สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การพัฒนาโปรแกรมประยุกต์ในเชิงวัตถุ

(Object Oriented Application Development)

T034090

โดย

นางสาวจันทรกานต์ จุหอม

นางสาวทิพย์เนตร แก้วปัตตะ

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เลขหมู่.....
เลขทะเบียน 34090
วัน, เดือน, ปี - 5 ต.ค. 2542

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาโปรแกรมประยุกต์ในเชิงวัตถุ

OBJECT ORIENTED APPLICATION DEVELOPMENT

ผู้จัดทำ

1. นางสาวจันทรกานต์ จุหอม รหัสประจำตัว 39013232
2. นางสาวทิพย์เนตร แก้วปัตตะ รหัสประจำตัว 39013237

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาโปรแกรมประยุกต์ในเชิงวัตถุ

นางสาวจันทรกานต์ จุหอม
นางสาวทิพย์เนตร แก้วปัตตะ

ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา

ปีการศึกษา 2541

บทคัดย่อ

การพัฒนาโปรแกรมประยุกต์ด้วยวิธีการเชิงวัตถุเป็นแนวทางใหม่ในการพัฒนา โดยมุ่งเน้นที่จะเพิ่มประสิทธิภาพในการพัฒนา ซึ่งวิธีการนี้ทำให้การวิเคราะห์, การออกแบบและการเขียนโปรแกรมมีความใกล้ชิดกันมากยิ่งขึ้น

ปริญญานิพนธ์ฉบับนี้เป็นการพัฒนาโปรแกรมประยุกต์ของระบบขาย (Order Entry System) โดยการพัฒนาจะเริ่มจากการวิเคราะห์และออกแบบในเชิงวัตถุ โดยเป็นไปตามหลักแนวคิดของยูเอ็มแอล (UML : Unified Modeling Language) เมื่อทำการวิเคราะห์และะออกแบบเรียบร้อยแล้วจากนั้นเริ่มทำการเขียนโปรแกรมโดยใช้บอร์แลนด์เดลไฟ 4 (Borland Delphi 4 Client/Server Suit) ในส่วนของระบบการจัดการฐานข้อมูลที่ใช้ในระบบนี้คือ ออราเคิล 8 ซึ่งเป็นระบบการจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ (ORDBMS : Object Relational Database Management System) ที่มีชนิดข้อมูลที่ผู้ใช้สามารถกำหนดได้เอง (user – defined data type) ซึ่งข้อมูลชนิดนี้จะมีคุณสมบัติในเชิงวัตถุเช่นเดียวกับโปรแกรมภาษาในเชิงวัตถุ แต่คุณสมบัติบางประการยังไม่เทียบเท่ากับโปรแกรมภาษาเหล่านั้น

เพื่อเพิ่มขีดความสามารถและประสิทธิภาพระบบงานของโปรแกรมประยุกต์ จึงได้นำระบบฐานข้อมูลแบบมัลติเทียร์ไคลเอนท์เซิฟร์เวอร์ (Multitier Client/Server) มาใช้ในระบบงาน โดยการแชร์ข้อมูลและการติดต่อระหว่างเครื่องคอมพิวเตอร์ที่เป็นไคล์เอนท์และเครื่องคอมพิวเตอร์ที่เป็นเซิฟร์เวอร์จะทำได้โดยผ่านทางระบบแลน (LAN) ซึ่งโปรแกรมภาษาเดลไฟ 4 สามารถรองรับระบบไคลเอนท์เซิฟร์เวอร์แบบหลายระดับได้ โดยความสามารถของไมดาส (MIDAS : Miti-tier Distributed Application Services)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

OBJECT ORIENTED APPLICATION DEVELOPMENT

Jantarakan Chuhom

Thipphanet Kaewpatta

Dr. Worawat Limpoka Adviser

1998

ABSTRACT

Object modeling is a strong step toward giving application developers the tools they need to support enterprise applications more effectively.

This thesis is concerned with an Object Oriented Application Development. An application has developed in this thesis is an Order Entry System. In the beginning of the development is Object Oriented analysis and design using Unified Modeling Language (UML). The programming language that used for developing is Delphi 4 Client/Server Suit and the Database management system is Oracle 8. Oracle 8 is an Object Relational Database Management System (ORDBMS) that has a new datatype. This datatype is the user defined type that has Object Oriented properties but they can't do all of the abilities of Object Oriented.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูปภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

การเขียนโปรแกรมในเชิงวัตถุ (OOP : Object Oriented Programming) เกิดขึ้นมาประมาณ 30 ปี แล้วตั้งแต่ปี ค.ศ. 1968 โดยภาษาแรกคือ Simula 68 ซึ่งใช้ในการทำ Simulation หรือการเลียนแบบโลกความจริงมาอยู่ในรูปของคอมพิวเตอร์ และได้รับการปรับปรุงมาเรื่อยๆ จนถึงปี ค.ศ. 1995 เมื่อทั่วโลกต่างพูดถึงการเขียนโปรแกรมในเชิงวัตถุกันอย่างจริงจัง โดยเริ่มต้นจากการเผยแพร่ของ Gary Booch โปรแกรมภาษาในเชิงวัตถุรุ่นแรกๆ เช่น SmallTalk , Effel และ PLC

การเขียนโปรแกรมในเชิงวัตถุเป็นภาษาที่พยายามนำสองสิ่งมาอัดรวมกันเป็นหนึ่งเดียวหรือการเอ็นแคปซูเลชั่น (Encapsulation) คือข้อมูล (data) และวิธีการ (method) ซึ่งแตกต่างจากการเขียนโปรแกรมแบบโครงสร้าง ที่นำข้อมูลไปป้อนใส่วิธีการทำ (procedure) เพื่อเอาผลกลับออกมา ซึ่งในระยะยาวยากแก่การออกแบบ ดังนั้นการเขียนโปรแกรมในเชิงวัตถุก็คือการนำข้อมูลและวิธีการทำมารวมกันในโครงสร้างเดียว ภาษาใหม่อย่าง C++ , DELPHI และ JAVA มีการพัฒนารูปแบบออกมาในทำนองนี้ ส่วนระบบพัฒนาอื่นๆ ที่ออกมาก่อนหน้านี้นั้น ไม่ว่าจะเป็น Visual Basic , Power Builder นั้นเป็น แค่พื้นฐานของการเขียนโปรแกรมในเชิงวัตถุ (object based) และภาษาเหล่านี้ไม่ได้บังคับการเขียนออกมาในรูปแบบของการเขียนโปรแกรมในเชิงวัตถุ รวมทั้งไม่มีโครงสร้างของ OOP ในลักษณะที่เป็น Source Code

ตามกฎของ ANSI (American National Standard Institute) แล้วภาษาที่ถือเป็น Object oriented แท้ๆ นั้น ต้องมีรูปแบบที่เป็นประโยชน์อย่างแท้จริง และต้องถูกกฎเกณฑ์ต่างๆ ทั้งสี่ประการ คือ

Abstraction ความสามารถในการเขียนอธิบายโครงสร้างชีวิตจริงได้ (MODEL OF THE REAL WORLD) และต้องเป็นโครงสร้างบังคับของภาษาแท้ๆ ไปเลย อย่างเช่นใน JAVA เช่น

```
Public class F1 extends DecoratedFrame { }
```

หรือใน DELPHI

```
Type
    F1 = Class(Tform)
End;
```

กล่าวง่ายๆ ก็คือ เมื่อมีโปรแกรมก็ต้องเป็นโครงสร้างรูปแบบของคลาส ไม่มีข้อยกเว้นทั้งการออกแบบโปรแกรมแบบวิชวลหรือตอนรัน ปัจจุบันเราจะเห็นโปรแกรมใหม่มีการเขียนโปรแกรมในเชิงวัตถุทั้งหมดไม่ว่าจะเป็น JDesignPro , JenAva , JBuilder , OptimaC++ , C++Builder , Clarion

Encapsulation โครงสร้างของโปรแกรมต้องจบในตัวเองเป็นก้อน ทำงานเป็นอิสระ และทำงานได้ด้วยตนเอง ทุกๆ คลาสจะต้องมีครบในตัว (Self-Contained) และทำงานได้อย่างเป็นเอกเทส แต่ในขณะเดียวกันก็ต้องสามารถแยกส่วนออกมาและทำงานได้เป็นอิสระเช่นกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Inheritance หมายถึงการถ่ายทอดกันได้ถ้ามีออบเจ็กต์ที่สร้างไว้แล้ว ซึ่งไม่ว่าจะสืบทอดไปกี่ชั้นก็สามารถเพิ่มความสามารถไปได้เรื่อยๆ

Polymorphism ความสามารถในการแตกหน่อในวิธีการทำและเอกลักษณ์ เช่น กำหนดออบเจ็กต์ให้บินได้ ทั้งแมลงวันและนกก็จะบินได้เหมือนกันแต่บินคนละแบบซึ่งแตกต่างกันโดยสิ้นเชิงแต่ทั้งคู่มาจากการบินของออบเจ็กต์เดียวกัน สมมติว่าเป็น TFlyObject แต่มีการแตกหน่อไปคนละสายพันธุ์ได้เพื่อมีรูปแบบที่แตกต่างออกไป

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 ศึกษาแนวความคิดขอออบเจ็กต์โอเรียนเต็ด (Object Oriented Concept) ทั้งทางด้านการวิเคราะห์ (Analysis) การออกแบบ (Design) และการเขียนโปรแกรม

1.2.2 ศึกษาระบบงานจำลองที่วิเคราะห์และออกแบบโดยใช้เทคนิคและความสามารถของแนวความคิดของออบเจ็กต์โอเรียนเต็ด และการสร้างคอมโปเนนท์ต่างๆ

1.2.3 ทำพัฒนาระบบงานชนิดออบเจ็กต์โอเรียนเต็ดเชื่อมต่อกับระบบฐานข้อมูลในรูปแบบของระบบไคลเอนท์เซิร์ฟเวอร์แบบมัลติเทียร์ (Multitier Client/Server)

1.3 ขอบเขตงานวิจัย

ในปริญญานิพนธ์ฉบับนี้จะเป็นการพัฒนาโปรแกรมประยุกต์ในเชิงวัตถุ โดยเริ่มตั้งแต่วิเคราะห์และออกแบบระบบโดยยึดหลักแนวความคิดของยูเอ็มแอล (UML: Unified Modeling Language) การเขียนโปรแกรมซึ่งใช้ภาษาเดลไฟ 4 (Delphi 4 Client/Server Suit) และใช้ระบบการจัดการฐานข้อมูลของบริษัทออราเคิล คือ ออราเคิล 8 (Oracle 8) ซึ่งเป็นระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (ORDBMS : Object Relational Database Management System) นอกจากนั้นยังทำการพัฒนาโปรแกรมประยุกต์ให้เป็นระบบไคลเอนท์เซิร์ฟเวอร์แบบมัลติเทียร์อีกด้วย

โปรแกรมประยุกต์ที่ทำการสร้างขึ้นนี้เป็นระบบการสั่งซื้อสินค้า (Order Entry System) ซึ่งเกี่ยวกับงานของฝ่ายขาย โดยได้ทำการจำลองมาจากระบบการขายของบริษัทไอซีซี (ICC) ในระบบบันทึกข้อมูลการขายนี้จะเป็นระบบการขายแบบมาสแชนเนล (Mass channel) คือลักษณะของสินค้าไม่ต้องการผู้แนะนำ ดังนั้นบริษัท ICC จึงขายสินค้าแบบขายขาดให้ร้านค้า

ในส่วนของระบบงานของโปรแกรมประยุกต์จะเป็นระบบงานแบบมัลติเทียร์ไคลเอนท์เซิร์ฟเวอร์ซึ่งสามารถทำการรันในเครื่องที่ต่างกันได้ โดยจะทำการแชร์ข้อมูลและติดต่อกันผ่านทางแลน (LAN) ซึ่งอาจเรียกว่า “three-tiered model” จะแบ่งออกเป็นสามส่วนคือ

- Client application จะประกอบด้วยส่วนที่ใช้ติดต่อกับผู้ใช้บนเครื่องของผู้ใช้เอง โดยจะใช้ระบบปฏิบัติการ Windows 98
- Application server จะอยู่ตรงกลางของระบบเครือข่ายซึ่งสามารถติดต่อกับไคลเอนท์ได้ทุกเครื่องและจะมีเป็นส่วนที่ให้บริการการใช้ข้อมูลร่วมกัน (Common data service) โดยจะใช้ระบบปฏิบัติการ Windows NT Server

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Remote database server: จะเป็นส่วนของระบบฐานข้อมูล โดยจะใช้ระบบปฏิบัติการ Windows NT Server โดยใช้ระบบจัดการฐานข้อมูล Oracle 8

ในระบบไคลเอนท์เซิร์ฟเวอร์แบบหลายระดับ แอปพลิเคชั่นเซิร์ฟเวอร์จะเป็นส่วนจัดการการไหลของข้อมูลระหว่างไคล์เอนท์และรีโมทดาต้าเบสเซิร์ฟเวอร์ (remote database server) ดังนั้นอาจเรียกเซิร์ฟเวอร์ได้ว่าเป็นดาต้าโบรกเกอร์ (data broker)

Delphi 4 สามารถสนับสนุนการทำงานของระบบไคลเอนท์เซิฟร์เวอร์แบบหลายระดับได้โดยพื้นฐานของ MIDAS (Multitier Distributed Application Service Suit)

1.4 วิธีการดำเนินการ

การศึกษาจะเริ่มจากการศึกษาแนวความคิดทั้งหมดของการเขียนโปรแกรมในเชิงวัตถุ ทั้งการวิเคราะห์และการออกแบบ รวมทั้งโปรแกรมภาษาต่างๆ ที่เป็นมีการเขียนในเชิงวัตถุ จากนั้นทำการศึกษารูปแบบในการวิเคราะห์และออกแบบในเชิงวัตถุ ซึ่งได้ทำการศึกษาในหลายๆ โมเดล เช่น Coad's Model , Bootch , Jacobson และ UML แต่โมเดลที่นำมาทำการวิเคราะห์และออกแบบระบบของโปรแกรมประยุกต์จะใช้โมเดลของยูเอ็มแอล จากนั้นทำการศึกษาระบบการจัดการฐานข้อมูล โดยได้ทำการเลือกระบบการจัดการฐานข้อมูล Oracle 8 มาใช้ในเก็บข้อมูลต่างๆ ของระบบบันทึกข้อมูลการขาย และได้ทำการเลือกโปรแกรมภาษาเดลไฟมาใช้ในการสร้างโปรแกรมประยุกต์ โดยได้ทำการศึกษาความสามารถของเดลไฟในด้านของการเชื่อมต่อระบบมัลติเทียร์ไคลเอนท์เซิร์ฟเวอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
แนวความคิดของออบเจ็กต์โอเรียนเต็ด
(Object Oriented Concept)

2.1 ความหมายของ Object-Oriented Programming

โอโอพี หรือ OOP เป็นคำย่อของ Object oriented Programming ซึ่งหมายถึง วิธีการเขียนโปรแกรมแบบออบเจ็กต์ หลักการสำคัญของโอโอพีก็คือ การสร้างข้อมูลให้มีชนิดเป็นออบเจ็กต์ แล้วดำเนินการประมวลผลออบเจ็กต์เพื่อให้ได้ผลลัพธ์ตามต้องการ

การเขียนโปรแกรมเชิงวัตถุแบบ OOP (Object Oriented Programming) คือวิธีการเขียนโปรแกรมคอมพิวเตอร์โดยใช้แนวความคิดเสมือนการมองภาพวัตถุ (object) ในโลกแห่งความเป็นจริง เป็นการจัดโครงสร้างของโปรแกรมให้เป็นระเบียบมากยิ่งขึ้น และเอื้ออำนวยต่อการพัฒนาโปรแกรมในรุ่นต่อๆ ไป OOP เป็นวิธีการจัดแบ่งประเภทของวัตถุในทางนามธรรม (abstract) ออกเป็นกลุ่มๆ (classes) ซึ่งในแต่ละกลุ่มจะมีสถานะ (states) และพฤติกรรม (behaviors) เฉพาะของตนเอง เพื่อเป็นต้นแบบให้แก่วัตถุที่จะถูกสร้างขึ้นใหม่ในอนาคต ข้อมูลหรือคุณสมบัติเฉพาะ (characteristic) จะถูกเก็บซ่อน (Encapsulation) ไว้ภายในกลุ่มไม่ให้ปะปนกับกลุ่มอื่น วัตถุสามารถสื่อสารซึ่งกันและกันโดยใช้ข่าวสาร (message) และที่สำคัญวัตถุสามารถสืบทอด (Inheritance) คุณสมบัติจากบรรพบุรุษไปสู่ลูกหลานได้

2.2 คุณสมบัติของโอโอพี

2.2.1 แอบสแตร็คชั่น (Abstraction)

แอ็บสแตร็คชั่น คือ ความสามารถในการเขียนอธิบายโครงสร้างชีวิตจริงได้และต้องเป็นโครงสร้างบังคับของภาษาแท้ๆ ไปเลย อย่างเช่นใน JAVA (ไม่ใช่ JavaScript) เช่น

```
Public class F1 extends DecoratedFrame { }
```

หรือใน DELPHI

```
Type
    F1 = Class(Tform)
End;
```

กล่าวง่ายๆ ก็คือ เมื่อมีโปรแกรมก็ต้องเป็นโครงสร้างรูปแบบของคลาส ไม่มีข้อยกเว้นทั้งการออกแบบโปรแกรมแบบวิชัวร์หรือตอนรัน ปัจจุบันเราจะเห็นโปรแกรมใหม่มีการเขียนโปรแกรมแบบ Object ทั้งหมดไม่ว่าจะเป็น JDesignPro , JenAva , JBuilder , OptimaC++ , C++Builder , Clarion

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2.2 เอ็นแคปซูเลชั่น (Encapsulation)

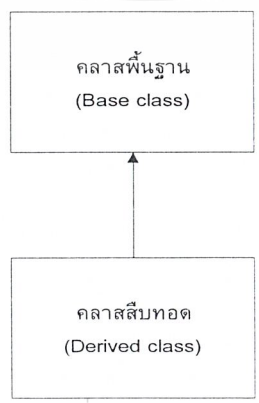
เอ็นแคปซูเลชั่นคือ โครงสร้างของโปรแกรมต้องจบในตัวเองเป็นก้อน ทำงานเป็นอิสระ และทำงานได้ด้วยตนเอง ทุกๆ คลาสจะต้องมีครบในตัว (Self-Contained) และทำงานได้อย่างเป็นเอกเทศ แต่ในขณะเดียวกันก็ต้องสามารถแยกส่วนออกมาและทำงานได้เป็นอิสระเช่นกัน

การรวมข้อมูลเข้ากับโพรซีเยอร์และฟังก์ชันเพื่อให้เป็นข้อมูลชนิดออบเจ็กต์ซึ่งจะแสดงให้เห็นถึงความสัมพันธ์ระหว่างข้อมูลภายในและเม็ทธอดของออบเจ็กต์ด้วย โดยออบเจ็กต์หนึ่งจะมีคุณสมบัติเหมือนสิ่งของอย่างหนึ่งคือ มีทั้งส่วนประกอบและการทำงาน ซึ่งสามารถใช้งานได้ทันที ซึ่งการรวมกันของข้อมูลทำให้เกิดความสามารถในการป้องกันการเข้าถึงข้อมูลภายในออบเจ็กต์จากฟังก์ชันภายนอก (Data hiding) ซึ่งวิธีการในการป้องกันนั้นจะอยู่ที่ชนิดของการประกาศข้อมูลภายในคลาส ซึ่งมีอยู่สามชนิด คือ

- Private การประกาศข้อมูลแบบนี้จะมีผลทำให้ไม่มีคำสั่งใดที่จะอ้างถึงข้อมูลภายในคลาสนั้นได้เลย นอกจากเม็ทธอด (method) ภายในคลาสนั้นเอง ซึ่งตามปกติแล้วจะมีอย่างน้อยหนึ่งเม็ทธอดที่ไม่ได้ประกาศให้เป็นข้อมูลชนิดนี้ เพื่อที่จะได้ใช้ในการเข้าถึงข้อมูลภายในคลาสได้
- Public การประกาศข้อมูลแบบนี้จะมีผลทำให้สามารถอ้างอิงข้อมูลภายในคลาสได้โดยอิสระ
- Protect การประกาศข้อมูลแบบนี้จะมีผลทำให้มีเฉพาะคลาสสืบทอด (Derived class) เท่านั้นที่จะอ้างอิงได้

2.2.3 การสืบทอด (Inheritance)

การสืบทอด คือ การที่คลาสเก่าที่มีอยู่แล้วเป็นต้นแบบให้กับคลาสใหม่ที่จะถูกสร้างขึ้น เกิดการถ่ายทอดคุณสมบัติจากคลาสแม่ (base class) ไปสู่คลาสลูก (derived class) นอกจากคลาสลูกสามารถสืบทอดคุณสมบัติทุกอย่างมาจากคลาสแม่แล้วในขณะเดียวกันคลาสลูกก็สามารถที่จะเพิ่มคุณสมบัติให้กับตัวเองได้อีกด้วย เช่น การเพิ่มตัวแปรหรือเม็ทธอดให้กับตัวเอง คลาสลูกสามารถที่จะใช้ชื่อของเม็ทธอดเหมือนกันกับชื่อเม็ทธอดของคลาสแม่ได้เพื่อเพิ่มความสามารถพิเศษให้แก่เม็ทธอดเดิม (Over riding)

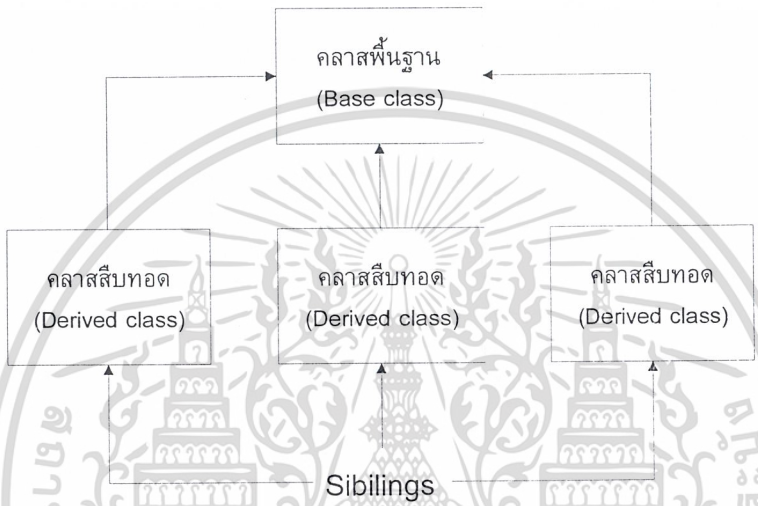


รูปที่ 2-1 แสดงการสืบทอดจากคลาสพื้นฐานหนึ่งคลาส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การสืบทอดแบบนี้นับว่าเป็นคุณสมบัติเด่นของการเขียนโปรแกรมแบบ OOP เพราะทำให้เราสามารถใช้โค้ดเดิมได้โดยไม่ต้องมีการเขียนโปรแกรมที่เคยเขียนไว้แล้วขึ้นมาใหม่อีก เพราะคอมไพเลอร์ (compiler) จะค้นหาโค้ดเองเมื่อทำการคอมไพล์โปรแกรม ผู้ใช้สามารถกำหนดต้นแบบของคลาส (abstract classes) เพื่อให้ผู้ใช้คนอื่นๆ นำไปพัฒนาต่อโดยการเพิ่มรายละเอียดเฉพาะของตัวเองเข้าไป

- คลาสพื้นฐาน (Base class) บางครั้งเรียกคลาสแม่ (Super class)
- คลาสสืบทอด (derived class) บางครั้งเรียกคลาสลูก (Sub class)



รูปที่ 2-2 แสดงการสืบทอดมากกว่าหนึ่งคลาสจากคลาสพื้นฐานหนึ่งคลาส

2.2.4 โพลีมอร์ฟิซึ่ม (Polymorphism)

โพลีมอร์ฟิซึ่มคือ ความสามารถที่จะใช้ได้ทั่วไปคุณสมบัติที่เมื่อออบเจ็กต์ต่างๆ ได้รับคำสั่งเดียวกันจากโปรแกรมแล้วแต่ละออบเจ็กต์จะทำงานตามแบบของตัวเอง ซึ่งทำให้ได้ผลลัพธ์แตกต่างกัน ซึ่งเป็นคลาสที่ได้รับการถ่ายทอดคุณสมบัติจากคลาสเดียวกัน การที่คลาสในกลุ่มความสัมพันธ์นี้จะทำงานด้วยวิธีที่ต่างกันด้วยเมสเสจอันเดียวกันคลาสหลักจะต้องประกาศเม็ทธอดที่จะใช้เป็นเวอร์ชวลเม็ทธอด การใช้เม็ทธอดเดียวกันจะทำงานต่างกันตามชนิดของคลาส

2.2.4.1 เม็ทธอดแบบสแตติก (Static method)

เม็ทธอดแบบสแตติกมีลักษณะการทำงานคล้ายกับการทำงานของโพรซีเยอร์และฟังก์ชันทั่วไปในปาสคาล กล่าวคือ เมื่อเม็ทธอดเหล่านี้เรียกใช้เม็ทธอดอื่นจะมีการเรียกซ้ำเม็ทธอดเดิมนั้นเสมอ ที่เป็นเช่นนี้เนื่องจากได้มีการนำเม็ทธอดเหล่านี้ผูกติดกันตั้งแต่ตอนคอมไพล์โปรแกรม ซึ่งเรียกกระบวนการนี้ว่า เออร์ลีไบน์ดิงก์ (Early binding – การผูกติดกันตั้งแต่ตอนแรก) ผลของเออร์ลีไบน์ดิงก์ ก็คือ ทำให้เม็ทธอดที่เรียก เม็ทธอดอื่นรู้ตั้งแต่ตอนคอมไพล์ว่าจะเรียกเม็ทธอดนั้นจากที่ใด และตำแหน่งการเรียก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เม็ทธอดจะคงที่อยู่ ณ ที่นั้นตลอดไป แสดงว่าเม็ทธอดแบบสแตติกมีการกำหนดตำแหน่งการเรียกเม็ทธอดต่างๆ ไว้เป็นการแน่นอนตั้งแต่ตอนคอมไพล์และไม่สามารถเปลี่ยนแปลงได้อีกเลย

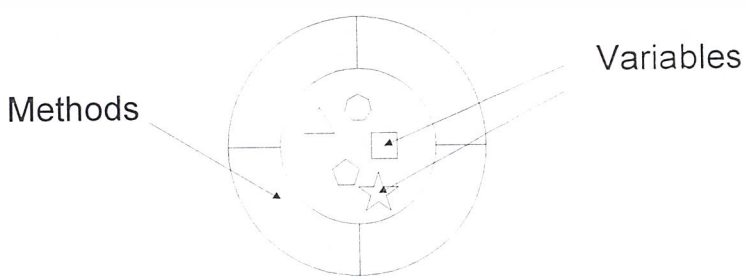
2.2.4.2 เม็ทธอดแบบเวอร์ชวล (Virtual method)

เม็ทธอดแบบนี้จะไม่ผูกติดกับเม็ทธอดอื่นที่เกี่ยวข้องตั้งแต่ตอนคอมไพล์ แต่จะมีการผูกติดกันในตอนรันโปรแกรม ซึ่งเราเรียกกระบวนการนี้ว่า เลตไบน์ดิงก์ (Late binding – การผูกติดกันในตอนหลัง) ผลของเลตไบน์ดิงก์ทำให้เม็ทธอดที่เรียกเม็ทธอดอื่นไม่รู้ล่วงหน้าว่าจะเรียกเม็ทธอดนั้นได้จากที่ไหน แต่จะรู้ในตอนรันโปรแกรม แสดงว่าเม็ทธอดแบบเวอร์ชวลจะไม่ผูกติดกับเม็ทธอดใดๆ ไว้ในลักษณะแน่นอนตายตัวตั้งแต่ตอนคอมไพล์แต่จะมีการเรียกเม็ทธอดต่างๆ เปลี่ยนแปลงไปตามความเหมาะสมในตอนรันโปรแกรม

2.3 ออบเจ็กต์ (Objects)

ออบเจ็กต์ คือ ซอฟท์แวร์ซึ่งประกอบด้วยข้อมูลและวิธีการจัดการกับข้อมูลนั้น ออบเจ็กต์เป็นกุญแจสำคัญในการทำความเข้าใจการเขียนโปรแกรมแบบ OOP โดยถ้าจะยกตัวอย่างของออบเจ็กต์ในโลกความเป็นจริง เช่น สุนัขเป็นสิ่งมีชีวิตที่มีสถานะ เช่น พันธุ์ สี ขนาด เป็นต้นและมีพฤติกรรมที่ตอบสนองสถานะ เช่น สามารถเห่าได้ สามารถดมกลิ่นได้ เป็นต้น และอีกตัวอย่างหนึ่งคือ หลอดไฟเป็นสิ่งของที่มีสถานะ เช่น รูปทรง กำลังวัตต์ สถานะของสวิทช์ไฟเปิดปิด เป็นต้น และมีพฤติกรรม เช่น การส่องสว่าง การเกิดความร้อน เป็นต้น สำหรับการเขียนโปรแกรมแบบ OOP เราจะแทนสถานะด้วยตัวแปร (variables) และแทนพฤติกรรมด้วยเม็ทธอด ดังรูปที่ 2-3

จากรูปที่ 2-3 จะเห็นได้ว่าตัวแปรของออบเจ็กต์ถูกจัดให้อยู่ตรงกลางของวงกลม ออบเจ็กต์และมีเม็ทธอดล้อมรอบเพื่อทำหน้าที่เก็บซ่อนส่วนกลางของออบเจ็กต์ไว้ ประโยชน์ของการเอ็นแคปซูเลชั่นนี้คือ ถ้าหากเราต้องเปลี่ยนแปลงข้อมูลภายในออบเจ็กต์ การเปลี่ยนแปลงจะไม่ส่งผลต่อออบเจ็กต์อื่นๆ ที่อยู่ภายนอกเพราะข้อมูลไม่ขึ้นต่อกัน เราจึงสามารถเปลี่ยนแปลงและปรับปรุงส่วนของโปรแกรมที่ต้องการการเปลี่ยนแปลงได้ ในขณะที่ผู้อื่นก็สามารถเปลี่ยนแปลงไปได้อย่างอิสระและประโยชน์อีกข้อหนึ่งก็คือ เป็นการจัดการกับข้อมูลภายในออบเจ็กต์ว่าข้อมูลส่วนนี้ต้องการที่จะเปิดเผยหรือไม่ ซึ่งสามารถกำหนดได้โดยการประกาศคุณสมบัติให้กับออบเจ็กต์

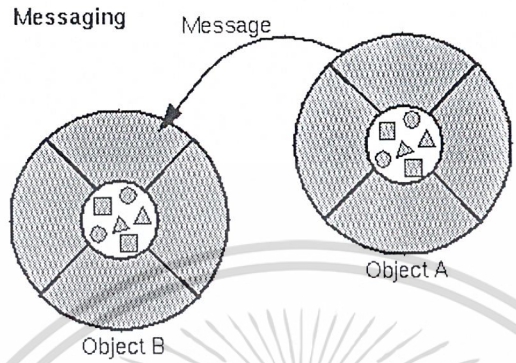


รูปที่ 2-3 แสดงการจำลองส่วนประกอบของออบเจ็กต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4 ข่าวสาร (Message)

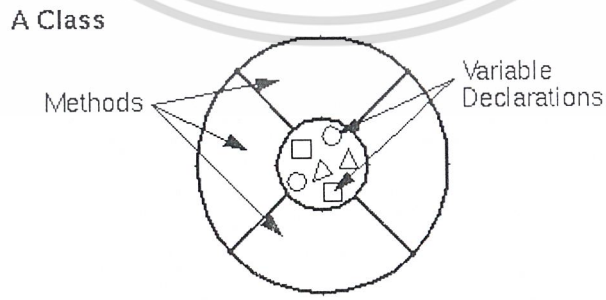
การติดต่อสื่อสารและแลกเปลี่ยนข้อมูลระหว่างออบเจ็กต์ทำได้โดยการรับส่งข่าวสาร ยกตัวอย่างเช่น เมื่อออบเจ็กต์ A ต้องการทำงานกับเม็ทธอด B ออบเจ็กต์ A ก็จะส่งข่าวสารไปบอกแก่ออบเจ็กต์ B ดังรูปที่ 2-4



รูปที่ 2-4 การส่งข่าวสารระหว่างออบเจ็กต์

2.5 คลาสและอินสแตนซ์ (Class and Instance)

คลาส คือ ชนิดของออบเจ็กต์ซึ่งจะแสดงถึงลักษณะเฉพาะของออบเจ็กต์ คลาสจะถูกกำหนดขึ้นด้วยการเขียนรายละเอียดของคลาส โดยปกติคลาสจะถูกออกแบบให้มีความสัมพันธ์กับคลาสอื่น ซึ่งคลาสหนึ่งคลาสสามารถเป็นต้นแบบให้แก่ออบเจ็กต์ ได้หลายออบเจ็กต์ การที่เราสร้างออบเจ็กต์ขึ้นมาหนึ่งออบเจ็กต์หรือหลายออบเจ็กต์เราเรียกว่า อินสแตนซ์ (instance) เราต้องสร้างอินสแตนซ์ขึ้นมาจากคลาสก่อนที่จะมีการใช้ตัวแปรหรือเม็ทธอดของคลาสนั้น ซึ่งจะเรียกอินสแตนซ์ว่าเป็นออบเจ็กต์เมื่อมีการเปลี่ยนแปลงข้อมูล หรือเรียกใช้เม็ทธอดของอินสแตนซ์นั้นเกิดขึ้น



รูปที่ 2-5 รายละเอียดของคลาส

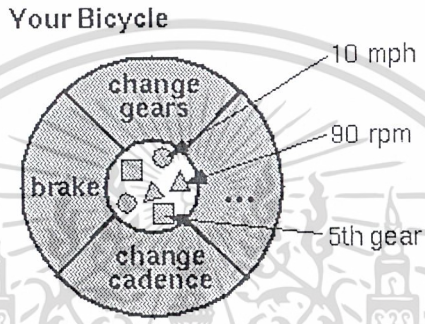
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.6 ตัวอย่างการออกแบบจักรยาน

ในตัวอย่างนี้จะเป็นการออกแบบในเชิงวัตถุ ซึ่งจะเป็นการออกแบบจักรยานที่มีการกำหนดตัวแปรดังต่อไปนี้

สภาวะของจักรยาน

1. จะวิ่งด้วยความเร็ว 10 เมตรต่อ 1 ชั่วโมง
2. ความเร็วรอบของที่วางเท้า 90 รอบต่อนาที
3. มีทั้งหมด 5 เกียร์



รูปที่ 2-6 ไดอะแกรมของตัวอย่างการออกแบบ

2.6.1 Method

กำหนดให้จักรยานมีความสามารถดังต่อไปนี้

1. ยอมให้มีการเบรก
2. ยอมให้มีการเปลี่ยนความเร็วรอบของที่วางเท้า
3. ยอมให้มีการเปลี่ยนเกียร์

ทุกสิ่งที่ตัวออบเจ็กต์เองไม่รู้หรือว่าไม่สามารถทำได้จะถูกแยกจากตัวออบเจ็กต์ เช่น ถ้าจักรยานที่ไม่มีชื่อไม่สามารถวิ่งได้ให้เอาออกมาเสีย จากรูปที่ 2-6 ตัวแปรของออบเจ็กต์จะอยู่ตรงส่วนกลางหรือตรงส่วนที่เป็นจุดศูนย์กลางของออบเจ็กต์ เม็ทธอดจะอยู่รอบๆ และจะปิดบังจุดศูนย์กลางของออบเจ็กต์จากออบเจ็กต์อื่นๆ ในโปรแกรม

การเก็บตัวแปรของออบเจ็กต์ไว้ภายในตัวป้องกันของเม็ทธอดเรียกว่าเอ็นแคปซูเลชั่นซึ่ง lation นี้จะใช้สำหรับปิดบังรายละเอียดของตัวดำเนินการที่ไม่สำคัญนักจากออบเจ็กต์อื่น เช่น เมื่อเราต้องการที่จะเปลี่ยนเกียร์ของจักรยานเราไม่จำเป็นต้องรู้ว่าระบบของเกียร์ทำงานอย่างไรบ้าง เพียงแต่เรารู้วิธีการที่จะเลื่อนคันโยกสำหรับเปลี่ยนเกียร์ก็พอแล้ว

2.6.2 Encapsulation

Encapsulation จะสัมพันธ์กับ ตัวแปรและเม็ทธอดถ้ากำหนดสิ่งต่างๆ เหล่านี้ได้อย่างเหมาะสม ซอร์ฟแวร์ก็จะมีความสามารถสูง ซึ่งประโยชน์ของ Encapsulation มีสองประการ คือ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. Modularity ซอร์สโค้ดของออบเจ็กต์จะสามารถเขียนและปรับปรุงแก้ไขได้โดยเป็นอิสระจากออบเจ็กต์อื่น ดังนั้นการส่งผ่านออบเจ็กต์ไปในระบบสามารทำได้โดยง่าย จากตัวอย่างการออกแบบ คือ เราสามารถให้จักรยานกับใครก็ได้โดยที่จักรยานยังสามารถใช้งานได้อยู่

2 Information hiding ออบเจ็กต์มีการติดต่อกันได้อย่างเปิดเผย ซึ่งออบเจ็กต์ทุกๆ ออบเจ็กต์สามารถติดสื่อสารกันได้ แต่ออบเจ็กต์สามารถเก็บรักษาหรือเปลี่ยนแปลงข้อมูลและเม็ทธอดได้เฉพาะของตัวออบเจ็กต์เอง โดยออบเจ็กต์หนึ่งอาจเปลี่ยนแปลงตลอดเวลาโดยไม่มีผลกระทบกับออบเจ็กต์อื่นที่ขึ้นอยู่กับตัวมัน จากตัวอย่างการออกแบบ คือ เราไม่ต้องเข้าใจระบบการทำงานของเกียร์แต่เราก็ใช้งานมันได้

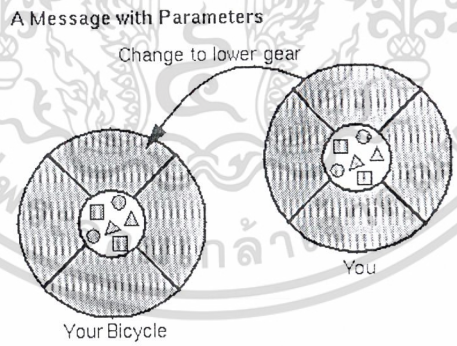
2.6.3 Messages

ในบางครั้งที่ออบเจ็กต์จะรับเมสเสจออบเจ็กต์ก็ต้องการข้อมูลที่มากพอเพื่อที่จะได้ทราบว่าจะต้องทำอะไร จากตัวอย่างการออกแบบ เมื่อต้องการเปลี่ยนเกียร์จะต้องทราบว่าจะเปลี่ยนเป็นเกียร์อะไร ข้อมูลนี้จะถูกส่งไปกับเมสเสจเหมือนพารามิเตอร์

ส่วนประกอบของเมสเสจ ประกอบด้วย

1. ออบเจ็กต์ที่เมสเสจมีการระบุตำแหน่ง (จักรยาน)
2. ชื่อของเม็ทธอดที่จะทำ (เปลี่ยนเกียร์)
3. พารามิเตอร์ต่างๆ ที่เม็ทธอดต้องใช้

ทั้งสามข้อเป็นข้อมูลที่เพียงพอที่จะให้ออบเจ็กต์ปฏิบัติตามเม็ทธอดที่ได้ตั้งไว้ โดยไม่จำเป็นต้องมีข้อมูลอื่นใดอีก

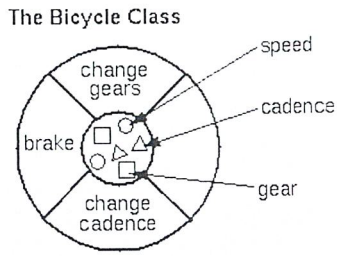


รูปที่ 2-7 การส่งเมสเสจที่มีพารามิเตอร์

2.6.4 Classes

เราต้องสร้างคลาสของจักรยานโดยประกาศตัวแปรให้ใช้ได้หลายคนเพื่อเก็บสถานะของเกียร์ปัจจุบัน , ที่วางเท้าสำหรับปั่น เป็นต้น ซึ่งจะต้องประกาศและจัดหาตัวดำเนินการเพื่อเม็ทธอดที่ยอมให้เปลี่ยนเกียร์ได้ , เบรคได้ และเปลี่ยนความเร็วรอบของบันไดได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2-8 คลาสไดอะแกรมของจักรยาน

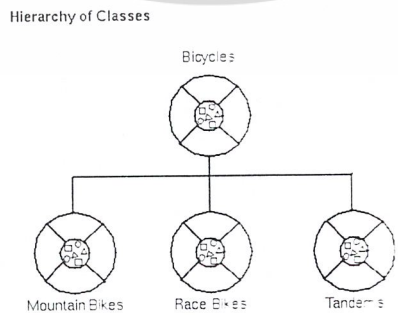
ค่าของตัวแปรจะถูกกำหนดโดยอินสแตนซ์ของคลาส ดังนั้นหลังจากที่ได้สร้างคลาสของจักรยานจะต้องทำการสร้างอินสแตนซ์ของคลาสด้วย ถ้าทำการสร้างอินสแตนซ์จาก คลาสตัวแปรที่ประกาศไว้โดยคลาสจะต้องเก็บไว้ในหน่วยความจำ จากนั้นจะสามารถใช้อินสแตนซ์ของเม็ทธอดเพื่อกำหนดค่าให้ตัวแปร

2.6.5 Iheritanc

ใน OOP จะทำการสืบทอดอย่างเป็นขั้นตอนและจะยอมให้คลาสหนึ่งถูกกำหนดเป็นเทอมของคลาสอื่นได้ เช่น จักรยานแบบเสือภูเขา , แบบไว้สำหรับการแข่งขัน และแบบซ้อนท้ายได้ ซึ่งแต่ละชนิดจะมีความแตกต่างกัน ซึ่งใน OOP จะถือว่าจักรยานทั้งสามแบบเป็นสับคลาส (sub class) ของคลาสจักรยาน ซึ่งคลาสจักรยานเป็นซุปเปอร์คลาส (super class) ของจักรยานทั้งสามแบบ

บางสับคลาสจะสืบทอดสถานะจากซุปเปอร์คลาส ซึ่งจักรยานแบบเสือภูเขา , แบบไว้สำหรับการแข่งขัน และแบบซ้อนท้ายได้จะมีสภาวะต่างๆ เหมือนกัน คือ จังหวะและความเร็ว และในบางสับคลาสจะสืบทอดเม็ทธอดจากซุปเปอร์คลาส คือ จักรยานแบบเสือภูเขา , แบบไว้สำหรับการแข่งขัน และแบบซ้อนท้ายได้จะมีคุณสมบัติเหมือนกันคือ การเบรกและการเปลี่ยนความเร็วของบันไดจักรยาน

สภาวะของสับคลาสมีไม่จำกัด สับคลาสสามารถเพิ่มตัวแปรและเม็ทธอด เช่น จักรยานแบบซ้อนท้ายได้ อาจจะเพิ่มที่จับเป็นสองที่ เพิ่มเบาะนั่งเป็นสองเบาะก็สามารถทำได้ และจักรยานเสือภูเขาอาจเพิ่มชุดเกียร์เข้าไป



รูปที่ 2-9 ลำดับชั้นของคลาส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.7 การเปรียบเทียบข้อแตกต่างของ Object Oriented Programming กับ Procedural Structure Programming

ข้อแตกต่างที่สำคัญของการเขียนโปรแกรมในเชิงออบเจ็กต์กับการเขียนโปรแกรมแบบโครสร้าง คือ

2.7.1 วิธีการจัดเก็บข้อมูล (Method for accomplishing actions with data)

OOP : เมื่อออบเจ็กต์หนึ่งต้องการทำงานกับเม็ทธอดของออบเจ็กต์อื่น การทำงานกับเม็ทธอดของออบเจ็กต์นั้นจะส่งเมสเสจไปที่ออบเจ็กต์ที่ต้องการ เมื่อออบเจ็กต์นั้นได้รับเมสเสจก็จะตอบสนองต่อเมสเสจนั้นตามเม็ทธอดที่เหมาะสมกับเมสเสจที่ได้รับ

Procedural : ก่อนที่โพรซีเยอร์จะทำการจัดการกับข้อมูลนั้นๆ จะมีการส่งค่าพารามิเตอร์ไปที่โพรซีเยอร์นั้นก่อน โดยจะต้องกำหนดโพรซีเยอร์ที่ต้องการใช้งานเลย ซึ่งตัวโปรแกรมจะไม่สามารถเลือกการทำงานได้เอง

2.7.2 ความสามารถในการเขียนอธิบายโครงสร้างชีวิตจริง (Abstraction)

OOP : โครงสร้างของข้อมูล (data abstraction) จะถูกแทนโดยคลาสของออบเจ็กต์ ส่วนโครสร้างการทำงาน (function abstraction) จะถูกแทนโดยเมสเสจ

Procedural : โครงสร้างของข้อมูลจะถูกแทนโดยชนิดของข้อมูล (data type) ส่วนโครงสร้างการทำงานจะถูกแทนโดยการทำงานของโพรซีเยอรัล

2.7.3 โครงสร้างของโปรแกรมต้องจบในตัวเอง (Encapsulation)

OOP : มีการรวมข้อมูลเข้ากับโพรซีเยอร์และฟังก์ชั่นเพื่อให้เป็นข้อมูลชนิดออบเจ็กต์ ซึ่งจะแสดงให้เห็นถึงความสัมพันธ์ระหว่างข้อมูลภายในและเม็ทธอดของออบเจ็กต์ด้วย โดยออบเจ็กต์หนึ่งจะมีคุณสมบัติเหมือนสิ่งของอย่างหนึ่งคือ มีทั้งส่วนประกอบและการทำงานซึ่งสามารถใช้งานได้ทันที

Procedural : การเอ็นแคปซูเลชั่นจะอยู่ในรูปของไลบารีซอฟท์แวร์คอมโพเนนท์ (Library Software Component) ซึ่งแต่ละคอมโพเนนท์จะมีมากกว่าหนึ่งโครงสร้างของข้อมูลอยู่ด้วยกัน

2.7.4 การสืบทอดความสัมพันธ์และความหลากหลาย (Inheritance and Polymorphysm)

OOP : สามารถรองรับการสืบทอดและความหลากหลายได้

Procedural : ไม่สามารถรองรับการสืบทอดและความหลากหลายได้

2.7.5 การอนุญาตให้ผู้ใช้กำหนดโครงสร้างของโปรแกรมใหม่ได้ (Extensibility and Relative status of new protocol)

OOP : อนุญาตให้ผู้ใช้สามารถกำหนดโครงสร้างขึ้นมาใหม่ได้ โดยที่โครงสร้างที่สร้างขึ้นมาใหม่จะมีสถานะเทียบเท่ากับที่ระบบเป็นผู้กำหนด

เอกสารนี้เป็นเอกสารสงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Procedural : ถึงแม้ว่าจะยอมให้ผู้ใช้กำหนดโครงสร้างขึ้นมาใหม่ได้ แต่โครงสร้างที่สร้างขึ้นมาใหม่จะมีความสำคัญและประสิทธิภาพน้อยกว่าโครงสร้างที่สร้างจากระบบ

2.7.6 ลำดับชั้นของคลาส (Class hierarchy)

OOP : สามารถรองรับลำดับชั้นของคลาสได้ ดังนั้นจึงมีประโยชน์ในกรณีที่ต้องการเพิ่มความสามารถให้กับลำดับชั้นของคลาสหรือเพิ่มคลาสย่อย

Procedural : แต่ละคอมโพเนนท์ของแต่ละซอฟท์แวร์จะมีสถานะเท่ากันหมด จึงทำให้เกิดความซ้ำซ้อนได้

2.7.7 การส่งผ่านพารามิเตอร์ (Passing Parameter)

OOP : การส่งผ่านพารามิเตอร์เข้า-ออก จะเป็นการส่งผ่านจากออบเจ็กต์หนึ่งไปยังอีกออบเจ็กต์หนึ่ง

