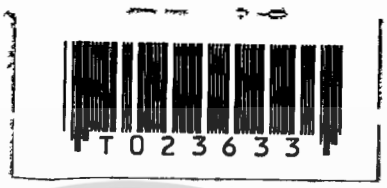
เครื่องมือออกแบบฐานข้อมูลเชิงวัตถุโดยใช้ NIAM++

AN OBJECT-ORIENTED DATABASE DESIGN TOOL USING NIAM++



พันตรีหญิง รัชนีพร ผดุงผล

MAJOR, RACHANEEPORN PHADUNGPOL

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2538

ISBN 974-621-346-6

AN OBJECT-ORIENTED DATABASE DESIGN TOOL USING NIAM^{++}

RACHANEEPORN PHADUNGPOL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE PROGRAM IN COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

GRADUATE SCHOOL

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1995

ISBN 974-621-346-6

หัวข้อวิทยานิพนธ์	เครื่องมือออกแบบฐานข้อมูลเชิงวัตถุโดยใช้ NIAM^{++}
นักศึกษา	พันตรีหญิงรัชนีพร ผดุงผล
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร.ศุภมิตร จิตตะยโศธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ และเทคโนโลยีสารสนเทศ
ภาควิชา	คณิตศาสตร์ และวิทยาการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2538

บทคัดย่อ

ในความเป็นจริงนั้น วัตถุสิ่งของต่าง ๆ ส่วนใหญ่จะมีโครงสร้างสลับซับซ้อน ซึ่งการการออกแบบฐานข้อมูลเชิงสัมพันธ์ (RDB : Relational Database) ไม่เหมาะสมกับข้อมูลที่มีโครงสร้างสลับซับซ้อน (Complex Object) จึงได้นำความคิดของ ออบเจกต์ (Object) มาใช้เพื่อแก้ไขข้อบกพร่องดังกล่าว และได้ใช้ทฤษฎีของ ไนแอม (NIAM : Nijssen's Information Analysis Methodology) ซึ่งเป็นนิยามข้อมูลระดับแนวความคิดที่เหมาะสมเป็นพื้นฐานในการออกแบบฐานข้อมูลเชิงวัตถุ แต่เนื่องจาก ไนแอม ไม่ได้กำหนดสัญลักษณ์ที่ใช้แทนความสัมพันธ์แบบ Aggregation ไว้ ดังนั้นจึงได้เพิ่มเติมสัญลักษณ์บางอย่างเข้าไป เพื่อที่จะแทนสภาพความเป็นจริงให้ได้มากที่สุดเท่าที่จะมากได้ จึงเรียกว่า ไนแอม$^{++}$ (NIAM^{++}) พร้อมทั้งเสนอ แอลกอริธึม (Algorithm) และเครื่องมือ (Tools) ในการแปลง ไนแอม$^{++}$ ไปเป็น Relational Schema และ Object-oriented Schema

Thesis Title	An Object-Oriented Database Design Tool Using NIAM^{++}
Student	Maj. Rachaneeporn Phadungpol
Thesis Advisor	Asst.Prof.Dr. Suphamit Chittayasotorn
Level of Study	Master of Science program in Computer Science and Information Technology
Department	Mathematic and Computer Science Department Faculty of Science King Mongkut's Institute of Technology Ladkrabang
Year	1995

ABSTRACT

Nijssen's Information Analysis Methodology (NIAM) is selected to be the right Conceptual Schema Model for designing the relational database. NIAM does not define the sign for Aggregation relation. But most objects are Complex Object in Aggregation relation form. Some signs are added to NIAM for representing the relation in the real world. The new methodology is called NIAM^{++}. Algorithm and tools for mapping NIAM^{++} to Relational Schema and Object-Oriented Schema are also presented.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีก็เพราะ ได้รับความเมตตาจาก ผู้ช่วยศาสตราจารย์ ดร.ศุภมิตร จิตตะยโศธร ที่ได้ให้ความกรุณาสละเวลามา กรมการสนเทศทหาร เพื่อแนะนำแก่ผู้วิจัยตลอดมา ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ พลตรี มนตรี ศุภาพร อดีตเจ้ากรมการสนเทศทหารกองบัญชาการทหารสูงสุด พลตรี ทวีศักดิ์ เรืองพงษ์ เจ้ากรมการสนเทศทหาร กองบัญชาการทหารสูงสุด และ พันเอก ประสงค์ ปานเจริญ รองผู้อำนวยการสถาบันคอมพิวเตอร์ กรมการสนเทศทหาร กองบัญชาการทหารสูงสุด ผู้ก่อตั้ง โครงการร่วมมือระดับปริญญาโทระหว่าง กองบัญชาการทหารสูงสุด กับสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และ อำนวยความสะดวกในการศึกษาและ การใช้ระบบคอมพิวเตอร์ในการทำวิจัย

ขอขอบคุณ นาวาเอกหญิง สมพร กาญจนโกศล ที่กรุณาให้คำแนะนำในการทำวิทยานิพนธ์ด้วยดีตลอดมา

ขอขอบคุณ เรืออากาศตรีหญิง ปราณี แคลลา และ จ่าสิบเอกหญิง วันสนา สุรโยธิน และเพื่อน ๆ ทุกคน ที่ให้ความช่วยเหลือในการพิมพ์วิทยานิพนธ์เป็นอย่างดี

พันตรีหญิง รัชนีพร ผดุงผล

สารบัญ

หน้า

บทที่ หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญภาพ

รูปที่ หน้า

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การออกแบบฐานข้อมูล นับได้ว่าเป็นงานที่มีความสลับซับซ้อนมากพอสมควร ซึ่งผู้ออกแบบมักต้องการที่จะสร้างให้ระบบมีประโยชน์มีประสิทธิภาพและสมบูรณ์มากที่สุด ซึ่งขั้นตอนที่สำคัญและเป็นขั้นตอนแรกของการออกแบบฐานข้อมูล คือ การกำหนดโครงสร้างของฐานข้อมูล โดยอาศัย แบบจำลองข้อมูล (Data Model)

แบบจำลองข้อมูลที่ใช้ในการออกแบบมีอยู่ด้วยกันหลายแบบ เช่น ER-Model และ EER-Model ฯลฯ แต่แบบจำลองข้อมูลดังกล่าวก็ยังมีความคลุมเครือไม่ชัดเจนในการนิยามออบเจกต์ (Object) คุณสมบัติของออบเจกต์ (Attribute) รวมทั้งความสัมพันธ์ (Relationship) ก็ไม่ชัดเจนเช่นกัน

ดังนั้นจึงจำเป็นต้องพิจารณาหา แบบจำลองข้อมูลที่เหมาะสม ในการออกแบบฐานข้อมูล และหาเครื่องมือ (Tool) ช่วยในการออกแบบฐานข้อมูลด้วย เพื่อให้ได้ระบบที่สมบูรณ์และมีประสิทธิภาพ

1.2 วัตถุประสงค์

การทำวิจัยนี้มีวัตถุประสงค์ 3 ประการ คือ

ประการที่หนึ่ง ศึกษาและพัฒนาแบบจำลองข้อมูลขึ้นมาใหม่ เพื่อให้สามารถแทนสภาพ ที่เป็นจริงให้ได้มากที่สุด

ประการที่สอง สร้างแอลกอริธึม (Algorithm) ในการแปลงแบบจำลองข้อมูลที่พัฒนาขึ้นมาให้เป็นโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ และ โครงสร้างฐานข้อมูลแบบเชิงวัตถุ

ประการที่สาม สร้างเครื่องมือ (Tool) ใช้ช่วยในการออกแบบ โครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ และโครงสร้างฐานข้อมูลแบบเชิงวัตถุ

1.3 แนวความคิดที่เกี่ยวข้องกับการวิจัย

ในปี 1982, ISO [2] (The International Standard Organization) เสนอสถาปัตยกรรมฐานข้อมูลแบบ 3 ระดับ (Three Schema Architecture) ขึ้นเป็นมาตรฐานของฐานข้อมูล ซึ่งนิยามของฐานข้อมูลถูกแบ่งออกเป็น 3 ระดับ คือ

1. External Schema จะอธิบาย User View ของ Conceptual Schema

2. Conceptual Schema ใช้แสดงความสัมพันธ์ระหว่างข้อมูลโดยอาศัยแบบจำลองข้อมูล เป็นตัวแสดงความสัมพันธ์ระหว่างข้อมูล

3. Internal Schema ใช้กำหนดว่าจะแทนข้อมูลลงในหน่วยความจำได้อย่างไร และการเข้าถึงข้อมูลเป็นอย่างไร

การจำลองข้อมูล (Data Modeling) เป็นขั้นตอนแรกในการออกแบบฐานข้อมูล คือ เป็นช่วงการนิยามโครงสร้างของฐานข้อมูล ซึ่งการจำลองข้อมูลมีขั้นตอนที่สำคัญดังนี้

1. ขั้นตอนการออกแบบแนวความคิด (A Conceptual Design phase) จะเกี่ยวกับการออกแบบ Conceptual Schema ซึ่งเป็น นามธรรม (Abstract) ของสภาพที่เป็นจริง (Real-world)

2. ขั้นตอนการออกแบบโครงสร้างข้อมูล (The Design Of A Logical Data Structure) เป็นการแทน Schema ซึ่งจะถูกแปลงส่งไปบน Actual Implementation

ขั้นตอนการออกแบบแนวความคิด

ในขั้นตอนนี้ จะคำนึงถึงการวิเคราะห์ข้อมูลข่าวสารที่ต้องการใช้ในระบบ เตรียมข้อกำหนด (ลักษณะ) และโครงสร้างของแบบจำลองข้อมูลในระดับสูง

หน้าที่ที่สำคัญของแบบจำลองข้อมูล

1. แบบจำลองข้อมูล จะต้องเก็บรายละเอียดของทั้ง คุณลักษณะที่คงที่ (Static Aspect) และคุณลักษณะที่เปลี่ยนแปลง (Dynamic Aspect) ดังนี้

- คุณลักษณะที่คงที่ จะคำนึงถึง นามธรรมของออบเจกต์ และ แนวความคิดที่เกี่ยวข้อง และ คุณสมบัติของออบเจกต์ (หรือ Attribute) พร้อมทั้งความสัมพันธ์ภายในของออบเจกต์ด้วย

- คุณลักษณะที่เปลี่ยนแปลง จะคำนึงถึงการกระทำบนออบเจกต์ และ คุณสมบัติที่ต้องการโดย Transaction และ Query (นั่นคือ Dynamic Property จำลองรูปแบบของ Behavior ของระบบ และ Task ที่เป็นรูปแบบของ การกระทำเฉพาะบนฐานข้อมูลนั้น)

2. แบบจำลองข้อมูล จะกำหนดกฎควบคุมความถูกต้อง (Integrity Rule) ที่จะต้องติดกับออบเจกต์ และ การกระทำ (Operation) บนออบเจกต์ ดังนี้

- กฎควบคุมความถูกต้อง จะเกี่ยวข้องกับ คุณสมบัติที่คงที่และคุณสมบัติที่เปลี่ยนแปลง เช่น การป้องกัน การแก้ไข (Update) ที่จะกระทำผิดใน สภาวะ (State) ของ ออบเจกต์

- การควบคุมความถูกต้องของความหมาย (Sematic Integrity Contraint) จะขึ้นอยู่กับแบบจำลองข้อมูลในระดับสูง

ในปี 1976 CHEN [5] ได้เสนอ Entity-Relationship Model (ER-Model) ใช้สำหรับออกแบบฐานข้อมูล ซึ่ง Entity-Relationship Model นี้ มีพื้นฐานอยู่บน Real-World ซึ่งประกอบด้วย เซ็ตของออบเจกต์ ที่เรียกว่า เอนทิตี และ ความสัมพันธ์ระหว่างออบเจกต์เหล่านั้น

เอนทิตี	คือ ออบเจกต์ที่มีอยู่และแตกต่างจาก ออบเจกต์อื่น อย่างเห็นได้ชัด
เอนทิตี เซ็ต	คือ เซ็ตของเอนทิตีที่เหมือนกัน
แอตตริบิวต์	คือ คุณสมบัติของ เอนทิตี
โดเมน	คือ ค่าที่อนุญาตให้เป็นไปได้ของ Attribute
ความสัมพันธ์	คือ ความสัมพันธ์ระหว่าง Entity

ข้อบังคับของความสัมพันธ์ (Constraints on Relatioship Type) มี 2 ชนิด

1. คาร์ดินัลลิตี เรโช (Cardinality Ratio) เป็นข้อบังคับ ที่เจาะจงหรือกำหนดจำนวนของ ความสัมพันธ์ที่เอนทิตีสามารถเข้าไปมีส่วนร่วมได้ คาร์ดินัลลิตี สำหรับ ความสัมพันธ์แบบไบนารี (Binary Relationship) คือ one-to-one (1:1), one-to-many (1:N), และ many-to-many (M:N)

2. พาร์ทิชิเพชัน (Participation) เป็นข้อบังคับที่กำหนด Existence ของเอนทิตีที่ขึ้นอยู่กับความสัมพันธ์ของมันกับเอนทิตีอื่นผ่านทางความสัมพันธ์ พาร์ทิชิเพชัน มี 2 ชนิด คือ Total และ Partial

Logical Structure ของ ER-Model สามารถใช้ รูปภาพแสดงได้เรียกว่า ER-Diagram

ต่อมามีนักวิจัยเสนอ Extended Entity Relationship Model (EER-Model) โดยการนำ ER-Model มาเป็นพื้นฐานและเพิ่มแนวความคิดเกี่ยวกับ ซับคลาส (Subclass) และ ซุปเปอร์คลาส(Superclass) เพื่อให้มีความสัมพันธ์ใกล้เคียงกับ แอตตริบิวต์อินเฮอริทัน (Attribute Inheritance) อย่างไรก็ตาม EER-Model ยังมีสิ่งที่ตลุมเครืออยู่ และต้องทำการแก้ไขโดยวิธี นอร์มอไลเซชัน (Normalization)

ในปี 1986 TEOREY [3] ได้เสนอ A Logical Design Methodology for Relational Database Using the Extended Entity-Relationship Model (LRDM) มีการใช้ทั้ง EER-Model และ Relational Model ซึ่งเป็นการนำข้อดีของ EER-Model คือ ง่ายในการใช้ และ โครงสร้างของ Relational Model มารวมกัน โดยมีขั้นตอนการทำ 3 ขั้นตอน คือ

1. วิเคราะห์ความต้องการแล้วใช้ EER-Model

2. ทำการแปลงจาก EER-Model ไปเป็นรีเลชัน

3. ทำการ นอร์มอไลเซชัน รีเลชัน

ในปี 1988 BLAHA [4] ได้เสนอ Relational Database Design Using an Object-Oriented Methodology (OMT) ซึ่งมีการใช้ทั้ง ER-Model และ LRDM เช่น การนำข้อดีของ ER-Model ที่ว่าง่าย และเข้าใจง่าย มี ER-Diagram ที่สร้างความคิดเกี่ยวกับการติดต่อสื่อสาร แต่ ER-Model ขาด โครงสร้างย่อย (Substructure) สำหรับเอนทิตี และ ความสัมพันธ์ ส่วน LRDM จะสนับสนุนแนวความคิดพื้นฐาน 4 ประการ คือ

1. เอนทิตี

2. เจนเนอรัลไลเซชัน

3. แอกกริเกชัน

4. แอสโซซิเอชัน

ซึ่งเป็นที่ยอมรับกันว่า LRDM นั้นมีความสามารถสูงกว่า ER-Model หลักการของ OMT แบ่งเป็น 3 ระดับ

1. ระดับสูง จะเน้นบนพื้นฐานโครงสร้างของข้อมูล และเป็น สัญลักษณ์รูปภาพ ที่พัฒนามาสำหรับ OOP (Object-Oriented Programming)

2. ระดับกลาง จะทำการแปลง ออบเจกต์ ไปยัง ตาราง, โดเมน และ คีย์

3. ระดับต่ำ เป็นภาษาที่ใช้นิยามข้อมูล (Data Definition Language) ของ DBMS ที่ต้องการ ในระดับต่ำนี้จะบรรจุด้วย คำสั่งของ DBMS จริงๆ ที่ใช้สร้างตาราง, แอตตริบิวต์ และ อินเดกซ์ และจะพิจารณาในรายละเอียดของ DBMS โดยเฉพาะเป็นการแทนตารางเข้าไปใน แฟ้มฐานข้อมูล (Database File)

ในปี 1977 NIJSSEN [3] ได้เสนอ NIJSSEN'S Information Analysis Methodology (NIAM) ซึ่งเป็นแบบจำลองข้อมูลระดับแนวความคิด ที่มีพื้นฐานมาจากวิธีความสัมพันธ์แบบไบนารี โดยแนวความคิดพื้นฐานประกอบด้วย อีลิเมนทารีแฟคท์(Elementary Fact), เอนทิตี และ ซับไทพ์ (Subtype), เลเบล (Label), เนสท์แฟคท์คอนสเทรน (Nested Fact Constraint) หรือ แวลิเดชันรูล (Validation Rule)

1. เอนทิตี คือ นามธรรม หรือ เอนทิตีจริง ๆ เช่น Person, Male, Sex,Hobby

2. เลเบล คือ ตัวที่ใช้บ่งบอกความแตกต่างของเอนทิตี ภายในเอนทิตีไทป์ตัวนั้น แต่ละสมาชิกของเอนทิตีในโครงสร้างตามแนวความคิด (Conceptual Schema) จะต้องมีอย่าง

น้อย 1 ชื่อ ที่ไม่ซ้ำกัน (Unique Name) คือ คีย์นั่นเอง

3. ออบเจกต์ คือ ชื่อทั่วๆไป ที่ใช้เรียก เอนทิตี และ เลเบล

4. อีลีเมนทารีแฟคท์ ใช้แทนความสัมพันธ์ระหว่าง ออบเจกต์ ตั้งแต่ 2 ตัวขึ้นไป

5. ซับไทพ์ ใช้ในการกำหนด ลำดับชั้นที่ซับซ้อนของ เอนทิตี และ ซับไทพ์มีการถ่ายทอดแฟคท์ (Fact) จาก ซุปเปอร์ไทพ์ (Supertype) ด้วย

6. เนสท์แฟคท์คอนสเทรน จะถูกทำเหมือนดั่งเป็น เอนทิตีไทพ์ เมื่อมันเกี่ยวข้องกับอีลีเมนทารีแฟคท์ตัวอื่น ซึ่ง เนสท์แฟคท์ (Nested Fact) นี้จะเหมือนกับเป็น เอนทิตีไทพ์ธรรมดา

7. คอนสเทรน เป็นเงื่อนไขที่จะทำให้แน่ใจว่าฐานข้อมูลถูกต้องและสมบูรณ์ตลอดเวลา

จากแบบจำลองข้อมูลต่างๆ ที่กล่าวไปแล้วนั้น ส่วนใหญ่มีพื้นฐานมาจาก ER-Model ยกเว้น ไนอม และแบบจำลองข้อมูลที่มีพื้นฐานจาก ER-Model นั้น มักมีความคลุมเคลือในการนิยาม ออบเจกต์ และ คุณสมบัติของออบเจกต์รวมทั้งความสัมพันธ์ก็ไม่ชัดเจน เช่นความสัมพันธ์ที่เป็น n-ary ซึ่งมีความจำเป็นในการ แทนความสัมพันธ์ในสภาพความเป็นจริง (Real-World) และแต่ละวิธี ต้องใช้ นอร์มอไลเซชันมาช่วย เพื่อให้ได้รูปแบบของโครงสร้างข้อมูลที่ดีขึ้น

ปัญหาที่กล่าวมานี้สามารถทำได้ใน ไนแอม ซึ่งมีการนิยาม ออบเจกต์ ที่ชัดเจน และความสัมพันธ์ ต่างๆ ก็เหมาะสมกับสภาพที่เป็นจริงรวมทั้งมี กฎข้อบังคับสำหรับความสัมพันธ์ด้วย ดังนั้นจึงเลือกใช้ ไนแอม เป็นพื้นฐานในการทำวิจัยนี้

1.4 โครงสร้างวิทยานิพจน์

ในวิทยานิพนธ์ฉบับนี้ จะกล่าวถึงทฤษฎีที่เกี่ยวข้องในงานวิจัยคือ NIAM, DB และ OODB ในบทที่ 2 สำหรับในบทที่ 3 จะกล่าวถึง NIAM^{++} ในบทที่ 4 จะกล่าวถึง System Architecture ว่ามี Module อะไรบ้าง และแต่ละ Module มีความสัมพันธ์กันอย่างไร สำหรับบทที่ 5 จะกล่าวถึง Tool ที่เกี่ยวข้องในการสร้างระบบนี้คือ PRO* COBOL, SQL และ ORACLE ส่วนสรุปผลการวิจัยจะอยู่ในบทที่ 6

บทที่ 2
ทฤษฎี

2.1 ฐานข้อมูล

ฐานข้อมูล เป็นการรวบรวมข้อมูลชนิดต่าง ๆ เข้าไว้ด้วยกันโดยมีคุณลักษณะดังนี้

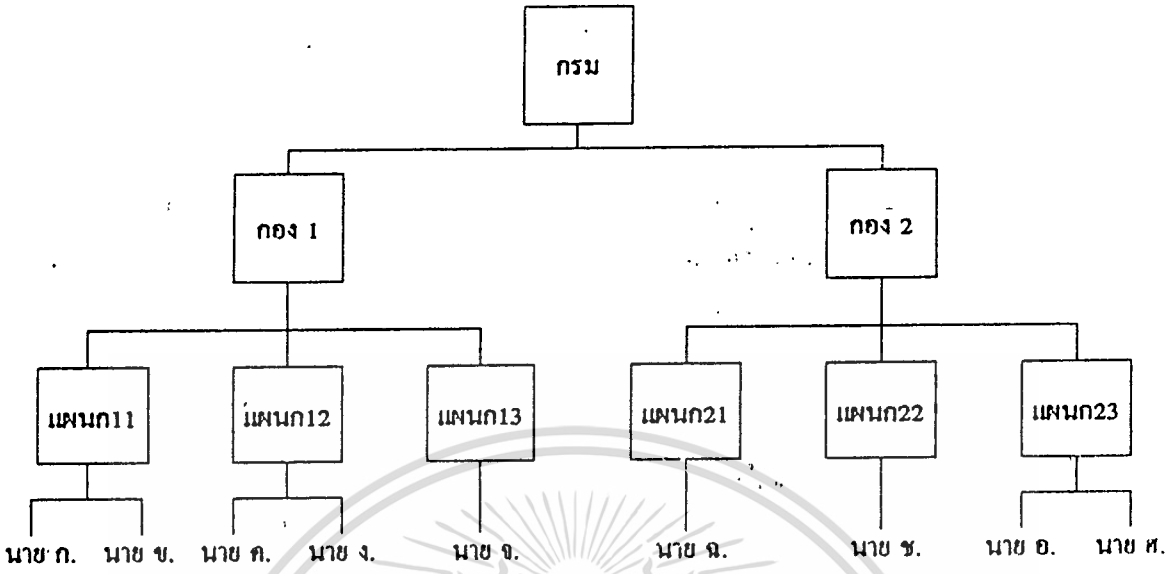
1. สามารถแสดงความสัมพันธ์ระหว่างข้อมูลได้
2. มีการซ้ำซ้อนของข้อมูลน้อย
3. เป็นอิสระจากโปรแกรมการใช้งาน โดยสามารถเปลี่ยนแปลงโครงสร้างฐานข้อมูลได้ตลอดเวลา
4. ผู้ใช้งานหลาย ๆ คนสามารถใช้ฐานข้อมูลในเวลาเดียวกันได้
5. มีระบบการรักษาความปลอดภัยที่ดี

2.1.1 โครงสร้างฐานข้อมูล

โครงสร้างของฐานข้อมูลมีการวิจัยและพัฒนามาเป็นลำดับ โดยนักค้นคว้าและวิจัยเพื่อให้สามารถจำลองความสัมพันธ์ระหว่างวัตถุ หรือสิ่งของ หรืออาจจะเป็นระบบที่เราสนใจออกมาเป็นสัญลักษณ์ทางคณิตศาสตร์ ทำให้นักค้นคว้าและวิจัย ได้ค้นพบโครงสร้างฐานข้อมูล 3 แบบคือ แบบจำลองลำดับชั้น (Hierarchy Model),แบบจำลองเครือข่าย (Network Model) และแบบจำลองเชิงสัมพันธ์ (Relational Model)

<u>แบบจำลองลำดับชั้น (Hierarchy Model)</u>

แบบจำลองลำดับชั้น เป็นโครงสร้างฐานข้อมูลแรกที่ค้นพบ และใช้กันมาเป็นเวลานานนับสิบปีแล้ว แนวความคิดในการพัฒนามาจากความตั้งใจที่จะสร้างฐานข้อมูลขององค์กร หรือหน่วยงานใดหน่วยงานหนึ่งที่มีข้อมูลจำนวนมาก จึงพิจารณาจากรูปแบบขององค์กร ดังรูปที่ 2.1

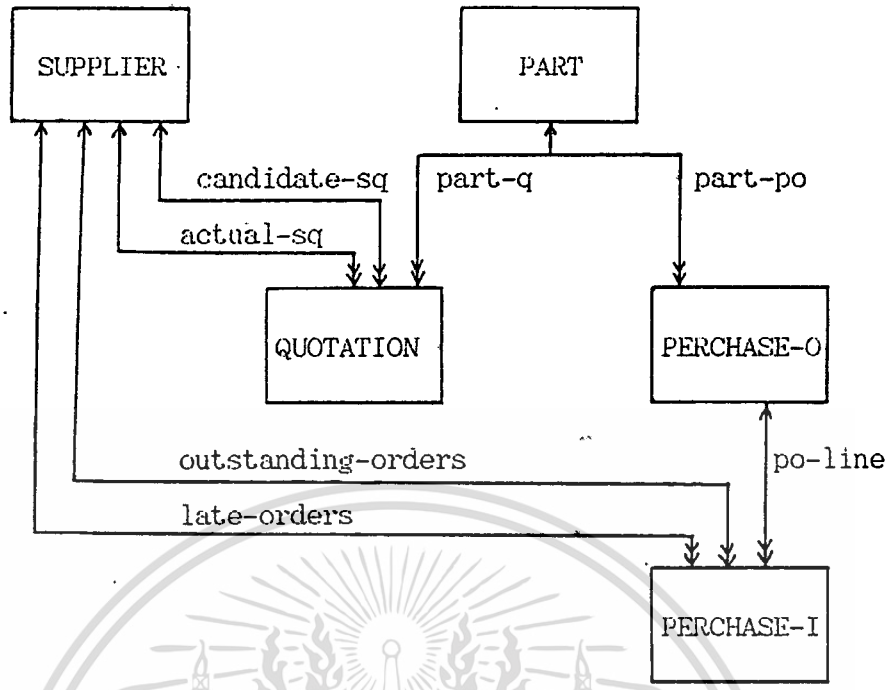


รูปที่ 2.1 โครงสร้างฐานข้อมูล แบบลำดับชั้น

จากรูปที่ 2.1 จะสังเกตเห็นได้ว่าความสัมพันธ์มีลักษณะเหมือนต้นไม้ จึงเรียกอีกชื่อหนึ่งว่า โครงสร้างต้นไม้ (Tree Structure) จากรูปร่างดังกล่าว นักค้นคว้าและวิจัยได้นำไปสร้างเป็นระบบจัดการฐานข้อมูลแบบลำดับชั้น

<u>แบบจำลองข้อมูลเครือข่าย (Network Model)</u>

แบบจำลองข้อมูลเครือข่าย เป็นโครงสร้างฐานข้อมูลที่พัฒนาต่อจากโครงสร้างฐานข้อมูลแบบลำดับชั้น เนื่องจากการประสบปัญหาว่าข้อมูลที่มีความสลับซับซ้อนนั้น ไม่สามารถใช้โครงสร้างฐานข้อมูลแบบลำดับชั้นออกแบบได้ หรือไม่เช่นนั้น ก็ประสบปัญหาเกี่ยวกับ ความเร็วในการค้นหาข้อมูลต่ำ เพราะการค้นหาข้อมูลแบบลำดับชั้น เป็นการค้นหาแบบบนลงล่าง และจากซ้ายไปขวา ตามการจัดของเรคอร์ดในฐานข้อมูล เช่น ถ้าต้องการหาว่า นาย จ. ทำงานอยู่แผนกใด จะต้องเริ่มค้นหาตั้งแต่ กรม ลงไป กอง 1 แล้วค่อยลงไปที่ แผนก 11 ถ้าไม่พบนาย จ. ก็กลับขึ้นไปหาแผนกต่อไปคือ แผนก 12 ยังไม่พบ นาย จ. ก็กลับขึ้นไปหา แผนกต่อไปคือ แผนก 8 จึงจะพบ นาย จ. จะเห็นได้ว่าถ้าฐานข้อมูลมีขนาดใหญ่มาก ๆ จะต้องเสียเวลาในการค้นหาข้อมูลมาก นักค้นคว้าและวิจัยจึงได้ทำการปรับปรุงโครงสร้าง โดยการเพิ่มขีดความสามารถ ให้สามารถเชื่อมโยงความสัมพันธ์ ไปยังข้อมูลที่ไม่ได้อยู่ภายใต้แขนงเดียวกัน ดังรูปที่ 2.2

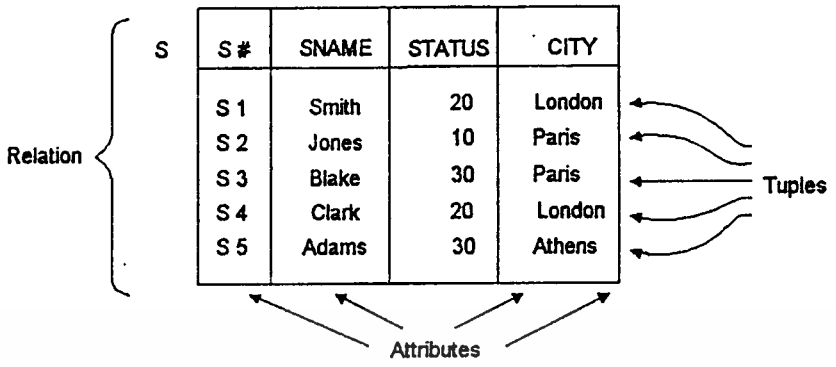


รูปที่ 2.2 โครงสร้างฐานข้อมูลแบบ เครือข่าย

จากรูปที่ 2.2 จะเห็นได้ว่าโครงสร้างฐานข้อมูล แบบเครือข่ายนั้นจะเน้นที่ ความสัมพันธ์ระหว่างข้อมูล ซึ่งแสดงด้วยเส้นเชื่อมระหว่างข้อมูล ถ้าข้อมูลมีจำนวนไม่มาก การออกแบบก็จะไม่ยากนัก แต่ถ้าข้อมูลมีจำนวนมาก ๆ และมีความสัมพันธ์ระหว่างกันมากแล้ว โครงสร้างแบบเครือข่าย ไม่เป็นที่นิยมมากนัก จะใช้แต่เฉพาะฐานข้อมูลที่มีข้อมูลไม่ยุ่งยาก และต้องการความเร็วในการค้นหาข้อมูลสูง

<u>แบบจำลองเชิงสัมพันธ์ (Relational Model)</u>

แบบจำลองเชิงสัมพันธ์ เป็นโครงสร้างฐานข้อมูลที่นักค้นคว้าและวิจัยพัฒนามาเพื่อ ใช้แก้ปัญหาและความยุ่งยากในการใช้งานของ แบบจำลองลำดับชั้น และ แบบจำลองเครือข่าย ดังที่ได้กล่าวมาแล้ว โดยที่ แบบจำลองเชิงสัมพันธ์ มีทฤษฎีทางคณิตศาสตร์รองรับ คือ ทฤษฎีกลุ่ม (Set Theory) และเป็นโครงสร้างข้อมูลแบบตาราง ที่ใช้กันอย่างแพร่หลายในปัจจุบัน ซึ่งฐานข้อมูลแบบเชิงสัมพันธ์ มีโครงสร้างเป็นตารางดังรูปที่ 2.3



รูปที่ 2.3 โครงสร้างฐานข้อมูลแบบ เชิงสัมพันธ์

แต่ละตารางเรียก เทเบิล (Table) หรือ รีเลชัน (Relation) ประกอบด้วยข้อมูลเรียงกันเป็นแถว เรียกว่า เรคอร์ด (Record) หรือ โรล (Row) หรือ ทิปเปิล (Tuple) ซึ่งประกอบด้วยข้อมูลจำนวน 1 ชุด ในแต่ละทิปเปิล แบ่งออกเป็นส่วน ๆ เรียก คอร์ลัม (Column)

2.2 แบบจำลองข้อมูล ไนแอม (The NIAM Conceptual Model)

ในราวปลายปี ค.ศ. 1977 [3] ศาสตราจารย์ จี เอ็ม ไนเซน ได้เสนอแบบจำลองข้อมูล ไนแอม ซึ่งถูกเลือกว่าเป็นแบบจำลองข้อมูลแนวความคิดที่เหมาะสม และมีการแทนในรูปของ กราฟ (Graph) ที่ง่ายต่อการอ่าน และเข้าใจในขั้นตอนการออกแบบ

2.2.1 ส่วนประกอบของ ไนแอม

แบบจำลองข้อมูล ไนแอม มีส่วนประกอบพื้นฐาน ดังนี้ คือ

1. เอนทิตี ไทพ์
2. เลเบล ไทพ์
3. อีลีเมนทารีแฟคท์ ไทพ์
4. ซับไทพ์
5. คอนสเทรน

เอนทิตี ไทพ์ คือ กลุ่มของ นามธรรม หรือ เอนทิตีจริง ๆ ซึ่งอาจเป็นสิ่งที่จับต้องได้ หรือไม่ได้ เช่น PERSON, DEPARTMENT, COMPANY

เลเบล ไทพ์ คือ ตัวที่ใช้บอกความแตกต่างของเอนทิตีภายในเอนทิตี ไทพ์ (ใช้เป็น คีย์)

อีลีเมนทารีแฟคท์ ไทพ์ คือ ตัวที่ใช้แสดงความสัมพันธ์ ระหว่าง เอนทิตี ไทพ์ 2 เอนทิตี ไทพ์ ขึ้นไป

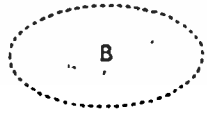
ซับไทพ์ คือตัวที่ใช้ในการกำหนด โครงสร้างลำดับชั้นที่ซับซ้อน ของเอนทิตี ไทพ์ และ ซับไทพ์ มีการถ่ายทอดแฟคท์ไทพ์ จาก ซุปเปอร์ไทพ์ ด้วย

คอนสเทรน คือตัวที่ใช้เป็นเงื่อนไขที่จะทำให้แน่ชัดว่าฐานข้อมูลนั้นถูกต้องและสมบูรณ์ตลอดเวลา (Consistency and Integrity) คอนสเทรน ที่ใช้ใน ไนแอม มีดังนี้

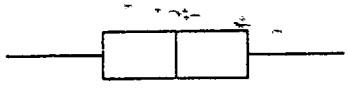
1. ยูนิกเนส คอนสเทรน (Uniqueness Constriant) แบ่งเป็น
   1.1 อินเตอร์ แฟคท์ ไทพ์ (Inter Fact Type)
   1.2 อินตร้า แฟคท์ ไทพ์ (Intra Fact Type)
2. แมนดาทอรี คอนสเทรน (Mandatory Fact)
3. เอ็กคูชั่น คอนสเทรน (Exclusion Constraint)
4. อีควลลิตี คอนสเทรน (Equality Constraint)
5. ซับเซ็ต คอนสเทรน (Subset Constraint)

2.2.2 สัญลักษณ์ที่ใช้ใน ไนแอม

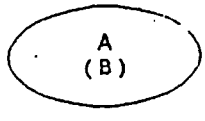
แทน เอนทิตี ไทพ์ A



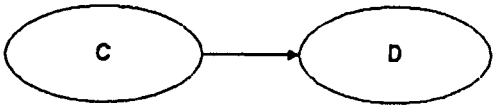
แทน เลเบล ไทพ์ B



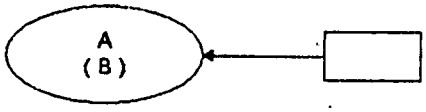
แทน ไบนารี แฟคท์ ไทพ์



แทน เอนทิตี ไทพ์ A และมี เลเบล ไทพ์ B



แทน เอนทิตี ไทพ์ C เป็น ซับไทพ์ ของ เอนทิตี ไทพ์ D ซึ่งเป็น ซุปเปอร์ไทพ์



แทน แมนดาทอรี แฟคท์ คอนสเทรน (ต้องการแมนดาทอรี ที่เอนทิตี ไทพ์ไหนก็ใส่ "." ที่เอนทิตี ไทพ์ นั้น)

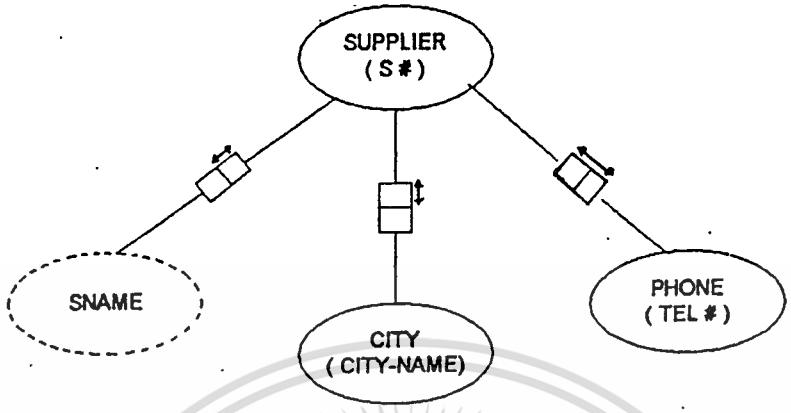
แทน อินตร้า แฟคท์ ไทพ์ ยูนิกเนส คอนสเทรน

แทน อินเตอร์ แฟคท์ ไทพ์ ยูนิกเนส คอนสเทรน

แทน เอ็กคูชัน คอนสเทรน

แทน อีควลลิตี คอนสเทรน

แทน ซับเซ็ต คอนสเทรน



รูปที่ 2.4 ตัวอย่าง ไนแอม

จากรูปที่ 2.4 สรุปได้ดังนี้

1. เอนทิตี ไทพ์ SUPPLIER มี เลเบล ไทพ์ เป็น S#
2. เอนทิตี ไทพ์ PHONE มี เลเบล ไทพ์ เป็น TEL#
3. เอนทิตี ไทพ์ CITY มี เลเบล ไทพ์ เป็น CITY-NAME
4. เอนทิตี ไทพ์ SUPPLIER มี อีลิเมนทารี แฟคท์ ไทพ์ เป็นตัวเชื่อมความสัมพันธ์กับ เอนทิตี ไทพ์ PHONE, CITY
5. เอนทิตี ไทพ์ SUPPLIER มี เรฟเฟอร์เรน ไทพ์ เป็นตัวเชื่อมความสัมพันธ์กับ เลเบล ไทพ์ SNAME

2.3 คอมเพลคซ์ ออบเจกต์ (Complex Object)

ที่จริงแล้ว ออบเจกต์ (Oject) หรือ วัตถุสิ่งของต่าง ๆ ส่วนใหญ่ จะมีโครงสร้างที่สลับซับซ้อน เช่น รถยนต์ ประกอบด้วยส่วนต่าง ๆ ดังนี้คือ ระบบไฟฟ้า, ระบบส่งกำลัง, ระบบช่วงล่าง, ระบบตัวถัง และระบบเครื่องยนต์ ซึ่งระบบเครื่องยนต์เองก็ยัง ประกอบด้วย ระบบน้ำมันเชื้อเพลิง, ระบบหล่อลื่น, ระบบหล่อเย็น และระบบจุดระเบิด ดังนั้น เครื่องยนต์ก็เป็น คอมเพลคซ์ ออบเจกต์ จะเห็นได้ว่า คอมเพลคซ์ ออบเจกต์ สามารถที่จะประกอบด้วย คอมเพลคซ์ ออบเจกต์ อื่นได้เช่นกัน

คอมเพลคซ์ ออบเจกต์ [1] หมายถึง การรวมกันของ ซับออบเจกต์ (Subobject) ที่เป็นโครงสร้าง ของ ออบเจกต์ แบ่งออกเป็น 2 ชนิด

