การประมวลผลคำถามแบบกระจาย
บนระบบจัดการฐานข้อมูล
DISTRIBUTED QUERY PROCESSING
ON DATABASE MANAGEMENT SYSTEMS

โดย
นายอนันต์ สุขสงเคราะห์
นายเอก โชติชวาลวงศ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิศวกรรมคอมพิวเตอร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2536

ปริญญานิพนธ์ปีการศึกษา 2536

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

เรื่อง การประมวลผลคำถามแบบกระจายบนระบบจัดการฐานข้อมูล

ผู้จัดทำ

1.นายอนันต์ สุขสงเคราะห์ รหัส 33100480

2.นายเอก โชติชวาลวงศ์ รหัส 33100530

..... อาจารย์ที่ปรึกษา

(ผศ. ดร. ศุภมิตร จิตตะยโศธร)

การประมวลผลคำถามแบบกระจาย
บนระบบจัดการฐานข้อมูล
DISTRIBUTED QUERY PROCESSING
ON DATABASE MANAGEMENT SYSTEMS

โดย นายอนันต์ สุขสงเคราะห์ รหัส 33100480
นายเอก โชติชวาลวงศ์ รหัส 33100530

อาจารย์ที่ปรึกษา ผศ.ดร.ศุภมิตร จิตตะยโศธร

บทคัดย่อ

ในช่วงทศวรรษที่ผ่านมานี้ องค์กรต่างๆ มีอัตราการเจริญเติบโตอยู่ในระดับที่สูงมาก รวมทั้งประสิทธิภาพของระบบคอมพิวเตอร์ทั้งในด้านฮาร์ดแวร์และซอฟท์แวร์ได้มีการพัฒนาเพิ่มขึ้นอย่างรวดเร็ว ด้วยเหตุผลข้างต้นนี้จึงได้มีความพยายามในการเชื่อมต่อระบบฐานข้อมูลของแต่ละองค์กรหรือหลายๆ องค์กรเข้าด้วยกัน และเทคโนโลยีที่นำมาใช้ในกระบวนการดังกล่าว ได้แก่ ระบบจัดการฐานข้อมูลแบบกระจาย

ปริญญานิพนธ์นี้ขอเสนอ แนวทางในการพัฒนาระบบประมวลผลคำถามแบบกระจาย ซึ่งถือเป็นส่วนหนึ่งของระบบจัดการฐานข้อมูลแบบกระจาย โดยระบบประมวลผลคำถามระบบนี้ใช้ภาษาเอสคิวแอล (SQL) ในการติดต่อกับผู้ใช้ และระบบนี้ได้รับการพัฒนาให้ทำงานบนระบบจัดการฐานข้อมูลออราเคิล ซึ่งทำงานบนระบบคอมพิวเตอร์ที่ใช้ UNIX เป็นระบบปฏิบัติการบนโครงข่ายทีซีพี/ไอพี (TCP/IP) โดยไม่จำเป็นต้องแก้ไขระบบฐานข้อมูลเดิมที่มีอยู่แล้วแต่ประการใด

ABSTRACT

In the past decade, the growth rate of many organizations is very high. Furthermore, the efficiency of computer systems -- in both hardware and software aspects -- is growing very rapidly. For these resons, the necessity to interconnect database system of organization(s) arises and a suggested technology for such a process is distributed database management system.

This paper presents guidelines to develope distributed query processing system which is one of the distributed database management system technology. The system uses SQL language to interface with the user(s). It is built on top of Oracle database management system that runs on UNIX machines in a TCP/IP network without any modification to the existing DBMS.

สารบัญ

สารบัญรูปภาพ

รูปที่ 5.4 แสดงต้นไม้พีชคณิตที่ได้จากตัวอย่างที่ 5.5

บทที่ 1
บทนำ

ในปัจจุบัน ข่าวสารและข้อมูลนับเป็นสิ่งสำคัญสำหรับทุกๆ องค์กร ทำให้ระบบจัดการฐานข้อมูลเริ่มมีบทบาทสำคัญมากขึ้น ในการช่วยให้การค้นหาและจัดเก็บข้อมูล เป็นไปอย่างถูกต้องและรวดเร็ว รวมไปถึงการบำรุงรักษาข้อมูลให้อยู่ในสภาพที่พร้อมใช้งานได้ตลอดเวลา

เดิมทีนั้น ระบบจัดการฐานข้อมูลที่ใช้ส่วนใหญ่ จะเป็นระบบจัดการฐานข้อมูลแบบรวม (Centralized database management system) ซึ่งการขยายระบบจะทำได้ค่อนข้างยาก และการนำระบบเดี่ยวๆ 2 ระบบมาเชื่อมต่อกันก็ทำได้ยากเช่นกัน ดังนั้น จึงมีการนำระบบจัดการฐานข้อมูลแบบกระจาย (Distributed database management system) มาใช้ ซึ่งในปัจจุบัน มีบริษัทต่างๆ หลายบริษัทได้ผลิตโปรแกรม ที่มีความสามารถในการเชื่อมต่อระบบฐานข้อมูลแบบรวมให้เป็นระบบฐานข้อมูลแบบกระจาย เช่น INGRES/NET [1,2] สามารถเชื่อมต่อระบบฐานข้อมูลที่ใช้ระบบจัดการฐานข้อมูลอินเกรส (INGRES) เข้าด้วยกัน เป็นระบบฐานข้อมูลแบบกระจาย และ SQL*NET [3] เป็นโปรแกรมที่เชื่อมต่อระบบฐานข้อมูลที่ใช้ระบบจัดการฐานข้อมูลออราเคิล ให้เป็นระบบฐานข้อมูลแบบกระจาย จะเห็นได้ว่า โปรแกรมต่างๆ เหล่านี้ จะถูกผลิตมาเพื่อสนับสนุนระบบจัดการฐานข้อมูลของบริษัทนั้นๆ โดยเฉพาะ

ในบทความนี้ได้นำเสนอ การเชื่อมต่อกันของระบบจัดการฐานข้อมูล 2 วิธี คือ

1. การส่งคำถามไปประมวลผลที่ระบบจัดการฐานข้อมูลแบบรวม โดยผ่านโครงข่ายคอมพิวเตอร์ที่ใช้ชุดโปรโตคอลทีซีพี/ไอพี แล้วส่งผลลัพท์กลับมาให้ระบบงานที่ขอใช้บริการของระบบฐานข้อมูลระบบนั้น ข้อดีของวิธีนี้คือ สามารถใช้ภาษา SQL ได้เต็มรูปแบบ แต่มีข้อเสียตรงที่ไม่สามารถอ้างถึงข้อมูลที่อยู่ต่างสถานที่ (site) ในคำถามเดียวกันได้

2. การเชื่อมต่อระบบจัดการฐานข้อมูลแบบรวม ให้เป็นระบบจัดการฐานข้อมูลแบบกระจาย โดยเป็นการพัฒนาต่อจากระบบประมวลผลคำถามแบบกระจาย DDQ/1 ซึ่งมีความสามารถในการเชื่อมต่อระบบจัดการฐานข้อมูลแบบรวม (ในที่นี้ใช้ระบบจัดการฐานข้อมูลออราเคิล) ที่สามารถทำงานเป็นระบบจัดการฐานข้อมูลได้โดยอิสระ ให้สามารถทำงานเป็นระบบจัดการฐานข้อมูลแบบกระจาย ในโครงข่ายคอมพิวเตอร์ที่ใช้โปรโตคอลทีซีพี/ไอพี (TCP/IP) ภายใต้ระบบปฏิบัติการยูนิกซ์ (UNIX) ระบบ DDQ/1 มีความสามารถในการวิเคราะห์คำถามและข้อมูล เพื่อให้ใช้เวลาในการประมวลผลน้อยที่สุด[4,5] โดยการตรวจสอบคำถามด้วยกฎบังคับความถูกต้อง, การปรับปรุงคำถาม และการกำหนดสถานที่ในการประมวลผลคำถาม แต่เนื่องจากระบบ DDQ/1 ใช้ภาษาพีชคณิตสัมพันธ์ในการติดต่อกับผู้ใช้ ซึ่งไม่เป็นที่คุ้นเคยกับผู้ใช้ จึงเกิดแนวความคิดในการพัฒนาระบบ DDQ/1 ให้สามารถใช้ภาษา SQL ในการติดต่อกับผู้ใช้ได้ ซึ่งจะขอเรียกระบบประมวลผลคำถามที่พัฒนาใหม่นี้ว่า "DDQ/II" ระบบประมวลผลคำถามแบบนี้มีข้อดีตรงที่สามารถอ้างถึงข้อมูลต่างสถานที่ในคำถามเดียวกันได้ แต่มีข้อเสียตรงที่ไม่สามารถใช้ภาษา SQL ได้เต็มรูปแบบ

บทที่ 2
หลักการและทฤษฎีที่เกี่ยวข้อง

2.1 หลักการของระบบฐานข้อมูลแบบกระจาย

2.1.1 ความหมายของฐานข้อมูลแบบกระจาย

เป็นที่ทราบกันดีแล้วว่า ในช่วงทศวรรษ 1970 คอมพิวเตอร์ได้ถูกนำมาใช้ประโยชน์ในด้านต่างๆอย่างมากมาย ซึ่งรวมทั้งการนำมาใช้งานเกี่ยวกับระบบฐานข้อมูลด้วย และจากการพัฒนาการของระบบฐานข้อมูล ที่เป็นไปอย่างรวดเร็ว ทำให้ในปัจจุบันมีโปรแกรมใช้งานขนาดต่างๆมากมาย ในขณะเดียวกันโครงข่ายคอมพิวเตอร์ก็ได้รับการพัฒนาอย่างต่อเนื่อง มีการเชื่อมต่อของคอมพิวเตอร์ต่างเครื่องกัน ทำให้สามารถจะแลกเปลี่ยนข้อมูลและทรัพยากรซึ่งกันและกันระหว่างคอมพิวเตอร์ต่างเครื่องกันได้

จากการพัฒนาของวิทยาการทั้งสองสาขา และการผสมผสานกันทำให้เกิดเป็นความรู้อีกสาขาหนึ่งซึ่งเรียกว่า "ระบบฐานข้อมูลแบบกระจาย" (Distributed Database Systems) หากจะกล่าวโดยสรุปแล้ว ระบบฐานข้อมูลแบบกระจายก็คือ "การเก็บรวบรวมข้อมูลที่มีความสัมพันธ์ ในทางตรรก เข้าไว้เป็นระบบเดียวกัน แต่จะกระจายข้อมูลไปเก็บตามสถานที่ (site) ต่างๆ โดยผ่านระบบโครงข่ายคอมพิวเตอร์" [5]

จากคำจำกัดความข้างต้น สามารถสรุปเป็นคุณสมบัติของระบบฐานข้อมูลแบบกระจายได้ 3 ประการคือ

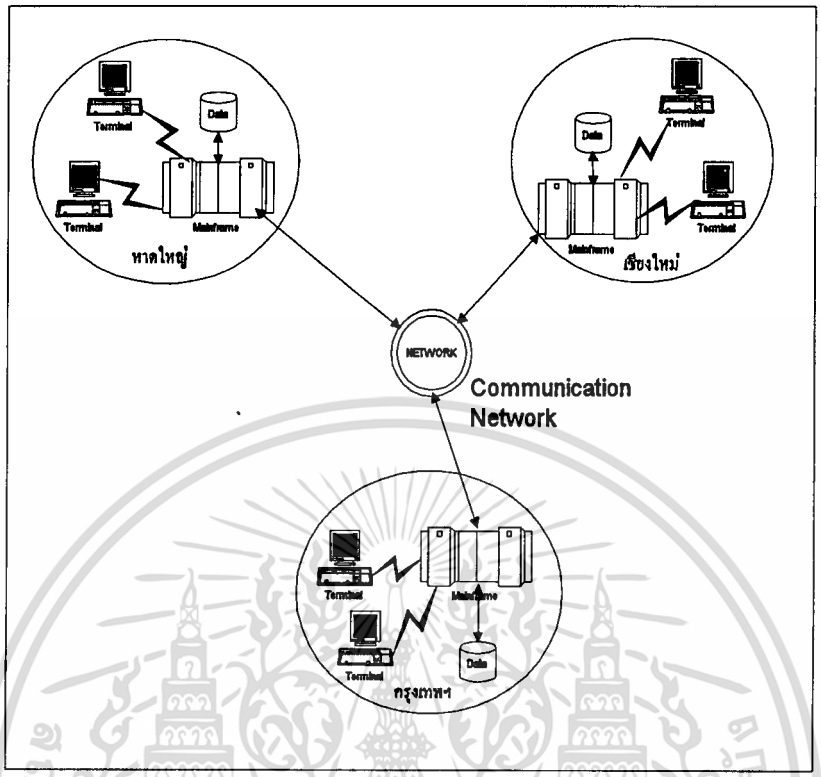
1. มีการกระจาย (Distribution) ของข้อมูล กล่าวคือ ข้อมูลจะไม่ถูกเก็บไว้ในสถานที่เดียวกัน (ตัวประมวลผลเดียวกัน) ดังนั้น เราสามารถที่จะแยกระบบฐานข้อมูลแบบรวมออกจากระบบฐานข้อมูลแบบกระจายได้ 2. มีความเกี่ยวพันทางตรรก (Logical Correlation) จากเหตุผลที่ข้อมูลในแต่ละสถานที่ ที่ประกอบกันเป็นระบบฐานข้อมูลแบบกระจายควรจะมีความสัมพันธ์กัน ดังนั้น เราสามารถที่จะชี้ให้เห็นความแตกต่างของระบบฐานข้อมูลแบบกระจาย กับกลุ่มของระบบฐานข้อมูลแบบรวมศูนย์ที่อยู่ในต่างสถานที่ได้

3. มีโปรแกรมประยุกต์ (Application Programs) โดยระบบฐานข้อมูลจะต้องมีทั้งโปรแกรมประยุกต์แบบรวม (Global Application Programs) และโปรแกรมประยุกต์แบบท้องถิ่น (Local application Programs)

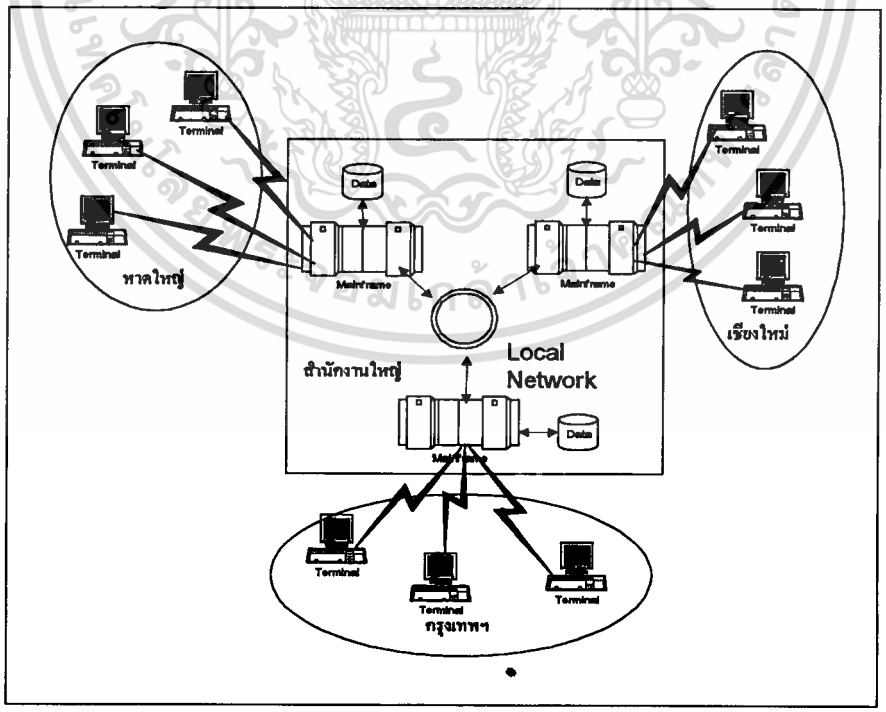
จะขอยกตัวอย่างเพื่อให้เห็นความแตกต่างชัดเจนมากยิ่งขึ้น ระหว่างระบบฐานข้อมูลแบบกระจายกับระบบฐานข้อมูลแบบรวม

ตัวอย่างที่ 2.1

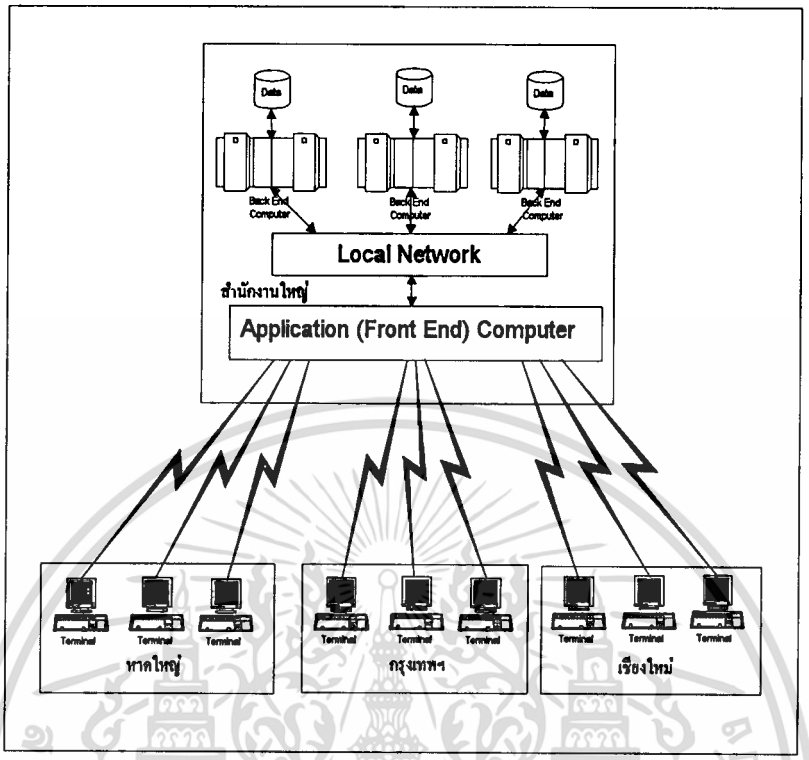
จากรูปที่ 2.1 แสดงการจำลอง ธนาคาร 3 สาขาที่อยู่ต่างสถานที่กัน แต่ละสาขาจะทำการควบคุมเทอร์มินัล และข้อมูลของตัวเอง คอมพิวเตอร์และ ข้อมูลในแต่ละสาขาประกอบกันเรียกว่า "หนึ่งสถานที่" (site) ของฐานข้อมูลแบบกระจาย คอมพิวเตอร์ในแต่ละสถานที่จะต่อถึงกันด้วยโครงข่ายสื่อสาร โดยปรกติแล้วโปรแกรมประยุกต์จะเรียกใช้ข้อมูลของสาขาที่โปรแกรมทำงานอยู่เท่านั้น ซึ่งเรียกโปรแกรมประยุกต์ชนิดนี้ว่า "โปรแกรมประยุกต์แบบท้องถิ่น" (Local Applications) ส่วนโปรแกรมที่มีการเรียกใช้ข้อมูลที่เก็บต่างสถานที่นั้นจะเรียกว่า "โปรแกรมประยุกต์แบบรวม" (Global Applications) หรือเรียกว่า "โปรแกรมประยุกต์แบบกระจาย" (Distributed Applications)



รูปที่ 2.1 แสดงระบบฐานข้อมูลที่กระจายตามลักษณะภูมิศาสตร์



รูปที่ 2.2 แสดงระบบฐานข้อมูลแบบกระจายที่ใช้โครงข่ายแบบท้องถิ่น



รูปที่ 2.3 แสดงระบบประมวลผลแบบหลายตัวประมวลผล

ตัวอย่างที่ 2.2

จากกรณีของธนาคารในตัวอย่างที่ 2.1 ถ้าโครงสร้างของระบบฐานข้อมูลถูกเปลี่ยนแปลงให้เป็นดังรูปที่ 2.2 คือ ย้ายคอมพิวเตอร์ทั้งหมดมาอยู่รวมกันที่ส่วนกลาง และต่อเทอร์มินัลผ่านสายโทรศัพท์ คอมพิวเตอร์แต่ละเครื่องและฐานข้อมูลของเครื่องนั้นๆ รวมกันเรียกว่าหนึ่งสถานที่ (site) เหมือนเดิม จะเห็นว่า แม้ลักษณะทางกายภาพจะถูกเปลี่ยนแปลงไป แต่สถาปัตยกรรมของระบบ และคุณสมบัติต่างๆ ทางตรรก ยังคงเหมือนเดิม การทำงานของโปรแกรมประยุกต์แบบท้องถิ่น ก็ยังคงประมวลผลที่เครื่องคอมพิวเตอร์สถานที่เดิม รวมทั้งเข้าถึงข้อมูลของสถานที่เดิมอีกด้วย ซึ่งสรุปว่า การแบ่งความเป็นท้องถิ่น ไม่ได้ขึ้นอยู่กับลักษณะทางกายภาพ แต่จะขึ้นอยู่กับสถานที่ (site) ประมวลผลและข้อมูลที่ใช้

ตัวอย่างที่ 2.3

จากรูปที่ 2.3 เป็นการแสดงโครงสร้างของธนาคาร ทั้ง 3 สาขา ซึ่งโครงสร้างจะเปลี่ยนไปจากตัวอย่างที่ 2.1 และ 2.2 คือ ข้อมูลของธนาคารทั้ง 3 จะเก็บไว้ที่เครื่องคอมพิวเตอร์ส่วนหลัง (backend) 3 เครื่อง ซึ่งจะทำหน้าที่จัดการฐานข้อมูล ส่วนโปรแกรมประยุกต์จะประมวลผลบนคอมพิวเตอร์เครื่องอื่น ซึ่งจะเรียกใช้ข้อมูลผ่านเครื่องคอมพิวเตอร์ส่วนหลัง จะเห็นว่าในตัวอย่างนี้ เป็นเพียงระบบฐานข้อมูลแบบรวม ที่ประมวลผลบนตัวประมวลผลหลายตัวเท่านั้น เนื่องจากโปรแกรมประยุกต์ใช้งาน ของธนาคารทั้ง 3 สาขาประมวลผลบน

คอมพิวเตอร์เครื่องเดียวกัน แม้ว่าจะแยกเก็บข้อมูลไว้ต่างที่กันก็ตาม ก็ไม่ได้จัดว่าเป็นระบบฐานข้อมูลแบบกระจาย

จากตัวอย่างที่กล่าวมาทั้งหมด คงจะเพียงพอสำหรับการทำความเข้าใจ ความหมาย และแยกความแตกต่างระหว่าง ระบบฐานข้อมูลแบบรวม และระบบฐานข้อมูลแบบกระจายได้

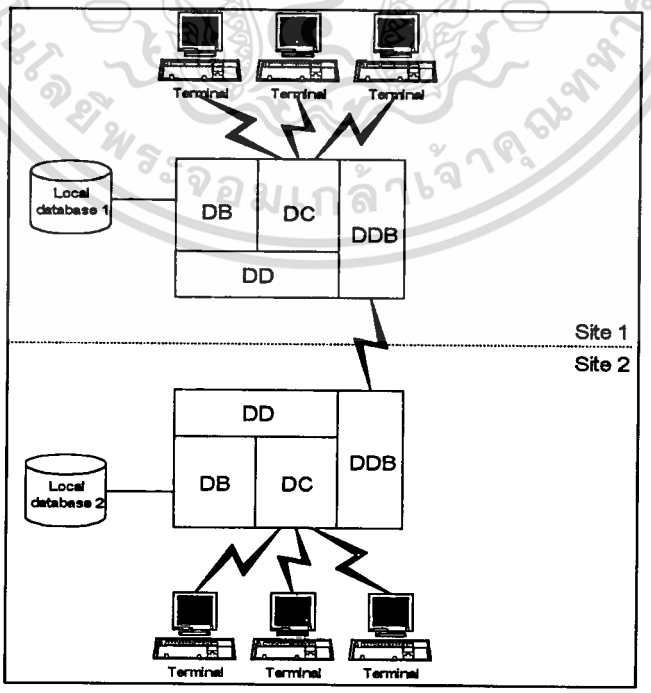
2.1.2 ระบบจัดการฐานข้อมูลแบบกระจาย

ระบบจัดการฐานข้อมูลแบบกระจายนั้น เป็นสิ่งจำเป็นสำหรับการสร้างระบบฐานข้อมูลแบบกระจาย และบำรุงรักษาข้อมูลในระบบฐานข้อมูลแบบกระจาย ซึ่งโดยปรกติแล้ว ผลิตภัณฑ์ระบบจัดการฐานข้อมูลแบบกระจายที่จำหน่ายในท้องตลาดนั้น จะมีความแตกต่างจากระบบจัดการฐานข้อมูลแบบกระจาย ที่กำลังพัฒนาในห้องทดลองบ้างพอสมควร แต่อย่างไรก็ตามโครงสร้างของระบบจัดการฐานข้อมูลแบบกระจาย จะมีลักษณะคล้ายกับโครงสร้างของระบบจัดการฐานข้อมูลแบบรวมอยู่บ้างดังรูปที่ 2.4 ซึ่งพอจะสรุปส่วนประกอบได้ดังนี้ คือ จะประกอบด้วยซอฟท์แวร์ 4 ส่วนด้วยกันได้แก่

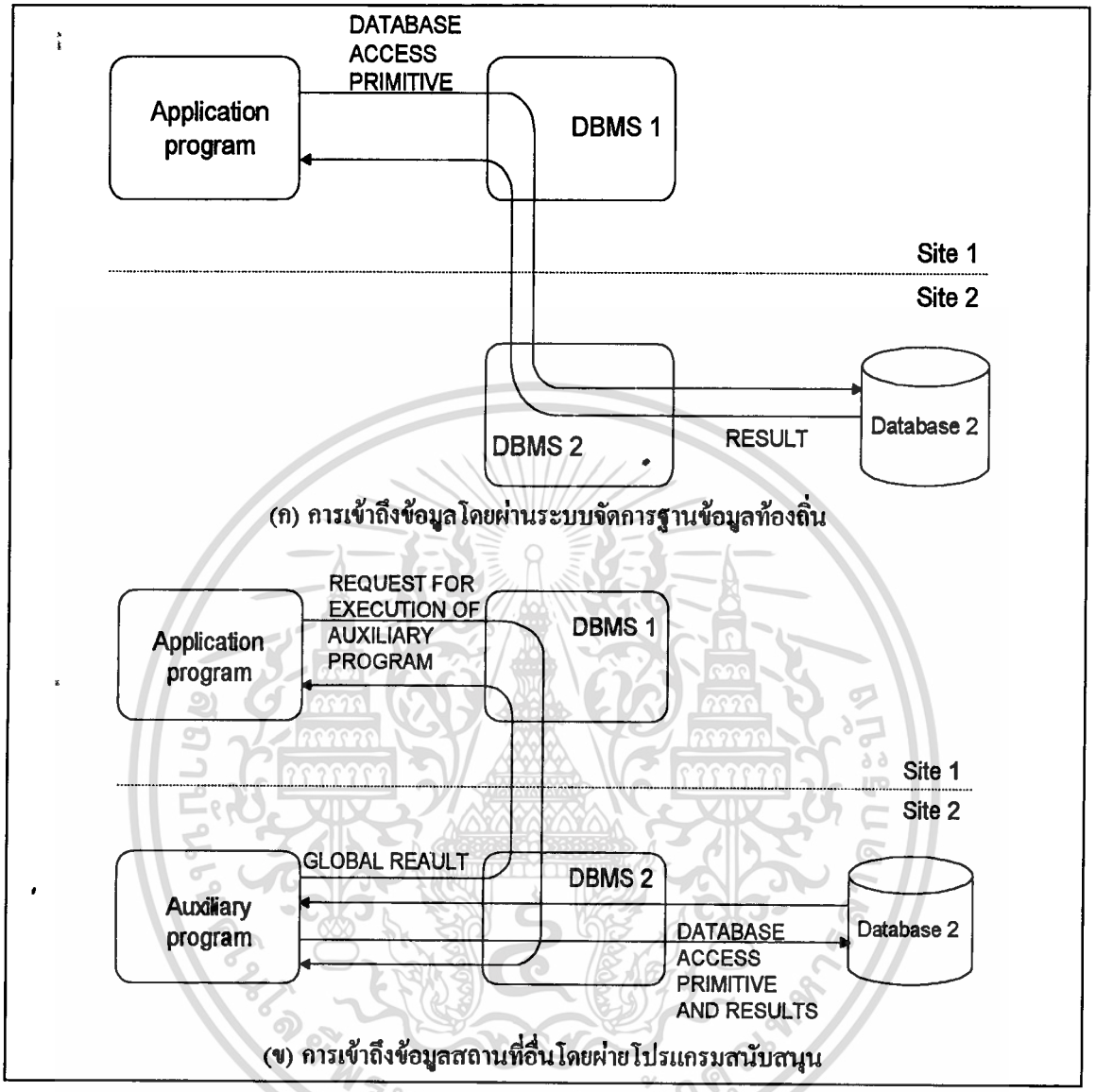
1. ส่วนจัดการฐานข้อมูล (DB)
2. ส่วนสื่อสารข้อมูล (DC)
3. ส่วนพจนานุกรมฐานข้อมูล (DD)
4. ส่วนเชื่อมต่อฐานข้อมูลแบบกระจาย (DDB)

