การศึกษาเอกลักษณ์ของออบเจ็กต์ และการแก้ปัญหาคีย์แสดงเอกลักษณ์ของระบบฐานข้อมูลเชิงสัมพันธ์ด้วยระบบฐานข้อมูลเชิงวัตถุ

A Study of Object Identifiers and Solutions of the Relational Identifier Problems Using Object-Oriented Database Approach

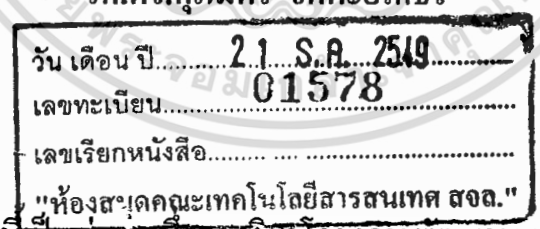
โดย

นางสาวสุมาลา หิรัญมงคลกุล

รหัส 40067028

อาจารย์ที่ปรึกษา

รศ.ดร.ศุภมิตร จิตตะยโศธร



รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2541

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การศึกษาเอกลักษณ์ของออบเจ็กต์ และการแก้ปัญหาคีย์แสดงเอกลักษณ์ของระบบฐานข้อมูลเชิงสัมพันธ์ด้วยระบบฐานข้อมูลเชิงวัตถุ
นักศึกษา	นางสาวสุมาลา หิรัญมงคลกุล
อาจารย์ที่ปรึกษา	รศ.ดร.ศุภมิตร จิตตะยโศธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
พ.ศ.	2541

บทคัดย่อ

หลายปีที่ผ่านมา ระบบฐานข้อมูลเชิงสัมพันธ์เป็นระบบฐานข้อมูลที่ได้รับความนิยมอย่างสูง ทำให้มีผู้พัฒนาระบบงานด้านสารสนเทศบนระบบฐานข้อมูลนี้เป็นจำนวนมาก อันเนื่องมาจากระบบฐานข้อมูลเชิงสัมพันธ์มีโครงสร้างข้อมูลที่เข้าใจได้ง่าย มีภาษาในการจัดการข้อมูลที่มีประสิทธิภาพสูง และมีมาตรฐานรองรับ สำหรับแนวทางการศึกษาเอกลักษณ์ของออบเจ็กต์ และการแก้ปัญหาคีย์แสดงเอกลักษณ์ของระบบฐานข้อมูลเชิงสัมพันธ์ด้วยระบบฐานข้อมูลเชิงวัตถุนี้ จะเน้นการศึกษาทางด้านเทคโนโลยีของออบเจ็กต์, ระบบการจัดการฐานข้อมูลเชิงวัตถุ, เอกลักษณ์ของออบเจ็กต์ และแนวทางการแก้ปัญหาคีย์แสดงเอกลักษณ์ของระบบฐานข้อมูลเชิงสัมพันธ์ด้วยระบบฐานข้อมูลเชิงวัตถุ พร้อมทั้งสร้างต้นแบบงานด้านสารสนเทศโดยใช้ระบบฐานข้อมูลเชิงวัตถุ

Title	A Study of Object Identifiers and Solutions of the Relational Identifier Problems Using Object-Oriented Database Approach
Student	Miss Sumala Hirunmongkholkul
Advisor	Assoc.Prof. Dr.Suphamit Chittayasothorn
Level of Study	Master of Science in Information Technology
Major	Information Science
Year	1998

ABSTRACT

Many years ago, relational database system is the very popular database system. There are many developers use this database system to develop the information system because relational database have a ease-of-understand structure, a high powerful data manipulate language and a certified standard. This project, Studying in Object Identity and Solving the Identifier Key Problems of Relational Database System, will aim to studying of object technology, Object-Oriented Database Management System (OO-DBMS), object identity and solving the identifier key problems of relational database system with object-oriented database system. In addition, also creating a prototype related information field with object-oriented database system.

กิตติกรรมประกาศ

ในการจัดทำโครงงานการศึกษาเอกลักษณ์ของออบเจ็กต์ และการแก้ปัญหาคีย์แสดงเอกลักษณ์ของระบบฐานเชิงสัมพันธ์ด้วยระบบฐานข้อมูลเชิงวัตถุที่จัดทำขึ้นนี้ได้รับความสนับสนุนจากหลายฝ่ายเป็นอย่างดีที่ให้คำแนะนำคำปรึกษา และสละเวลาอันมีค่าซึ่งทำให้การศึกษาโครงการนี้บรรลุผลตามเป้าหมายที่วางไว้ ผู้จัดทำจึงใคร่ขอขอบพระคุณบุคคลดังนี้

1. บิดา, มารดา และพี่ ที่คอยให้ความช่วยเหลือ และให้กำลังใจมาตลอด
2. รศ.ดร.ศุภมิตร จิตตะยโสธร อาจารย์ที่ปรึกษาโครงการนี้ที่คอยให้คำความรู้, คำปรึกษาตลอดจนแนะนำวิธีในการศึกษาโครงการนี้เป็นอย่างดี
3. ดร.เอื้อน ปิ่นเงิน ที่ให้คำแนะนำในการค้นคว้าหาข้อมูล
4. คุณขวัญตา เกษประดิษฐ์ ที่ช่วยติดต่อประสานงานระหว่าง รศ.ดร.ศุภมิตร กับข้าพเจ้า
5. คุณจักรกฤษณ์ กลิ่นสมิทธ์ ที่เอื้อเฟื้อเอกสารต่างๆ และให้คำปรึกษาแนะนำ ช่วยแก้ไขปัญหาต่างๆ
6. คุณศิริลักษณ์ เขมะประสิทธิ์ รุ่นพี่ IS2 ที่ช่วยติดต่อประสานงานกับ รศ.ดร.ศุภมิตร และให้คำปรึกษากำลังใจต่างๆในการทำโครงการนี้
7. คุณรวิช ภุมรินทร์ ที่คอยให้คำปรึกษาต่างๆ อีกทั้งยังให้ความรู้เกี่ยวกับระบบฐานข้อมูลและแนวคิดเชิงวัตถุ รวมถึงข้อสงสัยต่างๆที่เกิดขึ้นในระหว่างการทำโครงการนี้ นอกจากนี้ยังให้กำลังใจ และเป็นเพื่อนคุยในเวลาที่ข้าพเจ้าทำงาน (คุยผ่านระบบเครือข่าย)
8. บริษัท Computer Associates ที่เอื้อเฟื้อส่งแผ่น CD Jasmine Developer Edition มาให้ข้าพเจ้า
9. คุณรัชดาภรณ์ นาถอัจลา, คุณสดใส ไกรเลิศ, คุณสิริอร วสันต์สว่างวงศ์ และคุณอังคณา ศรีพยัพ เพื่อนๆสมัยมัธยมปลายโรงเรียนสวนกุหลาบวิทยาลัย นนทบุรีที่คอยให้กำลังใจและเป็นเพื่อนเที่ยวในเวลาที่เครียดจากการทำงาน
10. คุณกัญญรัตน์ อักษรอินทร์, คุณนิตยา ช่อฟ้า, คุณวรนุช มีภูมิรู้, คุณสุวรรณา เมธีภัทรากูล และ คุณเสาวภา สันติวภานนท์เพื่อนรุ่น IS3 ที่คอยให้คำปรึกษาและกำลังใจตลอดจนเป็นเพื่อนเที่ยวในเวลาที่เครียดจากการทำงาน

สุมาลา หิรัญมงคลกุล

ผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

หน้า

ภาพที่

สารบัญภาพ (ต่อ)

หน้า

ภาพที่

สารบัญภาพ (ต่อ)

หน้า

ภาพที่

สารบัญภาพ (ต่อ)

หน้า

ภาพที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของโครงการพัฒนาระบบงาน

ในปี 1970 Dr. E. F. Codd ได้สร้างแบบจำลองเชิงสัมพันธ์ขึ้น และเมื่อปี 1980 ได้เริ่มมีระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ (RDBMS) ที่เป็นเชิงการค้าออกมา ได้แก่ Ingres, Oracle และ Informix ซึ่งในปัจจุบันก็ยังได้รับความนิยมอยู่ด้วยเหตุที่ว่าเป็นแบบจำลองที่มีโครงสร้างข้อมูลที่ง่ายคือ เป็น รีเลชั่น (Relation) และมักถูกนำเสนอออกมาในรูปแบบของตาราง ทำให้ผู้พัฒนาและผู้ใช้เข้าใจได้ง่ายอันเนื่องมาจากผู้ใช้สามารถกำหนดคีย์แสดงเอกลักษณ์ (Identifier Key) ได้เองโดยเลือกจากแอตทริบิวต์หรือกลุ่มของแอตทริบิวต์ในรีเลชั่นนั้นๆ ด้วยเหตุนี้จึงทำให้เกิดปัญหาและข้อจำกัดบางอย่างเกิดขึ้นกับระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ที่ใช้ Identifier Key ที่มาจากค่าของแอตทริบิวต์ได้ เช่น ไม่ควรมีการเปลี่ยนแปลงค่าคีย์ที่ใช้เป็น Identifier Key ซึ่งในงานด้านสารสนเทศที่มีอยู่ปัจจุบันมักจะละเมิดข้อจำกัดนี้

นอกจากปัญหาที่เกิดจากการเปลี่ยนแปลงแก้ไขค่าคีย์แล้วยังพบว่าในฐานข้อมูลเชิงสัมพันธ์มีการใช้เลือกใช้คีย์ในแต่ละตารางที่ไม่เหมือนกัน กล่าวคือบางตารางใช้สตริง, Meaningful Digit, หรือจำนวนเต็มเป็นคีย์ หรือบางตารางอาจมีคีย์คู่แข่งหลายคีย์ทำให้เกิดความยุ่งยากในการเลือกว่าตัวใดจึงจะเหมาะสมที่สุดที่จะใช้เป็นคีย์ในตารางนั้น และเนื่องจากระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ไม่สนับสนุนโครงสร้างข้อมูลที่มีความซับซ้อน กล่าวคือมีเพียงชนิดข้อมูลที่เป็น Atomic เท่านั้นทำให้การสร้างออบเจ็กต์ที่ความซับซ้อนสามารถทำได้ลำบากมากต้องเกิดจากการ Join กันหลายตารางทำให้สูญเสีย Semantic ไป ซึ่งจะกล่าวรายละเอียดทั้งหมดในบทที่ 5

จากปัญหาต่างๆที่ได้กล่าวมาล้วนเป็นปัญหาของการแสดงเอกลักษณ์โดยใช้ค่าหรือกลุ่มของแอตทริบิวต์ในการแทนออบเจ็กต์ ซึ่งปัญหาเหล่านี้พบได้กับทุกหน่วยงานที่ใช้ระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ ด้วยเหตุนี้จึงทำให้เกิดการศึกษาโครงการพัฒนาระบบงานนี้ขึ้น เพื่อศึกษาการแสดงเอกลักษณ์ของออบเจ็กต์ และความเป็นไปได้ในการนำคุณสมบัติที่มีอยู่ในระบบการจัดการฐานข้อมูลเชิงวัตถุ (OO-DBMS) ซึ่งได้แก่ Object Identifier (OID) ที่ใช้ในการแสดงเอกลักษณ์ของออบเจ็กต์หรือเอนติตี้ที่สนใจเหล่านั้น เพื่อมาแก้ปัญหาที่เกิดขึ้นในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์

1.2 วัตถุประสงค์ของโครงการพัฒนาระบบงาน

1.2.1 ศึกษาทฤษฎีและโครงสร้างแบบจำลองฐานข้อมูลเชิงสัมพันธ์และเชิงวัตถุ

1.2.2 ศึกษาหัวข้อ Object Identity

1.2.3 ศึกษาถึงปัญหาของการแสดง Identity โดยใน Identifier Keys ของระบบการจัดการฐานข้อมูลเชิงสัมพันธ์

1.2.4 เพื่อศึกษาแนวทางหรือวิธีการของออบเจ็กต์โอเรียนเต็ด และ Jasmine ที่ใช้ในการแก้ปัญหา Identifier Key

1.2.5 สร้างต้นแบบทดลองโดยใช้ Jasmine Studio เป็นเครื่องมือที่ใช้ในการพัฒนาแอปพลิเคชั่น และใช้ Jasmine เป็น Database Server ซึ่งเป็นระบบการจัดการฐานข้อมูลเชิงวัตถุ เพื่อแสดงการแก้ปัญหาการแสดง Identity ด้วย Identifier Key ของระบบการจัดการฐานข้อมูลเชิงสัมพันธ์

1.3 ขอบเขตของโครงการพัฒนาระบบงาน

1.3.1 ศึกษาแบบจำลองเชิงวัตถุ (Object Model)

1.3.2 ศึกษาทฤษฎีที่เกี่ยวข้องกับโครงการได้แก่ระบบการจัดการฐานข้อมูลเชิงสัมพันธ์และเชิงวัตถุ

1.3.3 ศึกษาเอกลักษณ์ของออบเจ็กต์ (Object Identity)

1.3.4 ศึกษาปัญหาที่พบจากการใช้ค่าของคีย์มาแสดงเอกลักษณ์ในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์

1.3.5 ศึกษาและนำเสนอแนวทางการแก้ปัญหาโดยใช้ออบเจ็กต์โอเรียนเต็ด และ Jasmine

1.3.6 นำเสนอโปรโตไทพ์ที่พัฒนาโดยใช้ทูลคือ Jasmine Studio และมี Jasmine Object Database เป็นระบบการจัดการฐานข้อมูลเชิงวัตถุ เพื่อแก้ไขปัญหาการแสดงเอกลักษณ์

1.4 เนื้อหาในโครงการพัฒนาระบบงาน

เนื้อหาสำหรับโครงการพัฒนาระบบงานในบทแรกนี้จะกล่าวถึงความเป็นมา ความสำคัญและวัตถุประสงค์ในการทำโครงการพัฒนาระบบงานนี้ บทที่ 2 กล่าวถึงทฤษฎีที่เกี่ยวข้องทั้งระบบฐานข้อมูลเชิงสัมพันธ์และเทคโนโลยีออบเจ็กต์ ในบทที่ 3 จะกล่าวถึงระบบการจัดการฐานข้อมูลเชิงวัตถุ โดยจะมีเนื้อหาเกี่ยวกับวิวัฒนาการ และลักษณะพื้นฐานที่สำคัญของระบบการจัดการฐาน

ข้อมูลเชิงวัตถุ, ข้อดีของรูปแบบข้อมูลเชิงวัตถุ และ การเปรียบเทียบระหว่างแบบจำลองเชิงวัตถุและเชิงสัมพันธ์ บทที่ 4 กล่าวเอกลักษณ์ของออบเจ็กต์ ซึ่งจะมีนิยาม และความสำคัญของ Object Identity, ปัญหาของการใช้ Identifier Key ในการแสดงเอกลักษณ์ และเทคนิคการ Implement Object Identity บทที่ 5 จะเป็นปัญหาของ Identifier Key บทที่ 6 เป็นการแนะนำ Jasmine บทที่ 7 เป็นการพัฒนาแอปพลิเคชั่นด้วย Jasmine บทที่ 8 เป็นแนวทางที่ใช้ในการแก้ปัญหาของ Identifier Key ตามแนวทางของ OO-DBMS และ Jasmine บทที่ 9 จะเป็นบทสรุปของโครงการ และภาคผนวกจะเป็นโปรโตไทพ์ Registration Information System