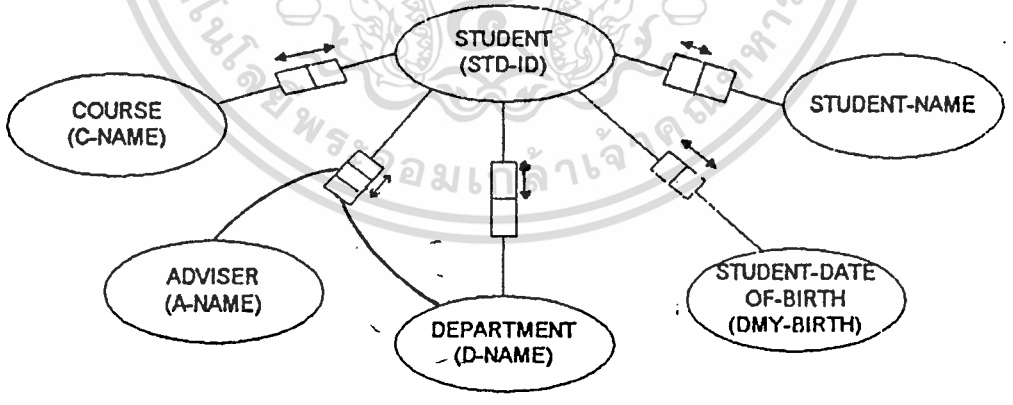
1. Non-disjoint Complex Object คือ คอมเพลคซ์ ออบเจกต์ ที่เกิดจากการรวมกันของหลาย ๆ ซับออบเจกต์ ที่มาจาก คอมเพลคซ์ ออบเจกต์ อื่นด้วย

2. Disjoint Complex Object คือ คอมเพลคซ์ ออบเจกต์ ที่เกิดจากการรวมกันของหลาย ๆ ซับออบเจกต์ ที่ไม่เกี่ยวข้องกับ คอมเพลคซ์ ออบเจกต์ อื่น

2.4 ชนิดของความสัมพันธ์ (Relationship Type)

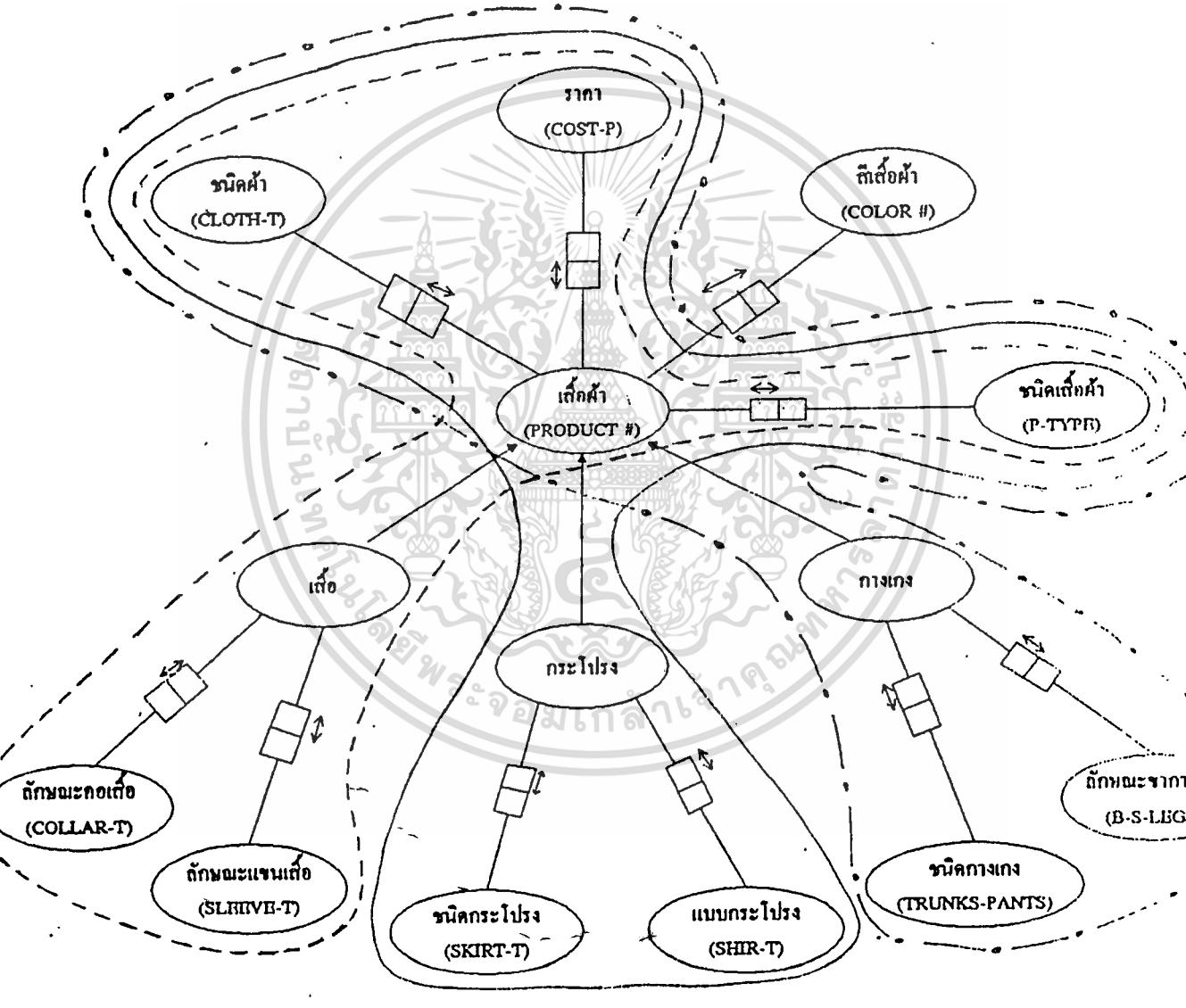
ชนิดของความสัมพันธ์ [4] ที่สำคัญ สามารถแบ่งได้เป็น 3 ชนิด

1. แอสโซซิเอชัน (Association) เป็นความสัมพันธ์ที่เชื่อม ออบเจกต์ ที่เป็นออบเจกต์ อิสระเข้าด้วยกัน เช่น STUDENT มีความสัมพันธ์กับ COURSE และ STUDENT-NAME, STUDENT-DATE-OF-BIRTH สามารถใช้ไนแอมออกแบบได้ดังรูป 2.5 และจะสังเกตเห็นได้ว่าทั้ง แฟคท์ ไทพ์ และ เรฟเฟอร์เรน ไทพ์ (Reference Type) ในไนแอมจัดได้ว่าเป็น แอสโซซิเอชัน



รูปที่ 2.5 แสดง ชนิดความสัมพันธ์แบบแอสโซซิเอชัน

2. เจนเนอรัลไลเซชัน (Generalization) หรือ ความสัมพันธ์แบบ ซุปเปอร์ไทพ์ กับ ซับไทพ์ นั้นคือ ความสัมพันธ์ที่แบ่งคุณลักษณะเฉพาะที่ร่วมกันออกเป็น ซุปเปอร์ไทพ์ และแบ่งคุณลักษณะเฉพาะตัวออกเป็น ซับไทพ์ เช่น ฐานข้อมูล ร้านขายเสื้อผ้า มีสินค้า เช่น เสื้อ, กระโปรง, กางเกง ซึ่งสินค้าแต่ละชนิดจะมีคุณลักษณะเฉพาะของตัวเอง แต่จะมีข้อมูลที่ใช้ร่วมกัน เช่น สีของกระโปรง, สีกางเกง, สีเสื้อ, ชนิดของผ้าที่ใช้ สามารถใช้ ไนแอมออกแบบได้ ดังรูปที่ 2.6



รูปที่ 2.6 แสดง ชนิดความสัมพันธ์แบบเจนเนอรัลไลเซชัน

3. แอกกริเกชัน (Aggregation) หรือความสัมพันธ์แบบ ซุปเปอร์-พาร์ท-ออฟ กับ ซับ-พาร์ท-ออฟ คือความสัมพันธ์ของหลาย ๆ ออบเจกต์รวมเป็นออบเจกต์เดียว ตัวอย่าง เช่น รถยนต์ เกิดจากการรวมกันหลาย ๆ องค์ประกอบนั่นคือ ระบบเครื่องยนต์ ระบบช่วงล่าง ระบบส่งกำลัง ฯลฯ

ไนแอม ไม่ได้กำหนดรูปแบบของความสัมพันธ์ประเภทนี้ไว้ จึงไม่สามารถใช้สัญลักษณ์ของ ไนแอม แทนความสัมพันธ์ประเภทนี้ได้ และจะเห็นได้ว่าในความเป็นจริงนั้น วัตถุ หรือสิ่งของต่าง ๆ ที่เกี่ยวข้องในชีวิตประจำวันมักเป็น คอมเพลกซ์ ออบเจกต์ ที่มีความสัมพันธ์แบบ แอกกริเกชัน เสียส่วนใหญ่

2.5 แนวความคิดเชิงวัตถุ (Object-Oriented Concept)

ผู้ใช้ไม่จำเป็นต้องรู้โครงสร้างของของการจัดการภายในคอมพิวเตอร์ เช่น เรคอร์ด, ไฟล์ แต่ควรจะต้องรู้เกี่ยวกับ ออบเจกต์ (Object) และ การดำเนินการ (Operation) มีคำศัพท์ที่เกี่ยวข้องกับ Object-Oriented อยู่ 5 คำ คือ

1. ออบเจกต์ (Object)
2. คลาส (Class)
3. เมตธอด (Method)
4. เมสเสจ (Message)
5. คลาส ไฮราคี (Class Hierarchy)

ตารางที่ 1

แสดงการเปรียบเทียบคำศัพท์ระหว่าง ความคิดเชิงวัตถุ กับ โปรแกรม และ ไนแอม

Object-Oriented Term	Programming Term	NIAM Term
Object	Variable	Entity
Class	Type	Entity Type
Method	Function	-
Message	Call	-
Class Hierarchy	Type Hierarchy	Subtype

ออบเจกต์ (Object)

ออบเจกต์ ใน Object-Oriented ตรงกับคำว่า เอนทิตี ในไนแอม และ ออบเจกต์ จะอยู่ภายใต้ คลาส ซึ่งเหมือนกับ เอนทิตี จะอยู่ภายใต้ เอนทิตี ไทพ์ และทุกออบเจกต์ ในระบบจะต้องไม่มีการซ้ำกัน (Unique)

ออบเจกต์ จะง่ายหรือซับซ้อนขึ้นอยู่กับระบบ สิ่งแรกจะต้องเตรียม เซ็ตของ ออบเจกต์ คลาส ดั้งเดิมที่มีอยู่แล้ว เช่น Integer หรือ Float ซึ่งผู้ใช้ สามารถที่จะสร้าง ออบเจกต์ ของตัวเองขึ้นมาจาก คลาสพื้นฐาน (Premitive Class) ก็ได้ ออบเจกต์ ที่ซับซ้อนสามารถสร้างจากการรวมกันเองของ ออบเจกต์ ที่มีอยู่แล้วก็ได้

ถ้า ออบเจกต์ ถูกสร้างขึ้นจาก ออบเจกต์ อื่น ส่วนประกอบของ ออบเจกต์ ที่ถูกอ้างถึงคือ Instance Variable ตัวอย่าง เช่น Student ทิปเปิล ออบเจกต์ บรรจุด้วย 3 Instance Variable คือ Student-id, Student-name, Student-address

คลาส (Class)

คลาส ใน Object-Oriented ตรงกับคำว่า เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ในไนแอม ใช้จัดกรุ๊ปของ ออบเจกต์ ตามคุณสมบัติ และผู้ใช้ สามารถติดต่อกับ คลาส ผ่านทาง เมตธนด

เมตธอด (Method)

เป็น การดำเนินการ (Operation) ที่สำคัญหรือเป็น ฟังค์ชัน (Function)ที่สามารถทำงานกับออบเจกต์ แต่ละคลาสจะมี เมตธอดของตังเอง เซ็ตของเมตธอดที่นำไปใช้กับคลาสที่กำหนดจะต้องคำนึงถึง นิยามที่กำหนดและเก็บใน คลาส นั้น

เมสเสจ (Message)

เมตธอดจะติดต่อกับ ออบเจกต์ ผ่านทาง เมสเสจ เช่น ออบเจกต์ จะประมวลผลตามฟังค์ชัน หรือ เมตธอดที่ต้องการ โดยผ่านเมสเสจแล้วจะส่งผลลัพธ์ไปยังผู้ส่งเมสเสจ)

คลาส ไฮรากี (Class Heirarchy)

คลาส ไฮรากี ใน Object-Oriented ตรงกับคำว่าซับไทพ์ใน ไนแอม และ เหมือนโครงสร้างเป็นลำดับชั้น(Heirachy Concept) เช่น ถ้า คลาส B เป็น ซับคลาสของ คลาส A แล้ว ทุก ๆ Instance ของ B จะเป็น Instance ของ A โดยอัตโนมัติ และ B ก็จะได้รับการถ่ายทอดคุณสมบัติจาก A ด้วย

การถ่ายทอดคุณสมบัติมี 2 ชนิด

1. การถ่ายทอดทางโครงสร้าง (Structure Inheritance) เช่น B ได้รับการถ่ายทอดคุณสมบัติ Instance Variable จาก A

2. การถ่ายทอดทางพฤติกรรม (Behavioral Inheritance) เช่น B ได้รับการถ่ายทอดเมตธอดจาก A

2.6 OODB (Object-Oriented Database)

OODB คือฐานข้อมูลชนิดหนึ่งที่ถูกแสดงด้วยชนิดของข้อมูลที่กำหนดขึ้น และ โอเปอร์เรชัน (Operation) ที่ถูกออกแบบมาเพื่อใช้กับชนิดของข้อมูลเหล่านั้น

OODB เป็นฐานข้อมูลที่สนับสนุนลักษณะ Object-Oriented โดยใช้ ออบเจกต์ เป็นพื้นฐาน OODB เกิดขึ้นมาจากการรวมกันของภาษา Object-Oriented Programming และเทคโนโลยีทางด้านฐานข้อมูล ซึ่งก่อให้เกิดประโยชน์อย่างมากดังนี้

1. การที่ Object-Oriented มีการเปลี่ยนแปลงได้ง่าย (Flexibility) จึงอำนวยความสะดวกในการออกแบบงานทางด้านฐานข้อมูลที่มีโครงสร้างสลับซับซ้อน เช่น CAD ซึ่งไม่ได้สนับสนุนโดยแบบจำลองข้อมูลเชิงสัมพันธ์

2. การใช้ภาษาที่เป็น Object-Oriented Programming จะได้รับประโยชน์จากคุณลักษณะของฐานข้อมูล เช่น การคงอยู่ของข้อมูล (Persistent Data), Set-Oriented Processing และ Transaction Management

3. สิ่งที่สำคัญที่สุดคือ งานทางด้านฐานข้อมูลที่มีโครงสร้างที่สลับซับซ้อน อาจถูกเขียนขึ้นภายใน Single OODB Programming Language

ถึงแม้ว่า Programming Language Object และ ฐานข้อมูลเชิงวัตถุจะมีความคล้ายคลึงกันในเรื่องของคุณสมบัติ แต่ก็มีความแตกต่างกัน ที่สำคัญหลายประการคือ

1. ข้อมูลของฐานข้อมูลเชิงวัตถุจะต้องคงอยู่ (Persist) ตลอดภายใต้การทำงานของโปรแกรมที่สร้างมันขึ้นมา

2. งานทางด้านฐานข้อมูลหลาย ๆ งานต้องการความสามารถในการสร้าง (Create) และ การเข้าถึง กับ ออบเจกต์ ในแบบของ Multiple Version
(ตัวอย่างในลักษณะที่พบคือ Historical Database , Database ที่ใช้สำหรับ Software Management และ Computer-Aided Design)

3. ฐานข้อมูลที่มี ความเร็ว (Active) สูง ๆ เป็น ฐานข้อมูลที่ใช้ในการควบคุม การจราจรทางอากาศ และ ฐานข้อมูลที่ใช้จัดการกับ Distribution ให้มีประสิทธิภาพต้องการความสามารถที่เกี่ยวข้องกับเงื่อนไข (Condition) และการกระทำ(Action) กับออบเจกต์ ซึ่งการดำเนินการจะถูกทำให้สมบูรณ์ เมื่อเงื่อนไขเหมาะสมกับ ออบเจกต์ นั้น

4. ฐานข้อมูลต้องการความสามารถ หรือ ประสิทธิภาพของ Query ใน Perdicate-Based บน ออบเจกต์

โดยปกติแล้ว Programming Language ทั่วไปส่วนใหญ่จะไม่สนับสนุนการคงอยู่ (Persistent) ออบเจกต์ หรือ Multiple Object Version และ ไม่อำนวยความสะดวกเกี่ยวกับเรื่อง คอนสเทรน

สิ่งที่สำคัญใน OODB คือ

1. ออบเจกต์ (Objects)
2. แบบข้อมูลชนิดนามธรรม (Abstract Data Types)
3. การสืบทอดคุณสมบัติ (Interitance)

2.6.1 ออบเจกต์ (Objects)

Object-Oriented Paradigm เป็นการรวมCode และ Data (Encapsulation) เข้าไว้ด้วยกันซึ่งเรียกว่า ออบเจกต์ ส่วนที่ทำหน้าที่ในการเชื่อมต่อ ออบเจกต์ ต่าง ๆ ในระบบคือ เซ็ตของเมสเสจ

โดยทั่วไปแล้ว ออบเจกต์ จะเกี่ยวข้องกับสิ่งต่าง ๆ ดังนี้

1. เซ็ตของตัวแปร (Variable)ที่ใช้เก็บข้อมูลสำหรับ ออบเจกต์ โดยที่ค่าของตัวแปรแต่ละตัวก็ถือเป็น ออบเจกต์ ด้วยเช่นกัน
2. เซ็ตของเมสเสจซึ่ง ออบเจกต์ ตอบสนองด้วย
3. เมตธอด ซึ่งเป็น Code ที่ใช้ในการสร้างแต่ละ เมสเสจ

เมสเสจ หมายถึง การส่ง Request ระหว่าง ออบเจกต์ โดยไม่คำนึงถึงรายละเอียดในการ Implement เพราะว่า External Interface เท่านั้นที่เป็น เซ็ตของเมสเสจซึ่งออบเจกต์จะตอบสนองด้วย จะมีทางเป็นไปได้ที่จะทำการปรับปรุง Definition ของเมตธอด และตัวแปรของออบเจกต์หนึ่งโดยไม่มีผลกระทบถึงออบเจกต์อื่น นอกจากนี้ยังสามารถแทนตัวแปรด้วย เมตธอด ที่ใช้คำนวณค่าได้อีกด้วย ยกตัวอย่างเช่น Document ออบเจกต์ประกอบด้วยตัวแปร "Size" ซึ่งใช้เก็บจำนวน ไบท์ (Byte) ของ Text ใน Document หรือ เมตธอด "Size" ที่ใช้คำนวนขนาดของ Document โดยการอ่าน Document เข้ามาแล้วทำการนับจำนวนไบท์ใน Document นั้น

ความสามารถในการเปลี่ยนนิยามของออบเจกต์หนึ่งโดยไม่ก่อให้เกิดผลกระทบไปถึงส่วนที่เหลือของระบบเป็นข้อดีอย่างหนึ่งของ Object-Oriented Programming Paradigm

2.6.1.1 Schema Evolution and Reuseability

ในระบบการจัดการฐานข้อมูลทั่ว ๆ ไป (Conventional Database Management System) มีข้อจำกัดในการขยาย หรือ ปรับปรุงเปลี่ยนแปลงโครงสร้างข้อมูล เช่น โดยปกติฐานข้อมูลแบบเชิงสัมพันธ์ จะอนุญาตให้ทำการ สร้าง (Create) หรือ ลบ (Delete) รีเลชัน และเพิ่ม คอร์ลัม (Column) ใหม่ เข้าไปในรีเลชันได้ ในการเปลี่ยนชนิดของโดเมน จะต้องสร้างรีเลชัน ที่เกี่ยวข้องกันขึ้นมาใหม่ และทำการเปลี่ยนแปลง โปรแกรม หรือ งานต่าง ๆ ที่เรียกใช้รีเลชันนั้นใหม่ด้วย

ทั้งการพัฒนาโครงสร้าง (Extension และ Refinement ของโครงสร้างที่มีอยู่แล้ว) และการ นำกลับมาใช้ (Reuse) ของ โปรแกรม หรือ ฟังค์ชัน เป็นส่วนที่สำคัญของแบบจำลองข้อมูลแบบเชิงวัตถุ (Object-Oriented Data Model) โดยธรรมชาติของแบบจำลองแบบเชิงวัตถุ (Object-Oriented Model) ได้เอื้อให้ Semantic Schema Evolution ถูกนิยาม และ แก้ไข (Validate) ได้ ในขณะที่ความสามารถทางด้านการถ่ายทอดคุณสมบัติของแบบจำลองก่อให้เกิด Generic Component ซึ่งสามารถนำมาใช้งานได้ใหม่ในหลาย ๆ ส่วนของระบบ

2.6.1.2 Concurrency

เมื่อโปรแกรมหลาย ๆ โปรแกรม หรือ Terminal User หลายหลายคนกำลังติดต่อกับฐานข้อมูลพร้อม ๆ กัน การดำเนินการที่กระทำบน ฐานข้อมูล จะต้องมีการควบคุมด้วยความระมัดระวังโดยเฉพาะอย่างยิ่ง ผลกระทบที่เกิดจากขบวนการทางด้าน ฐานข้อมูล (หรือ Transaction) จะต้องให้ผลเหมือนกับผลที่ได้เมื่อไม่มีการ ประมวลผลของ Transaction อื่น (เรียกว่า Serializable) และใน ระบบการจัดการฐานข้อมูลส่วนใหญ่ ก็ได้มีข้อจำกัดมากมายบน Transaction เพื่อเป็นการประกันถึง Serializability

ในการป้องกัน Transaction หนึ่ง ๆ ที่กำลังอ่านหรือแก้ไขข้อมูลอยู่ จากการแก้ไขของ Transaction อื่น ๆ อาจทำได้โดยใช้ Locking Mechanism อย่างไรก็ตาม การ Lock ทำให้เกิดค่าใช่จ่ายในการบำรุงรักษาระบบ และอาจก่อให้เกิด สภาวะที่ลำบากขึ้น โดยเฉพาะอย่างยิ่ง Item ซึ่งมีขนาดใหญ่ (เช่น รีเลชัน ทั้งหมด) ดังนั้น เมื่อไม่กี่ปีมานี้ ได้มีทางเลือกใหม่ ๆ เพื่อใช้แทนการLock เช่น Timestampingและ Optimistic Concurrency Control

ฐานข้อมูลเชิงวัตถุ มีความสามารถในด้าน การใช้งานพร้อม ๆ กัน (Concurrency) มากกว่า ระบบธรรมดา และจากความจริงที่ว่า ในระบบเชิงวัตถุ การดำเนินการบนฐานข้อมูล

ไม่ง่ายที่จะทำการอ่านหรือเขียน แต่ก็มีการสืบทอดของ Semantic อยู่มากเหมือนกัน นอกจากนี้ระบบยังสามารถใช้ประโยชน์จาก Semantic Scheduling Transaction ได้อีกด้วย

Object-Based Concurrency Control ในระบบการจัดการฐานข้อมูล ยังคงเป็นหัวข้อในการทำวิจัย เพื่อนำมาซึ่งการพัฒนา Model ใหม่ของ Concurrency Control สำหรับ Cooperative Transaction

2.6.1.3 Correctness

ความถูกต้องของ โปรแกรม เป็นสิ่งสำคัญในการทำงานเกี่ยวกับ วิศวกรรมชุดคำสั่ง (Software Engineering) และ การออกแบบภาษาชุดคำสั่ง (Programming Language) ในทางอุดมคติ ควรจะกำหนดความถูกต้อง โดยการพิสูจน์ทางคณิตศาสตร์ว่าโปรแกรมมีลักษณะตรงตามข้อกำหนดของมัน

เป็นความจริงที่ว่า ลักษณะของภาษาชุดคำสั่ง และทักษะที่ใช้ในการเขียนโปรแกรมภาษานั้น เป็นส่วนสำคัญสำหรับความถูกต้องของโปรแกรม การสนับสนุนภาษาที่เกี่ยวข้องกับแนวความคิดของข้อมูลชนิดนามธรรม และ Strong Type Checking มีส่วนทำให้การทดสอบและ ตรวจแก้โปรแกรมกระทำได้ง่ายขึ้น จุดมุ่งหมายสูงสุดก็คือ ความสามารถในการแยกโปรแกรมที่มีขนาดใหญ่ ออกเป็น โมดูล (Module) ซึ่งจะทำให้สามารถ compile และทดสอบแต่ละ โมดูล แยกกันได้

สำหรับระบบฐานข้อมูลความถูกต้องไม่ได้หมายความว่าลักษณะของระบบจะต้องตรงกับข้อกำหนด (Specification) ที่ได้กำหนดไว้เท่านั้น แต่ยังหมายรวมถึงค่า และความสัมพันธ์ที่ถูกรักษาไว้ในระบบ โดยให้เป็นไปตามการสืบทอด การควบคุมความถูกต้อง ในแบบจำลองข้อมูลด้วย และให้มีการแสดงสถานะของ Enterprise ที่เหมาะสมตลอดเวลา การรักษาความถูกต้อง และการร่วมกันของข้อมูลในระบบ Multiuser Database ที่มีขนาดใหญ่ถือว่าเป็นปัญหาสำคัญอย่างหนึ่ง

ความถูกต้องใน ภาษาชุดคำสั่งจะกำหนดใน เงื่อนไขของ Precondition, Postcondition และ Invariant การดูแลบำรุงรักษาความถูกต้อง ในระบบฐานข้อมูลสามารถทำได้โดย ใช้ความรู้ ซึ่งกำหนดเกี่ยวกับ ข้อมูล และ งานของระบบ ในรูปแบบของการควบคุมความถูกต้อง หรือ Trigger Integrity Constraint เป็น Predicate ที่จะต้องมีค่าเป็นจริงเสมอ ทำการกำหนด Trigger โดย Predicate และ Body ของ Code เมื่อไรก็ตามที่ Predicate มีค่าเป็นจริง Code ก็จะถูก Execute (Trigger ถูก Fire) ระบบฐานข้อมูลจะทำการตรวจสอบความรู้นี้อยู่ตลอดเวลา Transection ใดที่ละเมิด

กฎควบคุมความถูกต้อง จะถูกเอาทิ้งไป ในขณะเดียวกันกับที่ Trigger อาจจะถูก Fire เมื่อผลของ Transaction หนึ่ง ทำให้ Predicate ที่เหมาะสมมีค่าเป็นจริง ส่วน Body ของ Trigger อาจทำ การดำเนินการที่จำเป็นเพื่อรักษาความถูกต้องของฐานข้อมูล

2.6.1.4 Presistence

แนวความคิดในการขยายภาษาชุดคำสั่ง เพื่อรองรับแนวความคิดของ การจัดการฐานข้อมูล คือ การอนุญาตให้ทุก ๆ ข้อมูลของออบเจกต์ (ไม่ใช่ ไฟล์ หรือ รีเลชัน) มี Persistence นั่นคือ จะมีข้อมูลของออบเจกต์ ทั้งหมดอยู่จริงตลอดเวลาการทำงานของโปรแกรมที่ใช้สร้าง ข้อมูลของออบเจกต์ขึ้นมา

การสนับสนุนที่ไม่เพียงพอ สำหรับการคงอยู่ของข้อมูล เป็นข้อบกพร่องของการออกแบบภาษา ซึ่งการสนับสนุนการคงอยู่ของข้อมูลนี้ ไม่มีส่วนเกี่ยวข้องกับการเพิ่ม โครงสร้างข้อมูลใหม่เข้าไปในภาษา แต่เป็นการเตรียมคุณสมบัติที่สามารถนำไปใช้ได้กับ โครงสร้างข้อมูลเดิมที่มีอยู่แล้ว

2.6.2 Data Abstraction

เมสเสจ ที่ส่งเพื่อเรียกใช้ การดำเนินการ หรือ ฟังค์ชันบน ออบเจกต์ สนับสนุนหลักการสำคัญใน โปรแกรมโครงสร้างที่เรียกว่า ดาต้าแอบแทรกชัน (Data Abstraction) หลักการดังกล่าวได้แก่ โปรแกรมไม่ควรทำ ทำการที่เกี่ยวกับการ Implement และการแสดงลักษณะภายในของ ชนิดข้อมูลที่มันใช้ ด้วยวิธีนี้ มีทางเป็นไปได้ที่จะเปลี่ยนการ Implement ของชนิดข้อมูล โดยไม่ต้องเปลี่ยนโปรแกรม ดังนั้นวัตถุประสงค์ของ Abstraction ในการเขียนโปรแกรมก็คือ การแยก Behaviour ออกจากการ Implement ทำให้การออกแบบ Software ซึ่งมีขนาดใหญ่ถูกกระทำได้ง่าย เพราะ Abstraction ยอมให้มีการแยก และพัฒนาส่วนต่าง ๆ ของระบบได้อย่างอิสระ

กลไกง่าย ๆ เกี่ยวกับ Programming Abstraction คือ Procedure อย่างไรก็ตาม สำหรับระบบที่ใช้ Software ขนาดใหญ่ ต้องการ Abstraction Mechanism ที่มีความสามารถมากขึ้น ซึ่งก่อให้เกิดแนวความคิดของ แบบชนิดข้อมูลนามธรรม (ADT) ขึ้น

แนวความคิดพื้นฐานของ แบบชนิดข้อมูลนามธรรม คือ การแยกภาพการมองของผู้ใช้ออกจากรายละเอียดของการ Implement

2.6.2.1 Definition ของ Abstract Data Type (ADT)

ADT กำหนด คลาส ของ ออบเจกต์ (ซึ่งมีโครงสร้างข้อมูลนามธรรมเดียวกัน) โดยใช้การดำเนินการ และคุณสมบัติของ การดำเนินการ ใน คลาส นั้น ๆ เป็นตัวกำหนด นั้นคือ ADT จะกำหนด คลาส ของ ออบเจกต์ ซึ่งสามารถจัดการได้โดยใช้เซ็ตของการดำเนินการที่กำหนดให้เท่านั้น

ADT มีบทบาทต่อ Application ทางด้าน Database อยู่ ๓ ประการ คือ

1. ADT จะถูกใช้เพื่อขยาย Typing System ของภาษา, เพื่อเตรียม Type ของ Attribute ให้มีความซับซ้อนมากขึ้น เช่น Data, Time, Colour, Name ฯลฯ ซึ่ง Type เหล่านี้ไม่ใช่ Built-In Type

2. ADT อาจถูกสร้างขึ้น เพื่อแสดง Database Object ในระดับสูง ในกรณีนี้เราจะใช้ ADT เพื่อแสดง Time-Evolving Entity ซึ่งตรงกันข้ามกับ Inert Data โดย ADT จะจัดเตรียมความสะดวกในการ Implement Integrity คอนสเทรน และ Trig procedure ซึ่ง Enforce Behaviour ของ ออบเจกต์ ที่ Runtime

3. ADT อาจแสดงความสัมพันธ์ระหว่าง Entity ผ่านทาง Aggregation และ Generalifzation Abstraction

- Aggregation เป็น Abstraction ซึ่งมีความสัมพันธ์ระหว่าง ออบเจกต์ ถูกแสดงโดย Higher-Level Aggregate Object(หรือ Type)

- Generalization เป็น Abstraction ซึ่ง เซ็ตของ ออบเจกต์ ที่มีคุณสมบัติคล้ายคลึงกันจะถูกแสดงโดย Generic Object

2.6.2.2 Data Abstraction and Strong Typing

ภาษาประเภท Strong Type ได้ให้ความสามารถในการกำหนด Type ที่ Compatible กับทุก ๆ นิพจน์ (Expression) ที่แสดงค่าซึ่งได้จาก Static Program ที่ Compile time และ Strongly Typed Programming language ไม่ต้องกำหนดชนิดของนิพจน์ที่ Compile Time ซึ่งมี Compatibility ของตัวถูกดำเนินการ (Operand) หรือ ตัวดำเนินการ (Operator)

การเตรียมชนิดข้อมูลหลาย ๆ ชนิดโดยใช้ Strong Type Checking เป็นแนวความคิดของภาษาชุดคำสั่ง ที่มีวัตถุประสงค์เพื่อทำให้เกิดผลดีของ Software Integrity และการบำรุงรักษาสำหรับงานทางด้านฐานข้อมูล โปรแกรมเมอร์ ควรจะสามารถกำหนดคุณสมบัติ (Property) ของข้อมูลของออบเจกต์ และ รูล (Rule) ของภาษาควรจะ รับประกันได้ว่า

คุณสมบัติเหล่านี้ถูก บำรุงรักษาตลอดทั้งโปรแกรม คุณสมบัติที่เป็นของหลาย ๆ ออบเจกต์ ควรจะแยกออกมาเป็น Single Type Declaration และอ้างถึงคุณสมบัติเหล่านี้โดยใช้ชื่อ ถ้าระหว่างการกระทำบำรุงรักษาระบบ ทำให้คุณสมบัติมีการเปลี่ยนแปลง ก็ให้แก้ไขเฉพาะที่ Declaration นี้เท่านั้น ไม่ต้องตามไปแก้ไขในทุก ๆ Object Declaration ประโยชน์ของสิ่งนี้ คือ Mechanism ชื่อ Type Checker มีอยู่เพียง Mechanism เดียวเท่านั้น ซึ่งทำหน้าที่ในการรับผิดชอบการ ใช้ ออบเจกต์ และ คุณสมบัติของ ออบเจกต์ ได้อย่างถูกต้อง

Strongly Typed Object-Oriented Language มีความต้องการใช้เป็นอย่างมาก ในการพัฒนาระบบ และ โปรแกรมระบบงาน

2.6.2.3 การ Implement ADT ด้วย คลาส

ใน Object-Oriented Language ADT ถูก Implement เป็น Module ซึ่งปกติจะเรียกว่า คลาส. เช่น ใน C++ Class Definition มีรูปแบบดังนี้

```
Class Name
{
    Private Components
Public :
    Public Components
};
```

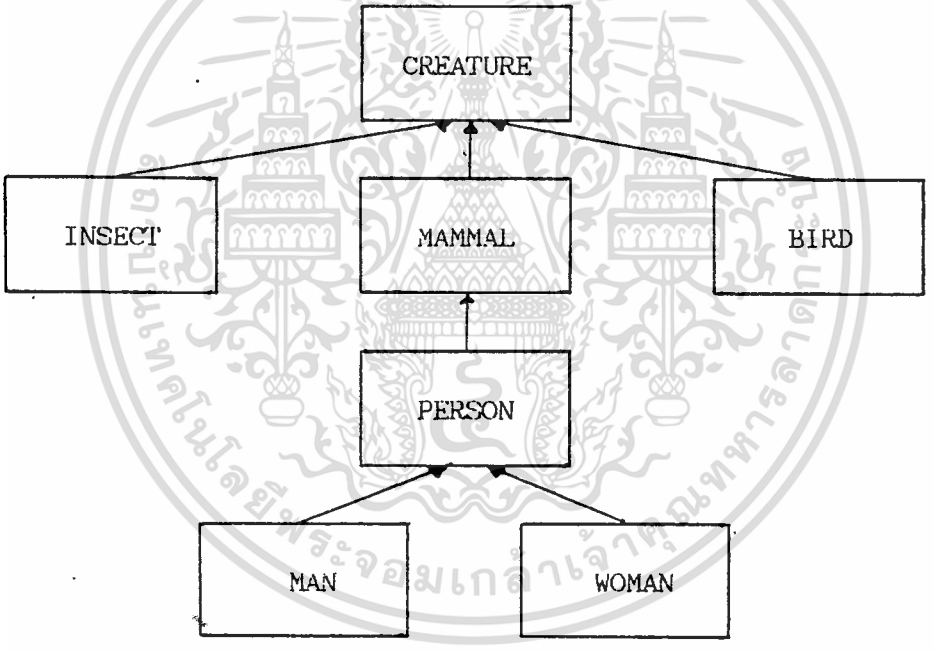
Private Components เป็น แอตตริบิวต์ และ ฟังค์ชัน ประเภทหนึ่ง ที่จำเป็นต้องใช้ในการ Implement คลาส และ ส่วนประกอบเหล่านี้ ไม่สามารถถูกเข้าถึง (Access) ได้โดย โปรแกรมอื่น ๆ ซึ่งใช้กับ คลาส นั้น

Public Components แสดงถึงการ Interface กับ ผู้ใช้ และ เป็น Component ของ คลาส ที่ ผู้ใช้สามารถอ้างถึงได้

Components เหล่านี้ อาจจะเป็น ฟังค์ชันใดฟังค์ชันหนึ่ง ดังต่อไปนี้ คือ ฟังค์ชัน ฟังค์ชันที่มีการส่งค่ากลับเป็นค่าของ แอตตริบิวต์, เป็น Constructor หรือ Destructor Function, Accessor Function หรือ Transformer Function

2.6.3 Inheritance (การสืบทอดคุณสมบัติ)

ในระบบเชิงวัตถุ แนวความคิดของ การสืบทอดคุณสมบัติทำให้ ออบเจกต์ ที่เป็น Specialized มีการสืบทอดคุณสมบัติ (Property) และ การดำเนินการ (Operation) ของออบเจกต์ ที่เป็น Generalized คลาส ของออบเจกต์ ซึ่งมีลักษณะคล้ายคลึงกัน มีการใช้คุณสมบัติ และ พังค์ชั่น บางอย่างร่วมกัน สามารถถูกออกแบบ ให้อยู่ในรูปของ ซุปเปอร์คลาส และจะทำให้ได้ ซุปเปอร์คลาส และจะทำให้ได้ Specialized คลาล (ซับคลาล) จาก ซุปเปอร์คลาส Feature เหล่านี้ได้มีการจัดเตรียมเพื่อสนับสนุนการนำกลับมาใช้ และ Extensibility เพราะว่า การนิยามของ ออบเจกต์ ที่ถูกกำหนดขึ้นใหม่ สามารถใช้ลักษณะพื้นฐานของ คลาส ที่มีอยู่แล้วได้



รูป 2.7 แสดง การสืบทอดคุณสมบัติแบบลำดับขั้น

จากรูป จะได้ว่า ออบเจกต์ คลาส Mammal, Bird และ Insect เป็น ซับคลาส ของ Creature ส่วน ออบเจกต์ คลาส Person เป็น ซับคลาส ของ Mammal, และ ออบเจกต์ คลาส Man, Woman เป็น ซับคลาส ของ Person ซึ่งจะสามารถกำหนด Class Definition ของ Hiererchy นี้ได้ในรูปแบบดังต่อไปนี้

```
Class Creature
      Properties
            Type : String ;
            Weight. : Real ;
            Habitat : (...Some Habitat Type...);
            ...
      Operations
            Create () -> Creature;
            Predators(Creature) -> Set(Creature);
            Life_Expectancy (Creature) -> Integer;
            ...
End Creature.

Class Mammal
     Inherit Creature;
     Properties
            Gestation_Period : Real;
     Opertions
            ...
End Mammal.

Class Person
     Inherit Mammal;
          Properties
               Surname, Firstname : String;
               Date_Of_Birth : Date;
               Origin : Country;
End Person.
```

```
Class Man
     Inherit Person;
     Properties
          Wife : Woman;
          ...
     Operations
          ...
End Man.

Class Woman
     Inherit Person;
     Properties
          Husband : Man ;
          Maodem_Name : String;
          ...
End Woman.
```

2.6.3.1 Inheritance For Extensibility(การสืบทอดคุณสมบัติเพื่อเพิ่มความสามารถ)

ในวิศวกรรมส่วนชุดคำสั่ง (Software Engineering) การสืบทอดคุณสมบัติ เป็น Mechanism ที่เกี่ยวข้องกับวิวัฒนาการของระบบ นั่นคือ Inheritance Mechanism อาจจะไม่ถูกใช้กับลักษณะเฉพาะเพียงอย่างเดียวเท่านั้น แต่ยังได้ถูกนำไปใช้สำหรับการขยาย Software Module เพื่อจัดเตรียมการดำเนินการ ให้เพิ่มมากขึ้น (Operation) อีกด้วย เช่น ถ้าเรามี คลาส (หรือ โมดูล) A ซึ่งมี ซับคลาส เป็น B แล้ว B จะมีการ Inherit Service จาก A ดังนั้น จึงอาจพิจารณาได้ว่า B เป็น Extension ของ A เพราะคุณสมบัติ และ การดำเนินการ ที่ใช้กับ Instance ของ A เป็น ซับเซ็ตของคุณสมบัติ และ การดำเนินการ ที่ใช้กับ Instance ของ B

คุณลมบัติดังกล่าวของการสืบทอดคุณสมบัติก่อให้เกิดประโยชน์อย่างมากในการสร้างระบบ Software ที่มีขนาดใหญ่ สำหรับงานทางด้านฐานข้อมูล การสืบทอดคุณสมบัติจะถูกใช้ประโยชน์เพื่อการจัดเตรียมความสะดวกในการออกแบบความสัมพันธ์ระหว่าง เอนทิตี ที่มี โครงสร้าง พฤติกรรม (Behaviour) เหมือนกัน