จากภาพที่ 2.4 เราจะเรียกส่วนประกอบทั้ง 4 ส่วนว่า ระบบจัดการฐานข้อมูลแบบกระจาย ซึ่งส่วนประกอบที่จัดว่ามีความสำคัญได้แก่ ส่วนเชื่อมต่อฐานข้อมูลแบบกระจาย ซึ่งจะมีหน้าที่ในการติดต่อกับฐานข้อมูลสถานที่อื่น

ในการเข้าถึงข้อมูลที่อยู่ต่างสถานที่ (remote site) นั้น สามารถกระทำได้ 2 วิธี ดังแสดงในรูปที่ 2.5 คือ



รูปที่ 2.4 แสดงส่วนประกอบและการต่อเชื่อมของระบบฐานข้อมูลแบบกระจาย



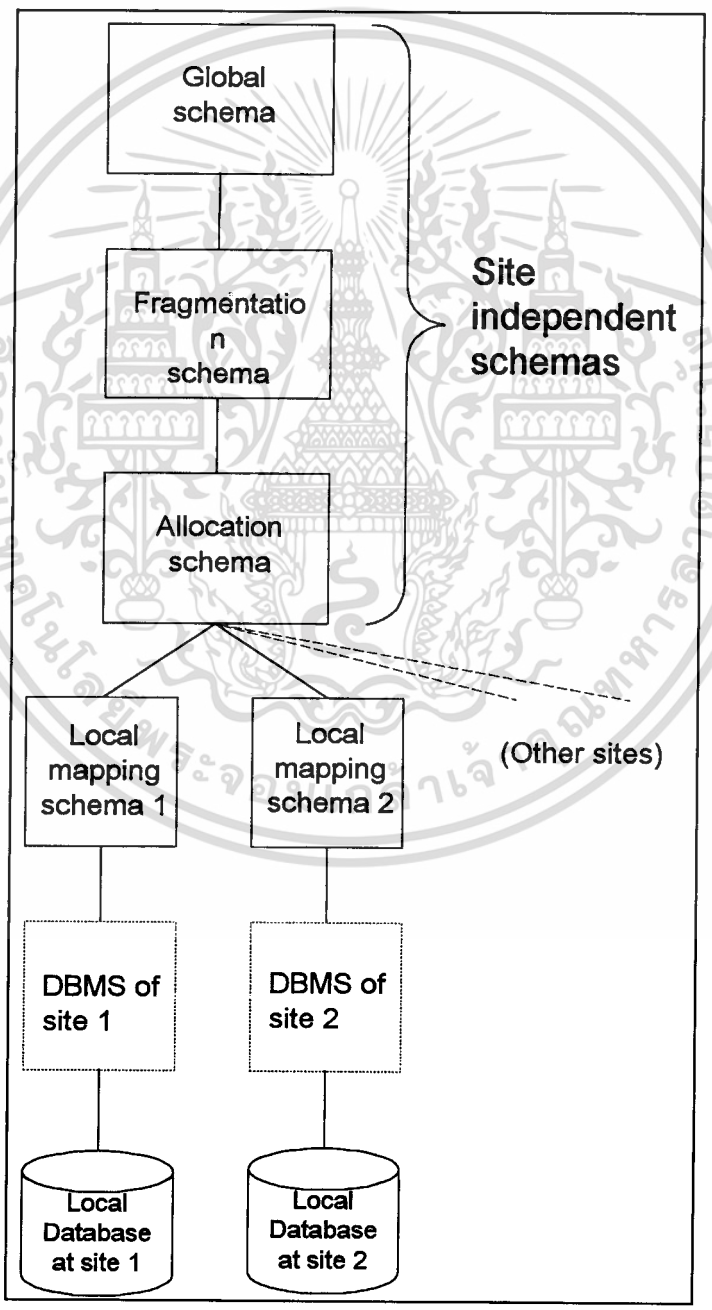
รูปที่ 2.5 แสดงวิธีการเข้าถึงข้อมูลในระบบฐานข้อมูลแบบกระจาย

วิธีแรก ดังรูปที่ 2.5(ก) โปรแกรมประยุกต์จะแสดงความต้องการข้อมูลกับระบบจัดการฐานข้อมูลแบบกระจายท้องถิ่น โดยอ้างถึงสถานที่ของฐานข้อมูล ระบบจัดการฐานข้อมูลท้องถิ่นจะค้นหาเส้นทางการเข้าถึงข้อมูล จากนั้นจะส่งความต้องการให้กับระบบจัดการฐานข้อมูลแบบกระจายของฐานข้อมูลนั้นๆ พร้อมทั้งรอรับข้อมูลที่ต้องการ ซึ่งวิธีนี้การติดต่อจะกระทำระหว่างระบบจัดการฐานข้อมูลกับระบบจัดการฐานข้อมูล

วิธีที่สอง แสดงในรูปที่ 2.5(ข) เมื่อโปรแกรมประยุกต์แสดงความต้องการข้อมูลต่อระบบจัดการฐานข้อมูลแบบกระจายท้องถิ่นแล้ว ระบบจัดการฐานข้อมูลแบบกระจายท้องถิ่น ก็จะค้นหาสถานที่เก็บข้อมูลที่ต้องการ จากนั้นก็จะส่งความต้องการให้กับโปรแกรมสนับสนุน ที่ทำงานในสถานที่เก็บข้อมูลนั้นๆ โปรแกรมสนับสนุนก็จะทำการประมวลผล และจัดส่งข้อมูลให้กับระบบจัดการฐานข้อมูลที่ส่งความต้องการมา ซึ่งจะเห็นว่าวิธีนี้เป็นการติดต่อระหว่างระบบจัดการฐานข้อมูลท้องถิ่นกับโปรแกรมสนับสนุน

2.1.3 ระดับการส่งผ่านการกระจายของฐานข้อมูล (Levels of Distribution Transparency)

การส่งผ่านการกระจายของฐานข้อมูล กล่าวโดยสรุปหมายถึง ลักษณะการเข้าถึงข้อมูลในระบบฐานข้อมูลแบบกระจายของโปรแกรมใช้งานประยุกต์ ว่าจะมีความเป็นอิสระ ไม่ขึ้นกับการกระจายของข้อมูลมากน้อยเพียงไร หรือกล่าวอีกนัยหนึ่งจะได้ว่า มีลักษณะการเข้าถึงข้อมูลคล้ายกับการเข้าถึงข้อมูลในระบบฐานข้อมูลแบบรวมมากน้อยเท่าใด ซึ่งระดับการส่งผ่านสามารถที่จะจัดแบ่งตามลักษณะแบบแผนการอ้างถึงข้อมูล ได้เป็น 3 ระดับ คือ



รูปที่ 2.6 แสดงสถาปัตยกรรมการอ้างถึงข้อมูลของฐานข้อมูลแบบกระจาย

2.1.3.1 แบบแผนโดยรวม (Global schema)

จากรูปที่ 2.6 เป็นการแสดงสถาปัตยกรรมการอ้างถึงข้อมูล ของระบบฐานข้อมูลแบบกระจาย ส่วนที่อยู่บนสุดคือแบบแผนการอ้างถึงโดยรวม โปรแกรมประยุกต์ที่มีการอ้างถึงข้อมูลโดยใช้แบบแผนนี้ จะอ้างถึงข้อมูลที่สถานที่ต่างๆ เสมือนกับว่าข้อมูลทั้งหมดไม่มีการกระจายเลย ซึ่งจะเหมือนกับการอ้างถึงข้อมูลในฐานข้อมูลแบบรวม และความสัมพันธ์ที่ถูกอ้างถึงนี้ จะเรียกว่า ความสัมพันธ์โดยรวม (Global relations)

2.1.3.2 แบบแผนส่วนย่อย (Fragmentation schema)

การอ้างถึงโดยแบบแผนส่วนย่อยนี้ เป็นการเข้าถึงข้อมูลโดย อ้างถึงบางส่วนของความสัมพันธ์โดยรวมที่ถูกแบ่งออกเป็นส่วนๆไม่ซ้ำกัน โดยจะเรียกแต่ละส่วนว่า ส่วนย่อย (fragment) ซึ่งวิธีการแบ่งความสัมพันธ์โดยรวมออกเป็นส่วนย่อยนี้ สามารถแบ่งได้หลายวิธี เช่น แบ่งตามแนวนอน (Horizontal Fragmentation) คือ แยกทัปเปิล (tuples) ออกเป็นกลุ่มๆ หรือแบ่งตามแนวตั้ง (Vertical Fragmentation) คือ แยกแอททริบิวออกเป็นกลุ่มๆ ส่วนย่อยที่แบ่งออกมานี้ สามารถที่จะนำกลับมารวมเข้าเป็นความสัมพันธ์โดยรวมได้ดังเดิม ด้วยการทำ UNION และ JOIN ตามลำดับ อย่างไรก็ตามการแบ่งส่วนย่อยนี้ สามารถที่จะกระทำทั้งสองวิธีผสมกัน (Mixed Fragmentation) ได้ ทั้งนี้จะต้องสามารถนำกลับมารวมเป็นความสัมพันธ์โดยรวมได้ดังเดิม

2.1.3.3 แบบแผนการอ้างถึงตำแหน่ง (Allocation schema)

ส่วนย่อยที่แยกมาจากความสัมพันธ์โดยรวมนั้น จะถูกกำหนดสถานที่ที่จะเก็บข้อมูลส่วนนั้นๆ (Local mapping schema) ซึ่งในแต่ละส่วนย่อยนี้ อาจจะมีสถานที่เก็บข้อมูลมากกว่าหนึ่งแห่งได้ ทั้งนี้ขึ้นอยู่กับการตัดสินใจของผู้ออกแบบระบบฐานข้อมูล ว่าจะยอมให้มีการซ้ำซ้อนของข้อมูลมากน้อยเพียงใด และสอดคล้องกับการใช้งานหรือไม่

2.1.4 การออกแบบฐานข้อมูลแบบกระจาย (Distributed Database Design)

ในการออกแบบฐานข้อมูลแบบกระจาย โดยทั่วไปนับว่าเป็นขั้นตอนที่มีความสำคัญขั้นตอนหนึ่ง ของการสร้างระบบฐานข้อมูลแบบกระจายและมีความซับซ้อนพอสมควร เนื่องจากการออกแบบฐานข้อมูลแบบกระจาย จะต้องคำนึงถึงปัจจัยหลายด้าน เช่น ปัญหาและความเป็นไปได้ทางเทคนิค, ข้อจำกัดของการจัดการองค์กร เป็นต้น

การออกแบบฐานข้อมูลแบบกระจายโดยทั่วไป จะมีอยู่ 2 วิธีด้วยกัน คือ วิธีการออกแบบจากบนลงล่าง และ การออกแบบจากล่างขึ้นบน ทั้งสองวิธีมีลำดับในการออกแบบที่แตกต่างกัน รวมทั้งมีความเหมาะสมที่จะใช้งานต่างกันด้วย ซึ่งสรุปได้ดังนี้

2.1.4.1 การออกแบบฐานข้อมูลจากบนลงล่าง (Top-Down design)

ในการออกแบบจากบนลงล่างนี้ ขั้นตอนเริ่มจากการวิเคราะห์ คือศึกษาข้อมูลและความต้องการของผู้ใช้งาน ผลของการวิเคราะห์จะนำมาใช้ในการออกแบบ Global Conceptual Schema เมื่อเสร็จขั้นตอนนี้จะได้ความสัมพันธ์โดยรวม (Global Relations) ของระบบ เพื่อใช้ในขั้นตอนแยกเป็นส่วนย่อย (Fragmentation) และ

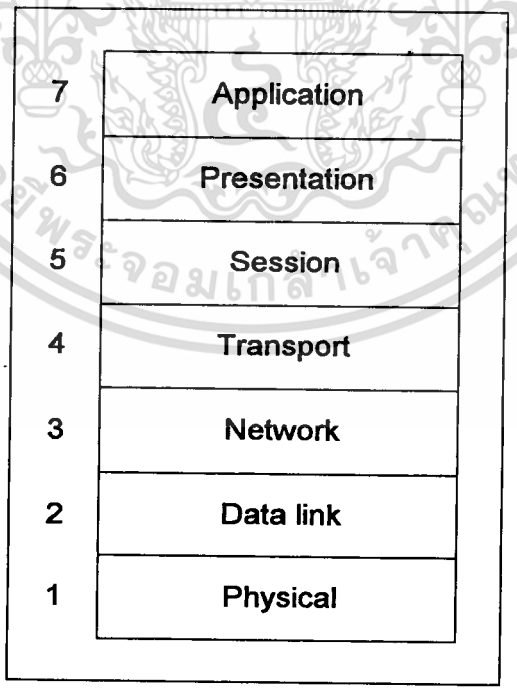
จากขั้นตอนที่กล่าวมาข้างต้นนั้น จะเห็นว่าเป็นการออกแบบระบบจากระบบฐานข้อมูลกระจายโดยรวมไปสู่ฐานข้อมูลในแต่ละสถานที่ ซึ่งลักษณะการออกแบบชนิดนี้เหมาะสมที่จะใช้ออกแบบระบบฐานข้อมูลแบบกระจายใหม่ทั้งระบบ

2.1.4.2 การออกแบบฐานข้อมูลจากล่างขึ้นบน (Buttom-Up Design)

การออกแบบจากล่างขึ้นบน เป็นลักษณะการออกแบบฐานข้อมูลแบบกระจาย จากฐานข้อมูลที่มีอยู่แล้วหลายสถานที่ ซึ่งขั้นตอนส่วนใหญ่ จะเป็นการรวม Conceptual Schema ของแต่ละสถานที่ให้เป็น Global Conceptual Schema ของระบบ จากนั้นจึงเป็นการพัฒนาระบบ และปรับแต่งการทำงานของระบบต่อไป ซึ่งการออกแบบลักษณะนี้เหมาะกับการสร้างระบบฐานข้อมูลแบบกระจาย จากฐานข้อมูลแบบรวมที่มีอยู่แล้ว

2.2 TCP/IP

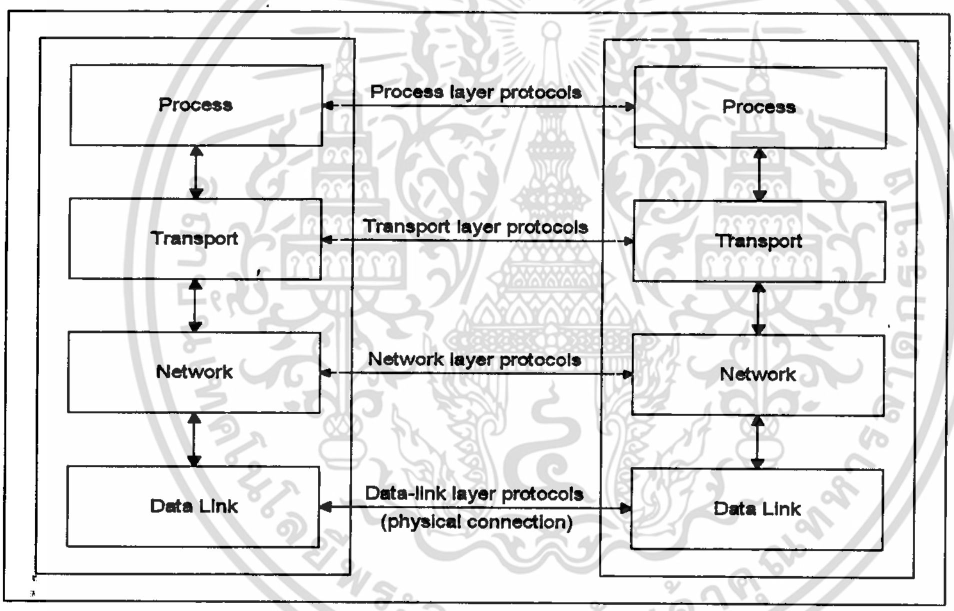
ในการติดต่อสื่อสารระหว่างกระบวนการประมวลผลนั้น โดยปรกติในแต่ละกระบวนการประมวลผลจะแบ่งการทำงานออกเป็นส่วนๆ แต่ละส่วนเรียกว่า เลเยอร์ (Layer) แต่ละเลเยอร์จะทำงานแตกต่างกันไปตามหน้าที่ที่ได้กำหนดไว้ รวมทั้งยังสามารถจะสื่อสารกับเลเยอร์ที่อยู่ใกล้เคียงกันได้ ซึ่งองค์การมาตรฐานสากล (International Standards Organization :ISO) ได้กำหนดรูปแบบสำหรับการติดต่อแบบระบบเปิด (Open System Interoonnection : OSI) ไว้ 7 ระดับชั้นเลเยอร์ ดังรูปที่ 2.7



รูปที่ 2.7 แสดงรูปแบบการติดต่อแบบระบบเปิด 7 ชั้นเลเยอร์ (OSI 7-Layer model)

ในจำนวนเลเยอร์ 7 ชั้นที่แสดงไว้ในรูปที่ 2.7 นั้น ชั้นเลเยอร์ transport นับว่าเป็นชั้นเลเยอร์ที่มีความสำคัญมากที่สุด เนื่องจากว่าเป็นชั้นเลเยอร์ที่ต่ำที่สุด ที่มีความเชื่อถือได้ในการส่งข้อมูล การเรียกใช้จากชั้นเลเยอร์ที่สูงกว่า จะถือว่าชั้นเลเยอร์ transport นี้ ไม่มีความผิดพลาดในการส่งข้อมูล ซึ่งขั้นตอนต่างๆ เช่น ควบคุมลำดับการส่ง ตรวจสอบความผิดพลาดในการส่งข้อมูล หรือ ส่งข้อมูลซ้ำ เป็นต้น จะถูกจัดการโดยชั้นเลเยอร์นี้ และชั้นเลเยอร์ที่ต่ำกว่า

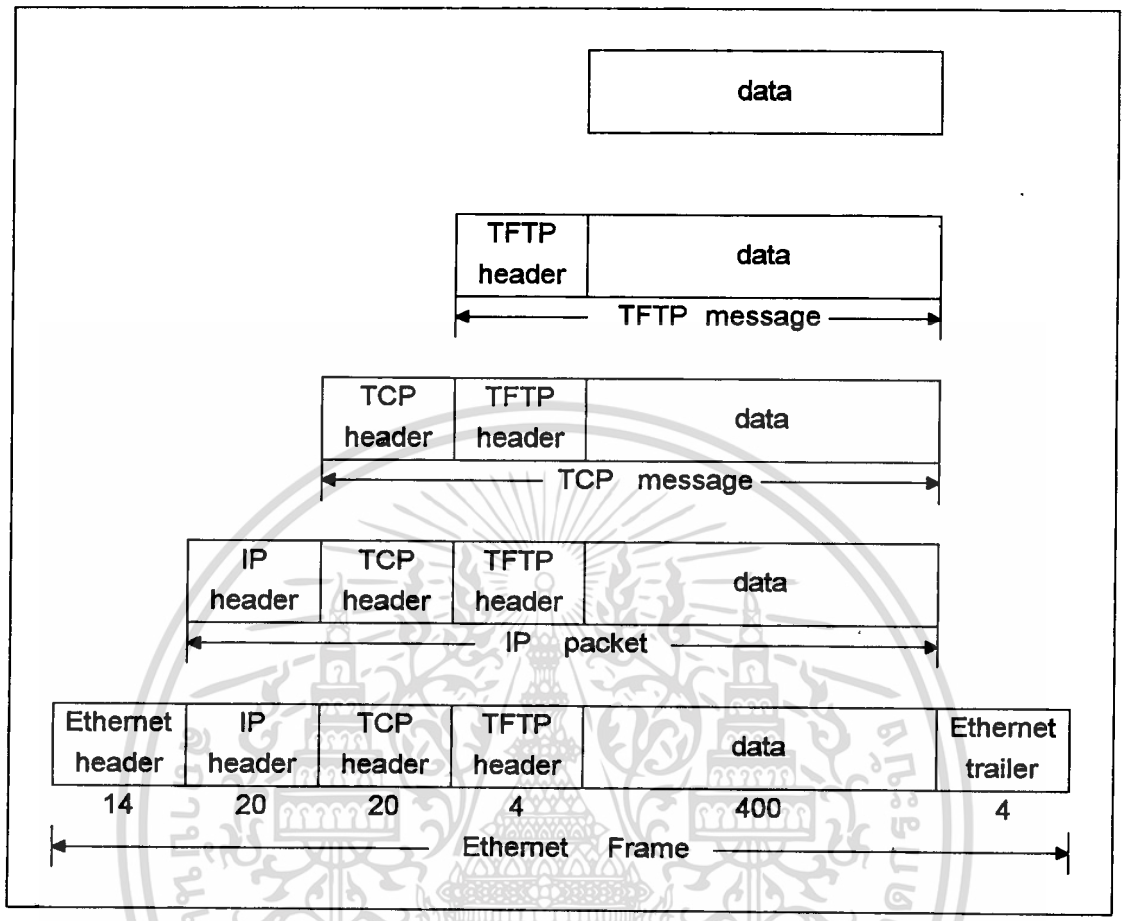
โดยทั่วไปแล้ว โปรแกรมประยุกต์มักจะติดต่อกันด้วยโปรโตคอลในสามเลเยอร์ คือ session, presentation และ application ดังนั้น ทั้งสามชั้นเลเยอร์จึงมักถูกเรียกว่าชั้นเลเยอร์ process รวมทั้งชั้นเลเยอร์ data link และชั้นเลเยอร์ physical มักจะเรียกรวมกันว่าชั้นเลเยอร์ data-link ดังแสดงในรูปที่ 2.8



รูปที่ 2.8 แสดงรูปแบบการติดต่ออย่างง่าย 4 ชั้นเลเยอร์

ในรูปที่ 2.8 เป็นการแสดงการเชื่อมต่อระบบ 2 ระบบด้วยโครงข่าย การเชื่อมต่อระบบทั้งสองจะเกิดขึ้นจริงๆที่ชั้นเลเยอร์ data-link (แสดงด้วยเส้นทึบ) แม้ว่ากระบวนการประมวลผลทั้งสองจะเสมือนติดต่อกันได้จริง (แสดงด้วยเส้นประ) แต่ข้อมูลจะถูกส่งผ่านจากชั้นเลเยอร์ process ลงมายังชั้นเลเยอร์ transport ชั้นเลเยอร์ network และชั้นเลเยอร์ data-link ข้ามโครงข่ายผ่านไปยังชั้นเลเยอร์ data-link ของอีกระบบหนึ่ง และผ่านชั้นเลเยอร์ network ขึ้นไปยังชั้นเลเยอร์ transport แล้วส่งให้กับระดับชั้นเลเยอร์ process จะเห็นว่าข้อมูลถูกส่งผ่านระหว่างชั้นเลเยอร์ ที่อยู่ติดกันทั้งบนและล่าง ดังนั้นการเชื่อมต่อ (interface) ระหว่างชั้นเลเยอร์นับว่ามีความสำคัญสำหรับการเขียนโปรแกรมโครงข่ายเป็นอย่างมาก

ในหัวข้อนี้ จะได้กล่าวถึงรายละเอียดของวิธีการเชื่อมต่อ ระหว่างชั้นเลเยอร์ Transport กับชั้นเลเยอร์ที่สูงกว่า ด้วยวิธีการเชื่อมต่อที่นิยมใช้กัน คือวิธีการเชื่อมต่อโดยการใช้ซอคเกต (socket) ของเบิคเลย์ (Berkeley) ด้วยการใช้โปรโตคอล TCP/IP ที่เรียกใช้ด้วยโปรแกรมที่เขียนด้วยภาษา C บนระบบปฏิบัติการยูนิกซ์เท่านั้น

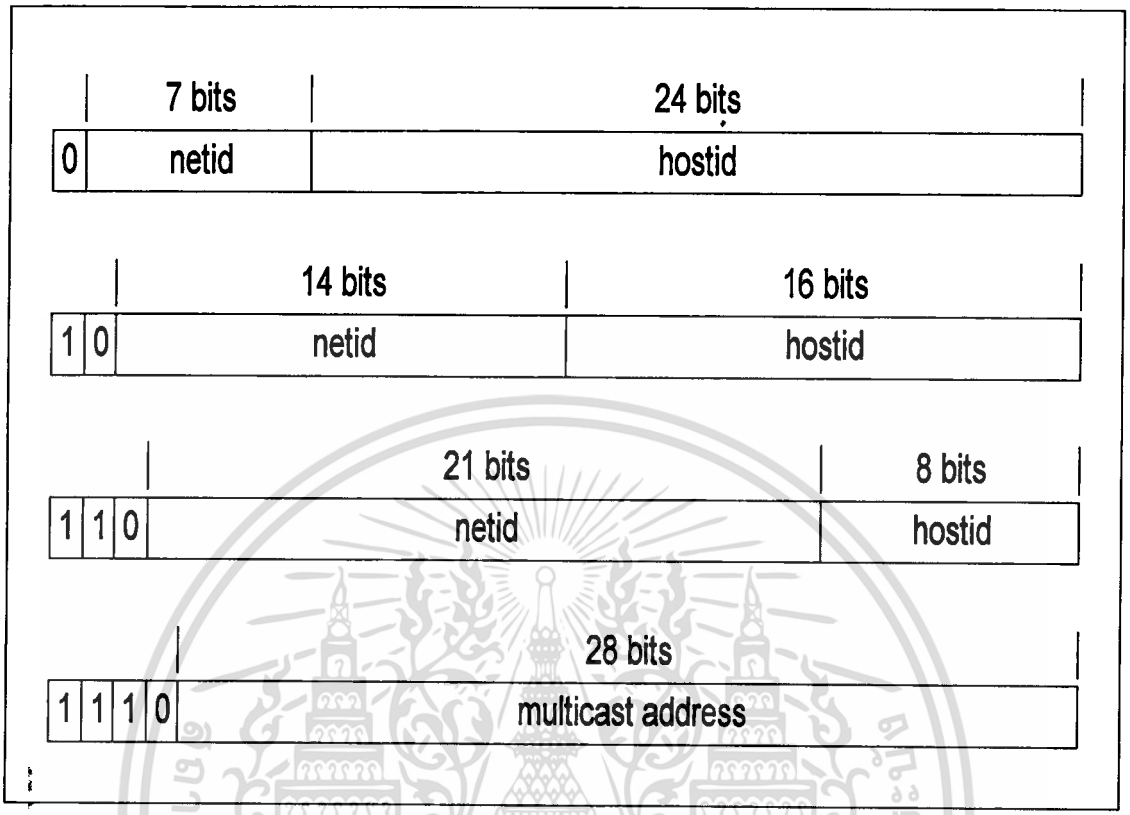


รูปที่ 2.9 แสดงการใส่เปลือกครอบรายละเอียดของโปรโตคอลแต่ละชั้น

2.2.1 การใส่เปลือกครอบ (Encapsulation)

จากที่ได้กล่าวมาข้างต้นนั้น เป็นที่ทราบแล้วว่า ในการส่งข้อมูลของแต่ละเลเยอร์ จะทำการส่งข้อมูลผ่านไปยังชั้นเลเยอร์ที่ต่ำกว่า ซึ่งชั้นเลเยอร์ที่ต่ำกว่าจะถือว่าข้อมูลที่ได้รับเป็นข้อมูลจริงที่ต้องส่งต่อ ก็จะทำการเพิ่มข้อมูลรายละเอียดของโปรโตคอลในชั้นเลเยอร์นั้นครอบข้อมูลจริงแล้วส่งต่อไปยังชั้นเลเยอร์ที่ต่ำกว่า ซึ่งชั้นเลเยอร์ที่ต่ำกว่าก็จะถือว่าข้อมูลที่ได้รับเป็นข้อมูลจริง และเพิ่มข้อมูลครอบเข้าไป จะเป็นเช่นนี้ไปเรื่อยๆจนกว่าจะถึงชั้นที่มีการส่งข้อมูลทางกายภาพจริงๆ จึงจะส่งข้อมูลข้ามโครงข่าย ส่วนฝ่ายรับข้อมูล จะรับข้อมูลที่ชั้นเลเยอร์ที่ต่ำสุด และจะถอดข้อมูลรายละเอียดโปรโตคอลของชั้นนั้นออก จากนั้นจะส่งข้อมูลให้กับชั้นเลเยอร์ที่สูงกว่าเป็นเช่นนี้ไปเรื่อยๆจนกว่าจะถึงชั้นเลเยอร์ระดับเดียวกันกับฝ่ายส่ง

