Object-Oriented Database

ฐานข้อมูลเชิงวัตถุ

โดย

นาย มณฑล	ขมหวาน	38013285
นาย สมศักดิ์	ธนาวุฒิวร	38013298

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

Object-Oriented Database

ฐานข้อมูลเชิงวัตถุ

โดย

นาย มณฑล ขมหวาน 38013285

นาย สมศักดิ์ ธนาวุฒิวร 38013298

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ฐานข้อมูลเชิงวัตถุ (Object-Oriented Database)

ผู้จัดทำ

1. นายมณฑล ขมหวาน 38013285
2. นายสมศักดิ์ ธนาวุฒิวร 38013298

(.....)

รศ.ดร. ศุภมิตร จิตตะยโศธร

อาจารย์ที่ปรึกษา

ฐานข้อมูลเชิงวัตถุ

นายมณฑล ขมหวาน
นายสมศักดิ์ ธนาวุฒิวร

อาจารย์ที่ปรึกษา
รศ.ดร.ศุภมิตร จิตตะยโศธร
ปีการศึกษา 2540

บทคัดย่อ

ในความเป็นจริงนั้น วัตถุสิ่งของต่างๆ ส่วนใหญ่จะมีโครงสร้างสลับซับซ้อน ซึ่งการออกแบบฐานข้อมูลเชิงสัมพันธ์ (RDB : Relational Database) ไม่เหมาะสมกับข้อมูลที่มีโครงสร้างสลับซับซ้อน (Complex Object) จึงได้นำความคิดของออบเจกต์ (Object) มาใช้เพื่อแก้ไขข้อบกพร่องดังกล่าว และได้ใช้ฐานข้อมูลชื่อ POSTGRES มาทำการทดสอบแนวความคิดเกี่ยวกับการออกแบบฐานข้อมูลเชิงวัตถุและจำลองระบบงานห้องสมุด

OBJECT-ORIENTED DATABASE

Mr.Monthon Khomwan

Mr.Somsak Thanawutiworn

Thesis Advisor

Assoc.Prof.Dr.Suphamit Chittayasotorn

1997

ABSTRACT

In the real world , most objects are complex object which Relational Database is not suitable complex data. Therefore , the concept of object has been used to solve the weak point.

This project presents how to create prototype system of library implement with POSTGRES DBMS and tests it for characteristics of object-oriented concept .

สารบัญ

	หน้าที่
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญของปัญหา	1
1.2 วัตถุประสงค์	1
บทที่ 2 ทฤษฎีและหลักการ	2
2.1 แนวความคิดเชิงวัตถุ (Object-Oriented Concept)	2
2.1.1 ออบเจกต์ (Object)	2
2.1.2 โพลิมอร์ฟิซึม (Polymorphism)	2
2.1.3 อินเฮอริแทนซ์ (Inheritance)	3
2.2 OODB (Object-Oriented Database)	3
2.2.1 ออบเจกต์ (Object)	4
2.2.1.1 Schema Evolution and Reuseability	5
2.2.1.2 Concurrency	5
2.2.2 Data Abstraction	5
2.2.2.1 Definition ของ Abstract Data Type (ADT)	6
2.2.2.2 Data Abstraction และ Strong Typing	6
2.2.2.3 การ Implement ADT ด้วยคลาส	7
2.2.3 การสืบทอดคุณสมบัติ (Inheritance)	7
2.2.3.1 การสืบทอดคุณสมบัติเพื่อเพิ่มความสามารถ	9
2.2.3.2 การสืบทอดคุณสมบัติและโพลิมอร์ฟิซึม	10
2.3 ระบบฐานข้อมูลแบบสัมพันธ์(Relational Database)	10
2.3.1 โครงสร้างข้อมูลแบบสัมพันธ์	11
2.3.1.1 ตัวอย่างตารางความสัมพันธ์	11
2.3.1.2 รีเลชัน	12
2.3.1.3 ความถูกต้องของข้อมูล	13
2.3.1.4 ทฤษฎีของระบบฐานข้อมูลเชิงสัมพันธ์	15
2.4 Object-Relational Database (ORDB)	18
2.5 Entity Relationship (ER) โมเดล	18
2.5.1 การแปลงความสัมพันธ์จาก ER โมเดลเป็นรีเลชัน	19
2.6 Extend - Entity Relationship (EER) โมเดล	20
2.6.1 ซับคลาสและซุปเปอร์คลาส	20

2.6.2 การแปลงจาก EER โมเดลเป็นรีเลชัน 21
บทที่ 3 POSTGRES DBMS 22
3.1 ประวัติความเป็นมา 22
3.2 สถาปัตยกรรมของ POSTGRES DBMS 22
3.3 คุณสมบัติที่เพิ่มขึ้นของ SQL ใน POSTGRES DBMS 23
3.4 ความยืดหยุ่น (Extensibitity) 24
3.4.1 ฟังก์ชัน (Function) 25
3.4.2 แบบชนิดข้อมูล (Type) 26
3.4.3 โอเปอร์เรเตอร์ (Operators) 28
3.4.4 แอกกริเกชัน (Aggregates) 30
3.5 LIBPQ ไลบารี 30
3.5.1 การควบคุมและการกำหนดค่าเริ่มต้น 31
3.5.2 ฟังก์ชันในการติดต่อระบบฐานข้อมูล 31
3.5.3 ชุดคำสั่งที่ใช้ในการปฏิบัติตามคิวรี (Query) 33
บทที่ 4 การสร้างแบบจำลองโดยใช้ POSTGRES DBMS 37
4.1 ออกแบบฐานข้อมูลระบบห้องสมุดโดยใช้ EER โมเดล 37
4.1.1 การแปลงจาก EER โมเดลเป็นรีเลชัน 38
4.2 แสดงคุณสมบัติ Object-Oriented ของ POSTGRES DBMS 39
4.2.1 การถ่ายทอดคุณสมบัติ (Inheritance) 39
4.2.2 คุณสมบัติโพลิมอร์ฟิซึม (Polymorphism) 41
บทที่ 5 บทวิจารณ์และสรุป 44
5.1 สรุปผลการทำโครงงาน 44
5.2 ปัญหาที่เกิดขึ้น 44
5.3 แนวทางการนำไปพัฒนาต่อ 44
ภาคผนวก ซอร์ดโค้ดของระบบงานห้องสมุด 45
กิตติกรรมประกาศ 63
บรรณานุกรม 64

สารบัญตาราง

	หน้าที่
ตารางที่ 2.1 แสดงความสัมพันธ์ระหว่างคีย์หลักของ R1 กับคีย์อื่นๆของ R1	16
ตารางที่ 2.2 แสดงความสัมพันธ์ระหว่างคีย์หลักของ R2 กับคีย์นอกของ R1	16
ตารางที่ 2.3 แสดงความสัมพันธ์ระหว่าง X กับ Y (Composite Key)	17
ตารางที่ 2.4 แสดงความสัมพันธ์ระหว่าง X กับ Y (Full Functionally Dependent)	17
ตารางที่ 3.1 แสดงแพล็ตฟอร์มที่สามารถติดตั้ง	22

สารบัญรูปภาพ

หน้าที่

รูปที่ 2.1 แสดงความสัมพันธ์แบบอินเฮอริแทนซ์ 3

รูปที่ 2.2 แสดงการสืบทอดคุณสมบัติแบบลำดับชั้น 8

รูปที่ 2.3 แสดงตารางความสัมพันธ์ข้อมูล Supplier 12

รูปที่ 2.4 แสดงตัวอย่างตารางความสัมพันธ์ที่มีและไม่กลุ่มซ้ำ 13

รูปที่ 2.5 แสดงความสัมพันธ์แบบหนึ่งต่อหนึ่ง (1:1) 14

รูปที่ 2.6 แสดงความสัมพันธ์แบบหนึ่งต่อกลุ่ม (1:n) 14

รูปที่ 2.7 แสดงความสัมพันธ์แบบกลุ่มต่อกลุ่ม (m:n) 15

รูปที่ 2.8 แสดงการทำงานของ RDB 18

รูปที่ 2.9 แสดงความสัมพันธ์ระหว่างซุปคลาสและซับคลาส 20

รูปที่ 3.1 แสดงเครื่องลูกข่ายร้องขอการติดต่อไป Postmaster 23

รูปที่ 3.2 แสดง Postmaster สร้าง Backend Server 23

รูปที่ 3.3 แสดงเครื่องลูกข่ายติดต่อกับ Back Server 23

รูปที่ 4.1 EER โมเดลของระบบห้องสมุด

รูปที่ 4.1.ก แสดงส่วนของผู้ใช้บริการ 37

รูปที่ 4.1.ข แสดงส่วนของสิ่งพิมพ์ 37

รูปที่ 4.1.ค แสดงส่วนของความสัมพันธ์ของผู้ใช้และสิ่งพิมพ์ 38

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบัน Management Information System (MIS) ได้เข้ามามีบทบาท ในวงการธุรกิจเพิ่มมากขึ้น ระบบฐานข้อมูล ก็เป็นส่วนหนึ่งในระบบ MIS ปัจจุบันโครงสร้างฐานข้อมูลที่นิยมใช้ในขณะนี้ ได้แก่ แบบจำลองเชิงสัมพันธ์ (Relational Model) แต่ถึงอย่างไรแบบจำลองเชิงสัมพันธ์ก็ยังมีข้อจำกัดอยู่ในหลายๆด้าน เช่น แบบชนิดข้อมูลที่ใช้ยังคงเป็นแบบง่ายๆ ไม่สามารถรองรับชนิดข้อมูลที่มีความซับซ้อนได้ เป็นต้น

ดังนั้นจึงได้มีการคิดค้น โครงสร้างฐานข้อมูลขึ้นใหม่ เพื่อแก้ปัญหาบางประการที่แบบจำลองเชิงสัมพันธ์ไม่สามารถทำได้ ซึ่งก็คือ แบบจำลองเชิงวัตถุ (Object-Oriented Model) ซึ่งพัฒนาขึ้นมาสำหรับการแก้ไขปัญหาบางประการของแบบจำลองเชิงสัมพันธ์ได้ ซึ่งเป็นจุดที่หน้าสนใจในการศึกษาแบบจำลองเชิงวัตถุนี้ว่าสามารถนำมาประยุกต์ใช้งานให้เหมาะสมกับความซับซ้อนของชนิดข้อมูลในปัจจุบันได้อย่างไร

1.2 วัตถุประสงค์

การทำโครงงานนี้มีวัตถุประสงค์ 3 ประการ คือ

1. ศึกษาหลักการของแบบจำลองเชิงวัตถุ (Object-Oriented Model)
2. ศึกษา Database Management System (DBMS) ซึ่งมีคุณสมบัติ Object-Relational Model ที่ชื่อ POSTGRES DBMS ว่ามีคุณสมบัติตามทฤษฎีของ Object-Oriented Model มากแค่ไหน
3. สร้างระบบงานห้องสมุด โดยใช้ POSTGRES DBMS เพื่อพิสูจน์ทฤษฎีของ Object-Oriented Model

บทที่ 2
ทฤษฎี

2.1 แนวความคิดเชิงวัตถุ (Object-Oriented Concept)

การเขียนโปรแกรมด้วยวิธี Object-Oriented นั้น ได้นำวิธีการเขียนโปรแกรมแบบโครงสร้างมารวมกับแนวคิดใหม่ที่จะมองปัญหาต่างๆ แยกออกเป็นกลุ่มย่อยๆ ที่มีความสัมพันธ์กัน และใช้ภาษาคอมพิวเตอร์มาเขียนให้ปัญหากลุ่มย่อยนั้นเป็นหน่วยที่เรียกว่า ออบเจกต์ (Objects) สิ่งที่ควรรู้จักเกี่ยวกับการเขียนโปรแกรมด้วยวิธี Object-Oriented ก็คือออบเจกต์ (Object), โพลีมอร์ฟีซึม (Polymorphism), อินเฮอริแทนซ์ (Inheritance)

2.1.1 ออบเจกต์ (Object)

ออบเจกต์ (Object) เป็นตัวแปรคลาส (Class variable) ที่มีลักษณะเป็นโมดูล ซึ่งประกอบด้วยตัวแปรชนิดข้อมูลต่างๆ ที่สัมพันธ์กันและฟังก์ชันต่าง ๆ ดูคล้าย ๆ กับ ข้อมูลชนิดโครงสร้าง สามารถนำออบเจกต์ต่างๆ มาประกอบเป็นโปรแกรม สามารถแทรกออบเจกต์เข้าไปในโปรแกรมหรือนำออก จะมีผลกระทบต่อคำสั่งอื่นๆ ในโปรแกรมน้อยมาก หรือไม่มีผลกระทบเลย ทำให้การตรวจสอบข้อผิดพลาดในโปรแกรมกระทำได้ง่าย ตลอดจนความสะดวกสบายและรวดเร็วในการปรับปรุงและพัฒนาโปรแกรมและออบเจกต์ต่างๆ ที่สร้างขึ้นยังสามารถเก็บไว้ใช้งานในโปรแกรมอื่นๆ ต่อไป

ตัวแปรคลาส (Class variable) จะนำมาใช้แทนตัวแปรโครงสร้าง เนื่องจากประสิทธิภาพสูงกว่ากันมาก คลาส (Class) จะห่อหุ้มข้อมูลและฟังก์ชันไว้รวมกันมีลักษณะเป็น Encapsulation ซึ่งมีความสะดวกในการใช้งานได้ง่าย สามารถป้องกันส่วนอื่นๆ ของโปรแกรมไม่ให้เข้าถึงตัวแปรชนิดโลคอล
(local variable) ภายในคลาส (Class) สามารถใช้คีย์เวิร์ด private และ public ควบคุมการเข้าถึงสมาชิกต่างๆ ของคลาสจากส่วนอื่นๆ ของโปรแกรม

2.1.2 โพลิมอร์ฟีซึม (Polymorphism)

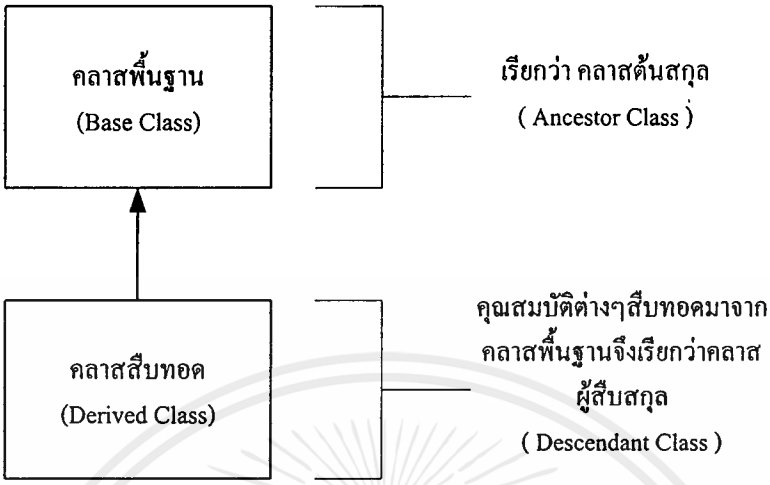
โพลิมอร์ฟีซึม คือ การสร้างฟังก์ชันที่มีหน้าที่แตกต่างกัน แต่เรียกใช้ได้ด้วยชื่อเดียวกัน เรียกว่า 'One interface, multiple methods' เพื่อให้ใช้งานได้ง่าย

โพลิมอร์ฟีซึม แบ่งออกได้เป็น 2 ลักษณะ คือ

1. ช่วงการคอมไพล์โปรแกรม (Compile time) การตรวจสอบชนิดออบเจกต์ของฟังก์ชันที่ถูกเรียกมาทำงาน รวมทั้งเตรียมข้อมูลบางอย่างที่จำเป็นจะทำขณะคอมไพล์โปรแกรม การตัดสินใจใช้ฟังก์ชันใดขึ้นอยู่กับคุณสมบัติของพารามิเตอร์ของฟังก์ชันนั้น
2. ช่วงการรันโปรแกรม (Run time) การตรวจสอบนั้นจะทำขณะที่โปรแกรมกำลังทำงาน จะทำการตัดสินใจเลือกฟังก์ชันใดฟังก์ชันหนึ่งที่มีชื่อซ้ำกันในแต่ละคลาสมาใช้งาน

2.1.3 อินเฮอริแทนซ์ (Inheritance)

อินเฮอริแทนซ์ หมายถึง การที่ออบเจกต์อันหนึ่งสามารถสืบทอดคุณสมบัติจากออบเจกต์อื่นนำมาใช้งานได้



รูปที่ 2.1 แสดงความสัมพันธ์แบบอินเฮอริแทนซ์

คลาสสืบทอด (derived class) จะสืบทอดสมาชิกข้อมูล (data member) และฟังก์ชันสมาชิกจากคลาสพื้นฐาน ทำให้สามารถใช้ข้อมูลและฟังก์ชันเดิมของคลาสแม่ผสมกับข้อมูล และฟังก์ชันของคลาสลูกที่สร้างขึ้นใหม่

2.2 OODB (Object-Oriented Database)

OODB คือ ฐานข้อมูลชนิดหนึ่งที่ถูกแสดงด้วยชนิดของข้อมูลที่กำหนดขึ้น และโอเปอร์เรชั่น (Operation) ที่ถูกออกแบบมาเพื่อใช้กับชนิดของข้อมูลเหล่านั้น

OODB เป็นฐานข้อมูลที่สนับสนุนลักษณะ Object-Oriented โดยใช้ออบเจกต์เป็นพื้นฐาน OODB เกิดขึ้นมาจากการรวมกันของภาษา Object-Oriented programming และเทคโนโลยีทางด้านฐานข้อมูล ซึ่งก่อให้เกิดประโยชน์อย่างมากดังนี้

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

2. การใช้ภาษาที่เป็น Object-Oriented Programming จะได้รับ ประโยชน์จาก คุณลักษณะของฐานข้อมูล เช่น การคงอยู่ของข้อมูล (Persistent Data), Set-oriented Processing และ Transaction Management

3. สิ่งที่สำคัญที่สุด คือ งานทางด้านฐานข้อมูลที่มีโครงสร้างที่สลับซับซ้อน อาจถูกเขียนขึ้นภายใน Single OODB Programming Language

ถึงแม้ว่า Programming Language Object และฐานข้อมูลเชิงวัตถุจะมีความคล้ายคลึงกันในเรื่องของคุณสมบัติ แต่ก็มีความแตกต่างกัน ที่สำคัญหลายประการ คือ

1. ข้อมูลของฐานข้อมูลเชิงวัตถุจะต้องคงอยู่ (Persist) ตลอดภายใต้การทำงานของโปรแกรมที่สร้างมันขึ้นมา

2. งานทางด้านฐานข้อมูล หลายๆ งานต้องการความสามารถในการสร้าง (Create) และการเข้าถึงกับออบเจกต์ในแบบของ Multiple Version

