การพัฒนาแอพพลิเคชันเชิงวัตถุโดยใช้สถาปัตยกรรม 3-เทียร์

3-tier Object Oriented Application Development

นายชญานนท์ โตวิกกัย

นายชาญชัย จอประเสริฐกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

การพัฒนาแอพพลิเคชันเชิงวัตถุโดยใช้สถาปัตยกรรม 3-เทียร์

3-tier Object Oriented Application Development

โดย

นายชญานนท์ โตวิกกัย

นายชาญชัย จอประเสริฐกุล

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาแอพพลิเคชันเชิงวัตถุโดยใช้สถาปัตยกรรม 3-เทียร์

ผู้จัดทำ

1. นายชญานนท์ โตวิกกัย รหัสประจำตัว 38014096
2. นายชาญชัย จอประเสริฐกุล รหัสประจำตัว 38014117

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาแอพพลิเคชันเชิงวัตถุโดยใช้สถาปัตยกรรม 3-เทียร์

นายชญานนท์ โตวิกกัย
นายชาญชัย จอประเสริฐกุล
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ในปัจจุบันคอมพิวเตอร์ได้เข้ามามีบทบาทสำคัญต่อชีวิตประจำวันของบุคคลทั่วไปมากขึ้น ผู้พัฒนาซอฟต์แวร์ในยุคใหม่จะพิจารณาถึงความสะดวกและง่ายต่อการบำรุงรักษาเป็นหลัก ดังนั้นแนวคิดการพัฒนาแอพพลิเคชันเชิงวัตถุจึงเริ่มได้รับความนิยม เนื่องจากคุณสมบัติเชิงวัตถุที่ง่ายต่อการแก้ไขและบำรุงรักษา

การพัฒนาแอพพลิเคชันผ่านระบบเครือข่ายคอมพิวเตอร์ในยุคก่อนจะอยู่ในลักษณะของสถาปัตยกรรม 2-เทียร์ ไคลเอนต์ / เซิร์ฟเวอร์ แต่ในปัจจุบันได้มีแนวความคิดใหม่ในการพัฒนาแอพพลิเคชันผ่านระบบเครือข่ายคอมพิวเตอร์ โดยใช้สถาปัตยกรรม 3-เทียร์ ไคลเอนต์ / เซิร์ฟเวอร์ ซึ่งทำให้การทำงานบนระบบเครือข่ายมีประสิทธิภาพขึ้น กอปรกับสามารถลดค่าใช้จ่ายในการบำรุงรักษาระบบลงได้อีกด้วย

ปริญญานิพนธ์นี้เป็นการนำเสนอแนวทางการวิเคราะห์ , ออกแบบและพัฒนาระบบงานโดยใช้หลักการเชิงวัตถุ ผ่านสถาปัตยกรรม 3-เทียร์ ไคลเอนต์ / เซิร์ฟเวอร์ ต่อเชื่อมกับระบบจัดการฐานข้อมูลเชิงวัตถุสัมพันธ์ Oracle โดยมีการนำมาประยุกต์ใช้งานจริงกับระบบการจัดเก็บสินค้า

3-tier Object Oriented Application Development

Mr. Chayanont Tovikkai

Mr. Chanchai Jorprasertkul

Advisor Dr. Worawat Limpoka

1998

Abstract

Computers have become a part of our daily life. Every application developers aim to develop the most efficiency software for users. The object oriented concept becomes popular in modern application development because of its useful characteristics such as the extensibility and reusability.

In network computing technologies, most client / server application developers have begun to develop their applications using 3-tier client /server architecture instead of 2-tier client /server architecture due to the several advantages of this architecture.

This thesis mainly presents the analysis, design, and development object oriented application using 3-tier client / server architecture connecting with "Oracle" object-relational database management system. We have implemented an object oriented application for the inventory control system.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี ด้วยความช่วยเหลือและร่วมมือจากบุคคลหลายๆฝ่าย ด้วยกัน พวกเราทั้งสองขอขอบพระคุณ

บิดา มารดา อันเป็นที่เคารพรักยิ่ง ที่อบรมเลี้ยงดูเราทั้งสองตั้งแต่เล็กจนถึงบัดนี้

อาจารย์ ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา ผู้คอยแนะนำแนวทางการทำงานที่ถูกต้องและให้ความรู้ในทุกๆด้าน ตลอดหนึ่งปีเต็มที่ผ่านมา

อาจารย์ทุกๆ ท่านที่ประสิทธิประสาทความรู้ให้เราทั้งสองตลอดสี่ปีที่ผ่านมา

พี่โม ,พี่เปิ้ล ,พี่สาโรจน์ บริษัท สหอินโฟ เทคโนโลยี จำกัด ผู้ให้คำแนะนำช่วยเหลือเมื่อเกิดปัญหา

สมาชิกห้องปฏิบัติการ Robotic and Hardware Application (ห้องฮาร์ดแวร์) สำหรับคำแนะนำ ช่วยเหลือ และกำลังใจที่มีให้กันมาโดยตลอด รวมถึงเอื้อเฟื้อสถานที่ทำงานตลอดหนึ่งปีที่ผ่านมา

ห้องวิจัยและพัฒนาระบบสื่อสารข้อมูลและเครือข่ายคอมพิวเตอร์ (ห้องเน็ตเวิร์ก) สำหรับสถานที่ทำงานและอุปกรณ์การทำงานทางคอมพิวเตอร์ทุกๆชิ้น

และสุดท้ายขอขอบคุณเพื่อนๆ ทีมงาน Gang4D ทุกท่าน สำหรับความร่วมมือ ความช่วยเหลือในทุกๆครั้งที่มีปัญหา คำแนะนำดีๆ เมื่อคิดอะไรไม่ออก ที่ขาดไม่ได้คือกำลังใจที่เพื่อนมีให้ตลอดเวลา

นายชญานนท์ โตวิกกัย

นายชาญชัย จอประเสริฐกุล

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 บทนำ

ในปัจจุบันนักพัฒนาโปรแกรมส่วนใหญ่ยังนิยมพัฒนาซอฟต์แวร์โดยใช้วิธีแบบโครงสร้าง (Structural Programming) ซึ่งมีประสิทธิภาพดีสำหรับงานที่มีขนาดไม่ใหญ่นัก และไม่มีความสลับซับซ้อนมาก (ใช้ภาษา C หรือภาษา Pascal) โดยใช้หลักการออกแบบชนิดบนลงล่าง (Top-Down Design) แต่สำหรับงานขนาดใหญ่และมีการทำงานในหลายๆส่วน ที่มีความสลับซับซ้อนรวมอยู่แล้ว การเขียนโปรแกรมโดยใช้วิธีแบบโครงสร้างนั้นค่อนข้างที่จะทำได้ลำบาก รวมทั้งสร้างความยุ่งยากต่อผู้ดูแลระบบในการปรับปรุงแก้ไข และบำรุงรักษาระหว่างการใช้งาน

การพัฒนาโปรแกรมเชิงวัตถุซึ่งได้ถูกคิดค้นขึ้นในปี ค.ศ.1968 โดยมีภาษาแรกได้แก่ภาษา Simula 68 และได้รับการพัฒนาอย่างต่อเนื่องจนกระทั่งถึงในปัจจุบัน มีทูลในการพัฒนาโปรแกรมเชิงวัตถุมากมาย เช่น Java , Delphi เป็นต้น จึงเป็นทางออกที่เหมาะสมที่สุดสำหรับนักพัฒนาโปรแกรมในปัจจุบัน

ประกอบกับขีดความสามารถของฐานข้อมูลเชิงสัมพันธ์ (Relational Database System : RDB) ที่มีข้อจำกัดหลายประการ อาทิเช่น

- ฐานข้อมูลเชิงสัมพันธ์ไม่รองรับข้อมูลที่มีความซับซ้อนได้
- ในการพัฒนาระบบโดยใช้ฐานข้อมูลเชิงสัมพันธ์ จะเกิดความยุ่งยากเมื่อต้องการแก้ไขเปลี่ยนแปลงในภายหลัง

ดังนั้นจึงมีการรวมหลักการเชิงวัตถุมาเข้ากับระบบฐานข้อมูลเชิงสัมพันธ์เพื่อทำให้ระบบฐานข้อมูลเชิงสัมพันธ์มีความสามารถทางออปเจ็กต์ได้ เรียกว่า "ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object-Relational Database System : ORDB) " โดยระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะจัดเก็บข้อมูลในรูปแบบตาราง และจัดการกับฐานข้อมูลแบบเชิงสัมพันธ์ แต่การติดต่อกับผู้ใช้งานได้นำหลักการของออปเจ็กต์โอเรียนเต็ด เข้ามาประยุกต์ใช้งานแทน

โครงงานนี้จึงนำเสนอการพัฒนาซอฟต์แวร์ระบบจัดการสินค้า (Stock System) ผ่านสถาปัตยกรรม 3-เทียร์ ไคลเอนต์/เซิร์ฟเวอร์โดยใช้วิธีการเชิงวัตถุ (Object-Oriented)

1.2 วัตถุประสงค์ของโครงงาน

1. วิเคราะห์และออกแบบระบบงานคลังสินค้าโดยใช้วิธีการเชิงวัตถุ
2. ศึกษาและใช้งานโมเดลที่ใช้ในการวิเคราะห์และออกแบบระบบ
3. วิเคราะห์และศึกษาค้นคว้าหลักการทำงานของสถาปัตยกรรม 3-เทียร์
4. ศึกษาการสร้างและออกแบบระบบฐานข้อมูลเชิงวัตถุสัมพันธ์
5. ศึกษาและทดลองใช้งานระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ Oracle 8.0.4

1.3 ขอบเขตของโครงงาน

โครงงานนี้ครอบคลุมถึงการวิเคราะห์และออกแบบเชิงวัตถุ (Object-Oriented Analysis and Design) โดยใช้การโมเดลแบบ UML (Unified Modeling Language) , การพัฒนาโปรแกรมการจัดเก็บสินค้า (Stock System) โดยใช้ทูล Delphi เวอร์ชัน 4.0 เป็นเครื่องมือในการพัฒนาผ่านระบบไคลเอนต์/เซิร์ฟเวอร์ ผ่านสถาปัตยกรรม 3-เทียร์ ซึ่งเป็นสถาปัตยกรรมใหม่ที่แตกต่างจากสถาปัตยกรรม 2-เทียร์ ไคลเอนต์/เซิร์ฟเวอร์ ที่ใช้กันในปัจจุบัน

ในส่วนของระบบฐานข้อมูล โครงงานนี้จะใช้ระบบการจัดเก็บข้อมูลแบบเชิงวัตถุสัมพันธ์ โดยเลือกใช้ระบบจัดการฐานข้อมูล Oracle เวอร์ชัน 8.0.4 รันบนระบบปฏิบัติการ WindowsNT เวอร์ชัน 4.0 เนื่องจากมีความยืดหยุ่นและประสิทธิภาพในการจัดเก็บข้อมูลเชิงวัตถุสัมพันธ์ในระดับหนึ่ง

1.4 ขั้นตอนการดำเนินงาน

เริ่มด้วยการศึกษาถึงทฤษฎีและหลักการเชิงวัตถุ ทั้งในส่วนการวิเคราะห์ , การออกแบบ ,และการโปรแกรม รวมถึงเลือกโมเดลและศึกวิธีการในการวิเคราะห์และออกแบบเชิงวัตถุ เพื่อนำมาประยุกต์ใช้งานโดยใช้ทูลเดลไฟล์เวอร์ชัน 4.0 เป็นทูลในการพัฒนาครั้งนี้ เนื่องจากมีคอมโพเนนต์ในการเชื่อมต่อผ่านสถาปัตยกรรม 3-เทียร์

จากนั้นทำการศึกษาถึงหลักการและการทำงานของสถาปัตยกรรม 3-เทียร์ ไคลเอนต์ / เซิร์ฟเวอร์ รวมถึงการทำงานของ Distributed Component Object Model (DCOM) และวิธีการตั้งค่าคอนฟิกูเรชันเพื่อให้การทำงานของ DCOM บนระบบปฏิบัติการ WindowsNT 4.0 เป็นไปอย่างสมบูรณ์แบบ

สุดท้ายก่อนที่จะเริ่มพัฒนาโปรแกรมได้นั้น เราต้องศึกษาถึงการจัดเก็บข้อมูลลงในฐานข้อมูลเพื่อให้ได้ระบบที่มีประสิทธิภาพสูงสุด โดยเริ่มศึกษาระบบฐานข้อมูลเชิงวัตถุ (Object-Oriented Database :OODB) ก่อน แต่ในการใช้งานจริงยังไม่มีทูลที่มีประสิทธิภาพรองรับ จึงเปลี่ยนมาใช้ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object-Relational Database :ORDB) โดยเลือกระบบจัดการฐานข้อมูลของ Oracle เวอร์ชัน 8.0.4 ซึ่งถือเป็นเวอร์ชันล่าสุดในขณะนั้น ก่อนที่จะนำหลักการทุกอย่างที่ได้ศึกษาค้นคว้าทั้งหมดมาประยุกต์ใช้งานเพื่อทำการพัฒนาระบบงานคลังสินค้าในโครงงานชิ้นนี้

บทที่ 2
ทฤษฎีและหลักการการพัฒนาโปรแกรมเชิงวัตถุ

การพัฒนาโปรแกรมในปัจจุบันส่วนใหญ่แล้วมักนิยมเลือกใช้วิธีแบบโครงสร้าง(Structural Programming) ซึ่งมีประสิทธิภาพดีสำหรับงานที่มีขนาดไม่ใหญ่มากนัก และมีความสลับซับซ้อนไม่มาก (เช่นโปรแกรมภาษา C และภาษา Pascal) โดยใช้หลักการออกแบบชนิดบนลงล่าง (Top-Down Design) แต่สำหรับงานขนาดใหญ่และมีการทำงานหลายๆส่วน ที่มีความสลับซับซ้อนนั้น การพัฒนาโปรแกรมด้วยวิธีแบบโครงสร้าง นั้น ค่อนข้างที่จะทำได้ลำบาก การพัฒนาโปรแกรมเชิงวัตถุ (Object-Oriented) เป็นวิธีการโปรแกรมวิธีใหม่ ที่เป็นคำตอบของปัญหาดังกล่าว

การพัฒนาโปรแกรมเชิงวัตถุเป็นวิธีการพัฒนาโปรแกรมคอมพิวเตอร์โดยใช้แนวความคิดเสมือนการมองภาพวัตถุ (Object) ในโลกแห่งความเป็นจริงและใช้การออกแบบชนิดล่างขึ้นบน (Bottom-Up Design) ซึ่งเป็นการจัดโครงสร้างของโปรแกรมให้เป็นระเบียบมากยิ่งขึ้นและเอื้ออำนวยต่อการพัฒนาโปรแกรมในเวลาต่อไป

การพัฒนาโปรแกรมเชิงวัตถุจะได้ผลอย่างแท้จริงและมีประสิทธิภาพเมื่อโปรแกรมที่ได้รับการพัฒนาผ่านขั้นตอนทั้ง 3 ขั้นตอน ดังนี้

1. ขั้นตอนการวิเคราะห์ (Analysis)
2. ขั้นตอนการออกแบบ (Design)
3. ขั้นตอนการเขียนโปรแกรม (Programming)

การพัฒนาโปรแกรมเชิงวัตถุเป็นการแบ่งรายละเอียดต่างๆออกเป็นหลายระดับแล้วนำมาวิเคราะห์และออกแบบ เป็นการวิเคราะห์ระบบในโลกความจริงซึ่งแบ่งทุกสิ่งทุกอย่างในโลกออกเป็นออปเจ็กต์ซึ่งมีความเกี่ยวข้องกัน ซึ่งหมายความว่าในหนึ่งโมเดลอย่างน้อยต้องประกอบด้วยหนึ่งออปเจ็กต์ และเราสามารถที่จะเปลี่ยนขอบเขตของปัญหาให้อยู่ในแบบโมเดล บางออปเจ็กต์สามารถที่จะแบ่งแยกออกมาได้ง่ายแต่บางออปเจ็กต์ก็ไม่สามารถแบ่งแยกออกมาได้ ดังนั้นเราจึงเรียกออปเจ็กต์ที่เราแบ่งแยกออกมาแล้วนั้นว่าเป็น Semantic Object เพราะว่าเราจะได้ออปเจ็กต์นั้นมาจากวัตถุหนึ่งที่มีความหมายเท่านั้น

2.1 การวิเคราะห์และออกแบบเชิงวัตถุ (Object-Oriented Analysis and Design)

ในปัจจุบันมีวิธีการสร้างโมเดลเชิงวัตถุหลากหลายวิธีด้วยกัน หนึ่งในนั้นได้แก่โมเดลแบบ UML (Unified Modeling Language) ซึ่งได้รับการพัฒนาโดย Gary Booch และ James Rumbaugh ในปี 1994 และได้ถูกนำมาสร้างเป็นมาตรฐาน โดย OMG (Object Management Group) ในปี 1997

หลักการในการวิเคราะห์ (Analysis) และการออกแบบ (Design) แบ่งเป็นส่วนต่าง ๆ ดังต่อไปนี้

การวิเคราะห์ แบ่งออกเป็น 7 ขั้นตอนได้แก่

1. define essential use cases

2. refine use case diagrams
3. refine conceptual model
4. refine glossary
5. define system sequence diagrams
6. define operation contracts
7. define state diagrams

การออกแบบ แบ่งออกเป็น 6 ขั้นตอนได้แก่

1. define real use cases
2. define reports, UI and storyboards
3. refine system architecture
4. define interaction diagrams
5. define design class diagrams
6. define database schema

2.1.1 การวิเคราะห์เชิงวัตถุ (Object-Oriented Analysis) มีทั้งสิ้น 7 ขั้นตอนตามลำดับดังนี้

2.1.1.1 Define Essential Use Cases

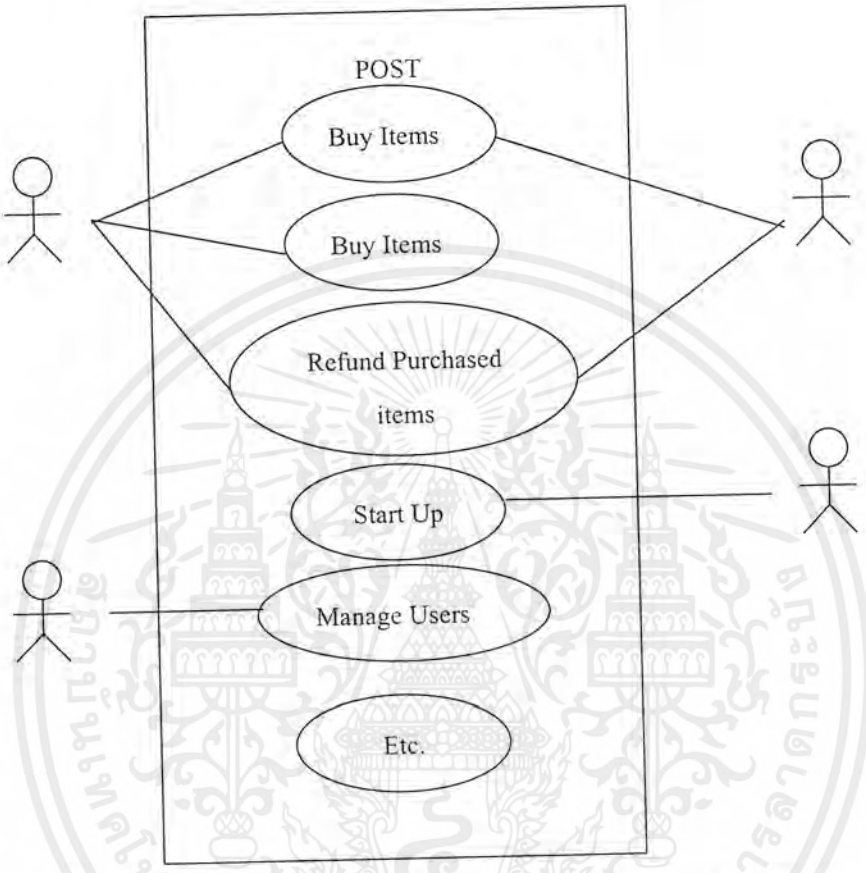
การกำหนด use case จำเป็นต้องเข้าใจความต้องการของระบบ, domain processes และปัจจัยภายนอกอื่น ๆ เพื่อนำมากำหนด requirement ในเชิงโครงสร้างดังต่อไปนี้

Actor Action	System response
1. use case นี้เริ่มจากเมื่อมีลูกค้าเข้ามาซื้อสินค้าและจ่ายเงินที่เครื่อง POST	
2. สินค้าแต่ละชิ้น แคชเชียร์ต้องประทับ Universal Product Code (UPC) ซึ่งสามารถกำหนดปริมาณของสินค้าได้ถ้าสินค้ามีหลายชิ้น	3. เพิ่มข้อมูลสินค้าเข้าไปยัง transaction ของการขายราคาของสินค้าปัจจุบันแสดงไว้ในช่อง B และ F
4. สิ้นสุดการขายเมื่อแคชเชียร์ระบุสินค้าที่ทำการขายเสร็จสิ้น	5. คำนวณและแสดงยอดรวมทั้งหมด
6. แคชเชียร์บอกราคาทั้งหมดให้แก่ลูกค้า	
7. ลูกค้าจ่ายเงินสด	
8. แคชเชียร์บันทึกจำนวนเงินที่ได้มา	9. แสดงเงินทอน
10. แคชเชียร์คืนเงินทอนพร้อมด้วยใบเสร็จรับเงิน	11. ทำการล็อคการขายที่สมบูรณ์
12. ลูกค้าออกจากร้านพร้อมสินค้า	

ตารางที่ 2-1 แสดงความต้องการของระบบ

2.1.1.2 Refine Use Case Diagrams

จากความต้องการหรือ use case ของระบบในเบื้องต้นสามารถนำมาสร้างเป็นไดอะแกรมได้ดังต่อไปนี้



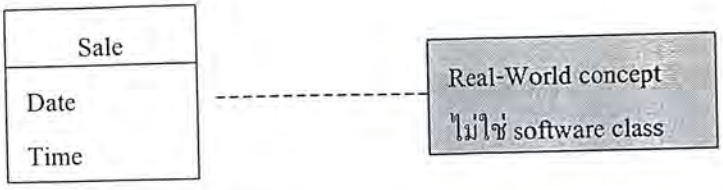
รูปที่ 2-1 แสดง Use-Case Diagram เพียงบางส่วนของ POST Application

2.1.1.3 Refine Conceptual Model

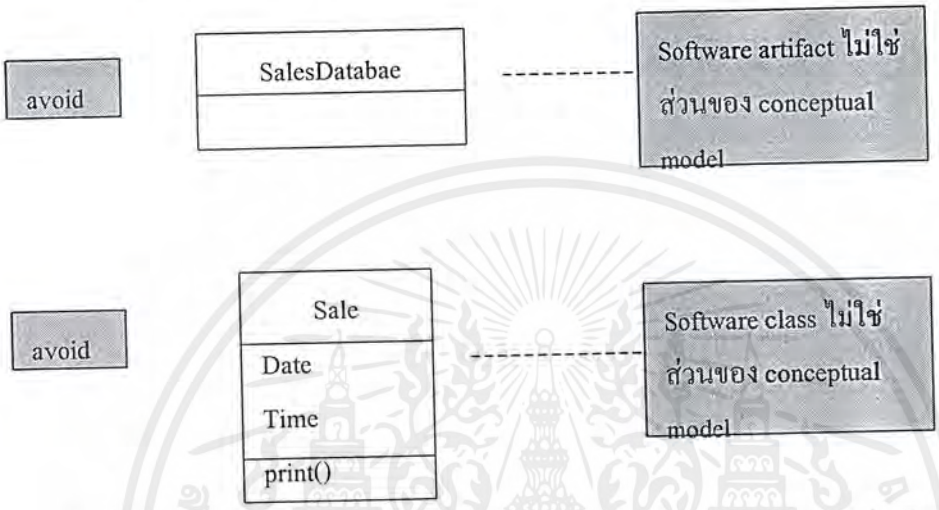
Conceptual Model แสดงถึงแนวความคิดของปัญหาซึ่งเป็นส่วนสำคัญในการสร้างส่วนของการวิเคราะห์เชิงวัตถุ และมีประโยชน์อย่างมากในการนำไปออกแบบ ซึ่งแสดงส่วนต่าง ๆ ได้ดังต่อไปนี้

- concepts
- association
- attributes

Conceptual Model เป็นเพียงแนวความคิดที่ไม่ใช่ส่วนประกอบของซอฟต์แวร์ดังนั้นจะไม่มีส่วนของวินโดว์หรือฐานข้อมูล และจะไม่มีส่วนที่เป็นเมทธอดแต่จะมีเพียงแอตทริบิวต์เท่านั้น



รูปที่ 2-2 แสดง Conceptual Model ที่แสดงถึง Real-World concepts



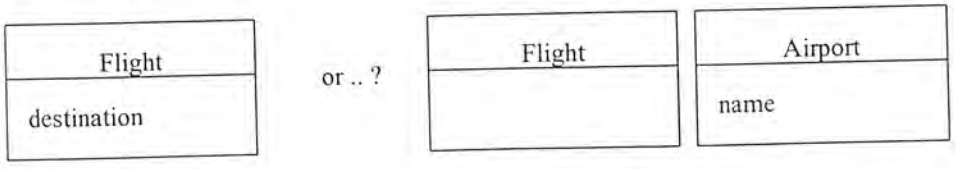
รูปที่ 2-3 แสดง Conceptual Model โดยจะต้องไม่แสดงถึงส่วนของซอฟต์แวร์หรือคลาส

วิธีการสร้าง Conceptual Model

1. จัด candidate concepts โดยใช้ concept category list และ noun phrase identification ที่สัมพันธ์กัน
2. วาดลงไปใน Conceptual Model
3. เพิ่ม association เพื่อแสดงถึงความสัมพันธ์ของแต่ละส่วน
4. เพิ่มแอตทริบิวต์เท่าที่ต้องการ

ความผิดพลาดในการระบุ Concept

ในการกำหนด Concept Model อาจจะเกิดความผิดพลาดในการระบุ Concept ได้ดังนั้น จึงมีกฎบางอย่างที่จะป้องกันความผิดพลาดเหล่านี้คือ " ถ้าเราไม่ได้คิดว่า concept X เป็นจำนวนหนึ่งหรือเป็นข้อความหนึ่งใน Real World แล้วนั้น X จะเป็น Concept ไม่ใช่แอตทริบิวต์ " ตัวอย่างเช่น destination ควรจะเป็นแอตทริบิวต์ของ Flight หรือ Concept อื่น ๆ ของ Airport

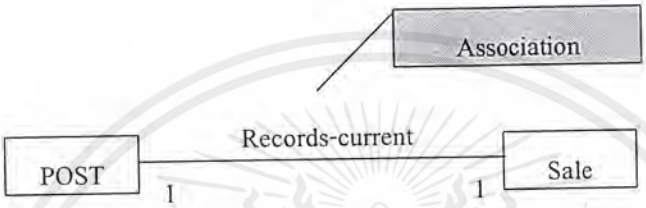


รูปที่ 2-4 แสดง Domain of airline reservations

ในโลกแห่งความจริง (Real World) นั้น destination airport ไม่ได้เป็นจำนวนหรือข้อความ ดังนั้น destination ควรจะเป็น concept

การเพิ่ม association

association เป็นความสัมพันธ์ระหว่าง concept ที่แสดงถึงความหมายและความสัมพันธ์ระหว่างกัน



รูปที่ 2-5 แสดง associations

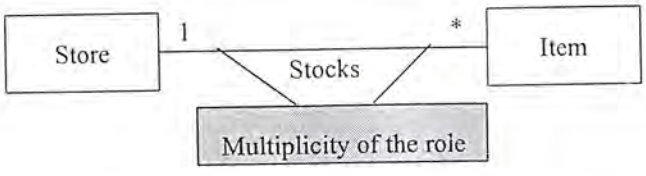
Roles

Roles อาจจะแสดงได้ถึงสิ่งต่อไปนี้

- name
- multiplicity expression
- navigability

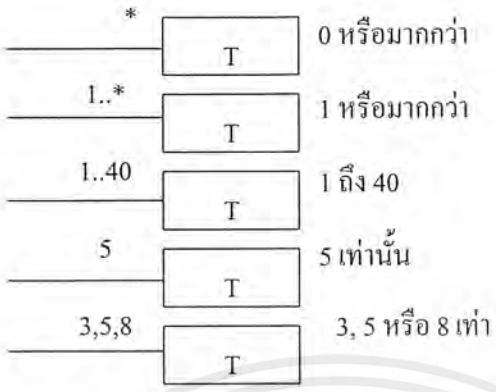
Multiplicity

Multiplicity แสดงถึงจำนวนของอินแสตนท์ของ type A ที่สามารถสัมพันธ์กับอินแสตนท์ของ type B ดังรูปที่ 2-6



รูปที่ 2-6 แสดง Multiplicity ระหว่าง association

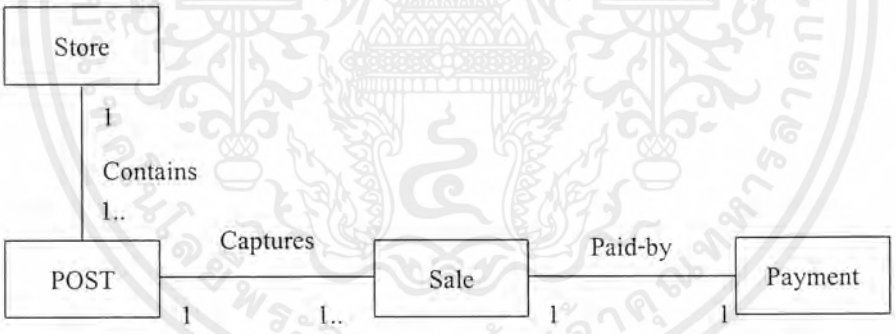
ประเภทของ Multiplicity



รูปที่ 2-7 แสดง Multiplicity Values

การให้ชื่อกับ associations

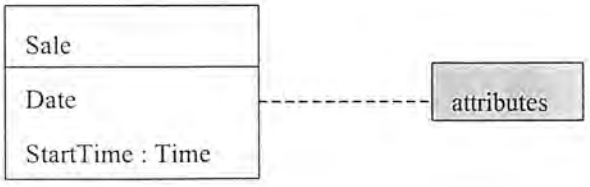
การให้ชื่อกับ association จะอยู่ในรูปแบบ *TypeName-VerbPhrase-TypeName* ซึ่งชื่อของ associate ควรจะเริ่มต้นด้วยตัวอักษรตัวใหญ่ ส่วน verb pharse ควรจะคั่นด้วยเครื่องหมาย – ดังรูปที่ 2-8



รูปที่ 2-8 แสดง Associate Name

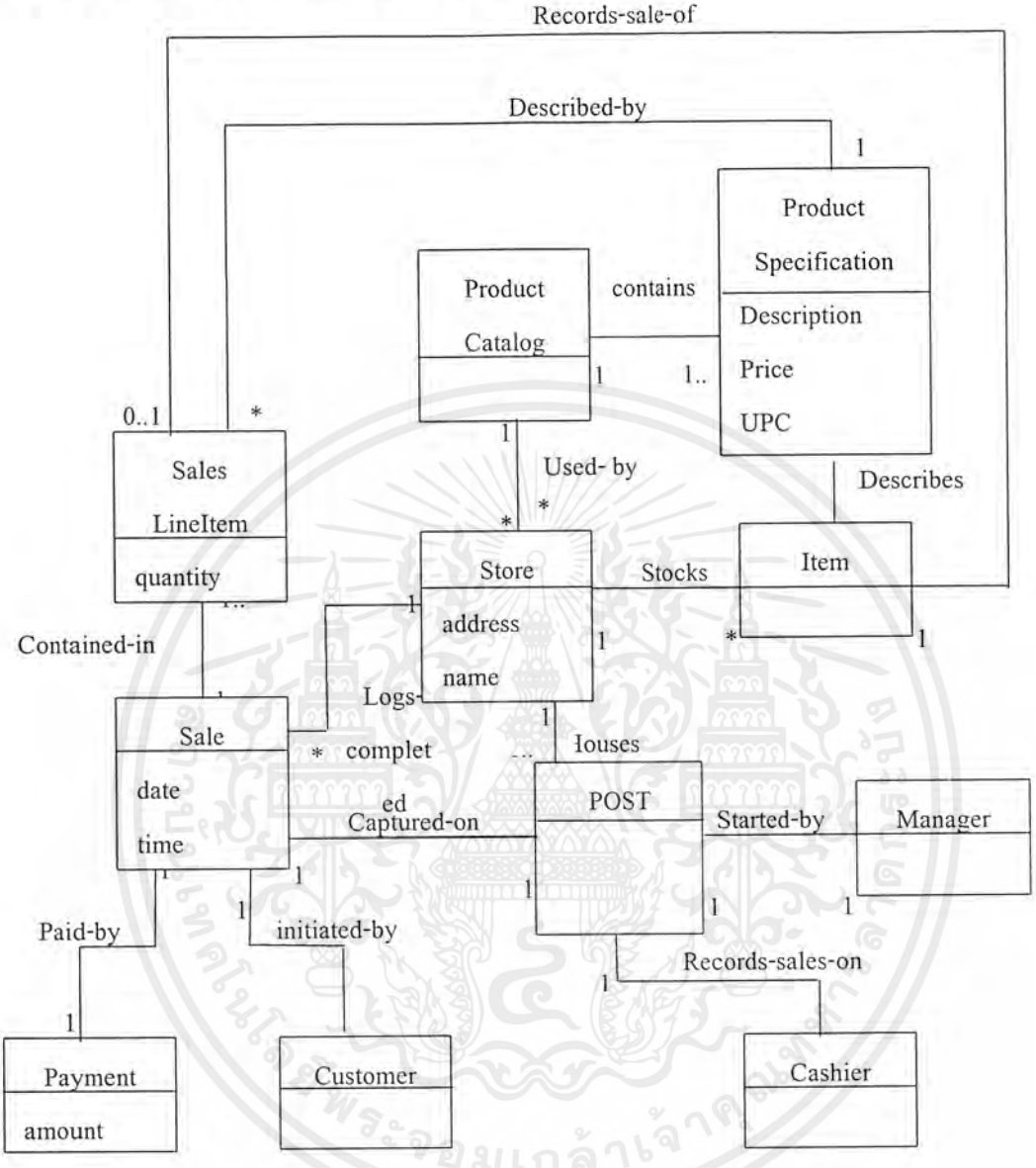
การเพิ่มแอตทริบิวต์ (attributes)

แอตทริบิวต์คือค่าของข้อมูลทางลอจิกของออปเจ็กต์ ซึ่งแสดงได้ดังรูปที่ 2-9



รูปที่ 2-9 แสดง concept และ attributes

ตัวอย่างของ Point-of-Sale Conceptual Model



รูปที่ 2-10 แสดง Conceptual Model สำหรับโดเมนของ Point-of-Sale

2.1.1.4 Refine Glossary

Glossary มีลักษณะคล้าย ๆ กับดาต้าดิกชันารี ซึ่งแสดงถึงรายละเอียดของหัวข้อต่าง ๆ เพื่อเก็บบันทึกกฎ ข้อบังคับต่าง ๆ ให้เข้าใจตรงกัน ซึ่งมีตัวอย่างดังต่อไปนี้

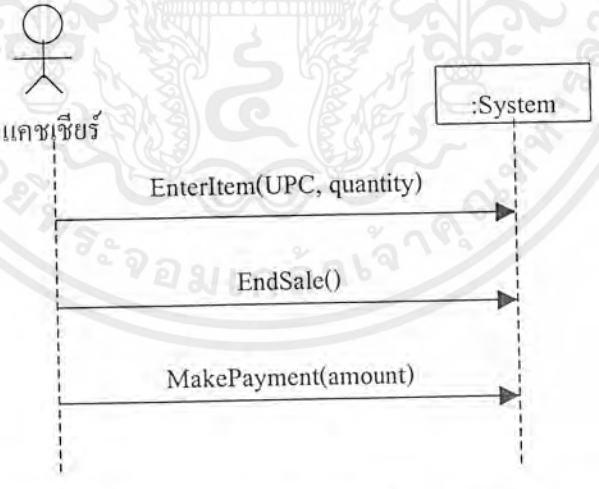
หัวข้อ	ประเภท	รายละเอียด
Buy Items	Use case	รายละเอียดของ process ในการซื้อของของลูกค้า
ProductSpecification.description : Text	Attribute	รายละเอียดของสิ่งของในร้าน ซึ่งสัมพันธ์กับ

		ProductSpecification
Item	Type	สิ่งของในร้าน
Payment	Type	การจ่ายเงินด้วยเงินสด
ProductSpecification.price : Quantity	Attribute	ราคาของสิ่งของซึ่งสัมพันธ์กับ *ProductSpecification*
SalesLineItem.quantity : Integer	Attribute	ปริมาณของสิ่งของแต่ละชนิด
Sale	Type	Transcation ของการขาย
SaleLineItem	Type	Line item ของการขาย
Store	Type	สถานที่เก็บสิ่งของ
Sale.total : Quantity	Attribute	ยอดขายทั้งหมด
Payment.amount : Quantity	Attribute	จำนวนเงินสด
ProductSpecification.upc : UPC	Attribute	รหัสสินค้า

ตารางที่ 2-2 แสดงรายละเอียดของหัวข้อต่างๆ

2.1.1.5 Define System Sequence Diagrams

System Sequence Diagram แสดงถึงเหตุการณ์ที่ actor ติดต่อกับระบบ ทำให้ทราบว่าระบบต้องทำอะไรบ้าง โดยมีตัวอย่างดังต่อไปนี้



รูปที่ 2-11 แสดง system sequence diagram ของ *Buy Items* use case

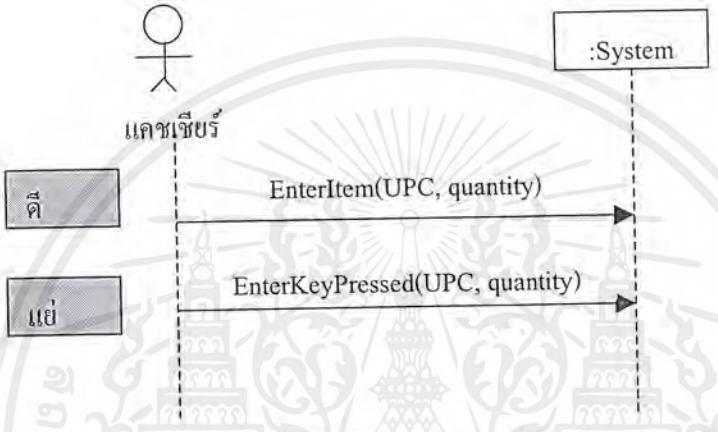
วิธีการสร้าง System Sequence Diagram

1. วาดเส้นแสดงถึงระบบเหมือนกล่องดำ (Black Box)
2. ระบุ actor ซึ่งทำหน้าที่โดยตรงกับระบบ โดยวาดเส้นสำหรับ actor ทุก ๆ คน
3. ระบุเหตุการณ์ซึ่งแต่ละ actor สร้างขึ้น ซึ่งแสดงไว้ในไดอะแกรม

4. นอกจากนั้นยังสามารถเพิ่มข้อความของ use case ไว้ในด้านซ้ายของไดอะแกรมได้เช่นกัน

การตั้งชื่อเหตุการณ์ของระบบและคำสั่ง

การตั้งชื่อเหตุการณ์ของระบบควรอยู่ในลักษณะที่เป็นความสนใจมากกว่าที่จะเป็นกลุ่มคำของอินพุทหรือในระดับอินเตอร์เฟส ดังนั้นเพื่อความกระจ่างชัดในการตั้งชื่อควรเริ่มจากการชื่อระบบตามด้วยกริยา ดังรูปที่ 2-12



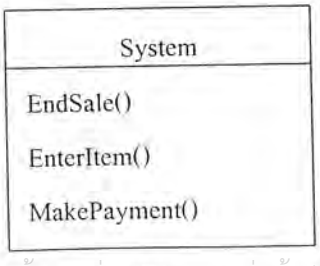
รูปที่ 2-12 แสดงการเลือกชื่อคำสั่งและเหตุการณ์ที่ในระดับ abstract

"endSale" เป็นชื่อที่ดีกว่า "EnterKeyPressed" เพราะว่าจะให้ความหมายของคำสั่งได้ดีกว่า

2.1.1.6 Define Operation Contracts

System Sequence Diagram แสดงถึงเหตุการณ์ที่ที่ actor ภายนอกสร้างขึ้น แต่ไม่ได้เกี่ยวข้องกับหน้าที่ต่าง ๆ ภายในคำสั่งของระบบ ทำให้ขาดรายละเอียดบางประการที่จะทำให้เข้าใจพฤติกรรมของระบบ