Procedural : การส่งผ่นพารามิเตอร์จะส่งผ่านชนิดของข้อมูลเท่านั้น

2.8 ข้อดีของการเขียนโปรแกรมในเชิงวัตถุ

2.8.1 Security : OOP มีการทำอินฟอร์เมชั่นไฮดิงค์ (Information hiding) เนื่องจาก OOP จะเก็บข้อมูลไว้ในออบเจ็กต์ซึ่งจะถูกจัดการโดยเมทธอดที่เหมาะสมเท่านั้น ดังนั้นจึงทำให้ข้อมูลมีความปลอดภัยมากกว่าแบบ procedural

2.8.2 Reusability : OOP สามารถนำคอมโพเนนท์ที่สร้างไว้แล้วกลับมาใช้งานได้อีก เช่น อนุญาตให้โปรแกรมที่ถูกสร้างขึ้นมาใหม่สามารถเรียกใช้เมทธอดจากโปรแกรมเดิมได้

2.8.3 Portability : เนื่องจาก OOP สามารถรองรับคุณสมบัติความหลากหลาย ทำให้ไม่มีปัญหาในการตั้งชื่อเมทธอด ซึ่งสามารถตั้งชื่อซ้ำกันได้เพราะออบเจ็กต์สามารถเลือกตอบสนองเมสเสจโดยใช้เมทธอดที่เหมาะสม

2.8.4 Realibility and Integrity : เนื่องจาก OOP มีคุณสมบัติในการเอ็นแคปซูเลชั่นจึงทำให้เกิดความผิดพลาดได้ยากและความน่าเชื่อถือสูง

2.8.5 Flexibility : เนื่องจาก OOP มีคุณสมบัติในการสืบทอดความสัมพันธ์จึงทำให้มีความยืดหยุ่นมาก คือ สามารถเพิ่มความสามารถให้แต่ละคลาสย่อยได้ โดยอาจเพิ่มที่คลาสต้นตระกูลเท่านั้นทุกๆ คลาสย่อยก็จะถูกเพิ่มความสามารถนั้นๆ ไปด้วย

2.8.6 Understandability : OOP มีลักษณะของโปรแกรมตรงกับความเข้าใจของผู้ใช้ ซึ่งทำให้ผู้ใช้สามารถเข้าใจการทำงานได้โดยง่าย

2.8.7 Maintainability : OOP ง่ายต่อการเพิ่มเติมเมื่อมีความต้องการเพิ่มเติมเมื่อมีความต้องการต่างๆ เพิ่มขึ้น ง่ายต่อการแก้ไขและบำรุงรักษาโปรแกรม

2.8.8 Easy to Development : ในการพัฒนาโปรแกรม OOP ไม่จำเป็นต้องสนใจโค้ดของโปรแกรมว่าเป็นอย่างไร แค่ทราบว่าแค่ส่งเมสเสจนั้นไปในออบเจ็กต์แล้วได้เอาท์พุทออกมาอย่างไรจึงทำให้สามารถพัฒนาโปรแกรมแบบขนานได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.8.9 Vifiability : OOP สามารถตรวจสอบหาข้อผิดพลาดได้ง่าย

2.9 ข้อเสียของการเขียนโปรแกรมในเชิงวัตถุ

1. เนื่องจาก OOP เป็นเทคนิคแบบใหม่ ดังนั้นทูลส์ (tools) ต่างๆ จึงยังไม่ค่อยพัฒนามากนัก โดยเฉพาะทูลส์ที่เป็นการเขียนโปรแกรมในเชิงวัตถุอย่างแท้จริง (pure object oriented programming)
2. ไม่มีสัญลักษณ์ที่เป็นมาตาฐานที่ใช้ในการวิเคราะห์และออกแบบ
3. การพัฒนาทางด้านโปรแกรมภาษายังพัฒนาไปไม่มาก
4. จากคุณสมบัติการสืบทอดอาจจะทำให้เกิดปัญหาในเรื่องความสัมพันธ์ระหว่างคลาสได้
5. จากคุณสมบัติความหลากหลายอาจทำให้การตรวจสอบโปรแกรมมีความยุ่งยากซับซ้อนขึ้น

จะเห็นได้ว่าข้อเสียส่วนใหญ่ของการพัฒนาโปรแกรมในเชิงวัตถุ คือ ยังมีการพัฒนาที่ไม่พอเพียง จากข้อเสียที่กล่าวมาถ้าผู้ทำการพัฒนาได้ทำการศึกษาและทำความเข้าใจเป็นอย่างดีแล้วก็จะสามารถพัฒนาโปรแกรมได้อย่างดี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
การเขียนโปรแกรมเชิงวัตถุกับภาษาปาสคาล
(Object Oriented Programming with Object Pascal)

3.1 What's Object-Oriented Programming ?

OOP (Object-Oriented Programming) คือ รูปแบบของการเขียนโปรแกรมรูปแบบหนึ่งที่ทำการออกแบบโดยใช้สิ่งต่างๆ บนโลกแห่งความจริง เช่น ปุ่มกด , วินโดวส์ , รูปทรงกลม , ลูกเต๋า , ส่วนประกอบของรถยนต์ , ลูกจ้าง ถ้าเป็นซอฟท์แวร์ที่เป็น OOP สิ่งต่างๆ เหล่านี้จะถูกเรียกว่าออบเจ็กต์ (Objects)

3.2 From the OOP Perspective

สิ่งสำคัญพื้นฐานของ OOP คือ ออบเจ็กต์คลาส หรือเรียกว่า คลาส และ ออบเจ็กต์ เช่น คลาสของปุ่มกดจะเป็นคลาสของปุ่มกดทั้งหมด ซึ่งปุ่มกดเหล่านี้จะมีบางสิ่งที่มีร่วมกัน ในการเขียนโปรแกรมปุ่มกดจะเป็นออบเจ็กต์หนึ่งภายในคลาสที่เป็นชนิดเทมเพลท (template) ประกอบด้วย ชื่อปุ่มและตำแหน่งเป็นต้น

ในการสร้างคลาสสามารถสร้างขึ้นใหม่ซึ่งยังคงสัมพันธ์กับคลาสที่สร้างขึ้นก่อนได้ โดยวิธีการที่เรียกว่าสับคลาสซิ่ง (subclassing) สับคลาสหรือผู้สืบทอดจะสืบทอดข้อมูลทั้งหมดและความสามารถทั้งหมดจากบรรพบุรุษ ซึ่งสับคลาสสามารถทำการโอเวอร์ไรด์สิ่งต่างๆ ที่ทำการสืบทอดมาได้และสามารถเพิ่มเติมข้อมูลและความสามารถต่างๆ ได้อีกด้วย

คุณลักษณะของคลาสปุ่มกดก็เหมือนข้อมูลชนิดเรคคอร์ดในภาษาปาสคาล คลาสปุ่มกดจะมีฟิลด์ข้อมูล (เรียกว่า "ตัวแปรอินสแตนซ์" ใน OOP) หลายชนิด เช่น คลาสปุ่มกดอาจมีฟิลด์ที่ทำให้มันมีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้า ซึ่งทุกๆ อินสแตนซ์ของคลาส (ทุกๆออบเจ็กต์ปุ่มกด) จะต้องมีฟิลด์นี้

ในภาษาออบเจ็กต์ปาสคาลคลาสปุ่มกดจะมีเมทธอดอยู่ด้วย ซึ่งเมทธอด คือ โพรซีเยอร์หรือฟังก์ชั่น ซึ่งในแต่ละออบเจ็กต์จะมีทั้งข้อมูลและการกระทำ เมื่อสับคลาสมีการสืบทอดมาจากบรรพบุรุษสับคลาสนั้นก็จะสืบทอดฟิลด์ข้อมูลและเมทธอดทั้งหมดของบรรพบุรุษมา แต่สำหรับเมทธอดนั้นจะมีการโอเวอร์ไรด์เกิดขึ้น

3.3 Messages and methods

ออบเจ็กต์หรืออินสแตนซ์ของคลาสจะทำงานโดยการส่งเมสเสจให้แต่ละออบเจ็กต์ ซึ่งการรับเมสเสจของออบเจ็กต์จะเป็นการปลุกเมทธอดที่ตรงกับความต้องการใช้งาน เช่นเมสเสจที่ต้องการให้แสดงผล ออบเจ็กต์จะทำการปลุกเมทธอด *Display*

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4 Polymorphism

ออบเจ็กต์สามารถส่งเมสเสจเดียวกัน เช่น เมสเสจ *Display* ไปยังออบเจ็กต์อื่นๆ หลายออบเจ็กต์ แม้แต่ในคลาสที่แตกต่างกันได้ (เช่น buttons , integers , employees) การรับเมสเสจของออบเจ็กต์ในแต่ละคลาสจะทำการเรียกใช้เมทธอด *Display* ของตัวออบเจ็กต์เอง แต่ละออบเจ็กต์จะตอบสนองเมสเสจ *Display* เฉพาะที่เป็นของออบเจ็กต์นั้น ออบเจ็กต์ในคลาสปุ่มกดก็จะวาดภาพปุ่มกดบนหน้าจอ ออบเจ็กต์ในคลาสอินทีเจอร์ก็จะพิมพ์ค่าที่เป็นอินทีเจอร์ของออบเจ็กต์นั้นออกมา และออบเจ็กต์ในคลาสลูกจ้างก็จะแสดงที่อยู่ของลูกจ้าง , ตำแหน่งงานและเงินเดือน ซึ่งความสามารถที่ออบเจ็กต์ในคลาสแต่ละคลาสสามารถตอบสนองเฉพาะเมทธอดที่เป็นของตัวออบเจ็กต์เองแม้ว่าจะเป็นเมสเสจที่มีชื่อเดียวกัน เรียกความสามารถนี้ว่าโพลีมอร์ฟิซึ่ม (polymorphism)

3.5 Runtime binding

ในออบเจ็กต์ปาสคาลโปรแกรมสามารถได้รับตัวแปรของออบเจ็กต์ภายในคลาสหนึ่งจากคลาสที่เป็นบรรพบุรุษได้ ดังนั้นออบเจ็กต์ของคลาส *TRadioButton* สามารถได้รับตัวแปรของคลาส *Tbutton* เนื่องจาก *TradioButton* เป็นสับคลาสของคลาส *Tbutton* ตัวแปรของคลาส *Tbutton* สามารถเปลี่ยนแปลงได้หลายรูปแบบโดยการกำหนดให้โปรแกรมส่งเมสเสจ Display ไปยังออบเจ็กต์ที่ตัวแปรจะอ้างถึงในขณะที่ทำการรันโปรแกรม และออบเจ็กต์ของคลาส *TradioButton* จะตอบสนองโดยการแสดงปุ่ม Radio การผูกติดกันภายหลังหรือเลทไบดิงก์ (late binding) ของเมสเสจเพื่อให้ออบเจ็กต์ทำงานกับเมท - ธอดที่เหมาะสมนี้จะเรียกว่ารันไทม์ไบดิงก์ (runtime binding)

3.6 Objects

ออบเจ็กต์ คือ โมดูลที่เก็บดาต้าและโค้ดที่สามารถรวมกันไว้ในออบเจ็กต์ได้ ซึ่งสามารถทำการประกาศออบเจ็กต์คลาสได้ดังนี้

```
type
  objectClassName = Object
      {Data fields – also known as "instance variables"}
      {Procedure and function headings – also known as "method headings"}
  end;
```

3.7 Objects and Records

object type จะประกอบด้วย field list การประการ record type จะไม่มีส่วนที่เป็น heading ของโพรซีเยอร์และฟังก์ชั่น แต่ object type จะต้องมี

Records + Procedures = objects

ออบเจ็กต์จะเหมือนกับเรคคอร์ดมากตรงที่โพรซีเยอร์จะถูกเพิ่ม โดยการเพิ่มโพรซีเยอร์นี้จะรวมเซ็ทของข้อมูลเข้ากับ operations ในโมดูลเดียว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

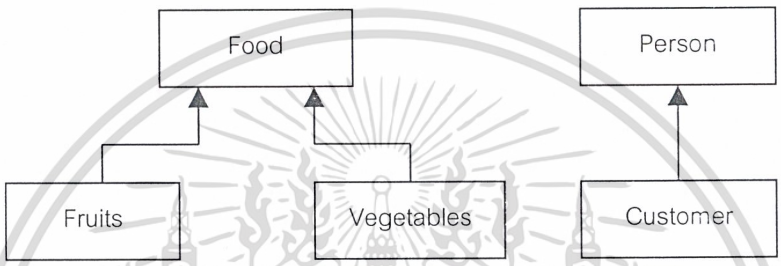
3.7 Object Types and Object Variables

ออบเจ็กต์คือตัวแปรของบาง object type คือเป็นอินแสตนซ์ของคลาส

An object variable (instance) declaration

```
var
    anObject: ItsType;      {Format of a declaration}
```

3.8 Program ตัวอย่างร้านขายผักและผลไม้



รูปที่ 3-1 ลำดับชั้นของคลาส

3.8.1 Data Fields (Instance Variables)

Instance variable คือตัวแปร (data field) ที่เป็นส่วนหนึ่งของอินแสตนซ์ (object) ของบางออบเจ็กต์คลาส

ออบเจ็กต์ **Food**

```
type
  TFood = class(TObject)
  Id :integer;
  Code : string[5];
  Description : string[50];
  OnHand : integer;
  UnitPrice : integer;
   function Sale(Quantity : integer):integer; virtual;
   function Store:integer; virtual;
   procedure
Init(ID:integer;Codes:string;Descriptions:string;OnHands:integer;UnitPrices:integer);
   virtual;
  procedure New;virtual;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

34090

```
    function GetAdd:string;virtual;
    function GetStore:string;virtual;
end;
```

ออบเจ็กต์ Fruit จะสืบทอดมาจากออบเจ็กต์ Food

```
unit Fruit;
interface
uses
Food;
        type
          TFruit = class(TFood);
```

ออบเจ็กต์ Vegetable

```
type
  TVegetable = class(TFood)
   function Store:integer; override;
   end;
```

ออบเจ็กต์ Person

```
type
  TPerson = class(TObject)
   Id : integer;
   FirstName : string[50];
   LastName : string[50];
   Country : string[50];

    procedure Init(Ids:integer;FName:string;LName:string;C:string);virtual;
    procedure New;virtual;
    end;
```

ออบเจ็กต์ Customer

```
type
   Customer = class(TPerson)
    procedure Init(Ids:integer;FName:string;LName:string;C:string);override;
    procedure New;override;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
function Buy(Quantity : integer):integer;
function GetAdd:string;
end;
```

ออบเจ็กต์ Banana

```
uses
  Food,Fruit;
type
  TBanana = class(TFruit);
```

ออบเจ็กต์ Mango

```
uses
  Food,Fruit;
type
  TMango = class(TFruit);
```

ออบเจ็กต์ Carrot

```
uses
  Food,Vegetable;
type
  TCarot = class(TVegetable);
```

3.8.2 Procedure and Function Declarations (Method Headings)

การประกาศออบเจ็กต์คลาสสามารถมี method headings ได้มากเท่าที่ต้องการ ซึ่งแต่ละ heading จะเป็นการประกาศโพรซีเยอร์หรือฟังก์ชั่นที่สมบูรณ์ ซึ่งออบเจ็กต์ไม่สามารถมี variant part ได้

3.8.3 Procedure and Function Implementations (Method bodies)

ออบเจ็กต์ Food

```
implementation
    function TFood.Sale(Quantity : integer):integer;
    begin
if Quantity > Self.OnHand then
Result := -1
else
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    begin
    Self.OnHand := Self.OnHand - Quantity;
    Result := Self.OnHand;
    end;
end;

    function TFood.Store:integer;
    var s : string;
    Quantity,i:integer;
    begin
        s:=inputbox('Store','Quantity','');
        val(s,Quantity,i);
        Self.OnHand := Self.OnHand+Quantity;
        Result := Self.OnHand;
    end;

    procedure
    TFood.Init(ID:integer;Codes:string;Descriptions:string;OnHands:integer;UnitPrices:integer);
    begin
        Self.Id := ID;
        Self.Code := Codes;
        Self.Description := Descriptions;
        Self.OnHand := OnHands;
        Self.UnitPrice := UnitPrices;
    end;

    procedure TFood.New;
    var s1,s2,s3,s4,s5 : string;
        n1,n2,n3,i:integer;
    begin
        s1 := Inputbox('NEW FOOD','ID','');
        val(s1,n1,i);
        s2 := inputbox('NEW FOOD','CODE','');
        s3 := inputbox('NEW FOOd','DESCRIPTION','');
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    s4 := inputbox('NEW FOOD','ON HAND','');
    val(s4,n2,i);
    s5 := inputbox('NEW FOOD','UNIT PRICE','');
    val(s5,n3,i);
    Self.Init(n1,s2,s3,n2,n3);
end;

function TFood.GetAdd:string;
var SqlString,s:string;
begin
    str(Self.id,s);
    SqlString := 'insert into xpoo.foods values';
    SqlString := SqlString + '(xpoo.food_type(';
    SqlString := SqlString + s + ',"';
    SqlString := SqlString + Self.Code + '","';
    SqlString := SqlString + Self.Description + '",';
    str(Self.OnHand,s);
    SqlString := SqlString + s + ',';
    str(Self.UnitPrice,s);
    SqlString := SqlString + s + '))';
    Result := SqlString;
end;

function TFood.GetStore:string;
var SqlString,s:string;
begin
    SqlString := 'update xpoo.foods set onhand = ';
    str(Self.OnHand,s);
    SqlString := SqlString + s;
    SqlString := SqlString + 'where id = ';
    str(Self.Id,s);
    SqlString := SqlString + s;
    Result := SqlString;
end;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ออบเจ็กต์ Vegetable

implementation

```
function TVegetable.Store:integer;
var s : string;
   Quantity,i:integer;
begin
   repeat
      s:=inputbox('Store','Quantity (10 min)','');
      val(s,Quantity,i);
   until Quantity>9;
   Self.OnHand := Self.OnHand+Quantity;
   Result := Self.OnHand;
end;
```

3.9 The self pseduvarible

ในการเขียนโค้ดของเม?ธอดในบางครั้งจะเป็นประโยชน์ในการอ้างถึงออบเจ็กต์ปัจจุบัน คือ เมื่อเมทธอดของออบเจ็กต์กำลังเอ๊กซีคิวท์อยู่เราอาจต้องการให้ออบเจ็กต์นั้นทำการเรียกตัวเอง ทำได้โดยใช้ "pseduvariable" *self* เช่น

```
{ Inside Init method code of object anObject }
self.Display;              { Call one of self's methods }
self.fRect := theRect;     { Access one of self's instance variables }
aVar := self.fLength;
```

Pseudovarible สามารถติดต่อได้เพียงทางเดียวจากภายในเมทธอดของออบเจ็กต์ โดยใช้ *self* เพื่อทำให้เมทธอดโค้ดชัดเจน คือตัวแปรอินแสตนซ์และเมทธอดที่เรากำลังทำการติดต่ออยู่นั้นเป็นออบเจ็กต์เดียวกัน

3.10 Polymorphism

An Example of polymorphism

คลาส *Tapple* สืบทอดตัวแปรอินแสตนซ์ทั้งหมดมาจาก *Tfruit* และ *Tfood* ซึ่งจะทำการสืบทอดเมทธอดทั้งหมดมาด้วย ยกเว้นที่ทำการ override เอาไว้ ชื่อของเมทธอดใน *Tapple* จะพบในคลาสที่เป็นบรรพบุรุษแล้วซึ่งจะแสดงให้เห็นถึงความสำคัญของการแชร์ชื่อของเมทธอด

```
var
   aFood: Tfood;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
aFruit: Tfruit;
anApple: Tapple;
```

เมื่อมีการทำโพลีมอร์ฟีซึ่มจะต้องทำการกำหนดตัวแปรแต่ละตัว

```
New(aFood);              { Allocate memory for an object instance from the heap }
aFood.Init(8);           { Set object's instance variables, etc. }
New(aFruit)
aFruit.Init(8);
New(anApple);
anApple.Init(8);
```

จากนั้นทำการสร้างตัวแปรดังต่อไปนี้

```
aFood := aFruit;         { Fruit descends from food }
aFood := anApple;        { Apple descends from food }
aFruit := anApple;       { Apple descends from fruit }
```

The Power of Polymorphism

- Store multiple, different (but related) types in a variable , an array, or another data structure
- Send the same message (call the same method) in a way that ensures the appropriate action for each item

3.11 Runtime Binding

3.11.1 Early binding

เมทธอดแบบสแตติกมีลักษณะการทำงานคล้ายกับการทำงานของโพรซีเยอร์และฟังก์ชันทั่วไปในปาส คาล กล่าวคือ เมื่อเมทธอดเหล่านี้เรียกใช้เมทธอดอื่นจะมีการเรียกซ้ำเมทธอดเดิมนั้นเสมอ ที่เป็นเช่นนี้เนื่องจากได้มีการนำเมทธอดเหล่านี้ผูกติดกันตั้งแต่ตอนคอมไพล์โปรแกรม ซึ่งเรียกกระบวนการนี้ว่า เออร์ลีไบน์ดิงก์ (Early binding – การผูกติดกันตั้งแต่ตอนแรก) ผลของเออร์ลีไบน์ดิงก์ ก็คือ ทำให้เมทธอดที่เรียกเมทธอดอื่นรู้ตั้งแต่ตอนคอมไพล์ว่าจะเรียกเมทธอดนั้นจากที่ใด และตำแหน่งการเรียกเมทธอดจะคงที่อยู่ ณ ที่นั้นตลอดไป แสดงว่าเมทธอดแบบสแตติกมีการกำหนดตำแหน่งการเรียกเมทธอดต่างๆ ไว้เป็นการแน่นอนตั้งแต่ตอนคอมไพล์และไม่สามารถเปลี่ยนแปลงได้อีกเลย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.11.2 Late Binding

เมทธอดแบบนี้จะไม่ผูกติดกับเมทธอดอื่นที่เกี่ยวข้องตั้งแต่ตอนคอมไพล์ แต่จะมีการผูกติดกันในตอนรันโปรแกรม ซึ่งเราเรียกกระบวนการนี้ว่า เลตไบน์ดิงก์ (Late binding – การผูกติดกันในตอนหลัง) ผลของเลตไบน์ดิงก์ทำให้เมทธอดที่เรียกเมทธอดอื่นไม่รู้ล่วงหน้าว่าจะเรียกเมทธอดนั้นได้จากที่ไหน แต่จะรู้ในตอนรันโปรแกรม แสดงว่าเมทธอดแบบเวอร์ชวลจะไม่ผูกติดกับเมทธอดใดๆ ไว้ในลักษณะแน่นอนตายตัวตั้งแต่ตอนคอมไพล์แต่จะมีการเรียกเมทธอดต่างๆ เปลี่ยนแปลงไปตามความเหมาะสมในตอนรันโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4
การวิเคราะห์และออกแบบโปรแกรมประยุกต์ในเชิงวัตถุโดยใช้ยูเอ็มแอล
(Object Oriented Analysis and Design using UML)

4.1 Introduction

ขั้นตอนในการวิเคราะห์และออกแบบระบบในเชิงวัตถุนั้นมีขั้นตอนในการวิเคาะห์และอกกแบบปลีกย่อยลงมาอีก แสดงดังต่อไปนี้

Analysis ประกอบด้วย

- Define Essential Use Cases
- Refine Use Cases Diagrams
- Refine Conceptual Model
- Refine Glossary
- Define System Sequence Diagrams
- Define Operation Contracts
- Define State Diagrams

Design ประกอบด้วย

- Define Real Use Cases
- Define Reports , UI and Storyboards
- Refine System Architecture
- Define Interaction Diagrams
- Define Design Class Diagrams
- Design Database Schema

รายละเอียดและความหมายต่างๆ แสดงดังต่อไปนี้

เทคนิคที่ทำให้สามารถเข้าใจถึงความต้องการ (requirements) ที่มีต่อระบบคือ การสร้างยูสเคส(use cases) หรือการอธิบายถึงขอบเขตของการปฏิบัติ ซึ่งต่อไปนี้จะแสดงถึงแนวคิดของยูสเคส โดยจะนำตัวอย่างของแอปพลิเคชั่นจุดที่ทำการขายมาบรรยายประกอบ

4.2 Case Study : Point – of – Sale

ตัวอย่างของแอปพลิเคชั่นที่จะใช้อธิบายขั้นตอนการวิเคราะห์และออกแบบระบบนั้นคือระบบ Piont-of-sale terminal (POST) ซึ่งเป็นระบบที่มีการคำณวนโดยใช้บันทึกการขายและลำดับการจ่าย

Architecture Layers and Case Study Emphasis

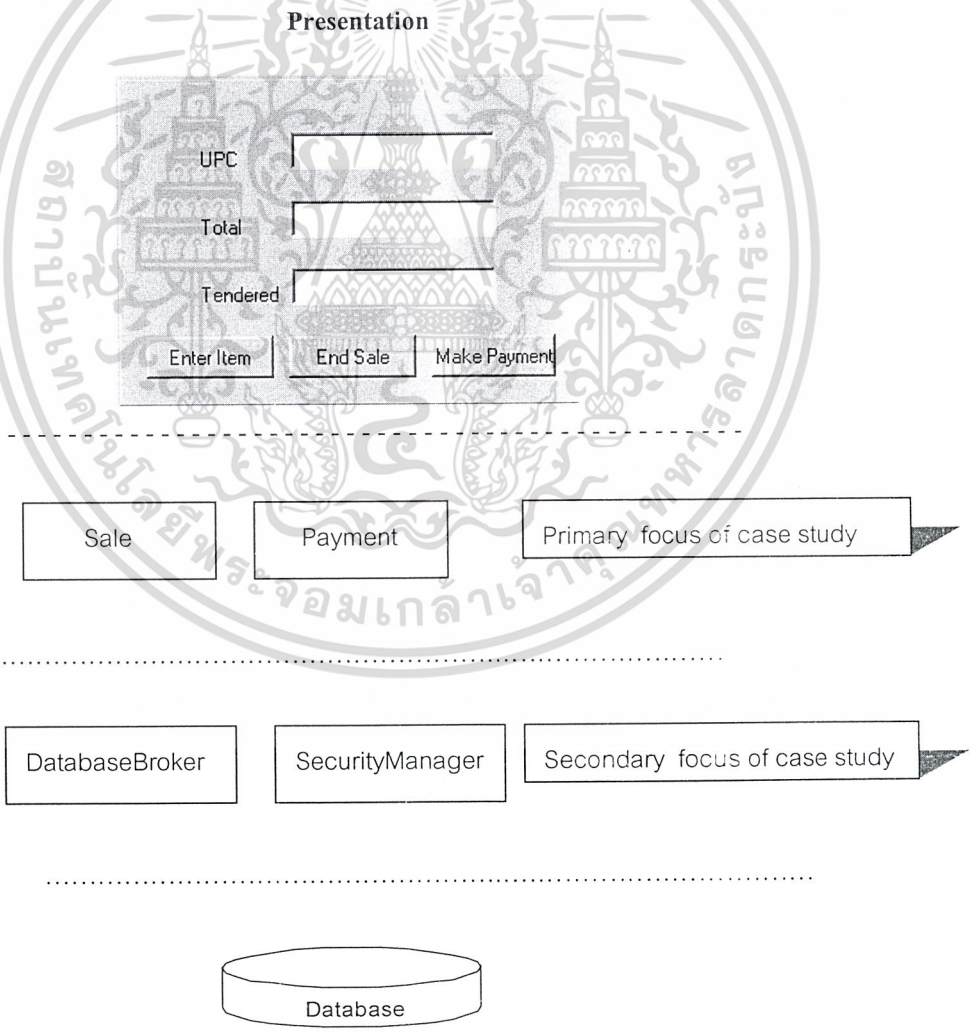
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โดยทั่วไปโครงสร้างของระบบจะประกอบด้วยส่วนที่ติดต่อผู้ใช้ที่เป็นแบบกราฟฟิก และระบบฐานข้อมูล เช่น

- Presentation คือ กราฟฟิกอินเตอร์เฟส (graphic interface) และวินโดวส์ (windows)
- Application Logic – Problem Domain Objects คือ การแสดงถึงออบเจ็กต์ที่เป็นขอบเขตของแนวความคิด (เช่นออบเจ็กต์การขาย : Sale Objects) ที่ตรงกับความต้องการของแอปพลิเคชั่น
- Application Logic – Service Objects คือ ออบเจ็กต์ที่ไม่ใช่ขอบเขตของปัญหา (problem domain) ที่สนับสนุนเซอร์วิส เช่น การเชื่อมต่อกับระบบฐานข้อมูล
- Storage คือ การเก็บข้อมูล เช่น ฐานข้อมูลในเชิงวัตถุหรือฐานข้อมูลเชิงสัมพันธ์

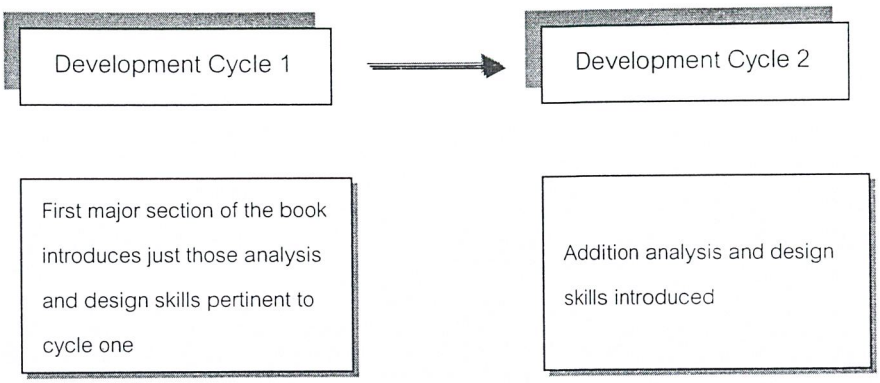
Our Strategy : Learning and Development

ในการวิเคราะห์และออกแบบแอปพลิเคชั่น Point-of-Sale นั้นจะแบ่งการพัฒนาออกเป็นสองส่วน ในส่วนแรกคือ ส่วนของการทำงานทั่วๆ ไป และการพัฒนาในส่วนที่สองคือการทำงานของระบบ



รูป 4-1 Layers in a typical object-Oriented information system

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูป 4-2 Learning path follows development cycles

4.3 Analysis Phase

4.3.1 Define Essebtial Use Cases

Use Cases : Describing Processes

Activities and Dependencies

ยูสเคสจะขึ้นอยู่กับความเข้าใจในความต้องการของระบบ โดยแสดงได้ด้วยรายงานการกำหนดความต้องการของระบบ

Use cases

Use case คือ เอกสารที่ใช้อธิบายถึงเหตุการณ์ต่าง ๆ ของ event ของ actor

รูปที่ 4-3 The UML icon for use case

ตัวอย่างของ High-level Use case : Buy Items

ต้องการอธิบายถึงโพรเซสในการซื้อสินค้าจากร้านค้า เมื่อ point-of-sales terminal ถูกใช้งาน

Use case: **Buy Items**

Actors : Customer , Cashier

Type : primary (to be discussed)

Description : A customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects payment. On completion , the Customer leaves with the items.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Expanded Use Case

Expanded use case แสดงรายละเอียดของ Use case มากกว่า high level ซึ่งทำให้ user สามารถเข้าใจถึง process และ requirement ต่างๆ ที่มีอยู่ในระบบได้ดีกว่า

- Explaning the Expanded Format

  ส่วนบนสุดของ expanded form จะเป็นการรวบรวมข้อมูลต่างๆ ที่มีอยู่ในระบบย่อย

Use case : **Name of use case**

Actors : List of actors (external agents) , indicating who initiates the use case.

Purpose : Intention of the use case.

Overview : Repetition of high-level use case, or some similar summary.

Type : 1. primary, secondary or optional (to be discussed)

2. essential or real (to be discussed)

Cross References : Related use cases and systems functions.

ส่วนกลาง คือ Typical course of events ซึ่งเป็นส่วนที่สำคัญที่สุดของ Expanded format ซึ่งจะอธิบายถึงรายละเอียดในการสนทนาเกี่ยวกับการกระทำต่างๆ ระหว่าง actors และ system

Typical Course of Events

Actor Action	**System Response**
Numbered actions of actors.	Numbered descriptions of system responses

ส่วนสุดท้ายคือ Alternative course of events จะอธิบายimportant alternatives or exceptions ซึ่งอาจจะเกิดขึ้นกับ typical course

Alternatives Courses : Alternatives that may arise at line number. Description of exception

ACTORS

Actor คือ entity ที่เกิดขึ้นภายนอกระบบซึ่งเป็นบุคคลที่เกี่ยวข้องกับ use case

Customer

รูปที่ 4-4 The UML icon for a use case actor.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Identifying Use Cases

ในการกำหนดยูสเคสสามารถกำหนดได้สองวิธี คือ

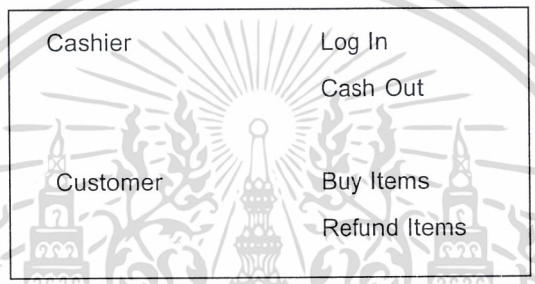
วิธีที่ 1 คือ Actor-based คือการกำหนดยูสเคสตามผู้กระทำ (actor)

1. กำหนดผู้กระทำ (actors) ที่เกี่ยวข้องกับระบบหรือการทำงาน
2. โดยแต่ละผู้กระทำ จะมีการกำหนดการทำงานที่จะต้องมีและผู้กระทำนั้นต้องรู้จัก

วิธีที่ 2 คือ Event-based คือการกำหนดยูสเคสตามการกระทำ (event)

1. ทำการกำหนดอีเวนท์ภายนอกที่ระบบจะต้องทำการตอบสนอง
2. ทำการกำหนดความสัมพันธ์ต่าง ๆ ของผู้กระทำและยูสเคส

สำหรับแอปพลิเคชั่น Point-of-Sale สามารถทำการกำหนดยูสเคสได้ดังนี้



Use Case and Doamain Processes

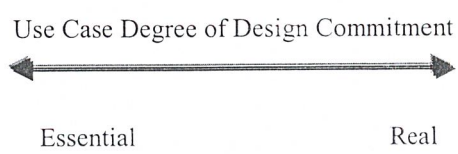
การใช้ยูสเคสอธิบายถึงโพรเซส (process) เช่น โพรเซสเกี่ยวกับธุรกิจ (business process) โดยโพรเซสจะอธิบายถึงการเริ่มต้นจนถึงจุดสิ้นสุด , การเรียงลำดับของอีเวนท์ , ผู้กระทำและทรานส์แอกชั่นที่ต้องทำงานบางงานให้เสร็จสิ้น

4.3.2 Refine Use Case Diagrams

ตัวอย่างของยูสเคสไดอะแกรมของแอปพลิเคชั่นระบบ Point-of-sale แสดงดังรูปที่ 4-5

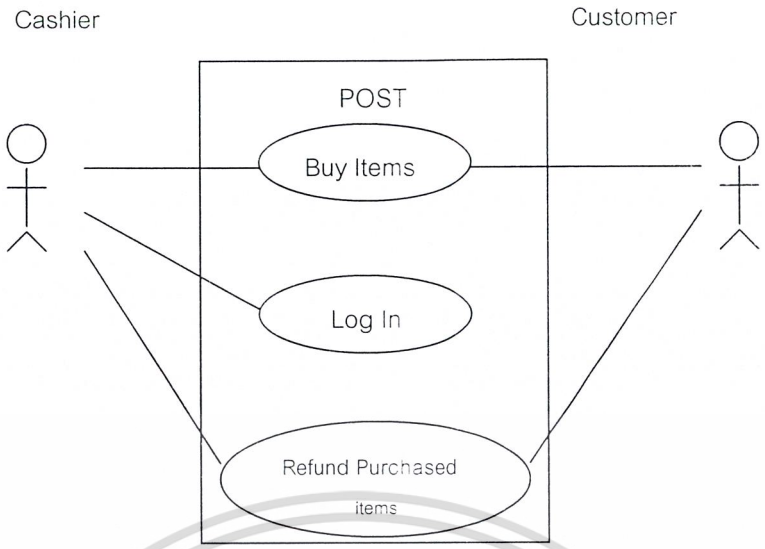
ยูสเคสไดอะแกรมเป็นการอธิบายโดยใช้ภาพประกอบถึง เซ็ทของยูสเคสของระบบ, ผู้กระทำและความสัมพันธ์ระหว่างยูสเคสกับผู้กระทำ ซึ่งยูสเคสจะแสดงด้วยวงรี ส่วน actor จะเป็นรูปภาพ และเส้นตรงจะเป็นส่วนติดต่อระหว่างผู้กระทำกับยูสเคส จุดประสงค์ของไดอะแกรมนี้คือการแสดงถึงลักษณะของไดอะแกรมที่สามารถที่จะเข้าใจระบบได้ง่ายและรวดเร็ว

Essentail versus Real Use Cases



รูปที่ 4-5 Essentail versus real use cases exist on a continuum

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4-6 Partial use case diagram

Essential Use Cases

ยูสเคสที่สำคัญกับระบบหรือเอสเซนเชี่ยลยูสเคส (Essentail use cases) จะเป็นยูสเคสที่อธิบายถึงโพรเซสอย่างละเอียด (expanded use cases)

Real Use Cases

เรียลยูสเคสจะอธิบายถึงโพรเซสในรูปแบบของการการออกแบบจริง เมื่อผู้ใช้ทำการติดต่อกับระบบจะมีการแสดงที่หน้าจอ

Notational Points

- Naming Use Cases (การตั้งชื่อให้ยูสเคส)

  การตั้งชื่อให้ยูสเคสจะเริ่มด้วยคำกริยาที่มีความสำคัญ คือเป็นโพรเซส เช่น

  - Buy Items
  - Enter an Order

- Startting of an Expanded Use Case

  เริ่มต้นการเขียน expanded use case ตามวิธีการดังต่อไปนี้

  1. This Use case begins when <Actor> <initiates an event>

  เช่น

  1. This Use case begins when a Customer arrives at a POST with items to purchase

- Notating Decision Points and Branching

  ในยูสเคสอาจต้องมีส่วนที่ต้องทำการตัดสินใจ เช่นใน *Buy Items* ลูกค้าต้องตัดสินใจว่าจะจ่ายเงินสด หรือใช้บัตรเครดิต ซึ่งจะต้องใช้โครงสร้างในการเขียนดังต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. ในส่วนที่เป็นส่วนหลัก (Section Main) ให้ใส่ *Typical Course of Events*
2. เขียนส่วนย่อยแต่ละส่วนอีกครั้งโดยใช้ Typical Course of Events เริ่มต้นด้วยหมายเลขของเหตุการณ์ในแต่ละส่วน
3. ในแต่ละส่วนย่อยที่มีหลากหลายก็ให้แบ่งย่อยออกไปเรื่อยๆ

Use Cases Within a Development Process

Plan and Elaborate Phase Steps

1. หลังจากที่ทำการกำหนดหน้าที่ของระบบแล้วให้ทำการกำหนดขอบเขตของระบบ และกำหนดผู้กระทำและยูสเคส
2. เขียนยูสเคสด้วยรูปแบบ *high-level* โดยจำแนกว่าเป็น primary หรือ secondary หรือ optional
3. เขียนยูสเคสไดอะแกรม
4. เชื่อมโยงความสัมพันธ์ในยูสเคส
5. เขียน *expended Essentail*
6. เรียลยูสเคสควรต้องเขียนในขั้นตอนการออกแบบ แต่บางเรียลยูสเคสก็จำเป็นต้องสร้างระหว่างการแสดงความต้องการของระบบ
7. เข้าสู่ขั้นตอนของยูสเคส (Rank use cases)

Iterative Development Cycle Phase Steps

1. Analyze phase คือ เขียนยูสเคสในแบบ expanded essentail
2. Design phase คือ เขียนเรียลยูสเคส

Process Steps for the Point-of-Sale System

Identify Actors and Use Cases

Cashier	Login Cash Out
Customer	Buy Items Refund Items
Manager	Start Up Shut Down
System Administrator	Add New Users

Write Use Cases in High-level Format

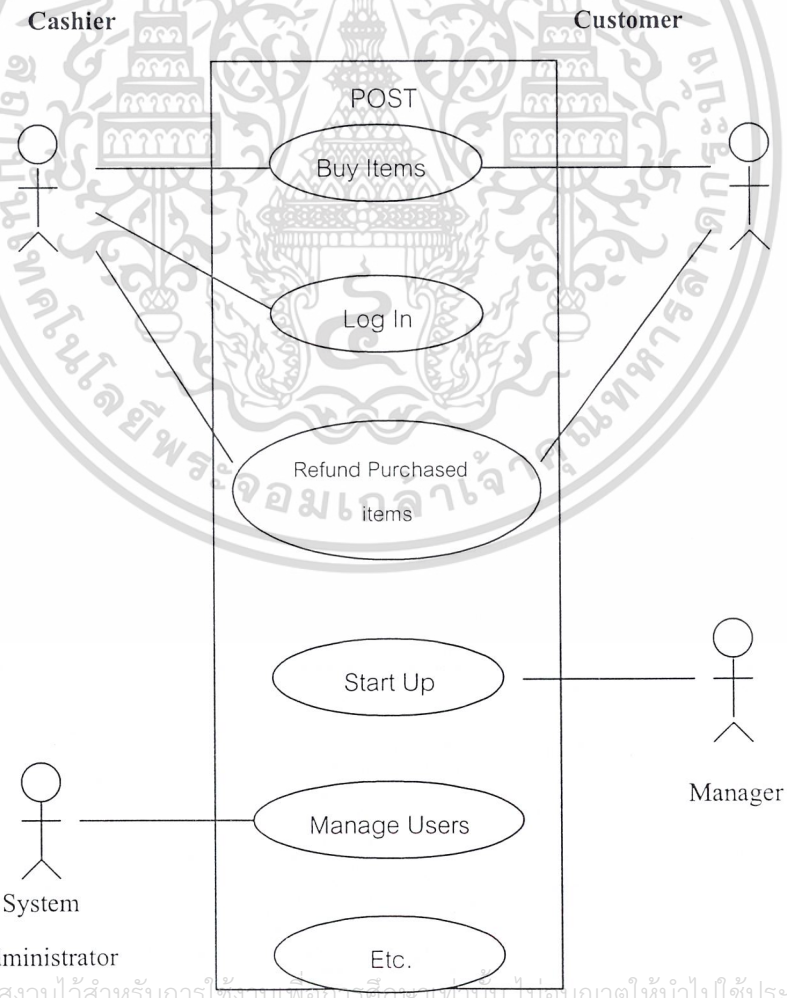
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างของยูสเคสแบบ High-level คือ

Use case : **Buy Items**

Actors:	Costomer (initiator) , Cashier
Type:	primary
Description:	A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a Paymeant. On completion, The Customers leaves with the Items.
Use case :	**Start Up**
Actors:	Manager
Type:	primary
Description:	A Manager powers on a POST in order to prepare it for Use by cahsiers. The Manager validate that the date And time are correct, after which the system is ready for Cashier use.

Draw Use Case Diagram



รูปที่ 4-7 Partial use case diagram for the POST application

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Relate use cases.

Write Some Expanded Essentail Use Cases

ตัวอย่างของยูสเคส Buy Items

Use Case : Buy Items

Section : Main

Use case :	**Buy Items**
Actors:	Costomer (initiator) , Cashier
Purpose:	Capture a sale and its payment
Overview:	A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a Paymeant, which may be authorized. On completion, the Customers leaves with the items.
Type:	primary and essentail
Cross References:	*Functions* : R1.1 , R1.2 , R1.3 , R1.7 , R1.9 , R2.1 , R2.2 , R2.3 , R2.4

Typical Course of Events

Actor Action	System Response
1. This use case begins when a Customer Arrives at the POST checkout with items to purchase.	
2. The Cashier records each item. If there is more than one of an item, the Cashier can enter the quantity as well.	**3.** Determine the item price and adds the item Information to the running sale transaction. The description and price of the current item are presented.
4. On completion of item entry, the Cashier indicates to the POST that item entry is complete.	**5.** Calculates and presents the sale total
6. The Cashier tells the customer the total.	
7. Customer chooses payment type:	

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

a. If cash payment, see section

Pay by Cash.

b. If credit payment, see section

Pay by Credit.

c. If check payment, see section

Pay by Check.

8. Logs the completed sale.

9. Updates inventory levels.

10. Generates a receirt.

11. The Cashier gives the receirt to the Customer.

12. The Customer leaves with the items purchased.

Alternative Courses

- Line 2: Invalid item identifier entered. Indicate error.
- Line 7: Customer could not pay. Cancel sales transaction.

Section : Pay by Cash

- - - - - - - -

Section : Pay by Credit

- - - - - - - -

Section : Pay by Check

- - - - - - - -

If Necessary, Write Some Real Use Cases.

Rank Use Cases.

Sample Models

High-level และ Essential use cases และยูสเคสไดอะแกรมเป็นส่วนหนึ่งของ Analysis Use Case Model ดังรูปที่ 4-8

4.3.3 Refine Conceptual Model

แบบจำลองแนวความคิด (conceptual models)

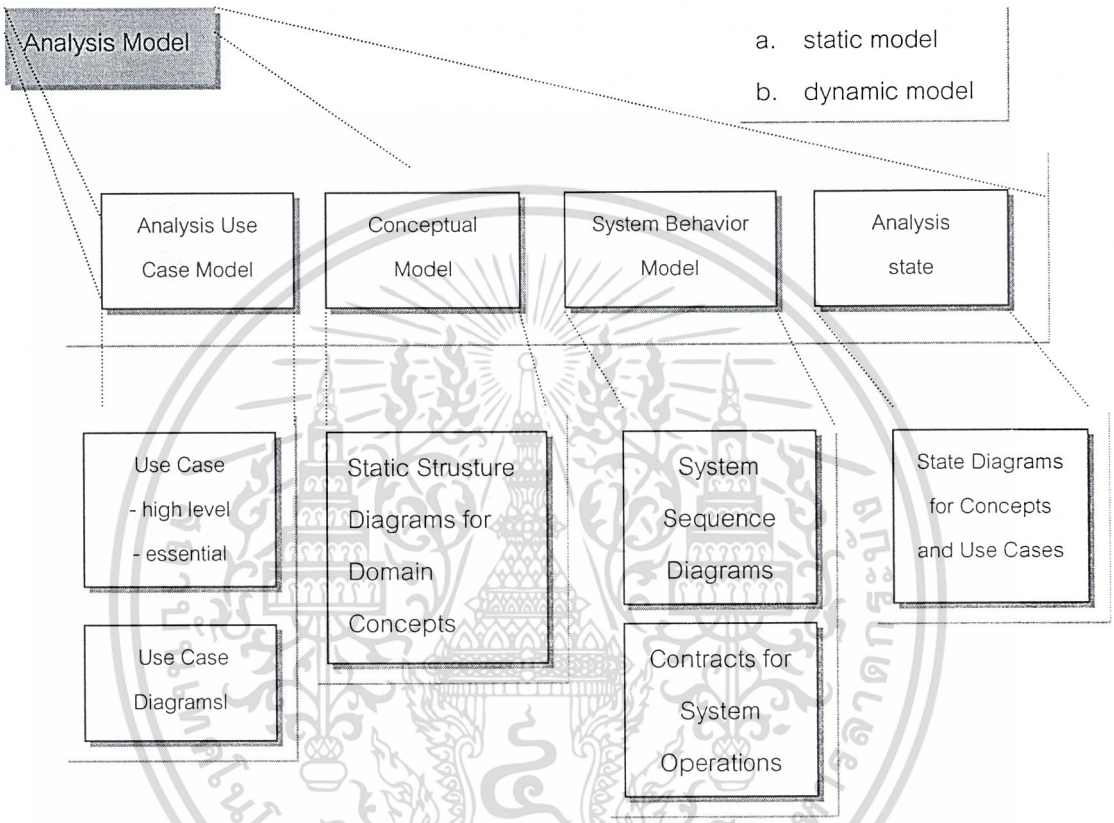
แบบจำลองแนวความคิดคือ การแสดงถึงแนวความคิดของขอบเขตของปัญหา (problem domain) ใน UML แบบจำลองแนวความคิดคือ การแสดงกลุ่มของโครงสร้างของไดอะแกรมแบบคงที่ (static structure diagrams) ซึ่งจะไม่มีการอธิบายถึงการกระทำต่างๆ ซึ่ง แบบจำลองแนวความ จะแสดงถึง

- แนวความคิด (concepts)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- แนวความคิด (concepts)
- การเชื่อมโยง (associations between concepts)
- แอตทริบิวท์ (attributes of concepts)

แบบจำลองแนวความคิดปีนเพียงแนวความคิดที่ไม่ใช่ส่วนประกอบของซอฟต์แวร์ดังนั้นจะไม่มีส่วนของวินโดวส์หรือฐานข้อมูล และจะไม่มีส่วนที่เป็นเมทธอดแต่จะมีเพียง แอตทริบิวท์เท่านั้น



รูปที่ 4-8 The Analysis Model

<u>แนวความคิด</u> (concepts)

แนวความคิด (concept) คือความคิดหรือสิ่งต่างๆ โดยอาจพิจารณาถึง symbol , intension และ extension

- Symbol : words or images representing a concept
- Intension : the definition of a concept.
- Extension : the set of examples to which the concept applies.

