การเก็บดาต้าโฟลว์ไดอะแกรมโดยใช้ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

STORING DATA FLOW DIAGRAM BY OBJECT-RELATIONAL DATABASE SYSTEM

โดย

นางสาวชญานี ตั้งสุขเกษมสันต์

นางสาวชุติมาภรณ์ แก้วประทุม

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การเก็บดาต้าโฟลว์ไดอะแกรมโดยใช้ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

STORING DATAFLOW DIAGRAM BY OBJECT-RELATIONAL DATABASE SYSTEM

ผู้จัดทำ

1. นางสาวชญานี ตั้งสุขเกษมสันต์ รหัสประจำตัว 38014097
2. นางสาวชุติมาภรณ์ แก้วประทุม รหัสประจำตัว 38014124

อาจารย์ที่ปรึกษา

(รศ.ดร. ศุภมิตร จิตตะยโศธร)

การเก็บดาต้าโฟลว์ไดอะแกรมโดยใช้ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

นางสาวชญานี ตั้งสุขเกษมสันต์ 38014097
นางสาวชุติมาภรณ์ แก้วประทุม 38014124
รศ.ดร. ศุภมิตร จิตตะยโสธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ในปัจจุบันเทคโนโลยีทางด้านระบบสารสนเทศ และระบบฐานข้อมูลต่างๆได้พัฒนาไปมาก ซึ่งที่นิยมกันมากคือ การพัฒนาระบบงานฐานข้อมูลเชิงสัมพันธ์ (Relational Database System) ซึ่งมีข้อดีที่ใช้งานง่าย สามารถนำไปประยุกต์เข้ากับการเก็บข้อมูลในเชิงธุรกิจได้ดี แต่ปัจจุบันมีการพัฒนาระบบฐานข้อมูลชนิดต่างๆด้วยเทคโนโลยีที่สูงขึ้น ซึ่งระบบฐานข้อมูลที่ได้รับการพัฒนาและการยอมรับในคุณภาพและมีความสะดวกในการใช้งาน คือ ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์เป็นระบบฐานข้อมูลที่มีการพัฒนาความสามารถให้สนับสนุนข้อมูลชนิดที่ซับซ้อนมากขึ้น เช่น ภาพ , วิดีโอ และเสียง เป็นต้น ซึ่งเป็นความได้เปรียบของฐานข้อมูลเชิงวัตถุสัมพันธ์ เนื่องจากระบบฐานข้อมูลเชิงสัมพันธ์ไม่สามารถที่จะจัดเก็บข้อมูลที่ซับซ้อนเหล่านี้ได้

ปริญญานิพนธ์นี้มีการเสนอการพัฒนาระบบงานโดยใช้เครื่องมือเชิงวัตถุสัมพันธ์อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ ในการสร้างแอพพลิเคชันของดาต้าโฟลว์ไดอะแกรมและนำความรู้พื้นฐานด้านแนวคิดเชิงออบเจ็กต์มาใช้ในการออกแบบระบบงาน

Storing Data Flow Diagram By Object-Relational Database System

Chayanee Tangsukkasemsant

Chutimaporn Kaewpratum

Assoc. Prof. Dr. Suphamit Chittayasothon Advisor

ABSTRACT

Presently , Information Technology and database systems were developed rapidly. The most popular approach is relational database management system . Relational database management system is easy to use and good applicate with business system. However, in recent years Object Relational database management system has been developed . It's resemble the capability of Relational database and Object-Oriented theory together.

Object Relational database is the database that support keeping more complex data types such as pictures, video and voices. This capability is its advantage over Relational database.

This thesis introduces the development of data flow diagram designing application using INFORMIX-Universal Server and applies the basic knowledge about Object-Oriented theory in designing database schema.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้ด้วยดีจากความช่วยเหลือจากอาจารย์และบุคคลหลายท่าน

รศ. ดร. ศุภมิตร จิตตะยโสธร อาจารย์ที่ปรึกษาที่ได้กรุณาให้คำปรึกษาชี้แนะแนวทางในการแก้ไขปัญหาต่าง ๆ ในการทำโครงงานและปริญญานิพนธ์

อาจารย์บัณฑิต พัสยา และอาจารย์พิทักษ์ ธรรมวารินทร์ที่คอยให้คำชี้แนะและช่วยเหลือในการให้ข้อมูลใหม่ๆ

ขอขอบคุณห้อง Hardware และเพื่อน ๆ ทุกคนที่คอยให้ความช่วยเหลือในการทำโครงงานและปริญญานิพนธ์

ขอขอบคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังสำหรับทุกสิ่งทุกอย่างที่เกิดขึ้นที่นี่

ขอขอบพระคุณสำหรับบุญคุณอันยิ่งใหญ่ที่สุดคือ คุณพ่อ คุณแม่ ของพวกเราทั้งสองที่คอยห่วงใยให้กำลังใจและสมองที่ดีกับเรา ตลอดจนให้ความช่วยเหลือในทุก ๆ เรื่องแก่พวกเราตลอดเวลาไม่เคยเปลี่ยนแปลง

ชญานี ตั้งสุขเกษมสันต์

ชุติมาภรณ์ แก้วประทุม

สารบัญ

หน้า

สารบัญตาราง

สารบัญรูป

หน้า

หน้า

บทที่ 1
บทนำ

1.1 บทนำ

การพัฒนาระบบงานสารสนเทศโดยส่วนใหญ่มีการใช้งานกันอย่างกว้างขวาง ทั้งในรูปแบบของบริษัท องค์กร ส่วนราชการและส่วนบุคคล ซึ่งส่วนใหญ่จะพัฒนาระบบงานด้วยระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database System) ซึ่งมีการนำไปใช้ในหลายรูปแบบและได้รับการยอมรับในมาตรฐานการใช้งานสูง แต่การพัฒนาระบบฐานข้อมูลเชิงสัมพันธ์มีลักษณะข้อมูลเป็นแบบตาราง ทำให้การจัดการกับข้อมูลและการมองข้อมูลมีขีดจำกัด . ซึ่งทำให้ไม่สามารถพัฒนาระบบฐานข้อมูลให้เหมาะสมกับลักษณะงานบางอย่างได้ ในปัจจุบันมีการค้นพบความก้าวหน้าทางด้านเทคโนโลยีสารสนเทศเพิ่มขึ้น ซึ่งเทคโนโลยีที่มีการพัฒนามาอย่างต่อเนื่องคือ การพัฒนาระบบงานฐานข้อมูลเชิงออบเจ็กต์ (Object Oriented Database System) โดยมีการศึกษาข้อมูลต่างๆเป็นแบบออบเจ็กต์อันหนึ่งแทนแบบตาราง ทำให้เราสามารถที่จะเข้าถึงและจัดการกับข้อมูลได้โดยตรง และพัฒนาเทคโนโลยีด้านการติดต่อกับผู้ใช้ (User Interface) รวมทั้งการใช้ชนิดข้อมูลที่ซับซ้อนขึ้น การพัฒนาระบบงานโดยการใช้เทคโนโลยีใหม่นี้จึงทำได้สะดวกและรวดเร็ว ซอฟต์แวร์ระบบงานฐานข้อมูลเชิงออบเจ็กต์นี้ก็มีหลายอย่าง เช่น Postgres , Ingres , Gupta , Informix ฯลฯ ในปัจจุบันการใช้แนวความคิดแบบออบเจ็กต์โอเรียนเต็ดช่วยทำให้การออกแบบระบบงานมีความยืดหยุ่นต่อการแก้ปัญหาและเหมาะกับระบบปัจจุบันที่ใช้การเชื่อมต่อแบบยูสเซอร์อินเตอร์เฟสโดยเน้นแนวความคิดของออบเจ็กต์ด้วย

ในการพัฒนาซอฟแวร์ทางด้านฐานข้อมูลในปัจจุบันส่วนใหญ่จะเป็นการเขียนโปรแกรมแบบโครงสร้างและใช้ระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database System) ซึ่งทำให้มีข้อจำกัดหลายประการ คือ

- ขีดจำกัดในการดูแลซอฟต์แวร์ที่เขียนโปรแกรมแบบโครงสร้างเกิดปัญหาในการแก้ไขโปรแกรม เนื่องจากความซับซ้อนและโปรแกรมเมอร์แต่ละคนมีมาตรฐานในการเขียนโปรแกรมต่างกัน ทำให้การดูแลรักษาระบบซอฟต์แวร์เกิดความยุ่งยากและทำให้ค่าใช้จ่ายในการดูแลระบบงานสูง
- การพัฒนาซอฟต์แวร์ที่ทำงานบนฐานข้อมูลเชิงสัมพันธ์ ผู้พัฒนาต้องควบคุมความถูกต้องของฐานข้อมูลเองโดยการป้อนเข้าไปในตัวโปรแกรม ทำให้เกิดความยุ่งยากเมื่อต้องนำโปรแกรมนี้ไปแก้ไขในภายหลัง
- ฐานข้อมูลเชิงสัมพันธ์เป็นฐานข้อมูลที่ได้รับการยอมรับและใช้งานอย่างกว้างขวาง แต่ระบบฐานข้อมูลชนิดนี้มีจุดด้อยหลายประการ ไม่สามารถรองรับข้อมูลที่มีความซับซ้อนได้ เช่น Complex Data (ข้อมูลจำพวกเซ็ตและลิสต์)

- ปัจจุบันเทคโนโลยีทางด้านมัลติมีเดียทางวีดีโอ อิมเมจ เสียง ข้อมูลเท็กซ์ กราฟฟิกต่างๆ ได้เจริญก้าวหน้าอย่างมาก ซึ่งระบบฐานข้อมูลเชิงสัมพันธ์นี้ไม่สามารถรองรับข้อมูลเหล่านี้ได้

จากข้อจำกัดข้างต้นนี้ ทำให้มีการนำเอาหลักการของการออกแบบซอฟต์แวร์เชิงออบเจ็กต์ (Object Oriented System Design) และฐานข้อมูลเชิงออบเจ็กต์สัมพันธ์ (Object Relational Database) มาใช้งานเพื่อพัฒนาระบบซอฟต์แวร์ที่ทำงานบนระบบฐานข้อมูลให้มีประสิทธิภาพสูงสุด หลักการเชิงออบเจ็กต์จะกำหนดสิ่งต่างๆที่อยู่ภายในขอบเขตของระบบงานเป็นออบเจ็กต์ (Object) ซึ่งในออบเจ็กต์จะประกอบด้วยข้อมูล (Data) และ เมทธอด (Method) รวมกับอยู่ภายในออบเจ็กต์ (Encapsulation) เมทธอดเปรียบเสมือนโพรซีเยอร์ (Procedure) ในโปรแกรมระบบงานแบบเดิม การเรียกใช้เมทธอดหรือการสั่งให้เกิดการกระทำตามที่กำหนดไว้ในเมทธอด คือ การส่งเมสเซจ (Message) ไปยังออบเจ็กต์นั้นๆ ดังนั้นหากมีการเปลี่ยนแปลงวิธีการกระทำ (Function) ใดๆของออบเจ็กต์จะไม่มีผลกระทบกับเมสเสจที่ส่งไปยังออบเจ็กต์นั้นและจะไม่มีผลกระทบกกับออบเจ็กต์อื่นๆด้วย ซึ่งนับว่าดีอย่างมากกับระบบงานที่มักจะมีการเปลี่ยนแปลงวิธีการปฏิบัติอยู่เสมอ โดยเมทธอดสามารถใช้อธิบายถึงพฤติการณ์ของออบเจ็กต์แต่ละออบเจ็กต์ได้ ออบเจ็กต์ที่มีพฤติการณ์แบบเดียวกันสามารถจัดรวมกลุ่มเป็นกลุ่มเดียวกันได้ เรียกว่า คลาส (Class) ในระหว่างคลาสสามารถที่จะถ่ายทอดคุณสมบัติ (Inheritance) จากคลาสหนึ่งไปยังอีกคลาสหนึ่งได้ โดยเรียกคลาสที่ถ่ายทอดคุณสมบัติว่า ซุปเปอร์คลาส (Superclass) และเรียกคลาสที่ได้รับการถ่ายทอดคุณสมบัติว่า ซับคลาส (Subclass) ซึ่งทำให้การบำรุงรักษาระบบทำได้ง่าย

1.2 วัตถุประสงค์ของโครงงาน

1. ศึกษาหลักการและการทำงานของระบบฐานข้อมูลเชิงสัมพันธ์ ระบบฐานข้อมูลเชิงออบเจ็กต์ และระบบฐานข้อมูลเชิงวัตถุสัมพันธ์
2. ศึกษาและทดลองใช้งานระบบฐานข้อมูลของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์
3. ศึกษาการสร้างและการออกแบบระบบฐานข้อมูลเพื่อนำความรู้ที่ได้ไปประยุกต์ใช้งาน
4. ศึกษาหลักการของการออกแบบระบบฐานข้อมูลด้วยแนวคิดทางออบเจ็กต์โอเรียนเต็ด
5. ออกแบบฐานข้อมูลโดยใช้แนวคิดทางออบเจ็กต์เพื่อเก็บข้อมูลร่วมกับฐานข้อมูลเชิงออบเจ็กต์สัมพันธ์ของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์
6. ออกแบบและสร้างโปรแกรมประยุกต์โดยใช้แนวคิดทางออบเจ็กต์ร่วมกับระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

1.3 ขอบเขตของโครงงาน

ในโครงงานนี้ได้นำความรู้เกี่ยวกับการออกแบบเชิงออบเจ็กต์มาใช้ในการออกแบบระบบฐานข้อมูลเพื่อสร้างแอพพลิเคชันในการสร้างดาต้าโฟลว์ไดอะแกรม (Data Flow Diagram) และใช้ระบบฐานข้อมูลของอินฟอร์มิกซ์ซึ่งเป็นฐานข้อมูลเชิงออบเจ็กต์สัมพันธ์ (Object Relational

Database System) และพัฒนาระบบงาน (Implementation) ด้วยดาต้าไดเร็กเตอร์ (Data Director) บนวิชวลเบสิก (Visual Basic) บนระบบปฏิบัติการวินโดว์ เอ็นที (WINDOWS NT)

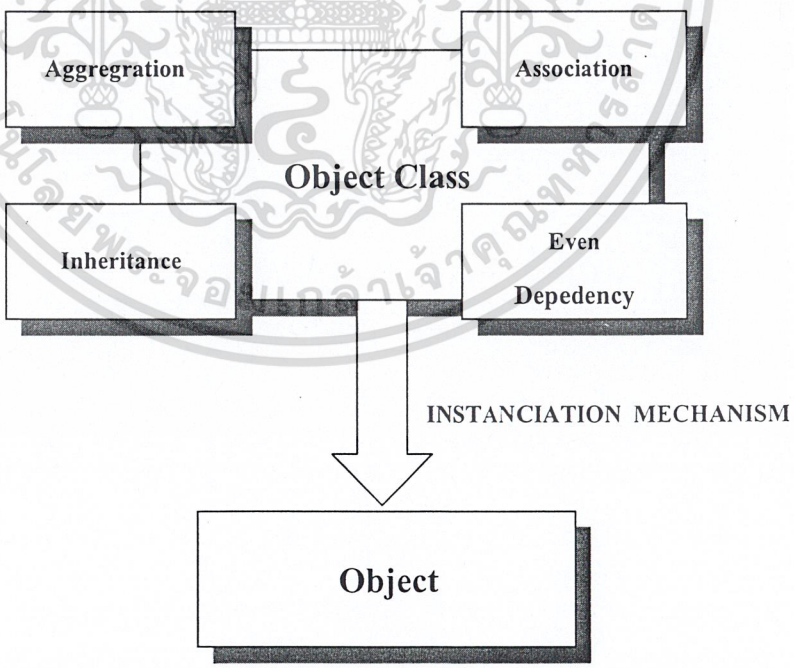
1.4 ขอบเขตของปริญญานิพนธ์

งานในโครงงานจะเริ่มจากการศึกษาทฤษฎีพื้นฐานต่างๆที่เกี่ยวข้องกับโครงงาน ซึ่งเรื่องหลักๆคือ ทฤษฎีพื้นฐานเกี่ยวกับแนวความคิดเชิงออบเจ็กต์ ระบบฐานข้อมูลแบบต่างๆ โดยเฉพาะระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ และศึกษาการใช้งานของผลิตภัณฑ์อินฟอร์มิกซ์ ยูนิเวอร์แซลเซิร์ฟเวอร์และการใช้งานของภาษาฐานข้อมูลมาตรฐานเอสคิวแอล 3 ซึ่งมีรายละเอียดดังในบทที่ 2 , 3 , 4 , 5 และ 6 จากนั้นนำเอาความรู้ที่ได้ศึกษามาออกแบบรูปแบบข้อมูลและสกีมา และทำการพัฒนาโปรแกรมเพื่อสร้างแอพพลิเคชันในการใช้งาน ซึ่งมีรายละเอียดในบทที่ 7 และ 8 ส่วนในบทที่ 9 จะเป็นการสรุปผลและปัญหาที่เกิดขึ้นในโครงงานนี้

บทที่ 2
แนวความคิดแบบออบเจ็กต์โอเรียนเต็ด

แนวความคิดที่กำลังได้รับความนิยมในวงการคอมพิวเตอร์ในปัจจุบัน คือ แนวความคิดแบบออบเจ็กต์โอเรียนเต็ด (Object Oriented Concept) ซึ่งมีความสามารถที่ดีที่เหมาะสมในการแก้ปัญหา และมีประโยชน์ต่อวงการสารสนเทศในปัจจุบัน การนำแนวความคิดนี้มาใช้ให้เกิดประโยชน์อย่างมีประสิทธิภาพนั้นต้องมีความเข้าใจถึงแนวความคิดนี้เป็นอย่างดี ออบเจ็กต์ไม่ได้จำกัดอยู่เฉพาะกับการเขียนโปรแกรมเพียงอย่างเดียว แต่ยังรวมไปถึงปรัชญาทั้งหมด เช่น การออกแบบฐานข้อมูล (Database Design) การออกแบบระบบ (System Design) การวิเคราะห์ระบบ (System Analysis) และเรื่องอื่นๆที่เกี่ยวข้อง

แนวความคิดแบบออบเจ็กต์โอเรียนเต็ดเป็นแนวความคิดที่มีคุณสมบัติแตกต่างจากแนวความคิดเดิมโดยเฉพาะแนวความคิดในการพัฒนาโปรแกรมแบบเดิมที่เป็นโปรแกรมโครงสร้าง (Structured Programming) ส่วนที่เป็นโปรแกรมและข้อมูลจะแยกจากกัน (Separates Data And Program) ในขณะที่แนวความคิดแบบออบเจ็กต์โอเรียนเต็ดจะมีออบเจ็กต์ (Object) เป็นที่รวมของข้อมูลและวิธีการ (Object Combines Data and Methods) โดยมีคลาส (Class) เป็นตัวกำหนดคุณสมบัติของออบเจ็กต์ซึ่งเป็นเครื่องมือสำคัญในการพัฒนาต้นแบบ (Prototyping Development) และสามารถนำออบเจ็กต์กลับมาใช้ใหม่ได้อีก



รูปที่ 2-1 แสดงภาพแนวความคิดของออบเจ็กต์โอเรียนเต็ด

2.1 คำศัพท์ที่เกี่ยวข้องกับแนวความคิดออบเจ็กต์โอเรียนเต็ด

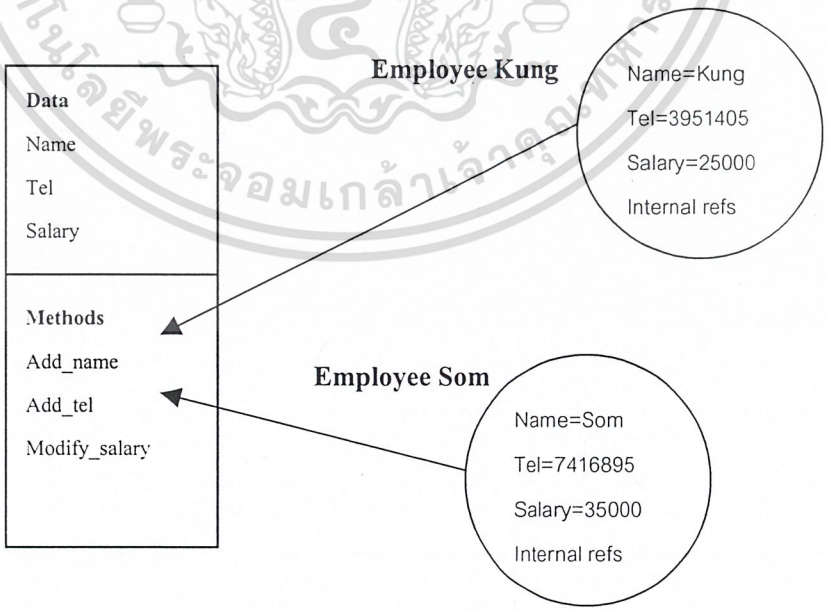
2.1.1 ออบเจ็กต์ (Object)

ออบเจ็กต์ หมายถึงสิ่งใดๆ ที่มีอยู่จริงและสามารถจับต้องได้ หรือสิ่งใดๆ ที่สามารถแสดงลักษณะใดๆได้ โดยที่ลักษณะเหล่านั้นเป็นคุณสมบัติเฉพาะของแต่ละออบเจ็กต์ที่สามารถแปรเปลี่ยนไปมาได้ตลอดการคงอยู่ของออบเจ็กต์เหล่านั้น หรือออบเจ็กต์คือ ตัวแปรคลาส (Class Variable) ที่มีลักษณะเป็นโมดูล ซึ่งประกอบด้วยตัวแปรชนิดข้อมูลต่างๆที่สัมพันธ์กันและฟังก์ชันต่างๆดูคล้ายๆกับข้อมูลชนิดโครงสร้าง สามารถนำออบเจ็กต์ต่างๆ มาประกอบเป็นโปรแกรม สามารถแทรกออบเจ็กต์เข้าไปในโปรแกรมหรือนำออกได้โดยจะมีผลกระทบต่อคำสั่งอื่นๆในโปรแกรมน้อยมากหรือไม่มีผลกระทบเลยทำให้การตรวจสอบข้อผิดพลาดในโปรแกรมกระทำได้ง่าย ตลอกจนความสะดวกสบายและรวดเร็วในการปรับปรุงและพัฒนาโปรแกรมและออบเจ็กต์ต่างๆที่สร้างขึ้นยังสามารถเก็บไว้ใช้งานในโปรแกรมอื่นๆต่อไป

ตัวแปรคลาส (Class Variable) จะนำมาใช้แทนตัวแปรโครงสร้าง เนื่องจากประสิทธิภาพสูงกว่ากันมาก คลาส (Class) จะห่อหุ้มข้อมูลและฟังก์ชันไว้รวมกันมีลักษณะเป็น Encapsulation ซึ่งสะดวกและง่ายต่อการใช้งาน สามารถป้องกันส่วนอื่นๆของโปรแกรมไม่ให้เข้าถึงตัวแปรชนิดโลคอล(Local Variable) ภายในคลาสได้ และ สามารถใช้คีย์เวิร์ดไพรเวท (Private) และพับบลิค (Public) ควบคุมการเข้าถึงสมาชิกต่างๆของคลาสจากส่วนอื่นๆของโปรแกรมได้

ออบเจ็กต์ ประกอบด้วย 2 ส่วน คือ

- ข้อมูล เป็นสถานะ (State) ของออบเจ็กต์นั้น
- เมทธอด เป็นวิธีการเข้าถึงข้อมูลและจัดการ ปรับปรุง เปลี่ยนแปลงสถานะของออบเจ็กต์นั้น



รูปที่ 2-2 แสดงความสัมพันธ์ระหว่างคลาสกับออบเจ็กต์

2.1.2 คลาส (Class)

คลาส คือ กลุ่มของออบเจ็กต์ที่แบ่งตามลักษณะเฉพาะและการใช้งาน หรือออบเจ็กต์ที่มีคุณสมบัติพื้นฐานเหมือนกัน ข้อมูลเชิงออบเจ็กต์ที่เก็บอยู่ในคลาส เรียกว่า อินสแตนท์ (Instance) ของคลาส โดยจะเก็บอยู่ในส่วนของฐานข้อมูล ดังนั้นข้อมูลต่างๆที่เป็นตัวกำหนดคลาสจะถูกเข้าถึงและถ่ายทอดผ่านทางออบเจ็กต์ของอินสแตนท์และเมทธอดของคลาส โดยคลาสจะห่อหุ้มคุณลักษณะทั้งหมดของออบเจ็กต์ต่างๆและกำหนดชนิดของข้อมูลที่บรรจุอยู่ในออบเจ็กต์ พร้อมกับกำหนดเมทธอดต่างๆ สำหรับการเข้าถึงข้อมูลไว้ด้วย

2.1.3 แอททริบิวต์ (Attributes)

แอททริบิวต์ คือค่าของข้อมูลที่เป็นส่วนบ่งบอกรายละเอียดของออบเจ็กต์ที่อยู่ในคลาส ซึ่งแอททริบิวต์จะแบ่งออกเป็น 2 ประเภท คือ

- คลาสแอททริบิวต์ (Class Attribute) คือ แอททริบิวต์ของคลาสนั้นๆ ที่ถูกเข้าถึงโดยตัวคลาสเอง อินสแตนท์และซับคลาสของตัวคลาส
- อินสแตนท์แอททริบิวต์ (Instance Attribute) คือ แอททริบิวต์ที่เข้าถึงโดยข้อมูลของคลาสที่กำหนดขึ้นเท่านั้น ซึ่งจะคล้ายกับแอททริบิวต์ในโครงสร้างอื่นๆ โดยที่แอททริบิวต์ของข้อมูลนั้นจะมีค่าตายตัวของมันเอง (Defaults Instance Attributes) หรืออาจจะมีค่าสากลที่ประยุกต์ใช้กับแต่ละข้อมูลของคลาสได้ (Share Instance Attributes)

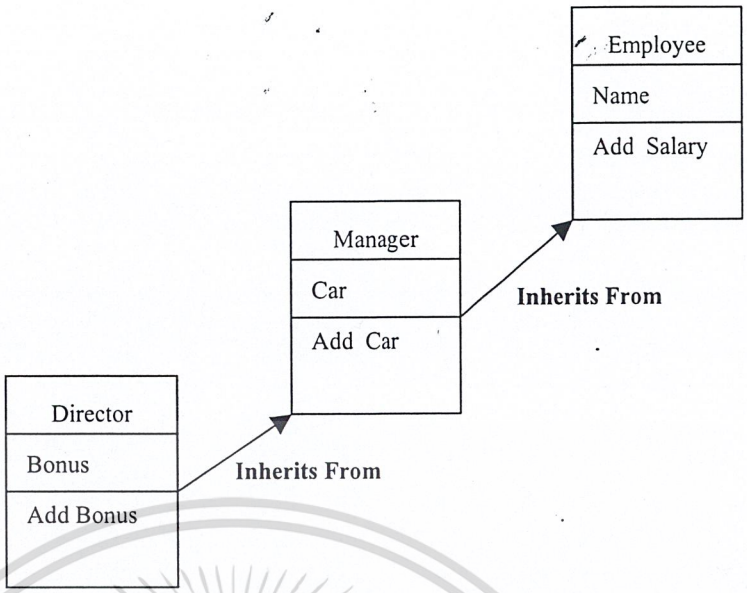
2.1.4 พฤติกรรมของคลาส (Behavior)

พฤติกรรมของคลาส คือ การที่ออบเจ็กต์สามารถทำสิ่งต่างๆ ให้กับตัวมันเองหรือมีสิ่งต่างๆ ทำให้มัน โดยจะพิจารณาที่ข้อมูลของคลาสนั้นมีการเปลี่ยนแปลงสถานะภายในของมัน หรือเมื่ออินสแตนท์ถูกขอร้องให้ทำบางสิ่งบางอย่างจากคลาส หรือจากออบเจ็กต์อื่น

2.1.5 ความสัมพันธ์ระหว่างคลาส (Relationship)

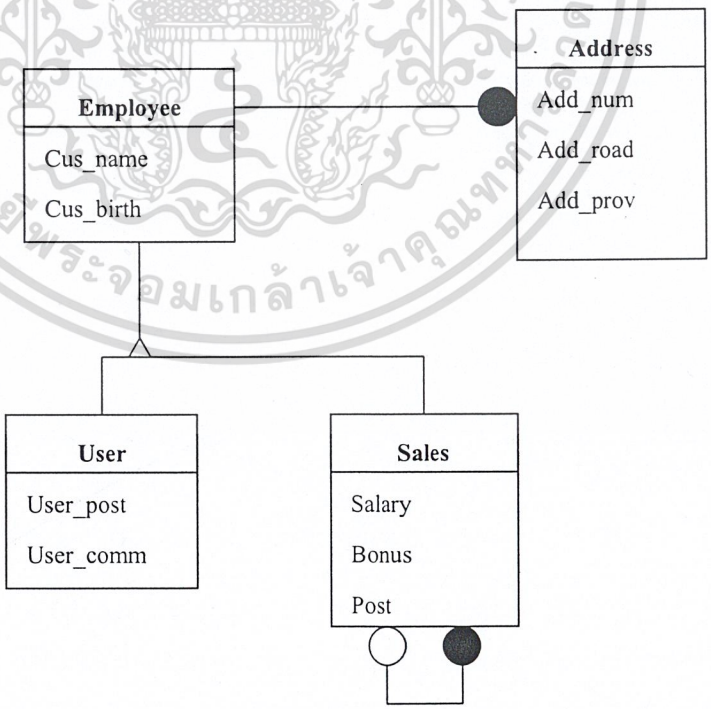
ความสัมพันธ์ระหว่างคลาส คือ สิ่งที่บ่งบอกถึงความสัมพันธ์ระหว่างคลาสกับคลาส ความสัมพันธ์ระหว่างคลาสสามารถแบ่งออกได้เป็น 2 ชนิด คือ

- ความสัมพันธ์เชิงลำดับขั้น (Hierarchical) คือ ความสัมพันธ์ที่เชื่อมระหว่างคลาสแบบเชิงลำดับขั้น



รูปที่ 2-3 แสดงความสัมพันธ์แบบ Hierarchical

- ความสัมพันธ์แบบไม่มีกฎเกณฑ์ (Non-Hierarchical) คือ ความสัมพันธ์ในแบบที่ไม่สามารถแสดงออกมาในรูปของความสัมพันธ์เชิงลำดับขั้นได้ แต่จะแสดงได้อย่างชัดเจนในรูปแบบของ Object Relationship Mapping นั่นคือ ความสัมพันธ์ในแบบของ 1-1 , 1-M , M-1 และ M-M



รูปที่ 2-4 แสดงความสัมพันธ์แบบ Non-Hierarchical

2.1.6 เมทธอด (Methods)

เมทธอด คือ การทำงาน (Operation) ที่สำคัญหรือเป็นวิธีการกระทำที่สามารถทำงานกับออบเจ็กต์ได้ เมทธอดจะถูกกำหนดขึ้นเพื่อใช้กับคลาส ให้มีหน้าที่ในการจัดการตามหน้าที่ หรือตามคลาสที่ระบุขึ้นมา แต่ละคลาสจะมีเมทธอดของตัวเอง กลุ่มของเมทธอดที่นำไปใช้กับคลาสที่กำหนดไว้จะต้องคำนึงถึงข้อจำกัด (Definition) ที่เก็บในคลาสนั้นด้วย

2.1.7 เมสเซจ (Message)

ออบเจ็กต์ต่างๆจะติดต่อกับออบเจ็กต์อื่นๆโดยผ่ายทางเมสเซจ การส่งเมสเซจ คือ การไปเรียกเมทธอดมาทำงาน แล้วส่งผลลัพธ์ไปยังผู้ส่งเมสเซจ (Sender)

2.2 คุณสมบัติของออบเจ็กต์โอเรียนเต็ด

คุณสมบัติของออบเจ็กต์มี 4 ประการ คือ

2.2.1 แอปสแตรกชัน (Abstraction)

แอปสแตรกชัน คือ การแทนความคิดที่ยุ่งยากซับซ้อนให้สั้น กระทัดรัดและรัดกุมขึ้นโดยใช้ออบเจ็กต์ ซึ่งในออบเจ็กต์จะรวมทั้งข้อมูล (Data) และการทำงาน (Process) ที่เกี่ยวข้องเขาไว้ด้วยกันเพื่อเป็นตัวแทนของออบเจ็กต์นั้น หรืออาจกล่าวได้ว่า ออบเจ็กต์ เป็นเอ็นแคปซูเลชันของแอปสแตรกชัน (Encapsulation Of Abstractions) นั่นเอง

การกำหนดแอปสแตรกชันขึ้นมาใหม่ในวิธีการแก้ปัญหาแบบออบเจ็กต์โอเรียนเต็ดจะต้องการออบเจ็กต์ใหม่ และสำหรับแต่ละออบเจ็กต์ใหม่ต้องกำหนดคุณสมบัติและเมสเซจที่จะติดต่อจัดการเกี่ยวกับอบบเจ็กต์นั้นได้

วิธีการแก้ปัญหาหนึ่งอาจประกอบด้วยหลายออบเจ็กต์ บางออบเจ็กต์ก็มีคุณสมบัติคล้ายกัน ซึ่งเรียกกลุ่มออบเจ็กต์ที่มีคุณสมบัติคล้ายกันนี้ว่า คลาสคุณสมบัติพื้นฐานของออบเจ็กต์ แต่ละออบเจ็กต์เป็นอินสแตนท์ของคลาส และการกระทำของออบเจ็กต์ถูกกำหนดโดยกลุ่มของคุณสมบัติ

2.2.2 เอ็นแคปซูเลชัน (Encapsulation)

เอ็นแคปซูเลชัน คือ การป้องกันโครงสร้างข้อมูล (Data Structure) กับวิธีการทำงานที่เกี่ยวข้องหรือความสามารถในการซ่อนข้อมูล (Information Hiding) จากระบบภายนอกได้

เอ็นแคปซูเลชัน มีลักษณะดังนี้

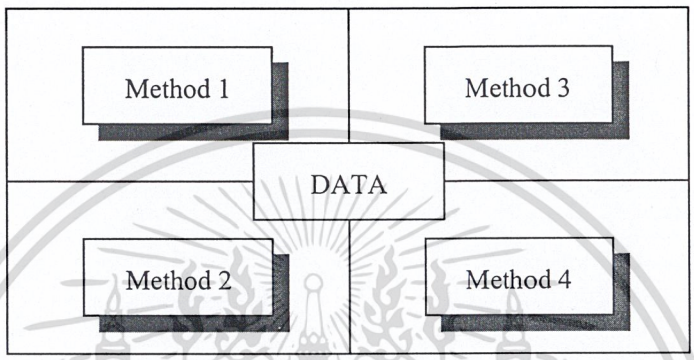
2.2.2.1 แชร์ดาต้า (Share Data) องค์ประกอบส่วนนี้จะมีค่าเหมือนกันสำหรับออบเจ็กต์ทั้งหมดที่เป็นอินสแตนท์ในคลาส อินสแตนท์ทั้งหมดในคลาสสามารถเข้าถึงและใช้ข้อมูลร่วมกันนี้ได้

2.2.2.2 ไพรเวทดาต้า (Private Data) แต่ละออบเจ็กต์จะกำหนดส่วนประกอบของแต่ละอินสแตนท์ในคลาส ซึ่งออบเจ็กต์ทั้งหมดที่เป็นอินสแตนท์ในคลาสจะมีองค์ประกอบส่วนนี้ แต่จะมีค่าต่างกัน ออบเจ็กต์หนึ่งๆจะสามารถเข้าถึงไพรเวทดาต้า (Private Data) ของตัวมันเองได้เท่านั้น

2.2.2.3. เมสเซส (Message) กลุ่มของเมสเซจ คือ สิ่งที่ออบเจ็กต์ตอบสนองต่อออบเจ็กต์ทั้งหมด ออบเจ็กต์ที่อยู่ในคลาสเดียวกันจะตอบสนองต่อกลุ่มของเมสเซจเดียวกัน

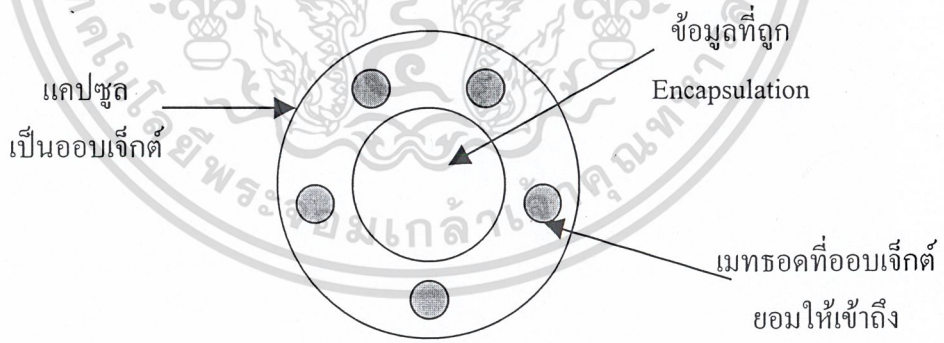
2.2.2.4. โกลบอล-แชร์ดาต้า (Global-Shared Data) เป็นข้อมูลที่สามารถถูกใช้ร่วมกันโดยอินสแตนท์ต่างคลาสได้ โกลบอล-แชร์ดาต้าจะมีค่าเหมือนกันสำหรับทุกออบเจ็กต์ที่ใช้งานร่วมกันอยู่

ข้อดีของการทำเอ็นแคปซูเลชัน คือ ทำให้เราสามารถแบ่งระดับของการเข้าถึงข้อมูลได้ด้วยเมทธอดของตัวออบเจ็กต์เอง



รูปที่ 2-5 แสดงโครงสร้างเอ็นแคปซูเลชัน

ตัวอย่างของเอ็นแคปซูเลชัน เช่น ยาเม็ดแคปซูล ซึ่งมองไม่เห็นตัวยาภายใน รู้เพียงแต่ว่ายานี้รักษาโรคอะไร ดังรูป



รูปที่ 2-6 แสดงตัวอย่างของ Encapsulation ของแคปซูล

2.2.3 โพลิมอร์ฟิซึม (Polymorphism)

โพลิมอร์ฟิซึม คือ การที่เราส่งเมสเซจที่เหมือนกันไปในออบเจ็กต์ที่ต่างกัน แต่ละออบเจ็กจะตอบสนองออกมาไม่เหมือนกันตามแต่ชนิดและหน้าที่ของออบเจ็กต์นั้น ซึ่งก็คือการสร้างฟังก์ชันที่มีหน้าที่แตกต่างกัน แต่เรียกใช้ได้ด้วยชื่อเดียวกัน เรียกว่า 'One interface , Multiple methods' เพื่อให้ใช้งานได้ง่าย หรือเรียกอีกอย่างว่า การทำโอเวอร์โหลดดิ้ง (Overloading) โดยจะแยกความแตกต่างของการใช้ชื่อที่เหมือนกันได้ด้วยจำนวนและชนิดของพารามิเตอร์

ตัวอย่างของโพลิมอร์ฟิซึม

- AB + CD ได้ผลลัพธ์ ABCD (Concat)
- 5 + 3 ได้ผลลัพธ์ 8 (Add)

ทั้งสองแบบนี้เป็นการส่งเมสเซจ '+' เข้าไปยังออบเจ็กต์ภายในคลาส String และ Integer ตามลำดับ

โพลิมอร์ฟิซึม แบ่งออกได้เป็น 2 ลักษณะ คือ

1. ช่วงการคอมไพล์โปรแกรม (Compile Time) การตรวจสอบะนิดออบเจ็กต์ของฟังก์ชันที่ถูกเรียกมาทำงาน รวมทั้งการเตรียมข้อมูลบางอย่างที่จำเป็น จะทำขณะคอมไพล์โปรแกรม การตัดสินว่าจะใช้ฟังก์ชันใดขึ้นอยู่กับคุณสมบัติของพารามิเตอร์ของฟังก์ชันนั้น
2. ช่วงการรันโปรแกรม (Run Time) การตรวจสอบจะทำขณะที่โปรแกรมกำลังทำงาน จะทำการติดสินใจเลือกฟังก์ชันใดฟังก์ชันหนึ่งที่มีชื่อซ้ำกันในแต่ละคลาส มาใช้งาน

2.2.4 อินเฮอริแทนซ์ (Inheritance)

อินเฮอริแทนซ์ คือ การสร้างคลาสขึ้นมาใหม่โดยมีการสืบทอดคุณสมบัติจากคลาสที่มีอยู่เดิม แต่จะมีข้อมูลหรือเมทธอดที่พิเศษเป็นของตัวเองเพิ่มขึ้นมาจากคลาสเดิม ซึ่งการสืบทอดคลาสนี้จะช่วยให้การพัฒนาโปรแกรมใหม่ทำได้เร็วยิ่งขึ้น

ถ้าซับคลาส B เป็นซับคลาสของคลาส A ออบเจ็กต์ที่เป็นอินสแตนท์ของซับคลาส B จะมีคุณสมบัติพิเศษกว่าอินสแตนท์ของคลาส A แต่อย่างน้อยที่สุด จะต้องมีคุณสมบัติเหมือนอินสแตนท์ของคลาส A เรียกว่า อินสแตนท์ของซับคลาส B จะถ่ายทอดคุณสมบัติมาจากคลาส A การถ่ายทอดคุณสมบัตินี้จะรวมทั้ง Private Data , Shared Data และ Message และคลาส A จะถูกเรียกว่าเป็นซุปเปอร์คลาสของคลาส B

อินเฮอริแทนซ์เป็นคุณสมบัติของออบเจ็กต์ที่เป็นอินสแตนท์ของคลาส ซึ่งจะถ่ายทอดคุณสมบัติจากซุปเปอร์คลาสของมัน ซับคลาสมีหลายระดับ ซึ่งซับคลาสที่อยู่ต่ำสุดก็จะมีคุณสมบัติของซุปเปอร์คลาสของมันทุกๆตัว

การสืบทอดคุณสมบัติต่างๆ จะเป็นดังนี้

- คลาสและอินสแตนท์ การขยายความของคลาสจะนำเสนอโดยกลุ่มของอินสแตนท์ที่มีคุณสมบัติเดียวกัน คือความสัมพันธ์ของซุปเปอร์คลาสและซับคลาสจะถูกนำเสนอออกมาโดยผ่านทางอินสแตนท์
- แอททริบิวต์ ซับคลาสจะสืบทอดแอททริบิวต์ต่างๆ จากซุปเปอร์คลาสโดยตรง ทั้งชื่อ ข้อจำกัด และความสัมพันธ์ต่างๆ ภายในแต่ละแอททริบิวต์
- ความสัมพันธ์ การสืบทอดคุณสมบัติความสัมพันธ์จะรวมในส่วนของชื่อ ชนิดของความสัมพันธ์ ข้อจำกัดต่างๆ และความสัมพันธ์ที่เกี่ยวข้องกับคลาสอื่นๆ
- โอเปอเรชัน จะถูกสืบทอดโดยซับคลาส ซึ่งซับคลาสสามารถเพิ่มโอเปอเรชันเข้าไปได้โดยอัตโนมัติ และ โอเปอเรชันของซับคลาสสามารถเป็นส่วนขยายของซุปเปอร์คลาสได้

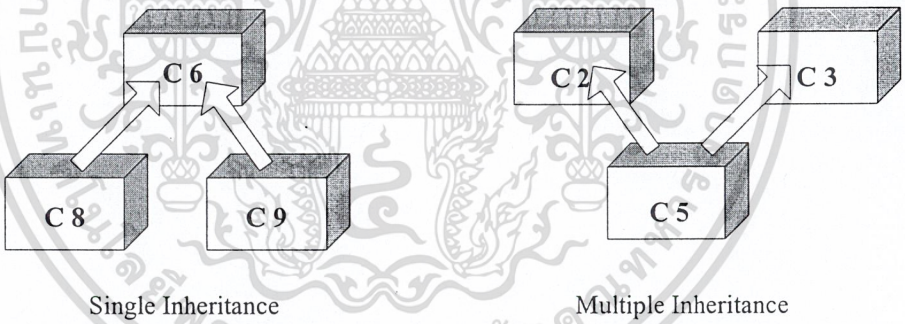
ตัวอย่างของอินเฮอริแทนซ์ เช่น คลาสของทหาร แต่ละประเภทของทหารจะถูกแยกออกเป็นชั้นๆ ที่ยอดบนสุดเป็นคลาสพื้นฐาน คือ ทหาร และชั้นถัดมาเป็นทหารบก ทหารเรือและทหารอากาศ ซึ่งได้รับการถ่ายทอดคุณสมบัติของซุปเปอร์คลาสข้างบนลงมา และมีคุณสมบัติต่างๆ เพิ่มเติมสำหรับทหารแต่ละประเภท ดังรูป

รูปที่ 2-7 แสดงคุณสมบัติอินเฮอริแทนซ์

อินเฮอริแทนซ์แบ่งได้เป็น 2 แบบ คือ

1. อินเฮอริแทนซ์แบบเดี่ยว หรือ Single Inheritance
2. อินเฮอริแทนซ์แบบรวม หรือ Multiple Inheritance

ซึ่งสามารถแสดงลักษณะของอินเฮอริแทนซ์ทั้งสองแบบได้ดังรูป



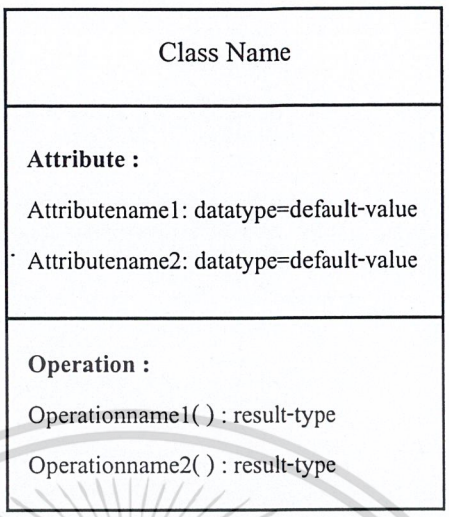
รูปที่ 2-8 แสดงคลาสแบบลำดับชั้นด้วยอินเฮอริแทนซ์แบบเดี่ยวและรวม

2.3 แบบจำลองออบเจ็กต์ (Object Modeling)

แบบจำลองออบเจ็กต์เป็นแบบจำลองที่แสดงถึงตัวออบเจ็กต์และความสัมพันธ์ระหว่างออบเจ็กต์ รวมทั้งแอททริบิวต์และเมทธอดของออบเจ็กต์คลาสด้วย แบบจำลองออบเจ็กต์ถือว่าเป็นแบบจำลองที่มีความสำคัญมาก เพราะเป็นแบบจำลองที่แสดงถึงโครงสร้างของระบบโดยใช้รูปภาพและสัญญลักษณ์ต่างๆ สื่อความหมาย ทำให้ทั้งผู้ออกแบบระบบและผู้ใช้ระบบสามารถทำความเข้าใจกับตัวระบบได้

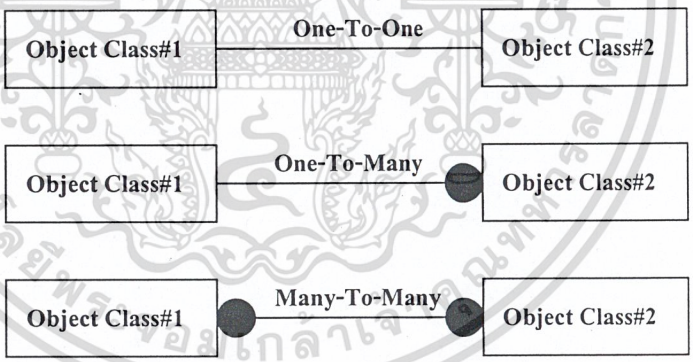
สัญลักษณ์ที่ใช้แทนออบเจ็กต์คลาส จะประกอบด้วยส่วนหลักๆ 3 ส่วน คือ

1. ส่วนแสดงชื่อคลาส
2. ส่วนแสดงรายการของแอททริบิวต์ของออบเจ็กต์คลาส เช่น ชนิดของข้อมูลและค่าเริ่มต้น
3. ส่วนเมทธอด ประอบด้วยรายละเอียดปลีกย่อย เช่น รายการและชนิดของผลลัพธ์



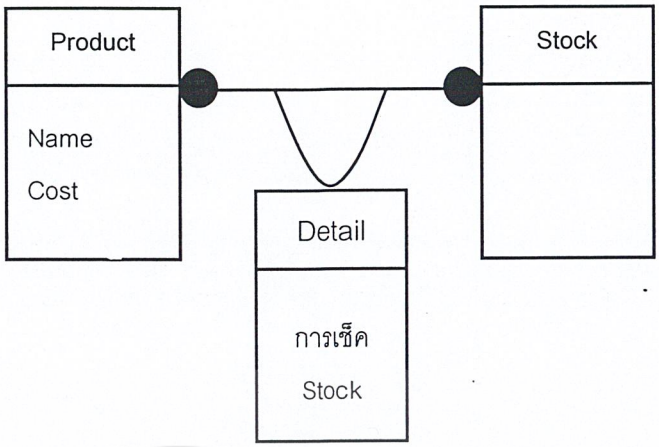
รูปที่ 2-9 แสดงสัญลักษณ์ที่ใช้แทนออบเจ็กต์คลาส