โดยทั่ว ๆ ไป contract เป็นเอกสารที่แสดงถึงคำสั่งพร้อมที่จะทำอะไร ซึ่งเป็นการเน้นย้ำว่า "อะไร" จะเกิดขึ้นมากกว่าที่จะอธิบายว่าจะต้องทำ "อย่างไร" ซึ่ง contract มักจะมีสถานะ pre และ post แสดงถึงการเปลี่ยนแปลง contract สามารถเขียนไว้เพื่อแสดงแต่ละ method ของคลาสนั้น ๆ ได้หรือแสดงเป็นคำสั่งของระบบอย่างกว้าง ๆ ก็ได้



รูปที่ 2-13 แสดงคำสั่งของระบบที่ต้องการรายละเอียดใน contract

ตัวอย่างของ contract ใน Buy Items Use Case

1. Contract for enterItem

Contract

Name :	enterItem(upc : number, quantity : integer)
Responsibilities :	เพิ่มเรคอร์ดสินค้า แสดงรายละเอียดสินค้าและราคา
Type :	ระบบ
Cross References :	หน้าที่ : R1.1, R1.3, R1.9
Notes :	ต้องติดต่อกับฐานข้อมูล
Exceptions :	ถ้า upc ไม่ถูกต้องจะเกิด error
Output :	
Pre-conditions :	ระบบต้องรู้จัก UPC
Post-conditions :	

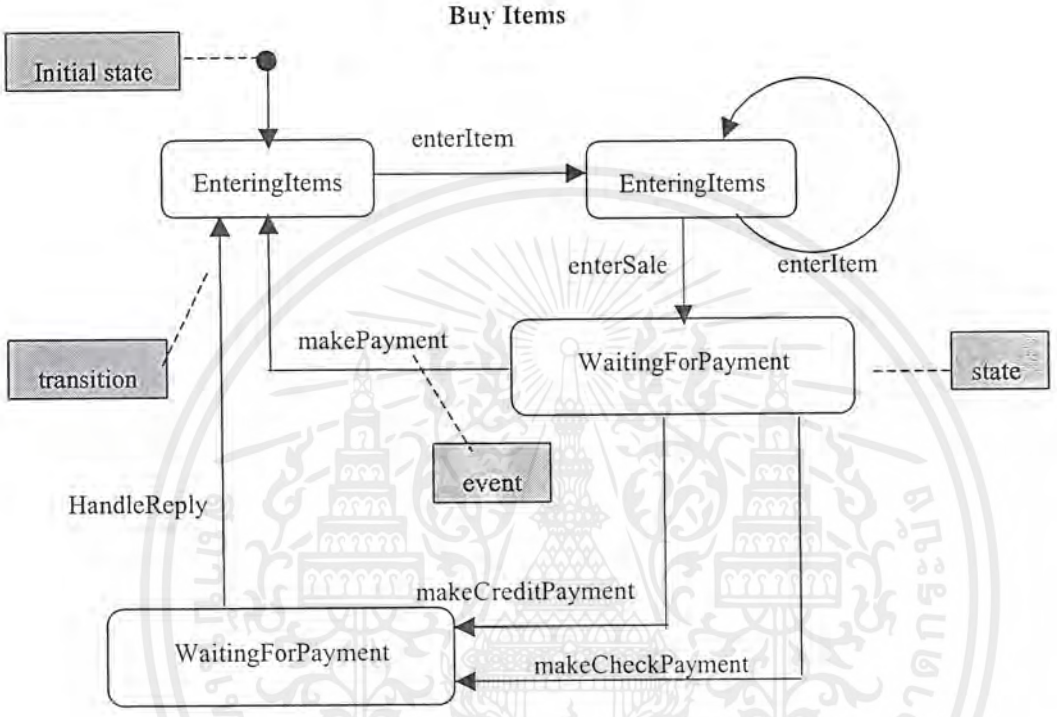
- ถ้ามีการขายเกิดขึ้น *a Sale* จะถูกสร้างขึ้น (instance creation)
- ถ้ามีการขายเกิดขึ้น จะต้องติดต่อกับ POST (association formed)
- A SalesLineItem ถูกสร้างขึ้น (instance creation)
- The SalesLineItem ถูกสร้างขึ้นพร้อมกับ the Sale (association formed)
- SalesLineItem.quantity ถูกเซ็ตให้เป็น quantity (attribute modification)
- The SalesLineItem ถูกรวมเข้ากับ a ProductSpecification ซึ่งขึ้นอยู่กับ UPC (association formed)

วิธีการสร้าง contract

1. ระบุคำสั่งของระบบจาก sequence diagram
2. สร้าง contract สำหรับแต่ละคำสั่งของระบบ
3. เริ่มต้นโดยการเขียนหน้าที่ เพื่ออธิบายวัตถุประสงค์ของคำสั่ง
4. หลังจากนั้นสร้าง Post-conditions section อธิบายการเปลี่ยนแปลงของแต่ละสถานะใน Conceptual Model
5. อธิบาย post-conditions ตามประเภทต่างได้ดังนี้
   - การสร้างและทำลายอินสแตนท์
   - การแก้ไขเปลี่ยนแปลงแอตทริบิวต์
   - การสร้างและทำลาย association

2.1.1.7 Define State Diagrams

State Diagram แสดงถึงเหตุการณ์และสถานะต่าง ๆ ของออปเจ็กต์ ซึ่งช่วยอธิบายวัฏจักรของออปเจ็กต์ที่เกิดขึ้น เช่นในตัวอย่างของ Buy Items จะไม่เกิดเหตุการณ์ MakeCreditPayment ขึ้นจนกว่าจะมีเหตุการณ์ endSale เกิดขึ้นก่อน ดังตัวอย่างต่อไปนี้



รูปที่ 2-14 แสดง State Diagram ของ Buy Items

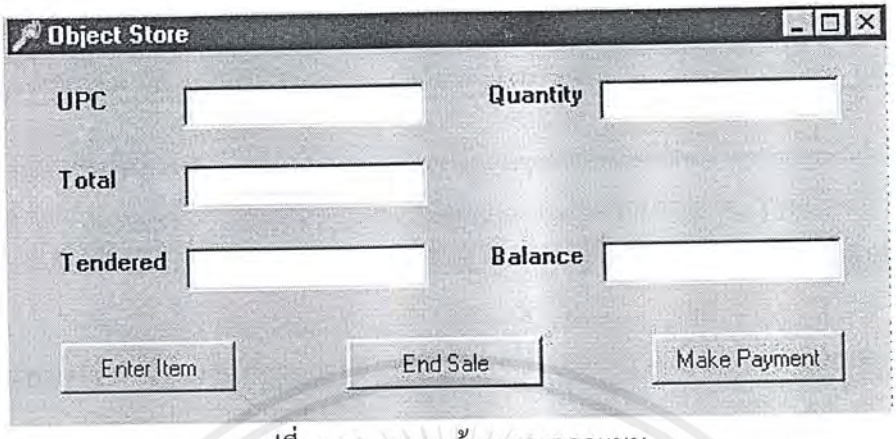
2.1.2 การออกแบบเชิงวัตถุ (Object-Oriented Design) มีทั้งสิ้น 6 ขั้นตอนตามลำดับดังนี้

2.1.2.1 Define Real Use Cases

Real use case อธิบายถึง use case diagram ที่นำมาใช้งานจริง ๆ ซึ่งต้องมีส่วนรับอินพุท และส่วนแสดงผลเอาท์พุทรวมถึงการใช้งานโดยอาจจะมีรายละเอียดในส่วนของ user interface ด้วยเช่นกัน ดังตัวอย่างต่อไปนี้

Use case :	**Buy Items – version 1 (Cash only)**
Actor :	ลูกค้า, แคชเชียร์
Purpose :	ทำการขายและจ่ายเงินด้วยเงินสด
Overview :	ลูกค้าเข้ามาซื้อสินค้า แคชเชียร์บันทึกรายการซื้อและรับเงินด้วยเงินสด สิ้นสุดที่ลูกค้าออกจากร้านพร้อมด้วยสินค้า
Type	primary and real

Cross References : ตาม functions : R1.1, R1.2, R1.3, R1.7, R1.9, R2.1



รูปที่ 2-15 แสดงหน้าจอการออกแบบ

เหตุการณ์ต่าง ๆ ที่เกิดขึ้น

Actor Action	System Response
1. ลูกค้ามาซื้อสินค้าที่เครื่อง POST	
2. สินค้าแต่ละชิ้น แคชเชียร์ต้องประทับ Universal Product Code (UPC) ใน A ของ Window-1 ถ้ามีสินค้าหลายชิ้น จะต้องระบุปริมาณใน E แล้วกด B	3. เพิ่มข้อมูลสินค้าเข้าไปยัง transaction ของการขาย ราคาของสินค้าปัจจุบันแสดงไว้ในช่อง B และ F
4. สิ้นสุดการขายโดยการกด I	5. คำนวณและแสดงยอดรวมทั้งหมดใน C
	6. …

ตารางที่ 2-3 แสดงเหตุการณ์ต่างๆที่เกิดขึ้น

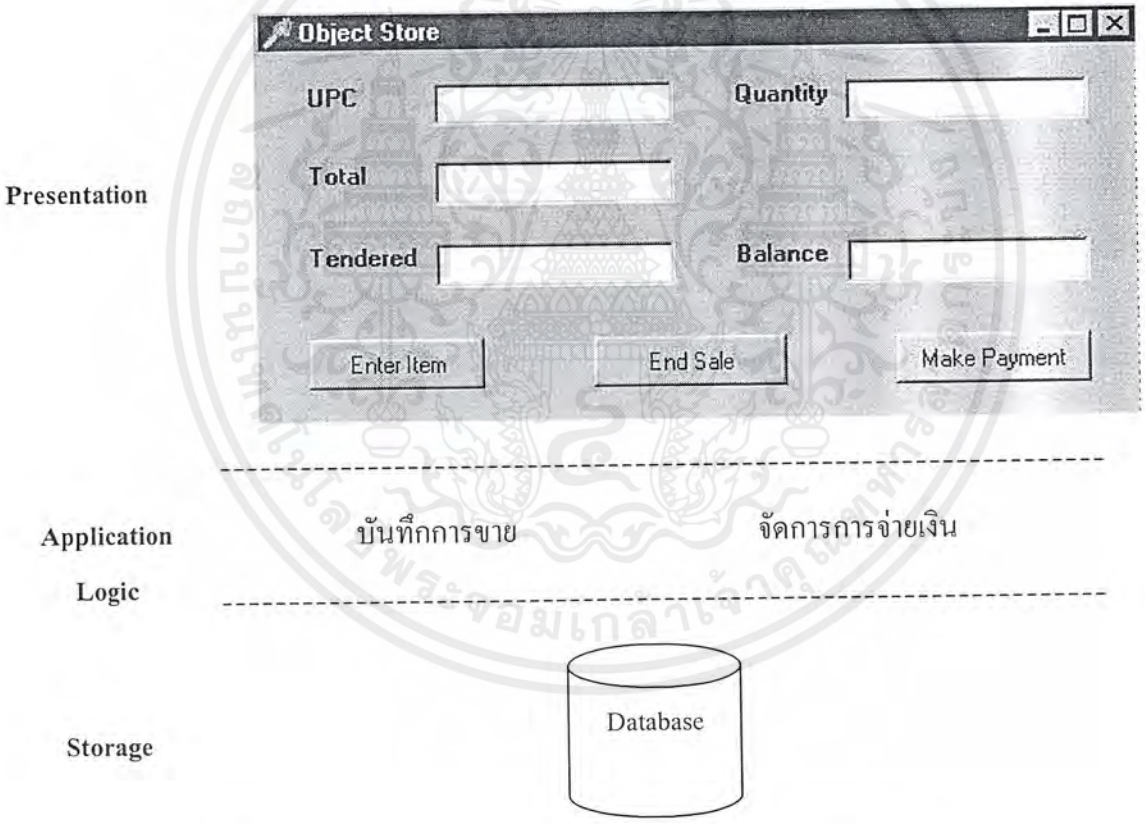
2.1.2.2 Define Reports, UI and Storyboards

เป็นส่วนเพิ่มความสมบูรณ์ให้กับ use case ซึ่งอาจจะแสดงผลเป็นรายงานหรือในรูปแบบอื่นก็ตาม

2.1.2.3 Refine System Architecture

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

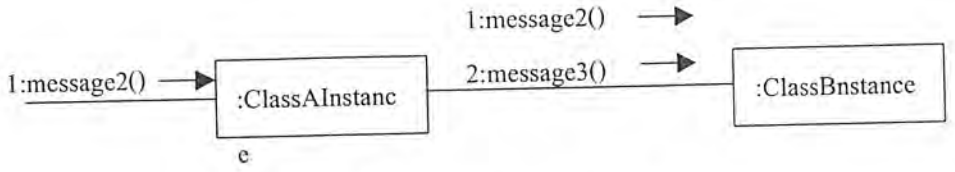
1. Presentation :– client
2. Application Logic :- application server
3. Storage :- database server



รูปที่ 2-16 แสดงมุมมองของระบบในลักษณะของสถาปัตยกรรม 3-เทียร์

2.1.2.4 Define Interaction Diagrams

Interaction Diagram ก็คือ Collaboration Diagram โดยแสดงถึงการเชื่อมต่อของออปเจ็กต์ในรูปแบบของกราฟฟิกหรือเน็ตเวิร์ก

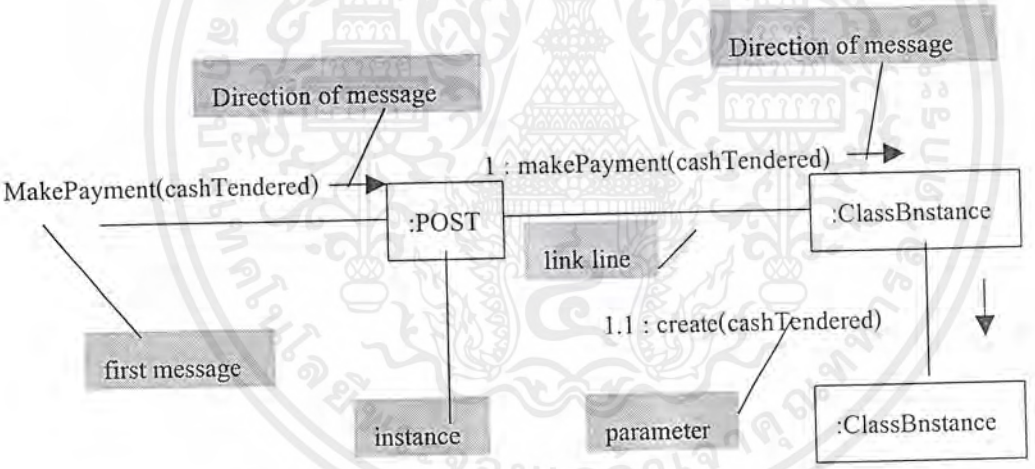


รูปที่ 2-17 แสดง Collaboration Diagram

วิธีการสร้าง 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

ตัวอย่างของ Collaboration Diagram : makePayment



รูปที่ 2-18 แสดง make payment collaboration diagram

2.1.2.5 Define Design Class Diagrams

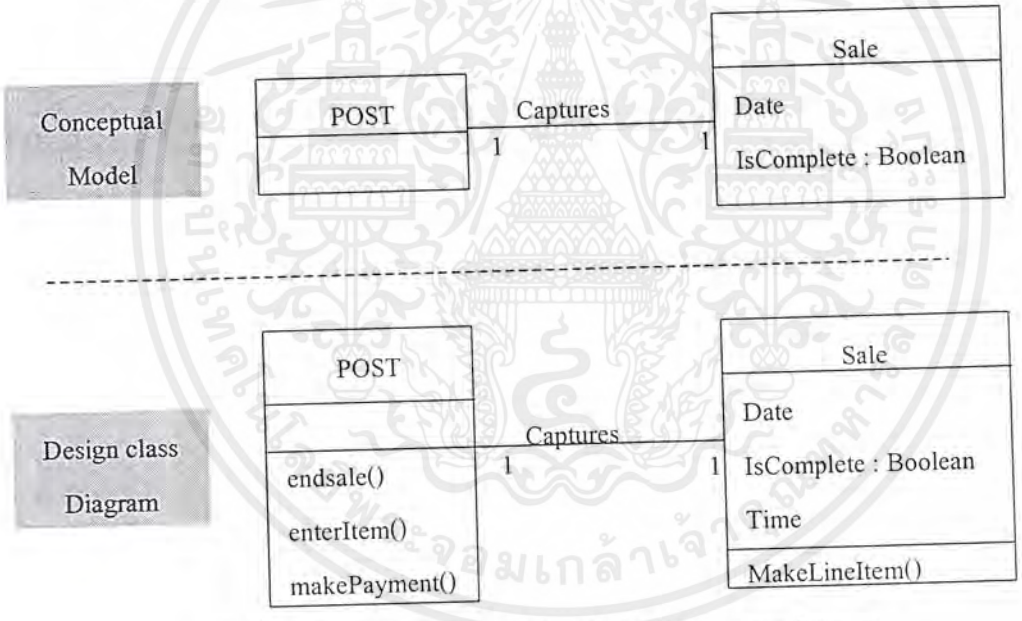
design class diagram แสดงคุณลักษณะของ software classes และ interface ของแอพพลิเคชัน ซึ่งประกอบด้วยส่วนต่าง ๆ ดังต่อไปนี้

- classes, associations และ attributes
- interface with operations และ constants
- methods
- attribute type information
- navigability
- dependencies

วิธีการสร้าง Design Class Diagram

1. ระบุคลาสทั้งหมดที่จะต้องใช้ในการสร้างแอพพลิเคชัน ซึ่งจะได้มาจากส่วนของ interaction diagram
2. วาด class diagram
3. สร้างแอตทริบิวต์ขึ้นมาใหม่จาก associated concepts ใน conceptual model
4. เพิ่มชื่อของเมธอดโดยวิเคราะห์จาก interaction diagrams
5. เพิ่มชนิดของข้อมูลให้กับแอตทริบิวต์และเมธอด
6. เพิ่มความสัมพันธ์ให้กับแอตทริบิวต์ที่ต้องการให้ชัดเจน
7. สร้างลูกศรแสดงความสัมพันธ์ในการบ่งบอกทิศทางของแอตทริบิวต์ให้ชัดเจน
8. สร้างเส้นที่แสดงความสัมพันธ์แบบขึ้นต่อกันเพิ่มบ่งบอกส่วนที่ไม่ใช่แอตทริบิวต์

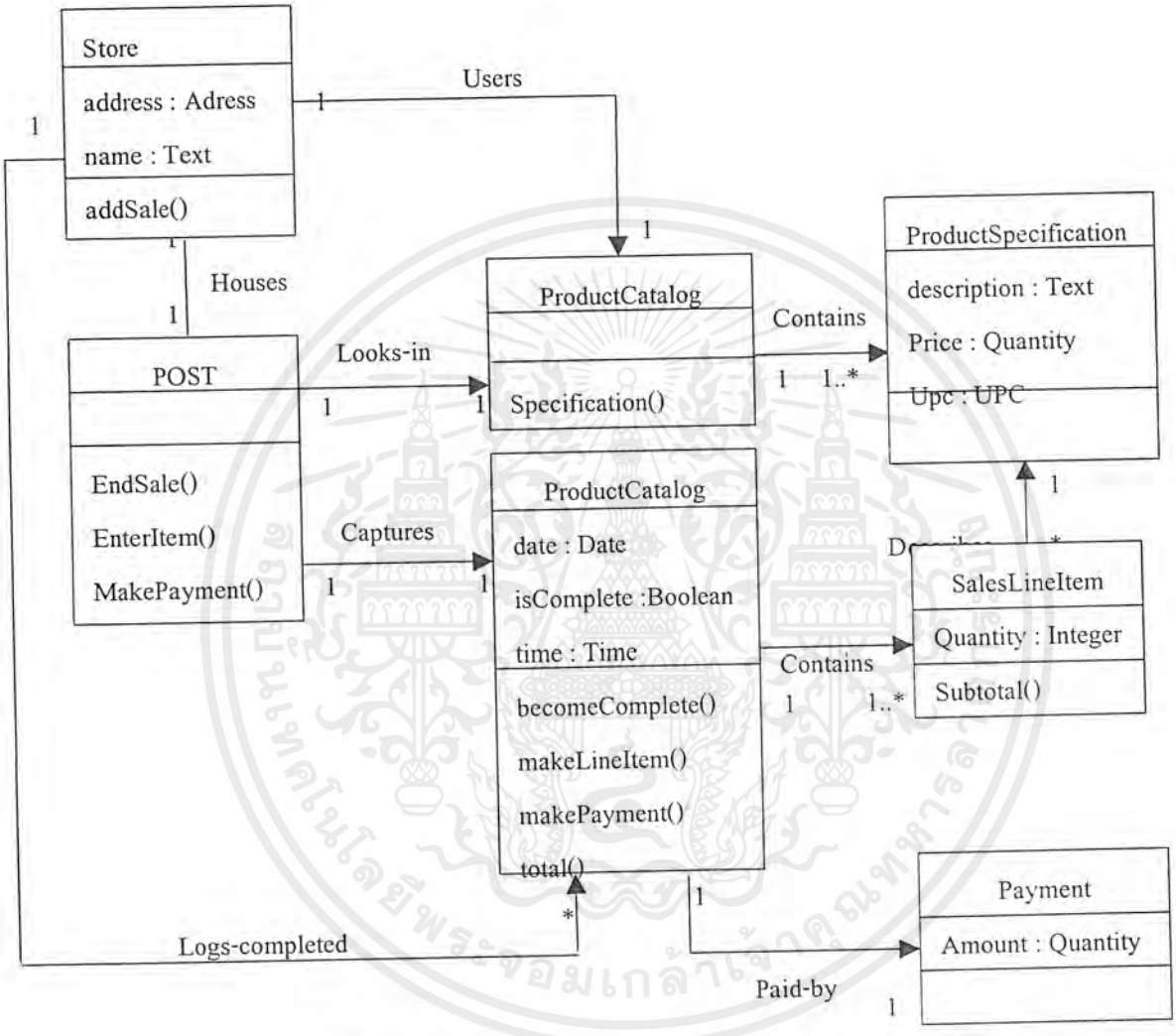
เปรียบเทียบ Conceptual Model และ Design class diagram



รูปที่ 2-19 แสดง Conceptual Model และ Design class diagram

ตัวอย่างของ Design Class Diagram

จากตัวอย่างของ Collaboration Diagram สามารถนำมาสร้าง Design Class Diagram ได้ดังต่อไปนี้



รูปที่ 2-20 แสดง Design class diagram ของ POST

2.1.2.6 Define Database Schema

Database schema เป็นส่วนของการออกแบบฐานข้อมูลที่สัมพันธ์กับระบบงาน เพื่อจัดเก็บข้อมูลให้อยู่ในรูปแบบที่ต้องการ

2.2 การโปรแกรมเชิงวัตถุ (Object-Orientd Programming : OOP)

OOP เป็นเทคนิคใหม่ของการเขียนโปรแกรมที่เราสามารถโมเดลออปเจ็กต์บนโลกแห่งความเป็นจริงได้ โครงสร้างพื้นฐานของ OOP ได้แก่คลาสของออปเจ็กต์ และออปเจ็กต์ที่เป็นสมาชิกของคลาสดังกล่าว

ความคิดเบื้องต้นของออปเจ็กต์ เมื่อเราจินตนาการถึงออปเจ็กต์ในโลกแห่งความจริง เราลองนึกถึงตัวต่อเลโก (lego) ที่เป็นพลาสติกชิ้นเล็กๆ มีสีสันและรูปแบบต่างๆกัน ใช้นำมาต่อกันเป็นรูปร่างต่างๆ เช่นเครื่องบิน รถไฟ เป็นต้น แต่ละชิ้นส่วนของเลโกนั้นเสมือนเป็นออปเจ็กต์ เมื่อนำมาต่อเข้าด้วยกันก็จะทำให้ได้ตัวต่อรูปร่างต่างๆ ที่ได้ออกแบบไว้ เช่นเดียวกับการนำออปเจ็กต์แต่ละชิ้นส่วนมารวมเข้าด้วยกัน ก็จะได้แอพพลิเคชันขนาดใหญ่ที่สามารถทำงานได้ตามที่ต้องการ

การนำชิ้นเลโกแต่ละชิ้นส่วนมาประกอบเข้าด้วยกันนั้น ถ้าเราต้องการเปลี่ยนแปลงแก้ไขโครงสร้างหรือรูปแบบของตัวต่อตรงไหน เราสามารถเปลี่ยนแปลงชิ้นส่วนแต่ละชิ้นได้โดยง่าย และไม่มีผลกระทบต่อชิ้นส่วนอื่น เช่นกัน ในการสร้างโปรแกรมเชิงออปเจ็กต์ การพัฒนา แก้ไขปรับปรุง สามารถทำได้ง่ายมาก โดยไม่มีผลกระทบต่อโครงสร้างส่วนใหญ่ของโปรแกรม ในขณะที่การเขียนโปรแกรมแบบโพรซีเดอรัล (procedural) ที่ประกอบด้วยคำสั่งมากมายหลายพันบรรทัดนั้น เมื่อต้องการแก้ไขปรับปรุง หรือพัฒนาโปรแกรม จะทำได้ยากมาก

หลักการทำงานพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ จะมีการกำหนดสิ่งต่างๆที่อยู่ภายในขอบเขตของระบบงานเป็นออปเจ็กต์ ซึ่งในแต่ละออปเจ็กต์ประกอบด้วยแอตทริบิวต์ (Attribute) และเมทธอด (Method) รวมกันอยู่ภายใน (การเอนแคปซูเลชัน) เมื่อออปเจ็กต์เรียกใช้เมทธอดหรือการสั่งให้เกิดการกระทำตามที่ได้กำหนดไว้ในเมทธอด จะมีการส่งเมสเซส (Message) ไปยังออปเจ็กต์นั้นๆ เพื่อดำเนินการต่อไป

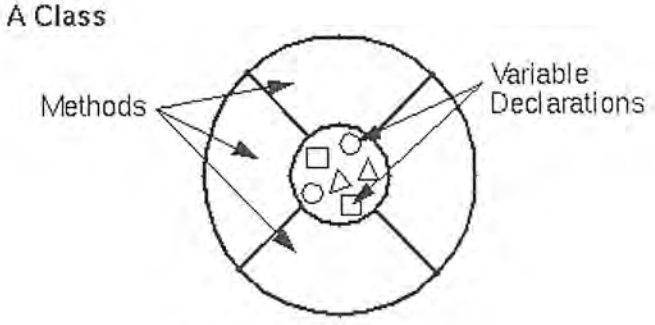
จุดมุ่งหมายหลักของการพัฒนาระบบเชิงวัตถุ ได้แก่

1. ทำให้การพัฒนาระบบใช้เวลาสั้นลง ต้นทุนต่ำ โดยความสามารถในการนำคลาสกลับมาใช้งานใหม่ได้อีก
2. ต้นทุนและค่าใช้จ่ายในการบำรุงรักษาและแก้ไขระบบต่ำลง เพราะเราสามารถหาสิ่งที่ต้องการเปลี่ยนแปลงในระบบได้ง่าย และการเปลี่ยนแปลงจะไม่มีผลต่อส่วนภายนอกคลาส

2.2.1 คำศัพท์ที่เกี่ยวข้องกับการพัฒนาโปรแกรมเชิงวัตถุ

2.2.1.1 คลาส (Class)

คลาส คือ กลุ่มของออปเจ็กต์ที่แบ่งตามลักษณะเฉพาะและการใช้งาน หรือออปเจ็กต์ที่มีคุณสมบัติพื้นฐานเหมือนกัน ข้อมูลเชิงออปเจ็กต์ที่เก็บอยู่ในคลาส เรียกว่า อินสแตนท์ (Instance) ของคลาส โดยที่อินสแตนท์จะเป็นส่วนขยายของคลาสและเก็บอยู่ในส่วนของฐานข้อมูล ดังนั้นข้อมูลต่างๆที่เป็นตัวกำหนดคลาสจะถูกเข้าถึงผ่านทางออปเจ็กต์ของอินสแตนท์และเมทธอดของคลาส โดยคลาสจะห่อหุ้มคุณลักษณะทั้งหมดของออปเจ็กต์ต่างๆและกำหนดชนิดของข้อมูลที่บรรจุอยู่ในออปเจ็กต์ พร้อมกับกำหนดเมทธอดต่างๆ สำหรับการเข้าถึงข้อมูลไว้ด้วย



รูปที่ 2-21 ลักษณะของคลาส

2.2.1.2 ออปเจ็กต์ (Object)

ออปเจ็กต์เป็นโมดูลของข้อมูล(Data) และโค้ดที่สามารถถูกรวมเข้ากับออปเจ็กต์อื่นๆ เพื่อรวมกันสร้างซอฟต์แวร์ขนาดใหญ่ขึ้น ภายในออปเจ็กต์จะประกอบด้วย 2 ส่วนได้แก่ ส่วน "แอตทริบิวต์" (Attribute) ซึ่งเป็นส่วนของคุณสมบัติต่างๆ ของออปเจ็กต์นั้น และส่วนโพรซีเยอร์หรือฟังก์ชัน ที่เป็นส่วนของการกระทำ ซึ่งเราเรียกว่า "เมทธอด" (Method)

2.2.1.3 แอตทริบิวต์ (Attribute)

แอตทริบิวต์ คือค่าของข้อมูลที่เป็นส่วนบอกคุณสมบัติของออปเจ็กต์ที่อยู่ในคลาส เช่น ออปเจ็กต์รถยนต์มีแอตทริบิวต์ สี, ยี่ห้อ , แรงม้า เป็นต้น

2.2.1.4 เมทธอด (Method)

เมทธอดคือการที่ออปเจ็กต์สามารถทำสิ่งต่างๆ ให้ตัวมันเอง หรือมีสิ่งต่างๆ ที่ทำให้มัน เช่นออปเจ็กต์ รถยนต์ อาจมีเมทธอดได้แก่ สตาร์ตเครื่อง , เร่งความเร็ว , เปลี่ยนเกียร์ เป็นต้น

ออปเจ็กต์จะมีการดำเนินการกับข้อมูล โดยใช้โพรซีเยอร์และฟังก์ชัน ที่เราเรียกกันรวมกันว่าเมทธอด การทำงานของเมทธอดแบ่งออกเป็น 3ส่วน ได้แก่

1. ส่วนประกาศเมทธอด (Declaration) อยู่ในส่วนการประกาศคลาส เช่น
   procedure Isphere (radius :Real; CenterV,CenterH :Integer);
2. ส่วนโครงสร้างของเมทธอด (Body) เช่น

```
procedure Tsphere.Isphere;
begin
        fRadius := Radius;
        fCenterV := CenterV;
        fcenterH := CenterH;
end;
```

3. ส่วนที่เรียกใช้จากโค้ด
   อยู่ที่ตำแหน่งที่เราต้องการเรียกใช้ในโค้ด โดยใช้ในรูป

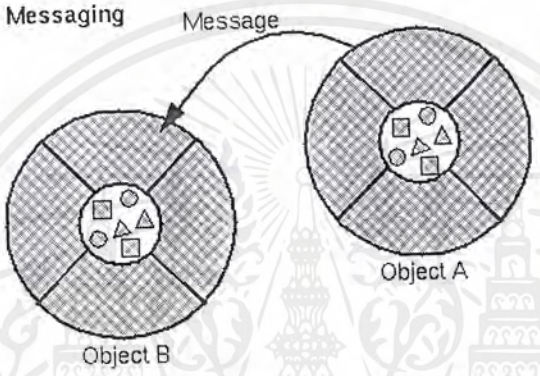
ชื่อออปเจ็กต์.ชื่อเมทธอด(พารามิเตอร์) เช่น aSphere.fCenter.h :=50;

2.2.1.5 เมสเซจ (Message)

ออปเจ็กต์จะติดต่อกับออปเจ็กต์อื่น โดยผ่านทางเมสเซจ การส่งเมสเซจเป็นการเรียกใช้เมทธอดที่ต้องการมาทำงาน แล้วส่งผลลัพธ์ไปยังผู้ส่งเมสเซจ

2.2.1.6 ตัวแปร self

ใช้เมื่อเราต้องการอ้างถึงออปเจ็กต์ปัจจุบันในคลาสนั้น ซึ่งมีประโยชน์อย่างมากเมื่อเราต้องการอ้างถึงออปเจ็กต์ในขณะนั้นเพื่อเรียกใช้เมทธอดหรือข้อมูลของออปเจ็กต์นั้น



รูปที่ 2-22 การส่งเมสเซสระหว่างออปเจ็กต์

ตัวอย่างโค้ด

```
unit Unit1;

interface

type
  TBook = class
  public
    id : integer;
    title : string;
    publisher : string;
    typebook : string;
    borrowed_id : integer;
    procedure setInit(i : integer; t : string; p : string; tb : string; b : integer);
  end;
```

```
TTextBook = class(TBook)
public
    author : string;
    edition : integer;
    borrowDay : integer;
    constructor create; //
end;

TMagazine = class(TBook)
public
    issue : integer;
    year : integer;
    borrowDay : integer;
    constructor create;
end;

implementation
constructor TTextBook.create;
begin
      borrowDay := 7;
end;

constructor TMagazine.create;
begin
      borrowDay := 1;
end;
procedure TBook.setInit(i : integer; t : string; p : string; tb : string; b : integer);
begin
    self.id := i;
    self.title := t;
    self.publisher := p;
    self.typebook := tb;
    self.borrowed_id := b;
end;
```

end.

2.2.1.7 คอนสตรักเตอร์ (Constructor)

ในการประกาศออปเจ็กต์ เช่น var Myobject : TObject นั้น เป็นเพียงการกำหนดว่าตัวแปร ? MyObject มีอยู่ในคลาส TObject เท่านั้น แต่ยังไม่ได้มีการสร้างออปเจ็กต์ MyObject ขึ้นมา เราจึงต้องใช้คอนสตรักเตอร์ ในการจองหน่วยความจำเพื่อรองรับออปเจ็กต์ที่สร้างขึ้น

คอนสตรักเตอร์เป็นเมทธอดพิเศษทำหน้าที่สร้างและกำหนดค่าเริ่มต้นให้แก่ออปเจ็กต์ การประกาศใช้คอนสตรักเตอร์คล้ายกับการประกาศโพรซีเยอร์ แต่จะขึ้นต้นด้วย constructor แทน เช่น

```
constructor Create;
```

โดยทั่วไปแล้วคลาสหนึ่งคลาสจะมีหนึ่งคอนสตรักเตอร์ เมื่อต้องการสร้างออปเจ็กต์ขึ้นมาใหม่ จะต้องเรียกคอนสตรักเตอร์ของคลาสที่ออปเจ็กต์นั้นเป็นสมาชิกอยู่ เช่น

```
MyObject := TMyClass.Create;
```

การประกาศเช่นนี้จะทำให้เกิดการจองพื้นที่สำหรับออปเจ็กต์ใหม่นี้บนฮีป (heap) แล้วคอนสตรักเตอร์จะส่งคืนค่ากลับในรูปออปเจ็กต์ที่สร้างขึ้นมาใหม่

2.2.1.8 ดีสตรักเตอร์ (Destructor)

ดีสตรักเตอร์ใช้คืนหน่วยความจำในส่วนที่ไม่ต้องการใช้เป็นเวลานานๆ เพื่อให้ออปเจ็กต์อื่นสามารถใช้งานหน่วยความจำนั้นในส่วนนั้นได้ เช่น

```
MyObject :=TMyClass.Free;
```

การประกาศเช่นนี้จะทำให้เกิดการคืนพื้นที่ของหน่วยความจำนี้ เพื่อให้ออปเจ็กต์อื่นๆ เข้ามาใช้งานแทนได้

2.2.2 คุณสมบัติที่สำคัญของการพัฒนาโปรแกรมเชิงวัตถุ

คุณสมบัติที่สำคัญของการพัฒนาโปรแกรม 3 ประการ ได้แก่

1. คุณสมบัติเอนแคปซูเลชัน (Encapsulation)
2. คุณสมบัติอินเฮอริแตนซ์ (Inheritance)
3. คุณสมบัติโพลีมอร์ฟิซึม (Polymorphism)

2.2.2.1 คุณสมบัติเอนแคปซูเลชัน

คุณสมบัติเอนแคปซูเลชันได้แก่ โครงสร้างของโปรแกรมต้องจบในตัวเอง, ทำงานเป็นอิสระและทำงานได้ด้วยตนเอง แต่ในขณะเดียวกันก็ต้องสามารถแยกส่วนออกมาและทำงานได้เป็นอิสระเช่นกัน มีการรวมแอตทริบิวต์และเมทธอดเพื่อให้เป็นข้อมูลเชิงออปเจ็กต์ ซึ่งจะแสดงให้เห็นถึงความสัมพันธ์ระหว่างแอตทริบิวต์ภายในและเมทธอดของออปเจ็กต์ด้วย โดยออปเจ็กต์หนึ่งจะมีคุณสมบัติเหมือนสิ่งของอย่างหนึ่งคือ มีทั้งส่วนประกอบและการทำงาน ซึ่งสามารถใช้งานได้ทันที

ข้อดีของการการเอนแคปซูเลชัน ได้แก่เมื่อส่วนใดส่วนหนึ่งของระบบถูกเปลี่ยนแปลง จะมีการพิจารณาแก้ไขข้อมูลเพียงส่วนเดียว โดยไม่มีผลกระทบกับส่วนอื่นๆ ในระบบ

วิธีการในการป้องกันนั้นการเข้าถึงข้อมูลภายในออปเจ็กต์จากฟังก์ชันภายนอก (Data hiding) จะอยู่ที่ชนิดของการประกาศข้อมูลภายในคลาส ซึ่งมีอยู่ 3 ชนิด ได้แก่

1. **private** การประกาศข้อมูลแบบนี้จะมีผลทำให้ไม่มีคำสั่งใดที่จะอ้างถึงข้อมูลภายในคลาสนั้นได้เลย นอกจากเมทธอดภายในคลาสนั้นเอง ซึ่งตามปกติแล้วจะมีอย่างน้อยหนึ่งเมทธอด ที่ไม่ได้ประกาศให้เป็นข้อมูลชนิดนี้ เพื่อที่จะได้ใช้ในการเข้าถึงข้อมูลที่ประกาศอยู่ภายในส่วน private ของคลาสได้

2. **public** การประกาศข้อมูลแบบนี้จะมีผลทำให้สามารถอ้างอิงข้อมูลภายในคลาสได้โดยอิสระ

3. **protected** การประกาศข้อมูลแบบนี้จะมีผลให้มีเฉพาะเมทธอดในคลาสสืบทอดเท่านั้นที่จะอ้างอิงข้อมูลที่ประกาศภายในส่วนนี้ได้

2.2.2.2 คุณสมบัติอินเฮอริแตนซ์

คุณสมบัติอินเฮอริแตนซ์ได้แก่การที่คลาสเก่าที่มีอยู่แล้วเป็นต้นแบบให้กับคลาสใหม่ที่จะถูกสร้างขึ้น เกิดการถ่ายทอดคุณสมบัติจากคลาสพ่อ (base class) ไปสู่คลาสลูก (derived class) นอกจากคลาสลูกสืบทอดคุณสมบัติทุกอย่างมาจากคลาสพ่อแล้ว คลาสลูกยังสามารถที่จะเพิ่มคุณสมบัติให้กับตัวเองได้อีกด้วย เช่น การเพิ่มแอตทริบิวต์หรือเมทธอดให้กับตัวเอง คลาสลูกสามารถที่จะใช้ชื่อของเมทธอดเหมือนกันกับชื่อเมทธอดของคลาสพ่อได้ ทั้งนี้เพื่อเพิ่มความสามารถให้แก่เมทธอดเดิมโดยใช้หลักการของเวอร์ชวล / โอเวอร์ราย (virtual/override) การสืบทอดแบบนี้นับว่าเป็นคุณสมบัติเด่นของการเขียนโปรแกรมแบบโอโอพี เพราะทำให้เราสามารถใช้โค้ดเดิมได้โดยไม่ต้องมีการเขียนโปรแกรมที่เคยเขียนไว้แล้วขึ้นมาใหม่อีก เพราะคอมไพเลอร์ (Compiler) จะค้นหาโค้ดเองเมื่อทำการคอมไพล์โปรแกรม ผู้ใช้สามารถกำหนดต้นแบบของคลาส (Abstract classes) เพื่อให้ผู้ใช้คนอื่นๆ นำไปพัฒนาต่อโดยการเพิ่มรายละเอียดเฉพาะของตัวเองเข้าไป

รูปที่ 2-23 แสดงการสืบทอดจากคลาสพื้นฐานหนึ่งคลาส

- คลาสพื้นฐาน (Base class) บางครั้งเรียกคลาสพ่อ
- คลาสสืบทอด (derived class) บางครั้งเรียกคลาสลูก

Override

การโอเวอร์รายเมทธอด คลาสสืบทอด(คลาสลูก) จะโอเวอร์รายเมทธอดจากคลาสพ่อมาเพื่อเพิ่มเติมเมทธอดให้แตกต่างจากคลาสพ่อ ทำได้โดยการเติมคีย์เวิร์ด "override" หลังการประกาศเมทธอดนั้น เช่น procedure Flash; override;