<u>Finding Concepts with the Concept Category List</u>

จะเริ่มการสร้างแบบจำลองแนวความคิดโดยการสร้างรายการของแนวความคิดที่น่าจะมีในระบบ เช่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Concept Catelogy	Examples
physical or tangible objects	POST
specifications, designs, or descriptions of things	ProductSpecification
places	Store
transactions	Sale , Payment
transaction line items	SaleLineItem
role of people	Cashier

Conceptual Modeling Guidelines

How to make a conceptual Model

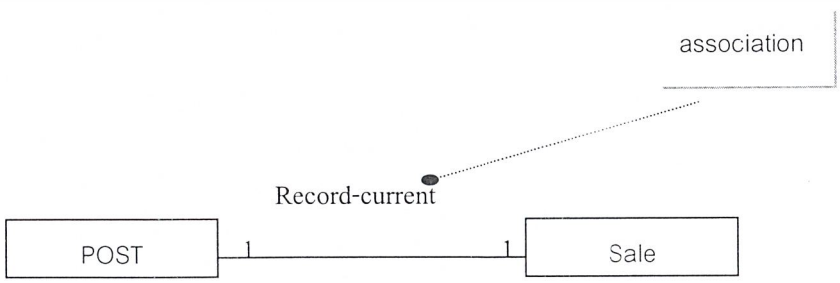
1. เขียนรายการแนวความคิดที่น่าจะมีใน ระบบโดยใช้รายการแนวความคิดและใช้คำนามในการกำหนดความสัมพันธ์ที่ต้องการ
2. ทำการสร้างแบบจำลองความคิด
3. ทำการเชื่อมต่อเพื่อแสดงถึงความสัมพันธ์ของแต่ละส่วน
4. เพิ่มแอตทริบิวต์ที่จำเป็นต้องมี

A Common Mistake in Identifying Concepts

ในการกำหนดแบบจำลองแนวความคิดอาจจะเกิดความผิดพลาดในการระบุแนวความคิดได้ ดังนั้นจึงมีกฎบางอย่างที่จะป้องกันความผิดพลาดเหล่านี้คือ 'ถ้าเราไม่ได้คิดว่า concept X เป็นจำนวนหนึ่งหรือเป็นข้อความหนึ่งใน read world แล้วนั้น X จะเป็น concept ไม่ใช่ attribute'

Conceptual Model Adding Associations

การเชื่อมโยงความสัมพันธ์เป็นความสัมพันธ์ระหว่างแนวความคิดที่แสดงถึงความหมายและความสัมพันธ์ระหว่างกัน ใน UML การเชื่อมโยงความสัมพันธ์จะแสดงถึงโครงสร้างความสัมพันธ์ระหว่างออบเจ็กต์



รูปที่ 4-9 แสดง associations

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

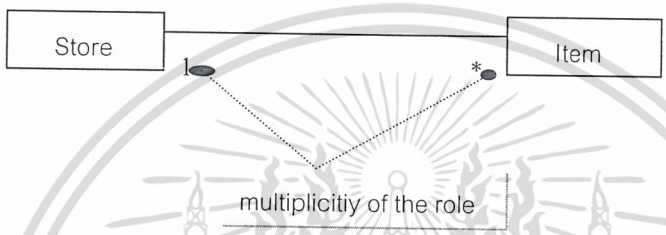
Roles

ในแต่ละการเชื่อมโยงความสัมพันธ์จะเรียกว่าโรล ซึ่งโรลจะประกอบด้วยสิ่งต่อไปนี้

- name
- multiplicity expression
- navigability

Multiplicity

Multiplicity แสดงถึงจำนวนของอินแสตนซ์ A ที่สามารถเชื่อมโยงความสัมพันธ์กับอินแสตนซ์ของ B ได้ ดังรูป



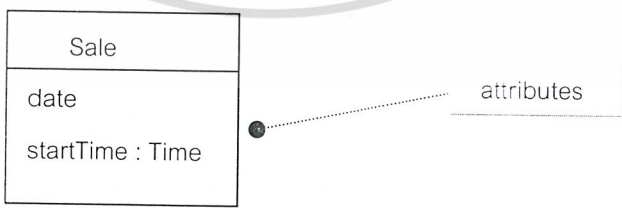
รูป 4-10 แสดง Multiplicity ระหว่าง association

Naming Associations

การตั้งชื่อให้การเชื่อมโยงความสัมพันธ์จะอยู่ในรูปแบบ *TypeName-VerbPhrase-TypeName* ซึ่งชื่อของการเชื่อมโยงความสัมพันธ์ควรจะเริ่มต้นด้วยตัวอักษรตัวใหญ่ ส่วนที่เป็น verb pharse ควรจะคั่นด้วยเครื่องหมาย – ดังรูป

Conceptual Model Adding Attributes

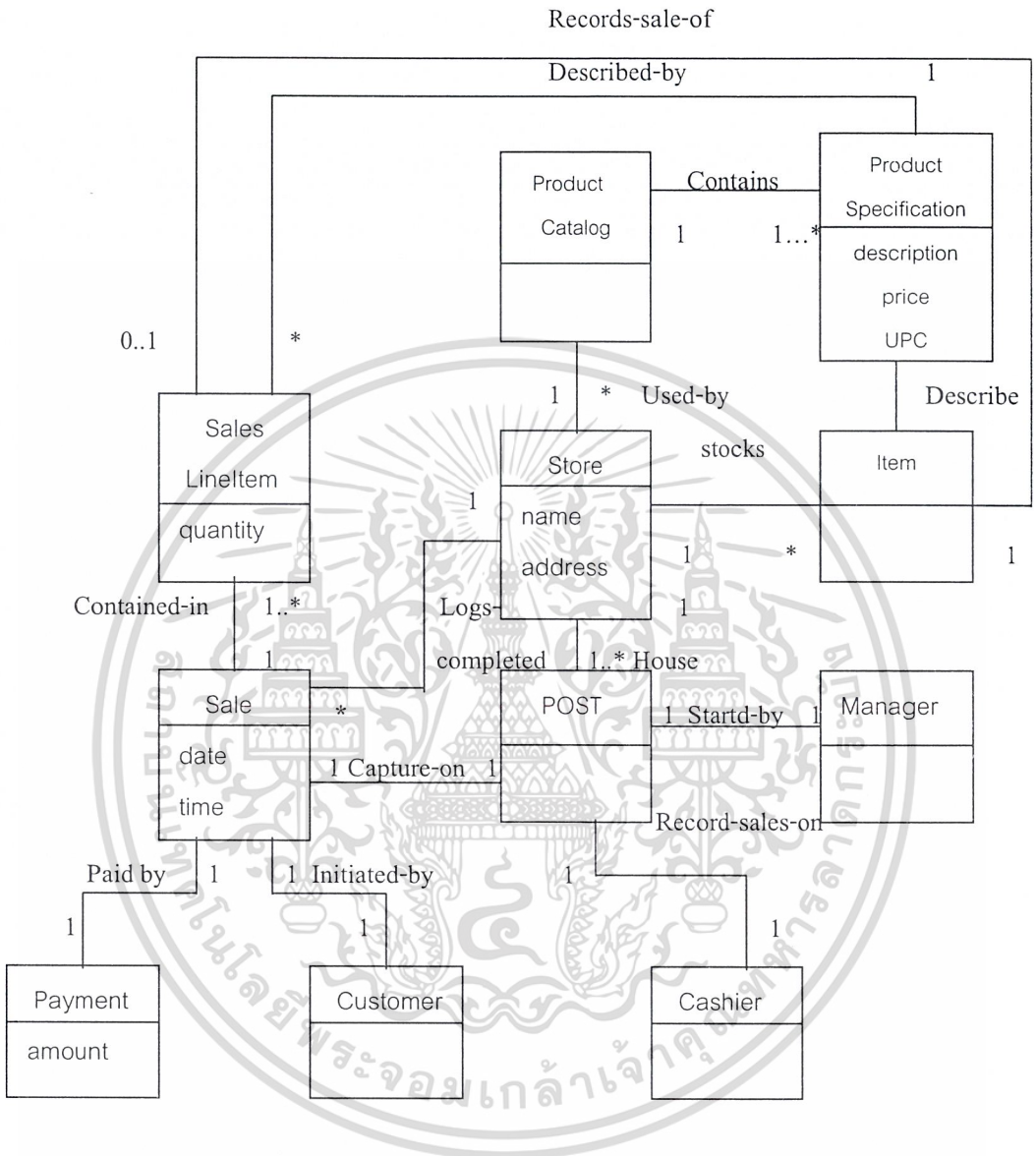
แอตทริบิวท์ คือ ค่าของข้อมูลของออบเจ็กต์ ซึ่งแสดงได้ดังรูป



รูปที่ 4-11 Concept and attributes

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Point-of-Sale Conceptual Model



รูป 4-12 Conceptual model for the point-of-sale domain

4.3.4 Refine Glossary

Glossary

คำแปลศัพท์ (glossary) หรือคำอธิบายรูปแบบจำลอง (model dictionary) มีลักษณะคล้ายกับคำอธิบายข้อมูล (data dictionary) โดยจะอธิบายในส่วนที่ต้องการความชัดเจนเพื่อป้องกันความเข้าใจที่ผิดพลาดที่อาจเกิดขึ้นได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Sample Point-of-Sale System Glossary

Term	Category	Comments
Buy Items	use case	Description of the process of customer buying items in a store.
ProductSpecification.description : Text	attribute	A short description of an item in a slae, and its associated *ProductSpecification.*.
Payment	type	A cash payment.
Item	Type	An item for sale in a *Store.*
ProductSpecification.price: Quantity	attribute	The price of an item in sale, and its associated *ProductSpecification.*
Sale	type	A sales transaction.
SalesLineItem	type	A line item for a particular item bought within a *Sale.*
Store	type	The place where sales of items occur
Sale.total:Quantity	attribute	The grand total of *Sale*
Payment.amount:Quantity	attribute	The amount of cash tendered, or presented, from the customer for payment.
ProductSpecificqtion.upc:UPC	attribute	The universal product code of the Item and its *ProductSpecification.*

รูปที่ 4-13 Conceptual model for the point-of-sale domain

4.3.5. Define System Sequence Diagram

System Sequence Diagram

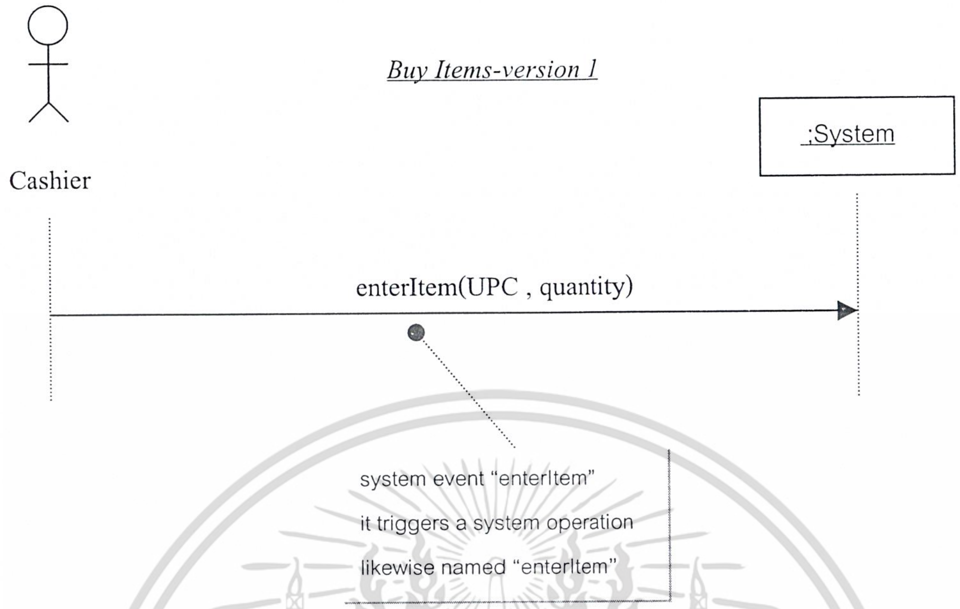
แผนผังแสดงลำดับการทำงานของระบบ (system sequence diagram) เป็นแผนภาพที่แสดงรายละเอียดของยูสเคส , เหตุการณ์ภายนอกที่กระทำกับระบบ , การสั่งงานและเหตุการณ์ต่างๆในระบบ

System Events and System Operation

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เหตุการณ์หรืออีเวนท์ของระบบ (system events) คือ อีเวนท์ที่เป็นอินพุทจากภายนอกที่กระทำกับระบบโดยผู้กระทำ การกระทำหรือโอเปอเรชั่นของระบบ (system operation) คือการทำงานของระบบที่ทำ

การตอบสนองอีเวน์ที่ทำกับระบบ เช่นในรูปที่ 16 เมื่อแคชเชียร์ทำงานกับอีเวนท์ *enterItem*



รูปที่ 4-14 System events initiate system operations.

How to Make a System Sequence Diagram

ในการสร้างแผนผังแสดงลำดับการทำงานของระบบสามารถทำได้ดังต่อไปนี้

1. วาดเส้นที่แสดงถึงระบบ
2. กำหนดผู้กระทำที่ทำงานโดยตรงกับระบบแล้ววาดเส้นไปยังแต่ละผู้กระทำ
3. กำหนดเหตุการณ์ซึ่งผู้กระทำทำงานกับระบบแล้วทำการวาดลงในไดอะแกรม
4. สามารถเพิ่มคำอธิบายเกี่ยวกับผู้กระทำไว้ทางซ้ายของไดอะแกรมได้

Naming System Events and Operations

การตั้งชื่อเหตุการณ์ของระบบควรเป็นลักษณะที่มีความมุ่งหมายชัดเจนกว่าในส่วนของอินพุททั่วๆ ไป เพื่อความชัดเจนควรตั้งชื่อตามการทำงานจริงของระบบ เช่นการกรอกข้อมูลรายการสินค้า ควรใช้ enterItem ดีกว่าใช้ enterKeyPressed ซึ่งจะเป็นการมองที่การทำงานของระบบที่ชัดเจนกว่า

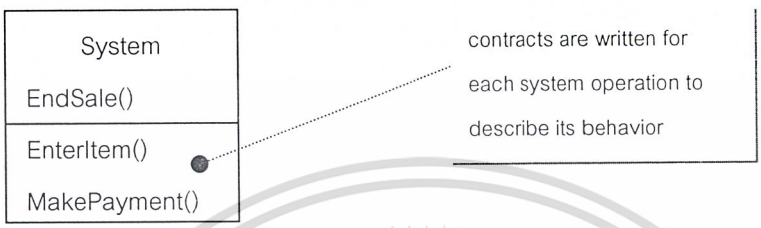
4.3.6 Define Operation Contracts

System Behavior

ก่อนที่จะทำการสร้างซอฟท์แวร์จำเป็นต้องทำการกำหนดคุณลักษณะของระบบในลักษณะของ "black box" ซึ่งการกำหนดคุณลักษณะของระบบจะทำการอธิบายว่าระบบทำอะไรบ้าง (what a system does) โดยไม่มีการอธิบายรายละเอียดว่าระบบทำอย่างไร (how it does it)

Contracts

เอกสารคำย่อ (contract) เป็นเอกสารที่อธิบายการกระทำที่สามารถทำงานได้จริง และอธิบายว่าเหตุการณ์อะไรจะเกิดขึ้น แต่จะไม่อธิบายว่าทำอย่างไร ซึ่งเอกสารคำย่อจะถูกแสดงในรูปของการเปลี่ยนแปลงสถานะแบบ pre condition และ post condition เอกสารคำย่อสามารถเขียนไว้เพื่อเมทธอดของคลาส ได้หรือแสดงการกระทำของระบบก็ได้



รูปที่ 4-15 System operations need contract descriptions

Contract Sections

ในการอธิบายแต่ละส่วนของเอกสารคำย่อแสดงดังต่อไปนี้

Contract

Name :	Name of operation, and parameters
Responsibilities:	An informal description of the responsibilities this operation must fulfill.
Type:	Name of type (concept, software class, interface)
Cross References:	System function refence numbers, use cases , etc.
Notes:	Design notes, algorithms, and so on.
Exceptions:	Exceptional cases.
Output:	Non-UI outputs, such as messages or records that are sent outside of the syatem.
Pre-conditions:	Assumption about the state of the system before execution of the operation.

Post-conditions:

- The state of the system after completion of the operation. Discussed in detail in the following section.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

How to Make A Contract

1. กำหนดการกระทำจากแผนผังแสดงลำดับการทำงานของระบบ

2. สร้างเอกสารคำย่อให้การกระทำของระบบ
3. เริ่มต้นโดยการเขียนส่วนของ *Responsibilities* เพื่ออธิบายวัตถุประสงค์ของการกระทำนั้น
4. หลังจากนั้นสร้างส่วนของ *Post-conditions* จากนั้นอธิบายการเปลี่ยนแปลงของแต่ละสถานะที่เกิดขึ้นกับบออบเจ็กต์ในรูปแบบของแบบจำลอง
5. อธิบาย post-conditions ดังต่อไปนี้
   - การสร้างและลบอินสแตนซ์ (Instance creation and deletion)
   - การแก้ไขแอตทริบิวต์ (Attribute modification)
   - การต่อการเชื่อมโยงและการนำออก (Associations formed and broken)

4.3.7 Define State Diagram

ไดอะแกรมสถานะหรือเสตทไดอะแกรม (state diagram) จะแสดงเหตุการณ์ที่น่าสนใจและสถานะของออบเจ็กต์และคุณลักษณะของออบเจ็กต์ในในการตอบสนองอีเวนท์ และช่วยอธิบายวัฏจักรของออปเจ็กต์ที่เกิดขึ้น

Subject of a State Diagram

ไดอะแกรมสถานะใน UML จะประกอบด้วย

- software classes
- types (concepts)
- use cases

Use Case State Diagrams

ไดอะแกรมสถานะของยูสเคส (use case state diagram) คือไดอะแกรมสถานะที่แสดงให้เห็นถึงอีเวนท์ของเหตุการณ์และลำดับการทำงานภายในยูสเคส

Start Up

ไดอะแกรมสถานะของ *Start Up* ไม่น่าสนใจฉะนั้นจะไม่มีการแสดงให้เห็น

4.4 Design Phase

4.4.1 Define Real Use Case

Real use case จะอธิบายยูสเคสที่มีการออกแบบจริง ในรูปแบบที่มีส่วนรับอินพุทและส่วนแสดงผลเอาท์พุท รวมถึงการใช้งานโดยอาจจะมีรายละเอียดในส่วนของการติดต่อกับผู้ใช้ด้วย

Example – Buy Items-Version 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเอกสารทุกครั้งที่มีการนำไปใช้

Use case : Buy Items – version 1 (Cash only)

Actor :	Customer (initiator) , Cashier.
Purpose :	Capture a sale and its cash payment.
Overview :	A customer arrives at the checkout with items to purchase. The Cashier records the purchase items and collects a cash payment. On completion, the Customer leaves with the items.
Type:	primary and real
Cross References :	*Function* : R1.1, R1.2, R1.3, R1.7, R1.9, R2.1,

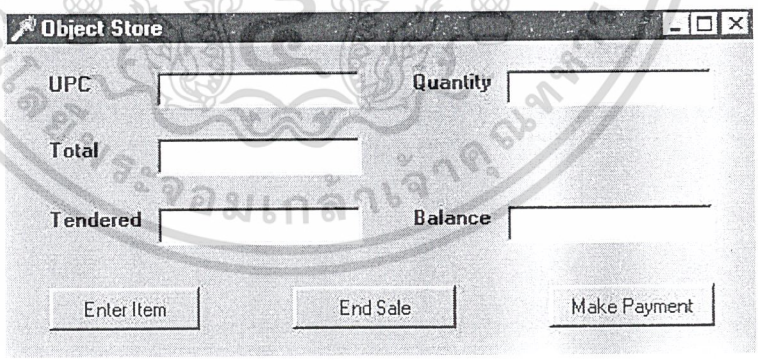
4.4.2 Define Reports, UI and Storyboards

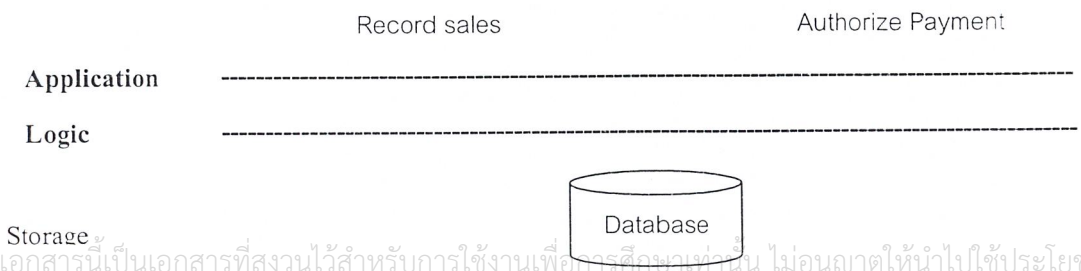
เป็นส่วนเพิ่มความสมบูรณ์ให้กับยูสเคส ซึ่งอาจจะแสดงผลเป็นรายงาน, การติดต่อกับผู้ใช้หรือ การแสดงเนื้อหา

4.4.3 Refine System Architecture

สถาปัตยกรรมของระบบในที่นี้จะกล่าวถึง 3 tier architecture ซึ่งประกอบด้วยสามส่วนด้วยกันคือ user interface 1 ส่วน และส่วนเก็บข้อมูลอีก 2 ส่วน โดยแบ่งแยกได้ดังต่อไปนี้

1. Presentation :– client
2. Application Logic :- application server
3. Storage :- database server





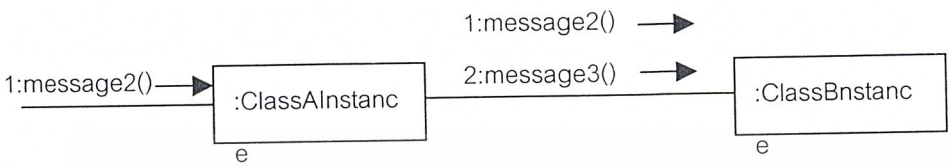
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4-16 Classic view of a three-tier architecture

4.4.4 Define Interaction Diagrams

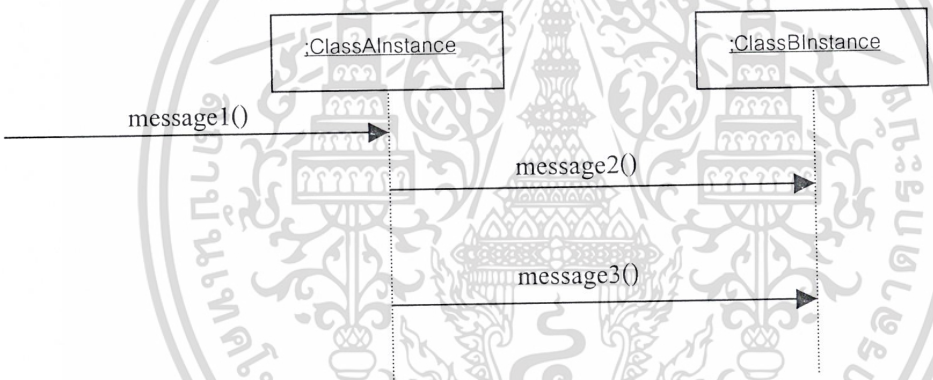
อินเตอร์แอคชั่นไดอะแกรมจะแสดงการตอบสนองเมสเสจระหว่างอินแสตนซ์และคลาสในแบบจำลองคลาส ซึ่งใน UML จะแบ่งอินเตอร์แอคชั่นไดอะแกรมออกเป็นสองประเภท คือ

1. collaboration diagrams จะแสดงออบเจ็กต์อินเตอร์แอคชั่นในรูปแบบกราฟหรือโครงข่าย



รูปที่ 4-17 Collaboration diagram

2. sequence diagrams จะแสดงอินเตอร์แอคชั่นในรูปแบบดังรูป



รูปที่ 4-18 Sequence Diagram

How to Make Collaboration Diagrams

1. สร้างไดอะแกรมต่าง ๆ แยกออกจากกันสำหรับแต่ละระบบ
2. ถ้าไดอะแกรมค่อนข้างซับซ้อน ให้แยกออกมาเป็นไดอะแกรมย่อย ๆ หลายไดอะแกรม
3. ใช้หน้าที่ที่ได้จาก contract และ post-condition และรายละเอียดของ use case มาออกแบบระบบและเขียนเป็น method ต่าง ๆ ให้หมดทั้งหมดระบบงาน

4.4.5 Define Design Class Diagrams

Design class diagram

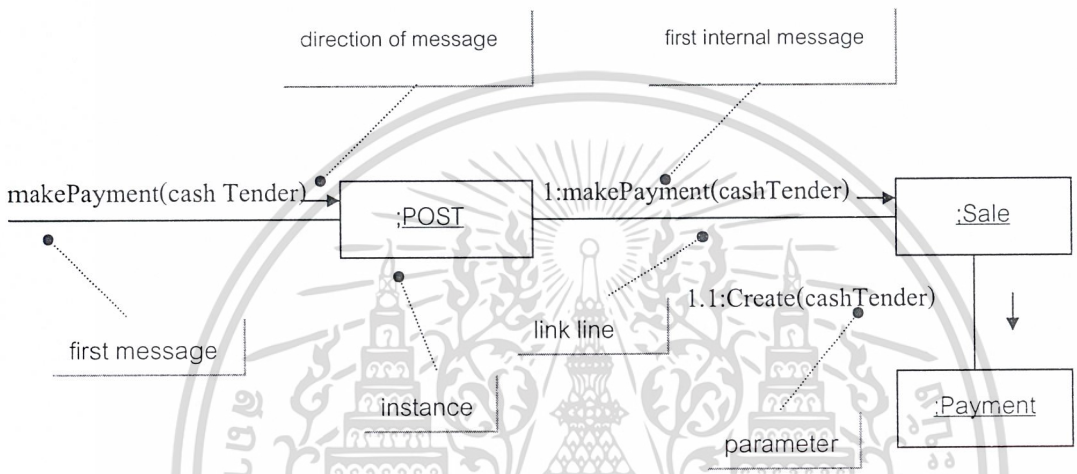
การออกแบบคลาสไดอะแกรมเป็นการแสดงรายละเอียดซอฟท์แวร์คลาสและการติดต่อกับผู้ใช้ในส่วนของแอปพลิเคชั่น ซึ่งคลาสไดอะแกรมประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- classes, associations and attributes

- interface with operations and constants
- methods
- attribute type information
- navigability
- dependencies

Example Collaboration Diagram : makePayment



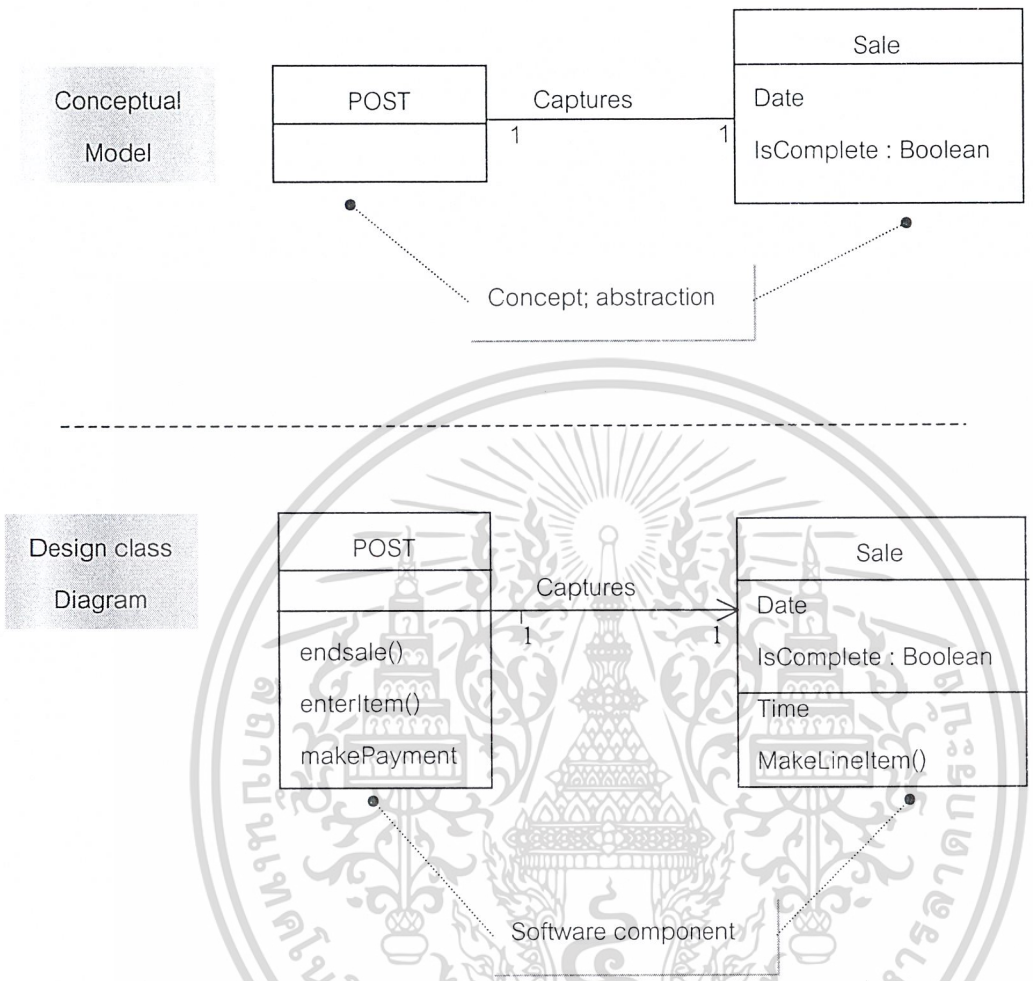
รูปที่ 4-19 Collaboration diagram

How to Make a Design Class Diagram

1. ระบุคลาสทั้งหมดที่จะต้องใช้ในการสร้างแอปพลิเคชั่น ซึ่งจะได้มาจากส่วนของอินเตอร์แอคชั่นไดอะแกรม
2. วาดคลาสไดอะแกรม
3. สร้างแอททริบิวต์ขึ้นมาใหม่จากการเชื่อมโยงแนวคิด (associated concepts) ในแบบจำลองแนวความคิด
4. เพิ่มชื่อของเมธอดโดยวิเคราะห์จากอินเตอร์แอคชั่นไดอะแกรม
5. เพิ่มชนิดของข้อมูลให้กับแอตทริบิวต์และเมทธอด
6. เพิ่มความสัมพันธ์ให้กับแอตทริบิวต์ที่ต้องการให้ชัดเจน
7. สร้างลูกศรแสดงความสัมพันธ์ในการบ่งบอกทิศทางของแอตทริบิวต์ให้ชัดเจน
8. สร้างเส้นที่แสดงความสัมพันธ์แบบขึ้นต่อกันเพิ่มบ่งบอกส่วนที่ไม่ใช่แอตทริบิวต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Conceptual model versus Design class diagrams



รูปที่ 4-20 Conceptual mode versus design class diagram

Creating the Point-of-Sale Design Class Diagrams

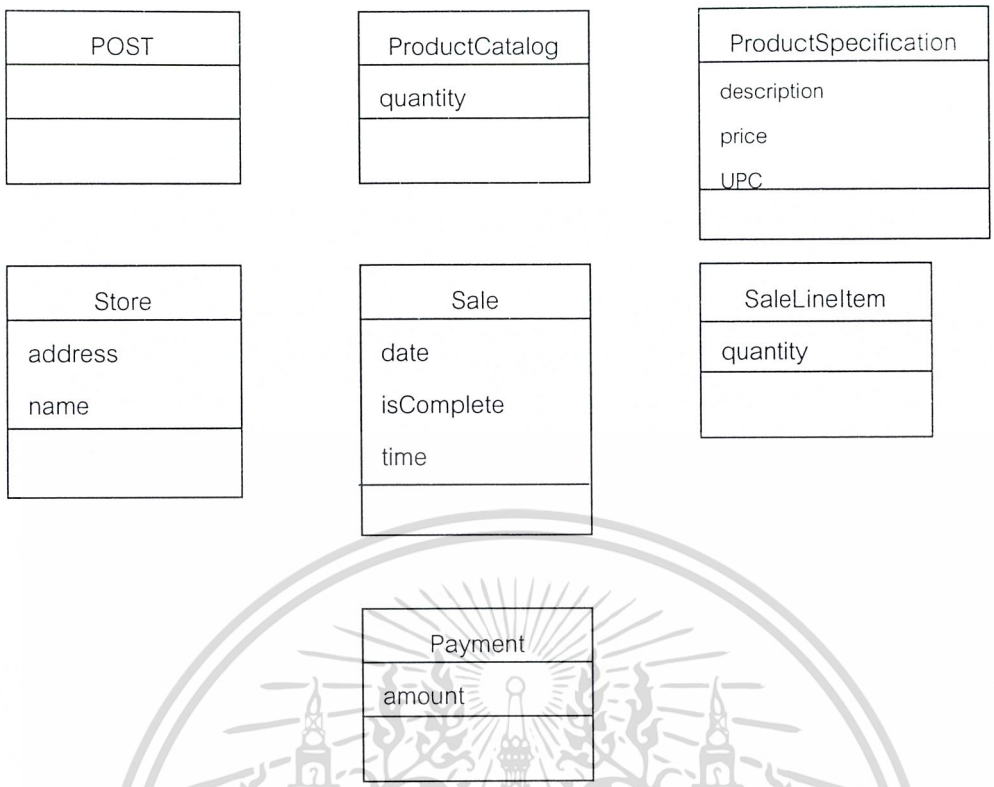
Identify Software Classes and Illustrate Them

ทำการกำหนดคลาสได้ดังต่อไปนี้

POST	Sale
ProductCatalog	ProductSpecification
Store	SalesLineItem
Payment	

ทำการวาดคลาสไดอะแกรม โดยใส่แอตทริบิวท์ให้แต่ละคลาส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4-21 Software Classes in the application

Add Method Names

POST : endSale()
enterItem()
makePayment()

Store : addSale()

ProductCatalog : specification()

Sale : becomeComplete()
makeLineItem()
makePayment()
total()

SaleLineItem : subtotal()

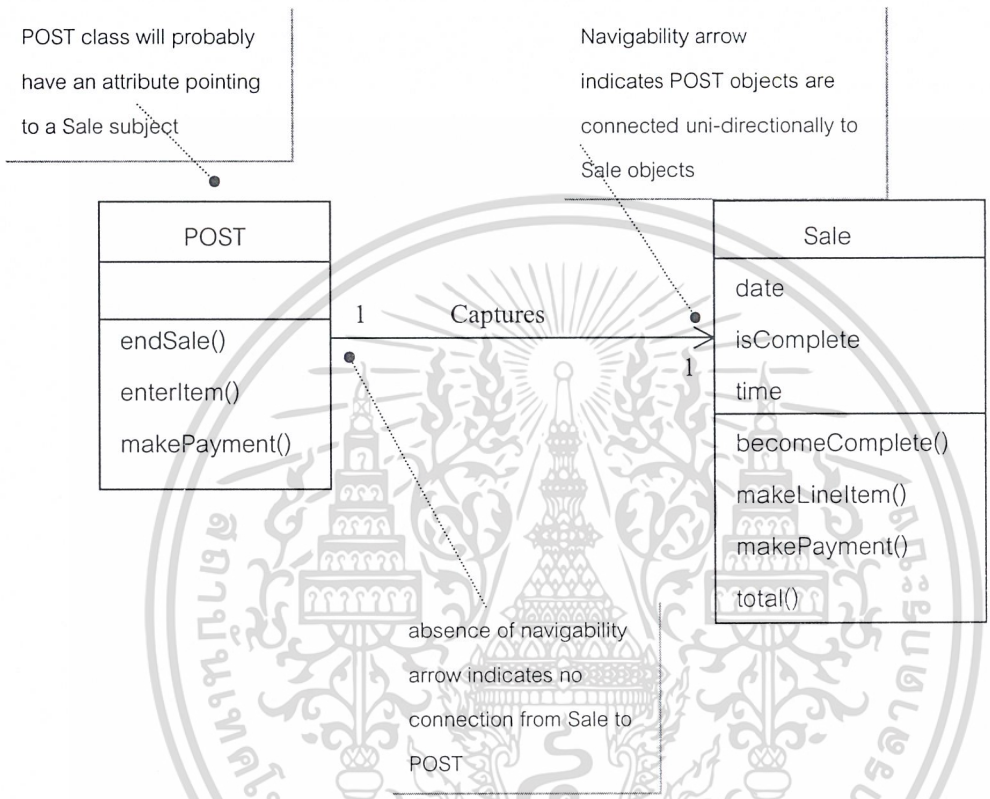
Adding Associations and Navigability

Navigability คือ คุณสมบัติของบทบาท (role) ที่แสดงการเชื่อมโยงโดยตรงเพียงเส้นเดียว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

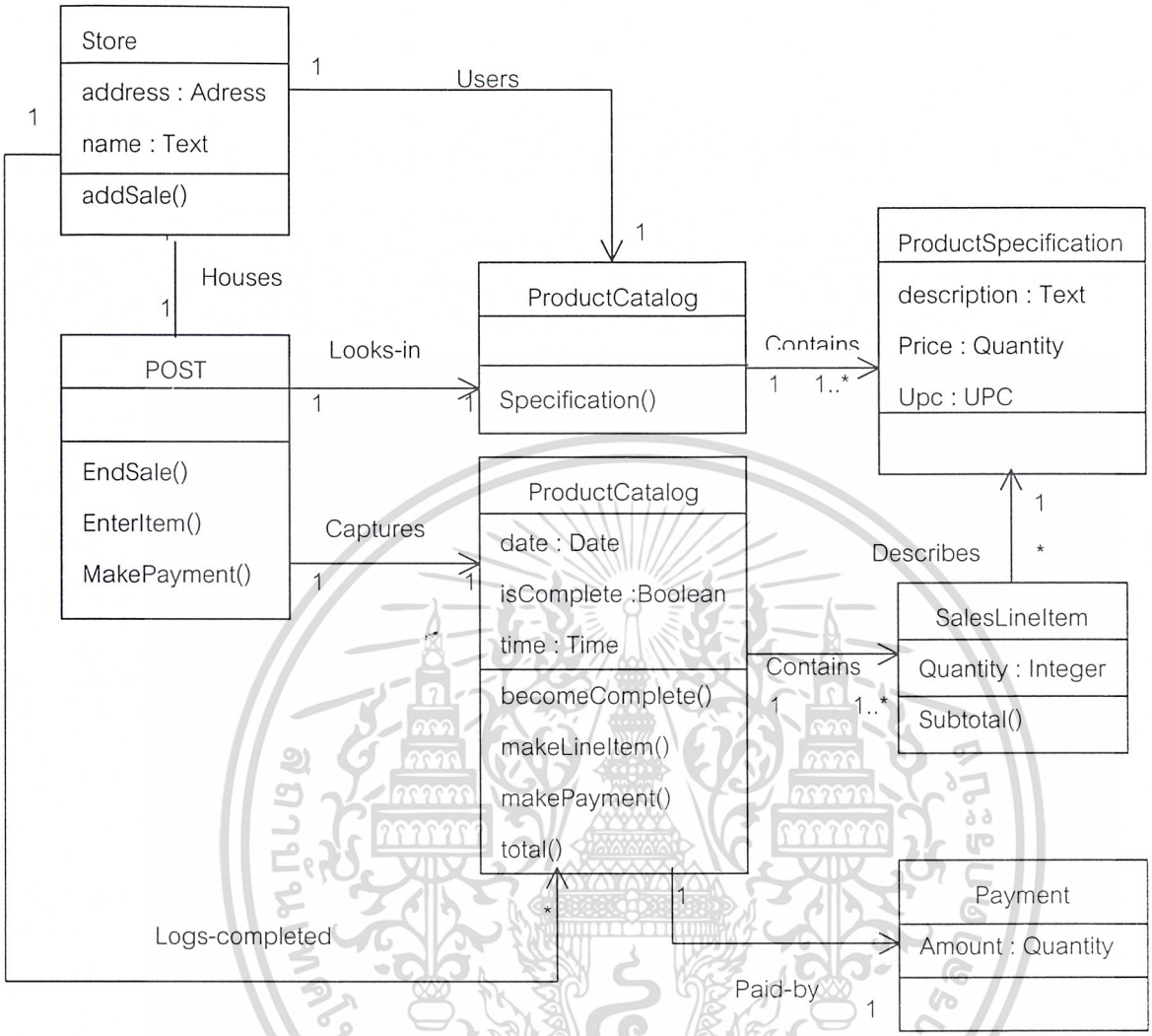
4.4.6 Define Database Schema

Database schema เป็นส่วนของการออกแบบฐานข้อมูลที่สัมพันธ์กับระบบงาน เพื่อจัดเก็บข้อมูลให้อยู่ในรูปแบบที่ต้องการ



รูปที่ 4-22 Showing navigability, or attribute visibility.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4-23 Associations with navigability adornments

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
ระบบไคลเอนท์เซิร์ฟเวอร์แบบมัลติเทียร์
(Multitier Client/Server)

5.1 Creating multi-tiered applications

Muti-tiered client/server application จะถูกแบ่งออกเป็นหน่วยย่อยในทางลอจิก ซึ่งสามารถทำการรันในเครื่องที่ต่างกันได้ Multi-tiered application จะแชร์ข้อมูลและติดต่อกันผ่านทาง LAN หรือทางอินเตอร์เน็ท ซึ่งอาจเรียกว่า "three-tiered model" จะแบ่งออกเป็นสามส่วนคือ

- Client application : จะประกอบด้วยส่วนที่ใช้ติดต่อกับผู้ใช้บนเครื่องของผู้ใช้เอง
- Application server: จะอยู่ตรงกลางของระบบเครือข่ายซึ่งสามารถติดต่อกับ client ได้ทุกเครื่องและจะมีส่วนที่เป็น common data service
- Remote database server: จะเป็นส่วนของระบบฐานข้อมูล

ในระบบ client/server แบบสามระดับ application server จะเป็นส่วนจัดการการไหลของข้อมูลระหว่าง client และ remote database server ดังนั้นอาจเรียก server ได้ว่าเป็น data broker

Delphi สามารถสนับสนุนระบบ client/server แบบสามระดับได้โดยพื้นฐานของ MIDAS (Multi-tier Distributed Application Service Suit)

5.2 คุณประโยชน์ของ Muti-tiered database model

ไคลน์แอปพลิเคชั่น (client application) สามารถกำหนดข้อมูลที่จะทำการแสดงและการติดต่อกับผู้ใช้ได้โดยที่ตัวไคลน์แอปพลิเคชั่นไม่จำเป็นต้องทราบว่าข้อมูลถูกเก็บรักษาไว้ที่ใด ในส่วนของแอปพลิเคชั่นเซิร์ฟเวอร์ (application server) ซึ่งเป็นส่วนกลางของการติดต่อ (middle ware) จะทำการเชื่อมโยงและดำเนินการตามการร้องขอและการเปลี่ยนแปลงแก้ไขข้อมูลจากไคลน์ ซึ่งคุณประโยชน์ของระบบไคลน์เซิร์ฟเวอร์แบบหลายระดับ คือ

- Encapsulation of business logic in a shared middle tier : มีการรวมการทำงานทางลอจิกเข้าไว้ด้วยกันในส่วนกลางหรือแอปพลิเคชั่นเซิร์ฟเวอร์ที่สามารถทำการแชร์ข้อมูลได้ โดยที่ ไคลน์แอปพลิเคชั่นทั้งหมดจะทำการติดต่อที่แอปพลิเคชั่นเซิร์ฟเวอร์ตัวเดียวกัน ซึ่งเป็นการง่ายต่อการบำรุงรักษาเมื่อมีการเปลี่ยนแปลงกฎของการทำงานในแต่ละไคลน์แอปพลิเคชั่นแต่ละตัว
- Thin client applications : สามารถทำการสร้างไคลน์แอปพลิเคชั่นที่มีขนาดเล็กได้โดยที่การทำงานที่มีกับแอปพลิเคชั่นเซิร์ฟเวอร์มีจำนวนมากได้ และการเพิ่มจำนวนไคลน์แอปพลิเคชั่นยังทำได้โดยง่าย
- Distributed data processing : การกระจายงานของแอปพลิเคชั่นบนแต่ละเครื่องสามารถเพิ่มประสิทธิภาพของระบบให้ดีขึ้นได้ เนื่องจากมีการทำ load balancing และยอมให้ระบบอื่นรับทำงานหนึ่งต่อ ไปเมื่อเซิร์ฟเวอร์ไม่สามารถทำงานได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Increase opportunity for security : Middle tiers สามารถจำกัดสิทธ์ในการเข้าติดต่อเครื่องที่อาจเกิดความเสียหายได้ง่ายหรืออาจยอมให้การติดต่อเป็นไปได้โดยง่าย ซึ่ง CORBA และ MTS สามารถรองรับรูปแบบการป้องกันเหล่านี้ได้

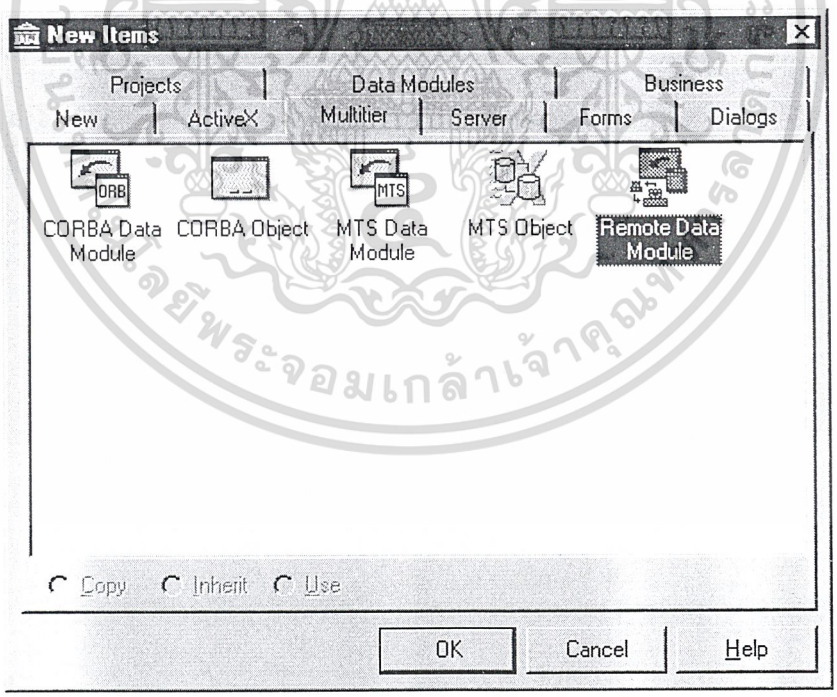
5.3 Building a multi-tiered application

ขั้นตอนโดยทั่วไปที่ใช้สร้าง multi-tiered database application มีอยู่สามขั้นตอน คือ