ในรูปที่ 2.9 โปรแกรมประยุกต์ TFTP ใช้โปรโตคอล TCP/IP และเชื่อมต่อ 2 ระบบด้วยอีเธอร์เนท (Ethernet) ถ้าต้องการส่งไฟล์ข้อมูลขนาด 400 ไบต์ จะเห็นว่า TFTP จะเพิ่มข้อมูล 4 ไบต์ และ TCP จะเพิ่มข้อมูล 20 ไบต์ จนสุดท้ายข้อมูลที่ส่งผ่าน อีเธอร์เนททั้งหมดจะเป็น 462 ไบต์

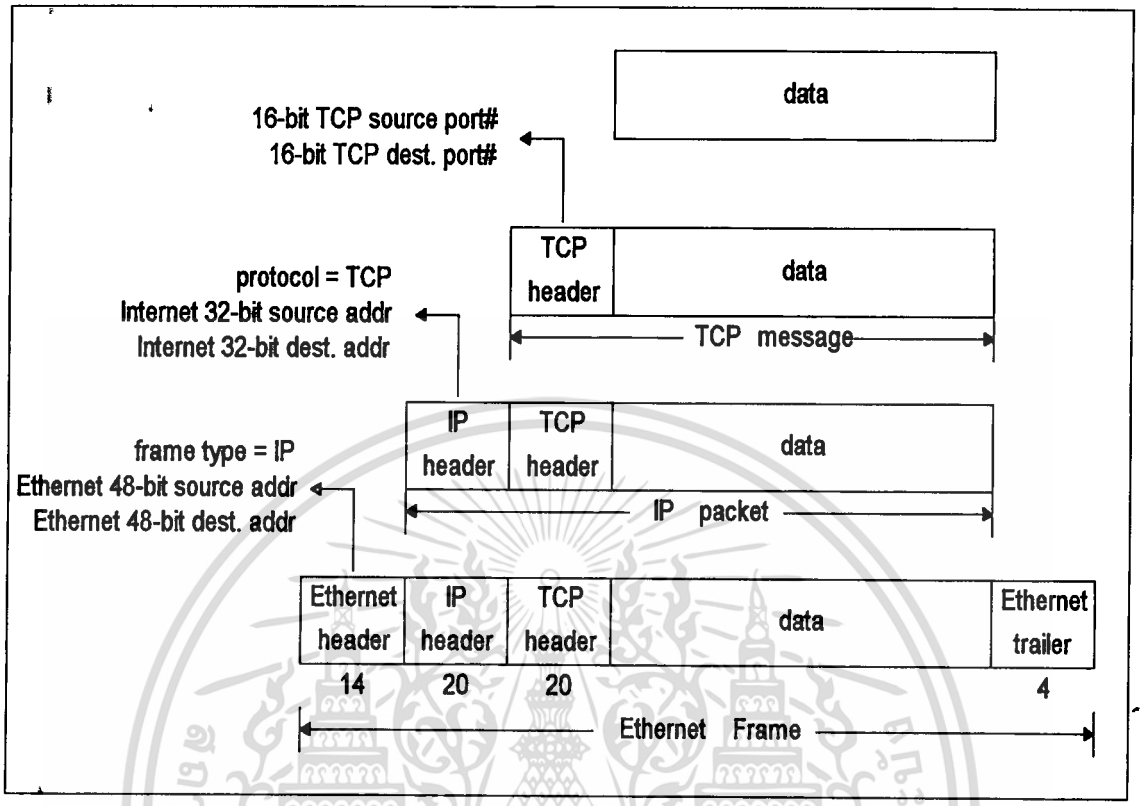


รูปที่ 2.10 แสดงรูปแบบที่อยู่แบบอินเทอร์เนท

2.2.2 ที่อยู่แบบอินเทอร์เนท (Internet Addresses)

โดยทั่วไปแล้ว โปรโตคอลแต่ละชนิดจะมีการกำหนดที่อยู่ (address) สำหรับคอมพิวเตอร์ เพื่อที่จะระบุโครงข่าย และเครื่องคอมพิวเตอร์ที่ต้องการอ้างถึง สำหรับที่อยู่แบบอินเทอร์เนทนี้ จะประกอบด้วยข้อมูล 32 บิท ซึ่งข้อมูลนี้จะเก็บหมายเลขโครงข่าย (network ID) และหมายเลขคอมพิวเตอร์ (host ID) ดังนั้นที่อยู่หนึ่งๆจะสามารถที่จะระบุคอมพิวเตอร์ได้เพียงเครื่องเท่านั้น อย่างไรก็ตาม ในกรณีที่เครื่องคอมพิวเตอร์อยู่ในโครงข่ายมากกว่าหนึ่งโครงข่าย ก็จะมีที่อยู่ได้มากกว่าหนึ่งที่อยู่ โดยจะสอดคล้องกับจำนวนโครงข่ายที่คอมพิวเตอร์เครื่องนั้นอยู่ ที่อยู่แบบอินเทอร์เนทนี้ จะถูกกำหนดโดยศูนย์ข้อมูลโครงข่าย (Network Information Center : NIC) ซึ่งจะมีรูปแบบเป็นหนึ่งในสี่รูปแบบที่แสดงไว้ในรูปที่ 2.10

ปรกติแล้ว ที่อยู่แบบอินเทอร์เนท มักจะถูกเขียนในรูปของตัวเลข ที่ถูกแบ่งด้วยจุดทศนิยมออกเป็น 4 ชุด ตัวเลขแต่ละชุดจะแทนข้อมูล 1 ไบต์ของที่อยู่ ตัวอย่าง เช่น ที่อยู่ 0x0102FF04 จะเขียนได้เป็น 1.2.255.4. ซึ่งในระบบปฏิบัติการยูนิกซ์ จะเก็บที่อยู่แบบตัวเลขไว้ในไฟล์ชื่อ /etc/hosts โดยไฟล์ /etc/hosts นี้จะเก็บที่อยู่ของคอมพิวเตอร์ทุกเครื่อง ที่อยู่ในโครงข่ายเดียวกันกับคอมพิวเตอร์ที่มีไฟล์ /etc/hosts อยู่



รูปที่ 2.11 แสดงการใส่เปลือกครอบของข้อมูล TCP บนการเชื่อมต่ออีเธอร์เนท

2.2.3 หมายเลขพอร์ท (Port Numbers)

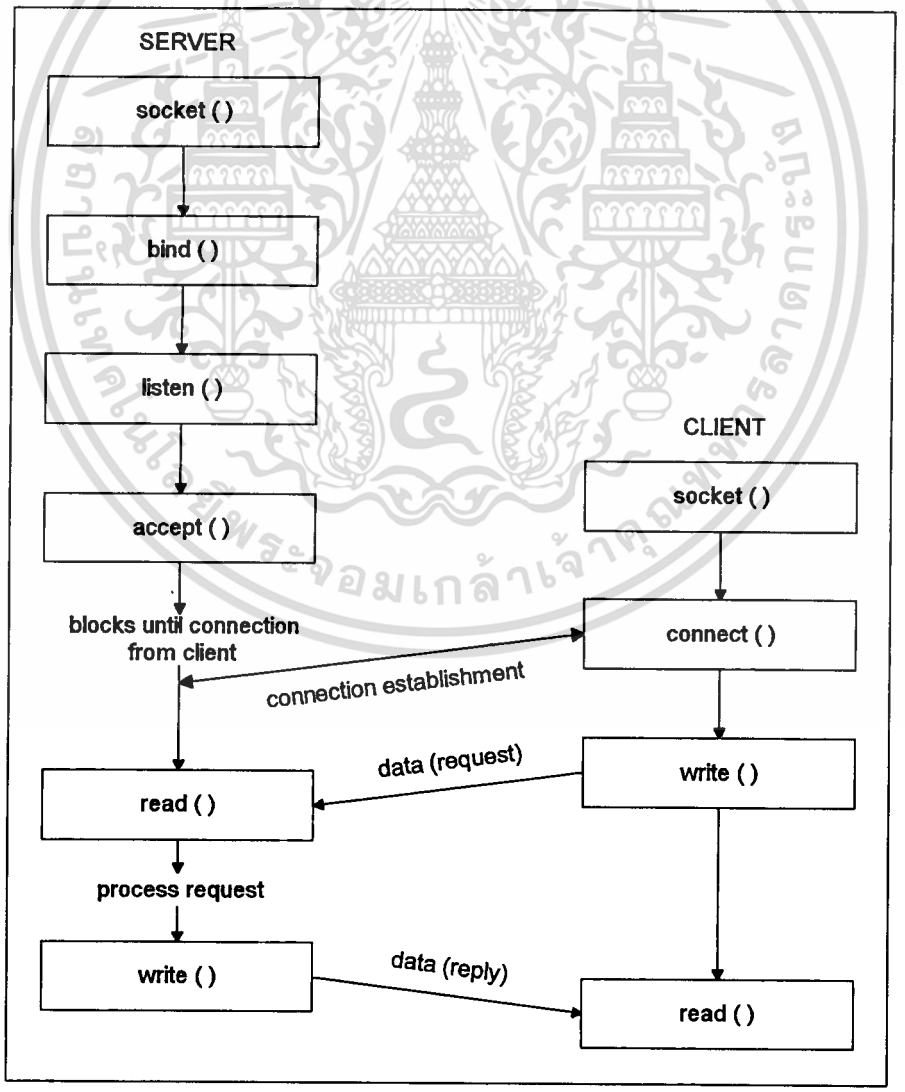
จากการที่ได้มีการกำหนดที่อยู่แบบอินเทอร์เนท ทำให้สามารถที่จะระบุคอมพิวเตอร์ที่ต้องการจะส่งข้อมูลไปถึงได้ อย่างไรก็ตาม ในคอมพิวเตอร์เครื่องหนึ่งๆอาจจะมีกระบวนการมากกว่าหนึ่งกระบวนการที่ใช้โปรโตคอล TCP/IP ได้ ดังนั้นจึงมีการกำหนดหมายเลขพอร์ท (Port Number) ขึ้น เพื่อใช้ระบุถึงกระบวนการที่ต้องการติดต่อ ซึ่งหมายเลขพอร์ทนี้จะเป็นข้อมูลขนาด 16 บิท กระบวนการบริการ (server) จะได้รับการกำหนดหมายเลขพอร์ทไว้ เพื่อรอรับการติดต่อจากกระบวนการลูกข่าย (client) โดยหมายเลขพอร์ทมักจะเป็นที่ทราบกันโดยทั่วไปในการติดต่อ (well-know port) เช่น โปรแกรมประยุกต์ FTP จะใช้หมายเลขพอร์ท 21 สำหรับกระบวนการบริการที่คอยรับการติดต่อจากกระบวนการลูกข่าย ที่ต้องการรับ-ส่งไฟล์

เมื่อกระบวนการลูกข่ายติดต่อไปยังกระบวนการบริการ กระบวนการบริการจะทราบได้ว่ามีการติดต่อมาจากคอมพิวเตอร์หมายเลขใด โดยทราบได้จากข้อมูลที่อยู่แบบอินเทอร์เนทขนาด 32 บิทในรายละเอียดของ IP (IP datagram) และหมายเลขพอร์ทของกระบวนการลูกข่ายจากหมายเลขพอร์ทขนาด 16 บิท ที่อยู่ในรายละเอียด TCP ดังแสดงไว้ในรูปที่ 2.11 ซึ่งหมายเลขพอร์ทนี้จะถูกกำหนดโดยคอมพิวเตอร์ท้องถิ่น เรียกว่า ephemeral port number เพื่อให้กระบวนการลูกข่ายใช้ในการติดต่อกับกระบวนการบริการ หมายเลขพอร์ท ephemeral port number ที่กำหนดโดยคอมพิวเตอร์ท้องถิ่น จะมีค่าไม่ซ้ำกันในการกำหนดให้แต่ละกระบวนการลูกข่าย รวมทั้งหมายเลขพอร์ทที่ผู้เขียนโปรแกรมกำหนดให้กับกระบวนการบริการก็จะต้องไม่ซ้ำกันด้วย ดังนั้นจึงมีหลักเกณฑ์ง่ายๆในการใช้หมายเลขพอร์ทดังนี้

ในการกำหนดหมายเลขพอร์ทของโปรโตคอลกลุ่มอินเทอร์เนท จะใช้หมายเลขพอร์ท 1 ถึง 255 เป็นหมายเลขพอร์ทสำรอง (reserved port) ซึ่งหมายเลขพอร์ทที่ทราบโดยทั่วไปก็จะอยู่ในกลุ่มนี้ ระบบปฏิบัติการบางรุ่น เช่น 4.3BSD จะสำรองพอร์ทหมายเลข 1 ถึง 1023 ไว้สำหรับผู้ดูแลระบบเป็นผู้เรียกใช้ ส่วนงานทั่วไปมักจะถูกกำหนดหมายเลขพอร์ทในช่วง 1024 ถึง 5000 อย่างไรก็ตาม สำหรับผู้ใช้ที่เป็นใหม่ๆควรเลือกใช้หมายเลขพอร์ทมากกว่า 5000 ขึ้นไป

2.2.4 การเรียกใช้ฟังก์ชันสำหรับโปรโตคอล TCP/IP

ในการเรียกใช้ฟังก์ชัน เพื่อเขียนโปรแกรมรับ-ส่งข้อมูล ที่ใช้โปรโตคอล TCP/IP นั้น สามารถที่จะเรียกใช้ได้ 2 วิธีคือ connection-oriented protocol ซึ่งส่วนรับและส่งข้อมูลจะสร้างการติดต่อทางลอกจิกก่อนที่จะมีการรับ-ส่งข้อมูลกัน และ connectionless protocol ซึ่งจะเป็นลักษณะที่ตรงกันข้ามกับวิธีแรก ตัวอย่างที่แสดงในรูป 2.11 เป็นการเรียกใช้ฟังก์ชัน โดยวิธี connection-oriented protocol สำหรับรายละเอียดจะได้กล่าวเป็นหัวข้อต่อไป



รูปที่ 2.12 แสดงลำดับการเรียกใช้ฟังก์ชัน ในการสื่อสารด้วยซอคเกต

2.2.4.1 โครงสร้างของตัวแปรที่อยู่

จากที่ได้กล่าวมาข้างต้นแล้วว่า การเขียนโปรแกรมรับ-ส่งข้อมูลทางระบบสื่อสารนั้น ตัวโปรแกรมจำเป็นต้องทราบ address ที่จะส่งไป หรือที่จะรับมา แน่นอน ซึ่ง sooket address เป็นรูปแบบโครงสร้างของ address ที่ใช้ใน Berkley Sooket กำหนดไว้ใน <sys/sooket.h> ดังนี้

```
struot sookaddr {
  u_short sa_family ;        /* address family: AF_xxx value */
  ohar    sa_data[14] ;      /* up to 14 bytes of protoool-specifio address */
} ;
```

ลักษณะของโครงสร้าง sookaddr นี้ จะใช้กับทุกโปรโตคอลที่เรียกใช้ฟังก์ชัน sooket สำหรับ internet family แล้ว โครงสร้างของ sookaddr จะกำหนดไว้ใน <netinet/in.h> ดังนี้

```
struot in_addr {
  u_long s_addr ; /* 32-bit netid/hostid */
   /* network byte ordered */
} ;
struot sookaddr_in {
  short    sin_family ;       /* AF_INET */
  u_short sin_port ;          /* 16-bit port number */
  /* network byte ordered */
  struot in_addr    sin_addr ;        /* 32-bit netid/hostid */
  /* network byte ordered */
  ohar     sin_zero[8] ;      /* unused */
} ;
```

2.2.4.2 ฟังก์ชันเรียกใช้ (function calls)

รายละเอียดของฟังก์ชันต่างๆที่ควรทราบ สำหรับการใช้โปรโตคอล TCP/IP มีดังต่อไปนี้ คือ

2.2.4.2.1 sooket

คำสั่งนี้จะทำหน้าที่จัดการกับ network I/O ซึ่งเป็นคำสั่งแรกของกลุ่มที่จะต้องเรียกใช้ โดยการเรียกใช้จะต้องกำหนดโปรโตคอลที่ต้องการใช้ มีรูปแบบการใช้ดังนี้

```
#inolude <sys/types.h>
#inolude <sys/sooket.h>
int        sooket (int family, int type, int protoool) ;
```

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	Yes	TCP	SPP
SOCK_DGRAM	Yes	UDP	IDP
SOCK_ARW		IP	Yes
SOCK_SEQPACKET			SPP

รูปที่ 2.13 แสดงโปรโตคอลที่สอดคล้องกันระหว่าง socket family และ type

family คือกลุ่มของโปรโตคอลที่เรียกใช้ ได้แก่

AF_UMIX

AF_INET

AF_NS

AF_IMPLINK

type เป็นการกำหนดลักษณะการส่งข้อมูลได้แก่

SOCK_STREAM

SOCK_DGRAM

SOCK_RAW

SOCK_SEQPACKET

SOCK_RDM

ซึ่งการเลือก type นี้จะต้องสัมพันธ์กับกลุ่มโปรโตคอลที่เลือกใช้ และโปรโตคอลที่ใช้ด้วย ดังแสดงในรูปที่ 2.13

protocol ปรกติแล้วพารามิเตอร์ตัวนี้ จะกำหนดเป็น 0

2.2.4.2.2 bind

เป็นการกำหนดชื่อ (address) ให้แก่ socket ที่ยังไม่มีชื่อ มีการใช้ดังนี้

```
#include <sys/types.h>
#include <sys/socket.h>
int     bind ( int sockfd, struct sockaddr *myaddr, int addrlen) ;
```

พารามิเตอร์ที่ 2 จะเป็นพอยน์เตอร์ที่ชี้ไปยัง address ของโปรโตคอลที่ระบุ และพารามิเตอร์ที่ 3 จะเป็นขนาดของ address นั้น

2.2.4.2.3 connect

กระบวนการลูกข่าย จะติดต่อกับกระบวนการบริการ โดยการเรียกใช้ฟังก์ชันนี้ ต่อจากฟังก์ชัน socket ซึ่งมีการใช้ดังนี้

```
#include <sys/types.h>
#include <sys/socket.h>
int     connect ( int sockfd, struct sockaddr *servaddr, int addrlen) ;
```

2.2.4.2.4 listen

ฟังก์ชันนี้ใช้สำหรับกระบวนการบริการแบบ connection-oriented เพื่อที่จะแสดงว่ากระบวนการบริการกำลังรอรับการติดต่อจากกระบวนการลูกข่าย ซึ่งมีการใช้ดังนี้

```
int     listen (int sockfd, int backlog) ;
```

โดยปรกติฟังก์ชัน listen จะเรียกใช้ตามหลังฟังก์ชัน socket และ bind แต่จะก่อน ฟังก์ชัน accept พารามิเตอร์ backlog เป็นจำนวนที่ระบุถึงจำนวนการติดต่อจากกระบวนการลูกข่ายที่จะสามารถรอในคิวได้ ขณะที่กระบวนการบริการกำลังประมวลผลฟังก์ชัน accept ซึ่งปรกตินิยมใช้จำนวนมากสุด คือ 5

2.2.4.2.5 accept

ในกระบวนการบริการแบบ connection-oriented จะเรียกใช้ฟังก์ชันนี้ ต่อจากฟังก์ชัน listen ทันที ขั้นตอนการรอการติดต่อจากกระบวนการลูกข่าย จะเกิดขึ้นจริงๆ หลังจากที่ได้ประมวลผลฟังก์ชันนี้แล้ว โดยคำสั่ง accept จะติดต่อกับการขอติดต่อที่อยู่ในคิว ด้วยการสร้าง socket ขึ้นอีก socket หนึ่ง ที่มีคุณสมบัติเหมือน8-bit fd ทุกประการ ซึ่งมีการใช้ดังนี้

```
#include <sys/types.h>
#include <sys/socket.h>
int     accept (int sockfd, struct sockaddr *peer, int *addrlen) ;
```

พารามิเตอร์ peer และ addrlen นั้น เป็นตัวแปรที่ส่งให้ฟังก์ชันเพื่อรับค่าที่อยู่ของกระบวนการที่ติดต่อเข้ามา และขนาดของโครงสร้างของที่อยู่ ตามลำดับ

2.2.4.2.6 close

เป็นฟังก์ชันที่ใช้สำหรับบอกการสิ้นสุดของการติดต่อ

```
int     close (int fd) ;
```

2.2.4.2.7 ฟังก์ชันเรียกใช้ในการลำดับไบต์ (byte ordering)

นอกจากฟังก์ชันที่กล่าวมาข้างต้นแล้ว ยังมีฟังก์ชันที่จำเป็นสำหรับการเรียกใช้ฟังก์ชัน ที่เกี่ยวกับโปรโตคอล อีกจำนวนหนึ่ง คือ

```
#include <sys/types.h>
#include <netinet/in.h>
u_long  htonl (u_long hostlong) ;
u_short  htons (u_short hostshort) ;
```

u_long ntohl (u_long netlong) ;

u_short ntohs (u_short netshort) ;

ซึ่งฟังก์ชันทั้ง 4 นี้เป็นการจัดลำดับของไบต์สูงและไบต์ต่ำ (byte ordering) ให้อยู่ในรูปแบบที่ต้องการ เช่น ฟังก์ชัน htonl เป็นการแปลงค่าตัวแปรชนิด u_long ในรูปแบบที่มีการเรียงลำดับไบต์ของ host ให้มีรูปแบบที่ใช้ในระบบเครือข่ายทั่วไป

2.2.4.2.8 boopy

มีรูปแบบการดังนี้คือ

boopy (char *sro, char *dest, int nbytes) ;

เป็นการสำเนาข้อมูลจำนวน nbytes จากตำแหน่งหน่วยความจำที่ชี้โดยตัวแปร sro ไปยังตำแหน่งหน่วยความจำที่ชี้โดยตัวแปร dest ซึ่งใน UNIX System V จะใช้ฟังก์ชัน memcopy แทน

2.2.4.2.9 bzero

มีรูปแบบการใช้ดังนี้คือ

bzero (char *dest, int nbytes) ;

เป็นการกำหนดค่า 0 ให้กับหน่วยความจำจำนวน nbytes เริ่มจากตำแหน่งหน่วยความจำที่ชี้โดยตัวแปร dest ฟังก์ชันนี้ใน UNIX System V จะใช้ฟังก์ชัน memset แทน

2.2.4.2.10 bcmp

มีรูปแบบการใช้ดังนี้คือ

int bcmp (char *ptr1, char *ptr2, int nbytes) ;

เป็นการเปรียบเทียบข้อมูลจำนวน nbytes ที่ชี้ตำแหน่งหน่วยความจำโดยตัวแปร ptr1 กับข้อมูลที่ชี้ตำแหน่งหน่วยความจำโดยตัวแปร ptr2 ฟังก์ชันนี้ใน UNIX System V จะใช้ฟังก์ชัน memcmp แทน

2.2.4.2.11 การแปลงรูปแบบที่อยู่ (Address Conversion)

นอกจากฟังก์ชันที่กล่าวมาข้างต้น ยังมีฟังก์ชันอีก 2 ฟังก์ชัน ที่ใช้สำหรับแปลงรูปแบบที่อยู่อินเทอร์เนทระหว่างรูปแบบที่เขียนในลักษณะทศนิยม 4 ตัว และรูปแบบเลขฐานสองขนาด 32 บิท ซึ่งได้แก่ฟังก์ชันต่อไปนี้