Virtual Method

การประกาศเวอร์ชวลเมทธอด ทำเมื่อเราต้องการสร้างรูปแบบการเข้าใช้งานโค้ดที่แตกต่างกัน เมื่อเรียกใช้เมทธอดชื่อเดียวกัน โดยใช้หลักการของ Late Binding เมื่อเราประกาศเวอร์ชวลเมทธอดให้กับเมทธอดในคลาสพ่อแล้ว การประกาศเมทธอดของคลาสลูกที่มีชื่อเดียวกับเมทธอดของคลาสพ่อ สามารถทำได้โดยใช้การโอเวอร์รายเมทธอด ซึ่งทำโดยใช้คีย์เวิร์ด **override** ต่อท้ายการประกาศเมทธอดนั้น หรือไม่ใช้การโอเวอร์ราย ก็ได้

ตัวอย่างโค้ด

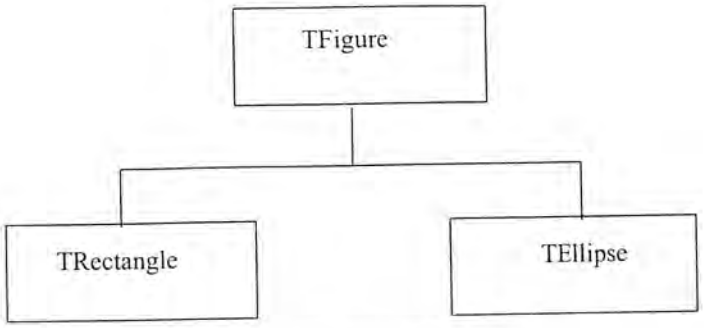
```
type
  TFigure = class
  public
    fig:integer;
    constructor create;
    procedure Draw; virtual;
  end;
  TRectangle = class(TFigure)
```

```
public
 procedure Draw;
end;
TEllipse = class(TFigure)
public
 procedure Draw; override;
end;

implementation
{TFigure Method}
constructor Tfigure.create;
begin
end;
procedure TFigure.Draw;
begin
   fig:=10;
end;

{TRectangle Method}
procedure TRectangle.Draw;
begin
   fig:=20;
end;
{TEllipse Method}
procedure TEllipse.Draw;
begin
   fig:=30;
end;
```

เขียนคลาสไดอะแกรมได้ดังนี้



รูปที่ 2-24 แสดงคลาสไดอะแกรม

ตัวอย่างโค้ดส่วนการเรียกใช้

```
uses Unit2;
var
 Figure: array[1..20] of TFigure;
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
 Figure[1] := TRectangle.Create;
 Figure[1].Draw;    // calls TFigure.Draw
 edit1.text:=inttostr(Figure[1].fig);          //edit1.text = '10'
 Figure[2] := TEllipse.Create;
 Figure[2].Draw;   // calls TEllipse.Draw
 edit2.text:=inttostr(Figure[2].fig);          //edit2.text = '30'
end;
```

จะสังเกตได้ว่าโพรซีเยอร์ Draw ของคลาส Trectangle ไม่ได้มีการประกาศโอเวอร์รายเมทธอด ในขณะที่โพรซีเยอร์ Draw ของคลาส Tellispe มีการประกาศโอเวอร์ราย การเรียกใช้ของทั้งสองเมทธอดนี้จะให้ผลต่างกัน จะเห็นได้จากส่วนการเรียกใช้ เมื่อออปเจ็กต์ของคลาส Trectangle เรียกใช้เมทธอด Draw จะเป็นการเรียกใช้เมทธอด Draw ของคลาสพ่อ (คลาส Tfigure) ไม่ได้เรียกใช้เมทธอด Draw ของคลาส Trectangle ในขณะที่ถ้าออปเจ็กต์ของคลาส Tellipse เรียกใช้เมทธอด Draw จะเป็นการเรียกใช้เมทธอด Draw ของคลาส Tellipse เอง เนื่องจากได้มีการประกาศคีย์เวิร์ด override ไว้หลังการประกาศเมทธอด Draw ของคลาส Tellipse

คีย์เวิร์ด inherited

การใช้คีย์เวิร์ด inherited ใช้เมื่อเวลาเราต้องการที่จะเรียกใช้เมทธอดของคลาสพ่อ โดยประกาศไว้ในส่วน body ของคลาสนั้น

```
type
 TFigure = class
 public
        fig:integer;
        constructor create;
        procedure Calc;
 end;
 TRectangle = class(TFigure)
 public
        procedure play;
end;
implementation
{TFigure Method}
constructor TFigure.create;
begin

end;
procedure Tfigure.calc;
begin
        fig:=100;
end;
{Trectangle Method}
procedure Trectangle.play;
begin
   inherited calc;
   ...
 end;
```

เมทธอด play ของคลาส Trectangle ซึ่งเป็นคลาสลูกของคลาส Tfigure จะเรียกใช้เมทธอด Calc ของคลาส Tfigure ได้ โดยผ่านคีย์เวิร์ด inherited เมื่อออปเจ็กต์ของคลาส Trectangle เรียกเมทธอด play จะส่งผลให้มีการเข้าไปทำงานในเมทธอด Tfigure.calc ซึ่งเป็นเมทธอดที่อยู่ในคลาสพ่อด้วย เป็นผลให้แอตทริบิวต์ fig ถูกกำหนดค่าเป็น 100

เออร์ลี ไบน์ดิง (Early Binding)

ในขณะที่คอมไพเลอร์เรียกใช้โพรซีเยอร์หรือฟังก์ชัน คอมไพเลอร์จะรู้ตำแหน่งที่แน่นอนของโค้ดแล้ว เนื่องจากโค้ดได้ผ่านการคอมไพล์และมีการจองพื้นที่ไว้สำหรับโพรซีเยอร์หรือฟังก์ชันแล้ว เมื่อมีการเรียกใช้โพรซีเยอร์หรือฟังก์ชันเกิดขึ้น คอมไพเลอร์จะหาตำแหน่งแอดเดรสของโพรซีเยอร์หรือฟังก์ชันนั้นแล้วข้ามไปทำงานต่อที่แอดเดรสดังกล่าวเลย การทำงานรูปแบบนี้เรียกว่า "Compile time Binding" หรือ "Early Binding"

เลต ไบน์ดิง (Late Binding)

การทำงานแบบ Early Binding นั้นไม่สามารถใช้งานได้กับการทำโพลีมอร์ฟิซึม ดังนั้นจึงมีวิธีการอื่น เรียกว่า Late Binding หรือ Runtime Binding ผลของเลตไบน์ดิงก์ทำให้โปรแกรมไม่รู้ล่วงหน้าว่าจะเรียกเมทธอดนั้นได้จากที่ไหน จะรู้ในตอนรันโปรแกรมเท่านั้นว่าจะเรียกใช้เมทธอดใดให้เหมาะสม แสดงว่าการทำงานแบบเลตไบน์ดิงก์จะไม่ผูกติดกับเมทธอดใดๆ ไว้ในลักษณะแน่นอนตายตัวตั้งแต่ตอนคอมไพล์แต่จะมีการเรียกเมทธอดต่างๆ เปลี่ยนแปลงไปตามความเหมาะสมในตอนรันโปรแกรม

2.2.2.3 คุณสมบัติโพลีมอร์ฟิซึม

คุณสมบัติโพลีมอร์ฟิซึมได้แก่ความสามารถที่จะใช้ได้ทั่วไปคุณสมบัติที่เมื่อออปเจ็กต์ต่างๆ ได้รับคำสั่งเดียวกันจากโปรแกรมแล้วแต่ละออปเจ็กต์จะทำงานตามแบบของตัวเอง ซึ่งทำให้ได้ผลลัพธ์แตกต่างกัน ซึ่งเป็นคลาสที่ได้รับการถ่ายทอดคุณสมบัติจากคลาสเดียวกัน

การทำโพลีมอร์ฟิซึมนั้นเป็นคุณสมบัติการเรียกใช้เมทธอดที่ใช้ชื่อเดียวกันโดยมีการเลือกใช้การทำงานของเมทธอด ขึ้นกับตำแหน่งของออปเจ็กต์ที่มีการเรียกใช้เมทธอดนั้น คลาสหลักจะต้องประกาศเมทธอดที่จะใช้เป็นเวอร์ชวลเมทธอด การใช้เมทธอดเดียวกันจะทำงานต่างกันตามชนิดของคลาส

คุณสมบัติโพลีมอร์ฟิซึมแบ่งได้เป็น 3 ประเภท ได้แก่

1. Inherited Polymorphism เกิดขึ้นเมื่อออปเจ็กต์ของคลาสที่ต่างกัน เมทธอดที่มีชื่อเดียวกันเนื่องจากคลาสเหล่านั้นสืบทอดมาจากคลาสเดียวกัน เช่นรถยนต์ และรถจักรยานยนต์สืบทอดมาจากคลาสพาหนะเหมือนกัน เพราะฉะนั้น ทั้งรถยนต์และรถจักรยานยนต์จึงวิ่งได้, เบรกได้ เป็นต้น
2. Independent Polymorphism เกิดขึ้นเมื่อออปเจ็กต์ของคลาสที่ต่างกัน ใช้เมทธอดที่มีชื่อเดียวกัน แต่วิธีการนั้นมีลักษณะการทำงานต่างกัน เช่นรถยนต์วิ่งได้โดยการกดคันเร่งแต่รถจักรยานยนต์วิ่งได้โดยการบิด
3. Coincidential Polymorphism เกิดขึ้นเมื่อออปเจ็กต์ของคลาสที่ต่างกัน ใช้เมทธอดที่มีชื่อเดียวกัน แต่วิธีการนั้นไม่มีความสัมพันธ์กันเลย

ตัวอย่างโค้ด

```
type
  Tstudent = class(Tmember)
  private
     day : integer;
  public
     procedure borrow(id:string);
  end;
  Tofficer = class(Tmember)
  private
     day  : integer;
  public
     procedure borrow(id:string);
  end;
implementation
  procedure Tstudent.borrow(id:string);
  begin
        day:=7;
  end;
  procedure Tofficer.borrow(id:string);
  begin
        day:=14;
  end;
```

จากตัวอย่างโค้ด เมทธอด borrow เป็นเมทธอดของทั้งคลาส Tstudent และคลาส Tofficer ในส่วนการเรียกใช้ แบ่งออกเป็น

1. ถ้าออปเจ็กต์ student1 เป็นสมาชิกของคลาส Tstudent แล้ว ประโยค student1.borrow จะเรียกใช้งานโพรซีเยอร์ Tstudent.borrow(id:string); ซึ่งจะทำให้แอตทริบิวต์ day มีค่าเท่ากับ 7
2. ถ้าออปเจ็กต์ officer1 เป็นสมาชิกของคลาส Tofficer แล้ว ประโยค officer1.borrow จะเรียกใช้งานโพรซีเยอร์ Tofficer.borrow(id:string); ซึ่งจะทำให้แอตทริบิวต์ day มีค่าเท่ากับ 14

บทที่ 3
ทฤษฎีและหลักการสถาปัตยกรรม 3-เทียร์

3.1 สถาปัตยกรรม 3-เทียร์

ในอดีตการพัฒนาโปรแกรมจะอยู่ในลักษณะสถาปัตยกรรม 2-เทียร์ ที่ประกอบด้วยการทำงาน 2 ส่วนได้แก่

1. ไคลเอนต์
2. ดาต้าเบสเซิร์ฟเวอร์

ซึ่งไคลเอนต์แต่ละตัวสามารถติดต่อกับข้อมูลในดาต้าเบสเซิร์ฟเวอร์ได้โดยตรงทำให้เกิดปัญหาตามมาอย่างมากทั้งในเรื่องของประสิทธิภาพการทำงานและเรื่องความปลอดภัยของข้อมูล

สถาปัตยกรรม 3-เทียร์ จึงถือกำเนิดขึ้นโดยมีการแยกการทำงานของระบบออกเป็น 3 ส่วน

1. **ไคลเอนต์** เป็นลักษณะธินไคลเอนต์ (Thin Client) เป็นเครื่องคอมพิวเตอร์ธรรมดา ไม่จำเป็นต้องลงโปรแกรมใดๆ เพิ่มเติม
2. **แอพพลิเคชันเซิร์ฟเวอร์** เป็นส่วนที่เพิ่มขึ้นมาจากสถาปัตยกรรม 2-เทียร์ จะดูแลเกี่ยวกับแอพพลิเคชันที่รันอยู่บนเครื่อง รวมถึงทำหน้าที่เป็นตัวกลางติดต่อระหว่างไคลเอนต์และดาต้าเบสเซิร์ฟเวอร์ ดังนั้นจึงต้องมีโปรแกรมที่ใช้ในการเชื่อมต่อกับฐานข้อมูล เช่น SQLNet ของ Oracle เป็นต้น
3. **ดาต้าเบสเซิร์ฟเวอร์** เป็นเซิร์ฟเวอร์ที่ตั้งขึ้นเพื่อการเก็บรักษาข้อมูลเพียงอย่างเดียว และทำการติดต่อกับแอพพลิเคชันเซิร์ฟเวอร์เท่านั้น ไม่สามารถติดต่อกับไคลเอนต์ได้โดยตรง

3.2 สภาวะแวดล้อมของระบบ

3.2.1 ไคลเอนต์

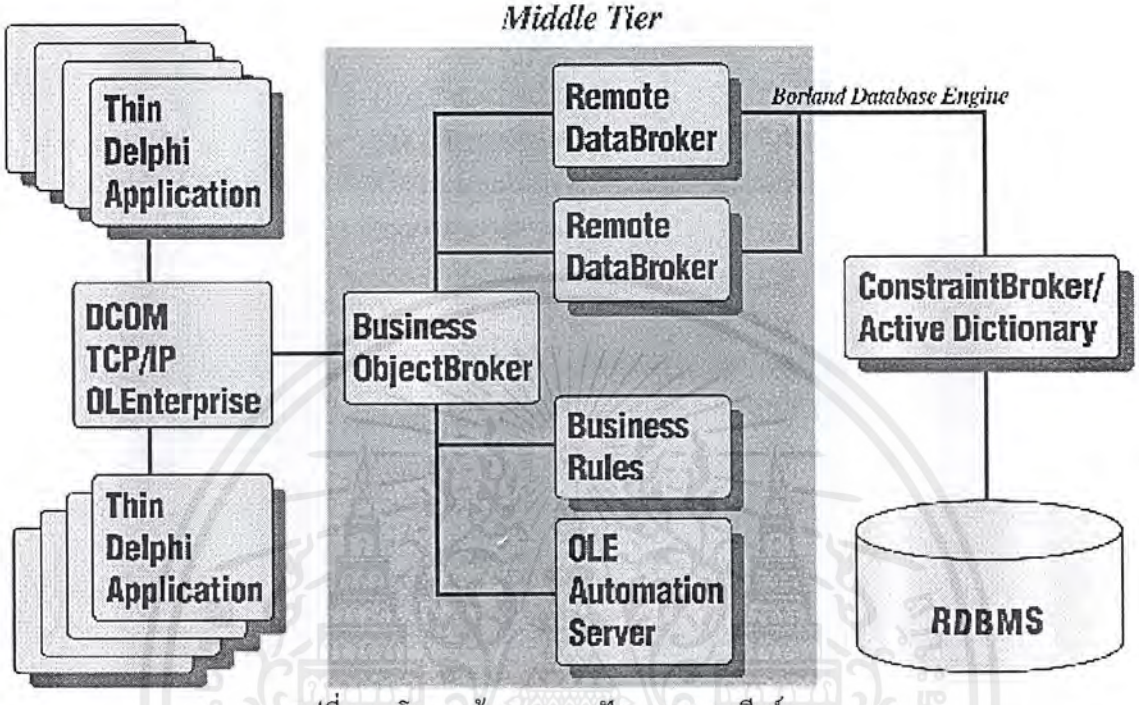
- เครื่องคอมพิวเตอร์ รันระบบปฏิบัติการ Windows98

3.2.2 แอพพลิเคชันเซิร์ฟเวอร์

- เครื่องคอมพิวเตอร์ รันระบบปฏิบัติการ WindowsNT Server 4.0

3.2.3 ดาต้าเบสเซิร์ฟเวอร์

- เครื่องคอมพิวเตอร์ รันระบบปฏิบัติการ WindowsNT Server 4.0



รูปที่ 3-1 โครงสร้างของสถาปัตยกรรม 3-เทียร์

3.3 ข้อเปรียบเทียบระหว่างสถาปัตยกรรม 3-เทียร์ และ 2-เทียร์

1. **การรวมการประมวลผลต่างๆ ไว้ในที่ที่เดียวคือที่เครื่องแอพพลิเคชันเซิร์ฟเวอร์** ในสถาปัตยกรรม 3-เทียร์นั้น แอพพลิเคชันที่รันบนไคลเอนต์แต่ละเครื่องจะเข้ามาทำงานบนแอพพลิเคชันเซิร์ฟเวอร์เพียงเครื่องเดียว ทำให้ช่วยลดปัญหาความซ้ำซ้อนของข้อมูล (Redundancy)
2. **แอพพลิเคชันบนไคลเอนต์มีขนาดเล็ก** ในสถาปัตยกรรม 3-เทียร์ โค้ดส่วนการประมวลผลจะถูกเขียนอยู่ที่แอพพลิเคชันเซิร์ฟเวอร์ นอกจากนั้นไคลเอนต์ยังไม่ต้องสนใจวิธีการในส่วนการติดตั้งและซอฟต์แวร์ในการเชื่อมต่อกับฐานข้อมูล เช่น BDE (Borland Database Engine) , SQLNet ของ Oracle เป็นต้น เหมือนในสถาปัตยกรรม 2-เทียร์ นอกจากนั้นเครื่องที่นำมาทำเป็นไคลเอนต์ไม่จำเป็นต้องมีสเป็กสูง เพราะไคลเอนต์จะรันเพียงแอพพลิเคชันที่เป็นไฟล์สกุล exe เท่านั้น แค่มีการเชื่อมต่อกับระบบเครือข่ายก็พอ
3. **การทำงานแบบกระจาย (Distributed)** การกระจายแอพพลิเคชันไปยังไคลเอนต์แต่ละตัว ในสถาปัตยกรรม 3-เทียร์ ทำให้การทำงานของเซิร์ฟเวอร์มีประสิทธิภาพสูงขึ้น
4. **ง่ายต่อการควบคุมและดูแลความปลอดภัยของข้อมูล** ในสถาปัตยกรรม 3-เทียร์นั้น การเข้าใช้งานข้อมูลที่ดาต้าเบสเซิร์ฟเวอร์ ของไคลเอนต์แต่ละตัวต้องเรียกใช้และกระทำผ่านแอพพลิเคชันเซิร์ฟเวอร์ ในขณะที่สถาปัตยกรรม 2-เทียร์ ไคลเอนต์แต่ละตัวสามารถติดต่อกับดาต้าเบสเซิร์ฟเวอร์ได้โดยตรง

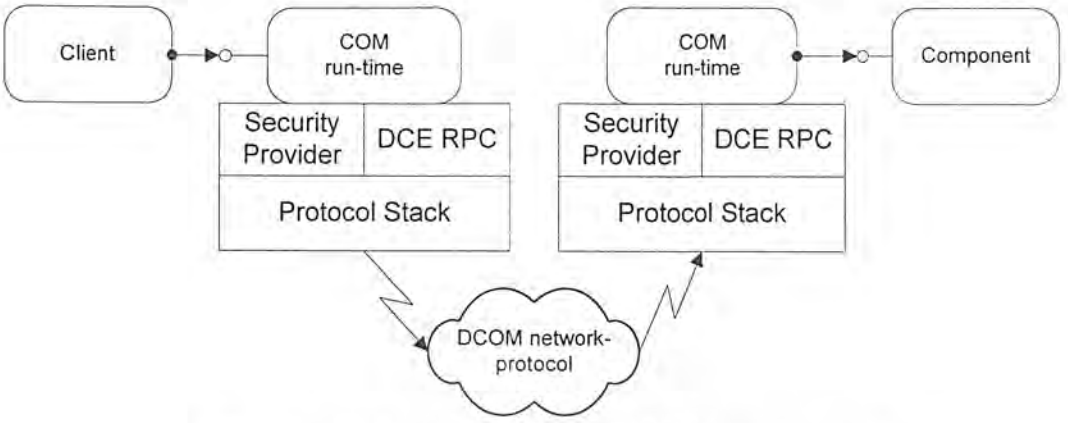
3.4 Distributed Component Object Model (DCOM)

Borland MIDAS (Multi-tier Distributed Applications Services Suite) ได้เพิ่มเติมเทคโนโลยีใหม่เพื่อพัฒนาแอพพลิเคชันแบบมัลติเทียร์ (Multi-tier) ซึ่งจะทำให้แอพพลิเคชันที่รันบนไคลเอนต์มีขนาดเล็กมาก (Thin Client) ไคลเอนต์แต่ละตัวไม่จำเป็นต้องมีการติดตั้ง (Install) , คอนฟิกูเรชันและดูแลรักษา (Maintenance) ซอฟต์แวร์การเชื่อมต่อกับฐานข้อมูลเลย แอพพลิเคชันแบบมัลติเทียร์นั้นจะลดขั้นตอนดังกล่าวของไคลเอนต์ลง และนำไปกระทำบนแอพพลิเคชันเซิร์ฟเวอร์เพียงเครื่องเดียวแทน โดยอาศัยการทำงานของ DCOM (Distributed Component Object Model)

ปัจจุบันการเจริญเติบโตของระบบเครือข่ายคอมพิวเตอร์ทำให้หลักการของ Distributed Computing ได้รับการกล่าวถึงเป็นอย่างมาก โดยเฉพาะอย่างยิ่ง DCOM (Distributed Component Object Model) ของบริษัทไมโครซอฟต์ และ CORBA (Common Object Request Broker Architecture) ของกลุ่มบริษัท OMG (Object Management Group)

DCOM เป็นเทคโนโลยีที่พัฒนาโดยบริษัทไมโครซอฟต์เพียงบริษัทเดียว ใช้เทคโนโลยีของ COM เป็นคอมโพเนนต์หลัก เพียงแต่ว่าเป็น COM ที่มีการเชื่อมต่อกันได้โดยมีพื้นฐานอยู่บน Open Software Foundation's DCE-RPC โดย DCOM จะเป็นตัวกำหนดว่าจะให้คอมโพเนนต์ทำการติดต่อกับไคลเอนต์อย่างไร ในปัจจุบันระบบปฏิบัติการทั่วไป โพรเซสแต่ละโพรเซสจะถูกห่อหุ้มไว้ทำให้ไคลเอนต์ที่ต้องการติดต่อกับคอมโพเนนต์ของโพรเซสอื่นไม่สามารถติดต่อกันได้โดยตรง แต่ต้องติดต่อผ่าน inter-process communication ซึ่งก็คือ DCOM นั้นเอง กล่าวได้ว่า DCOM เป็นโพรโตคอลที่ทำให้ซอฟต์แวร์คอมโพเนนต์สามารถติดต่อกันได้โดยตรงบนเครือข่าย

DCOM ยอมให้แอพพลิเคชันบนไคลเอนต์ ทำงานร่วมกับเซิร์ฟเวอร์ได้โดยใช้ COM API ที่พัฒนาโดยไมโครซอฟต์และสามารถใช้ได้กับเทคโนโลยี Dynamic Data Exchange ซึ่งทำให้นักพัฒนามีมาตรฐานแบบเปิดสำหรับการโต้ตอบของแอพพลิเคชันภายใต้ Windows แต่ DCOM ก็ยังมีจุดอ่อนได้แก่ความพยายามที่จะทำให้เครือข่ายปรากฏเสมือนเป็นสภาพแวดล้อมท้องถิ่น DCOM สร้างขึ้นสำหรับคอมพิวเตอร์ในรูปแบบเดิมที่มีผู้ใช้คนเดียวเป็นผู้ควบคุมจากศูนย์กลางที่ใช้ร่วมกันได้ และ DCOM จะสร้างและทำลายออปเจ็กต์แบบยืดหยุ่น ออปเจ็กต์ทุกตัวเป็นชุดของตัวชี้ไปยังตัวติดต่อซึ่งออปเจ็กต์หนึ่งเสนอแก่ออปเจ็กต์อื่น ถ้าไม่มีสิ่งใดต้องการใช้ออปเจ็กต์นั้น ออปเจ็กต์นั้นก็จะหายไป แต่ตราบใดที่ยังมีสิ่งที่ต้องการใช้งานออปเจ็กต์นั้น ออปเจ็กต์นั้นก็ยังคงอยู่ ในขณะที่เซิร์ฟเวอร์ของ DCOM มีไคลเอนต์กำลังทำงานอยู่ เซิร์ฟเวอร์นั้นจะต้องเก็บไคลเอนต์ไว้ในหน่วยความจำตลอดเวลา



รูปที่ 3-2 การติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์โดยผ่าน DCOM

DCOM เป็นส่วนขยายมาจาก Component Object Model (COM) ซึ่งเป็นตัวกำหนดว่าจะให้คอมโพเนนต์ทำการติดต่อกับไคลเอนต์อย่างไร ในปัจจุบันระบบปฏิบัติการทั่วไป โพรเซสแต่ละโพรเซสจะถูกห่อหุ้มไว้ทำให้ไคลเอนต์ที่ต้องการติดต่อกับคอมโพเนนต์ของโพรเซสอื่นไม่สามารถติดต่อกันได้โดยตรง แต่ต้องติดต่อผ่าน inter-process communication ซึ่งก็คือ DCOM นั้นเอง DCOM เป็นโพรโตคอลที่ทำให้ซอฟต์แวร์คอมโพเนนต์สามารถติดต่อกันได้โดยตรงบนเครือข่าย

DCOM จะถูกติดตั้งให้โดยอัตโนมัติสำหรับระบบปฏิบัติการวินโดว์เอ็นที 4.0 และวินโดว์ 98 ในขณะที่ระบบปฏิบัติการวินโดว์ 95 ยังต้องอาศัยการดาวน์โหลด DCOM จากเว็บไซต์ของไมโครซอฟต์ http://www.microsoft.com/oledev/ เพื่อติดตั้งเพิ่มเติมในภายหลัง

ข้อได้เปรียบที่สำคัญของ DCOM คือระบบปฏิบัติการที่ใช้ส่วนใหญ่บนเครื่องคอมพิวเตอร์ทั่วไปจะเป็นวินโดว์ ซึ่งใช้ COM เป็นมาตรฐานอยู่แล้ว ดังนั้นจึงมีฐานการสนับสนุนและได้รับความนิยมมากกว่า CORBA ที่ผู้ใช้ส่วนใหญ่จะเป็นนักพัฒนาโปรแกรมในองค์กรขนาดใหญ่

3.5 การสร้างแอพพลิเคชันโดยใช้สถาปัตยกรรม 3-เทียร์

ในการสร้างแอพพลิเคชันแบบ 3 เทียร์นั้น เครื่องที่ทำหน้าที่เป็นไคลเอนต์ต้องการเพียงไฟล์ DBCLIENT.DLL (ซึ่งมีขนาดเพียง 140 กิโลไบต์เท่านั้น) และคอมโพเนนต์ของเดลไฟล์ 2 ตัวได้แก่ TClientDataset และ TDCOMConnection เพื่อใช้ในการเชื่อมต่อกับแอพพลิเคชันเซิร์ฟเวอร์ที่จะเชื่อมต่อไปยังดาต้าเบสเซิร์ฟเวอร์ Oracle ต่อไป

TDCOMConnectionเป็นคอมโพเนนต์ที่ใช้สร้างและรักษาการเชื่อมต่อจากไคลเอนต์กับแอพพลิเคชันเซิร์ฟเวอร์ เมื่อเริ่มสร้างการเชื่อมต่อ ไคลเอนต์จะทำการรีจีสเตอร์ไคลเอนต์ดาต้าเซ็ตทั้งหมดที่มี เข้ากับแอพพลิเคชันเซิร์ฟเวอร์ ซึ่งไคลเอนต์ดาต้าเซ็ตทั้งหมดจะติดต่อกับคอมโพเนนต์ TProvider ที่อยู่ในฝั่งแอพพลิเคชันเซิร์ฟเวอร์

3.5.1 การสร้างและกำหนดค่าให้กับแอพพลิเคชันเซิร์ฟเวอร์ สำหรับการเชื่อมต่อแบบ 3-tier

1. หลังจากได้สร้างแอพพลิเคชันบนเดลไฟล์ขึ้นมาแล้วบันทึกไฟล์แล้ว (สมมติเป็นไฟล์ unit1.pas) เรา จะสร้างตัวRemote Data Module ซึ่งจะเป็นตัวที่ให้ไคลเอนต์เชื่อมต่อเข้ามาใช้งาน โดยมีขั้นตอนดังนี้

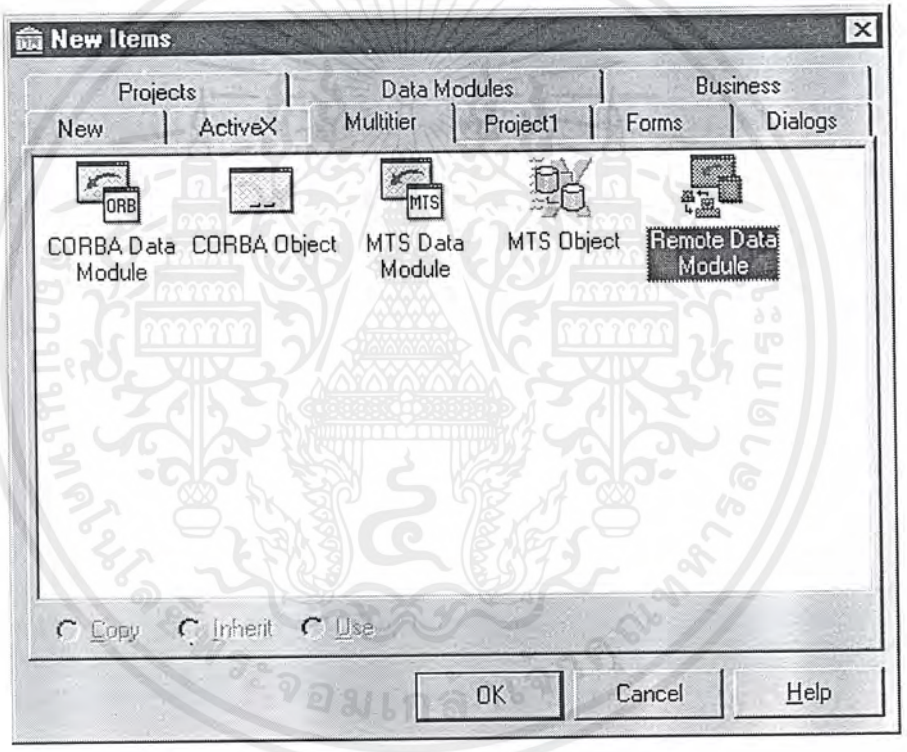
   1.1 เลือกเมนู File / New

   1.2 เลือกแถบ Multitier

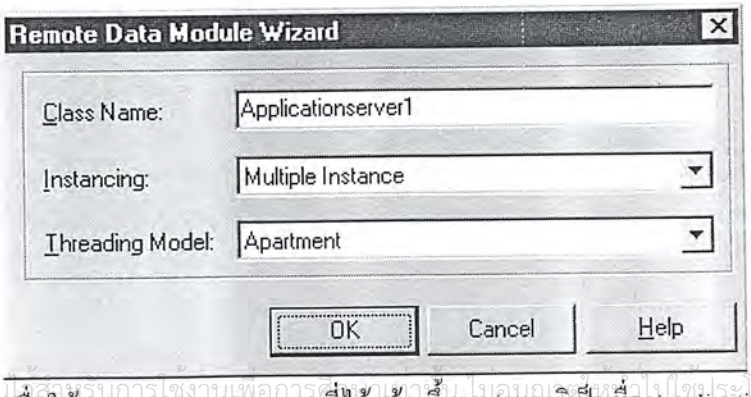
   1.3 เลือก Remote Data Module ดังรที่ 3-3

   1.4 ใส่ชื่อให้ Remote Data Module ที่จะสร้างเป็นแอพพลิเคชัน ดังรูปที่ 3-4

   1.5 บันทึกไฟล์ (สมมติเป็น unit2.pas)

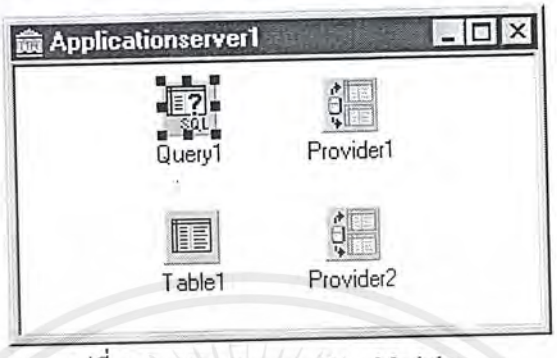


รูปที่ 3-3 การสร้าง Remote Data Module



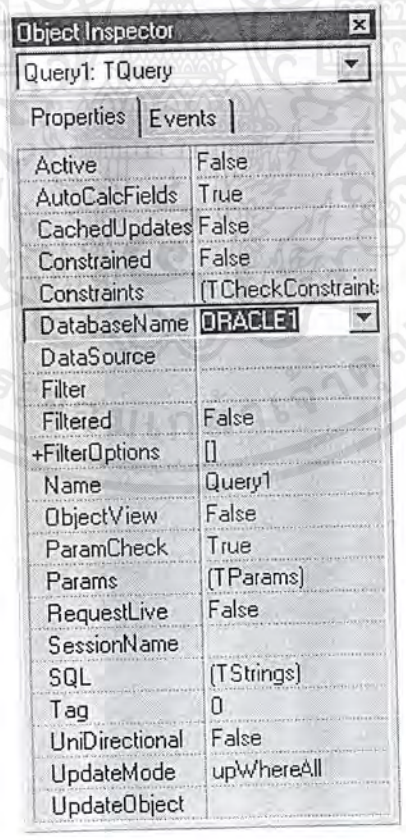
รูปที่ 3-4 กำหนดชื่อให้ Remote Data Module ที่ได้สร้างขึ้นมา (สมมติเป็นชื่อ Applicationserver1)

2. หลังจากได้สร้าง Remote Data Module แล้ว ให้นำคอมโพเนนต์ Provider จากแพ็กเกจ Midas มาวาง รวมถึงคอมโพเนนต์ที่ใช้เชื่อมต่อกับฐานข้อมูล เช่น Table และ Query จากแพ็กเกจ Data Access มาวางบน Remote Data Module ดังรูปที่ 3-5



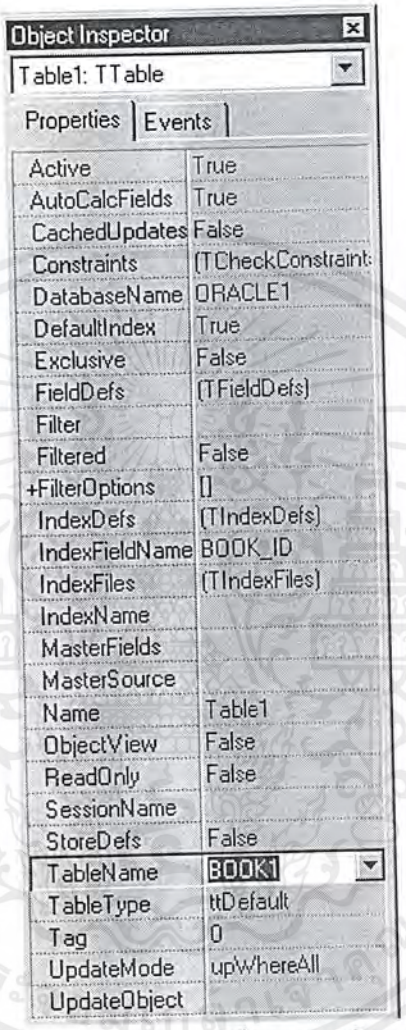
รูปที่ 3-5 แสดง Remote Data Module

3. กำหนดพร็อพเพอร์ตีของคอมโพเนนต์ Query1 โดยกำหนดให้ Database Name เป็นชื่อดาต้าเบสที่เราได้กำหนดเอาไว้ในส่วนการตั้งค่า BDE (Borland Database Engine) ในที่นี้กำหนดชื่อดาต้าเบสเป็น ORACLE1 ดังรูปที่ 3-6



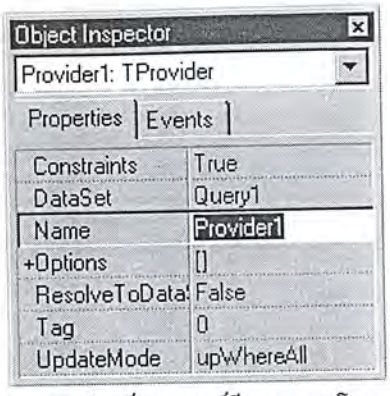
รูปที่ 3-6 การกำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ Query1

4. กำหนดพร็อพเพอร์ตีของคอมโพเนนต์ Table1 โดยกำหนดให้ Database Name เป็นชื่อดาต้าเบส ที่เราได้กำหนดเอาไว้ในส่วนการตั้งค่า BDE (Borland Database Engine) ในที่นี้กำหนดชื่อดาต้าเบสเป็น ORACLE1 กำหนดชื่อตารางที่จะให้แสดงผล และกำหนดฟิลด์ที่จะให้เป็นอินเด็กซ์ ดังรูปที่ 3-7



รูปที่ 3-7 การกำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ Table1

5. กำหนดพร็อพเพอร์ตีของคอมโพเนนต์ Provider โดยกำหนดพร็อพเพอร์ตี Dataset เป็นค่าคอมโพเนนต์ Query หรือ Table ซึ่งแล้วแต่การใช้งาน



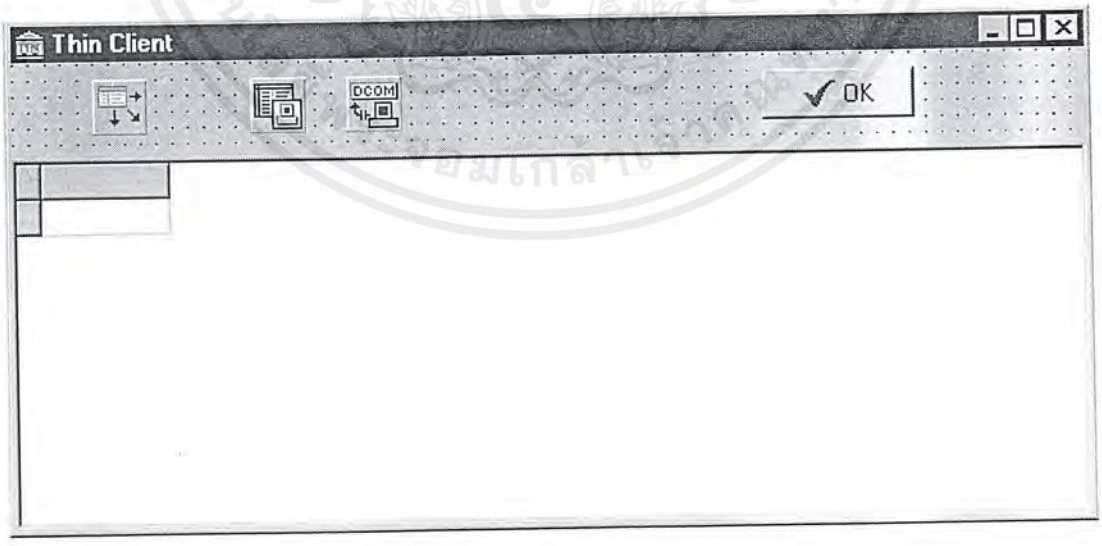
รูปที่ 3-8 การกำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ Provider1

6. รันโปรเจคนี้ 1ครั้ง เพื่อทำการรีจิสเตอร์แอพพลิเคชันบนเซิร์ฟเวอร์นี้ แล้วบันทึกโปรเจค
7. กลับไปตั้งค่าคอนฟิกูเรชันให้แก่แอพพลิเคชันบนแอพพลิเคชันเซิร์ฟเวอร์นี้ เพื่อกำหนดสิทธิการใช้งานต่างๆ ดังจะกล่าวต่อไปในหัวข้อ 3.6

3.5.2 การสร้างการเชื่อมต่อจากไคลเอนต์ไปสู่เซิร์ฟเวอร์

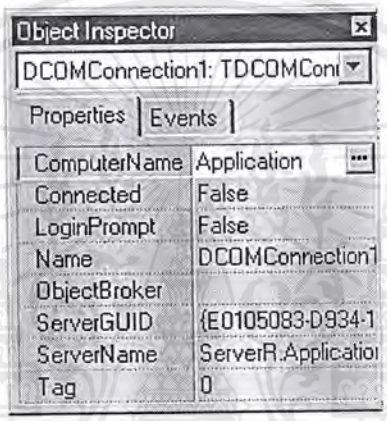
การเชื่อมต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์นั้น กระทำได้โดยผ่านคอมโพเนนต์ DCOMConnection และ ClientDataset โดยมีวิธีการสร้างการเชื่อมต่อดังนี้