3. ฐานข้อมูลที่มีความเร็ว (Active) สูงๆ เป็นฐานข้อมูลที่ใช้ในการควบคุมการจราจรทางอากาศและฐานข้อมูลที่ใช้จัดการกับ Distribution ให้มีประสิทธิภาพต้องการความสามารถที่เกี่ยวข้องกับเงื่อนไข (Condition) และการกระทำ (Action) ซึ่งการดำเนินการจะถูกทำให้สมบูรณ์ เมื่อเงื่อนไขเหมาะสมกับออบเจกต์นั้น

4. ฐานข้อมูล ต้องการความสามารถ หรือประสิทธิภาพของ Query ใน Predicate-Based บนออบเจกต์

โดยปกติแล้ว Programming Language ทั่วไปส่วนใหญ่จะไม่สนับสนุนการคงอยู่ (Persistent) ของออบเจกต์ หรือ Multiple Object Version และไม่อำนวยความสะดวกเกี่ยวกับเรื่องคอนสเทรน

สิ่งที่สำคัญใน OODB คือ

1. ออบเจกต์ (Object)
2. แบบข้อมูลชนิดนามธรรม (Abstract Data Types)
3. การสืบทอดคุณสมบัติ (Inheritance)

2.2.1 ออบเจกต์ (Object)

Object-Oriented Paradigm เป็นการรวมโค้ด (Code) และข้อมูลเข้าไว้ด้วยกัน (Encapsulation) ซึ่งเรียกว่า ออบเจกต์ ส่วนที่ทำหน้าที่ในการเชื่อมต่อออบเจกต์ต่างๆ ในระบบคือ เซ็ตของเมสเสจ

โดยทั่วไปแล้ว ออบเจกต์จะเกี่ยวข้องกับสิ่งต่างๆ ดังนี้

1. เซ็ตของตัวแปร (Variable) ที่ใช้เก็บข้อมูลสำหรับออบเจกต์ โดยที่ค่าของตัวแปรแต่ละตัวก็ถือเป็นออบเจกต์ด้วยเช่นกัน

2. เซ็ตของเมสเสจ ซึ่งออบเจกต์ตอบสนองด้วย

3. เมตธอด ซึ่งเป็นโค้ด (Code) ที่ใช้ในการสร้างแต่ละเมสเสจ

เมสเสจ หมายถึง การส่งคำร้องขอ (Request) ระหว่างออบเจกต์ โดยไม่คำนึงถึงรายละเอียดในการ Implement เพราะว่า External Interface เท่านั้นที่เป็นเซ็ตของเมสเสจ ซึ่งออบเจกต์จะตอบสนองด้วย จะมีทางเป็นไปได้ที่จะทำการปรับปรุง Definition ของเมตธอดและตัวแปรของออบเจกต์หนึ่งโดยไม่มีผลกระทบถึงออบเจกต์อื่นนอกจากนี้ยังสามารถแทนตัวแปรด้วยเมตธอดที่ใช้คำนวณค่าได้อีกด้วย

ความสามารถในการเปลี่ยนนิยามของออบเจกต์หนึ่งโดยไม่ก่อให้เกิดผลกระทบไปถึงส่วนที่เหลือของระบบเป็นข้อดีอย่างหนึ่งของ Object-Oriented Programming Paradigm

2.2.1.1 Schema Evolution and Reuseability

ในระบบการจัดการฐานข้อมูลทั่วๆ ไป (Conventional Database Management System) มีข้อจำกัดในการขยาย หรือปรับปรุงเปลี่ยนแปลงโครงสร้างข้อมูล เช่น โดยปกติฐานข้อมูลแบบเชิงสัมพันธ์ จะอนุญาตให้ทำการ สร้าง (Create) หรือลบ (Delete) รีเลชัน และเพิ่มคอลัมน์ (Column) ใหม่ เข้าไปในรีเลชันได้ ในการเปลี่ยนชนิดของโดเมนจะต้องสร้างรีเลชันที่เกี่ยวข้องกันขึ้นมาใหม่ และทำการเปลี่ยนแปลงโปรแกรม หรืองานต่างๆ ที่เรียกใช้รีเลชันนั้นใหม่ด้วย

รวมทั้งการพัฒนาโครงสร้าง (Extension และ Refinement ของโครงสร้างที่มีอยู่แล้ว) และการนำกลับมาใช้ (Reuse) ของโปรแกรมหรือฟังก์ชัน เป็นส่วนที่สำคัญของแบบจำลองข้อมูลแบบเชิงวัตถุ (Object-Oriented Data Model) โดยธรรมชาติของแบบจำลองเชิงวัตถุ (Object-Oriented Model) ได้เอื้อให้ Semantic Schema Evolution ถูกนิยาม และแก้ไข (Validate) ได้ ในขณะที่ความสามารถทางด้านการถ่ายทอดคุณสมบัติของแบบจำลองก่อให้เกิด Generic Component ซึ่งสามารถนำมาใช้งานได้ใหม่ในหลายๆ ส่วนของระบบ

2.2.1.2 Concurrency

เมื่อโปรแกรมหลายๆ โปรแกรม หรือ Terminal User หลาย ๆ คนกำลังติดต่อกับฐานข้อมูลพร้อมๆ กัน การดำเนินการที่กระทำบนฐานข้อมูล จะต้องมีการควบคุมด้วยความระมัดระวังโดยเฉพาะอย่างยิ่ง ผลกระทบที่เกิดจากขบวนการทางด้านฐานข้อมูล (หรือ Transaction) จะต้องให้ผลเหมือนกับผลที่ได้เมื่อไม่มีการประมวลผลของ Transaction อื่น (เรียกว่า Serializable) และในระบบการจัดการฐานข้อมูลส่วนใหญ่ ก็ได้มีข้อจำกัดมากมายบน Transaction เพื่อเป็นการประกันถึง Serializability

ในการป้องกัน Transaction หนึ่งๆ ที่กำลังอ่านหรือแก้ไขข้อมูลอยู่ จากการแก้ไขของ Transaction อื่นๆ อาจทำได้โดยใช้ Locking Mechanism อย่างไรก็ตาม การ Lock ทำให้เกิดค่าใช้จ่ายในการบำรุงรักษาระบบ และอาจก่อให้เกิดสภาวะที่ลำบากขึ้น โดยเฉพาะอย่างยิ่ง Item ซึ่งมีขนาดใหญ่ (เช่นรีเลชันทั้งหมด) ดังนั้น ได้มีทางเลือกใหม่ๆ เพื่อใช้แทนการ Lock เช่นTimestamping และ Optimistic Concurrency Control

ฐานข้อมูลเชิงวัตถุ มีความสามารถในด้าน การใช้งานพร้อมๆ กัน (Concurrency) มากกว่าระบบธรรมดา และจากความจริงที่ว่า ในระบบเชิงวัตถุ การดำเนินการบนฐานข้อมูลไม่ง่ายที่จะทำการอ่านหรือเขียน แต่ก็มีการสืบทอดของ Semantic อยู่มากเหมือนกัน นอกจากนี้ ระบบยังสามารถใช้ประโยชน์จาก Semantic Scheduling Transaction ได้อีกด้วย

2.2.2 Data Abstraction

เมสเสจ ที่ส่งเพื่อเรียกใช้ การดำเนินการ หรือฟังก์ชันบนออบเจกต์ สนับสนุนหลักการสำคัญในโปรแกรมโครงสร้างที่เรียกว่า ดาต้าแอบสแทรกชั่น (Data Abstraction) หลักการดังกล่าวได้แก่ โปรแกรมยูนิต (Program unit) จะไม่ทำการที่เกี่ยวกับการ Implement และการแสดงลักษณะภายในของชนิดข้อมูลที่มันใช้ ด้วยวิธีนี้ มีทางเป็นไปได้ที่จะเปลี่ยนการ Implement ของชนิดข้อมูลโดยไม่ต้องเปลี่ยนโปรแกรม ดังนั้น วัตถุประสงค์ของ Abstraction ในการเขียนโปรแกรมก็คือ การแยกพฤติกรรม

(Behaviour) ออกจากการ Implement ทำให้การออกแบบซอฟต์แวร์ ซึ่งมีขนาดใหญ่ถูกกระทำได้ง่าย เพราะ Abstraction ยอมให้มีการแยก และพัฒนาส่วนต่างๆ ของระบบได้อย่างอิสระ

กลไกง่ายๆ เกี่ยวกับ Programming Abstraction คือ Procedure อย่างไรก็ตาม สำหรับระบบที่ใช้ซอฟต์แวร์ขนาดใหญ่ ต้องการ Abstraction Mechanism ที่มีความสามารถมากขึ้น ซึ่งก่อให้เกิดแนวความคิดของแบบข้อมูลชนิดนามธรรม (ADT) ขึ้น

แนวความคิดพื้นฐานของแบบชนิดข้อมูลนามธรรม คือ การแยกภาพการมองของผู้ใช้ออกจากรายละเอียดของการ Implement

2.2.2.1 Definition ของ Abstract Data Type (ADT)

ADT กำหนดคลาสของออบเจกต์ (ซึ่งมีโครงสร้างข้อมูลนามธรรมเดียวกัน) โดยใช้การดำเนินการและคุณสมบัติของการดำเนินการในคลาสนั่นๆ เป็นตัวกำหนด

นั่นคือ ADT จะกำหนดคลาสของออบเจกต์ ซึ่งสามารถจัดการได้โดยใช้เซ็ตของการดำเนินการที่กำหนดให้เท่านั้น

ADT มีบทบาทต่อแอพพลิเคชั่นทางด้านฐานข้อมูลอยู่ 3 ประการ คือ

1. ADT จะถูกใช้เพื่อขยาย Typing System ของภาษา, เพื่อเตรียมชนิดข้อมูลของแอตทริบิวต์ ให้มีความซับซ้อนมากขึ้น เช่น Data, Time, Colour, Name ฯลฯ ซึ่งชนิดข้อมูลเหล่านี้ไม่ใช่ Built-in Type

2. ADT อาจถูกสร้างขึ้น เพื่อแสดง Database Object ในระดับสูง ในกรณีนี้ เราจะใช้ ADT เพื่อแสดง Time-Evolving Entity ซึ่งตรงกันข้ามกับ Inert Data โดย ADT จะจัดเตรียมความสะดวกในการ Implement Integrity คอนสเทรน และ Trig procedure ซึ่ง Enforce Behaviour ของออบเจกต์ที่ Runtime

3. ADT อาจแสดงความสัมพันธ์ระหว่าง Entity ผ่านทาง Aggregation และ Generalifzation Abstraction

- Aggregation เป็น Abstraction ซึ่งมีความสัมพันธ์ระหว่างออบเจกต์ ถูกแสดงโดย Higher-level Aggregate Object (หรือชนิดข้อมูล)
- Generalization เป็น Abstraction ซึ่งเซ็ตของออบเจกต์ที่มีคุณสมบัติคล้ายคลึงกัน จะถูกแสดงโดย Generic Object

2.2.2.2 Data Abstraction และ Strong Typing

ภาษาประเภท Strong Type ได้ให้ความสามารถในการกำหนดชนิดข้อมูลที่เข้ากันได้กับทุกๆ นิพจน์ (Expression) ที่แสดงค่าซึ่งได้จาก Static Program ที่ช่วงเวลาคอมไพล์ และ Strongly Typed Programming Language ไม่ต้องกำหนดชนิดของนิพจน์ที่ช่วงเวลาคอมไพล์ ซึ่งมีความเข้ากันได้กับตัวถูกดำเนินการ (Operand) หรือตัวดำเนินการ (Operator)

การเตรียมชนิดข้อมูลหลายๆ ชนิดโดยใช้ Strong Type Checking เป็นแนวความคิดของภาษาชุดคำสั่ง ที่มีวัตถุประสงค์เพื่อทำให้เกิดผลดีของ Software Integrity และการบำรุงรักษาสำหรับงานทางด้านฐานข้อมูล โปรแกรมเมอร์ควรจะสามารถกำหนดคุณสมบัติของข้อมูลของออบเจกต์ และกฎของภาษา

ควรจะรับประกันได้ว่า คุณสมบัติเหล่านี้ถูกบำรุงรักษาตลอดทั้งโปรแกรม คุณสมบัติที่เป็นของหลายๆ ออบเจกต์ควรจะแยกออกมาเป็น Single Type Declarationและอ้างถึงคุณสมบัติเหล่านี้โดยใช้ชื่อถ้าระหว่างการกระทำบำรุงรักษาระบบทำให้คุณสมบัติมีการเปลี่ยนแปลงก็ให้แก้ไขเฉพาะที่ประกาศนี้เท่านั้น ไม่ต้องตามไปแก้ไขในทุกๆ Object Declaration ประโยชน์ของสิ่งนี้ คือMechanism ชื่อ Type Checker มีอยู่เพียง Mechanism เดียวเท่านั้น ซึ่งทำหน้าที่ในการรับผิดชอบการใช้ออบเจกต์ และคุณสมบัติของออบเจกต์ได้อย่างถูกต้อง

2.2.2.3 การ Implement ADT ด้วยคลาส

ใน Object-Oriented Language ADT ถูกนำเสนอเป็นโมดูล ซึ่งปกติจะเรียกว่า คลาส เช่น ใน C++ Class Definition มีรูปแบบดังนี้

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

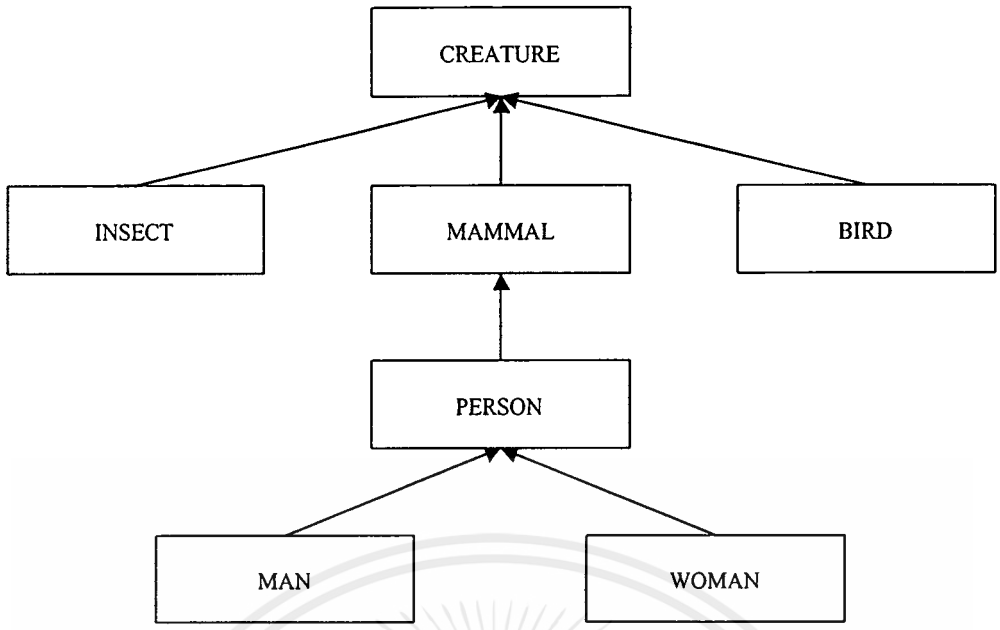
Private Components เป็นแอตทริบิวต์ และฟังก์ชันประเภทหนึ่งที่จำเป็นต้องใช้ในการนำเสนอคลาสและส่วนประกอบเหล่านี้ ไม่สามารถถูกเข้าถึงได้โดยโปรแกรมอื่นๆ ซึ่งใช้กับคลาสนั้น

Public Components แสดงถึงการติดต่อกับผู้ใช้และเป็นส่วนประกอบของคลาสที่ผู้ใช้สามารถอ้างถึงได้

Components เหล่านี้ อาจจะเป็นฟังก์ชันใดฟังก์ชันหนึ่ง ดังต่อไปนี้ คือ ฟังก์ชัน, ฟังก์ชันที่มีการส่งค่ากลับเป็นค่าของแอตทริบิวต์, เป็น Constructor หรือ Destructor Function, Accessor Function หรือ Transformer Function

2.2.3 การสืบทอดคุณสมบัติ (Inheritance)

ในระบบเชิงวัตถุ แนวความคิดของการสืบทอดคุณสมบัติ ทำให้ออบเจกต์ที่เป็น Specialized มีการสืบทอดคุณสมบัติ และการดำเนินการของออบเจกต์ที่เป็น Generalized คลาสของออบเจกต์ ให้อยู่ในรูปของซุปเปอร์คลาส และจะทำให้ได้ Specialized คลาส (ซับคลาส) จากซุปเปอร์คลาส คุณสมบัติพิเศษเหล่านี้ ได้มีการจัดเตรียมเพื่อสนับสนุนการนำกลับมาใช้ และเพื่อเพิ่มความสามารถ เพราะว่า การนิยามของออบเจกต์ที่ถูกกำหนดขึ้นใหม่ สามารถใช้ลักษณะพื้นฐานของคลาสที่มีอยู่แล้วได้



รูปที่ 2.2 แสดงการสืบทอดคุณสมบัติแบบลำดับชั้น

จากรูป จะได้ว่า ออบเจกต์ คลาส Mammal, Bird และ Insect เป็นซับคลาสของ Creature ส่วนออบเจกต์ คลาส Person เป็นซับคลาสของ Mammal และออบเจกต์ คลาส Man, Woman เป็นซับคลาสของ Person ซึ่งจะสามารถกำหนด Class Definition ของลำดับชั้น (Hierarchy) นี้ได้ในรูปแบบดังต่อไปนี้

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

Class Mammal
        Inherit Creature ;
        Properties
                Gestation_Period : Real ;
```

```
        Operations
                ...
End Creature.