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

2.1 ระบบฐานข้อมูลเชิงสัมพันธ์

2.1.1 โครงสร้างข้อมูล

โครงสร้างข้อมูลของระบบฐานข้อมูลเชิงสัมพันธ์จะเป็นรีเลชั่นซึ่งเป็นระดับลอจิกคอล มักนำเสนอข้อมูลออกมาในรูปของตาราง โดยตารางที่แสดงรีเลชั่นต้องมีคุณสมบัติดังนี้

- ข้อมูลแต่ละแถวจะต้องแตกต่างกัน
- การเรียงลำดับแถวไม่ถือว่ามีความสำคัญ
- การเรียงลำดับคอลัมน์ไม่ถือว่ามีความสำคัญ
- แต่ละคอลัมน์ในตารางจะเป็นค่าที่แบ่งย่อยต่อไปอีกไม่ได้ (Atomic Value) กล่าวคือในแต่ละคอลัมน์ต้องไม่มีค่าข้อมูลที่อยู่ในลักษณะของอาร์เรย์หรือกลุ่ม

2.1.2 กฎควบคุมความถูกต้องของข้อมูล (Data Integrity)

กฎควบคุมความถูกต้องของข้อมูลในโมเดลเชิงสัมพันธ์เป็นทฤษฎีที่ช่วยยืนยันความถูกต้องของความสัมพันธ์ระหว่างข้อมูลว่า รีเลชั่นใดที่เป็นไปตามกฎนี้แล้วย่อมจะมีความสัมพันธ์ระหว่างข้อมูลอย่างถูกต้องอยู่ตลอดเวลา ไม่ว่ารีเลชั่นนั้นจะมีการเปลี่ยนแปลงแก้ไขข้อมูลไปในรูปแบบใดก็ตาม กฎควบคุมความถูกต้องของข้อมูลมีอยู่สองลักษณะคือ

- Entity Integrity กล่าวว่า

  "แอตทริบิวต์ทุกตัวที่เป็นส่วนของคีย์หลักจะไม่อนุญาตให้มีค่าเป็น NULL"

  หมายความว่า คีย์หลักของทุกรีเลชั่นจะไม่สามารถเก็บค่าข้อมูลที่เป็นค่าว่างได้ เหตุผลของข้อกำหนดนี้คือ เพื่อให้การเข้าถึงข้อมูลในทูเปิลใดๆ ของรีเลชั่นมีความเป็นไปได้เสมอ เพราะถ้าคีย์หลักของทูเปิลใดมีค่าข้อมูลเป็นค่าว่างแล้ว ก็จะส่งผลให้การเข้าถึงข้อมูลในทูเปิลนั้นไม่สามารถกระทำได้อย่างแน่นอน

- Referential Integrity กล่าวว่า

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

ก. มีค่าเท่ากับค่า PK ในทูเปิลใดทูเปิลหนึ่งในรีเลชั่น R1 หรือ

ข. มีค่าของแอตทริบิวต์ทุกตัวใน FK เป็น NULL"

หมายความว่า แอตทริบิวต์ใดๆที่เป็นคีย์หลักของรีเลชั่นหนึ่ง เมื่อมีการนำเอาแอตทริบิวต์นั้นไปเป็นคีย์นอกของอีกรีเลชั่นหนึ่ง การเป็นคีย์นอกของแอตทริบิวต์นั้นจะต้องมีโดเมนเป็นโดเมนเดียวกันกับแอตทริบิวต์ที่เป็นคีย์หลัก ทั้งนี้ก็เพื่อให้การนำรีเลชั่นมาใช้งานร่วมกัน (การนำรีเลชั่นมา Join กัน) กระทำได้อย่างถูกต้อง คือ ทุกแอตทริบิวต์ที่เป็นคีย์นอกจะต้องมีข้อมูลซ้ำกับข้อมูลของแอตทริบิวต์ที่เป็นคีย์หลักอย่างแน่นอน แต่อาจมีบางค่าข้อมูลของแอตทริบิวต์ที่เป็นคีย์หลักเป็นข้อมูลไม่อยู่ในโดเมนของแอตทริบิวต์ที่เป็นคีย์นอกก็ได้ นั่นคือ โดเมนของคีย์นอกจะต้องเล็กกว่าหรือเท่ากับโดเมนของคีย์หลักเสมอ

นิยามคำศัพท์ที่เกี่ยวข้องเพิ่มเติม

คีย์คู่แข่ง (Candidate Key) คือ แอตทริบิวต์หรือกลุ่มของแอตทริบิวต์ที่มีคุณสมบัติ Unique (กล่าวคือสองแถวหรือทูเปิลใดๆ ต้องไม่มีค่าซ้ำกัน) และมีจำนวนแอตทริบิวต์ให้น้อยที่สุดที่ยัง Unique

คีย์หลัก (Primary Key) คือคีย์คู่แข่ง (Candidate Key) ตัวหนึ่งที่ถูกเลือกขึ้นมา มักเลือกคีย์ที่สั้น, กะทัดรัด, อ้างอิงได้ง่าย

คีย์นอก คือ แอตทริบิวต์ที่ไม่ได้เป็นคีย์ของรีเลชั่นหนึ่ง แต่ไปเป็นคีย์หลักของอีกรีเลชั่นหนึ่ง (หรือรีเลชั่นเดียวกัน)

2.1.3 การจัดการกับข้อมูล (Data Manipulation)

ภาษาที่ใช้ในการจัดการกับโครงสร้างข้อมูลที่เป็นรีเลชั่นในระบบฐานข้อมูลเชิงสัมพันธ์ที่นิยมใช้และเป็นรู้จักกันอย่างแพร่หลายได้แก่ SQL (Structured Query Language) ซึ่งสามารถอธิบายได้ในรูปแบบของคณิตศาสตร์ มีมาตรฐานรองรับเกี่ยวกับภาษานี้

แบบจำลองเชิงสัมพันธ์เป็นโมเดลที่มีโครงสร้างข้อมูลเป็นรีเลชั่นแต่ละแอตทริบิวต์ต้องเป็นอะตอมมิกไม่อนุญาตให้มีเซตหรืออาร์เรย์ กล่าวคือทุกๆแอตทริบิวต์ต้องเป็น Atomic Value ด้วยเหตุนี้การใช้แบบจำลองเชิงสัมพันธ์เพื่อแทนออบเจ็กต์ที่มีความซับซ้อนอย่าง เช่นเซต

หรืออาร์เรย์จะทำได้ลำบาก ต้องมีการทำนอร์มอลไลซ์เซชั่น (Normalization) เพื่อลดความซ้ำซ้อน และการทำเช่นนี้บางครั้งจะทำให้สูญเสีย Semantic ของออบเจ็กต์ไปบางส่วน

ในฐานข้อมูลเชิงสัมพันธ์ต้องระบุหรือเลือกแอตทริบิวต์เพื่อแสดงเอกลักษณ์ของแต่ละทูเปิลกล่าวคือเป็นค่าที่ไม่ซ้ำกันเลยในรีเลชั่นนั้นๆ หากค่าของคีย์ที่แสดงเอกลักษณ์นี้มีการเปลี่ยนแปลงจะทำให้เกิดปัญหาได้โดยเฉพาะกับคีย์ที่มีลักษณะเป็น Meaningful Digit ซึ่งใช้กันมากในระบบงานสารสนเทศทั้งของภาครัฐบาลและเอกชน หากต้องการใช้คีย์ประเภทนี้ก็ไม่ควรอนุญาตให้มีการแก้ไขเปลี่ยนแปลงค่าคีย์นี้ หากจะแก้ปัญหานี้ในระบบฐานข้อมูลเชิงสัมพันธ์อาจใช้ตัวเลขที่เป็นลำดับ (Sequence Number) แทนการใช้ Meaningful Digit เพื่อป้องกันปัญหาที่อาจเกิดขึ้นในภายหลัง

2.2 หลักการที่สำคัญของ Object Technology

Object-Oriented เป็น Paradigm ใหม่ในการจัดการกับปัญหาโดยมองเป็นออบเจ็กต์ ซึ่งกำลังได้รับความนิยมในปัจจุบัน และมี Tool ที่สำคัญคือ Object Model ซึ่งเป็นการรวมข้อมูล (Data) กับวิธีการ (Process) เข้าไว้ด้วยกัน ซึ่งภาษาโปรแกรมที่สนับสนุนเทคโนโลยีนี้อย่างเช่น C++, Smalltalk, Eiffel, และ Java

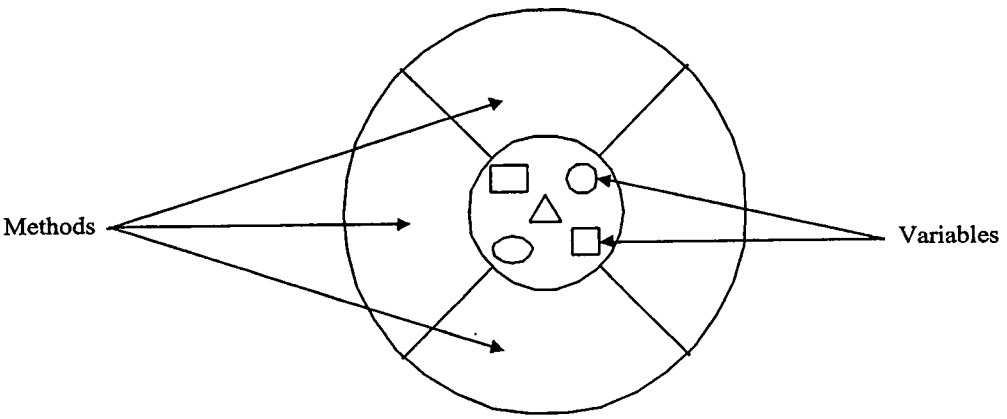
2.2.1 Data Abstraction

Data Abstraction เป็นกระบวนการของการกำหนดชนิดข้อมูล ซึ่งเรียกว่า Abstract Data Types (ADTs) ซึ่งไปด้วยกันกับหลักการของการซ่อนข้อมูล (Principle of Information Hiding)

ADT ซ่อนการเก็บทางกายภาพของข้อมูลสารสนเทศจากผู้ใช้ โดยจัดหาเซตสาธารณะของวิธีการในการที่จะใช้จัดการกับแอตทริบิวต์ของออบเจ็กต์ ซึ่งกระบวนการเช่นนี้เรียกว่า Information Hiding ทำให้มั่นใจได้ว่าโครงสร้างภายในของ ADT สามารถถูกเปลี่ยนแปลงได้โดยไม่มีผลต่อออบเจ็กต์ที่เกี่ยวข้องอื่นๆ

2.2.1.1 Object (Real World Object)

ออบเจ็กต์ คือ ทุกๆสิ่งที่อยู่รอบๆตัวเราที่อยู่ใน Real-world เช่น จักรยาน, ทีวี, สุนัข เป็นต้น ตามหลักความคิดพื้นฐานทางด้านฐานข้อมูลนั้น ออบเจ็กต์ก็คือแนวความคิดของ Entity นั่นเอง ทุกๆออบเจ็กต์จะมีคุณลักษณะอยู่ 2 ประการ ได้แก่ สถานะ (State) และ พฤติกรรม (Behavior) ตัวอย่างเช่น สุนัขมีอาจมี State เป็น ชื่อ, สี, พันธุ์ และมี Behavior คือ การเห่า, การกิน, การคุ้ยดิน เหล่านี้เป็นต้น



รูปที่ 1 แสดงออบเจ็กต์

Variables ถูกวางไว้ในตำแหน่งศูนย์กลางที่ถูกล้อมรอบไปด้วย Methods ซึ่ง Variables และ Methods เหล่านี้คือ Instance Variables และ Instance Methods ที่แตกต่างไปจาก Class Variables และ Class Methods จะเห็นได้ว่าศูนย์กลางของออบเจ็กต์หรือ Variables นี้จะถูกซ่อนจากออบเจ็กต์อื่นๆในโปรแกรม ซึ่งเป็นคุณสมบัติที่เรียกว่า Encapsulation ซึ่งเป็นการซ่อนรายละเอียดที่ออบเจ็กต์อื่นไม่จำเป็นต้องทราบ ด้วยความสามารถเช่นนี้ทำให้เกิดประโยชน์ 2 ประการคือ

- Modularity

  ซอร์สโค้ดของออบเจ็กต์สามารถถูกเขียนขึ้นและบำรุงรักษาได้โดยอิสระจากซอร์สโค้ดของออบเจ็กต์อื่น

- Information Hiding

  ออบเจ็กต์ติดต่อสื่อสารกันโดยผ่าน Public interface ทำให้ออบเจ็กต์สามารถดูแลเปลี่ยนแปลงข้อมูลสารสนเทศและ method ที่เป็น private ได้โดยไม่กระทบต่อออบเจ็กต์อื่นๆ ที่เรียกใช้ออบเจ็กต์นี้

2.2.1.2 Classes และ Instances

Class คือ กลุ่มของวัตถุโดยแบ่งตามลักษณะเฉพาะ และการใช้งาน หรือออบเจ็กต์ที่มีคุณสมบัติพื้นฐานเหมือนกัน ซึ่งข้อมูลเชิงวัตถุที่เก็บอยู่ภายในคลาสจะเรียกว่า Instance ของคลาส โดยที่ Instance จะเป็นส่วนขยายของคลาส และถูกเก็บอยู่ในส่วนของฐานข้อมูล ดังนั้นข้อมูลต่างๆที่เป็นตัวกำหนดคลาสจะถูกเข้าถึง และถ่ายทอดผ่านทางออบเจ็กต์ Instance และ Class Method

- Instance Method คือ โอเปอร์เรชั่นที่ถูกประยุกต์ใช้งานกับ Instance แต่ละตัวในคลาสที่กำหนด

- Class Method คือ วิธี (Method) ที่ถูกประยุกต์ใช้กับส่วนของคลาสทั้งหมด

2.2.1.3 Attributes

Attributes คือ โครงสร้างของฟิลด์ภายในทูเปิลที่ให้สื่อทางด้านความหมายของแต่ละทูเปิล สำหรับแอตทริบิวต์จะถูกแปลงไปเป็น Atomic Object หรือกลุ่มของ Atomic Object ซึ่งแอตริบิวต์จะแบ่งออกเป็น 2 ประเภท คือ

- Class Attributes คือ แอตทริบิวต์ของคลาสนั้นๆ ที่ถูกเข้าถึงโดยตัวคลาสเอง, Instance และ Subclass ของตัวคลาส
- Instance Attributes คือ แอตทริบิวต์ที่เข้าถึงโดย Instance ของคลาสที่กำหนดเท่านั้นซึ่งจะคล้ายคลึงกับแอตทริบิวต์ในโครงสร้างอื่นๆ โดยที่ Instance Attributes นั้นจะมีค่าตายตัวของมันเอง (Default Instance Attributes) หรืออาจจะมีค่าสากลที่ประยุกต์ใช้กับแต่ละ Instance ของคลาสได้ (Shared Instance Attributes)