1. นำคอมโพเนนต์ DCOMConnection และ ClientDataset มาวางบนฟอร์มเพื่อใช้ในการเชื่อมต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์
2. นำคอมโพเนนต์ DataSourceและ DBGrid มาวางบนฟอร์มเพื่อใช้ในการแสดงผลข้อมูลดัง รูปที่ 3-9



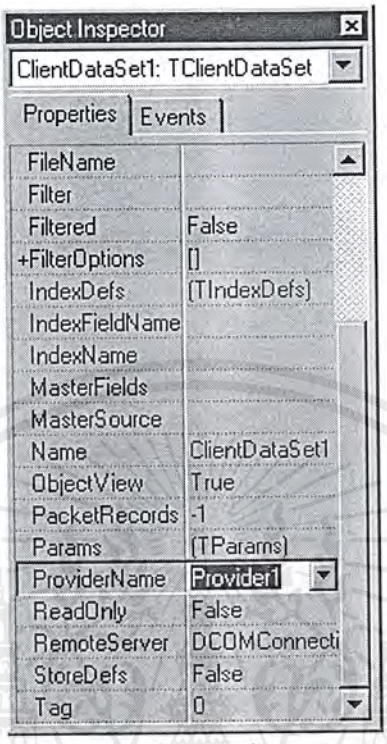
รูปที่ 3-9 การวางคอมโพเนนต์ที่ใช้ในการเชื่อมต่อระหว่างไคลเอนต์กับแอพพลิเคชันเซิร์ฟเวอร์

3. กำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ DCOMConnection โดย
   3.1 กำหนดพร็อพเพอร์ตี ComputerNameเข้ากับชื่อเครื่องที่ทำหน้าที่เป็นแอพพลิเคชันเซิร์ฟเวอร์
   3.2 กำหนดพร็อพเพอร์ตี ServerName ให้เป็นชื่อแอพพลิเคชันที่จะทำหน้าที่เป็นเซิร์ฟเวอร์ซึ่งเราได้สร้างไว้ก่อนหน้านี้ เมื่อเสร็จขั้นตอนนี้ เครื่องจะกำหนดค่าพร็อพเพอร์ตี ServerGUID ให้โดยอัตโนมัติ
   3.3 กำหนดให้พร็อพเพอร์ตี Connected เป็น TRUE เพื่อสร้างการเชื่อมต่อ ดังรูปที่ 3-10



รูปที่ 3-10 การกำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ DCOMConnection

4. กำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ ClientDataset โดย
   4.1 กำหนดพร็อพเพอร์ตี DCOMConnection ให้เป็น DCOMConnection ที่ได้เชื่อมต่อไว้จากข้อ 3
   4.2 กำหนดพร็อพเพอร์ตี Provider ให้เป็น Provider ของเซิร์ฟเวอร์ที่เราต้องการติดต่อด้วย ดังรูปที่ 3-11
5. กำหนดพร็อพเพอร์ตี Dataset ของคอมโพเนนต์ Datasource1 ให้เป็น ClientDataset1
6. กำหนดพร็อพเพอร์ตี Datasource ของคอมโพเนนต์ DBGrid1 ให้เป็น Datasource1
7. เสร็จสิ้นขั้นตอนการสร้างการเชื่อมต่อจากไคลเอนต์สู่เซิร์ฟเวอร์

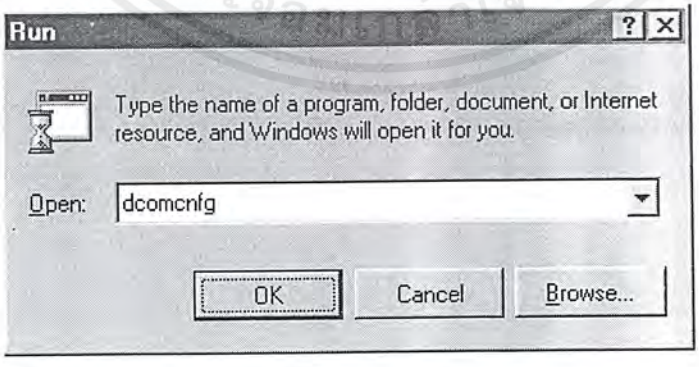


รูปที่ 3-11 การกำหนดค่าพร็อพเพอร์ตีของคอมโพเนนต์ ClientDataset1

3.6 ขั้นตอนการตั้งค่าคอนฟิกูเรชัน DCOM บนเครื่องแอพพลิเคชันเซิร์ฟเวอร์

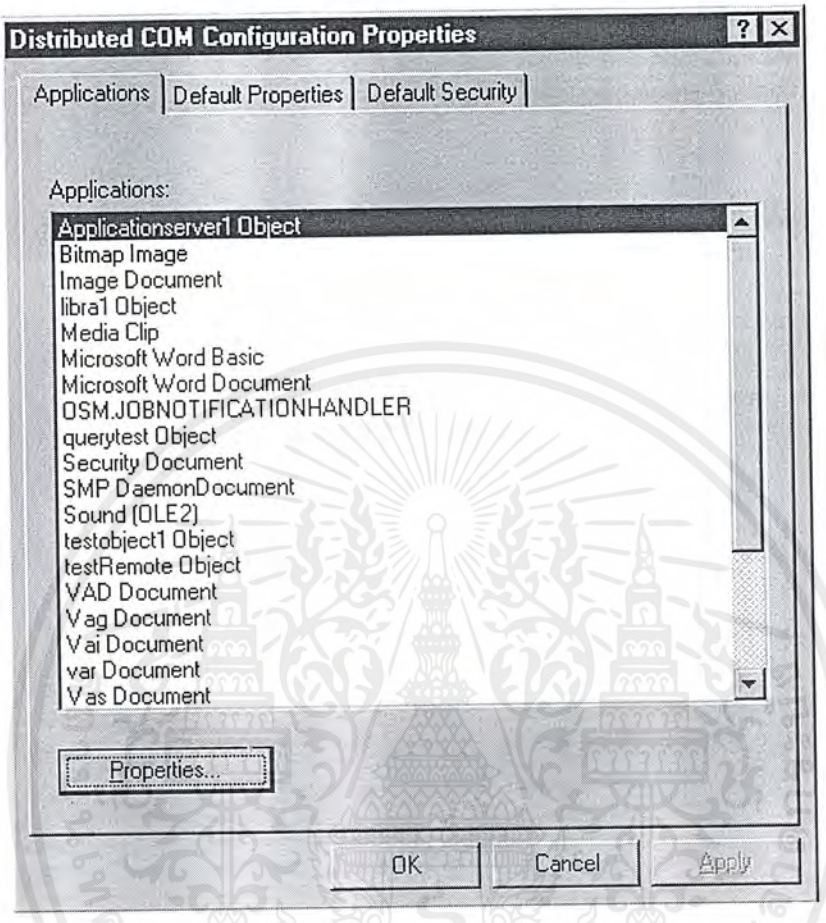
เราจำเป็นต้องมีการตั้งค่าคอนฟิกูเรชันให้กับแอพพลิเคชันที่เราได้สร้างขึ้นมา (จากหัวข้อ 3.5.1) และเก็บอยู่ในเครื่องแอพพลิเคชันเซิร์ฟเวอร์ เพื่อกำหนดสิทธิ์การเข้าใช้งานแอพพลิเคชันนั้นๆ โดยมีขั้นตอนดังนี้

1. เรียกเมนูบาร์ขึ้นมาสั่งรัน dcomcnfg



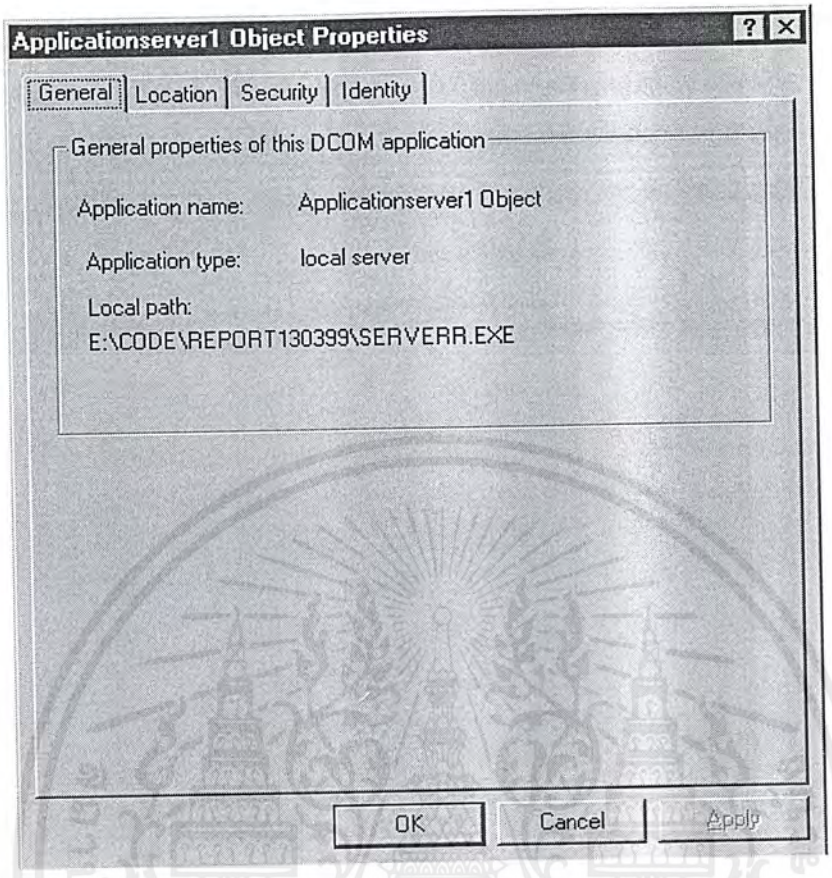
รูปที่ 3-12 การรัน dcomcnfg เพื่อตั้งค่าคอนฟิกูเรชัน DCOM

2. เลือกชื่อแอพพลิเคชันที่เราใช้เป็นเซิร์ฟเวอร์ แล้วคลิกปุ่ม Properties



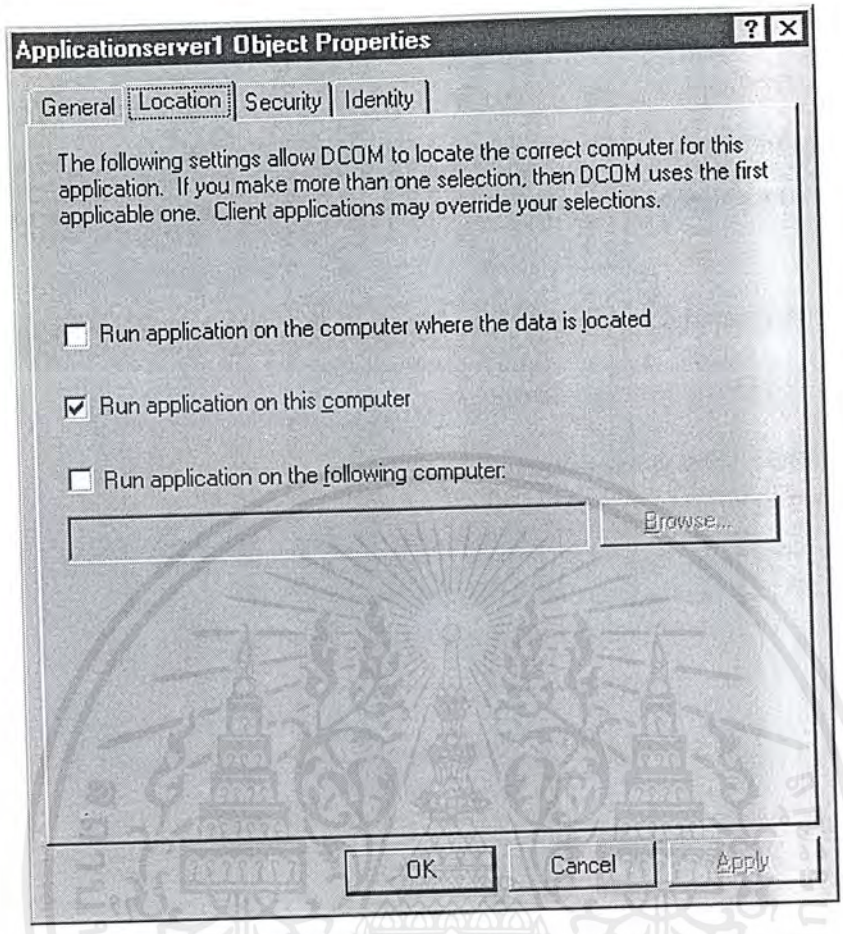
รูปที่ 3-13 การเลือกชื่อแอพพลิเคชันเซิร์ฟเวอร์

3. หลังจากเลือกแอพพลิเคชันที่จะรันเป็นเซิร์ฟเวอร์ได้แล้ว จะพบแถบพร็อพเพอร์ตี 4 แถบ แถบแรกได้แก่ แถบ General ที่จะแสดงรายละเอียดต่างๆ เกี่ยวกับตัวแอพพลิเคชันนี้ ดังรูปที่ 3-14



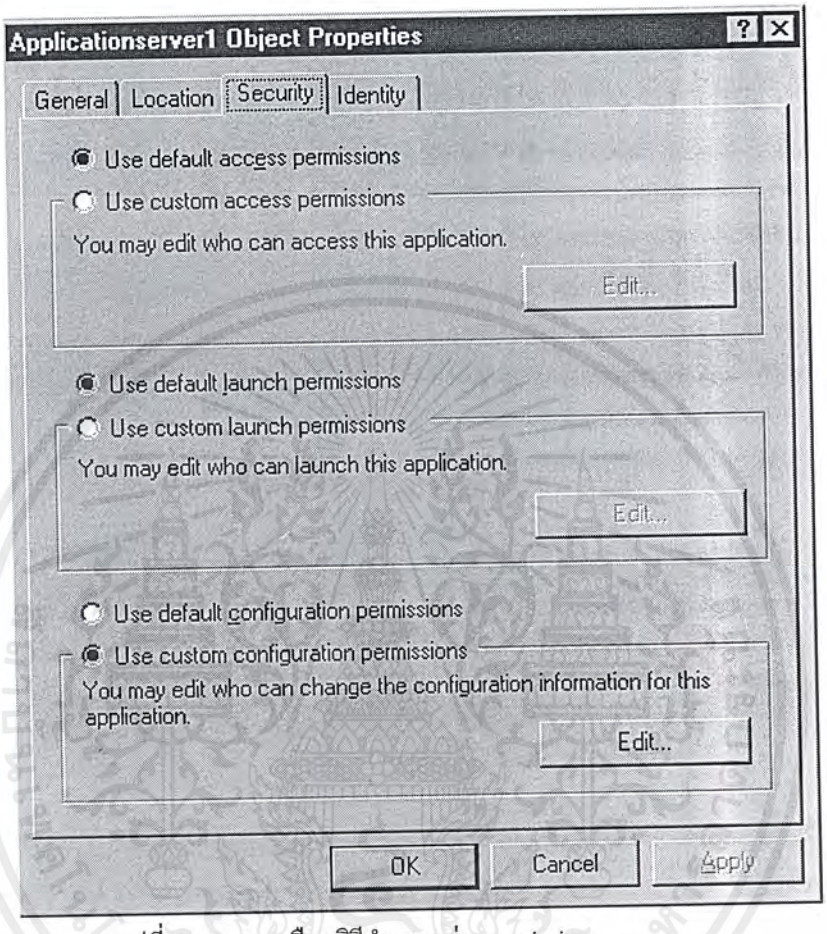
รูปที่ 3-14 แถบ General

4. แถบที่ 2 ได้แก่ แถบ Location เป็นการเลือกเครื่องคอมพิวเตอร์ที่จะทำการรันแอพพลิเคชันนี้ ซึ่งเป็นหน้าที่ของเครื่องที่ทำหน้าที่เป็นแอพพลิเคชันเซิร์ฟเวอร์



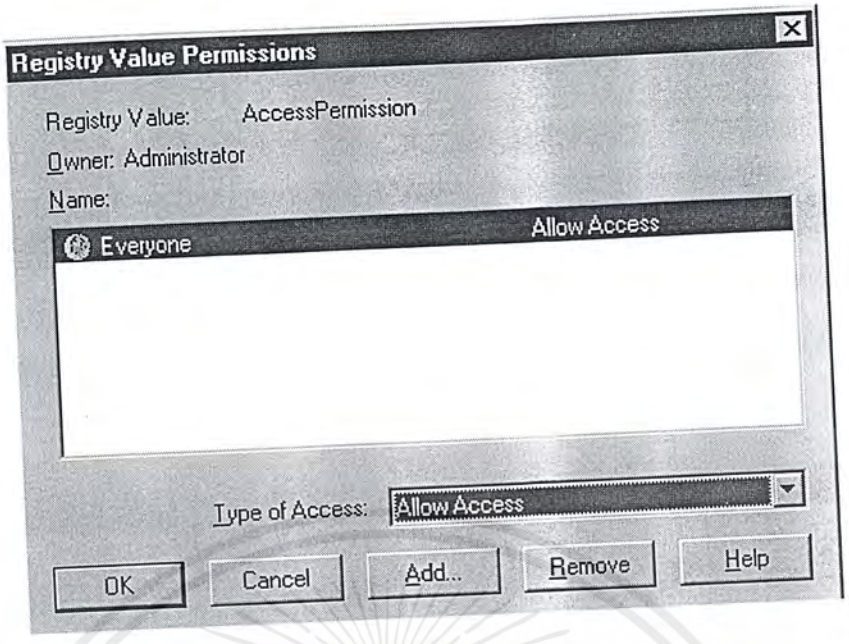
รูปที่ 3-15 การเลือกเครื่องที่จะให้ทำการรันแอพพลิเคชัน

5. แถบที่ 2 ได้แก่ แถบ Location เป็นการเลือกเครื่องคอมพิวเตอร์ที่จะทำการรันแอพพลิเคชันนี้ ซึ่งเป็นหน้าที่ของเครื่องที่ทำหน้าที่เป็นแอพพลิเคชันเซิร์ฟเวอร์
6. แถบที่ 3 ได้แก่ แถบ Security เป็นส่วนการเลือก permission ต่างๆ ว่าจะให้กำหนดแบบดีฟอลต์ของเครื่อง หรือจะเข้าไปกำหนดเองสำหรับแอพพลิเคชันนี้โดยเฉพาะ



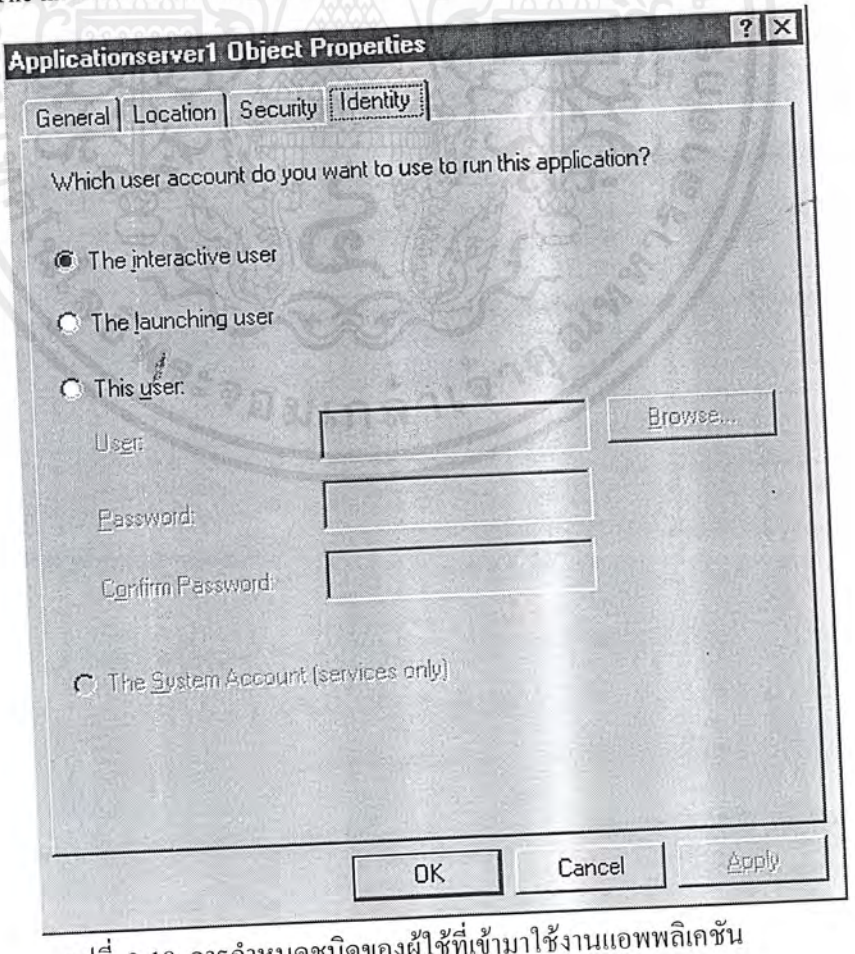
รูปที่ 3-16 การเลือกวิธีกำหนดค่า permission

ถ้าเราเลือกการกำหนดค่าสิทธิการใช้งาน (permission) เป็นแบบ Custom จะมีหน้าจอให้เพิ่มชื่อผู้ใช้ที่เราจะให้สิทธิในการเข้าใช้ แอพพลิเคชันนี้ ดังรูปที่ 3.16 โดยผู้ที่ได้รับสิทธิ์การใช้งานนั้นจะต้องเป็นผู้ที่มียูสเซอร์แอคเคาน์อยู่บนเครื่องเซิร์ฟเวอร์ด้วย



รูปที่ 3-17 การกำหนดสิทธ์แบบ Custom ให้ผู้เข้าใช้งาน

6. แถบสุดท้าย ได้แก่แถบ Identity ซึ่งเป็นแถบการกำหนดชนิดของผู้ใช้ ที่ให้เข้ามาใช้งาน ควรกำหนดให้เป็น The interactive user เพื่อรองรับการทำงานทุกสภาวะ



รูปที่ 3-18 การกำหนดชนิดของผู้ใช้ที่เข้ามาใช้งานแอพพลิเคชัน

3.7 Server GUID

GUID หรือ Globally Unique Identifier เป็นชุดของเลขจำนวนเต็ม 128บิต ซึ่งถูกสร้างโดยอัลกอริทึมที่กำหนดโดย OSF's (Open Software Foundation) Distributed Computing Environment ใช้สำหรับเป็นค่าเฉพาะให้กับแต่ละแอพพลิเคชันเซิร์ฟเวอร์เพื่อไม่ให้เกิดความซ้ำซ้อนในการเรียกใช้งานในแต่ละแอพพลิเคชัน ในการเชื่อมต่อของไคลเอนต์ผ่าน DCOM ไปยังเซิร์ฟเวอร์โดยใช้คอมโพเนนต์ TDCOMConnection นั้น เราจำเป็นต้องบอกชื่อเซิร์ฟเวอร์ที่ต้องการอ้างถึง และค่า GUID ของเซิร์ฟเวอร์ (Server GUID)นั้น

บทที่ 4
ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

4.1 ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object Relational Database : ORDB)

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ คือ ฐานข้อมูลชนิดที่นำแนวความคิดแบบออปเจ็กต์โอเรียนเต็ด (Object-Oriented Database) มาผสมผสานกับระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เพื่อทำให้ระบบฐานข้อมูลเชิงสัมพันธ์มีความสามารถทางด้านออปเจ็กต์ได้ โดยระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะมองข้อมูลและทำการจัดเก็บข้อมูลเป็นแบบตารางโดยมีการจัดการฐานข้อมูลแบบเชิงสัมพันธ์ แต่การติดต่อกับผู้ใช้งานได้นำระบบของออปเจ็กต์โอเรียนเต็ดมาใช้งาน ซึ่งระบบออปเจ็กต์โอเรียนเต็ดจะมีการสนับสนุนการพัฒนาและมีการดูแลระบบฐานข้อมูลขนาดใหญ่ๆได้ดีกว่า

ฐานข้อมูลเชิงวัตถุสัมพันธ์เสมือนการเพิ่มชั้นออปเจกต์โอเรียนเต็ดครอบบนชั้นของระบบฐานข้อมูลสัมพันธ์ที่มีอยู่

4.2 ความสามารถพื้นฐานของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์มีความสามารถพื้นฐานดังนี้

4.2.1 ความถูกต้องของข้อมูลและทรานแซกชัน (Data and Transaction Integrity)

4.2.2 ความปลอดภัย

4.2.3 การกู้ข้อมูลคืนกลับ (Recovery)

4.2.4 การใช้ข้อมูลร่วมกันจากผู้ใช้หลายๆคน (Sharing and Concurrent multi-user Access)

4.2.5 การเข้าถึงข้อมูลและการคิวรีข้อมูล (Access and Query Language)

4.3 การเปรียบเทียบระหว่างฐานข้อมูลเชิงวัตถุสัมพันธ์กับฐานข้อมูลเชิงสัมพันธ์ (Relational Database)

1. ฐานข้อมูลเชิงสัมพันธ์ใช้ภาษาเอสคิวแอล (Structure Query Language) ตามมาตรฐาน SQL-92 เป็นภาษาที่ใช้ในการกำหนดโครงสร้างและจัดการกับฐานข้อมูล ในขณะที่ในฐานข้อมูลเชิงวัตถุสัมพันธ์ใช้ภาษาเอสคิวแอลตามมาตรฐาน SQL3
2. ฐานข้อมูลเชิงสัมพันธ์จะสนับสนุนข้อมูลที่มีความซับซ้อนมากขึ้น โดยโดเมนของทุกๆแอตทริบิวต์จะต้องเป็นอะตอมมิก กล่าวคือในแอตทริบิวต์หนึ่งๆสามารถเก็บข้อมูลได้เพียงค่าเดียว ในขณะที่ฐานข้อมูลเชิงวัตถุสัมพันธ์อนุญาตให้ค่าของข้อมูลในแอตทริบิวต์มีหลายค่าได้
3. ฐานข้อมูลเชิงวัตถุสัมพันธ์สามารถจัดการกับที่มีคุณสมบัติเชิงวัตถุต่างๆ เช่น เอนแคปซูเลชัน (Encapsulation) , อินเฮอริแตนซ์(Inheritance) ได้

4.4 ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ Oracle 8

Oracle 8 เป็นระบบฐานข้อมูลแบบเชิงวัตถุสัมพันธ์ (Object Relational Database Management System : ORDBMS) ซึ่งเป็นระบบฐานข้อมูลที่สามารถรองรับข้อมูลที่มีความซับซ้อนรวมถึงข้อมูลเชิงวัตถุ โดย Oracle8 ได้จัดเตรียมชนิดข้อมูลชนิดใหม่ขึ้นมาเพื่อรองรับกับข้อมูลที่มีความซับซ้อนมาก ๆ เหล่านี้โดยเฉพาะยอมให้มีข้อมูลชนิดที่ผู้ใช้กำหนดขึ้นมาเอง (User-Defined Datatypes) ข้อมูลชนิดนี้จะมีคุณสมบัติทางออปเจ็กต์โอเรียนเต็ด เช่นเดียวกับ "ออปเจ็กต์" ในโปรแกรมภาษาที่สามารถเขียนโปรแกรมเชิงวัตถุต่าง ๆ แต่คุณสมบัติบางอย่างอาจยังไม่เทียบเท่าโปรแกรมภาษาเหล่านั้น เช่น คุณสมบัติการสืบทอด(Inheritance) ซึ่งใน Oracle เวอร์ชันปัจจุบัน (Oracle 8.0.4) ยังไม่สามารถกระทำได้

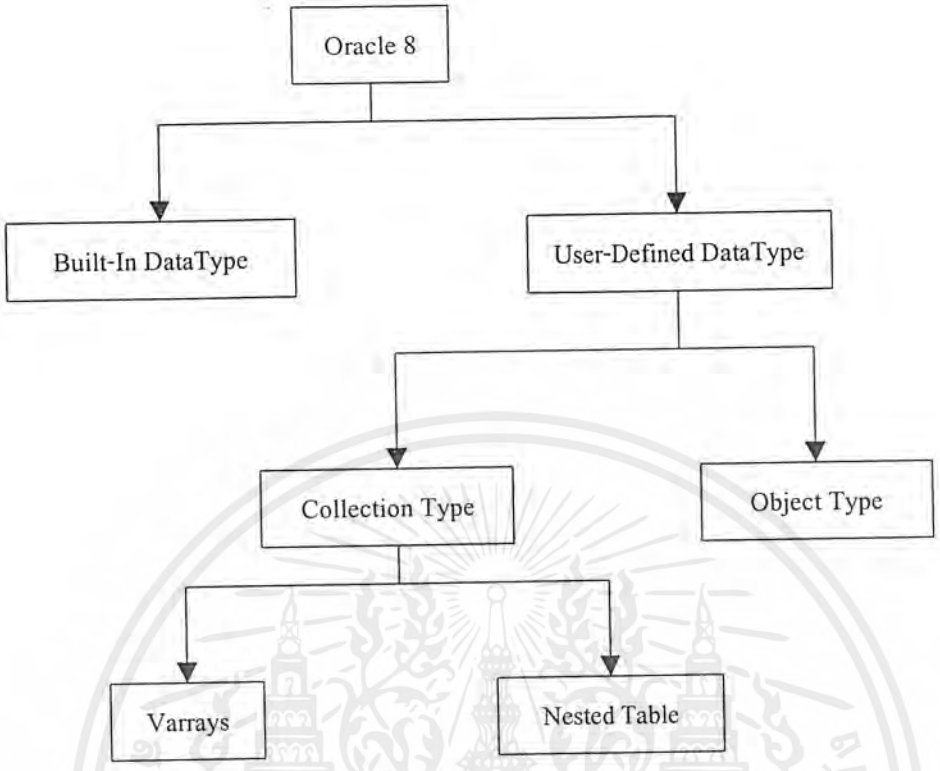
4.4.1 คุณสมบัติทางออปเจ็กต์โอเรียนเต็ด ของ Object Type ใน Oracle 8

1. ในการปกป้องข้อมูล (Encapsulation) นั้น Object Type ใน Oracle 8 จะแบ่งการ ปกป้องข้อมูลออกเป็น 2 ส่วนคือในส่วนของ Specification จะมีการปกป้องแบบ Public และในส่วนของ Body จะมีการปกป้องเป็นแบบ Private

2. สำหรับคุณสมบัติโพลิมอร์ฟีซึม (Polymorphism) นั้น Object Type ใน Oracle 8 สามารถมีเมทธอดที่ชื่อซ้ำกันแต่มีการทำงานที่แตกต่างกันได้

4.5 ชนิดของข้อมูลในระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ Oracle 8 (Oracle 8 Datatype)

ในระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ Oracle เวอร์ชัน 8.0.4 มีการจัดเก็บชนิดข้อมูลแยกออกเป็นชนิดต่างๆ ดังรูปที่ 4-1



รูปที่ 4-1 ชนิดของข้อมูลใน Oracle 8

4.5.1 Built-In DataType

Built-In DataType เป็นชนิดของข้อมูลทั่วไปที่มีอยู่ใน Oracle ได้แก่

4.5.1.1 ข้อมูลชนิด CHARACTER

4.5.1.1.1 ข้อมูลชนิด CHAR

ชนิดของข้อมูลประเภท CHAR เป็นชนิดของข้อมูลตัวอักษรที่มีขนาดของความยาวคงที่ โดยที่เมื่อมีการสร้างตารางที่มีคอลัมน์เป็นชนิดข้อมูลประเภท CHAR จะต้องมีการกำหนดความยาวข้อมูลเป็นไบต์ระหว่าง 1 ถึง 2000 ไบต์ (โดยจะมีค่าเริ่มต้นเป็น 1) โดย Oracle จะมีการรับประกันว่า เมื่อมีการ Insert และ Update แถวในตาราง ค่าที่เก็บอยู่ในคอลัมน์ที่มีข้อมูลประเภท CHAR จะมีขนาดคงที่ตามที่กำหนดไว้

- ถ้าหากข้อมูลที่จะ Insert หรือ Update มีขนาดของความยาวน้อยกว่าที่กำหนดไว้ ข้อมูลจะมีการเติมช่องว่างให้ได้ขนาดตามที่กำหนดไว้

- ถ้าหากข้อมูลที่จะ Insert หรือ Update มีขนาดของความยาวมากกว่าเนื่องจากช่องว่างตรงส่วนท้ายของข้อมูล ช่องว่างเหล่านั้นจะถูกตัดออกจนได้ขนาดตามที่กำหนดไว้

- ถ้าหากข้อมูลมีขนาดของความยาวมากกว่าที่กำหนดจะมีการแสดงข้อผิดพลาด

4.5.1.1.2 ข้อมูลชนิด VARCHAR2

ชนิดของข้อมูลประเภท VARCHAR2 เป็นชนิดของข้อมูลตัวอักษรที่มีขนาดของความยาวเปลี่ยนแปลงได้ โดยที่เมื่อมีการสร้างตารางที่มีคอลัมน์เป็นชนิดของข้อมูลประเภท VARCHAR2 จะต้องมีการกำหนดความยาวของข้อมูลที่มากที่สุดเป็นไบต์ โดยสามารถมีค่าได้ระหว่าง 1 ถึง 4000 ดังนั้นข้อมูลที่เก็บไว้จะมีขนาดเท่าใดก็ได้ที่อยู่ระหว่าง 1 ถึงความยาวสูงสุด (จะมีการแสดงข้อผิดพลาดถ้าหากข้อมูลมีขนาดใหญ่กว่าความยาวสูงสุด)

4.5.1.1.3 ข้อมูลชนิด VARCHAR

ชนิดของข้อมูลประเภท VARCHAR เป็นชนิดของข้อมูลที่มีลักษณะเหมือนกับชนิดของข้อมูลประเภท VARCHAR2

4.5.1.1.4 ข้อมูลชนิด NCHAR และ NVARCHAR2

ชนิดของข้อมูลประเภท NCHAR และ NVARCHAR2 เก็บข้อมูลประเภท NLS โดยที่ชนิดของข้อมูลประเภท NCHAR เก็บข้อมูลที่มีขนาดคงที่ที่สอดคล้องกับข้อมูลประเภทความยาวคงที่ หรือความยาวเปลี่ยนแปลงได้ สำหรับชนิดของข้อมูลประเภท NVARCHAR2 จะเก็บข้อมูลที่มีความยาวเปลี่ยนแปลงได้ ดังนั้นเมื่อมีการสร้างตารางที่มีคอลัมน์ของข้อมูลประเภท NCHAR หรือ NVARCHAR2 จะต้องมีการกำหนดขนาดที่มากที่สุดได้ทั้งในหน่วยตัวอักษร (สำหรับข้อมูลประเภทความยาวคงที่)หรือในหน่วยไบต์(สำหรับข้อมูลประเภทความยาวเปลี่ยนแปลงได้)

- ขนาดของความยาวที่มากที่สุดสำหรับชนิดของข้อมูลประเภท NCHAR คือ 2000 ไบต์หรือจำนวนของตัวอักษรสามารถเก็บได้ 2000 ไบต์
- ขนาดของความยาวที่มากที่สุดสำหรับชนิดของข้อมูลประเภท NVARCHAR2 คือ 4000 ไบต์หรือจำนวนของตัวอักษรสามารถเก็บได้ 4000 ไบต์

4.5.1.1.5 ข้อมูลชนิด LONG

ชนิดของข้อมูลประเภท LONG จะใช้ในการเก็บข้อมูลที่มีขนาดมากถึง 2 กิกะไบต์

4.5.1.2 ข้อมูลชนิด NUMBER

ชนิดของข้อมูลประเภทตัวเลข จะใช้ในการเก็บตัวเลขจำนวนเต็มและตัวเลขที่มีจุดทศนิยม (floating-point numbers) เก็บขนาดได้ถึง 21 ไบต์

4.5.1.3 ข้อมูลชนิด DATE

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

4.5.1.4 ข้อมูลชนิด BFILE

ชนิดของข้อมูลประเภท BFILE จะเก็บข้อมูลประเภทไบนารีที่เป็นแฟ้มข้อมูลนอกฐานข้อมูล โดยในคอลัมน์ BFILE จะเก็บค่าตำแหน่งของแฟ้มข้อมูล โดยชนิดของข้อมูลประเภทนี้สามารถเก็บข้อมูลได้ถึง 4 กิกะไบต์ โดยที่ชนิดของข้อมูลประเภทนี้จะเป็นอ่านได้อย่างเดียวไม่สามารถเปลี่ยนแปลงแก้ไขข้อมูลได้

4.5.2 User-Defined Datatype

User-Defined Datatype เป็นชนิดข้อมูลที่ผู้ใช้กำหนดขึ้นมาเองซึ่งจะแบ่งออกเป็น 2 ชนิด คือ Object Type และ Collection Type

4.5.2.1 Object Type

Object Type ใน Oracle8 เป็นชนิดของข้อมูลที่ผู้ใช้กำหนดขึ้นมาใช้เอง (user-defined) ซึ่งสามารถเอนแคปซูเลตโครงสร้างของข้อมูลประกอบด้วยฟังก์ชันและโพรซีเยอร์ ซึ่งเป็นตัวจัดการข้อมูล Object Type ประกอบด้วย 3 ส่วน คือ

1. ชื่อ (Name) เป็นชื่อที่ใช้ในการอ้างถึง Object Type ซึ่งจะไม่ซ้ำกันภายในสกีมา (Schema)เดียวกัน
2. แอตทริบิวต์ (Attribute) เป็นส่วนประกอบของ Object Type Attribute มีชนิดเป็น Built-In datatype หรือ ชนิดของข้อมูลที่ผู้ใช้กำหนดขึ้นมาเอง (User-define type)
3. เมทธอด (Method) เป็นฟังก์ชันหรือโพรซีเยอร์

4.5.2.2 Collection Type

Collection Type มีลักษณะที่เป็นหน่วยของข้อมูล (data unit) ที่มีจำนวนสมาชิกไม่จำกัดและสมาชิกทั้งหมดมีชนิดข้อมูลเหมือนกัน Collection Type ได้แก่ Array Type และ Table Type ซึ่งหน่วยของข้อมูลจะเรียกว่า Varrays และ NestedTable

Collection Type จะมี constructor methods ชื่อของ constructor methods ก็คือชื่อของ Collection Types นั้น ๆ

4.5.2.2.1 Varrays

มีลักษณะเป็นเซ็ตของข้อมูลแบบมีลำดับ (Array)โดยสมาชิกทุกตัวจะเป็นข้อมูลชนิดเดียวกัน สมาชิกแต่ละตัวจะมีตัวชี้ (Index) ซึ่งเป็นหมายเลขที่สอดคล้องกับตำแหน่งของสมาชิกในอาร์เรย์จำนวนของสมาชิกในอาร์เรย์ จำนวนของสมาชิกในอาร์เรย์ก็คือขนาดของอาร์เรย์ Oracle ยอมให้ขนาดของอาร์เรย์เปลี่ยนแปลงได้ ซึ่งนี่ก็คือสาเหตุที่ถูกเรียกว่า varrays เราจะต้องระบุขนาดที่ใหญ่ที่สุดเมื่อเรามีการกำหนดข้อมูลชนิดอาร์เรย์

4.5.2.2.2 Nested Table

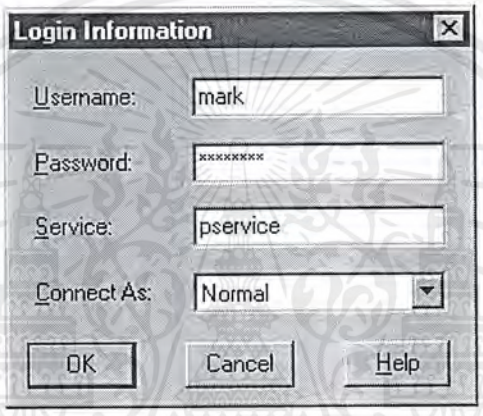
Nested Table เป็นเซ็ตของข้อมูลแบบที่ไม่เรียงลำดับ ซึ่งสมาชิกทุก ๆ ตัวจะเป็นข้อมูลชนิดเดียวกันและ Nested Table จะมีเพียงคอลัมน์เดียวและคอลัมน์จะ

เป็นข้อมูลชนิด Built-in Datatype หรือเป็น Object Type เราสามารถมองเป็นเหมือนกับเป็นตาราง ที่มีหลาย ๆ คอลัมน์ ได้ซึ่ง คอลัมน์ ก็คือแอตทริบิวต์ของ Object Type