1. Creating application server.
2. Register or install the application server.
3. Create a client application

5.3.1 Creating application server

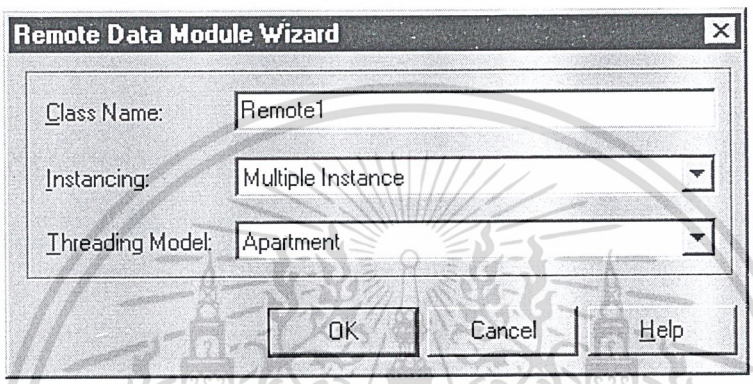
1. เปิด Project ใหม่แล้วทำการบันทึกไว้
2. ทำการเพิ่ม Remote Data Module ไว้ที่ Project จาก Main menu เลือก File|New แล้วเลือก Multitier page ใน new items dialog หลังจากทำการเลือกแล้วจะเห็น Remote Module wizard ดังรูปที่ 1



รูปที่ 5-1 Remote Data Module Wizard

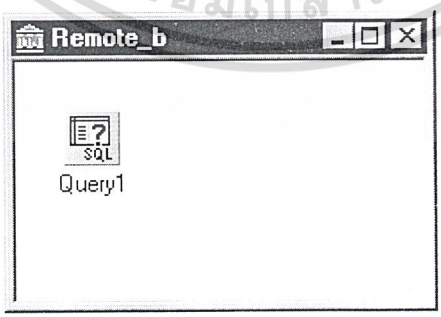
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. ผู้สร้างจะต้องทำการป้อนชื่อของ remote data module ซึ่งจะเป็นชื่อของผู้สืบทอดของ *TremoteData-Module* ซึ่งเป็นแอปพลิเคชั่นที่ทำการสร้างขึ้น ดังรูปจะป้อนชื่อ *Remote1* แล้วทำการกำหนดค่าอินสแตนซ์เป็นมัลติเพิ่ล



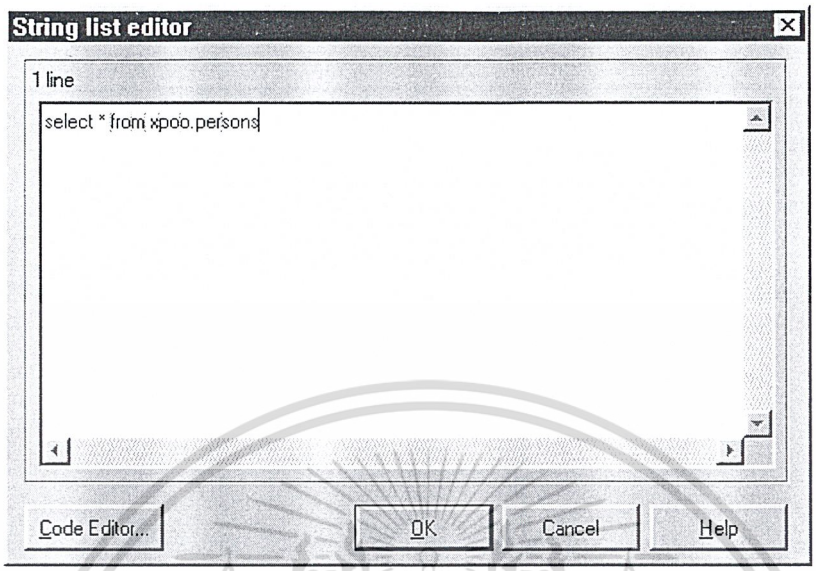
รูปที่ 5-2 Supply a class name for your remote data module

4. ใส่คอมโพเนนท์ Query ลงใน Remote data module ที่สร้างไว้ แล้วทำการเซ็ทค่า properties ต่าง ๆ ดังต่อไปนี้ โดยต้องทำการเขียนคำสั่ง SQL ลงในคอมโพเนนท์นี้ด้วย

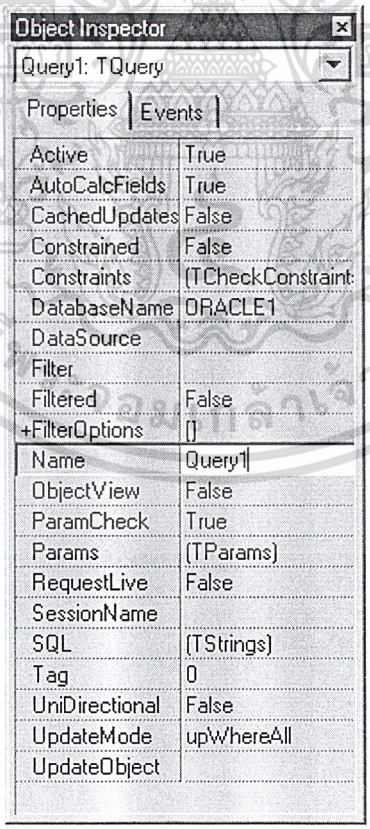


รูปที่ 5-3 Components of Remote data module

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5-4 หน้าจอแสดงการเขียนคำสั่งภาษา SQL



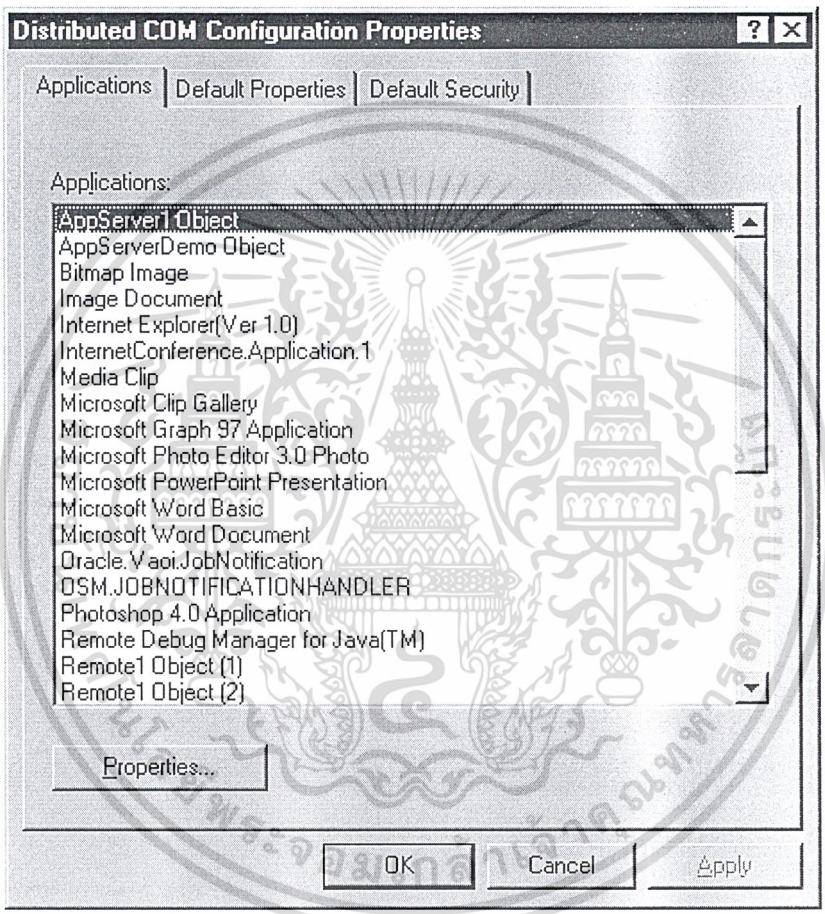
รูปที่ 5-5 การเซ็ทค่าต่างๆ ของคอมโปเนนท์ Query

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. ทำการบันทึกและคอมไพล์ Project จากนั้นทำการรีจิสเตอร์หรือทำการติดตั้ง Application server

5.3.2 Register or install the application server

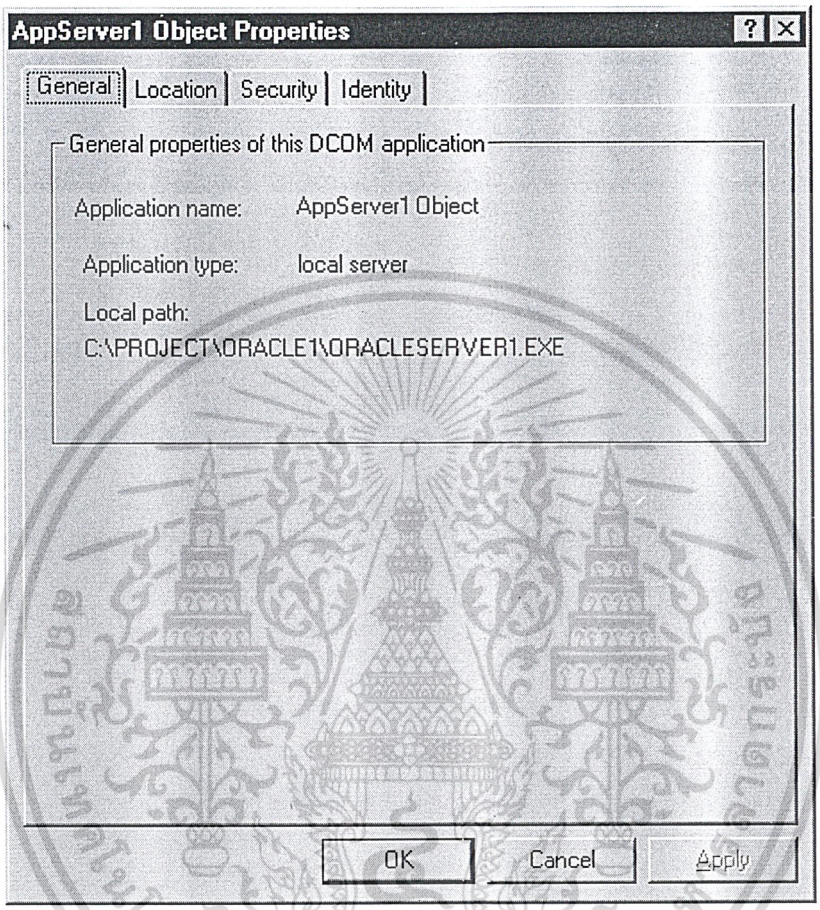
1. รันแอปพลิเคชั่นเพื่อทำการรีจิสเตอร์ จากนั้นเรียกคำสั่ง dcomconfig จะได้หน้าจอดังรูปที่ 5-6



รูปที่ 5-6 หน้าจอแสดงการรันคำสั่ง dcomcnfg

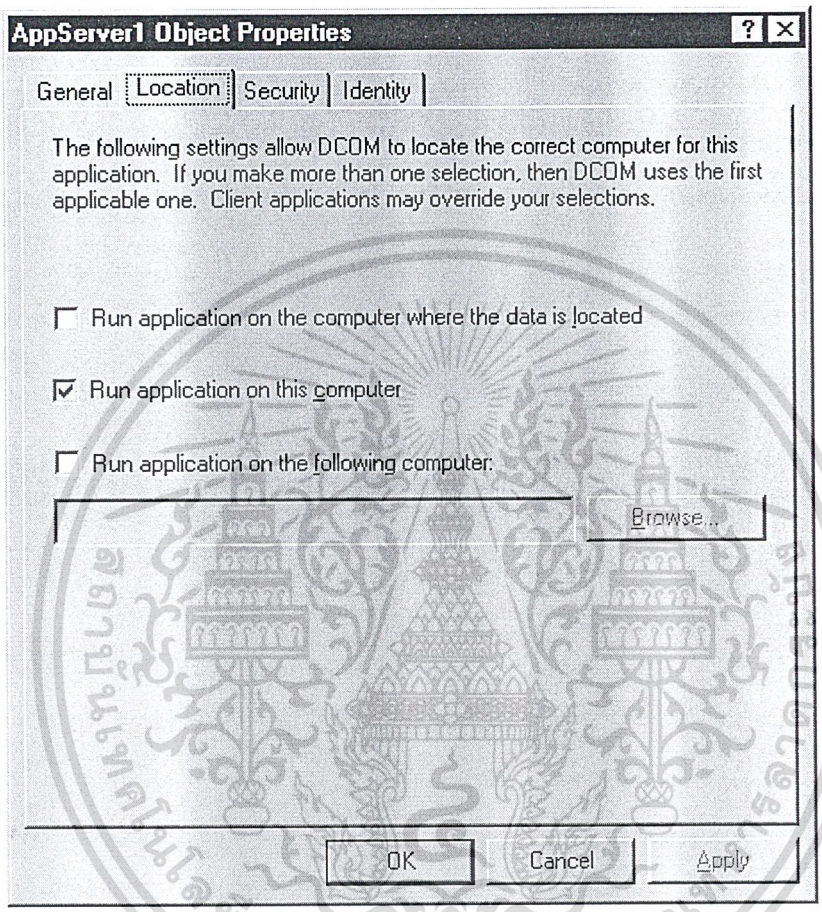
2. ทำการเซ็ทค่า property ต่างๆ ต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5-7 การเซ็ทค่า General

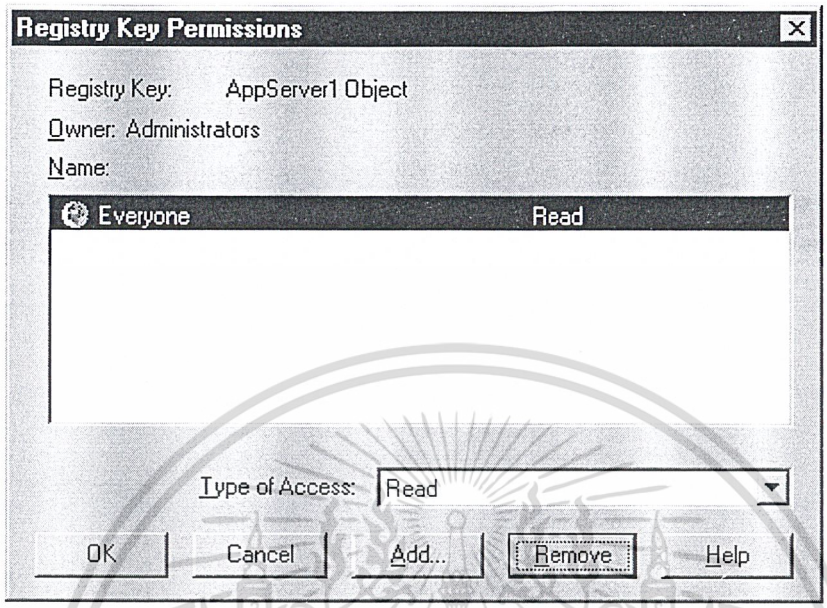
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



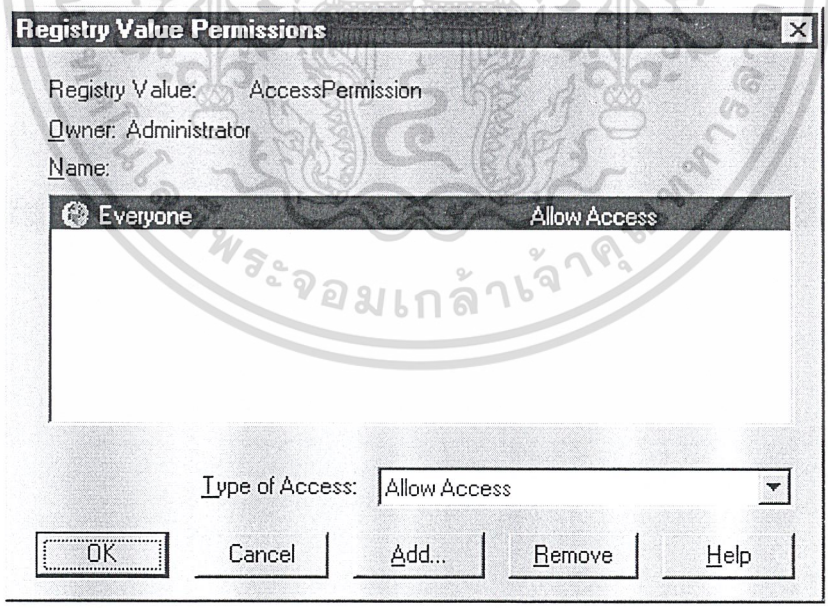
รูปที่ 5-8 การเซ็ท Location

3. ทำการเซ็ท Security ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

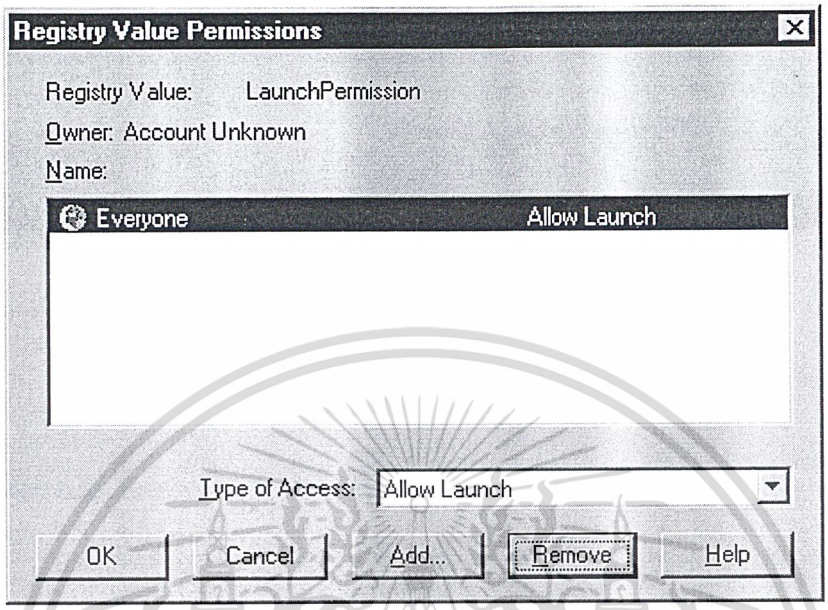


รูปที่ 5-9 การเซ็ท Application1 Object



รูปที่ 5-10 การเซ็ท Access Permission

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5-11 การเซ็ท Launch Permission

เมื่อคอนฟิก DCOM เรียบร้อยแล้ว เราจะสามารถรันแอพพลิเคชัน .exe จากเครื่อง Client ที่ใช้งานระบบปฏิบัติการ Windows98 หรือ WindowsNT ผ่านมายัง Application Server ที่รันบน WindowsNT Server ได้

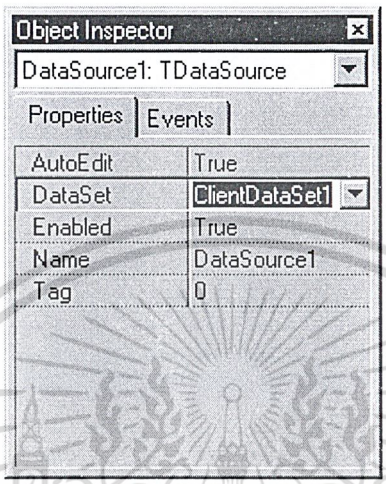
Creating the Client Application

1. เพิ่ม new data module ที่ project
   - DataSource component

2. ใส่ connectioncomponent บน data module ดังต่อไปนี้
   - RemoteServer

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. เซ็ทค่า properties ต่างๆ ที่ connection component เพื่อเจาะจง application server ที่จะทำการติดต่อ
4. เซ็ทค่า properties ของ connection component ที่จำเป็นต่อแอปพลิเคชั่น

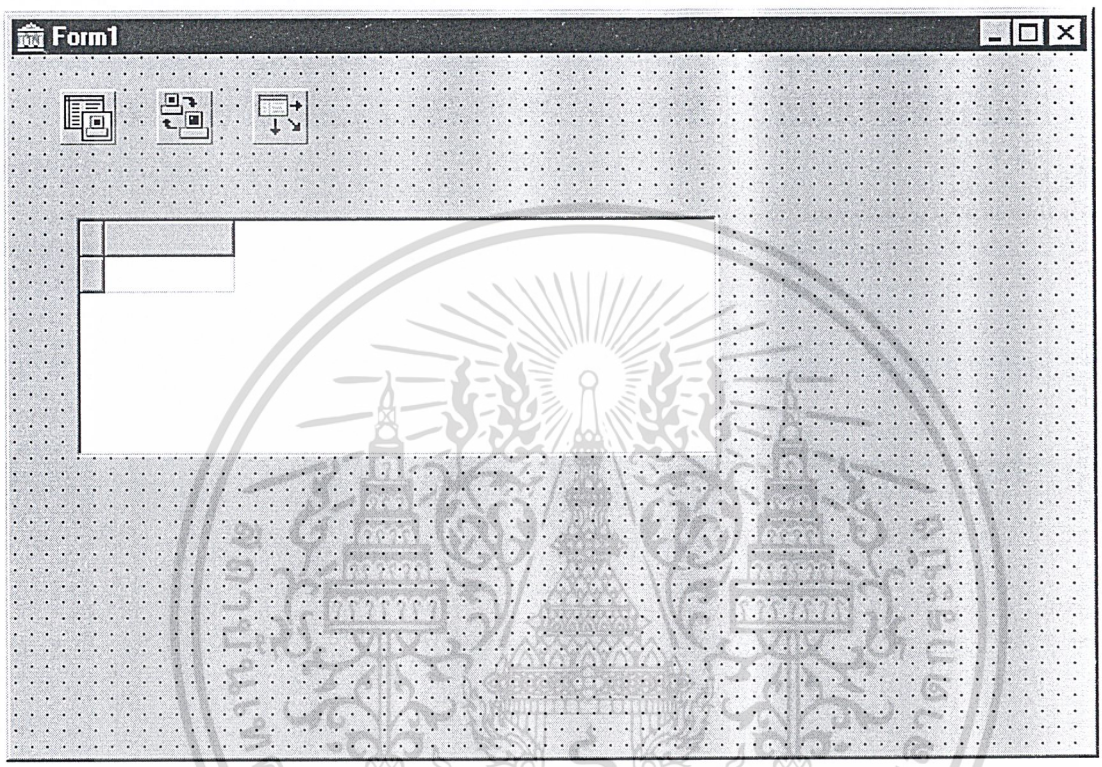


รูปที่ 5-12 การเซ็ทค่าของคอมโปเนนท์ DataSource

รูปที่ 5-13 การเซ็ทค่าของคอมโปเนนท์ RemoteServer

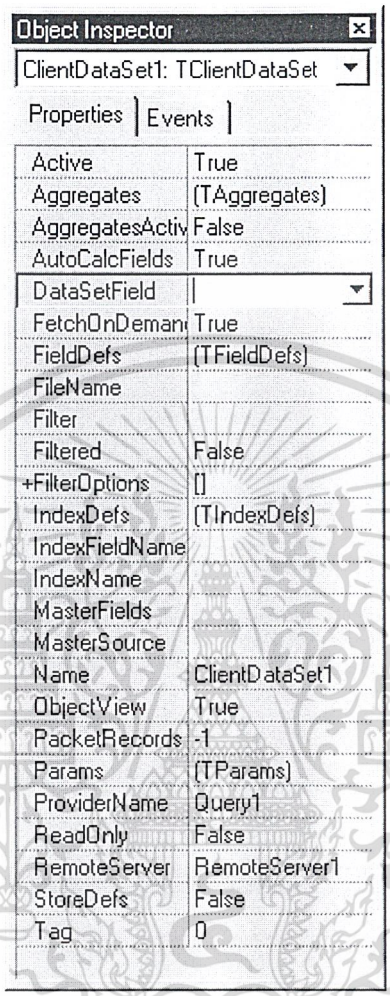
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. ใส่ *TclientDataSet components* ที่ต้องการใน data module
6. เซ็ท *ProviderName* propoty ของแต่ละ *TclientDataSet* component และเซ็ท *ProviderNAme* property



รูปที่ 5-14 DBGrid , RemoteServer , DataSource and ClientDataSet

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



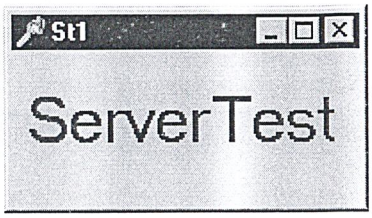
รูปที่ 5-15 การเซ็ทค่าของคอมโปเนนท์ ClientDataSet

การทดลองรันแอปพลิเคชั่น

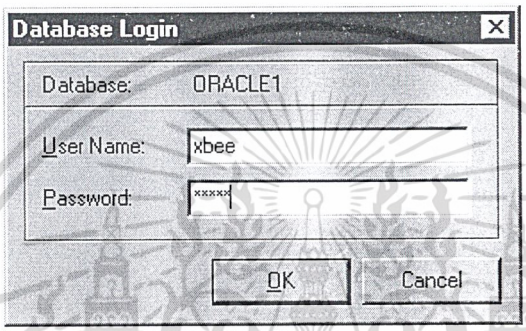
เมื่อทำการรันแอปพลิเคชั่นที่เครื่องที่เป็น Client และ Server ได้ผลดังนี้

1. เมื่อทำการรันแอปพลิเคชั่นในด้านของ Client ทางด้าน Application Server จะแสดงหน้าจอแสดงการ Database Login และแสดงฟอร์มของ Server ตามที่ได้สร้างไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

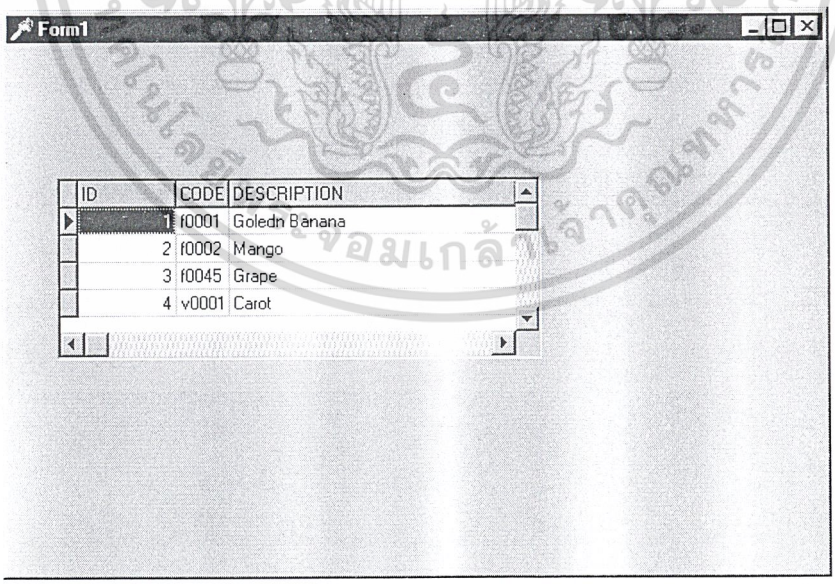


รูปที่ 5-16 หน้าจอแสดงทางด้าน Application Server



รูปที่ 5-17 หน้าจอแสดงทางด้าน Application Server

2. เมื่อใส่ User name และ password ถูกต้องแล้วทางด้าน Client สามารถแสดงผลได้ดังนี้



รูปที่ 5-18 แสดงรายละเอียดตารางตามที่ได้เขียนภาษา SQL ไว้ใน Query

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6
มาตรฐาน SQL 3 และระบบการจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์
(SQL 3 Standard and Object Relational Database Management System)

ในปัจจุบันนอกจากระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ (RDBMS : Relational Database Management System) แล้วยังมีระบบการจัดการฐานข้อมูลเชิงวัตถุ (OODBMS : Object Oriented Database Management System) ซึ่งเป็นระบบฐานข้อมูลที่สนับสนุนความสามารถในด้านความเป็นออบเจ็กต์โอเรียนเต็ด (Object-Oriented) โดยใช้ออบเจ็กต์เป็นพื้นฐานที่มีความสามารถในการสนับสนุนชนิดข้อมูลที่มีความซับซ้อนมาก แต่ก็ยังมีข้อเสียบางประการ เช่น ในเรื่องของ concurency และในเรื่องของภาษาที่ใช้กับระบบฐานข้อมูลแบบนี้ที่ยังไม่มีมาตรฐานรองรับที่แน่นอนและอีกทั้งยังเป็นการยากที่จะเปลี่ยนระบบฐานข้อมูลแบบเดิม (RDBMS) ที่ใช้ในปัจจุบันมาเป็นระบบฐานข้อมูลแบบเชิงวัตถุ (OODBMS) จึงเกิดความคิดที่จะพัฒนาระบบฐานข้อมูลขึ้นมาใหม่โดยพยายามที่จะนำข้อดีของทั้งสองระบบมาพัฒนาร่วมกันเป็นระบบฐานข้อมูลแบบใหม่คือ Obejct Relational Database Management System : ORDBMS

ซึ่งในความเป็นจริงนั้นการที่นำระบบฐานข้อมูลเชิงวัตถุมาใช้ร่วมกับระบบฐานข้อมูลเชิงสัมพันธ์นั้น จะทำให้คุณสมบัติในทางเชิงวัตถุลดน้อยลงกว่าระบบฐานข้อมูลแบบเชิงวัตถุอย่างเดียว และเนื่องจากมีการพัฒนาระบบฐานข้อมูลแบบ ORDBMS ก็จำเป็นที่จะต้องมีภาษาที่มาสนับสนุนคุณสมบัติต่างๆ ที่เพิ่มขึ้นมา ดังนั้นทาง ISO และ ANSI จึงได้มีการกำหนดมาตรฐานของภาษา SQL ออกมาเป็น SQL3 ที่มีคุณสมบัติต่างๆ ที่สนับสนุนในเรื่องของออบเจ็กต์โอเรียนเต็ด ซึ่งมีลักษณะคล้ายกับภาษา OQL (Object Query Language) ที่ใช้สำหรับระบบฐานข้อมูลที่เป็นแบบเชิงวัตถุ ซึ่งเป็นภาษาที่ทาง ODMG ได้กำหนดมาตรฐานออกมาเพื่อสนับสนุนการทำ Object query โดยมีความสามารถในการใช้งานร่วมกับภาษา Host (Host language เช่น C++, Smalltalk หรือ Java) ในการส่งผ่านค่า (transfer) ของ Object กล่าวคือภาษา OQL สามารถที่จะทำการ query แล้วส่งค่ากลับมาเป็นออบเจ็กต์ได้ ซึ่ง SQL3 ไม่สามารถทำได้ ซึ่งทำให้ภาษา OQL มีข้อดีกว่าการเขียน SQL embedded ในภาษา Host แต่อย่างไรก็ตามทั้งภาษา SQL และ OQL ก็ยึดหลักความต้องการให้ภาษามีลักษณะที่ง่ายต้องการใช้ในการทำ query กับ OODBMS และ ORDBMS

6.1 Tuple Objects in SQL3

ออบเจ็กต์ใน SQL3 นั้นแบ่งได้เป็นสองคุณลักษณะ คือ

1. Row Object
2. Abstract Data Types

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.2 Row types

Row Type ใน SQL3 สามารถกำหนดชนิดของ tuple (แถว) เองได้ เหมือนกับคลาสของออบเจ็กต์ Row Type ประกอบด้วย

- Keywords คือ CREATE ROW TYPE
- ชื่อของชนิด Row Type
- กลุ่มของ Attributes

รูปแบบของกำหนด Row Type มีรูปแบบดังนี้

CREATE ROW TYPE *name* (< *component declarations*>)

โดยที่ <name> เป็นชื่อที่กำหนดให้กับ row type ที่ต้องการสร้างและ <component declarations> เป็นการกำหนดค่าให้แต่ละ คอมโพเนนท์ภายใน row type ตัวอย่างเช่นสมมุติต้องการสร้าง row type ที่ใช้สำหรับเก็บรายละเอียดของดารา โดยกำหนดให้มีข้อมูลต่างๆ ดังนี้ ชื่อ, ถนนและเมือง จะต้องทำการประกาศการสร้าง row type ดังนี้

ตัวอย่างการสร้าง Row Type

```
CREATE ROW TYPE      AddressType(
                        Street CHAR(50),
                        City    CHAR(20)
);
CREATE ROW TYPE StarType(
                        Name CHAR(30),
                        Address AddressType
);
```

จากตัวอย่างจะเห็นว่า AddressType เป็นข้อมูลชนิด Row Type ที่มี Street และ city อยู่และมีชนิดของข้อมูลเป็น CHAR ส่วน StarType มีชนิดของข้อมูลเป็น Row Type ที่มี Name และ Address อยู่ ซึ่ง Address เป็นข้อมูลชนิด Row Type จากตัวอย่างจะเห็นได้ว่าข้อมูลชนิด Row Type สามารถเก็บข้อมูลชนิด Row Type ได้

6.2.1 การประกาศ Relation ด้วย Row Type

การประกาศความสัมพันธ์ด้วย ROW TYPE สามารถทำได้ดังตัวอย่างต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

OF TYPE <row type name>

เช่น CREATE TABLE MovieStar OF TYPE StarType;

6.2.2 การเข้าถึงคอมโพเนนท์ของ Row Type

ใน SQL3 การเข้าถึงคอมโพเนนท์ของ Row Type สามารถทำได้โดยใช้สัญลักษณ์จุดสองจุด ตัวอย่างเช่น

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

จากตัวอย่าง name มีชนิดของข้อมูลเป็น Char การเข้าถึงคอมโพเนนท์ทำได้โดย MovieStar.name ส่วน Address มีชนิดของข้อมูลเป็น Row Type ซึ่งประกอบด้วยข้อมูลของ Street และ City สามารถเข้าถึงได้โดยใช้จุดสองจุดดังนี้ *(MovieStar.name และ MovieStar.address..street)*

6.3 Abstract Data Type (ADT) ใน SQL3

Abstract Data Type ใน SQL3 จะมีลักษณะคล้ายกับ Row Type แต่แตกต่างกันที่ Row Type ไม่มีคุณสมบัติการเอ็นแคปซูเลชั่น (encapsulation) ส่วนใน ADT ได้กำหนดให้ทำการสนับสนุนการเอ็นแคปซูเลชั่น ADT เป็นชนิดข้อมูลแบบออบเจ็กต์ ซึ่งชนิดข้อมูลนี้เราสามารถที่จะระบุให้มีข้อมูล (attributes) และตัวดำเนินการ (operation) ต่างๆ ซึ่งจะใช้สำหรับการทำเปรียบเทียบหรือทำ เรียงข้อมูล (order ของ ADT) และใช้สำหรับการติดต่อกับภายนอก ซึ่งความแตกต่างระหว่าง ADT กับ row type ก็คือ Row type

6.3.1 การกำหนด ADT

จากตัวอย่าง บรรทัดที่ (1) เป็นการสร้างข้อมูลชนิด ADT ส่วนบรรทัดที่ (2) จะแสดงแอตทริบิวท์และข้อมูลชนิดต่างๆ

```
1) CREATE TYPE <type name> (
2)      list of attribute and their types
3)      optional declaration of = and < function for the type
4)      declaration of functions (methods) for the type
5)  );
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในบรรทัดที่ (1) <type_name> คือชื่อของ ADT ที่ต้องการประกาศ จากนั้นก็เป็นการประกาศแอตทริบิวท์ต่างๆ และชนิดข้อมูลของแต่ละแอตทริบิวท์และตามด้วยการประกาศฟังก์ชันที่ใช้สำหรับนำมาใช้ในการเปรียบเทียบซึ่งอนุญาติให้ประกาศได้สองเครื่องหมายคือ = และ < โดยในการประกาศจะใช้คำว่า QUALS และ LESS THAN และตามด้วยชื่อของฟังก์ชันที่เราสร้างขึ้นมาเพื่อสนับสนุนการเปรียบเทียบซึ่งการประกาศฟังก์ชันนี้เป็นทางเลือก ซึ่งถ้าเราไม่ได้สร้างฟังก์ชันนี้ขึ้นมาก็สามารถที่จะใช้คำว่า DEFAULT เพื่อให้ ADT ใช้ฟังก์ชันที่มีอยู่แล้วกับ ADT นี้ได้เลย แต่ถ้าไม่ต้องการให้ใช้เครื่องเหล่านี้กับ ADT ที่เรากำหนดก็ใช้คำว่า NONE แทน และส่วนสุดท้ายของการประกาศ ADT ก็คือส่วนของการประกาศฟังก์ชันต่างๆ ที่ต้องการสร้างขึ้นเพื่อใช้กับข้อมูลของ ADT ของที่ประกาศ ใน SQL3 ได้จัดเตรียม built-in function โดยไม่ต้องประกาศไว้ดังนี้

- ฟังก์ชัน ***constructor*** เป็นฟังก์ชันที่กำหนดให้ ADT โดยฟังก์ชัน constructor จะถูกกำหนดโดยอัตโนมัติโดยระบบเพื่อใช้สร้าง instances ของ ADT โดยชื่อของ constructor จะมีชื่อเดียวกับ ADT ที่เรากำหนด โดยฟังก์ชันนี้จะส่งค่าข้อมูลใหม่ให้แก่ออบเจ็กต์ โดยที่แอตทริบิวท์ของออบเจ็กต์จะเป็นค่า NULL ถ้า ***T*** ( ) เป็นชื่อของ ADT แล้ว ***T( )*** จะเป็นฟังก์ชัน ***constructor***

- ฟังก์ชัน ***Observer*** ฟังก์ชันจะถูกกำหนดให้มีชื่อเดียวกับแอตทริบิวท์ของ ADT และมีพารามิเตอร์ 1 ตัว ซึ่งเป็นชนิดเดียวกับ ADT ที่กำหนดโดยฟังก์ชันนี้จะใช้สำหรับส่งค่าของแอตทริบิวท์นั้นกลับ ถ้า ***A*** เป็นชื่อของแอตทริบิวท์และ ***X*** เป็นตัวแปรของค่าในออบเจ็กต์ ADT ดังนั้น ***A(X)*** จะเป็นค่าของ แอตทริบิวท์ ***A*** ของออบเจ็กต์ ***X*** ซึ่งสามารถเขียนให้อยู่ในรูปแบบ ***X.A*** ได้โดยมีความหมายเหมือนกับ ***A(X)***

- ฟังก์ชัน ***Mutator*** ใช้เพื่อทำการกำหนดค่าของแอตทริบิวท์ใหม่ โดยฟังก์ชั่นจะมีชื่อเดียวกันกับแอตทริบิวท์แต่จะมีพารามิเตอร์อยู่สองตัว โดยที่ตัวแรกจะเป็นชนิดเดียวกับ ADT ที่กำหนดและตัวที่สองจะเป็นชนิดข้อมูลของแอตทริบิวท์นั้น ซึ่งฟังก์ชันนี้จะทำการกำหนดค่าใหม่ให้กับแอตทริบิวท์ โดยใช้ค่าจากพารามิเตอร์ตัวที่สอง และจะทำการส่งค่า นี้มาให้ โดยระดับของการเอ็นแคปซูเลชั่นของทั้งสองฟังก์ชันจะขึ้นอยู่กับระดับการกำหนดเอ็นแคปซูเลชั่นของแอตทริบิวท์ใน ADT ของฟังก์ชันนั้นๆ ว่าอยู่ระดับใด

- ***Distinct data type*** เป็นการกำหนดชนิดของข้อมูลขึ้นมาใหม่โดยการใช้คุณสมบัติของการแสดงถึงข้อมูลชนิดอื่นที่มีอยู่แล้ว (source type) โดยใน SQL3 มาตรฐานนั้นได้กำหนดให้ distinct data type descriptor จะเป็น abstract data type ที่มีคอมโพเนนท์ต่างๆ เหมือนกัน descriptor ของ source type ที่เรากำหนดให้ distinct data type

- ***Encapsulation*** คือสามารถที่จะกำหนดระดับการเข้าถึงของคอมโพเนนท์ต่างๆ ภายใน ADT ได้หลายระดับดังนี้ public, private และ protected

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ***Subtypes และ supertype*** ของ ADT เป็นการทำถ่ายทอดคุณสมบัติของ ADT ที่สามารถที่จะให้มีการสืบทอดคุณสมบัติต่างๆของ ADT หนึ่งไปให้กับ ADT หนึ่งได้ โดยสมมุติเรากำหนดให้ ADT Ta จะเป็น subtype ของ ADT Tb ถ้าในส่วนของการประกาศ Ta ได้บ่งชี้ว่าตัวมันเป็น subtype ของ Tb แล้ว Ta จะถูกเรียกว่าเป็น direct subtype ของ Tb โดยที่ข้อมูลชนิด Ta จะเป็น subtype ของ Tb ก็ต่อเมื่อ
  - Ta และ Tb เป็นชนิดข้อมูลที่เหมือนกัน
  - Ta เป็น direct subtype ของ Tb หรือ
  - ถ้ามีข้อมูลชนิด Tc อยู่ และ Ta เป็น direct subtype ของ Tc และ Tc เป็น subtype ของ Tb

ชนิดข้อมูล Tb จะถูกเรียกว่าเป็น supertype ของ Ta ถ้า Ta เป็น subtype ของ Tb ถ้า Ta เป็น direct subtype ของ Tb แล้ว Tb จะถูกเรียกว่าเป็น direct supertype ของ Ta สำหรับชนิดข้อมูลที่ไม่ได้เป็น subtype ของ type ใดๆ เราจะเรียกว่า maximal supertype

ถ้าให้ Ta เป็น maximal subtype และให้ T เป็น subtype ของ Ta กลุ่มของ subtype ของ Ta โดยรวมชนิดข้อมูล Ta ด้วย เราจะเรียกกลุ่มของชนิดข้อมูลนี้ว่าเป็น subtype family ของ T หรือ ของ Ta ส่วน leaf type คือชนิดข้อมูลใดๆที่ไม่มี subtype เลย

instance ของ subtype ใดๆ จะเป็น instance ของ supertype ของมันด้วย โดยแต่ละ instance จะมีคุณสมบัติเฉพาะเจาะจงที่ถูกกำหนดโดยชนิดข้อมูลนั้นๆ ซึ่งในการที่จะสร้าง instance ขึ้นมาเราไม่จำเป็นที่จะใช้ชนิดข้อมูลที่ leaf type เท่านั้น เช่นสมมุติเรามีชนิดข้อมูล PERSON โดยมี STUDENT และ EMPLOYEE เป็น subtype และ STUDENT ยังมี UG_STUDENT และ PG_STUDENT เป็น subtype อีกที ในการสร้าง instance เราอาจจะใช้ชนิดข้อมูล STUDENT ได้โดยไม่จำเป็นต้องเป็นชนิดข้อมูลที่ leaf type

ถ้า Ta เป็น subtype ของ Tb แล้ว instance ของ Ta สามารถที่จะถูกใช้แทน instance ของ Tb เช่นเราสามารถที่จะกำหนดค่าของ Tb ด้วยค่าของ Ta หรืออาจจะทำการส่งค่า Ta ให้กับ argument ของฟังก์ชันที่เป็นชนิดของ Tb ได้

ในการที่จะทำการประการชนิดข้อมูลให้เป็น subtype ของชนิดข้อมูลอื่นๆ นั้นจะต้องมีการประกาศหรือใช้คำสั่ง UNDER ก่อน โดยชนิดข้อมูลสามารถที่จะมี subtype ได้หลาย subtype เช่นเดียวกับ supertype ที่ชนิดข้อมูลหนึ่งๆ จะมีหลาย supertype ได้

- ***Type templates*** เป็นชนิดข้อมูลที่ใช้ในการกำหนดกลุ่มของ ADT โดยในการกำหนด type template แต่ละตัวจะประกอบด้วยกลุ่มของ formal paramater และ body โดยในส่วนของ parameter จะใช้ในการกำหนดค่าหรือชนิดของข้อมูลที่ต้องใช้ในการนำ type template ไปใช้ในการ genarate type (ADT type) ส่วนของ body จะส่วนที่ใช้ในการกำหนดใน ADT

ตัวอย่างของการประกาศ ADT

```
CREATE TYPE  AddressADT  (
        street char(40),
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
city    char(20),
EQUALS addrEq,
LESS  THAN  addrLT
        other functions could be declared here
);
```

6.3.2 การกำหนดเมทธอดของ ADT

หลังจากประกาศแอตทริบิวท์ของ ADT สามารถทำการเพิ่มเติมฟังก์ชั่นได้โดยรูปแบบต่อไปนี้

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

ตัวอย่างฟังก์ชั่นของ ADT

```
FUNCTION AddressADT(:s CHAR(50),:c CHAR(20) RETURNS AddressADT;
:a AddressADT;
BEGIN
        :a := AddressADT();
        :a.street := :s;
        a.city := :c;
        RETURN :a;
END;

FUNCTION addrEq(:a1 AddressADT, :a2 AddressADT)
RETURNS BOOLEAN;
RETURN(:a1.street = : a2.street AND :a1.city = :a2.city);

FUNCTION addrLT(:a1 AddressADT,:a2 AddressADT)
RETURNS BOOLEAN;
RETURN ((:a1.city<a2.city) OR(:a1.city = :a2.city) AND
:a1.street<:a2.street));

FUNCTION fullAddr(:a AddressADT) RETURN CHAR(82);
:z CHAR(10);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
BEGIN
        :z = findZip(:a.street,:a.city);
        RETURN(:a.street || ' ' || :a.city || ' ' || :z);
        END;
```

6.4 Collection data type

เป็นชนิดข้อมูลซึ่งประกอบด้วยสมาชิกของข้อมูล (elements) โดยสมาชิกของ collection data type จะต้องเป็นข้อมูลชนิดเดียวกันโดยทุกสมาชิกของ collection จะอยู่ในคอลัมน์เดียวกันหมด โดยข้อมูลชนิด collection แบ่งออกเป็น 3 ชนิดคือ set, multiset และ list โดยชนิดของข้อมูลใน collection data type (element) อาจจะเป็น built-in data type , abstract data type หรือ collection เองก็ได้โดยที่ชนิดข้อมูลแบบต่างๆ แสดงไว้ในตาราง

Collection data type	คุณสมบัติ
Set	เป็นชนิดของ collection type ที่ไม่มีลำดับของข้อมูลและข้อมูลภายในจะซ้ำกันไม่ได้
Multiset	มีลักษณะคล้ายกับชนิดข้อมูลแบบ set แต่สามารถที่จะมีข้อมูลที่ซ้ำกันได้
List	เป็น collection type ที่มีลำดับของข้อมูลและสามารถที่จะมีค่าข้อมูลซ้ำกันได้

ตารางที่ 6-1แสดงชนิดของ Collection data type แบบต่างๆ

6.5 Reference

เป็นข้อมูลชนิดหนึ่งซึ่งมีประโยชน์ในการจัดการกับฐานข้อมูล ซึ่งจะเปรียบเสมือนชนิดข้อมูลแบบ pointer ในภาษา C และ C++ นั่นเองโดยมีวิธีการประการดังนี้

```
<component_name> REF (<type>);
```

โดยที่ <component_name> คือชื่อของคอมโพเนนท์ (component) ที่ต้องการประกาศให้มีชนิดเป็น reference ส่วน <type> คือชนิดของข้อมูลที่ต้องการให้ คอมโพเนนท์ นี้ชี้ ตัวอย่างเช่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

bestMovie REF(MovieType);

ซึ่งมีความหมายว่า bestMovie เป็นคอมโพเนนท์ชนิด reference ที่ใช้ชี้ไปยังข้อมูลชนิด MovieType วิธีการเข้าถึงข้อมูลที่ bestMovie ชี้อยู่ทำได้ด้วยการระบุชื่อและตามด้วยสัญลักษณ์ -> แล้วตามด้วยชื่อของ คอมโพเนนท์ ที่อยู่ใน MovieType ได้เลย (เหมือนการอ้างถึงของข้อมูล Reference type ในภาษา C++)

ข้อมูลแบบ reference นี้สามารถประยุกต์ใช้ได้กับการสร้างความสัมพันธ์แบบ many-to-many ได้ ในมาตรฐานของ SQL รุ่นเก่าๆ การสร้างความสัมพันธ์แบบ many-to-many สามารถทำได้โดยการแยกตารางออกมาหนึ่งตารางโดยที่ข้อมูลในตารางก็คือการจับคู่ระหว่าง key ของแต่ละ tuple ที่มีความสัมพันธ์กัน แต่สำหรับใน SQL3 แล้วยอมให้มีการใช้ข้อมูลแบบ reference แทนการใช้ key เพื่อใช้ในการอ้างถึง ออบเจ็กต์ ที่มีความสัมพันธ์กันได้โดยตรง

หากเราต้องการสร้างข้อมูลที่เก็บรายละเอียดของภาพยนต์แต่ละเรื่องและรายละเอียดของดาราแต่ละคน โดยที่หนังหนึ่งเรื่องจะประกอบด้วยดาราหลายคนและในขณะเดียวกัน ดาราหนึ่งคนก็สามารถแสดงหนังได้หลายเรื่อง จะเห็นว่าเป็นความสัมพันธ์แบบ many-to-many เราสามารถนำข้อมูลชนิด reference มาใช้ได้ดังนี้