2.2.1.4 Method

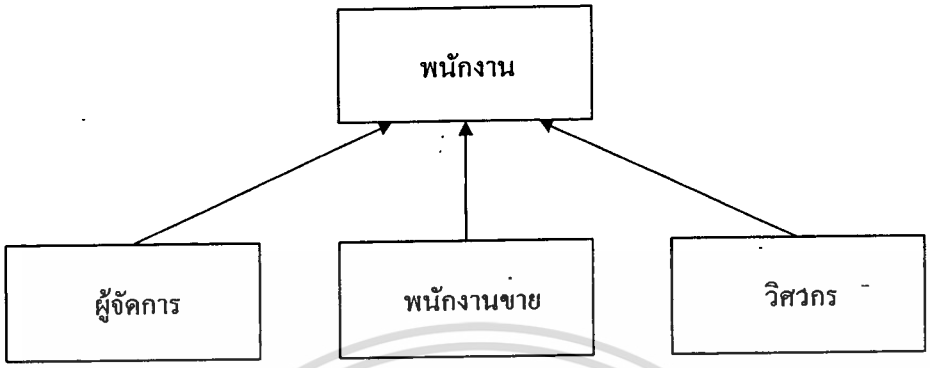
ฟังก์ชันที่ทำกับออบเจ็กต์จะเรียกว่า Method ซึ่งจะเป็นการกำหนดพฤติกรรมที่ออบเจ็กต์สามารถทำได้

2.2.2 Inheritance

Inheritance เป็นคุณสมบัติในการสืบทอดคุณสมบัติ กล่าวคือ Subclass จะสืบทอด Method และ Variable ทุกอย่างมาจาก Superclass ถ้าต้องการแก้ไขคุณสมบัติต่างๆทั้งหมดก็สามารถทำได้โดยการแก้ไขที่ Superclass เท่านั้น แต่ผลที่เกิดขึ้นจากการแก้ไขจะถ่ายทอดไปยัง Subclass ด้วย Subclass สามารถมี Method และ Variable เพิ่มเติมได้อีกนอกเหนือไปจากที่ได้โดยการสืบทอดจาก Superclass Subclass สามารถมีชื่อ Method หรือ Variable เหมือนกับที่มีใน Superclass แต่มีความหมายไม่เหมือนกัน ซึ่งเรียกว่า Overriding Mechanism

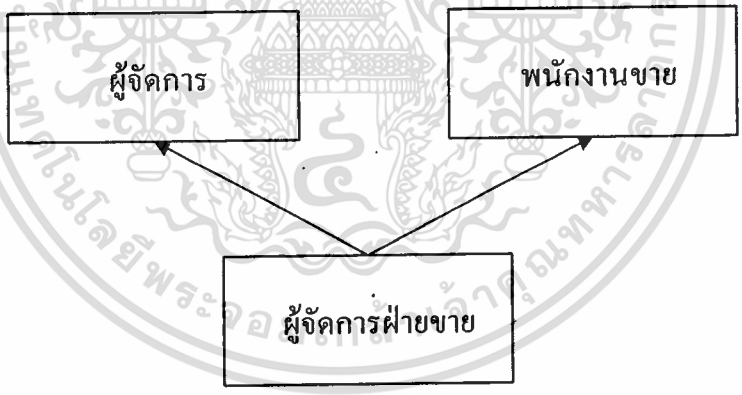
คุณสมบัติการสืบทอดนี้เป็นการนำกลับมาใช้ใหม่ (Reuse) ของคลาสซึ่งจะมีการสืบทอดกันเป็นลำดับขั้นไป (Hierarchy) Inheritance มีอยู่ 2 รูปแบบ คือ

2.2.2.1 Single Inheritance - เป็นการสืบทอดคุณสมบัติมาจากคลาสเพียงคลาสเดียว



รูปที่ 2 แสดง Single Inheritance

2.2.2.2 Multiple Inheritance - เป็นการสืบทอดคุณสมบัติมาจากคลาสมากกว่าหนึ่งคลาส



รูปที่ 3 แสดง Multiple Inheritance

สำหรับ Multiple Inheritance จะมีความยุ่งยากและอาจเกิดการขัดแย้งกันได้ เช่น ชื่อเหมือนกันอาจถูกใช้เพื่อแทน Method หรือ Variable ที่ต่างกันซึ่งถูกสืบทอดไปยัง Subclass เดียวกัน เป็นการเพิ่มความซับซ้อนให้กับโปรแกรม ภาษาโปรแกรมที่ใช้ต้องสามารถแก้ปัญหาของการเกิดการขัดแย้งในกรณีนี้ได้ ภาษาเชิงวัตถุที่สนับสนุน Multiple Inheritance เช่น C++ และ Smalltalk แต่ Java ไม่สนับสนุน

ประโยชน์ที่ได้จากคุณสมบัติการสืบทอด ได้แก่

- Reusability – โปรแกรมเมอร์สามารถนำโค้ดใน Superclass มาใช้ได้
- Extendibility – นักพัฒนาสามารถระบบเพิ่มเติมต่อจากระบบที่มีอยู่ ระบบถูกออกแบบให้มีลักษณะเป็นโมดูลสามารถเพิ่มขยายได้ซึ่งทำให้เกิดความสะดวกและง่ายต่อการแก้ไขเมื่อความต้องการเปลี่ยนไป

2.2.3 Polymorphism

สำหรับ Object-Oriented Programming คำว่า Polymorphism จะหมายถึง Single Operation สามารถมีได้หลาย Behavior ในออบเจ็กต์ที่แตกต่างกัน หรืออาจกล่าวได้ว่า ออบเจ็กต์ต่างกันจะมีการตอบสนองที่ต่างกันต่อข้อความ (Message) ที่เหมือนกัน เช่น โอเปอร์เรชั่นของการบวกอาจเป็นกระบวนการที่ไม่ใช่การบวกเลขสองจำนวนและให้ผลลัพธ์คือผลบวก แต่อาจเป็นการนำสตริงมาเรียงต่อกันก็ได้

Polymorphism เป็นคุณสมบัติที่อำนวยความสะดวกในแง่ของการนำกลับมาใช้ วิธีที่ภาษาเชิงวัตถุใช้ในการ Implement คุณสมบัติ Polymorphism จะเรียกว่า Dynamic Binding โดยที่ Variables, โพรซีเยอร์, ชนิดข้อมูล, ฟังก์ชันหรือ Method จะมีผลในตอน Run-time ของโปรแกรม

2.3 Relationship

Relationship เป็นความสัมพันธ์ระหว่างออบเจ็กต์ที่ต่างกันใน Real World

2.3.1 Relationships ระหว่าง Single Object ซึ่งมีอยู่ 3 รูปแบบที่เป็นพื้นฐาน คือ

- 1:1 (One-to-One Relationship)
- 1:M (One-to-Many)
- M:M (Many-to-Many)

2.3.2 Relationships ในหมู่ ออบเจ็กต์ภายในระบบ ซึ่งมีอยู่ 3 ชนิดที่เป็นพื้นฐาน คือ

2.3.2.1 Generalization/Specification Relationships

เป็นความสัมพันธ์ในการจัดกลุ่มสิ่งที่เหมือนกันระหว่าง Object หรือ Class ขึ้นมาเป็นอีก Object หนึ่งโดยพิจารณาจากข้อมูล หรือ พฤติกรรม (Behavior) ในกลุ่มของ Object นั้น จะมีลักษณะเป็นลักษณะทั่วไปไม่ชี้เฉพาะเจาะจง ซึ่งความสัมพันธ์แบบ Generalization/Specification จะอยู่ในรูปแบบดังนี้

- <subclass> is a (kind of) <superclass> เช่น รถยนต์เป็นรถ, รถบรรทุกเป็นชนิดหนึ่งของรถ เป็นต้น

- <superclass> is (can be a <subclass1> or <subclass2> ... เช่น รถจะเป็นรถยนต์หรือรถบรรทุก เป็นต้น

2.3.2.2 Aggregation Relationships

เป็นรูปแบบความสัมพันธ์ที่ถูกใช้ในการกำหนด Part ใน Component นั้น โดยมีการแยกระหว่าง Whole กับ Part ซึ่งจะได้ความสัมพันธ์แบบ "part-whole" หรือ "a part of" ซึ่งความสัมพันธ์ในลักษณะเช่นนี้ Superclass จะมีอิทธิพลต่อการดำรงอยู่ของ Subclass กล่าวคือ ถ้า Superclass ถูกทำลาย Subclass จะถูกทำลายไปด้วยโดยอัตโนมัติ ซึ่งความสัมพันธ์แบบ Aggregation จะอยู่ในรูปแบบดังนี้

- <component> is part of <assembly> เช่น ประตูเป็นส่วนหนึ่งของรถยนต์, หลอดภาพเป็นส่วนหนึ่งของจอภาพ เป็นต้น
- <assembly> contains <cardinality1> <component1> and <cardinality2> <component> เช่น รถมีหลายประตูและหลายล้อ เป็นต้น

2.3.2.3 Association Relationships

เป็นการอธิบายถึงความสัมพันธ์ระหว่างโครงสร้างกับ semantic ทั่วๆไป โดยอ้างถึงความสัมพันธ์ทั้งสองด้าน และมีการระบุชื่อของความสัมพันธ์หรือบทบาทหน้าที่ (Role) เพื่อให้เกิดความชัดเจน และเข้าใจง่าย ความสัมพันธ์แบบ Association จะอยู่ในรูปแบบดังนี้

- <role1> of <class1> is <cardinality1> <class2> เช่น อาจารย์สอนในหลายวิชา เป็นต้น

2.4 Message Passing

ออบเจ็กต์แต่ละออบเจ็กต์สามารถติดต่อสื่อสารกันได้โดยผ่านวิธีการที่เรียกว่า "การส่งผ่านข้อความ" (Message Passing) ที่เป็นตัวกระตุ้นการตอบสนองโดยผ่านโอเปอร์เรชั่นที่ซ่อนอยู่ในตัวของวัตถุที่เป็นผู้รับ นั่นคือ ข้อมูลที่จัดเก็บอยู่ภายในวัตถุผู้รับจะถูกเข้าถึง และดำเนินการโดยผ่านโอเปอร์เรชั่นของวัตถุนั้นเอง

Message Passing เปรียบได้กับ Function Call หรือ Procedure Call ที่มีใน Structured Programming โดยผ่าน Interface Method ของ Object นั้นๆ มีผลทำให้ Object ที่เป็นผู้รับข้อความ (Received Object) นั้นกระทำการอย่างใดอย่างหนึ่ง

2.5 หลักการที่สำคัญของ Object Technology

ในการพัฒนาแอปพลิเคชั่นด้วยเทคโนโลยีออบเจ็กต์นั้น ในปัจจุบันได้มีเครื่องมือช่วยเหลือและเทคนิควิธีการต่างๆ ออกมามากมาย แต่ละแบบล้วนมีวิธีการของตนเองในการอำนวยความสะดวกและเพิ่มความรวดเร็วในการสร้างผลงาน อย่างไรก็ตาม ดังที่ได้กล่าวมาแล้ว Component นับเป็นสิ่งสำคัญ เทคโนโลยีใดรองรับคอมโพเนนต์ และมีความยืดหยุ่นมากเท่าใด เทคโนโลยีนั้นก็จะประสพความสำเร็จมากขึ้นเท่านั้น

ในปัจจุบันเทคโนโลยีคอมโพเนนต์ที่เห็นเด่นชัดและได้รับความนิยมแพร่หลายมีด้วยกัน 2 พวก คือ ActiveX Control และ JavaBeans Technology

2.6.1 ActiveX Control เป็นเทคโนโลยีของบริษัทไมโครซอฟต์ที่ใช้เป็นเครื่องมือในการพัฒนาแอปพลิเคชั่นจำนวนมาก แต่ยังขาดเครื่องมือในการพัฒนาคอมโพเนนต์ที่ง่ายต่อการสร้าง การพัฒนาแอปพลิเคชั่นโดยใช้ ActiveX Technology นั้นง่าย แต่การสร้าง Component ขึ้นมาใช้เองนั้นมีความยุ่งยากและเข้าใจได้ยาก และขาดหลักการของ Object ที่สำคัญไป คือ Inheritance อย่างไรก็ตาม การใช้งานที่ง่ายและมีเครื่องมือที่สามารถใช้ ActiveX Control จำนวนมาก อีกทั้งมี ActiveX Control ที่ผลิตออกมาในท้องตลาดจำนวนมากทำให้ ActiveX Technology ยังได้รับความสนับสนุน และได้รับความนิยมต่อไปอีกนาน

เครื่องมือที่รองรับ ActiveX Control ในปัจจุบันนี้จะเป็นเครื่องมือพัฒนาทั้งหลายจาก บริษัทไมโครซอฟต์ ได้แก่ Visual Basic, Visual C++ , Visual InterDev, Microsoft Office เครื่องมือพัฒนาจากบริษัท Borland คือ Delphi, C++ Builder นอกจากนี้ยังมีผู้ผลิตชิ้นส่วนซอฟต์แวร์อีกเป็นจำนวนมากที่ผลิตชิ้นส่วนซอฟต์แวร์ออกมาสนับสนุนเครื่องมือพัฒนาเหล่านี้ ได้แก่ บริษัท Sheridan, MicroHelp, Crescent, Crystal Software เป็นต้น

2.6.2 JavaBeans Technology เป็นเทคโนโลยีของบริษัท ซัน ไมโครซิสเต็มส์ ซึ่งเป็นเทคโนโลยีในการสร้าง และใช้งาน Component โดยใช้ภาษา Java เทคโนโลยีนี้ค่อนข้างใหม่ แต่ก็ได้รับการสนับสนุนจากผู้ผลิตหลายๆราย อย่างเช่น IBM, Borloand, Netscape, Oracle และบริษัทอื่นๆ อีกเป็นจำนวนมาก JavaBeans เป็นเทคโนโลยีที่มีความยืดหยุ่นในการสร้างมากกว่าเครื่องมือที่รองรับ JavaBeans Technology ได้แก่ JavaStudio ของซันไมโครซิสเต็มส์, JBuilder ของบริษัทบอร์แลนด์, IBM VisualAge for Java ของบริษัท IBM และ Visual Café ของบริษัท Symantec นอกจากนี้ยังมีบริษัทที่ผลิตชิ้นส่วนซอฟต์แวร์สำหรับเครื่องมือพัฒนาเหล่าอีกได้แก่ KL Group, Bristrol Software