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
unsigned long     inet_addr (char *ptr) ;
char              * inet_ntoa (struct in_addr inaddr) ;
```

สำหรับตัวอย่างโปรแกรมที่ใช้โปรโตคอล TCP/IP ได้กล่าวไว้ในภาคผนวก ค ซึ่งเป็นการส่งข้อมูลจากกระบวนการลูกข่าย ไปยังกระบวนการบริการ แล้วส่งกลับมายังกระบวนการลูกข่ายแสดงผลออกทางจอภาพ

2.3 ทฤษฎีพีชคณิตสัมพันธ์

ภาษาพีชคณิตสัมพันธ์ ถูกนำเสนอโดย E. F. Codd เมื่อปี พ. ศ. 2515 [4,11] เป็นภาษาที่สามารถแสดงความสัมพันธ์ของข้อมูลที่สร้างขึ้นใหม่ ในรูปของความสัมพันธ์ของข้อมูลที่มีอยู่เดิม กับตัวปฏิบัติการ(Operation) ซึ่งตัวปฏิบัติการนี้มีทั้งหมด 8 ชนิดด้วยกัน โดยสามารถจะแบ่งตัวปฏิบัติการออกเป็น 2 กลุ่มได้ดังนี้

1.กลุ่มปฏิบัติการเบื้องต้น ได้แก่ UNION , INTERSECTION , DIFFERENCE , CARTESIAN PRODUCT

2 กลุ่มปฏิบัติการพิเศษ ได้แก่ SELECT , PROJECT , JOIN , DIVIDE

การทำงานของตัวปฏิบัติการ ทั้ง 8 ชนิด ได้แสดงไว้ในรูปที่ 2.14 ซึ่งอธิบายโดยสังเขปได้ดังต่อไปนี้

SELECT คือการสร้างความสัมพันธ์ จากการเลือกเอาข้อมูล TUPLES ที่มีคุณสมบัติตรงตามที่ระบุ จากความสัมพันธ์ที่มีอยู่แล้ว

PROJECT คือการสร้างความสัมพันธ์ จากความสัมพันธ์ที่ระบุถึง โดยเลือกเอาข้อมูลเฉพาะ ATTRIBUTE ที่กำหนด

PRODUCT คือการสร้างความสัมพันธ์ จากความสัมพันธ์สองความสัมพันธ์ ซึ่งความสัมพันธ์ที่สร้างขึ้นใหม่นี้ จะประกอบไปด้วยข้อมูลทั้งหมด ที่เกิดจากการนำข้อมูลใน TUPLES ของทั้งสองความสัมพันธ์มาจับคู่กัน

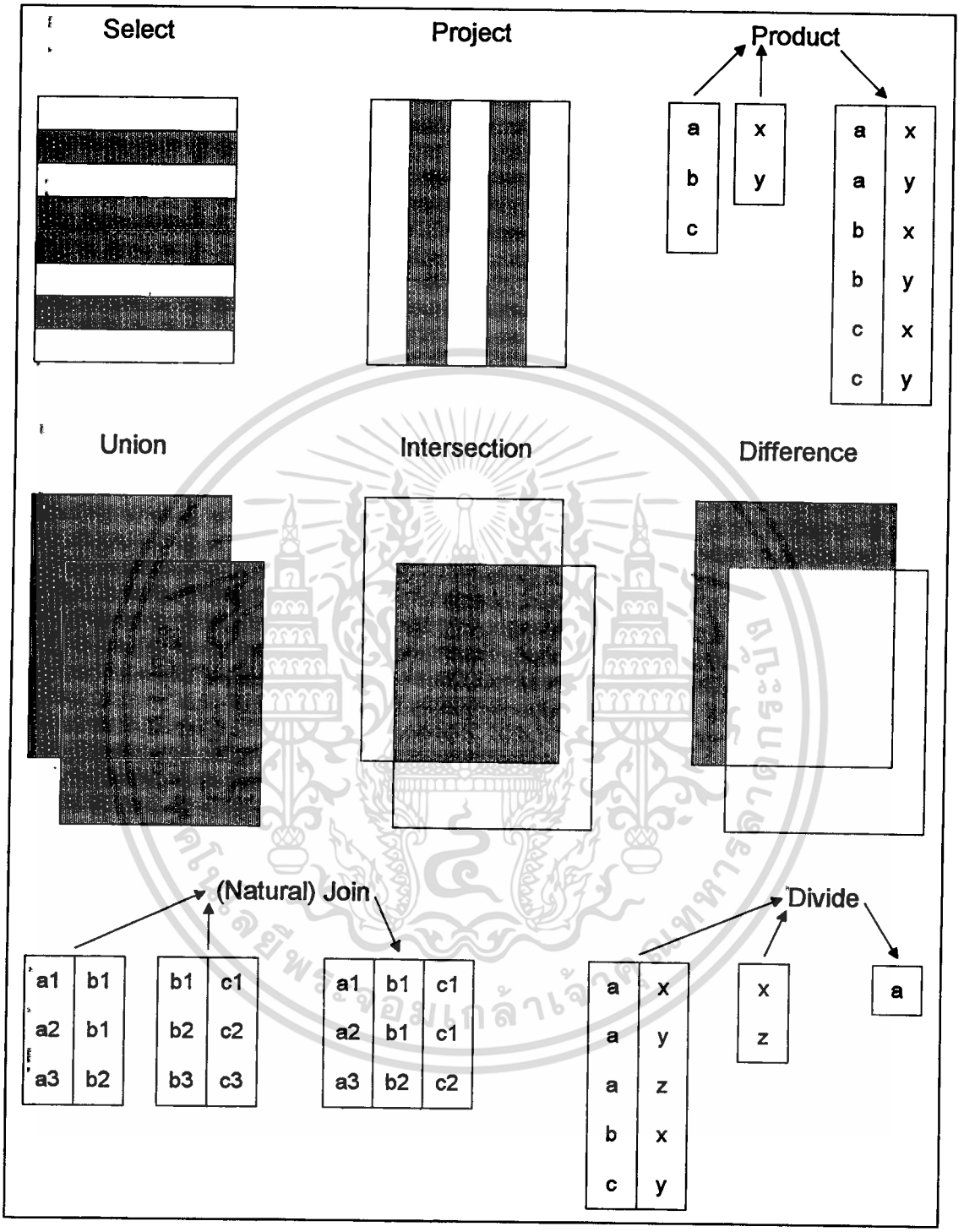
UNION คือการสร้างความสัมพันธ์ขึ้นใหม่ โดยการรวมข้อมูลของสองความสัมพันธ์ที่มีโดเมนเดียวกัน

INTERSECTION คือการสร้างความสัมพันธ์ จากความสัมพันธ์สองความสัมพันธ์ โดยข้อมูลของความสัมพันธ์ที่สร้างขึ้น จะประกอบไปด้วยข้อมูลที่อยู่ในทั้งสองความสัมพันธ์เดิม

DIFFERENCE คือการสร้างความสัมพันธ์ จากความสัมพันธ์สองความสัมพันธ์ โดยข้อมูลของความสัมพันธ์ที่สร้างขึ้นใหม่ ประกอบไปด้วยข้อมูลที่อยู่ในความสัมพันธ์แรกและไม่อยู่ในความสัมพันธ์ที่สอง

JOIN คือการสร้างความสัมพันธ์ จากความสัมพันธ์สองความสัมพันธ์ โดยข้อมูลของความสัมพันธ์ที่สร้างขึ้นใหม่ ประกอบไปด้วยข้อมูลที่เกิดจากการจับคู่ของ TUPLES ของความสัมพันธ์ทั้งสองที่ระบุ ภายใต้เงื่อนไขอย่างใดอย่างหนึ่ง

DIVIDE คือการสร้างความสัมพันธ์ขึ้นใหม่จากความสัมพันธ์ 2 ชนิด คือ ความสัมพันธ์ชนิด BINARY และความสัมพันธ์แบบ UNARY ซึ่งข้อมูลของความสัมพันธ์ที่สร้างขึ้นใหม่จะประกอบด้วย ข้อมูลของความสัมพันธ์แรก คือความสัมพันธ์ชนิด BINARY ใน ATTRIBUTE ที่ไม่ร่วมกัน (non-common) กับ ATTRIBUTE ของความสัมพันธ์ที่สอง คือความสัมพันธ์ชนิด UNARY โดยที่ข้อมูลใน ATTRIBUTE ที่ร่วมกันของความสัมพันธ์ชนิด BINARY จะเหมือนกับข้อมูลทั้งหมดในความสัมพันธ์แบบ UNARY



รูปที่ 2.14 แสดงการทำงานของตัวปฏิบัติการพีชคณิตสัมพันธ์

ซึ่งจะเห็นว่า ผลลัพธ์ของตัวปฏิบัติการพีชคณิตสัมพันธ์ทั้ง 8 ชนิดนี้ ยังคงเป็นความสัมพันธ์อยู่ นั่นคือ ตัวปฏิบัติการพีชคณิตสัมพันธ์ มีคุณสมบัติปิด ด้วยคุณสมบัติปิดนี้เองทำให้สามารถที่จะสร้างความสัมพันธ์ขึ้นใหม่ได้อีก จากความสัมพันธ์ที่มีอยู่และตัวปฏิบัติการทั้ง 8 ชนิด ซึ่งการแสดงความสัมพันธ์จะคล้ายกับพีชคณิตในคณิตศาสตร์ทั่วไป

2.3.1 ไวยากรณ์พีชคณิตสัมพันธ์

ในหัวข้อนี้ จะได้กล่าวถึงไวยากรณ์ของตัวปฏิบัติการพีชคณิตสัมพันธ์โดยละเอียด เพื่อจะได้เป็นพื้นฐานในการทำความเข้าใจในบทต่อๆไป

ไวยากรณ์ของตัวปฏิบัติการพื้นฐาน ได้ถูกกำหนดไว้ในรูปของ Backus Nauer Form (BNF) ดังนี้แสดงไว้ในรูปที่ 2.15

จากหลักการไวยากรณ์ของตัวปฏิบัติการพีชคณิตสัมพันธ์ ในรูปของ BNF นั้น ส่วนของวงเล็บก้ามปูเป็นส่วนที่ไม่สามารถที่จะละทิ้งได้ เพราะเป็นส่วนหนึ่งของไวยากรณ์

```
rel-def     ::=  DEFINE RELATION rel-name [attr-name-commalist]
alias-defn  ::=  DEFINE ALIAS rel-name FOR rel-name
expr        ::=  select | projection | infix-expr
selection   ::=  primitive WHERE selection-pred
primitive   ::=  rel-name | (expr)
projection  ::=  primitive | primitive [attr-spec-commalist]
attr-spec   ::=  attr-name | rel-name.attr-name
infix-expr  ::=  projection infix-op projection
infix-op    ::=  UNION | INTERSECT | MINUS | TIMES | JOIN | DIVIDEBY
```

รูปที่ 2.15 แสดงไวยากรณ์พีชคณิตสัมพันธ์ในรูปของ BNF

2.3.1.1 ข้อสังเกตเกี่ยวกับไวยากรณ์

2.3.1.1.1 ถ้า "xyz" คือกลุ่มของชื่อ attribute ดังนั้น "xyz-commalist" จะแทนกลุ่มของชื่อ attribute ที่ประกอบด้วยชื่อของ attribute ภายในกลุ่ม ตั้งแต่หนึ่งชื่อขึ้นไป เขียนเรียงตาม "xyz" และแต่ละชื่อจะถูกแยกออกจากกันด้วยเครื่องหมาย comma (,)

2.3.1.1.2 การกำหนดชื่อ rel-name และ attr-name จะต้องกำหนดให้แต่ละชื่อเป็นเอกเทศ (identifหมาย) ไม่ซ้ำกัน

2.3.1.1.3 selection-pred ใช้แสดงแทนรูปแบบการเปรียบเทียบ ในลักษณะของฟังก์ชันบูลีน บางครั้งอาจจะเป็นการเปรียบเทียบอย่างง่ายๆ ระหว่างค่าของ attribute กับ attribute หรือ ค่าของ attribute กับค่าคงที่

2.3.1.1.4 ใน projection จะไม่อนุญาตให้ attr-spec เขียนอยู่ในรูปของ expression อย่างเช่น ATTR1+ATTR2 และในทำนองเดียวกัน ในการเปรียบเทียบของฟังก์ชันบูลีนจะไม่อนุญาตให้เขียนอยู่ในรูปของ expression เช่นเดียวกัน

2.3.2 ตัวปฏิบัติการเบื้องต้น

ตัวปฏิบัติการกลุ่มเบื้องต้นประกอบด้วย union , intersection , difference และ production ซึ่งตัวปฏิบัติการแต่ละตัว จะใช้ความสัมพันธ์สองความสัมพันธ์ในการปฏิบัติการ ความสัมพันธ์ที่ใช้กับตัวปฏิบัติการ .

ดังกล่าว (ยกเว้น cartesian product) จะต้อง union-compatible กัน หรือกล่าวอีกนัยหนึ่ง คือ ความสัมพันธ์ทั้งสองจะต้องมี degree (จำนวนแอททริบิว) เท่ากัน เช่น degree = n และ attribute ที่ i (i = 1,2,3, ... ,n) ของแต่ละความสัมพันธ์ ควรจะมีโดเมนเดียวกัน จากกฎของ union-compatible จะทำให้ผลลัพธ์ที่ได้ยังคงอยู่ในรูปของความสัมพันธ์ หากความสัมพันธ์ที่ใช้กับตัวปฏิบัติการดังกล่าว ไม่เป็นไปตามกฎ union-compatible ผลลัพธ์ที่ได้จะไม่เป็นความสัมพันธ์ แต่จะอยู่ในรูปเซต

2.3.2.1 UNION

ให้ A และ B เป็นความสัมพันธ์ที่ union-compatible มี degree เป็น n ดังนั้นจะกล่าวได้ว่า A UNION B คือความสัมพันธ์ที่มี degree เท่ากับ n และ tuples ทั้งหมดของผลลัพธ์ที่ได้ จะมาจาก tuples ของความสัมพันธ์ A หรือ B

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	paris
P6	Cog	Red	19	London

J

J#	JNAME	CITY
J1	Sorter	Paris
J2	Punch	Rome
J3	Reader	Athens
J4	Console	Athens
J5	Collator	London
J6	Terminal	Oslo
J7	Tape	London

SPJ

S#	P#	J#	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

รูปที่ 2.16 แสดงตารางข้อมูลตัวอย่าง

ตัวอย่างที่ 2.4

จากตารางในรูปที่ 2.16 ให้ A เป็นเซตของ tuples ของ supplier ที่อาศัยอยู่ใน LONDON และ ให้ B เป็นเซตของ tuples ของ supplier ที่ขาย part P1 ดังนั้น A UNION B คือ เซตของ tuples ของ supplier ที่อาศัยอยู่ใน LONDON หรือ ขาย part P1 (หรือทั้งสองอย่าง) และ ผลลัพธ์ที่ได้จากตัวอย่างข้างต้นจะมี attribute qualified

name (ชื่อของแอททริบิวที่มีการอ้างถึงชื่อความสัมพันธ์ เช่น R.a) เช่นเดียวกับ ความสัมพันธ์แรก คือ ความสัมพันธ์ A

จากตัวอย่างข้างต้น ถ้าเราสลับที่ระหว่าง A และ B ผลลัพธ์ที่ได้ก็ยังคงเหมือนเดิม ดังนั้นจะได้ว่า UNION มีคุณสมบัติการสลับที่ และจากคุณสมบัติปิด ทำให้ UNION มีคุณสมบัติการจัดหมู่ด้วย กล่าวคือ (X UNION Y) UNION Z และ X UNION (Y UNION Z) จะให้ผลลัพธ์ที่เหมือนกัน ดังนั้นเพื่อความสะดวกจึงไม่นิยมเขียนใส่วงเล็บ เช่น X UNION Y UNION Z

2.3.2.2 INTERSECTION

เราจะกล่าวได้ว่า INTERSECTION ของความสัมพันธ์ A และ B ที่ union-compatible และมี degree เป็น n เขียนแทนด้วย A INTERSECT B คือ ความสัมพันธ์ที่มี degree เป็น n และ ประกอบด้วย tuples ที่เป็นของความสัมพันธ์ A และเป็นของความสัมพันธ์ B

ตัวอย่างที่ 2.5

Pri nร์ ย่างที่ 2.4 เราจะกล่าวได้ว่า A INTERSECT B คือเซตของ tuples ของ supplier ที่อาศัยใน LONDON และ ขาย part P1

จากตัวอย่างที่ 2.5 ผลลัพธ์ที่ได้ เหมือนกับผลลัพธ์ที่ได้จาก B INTERSECT A ดังนั้น INTERSECTION มีคุณสมบัติการสลับที่ และจากเหตุผลเดียวกันกับตัวปฏิบัติการ UNION จะได้ว่า INTERSECTION มีคุณสมบัติการจัดหมู่ด้วย

2.3.2.3 DIFFERENCE

การกระทำการ difference ของความสัมพันธ์ A และ B ที่ union-compatible และมี degree เท่ากับ n เขียนแทนด้วย A MINUS B ผลลัพธ์ที่ได้จะอยู่ในรูปของความสัมพันธ์ ที่มี degree เท่ากับ n และประกอบด้วย tuples ที่เป็น tuples ของความสัมพันธ์ A แต่ไม่เป็น tuples ของความสัมพันธ์ B

ตัวอย่างที่ 2.6

จากตัวอย่างที่ 2.4 ถ้า A MINUS B จะได้ผลลัพธ์เป็นเซตของ tuples ของ supplier ที่อาศัยอยู่ใน LONDON แต่ไม่ขาย part P1

จะเห็นได้ว่า DIFFERENCE ไม่มีคุณสมบัติการสลับที่ ดังนั้น DIFFERENCE จึงไม่มีคุณสมบัติของการจัดหมู่ไปด้วย

2.3.2.4 EXTENDED CARTESIAN PRODUCT

ผลลัพธ์ของตัวปฏิบัติการ EXTENDED CARTESIAN PRODUCT ของความสัมพันธ์ A และ ความสัมพันธ์ B ที่มี degree เป็น m และ n ตามลำดับ จะเขียนแทนด้วย A TIMES B ผลลัพธ์ที่ได้จะมี degree เท่ากับ m+n และประกอบไปด้วยเซตของ tuples t ทั้งหมดที่เกิดจากการนำทุก tuple a = $(a_1,a_2, \ldots ,a_m)$ ของความสัมพันธ์ A มาเขียนต่อด้วยทุก tuple b = $(b_1,b_2, \ldots ,b_n)$ ของความสัมพันธ์ B จะได้ tuple t = $(a_1,$

พันธ์ A มาเขียนต่อด้วยทุก tuple b = (b_1,b_2, ... ,b_n) ของความสัมพันธ์ B จะได้ tuple t = (a_1, ...,a_m,$b_{(m+1)}$, ... ,$b_{(m+n)}$)

ผลลัพธ์ที่ได้จากตัวปฏิบัติการ TIMES จะกำหนด qualified name โดย qualified name ของ A และ B เรียงลำดับจากซ้ายไปขวาดังนี้

A.a_1, ... ,A.a_m,B.$b_{(m+1)}$, ... ,B.$b_{(m+n)}$)

ตัวอย่างที่ 2.7

ให้ A เป็นเซตของ supplier และ B เป็นเซตของ part ดังนั้น A TIMES B จะเป็นเซตของคู่ลำดับ supplier/part

จากการที่ลำดับก่อนหลังของ attribute ในความสัมพันธ์ในระบบฐานข้อมูลแบบสัมพันธ์ ไม่มีความสำคัญใดๆ ดังนั้นจะได้ว่า A TIMES B และ B TIMES A ให้ผลลัพธ์ที่เหมือนกัน ซึ่งสรุปได้ว่า TIMES มีคุณสมบัติการสลับที่ และทำนองเดียวกันกับตัวปฏิบัติการ UNION จะได้ว่า TIMES มีคุณสมบัติการจัดหมู่ด้วย ซึ่งเขียนได้ดังนี้ (A TIMES B) TIMES C เท่ากับ A TIMES (B TIMES C)

2.3.3 ตัวปฏิบัติการความสัมพันธ์พิเศษ

สำหรับตัวปฏิบัติการที่จัดอยู่ในกลุ่มนี้ได้แก่ seleotion , projeotion , join และ divide ซึ่งมีรายละเอียดดังนี้

2.3.3.1 SELECTION

ให้ R เป็นความสัมพันธ์ ที่มี degree เท่ากับ n และให้ θ แทนการการเปรียบเทียบของตัวปฏิบัติการ (เช่น = , > , < ,<= เป็นต้น) θ-seleotion ของความสัมพันธ์ R ที่ใช้การเปรียบเทียบ attribute X และ Y เขียนแทนได้ดังนี้

R WHERE R.X θ R.Y

ผลลัพธ์ที่ได้จะเป็นกลุ่มของ tuples t ของความสัมพันธ์ R ที่การเปรียบเทียบ "R.X θ R.Y" เป็นจริง (attribute X และ Y ควรจะอยู่ในโดเมนเดียวกัน และ การเปรียบเทียบ θ จะต้องสมเหตุสมผลกับข้อมูลในโดเมนนั้นด้วย) R.Y อาจจะแทนด้วยค่าคงที่ก็ได้ดังนี้

R WHERE R.X θ o

เมื่อ o เป็นค่าคงที่

θ-seleotion จะให้ผลลัพธ์ที่เป็นบางส่วนในแนวนอน (horizontal) ของความสัมพันธ์ที่ปฏิบัติการกับตัวปฏิบัติการ seleotion และ การเปรียบเทียบสามารถที่จะเขียนในรูปของการเปรียบเทียบ ที่เชื่อมต่อด้วยตัวปฏิบัติการของฟังก์ชันบูลีน (Boolean) อันได้แก่ AND , OR และ NOT เช่น

ตัวอย่างที่ 2.8

R WHERE p_1 AND p_2 ซึ่งจะเหมือนกับการกำหนด

(R WHERE p_1) INTERSECT (R WHERE p_2)

ตัวอย่างที่ 2.9

R WHERE p_1 OR p_2 ซึ่งจะเหมือนกับการกำหนด

(R WHERE p_1) UNION (R WHERE p_2)

ตัวอย่างที่ 2.10

R WHERE NOT p ซึ่งจะเหมือนกับการกำหนด

R MINUS (R WHERE p)

2.3.3.2 PROJECTION

ให้ R เป็นความสัมพันธ์ที่มี degree เท่ากับ n และประกอบด้วย attribute a_1, ... ,a_n ดังนั้นการปฏิบัติการ projeotion กับความสัมพันธ์ R ใน attribute a_i, ... ,a_{ik} เมื่อ $k < n$ เขียนแทนได้ด้วย

R [ai, ... ,aik]

และผลลัพธ์ที่ได้คือกลุ่มของ tuples (ai, ... ,aik) ที่ tuple t เป็นสมาชิกของความสัมพันธ์ R ซึ่งเป็นการกำหนด บางส่วนในแนวตั้ง (vertioal) ของความสัมพันธ์ R นั่นเอง เช่น

ตัวอย่างที่ 2.11

S [CITY]

เป็นความสัมพันธ์ที่เกิดจากการเลือกค่าที่ไม่ซ้ำกันใน attribute CITY ของความสัมพันธ์ R และ attribute ที่ได้จะมี qualified name เหมือนกับความสัมพันธ์ R

2.3.3.3 JOIN

ให้ R_1 และ R_2 เป็นความสัมพันธ์ที่มี degree เท่ากับ m และ n ตามลำดับ และ θ เป็นตัวเปรียบเทียบทางคณิตศาสตร์ เหมือนกับในหัวข้อ 2.4.1 เราจะกล่าวได้ว่า θ-JOIN ของความสัมพันธ์ R_1 ด้วย attribute a กับความสัมพันธ์ R_2 ด้วย attribute b คือกลุ่มของ tuples t ที่เกิดจากการนำ tuples t_1 ในความสัมพันธ์ R_1 มาต่อท้ายด้วย tuples t_2 ในความสัมพันธ์ R_2 ภายใต้เงื่อนไข " $R_1.a$ θ $R_2.b$ " ที่เป็นจริง (a และ b ควรจะอยู่ในโดเมนเดียวกัน)

ตัวอย่างที่ 2.12

greater-than join ของความสัมพันธ์ S ด้วย attribute CITY กับความสัมพันธ์ P ด้วย attribute CITY

อย่างไรก็ตาม θ-JOIN มิใช่ตัวปฏิบัติการโดยแท้จริง นั่นคือเราสามารถที่จะเขียน θ-JOIN ให้อยู่ในรูปของ extended oartesian produot ของสองความสัมพันธ์ แล้วทำการเลือกเอา tuples ที่เหมาะสมด้วยตัวปฏิบัติการ seleotion ดังแสดงในตัวอย่างที่ 2.13

ตัวอย่างที่ 2.13

จากตัวอย่างที่ 2.12 เราสามารถที่จะเขียนในรูปของตัวปฏิบัติการที่แท้จริงได้เป็น

(S JOIN P) WHERE S.CITY > P.CITY

ถ้า θ แสดงความเท่ากัน เราจะเรียกการ JOIN นี้ว่า equi-join และการปฏิบัติการของ equi-join ที่ไม่ได้กำหนด attribute ในการกระทำการ จะเรียกการ join นั้นว่า natural-join ซึ่งในครั้งต่อๆไป เมื่อกล่าวถึงการปฏิบัติการ join แล้ว จะหมายถึง natural-join และจะเขียนแทนได้ด้วย JOIN เช่น S JOIN SP

ซึ่งโดยปรกติ การเขียนในลักษณะข้างต้นจะถูกนิยามก็ต่อเมื่อ unqualified name ของ attribute ของทั้งสองความสัมพันธ์ ที่กระทำการ จะต้องอยู่ภายใต้โดเมนเดียวกัน

จากที่กล่าวมาจะเห็นว่า join มีคุณสมบัติการสลับที่ นั่นคือ A JOIN B จะเท่ากับ B JOIN A และ JOIN ยังมีคุณสมบัติในการจัดหมู่ด้วย คือ (A JOIN B) JOIN C จะให้ผลลัพธ์ เหมือนกับ A JOIN (B JOIN C) ดังนั้น จึงมีผู้นิยมเขียนเป็น A JOIN B JOIN C

2.3.3.4 DIVISION

ให้ A เป็นความสัมพันธ์ที่มี degree เท่ากับ m+n และ B เป็นความสัมพันธ์ที่มี degree เท่ากับ n แล้วจะกล่าวได้ว่า ผลลัพธ์ของการ division ความสัมพันธ์ A ด้วยความสัมพันธ์ B เขียนแทนด้วย A DIVIDEBY B คือ ความสัมพันธ์ ที่มี degree เท่ากับ m โดยที่ความสัมพันธ์ A มีข้อมูลของ tuples ใน attributes ที่ m+1 ถึง m+n เหมือนกับข้อมูล ในทุก tuples ของความสัมพันธ์ B ในขณะที่ข้อมูลของ tuples เหล่านั้นใน attributes ที่ 1 ถึง m ของความสัมพันธ์ A มีค่าเพียงค่าเดียว

จากนิยามข้างต้น ข้อมูลใน attribute ที่ m+i ของความสัมพันธ์ A และข้อมูลใน attribute ที่ i ของความสัมพันธ์ B (เมื่อ i = 1, 2, ... ,n) ควรจะอยู่ในโดเมนเดียวกัน และ unqualified name ของผลที่ได้จะเหมือนกับ m attributes แรกของความสัมพันธ์ A

อย่างไรก็ตาม DIVIDEBY ไม่ได้เป็นตัวปฏิบัติการโดยแท้จริง ซึ่งสามารถที่จะเขียนให้อยู่ในรูปของตัวปฏิบัติการที่แท้จริงได้ เช่น

A DIVIDEBY B = A [X] MINUS ((A [X] TIMES B) MINUS A) [X]

เมื่อ X แทน attribute ของความสัมพันธ์ Aที่ไม่ร่วมกันกับ attribute ของความสัมพันธ์ B

ตัวอย่างที่ 2.14

จากตารางข้อมูลตัวอย่างนั้น ถ้าต้องการการหาว่าผู้ขายสินค้าหมายเลขใด ขายสินค้าทุกชนิด จะเขียนเป็นคำถามในภาษาพีชคณิตสัมพันธ์ได้ดังนี้

SP [S#, P#] DIVIDEBY P[P#]

ซึ่งผลลัพธ์ที่ได้คือ S5

2.3.4 ตัวอย่างการใช้งานพีชคณิตสัมพันธ์

สำหรับตัวอย่างการใช้งานในแต่ละตัวปฏิบัติการนั้น ได้กล่าวไว้ใน หัวข้อ 2.3.3 แล้ว ในหัวข้อนี้จะเป็นการยกตัวอย่างการใช้งานพีชคณิตสัมพันธ์ ซึ่งในการใช้งานจริงนั้น อาจจะมีหลายตัวปฏิบัติการในหนึ่งคำถาม โดยไม่อาจกำหนดแน่นอนได้ว่าต้องใช้ตัวปฏิบัติการเท่าใด และในหนึ่งคำถามนั้นอาจเขียนเป็นภาษาพีชคณิตสัมพันธ์ ได้มากกว่าหนึ่งแบบ ก็ได้

ดังนั้นในหัวข้อนี้จะได้ยกตัวอย่างเป็นแนวทางในการใช้งาน โดยใช้ข้อมูลจากตารางข้อมูลตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 2.15

จงหาชื่อผู้ขายสินค้า (supplier) ที่ขายสินค้า (part) P2

((S JOIN SP) WHERE P# = 'P2') [SNAME]

ตัวอย่างที่ 2.16

จงหาชื่อผู้ขายสินค้า ที่ขายสินค้าสีแดงอย่างน้อยหนึ่งชิ้น

(((P WHERE COLOR = 'Red') [P#] JOIN SP) [S#] JOIN S) [SNAME]

ตัวอย่างที่ 2.17

จงหาชื่อผู้ขายสินค้าที่ขายสินค้าทุกชนิด

((SP [S#, P#] DIVIDEBY P [P#]) JOIN S) [SNAME]

ตัวอย่างที่ 2.18

จงหาหมายเลขของผู้ขายสินค้า ซึ่งสินค้าทั้งหมดที่ขายไม่น้อยกว่าสินค้าที่ขายโดยผู้ขายหมายเลข S2

SP [S#, P#] DIVIDEBY (SP WHERE S# = 'S2') [P#]

ตัวอย่างที่ 2.19

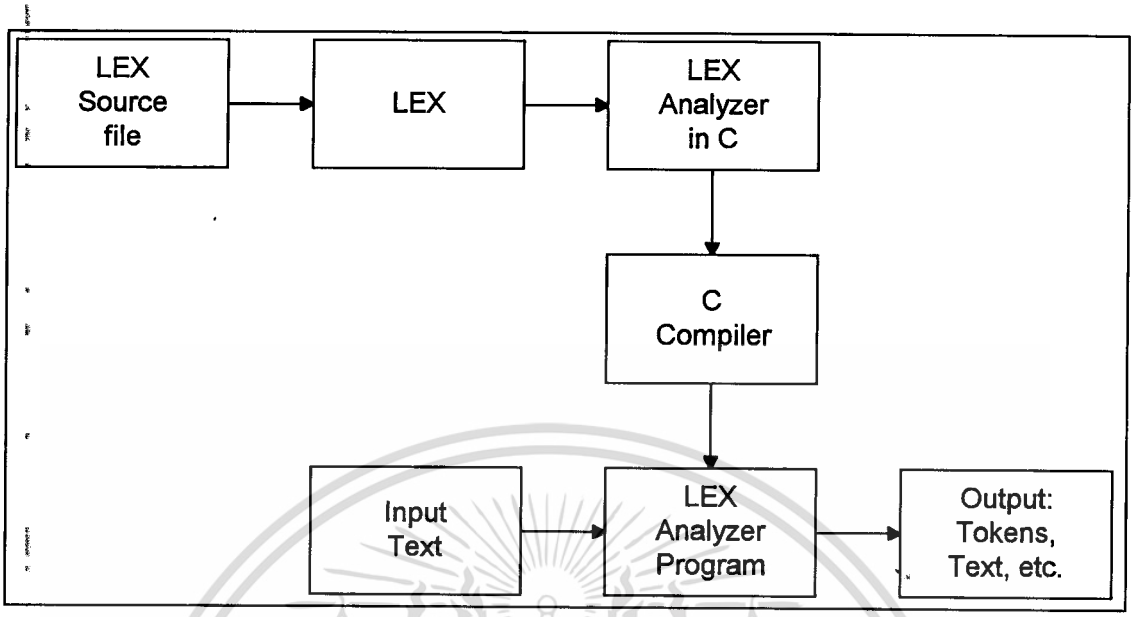
จงหาชื่อของผู้ขายสินค้า ที่ไม่ได้ขายสินค้าหมายเลข P2

((S [S#] MINUS (SP WHERE P# = 'P2') [S#]) JOIN S) [SNAME]

2.4 การพัฒนาโปรแกรมด้วย LEX และ YACC

2.4.1 การพัฒนาโปรแกรมด้วย LEX

LEX (LEXICAL ANALYZER) เป็นเครื่องมือที่ใช้พัฒนางานในด้านต่างๆ เช่น การประมวลผลอักษร การเข้ารหัส การเขียนตัวแปลภาษา เป็นต้น ซึ่งลักษณะร่วมกันของงานเหล่านี้ได้แก่การตรวจแยกคำและทำงานเมื่อตรวจพบคำที่กำหนดไว้ ขั้นตอนการพัฒนาโปรแกรมด้วย LEX ได้แสดงไว้ในรูปที่ 2.17 สำหรับวิธีการคอมไพล์โปรแกรม สามารถดูได้ในภาคผนวก ค



รูปที่ 2.17 แสดงขั้นตอนการพัฒนาโปรแกรมด้วยเครื่องมือซอฟท์แวร์ LEX

ในการเขียนโปรแกรม LEX โดยปรกติแล้ว จะประกอบด้วย 3 ส่วนด้วยกัน คือ

1.ส่วนนิยาม (Definition)

2.ส่วนกฎ (Rules)

3.ส่วนยูสเซอร์ซับรูทีน (User Subroutines)

โดยจะมีส่วนกฎจะเท่านั้นที่จำเป็นต้องมี แต่ส่วนนิยามและส่วนยูสเซอร์ซับรูทีนไม่จำเป็นต้องมีก็ได้ โดยในแต่ละส่วนสามารถอธิบายได้ดังนี้

2.4.1.1 ส่วนนิยาม

ส่วนนี้จะถูกประกาศไว้ก่อนเครื่องหมาย %% ส่วนประกอบที่สำคัญในส่วนนิยาม ได้แก่การอ้างนิยามจากภายนอก (ขึ้นต้นด้วยคำว่า extern) และส่วน preprocessor เช่น #include, #define โดยที่นิพจน์เหล่านี้จะอยู่ระหว่างเครื่องหมาย %{ และ %} หลังจากส่วนนี้จะเป็นคำย่อต่างๆ ที่ใช้แทนรูปแบบของตัวอักษรซึ่งสามารถนำไปอ้างถึงในส่วนกฎได้ คำย่อเหล่านี้จะช่วยลดความยุ่งยากซับซ้อนที่ไม่จำเป็นในการเขียนกฎได้ โดยคำย่อและความหมายของมันจะต้องมีช่องว่าง (space) คั่นอย่างน้อย 1 ตัว

ตัวอย่าง 2.20