```
CREATE ROW TYPE MovieType
(
        title     CHAR(30),
        year      INTEGER,
        inColor   BIT(1)
);
CREATE ROW TYPE AddressType
(
        Street    CHAR(50),
        City      CHAR(20)
);

CREATE ROW TYPE StarType
(
        name      CHAR(50),
        address   AddressType
);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
CREATE ROW TYPE StartInType
(
        start     REF(StarType),
        movie     REF(MovieType)
);

CREATE TABLE Movie OF TYPE MovieType;
CREATE TABLE MovieStar OF TYPE StarType;
CREATE TABLE StarIn OF TYPE StarInType;
```

จากตัวอย่างข้างต้นเป็นการสร้างความสัมพันธ์จากที่ได้กล่าวไว้ ซึ่งหากเราต้องการทราบรายชื่อของหนังทั้งหมดที่แสดงโดย Mel Gibson สามารถจะใช้คิวรี่ดังนี้

```
SELECT Movie->title
FROM StarIn
WHERE Star->name='Mel Gibson'
```

6.6 เปรียบเทียบ SQL3 กับ SQL92

นอกจากข้อดีต่างๆ ของภาษา SQL3 ที่ได้กล่าวไป SQL3 ยังมีข้อดีในการที่จะ code ง่ายกว่าภาษา SQL92 เนื่องจาก SQL3 มี "object navigation query" (การทำคิวรี่โดยอาศัย OID เพื่ออ้างถึงออบเจ็กต์อื่นๆ) ที่สามารถที่จะนำมาใช้ในการทำคิวรี่แทนการทำ "join query" ของภาษา SQL92 ดังตัวอย่างคิวรี่ต่อไปนี้

```
SELECT id,name,state,dno(major),name(major),building(major)
FROM Sutent
```

จากตัวอย่างคิวรี่ที่เห็นจะเห็นว่า id,name และ state เป็นแอตทริบิวต์ของออบเจ็กต์ student ส่วน department number,department name และ building เป็นแอตทริบิวต์ที่มาจากออบเจ็กต์ department โดยอาศัยความสัมพันธ์จาก major ที่อยู่ในออบเจ็กต์ student ในการที่จะเอาข้อมูลจากออบเจ็กต์ department อีกที่หนึ่ง ซึ่งจะเห็นว่าเป็นการแสดงข้อมูลของ department โดยไม่ต้องมีการอ้างออบเจ็กต์ department ตรงๆ ซึ่งถ้าเป็นการทำคิวรี่เดียวกับนี้โดยการใช้ภาษา SQL92 เราจะต้องทำการป้อนคิวรี่ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
SELECT s.id,s.name,s.state,d.dno,d.name,d.building
FROM Department d,Student s
WHERE s.majorDept=d.deptNo
UNION ALL
SELECT s.id,s.name,s.state,d.dno,d.name,d.building
FROM Department d,TA s
WHERE s.majorDept=d.deptNo
```

ซึ่งนอกจากนี้ข้อดีดังกล่าวแล้ว SQL3 ยังมีความสามารถที่จะเรียกฟังก์ชันที่ผู้ใช้เป็นคนสร้าง (user-derfined method) เช่นสมมุติเราต้องการที่จะทำการคิวรี่หาระยะทางระหว่าง staff member (โดยที่ staff เป็นออบเจ็กต์ ที่อยู่ใน staff คลาส) กับ staff member ที่มี ID 6966. สามารถทำได้ดังนี้

SQL3	*SQL92*
SELECT distance(s1.place,s2.place) FROM Staff s1,Staff s2 WHERE s1.id=6966	SELECT SQRT((s1.latitude-s2.latitude)* (s1.latitude-s2.latitude)+ (s1.longtitude-s2.longtitude)* (s1.longtitude-s2.longtitude)) FROM Staff s1,Staff s2 WHERE s1.id=6966

ตาราง 6-2 ตารางแสดง เปรียบเทียบภาษา SQL3 กับ SQL92

6.7 SQL3 กับการเรียกตัวเอง (SQL3 & Recursion)

SQL3 มีความสามารถในการที่ทำคิวรี่ในลักษณะของการเรียกตัวเอง (Recursive) กล่าวคือมีการเรียกตัวเอง ซึ่งจากความสามารถดังกล่าวนี้เองจะมีประโยชน์มากในการช่วยลดจำนวนข้อมูลที่จะเก็บในตาราง เช่นสมมุติว่าเรามีตารางที่เก็บข้อมูลของสายการบินของ Untried Airlines(UA) และ Arcane Airlines(AA) ซึ่งมีเที่ยวบินไปเมืองต่างๆ ดังนี้ San Francisco,Denver,Dallas,Chicago และ New York โดยมีการเก็บข้อมูลในตาราง ซึ่งประกอบด้วยแอตทริบิวต์ดังนี้ สายการบิน,เมืองต้นทาง,เมืองปลายทาง,เวลาออกและเวลาที่ถึง ถ้าเราต้องการทราบว่าเราสามารถที่จะเดินทางจากเมืองไหนไปอีกเมืองไหนได้บ้างจะเห็นว่าในการที่จะตอบปัญหาเช่นนี้จะต้องมีการเก็บข้อมูลว่าแต่ละเมืองสามารถที่จะเดินทางไปเมืองใดได้บ้างแต่ใน กรณีที่เราใช้คิวรี่แบบเรียกตัวเองเราสามารถที่จะเก็บเฉพาะเมืองที่สามารถที่จะเดินถึงกันโดยตรงดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Airline	From	To	Depart	Arrives
UA	SF	DEN	9:30	12:30
AA	SF	DAL	9:00	14:30
UA	DEN	CHI	15:00	18:00
UA	DEN	DAL	14:00	17:00
AA	DAL	CHI	15:30	17:30
AA	DAL	NY	15:00	19:30
AA	CHI	NY	19:00	22:00
UA	CHI	NY	18:30	21:30

ตาราง 6-3 แสดงถึงตาราง Flights

ซึ่งจากคำถามข้างต้นเราสามารถที่จะเขียนแสดงในรูปของ Rule คล้ายๆ กับการเขียน Rule ในภาษา Prolog เพื่อที่จะทำการ resolution ได้ดังนี้

1. Reaches(x,y) <- Flights(a,x,y,d,r)

2. Reaches(x,y) <- Reaches(x,z) AND Reaches(z,y)

โดยที่กฎข้อที่ 1 เป็นการระบุให้ Reaches ประกอบด้วยคู่ลำดับของเมืองที่มีสายการบินเดินทางถึงกันได้โดยตรง ส่วนกฎข้อที่ 2. เป็นการหาเมืองต่างๆ ที่สามารถที่จะเดินทางถึงกันได้โดยอาศัยผลที่ได้จากกฎข้อ 1 อีกทีหนึ่ง ซึ่งจากกฎข้อที่ 1 เป็นการเริ่มหาข้อเท็จจริง (fact) ต่างๆ มาจากตาราง Fligths เราเรียก Relation ที่เริ่มต้นนี้ว่า extensional database relation (EBR) และเรียก Relation ที่ได้จากการทำ resolution จะกฎข้อ 2 เรียกว่า intensional database relation (IBR)

ซึ่งจาก EBR และ IBR เราสามารถที่จะทำการแปลงเป็นภาษา SQL3 โดยใช้ คำสั่ง WITH ซึ่งจะได้เป็นคิวรี่ดังนี้

```
WITH RECURSIVE Reaches(frm,to) AS
        (SELECT frm,to FROM Flights)
UNION
        (SELECT R1.frm,R2.to
        FROM Reaches AS R1,Reaches AS R2
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                WHERE R1.to=R2.frm)
        SELECT * FROM Reaches;
```

ซึ่งในการคิดคิวรี่แบบเรียกตัวเองให้คิดถึงการทำงานของภาษา Prolog ในการที่จะทำการ resolution กฎต่างๆ ที่เราได้กำหนดให้จากดังที่ได้แสดงไว้ในตัวอย่างจากนั้นก็ทำการแปลงจากกฎที่เรากำหนดมาเป็นคำสั่ง SQL โดยใช้คำสั่ง WITH

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

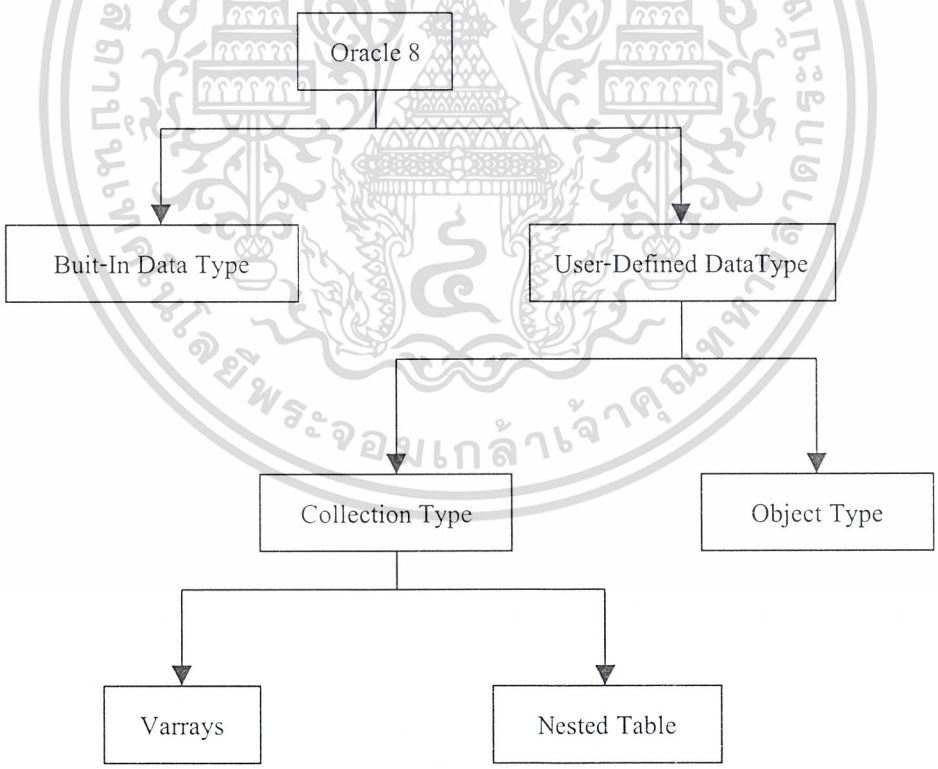
บทที่ 7

ระบบจัดการฐานข้อมูล ORACLE 8

(ORACLE 8 Database management System)

Oracle 8 เป็นระบบฐานข้อมูลแบบเชิงวัตถุสัมพันธ์ (Object Relational Database Management System : ORDBMS) ซึ่งเป็นระบบฐานข้อมูลที่สามารถรองรับข้อมูลที่มีความซับซ้อนมาก ๆ เช่น ข้อมูลมัลติมีเดีย (Multimedia), รูปภาพ, เสียง เป็นต้น สามารถจัดการกับข้อมูลเหล่านี้ได้อย่างมีประสิทธิภาพ โดย Oracle8 ได้จัดเตรียมชนิดข้อมูลชนิดใหม่ขึ้นมาเพื่อรองรับกับข้อมูลที่มีความซับซ้อนมาก ๆ เหล่านี้ โดยเฉพาะยอมให้มีข้อมูลชนิดที่ผู้ใช้กำหนดขึ้นมาเอง (User-Defined Datatypes) ข้อมูลชนิดนี้จะมีคุณสมบัติทาง Object Oriented เช่นเดียวกับ Object ในโปรแกรมภาษาที่สามารถเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) ต่าง ๆ แต่คุณสมบัติบางอย่างอาจยังไม่เทียบเท่าโปรแกรมภาษาเหล่านั้น เช่น คุณสมบัติการสืบทอด(Inheritance) ณ. Version ปัจจุบัน (Oracle 8.0.4) ยังไม่สามารถทำได้ ซึ่งจะได้อธิบายรายละเอียดต่อไปนี้

7.1 ชนิดข้อมูลของ Oracle 8 (Oracle8 Datatype)



รูปที่ 7-1 แสดงโครงสร้างของ Type ใน Oracle 8

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.1.1 Built-In DataType

Built-In DataType เป็นชนิดของข้อมูลที่มีอยู่ใน Oracle ได้แก่

7.1.1.1 ข้อมูลชนิด Character ประกอบด้วยชนิดข้อมูล 5 ชนิด คือ

1. ข้อมูลชนิด CHAR

ชนิดของข้อมูลประเภท CHAR เป็นชนิดของข้อมูลตัวอักษรที่มีขนาดของความยาวคงที่ โดยที่เมื่อมีการสร้างตารางที่มีคอลัมน์เป็นชนิดข้อมูลประเภท CHAR จะต้องมีการกำหนดความยาวข้อมูลเป็นไบท์ระหว่าง 1 และ 2000 ไบท์ (โดยจะมีค่าเริ่มต้นเป็น 1) โดยออราเคิลจะมีการรับประกันว่า

- เมื่อมีการ Insert และ Update แถวในตาราง ค่าที่เก็บอยู่ในคอลัมน์ที่มีข้อมูลประเภท CHAR จะมีขนาดคงที่ตามที่กำหนดไว้
- ถ้าหากข้อมูลที่จะ Insert หรือ Update มีขนาดของความยาวน้อยกว่าที่กำหนดไว้ ข้อมูลจะมีการเติมช่องว่างให้ได้ขนาดตามที่กำหนดไว้
- ถ้าหากข้อมูลที่จะ Insert หรือ Update มีขนาดของความยาวมากกว่าเนื่องจากช่องว่างตรงส่วนท้ายของข้อมูล ช่องว่างเหล่านั้นจะถูกตัดออกจนได้ขนาดตามที่กำหนดไว้
- ถ้าหากข้อมูลมีขนาดของความยาวมาก จะมีการ return ค่าเป็น error ออกมา

2. ข้อมูลชนิด VARCHAR2

ชนิดของข้อมูลประเภท VARCHAR2 เป็นชนิดของข้อมูลตัวอักษรที่มีขนาดของความยาวเปลี่ยนแปลงได้ โดยที่เมื่อมีการสร้างตารางที่มีคอลัมน์เป็นชนิดของข้อมูลประเภท VARCHAR2 จะต้องมีการกำหนดความยาวของข้อมูลที่มากที่สุดเป็นไบท์ โดยสามารถมีค่าได้ระหว่าง 1 และ 4000 ดังนั้นข้อมูลที่เก็บไว้จะมีขนาดเท่าใดก็ได้ที่อยู่ระหว่าง 1 และ ความยาวสูงสุด (แต่จะมีการ return ค่า error ออกมาถ้าหากข้อมูลมีขนาดใหญ่กว่าความยาวสูงสุด)

3. ข้อมูลชนิด VARCHAR

ชนิดของข้อมูลประเภท VARCHAR เป็นชนิดของข้อมูลที่มีลักษณะเหมือนกับชนิดของข้อมูลประเภท VARCHAR2 แต่อย่างไรก็ตามเวอร์ชั่นของออราเคิลในอนาคต ชนิดของข้อมูล VARCHAR จะมีการเก็บขนาดของข้อมูลที่สามารถเปลี่ยนแปลงได้

4. ข้อมูลชนิด NCHAR และ NVARCHAR2

ชนิดของข้อมูลประเภท NCHAR และ NVARCHAR2 เก็บข้อมูลประเภท NLS โดยที่ชนิดของข้อมูลประเภท NCHAR เก็บข้อมูลที่มีขนาดคงที่ที่สอดคล้องกับข้อมูลประเภทความยาวคงที่ (fixed-length national character set) หรือความยาวเปลี่ยนแปลงได้ (variable-length national character set) สำหรับชนิดของข้อมูลประเภท NVARCHAR2 จะเก็บข้อมูลที่มีความยาวเปลี่ยนแปลงได้ ดังนั้นเมื่อมีการสร้างตารางที่มีคอลัมน์ของข้อมูลประเภท NCHAR หรือ NVARCHAR2 จะต้องมีการกำหนดขนาดที่มากที่สุดได้ทั้งหน่วยตัวอักษร (สำหรับข้อมูลประเภทความยาวคงที่ fixed-length national character set) หรือหน่วยไบท์ (สำหรับข้อมูลประเภทความยาวเปลี่ยนแปลงได้ variable-length national character set)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ขนาดของความยาวที่มากที่สุดสำหรับชนิดของข้อมูลประเภท NCHAR คือ 2000 ไบท์ หรือจำนวนของตัวอักษรสามารถเก็บได้ 2000 ไบท์

- ขนาดของความยาวที่มากที่สุดสำหรับชนิดของข้อมูลประเภท NVARCHAR2 คือ 4000 ไบท์หรือจำนวนของตัวอักษรสามารถเก็บได้ 4000 ไบท์

5. ข้อมูลชนิด LONG

ชนิดของข้อมูลประเภท LONG จะใช้ในการเก็บข้อมูลที่มีขนาดมากถึง 2 gigabytes

7.1.1.2 ข้อมูลชนิด NUMBER

ชนิดของข้อมูลประเภทตัวเลข จะใช้ในการเก็บตัวเลขจำนวนเต็ม (fixed numbers) และตัวเลขที่มีจุดทศนิยม (floating-point numbers) เก็บขนาดได้ถึง 21 ไบท์

7.1.1.3 ข้อมูลชนิด DATE

ชนิดของข้อมูลประเภทวัน จะเก็บข้อมูลที่เป็นวันและเวลา โดยจะมีการเก็บปี , เดือน , วัน , ชั่วโมง , นาที และวินาที มีขนาดคงที่ 7 ไบท์

7.1.1.4 ข้อมูลชนิด LOB

1. ข้อมูลชนิด BLOB

ชนิดของข้อมูลประเภท BLOB จะเก็บข้อมูลประเภทไบนารี โดยสามารถเก็บข้อมูลไบนารีได้ถึง 4 gigabytes

2. ข้อมูลชนิด CLOB และ NCLOB

ชนิดของข้อมูล CLOB และ NCLOB จะเก็บข้อมูลประเภท ตัวอักษร ชนิดของข้อมูลประเภท CLOB จะเก็บข้อมูลประเภทที่เป็น Single-byte สำหรับชนิดของข้อมูลประเภท NCLOB จะใช้เก็บข้อมูลประเภท fixed-length หรือ NCHAR โดยชนิดของข้อมูลทั้งสองนี้สามารถเก็บข้อมูล 4 gigiabytes

3. ข้อมูลชนิด BFILE

ชนิดของข้อมูลประเภท BFILE จะเก็บข้อมูลประเภทไบนารีที่เป็นแฟ้มข้อมูลนอกฐานข้อมูล โดยในคอลัมน์ BFILE จะเก็บค่า file locator ที่ชี้ตำแหน่งของแฟ้มข้อมูล โดยชนิดของข้อมูลประเภทนี้สามารถเก็บข้อมูลได้ถึง 4 gigabytes (ชนิดของข้อมูลประเภทนี้จะเป็นอ่านได้อย่างเดียว read-only ไม่สามารถเปลี่ยนแปลงแก้ไขข้อมูลได้)

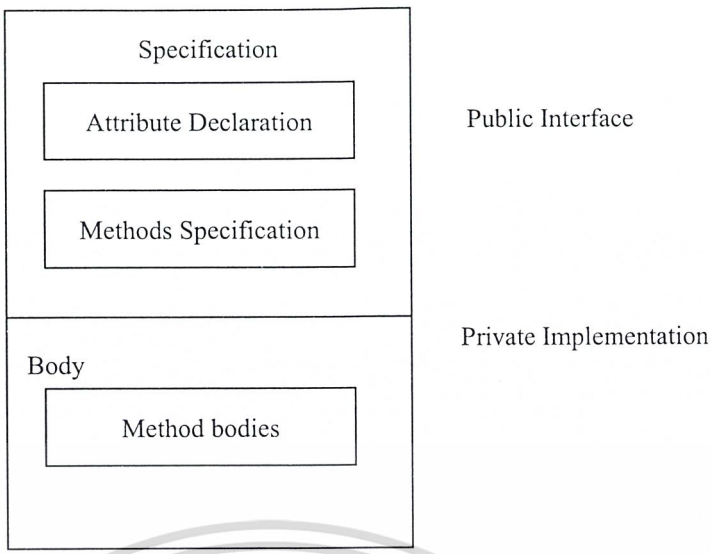
7.1.1.5 ข้อมูลชนิด RAW และ LONG RAW

ชนิดของข้อมูล RAW จะใช้ในการเก็บข้อมูลไบนารีประเภทเปลี่ยนแปลงขนาดได้ โดยที่ขนาดที่มากที่สุดจะต้องมีการกำหนดไว้ โดยสามารถมีขนาดได้สูงสุดถึง 2000 ไบท์ สำหรับชนิดของข้อมูลประเภท LONG RAW จะใช้เก็บข้อมูลไบนารีประเภทเปลี่ยนแปลงขนาดได้ โดยสามารถมีขนาดของข้อมูลสูงสุดถึง 2^31-1 ไบท์หรือ 2 gigabytes

7.1.1.6 ข้อมูลชนิด ROWID

ชนิดของข้อมูลประเภท ROWID จะใช้เก็บข้อมูลประเภทข้อมูลไบนารีที่บอกถึงตำแหน่งของแถว โดยที่มีขนาด 10 ไบท์สำหรับประเภท extended ROWID หรือ 6 ไบท์สำหรับประเภท restricted ROWID

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7-2 แสดงถึงส่วนประกอบของ Object ใน Oracle 8

ตัวอย่าง

```
create type complex as object(
        rpart real,
        ipart real,
        member function plus(x complex) return complex,
        member function less(x complex) return complex,
        member function times(x complex) return complex,
        member function divby(x complex) return complex);
create type body complex as
        member function plus(x complex) return complex is
        begin
                return complex(rpart+x.part,ipart+x.ipart);
        end plus;
        member function less(x complex) return complex is
        begin
                return complex(rpart-x,rpart,impart-x,ipart);
        end
        member function times(x Complex) return Complex is
        begin
                return Complex(rpart*x.rpart-impart*x.ipart,
                                rpart*x.ipart+ipart*x.rpart);
        end times;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        member funtion divby(x Complex) return Complex is
        z REAL := x.rpart**2+x.ipart**2;
        begin
                return Complex((rpart*x.rpart+ipart*x.ipart)/z
                                (rpart*x.rpart-rpart*x.ipart)/z);
        end divby;
end;
```

7.1.2.2 วิธีการเรียกใช้ Method ของ Object Type

การเรียกใช้ Method ของ Object Type เราสามารถเรียกใช้ได้แบบเดียวกับการเรียกใช้ Built-In Function ของภาษา SQL แต่ Method ที่สามารถเรียกใช้แบบเดียวกับการเรียกใช้ Built-In Function ของภาษา SQL จะต้องมีลักษณะดังต่อไปนี้

- Method นั้นจะต้องไม่ไปเปลี่ยนค่าข้อมูลในฐานข้อมูล (Database table) ดังนั้นใน Method จะต้องไม่มีคำสั่ง INSERT,DELETE, หรือ UPDATE
- Method ที่ถูกเรียกใช้จาก SELECT , VALUES หรือ SET สามารถที่จะเขียนค่าข้อมูลให้กับ Attribute ของ Object Type ได้
- Method นั้นไม่สามารถที่จะเรียกใช้โปรแกรมย่อย(subprogram) อื่น ๆ ที่ไม่มีลักษณะตามที่กล่าวมา

เพื่อที่จะทำให้ Method มีลักษณะตามที่ได้กล่าวมาแล้วเราจะต้องใช้ pragma (compiler directive) RESTRICT_REFERENCES ในการกำหนดลักษณะของ method

pragma เป็นตัวบอก PL/SQL Compiler จะไม่ยอมรับ Method ที่มีการเขียนหรืออ่านข้อมูลในฐานข้อมูล (database table) หรือ Attribute ของ Object หรือทั้งสองอย่าง การใช้ pragma RESTRICT_REFERENCES มีลักษณะดังนี้

```
PRAGMA RESTRICT_REFERENCES (
        Method_name,WNDS[,WNPS][,RNDS][,RNPS] );
```

- WNDS (writes no database state) คือไม่มีการเปลี่ยนแปลงข้อมูลในฐานข้อมูล (Database table) ไม่มีคำสั่ง INSERT ,DELETE หรือ UPDATE
- RNDS (read no database state) คือไม่มีการดึงข้อมูลมาจากฐานข้อมูล ซึ่งก็คือจะไม่มีคำสั่ง SELECT
- WNPS (writes no package state) คือไม่มีการเปลี่ยนแปลงค่าของตัวแปรใน package ซึ่งในที่นี้ก็คือ Attribute ของ Object
- RNPS (read no package state) คือไม่มีการอ้างถึงค่าของตัวแปรใน package

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.1.2.3 Object Type Constructor Methods

ทุก ๆ Object Type ระบบจะกำหนดให้มี Constructor ซึ่ง Method นั้นจะทำการสร้าง Object ใหม่ขึ้นมาตามที่ได้กำหนดไว้ในส่วน Specification ของ Object Type ชื่อของ constructor methods ก็คือชื่อของ Object Type นั้น

7.1.2.4 Comparision Methods

เป็น Method ที่ทำหน้าที่ในการเปรียบเทียบระหว่าง Object ซึ่ง Oracle จะช่วยในการเปรียบเทียบข้อมูล 2 ข้อมูลซึ่งมีชนิดเป็น built-in datatype ตัวอย่างเช่นเปรียบเทียบตัวเลข 2 จำนวนและบอกว่าข้อมูลหนึ่งมากกว่า , เท่ากับ หรือน้อยกว่าอีกข้อมูลหนึ่ง Oracle ไม่สามารถที่จะเปรียบเทียบข้อมูลซึ่งมีชนิดเป็นข้อมูลที่กำหนดขึ้นมาเอง (User-defined type) โดยตัวเองได้ ดังนั้น Oracle ได้เตรียมวิธีในการบอกความสัมพันธ์ระหว่าง Object ซึ่งมีชนิดเป็นข้อมูลที่กำหนดขึ้นมาเอง (User-defined type) ไว้ 2 วิธีคือ

- Map Method วิธีการนี้จะใช้ความสามารถของ Oracle ในการเปรียบเทียบข้อมูลที่มีชนิดเป็น built-in datatype ตัวอย่างเช่น เรากำหนด Object Type สี่เหลี่ยมซึ่งมี Attributes เป็นความสูงและความกว้าง เราสามารถกำหนด Map Method พื้นที่ (area) ซึ่งส่งค่าตัวเลขที่เป็นผลคูณของความกว้าง จากนั้น Oracle ก็สามารถที่จะเปรียบเทียบสี่เหลี่ยม 2 รูปโดยใช้การเปรียบเทียบพื้นที่ของสี่เหลี่ยม

ตัวอย่างการสร้าง MAP Method

```
create type purchase_order_t as object(
        Pono      number,
        Custref REF       customer_info_t,
        Orderdate         date,
        Shipdate          date,
        Map member function ret_value return number,
        Pragma RESTRICT_REFERENCES(
                  Ret_value,WNDS,RNDS,WNPS,RNPS)
);
create type body purchase_order_t as
        map member function ret_value return number is
        begin
                return pono;
        end;
end;
```

- Order Method ใช้วิธีการเปรียบภายใน Method และส่งค่าซึ่งจะบอกความสัมพันธ์ระหว่าง Object กลับออกมา เช่น ส่งค่ากลับออกมาเป็น –1 ถ้า Object แรกเล็ก จะส่งค่า 0 ถ้าเท่ากัน หรือส่งค่ากลับออกมาเป็น 1 ถ้าใหญ่กว่า

ตัวอย่างการสร้าง Order method

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
create type customer_info_t as object (
        Custno number,
        Custname varchar2(200),
        Address address_t,
        Phone_list phone_list_t,
        Order member function cust_order( x in customer_info_t ) return integer,
        Pragma restrict_references( cust_order,WNDS,RNDS,WNPS,RNPS)
);
```

สมมติว่าเรากำหนด Object type Address ขึ้นมาซึ่งประกอบด้วย street,city,state และ zip คำว่ามากกว่า หรือ น้อยกว่า จะใช้กับ Address ไม่ได้แต่เราต้องการ การทำงานที่ซับซ้อนยิ่งขึ้นเพื่อให้รู้ว่า Addresses นั้นเท่านั้น ในการกำหนด Object type เราสามารถที่จะมี Map Method หรือ Order Method ก็ได้ แต่ไม่สามารถมีทั้งสองอย่างใน Object Type เดียวกันได้ ถ้า Object type ไม่มี method ที่ใช้ในการเปรียบเทียบ (comparision method) Oracle ไม่สามารถจะบอกความสัมพันธ์ระหว่าง object ได้ว่าเล็กกว่า

หรือใหญ่กว่า Oracle จะเปรียบเทียบ 2 Object ที่ไม่มี method ที่ใช้ในการเปรียบเทียบ โดยการเปรียบเทียบ Attribute ที่เหมือนกัน

7.1.2.5 Collection type

Collection type มีลักษณะที่เป็นหน่วยของข้อมูล (data unit) ที่มีจำนวนสมาชิกไม่จำกัดและสมาชิกทั้งหมดมีชนิดข้อมูลเหมือนกัน Collection type ได้แก่ array type และ table type ซึ่งหน่วยของข้อมูลจะเรียกว่า varrays และ nestable table

Collection type จะมี constructor methods ชื่อของ constructor methods ก็คือชื่อของ collection types นั้น ๆ

1. Varrays

มีลักษณะเป็นเซ็ตของข้อมูลแบบมีลำดับ (Array)โดยสมาชิกทุกตัวจะเป็นข้อมูลชนิดเดียวกัน สมาชิกแต่ละตัวจะมีตัวชี้ (Index) ซึ่งเป็นหมายเลขที่สอดคล้องกับตำแหน่งของสมาชิกใน array จำนวนของสมาชิกใน array จำนวนของสมาชิกใน array ก็คือขนาดของ array Oracle ยอมให้ขนาดของ array เปลี่ยนแปลงได้ ซึ่งนี่ก็คือสาเหตุที่ถูกเรียกว่า varrays เราจะต้องระบุขนาดที่ใหญ่ที่สุดเมื่อเรามีการกำหนดข้อมูลชนิด array

ตัวอย่างการกำหนดข้อมุลชนิด array

```
create type prices as varray(10) of number(12,2);
```

varray ชนิด prices จะมีจำนวนสมาชิกได้ไม่เกิน 10 และสมาชิกแต่ละตัวมีชนิดเป็น number(12,2) การสร้างข้อมูลชนิด array จะไม่มีการจองเนื้อที่ แต่จะเป็นการกำหนดชนิดของข้อมูล ซึ่งเราสามารถนำไปใช้เป็น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ชนิดข้อมูลของ column ใน relational talble

- attribute ของ Object type

-เป็นตัวแปร PL/SQL , parameter หรือ ชนิดข้อมูลที่ส่งค่ากลับมาจาก function

2. Nested table

Nested Table เป็นเซ็ตของข้อมูลแบบที่ไม่เรียงลำดับ ซึ่งสมาชิกทุก ๆ ตัว จะเป็นข้อมูลชนิดเดียวกันและ Nested Table จะมีเพียง คอลัมน์ (column) เดียวและ คอลัมน์ จะเป็นข้อมูลชนิด built-in datatype หรือ เป็น Object type เราสามารถมองเป็นเหมือนกับเป็นตาราง ที่มีหลาย ๆ คอลัมน์ ได้ซึ่ง คอลัมน์ ก็คือ Attribute ของ Object type ตัวอย่างเช่น purchase order คำสั่งต่อไปนี้เป็นการกำหนดข้อมูลชนิดตาราง (table type) ของ lineitem

Create type lineitem_table as table of lineitem;

การสร้างข้อมูลชนิดตาราง จะไม่มีการจองเนื้อที่ แต่จะเป็นการกำหนดชนิดของข้อมูลซึ่งเราสามารถนำไปใช้เป็น

- ชนิดข้อมูลของคอลัมน์ใน relational table

- attribute ของ Object type

- เป็นตัวแปร PL/SQL ,parameter หรือ ชนิดของข้อมูลที่ส่งค่ากลับมาจาก function

ตัวอย่างของการสร้างตารางซึ่งมีชนิดของข้อมูลเป็น nested table

create table purchase_order_table of purchase_order

nested table lineitem store as lineitem_table;

7.2 คุณสมบัติทาง Object Oriented ของ Object type ใน Oracle 8

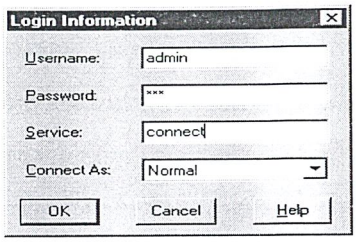
- การปกป้องข้อมูล (Encapsulation) Object Type ใน Oracle 8 จะแบ่งการ ปกป้องข้อมูล ออกเป็น 2 ส่วนคือในส่วนของ Specification จะมีการปกป้องแบบ Public และในส่วนของ Body จะมีการปกป้องเป็นแบบ private

- โพลิมอร์ฟิซึม (polymorphism) Object Type ใน Oracle 8 สามารถมี Method ที่ชื่อซ้ำกันแต่มีการทำงานที่แตกต่างกันได้

7.3 วิธีการสร้าง Object Type ของ Oracle 8 โดยใช้ Schema Manager

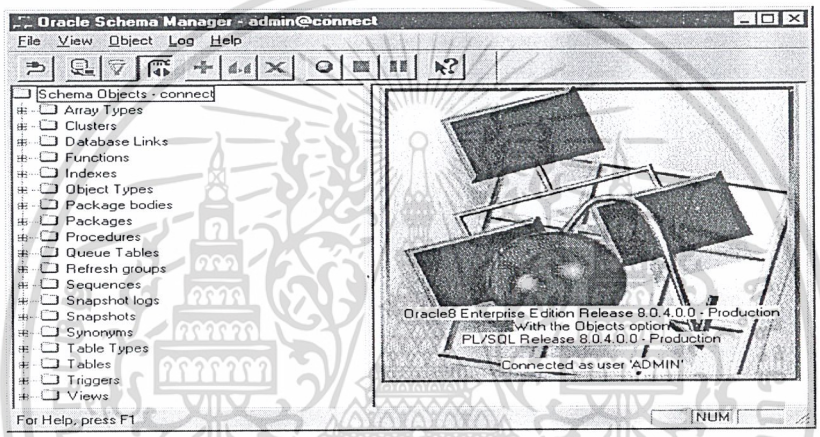
เมื่อเราเรียก Schema Manager ระบบก็จะถามชื่อผู้ที่เข้าระบบ(Username),รหัสผ่าน(Password), ชื่อบริการ(Service),ระบบการติดต่อ(Connect As) ดังรูปที่ 7-3 ซึ่งบริการเป็นบริการที่เราสร้างขึ้นเพื่อติดต่อเครื่องลูกข่ายกับเซิร์ฟเวอร์ซึ่งจะได้อธิบายในหัวข้อต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



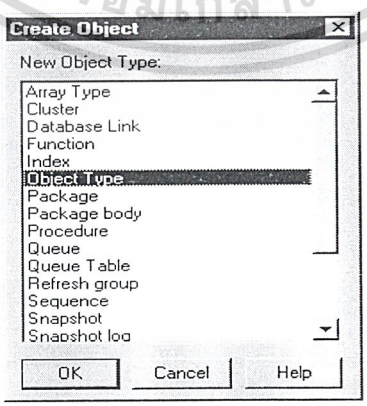
รูปที่ 7-3 แสดงถึงหน้าจอการติดต่อเครื่อง Client กับเครื่องServer

เมื่อทุกอย่างถูกต้องก็จะเข้าหน้าจอ Oracle Schema Manager ดังรูป แสดงว่าเราสามารถติดต่อเครื่องเซิร์ฟเวอร์ได้แล้วดังรูปที่ 7-4



รูปที่ 7-4 แสดงถึงการติดต่อระหว่างเครื่อง Client กับ Server

ในเมนู Oracle Schema Manager เลือก Object และให้เลือก Create Object เพื่อทำการสร้าง Object Type ซึ่งเมื่อเลือกแล้วจะได้เมนูดังรูป 7-5 แล้วก็เลือก Create Object Type



รูปที่ 7-5 แสดงการสร้าง Object Type

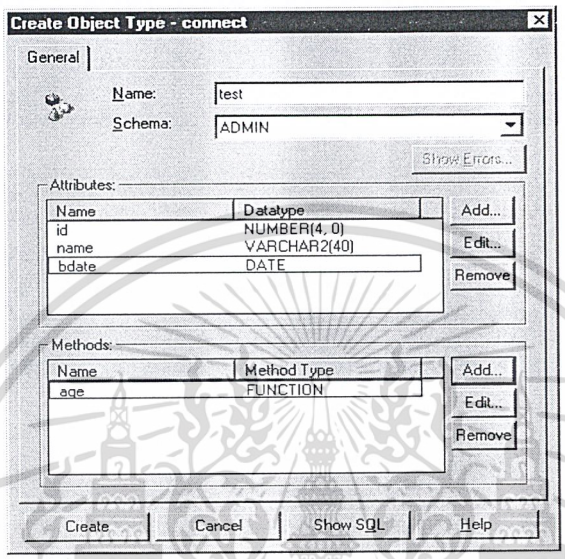
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อมาเลือกที่ Create Object Type แล้วก็จะได้หน้าจอดังรูปที่ 7-6 ซึ่งในรูปจะประกอบด้วย 3 ส่วนดังนี้

ส่วนที่ 1 เป็นชื่อของออบเจ็กค์ (Name of Object) และ Schema ซึ่ง Schema ก็คือ Object Type นี้เป็นของใคร

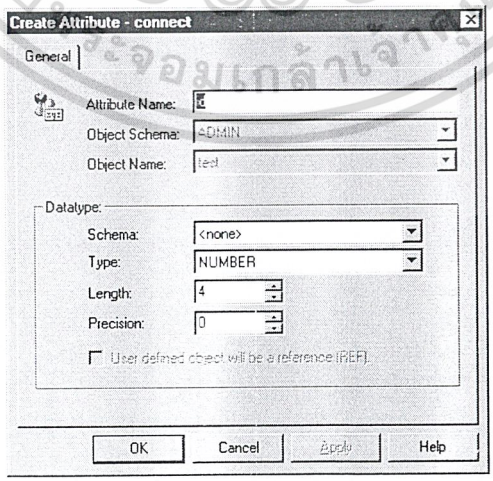
ส่วนที่ 2 เป็นส่วนของ Attribute เราสามารถเพิ่มหรือลบหรือแก้ไข Attribute ได้ในส่วนนี้

ส่วนที่ 3 เป็นส่วนของเมธธอด เราสามารถเพิ่มหรือลบหรือแก้ไขเมธธอดได้



รูปที่ 7-6 แสดงส่วนประกอบของ Object Type ใน Oracle 8

เมื่อเราเลือก add attributes จะเข้ามาในหน้าจอส่วนนี้ซึ่งเราสามารถใส่ชื่อ attributes เราสามารถเลือก Schema ในที่นี้คือเราสามารถใช้ Schema ของผู้อื่นมาเป็น Data Type ของเราได้ซึ่งถ้าเราไม่เลือก Schema ก็จะเป็น Data Type มาตราฐานของ Oracle ดังรูป 7-7

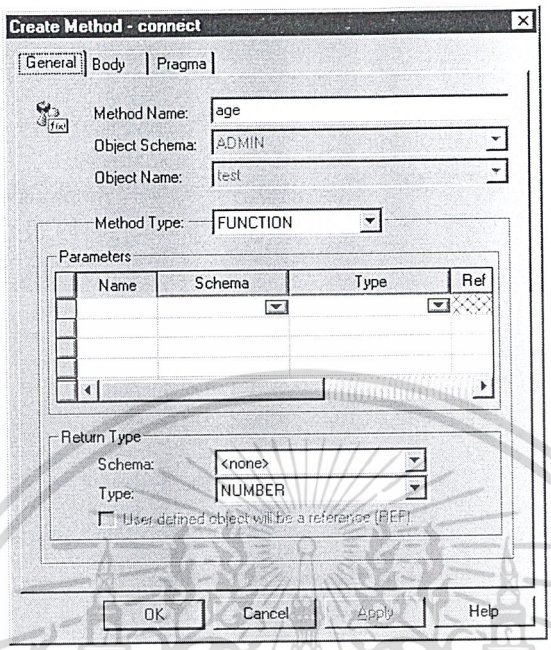


รูปที่ 7-7 แสดงถึงการ Add Attribute ลงใน Object Type

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

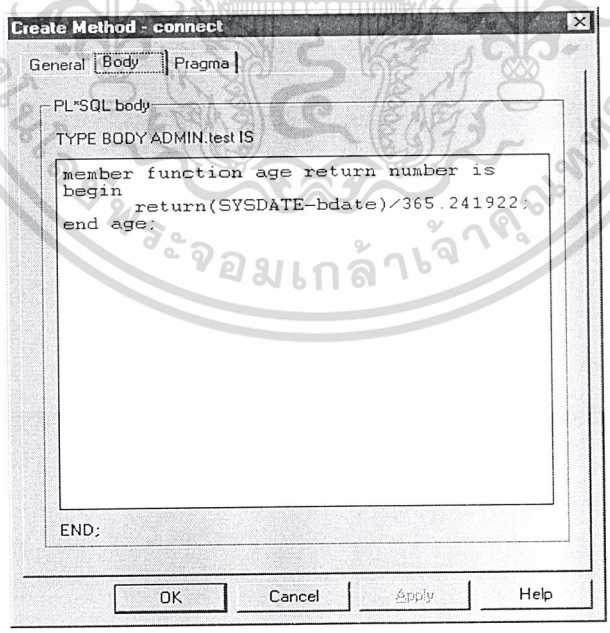
เมื่อเราเลือก Add Method จะปรากฏส่วนนี้ขึ้น ส่วนนี้ซึ่งจะประกอบไปด้วยสามส่วน คือ General , Body และ Pragma

ส่วน General เป็นส่วนที่กำหนดชื่อของ Method และ Method Type ดังรูปที่ 7-8



รูปที่ 7-8 แสดงถึงการกำหนดชื่อและชนิดของ Method

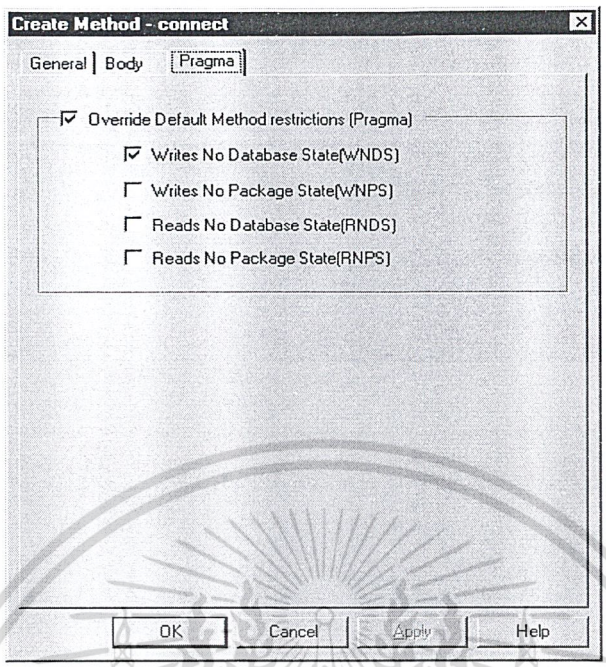
ส่วน Body เป็นส่วนที่ใช้ในการเขียน function หรือ procedure ดังรูปที่ 7-9



รูปที่ 7-9 แสดงถึงการเขียน method

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

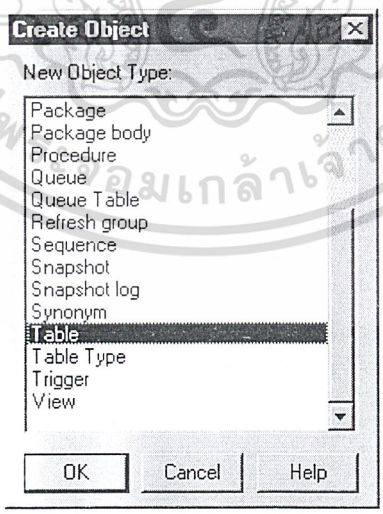
ส่วน Pragma เป็นส่วนที่ใช้การกำหนดสิทธิของ Method



รูปที่ 7-10 แสดงถึงการกำหนดสิทธิของ Method

7.4 วิธีการสร้างตารางที่เป็น Object Table

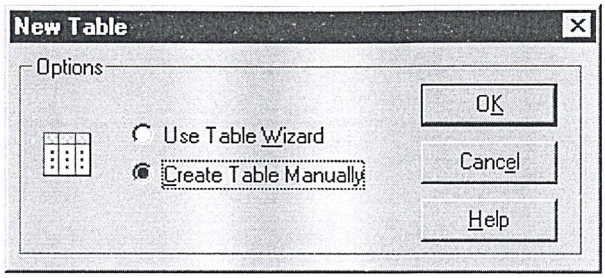
จาก Oracle Schema Manager ในเมนู Object เลือก Create จะได้ดังรูปแล้วเลือก Create Table ดังรูป 7-11



รูปที่ 7-11 แสดงถึงการสร้างตาราง

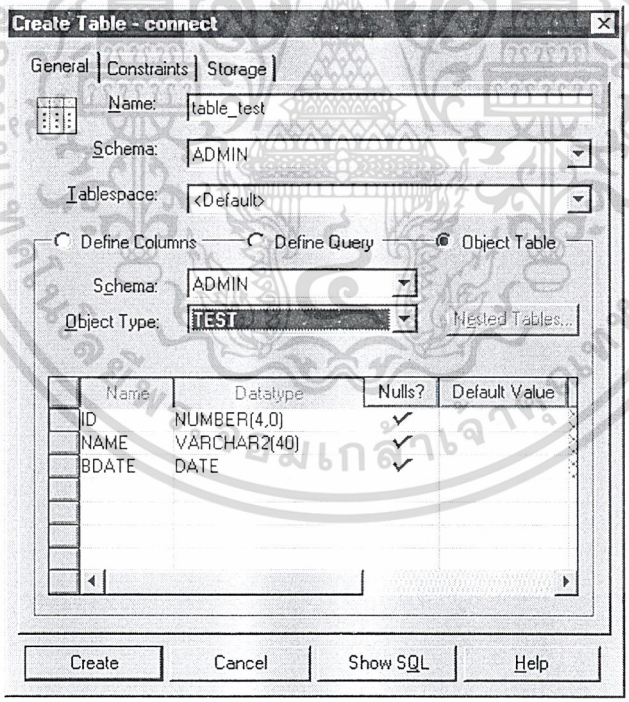
เมื่อเลือก Create Table จะ ได้ดังรูปที่ 7-12 แล้วให้เลือก Create Table แบบ Manually

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7-12 แสดงถึงการสร้างตารางแบบ Manually

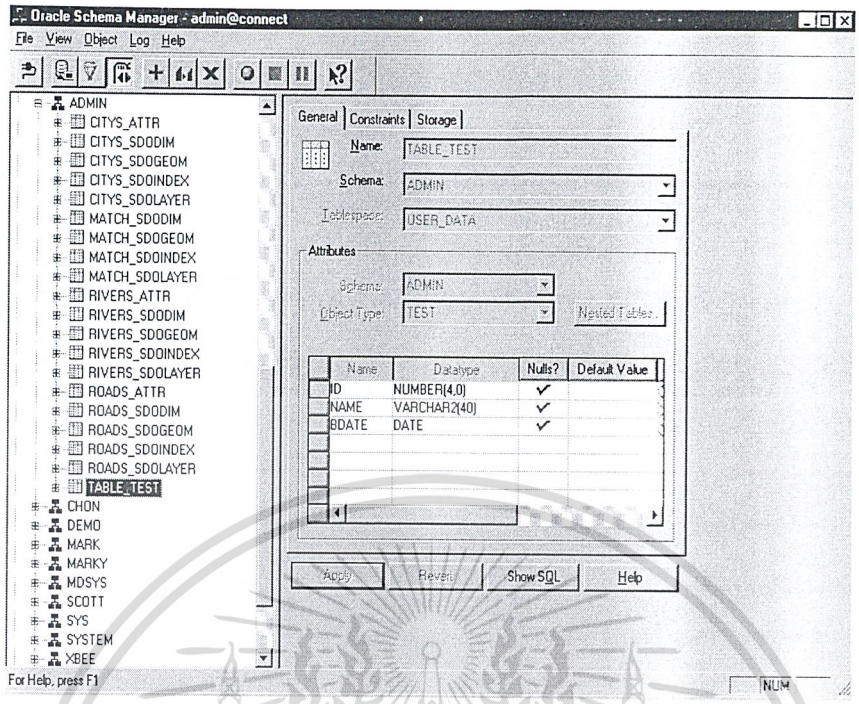
เมื่อเลือก Create Table แบบ Manually จะได้ดังรูปให้เราเลือก Object Table ดังรูปที่ 7-13 ในหน้าจอนี้ให้เราใส่ชื่อตารางแล้วก็ Schema ว่าจะใช้ Schema นี้เป็นของใคร ส่วน schema ที่อยู่ข้างล่างหมายถึงจะสร้างตารางที่มาจาก Schema ของใครเมื่อเราเลือก ว่าจะสร้างตาราง จาก Schema ของใครแล้วเราจะสามารถเลือก Object Type ได้เมื่อเราเลือก Object Type มันจะมี Attributes ขึ้นดังรูปที่ 7-13



รูปที่ 7-13 แสดงถึงชื่อและAttribute ที่มีอยู่ใน Object Type

เมื่อเราสร้างตารางที่เป็น Object Type แล้วจะปรากฏชื่อตาราง ชื่อของผู้สร้างและเราสามารถเข้าไปดูรายละเอียดได้ดังรูปที่ 7-14

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7-14 แสดงถึงตารางที่ได้สร้างขึ้น

ตัวอย่างการเรียกใช้ Object ที่สร้างขึ้น