2.6.3.2 Inheritance And Polymorphism (การสืบทอดคุณสมบัติ และ โพลิมอร์ฟิซึม)

ใน ภาษาโปรแกรมเชิงวัตถุ โพลิมอร์ฟิซึม หมายถึงความสามารถของโปรแกรมหนึ่ง ๆ ในการอ้างถึง Instance ของ หลาย ๆ ชนิด หรือ คลาส ที่ Run-time รูปแบบอย่างง่าย ของ โพลิมอร์ฟิซึม ก็คือ Overloading ซึ่งยอมให้มีการใช้ชื่อเดียวกันได้มากกว่าหนึ่งเอนทิตี ในหนึ่งโปรแกรม โดยระบบจะทำการตรวจสอบ Context ของแต่ละ Occurrence Name Overloading ช่วยอำนวยความสะดวกในด้าน Syntax โดยยอมให้โปรแกรมเมอร์ สามารถ กำหนดชื่อเหมือนกันสำหรับการ Implement ที่แตกต่างกันของ การดำเนินการ ที่คล้ายคลึงกัน เช่น อาจกำหนดชื่อ "Add" ให้กับ การดำเนินการ (Operation) ซึ่งทำการบวกเลขจำนวนเต็ม 2 จำนวน, การบวกเลขจำนวนจริง 2 จำนวน และ Character String 2 จำนวน เมื่อการดำเนินการ (Operation) Add นี้ ถูกเรียกใช้โดย

Add(v1,v2)

ระบบจะตัดสินใจว่า ฟังค์ชันใด จะถูกใช้ในการบวก ข้อมูลโดยทำการตรวจสอบ ชนิด ของ v1 และ v2

ถ้า คลาส P เป็น Parent ของ ซับคลาส S แล้ว ออบเจกต์ ของ ซับคลาส S จะสามารถนำไปใช้ในตำแหน่งใดก็ได้ ที่ ออบเจกต์ ของ คลาส P สามารถใช้งานได้ ซึ่ง แสดงถึงความหมายโดยนัยว่า ชุดของ เมสเสจ หนึ่ง สามารถส่งให้ทั้ง ออบเจกต์ ที่อยู่ใน คลาส P และ ออบเจกต์ ที่อยู่ใน คลาส S และ เรียกคุณสมบัติในการส่ง เมสเสจ ดังกล่าวนี้ว่า โพลิมอร์ฟิซึม

2.6.3.3 Inheritance And Dynamic Binding (การสืบทอดคุณสมบัติ,ไดนามิก บายดิง)

การดำเนินการ (Operation) ซึ่งถูกนิยามสำหรับ คลาส และ ซับคลาส ที่อยู่ในลำดับ ชั้นหนึ่ง มักจำเป็นต้องถูก Implement ให้เหมือนกันในแต่ละ คลาส หลาย ๆ ภาษาที่เป็น ออบเจกต์ ได้จัดเตรียมความสามารถในการ Redefine Operation ซึ่งได้รับการสืบทอด มาจาก ซุปเปอร์คลาส ชื่อของ การดำเนินการนั้นยังคงใช้ชื่อเดิม แต่สำหรับการ Implement การดำเนินการของ Specialzed คลาส ต่าง ๆ จะถูกปรับปรุงขึ้นมาใหม่ เช่น มมุติให้ ตัวแปร C ใน Program หนึ่งอ้างถึง ออบเจกต์ ของ Type Creature ดังนั้น Function Call

x := c.life_expectancy

เป็นการร้องขอ การดำเนินการ ที่ถูกกำหนดไว้ใน คลาส Creature โดยจะมีความ หมายเช่นเดียวกันกับที่ ตัวแปร P อ้างถึง ออบเจกต์ ของ Type Person แล้ว Call

```
x := p.life_expectancy
```

เป็นการร้องขอ การดำเนินการ ที่ได้ Redefine ใน คลาส Person อย่างไรก็ตาม ได้มีภาวะซึ่งตัวแปร C (ที่เป็น Polymorphic และถูก Declare เหมือนกับ Creature) อ้างถึง ออบเจกต์ Person อย่าง Dynamic นั่นคือ

```
c := p;
...
x := c.life_expectancy
```

เทคนิคที่ถูกใช้นี้ รู้จักกันในลักษณะของ Dynamic Binding กระบวนการจะถูกกำหนดโดย Dynamic Form ของออบเจกต์ ที่ Run-Time การใช้ Dynamic Binding ในกรณีที่จะทำให้มีการ Redefine Version ของ การดำเนินการ Life_Expectancy ในคลาส person

2.7 เปรียบเทียบ OODB กับ RDB

ข้อเปรียบเทียบระหว่าง RDB กับ OODB อาจจะสรุปภายใต้หัวข้อต่าง ๆ ดังนี้คือ Data Type, Data Integrity และ Schema Evolution

2.7.1 Data Type

ในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ มีแต่ชนิดของรีเลชันเท่านั้น สำหรับการจัดโครงสร้างของข้อมูล โดยนิยามแล้ว แอตตริบิวต์ ของนอร์มอไลซ์ รีเลชัน ไม่สามารถแยกออกเป็นส่วนย่อย ๆ ได้อีก และมี ชนิดของข้อมูลที่ค่อนข้างง่าย เช่น Integer, Real, String, Data. ฯลฯ การดำเนินการบนรีเลชันถูกจำกัดอยู่ที่ การเรียกดู และ การแก้ไขทัปเปิล (Tuples) ซึ่งถูกกำหนดโดยค่าของแอตตริบิวต์

คุณสมบัติของ คลาส ใน OODB ไม่จำเป็นที่จะต้องเป็นชนิดข้อมูลง่าย ๆ แต่สามารถอ้างถึง คลาส ของ ชนิดข้อมูลที่ซับซ้อนอื่น ๆ ได้ โดยปกติแล้ว ระบบเชิงวัตถุ จะจัดให้มีของเขตของการ Implement คลาส ซึ่งชนิดข้อมูลที่ถูกใช้บ่อย ๆ ผู้ใช้สามารถปรับ คลาส เหล่านี้ตามความต้องการของผู้ใช้เอง ในการใช้งาน โดยใช้ Genericity และ Inheritance ที่สามารถหามาได้สะดวก

2.7.2 Data Integrity

ฐานข้อมูลเชิงสัมพันธ์ไม่สามารถแสดง การควบคุมความถูกต้องด้วยการเก็บ Semantic ได้มากว่า Referential Integrity แบบตรงไปตรงมา ยกตัวอย่างเช่น ฐานข้อมูลเชิงสัมพันธ์ไม่สามารถที่จะแสดงความสัมพันธ์ ซึ่งเป็น One-To-One หรือ One-To-Many ได้ คอนสเทรน ดังกล่าวจะต้องถูกสร้างเก็บไว้ใน Application Codeที่ใช้ฐานข้อมูลเชิงสัมพันธ์ เพราะว่าโดยทั่วไปแล้ว Code เหล่านี้ไม่ได้ถูกใช้ร่วมกันระหว่าง Application ทั้งหมด จึงเป็นการยากที่จะมั่นใจว่าข้อมูลจะถูกแก้ไขให้มีความคงสภาพ(Consistent) อยู่ตลอดเวลา

ในแบบจำลองเชิงวัตถุ ใช้ คลาส นิยาม ดาต้าแอบแทรกชัน ซึ่ง Abstraction นี้ ได้รวบรวมรายละเอียดของ การดำเนินการ (Methods) ที่ถูกนำไปใช้กับ Instances ของ คลาส โดยนิยามของฐานข้อมูลในความหมายของ Abstraction ดังกล่าว มีความเป็นอิสระของข้อมูลสูงมาก นั่นก็คือ เป็นไปได้ที่จะเปลี่ยนวิธีการในการ Implement คลาส โดยปราศจากผลกระทบต่อ คลาส หรือ Transactions อื่นที่ใช้ Abstraction ดังนั้น Operation ซึ่งถูกกำหนดขึ้นโดยผู้ใช้สามารถถูกสร้างและเก็บไว้ในฐานข้อมูล การดำเนินการหนึ่งบน ออบเจกต์ สามารถเกิดขึ้นใน หลาย ๆ ฐานข้อมูล การดำเนินการ และ ทุก ๆ Transection ที่ร้องขอ การดำเนินการ จะใช้ Code เหมือนกัน (Code อยู่ในรูปของ Procedure ที่เก็บอยู่ใน Class Definition) ซึ่งเป็นข้อได้เปรียบอย่างมากสำหรับความถูกต้องของข้อมูล เนื่องจากการตรวจสอบ ความสอดคล้อง (Consistency) อาจถูกรวมเข้าไว้ใน Procedures และ โดยการเป็น การดำเนินการ เหมือนกับเป็นส่วนหนึ่งของฐานข้อมูล ดังนั้นส่วนใหญ่ของ Application Code จึงอยู่ภายใต้การควบคุมของผู้ควบคุมฐานข้อมูล (Database Administrator)และสามารถถูกจัดการโดย Database Management Utilities ทั้งหมด (กล่าวคือ มีความสะดวกในการทำ Concurrency Control, Recovery, Version Control และ Security)

Entity Integrity จะถูกจัดการด้วยวิธีการที่แตกต่างกันใน Object-Oriented Systems ทุก ๆ ออบเจกต์ (Class Instantiations) มี Unique Identity และออบเจกต์ อื่นสามารถอ้างถึง Identity นั้นได้ออบเจกต์ยังคงรักษา Indentity ของมันไว้ได้ตลอดการเปลี่ยน สภาวะ ของมัน ซึ่งขัดกับ แบบจำลองเชิงสัมพันธ์ ที่คุณสมบัติของ เอนทิตี จะต้องมีอย่างเพียงพอ เพื่อจะแยกความแตกต่างของ Entity หนึ่งออกจาก เอนทิตี อื่น ๆ ทั้งหมด นั่นก็คือ ต้องมีอย่างน้อยหนึ่ง แอตตริบิวต์ ขึ้นไปซึ่งค่าของ แอตตริบิวต์ ต้อง Unique และ คุณสมบัติไม่เปลี่ยนแปลง มักจะไม่แสดงใน Real World และ มักมีความจำเป็นสำหรับฐานข้อมูลเชิงสัมพันธ์ ในการทำ Artificial Identifiers สำหรับ เอนทิตี

2.7.3 Schema Evolution

ระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ มีแนวโน้มที่จะอำนวยความสะดวกอย่างมีขอบเขตจำกัดสำหรับการขยายหรือการแก้ไขโครงสร้างข้อมูลเดิม ตัวอย่าง เช่น ระบบฐานข้อมูลเชิงสัมพันธ์ ส่วนมากยอมให้มี Dynamic Creation, Dynamic Deletion ของ รีเลชัน และการเพิ่ม คอร์สัม ใหม่ลงใน รีเลชัน การเปลี่ยนแปลงชนิดของโดเมน มักจะเป็นการ Rewrite ของรีเลชันที่เกี่ยวข้อง และ เปลี่ยน Application Programs ที่ใช้รีเลชันเหล่านั้น เนื่องจากความจริงที่ว่า โครงสร้างข้อมูล ประกอบด้วย Database Schema และ Application Programs ที่เกี่ยวข้องกันเพียงเล็กน้อย

ความเกี่ยวข้องกันอย่างมาก (Tight Coupling) ระหว่างโปรแกรม และ ข้อมูลในแบบจำลองเชิงวัตถุ แสดงถึงขอบเขตที่กว้างกว่าสำหรับการเปลี่ยนแปลง Schema โดยการขยาย และ การ Refine ของ โครงสร้างข้อมูล ที่มีอยู่เดิมรวมไปถึง ประสิทธิภาพในการนำ Application Code กลับมาใช้ใหม่ ความมีประสิธิภาพที่มากกว่า ของ แบบจำลองข้อมูลอนุญาติให้ Semantic ของ Schema Evolution ถูก Define และถูกทำให้ ถูกต้องอย่างเข็มงวด (Rigidly)

บทที่ 3
ไนแอม$^{++}$ (NIAM^{++})

ไนแอม$^{++}$ เป็นนิยามข้อมูลระดับแนวความคิด (Conceptual Schema Model) ที่พัฒนามาจาก ไนแอม ซึ่งเป็นนิยามข้อมูลระดับแนวความคิดที่ถูกเลือกว่าเหมาะสม และ มีการ Represent ในรูปของ Graph ที่ง่ายต่อการอ่านและเข้าใจในขั้นตอนการออกแบบ

3.1 ไนแอม$^{++}$ (NIAM^{++})

พิจารณาสัญลักษณ์ของแบบจำลองข้อมูล ไนแอม มีแต่แนวความคิดของ ซุปเปอร์ไทพ์ และ ซับไทพ์ ซึ่งสิ่งที่เกี่ยวข้องในชีวิตประจำวันส่วนใหญ่มักจะเป็น คอมเพลคซ์ ออบเจกต์ ที่มีความสัมพันธ์แบบ ซุปเปอร์-พาร์ท-ออฟ และ ซับ-พาร์ท-ออฟ คือ ระบบใหญ่จะประกอบด้วยหลายส่วนย่อย ๆ และ สิ่งที่สำคัญอีกอย่างหนึ่งก็คือ ไนแอม ไม่สามารถแยก เอนทิตี ไทพ์ กับ เอนทิตี ไทพ์ คลาส ให้เห็นเด่นชัดได้ ดังนั้นจึงมี แนวความคิดที่จะเพิ่ม สัญลักษณ์บางประการเข้าไปใน ไนแอม เพื่อใช้ในการออกแบบฐานข้อมูลให้มีประสิทธิภาพมากยิ่งขึ้น ดังนั้นจึงเสนอ ไนแอม$^{++}$ และสัญลักษณ์ที่เพิ่มขึ้นมาดังนี้

รูป 3.1 สัญลักษณ์ที่เพิ่มขึ้นมาของ ไนแอม$^{++}$

รูป 3.1(a) แสดง เอนทิตี ไทพ์ คลาส PART ซึ่งมีสมาชิกเป็น PART แต่ละชนิด และแสดงได้ด้วย PART# (อาจมี PART ชนิดเดียวกัน จำนวนมากที่ใช้ PART# เดียวกัน)

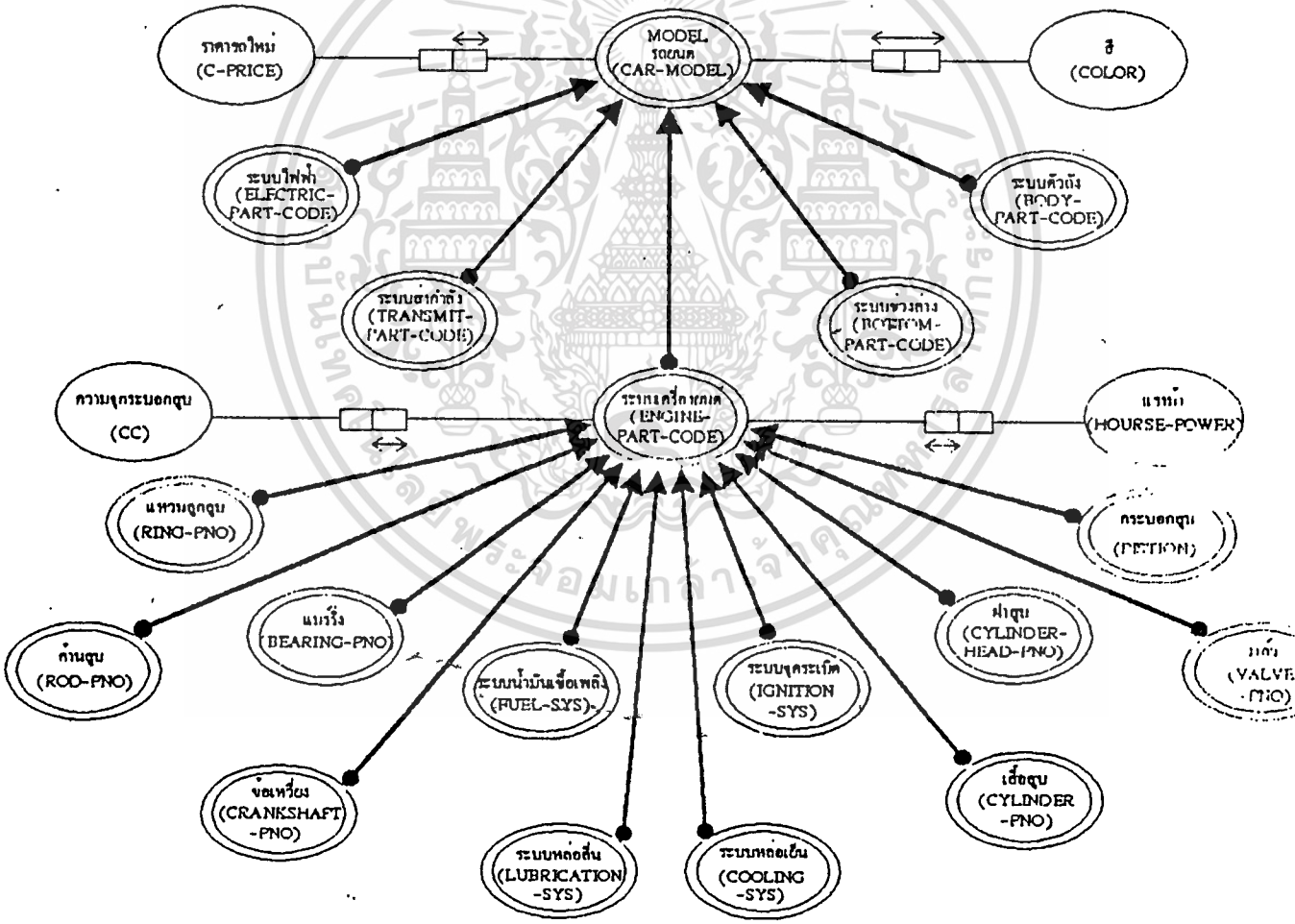
รูป 3.1(b) แสดง เอนทิตี ไทพ์ PART ซึ่งมีสมาชิกเป็น ตัว PART แต่ละชิ้น และแสดงได้ด้วย SERIAL# บนตัว PART แต่ละชิ้นซึ่งจะไม่ซ้ำ

รูป 3.1(C) A เป็น ซุปเปอร์-พาร์ท-ออฟ ของ B และ C โดยที่ B, C เป็น ซับ-พาร์ท-ออฟ ของ A

ตัวอย่าง คอมเพลคซ์ ออบเจกต์ เช่น รถยนต์ ซึ่งรถยนต์ประกอบด้วยส่วนประกอบดังนี้

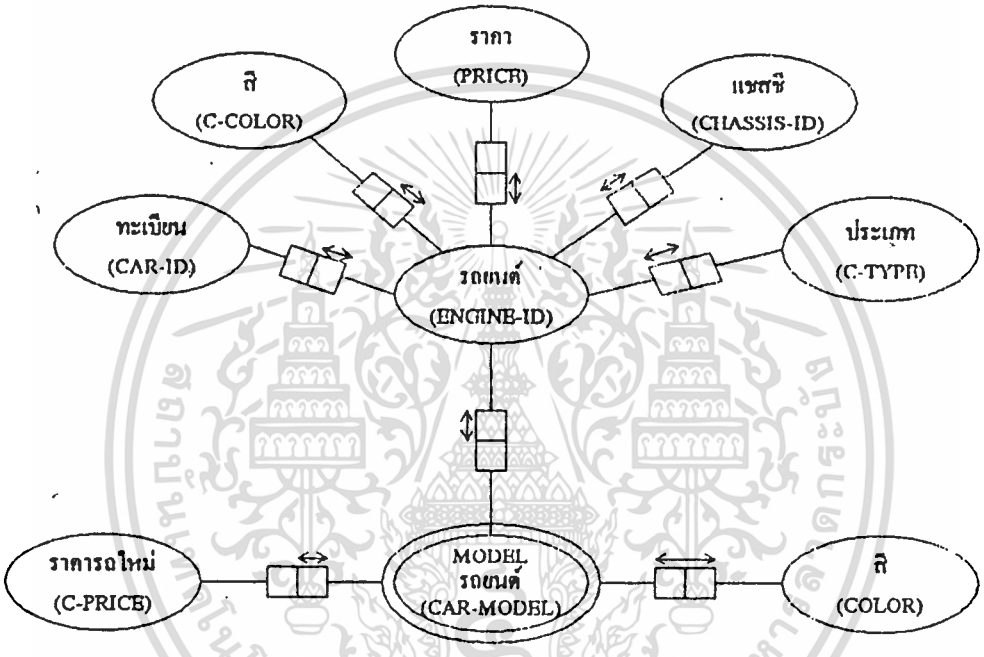
1. ระบบไฟฟ้า
2. ระบบส่งกำลัง
3. ระบบเครื่องยนต์
4. ระบบช่วงล่าง
5. ระบบตัวถัง

จะเห็นได้ว่า รถยนต์จะต้องประกอบด้วยส่วนประกอบพื้นฐานทั้ง 5 ระบบนี้ รถยนต์ถึงจะทำงานได้ ซึ่งรถยนต์มีความสัมพันธ์แบบ แอกกริเกชัน (Aggregation) สามารถใช้ไนแอม$^{++}$ ช่วยในการออกแบบได้ ดังรูปที่ 3.2



รูปที่ 3.2 แสดง ชนิดความสัมพันธ์แบบแอกกริเกชัน

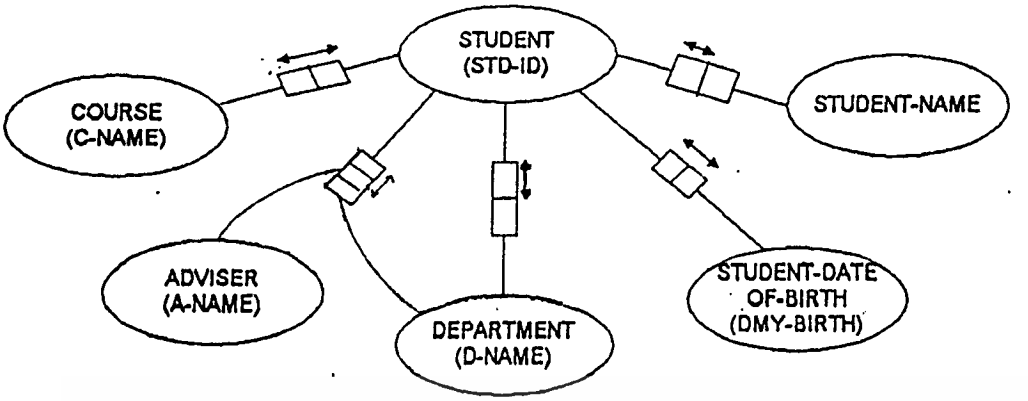
จากรูปที่ 3.2 จะเห็นว่า เอนทิตี ไทพ์ คลาส MODELรถยนต์ เป็นการระบุถึงแบบหรือรุ่นของรถยนต์ ซึ่งหมายถึงรถยนต์หลาย ๆ คัน เช่น รถยนต์ DATSUN รุ่น 120Y ก็จะหมายถึง รถยนต์ DATSUN รุ่น 120Y ทุกคัน (ไม่ได้ชี้เฉพาะลงไปถึง รถยนต์ DATSUN รุ่น 120Y คันใดคันหนึ่งโดยเฉาะ) ซึ่งถ้าพิจารณาถึงรถยนต์ในเต้นท์ขายรถยนต์ใช้แล้ว จะพิจารณาถึงรถยนต์เป็นคัน ๆ ไป โดยที่รถยนต์แต่ละคันอาจจะมีรายละเอียดที่เกี่ยวข้อง เช่น ทะเบียน, สี, ประเภท, ราคา, หมายเลขแชสซี สามารถใช้ ไนแอม$^{++}$ ออกแบบได้ ดังรูปที่ 3.3



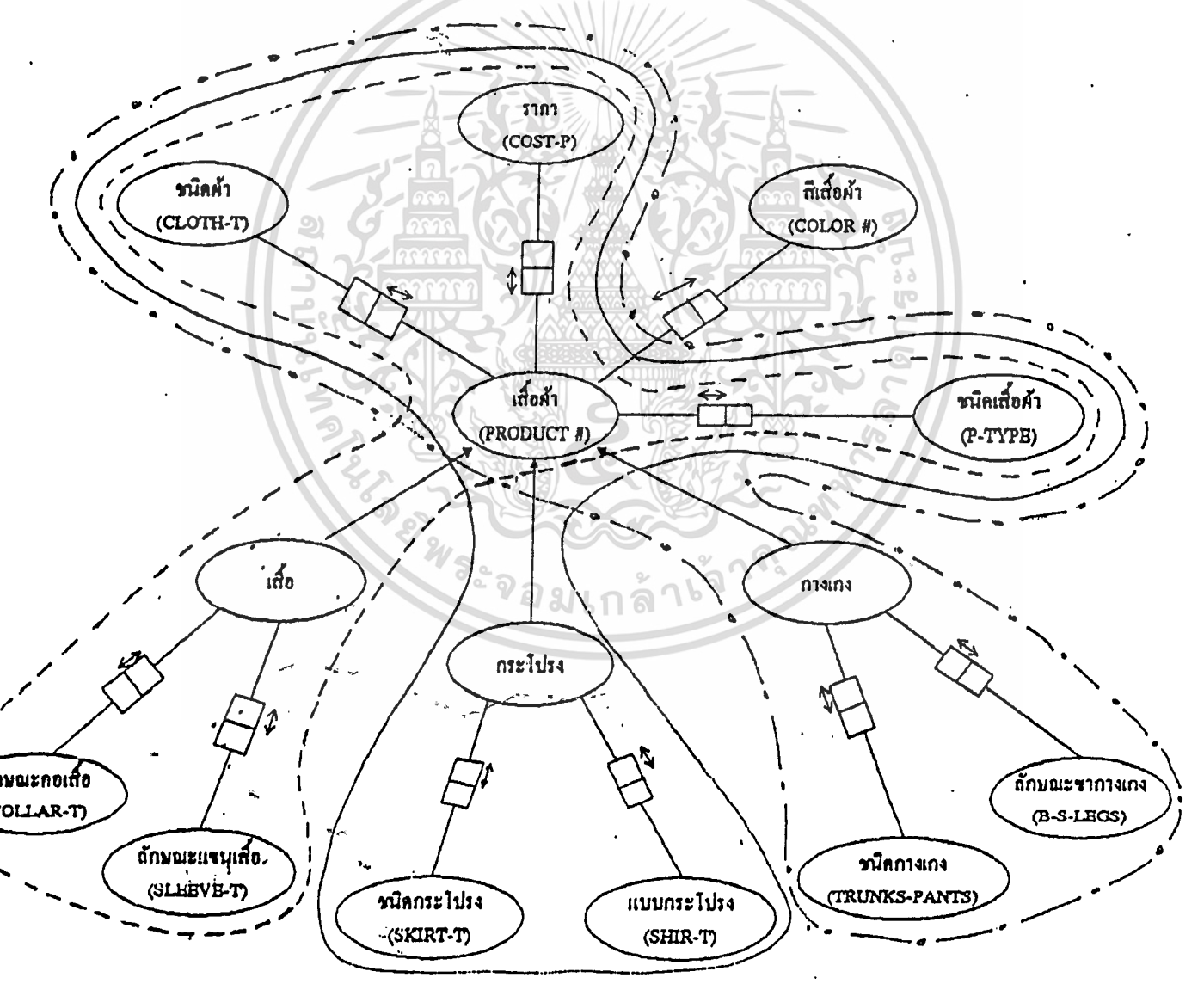
รูปที่ 3.3 ไนแอม$^{++}$ แสดงความแตกต่างระหว่าง
เอนทิตี ไทพ์ กับ เอนทิตี ไทพ์ คลาส

จากรูปที่ 3.2 และ 3.3 จะสามารถเห็นความแตกต่างอย่างเด่นชัดระหว่าง เอนทิตี ไทพ์ คลาส MODELรถยนต์ ซึ่งหมายถึงรถยนต์หลาย ๆ คัน ที่มีแบบเดียวกัน มีส่วนประกอบต่าง ๆ เหมือนกัน แต่ เอนทิตี ไทพ์ รถยนต์ หมายถึงรถยนต์แต่ละคันที่มี ทะเบียน, ราคา, สี, หมายเลขแชสซี, ประเภทรถ ต่างกันเป็นคัน ๆ ไป

จะเห็นได้ว่าถ้าใช้ไนแอมออกแบบระบบดังรูปที่ 3.2 และ 3.3 จะไม่สามารถแสดงความแตกต่างระหว่าง เอนทิตี ไทพ์ คลาส กับ เอนทิตี ไทพ์ ได้อย่างชัดเจน และไม่สามารถแสดงความสัมพันธ์แบบแอกกริเกชันได้



รูปที่ 3.4 แสดง ชนิดความสัมพันธ์แบบแอกกริเกชัน



รูปที่ 3.5 แสดง ชนิดความสัมพันธ์แบบเจนเนอรัลไลเซชัน

ดังนั้น ไนแอม$^{++}$จึงเป็นการทำให้ ไนแอม มีความสมบูรณ์มากยิ่งขึ้นคือ สามารถจะแทน (Represent) สภาพที่เป็นจริงใน Real World ได้มากขึ้นเท่าที่มากได้

3.2 การแปลงจาก ไนแอม$^{++}$ เป็น โครงสร้างข้อมูลแบบเชิงสัมพันธ์

(Transformation from NIAM^{++} to Relational Database Schema)

1. ในกรณีของ เอนทิตี ไทพ์ และ เอนทิตี ไทพ์ คลาส ธรรมดา

1.1 สร้าง 1 รีเลชัน (relation) สำหรับการรวม หลาย ๆ ไบนารี แฟคท์ ไทพ์ ที่มี อินตร้า ยูนิกเนส คอนสเทรน คลุมเพียง 1 Role การรวมจะพิจารณาทุก ไบนารี แฟคท์ ไทพ์ ที่มี ยูนิกเนส คอนสเทรน ที่อยู่ฝั่งเดียวกันกับ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่กำลังพิจารณาโดยนำ เลเบล ไทพ์ ของ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่พิจารณามาเป็น คีย์ และนำ เลเบล ไทพ์ ของฝั่งตรงข้ามมาเป็น แอตตริบิวต์

ตัวอย่าง เช่นรูปที่ 3.4 โดยพิจารณา เอนทิตี ไทพ์ STUDENT สามารถนำมาสร้างรีเลชันได้ดังรูปที่ 3.6 โดยมี STD-ID เป็น คีย์ ของรีเลชัน

STD-ID	STD-NAME	DMY-BIRTH	D-NAME

รูปที่ 3.6 รีเลชันที่ได้จากการแปลง รูปที่ 3.4 โดยใช้กฎข้อ 1.1

1.2 สร้าง 1 รีเลชันสำหรับแต่ละ n-nary Fact Type (n>2) ที่มี อินตร้า แฟคท์ ยูนิกเนส คอนสเทรน คลุมเพียง n-1 Role แต่ละ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ของรีเลชัน โดยที่ อินตร้า แฟคท์ ยูนิกเนส คอนสเทรน คลุม Role ไหน เลเบล ไทพ์ ของ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส นั้นๆ จะกลายเป็น คีย์

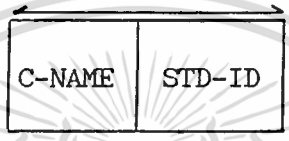
ตัวอย่าง เช่นรูปที่ 3.4 พิจารณา n-ary แฟคท์ ไทพ์ ที่มี n=3 สามารถนำมาสร้างรีเลชัน ได้ดังรูปที่ 3.7 โดยมี A-NAME และ STD-ID เป็น คีย์ ของรีเลชัน

A-NAME	STD-ID	D-NAME

รูปที่ 3.7 รีเลชันที่ได้จากการแปลง รูปที่ 3.4 โดยใช้กฎข้อ 1.2

1.3 สร้าง 1 รีเลชันสำหรับแต่ละ อินตร้า ยูนิกเนส คอนสเทรนคลุม n Roles (many to many) โดยที่ เลเบล ไทพ์ ของแต่ละ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์e คลาส ที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ และทุก แอตตริบิวต์ จะเป็น Null ไม่ได้ เพราะว่าทุก แอตตริบิวต์ รวมกันเป็น คีย์

ตัวอย่าง เช่นรูปที่ 3.4 พิจารณา n-nary Fact Type ที่มี n=2 คลุมทั้ง 2 Role สามารถนำมาสร้าง รีเลชันได้ดังรูปที่ 3.8 โดยมี C-NAME และ STD-ID เป็น คีย์ ของรีเลชัน



รูปที่ 3.8 รีเลชันที่ได้จากการแปลง รูปที่ 3.4 โดยใช้กฎข้อ 1.3

1.4 ถ้า เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ถูกควบคุมด้วย แมนดาทอรี คอนสเทรน แล้วแอตตริบิวต์ ฝั่งตรงข้ามจะเป็น Null ไม่ได้

2. ในกรณี Entity Type หรือ Entity Type Class ที่เป็น Subtype heirarchy (Generalization)

แยกพิจารณา ซุปเปอร์ไทพ์ และ ซับไทพ์ ทีละคู่ โดยการนำ แฟคท์ ไทพ์ ของ ซับไทพ์ ตัวที่กำลังพิจารณาไปเกาะกับ ซุปเปอร์ไทพ์ แล้วพิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

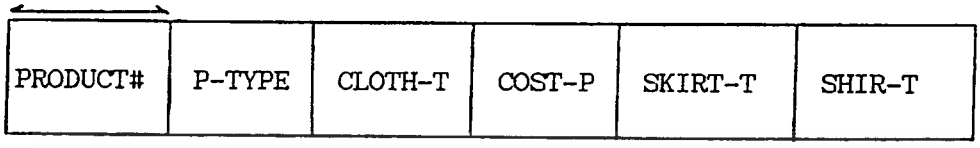
ตัวอย่าง เช่นรูปที่ 3.5 โดยพิจารณา ดังนี้

1. เอนทิตี ไทพ์ เสื้อผ้า ซึ่งเป็น ซุปเปอร์ไทพ์ คู่กับ เอนทิตี ไทพ์ เสื้อ ซึ่งเป็น ซับไทพ์ สามารถนำมาสร้างเป็น รีเลชันได้ดังที่ รูป 3.9 โดยมี PRODUCT# เป็น คีย์ ของรีเลชัน

PRODUCT#	P-TYPE	CLOTH-T	COST-P	COLLAR-T	SLEEVE-T

รูปที่ 3.9 รีเลชันที่ได้จากการแปลง รูปที่ 3.5 (a) โดยใช้กฎข้อ 2

2. เอนทิตี ไทพ์ เสื้อผ้าเป็น ซุปเปอร์ไทพ์ คู่กับ เอนทิตี ไทพ์ กระโปรง ซึ่งเป็น ซับไทพ์ สามารถนำมาสร้าง รีเลชัน ได้ดังรูปที่ 3.10 โดยมี PRODUCT# เป็น คีย์ ของรีเลชัน

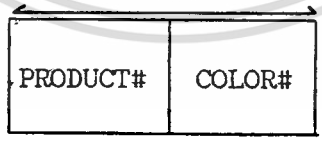


รูปที่ 3.10 รีเลชันที่ได้จากการแปลง รูปที่ 3.5 (b) โดยใช้กฎข้อ 2

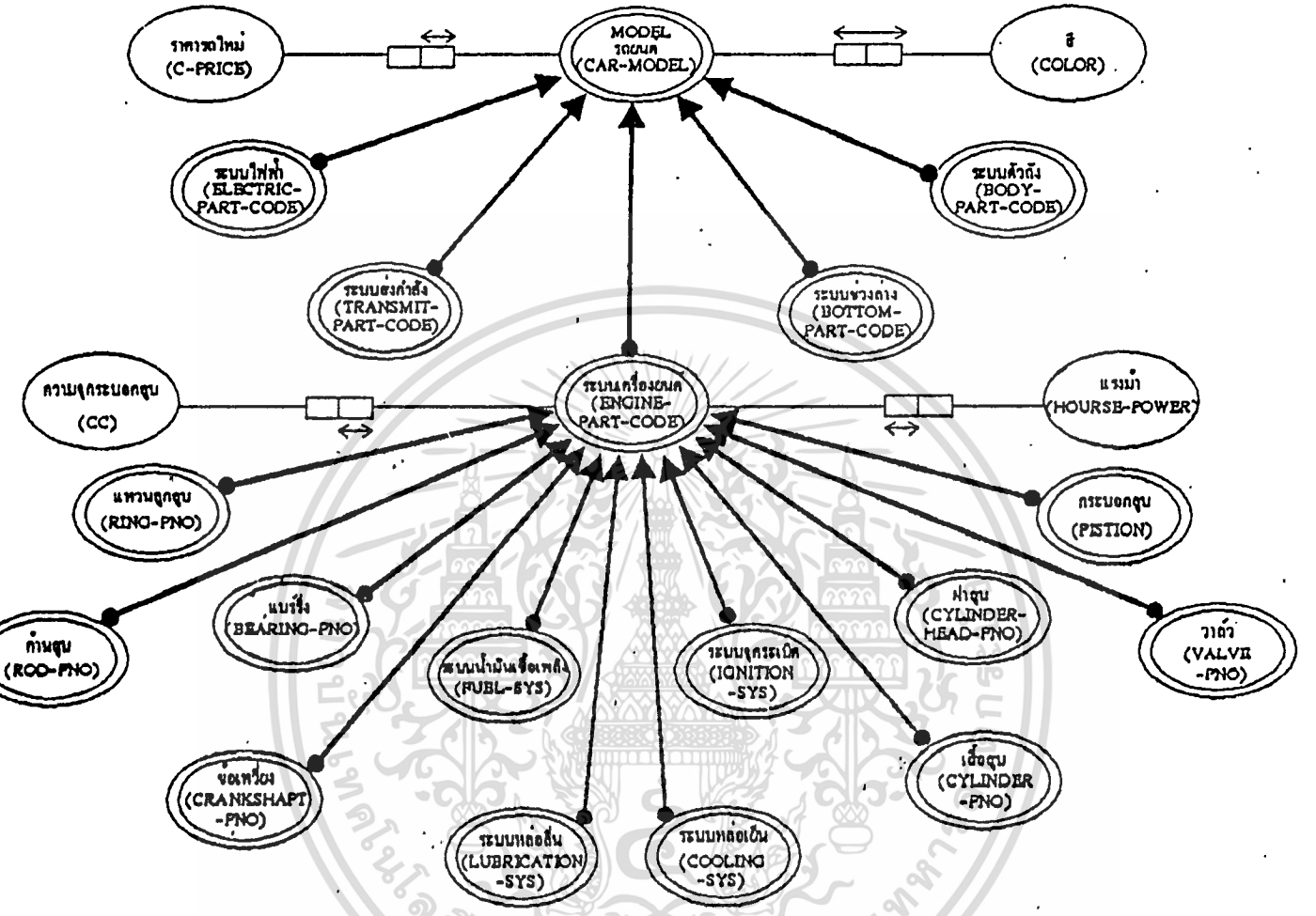
3. เอนทิตี ไทพ์ เสื้อผ้าซึ่งเป็น ซุปเปอร์ไทพ์ คู่กับ เอนทิตี ไทพ์ กางเกง ซึ่งเป็น ซับไทพ์ สามารถนำมาสร้าง รีเลชัน ได้ดังรูปที่ 3.11 โดยมี PRODUCT# เป็น คีย์ ของรีเลชัน

รูปที่ 3.11 รีเลชันที่ได้จากการแปลง รูปที่ 3.5 (c) โดยใช้กฎข้อ 2

4. พิจารณา เอนทิตี ไทพ์ ที่เหลือสามารถนำมาสร้างรีเลชันได้ ดังรูปที่ 3.12 โดยมี PRODUCT# และ COLOR# เป็น คีย์ ของรีเลชัน



รูปที่ 3.12 รีเลชันที่ได้จากการแปลง รูป 3.5 (d) โดยใช้กฎข้อ 2

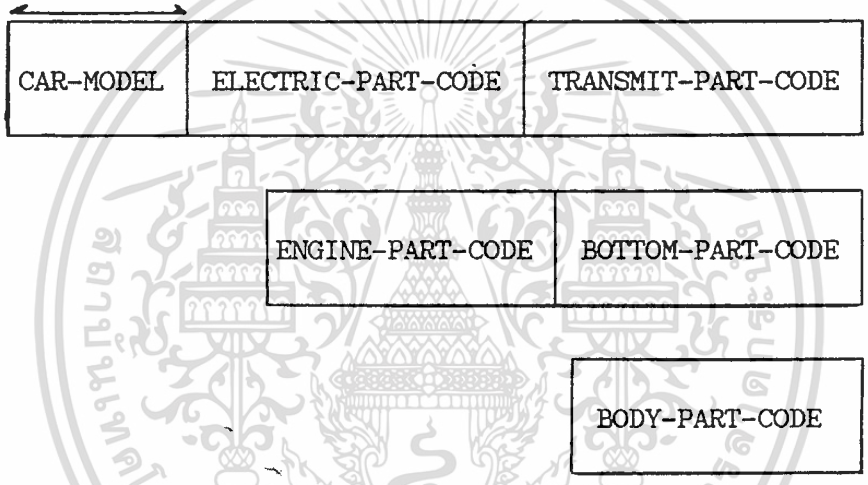


รูปที่ 3.13 แสดง ชนิดความสัมพันธ์แบบแอกกริเกชัน

3. ในกรณี คอมเพลดซ์ ออบเจกต์ ที่เป็น ซุปเปอร์-พาร์ท-ออฟ และ ซับ-พาร์ท-ออฟ (Aggregation)