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

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

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

2.2.3.1 การสืบทอดคุณสมบัติเพื่อเพิ่มความสามารถ

ในวิศวกรรมส่วนชุดคำสั่ง (Software Engineering) การสืบทอดคุณสมบัติเป็น Mechanism ที่เกี่ยวข้องกับวิวัฒนาการของระบบ นั่นคือ Inheritance Mechanism อาจจะไม่ถูกใช้กับลักษณะเฉพาะเพียงอย่างเดียวเท่านั้น แต่ยังได้ถูกนำไปใช้สำหรับการขยาย Software Module เพื่อจัดเตรียมการดำเนินการให้เพิ่มมากขึ้นอีกด้วย เช่น ถ้าเรามีคลาส (หรือโมดูล) A ซึ่งมีคลาสเป็น B แล้ว B จะมีการ Inherit

Service จาก A ดังนั้น จึงอาจพิจารณาได้ว่า B เป็นส่วนเพิ่มความสามารถของ A เพราะคุณสมบัติและการดำเนินการที่ใช้กับ Instance ของ A เป็นซับเซ็ตของคุณสมบัติและการดำเนินการที่ใช้กับ Instance ของ B

คุณสมบัติดังกล่าวของการสืบทอดคุณสมบัติ ก่อให้เกิดประโยชน์อย่างมากในการสร้างระบบซอฟต์แวร์ที่มีขนาดใหญ่ สำหรับงานทางด้านฐานข้อมูลการสืบทอดคุณสมบัติจะถูกใช้ประโยชน์เพื่อการจัดเตรียมความสะดวกในการออกแบบความสัมพันธ์ระหว่างเอนทิตี ที่มีโครงสร้างพฤติกรรม (Behaviour) เหมือนกัน

2.2.3.2 การสืบทอดคุณสมบัติและโพลิมอร์ฟิซึม

ในภาษาโปรแกรมเชิงวัตถุโพลิมอร์ฟิซึมหมายถึงความสามารถของโปรแกรมหนึ่งๆในการอ้างถึง Instance ของหลายๆชนิด หรือ คลาสที่ช่วงการทำงานรูปแบบอย่างง่ายของโพลิมอร์ฟิซึม ก็คือ Overloading ซึ่งยอมให้มีการใช้ชื่อเดียวกันได้มากกว่าหนึ่งเอนทิตีในหนึ่งโปรแกรม โดยระบบจะทำการตรวจสอบ Context ของแต่ละ Occurrence Name Overloading ช่วยอำนวยความสะดวกในด้านรูปแบบโดยยอมให้โปรแกรมเมอร์ สามารถกำหนดชื่อเหมือนกันสำหรับการ Implement ที่แตกต่างกันของการดำเนินการที่คล้ายคลึงกัน เช่น อาจกำหนดชื่อ "Add" ให้กับการดำเนินการ (Operation) ซึ่งทำการบวกเลขจำนวนเต็ม 2 จำนวน, การบวกเลขจำนวนจริง 2 จำนวน และสตริง (String) 2 จำนวน เมื่อการดำเนินการ Add นี้ถูกเรียกใช้โดย

Add (V1, V2)

ระบบจะตัดสินใจว่า ฟังก์ชันใดจะถูกใช้ในการบวกข้อมูล โดยทำการตรวจสอบชนิดของ V1 และ V2

ถ้าคลาส P เป็น Parent ของซับคลาส S แล้ว ออบเจกต์ของซับคลาส S จะสามารถนำไปใช้ในตำแหน่งใดก็ได้ ที่ออบเจกต์ของคลาส P สามารถใช้งานได้ ซึ่งแสดงถึงความหมายโดยนัยว่า ชุดของเมสเสจหนึ่ง สามารถส่งให้ทั้งออบเจกต์ที่อยู่ในคลาส P และออบเจกต์ที่อยู่ในคลาส S และเรียกคุณสมบัติในการส่งเมสเสจดังกล่าวนี้ว่า โพลิมอร์ฟิซึม

2.3 ระบบฐานข้อมูลแบบสัมพันธ์ (Relational database System)

ระบบฐานข้อมูลแบบสัมพันธ์ได้ถูกคิดค้นและพัฒนาขึ้นโดย ดร.เอดเกอร์ เอฟ.คอดด์ (Edger F.Codd) นำเสนอครั้งแรกในปี ค.ศ. 1969 ขณะที่เขาทำงานอยู่ที่บริษัท ไอบีเอ็ม ระบบฐานข้อมูลแบบสัมพันธ์นี้ก็มีพื้นฐานมาจากเซตทางคณิตศาสตร์ (Set Theory) และมีภาษา เอสคิวแอล(SQL) (Structure Query Language) เป็นภาษาที่ใช้ในกำหนดโครงสร้างและจัดการกับข้อมูลของฐานข้อมูลแบบสัมพันธ์นี้

การที่จะกล่าวว่าระบบฐานข้อมูลใดเป็นโมเดลแบบสัมพันธ์ (Relational Model) นั้น ต้องพิจารณาว่าฐานข้อมูลดังกล่าวมีองค์ประกอบครบทั้ง 3 ส่วนดังนี้

1. โครงสร้างข้อมูล เป็นไปตามนิยามคุณสมบัติของรีเลชัน (Relations)

2. ความถูกต้องของข้อมูล (Data Intergrity) เป็นไปตามกฎความถูกต้องของข้อมูลทั้งสอง (กล่าวอีกครั้งในหัวข้อ "กฎความถูกต้องของข้อมูล")

3. การจัดการข้อมูล (Data Manipulation) มีภาษาที่เป็น รีเลชั่นนัล คอมพลีท (Relational Complete) ในการจัดการฐานข้อมูล

2.3.1 โครงสร้างข้อมูลแบบสัมพันธ์ (Relational Data Structure)

2.3.1.1.ตัวอย่างตารางความสัมพันธ์

จากรูปที่ 2.3 มีคำเฉพาะที่ใช้เรียกส่วนต่าง ๆ ของตารางข้อมูล ดังนี้

1. โดเมน (Domain) หมายถึง หน่วยข้อมูลที่เก็บอยู่ในตารางความสัมพันธ์

2. แอททริบิวท์ (Attribute) หมายถึง ชื่อคอลัมน์ของตารางความสัมพันธ์

3. ทัพเพิล (Tuple) หมายถึง แถวของข้อมูลหนึ่งแถวของตารางความสัมพันธ์

4. ไพรมารี่คีย์ คือกรณีพิเศษของแคนดิเดตคีย์ (Candidate key) ความหมายของแคนดิเดตคีย์อธิบายได้ตามนิยามสองประการ เมื่อกำหนดให้ R เป็นรีเลชั่นใด ๆ A1, A2, ...,An เป็นแอททริบิวท์ของรีเลชั่นนั้น เซตของแอททริบิวท์ K เท่ากับ (Ai, Aj, ...,Ak) จะเป็นแคนดิเดตคีย์ของ R ก็ต่อเมื่อมีคุณสมบัติดังนี้

- ความเป็นหนึ่งเดียว (Uniqueness) คือ ในช่วงเวลาใด ๆ จะไม่ปรากฏว่าทัพเพิลสองทัพเพิลในรีเลชั่น R ที่มีค่าเท่ากันในแอททริบิวท์ที่ Ai, เท่ากันในแอททริบิวท์ที่ Aj, ..., และเท่ากันในแอททริบิวท์ที่ Ak
- มีขนาดเล็กที่สุด (Minimality) คือ จะไม่สามารถตัดแอททริบิวท์ใด ๆ ออกไปจากเซต K ได้โดยไม่ขัดกับคุณสมบัติความเป็นหนึ่งเดียว

สำหรับไพรมารี่คีย์ รีเลชั่นใด ๆ ก็คือ แคนดิเดตคีย์ตัวหนึ่งของรีเลชั่นนั้นที่ได้จากการกำหนดของผู้ออกแบบฐานข้อมูล เนื่องจากเป็นไปได้ว่ารีเลชั่นหนึ่งจะมีมากกว่าหนึ่งแคนดิเดตคีย์ จากรูปที่ 2.1 จะเห็นได้ว่าแอททริบิวท์ SNAME ก็เป็นแคนดิเดตคีย์เช่นกัน แต่ในกรณีนี้ได้กำหนดเอาแอททริบิวท์ S# เป็นไพรมารี่คีย์ และเรียก SNAME ว่าเป็น อัลเทอร์เนตคีย์ (Alternate key)

5. ฟอเรจ์นคีย์ (Foreign Key)

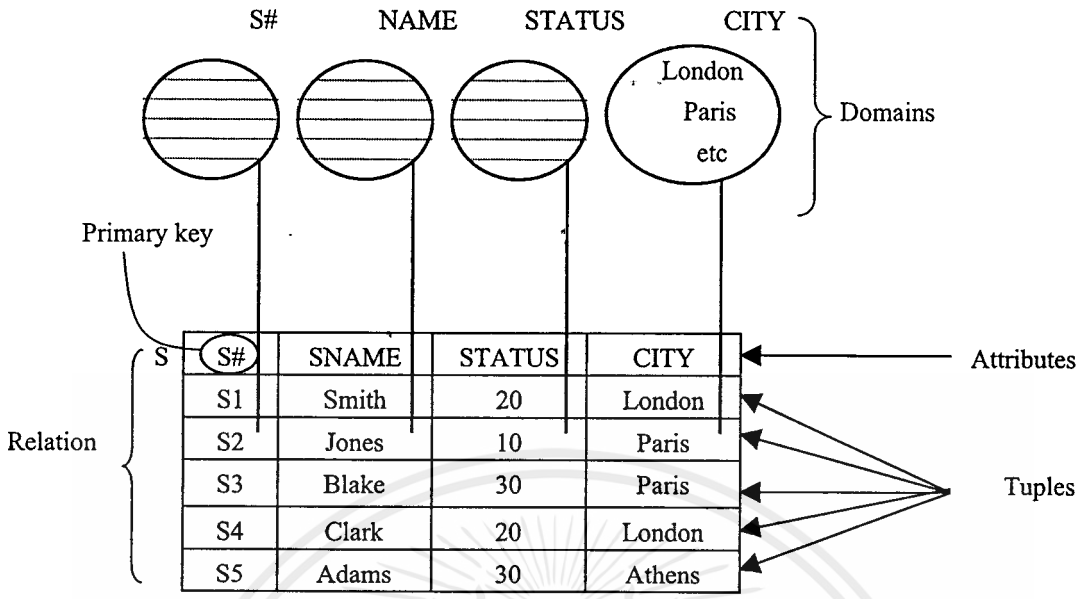
ฟอเรจ์นคีย์ คือ แอททริบิวท์หรือกลุ่มของแอททริบิวท์ในรีเลชั่น ซึ่งเป็นไพรมารี่คีย์ของรีเลชั่นอื่น (รีเลชั่นทั้งสองนี้ไม่จำเป็นต้องต่างกัน) ตัวอย่างเช่น

จากรีเลชั่น S (S#, SNAME, STATUS, CITY)

P (P#, PNAME, COLOUR)

และ SP (S#, P#, QTY)

S# และ P# เป็นฟอเรจ์นคีย์ในรีเลชั่น SP เพราะ S# เป็นไพรมารี่คีย์ของรีเลชั่น S และ P# ก็เป็นไพรมารี่คีย์ของรีเลชั่น P เช่นกัน ความสัมพันธ์ระหว่างฟอเรจ์นคีย์ไปยังไพรมารี่คีย์อื่นนี้เรียกว่า การอ้างถึง หรือ เรฟเฟอเรจ์น (References)



รูปที่ 2.3 แสดงตารางความสัมพันธ์ข้อมูล supplier

2.3.1.2. รีเลชั่น หมายถึง ผลคูณคาร์ทีเซียนของโดเมนที่สนใจในแอพพลิเคชั่น ประกอบขึ้นด้วยองค์ประกอบ 2 ส่วน คือ เฮดดิ้ง (Heading) และบอดี้ (Body) โดยเรากำหนดให้ D1, D2, ... ,Dn แทนโดเมนทั้งหลายในระบบงาน

1. เฮดดิ้ง ประกอบขึ้นจากเซตที่มีขนาดคงที่ (fixed set) ของแอททริบิวท์ A1, A2, ..., An โดย Ai คือ แอททริบิวท์ที่แทนในโดเมนที่ Di

2. บอดี้ ประกอบขึ้นจากเซตของทัพเพิลที่มีขนาดแปรผันตามเวลาของรีเลชั่น โดยแต่ละทัพเพิลประกอบด้วยค่าของข้อมูลของแอททริบิวท์ต่างๆเช่น Ai:Vi แทนค่าที่iของแอททริบิวท์ที่ i

หนึ่งทัพเพิลประกอบขึ้นด้วยค่าข้อมูลของแอททริบิวท์ที่อ้างอิงโดเมนต่างๆในตารางจากรูปที่ 2.2 เขียนได้เป็น

(S# : 'S1')
(SNAME : 'Smith')
(STATUS : 20)
(CITY : 'London')

คุณสมบัติของรีเลชั่น พิจารณาจากข้อกำหนด 4 ประการ คือ

1. ต้องไม่มีทัพเพิลที่ซ้ำกันภายในรีเลชั่น เนื่องจากบอดี้ของรีเลชั่นซึ่งประกอบขึ้นจากทัพเพิลหลายทัพเพิลมีลักษณะเป็นเซตทางคณิตศาสตร์ในรูปผลคูณคาร์ทีเซียนการซ้ำกันของสมาชิกภายในเซตซึ่งผิดนิยามทางคณิตศาสตร์ และเนื่องจากการไม่ซ้ำกันของแต่ละทัพเพิลนี้เอง ทำให้เราสามารถกล่าวได้ว่า "ทุก ๆ รีเลชั่นจะต้องมีไพรมารี่คีย์เกิดขึ้นเสมอ" เพราะอย่างน้อยที่สุดก็ต้องมีกลุ่มแอททริบิวท์ที่ค่าข้อมูลมีความเป็นหนึ่ง (ไม่ซ้ำกัน)

2. ลำดับของทัพเพิลไม่มีความสำคัญในการเก็บจากนิยามทางคณิตศาสตร์ที่ถือว่าสมาชิกภายในเซตจะอยู่กระจัดกระจายไม่มีลำดับ บอดี้ของรีเลชั่นซึ่งเป็นสมาชิกคือ ทัพเพิล จึงไม่มีความสำคัญกับลำดับของทัพเพิล
3. ลำดับของแอททริบิวท์ไม่มีความสำคัญในการเก็บ เช่นเดียวกับในคุณสมบัติที่ 2 ลำดับของแอททริบิวท์จึงไม่มีความสำคัญในการเก็บเช่นกัน
4. ค่าของข้อมูลแต่ละแอททริบิวท์ต้องเป็นค่าเดี่ยวหรืออะตอมมิกแวลลู (Atomic value)หมายความว่า เมื่อมีการบ่งชื่อรีเลชั่น ชื่อแอททริบิวท์และทัพเพิลที่ต้องการแล้ว จะต้องได้ค่าข้อมูลแอททริบิวท์ออกมาเพียงหนึ่งค่าเท่านั้น หรือกล่าวอีกอย่างหนึ่งได้ว่า"รีเลชั่นใด ๆ จะต้องไม่มีกลุ่มซ้ำ (Repeating group) อยู่ภายใน" ตัวอย่างดังรูปที่ 2.3 แสดงตารางความสัมพันธ์ที่มีกลุ่มซ้ำและตารางที่มีค่าข้อมูลเป็นค่าเดี่ยว

BEFORE

S#	PQ	
	P#	QTY
S1	P1	300
	P2	200
	P3	400
	P4	200
	P5	100
	P6	100
S2	P1	300
	P2	400
S3	P2	200
S3	P2	200
S4	P2	200
	P4	300
	P5	400

AFTER

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

รูปที่ 2.4 แสดงตัวอย่างตารางความสัมพันธ์ที่มีและไม่มีกลุ่มซ้ำ

2.3.1.3. ความถูกต้องของข้อมูล

ข้อมูลในระบบฐานข้อมูลแบบรีเลชั่นจะมีคุณสมบัติเป็นไปตามกฎความถูกต้อง (Intergrity Rule) อันได้แก่

1. กฎความถูกต้องของเอนติตี้ (Entity Integrity)
2. กฎความถูกต้องในการอ้างอิง (Referential Integrity)

1. กฎความถูกต้องของเอนติตี้

แอททริบิวท์ที่เป็นไพรมารี่คีย์ หรือเป็นส่วนหนึ่งของไพรมารี่คีย์ของรีเลชั่นใด ๆ ต้องไม่เป็นนัลแวลลู (Null values)

ค่าของข้อมูลที่เป็นนัลนั้น มีความหมายได้ 2 กรณีคือ กรณีแรก นัลแวลลูแสดงถึงว่าค่าข้อมูลนั้นนำไปใช้ไม่ได้ เช่น จำนวนวันลาคลอด ของผู้ชายจะถูกกำหนดให้เป็นนัล ซึ่งมีความหมายไม่เหมือนกับศูนย์ กรณีที่สอง นัลแวลลูแสดงถึงค่าข้อมูลที่ยังไม่ทราบค่าได้ว่าเป็นเท่าไร

ความสัมพันธ์ระหว่างเอนติตี้สามารถแบ่งออกได้เป็น 3 ชนิด คือ

1. ความสัมพันธ์แบบหนึ่งต่อหนึ่ง (one to one) หมายความว่า เมื่อเอนติตี้หนึ่งมีข้อมูลของคีย์หลักค่าหนึ่ง ค่าของข้อมูลดังกล่าวก็จะมีความสัมพันธ์กับค่าของข้อมูลของคีย์หลักของอีกเอนติตี้หนึ่งเพียงค่าเดียวเท่านั้น เช่น หากเรากำหนดให้ความสัมพันธ์ระหว่างเอนติตี้นักเรียนกับเอนติตี้ผู้ปกครองเป็นแบบหนึ่งต่อหนึ่งแล้วก็จะหมายความว่า การที่เราอ้างถึงนักเรียนคนใดคนหนึ่งก็จะสามารถอ้างถึงผู้ปกครองได้เพียงคนเดียวเท่านั้น และในทางตรงกันข้ามก็ต้องเป็นจริงด้วย คือเมื่อเราอ้างถึงผู้ปกครองคนใดคนหนึ่งแล้วก็จะสามารถอ้างถึงนักเรียนได้เพียงคนเท่านั้น

ชื่อนักเรียน	ชื่อผู้ปกครอง
A	A
B	B
C	C

รูปที่ 2.5 ความสัมพันธ์แบบหนึ่งต่อหนึ่ง

1. ความสัมพันธ์แบบหนึ่งต่อกลุ่ม (one to many) หมายความว่า เมื่อเอนติตี้หนึ่งมีข้อมูลของคีย์หลักค่าหนึ่ง ค่าข้อมูลดังกล่าวก็จะมีความสัมพันธ์กับค่าข้อมูลของคีย์หลักของอีกเอนติตี้หนึ่งหลายได้ค่า เช่น หากเรากำหนดให้ความสัมพันธ์ระหว่างเอนติตี้นักเรียนกับเอนติตี้ผู้ปกครองเป็นแบบหนึ่งต่อกลุ่มแล้วก็จะหมายความว่า การที่เราอ้างถึงนักเรียนคนใดคนหนึ่งก็จะสามารถอ้างถึงผู้ปกครองได้หลายคน และในทางตรงกันข้ามก็จะหมายความว่า เมื่อเราอ้างถึงผู้ปกครองคนใดคนหนึ่งแล้วก็จะสามารถอ้างถึงนักเรียนได้คนเดียวเท่านั้น แต่ถ้าผู้ปกครองที่เราอ้างถึงเป็นคนละคนกันก็อาจจะอ้างอิงถึงนักเรียนคนเดียวก็เป็นได้