ความสัมพันธ์ระหว่างออบเจ็กต์สามารถกำหนดได้ด้วยสัญลักษณ์ซึ่งมีทั้งแบบหนึ่งต่อหนึ่ง (One-To-One) แบบหนึ่งต่อหลาย (One-To-Many) หรือ แบบหลายต่อหลาย (Many-To-Many) ดังรูป



รูปที่ 2-10 แสดงความสัมพันธ์แบบต่างๆ

การเชื่อมโยงหรือลิ้งค์ที่มีแอททริบิวต์มากกว่าหนึ่งแอททริบิวต์ สามารถแสดงความสัมพันธ์ระหว่างคลาส ในรูปแบบต่างๆ ได้ เช่น



รูปที่ 2-11 แสดงการเชื่อมโยงแอททริบิวต์มากกว่าหนึ่งแอททริบิวต์

การใส่แอททริบิวต์ให้กับลิงค์ทำให้เราสามารถเปลี่ยนความสัมพันธ์ระหว่างออบเจ็กต์คลาสได้โดยไมมีผลกระทบกับออบเจ็กต์คลาสของเดิม และช่วยลดความซ้ำซ้อนในการจัดเก็บข้อมูลลงในฐานข้อมูลด้วย

บทที่ 3
ฐานข้อมูลเชิงวัตถุสัมพันธ์

3.1 ฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ด (Object Oriented Database : OODB)

ฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ด คือ ฐานข้อมูลชนิดหนึ่งที่ถูกแสดงด้วยชนิดของข้อมูลที่กำหนดขึ้น และโอเปอร์เรชัน (Operation) ที่ถูกออกแบบมาเพื่อใช้กับชนิดของข้อมูลเหล่านั้น

ฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ด เป็นฐานข้อมูลที่สนับสนุนลักษณะของออบเจ็กต์โอเรียนเต็ด โดยใช้ออบเจ็กต์เป็นพื้นฐาน ฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ดเกิดขึ้นมาจากการรวมกันของภาษาออบเจ็กต์โอเรียนเต็ดโปรแกรมมิ่ง(Object Oriented Programming) และเทคโนโลยีทางด้านฐานข้อมูล ซึ่งก่อให้เกิดประโยชน์อย่างมาก ดังนี้

1. การที่ออบเจ็กต์โอเรียนเต็ดมีการเปลี่ยนแปลงได้ง่าย (Flexibility) จึงอำนวยความสะดวกในการออกแบบงานทางด้านฐานข้อมูลที่มีโครงสร้างสลับซับซ้อน เช่น CAD
2. งานทางด้านฐานข้อมูลที่มีโครงสร้างสลับซับซ้อน อาจเขียนขึ้นภายในการโปรแกรมภาษาซิงเกิลออบเจ็กต์โอเรียนเต็ดดาต้าเบส (Single Object Oriented Database Programming Language)ได้
3. การใช้ภาษาที่เป็นการโปแกรมแบบออบเจ็กต์โอเรียนเต็ดจะได้รับประโยชน์จากคุณลักษณะของฐานข้อมูล เช่น การคงอยู่ของข้อมูล (Persistance) , การจัดหารทรานแซคชั่น (Transaction Management) และเซตโอเรียนเต็ดโพรเซสซิ่ง (Set-Oriented Processing)

ความแตกต่างของฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ดและการโปรแกรมภาษาออบเจ็กต์ (Programming Language Object)

1. งานทางด้านฐานข้อมูลหลายๆงานต้องการความสามารถในการสร้าง (Create) และ การเข้าถึง (Access) กับออบเจ็กต์ในแบบของมัลติเพิลเวอร์ชัน (Multiple Version)
2. ข้อมูลของฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ดจะต้องคงอยู่ (Persist) ตลอดภายใต้การทำงานของโปรแกรมที่สร้างมันขึ้นมา
3. ฐานข้อมูลต้องการความสามารถ หรือ ประสิทธิภาพของการคิวรี (Query) ในเพรดิเคทเบส (Predicate -Based) บนออบเจ็กต์
4. ฐานข้อมูลที่มีความเร็วสูงๆ เช่น ฐานข้อมูลที่ใช้ในการควบคุมการจราจรทางอากาศและฐานข้อมูลที่ใช้จัดการกับการกระจาย (Distribution) ให้มีประสิทธิภาพต้องการความสามารถที่เกี่ยวข้องกับเงื่อนไข (Condition) และการกระทำ (Action) ซึ่งการดำเนินการจะถูกทำให้สมบูรณ์ เมื่อมีเงื่อนไขที่เหมาะสมกับออบเจ็กต์นั้น

ซึ่งภาษาโปรแกรมมิ่ง (Programming Language) ทั่วไปส่วนใหญ่จะไม่สนับสนุนการคงอยู่ของออบเจ็กต์หรือมัลติเพิลเวอร์ชัน (Multiple Version) และไม่อำนวยความสะดวกเกี่ยวกับเรื่องคอนสเตรนท์ (Constraint)

สิ่งที่สำคัญในฐานข้อมูลแบบออบเจ็กต์โอเรียนเต็ด คือ

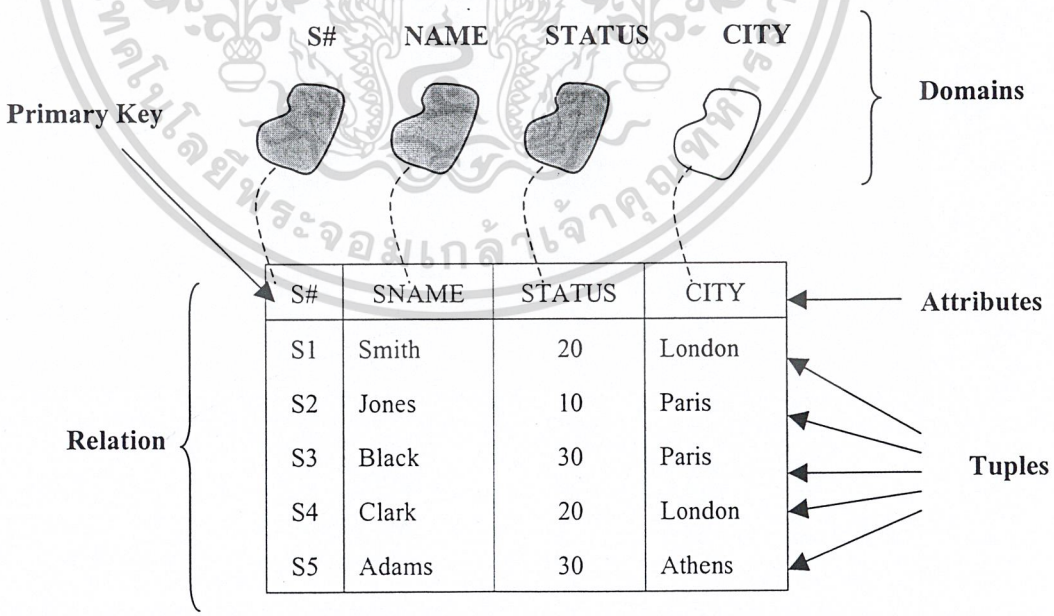
1. ออบเจ็กต์ (Object)
2. แบบข้อมูลชนิดนามธรรม (Abstract Data Type)
3. การสืบทอดคุณสมบัติ (Inheritance)

3.2 ระบบฐานข้อมูลแบบสัมพันธ์ (Relational Database System)

ระบบฐานข้อมูลแบบสัมพันธ์มีพื้นฐานมาจากเซตทางคณิตศาสตร์ และมีภาษาเอสคิวแอล (Structure Query Language) เป็นภาษาที่ใช้ในการกำหนดโครงสร้างและจัดการกับข้อมูลของฐานข้อมูลแบบสัมพันธ์นี้

การที่จะกล่าวว่าระบบฐานข้อมูลใดเป็นรีเลชันแนลโมเดล (Relation Model) นั้น ต้องพิจารณาว่าฐานข้อมูลดังกล่าวมีองค์ประกอบครบทั้ง 3 ส่วน ดังนี้

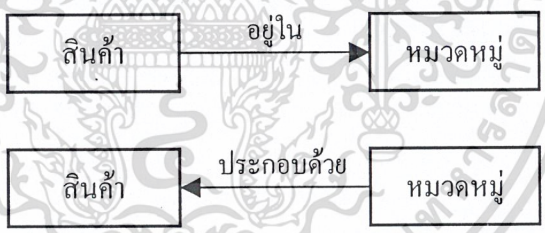
1. โครงสร้างข้อมูลเป็นไปตามนิยามคุณสมบัติของรีเลชัน (Relations)
2. ความถูกต้องของข้อมูล (Data Integrity) เป็นไปตามความถูกต้องของข้อมูลทั้งสอง
3. การจัดการข้อมูล (Data Manipulation) มีภาษาที่เป็นรีเลชันนัล คอมพลีท (Relational Complete) ในการจัดการฐานข้อมูล



รูปที่ 3-1 แสดงตัวอย่างตารางความสัมพันธ์

3.2.1 โครงสร้างข้อมูลแบบสัมพันธ์ (Relational Data Structure)

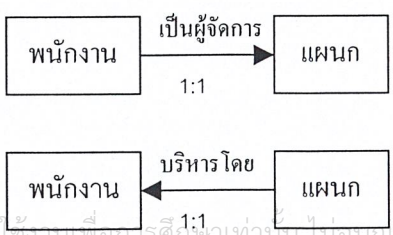
1. โดเมน (Domain) หมายถึง หน่วยข้อมูลที่เก็บอยู่ในตารางความสัมพันธ์
2. ทัพเพิล (Tuple) หมายถึง แถวของข้อมูลหนึ่งแถวของตารางความสัมพันธ์
3. เอนทิตี้ (Entity) หมายถึง ชื่อของสิ่งใดสิ่งหนึ่งอาจเกี่ยวข้องกับคน สถานที่ สิ่งของ การกระทำซึ่งต้องการจักเก็บข้อมูลไว้ เช่น เอ็นทิตี้ของสินค้า
4. แอททริบิวต์ (Attribute) หมายถึง รายละเอียดของข้อมูลในเอนทิตี้หนึ่งๆ เช่น เอนทิตี้ของสินค้าประกอบด้วยแอททริบิวต์รหัสสินค้า ชื่อ สินค้า และราคาสินค้า หรือถ้าในตารางความสัมพันธ์จะหมายถึง ชื่อคอลัมน์ของตารางความสัมพันธ์
5. ไพรมารีคีย์ (Primary Key) หมายถึง แอททริบิวต์หรือฟิล์ดที่มีลักษณะเฉพาะตัว (Unique) หรือหมายถึงแคนดิเดทคีย์ตัวหนึ่งของรีเลชันนั้นที่ได้จากการกำหนดของผู้ออกแบบฐานข้อมูล
6. ฟอเรจ์นคีย์ (Foreign Key) หมายถึง แอททริบิวต์หรือกลุ่มของแอททริบิวต์บนรีเลชัน ซึ่งเป็นไพรมารีคีย์ของรีเลชันอื่น ซึ่งความสัมพันธ์ระหว่างฟอเรจ์นคีย์ไปยังไพรมารีคีย์อื่นนี้เรียกว่า การอ้างถึง หรือ เรฟเฟอเรจ์น (Reference)
7. ความสัมพันธ์ (Relationship) หมายถึง คำที่แสดงความสัมพันธ์ระหว่าง 2 เอนทิตี้
8. ความสัมพันธ์ระหว่างเอนทิตี้ ซึ่งจะมีการระบุชื่อความสัมพันธ์ระหว่างเอนทิตี้ โดยกำหนดทิศทางความสัมพันธ์จากเอ็นทิตี้หนึ่งไปยังอีกเอ็นทิตี้หนึ่งว่ามีความสัมพันธ์กันว่าอะไร เช่น เอนทิตี้สินค้าจะอยู่ในเอนทิตี้หมวดหมู่ เอนทิตี้หมวดหมู่จะประกอบด้วยเอนทิตี้สินค้า ดังรูป



รูปที่ 3-2 แสดงความสัมพันธ์ระหว่างเอนทิตี้

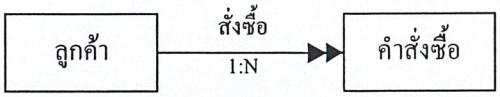
ชนิดของความสัมพันธ์ แบ่งออกเป็น

1. ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (One-To-One Relationship) คือ การแสดงความสัมพันธ์ของข้อมูลของเอนทิตี้หนึ่งว่ามีความสัมพันธ์กับข้อมูลของอีกเอนทิตี้หนึ่งอย่างมากเพียง 1 ข้อมูล



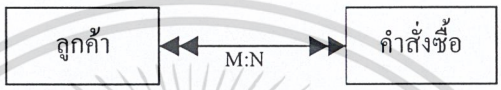
รูปที่ 3-3 แสดงความสัมพันธ์แบบ 1 : 1

2. ความสัมพันธ์แบบหนึ่งต่อกลุ่ม (One-To-Many Relationship) คือ การแสดงความสัมพันธ์ของข้อมูลของเอนทิตี้หนึ่งวามีความสัมพันธ์กับข้อมูลของอีกเอนทิตี้หนึ่งอยู่หลายข้อมูล



รูปที่ 3-4 แสดงความสัมพันธ์แบบ 1 : กลุ่ม

3. ความสัมพันธ์แบบกลุ่มต่อกลุ่ม (Many-To-Many Relationship) คือ การแสดงความสัมพันธ์ของข้อมูลของทั้งสองเอนทิตี้ในลักษณะแบบกลุ่มต่อกลุ่ม



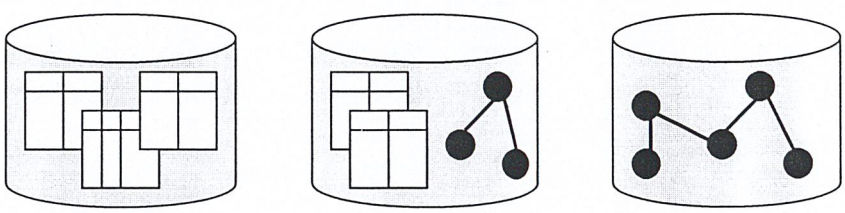
รูปที่ 3-5 แสดงความสัมพันธ์แบบ กลุ่ม : กลุ่ม

3.3 ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object Relational Database : ORDB)

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ คือ ฐานข้อมูลชนิดที่นำแนวความคิดแบบออบเจ็กต์โอเรียนเต็ด (Object-Oriented) มาผสมผสานกับระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เพื่อทำให้ระบบฐานข้อมูลเชิงสัมพันธ์มีความสามารถทางด้านออบเจ็กต์ได้ โดยระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะมองข้อมูลและทำการจัดเก็บข้อมูลเป็นแบบตารางโดยมีการจัดการฐานข้อมูลแบบเชิงสัมพันธ์ แต่การติดต่อกับผู้ใช้งานได้นำระบบของออบเจ็กต์โอเรียนเต็ดมาใช้งาน ซึ่งระบบออบเจ็กต์โอเรียนเต็ดจะมีการสนับสนุนการพัฒนาและมีการดูแลระบบฐานข้อมูลขนาดใหญ่ๆได้ดีกว่า

3.3.1 เทคโนโลยีที่ใช้ในระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

1. การเพิ่มชั้นออบเจ็กต์โอเรียนเต็ดครอบบนชั้นระบบฐานข้อมูลเชิงสัมพันธ์ที่มีอยู่
2. การออกแบบให้ระบบฐานข้อมูลเชิงสัมพันธ์รองรับงานประเภทมัลติมีเดีย และการกำหนดชนิดของข้อมูลเป็นออบเจ็กต์



รูปที่ 3-6 แสดงลักษณะของฐานข้อมูลแบบ RDBMS , ORDBMS และ ODBMS

3.3.2 คุณสมบัติของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

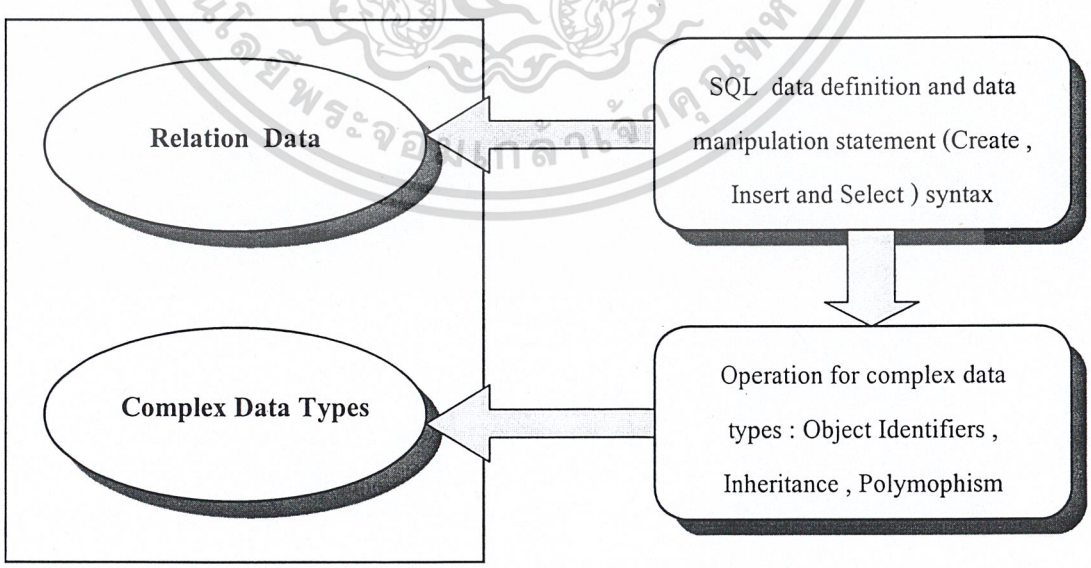
3.3.2.1) สนับสนุนตารางที่มีความซับซ้อนมากขึ้น ในโมเดลฐานข้อมูลเชิงสัมพันธ์ (Relational Data Model)นิยามไว้ว่าโดเมนของทุกๆ แอททริบิวต์ต้องอะตอมมิก หมายถึงในแอททริบิวต์หนึ่งๆสามารถเก็บข้อมูลได้เพียงค่าเดียวและข้อมูลนั้นไม่สามารถแยกย่อยต่อไปอีกได้ แต่ในระบบฐานข้อมูลเชิงสัมพันธ์อนุญาตให้ค่าของข้อมูลในแอททริบิวต์มีหลายค่าได้ ดังตารางตัวอย่างเช่น

Title	Author-list	Date (day,month,year)	Keyword-list
Salesplan	{Smith,Jones}	(1,April,89)	{profit,strategy}
Status report	{Jones,Frick}	(17,June,94)	{profit,personnel}

ตารางที่ 3-1 แสดงความสัมพันธ์ของแอททริบิวต์ที่ไม่อะตอมมิก

จากตารางจะเห็นได้ว่า ภายในคอลัมน์เก็บข้อมูลหลายค่า เนื่องจากในเอกสารหนึ่งอาจมีผู้แต่งหลายคน กำหนดคำเป็นคีย์ของเอกสารได้หลายคำ หรือการเก็บวันที่ที่แต่ง ซึ่งถึงแม้จะไม่ได้เก็บหลายๆ วันในหนึ่งคอลัมน์ของแถว แต่เมื่อพิจารณาแล้วจะพบว่า วันที่สามารถแยกเป็นฟิลด์ย่อยได้ คือ วัน เดือน ปี นั่นก็หมายความว่า วันที่ไม่อะตอมมิก

3.3.2.2) คำสั่งภาษาฐานข้อมูล (SQL) ที่มีความสมบูรณ์และมีประสิทธิภาพ มีการปรับปรุงให้คำสั่งภาษาฐานข้อมูลสามารถจัดการกับชนิดข้อมูลที่ซับซ้อนและนิยามคุณสมบัติของออบเจ็กต์ได้ เช่น การสืบทอดคุณสมบัติ(Inheritance) , โพลิมอร์ฟิซึม (Polymorphism) , การเอ็นแคปซูเลชัน (Encapsulation) ให้กับข้อมูลได้ ซึ่งภาษาฐานข้อมูลที่ถูกปรับปรุงขึ้นมาใหม่นี้ มีชื่อเรียกว่า SQL3



รูปที่ 3-7 แสดงการทำงานของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

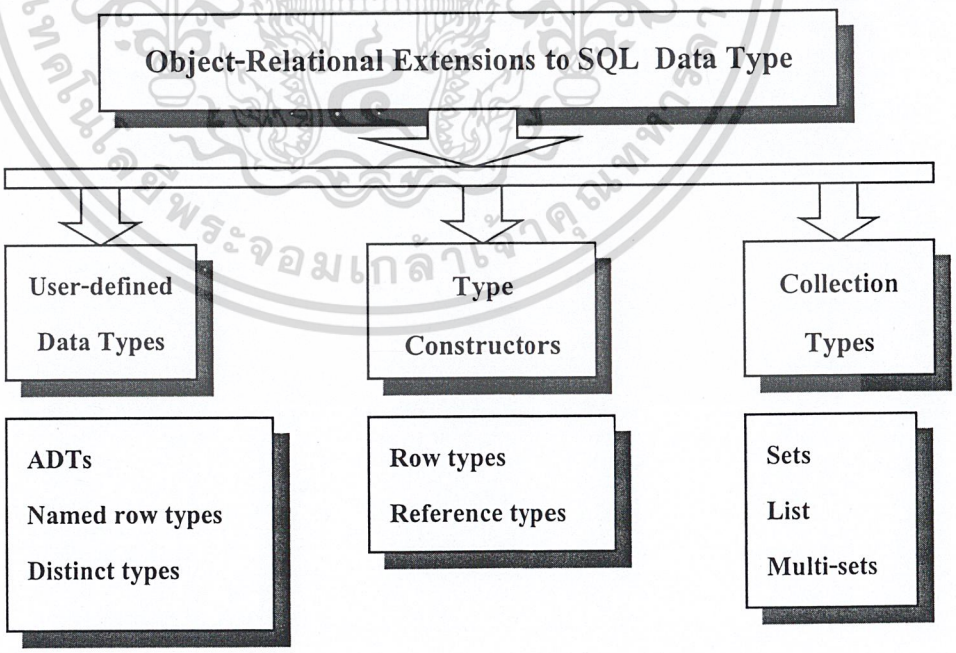
3.3.3 ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะสนับสนุนฟังก์ชันพื้นฐาน คือ

1. ความคงอยู่ของออบเจ็กต์ (Persistance of Object)
2. ความถูกต้องของข้อมูลและทรานแซกชัน (Data and Transaction Integrity)
3. ความปลอดภัย
4. การกู้ข้อมูลคืนกลับ (Recovery)
5. การใช้ข้อมูลร่วมกันจากผู้ใช้หลายๆคน (Sharing and Concurrent multi-user Access)
6. การเข้าถึงข้อมูลและการคิวรีข้อมูล (Access or Query Language)
7. การจัดการกับข้อมูลจำนวนมาก

บทที่ 4
ภาษาเรียกค้น SQL3

SQL3 เป็นภาษาฐานข้อมูลที่ยังคงสนับสนุนคุณสมบัติของ SQL 2 โดยมีการขยายเพิ่มในส่วนของอ็อบเจ็กต์เข้ามาและขยายโครงสร้างชนิดของข้อมูลให้มีชนิดใหม่และมีความสามารถสูงขึ้น รวมทั้งเพิ่มเติมขยายโครงสร้างของตารางด้วย ทำให้ SQL 3 มีการพัฒนาและมีความสามารถสูงขึ้น ซึ่งรายละเอียดของโครงสร้างข้อมูลที่มีการขยายเพิ่มเติมของ SQL 3 คือ

- ออบเจ็กต์ขนาดใหญ่ (Large Objects : LOBs) มีการยอมให้จัดเก็บข้อมูลชนิดใหม่และมีขนาดใหญ่ได้
- กำหนดข้อมูลชนิดใหม่ (User Defined Types : UDTs) สามารถสร้างชนิดของข้อมูลขึ้นมาใหม่ได้
- กำหนดฟังก์ชันใหม่ (User Defined Functions : UDFs) สามารถสร้างฟังก์ชันใหม่ได้
- ขยายขีดความสามารถของ SQL (SQL Extentions) มีการขยายความสามารถของ SQL ให้มีความสามารถเพิ่มขึ้น
- ทริกเกอร์และคอนสเตรนท์ (Triggers / Constraints) มีการตรวจสอบข้อมูลและความถูกต้องของข้อมูล



รูปที่ 4-1 แสดงโครงสร้างของข้อมูลที่มีการขยายเพิ่มเติมใน SQL 3

4.1 เมทธอดหรือโอเปอร์เรชัน (Method/Operation)

หมายถึง การจัดการกับข้อมูล ส่วนที่ถูกใช้อ้างถึงในการดึงข้อมูล เช่น SELECT , INSERT , UPDATE , DELETE รวมทั้งครอบคลุมถึงการกำหนดนอยามฟังก์ชันคุณสมบัติและรูทีน (Routines) ของแอปสแตรกดาต้าไทพ์ (ADTs : Abstact Data Types) ซึ่งทั้งสองอย่างนี้เป็นสิ่งจำเป็นสำหรับ ADTs คือ ใช้ในการกำหนดฟังก์ชันสำหรับการกำหนดไทพ์ ซึ่งการกำหนดฟังก์ชันจะเป็นตัวจัดการกับข้อมูลบน ADTs และมีการส่งกลับเพียงหนึ่งค่าตามที่มีการกำ หนดในชนิดของข้อมูลฟังก์ชันทั้งสองจะเป็นฟังก์ชันของ SQL ที่สมบูรณ์ในการกำหนดภาพแบบตามนิยามโครงร่างของ SQL หรือฟังก์ชันภายนอกที่ถูกกำหนดโดยภาษาโปรแกรมมาตรฐาน

4.2 ความสัมพันธ์ (Relationship)

หมายถึงความสัมพันธ์ของตารางโดยทั่วๆไปสามารถถูกใช้ในภาพแบบของ n-ary เหมือน SQL 2 ซึ่งการอ้างถึงและการจัดการเกี่ยวกับความถูกต้องสามารถที่จะกำหนดบนตารางได้ โดยมีส่วนที่เพิ่มเติมเข้ามาคือ ADTs ซึ่งเป็นภาพแบบของความสัมพันธ์ใน SQL 3 ซึ่งการอ้างถึงข้อมูลของออบเจ็กต์ใน SQL 3 สามารถที่จะกำหนดในแบบของ MULTISET(...) , LIST(..) และ SET(...) ได้

4.3 แอททริบิวต์ (Attributes)

คือ รายละเอียดของข้อมูลที่อยู่ในคลาสเป็นตัวบ่งบอกให้เราทราบว่า คลาสนั้นมีรายละเอียดอะไรบ้าง เช่น

```
CREATE  TABLE  Company
(  Name          Char(20),
   Position      Char(10),
   Age           Integer,
  Money  ROW (  Salary   Integer,
                Ot       Integer ) );
```

4.4 โพลิมอร์ฟิซึม (Polymorphism)

หมายถึง โปรแกรมที่มีชื่อเรียกเหมือนกันแต่มีความแตกต่างกันทางด้านการทำงานของโปรแกรม เช่น ยินยอมให้มีการสร้างฟังก์ชหรือโพรซิเยอร์ที่มีชื่อเหมือนกันขึ้นมาได้ แต่จะต้องมีการกำหนดให้พารามิเตอร์ที่ใช้ในแต่ละฟังก์ชันหรือโพรซิเยอร์นั้นมีความแตกต่างกัน หรือเรียกอีกอย่างว่า การทำโอเวอร์โหลด (Overloading) เช่น

```
CREATE  PROCEDURE  Multiply (a  integer, b  integer)
CREATE  PROCEDURE Multiply (a  integer , b  integer , c  integer)
```

4.5 เอ็นแคปซูเลชัน (Encapsulation)

ในแต่ละส่วนของ ADTs จะมีระดับของเอ็นแคปซูเลชันหลายระดับ เช่น PUBLIC , PRIVATE หรือ PROTECTED องค์ประกอบของ PUBLIC จะอยู่ในภาพของการเชื่อมต่อของ ADTs ซึ่งผู้ใช้สามารถมองเห็นได้ทุกคน ส่วนของ PRIVATE จะอยู่ในส่วนของเอ็นแคปซูเลชันทั้งหมด และจะยอมให้มีการกำหนดการใช้ในส่วนของ ADTs เท่านั้น ในส่วนของ PROTECTED จะเป็นรายละเอียดของ ADTs ซึ่งจะยอมให้มีการมองเห็นเฉพาะภายใน ADTs เองและภายในจะมีการกำหนดซับไทพ์ของ ADTs ซึ่ง SQL 3 จะส่วนที่ใช้ในการสนับสนุนเอ็นแคปซูเลชันสำหรับตาราง การขยายตารางหรือการมองตารางจะมีการกำนดการใช้เอ็นแคปซูเลชัน

4.6 ชนิดข้อมูลที่สนับสนุนคุณสมบัติของออบเจ็กต์

4.6.1 ข้อมูลชนิดโรว์ไทพ์ (Row Type)

เป็นโรว์ (Row) ที่มีการกำหนดชื่อให้กับโรว์ ซึ่งในระบบฐานข้อมูลในปัจจุบัน โรว์บนตารางเป็นสิ่งที่มีความสำคัญมากของออบเจ็กต์ เมื่อมีการออกแบบระบบฐานข้อมูลจึงมีการนำโรว์มาใช้เป็นพื้นฐานซึ่งมีหลักการออกแบบ คือ

1. การวิเคราะห์เอนทิตี้ที่เหมือนกัน
2. การออกแบบให้อยู่ในภาพโครงสร้างของ ER ไดอะแกรม
3. เปลี่ยนโครงสร้างของ ER ไปเป็นโครงสร้างของรีเลชันแนล

แบ่งข้อมูลชนิดโรว์ไทพ์ได้เป็น 2 แบบ คือ

4.6.1.1 โรว์ไทพ์ที่มีชื่อ (Named Row Type)

ในการประกาศข้อมูลชนิดโรว์ไทพ์ที่มีชื่อ มีหลักดังนี้

- ขึ้นต้นด้วยคำ CREATE ROW TYPE
- ใส่ชื่อชนิดตามหลังคีย์เวิร์ดข้างต้น
- กำหนดแอททริบิวต์ต่างๆไว้ภายในวงเล็บ

ตัวอย่างการกำหนดตารางโดยใช้คำสั่งของ SQL3

```
CREATE ROW TYPE AddressType
( Street   Char(50),
  City     Char(50) );

CREATE ROW TYPE StarType
( Name     Char(30),
  Address  AddressType );
```

```
CREATE TABLE MovieStar OF TYPE StarType ;
```

ตารางที่ได้จะประกอบด้วย 2 คอลัมน์คือชื่อและที่อยู่ของดารา โดยที่อยู่ (Address)เป็นแอททริบิวต์ที่ประกอบด้วยชื่อถนน (Street) และ ชื่อเมือง (City) ดังนี้

Name	**Address**
Carrie Fisher	Maple St. , Hollywood
Mark Hamill	Oak Rd. , Brentwood

*ตารางที่ 4-1 แสดงตาราง **MovieStar** ที่ได้จากการ CREATE TABLE*

4.6.1.2 โรว์ไทพ์ชนิดไม่มีชื่อ (Unnamed Row Type)

ในการประกาศข้อมูลชนิดโรว์ไทพ์ที่ไม่มีชื่อจะประกาศในขั้นตอนการ CREATE TABLE เช่น

```
CREATE TABLE Student
( Name          Char(20),
  Address       ROW (Number   Integer,
                     City     Char(10),
                     Zip      Char(5) ) );
```

ตารางที่ได้จะประกอบด้วย 2 คอลัมน์คือชื่อและที่อยู่ของนักเรียน โดยที่อยู่ (Address) เป็นแอททริบิวต์ที่ประกอบด้วยบ้านเลขที่ (Number) ,ชื่อเมือง (City) และรหัสไปรษณีย์ (Zip) ดังนี้

Name	**Address**
Joe Black	32 , Melbern , 54120
Jay White	24 , Brentwood , 58410

*ตารางที่ 4-2 แสดงตาราง **Student** ที่ได้จากการ CREATE TABLE*

4.6.2 การสืบทอดคุณสมบัติของออบเจ็กต์ (Inheritance)

การสืบทอดคุณสมบัติใน SQL 3 สามารถทำได้ทั้งในระดับไทพ์ (Type) และระดับตาราง (Table)

4.6.2.1 การสืบทอดคุณสมบัติในระดับไทพ์หรือระดับชนิดของข้อมูล

สมมติว่าเรากำหนดชนิดของข้อมูลเป็น

```
CREATE  ROW  TYPE  Person
        ( name              char(10) ,
          social-security   integer )
```

ถ้าเราต้องการเก็บข้อมูลพิเศษในดาต้าเบสเกี่ยวกับคนที่เป็นนักเรียนและคนที่เป็นครู เราจึงสืบทอดคุสมบัติจาก Person Type โดย

```
CREATE  ROW  TYPE  Student
        (degree             char(10),
         department         char(15))
        UNDER  Person

CREATE  ROW  TYPE  Teacher
        (salary             integer,
         department         char(15))
        UNDER  Person
```

ทั้ง Student และ Teacher จะสืบทอดคุณสมบัติจากแอททริบิวต์ของ Person คือ name และ social-security เราจะเรียกว่า Student และ Teacher เป็นซับไทพ์ (Subtype) ของ Person และ Person เป็นซุปเปอร์ไทพ์ (Supertype) ของ Student และ Teacher

ถ้าเราต้องการเก็บข้อมูลของผู้ช่วยครูซึ่งมีฐานะเป็นทั้งนักเรียนและครูในเวลาเดียวกัน ซึ่งจะมี department ที่แตกต่างกัน เราสามารถทำการสืบทอดคุณสมบัติจากหลายๆไทพ์ (Multiple Inheritance) ได้โดย

```
CREATE  ROW  TYPE  TeacherAssistant
        UNDER  Student , Teacher
```

TeacherAssistant จะสืบทอดคุณสมบัติของทุกแอททริบิวต์จาก Student และ Teacher ซึ่งอาจทำให้เกิดปัญหาเพราะมีแอททริบิวต์ name ,social-security และ department ซ้ำกันทั้งใน Student และ Teacher ซึ่งแอททริบิวต์ name และ social-security จะไม่ขัดแย้งกันเพราะเป็นแอททริบิวต์ที่สืบทอดคุณสมบัติมาจาก Person เหมือนกัน แต่แอททริบิวต์ department จะขัดแย้งกันเพราะสืบทอดคุณสมบัติมาจาก Student และ Teacher ซึ่งเป็นข้อมูลคนละตัวกัน เราสามารถแก้ไขได้โดยการเปลี่ยนชื่อแอททริบิวต์นั้นโดย

```
CREATE ROW TYPE TeacherAssistant
        UNDER Student WITH (department AS student-dept),
              Teacher WITH (department AS teacher-dept)
```

4.6.2.2 การสืบทอดคุณสมบัติในระดับตาราง

นอกจากการสืบทอดคุณสมบัติในระดับชนิดของข้อมูลที่กล่าวมาแล้วเราสามารถสร้างตารางเพื่อให้สืบทอดคุณสมบัติในระดับตารางได้โดย

```
CREATE TABLE People
        (name char(10)
         social-security integer)
```

และสร้างตาราง Student และ Teacher โดย

```
CREATE TABLE Student
        (degree char(10),
         department char(15))
    UNDER People

CREATE TABLE Teacher
        (salary integer,
         department char(15))
    UNDER People
```

ตาราง Student และ Teacher สืบทอดคุณสมบัติของทุกๆ แอททริบิวต์มาจากตาราง People ซึ่งเราเรียกว่า ตาราง Student และ Teacher เป็นซับเทเบิล (Subtable) ของตาราง People และตาราง People เป็นซุปเปอร์ (Supertable) ของตาราง Student และ Teacher เราสามารถทำการลืบทอดคุณสมบัติจากหลายๆไทพ์ (Multiple Inheritance) ในระดับตารางได้และสามารถเปลี่ยนชื่อแอททริบิวต์ที่ซ้ำกันได้เช่นเดียวกับในระดับชนิดข้อมูลโดย

```
CREATE TABLE TeacherAssistant
    UNDER Student WITH (department AS Student-dept),
          Teacher WITH (department AS Teacher-dept)
```

4.6.3 เรฟเฟอร์เรนซ์ไทพ์ (Reference Type)

เป็นสิ่งที่มีความสำคัญมากอย่างหนึ่งของระบบฐานข้อมูลในปัจจุบัน ในฐานข้อมูลแบบสัมพันธ์จะมีการอ้างอิงข้อมูลไปยังตารางต่างๆ โดยผ่านทางฟอเรจ์นคีย์ (Foreign Key) และไพร

มารีคีย์ (Primary Key) แต่ในกรณีของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะพิจารณาในภาพแบบการอ้างอิงทั้งของโรว์และออบเจ็กต์ ซึ่งแอททริบิวของโรว์ไทพ์หนึ่งสามารถอ้างถึงโรว์ไทพ์อื่นได้โดยใช้รูปแบบคำสั่ง **REF (ชื่อของ ROW Type)** ซึ่งเปรียบได้กับความสามารถของออบเจ็กต์ในการอ้างถึงออบเจ็กต์อื่นๆ ตัวอย่างเช่น

```
CREATE ROW TYPE MovieType
        ( title    CHAR(30) ,
        year       INTEGER ,
        inColor    BIT(1) ) ;

CREATE ROW TYPE AddressType
        ( street   CHAR(50) ,
        city       CHAR(20) ) ;

CREATE ROW TYPE StarsType
        ( name     CHAR(30) ,
        address    AddressType ) ;

CREATE ROW TYPE StarsInType
        ( star     REF(StarType) ,
        movie      REF(MovieType) ) ;

CREATE TABLE Movie OF TYPE MovieType ;
CREATE TABLE MovieStar OF TYPE StarType ;
CREATE TABLE StarsIn OF TYPE StarsInType ;
```

แอททริบิวต์ของ StarInType ในตัวอย่างที่ได้แสดงไว้จะอ้างถึงโรว์ไทพ์อื่นโดยแอททริบิวต์ star อ้างถึง StarType และ movie อ้างถึง MovieType ตารางซึ่งสร้างขึ้นมาจาก StarInType เป็นตารางที่มีความสัมพันธ์ระหว่างแอททริบิวต์แบบกลุ่มต่อกลุ่ม (many-to-many)

เพื่อไม่ให้เกิดความสับสนในการอ้างถึงโรว์ไทพ์อื่นๆ เราสามารถใช้คำสั่งของ SQL 3 บ่งบอกว่า แอททริบิวต์ในตารางหนึ่งอ้างถึงโรว์ไทพ์ของตารางใดบ้าง ดังนี้

```
CREATE TEBLE StarIn OF TYPE StarsInType
SCOPE FOR star IS MovieStar ,
SCOPE FOR movie IS Movie ;
```

คำสั่ง Scope For บอกให้ทราบว่า แอททริบิวต์ในตาราง StarIn คือ star อ้างถึงโรว์ไทพ์ของตาราง MovieStar และแอททริบิวต์ movie อ้างถึงโรว์ไทพ์ของตาราง Movie

4.6.4 แอปสแตรกดาต้าไทพ์ (Abstract Data Types : ADTs)

เป็นชนิดของข้อมูลใน SQL 3 ซึ่งสนับสนุนคุณสมบัติเอ็นแคปซูเลชันของออบเจ็กต์ นั่นหมายถึงว่าการเข้าถึงข้อมูลชนิดนี้จะต้องกระทำผ่านฟังก์ชันเฉพาะที่กำหนดขึ้นมากับข้อมูลชนิดนี้เท่านั้น ซึ่งแอปสแตรกดาต้าไทพ์ (ADTs) เป็นพื้นฐานของการพัฒนาชนิดของคอลัมน์ชนิดใหม่ ซึ่งเป็นข้อมูลชนิดใหม่ เช่น ข้อมูลเท็กซ์ (TEXT) , รูปภา พ (IMAGE) , ข้อมูลเกี่ยวกับระยะหรืออวกาศ (SPATIAL TYPE) และอื่นๆ ซึ่งมีการพัฒนาให้มีการใช้มัลติมีเดีย (Multimedia) ในระบบฐานข้อมูล และแอปสแตรกดาต้าไทพ์เป็นสิ่งที่ได้รับการพัฒนาในแนวทางนี้

ในการประกาศแอปสแตรกดาต้าไทพ์ ต้องใช้รูปแบบคำสั่งดังนี้

```
1)    CREATE TYPE <type name> (
2)        List of attributes and their type
3)        Optional declaration of = and < function for the type
4)        Declaration of functions (methods) for the type
5)    ) ;
```

หมายเหตุ ในบรรทัดที่สาม เป็นการประกาศฟังก์ชันเพิ่มเติมซึ่งจะถูกนำมาใช้ในการเปรียบเทียบ และในบรรทัดที่สี่ เป็นการประกาศฟังก์ชันอื่นๆ (method) ของแอปสแตรกดาต้าไทพ์

ใน SQL 3 จัดให้มีบิวท์อินฟังก์ชัน (Built-in Function) สำหรับใช้กับแอปสแตรกดาต้าไทพ์ ดังนี้

1. **ฟังก์ชันคอนสตักเตอร์ (Constructor)**

ถูกใช้เพื่อสร้างออบเจ็กต์ใหม่สำหรับข้อมูลชนิดนี้ โดยค่าเริ่มต้นของทุกแอททริบิวต์ในออบเจ็กต์มีค่าเป็น NULL รูปแบบของฟังก์ชัน คือ T( ) เมื่อให้ T เป็นชื่อของแอปสแตรกดาต้าไทพ์

2. **ฟังก์ชันอ็อบเซิร์ฟเวอร์ (Observer)**

เป็นฟังก์ชันซึ่งจะรีเทิร์นค่าของแอททริบิวต์มาเก็บไว้ในตัวแปรหนึ่ง มีรูปแบบของฟังก์ชัน คือ A(X) หรือ X.A เมื่อ A เป็นชื่อของแอททริบิวต์ และ X เป็นตัวแปรใดๆ

3. **ฟังก์ชันมิวเตเตอร์ (Mutator)**

ใช้กำหนดค่าข้อมูลค่าใหม่ให้กับแอททริบิวต์

ตัวอย่างการประกาศแอปสแตรกดาต้าไทพ์

```
1)      CREATE TYPE AddressADT (
2)              street    CHAR(50) ,
                city      CHAR(20) ,
3)              EQUALS addrEq ,
                LESS THAN addrLT
4)                      other function could be declare here
5)      );
```

ในตัวอย่างเป็นการประกาศ AddressADT ซึ่งประกอบด้วยสองแอททริบิวต์คือ street และ city โดยทั้งสองแอททริบิวต์จะถูกเข้าถึงได้เมื่อเรียกฟังก์ชันต่างๆ ที่ประกาศไว้ในแอปสแตรกดาต้าไทพ์เท่านั้น

4.6.4.1 การกำหนดเมทธอดสำหรับแอปสแตรกดาต้าไทพ์

หลังจากที่ประกาศแอททริบิวต์ต่างๆในแอปสแตรกดาต้าไทพ์เสร็จแล้ว เราสามารถกำหนดฟังก์ชันต่อท้ายได้โดยใช้รูปแบบดังนี้สำหรับการกำหนดฟังก์ชันแบบอินเทอร์นอล (Internal)

FUNCTION <name> (<arguments>) RETURNS <type> ;

และรูปแบบต่อไปนี้สำหรับการกำหนดฟังก์ชันแบบ external

DECLARE EXTERNAL <function name> <signature>
LANGUAGE <language name>

ฟังก์ชันแบบอินเทอร์นอล (Internal) หมายถึง ฟังก์ชันที่เขียนขึ้นโดยใช้ภาษา SQL 3
ฟังก์ชันแบบเอ็กซ์เทร์นอล (External) หมายถึง ฟังก์ชันที่เขียนขึ้นโดยใช้ภาษาอื่น

ตัวอย่างการกำหนดฟังก์ชันแบบอินเทอร์นอล สำหรับ AddressADT

```
1)      FUNCTION  AddressADT (:s CHAR(50) , :c CHAR(20)
                        RETURNS  AddressADT ;
2)              :a AddressADT;
                BEGIN
3)                      :a := AddressADT( ) ;
4)                      :a.street := :s ;
5)                      :a.city := :c ;
6)                      RETURN  :a ;
                END;
```

```
7)      FUNCTION addrEQ(:a1 AddressADT , :a2 AddressADT)
                    RETURNS  BOOLEAN;
8)              RETURN (:a1.street = :a2.street  AND
                    :a1.city = :a2.city);
9)      FUNCTION addrLT(:a1 AddressADT , :a2 AddressADT)
                    RETURNS  BOOLEAN;
10)             RETURN ((:a1.city<:a2.city) OR
                    (:a1.city = :a2.city) AND :a1.street<:a2.street));
11)     FUNCTION  fullAddr (:a AddressADT)  RETURNS  CHAR(82);
12)                 :z CHAR(10);
                    BEGIN
13)                     :z = findZip(:a.street , :a.city);
14)                     RETURN(:a.street || ' ' || :a.city || ' ' || :z);
                    END;
```

ช่วงบรรทัดที่หนึ่งถึงหก เป็นการสร้างฟังก์ชันคอนสตักเตอร์ใหม่สำหรับ AddressADT เพื่อให้สามารถกำหนดค่าเริ่มต้นให้กับแอททริบิวต์ street และ city ของออบเจ็กต์ที่สร้างขึ้นมาใหม่ได้ โดยมีการเรียกใช้ฟังก์ชัน AddressADT( ) ซึ่งเป็นฟังก์ชันคอนสตักเตอร์เดิมสร้างออบเจ็กต์ขึ้นมาก่อน

ช่วงบรรทัดที่เจ็ดถึงแปด และ เก้าถึงสิบ เป็นการสร้างฟังก์ชัน addrEq และ addrLT เปรียบเทียบค่าข้อมูลสองค่าที่รับเข้ามาและฟังก์ชันสุดท้ายใช้สำหรับเปลี่ยนที่อยู่ที่รับเข้ามาให้เป็นที่อยู่เต็มรูปแบบ เหมือนกันหมด

ตัวอย่างการกำหนดฟังก์ชันแบบเอ็กซ์เทอร์นอล

```
DECLARE  EXTERNAL  findZip
CHAR(50)  CHAR(20)  RETURNS  CHAR(10)
LANGUAGE C;
```

4.7 การคิวรีข้อมูล (Query)

การเข้าถึงส่วนต่างๆของโรว์ไทพ์จะใช้จุด (.) คั่น ดังตัวอย่าง

```
SELECT  MovieStar.name , MovieStar.address..street
FROM  MovieStar
WHERE  MovieStar.address..city = 'Beverly Hills' ;
```

จากตัวอย่างเป็นการคิวรีข้อมูลจากตาราง MovieStar ซึ่งสร้างมาจากโรว์ไทพ์ชื่อ StarType ซึ่งประกอบด้วยสองแอททริบิวต์ คือ name และ address โดยในแอททริบิวต์ address ยังประกอบด้วยสองแอททริบิวต์ย่อย ได้แก่ street และ city

ส่วนการเข้าถึงเรฟเฟอร์เรนซ์ไทพ์ สามารถทำได้โดยใช้เครื่องหมาย ' -> ' ดังตัวอย่าง

```
SELECT          movie -> title
FROM   StarsIn
WHERE  star -> name = 'Mel Gibson' ;
```

จากตัวอย่าง เป็นการเลือกแสดงชื่อของภาพยนตร์ทุกเรื่องที่มี Mel Gibson แสดง

4.8 คอนสเตรนท์ (Constraints)

4.8.1 คีย์คอนสเตรนท์ (Keys Constraints)

สามารถกำหนดไพรมารีย์คีย์ เมื่อทำการสร้างตารางโดยใช้วิธีใดในสองวิธีดังนี้

4.8.1.1 ใส่คำสั่ง PRIMARY KEY ตามหลังแอททริบิวของตาราง ดังเช่น

```
CREATE  TABLE  MovieStar
        ( name     CHAR(30)  PRIMARY KEY ,
        address   VARCHAR(255) ,
        gender    CHAR(1) ,
        birthdate  DATE    ) ;
```