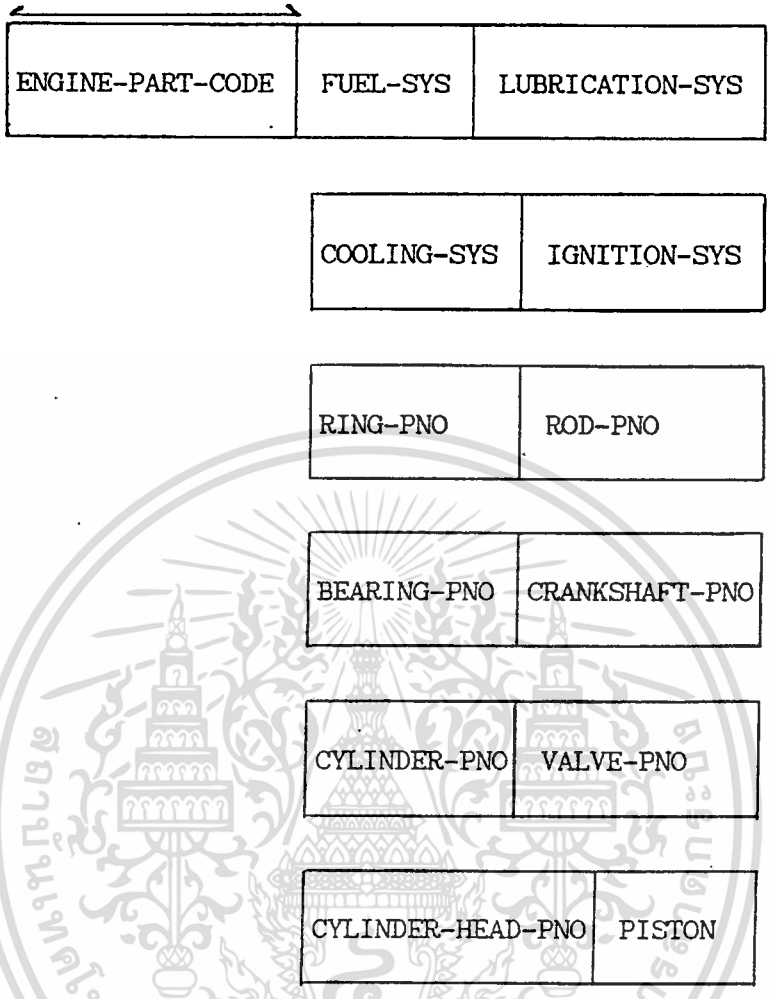
3.1 สร้าง 1 รีเลชันสำหรับแต่ละ ซุปเปอร์-พาร์ท-ออฟ โดยนำ เลเบล ไทพ์ ของ ซุปเปอร์-พาร์ท-ออฟ มาเป็น คีย์ และซับ-พาร์ท-ออฟ ใด ที่เป็นองค์ประกอบของ ซุปเปอร์-พาร์ท-ออฟ ให้นำ ซับ-พาร์ท-ออฟ นั้น มาเป็น แอตตริบิวต์ ของ รีเลชัน

ตัวอย่าง เช่นรูปที่ 3.13 โดยพิจารณา เอนทิตี ไทพ์ คลาส MODEL รถยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ สามารถนำมาสร้าง เป็นรีเลชัน ได้ดังรูปที่ 3.14 โดยมี CAR-MODEL เป็น คีย์ ของรีเลชัน



รูปที่ 3.14 รีเลชันที่ได้จากการแปลง รูปที่ 3.13 โดยพิจารณา เอนทิตี ไทพ์ คลาส MODEL รถยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟโดยใช้กฎข้อ 3.1

และพิจารณา เอนทิตี ไทพ์ คลาส ระบบเครื่องยนต์ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ สามารถนำมาสร้างเป็นรีเลชัน ได้ดังรูปที่ 3.15 โดยมี ENGINE-PART-CODE เป็น คีย์ ของรีเลชัน

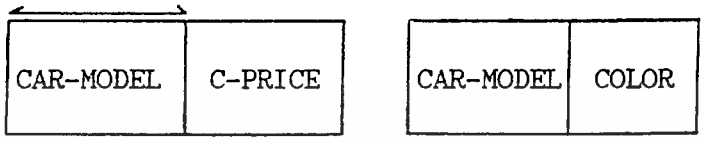


รูปที่ 3.15 รีเลชันที่ได้จากการแปลง รูปที่ 3.13 โดยพิจารณา เอนทิตี ไทพ์ คลาส ระบบเครื่องยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ โดยใช้กฎข้อ 3.1

จากรูปที่ 3.14 และรูปที่ 3.15 ไม่สามารถบอกได้ว่า เป็นโครงสร้างของ ซุปเปอร์-พาร์ท-ออฟ หรือโครงสร้างของเรคอร์ดธรรมดา เนื่องจาก RDB ไม่สามารถแสดงความหมาย (Semantic) ได้ดีเท่ากับ Semantic ระดับสูง เช่น ไนแอม++

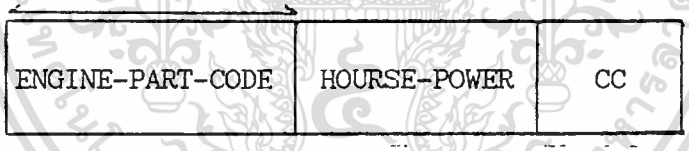
3.2 แยกพิจารณา ซุปเปอร์-พาร์ท-ออฟ และ ซับ-พาร์ท-ออฟ แต่ละตัวเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

ตัวอย่าง เช่นรูปที่ 3.13 พิจารณา เอนทิตี ไทพ์ คลาส MODELรถยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ สามารถนำมาสร้างได้ 2 รีเลชัน ดังรูปที่ 3.16 โดยมี CAR-MODEL เป็น คีย์ ของรีเลชั่นที่หนึ่ง มี CAR-MODEL และ COLOR เป็น คีย์ ของรีเลชัน ที่สอง ตามลำดับ

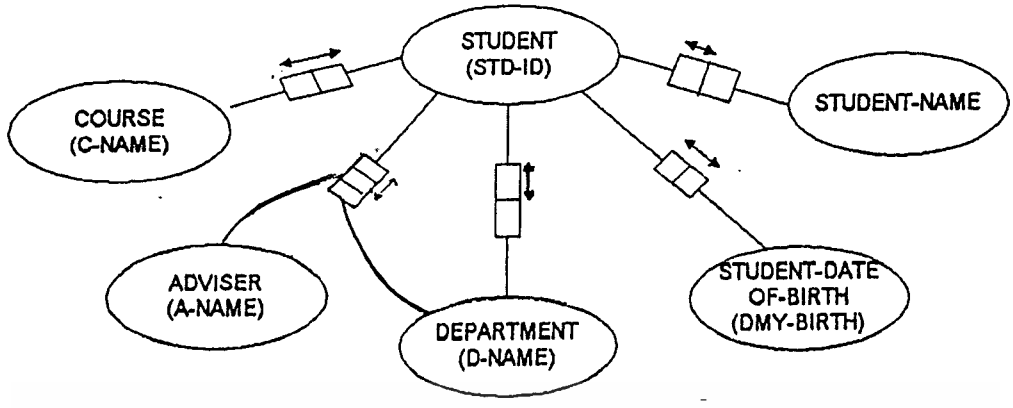


รูปที่ 3.16 รีเลชันที่ได้จากการแปลงรูปที่ 3.13 โดยพิจารณา เอนทิตี ไทพ์ คลาส MODELรถยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ โดยใช้กฎข้อ 3.2

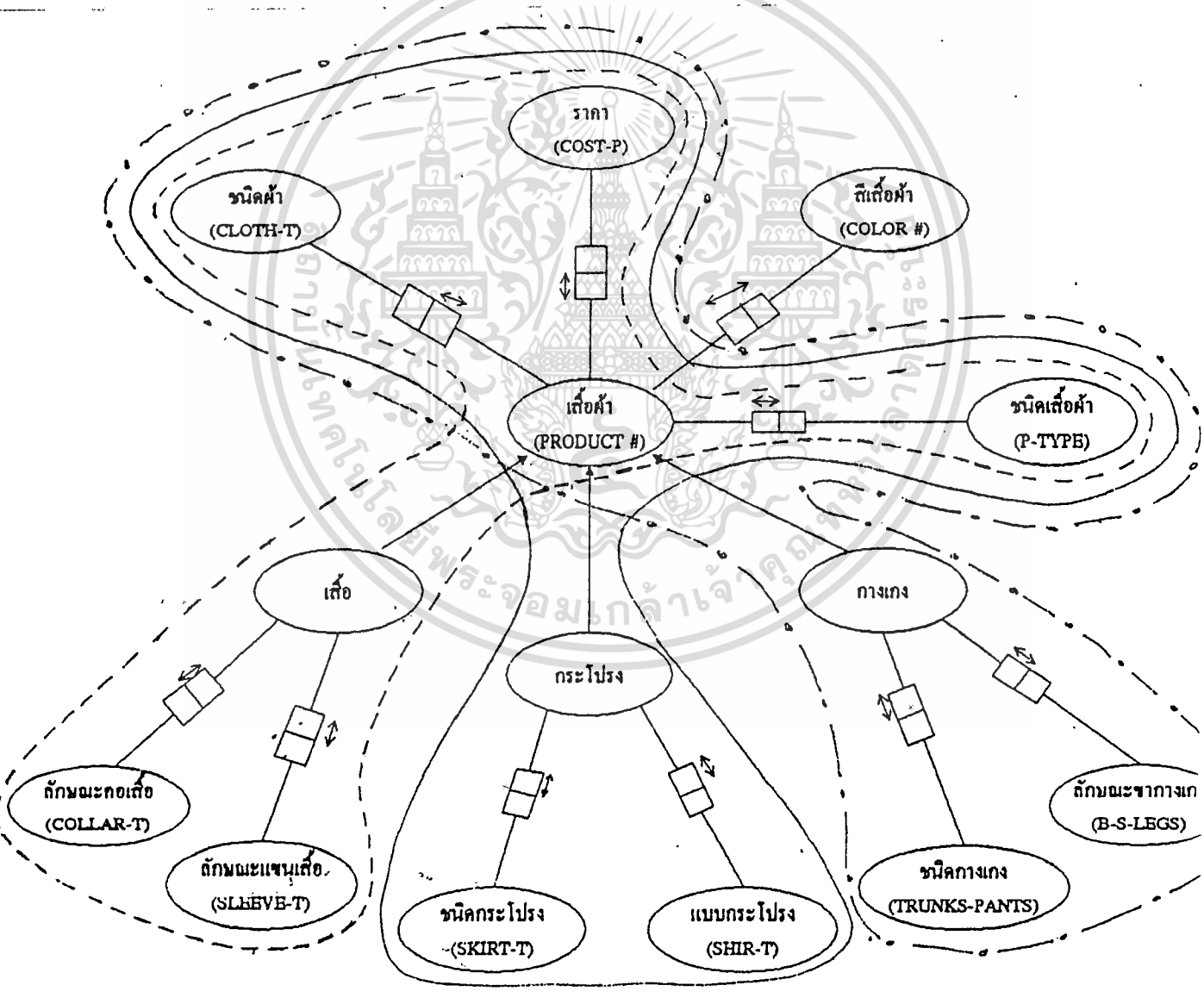
และพิจารณา เอนทิตี ไทพ์ ระบบเครื่องยนต์ ซึ่งเป็น ซับ-พาร์ท-ออฟ ของ เอนทิตี ไทพ์ MODELรถยนต์ และในขณะเดียวกัน เอนทิตี ไทพ์ ระบบเครื่องยนต์ ก็เป็น ซุปเปอร์-พาร์ท-ออฟ ของ เอนทิตี ไทพ์ อื่นๆ ด้วย สามารถนำมาสร้างเป็นรีเลชันได้ ดังรูปที่ 3.17 โดยมี ENGINE-PART-CODE เป็น คีย์ ของรีเลชัน



รูปที่ 3.17 รีเลชันที่ได้จากการแปลง รูปที่ 3.13 โดยพิจารณา เอนทิตี ไทพ์ คลาส ระบบเครื่องยนต์ ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ โดยใช้กฎข้อ 3.2



รูปที่ 3.18 แสดง ชนิดความสัมพันธ์แบบแอสโซซิเอชัน



รูปที่ 3.19 แสดง ชนิดความสัมพันธ์แบบเจนเนอรัลไลเซชัน

3.3 การแปลงจาก ไนแอม$^{++}$ เป็น โครงสร้างข้อมูลแบบเชิงวัตถุ

(Transformation from NIAM^{++} to Object-Oriented Database Schema)

1. ในกรณีของ เอนทิตี ไทพ์ และ เอนทิตี ไทพ์ คลาส

1.1 สร้าง 1 คลาส สำหรับ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่มี อินตร้า ยูนิกเนส คอนสเทรน อยู่ฝั่งเดียวกันกับ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่พิจารณาโดยนำ เลเบล ไทพ์ ของ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ฝั่งตรงข้ามมาเป็น คุณสมบัติ (Property) ของ คลาส

1.2 สำหรับ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ที่สัมพันธ์กับ แฟคท์ ไทพ์ ที่มี อินตร้า ยูนิกเนส คอนสเทรน คลุม n Role

- ถ้า เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส นั้นไม่มีความสัมพันธ์ กับ เอนทิตี ไทพ์ อื่น ไม่ต้องนำมาพิจารณา

- ถ้า เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส นั้น มีความสัมพันธ์กับเอนทิตี ไทพ์ อื่นให้สร้าง 1 คลาส สำหรับ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส นั้น แล้วพิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา แล้วนำ เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ของฝั่งตรงข้าม มาเป็น คุณสมบัติของ คลาส ด้วย

ตัวอย่าง เช่นรูปที่ 3.18 โดยพิจารณา entity STUDENT สามารถนำมาสร้างโครงสร้างฐานข้อมูลเชิงวัตถุได้ดังนี้

```
class  STUDENT
       properties
            STUDENT-ID          : integer;
            STUDENT-NAME        : string;
            DMY-BIRTH           : date;
            D-NAME              : string;
            A-NAME              : string;
            C-NAME              : string;
       OPERATIONS
            CREATE ()
            ----
end    STUDENT.
```

2. ในกรณี Entity Type หรือ Entity Type Class ที่เป็น Subtype heirarchy (Generalization)

2.1 สร้าง 1 คลาส สำหรับแต่ละ ซุปเปอร์ไทพ์ แล้วพิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

ตัวอย่าง เช่นรูปที่ 3.19 โดยพิจารณา เอนทิตี ไทพ์ เสื้อผ้าซึ่งเป็นซุปเปอร์ไทพ์ สามารถนำมาสร้าง โครงสร้างฐานข้อมูลเชิงวัตถุได้ดังนี้

```
class  CLOTHING (เสื้อผ้า)
       properties
              PRODUCT #        : integer;
              P-TYPE           : string;
              COLOR #          : string;
              COST-P           : integer;
              CLOTH-T          : string;
       OPERATIONS
              CREATE ()
              ----
end    CLOTHING.
```

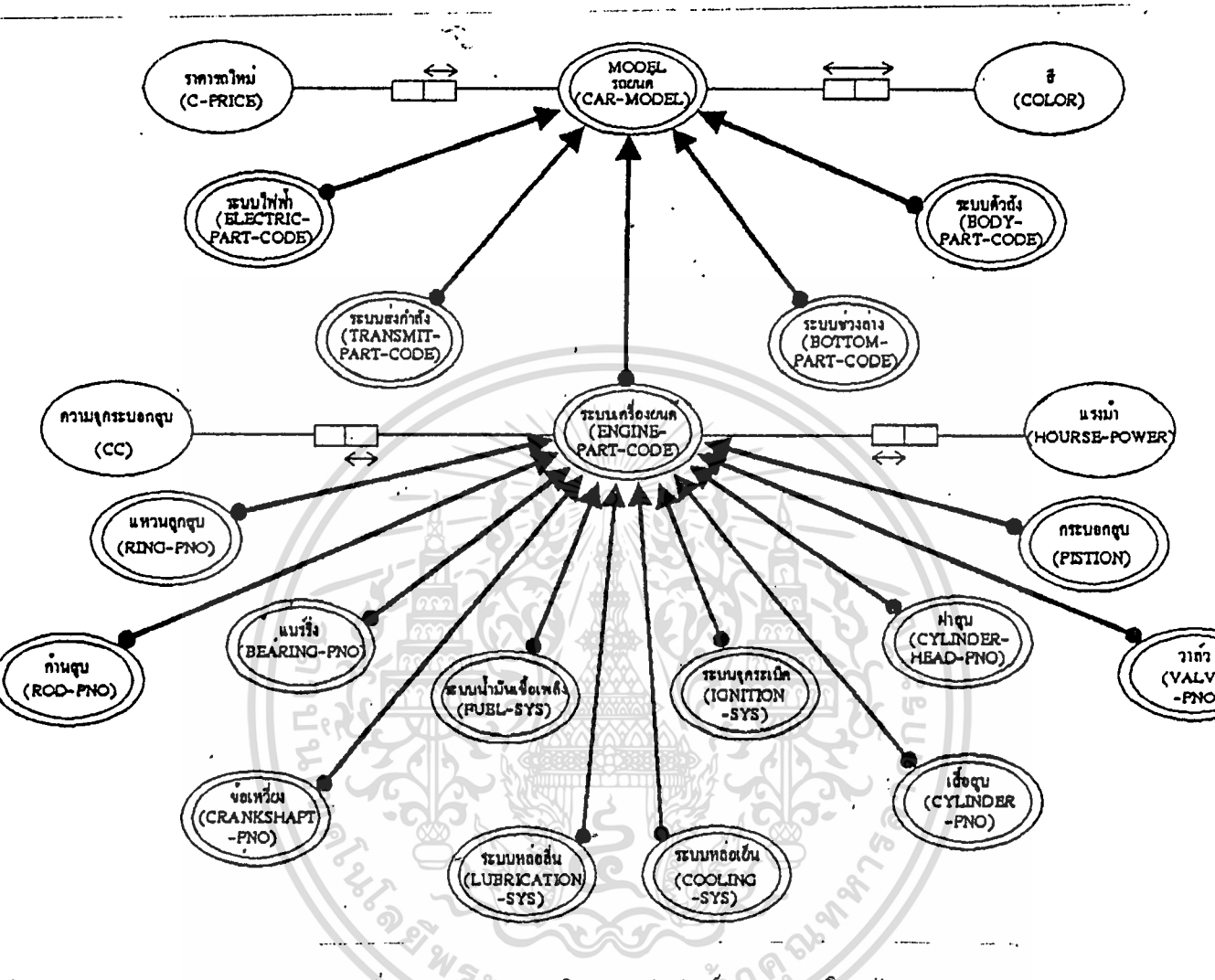
2.2 สร้าง 1 คลาส สำหรับแต่ละ ซับไทพ์ ซึ่งมีการถ่ายทอดคุณสมบัติ จากซุปเปอร์ไทพ์ ที่เป็น เบส (Base) คลาส โดยใช้คำว่า Inherit และ ตามด้วยชื่อซุปเปอร์ไทพ์ แล้ว พิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

ตัวอย่าง เช่นรูปที่ 3.19 โดยพิจารณา เอนทิตี ไทพ์ เสื้อ, กระโปรง, กางเกง ซึ่งเป็น ซับไทพ์ สามารถนำมาสร้าง โครงสร้างฐานข้อมูลเชิงวัตถุได้ดังนี้

```
class  SHIRT
       inherit CLOTHING
       properties
            COLLOR -T          : string;
            SLEEVE-T           : string;
       OPERATIONS
            CREATE ()
            ----
end    SHIRT.

class  SKIRT
       inherit CLOTHING
       properties
            SKIRT-T            : string;
            SHIR-T             : string;
       OPERATIONS
            CREATE ()
            ----
end    SKIRT.

class  TROUSER
       inherit CLOTHING
       properties
            TRUNKS-PANTS       : string;
            B-S-LEGS           : string;
       OPERATIONS
            CREATE ()
            ----
end    TROUSER.
```



รูปที่ 3.20 แสดง ขนิดความสัมพันธ์แบบแอกกริเกชัน

3. ในกรณี คอมเพลคซ์ ออบเจกต์ ที่เป็น ซุปเปอร์-พาร์ท-ออฟ และ ซับ-พาร์ท-ออฟ (Aggregation)

3.1 สร้าง 1 คลาส สำหรับแต่ละ ซุปเปอร์-พาร์ท-ออฟ แล้วพิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

3.2 ซับ-พาร์ท-ออฟ ใด ๆ ที่เป็นองค์ประกอบของ ซุปเปอร์-พาร์ท-ออฟ ให้นำ ซับ-พาร์ท-ออฟ นั้น ๆ มาเป็น คุณสมบัติของ คลาส ด้วย

ตัวอย่าง เช่นรูปที่ 3.20 โดยพิจารณา entity MODEL รถยนต์ และ ระบบเครื่องยนต์ซึ่งเป็น ซุปเปอร์-พาร์ท-ออฟ สามารถนำมาสร้างเป็นโครงสร้างฐานข้อมูลเชิงวัตถุได้ดังนี้

```
class  MODEL
       properties
            CAR-MODEL            : string;
            NO-COLOR             : string;
            C-PRICE              : integer;
            ELECTRIC-PART-CODE   : electric;
            TRANSMIT-PART-CODE   : transmit;
            EGINE-PART-CODE      : engine;
            BOTTOM-PART-CODE     : bottom;
            BODY-PART-CODE       : body;
       OPERATIONS
            CREATE ()
            ----
end    MODEL.
```

```
class  ENGINE
       properties
            ENGINE-PART-CODE    : string;
            HOURSE-POWER        : string;
            NO-PISTON           : string;
            VALVE-PNO           : string;
            CYLINDER-HEAD-PNO   : string;
            CYLINDER-PNO        : string;
            CRANKSHAFT-PNO      : string;
            BEARING-PNO         : string;
            ROD-PNO             : string;
            RING-PNO            : string;
            CC                  : integer;
            IGNITION-SYS        : ignition;
            COOLING-SYS         : cooling;
            LUBRICATION-SYS     : lubrication;
            FUEL-SYS            : fuel;
       OPERATIONS
            CREATE ()
            ----
end    ENGINE.
```

3.3 สร้าง 1 คลาส สำหรับแต่ละ ซับ-พาร์ท-ออฟ แล้วพิจารณาเหมือนกับเป็น เอนทิตี ไทพ์ หรือ เอนทิตี ไทพ์ คลาส ธรรมดา

ตัวอย่าง เช่นรูปที่ 3.20 โดยพิจารณา เอนทิตี ไทพ์ ระบบจุดระเบิด, ระบบหล่อเย็น, ระบบหล่อลื่น, ระบบนำมันเชื่อเพลิง สามารถนำมาสร้างเป็น โครงสร้างฐานข้อมูลเชิงวัตถุได้ดังนี้

```
class   IGNITION
        properties
                ----
        OPERATIONS
                CREATE ()
                ----
end     IGNITION.

class   COOLING
        properties
                ----
        OPERATIONS
                CREATE ()
                ----
end     COOLING.

class   LUBRICATION
        properties
                ----
        OPERATIONS
                CREATE ()
                ----
end     LUBRICATION.
class   FUEL
        properties
                ----
        OPERATIONS
                CREATE ()
                ----
end     FUEL.
```

บทที่ 4

สถาปัตยกรรมระบบ (System Architecture)

4.1 เมต้าคอนเซปชวลสกีมม่า (Meta Conceptual Schema)

จากบทความที่ ISO ได้เสนอ สถาปัตยกรรมฐานข้อมูล แบบ 3 ระดับ คือ

1. External Schema
2. Conceptual Schema
3. Internal Schema

คนสองคนจะสนทนากันในเรื่องใดเรื่องหนึ่งได้เข้าใจกัน จะต้องอาศัยองค์ประกอบดังนี้

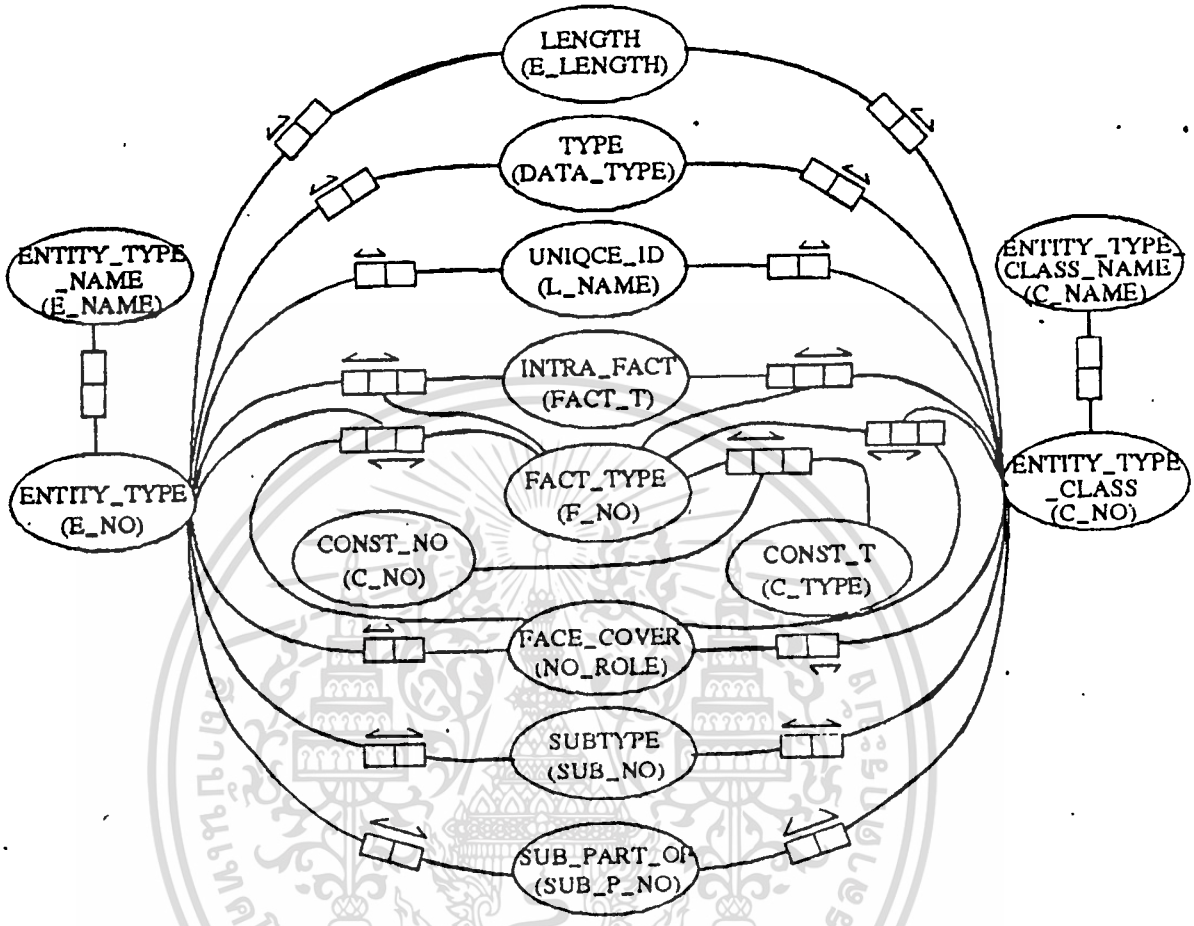
1. ต้องใช้ภาษาเดียวกัน (Common Language) ถือว่าเป็นซับเซ็ต (Subset) ของภาษาธรรมชาติเรียกว่า ภาษาธรรมชาติที่มีรูปแบบ (FNL : Formal Natural Language)

2. มี Common Knowledge ซึ่งสิ่งที่จะมานิยาม Common Knowledge ได้ก็คือ คอนเซปชวลสกีมม่า (Conceptual Schema) ซึ่งคู่สนทนาจะต้องพูดในเนื้อหาหรือ เรื่องราว (UoD : Universe of Discource) หรือ ยูโอดี เดียวกัน

ในวิทยานิพนธ์เล่มนี้เรียก คอนเซปชวลสกีมม่า ว่า เมต้าคอนเซปชวลสกีมม่า

4.1.1 การออกแบบเมต้าคอนเซปชวลสกีมม่า (Meta Conceptual Schema Designing)

จากที่กล่าวแนะนำไปแล้วว่า ไนแอมประกอบด้วยส่วนพื้นฐาน 5 ส่วนคือ เอนทิตี ไทพ์, เลเบล ไทพ์, อีลีเมนทารี แฟคท์ ไทพ์, ซับไทพ์ และ คอนสเทรน พร้อมทั้ง ซับ-พาร์ท-ออฟ และ เอนทิตี ไทพ์ คลาส ใน ไนแอม$^{++}$ ซึ่งส่วนประกอบพื้นฐานเหล่านี้จะนำมาเป็น เอนทิตี ไทพ์ ของ ไนแอม$^{++}$ ที่จะออกแบบเป็นเมต้าสคอนเซปชวลสกีมม่าสำหรับการออกแบบเมต้าคอนเซปชวลสกีมม่า จะพิจารณาเฉพาะส่วนประกอบที่สำคัญ ๆ และมีการใช้บ่อย ๆ ของไนแอม$^{++}$ เท่านั้น ซึ่งหลักเกณฑ์ในการออกแบบคอนเซปชวลสกีมม่าใช้หลักเกณฑ์ของ ไนแอม$^{++}$ นั่นเอง ทำให้เมต้าคอนเซปชวลสกีมม่าของระบบนี้เป็น ไนแอม$^{++}$ ที่อธิบายกฎไวกรณ์ของไนแอม$^{++}$ เอง

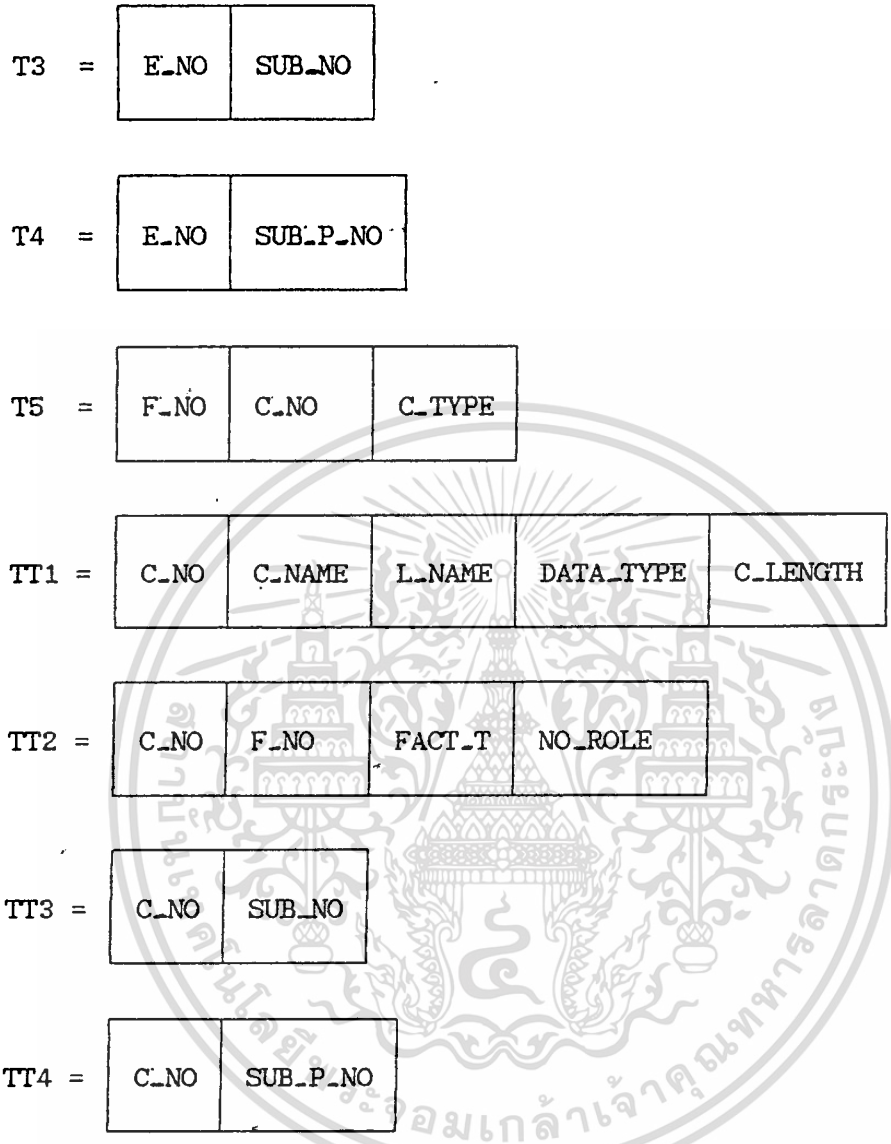


รูปที่ 4.1 แสดงเมต้าดอนเชบชวลสกีมม่า

จากรูปที่ 4.1 สามารถแปลงเป็นโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ได้ดังรูปที่ 4.2

T1 =	E_NO	E_NAME	L_NAME	DATA_TYPE	E_LENGTH

T2 =	E_NO	F_NO	FACT_T	NO_ROLE



รูปที่ 4.2 แสดงรีเลชันที่ได้จากการแปลงเมต้าดอนเซบซวลสกีมม่า

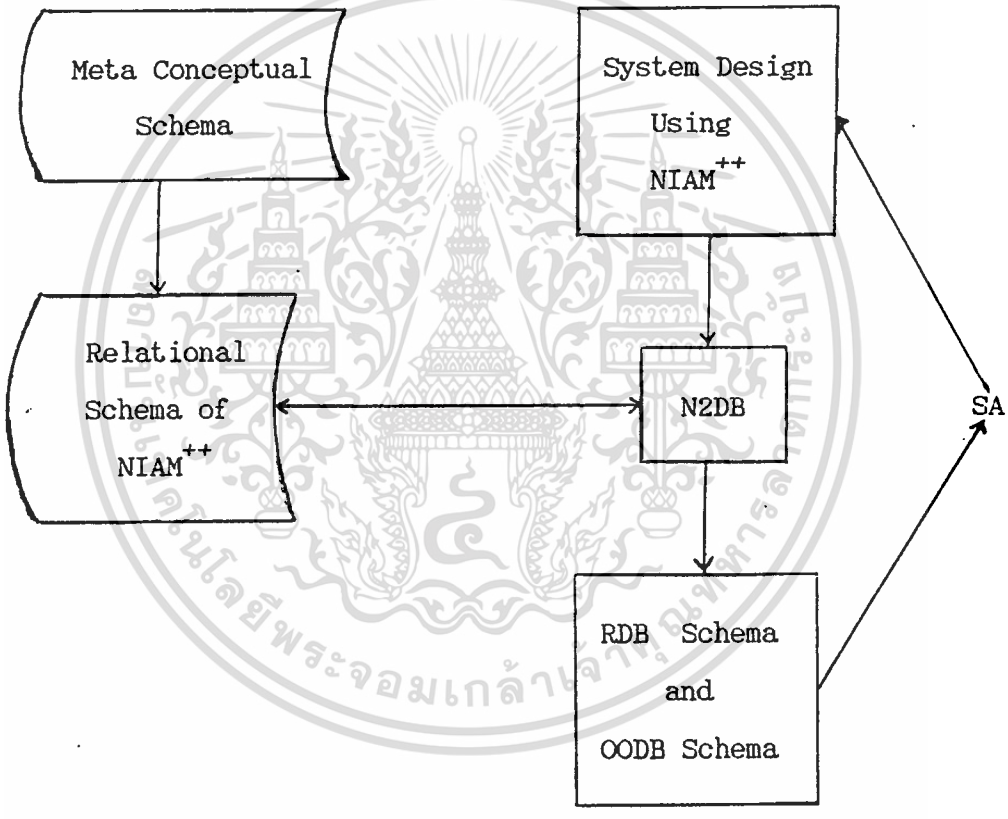
รีเลชันที่ได้จากการแปลงเมต้าดอนเซบซวลสกีมม่า ดังรูปที่ 4.2 สามารถนำมาสร้างเป็น เทเบิล (Table) เพื่อเตรียมทำ User Interface ของเครื่องมือ (Tool) ที่ใช้ในการแปลง ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์และเชิงวัตถุ

4.2 ระบบ N2DB

N2DB (NIAM^{++} TO DATABASE) เป็น User Interface สำหรับให้นักวิเคราะห์ระบบ ที่ทำการออกแบบระบบด้วย ไนแอม$^{++}$ ทำการแปลงระบบที่ออกแบบด้วย ไนแอม$^{++}$ ไปเป็นข้อมูลเก็บไว้ใน เทเบิล ต่าง ๆ ที่ได้จากการแปลงเมต้าคอนเซปชวลสกีมม่า เป็น เทเบิล ในหัวข้อ 4.1.1

4.2.1 สถาปัตยกรรมของระบบ N2DB

สถาปัตยกรรมของระบบ N2DB แสดงได้ดังรูปที่ 4.3



รูปที่ 4.3 สถาปัตยกรรมของระบบ N2DB

N2DB แบ่งการประมวลผลออกเป็น 2 ระบบ คือ Batch Processing และ Interactive Processing ซึ่งในแต่ละระบบแบ่งการทำงานออกเป็น Module ย่อย หรือ ระบบย่อย ๆ

N2DB (Interactive)

N2DB ในส่วนที่เป็น Interactive Processing จะเป็นการประมวลผลเพื่อ เก็บรักษาและบำรุงข้อมูล ที่นักวิเคราะห์ใช้ ไนแอม++ ออกแบบแล้วใช้ N2DB แปลง ไนแอม++ ไปเป็นข้อมูลเก็บไว้ใน Table ต่าง ๆ ในหัวข้อ 4.1.1

N2DB ในส่วนที่เป็น Interactive Processing แบ่งออกเป็น 9 ระบบ และในแต่ละระบบมี 4 Function คือ Add, Update, Delete และ List เพื่อใช้ทำการบำรุงรักษาข้อมูลในแต่ละระบบ

1. ระบบ ENTITY TYPE ใช้สำหรับแปลง Entity Type ทั้งหมดในระบบ ไปเป็นข้อมูลเก็บไว้ใน Table T1

2. ระบบ RELATION of ENTITY TYPE ใช้สำหรับแปลงความสัมพันธ์คือ Fact Type และ Entity Type ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table T2

3. ระบบ SUBTYPE of ENTITY TYPE ใช้สำหรับแปลง Entity Type ที่เป็น Subtype และ Supertype ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table T3

4. ระบบ SUB-PART-OF of ENTITY TYPE ใช้สำหรับแปลง Entity Type ที่เป็น Sub-part-of และ Super-part-of ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บ ไว้ใน Table T4

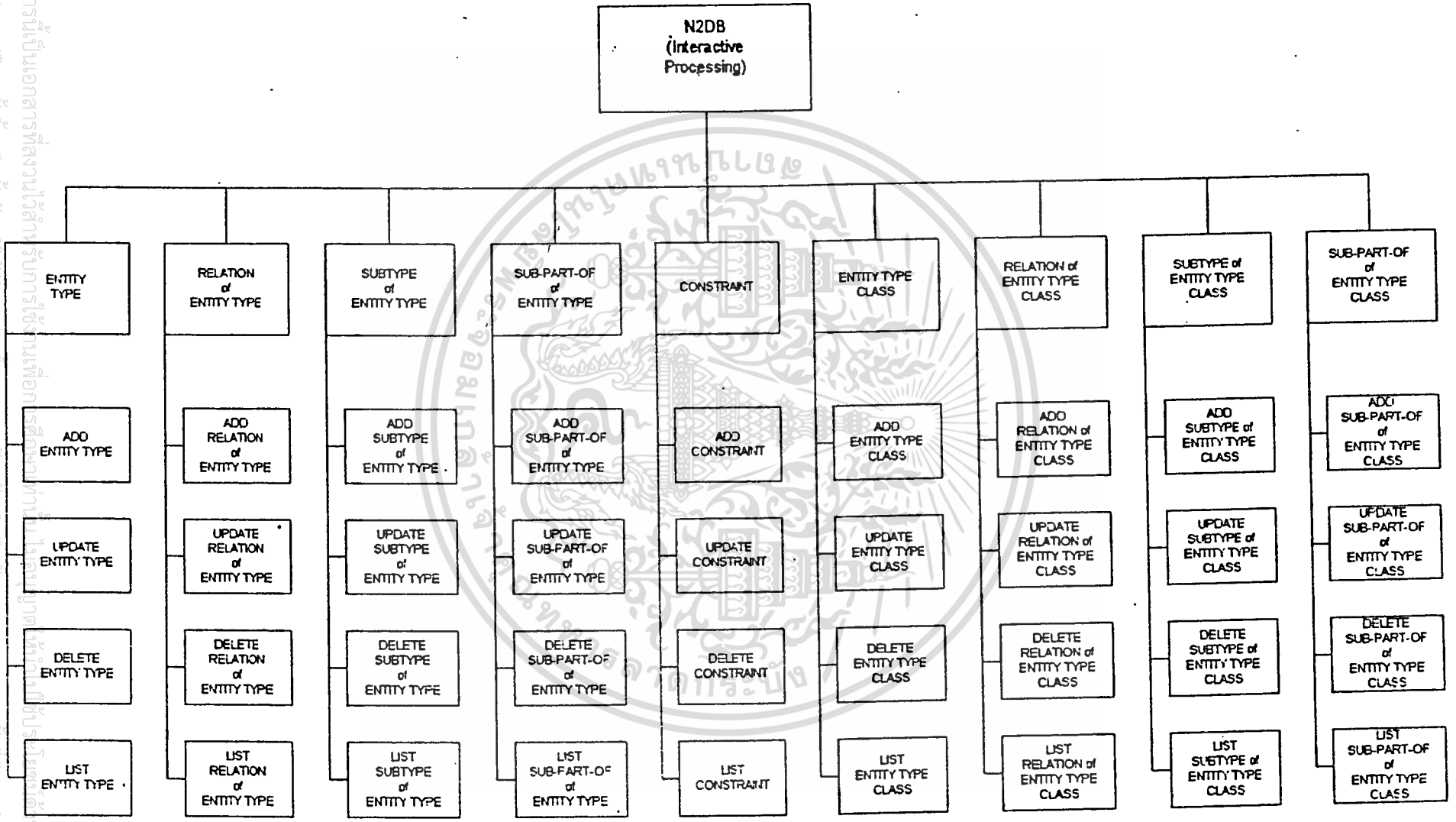
5. ระบบ CONSTRAINT ใช้สำหรับแปลง Constraint พร้อมทั้ง Fact Type และ Constraint Type ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูล เก็บไว้ใน Table T5