ชื่อนักเรียน	ชื่อผู้ปกครอง
A	A
B	A
C	C

รูปที่ 2.6 ความสัมพันธ์แบบหนึ่งต่อกลุ่ม

3. ความสัมพันธ์แบบกลุ่มต่อกลุ่ม (many to many) หมายความว่า ค่าข้อมูลของคีย์หลักของเอนติตี้หนึ่งที่ต่างกันอาจอ้างอิงถึงค่าข้อมูลของคีย์หลักของอีกเอนติตี้หนึ่งได้ค่าเดียวหรือหลายค่าก็ได้ เช่น หากเรากำหนดให้ความสัมพันธ์ระหว่างเอนติตี้นักเรียนและเอนติตี้ผู้ปกครองเป็นแบบกลุ่มต่อกลุ่มแล้วก็จะหมายความว่า การที่เราอ้างถึงนักเรียนคนใดคนหนึ่ง หรือหลายคนก็จะสามารถอ้างอิงถึงผู้ปกครองคนเดียวกันได้ และในทางกลับกัน การที่เราอ้างถึงผู้ปกครองคนหนึ่ง หรือหลายคนก็จะสามารถอ้างอิงถึงนักเรียนคนเดียวกันได้

ชื่อนักเรียน	ชื่อผู้ปกครอง
A	A
B	A
C	C
C	D

รูปที่ 2.7 ความสัมพันธ์แบบกลุ่มต่อกลุ่ม

2. กฎความถูกต้องในการอ้างถึง

ถ้า FK คือแอททริบิวท์ที่เป็นฟอเรจ์นในรีเลชั่น R2 และ PK คือแอททริบิวท์ที่เป็นไพรมารี่คีย์ของรีเลชั่น R1 แล้ว ค่าข้อมูลแต่ละค่าในแอททริบิวท์ FK ในรีเลชั่น R2 จะต้อง ก) ค่าเท่ากับค่าข้อมูลที่อยู่ในแอททริบิวท์ PK ในรีเลชั่น R1 ในทัพเพิลหนึ่ง หรือ ข) เป็นนัลทั้งหมดในทุก ๆ แอททริบิวท์ที่ประกอบกันเป็น FK

2.3.1.4 ทฤษฎีของระบบฐานข้อมูลเชิงสัมพันธ์

1. กฎของความคงสภาพ

กฎของความคงสภาพ ของโมเดลเชิงสัมพันธ์ เป็นทฤษฎีที่ช่วยยืนยันความถูกต้องของความสัมพันธ์ระหว่างข้อมูลว่า รีเลชั่นใดที่เป็นไปตามกฎความคงสภาพนี้แล้วย่อมจะมีความสัมพันธ์ระหว่างข้อมูลอย่างถูกต้องอยู่ตลอดเวลา ไม่ว่ารีเลชั่นนั้นจะมีการเปลี่ยนแปลงแก้ไขข้อมูลไปในรูปแบบใดก็ตาม

กฎของความคงสภาพมีความหมายอยู่ 2 ลักษณะ คือ กฎความคงสภาพของเอนติตี้ และกฎความคงสภาพของการอ้างอิง ดังอธิบายได้ดังนี้

1. กฎความคงสภาพของเอนติตี้ กล่าวว่า

"แอททริบิวท์ทุกตัวที่เป็นส่วนของคีย์หลักจะไม่อนุญาตให้มีค่าว่าง" หมายความว่า คีย์หลักของทุกรีเลชั่นจะไม่สามารถเก็บค่าข้อมูลที่เป็นค่าว่างได้ เหตุผลของข้อกำหนดนี้คือ เพื่อให้การเข้าถึงข้อมูลในทัพเพิลใดๆของรีเลชั่นมีความเป็นไปได้เสมอ เพราะถ้าคีย์หลักของทัพเพิลใดมีค่าข้อมูลเป็นค่าว่างแล้ว ก็จะส่งผลให้การเข้าถึงข้อมูลในทัพเพิลนั้นไม่สามารถกระทำได้อย่างแน่นอน

2. กฎความคงสภาพของการอ้างอิง กล่าวว่า

"ถ้าเรามีรีเลชั่น R2 ซึ่งมี FK เป็นคีย์นอกที่อ้างอิงถึงคีย์หลัก PK ในรีเลชั่น R1 สำหรับทุกค่าของ FK ใน R2 จะต้อง

2.1 มีค่าเท่ากับ PK ในทัพเพิลหนึ่งในรีเลชั่น R1

หรือ

2.2 มีค่าของแอททริบิวท์ทุกตัวใน FK เป็นค่าว่าง"

หมายความว่า แอททริบิวท์ใด ๆ ที่เป็นคีย์หลักของรีเลชั่นหนึ่ง เมื่อมีการนำแอททริบิวท์นั้นไปเป็นคีย์นอกของอีกรีเลชั่นหนึ่ง การเป็นคีย์นอกของแอททริบิวท์นั้นจะต้องมีโดเมนเป็นโดเมนเดียวกันกับแอททริบิวท์ที่เป็นคีย์หลัก ทั้งนี้ก็เพื่อให้การนำรีเลชั่นมาใช้ร่วมกัน (การนำรีเลชั่นมา join กัน) กระทำได้อย่างถูกต้อง คือ ทุกแอททริบิวท์ที่เป็นคีย์นอก จะต้องมีข้อมูลซ้ำกับข้อมูลของแอททริบิวท์ที่เป็นคีย์หลักอย่างแน่นอน แต่อาจมีบางค่าข้อมูลของแอททริบิวท์ที่เป็นคีย์หลักเป็นข้อมูลไม่อยู่ใน โดเมนของแอททริบิวท์ที่เป็นคีย์นอกก็ได้นั่นคือ โดเมนของคีย์นอกจะต้องเล็กกว่าหรือเท่ากับโดเมนของคีย์หลักเสมอ

รีเลชั่น R1

คีย์หลักของ R1	คีย์อื่น ๆ ของ R1
A	1
B	2
C	3
D	4

ตารางที่ 2.1 แสดงความสัมพันธ์ระหว่างคีย์หลักของ R1กับ คีย์อื่น ๆ ของ R1

รีเลชั่น R2

คีย์หลักของ R2	คีย์นอก ของ R1
A	A
B	B
C	C
D	D

ตารางที่ 2.2 แสดงความสัมพันธ์ระหว่างคีย์หลักของ R2 กับ คีย์นอก ของ R1

2. ฟังก์ชันการขึ้นต่อกัน

ฟังก์ชันการขึ้นต่อกันเป็นข้อกำหนดที่ช่วยเราเห็นถึงความสัมพันธ์ของแอททริบิวท์ต่าง ๆ ที่อยู่ในรีเลชั่น เพราะแอททริบิวท์ต่าง ๆ ที่อยู่ในแอนติตี้เดียวกันก็เป็นไปได้ที่แอททริบิวท์เหล่านั้นจะมีความสัมพันธ์กันเองโดยที่ความสัมพันธ์นี้อาจเกี่ยวข้องหรือไม่เกี่ยวกับความสัมพันธ์ที่มีต่อคีย์หลักของเอนติตี้

นั้นก็เป็นได้ ซึ่งการที่แอททริบิวท์เหล่านั้นมีความสัมพันธ์กันเองจะเป็นสิ่งที่เราต้องพิจารณาแยกเป็นรีเลชั่นย่อย ๆ เพราะแอททริบิวท์ของแต่ละรีเลชั่นก็ควรจะมีความสัมพันธ์กับคีย์หลักของรีเลชั่นของตนเท่านั้น กำหนดรีเลชั่น R ถ้ามีแอททริบิวท์ Y ของ R เป็นฟังก์ชันที่ขึ้นต่อแอททริบิวท์ X ของรีเลชั่น เราสามารถเขียนแทนได้ด้วยสัญลักษณ์

R.X ----> R.Y

อ่านว่า R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y

หรือ R.X มีฟังก์ชันในการเลือก R.Y

หรือ R.Y ขึ้นอยู่กับ R.X

นิยาม R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y ก็ต่อเมื่อ ทุกค่าข้อมูลของแอททริบิวท์ X ใน R จะมีค่าข้อมูลของแอททริบิวท์Y ใน R ได้เพียงค่าเดียวเสมอ โดยที่แอททริบิวท์ X และ Y อาจจะเป็นคีย์แบบรวม (composite key) ก็ได้

รีเลชั่น R

X	Y
A	1
B	2
C	1
D	2

ตารางที่ 2.3 แสดงความสัมพันธ์ระหว่าง X กับ Y (Composite Key)

นิยาม R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y อย่างเต็มที่ (R.Y fully functionally dependent on R.X) ก็ต่อเมื่อ R.Y มีฟังกชันการขึ้นอยู่กับ R.X และไม่ขึ้นอยู่กับข้อมูลเพียงบางส่วนของ R.X โดยที่แอททริบิวท์ X และ Y อาจจะเป็นคีย์แบบรวมก็ได้

รีเลชั่น R

X		Y
A	A	1
B	B	2
A	C	1
B	C	2

ตารางที่ 2.4 แสดงความสัมพันธ์ระหว่าง X กับ Y (Full Functionally Dependent)

2.4 ORDB (Object-Relational Database)

ORDB คือ ฐานข้อมูลชนิดที่นำแนวความคิดเชิงวัตถุ (Object-Oriented) มาผสมผสานกับ RDB (Relational Database) เพื่อทำให้ RDB มีความสามารถทางด้าน Object-Oriented ได้

เทคโนโลยีที่ใช้ในการทำให้ ORDB

1. การเพิ่มชั้น Object-Oriented ครอบบนชั้น RDB ที่มีอยู่

2. การค่อยๆ ออกแบบ RDB ให้รองรับงานประเภทมัลติมีเดีย และการกำหนดชนิดของข้อมูลที่มีความซับซ้อน (Complex Object)

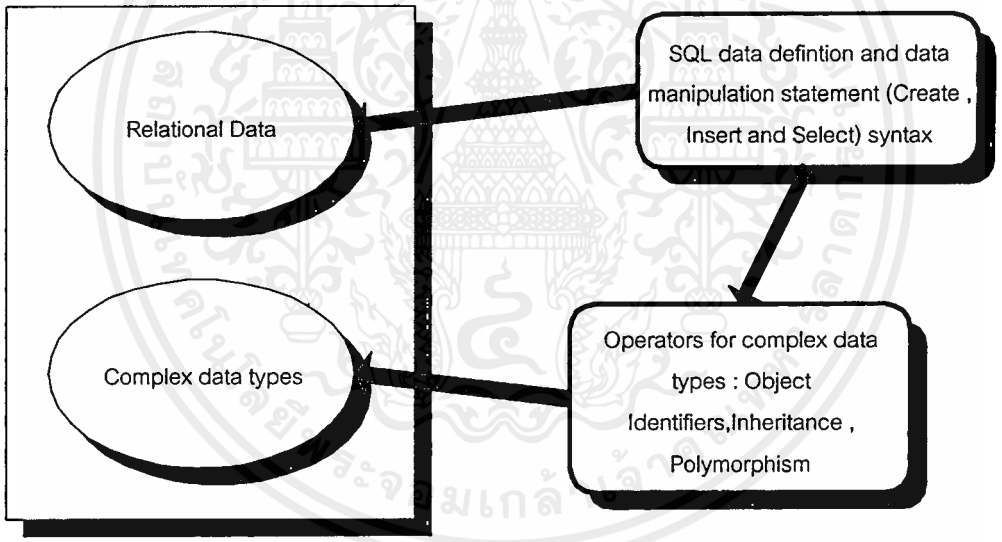
คุณสมบัติของ ORDB

1. ความสามารถของ RDB ที่มีความสมบูรณ์และประสิทธิภาพสูง

2. สนับสนุนการสร้างออบเจกต์ที่มีความซับซ้อน

3. มีความยืดหยุ่นสูง ยอมให้สร้างตัวดำเนินการ (Operators), ชนิดของข้อมูล (Type), ฟังก์ชัน

4. คุณสมบัติทาง Object-Oriented เช่น การสืบทอดคุณสมบัติ (Inheritance), โพลิมอร์ฟิซึม (Polymorphism), การปกป้องข้อมูล (Encapsulation)



รูปที่ 2.8 แสดงการทำงานของ RDB

2.5. อีอาร์โมเดล (ER-model)

อีอาร์โมเดล (ER-model : Entiry Relationship model) เป็นแผนภาพที่ถูกออกแบบมาเพื่อแสดงความสัมพันธ์ระหว่างเอนติตี้ต่าง ๆ ในรูปแบบที่เป็นรูปธรรมมากขึ้น เมื่อมีการใช้ อีอาร์โมเดล แสดงความสัมพันธ์ ระหว่างเอนติตี้แล้วก็จะไม่ต้องมีคำอธิบายความสัมพันธ์ใด ๆ อีก เพราะอีอาร์โมเดล ประกอบไปด้วยสัญลักษณ์ต่าง ๆ ที่แสดงถึงคุณลักษณะของเอนติตี้และแอททริบิวท์ได้ในตัวเองแล้ว

หลังจากที่เราสามารถหาความสัมพันธ์ระหว่างข้อมูลต่าง ๆ โดยแสดงความสัมพันธ์ระหว่างข้อมูลต่าง ๆ โดยแสดงความสัมพันธ์ของข้อมูลเหล่านั้นด้วย อีอาร์โมเดล แล้ว ขั้นตอนต่อไปนี้คือ การเปลี่ยน

ความสัมพันธ์ของข้อมูลที่อยู่บน อีอาร์โมเดล ให้อยู่ในรูปของรีเลชั่น ซึ่งการเปลี่ยนความสัมพันธ์ของข้อมูลที่อยู่บน อีอาร์โมเดล ให้อยู่ในรูปของรีเลชั่นนั้นจะมีขั้นตอนในการเปลี่ยนอยู่ 7 ขั้นตอน หลังจากนั้นก็ต้องทำการปรับปรุงรีเลชั่นที่ได้ให้มีความซ้ำซ้อนของความสัมพันธ์ระหว่างข้อมูลหมดไป (หรือเหลือน้อยที่สุดเพื่อความเหมาะสมในการนำไปใช้งานจริง) ซึ่งการปรับปรุงดังกล่าวก็จะมีทฤษฎีที่จะต้องอ้างถึงคือ กฎของความคงสภาพ (Integrity rule) ฟังก์ชันการขึ้นต่อกัน (Functional dependency) และทฤษฎีที่สำคัญที่สุดก็คือ ทฤษฎีการนอร์มัลไลซ์ (Normalization)

2.5.1 การแปลงความสัมพันธ์จากอีอาร์โมเดลไปสู่ในรูปของรีเลชั่น

การแปลงความสัมพันธ์ของข้อมูลจากอีอาร์โมเดลไปสู่ในรูปของรีเลชั่นมีขั้นตอนดังนี้

1. สำหรับแต่ละเอนติตี้ที่ไม่ใช่เอนติตี้แบบอ่อนของ อีอาร์โมเดล เราจะสร้างเป็นรีเลชั่น โดยมีแอททริบิวท์ที่เป็นแอททริบิวท์ธรรมดา (Simple Attribute) มาประกอบกัน แล้วทำการเลือกแอททริบิวท์ใดแอททริบิวท์หนึ่งหรือกลุ่มของแอททริบิวท์มาทำหน้าที่เป็นคีย์หลักของรีเลชั่น

2. สำหรับเอนติตี้แบบอ่อน เราจะสร้างรีเลชั่นที่เกิดจากการรวมกันของแอททริบิวท์ธรรมดาของแอนติตี้นั้น โดยที่รีเลชั่นนี้จะมีคีย์หลักคือคีย์รวม (combine key) ที่เกิดจากการรวมกันของคีย์หลักของเอนติตี้แบบอ่อน (partial key) กับคีย์หลักของเอนติตี้ที่ต้องอ้างถึง (ในกรณีนี้คีย์นี้จะเรียกว่าเป็นคีย์นอก)

3. สำหรับความสัมพันธ์ระหว่างเอนติตี้แบบหนึ่งต่อหนึ่ง เราจะสร้างรีเลชั่นจากความสัมพันธ์ดังกล่าวได้ 2 ลักษณะ คือ เลือกคีย์หลักของเอนติตี้ใดเอนติตี้หนึ่งมาเป็นคีย์หลักของรีเลชั่นนี้ แล้วให้คีย์ของเอนติตี้หนึ่งมาเป็นคีย์นอกของรีเลชั่นนี้ โดยถ้าความสัมพันธ์นี้มีแอททริบิวท์ให้นำแอททริบิวท์เหล่านั้นมารวมอยู่ในรีเลชั่นนี้ด้วย

4 .สำหรับความสัมพันธ์ระหว่างเอนติตี้แบบหนึ่งต่อกลุ่ม เราจะสร้างรีเลชั่นจากความสัมพันธ์ดังกล่าวโดยนำเอาคีย์หลักของเอนติตี้ฝั่งที่มีความสัมพันธ์แบบกลุ่มมาเป็นคีย์หลักของรีเลชั่นนี้ แล้วให้นำเอาคีย์หลักของเอนติตี้ฝั่งที่มีความสัมพันธ์แบบหนึ่งมาเป็นคีย์นอกของรีเลชั่นนี้ โดยถ้าความสัมพันธ์นี้มีแอททริบิวท์ก็ให้นำเอาแอททริบิวท์เหล่านั้นมารวมอยู่ในรีเลชั่นนี้ด้วย

5. สำหรับความสัมพันธ์ระหว่างเอนติตี้แบบกลุ่มต่อกลุ่มเราจะสร้างรีเลชั่นจากความสัมพันธ์นี้ โดยนำเอาคีย์หลักของทั้งสองเอนติตี้ มาประกอบกันเป็นคีย์หลักของรีเลชั่นนี้ โดยถ้าความสัมพันธ์นี้มีแอททริบิวท์ก็ให้นำเอาแอททริบิวท์เหล่านั้นมารวมอยู่ในรีเลชั่นนี้ด้วย

6. สำหรับเอนติตี้ใดที่มีแอททริบิวท์ที่มีค่าข้อมูลแบบหลายค่า (multivalue attribyte) หรือ repeating group) ก็ให้สร้างรีเลชั่นใหม่ โดยมีคีย์หลักของเอนติตี้นั้นรวมกับแอททริบิวท์ดังกล่าวเป็นคีย์หลักของรีเลชั่นนี้

7. สำหรับความสัมพันธ์ระหว่างเอนติตี้ที่เกิดจากเอนติตี้มากกว่า 2 เอนติตี้ ให้สร้างรีเลชั่นของความสัมพันธ์นี้โดยนำคีย์หลักของทุกเอนติตี้ประกอบกันเป็นคีย์หลักของเอนติตี้นี้ โดยถ้าความสัมพันธ์นี้มีแอททริบิวท์ ก็ให้นำแอททริบิวท์เหล่านั้นมารวมอยู่ในรีเลชั่นนี้ด้วย

