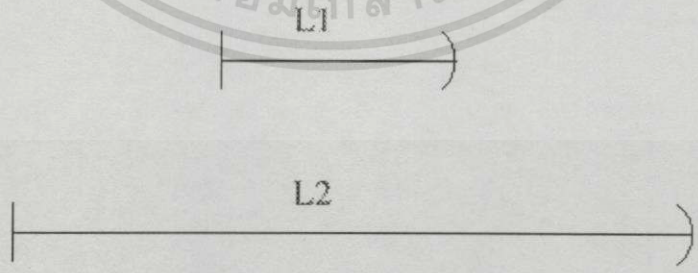
จะได้ผลลัพธ์ดังตารางที่ 3.7

ตารางที่ 3.7 คำตอบของตัวอย่างการตอบคำถามแบบนอนซีเควนซ์โดยใช้การจอย

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

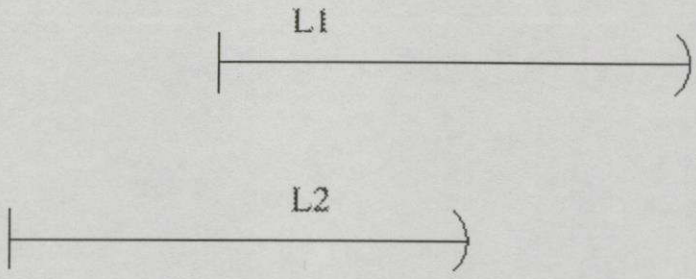
3.5.2.4 การค้นหาข้อมูลแบบซีเควนต์โดยใช้การจอย (Sequenced Join Query) พิจารณาคำถามที่ว่า "จงแสดงประวัติตั้งแต่ในอดีตจนถึงปัจจุบันว่ามีวัวล็อทใดบ้างที่อยู่ด้วยกันในแต่ละคอก" คำถามแบบนี้จะต้องแบ่งช่วงเวลาการซ้อนทับกันของตาราง ออกเป็น 4 กรณีดังนี้

กรณีที่ 1 ช่วงเวลาของตาราง L1 อยู่ในช่วงเวลาของตาราง L2 แสดงดังรูปที่ 3.7



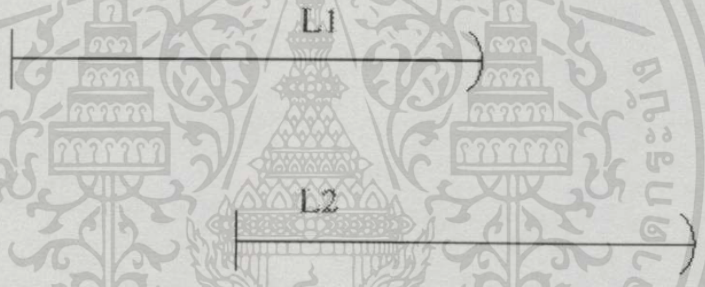
รูปที่ 3.7 กรณีที่ 1 ของการค้นหาข้อมูลแบบซีเควนต์โดยใช้การจอย

กรณีที่ 2 FROM_DATE ของตาราง L1 มากกว่า FROM_DATEของตาราง L2 แต่น้อยกว่า TO_DATE ของตาราง L2 และTO_DATE ของตาราง L1 มากกว่า TO_DATE ของตาราง L2 แสดงดังรูปที่ 3.8



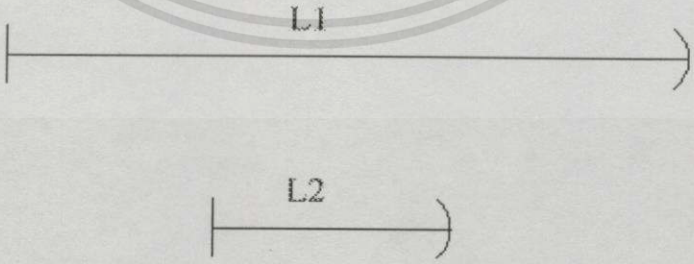
รูปที่ 3.8 กรณีที่ 2 ของการค้นหาข้อมูลแบบซีเควนต์โดยใช้การจอย

กรณีที่ 3 FROM_DATE ของตาราง L2 มากกว่า FROM_DATE ของตาราง L1 แต่น้อยกว่า TO_DATE ของตาราง L1 และ TO_DATE ของตาราง L2 มากกว่า TO_DATE ของตาราง L1 แสดงดังรูปที่ 3.9



รูปที่ 3.9 กรณีที่ 3 ของการค้นหาข้อมูลแบบซีเควนต์โดยใช้การจอย

กรณีที่ 4 ช่วงเวลาของตาราง L2 อยู่ในช่วงเวลาของตาราง L1 แสดงดังรูปที่ 3.10



รูปที่ 3.10 กรณีที่ 4 ของการค้นหาข้อมูลแบบซีเควนต์โดยใช้การจอย

ซึ่งสามารถใช้ SQL ในการตอบคำถามดังต่อไปนี้

```
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID, L1.FROM_DATE,
       L1.TO_DATE
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
       AND L1.FDYD_ID = L2.FDYD_ID
       AND L1.PEN_ID = L2.PEN_ID
       AND L2.FROM_DATE <= L1.FROM_DATE
       AND L1.TO_DATE <= L2.TO_DATE
UNION
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID, L1.FROM_DATE,
       L2.TO_DATE
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
       AND L1.FDYD_ID = L2.FDYD_ID
       AND L1.PEN_ID = L2.PEN_ID
       AND L1.FROM_DATE > L2.FROM_DATE
       AND L2.TO_DATE < L1.TO_DATE
       AND L1.FROM_DATE < L2.TO_DATE
UNION
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID, L2.FROM_DATE,
       L1.TO_DATE
FROM LOT_LOC AS L1, LOT_LOC AS L2
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
       AND L1.FDYD_ID = L2.FDYD_ID
       AND L1.PEN_ID = L2.PEN_ID
       AND L2.FROM_DATE > L1.FROM_DATE
       AND L1.TO_DATE < L2.TO_DATE
       AND L2.FROM_DATE < L1.TO_DATE
UNION
SELECT L1.LOT_ID_NUM, L2.LOT_ID_NUM, L1.PEN_ID, L2.FROM_DATE,
       L2.TO_DATE
FROM LOT_LOC AS L1, LOT_LOC AS L2
```

```
WHERE L1.LOT_ID_NUM < L2.LOT_ID_NUM
        AND L1.FDYD_ID = L2.FDYD_ID
        AND L1.PEN_ID = L2.PEN_ID
        AND L2.FROM_DATE >= L1.FROM_DATE
        AND L2.TO_DATE <= L1.TO_DATE
```

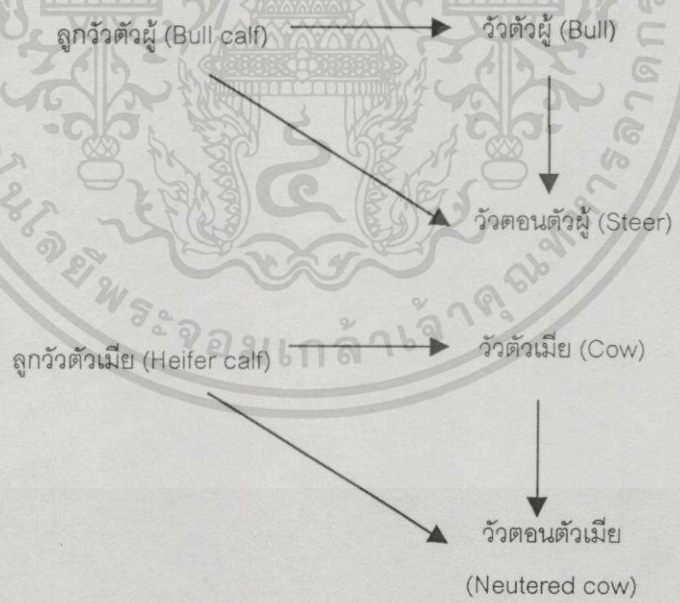
จะได้ผลลัพธ์ดังตารางที่ 3.8

ตารางที่ 3.8 คำตอบของตัวอย่างการตอบคำถามแบบซีเควนซ์โดยใช้การจอย

LOT_ID_NUM	LOT_ID_NUM	PEN_ID	FROM_DATE	TO_DATE
219	374	1	1998-02-25	1998-02-28
219	374	1	1998-03-01	1998-03-13

3.6 การแก้ไขข้อมูลเชิงเวลา

รูปที่ 3.11 จะแสดงถึงการเปลี่ยนแปลงเพศของวัว



รูปที่ 3.11 การเปลี่ยนแปลงเพศของวัว

เพื่อแสดงให้เห็นถึงการเปลี่ยนแปลงตารางวาลิดไทม์โดยใช้ภาษา SQL จึงขอยกตัวอย่างตารางเพศวัวดังตารางที่ 3.9

ตารางที่ 3.9 ตารางเพศวัว

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

3.6.1 การเปลี่ยนแปลงแบบปัจจุบัน (Current Modification)

3.6.1.1 การลบแบบปัจจุบัน (Current Deletion)

พิจารณาการเปลี่ยนแปลงที่ว่า "วัวล็อท 234 ย้ายออกจากฟาร์ม" จะสามารถทำได้โดยใช้ SQL ดังต่อไปนี้

```
UPDATE LOT
SET TO_DATE = CURRENT_DATE
WHERE LOT_ID_NUM = 234
      AND TO_DATE >= CURRENT_DATE
      AND FROM_DATE < CURRENT_DATE
DELETE FROM LOT
WHERE LOT_ID_NUM = 234
      AND FROM_DATE > CURRENT_DATEถ้า CURRENT
DATE = '1998-10-17'
```

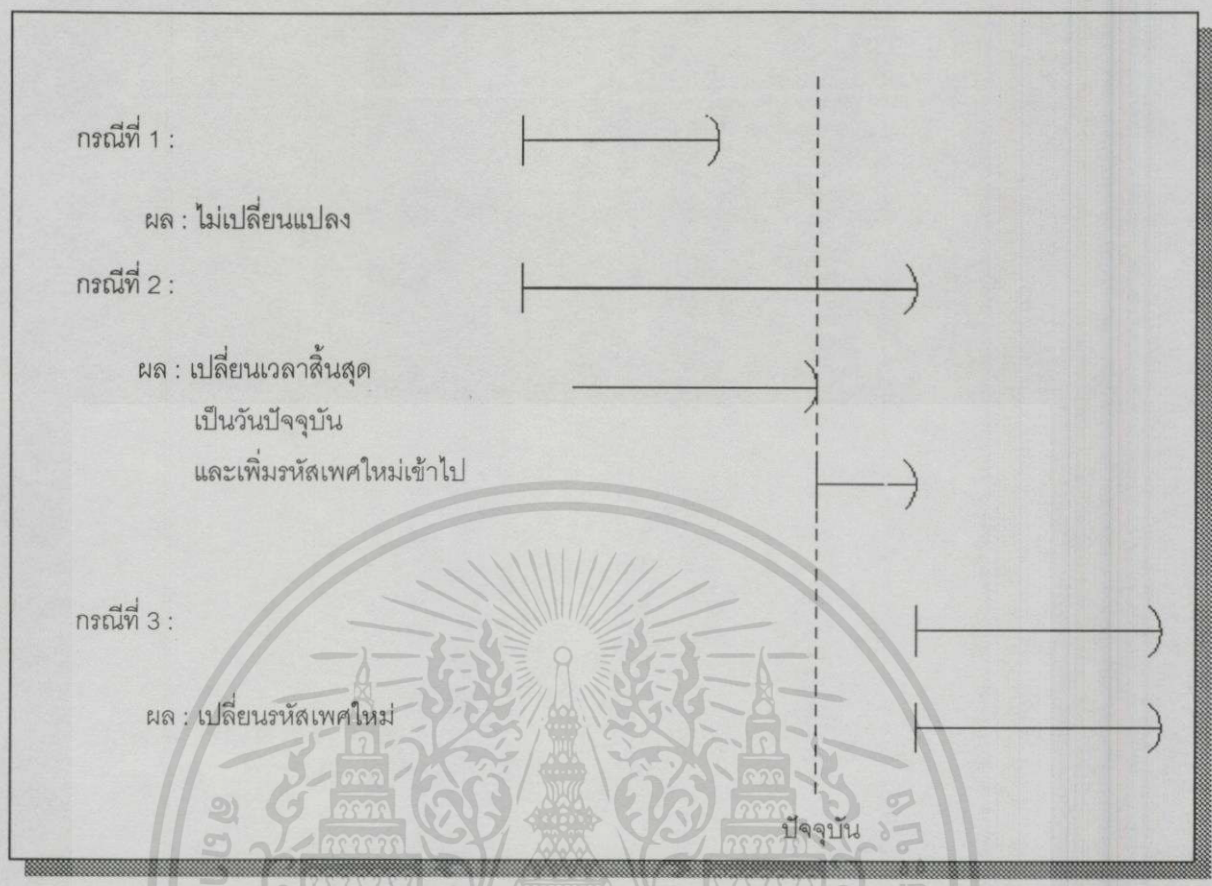
จะได้ผลดังตารางที่ 2-10

ตารางที่ 3.10 ผลของตัวอย่างการลบแบบปัจจุบัน

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

3.6.2.2 การแก้ไขแบบปัจจุบัน (Current Update)

พิจารณาการเปลี่ยนแปลงที่ว่า "เปลี่ยนวัวล็อท 799 เป็นวัวตอนตัวผู้" การเปลี่ยนแปลงแบบนี้จะต้องแบ่งออกเป็น 3 กรณี แสดงดังรูปที่ 3.12 ดังต่อไปนี้



รูปที่ 3.12 กรณีของการแก้ไขแบบปัจจุบัน

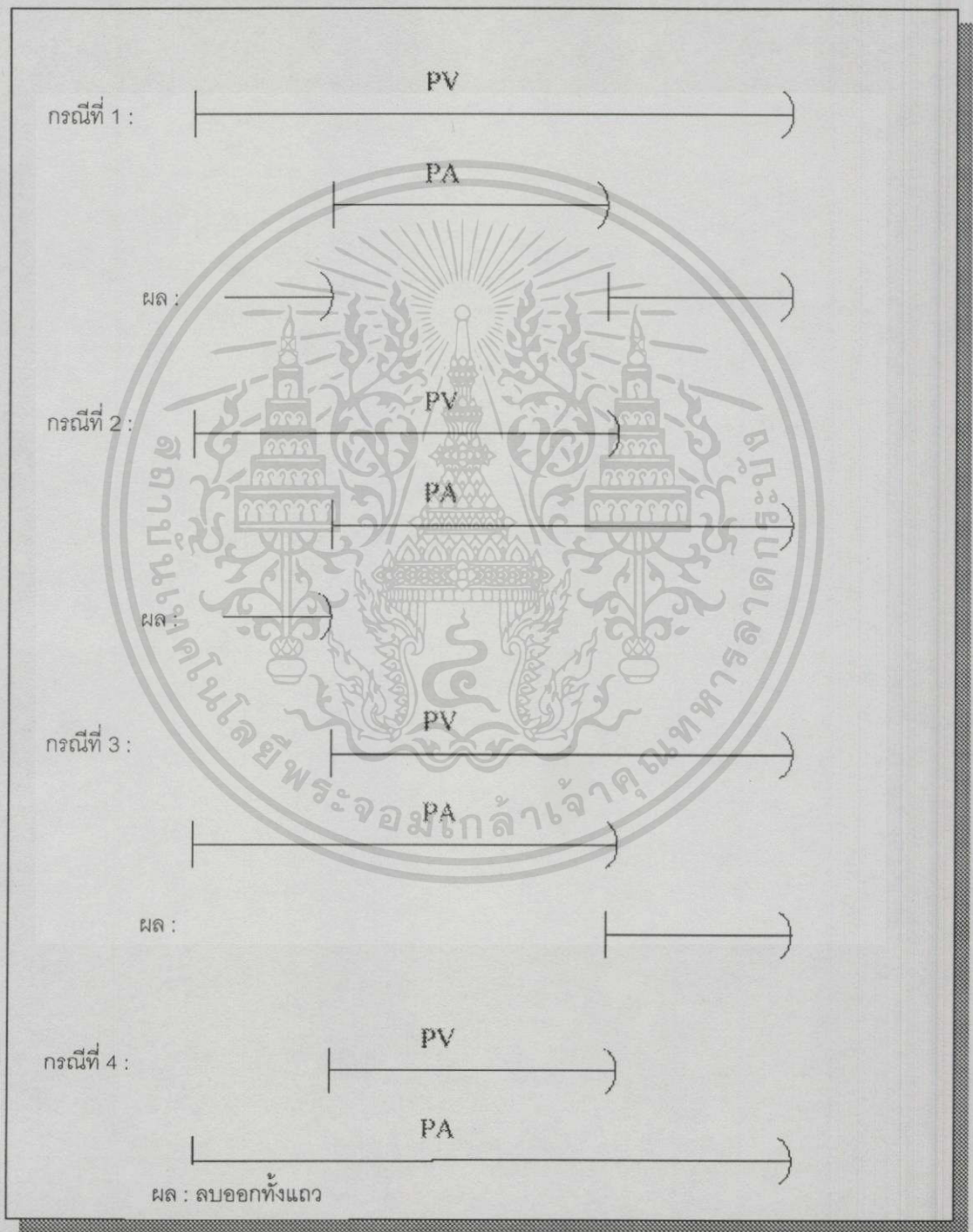
ดังนั้นจะสามารถแก้ไขแบบปัจจุบันได้โดยใช้ SQL ดังต่อไปนี้

```
INSERT INTO LOT
SELECT LOT_ID_NUM, 'S', CURRENT_DATE, TO_DATE
FROM LOT
WHERE LOT_ID_NUM = 799
        AND FROM_DATE <= CURRENT_DATE
        AND TO_DATE > CURRENT_DATE
UPDATE LOT
SET TO_DATE = CURRENT_DATE
WHERE LOT_ID_NUM = 799
        AND GNDR_CODE <> 'S'
        AND FROM_DATE < CURRENT_DATE
        AND TO_DATE > CURRENT_DATEUPDATE LOT
SET GNDR_CODE = 'S'
WHERE LOT_ID_NUM = 799 AND FROM_DATE > = CURRENT_DATE
```

3.6.2 การเปลี่ยนแปลงแบบซีเควนซ์ (Sequenced Modification)

2.6.2.1 การลบแบบซีเควนซ์ (Sequenced Deletion)

พิจารณาการเปลี่ยนแปลงที่ว่า "วัวล็อท 234 ย้ายออกจากฟาร์มใน 3 สัปดาห์แรกของเดือนตุลาคม ซึ่งในช่วงเวลานั้นได้มีการวางแผนที่จะเปลี่ยนวัวล็อท 234 นี้เป็นวัวตอนตัวผู้" การเปลี่ยนแปลงแบบนี้จะต้องแบ่งออกเป็น 4 กรณี แสดงดังรูปที่ 3.13 ดังต่อไปนี้



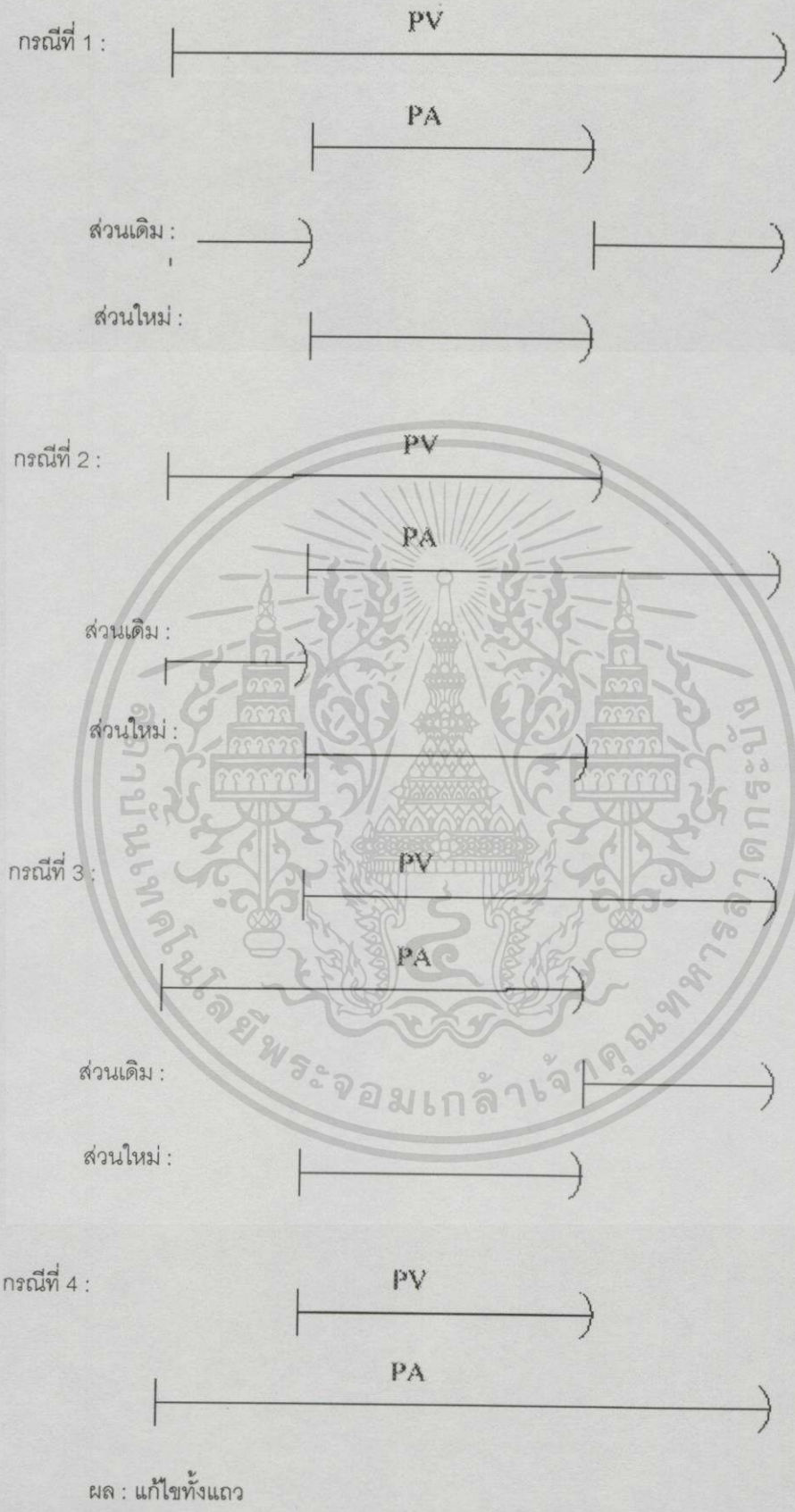
รูปที่ 3.13 กรณีของการลบแบบซีเควนซ์

โดย PV คือ ช่วงเวลาของข้อมูลเดิม (Period Of Validity) และ PA คือ ช่วงเวลาที่ต้องการเปลี่ยนแปลง (Period Of Applicability) ดังนั้นจะสามารถทำการลบแบบซีเควนต์ได้โดยใช้ SQL ดังต่อไปนี้

```
INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, '1998-10-22', TO_DATE
FROM LOT
WHERE LOT_ID_NUM = 234
        AND FROM_DATE <= '1998-10-01'
        AND TO_DATE > '1998-10-22'
UPDATE LOT
SET TO_DATE = '1998-10-01'
WHERE LOT_ID_NUM = 234
        AND FROM_DATE < '1998-10-01'
        AND TO_DATE >= '1998-10-01'UPDATE LOT
SET FROM_DATE = '1998-10-22'
WHERE LOT_ID_NUM = 234
        AND FROM_DATE < '1998-10-22'
        AND TO_DATE >= '1998-10-22'
DELETE FROM LOT
WHERE LOT_ID_NUM = 234
        AND FROM_DATE >= '1998-10-01'
        AND TO_DATE <= '1998-10-22'
```

3.6.2.2 การแก้ไขแบบซีเควนซ์ (Sequenced Update)

พิจารณาการเปลี่ยนแปลงที่ว่า "เปลี่ยนวัวล็อท 799 เป็นวัวตอนตัวผู้เฉพาะในเดือนมีนาคม" การเปลี่ยนแปลงแบบนี้จะต้องแบ่งออกเป็น 4 กรณี แสดงดังรูปที่ 3.14 ดังต่อไปนี้



รูปที่ 3.14 กรณีของการแก้ไขแบบซีเควนซ์

ดังนั้นจะสามารถทำการแก้ไขแบบซีเควนต์ได้โดยใช้ SQL ดังต่อไปนี้

```
INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, FROM_DATE, '1998-03-01'
FROM LOT
WHERE LOT_ID_NUM = 799
        AND FROM_DATE < '1998-03-01'
        AND TO_DATE > '1998-03-01'INSERT INTO LOT
SELECT LOT_ID_NUM, GNDR_CODE, '1998-04-01', TO_DATE
FROM LOT
WHERE LOT_ID_NUM = 799
        AND FROM_DATE < '1998-04-01'
        AND TO_DATE > '1998-04-01'
UPDATE LOT
SET GNDR_CODE = 'S'
WHERE LOT_ID_NUM = 799
        AND FROM_DATE < '1998-04-01'
        AND TO_DATE > '1998-03-01'UPDATE LOT
SET FROM_DATE = '1998-03-01'
WHERE LOT_ID_NUM = 799
        AND FROM_DATE < '1998-03-01'
        AND TO_DATE > '1998-03-01'
UPDATE LOT
SET TO_DATE = '1998-04-01'
WHERE LOT_ID_NUM = 799
        AND FROM_DATE < '1998-04-01'
        AND TO_DATE > '1998-04-01'
```

3.6.3 การเปลี่ยนแปลงแบบนอนซีเควนซ์ (Nonsequenced Modification)

พิจารณาการเปลี่ยนแปลงที่ว่า "ลบข้อมูลของวัวล็อท 234 ที่มีช่วงเวลามากกว่า 3 เดือน" จะสามารถทำได้โดยใช้ SQL ดังต่อไปนี้

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

3.7 การสนับสนุน Temporal ในภาษา SQL มาตรฐาน

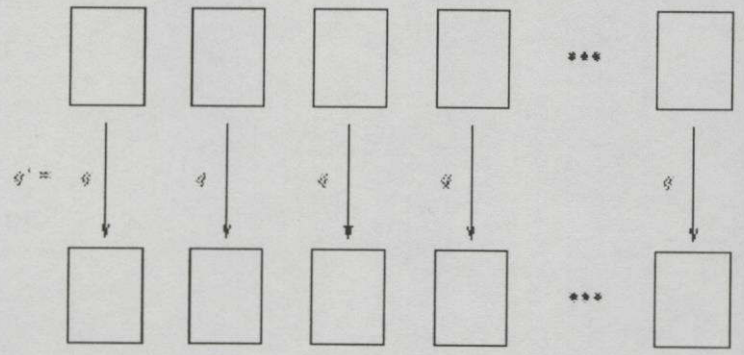
ในหัวข้อนี้จะอธิบายถึง ภาษา SQL ที่สามารถนำมาจัดการกับ ข้อมูลที่มีเวลามาเกี่ยวข้อง ถึงแม้ว่า SQL-92 นั้นจะสนับสนุนชนิดของข้อมูลที่เกี่ยวกับเวลาคือ DATE,TIME และ TIMESTAMP แต่ จริงๆแล้วSQL-92 ไม่มีความเข้าในเกี่ยวกับ ตารางที่ มีเวลามาแปรผัน จริงๆแล้ว SQL ไม่มีความคิดในหลักการของ Current หรือ Sequence query modifications หรือ views รวม ถึงไม่รู้ความแตกต่างของ Valid time และ Transaction time อีกด้วย

โชคดีที่ คณะกรรมการมาตรฐานได้พิจารณาถึงข้อจำกัดดังกล่าวจึงได้มีการสนับสนุน temporal ใน ภาษา SQL-3 และ ผู้ผลิตก็ได้รวมมาตรฐานดังกล่าวเข้าไว้ด้วยในผลิตภัณฑ์ของตัวเอง โดยที่ SQL-3 ได้เพิ่มชนิดของข้อมูลเข้าไปอีกได้แก่ PERIOD(DATE)สำหรับ วัน PERIOD(TIME) สำหรับ วินาที และ PERIOD(TIMESTAMP) สำหรับ มิลลิวินาที SQL-3 นั้น สนับสนุนทั้ง Valid-time, Transaction-time หรืออาจจะทั้ง 2 แบบพร้อมๆกัน(bitemporal)

นอกจากนั้นยังได้มีการเพิ่มทางเลือกทางด้านเวลาเข้าไปอีก 2 คำก็คือ SEQUENCED และ NONSEQUENCED สามารถใช้ร่วมกับ VALIDTIME และ TRANSACTIONTIME ทำให้แบ่งได้เป็น 4 กรณี

3.7.1 VALIDTIME SELECT

ใช้กับตารางที่สนับสนุน VALID-TIME ซึ่งจะให้ผลของการ query เป็นแบบ SEQUENCED VALIDTIME SELECT ซึ่งหมายความว่าผลลัพธ์ ของการ query จะได้มาจากแต่ละสถานะของตารางดังรูปที่ 3.15



รูปที่ 3.15 ผลลัพธ์ของการ query แบบ sequenced validtime

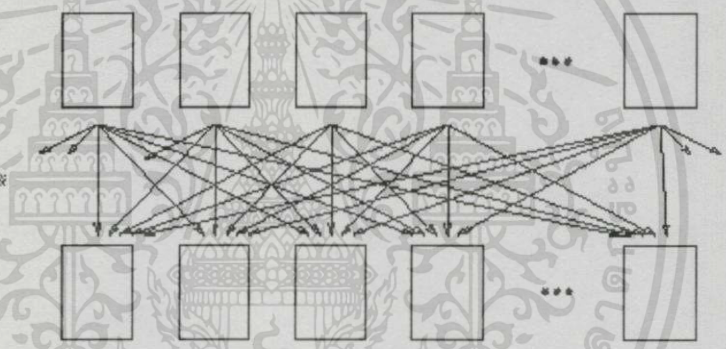
จากรูปจะเห็นได้ว่าผลลัพธ์ของการ query ก็คือ q' นั้นจะได้มาจากแต่ละสถานะ(state) ของตาราง input ที่สนับสนุน temporalโดยที่ตาราง input นั้นได้สร้างตาราง output สำหรับในแต่ละสถานะของตาราง เช่น คำถามที่ถามถึงประวัติของเงินเดือนของพนักงาน

3.7.2 VALIDTIME <period exp> SELECT

เหมือนกับ VALIDTIME SELECT แต่จะคืนค่า timestamp ภายในระยะเวลา ที่กำหนดใน period เช่น VALIDTIME PERIOD '[1995-01-01 – 1995-12-31] SELECT)

3.7.3 NONSEQUENCED VALIDTIME SELECT

มีความเป็นไปได้ว่าผลลัพธ์ของการquery ต้องการสถานะ(state)อื่นๆเพื่อมาประกอบการquery ด้วยดังรูป



รูปที่ 3.16 ผลลัพธ์จากการ query แบบ NONSEQUENCED VALIDTIME

จากรูปจะเห็นได้ว่า q ซึ่งเป็นผลของการ query นั้นได้มาจากหลายๆสถานะ(state) ของตาราง input เช่น คำถามที่ว่า "ใครมีเงินเดือนเพิ่มขึ้นบ้าง"

3.7.4 NONSEQUENCED VALIDTIME <period exp> SELECT

จะหมือนกับ NONSEQUENCED VALIDTIME SELECT แต่จะใช้ period เป็น timestamp

3.8 ภาษาเรียกค้น

ภาษาที่ใช้ในระบบฐานข้อมูลเชิงเวลานั้นคือภาษา SQL เชิงเวลา (SQL/Temporal)[2][10] ภาษา SQL/Temporal นี้เป็นมาตรฐานมีกำลังจะประกาศใช้พร้อมกับมาตรฐาน SQL3 ซึ่งได้รับการพัฒนามาจาก TSQL2 [10] โดยภาษานี้จะมีลักษณะเหมือนกับภาษา SQL ปกติที่เราใช้งานอยู่ เพียงแต่ว่าได้เพิ่มความสามารถทางด้านเวลาเข้าไป ดังตัวอย่าง

```
CREATE TABLE STUDENT
(
        STD_ID      VARCHAR(8),
        STD_NAME    VARCHAR(65),
        STD_ADD     VARCHAR(80),
) AS VALID TIME PERIOD(DATE) ;
```

คำสั่งข้างต้นเป็นคำสั่งเพื่อสร้างตารางที่ชื่อว่า STUDENT โดยมีคอลัมน์ 3 คอลัมน์คือ STD_ID, STD_NAME และ STD_ADD โดยที่คำสั่งนี้จะเพิ่มคอลัมน์เข้าไปอีก 2 คอลัมน์คือ ValidTimeStart และ ValidTimeEnd ดังตารางที่ 3.11 โดยที่ผู้ใช้ไม่ต้องสนใจเลยว่ามี 2 คอลัมน์นี้อยู่ การใช้งานก็ใช้งานเหมือนปกติระบบฐานข้อมูลจะคอยจัดการเรื่องเวลาให้

ตารางที่ 3.11 ตารางที่ได้จากภาษาเรียกค้น SQL/Temporal

STD_ID	STD_NAME	STD_ADD	ValidTimeStart	ValidTimeEnd
40621215	John Kandenski	London	05/05/1995	31/12/9999
40621245	Jean Clinnic	Bangkok	01/06/1994	02/07/1999

สำหรับตัวอย่างภาษาเรียกค้นอีกตัวอย่างคือ

```
VALIDTIME PERIOD
'[01/06/1994 – 30/05/1998)'
INSERT INTO STUDENT
VALUES ('38621212', 'BUNDIT PASAYA','Bangkok') ;
```

คำสั่งนี้เป็นการป้อนข้อมูลเข้าไปเก็บในฐานข้อมูล โดยระบุว่าข้อมูลที่ป้อนนั้นเป็นจริงในณ. วันที่ 1 มิถุนายน ค.ศ. 1994 ถึงก่อนวันที่ 30 พฤษภาคม ค.ศ. 1998 โดยไม่นับรวมวันที่ 30 ธันวาคม 1998

```
VALIDTIME
SELECT *
FROM STUDENT
WHERE STD.ID = 'BUNDIT PASAYA' ;
```

คำสั่งนี้เป็นการค้นหาข้อมูลของนักศึกษาที่ชื่อว่า BUNDIT PASAYA โดยจะแสดงข้อมูลนักศึกษาคนนี้ทุกช่วงเวลา กล่าวคือข้อมูลนักศึกษาคนหนึ่งอาจถูกจัดเก็บไว้หลายช่วงเวลา ทั้งนี้อาจ

เนื่องมาจาก การเปลี่ยนแปลงที่อยู่ การเปลี่ยนแปลงรหัสนักศึกษา หรือย้ายสาขาที่เรียนเป็นต้น แต่หากเราต้องการแสดงข้อมูลของนักศึกษาเฉพาะบางช่วงเวลาก็สามารถทำได้โดยการเปลี่ยนคำว่า VALIDTIME ให้เป็น VALIDTIME PERIOD 'ช่วงเวลา' เช่น VALIDTIME PERIOD '[20/09/1995– 01/10/1995]' เป็นต้น

3.9 ตัวอย่างในการใช้งานภาษา TSQL2

ตารางต่อไปนี้เป็นตารางที่ใช้ในการตอบคำถามเชิงเวลา (TSQL2)[10]

ตารางที่ 3.12 ตาราง EMP

ID	NAME	SALARY	GENDER	D-BIRTH	DEPT NAME	ValidTimeStart	ValidTimeEnd
1	สมชาย	20000	ชาย	07/01/1955	ของเล่น	01/02/1982	31/05/1982
1	สมชาย	30000	ชาย	07/01/1955	ของเล่น	01/06/1982	31/01/1985
1	สมชาย	40000	ชาย	07/01/1955	ของเล่น	01/02/1985	31/01/1987
1	สมชาย	40000	ชาย	07/01/1955	หนังสือ	01/04/1987	31/12/1987
1	ชาติชาย	40000	ชาย	07/01/1955	หนังสือ	01/01/1988	31/12/9999
2	สมหญิง	30000	หญิง	10/01/1960	ของเล่น	01/01/1982	31/07/1984
2	สมหญิง	40000	หญิง	10/01/1960	ของเล่น	01/08/1984	31/08/1986
2	สมหญิง	50000	หญิง	10/01/1960	ของเล่น	01/09/1986	31/12/9999

ตารางที่ 3.13 ตาราง SKILL

ID	SKILL	ValidTimeStart	ValidTimeEnd
1	พิมพ์ดีด	01/04/1982	31/12/9999
1	จัดเรียงเอกสาร	01/01/1985	31/12/9999
1	ขับรถ	01/01/1982	01/05/1982
1	ขับรถ	01/06/1984	31/05/1988
2	บริหารงาน	01/01/1982	31/12/9999

ตารางที่ 3.14 ตาราง DEPT

DEPTNAME	BUDGET	ID	ValidTimeStart	ValidTimeEnd
ของเล่น	150000	2	01/01/1982	31/07/1984
ของเล่น	200000	2	01/08/1984	31/12/1986
ของเล่น	100000	2	01/01/1987	31/12/9999
หนังสือ	50000	1	01/04/1987	31/12/9999

ภาษา TSQL2 ซึ่งเป็นภาษา SQL เชิงเวลามีคุณสมบัติในการจัดโครงสร้างตารางใหม่ (Restructuring) ซึ่งเป็นการรวมค่าที่เหมือนกันของคอลัมน์ที่เปลี่ยนแปลงตามเวลาเข้าไว้ด้วยกัน สามารถใช้ได้โดยเขียนชื่อตารางตามด้วยชื่อคอลัมน์ที่ต้องการจัดโครงสร้างตารางใหม่อยู่ในวงเล็บ ยกตัวอย่างเช่น การจัดโครงสร้างตารางใหม่ของตาราง EMP บนคอลัมน์ ID และคอลัมน์ NAME หรือ EMP (ID, NAME) จะมีลักษณะดังต่อไปนี้

ตารางที่ 3.15 ตาราง EMP

ID	NAME	SALARY	GENDER	D-BIRTH	DEPTNAME	ValidTimeStart	ValidTimeEnd
1	สมชาย	20000	ชาย	07/01/1955	ของเล่น	01/02/1982	31/05/1982
		30000	ชาย	07/01/1955	ของเล่น	01/06/1982	31/01/1985
		40000	ชาย	07/01/1955	ของเล่น	01/02/1985	31/01/1987
		40000	ชาย	07/01/1955	หนังสือ	01/04/1987	31/12/1987
1	ชาติชาย	40000	ชาย	07/01/1955	หนังสือ	01/01/1988	31/12/9999
2	สมหญิง	30000	หญิง	10/01/1960	ของเล่น	01/01/1982	31/07/1984
		40000	หญิง	10/01/1960	ของเล่น	01/08/1984	31/08/1986
		50000	หญิง	10/01/1960	ของเล่น	01/09/1986	31/12/9999

การจัดโครงสร้างตารางใหม่ของตาราง SKILL บนคอลัมน์ ID และคอลัมน์ SKILL หรือ SKILL (ID, SKILL) จะมีลักษณะดังต่อไปนี้

ตารางที่ 3.16 ตาราง SKILL

ID	SKILL	ValidTimeStart	ValidTimeEnd
1	พิมพ์ดีด	01/04/1982	31/12/9999
1	จัดเรียงเอกสาร	01/01/1985	31/12/9999
1	ขับรถ	01/01/1982	01/05/1982
		01/06/1984	31/05/1988
2	บริหารงาน	01/01/1982	31/12/9999

ภาษา TSQL2 ยังมีคุณสมบัติในการจัดโครงสร้างตารางใหม่แบบช่วงเวลา (Period) ซึ่งเป็นการรวมค่าที่เหมือนกันของคอลัมน์ที่เปลี่ยนแปลงตามเวลาที่ต่อเนื่องกันเข้าไว้ด้วยกัน ยกตัวอย่างเช่น การจัดโครงสร้างตารางใหม่แบบช่วงเวลาของตาราง SKILL บนคอลัมน์ ID และคอลัมน์ SKILL หรือ SKILL (ID, SKILL) (PERIOD) จะมีลักษณะดังต่อไปนี้

ตารางที่ 3.17 ตาราง SKILL

ID	SKILL	ValidTimeStart	ValidTimeEnd
1	พิมพ์ดีด	01/04/1982	31/12/9999
1	จัดเรียงเอกสาร	01/01/1985	31/12/9999
1	ขับรถ	01/01/1982	01/05/1982
1	ขับรถ	01/06/1984	31/05/1988
2	บริหารงาน	01/01/1982	31/12/9999

จะเห็นว่าทักษะขับรถไม่ได้อยู่รวมกันเนื่องจากช่วงเวลาไม่ได้ต่อเนื่องกัน แต่เนื่องจากภาษา SQL92 ไม่มีการจัดโครงสร้างตารางใหม่ จึงต้องทำการจัดโครงสร้างข้อมูลเองโดยการทำ Time Normalized ซึ่งจะคล้ายกับการจัดโครงสร้างตารางใหม่ โดยตารางจะเป็น Time Normal Form (TNF) ก็ต่อเมื่อเป็น BCNF (Boyce-Codd Normal Form) และไม่มี Temporal Dependency โดย Temporal Dependency จะเกิดขึ้นเมื่อข้อมูลที่มีการเปลี่ยนแปลงทางเวลาโดยไม่เกี่ยวข้องกัน 2 คอลัมน์ขึ้นไปอยู่รวมกันในตารางเดียวกันกัน ซึ่งจากตาราง EMP จะแปลงเป็น Time Normal Form ได้ 4 ตารางคือ ตาราง EMP_NAME, ตาราง EMP_SALARY, ตาราง EMP_DEPT และตาราง EMP_NOTIME ดังนี้

ตารางที่ 3.18 ตาราง EMP_NAME

ID	NAME	Gender	ValidTimeStart	ValidTimeEnd
1	สมชาย	ชาย	01/02/1982	31/12/1987
1	ชาติชาย	ชาย	01/01/1988	31/12/9999
2	สมหญิง	หญิง	01/01/1982	31/12/9999

ตารางที่ 3.19 ตาราง EMP_SALARY

ID	SALARY	ValidTimeStart	ValidTimeEnd
1	20000	01/02/1982	31/05/1982
1	30000	01/06/1982	31/01/1985
1	40000	01/02/1985	31/01/1987
1	40000	01/04/1987	31/12/9999
2	30000	01/01/1982	31/07/1984
2	40000	01/08/1984	31/08/1986
2	50000	01/09/1986	31/12/9999

ตารางที่ 3.20 ตาราง EMP_DEPT

ID	DEPTNAME	ValidTimeStart	ValidTimeEnd
1	ของเล่น	01/02/1982	31/01/1987
1	หนังสือ	01/04/1987	31/12/9999
2	ของเล่น	01/01/1982	31/12/9999

จากตาราง SKILL จะเห็นว่าตารางเป็น Time Normal Form แล้ว แต่เนื่องจากเราไม่ควรเปลี่ยนแปลงตารางเดิม ดังนั้นจึงควรมีอีกตารางหนึ่งเพิ่มขึ้นมาคือตาราง SKILL_SKILL ดังนี้

ตารางที่ 3.21 ตาราง SKILL_SKILL

ID	SKILL	ValidTimeStart	ValidTimeEnd
1	พิมพ์ดีด	01/04/1982	31/12/9999
1	จัดเรียงเอกสาร	01/01/1985	31/12/9999
1	ขับรถ	01/01/1982	01/05/1982
1	ขับรถ	01/06/1984	31/05/1988
2	บริหารงาน	01/01/1982	31/12/9999

จากตาราง DEPT จะแปลงเป็น Time Normal Form ได้ 2 ตารางคือ ตาราง DEPT_BUDGET และตาราง DEPT_MGR ดังนี้

ตารางที่ 3.22 ตาราง DEPT_BUDGET

DEPTNAME	BUDGET	ValidTimeStart	ValidTimeEnd
ของเล่น	150000	01/01/1982	31/07/1984
ของเล่น	200000	01/08/1984	31/12/1986
ของเล่น	100000	01/01/1987	31/12/9999
หนังสือ	50000	01/04/1987	31/12/9999

ตารางที่ 3.23 ตาราง DEPT_MGR

DEPTNAME	ID	ValidTimeStart	ValidTimeEnd
ของเล่น	2	01/01/1982	31/12/9999
หนังสือ	1	01/04/1987	31/12/9999

โดยฟังก์ชั่นที่ใช้สำหรับช่วยในการตอบคำถามมีดังนี้

1. ฟังก์ชั่นสำหรับแปลงวันที่ 12 เดือนธันวาคม ปี ค.ศ. 9999 ให้เป็นวันที่ปัจจุบัน

```
CREATE FUNCTION NOW (VE DATE) RETURNING DATE;
DEFINE NEWVE DATE;
IF VE = '31/12/9999' THEN LET NEWVE =  DATE (CURRENT);
ELSE LET NEWVE = VE;
END IF;
RETURN NEWVE;
END FUNCTION;
```

2. ฟังก์ชั่นสำหรับหาวันที่ที่มากกว่า

```
CREATE FUNCTION MAXDATE (DATE1 DATE, DATE2 DATE) RETURNING DATE;
DEFINE DATE3 DATE;
IF DATE1 > DATE2 THEN LET DATE3 = DATE1;
ELSE LET DATE3 = DATE2;
END IF;
RETURN DATE3;
END FUNCTION;
```

3. ฟังก์ชั่นสำหรับหาวันที่ที่น้อยกว่า

```
CREATE FUNCTION MINDATE (DATE1 DATE, DATE2 DATE) RETURNING DATE;
DEFINE DATE3 DATE;
IF DATE1 < DATE2 THEN LET DATE3 = DATE1;
ELSE LET DATE3 = DATE2;
END IF;
RETURN DATE3;
END FUNCTION;
```

ตัวอย่างการตอบคำถามเชิงเวลา

ตัวอย่างการตอบคำถามเชิงเวลาโดยใช้ภาษา TSQL2 ภาษา SQL92 บนตารางที่เป็น TNF และภาษา SQL92 มีดังนี้ (สมมติให้วันที่ปัจจุบันคือวันที่ 1 เดือนมกราคม ปี ค.ศ. 1990)

1. บอกชื่อแผนกและงบประมาณของแผนกในวันที่สมชายเข้าทำงาน

คำตอบ : {('ของเล่น', 150000), ('หนังสือ', 50000)}

ภาษา TSQL2 :

```
SELECT SNAPSHOT E.DEPTNAME, D.BUDGET
FROM EMP (NAME, DEPTNAME) AS E, DEPT (DEPTNAME, BUDGET) AS D
WHERE E.NAME = 'สมชาย'
AND E.DEPTNAME = D.DEPTNAME
AND VALID(D) CONTAINS FIRST (VALID (E))
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT E.DEPTNAME, D.BUDGET
FROM EMP_DEPT E, DEPT_BUDGET D
WHERE E.ID = (SELECT ID FROM EMP_NAME WHERE NAME = 'สมชาย')
AND E.DEPTNAME = D.DEPTNAME
AND D.VALIDTIMESTART <= E.VALIDTIMESTART
AND D.VALIDTIMEEND >= E.VALIDTIMESTART
```

ภาษา SQL92 :

```
SELECT E.DEPTNAME, D.BUDGET
FROM EMP E, DEPT D
WHERE E.NAME = 'สมชาย'
AND E.DEPTNAME = D.DEPTNAME
AND D.VALIDTIMESTART <= E.VALIDTIMESTART
AND D.VALIDTIMEEND >= E.VALIDTIMESTART
AND E.VALIDTIMESTART-1 NOT IN (SELECT E2.VALIDTIMEEND FROM EMP
E2
WHERE E.ID = E2.ID AND E.DEPTNAME = E2.DEPTNAME)
```

2. แผนกใดมีผู้จัดการคนเดิมบริหารงานติดต่อกันสั้นที่สุด

คำตอบ : {('หนังสือ')}

ภาษา TSQL2 :

```
SELECT SNAPSHOT NAME
FROM DEPT (DEPTNAME, ID) (PERIOD) AS D
WHERE CAST (VALID (D) AS INTERVAL DAY) <=
ALL (SELECT CAST (VALID (D2) AS INTERVAL DAY)
FROM DEPT (DEPTNAME, ID) (PERIOD) AS D2)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT D1.DEPTNAME
FROM DEPT_MGR D1
WHERE D1.VALIDTIMEEND - D1.VALIDTIMESTART <=
ALL (SELECT D2.VALIDTIMEEND - D2.VALIDTIMESTART FROM DEPT_MGR
D2)
```

ภาษา SQL92 :

???

3. ใครมีเงินเดือนเท่าเดิมติดต่อกันนานที่สุด

คำตอบ : {('สมหญิง')}

ภาษา TSQL2 :

```
SELECT SNAPSHOT E2.NAME
FROM EMP (ID, SALARY) (PERIOD) AS E, E (NAME) AS E2
WHERE CAST (VALID (E) AS INTERVAL DAY) >
ALL (SELECT CAST (VALID (E3) AS INTERVAL DAY)
FROM EMP (ID, SALARY) (PERIOD) AS E3
WHERE E3.ID <> E.ID)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT NAME
FROM EMP_SALARY E1 , EMP_NAME
WHERE NOW (E1.VALIDTIMEEND) - E1.VALIDTIMESTART >
ALL (SELECT NOW (E2.VALIDTIMEEND) - E2.VALIDTIMESTART
FROM EMP_SALARY E2
WHERE E1.ID <> E2.ID)
AND EMP_NAME.ID = E1.ID
```

ภาษา SQL92 :

???