6. ระบบ ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type Class ทั้งหมดในระบบ ไปเป็นข้อมูลเก็บไว้ใน Table TT1

7. ระบบ RELATION of ENTITY TYPE CLASS ใช้สำหรับแปลงความสัมพันธ์คือ Fact Type และ Entity Type Class ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table TT2

8. ระบบ SUBTYPE of ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type Class ที่เป็น Subtype และ Supertype ที่เกี่ยวข้องทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table TT3

9. ระบบ SUB-PART-OF of ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type Class ที่เป็น Sub-part-of และ Super-part-of ที่เกี่ยวข้องกันทั้งหมดในระบบ ไปเป็นข้อมูลเก็บ ไว้ใน Table TT4



รูปที่ 4.4 ระบบ N2DB ที่เป็น Interactive Processing

N2DB (Batch)

N2DB ในส่วนที่เป็น Batch Processing จะเป็นการนำข้อมูลที่เก็บอยู่ใน Table ต่าง ๆ ในหัวข้อ 4.1.1 มาประมวลผลตามแอลกอริธึมในการแปลงจาก ไนแอม$^{++}$ ไปเป็นโครงสร้างข้อมูลแบบเชิงสัมพันธ์ และแบบเชิงวัตถุดังหัวข้อ 3.2 และ 3.3 ที่ได้นำเสนอไปแล้ว พร้อมทั้ง Include File สำหรับใช้ในโปรแกรมเชิงวัตถุ (ภาษา C^{++}) ตามต้องการ

N2DB ในส่วนที่เป็น Batch Processing มี 3 โปรแกรม คือ

1. โปรแกรม PRT2RDB ใช้สำหรับสร้าง RDB schema
2. โปรแกรม PRT2OODB ใช้สำหรับสร้าง OODB Schema
3. โปรแกรม PRT2OOP ใช้สำหรับสร้าง Include File เพื่อใช้ในโปรแกรมเชิงวัตถุที่เป็นภาษา C^{++}

ขั้นตอนการทำงานของ โปรแกรม PRT2RDB, PRT2OODB และ PRT2OOP

1. ทำการรวม Table ต่าง ๆ ดังนี้

- รวม Table T1 กับ TT1 ไว้ที่ Table TEMP1 เพื่อเก็บรายละเอียดของ Entity Type กับ Entity Type Class ที่อยู่ใน System Name เดียวกัน
- รวม Table T2 กับ TT2 ไว้ที่ Table TEMP2 เพื่อเก็บรายละเอียดของ Fact Type ที่สัมพันธ์กับ Entity Type และ Entity Type Class ที่อยู่ใน System Name เดียวกัน
- รวม Table T3 กับ TT3 ไว้ที่ Table TEMP3 เพื่อเก็บรายละเอียดของ Entity Type และ Entity Type Class ที่เป็น Subtype, Supertype ที่อยู่ใน System Name เดียวกัน
- รวม Table T4 กับ TT4 ไว้ที่ Table TEMP4 เพื่อเก็บรายละเอียดของ Entity Type และ Entity Type Class ที่เป็น Subt-part-of, Super-part-of ที่อยู่ใน System Name เดียวกัน

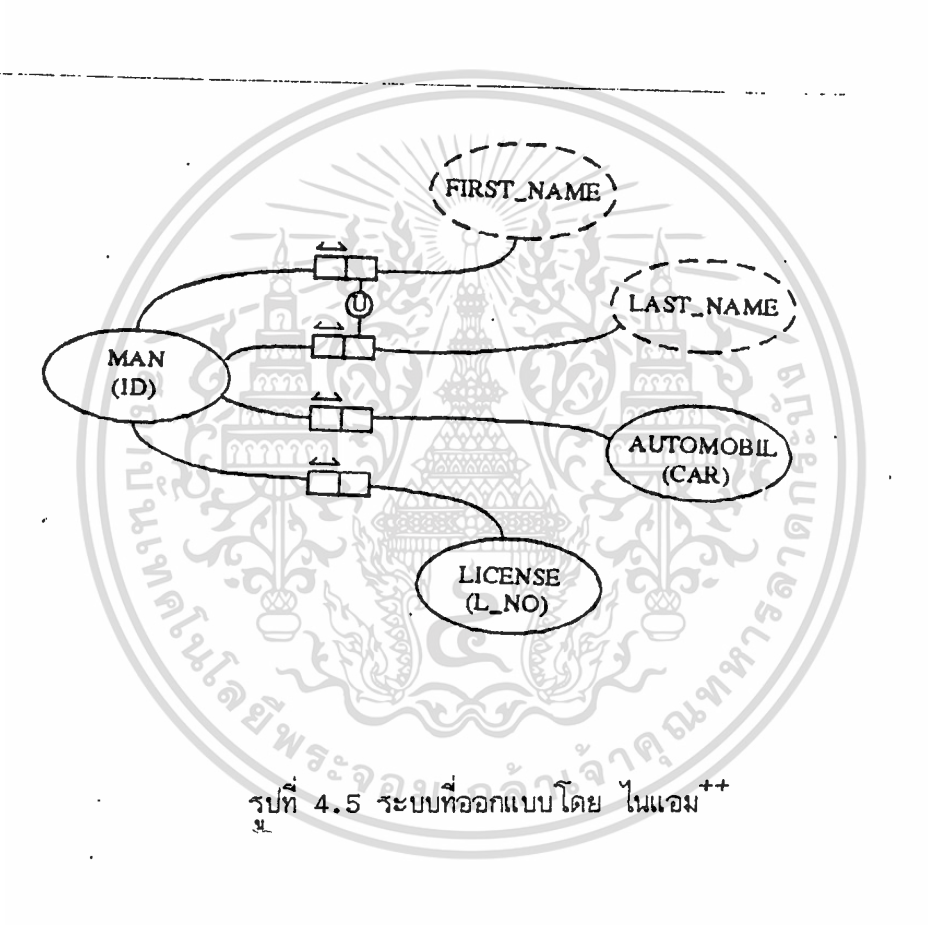
2. ทำการตรวจสอบ Sub-part-of ในระบบที่ออกแบบนี้มีโครงสร้าง ที่เป็น PART-OF หรือไม่ ถ้ามีโครงสร้าง PART-OF ก็จะทำงานตามแอลกอริธึม เพื่อสร้าง RDB schema และ OODB Schema

3. ทำการตรวจสอบ Subtype ในระบบที่ออกแบบนี้มีโครงสร้าง ที่เป็น Subtype หรือไม่ ถ้ามีโครงสร้าง Subtype ก็จะทำงานตามแอลกอริธึม เพื่อสร้าง RDB Schema และ OODB Schema

4. ทำการตรวจสอบ Entity Type ธรรมดา แล้วทำงานตามแอลกอริธึมเพื่อสร้าง RDB Schema และ OODB Schema

5. ทำการ Drop Table TEMP1, TEMP2, TEMP3 และ TEMP4 ออกจากระบบ

4.2.2 ตัวอย่างการใช้งาน ระบบ N2DB



รูปที่ 4.5 ระบบที่ออกแบบโดย ไนแอม++

จากรูปที่ 4.5 นำมาแปลงเป็นข้อมูลเก็บไว้ใน Table ต่าง ๆ โดยใช้ N2DB

```
                              N2DB
                          MAIN MENU

  1. ENTITY TYPE                    : 6. ENTITY TYPE CLASS
                                    :
  2. RELATION of ENTITY TYPE        : 7. RELATION of
                                    :    ENTITY TYPE CLASS
  3. SUBTYPE  of ENTITY TYPE        : 8. SUBTYPE of
                                    :    ENTITY TYPE CLASS
  4. SUB-PART-OF of ENTITY TYPE     : 9. SUB-PART-OF of
                                    :    ENTITY TYPE CLASS
  5. CONSTRAINT                     :
        <<< PLEASE SELECT ONE CHOICE ] >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.6 แสดง MAIN MENU ของ ระบบ N2DB

จากระบบที่ออกแบบด้วย ไนแอม[++] หัวข้อใน MAIN MENU ที่เกี่ยวข้อง คือ ENTITY TYPE, RELATION of ENTITY TYPE และ CONSTRAINT

ขั้นที่ 1 เลือกข้อ 1 จะได้รูปที่ 4.7 เพื่อแปลง Entity Type ของระบบที่ศึกษาไปเป็นข้อมูลเก็บไว้ใน Table

```
                        N2DB
                    ENTITY  TYPE

FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)
SYSTEM NAME --------
ENTITY TYPE NO. ---

          <<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.7 แสดงจอภาพสำหรับ MAINTAIN ENTITY TYPE

จากรูปที่ 4.7 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง ENTITY TYPE NO. แล้วกด ENTER จะได้จอภาพ ดังรูปที่ 4.8

```
                        N2DB
                  ADD ENTITY TYPE

SYSTEM NAME --------
ENTITY TYPE NO. ---   ENTITY TYPE NAME ----------------
                      LABEL  NAME       ----------------
                      DATA   TYPE       -
                      ENTITY LENGTH     ---

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.8 แสดงจอภาพใส่รายระเอียด ของ ENTITY TYPE

จากระบบที่ศึกษามี 3 Entity Type, 2 Label Type ใน N2DB จะรวม Entity Type และ Label Type ไว้ด้วยกัน ดังนั้นในระบบจะมี Entity Type ทั้งหมด 5 รายการ เมื่อ LIST ดู จะได้จอภาพดังรูปที่ 4.9

N2DB

LIST ENTITY TYPE

ENT-NO.	ENT-NAME	LABEL-NAME	D-TYPE	LENGHT
E01	MAN	ID	C	10
E02	FIRST_NAME	F_NAME	C	20
E03	LAST_NAME	L_NAME	C	20
E04	AUTOMOBIL	CAR	C	10
E05	LICENSE	LICENSE_NO	C	10

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.9 จอภาพแสดง ENTITY TYPE ทั้งหมดในระบบ

ขั้นที่ 2 เลือกข้อ 2 จะได้รูปที่ 4.10 เพื่อแปลง RELATION ของระบบที่ศึกษาไปเป็นข้อมูลเก็บไว้ใน Table

```
                              N2DB
                  RELATION of ENTITY TYPE

FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)
SYSTEM NAME --------
ENTITY TYPE NO. ___  FACT TYPE NO. ___

              <<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.10 แสดงจอภาพสำหรับ MAINTAIN RELATION

จากรูปที่ 4.10 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง ENTITY TYPE NO. และ FACT TYPE แล้วกด ENTER จะได้จอภาพ ดังรูปที่ 4.11

```
                              N2DB
                            RELATION

SYSTEM NAME --------
ENTITY TYPE NO. ___ FACT TYPE NO. ___
                    UNIQUENESS CONSTRAINT _ (Y,N )
                    UNIQUENESS CONSTRAINT _ (1=COVER 1 ROLE
                    COVER ROLE TYPE          2=COVER n-1 ROLE
                                             3=COVER n  ROLE)

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.11 แสดงจอภาพใส่รายระเอียด ของ RELATION

จากระบบที่ศึกษา เมื่อใส่ครบทุก Relation แล้ว LIST ดู จะได้จอภาพดังรูปที่ 4.12

N2DB

LIST RELATION

ENT-NO.	FACT]T]NO.	U]CONSTRAINT	U]CON]COVER]ROLE]T
E01	2	Y	1
E01	3	Y	1
E01	4	Y	1
E01	5	Y	1
E02	4	-	1
E03	1	-	1
E04	2	-	1
E05	3	-	1

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.12 จอภาพแสดง RELATION ทั้งหมดในระบบ

ขั้นที่ 3 เลือกข้อ 6 จะได้รูปที่ 4.12 เพื่อแปลง CONSTRAINT ของระบบที่ศึกษาไปเป็นข้อมูลเก็บไว้ใน Table

```
                              N2DB
                          CONSTRAINT

FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)
SYSTEM NAME --------
CONSTRAINT NO. ___   FACT TYPE NO. ___

              <<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.13 แสดงจอภาพสำหรับ MAINTAIN CONSTRAINT

จากรูปที่ 4.13 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง CONSTRAINT NO. และ FACT TYPE NO. แล้วกด ENTER จะได้จอภาพ ดังรูปที่ 4.14

```
                              N2DB
                          CONSTRAINT

SYSTEM NAME --------
CONSTRAINT  NO. ___   FACT TYPE NO. ___
                      CONSTRAINT TYPE _ (X:EXCLUSIVE
                                         C:CARDINALITY
                                         U:INTRA F-TYPE)

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >
```

รูปที่ 4.14 แสดงจอภาพใส่รายระเอียด ของ CONSTRAINT

จากระบบที่ศึกษา เมื่อใส่ครบทุก CONSTRAINT แล้ว LIST ดู จะได้จอภาพดังรูปที่ 4.15

N2DB

LIST CONSTRAINT

CONSTRAINT NO.	FACT TYPE NO.	CONSTRAINT TYPE
1	1	U
1	2	U
2	3	S
2	4	S
3	5	X
3	6	X

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.15 จอภาพแสดง CONSTRAINT ทั้งหมดในระบบ

ขั้นที่ 4 เมื่อแปลงรายละเอียดจาก ไนแอม$^{++}$ ไปเป็นข้อมูลใน Table ต่าง ๆ เรียบร้อยแล้วใช้โปรแกรม PRT2RDB หรือ PRT2OODB หรือ PRT2OOP เป็น PRO*COBOL คือโปรแกรมภาษา Cobol ที่ทำงานร่วมกับ SQL เพื่ออ่านข้อมูลต่างๆ จาก Table เพื่อแปลงเป็น RDB Schema หรือ OODB Schema หรือ Include File ภาษา C^{++} ตามต้องการ

จากระบบที่ศึกษาเมื่อ Run โปรแกรม PRT2RDB หรือ PRT2OODB หรือ PRT2OOP จะได้ RDB Schema หรือ OODB Schema หรือ Include File ภาษา C^{++} ดังต่อไปนี้

RDB Schema

```
CREATE TABLE TABLE01
            (ID.              CHAR (10) NOT NULL,
             F_NAME           CHAR (20),
             L_NAME           CHAR (20),
             CAR              CHAR (10),
             LICENSE_NO       CHAR (10));
```

OODB Schema

```
CLASS  MAN
       PROPERTIES
            ID          : STRING;
            F_NAME      : STRING;
            L_NAME      : STRING;
            CAR         : STRING;
            LICENSE_NO  : STRING;
       OPERATIONS
            CREATE ()
            ---
END    MAN
```

```
#INCLUDE <STRING.H>
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <CONIO.H>
#INCLUDE "HEADER.H"
 STRUCT DATA01
 {
        CHAR ID                 (10 );
        CHAR F_NAME             (20 );
        CHAR L_NAME             (20 );
        CHAR CAR                (10 );
        CHAR L_CENSE            (10 );
 };
 INT EQUAL_ID(CHAR *N1, CHAR *N2)
 {
        DATA01  *DATA1, *DATA2;
        DATA1 = (DATA*)N1;
        DATA2 = (DATA*)N2;
        RETURN(STRCMP(DATA1->ID,DATA2->ID) == 0);
 };
 INT LESS_THAN_ID(CHAR *N1, CHAR *N2)
 {
        DATA01  *DATA1, *DATA2;
        DATA1 = (DATA*)N1;
        DATA2 = (DATA*)N2;
        RETURN(STRCMP(DATA1->ID,DATA2->ID) < 0);
         };

         INT EQUAL_F_NAME(CHAR *N1, CHAR *N2)
         {
```

```
            DATA01  *DATA1, *DATA2;
            DATA1 = (DATA*)N1;
            DATA2 = (DATA*)N2;
            IF(STRCMP(DATA1->F_NAME,DATA2->L_NAME) == 0);
               RETURN(STRCMP(DATA1->L_NAME,DATA2->L_NAME) == 0);
               RETURN(0);
        };
        INT CMP_NO(CHAR *N1, CHAR *N2)
        {
               DATA01  *DATA1, *DATA2;
               DATA1 = (DATA*)N1;
               DATA2 = (DATA*)N2;
               IF(STRCMP(DATA1->NO,"1")
                     RETURN(STRCMP(DATA2->NO,"2") == 0);
               IF(STRCMP(DATA2->NO,"2")
                     RETURN(STRCMP(DATA2->NO,"1") == 0);
        };
        VOID SHOW1(CHAR *N)
{
     -----
     -----
     -----
}
MAIN()
{
     -----
     -----
     -----
}
```

```
//HEADER FILE FOR PGM1.CPP
//FILE HEADER.H
TYPEDEF INT(*LESSTHAN)(CHAR *,CHAR *);
TYPEDEF INT(*EQUAL)(CHAR *,CHAR *);
TYPEDEF VOID(*VISITTYPE)(CHAR *);
TYPEDEF INT(*COMPARETYPE)(CHAR *,CHAR *);
CLASS NODE
{
FRIEND CLASS SEARCH_TABLE;
PRIVATE:
        NODE *LEFT, *RIGHT;
        CHAR *CONTENTS;
};
CLASS SEARCH_TABLE
{
PRIVATE:
        NODE        *ROOT;
        INT          SIZE;
        LESSTHAN     LT;
        EQUAL        EQ1;
        EQUAL        EQ2;
        COMPARETYPE  CMP;
        VISITTYPE    VISIT;
PUBLIC:
        SEARCH_TABLE(INT S,LESSTHAN L,EQUAL E1,
                     EQUAL E2,
                     COMPARETYPE C,VISITTYPE V)
        {
                ROOT  = 0;
                SIZE  = S;
```

```
        LT    = L;
        EQ1   = E1;
        EQ2   = E2;
        CMP   = C;
        VISIT = V;
}
SEARCH_TABLE(INT S,LESSTHAN L,EQUAL E1,
             EQUAL E2,VISITTYPE V)
{
        ROOT  = 0;
        SIZE  = S;
        LT    = L;
        EQ1   = E1;
        EQ2   = E2;
        VISIT = V;
}
INT INSERT1(CHAR *A);
INT INSERT2(CHAR *A);
INT REMOVE(CHAR *A);
INT IS_PRESENT(CHAR *A);
  {
      CHAR* SEARCH_SHOW(CHAR *A);
      VOID PROCESS_NODE(NODE *N=0,INT FIRST =1);
      VOID CLEAR(NODE *N=0,INT FIRST =1)
      ~SEARCH_TABLE() {CLEAR();}
  };
```

```
VOID SEARCH_TABLE::PROCESS_NODE(NODE *N,INT FIRST)
        {
            NODE *CURRENT;
            iF (FIRST) {
                    CURRENT = ROOT;
                    FIRST   = 0;
            }
            ELSE
                    CURRENT = N;
            IF (CURRENT !=  0) {
                    PROCESS_NODE(CURRENT->LEFT,FIRST);
                    // DISPLAY THE CONTENTS
                    (*VISIT)(CURRENT->CONTENTS);
                    PROCESS NODE(CURRENT  RIGHT,FIRST);
            }
        }

//INSERT INTO BINARY TREE
INT SEARCH_TABLE::INSERT1(CHAR *A)
  {
     NODE *PARENT,*CURRENT;
     INT   FOUND = 0;
     PARENT = 0;
     CURRENT = ROOT;
     WHILE (CURRENT && !FOUND)
       {

            // CHECK UNIQUENESS CONSTRAINT (INTRA-FACT TYPE)
            IF ((*EQ1)(CURRENT->CONTENTS,A))
                    RETURN(1);
```

```
// CHECK INTER-FACT TYPE
IF ((*EQ2)(CURRENT->CONTENTS,A))
        RETURN(2);
// CHECK EXCLUSION CONSTRAINT
IF ((*EQ2)(CURRENT->CONTENTS,A))
        IF ((*CMP)(CURRENT->CONTENTS,A))
                RETURN(3);
PARENT = CURRENT;
IF ((*LT)(A,CURRENT->CONTENTS))
   CURRENT = CURRENT->LEFT;
ELSE
   CURRENT = CURRENT->RIGHT;
}
IF (!PARENT)
   {
    // FIRST NODE IN SEARCHITABLE
    ROOT = NEW NODE;
    ROOT->LEFT = ROOT->RIGHT = 0;
    // ALLOCATE SPACE FOR CONTENTS
    ROOT->CONTENTS = NEW CHAR(SIZE);
    // DO A BYTE-BY-BYTE TRANFER OF DATA
    FOR (INT I = 0; I<SIZE; I++)
        ROOT->CONTENTS (I) = A (I);
   }
ELSE
   {
    IF ((*LT)(A,PARENT->CONTENTS))
       {
        // ADD NEW NODE TO THE LEFT OF PARENT
        NODE *NEW_NODE = NEW NODE;
```

```
                    // ALLOCATE SPACE FOR THE CONTENTS
                    NEW_NODE->CONTENTS = NEW CHAR(SIZE);
                    // DO A BYTE-BY-BYTE TRANFER OF DATA
                    FOR (INT I = 0; I<SIZE; I++)
                    NEW_NODE->CONTENTS (I) = A (I);
                    NEW_NODE->LEFT = NEW_NODE->RIGHT = 0;
                    PARENT->LEFT = NEW_NODE;
                }
            ELSE
                {
                // ADD NEW NODE TO THE RIGHT OF PARENT
                NODE *NEW_NODE = NEW NODE;
                // ALLOCATE SPACE FOR THE CONTENTS
                NEW_NODE->CONTENTS = NEW CHAR(SIZE);
                // DO A BYTE-BY-BYTE TRANFER OF DATA
                FOR (INT I = 0; I<SIZE; I++)
                NEW_NODE->CONTENTS (I) = A (I);
                NEW_NODE->LEFT = NEW_NODE->RIGHT = 0;
                PARENT->RIGHT = NEW_NODE;
                }
            }
    RETURN(0);
}

INT SEARCH_TABLE::IS_PARENT(CHAR *A)
{
    NODE *CURRENT;
    INT FOUND = 0;
    CURRENT = ROOT;
    WHILE (CURRENT && !FOUND)
```

```
        {
            //THE NODE IS PARENT
            IF((*EQ1)(CURRENT->CONTENTS,A))
                    FOUND = 1;
            ELSE
               {
                    IF ((*LT)(A,CURRENT->CONTENTS))
                            CURRENT = CURRENT->LEFT;
                    ELSE
                            CURRENT = CURRENT->RIGHT;
               }
        }
      RETURN FOUND;
  }

//SEARCH THE NODE AND DISPLAY CONTENTS OF DATA
CHAR* SEARCH_TABLE::SEARCH_SHOW(CHAR *A)
  {
        NODE *CURRENT;
        INT FOUND = 0;
        CURRENT = ROOT;
        WHILE (CURRENT && !FOUND)
           {
              IF((*EQ1)(CURRENT->CONTENTS,A))
                      FOUND = 1;
              ELSE
                 {
                    IF ((*LT)(A,CURRENT->CONTENTS))
                            CURRENT = CURRENT->LEFT;
                    ELSE
```

```
                                    CURRENT = CURRENT->RIGHT;
                        }
                }

        IF (FOUND)
            {
                //display contents of node
                (*visit)(current->contents);
                RETURN(CURRENT->CONTENTS);
            }
            ELSE
                //NOT FOUND
                RETURN (0);
    }

//REMOVE EVERY NODE FROM BINARY TREE
VOID SEARCH_TABLE::CLEAR(NODE *N,INT FIRST)
    {
        NODE *CURRENT;
        IF(FIRST)
            {
                CURRENT = ROOT;
                FIRST = 0;
                ROOT = 0;
            }
        ELSE
                CURRENT = N;
        IF(CURRENT != 0)
            {
                CLEAR(CURRENT->LEFT,FIRST);
```

```
                CLEAR(CURRENT->RIGHT,FIRST);
                DELETE CURRENT->CONTENTS;
                DELETE CURRENT;
            }
    }

//REMOVE THE NODE FROM BINARY TREE
INT SEARCH_TABLE::REMOVE(CHAR *A)
    {
        NODE *PREVIOUS,
             *PRESENT,
             *REPLACE,
             *S,
             *PARENT;
        INT FOUND = 0;
        PREVIOUS = 0;
        PRESENT = ROOT;

        WHILE (PRESENT && !FOUND)
            {
              IF ((*EQ1)(A,PARENT->CONTENTS))
                   FOUND = 1;
              ELSE
                  {
                     PREVIOUS = PRESENT;
                     IF((*LT)(A,PRESENT->CONTENTS))
                         PARENT = PARENT->LEFT;
                     ELSE
                         PARENT = PARENT->RIGHT;
                  }
```

```
    }
IF(FOUND)
    {
        IF(PRESENT->LEFT == 0)
                REPLACE = PRESENT->RIGHT;
        ELSE
            IF (PRESENT->RIGHT == 0)
                REPLACE = PRESENT->LEFT;
            ELSE
                {
                PARENT = PRESENT;
                REPLACE = PRESENT->RIGHT;
                S = REPLACE->LEFT;
                WHILE (S != 0)
                    {
                    PARENT = REPLACE;
                    REPLACE = S;
                    S = REPLACE->LEFT;
                    }
                IF (PARENT != PRESENT)
                    {
                    PARENT->LEFT = REPLACE->RIGHT;
                    REPLACE->RIGHT = PRESENT->RIGHT;
                    }
                REPLACE->LEFT = PRESENT->LEFT;
                }
        IF (PREVIOUS == 0)
            ROOT = REPLACE;
        ELSE
            IF (PRESENT == PREVIOUS->LEFT)
```

```
                    PREVIOUS->LEFT = REPLACE;
                ELSE
                    PREVIOUS->RIGHT = REPLACE;
                    DELETE PRESENT->CONTENTS;
                    DELETE PRESENT;
                    RETURN (0);
            }
    // NOT REMOVE
    IF (!FOUND)
        RETURN (1);
}

//CHECK CADINALITY CONSTRAINT
VOID SEARCH_TABLE::CHECK(NODE *N,INT FIRST)
  {
    NODE *CURRENT;
    IF(FIRST)
       {
          CURRENT = ROOT;
          FIRST = 0;
       }
    ELSE
    CURRENT = N;
    IF(CURRENT != 0)
       {
          CHECK(CURRENT->LEFT,FIRST);
          (*CHK)(CURRENT->CONTENTS);
          CHECK(CURRENT->RIGHT,FIRST);
       }
  }
```

บทที่ 5

การจัดทำระบบ (System Implementation)

ในการสร้างเครื่องมือช่วยออกแบบฐานข้อมูลแบบเชิงสัมพันธ์และแบบเชิงวัตถุนั้นมีขั้นตอนต่าง ๆ ดังนี้

1. ใช้ ไนแอม$^{++}$ ซึ่งเป็นแบบจำลองข้อมูลระดับแนวความคิด ที่พัฒนามาจากไนแอมเพื่อช่วยในการออกแบบระบบ

2. ใช้ N2DB เป็น User Interface ซึ่งการเขียน User Interface นี้ใช้ Oracle RDBMS เป็นตัวจัดการฐานข้อมูล และใช้ SQL*Forms ซึ่งเป็น Product ของ Oracle เป็นตัวช่วยในการจัดการเกี่ยวกับข้อมูลในฐานข้อมูลโดยผ่านทางจอภาพ

3. ใช้ PRT2RDB และ PRT2OODB เป็นโปรแกรมสำหรับอ่านข้อมูลจากฐานข้อมูลแล้วทำงานตามแอลกอริธึมในการแปลง ไนแอม$^{++}$ เป็น RDB Schema และ OODB Schema เพื่อสร้าง RDB Schema และ OODB Schema พร้อมทั้ง Include File สำหรับเขียนโปรแกรมเชิงวัตถุที่เป็นภาษา C^{++} ซึ่ง PRT2RDB และ PRT2OODB นี้ใช้ PRO* COBOL ในการติดต่อกับฐานข้อมูล

การพัฒนาเครื่องมือช่วยออกแบบฐานข้อมูลแบบเชิงสัมพันธ์และแบบเชิงวัตถุนี้ทำการพัฒนาโดยใช้ เครื่องคอมพิวเตอร์ขนาด Mainframe ของ IBM 9121-311 ใช้ MVS เป็น Operating System

ดังนั้น Application Software ที่ใช้ในการพัฒนาระบบ N2DB คือ

1. Oracle RDBMS
2. SQL
3. SQL* Forms
4. Pro* COBOL

5.1 Oracle RDBMS

Oracle RDBMS เป็นโปรแกรมจัดการระบบฐานข้อมูล ที่น่าสนใจโปรแกรมหนึ่ง สำหรับระบบ Mainframe, Minicomputer และ Microcomputer โดยที่ Oracle RDBMS ใช้ SQL ที่เป็น Standard Language โดยที่ ANSI (The American Standards Institute) และ ISO (International Standard Organization) จัดให้เป็นมาตรฐานสำหรับฐานข้อมูลเชิงสัมพันธ์

Oracle RDBMS ใช้ SQL ตาม ANSI ที่จัดตั้งไว้เป็นมาตรฐาน และได้เพิ่ม SQL*Plus เข้าไปทำงานร่วมกับ SQL ด้วย

ลักษณะเฉพาะที่สำคัญของ Oracle (Version 6)

1. การค้นหาข้อมูลใน Oracle Database กระทำผ่านทาง SQL (Structure Query Language) โดยที่ SQL เป็น Nonprocedural Language และเป็น Set-oriented และสามารถใช้ 3GL (Third Generation Language) ทำการ Embeded กับ SQL Statement ได้

2. Oracle มี Optimiser ซึ่งเป็น Rule-based ทำให้การทำงานของ Queries ดีขึ้น

3. Oracle มี การค้นหาข้อมูลแบบ Index ซึ่ง Index ของ Oracle ใช้ B-tree

4. Oracle มี Locking Mechanism ทำการ Lock ข้อมูลตั้งแต่ระดับ Table และ Tupple

5. Oracle มี Application Tools เพื่ออำนวยความสะดวกในการพัฒนางาน เช่น SQL*Forms, SQL*Reportwriter และ SQL*Menu

6. Oracle มี Mechanism เป็น Area สำหรับใช้ติดต่อระหว่าง User Process กับ Oracle ซึ่ง Cursor Mechanism ทำหน้าที่ Link ระหว่างการ Process แบบ Set-at-a-time ของ Oracle กับการ Process แบบ Row-at-a-time ของ 3GL

5.2 SQL (Structured Query Language)

SQL เป็น Nonprocedural Language ประโยคคำสั่งเหมือนภาษาอังกฤษง่ายต่อการเรียนรู้ และ SQL เป็นทั้ง Interactive Query Language และ Database Programming Language ซึ่งเป็น Embeded SQL กับ Host Language เช่น ภาษา Cobol ฯลฯ

ตัวอย่าง คำสั่ง SQL ที่เป็น Interactive Query

```
SELECT CITY FROM S WHERE  S-NO = 'S4';
```

ตัวอย่าง คำสั่ง SQL ที่เป็น Embeded ใน Cobol

```
EXEC SQL SELECT CITY FROM S WHERE  S-NO = 'S4' END-EXEC.
```

Types of SQL Statement

SQL Statement สามารถแบ่งออกเป็น 3 กลุ่มได้ดังนี้

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)

DDL

สามารถใช้ SQL Statement ในส่วนที่เป็น DDL ทำการ CREATE, ALTER และ DROP

CREATE เป็นการสร้าง Table ใหม่

ALTER เป็นการเพิ่มเติมแก้ไข Table ที่มีอยู่แล้ว

DROP เป็นการลบ Table ที่มีอยู่แล้วออกจากระบบ

คำสั่ง CREATE และ DROP เป็นคำสั่งที่ใช้บ่อยมากใน DDL ตัวอย่างเช่น

```
CREATE TABLE
CREATE VIEW
CREATE INDEX
DROP   TABLE
DROP   VIEW
DROP   INDEX
```

DML

เมื่อสร้าง Database แล้ว SQL Statement ในส่วนที่เป็น DML จะมีการใช้มากกว่าคำสั่งของ DDL และ DCL ใช้ DML เพื่อเข้าไปใช้ข้อมูลภายใน Database

SELECT เป็นการสอบถาม (Query) ข้อมูลจาก Database

UPDATE เป็นการแก้ไขข้อมูลภายใน Row ที่มีอยู่แล้วของ Database

INSERT เป็นการเพิ่ม Row ใหม่เข้าไปใน Database

DELETE เป็นการลบ Row เดียวหรือหลาย ๆ Row ออกจาก Database

ตัวอย่างคำสั่ง SELECT และ UPDATE เช่น

```
SELECT S-NO FROM SP WHERE  P-NO = 'P2';
UPDATE S SET STATUS = 2 * STATUS WHERE  CITY = 'LONDON';
```

DCL

GRANT, REVOKE, COMMIT และ ROLLBACK เป็นคำสั่งในส่วนของ DCL ซึ่งใช้ในการควบคุม การเข้าถึงข้อมูลใน Database และยืนยัน หรือยกเลิก Database Transactions เป็นการรักษาความปลอดภัย Database

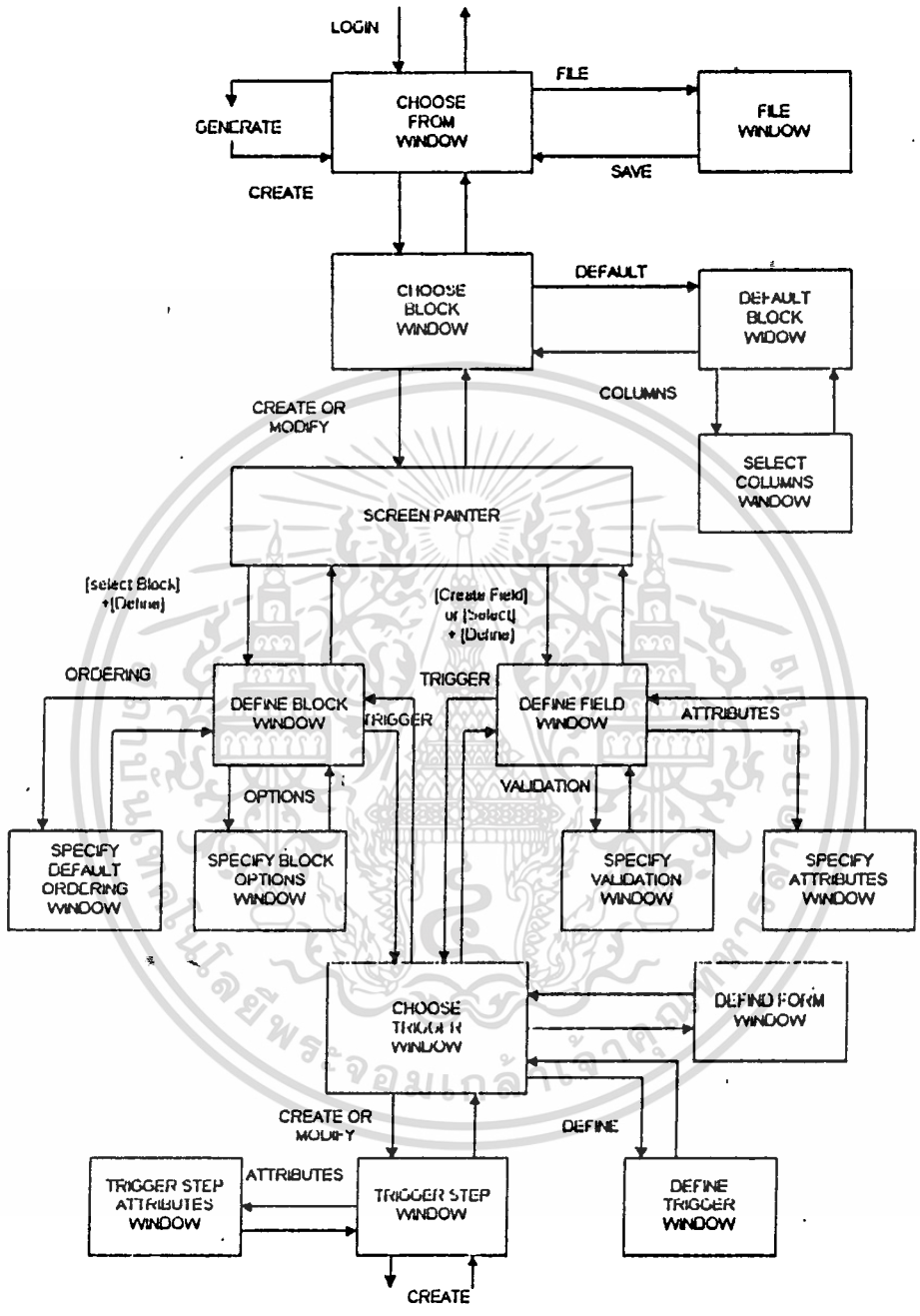
DBA (Database Administrator) จะใช้คำสั่งในส่วนของ DCL เพื่อ

1. ควบคุมว่า User ไหนสามารถจะเข้าไปทำอะไร กับ Table อะไรได้บ้าง
2. ควบคุมว่า User ไหนสามารถ Logon เข้าสู่ระบบได้บ้าง
3. ควบคุม Privileges ของ User แต่ละคน

5.3 SQL* Forms

SQL*Forms เป็น Product ของ Oracle ซึ่งถือว่าเป็น 4GL (4 Generation Language) เป็น NonProcedural Language

ใช้ SQL*Forms เป็นตัวจัดการเกี่ยวกับ Information ในฐานข้อมูลโดยผ่านทางจอภาพ การทำงานของ SQL*Forms ใช้แนวความคิดที่ว่า "What You want and not on how to get it" ซึ่งการทำงานในระบบ SQL*Forms แสดงได้ดังรูปที่ 5.1