เมื่อทำการเปลี่ยนความสัมพันธ์ระหว่างข้อมูลที่อยู่ในรูปของโมเดล ER มาอยู่ในรูปของรีเลชั่นแล้ว เราจะได้รีเลชั่นที่อยู่ในรูปแบบนอร์มัลระดับที่ 1 ซึ่งจะต้องมีการทำการนอร์มัลไลซ์ต่อไปเสียก่อนจึงจะสามารถนำรีเลชั่นเหล่านี้ไปใช้งานในระบบจัดการฐานข้อมูลเชิงสัมพันธ์ได้

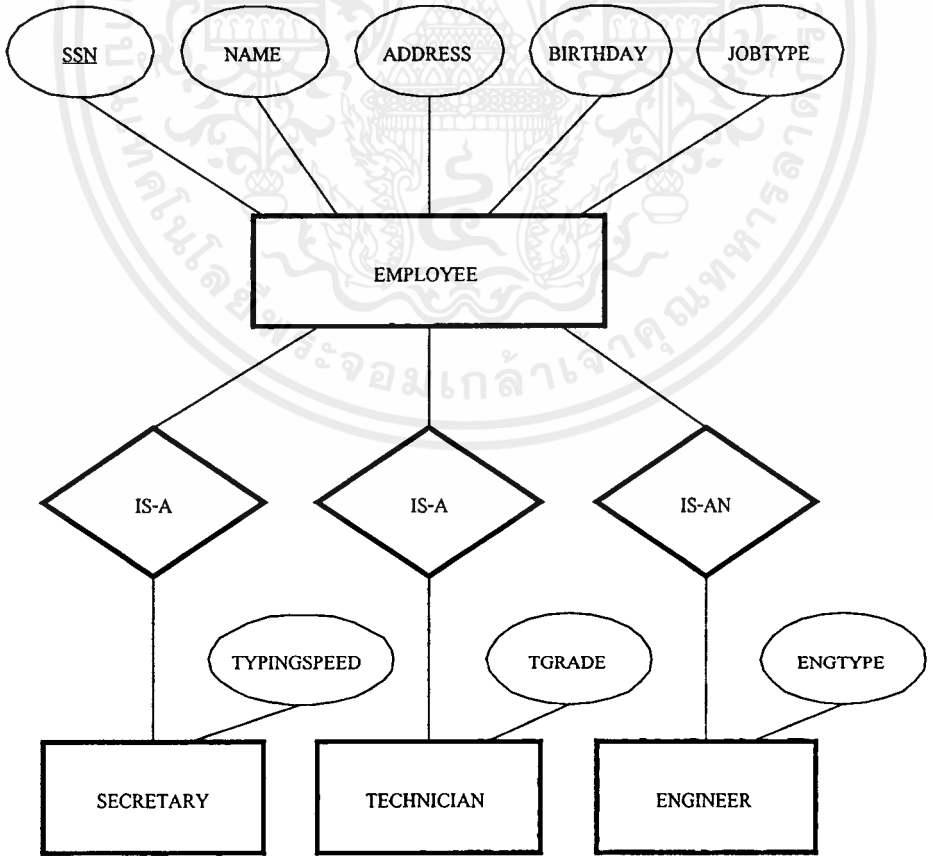
2.6 Extend-Entity Relationship (EER) โมเดล

EER โมเดลได้นำ ER โมเดล มาเป็นพื้นฐานและเพิ่มแนวความคิดเกี่ยวกับ ซับคลาส (Subclass) และซุปเปอร์คลาส (Superclass) เพื่อให้มีความสัมพันธ์ใกล้เคียงกับแอตทริบิวต์อินเฮอริแทนซ์

2.6.1 ซับคลาสและซุปเปอร์คลาส

EER โมเดล ได้กล่าวถึงแนวความคิดของซับคลาสของเอนทิตี้ไทป์ เนื่องจากเอนทิตี้ไทป์ อาจมีหลายๆกลุ่มย่อย เพื่อทำให้แสดงความหมายชัดเจนยิ่งขึ้น เช่น EMPLOYEE อาจแบ่งออกเป็น ENGINEER,MANAGER,SECRETARY และ TECHNICIAN เป็นต้น

เราจะเรียกกลุ่มย่อย ว่าเป็นซับคลาสของเอนทิตี้ไทป์ของ EMPLOYEE และเรียก EMPLOYEE ว่าเป็นซุปเปอร์คลาสของทุกๆซับคลาส ความสัมพันธ์ระหว่างซุปเปอร์คลาสและแต่ละซับคลาสของมันว่า Superclass/Subclass , Class/Subclass หรือ IS-A (IS-AN) และสมาชิกของซุปเปอร์คลาสทุกตัว จะเป็นสมาชิกของซับคลาสด้วย



รูปที่ 2.9 แสดงความสัมพันธ์ระหว่างซุปเปอร์คลาสและซับคลาส

2.6.2 การแปลงจาก EER โมเดลเป็นรีเลชัน

การแปลงจาก EER โมเดลจะเพิ่มเติมจาก ER โมเดล อัลกอริทึมในการแปลง ER โมเดล เป็นรีเลชันจะมี 7 ขั้นตอน แต่ใน EER โมเดล จะเพิ่มขั้นตอนที่ 8 เข้ามา

ขั้นตอนที่ 8 นำแอตทริบิวต์ทั้งหมดจากซุปเปอร์คลาสมารวมกับแอตทริบิวต์ที่เจาะจงขึ้นมาสำหรับซับคลาส

ตัวอย่างการแปลง EER โมเดลเป็นรีเลชัน

EMPLOYEE

SSN	NAME	BIRTHDATE	ADDRESS	JOBTYPE

SECRETARY

SSN	NAME	BIRTHDATE	ADDRESS	JOBTYPE	TYPINGSPEED

TECHNICIAN

SSN	NAME	BIRTHDATE	ADDRESS	JOBTYPE	TGRADE

ENGINEER

SSN	NAME	BIRTHDATE	ADDRESS	JOBTYPE	ENGTYPE

บทที่ 3

POSTGRES DBMS

3.1 ประวัติความเป็นมา

ในปี 1986 โครงการวิจัย POSTGRESQL ภายใต้การควบคุมของศาสตราจารย์ Michael Stonebraker แห่งมหาวิทยาลัยเบอร์กเลย์ รัฐแคลิฟอร์เนีย ได้เกิดขึ้น โดยมีจุดหมายเพื่อพัฒนาสถาปัตยกรรม ซึ่งสามารถเก็บ เรียกค้น และจัดการเก็บข้อมูลที่มีโครงสร้างไม่แน่นอนได้

POSTGRESQL เป็น DBMS ที่มีโมเดลเป็น Object-Relational DBMS มีคุณสมบัติดังนี้

- Relational เป็นคุณสมบัติของ RDBMS ที่มีความสมบูรณ์และประสิทธิภาพสูง เช่น การคิวรี (Query), การ Concurrency control, การทำ Transaction และการใช้งานหลายๆ คน (Multi-user)
- มีความยืดหยุ่นสูง (Highly Extensible) ยอมให้กำหนดโอเปอร์เรเตอร์ (Operator), ชนิดข้อมูล (Type) และฟังก์ชัน (Function) เองได้
- Object-Relational รองรับคุณสมบัติบางประการของแนวความคิดเชิงวัตถุ (Object-Oriented) เช่น การสืบทอดคุณสมบัติ (Inheritance)

สถาปัตยกรรม	โปรเซสเซอร์	ระบบปฏิบัติการ
DEC station 3000	Alpha AXP	OSF/1 2.1,3.0,3.2
DEC station 5000	MIPS	ULTRIX 4.4
Sun4	SPARC	SunOS 4.1.3
Hp 9000/700 and 800	PA-RISC	HP-UX 9.00,9.01
Intel	X86	Linux 1.2.8, ELF

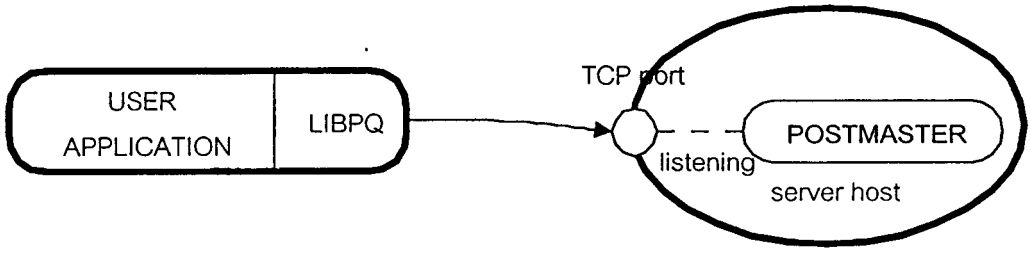
ตารางที่ 3.1 แสดงแพล็ตฟอร์มที่สามารถติดตั้ง

3.2 สถาปัตยกรรมของ POSTGRES DBMS

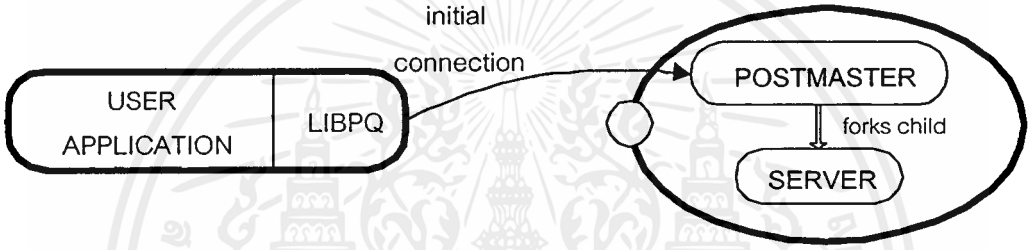
POSTGRES DBMS มีสถาปัตยกรรมแบบ Process-per-User Client/Server จะแบ่งออกเป็น 3 ส่วน คือ

1. Postmaster คือ โปรเซส (Process) ที่เป็นลักษณะโปรเซสปีศาจ (Daemon) จะจัดการการติดต่อสื่อสารระหว่าง Frontend และ Backend โดยเมื่อ Frontend ทำการร้องขอการติดต่อมายังเซอร์เวอร์ (Server) Postmaster จะทำการสร้าง Postgres server backend ขึ้นเพื่อรองรับ
2. Postgres Server Backend คือ ตัวรับคิวรี (Query) และส่งคำตอบไปยังโปรแกรม Frontend

3. Frontend Application คือโปรแกรมที่ทำงานบนเครื่องอื่น เช่น PSGL (เครื่องลูกข่าย) โดยจะติดต่อกับ Postgres Server Backend โดยผ่าน Postmaster



รูปที่ 3.1 แสดงเครื่องลูกข่ายร้องขอการติดต่อไป Postmaster



รูปที่ 3.2 แสดง Postmaster สร้าง Backend Server

รูปที่ 3.3 แสดงเครื่องลูกข่ายติดต่อกับ Backend Server

3.3 คุณสมบัติที่เพิ่มขึ้นของ SQL ใน POSTGRES DBMS

1. การสืบทอดคุณสมบัติ Inheritance

POSTGRES DBMS สามารถสืบทอดคุณสมบัติจากคลาสอื่นได้และคิวรี โดยอ้างถึงอินสแทน (Instance) ของคลาสที่สืบทอดคุณสมบัติได้

ยกตัวอย่าง

```
Create Table cities (
        name            text,
        poputation      float,
```

```
    altitude        int );

Create Table capitals (
    state           char2
    ) Inherits      ( cities );
```

คลาส capitals จะสืบทอดคุณสมบัติทุกๆ แอตทริบิวต์มาจากคลาส cities

2. Time Travel

POSTGRES DBMS รองรับความคิดของ Time Travel ยอมให้ผู้ใช้สามารถคิวรีแบบประวัติศาสตร์ (Historical) ได้ โดยการระบุช่วงเวลา

ยกตัวอย่าง

```
Select  name, population
From    cities [ '11/11/1996', 'now' ]
Where   name = 'Mariposa' ;
```

จากตัวอย่าง จะทำการคิวรีข้อมูลแอตทริบิวต์ name และ population ของ name = 'Mariposa' ในช่วงเวลา '11/11/1996' จนกระทั่งปัจจุบัน ถึงแม้ข้อมูลจะเปลี่ยนไปหรือถูกลบไปก็จะดึงข้อมูลที่เปลี่ยนแปลงออกมาด้วย

3. ข้อมูลไม่ต้อง Atomic (มีอาร์เรย์ได้)

Relational โมเดล มีข้อบังคับอยู่ข้อหนึ่งว่า แอตทริบิวต์ จะต้อง Atomic แต่ใน POSTGRES DBMS ยอมให้แอตทริบิวต์บรรจุข้อมูลเป็นอาร์เรย์ได้

ยกตัวอย่าง

สร้างคลาสที่มีลักษณะของแอตทริบิวต์เป็นอาร์เรย์

```
Create Table sal_emp (
    name                text
    pay_by_quarter      int4[ ],
    schedule            char16 [ ] [ ] );
```

ป้อนข้อมูลในคลาส sal_emp

```
Insert into sal_emp Values (
    'Bill',
    '{ 1000, 2000, 3000, 4000 }',
    '{{ "meeting", "lunch" }, { "talk" }}' );
```

3.4 ความยืดหยุ่น (Extensibility)

POSTGRES DBMS มีความยืดหยุ่น ทำให้สามารถขยายขีดความสามารถเกี่ยวกับการกำหนดโอเปอร์เรเตอร์, ชนิดข้อมูล, ฟังก์ชัน

การที่ POSTGRES DBMS สามารถขยายสิ่งต่างๆ เหล่านี้ได้ เนื่องจากการที่ POSTGRES DBMS มี Catalog-driven ถ้าเทียบกับ RDBMS ก็คือ System catalog (Data dictionary) ซึ่งจะเก็บข้อมูลเกี่ยวกับดาต้าเบส, ตาราง, คอลัมน์ เป็นต้น แต่ POSTGRES DBMS จะมีการเก็บข้อมูลที่มากกว่า โดยเพิ่มข้อมูลได้แก่ ชนิดข้อมูล, ฟังก์ชันและอื่นๆ ทำให้ POSTGRES สามารถสร้างฟังก์ชัน, ชนิดข้อมูล ขึ้นมาใหม่เองโดยผู้ใช้ให้เหมาะสมกับงานและข้อมูล

3.4.1 ฟังก์ชัน (Function)

POSTGRES DBMS สามารถสร้างฟังก์ชันโดยผู้ใช้ได้

รูปแบบในการนิยามฟังก์ชันใหม่

```
create function function_name (
        ( [ type1 {, type-n } ] )
        returns type-r
        as { '/full/path/to/objectfile' | 'sql-queries' }
        language { 'c' 'sql' 'internal' }
```

สามารถสร้างฟังก์ชันได้ 2 ลักษณะ คือ

1. การสร้างฟังก์ชันโดยใช้ภาษา SQL

ตัวอย่างเช่น

การสร้างฟังก์ชัน ชื่อ add_em โดยอ้างถึงอาร์กิวเมนต์ (Argument) ของฟังก์ชันด้วย $1 และ $2 ทำการบวกค่าส่งเข้ามาทางอาร์กิวเมนต์และส่งค่าผลลัพธ์

```
create function add_em ( int4, int4 ) returns int4
as 'select $1 + $2 ;' language 'sql' ;
select add_em ( 1, 2 ) as ANSWER
```

ANSWER
3

การสร้างฟังก์ชันโดยมีอาร์กิวเมนต์เป็น Composite type (เช่น Emp) เรามักจะไม่เจาะจงอาร์กิวเมนต์ที่ต้องการ

```
create function double_salary ( Emp ) returns int4
as 'select $1.salary * 2 as SALARY ;' language 'sql' ;
select name, double_salary ( Emp ) as DREAM
```

```
from Emp
where Emp.dept = 'toy' ;
```

Name	dream
Sam	2400

2. การสร้างฟังก์ชันโดยใช้ภาษาซี (C Programming Language)

ตัวอย่างเช่น

สร้างฟังก์ชัน c_overpaid ด้วยภาษาซี

```
#include "postgres.h"
#include "libpq-fe.h"
bool c_overpaid ( TUPLE t, int4 limit )
{
        bool    isnull = false ;
        int4    salary ;
        salary  = ( int4 ) GetAttributeByName ( t, "salary", &isnull ) ;
        if ( isnull )
                return ( false ),
        return ( salary > limit ) ;
}
```

แล้วทำการคอมไพล์เป็นไฟล์ชื่อ funcs.so และทำการสร้างฟังก์ชันภายใน POSTGRES DBMS

```
create function c_overpaid ( Emp, int4 ) returns bool
as 'funcs.so' language 'c' ;

select name, c_overpaid ( Emp, 1500 ) as overpaid
from Emp
where name = 'Bill' or name = 'Sam' ;
```

3.4.2 แบบข้อมูล (Type)

POSTGRES DBMS สามารถกำหนดแบบข้อมูลขึ้นมาใช้ใหม่ได้ตามที่ผู้ใช้ต้องการ โดยที่แบบข้อมูลที่สร้างขึ้นมาใหม่นี้ จำเป็นต้องมีฟังก์ชันนำข้อมูลเข้า (Input) และฟังก์ชันนำข้อมูลออกแสดง (Output) เสมอ การสร้างแบบข้อมูลใน POSTGRES DBMS มีรูปแบบดังนี้

```
create type typename ( internallength = ( number | variable ),
        [ externallength = ( number | variable ), ]
        input = input_function,
        output = output_function,
        [ ,element = typename ]
        [ ,delimiter = < character > ]
        [ ,default = "string" ]
        [ ,send = send_function ]
        [ ,receive = receive_function ]
        [ ,passedbyvalue ] )
```

ยกตัวอย่าง การสร้างชนิดข้อมูลที่เป็น Complex number โดยต้องมีการสร้างฟังก์ชันอินพุทและเอาท์พุทด้วยภาษาซี โดยเริ่มต้นกำหนดโครงสร้าง

```
typedef struct Complex {
        double   x;
        double   y;
}Complex ;
```

และกำหนดรูปของสตริง (String) ที่ใช้ในการแสดงคือ (x, y) เราจำเป็นต้องเขียนโปรแกรมในการแยกองค์ประกอบต่างๆ ให้ถูกต้อง

ฟังก์ชันอินพุท (INPUT)

```
Complex *complex_in ( char * str )
{
        double   x, y ;
        Complex  *result ;
        if ( sscanf ( str, "c%If, %If)", &x, &y ) ! = 2 ) {
                elog ( WARN, "complex_in : error in parsing return NULL ;
        }
        result = ( Complex * ) palloc ( size of ( Complex )) ;
        result  ->  x =  x;
        result  ->  y = y;
        return ( result );
}
```