```
%{
        #include <stdio.h>
        #include "y.tab.h"
        extern  int     tokval ;
        int     lineno ;
%}
D       [0-9]
```

```
L       [a-zA-Z]
W       [ \t]+
%%
        •
        •
```

2.4.1.2 ส่วนกฎ

ส่วนกฎเป็นส่วนที่ใช้ในการตรวจสอบคำทั้งหมด ซึ่งจะอยู่ระหว่างเครื่องหมาย %% โดยกฎแต่ละกฎจะประกอบด้วย 2 ส่วนคือ ส่วนที่เป็นการบรรยายถึงรูปแบบของโทคเคน(คำที่ต้องการจะทำการตรวจสอบ) ซึ่งประกอบด้วยกลุ่มของตัวอักษรและตัวปฏิบัติการ (operator) เรียกว่า นิพจน์สามัญ (regular expression) ชุดของตัวปฏิบัติการได้แสดงไว้ในรูปที่ 2.18 และ ส่วนที่ทำให้เกิดการทำงานเมื่อพบรูปแบบดังกล่าว ซึ่งส่วนนี้จะเป็นนิพจน์ของภาษาซี 1 นิพจน์ แต่ถ้าต้องการเขียนหลายนิพจน์จะต้องทำให้เป็นนิพจน์เชิงรวม (compound statement) โดยนิพจน์เหล่านี้จะอยู่ภายในเครื่องหมายปีกกา (braces)

นิพจน์สามัญ	ความหมาย
\x	x, ถ้า x เป็นตัวปฏิบัติการใน lex
"xy"	xy, โดยที่ x หรือ y อาจเป็นตัวปฏิบัติการใน lex ได้ (ยกเว้น \)
[xy]	x หรือ y
[x-z]	x หรือ y หรือ z
[^x]	ตัวอักษรใดๆ ยกเว้น x
.	ตัวอักษรใดๆ ยกเว้นตัวขึ้นบรรทัดใหม่ (newline)
^x	x ที่ต้นบรรทัด
<y>x	x เมื่อ lex อยู่ในเงื่อนไขเริ่มต้น y
x$	x ที่ท้ายบรรทัด
x?	x เลือกได้
x*	x ตั้งแต่ศูนย์ตัวขึ้นไป
x+	x อย่างน้อยหนึ่งตัวขึ้นไป
x{m,n}	x ตั้งแต่ m ถึง n ตัว
xx|yy	xx หรือ yy อย่างใดอย่างหนึ่ง
x |	การทำงานของกฎนี้ จะมาจากการทำงานของกฎถัดไป
(x)	x
x/y	x แต่ต้องตามด้วย y
{xx}	คำย่อ xx จากในส่วนนิยาม

รูป 2.18 ตัวปฏิบัติการที่ใช้ใน lex

ตัวอย่าง 2.21

```
%%
        -[0-9]+         printf ("negative value") ;
        +?[0-9]+        printf ("positive integer") ;
        -0.{D}+         printf ("negative fraction,no whole number part") ;
        rail[ ]+road    printf ("railroad is one word") ;
        crook           printf ("Here's a crook") ;
        function        subprogcount++ ;
        G[a-zA-Z]*      {
                            printf ("may have a G word here:",yytext) ;
                            Gstringcount++ ;
                        }
%%
```

2.4.1.3 ส่วนยูสเซอร์สับรูทีน (User Subroutine)

ส่วนยูสเซอร์สับรูทีนนี้จะเป็นส่วนของฟังก์ชันต่างๆที่ผู้เขียนโปรแกรมเขียนขึ้น เพื่อให้ทำงานอย่างใดอย่างหนึ่งเมื่อตรวจสอบพบคำที่กำหนดไว้ โดยจะเรียกใช้ได้จากส่วนกฎ ในการเขียนฟังก์ชันต่างๆเหล่านี้จะเขียนไว้ต่อจากส่วนกฎ ซึ่งในส่วนนี้ผู้เขียนโปรแกรมสามารถที่จะเขียนฟังก์ชันเหมือนกับฟังก์ชันในภาษาซีได้ตามปกติ

ตัวอย่าง 2.22

```
%%
.
.
"/*"                    skipcmnts ( ) ;
.
.               /* rest of rules */
%%
skipcmnts ( )
{
    for ( ; ; )
```

```
        {
                while (input () != '*') ;
                if (input () != \')
                        unput (yytext[yylen-1]) ;
                else
                        return ;
        }
}
```

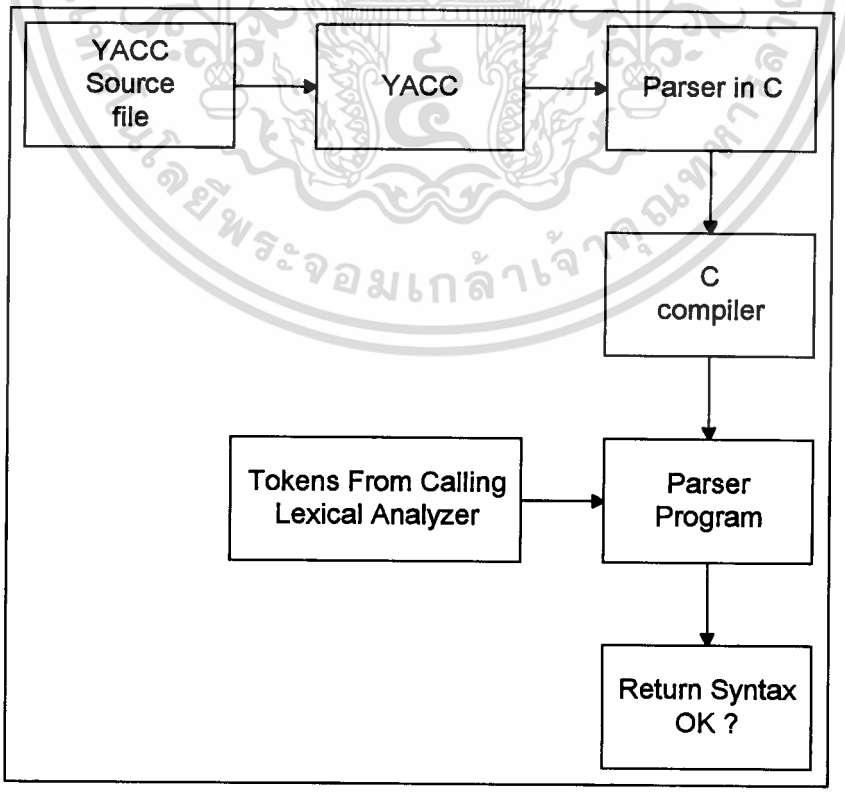
2.4.2 การพัฒนาโปรแกรมด้วย YACC

YACC เป็นเครื่องมือที่ใช้ในการวิเคราะห์ไวยากรณ์ (syntax) ของภาษา ซึ่งโดยปรกติแล้วจะใช้งานร่วมกับ LEX ดังรูปที่ 2.19 แสดงขั้นตอนการใช้งาน YACC

ส่วนประกอบของโปรแกรม YACC จะประกอบไปด้วยส่วนต่างๆ 4 ส่วนด้วยกันคือ

2.4.2.1 ส่วนนิยาม (Declarations)

ส่วนนิยามในโปรแกรม YACC นี้จะเหมือนกับส่วนนิยามในโปรแกรม LEX คือ จะใช้ในการประกาศตัวแปรที่ใช้เก็บค่าต่างๆ ทั้งตัวแปรภายใน และตัวแปรภายนอก พร้อมทั้งส่วนนิพจน์ #include ด้วย ซึ่งส่วนนี้จะอยู่ประกาศไว้ระหว่างเครื่องหมาย %{ และ %}



รูปที่ 2.19 แสดงการใช้งานเครื่องมือซอฟท์แวร์ YACC

2.4.2.2 ส่วนประกาศสัญกรณ์ (Declare Token)

ส่วนนี้จะอยู่ระหว่างส่วนนิยามและส่วนไวยากรณ์ ซึ่งจะใช้ในการประกาศว่าในไวยากรณ์มีการใช้สัญกรณ์ที่เป็นแบบเทอร์มินัล (สัญกรณ์ที่ไม่อยู่ทางด้านขวาของกฎใดๆ) อะไรบ้าง และจะเริ่มตรวจสอบด้วยไวยากรณ์ใด โดยในการประกาศแต่ละบรรทัดจะต้องนำหน้าด้วยเครื่องหมาย %

ตัวอย่าง 2.23

```
%token  CREATE , DELETE
%token WHERE
%start   begin
```

2.4.2.3 ส่วนไวยากรณ์ (Grammar rules)

ในส่วนนี้จะประกอบด้วยส่วนไวยากรณ์ที่อยู่ในรูปสัญกรณ์บีเอ็นเอฟ (BNF) และส่วนที่จะให้เกิดการทำงาน เมื่อสัญกรณ์ที่เข้ามาตรงกับไวยากรณ์นั้นๆ ซึ่งไวยากรณ์ทั้งหมดจะอยู่ระหว่างเครื่องหมาย %% และ %%

ตัวอย่าง 2.24

```
begin     :create
{
ins_constr(constr_name,fst_attr,sec_attr,thd_attr,opt1,val1,opt2,val2) ;
return( 1 ) ;
}
| delete
{
del_constr (constr_name) ;
return (1) ;
}
| EXIT
{
return (0) ;
}
;
```

ในส่วนของการทำงานด้านขวามือ เราสามารถอ้างถึงค่าของสัญกรณ์ที่ผ่านเข้ามาโดยผู้ใช้ได้ โดยใช้กลไกของ $n โดยที่ค่า n แทนลำดับที่ของ token ทางด้านขวามือของกฎ และ $$ จะใช้แทน non-terminal ทางด้านซ้ายมือของกฎ

ตัวอย่าง 2.25

กฎต่อไปนี้จะทำให้ x มีค่าเป็น 1 และ y มีค่าเป็นของ token ที่คืนกลับมาด้วย C

```
A       :       B
                  {
                    $$ = 1;
                  }
                  C
            {
                  x = $2;
                  y = $3;
            }
```

สิ่งที่น่าสนใจคือ เราจะต้องมองนิพจน์ (ในที่นี้อยู่ระหว่างสัญลักษณ์ B กับ C) เป็นสัญลักษณ์อีกตัวหนึ่งด้วย ดังนั้น x หรือ $2 จะเป็นค่าของนิพจน์ที่อยู่ระหว่าง B กับ C ซึ่งก็คือ 1 นั่นเอง

2.4.2.4 ส่วนยูสเซอร์สับรูทีน (User Subroutine)

ส่วนนี้จะประกอบด้วยฟังก์ชันต่างๆ ที่จะถูกเรียกใช้โดยส่วนไวยากรณ์ของ yacc ซึ่งจะเหมือนกันกับส่วนยูสเซอร์สับรูทีนในโปรแกรมที่ใช้กับ LEX โดยจะอยู่ต่อจากส่วนไวยากรณ์

จากที่ได้กล่าวมาข้างต้น พอจะสรุปโครงสร้างของโปรแกรมที่ใช้กับ yacc ได้ดังนี้