4.8.1.2 เพิ่มคำสั่ง PRIMARY KEY (attribute name) เข้าไปภายหลัง ควรใช้วิธีนี้เมื่อไพรมารีย์คีย์ประกอบด้วยสองแอททริบิวต์ขึ้นไป เช่น

```
CREATE  TABLE  MovieStar
        (ID          INTEGER,
        name         CHAR(30) ,
        address      VARCHAR(255) ,
        gender       CHAR(1) ,
        birthdate    DATE ,
PRIMARY  KEY (ID , name) ) ;
```

อีกวิธีในการกำหนดไพรมารีย์คีย์ คือ ใช้คำสั่ง UNIQUE ใส่ลงไปในตำแหน่งเดียวกันกับ คำสั่ง PRIMARY KEY

4.8.2 ฟอเรจ์นคีย์ คอนสเตรนท์ (Foreign-Key constraints)

ฟอเรจ์นคีย์ คือ แอททริบิวต์แบบนอนคีย์ (Nonkey) ในตารางหนึ่งซึ่งเป็นไพรมารีย์คีย์ ในอีกตารางหนึ่ง (หรือตารางเดียวกัน) การกำหนดฟอเรจ์นคีย์โดยใช้ SQL 3 มีรูปแบบดังนี้

4.8.2.1 ใช้คำสั่ง REFERENCES <table> (<attribuete>)

4.8.2.2 ใช้คำสั่ง FOREIGN KEY <attribute> REFERENCES <table> (<attribute>)

ตัวอย่าง

```
CREATE TABLE Studio
        ( name    CHAR(30)  PRIMARY KEY ,
        address  VARCHAR(255) ,
        presC#   INT        REFERENCES MovieExec(cert#) );
```

จากตัวอย่าง แอททริบิวต์ presC# ถูกประกาศเป็นฟอเรจ์นคีย์ของตาราง Studio หมายถึง แอททริบิวต์ presC# อ้างถึงแอททริบิวต์ cert# ซึ่งเป็นไพรมารีย์คีย์ของตาราง MovieExec หรือเราอาจจะกำหนดในอีกรูปแบบดังนี้

```
CREATE TABLE Studio (
        Name    CHAR(30) PRIMARY KEY
        address  VARCHAR(255) ,
        presC#   INT ,
    FOREIGN KEY presC# REFERENCES MovieExec(cert#) ) ;
```

ในการรักษาความถูกต้องของข้อมูลหลังจากมีการกำหนดฟอเรจ์นคีย์ขึ้นมา ระบบจะใช้นโยบายสี่อย่างต่อไปนี้

1. หากมีความพยายามจะอินเสิร์ท (Insert) ข้อมูลลงในตาราง Studio โดยที่ค่าของ presC# ไม่ใช่ NULL และ ไม่เป็นสมาชิกของแอททริบิวต์ cert# ในตาราง MovieExec ระบบจะไม่ยอมให้มีการใส่ข้อมูลนั้นลงไป
2. หากมีการอัพเดท (Update) ข้อมูลในตาราง Studio ทำให้ค่าของ presC# เปลี่ยนไปเป็น NULL ระบบจะไม่ยอมให้มีการเปลี่ยนแปลงเกิดขึ้น เพราะค่า NULL ไม่สามารถถูกเก็บในแอททริบิวต์ cert# ได้
3. หากมีการลบ (Delete) ข้อมูลแถวใดในตาราง MovieExec โดยที่ค่าของ cert# ยังปรากฏอยู่ในตาราง Studio ระบบจะไม่ยอมให้มีการลบข้อมูลแถวนั้น
4. หากมีการอัพเดทข้อมูลแถวใดในตาราง MovieExec ซึ่งทำให้ค่าของ cert# เปลี่ยนแปลงไปจากเดิม โดยที่ค่าเดิมนั้นยังถูกใช้ในตาราง MovieExec การเปลี่ยนแปลงนั้นจะไม่เกิดขึ้น

ใน SQL3 ได้กำหนดให้สามารถใช้คำสั่ง ON DELETE และ ON UPDATE ตามด้วย SET NULL หรือ CASCADE แก้ไขนโยบายที่กล่าวไปข้างต้น ดังตัวอย่างต่อไปนี้

```
1)      CREATE TABLE Studio (
2)              name    CHAR(30) PRIMARY KEY ,
3)              address VARCHAR (255) ,
4)              presC#   INT REFERENCES MovieExec(cert#)
5)                       ON DELETE SET NULL
6)                       ON UPDATE CASCADE ) ;
```

บรรทัดที่ห้า มีการเปลี่ยนนโยบายการลบข้อมูลใหม่เป็นว่า หากมีการลบข้อมูลแถวใดในตาราง MovieExec ค่าแอททริบิวต์ PresC# ในตาราง Studio ที่สัมพันธ์กับแอททริบิวต์ cert# จะถูกกำหนดให้เป็น NULL

บรรทัดที่ 6 มีการเปลี่ยนนโยบายการลบข้อมูลใหม่เป็นว่า หากมีการอัพเดทข้อมูล cert# ในแถวใดๆ ของตาราง MovieExec แล้วข้อมูลทุกๆแถว ในตาราง Studio ที่มีค่า presC# สัมพันธ์กับค่าใน cert# ที่ถูกเปลี่ยนจะเปลี่ยนแปลงตามไปด้วย

4.8.3 คอนสเตรนท์ของข้อมูล

4.8.3.1 NOT NULL คอนสเตรนท์

เป็นคอนสเตรนท์ที่กำหนดให้ค่าของแอททริบิวต์ห้ามเป็น NULL การกำหนดต้องใช้คำสั่ง NOT NULL ตามหลังการประกาศแอททริบิวต์ตอนสร้างตาราง เช่น ถ้าเราเปลี่ยนคำสั่งในตัวอย่างข้างต้นเป็น

```
4)      presC#   INT  REFERENCES MovieExec(cert#) NOT NULL
```

จะทำให้เราไม่สามารถเปลี่ยนแปลงค่าของ presC# ในตาราง Studioให้เป็น NULL ได้ รวมทั้งไม่สามารถใส่แถวข้อมูลซึ่งมีเฉพาะ ค่า name และ address ในตาราง Studio ได้ และไม่สามารถใช้คำสั่ง SET NULL ในบรรทัดที่ 5 ดังตัวอย่างข้างต้นได้

4.8.3.2 แอททริบิวต์-เบส เช็ค คอนสเตรนท์ (Attribute-Based CHECK Constraints)

เราสามารถใส่คำสั่ง CHECK เพื่อกำหนดขอบเขตค่าของข้อมูลมที่จะถูกเก็บในแอททริบิวต์ได้

ตัวอย่างเช่น ถ้าเราต้องการกำหนดให้ค่าของ presC# ในตาราง Studio เป็นเลขอย่างน้อย 6 หลัก จะทำได้โดยเปลี่ยนคำสั่งบรรทัดที่ 4 เป็น

```
4)      presC# INT REFERENCES        MovieExec(cert#)
                        CHECK  (presC# >= 100000)
```

หรือถ้าเราต้องการกำหนดให้ค่าของแอททริบิว Gender ในตาราง MovieStar เป็นได้สองค่าคือ F หรื อ M เท่านั้น จะต้องเพิ่มคำสั่งในบรรทัดที่ 4 เป็นดังนี้

```
4)      gender  CHAR(1)  CHECK (gender IN ('F' , 'M')) ,
```

4.8.4 โดเมน คอนสเรนท์ (Domain Constraints)

สามารถกำหนดโดเมนของแอททริบิวต์ต่างๆ ได้โดยใช้คำสั่ง CREATE DOMAIN ดังนี้

ตัวอย่าง

```
CREATE DOMAIN GenderDomain CHAR(1)
        CHECK (VALUE IN ('F' , 'M'));
```

และเมื่อจะประกาศแอททริบิวต์ gender ให้มีโดเมนตามที่กำหนดจะสามารถทำได้ดังนี้

```
4)      gender GenderDomain ,
```

4.8.5 โกลบอล คอนสเตรนท์ (Global Constraints)

แบ่งย่อยได้เป็น 2 ชนิดคือ

4.8.5.1 ทัพเพิลเบส เช็ค คอนสเตรนท์ (Tuple based CHECK Constraints)

เป็นเงื่อนไขที่กำหนดขึ้นให้กับข้อมูลในตาราง ก่อนที่จะใส่ข้อมูลหรือทำการเปลี่ยนแปลงข้อมูลใดๆ ในตาราง ต้องมีการตรวจสอบว่าข้อมูลตรงตามเงื่อนไขที่กำหนดไว้หรือไม่ หากไม่ถูกต้องตามเงื่อนไข การเปลี่ยนแปลงนั้นจะถูกยกเลิกทันที

ตัวอย่าง

```
1)      CREATE TABLE MovieStar (
2)              name CHAR(30) UNIQUE ,
3)              address VARCHAR(255) ,
4)              Ggender CHAR(1) ,
5)              Bbirthdate DATE ,
6)              CHECK (gender = 'F' OR name NOT LIKE 'ms.%') );
```

ตัวอย่างนี้เป็นการกำหนดเงื่อนไขว่า ถ้าเป็นเพศชายแล้ว ชื่อจะต้องไม่ขึ้นต้นด้วย Ms.

4.8.5.2 Assertion

ใช้เพื่อกำหนดเงื่อนไขที่ซับซ้อนยิ่งขึ้นให้กับข้อมูล มีรูปแบบการประกาศดังนี้

```
CREATE ASSERTION <name> CHECK (<condition>)
```

ตัวอย่าง

```
CREATE ASSERTION SumLength
        CHECK ( 10000 >= ALL ( SELECT SUM (length)
                FROM Movie
```

```
GROUP BY StudioName ) );
```

เป็นการกำหนดเงื่อนไขว่าความยาวรวมทั้งหมดของภาพยนตร์ของแต่ละสตูดิโอต้องไม่เกิน 10,000 นาที

4.9 ทริกเกอร์ (Trigger)

ทริกเกอร์ถือเป็นคอนสเตรนท์แบบหนึ่ง แต่มีข้อแตกต่างจากคอนสเตรนท์อื่นที่กล่าวไปแล้ว ตรงที่ ทริกเกอร์จะถูกตรวจสอบเมื่อมีเหตุการณ์ (Event) ที่กำหนดไว้เข้ามาเท่านั้น ระบบการจัดการฐานข้อมูล (DBMS) จะตอบสนองเหตุการณ์ที่เข้ามาก็ต่อเมื่อเงื่อนไขที่กำหนดไว้ถูกตรวจสอบแล้วว่าเป็นจริง ทั้งนี้โปรแกรมเมอร์จะเป็นผู้กำหนดว่าจะให้ระบบจัดการฐานข้อมูลตอบสนองเหตุการณ์นั้นอย่างไร

ตัวอย่าง

เราสามารถกำหนดทริกเกอร์สำหรับตาราง MovieExec(name, address, cert#, netWorth) ได้ดังนี้

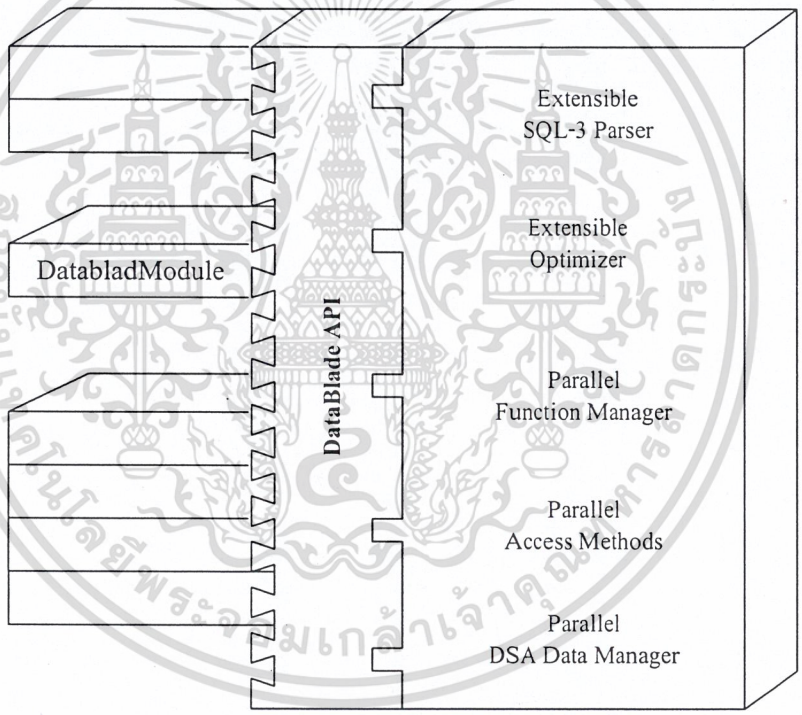
```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec      → EVENT
REFERENCE
    OLD AS OldTuple ,
    NEW AS NewTuple
WHEN (OldTuple.netWorth > NewTuple.netWorth)  → CONDITION
    UPDATE MovieExec
    SET netWorth = OldTuple.netWorth          } ACTION
    WHERE cert# = NewTuple.cert#
FOR EACH ROW
```

จากทริกเกอร์ที่กำหนด เมื่อจะมีการเปลี่ยนแปลงค่า netWorth ในตาราง MovieExec (เป็นเหตุการณ์) จะต้องมาตรวจสอบเสียก่อนว่า netWorth ค่าใหม่น้อยกว่าค่าเดิมหรือไม่ ถ้าน้อยกว่าระบบจัดการฐานข้อมูลจะทำตามแอคชัน (Action) ที่กำหนดไว้

บทที่ 5
อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์
(INFORMIX - Universal Server)

5.1 สถาปัตยกรรมของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

ในอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ ข้อมูลเกี่ยวกับชนิดของข้อมูล (Data type), ฟังก์ชันและการเข้าถึงเมทธอดจะเก็บไว้ในตารางแคตตาล็อก (Catalog Table) มากกว่าที่จะเก็บฮาร์ดโค้ด (Hard-coded) ไว้ในเซิร์ฟเวอร์



รูปที่ 5-1 แสดงสถาปัตยกรรมของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

ส่วนประกอบของเซิร์ฟเวอร์เอ็นจิน (Server Engine) และหน้าที่ต่างๆ มีดังนี้

- **พาสเซอร์ (Parser)** ทำหน้าที่อ่านและตรวจสอบไวยกรณ์ ยูสเซอร์ดีฟายด์ฟังก์ชัน (User-defined function) และ ชนิดของข้อมูลจะถูกบันทึกลงในตารางแคตตาล็อกและจะถูกตรวจสอบโดยพาสเซอร์
- **อ็อบติไมซ์เซอร์ (Optimizer)** ทำหน้าที่เลือกทางที่ดีที่สุดเพื่อเข้าถึงข้อมูลที่ถูกคิวรี ซึ่งจะทำโดยดาต้า

เบลด เดเวล็อปเปอร์ (DataBlad Developer)

- **ฟังก์ชันเมเนเจอร์ (Function Manager)** ทำหน้าที่หาและจัดการยูสเซอร์ดีฟายด์ฟังก์ชันเพราะว่ายูสเซอร์ดีฟายด์ฟังก์ชันจะถูกเก็บอยู่ในตารางแคตตาล็อก
- **แอคเซสเมทธอด (Access Method)** ที่เหมาะสมจะถูกเลือกมาสำหรับข้อมูลที่กำลังถูกแก้ไข ซึ่งจะทำโดยดาต้าเบลดเดเวล็อปเปอร์
- **ดาต้าเมเนเจอร์ (Data Manager)** ทำหน้าที่เคลื่อนย้ายข้อมูลเข้า-ออกจากดิสก์

ระบบฐานข้อมูลของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์เป็นระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ มีคุณสมบัติตามมาตรฐานของเอสคิวแอล 3 ซึ่งเป็นระบบฐานข้อมูลที่ผสมผสานความสามารถที่ดีที่สุดให้มารวมกันอยู่ในระบบเดียวกัน

ระบบการจัดการฐานข้อมูลนี้ได้รวมเอาระบบฐานข้อมูลที่มีประสิทธิภาพและสามารถขยายระบบได้อย่างต่เนื่อง (Scalability) ของสถาปัตยกรรมของอินฟอร์มิกซ์ ในแบบดีเอสเอ (Dynamic Scalable Architecture) การออกแบบเช่นนี้ทำให้อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์เป็นระบบจัดหารฐานข้อมูลที่เหมาะสมสำหรับองค์กรที่ต้องการระบบจัดการฐานข้อมูลที่มีความเชื่อถือได้และมีประสิทธิภาพสูง มีการขยายขีดความสามารถในการจดการการค้นหาข้อมูลที่ซับซ้อนและข้อมูลในทุกรูปแบบ เช่น ภาพ เสียง เว็บ หรือเอกสารอิเล็กทอรนิกส์หรือข้อมูลแบบ 2D 3D ได้ ซึ่งอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์จะใช้ดาต้าเบลดในการจัดการข้อมูลนั้นได้อย่างมีประสิทธิภาพ

5.2 จุดเด่นของโครงสร้างสถาปัตยกรรมของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

5.2.1 การออกแบบมีประสิทธิภาพสูง (Performance)

การออกแบบอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ ใช้โครงสร้างเดียวกบระบบฐานข้อมูลแบบขนานแบบดีเอสเอ ซึ่งออกแบบให้สามารถใช้ได้ทั้งระบบคอมพิวเตอร์แบบโปรเซสเซอร์เดียวและแบบหลายโปรเซสเซอร์ และเป็นระบบที่สามารถใช้ประสิทธิภาพของฮาร์ดแวร์ได้อย่างมีประสิทธิภาพ และอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ยังสามารถปรับแต่งให้การทำงานของระบบที่บางครั้งอาจเป็นโพรเซสที่ซับซ้อนให้สามารถจัดการกับข้อมูลที่เกี่ยวข้องอย่างใกล้ชิด ทำให้การพัฒนาโปรแกรมทำได้ง่ายโดยไม่จำเป็นต้องค้นหาข้อมูลที่ซับซ้อนเอง

5.2.2 การออกแบบให้สามารถขยายได้ (Extensible)

ในระบบการจัดการฐานข้อมูลโดยทั่วไปจะจัดการกับข้อมูลที่เป็นตัวเลข ตัวอักษร แต่อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ออกแบบให้สามารถใช้วานอย่างไม่มีขีดจำกัดโดยสามารถที่จะใช้งานกับข้อมูลประเภทใดก็ได้ เช่น ภาพ วิดีโอ เสียง แกนเวลา สองมิติ สามมิติ ภาพ ข้อความ และข้อมูลที่ใช้กับเว็บ

5.2.3 ออกแบบให้เชื่อมต่อกันอย่างดี (Integrated)

การออกแบบที่รวบรวมข้อดีของระบบจัดการฐานข้อมูลแบบขนานและโครงสร้างข้อมูลแบบออบเจ็กต์โอเรียนเต็ดที่ทำให้ระบบสามารถขยายขีดความสามารถและนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ การรวมกันนี้ทำให้เป็นระบบจัดการฐานข้อมูลที่ทั้งสามารถขยายได้และเพิ่มความสามารถของระบบได้เป็นอย่างดี การที่สามารถรวมอยู่ในระบบเดียวกันได้นี้ทำให้ได้ประสิทธิภาพสูงสุด และลดความซ้ำซ้อนของระบบจัดการฐานข้อมูลและระบบงานได้ และยังคงไว้ด้วยความปลอดภัยและความเชื่อถือได้ของระบบ

5.2.4 ออกแบบที่นวัตกรรมใหม่ (Innovation)

การออกแบบอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์ที่คำนึงถึงความเป็นระบบเปิด (Open System) ระบบที่ขยายได้โดยไม่มีขีดจำกัด (Unlimit Extensibility) และเพิ่มประสิทธิผลให้กับโปรแกรมเมอร์ทำให้สามารถสร้างระบบที่เป็นสากลต่างๆ เช่น มาตรฐานเอสคิวแอล 3 การเชื่อมต่อกับเน็ตสเคปหรือไมโครซอฟต์บราวซ์เซอร์ หรือการใช้งานในระบบขนาดเล็กจนถึงขนาดใหญ่ได้อย่างมีประสิทธิภาพ

5.2.5 การคิวรีข้อมูลของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

ระบบฐานข้อมูลของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์มีข้อมูลที่มีโครงสร้างเป็นแบบข้อมูลที่ซับซ้อน (Complex Data) ซึ่งเราสามารถที่จะคิวรีข้อมูลในภาพแบบของข้อมูลที่ซับซ้อนได้ รวมทั้งสามารถสร้างข้อมูลที่เป็นไทพ์ชนิดใหม่ได้

Query	Relational DBMS	Object Relational DBMS
No Query	File System	Object-Oriented DBMS
	Simple Data	**Complex Data**

รูปที่ 5-2 แสดงการคิวรีข้อมูลของระบบฐานข้อมูลแบบต่างๆ

5.3 ชนิดข้อมูลชนิดใหม่ (New Data Type)

ในอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์จะมีดาต้าไทพ์ชนิดใหม่เพิ่มเข้ามา คือ โรว์ไทพ์ , คอลเลคชัน (Collections) , ยูสเซอร์ดีฟายด์ดาต้าไทพ์ (User-defined data type) เช่น ดิสทิงค์ (Distinct) หรือ โอเปคไทพ์ (Opaque Type)

- **โรว์ไทพ์** จะให้ความสามารถในการรวมคอลัมน์หลายๆคอลัมน์เข้าด้วยกันเพื่อให้ง่ายต่อการเข้าถึงและการดำเนินการต่างๆ
- **คอลเลคชัน** จะอนุญาตให้มีข้อมูลได้หลายๆค่าในคอลัมน์เพียงคอลัมน์เดียว เช่น

```
Craete table customer_contact
        (name  char (20) ,
        Phone set  (row  ( phonetype  varchar(10) ,
                          Phoneno  char(10))
                not  null)  );
```

- **ดิสทิงค์ไทพ์** จะอนุญาตให้มีการแบ่งแยกความแตกต่างของข้อมูลทางธุรกิจ (Business data) ไว้อย่างชัดเจน เช่น เมื่อกำหนดไทพ์สำหรับหน่วยเงินตราที่แตกต่างกัน (ดอลล่าร์ , ฟรังค์, เยน ,...) ก็จะทำให้คอลัมน์ที่กำหนดสำหรับ ดอลล่าร์จะไม่สามารถรับข้อมูลที่เป็นฟรังค์หรือหน่วยอื่นๆ ได้ และคอลัมน์ที่เป็นหน่วยอื่นๆก็เช่นกัน
- **โอเปคไทพ์** เป็นแอปสแตรกดาต้าไทพ์ที่มีโครงสร้างภายในซับซ้อนใช้เก็บข้อมูลขนาดใหญ่ เช่น ข้อความ , เสียง ,วิดีโอ เป็นต้น เราสามารถสร้างโอเปคไทพ์และกำหนดฟังก์ชัน โอเปอร์เรเตอร์ และการข้าถึงเมทธอดเพื่อจัดการกับตัวมันเอง ซึ่งโอเปคจะสนับสนุนออบเจ็กต์โอเรียนเต็ดของข้อมูลไพรเวท (Private data)

5.4 การสืบทอดคุณสมบัติในอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

การสืบทอดคุณสมบัติในอินฟอร์มิกซ์ จะทำได้ 2 แบบ คือ

1. การสืบทอดคุณสมบัติสำหรับโรว์ไทพ์
2. การสืบทอดคุณสมบัติสำหรับตาราง

5.4.1 การสืบทอดคุณสมบัติสำหรับโรว์ไทพ์

เมื่อเราสร้างโรว์ไทพ์หนึ่งเป็นซุปเปอร์ไทพ์ เราสามารถสร้างซับไทพ์ของมันได้โดยใช้คำสั่ง

UNDER <ชื่อของ Supertype>

ตัวอย่างเช่น

```
CREATE ROW TYPE EmployeeType  (
        Name    char(20),
        Dept    integer,
        Salary  money);

CREATE ROW TYPE SalemanType  (
        Quota   money,
```

```
        Attainment    decimal(4,2),
        Commission  decimal(4,2),
        Accounts  set(varchar(30) not null))
    UNDER EmployeeType;

    CREATE ROW TYPE  EngineerType (
        MBO_bonus  money,
        Skills  set(varchar(30) not null))
    UNDER EmployeeType;
```

จากตัวอย่าง เป็นการสร้างให้ EmployeeType เป็นซุปเปอร์ไทพ์ของ SalemanType และ EngineerType ซับไทพ์จะสืบทอดคุณสมบัติทุกๆ ฟิล์ดข้อมูล (Data Field) ของซุปเปอร์ไทพ์นั่นคือเราจะมองได้ว่าฟิล์ด name ,dept และ salary ถูกรวมเป็นฟิล์ดข้อมูลของ SalemanType และ EngineerType ด้วย นอกจากนี้แล้วซับไทพ์ยังสามารถสืบทอดพฟติกรรมอื่นๆที่ ซุปเปอร์ไทพ์มีอีกด้วย เช่น ยูสเซอร์ดีฟายด์รูทีน(User-defined Routines)

5.4.2 การสืบทอดคุณสมบัติสำหรับตาราง

เมื่อเรา Create Table หนึ่งเป็น SuperTable เราสามารถ Create SubTable ของมันได้โดยใช้คำสั่ง

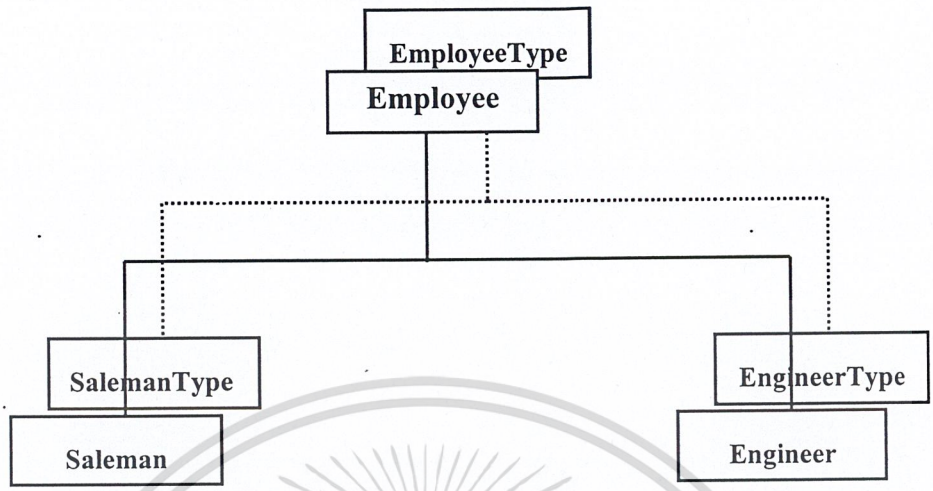
UNDER < ชื่อของ SuperTable>

ตัวอย่างเช่น

```
CREATE TABLE Employee OF TYPE EmployeeType;
CREATE TABLE Employee OF TYPE EmployeeType UNDER Employee;
CREATE TABLE Employee OF TYPE EmployeeType UNDER Employee;
```

SubTable ที่สร้างขึ้นนี้จะสามารถสืบทอกคุณสมบัติต่างๆของ SuperTable ได้แก่ คอลัมน์ , ดัชนี (Indices) , ทริกเกอร์ , การเลือกที่เก็บข้อมูล (Storage Options) และคอนสเตรนท์

ลักษณะความสัมพันธ์ของไทพ์และตาราง จะมีลักษณะดังรูป



รูปที่ 5-3 แสดงลักษณะการสืบทอดคุณสมบัติระดับโรว์ไทพ์และตาราง

5.4.3 ตัวอย่างแสดงการสืบทอดคุณสมบัติคอลัมน์จากซุปเปอร์เทเบิล

เมื่อเราทำการอินเสิร์ทข้อมูลลงในซับเทเบิล เราจะต้องเพิ่มข้อมูลในคอลัมน์ของซุปเปอร์เทเบิลลงไปด้วย

```
INSERT INTO Employee
        VALUES ('Fred Smith',
                111,
                1000.00);
INSERT INTO Saleman
        VALUES('Tom Jones',
                222,
                2000.00,
                50000.00,
                80.00,
                5.00,
                "SET {'ABC Shoes' , 'Leiner Boots'}" );
INSERT INTO Engineer
        VALUES('Janet Brown',
                333,
```

3000.00,

1000.00,

"SET {'Cobol' , 'Java'}");

หากเราลอง SELECT ข้อมูลจาก Supertable จะได้ผลลัพธ์ดังนี้

SELECT name,dept FROM Employee;

RETURN

Name	Dept
Fred Smith	111
Tom Jones	222
Janet Brown	333

แสดงว่า เมื่อมีการเรียกข้อมูลในซุปเปอร์เทเบิลขึ้นมา ข้อมูลในซับเทเบิลของมันก็จะถูกนำมาแสดงด้วย แต่ถ้าต้องการเรียกเฉพาะข้อมูลที่อยู่ในซุปเปอร์เทเบิลเท่านั้น จะสามารถทำได้โดยใช้คีย์เวิร์ด ONLY ดังนี้

SELECT name,dept FROM ONLY (Employee);

RETURN

Name	Dept
Fred Smith	111

** *<u>หมายเหตุ</u> การที่จะใช้คุณสมบัติการสืบทอดของโรว์ไทพ์และตารางได้จะต้องมีการกำหนดโรว์ไทพ์ก่อนที่จะกำหนดตารางเสมอ* **

5.4.4 ตัวอย่างการสืบทอดคุณสมบัติของยูสเซอร์ดีฟายด์รูทีนของซับไทพ์

ลักษณะของยูสเซอร์ดีฟายด์รูทีนเป็นดังนี้

```
CREATE  FUNCTION Compensation (emp EmployeeType)
                                RETURNING money;
        RETURN emp.salary;
END  FUNCTION;
```

อาจสามารถเรียกใช้ฟังก์ชันดังนี้

```
SELECT  name,salary,compensation(e)  Compensation
FROM Employee e;
```

RETURNS

Name	Salary	Compensation
Fred Smith	$1000.00	$1000.00
Tom Jones	$2000.00	$2000.00
Janet Brown	$3000.00	$3000.00

จากผลลัพธ์ที่ได้แสดงให้เห็นว่า เมื่อเรากำหนดฟังก์ชันให้ทำการเข้าถึงฟิลด์ของซุปเปอร์ไทพ์ที่ชื่อ EmployeeType แล้วเรียกใช้ฟังก์ชันนั้น การสืบทอดคุณสมบัติฟิลด์ในซับไทพ์ก็จะถูกเข้าถึงด้วยเช่นกัน ถ้าเรากำหนดฟังก์ชันให้เข้าถึงเฉพาะซับไทพ์รวมเข้าไปด้วย

```
CREATE  FUNCTION Compensation (sales SalemanType)
                                        RETURNING money;
    RETURN (sales.salary
                + sales.commission/100
                * sales.attainment/100*sales.quota);
END  FUNCTION;
```

และทำการ SELECT เช่นเดิมจะได้ผลลัพธ์ต่างออกไปดังนี้

Name	Salary	Compensation
Fred Smith	**$1000.00**	**$1000.00**
Tom Jones	$2000.00	***$4000.00***
Janet Brown	$3000.00	$3000.00

เพราะรูทีนซึ่งถูกกำหนดไว้เฉพาะดังกล่าว จะมีผลกับซับไทพ์แทนที่รูทีนของซุปเปอร์ไทพ์นั้น

5.5 ชนิดข้อมูลในอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

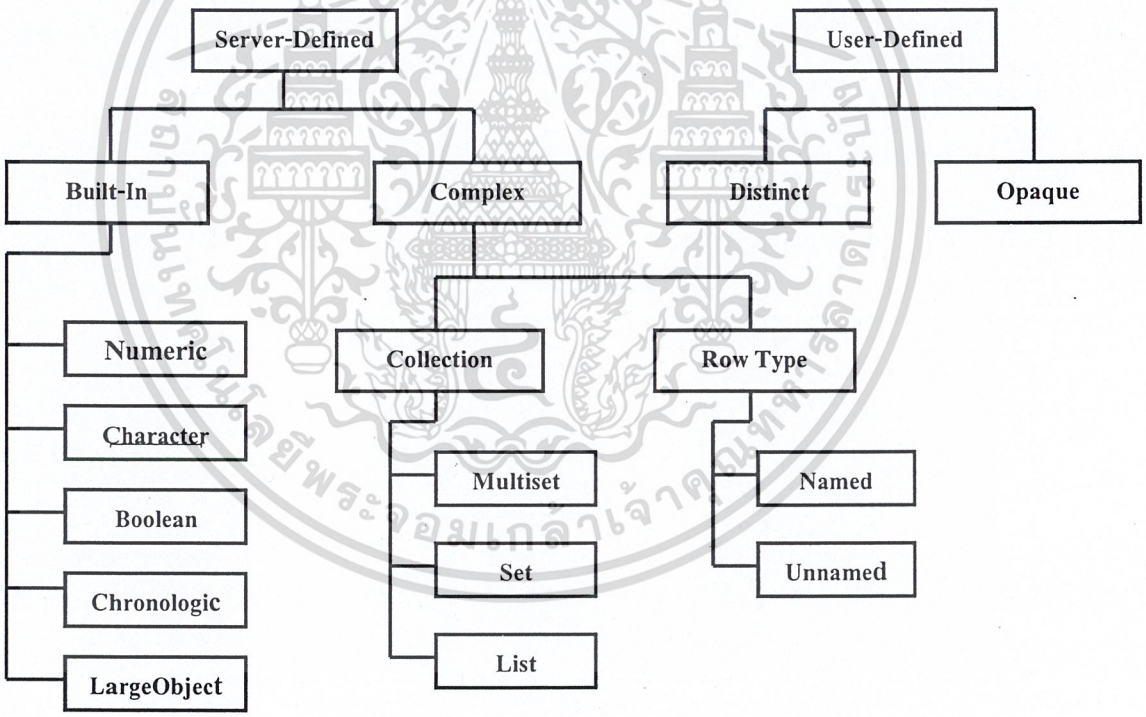
แบ่งอธิบายได้เป็น 2 หัวข้อใหญ่ๆ ดังนี้

1. ชนิดข้อมูลที่อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์กำหนด (Data Types That Universal Server Provides)
2. ชนิดข้อมูลที่ผู้ใช้กำหนด (Data Types That User Provides)

5.5.1 ชนิดข้อมูลที่อินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์กำหนด

ยูนิเวอร์แซล เซิร์ฟเวอร์จะรู้จักโครงสร้างภายใน (Internal Structure) ของชนิดข้อมูลต่อไปนี้

- บิวท์อินดาต้าไทพ์ (Built-in data types)
- คอมเพล็กซ์ดาต้าไทพ์ (Complex data types)



รูปที่ 5-4 แสดงชนิดข้อมูลของอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

5.5.1.1 บิวท์อินดาต้าไทพ์ (Built-in data types) มีลักษณะดังนี้

1> เป็นชนิดข้อมูลพื้นฐานซึ่งดาต้าเบสเซิร์ฟเวอร์จัดให้มีไว้

2> ไม่สามารถแบ่งย่อยออกเป็นส่วนเล็กๆ ต่อไปได้อีก (Atomic)

3> บิวท์อินดาต้าไทพ์ของยูนิเวอร์แซล เซิร์ฟเวอร์ มีดังนี้

DATA TYPE	EXPLANATION
BLOB	เก็บข้อมูลไบนารีในชังค์ที่เข้าถึงแบบสุ่ม (Random-access Chunks)
BOOLEAN	เก็บค่าบูลีน คือ True และ False.
BYTE	เก็บข้อมูลไบนารีในชังค์ที่ไม่เข้าถึงแบบสุ่ม
CHAR(n)	เก็บซิงเกิลไบท์ (Single-byte) หรือมัลติเพิล (Multibyte) ตามลำดับของตัวอักษร รวมทั้งตัวหนังสือ,ตัวเลขและสัญลักษณ์ซึ่งมีความยาวจำกัดตามค่า n
CHARACTER(n)	เหมือนกับข้อมูลชนิด CHAR
CHARACTER	เก็บซิงเกิลไบท์ (Single-byte) หรือมัลติเพิล (Multibyte) ตามลำดับของตัวอักษร
VARYING(m,r)	รวมทั้งตัวหนังสือ,ตัวเลขและสัญลักษณ์
CLOB	เก็บข้อมูลที่เป็น TEXT ในชังค์ที่เข้าถึงแบบสุ่ม
DATETIME	เก็บข้อมูลวันที่และเวลา
DATE	เก็บข้อมูลวันที่ตามปฏิทิน
DECIMAL	เก็บค่าตัวเลขที่เป็นทศนิยมที่แน่นอน
DEC	เหมือนกับ DECIMAL
FLOAT(n)	เก็บค่า floating-point numbทำให้เกิด double data type ในภาษาซี
DOUBLE PRECISION	เหมือนกับ FLOAT.
INT8	เก็บค่าจำนวนเต็มขนาด 8 ไบท์ มีค่าระหว่าง –9,223,372,036,854,775,87 ถึง 223,372,036,854,775,807 หรือ $-(2^{63}-1)$ ถึง $(2^{63}-1)$
INTEGER	เก็บค่าจำนวนเต็ม ที่มีค่าระหว่าง –2,147,483,647 ถึง +2,147,483,647 หรือ $-(2^{31}-1)$ ถึง $(2^{31}-1)$
INT	เมือนกับ INTEGER
INTERVAL	เก็บค่าช่วงของเวลา
LVARCHAR	เก็บค่าซิงเกิลไบท์หรือมัลติเพิลไบท์ของตัวอักษร , ตัวเลข และสัญลักษณ์ซึ่งมีความยาวเปลี่ยนแปลงได้มากถึง 32 กิโลไบท์ ส่วนใหญ่จะเป็นรูปแบบเอ็กซ์เทอนอลสตอร์เรจสำหรับโอเปคไทพ์
MONEY(p,S)	เก็บอัตราแลกเปลี่ยนของเงิน
NCHAR(n)	เก็บค่าซิงเกิลไบท์และมัลติเพิลไบท์ของตัวอักษร , ตัวเลข และสัญลักษณ์ต่างๆ
NUMERIC(p,s)	เก็บค่าซิงเกิลไบท์และมัลติเพิลไบท์ของตัวอักษร , ตัวเลข และสัญลักษณ์ต่างๆ

DATA TYPE	EXPLANATION
NUMERIC(p,s)	เหมือนกับ DECIMAL
NVARCHAR(m,r)	เก็บค่าชิงเกิลไบท์และมัลติเพิลไบท์ของตัวอักษร , ตัวเลข และสัญลักษณ์ต่างๆ ซึ่งมีความยาวได้มากที่สุด 255 ไบท์
REAL	เหมือนกับ SMALLFLOAT.
SERIAL	เก็บจำนวนเต็มที่เป็นลำดับ ซึ่งมีขอบเขตของค่าเหมือน INTEGER
SERIAL8	เก็บจำนวนเต็มที่เป็นลำดับ ซึ่งมีขอบเขตของค่าเหมือน INT8
SMALLFLOAT	เก็บค่า single-precision floating-point numbers ซึ่งเป็น float data type ในภาษาซี
SMALLINT	เก็บค่าตัวเลขตั้งแต่ค่า –32,767 ถึง +32,767 หรือ $-(2^{15}-1)$ ถึง $(2^{15}-1)$
TEXT	เก็บข้อมูลชนิดเท็กซ์ในชังค์ที่ไม่เข้าถึงแบบสุ่ม
VARCHAR(m,r)	เก็บค่าชิงเกิลไบท์และมัลติเพิลไบท์ของตัวอักษร , ตัวเลข และสัญลักษณ์ต่างๆ ซึ่งมีความยาวได้มากที่สุด 255 ไบท์ เก็บข้อมูลไบนารีในชังค์ที่ไม่เข้าถึงแบบสุ่ม

ตารางที่ 5-1 แสดง Built-in Data Type ในอินฟอร์มิกซ์ ยูนิเวอร์แซล เซิร์ฟเวอร์

ตัวอย่างการกำหนดบิวท์อินดาต้าไทพ์

```
CREATE TABLE restaurant(
    Rcode   integer,
    Name    char(50),
    Address char(10));
```

5.5.1.2 คอมเพล็กซ์ดาต้าไทพ์ ถูกสร้างขึ้นจากการใช้เอสคิวแอลไทพ์คอนสตรักเตอร์ (SQL type constructor) จัดการกับชนิดข้อมูลต่างๆ ที่มีอยู่ แบ่งได้เป็น 2 ชนิด คือ

- **คอลเล็กชันไทพ์ (Collection Types)** เกิดจากการรวมกันของดาต้าไทพ์ชนิดเดียวกัน (จะเป็นดาต้าไทพ์พื้นฐานหรือคอมเพล็กซ์ดาต้าไทพ์ก็ได้) โดยใช้เอสคิวแอลไทพ์คอนสตรักเตอร์ดังต่อไปนี้ SET, LIST หรือ MULTISET

DATA TYPE	EXPLANATION
LIST(e)	เก็บค่าข้อมูลหลายๆ ค่าแบบมีลำดับและอนุญาตให้มีสมาชิกซ้ำกันได้ สมากชิกทุกตัวต้องเป็นข้อมูลชนิดเดียวกัน
MULTISET(e)	เก็บค่าข้อมูลหลายๆ ค่าแบบไม่มีลำดับและอนุญาตให้มีสมาชิกซ้ำกันได้ สมากชิกทุกตัวต้องเป็นข้อมูลชนิดเดียวกัน
Set(e)	เก็บค่าข้อมูลหลายๆ ค่าแบบไม่มีลำดับและสมากชิกไม่ซ้ำกัน สมากชิกทุกตัวต้องเป็นข้อมูลชนิดเดียวกัน

ตารางที่ 5-2 แสดงชนิดของคอลเล็คชันไทพ์

ตัวอย่างการกำหนดคอลเล็คชันไทพ์

```
CREATE TABLE items(
        Item_no     integer,
        Description  varchar(40),
        Colors       multiset (char(10) not null));
```

ลักษณะของโรว์ของตาราง items เป็นดังนี้

```
INSERT INTO items VALUES(
        1234,
        'gardening clogs',
        "multiset { 'red', 'blue', 'green' }" );
```

- **โรว์ไทพ์ (Row Type)** เป็นชุดของฟิล์ดข้อมูลที่สัมพันธ์กัน ซึ่งอ้างถึงดาต้าไทพ์ใดๆ แบ่งออกเป็น 2 ชนิด คือ

  1> โรว์ไทพ์ชนิดไม่มีชื่อ (Unnamed Row Type) มีลักษณะดังนี้

```
CREATE TABLE customer (
        Cus_no   integer,
        Name     char(30),
Address row( street      char(30),
             City        char(20),
             State       char(2),
```

Zip char(5)));

2> โรว์ไทพ์ชนิดมีชื่อ (Named Row Type) ลักษณะดังนี้

CREATE **ROW TYPE** Customer(

Name char(20),

Order char(30),

Cost integer);

5.5.2 ชนิดข้อมูลที่ผู้ใช้กำหนด

ผู้ใช้สามารถกำหนดชนิดของข้อมูลต่อไปนี้ไว้ในยูนิเวอร์แซล เซิร์ฟเวอร์ได้

- ชนิดข้อมูลที่ผู้ใช้กำหนดเอง (User-defined data type) ได้แก่
  1. โอเปคไทพ์ (Opaque Type)
  2. ดิสทิงค์ดาต้าไทพ์ (Distinct Type)
- โรว์ไทพ์ชนิดมีชื่อ (Named row type)
- ดาต้าเบลด โมดูลดาต้าไทพ์ (DataBlade module data type)

5.5.2.1 ข้อมูลชนิดโอเปค (Opaque Type)

เป็นข้อมูลชนิดที่ไม่สามารถแบ่งย่อยลงไปอีกได้ ดาต้าเบสเซิร์ฟเวอร์จะไม่รู้จักโครงสร้างภายในของโอเปคไทพ์ ดังนั้นในการกำหนดโอเปคไทพ์จึงต้องมีสร้างข้อมูลต่อไปนี้ไว้ด้วย

1.) ฟังก์ชันที่สนับสนุน (Support Function) เพื่อใช้บอกวิธีที่ดาต้าเบสเซิร์ฟเวอร์จะสามารถติดต่อกับโครงสร้างภายในของโอเปคไทพ์ได้

2.) โอเปอร์เรชัน (Operations) ใช้จัดการกับข้อมูล

3.) โอเปอร์เรเตอร์คลาส (Operator Class)

4.) แคสท์ติ้งฟังก์ชัน (Casting Function) ทำหน้าที่แปลงข้อมูลไปเป็นชนิดโอเปคหรือแปลงข้อมูลชนิดโอเปคไปเป็นข้อมูลแบบอื่น

5.5.2.2 ดิสทิงค์ดาต้าไทพ์ (Distinct Type)

ถูกสร้างขึ้นมาจากการ กำหนดชื่อให้กับชนิดข้อมูลที่มีอยู่ ตัวอย่างเช่น

CREATE DISTINCT TYPE fahrenheit AS integer;

เป็นการสร้างดาต้าไทพ์ชื่อว่า Fahrenheit จากไทพ์ integer ซึ่งมีอยู่เดิม

5.5.2.3 โรว์ไทพ์ชนิดมีชื่อ

มีการกำหนดคล้ายกับโรว์ไทพ์ชนิดที่ไม่มีชื่อ แต่ต้องแยกกำหนดต่างหากและต้องเพิ่มชื่อของโรว์ไทพ์เข้าไปดังนี้

```
CREATE ROW TYPE  Address_t (
        Street    char(30),
        City      char(20),
        State     char(2),
        Zip       char(5) );
```

5.5.2.4 ดาต้าเบลด โมดูลดาต้าไทพ์

เป็นชนิดข้อมูลซึ่งถูกกำหนดไว้ภายนอก ดาต้าเบลดโมดูล (DataBlade Module) จะต้องประกอบด้วย Data type, Casts, Operations และ Secondary access method ของดาต้าไทพ์นั้น

5.6 อธิบายศัพท์ที่เกี่ยวข้องกับดาต้าไทพ์

5.6.1 โอเปอร์เรชัน (Operations) มีทั้งหมด 4 แบบ

5.6.1.1 โอเปอร์เรเตอร์ฟังก์ชันก์ (Operator Function) ใช้ในการสร้างสัญลักษณ์ของโอแปอร์เรเตอร์ เช่น

Plus() เป็นโอเปอร์เรเตอร์ฟังก์ชันก์ของโอเปอร์เรเตอร์ “ + “

Times() เป็นโอเปอร์เรเตอร์ฟังก์ชันก์ของโอเปอร์เรเตอร์ “ * ”

โอเปอร์เรเตอร์ฟังก์ชันก์ที่ยูนิเวอร์แซล เซิร์ฟเวอร์จัดไว้ให้ใช้กับบิวท์อินดาต้าไทพ์มีดังนี้

Arithmetic Operator	Operator Function
+(binary)	Plus()
-(binary)	minus()
*	times()
/	divide()
+(unary)	positive()
+(unary)	negative()

ตารางที่ 5-3 แสดงโอเปอร์เรเตอร์ฟังก์ชันก์ที่ยูนิเวอร์แซล เซิร์ฟเวอร์จัดไว้ให้ใช้กับบิวท์อินดาต้าไทพ์

TEXT OPERATOR	OPERATOR FUNCTION
LIKE	like()
MATCHES	matches()
||	concat()

ตารางที่ 5-4 แสดงเท็กซ์โอเปอร์เรเตอร์ที่ยูนิเวอร์แซล เซิร์ฟเวอร์จัดไว้ให้ใช้กับบิวท์อินดาต้าไทพ์

RELATIONAL OPERATOR	OPERATOR FUNCTION
=	Equal()
<> and !=	Notequal()
>	greaterthan()
<	lessthan()
>=	greaterthanorequal()
<=	lessthanorequal()

ตารางที่ 5-5 แสดงรีเลชันแนลโอเปอร์เรเตอร์ที่ยูนิเวอร์แซล เซิร์ฟเวอร์จัดไว้ให้ใช้กับบิวท์อินดาต้าไทพ์

5.6.1.2 บิวท์อินดาต้าไทพ์ (Built-in Function)

เป็นฟังก์ชันที่ดาต้าเบสเซิร์ฟเวอร์กำหนดให้ใช้ในเอสคิวแอลสเตทเมนท์ได้ ฟังก์ชันประเภทนี้เป็น โอ โอเปอร์เรชันทางคณิตศาสตร์ ได้แก่

Built-In Function	Number 0f Parameters	Parameter Types
Abs()	1	Real number
Mod()	2	Integer-number expression, Integer-number expression
Pow()	2	real-number expression, real-number expression
Root()	1 or 2	real-number expression [,real-number expression]
Round()	1 or 2	expression [,literal integer]
Sqrt()	1	real-number expression
Trunc()	1 or 2	expression [,literal integer]
Exp(),log(),logn()	1	positive real-number expression
Cos(),sin(),tan()	1	numeric expression
Acos(),asin(),atan()	1	numeric expression
Atan2()	2	numeric xpression, numeric expression
Hex()	1	integer expression
Length(), Char_length(), Character_length(), Octet_length()	1	character string
User()	0	None
Current()	0	None
Dbservername() *Sitename()*	0	None
Today()	0	None
Extend()	1	Date/time expression