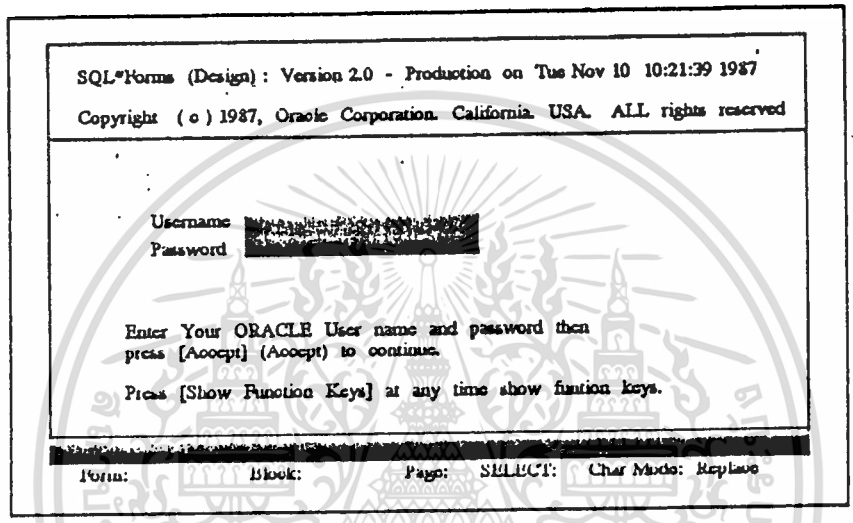


รูปที่ 5.1 การทำงานในระบบ SQL*Forms

จากรูปที่ 5.1 สามารถสรุปขั้นตอนการทำงานของการสร้าง Form ใน SQL*Forms ได้ดังนี้

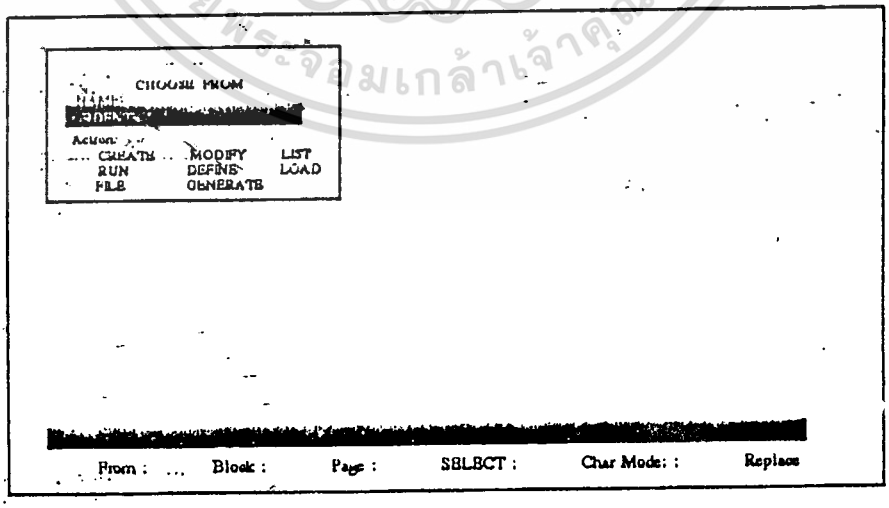
ขั้นตอนการ Design Form

1. Login เข้าสู่ระบบของ SQL*Forms โดยระบุ Userid และ Password ตามที่ DBA กำหนดให้ผ่านทางจอภาพ ดังรูปที่ 5.2



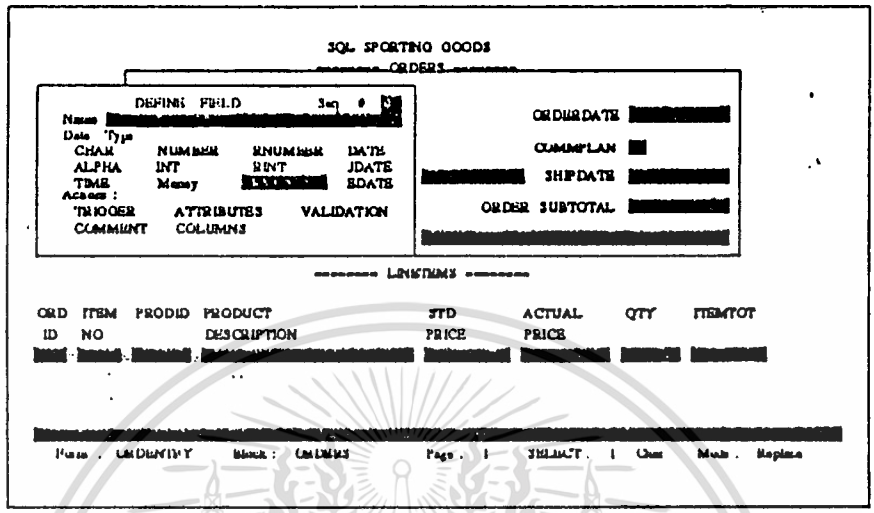
รูปที่ 5.2 แสดง Login Window

2. เลือก Form ที่สร้างไว้แล้ว หรือ จะสร้าง Form ใหม่ก็ได้ ผ่านทาง Form Window ดังรูปที่ 5.3



รูปที่ 5.3 แสดง Form Window

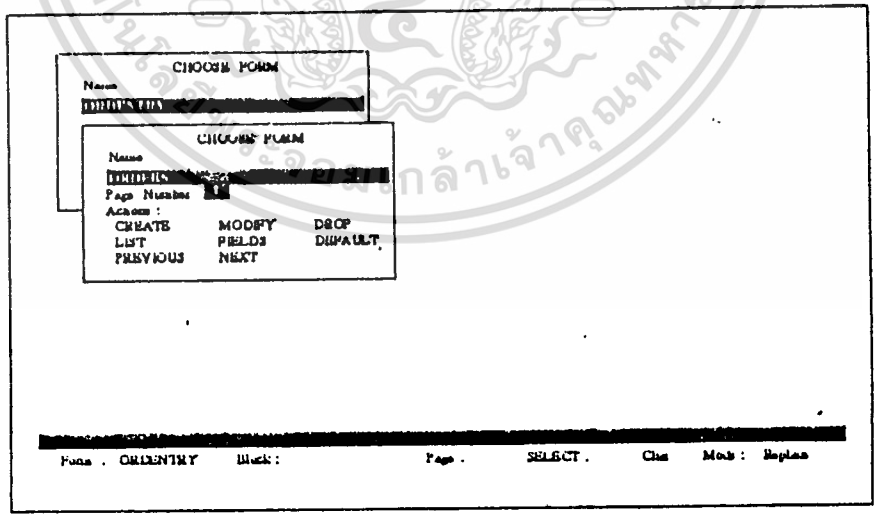
3. เลือก Block ที่สร้างไว้แล้ว หรือ จะสร้าง Block ใหม่ก็ได้ ผ่านทาง Block Window ดังรูปที่ 5.4



รูปที่ 5.4 แสดง Block Window

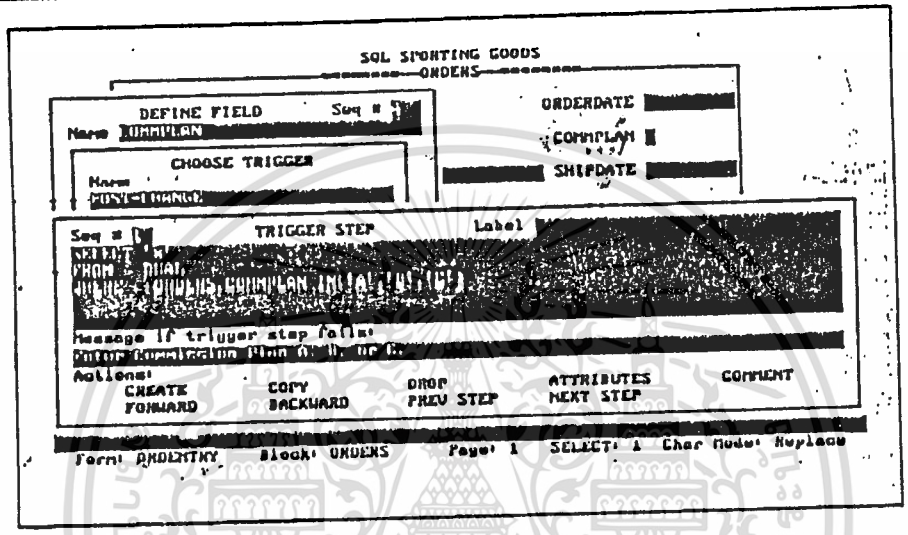
4. วาดจอภาพตามต้องการ

5. ทำการ Define Field และ กำหนด Attributes ต่าง ๆ ผ่านทาง Define Field Window ดังรูปที่ 5.5



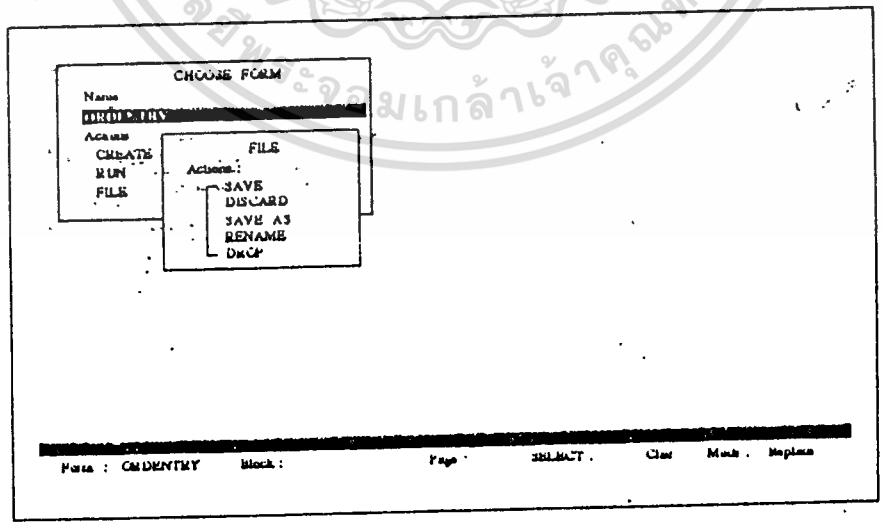
รูปที่ 5.5 แสดง Field Window

6. ทำการ Define Triggers เพื่อควบคุมการทำงานของ Form Triggers มี 3 ระดับคือ Form Level, Block Level และ Field Level เพื่อเป็นการควบคุมการทำงาน หรือเหตุการณ์ก่อนที่จะเข้าหรือ ออกจาก Form, Block หรือ Field ตามลำดับ การ Define Trigger จะกระทำผ่านทาง Triggers Window ดังรูปที่ 5.6



รูปที่ 5.6 แสดง Triggers Window

7. ทำการ Save และ Generate Form ที่เราสร้างผ่านทาง File Window ดังรูปที่ 5.7



รูปที่ 5.7 แสดง File Window

5.4 Pro*COBOL

Programmer สามารถที่จะใช้ภาษาโปรแกรมมิ่งที่คุ้นเคยได้โดยการใส่ SQL Command เข้าไปใน Source Code ของโปรแกรมภาษาที่คุ้นเคยได้ เช่น Fortran, Cobol ฯลฯ ซึ่ง Oracle ได้เสนอ Pro Series สำหรับทำการ Embeded SQL Command คือ Pro*Fortran, Pro*C และ Pro*Cobol ในวิทยานิพนธ์ฉบับนี้ใช้ Pro*Cobol

Pro* Cobol เป็น Embeded SQL ซึ่งเป็นภาษาโคบอลที่ทำงานร่วมกับ SQL และมีตัวแปลภาษา (Compiler) คือ Pro* Cobol Precompiler เป็นตัวแปล SQL ให้เป็นภาษา Cobol เสียก่อน แล้วจึง Compile และ Run ภายใต้ Compiler ภาษา Cobol

Embeded SQL Syntax

1. SQL Command ในโปรแกรมภาษา Cobol ต้อง
   - ขึ้นต้นด้วย EXEC SQL
   - ลงท้ายด้วย END-EXEC

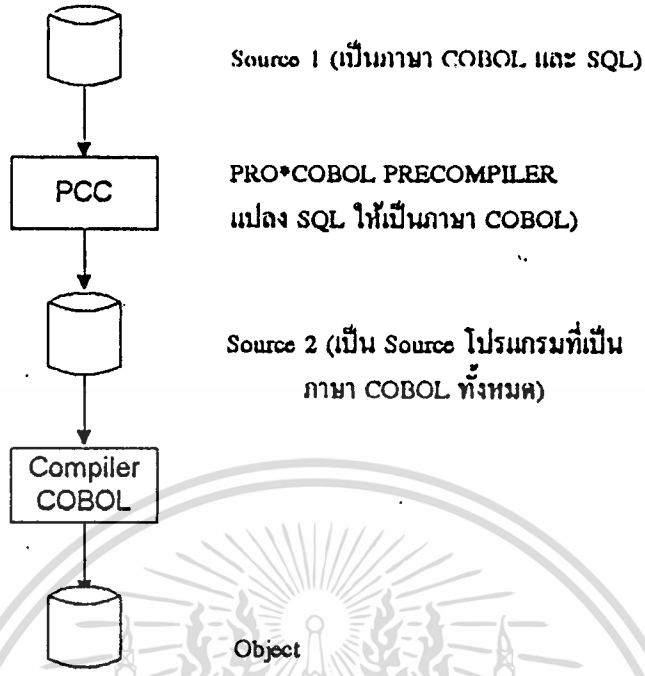
เช่น EXEC SQL DELETE FROM EMP WHERE COMM = ZERO END-EXEC.

2. ตัวแปรที่จะใน SQL Statement ต้องกำหนดใน WORKING-STORAGE SECTION และจะต้องกำหนดส่วนที่ใช้ติดต่อกับ Oracle Communication Area (ORACA) โดยมีรูปแบบดังนี้

```
WORKING-STORAGE SECTION.
---
---
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 EMP-REC.
    05 EMP-NO    PIC X(3).
    05 EMP-NAME  PIC X(40).
EXEC SQL END DECLARE SECTION END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

3. ใน PROCEDURE DIVISION ทุกโปรแกรมต้องมีการ Connect เข้าสู่ระบบของ Oracle โดยใช้คำสั่ง

EXEC SQL CONNECT :USERNAME IDENTIFIED BY :PASSWORD END-EXEC.



รูปที่ 5.8 การทำงานของ Precompiler ภาษาโคบอล

ตัวอย่างโปรแกรม PRO* COBOL

```
ID  DIVISION.
PROGRAM-ID. EXP1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*-----------------------------------------------*
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 USERNAME              PIC X(8).
01 PASSW                 PIC X(8).
01 EMP-REC.
   05 EMP-ID             PIC X(3).
   05 EMP-NAME           PIC X(40).
EXEC SQL END DECLARE SECTION END-EXEC.
*-----------------------------------------------*
```

```
EXEC SQL INCLUDE SQLCA END-EXEC.
01 DISPLAY-REC.
   05 D-ID                PIC X(3).
   05 D-NAME              PIC X(40).
PROCEDURE DIVISION.
BEGIN-PGM.
    EXEC SQL WHENEVER SQLERROR GOTO SQL-ERR END-EXEC.
    PERFORM LOGON.
    EXEC SQL WHENEVER NOT FOUND GOTO NO-EMP END-EXEC.
    PERFORM DO-SELECT.
    PERFORM DISPLAY-RECORD.
    EXEC SQL COMMIT RELEASE END-EXEC.
    STOP RUN.
LOGON.
    MOVE 'RACHANEE' TO USERNAME.
    MOVE 'RACHANEE' TO PASSW.
    EXEC SQL CONNECT :USERNAME IDENTFIED BY :PASSW
        END-EXEC.
    DISPLAY '*** CONNECT TO ORALE ALREADY ***'
DO-SELECT.
    EXEC SQL SELECT EMP]ID, EMP]NAME INTO
                    :EMP-ID, :EMP-NAME
             FROM   EMP
             WHERE  EMP-ID = 'E10'
    END-EXEC.
NO-EMP.
    DISPLAY 'NO EMO-ID'.
    GO TO END-PGM.
```

```
DISPLAY-RECORD.
    DISPLAY 'EMPLOYEE NO.    EMPLOYEE NAME'.
    DISPLAY '------------    -------------'.
    MOVE EMP-ID   TO  D-ID.
    MOVE EMP-NAME TO  D-NAME.
SQL-ERR.
    DISPLAY 'HAVE  ERROR WITH CODE ===> ' SQLERRMC.
END-PGM.
    EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
    EXEC SQL ROLLBACK RELEASE END-EXEC.
    STOP RUN.
```

บทที่ 6
สรุปผลการวิจัย

ไนแอม เป็นแบบจำลองข้อมูลระดับแนวความคิดที่มีพื้นฐานมาจากวิธี ไบนารี รีเลชันชิพ โดยที่ มีแนวความคิดเกี่ยวกับโครงสร้างว่า ประกอบด้วย เอนทิตี ไทพ์, เลเบล ไทพ์, ซับไทพ์, อีลีเมนทารี แฟคท์ ไทพ์ และ คอนสเทรน และมีการ แทน (Represent) ด้วยรูปภาพ ซึ่งง่ายต่อการอ่านและเข้าใจในขั้นตอนการออกแบบ แต่ไนแอมก็ยังไม่สมบูรณ์พอที่จะแทนความสัมพันธ์ที่มักพบบ่อย ๆ ใน Real World ได้ และ ไนแอมก็ไม่สามารถที่ จะแยก เอนทิตี ไทพ์ กับ เอนทิตี ไทพ์ คลาส ได้

ดังนั้นผู้จัดทำจึงได้ทำการพัฒนาปรับปรุงไนแอมให้มีความสมบูรณ์และมีประสิทธิภาพมากขึ้น โดยทำให้ไนแอมสามารถแทนสภาพที่เป็นจริงใน Real World ให้ได้มากที่สุดเท่าที่จะมากได้ จึงเรียกว่า ไนแอม$^{++}$ พร้อมทั้งได้สร้างแอลกอริธึมสำหรับใช้ในการแปลงจาก ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ และแบบเชิงวัตถุ เนื่องจากว่าปัจจุบันระบบจัดการฐานข้อมูลที่นิยมใช้กันอยู่เป็นจำนวนมากคือ ระบบจัดการฐานข้อมูลแบบเชิงสัมพันธ์ และ ลักษณะงานบางอย่างเหมาะสมกับโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ด้วย และได้สร้างแอลกอริธึมในการแปลง ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบเชิงวัตถุ เนื่องจากลักษณะของงานบางระบบมีข้อมูลที่มีโครงสร้างสลับซับซ้อน (Complex Object) ซึ่งไม่เหมาะกับโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ เช่น งานออกแบบทางวิศวกรรม และข้อมูลรายละเอียดของวงจรไฟฟ้า หรือเครื่องจักร หรือส่วนประกอบของรถยนต์ หรืออะไรก็ตามที่เป็นวัตถุเชิงซ้อน (Complex Object) และสร้าง Source โปรแกรมซึ่งเป็น Include File ภาษา C^{++} สำหรับนำไปเขียนโปรแกรมเชิงวัตถุ (OOP) ด้วย

เพื่ออำนวยความสะดวกให้กับนักวิเคราะห์ระบบที่ใช้ ไนแอม$^{++}$ เป็นแบบจำลองข้อมูลจึงได้ออกแบบ เมต้าดอนเวบชวลสกีมม่า โดยใช้หลักเกณฑ์ของไนแอม$^{++}$ ทำให้ดอนเซบชวลสกีมม่าคือ ไนแอม$^{++}$ ที่อธิบายกฎไวยกรณ์ของไนแอม$^{++}$ เอง แล้วนำเมต้าดอนเซบชวลสกีมม่าที่ได้มาสร้างเป็น เทเบิล (Table) เพื่อเก็บข้อมูลสำหรับทำ User Interface ของเครื่องมือ (Tool) คือ N2DB (NIAM^{++} TO DATABASE) ซึ่งเป็นเครื่องมือช่วยในการแปลง ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบเชิงสัมพันธ์ และแบบเชิงวัตถุตามแอลกอริธึมที่ได้เสนอไปแล้ว

การออกแบบฐานข้อมูลแบบเชิงสัมพันธ์ (RDB) ถือว่าเป็นศาสตร์ (Science) เพราะว่ามีทฤษฎีทางคณิตศาสตร์รองรับ คือทฤษฎีของกลุ่ม (Set Theory) แต่ การออกแบบ

ฐานข้อมูลเชิงวัตถุ (OODB) ยังถือว่าเป็นศิลป์ (Art) ไม่ใช้ศาสตร์ (Science) เนื่องจากไม่มีทฤษฎีทางคณิตศาสตร์มาอธิบาย และรองรับอย่างชัดเจนจึงทำให้ความสำเร็จของการออกแบบฐานข้อมูลแบบเชิงวัตถุนี้ขึ้นอยู่กับ ความสามารถ ความชำนาญ และทักษะส่วนบุคคลของผู้ออกแบบเอง ซึ่งไม่มีหลักเกณฑ์ที่จะนำมาอธิบายได้อย่างแน่นอน

ดังนั้น งานวิจัยที่เสนอในวิทยานิพนธ์ฉบับนี้จึงมีประโยชน์มาก สำหรับผู้ออกแบบฐานข้อมูล เพราะสามารถใช้ ไนแอม$^{++}$ เป็น แบบจำลองข้อมูล ในการออกแบบระบบ แล้วใช้ N2DB สร้างโครงสร้างฐานข้อมูลทั้งแบบเชิงสัมพันธ์ และแบบเชิงวัตถุ ได้ตามวัตถุประสงค์การใช้งาน

เอกสารอ้างอิง

[1] Thorsten Hartmann, Ralf Jungclavs, Gunter Saake,"Aggregation in a Behavior Oriented Objected Model*", Lecture Notes in Computer Science, The Netherlands, June/July 1992

[2] Griethuysen, J.J.V. et al.(ed.), "Concepts and Terminology for the Conceptual Schema", ISO/TC97/SC5/WG3.

[3] JOBY J. TEOREY, DONGQING YANG, JAMES P. FRY, "A Logical Design Methodology for Relational Database Using the Extended Entity Relationship Model", Computing Serveys, Vol.18,No.2, June 1986

[4] Michael R. Blaha, William J., James E., "Relational Database Design Using an Objected-Oriented Methodology", Communications of the ACM, Vol.31, No.4, April 1988

[5] James T.Perry, Joseph G.Lateer,"Understandanding Oracle" Oracle Corporation,Belmont,California,USA. 1987

[6] "SQL*Forms Designer's Reference" Oracle Corporation,Belmont, California,USA. 1987

[7] "Pro*Cobol Reference" Oracle Corporation, Belmont, California, USA. 1987

[8] "Pro*Cobol Supplememt" Oracle Corporation, Belmont, California, USA. 1987

[9] C.J. Date, "An Introduction To Database System", Volume I,Fifth Edition, Addison-Wesley Publishing Company, inc.

[10] John G Hughe, "Object-Oriented Database", Prentice-Hall

[11] Ian Sommerville, "Software Enginerring", Fifth Edition,Addison-Wesley Publishing Company, inc.

[12] Ulka Rodgers, "Oracle A Database Developer's Guide", Yourdon Press Prentice Hall Buliding Englewood Cliffs, New Jersy 07632

[13] ดร.ศุภมิตร จิตตะยโศธร,(2535),"อนาคตฐานข้อมูล Object-Oriented", คอมพิวเตอร์รีวิว,ปีที่10,ฉบับที่ 96,สิงหาคม

[14] วีร์ ตั้งมั่นภักดีพงษ์, "การพัฒนาระบบฐานข้อมูลด้วยภาษาซี", บริษัทซีเอ็ดยูเคชั่นจำกัด

[15] ดร.ดวงแก้ว สวามิภักดิ์,"ระบบฐานข้อมูล", บริษัทซีเอ็ดยูเคชั่นจำกัด

[16] นพดล ตังควรรณวานิชม,สรรค์เสกขุนทด,"หลักการโปรแกรม Object-Oriented และภาษา C^{++}", บริษัทซีเอ็ดยูเคชั่นจำกัด

[17] Richard S.Wiener, Lewis J.Pirson เรียบเรียงโดย ปรีดา ลิ้มถาวิลม, "ภาษา C^{++} เพื่อการใช้งาน", บริษัทซีเอ็ดยูเคชั่นจำกัด

ภาคผนวก ก.

คู่มือการใช้ระบบ N2DB

คู่มือ
การใช้ระบบ N2DB

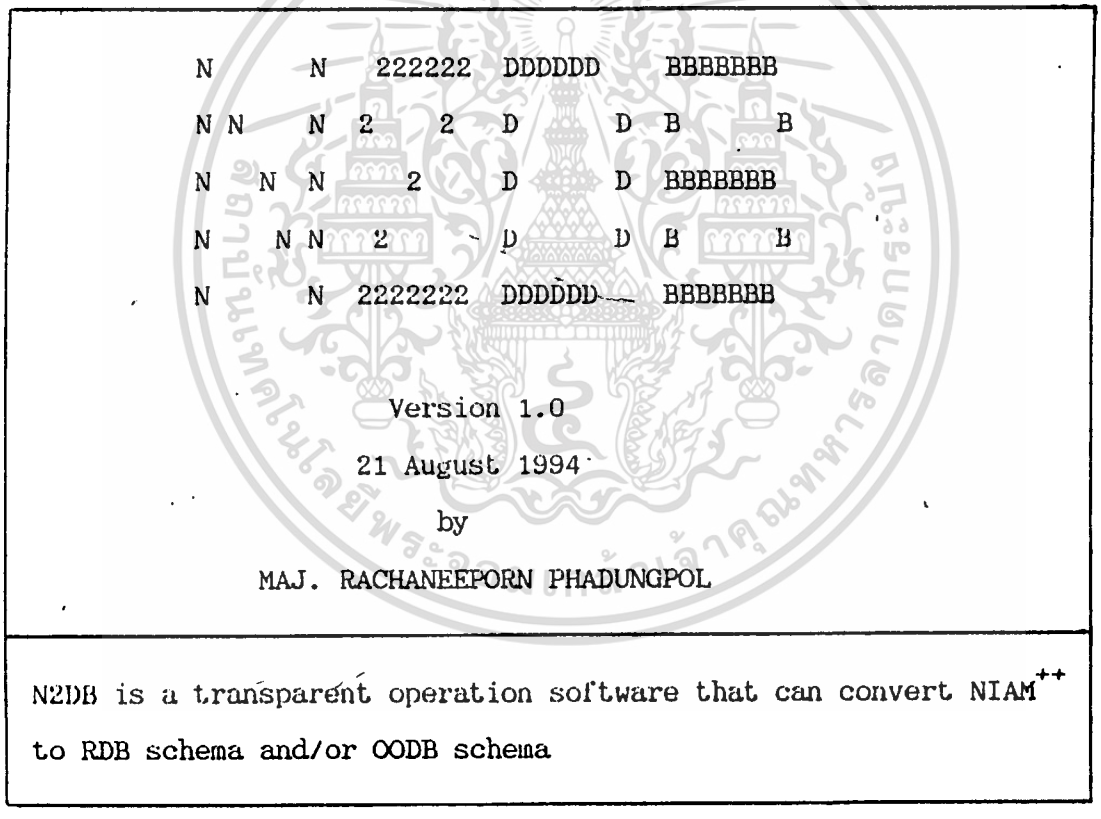
N2DB ใช้สำหรับ MAP รูปภาพจาก ไนแอม$^{++}$ ไปเป็นข้อมูลเก็บไว้ในฐานข้อมูลแบบเชิงสัมพันธ์

N2DB มีรายละเอียดทั้งหมด 9 รายการ ตามองค์ประกอบพื้นที่เป็นรูปภาพ (Graph) ในไนแอม$^{++}$

ขั้นตอนการทำงานในระบบ N2DB

1. Login เข้าสู่ระบบของ Oracle

2. Run Forms ชื่อ N2DB จะได้จอภาพดังรูปที่ 1 เป็นการบอกให้รู้ว่า เข้าสู่ระบบ N2DB แล้วกด Enter จะได้จอภาพดังรูปที่ 2



รูปที่ 1 แสดง Logo ของระบบ N2DB

```
                              N2DB
                          MAIN  MENU

1. ENTITY TYPE                    : 6. ENTITY TYPE CLASS
2. RELATION of ENTITY TYPE        : 7. RELATION    of ENTITY TYPE CLASS
3. SUBTYPE  of ENTITY TYPE        : 8. SUBTYPE     of ENTITY TYPE CLASS
4. SUB-PART-OF of ENTITY TYPE     : 9. SUB-PART-OF of ENTITY TYPE CLASS
5. CONSTRAINT                     :

                <<< PLEASE SELECT ONE CHOICE   _ >>>

 < F3 : EXIT >    < ENTER : CONTINUE >
```

รูปที่ 2 MAIN MENU ของระบบ N2DB

3. เลือกหัวข้อตั้งแต่ 1 ถึง 9 เพื่อเข้าไปทำงานในระบบย่อยที่ต้องการ โดย Key ตัวเลข 1 ถึง 9 แล้วกด Enter

4. ถ้าเลือกข้อ 1 จะได้จอภาพดังรูปที่ 3

```
                              N2DB
                        ENTITY  TYPE

FUNCTION ==> _        (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME -------
ENTITY TYPE NO. ---




              <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 3 MENU สำหรับ Maintenance ENTITY TYPE

- เลือก Function ที่ต้องการ Maintenance ข้อมูล โดยที่ A = Add, U = Update, D = Delete และ L = List ข้อมูลทั้งหมดที่ต้องการออกมาดู สิ่งที่สำคัญที่ต้องใส่ คือ System Name เพื่อบอกว่าต้องการที่จะ Maintenance ข้อมูลในระบบใด และสิ่งที่สำคัญอีกประการหนึ่งก็คือ Entity Type No. จะต้องขึ้นต้นด้วย "E" เสมอ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name และ Entity Type No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 4 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 3

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name และ Entity Type No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 4 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 3

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name และ Entity Type No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 4 จะทำการลบข้อมูล

ดังกล่าวออกจากระบบ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 3

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง System Name และ Entity Type No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 5 กด Enter จะกลับไปที่จอภาพดังรูปที่ 3

รายละเอียดที่ต้องใส่ในแต่ละจอภาพของระบบ Entity คือ

- ENTITY TYPE NAME ใส่ได้ 20 ตัวอักษร
- LABEL NAME ใส่ได้ 20 ตัวอักษร
- DATA TYPE ใส่ได้ 1 ตัวอักษร
  - C = Character
  - N = Number
  - D = Date
- ENTITY LENGTH ใส่ได้ 3 ตัวอักษร

```
                              N2DB
                         ENTITY  TYPE

SYSTEM NAME  --------

ENTITY TYPE NO. ---   ENTITY TYPE NAME ____________________
                      LABEL NAME      ____________________
                      DATA TYPE       _ (C:CHAR, N:NUMBER, D:DATE)
                      ENTITY LENGTH   ---

< F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 4 แสดงรายละเอียดต่าง ๆของ ENTITY TYPE

N2DB

ENTITY TYPE

ENTITY TYPE NO.	ENTITY NAME	LABEL NAME	DATA TYPE	LENGTH
---	----------	----------	-	---
---	----------	----------	-	---
---	----------	----------	-	---
---	----------	----------	-	---

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 5 LIST ของข้อมูล ENTITY TYPE

5. ถ้าเลือกข้อ 2. จะได้จอภาพดังรูปที่ 6

```
                              N2DB
                RELATION of ENTITY  TYPE

FUNCTION ==> _          (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
ENTITY TYPE NO. ---    FACT TYPE NO. ---




              <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 6 MENU สำหรับ Maintenance RELATION of ENTITY TYPE

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name, Entity Type No. และ Fact Type No. ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type No. และ Fact Type No. กด Enter ได้จอภาพดังรูปที่ 7 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 6

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type No. และ Fact Type No. กด Enter ได้จอภาพดังรูปที่ 7 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 6

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type No. และ Fact Type No. กด Enter จะได้จอภาพดังรูปที่ 7 จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 6

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง

System Name ที่ต้องการ แล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพ ดังรูปที่ 8. กด Enter จะกลับไปจอภาพดังรูปที่ 6

รายละเอียดที่ต้องใส่ในแต่ละจอภาพของระบบ Relation of Entity Type คือ

- FACT TYPE NO. ใส่ตัวเลขได้ 2 ตัว สำหรับบอกว่าเป็น Fact Type ตัวที่เท่าใดในระบบ

- UNIQUENESS CONSTRAINT ใส่ Y หรือ N เท่านั้น เพื่อบอกว่ามี Uniqueness Constraint คลุมอยู่ที่ Role ของ Fact Type ที่ติดต่อกับ Entity Type นี้หรือไม่

- UNIQUENESS CONSTRAINT COVER ROLE TYPE ใส่เลข 1, 2, หรือ 3 เพื่อบ่งบอกชนิดของ Uniqueness Constraint ที่คลุมจำนวน Role ของ Fact Type ว่าเป็นแบบใด

ใส่ 1 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ 1 Role และ Fact Type เป็น Binary Fact Type

ใส่ 2 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ n-1 Role

ใส่ 3 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ n Role

```
                        N2DB
                RELATION of ENTITY TYPE


SYSTEM NAME  --------
ENTITY TYPE NO. ---  FACT TYPE NO. ---
            UNIQUENESS CONSTRAINT  _ (Y:YES, N:NO)
            UNIQUENESS CONSTRAINT  _ (1:COVER 1   ROLE, n =  2
            COVER ROLE TYPE          2:COVER n-1 ROLE, n =  2
                                     3:COVER n   ROLE,. n => 2)

< F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 7 แสดงรายละเอียดต่าง ๆของ RELATION of ENTITY TYPE

N2DB

RELATION of ENTITY TYPE

ENTITY TYPE NO.	FACT TYPE NO.	UNIQUENESS	UNIQUENESS COVER
---	---	-	-
---	---	-	-
---	---	-	-
---	---	-	-

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 8 LIST ของข้อมูล RELATION of ENTITY TYPE

6. ถ้าเลือกข้อ 3 จะได้จอภาพดังรูปที่ 9

```
                              N2DB
                  SUBTYPE of ENTITY  TYPE

FUNCTION ==> _        (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
ENTITY TYPE NO. ___   SUBTYPE NO. ___




              <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 9 MENU สำหรับ Maintenance SUBTYPE of ENTITY TYPE

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name ที่ต้องการ

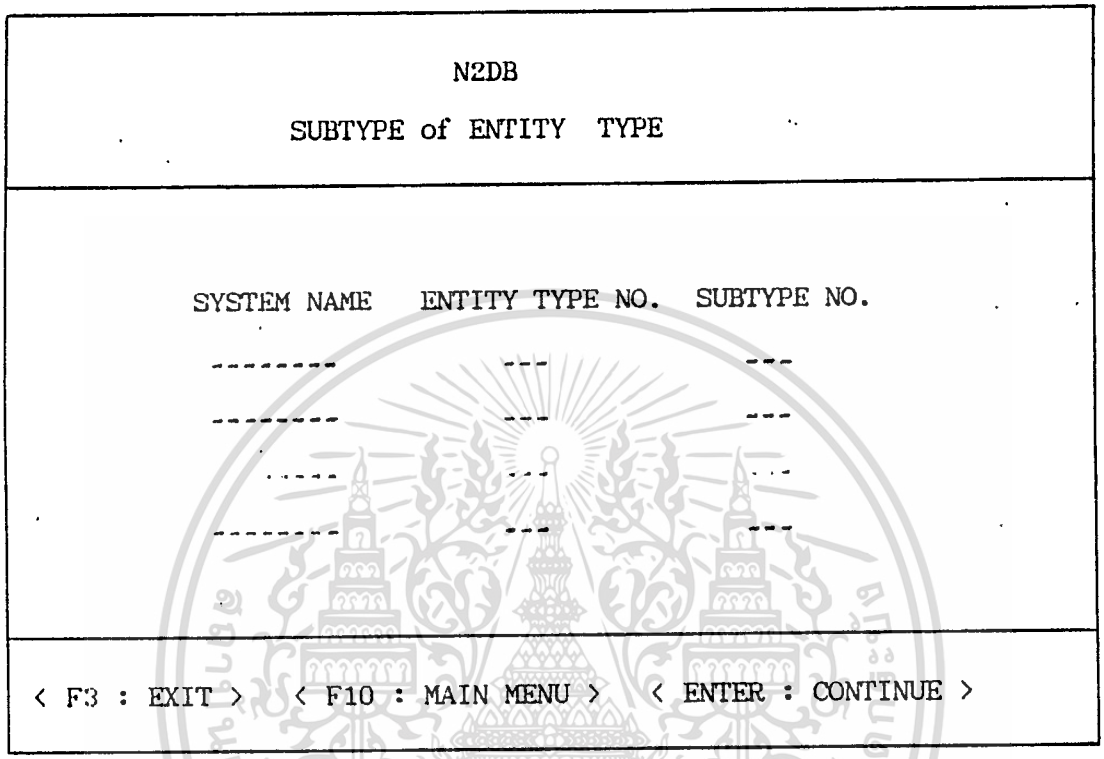
- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด Enter จะได้จอภาพดังรูปที่ 9 กด Enter จะได้จอภาพดังรูปที่ 9

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด Enter จะได้จอภาพดังรูปที่ 9 แล้วกด Enter จะได้จอภาพดังรูปที่ 9

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด Enter จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะได้จอภาพดังรูปที่ 9

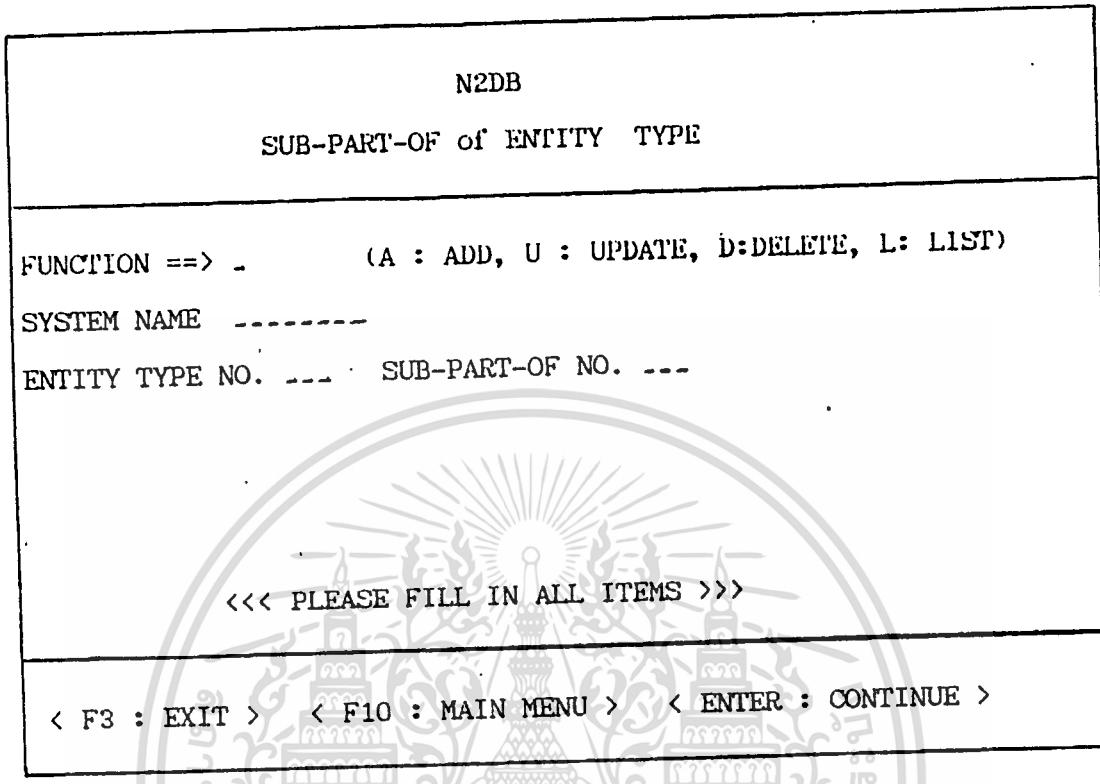
- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง

System Name ที่ต้องการแล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดังรูปที่ 10 กด Enter จะกลับไปจอภาพดังรูปที่ 9



รูปที่ 10 LIST ของข้อมูล SUBTYPE of ENTITY TYPE

7. ถ้าเลือกข้อ 4 จะได้จอภาพดังรูปที่ 11



รูปที่ 11 MENU สำหรับ Maintenance SUB-PART-OF of ENTITY TYPE

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-of กด Enter จะได้จอภาพดังรูปที่ 11 แล้วกด Enter ได้จอภาพดังรูปที่ 11

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-of กด Enter จะได้จอภาพดังรูปที่ 11 ใส่รายละเอียดต่างๆ แล้วกด Enter ได้จอภาพดังรูปที่ 11

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-of กด Enter จะได้จอภาพดังรูปที่ 11 จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะได้จอภาพดังรูปที่ 11

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง System Name ที่ต้องการแล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดัง รูปที่ 12 กด Enter จะกลับไปจอภาพดังรูปที่ 11

```
                              N2DB
                SUB-PART-OF of ENTITY  TYPE

    SYSTEM NAME    ENTITY TYPE NO.  SUB-PART-OF NO.
    --------          ---              ---
    --------          ---              ---
    --------          ---              ---
    --------          ---              ---

< F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 12 LIST ของข้อมูล SUB-PART-OF of ENTITY TYPE

8. ถ้าเลือกข้อ 5 จะได้จอภาพดังรูปที่ 13

```
                              N2DB
                          CONSTRAINT

FUNCTION ==> _        (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
CONSTRAINT NO. ___   FACT TYPE NO. ___




            <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 13 MENU สำหรับ Maintenance CONSTRAINT

- ถ้าเลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Constraint No. และ Type No. แล้วกด Enter จะได้จอภาพดังรูปที่ 14 ใส่รายละเอียดต่างๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 13

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Constraint No. และ Type No. แล้วกด Enter จะได้จอภาพดังรูปที่ 14 ใส่รายละเอียดต่างๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 13

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Constraint No. และ Type No. แล้วกด Enter จะได้จอภาพดังรูปที่ 14 จะทำการลบข้อมูลดังกล่าวออกจากระบบแล้วกด Enter จะกลับไปจอภาพดังรูปที่ 13

- ถ้าต้องการดูข้อมูลทั้งหมดที่ต้องการใส่ Function = L พร้อมทั้ง System

Name แล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดังรูปที่ 15 กด Enter จะกลับไปจอภาพดังรูปที่ 13

รายละเอียดที่สำคัญที่ต้องใส่ในระบบ Constraint คือ Constraint Type ในระบบนี้เตรียมไว้เพื่อ 3 ชนิด คือ

X = Exclusion

C = Cardinality

U = Intra Fact Type

```
                                N2DB
                            CONSTRAINT

FUNCTION ==> _          (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
CONSTRAINT NO. ___   FACT TYPE NO. ___
                     CONSTRAINT TYPE _ (X : EXCLUSION,
                                        C : CADINALITY,
                                        U : INTRA FACT TYPE)
               <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 14 MENU สำหรับ Maintenance CONSTRAINT