```
%{
        declarations
%}
%token ...
%start ...
%%
        grammar rules
%%
        subroutines
```

บทที่ 3
การส่งคำถามภาษา SQL
ไปประมวลผลที่ระบบคอมพิวเตอร์อื่น
(Remote Query Processing)

3.1 รูปแบบของคำถาม

การส่งคำถามไปประมวลผลที่ระบบคอมพิวเตอร์อื่น เป็นอีกวิธีที่จะอ้างถึงข้อมูลที่อยู่ต่างระบบคอมพิวเตอร์ได้ โดยผลลัพธ์ที่ได้จะถูกส่งกลับมายังระบบที่ส่งคำถาม คำถามที่ถูกส่งไปนี้ สามารถแบ่งได้เป็น 2 ลักษณะดังนี้ คือ

1. คำถามที่ถูกกำหนดล่วงหน้า
2. คำถามที่ไม่ถูกกำหนดล่วงหน้า

3.1.1 คำถามที่ถูกกำหนดล่วงหน้า

คำถามลักษณะนี้จะใช้กับคำถามที่ถูกใช้ซ้ำๆ เพราะว่าคำถามจะถูกกำหนดไว้ล่วงหน้า ไม่สามารเปลี่ยนแปลงได้ เหมาะกับงานที่ต้องทำเป็นประจำและการรับข้อมูลของงานเป็นลักษณะเดิมตลอด เช่น การตรวจสอบบัญชีในธนาคาร จะมีข้อมูลที่รับเข้ามา คือ เลขบัญชี

คำถามลักษณะนี้ สามารถทำการรับข้อมูลและแสดงผลได้ง่าย เพราะว่าข้อมูลที่รับเข้ามาและผลลัพธ์ จะมีรูปแบบที่แน่นอน

3.1.2 คำถามที่ไม่ถูกกำหนดล่วงหน้า

คำถามลักษณะนี้ เหมาะกับงานที่ไม่สามารถกำหนดอินพุทล่วงหน้าได้ หรืองานที่มีคำถามหลายรูปแบบการรับข้อมูลและแสดงผลจะทำได้ยากกว่าคำถามที่ถูกกำหนดไว้แล้ว เพราะว่าไม่สามารถรู้ล่วงหน้าได้ว่าคำถามจะมีลักษณะอย่างไร และผลลัพธ์ที่ได้จะเป็นอย่างไรบ้าง

3.2 โครงสร้างการทำงาน

โครงสร้างการทำงานสามารถแบ่งได้เป็น 3 ส่วนหลัก คือ

3.2.1 ส่วนติดต่อกับผู้ใช้

ส่วนนี้จะทำหน้าที่ 2 อย่าง คือ

1. รับคำถามจากผู้ใช้ แล้วส่งต่อไปให้ส่วนรับส่งข้อมูล

2. แสดงผลที่ได้จากส่วนรับส่งข้อมูล

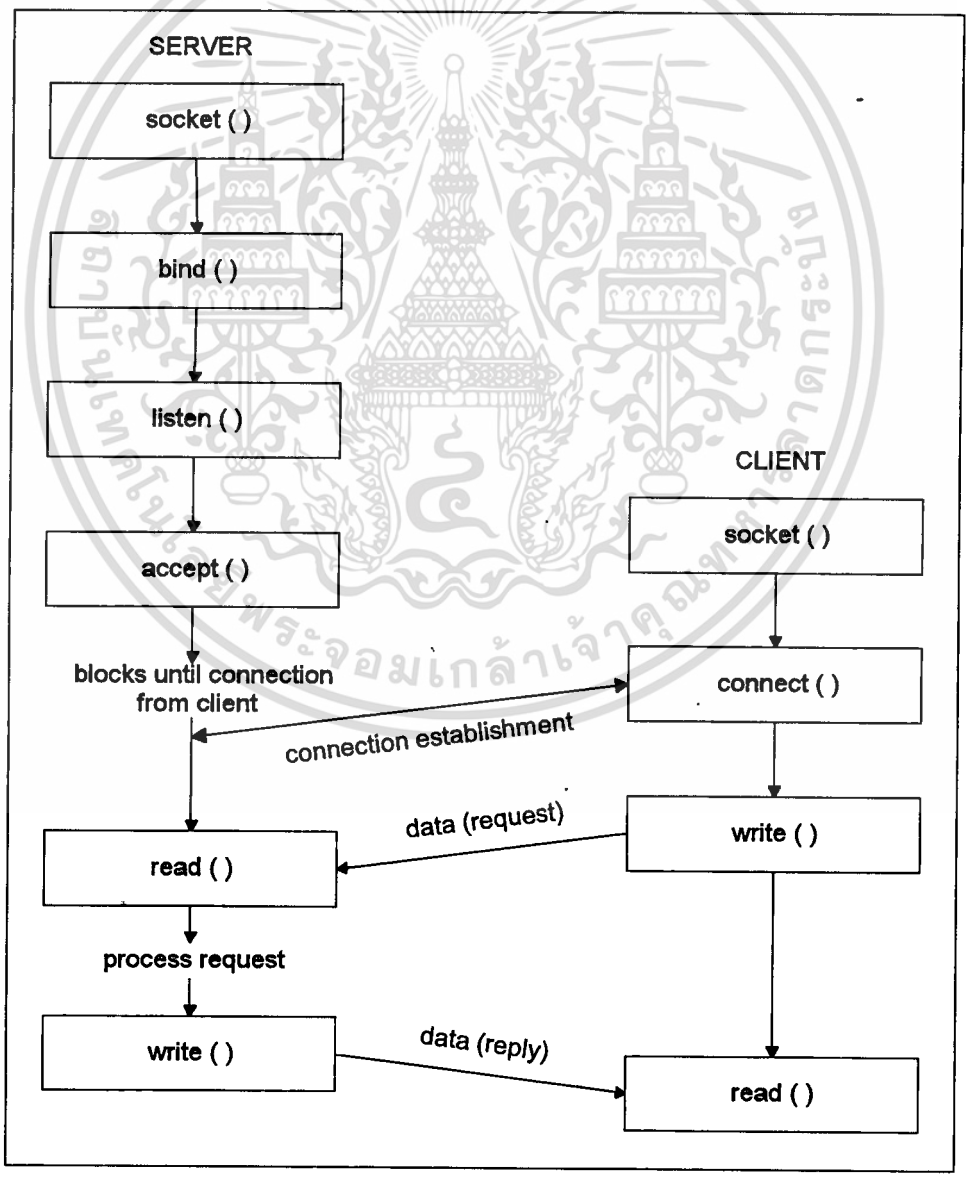
3.2.2 ส่วนรับส่งข้อมูล

ส่วนนี้จะทำหน้าที่ 2 อย่างคือ

1. รับคำถามจากส่วนติดต่อผู้ใช้ และส่งผลที่ได้ให้ส่วนติดต่อกับผู้ใช้

2. ส่งคำถามให้ส่วนประมวลผลคำถาม และรับผลที่ได้จากส่วนประมวลผลคำถาม

ส่วนรับส่งข้อมูลของแต่ละระบบจะทำการติดต่อและรับส่งข้อมูลถึงกันและกันโดยอาศัยกระบวนการต่างๆ ดังรูปที่ 3.1

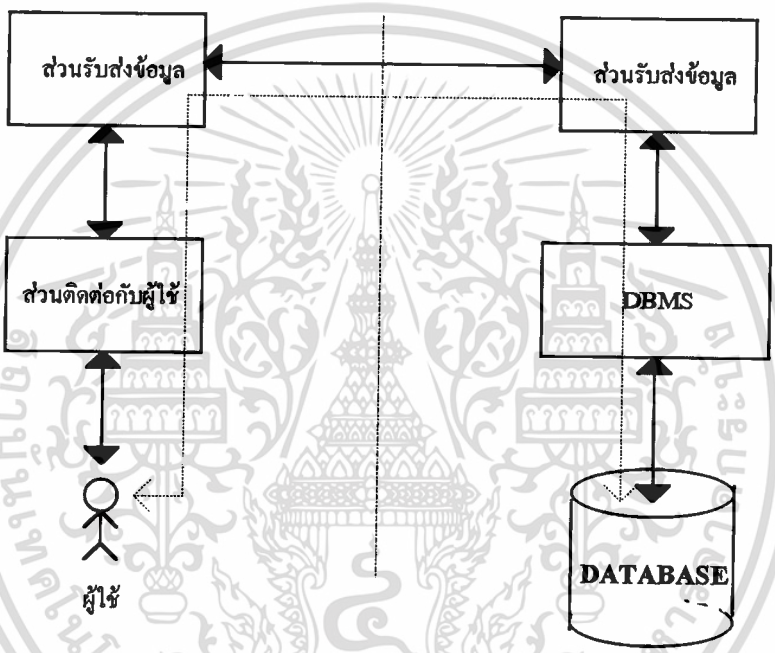


รูปที่ 3.1 โครงสร้างของส่วนรับส่งข้อมูล

3.2.3 ส่วนประมวลผลคำถาม

ส่วนนี้จะรับคำถามจากส่วนรับส่งข้อมูล แล้วนำคำถามที่ได้ไปประมวลผล และนำผลที่ได้ส่งกลับไปให้ส่วนรับส่งข้อมูล

กล่าวโดยสรุปแล้ว สถาปัตยกรรมของระบบในการส่งคำถามไปประมวลผลที่ระบบคอมพิวเตอร์อื่น จะมีลักษณะดังรูปที่ 3.2



รูปที่ 3.2 แสดงโครงสร้างของการประมวลผลคำถามต่างระบบ

บทที่ 4
สถาปัตยกรรมของระบบ
ประมวลผลคำถามแบบกระจาย DDQ/1

ระบบประมวลผลคำถามแบบกระจาย DDQ/1 ซึ่งถูกพัฒนาโดยคุณสมชิต ศิริผลหลาย เป็นแนวทางในการวิจัยของปริญญานิพนธ์ฉบับนี้ จึงมีความจำเป็นที่จะต้องกล่าวถึงสถาปัตยกรรมของระบบ DDQ/1 ซึ่งได้อ้างอิงมาจากวิทยานิพนธ์ของคุณสมชิต ศิริผลหลาย[33]

4.1 สถาปัตยกรรมของระบบ DDQ/1 โดยรวม

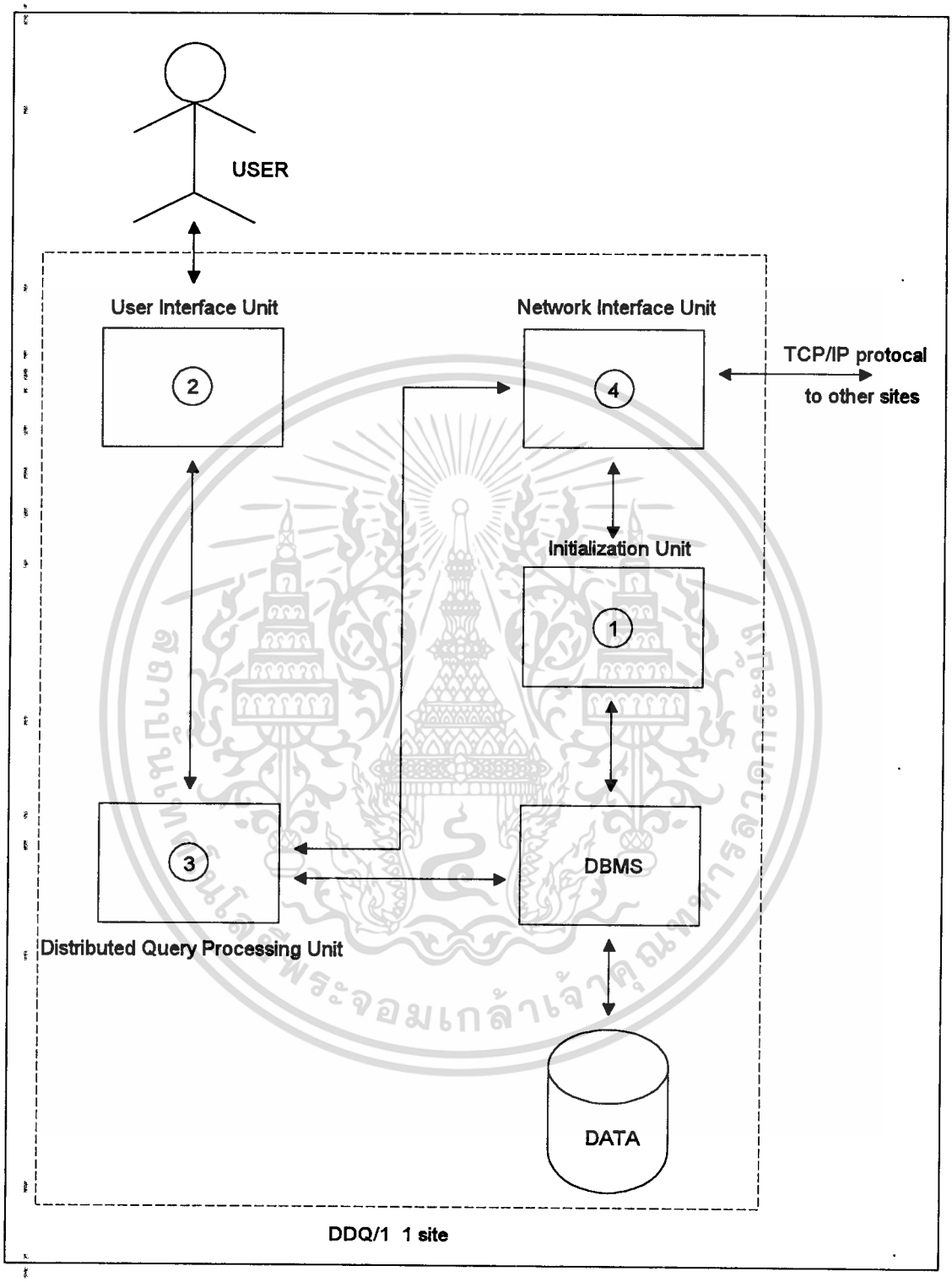
เนื่องจากระบบประมวลผลคำถามแบบกระจาย DDQ/1 ที่ได้พัฒนาขึ้นนี้ ได้ออกแบบสำหรับใช้กับระบบปฏิบัติการทั่วไป ที่เป็นระบบปฏิบัติการแบบหลายงาน (multitasking) คือ สามารถทำงานพร้อมกันได้หลายๆงาน รวมทั้งมีคุณสมบัติที่สามารถสร้างกระบวนการ (process) ขึ้นจากกระบวนการที่ประมวลผลอยู่แล้วได้ โดยอาศัยคุณสมบัติของระบบปฏิบัติการที่กล่าวมา ทำให้สามารถออกแบบระบบฐานข้อมูลแบบกระจาย DDQ/1 ให้มีความสามารถที่จะบริการผู้ใช้งานในสถานที่หนึ่ง ได้หลายผู้ใช้งานในเวลาเดียวกัน

เนื่องจากระบบปฏิบัติการยูนิกซ์ (UNIX) เป็นระบบปฏิบัติการที่มีคุณสมบัติตามที่ได้กล่าวมาข้างต้น [18] และเป็นระบบปฏิบัติการที่นิยมใช้กันอย่างแพร่หลาย จึงได้เลือกใช้ระบบปฏิบัติการยูนิกซ์ เป็นระบบปฏิบัติการในการทดลองสร้างระบบประมวลผลคำถามแบบกระจาย DDQ/1 โดยการออกแบบส่วนจัดการฐานข้อมูลแบบกระจาย ที่อยู่บนเครื่องคอมพิวเตอร์แต่ละเครื่อง จะประกอบไปด้วยส่วนทำงานหลักๆ 4 ส่วนด้วยกัน ที่ทำงานประสานกันดังแสดงไว้ในรูปที่ 4.1 ในหัวข้อนี้จะกล่าวถึงการทำงานของแต่ละส่วนโดยสังเขปเท่านั้น เพื่อจะได้เข้าใจถึงการทำงานร่วมกันของทั้ง 4 ส่วน และการเชื่อมต่อกับฐานข้อมูลที่อยู่บนคอมพิวเตอร์เครื่องอื่น รายละเอียดในแต่ละส่วนจะได้กล่าวถึงในหัวข้อ 4.2 ถึง 4.5 ต่อไป สำหรับการทำงานร่วมกันของทั้ง 4 ส่วน มีดังต่อไปนี้

4.1.1 ส่วนจัดการกระบวนการเริ่มต้น

ส่วนนี้จะเป็นโปรแกรมที่มีหน้าที่จัดการเกี่ยวกับพจนานุกรมฐานข้อมูล (Data Diotionary หรือ System Catalog) ของระบบฐานข้อมูลแบบกระจาย และจัดเก็บกฎบังคับความถูกต้อง (Constraints หรือ Validation Rules) เข้าสู่ฐานข้อมูล โดยผู้ดูแลระบบฐานข้อมูลท้องถิ่น (Local Database Administrator) จะเป็นผู้เรียกใช้โปรแกรมส่วนนี้ในครั้งแรกที่ทำการติดตั้งระบบฐานข้อมูล DDQ/1

ในส่วนจัดเก็บกฎบังคับความถูกต้องนั้น จะรับกฎบังคับความถูกต้องจากผู้ดูแลระบบฐานข้อมูล และจัดเก็บเข้าสู่ฐานข้อมูล ปรกติโปรแกรมส่วนนี้จะถูกเรียกใช้ในตอนติดตั้งระบบเท่านั้น แต่ก็สามารถที่จะเรียกใช้ได้เมื่อมีการเปลี่ยนแปลงกฎบังคับความถูกต้อง



รูปที่ 4.1 แสดงโครงสร้างระบบฐานข้อมูลแบบกระจาย DDQ/1 บนคอมพิวเตอร์แต่ละเครื่อง

สำหรับส่วนจัดการเกี่ยวกับพจนานุกรมฐานข้อมูล จะทำหน้าที่ในการรับข้อมูลจากผู้ดูแลระบบฐานข้อมูล เก็บเข้าสู่พจนานุกรมฐานข้อมูลท้องถิ่น และยังนำข้อมูลของพจนานุกรมฐานข้อมูลท้องถิ่นจากคอมพิวเตอร์เครื่องอื่นๆ มาสร้างเป็นพจนานุกรมฐานข้อมูลโดยรวมของฐานข้อมูลบนคอมพิวเตอร์เครื่องนั้นๆ ด้วยวิธีการส่ง

การส่งคำถามไปยังคอมพิวเตอร์ทีละเครื่องที่อยู่ในระบบฐานข้อมูลแบบกระจายเดียวกัน ผ่านส่วนเชื่อมต่อโครงข่าย (Network Interface) จากนั้นจะนำข้อมูลที่ได้ส่งให้กับระบบจัดการฐานข้อมูลออราเคิล (ORACLE) จัดเก็บไว้ในตารางพจนานุกรมฐานข้อมูลโดยรวม เพื่อใช้ในการประมวลผลของส่วนอื่นๆต่อไป

หลังจากผู้ดูแลระบบฐานข้อมูลเรียกใช้โปรแกรมส่วนนี้ และโปรแกรมทำงานเสร็จสมบูรณ์แล้ว กระบวนการของโปรแกรมก็จะสิ้นสุดลง และผู้ใช้งานสามารถจะเข้าถึงข้อมูลได้ โดยไม่จำเป็นจะต้องมีกระบวนการของส่วนเริ่มต้นทำงานอยู่

4.1.2 ส่วนติดต่อผู้ใช้งาน

ส่วนติดต่อผู้ใช้งานนี้ จะเป็นโปรแกรมที่ผู้ใช้งานเป็นผู้เรียกใช้ในการเข้าถึงข้อมูล โดยส่วนนี้จะมีหน้าที่ในการรับคำถามจากผู้ใช้งาน เป็นภาษาพีชคณิตสัมพันธ์ และส่งคำถามที่รับมาจากผู้ใช้งานให้กับส่วนประมวลผลคำถามแบบกระจาย ในรูปแบบของต้นไม้เชิงพีชคณิตสัมพันธ์ (Relational Algebra Tree) จากนั้นจะรอรับข้อมูลที่เป็นผลลัพธ์ของคำถามจากส่วนประมวลผลคำถามแบบกระจาย เพื่อนำกลับมาแสดงผลแก่ผู้ใช้งาน และรอรับคำถามใหม่จากผู้ใช้งานต่อไป จนกว่าการทำงานจะสิ้นสุดด้วยการรับคำสั่งสิ้นสุดการทำงานจากผู้ใช้งานโดยตรง

จากการที่ผู้ใช้งานแต่ละคน สามารถเรียกใช้โปรแกรมนี้ได้อย่างอิสระ ทำให้กระบวนการติดต่อผู้ใช้งานในแต่ละกระบวนการของผู้ใช้งานแต่ละคน มีการใช้หน่วยความจำเพื่อเก็บข้อมูลเกี่ยวกับกระบวนการ (process information) และคำสั่งทำงาน (instructions) ได้อย่างอิสระ หลังจากที่ผู้ใช้งานเสร็จสิ้นการใช้งานแล้ว กระบวนการติดต่อผู้ใช้งานของผู้ใช้งานนั้นๆ ก็จะสิ้นสุดลงด้วย ทำให้ระบบปฏิบัติการสามารถที่จะนำหน่วยความจำที่ถูกใช้งานโดยกระบวนการที่สิ้นสุดลงไปใช้งานอย่างอื่นได้

4.1.3 ส่วนประมวลผลคำถามแบบกระจาย

ส่วนนี้จะรับคำถามในลักษณะต้นไม้เชิงพีชคณิตสัมพันธ์จากส่วนติดต่อผู้ใช้งาน หรือจากคอมพิวเตอร์เครื่องอื่นที่ส่งผ่านมาทางส่วนเชื่อมต่อโครงข่าย แล้วทำการวิเคราะห์คำถามที่รับมาได้ เพื่อให้การประมวลผลคำถามได้ผลลัพธ์ในเวลาน้อยที่สุด และให้มีการส่งข้อมูลน้อยที่สุดในกรณีที่ต้องส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ จากนั้นจะเปลี่ยนคำถามที่ผ่านการวิเคราะห์แล้วเป็นภาษาสอบถามเชิงโครงสร้าง (SQL) เพื่อส่งให้ระบบจัดการฐานข้อมูลออราเคิล ในกรณีที่คำถามต้องการเข้าถึงข้อมูลในสถานที่อื่น ส่วนประมวลผลคำถามก็จะส่งคำถามในลักษณะต้นไม้เชิงพีชคณิตสัมพันธ์ที่ผ่านการวิเคราะห์แล้ว ไปยังเครื่องคอมพิวเตอร์ที่เก็บข้อมูลอยู่ และรอรับข้อมูลคำตอบส่งกลับให้กับส่วนที่ส่งคำถามมา การทำงานของกระบวนการประมวลผลคำถามแบบกระจายนี้ จะเป็นลักษณะการทำงานแบบหน่วยบริการ (server) กล่าวคือ โปรแกรมส่วนนี้จะถูกเรียกใช้ครั้งแรกโดยผู้ดูแลเครื่องคอมพิวเตอร์ (System Administrator) แล้วกระบวนการจะรอรับการติดต่อจากกระบวนการส่วนอื่นๆ เมื่อมีการติดต่อมาจากกระบวนการส่วนอื่น กระบวนการประมวลผลคำถามแบบกระจาย ก็จะทำการสร้างกระบวนการขึ้นใหม่ที่มีคุณสมบัติเหมือนเดิม ด้วยการแยก (fork) กระบวนการเป็นสองกระบวนการ ได้แก่ กระบวนการพ่อ (parent process) และกระบวนการลูก (child process) กระบวนการพ่อจะรอรับการติดต่อจากกระบวนการอื่นทันทีที่แยกกระบวนการลูกเสร็จ ส่วนกระบวนการลูกจะทำการประมวลผลคำถามต่อไปจนเสร็จ จากนั้นกระบวนการลูกก็จะสิ้นสุดลง

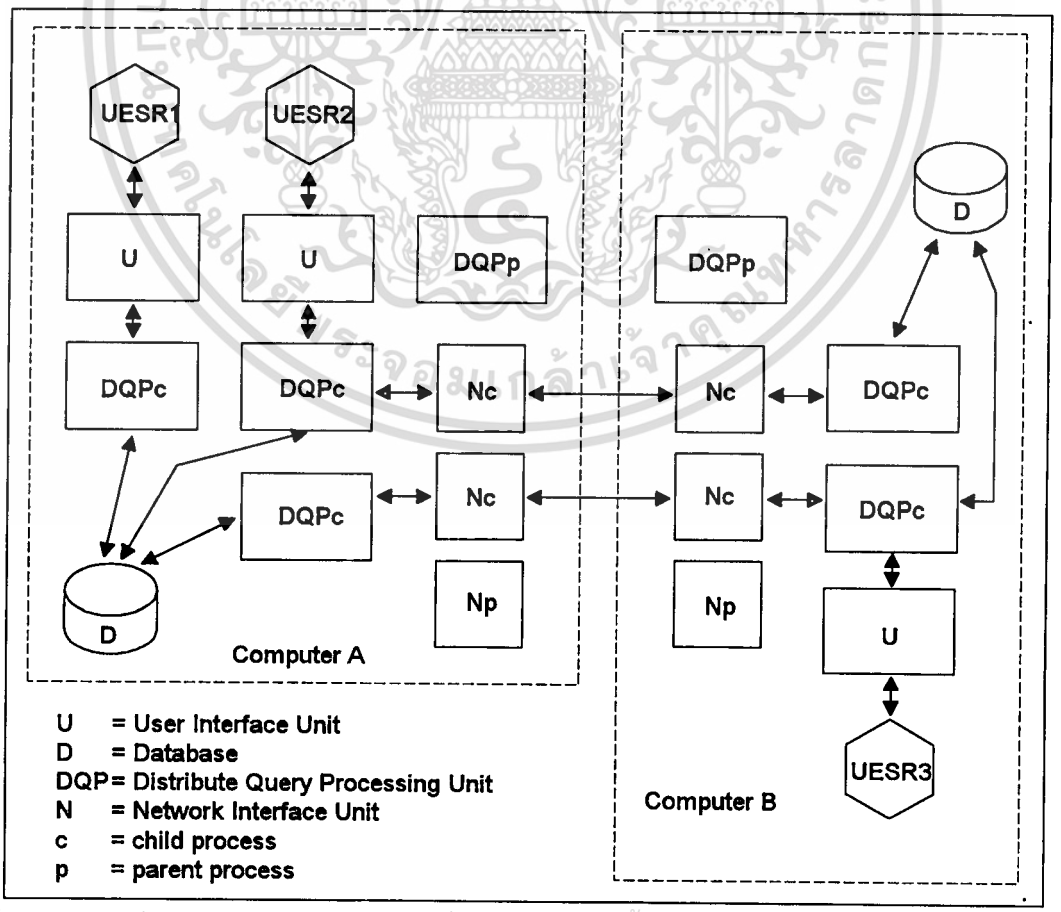
ด้วยคุณสมบัติการแยกกระบวนการนี้เอง ทำให้เวลาขณะหนึ่งๆ อาจจะมีกระบวนการลูกอยู่หลายกระบวนการก็ได้ ซึ่งกระบวนการลูกทั้งหมดที่แยกตัวออกมาจากกระบวนการพ่อ รวมทั้งกระบวนการพ่อเองด้วย

จะมีการใช้หน่วยความจำสำหรับเก็บคำสั่งทำงานร่วมกัน แต่จะแยกเก็บข้อมูลเกี่ยวกับกระบวนการออกจากกัน [18] โดยการทำงานในลักษณะหน่วยบริการนี้ จะทำให้สามารถลดปริมาณการใช้หน่วยความจำลงได้ อย่างไรก็ตามจำนวนกระบวนการจะขึ้นอยู่กับระบบปฏิบัติการ

4.1.4 ส่วนเชื่อมต่อโครงข่าย

ส่วนนี้จะทำหน้าที่คอยรับการติดต่อจากส่วนประมวลผลคำถามแบบกระจาย หรือ ส่วนจัดการกระบวนการเริ่มต้น เพื่อทำหน้าที่เชื่อมต่อกับส่วนเชื่อมต่อโครงข่ายบนคอมพิวเตอร์เครื่องอื่นได้ และทำนองกลับกันก็ทำหน้าที่รับการติดต่อจากส่วนเชื่อมต่อโครงข่ายที่อยู่บนคอมพิวเตอร์เครื่องอื่น ให้สามารถเชื่อมต่อกับส่วนประมวลผลคำถามแบบกระจายได้ เพื่อรับ-ส่งคำถาม หรือ ข้อมูลระหว่างเครื่องคอมพิวเตอร์ ลักษณะการทำงานของส่วนเชื่อมต่อโครงข่ายนี้ จะมีลักษณะเช่นเดียวกับกระบวนการประมวลผลคำถามแบบกระจาย คือ เป็นแบบหน่วยบริการ กล่าวคือ โปรแกรมจะถูกเรียกใช้โดยผู้ดูแลเครื่องคอมพิวเตอร์ครั้งแรกเพียงครั้งเดียว จากนั้นกระบวนการพ่อจะแยกกระบวนการทุกครั้งที่มีการติดต่อจากกระบวนการอื่น

จากสถาปัตยกรรมที่กล่าวมาข้างต้น จะขอยกตัวอย่างแสดงลักษณะโครงสร้างของกระบวนการทั้งหมดขณะใช้งานจริง ดังรูปที่ 4.2 ซึ่งอธิบายได้ดังนี้คือ ที่เครื่องคอมพิวเตอร์ A ผู้ใช้งาน user1 เรียกใช้โปรแกรมส่วนติดต่อผู้ใช้งาน และมีการป้อนคำถามที่เข้าถึงข้อมูล เฉพาะที่อยู่บนเครื่องคอมพิวเตอร์ A เท่านั้น ส่วนผู้ใช้งาน user2 เรียกใช้โปรแกรมส่วนติดต่อผู้ใช้งานจาก



รูปที่ 4.2 แสดงลักษณะโครงสร้างกระบวนการประมวลผลของฐานข้อมูล DDQ/1 ขณะใช้งาน

เครื่องคอมพิวเตอร์ A มีการเข้าถึงข้อมูลที่เก็บไว้ในเครื่องคอมพิวเตอร์ A และเครื่องคอมพิวเตอร์ B สำหรับผู้ใช้งาน user3 นั้น เรียกใช้โปรแกรมส่วนติดต่อผู้ใช้งานจากเครื่องคอมพิวเตอร์ B และคำถามที่ป้อนให้กับโปรแกรมมีการเข้าถึงข้อมูลที่อยู่บนเครื่องคอมพิวเตอร์ A และ B

4.2 ส่วนจัดการกระบวนการเริ่มต้น (Initialization Unit)

ส่วนจัดการกระบวนการเริ่มต้นนี้ จะทำหน้าที่ในการสร้างและปรับปรุงข้อมูลของพจนานุกรมฐานข้อมูล ของระบบฐานข้อมูลแบบกระจายที่ใช้สำหรับเครื่องคอมพิวเตอร์นั้นๆ และยังทำหน้าที่ในการจัดเก็บกฎบังคับความถูกต้องด้วย ดังนั้นในส่วนรายละเอียดจะแยกกล่าวตามการทำงานเป็นสองส่วน คือ ส่วนพจนานุกรมฐานข้อมูล และส่วนจัดเก็บกฎบังคับความถูกต้องดังต่อไปนี้

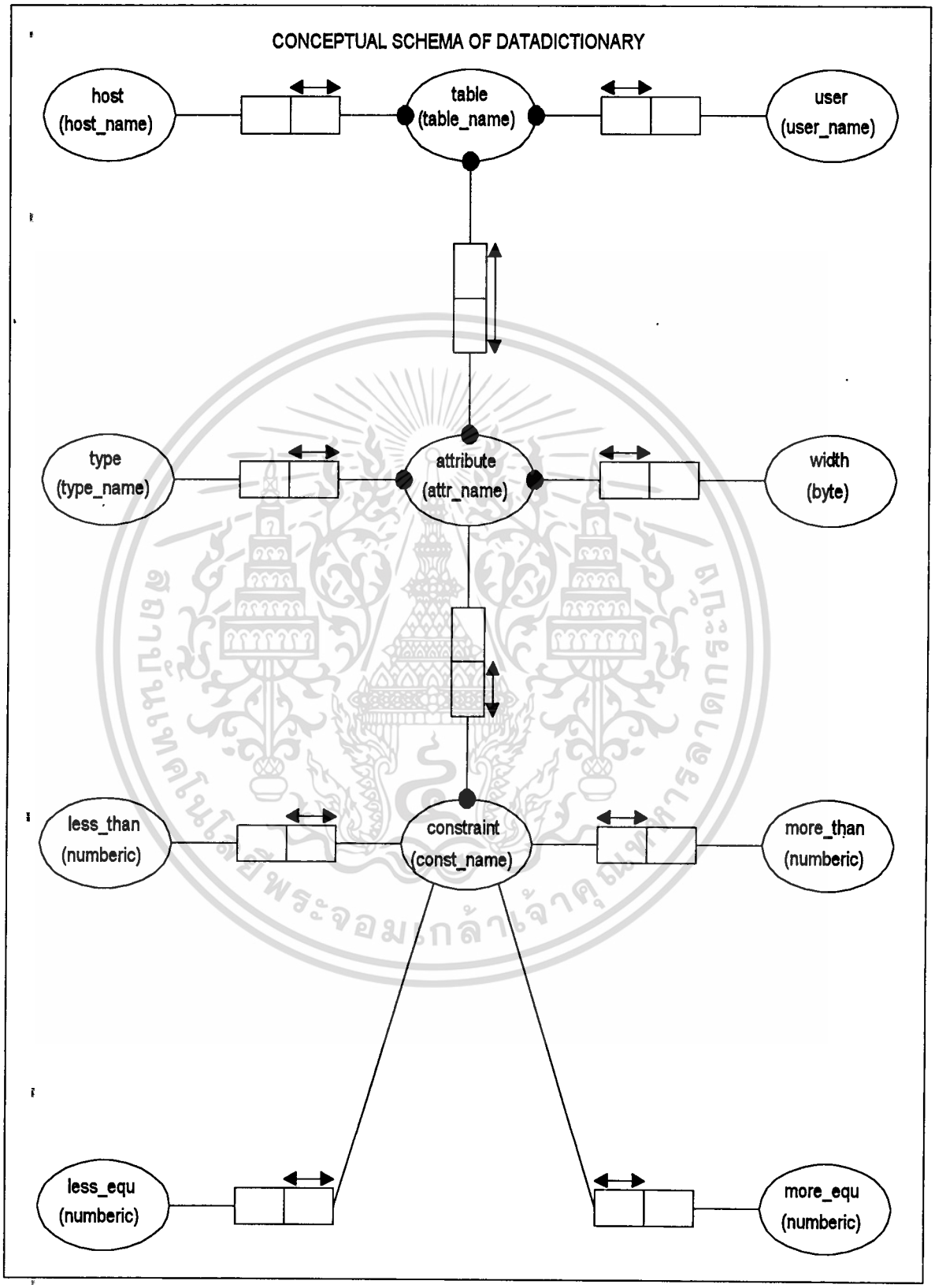
4.2.1 ส่วนพจนานุกรมฐานข้อมูล

โดยปรกติแล้ว ระบบฐานข้อมูลที่ทำงานได้โดยอิสระ จะมีพจนานุกรมฐานข้อมูลใช้อยู่แล้ว เพื่อเก็บข้อมูลที่เกี่ยวกับฐานข้อมูลนั้นๆ เมื่อระบบฐานข้อมูลถูกนำมาสร้างเป็นระบบฐานข้อมูลแบบกระจายจำเป็นจะต้องมีพจนานุกรมฐานข้อมูลสำหรับระบบฐานข้อมูลแบบกระจายด้วย สำหรับเก็บข้อมูลที่เกี่ยวกับระบบฐานข้อมูลแบบกระจาย เพื่อใช้สำหรับการประมวลผลในขั้นตอนต่างๆ ซึ่งส่วนพจนานุกรมนี้จะประกอบไปด้วยส่วนย่อยๆหลายส่วนด้วยกัน อธิบายได้ดังนี้ คือ

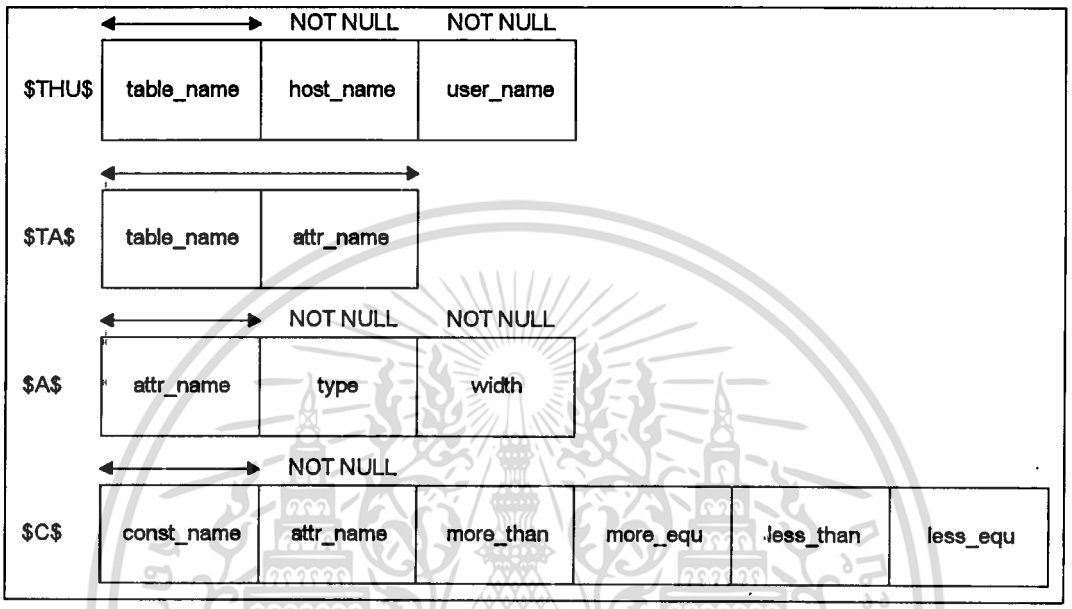
4.2.1.1 ส่วนสร้างตารางพจนานุกรมฐานข้อมูล

ตารางพจนานุกรมฐานข้อมูล นับเป็นส่วนสำคัญส่วนหนึ่งในการออกแบบ และสร้างระบบฐานข้อมูล ซึ่งการออกแบบตารางพจนานุกรมฐานข้อมูล จะมีผลถึงการออกแบบส่วนอื่นๆด้วย สำหรับตารางพจนานุกรมฐานข้อมูลของระบบ DDQ/1 ที่สร้างขึ้น ได้ออกแบบให้สามารถเก็บกฎบังคับความถูกต้องไว้ด้วย โดยการใช้รูปแบบการแสดง Conceptual Sohema ในรูปของ NIAM (Nijssen Information Analysis Methodology) [21] เป็นบรรทัดฐานในการออกแบบ โดยใช้ข้อกำหนดว่า ในระบบฐานข้อมูลที่ทำงานได้โดยอิสระทั้งหมดนั้น จะไม่มีความสัมพันธ์(ตารางข้อมูล)ใดที่อยู่ในระบบฐานข้อมูลที่ทำงานได้โดยอิสระมากกว่าหนึ่งระบบ และชื่อของแอททริบิวที่เหมือนกันจะต้องเก็บข้อมูลที่มีโดเมนเดียวกันด้วย และหากมีกฎบังคับความถูกต้องที่ระบุถึงชื่อแอททริบิวใด กฎบังคับความถูกต้องนั้นก็จะมีผลต่อทุกๆแอททริบิวที่มีชื่อเหมือนกัน โดยกฎบังคับความถูกต้องจะเป็นชนิดบอกโดเมน (domain constraint) ด้วยการใช้ข้อกำหนดดังกล่าวสามารถที่จะสร้างรูปแบบของ NIAM ได้ดังแสดงในรูปที่ 4.3 และเปลี่ยนเป็นตารางสำหรับเก็บข้อมูลของพจนานุกรมฐานข้อมูลได้ ดังแสดงในรูปที่ 4.4 ซึ่งเครื่องคอมพิวเตอร์แต่ละเครื่องที่ประกอบขึ้นเป็นระบบฐานข้อมูลแบบกระจาย DDQ/1 จะต้องมีตารางพจนานุกรมฐานข้อมูลแบบกระจายเป็นของตัวเอง

สำหรับโปรแกรมในส่วนสร้างตารางพจนานุกรมฐานข้อมูล จะทำหน้าที่ในการสร้างตารางพจนานุกรมฐานข้อมูลท้องถิ่น และตารางพจนานุกรมฐานข้อมูลโดยรวม รวมทั้งสร้างตารางเก็บกฎบังคับความถูกต้องท้องถิ่นและตารางเก็บกฎบังคับความถูกต้องโดยรวมด้วย โดยโปรแกรมในส่วนนี้ผู้ดูแลระบบฐานข้อมูลท้องถิ่น จะเป็นผู้เรียกใช้ในขั้นตอนการติดตั้งระบบเพียงครั้งเดียวเท่านั้น หลังจากนั้นจะไม่มีการเรียกใช้โปรแกรมอีกเลย เว้นแต่ว่าจะทำการติดตั้งระบบใหม่



รูปที่ 4.3 แสดง Conceptual Schema ในรูปแบบ NIAM



รูปที่ 4.4 แสดงรูปแบบตารางของพจนานุกรมฐานข้อมูล DDQ/1

4.2.1.2 ส่วนป้อนข้อมูลพจนานุกรมฐานข้อมูลท้องถิ่น

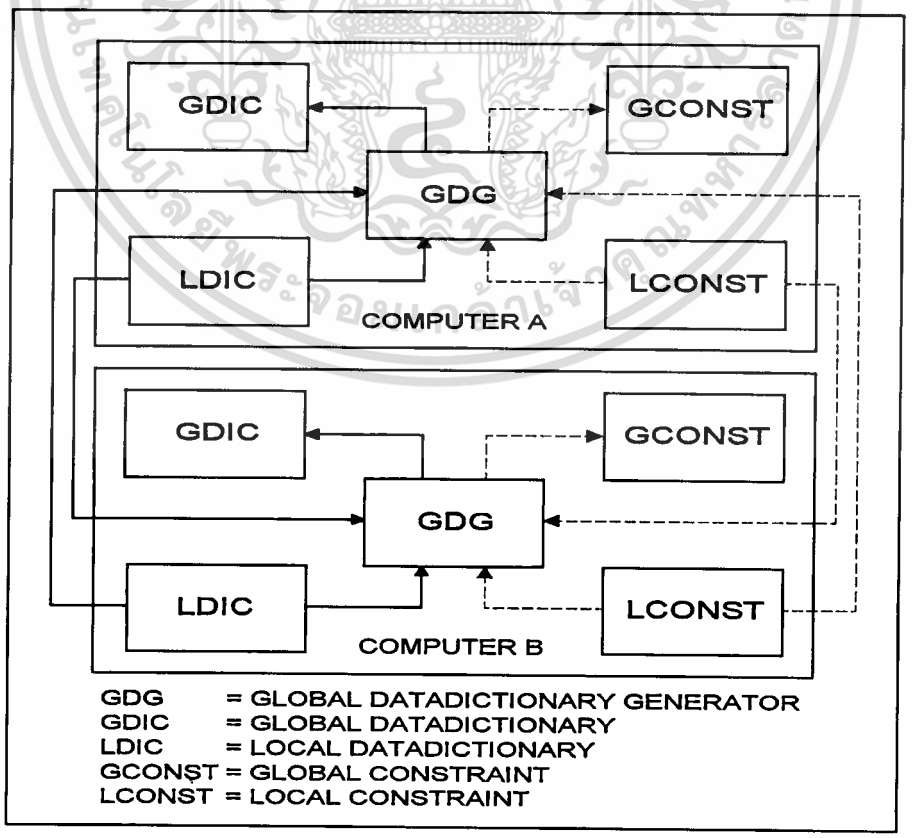
พจนานุกรมฐานข้อมูลท้องถิ่น เป็นตารางข้อมูลที่สร้างขึ้นเพื่อเตรียมข้อมูล สำหรับสร้างพจนานุกรมฐานข้อมูลโดยรวมของระบบฐานข้อมูลแบบกระจาย DDQ/1 ซึ่งผู้ดูแลระบบฐานข้อมูลท้องถิ่นจะเป็นผู้ป้อนข้อมูลเข้าสู่พจนานุกรมฐานข้อมูลท้องถิ่น ด้วยการเรียกใช้โปรแกรม ซึ่งโปรแกรมในส่วนนี้จะทำการค้นหาข้อมูลจากพจนานุกรมฐานข้อมูลของระบบฐานข้อมูลที่ทำงานได้โดยอิสระ แล้วทำการเพิ่มข้อมูลเข้าสู่พจนานุกรมฐานข้อมูลท้องถิ่น หลังจานนั้นส่วนป้อนข้อมูลพจนานุกรมฐานข้อมูลโดยรวม จะเป็นส่วนที่นำข้อมูลนี้ไปใช้ในการสร้างพจนานุกรมฐานข้อมูลโดยรวมต่อไป ซึ่งขั้นตอนในการสร้างพจนานุกรมฐานข้อมูลโดยรวม อธิบายไว้ในหัวข้อต่อไป

เนื่องจากพจนานุกรมฐานข้อมูลท้องถิ่น ใช้เก็บข้อมูลที่จะนำไปสร้างพจนานุกรมฐานข้อมูลโดยรวม ดังนั้นในการออกแบบ Conceptual Sohema ในรูปแบบของ NIAM สำหรับพจนานุกรมฐานข้อมูลท้องถิ่น จึงมีลักษณะเหมือนกับ Conceptual Sohema ของพจนานุกรมฐานข้อมูลโดยรวม ดังนั้น Conceptual Sohema ในรูปแบบของ NIAM จึงมีลักษณะเช่นเดียวกับในรูปที่ 4.3 และเปลี่ยนเป็นตารางพจนานุกรมฐานข้อมูลท้องถิ่นได้เช่นเดียวกับตารางในรูปที่ 4.4 แต่เนื่องจากในฐานข้อมูลหนึ่งๆที่สร้างเป็นระบบฐานข้อมูลแบบกระจาย DDQ/1 จะต้องมีการสร้างพจนานุกรมฐานข้อมูล 2 ส่วน คือ พจนานุกรมฐานข้อมูลท้องถิ่น และพจนานุกรมฐานข้อมูลโดยรวม ซึ่งทั้งสองจะมีลักษณะเหมือนกัน ดังนั้นเพื่อให้เกิดความแตกต่าง ในการตั้งชื่อตารางพจนานุกรมฐานข้อมูลท้องถิ่น จึงได้ใช้ตัวอักษรภาษาอังกฤษ " L " นำหน้าชื่อตารางที่ได้จากรูปที่ 4.4 ซึ่งชื่อตารางทั้ง 4 จะเป็นดังนี้ LTHU , LTA , LA , LC

4.2.1.3 ส่วนป้อนข้อมูลพจนานุกรมฐานข้อมูลโดยรวม

ในการสร้างพจนานุกรมฐานข้อมูลโดยรวมในแต่ละเครื่องนั้น จะกระทำได้โดยการนำข้อมูลในพจนานุกรมฐานข้อมูลท้องถิ่น จากคอมพิวเตอร์เครื่องอื่นๆ ที่ต้องการสร้างเป็นระบบฐานข้อมูลแบบกระจาย DDQ/1 มารวมกับพจนานุกรมฐานข้อมูลท้องถิ่นที่อยู่บนคอมพิวเตอร์เครื่องนั้นๆ โดยการส่งคำถามในรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ ผ่านทางส่วนเชื่อมต่อโครงข่าย ไปยังคอมพิวเตอร์เครื่องที่ต้องการจะติดต่อด้วย และนำข้อมูลที่ได้ป้อนเข้าสู่พจนานุกรมฐานข้อมูลโดยรวมของคอมพิวเตอร์เครื่องนั้นๆ ในขั้นตอนการส่งคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์นั้น โปรแกรมในส่วนนี้ จะส่งคำถามที่เข้าถึงข้อมูลกฎบังคับความถูกต้องท้องถิ่นในคอมพิวเตอร์เครื่องที่กำลังติดต่ออยู่ด้วย และนำข้อมูลป้อนเข้าสู่ตารางข้อมูลกฎบังคับความถูกต้องโดยรวม เช่นเดียวกับข้อมูลพจนานุกรมฐานข้อมูล ดังแสดงตัวอย่างในรูปที่ 4.5 ซึ่งเป็นการสร้างพจนานุกรมฐานข้อมูลของระบบฐานข้อมูลแบบกระจาย DDQ/1ที่ประกอบด้วยระบบฐานข้อมูล 2 ระบบ

ในระบบปฏิบัติการยูนิกซ์ โดยปรกติแล้วจะมีไฟล์ชื่อ /etc/hosts อยู่ ซึ่งไฟล์นี้จะเก็บชื่อของเครื่องคอมพิวเตอร์ (host name) และที่อยู่ (address) ของคอมพิวเตอร์ที่เชื่อมต่อกันเป็นระบบโครงข่าย[19] ดังนั้นในการกำหนดเครื่องคอมพิวเตอร์ที่จะติดต่อด้วย จะกระทำได้ด้วยการดูรายชื่อเครื่องคอมพิวเตอร์ในไฟล์ /etc/hosts และทำการติดต่อทีละเครื่องจนกว่าจะครบทุกเครื่อง ในกรณีที่มีคอมพิวเตอร์เครื่องใดเครื่องหนึ่ง ที่ไม่ได้อยู่ในระบบฐานข้อมูล DDQ/1 แต่ถูกเชื่อมต่อไว้ในโครงข่ายเดียวกันนั้น ในการติดต่อเพื่อที่จะส่งคำถามนั้นจะไม่มีการตอบสนองกลับมา ทำให้ทราบได้ว่า คอมพิวเตอร์เครื่องนั้นไม่ได้อยู่ในระบบ DDQ/1



รูปที่ 4.5 แสดงตัวอย่างการสร้างพจนานุกรมฐานข้อมูล

4.2.2 ส่วนป้อนกฎบังคับความถูกต้อง

ส่วนป้อนกฎบังคับความถูกต้องนี้จะเป็นโปรแกรมที่ถูกเรียกใช้โดยผู้ดูแลระบบฐานข้อมูลท้องถิ่น เพื่อใช้ในการป้อนกฎบังคับความถูกต้องเข้าสู่ตารางเก็บกฎบังคับความถูกต้องท้องถิ่น สำหรับนำไปสร้างตารางเก็บกฎบังคับความถูกต้องโดยรวม ในการเรียกใช้โปรแกรมนั้น ผู้ดูแลระบบจะป้อนกฎบังคับความถูกต้องในรูปของประโยคคำสั่ง ที่มีไวยากรณ์ดังนี้คือ

command	:=	EXIT ; | statement
statement	:=	CREATE CONSTRAINT <constraint_name> ON ATTRIBUTE <attribute_name> WHERE predicate ; | DELETE CONSTRAINT <constraint_name> ;
predicate	:=	comparison | predicate conjunction comparison
comparison	:=	<attribute_name> comp <constraint>
conjunction	:=	AND | OR
comp	:=	= | ^= | < | > | <= | >=

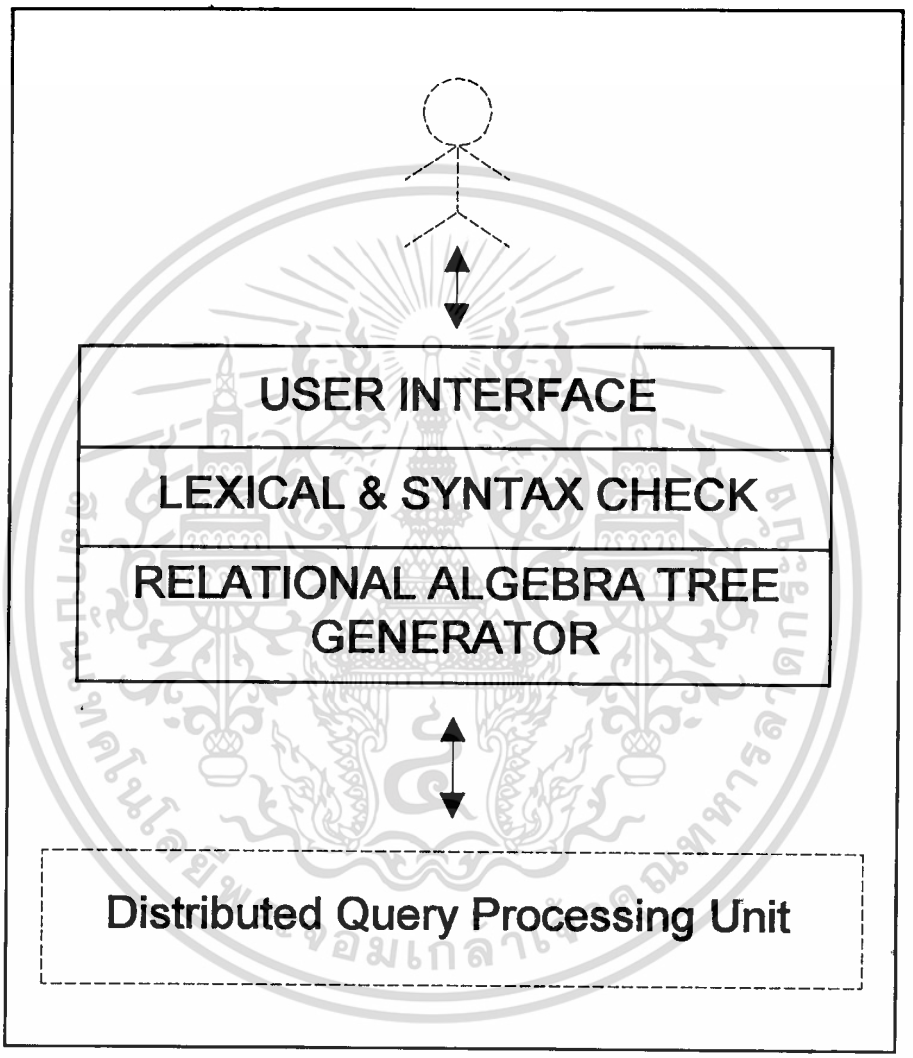
จากไวยากรณ์ที่กล่าวมาข้างต้น constraint_name ที่ผู้ดูแลระบบฐานข้อมูลท้องถิ่นใช้นั้น จะถูกกำหนดจากผู้ดูแลระบบฐานข้อมูลโดยรวม เพื่อไม่ให้มีการกำหนด constraint_name ซ้ำกันในแต่ละสถานที่ และแอททริบิวหนึ่งๆนั้นสามารถมีกฎบังคับความถูกต้องได้มากกว่าหนึ่งกฎ อย่างไรก็ตาม แอททริบิวอาจจะไม่มีการกำหนดกฎบังคับความถูกต้องก็ได้

4.3 ส่วนติดต่อผู้ใช้งาน

ส่วนติดต่อผู้ใช้งานนี้ เป็นโปรแกรมที่ถูกเรียกใช้จากผู้ใช้งาน เพื่อเข้าถึงข้อมูลที่ต้องการ ดังที่ได้อธิบายไว้ในหัวข้อที่ 4.1.2 แล้วนั้น ประกอบไปด้วยส่วนทำงาน 3 ส่วน คือ ส่วนติดต่อผู้ใช้ (user interface) ส่วนแยกคำและตรวจสอบไวยากรณ์ (lexical and syntax check) และส่วนสร้างคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ (relational algebra tree generator) โดยส่วนติดต่อผู้ใช้จะรับคำถามเป็นภาษาพีชคณิตสัมพันธ์ในรูปประโยคข้อความ เพื่อส่งให้กับส่วนแยกคำและตรวจสอบไวยากรณ์ ส่วนแยกคำและตรวจสอบไวยากรณ์จะรับคำถามในรูปประโยคข้อความ แล้วทำการแยกคำ จากนั้นจะทำการตรวจสอบประโยคคำถามว่าถูกต้องตามหลักไวยากรณ์ภาษาพีชคณิตสัมพันธ์หรือไม่ หากคำถามไม่เป็นไปตามหลักไวยากรณ์ภาษาพีชคณิตสัมพันธ์ ส่วนแยกคำและตรวจสอบไวยากรณ์ก็จะส่งข้อความแสดงข้อผิดพลาดไปยังส่วนติดต่อผู้ใช้เพื่อแสดงผล และส่วนติดต่อผู้ใช้ก็จะกลับมาทำงานเพื่อรับคำถามใหม่ต่อไป

ในกรณีที่ประโยคคำถามถูกต้องตามหลักไวยากรณ์พีชคณิตสัมพันธ์ ส่วนแยกคำและตรวจสอบไวยากรณ์ ก็จะส่งกลุ่มคำที่ได้แยกไว้ต่อไปยังส่วนสร้างคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ เพื่อทำการสร้างคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ จากนั้นส่วนสร้างคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ จะส่งคำถามรูปแบบต้นไม้เชิงพีชคณิตสัมพันธ์ไปยังส่วนประมวลผลคำถามแบบกระจายต่อไป

หลังจากที่ส่วนประมวลผลคำถามแบบกระจายทำการประมวลผลเสร็จ ส่วนติดต่อผู้ใช้ก็จะนำข้อมูลไปแสดงผลต่อไป และพร้อมที่จะรับคำถามใหม่ต่อไป ซึ่งโครงสร้างของส่วนติดต่อผู้ใช้งานเป็นไปดังแสดงไว้ในรูปที่ 4.6



รูปที่ 4.6 แสดงโครงสร้างส่วนติดต่อผู้ใช้งาน

4.4 ส่วนประมวลผลคำถามแบบกระจาย

ส่วนประมวลผลคำถามแบบกระจาย ประกอบไปด้วยส่วนทำงาน 3 ส่วนด้วยกัน คือ ส่วนปรับปรุงและตรวจสอบคำถาม (Global Query Optimization) ส่วนประมวลผลคำถาม (Query Evaluation) และส่วนติดต่อระบบจัดการฐานข้อมูล (RDBMS Interface) ดังแสดงโครงสร้างไว้ในรูปที่ 4.7

บทที่ 5
การประมวลผลคำถาม
บนระบบฐานข้อมูลแบบกระจาย

5.1 ระบบประมวลผลคำถามแบบกระจาย DDQ/II

ระบบประมวลผลคำถามแบบกระจาย DDQ/1 ติดต่อกับผู้ใช้ด้วยภาษาพีชคณิตสัมพันธ์ ซึ่งไม่เป็นที่คุ้นเคยกับผู้ใช้ จึงได้พัฒนาระบบประมวลผลคำถามแบบกระจาย DDQ/II ที่ติดต่อกับผู้ใช้ด้วยภาษา SQL ซึ่งเป็นที่คุ้นเคยกับผู้ใช้มากกว่า แต่เนื่องจากระบบประมวลผมคำถามแบบกระจาย DDQ/1 ยังไม่สามารถรองรับภาษา SQL ได้เต็มรูปแบบ ประกอบกับภาษา SQL มีความซับซ้อนมาก ในระบบประมวลผลคำถามแบบกระจาย DDQ/II จึงได้พัฒนาเพียงแค่บางส่วนของภาษา SQL เท่านั้น

5.1.1 รูปแบบของภาษาที่ใช้ใน DDQ/II

รูปแบบภาษาที่ใช้ในระบบประมวลผลคำถาม DDQ/II เป็นเพียงส่วนหนึ่งของภาษา SQL ซึ่งแบ่งได้เป็น 3 ส่วนหลัก คือ

5.1.1.1 SELECT CLAUSE

ใน SELECT CLAUSE จะมีได้เพียงแค่ชื่อฟิลด์ ซึ่งอาจจะมีชื่อตารางนำได้ แต่ไม่สามารถมีชื่อผู้ใช้นำ และไม่สามารถมีฟังก์ชัน เช่น COUNT, SUM, AVG, MAX และ MIN ใน SELECT CLAUSE ด้วย รูปแบบของ SELECT CLAUSE มีดังนี้ คือ

SELECT [Table-name.]Field-name-list

Table Supplier

SNUM	SNAME

Table Part

PNUM	PNAME

Table Supply

SNQ	PNO	QTY

รูปที่ 5.1 แบบแผนของตาราง Supplier, Part และ Supply

ตัวอย่างที่ 5.1

จากรูปที่ 5.1 SELECT ฟิลด์ SNUM กับ SNAME ของตาราง Supplier และ PNUM กับ PNAME ของตาราง Part จะได้ SELECT CLAUSE ดังนี้