ตารางที่ 5- 6 แสดงโอโอเปอร์เรชันทางคณิตศาสตร์ของบิวท์อินดาต้าไทพ์

5.6.1.3 แอกกรีเกรชันฟังก์ชัน (Aggregate Function)

เป็นฟังก์ชันที่มีการส่งค่ากลับเป็นค่าเดียว(Single value)หรือหลายๆค่า (Set of values) ใช้ได้ในเอสคิวแอลสเตทเมนท์ ได้แก่ COUNT, AVG, MAX, MIN และ SUM

5.6.1.4 เอ็นด์-ยูสเซอร์รูทีน (End-user Routines)

คือ ยูสเซอร์ดีฟายด์รูทีนซึ่งเขียนขึ้นโดยใช้ภาษา SPL(Stored Procedure Language) หรือภาษาภายนอก (External Language) เช่น ภาษาซี แบ่งเป็น 2 ชนิดคือ

- เอ็นด์-ยูสเซอร์ฟังก์ชัน (End-user Function) ส่งค่ากลับให้กับเอสคิวแอลสเตทเมนท์
- เอ็นด์-ยูสเซอร์โพรซิเยอร์ (End-user Procedure) ใช้ในการทำงานบางอย่างที่ไม่มีการส่งค่ากลับ

5.6.2 แคสติ้งฟังก์ชัน (Casting Function)

อาจถือเป็นโอเปอร์เรเตอร์ฟังก์ชันก์อีกประเภทหนึ่ง ใช้สำหรับแปลงดาต้าไทพ์ชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง

5.6.3 โอเปอร์เรเตอร์กลาส (Operator Class)

คือกลุ่มของฟังก์ชันที่ทำหน้าที่ในการ เก็บและค้นหาค่าของดาต้าไทพ์พิเศษซึ่งถูกกำหนดไว้ให้เป็น อินเด็กซ์แบ่งออกเป็น 2 ชนิดดังนี้

5.6.3.1 สเตรทจี ฟังก์ชัน (Strategy Functions) ทำหน้าที่ช่วยคิวรีอ็อบติไมเซอร์ (Query Optimizer) ตัดสินว่า อินเด็กซ์สามารถถูกนำมาใช้ให้เกิดประโยชน์ในการจัดการกับข้อมูลหรือไม่

5.6.3.2 ซับพอร์ทฟังก์ชัน (Support Functions) ใช้ในการสร้างและเข้าถึงอินเด็กซ์

บทที่ 6
สมาร์ทลาร์จออบเจ็กต์
(Smart Large Object)

6.1 ลักษณะของสมาร์ทลาร์จออบเจ็กต์

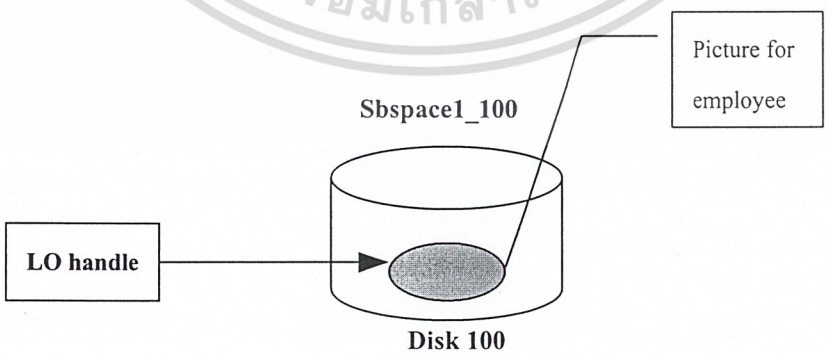
สมาร์ทลาร์จออบเจ็กต์เป็นออบเจ็กต์ขนาดใหญ่ที่มีลักษณะ ดังนี้

- สมาร์ทลาร์จออบเจ็กต์จะเก็บข้อมูลที่มีขนาดใหญ่มาก สามารถเก็บข้อมูลได้มากถึง 4 เทราไบท์ (Terabyte) ซึ่งข้อมูลเหล่านี้จะเก็บอยู่ในดิสก์สเปซ (Disk Space) ที่แยกออกมาต่างหาก เรียกว่า เอสบี-สเปซ (sbspace)
- สมาร์ทลาร์จออบเจ็กต์สามารถกู้ข้อมูลคืนกลับได้ (Recoverable) ดาต้าเบสเซิร์ฟเวอร์สามารถที่จะล็อกการเปลี่ยนแปลงสมาร์ทลาร์จออบเจ็กต์และสามารถกู้ข้อมูลที่เป็นสมาร์ทลาร์จออบเจ็กต์เมื่อระบบหรือฮาร์ดแวร์เกิดความผิดพลาดหรือขัดข้องได้
- สมาร์ทลาร์จออบเจ็กต์จะเข้าถึงข้อมูลแบบสุ่ม ซึ่งสามารถไปที่ตำแหน่งใดๆที่ต้องการแล้วทำการอ่านหรือเขียนข้อมูลที่ตำแน่งนั้นได้ตามต้องการ
- เราสามารถจัดการกับลักษณะต่างๆของที่เก็บข้อมูลของสมาร์ทลาร์จออบเจ็กต์ได้

6.2 ส่วนประกอบของสมาร์ทลาร์จออบเจ็กต์

สมาร์ทลาร์จออบเจ็กต์มี 2 ส่วน คือ

1. เอสบี-สเปซ (sbspace) ซึ่งเป็นที่เก็บข้อมูลของสมาร์ทลาร์จออบเจ็กต์
2. โล-แฮนด์เดิล (LO handle) ซึ่งเป็นตัวชี้ตำแหน่งข้อมูลของสมาร์ทลาร์จออบเจ็กต์ที่อยู่ในเอสบี-สเปซ



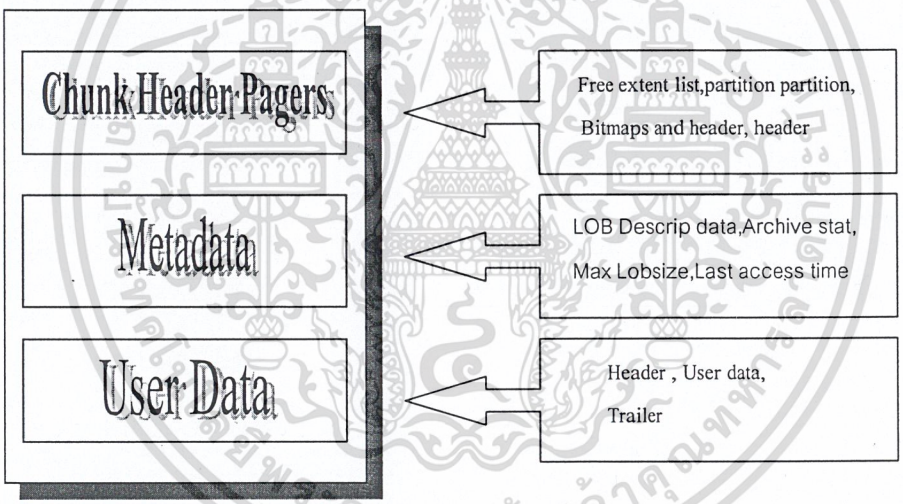
รูปที่ 6-1 แสดงตัวอย่างการเก็บข้อมูลของสมาร์ทลาร์จออบเจ็กต์

จากรูปจะแสดงตัวอย่างการเก็บข้อมูลที่เป็นรูปภาพชื่อ employee ที่เป็นสมาร์ทลาร์จออบเจ็กต์ ซึ่งโล-แฮนด์เดิลจะเก็บตำแหน่งจริงๆของรูป employee ใน sbspace1_100 ในเอสบี-สเปซ

6.2.1 เอสบี-สเปซ (sbspace)

เอสบี-สเปซเป็นพื้นที่เก็บข้อมูลทางลอจิกคอลซึ่งประกอบด้วยชังค์ (Chunk) อย่างน้อย 1 ชังค์ และเก็บเฉพาะข้อมูลที่เป็นสมาร์ทลาร์จออบเจ็กต์เท่านั้น เอสบี-สเปซจะประกอบด้วยส่วนต่างๆ ดังนี้

- เมต้าดาต้า แอเรีย (Metadata Area) จะใช้เก็บข้อมูลต่างๆ เกี่ยวกับสมาร์ทลาร์จออบเจ็กต์ ซึ่งข้อมูลเหล่านั้นจะรวมถึง
  - ข้อมูลภายใน (Internal Information) ซึ่งช่วยให้สามารถจัดการกับข้อมูลได้อย่างมีประสิทธิภาพ
  - สตอร์เรจ คาแรกเตอร์ริสติก (Storage Characteristic) ของสมาร์ทลาร์จออบเจ็กต์
  - ข้อมูลสถานะ (Status Information) ของสมาร์ทลาร์จออบเจ็กต์
- ยูสเซอร์ดาต้าแอเรีย (User Data Area) จะใช้เก็บข้อมูลของสมาร์ทลาร์จออบเจ็กต์



รูปที่ 6-2 แสดงรูปแบบของเอสบี-สเปซ

6.2.1.1 ข้อมูลเกี่ยวกับสมาร์ทลาร์จออบเจ็กต์

จะเก็บไว้ในส่วนของเมต้าดาต้าแอเรีย ซึ่งประกอบด้วย

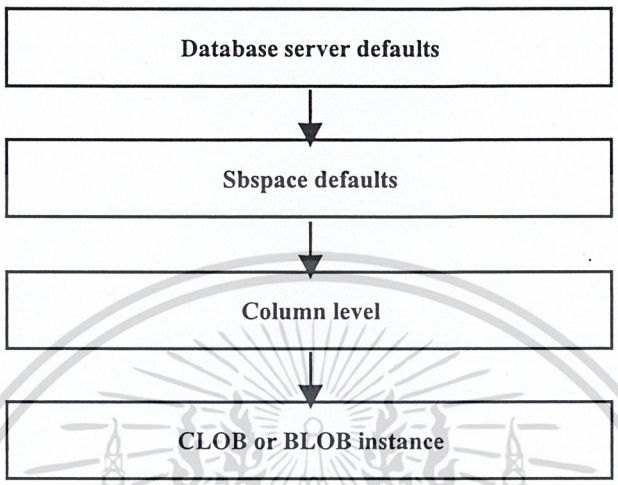
- สตอร์เรจ คาแรกเตอร์ริสติก (Storage Characteristic)

  จะประกอบด้วยข้อมูล

  - Disk-storage information เป็นข้อมูลที่สำหรับจัดการกับข้อมูลบนดิสก์ เช่น allocation extent information , sizing information , location information
  - Attribute information เป็นการกำหนดว่าต้องการให้มีออพชันอะไรบ้างในการเข้าถึงข้อมูลซึ่งประกอบด้วย logging indicator , last-access-time

ซึ่งในการระบุสตอร์เรจ คาแรกเตอร์ริสติกทำได้หลายระดับโดยสามารถกระทำได้ตอน

- สร้างเอสบี-สเปซโดยใช้ onspace utility
- สร้างตารางโดยใช้คีย์เวิร์ด PUT clause ในสเตทเมนท์ CREATE TABLE
- สร้างสมาร์ทลาร์จออบเจ็กต์โดยใช้ดาต้าเบลด เอพีไอ ฟังก์ชัน (DataBlade API function)



รูปที่ 6-3 แสดง Storage-Characteristics Hierarchy

- ข้อมูลสถานะ (Status Information)
  เป็นข้อมูลที่เกี่ยวกับสถานะต่างๆของสมาร์ทลาร์จออบเจ็กต์ที่จัดเก็บไว้ในส่วนของเมต้าดาต้าแอเรียของเอสบี-สเปซ

6.2.2 โล-แฮนด์เดิล (LO handle)

โล-แฮนด์เดิลเป็นข้อมูลที่มีโครงสร้างเป็นโอเปคซึ่งจะชี้ตำแหน่งข้อมูลของสมาร์ทลาร์จออบเจ็กต์ในเอสบีสเปซ สมาร์ทลาร์จออบเจ็กต์เป็นข้อมูลที่ใหญ่มากดังนั้นดาต้าเบสเซิร์ฟเวอร์จึงเก็บเฉพาะโล-แฮนด์เดิลในดาต้าเบสเท่านั้นเพื่อเป็นการลดขนาดของตารางให้เล็กที่สุด

6.3 การเข้าถึงสมาร์ทลาร์จออบเจ็กต์

การติดต่อหรือเข้าถึงข้อมูลที่เป็นสมาร์ทลาร์จออบเจ็กต์จะทำผ่านโล-แฮนด์เดิลได้ 2 แบบ คือ

1. การ SELECT คอลัมน์ซึ่งเก็บโล-แฮนด์เดิลอยู่ ซึ่งเราสามารถใช้แอนด์เดิลนี้ในการเข้าถึงข้อมูลของสมาร์ทลาร์จออบเจ็กต์ในเอสบี-สเปซ
2. การเก็บ (STORE) สมาร์ทลาร์จออบเจ็กต์ใหม่ เราสามารถสร้างโล-แฮนด์เดิลตัวใหม่ และเขียนข้อมูลลงเอสบี-สเปซและเก็บโล-แฮนด์เดิลลงในคอลัมน์

6.4 การเก็บสมาร์ทลาร์จออบเจ็กต์ในดาต้าเบส

ในดาต้าเบสเราสามารถเก็บสมาร์ทลาร์จออบเจ็กต์ในคอลัมน์ในตารางได้โดย

- การเข้าถึงข้อมูลในสมาร์ทลาร์จออบเจ็กต์โดยตรง โดยการสร้างคอลัมน์เป็นข้อมูลชนิด CLOB หรือ BLOB
- การซ่อนสมาร์ทลาร์จออบเจ็กต์ด้วยข้อมูลชนิดที่เป็นอะตอมมิก โดยการสร้างคอลัมน์เป็นข้อมูลชนิดโอเปคซึ่งเป็นสมาร์ทลาร์จออบเจ็กต์

6.5 CLOB

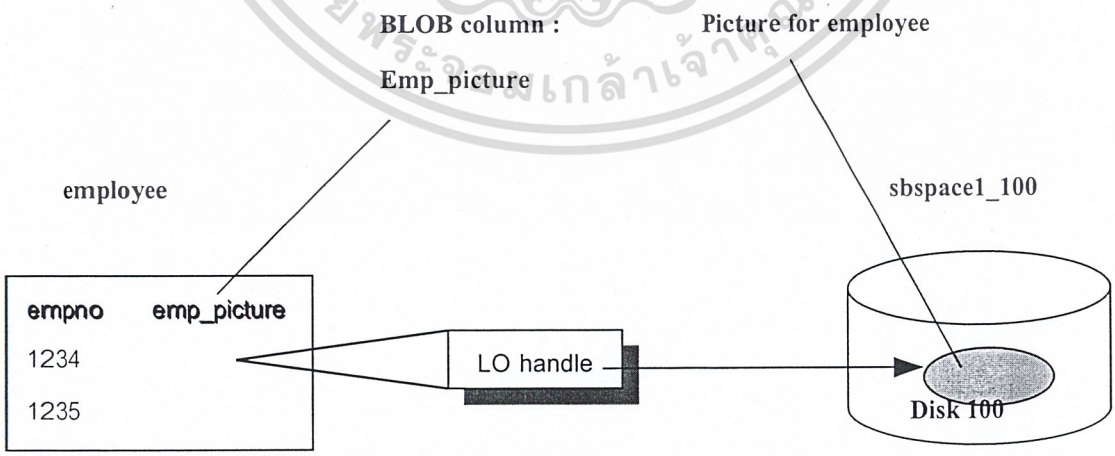
ข้อมูลชนิด CLOB ใช้เก็บข้อมูลที่เป็นบล็อคของเท็กซ์ (Block of TEXT) โดยปกติแล้ว CLOB ถูกใช้เก็บข้อมูลเท็กซ์แอสกี (ASCII text data) รวมไปถึงข้อมูลรูปแบบเท็กซ์ (Formatted text data) เช่น HTML หรือ PostScript แต่สำหรับอินฟอร์มิกซ์แล้วอนุญาตให้ CLOB เก็บได้เฉพาะข้อมูลที่สามารถเขียนเป็นเท็กซ์แอสกีได้เท่านั้น

ข้อดีของ CLOB

1. แอพพลิเคชันโปรแกรมสามารถอ่านหรือเขียนลงที่ส่วนใดของ CLOB ก็ได้
2. ผู้ใช้สามารถใช้เครื่องหมายเท่ากับ (=) ในการทดสอบว่าค่าของ CLOB 2 ค่า เท่ากันหรือไม่
3. ดาต้าเบลด เดเวลอปเปอร์ (DataBlade Developer) สามารถสร้างดัชนี (Index) จากข้อมูลชนิด CLOB ได้

6.6 BLOB

ข้อมูลชนิด BLOB ใช้เก็บข้อมูลใดๆ ก็ตามซึ่งโปรแกรมสามารถสร้างขึ้นมาได้ เช่น ภาพกราฟฟิก, ภาพทางดาวเทียม (Satellite images), วิดีโอ , เสียง หรือรูปแบบของเอกสาร ซึ่งคุณสมบัติของ BLOB เหมือนกันกับ CLOB ทุกประการ



รูปที่ 6-4 แสดง Smart Large Object ในคอลัมน์ในดาต้าเบส

6.7 การใช้สมาร์ทลาร์จออบเจ็กต์

1. เอสบี-สเปซ คือ logical storage unit ซึ่งผู้ใช้ต้องกำหนดขึ้นเพื่อใช้เก็บข้อมูลชนิด CLOB หรือ BLOB
2. คอลัมน์ที่เป็น CLOB หรือ BLOB ไม่สามารถปรากฏได้ในเอสคิวแอลสเตทเมนท์ ต่อไปนี้
   - นิพจน์ทางคณิตศาสตร์หรือบูลีน
   - Group By หรือ Order By clause
   - UNIQUE test
   - ประโยคนิยาม Informix index

อินฟอร์มิกซ์กำหนดฟังก์ชันต่อไปนี้ขึ้นมาเพื่อให้ผู้ใช้สามารถอิมพอร์ท (Import) และเอ็กซ์พอร์ท (Export) สมาร์ทลาร์จออบเจ็กต์โดยใช้เอสคิวแอลสเตทเมนท์ได้

Function Name	Purpose
FILETOBLOB()	ทำการคัดลอกไฟล์ลงในคอลัมน์ BLOB
FILETOCLOB()	ทำการคัดลอกไฟล์ลงในคอลัมน์ CLOB
LOCOPY()	ทำการคัดลอกข้อมูล BLOB หรือ CLOB ลงในคอลัมน์ BLOB หรือ CLOB อื่น
LOTOFILE()	ทำการคัดลอก BLOB หรือ CLOB ลงในไฟล์

*ตารางที่ 6-1 แสดงฟังก์ชันที่ใช่สำหรับข้อมูลชนิด **CLOB** หรือ **BLOB***

6.7.1 ตัวอย่างการสร้างสมาร์ทลาร์จออบเจ็กต์

```
CREATE TABLE inmate (
        Id_num  INT,
        Picture  BLOB,
        Felony  CLOB );

CREATE TABLE fbi_list (
        Id      INTEGER,
        MugshotBLOB  )  PUT mugshot IN (sbspace1);
```

จากตัวอย่างมีการให้ค่า SbspaceName parameter เป็นชื่อ “ sbspace1 “

6.7.2 ตัวอย่างการใส่ข้อมูลให้กับ CLOB-BLOB

```
INSERT INTO inmate
VALUES ( 437, FILETOBLOB( ' datafile ', ' client '),
          FILETOCLOB(' tmp/text ', ' server '))
```

การใส่ข้อมูลทำได้โดยการใช้ INSERT ร่วมกับฟังก์ชัน FILETOBLOB() และ FILETOCLOB() โดยอากิวเมนท์แรกของฟังก์ชันเป็นชื่อพาร์ท (path) ของไฟล์ต้นฉบับ (source file) ซึ่งจะถูกคัดลอกมาใส่ในคอลัมน์ BLOB และ CLOB ส่วนอากิวเมนท์ที่สองของฟังก์ชันใช้บอกถึงที่ตั้งของไฟล์ต้นฉบับ ว่าอยู่ในเครื่องไคล์เอนท์หรือเซิร์ฟเวอร์

6.7.3 ตัวอย่างการอัพเดท CLOB-BLOB

```
UPDATE inmate (picture)
   SET picture = LOCOPY( mugshot, 'fbi_list', 'mugshot' )
   WHERE innmate.id_num =  437 AND fbi_list.id = 669;
```

ข้อมูลในคอลัมน์ picture ของตาราง inmate ถูกเซตให้มีค่าเดียวกันกับค่าที่เก็บอยู่ในคอลัมน์ mugshot ของตาราง fbi_list

6.7.4 ตัวอย่างการคัดลอกข้อมูลจาก CLOB column ไปยังไฟล์

```
SELECT id_num, LOTOFILE( felony, 'felon_322.txt', 'client')
FROM   inmate
WHERE  id = 322
```

ข้อมูลบางส่วนของคอลัมน์ felony จะถูกคัดลอกมาไว้ยังไฟล์ชื่อ felon_322.txt ซึ่งอยู่ในเครื่องไคลเอนท์ ข้อมูลดังกล่าวเป็นส่วนหนึ่งของโรว์ซึ่งมี id เท่ากับ 322

6.8 โอเปคไทพ์ (OPAQUE TYPE)

ข้อมูลชนิดโอเปค คือ ข้อมูลที่เป็นอะตอมมิกซึ่งผู้ใช้สามารถกำหนดขึ้นมาให้กับดาต้าเบสโดยที่ดาต้าเบสจะไม่รู้จักรูปแบบของข้อมูลชนิดนี้มาก่อน เมื่อโอเปคไทพ์ถูกกำหนดขึ้นมาแล้ว จะสามารถถูกนำมาใช้ได้ในลักษณะเดียวกันกับบิวท์อินดาต้าไทพ์ ก่อนที่จะกำหนดโอเปคไทพ์ให้กับดาต้าเบสผู้ใช้จะต้องสร้างข้อมูลต่อไปนี้ขึ้นมาโดยใช้ภาษาซี

1. โครงสร้างข้อมูล (Data Structure) เปรียบเสมือนที่เก็บข้อมูลภายในของข้อมูลชนิดโอเปค
2. ซัพพอร์ทฟังก์ชัน (Support Functions) เป็นฟังก์ชันที่ดาตดาเบสใช้เข้าถึงโครงสร้างภายในของข้อมูลชนิดโอเปค
3. รูทีนที่เพิ่มเข้ามา (Optional Additional Routines) จะถูกเรียกใช้โดยซัพพอร์ทฟังก์ชันหรือเอ็นด์ยูสเซอร์เพื่อจัดการกับข้อมูลชนิดโอเปค

6.8.1 โครงสร้างภายใน (Internal Structure)

เป็นสิ่งซึ่งบ่งบอกว่าข้อมูลชนิดโอเปคถูกเก็บอยู่บนดิสก์อย่างไร ผู้ใช้สามารถกำหนดโครงสร้างภายในได้โดยใช้ภาษาซี เพื่อซัพพอร์ทโอเปคไทพ์ 2 แบบ คือ

1. โอเปคไทพ์ชนิดความยาวคงที่ (Fixed-length Opaque Type) ขนาดของโครงสร้างภายในถูกกำหนดไว้ตายตัว สามารถใช้โอเปคไทพ์ชนิดนี้เก็บค่าตัวเลข เป็นต้น
2. โอเปคไทพ์ชนิดความยาวไม่คงที่ (Varying-length Opaque Type) ขนาดของโครงสร้างภายในเปลี่ยนแปลงไปตามค่าของโอเปคไทพ์

ตามปกติแล้วโอเปคไทพ์ชนิดนี้จะถูกใช้สำหรับข้อมูลประเภทรูปภาพ,วิดีโอ,เสียง เป็นต้น จากข้อมูลที่ยกตัวอย่างมานั้นจะเห็นว่าขนาดของโอเปคเปลี่ยนแปลงตามขนาดของรูปภาพแต่ละรูป

สังเกตุ : ถ้าต้องการให้อินฟอร์มิกซ์รู้จักโครงสร้างภายในซึ่งสร้างขึ้นด้วยภาษาซี จะต้องใส่สตริง "_t" ต่อท้ายเข้าไปที่ชื่อ ตัวอย่างเช่น ข้อมูลชนิดโอเปค "circle" จะอ้างถึงโครงสร้างภายในชื่อ "circle_t"

ตัวอย่างการสร้างโครงสร้างภายในแบบความยาวคงที่

```
TYPEDEF STRUCT {                    TYPE STRUCT {
        Double  x;                          Point_t  center;
        Double  y;                          Double Radius;
              } point_t;                                  } circle_t;
```

โครงสร้างภายในนี้เป็นโครงสร้างของข้อมูลชนิดโอเปค **circle** ซึ่งมี X , Y แทนจุดศูนย์กลางของวงกลมและมี radius แทนรัศมีของวงกลม โดยทั้งหมดมีค่าเป็น double value และเนื่องจาก double value มีขนาด 8 ไบท์ ดังนั้นขนาดของ circle structure จึงเท่ากับ 24 ไบท์

6.8.2 ซัพพอร์ทฟังก์ชัน (Support Function)

เป็นฟังก์ชันที่ใช้จัดการกับโครงสร้างภายในของโอเปคไทพ์ซึ่งเป็นเสมือนตัวเชื่อมให้ดาต้าเบส รู้จักโอเปคไทพ์ที่กำหนดขึ้น ดาต้าเบสเซิร์ฟเวอร์จะเรียก ซัพพอร์ทฟังก์ชันเหล่านี้เมื่อต้องการจัดการกับโอเปคไทพ์

6.8.3 การรีจิสเตอร์โอเปคไทพ์ลงในดาต้าเบส

ในการรีจิสเตอร์โอเปคไทพ์ลงในดาต้าเบสต้องใช้เอสคิวแอล สเตทเมนท์ต่อไปนี้

6.8.3.1 CREATE OPAQUE TYPE เป็นประโยคประกาศว่าเป็นโอเปคไทพ์ในประโยคนี้ต้องมีส่วนประกอบเพิ่มเติมเพื่อใช้บอกข้อมูลอื่นของโครงสร้างภายในอีก ดังนี้

Final Structure Size

ใช้คีย์เวิร์ด "INTERNALLENGTH" สำหรับบอกขนาดของโครงสร้างภายในได้ 2 ลักษณะดังนี้

- CREATE OPAQUE TYPE var_type **(INTERNALLENGTH=VARIABLE , MAXLEN=4096);**

คำว่า VARIABLE บ่งบอกว่าโครงสร้างภายในของโอเปคไทพ์ชื่อ var_type มีขนาดไม่แน่นอน โดยจะมีขนาดสูงสุดได้ไม่เกิน 4096 ไบท์ ซึ่งเป็นค่าซึ่งกำหนดให้กับคีย์เวิร์ด “MAXLEN”

- CREATE OPAQUE TYPE circle **(INTERNALLENGTH=24);**

  แสดงว่าโครงสร้างภายในมีขนาดเท่ากับ 24 ไบท์

Memory Alignment

สามารถใช้คีย์เวิร์ด “ALIGNMENT” กำหนดขนาดของ Memory alignment ได้ โดยค่าที่สามารถกำหนดให้กับ ALIGNMENT ได้มี 4 ค่าดังนี้ 1, 2, 4, และ 8 ซึ่งค่าเหล่านี้มีความหมายเฉพาะสำหรับแต่ละค่า

ตัวอย่างประโยค CREATE OPAQUE TYPE longlong(INTERNALLENGTH=18,ALIGNMENT=1);

ถ้าหากไม่มีการใช้ keyword ALIGNMENT ในประโยค ให้ถือว่า default alignment คือ 4-byte

การส่งผ่านพารามิเตอร์ (Parameter Passing)

ยูนิเวอร์แซล เซิร์ฟเวอร์กำหนดวิธีการผ่านค่าของโอเปคไทพ์ให้กับยูสเซอร์ดีฟายด์รูทีนไว้ 2 แบบดังนี้

- การส่งผ่านค่า (Pass by value) เป็นการผ่านค่าจริงของโอเปคไทพ์ให้กับยูสเซอร์ดีฟายด์รูทีน
- การส่งผ่านแบบอ้างถึง (Pass by reference) เป็นการผ่านค่าพอยท์เตอร์ซึ่งชี้ที่ค่าของโอเปคไทพ์ให้กับยูสเซอร์ดีฟายด์รูทีน

ตัวอย่างประโยค CREATE OPAQUE TYPE two_bytes (INTERNALLENGTH=18 ,PASSEDBYVALUE);

ถ้าไม่มีการกำหนดวิธีในการผ่านค่า จะถือว่าเป็นการผ่านค่าแบบอ้างถึง

6.8.3.2 CREATE FUNCTION ใช้ประกาศซัพพอร์ทของโอเปคไทพ์มีรูปแบบดังนี้

CREATE FUNCTION ***func_name*** (***parameter_list***)
RETURN ***ret_type***
EXTERNAL NAME ‘***pathname***’
LANGUAGE C NOT VARIANT

โดย ***func_name*** เป็นชื่อของฟังก์ชัน
parameter_list บอกชนิดข้อมูลของพารามิเตอร์
ret_type ชนิดของค่าที่ส่งกลับของฟังก์ชัน
pathname ที่ตั้งของซอสโค้ดสำหรับซัพพอร์ทฟังก์ชัน

6.8.3.3 CREATE CAST ใช้เพื่อประกาศว่าซัพพอร์ทฟังก์ชันใด ทำหน้าที่เป็นแคสท์ฟังก์ชันบ้าง

ตัวอย่างประโยค CREATE CAST (circle AS LVARCHAR WITH circle_out)

เป็นการประกาศว่าฟังก์ชันชื่อ circle_out ทำหน้าที่แปลงข้อมูลชนิด circle ไปเป็นข้อมูลชนิด LVARCHAR

บทที่ 7
การสร้างและออกแบบ

โครงงานนี้เป็นการสร้างแอพพลิเคชันเพื่อสร้างดาต้าโฟลว์ไดอะแกรมเมอร์และเก็บความสัมพันธ์ของคอมโพเนนท์ต่างๆ ลงในระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ของอินฟอร์มิกซ์ ยูนิเวอร์แซลเซิร์ฟเวอร์ ซึ่งจะต้องศึกษาถึงคอมโพเนนท์และความสัมพันธ์ต่างๆของดาต้าโฟลว์ไดอะแกรม และทำการออกแบบความสัมพันธ์โดยใช้ ER-Diagram ในการออกแบบแล้วทำการแม็ปเป็นตารางความสัมพันธ์และสร้างแอพพลิเคชันในการติดต่อกับผู้ใช้งานผ่านระบบฐานข้อมูล

7.1 ดาต้าโฟลว์ไดอะแกรมเมอร์ (Dataflow Diagrammer)

ดาต้าโฟลว์ไดอะแกรมเมอร์เป็นแอพพลิเคชันที่สร้างขึ้นมาเพื่อใช้สำหรับการสร้างและปรับปรุงฟังก์ชันในระบบงาน ดาต้าสตอร์ (Data Store) ดาต้าโฟลว์ (Data Flow) และสิ่งภายนอกอื่นๆ(External Entity) ที่ถูกเก็บอยู่ในรีโพซิทอรี่ (Repository) ดาต้าโฟลว์ไดอะแกรมจะแสดงให้เห็นว่าข้อมูลมีการไหลอย่างไรในระบบงาน นอกจากนี้ดาต้าโฟลว์ไดอะแกรมยังใช้ในการแสดงความสัมพันธ์ของข้อมูลและองค์ประกอบอื่นๆของระบบอีกด้วย

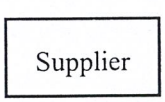
การใช้ดาต้าโฟลว์ไดอะแกรมเป็นเทคนิคอย่างหนึ่งสำหรับการโมเดลระบบ โดยจะแสดงให้เห็นถึงการไหลของข้อมูลระหว่างฟังก์ชัน (Function) หรือโพรเซส (Process) ซึ่งดาต้าโฟลว์ไดอะแกรมนี้สามารถนำมาใช้ได้หลายวิธี

- ใช้เพื่อแสดงให้เห็นถึงขอบเขตของระบบแอพพลิเคชัน (Application)
- ใช้เพื่อออกแบบและโมเดลความสัมพันธ์ระหว่างฟังก์ชันหรือโพรเซส
- ใช้ในการสร้างรูปภาพที่มีรายละเอียดของข้อมูลที่ถูกใช้โดยฟังก์ชันหรือโพรเซส

7.1.1 องค์ประกอบของดาต้าโฟลว์ไดอะแกรม

7.1.1.1 สิ่งภายนอก (External Entity) หรือ เอ็กซ์เทอร์นอลเอนทิตี้

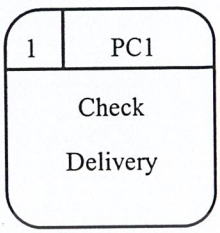
เอ็กซ์เทอร์นอลเอนทิตี้ใช้แทนสิ่งที่อยู่ภายนอกระบบ อาจจะเป็นได้ทั้งคน องค์กร หรือระบบคอมพิวเตอร์อื่นๆ ที่เป็นแหล่งที่มาหรือเป็นผู้รับข้อมูลจากระบบหรือขอบเขตที่ทำการศึกษา ชื่อของเอ็กซ์เทอร์นอลเอนทิตี้ จะหมายถึง ชนิดของเอนทิตี้เท่านั้น ไม่ใช่เหตุการณ์ของเอนทิตี้ เช่น Supplier จะหมายถึงสิ่งภายนอกที่เป็นผู้รับ Purchase Orders



รูปที่ 7-1 แสดงสัญลักษณ์ของเอ็กซ์เทอร์นอลเอนทิตี้

7.1.1.2 โพรเซส (Process)

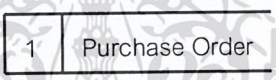
โพรเซสใช้แสดงการกระทำต่อข้อมูล ซึ่งอาจจะกระทำโดยคนหรือคอมพิวเตอร์ก็ได้เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ ชื่อของโพรเซสจะเป็นคำที่ใช้แสดงการกระทำอย่างสั้นๆ และมีความหมายครอบคลุมการกระทำนั้น เช่น Check Delivery , Check Stock เป็นต้น



รูปที่ 7-2 แสดงสัญลักษณ์ของโพรเซส

7.1.1.3 ดาต้าสตอร์ (Data Store)

ดาต้าสตอร์ หมายถึง แหล่งเก็บข้อมูลชั่วคราวหรือถาวร เช่น Purchase Orders เป็นที่เก็บข้อมูลซึ่งประกอบด้วยรายการของใบสั่งซื้อสินค้าจำนวนมากเพื่อส่งไปยัง Suppliers



รูปที่ 7-3 แสดงสัญลักษณ์ของดาต้าสตอร์

7.1.1.4 ดาต้าโฟลว์ (Data Flow)

ดาต้าโฟลว์เป็นลูกศรที่แสดงถึงการไหลของข้อมูลทั้งเข้าและออกของระบบ ซึ่งเป็นสัญลักษณ์แสดงอินพุทและเอาท์พุทของโพรเซสและดาต้าสตอร์ในระบบ และข้อมูลที่ผ่านเข้ามาหรือไหลออกนอกระบบ

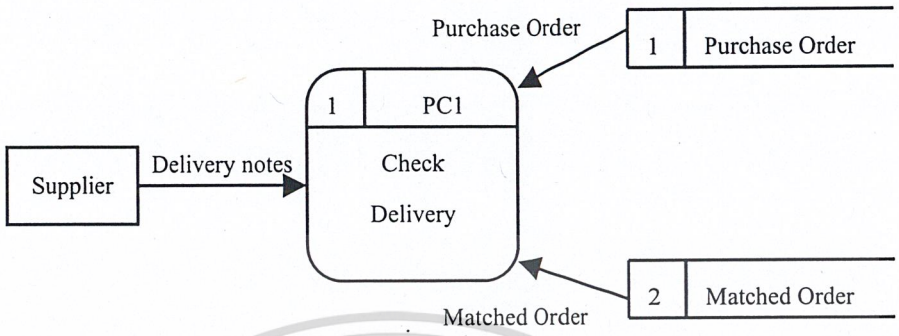
รูปที่ 7-4 แสดงสัญลักษณ์ของดาต้าโฟลว์

ดาต้าโฟลว์สามารถใช้แสดงการไหลของข้อมูลระหว่างสิ่งต่อไปนี้

- โพรเซส 2 โพรเซส
- ดาต้าสตอร์ และ โพรเซส
- โพรเซส และ เอ็กซ์เทอร์นอลเอนทิตี้

เราไม่สามารถใช้ดาต้าโฟลว์เชื่อมต่อระหว่างดาต้าสตอร์ กับ ดาต้าสตอร์ หรือ ระหว่างเอ็กซ์เทอร์นอลเอนทิตี้ กับ เอ็กซ์เทอร์นอลเอนทิตี้ หรือ ระหว่างดาต้าสตอร์กับเอ็กซ์เทอร์นอลเอนทิตี้ ได้ เพราะเป็นการเชื่อมต่อระหว่างข้อมูลซึ่งไม่ผ่านการกระทำใดๆ เลย ดังตัวอย่าง

1. Supplier ส่ง Delivery notes ไปยัง Process Goods-In
2. Process Goods-In จะทำการเช็คข้อมูลกับ Purchase Order ที่รับเข้ามา
3. เพื่อเช็คข้อมูลแล้วจะนำข้อมูลที่ตรงกันไปเก็บที่ไฟล์ชั่วคราว



รูปที่ 7-5 แสดงตัวอย่างดาต้าโฟลว์ไดอะแกรม

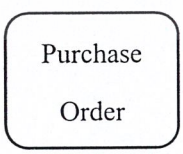
7.2 เอนทิตี้รีเลชันชิพไดอะแกรม (Entity Relationship Diagram)

เอนทิตี้รีเลชันชิพไดอะแกรม หรือ ไดอะแกรมแสดงความสัมพันธ์ระหว่างเอนทิตี้เป็นเครื่องมือที่ใช้ในการสร้างแบบจำลองสำหรับการกำหนดถึงข้อมูลที่ต้องการในรูปของเอนทิตี้รีเลชันชิพโมเดล (Entity Relationship Model) การสร้างเอนทิตี้รีเลชันชิพโมเดลนี้จะเกี่ยวข้องกับการระบุส่วนประกอบที่สำคัญในระบบงาน (Entity) และความสัมพันธ์กันของแต่ละส่วนประกอบ (Relationship) โมเดลที่ได้นี้จะไม่ขึ้นกับที่เก็บข้อมูลหรือวิธีการเข้าถึงข้อมูลเลย

องค์ประกอบของไดอะแกรมแสดงความสัมพันธ์ระหว่างเอนทิตี้ ประกอบด้วย

7.2.1 เอนทิตี้ (Entity)

เอนทิตี้ คือ สิ่งที่เราสนใจในระบบหรือองค์กร อาจจะเป็นสิ่งที่มีอยู่หรือสิ่งที่เราสมมติขึ้นมาก็ได้ จะเป็นสิ่งที่บรรจุหรือเกี่ยวข้องกับข้อมูลด้านต่างๆ ที่จำเป็นต้องรู้ โดยจะแสดงอยู่ในสัญลักษณ์สี่เหลี่ยมขอบมน ดังรูป ซึ่งเอนทิตี้แต่ละเอนทิตี้ไม่สามารถคงอยู่โดยไม่มีความสัมพันธ์ได้

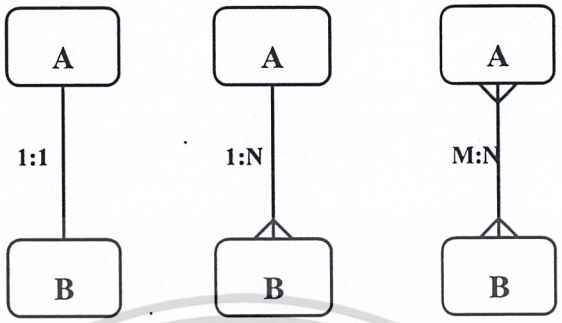


รูปที่ 7-6 แสดงสัญลักษณ์เอนทิตี้

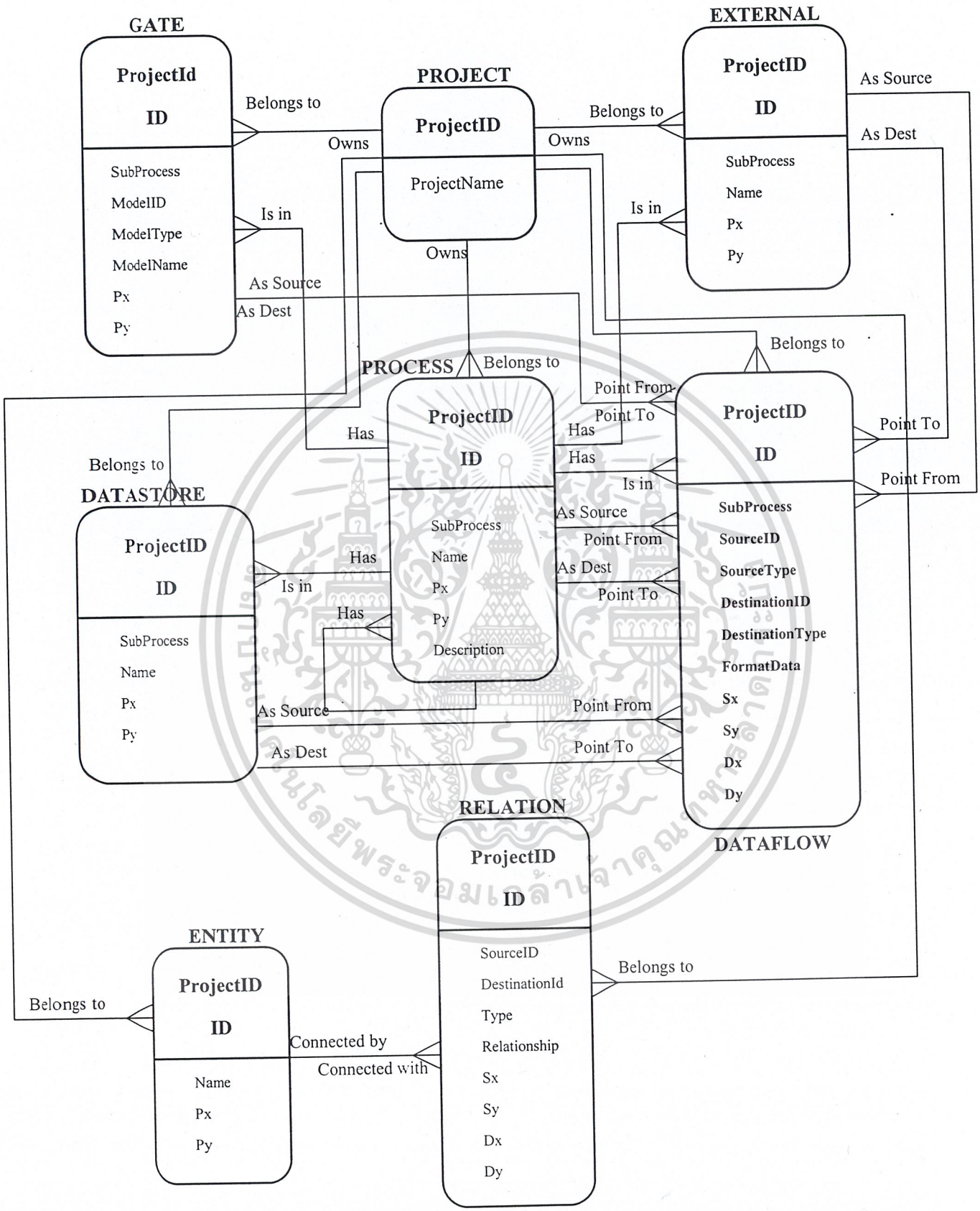
7.2.2 ความสัมพันธ์ (Relationship)

ความสัมพันธ์ เป็นสิ่งที่ใช้เชื่อมโยงระหว่างเอนทิตี้ 2 เอนทิตี้ เพื่อแสดงความสัมพันธ์ระหว่างเอนทิตี้ทั้งสอง ซึ่งแบ่งความสัมพันธ์ออกเป็น 3 ลักษณะ คือ

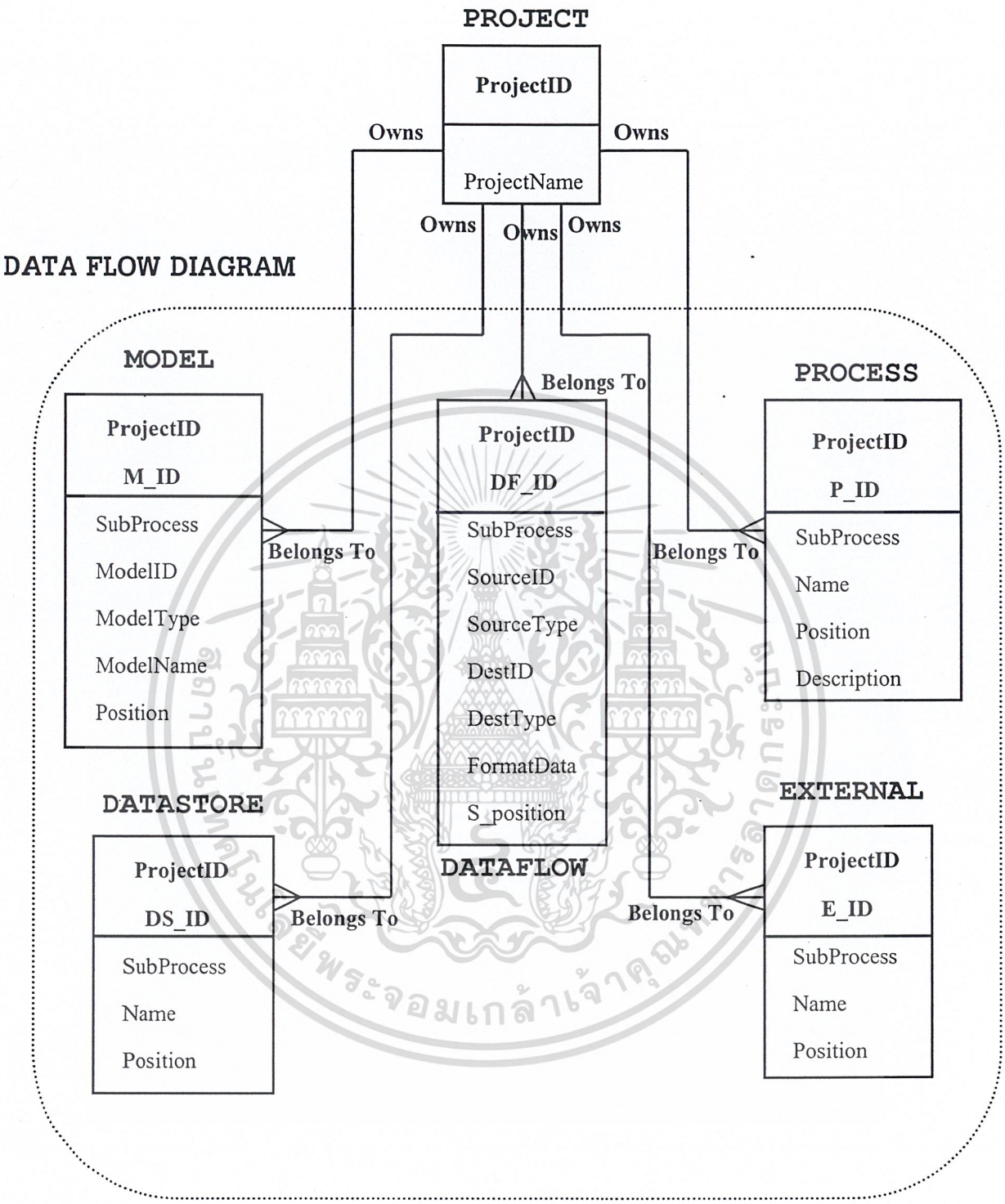
- ความสัมพันธ์แบบหนึ่งต่อหนึ่ง
- ความสัมพันธ์แบบหนึ่งต่อกลุ่ม
- ความสัมพันธ์แบบกลุ่มต่อกลุ่ม