4. ใครมีเงินเดือนเท่าเดิมนานที่สุด

คำตอบ : {('สมชาย')}

ภาษา TSQL2 :

```
SELECT SNAPSHOT E2.NAME
FROM EMP (ID, SALARY) AS E1, E1 (NAME) AS E2
WHERE CAST (VALID (E1) AS INTERVAL DAY) >
ALL (SELECT CAST (VALID (E3) AS INTERVAL DAY)
FROM EMP (ID, SALARY) AS E3
WHERE E3.ID <> E1.ID)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
CREATE VIEW V1 (ID, SALARY, INTERVAL) AS
SELECT ID, SALARY, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM EMP_SALARY
GROUP BY  ID, SALARY;


SELECT E2.NAME
FROM V1, EMP_NAME E2
WHERE E2.ID = V1.ID
AND V1.INTERVAL >= ALL (SELECT V2.INTERVAL FROM V1 V2);
```

```
DROP VIEW V1;
```

ภาษา SQL92 :

```
CREATE VIEW V1 (ID, SALARY, INTERVAL) AS
SELECT ID, SALARY, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM EMP
GROUP BY  ID, SALARY;

SELECT DISTINCT (E2.NAME)
FROM V1, EMP E2
WHERE E2.ID = V1.ID
AND V1.INTERVAL >= ALL (SELECT V2.INTERVAL FROM V1 V2);

DROP VIEW V1;
```

5. ใครเป็นผู้จัดการแผนกหนังสือเป็นเวลาอย่างน้อย 2 ปี

คำตอบ : {('สมชาย'), ('ชาติชาย')}

ภาษา TSQL2 :

```
SELECT SNAPSHOT NAME
FROM EMP (ID, NAME) AS E, DEPT (DEPTNAME, ID) AS D
WHERE E.ID = D.ID AND DEPT.DEPTNAME = 'หนังสือ'
AND CAST (VALID (D) AS INTERVAL YEAR) >= INTERVAL '2' YEAR
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT DISTINCT NAME
FROM EMP_NAME E1, DEPT_MGR D1
WHERE E1.ID = D1.ID
AND D1.DEPTNAME = 'หนังสือ'
AND 2 <= (SELECT SUM (YEAR (NOW (VALIDTIMEEND)) – YEAR
(VALIDTIMESTART))
FROM DEPT_MGR D2
WHERE D1.ID = D2.ID AND D1.DEPTNAME = D2.DEPTNAME
```

GROUP BY DEPTNAME, ID)

ภาษา SQL92 :

SELECT DISTINCT NAME

FROM EMP E1, DEPT D1

WHERE E1.ID = D1.ID

AND D1.DEPTNAME = 'หนังสือ'

AND 2 <= (SELECT SUM (YEAR (NOW (VALIDTIMEEND)) – YEAR (VALIDTIMESTART))

FROM DEPT D2

WHERE D1.ID = D2.ID AND D1.DEPTNAME = D2.DEPTNAME

GROUP BY DEPTNAME, ID)

6. บอกวันที่ที่สมชายมีทักษะต่างๆ

คำตอบ : {(01/01/1982), (04/01/1982), (06/01/1984), (01/01/1985)}

ภาษา TSQL2 :

SELECT BEGIN (S)

FROM EMP (ID, NAME) AS E, SKILL (ID, SKILL) AS S

WHERE E.ID = S.ID

AND E.NAME = 'สมชาย'

ภาษา SQL92 บนตารางที่เป็น TNF :

SELECT S1.VALIDTIMESTART

FROM EMP_NAME E, SKILL_SKILL S1

WHERE E.NAME = 'สมชาย'

AND E.ID = S1.ID

AND S1.VALIDTIMESTART-1 NOT IN (SELECT S2.VALIDTIMEEND FROM SKILL_SKILL S2

WHERE S1.ID = S2.ID AND S1.SKILL = S2.SKILL)

ภาษา SQL92 :

SELECT DISTINCT S1.VALIDTIMESTART

```
FROM EMP E, SKILL S1
WHERE E.NAME = 'สมชาย'
AND E.ID = S1.ID
AND S1.VALIDTIMESTART-1 NOT IN (SELECT S2.VALIDTIMEEND FROM
SKILL_SKILL S2
WHERE S1.ID = S2.ID AND S1.SKILL = S2.SKILL)
```

7. บอกช่วงเวลาในการทำงานของพนักงานที่เคยได้รับเงินเดือน 40000 บาทนานกว่าช่วงเวลาที่สมหญิงได้รับเงินเดือน 40000 บาท

คำตอบ : {([02/01/1982 – 01/31/1987]), ([04/01/1987 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT VALID (E1)
FROM EMP (ID, SALARY) AS E1 E2, E2 (NAME) AS E3
WHERE E1.SALARY = 40000 AND E2.SALARY = 40000 AND E3.NAME = 'สมหญิง'
AND CAST (VALID (E1) AS INTERVAL DAY) > CAST (VALID (E2) AS INTERVAL
DAY)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
CREATE VIEW V1 (ID, SALARY, INTERVAL) AS
SELECT ID, SALARY, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM EMP_SALARY
GROUP BY ID, SALARY;

SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM V1 E1 , EMP_DEPT E4
WHERE E1.SALARY = 40000
AND E1.INTERVAL > (SELECT E2.INTERVAL FROM V1 E2, EMP_NAME E3
WHERE E2.ID = E3.ID AND E2.SALARY = 40000 AND E3.NAME = 'สมหญิง')
AND E1.ID = E4.ID;

DROP VIEW V1;
```

ภาษา SQL92 :

```
CREATE VIEW V1 (ID, SALARY, INTERVAL) AS
SELECT ID, SALARY, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM EMP
GROUP BY  ID, SALARY;

SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM V1 E1, EMP_DEPT E4
WHERE E1.SALARY = 40000
AND E1.INTERVAL > (SELECT E2.INTERVAL FROM V1 E2, EMP_NAME E3
WHERE E2.ID = E3.ID AND E2.SALARY = 40000 AND E3.NAME = 'สมหญิง')
AND E1.ID = E4.ID;

DROP VIEW V1;
```

8. บอกช่วงเวลาที่งบประมาณของแผนกคงที่นานกว่า 2 ปี

คำตอบ : {([01/01/1987 – 01/01/1990], [01/04/1987 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT VALID (D)
FROM DEPT (DEPTNAME, BUDGET) (PERIOD) AS D
WHERE CAST (VALID (D) AS INTERVAL YEAR) > INTERVAL '2' YEAR
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM DEPT_BUDGET D
WHERE YEAR (NOW (VALIDTIMEEND)) – YEAR (VALIDTIMESTART) > 2
```

ภาษา SQL92 :

???

9. บอกช่วงเวลาทำงานของพนักงานที่ทำงานในแผนกของเล่นนั้นติดต่อกันอย่างน้อย 8 ปี

คำตอบ : {([01/01/1982 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT VALID (E)
FROM EMP (ID) AS E
WHERE DEPTNAME = 'ของเล่น'
AND CAST (VALID (EMP) AS INTERVAL YEAR) >= INTERVAL '8' YEAR
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM EMP_DEPT E1
WHERE E1.DEPTNAME = 'ของเล่น'
AND 8 <= (SELECT SUM (YEAR (NOW (VALIDTIMEEND)) – YEAR
(VALIDTIMESTART))
FROM EMP_DEPT E2
WHERE E1.ID = E2.ID AND E2.DEPTNAME = 'ของเล่น'
GROUP BY ID, DEPTNAME)
```

ภาษา SQL92 :

```
SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM EMP E1
WHERE E1.DEPTNAME = 'ของเล่น'
AND 8 <= (SELECT SUM (YEAR (NOW (VALIDTIMEEND)) – YEAR
(VALIDTIMESTART))
FROM EMP E2 WHERE E1.ID = E2.ID AND E2.DEPTNAME = 'ของเล่น'
GROUP BY ID, DEPTNAME)
```

10. เมื่อใดที่มีแผนกได้รับงบประมาณลดลง

คำตอบ : {(01/01/1987)}

ภาษา TSQL2 :

```
SELECT SNAPSHOT BEGIN (VALID (D2))
FROM DEPT (DEPTNAME) AS D, D (BUDGET) (PERIOD) AS D1 D2
WHERE VALID (D1) MEETS VALID (D2) AND D1.BUDGET > D2.BUDGET
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT D1.VALIDTIMESTART
FROM DEPT_BUDGET D1, DEPT_BUDGET D2
WHERE D1.DEPTNAME = D2.DEPTNAME
AND D1.VALIDTIMESTART-1 = NOW (D2.VALIDTIMEEND)
AND D1.BUDGET < D2.BUDGET
```

ภาษา SQL92 :

```
SELECT D1.VALIDTIMESTART
FROM DEPT D1, DEPT D2
WHERE D1.DEPTNAME = D2.DEPTNAME
AND D1.VALIDTIMESTART-1 = NOW (D2.VALIDTIMEEND)
AND D1.BUDGET < D2.BUDGET
```

11. ช่วงเวลาใดที่แผนกของเล่นมีงบประมาณมากกว่า 100000 บาท

คำตอบ : {([01/01/1982 – 12/31/1986])}

ภาษา TSQL2 :

```
SELECT VALID (D)
FROM DEPT (DEPTNAME, BUDGET) AS D
WHERE D.BUDGET > 100000 AND D.DEPTNAME = 'ของเล่น'
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM DEPT_BUDGET D
WHERE D.BUDGET > 100000 AND D.DEPTNAME = 'ของเล่น'
```

ภาษา SQL92 :

```
SELECT VALIDTIMESTART, NOW (VALIDTIMEEND)
FROM DEPT D
WHERE D.BUDGET > 100000 AND D.DEPTNAME = 'ของเล่น'
```

12. บอกชื่อแผนก ชื่อผู้จัดการ และช่วงเวลาทำงานของแผนกที่มีผู้จัดการที่ทำงานเป็นเวลาสั้นที่สุด

คำตอบ : {('หนังสือ', 'สมชาย', [04/01/1987 – 12/31/1987]) และ ('หนังสือ', 'ชาติชาย', [01/01/1988 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT D1.DEPTNAME, E1.NAME
FROM DEPT (DEPTNAME, ID) AS D1, EMP AS E1
WHERE E1.ID = D1.ID
AND CAST (VALID (D1) AS INTERVAL DAY) =
(SELECT MIN (CAST (VALID (D3) AS INTERVAL DAY)) FROM DEPT (ID) AS D3)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
CREATE VIEW V1 (DEPTNAME, ID, INTERVAL) AS
SELECT DEPTNAME, ID, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM DEPT_MGR
GROUP BY  DEPTNAME, ID;

SELECT D1.DEPTNAME, E1.NAME, E1.VALIDTIMESTART, NOW
(E1.VALIDTIMEEND)
FROM V1 D1, EMP E1
WHERE E1.ID = D1.ID
AND D1.INTERVAL = (SELECT MIN (D3.INTERVAL) FROM V1 D3)
AND E1.DEPTNAME = D1.DEPTNAME;

DROP VIEW V1;
```

ภาษา SQL92 :

```
CREATE VIEW V1 (DEPTNAME, ID, INTERVAL) AS
SELECT DEPTNAME, ID, (SUM (NOW (VALIDTIMEEND) - VALIDTIMESTART))
FROM DEPT
GROUP BY  DEPTNAME, ID;

SELECT D1.DEPTNAME, E1.NAME, E1.VALIDTIMESTART, NOW
(E1.VALIDTIMEEND)
FROM V1 D1, EMP E1
WHERE E1.ID = D1.ID
AND D1.INTERVAL = (SELECT MIN (D3.INTERVAL) FROM V1 D3)
AND E1.DEPTNAME = D1.DEPTNAME;

DROP VIEW V1;
```

13. บอกเงินเดือนที่มากที่สุดที่น้อยกว่า 50000 บาทที่สมชายได้และช่วงเวลาด้วย

คำตอบ : {(40000 | [02/01/1985 – 01/31/1987]) และ (40000 | [04/01/1987 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT MAX (E1.SALARY)
FROM EMP AS E1, EMP AS E2
WHERE E2.NAME = 'สมชาย' AND E1.ID = E2.ID
GROUP BY E1.ID
HAVING MAX (E1.SALARY) < 50000
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT DISTINCT E1.SALARY, E1.VALIDTIMESTART, NOW
                (E1.VALIDTIMEEND)
FROM EMP_SALARY E1, EMP_NAME E2
WHERE E2.NAME = 'สมชาย' AND E1.ID = E2.ID
AND E1.SALARY = (SELECT MAX (SALARY) FROM EMP_SALARY E3 WHERE
E3.ID = E2.ID AND E3.SALARY < 50000)
```

ภาษา SQL92 :

```
SELECT DISTINCT E1.SALARY, E1.VALIDTIMESTART, NOW
(E1.VALIDTIMEEND)
FROM EMP E1, EMP E2
WHERE E2.NAME = 'สมชาย' AND E1.ID = E2.ID
AND E1.SALARY = (SELECT MAX (SALARY) FROM EMP_SALARY E3 WHERE
E3.ID = E2.ID AND E3.SALARY < 50000)
```

14. บอกชื่อ เงินเดือน และช่วงเวลาที่มีพนักงานมีทักษะขับรถและมีเงินเดือนน้อยกว่า 40000 บาท

คำตอบ : {('สมชาย', 20000 | [02/01/1982 – 05/01/1982]) และ ('สมชาย', 30000 | [06/01/1984 – 01/31/1985])}

ภาษา TSQL2 :

```
SELECT EMP.NAME, EMP.SALARY
FROM EMP, SKILL
WHERE EMP.ID = SKILL.ID AND SKILL.SKILL = 'ขับรถ' AND EMP.SALARY <
40000
AND VALID (EMP) OVERLAPS VALID (SKILL)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT E1.NAME, E2.SALARY, MAXDATE (E2.VALIDTIMESTART,
S1.VALIDTIMESTART),
MINDATE (NOW (E2.VALIDTIMEEND), NOW (S1.VALIDTIMEEND))
FROM EMP_NAME E1, EMP_SALARY E2, SKILL_SKILL S1
WHERE E1.ID = S1.ID AND E2.ID = S1.ID
AND S1.SKILL = 'ขับรถ' AND E2.SALARY < 40000
AND E1.VALIDTIMESTART <= S1.VALIDTIMEEND
AND S1.VALIDTIMESTART <= E1.VALIDTIMEEND
AND E2.VALIDTIMESTART <= S1.VALIDTIMEEND
AND S1.VALIDTIMESTART <= E2.VALIDTIMEEND
AND E2.VALIDTIMESTART <= E1.VALIDTIMEEND
```

AND E1.VALIDTIMESTART <= E2.VALIDTIMEEND

ภาษา SQL92 :

SELECT E1.NAME, E1.SALARY, MAXDATE (E1.VALIDTIMESTART, S1.VALIDTIMESTART),
MINDATE (NOW (E1.VALIDTIMEEND), NOW (S1.VALIDTIMEEND))
FROM EMP E1, SKILL_SKILL S1
WHERE E1.ID = S1.ID
AND S1.SKILL = 'ขับรถ' AND E1.SALARY < 40000
AND E1.VALIDTIMESTART <= S1.VALIDTIMEEND
AND S1.VALIDTIMESTART <= E1.VALIDTIMEEND

15. บอกชื่อ เงินเดือน และช่วงเวลาที่มีพนักงานได้เงินเดือนมากกว่าผู้จัดการ

คำตอบ : {('สมชาย', 30000 | [01/06/1982 – 31/07/1984]) และ ('สมชาย', 40000 | 01/02/1985 – 31/08/1986])}

ภาษา TSQL2 :

SELECT E.NAME, E.SALARY
FROM EMP (ID, NAME, SALARY) AS E, EMP (ID, SALARY) AS Mgr, DEPT AS D1
WHERE E.DEPTNAME = D1.DEPTNAME AND VALID (E) OVERLAPS VALID (D1)
AND D1.ID = Mgr.ID AND VALID (D1) OVERLAPS VALID (Mgr)
AND E.SALARY = Mgr.SALARY AND E.ID <> Mgr.ID
AND VALID (E) OVERLAPS VALID (Mgr)

ภาษา SQL92 บนตารางที่เป็น TNF :

???

ภาษา SQL92 :

???

16. ช่วงเวลาใดที่แผนกของเล่นมีงบประมาณมากกว่า 175000 บาทติดต่อกันเป็นเวลายาวนานมากกว่า 1 ปี และงบประมาณนั้นเท่ากับเท่าไร

คำตอบ : {(200000 | [01/08/1984 – 31/12/1986])}

ภาษา TSQL2 :

```
SELECT D1.BUDGET
FROM DEPT (DEPTNAME, BUDGET) (PERIOD) AS D1
WHERE D1.DEPTNAME = 'ของเล่น' AND D1.BUDGET > 175000
AND CAST (VALID (D1) AS INTERVAL YEAR) > INTERVAL '1' YEAR
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT D1.BUDGET, D1.VALIDTIMESTART, D1.VALIDTIMEEND
FROM DEPT_BUDGET D1
WHERE D1.DEPTNAME = 'ของเล่น' AND D1.BUDGET > 175000
AND YEAR (NOW (D1.VALIDTIMEEND)) – YEAR (D1.VALIDTIMESTART) > 1
```

ภาษา SQL92 :

???

17. บอกชื่อและเงินเดือนทั้งหมดของพนักงานที่เกิดหลังวันที่ 1 เดือนมกราคม ปี ค.ศ. 1956

คำตอบ : {('สมหญิง', 30000 | [01/01/1982 – 31/07/1984]), ('สมหญิง', 40000 | [08/01/1984 – 31/08/1986]), ('สมหญิง', 50000 | [01/09/1986 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT E1.NAME, E1.SALARY
FROM EMP AS E1
WHERE E1.D-BIRTH > DATE '01/01/1956'
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT E1.NAME, E2.SALARY, E2.VALIDTIMESTART, NOW
(E2.VALIDTIMEEND)
FROM EMP_NAME E1, EMP_SALARY E2, EMP_NOTIME E3
WHERE E1.ID = E2.ID AND E1.ID = E3.ID AND E3.D-BIRTH > '01/01/1956'
```

ภาษา SQL92 :

```
SELECT E1.NAME, E1.SALARY, E1.VALIDTIMESTART, NOW
```

(E1.VALIDTIMEEND)

FROM EMP E1

WHERE E1.D-BIRTH > '01/01/1956'

18. บอกเงินเดือนทั้งหมดของสมชาย

คำตอบ : {(20000 | [01/02/1982 – 31/05/1982]), (30000 | 01/06/01982 – 31/01/1985], (40000 | [01/02/1985 – 31/01/1987] ∪ [01/04/1987 – 01/01/1990])}

ภาษา TSQL2 :

SELECT E1.SALARY

FROM EMP AS E1, EMP AS E2

WHERE E2.NAME = 'สมชาย' AND E1.ID = E2.ID AND VALID (E1) OVERLAPS VALID (E2)

ภาษา SQL92 บนตารางที่เป็น TNF :

SELECT E1.SALARY, E1.VALIDTIMESTART, NOW (E1.VALIDTIMEEND)

FROM EMP_SALARY E1, EMP_NAME E2

WHERE E2.NAME = 'สมชาย' AND E1.ID = E2.ID

AND E1.VALIDTIMESTART <= E2.VALIDTIMEEND

AND E2.VALIDTIMESTART <= E1.VALIDTIMEEND

ภาษา SQL92 :

SELECT E1.SALARY, E1.VALIDTIMESTART, NOW (E1.VALIDTIMEEND)

FROM EMP E1

WHERE E1.ID IN (SELECT ID FROM EMP WHERE NAME = 'สมชาย')

19. บอกชื่อ และเงินเดือนของพนักงานที่เคยเป็นผู้จัดการและได้เงินเดือนอย่างน้อย 36000 บาท

คำตอบ : {('สมหญิง', 40000 | [01/08/1984 – 31/08/1986]), ('สมหญิง', 50000 | [01/09/1986 – 01/01/1990]), ('สมชาย', 40000 | [01/04/1987 – 31/12/1987]), ('ชาติชาย', 40000 | [01/01/1988 – 01/01/1990])}

ภาษา TSQL2 :

```
SELECT EMP.NAME, EMP.SALARY
FROM EMP, DEPT
WHERE EMP.SALARY > 36000 AND EMP.ID = DEPT.ID
AND VALID (EMP) OVERLAPS VALID (DEPT)
```

ภาษา SQL92 บนตารางที่เป็น TNF :

```
SELECT E1.NAME, E2.SALARY, MAXDATE (E1.VALIDTIMESTART,
E2.VALIDTIMESTART),
MINDATE (E1.VALIDTIMEEND, E2.VALIDTIMEEND)
FROM EMP_NAME E1, EMP_SALARY E2
WHERE E2.SALARY > 36000
AND E1.ID = E2.ID
AND E1.ID IN (SELECT ID FROM DEPT_MGR)
AND E1.VALIDTIMESTART <= E2.VALIDTIMEEND
AND E2.VALIDTIMESTART <= E1.VALIDTIMEEND
```

ภาษา SQL92 :

```
SELECT E1.NAME, E1.SALARY, VALIDTIMESTART, VALIDTIMEEND
FROM EMP E1
WHERE SALARY > 36000
AND E1.ID IN (SELECT ID FROM DEPT_MGR)
```

ตารางที่ 3.24 ข้อดีและข้อเสียของภาษาที่ใช้ในการตอบคำถามเชิงเวลาแบบต่างๆ

	ข้อดี	ข้อเสีย
ภาษา TSQL2	1. สามารถจัดโครงสร้างตารางใหม่ (Restructuring) ได้ ทำให้สามารถตอบคำถามเชิงเวลาได้สะดวก 2. สนับสนุน Temporal Comparison Operator ทำให้สามารถเปรียบเทียบข้อมูลเชิงเวลาได้ง่าย 3. มี Temporal Built-in Function ทำให้สามารถตอบคำถามเชิงเวลาได้สะดวก	1. มีความซับซ้อนในการใช้คำสั่งมากกว่าภาษา SQL92
ภาษา SQL92 บนตารางที่เป็น TNF	1. เนื่องจากในแต่ละตารางจะมีแอททริบิวต์ที่เปลี่ยนแปลงตามเวลาเพียงแอททริบิวต์เดียว ทำให้เสมือนว่าเป็นการจัดโครงสร้างตารางใหม่บนคอลัมน์ Primary Key กับแอททริบิวต์ที่เปลี่ยนแปลงตามเวลา 2. ใช้งานง่ายเนื่องจากใช้ภาษา SQL92	1. มีความยุ่งยากในการตอบคำถามเนื่องจากต้องมีการ Join ตารางที่ทำ Time Normalized เข้าด้วยกัน 2. ต้องทราบว่าคอลัมน์ที่เปลี่ยนแปลงตามเวลาของตารางเดิม เมื่อทำ Time Normalized แล้วไปอยู่บนตารางใด

ตารางที่ 3.24 ข้อดีและข้อเสียของภาษาที่ใช้ในการตอบคำถามเชิงเวลาแบบต่างๆ (ต่อ)

	ข้อดี	ข้อเสีย
ภาษา SQL92	1. ใช้งานง่ายเนื่องจากใช้ภาษา SQL92 2. ไม่ต้อง Join ตาราง เนื่องจากไม่ได้ทำการแยกตาราง	1. คอลัมน์ที่เปลี่ยนแปลงตามเวลาอยู่รวมกันบนตารางเดียวกันทำให้เกิดความซ้ำซ้อน และไม่ทราบว่าคอลัมน์ใดที่มีการเปลี่ยนแปลงจริง

ในภาษา SQL92 ในการตอบคำถามเชิงเวลาบางครั้งจะได้ช่วงเวลาที่ไม่รวมกัน เช่น คำตอบควรจะเป็น (20000 | [01/01/1982 – 31/12/1986]) แต่จะได้เป็น (20000 | [01/01/1982 – 31/12/1984]), (20000 | [01/01/1985 – 31/12/1986]) ซึ่งอาจทำให้การสื่อความหมายผิดไปได้ และ

ภาษา SQL92 ไม่สามารถหาช่วงเวลาที่ต่อเนื่องกันได้ ทำให้ตอบคำถามไม่ครอบคลุมทั้งหมด เช่น ตอบคำถามที่ว่า "ใครมีเงินเดือนเท่าเดิมนานที่สุด" ได้ แต่ไม่สามารถตอบคำถามที่ว่า "ใครมีเงินเดือนเท่าเดิมติดต่อกันนานที่สุด" ได้ แต่ถ้าเป็นภาษา SQL92 บนตารางที่เป็น TNF จะสามารถตอบได้ครอบคลุมมากกว่า

อนึ่งภาษา SQL/Temporal นี้เป็นภาษาที่ใช้งานบนฐานข้อมูลเชิงเวลา ซึ่งได้ประสิทธิภาพที่ดี แต่ก็ไม่สามารถนำมาใช้งานกับฐานข้อมูลที่มีอยู่ได้ เนื่องจากไม่มีระบบฐานข้อมูลใดที่ใช้ภาษานี้ได้ ที่สำคัญข้อมูลต่าง ๆ ที่เรามีอยู่นั้นจัดเก็บอยู่บนฐานข้อมูลเชิงวัตถุสัมพันธ์ ซึ่งไม่สามารถใช้ภาษา SQL/Temporal ได้

ปัญหาของการใช้ SQL ธรรมดาในการจัดการกับข้อมูลเชิงเวลา

1. ถ้าต้องการหลีกเลี่ยงความซ้ำซ้อนในการจัดเก็บลงตารางที่เก็บข้อมูลที่มีการเปลี่ยนแปลงตามเวลา จำเป็นต้องการใช้ aggregate และ trigger ที่ซับซ้อน
2. การจอยตารางปกติ ซึ่งจะใช้แค่ 3 บรรทัด คือ

```
SELECT X1.Id, X1.Name, X2.City
FROM  X1 AS TABLE1,  X2 AS TABLE1
WHERE X1.Id = X2.Id
```

แต่ถ้านำมาประยุกต์ใช้ในตารางที่เก็บข้อมูลที่มีการเปลี่ยนแปลงตามเวลา จะต้องใช้ถึง 37 บรรทัด ซึ่งประกอบด้วย 4 SELECT ดังนี้

```
SELECT X1.Id, X1.Name, X2.City
FROM  X1 AS TABLE1,  X2 AS TABLE1
WHERE X1.Id < X2.Id
    AND X2.FROM_DATE <= X1.FROM_DATE
    AND X1.TO_DATE <= X2.TO_DATE
UNION
SELECT X1.Id, X1.Name, X2.City
FROM  X1 AS TABLE1,  X2 AS TABLE1
WHERE X1.Id < X2.Id
    AND X1.FROM_DATE > X2.FROM_DATE
    AND X2.TO_DATE < X1.TO_DATE
    AND X1.FROM_DATE < X2.TO_DATE
UNION
SELECT X1.Id, X1.Name, X2.City
```

```
FROM  X1 AS TABLE1,  X2 AS TABLE1
WHERE X1.Id < X2.Id
    AND X2.FROM_DATE > X1.FROM_DATE
    AND X1.TO_DATE < X2.TO_DATE
    AND X2.FROM_DATE < X1.TO_DATE
UNION
SELECT X1.Id, X1.Name, X2.City
FROM  X1 AS TABLE1,  X2 AS TABLE1
WHERE X1.Id < X2.Id
    AND X2.FROM_DATE > = X1.FROM_DATE
    AND X2.TO_DATE <= X1.TO_DATE
```

3. การอับเดทตารางปกติ ซึ่งจะใช้แค่ 3 บรรทัด คือ

```
UPDATE INTO TABLE1
SET NAME = 'SOMCHAI'
WHERE ID = 'SOMC'
```

แต่ถ้านำมาประยุกต์ใช้ในตารางที่เก็บข้อมูลที่มีการเปลี่ยนแปลงตามเวลา จะต้องใช้โค้ดมากขึ้น

4. การปรับปรุงเปลี่ยนแปลง ต้องการการทริกเกอร์หลายครั้ง ซึ่งจะทำให้ต้องเขียนโค้ดหลายบรรทัดมากกว่าปกติ

SQL-92 รองรับการจัดการข้อมูลที่เปลี่ยนแปลงตามเวลาผ่านชนิดข้อมูล DATE, TIME และ TIMESTAMP แต่ภาษาจะไม่เข้าใจข้อมูลเชิงเวลา ไม่สามารถเข้าถึงการค้นหา และการเปลี่ยนแปลงข้อมูล แบบปัจจุบัน และแบบซีเควนต์ ทั้งตารางแบบวาลิดไทม์ ทรานแซคชั่นไทม์ และไบเท็มโพรอล แต่จะรองรับกระบวนการแบบนอนซีเควนต์ทั้งหมด

3.10 ปัญหาของฐานข้อมูลเชิงสัมพันธ์กับฐานข้อมูลเชิงเวลา

ฐานข้อมูลเชิงสัมพันธ์ โดยปกติจะเก็บเฉพาะข้อมูลที่เป็นจริงในปัจจุบันเท่านั้น และจะมีคีย์หลัก (Primary Key) ของตาราง ดังเช่นตารางที่ 3.25

ตารางที่ 3.25 การจัดเก็บฐานข้อมูลเชิงสัมพันธ์ทั่วไป

ID	NAME	SURNAME	SALARY
1	JAME	SMITH	20000
2	ANNA	CUNICOVA	25000

หากมีการแก้ไขค่าของแอตทริบิวต์ใดแอตทริบิวต์หนึ่ง จำเป็นต้องมีการอับเดท (Insert) ทับข้อมูลเดิม ถ้ายังคงต้องการใช้ ID ตัวเดิมในการอ้างถึงอยู่

ตารางที่ 3.26 การอับเดทข้อมูลฐานข้อมูลเชิงสัมพันธ์ทั่วไป

ID	NAME	SURNAME	SALARY
~~1~~	~~JAME~~	~~SMITH~~	~~20000~~
2	ANNA	CUNICOVA	25000
1	JOHN	SMITH	20000

จากตารางที่ 3.26 จะเป็นการอับเดทแอตทริบิวต์ Name ซึ่ง JAME เปลี่ยนชื่อเป็น JOHN ซึ่งจะทำให้ต้องทำการอับเดททับโรว์เดิม ทำให้ไม่สามารถทราบได้ว่า JOHN เคยมีชื่อเป็น JAME มาก่อน และไม่สามารถทราบได้ว่าแอตทริบิวต์ใดเปลี่ยน

หากต้องการให้ตารางฐานข้อมูลเชิงสัมพันธ์สามารถจัดเก็บข้อมูลเชิงเวลาได้ จำเป็นต้องเพิ่ม 2 คอลัมน์คือ FROM_DATE และ TO_DATE ซึ่งจะเป็นตัวบอกว่าข้อมูลมีการเปลี่ยนแปลงเมื่อใด และต้องคอมไบน์คีย์หลัก ให้เป็น ID, FROM_DATE และ TO_DATE ดังตารางที่ 3.27

ตารางที่ 3.27 การจัดเก็บฐานข้อมูลเชิงเวลาโดยใช้ฐานข้อมูลเชิงสัมพันธ์

ID	NAME	SURNAME	SALARY	FROM_DATE	TO_DATE
1	JAME	SMITH	20000	1/1/1980	1/1/1990
2	ANNA	CUNICOVA	25000	1/1/1980	31/12/9999
1	JOHN	SMITH	20000	1/1/1990	31/12/1999

การจัดเก็บข้อมูลดังตารางที่ 3.27 จะทำให้ฐานข้อมูลเชิงสัมพันธ์สามารถจัดเก็บข้อมูลเชิงเวลาได้ ซึ่งจากตาราง JAME จะมีการเปลี่ยนชื่อ JOHN เมื่อวันที่ 1/1/1990 ซึ่งจะทำให้ทราบว่าชื่อ JAME และ JOHN มีค่าช่วงใดโดยดูจากคอลัมน์ FROM_DATE และ TO_DATE และสามารถที่จะอ้างได้ว่า JAME และ JOHN เป็นคนคนเดียวกัน โดยดูจากหมายเลข ID

แต่การจัดเก็บข้อมูลเชิงเวลาโดยใช้ฐานข้อมูลเชิงสัมพันธ์ก็มีปัญหา ดังนี้

1. การเอาช่วงเวลาเกาะกับทั้งทัพเปิล ทำให้ไม่ทราบว่าแอตทริบิวต์ใดที่มีการเปลี่ยนค่าตามเวลา
2. การอ้างอิงคีย์หลัก (Primary Key) คีย์อ้างอิง (Foreign Key) ทำได้ยุ่งยาก จากตารางจะเห็นได้ว่าคีย์หลักจะเพิ่มเป็น 3 ตัว ซึ่งถ้านำไปเป็นคีย์อ้างอิงกับตารางอื่น ก็จะเกิดความยุ่งยากขึ้น

บทที่ 4
แนวคิดเชิงวัตถุและฐานข้อมูลเชิงวัตถุ

4.1 แนวความคิดเชิงวัตถุ

รายละเอียดหลักการต่าง ๆ ของแนวคิดเชิงวัตถุ[6]มีดังนี้

4.1.1 ออบเจ็กต์ (Object)

ออบเจ็กต์คือการนำเสนอ Abstract ของ Real World Entity ที่มี

- Unique Identity : ออบเจ็กต์ต้องมีสิ่งที่ระบุถึงความแตกต่างจากออบเจ็กต์อื่น
- Embedded Properties
- ความสามารถในการติดต่อสื่อสารกับออบเจ็กต์อื่นและตัวเอง

4.1.2 รหัสเฉพาะของออบเจ็กต์ (Object Identity : OID)

Object Identity (OID) เป็นสิ่งที่ใช้ระบุหรืออ้างอิงออบเจ็กต์ ซึ่งแต่ละออบเจ็กต์จะมี OID ไม่ซ้ำกัน OID จะถูกกำหนดให้โดยระบบตั้งแต่เมื่อออบเจ็กต์ถูกสร้างขึ้นและจะไม่สามารถเปลี่ยนได้

OID จะแตกต่างจากคีย์หลักของระบบฐานข้อมูลเชิงสัมพันธ์ เพราะ คีย์หลักจะขึ้นอยู่กับค่าของ Attribute ที่ผู้ใช้กำหนด ซึ่งสามารถเปลี่ยนแปลงในภายหลังได้ แต่ OID ถูกกำหนดโดยระบบซึ่งจะไม่ขึ้นกับค่า Attribute ของออบเจ็กต์และเปลี่ยนแปลงไม่ได้ เมื่อออบเจ็กต์ถูกลบออกไปจากระบบ OID ของออบเจ็กต์นั้นก็จะถูกลบตามไปด้วยและจะไม่มีการนำ OID ที่ถูกลบไปแล้วกลับมาใช้ใหม่ ค่าของ OID จะไม่ผูกติดกับ Physical Address ของหน่วยความจำถาวร (Permanent Memory) ทำให้ระบบของ Object-Oriented เป็น Physical Data Independence

4.1.3 แอตทริบิวต์ (Attributes, Instance Variable)

แอตทริบิวต์ (Attribute) หรือ Instance Variable ก็คือ data ของออบเจ็กต์ อาจเป็นได้ทั้งข้อมูลชนิดพื้นฐานหรือเป็นออบเจ็กต์ก็ได้

4.1.4 สถานะของออบเจ็กต์ (Object State)

สถานะของออบเจ็กต์ (Object State) ขึ้นกับค่าของแอตทริบิวต์ของออบเจ็กต์ ณ เวลาที่กำหนด ถึงแม้ว่าสถานะของออบเจ็กต์จะเปลี่ยนแปลงไปแต่ OID ยังคงเดิม ถ้าต้องการเปลี่ยนสถานะของออบเจ็กต์ต้องเปลี่ยนค่าของแอตทริบิวต์โดยจะต้องส่งเมสเซจไปยังออบเจ็กต์ แล้วเมสเซจที่ส่งไปนี้จะเรียกเมธอดที่เกี่ยวข้องให้ทำงาน

4.1.5 เมสเซจและเมธอด(Message and Method)

ตามคุณสมบัติของเอนแคปซูเลชั่นการที่จะกระทำการใดๆ กับออบเจ็กต์ต้องกระทำผ่านเมธอดเท่านั้นเมธอดจะถูกใช้หรือเรียกดูหรือเปลี่ยนค่า Attribute ของออบเจ็กต์ ในการเรียกใช้เมธอดนั้นจะต้องส่งเมสเซจไปยังออบเจ็กต์ เมสเซจที่ส่งจะต้องระบุออบเจ็กต์ที่จะรับเมสเซจ ชื่อเมธอด และพารามิเตอร์ที่เกี่ยวข้อง

4.1.6 การรวมกันของข้อมูลและการจัดการข้อมูล (Encapsulation)

ออบเจ็กต์จะเป็นการรวมเอาแอตทริบิวต์และเมธอดไว้ด้วยกัน การจะเข้าถึงแอตทริบิวต์ต้องทำผ่านเมธอดเท่านั้น ซึ่งจะช่วยปกป้องข้อมูลของออบเจ็กต์

4.1.7 คลาส (Class)

ในระบบเชิงวัตถุจะนำออบเจ็กต์ที่มีคุณสมบัติเหมือนกันเข้าไว้ด้วยกันเป็นคลาส หรืออาจพูดอีกอย่างได้ว่า คลาสก็คือ Collection ของออบเจ็กต์ที่มีคุณสมบัติเหมือนกันและใช้แอตทริบิวต์และเมธอดร่วมกัน

4.1.8 การสืบทอดคุณสมบัติ (Inheritance)

Inheritance เป็นการสร้างคลาสใหม่ (Subclass) โดยสืบทอดคุณสมบัติ (ทั้งแอตทริบิวต์และเมธอด) จากคลาสที่มีอยู่แล้ว (Superclass) นอกจากนี้ยังสามารถเพิ่มรายละเอียดให้ Subclass ได้อีกด้วย Inheritance นี้มีประโยชน์ในเรื่องการนำกลับมาใช้ใหม่

Single Inheritance เป็นการสืบทอดคุณสมบัติมาจากคลาสเดียว

Multiple Inheritance เป็นการสืบทอดคุณสมบัติจากหลายๆคลาส

4.1.9 Method Overriding and Polymorphism

Subclass สามารถสร้างเมธอดที่ชื่อซ้ำกับเมธอดที่สืบทอดมาจาก Superclass ได้ เพื่อทำให้ Subclass นั้นมีความเฉพาะเจาะจงมากขึ้น เมธอดที่ Subclass สร้างขึ้นนี้จะ Override เมธอดของ Superclass

โพลิมอร์ฟีซึม (Polymorphism) เป็นการทำให้ออบเจ็กต์ที่ต่างกันสามารถตอบสนองต่อเมสเซจเดียวกันได้ในหลายๆวิธีการ โพลิมอร์ฟีซึมเป็นคุณสมบัติที่สำคัญที่สำคัญของระบบเชิงวัตถุ เพราะช่วยให้แต่ละออบเจ็กต์มีความเฉพาะเจาะจงมากขึ้น ในระบบเชิงวัตถุคำว่าโพลิมอร์ฟีซึมจะหมายถึง

1. เมธอดสามารถใช้ชื่อเดียวกันได้ในหลายๆคลาส
2. ผู้ใช้ส่งเมสเซจเดียวกันไปยังออบเจ็กต์จากคลาสต่างๆกัน ก็ยังคงได้ผลลัพธ์ที่ถูกต้อง

4.1.10 Abstract Data Type

Abstract Data Type (ADT) เป็นคุณสมบัติที่ใช้สร้างชนิดของข้อมูลใหม่ขึ้นมา โดยกำหนดโครงสร้างข้อมูลและโอเปอร์เรชันที่ใช้จัดการข้อมูลขึ้นมาจากข้อมูลพื้นฐาน

จะสังเกตได้ว่า Abstract Data Type และคลาสมีความหมายใกล้เคียงกัน แต่ในระบบเชิงวัตถุสองคำนี้มีความหมายต่างกัน โดย Type จะหมายถึง โครงสร้างข้อมูลและเมธอดของคลาส และคลาสหมายถึง Collection ของ Object Instance เมื่อกำหนดคลาสใหม่นี้ขึ้นมาก็เป็นการกำหนด Type ใหม่ด้วย Type ที่กำหนดขึ้นจะถูกใช้เป็นต้นแบบในการสร้างออบเจ็กต์ใหม่ ซึ่งจะถูกจัดการโดยคลาสขณะ run-time

ด้วยคุณสมบัติของ ADT และ Inheritance จะสนับสนุนให้มี Complex Object โดย Complex Object ถูกสร้างขึ้นโดยนำออบเจ็กต์อื่นเข้ามารวมไว้ด้วยในรูปของ Set ของ Complex Relation

4.2 คุณลักษณะของ Object-Oriented Data Model (OODM)

แบบจำลองใด ๆ ที่จะเป็น OODM อย่างน้อยที่สุดควรมีลักษณะดังนี้

4.2.1 ต้องสนับสนุนการทำ Complex Object

4.2.2 Extensible : ต้องมีความสามารถในการกำหนด Data Type และ Operation ที่เกี่ยวข้องขึ้นมาใหม่ได้

4.2.3 ต้องสนับสนุน Encapsulation : รูปแบบของข้อมูลและการจัดการของเมธอดต้องถูกซ่อนจากภายนอก

4.2.4 ต้องมีคุณสมบัติ Inheritance : ออบเจ็กต์ต้องสามารถสืบทอดคุณสมบัติ (ข้อมูลและเมธอด) จากออบเจ็กต์อื่นได้

4.2.5 ต้องสนับสนุน Object Identity (OID)

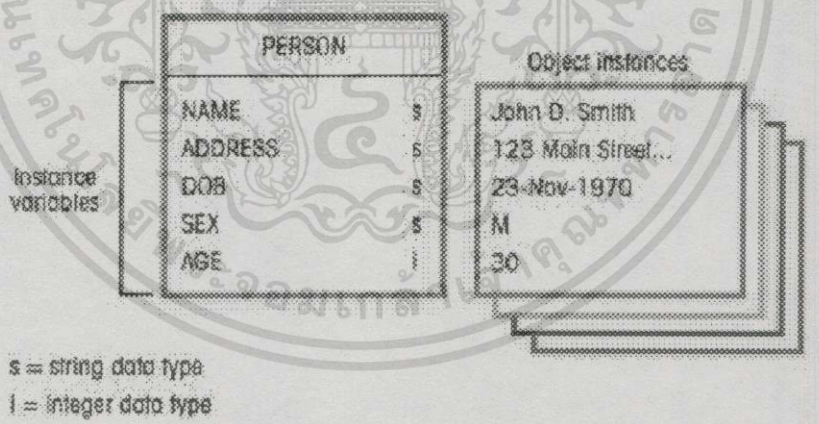
เปรียบเทียบ OO Data Model และ ER Model ดังตารางที่ 4.1

ตารางที่ 4.1 เปรียบเทียบ OO Data Model และ ER Model [17]

OODM	ER Model
Type	Entity Definition
Object	Entity
Class	Entity Set
Instance Variable	Attribute
N/A	Primary Key
OID	N/A
Method	N/A
Class Hierarchy	ER Diagram (Database Schema)

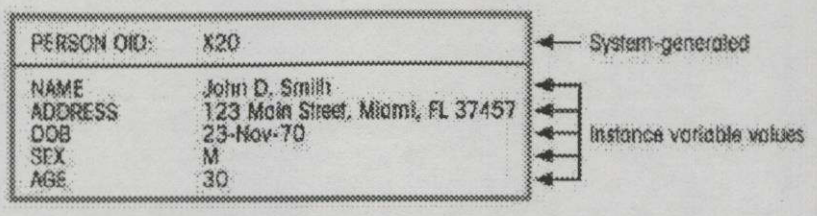
4.3 การนำเสนอออบเจ็กต์ในรูปของ Object Diagram

ใน Object Diagram จะแทนออบเจ็กต์ด้วยรูปสี่เหลี่ยมมุมฉาก มี Instance Variable อยู่ภายใน รูปที่ 4.1 แสดงคลาส Person มี Instance Variable คือ NAME, ADDRESS, DOB, (Date Of Birth) และ SEX ซึ่งมี Data Type เป็นสตริง และ AGE มี Data Type เป็น Integer

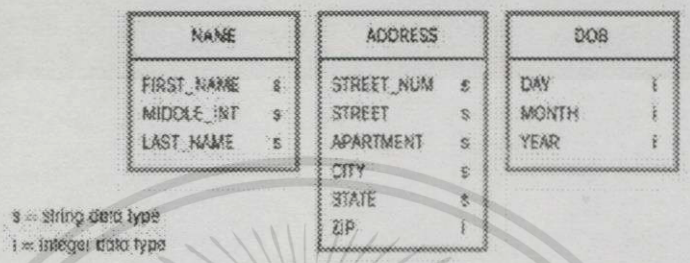


รูปที่ 4.1 คลาส Person

พิจารณา State ของออบเจ็กต์ Person ได้ในรูปที่ 4.2 ในระบบเชิงวัตถุสามารถที่จะสร้าง ADT ใหม่ได้จาก Data Type พื้นฐานที่มีมาให้ เช่น ถ้าให้ NAME, ADDRESS และ DOB เปลี่ยนเป็น Composite Attribute ซึ่งสามารถทำได้โดยจัดการผ่านคลาสหรือ ADT ดังรูปที่ 4.3 เป็นผลให้คลาส Person มี Attribute ที่ชี้ไปยังออบเจ็กต์ของคลาสหรือ ADT อื่น ดังรูปที่ 4.4 แทนที่จะเป็นข้อมูลชนิดพื้นฐาน



รูปที่ 4.2 State ของออบเจ็กต์จากคลาส Person

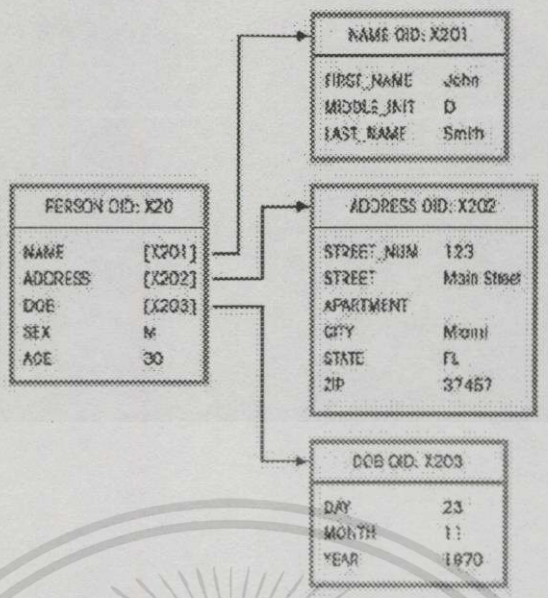


รูปที่ 4.3 การกำหนด Abstract Data Type 3 ชนิด

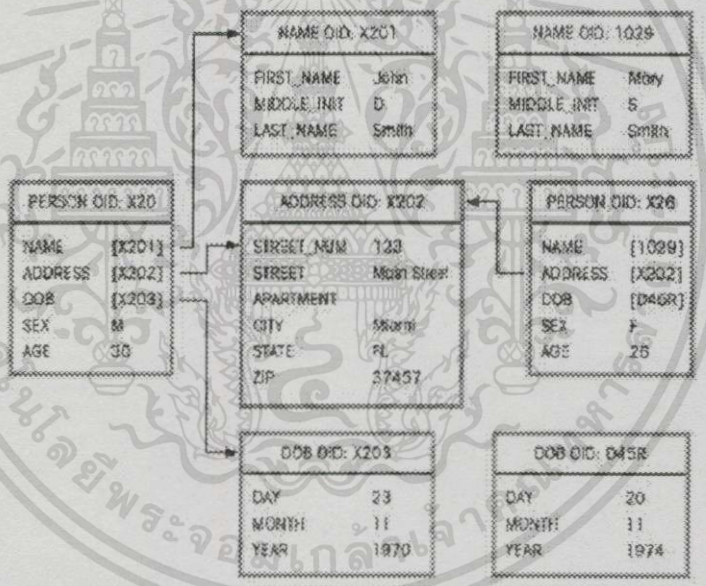
รูปที่ 4.4 ออบเจ็กต์ของคลาส Person กับ ADT

คำว่า Object Space หรือ Object Schema ก็คือ Database Schema ณ เวลาที่กำหนด ใช้ Object Space เพื่อแสดงให้เห็นสถานะต่างๆของออบเจ็กต์ ณ เวลาหนึ่ง Object Space ของคลาส Person เป็นดังรูปที่ 4.5 สังเกตได้ว่าใช้ OID ในการอ้างอิงไปยังออบเจ็กต์อื่น โดย Attribute NAME, ADDRESS และ DOB ขณะนี้มีค่าเป็น OID ของออบเจ็กต์จากคลาสหรือ ADT ที่เกี่ยวข้องแทนที่จะเป็นค่าของข้อมูลชนิดพื้นฐาน การใช้ OID ในการอ้างอิงถึงออบเจ็กต์อื่นทำให้สามารถหลีกลี่ยงปัญหา Data Inconsistency ได้ เพราะ OID ไม่ได้ขึ้นอยู่กับ State ของออบเจ็กต์ ซึ่งปัญหานี้จะปรากฏอยู่ในระบบฐานข้อมูลเชิงสัมพันธ์ เมื่อ Primary Key ถูกเปลี่ยนค่าโดนผู้ใช้

ตัวอย่างเช่น ถ้ามีคนสองคน อาศัยอยู่ในบ้านเดียวกัน ควรอ้างอิงไปยัง Object Instance เดียวกันของคลาส Address แทนที่จะเป็น 2 Object Instance ที่มี state เดียวกัน สภาวะเช่นนี้ถูกเรียกว่า referential object sharing ซึ่งการเปลี่ยน share ของออบเจ็กต์จากคลาส Address จะส่งผลกระทบต่อออบเจ็กต์จากคลาส Person ทั้งสองออบเจ็กต์ ดังรูปที่ 4.6 ซึ่งมี 4 คลาสประกอบด้วย คลาส Person (2 Instance) คลาส Name (2 Instance) คลาส Address และคลาส DOB (2 Instance)



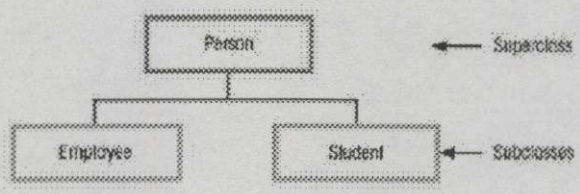
รูปที่ 4.5 สถานะของออบเจ็กต์ที่เป็น Instance ของคลาส Person ที่เป็น ADT



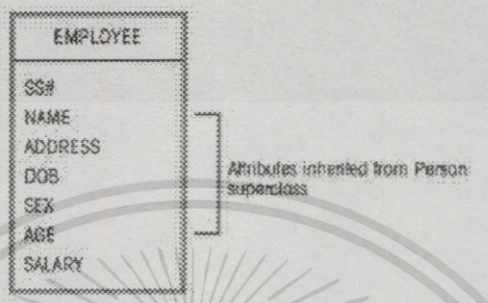
รูปที่ 4.6 Referential Object Sharing

4.3.1 ความสัมพันธ์ระหว่างคลาสและ subclass (Class-Subclass Relationship)

ในการสืบทอดคุณสมบัติของคลาสมาจาก Superclass จะเรียกความสัมพันธ์แบบนี้ว่า 'is-a relationship' เช่น a employee is a person, a student is a person ตามรูปที่ 4.7 ซึ่งออบเจ็กต์ Employee ในรูปที่ 4.8 ประกอบด้วย Attribute ต่างๆดังนี้ หมายเลขประกันสังคม (SS#) เก็บเป็นสตริง เงินเดือน (SALARY) เก็บเป็นตัวเลข และมี NAME, ADDRESS, DOB และ AGE ที่สืบทอดมาจาก Supercalss ความสัมพันธ์ระหว่างคลาสและ Subclass นี้จะเป็นแบบ 1:1 ถ้าเป็น Single Inheritance



รูปที่ 4.7 Class Hierarchy

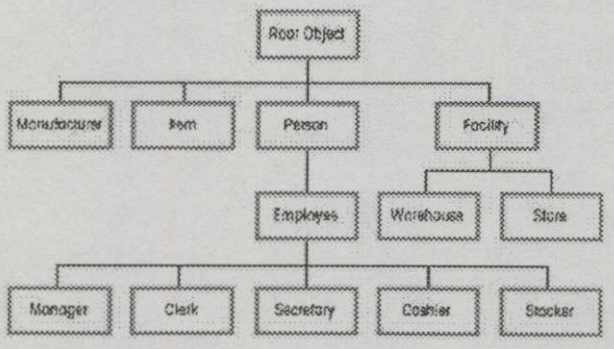


รูปที่ 4.8 ออบเจ็กต์ของคลาส Employee

4.3.2 ความสัมพันธ์ระหว่างคลาส แบบ Interclass Relationships : Attribute-Class Links

นอกจากความสัมพันธ์ระหว่างคลาสและ Subclass แล้ว OODM ยังสนับสนุนความสัมพันธ์ระหว่างคลาส (Interclass Relationships) ด้วย ความสัมพันธ์ระหว่างคลาสจะเกิดขึ้นเมื่อ Data Type ของ Attribute มีการอ้างอิงไปยัง Attribute Data Type อื่น ความสัมพันธ์ระหว่างคลาสมีความแตกต่างจากความสัมพันธ์ระหว่างคลาสและ Subclass ในหัวข้อก่อนหน้านี้ ลองพิจารณา Class Hierarchy ของ EDLP (Every Day Low Price) Retail Corporation ดังรูปที่ 4.9

ตามรูปที่ 4.9 ทุกๆคลาสสืบทอดคุณสมบัติมาจากคลาส Root Object Class Hierarchy ประกอบไปด้วยคลาส Manufacturer, Item, Person และ Facility คลาส Facility ประกอบไปด้วย Subclass Warehouse และ Store คลาส Person ประกอบไปด้วย Subclass Employee ที่มี Subclass Manager, Clerk, Secretary, Cashier และ Stoker จากตัวอย่างนี้จะแสดงให้เห็นถึงความสัมพันธ์แบบ 1:M และM:N และจะ Implement ความสัมพันธ์แบบ M:N

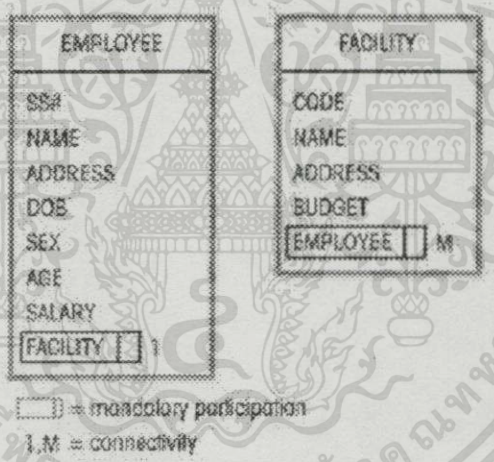


รูปที่ 4.9 Class Hierarchy ของ EDLP Retail Corp.