```
                              N2DB

                          CONSTRAINT

     CONSTRAINT NO.   FACT TYPE NO.   CONSTRAINT TYPE

          ---              ---              -
          ---              ---              -
          ---              ---              -
          ---              ---              -


< F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 15 LIST ของข้อมูล CONSTRAINT

9. ถ้าเลือกข้อ 6 จะได้จอภาพดังรูปที่ 16

```
                              N2DB

                      ENTITY TYPE CLASS

FUNCTION ==> _        (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  ________
ENTITY TYPE CLASS NO. ___



            <<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 16 MENU สำหรับ Maintenance ENTITY TYPE CLASS

- เลือก Function ที่ต้องการ Maintenance ข้อมูลโดยที่ A = Add, U = Update, D = Delete และ L = List ข้อมูลทั้งหมดที่ต้องการออกมาดู สิ่งที่สำคัญที่ต้องใส่ คือ System Name เพื่อบอกว่าต้องการที่จะ Maintenance ข้อมูลในระบบใด และสิ่งที่สำคัญอีกประการหนึ่งก็คือ Entity Type Class No. จะต้องขึ้นต้นด้วย "C" เสมอ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name และ Entity Type Class No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 17 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 16

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name และ Entity Type Class No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 17 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 16

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name และ Entity TypeClass No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 17 จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะกลับไปที่จอภาพดังรูปที่ 16

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง System Name และ Entity Type Class No. ที่ต้องการ กด Enter จะได้จอภาพดังรูปที่ 18 กด Enter จะกลับไปที่จอภาพดังรูปที่ 16

รายละเอียดที่ต้องใส่ในแต่ละจอภาพของระบบ Entity คือ

- ENTITY TYPE CLASS NAME ใส่ได้ 20 ตัวอักษร
- LABEL CLASS NAME ใส่ได้ 20 ตัวอักษร
- DATA TYPE ใส่ได้ 1 ตัวอักษร
  - C = Character
  - N = Numeber
  - D = Date
- ENTITY LENGTH ใส่ได้ 3 ตัวอักษร

```
                              N2DB
                    ENTITY TYPE CLASS

SYSTEM NAME --------

ENTITY TYPE CLASS NO. ---  ENTITY TYPE CALSSNAME ----------------
                      LABEL NAME        --------------------
                      DATA TYPE         _ (C:CHAR, N:NUMBER, D:DATE)
                      ENTITY LENGTH     ---

 < F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 17 แสดงรายละเอียดต่าง ๆของ ENTITY TYPE CLASS

```
                              N2DB
                    ENTITY TYPE CLASS

ENTITY TYPE     ENTITY  TYPE      LABEL NAME     DATA TYPE   LENGTH
  CLASS NO.       CLASS NAME

    ---        ---------------- ----------------    -         ---
    ---        ---------------- ----------------    -         ---
    ---        ---------------- ----------------    -         ---
    ---        ---------------- ----------------    -         ---

 < F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 18 LIST ของข้อมูล ENTITY TYPE CLASS

10. ถ้าเลือกข้อ 7 จะได้จอภาพดังรูปที่ 19

```
                              N2DB
                RELATION of ENTITY TYPE CLASS

FUNCTION ==> _          (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
ENTITY TYPE CLASS NO. ---    FACT TYPE NO. ---




              <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 19 MENU สำหรับ maintenance RELATION of ENTITY TYPE CLASS

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name, Entity Type Class No. และ Fact Type No. ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type Class No. และ Fact Type No. กด Enter ได้จอภาพดังรูปที่ 20 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 19

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type Class No. และ Fact Type No. กด Enter ได้จอภาพดังรูปที่ 20 ใส่รายละเอียดต่าง ๆ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 19

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type Class No. และ Fact Type No. กด Enter จะได้จอภาพดังรูปที่ 20 จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะกลับไปจอภาพดังรูปที่ 19

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง

System Name ที่ต้องการ แล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดังรูปที่ 21 กด Enter จะกลับไปจอภาพดังรูปที่ 19

รายละเอียดที่ต้องใส่ในแต่ละจอภาพของระบบ Relation of Entity Type Class คือ

- FACT TYPE NO. ใส่ตัวเลขได้ 2 ตัว สำหรับบอกว่าเป็น Fact Type ตัวที่เท่าใดในระบบ

- UNIQUENESS CONSTRAINT ใส่ Y หรือ N เท่านั้น เพื่อบอกว่ามี Uniqueness Constraint คลุมอยู่ที่ Role ของ Fact Type ที่ติดต่อกับ Entity Type Class นี้หรือไม่

- UNIQUENESS CONSTRAINT COVER ROLE TYPE ใส่เลข 1, 2, หรือ 3 เพื่อบ่งบอกชนิดของ Uniqueness Constraint ที่คลุมจำนวน Role ของ Fact Type ว่าเป็นแบบใด

ใส่ 1 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ 1 Role และ Fact Type เป็น Binary Fact Type

ใส่ 2 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ N-1 Role

ใส่ 3 หมายถึง Uniqueness Constraint คลุม Fact Type อยู่ N Role

```
                          N2DB
                RELATION of ENTITY TYPE CLASS

SYSTEM NAME  --------
ENTITY TYPE CLASS NO. ___  FACT TYPE NO. ]]]
            UNIGUENESS CONSTRAINT  - (Y:YES, N:NO)
            UNIQUENESS CONSTRAINT  - (1:COVER 1   ROLE, n =  2
            COVER ROLE TYPE          2:COVER n-1 ROLE, n =  2
                                     3:COVER n   ROLE, n => 2)

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 20 แสดงรายละเอียดต่าง ๆของ RELATION of ENTITY TYPE CLASS

N2DB

RELATION of ENTITY TYPE CLASS

ENTITY TYPE CLASS NO.	FACT TYPE NO.	UNIQUENESS	UNIQUENESS COVER
---	---	-	-
---	---	-	-
---	---	-	-
---	---	-	-

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 21 LIST ของข้อมูล RELATION of ENTITY TYPE CLASS

11. ถ้าเลือกข้อ 8 จะได้จอภาพดังรูปที่ 22

```
                              N2DB
                SUBTYPE of ENTITY TYPE CLASS

FUNCTION --> _          (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME --------
ENTITY TYPE CLASS NO. ---    SUBTYPE NO. ---




                  <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >   < F10 : MAIN MENU >   < ENTER : CONTINUE >
```

รูปที่ 22 MENU สำหรับ maintenance SUBTYPE of ENTITY TYPE CLASS

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด Enter จะได้จอภาพดังรูปที่ 22 ใส่รายละเอียดต่างๆ แล้วกด Enter จะได้จอภาพดังรูปที่ 22

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด Enter จะได้จอภาพดังรูปที่ 22 ใส่รายละเอียดต่างๆ แล้วกด Enter จะได้จอภาพดังรูปที่ 22

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Supertype และ Entity Type No. ที่เป็น Subtype กด

Enter จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะได้จอภาพดังรูปที่ 22

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง System Name ที่ต้องการแล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดังรูปที่ 23 กด Enter จะกลับไปจอภาพดังรูปที่ 22

N2DB

SUBTYPE of ENTITY TYPE CLASS

SYSTEM NAME	ENTITY TYPE CLASS NO.	SUBTYPE NO.
--------	---	---
--------	---	---
--------	---	---
--------	---	---

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 23 LIST ของข้อมูล SUBTYPE of ENTITY TYPE CLASS

12. ถ้าเลือกข้อ 9 จะได้จอภาพดังรูปที่ 24

```
                              N2DB
                SUB-PART-OF of ENTITY TYPE CLASS

FUNCTION ==> _          (A : ADD, U : UPDATE, D:DELETE, L: LIST)
SYSTEM NAME  --------
ENTITY TYPE CLASS NO. ___    SUB-PART-OF NO. ]]]




                   <<< PLEASE FILL IN ALL ITEMS >>>

 < F3 : EXIT >    < F10 : MAIN MENU >    < ENTER : CONTINUE >
```

รูปที่ 24 MENU สำหรับ maintenance SUB-PART-OF of ENTITY TYPE CLASS

- เลือก Function ที่ต้องการ Maintenance ข้อมูล พร้อมทั้ง System Name ที่ต้องการ

- ถ้าต้องการเพิ่มข้อมูลใส่ Function = A พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-of กด Enter จะได้จอภาพดังรูปที่ 24 ใส่รายละเอียดต่างๆ แล้วกด Enter ได้จอภาพดังรูปที่ 24

- ถ้าต้องการแก้ไขข้อมูลใส่ Function = U พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-of กด Enter จะได้จอภาพดังรูปที่ 24 ใส่รายละเอียดต่างๆ แล้วกด Enter ได้จอภาพดังรูปที่ 24

- ถ้าต้องการลบข้อมูลใส่ Function = D พร้อมทั้ง System Name, Entity Type Class No. ที่เป็น Super-part-of และ Entity Type No. ที่เป็น Sub-part-

of กด Enter จะได้จอภาพดังรูปที่ 24 จะทำการลบข้อมูลดังกล่าวออกจากระบบ แล้วกด Enter จะได้จอภาพดังรูปที่ 24

- ถ้าต้องการดูข้อมูลทั้งหมดในระบบที่ต้องการใส่ Function = L พร้อมทั้ง System Name ที่ต้องการแล้วกด Enter จะได้รายละเอียดของข้อมูลตามจอภาพดัง รูปที่ 25 กด Enter จะกลับไปจอภาพดังรูปที่ 23

N2DB

SUB-PART-OF of ENTITY TYPE CLASS

SYSTEM NAME	ENTITY TYPE CLASS NO.	SUB-PART-OF NO.
--------	---	---
--------	---	---
--------	---	---
--------	---	---

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 25 LIST ของข้อมูล SUB-PART-OF of ENTITY TYPE CLASS

ภาคผนวก ข

บทความเรื่อง การออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ ENIAM

ได้ลงตีพิมพ์ในวิศวสารลาดกระบัง

วิศวสารลาดกระบัง

Ladkrabang Engineering Journal

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง กรุงเทพฯ 10520

Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520

10 เมษายน พ.ศ. 2538

อ้างอิง LEJ_021

เรื่อง ตอบรับการตีพิมพ์บทความในวิศวสารลาดกระบัง

เรียน คุณ รัชนีพร ผดุงผล

ผศ.ดร. ศุภมิตร จิตตะยโศธร,

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

บทความเรื่อง "การออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ ENIAM"

ตามที่ท่านได้ส่งบทความดังกล่าว ได้ผ่านการพิจารณาจากกรรมการผู้ทรงคุณวุฒิแล้ว ขณะนี้ทางกองบรรณาธิการวิศวสารลาดกระบังกำลังเตรียมการเพื่อตีพิมพ์ในวิศวสารลาดกระบังโดยเร็วที่สุด

จึงเรียนมาเพื่อทราบและขอขอบคุณในการส่งบทความมายังวิศวสารลาดกระบัง

ด้วยความนับถืออย่างสูง

(ดร. สมศักดิ์ ชุมช่วย)

บรรณาธิการวิศวสารลาดกระบัง

การออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ ENIAM

(Relational Database Design Using ENIAM)

ศุภมิตร จิตตะยโศธร * พ.ต.หญิง รัชนีพร ผดุงผล **

บทคัดย่อ

ในความเป็นจริงนั้น วัตถุสิ่งของต่าง ๆ ส่วนใหญ่จะมีโครงสร้างสลับซับซ้อน ดังนั้นในขั้นตอนการออกแบบฐานข้อมูล จึงจำเป็นต้องพิจารณาหาแบบจำลองข้อมูลที่เหมาะสม ซึ่งทฤษฎีของ ไนแอม (NIAM : Nijsson's Information Analysis Methodology) ถูกเลือกว่าเป็น นิยามข้อมูลระดับแนวความคิด (Conceptual Schema Model) ที่เหมาะสม ดังนั้นจึงใช้ ไนแอม เป็นพื้นฐานในการออกแบบฐานข้อมูลเชิงสัมพันธ์แต่เนื่องจากไนแอม ไม่ได้ กำหนดสัญลักษณ์ที่ใช้แทนความสัมพันธ์แบบ Aggregation ไว้ ซึ่งในความจริงนั้น วัตถุ หรือ สิ่งต่าง ๆ ที่เกี่ยวข้องในชีวิตประจำวัน มักเป็น คอมเพลกซ์ ออบเจกต์ (Complex Object) ที่มีความสัมพันธ์แบบ Aggregation เสียส่วนใหญ่ ดังนั้นจึงได้เพิ่มเติมสัญลักษณ์บางอย่างเข้า ไปเพื่อที่จะแทนสภาพที่เป็นจริงให้ได้มากที่สุดเท่าที่จะมากได้ จึงเรียกว่า อีไนแอม (ENIAM : Extended NIAM) พร้อมทั้ง เสนอ แอลกอริธึม (Algorithm) ในการแปลง อีไนแอม ไปเป็น Relational Schema

ABSTRACT

Nijssen's Information Analysis Methodology(NIAM) is selected to be the right Conceptual Schema Model for designing the relational database. NIAM does not define the sign for Aggregation relation. But most objects are Complex Object in Aggregation relation form. Some signs are added to NIAM for representing the relation in the real world. The new methodology is called Extended NIAM (ENIAM). Algorithm for mapping ENIAM to Relational Schema is also presented.

* ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

** นักศึกษาปริญญาโท โครงการคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

1. บทนำ

ที่จริงแล้ว ออบเจกต์ (Object) หรือ วัตถุสิ่งของต่างๆ ส่วนใหญ่จะมีโครงสร้างที่สลับซับซ้อน เช่น รถยนต์ ประกอบด้วยส่วนต่าง ๆ ดังนี้คือ ระบบไฟฟ้า, ระบบส่งกำลัง, ระบบช่วงล่าง, ระบบตัวถังและระบบเครื่องยนต์ ซึ่งระบบเครื่องยนต์เองก็ยัง ประกอบด้วย ระบบน้ำมันเชื้อเพลิง, ระบบหล่อลื่น, ระบบหล่อเย็น และระบบจุดระเบิด ดังนั้น เครื่องยนต์ก็ เป็น คอมเพลกซ์ ออบเจกต์ จะเห็นได้ว่า คอมเพลกซ์ ออบเจกต์สามารถที่จะประกอบด้วย คอมเพลกซ์ ออบเจกต์ อื่นได้เช่นกัน

คอมเพลกซ์ ออบเจกต์ [1] หมายถึง การรวมกันของ ซับออบเจกต์ (Subobject) ที่เป็นโครงสร้าง ของ ออบเจกต์ แบ่งออกเป็น 2 ชนิด

1. Non-disjoint Complex Object
2. Disjoint Complex Object

Non-disjoint Complex Object คือ คอมเพลกซ์ ออบเจกต์ที่เกิดจากการรวมกันของหลาย ๆ ซับออบเจกต์ ที่มาจาก คอมเพลกซ์ ออบเจกต์ อื่นด้วย

Disjoint Complex Object คือ คอมเพลกซ์ ออบเจกต์ ที่เกิดจากการรวมกันของหลาย ๆ ซับออบเจกต์ ที่ไม่เกี่ยวข้องกับ คอมเพลกซ์ ออบเจกต์ อื่น

2. แบบจำลองข้อมูลไนแอม (The NIAM Conceptual Model)

ISO (The International Standard Organization) เสนอ สถาปัตยกรรมฐานข้อมูล แบบ 3 ระดับ (Three-schema Architecture [2]) ขึ้นเป็นมาตรฐานของ ฐานข้อมูล ซึ่งนิยามของ ฐานข้อมูลถูกแบ่งออกเป็น 3 ระดับ คือ

1. External Schema เป็นการอธิบาย User View ของ Conceptual Schema
2. Conceptual Schema เป็นการแสดงความสัมพันธ์ระหว่างข้อมูลโดยอาศัย แบบจำลองข้อมูล (Data Model) เป็นตัวแสดงความสัมพันธ์ระหว่างข้อมูล
3. Internal Schema เป็นการกำหนดว่า Fact จะถูกแทน (Represent) ในหน่วยความจำข้างใน (Internal Storage) และการเข้าถึงข้อมูลเป็นอย่างไร

Data Modeling เป็นขั้นตอนแรกในการออกแบบฐานข้อมูล คือเป็นช่วงการกำหนดโครงสร้างของ ฐานข้อมูล ซึ่ง Data Modeling มีขั้นตอนที่สำคัญดังนี้

1. A Conceptual Design Phase จะเกี่ยวข้องกับการออกแบบ Conceptual Schema ที่เป็น Abstract ของ Real World
2. The Design of a Logical Data Structure คือ การ Represent Schema ซึ่งจะถูก Map ไปบน Actual Implementation

2.1 ส่วนประกอบของ ไนแอม

ในราวปลายปี ค.ศ. 1982 [3] ศาสตราจารย์ จี เอ็ม ไนเซน ได้เสนอแบบจำลองข้อมูล ไนแอม ซึ่งถูกเลือกว่าเป็น Conceptual Model ที่เหมาะสม และมีการแทน (Represent) ในรูปของ Graph ที่ง่ายต่อการอ่าน และเข้าใจในขั้นตอนการออกแบบไนแอมมีส่วนประกอบพื้นฐานที่สำคัญดังนี้

1. Entity Type
2. Label Type
3. Elementary Fact Type
4. Subtype
5. Constraint

Entity Type คือ กลุ่มของ Abstract หรือ Real Entity ซึ่งอาจเป็นสิ่งที่จับต้องได้หรือไม่ได้ เช่น PERSON, DEPARTMENT, COMPANY

Label Type คือ ตัวที่ใช้บอกความแตกต่างของ Entity ภายใน Entity Type (ใช้เป็น Identifer หรือ Key)

Elementary Fact Type คือ ตัวที่ใช้แสดงความสัมพันธ์ระหว่าง Entity type ตั้งแต่ 2 Entity type ขึ้นไป

Subtype ใช้ในการกำหนด Hierarchies ที่ซับซ้อน ของEntity type และ Subtype ที่มีการถ่ายทอด Fact Type จาก Supertype ด้วย

Constraint เป็นเงื่อนไขที่จะทำให้แน่ชัดว่าฐานข้อมูลนั้น ถูกต้องและสมบูรณ์ตลอด (Consistency and Integrity) Constraint ที่ใช้ใน ไนแอม มีดังนี้

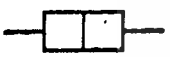
- Uniqueness Constraint แบ่งเป็น Inter Fact Type และ Intra Fact Type
- Mandatory Fact Constraint
- Exclusion Constraint
- Equality Constraint
- Subset Constraint

2.2 สัญลักษณ์ที่ใช้ใน ไนแอม

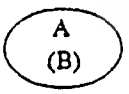
ไนแอม มีการแทน (Represent) ในรูปของ Graph ที่ง่ายต่อการอ่าน และ เข้าใจ สัญลักษณ์ที่ใช้ใน ไนแอม มีดังต่อไปนี้

แทน Entity Type A

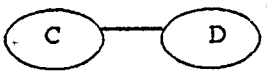
แทน Label Type B



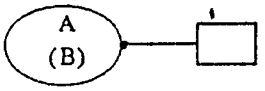
แทน Binary Fact Type



แทน Entity Type A และมี Label Type B



แทน Entity Type C เป็น Subtype ของ Entity Type D ซึ่งเป็น Supertype



แทน Mandatory Fact Constraint (ต้องการ Mandatory ที่ Entity Type ไหนก็ใส่ "." ที่ Entity Type นั้น)

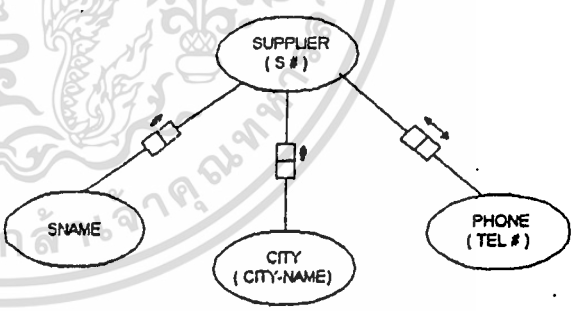
แทน Intra Fact Type Uniqueness Constraint

แทน Inter Fact Type Uniqueness Constraint

แทน Exclusion Constraint

แทน Equality Constraint

แทน Subset Constraint



รูป 2.1 ตัวอย่าง ไนแอม

จากรูป 2.1 สรุปได้ดังนี้

1. Entity Type SUPPLIER มี Label Type เป็น S#
2. Entity Type PHONE มี Label Type เป็น TEL#
3. Entity Type CITY มี Label Type เป็น CITY-NAME

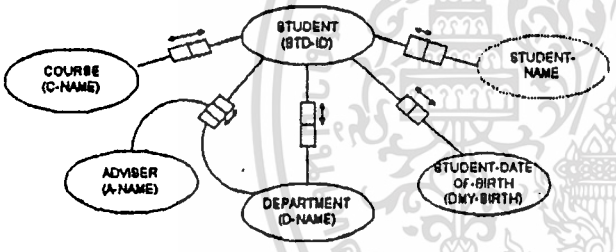
4. Entity Type SUPPLIER มี Elementary Fact Type เป็นตัวเชื่อมความสัมพันธ์กับ Entity Type PHONE, CITY

5.Entity Type SUPPLIER มี Reference Type เป็นตัวเชื่อมความสัมพันธ์กับ Label Type SNAME

2.3 Relationship Type

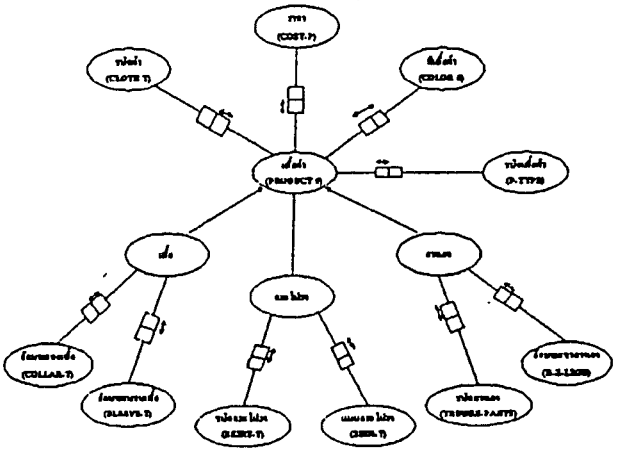
Relationship Type [4] ที่สำคัญ สามารถแบ่งได้เป็น 3 ชนิด

1. Association Relationship Type เป็นความสัมพันธ์ที่เชื่อม ออบเจกต์ ที่เป็น ออบเจกต์ อิสระเข้าด้วยกัน เช่น STUDENT มีความสัมพันธ์กับ COURSE และ STUDENT-NAME,STUDENT-DATE-OF-BIRTH สามารถใช้ไนแอมออกแบบได้ดังรูป 2.2 และจะสังเกตุ เห็นได้ว่าทั้ง Fact Type และ Reference Type ในไนแอมจัดได้ว่าเป็น Association Relationship Type



รูป 2.2 Association Relationship Type

2. Generalization Relationship Type หรือความสัมพันธ์แบบ Supertypo กับ Subtype นั้นคือ ความสัมพันธ์ที่แบ่งคุณลักษณะเฉพาะที่ร่วมกันออกเป็น Supertype และ แบ่งคุณลักษณะเฉพาะตัวออกเป็น Subtype เช่น ฐานข้อมูล ร้านขายเสื้อผ้า มีสินค้า เช่น
เสื้อ กระโปรง กางเกง ซึ่งสินค้าแต่ละชนิดจะมีคุณลักษณะเฉพาะของตัวเอง แต่จะมีข้อมูล ที่ใช้ร่วมกัน เช่น สีของกระโปรง สีกางเกง สีเสื้อ ชนิดของผ้าที่ใช้ สามารถใช้ไนแอม ออกแบบได้ ดังรูป 2.3



รูป 2.3 Generalization Relationsip Type

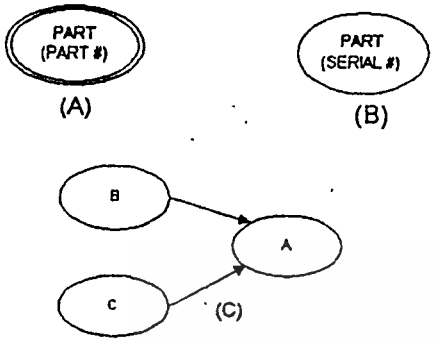
3. Aggregation Relationship Type หรือความสัมพันธ์แบบ Super-part-of กับ Sub-part-of คือความสัมพันธ์ของหลาย ๆ ออบเจกต์รวมเป็นออบเจกต์เดียว ตัวอย่าง เช่น รถยนต์ เกิดจากการรวมกันหลาย ๆ องค์ประกอบนั่นคือ ระบบเครื่องยนต์ ระบบช่วงล่าง ระบบส่งกำลัง ฯลฯ

ไนแอม ไม่ได้กำหนดรูปแบบของความสัมพันธ์ประเภทนี้ไว้ จึงไม่สามารถใช้สัญญลักษณ์ ของ ไนแอม แทนความสัมพันธ์ประเภทนี้ได้ และจะเห็นได้ว่าในความเป็นจริงนั้น วัตถุ หรือ สิ่งของต่าง ๆ ที่เกี่ยวข้องในชีวิตประจำวันมักเป็น คอมเพลกซ์ ออบเจกต์ ที่มีความสัมพันธ์แบบ Aggregation เสียส่วนใหญ่

3. อีไนแอม (ENIAM : Extended NIAM)

พิจารณาสัญลักษณ์ของแบบจำลองข้อมูลไนแอม มีแต่แนวความคิดของ Supertype และ Subtype ซึ่งสิ่งที่เกี่ยวข้องในชีวิตประจำวันส่วนใหญ่มักจะเป็น คอมเพลกซ์ ออบเจกต์ ที่มี ความสัมพันธ์แบบ Super-part-of และ Sub-part-of คือ ระบบใหญ่จะประกอบด้วยหลาย ส่วนย่อย ๆ และสิ่งที่สำคัญอีกอย่างหนึ่งก็คือ ไนแอม ไม่สามารถแยก Entity Type กับ
Entity Type Class ให้เห็นเด่นชัดได้ ดังนั้นจึงมี แนวความคิดที่จะเพิ่ม สัญลักษณ์บางประการเข้าไปใน ไนแอม เพื่อ

ใช้ในการออกแบบฐานข้อมูล ให้มีประสิทธิภาพมากยิ่งขึ้น ดังนั้นจึง เสนอ อีไนแอม และสัญลักษณ์ที่เพิ่มขึ้นมาดังนี้



รูป 3.1 สัญลักษณ์ที่เพิ่มขึ้นมาของ อีไนแอม

ตามรูป 3.1 (a) แสดง Entity Type Class PART ซึ่งมีสมาชิกเป็น <u>PART แต่ละชนิด</u> และแสดงได้ด้วย PART# (อาจมี PART ชนิดเดียวกันจำนวน มากที่ใช้ PART# เดียวกัน)

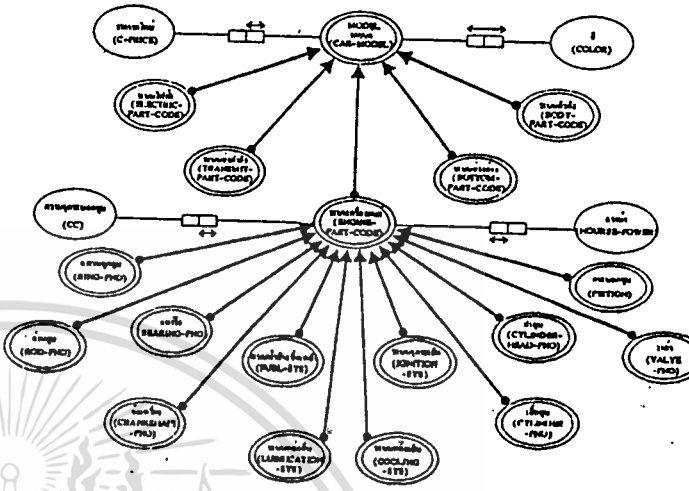
ตามรูป 3.1 (b) แสดง Entity Type PART สมาชิกเป็น <u>ตัว PART แต่ละชิ้น</u> และแสดงได้ด้วย SERIAL# บนตัว PART แต่ละชิ้นซึ่งจะไม่ซ้ำ

ตามรูป 3.1 (C) A เป็น Super-part-of ของ B และ C โดยที่ B,C เป็น Sub-part-of ของ A

ตัวอย่าง คอมเพลกซ์ ออบเจกต์ เช่น รถยนต์ ซึ่งรถยนต์ประกอบด้วยส่วนประกอบดังนี้

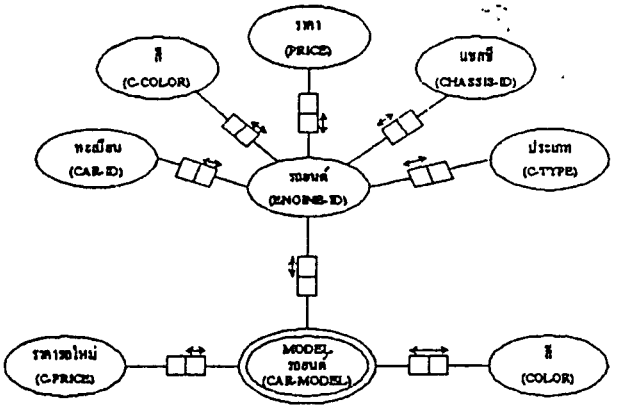
1. ระบบไฟฟ้า
2. ระบบส่งกำลัง
3. ระบบเครื่องยนต์
4. ระบบช่วงล่าง
5. ระบบตัวถัง

จะเห็นได้ว่า รถยนต์จะต้องประกอบด้วยส่วนประกอบพื้นฐานทั้ง 5 ระบบนี้ รถยนต์ถึงจะทำงานได้ ซึ่งรถยนต์มีความสัมพันธ์แบบ Aggregation สามารถใช้อีไนแอม ช่วยในการออกแบบได้ ดังรูป 3.2



รูปที่ 3.2 Aggregation Relationship Type

จากรูป 3.2 จะเห็นว่า Entity Type Class MODELรถยนต์ เป็นการระบุถึงแบบหรือรุ่นของรถยนต์ ซึ่งหมายถึงรถยนต์หลาย ๆ คัน เช่น รถยนต์ DATSUN รุ่น 120Y ก็จะหมายถึง รถยนต์ DATSUN รุ่น 120Y ทุกคัน (ไม่ได้ชี้เฉพาะลงไปถึง รถยนต์ DATSUN รุ่น120Y คันใดคันหนึ่งโดยเฉาะ) ซึ่งถ้าพิจารณาถึงรถยนต์ในเต้นท์ขายรถยนต์ใช้แล้ว จะพิจารณา ถึงรถยนต์เป็นคัน ๆ ไป โดยที่รถยนต์แต่ละคันอาจจะมีรายละเอียดที่เกี่ยวข้อง เช่น ทะเบียน, สี, ประเภท, ราคา, หมายเลขแชสซี สามารถใช้ อีไนแอม ออกแบบได้ดังรูป 3.3



รูป 3.3 อีไนแอม แสดงความแตกต่างระหว่าง Entity Type กับ Entity Type Class

จากรูป 3.2 และ 3.3 จะสามารถเห็นความแตกต่างอย่างเด่นชัดระหว่าง Entity Type Class MODELรถยนต์ ซึ่งหมายถึงรถยนต์หลาย ๆ คัน ที่มีแบบเดียวกัน มีส่วนประกอบต่าง ๆ เหมือนกัน แต่ Entity Type รถยนต์ หมายถึงรถยนต์แต่ละคันที่มี ทะเบียน, ราคา, สี, หมายเลขแชสซี, ประเภทรถ ต่างกันเป็นคัน ๆ ไป

จะเห็นได้ว่าถ้าใช้ไนแอมออกแบบระบบดังรูป 3.2 และ 3.3 จะไม่สามารถแสดงความแตกต่างระหว่าง Entity Type Class กับ Entity Type ได้อย่างชัดเจนและไม่สามารถแสดงความสัมพันธ์แบบ Aggregation ได้

4. การแปลงจาก อีไนแอม เป็น โครงสร้างข้อมูลแบบรีเลชันนัล (Transformation from ENIAM to Relational Database Schema)

1. ในกรณีของ Entity Type และ Entity Type Class

1.1 สร้าง 1 รีเลชัน (relation) สำหรับการรวมหลาย ๆ Binary Fact Type ที่มี Intra Uniqueness Constraint คลุมเพียง 1 Role การรวมจะพิจารณาทุก Birary Fact Type ที่มี Uniqueness Constraint ที่อยู่ฝั่งเดียวกันกับ Entity type ที่กำลังพิจารณา โดยนำ Label Type ของ Entity Type ที่พิจารณามาเป็น Unique Identity และนำ Label Type ของฝั่งตรงข้ามมาเป็น แอตตริบิวต์ (Attribute) ตัวอย่างเช่นรูป 2.2 โดยพิจารณา Entity Type STUDENT สามารถนำ มาสร้างรีเลชันได้ดังรูป 4.1 โดยมี STD-ID เป็น UniqueIdentity ของรีเลชัน

STD-ID	STD-NAME	DMY-BIRTH	D-NAME

รูป 4.1 รีเลชันที่ได้จากการแปลง รูปที่ 2.2 โดยใช้กฎข้อ 1.1

1.2 สร้าง 1 รีเลชันสำหรับแต่ละ n-nary Fact Type (n>2) ที่มี Intra Fact Uniqueness Constraint คลุมเพียง n-1 Role แต่ละ Entity Type ที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ ของรีเลชัน โดยที่ Intra Fact Uniqueness Constraint คลุม Role ไหน Label Type ของ Entity Type นั้น ๆ จะกลายเป็น Unique Identity ตัวอย่างเช่น รูป 2.2 พิจารณา n-nary Fact Type ที่มี n=3 สามารถ นำมาสร้าง รีเลชัน ได้ดังรูป 4.2 โดยมี A-NAME และ STD-ID เป็น Unique Identity ของ รีเลชัน

A-NAME	STD-ID	D-NAME

รูป 4.2 รีเลชันที่ได้จากการแปลง รูปที่ 2.2 โดยใช้กฎข้อ 1.2

1.3 สร้าง 1 รีเลชันสำหรับแต่ละ Intra Uniqueness Constraint คลุม n Roles (many to many) โดยที่ Label Type ของแต่ละ Entity Typeที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ และทุก แอตตริบิวต์ จะเป็น Null ไม่ได้ เพราะว่าทุก แอตตริบิวต์ รวมกันเป็น Unigue Identity ตัวอย่างเช่น รูป 2.2 พิจารณา n-nary Fact Type ที่มี n=2 คลุมทั้ง 2 Role สามารถนำมาสร้าง รีเลชัน ได้ดังรูป 4.3 โดยมี C-NAME และ STD-ID เป็น Unique Identity ของรีเลชัน

C-NAME	STD-ID

รูป 4.3 รีเลชันที่ได้จากการแปลง รูปที่ 2.2 โดยใช้กฎข้อ 1.3

1.4 ถ้า Entity Type ถูกควบคุมด้วยMandatory Constraint แล้วแอตตริบิวต์ ฝั่งตรงข้ามจะเป็น Null ไม่ได้

2. ในกรณี Entity Type หรือ Entity Type Class ที่เป็น Subtype heirarchy (Generalization)

แยกพิจารณา Supertype และ Subtype ทีละคู่ โดยการนำ Fact Type ของ Subtype ตัวที่กำลังพิจารณาไปเกาะกับ Supertype แล้วพิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา ตัวอย่างเช่น รูป 2.3 โดยพิจารณา ดังนี้

1. Entity Type เสื้อผ้า ซึ่งเป็น Supertype คู่กับ Entity Type เสื้อซึ่งเป็น Subtype สามารถนำมาสร้างเป็นรีเลชันได้ดัง รูป 4.4 โดยมี PRODUCT# เป็น Unique Identity ของรีเลชัน

PRODUCT #	P-TYPE	CLOTH-T	COST-P	COLLAR-T	SLEEVE-T

รูป 4.4 รีเลชันที่ได้จากการแปลง รูปที่ 2.3 (a) โดยใช้กฎข้อ 2

2. Entity Type เสื้อผ้าเป็น Supertype คู่กับ Entity Type กระโปรงซึ่งเป็น Subtype สามารถนำมาสร้าง รีเลชัน ได้ดังรูป 4.5 โดยมี PRODUCT# เป็น Unique Identity ของรีเลชัน

PRODUCT #	P-TYPE	CLOTH-T	COST-P	SKIRT-T	SHIR-T

รูป 4.5 รีเลชันที่ได้จากการแปลง รูปที่ 2.3 (b) โดยใช้กฎข้อ 2

3. Entity Type เสื้อผ้าซึ่งเป็น Supertype คู่กับ Entity Type กางเกง ซึ่งเป็น Subtype สามารถนำมาสร้างรีเลชัน ได้ดังรูป 4.6 โดยมี PRODUCT# เป็น Unique Identity ของรีเลชัน

PRODUCT #	P-TYPE	CLOTH-T	COST-P	TRUNKS-PANTS	B-S-LEGS

รูป 4.6 รีเลชันที่ได้จากการแปลง รูปที่ 2.3 (c) โดยใช้กฎข้อ 2

4. พิจารณา Entity Type ที่เหลือสามารถนำมาสร้าง รีเลชัน ได้ดังรูป 4.7 โดยมี PRODUCT# และ COLOR# เป็น Unique Identity

PRODUCT#	COLOR#

รูป 4.7 รีเลชันที่ได้จากการแปลง รูปที่ 2.3 (d) โดยใช้กฎข้อ 2

3. ในกรณี คอมเพลกซ์ ออบเจกต์ ที่เป็น Super-part-of และ Sub-part-of (Aggregation)

3.1 สร้าง 1 รีเลชันสำหรับแต่ละ Super-part-of โดยนำ Label Type ของ Super-part-of มาเป็น Unique Identity และsub-part-of ใดที่เป็นองค์ประกอบของ Super-part-of ให้นำ Sub-part-of นั้นมาเป็น แอตตริบิวต์ ของ รีเลชัน ตัวอย่างเช่นรูป 3.2 โดยพิจารณา Entity Type MODELรถยนต์ ซึ่งเป็น Super-part-of สามารถนำมาสร้างเป็น รีเลชัน ได้ดังรูป 4.8 โดย มี CAR-MODEL เป็น Unique Identity ของรีเลชัน

CAR-MODEL	ELECTRIC-PART-CODE	TRANSMIT-PART-CODE

ENGINE-PART-CODE	BOTTOM-PART-CODE

BODY-PART-CODE

รูป 4.8 รีเลชันที่ได้จากการแปลง รูปที่ 3.2 โดยพิจารณา Entity Type Class MODEL รถยนต์ ซึ่งเป็น Super-part-ofโดยใช้กฎข้อ 3.1

และพิจารณา Entity Type ระบบเครื่องยนต์ ซึ่งเป็น Super-part-of สามารถนำมาสร้างเป็น รีเลชัน ได้ดังรูป 4.9 โดยมี ENGINE-PART เป็น Unique Identity ของรีเลชัน

ENGINE-PART-CODE	FUEL-SYS	LUBRICATION-SYS
	COOLING-SYS	IGNITION-SYS
	RING-PNO	ROD-PNO
	BEARING-PNO	CRANKSHAFT-PNO
	CYLINDER-PNO	VALVE-PNO
	CYLINDER-HEAD-PNO	PISTON

รูป 4.9 รีเลชันที่ได้จากการแปลง รูปที่ 3.2 โดยพิจารณา Entity Type Class ระบบเครื่องยนต์ ซึ่งเป็น Super-part-of โดยใช้กฎข้อ 3.1

จากรูป 4.8 และ 4.9 ไม่สามารถบอกได้ว่า เป็นโครงสร้าง ของ Super-part-of หรือโครงสร้างของ record ธรรมดาเนื่องจาก RDB ไม่สามารถแสดงความหมาย (Semantic) ได้ดีเท่ากับ Semantic ระดับสูงเช่น อีไนแอม

3.2 แยกพิจารณาแต่ละ Super-part-of และ Sub-part-of แต่ละตัวเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา ตัวอย่างเช่น รูป 3.1 พิจารณา Entity Type MODELรถยนต์ ซึ่งเป็น Super-part-of สามารถนำมาสร้างได้ 2 รีเลชัน ดังรูป 4.10 โดยมี CAR-MODEL เป็น Unique Identity ของรีเลชันที่หนึ่ง มี CAR-MODEL และ COLOR เป็น Unique Identity ของรีเลชันที่สอง ตามลำดับ

CAR-MODEL	C-PRICE

CAR-MODEL	COLOR

รูป 4.10 รีเลชันที่ได้จากการแปลงรูป3.2 โดยพิจารณา Entity Type Class MODELรถยนต์ ซึ่งเป็น Super-part-of โดยใช้กฎข้อ 3.2

และพิจารณา Entity Type และระบบเครื่องยนต์ ซึ่งเป็น Sub-part-of ของ Entity Type MODEL รถยนต์ และในขณะเดียวกัน Entity Type ระบบเครื่องยนต์ก็เป็น Super-part-of ของ Entity Type อื่น ๆ ด้วย สามารถนำมาสร้างเป็นรีเลชันได้ ดังรูป 4.11 โดยมี ENGINE-PART-CODE เป็น Unique Identity ของรีเลชัน

ENGINE-PART-CODE	HOURSE-POWER	CC

รูป 4.11 รีเลชัน ที่ได้จากการแปลงรูป 3.2 โดยพิจารณา Entity Type ระบบเครื่องยนต์ ซึ่งเป็น Super-part-of โดยใช้กฎข้อ 3.2

5. สรุป

จะเห็นได้ว่าในความเป็นจริงนั้น วัตถุ หรือ สิ่งของต่างๆ มักจะมีความสัมพันธ์แบบAggregation ดังนั้น อีไนแอม จึงเป็นการทำให้ ไนแอม มีความสมบูรณ์มากยิ่งขึ้น คือ สามารถจะแทน (Represent) สภาพที่เป็นจริงได้มากที่สุดเท่าที่จะมากได้ และสิ่งที่น่าสังเกตอีกอย่างหนึ่งของ RDB Schema คือไม่สามารถจะบอกได้ว่า รีเลชัน ที่กำลังพิจารณาอยู่นั้น เป็น Supertype หรือ Super-part-of หรือ โครงสร้างของ record ธรรมดาจะต้อง ดูจากแบบจำลองข้อมูลเท่านั้น จึงจะรู้

เอกสารอ้างอิง

[1] Aggregation in a Behavior Oriented Objected Model*, Thorsten Hartmann,Ralf Jungclavs, Gunter Saake, Lecture Notes in Computer Science, The Netherlands, June/July 1992

[2] Concepts and Terminology for the Conceptual Schema, Griethuysen,J.J.V. et al.(ed.), ISO/TC97/SC5/WG3.

[3] A Logical Design Methodology for Relational Database Using the Extended Entity-Relationship Model, JOBY J. TEOREY, DONGQING YANG, JAMES P. FRY, Computing Serveys, Vol.18, No.2, June 1986

[4] Relational Database Design Using an Objected-Oriented Methodology Michael R. Blaha, Willinm J., James E.,Communications of the ACM, Vol.31, No.4, April 1988

บทความเรื่อง เครื่องมือออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ NIAM^{++}

เป็นผลงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์และอยู่ในระหว่างการนำเสนอ เพื่อพิจารณา
ลงตีพิมพ์ใน วารสารทางวิชาการ

เครื่องมือออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ NIAM^{++}

(An object-Oriented Database Design Tool Using NIAM^{++})

ดร.ศุภมิตร จิตตะยโศธร *

พ.ต.หญิง รัชนีพร ผดุงผล **

บทคัดย่อ

ในความเป็นจริงนั้น วัตถุสิ่งของต่าง ๆ ส่วนใหญ่จะมีโครงสร้างสลับซับซ้อน ซึ่งการออกแบบฐานข้อมูลเชิงสัมพันธ์ (RDB : Relational Database) ไม่เหมาะสมกับข้อมูลที่มีโครงสร้างสลับซับซ้อน (Complex Object) จึงได้นำความคิดของออบเจกต์ (Object) มาใช้เพื่อแก้ไขข้อบกพร่องดังกล่าว และได้ใช้ทฤษฎีของ ไนแอม (NIAM : Nijssen's Information Analysis Methodology) ซึ่งเป็นนิยามข้อมูลระดับแนวความคิดที่เหมาะสม เป็นพื้นฐานในการออกแบบฐานข้อมูลเชิงวัตถุแต่เนื่องจาก ไนแอม ไม่ได้กำหนดสัญลักษณ์ที่ใช้ แทนความสัมพันธ์แบบ Aggregation ไว้ ดังนั้นจึงได้เพิ่มเติมสัญลักษณ์บางอย่างเข้าไป เพื่อที่จะแทนสภาพความเป็นจริงให้ได้มากที่สุดเท่าที่จะมากได้ จึงเรียกว่า ไนแอม$^{++}$ (NIAM^{++}) พร้อมทั้งเสนอ แอลกริธึม (Algorithm) และเครื่องมือ (Tools) ในการแปลง ไนแอม$^{++}$ไปเป็น Relational Schema และ Object-oriented Schema

ABSTRACT

Nijssen's Information Analysis Methodology(NIAM) is selected to be the right Conceptual Schema Model for designing the relational database. NIAM does not define the sign for Aggregation relation. But most objects are Complex Object in Aggregation relation form. Some signs are added to NIAM for representing the relation in the real world. The new methodology is called Extended NIAM^{++} to Relational Schema and Obfect-Oriented Schema are also presented.

---

* ผู้ช่วยศาสตราจารย์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

** นักศึกษาปริญญาโท ภาควิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

1. บทนำ

ที่จริงแล้ว ออบเจกต์ (Object) หรือ วัตถุสิ่งของต่างๆ ส่วนใหญ่จะมีโครงสร้างที่สลับซับซ้อน เช่น รถยนต์ ประกอบด้วยส่วนต่าง ๆ ดังนี้คือ ระบบไฟฟ้า, ระบบส่งกำลัง, ระบบช่วงล่าง, ระบบตัวถังและระบบเครื่องยนต์ ซึ่งระบบเครื่องยนต์เองก็ยัง ประกอบด้วย ระบบน้ำมันเชื้อเพลิง, ระบบหล่อลื่น, ระบบหล่อเย็น และระบบจุดระเบิด ดังนั้น เครื่องยนต์ก็ เป็น คอมเพลกซ์ ออบเจกต์ จะเห็นได้ว่า คอมเพลกซ์ ออบเจกต์ สามารถที่จะประกอบด้วย คอมเพลกซ์ ออบเจกต์ อื่นได้เช่นกัน

คอมเพลกซ์ ออบเจกต์[1] หมายถึง การรวมกันของ ซับออบเจกต์ (Subobject) ที่เป็นโครงสร้างของ ออบเจกต์ แบ่งออกเป็น 2 ชนิด

1. Non-disjoint Complex Object คือ คอมเพลกซ์ ออบเจกต์ที่เกิดจากการ รวมกันของหลาย ๆ ซับออบเจกต์ ที่มาจาก คอมเพลกซ์ ออบเจกต์ อื่นด้วย

2. Disjoint Complex Object คือ คอมเพลกซ์ ออบเจกต์ ที่เกิดจากการรวมกันของหลาย ๆ ซับออบเจกต์ ที่ไม่เกี่ยวข้องกับ คอมเพลกซ์ ออบเจกต์ อื่น

2. ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ไนแอม

ไนแอม เป็น Conceptual Modal ที่มีการ Represent ในรูป Graph ซึ่งง่ายต่อการอ่านและเข้าใจในขั้นตอนการออกแบบ ไนแอมประกอบด้วย

- Entity Type คือ กลุ่มของ Abstract หรือ Real Entity ซึ่งเป็นสิ่งที่จับต้องได้หรือไม่ได้ เช่น PERSON, DEPARTMENT, COMPANY

- Label Type คือ ตัวที่ใช้บอกความแตกต่างของ Entity ภายใน Entity Type (ใช้เป็น Identifer หรือ Key)

- Elementary Fact Type คือ ตัวที่ใช้แสดงความสัมพันธ์ระหว่าง Entity type ตั้งแต่ 2 Entity type ขึ้นไป

- Subtype ใช้ในการกำหนด Hierarchies ที่ซับ-ซ้อน ของEntity type และ Subtype ที่มีการถ่ายทอด Fact Type จาก Supertype ด้วย

- Constraint เป็นเงื่อนไขที่จะทำให้แน่ชัดว่าฐานข้อมูลนั้น ถูกต้องและสมบูรณ์ตลอดเวลา (Consistency and Integrity)

2.2 ไนแอม ++

แบบจำลองข้อมูล ไนแอม มีแต่แนวความคิดของSupertype และ Subtype ซึ่งสิ่งที่เกี่ยวข้องในชีวิตประจำวันส่วนใหญ่มักเป็นคอมเพล็กซ์ ออบเจกต์ ที่มีความสัมพันธ์แบบ Super-part-of และ Sub-part-of คือ ระบบใหญ่จะประกอบด้วยระบบย่อยหลายระบบและสิ่งที่สำคัญอีกอย่างหนึ่งก็คือ ไนแอม ไม่สามารถแยก Entity Type กับ Entity Type Class ให้เห็นเด่นชัดได้ ดังนั้นจึงได้เพิ่มสัญลักษณ์ที่ใช้แทน Entity Type Classและความสัมพันธ์แบบSub-part-of เข้าไปใน ไนแอม เพื่อทำให้ไนแอม มีความสมบูรณ์มากยิ่งขึ้น จึงเรียกว่า ไนแอม++

2.3 PRO* COBOL

PRO* COBOL เป็น Embeded SQL ซึ่งเป็นภาษาโคบอลที่ทำงานร่วมกับ SQL และ มีตัวแปลภาษา (Compiler) คือ PRO*COBOL Precompiler เป็นตัวแปร SQL ให้เป็นภาษาโคบอลเสียก่อนแล้วจึง Compile และ Run ภายใต้ Compiler ภาษาโคบอล

Embeded SQL Syntax

1. SQL Command ในโปรแกรมภาษาโคบอล ต้อง
   - ขึ้นต้นด้วย EXEC SQL
   - ลงท้ายด้วย END-EXEC

เช่น EXEC SQL DELETE FROM EMP WHERE COMM = ZERO END-EXEC.

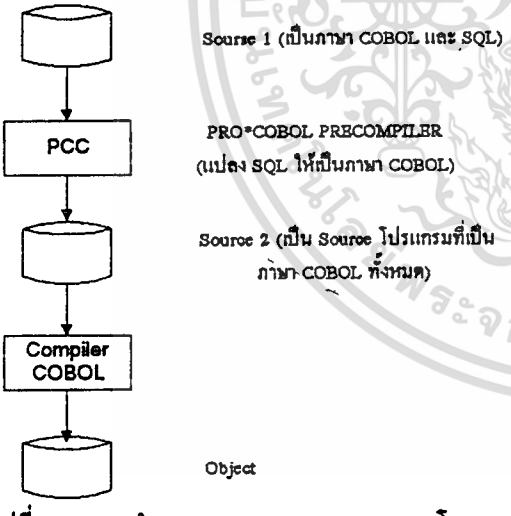
2. ตัวแปรที่จะใช้ใน SQL Statement ต้องกำหนดใน WORKING STORAGE SECTION และต้องกำหนด

ส่วนที่ใช้ติดต่อกับ Oracle Communication Area (ORACA)โดยมีรูปแบบดังนี้