บทที่ 3

ระบบการจัดการฐานข้อมูลเชิงวัตถุ

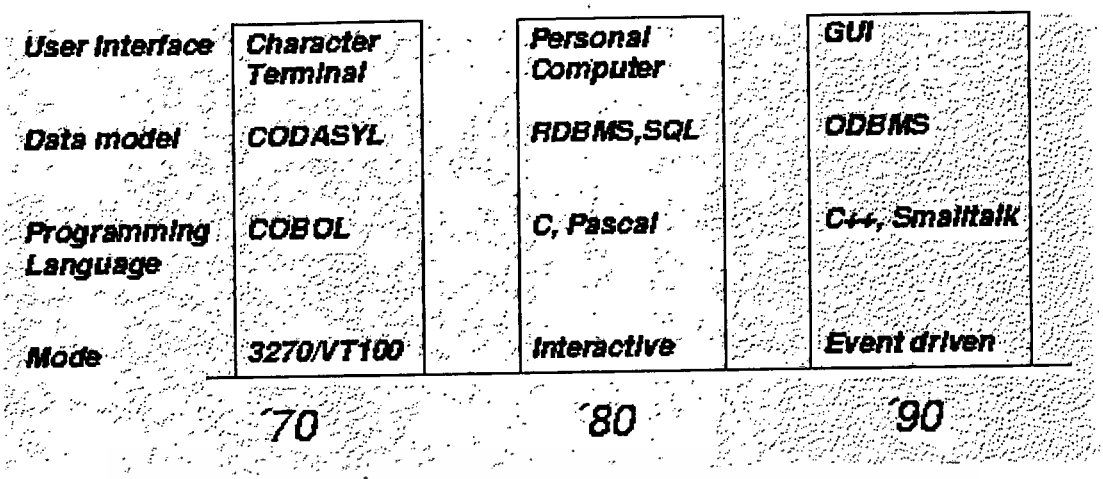
3.1 บทนำ

ระบบการจัดการฐานข้อมูลเชิงวัตถุ (Object-Oriented Database Management System : OO-DBMS) คือ ระบบการจัดการเกี่ยวกับฐานข้อมูลในรูปแบบของ Object ซึ่งต่างจากระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database Management System : RDBMS) ซึ่งเป็นระบบการจัดการเกี่ยวกับฐานข้อมูลในรูปแบบของตาราง (Table) ระบบฐานข้อมูล RDBMS นั้นไม่ได้ถูกออกแบบมาเพื่อการเก็บข้อมูลในลักษณะของออบเจ็กต์โดยตรง ในขณะที่ OO-DBMS นั้นถูกออกแบบมาสำหรับการเก็บข้อมูลในรูปแบบของออบเจ็กต์ โดยเฉพาะ ซึ่งทำให้มีประสิทธิภาพในการจัดเก็บข้อมูลที่มีลักษณะเป็นออบเจ็กต์ได้ดีกว่า RDBMS

3.2 วิวัฒนาการของระบบการจัดการฐานข้อมูลเชิงวัตถุ

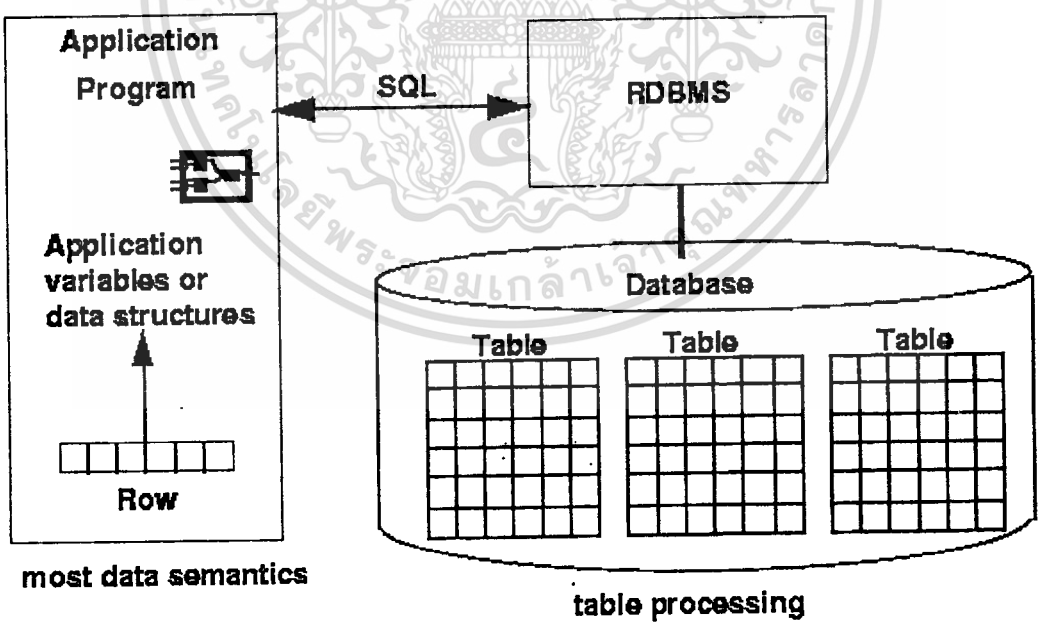
ในการพัฒนา Software ระบบฐานข้อมูลในอดีต ส่วนมากการเก็บข้อมูลนั้นอ้างอิงอยู่กับระบบฐานข้อมูลแบบ RDBMS ซึ่งเริ่มมีการพัฒนามาตั้งแต่ปี ค.ศ. 1980 ดังรูปที่ 4 และการพัฒนาทางด้าน Programming ของระบบฐานข้อมูลได้เริ่มเปลี่ยนมาเป็น Object - Oriented Programming Languages ซึ่งเป็นการพัฒนา Software ในรูปแบบของ Object Model แทนรูปแบบเดิม ซึ่งเป็นการพัฒนาในรูปแบบของ Relational Model และจากจุดนี้เองทำให้เกิดการเปลี่ยนแปลงขึ้น และนำไปสู่การพัฒนา Software ระบบฐานข้อมูลแบบใหม่ขึ้นมา คือ ระบบการจัดการฐานข้อมูลเชิงวัตถุ ซึ่งสนับสนุนในเรื่องของการเก็บข้อมูลในลักษณะของออบเจ็กต์ และสนับสนุน Object - Oriented Programming Languages โดยตรง

ในปัจจุบันมีผลิตภัณฑ์ทางด้านระบบการจัดการเกี่ยวกับฐานข้อมูล ของผู้ขายบริษัทต่างๆ แพร่หลายอยู่ในตลาดมากมาย ซึ่งมีทั้ง Relational Database Management System (RDBMS) และ Object - Oriented Database Management System (OO-DBMS)



รูปที่ 4 แสดงวิวัฒนาการของ OO-DBMS

เนื่องจากความเจริญก้าวหน้าทางด้านเทคโนโลยีในการพัฒนาระบบ Software โดยเฉพาะทางด้าน Programming มีการพัฒนามาใช้ Object - Oriented Programming Language ทำให้เกิดปัญหาเกี่ยวกับการเก็บข้อมูลลงในระบบฐานข้อมูลในรูปแบบของ Relational เพราะว่า Relational Database เป็นฐานข้อมูลในรูปแบบของ Relational Model ซึ่งมีการเก็บข้อมูลต่างๆ อยู่ในรูปแบบของตารางที่มีความสัมพันธ์กัน ดังแสดงไว้ในรูปที่ 5

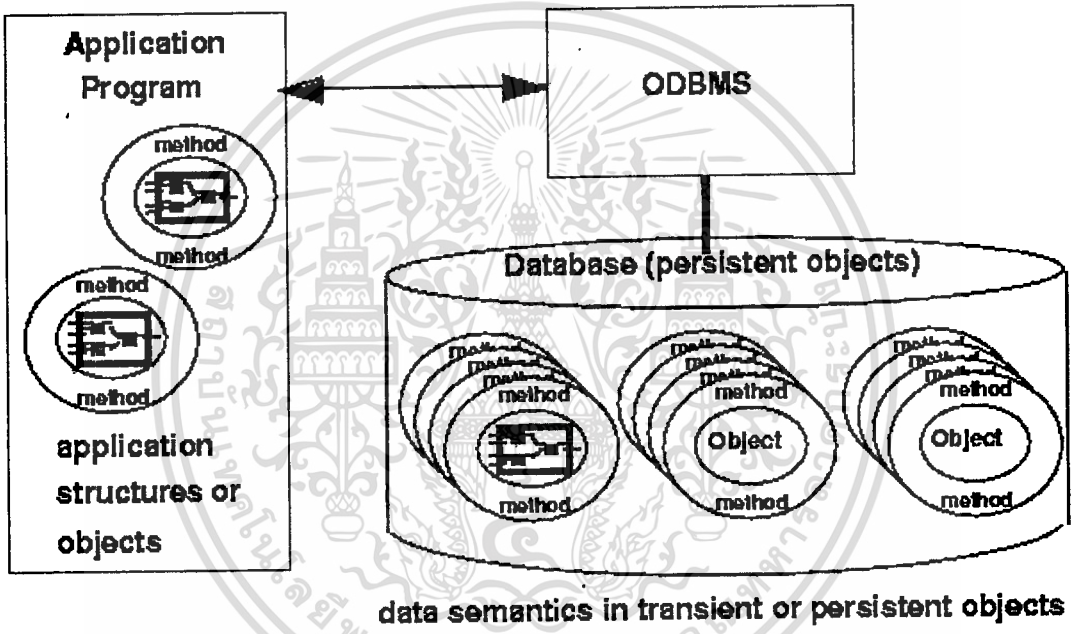


รูปที่ 5 แสดง RDBMS

ส่วนการพัฒนาโดยใช้ Object - Oriented Programming เป็นการมองลักษณะของข้อมูลในรูปแบบของ Object Model โดยที่แต่ละออบเจ็กต์มีความสัมพันธ์กันโดย Reference ทำให้มีความ

ลำบากและยุ่งยากในการเก็บข้อมูลในลักษณะของออบเจ็กต์ลงในฐานข้อมูลที่มีลักษณะเป็น Relational

แนวคิดทั่วไปของ OO-DBMS เป็น DBMS ที่ไม่ใช่เก็บข้อมูล แต่เป็นการเก็บออบเจ็กต์ ซึ่งเป็นการรวมกันของโครงสร้างข้อมูลเข้าด้วยกันกับ Method ที่มีความสัมพันธ์เกี่ยวข้องกัน OO-DBMS จะจัดทำการรวมข้อมูล (Data Integration), การควบคุมทั้งหมด (Overall Control) และ DBMS ที่สนับสนุนความสะดวกสำหรับทุกประเภทของออบเจ็กต์ แอปพลิเคชั่นจะเรียกใช้ออบเจ็กต์ใดๆ ก็ได้โดยการติดต่อไปที่ OO-DBMS ดังแสดงไว้ในรูปที่ 6



รูปที่ 6 แสดง OO-DBMS

ดังนั้นระบบการจัดการฐานข้อมูลเชิงวัตถุซึ่งเป็นระบบการจัดการฐานข้อมูลแบบ Object Model และให้การสนับสนุนการพัฒนา Software ระบบฐานข้อมูลในลักษณะของ Object - Oriented Programming จึงเป็นสิ่งที่ดี และเหมาะสมกว่าระบบฐานข้อมูลเชิงสัมพันธ์

3.3 ลักษณะพื้นฐานของระบบฐานข้อมูลเชิงวัตถุ

3.3.1 Complex Objects (General and Part-of)

ออบเจ็กต์ที่มีความซับซ้อน แต่ละทูเปิลนั้นจะประกอบด้วยแอตทริบิวต์ และแต่ละแอตทริบิวต์มีค่าที่ประกอบด้วย Composite Element อื่นๆ

Composite Objects ที่รู้จักกันดี คือ Part-of โดยที่ Composite Object ซึ่งเป็นการอ้างถึงออบเจ็กต์ตัวอื่นๆ หรือที่เรียกว่า Composite Reference และโอเปอร์เรชั่นสำหรับ Composite Object ได้แก่

- Compose/decompose a composite objects
- Copy a composite objects

3.3.2 User-Defined Operations

ลักษณะที่แตกต่างของภาษาเชิงวัตถุ คือ การจัดเตรียมให้กับผู้ใช้ในการกำหนดคลาส และความสัมพันธ์ของโอเปอร์เรชั่น

โครงสร้างการจัดการเชิงวัตถุนั้น โอเปอร์เรชั่นสามารถกำหนดโดยผู้ใช้ ซึ่งโอเปอร์เรชั่นเหล่านั้น ได้แก่

- วิธีการต่างๆ ที่ซ่อนอยู่ภายใต้ออบเจ็กต์
- ผู้ใช้ได้กำหนดโอเปอร์เรเตอร์ขึ้นมาสำหรับ User-defined Type

3.3.3 Encapsulation

Encapsulation คือ คุณสมบัติที่ Object สามารถป้องกันข้อมูล (Data) ที่อยู่ภายใน Object นั้นๆ ไม่ให้ถูกเปลี่ยนแปลงหรือแก้ไขได้โดยตรงจาก Object อื่นที่อยู่ภายนอก เป็นการซ่อนข้อมูลจากภายนอก (Data Hiding) ยกเว้นมีการร้องขอผ่านทาง Method ที่ Object นั้นมีการเตรียมไว้ให้เท่านั้น นั่นหมายความว่า เมื่อ Object นั้นถูกอ่านขึ้นมาจากฐานข้อมูล ก็จะมีข้อมูลทุกอย่างเหมือนกับตอนที่ Object นั้นถูกจัดเก็บลงในฐานข้อมูล

ข้อดีของการทำ Encapsulation คือ สามารถแบ่งระดับของการเข้าถึงข้อมูลได้ด้วยตัว Methods ของวัตถุเอง

3.3.4 Inheritance

Inheritance คือ การที่ Class หนึ่ง (เรียกว่า Subclass) มีการสืบทอดคุณสมบัติทุกอย่างมาจากอีก Class หนึ่ง (เรียกว่า Superclass) โดยที่ Subclass สามารถมีคุณสมบัติอื่นเพิ่มเติมได้อีกนอกเหนือไปจากที่ได้สืบทอดจาก Superclass

- Class และ Instance การขยายความของคลาสจะนำเสนอโดยกลุ่มของ Instance ที่มีคุณสมบัติเดียวกัน นั่นคือ ความสัมพันธ์ของคลาสแม่ (Superclass) และคลาสลูก (Subclass) จะถูกนำเสนอออกมาโดยผ่านทาง Instance
- Attributes คลาสลูกจะสืบทอดแอตทริบิวต์ต่างๆจากคลาสแม่โดยตรง นั่นคือ ถ้ามีการเปลี่ยนแปลงแอตทริบิวต์ใดๆที่คลาสแม่แล้ว คลาสลูกจะได้รับการ

เปลี่ยนแปลงแอตทริบิวต์นั้นโดยตรงด้วย การสืบทอดแอตทริบิวต์จะสืบทอดทั้งชื่อ, ข้อจำกัด และความสัมพันธ์ต่างๆภายในแต่ละแอตทริบิวต์ ซึ่งข้อจำกัดที่คลาสลูกได้รับนั้นอาจจะเป็นเพียงส่วนหนึ่งของคลาสแม่ หรือได้รับการถ่ายทอดมาทั้งหมด ซึ่งจะรวมทั้งค่า Default Value ด้วย