```
SELECT  Supplier.SNUM,SNAME,Part.PNUM,PNAME
```

เนื่องจากข้อจำกัดของระบบประมวลผลคำถามแบบกระจาย DDQ/1 ทำให้ไม่สามารถมีชื่อฟิลด์ใดๆ ซ้ำกันได้ แม้ว่าจะอยู่ต่างตารางกัน ดังนั้นจึงไม่จำเป็นที่จะต้องใส่ชื่อตารางนำหน้าชื่อฟิลด์ และถ้าต้องการเลือกทุกฟิลด์จากตารางก็สามารถแทนด้วยเครื่องหมาย "*" เหมือนในภาษา SQL ปกติ

ตัวอย่างที่ 5.2

ต้องการเลือกทุกฟิลด์ ในตาราง Supplier จะได้ SELECT CLAUSE ดังนี้

```
SELECT  SNUM,SNUME
```

หรือ

```
SELECT  *
```

5.1.1.2 FROM CLAUSE

ใน FROM CLAUSE จะมีได้เพียงแค่ชื่อตารางเท่านั้น ไม่สามารถใส่ชื่อผู้ใช้นำชื่อตาราง แต่อาจทำชื่อเล่น(alias) ได้เหมือนในภาษา SQL ปกติ รูปแบบของ FROM CLAUSE มีดังนี้ คือ

```
FROM  Table-name-list
```

ตัวอย่างที่ 5.3

จากตัวอย่างที่ 5.1 จะได้ FROM CLAUSE ดังนี้

```
FROM  Supplier,Part
```

ตัวอย่างที่ 5.4

จากตัวอย่างที่ 5.1 และ 5.3 ถ้าทำชื่อเล่นโดยให้ตาราง Supplier และ Part มีชื่อเล่นเป็น S และ P ตามลำดับ จะได้ SELECT CLAUSE และ FROM CLAUSE ดังนี้

```
SELECT  S.SNUM,SNAME,P.PNUM,PNAME
FROM    Supplier  S, Part  P
```

5.1.1.3 WHERE CLAUSE

ใน WHERE CLAUSE มีรูปแบบดังนี้

WHERE Search-condition [And/OR Search-condition]

โดย Search-condition มี 2 รูปแบบคือ

1. Operand1 Operator Operand2
2. Operand BETWEEN Constant1 AND Constant2

เมื่อ
- Operand1,Operand2 คือ ชื่อฟิลด์ ซึ่งจะมีชื่อตารางนำหรือไม่มีก็ได้ หรือเป็นค่าคงที่ โดย Operand1 และ Operand2 จะต้องไม่เป็นค่าคงที่พร้อมกัน และชนิดของ Operand1 และ Operand2 จะต้องเหมือนกัน
- Operator คือ >, <, >=, <=, =, <>
- Operand คือ ชื่อฟิลด์ ซึ่งจะมีชื่อตารางนำหรือไม่มีก็ได้ แต่จะต้องมีชนิดเป็นตัวเลข
- Constant1,Constant2 คือ ค่าคงที่ที่เป็นตัวเลข

ตัวอย่างที่ 5.5

ต้องการหา Supplier ที่ไม่ได้ Supply Part P1 โดยแสดง SNAME และ PNAME

```
SELECT  SNAME, PNAME
FROM    Supplier, Part, Supply
WHERE   SNUM = SNO  AND  PNUM = PNO
        AND  PNO <> 'P1'
```

ตัวอย่างที่ 5.6

ต้องการหา Suplier ที่ Supply Part P2 มากกว่า 5 แต่ไม่เกิน 10 โดยแสดง SNAME และ QTY

```
SELECT Supplier.SNAME, QTY
FROM    Supplier, Supply
WHERE  Supplier.SNUM = Supply.SNO  AND  PNO = 'P2'
        AND  QTY BETWEEN 6 AND 10
```

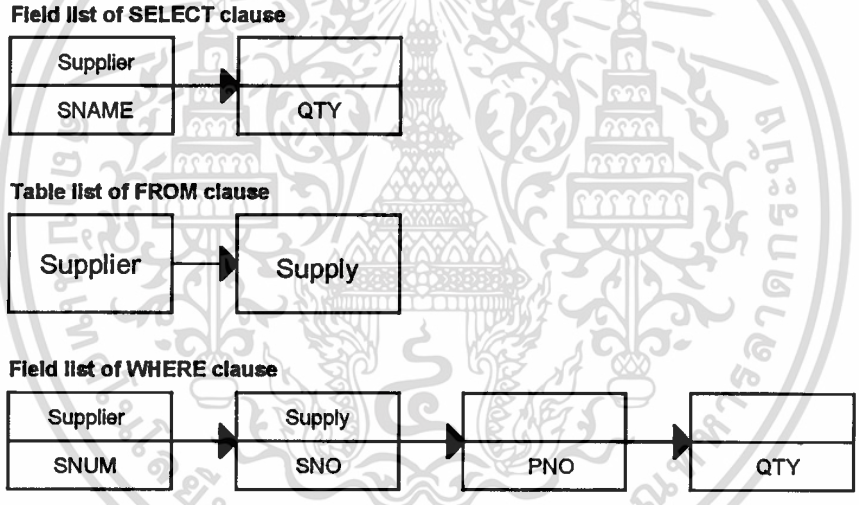
ใน WHERE CLAUSE นี้ ไม่สามารถใช้โอเปอเรเตอร์อื่นๆ นอกจากที่กล่าวมาได้ เช่น LIKE, NOT, ANY และอื่นๆ และไม่สามารถทำคำถามย่อย (sub-query) ด้วย

5.2 การพัฒนาระบบคำถามแบบกระจาย DDQ/II

ระบบประมวลผลคำถามแบบกระจาย DDQ/II ได้พัฒนาเพิ่มจากระบบประมวลผลคำถามแบบกระจาย DDQ/1 เฉพาะส่วนติดต่อกับผู้ใช้เพียงอย่างเดียว ดังนั้นในส่วนอื่นๆ จะไม่ขอกล่าวถึงอีก ส่วนติดต่อกับผู้ใช้ของระบบประมวลผลคำถามแบบกระจาย DDQ/II นี้จะทำการแปลงภาษา SQL ที่รับมาให้เป็นต้นไม้พีชคณิต เพื่อส่งต่อไปให้ส่วนประมวลผลคำถามแบบกระจายของ DDQ/1

5.2.1 ขั้นตอนวิเคราะห์คำและวิเคราะห์ไวยากรณ์

ในขั้นตอนวิเคราะห์คำได้ใช้โปรแกรม LEX เข้ามาช่วย ส่วนในขั้นตอนตรวจสอบไวยากรณ์ก็ใช้โปรแกรม YACC ตรวจสอบตามไวยากรณ์ของคำถามในภาษา SQL ดังแสดงไว้ในภาคผนวก ข หลังจากขั้นตอนทั้งสองนี้แล้ว จะได้ลิสของฟิลด์ที่อยู่ใน SELECT CLAUSE และ WHERE CLAUSE ได้ลิสของตารางที่อยู่ใน FROM CLAUSE ซึ่งจะถูกนำไปทำวิเคราะห์ความหมายต่อไป



รูปที่ 5.2 แสดงลิสของฟิลด์และตารางจากตัวอย่างที่ 5.6

5.2.2 ขั้นตอนวิเคราะห์ความหมาย

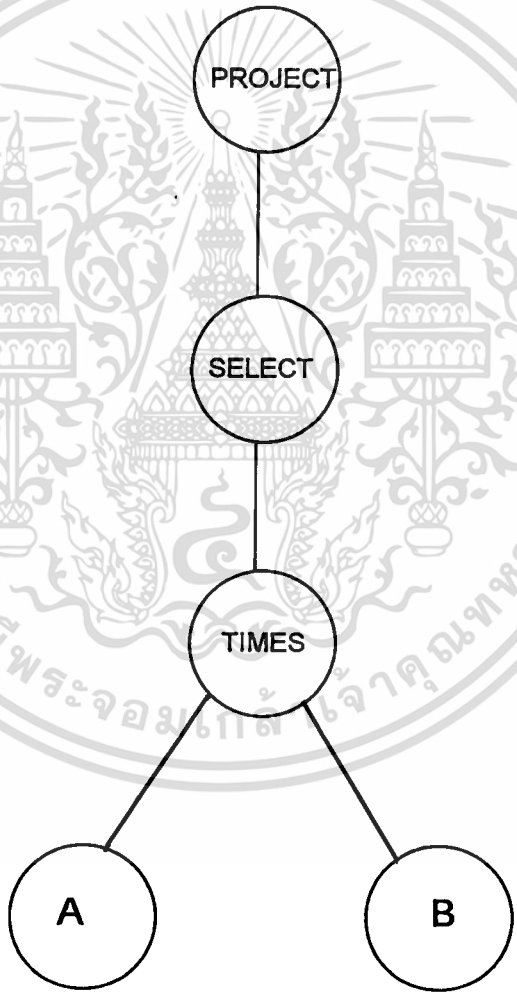
ในขั้นตอนวิเคราะห์ความหมายจะนำลิสที่ได้จากขั้นตอน 5.2.1 มาทำการตรวจสอบกับข้อมูลในพจนานุกรมโดยรวมดังนี้

- ตรวจสอบลิสของตารางใน FROM CLAUSE ว่าตารางมีอยู่จริงหรือไม่
- ตรวจสอบลิสของฟิลด์ใน SELECT CLAUSE และ WHERE CLAUSE ว่าฟิลด์มีอยู่จริงหรือไม่
- ตรวจสอบชนิดของข้อมูลที่นำมาใช้ในการค้นหาใน WHERE CLAUSE ว่าสอดคล้องกันหรือไม่

5.2.3 ขั้นตอนการสร้างต้นไม้พีชคณิต

ในขั้นตอนการสร้างต้นไม้พีชคณิตนี้ จะไม่สร้างต้นไม้พีชคณิตให้อยู่ในรูปที่ดีที่สุด แต่จะสร้างต้นไม้พีชคณิตแบบง่ายๆ เท่านั้น เพราะว่าต้นไม้พีชคณิตนี้จะถูกนำไปปรับปรุงอีกครั้งโดยส่วนประมวลผลคำถามแบบกระจาย รูปทั่วไปของต้นไม้พีชคณิตที่ได้จะเป็นดังรูปที่ 5.3 และขั้นตอนการสร้างต้นไม้พีชคณิตมีดังนี้ คือ

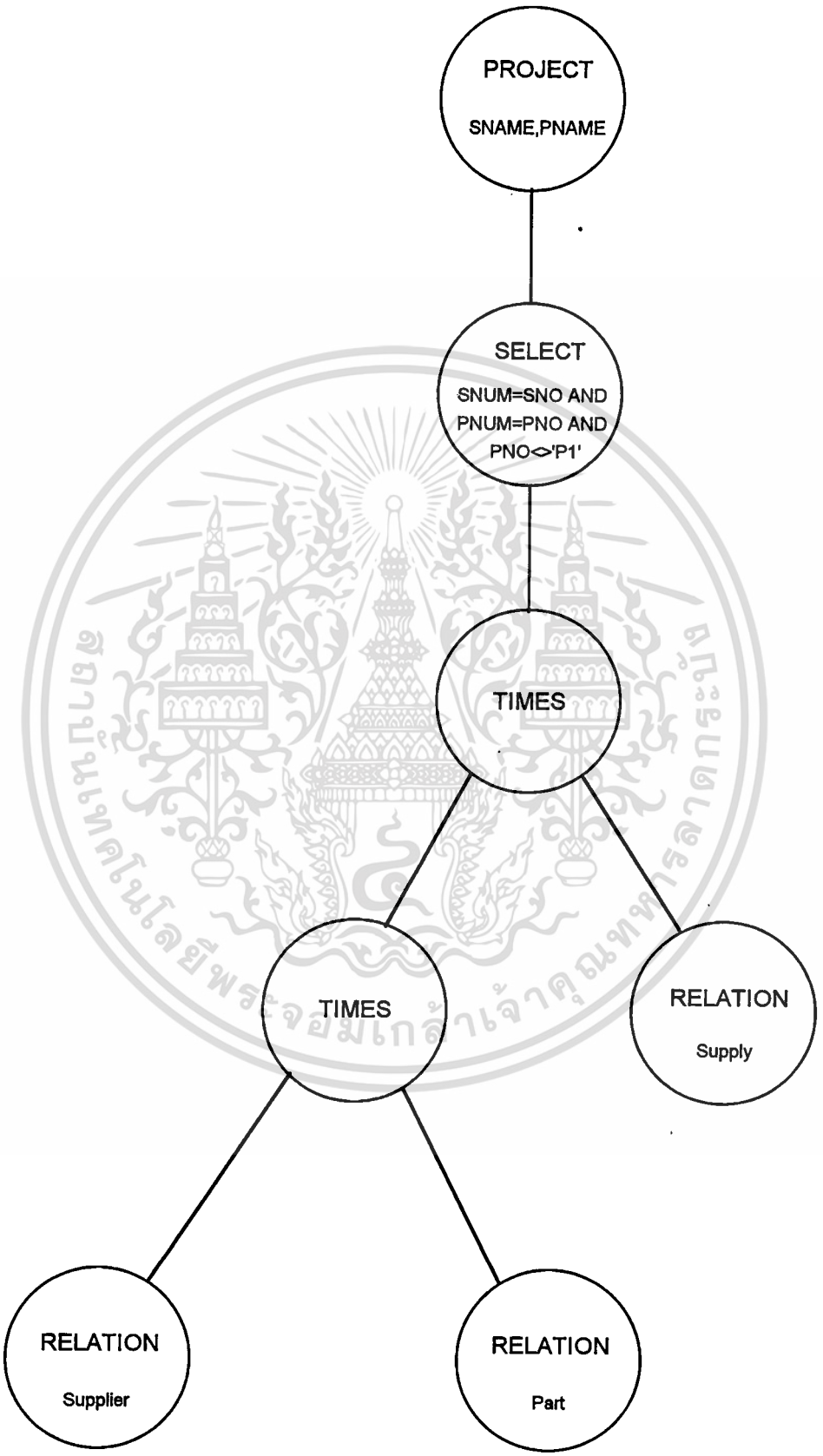
1. นำตารางทั้งหมดในลิสของ FROM CLAUSE มาทำการคูณแบบคาร์ทีเซียล
2. นำฟิลด์ทั้งหมดในลิสของ SELECT CLAUSE มาทำ PROJECT แบบพีชคณิตสัมพันธ์ ถ้าในส่วนของ SELECT CLAUSE ไม่เป็น "*"
3. นำส่วนที่อยู่ใน WHERE CLAUSE มาทำ SELECT แบบพีชคณิตสัมพันธ์



รูปที่ 5.3 รูปทั่วไปของต้นไม้พีชคณิตที่ได้จาก DDQ/II

ตัวอย่างที่ 5.7

จากตัวอย่างที่ 5.5 จะได้ต้นไม้พีชคณิตดังรูปที่ 5.4



รูปที่ 5.4 แสดงต้นไม้พีชคณิตที่ได้จากตัวอย่างที่ 5.5

บทที่ 6
สรุปและแนวทางการวิจัย

6.1 สรุป

จากการที่ระบบฐานข้อมูลถูกนำมาใช้มากขึ้น เพื่อช่วยให้การจัดเก็บและการค้นหาข้อมูล เป็นไปอย่างถูกต้องและรวดเร็ว รวมทั้งการบำรุงรักษาข้อมูลให้อยู่ในสภาพที่พร้อมจะใช้ได้เสมอ ระบบจัดการฐานข้อมูลจึงได้รับการพัฒนาอย่างต่อเนื่องตลอดมา จากระบบฐานข้อมูลแบบรวมที่สามารถทำงานได้โดยอิสระ จนถึงระบบจัดการฐานข้อมูลแบบกระจาย ที่เชื่อมต่อระบบฐานข้อมูลแบบรวมให้สามารถทำงานประสานกันได้ เสมือนกับเป็นระบบฐานข้อมูลเดียวกัน

ระบบประมวลผลคำถามแบบกระจาย DDQ/II ได้พัฒนามาจากระบบประมวลผลคำถามแบบกระจาย DDQ/1 ซึ่งสามารถเชื่อมต่อระบบจัดการฐานข้อมูลแบบรวมหลายระบบ ที่ทำงานได้โดยอิสระในโครงข่ายคอมพิวเตอร์เดียวกัน ให้สามารถทำงานร่วมกันเป็นระบบฐานข้อมูแบบกระจายได้ โดยไม่มีการแก้ไขเปลี่ยนแปลงระบบฐานข้อมูลเดิม ให้สามารถรับคำถามจากผู้ใช้งานเป็นภาษา SQL ได้ จากเดิมที่รับคำถามจากผู้ใช้งานเป็นภาษาพีชคณิตสัมพันธ์ซึ่งไม่ค่อยเป็นที่คุ้นเคยกับผู้ใช้นัก แต่ภาษาที่ใช้ในระบบ DDQ/II นี้เป็นเพียงแค่ส่วนง่ายๆ ของภาษา SQL เท่านั้น ซึ่งมี ทั้งหมด 3 ส่วนด้วยกัน คือ ส่วน SELECT ส่วน FROM และส่วน WHERE ซึ่งรูปแบบโดยทั่วๆไปคล้ายกับภาษา SQL แต่ไม่สามารถใช้คำถามย่อย ฟังก์ชัน และโอเปอเรเตอร์บางตัวของภาษา SQL ได้

จากการที่ได้ทำการทดลองใช้ระบบประมวลผลคำถาม DDQ/II ที่ได้พัฒนาขึ้นนี้ สามารถทำงานได้ตามวัตถุประสงค์ที่ได้ตั้งไว้ คือ สามารถรับคำถามเป็นภาษาที่ใกล้เคียงกับภาษา SQL ซึ่งเป็นที่คุ้นเคยกับผู้ใช้งาน และทำงานร่วมกับระบบประมวลผลคำถามแบบกระจาย DDQ/1 โดยไม่ต้องมีการแก้ไขระบบ DDQ/1 ซึ่งสามารถเชื่อมต่อระบบจัดการฐานข้อมูลแบบรวมหลายระบบ ที่ทำงานได้โดยอิสระในโครงข่ายคอมพิวเตอร์เดียวกัน ให้สามารถทำงานร่วมกันเป็นระบบฐานข้อมูลแบบกระจายได้ ซึ่งผลการทดลองใช้เป็นที่น่าพอใจ

6.2 แนวทางการวิจัยและพัฒนา

6.2.1 วิธีในการคำนวณเพื่อให้ใช้เวลาในการประมวลผลน้อยที่สุด โดยการกำหนดสถานที่ประมวลผลของระบบประมวลผลแบบกระจาย DDQ/1 จะคำนวณจากจำนวนแถวของข้อมูลที่จะต้องส่ง และเลือกประมวลผลในสถานที่ที่มีการส่งจำนวนแถวของข้อมูลน้อยที่สุด แต่ในความเป็นจริง จำนวนแถวของข้อมูลไม่สามารถวัดขนาดที่แท้จริงของข้อมูลที่จะต้องส่งได้ เพราะในแต่ละแถวของข้อมูล อาจจะมีขนาดของข้อมูลไม่เท่ากันก็ได้ จึงควรมีการพัฒนาในส่วนนี้ให้คำนวณจากขนาดที่แท้จริงของข้อมูล

6.2.2 ระบบประมวลผลคำถามแบบกระจาย DDQ/1 ถูกสร้างเพื่อรองรับภาษาพีชคณิตสัมพันธ์เท่านั้น ดังนั้น การพัฒนาให้ใช้กับภาษา SQL จึงไม่สามารถทำได้อย่างเต็มรูปแบบ เช่น ฟังก์ชัน SUM, AVG, COUNT,

MIN, MAX, การจัดลำดับและการจัดกลุ่มของข้อมูลไม่สามารถทำได้ในระบบ DDQ/1 จึงควรมีการพัฒนาในส่วนนี้เพื่อให้สามารถใช้ภาษา SQL ได้เต็มรูปแบบ

6.2.3 ระบบประมวลผลคำถามแบบกระจาย DDQ/II สามารถใช้ภาษา SQL แต่เพียงง่ายๆ เท่านั้น ไม่สามารถใช้คำถามย่อย และโอเปอเรเตอร์บางตัวได้ เช่น NOT จึงควรพัฒนาในส่วนนี้ให้สามารถใช้ภาษา SQL ได้เต็มรูปแบบ

6.2.4 ระบบประมวลผลคำถามแบบกระจายนี้ สามารถทำได้เพียงแค่การประมวลผลคำถามเท่านั้น ไม่สามารถเปลี่ยนแปลงหรือแก้ไขข้อมูลได้ จึงควรพัฒนาในส่วนนี้ให้สามารถแก้ไขข้อมูลได้

ภาคผนวก ก
ตัวอย่างโปรแกรมที่ติดต่อสื่อสารด้วยโปรโตคอล TCP/IP