4.6 การสร้างตารางเพื่อใช้งานข้อมูลแบบออปเจ็กต์ (Object Type) ใน Oracle 8

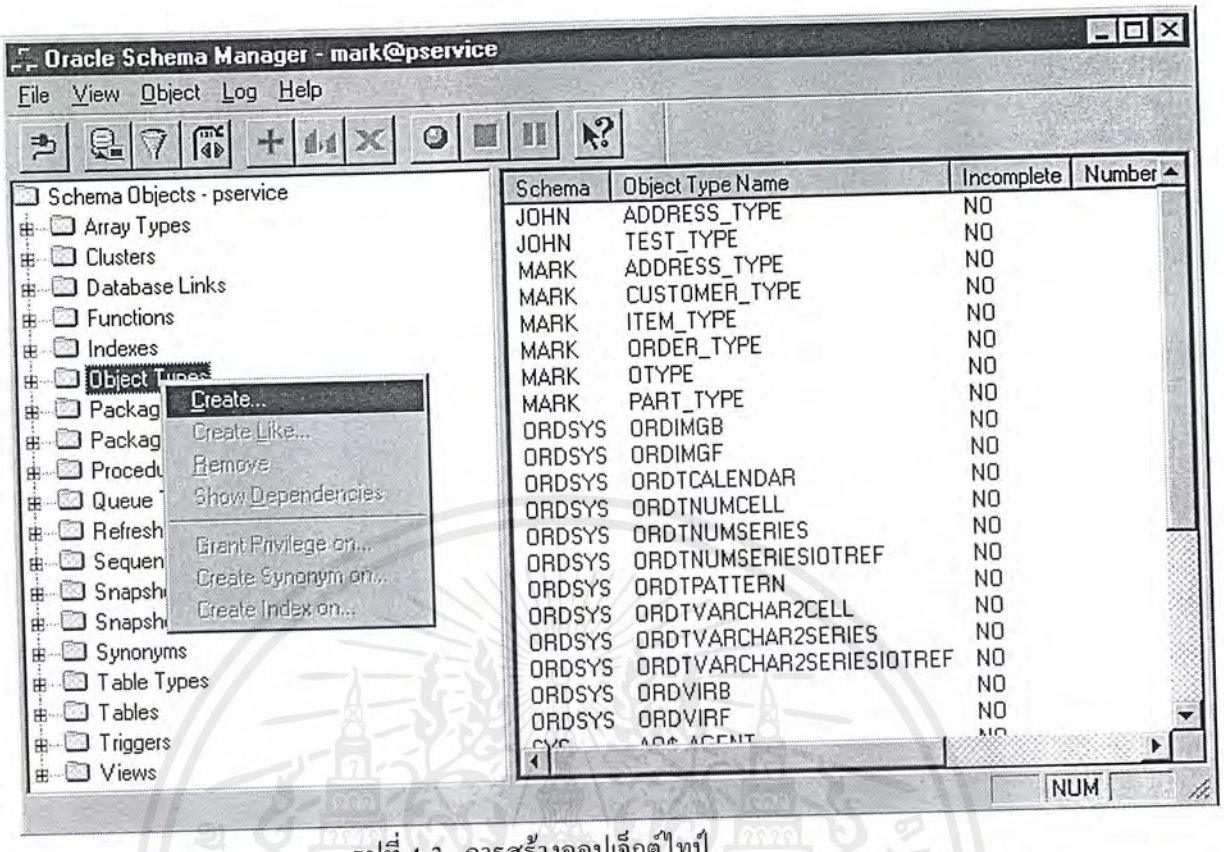
ในการใช้งานออปเจ็กต์ไทป์นั้น เราจำเป็นต้องสร้างตารางเพื่อใช้งานข้อมูลแบบออปเจ็กต์ โดยมีขั้นตอนดังต่อไปนี้

1. เรียกการใช้งาน Oracle Schema Manager จะพบหน้าจอล็อกอินให้ใส่ชื่อผู้ใช้ ,รหัสผ่านและชื่อเซอร์วิสที่ต้องการติดต่อ ดังรูปที่ 4-2



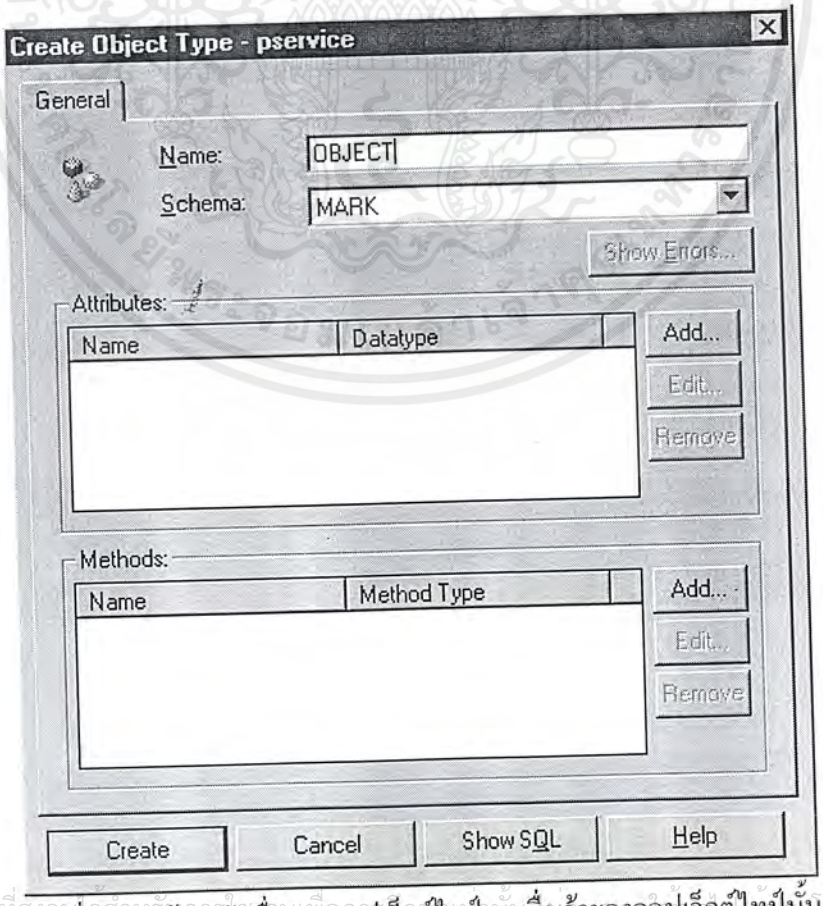
รูปที่ 4-2 หน้าจอล็อกอินก่อนเข้าใช้งาน Oracle Schema Manager

2. หลังจากผ่านการล็อกอินเข้าสู่ Oracle Schema Manager แล้ว คลิกขวาที่ Object Types เลือกเมนู Create เพื่อทำการสร้างออปเจ็กต์ไทป์ ใหม่ขึ้นมา ดังรูปที่ 4-3

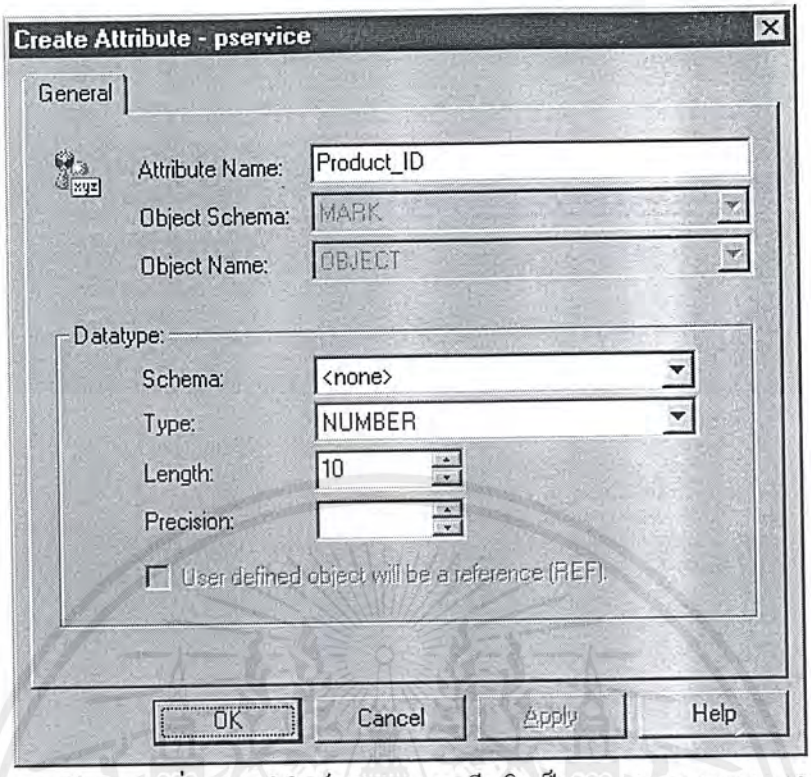


รูปที่ 4-3 การสร้างออปเจ็กต์ไทป์

3. กำหนดชื่อของออปเจ็กต์ไทป์ที่ได้สร้างขึ้นมา รวมถึงชื่อเจ้าของออปเจ็กต์ไทป์นั้น

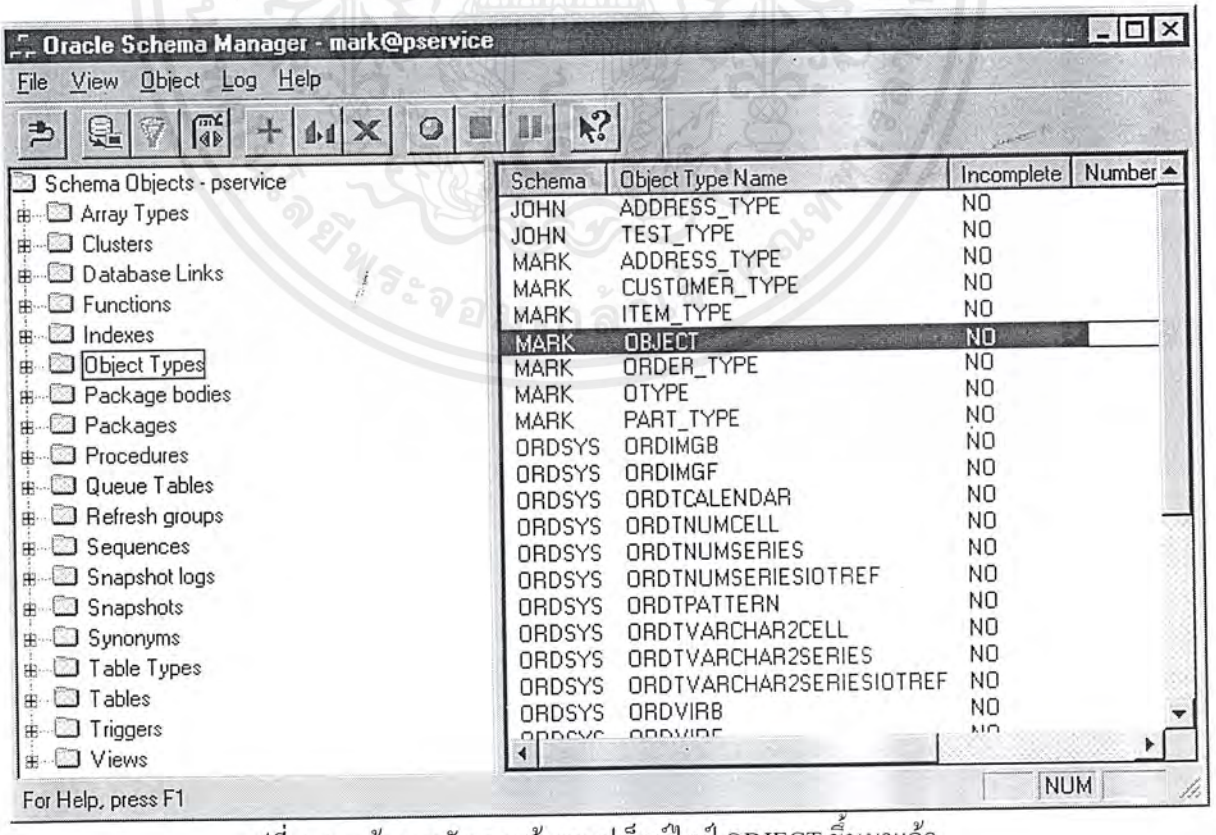


รูปที่ 4-4 หน้าจอรับชื่อของออปเจ็กต์ไทป์และชื่อเจ้าของออปเจ็กต์ไทป์นั้น



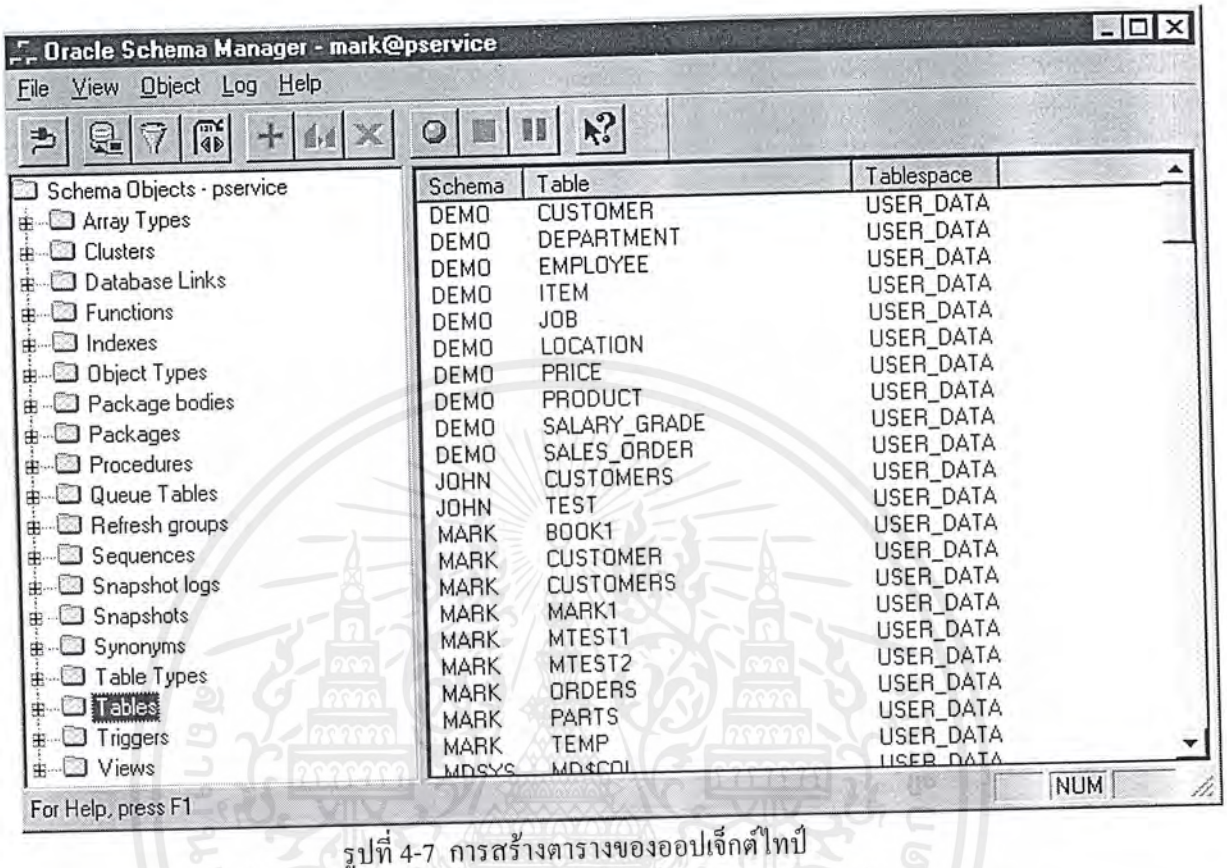
รูปที่ 4-5 หน้าจอการเพิ่มแอตทริบิวต์ Product_ID มีชนิดเป็น Number ความยาว 10 หลัก

4. กำหนดแอตทริบิวต์และเมทธอดของออปเจ็กต์ไทป์นั้น โดยคลิกปุ่ม Add จะได้หน้าจอดังรูปที่ 4-5 ให้ใส่ชื่อแอตทริบิวต์ และประเภทข้อมูลของแอตทริบิวต์นั้น ทำเช่นนี้ไปเรื่อยๆ สำหรับทุกแอตทริบิวต์



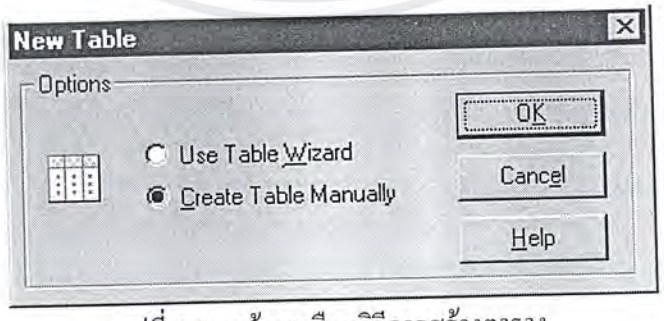
รูปที่ 4-6 หน้าจอหลังจากสร้างออปเจ็กต์ไทป์ OBJECT ขึ้นมาแล้ว

5. หลังเสร็จทุกขั้นตอนแล้ว เราจะได้ออปเจ็กต์ไทป์ (ในที่นี้ได้แก่ OBJECT) ขึ้นมาดังรูปที่ 4-6



รูปที่ 4-7 การสร้างตารางของออปเจ็กต์ไทป์

6. สร้างตารางของออปเจ็กต์ไทป์ที่เราได้สร้างขึ้นมาก่อนหน้านี้ โดยคลิกขวาที่ Tables เลือกเมนู Create เพื่อทำการสร้างตารางใหม่ขึ้นมา ดังรูปที่ 4-7
7. หลังจากนั้นจะมีหน้าจอให้เลือกวิธีสร้างตาราง ในที่นี้ให้เลือก Create Table Manually ดังรูปที่ 4-8



รูปที่ 4-8 หน้าจอเลือกวิธีการสร้างตาราง

8. กำหนดคุณสมบัติของตารางที่ต้องการสร้างขึ้น

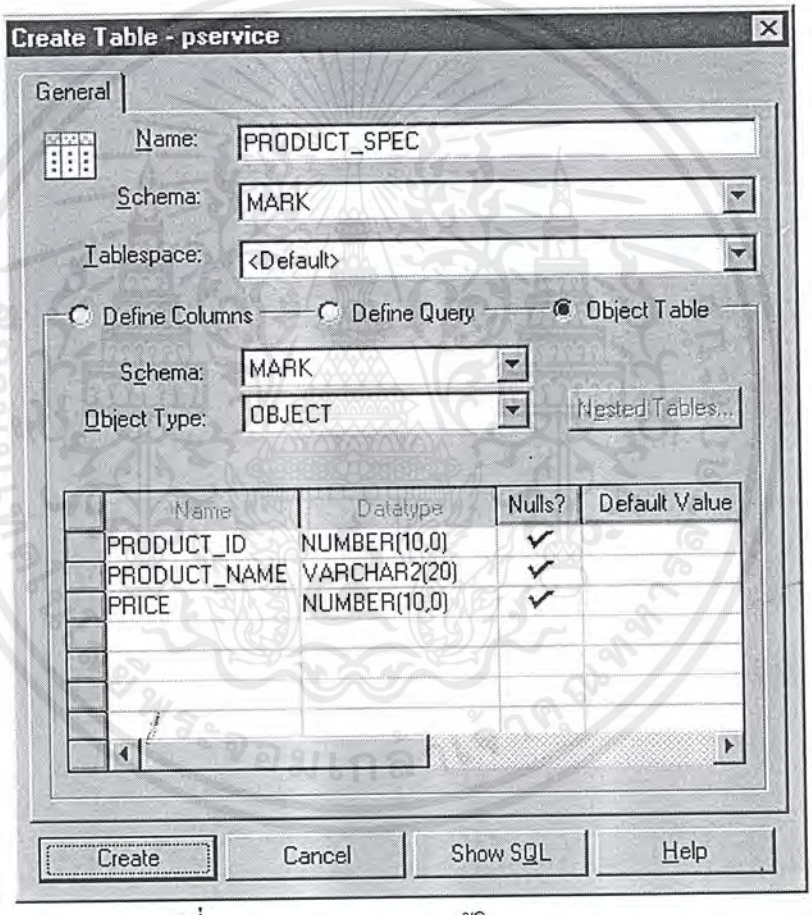
   8.1 กำหนดชื่อตารางที่เราต้องการสร้างขึ้น ในที่นี้เป็นตาราง PRODUCT_SPEC

   8.2 กำหนดชื่อเจ้าของตาราง (Schema) ในที่นี้เป็น MARK

   8.3 กำหนด Tablespace

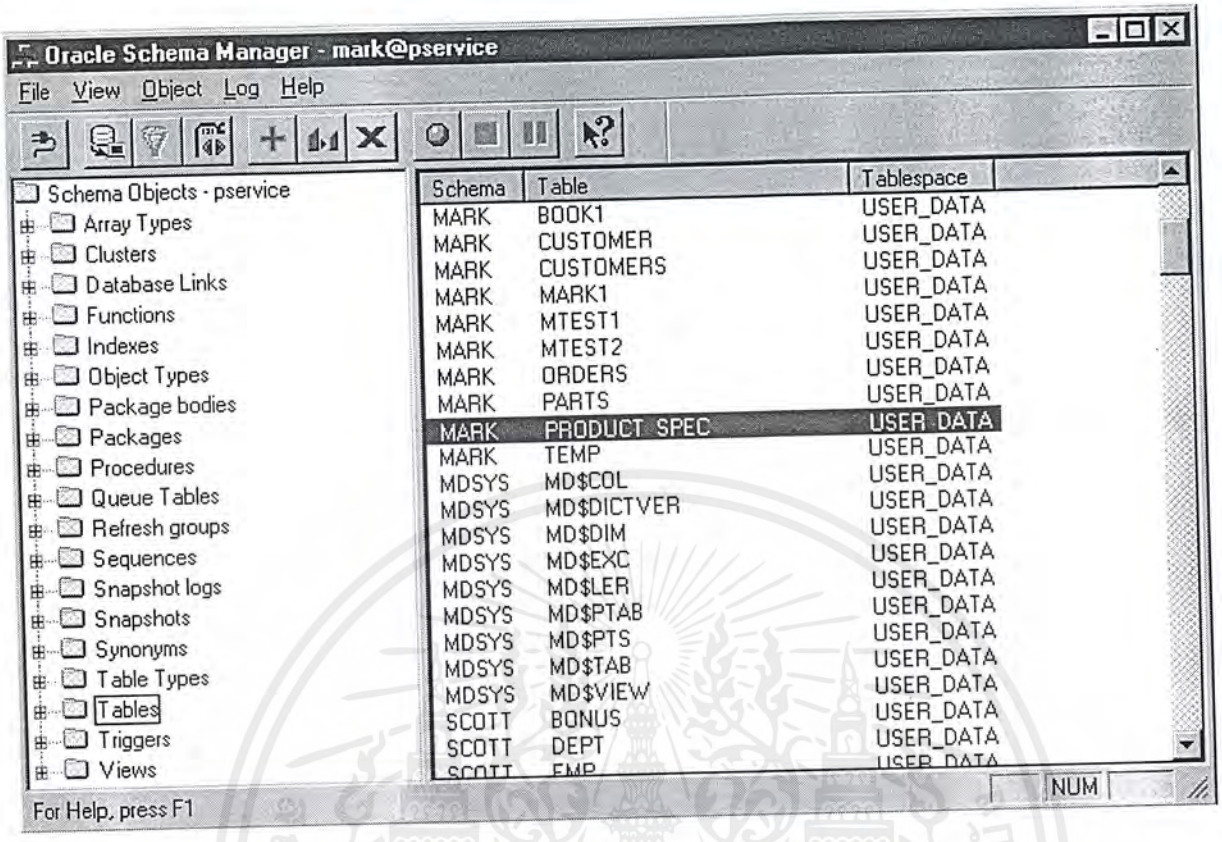
   8.4 เลือกชนิดของตารางเป็น Object Table

   8.5 กำหนดชื่อออปเจ็คไทป์ และชื่อเจ้าของออปเจ็คไทป์นั้น ในที่นี้ได้แก่ออปเจ็กต์ไทป์ OBJECT ซึ่งมี MARK เป็นเจ้าของ ซึ่งจะทำให้มีการแสดงรายละเอียดของแต่ละแอตทริบิวต์ของออปเจ็กต์ไทป์ OBJECT ที่เราได้กำหนดไว้ก่อนหน้านี้ออกมา



รูปที่ 4-9 การกำหนดคุณสมบัติของตาราง

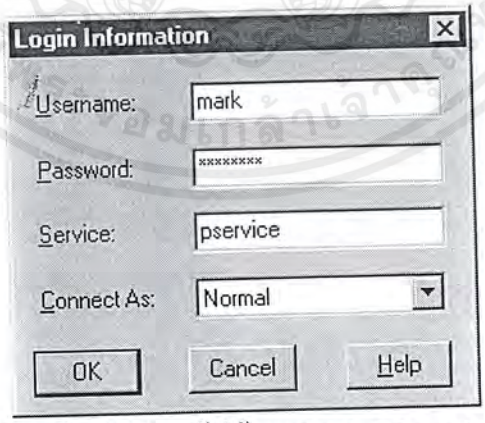
9. เมื่อคลิกปุ่ม Create ก็จะเสร็จสิ้นขั้นตอนการสร้างตารางของออปเจ็กต์ไทป์ ดังรูปที่ 4-10 จะเห็นได้ว่ามีตาราง PRODUCT_SPEC ที่เราได้สร้างไว้เพิ่มขึ้นมา



รูปที่ 4-10 ตาราง PRODUCT_SPEC ของสกีมา MARK ได้ถูกเพิ่มขึ้นมา

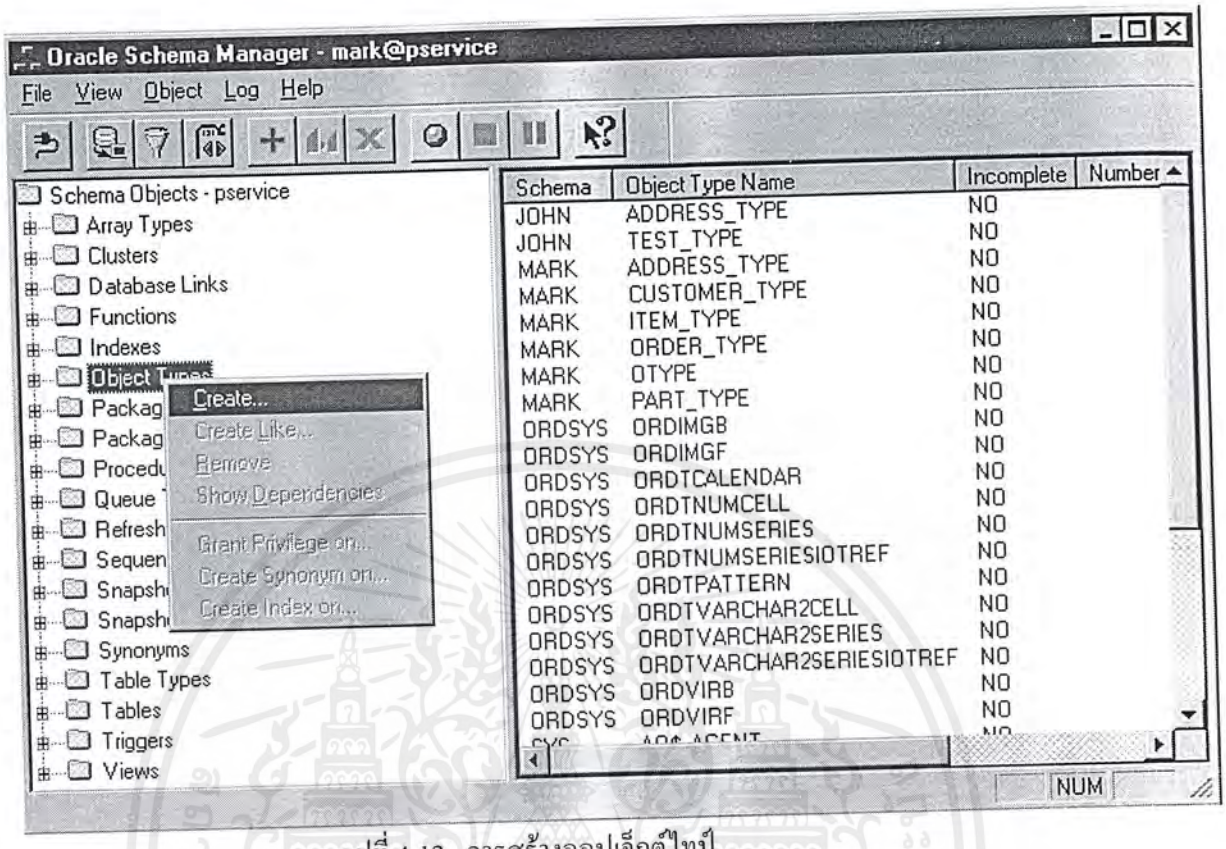
4.7 การสร้างตารางเพื่อใช้งานข้อมูลแบบ Nested Table ใน Oracle 8

1. เรียกการใช้งาน Oracle Schema Manager จะพบหน้าจอล็อกอินให้ใส่ชื่อผู้ใช้ ,รหัสผ่านและชื่อเซอร์วิสที่ต้องการติดต่อ ดังรูปที่ 4-11



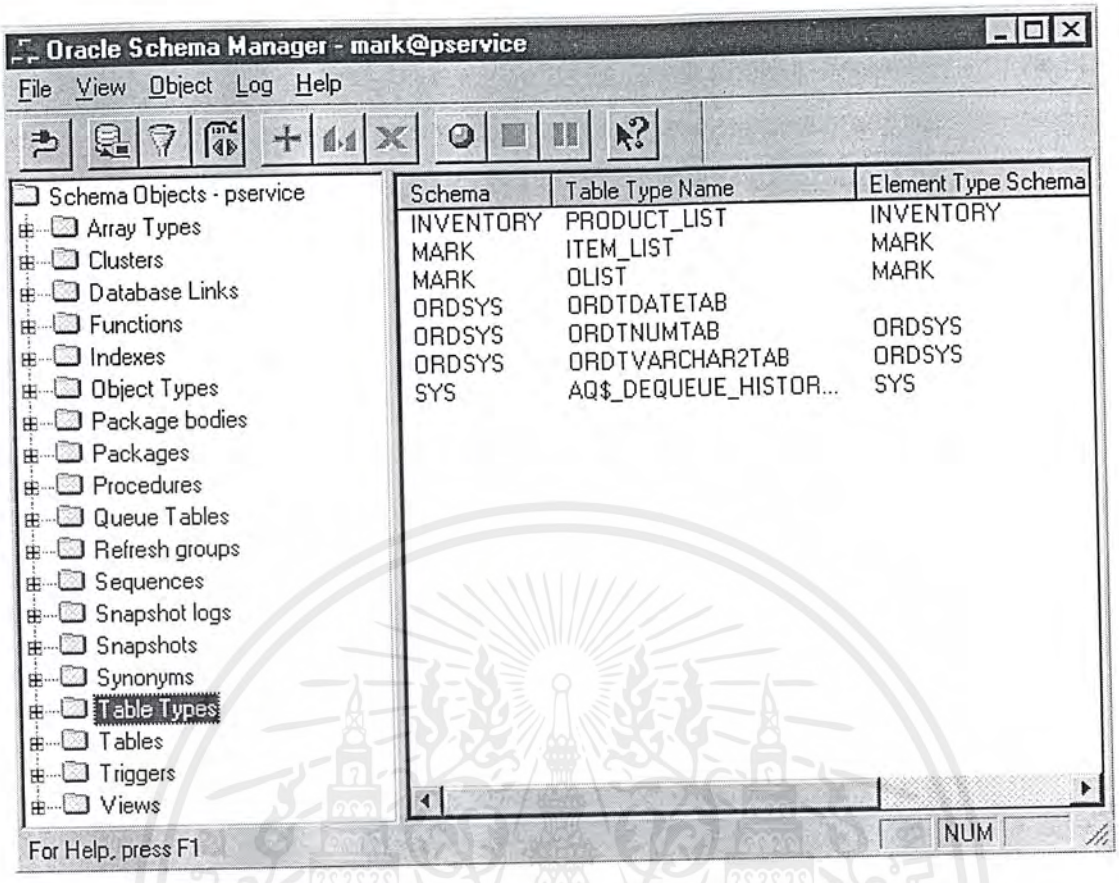
รูปที่ 4-11 หน้าจอล็อกอินก่อนเข้าใช้งาน Oracle Schema Manager

2. หลังจากผ่านการล็อกอินเข้าสู่ Oracle Schema Manager แล้ว คลิกขวาที่ Object Types เลือกเมนู Create เพื่อทำการสร้างออปเจ็กต์ไทป์ ใหม่ขึ้นมา ดังรูปที่ 4-12

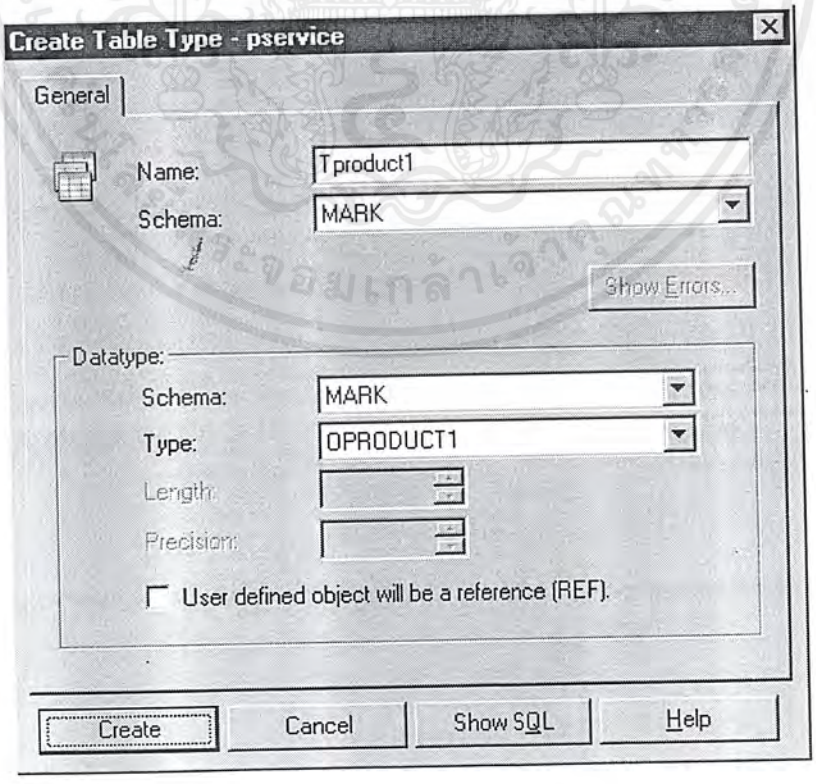


รูปที่ 4-12 การสร้างออปเจ็กต์ไทป์

3. สร้างเทเบิลไทป์ (Table Type)เพื่อกำหนดชนิดของตาราง Nested Table โดยคลิกขวาที่ Table Types เลือกเมนู Create เพื่อทำการสร้างเทเบิลไทป์ใหม่ขึ้นมา ดังรูปที่ 4-13



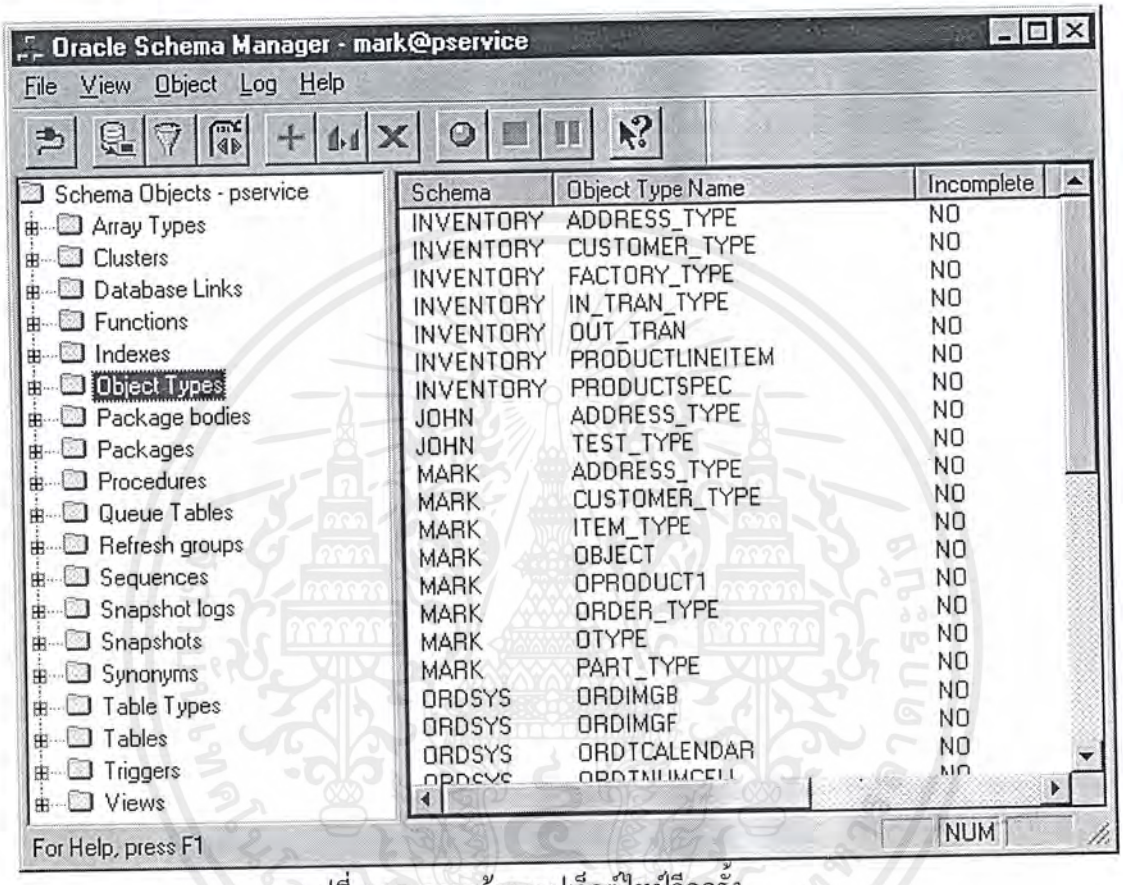
รูปที่ 4-13 การสร้างเทเบิลไทป์



รูปที่ 4-14 การกำหนดค่าให้เทเบิลไทป์ที่สร้างขึ้น

4. ตั้งชื่อให้กับเทเบิลไทป์ที่ได้สร้างขึ้น และเลือกชื่อสกีมาและชนิดของข้อมูล แล้วคลิกปุ่ม Create เพื่อยืนยันการสร้างเทเบิลไทป์ ดังรูปที่ 4-14

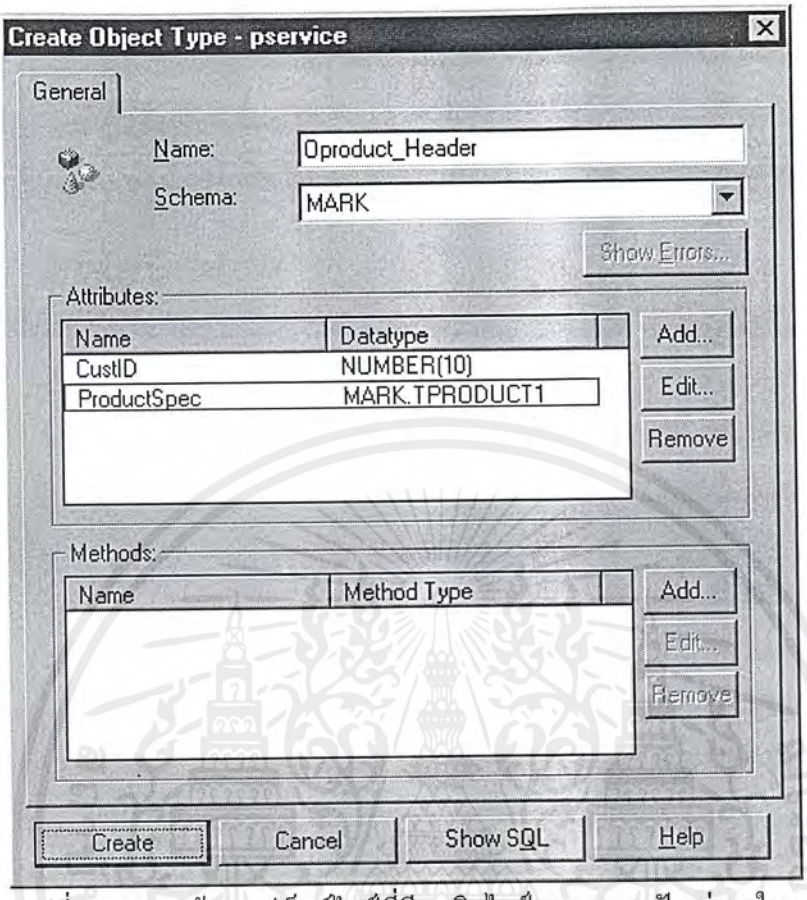
5. กลับมาสร้างออปเจ็กต์ไทป์เพื่อเก็บแอตทริบิวต์ที่เป็นชนิดของข้อมูลแบบเทเบิลไทป์ เป็นการสร้างความเป็น Nested Table