- Relationships การสืบทอดคุณสมบัติความสัมพันธ์จะรวมในส่วนของชื่อ, ชนิดของความสัมพันธ์, ข้อจำกัดต่างๆ และความสัมพันธ์ที่เกี่ยวข้องกับคลาสอื่นๆ
- Operations โอเปอร์เรชั่นจะถูกสืบทอดโดยคลาสลูก ซึ่งจะเกี่ยวข้องกับ
    1. คลาสลูกสามารถเพิ่มโอเปอร์เรชั่นเข้าไปได้โดยอัตโนมัติ
    2. โอเปอร์เรชั่นของคลาสลูกสามารถเป็นส่วนขยายของคลาสแม่ได้

3.3.5 Object Persistency

Object Persistency เป็นทั้งคุณสมบัติพื้นฐาน และลักษณะเด่นของระบบฐานข้อมูลเชิงวัตถุ ซึ่ง Persistency คือ ความสามารถของออบเจ็กต์ที่จะสืบทอดชีวิตอยู่ภายในระบบในขณะที่ทำการประมวลผลโปรแกรม

- Persistent Object จะไม่ถูกทำลายเมื่อโปรแกรมได้ถูกกำหนดจุดสิ้นสุดไว้ ซึ่งจะเกี่ยวข้องกับ Internal garbage collection
- Persistent Object จะประกอบด้วยชื่อที่เป็นสากล ซึ่งสามารถถูกอ้างถึงได้โดยโปรแกรมอื่นๆ

3.3.6 Polymorphism

Polymorphism คือ การที่ Objects ที่ต่างกันสามารถแสดงพฤติกรรมตอบสนองที่ต่างกันเมื่อได้รับ Message ชนิดเดียวกัน หรืออาจกล่าวได้ว่าการที่คลาสหนึ่งๆ สามารถแปรเปลี่ยนไปได้หลายรูปแบบ ขึ้นกับสภาพแวดล้อม หรือสถานการณ์ในขณะนั้น เช่น ในกรณีของการแสดงข้อมูลของ Objects 2 Objects ซึ่งทั้ง 2 Objects ต่างก็ได้รับ Message ชนิดเดียวกัน คือ display() ซึ่ง Object หนึ่งอาจจะเป็นการแสดงข้อมูลในรูปแบบของรายละเอียดของข้อมูล (Text) ในขณะที่อีก Object หนึ่งอาจจะเป็นการแสดงข้อมูลในรูปแบบของรูปภาพ (Graphic) ความสามารถอีกอย่างหนึ่งของ Polymorphism คือ สามารถกำหนดการดำเนินการใหม่ให้กับตัวดำเนินการ หรือแม้แต่ฟังก์ชันได้ เรียกว่าการทำ Overloading ซึ่งเป็นการเรียกใช้ฟังก์ชัน หรือตัวดำเนินการเดิมให้ไปทำงานอีกอย่างหนึ่งได้

3.3.7 Object Identity

Object แต่ละ Object ใน OO-DBMS จะมี Logical Object Identifier (LOID) เป็นเลขที่บอก ID ของ Object นั้น โดยที่ LOID จะไม่ซ้ำกันเลย และจะไม่มีการนำกลับมาใช้ใหม่ถึงแม้ว่า Object นั้นจะถูกลบออกไปจากฐานข้อมูลแล้วก็ตาม นั่นคือ Object Identity จะสนับสนุนในด้านความถูกต้องของข้อมูล กล่าวคือ การเปลี่ยนแปลงใดๆที่เกิดขึ้นกับค่า หรือโครงสร้างจะไม่ส่งผลกระทบกับตัวกำหนดออบเจ็กต์ ซึ่งเป็นตัวแสดงว่าข้อมูลเป็นอิสระต่อกัน

ในทางกลับกันเมื่อมีการสร้าง Object ใหม่ขึ้นมา ก็จะมีการกำหนด LOID ให้กับ Object นั้นด้วย โดยที่ LOID นี้จะไม่มีการเปลี่ยนแปลง ถึงแม้ว่าจะมีการนำ Object นั้นไปเก็บยังฐานข้อมูลตัวอื่นก็ตาม

สำหรับ LOID (ใน Versant, Object - Oriented Database Management System) ประกอบด้วยตัวเลข 64-bit ซึ่งแบ่งเป็น 2 ส่วน คือ

- ส่วนแรกเป็นตัวเลข 16-bit ที่แสดงถึงเลขของ Database ที่ไม่ซ้ำกัน
- ส่วนที่สองเป็นตัวเลข 48-bit ที่แสดงถึงเลข ID ของ Object นั้น โดยใน 48-bit นี้จะแสดงแยกเป็น 2 ส่วน คือ ส่วนแรกเป็น 16-bit (ส่วนมากเป็น 0) ส่วนที่สองเป็นเลข 32-bit ซึ่งแสดง Object ID

3.3.8 Reference Among Objects

OO-DBMS มีการอ้างอิงถึง Objects ต่างๆโดยใช้ Links ซึ่ง Link ของ Object ในฐานข้อมูลจะ แทนด้วย LOID ของ Object

3.3.9 Version

การเปลี่ยนแปลงของออบเจ็กต์จะเปลี่ยนรูปออบเจ็กต์จากสถานะหนึ่งไปยังอีกสถานะอื่นๆ ซึ่ง เวอร์ชัน ก็คือกระบวนการในการเก็บผลลัพธ์ของออบเจ็กต์ที่มีการเปลี่ยนแปลงในแต่ละสถานะอย่างรวดเร็ว คลาสที่อนุญาตให้มีการทำเวอร์ชัน เรียกว่า คลาสเวอร์ชัน (Version Class) และทุกๆ Instance ที่อยู่ภายใต้คลาสนั้นจะเรียกว่า Version Instance ก็คือ การทำเวอร์ชันให้กับคลาสที่มีความสัมพันธ์กัน การกำหนดให้คลาสเป็น Version Class จะต้องทำการกำหนดก่อน Version Instance ซึ่งการกำหนด Version Instance จะเกิดภายใต้ 3 สถานะคือ

- Transient State จะเกิดขึ้นเมื่อมีการเปลี่ยนแปลง และการลบแสดงออกมา ซึ่ง Version Instance ที่พิจารณานี้จะไม่คงทนเมื่ออยู่ในสถานะ Transient State
- Working State จะเกิดขึ้นเมื่อ Version Instance นั้นคงทน และไม่สามารถเปลี่ยนแปลงได้ แต่อย่างไรก็ตามการลบก็ยังสามารถกระทำได้

- Released State คือ Version Instance ที่กำหนดขึ้นในสถานะสุดท้าย ซึ่งจะมีความแข็งแรง คงทน (Robust) และไม่สามารถทำการเปลี่ยนแปลงใดๆ ได้ทั้งการปรับปรุง และการลบออบเจ็กต์

Version Instance ถูกนำเสนอจากสถานะหนึ่งไปอีกสถานะหนึ่ง นั่นคือ จาก Transient ไปยัง Working และไปสู่ Released State และการนำเสนอการใช้งาน Version Instance กระทำได้ 2 วิธีการ ดังนี้

- การอ้างอิง Version Instance โดยตรงโดยการใช้ Object Identifier
- การอ้างอิงกับ Generic Object และเรียกใช้งานเวอร์ชันล่าสุด

3.3.10 Basic Database Operations

โอเปอร์เรชั่นพื้นฐานของ OO-DBMS มีดังนี้

3.3.10.1 Storing Objects

Storing Object คือ การจัดเก็บข้อมูลออบเจ็กต์ลงในฐานข้อมูล ซึ่งเมื่อเปรียบเทียบกันระหว่าง RDBMS กับ OO-DBMS แล้ว OO-DBMS จะทำการจัดเก็บข้อมูลได้ง่ายกว่าและไม่ค่อยยุ่งยาก ในการเก็บข้อมูลลงใน RDBMS นั้นไม่สามารถเก็บลงไปได้โดยตรง ต้องมีการแปลงข้อมูลจากออบเจ็กต์ให้อยู่ในรูปของตารางก่อน โดยผ่านกระบวนการของการ Mapping เสียก่อน จึงจะนำไปเก็บลงในฐานข้อมูลได้ ซึ่งค่อนข้างจะยุ่งยากกว่า ซึ่งตรงกันข้ามกับ OO-DBMS ซึ่งสามารถทำการเก็บข้อมูลในรูปแบบของออบเจ็กต์ได้โดยตรง ไม่ต้องมีกระบวนการในการ Mapping

3.3.10.2 Changing Objects

Changing Objects คือ การเปลี่ยนแปลงแก้ไขข้อมูลของ Objects ซึ่งสามารถทำได้โดยการเรียก Object นั้นขึ้นมาจากฐานข้อมูลแล้วทำการแก้ไขตามต้องการ จากนั้นจึงทำการจัดเก็บ Object นั้นลงในฐานข้อมูล เนื่องจาก Object ที่ทำการแก้ไขนั้นถูกเรียกขึ้นมาจากฐานข้อมูลซึ่งมี LOID อยู่แล้ว เมื่อทำการจัดเก็บลงฐานข้อมูลก็จะทำการ Update Object นั้นแทนที่จะสร้าง Object นั้นขึ้นมาใหม่

3.3.10.3 Deleting Objects

Deleting Object คือ การลบ Objects ออกจากระบบฐานข้อมูล ซึ่งทำได้โดยการค้นหา Object ที่ต้องการจากระบบฐานข้อมูล เมื่อพบแล้วออบเจ็กต์ที่ต้องการแล้วจึงทำการลบ Object นั้นตามปกติ

3.3.10.4 Query

ในการค้นหาข้อมูลจากฐานข้อมูล ทั้ง RDBMS และ OO-DBMS จะใช้ Query Language ในการค้นหาข้อมูล ซึ่ง OO-DBMS จะได้เปรียบ RDBMS ในด้านต่อไปนี้

- ความเร็วในการค้นหาข้อมูล เช่น ในกรณีที่ทำการค้นหาข้อมูลซึ่งมีความเกี่ยวข้องสัมพันธ์กับข้อมูลในตารางอื่นอีกหลายตาราง ซึ่งทำให้ข้อมูลค่อนข้างจะซับซ้อน สำหรับ RDBMS ทำให้ต้องเสียเวลามากในการค้นหา ส่วน OO-DBMS จะใช้เวลาน้อยกว่า เนื่องจากใน OO-DBMS ออบเจ็กต์ สามารถอ้างถึงกันโดยใช้ Link ดังที่ได้กล่าวมาแล้ว เพราะฉะนั้นในการค้นหาข้อมูลก็เพียงแต่ไปที่ Object ที่ต้องการ เมื่อได้แล้วก็สามารถดูจาก Link ของ Object นั้น เพื่อที่ไปหา Object ที่ต้องการได้ จึงทำให้เร็วกว่า RDBMS ในการค้นหาข้อมูล
- OO-DBMS ช่วยลดค่าใช้จ่ายในเรื่องของการ joining เนื่องจาก Objects ใน OO-DBMS นั้นใช้ Link ในการอ้างอิงถึง Objects อื่นๆ แต่ใน RDBMS ใช้ Key ช่วยในการ joining ระหว่างตาราง จึงทำให้สิ้นเปลืองค่าใช้จ่ายในเรื่องของการ joining และมีความยุ่งยากในการจัดการกับข้อมูลที่มีความซับซ้อนมากๆ

ใน RDBMS เมื่อข้อมูลมีความสัมพันธ์ที่ซับซ้อนมากขึ้น การจัดการกับข้อมูลก็ยากมากขึ้น เช่น ในความสัมพันธ์ของ Student กับ Course ซึ่งเป็นความสัมพันธ์แบบ many-to-many ซึ่งเกิดยากแก่การจัดการใน RDBMS ทำให้ต้องมีการสร้างตารางใหม่ขึ้นมาอีกหนึ่งตารางให้ชื่อว่า Student/Course เพื่อช่วยในการจัดการกับข้อมูลที่มีความสัมพันธ์ในลักษณะนี้ แต่สำหรับ OO-DBMS เนื่องจากการเก็บข้อมูลอยู่ในรูปแบบของออบเจ็กต์อยู่แล้ว และใช้ LOIDs ในการอ้างถึงออบเจ็กต์ต่างๆ ทำให้ง่ายต่อการค้นหาและจัดเก็บข้อมูล

3.4 ข้อดีของรูปแบบข้อมูลเชิงวัตถุ

3.4.1 Extensibility

สำหรับ User-defined Types และ Overloaded Operations นั้น ส่วนของคลาสและการจัดการความสัมพันธ์จะสามารถปรับเปลี่ยน หรือเพิ่มได้ตามต้องการ การเปลี่ยนแปลงหรือการเพิ่มนี้จะถูกจัดการให้กับคลาสที่มีความสัมพันธ์กันโดยอัตโนมัติ นั่นคือ คลาส และโอเปอร์เรชั่นใหม่ๆ สามารถรวมเข้าไปในระบบ โดยไม่ส่งผลกระทบกับออบเจ็กต์อื่นๆ ภายในระบบ

3.4.2 Information Hiding

การซ่อนข้อมูล ออบเจ็กต์จะอนุญาตให้ข้อมูล และโอเปอร์เรชั่นถูกกำหนดอยู่ภายในโครงสร้างข้อมูลเดี่ยว (Single Model) นั่นคือ ข้อมูลจะถูกกำหนดเฉพาะที่ (Local) และถูกซ่อนอยู่ภายในออบเจ็กต์เอง เมื่อมีการเชื่อมต่อเกิดขึ้นจะมีการแสดงออกมาโดยอัตโนมัติ การทำโอเปอร์เรชั่นต่างๆ กับออบเจ็กต์จะกระทำโดยผ่านฟังก์ชันด้วยวิธี Message Passing

3.4.3 Flexibility of Type Definition

ความสามารถที่อนุญาตให้ผู้ใช้กำหนดคลาส และโอเปอร์เรชั่นใหม่ กล่าวคือ ให้ความยืดหยุ่น และการควบคุมให้กับผู้ใช้ ซึ่งเป็นประโยชน์กับการประยุกต์ใช้ฐานข้อมูลใหม่ๆ ในการปรับเปลี่ยนข้อมูล

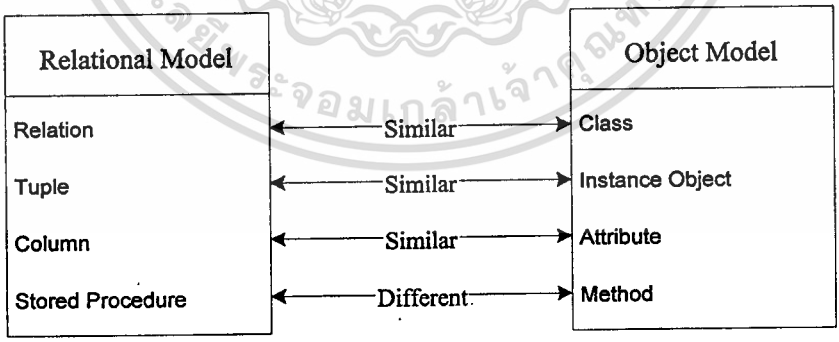
3.4.4 Code Reusability

สืบเนื่องมาจากคุณสมบัติการถ่ายทอดจากคลาสแม่ไปยังคลาสลูก ทำให้ไม่ต้องมีการกำหนดวิธีการ และโอเปอร์เรชั่นขึ้นใหม่ในแต่ละคลาส ซึ่งช่วยลดความซ้ำซ้อน และการทำงานที่ไม่ควบคู่กันได้