จากตัวอย่างในรูปที่ 4.9 ระหว่างคลาส Employee และ Facility มีความสัมพันธ์แบบ one to many หรือ 1:M เกิดขึ้น โดย Employee ทุกคนจะทำงานเพียงหนึ่ง Facility ในขณะเดียวกันหนึ่ง Facility จะมีหลายๆ Employee ทำงานอยู่

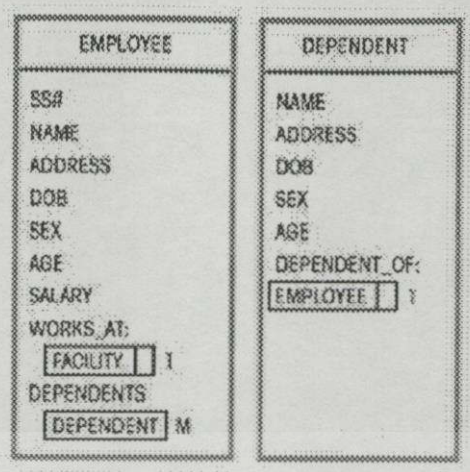
ตามรูปที่ 4.10 แสดงความสันพันธ์ระหว่าง Employee และ Facility โดยออบเจ็กต์ Facility จะถูกรวมไว้ในออบเจ็กต์ Employee และออบเจ็กต์ Employee ก็ถูกรวมไว้ในออบเจ็กต์ Facility โดยมีรายละเอียดดังนี้

1. คลาสที่เกี่ยวข้องต้องถูกล้อมรอบโดยสี่เหลี่ยมเพื่อที่จะได้เห็นเด่นชัด
2. เส้นคู่ทางด้านขวาของรูปสี่เหลี่ยมแสดงว่าความสัมพันธ์นั้นเป็นแบบ Mandatory
3. Connectivity จะระบุโดยเขียนกำกับไว้ที่ข้างรูปสี่เหลี่ยมนั้น ในกรณีที่เขียน 1 ถัดจาก Facility ในออบเจ็กต์ Employee หมายถึง Employee ทุกคนจะทำงานเพียงหนึ่ง Facility และ M ที่อยู่ข้าง Employee ในออบเจ็กต์ Facility หมายถึง หนึ่ง Facility จะประกอบไปด้วยหลายๆ Employee



รูปที่ 4.10 ความสัมพันธ์แบบ 1:M

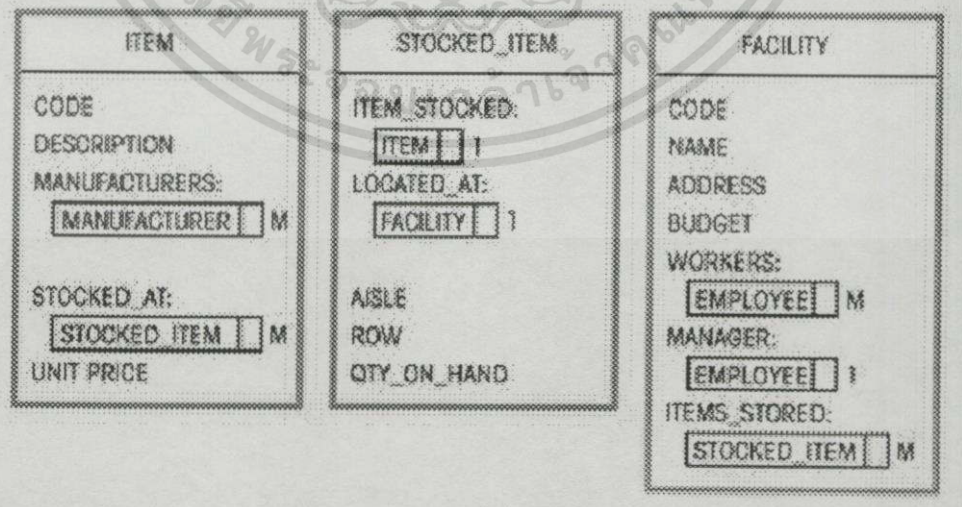
ตามตัวอย่าง EDLP Retail Corp. ความสัมพันธ์แบบ many to many (M:N) เกิดขึ้นระหว่างคลาส Manufacturer และ Item ตามรูปที่ 4.11 ซึ่งแสดง Conceptual view โดยที่แต่ละ Item สามารถผลิตได้จากหลายๆ Manufacturer และแต่ละ Manufacturer ก็สามารถผลิตได้หลายๆ Item



รูปที่ 4.11 ความสัมพันธ์แบบ M:N

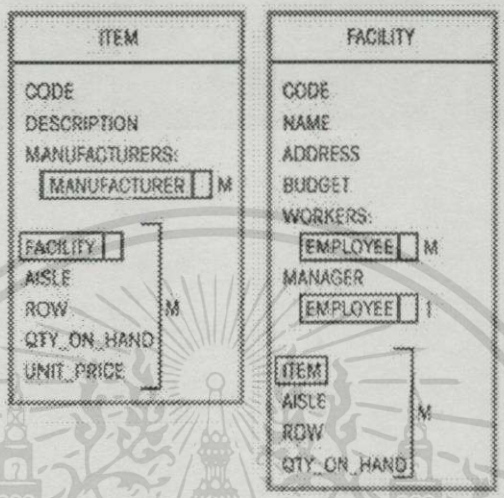
4.3.3 ความสัมพันธ์ many to many แบบ Intersection Class

ถ้าเพิ่มเงื่อนไขให้กับความสัมพันธ์แบบ M:N เพื่อให้สามารถติดตาม (Keep Track) ข้อมูลที่จะเพิ่มเข้าไปได้ เช่น ความสัมพันธ์ระหว่างคลาส Item และ Facility โดยที่แต่ละ Facility จะมีหลาย Item เก็บอยู่ และแต่ละ Item ก็จะเก็บได้หลายๆ Facility ถ้าต้องการรู้เพิ่มว่าแต่ละ Item ในแต่ละ Facility มีปริมาณเท่าไรและเก็บที่ตำแหน่ง (aisle และ row) ไหน โดยแสดงตัวอย่างที่รูป 4.12 วงเล็บใหญ่ ( ]) ที่ใช้ในรูปหมายถึงรวม Attribute ทั้งหมดในวงเล็บถือเป็นหนึ่ง Logical Unit เพราะฉะนั้น แต่ละ Item Instance จะประกอบไปด้วยหลายๆ Facility ซึ่งแต่ละ Facility จะประกอบไปด้วย Attribute 3 ตัว ได้แก่ AISLE, ROW และ QTY_ON_HAND ซึ่งในกรณีกลับกันก็เป็นจริงสำหรับทุกๆ Facility

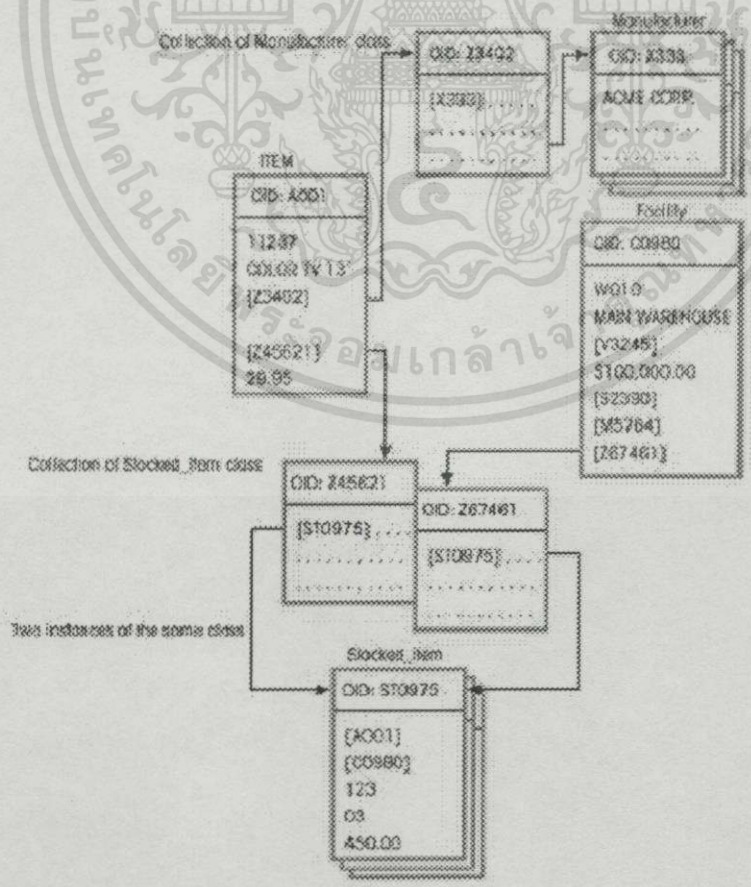


รูปที่ 4.12 ความสัมพันธ์แบบ M:N และ attribute ที่เกี่ยวข้อง

ถ้าจะให้มองในมุมของ Relational ที่เพิ่มขึ้นสำหรับความสัมพันธ์แบบ M:N นี้ ก็จะต้องกำหนด Intersection (bridge) class เพื่อที่จะเชื่อมทั้ง Facility กับ Item และ Attribute ที่เกี่ยวข้อง โดยทำการสร้างคลาส Stocked_Item ซึ่งประกอบไปด้วย Facility และ Item Instance และ Attribute AISLE, ROW และ QTY_ON_HAND ตามรูปที่ 4.13



รูปที่ 4.13 ความสัมพันธ์แบบ M:N กับ Intersection class



รูปที่ 4.14 Object Space

รูปที่ 4.14 แสดง Object Space ตามแนวคิดข้างต้น อย่างไรก็ดีการออกแบบเช่นนี้ก็เป็นการออกแบบที่ต้องระมัดระวังเป็นพิเศษ โดยมีจุดที่ต้องสนใจคือ

- Stocked_Item Object Instance จะมีการอ้างอิงไปยัง Instance ของคลาสอื่นที่เกี่ยวข้อง ซึ่งการทำ Intersection Class เช่นนี้ก็เพื่อต้องการติดตามข้อมูลที่เพิ่มเข้ามาเท่านั้น
- Item Object Instance ใน Object Schema นี้ประกอบไปด้วย collection of Stocked_Item Object Instance ซึ่งแต่ละอันก็ประกอบไปด้วย 1 Facility Object Instance ความสัมพันธ์ในทางกลับกันก็ยังคงเป็นจริง
- ใน Object Space การอ้างอิงระหว่างคลาสจะใช้ OID ของออบเจ็กต์ที่ถูกอ้าง
- ค่าที่อยู่ในวงเล็บ [ ] จะแทน OID ของ Object Instance ของบางคลาส เช่น ค่า Z3402 และ Z45621 เป็นต้น เป็น OID ของออบเจ็กต์ที่ถูกอ้างถึง ซึ่งก็คือ Collection ของ Manufacturer และ Stocked_Item ตามลำดับ

ใน Relational Model ตาราง ITEM จะไม่มีข้อมูลที่เกี่ยวข้องกับ MANUFACTURER หรือ STOCKED_ITEM เลย ซึ่งถ้าต้องการใช้ข้อมูลที่เกี่ยวพันกันของทั้ง 3 ตาราง จะต้อง Join ตารางเข้าด้วยกัน แต่ OODM ไม่ต้องการการ Join เพราะออบเจ็กต์ Item จะมีการอ้างอิงไปยังออบเจ็กต์อื่นที่เกี่ยวข้องอยู่แล้ว การอ้างอิงนี้จะนำออบเจ็กต์ Item เข้าสู่ Object Space โดยอัตโนมัติเมื่อถูกเรียกใช้

4.4 ระบบจัดการฐานข้อมูลเชิงวัตถุ (OODBMS)

OODBMS เป็นระบบฐานข้อมูลที่นำเอาแนวคิดเชิงวัตถุมาผสานกับเทคโนโลยีทางด้านฐานข้อมูล โดยยังไม่มีมาตรฐานที่ใช้กำหนดถึงคุณสมบัติของ OODBMS ที่ต้องสามารถทำได้ ทำให้ OODBMS ของแต่ละผู้ผลิตมีอยู่มีคุณสมบัติแตกต่างกัน อย่างไรก็ดี OODBMS ก็มีคุณสมบัติที่จำเป็นต้องมีอยู่ 14 ข้อด้วยกัน ซึ่งสามารถแบ่งได้เป็น 2 ส่วน ดังนี้

ส่วนของ OO System

1. ระบบต้องสนับสนุน Complex Object
2. สนับสนุน Object Identity (OID)
3. มีคุณสมบัติ Encapsulation
4. ระบบต้องสนับสนุน Type และ Class
5. ระบบต้องสนับสนุนการสืบทอดคุณสมบัติ (Inheritance)
6. ระบบต้องมีคุณสมบัติ Overriding, Overloading และ late binding
7. ระบบต้องเป็น Computational Complete
8. ระบบต้อง Extensible
9. ระบบต้องจัดการ Persistence ให้

ส่วนของ DBMS

10. ระบบต้องมีการจัดเก็บข้อมูล (Secondary storage managment)

11. ต้องสามารถจัดการกับฐานข้อมูลที่มีขนาดใหญ่ได้

12. สนับสนุนการทำงานแบบ Concurrency

13. ต้องสามารถกู้ระบบ (Recovery) จากการทำงานที่ผิดพลาดของ Hardware และ Software ได้

14. ต้องสามารถทำการค้นหาข้อมูลได้ง่าย (Ad-hoc query)

4.5 มาตรฐาน ODMG

ระบบฐานข้อมูลเชิงวัตถุได้รับการพัฒนาอย่างต่อเนื่อง ซึ่งระบบฐานข้อมูลเชิงวัตถุนี้มีคุณสมบัติต่าง ๆ ที่สามารถรองรับข้อมูลที่ซับซ้อน นอกจากนี้ยังมีองค์ประกอบต่าง ๆ ที่สามารถในงานขนาดใหญ่ได้ แต่เหตุผลประการหนึ่งที่ทำให้ระบบฐานข้อมูลประเภทนี้ยังใช้งานกันไม่แพร่หลายก็คือ การไม่มีมาตรฐานรองรับ ซึ่งความพยายามที่จะสร้างมาตรฐานให้กับระบบฐานข้อมูลเชิงวัตถุนี้ได้ดำเนินการเรื่อยมา โดยมีหน่วยงาน The Object Data Management Group (ODMG) มีหน้าที่ในการสร้างมาตรฐานดังกล่าว ซึ่งมาตรฐานที่หน่วยงานนี้ประกาศใช้ล่าสุดคือ ODMG 2.0 โดยองค์ประกอบของ ODMG 2.0 นี้แบ่งออกเป็น 3 ส่วนคือ

4.5.1 โมเดลของออบเจ็กต์ (The ODMG Object Model)

ในส่วนของโมเดลของออบเจ็กต์นั้น เป็นการกำหนดคุณลักษณะของออบเจ็กต์ที่ควรจะมี โดยออบเจ็กต์จะต้องมีคุณสมบัติดังนี้

ในระบบ database แบบเก่า คำว่า entity type นั้นจะหมายถึง class และ entity จะหมายถึง object แต่ใน ODMG Model entity และ object มีความหมายไม่ต่างกันนัก ODMG Model สนับสนุน complex object ผ่าน entity(object) และ literal (value)

- Class : ทุกๆ ออบเจ็กต์จะเป็น instance ของ class ใน class หนึ่งๆ จะประกอบไปด้วย attribute และ method value ของ attribute อาจเป็นการ reference ไปยังออบเจ็กต์อื่น หรือ atomic literal หรือ collection อย่างเช่น set ก็ได้ และ class ก็สามารถมี attribute และ method เป็นของตัวเองได้ เรียก class attribute และ class method ซึ่งจะต่างจาก instance attribute และ instance method

- Object identity : แต่ละ instance ของคลาส จะมี object identity แต่ literal จะไม่มี และออบเจ็กต์สามารถ share ได้ ด้วยการ referential share

- Literal : คือ atomic value หรือ structured atomic value อย่างเช่น set of integer โดย atomic literal นั้นอาจเป็น integer, date, decimal, real, Boolean, bytesequence รูปแบบของ decimal

คือ [precision, scale] precision หมายถึง จำนวนหลักของเลขหน้าจุดทศนิยม มีค่าตั้งแต่ 1-18 scale หมายถึง จำนวนหลักหลังจุดทศนิยม string และ bytesequence มีขนาดไม่เกิน 64 k

- Collection : collection มีหลายๆรูปแบบ ไม่ว่าจะเป็น set, bag, list, array และสามารถกำหนดเป็น collection ของ literal หรือออบเจ็กต์ก็ได้ แต่ไม่สามารถ referential share ได้ set คือ collection ที่แต่ละค่าใน collection ไม่ซ้ำกันและไม่เรียงลำดับ bag จะเหมือน set แต่ค่าใน collection ซ้ำได้ list คือ collection ของข้อมูลที่มีลำดับ และเราสามารถจะ retrieve ค่าของสมาชิกตัวแรก ตัวสุดท้าย หรือ ณ ตำแหน่งที่ต้องการได้ array คือ collection mมีจำนวนสมาชิกแน่นอน และสามารถ access ค่าได้จากตำแหน่งของ element นั้น

- Tuple : รูปแบบของ tuple คือ [a1:v1, a2:v2, ..., an:vn] attribute a1, a2, ..., an จะหมายถึง attribute name value v1, v2, ..., vn จะเป็น atomic literal value (อย่างเช่น string, integer) หรือ object หรือ collection of literal value หรือ collection of object ก็ได้ tuple จะถูกใช้เพื่อวัตถุประสงค์เดียวเท่านั้น นั่นคือ เพื่อกำหนด data structure ซึ่งจะกำหนดไว้ใน ODQL expression และจะไม่ปรากฏใน persistent Jasmine database

- Relationship: attribute ของ object สามารถอ้างอิง (reference) ไปยัง object อื่นๆได้ ODQL สนับสนุนทั้ง single-value (single instance ของ object) และ collection-value (set of object)

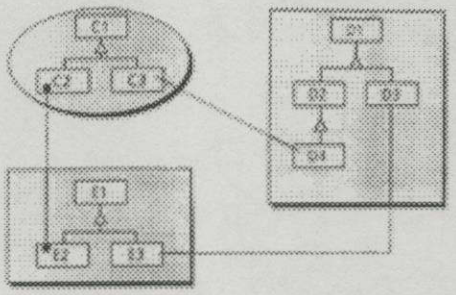
- Inheritance คลาสต่าง ๆ ต้องสามารถถ่ายทอดคุณสมบัติไปยังคลาสลูก ๆ ได้

4.5.2 ภาษากำหนดรูปแบบเชิงวัตถุ (Object Definition Language : ODL)

ในมาตรฐาน ODMG 2.0 นี้ได้มีการกำหนดรูปแบบของภาษา ODL ไว้เพื่อเป็นภาษาในการสร้างออบเจ็กต์ เพื่อนำไปใช้งานในระบบฐานข้อมูลอีกทอดหนึ่ง ซึ่งมีรายละเอียดดังนี้

การกำหนดคลาสใน ODQL

ใน ODL คลาสจะถูกจัดให้อยู่ใน class family (class family คือ logical organization ของคลาส) อย่างเช่น คลาส ที่เกี่ยวกับ user และ group อาจจัดอยู่ใน class family เดียวกัน class family สามารถอ้างอิง object ใน class family อื่นได้ คลาส สามารถ inherit กันได้จากคลาส ภายใน class family เดียวกัน แต่สามารถมี attribute (property) ที่อ้างอิง object ใน class family อื่น ดังในรูปที่ 4.15



รูปที่ 4.15 class family

เมื่อเรากำหนด คลาส เราจำเป็นต้องกำหนด class family ด้วย คลาส หนึ่งๆจะอยู่ใน class family ได้เพียง คลาส เดียวเท่านั้น ชื่อของคลาสจะต้อง unique ภายใน class family หนึ่งๆ แต่เราสามารถมีคลาสชื่อเดียวกันในคนละ class family ได้

ทุกๆ persistent object ใน ODL จะต้องเป็น instance ของคลาสและในการกำหนด คลาส นั้นจำเป็นต้องมี

- ชื่อ คลาส
- **superclass**: จะมีหรือไม่ก็ได้ ทุกๆ user-defined class จะเป็น subclass หรือ subordinate class ของ system class Composite โดยอัตโนมัติ superclass ที่กำหนดใน class definition จะเป็น immediate superclass ของ class ที่กำหนด เช่นเดียวกับ object-oriented language ทั่วๆไป
- **characteristic (method และ property) ของ คลาส**: Jasmine สนับสนุน characteristic หลายๆรูปแบบ ดังนี้

- *Instance characteristic*: เป็น characteristic ที่สำคัญที่สุด ได้แก่ instance-level property และ method

- *Class characteristic*: ได้แก่ class-level property และ method โดย class-level property จะถูก share โดยทุกๆ instance ของ คลาส นั้น เช่น class User มี attribute average salary ซึ่งเก็บเงินเดือนเฉลี่ยของพนักงานทุกคนใน คลาส นั้น

- *Method on collection*: ODQL สนับสนุน method ที่จะทำงานกับ collection ของ object หรือของ คลาส อย่างเช่น method ที่ทำหน้าที่นับจำนวนของ instance ภายใน collection หรือ method ที่ return instance ตัวแรกและตัวสุดท้ายใน list และ method ที่ทำงานกับ collection of class นั้น คลาส ใน collection ควรจะเป็น คลาส นั้น หรือเป็น subordinate ของ คลาส นั้น

เราอาจกำหนดค่าเริ่มต้นให้กับ class-level หรือ instance-level property ได้ และสำหรับ instance-level property นั้นเราสามารถกำหนด constraint อื่นๆ เช่น กำหนดให้ property นั้นเป็นแบบ "mandatory" คือ จะห้าม property ตัวนั้นมีค่าเป็น null เป็นต้น และสำหรับ property ที่ไม่ใช่

collection เราอาจกำหนดให้ property ตัวนั้นต้อง "unique" ซึ่งก็เหมือนกับ uniqueness constraint ใน relational DBMS ทั่วๆไป คือ ค่าของ property ตัวนี้จะต้อง unique ใน คลาส นั้นๆ ซึ่ง Jasmine system จะคอยตรวจสอบทุกๆครั้งที่มีการสร้าง object ใหม่ขึ้นมาและทุกครั้งที่ค่าของ property นี้ถูกแก้ไข

การกำหนด method สามารถทำได้โดย กำหนด return type ของ method นั้น ชื่อ method และ parameter ที่ต้องการ

รูปแบบทั่วๆไปของ class definition คือ

```
DefineClass class-identifier
[super : class-identifier [ {,class-identifier}...] ]
[description : string-constant]
{ [ maxInstanceSize : size; ]
characteristic } ;
```

ตัวอย่างของการกำหนดคลาส

```
DefineClass     User
super :  applicationObject
description : "the class implementing users"
{
  class :
    User createObject(Session Session, String lastName
                      String firstName, String mi,
                      String loginName, String password);
/* This method will retrieve all users except the
** user that have the status of "Out of Service".
*/
    User getUserWithName(Session Session, String name);
  Instance:
    String lastName       default: "";
    String firstName      default: "";
    String middleInitial  default: "";
```

```
        Department          userDepartment;
        Set<Manager>                managers;
        Manager             mainManager;
        Integer salary;
        String rank         default: "MTS";
        BasicAddress                address;
        Company             userCompany;
        Date                        dataHired;
        Real hourRate               default: 0;

        Void setUserName(Session Session,
                            String lastname, String firstName,
                            String middleInitial);
    /* The last, first amd middle name through these accessors
    ** are obtained and stored in the parameters
    */
        String getLastName(Session Session);
        String getFirstName(Session Session);
        String getMiddleInitial(Session Session);
        String getFullName(Session Session);
        Void setHomeAddress(Session Session, BasicAddress newAddress);
        BasicAddress getHomeAddress(Session Session);
        Void setHourRate(Session Session);
        Real getHourRate(Session Session);

    /* Other method */
        Set<Manager> getManagers();
        Real evaluationBonus();
        Manager getMainManager();
    ...
};
```

ตัวอย่างของ ODL ในการสร้างคลาส

```
interface Person
// type properties:
(       extent Persons )
{
// instance properties:
        attribute String Name;
        attribute enum Gender{ F,M };
        attribute Date D-Birth;

// instance operations:
        Short Age();
};

interface Employee:Person
(       extent Employees)
{
        attribute Unsigned Short Salary;
        relationship Set<Skill> Has_Skills
                inverse Skill::Belongs_to;
        relationship Department Works_for
                inverse Department::Employees;
        relationship Department Manages
                inverse Department::Manager;

        Unsigned Short AvgSalary();
};

interface Skill
(       extent Skills
        key Name)
{
        attribute String Name;
```

```
        relationship Set<Employee> Belongs_to
                inverse Employee::Has_Skils;
};

interface Department
(       extent Departments
        keys Name)
{
        attribute String Name;
        attribute Long Budget;
        relationship Set<Employee> Employees
                inverse Employee:: Works_for;
        relationship Employee Manager
                inverse Employee:: Manages;
        Unsinged Short NumOfEmployee();
};
```

4.5.3 ภาษาเรียกค้นเชิงวัตถุ (Object Query Language: OQL)

โดยทั่วไปแล้ว มาตรฐานของการเขียนโปรแกรมบนฐานข้อมูล ก็จะประกอบไปด้วย data definition sublanguage ที่ใช้ในการกำหนด persistent database object, การเรียกค้นใช้เพื่อเรียกค้นออบเจ็กต์จากฐานข้อมูล, data manipulation construct เพื่อแก้ไข, เพิ่มและลบออบเจ็กต์ ซึ่งในระบบฐานข้อมูลเชิงสัมพันธ์ มาตรฐานสำหรับ data definition และ data manipulation (ทั้งการ query, insert, update, delete data) คือ SQL ซึ่ง SQL นี้จะเป็นทั้ง Data Definition Language (DDL) และ SQL Data Manipulation Language (DML) สำหรับระบบฐานข้อมูลเชิงวัตถุแล้ว ตามมาตรฐาน ODMG 2.0 จะใช้ Object Definition Language (ODL) และ Object Query Language (OQL) ตามลำดับ

ODMG standard (version 2.0) นั้นไม่ได้ให้ความสนใจกับปัญหาใหญ่ข้อหนึ่งของ object-oriented database มากนัก ปัญหานั้นคือ ความไม่เข้ากันระหว่าง programming language และ database sublanguage (the impedance mismatch between a programming language and a database sublanguage) มีการพูดถึงปัญหาข้อนี้มาตั้งแต่กลางทศวรรษที่ 80 เนื่องจากความไม่เข้ากันของ host programming language อย่างเช่น C กับ database sublanguage อย่าง SQL เนื่องจาก SQL เป็นภาษา

ที่ไม่ computational complete ผู้พัฒนาระบบจึงต้องยุ่งเกี่ยวกับภาษาทั้ง 2 ภาษา ยังไม่นับถึงปัญหาเรื่องการ matching value, ชนิดของตัวแปร และอื่นๆอีกมากมาย

แต่มาตรฐานของ ODMG ก็ยังไม่ได้สนใจที่จะแก้ปัญหาข้อนี้ กลับไปเพิ่ม CORBA's Interface Definition Language (IDL) (ซึ่งเรียกว่า ODL) และ query sublanguage (เรียก OQL) เข้ามา ซึ่งทำให้สามารถติดต่อ (binding) กับ object-oriented language อื่นๆได้เป็นจำนวนมาก และมีข้อกำหนดสำหรับ C++ binding, Smalltalk binding หรือล่าสุด Java Binding ซึ่งทำให้สามารถมี method ที่ส่ง OQL statement เป็น argument ไปประมวลผลยัง object-oriented database engine ที่อยู่ด้านล่างได้

โครงสร้างการเขียนโปรแกรมของ OQL อาจสรุปได้ดังนี้

4.5.3.1 ตัวแปร

การประกาศตัวแปรใน OQL มีรูปแบบดังนี้

<Class Name> <Variable Name>;

อย่างเช่น User e;

เป็นการประกาศว่า e เป็น instance ของ class User

ตัวแปรใน ODQL สามารถอ้างอิงกับ entiry (object หรือ instance), atomic literal, class, tuple รวมทั้ง collection of entity อย่างเช่น bag of object หรือ collection of class เช่น ทุก subclass ของ class Person หรือ collection of tuple ตัวอย่างเช่น

```
String name;
Bag<User> us;
Document class DC;
ResultTuple [String] lastName,
                Company userCompany, Integeer salary]
UserInfo;
```

scope ของตัวแปรใน ODQL จะเหมือนกับใน blocked-structured language ทั่วๆอย่างเช่น C\C++ ตัวแปรที่ประกาศไว้ใน block ก็จะคืนหน่วยความจำเมื่อออกจาก block ถ้าเราประกาศตัวแปรใน ODQL interpreter prompt (ซึ่งไม่เป็น block) ค่าของตัวแปรก็จะคงอยู่ไปเรื่อยๆจนกว่าจะจบ session การทำงาน

4.5.3.2 Expression ใน OQL

นอกจาก query และการประกาศตัวแปรแล้ว ODQL ยังสามารถใช้ expression ทางคณิตศาสตร์ทางตรรกศาสตร์ได้ด้วย

Expression ทางคณิตศาสตร์จะใช้เครื่องหมายทางคณิตศาสตร์ทั่วไป ได้แก่ บวก(+) ลบ(-) คูณ(*) หาร(/) หารเอาเฉพาะเศษ(remainder, %) โดยจะทำการคูณหารก่อนการบวกลบ จากซ้ายไปขวา และสามารถคำนวณได้ทั้งตัวเลขและตัวแปร ตัวอย่าง expression ทางคณิตศาสตร์ เช่น

(15 + YearsInCompany) * 2 – 20.5 * Bonus

โดยที่ YearsInCompany เป็นตัวแปรชนิด integer และ Bonus เป็นตัวแปร real expression ทางคณิตศาสตร์อาจอยู่ได้ทั้งใน ด้านขวาของ assignment statement หรืออยู่ใน query expression หรือที่อื่นๆก็ได้

Expression ทางตรรกศาสตร์ก็จะประกอบไปด้วยเครื่องหมาย เท่ากับ(==) ไม่เท่ากับ(!=) น้อยกว่า(<) น้อยกว่าหรือเท่ากับ(<=) มากกว่า(>) มากกว่าหรือเท่ากับ(>=) โดยสามารถใช้ได้กับ literal ทุกชนิด และเครื่องหมายเท่ากับและไม่เท่ากับสามารถใช้กับ object ได้ด้วย ซึ่งเครื่องหมายเท่ากับ(==) จะให้ค่า TRUE เมื่อ argument ทั้งสองอ้างอิงไปยัง object เดียวกัน

นอกจากนั้น ODQL ยังสนับสนุนเครื่องหมาย AND, OR, NOT ซึ่ง operator จะต้องเป็น Boolean expression เช่น

```
Boolean b;
Integer year;
User john, bob;
...
b = (year <= 1998 and john.evaluateBonus() < 100.0)
    or (john.getMainManager() ==
    bob.getDepartment().getMainManager());
```

Boolean expression จะใช้ใน where clause ของ query expression เพื่อระบุ object ที่ต้องการ และนอกจากนั้นยังใช้ใน if และ while เพื่อควบคุมลำดับการทำงานของ ODQL program

4.5.3.3 Queries

มีรูปแบบทั่วๆไปดังนี้

<target specification> **from** <from specification> **where** <where specification>

ซึ่งจะได้ผลลัพธ์ออกมาเป็น collection ของ object ซึ่งสามารถนำมาเก็บไว้ในตัวแปรได้ และตัวแปรต่างๆก็สามารถนำมาใช้ได้ทั้งใน target specification, from specification หรือ where specification ตัวอย่างเช่น

```
List<User> ee;
ee = User from User where User.Salary >= 50000;
```

โดยทั่วไปแล้ว ผลของการ query จะเก็บไว้ใน collection เดียวกับใน from specification ซึ่งโดยมากก็จะเป็น instance ของ class นั้นนั่นเอง

เช่น เราสนใจชื่อพนักงานและชื่อผู้จัดการใน class User โดยให้ผลลัพธ์มาเป็น list ของ tuple ซึ่งกำหนดใน target specification

```
List< empNameManagers [empName : String, empManagers : Set
<User>]> ee;
Integer s;
...
ee=[u.lastName, u.Managers] from Users u
    where u.Salary >=s and u.rank=="Senior";
```

จะเห็นว่าเราสามารถใช้ตัวแปรใน where specification ได้อย่างไร และ target specification เป็น tuple ซึ่งทำให้ผลลัพธ์ที่ได้มาอยู่ในรูปของ set of tuple และ type ของผลลัพธ์ที่ได้ออกมากับตัวแปร ee จะต้องเป็นชนิดเดียวกัน

4.5.3.4 Path Expressions และ joins

path expression จะถูกใช้ในการ traverse relationship ระหว่าง class เช่น สมมติให้ u เป็น instance ของ class Users เราสามารถอ้างถึงเงินเดือนของผู้จัดการของพนักงานคนนั้นได้ดังนี้

```
u.mainManager.salary
```

path expression ไม่เพียงสามารถอ้างถึง object-valued property(object reference) ได้เท่านั้น แต่ยังสามารถอ้าง method ได้ด้วย อย่างเช่น

```
u.getMainManager().salary
```

เนื่องจากเราสามารถกำหนด path expression ได้โดยตรงเช่นนี้ จึงมักมีเพียงหนึ่ง collection เท่านั้นใน from specification เช่น ถ้าเราต้องการ lastname ของ user ที่ main manager department อยู่ใน New York ก็สารถทำได้ง่ายๆโดยใช้คำสั่ง

```
Set<String> us;
Us = u.lastaName from User u where
        u.mainManager.userDepartmant.address.city == "New York";
```

ซึ่งถ้าไม่ใช้ path expression และความสามารถในการอ้างอิงกันของ object แล้ว มันก็จะเป็น query ที่ซับซ้อนมาก หรือในบางครั้งเราอาจมีมากกว่าหนึ่ง collection ใน from specification ก็ได้ อย่างเช่น query ด้านล่างนี้จะให้ผลมาเป็น ทุกๆ pair of user ที่อาศัยอยู่ในเมืองเดียวกัน

```
[u1.lastName, u2.lastName] from User u1, User u2
where u1.basicAddress.city==u2.basicAddress.city;
```

หรืออาจเป็นคนละ collection กันก็ได้ เช่น query ที่ให้ lastName ของหนักงานทุกๆคนที่ได้รับเงินเดือนมากกว่า manager สามารถเขียนได้ดังนี้

u.lastName **from** User u, Manager m **where** u.salary>=m.salary;

4.5.3.5 Exclusive Queries

ใน form specification ถ้าชื่อ class นำหน้าด้วย keyword "alone" นั่นหมายความว่า การค้นหาจะทำที่ instance ของ class นั้นเท่านั้น จะไม่นำเอา instance ของ subordinate class ของ คลาสนั้นมารวม

4.5.3.6 Control Flow Constructs

if-else Statement : มีรูปแบบทั่วไปดังนี้

```
if (<boolean expression>) {
        <statement>
}
else {
        <statement>
};
```

ถ้าค่าใน boolean expression เป็น TRUE ก็จะไปทำ <statement> ใน if clause ถ้าไม่ก็จะไปทำใน else clause

while statement: มีรูปแบบดังนี้

```
while(<boolean expression>) { <statement> };
```

ถ้าค่าใน boolean expression เป็นจริง ก็จะทำ <statement> ซ้ำๆไปเรื่อย จนกว่าค่าใน boolean expression จะเป็น FAULSE หรือ NIL ก็จะหยุดทำ แล้วออกจาก loop และเราสามารถใช้คำสั่ง break เพื่อบังคับให้ออกจาก loop ทันทีที่โปรแกรมทำงานมาจนถึงคำสั่งนี้ได้อีกด้วย

scan Statement: เป็นคำสั่งที่ใช้กำหนดการทำงานซ้ำๆบนแต่ละ element ของ collection เช่น print ค่าของทุกๆ element เป็นต้น รูปแบบคือ

```
scan (<collection expression variable>,
        <collection element variable> )
{ <statement> };
```

<collection expression variable> หมายถึง collection ที่ต้องการจะทำซ้ำๆ ส่วน <collection element variable> ใช้อ้างถึง element ใน collection เช่น ถ้าเราต้องการแสดงชื่อของทุกๆ element ใน group ก็จะทำได้ดังนี้

```
set<group> GS, ge;
...
scan ( GS, ge) {
        m = ge.name();
        m.print();
}
```

เมื่อครบทุกๆ element ใน GS แล้ว ก็จะออกจาก loop และ ge ก็จะถูก set ให้เป็น NIL และสามารถใช้คำสั่ง **break** ได้เช่นเดียวกับ while โดย ge ก็จะต้องถูก set ให้เป็น NIL เช่นเดียวกัน

4.5.3.7 Collection (Set) Operators

ใน OQL เราสามารถนำคำสั่งที่เกี่ยวกับ set มาใช้กับ collection ได้ ถ้ามีคำสั่ง Op มากระทำกับ set s1 และ s2

```
S1 Op s2
```

OQL syntax จะเป็น

```
s1.Op(s2)
```

s1 คือ target object Op คือ operation r2 เป็น argument โดย operation ที่ ODQL สนับสนุนได้แก่ union, intersection และ differ

ยกตัวอย่างเช่น บริษัท 2 บริษัท Acme1 และ Acame2 มี method branchesInCities ซึ่งจะ return set ของเมืองทุกเมืองที่บริษัทมีสาขาอยู่ ถ้าเราต้องการทราบว่าเมืองใดใน California บ้างที่ทั้งสองบริษัทนี้มีสาขาอยู่ ก็จะเขียนได้ดังนี้

```
caAcme1 = Acme1.branchesInCities(CA);
caAcme2 = Acme2.branchesInCities(CA);
commonCities = caAcme1.intersect(caAcme2);
```

หรือถ้าต้องการทราบว่าทั้งสองบริษัทมีสาขาอยู่ที่เมืองใดบ้าง ก็จะเขียนได้ว่า

```
UnionOfBranches = caAcme1.union(caAcme2);
```

หรือถ้าต้องการทราบว่ามีสาขาในเมืองใดบ้างที่บริษัท Acme1 มีสาขาอยู่แต่ Acme2 ไม่มี

```
Acme1Only = caAcme1.differ(caAcme2);
```

4.5.3.8 Populating และ Depopulating Collections

OQL จะมี method สำหรับเพิ่มเข้าและลบ element ออกจาก collection ได้ โดยการเพิ่ง element เข้าใน collection ทำได้โดยใช้ method **add()** และ **directAdd()** ส่วนการลบ element ออกจาก collection จะใช้ method **remove()** และ **directRemove()** เช่น

```
City walnutCreek;
City set branches;
...
Company acme;
...
branches = acme.hasBranches;
branches = branches.add(walnutCreek);
acme.hasBranches = branches;
...
```

คำสั่งข้างต้นนี้จะ load สาขาทั้งหมดที่บริษัทนี้มีไว้ใน working space แล้ว เพิ่ม element เข้าไป จากนั้นก็นำกลับไปใส่ใน property ชื่อ hasBranches ตามเดิม ซึ่ง วิธีนี้มีข้อเสียคือ สิ้นเปลืองทั้งเวลา เนื้อที่ และ I/O ที่ต้องขนถ่ายข้อมูลที่ไม่ได้ใช้ไปมา

เพื่อหลีกเลี่ยงปัญหาเหล่านี้ Jasmine จึงมี method **directAdd()** ขึ้นมา โปรแกรมข้างต้นสามารถเขียนแทนได้ด้วย

```
Acme.hasBranches.directAdd(walnutCreek);
```

และสำหรับการลบ element ออกจาก collection ก็จะใช้คำสั่ง **remove()** หรือ **directRemove()** ซึ่งก็เช่นเดียวกับคำสั่ง **add()** คือ **remove()** จะกระทำกับ collection บน working space ส่วน **directRemove()** ก็จะทำกับ DBspace โดยตรง

4.5.3.9 Method อื่นๆของ Collections

- Aggregate function: เช่น method **count()** ซึ่งจะนับจำนวนของ element ใน collection นั้น

- สำหรับ element ของ collection ที่เป็น numeric จะมี method **sum()**, **average()**, **max()** และ **min()** ซึ่งใช้ในการรวมค่า หาค่าเฉลี่ย ค่าที่มากที่สุด น้องที่สุดของ element ใน collection

- Sorting: method **sort()** จะมี argument 2 ตัวคือ attribute ที่ต้องการเรียง และลำดับการเรียง (จากน้อยไปมา (ASCEND) หรือมากไปน้อย (DESCEND)) เช่น ถ้าเราต้องการเรียงลำดับและ print collection ชื่อ branches โดยเรียงตามชื่อเมืองแบบมากไปน้อยก็จะเขียนได้ว่า

```
branches.sort("city name", DESCEND).print();
```

- Duplicate elimination: ถ้า collection ของเรามี literal หรือ object ที่ซ้ำกันอยู่ เราสามารถใช้ method **unique()** กำจัดตัวซ้ำเหล่านั้นได้

4.5.3.10 Update, Creating and Deleting Persistent Objects

ใน ODQL เราสามารถ create และ delete object ได้แม้กระทั่งใน transaction การ update จะมองเห็นเฉพาะ work space ของตนเองจนกระทั่ง transaction นั้น commit ผู้ใช้อื่นจึงเห็นการเปลี่ยนแปลงนั้น

- การ Update object: มี 2 แนวทาง วิธีแรกคือ เรียก method สำหรับ update attribute นั้น ซึ่งเป็นวิธีที่ดี เพราะจะทำให้การเปลี่ยนแปลง implementation ภายใน คลาส ไม่มีกระทบกับ application program เช่น

```
john.changeStatus("vacationing");
```

และวิธีที่ 2 ใช้ assignment statement ใน OQL version ปัจจุบันจะยังไม่มีการ encapsulation ทำให้เราสามารถแก้ไข property ของ object ได้โดยตรง เช่น

```
john.status = "Vacationing";
```

- การสร้าง object: OQL จะมี method **new()** สำหรับ create object เช่น

```
classTask t;
t = classTask.new();
```

หรือ

```
t = classTask.new(priority := "High",
                  Status := "Open");
```

- การลบ object: OQL จะมี method **delete()** ใช้ในการลบ object ออกจาก database เช่น

```
t.delete();
```

แต่ OQL จะไม่จัดการ relationship ระหว่าง object ให้ เช่น ถ้ามี object reference object ที่เราต้องการลบ OQL ก็จะไม่จัดการให้ ซึ่งอาจทำให้เกิด database error ขึ้นได้

ปัญหานี้อาจแก้ไขได้โดย เราเป็นผู้จัดการกับ relationship นั้นเอง เช่น object user จะอ้างอิงไปถึง object task เราก็อาจกำหนดให้ method delete ของ task ไปลบตัวเองออกจาก object user ด้วย เป็นต้น

ตัวอย่างภาษา OQL ในการค้นหาชื่อและประสบการณ์ของพนักงานที่ทำงานในแผนก Toy และมีเงินเดือนมากกว่า 10,000

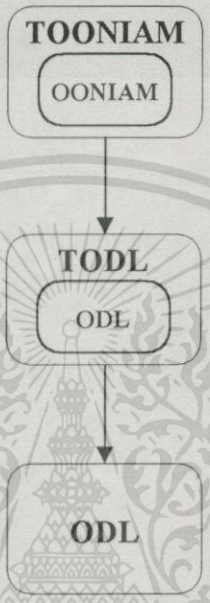
```
select struct(name: E.Name, skills: E.Has_Skills)
from E in Employees
where E. Works_for.Name = "Toy"
and E.Salary > 10000
```

ผลิตภัณฑ์ระบบฐานข้อมูลเชิงวัตถุที่มีอยู่ในตลาดซอฟต์แวร์ยังไม่สามารถรองรับมาตรฐาน ODMG 2.0 นี้ ได้อย่างครบถ้วน ดังนั้นแต่ละระบบฐานข้อมูลก็มีการจัดการที่ไม่เหมือนกัน บางผลิตภัณฑ์ก็มีภาษาเป็นของตัวเองไม่ได้รองรับ ODQL เลยก็มี ซึ่งในระยะเวลาอันใกล้ปัญหาเหล่านี้ก็จะหมดไปเพราะ ผู้ผลิตระบบฐานข้อมูลจะต้องผลิตตามมาตรฐานที่ตั้งขึ้น

บทที่ 5

แบบจำลองแนวคิด TOONIAM และการแปลงรูป

การสร้างแบบจำลองแนวคิด TOONIAM ให้สามารถทำงานร่วมกับฐานข้อมูลเชิงวัตถุได้ นั้นมีขั้นตอนดังรูปที่ 5.1



รูปที่ 5.1 ขั้นตอนการทำงานเพื่อนำ TOONIAM ไปทำงานร่วมกับฐานข้อมูลเชิงวัตถุ

จากรูปที่ 5.1 TOONIAM เป็นโมเดลสำหรับการออกแบบฐานข้อมูลเชิงเวลา เป็นโมเดลที่นำ NIAM เดิมมาเพิ่มขยายความสามารถให้รองรับแนวคิดเชิงวัตถุและข้อมูลเชิงเวลาได้ สำหรับ TODL นั้นเป็นภาษานิยามเชิงเวลาบนฐานข้อมูลเชิงวัตถุ กล่าวคือได้นำเอา ODL ตามมาตรฐานของ ODMG มาเพิ่มขีดความสามารถให้รองรับข้อมูลที่แปรผันตามเวลาได้ หรือกล่าวอีกนัยหนึ่งก็คือ TODL ก็คือ ODL ที่เพิ่มส่วนของเวลาเข้าไปนั่นเอง โดยโมเดล TOONIAM มีโมเดล OONIAM เป็นโครงสร้างพื้นฐาน ซึ่งจะกล่าวรายละเอียดในหัวข้อ 5.1