```
/*
 * Read "n" bytes from a descriptor.
 * Use in place of read() when fd is a stream socket.
 */

int readn(fd, ptr, nbytes)
register int        fd ;
register char       *ptr ;
register int        nbytes ;
{
        int         nleft, nread ;

        nleft = nbytes ;
        while (nleft > 0)
        {
                nread = read (fd, ptr, nleft) ;
                if ( nread < 0)
                        return (nread) ;
                else if (nread == 0)
                        break ;
                nleft -= nread ;
                ptr += nread ;
        }
        return (nbytes - nleft) ;
}
/*
 * Write "n" bytes to a descriptor.
 * Use in place of write() when fd is a stream socket.
 */

int writen( fd, ptr, nbytes)
```

```
register int        fd ;
register char       *ptr ;
register int        nbytes ;
{
        int         nleft, nwritten ;

        nleft = nbytes ;
        while(nleft > 0)
        {
                nwritten = write(fd, ptr, nleft) ;
                if (nwritten <= 0)
                return (nwritten) ;
                nleft -= nwritten ;
                ptr += nwritten ;
        }
        return (nbytes - nleft) ;
}

/*
 * Read a line from a descriptor. Read the line one byte at a time,
 * looking for the newline. We store the newline in the buffer,
 * then follow it with a null ( the same sa fgets(3)).
 * We return the number of characters up to, but not including,
 * the null (the same as strlen(3)).
 */

int readline(fd, ptr, maxlen)
register int        fd ;
register char       *ptr ;
register int        maxlen ;
{
        int         n, rc ;
        char        c ;
```

```
        for (n = 1; n < maxlen; n++)
        {
                if ( (ro = read(fd, &o, 1)) == 1)
                {
                        *ptr++ = o ;
                        if (o == '\n')
                        break ;
                }
                else if (ro == 0)
                {
                        if (n == 1)             /* EOF, no data read */
                                return (0) ;
                        else
                                break;    /* EOF, some data was read */
                }
                else
                        return (-1) ;           /* error */
        }
        *ptr = 0 ;
        return (n) ;
}

/*
 * Read a stream socket one line at a time, and write each line back
 * to the sender.
 *
 * Return when the connection is terminated.
 */

#define   MAXLINE         512

str_echo(sockfd)
int         sockfd ;
{
```

```
        int     n ;
        char    line[MAXLINE] ;

        for(;;)
        {
                n = readline (sockfd, line, MAXLINE) ;
                if ( n == 0)
                        return ;            /* connection terminated */
                else if (n < 0)
                        err_dump ("str_echo: readline error") ;
                if (writen (sockfd, line, n) != n )
                        err_dump ("str_echo: writen error") ;
        }
}

/*
 * Read the contents of the FILE *fp, write each line to the
 * stream socket (to the server process), then read a line back from
 * the socket and write it to the standard output.
 *
 * Return to caller when an EOF is encountered on the input file.
 */

#include <stdio.h>
#define  MAXLINE        512

str_cli(fp, sockfd)
register FILE   *fp ;
register int    sockfd ;
{
        int     n ;
        char    sendline[MAXLINE], recvline[MAXLINE + 1] ;

        while ( fgets (sendline, MAXLINE, fp) != NULL)
```

```
        {
                n = strlen (sendline) ;
                if (writen (sockfd, sendline, n) != n)
                        err_sys ("str_cli: writen error on socket") ;
                n = readline (sockfd, recvline, MAXLINE) ;
                if (n < 0 )
                        err_dump ("str_cli: readline error") ;
                recvline[n] = 0 ;               /* null terminate */
                fputs (recvline, stdout) ;
        }
        if (ferror (fp))
                err_sys ("str_cli: error reading file") ;
}

/* Example of client using TCP protocol. */
#include "inet.h"
#include <memory.h>
main(argc,argv)
int     argc ;
char    *argv[ ] ;
{
        int         sockfd ;
        struct sockaddr_in serv_addr ;

        pname = argv[0] ;
        memset ((char *) &serv_addr,0, sizeof (serv_addr)) ;
        serv_addr.sin_family = AF_INET ;
        serv_addr.sin_addr.s_addr = inet_addr (SERV_HOST_ADDR) ;
        serv_addr.sin_port = htons(SERV_TCP_PORT) ;
        if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
                err_sys ("client: can't open stream socket") ;
        if (connect (sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
                err_sys ("client: can't connect to server") ;
        str_cli (stdin, sockfd) ;
```

```
        close (sockfd) ;
        exit (0) ;
}

/* Example of server using TCP protocol. */

#include "inet.h"
main(argc,argv)
int     argc ;
char    *argv[ ] ;
{
        int         sockfd, newsockfd, clilen, childpid ;
        struct sockaddr_in cli_addr, serv_addr ;
        pname = argv[0] ;
        if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
                err_dump ("server: can't open stream socket") ;
        bzero ((char *) &serv_addr, sizeof (serv_addr)) ;
        serv_addr.sin_family = AF_INET ;
        serv_addr.sin_addr.s_addr = htonl (INADDR_ANY) ;
        serv_addr.sin_port = htons (SERV_TCP_PORT) ;
        if (bind (sockfd, (struct sockaddr *) &serv_addr, sizeof (serv_addr)) < 0)
                err_dump ("server: can't bind local address") ;
        listen (sockfd,5) ;
        for ( ;; )
        {
                clilen = sizeof (cli_addr) ;
                newsockfd = accept (sockfd, (struct sockaddr *) &cli_addr, &clelen) ;
                if (newsockfd < 0)
                        err_dump ("server: accept error") ;
                if ((childpid = fork ()) < 0)
                        err_dump ("server: fork error") ;
                else if (childpid == 0)
                {
                        close (sockfd) ;
```

```
                    str_echo (newsockfd) ;
                    exit (0) ;
                }
                close (newsockfd) ;
            }
        }
```

ภาคผนวก ข
ไวยากรณ์สำหรับคำสั่ง SELECT
ในภาษา SQL

MANAPULATIVE STATEMENTS

manipulative-statement	::=	select-statement
select-statement	::=	SELECT [ALL | DISTINCT] selection table-exp

QUERY EXPRESSIONS

selection	::=	scalar-exp-commalist
table-exp	::=	from-clause
		[where-clause]
		[group-by-clause]
		[having-clause]
		[order-by-clause]
from-clause	::=	FROM table-ref-commalist
table-ref	::=	table [range-variable]
where-clause	::=	WHERE search-condition
group-by-clause	::=	GROUP BY column-ref-commalist
having-clause	::=	HAVING search-condition
order-by-clause	::=	ORDER BY ordering-spec-commalist

SEARCH CONDITIONS

search-condition	::=	boolean-term
		| search-condition OR boolean-term
boolean-term	::=	boolean-factor
		| boolean-term AND boolean-factor
boolean-factor	::=	[NOT] boolean-primary
boolean-primary	::=	predicate | (search-condition)
predicate	::=	comparison-predicate

		| between-predicate
		| like-predicate
		| test-for-null
		| in-predicate
		| all-or-any-predicate
		| existence-test
comparison-predicate	::=	scalar-exp comparison { scalar-exp | subquery }
comparison	::=	= | <> | < | > | <= | >=
between-predicate	::=	scalar-exp [NOT] BETWEEN scalar-exp AND scalar-exp
like-predicate	::=	column-ref [NOT] LIKE atom [ESCAPE atom]
test-for-null	::=	column-ref IS [NOT] NULL
in-predicate	::=	scalar-exp [NOT] IN
		{ subquery | atom [, atom-commalist] }
all-or-any-predicate	::=	scalar-exp comparison [ALL | ANY | SOME] subquery
existence-test	::=	EXISTS subquery
subquery	::=	(SELECT [ALL | DISTINCT] selection table-exp)

SCALAR EXPRESSIONS

scalar-exp	::=	term
		| scalar-exp { + | - } term
term	::=	factor
		| term { * | / } factor
factor	::=	[+ | -] primary
primary	::=	atom
		| column-ref
		| function-ref
		| (scalar-exp)
atom	::=	literal
		| USER
function-ref	::=	COUNT (*)
		| distinct-function-ref
		| all-function-ref
distinct-function-ref	::=	{ AVG | MAX | MIN | SUM | COUNT } { DISTINCT column-ref }

all-function-ref ::= { AVG | MAX | MIN | SUM | COUNT } { [ALL] scalar-exp }

MISCELLANEOUS

table ::= base-table | view

base-table ::= [user .] identifier

view ::= [user .] identifier

user ::= authorization-identifier

column-ref ::= [column-qualifer .] column

column-qualifier ::= table | range-variable

ordering-spec ::= [integer | column-ref] [ASC | DESC]

ภาคผนวก ค
การคอมไพล์โปรแกรมโดยใช้
LEX และ YACC

ในหัวข้อนี้จะกล่าวถึง การคอมไพล์โปรแกรมด้วย LEX หรือ FLEX และ YACC โดยตัวเลือกที่จะกล่าวถึง ในกรณีที่เป็นการคอมไพล์โปรแกรมด้วย LEX จะสามารถนำไปใช้กับการคอมไพล์ด้วย FLEX ได้ ส่วนตัวเลือกเฉพาะสำหรับ FLEX อย่างเดียวจะมีการระบุไว้ตามแต่ละกรณีไป

ในการที่จะสร้างโปรแกรมแยกคำ (lexer หรือ lexical analyzer หรือ scanner) เราต้องคอมไพล์ source file ด้วยโปรแกรม lex (flex) ก่อน เพื่อให้ได้โมดูลในภาษา C แล้วจึงนำไปเชื่อมต่อ (link) ร่วมกับโมดูลอื่นๆ ต่อไป ในการพรีคอมไพล์โปรแกรมด้วย lex (flex) ไฟล์ผลลัพท์ในภาษา C จะมีชื่อว่า lex.yy.c เสมอ ดังนั้น ในกรณี ที่มีโมดูลที่ตรวจแยกคำหลายๆ ตัว จะต้องทำการเปลี่ยนชื่อไฟล์ก่อนที่จะทำการพรีคอมไพล์โปรแกรมที่ต้องการ

คำสั่งต่อไปนี้จะสร้าง lex.yy.c จากไฟล์ lex.l

```
$ lex lex.l
```

ในกรณีที่ต้องการสร้างเอาท์พุทไฟล์ชื่ออื่นในคำสั่งเดียว สามารถทำได้โดยใช้ทางเลือก -t ซึ่งจะสร้างเอาท์พุทไฟล์บน stdout ซึ่งเราสามารถที่จะทำการย้ายทิศทาง (redirect) ไปบนชื่อไฟล์ที่เราต้องการได้ เช่น

```
$ lex -t lex.l > lex.c
```

ในกรณีที่โปรแกรมของเรามีเพียงโมดูลเดียวคือ โมดูลจาก lex เราสามารถที่จะทำการ link ไฟล์ โดยใช้ทางเลือก -ll ของ C-Compiler : cc ซึ่งจะทำการสร้างฟังก์ชัน main() ให้ไปเรียกใช้ yylex() ได้โดยอัตโนมัติ นอกจากนี้ จะสร้าง yyerror() ให้ด้วยในกรณีที่ไม่ได้มีการประกาศไว้ล่วงหน้า เช่น

```
$ cc lex.yy.c -ll
```

ทางเลือกพิเศษ ในกรณีที่ใช้ FLEX ได้แก่ -I ซึ่งมักใช้กับโปรแกรมที่มีการติดต่อกับผู้ใช้ ซึ่งโดยปรกติ LEX จะต้องรอสัญกรณ์ตัวถัดมา (lookahead token) เสมอ ทำให้ผู้ใช้ต้องกดตัวอักษรเพิ่มก่อนที่กฎข้อที่ผ่านมาจะสามารถผ่านการตรวจสอบได้ การใช้ทางเลือกนี้จะสามารถทำให้ FLEX รอรับสัญกรณ์ตัวถัดมาในกรณีที่จำเป็นเท่านั้น เช่น

```
$ flex -I myson.l
```

ในการใช้งานโปรแกรม YACC ร่วมกับ LEX เราสามารถที่จะเรียกใช้โปรแกรมใดก่อนก็ได้ นอกจากในกรณีที่เราใช้ header ไฟล์ ซึ่งมีการ #define สัญกรณ์ร่วมกัน เราควรที่จะเรียกใช้ YACC ก่อน โดยทางเลือกที่ใช้จะเป็น -d ซึ่งจะสร้าง header ไฟล์ ที่มีชื่อว่า y.tab.h ซึ่งจะมีนิยามของสัญกรณ์ที่อาจใช้ร่วมกันกับ LEX ได้ ส่วนไฟล์ที่เป็นโมดูลตรวจสอบไวยากรณ์ (parser) จะมีชื่อว่า y.tab.c เช่น

```
$ yacc -d grammar.y
```

หลังจากนั้นจึงเรียกใช้ LEX เช่น

```
$ lex lex.l
```

หลังจากนั้นจึงทำการคอมไพล์และลิงก์ร่วมกันด้วย cc เช่น

```
$ cc lex.yy.c y.tab.c -ly -ll
```

ในกรณีนี้จะมีการเรียกใช้ไลบรารี่ไฟล์ ของ YACC และ LEX ด้วยโดย -ly จะเรียกใช้ไลบรารี่ของ YACC ก่อน ซึ่งจะทำให้โปรแกรมของเราใช้ main() ของ YACC ซึ่งจะเรียกใช้ yyparse() ซึ่งจะไปเรียกใช้ yylex() อีกทีหนึ่ง ข้อให้สังเกตว่า จะต้องเรียกใช้ -ly ก่อน -ll เท่านั้น

เราอาจไม่ใช้ไลบรารี่ของ LEX กับ YACC ในการคอมไพล์และลิงก์ก็ได้ แต่เราจะต้องเรียกใช้ yyparse() ด้วยตัวเอง และจะต้องสร้าง yyerror() ด้วย

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จลงได้ด้วยดี ก็เพราะได้รับความกรุณา และความเอาใจใส่จาก ผู้ช่วยศาสตราจารย์ ดร. ศุภมิตร จิตตะยโศธร ที่ได้ให้คำแนะนำ และเป็นผู้สร้างกำลังใจให้ผู้จัดทำตลอดมา ผู้จัดทำซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ บุคคลากรแผนกสารสนเทศ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ที่ได้ให้ความช่วยเหลือผู้จัดทำ ในการทำปริญญานิพนธ์ เป็นอย่างดี

ขอขอบคุณ คุณสมชิด ศิริผลหลาย ที่ได้ให้คำปรึกษา และเอกสารเกี่ยวกับระบบประมวลผลคำถามแบบกระจาย DDQ/1

ขอขอบคุณ เพื่อนๆ ที่ให้ความช่วยเหลือ และคำแนะนำผู้จัดทำมาโดยตลอด

สุดท้ายนี้ ผู้จัดทำขอขอบพระคุณบิดาและมารดา ที่ได้ให้การเลี้ยงดูผู้จัดทำมาโดยตลอด อีกทั้งเป็นบุคคลที่สร้างกำลังใจแก่ผู้จัดทำเสมอมา

ผู้จัดทำ

เอกสารอ้างอิง

[1] C. J. Date, "A Guide to Ingres," Addison-Wesley publishing company, 1987 :

[2] Ulka Rodgers, "Unix database management system," Prentice-Hall, 1990 :

[3] ORACLE, Company. "SQL*NET TCP/IP USER 'S GUIDE," Oracle corporation, 1987.

[4] Sitansu S. Mittra, " Principles of Relational Database System," Prentice-Hall, 1991 :69-94,116-129.

[5] Stefano Ceri and Giuseppe Pelagatti, "Distribute database principles & systems," MoGraw-Hill, 1984 :1-172.

[6] U. S. Chakravarthy, D. H. Fishman and J. Minker, "Semantic Query Optimization in Expert Systems and Database Systems," Expert Database Systems, The Benjamin/ Cummings Publishing Company, Ino. ,1986 :659-674.

[7] Larry Kersohberg ,Peter D. Ting and S. Bing Yao, "Query Optimization in Star Computer," ACM Transactions on Database Systems, Vol. 7, No. 4, December 1982 :678-711.

[8] S. Bing Yao, "Optimization of Query Evaluation Algorithms," ACM Transactions on Database Systems, Vlo. 4, No. 2, June 1979 :133-155.

[9] Stefano Ceri and Giuseppe Pelagatti, "Allocation of Operations in Distributed Database Access," IEEE Transactions on Computers, Vol. c-31, No. 2, February 1982 :119-129.

[10] C. J. Date, " An Introduction to Database Systems Volume 1," Fourth Edition, Addison-Wesley, World Student Series, 1986 :12-15

[11] E. F. Codd, " Relational Completeness of Data Base Sublanguage," Data Base System, Courant Computer Science Symposia Series, Vol. 6, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[12] Ralph B. Bisland, " Database Management Developing Application Systems Using Oracle," Prentice-Hall International Editions, Prentice-Hall , Inc., A Division of Simon & Schuster, Englewood cliffs, NJ ,1989 :53-86

[13] Carolyn J. Hursch and Jack L. Hursch, "INGRES SQL Developer's Guide," Windcrest/McGraw-Hill , 1992 :111-120.

[14] J. D. Ullman, " Principles of Database Systems, " 2nd ed., Computer science Press, 1983.

[15] M. Tamer Özsu and Patrick Valduriez, "Principles of Distributed Database Systems," Prentice-Hall International Editions, Prentice-Hall , Inc., A Division of Simon & Schuster, Englewood cliffs, NJ ,1991 :66-257,425-445.

[16] Esen Ozkarahan, "Database Maohines and Database Management," Prentice-Hall International Editions, Prentice-Hall , Inc., A Division of Simon & Sohuster, Englewood oliffs, NJ ,1986 :408-453.

[17] Henry F. Korth and Abraham Sibersohatz, "Database System Concepts," MoGraw-Hill International Editions, MoGraw-Hill , Inc., 1986 :301-325, 403-449.

[18] W. Riohard Stevens, "Unix Network Programming," Prentice-Hall International Editions, Prentice-Hall , Inc., A Division of Simon & Sohuster, Englewood oliffs, NJ ,1991 :1-420.

[19] AT&T, "Unix System V Programmer' s reference manual AT&T," Prentice-Hall , Inc., A Division of Simon & Sohuster, Englewood oliffs, NJ ,1987.

[20] M. Negri and G. Pelagatti, "Distributive Join: A new algorithm for joining relations," ACM Transactions on Database Systems, Vol. 16, No. 4, December 1991, :655-669.

[21] G. M. Nijssen and T. A. Halpin, "Conceptual Sohema and Relational Database Design," Prentice-Hall of Australia Pty Ltd, 1989 :1-303.

[22] G. M. Nijssen and E. D. Falkenberg, "Introduotion to IBM SQL," Nijssen Data, Bases Pty. Ltd., 1984 :1-193.

[23] "Networking Software Paokage",Nixdorf Computer AG ,W-Germany ,1990 :1-1 - 5-6 .

[24] ORACLE, Company. "PRO*C User's Guide," Oraole oorporation, 1987 :1-258.

[25] ORACLE, Company. "SQL*Plus Reference Guide," Oraole oorporation, 1987 : 1-1 - 2-111.

[26] ORACLE, Company. "SQL*NET User's Guide," Oraole oorporation, 1986.

[27] E . F . CODD, "The Relational Model for Database Management Version 2," Addision-Wesley Publishing Company, 1990 :351-430.

[28] Relational Teohnology Ino.,"Destributed INGRES for the UNIX & VMS Operating Systems," Relational Teohnology Ino., Alameda, California U.S.A., 1989:1-1 - 8-20.

[29] AT&T, "UNIX® System V Release 4 - Programmer's Guide : ANSI C and Programming Support Tools," Prentice-Hall , Ino., A Division of Simon & Sohuster, Englewood oliffs, NJ ,1990.

[30] AT&T 1989a, "UNIX System V Release 3.2 - Network Programmer's Guide," Prentice-Hall , Ino., A Division of Simon & Sohuster, Englewood oliffs, NJ , 1989.

[31] Halsall F. 1992, "Data Communioations, Computer Networks and Open Systems," 3rd edition, Addison-Wesley, Workingham, England, 1992.

[32] Date, C. J., "A Guide to the SQL standard," 2nd edition, Addison-Wesley, Reading Massaohusetts, 1989.

[33] สมชิต ศิริผลหลาย, วิทยานิพนธ์เรื่อง "ระบบประมวลผลคำถาม สำหรับฐานข้อมูลแบบกระจายในโครงข่ายคอมพิวเตอร์ TCP/IP", คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง, 2536.