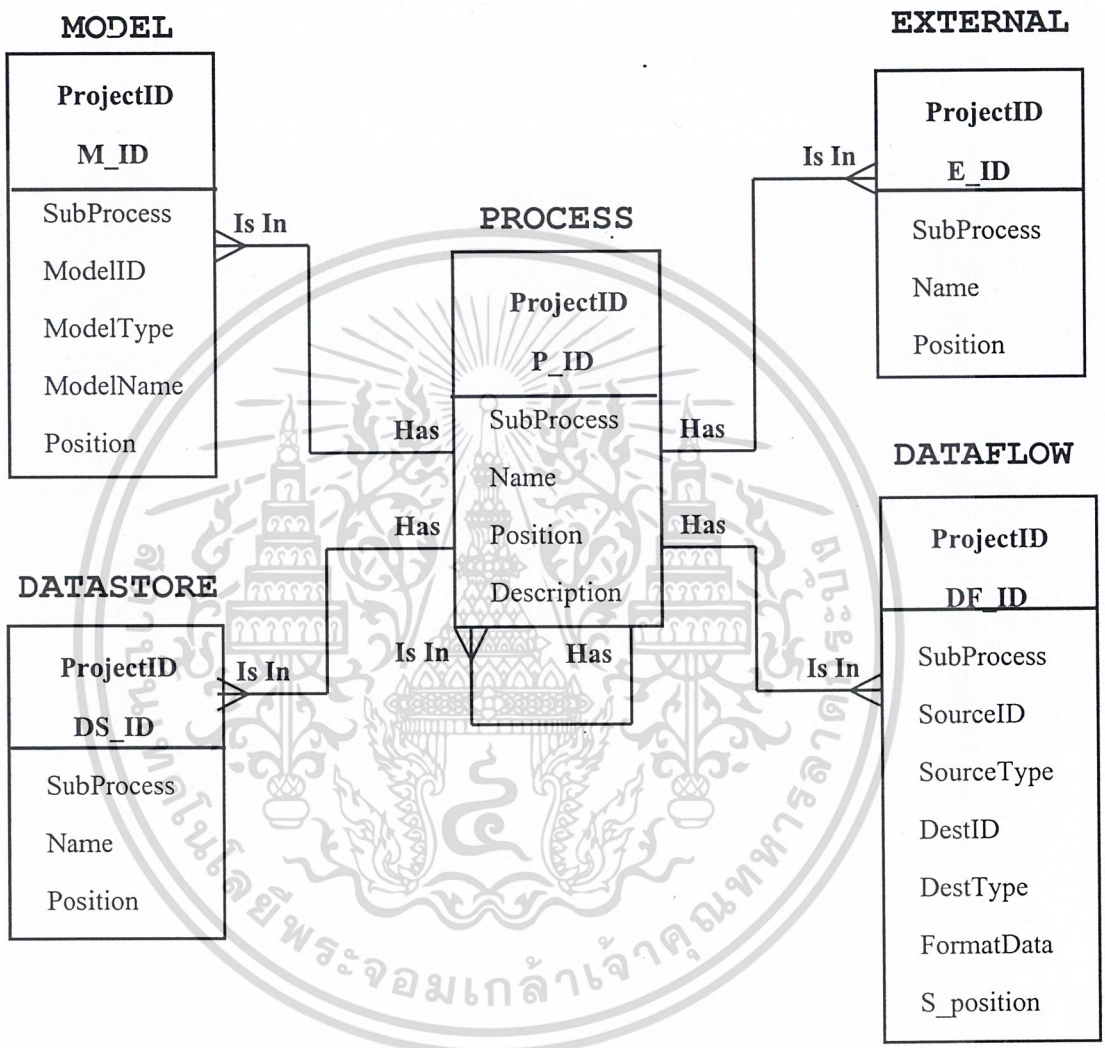
รูปที่ 7-7 แสดงความสัมพันธ์ระหว่างเอนทิตี้ในลักษณะต่างๆ



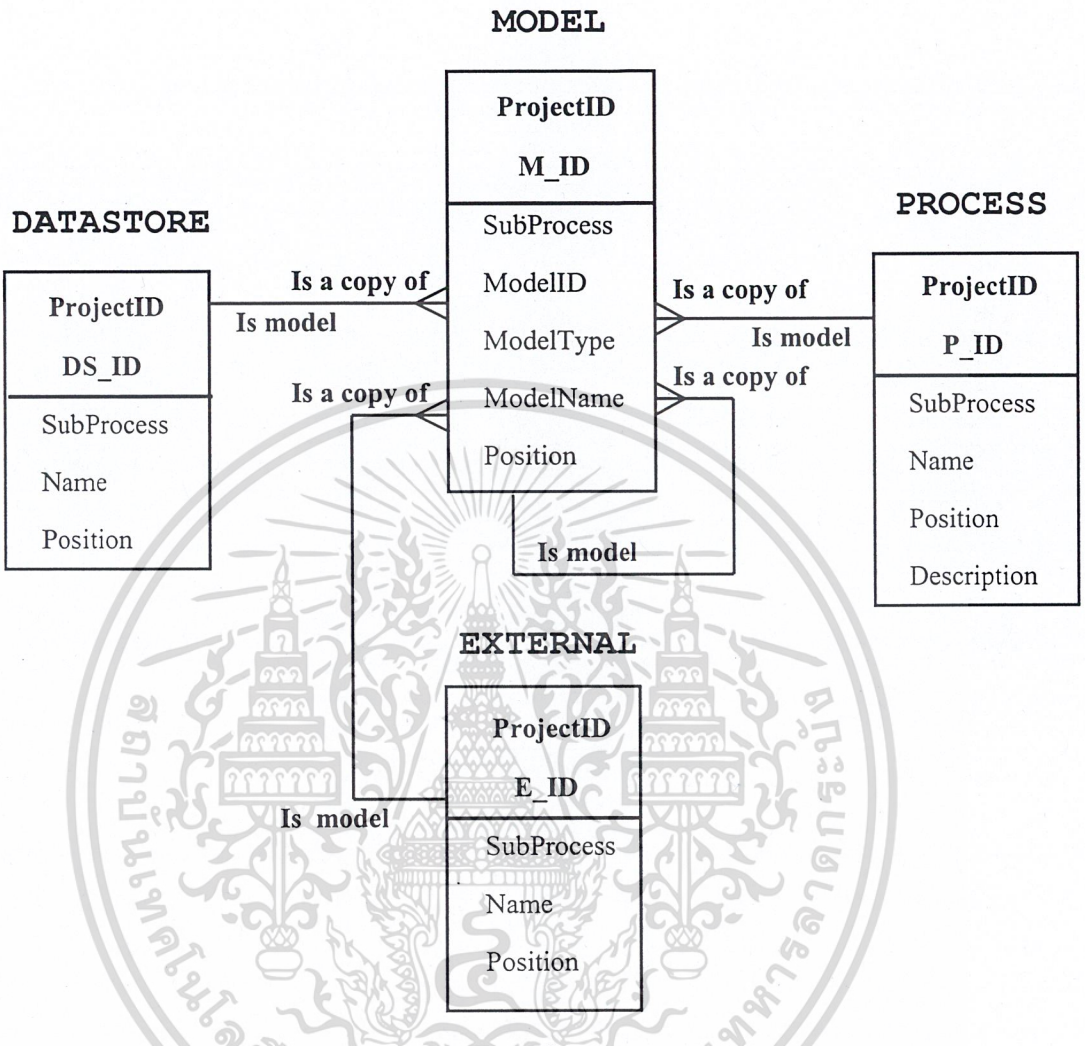
รูปที่ 7-8 แสดง ER-Diagram ที่ได้จากการออกแบบความสัมพันธ์ของคอมโพเนนท์ต่างๆ



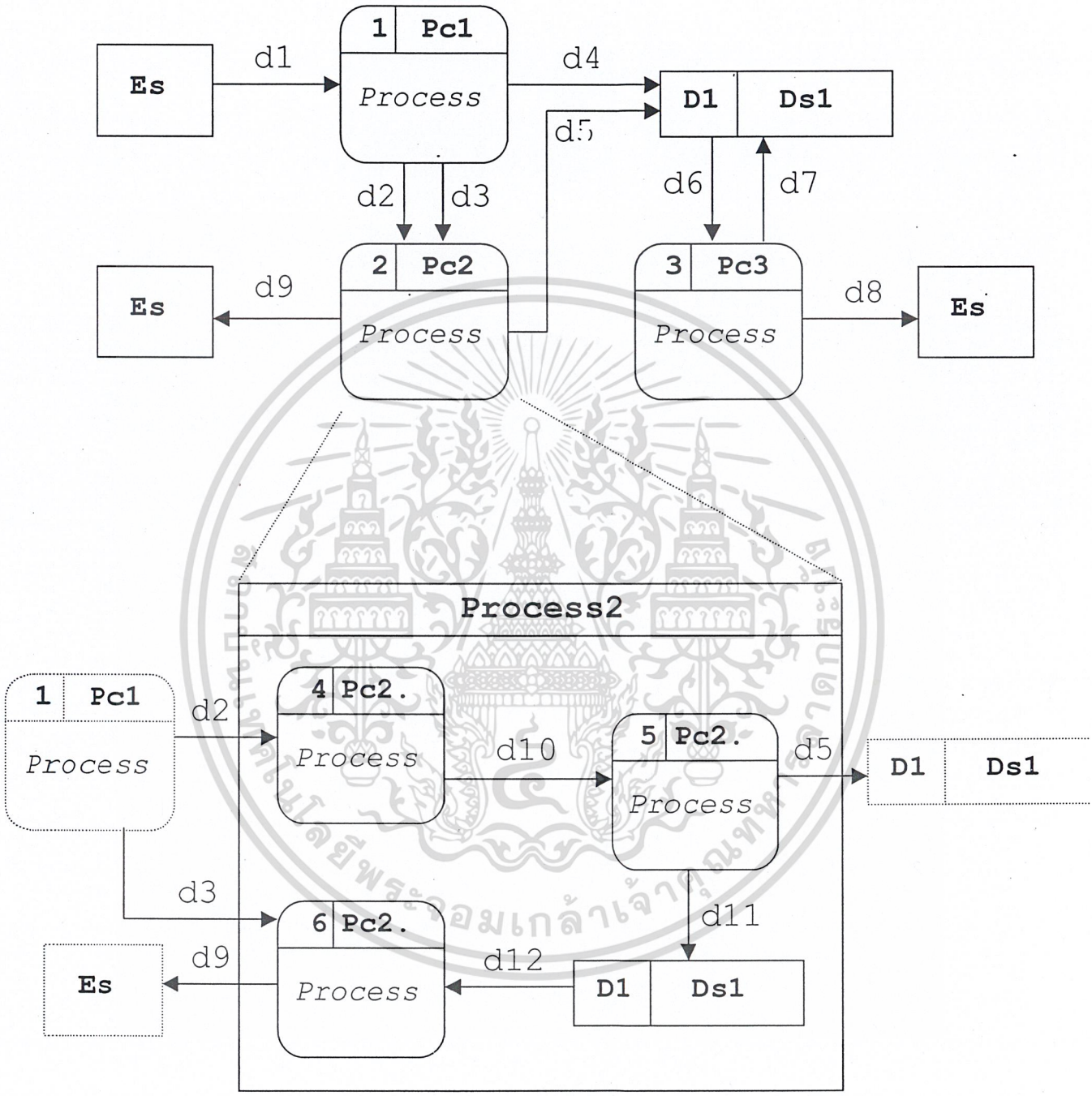
รูปที่ 7-9 แสดงความสัมพันธ์ระหว่างโปรเจคกับคอมโพเนนท์ต่างๆของดาต้าโฟลว์ไดอะแกรม



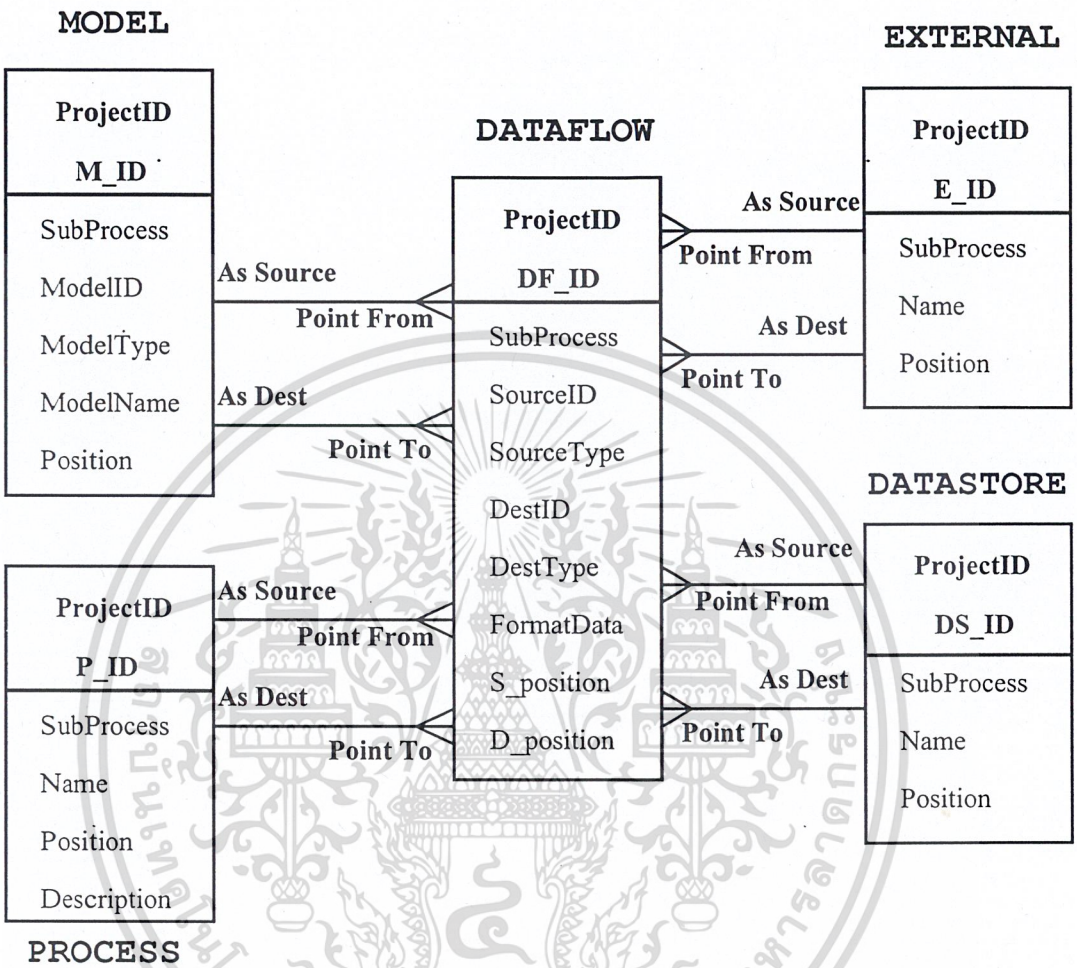
รูปที่ 7-10 แสดงความสัมพันธ์ระหว่างคอมโพเนนท์โพรเซสกับคอมโพเนนท์อื่นๆ



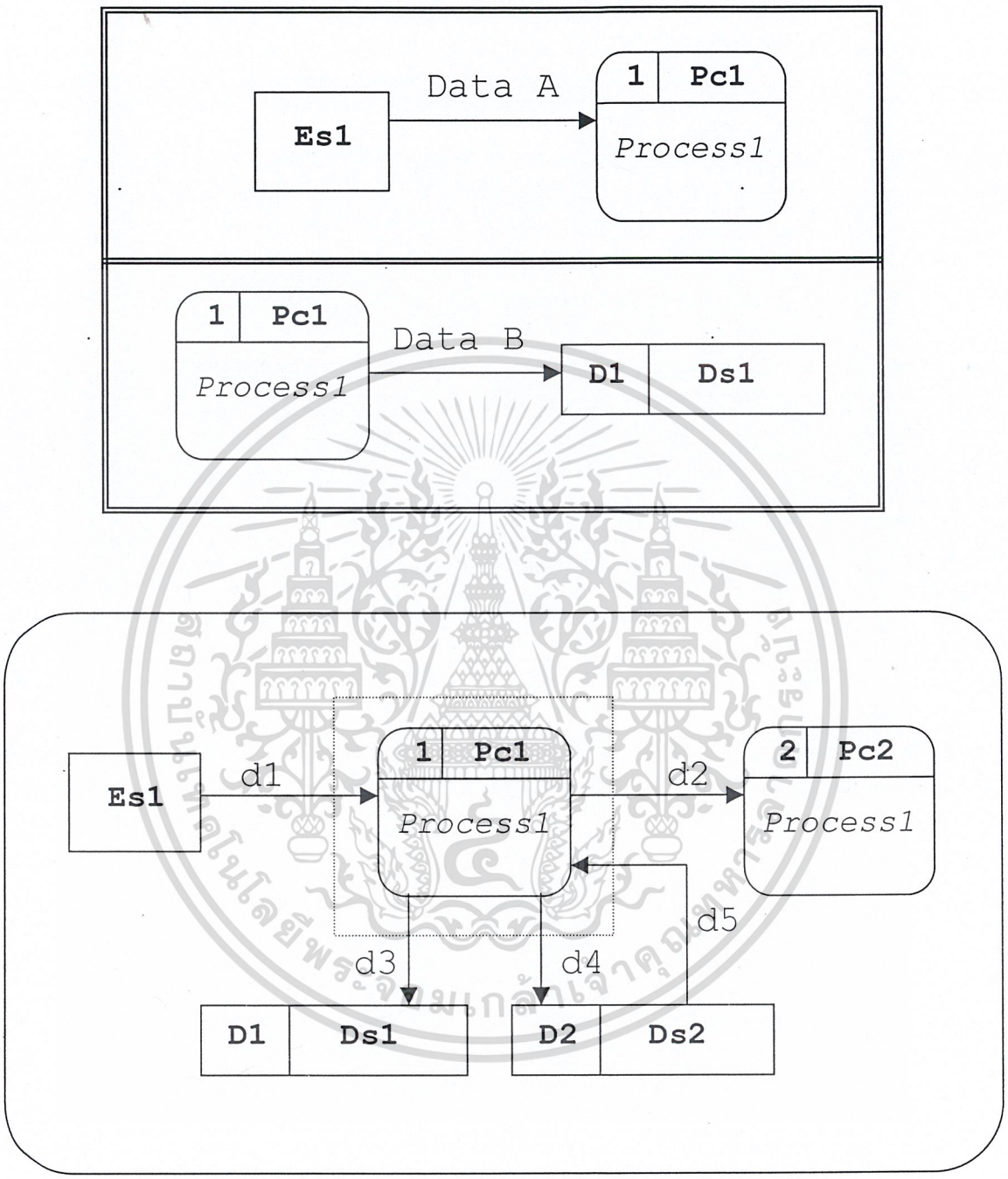
รูปที่ 7-11 แสดงความสัมพันธ์ระหว่างคอมโพเนนท์โมเดลกับคอมโพเนนท์อื่นๆ



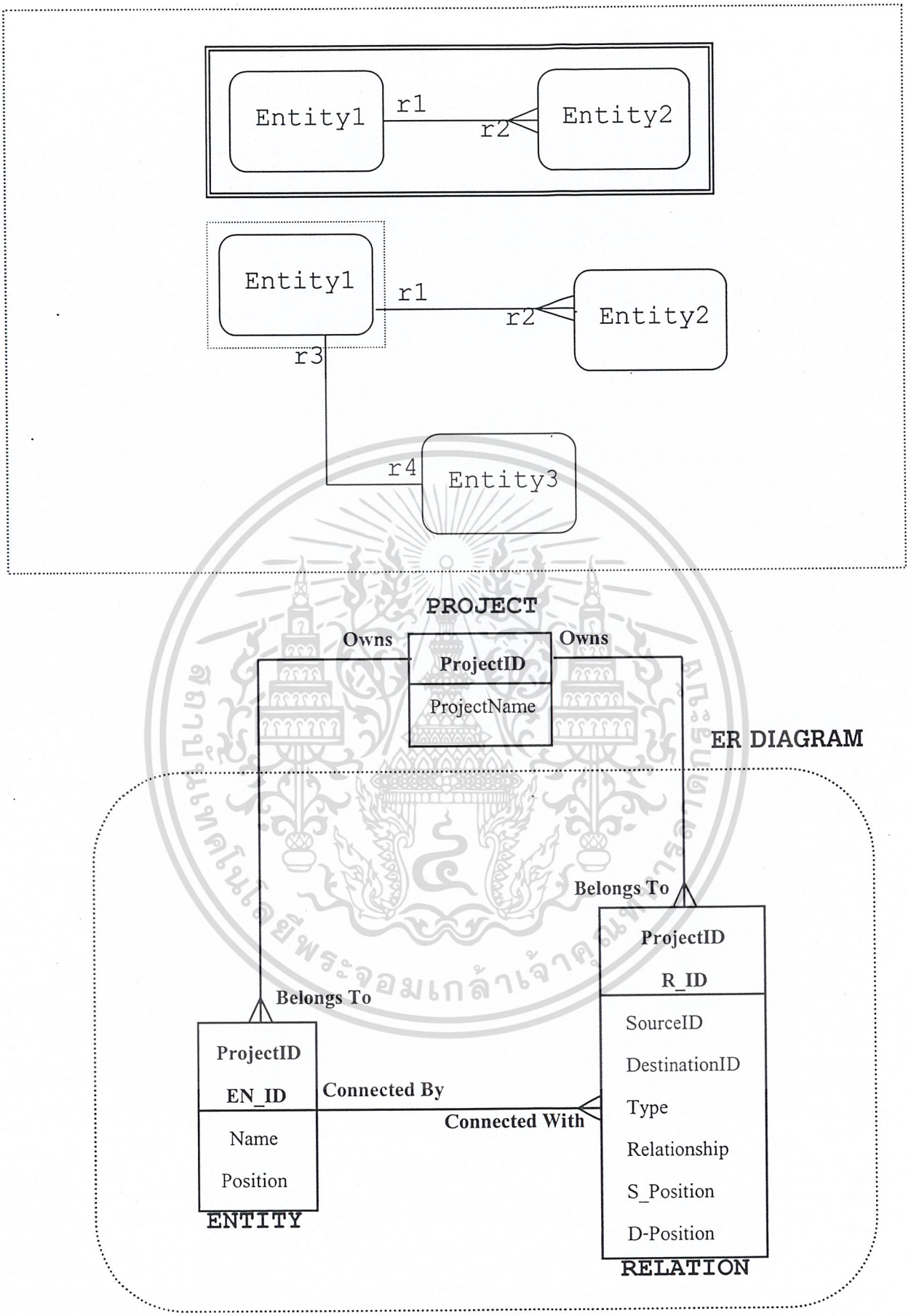
รูปที่ 7-12 แสดงตัวอย่างแสดงความสัมพันธ์ระหว่างคอมโพเนนท์โมเดลกับคอมโพเนนท์อื่นๆ



รูปที่ 7-13 แสดงความสัมพันธ์ระหว่างคอมโพเนนท์ดาต้าโฟลว์กับคอมโพเนนท์อื่นๆ



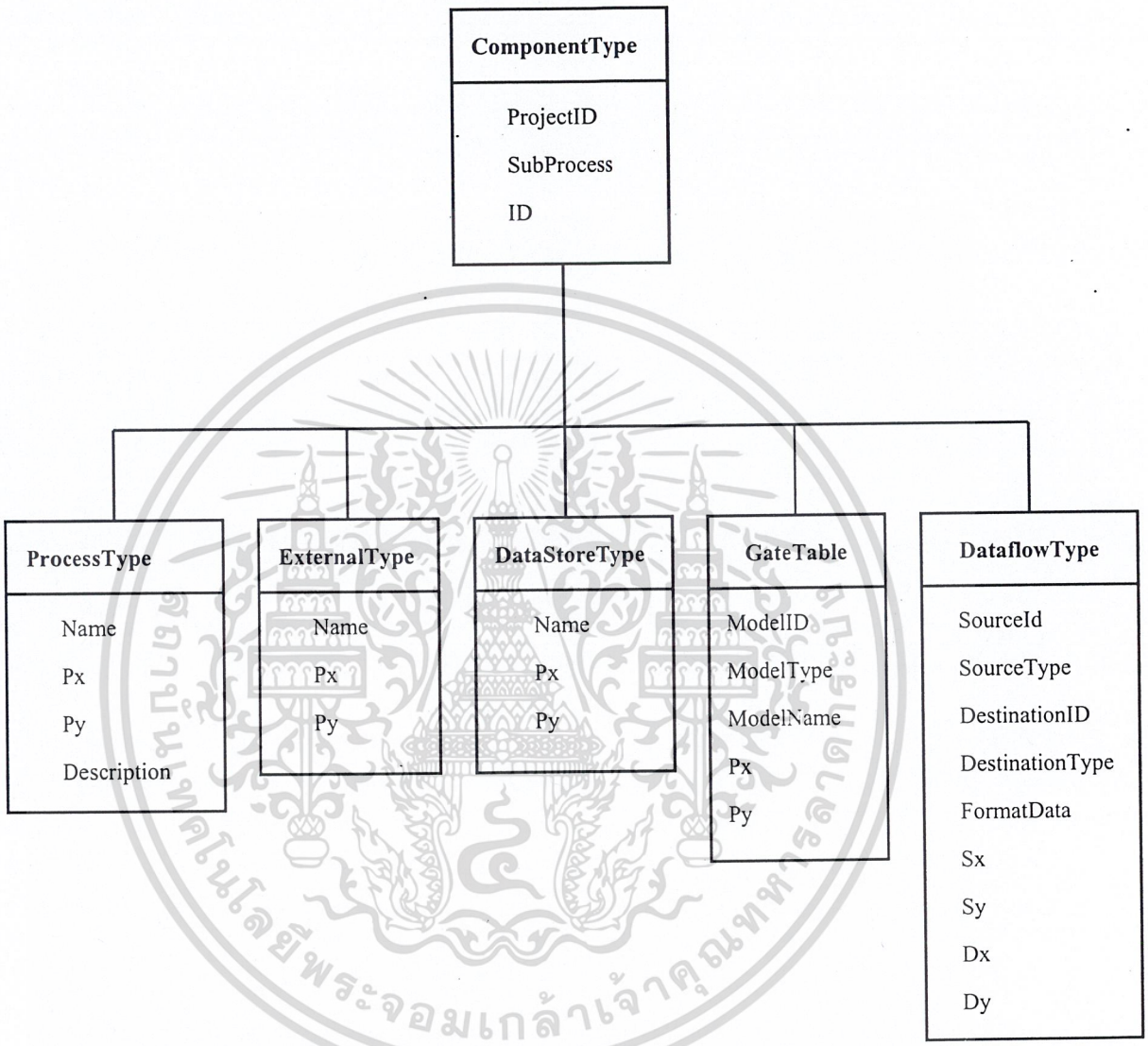
รูปที่ 7-14 แสดงตัวอย่างแสดงความสัมพันธ์ระหว่างคอมโพเนนท์ดาต้าโฟลว์กับคอมโพเนนท์อื่นๆ



รูปที่ 7-15 แสดงความสัมพันธ์ระหว่างคอมโพเนนท์ต่างๆของ ER Diagram

7.3 การออกแบบโรว์ไทพ์และตาราง

โรว์ไทพ์ที่ใช้สำหรับเก็บคอมโพเนนท์ของดาต้าโฟลว์ไดอะแกรมทุกๆไทพ์ (ProcessType , ExternalType , DatastoreType , GateType , DataType) จะอินเฮอร์ริทมาจากคอมโพเนนท์ไทพ์ซึ่งมีแอททริบิวต์เป็น ProjectID , SubProcess และ ID ดังรูป



รูปที่ 7-16 แสดงการอินเฮอร์ริทคอมโพเนนท์ไทพ์ของคอมโพเนนท์ต่างในดาต้าโฟลว์ไดอะแกรม

ซึ่งสามารถเขียนเป็นเอสคิวแอสสเตทเมนท์ได้ดังนี้

```
CREATE ROW TYPE  componentType
(       projectID       integer,
        subProcess      integer,
        ID              integer  );
```

แอททริบิวท์ projectID สำหรับเก็บ ID ของโปรเจคของดาต้าโฟลว์ไดอะแกรม

แอททริบิวท์ subprocess สำหรับเก็บ ID ของโพรเซสที่ตัวมันเองเป็นซับโพรเซสอยู่

แอททริบิวท์ ID สำหรับเก็บ ID ของแต่ละคอมโพเนนท์

```
CREATE ROW TYPE  processType
(       name            char(15),
        Px              Integer,
        Py              Integer,
        description     lvarchar
) UNDER  componentType;
```

แอททริบิวท์ name สำหรับเก็บชื่อของคอมโพเนนท์โพรเซส

แอททริบิวท์ Px และ Py สำหรับเก็บตำแหน่งของคอมโพเนนท์โพรเซส

```
CREATE ROW TYPE  externalType
(       name            char(15),
        Px              Integer,
        Py              Integer
) UNDER  componentType;
```

แอททริบิวท์ name สำหรับเก็บชื่อของคอมโพเนนท์เอ็กซ์เทอร์นอลเอนทิตี้

แอททริบิวท์ Px และ Py สำหรับเก็บตำแหน่งของคอมโพเนนท์เอ็กซ์เทอร์นอลเอนทิตี้

```
CREATE ROW TYPE  datastoreType
(       name            char(15),
        Px              Integer,
        Py              Integer
) UNDER componentType;
```

แอททริบิวท์ name สำหรับเก็บชื่อของคอมโพเนนท์ดาต้าสตอร์

แอททริบิวท์ Px และ Py สำหรับเก็บตำแหน่งของคอมโพเนนท์ดาต้าสตอร์

```
CREATE ROW TYPE  dataflowType
(       SourceID        integer,
        SourceType      char(2),
        DestinationID   integer,
        DestinationType char(2),
        formatData      char(20),
        Sx              Integer,
```

Sy Integer,

Dx Integer,

Dy Integer

) UNDER componentType;

แอททริบิวท์ sourceID สำหรับเก็บ ID ของคอมโพเนนท์ที่เป็นข้อมูลต้นทาง (Source)

แอททริบิวท์ sourceType สำหรับเก็บชนิดของคอมโพเนนท์ที่เป็นข้อมูลต้นทาง

แอททริบิวท์ destinationID สำหรับเก็บ ID ของคอมโพเนนท์ที่เป็นข้อมูลปลายทาง (Destination)

แอททริบิวท์ destinationType สำหรับเก็บชนิดของคอมโพเนนท์ที่เป็นข้อมูลปลายทาง

แอททริบิวท์ formatdata สำหรับเก็บรูปแบบของข้อมูลที่ไหลผ่านดาต้าโฟล์วนั้น

แอททริบิวท์ Sx,Sy สำหรับเก็บตำแหน่งต้นทางของดาต้าโฟลว์

แอททริบิวท์ Dx,Dy สำหรับเก็บตำแหน่งปลายทางของดาต้าโฟลว์

CREATE ROW TYPE GateType

(ModelID Integer,

ModelType char(2),

ModelName char(15),

Px Integer,

Py Integer

) UNDER componentType;

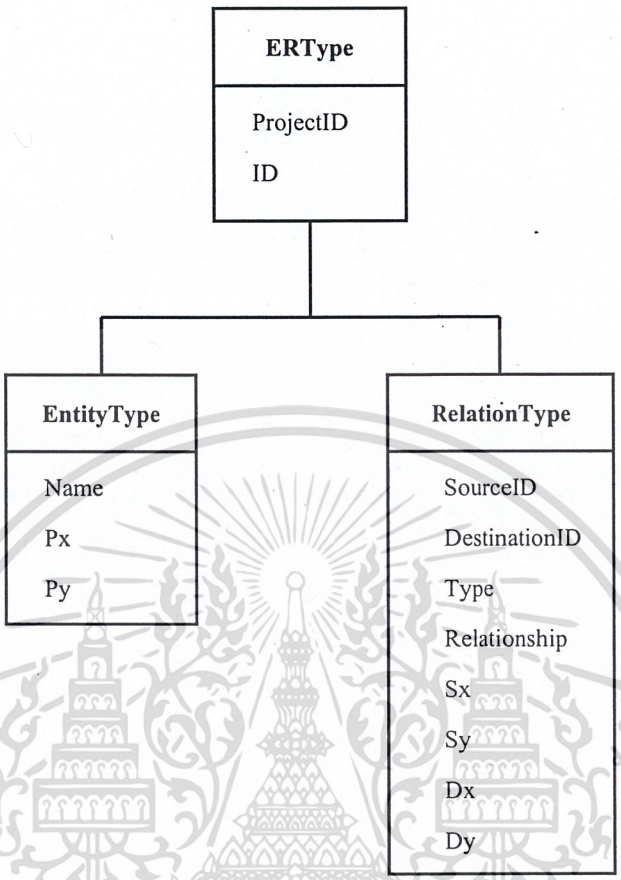
แอททริบิวท์ ModelID สำหรับเก็บ ID ของคอมโพเนนท์ไทพ์ที่เกท(Gate) นั้นอ้างถึง

แอททริบิวท์ ModelType สำหรับเก็บ Type ของคอมโพเนนท์ไทพ์ที่เกทนั้นอ้างถึง

แอททริบิวท์ ModelName สำหรับเก็บชื่อของคอมโพเนนท์เกท

แอททริบิวท์ Px , Py สำหรับเก็บตำแหน่งของคอมโพเนนท์เกท

ในส่วน ของโรว์ไทพ์ที่ใช้เก็บคอมโพเนนท์ของเอนทิตี้รีเลชันชิพไดอะแกรมจะอินเฮอร์ริทไทพ์มาจาก ERType ดังรูป



รูปที่ 7-17 แสดงอินเฮอร์ริทไทพ์ของคอมโพเนนท์ในเอนทิตี้รีเลชันชิพไดอะแกรม

ซึ่งสามารถเขียนเป็นเอสคิวแอลสเตทเมนท์ได้ดังนี้

CREATE ROW TYPE ERType

```
(       ProjectID       integer,
        ID              integer );
```

แอททริบิวท์ ProjectID สำหรับเก็บ ID ของโปรเจคของดาต้าโฟลว์ไดอะแกรมที่ ER นี้อ้างถึง

แอททริบิวท์ ID สำหรับเก็บ ID ของคอมโพเนนท์

CREATE ROW TYPE entityType

```
(       Name    char(20),
        Px      Integer,
        Py      Integer
) UNDER ERType;
```

แอททริบิวท์ Name สำหรับเก็บชื่อของคอมโพแนนท์เอนทิตี้

แอททริบิวท์ Px , Py สำหรับเก็บตำแหน่งของคอมโพเนนท์เอนทิตี้

```
CREATE ROW TYPE RelationType
(       SourceID        integer,
        DestinationID   integer,
        Type            char(5),
        Relationship    char(30),
        Sx              Integer,
        Sy              Integer,
        Dx              Integer,
        Dy              Integer
) UNDER ERType;
```

แอททริบิวท์ SourceID สำหรับเก็บ ID ของคอมโพเนนท์เอนทิตี้ที่เป็นต้นทาง

แอททริบิวท์ DestinationID สำหรับเก็บ ID ของคอมโพเนนท์เอนทิตี้ที่เป็นปลายทาง

แอททริบิวท์ Type สำหรับเก็บชนิดของความสัมพันธ์ระหว่างเอนทิตี้

แอททริบิวท์ Relationship สำหรับเก็บชื่อของความสัมพันธ์

แอททริบิวท์ Sx , Sy สำหรับเก็บตำแหน่งต้นทางของความสัมพันธ์

แอททริบิวท์ Dx , Dy สำหรับเก็บตำแหน่งปลายทางของความสัมพันธ์

หลังจากออกแบบโรว์ไทพ์เรียบร้อยแล้ว จะทำการสร้างตารางจากโรว์ไทพ์ที่สร้างขึ้นโดยเขียนเป็นเอสคิวแอลสเตทเมนท์ได้ดังนี้

```
CREATE TABLE projectTable
(projectID      integer,
 projectName    char(20),
 PRIMARY KEY ( projectID),
 CHECK ( projectName IS NOT NULL ),
 CHECK ( projectID >=1 )    );
```

ตาราง ProojectTable นี้ใช้เก็บความสัมพันธ์ของหมายเลขโปรเจคกับชื่อของโปรเจคซึ่งจะเป็นตารางหลักที่ใช้อ้างถึงโปรเจคทั้งหมดที่มีอยู่ในดาต้าเบส

```
CREATE TABLE processTable OF TYPE processType
        (PRIMARY KEY ( projectID , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( name IS NOT NULL ),
```

```
CHECK ( description IS NOT NULL),
CHECK ( Px IS NOT NULL ),
CHECK ( Py IS NOT NULL ),
CHECK ( ID >= 1 ),
CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL )) );
```

ตาราง ProcessTable มีแอททริบิวต์เป็นไทพ์ของ ProcessType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ name , subpocess , description , Px, Py จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Subprocess ต้องมากกว่าหรือเท่ากับ 0

```
CREATE TABLE externalTable OF TYPE externalType
(PRIMARY KEY ( projectID , ID ),
FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
CHECK ( name IS NOT NULL ),
CHECK ( Px IS NOT NULL ),
CHECK ( Py IS NOT NULL ),
CHECK ( ID >= 1 ),
CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL )));
```

ตาราง ExternalTable มีแอททริบิวต์เป็นไทพ์ของ ExternalType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ name , subpocess , Px, Py จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Subprocess ต้องมากกว่าหรือเท่ากับ 0

```
CREATE TABLE datastoreTable OF TYPE datastoreType
(PRIMARY KEY ( projectID , ID ),
FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
CHECK ( name IS NOT NULL ),
CHECK ( Px IS NOT NULL ),
CHECK ( Py IS NOT NULL ),
CHECK ( ID >= 1 ),
CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL )));
```

ตาราง DatastoreTable มีแอททริบิวต์เป็นไทพ์ของ DatastoreType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง

ProjectTable แอททริบิวต์ name , subpocess , Px, Py จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Subprocess ต้องมากกว่าหรือเท่ากับ 0

```
CREATE TABLE dataflowTable OF TYPE dataflowType
        (PRIMARY KEY ( projectID , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL )),
        CHECK (formatData IS NOT NULL),
        CHECK (( SourceID IS NOT NULL) and ( SourceID >=1)),
        CHECK (( DestinationID IS NOT NULL) and( DestinationID >=1)),
    CHECK (( SourceType IS NOT NULL) and ( SourceType in ('P','E','D','GP','GD','GE'))),
CHECK (( DestinationType IS NOT NULL) and ( DestinationType in ('P','E','D','GP','GD','GE'))),
        CHECK (Sx IS NOT NULL),
        CHECK (Sy IS NOT NULL),
        CHECK (Dx IS NOT NULL),
        CHECK (Dy IS NOT NULL) );
```

ตาราง DataflowTable มีแอททริบิวต์เป็นไทพ์ของ DataflowType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ subpocess, Formatdata , SourceID , SourceType , DestinationID , DestinationType , Sx , Sy , Dx ,Dy จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Subprocess ต้องมากกว่าหรือเท่ากับ 0 ค่า SourceId,DestinationId ต้องมากกว่าหรือเท่ากับ 0 ค่า SourceType,DestinationType ต้องมีค่าเป็น P,E,D,GP,GD หรือ GE เท่านั้น

```
CREATE TABLE GateTable OF TYPE GateType
        (PRIMARY KEY ( projectID,subprocess,ID),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >0 ) and ( subProcess IS NOT NULL )),
        CHECK ( ModelID IS NOT NULL ),
    CHECK (( ModelType IS NOT NULL ) and (ModelType IN ('P','E','D','GP','GE','GP'))),
        CHECK ( ModelName IS NOT NULL ),
        CHECK ( Px IS NOT NULL ),
        CHECK ( Py IS NOT NULL ) );
```

ตาราง GateTable มีแอททริบิวต์เป็นไทพ์ของ GateType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ subpocess, ModelID , ModelType , ModelName , Px ,Py จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Subprocess ต้องมากกว่าหรือเท่ากับ 0 ค่า ModelType ต้องมีค่าเป็น P,E,D,GP,GD หรือ GE เท่านั้น

```
CREATE TABLE entityTable OF TYPE entityType
        (PRIMARY KEY (projectID , ID),
        FOREIGN KEY ( projectID) REFERENCES projectTable (projectID) ,
        CHECK (name IS NOT NULL),
        CHECK (Px IS NOT NULL),
        HECK (Py IS NOT NULL),
        CHECK (ID >=1) );
```

ตาราง EntityTable มีแอททริบิวต์เป็นไทพ์ของ EntityType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ Name, Px ,Py จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1

```
CREATE TABLE RelationTable OF TYPE relationType
        (PRIMARY KEY (projectID , ID),
        FOREIGN KEY ( projectID) REFERENCES projectTable (projectID) ,
        FOREIGN KEY ( SourceID) REFERENCES EntityTable (ID) ,
        FOREIGN KEY ( DestinationID) REFERENCES EntityTable (ID) ,
        CHECK (Relationship IS NOT NULL),
        CHECK ((Type IS NOT NULL) and(Type in ('OTO','OTM','MTM'))),
        CHECK (Sx IS NOT NULL),
        CHECK (Sy IS NOT NULL),
        CHECK (Dx IS NOT NULL),
        CHECK (Dy IS NOT NULL),
        CHECK (ID >=1) ) ;
```

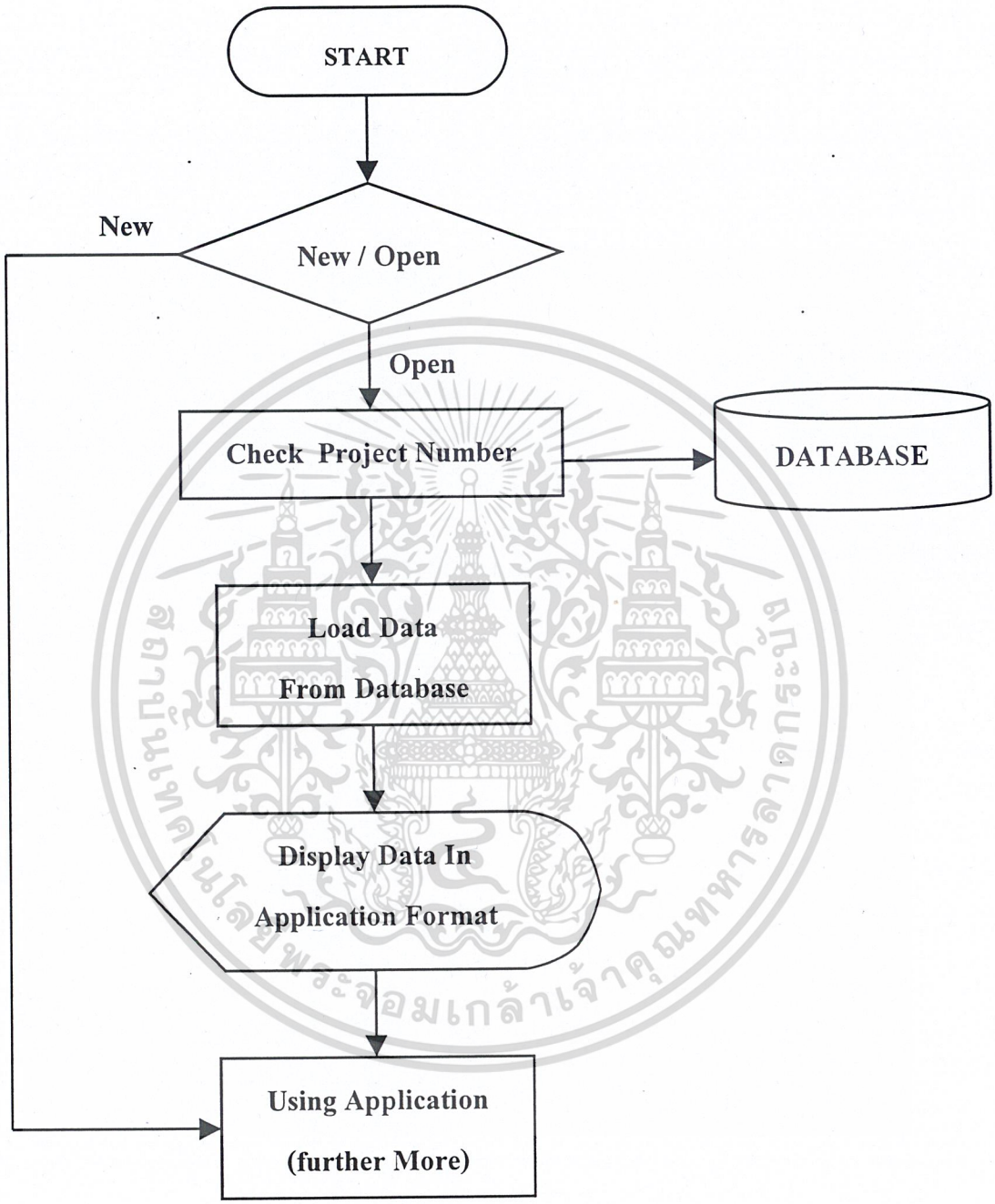
ตาราง RelationTable มีแอททริบิวต์เป็นไทพ์ของ RelationType ซึ่งจะกำหนดให้แอททริบิวต์ ProjectID และ ID เป็นไพรมารีย์คีย์ แอททริบิวต์ ProjectID จะเป็นฟอเรจ์นคีย์ซึ่งอ้างมาจากตาราง ProjectTable แอททริบิวต์ Relationship , Type ,Sx , Sy , Dx , Dy จะต้องไม่เป็น NULL และค่า ID ต้องมากกว่า 1 ค่า Type ต้องมีค่าเป็น OTO , OTM หรือ MTM เท่านั้น

7.4 รูทีนที่สร้างเพื่อใช้งานในฐานข้อมูล

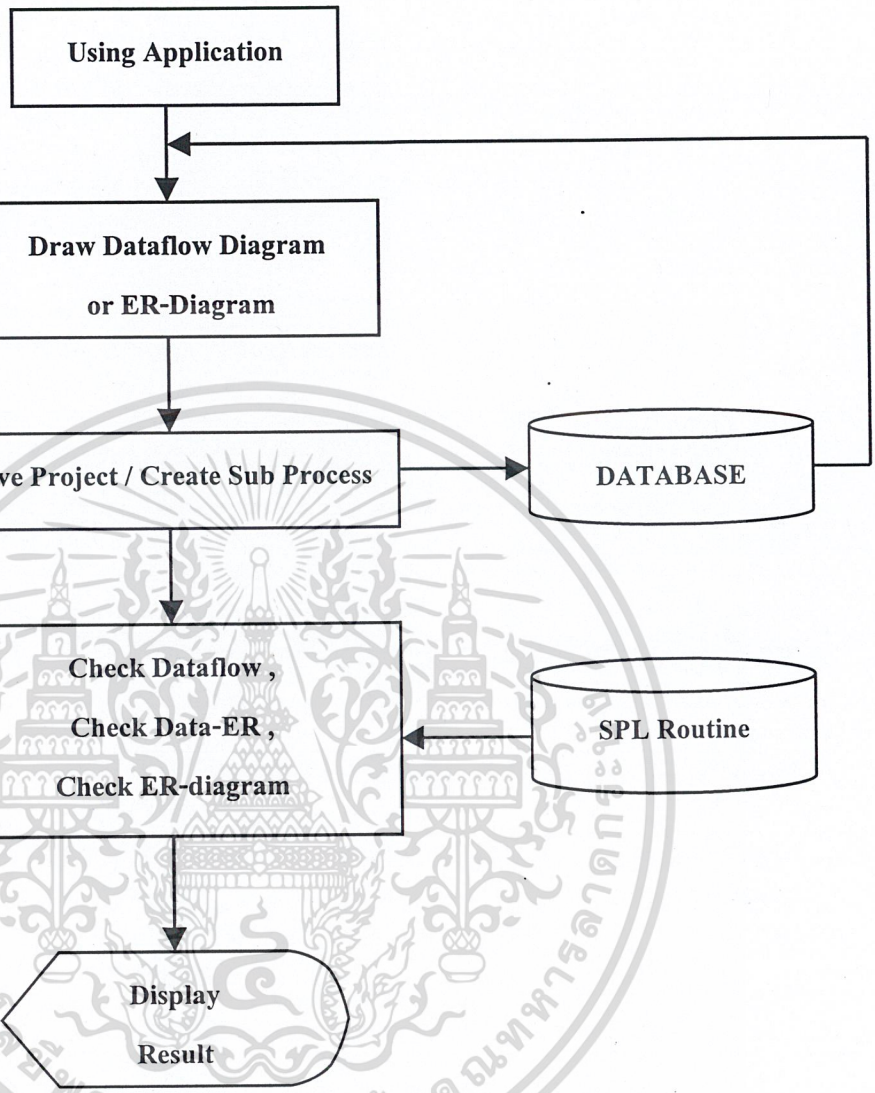
ฟังก์ชันที่สร้างขึ้นเพื่อทำการเช็คความถูกต้องของข้อมูลในฐานข้อมูล ได้แก่