ในบทนี้กล่าวถึง TOONIAMและการแปลงรูปเป็นภาษานิยามเชิงวัตถุ (ODL) ซึ่งจะแบ่งส่วนในการอธิบายดังนี้ หัวข้อ 5.1 เป็นการนำเสนอโมเดล OONIAM เพื่อทำให้ NIAM ที่มีอยู่เดิมสามารถรองรับแนวคิดเชิงวัตถุได้ ซึ่งในหัวข้อนี้จะได้กล่าวถึงโครงร่างหลัก (Main Schema) และโครงร่างย่อย (Sub Schema) ที่เป็นส่วนสำคัญในการสร้าง OONIAM โดยโครงร่างทั้งสองจะนำเสนอความสัมพันธ์ต่าง ๆ ของแอตทริบิวต์ และความสัมพันธ์ของออบเจ็กต์ ในหัวข้อ 5.2 อธิบายถึง TOONIAM โดยได้นำเอาหลักการของ OONIAM มาเพิ่มขยายส่วนของเวลาเข้าไป สำหรับเวลาที่เพิ่มเข้าไปนี้ประกอบด้วย Valid Time, Transaction Time และ Bitemporal Time [9][11]โดยผู้ใช้

สามารถเลือกใส่เวลาไว้ที่แอตทริบิวต์หรือความสัมพันธ์ต่าง ๆ ก็ได้ และผู้ใช้สามารถเลือกเวลาแบบใดก็ได้ ตามแต่ที่ได้ออกแบบไว้ สำหรับในหัวข้อ 5.3 จะกล่าวถึง การแปลง OONIAM ให้เป็น ODL โดยจะนำโครงร่างหลักและโครงร่างย่อยมาผ่านการประมวลผล เพื่อแปลงโครงร่างทั้งสองให้เป็นภาษา ODL หัวข้อ 5.4 อธิบายถึงการแปลงถึงการแปลง TOONIAM ให้เป็นภาษานิยามเชิงวัตถุอิงเวลา (TODL) โดยภาษานี้เป็นการนำเอาภาษานิยามเชิงวัตถุ (ODL) มาเพิ่มเติมส่วนของเวลาเข้าไป สำหรับในหัวข้อ 5.5 ซึ่งเป็นหัวข้อสุดท้ายนั้นจะกล่าวถึง หลักการแปลงภาษา TODL ให้เป็นภาษา ODL ซึ่งจะนำไปใช้งานกับระบบฐานเชิงวัตถุต่อไป สำหรับหัวข้อ 5.6 เป็นตัวอย่างที่ครอบคลุมเนื้อหาทั้งหมดในบทนี้

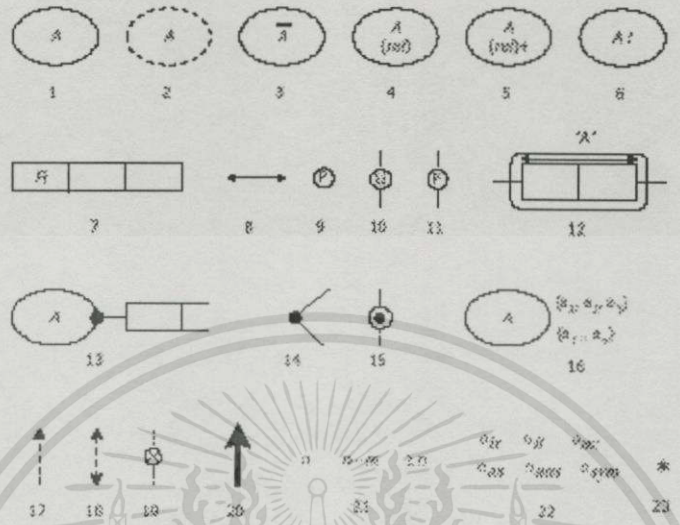
5.1 แบบจำลองแนวคิด OONIAM

เมื่อผู้ใช้งานต้องการที่จะออกแบบฐานข้อมูลบนระบบฐานข้อมูลเชิงสัมพันธ์ ผู้ใช้งานก็ต้องศึกษาเครื่องมือในการออกแบบฐานข้อมูล ซึ่งในปัจจุบันเครื่องมือที่เป็นที่นิยมนำมาช่วยในการออกแบบฐานข้อมูลมีด้วยกัน 2 เครื่องมือคือ ER และ NIAM ซึ่งเครื่องมือทั้งสองก็มีข้อดีข้อเสียที่แตกต่างกันออกไป การที่ผู้ใช้งานจะเครื่องมือใดนั้นก็แล้วแต่ความถนัด คุ้นเคยและประสบการณ์ของแต่ละคน แต่มีข้อสังเกตอยู่ประการหนึ่งก็คือ NIAM นั้นสามารถออกแบบฐานข้อมูลได้ถึงระดับ 5NF เลย แต่ ER ไม่สามารถทำได้เพราะ ER ไม่มีการระบุถึงความสัมพันธ์ระหว่างแอตทริบิวต์ จึงทำให้ ER สามารถออกแบบฐานข้อมูลได้แค่ 3NF เท่านั้น ซึ่งในการออกแบบฐานข้อมูลที่ใช้งานทั่วไปนั้น ส่วนมากแล้วการออกแบบแค่ 3 NF ก็สามารถทำงานได้แล้ว

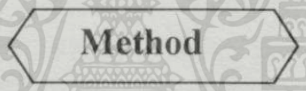
การใช้งาน ER และ NIAM นั้นจะใช้กับระบบฐานข้อมูลเชิงสัมพันธ์เท่านั้น แต่ในปัจจุบันระบบฐานข้อมูลมิได้มีเพียงระบบฐานข้อมูลเชิงสัมพันธ์เท่านั้น แต่มีระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ และระบบฐานข้อมูลเชิงวัตถุอีกด้วย ซึ่งระบบฐานข้อมูลทั้งสองได้รับการพัฒนาให้มีขีดความสามารถที่สูงขึ้น โดยมีแนวคิดให้สามารถรองรับข้อมูลและกระบวณการของข้อมูลที่สลับซับซ้อนมากขึ้นได้ เมื่อมีระบบฐานข้อมูลแบบใหม่ขึ้นมาแล้ว ก็ต้องมีการเครื่องมือในการออกแบบใหม่ด้วย ซึ่ง ER และ NIAM ที่มีแต่เดิมไม่สามารถใช้งานกับระบบฐานข้อมูลใหม่ทั้งสองนี้

นอกจากนี้หากผู้ใช้งานต้องการสร้างเป็นระบบฐานข้อมูลเชิงเวลาที่ทำงานบนระบบฐานข้อมูลเชิงวัตถุ ผู้ใช้จำเป็นต้องมีเครื่องมือในการออกแบบเช่นเดียวกัน ในหัวข้อนี้จะกล่าวถึงการนำเอา NIAM ที่มีแต่เดิมมาเพิ่มขยายความสามารถให้รองรับหลักการเชิงวัตถุได้ ซึ่งเครื่องมือใหม่นี้เรียกว่า โมเดลไนแอมเชิงวัตถุ (Object Oriented NIAM Schema Model) หรือ เรียกย่อ ๆ ว่า OONIAM เมื่อผู้ใช้งานสร้างแบบจำลองของฐานข้อมูลเชิงเวลาโดยใช้ OONIAM แล้ว ผู้ใช้งานสามารถแปลงแบบจำลองที่สร้างขึ้นให้เป็นภาษานิยามเชิงวัตถุ (ODL) ได้เลย

สำหรับหลักการสร้าง OONIAM นั้น จะนำสัญลักษณ์เดิมที่มีอยู่ใน NIAM ดังรูปที่ 5.2 มาใช้ร่วมกับสัญลักษณ์ใหม่ที่เพิ่มเติมเข้าไปดังรูปที่ 5.3 เพื่อให้ NIAM ใหม่นี้รองรับหลักการเชิงเวัตถุได้



รูปที่ 5.2 สัญลักษณ์พื้นฐานของ NIAM



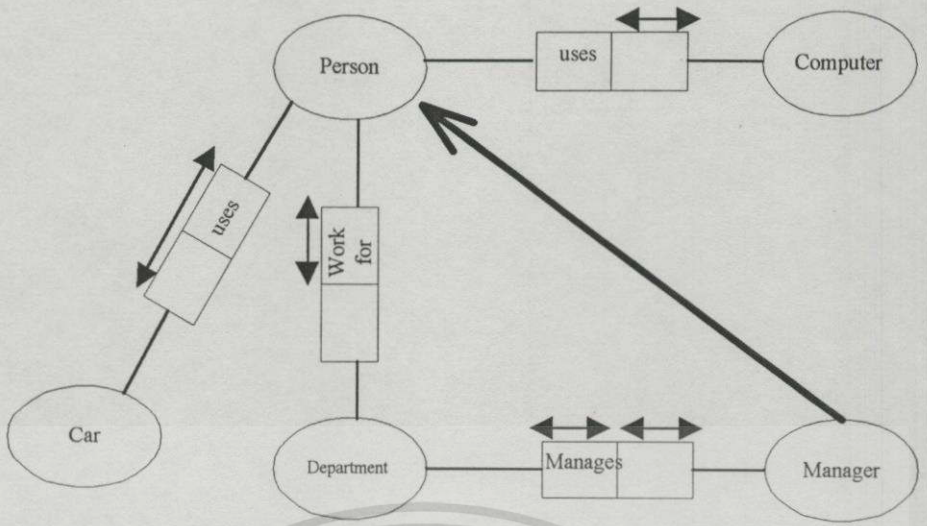
รูปที่ 5.3 สัญลักษณ์ของการประมวลผลข้อมูล (เมธอด)

สำหรับสัญลักษณ์ที่เพิ่มเข้าไปในรูปที่ 5.3 นั้นคือ กระบวณการประมวลผลข้อมูลนี้จะมีไว้ให้ผู้ใช้งานเพิ่มเข้าไปที่เอนติตี้ที่ต้องการ เพื่อให้เมธอดที่เพิ่มเข้าไปนี้จัดการกับข้อมูลที่ต้องการได้ โดยรายละเอียดจะกล่าวถึงในหัวข้อ โครงร่างย่อย (Sub Schema)

OONIAM แบ่งส่วนสำคัญออกเป็นสองส่วนดังนี้

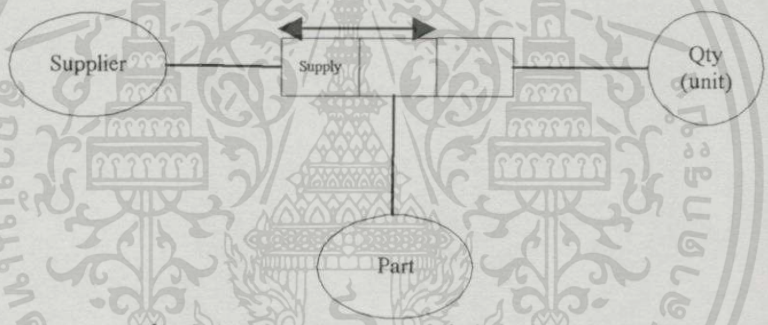
5.1.1 โครงร่างหลัก (Main Schema)

โครงร่างหลักนี้เป็นภาพรวมของความสัมพันธ์ระหว่างคลาส ดังนั้นในแต่ละคลาสจะไม่มีการแสดงแอตทริบิวต์ และไม่มี Uniqueness Identifier ดังตัวอย่างในรูปที่ 5.4



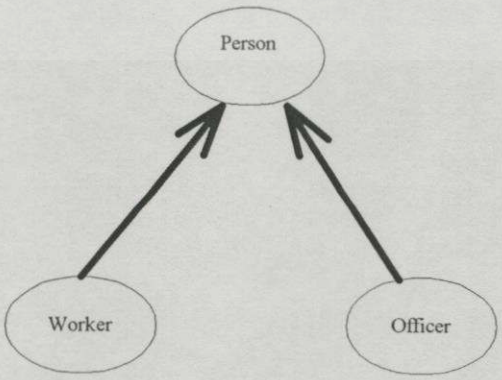
รูปที่ 5.4 โครงร่างหลักของคลาสหลายคลาส

หากมีความสัมพันธ์กันแบบ N-ary ก็จะสามารถมี Uniqueness Identifier ได้ที่ entity ดังรูปที่ 5.5



รูปที่ 5.5 โครงร่างหลักแบบที่มี Uniqueness Identifier ที่ entity

นอกจากการอธิบายถึงความสัมพันธ์ระหว่างคลาสแล้ว ในโครงร่างหลักยังได้กล่าวถึง การถ่ายทอดคุณสมบัติของคลาส (Inheritance) อีกด้วย ดังรูปที่ 5.6

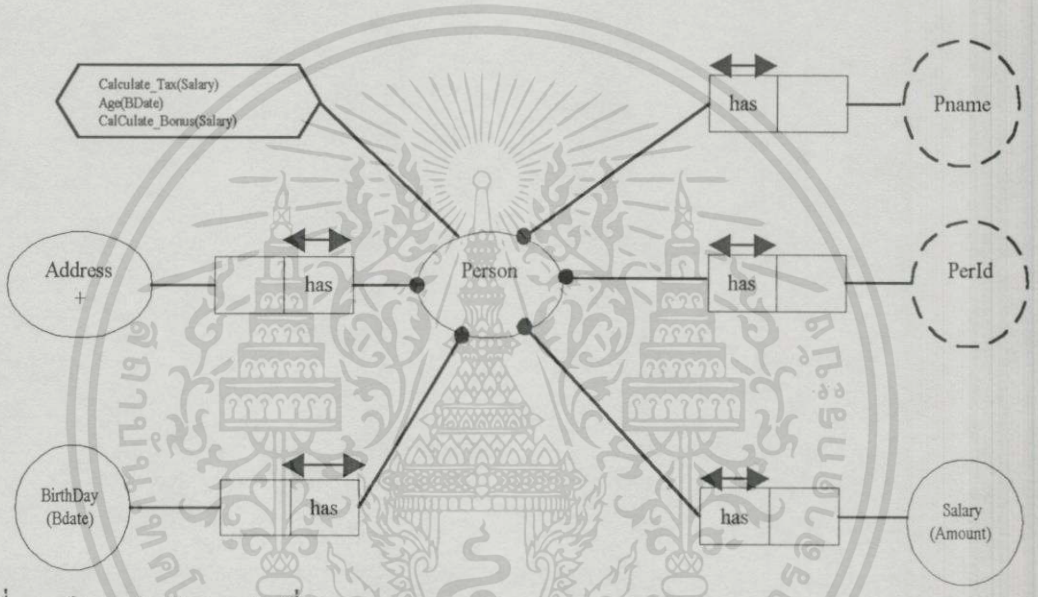


รูปที่ 5.6 การถ่ายทอดคุณสมบัติ

สำหรับโครงร่างหลักนี้จะแสดงความสัมพันธ์ระหว่างคลาสแบบหนึ่งต่อหนึ่ง หนึ่งต่อหลาย หลายต่อหนึ่ง หรือ หลายต่อหลายก็ได้ แล้วแต่การออกแบบคลาสนั้น ๆ ซึ่งแอตทริบิวต์ของแต่ละคลาสและ Uniqueness Identifier นั้นจะแสดงในโครงร่างย่อย

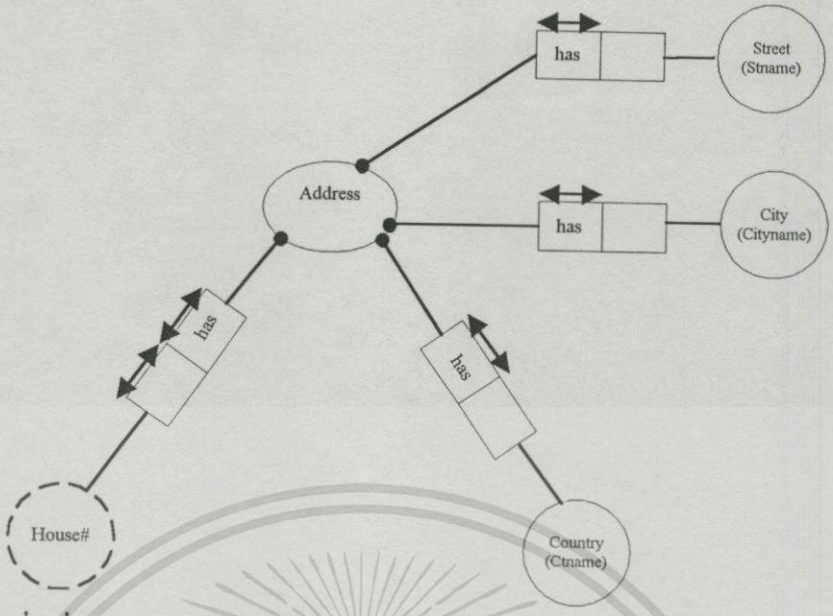
5.1.2 โครงร่างย่อย (Sub Schema)

โครงร่างย่อยเป็นการแสดงรายละเอียดของแต่ละคลาส ซึ่งแต่ละคลาสนั้นอาจเป็นคลาสที่ซับซ้อน (Complex Object) ก็ได้ กล่าวคือ คลาสดังกล่าวอาจมีคลาสอื่นเป็นส่วนประกอบก็ได้ โดยโครงร่างย่อยนี้ประกอบด้วย แอตทริบิวต์ , เมธอด และ คลาสที่อยู่ภายในคลาส ดังรูปที่ 5.7



รูปที่ 5.7 โครงร่างย่อยระดับที่ 1 ของคลาส Person

จากรูปที่ 5.7 เป็นการเขียนเพิ่มขยายโครงร่างหลักในรูปที่ 5.4 กล่าวคือในรูป 5.4 ได้บอกถึงความสัมพันธ์ของคลาส Person ที่มีต่อคลาสอื่น ๆ แต่ยังไม่ได้อธิบายถึงรายละเอียดของคลาส Person เมื่อต้องการอธิบายรายละเอียดต่าง ๆ ของคลาสดังกล่าวจำเป็นต้องอธิบายในโครงร่างย่อยเท่านั้น จากโครงร่างย่อยในรูปที่ 5.7 ซึ่งเป็นโครงร่างย่อยระดับที่ 1 จะเห็นว่าคลาส Person จะมีแอตทริบิวต์ต่าง ๆ ดังแสดงในรูป และมีเมธอดสำหรับจัดการกับคลาสนี้ด้วยกัน 3 เมธอด นอกจากนี้แล้วสัญลักษณ์ที่แสดงชื่อคลาสหลักนี้จะไม่มี Uniqueness Identifier ที่สำคัญคลาสนี้ยังมีคลาส Address เป็นคลาสย่อย (Embedded Class) อีก โดยคลาสย่อยนี้สแดงโดยการใช้เครื่องหมายบวก (+) ไว้ได้ชื่อคลาส ซึ่งหากต้องการเขียนโครงร่างย่อยของคลาส Address และ คลาส Land สามารถเขียนได้ดังรูปที่ 5.8 ซึ่งเป็นโครงร่างย่อยระดับที่ 2



รูปที่ 5.8 โครงร่างย่อยของคลาส Address

จากรูปที่ 5.7 และรูปที่ 5.8 จะเห็นได้ว่าโครงร่างย่อยของคลาส Person มีด้วยกัน 2 ระดับคือ ระดับแรก เป็นการอธิบายคลาส Person (รูป 5.7) ส่วนระดับที่ 2 เป็นการอธิบายคลาสที่เกี่ยวเนื่องกับคลาส Person (รูป 5.8) ซึ่งจะสังเกตได้ว่าในโครงร่างย่อยแต่ละระดับนั้นหากต้องการอธิบายรายละเอียดของคลาสใด คลาสนั้นจะไม่มีการแสดง Uniqueness Indentifier ในโมเดล ส่วนคลาสที่เกี่ยวเนื่องนั้นจะแสดงในโครงร่างย่อยในระดับถัด ๆ ไป

สำหรับโครงร่างย่อยทั่ว ๆ ไปนั้นอาจมีหลายระดับ ขึ้นอยู่กับความซับซ้อนของข้อมูลและการออกแบบของผู้ออกแบบ

5.2 แบบจำลองแนวคิด TOONIAM

แบบจำลองแนวคิด TOONIAM นี้ ได้นำเอา OONIAM มาเพิ่มขยายโครงสร้างให้สามารถรองรับเวลาของระบบฐานข้อมูลเชิงเวลาได้ โดยเวลาของระบบฐานข้อมูลเชิงเวลาประกอบด้วยเวลา 3 ชนิดคือ

- Transaction Time (TT) เป็นเวลาที่ข้อมูลอยู่ในฐานข้อมูลและเวลาที่ข้อมูลอาจได้รับการแก้ไข โดยเวลาแบบนี้ระบบจะจัดหาให้ ผู้ใช้ไม่ต้องป้อนเข้าไป ดังนั้นเวลาแบบนี้จะใช้กับข้อมูลที่มีการเปลี่ยนแปลงไม่บ่อยนัก หรือเปลี่ยนแปลงเมื่อมีความเข้าใจผิด เช่น ความสูงของคน ความสว่างของดวงดาว น้ำหนักของคน เป็นต้น

- Valid Time (VT) คือเวลาที่ข้อมูลเป็นจริง ซึ่งผู้ใช้งานจะต้องป้อนเวลาที่ข้อมูลนั้นเป็นจริงเข้าไปในระบบ โดยผู้ใช้งานต้องกำหนดเวลาที่ข้อมูลเป็นจริง เวลาดังกล่าวจะระบุหน่วยของเวลาด้วย เช่น ปี เดือนหรือวัน หรืออาจระบุเป็นช่วงของเวลาก็ได้

-Bitemporal Time (BT) คือการจัดเก็บทั้ง Transaction Time และ Valid Time เข้าด้วยกัน สำหรับข้อมูลที่จะใช้กับเวลาประเภทนี้คือข้อมูลที่ต้องการจัดเก็บเวลาที่ข้อมูลเป็นจริง และเวลาที่จัดเก็บข้อมูลหรือเวลาที่ข้อมูลได้รับการแก้ไข

จากนิยามของเวลาทั้งสามแบบนั้น หากผู้ใช้งานจำเป็นที่จะต้องใช้งานเวลากับข้อมูลหรือใช้งานเวลากับความสัมพันธ์ต่าง ๆ แล้ว ผู้ใช้ต้องเลือกตามความเหมาะสมเอง

สำหรับสัญลักษณ์ของเวลาที่เพิ่มเข้าไปในโมเดล TOONIAM นั้นแสดงในรูปที่ 5.9

รูปที่ 5.9 สัญลักษณ์ของเวลาที่ใช้ใน TOONIAM

จากรูปที่ 5.9 เป็นสัญลักษณ์ของเวลาที่นำมาใช้ใน TOONIAM โดยผู้ใช้สามารถเลือกใช้ประเภทของเวลาตามที่ต้องการออกแบบ

5.2.1 หน่วยของเวลา (Temporal Data type)

หน่วยของเวลาที่จะนำมาใช้ในโมเดลนี้ แบ่งออกเป็นหลาย ๆ ชนิดคือ

- Date วัน เดือน ปี
- Time เป็นเวลาที่บ่งชี้ ชั่วโมง นาที และ วินาที
- Timestamp เป็นจุดหนึ่งของเวลาประกอบด้วย วันเดือนปี ชั่วโมง นาที และวินาที
- Interval ช่วงเวลาระหว่าง 2 จุดเวลาประกอบด้วย วัน ชั่วโมง นาที และวินาที
- Timepoint นำ Timestamp มาเพิ่มเติมหน่วยของเวลาเข้าไป เพื่อต้องการให้วางเวลาเฉพาะหน่วยที่ต้องการเท่านั้น มิได้วางเวลาตาม Timestamp
- Period ช่วงเวลาระหว่าง 2 จุดเวลา ที่นำ Inteval มาเพิ่มเติมหน่วยของเวลาเข้าไป มี 4 แบบดังตารางที่ 5.1

ตารางที่ 5.1 แสดง Period 4 แบบ

Number	Type	Category
1	$[t_1, t_2]$	closed-closed
2	$[t_1, t_2)$	closed-open
3	$(t_1, t_2]$	open-closed
4	(t_1, t_2)	open-open

สำหรับในงานวิจัยนี้จะใช้แบบที่ 2 คือ $[t_1, t_2)$ หรือ closed-open เท่านั้น กล่าวคือ Period เริ่มจากเวลา t_1 และไปสิ้นสุดที่เวลา t_2 โดยเริ่มนับตรงจุดของเวลา t_1 แต่ไม่นับรวมเวลา t_2

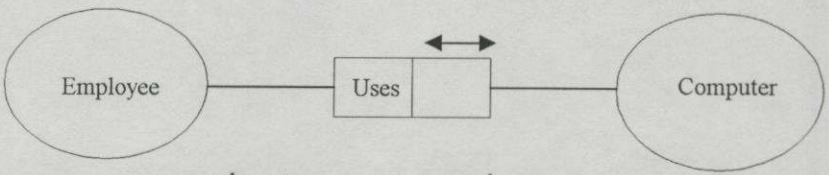
สำหรับเวลาที่ได้กล่าวมาแล้วนั้น เวลาที่อยู่ในมาตรฐานของ ODMG มีด้วยกัน 4 ประเภท คือ Date, Time , Timestamp และ Interval แต่ TimePoint และ Period ได้มีการสร้างขึ้นมาใหม่เพื่อที่จะใช้กับข้อมูลเชิงเวลาได้ ซึ่งรายละเอียดต่าง ๆ ของการสร้างเวลารูปแบบต่าง ๆ ปรากฏในหัวข้อ 5.5

5.2.2 เวลาที่เพิ่มเข้าไปในโครงร่างหลัก (Main Schema)

โครงร่างหลักนั้นเป็นการแสดงความสัมพันธ์ระหว่างคลาส ดังนั้นเวลาที่เพิ่มเข้าไปในโครงร่างหลักนี้จะเป็นเวลาของความสัมพันธ์ (Relationships with time) ซึ่งหากผู้ใช้งานโมเดลนี้ต้องการให้ความสัมพันธ์ของคลาสมีเวลาแบบใดมาเกี่ยวข้อง ผู้ใช้ก็สามารถนำสัญลักษณ์ของเวลาแบบนั้น ๆ ไปใส่ไว้ยังความสัมพันธ์ที่ต้องการ สำหรับการใช้เวลาแบบต่าง ๆ นั้นมีรายละเอียดดังต่อไปนี้

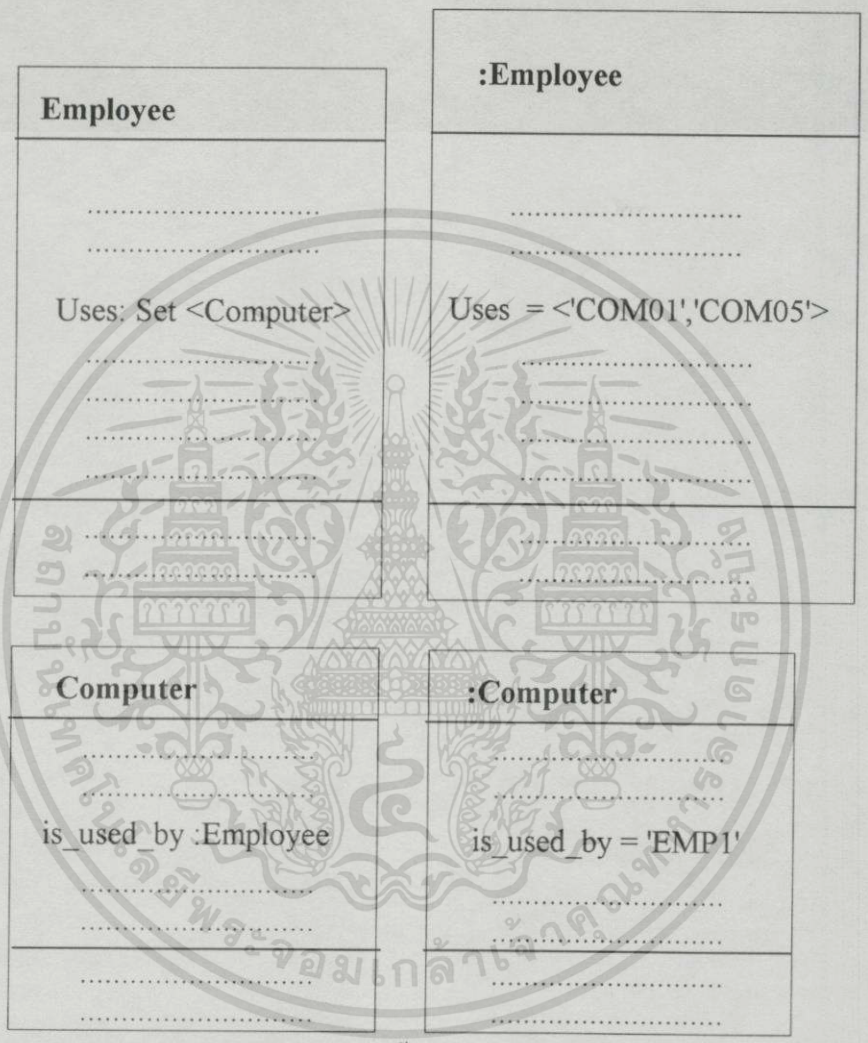
5.2.2.1 ความสัมพันธ์ไม่มีเวลามาเกี่ยวข้อง (Static Relationships)

ความสัมพันธ์นี้ก็คือความสัมพันธ์เดิม ๆ ที่ผู้ใช้งานฐานข้อมูลรู้จัก และใช้งานกันตามปกติ โดยความสัมพันธ์นี้จะไม่มีเวลามาเกี่ยวข้อง ดังตัวอย่างในรูปที่ 5.10



รูปที่ 5.10 ตัวอย่างความสัมพันธ์ที่ไม่มีส่วนของเวลามาเกี่ยวข้อง

จากรูปที่ 5.14 จะเห็นได้ว่าพนักงานมีความสัมพันธ์กับคอมพิวเตอร์ โดยพนักงานคนหนึ่งสามารถใช้งานคอมพิวเตอร์ได้หลายเครื่อง แต่คอมพิวเตอร์เครื่องหนึ่งจะมีพนักงานเพียงคนเดียวเท่านั้นที่สามารถใช้งานได้ นั่นคือความสัมพันธ์แบบหนึ่งต่อหลาย (One to many) นั่นเอง ความสัมพันธ์แบบนี้เป็นความสัมพันธ์ที่ใช้งานกันอยู่โดยทั่วไป ไม่มีส่วนใดเพิ่มเติมจากของเดิม โดยตัวอย่างของการป้อนข้อมูลมีดังรูปที่ 5.11



รูปที่ 5.11 ตัวอย่างข้อมูลความสัมพันธ์แบบไม่ขึ้นกับเวลา

จากรูปที่ 5.11 จะเห็นได้ว่าการใช้ความสัมพันธ์แบบนี้ ต้องใช้กับความสัมพันธ์ที่ไม่ต้องการจัดเก็บประวัติของความสัมพันธ์นั้นไว้ จากรูปสรุปได้ว่าพนักงานและเครื่องคอมพิวเตอร์จะมีความสัมพันธ์กันแบบไม่เกี่ยวข้องกับเวลา โดยการจัดเก็บนั้นจะจัดเก็บไว้เฉพาะข้อมูลล่าสุดที่พนักงานคนใดใช้งานเครื่องคอมพิวเตอร์เครื่องใดบ้าง หากมีการเปลี่ยนแปลงการใช้งาน ก็ต้องนำข้อมูลเดิมมาแก้ไข เพื่อให้จัดเก็บข้อมูลจริงที่พนักงานนั้น ๆ ใช้เครื่องคอมพิวเตอร์เครื่องใดบ้าง

ดังนั้นหากผู้ใช้งานต้องการตอบคำถามประเภทที่ถามประวัติการใช้งานเครื่องคอมพิวเตอร์ของพนักงาน เช่น ต้องการทราบว่าพนักงานคนใดเคยใช้งานเครื่องคอมพิวเตอร์รหัส COM01 บ้าง

ผู้ใช้งานจะใช้งานความสัมพันธ์ชนิดนี้ไม่ได้ ความสัมพันธ์ชนิดนี้จะจัดเก็บข้อมูล State ล่าสุดเท่านั้น ไม่สามารถจัดเก็บประวัติได้

5.2.2.2 ความสัมพันธ์ที่จัดเก็บเวลาที่ความสัมพันธ์นั้นอยู่ในฐานข้อมูล (Relationships with Transaction Time)

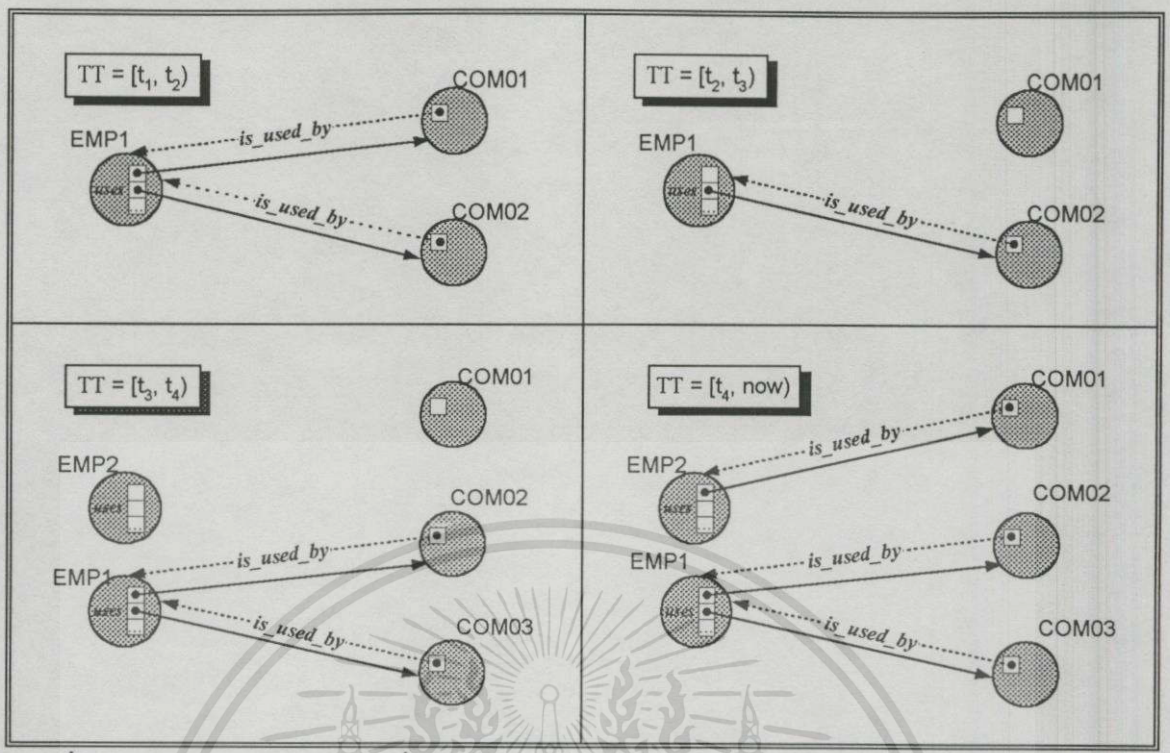
ความสัมพันธ์ประเภทนี้จะจัดเก็บประวัติของความสัมพันธ์ที่มีต่อกัน นั่นคือมิได้จัดเก็บข้อมูลเพียง State เดียวเท่านั้น โดยจะจัดเก็บความสัมพันธ์ในหลาย ๆ ช่วงเวลา ซึ่งผู้ใช้งานสามารถเรียกค้นประวัติเหล่านี้ตามช่วงเวลาที่ต้องการได้ สำหรับเวลาที่จัดเก็บนั้นระบบจะเป็นผู้จัดหาให้ กล่าวคือ ใช้เวลาของระบบนั่นเอง โดยหลักการในการเลือกใช้ความสัมพันธ์แบบนี้ ควรเลือกใช้กับความสัมพันธ์ที่มีการเปลี่ยนแปลงไม่บ่อยนัก สำหรับตัวอย่างของความสัมพันธ์แบบนี้มีดังนี้

จากรูปที่ 5.10 และ 5.11 เป็นความสัมพันธ์เพียง State เท่านั้น แต่หากมีสถานการณ์ดังตารางที่ 5.2 ผู้ใช้งานก็ต้องจัดเก็บเวลาของความสัมพันธ์ด้วย

ตารางที่ 5.2 การใช้งานคอมพิวเตอร์ในช่วงเวลาของพนักงาน

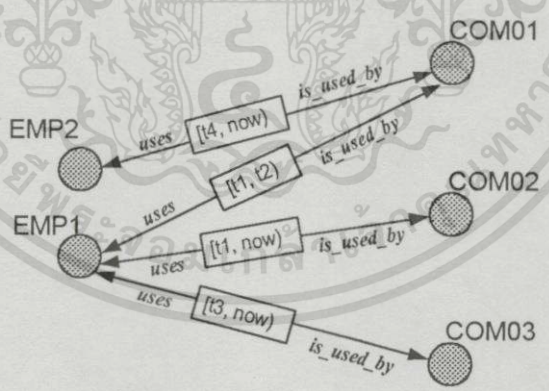
เวลา	การใช้งานคอมพิวเตอร์
t_1	EMP1 ใช้งานคอมพิวเตอร์ COM01 and COM02
t_2	EMP1 ไม่ได้ใช้งานคอมพิวเตอร์ COM01
t_3	EMP1 ใช้คอมพิวเตอร์ COM03
t_4	EMP2 ใช้คอมพิวเตอร์ COM01

จากตารางที่ 5.2 เป็นการสรุปการใช้งานคอมพิวเตอร์ของพนักงานในช่วงเวลาต่าง ๆ ซึ่งหากไม่จัดเก็บข้อมูลเวลาเอาไว้ ก็จะไม่สามารถตอบคำถามที่ต้องการประวัติของการใช้งานได้เลย จากตารางดังกล่าวสามารถสรุปเป็นแผนภาพดังรูปที่ 5.16



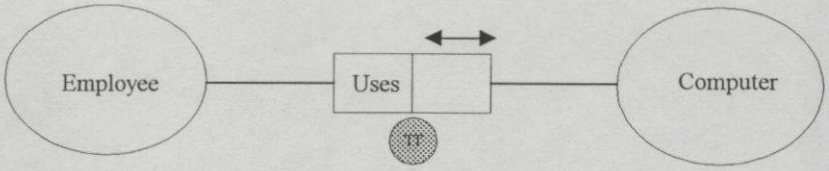
รูปที่ 5.12 แผนภาพการใช้งานเครื่องคอมพิวเตอร์ของพนักงาน

จากรูปที่ 5.12 จะเห็นได้ว่ามีการใช้งานเครื่องคอมพิวเตอร์ในหลายช่วงเวลา ซึ่งสามารถสรุปแผนภาพที่ 5.12 ให้อยู่ในรูปที่ง่ายขึ้นในรูปที่ 5.13



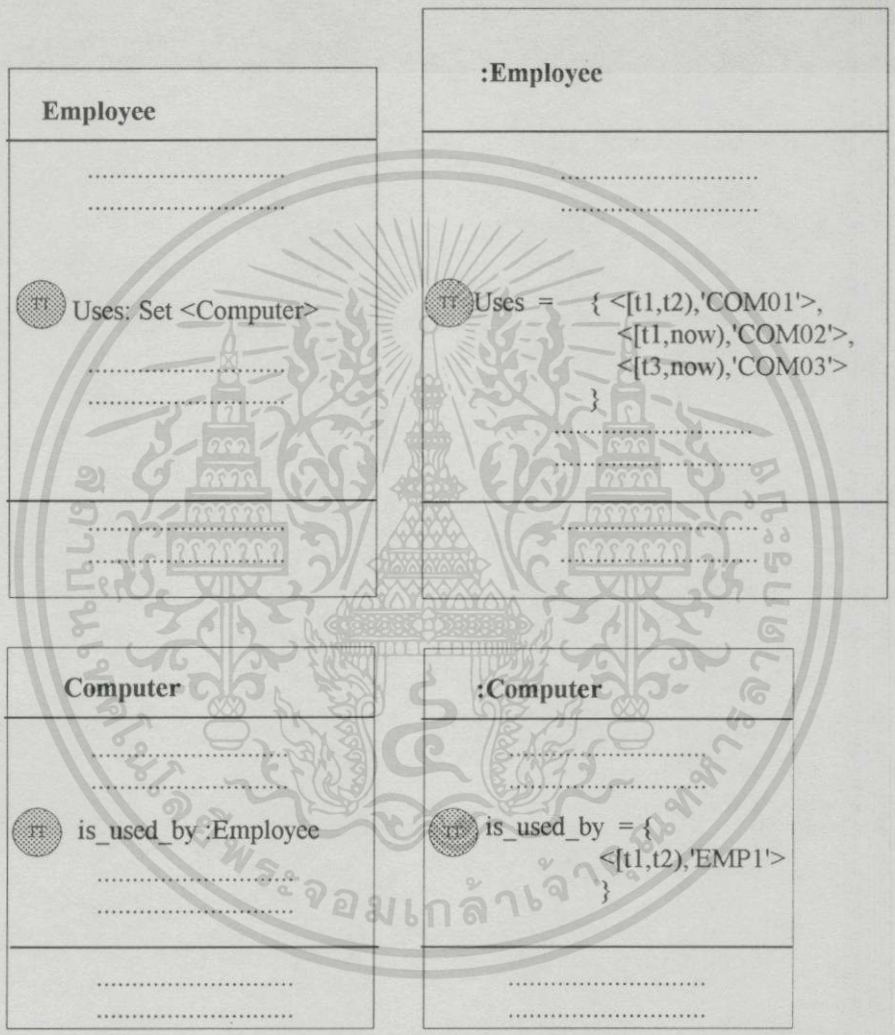
รูปที่ 5.13 สรุปการใช้งานและเวลาที่ใช้งานเครื่องคอมพิวเตอร์ของพนักงาน

จากรูปที่ 5.13 สามารถเขียนเป็น TOONIAM ได้ดังรูปที่ 5.14



รูปที่ 5.14 ความสัมพันธ์แบบมีเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล

จากรูปที่ 5.14 พนักงานและคอมพิวเตอร์มีความสัมพันธ์กัน โดยที่ความสัมพันธ์นั้นเป็นแบบหนึ่งต่อหลาย ซึ่งพนักงานคนหนึ่งสามารถใช้งานคอมพิวเตอร์ได้หลายเครื่อง แต่เครื่องคอมพิวเตอร์เครื่องหนึ่งจะมีพนักงานคนเดียวเท่านั้นที่จะใช้งานได้ ซึ่งหากผู้ใช้ต้องการให้ความสัมพันธ์ใดเป็นแบบที่มีเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล สามารถใส่สัญลักษณ์ ไว้ที่ความสัมพันธ์นั้น ดังรูปที่ 5.14 นั่นเอง สำหรับตัวอย่างในการป้อนข้อมูลนั้นแสดงในรูปที่ 5.15

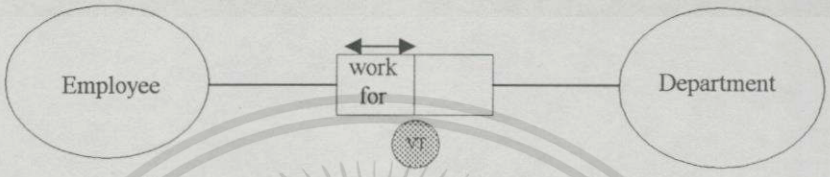


รูปที่ 5.15 ตัวอย่างการป้อนข้อมูลของความสัมพันธ์แบบมีเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล

5.2.2.3 ความสัมพันธ์แบบที่มีเวลาที่ความสัมพันธ์เป็นจริง (Relationships with Valid Time)

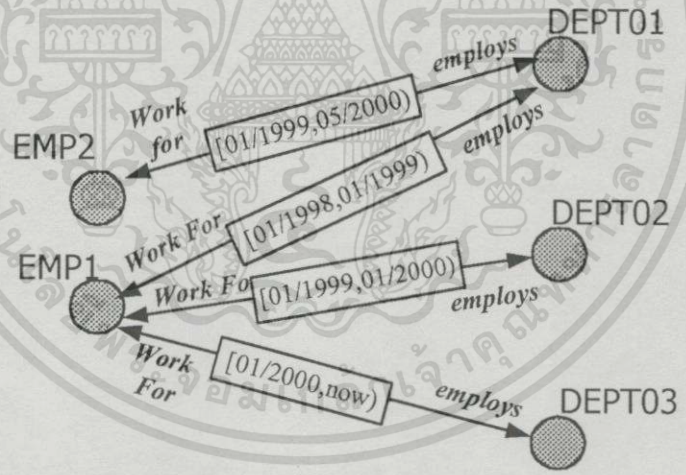
ความสัมพันธ์ประเภทนี้จะเหมือนกับความสัมพันธ์ในหัวข้อ 5.2.2.2 เพียงแต่เวลาที่จัดเก็บสำหรับความสัมพันธ์นั้น เป็นเวลาที่ผู้ใช้งานป้อนเข้าไปเอง ส่วนในหัวข้อ 5.1.4.2 นั่นเป็นเวลาที่ระบบจัดหาให้ ดังนั้นในความสัมพันธ์แบบนี้ผู้ใช้งานสามารถวางนโยบายในการเปลี่ยนแปลงเวลาได้ เช่น มีนโยบายให้พนักงานคนที่ EMP1 ไปใช้งานเครื่องคอมพิวเตอร์ COM18 ใน

อนาคต ซึ่งสามารถป้อนข้อมูลนี้ไว้ก่อนวันจริงได้ หากผู้ใช้งานต้องการให้ความสัมพันธ์ใดเป็นความสัมพันธ์แบบนี้ให้นำสัญลักษณ์ VT ไว้ที่ความสัมพันธ์นั้น สำหรับตัวอย่างในการใช้งานความสัมพันธ์นี้ จะขอยกตัวอย่างเรื่องพนักงานและแผนกของบริษัท กล่าวคือในบริษัทแต่ละบริษัทได้แบ่งออกเป็นแผนกต่าง ๆ ดังนั้นพนักงานคนหนึ่งจะสังกัดแผนกได้หนึ่งแผนกเท่านั้น แต่แผนกหนึ่งแผนกจะมีพนักงานได้หลายคน ซึ่งความสัมพันธ์ดังกล่าวสามารถแสดงเป็นตัวอย่างในรูปที่ 5.16



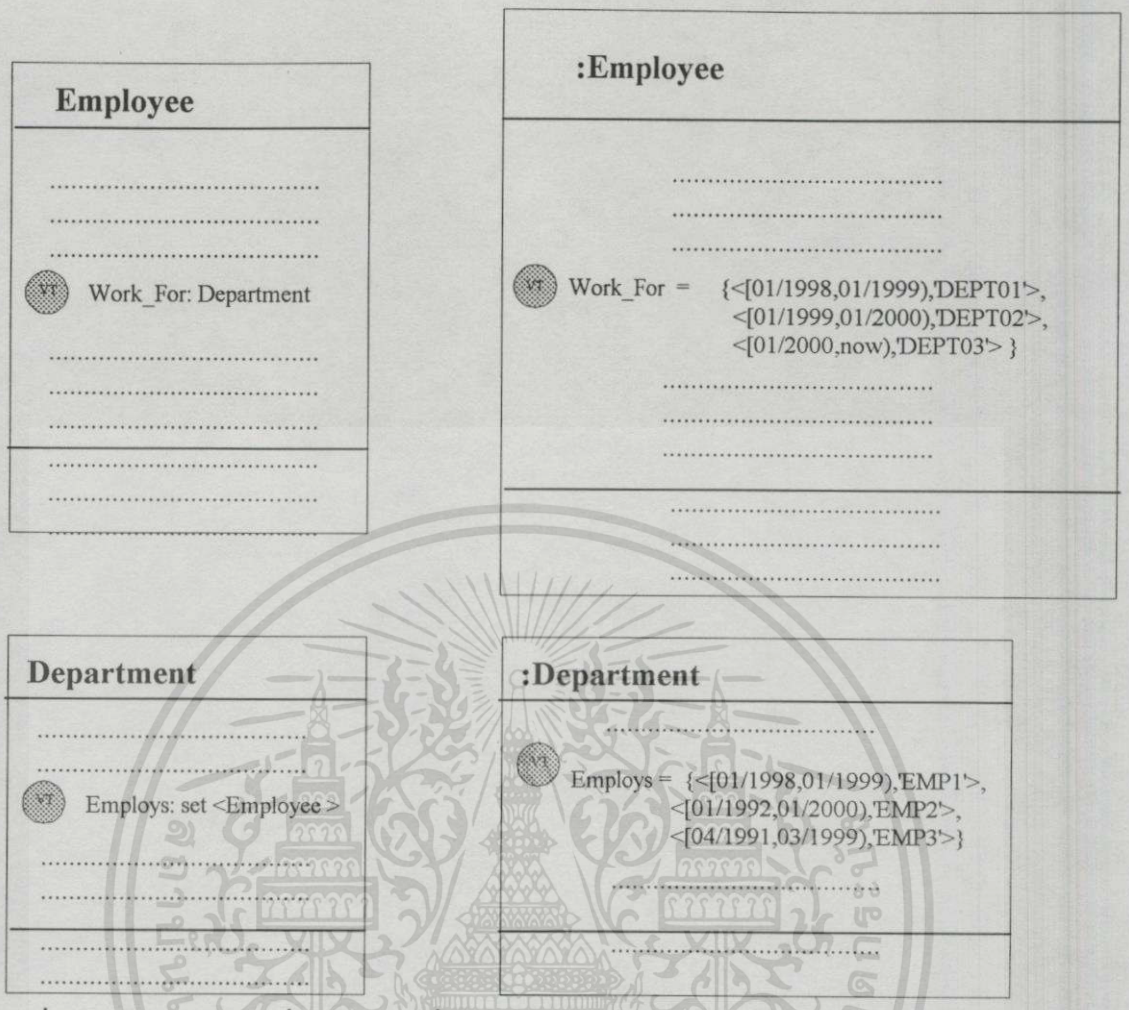
รูปที่ 5.16 ความสัมพันธ์ที่มีเวลาที่ความสัมพันธ์เป็นจริง

จากรูปที่ 5.16 จะเห็นว่าความสัมพันธ์ระหว่างพนักงานและแผนกจะมีเวลาที่ความสัมพันธ์มาเกี่ยวข้อง ซึ่งสามารถยกตัวอย่างได้ดังรูปที่ 5.17



รูปที่ 5.17 ตัวอย่างข้อมูลพนักงานและแผนกที่สังกัด

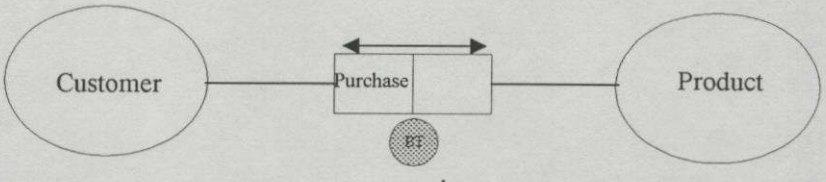
จากรูปที่ 5.16 และ 5.17 สามารถแสดงการป้อนข้อมูลได้ดังรูปที่ 5.18