```
WORKING-STORAGE SECTION.
---
---
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 EMP-REC.
    05 EMP-NO     PIC X(3).
    05 EMP-NAME   PIC X(40).
EXEC SQL END DECLARE SECTION END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

3. ใน PROCEDURE DIVISION ทุกโปรแกรมต้องมีการ Connect เข้าสู่ระบบของ Oracle โดยใช้คำสั่ง

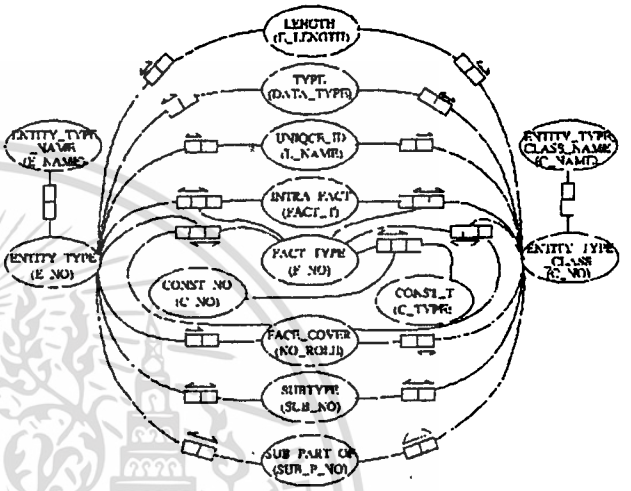
```
EXEC SQL CONNECT :USERNAME IDENTIFIED BY
                 : PASSWORD END-EXEC.
```



รูปที่ 2.1 การทำงานของ Precompiler ภาษาโคบอล

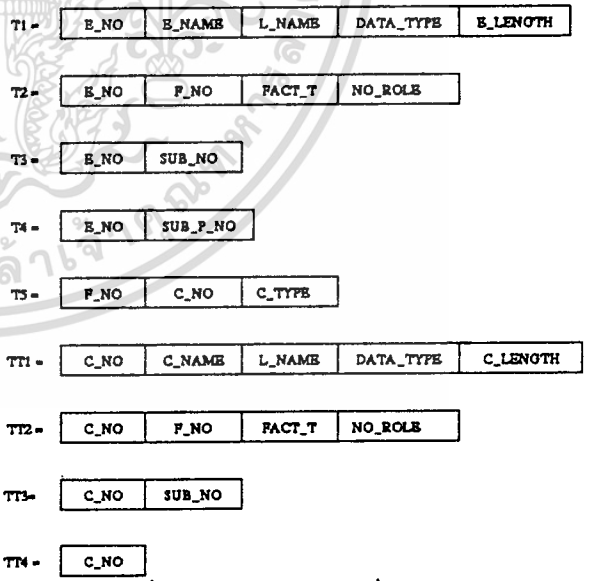
3.การออกแบบเมต้าคอนเซบชวลสกีมม่า (Meta Conceptual Sehema Designing)

จากที่กล่าวแนะนำไปแล้วว่า ไนแอม ประกอบด้วยส่วนประกอบพื้นฐาน 5 ส่วนคือ Entity Type, Label Type, Elementary Fact Type, Subtype และ Constraint ซึ่งส่วนประกอบ พื้นฐานเหล่านี้จะนำมาเป็น Entity Type ของ ไนแอม ที่จะออกแบบเป็น เมต้าคอนเซบชวล สกีมม่า สำหรับการออกแบบเมต้าคอนเซบชวลสกีมม่า จะพิจารณาเฉพาะส่วนประกอบที่สำคัญ ๆ และมีการใช้บ่อย ๆ ของไนแอมเท่านั้นซึ่งหลักเกณฑ์ในการออกแบบคอนเซบชวลสกีม่า ใช้หลักเกณฑ์ของไนแอมนั่นเอง



รูปที่ 3.1 แสดงเมต้าคอนเซบชวลสกีมม่า

จากรูปที่ 3.1 สามารถแปลงเป็นโครงสร้างฐานข้อมูลแบบรีเรชันนัล ได้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงรีเลชันที่ได้จากการแปลงเมต้าคอนเซบชวลสกีมม่า

จากรีเลชันที่ได้จากการแปลงเมต้าดอนเซบชวลสกีมม่า ดังรูปที่ 3.2 สามารถนำมาสร้างเป็น Table เพื่อเตรียมทำ User Interface ของ Tool ที่ใช้ในการแปลง ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบรีเลชันนัล

4. เครื่องมือที่ใช้แปลง ไนแอม$^{++}$ไปเป็นโครงสร้างฐานข้อมูลแบบรีเลชันนัลและเชิงวัตถุ

เมื่อทำการออกแบบโครงสร้างฐานข้อมูล เพื่อใช้เก็บข้อมูลต่าง ๆ ได้แล้ว จะต้องทำส่วนที่ใช้ในการติดต่อระหว่างผู้ใช้กับระบบ ซึ่งเครื่องมือที่ใช้แปลง ไนแอม$^{++}$ ไปเป็นโครงสร้างฐานข้อมูลแบบรีเลชันนัลหรือเชิงวัตถุ ประกอบด้วย

4.1 PRO*COBOL คือส่วนที่เป็นโปรแกรมภาษาโคบอลที่ใช้งานร่วมกับ SQL. เพื่อสร้างโครงฐานข้อมูลแบบรีเลชันและเชิงวัตถุ โดยทำงานตามแอลกอริธึมที่กำหนด

การแปลงจาก ไนแอม$^{++}$เป็นโครงสร้างข้อมูลแบบรีเลชันนัล

(Transformation from NIAM^{++} to Relational Database Schema)

1. ในกรณีของ Entity Type และ Entity Type Class

1.1 สร้าง 1 รีเลชัน (relation) สำหรับการรวมหลาย ๆ Binary Fact Type ที่มี Intra Uniqueness Constraint คลุมเพียง 1 Role การรวมจะพิจารณาทุก Birary Fact Type ที่มี Uniqueness Constraint ที่อยู่ฝั่งเดียวกันกับ Entity Type ที่กำลังพิจารณาโดยนำ Label Type ของ Entity Type ที่พิจารณามาเป็น Unique Identity และนำ Label Type ของฝั่งตรงข้ามมาเป็น แอตตริบิวต์ (Attribute)

1.2 สร้าง 1 รีเลชันสำหรับแต่ละ n-nary Fact Type (n>2) ที่มี Intra Fact Uniqueness Constraint คลุมเพียง n-1 Role แต่ละ Entity Type ที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ ของรีเลชัน โดยที่ Intra Fact Uniqueness Constraint คลุม Role ไหน Label Type ของ Entity Type นั้น ๆ จะกลายเป็น Unique Identity

1.3 สร้าง 1 รีเลชันสำหรับแต่ละ Intra Uniqueness Constraint คลุม n Roles (many to many) โดยที่ Label Type ของแต่ละ Entity Type ที่เกี่ยวข้องจะกลายเป็น แอตตริบิวต์ และทุก แอตตริบิวต์ จะเป็น Null ไม่ได้ เพราะว่าทุก แอตตริบิวต์ รวมกันเป็น Unigue Identity

1.4 ถ้า Entity Type ถูกควบคุมด้วย Mandatory Constraint แล้ว แอตตริบิวต์ ฝั่งตรงข้ามจะเป็น Null ไม่ได้

2. ในกรณี Entity Type หรือ Entity Type Class ที่เป็น Subtype heirarchy

แยกพิจารณา Supertype และ Subtype ทีละคู่ โดยการนำ Fact Type ของ Subtype ตัวที่กำลังพิจารณาไปเกาะกับ Supertype แล้วพิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

3. ในกรณี คอมโพสเซ่ อ็อบเจกต์ ที่เป็น Super-part-of และ Sub-part-of

3.1 สร้าง 1 รีเลชันสำหรับแต่ละ Super-part-of โดยนำ Label Type ของ Super-part-of มาเป็น Unique Identity และsub-part-of ใด ที่เป็นองค์ประกอบของ Super-part-of ให้นำ Sub-part-of นั้น มาเป็น แอตตริบิวต์ ของ รีเลชัน

3.2 แยกพิจารณาแต่ละ Super-part-of และ Sub-part-of แต่ละตัวเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

การแปลงจาก ไนแอม$^{++}$ เป็น โครงสร้างข้อมูลแบบเชิงวัตถุ

(Transformation from NIAM^{++} to Object-Oriented Database Schema)

1. ในกรณีของ Etity Type และ Entity Type Class

1.1 สร้าง 1 Class สำหรับ Entity ที่มี Intra Unigueness Constraint อยู่ฝั่งเดียวกันกับ Entity Type

ที่พิจารณา โดยนำ Label Type ของ Entity ฝั่งตรงข้ามมาเป็น Property ของ Class

1.2 สำหรับ Entity Type ที่สัมพันธ์กับ Fact Type ที่มี Intra Uniqueness Constraint คลุม n Role

- ถ้า Entity Type นั้นไม่มีความสัมพันธ์กับ Entity Type อื่น ไม่ต้องนำมาพิจารณา

- ถ้า Entity Type นั้น มีความสัมพันธ์กับ Entity Type อื่นให้สร้าง 1 Class สำหรับ Entity Type นั้น แล้วพิจารณาเหมือนกับเป็น

Entity Type หรือ Entity Type Classธรรมดาแล้วนำ Entity Type หรือ Entity Type Class ของฝั่งตรงข้ามมาเป็น Property ของ Class ด้วย

2. ในกรณี Entity Type หรือ Entity Type Class ที่เป็น Subtype heirarchy (Generalization)

2.1 สร้าง 1 Class สำหรับแต่ละ Supertype แล้วพิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

2.2 สร้าง 1 Class สำหรับแต่ละ Subtype ซึ่งมีการถ่ายทอด Property จาก Supertype ที่เป็น Base Class โดยใช้คำว่า Inherit และ ตามด้วยชื่อ Supertype แล้ว พิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

3. ในกรณี Complex Object ที่เป็น Super-part-of และ Sub-part-of

3.1 สร้าง 1 Class สำหรับแต่ละ Super-part-of แล้วพิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

3.2 Sub-part-of ใด ๆ ที่เป็นองค์ประกอบของ Super-part-ofให้นำ Sub- part-of นั้น ๆ มาเป็น Porperty ของ Class ด้วย

3.3 สร้าง 1 Class สำหรับแต่ละ Sub-part-of แล้วพิจารณาเหมือนกับเป็น Entity Type หรือ Entity Type Class ธรรมดา

4.2 User Interface คือส่วนที่เป็นจอภาพสำหรับติดต่อระหว่างผู้ใช้กับระบบเพื่อทำการ แปลงระบบที่ออกแบบด้วย ไนแอม++ ไปเป็นข้อมูลเก็บไว้ใน Table ต่าง ๆ และบำรุงรักษา ข้อมูล User Interface นี้เรียกว่า **N2DB**(อ่านว่า NIAM TO DATABASE)

N2DB แบ่งออกเป็น 9 ระบบ และในแต่ละระบบมี 4 Function คือ Add, Update, Delete และ List เพื่อใช้ทำการบำรุงรักษาข้อมูลในแต่ละระบบ

1. ระบบ ENTITY TYPE ใช้สำหรับแปลง Entity Type ทั้งหมดในระบบ ไปเป็นข้อมูลเก็บไว้ใน Table T1

2. ระบบ RELATION of ENTITY TYPE ใช้สำหรับแปลงความสัมพันธ์คือ Fact Type และ Entity Type ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table T2

3. ระบบ SUBTYPE of ENTITY TYPE ใช้สำหรับแปลง Entity Type ที่เป็น Subtype และ Supertype ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table T3

4. ระบบ SUB-PART-OF of ENTITY TYPE ใช้สำหรับแปลง Entity Type ที่เป็น Sub-part-of และ Super-part-of ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บ ไว้ใน Table T4

5. ระบบ CONSTRAINT ใช้สำหรับแปลง Constraint พร้อมทั้ง Fact Type และ Constraint type ที่เป็นที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูล เก็บไว้ใน Table T5

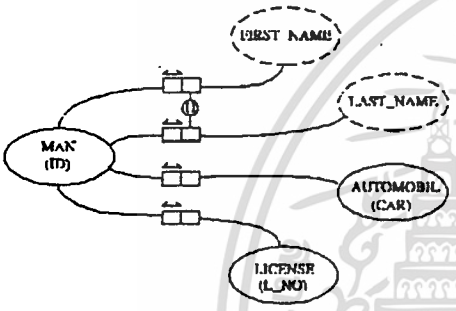
6. ระบบ ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type Class ทั้งหมดใน ระบบ ไปเป็นข้อมูลเก็บไว้ใน Table TT1

7. ระบบ RELATION of ENTITY TYPE CLASS ใช้สำหรับแปลงความสัมพันธ์คือ Fact Type และ Entity Type Class ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table TT2

8. ระบบ SUBTYPE of ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type ที่เป็น Subtype และ Supertype ที่เกี่ยวข้องทั้งหมดในระบบไปเป็นข้อมูลเก็บไว้ใน Table TT3

9. ระบบ SUB-PART-OF of ENTITY TYPE CLASS ใช้สำหรับแปลง Entity Type Class ที่เป็น Sub-part-of และ Super-part-of ที่เกี่ยวข้องกันทั้งหมดในระบบไปเป็น ข้อมูลเก็บ ไว้ใน Table TT4

ตัวอย่างการใช้งาน ระบบ N2DB



รูปที่ 4.1 ระบบที่ออกแบบโดย ไนแอม++

จากรูปที่ 4.1 นำมาแปลงเป็นข้อมูลเก็บไว้ใน Table ต่าง ๆ โดยใช้ N2DB

N2DB MAIN MENU	
1. ENTITY TYPE	: 6. ENTITY TYPE CLASS
2. RELATION of ENTITY TYPE	: 7. RELATION of : ENTITY TYPE CLASS
3. SUBTYPE of ENTITY TYPE	: 8. SUBTYPE of : ENTITY TYPE CLASS
4. SUB-PART-OF of ENTITY TYPE	: 9. SUB-PART-OF of : ENTITY TYPE CLASS
5. CONSTRAINT	:
<<< PLEASE SELECT ONE CHOICE _ >>>	
< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >	

รูปที่ 4.2 แสดง MAIN MENU ของ ระบบ N2DB

จากระบบที่ออกแบบด้วย ไนแอม++ หัวข้อใน MAIN MENU ที่เกี่ยวข้อง คือ ENTITY TYPE, RELATION of ENTITY TYPE และ CONSTRAINT

ขั้นที่ 1 เลือกข้อ 1 จะได้รูปที่ 4.3 เพื่อแปลง Entity Type ของระบบที่ศึกษา ไปเป็น ข้อมูลเก็บไว้ใน Table

N2DB ENTITY TYPE
FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)
SYSTEM NAME _____
ENTITY TYPE NO. ___
<<< PLEASE FILL IN ALL ITEMS >>>
< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

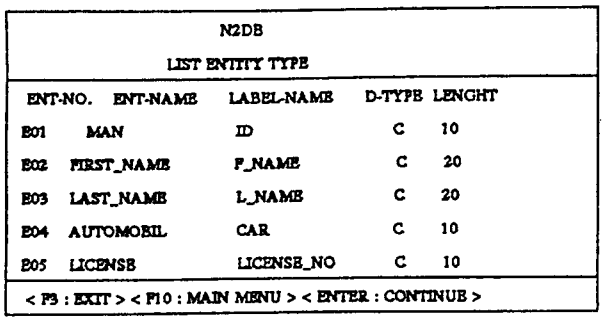
รูปที่ 4.3 แสดงจอภาพสำหรับ MAINTAIN ENTITY TYPE

จากรูปที่ 4.3 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง ENTITY TYPE NO. แล้วกด ENTER จะได้จอภาพ ดังรูปที่ 4.4

N2DB ADD ENTITY TYPE	
SYSTEM NAME _____	
ENTITY TYPE NO. ___	ENTITY TYPE NAME _____
	LABEL NAME _____
	DATA TYPE _
	ENTITY LENGTH ___
< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >	

รูปที่ 4.4 แสดงจอภาพใส่รายระเอียด ของ ENTITY TYPE

จากระบบที่ศึกษามี 3 Entity Type, 2 Label Type ใน N2DB จะรวม Entity TypeและLabel Typeไว้ด้วยกัน ดังนั้นในระบบจะมี Entity Type ทั้งหมด 5 รายการเมื่อ LIST ดู จะได้จอภาพดังรูปที่4.5



N2DB

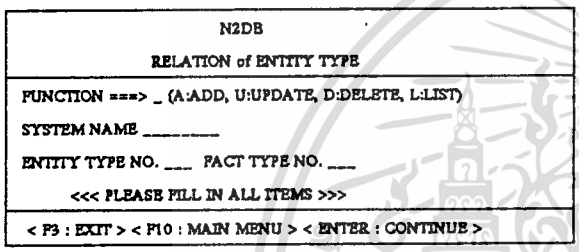
LIST ENTITY TYPE

ENT-NO.	ENT-NAME	LABEL-NAME	D-TYPE	LENGHT
E01	MAN	ID	C	10
E02	FIRST_NAME	F_NAME	C	20
E03	LAST_NAME	L_NAME	C	20
E04	AUTOMOBIL	CAR	C	10
E05	LICENSE	LICENSE_NO	C	10

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.5 จอภาพแสดง ENTITY TYPE ทั้งหมดในระบบ

ขั้นที่ 2 เลือกข้อ 2 จะรูปที่ 4.6 เพื่อแปลง ELATION ของระบบที่ศึกษาไปเป็นข้อมูล ข้อมูลเก็บไว้ใน Table



N2DB

RELATION of ENTITY TYPE

FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)

SYSTEM NAME _____

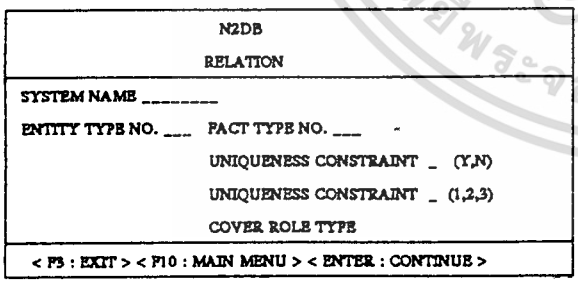
ENTITY TYPE NO. ___ FACT TYPE NO. ___

<<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.6 แสดงจอภาพสำหรับ MAINTAIN RELATION

จากรูปที่ 4.6 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง ENTITY TYPE NO. และ FACT TYPE แล้วกด ENTER จะได้จอภาพ ดังรูปที่ 4.7



N2DB

RELATION

SYSTEM NAME _____

ENTITY TYPE NO. ___ FACT TYPE NO. ___

UNIQUENESS CONSTRAINT _ (Y,N)

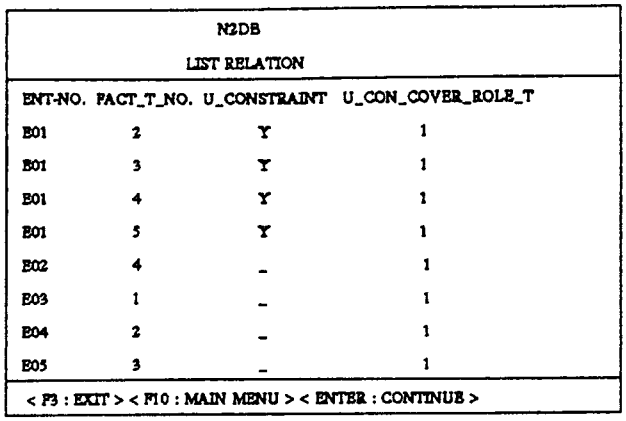
UNIQUENESS CONSTRAINT _ (1,2,3)

COVER ROLE TYPE

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.7 แสดงจอภาพใส่รายระเอียด ของ RELATION

จากระบบที่ศึกษา เมื่อใส่ครบทุก Relation แล้ว LIST ดู จะได้จอภาพดังรูปที่ 4.8



N2DB

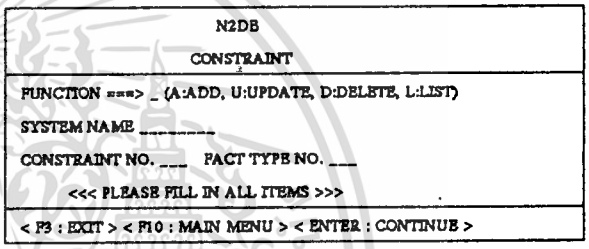
LIST RELATION

ENT-NO.	FACT_T_NO.	U_CONSTRAINT	U_CON_COVER_ROLE_T
E01	2	Y	1
E01	3	Y	1
E01	4	Y	1
E01	5	Y	1
E02	4	_	1
E03	1	_	1
E04	2	_	1
E05	3	_	1

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.8 จอภาพแสดง RELATION ทั้งหมดในระบบ

ขั้นที่ 3 เลือกข้อ 6 จะรูปที่ 4.9 เพื่อแปลง CONSTRAINT ของระบบที่ศึกษาไปเป็นข้อมูลข้อมูลเก็บไว้ใน Table



N2DB

CONSTRAINT

FUNCTION ===> _ (A:ADD, U:UPDATE, D:DELETE, L:LIST)

SYSTEM NAME _____

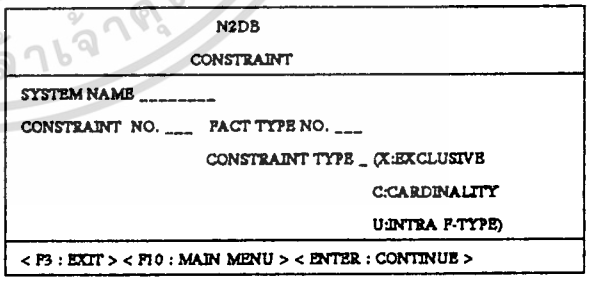
CONSTRAINT NO. ___ FACT TYPE NO. ___

<<< PLEASE FILL IN ALL ITEMS >>>

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.9 แสดงจอภาพสำหรับ MAINTAIN CONSTRAINT

จากรูปที่ 4.9 ใส่ FUNCTION ที่ต้องการและ SYSTEM NAME พร้อมทั้ง CONSTRAINT NO. และ FACT TYPE NO. แล้งกด ENTER จะได้จอภาพ ดังรูปที่ 4.10



N2DB

CONSTRAINT

SYSTEM NAME _____

CONSTRAINT NO. ___ FACT TYPE NO. ___

CONSTRAINT TYPE _ (X:EXCLUSIVE

C:CARDINALITY

U:INTRA F-TYPE)

< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >

รูปที่ 4.10 แสดงจอภาพใส่รายระเอียด ของ CONSTRAINT

จากระบบที่ศึกษา เมื่อใส่ครบทุก CONSTRAINT แล้ว LIST ดู จะได้จอภาพดังรูปที่ 4.11

N2DB LIST CONSTRAINT		
CONSTRAINT NO.	FACT TYPE NO.	CONSTRAINT TYPE
1	1	U
1	2	U
2	3	S
2	4	S
3	5	X
3	6	X
< F3 : EXIT > < F10 : MAIN MENU > < ENTER : CONTINUE >		

รูปที่ 4.11 จอภาพแสดง CONSTRAINT ทั้งหมดในระบบ

ขั้นที่ 4 เมื่อแปลงรายละเอียดจาก ไนแอม++ ไปเป็นข้อมูลใน Table ต่าง ๆ เรียบร้อยแล้วใช้ โปรแกรม PRT2RDB หรือ PRT2OODB เป็น PRO* COBOL คือ โปรแกรมภาษาโคบอล ที่ทำงานร่วมกับ SQL เพื่ออ่านข้อมูลต่าง ๆ จาก Table เพื่อแปลงเป็น RDB Schema หรือ OODB Schema ตามต้องการ จากระบบที่ศึกษา เมื่อ Run โปรแกรม PRT2RDB หรือ PRT2OODB จะได้ RDB Schema และ OODB Schema ดังต่อไปนี้

```
RDB Schema
CREATE TABLE TABLE01
        (ID             CHAR (10) NOT NULL,
         F_NAME         CHAR (20),
         L_NAME         CHAR (20),
         CAR            CHAR (10),
         LICENSE_NO     CHAR (10));
```

```
OODB Schema
STRUCT DATA01
        CHAR ID         (10);
        CHAR F_NAME     (20);
        CHAR L_NAME     (20);
        CHAR CAR        (10);
        CHAR LINENSE_NO (10);
```

5. สรุป

จะเห็นได้ว่าในความเป็นจริงนั้น วัตถุ หรือ สิ่งของต่างๆ มักจะมีความสัมพันธ์แบบ Aggregation ดังนั้น ไนแอม++ จึงเป็นการทำให้ ไนแอม มีความสมบูรณ์มากยิ่งขึ้น คือ สามารถจะแทน (Represent) สภาพที่เป็นจริงได้มากที่สุดเท่าที่จะมากได้ และสิ่งที่น่าสังเกตุ อีกอย่างหนึ่งของ RDB Schema คือไม่สามารถจะบอกได้ว่า รีเลชัน ที่กำลังพิจารณาอยู่นั้น เป็น Supertype หรือ Super-part-of หรือ โครงสร้างของ เรคอร์ด ธรรมดาจะต้อง ดูจากแบบจำลองข้อมูลเท่านั้นจึงจะรู้ แต่ OODB Schema สามารถบอกได้ว่าเป็น Supertype หรือ Super-part-of หรือ โครงสร้างของเรคอร์ดธรรมดา และสามารถใช้ N2DB เป็น เครื่องมือช่วยในการออกแบบฐานข้อมูลรีเลชันนัลและเชิงวัตถุได้อย่างมีประสิทธิภาพ

ภาคผนวก

ตัวอย่างโปรแกรม PRO* COBOL

```
ID  DIVISION.
PROGRAM-ID. EXP1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*-------------------------------------------------*
```

```
EXEC SQL BEGIN DECLARE SECTION END-
EXEC.
01 USERNAME          PIC X(8).
01 PASSW             PIC X(8).
01 EMP-REC.
     05 EMP-ID       PIC X(3).
     05 EMP-NAME     PIC X(40).

EXEC SQL END DECLARE SECTION END-EXEC.
*-------------------------------------------------------------*
EXEC SQL INCLUDE SQLCA END-EXEC.
01 DISPLAY-REC.
   05 D-ID           PIC X(3).
   05 D-NAME         PIC X(40).
PROCEDURE DIVISION.
BEGIN-PGM.
   EXEC SQL WHENEVER SQLERROR GOTO
   SQL-ERR END-EXEC.
   PERFORM LOGON.
   EXEC SQL WHENEVER NOT FOUND GOTO
   NO-EMP END-EXEC.
   PERFORM DO-SELECT.
   PERFORM DISPLAY-RECORD.
   EXEC SQL COMMIT RELEASE END-EXEC.
   STOP RUN.

LOGON.
     MOVE 'RACHANEE' TO USERNAME.
     MOVE 'RACHANEE' TO PASSW.
     EXEC SQL CONNECT :USERNAME
          IDENTIFIED BY :PASSW
          END-EXEC.
     DISPLAY '*** CONNECT TO ORALE
     ALREADY ***'

DO-SELECT.
   EXEC SQL SELECT EMP_ID, EMP_NAME
   INTO    :EMP-ID, :EMP-NAME
           FROM  EMP
           WHERE EMP-ID = 'E10'
   END-EXEC.

NO-EMP.
   DISPLAY 'NO EMO-ID'.
   GO TO END-PGM.
DISPLAY-RECORD.
   DISPLAY 'EMPLOYEE NO.  EMPLOYEE NAME'.
   DISPLAY '----------  ----------'.
   MOVE EMP-ID   TO D-ID.
   MOVE EMP-NAME TO D-NAME.
SQL-ERR.
   DISPLAY 'HAVE ERROR WITH
   CODE ===> ' SQLERRMC.
END-PGM.
   EXEC SQL WHENEVER SQLERROR CONTINUE
   END-EXEC.
   EXEC SQL ROLLBACK RELEASE
   END-EXEC.
   STOP RUN.
```

เอกสารอ้างอิง

[1] Aggregation in a Behavior Oriented Objected Model*, Thorsten Hartmann,Ralf Jungclavs, Gunter Saake, Lecture Notes in Computer Science, The Netherlands, June/July 1992

[2] Concepts and Terminology for the Conceptual Schema, Griethuysen, J.J.V. et al.(ed.), ISO/TC97/SC5/WG3.

[3] A Logical Design Methodology for Relational Database Using the Extended Entity-Relationship Model, JOBY J. TEOREY, DONGQING YANG, JAMES P. FRY, Computing Serveys, Vol.18, No.2, June 1986

[4] Relational Database Design Using an Objected-Oriented Methodology Michael R. Blaha, William J., James E.,Communications of the ACM, Vol.31, No.4, April 1988

ประวัติผู้เขียน

ชื่อผู้เขียน	พันตรีหญิง รัชนีพร ผดุงผล
วัน เดือน ปี เกิด	วันที่ 27 มิถุนายน พ.ศ. 2505
สถานที่เกิด	กรุงเทพ ฯ
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์
สถานที่สำเร็จการศึกษา	มหาวิทยาลัยรามคำแหง
ปีที่สำเร็จการศึกษา	ปีการศึกษา 2525
ผลงานทางวิชาการที่ได้รับการตีพิมพ์	การออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ ENIAM
ประสพการณ์การทำงาน	นายทหารโปรแกรม ศูนย์กรรมวิธีข้อมูล สำนักงานปลัดบัญชีทหาร กองบัญชาการทหารสูงสุด
อาชีพปัจจุบัน	รับราชการตำแหน่ง นักวิเคราะห์ระบบ แผนกวิเคราะห์และพัฒนาระบบ กองพัฒนาระบบบังคับการ กรมการสนเทศทหาร กองบัญชาการทหารสูงสุด

ประวัติผู้เขียน

ชื่อผู้เขียน	พันตรีหญิง รัชนีพร ผดุงผล
วัน เดือนปี เกิด	วันที่ 27 มิถุนายน พ.ศ. 2505
สถานที่เกิด	กรุงเทพ ฯ
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์
สถานที่สำเร็จการศึกษา	มหาวิทยาลัยรามคำแหง
ปีที่สำเร็จการศึกษา	ปีการศึกษา 2525
ผลงานทางวิชาการที่ได้รับการตีพิมพ์	การออกแบบฐานข้อมูลเชิงสัมพันธ์โดยใช้ ENIAM
ประสพการณ์การทำงาน	นายทหารโปรแกรม ศูนย์กรรมวิธีข้อมูล สำนักงานปลัดบัญชีทหาร กองบัญชาการทหารสูงสุด
อาชีพปัจจุบัน	รับราชการตำแหน่ง นักวิเคราะห์ระบบ แผนกวิเคราะห์และพัฒนาระบบ กองพัฒนาระบบงาน กรมการสนเทศทหาร กองบัญชาการทหารสูงสุด