- GenProjectID ( ) สำหรับการสร้างเลขที่ของโปรเจคขึ้นมาใหม่เมื่อมีการสร้างโปรเจคใหม่
- CheckERdiagram (projectId integer) สำหรับเช็คความถูกต้องของ ER diagram ที่สร้างขึ้น
- CheckDFlevel (projectId integer , subprocess integr) สำหรับเช็คความถูกต้องของข้อมูลของดาต้าโฟลว์ในแต่ละ Level
- CheckDataER (projectId integer) สำหรับเช็คความถูกต้องของดาต้าโฟลว์ไดอะแกรมกับ ER ไดอะแกรมที่สัมพันธ์กัน ซึ่งจะเรียกใช้
  - CheckNoHaveSub (projectId integer , subprocess integer) สำหรับเช็คว่า Level นั้นเป็น Level สุดท้ายหรือเปล่า
  - TestCheck3(projectId integer , Subprocess integer) สำหรับเช็คความถูกต้องของดาต้าโฟลว์ไดอะแกรมกับ ER ไดอะแกรมที่สัมพันธ์กัน

7.5 การทำงานของโปรแกรมประยุกต์



รูปที่ 7-18 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรมประยุกต์ (1)



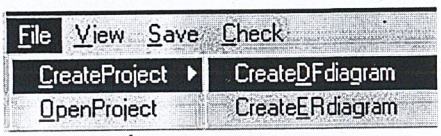
รูปที่ 7-19 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรมประยุกต์ (2)

บทที่ 8
การใช้งานแอพพลิเคชัน

8.1 อธิบายการใช้งานแอพพลิเคชัน

1. เมนูต่างๆ

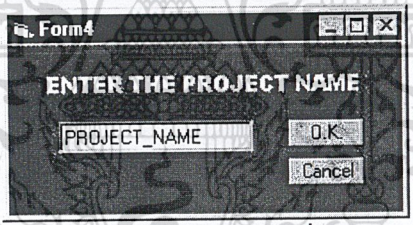
1.1 เมนูไฟล์ ประกอบด้วยเมนูย่อยดังต่อไปนี้



รูปที่ 8-1 แสดงเมนูไฟล์

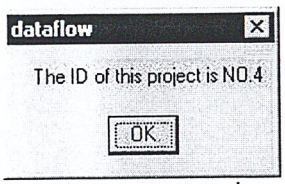
1.1.1 เมนูสร้างโปรเจค ใช้สำหรับสร้างไดอะแกรมใหม่

ในโปรเจคหนึ่งๆจะประกอบด้วยไดอะแกรมสองชนิดคือ ดาต้าโฟลว์ไดอะแกรมและอีอาร์ไดอะแกรม ผู้ใช้สามารถเลือกสร้างไดอะแกรมทั้งสองชนิดนี้โดยการคลิกที่เมนูดังรูป ทั้งนี้ในการเริ่มต้นสร้างโปรเจค จะเริ่มต้นโดยการสร้างไดอะแกรมใดก่อนก็ได้ ตามความพอใจของผู้ใช้ เมื่อผู้ใช้เลือกคลิกที่เมนูย่อยอันใดอันหนึ่งจะปรากฏไดอะล็อกดังต่อไปนี้



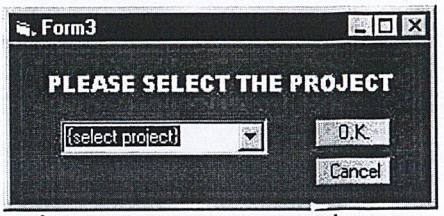
รูปที่ 8-2 แสดงไดอะล็อกระบุชื่อโปรเจค

ผู้ใช้จะต้องใส่ชื่อของโปรเจคที่ต้องการสร้างใหม่แล้วคลิกปุ่มโอเคเพื่อตกลงที่จะสร้างโปรเจคใหม่ จากนั้นจะปรากฏกล่องข้อความ แสดงเลขที่ของโปรเจคใหม่ขึ้นที่หน้าจอดังนี้



รูปที่ 8-3 แสดงกล่องข้อความเลขที่ของโปรเจคใหม่

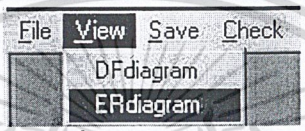
1.1.2 เมนูเปิดโปรเจค ใช้สำหรับเลือกแสดงไดอะแกรมที่มีอยู่แล้ว เมื่อผู้ใช้คลิกที่เมนูนี้จะมีไดอะล็อกดังต่อไปนี้แสดงขึ้นที่หน้าจอ



รูปที่ 8-4 แสดงไดอะล็อกเลือกชื่อโปรเจค

เมื่อคลิกเลือกโปรเจคที่ต้องการและคลิกปุ่มโอเคเรียบร้อยแล้ว ไดอะแกรมของโปรเจคที่เลือกจะถูกนำมาแสดงบนหน้าจอ

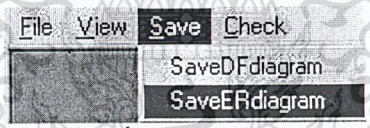
1.2 เมนูวิว



รูปที่ 8-5 แสดงเมนูวิว

ประกอบด้วยสองเมนูย่อยคือ เมนูวิวดาต้าโฟลว์ไดอะแกรมและเมนูวิวอีอาร์ไดอะแกรม ใช้สำหรับเลือกดูไดอะแกรมทั้งสองไดอะแกรมของโปรเจคเมื่อโปรเจคถูกนำมาแสดงบนหน้าจอเรียบร้อยแล้ว

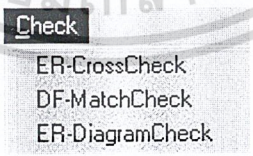
1.3 เมนูเซฟ ใช้สำหรับบันทึกไดอะแกรมที่ได้สร้างและแก้ไขเรียบร้อยแล้วลงในฐานข้อมูล



รูปที่ 8-6 แสดงเมนูเซฟ

ผู้ใช้สามารถเลือกบันทึกไดอะแกรมของโปรเจคได้ทีละไดอะแกรมตามต้องการ

1.4 เมนูเช็ค



รูปที่ 8-7 แสดงเมนูเช็ค

1.4.1 เมนูอีอาร์ครอสเช็ค

ใช้สำหรับตรวจสอบว่าดาต้าโฟลว์ไดอะแกรมที่สร้างขึ้นสัมพันธ์กับอีอาร์ไดอะแกรมที่สร้างขึ้นหรือไม่ โดยในการตรวจสอบจะทำการตรวจสอบชื่อของดาต้าสตอร์ที่อยู่ในโพรเซสย่อยชั้นล่างสุดว่าตรงกับชื่อของเอ็นติตี้ไทพ์ในอีอาร์ไดอะแกรมหรือไม่ ผลการตรวจสอบจะถูกแสดงในรูปของกล่องข้อความ “Pass” หรือ “Not Pass”

1.4.2 เมนูดีเอฟแมทช์เช็ค

ใช้ตรวจสอบดาต้าโฟลว์ไดอะแกรมในระดับล่างว่าสอดคล้องกับไดอะแกรมในระดับบนหรือไม่ ไดอะแกรมระดับล่างที่กล่าวถึงในที่นี้หมายถึง ไดอะแกรมอธิบายการทำงานของโพรเซสโดยละเอียด

1.4.3 เมนูอีอาร์ไดอะแกรมเช็ค

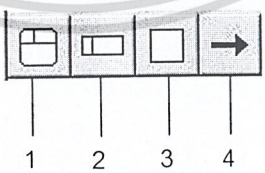
ใช้ตรวจสอบว่าอีอาร์ไดอะแกรมที่สร้างขึ้นมีความถูกต้องหรือไม่

2. หน้าต่างและแถบเครื่องมือที่ใช้สร้างดาต้าโฟลว์ไดอะแกรม

รูปที่ 8-8 แสดงลักษณะหน้าต่างที่ใช้สร้างดาต้าโฟลว์ไดอะแกรม

หน้าต่างที่เห็นดังรูปจะปรากฏหลังจากที่ผู้ใช้คลิกเลือกเมนู สร้างดาต้าโฟลว์ไดอะแกรม เรียบร้อยแล้ว ที่ส่วนหัวของหน้าต่างจะมีแถบเครื่องมือซึ่งประกอบด้วยเครื่องมือ 4 อย่างต่อไปนี้

2.1 โพรเซส เป็นเครื่องมือใช้สร้างรูปแสดงโพรเซสของดาต้าโฟลว์ไดอะแกรม เมื่อผู้ใช้คลิกที่



รูปที่ 8-9 แสดงแถบเครื่องมือของหน้าต่างดาต้าโฟลว์ไดอะแกรม

เครื่องมือนี้แล้วทำการคลิกลงบนหน้าต่าง จะปรากฏรูปแสดงโพรเซสขึ้นที่มุมบนซ้ายของหน้าต่าง ผู้ใช้สามารถเคลื่อนย้ายรูปดังกล่าวไปยังตำแหน่งที่ต้องการโดยการคลิกค้างที่รูปแล้วลากรูปไปยังตำแหน่งนั้น

2.2 ดาต้าสตอร์ เป็นเครื่องมือใช้สร้างรูปแสดงดาต้าสตอร์ของดาต้าโฟลว์ไดอะแกรม เมื่อผู้ใช้คลิกที่เครื่องมือนี้ แล้วทำการคลิกลงบนหน้าต่าง จะปรากฏรูปแสดงดาต้าสตอร์ขึ้นที่มุมบน

ซ้ายของหน้าต่าง ผู้ใช้สามารถเคลื่อนย้ายรูปดังกล่าวไปยังตำแหน่งที่ต้องการโดยการคลิกค้างที่รูปแล้วลากรูปไปยังตำแหน่งนั้น

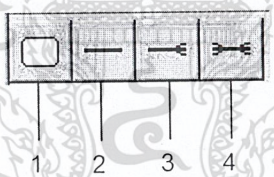
2.3 เอ็กซ์เทอร์นอล เป็นเครื่องมือใช้สร้างรูปแสดงเอ็กซ์เทอร์นอลเอ็นติตี้ของดาต้าโฟลว์ไดอะแกรม เมื่อผู้ใช้คลิกที่เครื่องมือนี้ แล้วทำการคลิกลงบนหน้าต่าง จะปรากฏรูปแสดงเอ็กซ์เทอร์นอลเอ็นติตี้ขึ้นที่มุมบนซ้ายของหน้าต่าง ผู้ใช้สามารถเคลื่อนย้ายรูปดังกล่าวไปยังตำแหน่งที่ต้องการโดยการคลิกค้างที่รูปแล้วลากรูปไปยังตำแหน่งนั้น

2.4 ดาต้าโฟลว์ เป็นเครื่องมือใช้ลากเส้นลูกศรเชื่อมต่อระหว่างรูปแสดงอุปกรณ์ทั้งสามอุปกรณ์ของดาต้าโฟลว์ไดอะแกรม โดยเมื่อผู้ใช้คลิกที่เครื่องมือนี้แล้ว ผู้ใช้จะต้องคลิกเลือกรูปที่ต้องการให้อยู่ติดกับท้ายลูกศรเป็นอันดับแรก จากนั้นจึงคลิกเลือกรูปที่ต้องการให้อยู่ติดกับหัวลูกศร หลังจากที่ผู้ใช้คลิกเลือกรูปทั้งสองเสร็จเรียบร้อยแล้ว จะปรากฏเส้นเชื่อมต่อระหว่างรูปทั้งสอง

หลักในการ เชื่อมต่อรูปแสดงอุปกรณ์ของดาต้าโฟลว์ไดอะแกรมคือ ดาต้าสตอร์ และเอกซ์เทอนอลเอ็นติตี้ สามารถเชื่อมต่อกับโพรเซสได้เท่านั้น ไม่สามารถเชื่อมต่อระหว่างกันหรือระหว่างอุปกรณ์เดียวกันได้

3 หน้าต่างและแถบเครื่องมือที่ใช้สร้างอีอาร์ไดอะแกรม

ลักษณะของหน้าต่างใช้สร้างอีอาร์ไดอะแกรมจะเป็นแบบเดียวกับหน้าต่างใช้สร้างดาต้าโฟลว์ไดอะแกรม ต่างกันเพียงแค่ส่วนแสดงแถบเครื่องมือเท่านั้น โดยแถบเครื่องมือที่ใช้สร้างอีอาร์ไดอะแกรมจะมีทั้งหมด 4 อย่างดังต่อไปนี้



รูปที่ 8-10 แสดงแถบเครื่องมือของหน้าต่างดาต้าโฟลว์ไดอะแกรม

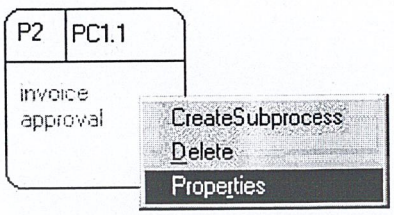
3.1 เอ็นทิตี้ เป็นเครื่องมือใช้สร้างรูปแสดงเอ็นทิตี้ไทพ์ของอีอาร์ไดอะแกรม เมื่อผู้ใช้คลิกที่เครื่องมือนี้ แล้วทำการคลิกลงบนหน้าต่าง จะปรากฏรูปแสดงเอ็นทิตี้ไทพ์ ขึ้นที่มุมบนซ้ายของหน้าต่าง ผู้ใช้สามารถเคลื่อนย้ายรูปดังกล่าวไปยังตำแหน่งที่ต้องการโดยการคลิกค้างที่รูปแล้วลากรูปไปยังตำแหน่งนั้น

3.2 ความสัมพันธ์แบบวันทูวัน เป็นเครื่องมือใช้สำหรับสร้างเส้นความสัมพันธ์ระหว่างเอ็นทิตี้ไทพ์ โดยเมื่อผู้ใช้คลิกที่เครื่องมือนี้แล้ว ผู้ใช้จะต้องคลิกที่รูปแสดงเอ็นทิตี้ไทพ์สองรูป จากนั้นจะปรากฏเส้นความสัมพันธ์แบบวันทูวันเชื่อมระหว่างรูปเอ็นทิตี้ไทพ์ทั้งสอง

3.3 ความสัมพันธ์แบบวันทูแมนี่ เป็นเครื่องมือใช้สำหรับสร้างเส้นความสัมพันธ์ระหว่างเอ็นทิตี้ไทพ์ โดยเมื่อผู้ใช้คลิกที่เครื่องมือนี้แล้ว ผู้ใช้จะต้องคลิกที่รูปแสดงเอ็นทิตี้ไทพ์สองรูป จากนั้นจะปรากฏเส้นความสัมพันธ์แบบวันทูแมนี่เชื่อมระหว่างรูปเอ็นทิตี้ไทพ์ทั้งสอง

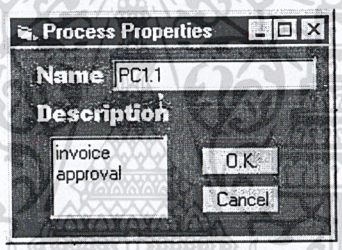
3.4 ความสัมพันธ์แบบแมนี่ทูแมนี่ เป็นเครื่องมือใช้สำหรับสร้างเส้นความสัมพันธ์ระหว่างเอ็นทิตี้ไทพ์ โดยเมื่อผู้ใช้คลิกที่เครื่องมือนี้แล้ว ผู้ใช้จะต้องคลิกที่รูปแสดงเอ็นทิตี้ไทพ์สองรูป จากนั้นจะปรากฏเส้นความสัมพันธ์แบบแมนี่ทูแมนี่เชื่อมระหว่างรูปเอ็นทิตี้ไทพ์ทั้งสอง

4 การเปลี่ยนแปลงคุณสมบัติของคอมโพเนนท์



รูปที่ 8-11 แสดงการเลือกเมนูเพื่อแก้ไขคุณสมบัติของโพรเซส

ผู้ใช้สามารถเปลี่ยนแปลงคุณสมบัติของคอมโพเนนท์ต่างๆ ทั้งในดาต้าโฟลว์ไดอะแกรมและอีอาร์ไดอะแกรมได้โดยเลื่อนเมาส์ไปยังคอมโพเนนท์แล้วคลิกขวา จะปรากฏเมนูพร็อพเพอร์ตี้ขึ้นที่หน้าจอให้ทำการคลิกที่เมนู จากนั้นผู้ใช้จะสามารถแก้ไขคุณสมบัติของคอมโพเนนท์ ได้ตามต้องการ

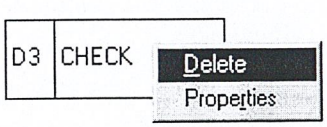


รูปที่ 8-12 แสดงไดอะล็อกแก้ไขคุณสมบัติของโพรเซส

สำหรับคอมโพเนนท์ดาต้าโฟลว์ และ รีเลชั่น นั้น ผู้ใช้จะต้องคลิกที่เส้นดาต้าโฟลว์หรือรีเลชั่นจนกระทั่งเส้นกลายเป็นสีแดง ผู้ใช้จึงจะสามารถคลิกขวาเลือกเมนูพร็อพเพอร์ตี้เพื่อทำการแก้ไขคุณสมบัติของคอมโพเนนท์ได้

5 การลบคอมโพเนนท์

หากต้องการจะทำการลบคอมโพเนนท์ใดๆ ให้ทำการเลื่อนเมาส์ไปยังคอมโพเนนท์นั้นและทำการคลิกขวา จากนั้นเลือกเมนูดีลีท (Delete) เพื่อทำการลบ

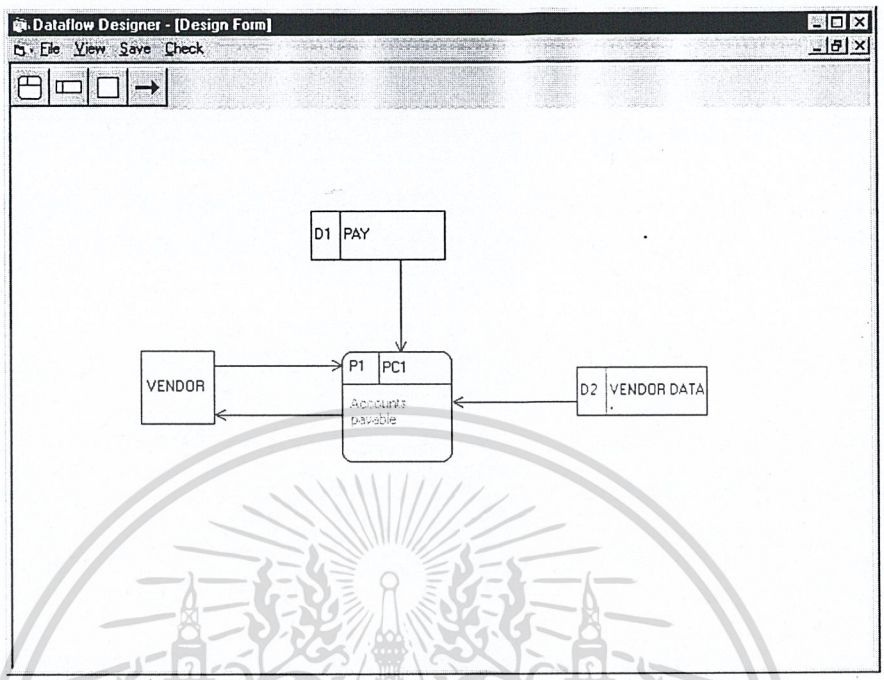


รูปที่ 8-13 แสดงการเลือกเมนูดีลีทเพื่อลบดาต้าสตอร์

สำหรับคอมโพเนนท์ดาต้าโฟลว์ และ รีเลชั่น นั้น ผู้ใช้จะต้องคลิกที่เส้นดาต้าโฟลว์หรือรีเลชั่นจนกระทั่งเส้นกลายเป็นสีแดงจึงจะสามารถคลิกขวาแล้วเลือกเมนูดีลีท (Delete) ได้

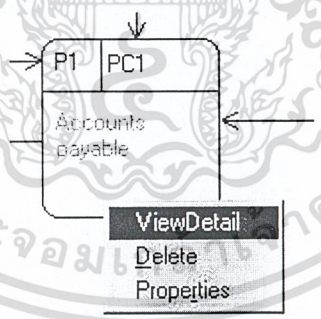
6 ตัวอย่างดาต้าโฟลว์ไดอะแกรม ที่สร้างโดยใช้แอพพลิเคชั่น

6.1 ดาต้าโฟลว์ไดอะแกรมที่ คอนเท็กซ์เลเวล



รูปที่ 8-14 แสดงตัวอย่างหน้าต่างดาต้าโฟลว์ไดอะแกรมที่คอนเท็กซ์เลเวล

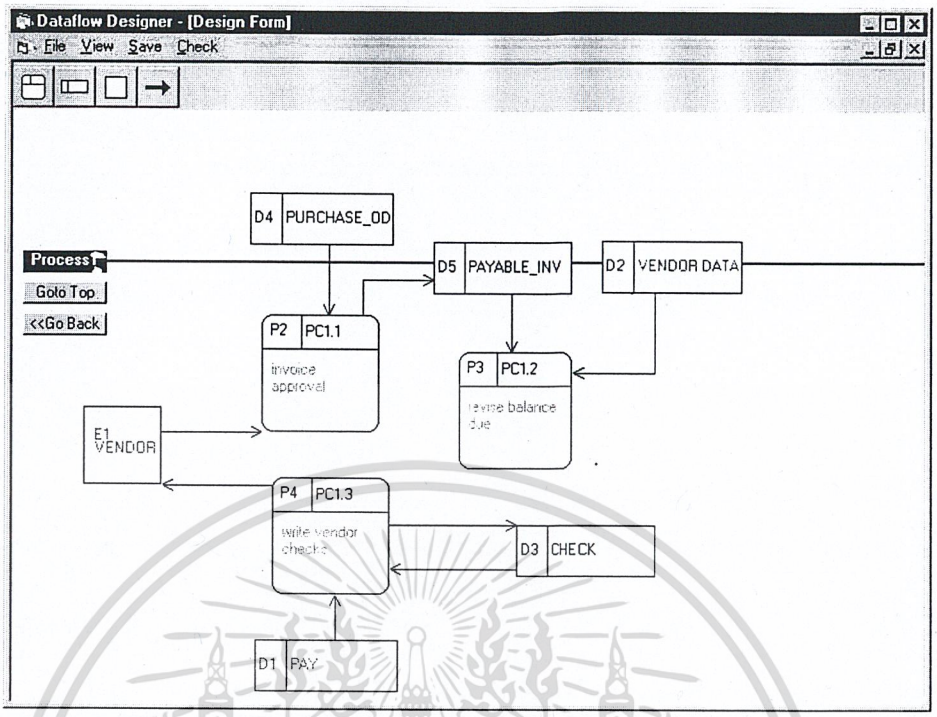
เมื่อต้องการให้หน้าจอแสดงรายละเอียดในระดับย่อยของโพรเซส สามารถทำได้โดยการคลิกขวา



รูปที่ 8-15 แสดงการเลือกเมนูวิวดีเทลส์

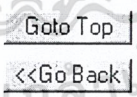
ที่โพรเซส และเลือกเมนูวิวดีเทลส์ ดังรูป

6.2 รายละเอียดในระดับย่อยของโพรเซส “พีหนึ่ง” จะปรากฏดังนี้



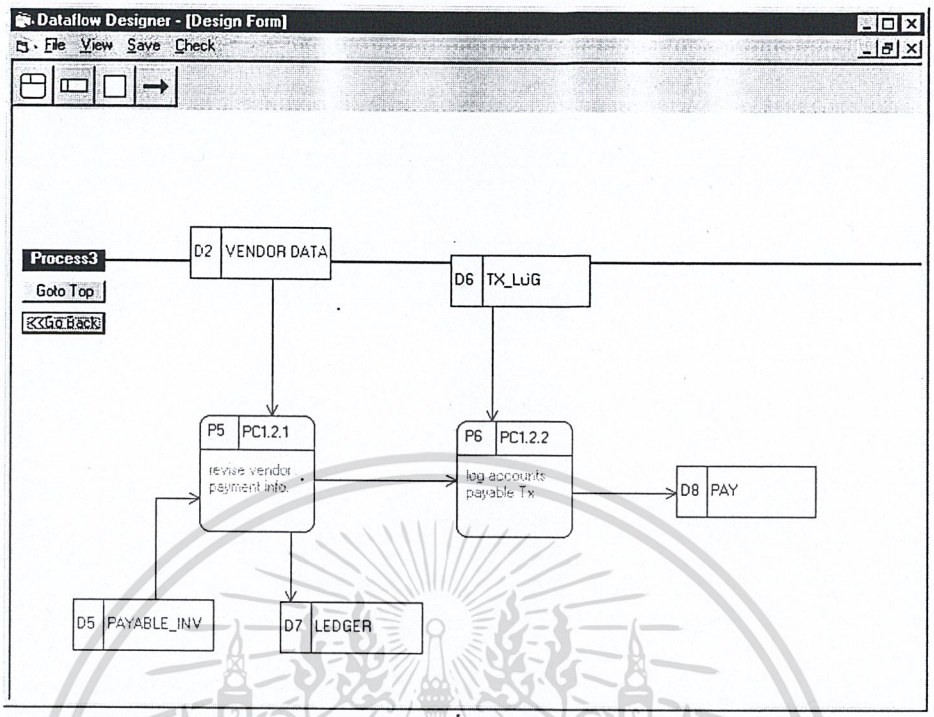
รูปที่ 8-16 แสดงหน้าต่างดาต้าโฟลว์ไดอะแกรมซึ่งแสดงรายละเอียดในระดับย่อยของโพรเซสพีหนึ่ง

6.3 หากต้องการดูรายละเอียดในระดับย่อยของโพรเซสใดๆ อีก ก็สามารถทำได้โดยการ คลิกขวาที่รูปโพรเซส และเลือกเมนูวิวดีเทลส์เช่นเดิม และหาก ต้องการเรียกหน้าจอเดิมกลับมาแสดงใหม่อีกครั้ง สามารถทำได้โดยคลิกปุ่มโกแบ็ค หรือ หากต้องการเรียกหน้าจอแสดงไดอะแกรม ณ คอนเท็กซ์เลเวล กลับมาแสดงใหม่ ก็สามารถทำได้โดยคลิกปุ่มโกทูท็อป



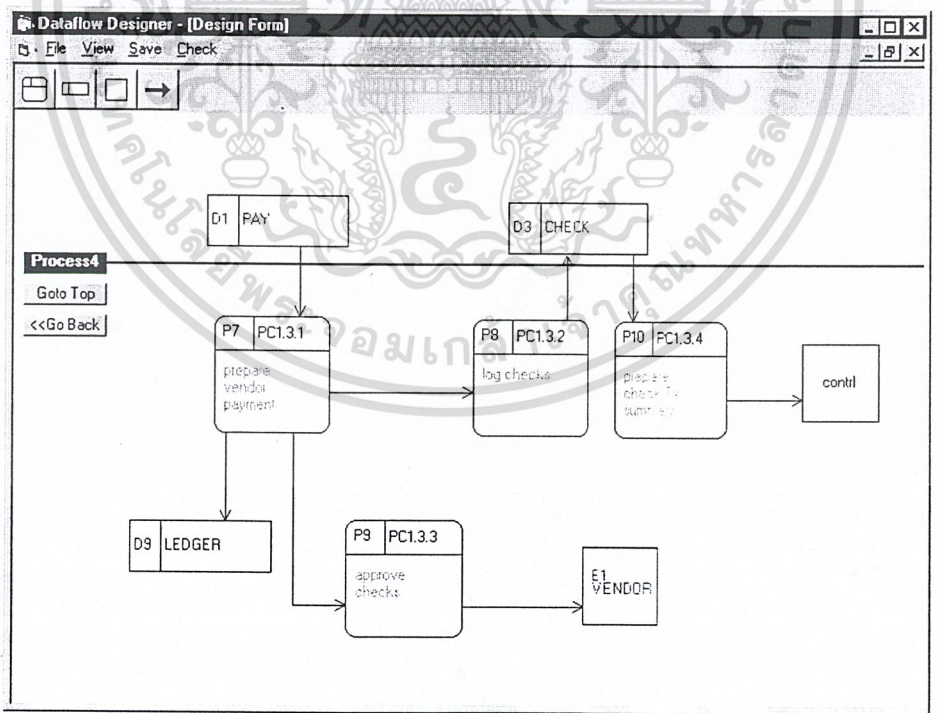
รูปที่ 8-17 แสดงปุ่มโกแบ็คและโกทูท็อป

6.4 รูปแสดงตัวอย่างหน้าจอรายละเอียดในระดับย่อยของโพรเซส “พีสาม”



รูปที่ 8-18 แสดงหน้าต่างดาต้าโฟลว์ไดอะแกรมซึ่งแสดงรายละเอียดในระดับย่อยของโพรเซสพีสาม

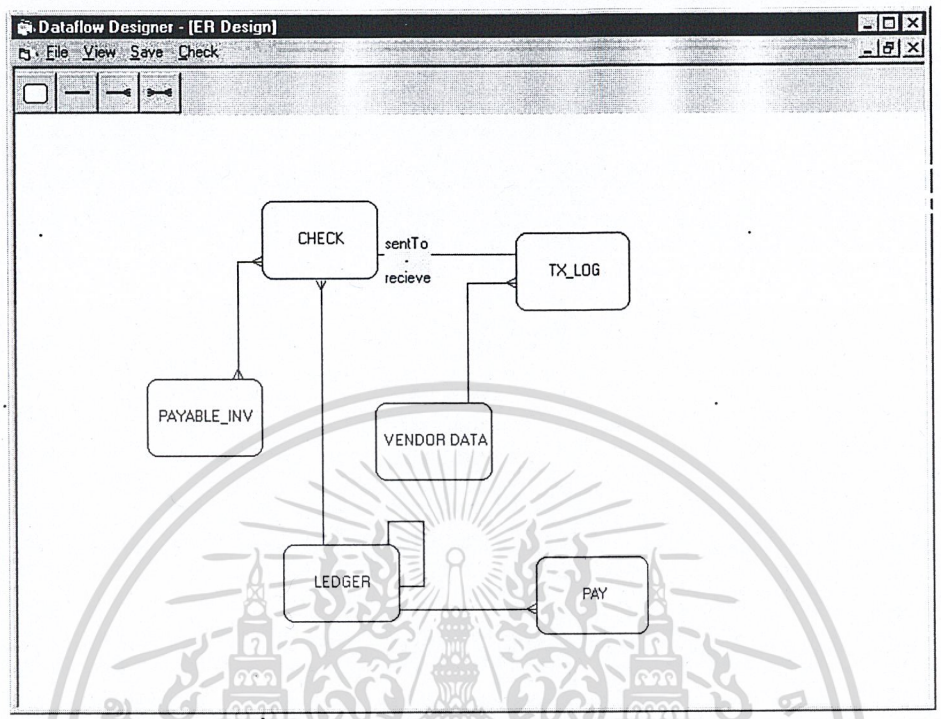
6.5 รูปแสดงตัวอย่างหน้าจอรายละเอียดในระดับย่อยของโพรเซส “พีสี่”



รูปที่ 8-19 แสดงหน้าต่างดาต้าโฟลว์ไดอะแกรมซึ่งแสดงรายละเอียดในระดับย่อยของโพรเซสพีสี่

7 ตัวอย่างอีอาร์ไดอะแกรม ที่สร้างโดยใช้แอพพลิเคชั่น

ผู้ใช้สามารถเรียกดูอีอาร์ไดอะแกรมได้โดยคลิกที่เมนูวิวอีอาร์ไดอะแกรม



รูปที่ 8-20 แสดงตัวอย่างหน้าต่างอีอาร์ไดอะแกรม

8.2 ข้อมูลที่ถูกเก็บในฐานข้อมูลของดาต้าโฟลว์ไดอะแกรมตัวอย่าง จะถูกเก็บไว้ในห้าตารางดังนี้

1. ตารางโพรเซส

	projectid	subprocess	id	name	px	py	description
5	1	3	5	PC1.2.1	146	270	revise vendor~~payment info.
6	1	3	6	PC1.2.2	344	273	log accounts~~payable Tx
7	1	4	7	PC1.3.1	156	192	prepare ~~vendor payment
8	1	4	8	PC1.3.2	355	194	log checks
9	1	4	9	PC1.3.3	258	348	approve checks
10	1	4	10	PC1.3.4	465	195	prepare~~check Tx~~summary

รูปที่ 8-21 แสดงตารางโพรเซส

2. ตารางเอ็กเทอร์นอล

	projectid	subprocess	id	name	px	py
1	1	0	1	VENDOR	105	232
2	1	4	2	contrl	614	212

รูปที่ 8-22 แสดงตารางเอ็กซ์เทอร์นอล

3. ดาต้าต้าสตอร์

	projectid	subprocess	id	name	px	py
1	1	0	1	PAY	244	119
2	1	0	2	VENDOR DATA	463	246
3	1	1	3	CHECK	388	353
4	1	1	4	PURCHASE_OD	182	99
5	1	1	5	PAYABLE_INV	323	136
6	1	3	6	TX_LOG	339	146
7	1	3	7	LEDGER	209	413
8	1	3	8	PAY	517	307
9	1	4	9	LEDGER	91	349

รูปที่ 8-23 แสดงตารางดาต้าสตอร์

4. ตารางเกท

	projectid	subprocess	id	modelid	modeltype	modelname	px	py
1	1	1	1	1	E	VENDOR	55	263
2	1	1	2	2	D	VENDOR DATA	456	135
3	1	1	3	1	D	PAY	186	442
4	1	3	4	5	D	PAYABLE_INV	48	412
5	1	3	5	2	GD	VENDOR DATA	138	126
6	1	4	6	1	GE	VENDOR	440	368
7	1	4	7	3	D	CHECK	382	104
8	1	4	8	3	GD	PAY	150	99

รูปที่ 8-24 แสดงตารางเกท

5. ตารางดาต้าโฟลว์

	projectid	subprocess	id	sourceid	sourcetype	destinationid	destinationtype	formatdata	sx	sy	dx	dy
1	1	0	1	1	E	1	P	vender invoice	146	244	271	251
2	1	0	2	1	P	1	E	check	276	284	157	284
3	1	0	3	2	D	1	P	mailing address	471	274	347	286
4	1	0	4	1	D	1	P	balance	317	145	327	251
5	1	1	5	1	GE	2	P	vender invoice	100	282	196	250
6	1	1	6	4	D	2	P	invoice	242	129	241	197
7	1	1	7	4	P	1	GE	check	200	323	103	312
8	1	1	8	2	P	5	D	payment voucher	268	213	333	165
9	1	1	9	5	D	3	P	batched invoice	384	165	377	223
10	1	1	10	2	GD	3	P	mailing address	497	168	420	236

รูปที่ 8-25 แสดงตารางดาต้าโฟลว์ส่วนที่หนึ่ง

	projectid	subprocess	id	sourceid	sourcetype	destinationid	destinationtype	formatdata	sx	sy	dx	dy
11	1	1	11	3	GD	4	P	balance	247	448	258	395
12	1	1	12	4	P	3	D	check amount	268	353	396	363
13	1	1	13	3	D	4	P	account balance	396	387	282	379
14	1	3	14	6	D	6	P	tx_details	371	177	383	277
15	1	3	15	5	GD	5	P	mailing address	202	162	195	273
16	1	3	16	4	GD	5	P	batched invoice	112	423	156	334
17	1	3	17	5	P	6	P	payable details	225	319	350	330
18	1	3	18	5	P	7	D	account tx_details	217	356	223	417
19	1	3	19	6	P	8	D	acc_due details	425	328	532	336
20	1	4	20	8	GD	7	P	balance	222	122	224	194

รูปที่ 8-26 แสดงตารางดาต้าโฟลว์ส่วนที่สอง

	projectid	subprocess	id	sourceid	sourcetype	destinationid	destinationtype	formatdata	sx	sy	dx	dy
21	1	4	21	7	P	8	P	check details	237	250	382	262
22	1	4	22	7	P	9	D	payment tx_details	165	268	163	358
23	1	4	23	7	P	9	P	printed check	217	277	266	413
24	1	4	24	9	P	6	GE	check	332	414	451	415
25	1	4	25	8	P	7	GD	check amount	427	203	428	133
26	1	4	26	7	GD	10	P	account balance	480	131	479	197
27	1	4	27	10	P	2	E	check lists	547	255	630	247

รูปที่ 8-27 แสดงตารางดาต้าโฟลว์ส่วนที่สาม

8.3 ข้อมูลที่ถูกเก็บในฐานข้อมูลของอีอาร์ไดอะแกรมตัวอย่าง จะถูกเก็บไว้ในสองตารางดังนี้

1. ตารางเอ็นติตี้

	projectid	id	name	px	py
1	1	1	VENDOR DATA	277	257
2	1	2	TX_LOG	386	127
3	1	3	PAY	400	376
4	1	4	LEDGER	205	366
5	1	5	PAYABLE_INV	100	238
6	1	6	CHECK	190	102

รูปที่ 8-28 แสดงตารางเอ็นติตี้

2. ตารางรีเลชัน

	projectid	id	sourceid	destinationid	type	relationship	sx	sy	dx	dy
1	1	1	5	6	MTM	Entity1 - Entity2	171	247	211	148
2	1	2	6	2	OTO	sentTo - recieve	263	143	408	155
3	1	3	1	2	OTM	Entity1 - Entity2	348	272	393	165
4	1	4	4	6	OTM	Entity1 - Entity2	236	375	230	153
5	1	5	4	4	OTO	Entity1 - Entity2	286	389	280	398
6	1	6	4	3	OTM	Entity1 - Entity2	280	416	422	420

รูปที่ 8-29 แสดงตารางรีเลชัน

บทที่ 9
สรุป

9.1 สรุปความแตกต่างระหว่างระบบฐานข้อมูลเชิงวัตถุสัมพันธ์และฐานข้อมูลแบบรีเลชันแนล

โครงสร้างข้อมูลของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ เป็นโครงสร้างข้อมูลซึ่งปรับปรุงมาจากโครงสร้างข้อมูลของระบบฐานข้อมูลแบบรีเลชันแนล โดยมีการเพิ่มชนิดข้อมูลเพื่อให้ แอททริบิวในตารางเก็บข้อมูลซึ่งมีความซับซ้อนยิ่งขึ้น และนอกจากนี้ ในส่วนของคำสั่งภาษาฐานข้อมูล ยังมีการเพิ่มคำสั่งในการจัดการกับข้อมูลชนิดใหม่ๆที่ถูกเพิ่มเติมเข้าไปอีกด้วย

1. ตารางที่ซับซ้อนมากขึ้น

ในโครงสร้างข้อมูลของระบบฐานข้อมูลแบบรีเลชันแนล นิยามไว้ว่าโดเมนของทุกๆแอททริบิวต้องอะติมมิก หมายถึงในแอททริบิวหนึ่งๆ สามารถเก็บข้อมูลได้เพียงค่าเดียวและข้อมูลนั้นไม่สามารถแยกย่อยต่อไปอีกได้ แต่ในโครงสร้างข้อมูลของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์อนุญาตให้ค่าของข้อมูลในแอททริบิวมีหลายค่าได้ ตัวอย่างเช่น

Title	Author-list	Date (day,month,year)	Keyword-list
Salesplan	{Smith,Jones}	(1,April,89)	{profit,strategy}
Status report	{Jones,Frick}	(17,June,94)	{profit,personnel}

ตารางที่ 9-1 แสดงตัวอย่างคอลัมน์ที่เก็บข้อมูลหลายค่า

จากตารางที่ 9-1 ด้านบน จะเห็นได้ว่า ภายในคอลัมน์เก็บข้อมูลหลายค่า เนื่องจากในเอกสารหนึ่งอาจมีผู้แต่งหลายคน กำหนดคำเป็นคีย์ของเอกสารได้หลายคำ หรือการเก็บวันที่ที่แต่ง ซึ่งถึงแม้จะไม่ได้เก็บหลายๆ วันในหนึ่งคอลัมน์ของแถว แต่เมื่อพิจารณาแล้วจะพบว่า วันที่สามารถแยกเป็นฟิลด์ย่อยได้ คือ วัน เดือน ปี นั่นก็หมายความว่า วันที่ไม่อะตอมมิก

2. คำสั่งภาษาฐานข้อมูลที่มีประสิทธิภาพ

มีการปรับปรุงคำสั่งภาษาฐานข้อมูล (SQL) ที่ใช้ในระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ให้สามารถจัดการกับชนิดข้อมูลที่ซับซ้อน เช่น ภาพ วีดีโอ เสียง และกำหนดคุณสมบัติของอ็อบเจค เช่น คุณสมบัติการสืบทอด(inheritance)ให้กับข้อมูลได้ ภาษาฐานข้อมูลที่ถูกปรับปรุงขึ้นมาใหม่นี้ มีชื่อเรียกว่า SQL3

9.2 สรุปลักษณะทั่วไปของแอพพลิเคชัน

แอพพลิเคชันที่ได้ทำการพัฒนาขึ้นเป็นแอพพลิเคชันที่ช่วยให้ผู้ใช้สามารถสร้างดาต้าโฟลไดอะ

แกรมและอีอาร์ไดอะแกรมได้สะดวกขึ้น เนื่องจากมีการติดต่อกับผู้ใช้แบบกราฟฟิก และได้ทำการเก็บข้อมูลของไดอะแกรมทั้งหมดลงใน,ฐานข้อมูล แทนการเก็บลงไฟล์แบบแอพพลิเคชันทั่วไป

9.3 ปัญหาและอุปสรรคในการจัดทำโครงงาน

1. เอกสารประกอบการใช้งานเครื่องมือติดต่อระหว่างแอพพลิเคชันและฐานข้อมูลอินฟอร์มิกซ์มีตัวอย่างการใช้งานค่อนข้างน้อย ทำให้ต้องลองปฏิบัติจริง โดยไม่ทราบผลล่วงหน้า
2. ดาต้าเบส สกีมา ที่ออกแบบในตอนต้นๆ เมื่อเริ่มทำโครงงาน ไม่สามารถใช้งานได้จริงเมื่อนำมาประยุกต์ใช้ในการสร้างแอพพลิเคชัน ทำให้ต้องมีการปรับปรุงใหม่หลายครั้ง
3. เวลาที่ใช้ในการติดต่อระหว่างฐานข้อมูลและแอพพลิเคชัน ต้องใช้เวลามาก ทำให้การทำงานของโปรแกรมแอพพลิเคชันช้าตามไปด้วย
4. ฐานข้อมูลอินฟอร์มิกรุ่นที่ใช้ยังค่อนข้างใหม่ มีเอกสารประกอบน้อย

9.4 แนวทางการวิจัยต่อ

ควรมีการปรับปรุงพัฒนาแอพพลิเคชันให้สามารถสร้างดาต้าดิกชันนารี (Data Dictionary) เพื่ออธิบายรายละเอียดของดาต้าโฟลว์ไดอะแกรมแต่ละไดอะแกรมได้ , พัฒนาการเช็คความถูกต้องของอีอาร์ไดอะแกรมกับดาต้าโฟลว์ไดอะแกรมที่สัมพันธ์กันและความถูกต้องของข้อมูลที่ไหลผ่านดาต้าโฟลว์ให้มีความยืดหยุ่นมากขึ้น และพัฒนาการเก็บข้อมูลลงในฐานข้อมูลให้เป็นแบบออบเจ็กต์มากขึ้น

ภาคผนวก

ภาคผนวก ก

ผลิตภัณฑ์ INFORMIX

การติดตั้ง Informix Universal Server

1.) สำรวจสเปคของเครื่องคอมพิวเตอร์

ก่อนที่จะติดตั้งผลิตภัณฑ์ จะต้องเตรียมเครื่องคอมพิวเตอร์ให้ได้มาตรฐานดังต่อไปนี้

1.1 ไดรฟ์ที่จะติดตั้งต้องมีระบบไฟล์แบบ Windows NT file system (NTFS)

1.2 มีหน่วยความจำหลักอย่างน้อย 16 เมกะไบท์

1.3 เนื้อที่ว่างในฮาร์ดดิสอย่างน้อย 40 เมกะไบท์สำหรับตัวโปรแกรม และ 20 เมกะไบท์สำหรับ dbspace

1.4 โปรโตคอลที่ใช้ในการสื่อสารคือ TCP/IP

2.) ลำดับในการติดตั้งผลิตภัณฑ์

หากต้องการจะติดตั้งผลิตภัณฑ์ Informix หลายตัวในเครื่องเดียวกันต้องจัดลำดับการติดตั้งดังนี้

2.1 Clients ผลิตภัณฑ์ไคลเอนท์ได้แก่ DataBlade Developers Kit, Informix-Connect for Windows เป็นต้น

2.2 Database servers คือระบบจัดการฐานข้อมูล

2.3 Datablade modules

3.) ขั้นตอนการติดตั้ง Universal Server

โปรแกรมติดตั้งจะสร้างชื่อผู้ใช้ใหม่ และ กลุ่มผู้ดูแล-จัดการระบบฐานข้อมูลคือ Informix และ Informix-Adminขึ้นมาระหว่างทำการติดตั้ง โดยผู้ใช้ใหม่จะถูกจัดให้อยู่ในกลุ่ม Informix-Admin และกลุ่ม Admin ของระบบ

ขั้นตอนในการติดตั้ง

3.1 เริ่มต้นด้วยการรันโปรแกรม setup.exe

3.2 ใส่ serial no. และ serial no. key ให้ถูกต้อง

3.3 กรอกข้อมูลเพื่อลงทะเบียนผลิตภัณฑ์ ได้แก่ ชื่อผู้ติดตั้ง, บริษัท เป็นต้น

3.4 กรอกที่อยู่ของผู้ติดตั้ง

3.5 กรอกข้อมูลเพื่อใช้ในการติดต่อกับผู้ติดตั้ง ได้แก่ เบอร์โทรศัพท์, เบอร์โทรสาร หรือ internet address

3.6 ระบุไดเรกทอรี่ที่จะติดตั้ง

3.7 เลือกที่จะแยกหรือไม่แยกบทบาทการทำงานของผู้ดูแล-จัดการฐานข้อมูล

หากเลือกที่จะแยกบทบาทในการดูแลระบบจะต้องปฏิบัติตามขั้นตอนต่อไปนี้เพิ่มเติม

3.7.1 จัดประเภทงานของผู้ดูแลระบบ เช่น งานตรวจสอบระบบ, งานรักษาความปลอดภัยให้กับฐานข้อมูล

3.7.2 ระบุชื่อผู้ตรวจสอบระบบ

3.7.3 ระบุชื่อผู้คอยรักษาความปลอดภัยให้กับฐานข้อมูล

3.8 ตั้งชื่อของ database server

3.9 ระบุที่ตั้งและขนาดของ dbspace ที่จะใช้สำหรับเก็บฐานข้อมูลและตารางข้อมูลต่างๆ

3.10 ระบุที่ตั้งและขนาดของ sbspace เพื่อใช้เก็บ Smart large object

3.11 ระบุข้อมูลของอุปกรณ์ที่ใช้เก็บข้อมูลสำรองได้แก่ ความจุของอุปกรณ์, ที่อยู่ของอุปกรณ์ เพื่อใช้ในการกู้คืนข้อมูลหากระบบพัง

3.12 กำหนด password สำหรับผู้ใช้ชื่อ Informix

3.13 กำหนดที่ตั้งของตาราง sqlhosts

หลังจากการติดตั้งสำเร็จเรียบร้อยแล้ว จะพบกลุ่มของโปรแกรมเหล่านี้ใน Start Menu ของ Windows NT

- **Command-Line Utilities** เป็นหน้าต่าง Ms DOS ใช้สำหรับป้อนคำสั่งจัดการกับฐานข้อมูล เช่น onstat , oncheck และ onspaces
- **DB-Access** เป็นรายการให้เลือกเพื่อเข้าถึงฐานข้อมูล
- **Documentation Notes** เป็นเอกสารอธิบายลักษณะสำคัญของผลิตภัณฑ์
- **Find Error** เป็น help file อธิบายความหมายของ error code ต่างๆ ของ Universal server
- **Release Notes** อธิบายความแตกต่างของผลิตภัณฑ์เวอร์ชันที่เพิ่งติดตั้งเสร็จกับเวอร์ชันก่อนๆ รวมทั้งแจกแจงปัญหาที่เคยเกิดขึ้นและแนวทางการแก้ไขด้วย
- **Uninstall** ใช้เพื่อยกเลิกการติดตั้ง Universal server