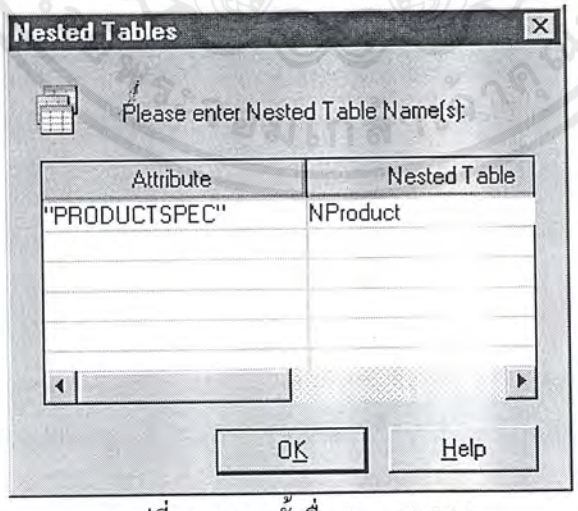
รูปที่ 4-15 การสร้างออปเจ็กต์ไทป์อีกครั้ง

6. ตั้งชื่อออปเจ็กต์ไทป์นั้น กำหนดชื่อสกีมา และกำหนดแอตทริบิวต์ ดังรูปที่ 4-16 จะเห็นได้ว่าแอตทริบิวต์ ProductSpec เป็นข้อมูลในชนิดของเทเบิลไทป์ Tproduct1

7. หลังจากที่เรากำหนดให้แอตทริบิวต์ ProductSpec เป็นข้อมูลในชนิดของเทเบิลไทป์ Tproduct1 แล้ว จะมีหน้าจอให้เราใส่ชื่อ Nested Table ดังรูปที่ 4-17



รูปที่ 4-16 การสร้างออปเจ็กต์ไทป์ที่มีเทเบิลไทป์ Tproduct1 ฝังอยู่ภายใน



รูปที่ 4-17 การตั้งชื่อ Nested Table

8. สร้างตารางทั่วไปโดยมีวิธีการตามที่กล่าวในช่วงต้น

4.8 Oracle 8 กับฐานข้อมูลเชิงวัตถุสัมพันธ์

Oracle8 สนับสนุนการพัฒนาแอพพลิเคชันฐานข้อมูลเชิงวัตถุโดยการเพิ่มความสามารถในการสร้างข้อมูลประเภท Object Type ซึ่งเป็นไปตามมาตรฐานของ ANSI / ISO SQL3 โดยข้อมูลประเภท Object Type สามารถที่จะสร้างฐานข้อมูลเชิงวัตถุสัมพันธ์แบบ hybrid และง่ายแก่การพัฒนาแอพพลิเคชัน ซึ่งมีตัวอย่างดังต่อไปนี้

```
CREATE OR REPLACE TYPE sales.part_type AS OBJECT (
        id          INTEGER,
        description         VARCHAR2(50),
        on_hand INTEGER,
        record_point        INTEGER,
        MEMBER FUNCTION  part_id (descr IN VARCHAR2) RETURN INTEGER,
        MEMBER FUNCTION parts-on_hand (part_id IN INTEGER) RETURN INTEGER,
        MEMBER PROCEDURE order_part (part-id IN INTEGER, quantity IN INTEGER),
        MEMBER PROCEDURE return_part (part_id IN INTEGER, quantity IN INTEGER)
);
```

จากตัวอย่างข้างต้น มีเหตุผลสองประการที่ใช้ Object Type ประการแรกคือ Object Type สามารถสร้าง complex datatypes ได้ซึ่งใกล้เคียงกับโลกของความเป็นจริง เช่น PART_TYPE สามารถแทนส่วนต่าง ๆ ภายในระบบออกเป็นส่วน ๆ ได้ ประการที่สองคือสามารถรวมข้อมูลเข้ากับคำสั่ง (operation) ได้ ตัวอย่างเมทธอดของ PART_TYPE

- แสดง ID number เมื่อคุณมี part's description
- แสดงสถานะของรายการสินค้าปัจจุบันเมื่อคุณมี ID number
- ลดจำนวนของสถานะของรายการสินค้าเมื่อมีการสั่งของ
- เพิ่มจำนวนของสถานะของรายการสินค้าเมื่อลูกค้าคืนสินค้า

4.8.1 คำศัพท์ที่ควรรู้

Object คือกลุ่มของข้อมูลที่มีความสัมพันธ์กันซึ่งโดยปกติในฐานข้อมูลใน Oracle มักจะแทนด้วยสิ่งที่อยู่ในโลกของความจริงเช่น บุคคล, สถานที่ เป็นต้น หรือมีความหมายอีกนัยหนึ่งคือเป็นอินสแตนซ์ของคลาสหรือออปเจ็กต์ไทป์

Attribute คือส่วนของข้อมูลที่อยู่ในออปเจ็กต์ ตัวอย่างเช่น ID, description เป็นต้น

Method คือส่วนติดต่อกับข้อมูลซึ่งจะทำการห่อหุ้ม (encapsulate) ข้อมูลไว้ทำให้ผู้ใช้ไม่จำเป็นต้องรู้รายละเอียดภายใน เพียงแต่เรียกใช้เมทธอดตามที่ต้องการเท่านั้น

Subclass คือคลาสเฉพาะของออปเจ็กต์ไทป์ที่ทำการสืบทอด (Inherit) แอตทริบิวต์และเมทธอทจากคลาสพ่อ และสามารถเพิ่มเติมคุณลักษณะเฉพาะตามที่ต้องการได้เช่นกัน (Oracle8 ไม่สนับสนุนการสืบทอดทั้งแอตทริบิวต์และเมทธอด)

4.8.2 วิธีการใช้ Object Type ในการออกแบบฐานข้อมูล

1. ใช้ User-Defined Datatypes สร้างโมเดลของ Relational Database
2. สร้าง Nested Table ในตารางของ Relational Database
3. สร้าง Object Table ซึ่งออกแบบโดยวิธีแบบ Object-Oriented Database แทนการออกแบบด้วยวิธี Relational Database

4.8.3 การใช้งาน Object Type

ต่อไปจะเป็นการกล่าวถึงเรื่องของการสร้าง Object Type ในฐานข้อมูล, วิธีการสร้างเมทธอดให้แก่ Object Type และวิธีการใช้งาน Object Type เพื่อสร้างตารางและโครงสร้างอื่น ๆ

4.8.4 ตัวอย่างของการสร้างใช้งาน Object Type

ตัวอย่างของการสร้าง ADDRESS_TYPE

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

เมื่อต้องการสร้างตารางก็เรียกใช้ ADDRESS_TYPE ได้ดังตัวอย่างต่อไปนี้

```
CREATE TABLE sales.customers (
    Id INTEGER PRIMARY KEY,
    last_name VARCHAR2(50),
    first_name VARCHAR2(50),
    company_name VARCHAR2(50),
    addresss pub.Address_Type,
    ...
```

หรือสามารถนำมาใช้สร้างพารามิเตอร์ใน Stored Procedure ได้ดังต่อไปนี้

```
CREATE OR REPLACE PROCEDURE sales.new_customer (
        custid IN INTEGER,
        last IN VARCHAR2,
        first IN VARCHAR2,
        address IN pub.Address_Type)
BEGIN
...
```

ตัวอย่างของการคิวรีในตาราง CUSTOMER ข้างต้น สามารถคิวรี ADDRESS_TYPE ได้เช่นกัน แต่ต้องมีเครื่องหมาย '.' ตามหลัง ADDRESS แล้วตามด้วยแอตทริบิวต์ ดังนี้

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

ผลที่ได้เป็นดังนี้

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

ซึ่งการ update ก็จะเป็นในลักษณะเดียวกัน

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

สำหรับการ insert แถวเข้าไปในตาราง จำเป็นต้องใช้ Constructor Method ซึ่ง Oracle จะสร้างขึ้นโดยอัตโนมัติเมื่อมีการสร้าง Object Type ดังตัวอย่างต่อไปนี้

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

4.8.5 การสร้าง Nested Table โดยใช้ภาษา SQL

ต่อไปนี้จะเป็นตัวอย่างของการเพิ่มตารางหนึ่งเข้าไปในอีกตารางหนึ่งโดยการใช้ Object Type โดย Oracle จะสร้างความสัมพันธ์ในแต่ละแถวของตัวลูกและของตัวพ่อให้โดยอัตโนมัติ ซึ่งจะช่วยลดความซับซ้อนในการ join ตาราง ดังตัวอย่างต่อไปนี้

```
CREATE OR REPLACE TYPE sales.item_type AS OBJECT (
        item_id INTEGER,
        quantity INTEGER );

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

CREATE TABLE SALES.ORDERS (
        id INTEGER PRIMARY KEY,
        order_date DATE,
        ship_date DATE,
        ship_date DATE,
        line-items sales.Item_List )
 NESTED TABLE line_items STORE AS items;
```

เมื่อเราทำการ nest ตารางหนึ่งไปยังอีกตารางหนึ่ง Oracle จะทำการเก็บข้อมูลเพียงหนึ่ง physical data segment แต่สร้างสอง logical tables ใน Data Dictionary ซึ่งการใช้งานกับภาษา DML ก็สามารถทำได้ดังนี้

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

เมื่อเราต้องการใช้งาน row ใน Nested Table เราจำเป็นต้องใช้ SQL ที่เรียกว่า Flattened Subquery โดยการใช้คำสั่ง THE ดังตัวอย่างต่อไปนี้

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

นอกจากนั้นในการเรียก row จาก Nested Table ขึ้นมาดู ก็จำเป็นต้องใช้ Flattened Subquery เช่นกัน

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

ITEM_ID	QUANTITY
1	22
2	100
3	200

และเมื่อต้องการลบแถวใน Nested Table ก็สามารถทำได้ดังนี้

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

4.8.6 การสร้าง Object Table โดยใช้ภาษา SQL

ต่อไปนี้จะเป็นการสร้างตารางแบบออปเจ็กต์ซึ่งเรียกว่า Object Table ซึ่งเป็นตารางฐานข้อมูลที่ประกาศใช้ข้อมูลแบบ Object Type เท่านั้น ไม่ใช่ Relational Column เมื่อเราสร้างตารางเป็นแบบออปเจ็กต์แล้วคอลัมน์จะเป็นแอททริบิวต์ของ Object Type ซึ่งเราจะนำมาสร้างตาราง

แถวในตารางออปเจ็กต์ก็จะเป็นออปเจ็กต์ของ Table Type โดยแต่ละออปเจ็กต์ในตารางออปเจ็กต์จะมี Object Identifier (OID) ซึ่งจะไม่ซ้ำกับใคร Oracle จะใช้ OID นี้สร้างความสัมพันธ์ระหว่างตารางออปเจ็กต์ในตาราง

ตัวอย่างต่อไปนี้จะเป็นการสร้างตารางออปเจ็กต์ของ CUSTOMER

```
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);

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

ต่อไปจะเป็นตัวอย่างการสร้างตารางออปเจ็กต์ของ PARIS

```
CREATE OR REPLACE TYPE sales.part_type AS OBJECT (
        id INTEGER,
        descriptiton VARCHAR2(50),
        unit_price NUMBER(10,2),
        on_hand INTEGER,
        reorder_point INTEGER);

CREATE TABLE sales.parts OF sales.Part_Type
        (id PRIMARY KEY);
```

ต่อไปเป็นตัวอย่างของการสร้างตารางออปเจ็กต์ของ ORDERS ซึ่งจะ nest ตาราง ITEMS เข้าไป

```
CREATE OR REPLACE TYPE sales.item_type AS OBJECT (
        item_id INTEGER,
        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 dales.orders OF sales.Order_Type
        (id PRIMARY KEY)
NESTED TABLE line_items STORE AS items;
```

สังเกตได้ว่า ITEM_TYPE และ ORDER_TYPE จะมี Object Reference ซึ่งแอททริบิวต์ PART จะเป็น Reference Object ของ PART_TYPE และแอททริบิวต์ CUSTOMER จะเป็น Reference Object ของ CUSTOMER_TYPE ซึ่งมีลักษณะคล้ายกับ Foreign Key ใน Relational Database Table ซึ่ง Object Reference ก็คือแอททริบิวต์ของออปเจ็กต์ซึ่งชี้ไปยังออปเจ็กต์อื่นในฐานข้อมูลโดยใช้ OID

ต่อไปจะเป็นตัวอย่างของ Constructor Method ในการ insert ข้อมูลไปยังตาราง PARTS และ CUSTOMERS

```
INSERT INTO sales.parts
        VALUES (sales.Part_type(1,'Pentium 200 CUU', 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')));
```

ซึ่งการใช้งาน Object Reference จะต้องใช้คำสั่ง REF ซึ่งเป็นฟังก์ชันที่จะให้ค่า Reference ที่ชี้ไปยัง OID ซึ่งการใช้งานต้องใช้ Subquery เพื่อสร้างค่าให้แก่ออปเจ็กต์ใหม่

```
INSERT INTO sales.orders
        SELECT 1, REF(c ), SYSDATE, NULL, sales.Item_List()
        WHERE id=1;
```

และต้องใช้ Flattened Subquery ด้วยเช่นกัน

```
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;
...
```

สำหรับ PL/SQL ก็สามารถใช้ Object Reference ได้ดังตัวอย่างต่อไปนี้

```
DECLARE
        custold REF sales.Customer_Type;
        partoid REF sales.Part_Type;
BEGIN
        SELECT REF(c ) INTO custoid FROM sales.customers c
        WHERE c.last_name = 'Ellison'
        AND c.first_name = 'Lawrence';
```

```
SELECT REF(p) INTO partoid FROM sales.parts p
        WHERE p.description = 'Pentium 200 CPU';

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(-20000, 'No data found');
END;
```

ประโยชน์ของ Object Tables, OIDs และ Object Reference ทำให้สามารถไม่ต้องทำการ join เหมือนใน Relational Models

```
SELECT o.id, o.customer.company_name
        FROM sales.orders o;
```

ID	CUSTOMER.COMPANY_NAME
1	Oracle Corporation

ใน Relational System ผู้พัฒนาระบบจำเป็นต้องรู้ความสัมพันธ์ระหว่างสองตารางแล้วทำการคิวรีข้อมูลเพื่อทำการ join ดังนี้

```
SELECT o.id, c.company_name
        FROM slaes.orders o, sales.customers c
        WHERE o.cust_id = c.id;
```

แต่ใน Object Tables เพียงแต่ใช้ Nested Table ก็สามารถคิวรีข้อมูลมาได้

```
SELECT o.item_id, o.part.description, o.quantity
        FROM THE(SELECT line_items FROM sales.orders WHERE id=1) o;
```

ITEM_ID	PART.DESCRIPTION	QUANTITY
1	Pentium 200 CPU	50

4.8.7 เกี่ยวกับ Methods

เราสามารถสร้างเมทธอดใน Object Type โดยเมทธอดจะเปรียบเสมือนฟังก์ชันแต่สามารถทำการเอนแคปซูเลตข้อมูลได้ (ใน Oracle8 เวอร์ชันแรก ยังทำไม่ได้) โดยคอนสตรักเตอร์ของออปเจ็กต์จะสร้างโดยอัตโนมัติซึ่งเป็นชื่อเดียวกับชื่อ Object Type

ส่วนการสร้าง Member Method จะมีลักษณะคล้ายกับ Stored Procedure หรือฟังก์ชันซึ่งติดอยู่กับ Object Type ดังตัวอย่างต่อไปนี้

```
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.Items_List,
MEMBER FUNCTION order_total RETURN NUMBER,
PRAGMA RESTRICT_REFERENCES(order_total, WNDS, WNPS) );

CREATE TABLE sales.orders OF sales.Order_Type
        (id PRIMARY KEY)
NESTED TABLE line_items STORE AS items;
```

สังเกตได้ว่า Member Method จะมีลักษณะคล้ายกับ PL/SQL ดังต่อไปนี้

- แต่ละเมทธอดมีหลายพารามิเตอร์หรือมีพารามิเตอร์เดียวก็ได้
- Member Function ต้องประกาศให้มีค่าเดียว
- แต่ละเมทธอดจะต้องมี Pragma Compiler Directive

เราสามารถใช้ Member Method ในส่วนของ Body ได้ดังตัวอย่างต่อไปนี้

```
CREATE OR REPLACE TYPE BODY sales.order_type (
        MEMBER FUNCTION order_total RETURN NUMBER IS
        return-value NUMBER;
BEGIN
        SELECT SUM(1.quantity * 1.part.unit_price) INTO return_value
        FROM THE(SELECT o.line_items FROM sales.orders o WHERE o.id = SELF.id) 1;
        RETURN return_value:
END order_total;
);
```

ตัวอย่างต่อไปจะเป็นการคิวรี order_total เมื่อ id มีค่าเท่ากับ 1

```
SELECT o.order_total()
        FROM sales.orders o WHERE id=1;

O.ORDER_TOTAL
-----------------
             5000
```

สังเกตได้ว่า order_total method มีลักษณะเดียวกับตัวอย่างต่อไปนี้

```
SELECT SUM(i.quantity * p.unit_price)
        FROM sales.items I, sales.parts p
        WHERE i.order_id = 1
        AND     i.part_id = p.id;
```

4.8.8 Object Views of Object Tables

เมื่อเราทำการสร้างตารางออปเจ็กต์เราก็สามารถสร้าง Object View ได้เช่นกันโดยในอันดับแรกเราต้องมี Object Type ที่บรรยายลักษณะของ View ดังตัวอย่างต่อไปนี้

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

CREATE OF 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;

SELECT * FROM customer_orders;
```

	ORDER_COUNT	COMPANY
...	31	Oracle Corporation

บทที่ 5
สรุป

5.1 บทสรุป

ในการวิเคราะห์ ,ออกแบบ และพัฒนาโปรแกรมเชิงวัตถุนั้น ถึงแม้ว่าจะเป็นแนวคิดที่ค่อนข้างใหม่สำหรับคนทั่วไป แต่เมื่อมองถึงการใช้งานในระยะยาวแล้ว การพัฒนาโปรแกรมเป็นเชิงวัตถุ มีจุดดีกว่าการพัฒนาโปรแกรมแบบโครงสร้างอยู่มาก ทั้งในด้านการแบ่งส่วนการทำงานของโปรแกรมที่ค่อนข้างอิสระจากกัน และง่ายต่อการบำรุงรักษา และปรับปรุงแก้ไขโปรแกรมในอนาคต ซึ่งแทบจะทำไม่ได้เลยหรือทำได้น้อยมากสำหรับการโปรแกรมแบบโครงสร้าง

รวมถึงความสามารถในการจัดเก็บข้อมูลของระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ ที่มีมีการจัดเก็บและเรียกค้นข้อมูลได้อย่างมีประสิทธิภาพมากขึ้น และยังมีความสะดวกในการแก้ไขรายละเอียดในภายหลังตามหลักการเชิงวัตถุสัมพันธ์ (Object-Relational) ด้วย

อีกทั้งความสามารถของสถาปัตยกรรม 3-เทียร์ ไคลเอนต์ / เซิร์ฟเวอร์ ที่แยกเซิร์ฟเวอร์ออกเป็น 2 ส่วน ได้แก่ แอพพลิเคชันเซิร์ฟเวอร์ และดาต้าเบสเซิร์ฟเวอร์ ทำให้เซิร์ฟเวอร์แต่ละตัวมีประสิทธิภาพในการทำงานของตัวเองมากขึ้น รวมถึงความสะดวกในการใช้งานผ่านไคลเอนต์ ที่ไม่จำเป็นต้องลงโปรแกรมใดๆ ในการเชื่อมต่อกับฐานข้อมูลในดาต้าเบสเซิร์ฟเวอร์เลย เพียงแค่รันไฟล์สกุล exe เพียงไฟล์เดียวก็สามารถทำงานได้อย่างสมบูรณ์

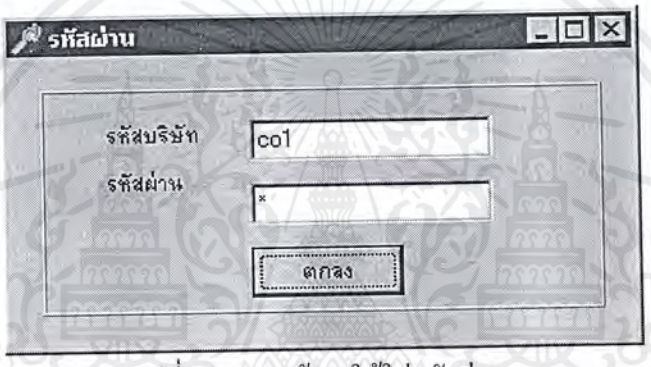
5.2 ปัญหาและอุปสรรคในการจัดทำโครงงาน

1. สถาปัตยกรรม 3-เทียร์ ไคลเอนต์/เซิร์ฟเวอร์ เป็นหลักการที่ค่อนข้างใหม่มาก ทำให้ยากต่อการหาข้อมูลเพื่อทำการทดลองในแต่ละขั้นตอน
2. การวิเคราะห์และออกแบบโปรแกรมเชิงวัตถุ ยังไม่มีโมเดลที่เป็นมาตรฐานให้เลือกใช้
3. ระบบจัดการฐานข้อมูล Oracle เวอร์ชัน 8.0.4 ยังไม่รองรับการทำงานเชิงวัตถุอย่างสมบูรณ์
4. เครื่องแอพพลิเคชันเซิร์ฟเวอร์ต้องใช้รีซอร์สมาก จึงต้องหาเครื่องที่มีสเป็กค่อนข้างสูง และมีหน่วยความจำ (RAM) มาก

ภาคผนวก ก
คู่มือการใช้งานโปรแกรมระบบงานคลังสินค้าสำหรับผู้ใช้งาน

เมื่อมีการเริ่มการทำงานของโปรแกรมไคลเอนต์ตัวแรก จะมีการติดต่อไปยังเครื่องแอพพลิเคชันเซิร์ฟเวอร์ ให้เริ่มการทำงานโปรแกรมทางด้านเซิร์ฟเวอร์โดยอัตโนมัติ

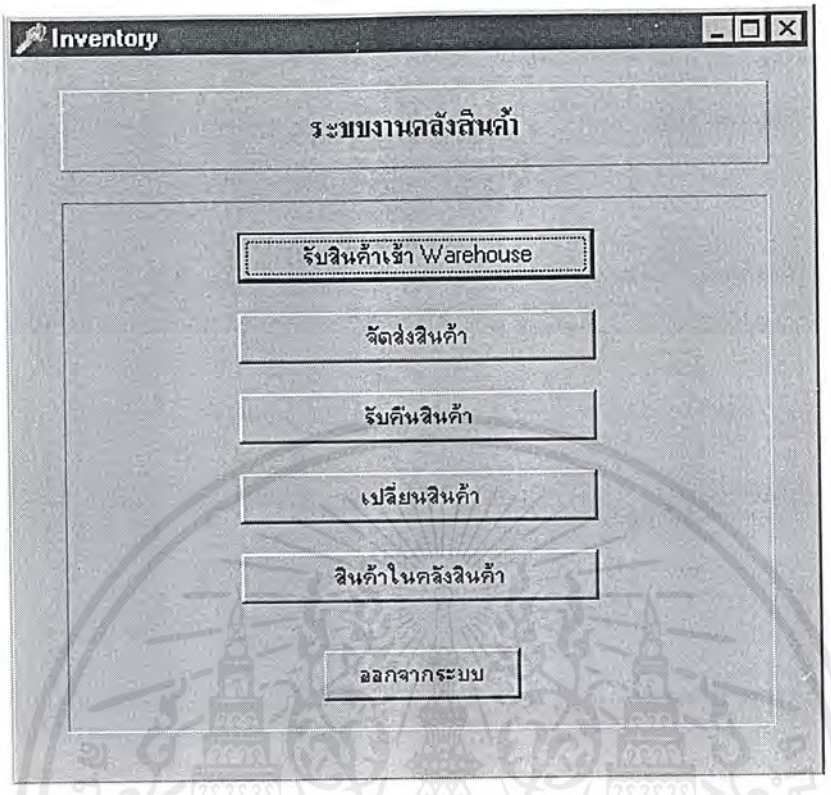
การทำงานของโปรแกรมในส่วนของไคลเอนต์ จะเริ่มจากหน้าจอการกรอกรหัสบริษัท และรหัสผ่านของบริษัท เพื่อตรวจสอบสิทธิการใช้งาน ตัวอย่างเช่น บริษัท "co1" และรหัสผ่าน "o" โดยการกรอกรหัสผ่านนั้นจะไม่แสดงตัวรหัสผ่านจริง แต่จะแสดงเพียง เครื่องหมาย "*" เท่านั้น ดังรูปที่ 1



รูปที่ 1 แสดงหน้าจอให้ใส่รหัสผ่าน

เมื่อกรอกรหัสบริษัทและรหัสผ่านถูกต้องแล้วจะเข้าสู่หน้าจอเมนูหลักของระบบงานคลังสินค้า โดยจะแบ่งเป็น 5 รายการ ได้แก่

1. การรับสินค้าเข้าสู่คลังสินค้า
2. การจัดส่งสินค้า
3. การรับคืนสินค้า
4. การเปลี่ยนสินค้า
5. การแสดงและเพิ่มเติมรายละเอียดของชนิดของสินค้าที่อยู่ในคลังสินค้า



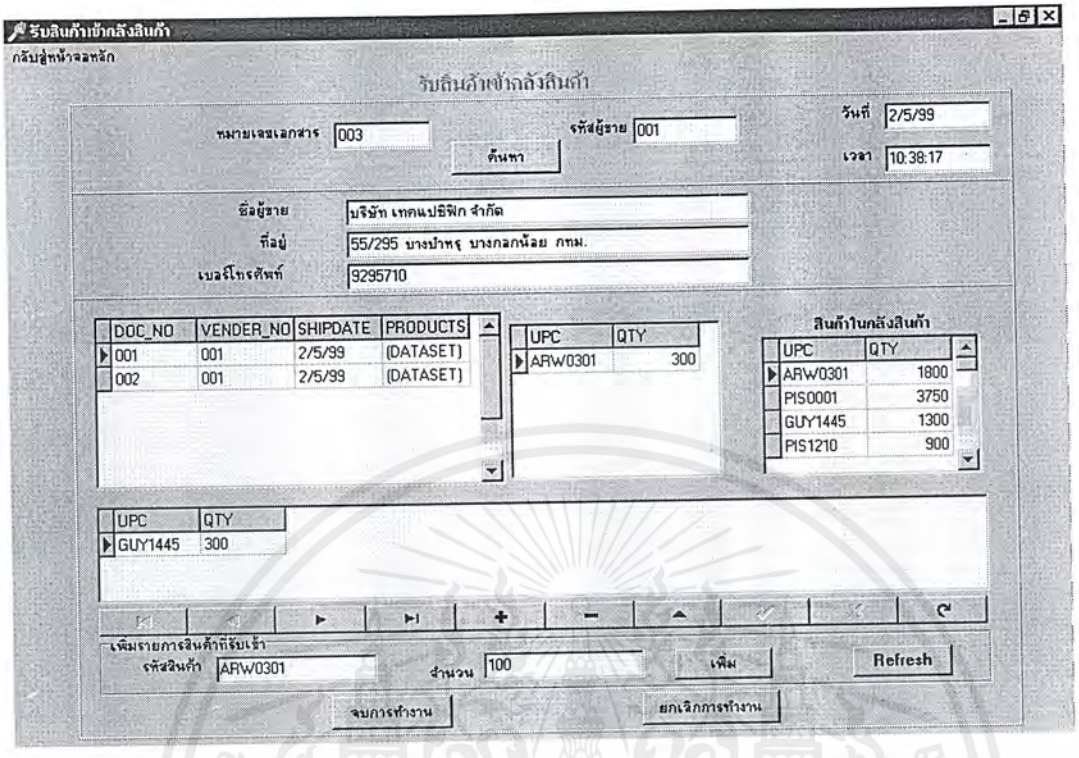
รูปที่ 2 แสดงหน้าจอหลัก หลังการล็อกอิน

1. การรับสินค้าเข้าสู่คลังสินค้า

เมื่อผู้ใช้เลือกรายการรับสินค้าเข้าสู่คลังสินค้าแล้ว หน้าจอจะสลับเข้าสู่เมนูย่อยของรายการรับสินค้า โดยผู้ใช้จะต้องกรอก "หมายเลขเอกสาร" ซึ่งจะเป็นหมายเลขของเอกสารที่ส่งเข้ามา โดยหมายเลขนี้จะต้องไม่มีการซ้ำกันเลย ต่อจากนั้นก็กรอก "รหัสของผู้ขาย" ที่เป็นผู้ส่งสินค้านั้นเข้ามาสู่คลังสินค้า เมื่อกรอกเสร็จแล้วให้ผู้ใช้กดปุ่ม "ค้นหา"

เมื่อกดปุ่มค้นหาแล้วจะแสดงรายละเอียดของผู้ขายตาม รหัสของผู้ขาย ที่ได้กรอกมาเพื่อตรวจสอบว่ากรอกถูกต้องหรือไม่ แสดงรายการการรับสินค้า เรียงตามลำดับ พร้อมแสดงรายละเอียดของการรับแต่ละครั้งรวมถึงจำนวนสินค้าแต่ละชนิดที่คงเหลือในคลัง

ในส่วนรายการ "เพิ่มรายการสินค้าที่รับเข้า" จะให้ผู้ใช้กรอกรายการของสินค้าที่รับเข้ามา โดยผู้ใช้จะต้องกรอก "รหัสสินค้า" และ "จำนวน" ของสินค้า เมื่อกรอกแต่ละรายการเสร็จ ให้กดปุ่ม "เพิ่ม" และเมื่อกรอกทุกรายการเสร็จแล้ว ก็ให้กดปุ่ม "จบการทำงาน" หรือถ้าหากมีการกรอกผิดพลาดก็สามารถกดปุ่ม "ยกเลิกการทำงาน" ได้ แล้วกลับสู่เมนูหลักโดยเลือกเมนู "กลับสู่หน้าจอหลัก"

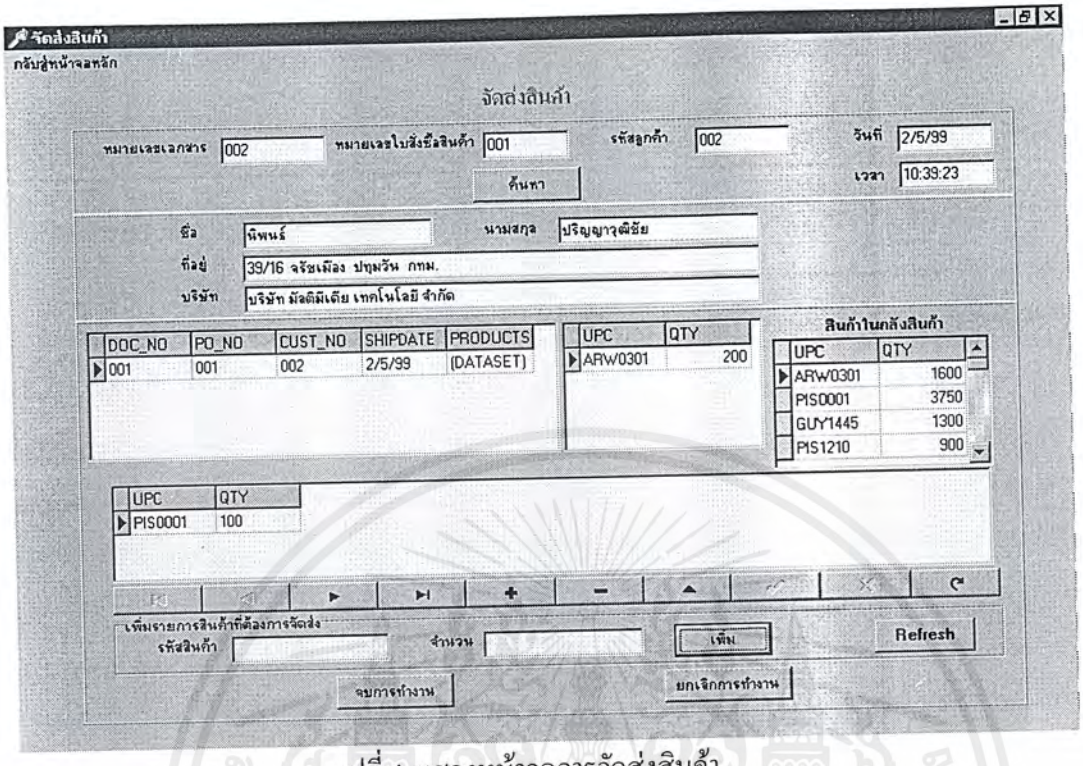


รูปที่ 3 แสดงหน้าจอการรับสินค้าเข้าคลังสินค้า

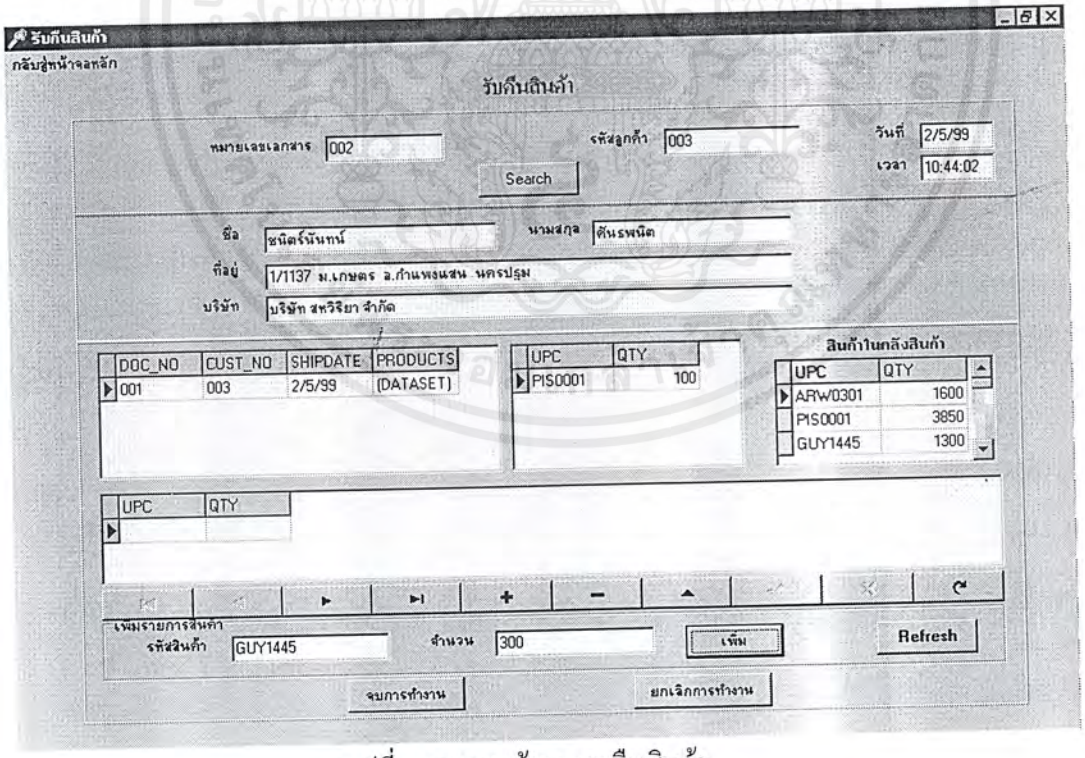
2. การจัดส่งสินค้า

หน้าจอและหลักการใช้งาน ของเมนูย่อยนี้จะมีลักษณะคล้ายกับการรับสินค้าเข้า โดยผู้ใช้จะต้องกรอก "หมายเลขเอกสาร" โดยหมายเลขนี้จะเป็นหมายเลขของเอกสารการจัดส่งสินค้าซึ่งเป็นคนละอันกับการรับสินค้าเข้าแต่จะไม่มีการซ้ำเช่นเดียวกัน แล้วให้ผู้ใช้จะกรอก "หมายเลขใบสั่งซื้อสินค้า" และ "รหัสลูกค้า" แล้วกดปุ่มค้นหา

เมื่อกดปุ่มค้นหาแล้ว จะแสดงรายการซึ่งมีรายละเอียดเช่นเดียวกับการรับสินค้าเข้า โดยผู้ใช้จะต้องกรอก "รหัสสินค้า" และ "จำนวน" ในส่วนรายการ "เพิ่มรายการสินค้าที่ต้องการจัดส่ง" แล้วกดปุ่ม "เพิ่ม" ทีละรายการจนเสร็จ แล้วกดปุ่มจบการทำงาน เลือกเมนู "กลับสู่หน้าจอหลัก" เพื่อกลับไปยังหน้าจอแรก



รูปที่ 4 แสดงหน้าจอการจัดส่งสินค้า



รูปที่ 5 แสดงหน้าจอการคืนสินค้า

3. การรับคืนสินค้า

จะมีลักษณะการทำงานกลับกันกับ การจัดส่งสินค้า โดยผู้ใช้จะต้องกรอก "หมายเลขเอกสาร" ซึ่งเป็นคนละอันกับหมายเลขของการจัดส่งสินค้า และ "รหัสลูกค้า" แล้วกดปุ่ม "ค้นหา"

เมื่อกดปุ่ม "ค้นหา" แล้ว จะแสดงรายการเช่นเดียวกับการจัดส่งสินค้า โดยจะต้องกรอก "รหัสสินค้า" และ "จำนวน" ที่จะคืน ในรายการย่อย "เพิ่มรายการสินค้า" เมื่อเสร็จแล้วกดปุ่ม "จบการทำงาน" แล้วเลือก เมนู "กลับสู่หน้าจอหลัก"

4. การเปลี่ยนสินค้า

การเปลี่ยนสินค้าจะมีหลักการทำงานเหมือนกับการรับสินค้าคืนแล้วจัดส่งสินค้าใหม่ออกไปในเวลาเดียวกัน โดยเมื่อผู้ใช้กรอก "หมายเลขเอกสาร" และ "รหัสลูกค้าแล้ว" กดปุ่ม "ค้นหา"

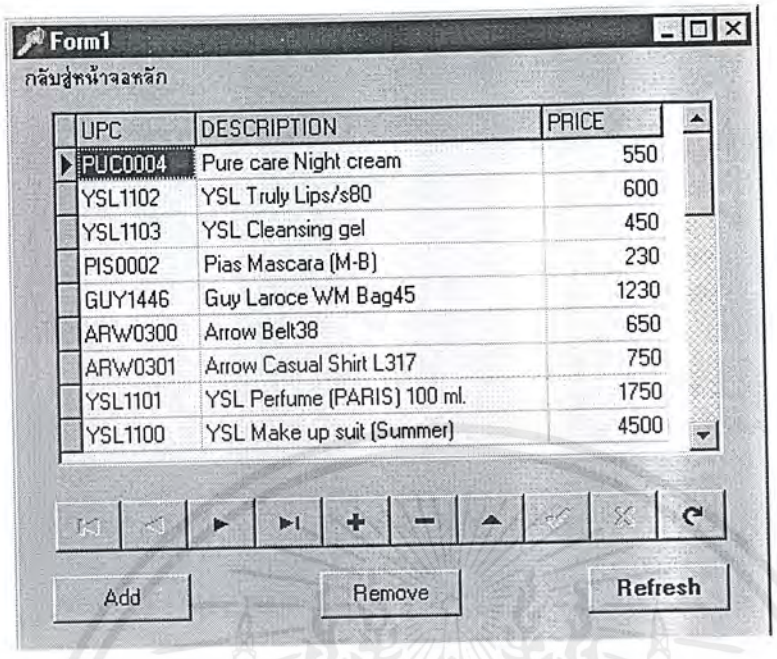
หน้าจอจะแบ่งออกเป็นสองส่วนคือ รายการสินค้าที่ส่งคืนมา และรายการสินค้าที่จะจัดส่งกลับไปให้ลูกค้า โดยผู้ใช้จะต้องกรอก "รหัสสินค้า" และ "จำนวน" ที่ได้รับคืนมา และกดปุ่ม "เข้า" แล้วกรอก "รหัสสินค้า" และ "จำนวน" ของสินค้าที่จะส่งกลับไปให้ลูกค้า แล้วกดปุ่ม "ออก" จนครบทุกรายการแล้ว กดปุ่ม "จบการทำงาน" แล้วเลือกเมนู "กลับสู่หน้าจอหลัก"

5. การแสดงและเพิ่มเติมรายละเอียดของชนิดของสินค้าที่อยู่ในคลังสินค้า

รายการย่อยนี้จะแสดงรายการของสินค้าที่มีอยู่ในคลังสินค้า โดยแสดงรายละเอียด ว่ามีสินค้า มีรหัสอะไร(UPC) มีชื่อเต็มว่าอะไร(DESCRIPTION) และมีราคาเท่าไร(PRICE)

ถ้าหากต้องการเพิ่มชนิดของสินค้า ผู้ใช้จะต้องเลื่อนแถบสว่างไปที่ช่องว่างท้ายตาราง แล้วกรอกรายละเอียด แล้วกดปุ่ม "Add"

ถ้าหากต้องการลบชนิดของสินค้า ผู้ใช้จะต้องเลื่อนแถบสว่างไปยังรายการที่ต้องการลบ แล้วกดปุ่ม "Remove"