รูปที่ 5.18 ตัวอย่างข้อมูลที่จัดเก็บเวลาที่ความสัมพันธ์เป็นจริง

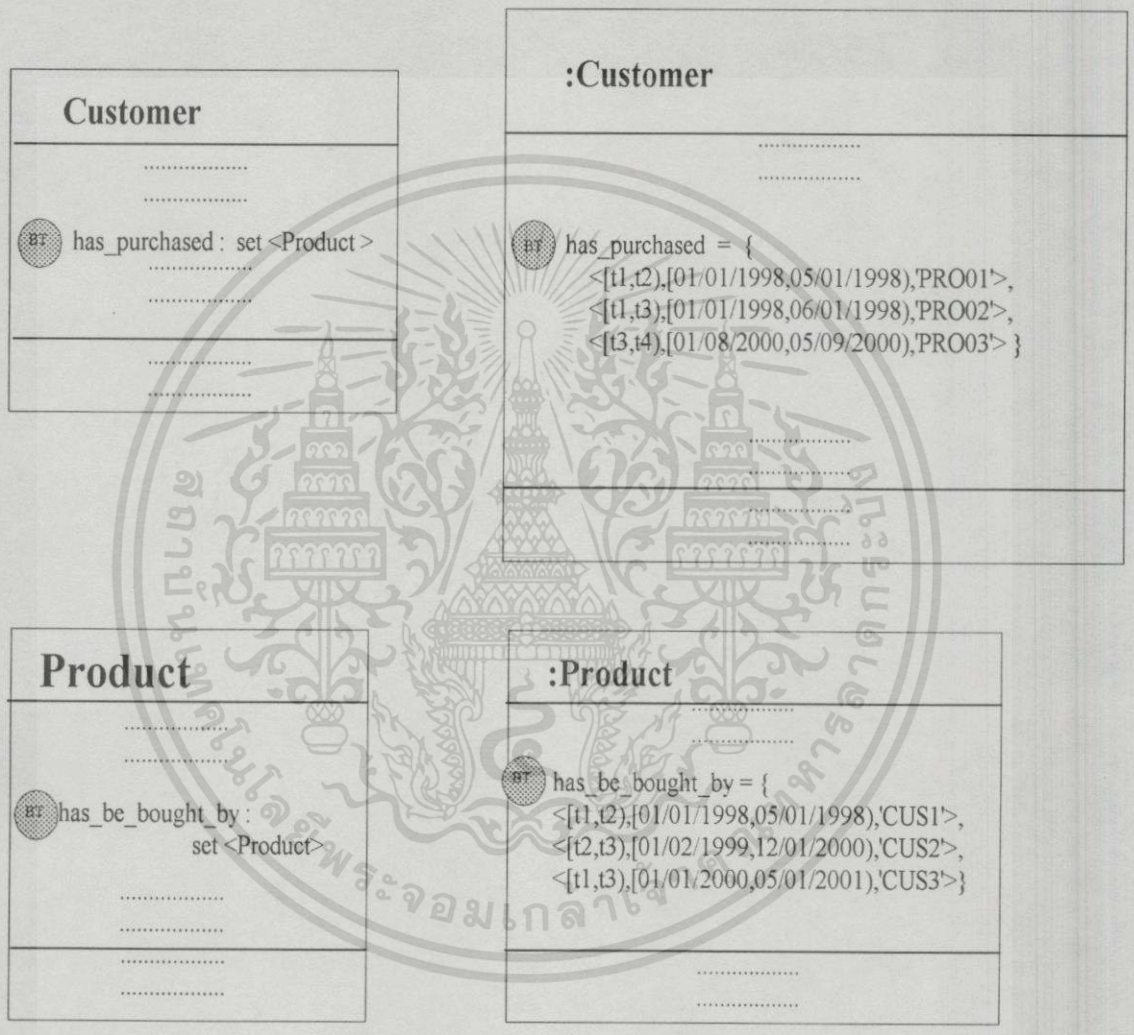
5.2.2.4 ความสัมพันธ์ที่จัดเก็บทั้งเวลาที่ความสัมพันธ์เป็นจริงและเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล (Relationships with Bitemporal Time)

ความสัมพันธ์แบบนี้จะจัดเก็บเวลาทั้งสองลงในฐานข้อมูลเพื่อจัดเก็บทั้งเวลาที่ความสัมพันธ์นั้นเป็นจริง และเวลาที่จัดเก็บหรือเลาที่ความสัมพันธ์นั้น ๆ เปลี่ยนแปลง หากผู้ใช้งานต้องการให้ความสัมพันธ์ใดเป็นความสัมพันธ์แบบนี้ให้นำสัญลักษณ์ BT ไว้ที่ความสัมพันธ์นั้น ดังตัวอย่างในเรื่องลูกค้าและสินค้าในรูปที่ 5.19



รูปที่ 5.19 ตัวอย่างความสัมพันธ์แบบจัดเก็บเวลาที่ความสัมพันธ์เป็นจริงและเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล

จากรูปที่ 5.19 เป็นความสัมพันธ์ระหว่างลูกค้าและสินค้า โดยลูกค้าคนหนึ่งสามารถซื้อสินค้าได้หลายชนิด และสินค้าหนึ่งชนิดลูกค้าสามารถเลือกซื้อได้หลายคน ความสัมพันธ์ดังกล่าวจึงเป็นความสัมพันธ์แบบหลายต่อหลาย ซึ่งในรูปได้กำหนดให้ความสัมพันธ์นี้จัดเก็บเวลาที่ความสัมพันธ์เป็นจริงและเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล โดยมีตัวอย่างในรูปที่ 5.20

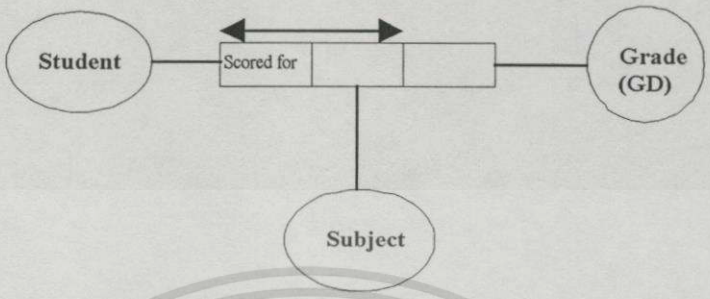


รูปที่ 5.20 ความสัมพันธ์ที่จัดเก็บเวลาที่ความสัมพันธ์เป็นจริงและเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล

จากรูปที่ 5.20 จะเห็นได้ว่ามีการจัดเก็บเวลาของความสัมพันธ์ทั้งสองลงไปด้วย ดังนั้นจึงสามารถตอบคำถามประเภทที่ต้องการทราบประวัติของความสัมพันธ์และคำถามที่ต้องการทราบช่วงเวลาในการบันทึกความสัมพันธ์ได้อีกด้วย

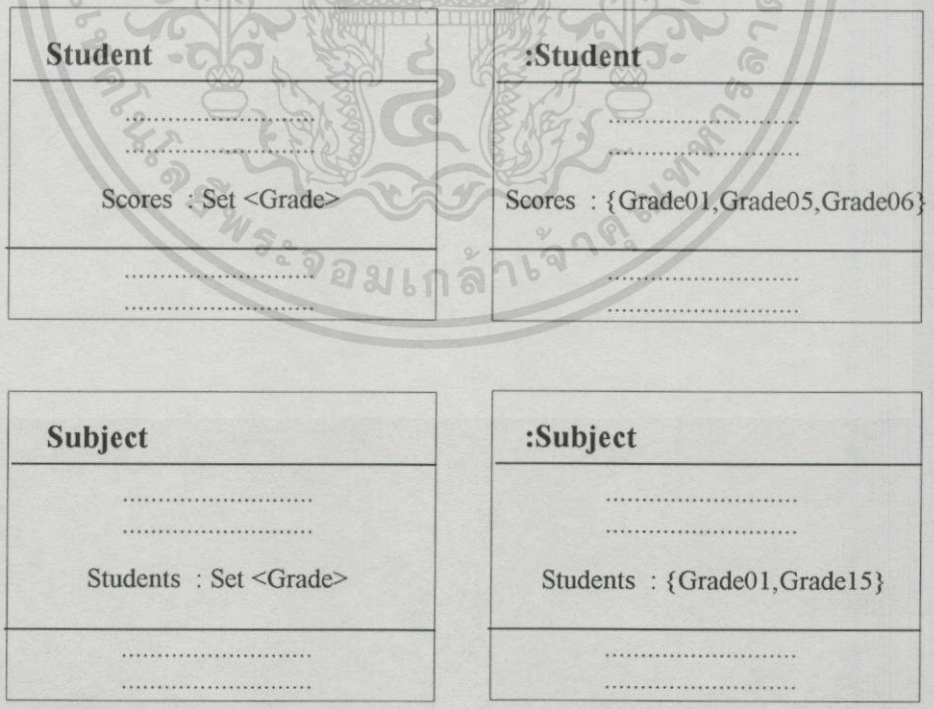
5.2.2.5 ความสัมพันธ์แบบหลายบทบาท (N-ary Relationships)

ความสัมพันธ์แบบนี้เป็นความสัมพันธ์ที่มีมากกว่า 2 บทบาท ซึ่งเมื่อแปลงให้อยู่ในรูปคลาสแล้วจะได้คลาสมากกว่า 1 คลาสเสมอดังตัวอย่างในรูปที่ 5.21

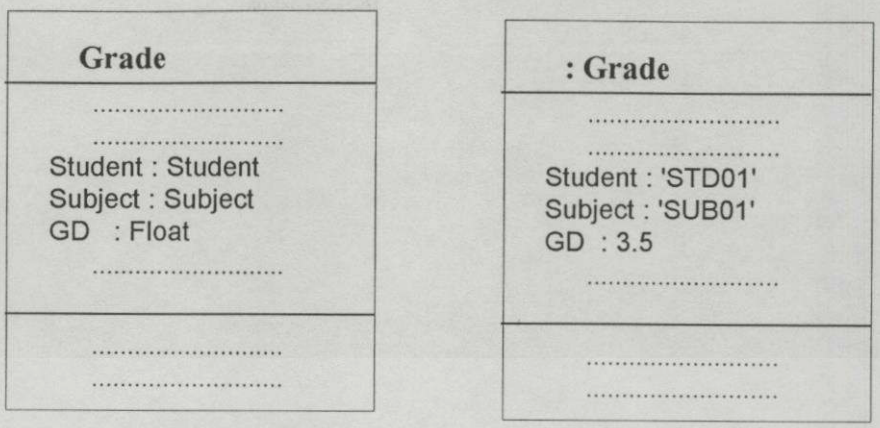


รูปที่ 5.21 ตัวอย่างของความสัมพันธ์แบบหลายบทบาท

จากรูปที่ 5.21 เป็นความสัมพันธ์ระหว่างนักเรียน วิชา และ เกรดของวิชาต่าง ๆ ที่นักเรียนได้ โดยที่ Grade นั้นมีความสัมพันธ์กับเอนติตี้นักเรียน และวิชา โดยมิได้มีความสัมพันธ์กับเอนติตี้ใดเอนติตี้หนึ่ง ต้องมีความสัมพันธ์กับทั้งสองเอนติตี้ ซึ่งในบทที่ 2 ได้อธิบายรายละเอียดในการแปลงให้เป็นตารางแล้ว สำหรับการแปลงให้เป็นคลาสนั้นก็จะเหมือนกับความสัมพันธ์ที่ในกล่าวมาแล้วในหัวข้อก่อน ๆ โดยการแปลงให้เป็นคลาสย่อย ๆ ดังรูปที่ 5.22

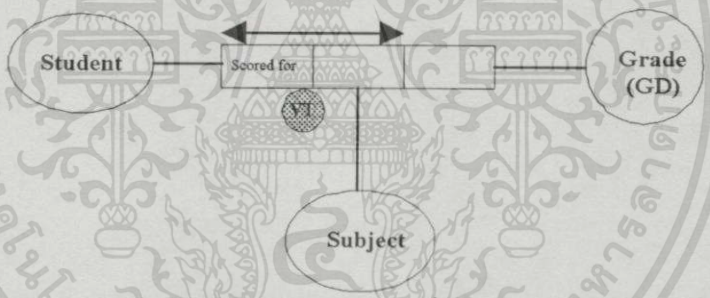


รูปที่ 5.22 ตัวอย่างข้อมูลของความสัมพันธ์แบบหลายบทบาทที่อยู่ในรูปของคลาส



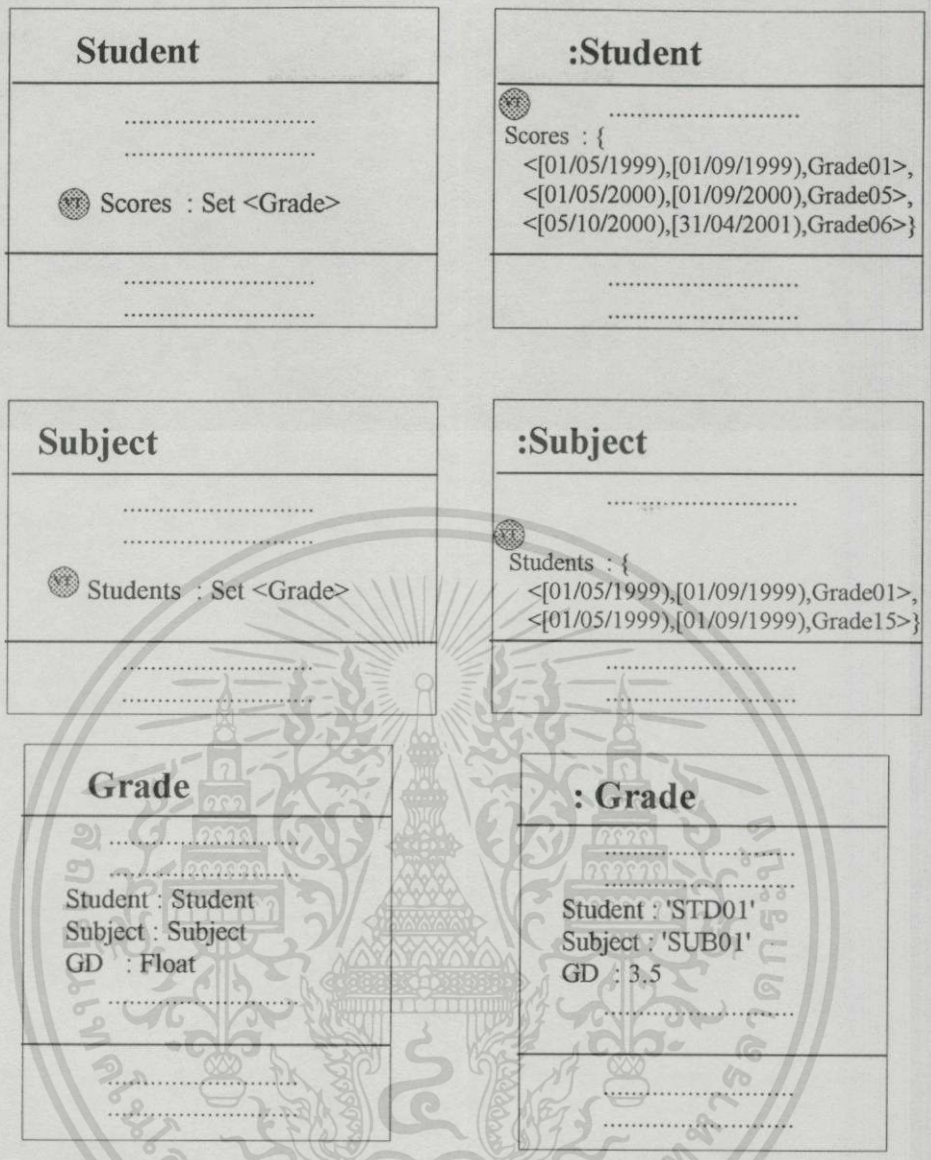
รูปที่ 5.22 ตัวอย่างข้อมูลของความสัมพันธ์แบบหลายบทบาทที่อยู่ในรูปของคลาส (ต่อ)

อนึ่งหากความสัมพันธ์แบบหลายบทบาทมีเวลาต่าง ๆ มาเกี่ยวข้องด้วย การแปลงให้อยู่ในรูปของคลาสนั้นจะใช้กฎเกณฑ์ต่าง ๆ เหมือนกับการแปลงความสัมพันธ์เกี่ยวกับเวลาที่ได้กล่าวมาก่อนหน้านี้แล้ว โดยจะเพิ่มเติมเฉพาะส่วนของความสัมพันธ์หลายบทบาทนี้เท่านั้น ดังตัวอย่างในรูปที่ 5.23



รูปที่ 5.23 ตัวอย่างของความสัมพันธ์แบบหลายบทบาทที่มี Valid Time

จากรูปที่ 5.23 เป็นการจัดเก็บประวัติของความสัมพันธ์ โดยใช้ Valid Time เพิ่มไว้ที่ความสัมพันธ์ ซึ่งจะมีตัวอย่างในการป้อนข้อมูลดังรูปที่ 5.24



รูปที่ 5.24 ตัวอย่างข้อมูลของความสัมพันธ์แบบหลายบทบาทที่มี Valid Time

จากรูปที่ 5.24 เป็นตัวอย่างข้อมูลของความสัมพันธ์แบบหลายบทบาทที่มี Valid Time โดยนำเวลา เพิ่มไว้ที่แอตทริบิวต์ที่มีความสัมพันธ์ ซึ่งจะทำให้สามารถจัดเก็บประวัติของความสัมพันธ์นั้น ๆ ได้ ซึ่งหากเพิ่มเวลาแบบอื่นเข้าไปในความสัมพันธ์แบบหลายบทบาทนี้ ก็จะใช้หลักการเดียวกัน เพียงแต่เปลี่ยนรูปแบบของเวลาเท่านั้น

5.2.3 เวลาที่เพิ่มเข้าไปในโครงร่างย่อย (Sub Schema)

ในโครงร่างย่อยนั้นเป็นการอธิบายถึงรายละเอียดของแต่ละคลาส ซึ่งประกอบด้วย แอตทริบิวต์ คลาสย่อย และเมธอดของคลาสนั้น โดยหากต้องการนำเวลาเพิ่มเข้าไปในโครงร่างย่อยนั้นสามารถเพิ่มได้ที่แอตทริบิวต์ และ คลาสย่อยเท่านั้น ไม่สามารถเพิ่มเวลาไว้ที่เมธอดได้ นั่นคือเรา

สามารถจัดเก็บประวัติของแอตทริบิวต์และประวัติของคลาสย่อยได้เท่านั้น ไม่สามารถจัดเก็บประวัติของเมธอดได้ ซึ่งมีรายละเอียดในการเพิ่มเวลาเข้าไปในโครงร่างย่อยได้ดังนี้

5.2.3.1 แอตทริบิวต์ที่ไม่มีเวลามาเกี่ยวข้อง (Static Attributes)

แอตทริบิวต์ประเภทนี้จะไม่มีรูปแบบของเวลาเข้าไปเกี่ยวข้อง ซึ่งในการใช้งานปกติของผู้ใช้งานระบบฐานข้อมูลจะพบเห็นอยู่ประจำ หรือกล่าวอีกนัยหนึ่งได้ว่า แอตทริบิวต์ประเภทนี้ก็คือแอตทริบิวต์ที่เรา ๆ ใช้งานกันอยู่ตามปกติ ไม่มีการจัดเก็บ State ต่าง ๆ มีการเปลี่ยนแปลงก็เอา State ล่าสุดมาเปลี่ยนแปลง จากนั้นก็แทนที่ด้วย State ใหม่เข้าไปเลย หรือหากต้องการลบข้อมูลของแอตทริบิวต์ประเภทนี้ก็ลบข้อมูลที่ต้องการออกไปจากฐานข้อมูลได้เลยทันที ไม่ต้องคำนึงถึงการสูญหาย หรือค่าเก่า ๆ ที่เคยเป็น สำหรับตัวอย่างการกำหนดแอตทริบิวต์คือ

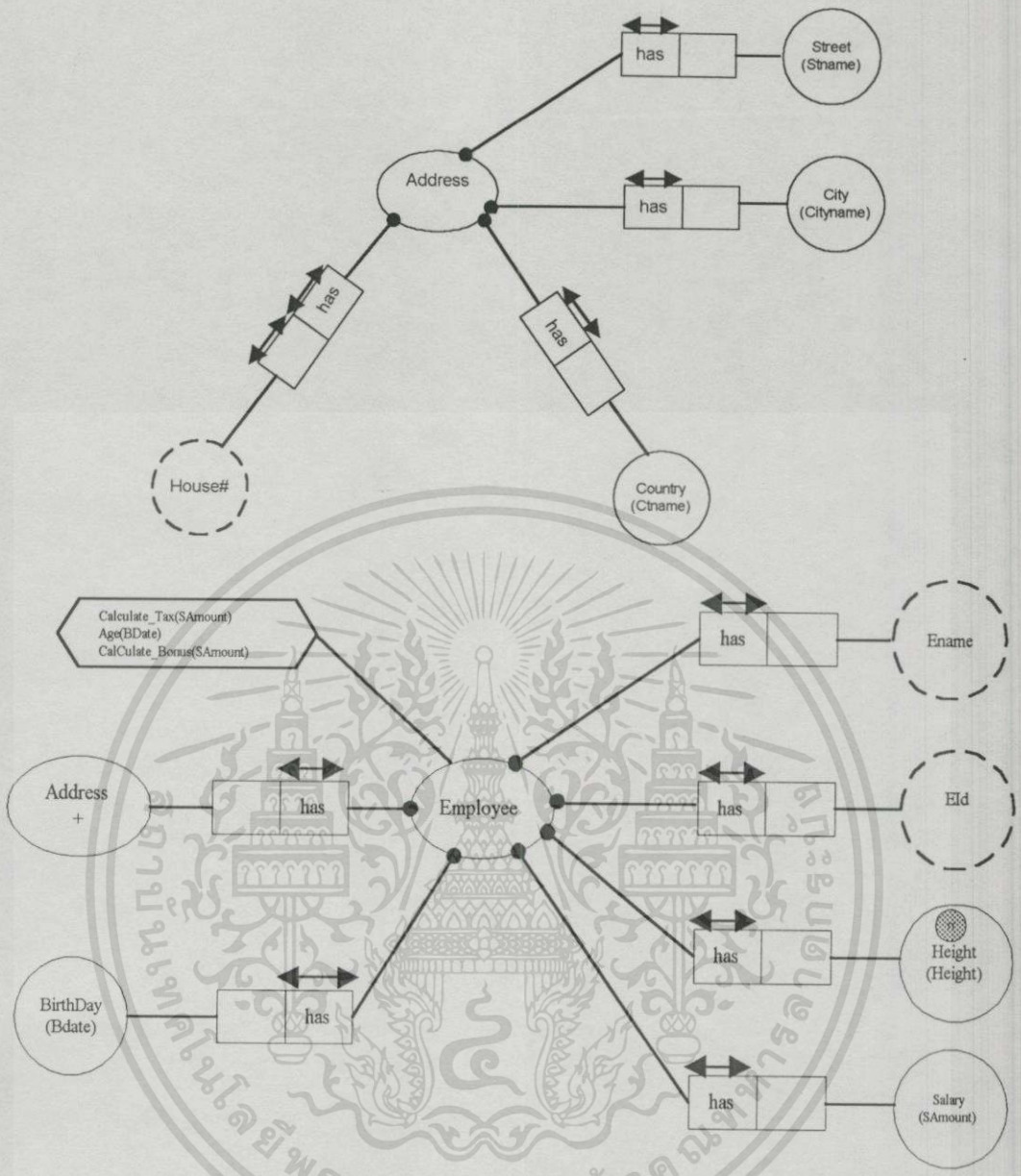
ename : string

จะเห็นได้ว่าการประกาศแอตทริบิวต์แบบนี้ก็จะประกาศเหมือนเดิมทุกประการ

หากต้องการเขียนแบบจำลองของข้อมูลเหล่านี้ ก็สามารถใช้ NIAM ปกติเขียนได้เลย ไม่ต้องเพิ่มเติมส่วนใด ๆ เข้าไปใน NIAM เลย

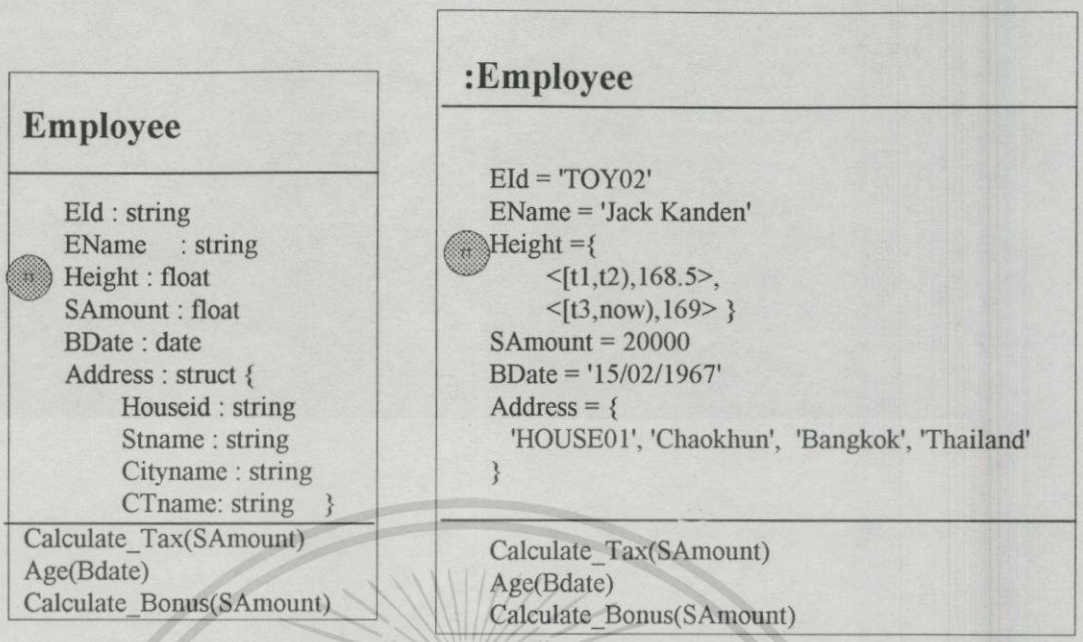
5.2.3.2 แอตทริบิวต์ที่จัดเก็บข้อมูลและเวลาที่ข้อมูลอยู่ในฐานข้อมูล (Transaction Time Attributes)

แอตทริบิวต์ประเภทนี้จะมีการจัดเก็บเวลาที่ข้อมูลอยู่ในฐานข้อมูลหรือเวลาที่ข้อมูลได้รับการเปลี่ยนแปลงไว้ด้วย ดังที่กล่าวมาแล้วนั้นว่า เวลาประเภทนี้ระบบจะจัดหาให้อย่างอัตโนมัติ ผู้ใช้งานไม่ต้องป้อนเข้าไปและไม่สามารถจะแก้ไขได้ สำหรับแอตทริบิวต์ที่เหมาะสมที่จะใช้กับเวลาประเภทนี้ก็คือ แอตทริบิวต์ที่ไม่ค่อยมีการเปลี่ยนแปลงมากนัก ซึ่งหากผู้ใช้ต้องการให้แอตทริบิวต์ใดวางเวลาประเภทนี้ไว้ ก็ให้เขียน ⊛ ไว้หน้าแอตทริบิวต์นั้น ๆ ดังตัวอย่างในรูปที่ 5.25



รูปที่ 5.25 โครงร่างย่อยของข้อมูลบุคคลที่จัดเก็บเวลาที่ข้อมูลอยู่ในฐานข้อมูลไว้ด้วย

จากรูปที่ 5.25 เป็นโครงร่างย่อยของข้อมูลบุคคล สิ่งที่เพิ่มเข้าไปก็คือการจัดเก็บเวลา Transaction Time เข้าไปที่แอตทริบิวต์ความสูง นั่นหมายความว่าระบบจะมีการจัดเก็บประวัติความสูงของบุคคลลงในฐานข้อมูล โดยเวลาที่ใช้ในการจัดเก็บนั้นคือเวลาที่ระบบจัดหาให้ ซึ่งสามารถแปลงให้เป็นรูปแบบตัวอย่างของข้อมูลดังรูปที่ 5.26

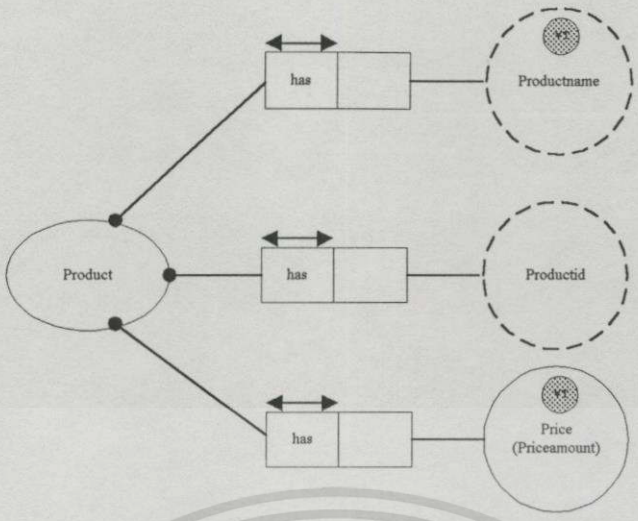


รูปที่ 5.26 ตัวอย่างของข้อมูลที่จัดเก็บเวลาที่ข้อมูลอยู่ในฐานข้อมูล

จากรูปที่ 5.26 เป็นตัวอย่างของข้อมูลที่อยู่ในรูปของคลาส จากตัวอย่างจะเห็นได้ว่า มีแอตทริบิวต์ที่จัดเก็บข้อมูลตามเวลาทรานแซ็กชั่น คือ แอตทริบิวต์ความสูง

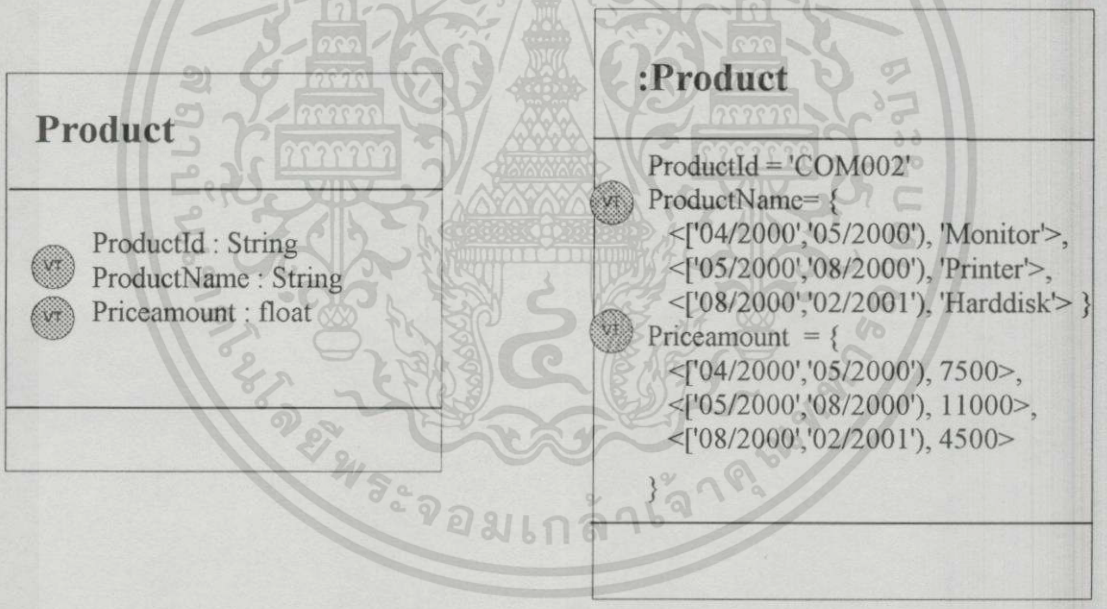
5.2.3.3 แอตทริบิวต์ที่จัดเก็บข้อมูลและเวลาที่ข้อมูลเป็นจริง (Valid Time Attributes)

แอตทริบิวต์ประเภทนี้จะเป็นแอตทริบิวต์ที่ผู้ใช้จะต้องป้อนเวลาที่ต้องการให้ข้อมูลนั้น ๆ เป็นจริงเข้าไปด้วย ซึ่งเวลาที่กล่าวถึงนี้เป็นเวลาที่ข้อมูลนั้น ๆ เป็นจริง เช่น มีนโยบายในการเปลี่ยนชื่อจากสมศรีให้เป็นสมชายในวันที่ 5 มิถุนายน 2545 นั่นคือชื่อสมชายจะเป็นจริงในวันที่ 5 มิถุนายน 2545 ดังนั้นในวันดังกล่าวจึงเป็นเวลาที่ผู้ใช้งานต้องป้อนเข้าไป หากผู้ใช้งานต้องการที่จะให้แอตทริบิวต์ใดมีลักษณะเป็นแอตทริบิวต์ประเภทนี้ ก็สามารถป้อนสัญลักษณ์ ⓥ ไว้ที่หน้าแอตทริบิวต์นั้น ตัวอย่างของแอตทริบิวต์ประเภทนี้มีในรูปที่ 5.27



รูปที่ 5.27 ตัวอย่างแอตทริบิวต์ที่จัดเก็บข้อมูลและเวลาที่ข้อมูลนั้นเป็นจริง

จากรูป 5.27 สามารถแสดงตัวอย่างของการนำไปใช้ได้ดังรูปที่ 5.28



รูปที่ 5.28 ตัวอย่างของข้อมูลที่มีการจัดเก็บข้อมูลและเวลาที่ข้อมูลเป็นจริง

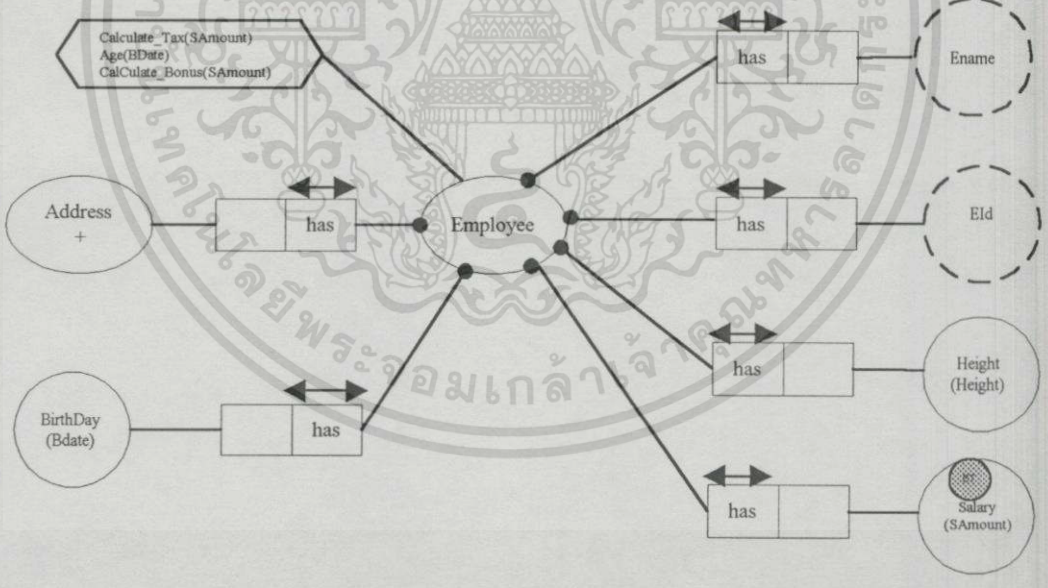
จากรูปที่ 5.28 เป็นตัวอย่างของข้อมูลที่มีการจัดเก็บข้อมูลและเวลาที่ข้อมูลเป็นจริง โดยยกตัวอย่างเรื่องของสินค้า โดยสินค้าอาจจะมีการเปลี่ยนแปลงทั้งชื่อและราคาได้ จะเห็นได้ว่าช่วงเวลาทั้งหมดมีด้วยกัน 3 ช่วง แต่ละช่วงก็มีความจริงที่แตกต่างกัน ซึ่งแต่ละช่วงของเวลาก็มีชื่อของสินค้าและราคาของสินค้ากำกับอยู่ จากตัวอย่างชื่อสินค้าและราคาของสินค้ามี 3 ช่วงเท่ากัน โดยแต่ละช่วงเวลาของทั้งสองรายการนี้สอดคล้องกัน แต่ในความเป็นจริงแล้ว รายการทั้งสองอาจจะมีช่วงเวลาที่ต้องตรงกันก็ได้ เช่น อาจเป็นไปได้ที่ราคาของเครื่องพิมพ์และจอภาพอาจมีราคาเท่ากันใน

ช่วงนั้นพอดี จึงไม่จำเป็นต้องเขียนราคาที่เท่ากันกำกับในช่วงเวลาต่าง ๆ อาจมีการรวบ 2 ช่วงเวลาเข้าด้วยกัน โดยใช้ราคาเดียวกันก็ได้

5.2.3.4 แอตทริบิวต์ที่จัดเก็บข้อมูล เวลาที่ข้อมูลเป็นจริง และเวลาที่ข้อมูลอยู่ในฐานข้อมูล (Bitemporal Time Attributes)

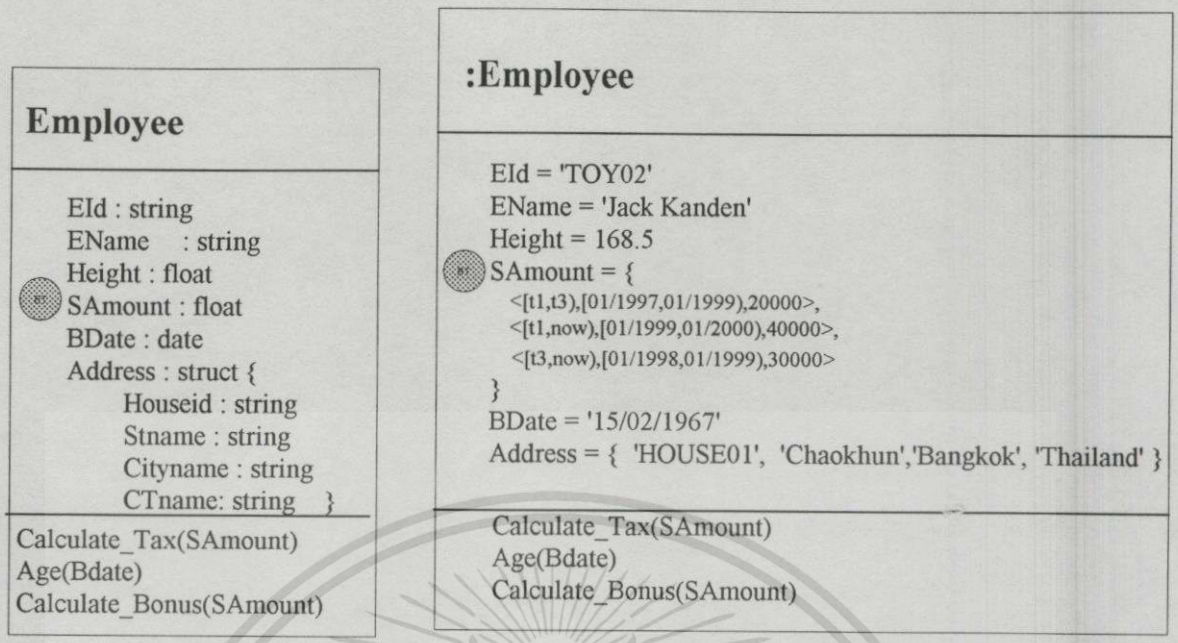
แอตทริบิวต์ประเภทนี้จะจัดเก็บเวลาทั้งสองชนิดของข้อมูลลงไปด้วย เพื่อให้การสืบค้นได้ค่าที่ถูกต้องมากขึ้น เนื่องจากการจัดเก็บเวลาที่ข้อมูลเป็นจริงนั้น อาจไม่ทราบว่าบันทึกไว้เมื่อใด และมีการเปลี่ยนแปลงข้อมูลอย่างไร จะทราบเพียงว่าความจริงนั้นเป็นจริงเมื่อใดเท่านั้น ส่วนหากจัดเก็บเฉพาะเวลาข้อมูลนั้นอยู่ในฐานข้อมูล หรือเวลาที่ข้อมูลนั้นมีการเปลี่ยนแปลงนั้น ก็จะไม่ทราบเหมือนกันว่าเวลาที่ข้อมูลนั้นเป็นจริงเมื่อใด กล่าวคือ ผู้ใช้จะไม่สามารถกำหนดนโยบายเกี่ยวกับเวลาเอง ใช้เพียงแต่เวลาที่ระบบจัดให้เท่านั้น

ดังนั้นหากแอตทริบิวต์ใดต้องการที่จะจัดเก็บเวลาทั้งสอง เพื่อประโยชน์ในการสืบค้นและประโยชน์ในการประมวลผลข้อมูลแล้ว ผู้ใช้งานจะต้องใส่สัญลักษณ์ BT ไว้หน้าแอตทริบิวต์ที่ต้องการ สำหรับตัวอย่างที่ของแอตทริบิวต์ประเภทนี้มีอยู่ในรูปที่ 5.29



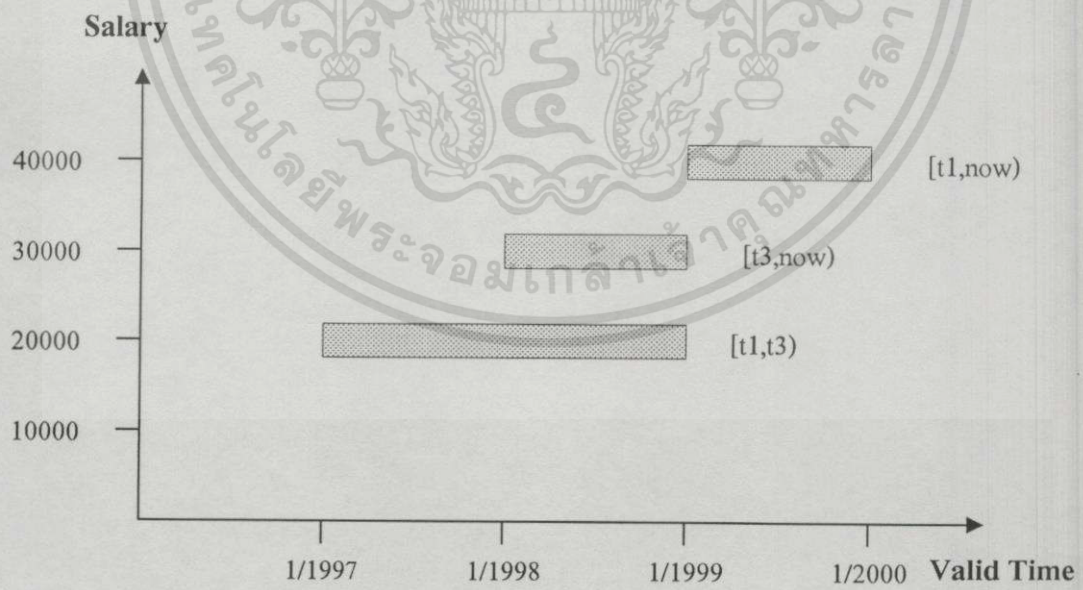
รูปที่ 5.29 ตัวอย่างของข้อมูลที่จัดเก็บเวลา Bitemporal

จากรูปที่ 5.29 มีตัวอย่างในการใช้ข้อมูลจริงดังรูปที่ 5.30



รูปที่ 5.30 ตัวอย่างข้อมูล Bitemporal Time ที่จัดเก็บทั้ง Valid Time และ Transaction Time

จากรูปที่ 5.30 เป็นตัวอย่างในการป้อนข้อมูลของแอตทริบิวต์ที่เป็นแบบ Bitemporal Time ซึ่งแอตทริบิวต์ที่มีหน้าที่ดังกล่าวคือแอตทริบิวต์เงินเดือน สำหรับเงินเดือนดังกล่าวนั้นสามารถสรุปได้ดังรูปที่ 15.31



รูปที่ 5.31 รายละเอียดของเงินเดือนจากรูปที่ 5.30

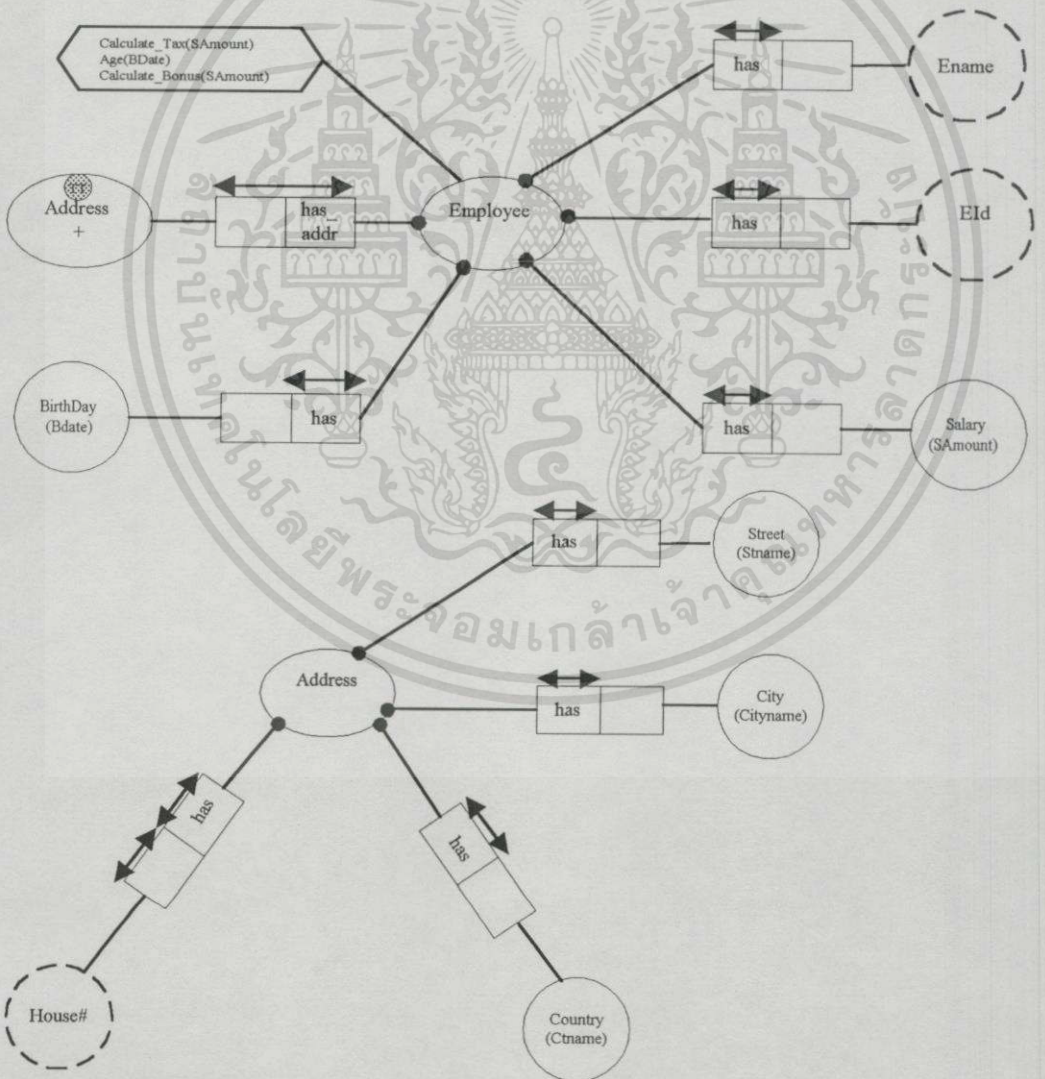
จากรูปที่ 5.31 สามารถพิจารณาได้ว่า ในการป้อนข้อมูลครั้งแรกนั้น (t1) พนักงานดังกล่าวมีเงินเดือน 20,000 บาท เมื่อเดือนมกราคม ปี 1997 ถึงเดือนธันวาคม ปี 1998 และมีเงินเดือน 40,000

บาทในเดือนมกราคม ปี 1999 ถึงเดือนธันวาคม ปี 1999 แต่เมื่อเวลา t3 ได้มีการป้อนข้อมูลใหม่ก็คือ ในเดือนมกราคม ปี 1998 ถึงเดือนธันวาคม ปี 1998 มีเงินเดือนเป็น 30,000 บาท แทนเงินเดือนเดิม

จะเห็นได้ว่าการจัดเก็บแบบ Bitemporal Time นั้นจะช่วยให้ผู้ใช้งานระบบจะทราบถึงเวลาที่ข้อมูลนั้นเป็นจริงและเวลาที่ป้อนข้อมูลหรือเวลาที่ข้อมูลนั้นเปลี่ยนแปลงด้วย

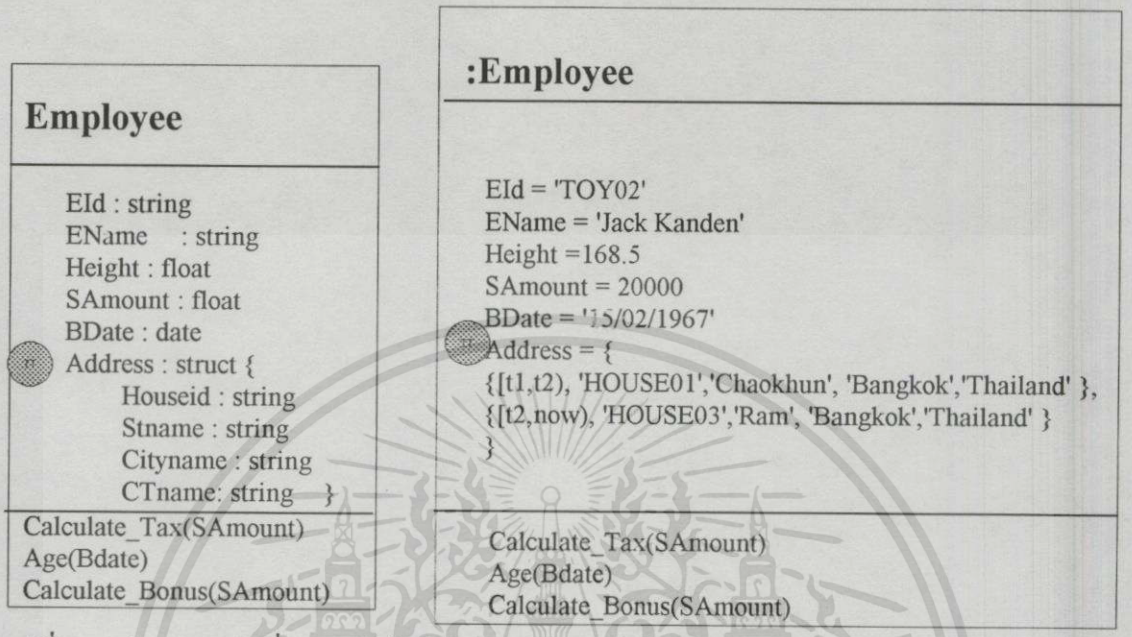
5.2.3.5 การเพิ่มเวลาไว้ที่คลาสย่อย (Embeded Class Timestamp)