ฟังก์ชันเอาท์พุท (OUTPUT)

```
char * complex_out ( Complex * complex )
```

```
{
        char * result;
        If ( complex == NULL )
                return ( NULL );
        result = ( char * ) palloc ( 60 ),
        sprintf ( result, "%g, %g )", complex -> x, complex -> y ) ;
        return ( result );
}
```

หลังจากการเขียนโปรแกรมอินพุทและเอาท์พุท แล้วทำการคอมไพล์เป็นไฟล์ชื่อ complex.so ทำการสร้างฟังก์ชันอินพุทและเอาท์พุทของชนิดข้อมูล Complex number และชนิดข้อมูล Complex number ใน POSTGRES DBMS

```
create function  complex_in ( opaqne )
        returns  complex
        as  'complex.so'
        language  'c';

create function  complex_out ( opaque )
        returns  opaque
        as  'complex.so'
        language  'c' ;

create type  complex (
        intrenallength = 16,
        input = complex_in,
        output = complex_out ) ;
```

3.4.3 โอเปอร์เรเตอร์ (Operators)

POSTGRES DBMS สามารถสร้างโอเปอร์เรเตอร์ขึ้นมาใช้กับชนิดข้อมูล (Type) ต่างๆ ได้เป็นลักษณะโอเวอร์โหลด (Overloaded) หรือการใช้จำนวนและชนิดข้อมูลของอาร์กิวเมนต์ที่แตกต่างกัน การสร้างโอเปอร์เรเตอร์ใน POSTGRES DBMS มีรูปแบบดังนี้

```
create operator  operator_name
        ( [ leftarg = type-1 ]
          [ ,rightarg = type-2 ]
          ,procedure = func_name
```

```
[, commutator = com_op ]
[, negator = neg_op ]
[, restrict = res_proc ]
[, hashes ]
[, join = join_proc ]
[,sort = sor_op1 {, sor_op2 } ]
)
```

ยกตัวอย่าง การสร้างโอเปอร์เรเตอร์ในการบวก complex number 2 จำนวน

1. สร้างฟังก์ชัน complex_add ด้วยภาษาซี และทำการคอมไพล์เป็น Complex.so

```
Complex * complex_add ( Complex *a, Complex *b )
{
        Complex *result;
        result =  ( Complex * ) palloc ( sizeof ( Complex ));
        result -> x = a -> x + b -> x ;
        result -> y = a -> y + b -> y ;
        return ( result ) ;
}
```

2. สร้างฟังก์ชันภายใน POSTGRES DBMS

```
create function complex_add ( Complex,Complex )
        returns Complex
        as 'complex.so'
        language 'c' ;
```

3. สร้างโอเปอร์เรเตอร์ภายใน POSTGRES DBMS

```
create operator + (
        leftarg = complex,
        rightarg  = complex,
        procedure = complex_add
        commutator = +
) ;
```

4. ทำการทดสอบ

```
select ( a+b ) as c from test_complex;
```

C
(5.2, 6.05)
(133.42, 144.95)

3.4.4 แอกกริเกชัน (Aggregates)

แอกกริเกชัน คือ ความสัมพันธ์ของหลายๆ ออบเจกต์รวมเป็นออบเจกต์เดียวกัน เช่น การหาผลรวมจากหลายๆ อินสแทน (Instance) โดยที่ POSTGRES DBMS สามารถสร้างแอกกริเกชัน โดยมีรูปแบบดังนี้

```
create aggregate   agg_name [ as ]
        ( [ sfunc1 = state-transition-function-1
         ,basetype = data-type
         ,stype1 = sfunc1-return-type ]
         [, sfunc2 = state-transition-function-2
         , stype2 = sfunc2-return-type ]
         [, finalfune = final-function ]
         [, initcond1 = initial-condition-1 ]
         [, initcond2 = initial-condition-2 ] )
```

ยกตัวอย่าง การสร้างแอกกริเกชัน ในการหาผลรวมของ Complex number

1. การสร้างโดยใช้ฟังก์ชัน complex_add ในการบวกค่าของ Complex number

```
create aggregate complex_sum (
        sfunc1 = complex_add,
        basetype = complex,
        stype1 = complex,
        initcond1 = '( 0,0 )'
);
```

2. ทำการทดสอบ

```
select complex_sum ( a ) from test_complex ;
```

Complex_sum
(34, 53.9)

3.5 LIBPQ Library

LIBPQ เป็นชุดคำสั่งที่ช่วยให้ผู้เขียนโปรแกรม (Programmer) สามารถทำการติดต่อกับ Postgres ได้ง่ายและมีประสิทธิภาพ โดย LIBPQ เป็นชุดคำสั่งที่อนุญาตให้เครื่องลูกข่าย (Client) สามารถส่งคำสั่งสอบถาม (Query) ไปยังเครื่องแม่ข่าย (Server) ที่ติดตั้งระบบจัดการฐานข้อมูล POSTGRES DBMS อยู่ และรับผลลัพธ์ที่ได้จากการส่งคิวรีกลับมา

โดยในเอกสารชุดนี้จะอธิบายถึงชุดคำสั่งการติดต่อกับ POSTGRES DBMS ด้วยชุดคำสั่งภาษาซี (C Language) โดยจะมีรูปแบบโครงสร้าง รูปแบบคำสั่ง และแนวทางในการเขียนโปรแกรมประกอบและข้อเสนอแนะต่างๆ และทั้งนี้ผู้สนใจยังสามารถดูตัวอย่างในการเขียนโปรแกรมที่เรียกใช้ชุดคำสั่งของ LIBPQ ได้จากแฟ้มข้อมูลดังต่อไปนี้

usr/src/pgsql/src/test/examples

usr/src/pgsql/src/test/regress

usr/src/pgsql/src/bin/psql

3.5.1 การควบคุมและกำหนดค่าเริ่มต้น (Control and Initialization)

- **PGHOST** เป็นการกำหนดชื่อเครื่อง Server
- **PGOPTIONS** กำหนด Option ที่จะถูกส่งไปยัง Postgres Backend
- **PGPORT** กำหนดช่องทางการสื่อสารกับ Postgres Backend
- **PGTTY** กำหนดชื่อไฟล์หรือ TTY สำหรับข้อความที่ได้จากการตรวจสอบจาก Postgres Backend
- **PGDATABASE** กำหนดชื่อฐานข้อมูล (Database Name)
- **PGREALM**

3.5.2 ฟังก์ชั่นในการติดต่อระบบฐานข้อมูล (Database Connection Function)

ฟังก์ชันดังต่อไปนี้ จะทำการสร้างช่องทางการติดต่อสื่อสารไปยัง Postgres Backend โดยฟังก์ชั่นเหล่านี้ได้ทำการพัฒนาจากภาษาซี

PQsetdb

ทำการสร้างช่องทางการสื่อสารกับ Postgres DBMS

รูปแบบคำสั่ง

```
Pgconn  *PQsetdb(     char    *PGHOST,
                      char    *PGPORT,
                      char    *PGOPTIONS,
                      char    *PGTTY,
                      char    *dbname);
```

ถ้าค่าที่โดยส่งไปในตัวแปรต่างๆเป็น NULL จะเป็นการบอกว่าให้ระบบทำการตรวจสอบสภาพแวดล้อมภายในตัวระบบเอง

โดยเมื่อทำการเรียกใช้ฟังก์ชั่น PQsetdb แล้ว ฟังก์ชั่นดังกล่าวจะส่งตัวชี้ PGconn กลับมาโดยถ้ามีการกระทำใดๆไปยัง Postgres DBMS จะต้องถูกกระทำผ่าน ตัวชี้ PQconn เท่านั้น

และฟังก์ชั่นดังกล่าวยังได้ส่งสถานะของการติดต่อกลับมาด้วย โดยสถานะดังกล่าวจะถูกส่งไปยังตัวแปร PQstatus ซึ่งจะได้อธิบายภายหลังต่อไป

PQdb ฟังก์ชั่นนี้จะส่ง ชื่อของฐานข้อมูลที่ได้มีการทำการติดต่อกับ Postgres DBMS

รูปแบบคำสั่ง

```
char    *PQdb(PGconn *conn)
```

PQhost ฟังก์ชั่นนี้จะส่ง ชื่อ HOST NAME ที่ได้ทำการติดต่อกับ Postgres DBMS

รูปแบบคำสั่ง

```
char    *PQhost(PGconn *conn)
```

PQoptions ฟังก์ชั่นนี้จะส่ง PGOPTIONS ที่ใช้ในการติดต่อกับ Postgres DBMS

รูปแบบคำสั่ง

```
char    *PQoptions(PGconn *conn)
```

PQport ฟังก์ชั่นนี้จะส่ง PGPORT ที่ใช้ในการติดต่อกับ Postgres DBMS

รูปแบบคำสั่ง

```
char    *PQport(PGconn *conn)
```

PQtty ฟังก์ชั่นนี้จะส่ง PGTTY ที่ใช้ในการติดต่อกับ Postgres DBMS

รูปแบบคำสั่ง

```
char    *PQtty(PGconn *conn)
```

PQstatus ฟังก์ชั่นนี้จะส่งสถานะของการติดต่อกับ Postgres DBMS โดยจะมีผลลัพธ์ดังนี้

CONNECTION_OK หมายถึงการติดต่อไปยัง Postgres Backend สำเร็จ

CONNECTION_BAD หมายถึงการติดต่อไปยัง Postgres Backend ไม่สำเร็จ

รูปแบบคำสั่ง

```
ConnStatusType  *PQstatus(PGconn  *conn)
```

ดังนั้นจึงควรทำการตรวจสอบก่อนเสมอว่าการติดต่อไปยัง Postgres DBMS สำเร็จหรือไม่ ถ้าสำเร็จจึงจะทำการส่ง Query ไปยัง Postgres DBMS ต่อไป เพื่อเป็นการป้องกันการเกิดข้อผิดพลาดขึ้นได้

PQerrorMessage ฟังก์ชั่นนี้จะแจ้งข้อความของสาเหตุที่ไม่สามารถติดต่อกับฐานข้อมูลได้

รูปแบบคำสั่ง

```
char    *PQerrorMessage(PGconn  *conn)
```

PQfinish

ฟังก์ชั่นนี้จะทำการยกเลิกการติดต่อสื่อสารกับ Backend และคืนหน่วยความจำที่ใช้โดยโครงสร้าง PGconn ข้อควรจำ หลังจากเรียกใช้ฟังก์ชั่นนี้แล้ว ตัวชี้ PGconn ไม่ควรจะถูกเรียกใช้อีกเพราะตัวชี้ PGconn จะไม่สามารถอ้างอิงได้อีก

รูปแบบคำสั่ง

```
void      PQfinish(PGconn  *conn)
```

PQreset

ฟังก์ชั่นนี้จะทำการปิดช่องทางการสื่อสารกับ Backend โดยการดำเนินการนี้จะปิด IPC Socket การติดต่อกับ Backend และจะสร้างการติดต่อไปยัง Backend เดิมใหม่อีกครั้ง

รูปแบบคำสั่ง

```
void      PQreset(PGconn  *conn)
```

PQtrace

ฟังก์ชั่นนี้จะทำให้ส่งข้อความระหว่าง Frontend และ Backend ได้

รูปแบบคำสั่ง

```
void      PQtrace(PGconn  *conn,FILE*  debug_port)
```

PQUntrace

ฟังก์ชั่นนี้จะทำให้ไม่สามารถส่งข้อความระหว่าง Frontend และ Backend ได้

รูปแบบคำสั่ง

```
void      PQuntrace(PGconn  *conn)
```

3.5.3 ชุดคำสั่งที่ใช้ในการปฏิบัติ Query (Query Execution Function)

PQexec

ฟังก์ชั่นนี้จะทำส่ง Query ไปยัง Postgres DBMS และรับผลลัพธ์ที่ได้จากการส่ง Query กลับมา ถ้าผลลัพธ์ที่ได้มีค่าเป็น NULL จะต้องเรียกฟังก์ชั่น PQerrorMessage เพื่อดูว่ามี Error เกิดขึ้น

รูปแบบคำสั่ง

```
PGresult  *PQexec( PGconn  *conn,
                   char      *Query)
```

PQresultStatus

ฟังก์ชั่นนี้จะทำการส่ง สถานะของผลลัพธ์ที่ได้จากการส่ง Query ไปยัง POSTGRES DBMS โดย PQresultStatus จะสามารถส่งสถานะต่างๆได้ดังต่อไปนี้

PGRES_EMPTY_QUERY,

PGRES_COMMAND_OK,

PGRES_TUPLES_OK,

PGRES_COPY_OUT,

PGRES_COPY_IN,

PGRES_BAD_RESPONSE,

PGRES_NONFATAL_ERROR,

PGRES_FATAL_ERROR

ถ้าสถานะของผลลัพธ์เป็น PGRES_TUPLES_OK แล้วฟังก์ชั่นดังต่อไปนี้สามารถถูกเรียกใช้เพื่อนำผลลัพธ์ที่ได้จากส่ง Query มาใช้งานได้

PQntuples ฟังก์ชั่นนี้จะส่งจำนวนแถวของผลลัพธ์ว่ามีทั้งสิ้นกี่แถว

รูปแบบคำสั่ง

```
int     PQntuples(PGresult  *res)
```

PQnfields ฟังก์ชั่นนี้จะทำการส่งจำนวน fields ที่ได้จากการ Query

รูปแบบคำสั่ง

```
int     PQnfields(PGresult  *res)
```

PQfname ฟังก์ชั่นนี้จะทำการส่งชื่อ field ที่ได้จากการ Query โดยจะเริ่มต้นที่ตำแหน่งที่ 0

รูปแบบคำสั่ง

```
char    *PQfname( PGresult  *res,
                  int   field_index)
```

PQfnumber ฟังก์ชั่นนี้จะทำการส่งตำแหน่งของ field ที่มีชื่อ field ตามที่ต้องการ

รูปแบบคำสั่ง

```
int     PQfnumber(PGresult  *res,
                  char*  field_name)
```

PQftype ฟังก์ชั่นนี้จะการส่งชนิดของข้อมูล โดยจะอ้างอิงที่ ตำแหน่งของ field ที่ต้องการทราบ

รูปแบบคำสั่ง

```
Oid     PQftype(PGresult *res,
                int field_num);
```

PQfsize ฟังก์ชั่นนี้จะส่งขนาดของ field ที่ต้องการโดยอ้างอิงที่ ตำแหน่งของ field ที่ต้องการ ซึ่งผลลัพธ์จะมีหน่วยเป็นไบต์ โดยถ้าผลลัพธ์ที่ได้เท่ากับ -1 แสดงว่า field นั้นเป็นแบบความยาวไม่คงที่

รูปแบบคำสั่ง

```
int2    PQfsize(PGresult *res,
                int field_index)
```

PQgetvalue ฟังก์ชั่นนี้จะส่งข้อมูลที่ได้จากการ Query โดยต้องอ้างอิงว่าต้องการตำแหน่งที่เท่าใดทางแนวตั้งและตำแหน่งที่เท่าใดทางแนวนอน

รูปแบบคำสั่ง

```
char*   PQgetvalue(PGresult *res,
                   int tup_num,
                   int field_num)
```

PQgetlength ฟังก์ชั่นนี้จะส่งขนาดของ fields ที่ต้องการ โดยจะต้องบอกว่าต้องการตำแหน่งใดในแนวตั้งและตำแหน่งใดในแนวนอน

รูปแบบคำสั่ง

```
int     PQgetlength(PGresult *res,
                    int tup_num,
                    int field_num)
```

PQcmdStatus

ฟังก์ชั่นนี้จะส่งสถานะของคำสั่งที่เกี่ยวเนื่องกับคำสั่ง Query ครั้งสุดท้าย

รูปแบบคำสั่ง

```
char*   PQcmdStatus(PGresult *res)
```

PQoidStatus

ฟังก์ชั่นนี้จะส่งสตริงกับ หมายเลขประจำตัวของวัตถุ ของแถวที่ทำการเพิ่มเข้า ถ้าคำสั่ง Query สุดท้ายเป็นคำสั่ง INSERT ส่วนกรณีอื่นๆจะส่งค่าสตริงว่างกลับมา

รูปแบบคำสั่ง

```
char*    PQoidStatus(PGresult  *res)
```

PQdisplayTuples

ฟังก์ชั่นนี้จะทำการพิมพ์ผลลัพธ์ทั้งหมด และชื่อ fields ของผลลัพธ์ที่ได้จากการ Query

รูปแบบคำสั่ง

```
void    PQdisplayTuples(
        PGresult        *res,
        FILE*           fout,
        int             fillAlign,
        char            *filedSep,
        int             printHeader,
        int             quiet    )
```

PQclear

ฟังก์ชั่นนี้จะคืนหน่วยความจำที่ใช้เก็บผลลัพธ์ที่ได้จากการ Query ดังนั้นควรจะคืนหน่วยความจำส่วนนี้เมื่อไม่ได้ใช้เป็นเวลานาน

รูปแบบคำสั่ง

```
void    PQclear(PGresult  *res)
```

ส่วนฟังก์ชั่นอื่นๆที่ไม่ได้กล่าวถึงในเอกสารชุดนี้ ท่านผู้สนใจสามารถหาอ่านได้ในคู่มือการใช้งาน POSTGRES (The POSTGRES95 User Manual) ในบทที่ 12 ซึ่งผู้จัดทำได้นำเพียงบางส่วนที่สำคัญมากล่าวถึงเท่านั้นต้องขอกราบขออภัยไว้ ณ. ที่นี้ ขอขอบพระคุณอย่างยิ่ง