4.) เซตให้ database server เริ่มทำงาน

4.1 ขั้นตอนการเซตให้ database server ทำงาน

4.1.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

4.1.2 ดับเบิลคลิกที่ไอคอน Services

4.1.3 เลือก INFORMIX-Universal Server จากกล่องรายการ service

4.1.4 คลิก Start

4.2 หากเป็นการเซตให้ server ทำงานเป็นครั้งแรกต้องทำตามขั้นตอนดังนี้

4.2.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

4.2.2 ดับเบิลคลิกที่ไอคอน Services

4.2.2 เลือก INFORMIX-Universal Server จากกล่องรายการ service

4.2.3 ใส่ข้อความ -iy ลงในกล่อง Startup Parameters

4.2.4 คลิก Start

4.3 ขั้นตอนการเซตให้ server ทำงานโดยอัตโนมัติ

4.3.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

4.3.2 ดับเบิลคลิกที่ไอคอน Services

4.3.3 เลือก INFORMIX-Universal Server จากกล่องรายการ service

4.3.4 คลิก Start และ Automatic ใน service dialog box

4.3.5 จากนั้นคลิก OK

4.3.6 ตรวจสอบว่าไม่มีข้อความใดๆ ในกล่อง Startup Parameters

4.3.7 คลิก Start

5.) ขั้นตอนการเซตให้ database server หยุดทำงาน

5.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

5.2 ดับเบิลคลิกที่ไอคอน Services

5.3 เลือก INFORMIX-Universal Server จากกล่องรายการ service

5.4 คลิก Stop

สร้างฐานข้อมูล ขั้นตอนการสร้างฐานข้อมูลลงใน database server

1.) สร้างไฟล์เปล่าขึ้นมาโดยใช้คำสั่ง copy con ดังนี้

d:\> copy con ชื่อไฟล์

กด ctrl+d แล้วกดปุ่ม Enter

2.) เรียกโปรแกรม command-line utilities

3.) ป้อนคำสั่งสร้าง dbspace ดังนี้

d:\> onspaces -c -d ชื่อdbspace -p ชื่อไฟล์เปล่า -o 5000 -s 10000

4.) เรียกโปรแกรม DB-access และเลือกเมนูสร้างฐานข้อมูล

สร้าง Sbspace เพื่อใช้เก็บข้อมูลแบบ Smart large object ขั้นตอนการสร้าง

1.) สร้างไฟล์เปล่าขึ้นมาโดยใช้คำสั่ง copy con ดังนี้

2.) เรียกโปรแกรม command-line utilities

3.) ป้อนคำสั่งสร้าง sbspace ดังนี้

d:\> onspaces -c -S sbspace -g 5 -p ชื่อไฟล์เปล่า -o 500 -s 20000 -Ms 150 -Mo 200

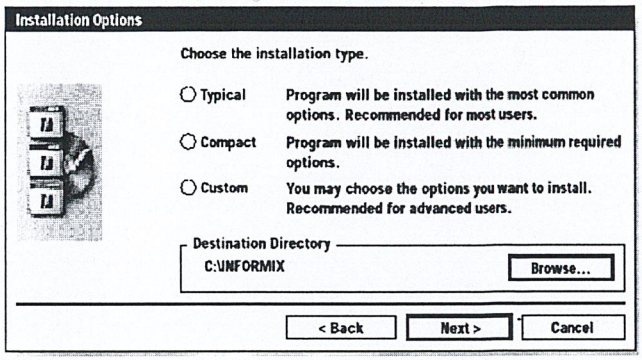
การติดตั้งโปรแกรมไคลเอนท์ ในการทำโครงงานครั้งนี้ได้เลือกทำการติดตั้งโปรแกรมไคลเอนท์สองตัวคือ

1. DataBlade Developers Kit เป็นเครื่องมือที่ใช้สร้าง module
2. Informix-Connect for Windows Version2.0 ถูกติดตั้งเพื่อให้เครื่องไคลเอนท์ติดต่อกับ database server ได้

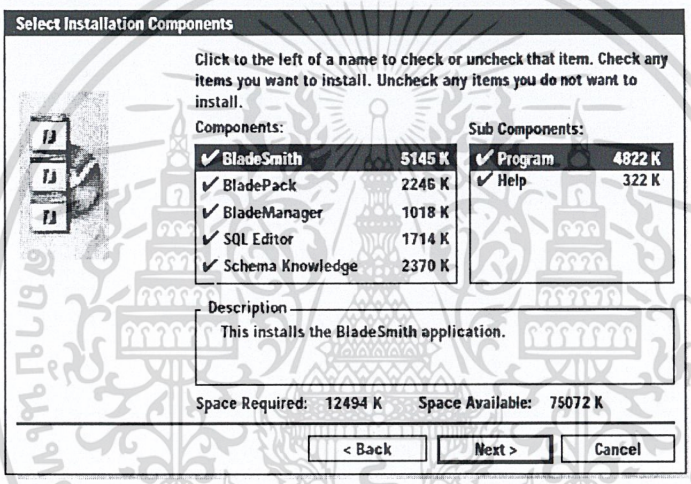
การติดตั้งโปรแกรม DataBlade Developers Kit ขั้นตอนในการติดตั้ง

1. เรียกโปรแกรม setup.exe
2. ใส่ serial no. และ serial no. key ให้ถูกต้อง เสร็จแล้วคลิก Next

3. กรอกข้อมูลของผู้ใช้ ได้แก่ ชื่อผู้ใช้ และ บริษัท แล้วคลิก Next จะปรากฏ dialog box ดังนี้



4. เลือกรูปแบบของการติดตั้งและไดเรกทอรี่ที่จะทำการติดตั้ง แล้วคลิก Next หากเลือกที่จะทำการติดตั้งแบบ Custom เมื่อคลิก Next แล้วจะปรากฏ dialog box ดังนี้

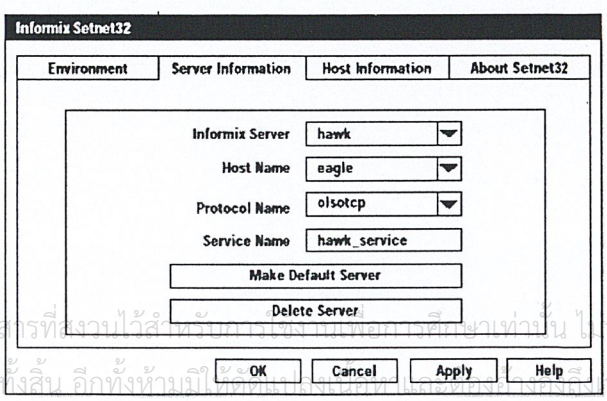


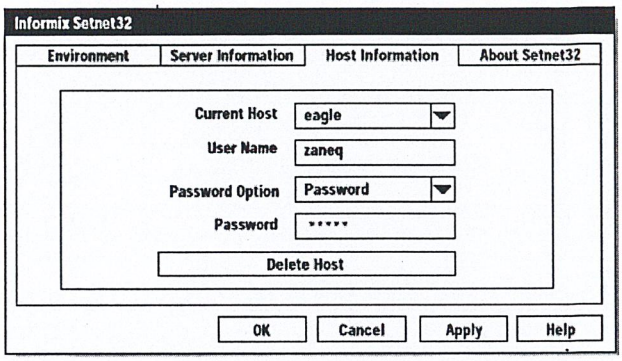
ให้ทำการเลือกคอมโพเนนท์ที่จะทำการติดตั้ง แล้วคลิก Next

5. คลิก Finish เพื่อจบการติดตั้ง

ในการติดตั้งโปรแกรม โปรแกรม Setup จะทำการคัดลอกซอฟแวร์ของผลิตภัณฑ์ไคลเอนท์ลงในเครื่องคอมพิวเตอร์ สร้างไอคอน Setnet32 และไอคอนต่างๆของผลิตภัณฑ์ ใน Start Menu

6. กำหนดข้อมูลต่างๆของ server และ host โดยใช้โปรแกรม Setnet32 ดังตัวอย่างในรูปต่อไปนี้





ชุดของโปรแกรมของ DataBlade Developers Kit ประกอบด้วย

- **BladeSmith** เป็นเครื่องมือสำหรับใช้สร้าง DataBlade Module โดย BladeSmith จะทำการสร้างไฟล์ต่างๆเพื่อนิยามโมดูลที่ถูกสร้างขึ้นดังนี้
  1. SQL Scripts วิธีในการขึ้นทะเบียนโมดูลกับฐานข้อมูล โดย Scripts นี้จะถูกเรียกโดย BladeManager อีกทีหนึ่ง
  2. Source file เป็นโค้ดของรูทีนที่ถูกกำหนดไว้ในโมดูล
  3. Setup file ใช้สำหรับสร้างไฟล์ในการติดตั้งโมดูล
  4. Functional test file
- **BladePack** จะเรียก setup file ขึ้นมาเพื่อทำการสร้างไฟล์ติดตั้งโมดูล
- **BladeManager** ใช้ขึ้นทะเบียนโมดูลกับฐานข้อมูล
- **SQL Editor** เป็นเครื่องมือใช้เรียกดูข้อมูล ในการเรียกดูต้องเขียนคำสั่งภาษา SQL ลงไปใน editor นี้และสั่งรัน
- **Schema Knowledge** เป็นเครื่องมือสำหรับดูว่าในฐานข้อมูลมีอ็อบเจคอะไรบ้าง อ็อบเจคที่ว่าได้แก่ ตาราง, รูทีนต่างๆ เป็นต้น

หมายเหตุ DataBlade Module โดยทั่วไปประกอบด้วย กลุ่มของชนิดข้อมูล (User-Defined Type) และ รูทีน

ที่ใช้จัดการกับข้อมูลในโมดูลนั้น ส่วนใหญ่แล้ว โมดูลเหล่านี้จะถูกสร้างขึ้นมาเพื่อใช้ในงานเฉพาะ

สร้างและเรียกใช้ DataBlade Module

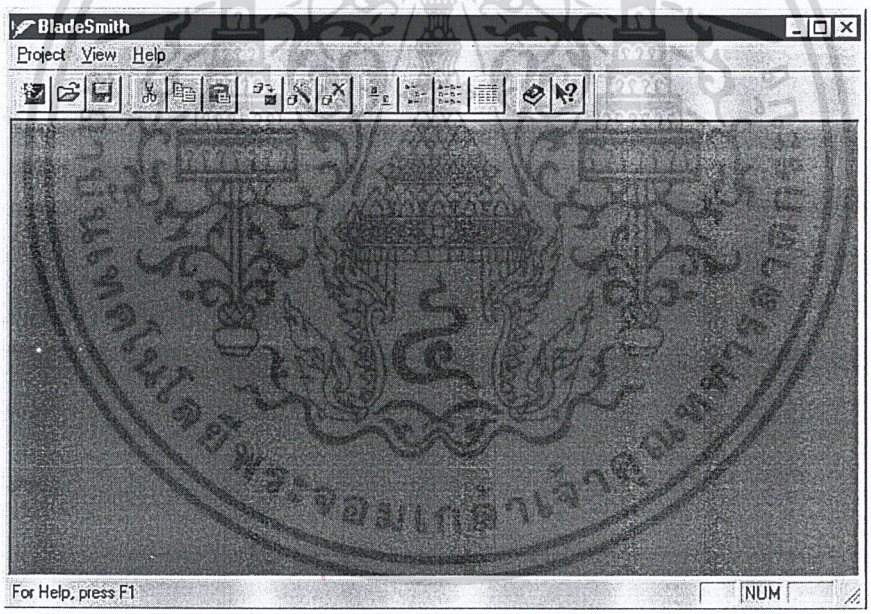
1.) ตัวอย่างการสร้างโมดูล

ในตัวอย่างนี้จะแสดงการสร้างโมดูลชื่อ test ที่ประกอบด้วย

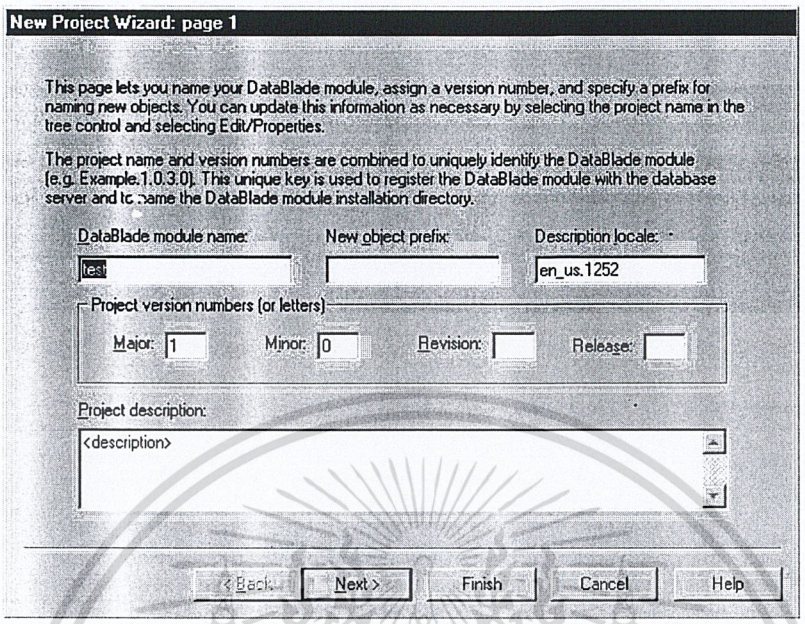
- ชนิดข้อมูลแบบ Opaque ที่ชื่อว่า "test" ซึ่งมีส่วนประกอบย่อยอีกสองส่วนคือ
    1. ชนิดข้อมูลแบบ MI_LO_HANDLE ชื่อ "txt" (ข้อมูลประเภทอักขระ)
    2. ชนิดข้อมูลแบบ MI_LO_HANDLE ชื่อ "img" (ข้อมูลประเภทรูปภาพ)
- รูทีนที่ใช้จัดการกับชนิดข้อมูลที่สร้างขึ้น ได้แก่
    1. txt_to_file สร้างเพื่อใช้นำข้อมูลชนิด "txt" มาเก็บในไฟล์ .txt
    2. img_to_file สร้างเพื่อใช้นำข้อมูลชนิด "img" มาเก็บในไฟล์ .bmp
    3. get_txt ใช้เพื่อนำข้อมูลประเภทอักขระที่ถูกเก็บไว้มาแสดง

1.1) ขั้นตอนการสร้างโมดูลใหม่โดยใช้ BladeSmith

1.1.1) เรียกโปรแกรม BladeSmith สร้างโมดูล โดยคลิกไอคอนจาก Start Menu จะปรากฏหน้าต่างดังนี้

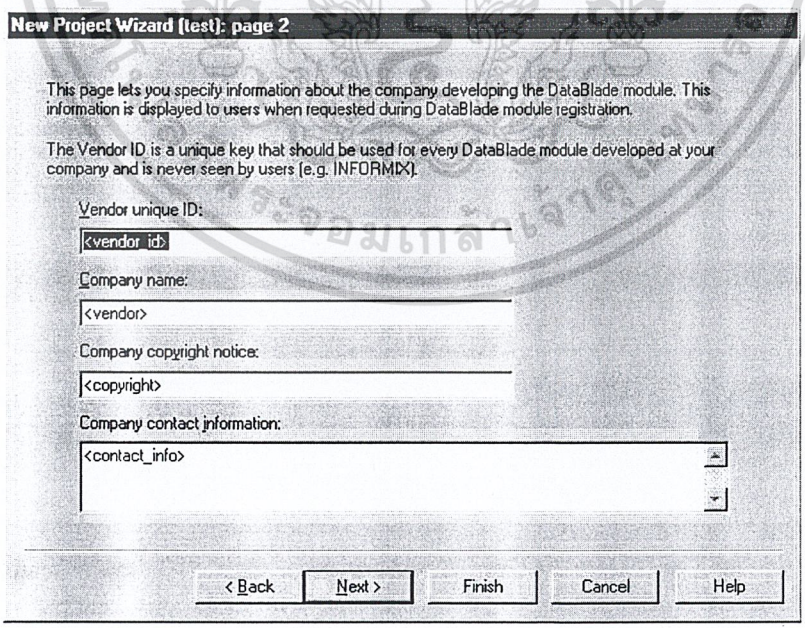


1.1.2) เริ่มสร้างโมดูลโดยเลือกรายการ Project -> New จาก menu bar จะปรากฏไดอะล็อกให้กรอกชื่อโมดูลดังนี้

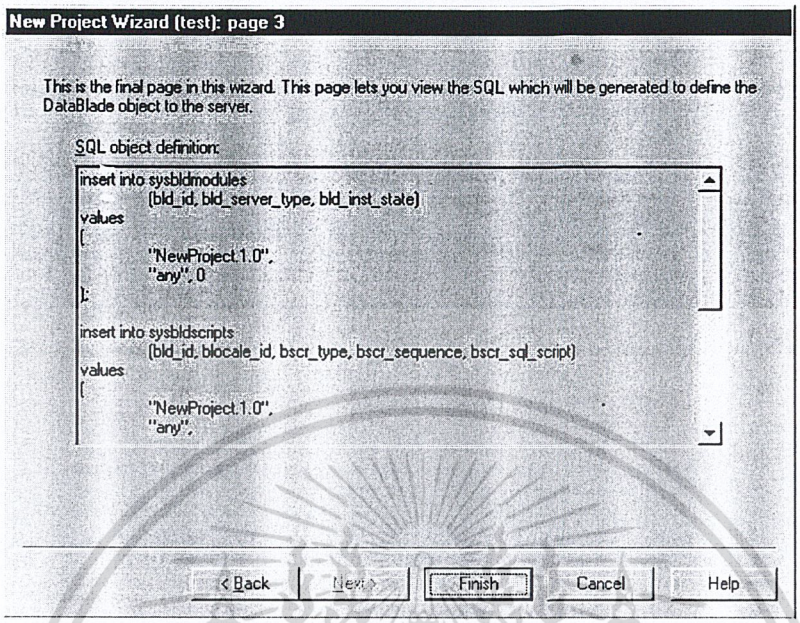


หลังจากกรอกชื่อโมดูล "test" เรียบร้อยแล้ว คลิกปุ่ม Next

1.1.3) กรอกข้อมูลเกี่ยวกับบริษัทของผู้สร้าง แล้วคลิก Next

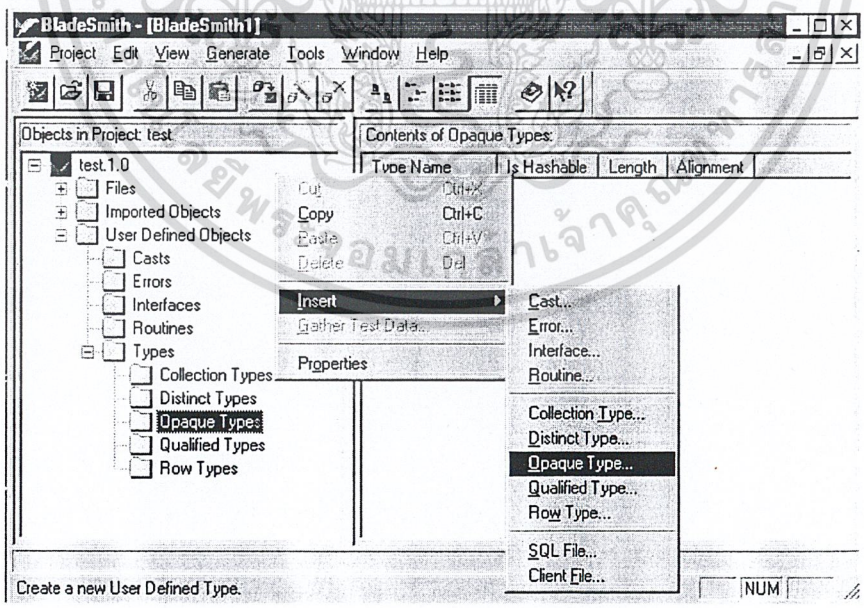


1.1.4) คลิก Finish ในไดอะล็อกแสดงโค้ด SQL ที่จะใช้ขึ้นทะเบียนโมดูลกับ server

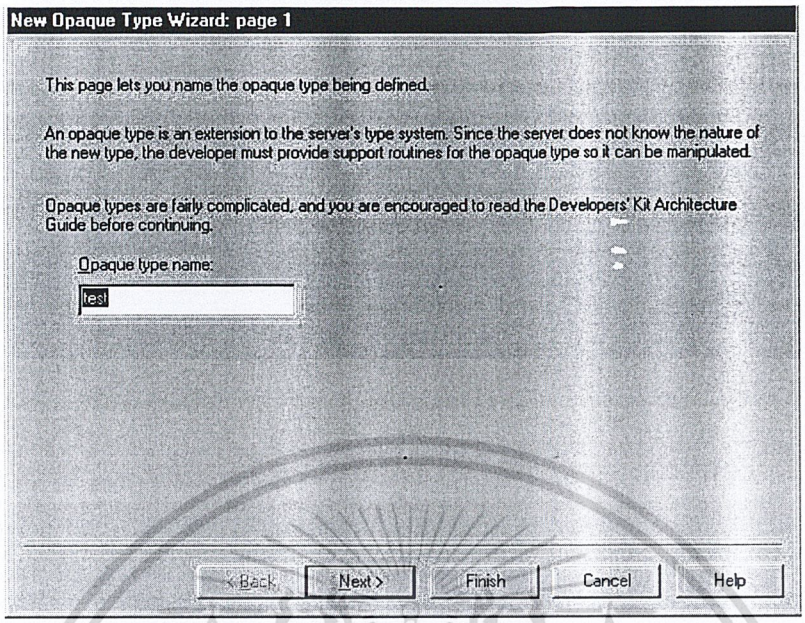


1.2) ขั้นตอนการสร้าง **Opaque Type** ลงในโมดูล

1.2.1) เริ่มสร้างชนิดข้อมูลแบบ opaque โดยคลิกขวาที่หน้าต่าง และที่รายการคำสั่ง insert ให้คลิกเลือกรายการ opaque type ดังรูป

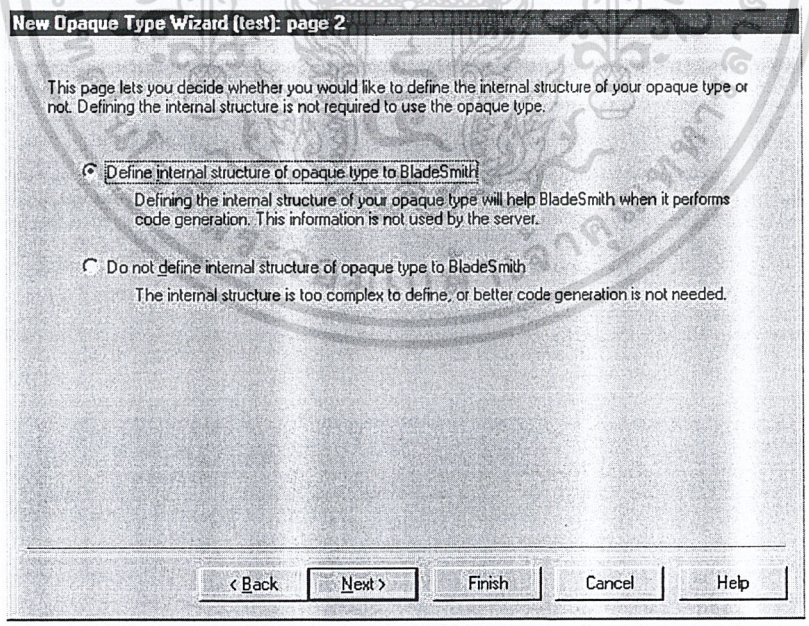


1.2.2) กำหนดชื่อให้กับชนิดข้อมูลแบบ opaque

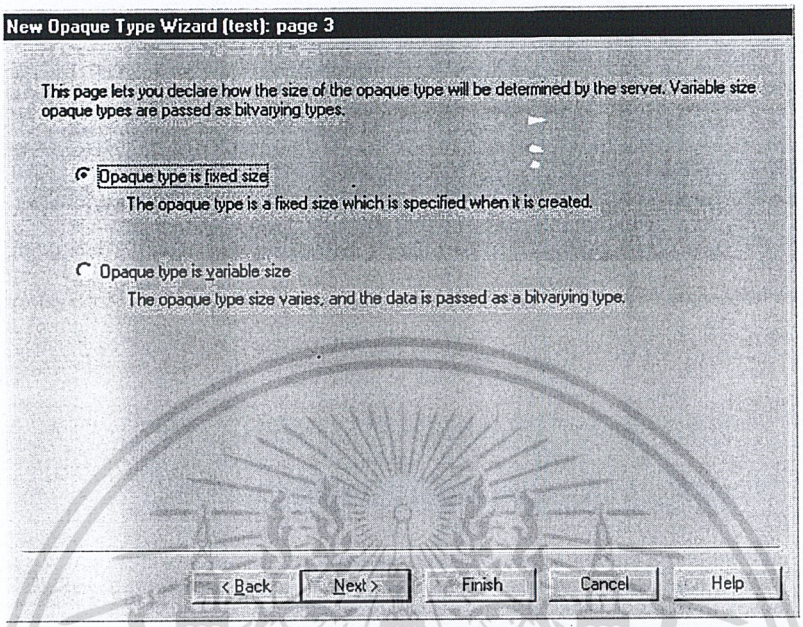


คลิก Next หลังจากที่กรอกชื่อ “test” ลงในไดอะล็อกเรียบร้อยแล้ว

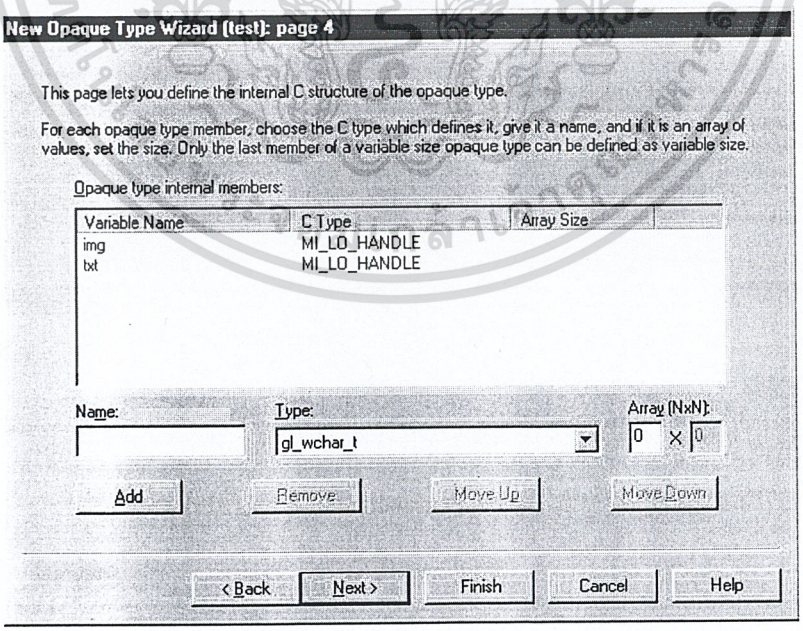
1.2.3) เลือกให้ BladeSmith สร้างโค้ดกำหนดโครงสร้างภายในของ opaque type จากไดอะล็อกด้านล่าง เสร็จแล้วคลิก Next



1.2.4) กำหนดให้ข้อมูลชนิด opaque type มีขนาดไม่เกิน 32 KB โดยเลือกข้อความ “Opaque type is fixed size” แล้วคลิก Next

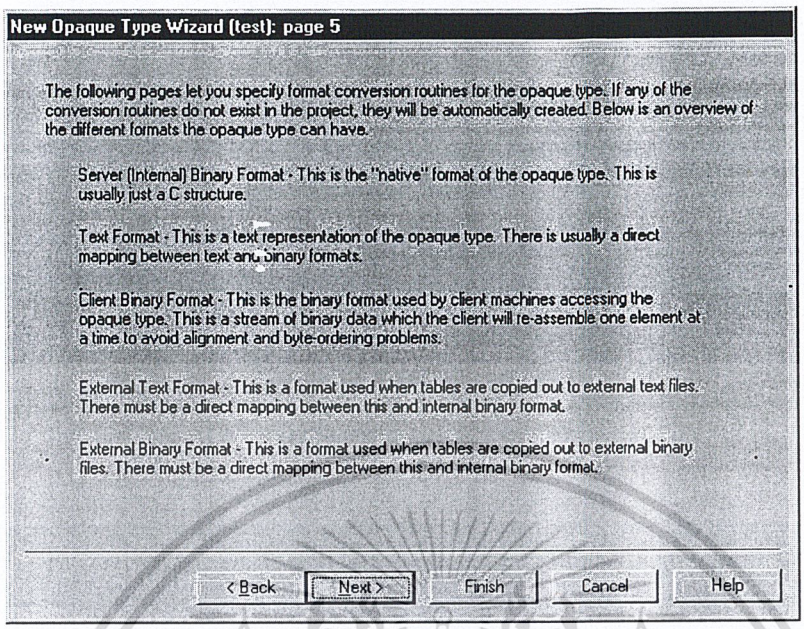


1.2.5) กำหนดสมาชิกของ opaque type โดยใส่ ชื่อและเลือกโครงสร้างภาษาซี ให้กับสมาชิกแต่ละตัว แล้วคลิกปุ่ม Add

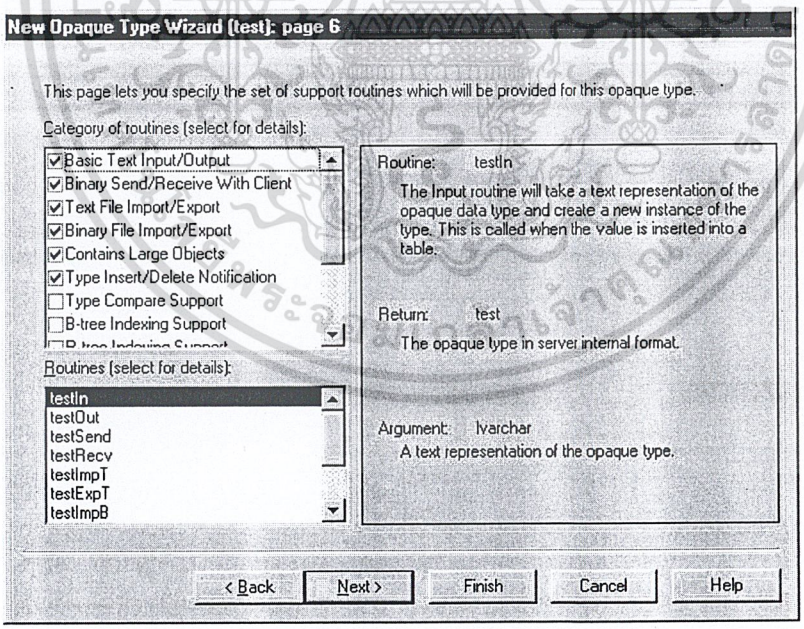


เมื่อกำหนดสมาชิกคือ img และ txt ครบตามที่ต้องการแล้ว ให้คลิก Next

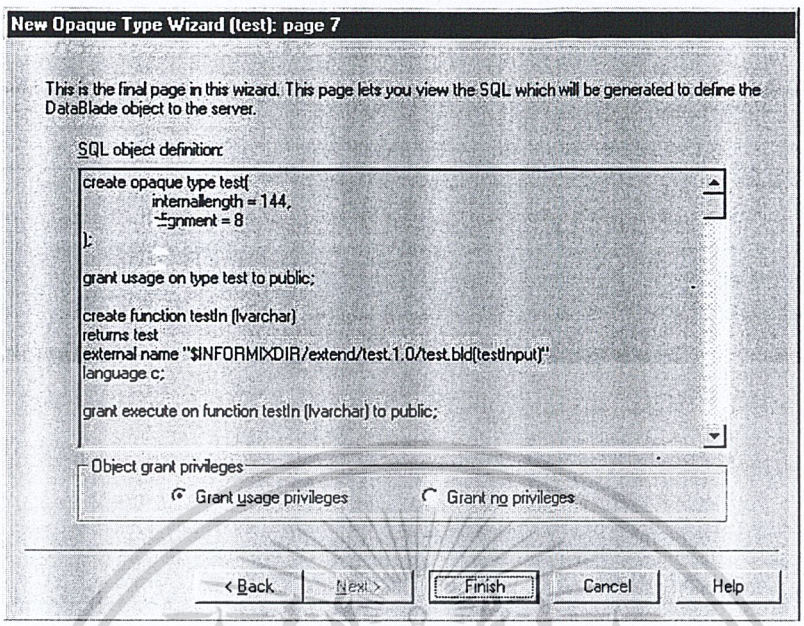
1.2.6) คลิกปุ่ม Next



1.2.7) เลือกรูทีนที่ใช้จัดการกับ opaque type โค้ดของรูทีนที่เลือกจะถูก BladeSmith สร้างให้โดยอัตโนมัติ

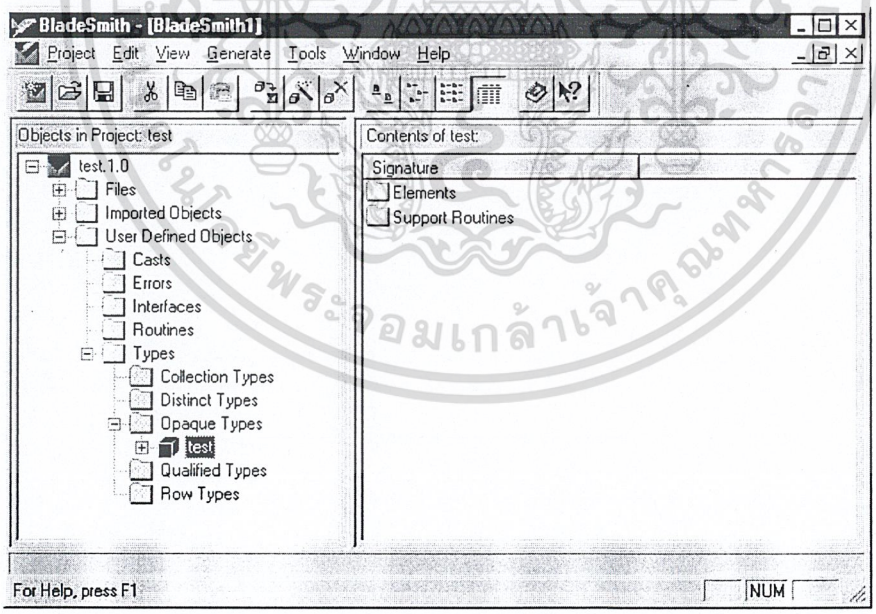


1.2.8) สามารถอนุญาตให้ผู้อื่นนำชนิดข้อมูลที่สร้างขึ้นไปใช้ได้ โดยกำหนดในไดอะล็อกดังรูปด้านล่าง



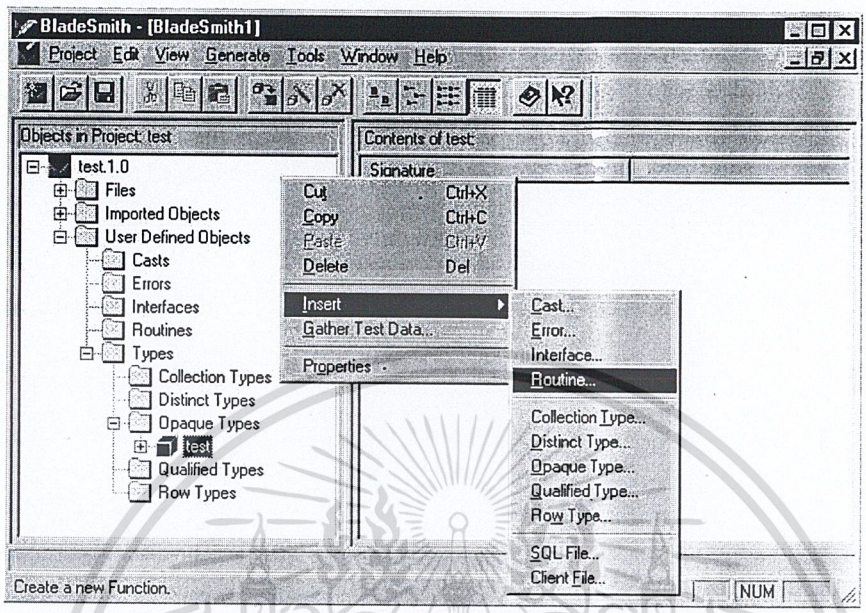
คลิกปุ่ม Finish เพื่อจบการสร้าง

1.2.9) หลังจากจบการสร้างจะปรากฏ opaque type ชื่อ test ในหน้าต่างของโมดูล test ดังนี้

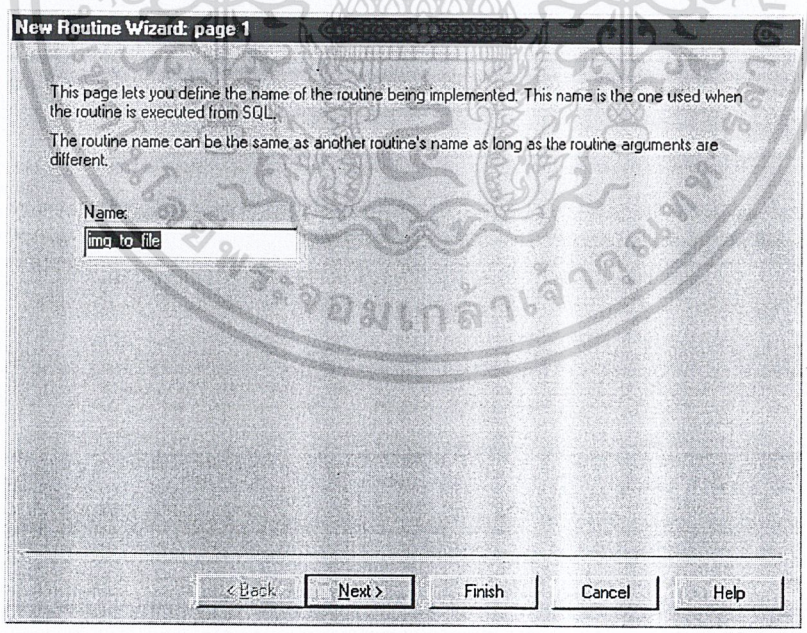


1.3) ขั้นตอนการสร้างรูทีนจัดการข้อมูล ลงในโมดูล

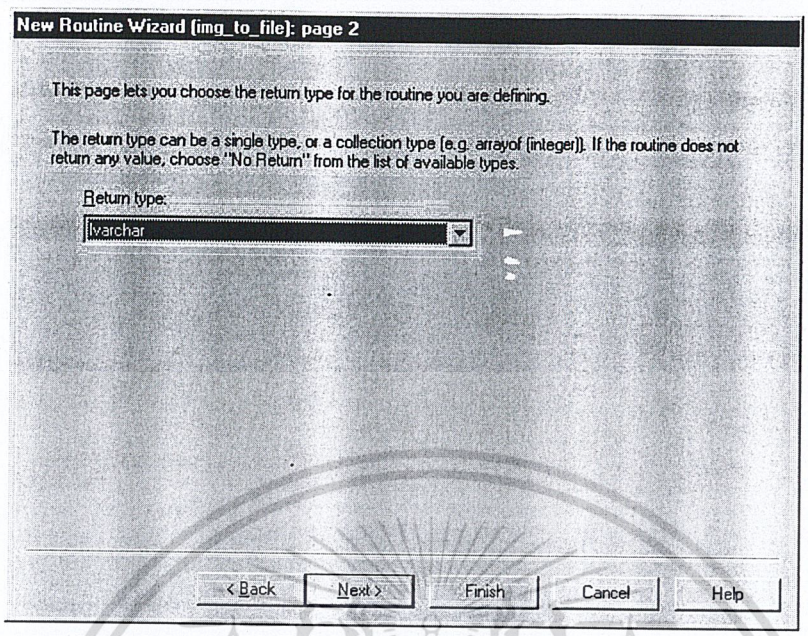
1.3.1) เริ่มสร้างรูทีน โดยคลิกขวาที่หน้าต่าง และที่รายการคำสั่ง insert ให้คลิกเลือก รายการ Routine ดังรูป



1.3.2) กำหนดชื่อให้กับรูทีน แล้วคลิก Next

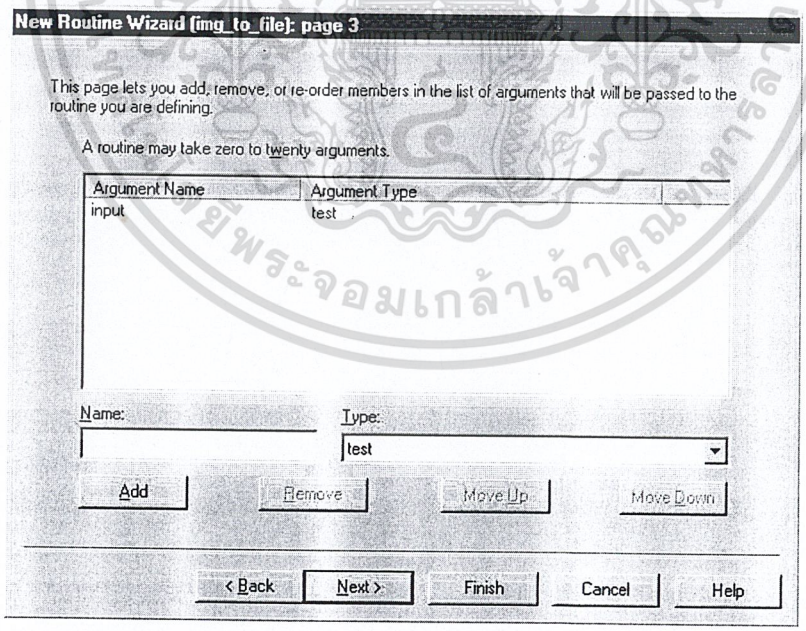


1.3.3) กำหนดชนิดข้อมูลที่รูทีนจะต้องส่งกลับมา โดยเลือกจากรายการในช่อง Return type



เมื่อเลือกรายการเสร็จแล้ว คลิก Next

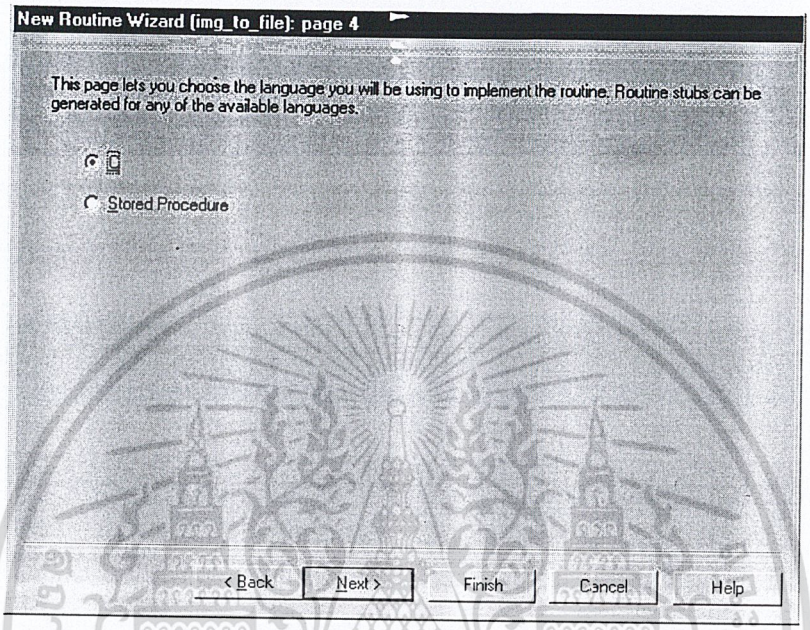
1.3.4) กำหนดตัวแปรอินพุทของรูทีน โดยใส่ชื่อตัวแปรในช่อง Name และเลือกชนิดของตัวแปรจากช่อง Type แล้วคลิกปุ่ม Add



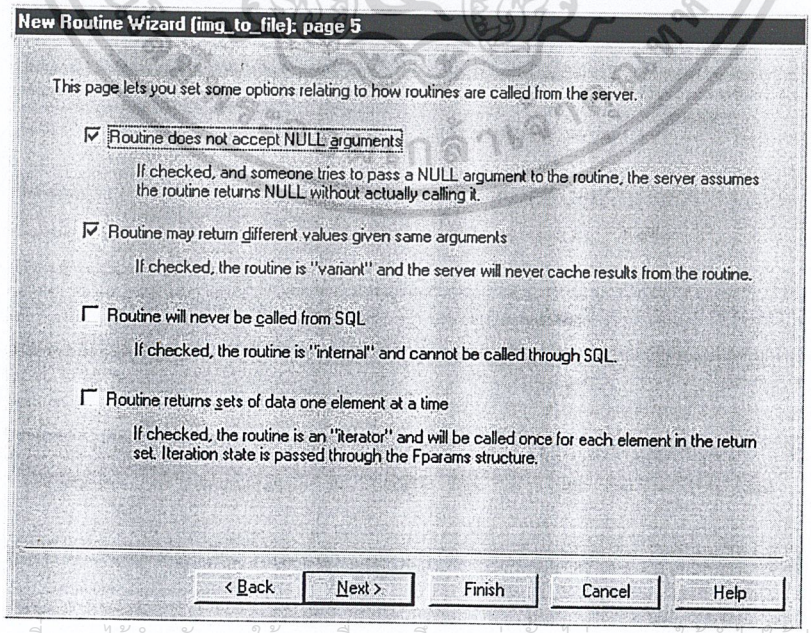
เมื่อกำหนดอินพุทครบตามที่ต้องการแล้ว คลิกที่ปุ่ม Next

1.3.5) เลือกว่าจะสร้างรูทีนด้วยภาษาใดในสองภาษาต่อไปนี้

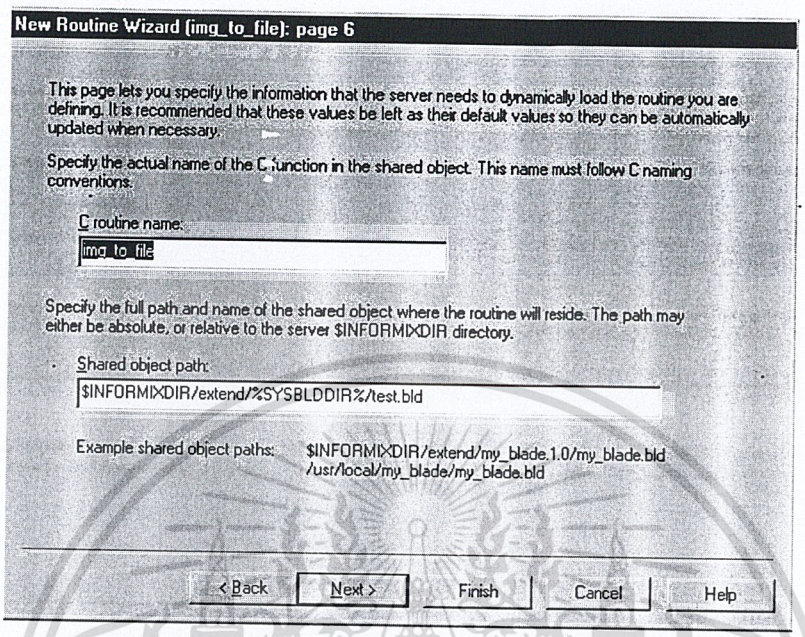
- *ภาษาซี* หากเลือกสร้างรูทีนด้วยภาษาซี จะต้องไปทำการเพิ่มโค้ดของรูทีนใน source file ที่ BladeSmith จะสร้างขึ้นมาให้ภายหลัง
- *ภาษา SPL(stored procedure language)* รูทีนจะถูกสร้างขึ้นจากภาษา SQL โค้ดของรูทีนจะต้องถูกเขียนลงในไดอะล็อกถัดไปหลังจากที่คลิกปุ่ม Next



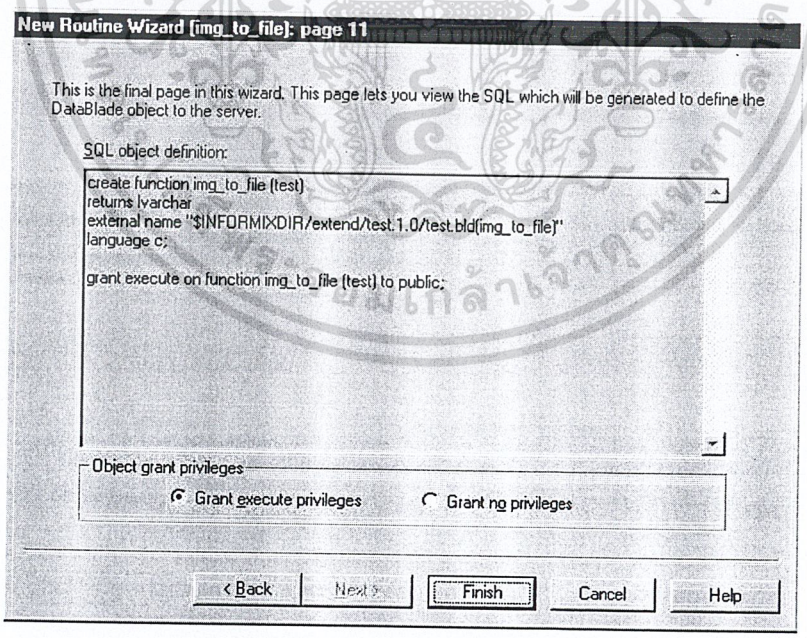
1.3.6) เลือกกำหนดอ็อพชันของรูทีนตามรายการที่จัดไว้ให้ในไดอะล็อกด้านล่าง