การเพิ่มเวลาไว้ที่คลาสย่อยนั้น เป็นการจัดเก็บประวัติของข้อมูลของคลาสย่อย ซึ่งผู้ใช้งานสามารถเพิ่มเวลาได้ทั้ง 3 แบบ คือ TT,VT และ BT โดยลักษณะการใช้งานนั้นจะเหมือนกับการเพิ่มเวลาในหัวข้อ 5.2.3.2 ถึง 5.2.3.4 สำหรับในคลาสย่อยนี้ผู้ใช้สามารถนำสัญลักษณ์ของเวลามาเพิ่มไว้ในชื่อของคลาสย่อยที่มี Fact type กับคลาสหลัก ดังรูปที่ 5.32



รูปที่ 5.32 การเพิ่มเวลาไว้ที่คลาสย่อย

จากรูปที่ 5.32 เป็นการเพิ่มเวลา TT ไว้ที่คลาสย่อย Address นั่นคือมีการจัดเก็บประวัติของการอยู่อาศัย โดยมีตัวอย่างในการป้อนข้อมูลดังรูปที่ 5.33



รูปที่ 5.33 ตัวอย่างการเพิ่มเวลาเข้าไปในคลาสย่อย

5.2.4 สรุป

การเพิ่มสัญลักษณ์ของเวลาเข้าไปในโมเดลนั้นจำเป็นต้องมีข้อจำกัดบางประการ ซึ่งสามารถสรุปการเพิ่มเวลาเข้าไปได้ดังตารางที่ 5.3

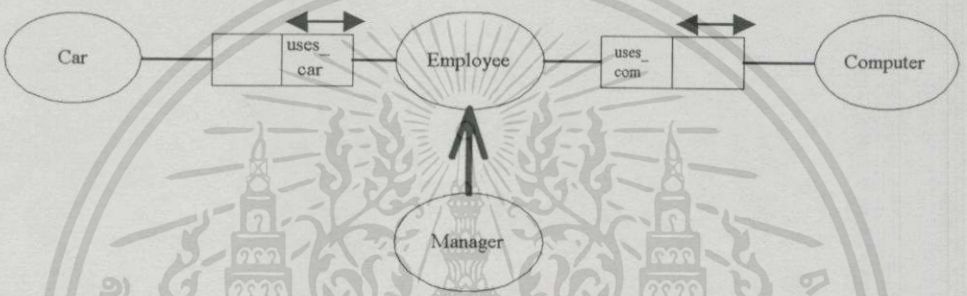
ตารางที่ 5.3 สรุปการเพิ่มเวลาเข้าไปในโมเดล TOONIAM

Schema	Attribute	Class	Relationship
Main Schema	N-ary Only	N	Y
Sub Schema	Y	EmbeddedOnly	N

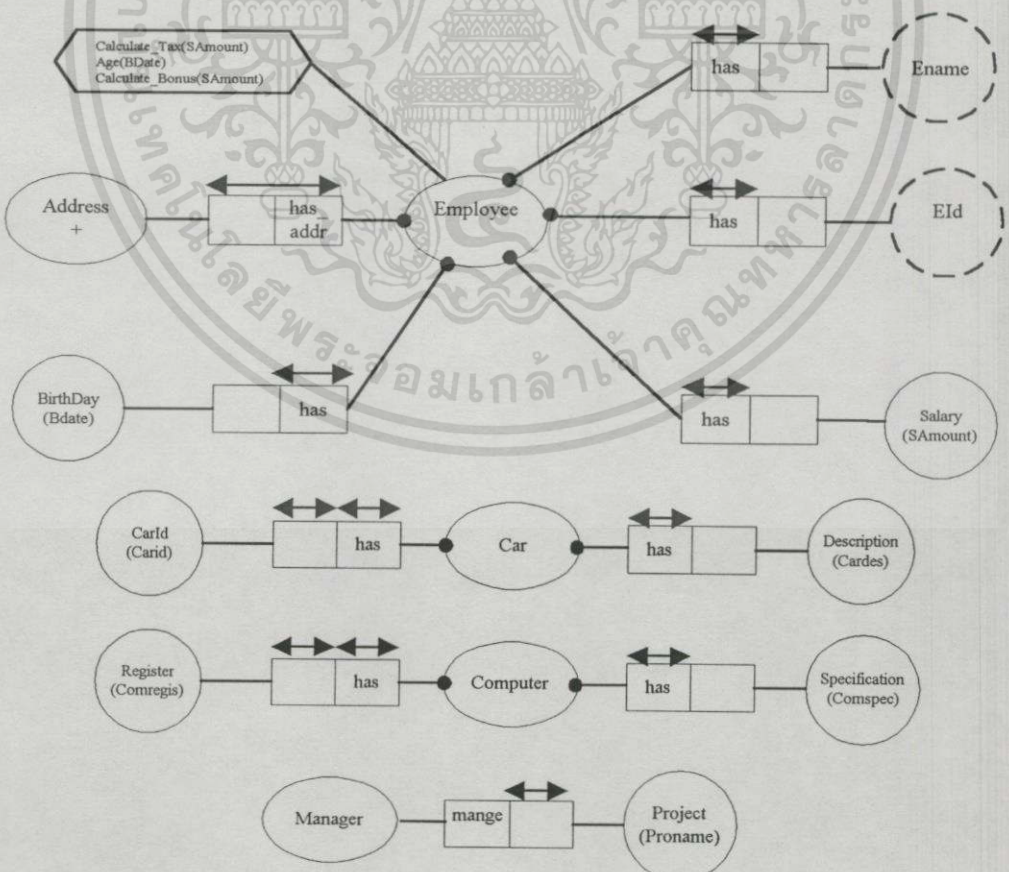
จากตารางที่ 5.3 เป็นข้อสรุปของการเพิ่มเวลาเข้าไปในโมเดล จะสังเกตได้ว่าใน โครงร่างหลักนั้นจะเพิ่มเวลาเข้าไปได้เฉพาะความสัมพันธ์ระหว่างคลาส และเพิ่มเวลาให้กับแอตทริบิวต์ของความสัมพันธ์หลายบทบาทเท่านั้น ไม่สามารถเพิ่มเวลาเข้าไปในคลาสหรือส่วนอื่น ๆ ได้เลย สำหรับในโครงร่างย่อยนั้น จะสามารถเพิ่มเวลาเข้าไปในแอตทริบิวต์และความสัมพันธ์ที่มีไปยังคลาสอ้างอิงเท่านั้น ไม่สามารถเพิ่มเวลาไปในคลาสได้

5.3 การแปลงโมเดล OONIAM ให้เป็นภาษานิยามเชิงวัตถุ ODL

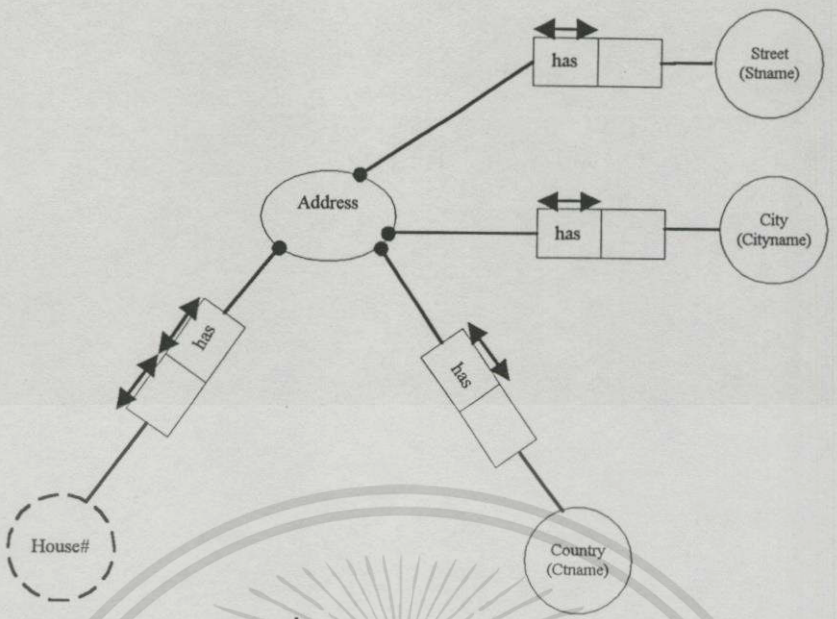
ก่อนที่จะกล่าวถึงการใช้เวลาในโมเดลนี้ จะขอกล่าวถึงการแปลงโมเดล OONIAM ซึ่งเป็นโมเดล NIAM เชิงวัตถุให้เป็นภาษานิยามเชิงวัตถุ ODL ก่อน ซึ่งหลังจากได้มีการแปลงขั้นตอนนี้แล้ว เมื่อโมเดลมีเวลาใน TOONIAM มาเกี่ยวข้อง สิ่งที่แปลงไว้ก่อนนี้จะช่วยให้การแปลงจาก TOONIAM ให้เป็น ODL จะง่ายขึ้น สำหรับขั้นตอนในการแปลงนั้นเริ่มต้นต้องแปลงจากโครงร่างย่อยก่อน จากนั้นจึงไปแปลงในโครงร่างหลัก เนื่องจากในโครงร่างย่อยได้อธิบายโครงสร้างของคลาสไว้อย่างละเอียด ดังนั้นเราจำเป็นต้องสร้างคลาสต่าง ๆ ก่อน จึงจะนำโครงร่างหลักมาใช้งานร่วมเพื่ออธิบายถึงความสัมพันธ์ระหว่างคลาส ซึ่งการแปลง OONIAM ให้เป็น ODL ดังที่กล่าวมาแล้วนั้นแสดงตัวอย่างในรูปที่ 5.34 ,5.35 และ 5.36



รูปที่ 5.34 ตัวอย่างโครงร่างหลักของระบบ



รูปที่ 5.35 ตัวอย่างโครงร่างย่อยระดับที่ 1



รูปที่ 5.36 ตัวอย่างโครงร่างย่อยระดับที่ 2

จากรูปที่ 5.34 , 5.35 และ 5.36 สามารถแปลงเป็นภาษานิยามเชิงวัตถุได้ดังนี้

```
interface Employee
(           extent Employees )
{
            typedef  Struct  AddressStruct {
                        string Houseid ,
                        string Ctname,
                        string Cityname,
                        string Stname ;}

            attribute string EId ;
            attribute string Ename ;
            attribute float Salary ;
            attribute date Bdate ;

            attribute  set<AddressStruct>  Address ;

            relationship  Car uses_car  inverse Car :: is_used_by ;
            relationship set<Computer> uses_com inverse
                                          Computer :: is_used_by ;

            float Calulate_Tax(in float SAmount) ;
            float Age(in float BDate) ;
            float Calulate_Bonus(in float SAmount)
};


interface Manager : Employee
(           extent Managers)
{
            attribute string ProName ;
};
```

```
interface Computer
(       extent Computers )
{
        attribute string Comregis ;
        attribute string Comspec ;

        relationship Employee is_used_by  inverse
                                          Employee :: uses_com ;
};

interface Car
(       extent Cars )
{
        attribute string Carid ;
        attribute string Cardes ;

        relationship set < Employee >  is_used_by  inverse
                                          Employee :: uses_car ;
};
```

สำหรับหลักการในการแปลง OONIAM ให้เป็นภาษา ODL มีหลักการดังนี้

5.3.1 การแปลงโครงร่างย่อย (Sub Schema)

โครงร่างย่อยเป็นการอธิบายรายละเอียดของแต่ละคลาส ซึ่งจะอธิบายแอตทริบิวต์ คลาสย่อยและคลาสอ้างอิง โดยจะต้องแยกเป็นกรณี ๆ ไป ซึ่งคลาสทั้งหมดได้ถูกสร้างขึ้นแล้วในขั้นตอนการแปลงโครงร่างหลัก ดังนั้นในโครงร่างย่อยนี้เป็นส่วนของการเพิ่มรายละเอียดของแอตทริบิวต์ สร้างคลาสย่อยและสร้างคลาสอ้างอิงเท่านั้น สำหรับอัลกอริธึมในการแปลงโครงร่างย่อยมีดังนี้

นำโครงร่างย่อยมาสร้างเป็นคลาสใหม่จัดเก็บลงใน Meta Class โดย

1. สร้างแอตทริบิวต์ของคลาส โดยใช้ Uniqueness Identifier เป็นชื่อแอตทริบิวต์ โดยสามารถแบ่งลักษณะของชนิดความจริง (Fact Type) ที่คลาสหลักมีกับแอตทริบิวต์นั้น ได้ดังนี้

   1.1 ชนิดความจริงแบบหนึ่งต่อหนึ่ง (One to one) แอตทริบิวต์นั้นเป็น Simple

   1.2 ชนิดความจริงแบบหนึ่งต่อหลาย (One to many) แอตทริบิวต์นั้นเป็น Collection

   1.3 ชนิดความจริงแบบหลายต่อหนึ่ง (Many to one) แอตทริบิวต์นั้นเป็น Simple

   1.4 ชนิดความจริงแบบหลายต่อหลาย (Many to many) แอตทริบิวต์นั้นเป็น Collection

2. หากคลาสใดมีคลาสย่อย นำคลาสที่สร้างไว้ใน Meta Class มาสร้างเป็นคลาสย่อยของคลาสนั้น โดยสามารถแบ่งลักษณะของชนิดความจริง (Fact Type) ที่คลาสหลักมีกับคลาสย่อยนั้นได้ดังนี้

   2.1 ชนิดความจริงแบบหนึ่งต่อหนึ่ง (One to one) คลาสย่อยนั้นเป็น Simple

   2.2 ชนิดความจริงแบบหนึ่งต่อหลาย (One to many) คลาสย่อยนั้นเป็น Collection

   2.3 ชนิดความจริงแบบหลายต่อหนึ่ง (Many to one) คลาสย่อยนั้นเป็น Simple

   2.4 ชนิดความจริงแบบหลายต่อหลาย (Many to many) คลาสย่อยนั้นเป็น Collection

5.3.2 การแปลงโครงร่างหลัก (Main Schema)

โครงร่างหลักนี้ได้อธิบายถึงความสัมพันธ์ในแต่ละคลาส รวมถึงการถ่ายทอดคุณสมบัติของคลาสอีกด้วย นอกจากนี้ยังอธิบายถึงความสัมพันธ์แบบ N-ary ด้วย สำหรับอัลกอริธึมในการแปลงโครงร่างหลักมีดังนี้

1. กำหนดคุณสมบัติให้กับคลาสที่มีการถ่ายทอดคุณสมบัติ
2. กำหนดความสัมพันธ์ระหว่างคลาส ดังนี้

◆ ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (One to one) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Reference ทั้งสองคลาส

◆ ความสัมพันธ์แบบหนึ่งต่อหลาย (One to many) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Collection ของ Reference และ Reference ตามลำดับ

◆ ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (Many to one) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Reference และ Collection ของ Reference ตามลำดับ

◆ ความสัมพันธ์แบบหลายต่อหลาย (Many to many) กำหนดให้ทั้งสองคลาสมีความสัมพันธ์กันแบบ Collection ของ Reference และ Collection ของ Reference ตามลำดับ

3. หากมีความสัมพันธ์แบบหลายบทบาท ให้สร้างคลาสใหม่เพื่อจัดเก็บ Simple Object ที่เกี่ยวข้องกับ N-ary โดยสร้างความสัมพันธ์หนึ่งต่อหนึ่งไปยังคลาส Complex และกำหนดให้คลาส Complex มีความสัมพันธ์แบบหนึ่งต่อหลายไปยังคลาสที่สร้างใหม่

5.4 การแปลงโมเดล TOONIAM ให้เป็นภาษานิยามเชิงวัตถุอิงเวลา TODL

TODL (Temporal Object Definition Language) เป็นภาษานิยามเชิงเวลานี้ได้สร้างขึ้นเพื่อเพิ่มคุณสมบัติเชิงเวลาเข้าไปในภาษานิยามเชิงวัตถุ ODL ซึ่งภาษา ODL นี้เป็นภาษามาตรฐานบนฐานข้อมูลเชิงวัตถุ ที่สามารถทำงานบนฐานข้อมูลเชิงวัตถุได้อยู่แล้ว ซึ่งภาษา ODL นี้ได้กล่าวมาแล้วบางส่วนในบทที่ 4 แต่จะนำบางส่วนที่สำคัญมากล่าวอีกครั้งหนึ่งในบทนี้

ภาษา TODL นี้ ได้มีการเพิ่มเติมส่วนของเวลาเข้าไปในภาษา ODL ทำให้ไม่สามารถทำงานได้โดยตรงบนฐานข้อมูลเชิงวัตถุ จะต้องมีการแปลงให้เป็นภาษา ODL เสียก่อน จึงจะสามารถนำ TODL ไปสร้างอ๊อบเจ็คบนฐานข้อมูลเชิงวัตถุได้ โดยมีรายละเอียดดังต่อไปนี้

สำหรับเวลาที่ใช้ใน TODL ประกอบด้วย

- Date วัน เดือน ปี
- Time เป็นเวลาที่บ่งชี้ ชั่วโมง นาที และ วินาที
- Timestamp เป็นจุดหนึ่งของเวลาประกอบด้วย วันเดือนปี ชั่วโมง นาที และวินาที

- Interval — ช่วงเวลาระหว่าง 2 จุดเวลาประกอบด้วย วัน ชั่วโมง นาที และวินาที
- Timepoint — นำ Timestamp มาเพิ่มเติมหน่วยของเวลาเข้าไป เพื่อต้องการให้วางเวลาเฉพาะหน่วยที่ต้องการเท่านั้น มิได้วางเวลาตาม Timestamp
- Period — ช่วงเวลาระหว่าง 2 จุดเวลา ที่นำ Inteval มาเพิ่มเติมหน่วยของเวลาเข้า

สำหรับเวลาที่กล่าวมาทั้งหมดนั้น ODMG จะสนับสนุนเฉพาะ 4 แบบแรก ส่วน 2 แบบหลังได้มีการสร้างขึ้นใหม่ เพื่อให้เหมาะสมสำหรับการทำงานในส่วนของการจัดเก็บเวลา สำหรับการสร้างเวลาเหล่านี้จะมีรายละเอียดในหัวข้อ 5.5

โดยตัวอย่างของภาษา TODL มีดังนี้

```
interface Employee
(       extent Employees )
{
        typedef  Struct  AddressStruct {
                        string Houseid ,
                        string Ctname,
                        string Cityname,
                        string Stname ;}

        tt_attribute string EId ;
        vt_attribute<P,DAY> string Ename ;
        vt_attribute<P,MONTH> float Salary ;
        attribute date Bdate ;

        tt_attribute  set<AddressStruct>  Address ;

        vt_relationship<P,MONTH>Car uses_car inverse
                                        Car :: is_used_by ;
        relationship set<Computer> uses_com inverse
                                        Computer :: is_used_by ;

        float Calulate_Tax(in float SAmount) ;
        float Age(in float BDate) ;
        float Calulate_Bonus(in float SAmount)
};


interface Manager : Employee
(       extent Managers)
{
        vt_attribute <P,YEAR> string ProName ;
};
```

```
interface Computer
(       extent Computers )
{
        tt_attribute string Comregis ;
        vt_attribute<P,YEAR> string Comspec ;

        relationship Employee is_used_by  inverse
                                          Employee :: uses_com ;
};
```

ตัวอย่างด้านบนเป็นตัวอย่างของภาษา TODL ซึ่งจะเพิ่มเวลาเข้าไปในภาษา ODL จากตัวอย่างจะสังเกตได้ว่า มีการเพิ่มเวลาเข้าไป 2 แห่งคือ

- แอตทริบิวต์ หากแอตทริบิวต์ใดเพิ่มเวลาเข้าไป แอตทริบิวต์นั้นจะเขียนสัญลักษณ์ของเวลาไว้ที่หน้าคำสั่ง Attribute เช่น vt_attribute หมายถึงแอตทริบิวต์นั้นเพิ่ม Valid time เข้าไป หากเป็น tt_attribute หมายถึงแอตทริบิวต์นั้นเพิ่ม Transaction time เข้าไป หรือหากเป็น bt_attribute หมายถึงแอตทริบิวต์นั้นเพิ่ม Bitemporal time เข้าไป แต่มีข้อแม้อยู่ประการหนึ่งก็คือ เวลาที่เพิ่มเข้าไปนี้จะต้องบอกประเภทและหน่วยของเวลาที่เพิ่มเข้าไปด้วย

จากตัวอย่าง ในคลาส Employee มีคำสั่ง vt_attribute<P,DAY> string Position ; หมายความว่าแอตทริบิวต์ตำแหน่งของพนักงานนี้เพิ่ม Valid Time เข้าไป โดย Valid Time นี้เป็นช่วงของเวลาและมีหน่วยเป็นวันเช่น [01/02/1999,05/08/2000) เป็นต้น สำหรับสัญลักษณ์ P นั่นหมายถึงช่วงเวลา (Period) ในวิทยานิพนธ์นี้การกำหนดชนิดของเวลามีด้วยกัน 2 ชนิดคือ P หมายถึงช่วงเวลา (Period) และ T หมายถึงจุดของเวลา (Timepoint) สำหรับการเพิ่ม Transaction Time เข้าไปที่แอตทริบิวต์นั้นไม่ต้องบอกชนิดและหน่วยของเวลา เพราะจะเอาเวลาของระบบเพิ่มเข้าไปให้อัตโนมัติ ดังนั้นผู้ใช้ก็ไม่จำเป็นต้องป้อนเวลาเข้าไป ส่วนการเพิ่ม Bitemporal Time เข้าไปนั้นผู้ใช้ต้องระบุชนิดและหน่วยของเวลาในส่วนของ Valid Time ให้กับระบบด้วย

- ความสัมพันธ์ (Relationships) การเพิ่มเวลาไว้ที่ความสัมพันธ์ของคลาสนั้นทำได้เหมือนกับการเพิ่มเวลาในแอตทริบิวต์ กล่าวคือสามารถเพิ่มสัญลักษณ์ของเวลา vt,tt และ bt ไว้หน้าคำสั่ง relationship โดยหลักการการใส่ชนิดและหน่วยของเวลานั้นจะเป็นหลักการเดียวกับการเพิ่มเวลาไว้ที่แอตทริบิวต์

สำหรับหลักการแปลง TOONIAM ให้เป็นภาษานิยามเชิงวัตถุอิงเวลา TODL นั้น ต้องทำกระบวนการในหัวข้อ 5.3 ก่อน ซึ่งจะได้ ODL ของคลาสทั้งหมด จากนั้นนำคลาสทั้งหมดนี้มาเพิ่มส่วนของเวลาเข้าไป โดยกลับไปพิจารณาโมเดลที่สร้างขึ้นอีกครั้ง ซึ่งมีหลักเกณฑ์ดังนี้

5.4.1 โครงร่างย่อย

โครงร่างย่อยอธิบายถึงรายละเอียดของคลาสแต่ละคลาส ซึ่งประกอบด้วยแอตทริบิวต์และคลาสย่อยของคลาสนั้น ดังที่ได้กล่าวมาแล้ว การนำเวลาเพิ่มไว้ในโครงร่างย่อยสามารถเพิ่มได้ที่แอตทริบิวต์เท่านั้น ดังนั้นเมื่อนำเวลามาเพิ่มที่แอตทริบิวต์ก็จำเป็นต้องให้เพิ่มสัญลักษณ์ของเวลาเข้าไปที่คำสั่ง attribute ซึ่งการเพิ่มเวลาไว้ที่แอตทริบิวต์สามารถแบ่งรายละเอียดได้ดังนี้

5.4.1.1 แอตทริบิวต์ที่ไม่มีเวลาเข้ามาเกี่ยวข้อง (Static Attributes)

สำหรับการประกาศแอตทริบิวต์แบบนี้จะเหมือนกับการประกาศในภาษา ODL ปกติ คือไม่มีส่วนที่เพิ่มขยายเข้าไป เช่น

```
attribute String Name;
attribute enum gender {male,female};
```

จากตัวอย่างจะเห็นได้ว่าไม่มีรูปแบบใด ๆ เพิ่มขยายเข้าไปเลย ดังนั้นแอตทริบิวต์นี้จึงใช้ ODL เดิมมาสร้างได้เลย

5.4.1.2 แอตทริบิวต์ที่จัดเก็บข้อมูลและเวลาที่ข้อมูลอยู่ในฐานข้อมูล (Transaction Time Attribute)

แอตทริบิวต์นี้จะมีสัญลักษณ์  อยู่หน้าแอตทริบิวต์นั้น ดังรูปที่ 5.25 ซึ่งสามารถแปลงเป็น TODL โดยมีคำว่า tt_attribute แทน attribute ดังนี้

```
interface Employee
( extent Employees )
{
        attribute string EId ;
        attribute string EName ;
        attribute char Sex ;
        tt_attribute float  Height ;
        attribute float Salary ;
        attribute date Bdate ;
        attribute struct AddressStruct {
                        attribute string Houseid ;
                        attribute string Ctname ;
                        attribute string Cityname ;
                        attribute string Stname ;
                }Address ;

        float Calulate_Tax(in float SAmount) ;
        short Age(in date BDate) ;
        float Calculate_Bonus(in float SAmount) ;
} ;
```

จาก TODL ด้านบนจะเห็นได้ว่าเป็นการประกาศให้มีการนำเวลาที่แอตทริบิวต์นั้นอยู่ในฐานข้อมูลจัดเก็บไว้ด้วย โดยใช้ tt_attribute ซึ่งใน type จะเป็นการบอกให้มีการวางเวลาไว้ที่ แอตทริบิวต์นั้น ๆ โดยเวลาที่วางไว้นั้นเป็นแบบ Period จึงไม่ต้องประกาศใน type ว่าวางเวลาแบบใด

5.4.1.3 แอตทริบิวต์ที่จัดเก็บข้อมูลและเวลาที่ข้อมูลเป็นจริง (Valid Time Attributes)

แอตทริบิวต์ประเภทนี้จะมีสัญลักษณ์ ◎ อยู่หน้าแอตทริบิวต์นั้น ๆ การเขียนเป็น TODL จะใส่ vt_attribute เข้าไปแทน attribute ปกติ ดังนั้นจากรูปที่ 5.27 สามารถเขียนให้อยู่ในรูป TODL ได้ดังนี้

```
interface Product
(extent Products )
{
        attribute string ProductId ;
        vt_attribute <P,DAY> string ProductName ;
        vt_attribute <P,DAY> float Priceamount ;
};
```

จาก TODL ด้านบนเป็นการวางเวลาที่แอตทริบิวต์นั้นเป็นจริงเข้าไปด้วย เวลาที่วางเข้าไปนี้ผู้ใช้จำเป็นต้องป้อนเข้าไปเอง จึงต้องมีการระบุว่าวางเวลาประเภทใด และใช้หน่วยของเวลาใดเป็นหน่วยเล็กที่สุด ดังตัวอย่าง vt_attribute <P,DAY> float Priceamount เป็นการวางเวลาแบบ Period โดยใช้หน่วยเล็กสุดเป็นวัน กล่าวคือ สามารถวางเวลาได้เป็นช่วงของวันเช่น [21/12/1999, 01/10/2000) ดังนั้นพารามิเตอร์ตัวแรกของ vt_attribute คือ ประเภทของเวลาที่ต้องการวางไว้ที่แอตทริบิวต์ ส่วนพารามิเตอร์ตัวที่สองคือ หน่วยของเวลาที่ต้องการจะวางเข้าไป จึงสรุปได้ดัง ตารางที่ 5.4 และตารางที่ 5.5

ตารางที่ 5.4 ประเภทของเวลาที่ป้อนเข้าไปใน vt_attribute

ประเภทของเวลา	ชนิด
T	Time Point
P	Period

ตารางที่ 5.5 หน่วยของเวลาที่เล็กที่สุดที่ต้องการป้อนเข้าไปใน vt_attribute

หน่วยของเวลา	ชนิด
DAY	วัน
MONTH	เดือน
YEAR	ปี
HOUR	ชั่วโมง
MINUTE	นาที
SECOND	วินาที

จากตารางที่ 5.4 และ 5.5 ผู้ใช้สามารถเลือกใช้ประเภทของเวลาแบบใดก็ได้ แล้วแต่ความต้องการของโปรแกรมประยุกต์ที่ออกแบบไว้

5.2.2.4 แอตทริบิวต์ที่จัดเก็บข้อมูล เวลาที่ข้อมูลเป็นจริง และเวลาที่ข้อมูลอยู่ในฐานข้อมูล (Bitemporal Time Attributes)

แอตทริบิวต์ประเภทนี้จะมีสัญลักษณ์  อยู่หน้าแอตทริบิวต์นั้น ๆ ซึ่งเมื่อแปลงเป็น TODL แล้วจะใช้ bt_attribute แทน attribute กำกับไว้หน้าแอตทริบิวต์ประเภทนี้ สำหรับตัวอย่างนั้น เมื่อเราแปลงรูปที่ 5.29 ให้เป็น TODL แล้วจะได้ดังตัวอย่างด้านล่างนี้

```
interface Employee
( extent Employees )
{
        attribute string EId ;
        attribute string EName ;
        attribute char Sex ;
        attribute float  Height ;
        bt_attribute<P,DAY> float Salary ;
        attribute date Bdate ;
        attribute struct AddressStruct {
                        attribute string Houseid ;
                        attribute string Ctname ;
                        attribute string Cityname ;
                        attribute string Stname ;
                }Address ;

        float Calulate_Tax(in float SAmount) ;
        short Age(in date BDate) ;
        float Calculate_Bonus(in float SAmount) ;
};
```

จาก TODL ด้านบนจะเห็นได้ว่าในการวางเวลาทั้งเวลาที่ข้อมูลเป็นจริงและเวลาที่ข้อมูลอยู่ในฐานข้อมูลนั้น จะต้องมีการระบุประเภทของเวลาและหน่วยของเวลาด้วย สิ่งที่ระบุนี้จะนำไปใส่ให้กับเวลาที่ข้อมูลเป็นจริง ซึ่งหลักเกณฑ์ในการเลือกประเภทของเวลาและหน่วยของเวลานั้น มีหลักเกณฑ์เดียวกับหัวข้อก่อนหน้านี้ และใช้ตารางที่ 5.3 และ 5.4 เหมือนกัน สำหรับเวลาที่ข้อมูลอยู่ในฐานข้อมูลนั้นไม่ต้องระบุเพราะระบบจะจัดการให้อยู่แล้ว

5.4.2 โครงร่างหลัก

สำหรับโครงร่างหลักนั้นเป็นการแสดงความสัมพันธ์ระหว่างคลาส การถ่ายทอดคุณสมบัติ ความสัมพันธ์หลายบทบาท การนำเวลาเข้าไปเพิ่มในโครงร่างหลักนี้ ทำได้เฉพาะความสัมพันธ์ระหว่างคลาส และความสัมพันธ์หลายบทบาทเท่านั้น ดังนั้นหากต้องการแปลงโครงร่างหลักให้อยู่ในรูปของภาษา TODL ต้องทำเฉพาะความสัมพันธ์ระหว่างคลาสและความสัมพันธ์หลายบทบาทเท่านั้น ซึ่งสามารถสรุปได้ดังนี้

5.4.2.1 ความสัมพันธ์ไม่มีเวลามาเกี่ยวข้อง (Static Relationships)

ความสัมพันธ์นี้จะไม่มีการจัดเก็บเวลาของความสัมพันธ์ ดังนั้นจึงไม่ต้องคำนึงถึงส่วนที่ต้องเพิ่มเข้าไป ทำให้ ความสัมพันธ์ที่ปรากฏใน TODL นี้มีลักษณะคล้ายกับ ODL แต่จะต่างกันตรงที่แอตทริบิวส์ ซึ่งหากในคลาสนั้นมีแอตทริบิวต์ที่มีเวลาเข้ามาเกี่ยวข้องก็อาจจะต้องใช้สัญญลักษณะของเวลาเข้าไปเกี่ยวข้อง แต่สำหรับความสัมพันธ์แล้วนั้นยังมีลักษณะเหมือนกับ ODL ทุกประการ สำหรับตัวอย่างในกรณีนั้นจะปรากฏในรูปที่ 5.10 ซึ่งสามารถแปลงให้เป็น TODL ได้ดังนี้

```
interface Employee
(……..)
{
        ……..
        relationship Set<Computer> uses
                inverse Computer :: is_used_by ;
        ……..
};

interface Computer
(……..)
{
……..
        relationship Employee : is_used_by
                inverse Employee :: uses;
……..
};
```

5.4.2.2 ความสัมพันธ์ที่จัดเก็บเวลาที่ความสัมพันธ์นั้นอยู่ในฐานข้อมูล (Relationships with Transaction Time)

ความสัมพันธ์นี้มีการจัดเก็บเวลาที่ความสัมพันธ์นั้นอยู่ในฐานข้อมูล ซึ่งเวลาที่จัดเก็บนี้ระบบจะจัดหาให้ โดยความสัมพันธ์ใดเป็นความสัมพันธ์ประเภทนี้จะต้องมีสัญลักษณ์  ไว้ที่ความสัมพันธ์นั้น ๆ ซึ่งเมื่อแปลงจากรูป NIAM ที่มีความสัมพันธ์นี้อยู่ให้กลายเป็นภาษา TODL นั้นจะใช้ tt_relationship แทน relationship ใน ODL ปกติ เพราะการจัดเก็บเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูลเข้าไปด้วยนั้น จะต้องมีกระบวณการเพิ่มเติมจากเดิมอีกขั้นหนึ่ง ซึ่งตัวอย่างของความสัมพันธ์แบบนี้มีอยู่ในรูปที่ 5.14 โดยสามารถแปลงตัวอย่างดังกล่าวให้อยู่ในรูปภาษา TODL ได้ดังนี้

```
interface Employee
(……..)
{
        ……..
        tt_relationship Set<Computer> uses
                inverse Computer :: is_used_by ;
        ……..
};

interface Computer
(……..)
{
        ……..
        tt_relationship Employee : is_used_by
                inverse Employee :: uses;
        ……..
};
```

จะเห็นได้ว่า TODL ของความสัมพันธ์แบบนี้แตกต่างจากความสัมพันธ์ในหัวข้อ 5.2.3.1 เพียงแค่เปลี่ยน relationship ให้เป็น tt_relationship เท่านั้นเอง แต่หากต้องการให้ทำงานได้ก็ต้องแปลงกลับให้เป็นภาษา ODL ก่อน

5.4.2.3 ความสัมพันธ์แบบที่มีเวลาที่ความสัมพันธ์เป็นจริง (Relationships with Valid Time)

ความสัมพันธ์นี้จะต้องจัดเก็บเวลาที่ความสัมพันธ์เป็นจริงด้วย ดังนั้นผู้ใช้งานจะต้องป้อนเวลาเข้าไป ซึ่งเวลาที่ต้องป้อนเข้าไปนั้นผู้ใช้สามารถเลือกได้ตามต้องการ ดังที่กล่าวมาแล้วในตารางที่ 5.3 และ 5.4 สำหรับตัวอย่างของความสัมพันธ์ประเภทนี้อยู่ในรูปที่ 5.16 การที่จะแปลงตัวอย่างดังกล่าวให้เป็นภาษา TODL นั้น ผู้ใช้งานจะต้องป้อนเวลาเข้าไปในความสัมพันธ์ ซึ่งจะมีรูปแบบดังนี้

```
interface Employee
(……..)
{
        ……..
        vt_relationship <P,DAY > Set<Department> work_for
                inverse Department :: employs ;
        ……..
};

interface Department
(……..)
{
        ……..
        vt_relationship <P,DAY> Set<Employee> employs
                inverse Employee :: work_for;
        ……..
};
```

จะเห็นได้ว่าความสัมพันธ์ที่จัดเก็บเวลาที่ความสัมพันธ์เป็นจริงนั้น ผู้ใช้จะต้องป้อนประเภทของเวลาและหน่วยของเวลาที่เล็กที่สุดให้กับระบบ ระบบจึงจะสามารถนำรูปแบบเวลาที่ต้องการไปป้อนให้ได้ สำหรับการเลือกประเภทของเวลาและหน่วยของเวลานั้นจะเลือกโดยใช้ตารางที่ 5.3 และ 5.4

5.4.2.4 ความสัมพันธ์ที่จัดเก็บทั้งเวลาที่ความสัมพันธ์เป็นจริงและเวลาที่ความสัมพันธ์อยู่ในฐานข้อมูล (Relationships with Bitemporal Time)

ความสัมพันธ์แบบนี้จะต้องป้อนเวลาที่ความสัมพันธ์นั้นเป็นจริงและเวลาที่ความสัมพันธ์นั้นอยู่ในฐานข้อมูลเข้าไปด้วย ดังนั้นผู้ใช้จะต้องป้อนประเภทของเวลาและหน่วยของเวลาที่ต้องการลงไปด้วย ดังนั้นจะต้องเลือกประเภทและหน่วยของเวลาจากตารางที่ 5.3 และ 5.4 ก่อน ซึ่งรูปแบบของภาษา TODL ของความสัมพันธ์แบบนี้จะเหมือนกับภาษาในหัวข้อ 5.4.2.3 แต่จะใช้ bt_relationship แทน ตัวอย่างของความสัมพันธ์นี้มีอยู่ในรูปที่ 5.19 ซึ่งแปลงให้เป็น TODL ได้ดังนี้

```
interface Customer
(……..)
{
        ……..
        bt_relationship <T,DAY > Set<Product> has_perchased
                inverse Product :: has_be_bought_by ;
        ……..
};
interface Product
(……..)
{
        ……..
        bt_relationship <T,DAY> Set<Customer> has_be_bought_by
                        inverse Customer :: has_purchased;
        ……..
};
```

5.4.2.5 ความสัมพันธ์แบบหลายบทบาท (N-ary Relationships)

ความสัมพันธ์นี้เป็นความสัมพันธ์หลายบทบาทในความสัมพันธ์เดียว ดังรูปที่ 5.21 ซึ่งสามารถแปลงให้เป็นภาษา TODL โดยการสร้างคลาสใหม่เพื่อจัดเก็บ Simple Object ที่เกี่ยวข้อง N-ary fact type และสร้างความสัมพันธ์ของคลาสที่เกี่ยวเนื่องเข้าหาคลาสใหม่นี้ ดังรายละเอียดต่อไปนี้

```
interface Student
(……..)
{
        ……..
        relationship Set<Grade>  Scores
                inverse Grade :: Student;
        ……..
};

interface Subject
(……..)
{
        ……..
        relationship Set<Grade>  Students
                inverse Grade :: Subject;
        ……..
};

interface Grade
(extent Grades)
{
        relationship Student  Student
                inverse Student :: Scores;
        relationship Subject Subject
                inverse Subject ::  Students ;
        attribute float GD ;
};
```

หากมีเวลาเข้ามาเป็นส่วนประกอบในความสัมพันธ์ประเภทนี้ การแปลงเป็น TODL ก็เพียงแต่เพิ่มเติมรูปแบบ relationship และเพิ่มเติมเซ็ตของแอตทริบิวต์เข้าไปเท่านั้น

จาก TODL ทั้งหมดที่กล่าวมา สิ่งที่แตกต่างไปจาก ODL คือ tt_attribute, vt_attribute, bt_attribute, tt_relationship, vt_relationship, bt_relationship ซึ่งเป็นการประกาศในภาษา TODL ซึ่งยังใช้งานไม่ได้กับระบบฐานข้อมูลเชิงวัตถุ โดยต้องแปลงเป็นภาษา ODL เสียก่อนจึงจะทำงานได้ ซึ่งการแปลง TODL ให้เป็น ODL จะอยู่ในหัวข้อ 5.5

5.5 การแปลงภาษา TODL ให้เป็นภาษานิยามเชิงวัตถุ ODL

ภาษานิยามเชิงวัตถุ (Object Definition Language : ODL) เป็นภาษาสำหรับการสร้างอ๊อบเจ็กต์บนฐานข้อมูลเชิงวัตถุ เพื่อนำอ๊อบเจ็กต์ที่สร้างนี้ไปทำงานเพื่อจัดเก็บ และเรียกค้นข้อมูลตามต้องการได้ โดยภาษานี้เป็นส่วนหนึ่งของภาษา ODQL ซึ่งได้มีมาตรฐานของ ODMG รองรับ โดยมาตรฐานปัจจุบันนั้นอยู่ในเวอร์ชั่นที่ 2 (ODMG 2.0) ซึ่งระบบฐานข้อมูลเชิงวัตถุทั่วไปจะต้องใช้มาตรฐานนี้เป็นหัวใจหลัก แต่ระบบฐานข้อมูลเชิงวัตถุที่มีใช้งานกันอยู่นั้นส่วนมากแล้วยังไม่ได้รองรับมาตรฐาน ODMG 2.0 นี้ครบถ้วน ดังนั้นระบบฐานข้อมูลเชิงวัตถุหลาย ๆ ผลิตภัณฑ์ อาจมีโครงสร้างของภาษาที่แตกต่างออกไปจากมาตรฐานดังกล่าว จึงเป็นหน้าที่ของผู้ใช้งานจะต้องศึกษาและทำความเข้าใจกับภาษาที่เพิ่มเติมเข้ามาของระบบฐานข้อมูลเหล่านั้นเอง

สำหรับภาษา ODL นี้ได้กล่าวมาแล้วบางส่วนในบทที่ 4 ซึ่งในหัวข้อนี้จะกล่าวเฉพาะการนำ ODL มาใช้ในงานฐานข้อมูลเชิงเวลา โดยจะนำภาษา TODL ที่ปรากฏอยู่ในหัวข้อ 5.4 มาสร้างเป็นภาษา ODL เพื่อนำไปใช้งานจริง สำหรับรายละเอียดมีดังนี้

5.5.1 เวลาที่จะจัดเก็บ (Time)

สำหรับเวลาที่จะจัดเก็บของไปในแอตทริบิวต์หรือความสัมพันธ์นั้น เป็นเรื่องที่สำคัญมาก ๆ เพราะผู้ใช้จะต้องกำหนดถึงลักษณะของเวลาที่จะนำเข้าไปจัดเก็บ สำหรับเวลาทั้งหมดที่มีใช้งานมีดังนี้

- Timepoint ประกอบด้วย วันเดือนปี ชั่วโมง นาที และวินาที โดยผู้ใช้จะกำหนดหน่วยของเวลาเข้าไป เพื่อต้องการให้วางเวลาเฉพาะหน่วยที่ต้องการเท่านั้น มิได้วางเวลาตาม Timestamp
- Period ช่วงเวลาระหว่าง 2 จุดเวลา ที่นำ Timepoint มาใช้งาน

เวลาต่าง ๆ ข้างต้นเป็นเวลาที่ใช้งานในระบบ โดยที่เวลา 4 ลำดับแรกนั้น จะมีให้ใช้ในมาตรฐาน ODMG 2.0 สำหรับเวลา 2 ลำดับท้ายเป็นเวลาที่สร้างขึ้นมาใหม่ โดยใช้โครงสร้างมาจากเวลาที่มีอยู่แล้วเป็นพื้นฐาน สำหรับการสร้างเวลามีรายละเอียดดังนี้

5.5.1.1 Timepoint

เป็นเวลาที่นำ Timestamp มาเพิ่มหน่วยของเวลาเข้าไป กล่าวคือผู้ใช้สามารถกำหนดหน่วยของเวลาเข้าไปด้วยว่าจะนำเวลาส่วนใดมาใช้งาน มีคลาสคือ

```
class Timepoint {
    // Type Declarations
        enum Granularity { YEAR =0, MONTH=1, DAY=2, HOUR=3,
                           MINUTE=4,  SECOND=5 }

    // Properties
        attribute  Timestamp  Point;
        attribute Granularity GRANULARITY ;

    // Constructors
        Timepoint() ;                                          (1)
        Timepoint( g: Granularity);                            (2)

}
```

สำหรับ Constructors มีด้วยกัน 8 แบบคือ

(1) ให้เวลาเท่ากับเวลาปัจจุบันของระบบและมีหน่วยเวลาเป็น วินาที

(2) ใช้ เวลาที่ Timestamp โดยมีหน่วยของเวลาเป็น g

5.5.1.2 Period

เวลาชนิดนี้จะกำหนดช่วงเวลา โดยการนำหน่วยของเวลาเข้าไปเพิ่มขยายจาก Interval โดยมีคลาสดังนี้

```
class Period {
    // Type Declaration
        enum Granularity { YEAR =0, MONTH=1, DAY=2, HOUR=3,
                           MINUTE=4,  SECOND=5 }
    // Properties
        attribute Timepoint BEGIN  ;
        attribute Timepoint END  ;
        attribute Granularity GRANULARITY ;
    // Constructors
        Period(g: Granularity)  ;
    }
```

สำหรับ Constructor มีด้วยกัน 1 แบบคือ การกำหนดช่วงเวลาโดยใช้ หน่วยของเวลากำหนดรูปแบบของช่วงเวลา

5.5.2 แอตทริบิวต์

แอตทริบิวต์ที่เป็น TODL มีด้วยกัน 3 ประเภทคือ tt_attribute,vt_attribute และ bt_attribute ดังนั้นการแปลง TODL มีดังนี้

5.5.2.1 การแปลง tt_attribute .