3.5 การเปรียบเทียบระหว่าง Object-Oriented และ Relational Data Model

ข้อเปรียบเทียบระหว่าง Object-Oriented และ Relational Data Model สามารถสรุปได้ 2 ทิศทางดังนี้ คือ

3.5.1 ความเหมือน และความแตกต่างทางด้านรูปแบบสามารถแสดงออกให้เห็นได้ดังรูปที่ 7



รูปที่ 7 แสดงการเปรียบเทียบระหว่างแบบจำลองเชิงวัตถุ และเชิงสัมพันธ์

3.5.2 ความเหมือน และความแตกต่างทางด้านแนวความคิด

3.5.2.1 Data Type

ลักษณะของชนิดข้อมูลที่ใช้กับระบบการจัดการฐานข้อมูลเชิงสัมพันธ์จะเป็นลักษณะที่เรียกว่า Built-In Data Type เช่น Int, Real, String เป็นต้น ซึ่งลักษณะของชนิดข้อมูลเหล่านี้ ระบบจะเป็นผู้ทำการกำหนดให้ แต่สำหรับระบบการจัดการฐานข้อมูลเชิงวัตถุนั้น ชนิดของข้อมูลที่สามารถใช้งานได้ จะเป็นทั้ง Built-In Data Type และชนิดของข้อมูลที่ผู้ใช้สามารถทำการกำหนดขึ้นมาเองได้ เช่น Complex Number เป็นต้น

3.5.2.2 Data Integrity

สำหรับระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ การทำงานลักษณะที่เรียกว่า Referential Integrity ซึ่งเป็นการสื่อความหมายในการอ้างถึงโดยผ่าน Foreign Key แต่ในทางตรงกันข้าม ระบบการจัดการฐานข้อมูลเชิงวัตถุ คลาสจะเป็นตัวบรรยายถึง Data Abstraction และรวมไปถึงการกำหนด Specific ให้กับโอเปอร์เรชั่นต่างๆ ที่จะถูกประยุกต์ใช้กับ Instances ของคลาส ซึ่งการกำหนดเช่นนี้จะทำให้ข้อมูลมีความเป็นอิสระต่อกันมากขึ้น กล่าวคือ เมื่อมีการเปลี่ยนแปลง หรือมีการพัฒนาระบบจะไม่ส่งผลกระทบต่อคลาสหรือกระบวนการทำงานอื่นๆ ซึ่งทำให้ข้อมูลมีความถูกต้อง (Data Integrity)

3.5.2.3 Data Manipulation

สำหรับภาษาที่ใช้ในการจัดการฐานข้อมูลเชิงสัมพันธ์จะใช้ SQL ซึ่งเป็นมาตรฐานเป็นภาษาที่มีประสิทธิภาพ และง่ายต่อการเข้าใจ ซึ่ง SQL จะมีคณิตศาสตร์มาสนับสนุนภาษานี้ ส่วนระบบฐานข้อมูลเชิงวัตถุนั้นยังไม่มีมาตรฐานที่แน่นอน ทำให้มีหลายองค์กรพยายามทำมาตรฐานขึ้น ในปัจจุบันได้มีการจัดทำมาตรฐานของ OO-DBMS ขึ้นมา เพื่อใช้เป็นมาตรฐานสำหรับระบบฐานข้อมูลเชิงวัตถุ โดยมีองค์กรที่ชื่อว่า Object Database Management Group (ODMG) ซึ่งเป็นองค์กรที่มีหน้าที่กำหนดมาตรฐานสำหรับระบบฐานข้อมูลเชิงวัตถุ โดย OO-DBMS ที่เป็นมาตรฐานเวอร์ชันแรกคือ ODMG-93 ซึ่งออกมาในเดือนกันยายน ปี ค.ศ. 1993 ซึ่งเป็นมาตรฐานครอบคลุมในเรื่อง Object Database โดยเฉพาะในด้าน Object-Oriented Model และ Object-Oriented Programming Languages

บทที่ 4

Object Identity

4.1 นิยามของ Object Identity

Object Identity เป็นคุณสมบัติของออบเจ็กต์ที่ใช้จำแนกแต่ละออบเจ็กต์ออกจากออบเจ็กต์ทั้งหมด โดยจะต้องมีคุณลักษณะพิเศษคือมีความเป็นหนึ่งเดียวเป็นสิ่งเดียว เพื่อใช้แสดงถึงหรืออ้างถึงออบเจ็กต์โดยต้องมีความเป็นอิสระต่อ State หรือ Behavior ของออบเจ็กต์นั้น

ความเป็นหนึ่งเดียวเป็นสิ่งเดียว (Uniqueness) ของออบเจ็กต์นี้จะใช้สิ่งหนึ่งที่เรียกว่า Object Identifier ซึ่ง Identifier นี้จะไม่ขึ้นกับค่าของแอตทริบิวต์ของออบเจ็กต์เลย นั่นคือออบเจ็กต์หนึ่งๆ สามารถถูกจำแนกแบ่งแยกออกได้จากออบเจ็กต์อื่นๆ โดยไม่ต้องใช้ Value หรือ Behavior ของออบเจ็กต์นั้นเลย ซึ่ง Object Identity จะถูกสร้างขึ้นโดยระบบและไม่สามารถถูกละเมิดได้โดยผู้ใช้

4.2 ความสำคัญของ Object Identity

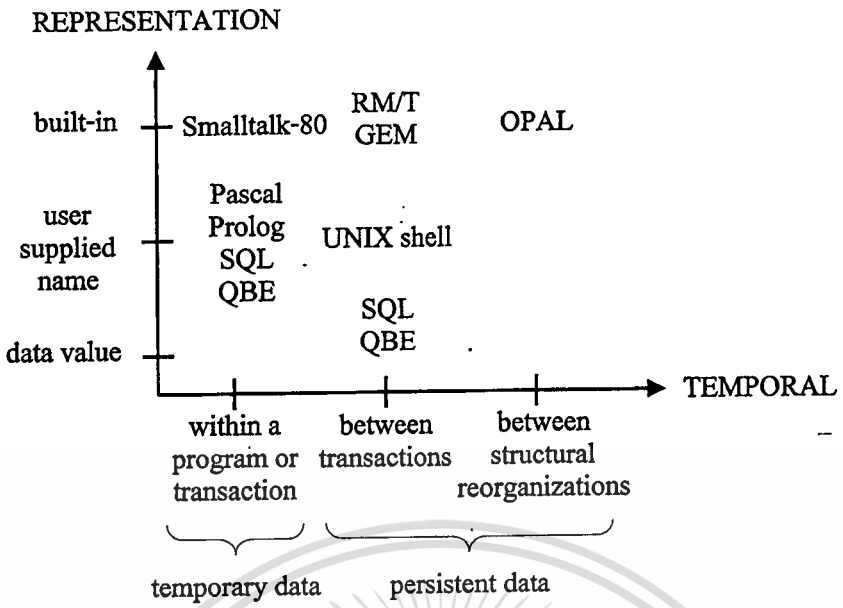
ภาษาโปรแกรมที่มีวัตถุประสงค์การใช้งานทั่วๆไปกับภาษาด้านฐานข้อมูลจะถูกแยกออกจากกันโดยอิสระในการพิจารณาถึง Identity ที่จะทำเกิดการพัฒนาใน 2 สภาวะแวดล้อม กล่าวคือ การวิวัฒนาการ และการรวมกัน

4.2.1 ระดับของการสนับสนุนของ Identity

ระดับของการสนับสนุนของ Identity มีอยู่อย่างน้อย 2 มิติได้แก่

- Representation Dimension
- Temporal Dimension

ดังรูปที่ 8 ซึ่งแสดง Identity Space กับตัวอย่างภาษาบางภาษาที่อยู่ใน Identity Space



รูปที่ 8 แสดงภาษาใน Identity Space

สำหรับ Represent Dimension จะแบ่งออกเป็น 3 ลักษณะขึ้นกับความแตกต่างโดยพื้นฐานของแต่ละภาษา ได้แก่

- Data Value เป็นการแสดง Identity ของออบเจ็กต์โดยใช้ค่าของข้อมูล อย่างเช่น ในการ Identify พนักงานโดยใช้หมายเลขประกันสังคม เป็นต้น
- User-defined Name ได้แก่พวกชื่อตัวแปร หรือชื่อไฟล์ที่ผู้ใช้กำหนดขึ้นเอง
- Built-In เป็นการแทนที่ติดมากับระบบภาษานั้นๆ อยู่แล้ว ภาษาประเภทนี้ได้แก่ Smaltalk-80

ในแต่ละภาษาต้องจัดหาแนวคิดที่แข็งแรงในเรื่อง Identity ซึ่งต้องดูแลรักษาการแทนความเป็นเอกลักษณ์ในระหว่างการ Update, การใช้ Identity ในแง่ของความหมายของโอเปอร์เรเตอร์ที่จะกระทำ และจัดหาโอเปอร์เรเตอร์ในการจัดการเกี่ยวกับ Identity เหล่านี้ด้วย

สำหรับ Temporal Dimension จะแบ่งออกเป็น 3 ลักษณะขึ้นกับความแตกต่างโดยพื้นฐานของแต่ละภาษาว่ามีลักษณะการจองหรือสงวนการแทนของ Identity อย่างไรซึ่งอาจจะสงวนไว้ภายในโปรแกรมหรือทรานแซคชั่นเดี่ยวๆ , ระหว่างทรานแซคชั่น หรือระหว่างการจัดการในเชิงโครงสร้างใหม่ (Structural Reorganizations) ในแต่ละภาษา Temporal Dimension ต้องเทคนิคในการ Implement ที่มีความคงทน ทนทาน (Robust) มากเพียงพอในการสงวนการแทนหรือการแสดงเอกลักษณ์ของมัน

4.2.2 Identity ในภาษาโปรแกรม

ภาษาโปรแกรมที่มีวัตถุประสงค์ในการใช้งานทั่วไปส่วนใหญ่จะถูกออบแบบมาโดยไม่ได้รวมแนวคิดของ Persistent Data ด้วยเหตุนี้ภาษาเหล่าจึงมีการสนับสนุนในเรื่องของ Identity แบบ Weak ใน Temporal Dimension กล่าวคือข้อมูลจะอยู่ในระหว่างการเอ็กซีคิวท์โปรแกรมเท่านั้น ระบบไฟล์ซึ่งไม่ได้เป็นส่วนหนึ่งของโปรแกรมถูกใช้สำหรับ Persistent Data อินพุตและเอาท์พุตโดยปกติจะถูกออกแบบมาเพื่อการโต้ตอบกับผู้ใช้ ทำให้เกิดโมเดลอินพุต และเอาท์พุตที่เหมือนกัน จากนั้นจึงถ่ายไปสู่ระบบไฟล์ โครงสร้างที่สนับสนุนในเนื้อที่ที่เป็น Virtual Address ของโปรแกรมโดยปกติจะไม่สนับสนุนในระบบไฟล์

ภาษาโปรแกรมและระบบไฟล์ส่วนใหญ่จะใช้ชื่อตัวแปรและชื่อไฟล์ในการแสดง Identity. จากรูปที่ 8 ภาษาปาสคาล และ Prolog จะใช้วิธีนี้ การใช้ชื่อตัวแปรโดยที่ไม่มีการแสดง Identity และโอเปอร์เรเตอร์ เพื่อทดสอบและจัดการกับการ Represent ในระดับนามธรรม ปัญหาหนึ่งที่ออบเจ็กต์เดี่ยวๆ อาจถูกเข้าถึงได้ในหลายทาง อาจผูกติดกับตัวแปรที่แตกต่างกัน ซึ่งตัวแปรเหล่านี้ไม่มีทางที่จะบรรลุได้ถ้าตัวแปรเหล่านั้นอ้างถึงออบเจ็กต์เดียวกัน ตัวอย่างเช่น ออบเจ็กต์ Employee อาจถูกใช้ในฐานะของรองผู้จัดการฝ่ายขายและผูกติดไว้กับตัวแปร X และออบเจ็กต์ Employee เดียวกันถูกใช้สำหรับพนักงานที่เงินเดือนมากกว่า 60,000 บาท และผูกติดไว้กับตัวแปร Y ภาษาเชิงวัตถุอย่างเช่น Smalltalk-80 จะจัดหาการทดสอบเอกลักษณ์ (Identity Test) อย่างง่ายให้ด้วย Expression X==Y ซึ่งต่างจากการทดสอบความเท่ากัน (Equality Test) X=Y เนื่องจากการทดสอบความเป็นเอกลักษณ์จะตรวจสอบว่าสองออบเจ็กต์เป็นออบเจ็กต์ที่เดียวกันเหมือนกันหรือไม่ ในขณะที่การทดสอบความเท่ากันจะตรวจสอบถึงเนื้อหา (Content) ของสองออบเจ็กต์ว่าเหมือนกันหรือไม่ ในระบบยูนิกซ์จะมี Built-in Representation สำหรับ File Identity เพื่อสนับสนุนการลิงค์ระหว่างไฟล์ แต่ไม่มีทางที่จะทดสอบได้โดยตรงเลยว่าสองไฟล์ที่มาถึงโดยใช้ทางเดินที่ต่างกันเหมือนกันหรือไม่

4.2.3 Identity ในภาษาด้านฐานข้อมูล

ภาษาด้านฐานข้อมูลจะถูกออกแบบมาให้สนับสนุนข้อมูลขนาดใหญ่ และ Persistent Data คุณลักษณะเช่นนี้ต้องการการสนับสนุนของ Identity ที่ Strong ทั้ง Represent และ Temporal Dimension

ในการสร้างแบบจำลองออบเจ็กต์จะเป็นการรวมของบางสับเซตของรายละเอียดของออบเจ็กต์เข้าไว้ในแบบจำลองเท่านั้น ซึ่งสับเซตเหล่านี้อาจจะไม่สมบูรณ์เพียงพอที่จะทำคลุมความเป็นหนึ่งเดียวของออบเจ็กต์ได้ ในบางกรณี Uniqueness เป็นแบบ External (ถ้ามีบางค่าของ

โลคอลแอตทริบิวต์และเป็นของต่างเซตหรือถูกเกี่ยวข้องกับออบเจ็กต์ที่ต่างกัน ปัญหานี้เกิดขึ้นในฐานข้อมูลในการที่จะพยายามสร้างแบบจำลองระบบ Real-world

ในปี 1970 Codd แนะนำแนวคิดของ User-defined Identifier Key เพื่อแทน Identity ของออบเจ็กต์ Identifier Key เป็นบางสับเซตของแอตทริบิวต์ของออบเจ็กต์ซึ่งทำให้ Unique สำหรับทุกๆ ออบเจ็กต์ใน Relation ซึ่งเป็นแนวคิดที่พบบ่อยในระบบฐานข้อมูลที่มีอยู่จากรูปที่ 8 SQL และ QBE เป็นภาษาที่ใช้ Identifier Key ในการ Identify ออบเจ็กต์ Persistent โดยที่ SQL ใช้ Tuple Variable และ QBE ใช้ Domain Variable ทำให้เกิดปัญหาหลายอย่างกับ Identifier Key ซึ่งเป็นแนวคิดที่ผสมระหว่างค่าข้อมูล กับ Identity เข้าไว้ด้วยกัน ปัญหาเหล่านี้ได้แก่