```
INSERT INTO TABLE_TEST VALUES(1,'EKARAT','6 NOV 1975')
INSERT INTO TABLE_TEST VALUES(2,'JARUWAN','10 DEC 1976')
INSERT INTO TABLE_TEST VALUES(3,'JARUWONG','24 DEC 1977')
```

ผลที่ได้จากการ Query

```
SELECT * FROM TABLE_TEST

ID         NAME                                     BDATE
---------- ---------------------------------------- --------------------
1          EKARAT                                   06-NOV-75
2          JARUWAN                                  10-DEC-76
3          JARUWONG                                 24-DEC-77
2 rows selected.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผลที่ได้จากการ Query โดยเรียกใช้ Method AGE

NAME	BDATE	A.AGE()
EKARAT	06-NOV-75	23.3734357
JARUWAN	10-DEC-76	22.2782712
JARUWONG	24-DEC-77	21.2406028

3 rows selected.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 8
ข้อมูลชนิดออบเจ็กต์ไทพ์ของออราเคิล 8 และวิธีการใช้งาน
(ORACLE OBJECT TYPE AND HOW TO USE IT)

8.1 ความหมายของศัพท์ต่างๆ ที่ควรทราบ

8.1.1 ออบเจ็กต์ไทพ์ (Object Type)

ออบเจ็กต์ไทพ์คือ ชนิดของข้อมูลที่มีความซับซ้อน และ ใกล้เคียงโลกของความเป็นจริงมากขึ้น ซึ่งจะนำข้อมูล (data) และการกระทำ (operation) มาไว้ด้วยกัน ผู้พัฒนาสามารถทำงานได้โดยไม่ต้องรู้รายละเอียดภายในของข้อมูลชนิดออบเจ็กต์นี้เลย และยังสามารถนำข้อมูลชนิดออบเจ็กต์นี้กลับมาใช้ใหม่ได้

8.1.2 ออบเจ็กต์ (Object)

ออบเจ็กต์คือ ชุดของข้อมูลที่มีความสัมพันธ์กัน (ใน ระบบ relational Object คือ แถวในตาราง)

8.1.3 แอตทริบิวท์ (Attributes)

แอตทริบิวท์ ใช้อธิบายส่วนของข้อมูลในออบเจ็กต์นั้น (ใน ระบบ relational Attributes คือ ช่องในตาราง)

8.1.4 เมทธอด (Methods)

เมทธอด คือ การกระทำที่ทำกับข้อมูลซึ่ง จะถูกรวมอยู่ในออบเจ็กต์ ผู้พัฒนาจะเรียกใช้เมทธอดเพื่อเข้าถึงข้อมูล (ในระบบ relational methods คือชุดของ procedure , function ที่มีไว้เพื่อติดต่อทำงานกับข้อมูลในตาราง)

8.1.5 สับคลาส (Subclass)

สับคลาสหรือคลาสย่อย คือคลาสเฉพาะของออบเจ็กต์ไทพ์ ในระบบออบเจ็กต์โอเรียนเต็ดคลาสย่อยจะสามารถถ่ายทอดแอตทริบิวท์และเมทธอดของคลาสพ่อ และเพิ่มแอตทริบิวท์และเมทธอดเฉพาะของคลาสย่อยนั้นลงไปได้ แต่ ใน Oracle8 นั้นไม่สนับสนุนการถ่ายทอดของแอตทริบิวท์และเมทธอด คือไม่สามารถทำการสืบทอดคุณสมบัติต่างๆ ได้

Class คือ attributes , methods

Instance หรือ Object ของ Object Type คือ ข้อมูล

เราสามารถใช้ Object Type ในลักษณะต่อไปนี้

- สนับสนุนชนิดของข้อมูลที่ผู้ใช้กำหนดขึ้นเอง ที่ สามารถสนับสนุนการสร้าง ฐานข้อมูลแบบความสัมพันธ์
- สร้าง nested table ใน ฐานข้อมูลแบบสัมพันธ์ได้
- ใช้สร้าง Object Tables ที่สนับสนุนการออกแบบฐานข้อมูลแบบวัตถุมากกว่าแบบสัมพันธ์

ข้อได้เปรียบของการใช้ Object Type คือ การสร้าง nested table type เช่น สร้าง ITEM_TYPE แล้วเอาไปทำการ nest กับ Order Table ซึ่งจะดีกว่าการ join table ซึ่งการทำ query จะนำเอาข้อมูลออกมาในคราวเดียว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.2 การสร้าง Object Type

ตัวอย่างการสร้าง Object Type ที่ใช้เก็บข้อมูลของที่อยู่ (address)

```
CREATE OR REPLACE TYPE pub.address_type AS OBJECT (
        street1 VARCHAR2(50),
        street2 VARCHAR2(50),
        city VARCHAR2(50),
        state VARCHAR2(25),
        zipcode VARCHAR2(10),
        country VARCHAR2(50) );
```

แสดงการสร้าง Object Type ชื่อ address_type ซึ่งเก็บ ข้อมูล ชื่อถนน ชื่อเมือง ชื่อรัฐ รหัสไปรษณีย์ ชื่อประเทศ

ตัวอย่างแสดงการสร้างตารางความสัมพันธ์ที่นำเอา address_type ด้านบนมาใช้

```
CREATE OR REPLACE TABLE sales.customer (
        id INTEGER PRIMARY KEY,
        last_name VARCHAR2(50),
        first_name VARCHAR2(50),
        company_name VARCHAR2(50),
        address pub.address_type,
        ...
```

ตัวอย่างแสดงคำสั่งที่กระทำกับตาราง sales.customers

```
SELECT
        id,last_name,first_name,address.street1,
        address.street2,address.city,address.state,address.zipcode,
        address.country
FROM sales.customers;
```

ผลจากการทำ query คำสั่งข้างต้นเป็นดังนี้

```
ID LAST_NAME       FIRST_NAME       ADDRESS.STREET1
-- --------------- ---------------- ------------------
 1 Ellison         Lawrence         500 Oracle Parkway ...
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ถ้าต้องการเปลี่ยนแปลงแก้ไขข้อมูลที่มีอยู่ก็สามารถใช้คำสั่ง UPDATE ดังต่อไปนี้ ซึ่งตัวอย่างต่อไปนี้จะทำการเปลี่ยนแปลงค่า address.zipcode ใหม่ให้เป็นค่า '94065' ในแถวของข้อมูลของลูกค้าที่มี id หมายเลข 1

```
UPDATE sales.customers
SET address.zipcode = '94065'
WHERE id=1;
```

ตัวอย่างต่อไปนี้แสดงการเพิ่มข้อมูลเข้าไปในตารางโดยเพิ่มข้อมูลเข้าไปทั้งแถว

```
INSERT INTO sales.customers VALUES(
        1,'Ellison','Lawrence','Oracle Corporation',
        pub.Address_Type(
                '500 Oracle Parkway','Box 659510',
                'Redwood Shores','CA','95045','USA'));
```

8.3 การสร้าง nested table แบบ relational table

ส่วนนี้จะนำเอาตัวอย่างของใบสั่งซื้อมาทำเป็น nested table โดยจะมี ตารางของรายชื่อสินค้าที่สั่งซื้อ และใบสั่งซื้อ

```
CREATE OR REPLEACE TYPE sales.item_type AS OBJECT (
        item_id INTEGER,
        quantity INTEGER );      สร้าง Object ชนิด sales.item_type

CREATE OR REPLACE TYPE sales.item_list AS TABLE OF sales.Item_Type;
```

สร้าง Object Table ชื่อ sales.item_list จาก Object sales.item_type

```
CREATE OR REPLACE sales.orders (
        id INTEGER PRIMARY KEY,
        order_date DATE,
        ship_date DATE,
        line_items sales.item_list )
NESTED TABLE line_item STORE AS items;
```

สร้าง table sales.orders แล้วนำ ตาราง sales.item_list มาทำการ nested อยู่ใน ตาราง sales.orders

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างแสดงการเพิ่มข้อมูลเข้าไปในตาราง sales.orders

```
INSERT INTO sales.orders VALUES (
        1,SYSDATE,NULL,
        sales.item_list(
                sales.item_type(1,22),
                sales.item_type(2,100) ) );
```

จากตัวอย่าง แสดงการเพิ่มข้อมูลการสั่งซื้อ โดยเป็นข้อมูลชุดแรก มีวันสั่งซื้อเป็นวันที่ในระบบ แต่ยังไม่ระบุวันที่ส่งสินค้า มีรายการสินค้าที่สั่ง 2 รายการ คือ สินค้าหมายเลข 1 จำนวน 22 หน่วย และสินค้าหมายเลข 2 จำนวน 100 หน่วย และในการเพิ่มข้อมูลใน line_items จะทำการเรียก constructor sales.item_list

ตัวอย่างการเพิ่มสินค้าที่สั่งซื้อเข้าไปในใบสั่งซื้อที่ 1 โดยในการเพิ่มข้อมูลนี้จะใช้ Expression THE ช่วย ซึ่งจะทำการ select line_items ออกมาก่อน

```
INSERT INTO THE(SELECT line_items FROM sales.orders WHERE id = 1)
VALUES (3,200);
```

ตัวอย่างการเรียกดูข้อมูลจาก nested table

```
SELECT item_id , quantity
FROM THE (SELECT line_items FROM sales.orders WHERE id = 1)
ORDER BY item_id;
```

ITEM_ID	QUANTITY
1	22
2	100
3	200

ตัวอย่างแสดงการยกเลิกรายการสินค้าที่สั่งซื้อรายการที่ 3 (ลบข้อมูล)

```
DELETE THE ( SELECT line_items FROM sales.orders WHERE id = 1) o
WHERE o.items_id = 3;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.4 การสร้าง Object Tables

Object Tables คือ ฐานข้อมูลที่ใช้เก็บข้อมูลชนิด Object เท่านั้น โดยจะไม่มีช่องของข้อมูลชนิดความสัมพันธ์อยู่เลย แต่ละแถวของ Object Table คือ Object ของชนิดของ table นั้นซึ่งแต่ละ object ใน Object Table จะมี Object identifier (OID) เป็นของตัวเองและไม่ซ้ำกัน

ต่อไปเป็นตัวอย่างแสดงการใช้ object type และ object table

```
CREATE OR REPLACE TYPE sales.customer_type AS OBJECT  (
        id INTEGER,
        last_name VARCHAR2(50),
        first_name VARCHAR2(50),
        company_name VARCHAR2(50),
        address pub.address_type );
```

สร้างข้อมูลชนิด Object ชื่อ sales.customer_type ที่เก็บข้อมูล หมายเลขลูกค้า ชื่อสกุล บริษัท และ ที่อยู่

```
CREATE TABLE sales.customers OF sales.customer_type
(id PRIMARY KEY);
```

สร้าง object table ชื่อ sales.customers จากข้อมูลชนิด object ชื่อ sales.customer_type

```
CREATE OR REPLACE TYPE sales.part_type AS OBJECT (
        id INTEGER,
        description VARCHAR2(50),
        unit_price NUMBER(10,2),
        on_hand INTEGER,
        reorder_point INTEGER );
CREATE TABLE sales.patrs OF sales.part_type
(id PRIMARY KEY);
```

สร้าง object table จากข้อมูลชนิด object ที่เก็บข้อมูลของสินค้าแต่ละชนิด

```
CREATE OR REPLACE TYPE sales.item_type AS OBJECT
        item_id INTEGER,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        part REF sales.part_type,
        quantity INTEGER );
CREATE OR REPLACE TYPE sales.item_list AS TABLE OF sales.item_type;
CREATE OR REPLACE TYPE sales.order_type AS OBJECT (
        id INTEGER,
        customer REF sales.customer_type,
        order_date DATE,
        ship_date DATE,
        line_items sales.item_list );
CREATE TABLE sales.orders OF sales.order_type
(id PRIMARY KEY)
NESTED TABLE line_items STORE AS items;
```

สร้าง object table orders

part คือ attribute ที่อ้างอิงถึง object ของข้อมูลชนิด part_type

customer คือ attribute ที่อ้างอิงถึง object ของข้อมูลชนิด customer_type

object reference คือ attribute ของ object หนึ่งที่ชี้ไปที่ object อื่นในฐานข้อมูลที่ใช้ OID

ต่อไปคือตัวอย่างการกระทำกับ object table ที่สร้างขึ้น

ตัวอย่างการเพิ่มข้อมูลของสินค้าและลูกค้า

```
INSERT INTO sales.parts
VALUES (sales.part_type(1,'Pentium 200 CPU',250.00,1000,300));

INSERT INTO sales.customers
VALUES (sales.customer_type(1,'Ellison','Lawrence','Oracle Corporation',
        pub.address_type('500 Oracle Parkway','Box 659510',
        'Redwood Shores','CA','95045','USA')));
```

ตัวอย่างการเพิ่มข้อมูลในใบสั่งซื้อและรายการสั่งซื้อ

```
INSERT INTO sales.orders
        SELECT 1,REF(c),SYSDATE,NULL,sales.item_list()
        FROM sales.customers c
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        WHERE id = 1;
```

ตัวอย่างการเพิ่มข้อมูลในรายการสั่งซื้อ

```
INSERT INTO THE(SELECT o.line_items FROM orders o WHERE o.id =1 )
        SELECT 1,REF(p),20 FROM parts p
        WHERE id = 2

INSERT INTO THE ( SELECT o.line_items FROM orders o WHERE o.id = 1 )
        SELECT 2,REF(p),10 FROM parts p
        WHERE id = 11;
```

ตัวอย่างการใช้ภาษา PL/SQL เพื่อการเพิ่มค่าใน object tables

```
DECLARE
-- กำหนดตัวแปรเพื่อรองรับค่า OID reference
        custoid REF sales.customer_type;
        partoid REF sales.part_type;
BEGIN
-- กำหนดค่าให้กับ custoid
        SELECT REF(c) INTO cutsoid FROM sales.customers c
        WHERE c.last_name = 'Ellison' AND c.first_name = 'Lawrence'

-- กำหนดค่าให้กับ partoid
        SELECT REF(p) INTO partoid FROM sales.parts p
        WHERE p.descroption = 'Pentium 200 CPU';

-- เพิ่มข้อมูลสินค้าที่สั่งซื้อ
        INSERT INTO sales.orders
        VALUES (sales.order_type (1,custoid,SYSDATE,NULL,
                sales.item_list(sales.item_type(1,partoid,50))));
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                        Raise_application_error(-2000,'No data found');
END;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.5 Collection Methodds for Nested Table Available with PL/SQL

Collection Method	**Description**
EXISTS(x)	ส่งกลับค่า true ถ้า element ที่ x อยู่ในตารางและ ส่งกลับค่า false เมื่อไม่อยู่
COUNT	ส่งกลับจำนวนของ element ในตาราง
FIRST	ส่งกลับสมาชิกแรกของตาราง
LAST	ส่งกลับสมาชิกสุดท้ายของตาราง
PRIOR(x)	ส่งกลับสมาชิกลำดับที่ x-1
NEXT(x)	ส่งกลับสมาชิกลำดับที่ x+1
EXTEND(x,y)	Appends x copies of the y^{th} element
TRIM(x)	Trims x elements from the end of a table
DELETE	ลบ element ในตาราง

8.6 METHODS ของ OBJECT

Methods ใน object มีด้วยกัน 2 แบบ คือ

1. Methods ที่ Oracle สร้างขึ้นให้โดยอัตโนมัติที่เรียกว่า Constructor ซึ่งมีชื่อเดียวกันกับชื่อของ object นั้น
2. Methods ที่ ผู้ใช้สร้างขึ้นซึ่งจะต้องเขียนในส่วนของ body เรียกว่า member methods
   - method แต่ละ method สามารถมี parameter ได้ 0-n ตัว
   - จะส่งกลับค่าได้เพียงค่าเดียว
   - แต่ละ method จะต้องมี praga compiler directive

ข้อมูลชนิด object จะไม่สามารถใช้กับการทำ order by ได้ จึงจำเป็นต้องมีการเขียน method ขึ้น คือ map method หรือ order method อย่างใดอย่างหนึ่งเท่านั้น เช่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
CREATE OR REPLACE TYPE pub.address_type AS OBJECT (
        street1 VARCHAR2(50),
        street2 VARCHAR2(50),
        city VARCHAR2(50),
        state VARCHAR2(25),
        zipcode VARCHAR2(10),
        country VARCHAR2(50),
MAP MEMBER FUNCTION address_map RETURN VARCHAR2);
CREATE OR REPLACE TYPE BODY address_type (
MAP MEMBER FUNCTION address_map RETURN VARCHAR2 IS
BEGIN
        RETURN zipcode || city || street1;
END address_map;
);

CREATE OR REPLACE TYPE pub.address_type AS OBJECT (
        street1 VARCHAR2(50),
        street2 VARCHAR2(50),
        city VARCHAR2(50),
        state VARCHAR2(25),
        zipcode VARCHAR2(10),
        country VARCHAR2(50),
ORDER MEMBER FUNCTION address_map (other address_type)
RETURN INTEGER);
CREATA OR REPLACE TYPE BODY PUB.ADDRESS_TYPE (
ORDER MEMBER FUNCTION address_map (other address_type)
RETURN INTEGER IS
        self_address VARCHAR2(150) := self.zipcode || self.city || self.street1;
        other_address VARCHAR2(150) := other.zipcode || other.city ||
                                        other.street1;
BEGIN
        IF self_address < other_address THEN RETURN –1;
        ELSIF self_address > other_address THEN RETURN 1;
        ELSE RETURN 0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        END IF;
    END address_map;
    };
```

8.7 Object And Views

View คือ object ฐานข้อมูลที่อาจสร้างขึ้นเพื่อหลายสาเหตุ ซึ่ง view มีหลายแบบ คือ

8.7.1 Object Views of Object Tables เช่น

```
CREATE OR REPLACE VIEW sales.cust OF sales.customer_type AS
SELECT * FROM sales.customers
```

หรือ

```
CREATE OR REPLACE TYPE sales.customer_order_type AS OBJECT (
        order_count INTEGER,
        company VARCHAR2(50));
CREATE OR REPLACE VIEW sales.customer_orders OF
sales.customer_order_type
WITH OBJECT OID (company) AS
        SELECT COUNT(o.id), o.customer.company_name
        FROM sales.orders o
        GROUP BY o.customer.company_name;
```

8.7.2 Views of Relational Tables

Views That Establish Object Abstractions for Relational Data เช่น

```
CREATE OR REPLACE VIEW cust (id,last,first,company,address) AS
SELECT id,last_name,first_name,company_name,
        Pub.address_type(street1,street2,city,state,zipcode,country)
FROM customers;
```

Object Views of Relational Tebles with OIDs เช่น

```
CREATE OR REPLACE TYPE sales.customer_type AS OBJECT (
        id INTEGER,
        last_name VARCHAR2(50),
        first_name VARCHAR2(50),
        company_name VARCHAR2(50),
        address pub.address_type);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
CREATE OR REPLACE VIEW sales.cust OF sales.customer_type
WITH OBJECT OID (id) AS
        SELECT id,last_name,first_name,company_name,
                Pub.address_type(street1,street2,city,state,zipcode,
                country)
        FROM sales.customers;

CREATE OR REPLACE VIEW sales.parts OF sales.part_type
WITH OBJECT OID (id) AS
        SELECT * FROM sales.parts;

CREATE OR REPLACE VIEW sales.ord OF sales.order_type
WITH OBJECT OID (id) AS
        SELECT id,MAKE_REF(sales.cust,cust_id),order_date,ship_date,
            CAST(MULTISET(
                SELECT id,MAKE_REF(sales.part,id),quantity
                FROM sales.items l
                WHER l.order_id = o.id)
            AS sales.item_list)
        FROM sales.orders o);
```

- CAST คือ การกำหนดค่าของผลของการทำ query กับ nested table
- MAKE_REF คือ การสร้างการอ้างอิงถึง OID เช่น การใช้ MAKE_REF ครั้งแรก คือ การสร้างการอ้างอิง ค่า OID ของ customer ใน cust view

INSTEAD OF trigger คือ trigger ชนิดพิเศษของฐานข้อมูลที่สร้างขึ้นใช้สำหรับ view

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 9
แนวความคิดและการออกแบบโปรแกรม

ในการออกแบบโปรแกรมระบบการสั่งซื้อสินค้านี้ ได้ออกแบบตามมาตรฐาน UML โดยเริ่มจากการทำ Conceptual Model ก่อนเพื่อแสดงให้เห็นถึงส่วนประกอบต่าง ๆ ที่จำเป็นพร้อมทั้งคุณสมบัติต่าง ๆ ที่แต่ละส่วนประกอบนั้นจำเป็นต้องมี รวมถึงความสัมพันธ์ของแต่ละส่วนประกอบที่มีต่อกัน จากนั้นจึงทำการเขียน Colaborlation Diagram ขึ้นซึ่งไดอะแกรมนี้จะแสดงถึงข้อความ(Message)ที่ติดต่อถึงกัน เพราะฉะนั้นจากไดอะแกรมนี้เราจะได้ Medthod ของแต่ละส่วนประกอบที่มีอยู่ ในขั้นสุดท้ายนี้ได้นำ Conceptual Model และ Colaborlation Diagram มาประกอบการพิจารณาเพื่อเขียน Class Diagram

9.1 แนวความคิดของโปรแกรม

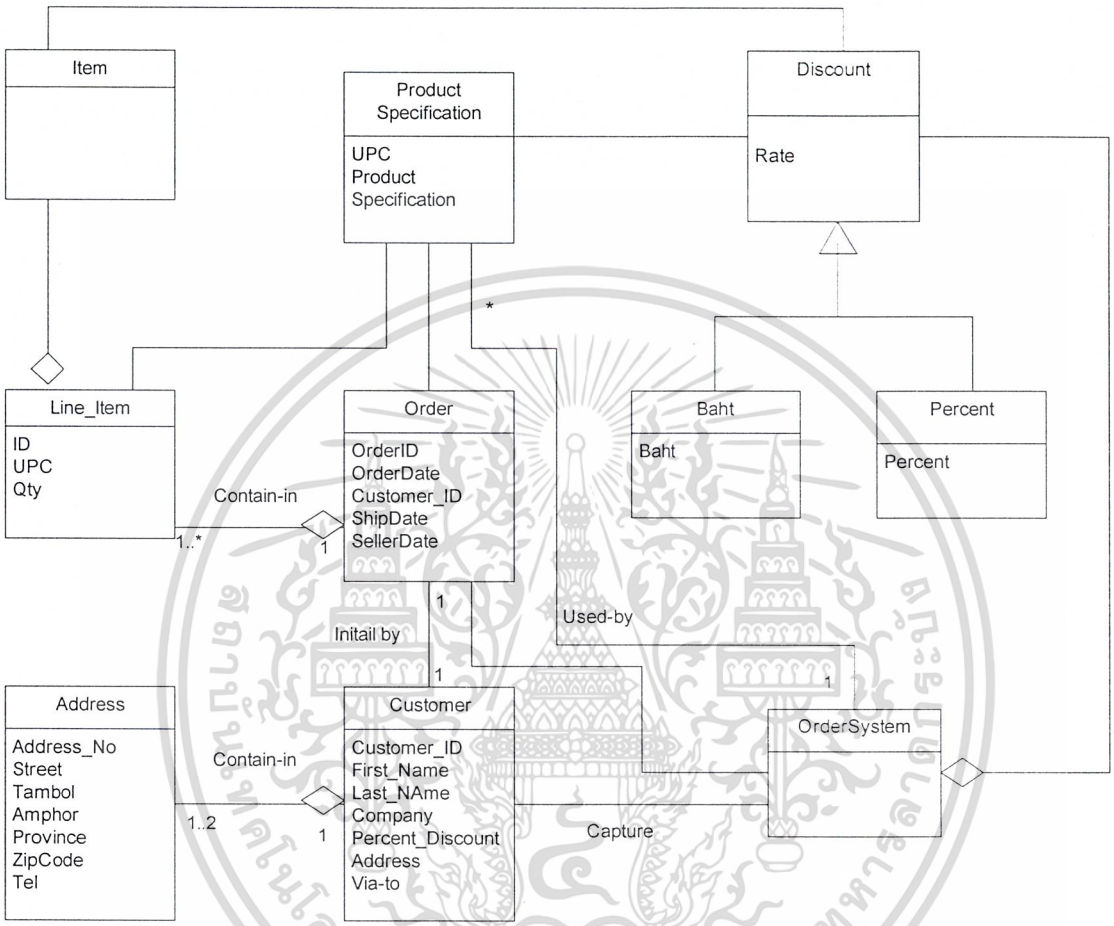
เมื่อเราพิจารณาจาระบบจริงของการสั่งซื้อสินค้านั้น จะมีส่วนประกอบต่าง ๆ ที่มีความซับซ้อนมากมาย ในโครงงานนี้จึงตัดระบบจริงเอามาทำเป็นโปรแกรมประยุกต์เพียงเล็กน้อย คือ

- ระบบสั่งซื้อสินค้าจะต้องสามารถจัดทำใบสั่งซื้อใหม่ได้
- สามารถทำการแก้ไขใบสั่งซื้อที่มีอยู่แล้วได้
- สามารถลบใบสั่งซื้อสินค้าที่มีอยู่ได้
- ใบสั่งซื้อสินค้าใหม่จะต้องมีเลขที่ของใบสั่งซื้อต่อจากใบสั่งซื้อล่าสุดที่มีอยู่ในฐานข้อมูล
- สามารถทำใบสั่งซื้อใหม่ได้โดยการกำหนดหมายเลขรหัสผู้สั่งซื้อ แต่หมายเลขใบสั่งซื้อจะเรียงต่อกันไปโดยไม่สนใจว่าใบสั่งซื้อจะเป็นของใคร
- จะต้องทำการ เพิ่ม/ลบ รายการสั่งซื้อในใบสั่งซื้อสินค้าที่ต้องการได้
- จะต้องสามารถคำนวณส่วนลดให้กับแต่ละรายการสินค้าที่สั่งซื้อได้ โดยส่วนลดของแต่ละรายการจะขึ้นอยู่กับจำนวนที่ซื้อรายการนั้น
- สามารถคำนวณส่วนลดรวมในแต่ละใบสั่ง โดยส่วนลดของแต่ละใบสั่งซื้อจะขึ้นอยู่กับลูกค้าที่เป็นเจ้าของใบสั่งซื้อนั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9.2 Conceptual Model

จากแนวความคิดข้างต้นสามารถเขียนเป็น Conceptual Model ได้ดังต่อไปนี้



รูปที่ 9-1 Conceptual Model

จากรูปจะเห็นได้ว่ามีส่วนประกอบดังต่อไปนี้

- Customer คือ ลูกค้าซึ่งจะมีคุณสมบัติ
  - CustomerID : รหัสลูกค้า
  - FirstName : ชื่อหน้า
  - LastName : นามสกุล
  - Company : บริษัท
  - Address คือ ที่อยู่ซึ่งในที่นี้ใช้เก็บทั้งที่อยู่ของลูกค้าและสถานที่ส่งของ และมีคุณสมบัติดังต่อไปนี้
  - AddressNo : เลขที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Road : ถนน
- Tambol : ตำบล
- Amphor : อำเภอ
- Province : จังหวัด
- ZipCode : รหัสไปรษณีย์
- Tel : หมายเลขโทรศัพท์

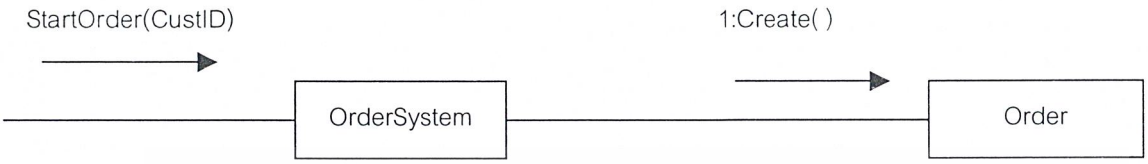
- OrderSystem คือ ระบบการสั่งซื้อสินค้าเป็นส่วนประกอบที่ใช้ควบคุมการทำงานของระบบ ซึ่งส่วนประกอบนี้จะไม่เก็บค่าคุณสมบัติไว้
- Order คือ ใบสั่งซื้อสินค้าแต่ละใบ แต่จากรูปจะเห็นได้ว่า Order กับ OrderSystem จะมีความสัมพันธ์กันแบบ 1 : 1 นั่นคือกัหนดให้เริ่มทำการเริ่มต้นระบบใหม่เมื่อมีการจัดทำหรือแก้ไขใบสั่งซื้อและสินสุดระบบเมื่อจัดทำหรือแก้ไขใบสั่งซื้อตัวนั้นเรียบร้อยแล้ว ซึ่งมีคุณสมบัติดังต่อไปนี้
  - OrderID : หมายเลขใบสั่งซื้อ
  - CustomerID : รหัสลูกค้า
  - OrderDate : วันที่สั่งซื้อ
  - ShipDate : วันที่ส่งสินค้า
  - SalerId : รหัสผู้ขาย
- LineItem คือ รายการสินค้าที่สั่งซื้อ ซึ่งในใบสั่งซื้อ 1 ใบ สามารถมีรายการสินค้าที่สั่งซื้อได้หลายรายการ ดังจะเห็นได้ว่า LineItem กับ Order มีความสัมพันกันแบบ 1 : m ซึ่งจะมีคุณสมบัติดังนี้
  - ID : รายการที่
  - Qty : จำนวนที่สั่งซื้อ
- Item คือ ตัวสินค้า
- ProductSpecification ส่วนประกอบนี้ใช้เก็บข้อมูลรายละเอียดของสินค้าแต่ละชนิด มีคุณสมบัติดังนี้
  - UPC : Universal Product Code รหัสสินค้า
  - Description : คำอธิบาย
  - Price : ราคาต่อหน่วย
- Discount ส่วนลด ซึ่งจะเห็นว่าส่วนลดนี้จะมีอยู่ใน Customer และ LineItem ด้วยคือลูกค้าแต่ละคนจะส่วนลดเป็นของตัวเองหรือไม่มีก็ได้ แต่ส่วนลดในรายการสั่งซื้อจะเกิดขึ้นเมื่อมีการสั่งซื้อถึงจำนวนตามที่กำหนดให้ส่วนลด มีคุณสมบัติ คือ
  - Rate : จำนวนส่วนลด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9.3 Colaborlation Diagram

เป็นการแสดงถึงการส่งข้อความระหว่างส่วนประกอบต่าง ๆ

9.3.1 StartOrder(CustID)

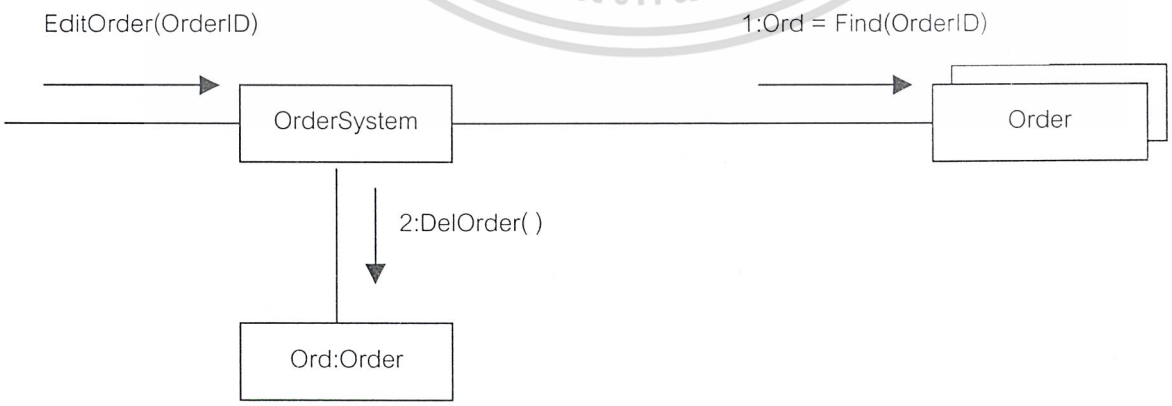


รับรหัสลูกค้ามาทำการเปิดใบสั่งซื้อใหม่โดยหมายเลขใบสั่งซื้อที่เปิดใหม่จะเป็นหมายเลขสุดท้ายเสมอ

9.3.2 EditOrder(OrderID)

รับหมายเลขใบสั่งซื้อที่ต้องการแก้ไขมาทำการค้นหา

9.3.3 EndOrder( )

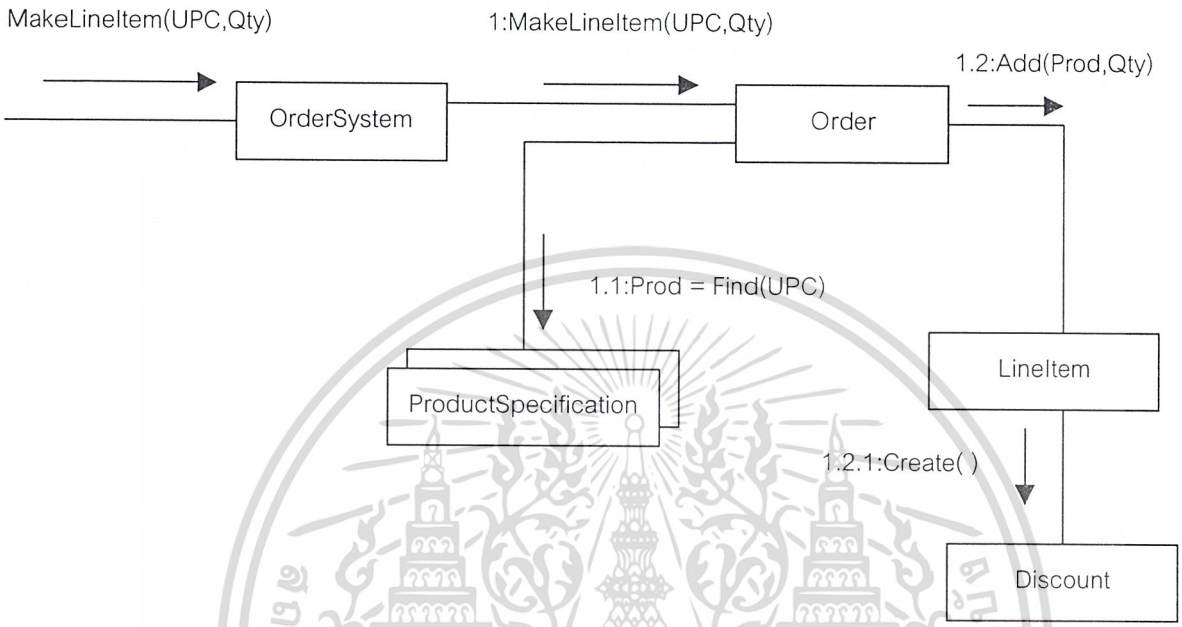


รับหมายเลขใบสั่งซื้อมาทำการค้นหาแล้วทำการลบใบสั่งซื้อนั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

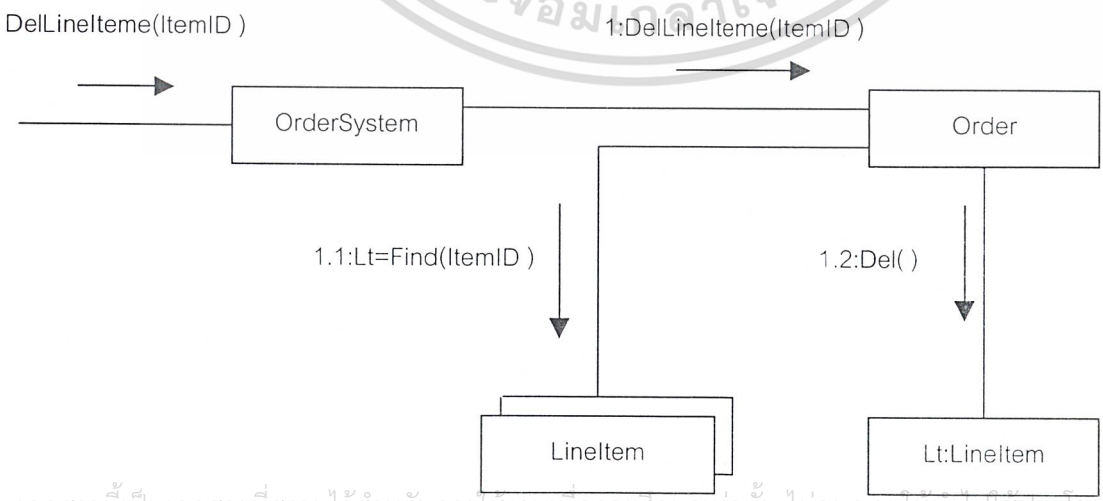
9.3.4 MakeLineItem(UPC,Qty)

รับรายการที่ต้องการลบมาทำการค้นหา ถ้ามีจึงจะทำการลบรายการสั่งซื้อนั้น



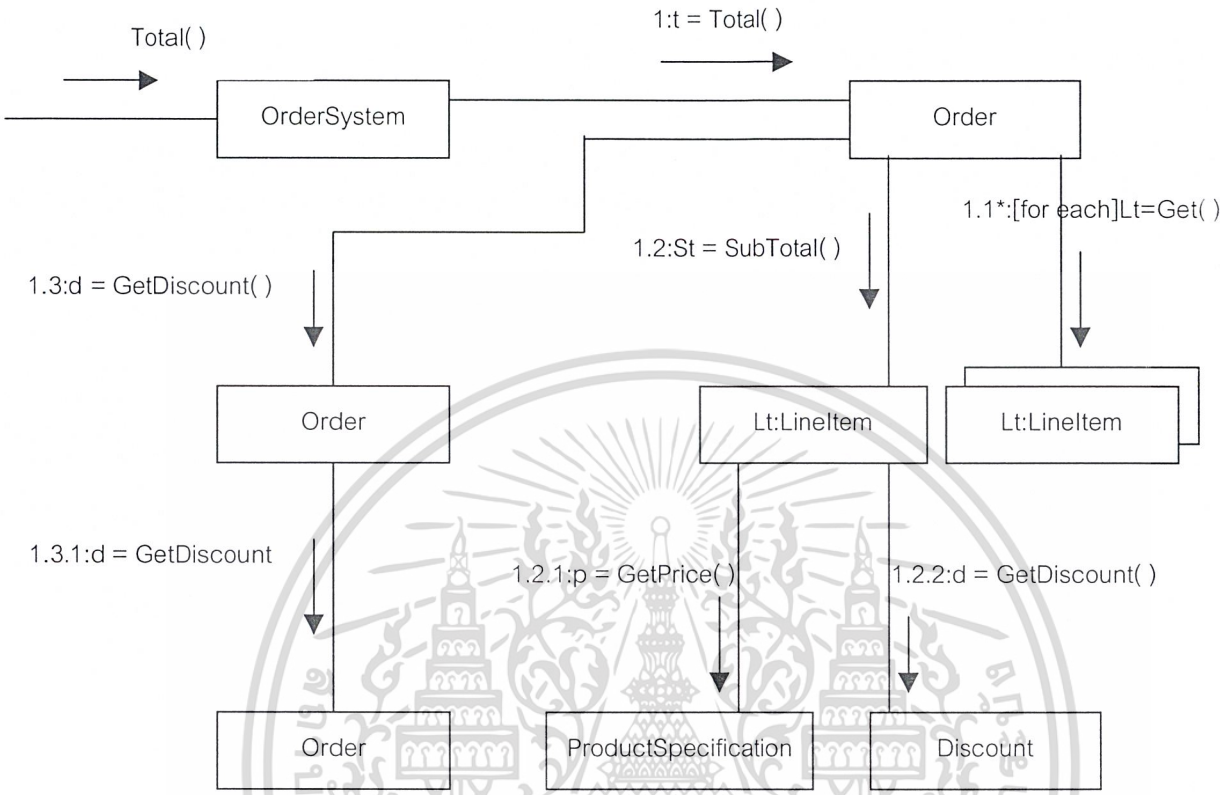
ทำการรับรหัสสินค้า และจำนวนที่ต้องการมาทำการสร้างรายการสั่งซื้อโดยการค้นหาว่ามีสินค้าที่ระบุหรือไม่ ถ้ามีจึงจะทำการสร้างรายการสั่งซื้อแล้วทำการคำนวณหาส่วนลด

9.3.5 DelLineItem(ItemID)



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9.3.6 Total( )

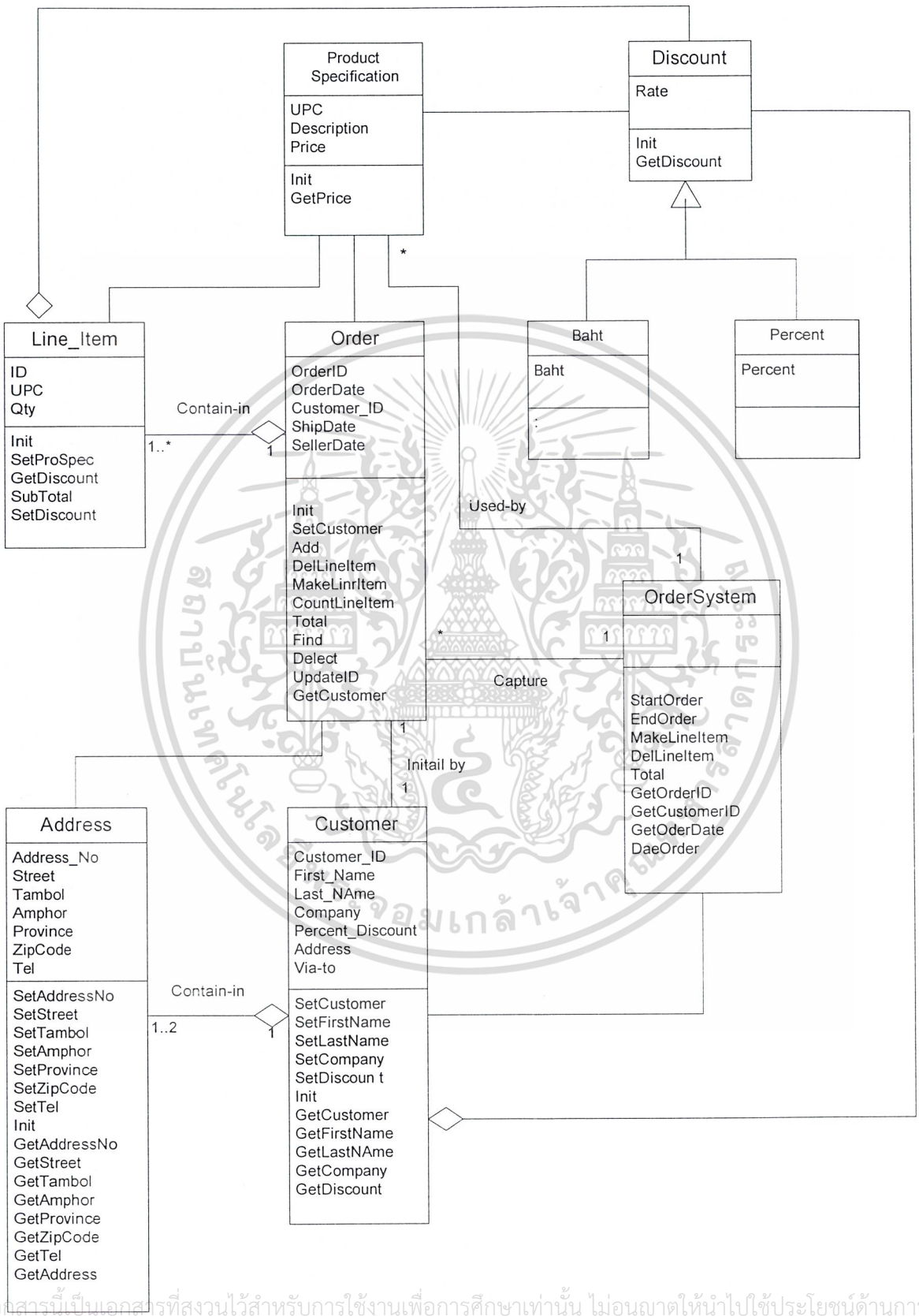


ระบบจะทำการคำนวณหาเงินรวมทั้งหมดโดยจะนำมาหักส่วนลดทั้งส่วนลดต่อรายการ และ ส่วนลดต่อใบสั่งซื้อ

9.4 Class Diagram

จะนำ Conceptual Model และ Colaborlation Diagram มาพิจารณาเพื่อเขียน Class Diagram โดยส่วนประกอบใดที่ไม่มีทั้งคุณสมบัติและ Method ก็ไม่จำเป็นต้องเขียนเป็น Class ดังรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9-2 Class Diagram

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

9.5 การสร้างระบบฐานข้อมูล

ในระบบการขายได้ทำการสร้างตารางที่เป็นตารางชนิดออบเจ็กต์ไทพ์ และในบางตารางอาจมีเนสเต็ดเทเบิ้ลรวมอยู่ด้วย ซึ่งประกอบด้วยตารางดังต่อไปนี้

9.5.1 ตาราง CUSTOMERS

ตาราง CUSTOMERS จะเป็นตารางที่เก็บรายละเอียดต่างๆ ของลูกค้า ประกอบด้วยแอตทริบิวท์ดังต่อไปนี้

Name	Datatype	Null?	Description
Customer_ID	VARCHAR2	No	หมายเลขประจำตัวลูกค้า
First_Name	VARCHAR2	Yes	ชื่อจริง
Last_Name	VARCHAR2	Yes	ชื่อสกุล
Company	VARCHAR2	Yes	ชื่อบริษัท
Percent_Discount	NUMBER	Yes	จำนวนเปอร์เซ็นต์ส่วนลด
Address	ADDRESS	Yes	ที่อยู่
Via_to	ADDRESS	Yes	ที่ส่งสินค้า