รูปแบบของ tt_attribute คือ

tt_attribute Type tta ;

เมื่อแปลงแล้วจะได้

attribute set<struct element{**Type** value, **Period<SECOND>** tt_timestamp}> tta;

จะเห็นได้ว่ามีการแปลง tt_attribute ไปเป็นเซ็ตของข้อมูลและเวลาที่เป็น period

5.5.2.2 การแปลง vt_attribute

รูปแบบของ vt_attribute คือ

vt_attribute(tsc,gr) Type vta ;

เมื่อแปลงแล้วจะได้

1. ถ้า tsc มีค่าเท่ากับ P (Period)

attribute set<struct element{**Type** value, **Period<gr>** vt_timestamp}> vta;

2. ถ้า tsc มีค่าเท่ากับ T (Timepoint)

attribute set<struct element{**Type** value, **Timepoint<gr>** vt_timestamp}> vta;

5.3.2.3 การแปลง bt_attribute

รูปแบบของ bt_attribute คือ

bt_attribute(tsc,gr) Type vta ;

เมื่อแปลงแล้วจะได้

1. ถ้า tsc มีค่าเท่ากับ P (Period)

```
attribute set<struct element{Type value,  Period <SECOND>  tt_timestamp,
                                          Period<gr> vt_timestamp}> bta;
```

2. ถ้า tsc มีค่าเท่ากับ T (Timepoint)

```
attribute set<struct element{Type value, Period <SECOND>  tt_timestamp,
                                          Timepoint<gr> vt_timestamp}> bta;
```

5.5.3 ความสัมพันธ์ (Relationships)

สำหรับความสัมพันธ์ที่มีเวลาเข้ามาเกี่ยวข้องนั้น เมื่อแปลง TODL ให้กลายเป็น ODL แล้วจะมีการสร้างคลาสใหม่เพื่อแสดงความสัมพันธ์ดังตัวอย่าง

```
interface Manager
(……..)
{
...
        tt_relationship Set<MComputer> uses;
                inverse MComputer::is_used_by
        ...
};

interface MComputer
(……..)
{
...
        tt_relationship Manager is_used_by;
                inverse Manager:: uses;
        ...
};
```

จาก TODL ด้านบนสามารถแปลงเป็น ODLได้ดังนี้

```
interface Manager_uses_MComputer_is_used_by {
        attribute Period<SECOND> tt_timestamp;
        relationship Manager  i_uses
                inverse Manager::uses;
        relationship MComputer  i_is_used_by
                inverse MComputer::is_used_by;
}

interface Manager{
        ...
        relationship Set< Manager_uses_MComputer_is_used_by > uses
                inverse Manager_uses_MComputer_is_used_by:: i_uses;
        ...
};

interface MComputer {
        ...
        relationship Set< Manager_uses_MComputer_is_used_by > is_used_by
                inverse Manager_uses_MComputer_is_used_by:: i_ is_used_by;
        ...
};
```

หากเป็นความสัมพันธ์ของเวลาแบบอื่นก็สามารถแปลงได้ เพียงเปลี่ยนหน่วยของเวลาในคลาสของความสัมพันธ์เท่านั้น

สำหรับการแปลงความสัมพันธ์ระหว่างคลาสนั้นสามารถสรุปเป็นขั้นตอนดังนี้

<u>คลาส A :</u>

tt_relationship B r_A
inverse B:: r_B;

<u>คลาส B :</u>

tt_relationship A r_B
inverse A:: r_A;

1. สร้างคลาสใหม่โดยกำหนดคุณสมบัติดังนี้

```
interface A_r_A_B_r_B {
        attribute Period<SECOND> tt_timestamp;
        relationship A  i_r_A
                inverse A r_A;
        relationship B  i_r_B
                inverse B r_B;
};
```

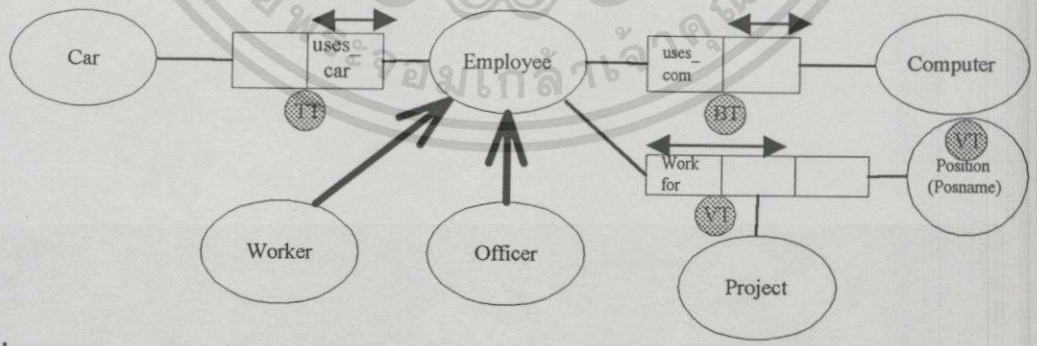
2. กำหนดความสัมพันธ์ของคลาส A ใหม่ โดยกำหนดความสัมพันธ์ไปที่คลาสที่สร้างขึ้นใหม่

```
relationship Set< A_r_A_B_r_B > r_A
        inverse A_r_A_B_r_B:: i_r_A;
```

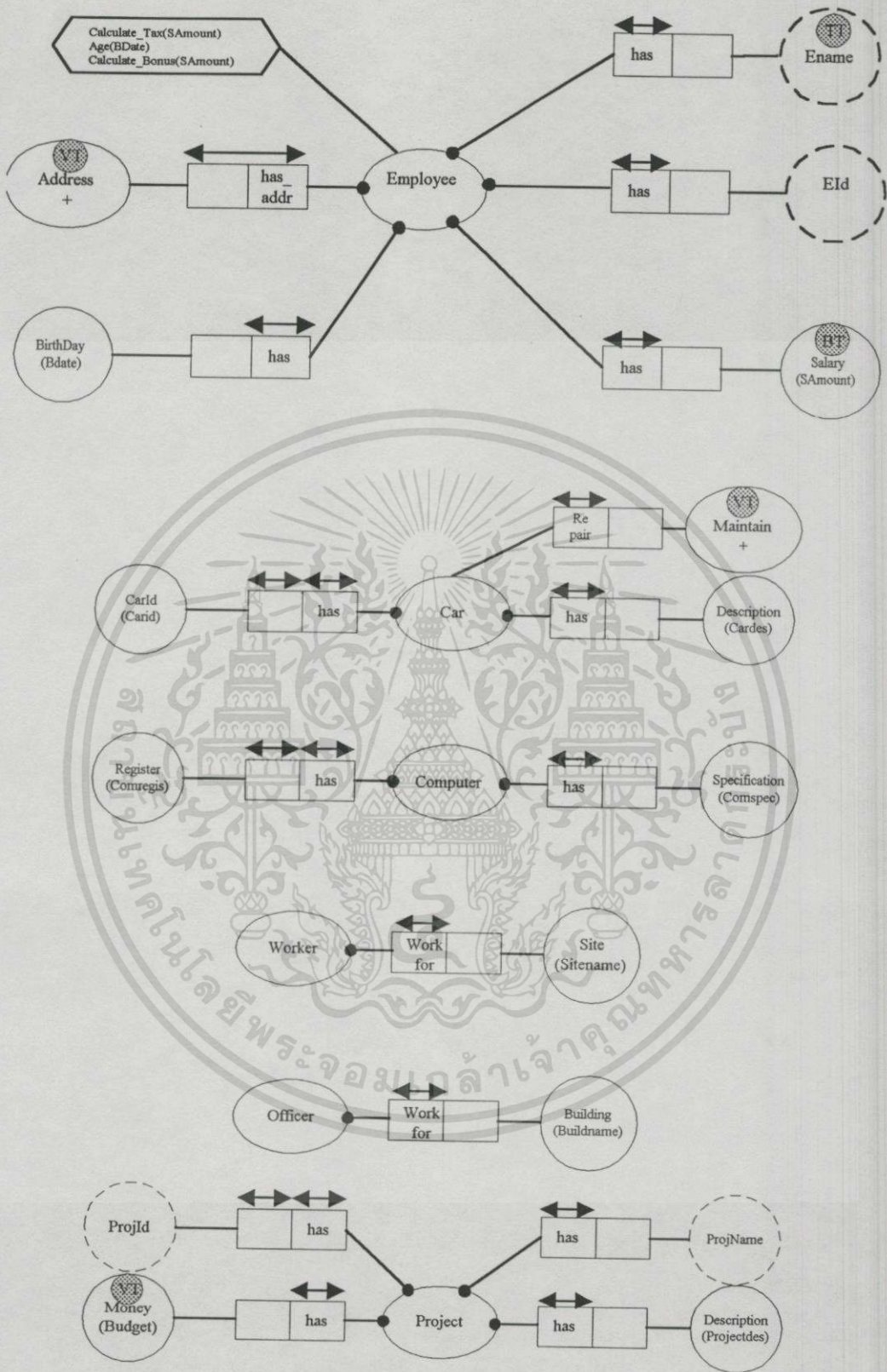
3. กำหนดความสัมพันธ์ของคลาส B ใหม่ โดยกำหนดความสัมพันธ์ไปที่คลาสที่สร้างขึ้นใหม่

```
relationship Set< A_r_A_B_r_B > r_B
        inverse A_r_A_B_r_B:: i_r_B;
```

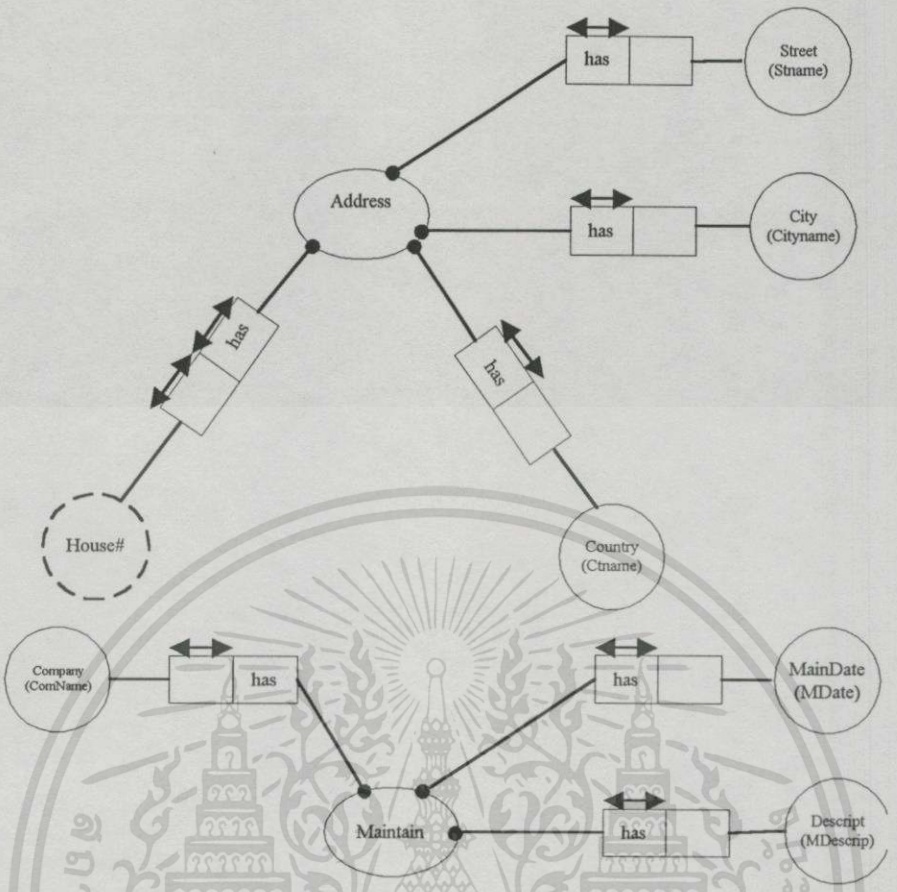
5.6 ตัวอย่างของ TOONIAM และการแปลงให้เป็นภาษา ODL



รูปที่ 5.37 ตัวอย่างโครงร่างหลักของระบบ



รูปที่ 5.38 ตัวอย่างโครงร่างย่อยระดับที่ 1



รูปที่ 5.39 ตัวอย่างโครงร่างย่อยระดับที่ 2

5.6.1 แปลงตัวอย่าง OONIAM ให้เป็นภาษา ODL

จากตัวอย่างด้านบนสามารถแปลง OONIAM ให้เป็นภาษา ODL ได้ดังนี้

```
interface Employee
( extent Employees )
{
        typedef  Struct  AddressStruct {
                          string Houseid ,
                          string Ctname,
                          string Cityname,
                          string Stname ;}

        attribute string EId ;
        attribute string Ename ;
        attribute float Salary ;
        attribute date Bdate ;

        attribute  set<AddressStruct>  Address ;

        relationship  Car uses_car  inverse Car :: is_used_by ;
        relationship set<Computer> uses_com inverse
                                        Computer :: is_used_by ;
        relationship set<Position>  work_for inverse
                                        Position :: Employees ;
```

```
        float Calulate_Tax(in float SAmount) ;
        float Age(in float BDate) ;
        float Calulate_Bonus(in float SAmount)
};


interface Worker : Employee
(       extent Workers)
{
        attribute string SiteName ;
};

interface Officer : Employee
(       extent Officers)
{
        attribute string Buildname ;
};


interface Computer
(       extent Computers )
{
        attribute string Comregis ;
        attribute string Comspec ;

        relationship Employee is_used_by  inverse
                                        Employee :: uses_com ;
};
interface Car
(       extent Cars )
{
        typedef struct MaintainStruct{
                date Mdate,
                string Mdescrip,
                string ComName ;
                } ;

        attribute string Carid ;
        attribute string Cardes ;
        attribute MaintainStruct Maintain ;

        relationship set < Employee >  is_used_by  inverse
                                        Employee :: uses_car ;
};

interface Project
(       extent Projects )
{

        attribute string ProjId ;
        attribute string ProjName ;
        attribute string Projectdes ;
        attribute float Budget ;

        relationship set < Position >  Employees  inverse
                                        Position :: Projects ;
};
```

```
interface Position
(       extent Positions )
{


        relationship Employee  Employees  inverse
                                        Employee :: Work_for ;
        relationship Project  Projects  inverse
                                        Project :: Employees ;
        Attribute string Posname ;

};
```

5.6.2 แปลงตัวอย่าง TOONIAM ให้เป็นภาษา TODL

จากตัวอย่างด้านบนสามารถแปลง TOONIAM ให้เป็นภาษา TODL ได้ดังนี้

```
interface Employee
( extent Employees )
{
        typedef  Struct  AddressStruct {
                        string Houseid ,
                        string Ctname,
                        string Cityname,
                        string Stname ;}

        attribute string EId ;
        tt_attribute string Ename ;
        bt_attribute<P,MONTH> float Salary ;
        attribute date Bdate ;

        vt_attribute<T,DAY>  set<AddressStruct>  Address ;

        tt_relationship  Car uses_car  inverse Car :: is_used_by ;
        bt_relationship<P,DAY> set<Computer> uses_com inverse
                                        Computer :: is_used_by ;
        vt_relationship<P,MONTH> set<Position>  work_for inverse
                                        Position :: Employees ;

        float Calulate_Tax(in float SAmount) ;
        float Age(in float BDate) ;
        float Calulate_Bonus(in float SAmount)
};


interface Worker : Employee
(       extent Workers)
{
        attribute string SiteName ;
};

interface Officer : Employee
(       extent Officers)
{
        attribute string Buildname ;
};
```

```
interface Computer
(       extent Computers )
{
        attribute string Comregis ;
        attribute string Comspec ;

        bt_relationship<P,DAY> Employee is_used_by  inverse
                                                Employee :: uses_com ;
};

interface Car
(       extent Cars )
{
        typedef struct MaintainStruct{
                date Mdate,
                string Mdescrip,
                string ComName ;
                } ;

        attribute string Carid ;
        attribute string Cardes ;
        vt_attribute<T,DAY> MaintainStruct Maintain ;

        tt_relationship set < Employee >  is_used_by  inverse
                                                Employee :: uses_car ;
};

interface Project
(       extent Projects )
{

        attribute string ProjId ;
        attribute string ProjName ;
        attribute string Projectdes ;
        vt_attribute<P,YEAR> float Budget ;

        vt_relationship<P,MONTH> set < Position >  Employees  inverse
                                                Position :: Projects ;
};

interface Position
(       extent Positions )
{


        vt_relationship<P,MONTH> Employee  Employees  inverse
                                                Employee :: Work_for ;
        vt_relationship<P,MONTH> Project  Projects  inverse
                                                Project :: Employees ;
        Attribute string Posname ;

};
```

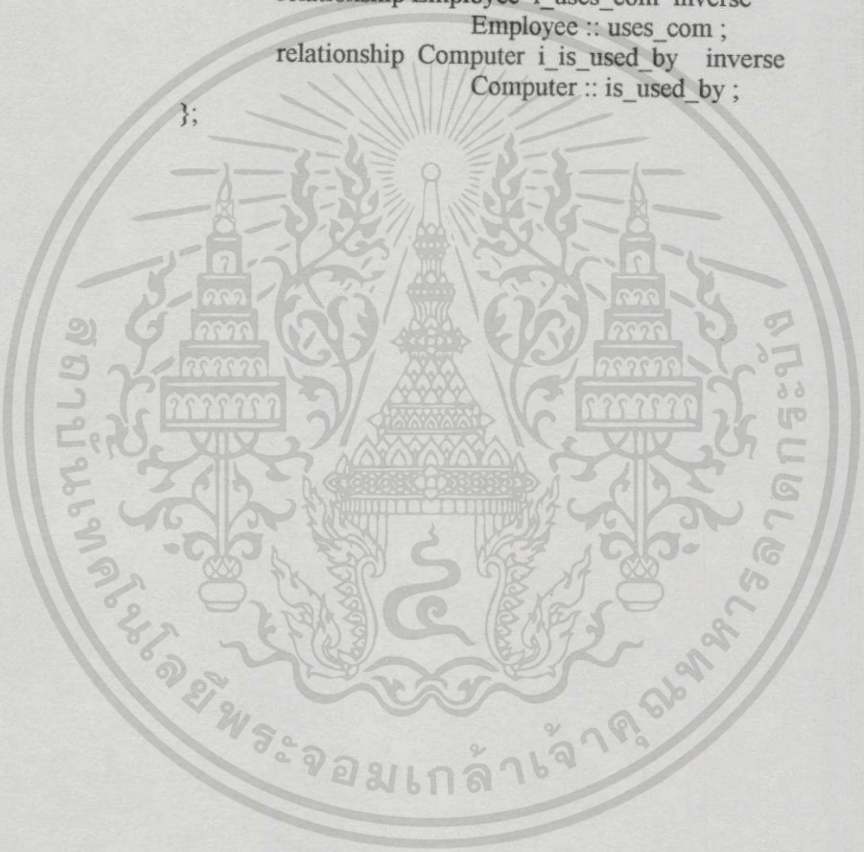
5.6.3 แปลงตัวอย่าง TODL ให้เป็นภาษา ODL

จากตัวอย่างด้านบนสามารถแปลง TODL ให้เป็นภาษา ODL ได้ดังนี้

```
interface Employee
( extent Employees )
{
        typedef  Struct  AddressStruct {
                        string Houseid ,
                        string Ctname,
                        string Cityname,
                        string Stname ;}

        attribute string EId ;
        attribute set <struct EnameStruct { string value,
                Period<SECOND> tt_timestamp}>  Ename ;
        attribute set<struct SalaryStruct{float value,
                Period<SECOND> tt_timestamp,
                Period<MONTH> vt_timestamp}> Salary ;
        attribute date Bdate ;

        attribute set<struct AddressStruct{AddressStruct  value,
                Timepoint<MONTH> vt_timestamp}> Address ;

    relationship set<Employee_uses_car_Car_is_used_by> uses_car
                inverse Employee_uses_car_Car_is_used_by:: i_uses_car ;
    relationship set<Employee_uses_com_Computer_is_used_by> uses_com
                inverse Employee_uses_com_Computer_is_used_by ::
                i_is_used_com ;
    relationship set< Position >  work_for inverse
                Position  :: employee ;

        float Calulate_Tax(in float SAmount) ;
        float Age(in float BDate) ;
        float Calulate_Bonus(in float SAmount)
};

interface Worker : Employee
(       extent Workers)
{
        attribute string SiteName ;
};

interface Officer : Employee
(       extent Officers)
{
        attribute string Buildname ;
};
```

```
interface Computer
(       extent Computers )
{
        attribute string Comregis ;
        attribute string Comspec ;

   relationship set< Employee_uses_com_Computer_is_used_by >
        is_used_by  inverse Employee_uses_com_Computer_is_used_by ::
        i_is_used_by ;
};


interface Car
(       extent Cars )
{
        typedef struct MaintainStruct{
                date Mdate,
                string Mdescrip,
                string ComName ;
                } ;

        attribute string Carid ;
        attribute string Cardes ;
        attribute set<struct MaintainStruct{MaintainStruct value,
                Timepoint<DAY> vt_timestamp}> Maintain ;

        relationship set<Employee_uses_car_Car_is_used_by> is_used_by
                inverse  Employee :: i_is_used_by ;
};

interface Project
(       extent Projects )
{

        attribute string ProjId ;
        attribute string ProjName ;
        attribute string Projectdes ;
        attribute set<struct BudgetStruct{float value,
                Period<YEAR> vt_timestamp}> Budget ;


   relationship set<Position>  employee  inverse
                Position  :: project ;
};

interface Position
(       extent Positions )
{

        attribute string Posname ;
        attribute Period <DATE> vt_timestamp ;

   relationship Emplyee employee  inverse
        Employee:: work_for
   relationship Project  Project
         inverse Project:: employee ;
};
```

```
interface Employee_uses_car_Car_is_used_by
{
        attribute Period<SECOND> tt_timestamp ;

        relationship Employee  i_uses_car  inverse
                        Employee :: uses_car ;
        relationship  Car i_is_used_by    inverse
                        Car :: is_used_by ;
};

interface Employee_uses_com_Computer_is_used_by
{
        attribute Period<DAY> vt_timestamp ;

        relationship Employee  i_uses_com  inverse
                        Employee :: uses_com ;
        relationship  Computer  i_is_used_by    inverse
                        Computer :: is_used_by ;
};
```



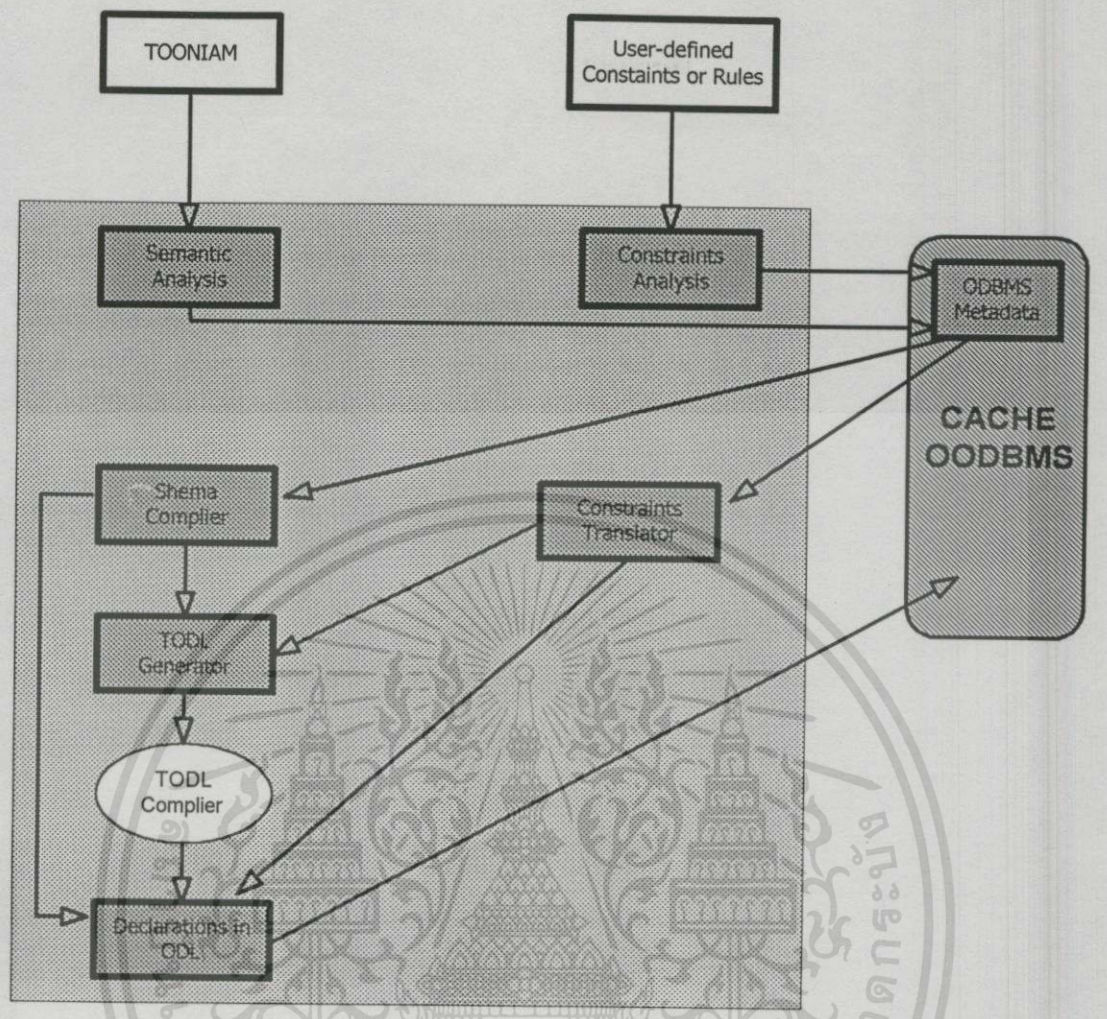
บทที่ 6

การพัฒนาโปรแกรมแบบจำลอง TOONIAM

ในบทนี้จะอธิบายถึงการพัฒนาโปรแกรมในการสร้างโมเดล TOONIAM และการแปลงโมเดลที่สร้างขึ้นให้เป็นภาษานิยามเชิงวัตถุ (ODL) ซึ่งได้มีการแยกการอธิบายเป็นส่วน ๆ คือ ในหัวข้อ 6.1 อธิบายถึงสถาปัตยกรรมของระบบ ซึ่งเป็นส่วนสำคัญที่จะแสดงถึงขั้นตอนการทำงานทั้งหมด ในหัวข้อ 6.2 คืออุปกรณ์ที่ใช้สร้างโปรแกรมนี้ สำหรับหัวข้อ 6.3 ได้อธิบายถึงการออกแบบฐานข้อมูล Meta data เพื่อจัดเก็บโครงสร้างของ TOONIAM ที่ผู้ใช้สร้างขึ้น เพื่อนำฐานข้อมูล Meta นี้ไปสร้างเป็นภาษา ODL อีกทอดหนึ่ง หัวข้อสุดท้ายคือหัวข้อ 6.4 อธิบายถึงขั้นตอนการประมวลผลของโปรแกรม หรือ Flow Chart ต่าง ๆ นั่นเอง

6.1 สถาปัตยกรรมของระบบ

สำหรับโปรแกรมประยุกต์ที่สร้างขึ้นเพื่อแปลงแบบจำลอง TOONIAM ให้กลายเป็นภาษา ODL นั้น มีรายละเอียดดังรูปที่ 6.1



รูปที่ 6.1 สถาปัตยกรรมของระบบการแปลง TOONIAM ให้เป็นภาษา ODL

ซอฟต์แวร์ที่สร้างขึ้นนี้ มีจุดประสงค์ที่จะนำแบบจำลอง TOONIAM ที่ผู้ใช้งานออกแบบมาแปลงให้เป็นภาษา ODL เพื่อสร้าง Schema ของฐานข้อมูลเชิงเวลา และนำ Schema ที่ได้นี้ไปใช้งานกับระบบฐานข้อมูลเชิงวัตถุ สำหรับระบบฐานข้อมูลเชิงวัตถุที่นำมาใช้งานคือ ระบบฐานข้อมูล Caché โดยระบบฐานข้อมูลดังกล่าวนี้จะจัดเก็บ Meta data ของ TOONIAM จากนั้นจะนำ Meta Data นี้ไปผ่านกระบวนการเพื่อแปลงเป็น ODL จากนั้นจะนำภาษา ODL นี้กลับไปทำงานบนระบบฐานข้อมูล Caché อีกครั้ง เพื่อให้ระบบฐานข้อมูลสร้างฐานข้อมูล ตามที่ผู้ใช้งานออกแบบไว้

6.2 อุปกรณ์ที่ใช้

สำหรับอุปกรณ์ที่นำมาใช้เพื่อสร้างโปรแกรมนี้นั้นประกอบด้วยฮาร์ดแวร์และซอฟต์แวรต์ดังนี้

1. เครื่องคอมพิวเตอร์

- หน่วยประมวลผลกลาง AMD-K6-2 ความเร็ว 400 เมกกะเฮริซ์
- หน่วยความจำ 64 เมกกะไบต์
- ฮาร์ดดิสก์ 6 กิกะไบต์

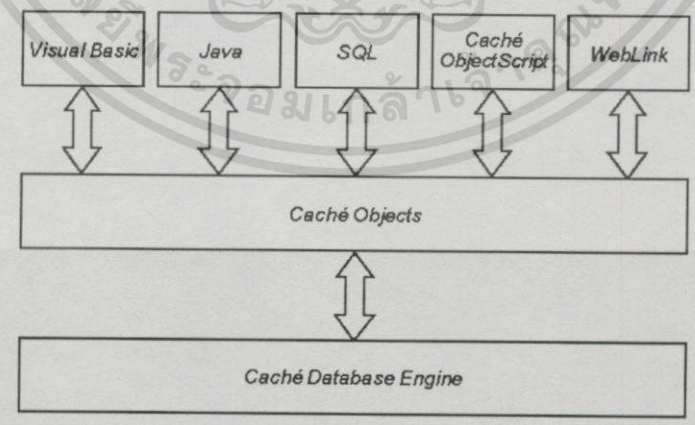
2. ระบบปฏิบัติการ Microsoft Windows 98

3. ระบบฐานข้อมูล Caché version 3.2.1.7

4. โปรแกรม Visual Basic version 6.0

สำหรับระบบฐานข้อมูลที่ใช้ในการจัดเก็บ Meta data และนำภาษา ODL มาทดสอบนั้นคือ ระบบฐานข้อมูล Caché ซึ่งเป็นระบบฐานข้อมูลที่สามารถรองรับโครงสร้างข้อมูลได้ทั้งฐานข้อมูลเชิงสัมพันธ์และฐานข้อมูลเชิงวัตถุ ซึ่งมีรายละเอียดดังนี้

ระบบจัดการฐานข้อมูลเชิงวัตถุที่ใช้ (ODBMS) ได้แก่ **Caché** (อ่านว่า คา-เช่) พัฒนาโดย InterSystems Corporation ซึ่งเป็นระบบจัดการฐานข้อมูลที่สนับสนุนเทคโนโลยีเชิงวัตถุ (Object-Oriented Technology) สามารถจำลองโครงสร้างของข้อมูลที่มีความซับซ้อนให้คงสภาพตามที่เป็นจริงโดยไม่ต้องเปลี่ยนรูปร่างของโครงสร้างข้อมูล ทำให้ลดความสับสนในการกำหนดความสัมพันธ์ของข้อมูลไปได้มาก สำหรับการใช้งาน Caché สามารถทำงานได้บนหลากหลายแพลตฟอร์ม (Hardware Platform) และหลากหลายระบบปฏิบัติการ (Operating System) อาทิเช่น Window 95/98/NT, UNIX และ Linux เป็นต้น



รูปที่ 6.2 การติดต่อกับระบบจัดการฐานข้อมูล Caché

6.2.1 คุณสมบัติด้านต่าง ๆ ของ Caché

6.2.1.1 Caché ใช้ ODBC ที่เรียกว่า Ultra Fast ODBC เพื่อให้สามารถใช้ภาษา SQL เป็นตัวติดต่อกับระบบฐานข้อมูลได้ และใช้ Visual Caché เพื่อให้ออบเจ็กต์ของ Caché สามารถใช้ได้กับ Visual Basic และเครื่องมือในการพัฒนา GUI อื่นๆ นอกจากนั้นยังมี Weblink ซึ่งทำให้สามารถเชื่อต่อฐานข้อมูล Caché ผ่านทางเครือข่ายอินเทอร์เน็ตได้ด้วย

6.2.1.2 Caché Object Script เป็นส่วนที่ใช้ในการพัฒนาโปรแกรมให้เป็นแบบออบเจ็กต์ได้ เพื่อเพิ่มความยืดหยุ่นในการใช้งาน และยังสามารถใช้ JAVA หรือ C++ ในการแก้ไขโปรแกรมได้อีกด้วย

6.2.1.3 วิธีการเข้าถึงข้อมูล สามารถเข้าถึงข้อมูลได้ 3 แบบ ดังตารางที่ 6.1

6.2.1.4 Transaction Database Engine ซึ่งเป็นตัวจัดการการทำงานของแอพพลิเคชั่น

6.2.1.5 Distributed Caché Protocol ช่วยในการทำงานทางด้านการติดต่อของระบบเครือข่าย

ตารางที่ 6.1 แสดงการเข้าถึงข้อมูลของ Caché

เครื่องมือ	รายละเอียด
Caché Multidimensional Access	เป็นการเข้าถึงข้อมูลแบบ Global variable ซึ่งจัดเก็บแบบ Multidimentional array
Caché SQL	เป็นเข้าถึงข้อมูลโดยผ่านทางภาษา SQL
Caché Objects	เป็นการเข้าถึงด้วย JAVA C++ และเทคโนโลยีในการพัฒนาทางด้านออบเจ็กต์อื่น ๆ

6.2.2 องค์ประกอบของ Caché

ตารางที่ 6.2 แสดงส่วนประกอบของ Caché

เครื่องมือ	รายละเอียด
Caché Object Server	ทำให้สามารถใช้งานออบเจ็กต์ของ Caché ได้โดยตรงผ่านทาง JAVA หรือ C++
Caché SQL Server	สามารถเข้าถึงข้อมูลผ่านทาง SQL และ ODBC
Caché Studio	ใช้ในการพัฒนา GUI ในการกำหนดคลาสของออบเจ็กต์และสร้างส่วนประกอบของแอพพลิเคชัน

ตารางที่ 6.2 แสดงส่วนประกอบของ Caché (ต่อ)

เครื่องมือ	รายละเอียด
Caché Distributed Caché Protocol (DCP)	สามารถกระจายฐานข้อมูลทั่วระบบเครือข่ายในการตอบรับต่อความต้องการ เพื่อให้ประสิทธิภาพดีที่สุด
Caché Object Script	เป็นภาษาในการเขียนโปรแกรมที่มีมาสำหรับแอพพลิเคชันด้านการประมวลผล ทรานแซคชั่นและควบคุมออบเจ็กต์ในระบบ
Caché Weblink	รองรับการเชื่อมต่อระหว่าง Caché กับเว็บเซิร์ฟเวอร์สำหรับประมวลผล ทรานแซคชั่นผ่านทางเว็บ
Visual Caché	เป็นเครื่องมือสร้างฟอร์มวิชวลเบสิกของไมโครซอฟท์อย่างอัตโนมัติและยังให้การเชื่อมต่อระหว่างออบเจ็กต์ของ Caché กับวิชวลเบสิกได้อัตโนมัติ

6.2.3 Object Model

Caché เป็นระบบจัดการฐานข้อมูลเชิงวัตถุ ดังนั้นจึงจัดการและออกแบบระบบในเชิงวัตถุทั้งหมด ซึ่ง Caché Object Model สนับสนุนลักษณะดังต่อไปนี้

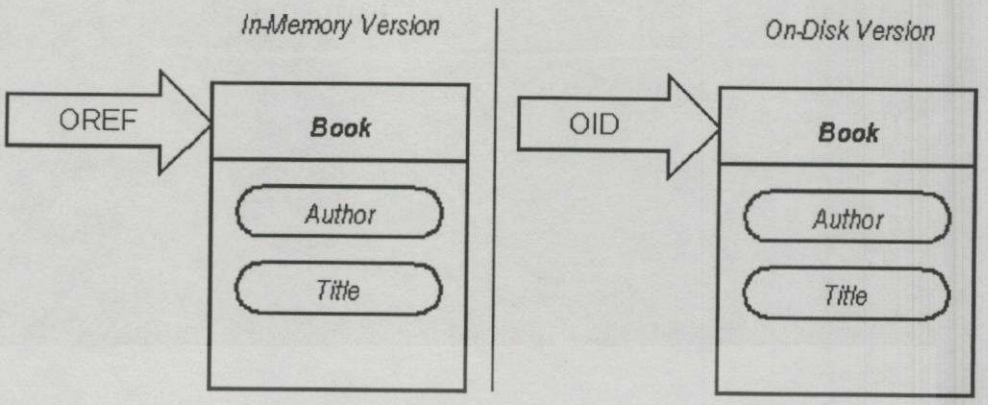
- คลาสหลักใน Caché จะเป็น Object Class และ Data Type Class (รวมทั้ง Literal ด้วย) โดย Object จะมีตัวชี้เฉพาะ (Unique Identifier) ส่วน Literal จะไม่มีตัวชี้เฉพาะ มีเพียงค่า (Value) เท่านั้น
- Object จะมีพฤติกรรมโดยอัตโนมัติซึ่งจัดการโดย Caché หรือถูกกำหนดโดยผู้ใช้ก็ได้
- Persistent Object สามารถถูกจัดเก็บในฐานข้อมูล Caché ได้

6.2.4 การอ้างอิงออบเจ็กต์

ในระบบจัดการของ Caché ออบเจ็กต์อาจอยู่ในดิสก์หรือในหน่วยความจำ ออบเจ็กต์ที่อยู่ในดิสก์จะเป็นออบเจ็กต์ที่ถูกจัดเก็บลงฐานข้อมูล ส่วนออบเจ็กต์ที่อยู่ในหน่วยความจำนั้นจะถูกโหลดดึงมาจากฐานข้อมูลและสามารถควบคุมได้ เราสามารถอ้างถึงออบเจ็กต์เหล่านี้ได้ 2 รูปแบบขึ้นอยู่กับว่าจะอ้างออบเจ็กต์นั้นหากมันถูกโหลดหรือถูกจัดเก็บอยู่ การอ้างอิงนั้นได้แก่

- OREF (Object Reference)
- OID (Object Identifier)

ดังแสดงในรูปที่ 6.3



รูปที่ 6.3 OREF และ OID

- OREF เป็นการอ้างถึงออบเจ็กต์ที่อยู่ในหน่วยความจำ แต่ละครั้งที่ออบเจ็กต์ถูกโหลดขึ้นมาอยู่บนหน่วยความจำ มันอาจจะมีค่า OREF ที่แตกต่างกันได้

- OID เป็นการอ้างถึงออบเจ็กต์ที่อยู่ในดิสก์ เป็นตัวชี้ออบเจ็กต์แบบ Persistent และมีค่าไม่ซ้ำ (Unique) ซึ่งถูกเก็บอยู่ในดิสก์ เมื่อออบเจ็กต์ถูกสร้างขึ้นและได้รับ OID แล้ว ก็จะมีค่านั้นไม่เปลี่ยนแปลง

เราสามารถใช้ OID ของออบเจ็กต์ Persistent นี้โหลดดึงออบเจ็กต์ขึ้นมาอยู่ในหน่วยความจำ จากนั้นระบบสามารถแทนออบเจ็กต์ตัวนั้นด้วย OREF ทำให้แอพพลิเคชันสามารถอ้างออบเจ็กต์และเข้าถึงค่าต่าง ๆ ของออบเจ็กต์ตัวนั้น ๆ ได้

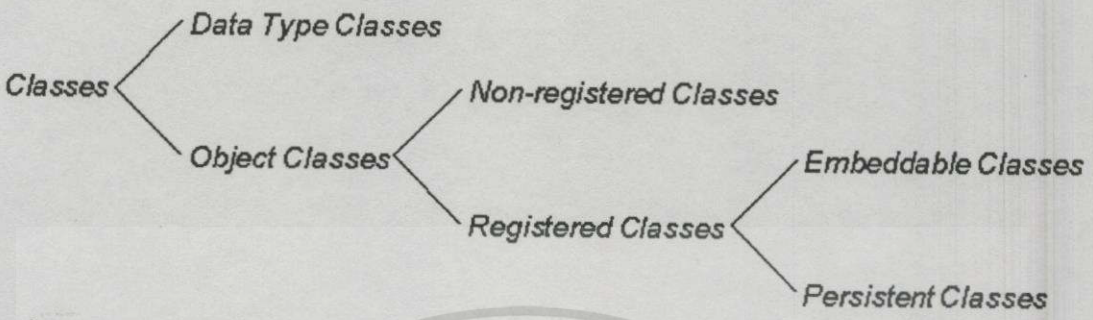
6.2.5 คลาส (Class)

คลาสใน Caché มี 5 ประเภท ได้แก่

- Registered Class
- Persistent Class
- Embeddable Class
- Non-Registered Class
- Data Type Class

คลาสต่าง ๆ เหล่านี้ถูกจัดเป็น Data Type class และ Object class โดย Data Type class จะแทน Literal Value เช่น สตริง (String), ตัวเลข (Integer) หรือวันที่ (Date) เป็นต้น Object class ส่วนใหญ่จะสืบทอดคุณสมบัติมาจากคลาสของระบบที่เรียกว่า %RegisteredObject ซึ่งจัดการและมีเมธอดหลัก ๆ ของออบเจ็กต์ให้มา ส่วนคลาสที่ไม่ได้สืบทอดคุณสมบัติมาจาก %RegisteredObject

เรียกว่า Non-Registed class ซึ่งคลาสชนิดนี้จะมีเพียงลักษณะพื้นฐานเท่านั้น เราจะต้องสร้างเมธอดให้แก่ออบเจ็กต์เหล่านี้เข้าไปเอง คลาสต่าง ๆ ใน Caché แสดงดังรูปที่ 6.4



รูปที่ 6.4 ประเภทของคลาส

6.2.5.1 Registered Class

อินสแตนซ์ของคลาสนี้สืบทอดคุณสมบัติมาจาก %RegisteredObject หรือเรียกว่าเป็น registered object โดยออบเจ็กต์เหล่านี้จะมีเซตของเมธอดพื้นฐานทั้งหมดที่ใช้จัดการออบเจ็กต์ที่อยู่ในหน่วยความจำ ครั้งเมื่อ Registered object นี้ถูกโหลดสู่หน่วยความจำ เราสามารถอ้างถึงออบเจ็กต์นี้โดยผ่าน OREF ได้

6.2.5.2 Persistent Class

อินสแตนซ์ของคลาสนี้สืบทอดคุณสมบัติมาจาก %Persistent และถูกเรียกว่า persistent object ซึ่งออบเจ็กต์เหล่านี้เป็น registered object ด้วย ออบเจ็กต์นี้มีความสามารถที่จะจัดการและเก็บตัวมันเองลงในฐานข้อมูลได้ จากถูกโหลดสู่หน่วยความจำ ออบเจ็กต์อื่นที่มันอ้างถึงจะไม่ถูกโหลดมาด้วย เพราะออบเจ็กต์เป็นอิสระต่อกันและเพียงแค่ชี้หากันเท่านั้น

หาก persistent object ถูกใช้เป็น Attribute หนึ่งของออบเจ็กต์อื่น มันจะถูกชี้มากจากออบเจ็กต์นั้น ๆ ยกตัวอย่างเช่น ให้ Doctor เป็น persistent object และเรากำหนดให้ Attribute ในออบเจ็กต์ตัวหนึ่งมีชนิดเป็น Doctor จะได้เป็น

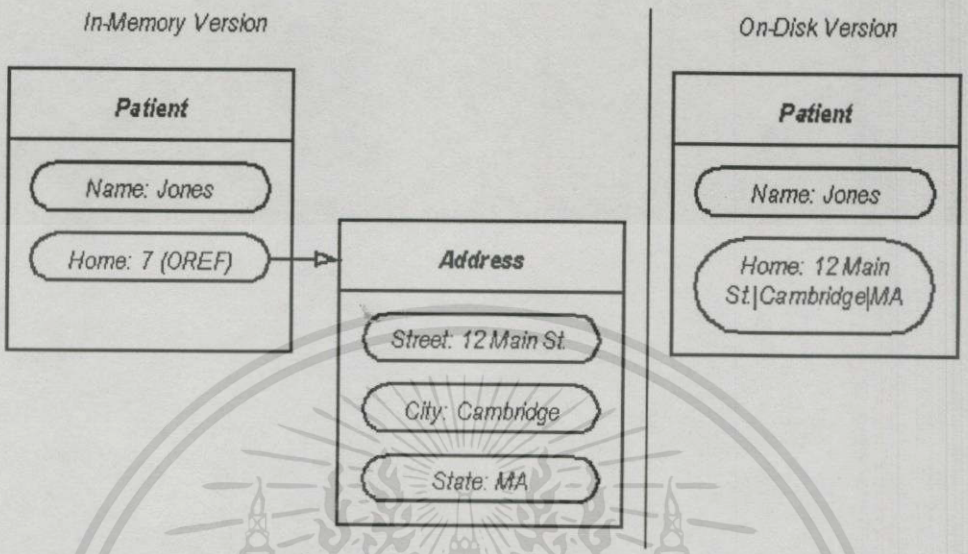
```
ATTRIBUTE TheDoc {TYPE = Doctor ;}
```

ซึ่ง Attribute TheDoc จะชี้ไปยังออบเจ็กต์ชนิด Doctor

6.2.5.3 Embeddable Class

อินสแตนซ์ของคลาสนี้สืบทอดคุณสมบัติมาจาก %SerialObject และถูกเรียกว่า Embedded object ซึ่งออบเจ็กต์เหล่านี้อ้างอิงได้อย่างอิสระเมื่อถูกโหลดสู่หน่วยความจำ แต่เมื่อจัด

เก็บลงฐานข้อมูลแล้ว ออบเจ็กต์เหล่านี้จะต้องฝังตัวอยู่ภายใน persistent object เท่านั้น Embedded object มีรูปแบบ 2 ลักษณะแสดงดังรูปที่ 6.5



รูปที่ 6.5 Embedded Object ในหน่วยความจำและดิสก์

- ในหน่วยความจำ, Embedded object ถูกแทนด้วยออบเจ็กต์อิสระคล้าย ๆ กับ persistent object หาก Attribute ใดอ้างชนิดเป็น Embedded object แล้วการจัดการในหน่วยความจำก็สามารถกำหนดเป็น OREF ได้
- ในดิสก์, Embedded object ถูกจัดเก็บเป็นเหมือน Attribute หนึ่งและถูกบรรจุลง persistent object ตัวหนึ่ง โดยมันจะไม่มี OID เป็นของตัวเองและไม่สามารถถูกอ้างโดยออบเจ็กต์ตัวอื่น ๆ อีกได้เลย ค่าของ Embedded object จะถูกเก็บลงฐานข้อมูลคล้าย ๆ เป็นเพียงสตริงใน Attribute ของออบเจ็กต์หลักเท่านั้น เหตุนี้จึงเรียกว่าเป็น SerialObject

ยกตัวอย่างเช่น

```
CLASS Address
{
        SUPER = %SerialObject ;
        ATTRIBUTE Street { TYPE = %String(MAXLEN=80) ; }
        ATTRIBUTE City { TYPE = %String(MAXLEN=3) ; }
        ATTRIBUTE State { TYPE = %String(MAXLEN=2) ; }
}
```

เมื่อกำหนดให้ออบเจ็กต์จากคลาส Address เป็น Embedded object attribute สำหรับ persistent object เช่นคลาส Patient จะได้

```
CLASS Patient
{
        SUPER = %Persistent ;
        PERSISTENT ;
        ATTRIBUTE Name { TYPE = %String ; }
        ATTRIBUTE Home { TYPE = Address ; }
}
```

6.2.5.4 Non-Registered Class

อินสแตนซ์ของคลาสนี้เป็นออบเจ็กต์ที่เหลือ ที่ไม่ได้สืบทอดคุณสมบัติมาจากคลาส %RegisteredObject อาจเกิดขึ้นจากการโปรแกรมของผู้ใช้ขึ้นเอง และเราจำเป็นต้องกำหนดค่าสำหรับ OREF ขึ้นมาเองเพราะระบบไม่ได้เตรียมหรือ register ค่าให้

6.2.5.5 Data Type Class

คลาสนี้กำหนดและควบคุมค่าของ Literal โดย data type ไม่มีตัวชี้อิสระเหมือนออบเจ็กต์ทั่วไปและไม่สามารถมีอินสแตนซ์ได้ ซึ่งมันเป็นเพียง Attribute ของออบเจ็กต์ซึ่งบรรจุมันเอาไว้ และตัวมันเองก็ไม่มี Property ด้วย Data Type class จะแทน Literal Value เช่น สตริง (String), ตัวเลข (Integer), วันที่ (Date), ตัวเลขทศนิยม (Float) และชนิดของข้อมูลพื้นฐานต่าง ๆ

6.2.6 Caché กับ Inheritance

Caché object model อนุญาตให้เราสามารถออกแบบคลาสด้วยการสืบทอดคุณสมบัติจากคลาสอื่นมาได้โดยการใช้คีย์เวิร์ด SUPER คลาสที่ได้จะมี specification เหมือนกับ Superclass ทุกประการรวมทั้ง Property, Method และ Class Parameter ด้วย ยกตัวอย่างเช่นคลาส Person และมีคลาส Employee มาทำการสืบทอดคุณสมบัติจากคลาส Person ดังนี้

```
CLASS Person
{
        SUPER = %Persistent ;
        PERSISTENT ;
        ATTRIBUTE Name { TYPE = %String ; }
        ATTRIBUTE DOB { TYPE = %Date ; }
}
CLASS Employee
{
        SUPER = Person ;
        PERSISTENT ;
        ATTRIBUTE Salary { TYPE = %Integer ; }
        ATTRIBUTE Department { TYPE = %String ; }
}
```

6.2.7 Caché กับ Multiple Inheritance

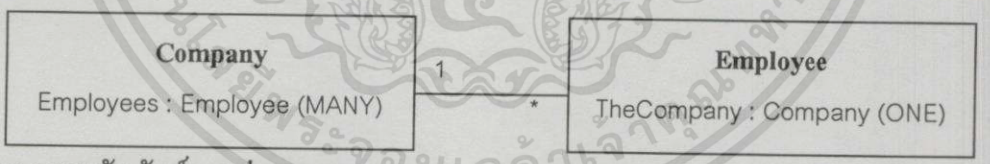
คลาสสามารถสืบทอดคุณสมบัติมาได้จากหลาย ๆ Superclass โดยใช้คีย์เวิร์ด SUPER เช่นเดียวกัน เมื่อทำการคอมไพล์คลาส หากพบว่าสมาชิกต่าง ๆ เช่น ชื่อเมธอดของ Superclass มีชื่อซ้ำกัน Caché จะกำหนดให้คลาสที่สืบทอดลำดับหลังถูกเรียกใช้งานก่อน ยกตัวอย่างเช่น คลาส X ทำการสืบทอดคุณสมบัติจากคลาส A, B และ C จะได้ว่า

```
CLASS X
{
        SUPER = A, B, C ;
}
```

ในกรณีนี้ค่าต่าง ๆ ของคลาส X จะสืบทอดคุณสมบัติและค่าต่าง ๆ จากคลาส A, B และ C ตามลำดับ (ดูจากลิสต์ก่อนหลัง) หากพบว่าคลาส B มีสมาชิกชื่อซ้ำกับคลาส A ที่ทำการสืบทอดคุณสมบัติมาแล้ว สมาชิกจากคลาส A จะถูกทับ (Override) ด้วยสมาชิกของคลาส B ทันที

6.2.8 Caché กับ Relationship

ความสัมพันธ์ (Relationship) เป็น Property ชนิดพิเศษซึ่งกำหนดความเกี่ยวข้องกันของออบเจ็กต์สองตัว เช่น คลาส Company มีความสัมพันธ์กับคลาส Employee ดังรูปที่ 6.6 ในกรณีนี้อาจมีออบเจ็กต์ Employee ศูนย์หรือมากกว่าหนึ่งตัวที่เกี่ยวข้องกับออบเจ็กต์ Company แต่ละตัว



รูปที่ 6.6 ความสัมพันธ์ระหว่างคลาส Company และคลาส Employee

สำหรับ Caché เวอร์ชัน 3.2.1.7 นี้ ความสัมพันธ์มีคุณลักษณะหลัก ๆ ดังนี้

- ความสัมพันธ์ถูกกำหนดขึ้นระหว่าง Persistent class 2 คลาสเท่านั้น
- ความสัมพันธ์เป็น bi-directional ซึ่งต้องกำหนดความสัมพันธ์ในทั้ง 2 ฝั่งคลาส
- สนับสนุนความสัมพันธ์แบบ One-to-Many

6.2.9 การกำหนดความสัมพันธ์

มีคีย์เวิร์ดที่เกี่ยวข้องอยู่ 3 ตัว ได้แก่

- TYPE เป็นชนิด (ชื่อคลาส) ของคลาสที่เกี่ยวข้อง และต้องเป็น Persistent Class

- INVERSE ชื่อของความสัมพันธ์ของอีกคลาสที่มีความสัมพันธ์ต่อกัน
- CARDINALITY มีค่าเป็น ONE หรือ MANY

จากรูปที่ 6.6 ได้ดังนี้

```
CLASS Company {
    SUPER = %Persistent ;
    PERSISTENT ;
    ATTRIBUTE Name { TYPE = %String ; }

    // a Company has MANY Employees
    RELATIONSHIP Employees { TYPE = Employee; INVERSE = the
        company; CARDINALITY = MANY;}
}

CLASS Employee {
    SUPER = %Persistent ;
    PERSISTENT ;
    ATTRIBUTE Name { TYPE = %String ; }
    ATTRIBUTE Title { TYPE = %String ; }

    // an Employee has ONE Company
    RELATIONSHIP TheCompany { TYPE = Company; INVERSE =
        Employees; CARDINALITY = ONE;}
}
```

6.2.10 Caché กับ Application

ในส่วนของการ Implementing the Application นั้น Caché ได้แบ่งออกเป็นขั้นตอนใหญ่ ดังนี้