4.2.3.1 Identifier Key จะไม่อนุญาตให้เปลี่ยนแปลงได้แม้ว่าจะเป็นข้อมูลที่อธิบายโดยผู้ใช้กำหนดขึ้นเองก็ตาม เช่น ชื่อของแผนกอาจถูกใช้เป็น Identifier Key และใช้กับออบเจ็กต์ Employee เพื่อบ่งบอกว่าพนักงานคนใดอยู่แผนกอะไร แต่ชื่อแผนกอาจจำเป็นที่จะต้องเปลี่ยนภายใต้การจัดองค์กรใหม่ของบริษัท ซึ่งเป็นสาเหตุทำให้เกิดความไม่ต่อเนื่องใน Identity สำหรับแผนก ทำให้เกิดปัญหาการ Update กับทุกๆ ออบเจ็กต์ที่อ้างถึงแผนก

4.2.3.2 Identifier Key ไม่สามารถจัดหา Identity สำหรับทุกๆออบเจ็กต์ในแบบจำลองเชิงสัมพันธ์ แต่ละแอตทริบิวต์หรือ สับเซตของแอตทริบิวต์ที่มีความหมาย (Meaningful) ไม่สามารถมี Identity เช่น ออบเจ็กต์ Employee อาจมีแอตทริบิวต์ของคู่สมรสโดยระบุเป็นชื่อ ต่อมาคู่สมรสนั้นๆ อาจเป็นพนักงานได้ด้วยเช่นกัน ทำให้เกิดความไม่ต่อเนื่องใน Identity สำหรับคู่สมรส

4.2.3.3 ปัญหาที่สามนี้เป็นปัญหาของทางเลือกของการเลือกแอตทริบิวต์ที่จะใช้เป็น Identifier Key ซึ่งอาจจำเป็นต้องเปลี่ยนแปลง เช่น บริษัท A อาจใช้หมายเลขพนักงานในการ Identify พนักงาน ในขณะที่ บริษัท B อาจใช้หมายเลขประกันสังคมเพื่อจุดประสงค์เดียวกัน หากต้องมีการรวมกันของ 2 บริษัท ทำให้ต้องมีการเปลี่ยนแปลง Identifier ของบริษัทใดบริษัทหนึ่งอันเนื่องมาจากความไม่ต่อเนื่องกันใน Identity

4.2.3.4 การใช้ Identifier Key เพื่อทำการ join เช่น สมมติว่ามี ความสัมพันธ์ EMPLOYEE และ DEPARTMENT เป็นดังนี้

EMPLOYEE(NAME, SS#, BIRTHDATE, ASSIGNMENT)

DEPARTMENT(NAME, BUDGET, LOCATION)

แอตทริบิวต์ ASSIGNMENT จะสร้างความสัมพันธ์ต่อกันระหว่าง EMPLOYEE และ DEPARTMENT การดึง tuple ทั้งสองต้องมีการ join กันดังนี้

- Tuple Variable

SELECT E.name, D.location WHERE

E.assignment = D.name

and E IN employee and D IN department

เมื่อ E และ D เป็น Tuple Variable

- Domain Variable

[X, Z] <== employee[x, -, -, Y], department[Y, -, Z]

เมื่อ X, Y และ Z เป็น Domain Variable

4.3 โมเดลออบเจ็กต์

4.3.1 โครงสร้างของออบเจ็กต์ (Object Structure)

สมมติให้เซตของแอตทริบิวต์ชื่อ A และเซตของ Identifier เป็น I ออบเจ็กต์ O จะกำหนดเป็น triple(identifier, type, value) เมื่อ

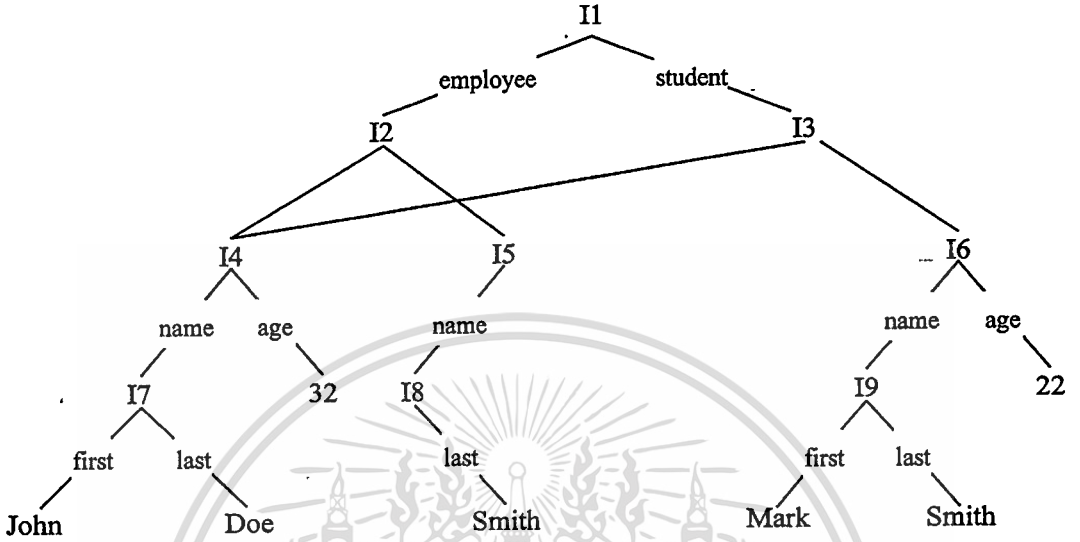
- Identifier อยู่ใน I โดยที่ Identifier ของออบเจ็กต์ O จะแทนด้วย O.identity
- ชนิดข้อมูล (Type) อยู่ใน {atom, set, tuple}
- ค่า (Value) จะเป็นไปตามข้อใดข้อหนึ่งดังต่อไปนี้
  - ถ้าออบเจ็กต์เป็นชนิด atom แล้ว จะได้ value เป็น element ของ User-defined Domain ของ atom แต่ละ element จะไม่มีส่วนย่อย
  - ถ้าออบเจ็กต์เป็นชนิด set แล้ว จะได้ value เป็น เซตของ Identifiers ที่แตกต่างกัน จาก I
  - ถ้าออบเจ็กต์เป็นชนิด tuple แล้ว จะได้ value อยู่ในรูป

    [A1:I1, A2:I2, …, An:In]

    เมื่อ Ai เป็นชื่อแอตทริบิวต์ที่ต่างกัน และ Ii เป็น Identifier ที่แตกต่างกันจาก I

ออบเจ็กต์สามารถถูกแทนในแบบกราฟฟิกได้โดยจะแทน Atomic Object ด้วยโหนดที่เลเบลโดยใช้ value ของออบเจ็กต์นั้น, tuple object O = [A1:O1, …, An:On] ด้วยโหนดที่เลเบลโดย identifier ของออบเจ็กต์ ซึ่งมี arc ที่ถูกเลเบลด้วย Ai เชื่อมจาก O ไปยัง Oi, และ เซต O = {O1, …, On} โดยโหนดที่เลเบลด้วย Identifier ของมัน ซึ่งจะมี arc ที่ไม่ได้เลเบลจาก O ไปยังทุกๆ Oi ตัวอย่างเช่น มีฐานข้อมูลที่ประกอบด้วย Employees และ Students ซึ่งแต่ละอันจะมี Name ซึ่ง

ประกอบด้วย first name, last name และ age instance ของฐานข้อมูลนี้สามารถถูกแทนได้ดังรูปที่ 9



รูปที่ 9 แสดงตัวอย่างออบเจ็กต์โมเดล

4.3.2 ระบบออบเจ็กต์ (Object System)

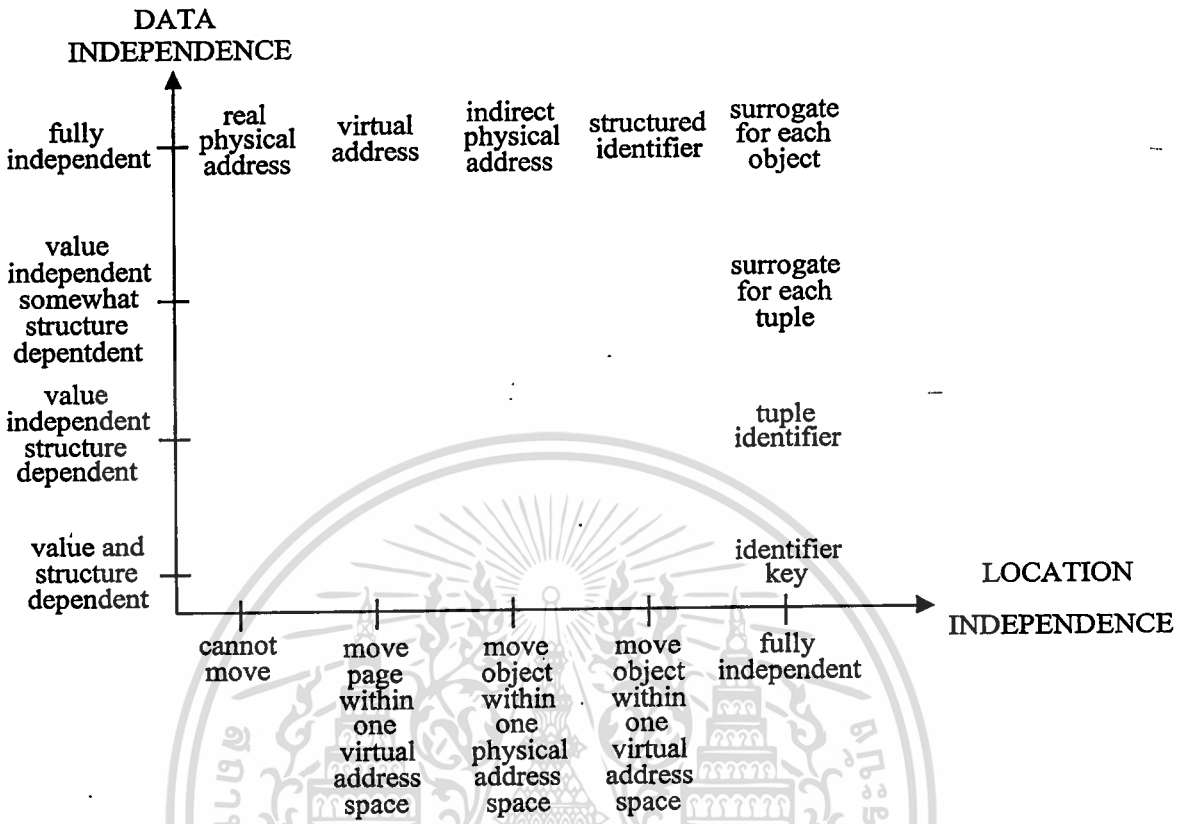
ระบบออบเจ็กต์ คือ เซตของออบเจ็กต์ ระบบออบเจ็กต์จะ Consistent ถ้า

1. Unique Identifier Assumption – ไม่มีสองออบเจ็กต์ใดๆ ที่ต่างกันที่มี Identifier เหมือนกัน
2. No Dangling Identifier Assumption – แต่ละ Identifier ที่มีอยู่ในระบบจะต้องมีออบเจ็กต์ที่สอดคล้องกับ Identifier นี้ด้วย

4.4 เทคนิคการ Implement

4.4.1 การแบ่งแยกการ Implement (Implementation Taxonomy)

การวัดแต่ละเทคนิคในการ Implement จะวัดโดยใช้ระดับของความเป็นอิสระของค่า (Value), โครงสร้าง (Structure) และที่ตั้ง (Location) ที่แต่ละเทคนิคจัดไว้ให้ ความเป็นอิสระของข้อมูลหมายถึงว่า Identity นั้นยังถูกสงวนหรือจองไว้ตลอดการเปลี่ยนแปลงค่าข้อมูลหรือโครงสร้าง และความเป็นอิสระของที่ตั้งจะหมายถึงว่า Identity นั้นยังถูกสงวนหรือจองไว้ตลอดการเคลื่อนย้ายของออบเจ็กต์ในระหว่างที่ตั้งทางกายภาพ (Physical Locations) หรือ Address Spaces ซึ่งทั้งความอิสระของข้อมูลและที่ตั้งมีความสำคัญเมื่อกำลัง Implement ออบเจ็กต์ด้วยแนวคิดเกี่ยวกับ Identity แบบ Strong ทั้งสองมิติ คือ Representation และ Temporal Dimension



รูปที่ 10 แสดง Implementation Taxonomy

จากรูปที่ 10 ซึ่งอธิบายเทคนิคการ Implement ใน 2 มิติ ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.4.1.1 Identity Through Physical Address

การ Implement ที่ง่ายที่สุดของ Identity ของออบเจ็กต์ คือแอดเดรสทางกายภาพของออบเจ็กต์ ซึ่งแอดเดรสทางกายภาพสามารถเป็นได้แอดเดรสจริงหรือแอดเดรสเสมือนของออบเจ็กต์ (ถ้าระบบออบเจ็กต์กำลังทำกับสภาวะแวดล้อมที่เป็นหน่วยความจำเสมือน) เช่น ในปาสคาล "Identity" ของเรคคอร์ด นั่นคือเป็นพอยต์เตอร์ที่ชี้ไปที่เรคคอร์ด จะถูก Implement ตลอดใน Virtual Heap Address การ Implement แอดเดรสทางกายภาพนี้ไม่อนุญาตให้ออบเจ็กต์ถูกย้าย ดังนั้นจึงขาดความเป็นอิสระทางด้านที่ตั้ง การ Implement แบบแอดเดรสเสมือนจะยอมให้เพียงเพจของออบเจ็กต์ทั้งหมดเท่านั้น ไม่ได้แยกแต่ละออบเจ็กต์เดี่ยวๆ ในการที่จะย้ายภายในหนึ่ง Virtual Address Space การทำเช่นนี้ทำให้มีความเป็นอิสระทางด้านที่ตั้งเล็กน้อย แต่อย่างไรก็ตาม เนื่องจากออบเจ็กต์ไม่สามารถย้ายได้ระหว่าง Address Space การใช้ออบเจ็กต์ร่วมกันในระหว่างหลายๆ โปรแกรม (Multiple Program) ก็ถูกจำกัดไปด้วย ทั้งการ Implement แบบแอดเดรสทาง

กายภาพจริงและเสมือนจะมีความเป็นอิสระของข้อมูล ถ้าการเปลี่ยนแปลงข้อมูลนั้นไม่เป็นผลให้ออบเจ็กต์นั้นถูกย้ายตำแหน่ง