รูปที่ 6 แสดงรายละเอียดของสินค้าที่มีอยู่ในคลังสินค้าในเวลาปัจจุบัน

6. การออกจากโปรแกรม

การออกจากโปรแกรมทำโดยกลับไปยังหน้าจอหลักเลือกรายการ "ออกจากระบบ" เมื่อผู้ใช้ต้องการเลิกทำงาน โดยเมื่อผู้ใช้คนสุดท้ายออกจะโปรแกรม โปรแกรมที่เครื่องเซิร์ฟเวอร์ก็จะจบการทำงานด้วย

ภาคผนวก ข

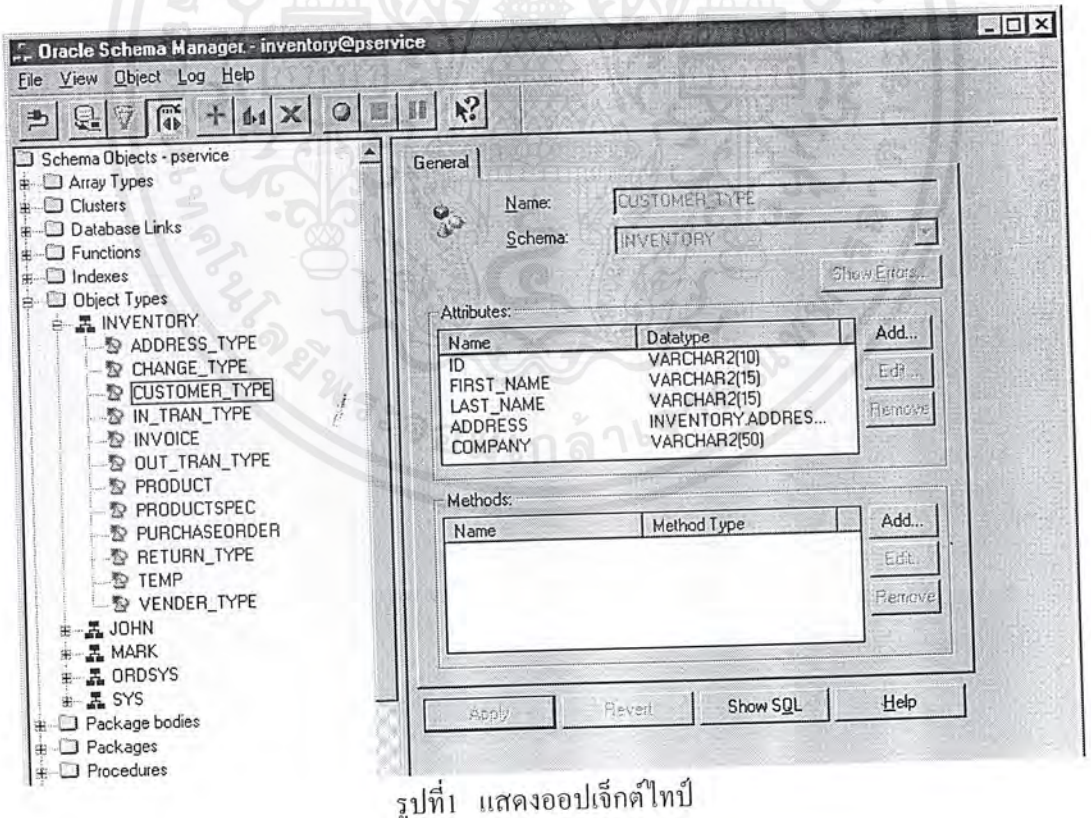
คู่มือการใช้งานโปรแกรมระบบงานคลังสินค้าสำหรับนักพัฒนาโปรแกรม

โปรแกรมระบบงานคลังสินค้าเป็นระบบงานที่แบ่งการทำงานออกเป็น 3 ส่วนคือ

1. ส่วนของดาต้าเบสเซิร์ฟเวอร์ (Database Server)
2. ส่วนของแอพพลิเคชันเซิร์ฟเวอร์ (Application Server)
3. ส่วนของไคลเอนต์ (Client)

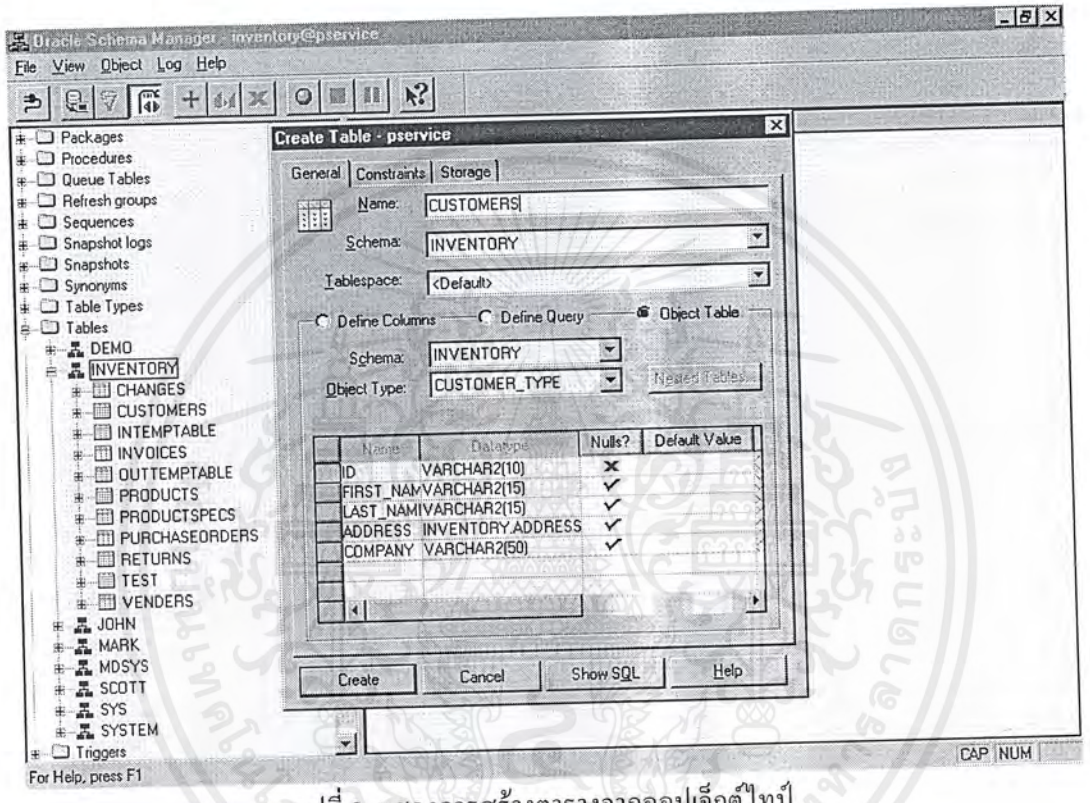
ส่วนที่1 : ดาต้าเบสเซิร์ฟเวอร์

ส่วนนี้เป็นส่วนสำคัญในการสร้างระบบฐานข้อมูลโดยในระบบงานคลังสินค้าจะใช้ ระบบจัดการฐานข้อมูล Oracle8 เป็นดาต้าเบสเซิร์ฟเวอร์ ซึ่งสามารถสร้างฐานข้อมูลที่มีชนิดของข้อมูลเป็นแบบออปเจ็กต์ไทป์ (Object Type) ได้ ดังรูปที่ 1



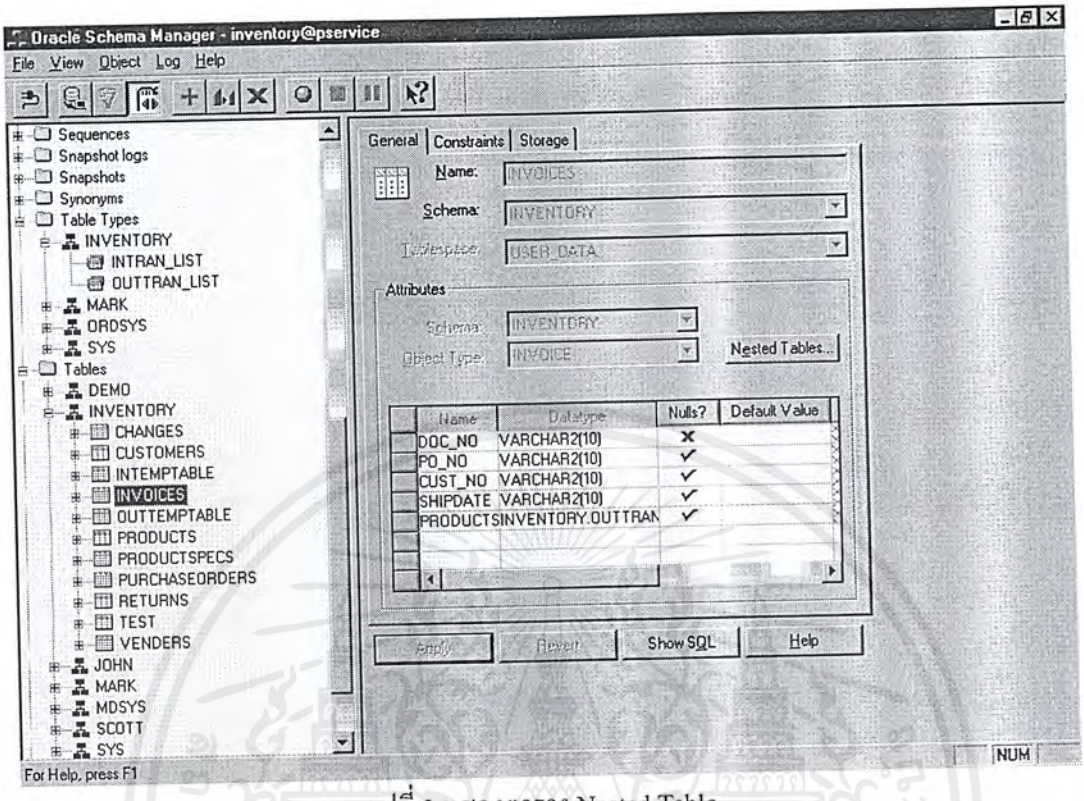
รูปที่1 แสดงออปเจ็กต์ไทป์

ในเบื้องต้นจะต้องทำการสร้างออปเจ็กต์ไทป์โดยการกำหนดแอตทริบิวต์ (Attribute) ต่างๆ ซึ่งสามารถมีข้อมูลแบบบิวท์อินดาต้าไทป์ (Built-In Datatype) หรือแบบยูเซอร์ดีฟายดาต้าไทป์ (User Defined Datatype) ก็ได้ หลังจากนั้นจึงนำออปเจ็กต์ไทป์เหล่านี้มากำหนดเป็นตาราง โดยการสร้างตารางจากออปเจ็กต์ไทป์ดังรูปที่ 2



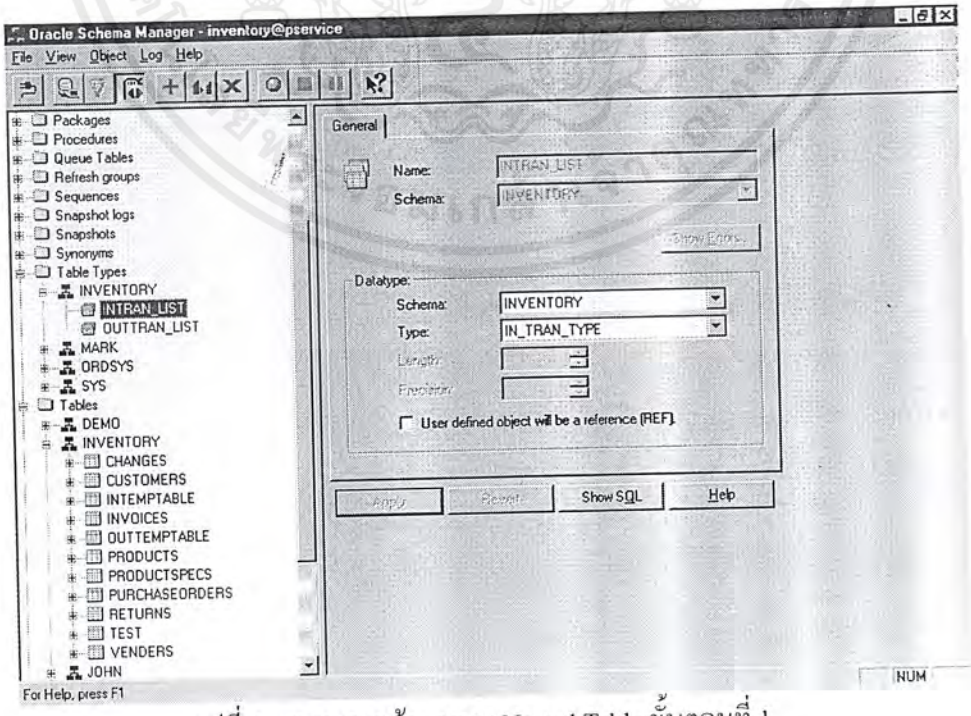
รูปที่ 2 แสดงการสร้างตารางจากออปเจ็กต์ไทป์

แต่ในระบบฐานข้อมูล Oracle 8 สามารถที่จะสร้างตารางที่เป็นเน็สเต็ดเทเบิล (Nested Table) ได้ดังรูปที่ 3

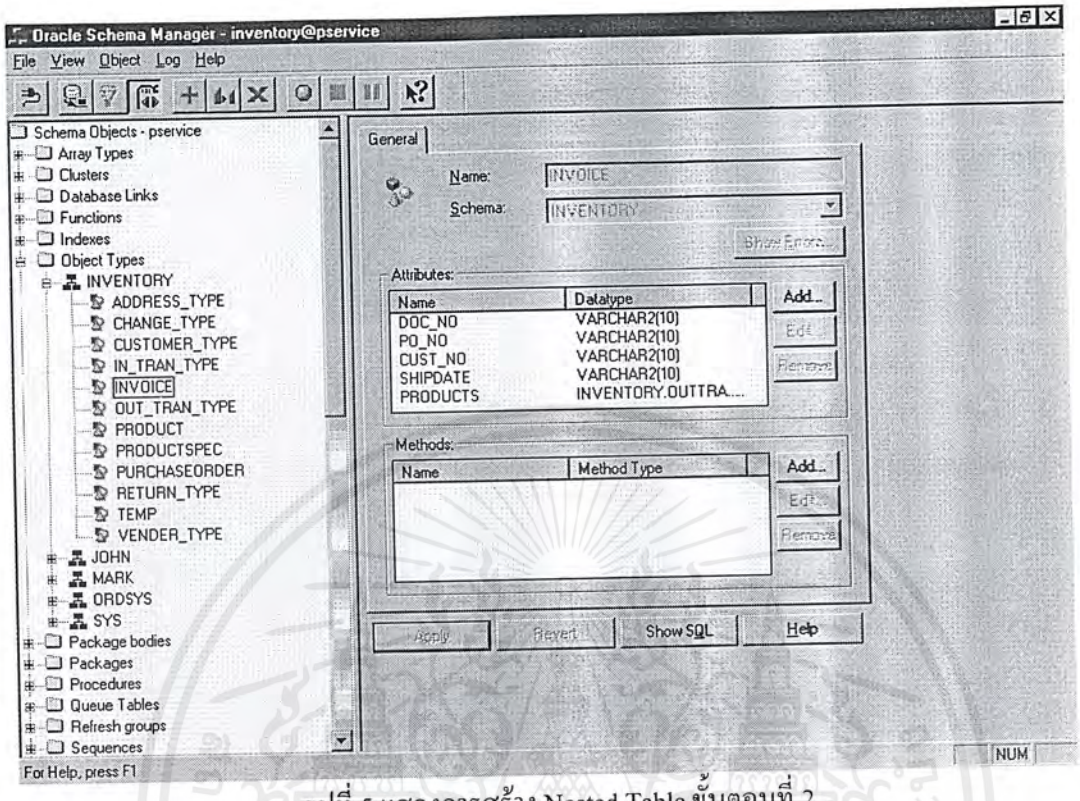


รูปที่ 3 แสดงตาราง Nested Table

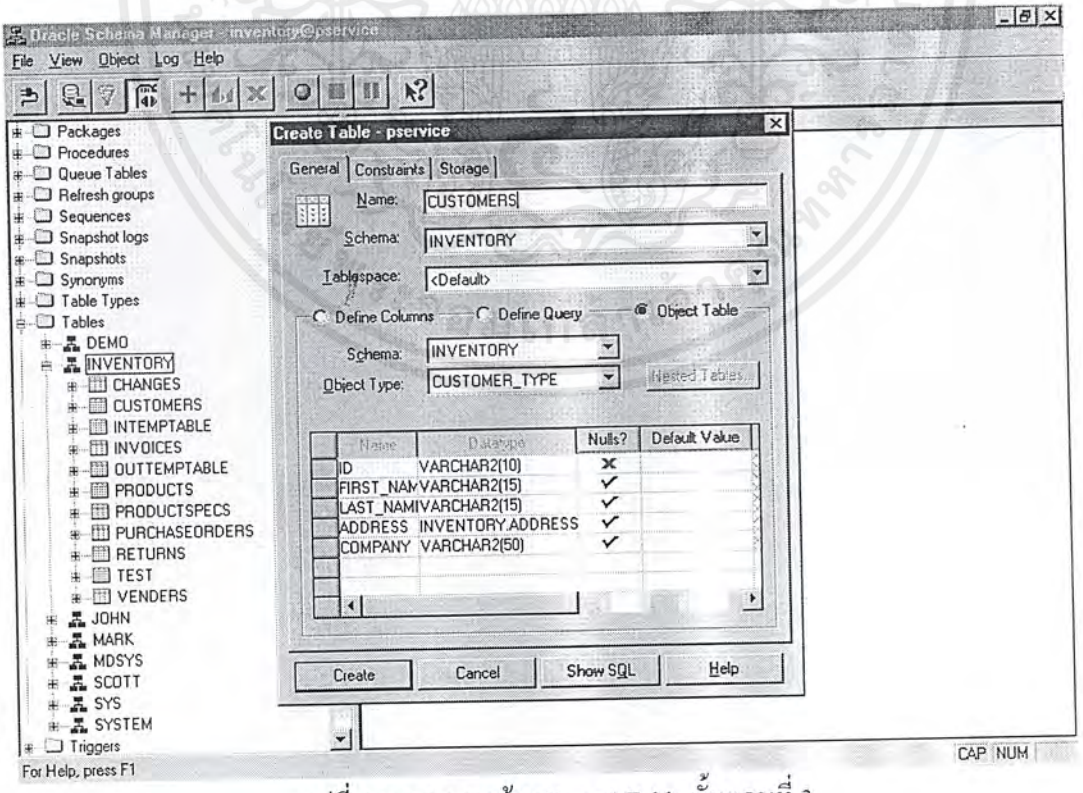
โดยในการสร้างตารางที่เป็นเน็สเต็ดเทเบิล จำเป็นต้องกำหนดประเภทของตารางที่เป็นเทเบิลไทป์ และจึงนำมาสร้างเป็นตารางแบบเน็สเต็ดเทเบิล ดังลำดับขั้นดังต่อไปนี้



รูปที่ 4 แสดงการสร้างตาราง Nested Table ขั้นตอนที่ 1



รูปที่ 5 แสดงการสร้าง Nested Table ขั้นตอนที่ 2

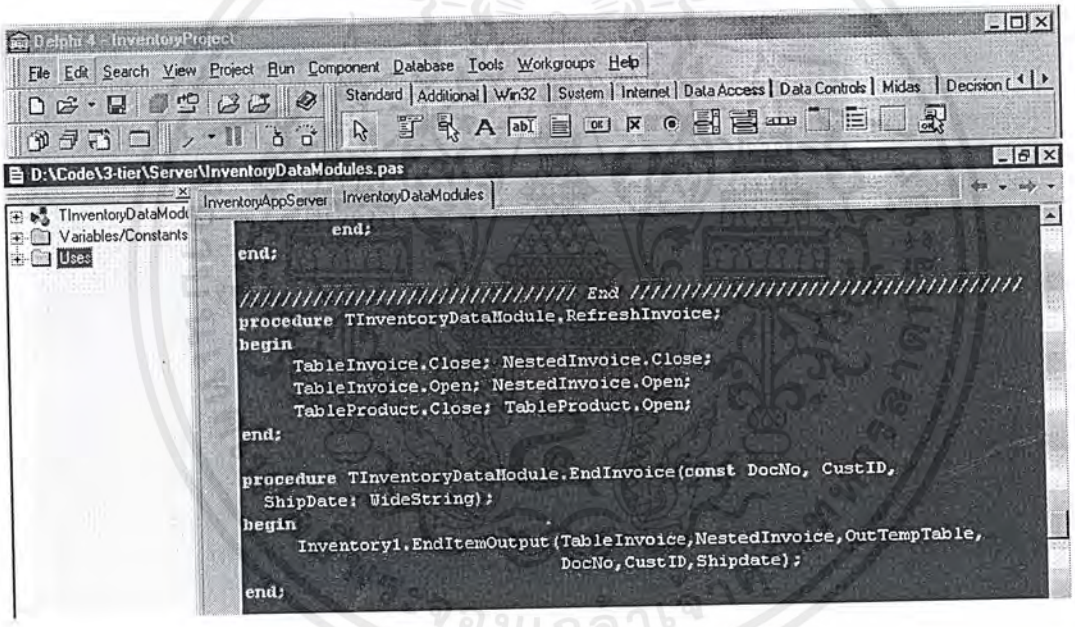


รูปที่ 6 แสดงการสร้าง Nested Table ขั้นตอนที่ 3

หลังจากได้ฐานข้อมูลครบทุกตารางแล้ว ก็จะเป็นส่วนของแอพพลิเคชันซึ่งแบ่งการทำงานเป็น 2 ส่วนคือ แอพพลิเคชันเซิร์ฟเวอร์ และ ไคลเอนต์

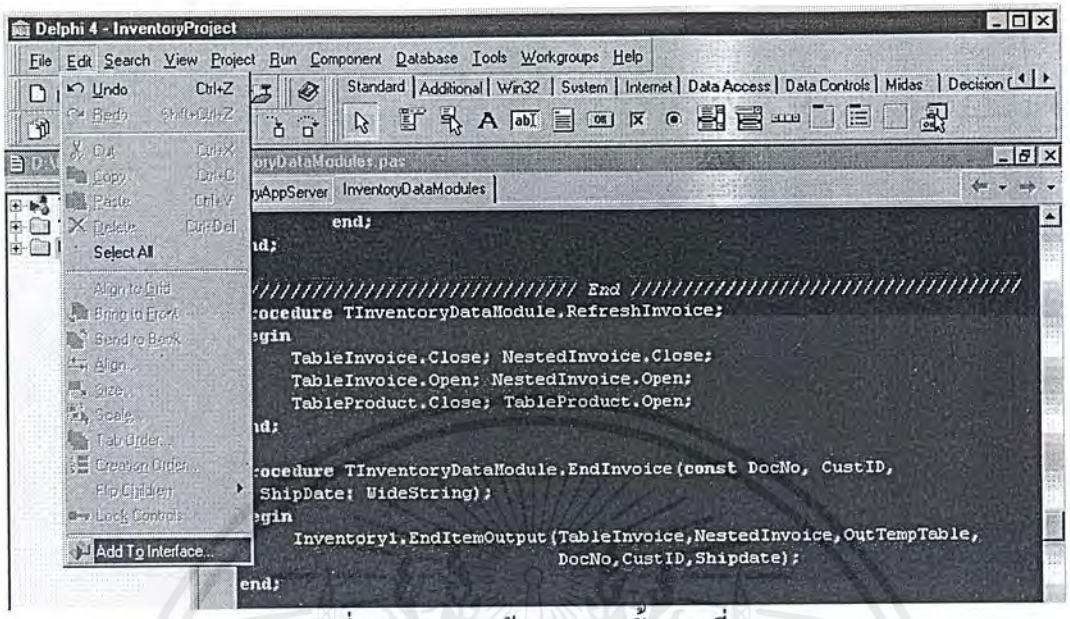
ส่วนที่ 2 : แอพพลิเคชันเซิร์ฟเวอร์

ส่วนนี้จะเป็นส่วนกลางระหว่างไคลเอนต์กับดาต้าเบสเซิร์ฟเวอร์ โดยคลาสต่างๆจะถูกเก็บรวบอยู่ในส่วนนี้ และคลาสอินเวนทอรี (Inventory) จะเป็นส่วนหลักในการสร้างออปเจ็กต์จากคลาสต่างๆ และกำหนดเมทธอด (Method) สำหรับให้ไคลเอนต์เรียกใช้งาน ซึ่งเมทธอดที่จะถูกเรียกใช้งานนั้นจะต้องกำหนดขึ้นจากรีโมตดาต้าโมดูล (Remote Data Module) ซึ่งเป็นส่วนเชื่อมต่อระหว่างแอพพลิเคชันเซิร์ฟเวอร์กับไคลเอนต์โดยแท้จริง ซึ่งเมทธอดที่จะถูกเรียกนั้นสามารถกำหนดได้ดังต่อไปนี้



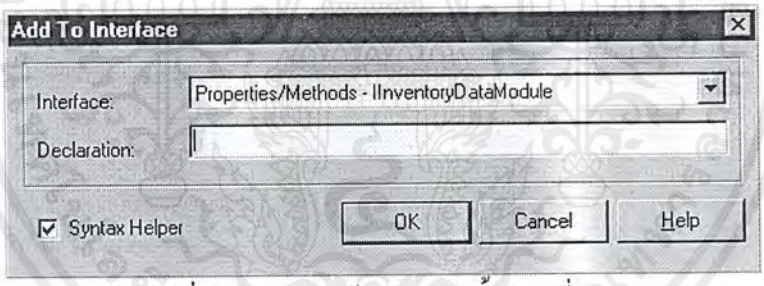
รูปที่ 7 แสดงการเรียกสร้างเมทธอดขั้นตอนที่ 1

1. เลือกเมนู 'Edit'
2. เลือกเมนู 'Add to Interface'



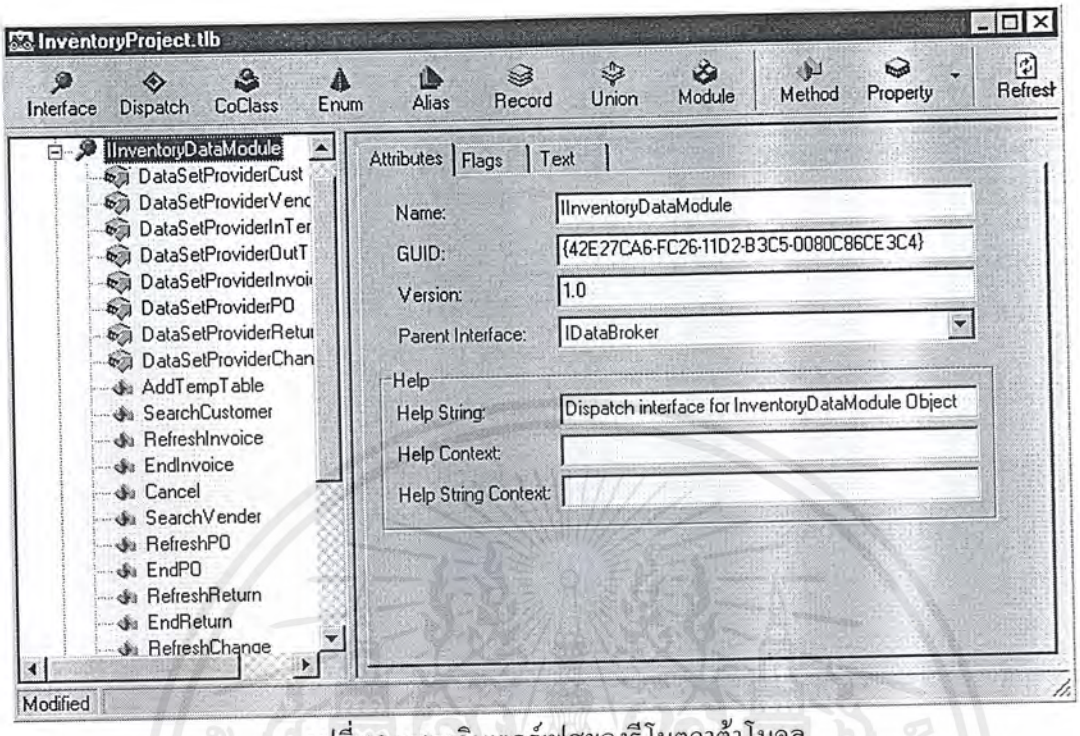
รูปที่ 8 แสดงการสร้างเมทธอดขั้นตอนที่ 2

3. ใส่ชื่อโพรซีเยอร์ (Procedure) หรือฟังก์ชัน (Function)



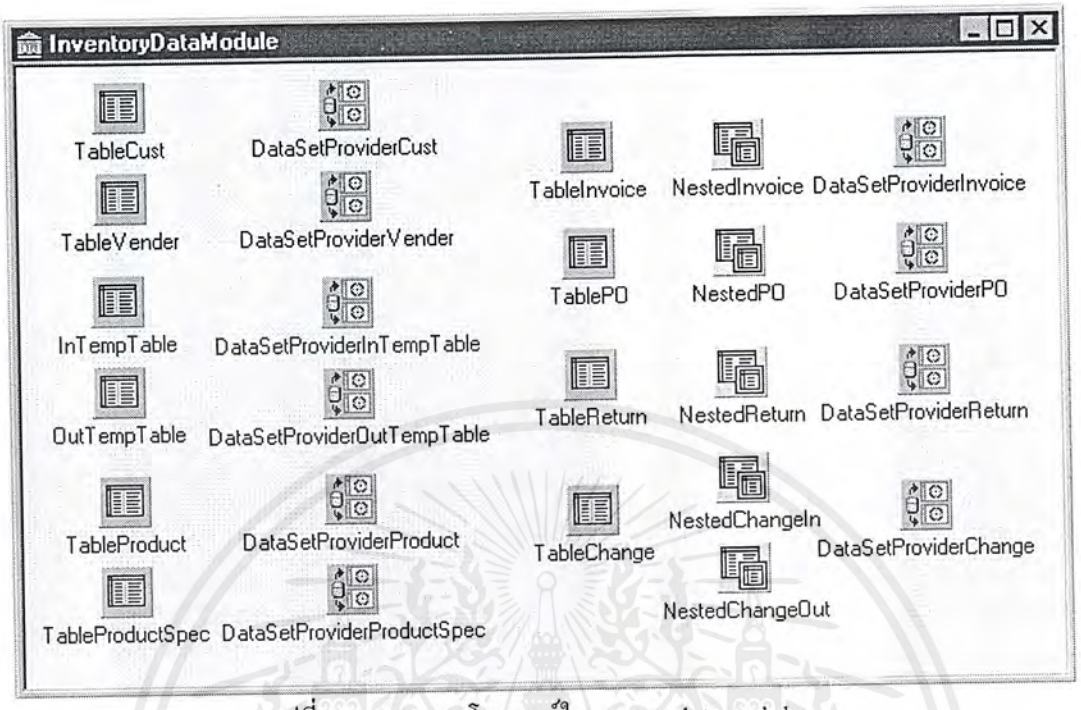
รูปที่ 9 แสดงการสร้างเมทธอดขั้นตอนที่ 3

เมทธอดต่าง ๆ เหล่านี้ก็จะเรียกใช้เมทธอดของคลาสอินเวนทอรีอีกครั้งหนึ่ง (ถ้าหากเป็นสถาปัตยกรรมแบบ 2-เทียร์ สามารถเรียกใช้เมทธอดของอินเวนทอรีได้โดยตรงโดยไม่จำเป็นต้องเรียกผ่านตัวรีโมตดาต้าโมดูล) ซึ่งเมทธอดต่าง ๆ ในรีโมตดาต้าโมดูลที่ได้สร้างไปแล้วนั้น ก็สามารถที่จะแก้ไขเพิ่มเติมได้โดยการเรียกไฟล์ที่มีนามสกุล .TLB ก็จะได้หน้าจอดังรูป



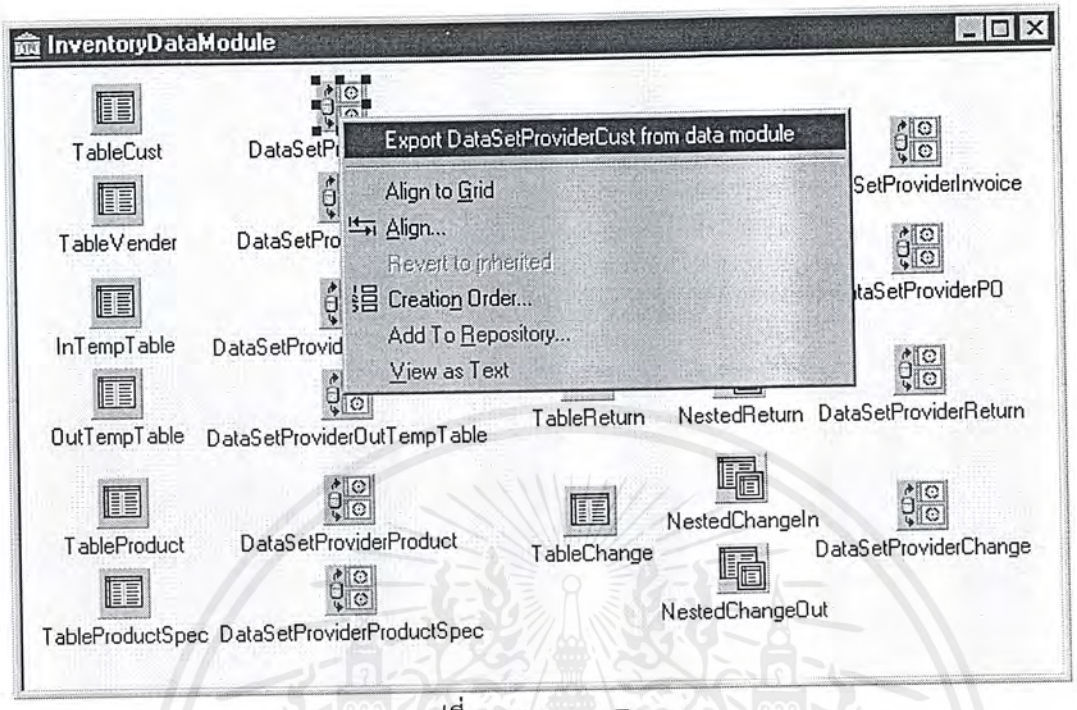
รูปที่ 10 แสดงอินเตอร์เฟสของรีโมตดาต้าโมดูล

ซึ่งส่วนนี้จะเป็นส่วนของรีโมตดาต้าโมดูลที่เก็บรวบรวมอินเตอร์เฟส (Interface) หรือเมทธอดต่าง ๆ เข้าไว้ด้วยกัน และสามารถแก้ไขได้แต่ที่สำคัญจะต้องทำการรีเฟรส (Refresh) ทุกครั้งเมื่อมีการแก้ไขหรือกำหนดค่าต่าง ๆ ที่ปุ่ม "REFRESH" หลังจากนั้นก็จะต้องนำคอมโพเนนต์ที่ต้องการมาระบุไว้ที่ฟอร์มของรีโมตดาต้าโมดูลดังรูปที่ 11



รูปที่ 11 แสดงคอมโพเนนต์ใน remote datamodule

โดยการส่งผ่านข้อมูลจากดาต้าเบสเซิร์ฟเวอร์ไปยังไคลเอนต์ จะเชื่อมต่อผ่านคอมโพเนนต์ TdataSetProvider ซึ่งจะต้องคลิกขวาแล้วทำการเอ็กซ์พอร์ต (export) ทุกครั้ง ไคลเอนต์จึงจะสามารถเชื่อมต่อได้ดังรูปที่ 12



รูปที่ 12 แสดงการ Export

ต่อไปจะเป็นส่วนของคลาสต่างๆ ที่นักพัฒนาโปรแกรมสามารถนำไปพัฒนาต่อได้ โดยมีคลาสที่ใช้ในระบบงานคลังสินค้าทั้งหมดดังต่อไปนี้

1. คลาสอินเวนทอรี (Inventory Class)

เมทธอด :

- procedure AddTempTable ทำหน้าที่เพิ่มสินค้าเก็บไว้ในตารางชั่วคราว
- procedure ClearTemptable ทำหน้าที่เคลียร์รายการสินค้าทั้งหมดในตารางชั่วคราว
- procedure EndItemInput ทำหน้าที่รับสินค้าเข้า
- procedure EndItemOutput ทำหน้าที่ส่งสินค้าออก
- procedure EndReturnItem ทำหน้าที่รับคืนสินค้า
- procedure EndChangeItem ทำหน้าที่เปลี่ยนสินค้า
- function FindCustomer ทำหน้าที่ค้นหาลูกค้า ถ้าเจอให้ส่งค่ากลับเป็น True
- function FindVender ทำหน้าที่ค้นหาผู้ขาย ถ้าเจอให้ส่งค่ากลับเป็น True
- procedure GetCustomer ทำหน้าที่รับข้อมูลของลูกค้า
- procedure GetVender ทำหน้าที่รับข้อมูลของผู้ขาย
- procedure AdjustStock ทำหน้าที่ปรับยอดสินค้าคงคลัง
- procedure RollBack ทำหน้าที่ยกเลิกทรานแซกชันเดิมที่ได้ทำไปแล้ว

2. คลาสทรานแซคชัน (Transaction Class)

แอตทริบิวต์ :

- Doc_No : String
- Shipdate : String

เมทธอด :

- procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล
- procedure SetTotalOutProduct ทำหน้าที่นับจำนวนสินค้าทั้งหมดในทรานแซกชัน
- procedure GetProduct ทำหน้าที่รับข้อมูลสินค้า
- function FindDoc_No ทำหน้าที่ค้นหาเอกสาร
- function GetShipdate ทำหน้าที่รับวันที่บันทึกข้อมูล

3. คลาสอินทราน (InTran Class)

แอตทริบิวต์ :

- TotalInProduct : Integer
- Inproduct :Array of String

เมทธอด :

- Procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล
- Procedure SetTotalOutProduct ทำหน้าที่ตั้งค่าสินค้าทั้งหมด
- Procedure GetProduct ทำหน้าที่รับสินค้า
- Function FindDoc_no ทำหน้าที่ค้นหาเอกสาร
- Function GetShipDate ทำหน้าที่รับวันที่บันทึกข้อมูล

4. คลาสเอาท์ทราน (OutTran Class)

แอตทริบิวต์ :

- TotalOutProduct : Integer
- OutProduct : Array of TProduct

เมทธอด :

- procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล
- procedure SetTotalOutputProduct ทำหน้าที่ตั้งค่าสินค้าทั้งหมด
- procedure GetProduct ทำหน้าที่รับสินค้า

5. คลาสอินเอาท์ทราน (InOutTran Class)

แอตทริบิวต์ :

- InProduct : Array of String
- OutProduct : Array of String

เมทธอด :

- Procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล
- Procedure SetTotalInProduct ทำหน้าที่ตั้งค่าสินค้าทั้งหมด
- Procedure GetProduct ทำหน้าที่รับสินค้า

6. คลาสเพอเชสออเดอร์(PurchaseOrder Class)

แอตทริบิวต์ :

- Vender_No : String

เมทธอด :

- procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล

7. คลาสรีเทิร์น (Return Class)

แอตทริบิวต์ :

- Cust_No : String

เมทธอด :

- Procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล

8. คลาสอินวอย (Invoice Class)

แอตทริบิวต์ :

- Cust_No : String

เมทธอด :

- procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล

9. คลาสเช้นจ์ (Change Class)

แอตทริบิวต์ :

- Cust_No : String

เมทธอด :

- procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงในฐานข้อมูล

10. คลาสเวนเดอร์ (Vender Class)

แอตทริบิวต์ :

- ID : String
- Name : String
- Address : Taddress

เมทธอด :

- Funciton Find ทำหน้าที่ค้นหาผู้ขาย
- Funciton GetName ทำหน้าที่รับชื่อผู้ขาย
- Funciton GetAddress ทำหน้าที่รับที่อยู่
- Funciton GetPhone ทำหน้าที่รับเบอร์โทรศัพท์

11. คลาสคัสโตเมอร์ (Customer Class)

แอตทริบิวต์ :

- ID : String
- FristName : String
- LastName : String
- Address : Taddress
- Company : String

เมทธอด :

- Function Find ทำหน้าที่ค้นหาลูกค้า
- Function GetFirstName ทำหน้าที่รับค่าชื่อ
- Function GetLastName ทำหน้าที่รับค่านามสกุล
- Function GetAddress ทำหน้าที่รับค่าที่อยู่
- Function GetCompany

12. คลาสแอดเดรส (Address Class)

แอตทริบิวต์ :

- Address_No : Integer
- Road : String
- City : String
- Province : String

เมทธอด :