- Setting Up a Namespace เป็นการกำหนดพื้นที่ในการจัดเก็บข้อมูลของ Application โดยที่จะเป็นการกำหนดทาง Logical เท่านั้น
- Creating the Application and its Classes สามารถสร้างคลาสโดยใช้ Caché Object Architecture สามารถกำหนดชื่อ คำอธิบายและชนิดของคลาสได้ สำหรับ Caché จะมี 3 คลาสที่ใช้ประจำ คือ Persistent Class (ออบเจ็กต์ที่ได้จะเป็นอิสระ เก็บข้อมูลได้เอง) , Embeddable Class (ออบเจ็กต์ของคลาสนี้จะอยู่ในออบเจ็กต์ของคลาสอื่นอีกที) และสุดท้ายคือ Data type Class เป็นคลาสที่ไม่ใช้เก็บข้อมูลแต่มีไว้เพื่อควบคุมการทำงานของแอพพลิเคชัน
- Properties เป็นการระบุข้อมูลของคลาสที่ได้สร้างไว้แล้วซึ่งจะต้องระบุชนิดของข้อมูลและเงื่อนไขต่าง ๆ ของ Property ไว้ด้วยเช่น unique , indexed , require เป็นต้น ขั้นตอนนี้ใช้ Caché Object Architecture ช่วยในการทำงาน

- Method หลังจากกำหนดค่า Properties ของคลาสแล้ว จะต้องทำการกำหนดเมธอดของคลาสต่าง ๆ ขึ้นมาโดยใช้ภาษา Caché ObjectScript เขียนขึ้นมาว่าจะให้คลาสที่เรากำหนดมานั้นสามารถทำงานหรือมีพฤติกรรมอย่างไรบ้าง
- Creating Query สร้างขึ้นเพื่อแสดงข้อมูลที่มีอยู่ในฐานข้อมูลของแอพพลิเคชันแทนที่จะเข้าไปดูข้อมูลโดยตรงกับออบเจ็กต์แต่ละตัวผ่านทาง OID โดยใช้ Caché SQL Query หรือ Caché ObjectScript

ในส่วนของการสร้าง Appilcation Front-end สำหรับผู้ใช้นั้น สามารถทำได้โดยใช้ ActiveX, C++, Delphi, PowerBuilder ซึ่งจะติดต่อกับ Caché ผ่านทาง Caché ObjectServer แต่วิธีที่นิยมและใช้งานง่ายที่สุดคือการสร้างผ่านทาง Visual Basic

6.3 ฐานข้อมูล Meta

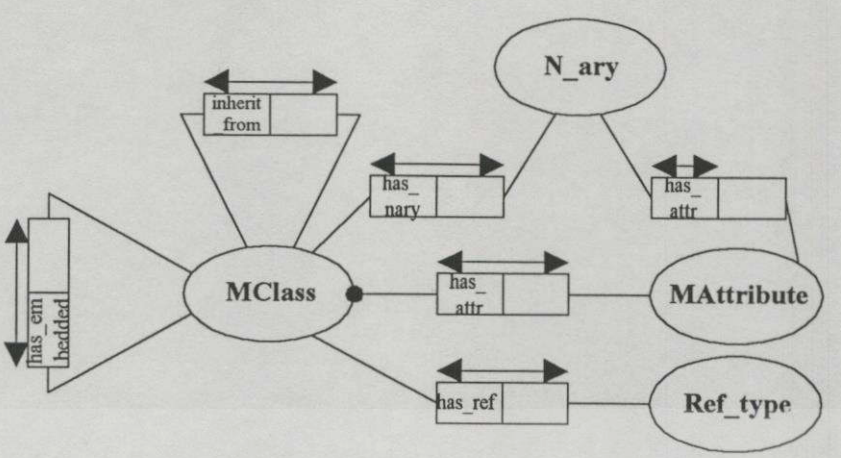
ฐานข้อมูล Meta เป็นฐานข้อมูลที่จัดเก็บ TOONIAM ที่ผู้ใช้สร้าง โดยจะจัดเก็บรายละเอียดต่าง ๆ ของคลาส และความสัมพันธ์ของคลาส รวมถึงจัดเก็บการถ่ายทอดคุณสมบัติของคลาสต่าง ๆ อีกด้วย หรือกล่าวอีกนัยหนึ่งได้ว่า ฐานข้อมูล Meta นี้มีหน้าที่จัดเก็บทุกสิ่งทุกอย่างของ TOONIAM ที่ผู้ใช้ได้ออกแบบขึ้น ซึ่งฐานข้อมูล Meta นี้ ได้รับการออกแบบโดยใช้ OONIAM ซึ่งมีรายละเอียดดังนี้

6.3.1 โมเดล OONIAM ของฐานข้อมูล Meta

โมเดล OONIAM ของฐานข้อมูลแบ่งออกเป็น 2 ส่วนคือ โครงร่างหลักและโครงร่างย่อย ดังนี้

6.3.1.1 โครงร่างหลัก (Main Schema)

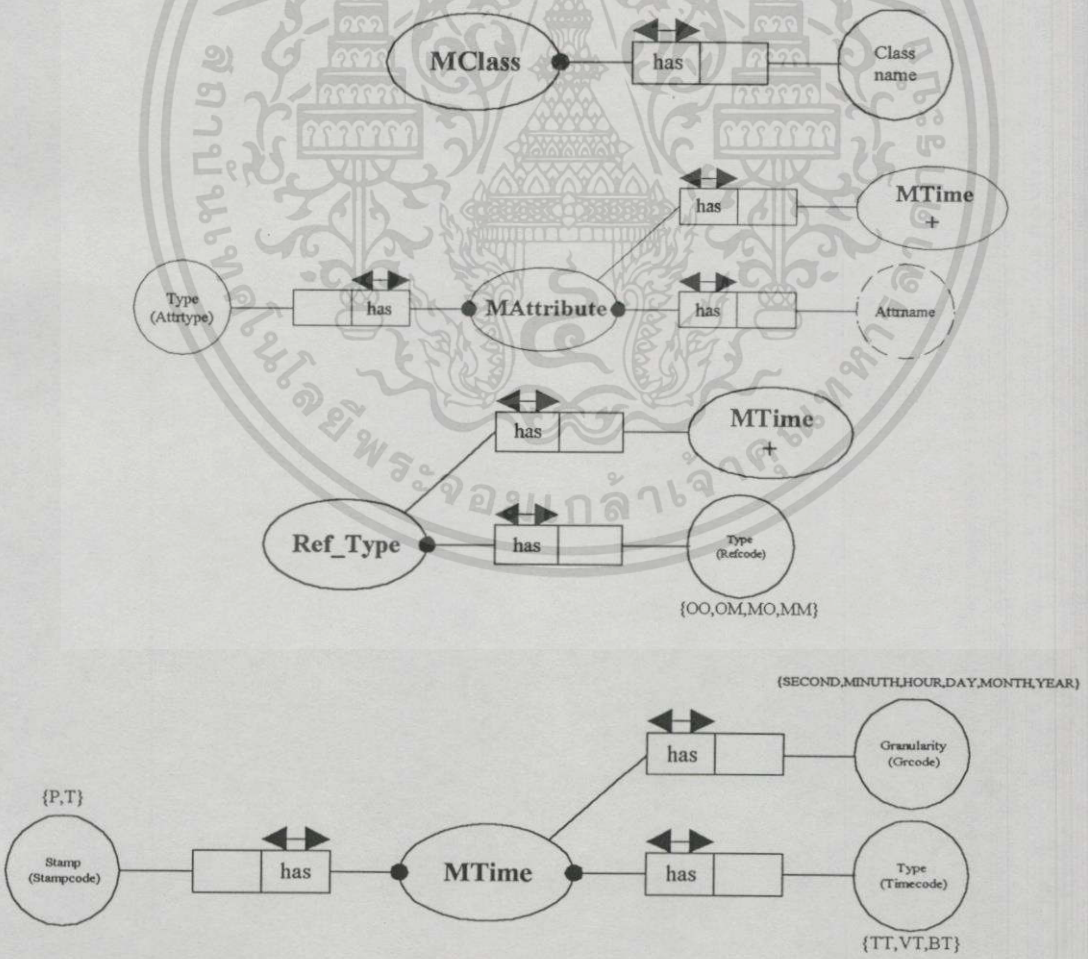
โครงร่างหลักของฐานข้อมูล Meta มีดังรูปที่ 6.7



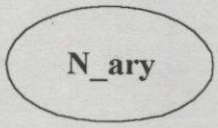
รูปที่ 6.7 โครงร่างหลักของฐานข้อมูล Meta

6.3.1.2 โครงร่างย่อย (Sub Schema)

โครงร่างย่อยแสดงในรูปที่ 6.8



รูปที่ 6.8 โครงร่างย่อยของฐานข้อมูล Meta



รูปที่ 6.8 โครงร่างย่อยของฐานข้อมูล Meta (ต่อ)

6.3.2 ODL ของฐานข้อมูล Meta

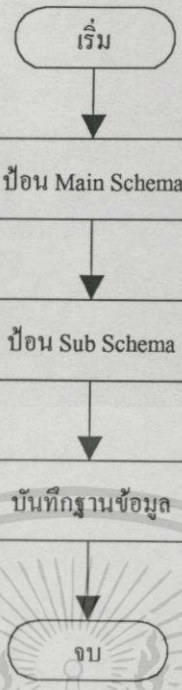
จากรูปที่ 6.6 และ 6.7 สามารถแปลงให้เป็นภาษา ODL ได้ดังนี้

```
typedef struct Mtime_Struct {
        string Timecode,
        string Stampcode,
        string Grcode
} ;

interface  MClass
(extent MClasss)
{
        attribute string Classname ;

        relationship set<MAttribute> Has_attr inverse
                        MAttribute :: Attr_of ;
        relationship set<MClass> Inherit_from  inverse
                        MClass :: Super_of;
        relationship set<MClass> Super_of  inverse
                        MClass :: Inherit_from ;
        relationship set<MClass> Has_embedded  inverse
                        MClass :: Embed_of;
        relationship set<MClass> Embed_of inverse
                        MClass :: Has_embedded;
        relationship set<Ref_type>  Has_ref  inverse
                        Ref_type :: Ref_of ;
        relationship set<N_ary> Has_nary  inverse
                        N_ary :: Nary_of ;

} ;
```

```
interface MAttribute
(extent MAttributes)
{
        attribute string Attrname ;
        attribute string Attrtype ;
        attribute MTime_Struct Mtime ;

        relationship set<MClass> Attr_of inverse
                        MClass :: Has_attr ;
        relationship  set <N_ary> Attr_of inverse
                        N_ary :: Has_attr  ;
} ;

interface  Ref_Type
(extent Ref_Types)
{
        attribute string Refcode ;
        attribute MTime_Struct Mtime ;

        relationship set <MClass>  Ref_of  inverse
                        MClass :: Has_ref ;
} ;

interface  N_ary
(extent N_arys)
{

        relationship set<MClass>  Nary_of  inverse
                        MClass :: Has_nary ;
        relationship  MAttribute   Has_attr inverse
                        MAttribute :: Attr_of ;
} ;
```

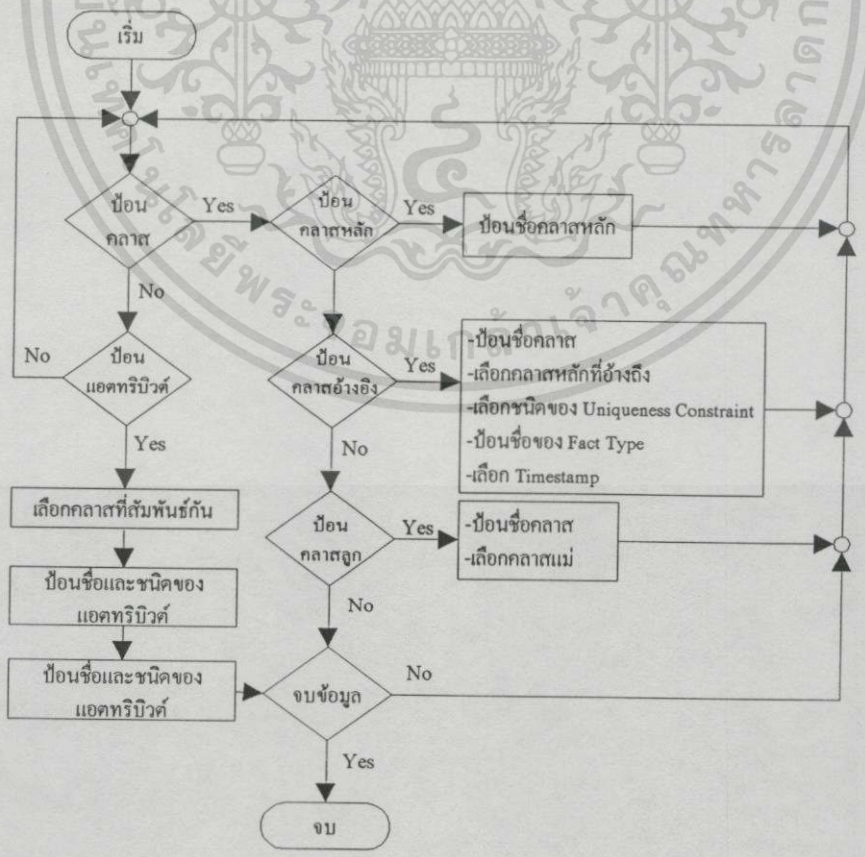
6.4 ขั้นตอนการประมวลผล

ขั้นตอนในการประมวลผลหลักแสดงรายละเอียดในรูปที่ 6.9

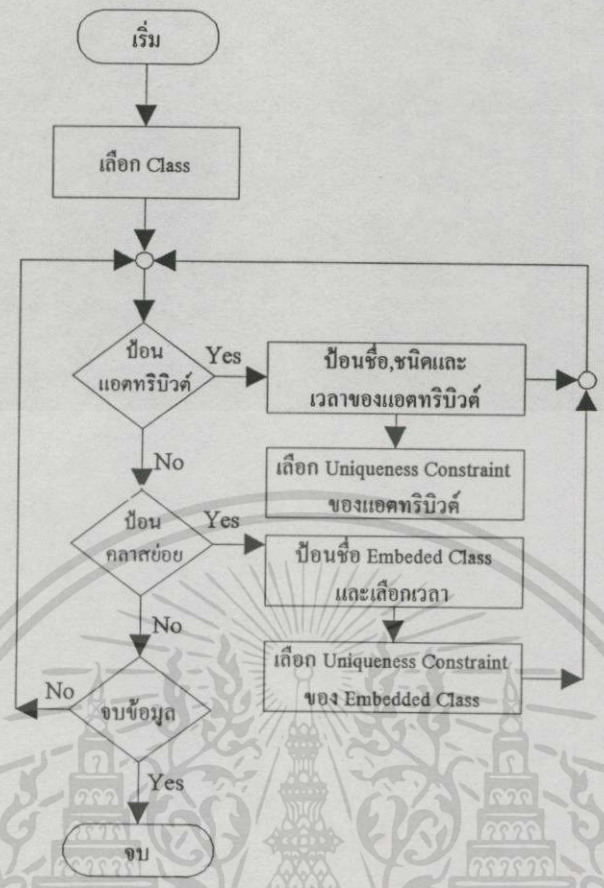


รูปที่ 6.9 ขั้นตอนหลักในการประมวลผล

จากรูปที่ 6.9 สามารถแยกแสดงขั้นตอนการป้อน Main Schema ได้ดังรูปที่ 6.10 ส่วนขั้นตอนการป้อน Sub Schema แสดงในรูปที่ 6.11



รูปที่ 6.10 การป้อน Main Schema



รูปที่ 6.11 การป้อน Sub Schema

6.5 การพัฒนาระบบสารสนเทศเชิงเวลา

ระบบสารสนเทศเชิงเวลาที่พัฒนาขึ้นนี้เป็นการเพิ่มเวลาเข้าไปในแอตทริบิวต์ที่ต้องการและเขียนโปรแกรมเพื่อเรียกค้นข้อมูลเชิงเวลาที่อยู่ในฐานข้อมูล สำหรับระบบสารสนเทศที่นำเสนอนี้คือ ระบบสารสนเทศของนักศึกษา ซึ่งมีข้อมูลที่เปลี่ยนแปลงตามเวลาดังมีรายละเอียดต่อไปนี้

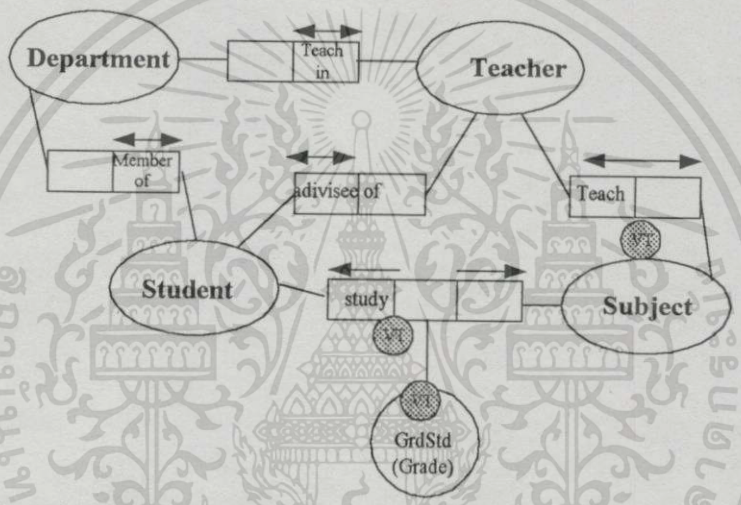
6.5.1 อุปกรณ์ที่ใช้

1. เครื่องคอมพิวเตอร์
   - หน่วยประมวลผลกลาง AMD-K6-2 ความเร็ว 400 เมกกะเฮริซ์
   - หน่วยความจำ 64 เมกกะไบต์
   - ฮาร์ดดิสก์ 6 กิกะไบต์
2. ระบบปฏิบัติการ Microsoft Windows 98
3. ระบบฐานข้อมูล Caché version 3.2.1.7

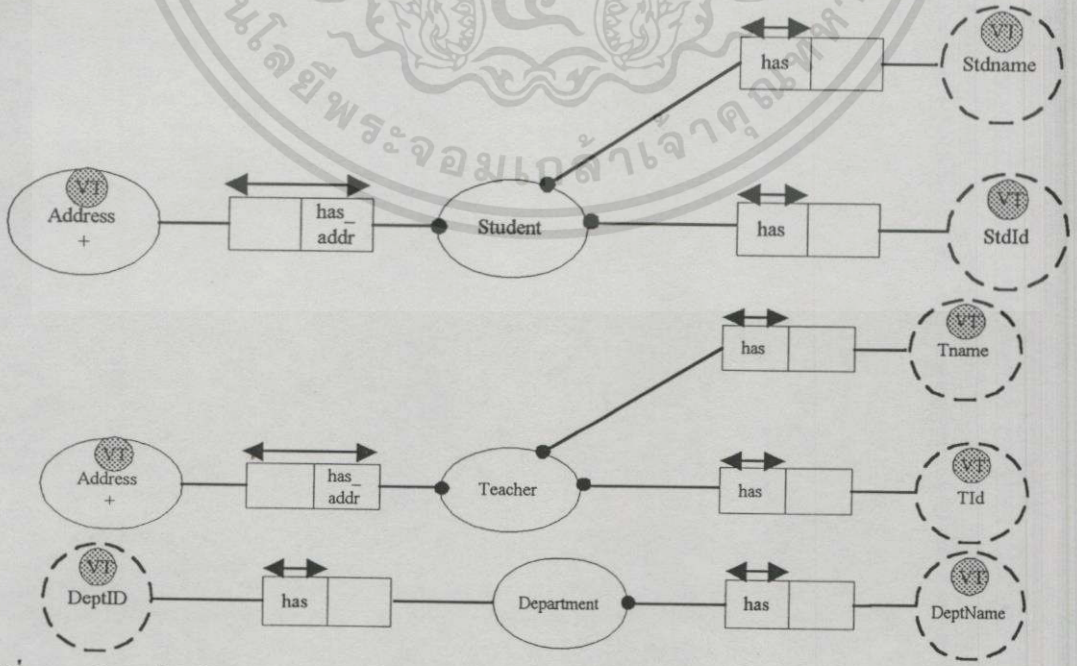
4. โปรแกรม Visual Basic version 6.0

6.5.2 การออกแบบฐานข้อมูล

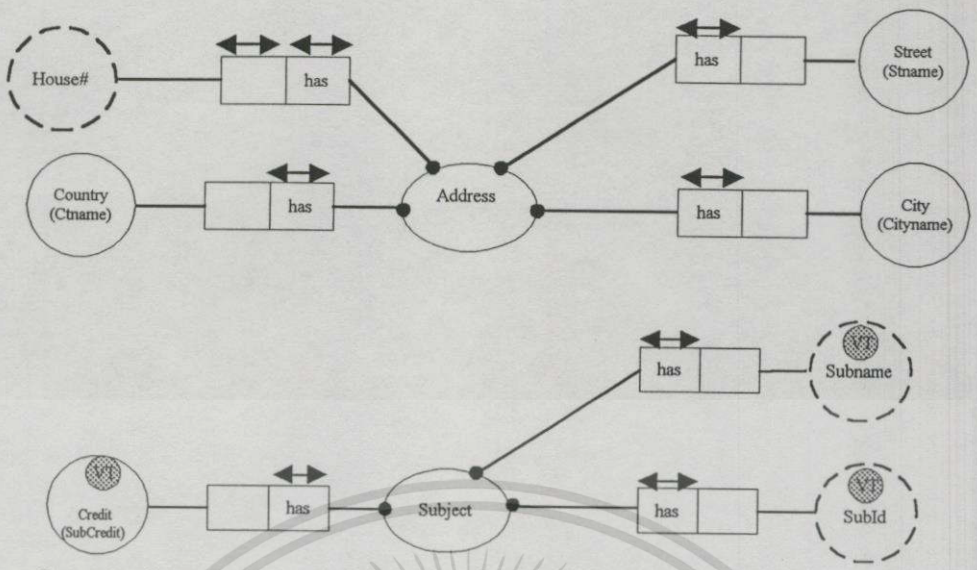
ระบบงานทะเบียนนักศึกษาเป็นระบบสารสนเทศหนึ่งที่ข้อมูลมีการเปลี่ยนแปลงตามเวลา เช่น รหัสนักศึกษาเปลี่ยนแปลงตามเวลา ชื่อวิชาเรียนเปลี่ยนแปลงไปเป็นต้น ดังนั้นหากต้องการให้ระบบสารสนเทศนักศึกษามีความสมบูรณ์ จำเป็นอย่างยิ่งที่จะต้องสร้างให้ระบบสารสนเทศนักศึกษาเป็นระบบสารสนเทศเชิงเวลา โดยออกแบบให้แอตทริบิวต์และความสัมพันธ์ระหว่างคลาสเป็นข้อมูลเชิงเวลา ซึ่งการออกแบบระบบสารสนเทศนักศึกษาดังกล่าวปรากฏในรูปที่ 6.12 และรูปที่ 6.13



รูปที่ 6.12 โครงร่างหลักของระบบสารสนเทศนักศึกษาเชิงเวลา



รูปที่ 6.13 โครงร่างย่อยของระบบสารสนเทศนักศึกษาเชิงเวลา



รูปที่ 6.13 โครงร่างย่อยของระบบสารสนเทศนักศึกษาเชิงเวลา (ต่อ)

จากรูปที่ 6.12 และ 6.13 สามารถแปลงเป็น ODL ได้ดังนี้

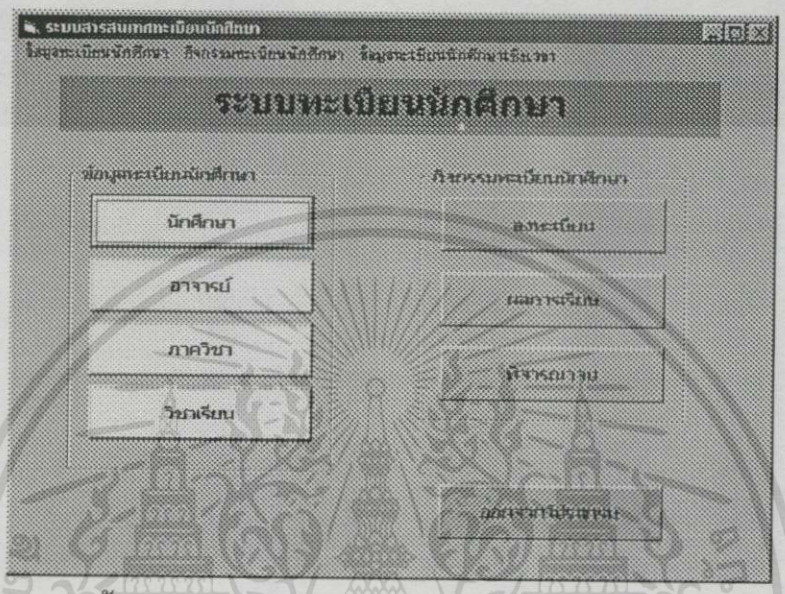
```
interface Student
( extent Students )
{
        typedef  Struct  AddressStruct {
                        string Houseid ,
                        string Ctname,
                        string Cityname,
                        string Stname ;}

        attribute set<struct AddressStruct{AddressStruct  value,
                        Period<DATE> vt_timestamp}> Address ;

        attribute set <struct StdnameStruct { string value,
                        Period<DATE>  vt_timestamp}>  Stdname ;
        attribute set<struct StdIdStruct{float value,
                        Period<DATE> vt_timestamp}> StdId ;


        relationship Department  member_of  inverse
                        Department:: student;
        relationship Teacher  adivisee_of inverse
                        Teacher :: advise
        relationship set< GrdStd >  study
                inverse  GrdStd:: student ;
};


interface Subject
(       extent Subjects)
{
        attribute set <struct SubnameStruct { string value,
                        Period<DATE>  vt_timestamp}>  Subname ;
        attribute set <struct SubIdStruct { string value,
```

```
                    Period<DATE> vt_timestamp}> SubId ;
        attribute set <struct SubcreditStruct { float value,
                    Period<DATE> vt_timestamp}> Subcredit ;

        relationship set< Teacher_teach_Subject_teacher> teacher inverse
                    Teacher_teach_Subject_teacher: i_teacher;
        relationship set< GrdStd > subject
                inverse GrdStd:: grsubject ;

};

interface Teacher
(       extent Teachers)
{
        typedef Struct AddressStruct {
                    string Houseid ,
                    string Ctname,
                    string Cityname,
                    string Stname ;}


        attribute set<struct AddressStruct{AddressStruct value,
                    Period<DATE> vt_timestamp}> Address ;


        attribute set <struct TnameStruct { string value,
                    Period<DATE> vt_timestamp}> Tname ;
        attribute set <struct TIdStruct { string value,
                    Period<DATE> vt_timestamp}> TId ;

        relationship set< Teacher_teach_Subject_teacher > teach inverse
                    Teacher_teach_Subject_teacher:: i_teach;
        relationship set<Student> advise inverse
                    Student:: advisee_of;
        relationship Department : teach_in inverse
                    Department:: teacher;


};


interface Teacher_teach_Subject_teacher
(       extent Teacher_teach_Subject_teachers )
{
        attribute Period<DATE> vt_timestamp ;
        relationship Subject : i_teacher inverse
                    Subject :: teacher
        relationship Teacher : i_teach inverse
                    Teacher :: teach
};

interface GrdStd
(extent GrdStds )
{
        attribute Period<DATE> vt_timestamp ;
        attribute float grade ;
        relationship Student student inverse Student ::study ;
        relationship Subject grsubject inverse Subject::subject
}
```

เมื่อได้ ODL เรียบร้อยแล้ว จึงนำ ODL ที่ได้ไปทำงานบนฐานข้อมูลเชิงวัตถุ จากนั้นผู้ใช้งานก็สามารถจัดเก็บและเรียกค้นข้อมูลที่เปลี่ยนแปลงตามเวลาได้

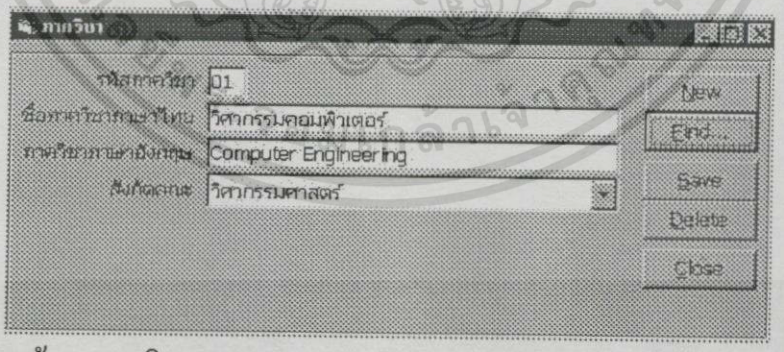
6.5.3 จอภาพต่าง ๆ ของระบบสารสนเทศนักศึกษา

จอภาพของโปรแกรมระบบสารสนเทศนักศึกษาเชิงเวลามีดังนี้



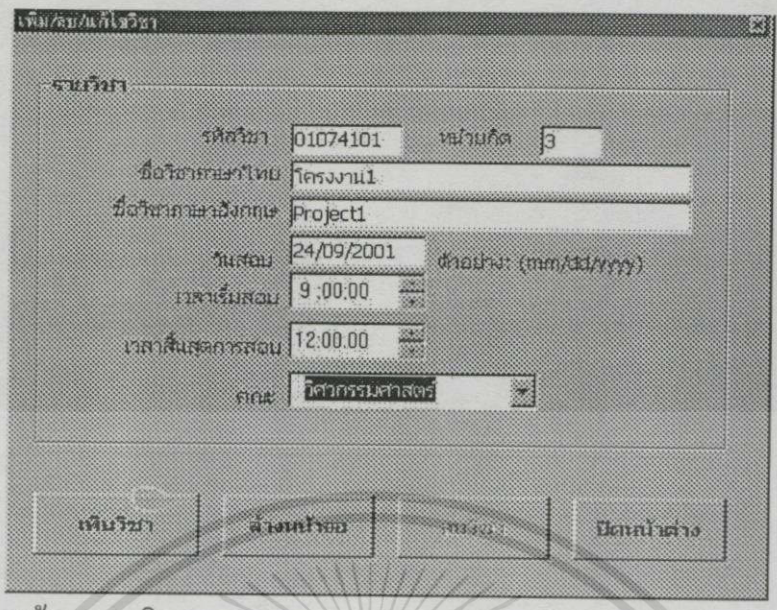
รูปที่ 6.14 จอภาพเมนูหลัก

รูปที่ 6.14 เป็นรูปของเมนูหลัก ผู้ใช้สามารถเลือกใช้งานระบบงานสารสนเทศเชิงเวลาได้อย่างอิสระ



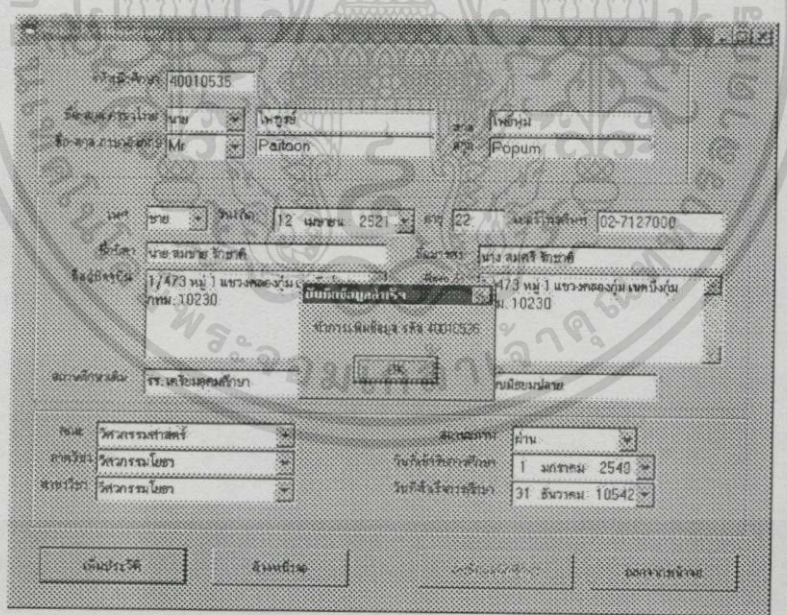
รูปที่ 6.15 จอภาพข้อมูลภาควิชา

จากรูปที่ 6.15 เป็นการแสดงข้อมูลภาควิชาที่ผู้ใช้งานสามารถแก้ไข เพิ่มเติม ตามความเป็นจริง



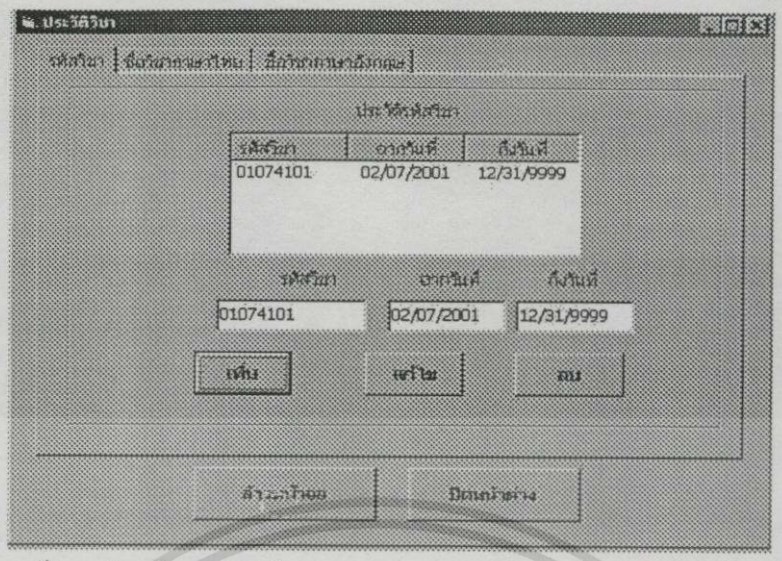
รูปที่ 6.16 จอภาพข้อมูลรายวิชา

รูปที่ 6.16 เป็นข้อมูลรายวิชาที่ผู้ใช้งานสามารถกำหนดวันสอบ ซึ่งข้อมูลรายวิชานี้เป็นข้อมูลที่เปลี่ยนแปลงตามเวลา โดยจะมีการจัดเก็บข้อมูลหลาย ๆ State และให้ผู้ใช้งานสืบค้นในภายหลัง



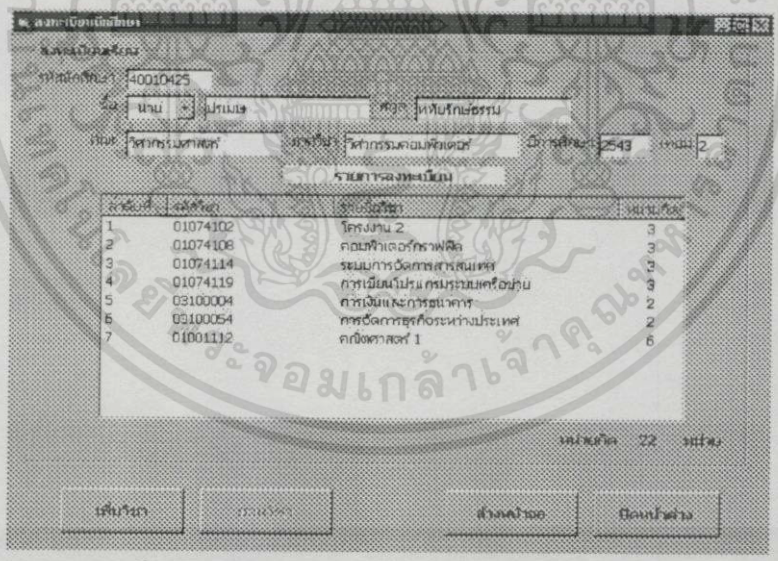
รูปที่ 6.17 จอภาพข้อมูลนักศึกษา

จากรูปที่ 6.17 เป็นข้อมูลนักศึกษาที่สามารถจัดเก็บข้อมูลของนักศึกษา โดยจัดเก็บข้อมูลหลาย ๆ State ซึ่งจะให้ผู้ใช้งานสืบค้นข้อมูลนั้น ๆ ได้



รูปที่ 6.18 จอภาพข้อมูลประวัติของรายวิชา

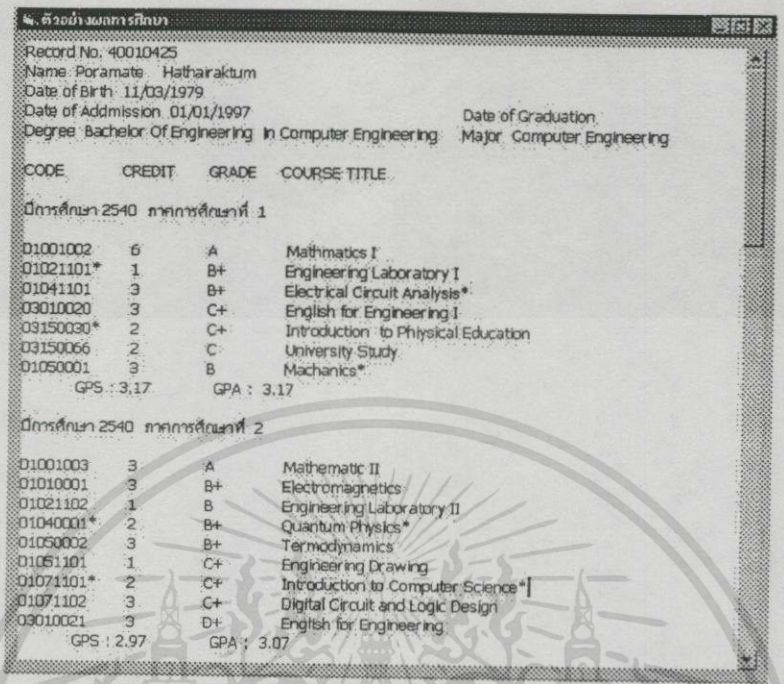
รูปที่ 6.18 แสดงประวัติของรายวิชา ซึ่งจะแสดงข้อมูลใน State ต่าง ๆ โดยข้อมูลที่สามารถเปลี่ยนแปลงได้นั้น จะจัดเก็บเป็นแบบ Collection ซึ่งผู้ใช้งานสามารถเรียกค้น และพิจารณาข้อมูลใน State ต่าง ๆ ได้



รูปที่ 6.19 ข้อมูลการลงทะเบียน

สำหรับรูปที่ 6.19 เป็นข้อมูลการลงทะเบียนเป็นการจัดเก็บประวัติของการลงทะเบียนของนักศึกษาแต่ละคน ซึ่งการลงทะเบียนนี้จะจัดเก็บเวลาของการลงทะเบียนไว้ด้วย ดังนั้นจึงสามารถเรียกค้นข้อมูลได้ตรงตามจริง กล่าวคือ หากมีการลงทะเบียนในวิชาใด ในช่วงเวลาใด เมื่อมีการสืบค้นในภายหลังก็จะได้ข้อมูลตามจริง ถึงแม้ว่ารายวิชา หรือผู้สอนจะเปลี่ยนแปลงไปแล้ว แต่สิ่งที่ได้ก็จะเป็นสิ่งที่เกิดขึ้นจริง ๆ

สำหรับการแสดงผลการเรียน จะแสดงดังรูปที่ 6.20



ตัวอย่างผลการศึกษา

Record No. 40010425
Name Poramate Hathairaktum
Date of Birth 11/03/1979
Date of Addmission 01/01/1997 Date of Graduation
Degree Bachelor Of Engineering In Computer Engineering Major Computer Engineering

CODE	CREDIT	GRADE	COURSE TITLE
ปีการศึกษา 2540 ภาคการศึกษาที่ 1			
01001002	6	A	Mathmatics I
01021101*	1	B+	Engineering Laboratory I
01041101	3	B+	Electrical Circuit Analysis*
03010020	3	C+	English for Engineering I
03150030*	2	C+	Introduction to Phiysical Education
03150066	2	C	University Study
01050001	3	B	Machanics*
GPS : 3.17		GPA : 3.17	
ปีการศึกษา 2540 ภาคการศึกษาที่ 2			
01001003	3	A	Mathematic II
01010001	3	B+	Electromagnetics
01021102	1	B	Engineering Laboratory II
01040001*	2	B+	Quantum Physics*
01050002	3	B+	Termodynamics
01051101	1	C+	Engineering Drawing
01071101*	2	C+	Introduction to Computer Science*
01071102	3	C+	Digital Circuit and Logic Design
03010021	3	D+	English for Engineering
GPS : 2.97		GPA : 3.07	

รูปที่ 6.20 ตัวอย่างผลการเรียน

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการวิจัย

ในงานวิจัยนี้ได้นำเสนอโมเดล TOONIAM ซึ่งเป็นโมเดลสำหรับออกแบบฐานข้อมูลเชิงเวลา โดย TOONIAM นี้ได้พัฒนามาจาก NIAM ซึ่งได้เพิ่มสัญลักษณ์บางอย่างเข้าไปในโครงสร้างของ NIAM เพื่อให้สามารถรองรับหลักการเชิงวัตถุและข้อมูลเชิงเวลาได้ สำหรับ TOONIAM นี้จะสามารถแก้ปัญหาการออกแบบฐานข้อมูลโดยใช้ UML ดังนี้

1. Class Diagram ของ UML (Unified Modeling Language) ไม่สามารถบอกได้ว่าส่วนใดเป็น Embedded Class และส่วนใดเป็น Class ที่ต้องสร้างใหม่
2. Class Diagram ไม่ได้นำเสนอ Function Dependency (FD) ซึ่งอาจทำให้แอตทริบิวต์ซ้ำซ้อนกันได้
3. Class Diagram ไม่สามารถรองรับหลักการเชิงเวลาได้

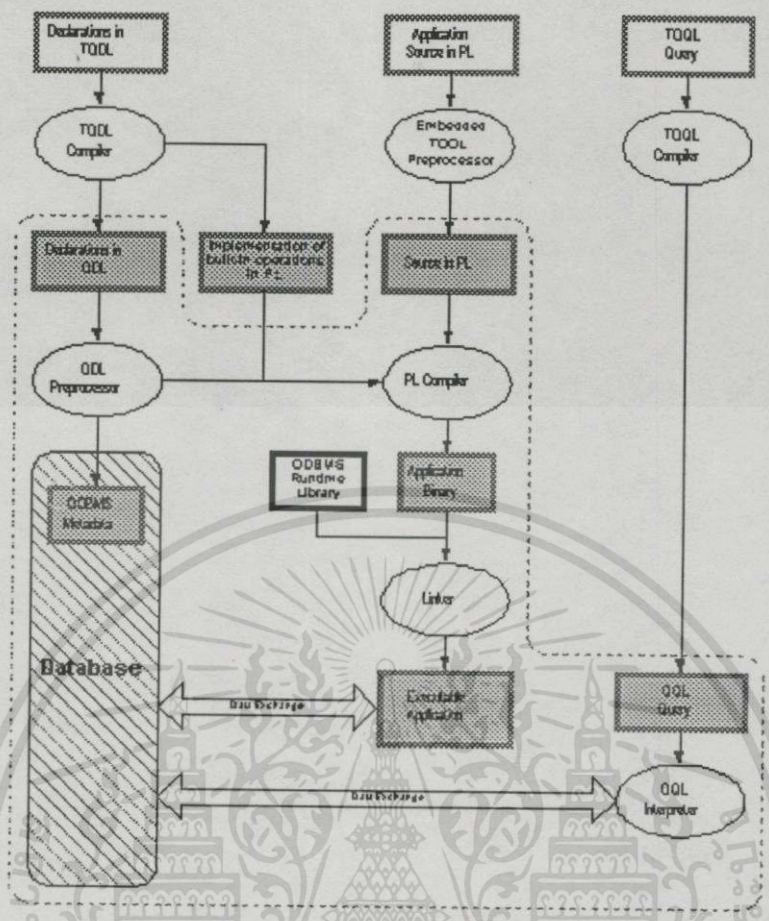
สำหรับข้อจำกัดทั้งสามประการของ UML ดังกล่าวนั้น TOONIAM จะสามารถแก้ปัญหาได้ทั้งสิ้น

นอกจากการนำเสนอโมเดล TOONIAM แล้ว วิทยานิพนธ์นี้ได้นำเสนออัลกอริธึมในการแปลงโมเดลดังกล่าวให้เป็นภาษานิยามเชิงวัตถุ(ODL) โดยภาษานิยามเชิงวัตถุนี้ผู้ใช้สามารถนำไปใช้กับฐานข้อมูลเชิงวัตถุเพื่อสร้างเป็นฐานข้อมูลเชิงเวลาได้เลย นอกจากนี้ผู้วิจัยยังได้สร้างโปรแกรมที่เป็นเครื่องมือในการออกแบบฐานข้อมูล TOONIAM โดยจะทำให้ผู้ใช้งานมีความสะดวกและคล่องตัวมากในการออกแบบ ผู้ใช้งานสามารถใช้โปรแกรมนี้สร้างโมเดลตามต้องการและแปลงโมเดลที่สร้างเป็นภาษานิยามเชิงวัตถุได้เลย

อนึ่งฐานข้อมูลเชิงวัตถุที่ใช้งานกันอยู่ในปัจจุบันนั้น โดยมากยังไม่รองรับภาษานิยามเชิงวัตถุ (ODL) ที่เป็นมาตรฐานของ ODMG ได้ครบถ้วนทุกคำสั่ง ดังนั้นหากนำ ODL ที่ได้นี้ไปทำงานบนฐานข้อมูลเชิงวัตถุทั่วไป อาจมีบางคำสั่งที่ไม่สามารถทำงานได้ แต่ในอนาคตอันใกล้ระบบฐานข้อมูลเชิงวัตถุทุกผลิตภัณฑ์จะรองรับภาษานิยามเชิงวัตถุนี้ได้ครบถ้วน

7.2 ข้อเสนอแนะ

ระบบการทำงานบนฐานข้อมูลเชิงเวลาโดยทั้งหมดนั้น ปรากฏอยู่ในรูปที่ 7.1



รูปที่ 7.1 ระบบงานทั้งหมดในฐานข้อมูลเชิงเวลา

จากรูปจะเห็นได้ว่า ในระบบงานฐานข้อมูลเชิงเวลานั้น มีการทำงานในหลาย ๆ ส่วน ในวิทยานิพนธ์นี้เป็นเพียงแค่ส่วนเดียวเท่านั้น กล่าวคือ ในวิทยานิพนธ์นี้ทำงานเฉพาะในส่วนซ้ายสุดของรูปที่ 7.1 เท่านั้นในส่วนอื่น ๆ ยังไม่ได้มีการกล่าวถึงเลย หรือกล่าวอีกนัยหนึ่งได้ว่า ในงานวิจัยนี้ได้นำเสนอ และใช้งานส่วนของ TODL เท่านั้น ยังไม่ได้กล่าวถึง TOQL และส่วนอื่น ๆ ในรูปที่ 7.1 เลย

ดังนั้นหากนักวิจัยท่านใดสนใจในหัวข้อนี้ การนำเสนอส่วนอื่น ๆ ไม่ว่าจะเป็นส่วนของ TOQL หรือ ส่วนของการแปลง TOQL ให้เป็น OQL ก็เป็นเรื่องที่น่าสนใจและสมควรได้รับการค้นคว้าวิจัยต่อไป นอกจากนี้ในระบบฐานข้อมูลเชิงเวลาควรจะมีเครื่องมือสำหรับทำงานภาษา SQL ทั้งในโหมด Interactive และ Embedded ด้วย

เอกสารอ้างอิง

[1] Andres Steiner, "Implementig Temporal Databases in Object Oriented System",TimeCenter Technical Report,1997,pp 1-14.

[2] Andres Steiner, Unpublic Doctor Theses "A Generalisation Appoach to Temporal Data Model and Their Implementations", Swiss Federal Institute of Technology Zurich,1998, pp 15-98.

[3] Christian S. Jensen, Richard Snodgrass,"Temporal Specialization and Generalization",IEEE Transaction on Knowledge and Data Engineering, Vol. 6, No. 6,December 1994,pp. 954-974.

[4] G.M. Nijssen , T.A. Halpin , " Conceptual schema and relational database design , a fact oriented approach " ,Prentice hall,inc., United States of America, 1989

[5] Jeffrey D. Ullman,Jennifer Widom, "A First Course in Database System ". Prentice-Hall International,Inc., United States of America, 1997

[6] Peter Rob, "Database System design, Implementation and Management (Third Edition)". Course Technology and the Open Book, United States of America,1997

[7] R. Elmasri , G. Wuu, "A Temporal Model and Query Language for EER Databases", In Proceedings of the 6th International Conference on Data Engineering, February 1990, pp. 76-83.

[8] R. Elmasri , S.B. Navathe,"Fundamentals of Database Systems (Second Edition)" . Addision-Wesley, Amsterdam, 1994

[9] Richard T. Snodgrass ,Temporal Databases Theory, Design, And Implementation , The Benjamin/Cummings Publishing Company,Inc, 1993, pp.10-99.

[10] Richard T. Snodgrass, Michael H. Bohlen, Christian S. Jensen, "Evaluation the completeness of TSQL2". Recent Advances in Temporal Database, 1995 pages 153-172.

[11] Richard T. Snodgrass, "Managing Temporal Data A Five-Part Series" TimeCenter Technical Report,1998,pp 1-35

[12] Richard T. Snodgrass,Cristian S. Jensen,"Temporal Data Management",IEEE Transaction on Knowledge and Data Engineering, Vol. 11 ,No. 1, January/February 1998,pp.36-44.

[13] Rick Snodgrass, "Special Series On Temporal Database",Database Programming Design,July 1998, pp 60-65 .

[14] Xiaoyang Sean Wang, Claudio Bettini, Alexander Brodsky, Sushil Jajodia, "Logical design for temporal databases with multiple granularities", ACM Transactions on Database Systems (TODS),Vol. 22, No. 2 , June 1997, pp. 115-170.

ประวัติผู้เขียน

ชื่อผู้เขียน	นาย บัณฑิต พัสยา
วันเดือนปีเกิด	28 พฤศจิกายน 2510
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตร์บัณฑิต สาขาคณิตศาสตร์ประยุกต์
สถานศึกษาที่สำเร็จการศึกษา	สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง
ปีการศึกษา	2534
อาชีพปัจจุบัน	อาจารย์ ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยี พระจอมเกล้า เจ้าคุณทหารลาดกระบัง
ผลงานทางวิชาการ	"A Temporal Object Oriented Conceptual Schema Model ", 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing , August 26-28, 2001, Victoria, B.C., Canada