บทที่ 4

การสร้างแบบจำลองโดยใช้ POSTGRES DBMS

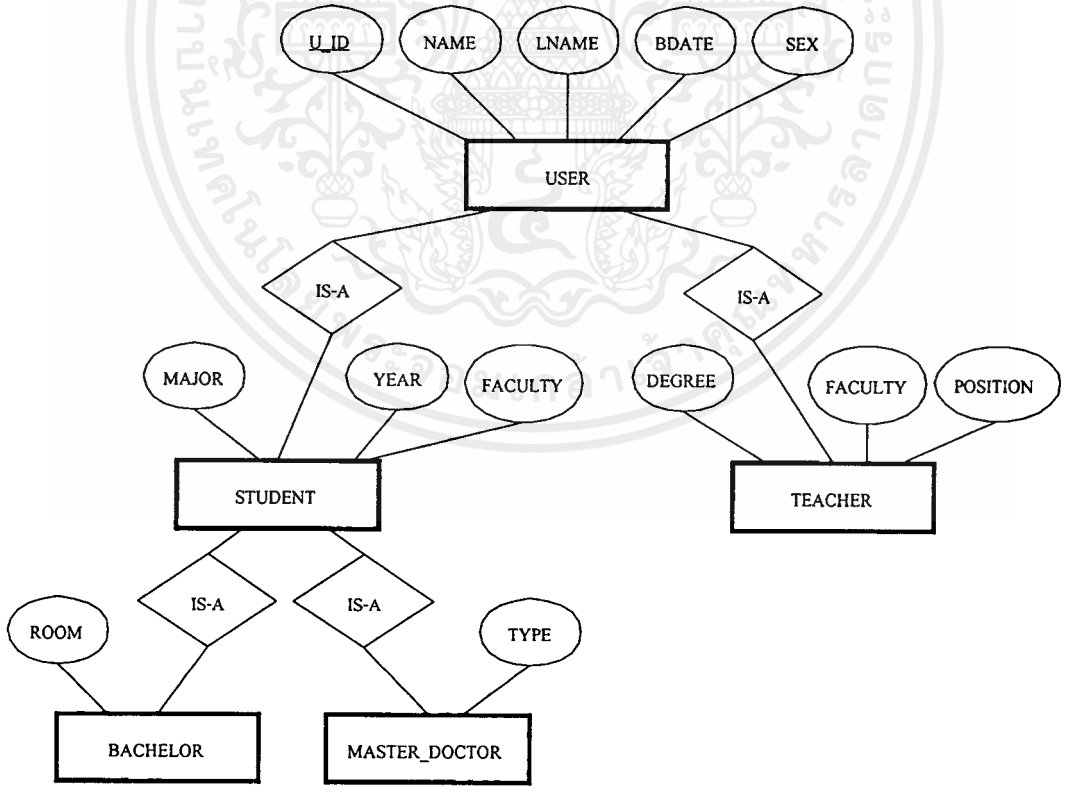
แบบจำลองที่จะใช้ในการทดลองคุณสมบัติ Object-Oriented DBMS คือระบบงานห้องสมุด ซึ่งมีรายละเอียดเหล่านี้คือ

1. สิ่งพิมพ์ ซึ่งจะแบ่งออกเป็นวารสาร,หนังสือทั่วไป
2. ผู้ใช้บริการ ซึ่งจะแบ่งออกเป็นผู้ใช้ทั่วไป,อาจารย์,นักศึกษาปริญญาตรี,ปริญญาโทและปริญญาเอก
3. ระเบียบการยืมขึ้นอยู่กับผู้ใช้บริการและประเภทสิ่งพิมพ์

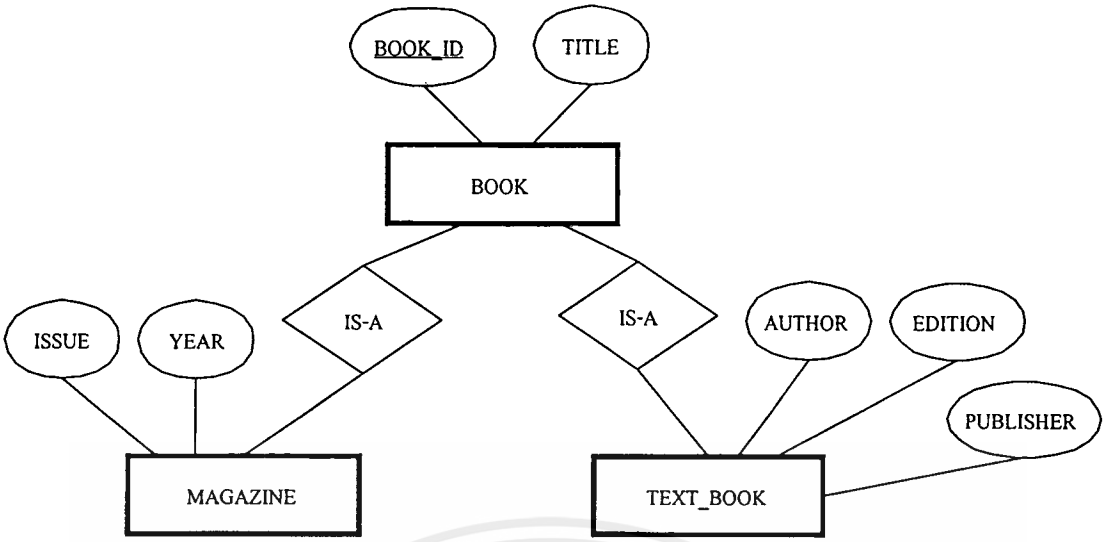
ส่วนประกอบในการจำลอง ระบบห้องสมุด

1. DBMS ใช้ POSTGRES DBMS
2. ภาษาโปรแกรมมิ่งที่ใช้ในการพัฒนาระบบ คือ ภาษาซี

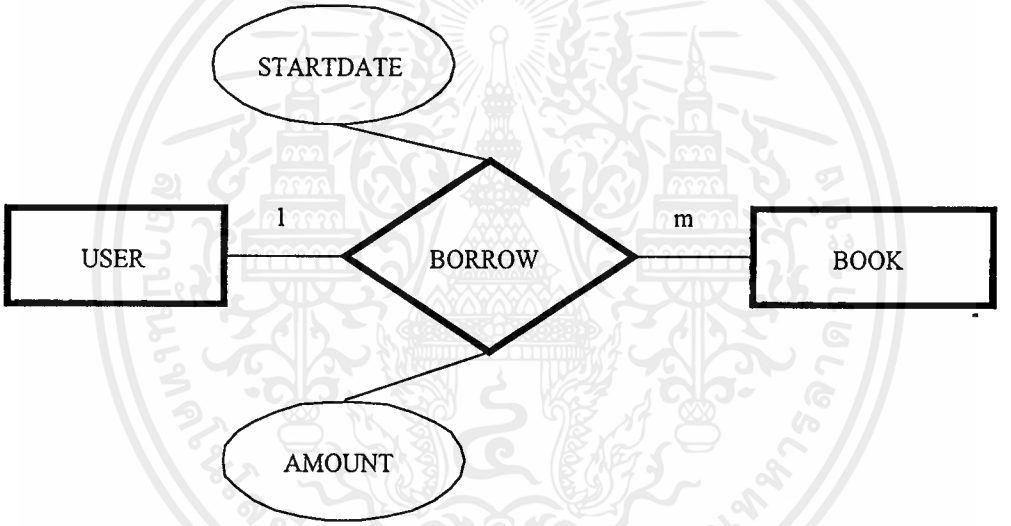
4.1 ออกแบบฐานข้อมูลระบบห้องสมุดโดย EER โมเดล



รูปที่ 4.1.ก แสดงส่วนของผู้ใช้บริการ



รูปที่ 4.1.ข แสดงส่วนของสิ่งพิมพ์



รูปที่ 4.1.ค แสดงส่วนของความสัมพันธ์ของผู้ใช้และสิ่งพิมพ์

รูปที่ 4.1 EER โมเดลของระบบห้องสมุด

4.1.1 การแปลงจาก EER โมเดลเป็นรีเลชัน จะได้

รีเลชันของบุคคลทั่วไป ชื่อตาราง USER

U_ID	NAME	LNAME	BDATE	SEX

รีเลชันของนักศึกษา ชื่อตาราง STUDENT

U_ID	NAME	LNAME	BDATE	SEX	FACULTY	MAJOR	YEAR

รีเลชันของ นักศึกษาปริญญาตรี ชื่อตาราง BACHELOR

U_ID	NAME	LNAME	BDATE	SEX	FACULTY	MAJOR	YEAR	ROOM

รีเลชันของนักศึกษาปริญญาโทและปริญญาเอก ชื่อตาราง MASTER_DOCTOR

U_ID	NAME	LNAME	BDATE	SEX	FACULTY	MAJOR	YEAR	TYPE

รีเลชันของอาจารย์ ชื่อตาราง TEACHER

U_ID	NAME	LNAME	BDATE	SEX	DEGREE	FACULTY	POSITION

รีเลชันของหนังสือ ชื่อตาราง BOOK

BOOK_ID	TITLE	STARTDATE	U_ID	AMOUNT

รีเลชันของหนังสือทั่วไป ชื่อตาราง TEXT_BOOK

BOOK_ID	TITLE	AUTHOR	EDITION	PUBLISHER	STARTDATE	U_ID	AMOUNT

รีเลชันของวารสาร ชื่อตาราง MAGAZINE

BOOK_ID	TITLE	ISSUE	YEAR	STARTDATE	U_ID	AMOUNT

4.2 แสดงคุณสมบัติ Object-Oriented ของ POSTGRES DBMS

ในการจำลองระบบงานห้องสมุดได้พยายามนำคุณสมบัติ Object-Oriented มาใช้อย่างเต็มที่ โดยทำการทดสอบคุณสมบัติดังต่อไปนี้

4.2.1 การถ่ายทอดคุณสมบัติ (Inheritance)

การทดสอบคุณสมบัตินี้ โดยจะทำการสร้างตารางซึ่งมีการถ่ายทอดคุณสมบัติ ซึ่งใน POSTGRES DBMS จะนิยามตาราง คือ คลาส

สร้างคลาส USER จากรีเลชันที่แปลงจาก EER โมเดล

```
create table USER (
        U_ID        varchar(8),
        NAME        varchar(25),
        LNAME       varchar(25),
        BDATE       date,
        SEX         bool );
```

สร้างคลาส STUDENT ที่สืบทอดคุณสมบัติจากคลาส USER

```
create table STUDENT (
        FACULTY     varchar(25),
```

```
    MAJOR       varchar(25),
    YEAR        char4 ) inherits (USER);
```

สร้างคลาส TEACHER ที่สืบทอดคุณสมบัติจากคลาส USER

```
create table TEACHER (
    FACULTY     varchar(25),
    DEGREE      varchar(25),
    POSITION    varchar(25)) inherits (USER);
```

สร้างคลาส BACHELOR ที่สืบทอดคุณสมบัติจากคลาส STUDENT

```
create table BACHELOR (
    ROOM        varchar(3)) inherits (STUDENT);
```

สร้างคลาส MASTER_DOCTOR ที่สืบทอดคุณสมบัติจากคลาส STUDENT

```
create table MASTER_DOCTOR (
    TYPE        varchar(10)) inherits (STUDENT);
```

สร้างคลาส BOOK

```
create table BOOK (
    BOOK_ID     varchar(8),
    TITLE       varchar(50),
    STARTDATE   date,
    AMOUNT      integer,
    U_ID        varchar(8));
```

สร้างคลาส MAGAZINE ที่สืบทอดคุณสมบัติจากคลาส BOOK

```
create table MAGAZINE (
    ISSUE       varchar(4),
    YEAR        varchar(4)) inherits (BOOK);
```

สร้างคลาส TEXT_BOOK ที่สืบทอดคุณสมบัติจากคลาส BOOK

```
create table TEXT_BOOK (
    AUTHOR      varchar(25),
    EDITION     varchar(2),
```

PUBLISHER varchar(25)) inherits (BOOK);

จากลักษณะการถ่ายทอดคุณสมบัติ (Inheritance) ภายใน POSTGRES DBMS ซึ่งขาดความสมบูรณ์ เนื่องจาก

- คลาส ใน POSTGRES DBMS ไม่รองรับการสร้างเมตธอด สำหรับคลาส ซึ่งทำให้การถ่ายทอดคุณสมบัติขาดการถ่ายทอดของเมตธอด

4.2.2 คุณสมบัติโพลิมอร์ฟิซึม (Polymorphism)

การทดสอบคุณสมบัตินี้ โดยจะทำการสร้างฟังก์ชัน ซึ่งมีลักษณะ Overloading ซึ่งยอมให้มีการใช้ชื่อเดียวกันได้มากกว่า 1 ฟังก์ชัน โดยมีพารามิเตอร์ที่แตกต่างกัน

ในระบบงานห้องสมุด จะมีระเบียบการยืมสิ่งพิมพ์ หนังสือทั่วไป

- นักศึกษาปริญญาตรี ยืมได้ 7 วัน
- นักศึกษาปริญญาโท/เอก ยืมได้ 14 วัน
- อาจารย์ ยืมได้ 30 วัน

สร้างฟังก์ชั่น ชื่อ BORROW ในการยืมหนังสือ โดยจะสร้างสำหรับ

1. ฟังก์ชันการยืมหนังสือสำหรับบุคคลทั่วไป

```
CREATE FUNCTION Borrow(Users,char8,char8,date) RETURNS int4 as
'update Text_Book set U_ID     = $2,
                      Amount   = 0,
                      startdate = $4
 where Book_ID = $3;
 select 1 as result;' LANGUAGE 'sql';
```

2. ฟังก์ชันการยืมหนังสือสำหรับนักศึกษาทั่วไป

```
CREATE FUNCTION Borrow(Student,char8,char8,date) RETURNS int4 as
'update Text_Book set U_ID     = $2,
                      Amount   = 5,
                      startdate = $4
where Book_ID = $3;
select 2 as result;'  LANGUAGE 'sql';
```

3. ฟังก์ชันการยืมหนังสือสำหรับนักศึกษาปริญญาตรี

```
CREATE FUNCTION Borrow(Bachelor,char8,char8,date) RETURNS int4 as
'update Text_Book set U_ID     = $2,
                      Amount   = 7,
```

```
                    startdate = $4
where Book_ID = $3;
select 3 as result' LANGUAGE 'sql';
```

4. ฟังก์ชันการยืมหนังสือสำหรับนักศึกษาปริญญาโท/เอก

```
CREATE FUNCTION Borrow(Master_Doctor,char8,char8,date) RETURNS int4 as
'update Text_Book set U_ID     = $2,
                    Amount    = 14,
                    startdate = $4
where Book_ID = $3;
select 4 as result' LANGUAGE 'sql';
```

5. ฟังก์ชันการยืมหนังสือสำหรับอาจารย์

```
CREATE FUNCTION Borrow(Teacher,char8,char8,date) RETURNS int4 as
'update Text_Book set U_ID     = $2,
                    Amount    = 30,
                    startdate = $4
where Book_ID = $3;
select 5 as result' LANGUAGE 'sql';
```

สร้างฟังก์ชัน Restore สำหรับคืนหนังสือ

```
CREATE FUNCTION Restore(char8) RETURNS int4 as
'update Text_Book set      U_ID     = NULL,
                           Amount   = NULL,
                           StartDate = NULL
where Book_ID = $1;
select 1 as result;' LANGUAGE 'sql';
```

ตัวอย่างในการใช้งาน

หากนักศึกษาปริญญาตรีรหัส 38013298 ต้องการยืมหนังสือรหัส TB000001

```
select Borrow(BACHELOR,'38013298','TB000001',now)
from BACHELOR;
```

จะได้ผลลัพธ์ คือ บันทึกข้อมูลในตาราง TEXT_BOOK โดยที่ BOOK_ID = 'TB000001' และคอลัมน์ U_ID = '38013298' , STARTDATE = เวลาปัจจุบัน(now) , AMOUNT = 7

จากการทดสอบคุณสมบัติโพลิมอร์ฟีซึมใน POSTGRES DBMS นั้นได้ผลการทดสอบดังต่อไปนี้

1. สามารถสร้างฟังก์ชันโพลิมอร์ฟีซึม ที่เกิดขึ้นขณะคอมไพล์โปรแกรม คือจะทำการตรวจสอบชนิดของพารามิเตอร์ของฟังก์ชัน
2. POSTGRES DBMS ไม่สามารถสร้างฟังก์ชันโพลิมอร์ฟีซึมที่เกิดขึ้นขณะรันโปรแกรมได้ เนื่องจาก POSTGRES DBMS ไม่มีการสร้างฟังก์ชันอยู่ภายในคลาสได้ จึงไม่สามารถสร้างฟังก์ชันที่มีชื่อซ้ำกันและอาร์กิวเมนต์เดียวกันได้แต่อยู่แต่ละคลาสได้

บทที่ 5

บทวิจารณ์และสรุป

5.1 สรุปผลการทำโครงงาน

ในการทำโครงงาน สามารถเรียนรู้ได้ตามวัตถุประสงค์ที่ได้กำหนดไว้ คือ

1. ศึกษาหลักการของแบบจำลองเชิงวัตถุ (Object-Oriented Model) ว่ามีการทำงานเช่นไร และมีคุณลักษณะอย่างไร
2. ศึกษาการทำงานของ POSTGRES DBMS ว่ามีคุณลักษณะอย่างไร
3. ทำการสร้างแบบจำลองระบบงานห้องสมุดแบบง่ายๆ เพื่อทดสอบคุณสมบัติ Object-Oriented Model ของ POSTGRES DBMS

5.2 ปัญหาที่เกิดขึ้น

ปัญหาที่เกิดขึ้นในการทำโครงงาน คือ

1. เนื่องจาก POSTGRES DBMS เป็นผลิตภัณฑ์ที่ใช้ในการศึกษาไม่ได้เป็นผลิตภัณฑ์ทางการค้า จึงทำให้มีข้อมูลเกี่ยวกับ POSTGRES DBMS หาได้ยาก และมีรายละเอียดน้อยมาก
2. เนื่องจาก POSTGRES DBMS เป็นผลิตภัณฑ์ที่ทำงานบนระบบปฏิบัติการยูนิกต์ จึงต้องเรียนรู้เกี่ยวกับการใช้งานระบบยูนิกต์ และ ภาษาซีบนยูนิกต์ด้วย

5.3 แนวทางการนำไปพัฒนาต่อ

จากการทำโครงงานนี้ ยังเป็นการทดสอบคุณสมบัติทาง Object-Oriented Database อย่างง่ายๆ ภายใต้ POSTGRES DBMS เท่านั้น ยังไม่ได้มีการประยุกต์ใช้กับชนิดข้อมูลที่มีความซับซ้อนมาก เช่น ภาพ , เสียง , ลายนิ้วมือ เป็นต้น ดังนั้นในการพัฒนาต่ออาจเพิ่มความสามารถใช้งานกับชนิดข้อมูลที่มีความซับซ้อนมาก เพื่อประยุกต์ใช้งานได้จริงต่อไป