- Function GetAddrees_No ทำหน้าที่รับค่าบ้านเลขที่
- Function GetRoad ทำหน้าที่รับค่าถนน
- Function GetCity ทำหน้าที่รับค่าเมือง
- Function GetProvince ทำหน้าที่รับค่าจังหวัด

13. คลาสโปรดักส์ (Product Class)

แอตทริบิวต์ :

- UPC : String
- QTY : String

เมทธอด :

- Function GetUPC ทำหน้าที่รับค่า UPC
- Function GetQTY ทำหน้าที่รับค่าปริมาณ

14. คลาสโปรดักส์สเปกซิฟิเคชัน (ProductSpecification Class)

แอตทริบิวต์ :

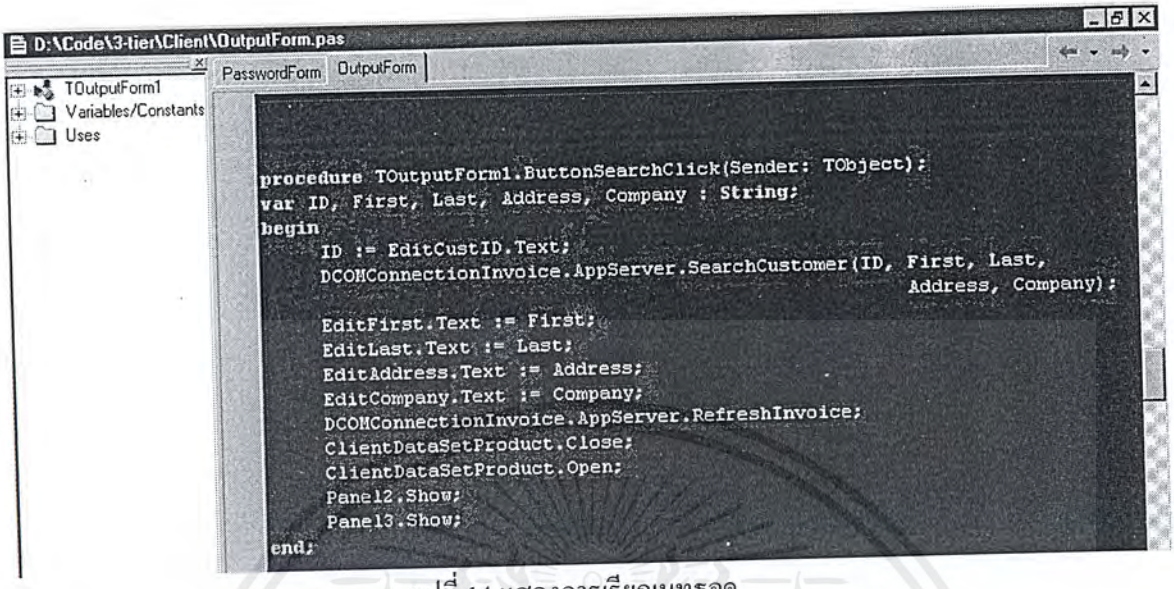
- UPC : Integer
- Description : String
- Price : Integer

เมทธอด :

- Procedure Find ทำหน้าที่ค้นหาสินค้า
- Procedure RecordToDB ทำหน้าที่เก็บข้อมูลลงฐานข้อมูล
- Procedure GetUPC ทำหน้าที่รับค่า UPC
- Procedure GetDescription ทำหน้าที่รับค่า Description
- Procedure GetPrice ทำหน้าที่รับราคา

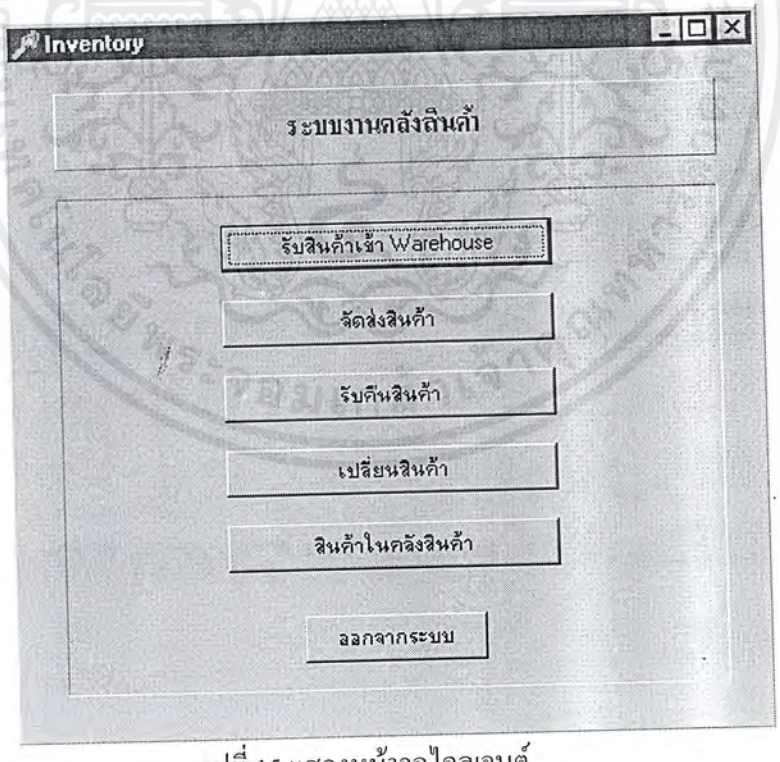
ส่วนที่ 3 : ไคลเอนต์

ส่วนนี้จะเป็นส่วนที่ติดต่อกับผู้ใช้ โดยจะทำการเรียกเมทธอดที่เซิร์ฟเวอร์แล้วนำมาใช้งาน โดยการเรียกเมทธอดที่เซิร์ฟเวอร์สามารถกระทำได้ดังนี้



รูปที่ 14 แสดงการเรียกเมทธอด

ส่วนการเชื่อมต่อฐานข้อมูลนั้นจะใช้คอมโพเนนต์ดังต่อไปนี้



รูปที่ 15 แสดงหน้าจอไคลเอนต์

การเรียกดูข้อมูลจากแอพพลิเคชันเซิร์ฟเวอร์จะทำได้โดยการตั้งค่าตามขั้นตอนดังต่อไปนี้

1. **DCOMConnection**

   Properties :

   ServerName : InventoryProject.InventoryDataModule

2. **ClientDataSet**

   Properties :

   RemoteServer : DCOMConnection1

   ProviderName : DataSetProvider1

3. **DataSource**

   Properties :

   DataSet : ClientDataSet1

4. **DBGrid**

   Properties :

   DataSource : DataSource1

ภาคผนวก ค

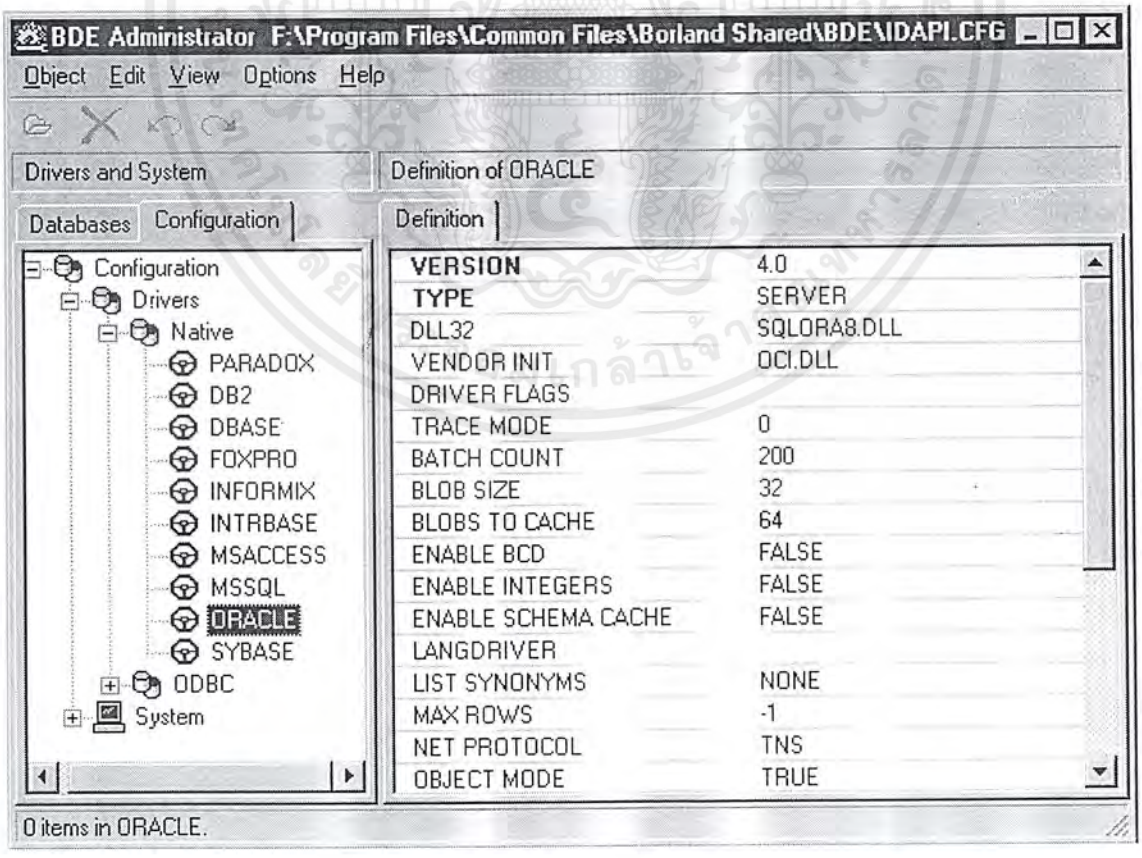
การตั้งค่า Borland Database Engine

Borland Database Engine (BDE)

เนื่องจาก Delphi ใช้ BDE เป็นเครื่องมือสำหรับกำหนดคุณสมบัติและลักษณะในการเชื่อมต่อกับฐานข้อมูล ดังนั้น Delphi จึงมีทูล BDE Administrator เพื่อใช้ในการตั้งค่าและกำหนดคุณสมบัติต่างๆ สำหรับ BDE

การตั้งค่าคอนฟิกูเรชัน BDE สำหรับฐานข้อมูล ORACLE

1. เลือกแถบ Configuration
2. เลือก Drivers / Native / ORACLE
3. กำหนดคุณสมบัติ DDL32 ให้เป็น SQLORA8.DLL
4. กำหนดคุณสมบัติ VENDOR INIT ให้เป็น OCI.DLL
5. คุณสมบัติอื่นๆ ให้เป็นไปตามค่าดีฟอลต์ ดังรูปที่ 1

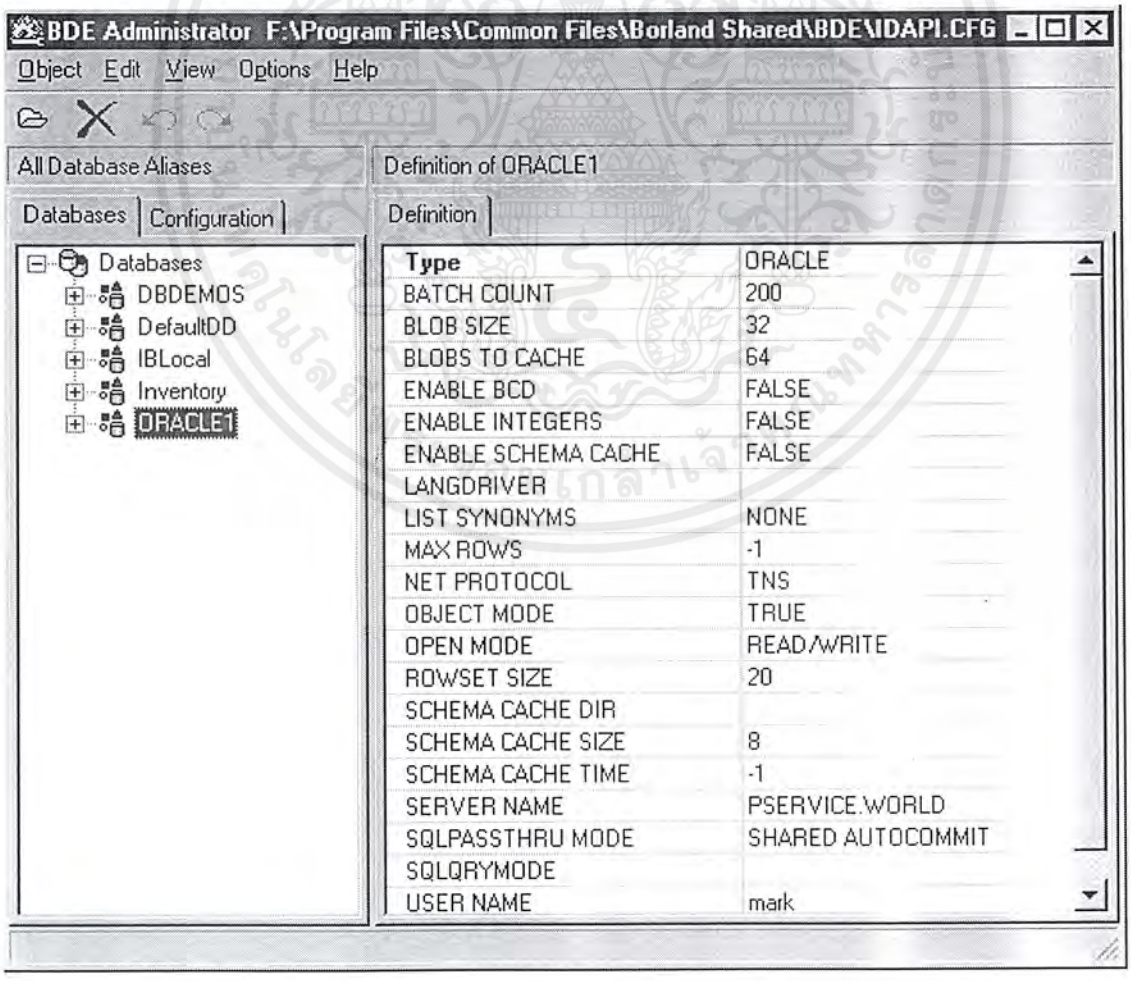


รูปที่ 1 การตั้งค่า Configuration ของ BDE

การสร้าง Alias Name เพื่อใช้ในการเชื่อมต่อฐานข้อมูล

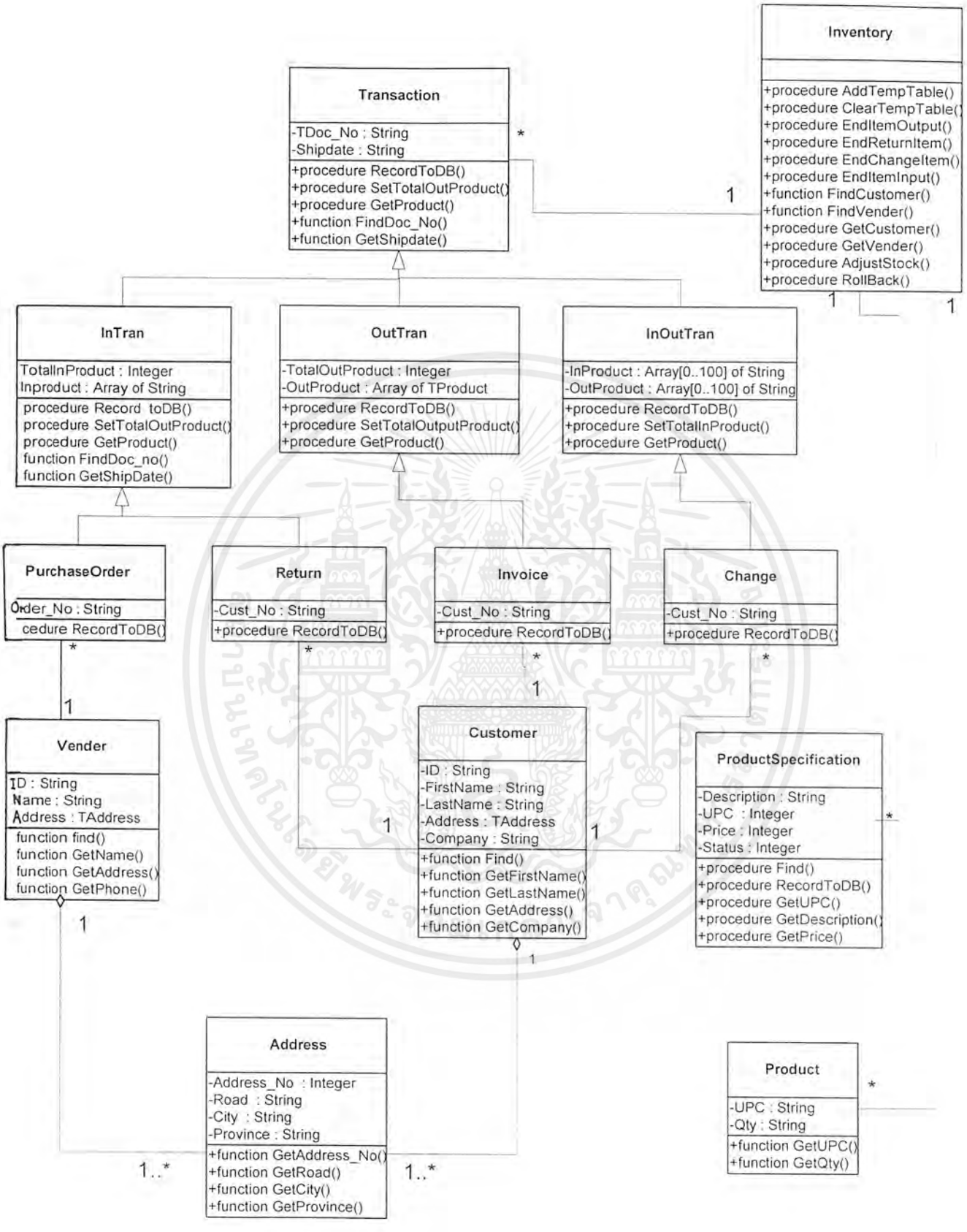
การเรียกใช้ฐานข้อมูลผ่าน BDE นั้น จำเป็นต้องมีการสร้างชื่อเสมือน (Alias Name) ที่จะใช้เป็นชื่อในการเชื่อมต่อกับฐานข้อมูล โดยมีขั้นตอนการสร้างดังนี้

1. หลังจากเรียก BDE Administrator ขึ้นมาแล้ว ให้เลือกเมนู Object / New
2. เลือกชนิดของไดร์เวอร์เป็น ORACLE
3. กำหนดชื่อเสมือน ในที่นี้สมมติเป็น ORACLE1
4. คลิกปุ่ม Apply เพื่อยืนยันการเปลี่ยนแปลงที่เกิดขึ้น
5. ที่คุณสมบัติ SERVER NAME ให้กำหนดชื่อเซอร์วิสในการเชื่อมต่อกับฐานข้อมูล ORACLE (service name) ในที่นี้ได้แก่ PSERVICE.WORLD
6. กำหนดคุณสมบัติ USER NAME เป็นชื่อของผู้ใช้งานฐานข้อมูล
7. คลิกปุ่ม Apply เพื่อยืนยันการเปลี่ยนแปลงที่เกิดขึ้น
8. เสร็จสิ้นขั้นตอนการสร้างชื่อเสมือน หลังจากนี้เมื่อเราต้องการเชื่อมต่อกับฐานข้อมูล ORACLE เราจะกระทำผ่านชื่อเสมือน ORACLE1 ดังรูปที่ 2



รูปที่ 2 การสร้างชื่อเสมือนสำหรับการเชื่อมต่อฐานข้อมูลผ่าน BDE

ภาคผนวก ง
คลาสไดอะแกรมของระบบงานคลังสินค้า



รูปที่ 1 Class Diagram ของระบบคลังสินค้า

ภาคผนวก จ

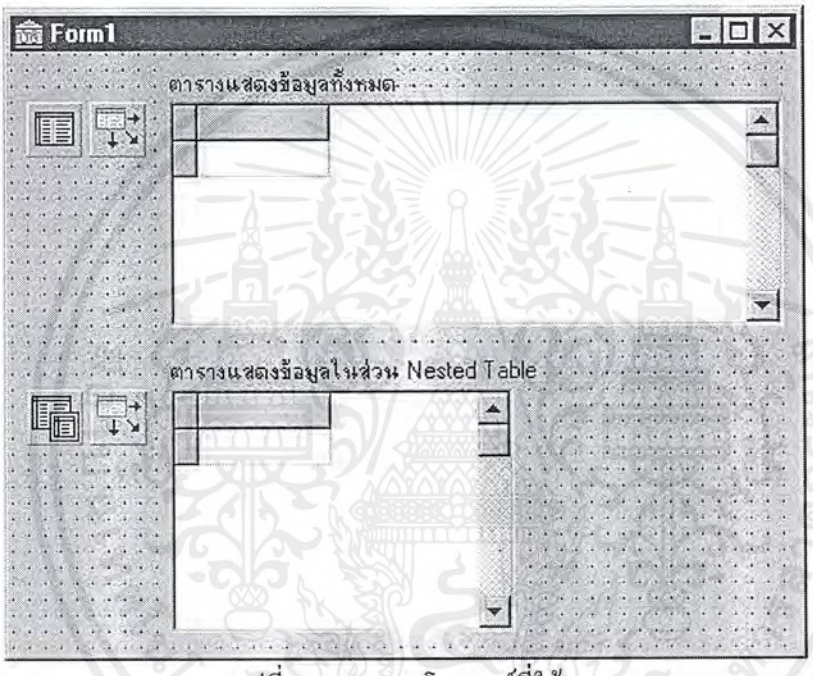
การใช้งาน Nested Table ในเดลไฟล์ 4

1. การใช้งานผ่านสถาปัตยกรรม 2-เทียร์

1.1 การตั้งค่าคอมโพเนนต์

การใช้งาน Nested Table จะต้องตั้งค่าคอมโพเนนต์ต่างๆ ดังต่อไปนี้

1.1.1 วางคอมโพเนนต์ดังรูปที่ 1



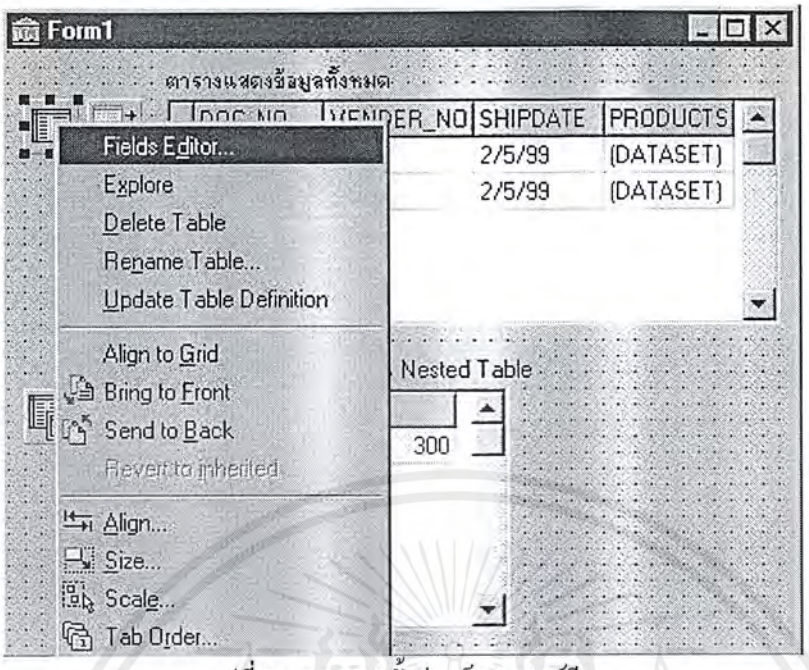
รูปที่ 1 แสดงคอมโพเนนต์ที่ใช้

1.1.2 ตั้งค่าพร็อพเพอร์ตีต่างๆ ดังต่อไปนี้

Table1 : Databasename – เลือกฐานข้อมูลที่เป็น Nested Table

Tablename – เลือกตารางที่เป็น Nested Table

คลิกขวาที่คอมโพเนนต์ Table ดังรูปที่ 2



รูปที่ 2 แสดงการตั้งค่าพร็อพเพอร์ตี

เลือกทุกฟิลด์ดังรูปที่ 3

รูปที่ 3 แสดงฟิลด์ทั้งหมด

Nested Table1 : DataSetField – ดับเบิ้ลคลิกดังรูปที่ 4

รูปที่ 4 แสดงการตั้งค่าพร็อพเพอร์ตี

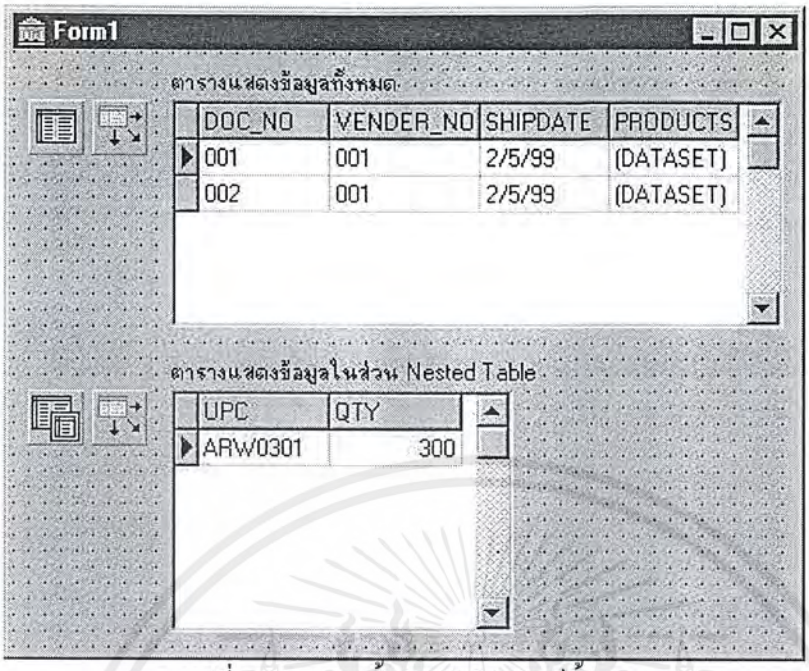
DataSource1 : DataSet – Table1

DataSource2 : DataSet – NestedTable1

DBGrid1 : DataSource – DataSource1

DBGrid2 : DataSource – DataSource2

1.1.3 ตั้งค่า Active ของเทเบิลและNested Tableเป็น True ดังรูปที่ 5



รูปที่ 5 แสดงการตั้งค่าคอมโพเนนต์ทั้งหมด

1.2 การเรียกใช้งาน

1.2.1 การเพิ่มข้อมูล

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    NestedTable1.Last;
    NestedTable1.Insert;
    NestedTable1.FieldByname('UPC').AsString := Edit1.Text;
    NestedTable1.FieldByname('QTY').AsString := Edit2.Text;
    NestedTable1.Post
end;
```

รูปที่ 6 แสดงคำสั่งที่ใช้ในการเพิ่มข้อมูล

1.2.2 การแก้ไขข้อมูล

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    NestedTable1.Edit;
    NestedTable1.FieldByname('UPC').AsString := Edit1.Text;
    NestedTable1.FieldByname('QTY').AsString := Edit2.Text;
    NestedTable1.UpdateRecord;
end;
```

รูปที่ 7 แสดงคำสั่งที่ใช้ในการแก้ไขข้อมูล

1.2.3 การลบข้อมูล

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    NestedTable1.Edit;
    NestedTable1.Delete;
end;
```

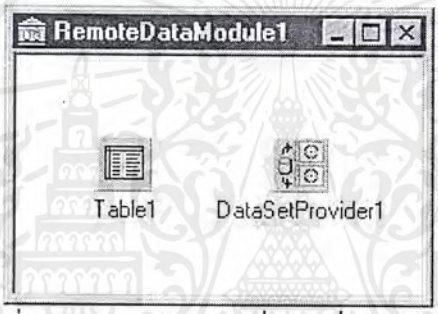
รูปที่ 8 แสดงคำสั่งที่ใช้ในการลบข้อมูล

2. การใช้งานผ่านสถาปัตยกรรม 3-เทียร์

2.1 การตั้งค่าคอมโพเนนต์

การใช้งาน Nested Table จะต้องตั้งค่าคอมโพเนนต์ต่างๆดังต่อไปนี้

2.1.1 วางคอมโพเนนต์บนฝั่งเซิร์ฟเวอร์ ดังรูปที่ 9



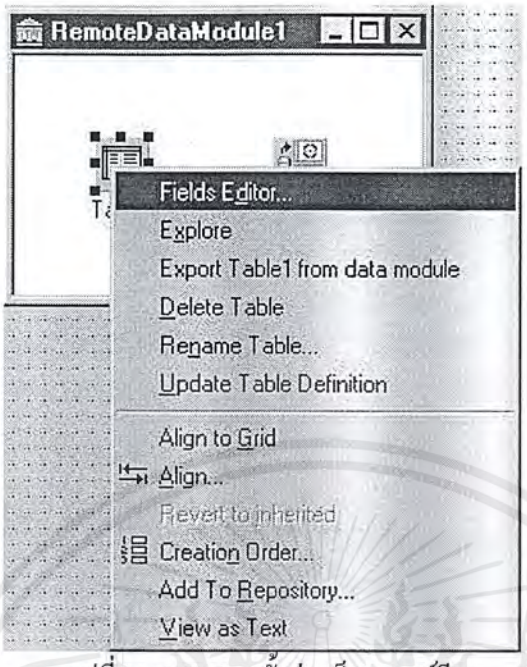
รูปที่ 9 แสดงคอมโพเนนต์ที่ใช้บนฝั่งเซิร์ฟเวอร์

2.1.2 ตั้งค่าพร็อพเพอร์ตีต่างๆ ดังต่อไปนี้

Table1 : Databasename – เลือกฐานข้อมูลที่เป็น Nested Table

Tablename – เลือกตารางที่เป็น Nested Table

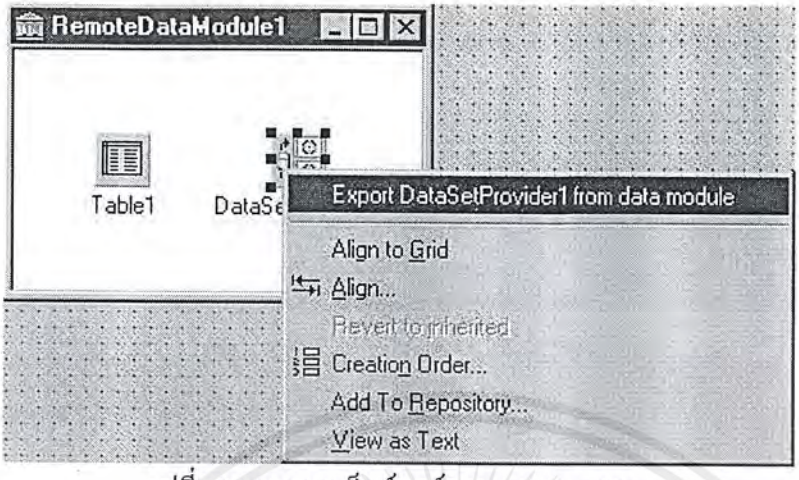
คลิกขวาที่คอมโพเนนต์ Table ดังรูปที่ 10



รูปที่ 10 แสดงการตั้งค่าพร็อพเพอร์ตี

เลือกทุกฟิลด์ดังรูปที่ 11

รูปที่ 11 แสดงฟิลด์ทั้งหมด

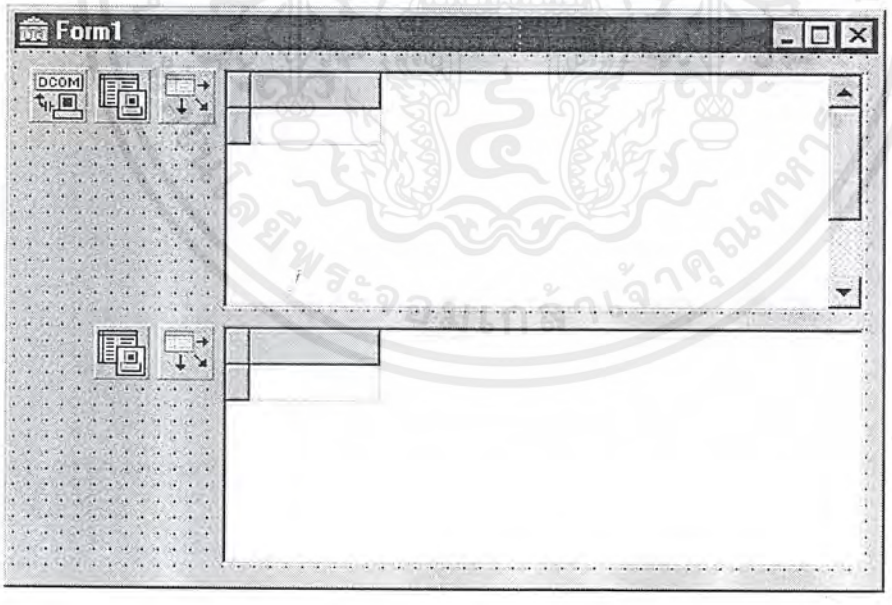


รูปที่ 12 แสดงการเอ็กซ์พอร์ต DataSetProvider

คลิกขวาเพื่อทำการเอ็กซ์พอร์ต DataSetProvider ดังรูปที่ 12

DataSetProvider1 : DataSet – Table1

2.1.3 วางคอมโพเนนต์บนฝั่งไคลเอนต์ ดังรูปที่ 13



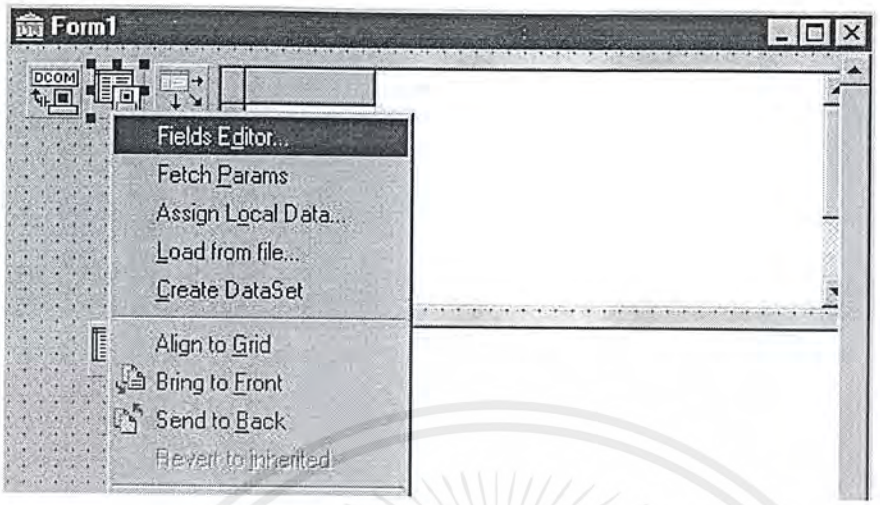
รูปที่ 13 แสดงคอมโพเนนต์ที่ใช้บนฝั่งไคลเอนต์

2.1.4 ตั้งค่าพร็อพเพอร์ตีต่างๆ ดังต่อไปนี้

DCOMConnection : ServerName – เลือกเซิร์ฟเวอร์ที่ต้องการ

ClientDataSet1 : RemoteServer – DCOMConnection1

ProviderName1 – DataSetProvider1



รูปที่ 14 แสดงการตั้งค่าพร็อพเพอร์ตี

คลิกขวาที่ ClientDataSet1 เลือก Field Editor.. ดังรูปที่ 14 แล้วเลือกทุกฟิลด์ ดังรูปที่ 15

รูปที่ 15 แสดงฟิลด์ทั้งหมด

ClientDataSet2 : DataSetField - ดับเบิ้ลคลิก

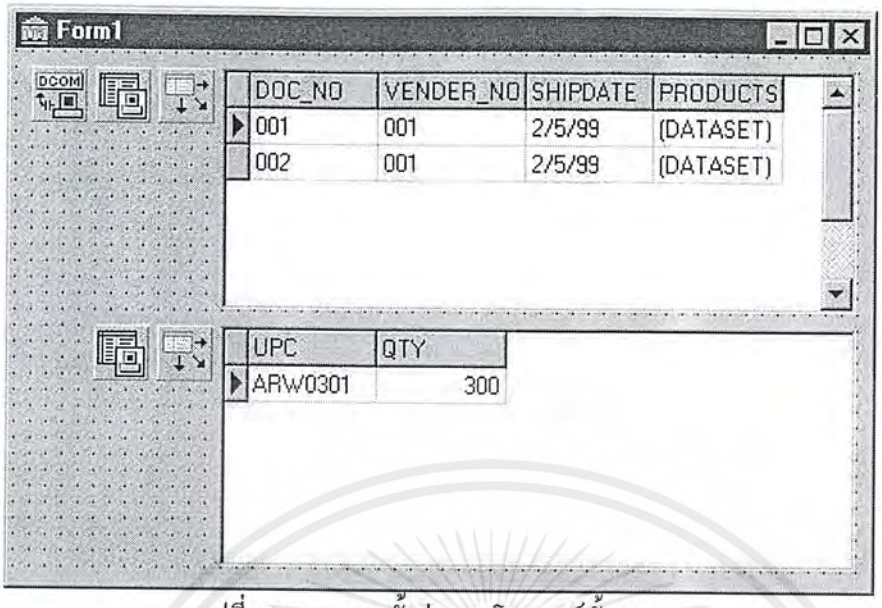
DataSource1 : DataSet – ClientDataSet1

DataSource2 : DataSet – ClientDataSet2

DBGrid1 : DataSource – DataSource1

DBGrid2 : DataSource – DataSource2

2.1.5 ตั้งค่า Active ของเทเบิลและ Nested Table เป็น True ดังรูปที่ 16



รูปที่ 16 แสดงการตั้งค่าคอมโพเนนต์ทั้งหมด

2.2 การเรียกใช้งาน

คล้ายกับการใช้งานแบบ 2 เทียร์ เพียงเปลี่ยนจากNested Tableเป็น ClintDataSet ดังนี้

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     ClientDataSet2.Last;
     ClientDataSet2.Insert;
     ClientDataSet2.FieldByName('UPC').AsString := Edit1.Text;
     ClientDataSet2.FieldByName('QTY').AsString := Edit2.Text;
     ClientDataSet2.Post;
     ClientDataSet1.ApplyUpdates(100);
end;
```

รูปที่ 17 แสดงคำสั่งที่ใช้ในการเพิ่มข้อมูล

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     ClientDataSet2.Edit;
     ClientDataSet2.FieldByName('UPC').AsString := Edit1.Text;
     ClientDataSet2.FieldByName('QTY').AsString := Edit2.Text;
     ClientDataSet2.UpdateRecord;
     ClientDataSet1.ApplyUpdates(100);
end;
```

รูปที่ 18 แสดงคำสั่งที่ใช้ในการแก้ไขข้อมูล

2.2.3 การลบข้อมูล

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ClientDataSet2.Edit;
    ClientDataSet2.Delete;
    ClientDataSet1.ApplyUpdates(100);
end;
```

รูปที่ 19 แสดงคำสั่งที่ใช้ในการลบข้อมูล

บรรณานุกรม

หนังสืออ้างอิง

[1] Larman Craig , *"Applying UML and Patterns : an introduction to object-oriented analysis and design"* ,Prentice Hall PTR , 1998.

[2] Tkach Daniel , *"Object technology in Application Development "* , 2nd ed , Addison-Wesley Publishing Company , 1996.

[3] Sphar Chuck , *"Object-oriented programming power for THINK Pascal programmers"* , Microsoft Press , 1991.

[4] Weber Edward C. , Ford J.Neal , Weber Christopher R. , *"Developing with Delphi : Object-Oriented Techniques"* , Prentice Hall PTR , 1996.

[5] Bobrowski Steve , *"Oracle8 Architecture"* , Osborne McGraw-Hill , 1998.

[6] Kanid Tkach , Danid Tkach , *"Object in Technology in Application Development"* , Addison-Wesley Publishing Company.

[7] George Wilkie , *"Object-Oriented Software Engineering (the professional developer's guide)"* , Addison-Wesley Publishing Company.

[8] Edward C.Weber , J. Neal Ford , Christopher R. Weber , *"Developing with DELPHI Object – Oriented Techiques"* , Prentice Hall PTR.

[9] กนก กุสุมาลย์นุกูล , ไกรวุฒิ มั่นเสถียรสิน , *"คู่มือการเขียนโปรแกรม Delphi 4"* , สำนักพิมพ์ซัคเซสมีเดีย .

เว็ปไซต์อ้างอิง

[10] http://www.inprise.com/midas

[11] http://www.borland.com

[12] http://www.borland.com/midas/papers

[13] http://www.borland.com/midas/papers/multitier/multi.html

[14] http://www.rational.com/uml

[15] http://delphi4.com

[16] http://www.thaidelphi.com

[17] http://www.doit.com/delphi

[18] http://isecwww.nida.ac.th/F4/f4-topic.html

[19] http://isecwww.nida.ac.th/F4/f4-16.htm#Top

[20] http://www.oracle.com