ในส่วนของการเขียนโค้ดภาษา SQL จะเริ่มต้นจากการสร้างออบเจ็กต์ไทพ์ก่อน เนื่องจากในตาราง CUSTOMERS มีแอตทริบิวท์ที่เป็นออบเจ็กต์ไทพ์ คือ Address และ Via_to ดังนั้นจึงต้องทำการสร้างออบเจ็กต์ไทพ์ที่ชื่อ ADDRESS ก่อน ดังต่อไปนี้

```
CREATE OR REPLACE TYPE address_type AS OBJECT (
address_ID          VARCHAR2(10),
street              VARCHAR2(30),
tambol              VARCHAR2(30),
amphor              VARCHAR2(30),
province            VARCHAR2(30),
zipcode             VARCHAR2(5),
tel                 VARCHAR2(10));

CREATE OR REPLACE TYPE customer_type AS OBJECT (
customer_ID         VARCHAR2(7),
first_name          VARCHAR2(30),
last_name           VARCHAR2(30),
company             VARCHAR2(30),
percent_discount    NUMBER,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    address         ADDRESS,
    via_to          ADDRESS);

CREATE TABLE customers OF customer_type;
```

การ INSERT ข้อมูลลงในตาราง

```
INSERT INTO customers VALUES (
 'AC4/413' , null , null , 'คาร์ฟูร์ บางใหญ่' , 5 ,
address_type( '300' , 'กรุงเทพฯ-สุพรรณบุรี' , 'บางใหญ่' , 'บางใหญ่' , 'นนทบุรี' , '11140' , '9034204' ),
address_type( '300' , 'กรุงเทพฯ-สุพรรณบุรี' , 'บางใหญ่' , 'บางใหญ่' , 'นนทบุรี' , '11140' , '9034204' ));
```

9.5.2 ตาราง PRODUCT_SPECIFICATIONS

ตาราง PRODUCT_SPECIFICATIONS จะเก็บรายละเอียดของสินค้าประเภทต่างๆ ประกอบด้วยแอตทริบิวท์ดังต่อไปนี้

Name	Datatype	Null?	Description
UPC	VARCHAR2(7)	No	รหัสสินค้า
Description	VARCHAR2(50)	Yes	รายละเอียดสินค้า
Price	NUMBER	Yes	ราคาสินค้า
Discount_list	Discount_lists	Yes	รายการส่วนลด

ในตารางนี้จะมี Nested table อยู่ด้วยคือ ตาราง Discount_lists ดังนั้นการสร้างตารางต้องเริ่มสร้างจาก Nested table ก่อน ดังต่อไปนี้

```
CREATE OR REPLACE TYPE discount_type AS OBJECT (
    baht            NUMBER,
    percent         NUMBER);

CREATE OR REPLACE TYPE discount_list_type AS OBJECT (
    step            NUMBER,
    discount        Discount_type);
CREATE OR REPLACE TYPE discount_lists AS TABLE OF discount_list_type;
CREATE OR REPLACE TYPE product_specification_type AS OBJECT (
    upc             VARCHAR2(7),
    description     VARCHAR2(7),
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    price                NUMBER,
    discount_list        discount_lists);

CREATE TABLE product_specification OF product_specification_type
    NESTED TABLE promotion_level STORE AS pm_levels;
```

การ INSERT ข้อมูลลงในตาราง

```
INSERT INTO product_specifications VALUES (
    'PIS0001' , 'Pias Super Powder B3' , 120 , discount_lists(discount_list_type(100,
    discount_type(0,5))));
```

9.5.3 ตาราง ORDERS

ตาราง ORDERS จะเก็บรายละเอียดต่างๆ ของการสั่งซื้อ ประกอบด้วยแอตทริบิวท์ดังต่อไปนี้

Name	Datatype	Null?	Description
Order_ID	NUMBER	No	หมายเลขลำดับการสั่งซื้อสินค้า
Customer_ID	VARCHAR2(7)	Yes	หมายเลขประจำตัวลูกค้า
Order_date	VARCHAR2(10)	Yes	วันที่สั่งซื้อสินค้า
Ship_date	VARCHAR2(10)	Yes	วันที่ทำการตัด stock สินค้า
Seller_ID	VARCHAR2(10)	Yes	หมายเลขประจำตัวผู้ขาย
Line_item	LINE_ITEMS	Yes	Nested table ชื่อ LINE_ITEMS

ในตารางนี้จะมี Nested table อยู่ด้วยคือ ตาราง Line_items ดังนั้นการสร้างตารางต้องเริ่มสร้างจาก Nested table ก่อน ดังต่อไปนี้

```
CREATE OR REPLACE TYPE line_item_type AS OBJECT (
    id      NUMBER ,
    upc     VARCHAR2(7) ,
    qty     NUMBER) ;

CREATE OR REPLACE TYPE line_items AS TABLE OF line_item_type ;

CREATE OR REPLACE TYPE order_type AS OBJECT (
    order_id NUMBER,
    customer_id     VARCHAR2(7),
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    order_date          VARCHAR2(10),
    ship_dateVARCHAR2(10),
    seller_id           VARCHAR2(7),
    line_itemline_items;

CREATE TABLE orders OF order_type
    NESTED TABLE line_item STORE AS items;
```

การ INSERT ข้อมูลลงในตาราง

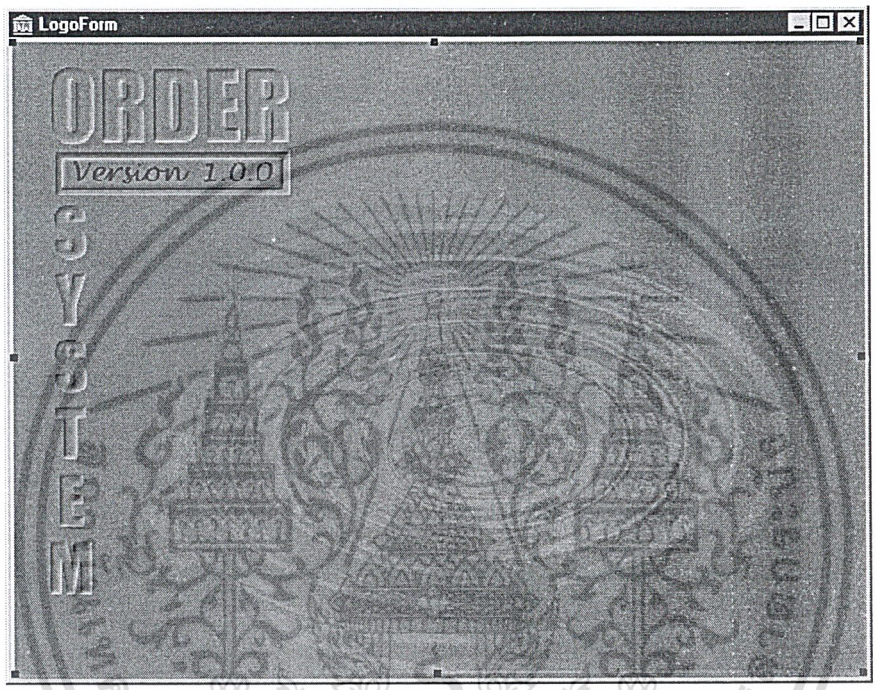
```
INSERT INTO orders VALUES (
    1 , AA0/001 , '12/04/2542' , '19/04/2542' , 'SA1/123' ,
    line_items(line_item_type( 1 , 'PIS0001' , 500 ));
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 10
ผลการดำเนินงาน

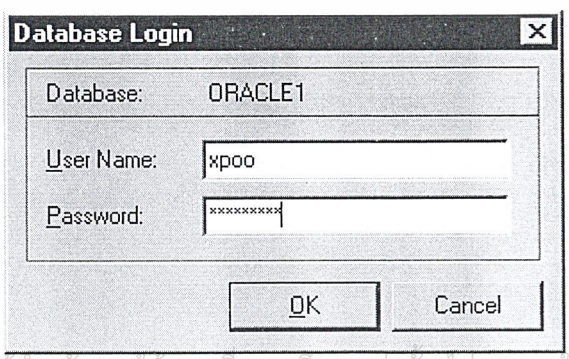
10.1 วิธีการใช้งานและการตรวจสอบโปรแกรม

เมื่อทำการแยกเครื่องเป็นแบบสามระดับและนำไฟล์ .exe ไปรันที่เครื่องที่เป็นไคลเอินท์แล้วก็ทำการรันโปรแกรม ซึ่งหลังจากรันโปรแกรมแล้วในส่วนไคลเอินท์และเซิร์ฟเวอร์จะแสดงผลดังต่อไปนี้



รูปที่ 10-1 หน้าจอแสดงผลหน้าจอแรกของระบบการสั่งซื้อที่แสดงในส่วนของเครื่องไคลเอินท์

ส่วนเครื่องที่เป็นเซิร์ฟเวอร์จะให้ผู้ใช้ทำการ Login เข้าไปยังระบบการจัดการฐานข้อมูลออราเคิลก่อน จากนั้นก็จะแสดงหน้าจอว่าสามารถทำการรันเซิร์ฟเวอร์ได้เรียบร้อยแล้ว ดังต่อไปนี้

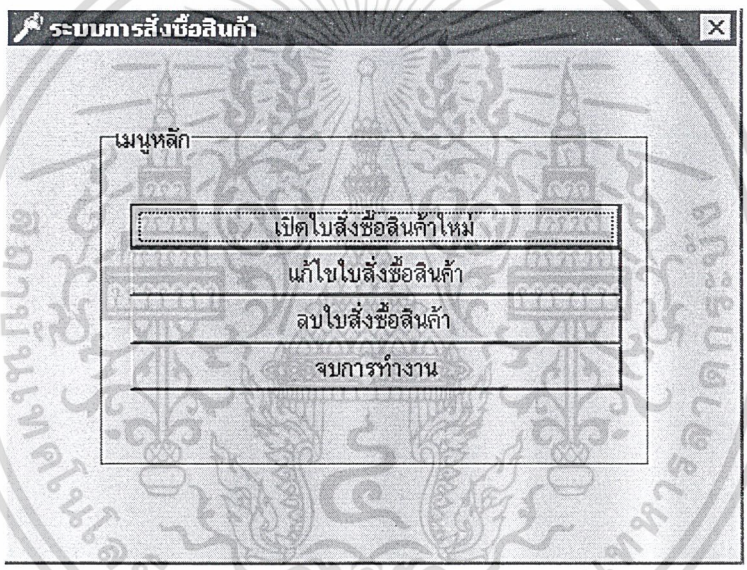


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเอกสารนี้ทุกครั้งที่มีการนำไปใช้

รูปที่ 10-2 แสดงหน้าจอการ Login เข้าระบบการจัดการฐานข้อมูลออราเคิล 8

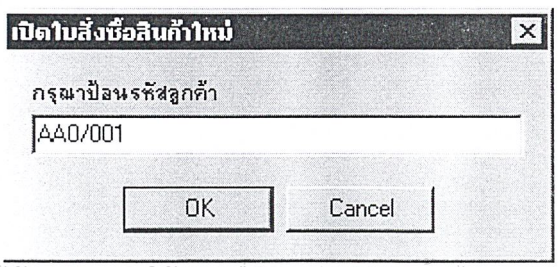
รูปที่ 10-3 แสดงหน้าจอว่าสามารถทำการรันเซิร์ฟเวอร์ได้แล้ว

เมื่อสามารทำการรันส่วนเซิร์ฟเวอร์เป็นที่เรียบร้อยแล้ว จากนั้นทำการคลิ้กที่หน้าจอแรกของไคลเอินท์ คือรูปที่ 10-1 โปรแกรมจะทำการแสดงเมนูออกมา ดังรูป



รูปที่ 10-4 เมนูของระบบ

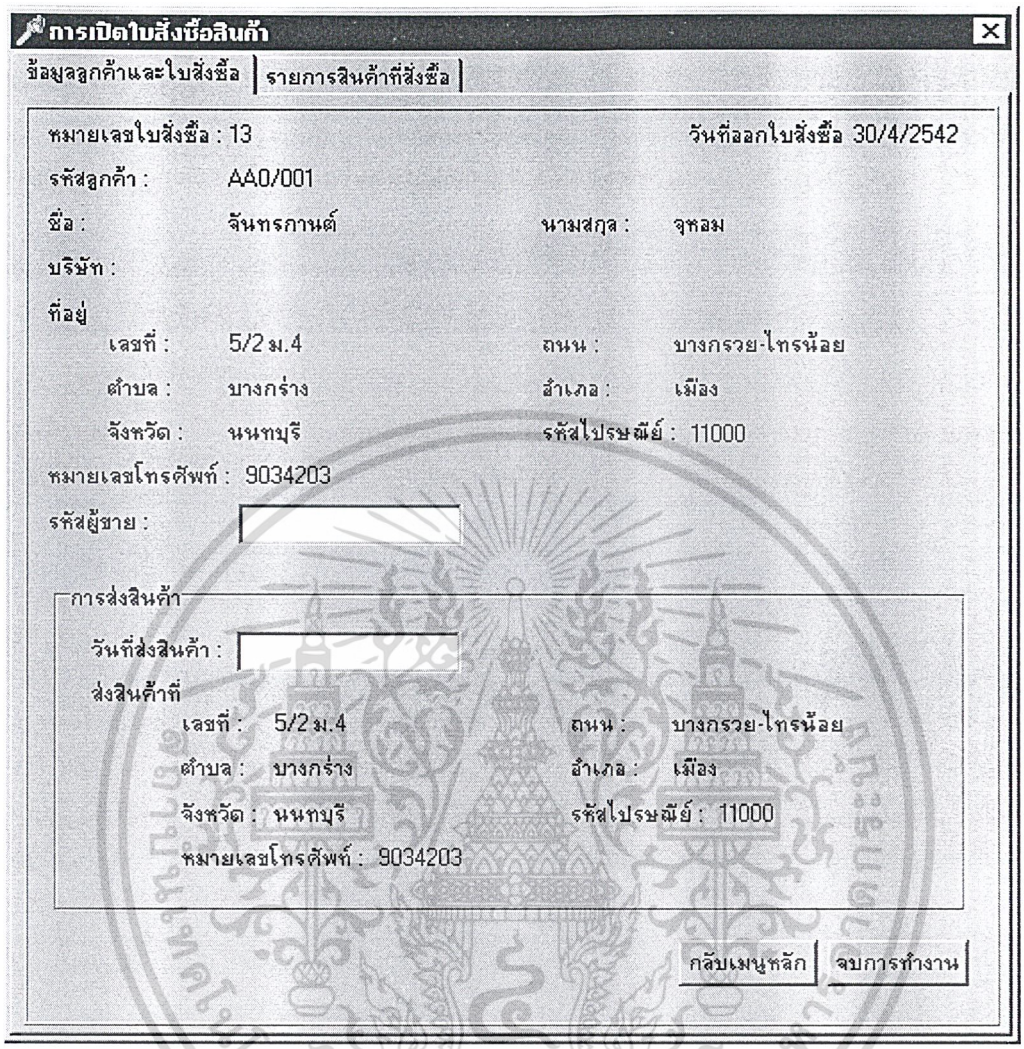
ต่อไปทำการเลือกขั้นตอนจากเมนูตามที่ต้องการให้โปรแกรมทำงาน ซึ่งในลำดับนี้จะแสดงถึงการเปิดใบสั่งซื้อใหม่ก่อน เมื่อคลิ้กที่เมนูเปิดใบสั่งซื้อใหม่โปรแกรมจะทำการแสดงผลดังต่อไปนี้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 10-5 โปรแกรมจะให้ทำการป้อนรหัสลูกค้า

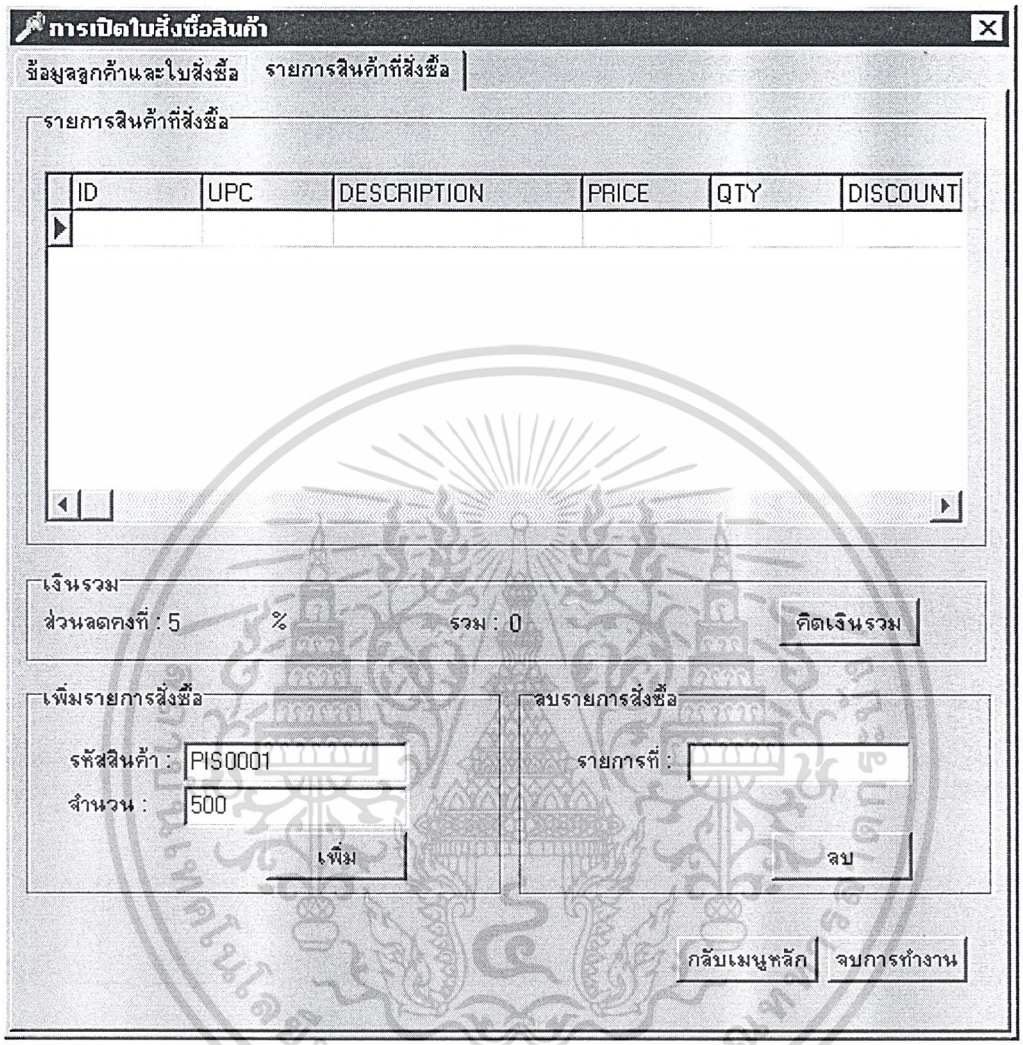
รูปที่ 10-6 จะเป็นการแสดงข้อมูลลูกค้า ในใบสั่งซื้อ



รูปที่ 10-6 แสดงข้อมูลของลูกค้า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

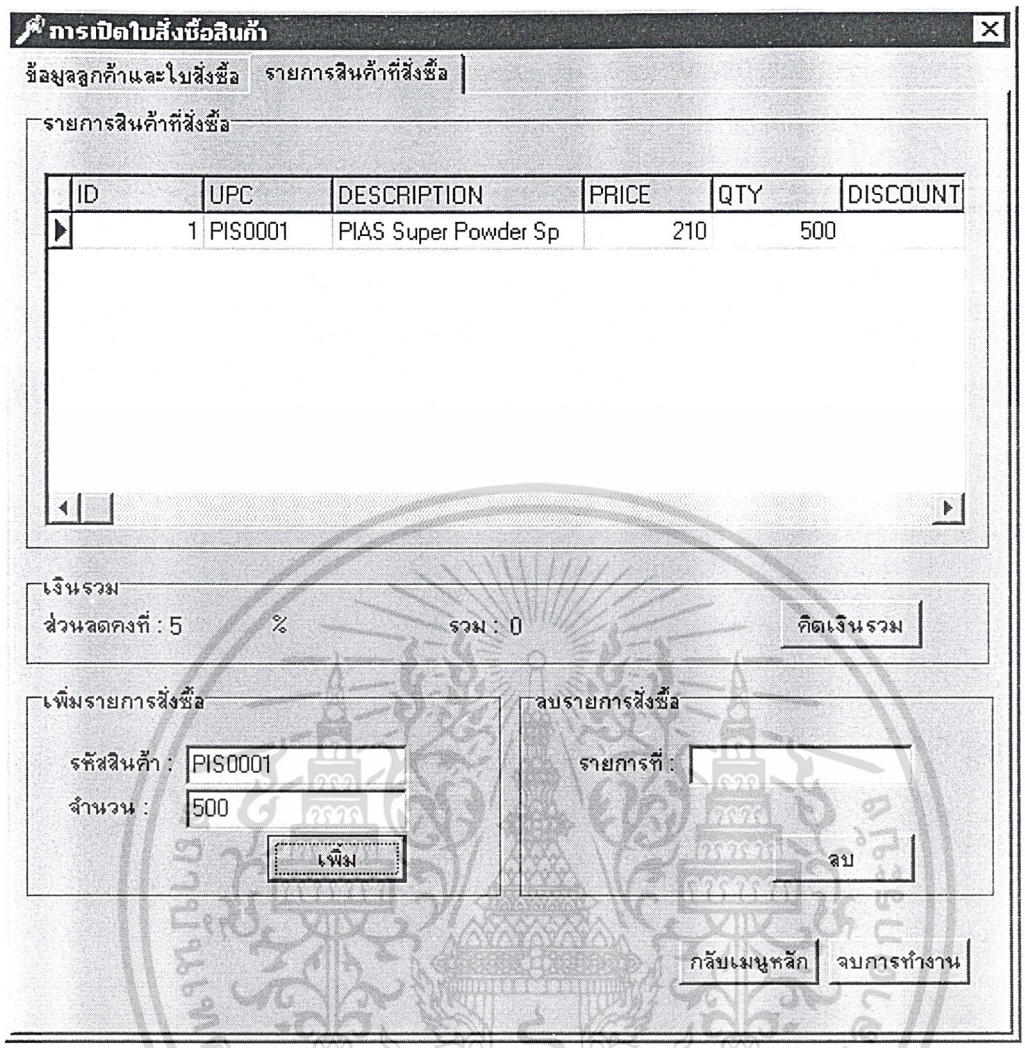
เมื่อป้อนรหัสลูกค้าแล้วโปรแกรมจะแสดงหน้าจอการสั่งซื้อดังนี้ โดยจากตัวอย่างจะแสดงการป้อนรหัสสินค้าคือ PIS0001 จำนวน 500 ชิ้น



รูปที่ 10-7 แสดงการป้อนรหัสสินค้าและจำนวนที่ต้องการสั่งซื้อ

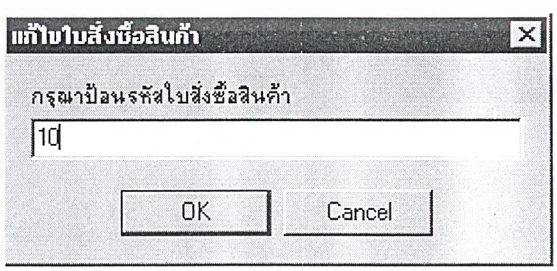
เมื่อทำการป้อนชื่อสินค้าและจำนวนแล้วให้กดปุ่มเพิ่ม โปรแกรมจะทำการเพิ่มรายชื่อสินค้าเข้าไปดังรูปที่ 10-8

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 10-8 แสดงรายการสินค้าที่ได้เพิ่มแล้ว

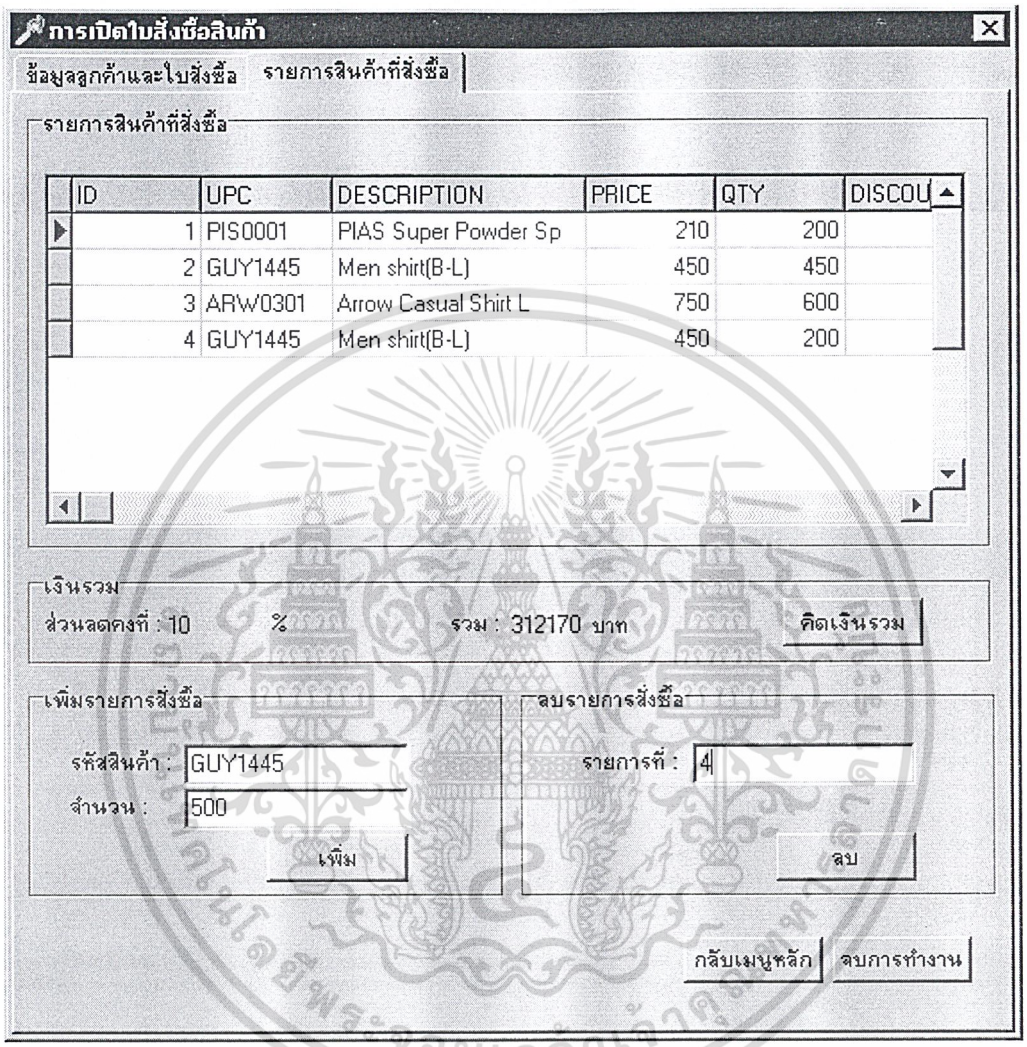
เมื่อต้องการทำการลบหรือทำการแก้ไขใบสั่งซื้อ ให้ทำการเลือกที่เมนู ซึ่งระบบจะให้ป้อนหมายเลขสั่งซื้อก่อนดังรูปที่ 10-9



รูปที่ 10-9 การป้อนรหัสใบสั่งซื้อเพื่อทำการแก้ไขใบสั่งซื้อ

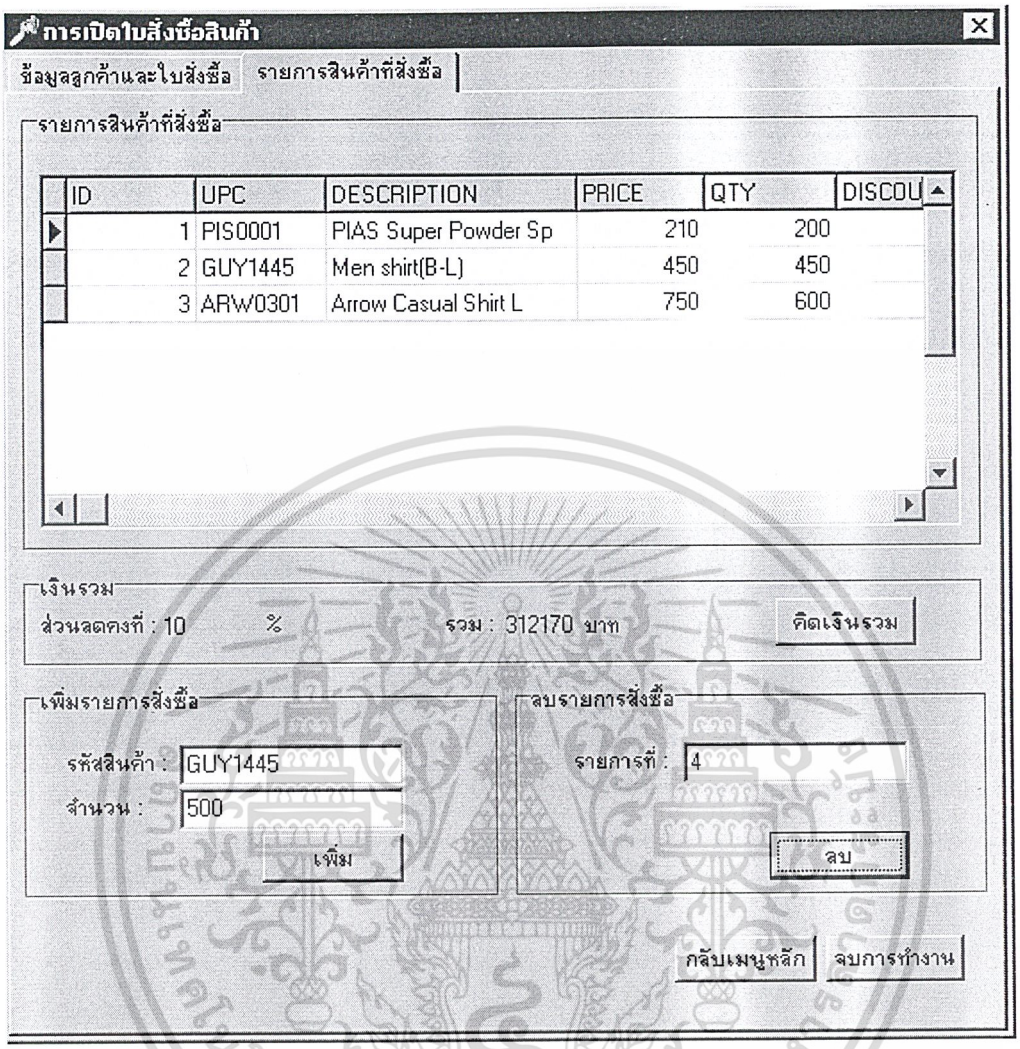
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากนั้นก็ทำการลบ เช่นรูปที่ 10-10 จะประกอบด้วยรายการการสั่งซื้อทั้งหมดสี่รายการ หลังจากที่ทำการลบรายการที่สี่ออกแล้วก็จะเหลืออยู่สามรายการดังรูปที่ 10-11 ซึ่งการแก้ไขก็ทำตามขั้นตอนคล้ายๆ กัน



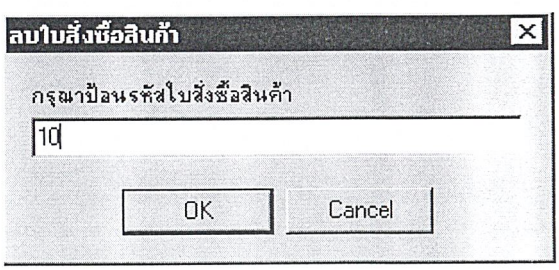
รูปที่ 10-10 แสดงรายการการสั่งซื้อก่อนที่จะทำการลบรายการการสั่งซื้อที่สาม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



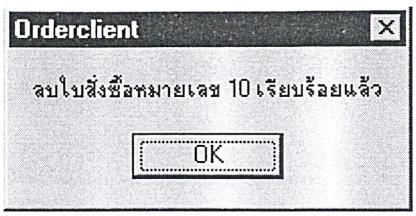
รูปที่ 10-11 แสดงรายการสินค้าหลังจากที่ทำการลบรายการที่สี่ออก

เมื่อต้องการลบใบสั่งซื้อสินค้าให้เลือเมนูลบใบสั่งซื้อจากนั้นป้อนหมายเลขใบสั่งซื้อสินค้า แสดงดังรูปที่ 10-12 เมื่อระบบทำการลบเรียบร้อยแล้วจะแสดงดังรูปที่ 10-13



รูปที่ 10-12 การป้อนรหัสใบสั่งสินค้าเพื่อทำการลบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 10-13 หน้าจอแสดงผลเมื่อระบบทำการลบใบสั่งซื้อสินค้านั้นแล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 11
บทวิจารณ์และสรุป

11.1 สรุปผลการดำเนินงาน

การทดลองวิเคราะห์ , ออกแบบและเขียนโปรแกรมในเชิงวัตถุนั้นทำให้ทราบข้อดีและข้อได้เปรียบจากการพัฒนาโปรแกรมในแบบโครงสร้าง ดังต่อไปนี้

- มีการทำอินฟอร์เมชั่นไฮดิงค์ (Information hiding) เนื่องจาก OOP จะเก็บข้อมูลไว้ในออบเจ็กต์ซึ่งจะถูกจัดการโดยเมทธอดที่เหมาะสมเท่านั้น ดังนั้นจึงทำให้ข้อมูลมีความปลอดภัยมากกว่าแบบ procedural
- สามารถนำคอมโพเนนท์ที่สร้างไว้แล้วกลับมาใช้งานได้อีก เช่น อนุญาตให้โปรแกรมที่ถูกสร้างขึ้นมาใหม่สามารถเรียกใช้เมทธอดจากโปรแกรมเดิมได้
- ไม่มีปัญหาในการตั้งชื่อเมทธอด ซึ่งสามารถตั้งชื่อซ้ำกันได้เพราะออบเจ็กต์สามารถเลือกตอบสนองเมสเสจโดยใช้เมทธอดที่เหมาะสม
- มีคุณสมบัติในการเอ็นแคปซูเลชั่นจึงทำให้เกิดความผิดพลาดได้ยากและความน่าเชื่อถือสูง
- มีคุณสมบัติในการสืบทอดความสัมพันธ์จึงทำให้มีความยืดหยุ่นมาก คือ สามารถเพิ่มความสามารถให้แต่ละคลาสย่อยได้ โดยอาจเพิ่มที่คลาสต้นตระกูลเท่านั้นทุกๆ คลาสย่อยก็จะถูกเพิ่มความสามารถนั้นๆ ไปด้วย
- มีลักษณะของโปรแกรมตรงกับความเข้าใจของผู้ใช้ ซึ่งทำให้ผู้ใช้สามารถเข้าใจการทำงานได้โดยง่าย
- ง่ายต่อการเพิ่มเติมเมื่อมีความต้องการเพิ่มเติมเมื่อมีความต้องการต่างๆ เพิ่มขึ้น ง่ายต่อการแก้ไขและบำรุงรักษาโปรแกรม
- ในการพัฒนาโปรแกรมไม่จำเป็นต้องสนใจโค้ดของโปรแกรมว่าเป็นอย่างไร แค่ทราบว่าแค่ส่งเมสเสจนั้นไปในออบเจ็กต์แล้วได้เอาท์พุทออกมาอย่างไรจึงทำให้สามารถพัมนาโปรแกรมแบบขนานได้
- สามารถตรวจสอบหาข้อผิดพลาดได้ง่าย

และจากการที่ได้ทำการสร้างระบบฐานข้อมูลแบบมัลติเทียร์ไคล์เอนท์เซิร์ฟเวอร์ ทำให้ได้ทราบถึงคุณประโยชน์ ดังต่อไปนี้

- ง่ายต่อการบำรุงรักษาเมื่อมีการเปลี่ยนแปลงกฎของการทำงานในแต่ละไคลน์แอปพลิเคชั่นแต่ละตัว
- สามารถทำการสร้างไคลน์แอปพลิเคชั่นที่มีขนาดเล็กได้โดยที่การทำงานที่มีกับแอปพลิเคชั่นเซิร์ฟเวอร์มีจำนวนมากได้ และการเพิ่มจำนวนไคลน์แอปพลิเคชั่นยังทำได้โดยง่าย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- การกระจายงานของแอปพลิเคชั่นแต่ละเครื่องสามารถเพิ่มประสิทธิภาพของระบบให้ดีขึ้นได้ เนื่องจากมีการทำ load balancing และยอมให้ระบบอื่นรับทำงานหนึ่งต่อไปเมื่อเซิร์ฟเวอร์ไม่สามารถทำงานได้
- สามารถจำกัดสิทธิ์ในการเข้าติดต่อเครื่องที่อาจเกิดความเสียหายได้ง่ายหรืออาจยอมให้การติดต่อเป็นไปได้โดยง่าย

11.2 แนวทางในการพัฒนาต่อ

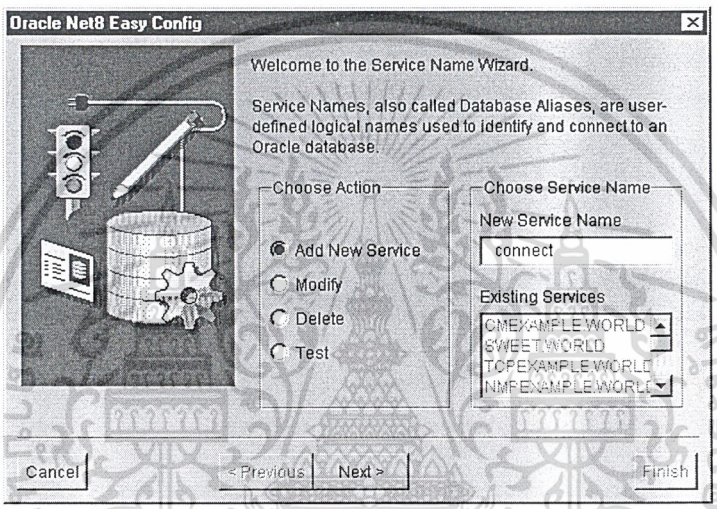
จากที่ได้ทำการศึกษา และทำการทดสอบนับว่าการพัฒนาโปรแกรมในเชิงวัตถุนั้นเป็นสิ่งใหม่ที่จะได้รับการพัฒนาต่อไปอย่างสูงสุด เนื่องจากคุณสมบัติต่างๆ ที่มีความสามารถเหนือกว่าการพัฒนาในแบบโครงสร้าง เพื่อการเข้าใจที่ชัดเจนยิ่งขึ้นว่าระบบที่ได้รับการพัฒนาในเชิงวัตถุนั้นมีขีดความสามารถสูงกว่าการพัฒนาในแบบโครงสร้าง ทางผู้จัดทำมีความเห็นว่าน่าจะลองสร้างระบบที่มีขนาดใหญ่ และมีความซับซ้อนมากกว่าระบบที่ผู้จัดทำได้ทำการสร้างขึ้นมา และเพื่อให้เป็นไปตามแนวความคิดในการพัฒนาโปรแกรมในเชิงวัตถุอย่างแท้จริง ควรจะนำระบบการจัดการฐานข้อมูลซึ่งมีขีดความสามารถในการรองรับคุณสมบัติต่างๆ ของออบเจ็กต์โอเรียนเต็ดอย่างแท้จริงและเต็มร้อยเปอร์เซ็นต์ และควรทำระบบใหญ่ๆ และมีการทดลองรันพร้อมกันหลายๆ เครื่อง เพื่อศึกษาถึงประสิทธิภาพของมัลติเทียร์ไคล์เอนท์เซิร์ฟเวอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก
วิธีการสร้าง Service ติดต่อเครื่อง Client กับเครื่อง Server

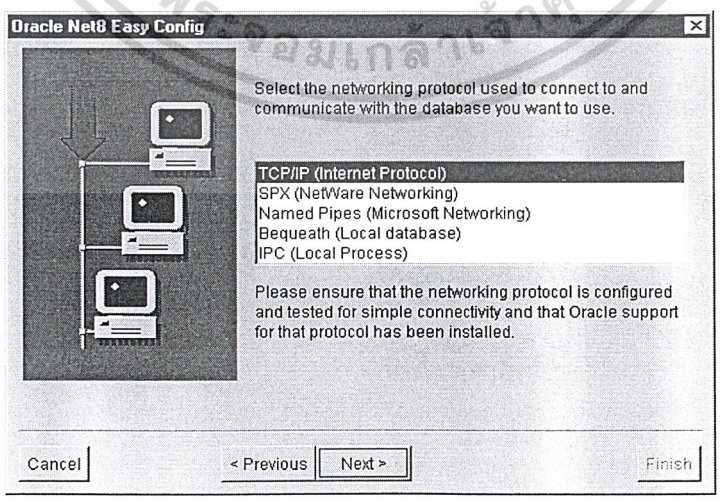
วิธีการติดต่อเครื่องลูกข่าย (Orcle Client)กับเครื่องเซิร์ฟเวอร์ของ Oracle (Oracle Server)

ในการติดต่อระหว่างเครื่องลูกข่ายกับเซิร์ฟเวอร์เราจะต้องสร้างบริการ (Service) ในการติดต่อ Tool ที่ใช้สร้างในการบริการคือ Oracle Net8 Easy Config ดังรูป ในหน้าจอนี้จะประกอบด้วย การสร้างบริการ การปรับปรุง ลบและทดสอบ บริการ ในที่นี้เราจะสร้างบริการก็ใส่ชื่อบริการ ในที่นี้ใส่ชื่อว่า connect หลังจากนั้นก็ กดปุ่ม Next จะได้หน้าถัดไป



รูปที่ ก-1

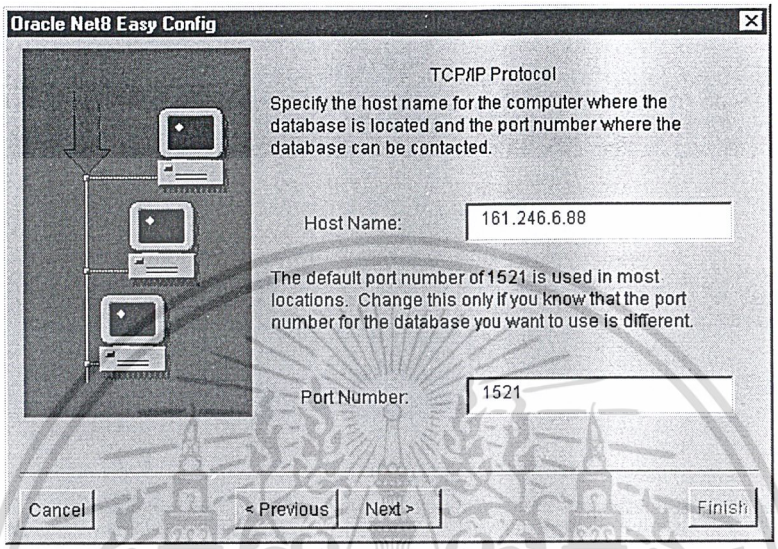
หน้าจอต่อไปนี้เป็นการเลือก Protocol ในการติดต่อระหว่าง เครื่องลูกข่ายกับเครื่องเซิร์ฟเวอร์ ในที่นี้ เลือก Protocol TCP/IP ในการติดต่อดังรูปที่ ก-2 แล้วก็กดปุ่ม Next ไปหน้าถัดไป



รูปที่ ก-2

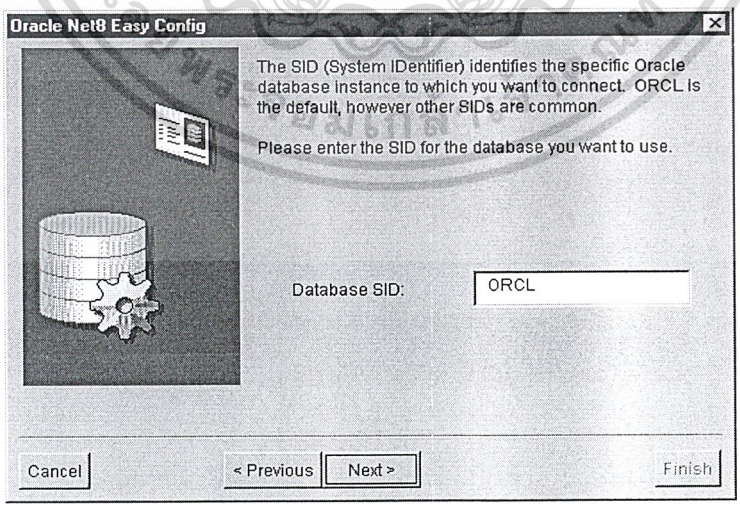
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน้าจอนี้เป็นหน้าจอต่อจากการที่เราเลือกการติดต่อโดยใช้ Protocol TCP/IP ซึ่งหน้าจอนี้จะถามหา Host Name หรือ IP Address และ Port Number ซึ่ง Host ที่ใช้อยู่มี IP 161.246.6.88 หรือเราสามารถใส่ชื่อ Host ลงไปได้ เช่น database01.ce.kmitl.ac.th และ port 1521 ซึ่งเป็นค่า Default อยู่แล้ว แล้วก็กดปุ่ม Nextเพื่อไปหน้าจอถัดไป



รูปที่ ก-3

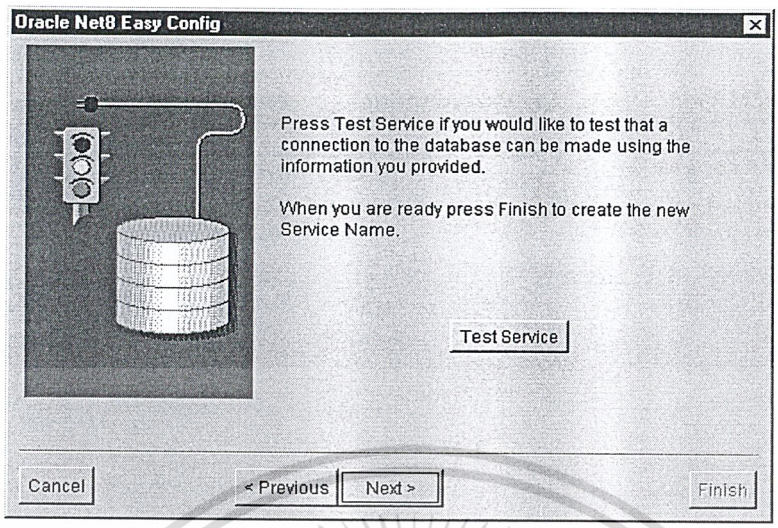
หน้าจอนี้เป็นการกำหนด System Identifier ซึ่ง ORCL เป็นค่า Default ของ Database ที่ระบบสร้างเป็นให้อยู่แล้ว ถ้าเราสร้าง database ตัวใหม่เราก็กำหนด System Identifier ตามที่ชื่อที่เราสร้าง แล้วก็กดปุ่ม Next เพื่อไปหน้าจอถัดไป



รูปที่ ก-4

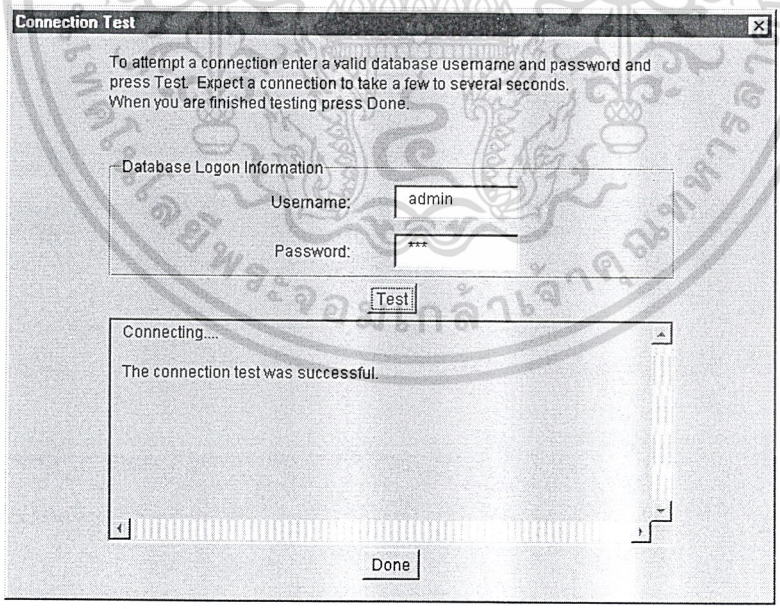
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากเราได้ใส่ SID ก็จะเข้าสู่การ Test หน้าจอนี้เป็นหน้าจอที่ใช้ทดสอบบริการที่เราสร้างขึ้น



รูปที่ ก-5

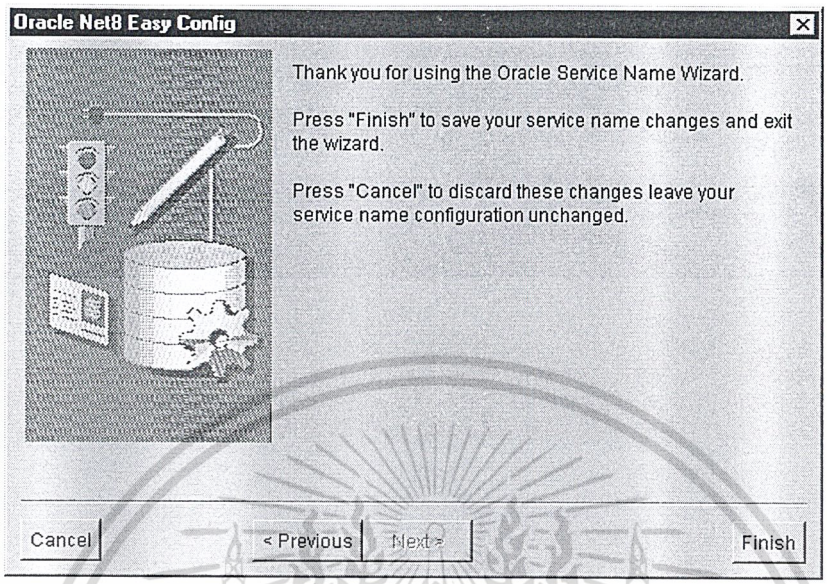
เมื่อเรากด Next จากหน้าจอ Test Service เราก็เข้าสู่การทดสอบโดยให้ใส่ชื่อ User Name และ Password แล้วก็ทำการทดสอบ โดยกดปุ่ม Test และเมื่อระบบติดต่อสำเร็จ ก็จะขึ้นคำว่า The connection test was successful ก็แสดงว่าบริการที่เราสร้างขึ้นใช้ได้แล้ว ดังรูป



รูปที่ ก-6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อเราทดสอบบริการผ่านแล้ว ก็จะได้หน้าจอนี้ซึ่งเมื่อกดปุ่ม Finish ก็จะทำการ Save บริการให้เรา ต่อไปถ้าเราจะติดต่อเครื่องลูกข่ายกับเครื่องเซิร์ฟเวอร์เราก็เรียกบริการผ่านทางบริการที่เราสร้างขึ้น



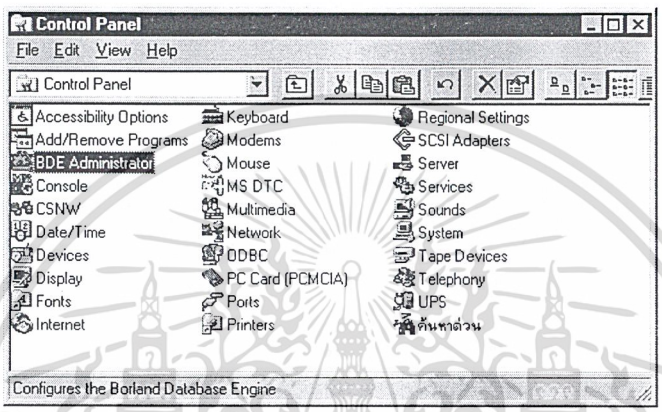
รูปที่ ก-7

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ข
วิธีการสร้าง BDE ในการเชื่อมต่อกับ Database

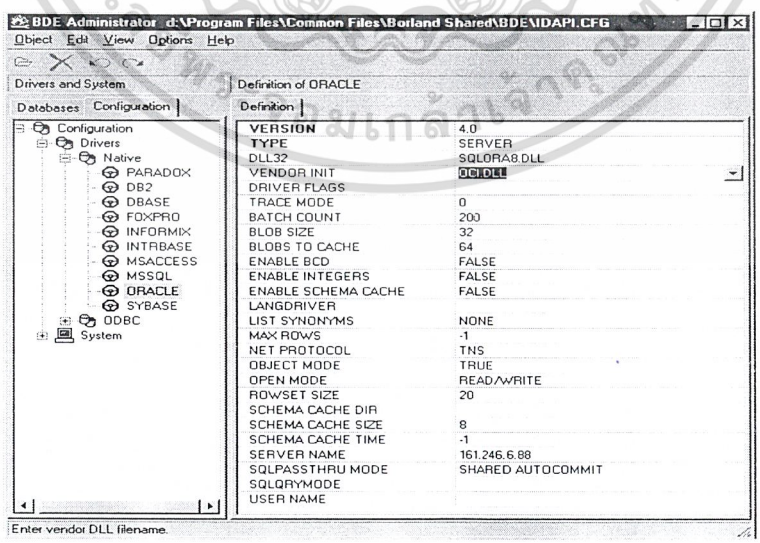
BDE (Borland Database Engine) คือ วิธีการเชื่อมต่อ (Driver) ระหว่าง Application ที่อยู่ฝั่ง Client กับ ฐานข้อมูลที่อยู่ฝั่ง Server

เรียก BDE Administrtor จาก Control Panel ดังรูปที่ ข-1



รูปที่ ข-1

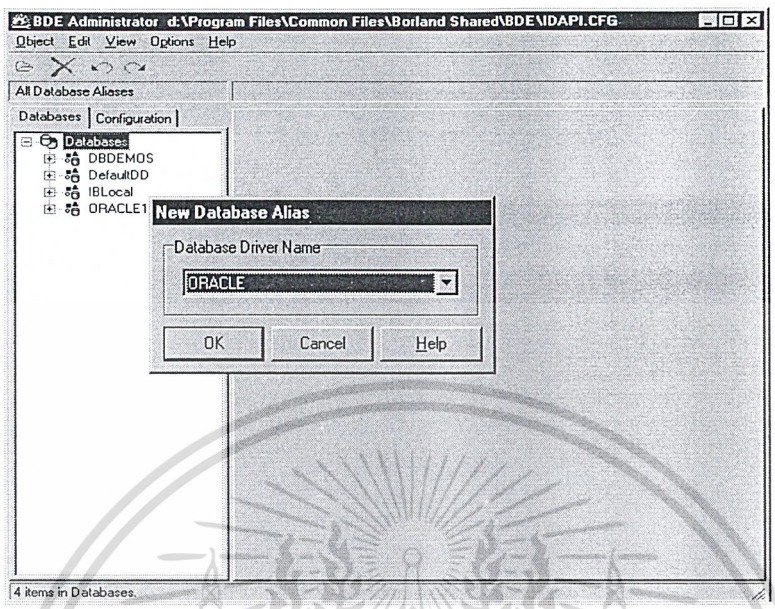
เมื่อเรียกแล้วจะปรากฏดังรูปที่ ข-2 ให้เลือก Configuration แล้วเลือก DLL32 เป็น SQLORA8.DLL และ เลือก VENDOR INIT เป็น OCI.DLL และก็ใส่ SERVER NAME หรือ IPในที่นี้ใช้ IP 161.246.6.88 รูปที่ ข-2



รูปที่ ข-2

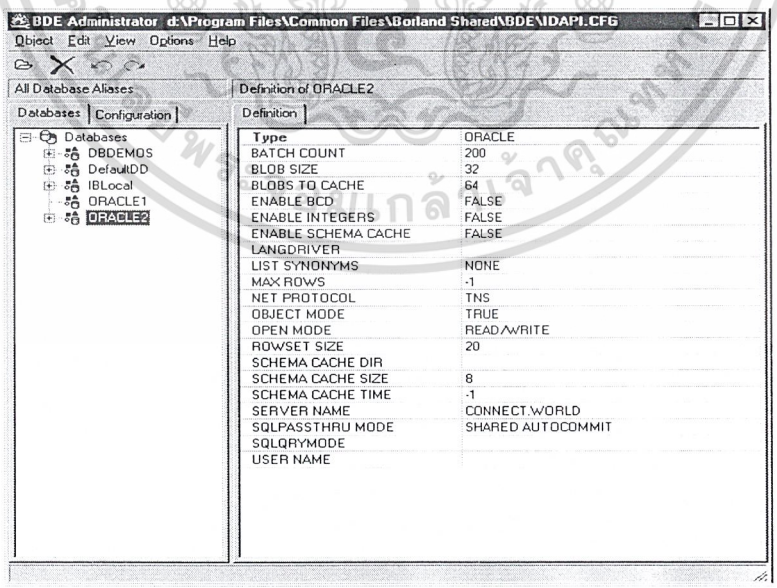
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากเลือกเสร็จแล้วก็ Apply จากนั้นก็เลือก Databases ดังรูป ข-3 แล้วก็ New Database Alias เลือก Database Driver Name เป็น Oracle จาก Menu Object



รูปที่ ข-3

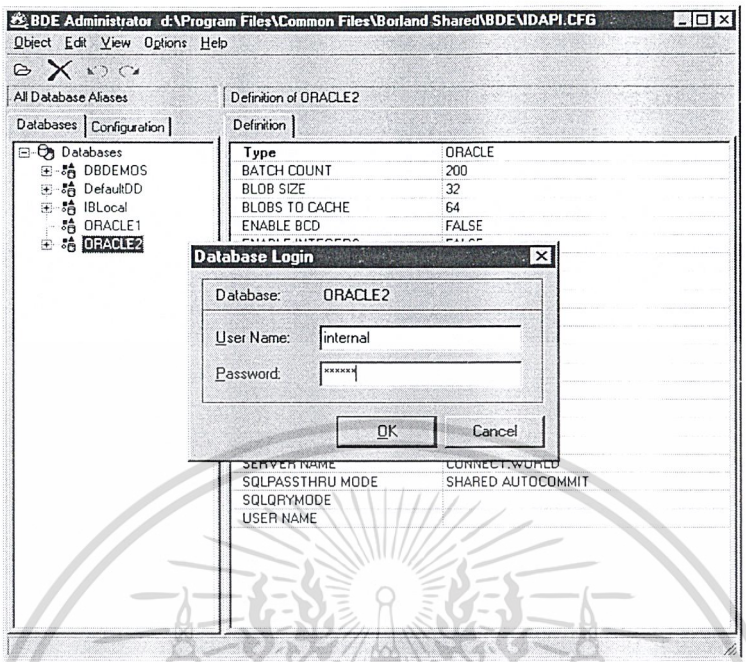
หลักจากตกลงสร้าง Database Alias ตัวใหม่แล้วจะได้ชื่อ BDE ขึ้นมาจากนั้นก็ Apply แล้วเลือก SERVER NAME เป็น ชื่อ service ที่เราได้สร้างขึ้นจาก ภาคผนวก ก



รูปที่ ข-4

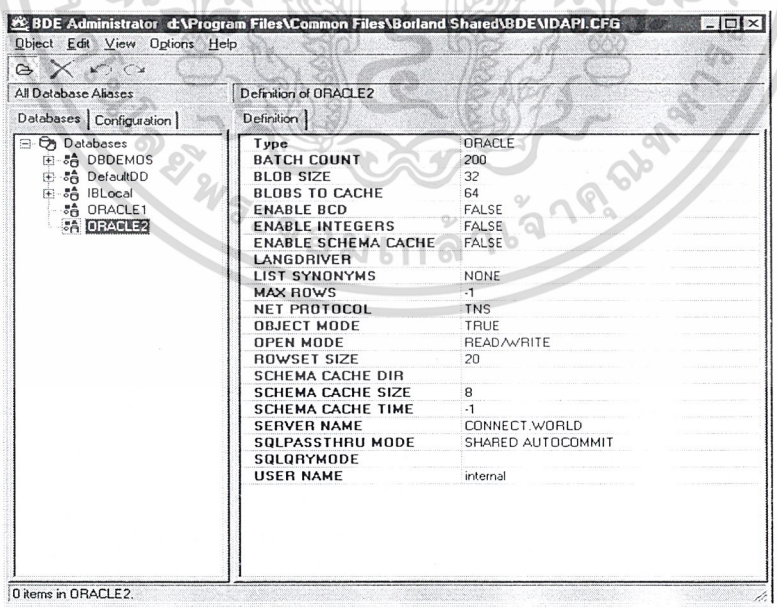
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากนั้นก็ทำการทดสอบ BDE ที่เราได้สร้างโดยการดับเบิ้ลคลิ๊ก ดังรูปที่ ข-5



รูปที่ ข-5

เมื่อทุกอย่างถูกต้องจะได้ดังรูปที่ ข-6



รูปที่ ข-6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ค

Source Program

Source Code ในส่วนของ Client

1. LOGO. PAS