1.3.7) หากในขั้นตอนที่ 1.3.5 ได้เลือกสร้างโค้ดของรูทีนด้วยภาษาซี จะปรากฏไดอะล็อกดังรูปในขั้นตอนนี้ เพื่อให้ทำการกำหนดที่อยู่ของ shared object และ ชื่อของรูทีนที่จะใช้ใน shared object นั้น

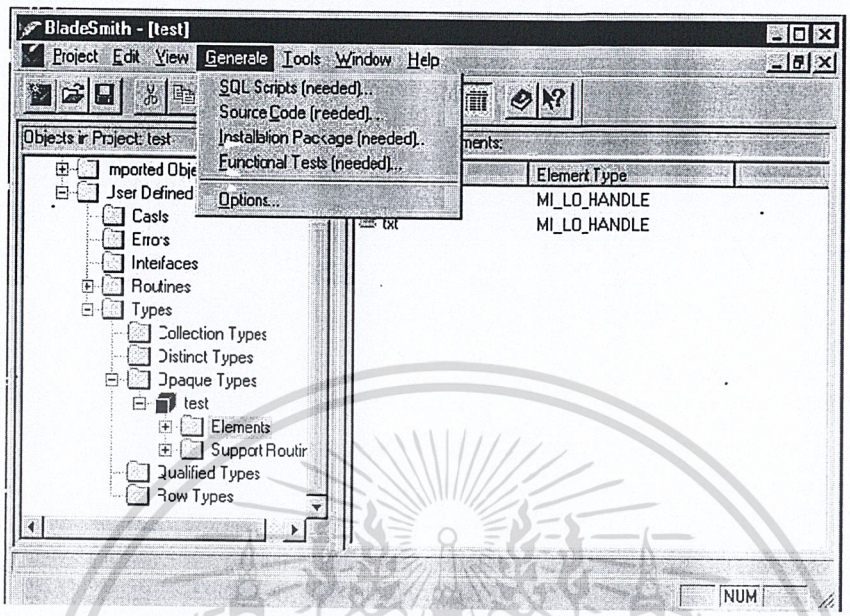


1.3.8) คลิกปุ่ม Next ในไดอะล็อกถัดไปจนกระทั่งถึงไดอะล็อกสุดท้ายดังรูป



สามารถกำหนดให้ผู้ใช้อื่นเรียกใช้รูทีนนี้ได้โดยเลือกคลิกที่ข้อความ “Grant execute priviledges” เสร็จแล้ว คลิก Finish เพื่อจบการสร้างรูทีน

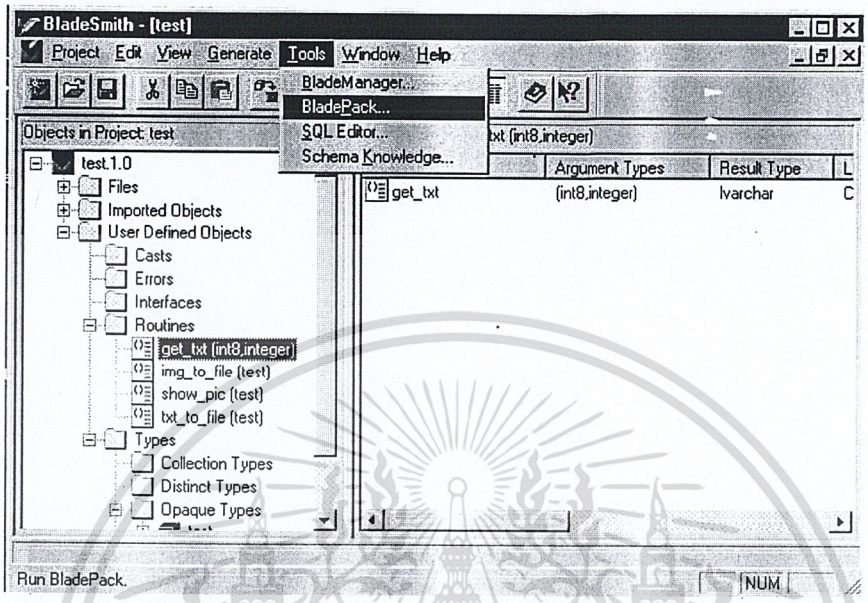
หลังจากที่ทำการสร้างโมดูลเรียบร้อยแล้วให้ทำการเลือกคำสั่งในรายการคำสั่ง Generate บน menu bar ของหน้าต่างที่ปรากฏดังรูป เพื่อสร้างโค้ดสามชนิดได้แก่ SQL Scripts, Source Code และ Installation Package



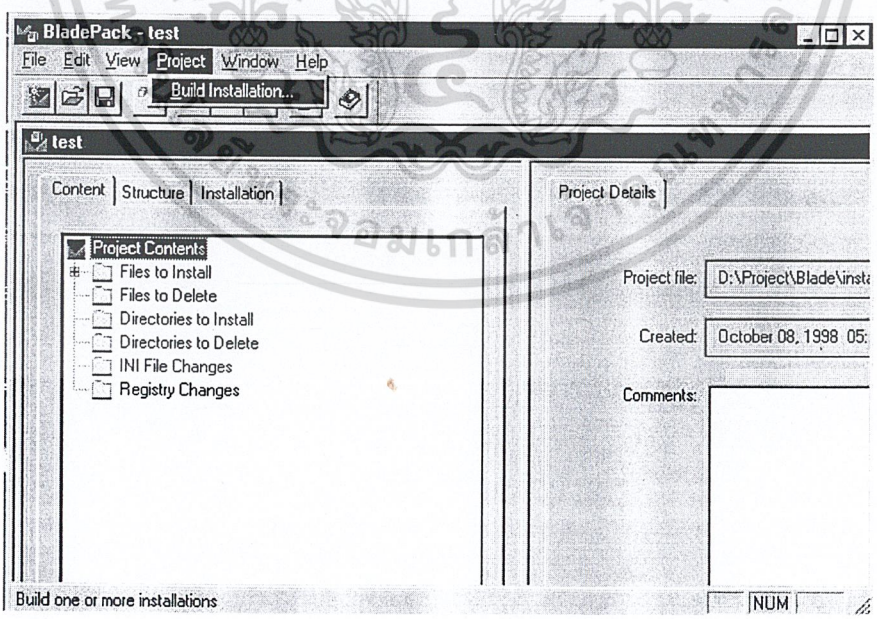
2.) สร้างไฟล์ติดตั้งโมดูลโดยใช้ BladePack

เมื่อทำการสร้างโมดูลใหม่เสร็จแล้ว ขั้นตอนต่อไปที่จะต้องทำคือ ใช้ BladePack สร้างไฟล์ติดตั้งโมดูล ขั้นตอนการใช้งาน BladePack มีดังนี้

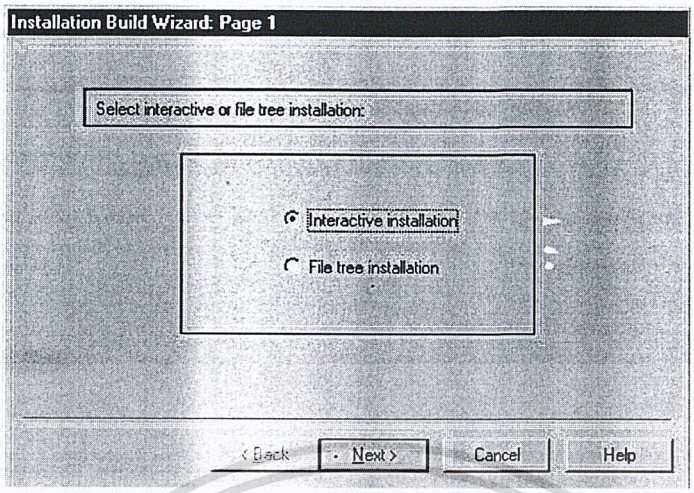
2.1) เรียก BladePack จากรายการ Tool บน menu bar ของ BladeSmith ดังรูป



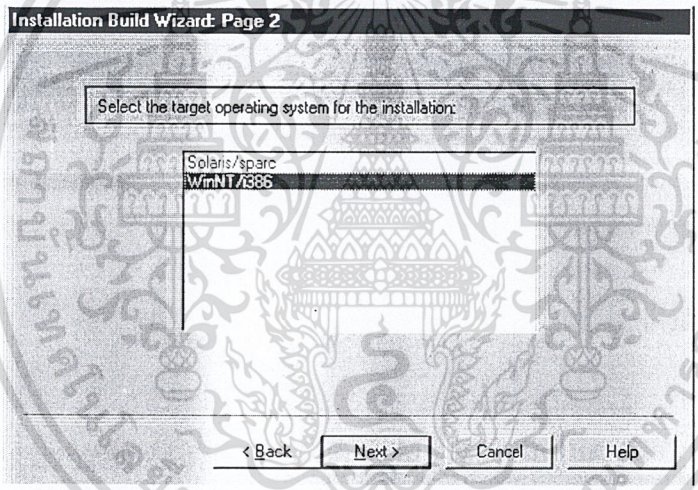
2.2) เริ่มสร้างไฟล์โดยเลือกรายการ Project->Build Installation บน menu bar ในหน้าต่างโปรแกรม BladePack ดังรูป



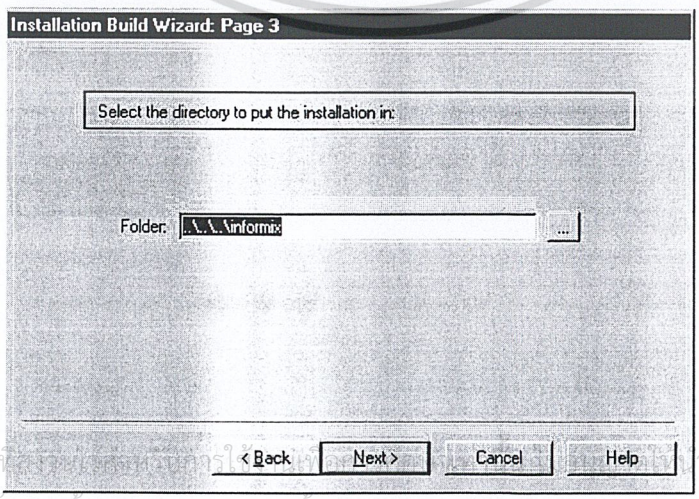
2.3) เลือกประเภทการติดตั้งแบบ file tree แล้วคลิก Next



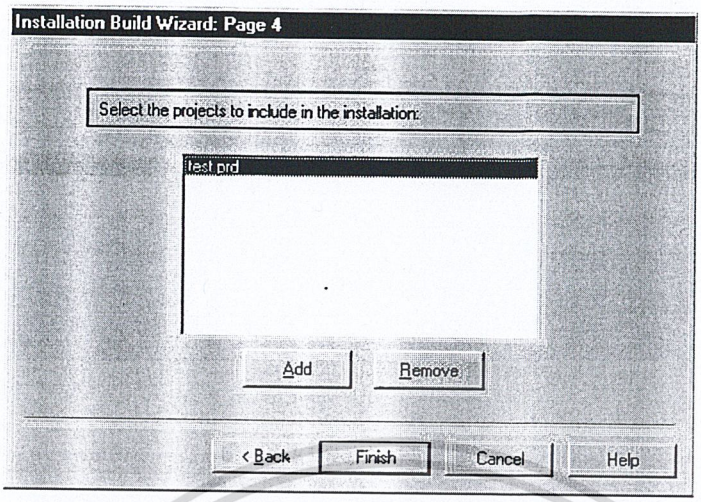
2.4) เลือกระบบปฏิบัติการสำหรับการติดตั้ง แล้วคลิก Next



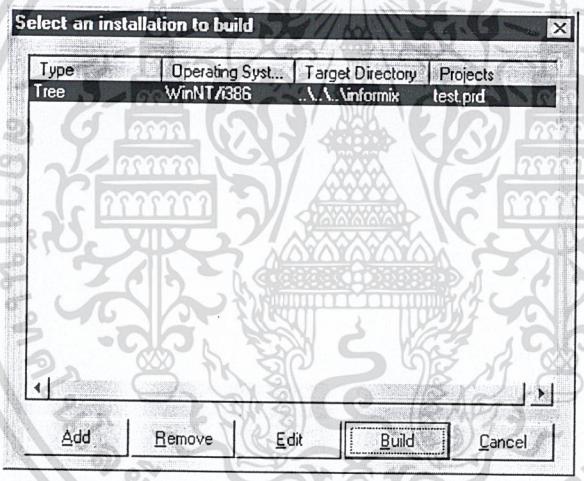
2.5) เลือกไดเรกทอรี่ที่จะทำการติดตั้งโมดูล แล้วคลิก Next



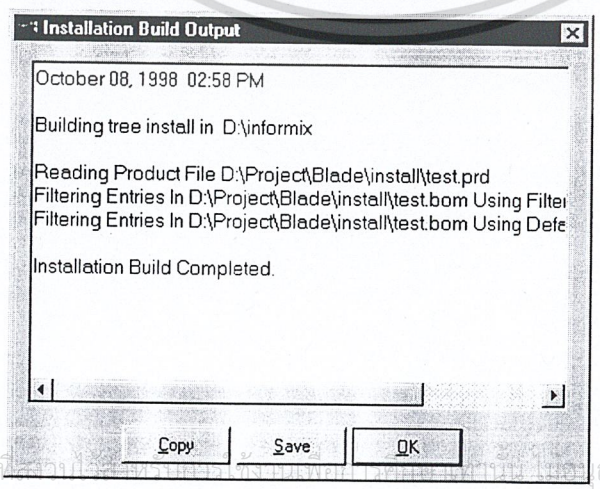
2.6) คลิก Finish



2.7) คลิกปุ่ม Build

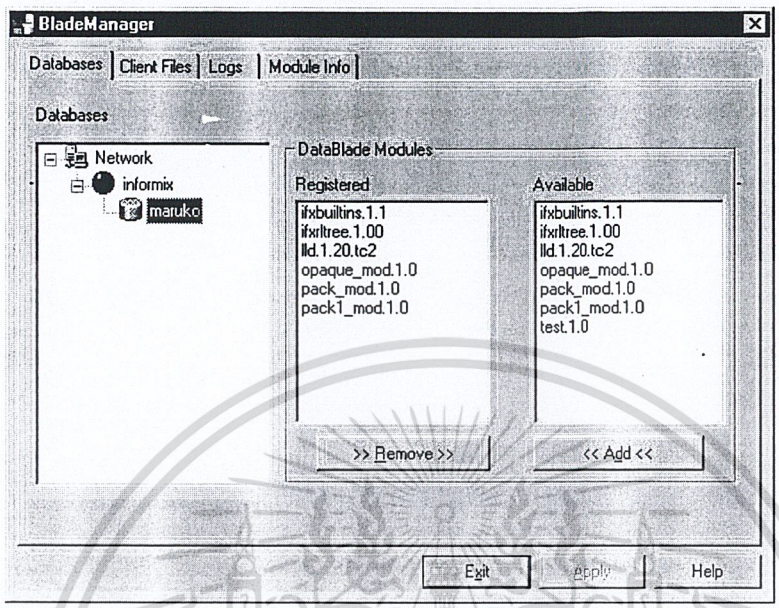


2.8) หากการสร้างไฟล์สำเร็จ จะปรากฏไดอะล็อกดังรูป



3) ขึ้นทะเบียนโมดูลกับฐานข้อมูลโดยใช้ BladeManager

ทำการขึ้นทะเบียนโมดูลกับฐานข้อมูลโดยเรียกโปรแกรม BladeManager จาก Start Menu จะปรากฏหน้าต่างดังนี้



ให้เลือกโมดูลจากกล่องรายการ Available แล้วคลิกปุ่ม Add ชื่อโมดูลที่เลือกจะไปปรากฏในกล่องรายการ Registered ทางซ้ายมือ จากนั้นให้คลิกปุ่ม Apply เพื่อทำการขึ้นทะเบียน

4.) ตัวอย่างการสร้างตารางเรียกดูข้อมูลจากตาราง

ในตารางที่จะสร้างต่อไปนี้จะประกอบด้วยสองคอลัมน์คือ name และ tower โดย

- *คอลัมน์ name* มีชนิดข้อมูลเป็นสตริงขนาด 15 ตัวอักษร
- *คอลัมน์ tower* มีชนิดข้อมูลเป็น opaque type ชื่อ test ที่เราได้สร้างขึ้นในโมดูล test

SQL Editor จะถูกเรียกจาก Start Menu เพื่อรอรับ คำสั่งภาษา SQL และ แสดงผลที่เกิดจากการปฏิบัติคำสั่ง รูปต่อไปนี้แสดงหน้าต่างของโปรแกรม SQL Editor ที่รับและแสดงผลการปฏิบัติคำสั่งที่ใช้สร้างตารางชื่อ test และคำสั่งใส่ข้อมูลลงในตาราง

หมายเหตุ ก่อนที่จะรันคำสั่งใน SQL Editor ต้องทำการเชื่อมต่อกับฐานข้อมูลโดยการคลิกเลือกในช่อง Server/Database เสียก่อน

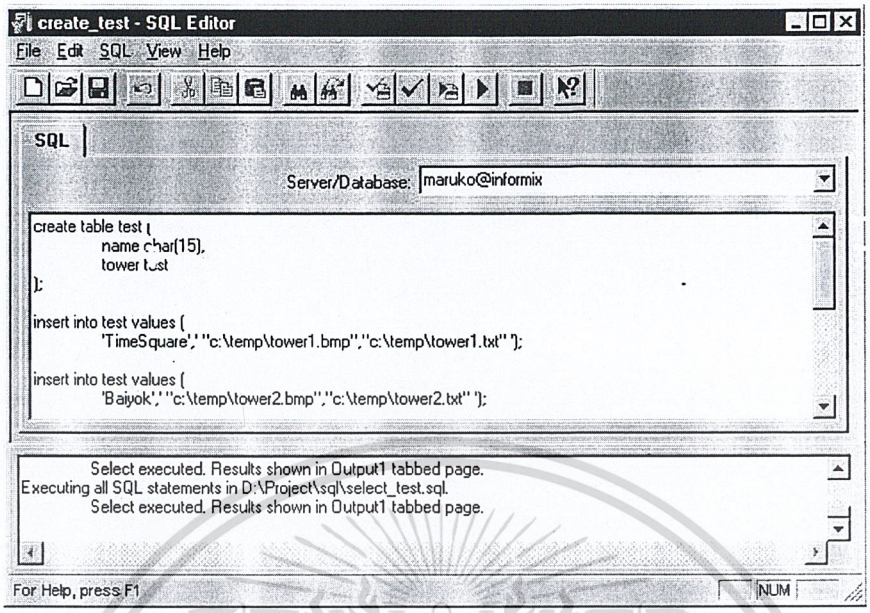
สร้างตาราง test

```
CREATE TABLE test (
        Name char(15),
        Tower test    );
```

ใส่ข้อมูลลงในตาราง

```
INSERT INTO test VALUES (
'TimeSquare', '"c:\temp\tower1.bmp"', '"c:\temp\tower1.txt"' )
```

หน้าต่างของ SQL Editor

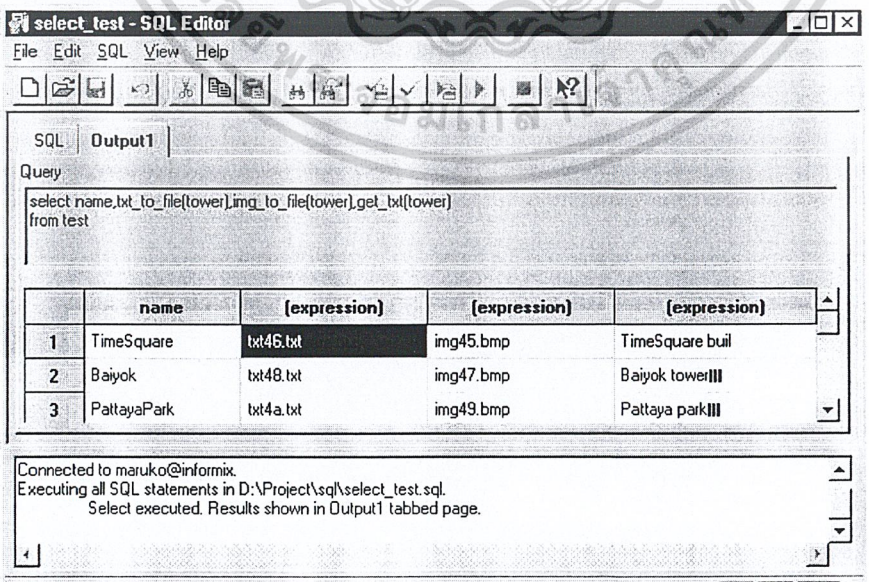


ในการเข้าถึงข้อมูลชนิด opaque type จะต้องเรียกใช้รูทีนที่เราสร้างขึ้นในโมดูล test ได้แก่ txt_to_file, img_to_file และ get_txt

ตัวอย่าง เรียกใช้รูทีนในลักษณะเดียวกับ built-in function ของภาษา SQL

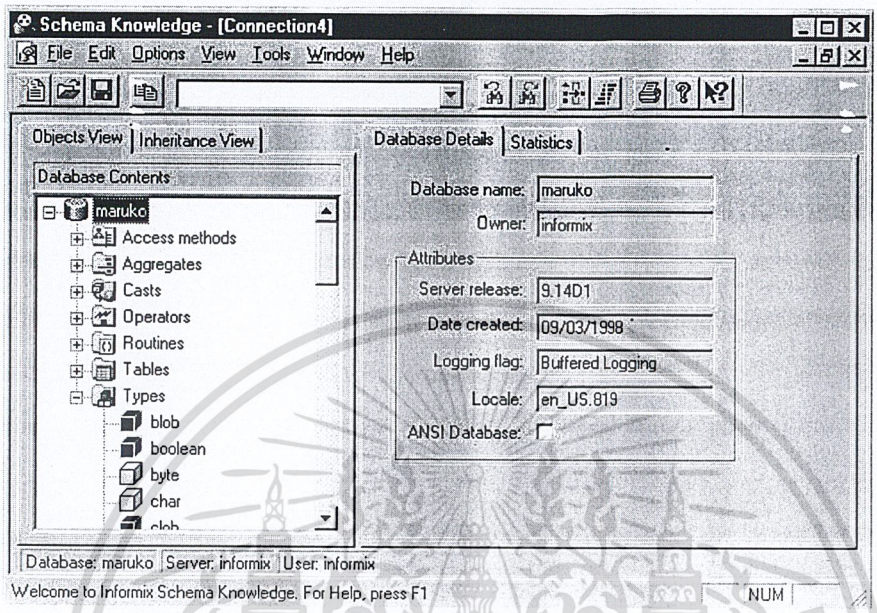
```
SELECT name , txt_to_file(tower) , img_to_file(tower) , get_txt(tower)
FROM   test
```

เมื่อรันคำสั่งตามตัวอย่างจะได้ผลดังรูปต่อไปนี้



5.) โปรแกรม Schema Knowledge

เราสามารถดูว่ามีอ็อบเจคใดถูกเก็บไว้ในฐานข้อมูลที่สร้างขึ้นบ้างจากหน้าต่างของโปรแกรม Schema Knowledge ดังรูป



Support Function ของ Opaque Type

Function	Purpose
Input	แปลงข้อมูลชนิดโอเปคจาก external representation ให้เป็น internal representation
Output	แปลงข้อมูลชนิดโอเปคจาก external representation ให้เป็น internal representation
Receive	แปลงข้อมูลชนิดโอเปคจาก internal representation บนไคลเอนท์คอมพิวเตอร์ให้เป็น internal representation บนเซิร์ฟเวอร์คอมพิวเตอร์
Send	แปลงข้อมูลชนิดโอเปคจาก internal representation บนเซิร์ฟเวอร์คอมพิวเตอร์ให้เป็น internal representation บนไคลเอนท์คอมพิวเตอร์
Import	Performs any tasks needed to process an opaque type when a bulk copy imports the opaque type in its external representation.
Export	Performs any tasks needed to process an opaque type when a bulk copy imports the opaque type in its external representation.
Importbinary	Performs any tasks needed to process an opaque type when a bulk copy imports the opaque type in its external representation.
Exportbinary	Performs any tasks needed to process an opaque type when a bulk copy imports the opaque type in its external representation.
Assign()	Performs any tasks needed to process an opaque type before storing it to disk.
Destroy()	Performs any tasks needed to process an opaque type necessary before the database server removes a row that contains the type.
Iohandles()	ส่งค่ารายชื่อของ Smart Large Object ติดอยู่กับ Opaque type.
Compare()	เปรียบเทียบค่าข้อมูลของ Opaque type สองค่าระหว่างการจัดเรียง (Sort)

ความหมายของค่า ALIGNMENT ใน Memory Alignment

ALIGNMENT

Value	**Meaning**	**Purpose**
1	Align structure on single-byte	Structures that begin with 1-byte quantities boundary
2	Align structure on 2-byte boundary	Structures that begin with 2-byte quantities such as **mi_unsigned_smallint**
4	Align structure on 4-byte boundary	Structures that begin with 4-byte quantities such as float or **mi_unsigned_integer**
8	Align structure on 8-byte boundary	Structures that contain members of the C double data type.

ภาคผนวก ข

มาตราฐานของ SQL3

This section enumerates the features that make up SQL3/Foundation (3) and SQL3/Bindings (4) , over and above those of SQL-92 and SQL-92/PSM. There is no significance to the order in which the features appear. A summary of these features appears first , followed by a more detailed definition of the features.

SQL3/Foundation and SQL3/Bindings Features

1. Timestamps in information schema for configuration management
2. Extensions to Embedded SQL exception declarations
3. BOOLEAN data type
4. BLOB , CLOB , and NCLOB data types
5. ROW data type
6. Collection data types
7. CASCADE option for DROP COLLATION Large objects
8. Abstract data type (ADT)
9. Abstract data type (ADT) subtypes
10. Distinct data type
11. User-defined operators
12. Updateable joined tables
13. WITH in <query expression>
14. Recursive query
15. SIMILAR predicate
16. DISTINCT predicate
17. Boolean predicate
18. Optional interval qualifier
19. LIKE clause in table definition
20. Subtables
21. Referential action RESTRICT
22. Comparable data types for referential constraints

SQL3/Foundation and SQL3/Bindings Features

25. SENSITIVE cursors
26. START TRANSACTION statement
27. LOCAL option for SET TRANSACTION statement
28. Chained transactions
29. Save points
30. SELECT privilege with column granularity
31. Static and Dynamic execution rights
32. Functional Dependencies
33. OVERLAY function for characters and

23. Triggers

24. WITH HOLD cursors PAGE 6

1. Timestamps in information schema for configuration management

This appears to only have been added to clause 19.3.36 , "ROUTINES base table" , of SQL3/Foundation. This definition uses clause 19.2.48 , " TIME_STAMP domain" , of SQL3/Foundation.

2. Extensions to Embedded SQL exception declarations.

SQLSTATE , CONSTRAINT , SQLEXCEPTION , SQLWARNING have been added to clause 13.2 , "<embedded exception declaration>" , of SQL3/Bindings.

EXEC SQL WHENEVER SQLSTATE (02,001) CONTINUE;

3. BOOLEAN data type.

Boolean literals TRUE , FALSE , and UNKNOWN.CAST extension. Boolean value expressions (IMPLIES , IFF , and IMPLIDBY). Aggregate operators (EVERU.AMY , SOME).

4. BLOB , CLOB , and NCLOB data types.

Large object locators (holdable and not holdable). May be used in <like predicate> , <position expression> , <comparison predicate> (With an <equals operator> or <not equals operator>) , or <quantified comparison predicate> (With the <equals operator> or <not equals operator>). Clause 13.12 , "<hold locator statement>" , and clause 13.13 , "<free locator statement>".

5. ROW data type.

ROW reference. Field reference.

Allow ROW to be specified in a <row value constructor> in clause 7.1 , "<row value constructor>" , of SQL3/Foundation.

```
SELECT      *
FROM        employees
WHERE       ROW (lname , fname) = ROW ('Cannan' , 'Steve');
```

6. Collection data types.

SET , LIST , and MULTISET. Comparison of collections. TABLE (x) as a synonym for MULTISET (ROW (X)). <collection derived table> in <table reference>. <collection reference>. Set operators (S_UNION , S_INTERSECT , and S_EXCEPT). Collection aggregate operators (S_UNION , S-INTERSECT , and S_EXCEPT). CARDINALITY expression. ELEMENT value function. LIST value functions (SUBLIST and CONCATENATE). EMPTY as cast operand. EMPTY as a <row value constructor> element and in clause 11.9 , "<default clause>" , of SQL3?Foundation. SET , MULTISET , and LIST constructors.

Example:

```
        ALTER TABLE employees ADD COLUMN
                Hobbies SET (CHARACTER VARYING (20));
        ALTER TABLE employees ADD COLUMN
                Dependents TABLE (fname CHARACTER VARYING (20) , Lname CHARACTER
VARYING (20) , bodate DATE);
SELECT          COUNT (*)
FORM            TABLE (SELECT S_UNION (hobbies)
FROM            employees
) hobbies (hobby);
SELECT          e.lname , e.fname
FROM            employees e
WHERE           CARDINALITY (hobbies) >= 3;
```

SET , MULTISET , and LIST constructors

```
        UPDATE          employees
        SET             hobbies = SET ('Fly fishing' , 'Origami');
```

A<query specification> may contain a <collection expression>.

```
SELECT          *
FROM            hobbies IN (SELECT hobbies
                        FROM            employees
                        WHERE           lname = 'Cannan')
                dependents IN (SELECT dependents
                        FROM            employees
                        WHERE           lname = 'Kulkarni')
```

ITEM may be specified in a <query specification> or <select statement : single row>.

```
        BEGIN
        DECLARE     result_set MULTISET (CHARACTER VARYING (30));
        SET         result_set = (SELECT ITEM lname
                                FROM        employees
                                WHERE       hair_color = 'Black');
        END;
```

THE in <subquery> (needs to be expanded).

<collection value expression> may be specified in an IN predicate , an <exist predicate> , a <unique predicate> , or a <match predicate>.

```
SELECT          *
FROM            employees
WHERE           hobbies IN (SELECT hobbies
                        FROM        employees
                        WHERE       lname = 'Mattos'
                        S_INTERSECT
                        SELECT              hobbies
                        FROM        employees
                        WHERE       lname = 'Richie'
                        );
```

7. CASECADE option for DROP COLLATION.

<drop behavior> may be specified in clause 11.37 , "<drop collation statement>" , of SQL3/Foundation.

```
DROP COLLATION latinl_insensitive CASECADE;
```

8. Abstract data type (ADT)

Clause 11.45, "<abstract data type bodyt>", of SQL3/Foundation, without the specification of <subtypeclause>. CREATE and DROP ,ordering clause, operator name list, CAST,encapsulation levels, constructors, attribute reference. ACT locators (holdable and not holdable). Component reference Clause 11.47,"<drop data type statement>",of SQL3/Foundation (also appears in DISTINCT datatype).

9. Abstract data type ADT) subtypes.

UNDER privilege. TREAT subtype as supertype.

<genralized expression> in clause 10.4,">routine invocation>", of SQL3/Foundation

TYPE predicate as specified in clause 8016,"<type predicate>",of SQL3/Foundation

RESULT may be specified for a <parameter declaration> in clause 12.5 "<SQL-invokedroutine>", of SQL3/Foundation

10. Distinct data type

Specified in clause 11.46,"<distinct type definition>",of SQL3/Foundation.

```
CREATE DISTINCT TYPE weight_lbs AS DECIMAL (3);
ALTER TABLE employees ADD COLUMN weight weight_lbs ;
```

Clause 11.47 ,"<drop data type statement>", of SQL3/Foundation (also appears in ADT data type).

11. User-defined operators.

Clause 11.48,"operators definition>", of SQL3Foundation. <add operators definition> in clause 11.2," <alter schema statement>", of SQL3/Foundation.

12. Updatable joined tables.

SR 12)d and 12)e of 7.9 ,"<query specification>", of SQL-92 do not appear to have corresponding rules in SQL3/Foundation.

d) if the <table expression> immediately contained in QS immediately contains a <where clause> WC, then no leaf generally underlying table of QS shall be a generally underlying of ant <query expression> contained in WC,

e) The <table expression> immediately contained in QS does not include a <group by clause> or <having clause>. Immediately contained in QS does not include a <group by clause> or a <having clause>.

13. WITH in <query expression>

<with clause> when RECURSIVE has not been specified.

```
SELECT *
FROM   (WITH top_east_coast_sales AS
              (SELECT *
              FROM employees_east
              WHERE job = 'SALE'
              AND  compensation>250000),
        Top_west_coast_sales AS
              (SELECT *
              FROM employees_west
              WHERE job = 'SALE'
              AND  compensation > 250000
         Top_east_coast_sales UNION top_west_coast_sales
       )AS top_sales ;
```

14. Recursive query.

<with clause> or <view definition> when RECURSIVE has been specified.

```
WITH RECURSIVE Reachable_From (Source,Destin) AS
        (SELECT         Source,Destin
                        FROM   Flights
                        WHERE  Source = 'Paris'
                        UNION
                                SELECT          in.Source, out.Destin
                                FROM            Reachable_From in, Flights out
                                WHERE           in.Destin = out.Source
```

)

SELECT * FROM Reachable_From

Functional Dependencies

Keys in Descriptor Area.

33. OVERLAY function for characters and large objects.

<character overlay function>and<blob overlay function> in clause 6.10,"<string value function>", of SQL3/Foundation.

```
SELECT          '+1.'
                OVERLAY (phone number
                        PLACING'.'
                        FROM 4
                        FOR 1 )
                PLACING '.'
                FROM 8
                FOR 1
        FROM  employees ;
```

34. SQL - invoked routines.

SQL-paths. CURRENT_PATH value specification. CURRENT_PATH in 11.9 <default clause>", of SQL3/Foundation. Procedures and function. SQL routine and external routine. Polymorphism Subject-routine determination. Return statement. EXECUTE privilege. Clause 11.49 <drop routine statement>.

35. Roles.

Clause11.51,"<ROLE DEFINITION>", of SQL3/Foundation,11.52,"<grant role statement>", of SQL3/Foundation, 11.53,"<revoke role statement>",of SQL3/Foundation. Grant and revoke of WITH ADMIN OPTION. Clause 17.3. "<set role statement>", of SQL3/Foundation

```
CREATE ROLE payroll ;
GRANT PAYROLL To vickers WITH ADMIN OPTION;
GRANT UPDATE (salary) ON TABLE employees To payroll ;
.....
SET ROLE payroll ;
```

36. Upper SQL Flagging.

37. Bracketed comments.

<bracketed comment> in clause 5.2,"<token>and<separator>", of SQL3/Foundation.

```
/* This is a comment */
```


38. User-defined aggregate operators.

<routine name> may be specified for <set function type> in clause 6.8,"<set function specification>", of SQL3/Foundation

39. Quantified predicate.

Existential and universal quantification may be specified in clause 8.13,"<quantified predicate>", of SQL3/Foundation.

```
SELECT    *
FROM      employee
WHERE     FOR SOME hobby IN e.hobbies (hobby LIKE '0%') ;
```

40. ALL SCHEMA PRIVILEGES.

ALL SCHEMA PRIVILEGES may be specified for <privileges> in clause 10.5." <privileges>", in SQL3/Foundation.

41. Value expression in order by clause.

Removal of the SQL-92 restriction that <sort key> must be a <column name> or an <unsigned integer>.

```
SELECT                *.
FROM                  employees
ORDER BY CARDINALITY      (dependents) ;
```

42. Table name not required in <delete statement :positioned>or<update statement :positioned>.

Remove the SQL-92 requirement that "FROM<table reference>" be specified in clause 13.6,"<delete statement: positioned>", of SQL3/Foundation and clause 13.9,"<update statement :positioned>",of SQL3/Foundation.

```
DECLARE curs 1 CURSOR FOR SELECT * FROM employees ;
OPEN curs 1 ;
FETCH curs1 INTO ……;
UPDATE
SET     status = "Sabbatical"
WHERE CURRENT OF curs1;
FETCH curs1 INTO ……;
DELETE WHERE CURRENT OF curs 1 ;
CLOSE curs1 ;
```

43. INSERT in a cursor.

A cursor name may be specified instead of a table name in clause 13.8."<insert statement>", of SQL3/Foundation.

```
DECLARE curs1 CURSOR FOR SELECT * FROM employees ;
OPEN curs1
```

FETCH curs 1 INTO .. ;

INSERT INTO CURSOR curs1 VALUES ('Darwen','Hugh',....);

CLOSE curs1;

44.ROW may be specified for an <update target> in clause 13.9,"<update statement :positioned>",of SQL3/Foundation.

DECLARE curs1 CURSOR FOR SELECT * FROM employees ;

OPEN curs1;

FETCH curs1 INTO ...;

UPDATE employees

SET ROW = (...)

WHERE CURRENT OF curs1:

CLOSE curs1 ;

ภาคผนวก ค

Source Code SQL ที่ใช้สร้างฐานข้อมูล

```
CREATE ROW TYPE componentType
(       projectID       integer,
        subProcess      integer,
        ID              integer
);

CREATE ROW TYPE processType
(       name            char(15),
        Px              Integer,
        Py              Integer,
        description     lvarchar
) UNDER componentType;

CREATE ROW TYPE externalType
(       name            char(15),
        Px              Integer,
        Py              Integer
) UNDER componentType;

CREATE ROW TYPE datastoreType
(       name            char(15),
        Px              Integer,
        Py              Integer
) UNDER componentType;
```

```
CREATE ROW TYPE  dataflowType
(       SourceID        integer,
        SourceType      char(2),
        DestinationID   integer,
        DestinationType char(2),
        formatData      char(20),
        Sx              Integer,
        Sy              Integer,
        Dx              Integer,
        Dy              Integer
) UNDER componentType;

CREATE ROW TYPE  GateType
(       ModelID         Integer,
        ModelType       char(2),
        ModelName       char(15),
        Px              Integer,
        Py              Integer
) UNDER componentType;

CREATE ROW TYPE ERType
(       ProjectID       integer,
        ID              integer
);

CREATE ROW TYPE entityType
(       Name    char(20),
        Px      Integer,
        Py      Integer
) UNDER ERType;
```

```
CREATE ROW TYPE RelationType
(       SourceID        integer,
        DestinationID   integer,
        Type            char(5),
        Relationship    char(30),
        Sx              Integer,
        Sy              Integer,
        Dx              Integer,
        Dy              Integer
) UNDER ERType;

CREATE TABLE projectTable
(       projectID integer,
        projectName char(20),
        PRIMARY KEY ( projectID),
        CHECK ( projectName IS NOT NULL ),
        CHECK ( projectID >=1 )
);

CREATE TABLE  processTable  OF TYPE  processType
(       PRIMARY KEY ( projectID , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( name IS NOT NULL ),
        CHECK ( description IS NOT NULL),
        CHECK ( Px IS NOT NULL ),
        CHECK ( Py IS NOT NULL ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL ))
 );
```

```
CREATE TABLE externalTable OF TYPE externalType
(       PRIMARY KEY ( projectID , subProcess , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( name IS NOT NULL ),
        CHECK ( Px IS NOT NULL ),
        CHECK ( Py IS NOT NULL ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL ))
);


CREATE TABLE datastoreTable OF TYPE datastoreType
(       PRIMARY KEY ( projectID , subProcess , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( name IS NOT NULL ),
        CHECK ( Px IS NOT NULL ),
        CHECK ( Py IS NOT NULL ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL ))
);


CREATE TABLE dataflowTable OF TYPE dataflowType
(       PRIMARY KEY ( projectID , subProcess , ID ),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >= 0 ) and ( subProcess IS NOT NULL )),
        CHECK (formatData IS NOT NULL),
        CHECK (( SourceID IS NOT NULL) and ( SourceID >=1)),
        CHECK (( DestinationID IS NOT NULL)  and( DestinationID >=1)),
        CHECK (( SourceType IS NOT NULL)  and ( SourceType in ('P','E','D','GP','GD','GE'))),
```

```
        CHECK (( DestinationType IS NOT NULL)  and ( DestinationType in
('P','E','D','GP','GD','GE'))),
        CHECK (Sx  IS NOT NULL),
        CHECK (Sy  IS NOT NULL),
        CHECK (Dx  IS NOT NULL),
        CHECK (Dy IS NOT NULL)
);

CREATE TABLE GateTable OF TYPE GateType
(       PRIMARY KEY ( projectID,subprocess,ID),
        FOREIGN KEY ( projectID ) REFERENCES projectTable ( projectID ),
        CHECK ( ID >= 1 ),
        CHECK (( subProcess >0 ) and ( subProcess IS NOT NULL )),
        CHECK ( ModelID IS NOT NULL ),
        CHECK (( ModelType IS NOT NULL ) and (ModelType IN ('P','E','D','GP','GE','GP'))),
        CHECK ( ModelName IS NOT NULL ),
        CHECK ( Px  IS NOT NULL ),
        CHECK ( Py  IS NOT NULL )
);

CREATE TABLE entityTable OF TYPE entityType
(       PRIMARY KEY (projectID , ID),
        FOREIGN KEY ( projectID) REFERENCES projectTable (projectID) ,
        CHECK (name IS NOT NULL),
        CHECK (Px IS NOT NULL),
        CHECK (Py IS NOT NULL),
        CHECK (ID >=1)
);

CREATE TABLE RelationTable OF TYPE relationType
```

```
(       PRIMARY KEY (projectID , ID),
        FOREIGN KEY ( projectID) REFERENCES projectTable (projectID) ,
        CHECK (Relationship IS NOT NULL),
        CHECK ((SourceID IS NOT NULL) and (SourceID >=1)),
        CHECK ((DestinationID IS NOT NULL) and (DestinationID >=1)),
        CHECK ((Type IS NOT NULL) and(Type  in ('OTO','OTM','MTM'))),
        CHECK (Sx IS NOT NULL),
        CHECK (Sy IS NOT NULL),
        CHECK (Dx IS NOT NULL),
        CHECK (Dy IS NOT NULL),
        CHECK (ID >=1)
);
```

Source Code ของ Rountine ที่สร้างขึ้น

```
CREATE FUNCTION  genProjectID()
        RETURNING integer;

        DEFINE rowNum integer;
        DEFINE resultNum integer;
        DEFINE i integer ;
        LET i = 1;

        SELECT count(*)  INTO rowNum FROM  projectTable ;

        IF rowNum > 0 THEN
                WHILE ( i <= rowNum )
                        SELECT count(*) INTO resultNum
                        FROM projectTable
                        WHERE projectID=i;

                        IF resultNum = 0
                        THEN    EXIT WHILE;
                        END IF;

                        LET i=i+1;
                END WHILE;
        ELIF rowNum = 0 THEN LET i=1;
        END IF;

        RETURN i;
END FUNCTION
```

```
CREATE FUNCTION CheckDataER (pj integer)
 RETURNING     Char(1);

 DEFINE Answer1              Integer;
 DEFINE Answer2              Char(1);
 DEFINE SubProc              Integer;
 DEFINE LastCheck            Char(3);
 DEFINE Final                Char(1);
 DEFINE LastAnswer           Char(3);

 LET LastAnswer='Y' ;
    FOREACH Sub_Cursor FOR
        SELECT Distinct subprocess      INTO SubProc
        FROM ProcessTable
        WHERE ((projectID=pj) AND (subprocess<>0))

        Execute Function CheckNoHaveSub(pj,SubProc) INTO Answer1 , Answer2;

        LET    Final='Y';

        If  Answer2 ='T'
        Then Execute Function TestCheck3(pj,SubProc) Into Final ;
                If Final = 'N'
                Then LET LastAnswer='N';
                End If;
        End If;

        If Final = 'N'
        Then LET LastAnswer='N';
        End If;
    END FOREACH;
RETURN  LastAnswer ;
END FUNCTION;
```

```
CREATE FUNCTION CheckNoHaveSub (project integer, SubProc Integer)
 RETURNING  Integer , Char(1);

 DEFINE     Ans     Char(1);
 DEFINE     Number Integer;

 LET Ans='T';

FOREACH Gate_cursor FOR
        SELECT ID INTO Number
        FROM ProcessTable
        WHERE (projectID=project) AND (subprocess=SubProc)

        IF Number IN (Select subprocess
                From ProcessTable
                Where (projectID=project) AND (subprocess<>0))
        Then LET Ans = 'F';
        END IF;

END FOREACH;

        RETURN SubProc , Ans ;

END FUNCTION;
```

```
CREATE FUNCTION TestCheck3(ProjID Integer , SubProc Integer)
 RETURNING  Char(1);

DEFINE TempID        Integer ;
DEFINE TempName      Char(15) ;
DEFINELast           Char(1) ;

        Let      Last='N';
FOREACH Cursor1 FOR
        Select Id , Name
        INTO TempID,TempName
        From DataStoreTable
        Where (ProjectID=ProjID) And (SubProcess=SubProc)

        Let      Last='N';

        IF TempName In (        Select Name
                        From EntityTable
                        Where ProjectID=ProjID )
        THEN Let Last='Y';
        End If;

END FOREACH;
RETURN Last ;
END FUNCTION;
```

```
CREATE FUNCTION CheckERdiagram(pj integer)
  RETURNING char(1);

DEFINE EID          Integer;
DEFINE Tempname     Char(20);
DEFINE Answer       Char(1);

        LET Answer='Y';
  FOREACH ERcursor FOR
        SELECT ID, Name INTO EID,TempName
        FROM EntityTable
        WHERE ProjectID=pj

        IF TempName IN (    SELECT Name
                            FROM EntityTable
                            WHERE ( (ProjectID=pj) AND (ID<>EID) ) )
        THEN LET Answer='N';
        END IF;
  END FOREACH ;
        RETURN Answer ;
END FUNCTION;
```

```
CREATE FUNCTION NewFlow10(project integer,sub integer )
 RETURNING   Char(1) ;

 DEFINE          MoID          Integer;
 DEFINE          MoType        Char(2);
 DEFINE          NewMcType     Char(2);
 DEFINE          GID           Integer;
 DEFINE          Answer        Char(1);
 DEFINE          LastAnswer    Char(1);

        LET LastAnswer='Y';
        FOREACH Gate FOR
                SELECT id , ModelId , ModelType  INTO GID , MoID , MoType
                FROM GateTable
                WHERE (projectid=project) AND (Subprocess=sub)

                If MoType='P' Then LET   NewMoType='GP' ; End If;
                If MoType='E' Then LET   NewMoType='GE' ; End If;
                If MoType='D' Then LET   NewMoType='GD' ; End If;
                If MoType='GP' Then LET  NewMoType='GP' ; End If;
                If MoType='GE' Then LET  NewMoType='GE' ; End If;
                If MoType='GD' Then LET NewMoType='GD' ; End If;

                EXECUTE FUNCTION NewFlow5(project,sub,GID,MoID,MoType,NewMoType)
        INTO Answer ;

                If Answer='N' Then LET LastAnswer='N';  End If;

        END FOREACH;
        RETURN LastAnswer ;
END FUNCTION;
```

บรรณานุกรม

[1] Informix Thailand , "INFORMIX 9.14 MANUAL" , Informix , 1997

[2] John McNally,Glenn Miller,Jim Prajesh,Jose Fortuny, et al. , "INFORMIX UNLEASHED" , SAMA PUBLISHING , 1997

[3] Michael Blaha and William Premerlani , "Object-Oriented Modeling and design for Database Applications" , PRENTICE-HALL INTERNATIONAL INC , 1997

[4] James A. Senn . "Analysis and Design of Information Systems" , McGRAW-HILL INTERNATIONAL EDITIONS , 1989

[5] C.J. Date , "An Introduction to Database Systems Volume I " , ADDISON-WESLEY PUBLISHING COMPANY , 1986

[6] G.M. NYSSEN and E.D. FALKENBERG , "Introduction to IBM SQL" , 1984

[7] Jeffrey D. Ullman and Jennifer Widom , "A First Course in Database Systems" , PRENTICE-HALL International Inc. , 1997

[8] ABRAHM SILBERSCHATZ,HENRY F. KORTH, S. SUDARSHAN , " DATABASE SYSTEM CONCEPTS" , Mc GRAW-HILL INTERNATIONAL EDITION

[9] สุทธิศุกดิ์ พงศ์ธนาพานิช , "Visual Basic 5.0" ,ซีเอ็ดยูเคชั่น , 1998

[10] สัจจะ จรัสรุ่งรวีวร , "เรียนรู้และใช้งานได้จริง Visual Basic 5.0" , บริษัท ดวงกมล สมัย จำกัด , 1989