ภาคผนวก

ซอร์ดโค้ดของระบบงานห้องสมุด

```
/* ---------------------------------------------------------------
 * Library.c  16-03-98
 *
 *---------------------------------------------------------------
 */

#include <stdio.h>
#include <string.h>
#include "libpq-fe.h"

static PGconn*   conn;
static PGresult* res = NULL;

void exit_nicely(PGconn* conn)
{
 PQfinish(conn);
 exit(1);
}

void show_data(char *Query)
{
   int i,j,nFields;
   char strTemp[2000] = "DECLARE tempcursor CURSOR FOR ";

   res = PQexec(conn,"BEGIN");
   PQclear(res);
   strcat(strTemp,Query);
   res = PQexec(conn,strTemp);
   PQclear(res);
   sprintf(strTemp,"FETCH ALL in tempcursor");
   res = PQexec(conn,strTemp);
```

```
        nFields = PQnfields(res);

        for (i=0;i < nFields;i++) printf("%-11s",PQfname(res,i));
        printf("\n");

        for (i=0;i < PQntuples(res); i++) {
          for(j=0;j < nFields;j++) {
            printf("%-11s",PQgetvalue(res,i,j));
          }
          printf("\n");
        }
        PQclear(res);
        res = PQexec(conn,"CLOSE tempcursor");
        PQclear(res);
        res = PQexec(conn,"END");
        PQclear(res);
      }

      void main()
      {
        char    ckey,User_ID[9],Book_ID[9],Query[2000];
        char    delkey,Table_Name[15],tempkey;
        char*   dbname;
        char    Fname[40],Lname[40],Bdate[10],Sex[2];
        char    Major[40],Faculty[40],User_Year[5],Class_Room[3];
        char    Department[40],Mtype[20],Degree[20];
        char    Title[40],Author[40],Edition[4],Publisher[40];
        char    Issue[5],Book_Year[5];
        int     nFields,i,j;

      /* PGconn*   conn;
        PGresult* res;
        PGnotify* notify;
      */
```

```
/* Make a connection to the database */
dbname = "Library";
conn = PQsetdb(NULL,NULL,NULL,NULL,dbname);

/* Check to see that the backend connection was successfully made. */
if (PQstatus(conn) == CONNECTION_BAD)
{
    fprintf(stderr,"Connection to database '%s' failed. \n ",dbname);
    fprintf(stderr,"%s",PQerrorMessage(conn));
    exit_nicely(conn);
}

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("=========================================================\n");
printf("=              MENU                    =\n");
printf("=========================================================\n");
printf("=       1.Borrow  Book                 =\n");
printf("=       2.Restore Book                 =\n");
printf("=       3.Add User into database       =\n");
printf("=       4.Add Book into database       =\n");
printf("=       5.Show data in user            =\n");
printf("=       6.Show data in book            =\n");
printf("=       7.Exit                         =\n");
printf("=========================================================\n");
printf("\n\n\n\n\n\n");
printf("Press number [1,2,3,4,5,6,7] to process : ");
ckey = getchar();
while (ckey != '7')
{
        if (ckey == '1')
        {
           while(1)
           {
                /* Borrow BOOK */
```

```
printf("================================================================\n");
        printf("=          Press enter to continue          =");
printf("\n================================================================\n");
        delkey = getchar();
        printf("\n User ID : ");
        gets(User_ID);
        res = PQexec(conn,"BEGIN");
        PQclear(res);

        sprintf(Query,"select * from users* where u_id = '%s';",User_ID);
    res = PQexec(conn,Query);
        if (PQresultStatus(res) != PGRES_TUPLES_OK)
        {
          fprintf(stderr,"FETCH ALL command didn't return tuples properly \n");

          PQclear(res);
          exit_nicely(conn);
        }
        nFields = PQnfields(res);
        Table_Name = "12345678901234";
        if (PQntuples(res) > 0 )
        {
          PQclear(res);
          while(1)
          {.
                printf(" Book ID : ");
                gets(Book_ID);
                sprintf(Query,"select * from book* where book_id = '%s';",Book_ID);
                res = PQexec(conn,Query);
                if (PQntuples(res) > 0)
                {
                  PQclear(res);
                  sprintf(Query,"select * from text_book where book_id = '%s';",Book_ID);
                  res = PQexec(conn,Query);
```

```
if (PQntuples(res) > 0)
{
        sprintf(Query,"select * from Bachelor where u_id =
        '%s';",User_ID);
        res = PQexec(conn,Query);
        if  (PQntuples(res) > 0 )
        {
           Table_Name = "Bachelor     ";
        }
        PQclear(res);
        sprintf(Query,"select * from Master_Degree where u_id =
        '%s';",User_ID);
        res = PQexec(conn,Query);
        if (PQntuples(res) > 0 )
        {
          Table_Name = "Master_Degree ";
        }

        PQclear(res);
        sprintf(Query,"select * from Student where u_id = '%s';",User_ID);
        res = PQexec(conn,Query);
        if (PQntuples(res) > 0 )
        {
          Table_Name = "Student      ";


        }
        PQclear(res);

        sprintf(Query,"select * from Teacher where u_id = '%s';",User_ID);


        res = PQexec(conn,Query);
        if (PQntuples(res) > 0)
        {
          Table_Name = "Teacher      ";
```

```
        }
        PQclear(res);

        sprintf(Query,"select * from Users where u_id = '%s';",User_ID);
        res = PQexec(conn,Query);
        if (PQntuples(res) > 0)
        {
Table_Name =  "Users        ";
        }
        PQclear(res);

        printf("\n Table Name = %s \n ",Table_Name);
        if (Table_Name != "12345678901234")
        {
        sprintf(Query,"select Borrow(%s,'%s','TB000002','now') from
        %s;",Table_Name,User_ID,Table_Name);
          res = PQexec(conn,Query);
        sprintf(Query,"DECLARE myportal CURSOR FOR select * from
        text_book where U_ID = '%s'",User_ID);
          res = PQexec(conn,Query);
          PQclear(res);

          sprintf(Query,"FETCH ALL in myportal");
          res = PQexec(conn,Query);
          nFields = PQnfields(res);

for (i=0;i < nFields;i++)
{
        printf("%-15s",PQfname(res,i));
          }
          printf("\n");

          for (i=0;i < PQntuples(res); i++) {
        for(j=0;j < nFields;j++) {
```

```
                                printf("%-15s",PQgetvalue(res,i,j));
                            }
                            printf("\n");
                        }
                        PQclear(res);
                    res = PQexec(conn,"CLOSE myportal");
                        PQclear(res);
                    }
                    break;
                }
                else
                {
                    printf(" <====== This book you can't borrow ======>\n");
                  break;
                }
              }
            else
            {
              printf("\n <====== No found book in database ======>\n");
              printf(" <====== Please try again  ======>\n");
            }
          }
        }
        else
        {
          printf("\n <====== Not found User_ID in database ! ======>\n");
        }
        res = PQexec(conn,"END");
        PQclear(res);
        break;
    }
}
if (ckey == '2')
{
```

```
res = PQexec(conn,"BEGIN");
PQclear(res);
printf("\n Press Enter To continue \n");
delkey = getchar();
printf("Book ID to Restore : ");
gets(Book_ID);
sprintf(Query,"select * from book* where book_id = '%s';",Book_ID);
res = PQexec(conn,Query);
if      (PQntuples(res) > 0 )
{
        PQclear(res);
        sprintf(Query,"select * from text_book where book_id = '%s';",Book_ID);
        res = PQexec(conn,Query);
        if (PQntuples(res) > 0)
        {
          PQclear(res);
          sprintf(Query,"select restore('%s');",Book_ID);
          res = PQexec(conn,Query);
          PQclear(res);
          sprintf(Query,"DECLARE myportal CURSOR FOR select * from text_book where
book_id = '%s';",Book_ID);
          res = PQexec(conn,Query);
          PQclear(res);
          res = PQexec(conn,"FETCH ALL in myportal");
          nFields = PQnfields(res);
          for (i=0;i < nFields; i++) {
                printf("%-15s",PQfname(res,i));
          }
          printf("\n");
          for (i=0; i < PQntuples(res); i++) {
                for (j=0; j < nFields; j++) {
                  printf("%-15s",PQgetvalue(res,i,j));
                }
                printf("/n");
```

```
                }
                PQclear(res);

                res = PQexec(conn,"CLOSE myportal");
                PQclear(res);
            }
            else
            {
                PQclear(res);
                printf("This book can't borrow so this book can't restore\n");
            }
        }
        else
        {
            PQclear(res);
            printf(" <===== Not book in database =====>\n");
        }
        res = PQexec(conn,"END");
        PQclear(res);
    }
    if (ckey == '3')
    {
        printf(" Press enter to continue \n");
        delkey = getchar();
        printf(" Add new user into database \n");
        printf(" 1.Users                \n");
        printf(" 2.Student              \n");
        printf(" 3.Bachelor Degree      \n");
    printf(" 4.Master Degree        \n");
    printf(" 5.Teacher              \n");
    printf(" 6.Exit                 \n\n\n\n\n\n");
        printf(" Please choice number of type user : ");
        delkey = getchar();
        printf(" Press Enter to continue \n");
```

```
tempkey = getchar();
switch(delkey)
{
        case '1': {
                        printf("\n User ID : ");
                        gets(User_ID);
                        printf(" First Name : ");
                        gets(Fname);
                        printf(" Last Name : ");
                        gets(Lname);
                        printf(" Birthday [MM-DD-YYYY] : ");
                        gets(Bdate);
                        printf(" Sex [M,F] : ");
                        gets(Sex);
                        sprintf(Query,"Insert into Users values( \
                                        '%s','%s','%s','%s','%s' \
                                        )",User_ID,Fname,Lname,  \
                                        Bdate,Sex);
                        res = PQexec(conn,Query);
                        PQclear(res);
                        printf("Add users OK \n");
                        break;
                }
        case '2': {
                        printf("\n User ID : ");
                        gets(User_ID);
                        printf(" First Name : ");
                        gets(Fname);
                        printf(" Last Name : ");
                        gets(Lname);
                        printf(" Birthday [MM-DD-YYYY] : ");
                        gets(Bdate);
                        printf(" Sex [M,F] : ");
                        gets(Sex);
```

```
                printf(" Major : ");
                gets(Major);
                printf(" Faculty :");
                gets(Faculty);
                printf(" Year : ");
                gets(User_Year);
                sprintf(Query," Insert into student values (\

'%s','%s','%s','%s','%s','%s','%s','%s');",User_ID,Fname,Lname,Bdate,Sex,Major,Faculty,User_
Year);
                res = PQexec(conn,Query);
                PQclear(res);
                printf("Add Student OK \n");
                break;
            }
        case '3': {
            printf("\n User ID : ");
                gets(User_ID);
                printf(" First Name : ");
                gets(Fname);
                printf(" Last Name : ");
                gets(Lname);
                printf(" Birthday [MM-DD-YYYY] : ");
                gets(Bdate);
                printf(" Sex [M,F] : ");
                gets(Sex);
                printf(" Major : ");
                gets(Major);
                printf(" Faculty :");
                gets(Faculty);
                printf(" Year : ");
                gets(User_Year);
            printf(" Class : ");
                gets(Class_Room);
```

```
                printf(" Department : ");
                gets(Department);
                sprintf(Query," Insert into bachelor values (\
                '%s','%s','%s','%s','%s','%s','%s','%s',    \

        '%s','%s');",User_ID,Fname,Lname,Bdate,Sex,Major,Faculty,User_Year,Class_Room,Departm
ent);
                        res = PQexec(conn,Query);
                        PQclear(res);
                printf("Add Bachelor OK \n");
                break;
                }
        case '4':{
                printf("\n User ID : ");
                        gets(User_ID);
                        printf(" First Name : ");
                        gets(Fname);
                        printf(" Last Name : ");
                        gets(Lname);
                        printf(" Birthday [MM-DD-YYYY] : ");
                        gets(Bdate);
                        printf(" Sex [M,F] : ");
                        gets(Sex);
                        printf(" Major : ");
                        gets(Major);
                        printf(" Faculty :");
                        gets(Faculty);
                        printf(" Year : ");
                        gets(User_Year);
                printf(" Type Of Master Degree : ");
                gets(Mtype);
                sprintf(Query," Insert into master_degree values (\
                '%s','%s','%s','%s','%s','%s','%s','%s','%s'\
                );",User_ID,Fname,Lname,Bdate,Sex,Major,Faculty,User_Year,Mtype);
```

```
                    res = PQexec(conn,Query);
                    PQclear(res);
                printf("Add Master Degree OK \n");
                break;
            }
        case '5':{
                    printf("\n User ID : ");
                    gets(User_ID);
                    printf(" First Name : ");
                    gets(Fname);
                    printf(" Last Name : ");
                    gets(Lname);
                    printf(" Birthday [MM-DD-YYYY] : ");
                    gets(Bdate);
                    printf(" Sex [M,F] : ");
                    gets(Sex);
                printf(" Degree : ");
                gets(Degree);
                    printf(" Faculty :");
                    gets(Faculty);
                sprintf(Query," Insert into teacher values (\

'%s','%s','%s','%s','%s','%s','%s');",User_ID,Fname,Lname,Bdate,Sex,Degree,Faculty);
                    res = PQexec(conn,Query);
                    PQclear(res);
                    printf("Add Teacher OK \n");
                    break;
            }
            case '6' : break;
        }
        }
    if (ckey == '4')
    {
      printf("Press enter to continue \n");
```

```
delkey = getchar();
printf(" Add new book into database \n");
printf(" 1.Book                \n");
printf(" 2.Text Book           \n");
printf(" 3.Magazine            \n");
printf(" 4.Exit              \n\n\n\n\n\n");
printf(" Please choice number of book type : ");
    delkey = getchar();
    printf(" Press Enter to continue \n");
    tempkey = getchar();
    switch(delkey)
    {
        case '1': {
                printf("\n Book ID : ");
                gets(Book_ID);
                printf(" Title : ");
                gets(Title);
                sprintf(Query,"Insert into Book values( \
        '%s','%s',NULL,NULL,NULL)",Book_ID,Title);
                    res = PQexec(conn,Query);
                    PQclear(res);
                printf("Add Book OK \n");
                    break;
        }

 case '2': {
                printf("\n Text Book ID [TBxxxxxx] : ");
                gets(Book_ID);
                printf(" Title : ");
                gets(Title);
                printf(" Author : ");
                gets(Author);
                printf(" Edition : ");
                gets(Edition);
```

```
                printf(" Publisher : ");
                gets(Publisher);
                sprintf(Query,"Insert into text_book
values('%s','%s',NULL,NULL,NULL,'%s','%s','%s')",Book_ID,Title,Author,Edition,Publisher);
                res = PQexec(conn,Query);
                PQclear(res);
                printf(" Add Text Book OK \n");
                break;
            }

        case '3': {
                printf("\n Magazine ID [MGxxxxxx] : ");
                gets(Book_ID);
                printf(" Title : ");
                gets(Title);
                printf(" Issue : ");
                gets(Issue);
                printf(" Magazine Year : ");
                gets(Book_Year);
                sprintf(Query,"Insert into Magazine values( \
            '%s','%s',NULL,NULL,NULL,'%s',  \
            '%s')",Book_ID,Title,Issue,Book_Year);
                res = PQexec(conn,Query);
                PQclear(res);
                printf("Add Magazine OK \n");
                break;
            }
        case '4': break;
        }
        }
    if (ckey == '5')
    {
      printf("Press enter to continue \n");
      delkey = getchar();
```

```
printf(" Show Data in table  \n");
printf(" 1.Users            \n");
printf(" 2.Student          \n");
printf(" 3.Bachelor Degree   \n");
printf(" 4.Master Degree     \n");
printf(" 5.Teacher          \n");
printf(" 6.Exit     \n\n\n\n\n\n\n\n");
printf(" Please choice number of table to show : ");
delkey = getchar();
printf(" Press Enter to continue \n");
tempkey = getchar();
    switch(delkey)
    {
            case '1': {
      sprintf(Query,"select * from Users");
      show_data(Query);
      break;
            }
       case '2': {
      sprintf(Query,"select * from student");
      show_data(Query);
      break;
            }
  case '3': {
      sprintf(Query,"select * from bachelor");
      show_data(Query);
      break;
      }
 case '4': {
      sprintf(Query,"select * from master_degree");
      show_data(Query);
      break;
      }
 case '5' : {
```

```
        sprintf(Query,"select * from teacher");
        show_data(Query);
        break;
        }
 case '6' : break;
 }
  }
  if (ckey == '6')
  {
printf("Press enter to continue \n");
delkey = getchar();
printf(" Show Data in table  \n");
printf(" 1.Book            \n");
printf(" 2.Text Book       \n");
printf(" 3.Magazine        \n");
printf(" 4.Exit    \n\n\n\n\n\n\n\n");
printf(" Please choice number of table to show : ");
delkey = getchar();
printf(" Press Enter to continue \n");
tempkey = getchar();
    switch(delkey)
    {
            case '1': {
      sprintf(Query,"select * from Book");
      show_data(Query);
      break;
            }
  case '2': {
      sprintf(Query,"select * from text_book");
      show_data(Query);
      break;
            }

  case '3': {
```

```
                sprintf(Query,"select * from Magazine");
                show_data(Query);
                break;
                }
            case '4': break;
                }
             }
             printf("\n Press Enter to Continue ");
             delkey = getchar();
             printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
             printf("=========================================================\n");
             printf("=                MENU                    =\n");
             printf("=========================================================\n");
             printf("=        1.Borrow  Book                  =\n");
             printf("=        2.Restore Book                  =\n");
             printf("=        3.Add User into database          =\n");
        printf("=        4.Add Book into database          =\n");
             printf("=        5.Show data in users             =\n");
             printf("=        6.Show data in book              =\n");
             printf("=        7.Exit                          =\n");
             printf("=========================================================\n");
             printf("\n\n\n\n\n\n");
             printf("Press number [1,2,3,4,5,6,7] to process : ");
             ckey = getchar();
      if (ckey == '7') break;
     }
     printf("Exit to main program \n");
     printf("Good bye \n");
     exit_nicely(conn);
}
```

กิตติกรรมประกาศ

โครงงานนี้สำเร็จได้ด้วยความช่วยเหลือและความร่วมจากท่านผู้มีพระคุณทั้งหลายดังต่อไปนี้

- ขอขอบพระคุณ รศ.ดร.ศุภมิตร จิตตะโศธร อาจารย์ที่ปรึกษาโครงงานที่ ให้การสนับสนุนและให้คำปรึกษาแนะนำในด้านต่างๆเป็นอย่างดี
- ขอขอบพระคุณ ท่านอาจารย์ทุกท่านที่ได้ให้ความรู้ตั้งแต่เล็กจนโต
- ขอขอบพระคุณ คุณพ่อและคุณแม่ ที่เลี้ยงดูจนเติบโตเป็นคนดี
- ขอขอบพระคุณ พี่แจ้ นักศึกษาปริญญาโท ที่ให้ข้อมูลเกี่ยวกับ POSTGRES DBMS
- ขอขอบคุณ น้องเอ ที่ช่วยในการพิมพ์รายงาน
- และสุดท้ายขอบคุณเพื่อนๆห้อง 3P ทุกคนที่มีความเป็นห่วง ถามไถ่เกี่ยวกับความคืบหน้าของโครงงานเสมอมา

บรรณานุกรม

1. Ramez Elmasri , Shamkant B.Navathe , "Fundamentals of Database System" , The Banjamin/Cammings Publishing Company Inc.
2. J.G Hughs , The University of Ulster , "Object-Oriented Database" , Prentice Hall
3. Setrag Khoshafian , "Object-Oriented Databases" , John Wiley & Sons Inc.