```
unit logo;

interface

uses
  Windows, Messages, Controls, SysUtils, Classes, Graphics, Forms, Dialogs,
  ExtCtrls;


type
  TLogoForm = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    procedure Image1Click(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;


var
  LogoForm: TLogoForm;


implementation


uses
    Main_Menu;


{$R *.DFM}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
procedure TLogoForm.Image1Click(Sender: TObject);
begin
    Self.hide;
    MainMenu.show;
end;

procedure TLogoForm.FormKeyPress(Sender: TObject; var Key: Char);
begin
    Self.hide;
    MainMenu.show;
end;

end.
```

2. MAIN_MENU. PAS

```
unit Main_menu;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Db, DBClient, MConnect, MidasCon, Grids, DBGrids, DBTables;

type
  TMainMenu = class(TForm)
    GroupBox1: TGroupBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    RemoteServer1: TRemoteServer;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button4Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
//

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainMenu: TMainMenu;

implementation

uses
    Open_Order;

{$R *.DFM}

procedure TMainMenu.FormClose(Sender: TObject; var Action: TCloseAction);
begin
     Self.RemoteServer1.Connected := false;
     Application.Terminate;
end;

procedure TMainMenu.Button4Click(Sender: TObject);
begin
//    Self.RemoteServer1.AppServer.endorder;
     Application.Terminate;
end;

procedure TMainMenu.Button1Click(Sender: TObject);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
var s:string;
   Rs : widestring;
begin
   s := inputbox('เปิดใบสั่งซื้อสินค้าใหม่','กรุณาป้อนรหัสลูกค้า','');
   if s<>'' then
   begin
      Rs := RemoteServer1.AppServer.StartOrder(s);
      if Rs<>'' then
         showmessage(Rs)
      else OpenOrders.Show;
   end;

end;

procedure TMainMenu.Button2Click(Sender: TObject);
var s:string;
   Rs:widestring;
begin
   s := '';
   s := inputbox('แก้ไขใบสั่งซื้อสินค้า','กรุณาป้อนรหัสใบสั่งซื้อสินค้า','');
   if s<>'' then
   begin
      Rs := RemoteServer1.AppServer.EditOrder(s);
      if Rs<>'' then
         showmessage(Rs)
      else OpenOrders.Show;
   end;
end;

procedure TMainMenu.Button3Click(Sender: TObject);
var s:string;
   RS : widestring;
begin
   s := '';
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    s := inputbox('ลบใบสั่งซื้อสินค้า ','กรุณาป้อนรหัสใบสั่งซื้อสินค้า','');
    if s<>'' then
    begin
        Rs := RemoteServer1.AppServer.DelOrder(s);
        showmessage(Rs);
    end;
end;

procedure TMainMenu.FormActivate(Sender: TObject);
begin
    Self.RemoteServer1.Connected := true;
end;

end.
```

3. OPEN_ORDER . PAS

```
unit Open_Order;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, StdCtrls, Grids, DBGrids, ExtCtrls, Mask, DBCtrls, Db, DBClient,
  MConnect, MidasCon,Report;

type
  TOpenOrders = class(TForm)
    PageControl1: TPageControl;
    Sheet1: TTabSheet;
    Sheet2: TTabSheet;
    Panel1: TPanel;
    GroupBox1: TGroupBox;
    DBGrid1: TDBGrid;
    OrderId: TLabel;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DBText1: TDBText;

Date: TLabel;

DBText2: TDBText;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Label11: TLabel;

Label12: TLabel;

Label13: TLabel;

GroupBox2: TGroupBox;

Label14: TLabel;

Label15: TLabel;

Label16: TLabel;

Label17: TLabel;

Label18: TLabel;

Label19: TLabel;

Label20: TLabel;

Label21: TLabel;

Label22: TLabel;

DBEdit1: TDBEdit;

DBEdit2: TDBEdit;

DBText3: TDBText;

DBText4: TDBText;

DBText5: TDBText;

DBText6: TDBText;

DBText7: TDBText;

DBText8: TDBText;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น ทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DBText9: TDBText;

DBText10: TDBText;

DBText11: TDBText;

DBText12: TDBText;

DBText13: TDBText;

DBText14: TDBText;

DBText15: TDBText;

DBText16: TDBText;

DBText17: TDBText;

DBText18: TDBText;

DBText19: TDBText;

DBText20: TDBText;

Button1: TButton;

Button2: TButton;

GroupBox3: TGroupBox;

GroupBox4: TGroupBox;

GroupBox5: TGroupBox;

Label23: TLabel;

Label24: TLabel;

Label25: TLabel;

Label26: TLabel;

Label27: TLabel;

Edit1: TEdit;

Edit2: TEdit;

Button5: TButton;

Edit3: TEdit;

Button6: TButton;

DBText21: TDBText;

Label28: TLabel;

Label30: TLabel;

ClientDataSet1: TClientDataSet;

ClientDataSet2: TClientDataSet;

DataSource1: TDataSource;

DataSource2: TDataSource;

เอกสารนี้สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ClientDataSet5: TClientDataSet;

DataSource5: TDataSource;

Button4: TButton;

ClientDataSet1ORDER_ID: TFloatField;

ClientDataSet1CUSTOMER_ID: TStringField;

ClientDataSet1ORDER_DATE: TStringField;

ClientDataSet1SHIP_DATE: TStringField;

ClientDataSet1SELLER_ID: TStringField;

ClientDataSet1LINE_ITEM: TDataSetField;

ClientDataSet2ID: TFloatField;

ClientDataSet2UPC: TStringField;

ClientDataSet2DESCRIPTION: TStringField;

ClientDataSet2PRICE: TFloatField;

ClientDataSet2QTY: TFloatField;

ClientDataSet2DISCOUNT: TFloatField;

ClientDataSet2DISCOUNTBaht: TFloatField;

ClientDataSet2SUB_TOTAL: TFloatField;

ClientDataSet5CUSTOMER_ID: TStringField;

ClientDataSet5FIRST_NAME: TStringField;

ClientDataSet5LAST_NAME: TStringField;

ClientDataSet5COMPANY: TStringField;

ClientDataSet5PERCENT_DISCOUNT: TFloatField;

ClientDataSet5ADDRESS: TADTField;

ClientDataSet5ADDRESS_ID: TStringField;

ClientDataSet5STREET: TStringField;

ClientDataSet5TAMBOL: TStringField;

ClientDataSet5AMPHOR: TStringField;

ClientDataSet5PROVINCE: TStringField;

ClientDataSet5ZIPCODE: TStringField;

ClientDataSet5TEL: TStringField;

ClientDataSet5VIA_TO: TADTField;

ClientDataSet5ADDRESS_ID2: TStringField;

ClientDataSet5STREET2: TStringField;

ClientDataSet5TAMBOL2: TStringField;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    ClientDataSet5AMPHOR2: TStringField;
    ClientDataSet5PROVINCE2: TStringField;
    ClientDataSet5ZIPCODE2: TStringField;
    ClientDataSet5TEL2: TStringField;
    RemoteServer1: TRemoteServer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  //poo
    function Total:real;
    procedure MakeLineItem(UPC:string;QTY:integer);
    procedure DelLineItem(ID:integer);
    procedure FormActivate(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  OpenOrders: TOpenOrders;

implementation

{$R *.DFM}

var Order_ID : integer;

procedure TOpenOrders.Button1Click(Sender: TObject);
begin
```

เอกสารนี้มีลิขสิทธิ์ สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    ClientDataSet1.ApplyUpdates(500);
    Self.RemoteServer1.AppServer.endorder;
    Application.Terminate;
end;

procedure TOpenOrders.Button2Click(Sender: TObject);
begin
    ClientDataSet2.ApplyUpdates(500);
    ClientDataSet1.ApplyUpdates(500);
    Self.RemoteServer1.AppServer.endorder;
    Self.Close;
end;

function TOpenOrders.Total:real;
begin
    Result := StrToFloat(RemoteServer1.AppServer.Total);
end;

procedure TOpenOrders.MakeLineItem(UPC:string;QTY:integer);
var ID,KindOf : integer;
   Price,Discount : real;
begin
end;

procedure TOpenOrders.DelLineItem(ID:integer);
var m,n : integer;
begin
end;

procedure TOpenOrders.FormActivate(Sender: TObject);
begin
    Self.RemoteServer1.Connected := true;
    Order_ID := RemoteServer1.AppServer.GetOrderID;
    ClientDataSet1.Refresh;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    ClientDataSet1.SetKey;
    ClientDataSet1.FieldByName('ORDER_ID').AsInteger := Order_ID;
    ClientDataSet1.GotoKey;
    ClientDataSet5.Refresh;
    ClientDataSet2.Open;
end;

procedure TOpenOrders.Button5Click(Sender: TObject);
var Rs,UPC:widestring;
   QTY,OrderID:integer;
begin
    UPC := Self.Edit1.Text;
    QTY := StrToInt(Self.Edit2.Text);
    if (UPC<>'') and (QTY>0) then
    begin
       Rs := RemoteServer1.AppServer.MakeLineItem(UPC,QTY);
       if Rs <> '' then
          showmessage(Rs)
       else
       begin
          OrderID := RemoteServer1.AppServer.GetOrderID;
          ClientDataSet1.Close;
          ClientDataSet1.Open;
          ClientDataSet5.Close;
          ClientDataSet5.Open;
          ClientDataSet1.SetKey;
          ClientDataSet1.FieldByName('ORDER_ID').AsInteger := OrderID;
          ClientDataSet1.GotoKey;
          ClientDataSet2.Open;
       end;
    end;
end;

procedure TOpenOrders.Button6Click(Sender: TObject);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
var ID,ID1,n,OrderID,m:integer;
   Rs:widestring;
begin
   ID := StrToInt(EDit3.Text);
   ID1 := ID;
   ClientDataSet2.Close;
   ClientDataSet2.Open;
   if ID>0 then
   begin
      ClientDataSet2.SetKey;
      ClientDataSet2.FieldByName('ID').AsString := Edit3.Text;
      if ClientDataSet2.GotoKey then
      begin
         ClientDataSet2.Delete;
         while not ClientDataSet2.Eof do
         begin
            ClientDataSet2.Edit;
            ClientDataSet2.FieldByName('ID').AsInteger := ID;
            ID:=ID+1;
            ClientDAtaSet2.Next;
         end;
         ClientDataSet1.ApplyUpdates(500);
         Self.RemoteServer1.AppServer.DelLineItem(ID1);
      end
      else showmessage('ไม่พบรายการสินค้าที่ต้องการ');
   end;
end;

procedure TOpenOrders.Button4Click(Sender: TObject);
var Rs:widestring;
begin
   Rs := Self.RemoteServer1.AppServer.Total;
   Self.Label30.Caption := Rs + ' บาท';
end;
```

เอกสารนี้สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
procedure TOpenOrders.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Self.RemoteServer1.Connected := false;
end;
end.
```

Source Code ในส่วนของ Client

1. CUSTOMER . PAS

```
unit Customer;
interface

uses
   Address,Discount,MoneyDiscount,PercentDiscount,
   PartDiscount,AllDiscount;

type
   TCustomer = class
      CustomerID : string;
      FirstName : string;
      LastName : string;
      Company : string;
      Discount : TDiscount;
      Address : TAddress;
      ViaTo : TAddress;

      procedure SetCustomerID(CustomerID:string);
      procedure SetFirstName(FirstName:string);
      procedure SetLastName(LastNameD:string);
      procedure SetCompany(Company:string);
      procedure SetDiscount(Discount:TDiscount);
      procedure Init(CustomerID,FirstName,LastName,Company:string);
      function GetCustomerID:string;
      function GetFirstName:string;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น และห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    function GetLastName:string;
    function GetCompany:string;
    function GetDiscount:real;
  end;

implementation

procedure TCustomer.SetCustomerID(CustomerID:string);
begin
  Self.CustomerID := CustomerID;
end;

procedure TCustomer.SetFirstName(FirstName:string);
begin
  Self.FirstName := FirstName;
end;

procedure TCustomer.SetLastName(LastNameD:string);
begin
  Self.LastName := LastName;
end;

procedure TCustomer.SetCompany(Company:string);
begin
  Self.Company := Company;
end;

procedure TCustomer.SetDiscount(Discount:TDiscount);
begin
  Self.Discount := Discount;
end;

procedure TCustomer.Init(CustomerID,FirstName,LastName,Company:string);
begin
```

เอกสารนี้มีลิขสิทธิ์ ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    Self.CustomerID := CustomerID;
    Self.FirstName := FirstName;
    Self.LastName := LastName;
    Self.Company := Company;
    Self.Discount := TDiscount.Create;
    Self.Address := TAddress.Create;
    Self.ViaTo := TAddress.Create;
end;

function TCustomer.GetCustomerID:string;
begin
    Result := Self.CustomerID;
end;

function TCustomer.GetFirstName:string;
begin
    Result := Self.FirstName;
end;

function TCustomer.GetLastName:string;
begin
    Result := Self.LastName;
end;

function TCustomer.GetCompany:string;
begin
    Result := Self.Company;
end;

function TCustomer.GetDiscount:real;
begin
    Result := Self.Discount.GetDiscount;
end;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. LINE_ITEM . PAS

```
unit LineItem;
interface

uses
   Discount,MoneyDiscount,PercentDiscount,
   ProductSpecification,Dialogs,Sysutils;

type
   TLineItem = class
      ID : integer;
      Quantity : integer;
   ProdSpec : TProductSpecification;
   Discounts : TDiscount;

      procedure Init(ID:integer;Quantity:integer); virtual;
      procedure SetProSpec(Spec:TProductSpecification);
      function GetDiscount:real; virtual;
      function SubTotal:real; virtual;
      procedure SetDiscount(Disc : TDiscount);
   end;


implementation

procedure TLineItem.Init(ID:integer;Quantity:integer);
begin
   Self.ID := ID;
   Self.Quantity := Quantity;
end;

procedure TLineItem.SetProSpec(Spec:TProductSpecification);
begin
   Self.ProdSpec := TProductSpecification.Create;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    Self.ProdSpec := Spec;
  end;

  function TLineItem.GetDiscount:real;
  //var Quantity : integer;
  begin
    Quantity := Self.Quantity;
    result := Self.Discounts.GetDiscount; //ProdSpec.GetDiscount(Quantity);
  end;

  function TLineItem.SubTotal:real;
  var Sub : real;
  begin
    Sub := Self.Quantity * Self.ProdSpec.GetPrice;
    showmessage(IntToStr(Self.Quantity)+'*'+floattostr(Self.ProdSpec.GetPrice)+'*'+floattostr
 (Self.Discounts.GetDiscount));
    if Self.Discounts.ClassName='TMoneyDiscount' then
      Result := Sub - Self.Discounts.GetDiscount
    else
      Result := Sub - (Sub*Self.Discounts.GetDiscount);
    showmessage(
    'ID :'+IntToStr(Self.ID) +
    'SubTotal1 :'+FloatToStr(Sub)+
    'SubTotal2 :'+FloatToStr(Result));
  end;

 procedure TLineItem.SetDiscount(Disc : TDiscount);
 begin
   Self.Discounts := TDiscount.Create;
   Self.Discounts := Disc;
 end;

 end.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. ORDER . PAS

```
unit Order;

interface

uses
   ProductSpecification,LineItem,Address,Customer,
   Discount,MoneyDiscount,PercentDiscount,dialogs,sysutils;

type
   LineItemPtr = ^LineItemRec;
   LineItemRec = record
     Info : TLineItem;
     Next : LineItemPtr;
   end;

   TOrder = class
     OrderID : integer;
     OrderDate : string;
     ShipDate : string;
     SalerID : string;
     Count : integer;
     First : LineItemPtr;
     Last : LineItemPtr;
     Customers : TCustomer;
     procedure Init(OrderID:integer;
           OrderDate,ShipDate,SalerID:string);
     procedure SetCustomer(Cust : TCustomer);
     Procedure Add(LineItems:LineItemPtr);
     Procedure DelLineItem(ID:integer);
     procedure MakeLineItem(ID:integer;ProdSpec:TProductSpecification;
           Quantity:integer;Discounts:TDiscount);
     function CountLineItem:integer;
     function Total:real;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น ทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    procedure Find(ID:integer;var Trav:LineItemPtr);
    procedure Delete(var Trav:LineItemPtr);
    procedure UpdateID(ID:Integer);
    function GetCustomerID:string;
  end;

implementation

procedure TOrder.Find(ID:integer;var Trav:LineItemPtr);
begin
  Trav := Self.First;
  while (Trav^.Info.ID<>ID) and
      (Trav<>nil) do
  begin
    Trav := Trav.Next;
  end;
end;

procedure TOrder.Delete(var Trav:LineITemPtr);
var Temp:LineItemPtr;
begin
  if (Trav=Self.First) then
  begin
    Self.First:=nil;
    Self.Last:=nil;
    Trav:=nil;
    temp:=Trav;
  end
  else
  begin
    Temp := Self.First;
    while Temp.Next<>Trav do
      Temp := Temp.Next;
    if Trav<>Self.Last then
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
      begin
        Temp.Next := Trav.Next;
        Temp := Trav;
        Trav := Trav.Next;
      end
      else
      begin
        Self.Last := Temp;
        Temp := Temp.Next;
        Self.Last.Next := NIL;
      end;
    end;
    Temp^.Info.Free;
    Self.Count := Self.Count-1;
end;

procedure TOrder.UpdateID(ID:Integer);
var Trav:LineItemPtr;
begin
  Trav := Self.First;
  while Trav<>nil do
  begin
    if Trav^.Info.ID>ID
    then Trav^.Info.ID:=Trav^.Info.ID-1;
    Trav := Trav.Next;
  end;
end;

procedure TOrder.Init(OrderID:integer;OrderDate,ShipDate,SalerID:string);
begin
    Self.OrderID := OrderID;
    Self.OrderDate := OrderDate;
    Self.SalerID := SalerID;
    Self.Count := 0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    new(Self.First);
    Self.First := NIL;
    new(Self.Last);
    Self.Last := NIL;
end;

procedure TOrder.SetCustomer(Cust : TCustomer);
begin
    self.Customers := TCustomer.Create;
    self.Customers := Cust;
end;

procedure TOrder.Add(LineItems:LineItemPtr);
begin
    if Self.First=NIL then
    begin
        Self.First := LineItems;
        Self.First.Next := NIL;
    end
    else
    begin
        Self.Last.Next := LineItems;
    end;
    Self.Last := LineItems;
    Self.Last.Next := NIL;
    Self.Count:=Self.Count+1;
end;

procedure TOrder.DelLineItem(ID:integer);
var Trav:LineItemPtr;
begin
    if Self.First<>nil then
    begin
        Self.Find(ID,Trav);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
      Self.Delete(Trav);
      Self.UpdateID(ID);
    end;
end;

procedure TOrder.MakeLineItem(ID:integer;ProdSpec:TProductSpecification;
        Quantity:integer;Discounts:TDiscount);
var LineItems:LineItemPtr;

begin
    new(LineItems);
    LineItems^.Info := TLineItem.Create;
    LineItems^.Info.Init(ID,Quantity);
    LineItems^.Info.SetDiscount(Discounts);
    LineItems^.Info.SetProSpec(ProdSpec);
    Self.Add(LineItems);

end;

function TOrder.CountLineItem:integer;
begin
    Result := Self.Count;
end;

function TOrder.Total:real;
var Trav:LineItemPtr;
   Sum : real;
begin
    Sum := 0;
    Trav := Self.First;
    while Trav<>NIL do
    begin
        Sum := Sum + Trav^.Info.SubTotal;
        Trav := Trav.Next;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    end;
    Sum := Sum - ((Sum*self.Customers.GetDiscount)/100);
    Result := Sum;
end;

function TOrder.GetCustomerID:string;
begin
    Result := Self.Customers.CustomerID;
end;

end.
```

4. ORDERSYSTEMFR . PAS

```
unit OrderSystemFr;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComServ, ComObj, VCLCom, StdVcl, BdeProv, DataBkr, DBClient, OrderServer_TLB,
  Db, DBTables,
  ProductSpecification,LineItem,Address,Customer,
  Discount,MoneyDiscount,PercentDiscount,Order,
  OrderSystems, Provider;

type
  TOrderSystem = class(TRemoteDataModule, IOrderSystem)
    Table1: TTable;
    Table2: TTable;
    Table3: TTable;
    Table3CUSTOMER_ID: TStringField;
    Table3FIRST_NAME: TStringField;
    Table3LAST_NAME: TStringField;
    Table3COMPANY: TStringField;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table3PERCENT_DISCOUNT: TFloatField;

Table3ADDRESS: TADTField;

Table3ADDRESS_ID: TStringField;

Table3STREET: TStringField;

Table3TAMBOL: TStringField;

Table3AMPHOR: TStringField;

Table3PROVINCE: TStringField;

Table3ZIPCODE: TStringField;

Table3TEL: TStringField;

Table3VIA_TO: TADTField;

Table3ADDRESS_ID2: TStringField;

Table3STREET2: TStringField;

Table3TAMBOL2: TStringField;

Table3AMPHOR2: TStringField;

Table3PROVINCE2: TStringField;

Table3ZIPCODE2: TStringField;

Table3TEL2: TStringField;

NestedTable1: TNestedTable;

NestedTable2: TNestedTable;

NestedTable1ID: TFloatField;

NestedTable1UPC: TStringField;

NestedTable1QTY: TFloatField;

NestedTable2STEP: TFloatField;

NestedTable2DISCOUNT: TADTField;

NestedTable2BAHT: TFloatField;

NestedTable2PERCENT: TFloatField;

DataSetProvider1: TDataSetProvider;

DataSetProvider2: TDataSetProvider;

NestedTable1DESCRIPTION: TStringField;

NestedTable1PRICE: TFloatField;

NestedTable1DISCOUNT: TFloatField;

NestedTable1DISCOUNTBaht: TFloatField;

NestedTable1SUB_TOTAL: TFloatField;

Table1ORDER_ID: TFloatField;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    Table1CUSTOMER_ID: TStringField;
    Table1ORDER_DATE: TStringField;
    Table1SHIP_DATE: TStringField;
    Table1SELLER_ID: TStringField;
    Table1LINE_ITEM: TDataSetField;
    Query1: TQuery;
    Table2UPC: TStringField;
    Table2DESCRIPTION: TStringField;
    Table2PRICE: TFloatField;
    Table2DISCOUNT_LIST: TDataSetField;
    NestedTable1MATCH_STEP: TIntegerField;
    procedure NestedTable1CalcFields(DataSet: TDataSet);
  private
    { Private declarations }
  public
    { Public declarations }
  protected
    function StartOrder(const CustomerID: WideString): WideString; safecall;
    procedure EndOrder; safecall;
    function MakeLineItem(const UPC: WideString; QTY: Integer): WideString;
      safecall;
    function DelLineItem(ID: Integer): WideString; safecall;
    function Total: WideString; safecall;
    function GetOrderID: Integer; safecall;
    function GetCustomerID: WideString; safecall;
    function GetOrderDate: WideString; safecall;
    function Get_Table1: IProvider; safecall;
    function Get_Table2: IProvider; safecall;
    function Get_Table3: IProvider; safecall;
    function Get_DataSetProvider1: IProvider; safecall;
    function Get_DataSetProvider2: IProvider; safecall;
    function DelOrder(const OrderID: WideString): WideString; safecall;
    function EditOrder(const OrderID: WideString): WideString; safecall;
  //  procedure CalcNested1(Param1: Integer); safecall;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
end;

var
 OrderSystem: TOrderSystem;
 OrderSys : TOrderSystems;

implementation

{$R *.DFM}

function TOrderSystem.StartOrder(const CustomerID: WideString): WideString;
var OrderID : integer;
   Discount : real;
begin
    Self.Table1.Close;
    Self.Table2.CLose;
    Self.Table1.Open;
    Self.Table2.Open;

    Self.Table3.SetKey;
    Self.Table3CUSTOMER_ID.AsString := CustomerID;
    if Self.Table3.GotoKey then
    begin
        Discount := Self.Table3PERCENT_DISCOUNT.AsFloat;
        Self.Table1.Last;
        OrderID := Self.Table1ORDER_ID.AsInteger + 1;

        OrderSys := TOrderSystems.Create;
        OrderSys.StartOrder(OrderID,CustomerID,Discount);
        Self.Table1.Insert;
        Self.Table1ORDER_ID.AsString := IntToStr(OrderID);
        Self.Table1CUSTOMER_ID.AsString := CustomerID;
        Self.Table1ORDER_DATE.AsString := OrderSys.Orders.OrderDate;
//
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        Self.Table1.ApplyUpdates;
        sELF.Table1.CommitUpdates;
//
        Result := '';
    end
    else Result := 'ไม่พบรหัสลูกค้าหมายเลข '+CustomerID;


end;

procedure TOrderSystem.EndOrder;
begin
    OrderSys.EndOrder;
    OrderSys.Free;
end;

function TOrderSystem.MakeLineItem(const UPC: WideString;
  QTY: Integer): WideString;
var ID : integer;
   Baht,Percent,Price : real;
begin
    Self.Table1.Refresh;
    Self.Table2.Refresh;

    Self.Table1.SetKey;
    Self.Table1ORDER_ID.AsInteger := OrderSys.GetOrderID;
    Self.Table1.GotoKey;

    Self.NestedTable1.Open;
    Self.Table2.SetKey;
    Self.Table2UPC.AsString := UPC;
    if Self.Table2.GotoKey then
    begin
        Self.NestedTable1.Last;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
//        showmessage('BEFORE
'+Self.NestedTable1ID.AsString+'*'+IntToStr(Self.NestedTable1.RecNo)+'*'+IntToStr(Self.NestedTabl
e1.RecordCount));

        ID := Ordersys.Orders.Count;
//        showmessage(IntToStr(ID));
        ID := ID+1;

//
showmessage(Self.NestedTable1ID.AsString+'*'+IntToStr(Self.NestedTable1.RecNo)+'*'+IntToStr
(Self.NestedTable1.RecordCount)+'*'+IntToStr(ID));

        Self.NestedTable1.Insert;
        Self.NestedTable1ID.AsInteger := ID;
        Self.NestedTable1UPC.AsString := UPC;
        Self.NestedTable1QTY.AsInteger := QTY;
//
        Self.NestedTable1.ApplyUpdates;
        Self.NestedTable1.CommitUpdates;
        Self.Table1.ApplyUpdates;
        Self.Table1.CommitUpdates;

        Self.Table1.Refresh;
        Self.Table1.SetKey;
        Self.Table1ORDER_ID.AsInteger := OrderSys.GetOrderID;
        Self.Table1.GotoKey;
        Self.NestedTable1.Open;
        Self.NestedTable1.First;
        While (not Self.NestedTable1.Eof) and
            (Self.NestedTable1UPC.AsString <> UPC)
        do Self.NestedTable1.Next;
        Baht := 0;
        Percent := 0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        Baht := Self.NestedTable1DISCOUNTBaht.AsFloat;
        Percent := Self.NestedTable1DISCOUNT.AsFloat;
        Price := Self.NestedTable1PRICE.AsFloat;
        ShowMessage(
         ' Baht :: ' + FloatToStr(Baht) +
         ' Percent :: ' + FloatToStr(Percent) +
         ' Price :: ' + FloatToStr(Price) +
         ' RecNo :: ' + IntToStr(Self.NestedTable1.RecNo)
         );
         //
        if Baht<>0 then
        begin
          OrderSys.MakeLineItem(ID,UPC,QTY,Price,Baht,1);
        end
        else
        begin
          OrderSys.MakeLineItem(ID,UPC,QTY,Price,Percent,0);
        end;
        Result := '';
    end
    else Result := 'ไม่พบสินค้าที่ต้องการ';

end;

function TOrderSystem.DelLineItem(ID: Integer): WideString;
begin

        OrderSys.DelLineItem(ID);

        Result := '';
end;

function TOrderSystem.Total: WideString;
var s : widestring;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
begin
    s := FLoatToStr(OrderSys.Total);
    Result := s;
end;

function TOrderSystem.GetOrderID: Integer;
begin
    Result := OrderSys.GetOrderID;
end;

function TOrderSystem.GetCustomerID: WideString;
var s:widestring;
begin
    s := OrderSys.GetCustomerID;
    Result := s;
end;

function TOrderSystem.GetOrderDate: WideString;
var s:widestring;
begin
    s := OrderSys.GetOrderDate;
    Result := s;
end;

function TOrderSystem.Get_Table1: IProvider;
begin
  Result := Table1.Provider;
end;

function TOrderSystem.Get_Table2: IProvider;
begin
  Result := Table2.Provider;
end;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
function TOrderSystem.Get_Table3: IProvider;
begin
  Result := Table3.Provider;
end;

function TOrderSystem.Get_DataSetProvider1: IProvider;
begin
  Result := DataSetProvider1.Provider;
end;

function TOrderSystem.Get_DataSetProvider2: IProvider;
begin
  Result := DataSetProvider2.Provider;
end;

function TOrderSystem.DelOrder(const OrderID: WideString): WideString;
begin
    Self.Table1.SetKey;
    Self.Table1ORDER_ID.AsString := OrderID;
    if Self.Table1.GotoKey then
    begin
       //Self.Table1.Edit;
       Self.Table1.Delete;
 //
       Self.Table1.ApplyUpdates;
       Self.Table1.CommitUpdates;
 //
       Result := ('ลบใบสั่งซื้อหมายเลข '+OrderID+' เรียบร้อยแล้ว');
    end
    else Result := ('ไม่พบใบสั่งซื้อที่ต้องการ');
 end;

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
function TOrderSystem.EditOrder(const OrderID: WideString): WideString;
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
var Discount : real;
```

```
    Baht,Percent : real;
    ID,QTY,KindOf : integer;
    CustomerID,UPC : string;
    Price : real;
    n,m : integer;
begin
    Self.Table1.Close;
    Self.Table3.Close;

    Self.Table1.Open;
    Self.Table3.Open;


    Self.Table1.SetKey;
    Self.Table1ORDER_ID.AsString := OrderID;
    if Self.Table1.GotoKey then
    begin

        Self.NestedTable1.Close;
        Self.NestedTable1.Open;

        CustomerID := Self.Table1CUSTOMER_ID.AsString;      Self.Table3.SetKey;
        Self.Table3CUSTOMER_ID.AsString := CustomerID;
        Self.Table3.GotoKey;
        Discount := Self.Table3PERCENT_DISCOUNT.AsFloat;

        OrderSys := TOrderSystems.Create;
        OrderSys.StartOrder(StrToInt(OrderID),CustomerID,Discount);

//        n := Self.NestedTable1.RecordCount;
        Self.NestedTable1.First;
//        m := 1;
        while not Self.NestedTable1.Eof do
        begin
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        ID := Self.NestedTable1ID.AsInteger;
        UPC := Self.NestedTable1UPC.AsString;
        QTY := Self.NestedTable1QTY.AsInteger;
        Price := Self.NestedTable1PRICE.AsFloat;

        //Self.NestedTable1.Edit;
        //Self.NestedTable1.OnCalcFields;

        Baht := 0;
        Percent := 0;
        Baht := Self.NestedTable1DISCOUNTBaht.AsInteger;
        Percent := Self.NestedTable1DISCOUNT.AsInteger;
        showmessage(FloatToStr(Baht)+'*'+FloatToStr(Percent)+'*'+IntToStr(QTY));
        if Baht<>0 then
          OrderSys.MakeLineItem(ID,UPC,QTY,Price,Baht,1)
        else
          OrderSys.MakeLineItem(ID,UPC,QTY,Price,Percent,0);

//        m := m+1;
        Self.NestedTable1.Next;
      end;

      Result := '';
    end
    else Result := 'ไมพบใบสั่งซื้อที่ต้องการ';
end;

{procedure TOrderSystem.NestedTable1CalcFields(DataSet: TDataSet);
var QTY,Step : integer;
  Price : real;
  Baht,Percent : real;
  UPC : string;
begin
  QTY := Self.NestedTable1QTY.AsInteger;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
  Price := Self.NestedTable1PRICE.AsFloat;
  Baht := 0;
  Percent := 0;

  UPC := Self.NestedTable1UPC.AsString;
  Self.Table2.SetKey;
  Self.Table2UPC.AsString := UPC;
  Self.Table2.GotoKey;

  Self.NestedTable2.Last;
  Step := Self.NestedTable2STEP.AsInteger;

  while Step>QTY do
  begin
     Self.NestedTable2.Prior;
  end;

  Baht := Self.NestedTable2BAHT.AsFloat;
  Percent := Self.NestedTable2PERCENT.AsFloat;

  Self.NestedTable1DISCOUNTBaht.AsFloat := Baht;
  Self.NestedTable1DISCOUNT.AsFloat := Percent;

  Self.NestedTable1SUB_TOTAL.AsFloat := Price - Baht - ((Percent*Price)/100);

end;}

{procedure TOrderSystem.CalcNested1(Param1: Integer);
var QTY,Step : integer;
   Price : real;
   Baht,Percent : real;
   UPC : string;
begin
   QTY := Self.NestedTable1QTY.AsInteger;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    Price := Self.NestedTable1PRICE.AsFloat;
    Baht := 0;
    Percent := 0;

    UPC := Self.NestedTable1UPC.AsString;
    Self.Table2.SetKey;
    Self.Table2UPC.AsString := UPC;
    Self.Table2.GotoKey;

    Self.NestedTable2.Last;
    Step := Self.NestedTable2STEP.AsInteger;

    while Step>QTY do
    begin
       Self.NestedTable2.Prior;
    end;

    Baht := Self.NestedTable2BAHT.AsFloat;
    Percent := Self.NestedTable2PERCENT.AsFloat;

    Self.NestedTable1DISCOUNTBaht.AsFloat := Baht;
    Self.NestedTable1DISCOUNT.AsFloat := Percent;

    Self.NestedTable1SUB_TOTAL.AsFloat := Price - Baht - ((Percent*Price)/100);

end;}


procedure TOrderSystem.NestedTable1CalcFields(DataSet: TDataSet);
var QTY,Step,MatchStep : integer;
    Price : real;
    Baht,Percent : real;
    UPC,s : string;
    Saveplace : TBookmark;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
begin
    QTY := Self.NestedTable1QTY.AsInteger;
    Price := Self.NestedTable1PRICE.AsFloat;
    Baht := 0;
    Percent := 0;
    UPC := Self.NestedTable1UPC.AsString;
    Self.Table2.Close;
    Self.Table2.Open;
    Self.Table2.SetKey;
    Self.Table2UPC.AsString := UPC;
    Self.Table2.GotoKey;

    Self.NestedTable2.Open;

    MatchStep := 0;
    Self.NestedTable2.First;
   while not Self.NestedTable2.Eof do
    begin
       Step := Self.NestedTable2STEP.AsInteger;
       if (QTY>Step) and (Step>MatchStep) then
       begin
         MatchStep:=Step;
       end;
     Self.NestedTable2.Next;
    end;
   Self.NestedTable1MATCH_STEP.AsInteger := MatchStep;
   if MatchStep <> 0 then
   begin
    Self.NestedTable2.First;
    while (not Self.NestedTable2.Eof) and
       (Self.NestedTable2STEP.AsInteger<>MatchStep)
       do Self.NestedTable2.Next;
    Baht := Self.NestedTable2BAHT.AsFloat;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    Percent := Self.NestedTable2PERCENT.AsFloat;
   end;

    Self.NestedTable1DISCOUNTBaht.AsFloat := Baht;
    Self.NestedTable1DISCOUNT.AsFloat := Percent;
    Self.NestedTable1SUB_TOTAL.AsFloat
    :=(Price*QTY) - Baht - ((Percent*Price*QTY)/100);
end;

initialization
 TComponentFactory.Create(ComServer, TOrderSystem,
   Class_OrderSystem, ciMultiInstance, tmApartment);
end.
```

5. SERVER . PAS

```
unit Server;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
{  ProductSpecification,LineItem,Address,Customer,
  Discount,MoneyDiscount,PercentDiscount,Order,
  OrderSystems,OrderSystemFr, Db, DBTables, Grids, DBGrids, DBClient,
  MConnect, MidasCon;}

type
  TServerForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
  private
    { Private declarations }
  public
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    { Public declarations }
  end;

var
  ServerForm: TServerForm;

implementation

{SR *.DFM}

end.
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้