4.4.1.2 Identity Through Indirection

สำหรับ Smalltalk-80 object-oriented pointer (oop) ถูกใช้ในการ implement Identity จะมีตารางออบเจ็กต์ที่เก็บ oop นี้ ซึ่งทำให้ Identity นี้ถูก Implement โดยทางอ้อม การ Implement แบบการเข้าถึงแอดเดรสทางกายภาพหรือเสมือนโดยอ้อมจะยอมให้แต่ละออบเจ็กต์ถูกย้ายภายในหนึ่ง address space ได้ เป็นการเพิ่มความแข็งแรงของความเป็นอิสระทางด้านที่ตั้งมากกว่าแบบการ Implement การเข้าถึงแอดเดรสโดยตรง แต่ไม่ยอมให้มีการใช้ออบเจ็กต์ร่วมกันในระหว่างโปรแกรมหลายๆ โปรแกรม การ implement แบบนี้จะมีความเป็นอิสระด้านข้อมูลเต็มที่

4.4.1.3 Identity Through Structured Identifier

ระบบกระจายบางระบบ Identifier ของไฟล์ (ออบเจ็กต์ของระบบ) ถูกจัดให้มีโครงสร้าง Structured Identifier จัดให้มีความเป็นอิสระของข้อมูลอย่างเต็มที่ ยอมให้แต่ละออบเจ็กต์ถูกย้ายภายในหนึ่งดิสก์หรือเซิร์ฟเวอร์ การย้ายในระหว่างหลายๆแอดเดรสสามารถกระทำได้ ดังนั้นออบเจ็กต์จึงสามารถถูกใช้ร่วนกันกับหลายๆ โปรแกรม การย้ายที่ออบเจ็กต์ทำให้มีการกระจายโหลดการทำงานในระบบกระจาย

4.4.1.4 Identity Through Identifier Keys

แนวทางหลักๆ สำหรับการสนับสนุน Identity ในระบบการจัดการฐานข้อมูล คือ การใช้ Identifier Key ซึ่งผู้ใช้เป็นผู้กำหนดได้โดยตรง tuple มีการเรียงลำดับตาม Identifier Key และมีโครงสร้างช่วย เช่น B-Tree ซึ่งถูกสร้างบนเซตของ tuple ที่จะทำให้การเข้าถึงออบเจ็กต์เร็วขึ้นโดยผ่าน Identifier Key การ Implement แบบใช้ Identifier Key จะจัดให้เกิดความเป็นอิสระด้านที่ตั้งอย่างเต็มที่ แต่ไม่ได้ทำให้เกิดความเป็นอิสระของค่าข้อมูล ไม่สนับสนุนความเป็นอิสระของโครงสร้างเพราะว่า Identifier Key จะ Unique เพียงภายในรีเลชั่นเดี่ยวๆเท่านั้น ไม่ลงไปถึงระดับแอตทริบิวต์

4.4.1.5 Identity Through Tuple Identifiers

ในบางระบบเช่น System R, INGRES และ WiSS Internal Tuple Identifiers ที่อยู่ในเลเยอร์ภายในเพื่อให้มีอินเตอร์เฟสที่ง่ายของ DBMS และเพื่อดูแลโมดูลในการควบคุม Concurrency/Recovery ของ DBMS ซึ่ง Tuple Identifier ไม่ได้สอดคล้องกันโดยตรงกับแนวคิดของ Identity แต่อย่างไรก็ตาม Tuple Identifier สามารถใช้ในการ Implement Identity ได้ โดยถูกสร้างจากระบบซึ่ง Unique สำหรับทุกๆ รีเลชั่นเดี่ยวๆ โดย

ที่ไม่มีความสัมพันธ์กับที่ตั้งทางกายภาพ ดังนั้น Tuple Identifier จะสนับสนุนความเป็นอิสระด้านที่ตั้งอย่างเต็มที่ นอกจากนี้ยังสนับสนุนความเป็นอิสระของค่า แต่ไม่อิสระต่อโครงสร้าง อันเนื่องมาจาก Tuple Identifier จะ Unique ภายในรีเลชั่นเดี่ยวๆ เท่านั้น

4.4.1.6 Identity Through Surrogates

Surrogates เป็นเทคนิคที่มีประสิทธิภาพที่สุดในการสนับสนุนเรื่องของ Identity ซึ่ง Surrogate เป็นตัวที่ระบบสร้างให้เป็น Globally Unique Identifier มีความเป็นอิสระต่อที่ตั้งทางกายภายอย่างสมบูรณ์

4.4.2 การ Implement แบบจำลองออบเจ็กต์ด้วย Surrogates

แต่ละออบเจ็กต์ของชนิดข้อมูลใดๆก็ตามจะมีความสัมพันธ์กับ Globally Unique Surrogate ซึ่ง Surrogate ถูกใช้เพื่อแทน Identity ของออบเจ็กต์เป็นการภายในตลอดช่วงชีวิตของออบเจ็กต์

มีหลายๆ เหตุผลที่บอกว่าทำไมลักษณะทางกายภาพของแนวคิดของออบเจ็กต์ไม่ได้ถูกเก็บในตำแหน่งเดียว เหตุผลหนึ่งคือบางส่วนของออบเจ็กต์อาจถูกใช้ร่วมกันกับออบเจ็กต์อื่น อันเนื่องมาจากกราฟโครงสร้างของแบบจำลองออบเจ็กต์ ออบเจ็กต์ที่อ้างอิงโดยหลายๆ ออบเจ็กต์โดยทางกายภาพแล้วไม่สามารถถูกเก็บเข้าด้วยกันกับแต่ละออบเจ็กต์ที่เป็นออบเจ็กต์อ้างอิงของมันได้ เหตุผลข้อที่สองคือการถอดแบบที่ควบคุมได้อาจถูกใช้เพื่อประโยชน์ในการทำการกู้คืนข้อมูล (Data Recovery) ทางกายภาพแล้ว การถอดแบบลักษณะนี้ต้องถูกเก็บบนสื่อที่แยกกันเพื่อความสามารถในการกู้คืนจะได้มีประสิทธิภาพ เหตุผลข้อที่สามคือแต่ละส่วนของออบเจ็กต์อาจถูกแบ่งแยกทั้งนี้ขึ้นกับความถี่ของการใช้เพื่อเสริมประสิทธิภาพให้กับข้อมูลที่อยู่ในดิสก์ เหตุผลข้อที่สี่เกี่ยวกับการสนับสนุนของแบบจำลอง Temporal Data ซึ่งเวอร์ชันปัจจุบันอาจถูกเก็บแยกกันจากเวอร์ชันก่อนๆ ของออบเจ็กต์ ซึ่งจะทำให้ความเร็วในการเข้าถึงข้อมูลปัจจุบันไม่ถูกลดลง จากทั้งหมดที่ได้กล่าวมา Surrogate ของออบเจ็กต์จะจัดหาวิธีที่เหมาะสม เพื่อให้สัมพันธ์กับการเก็บส่วนของออบเจ็กต์ในหลายที่แยกกัน

การทดสอบสำหรับ Identity ได้แก่ O1.identity = O2.identity นั้นจะใช้การทดสอบความเท่ากันของ surrogate นั่นคือ O1.surrogate = O2.surrogate

โอเปอร์เรเตอร์การลบ (Remove-element Operator) ต้องมีการตรวจสอบสำหรับ Dangling Identifier เพื่อให้แน่ใจว่าเกิดความตรงกันของระบบออบเจ็กต์ นั่นคือไม่ควรยังมีการอ้างถึงใดๆ ไปยังออบเจ็กต์ที่ไม่มีตัวตนอีก สิ่งนี้สามารถ Implement โดยการค้นหา Dangling Surrogate แต่ละครั้งเมื่อมีการทำการ update ซึ่งเทคนิคนี้เป็นเทคนิคที่คล้ายกับที่ใช้ในฐานข้อมูลเชิงสัมพันธ์ในเรื่องของ Referential Integrity Enforcement

โอเปอร์เรเตอร์การให้ค่า (Assign Operator) อาจมีการอ้างถึงออบเจ็กต์ที่ไม่ปรากฏในระบบ ซึ่งไม่สามารถเข้าถึงออบเจ็กต์ตัวนั้นได้ และเนื้อที่ที่เก็บออบเจ็กต์นั้นก็ไม่ได้ถูกเก็บอีกแล้ว การ Implement ทำได้โดยการเก็บตัวนับการอ้างถึงสำหรับแต่ละออบเจ็กต์ ได้แก่จำนวนของการอ้างถึงยังออบเจ็กต์ ซึ่งถูก Update แต่ละครั้งเมื่อมีการเพิ่มหรือลบออบเจ็กต์ เมื่อตัวนับการอ้างถึงของออบเจ็กต์มีค่าเป็นศูนย์แสดงว่ากำลังอ้างไปถึงพื้นที่เป็นขยะ (Garbage)

โอเปอร์เรเตอร์การรวม (Merge Operator) เป็นการรวมสองออบเจ็กต์ให้กลายเป็นหนึ่งออบเจ็กต์ การ Implement ทำได้โดยการดอยบำรุงดูแลความสัมพันธ์สมดุล (Equivalence Relationship) ระหว่างสอง Surrogate

บทที่ 5

ปัญหาของการใช้ Identifier Key ในการแสดง Identity ของออบเจ็กต์

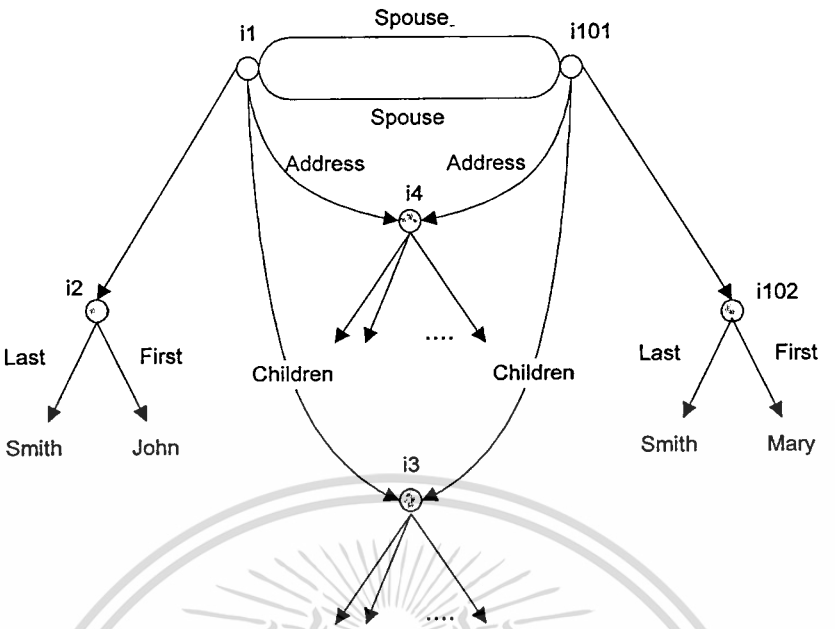
5.1 Modifying Identifier Keys

Identifier key ไม่สามารถ (หรือไม่ควร) ให้มีการเปลี่ยนแปลงแม้ว่า Identifier key จะเป็นข้อมูลอธิบายที่เป็น User-defined ก็ตาม เช่น ถ้าใช้ชื่อของผู้จัดการฝ่ายขายเป็น Identifier key และอาจมีการ Replicate ไปยังออบเจ็กต์พนักงานขายเพื่อบอกว่าพนักงานขายนั้นทำงานกับใครมีใครเป็นผู้จัดการ แต่ชื่อของผู้จัดการฝ่ายขายนั้นอาจมีความจำเป็นต้องมีการเปลี่ยนแปลง เช่นเปลี่ยนนามสกุลเมื่อแต่งงานไป ทำให้เกิดความไม่ต่อเนื่องกันในเอกลักษณ์ (Discontinuity in identity) สำหรับออบเจ็กต์ผู้จัดการฝ่ายขาย

การเปลี่ยน Content ทำให้ Identifier เปลี่ยน ในการออกแบบฐานข้อมูลใน Relational มักจะใช้ Identifier ในลักษณะ semantically meaningful เพื่อแสดง Identity ของออบเจ็กต์นั้นๆ ดังนั้นเมื่อมีการเปลี่ยนแปลงตัว Identifier นี้จึงทำให้ต้องมีการใช้ Referential Integrity เพื่อดูแล Consistency เอนติตี้อื่นๆที่อ้างถึงออบเจ็กต์นี้ซึ่งได้ถูกเปลี่ยน Identity ไปแล้ว ทำให้การอ้างแบบเดิมจะ invalid และต้องมีการแก้ไขให้ถูกต้อง

เมื่อมีการเปลี่ยนแปลง Content (Identifier เปลี่ยน) ของ Key ที่แสดง Identity ของออบเจ็กต์ใน Relational นั้นต้องมีกฎควบคุมความถูกต้องของข้อมูลมารองรับปัญหานี้ที่เรียกว่า Referential Integrity Rule ซึ่งใน RDBMS ในเกือบทุก product ในปัจจุบันมีพร้อมและสามารถรองรับปัญหานี้ได้แล้ว แต่ในบางครั้งก็อาจจะยากในการดูแลและจัดการกับข้อมูลเพื่อให้เกิด Consistency

ยกตัวอย่าง สมมติมีพนักงานชื่อ Mary Smith แต่งงานกับ John Smith และอยู่บ้านเดียวกัน นั่นคือมี Address เหมือนกัน หากออบเจ็กต์ที่อยู่นี้มิได้เป็นระบุเป็นคีย์นอกที่เชื่อมไปยังคีย์หลักของอีกตารางหนึ่งใน RDBMS แล้ว การดูแลความถูกต้องของแอตทริบิวต์นี้ทำได้ยากคือต้องใช้โพรซีเยอร์ แต่สำหรับการใช้ Object Identity แล้วสามารถดูแลปัญหานี้ได้อย่างง่าย เนื่องจาก Address จะมี Object Identifier ซึ่งทั้ง John และ Mary จะชี้ไปที่ Address เดียวกัน ดังรูปที่ 11



รูปที่ 11 แสดงแผนภาพการอ้างถึงออบเจ็กต์โดยใช้ Object Identifier

และสามารถเขียนจากแผนภาพรูปที่ 11 มาเป็นออบเจ็กต์ได้ดังรูปที่ 12

Object	i1
name	i2
children	i3
address	i4
spouse	i101

Object	i101
name	i102
children	i3
address	i4
spouse	i1

Object	i4
street	1212
StName	Main
state	CA
Zip	94596

Object	i102
last	Smith
First	Mary

Object	i2
last	Smith
First	John

รูปที่ 12 แสดงออบเจ็กต์ที่ได้จากแผนภาพรูปที่ 11

จะเห็นได้ว่าแม้ว่าจะแก้ออบเจ็กต์ที่อยู่อย่างไร ก็ไม่กระทบต่อออบเจ็กต์อื่นๆ หรืออาจกล่าวได้ว่า OID ทำให้ไม่ต้อง maintain referential integrity

5.2 Nonuniformity (ปัญหาการเลือกแอตทริบิวต์ที่จะใช้เป็น Key)

ปัญหานี้เป็นปัญหาของ Identifier Key ในต่าง table ที่มีชนิดข้อมูลต่างกัน (Types: Integer or Character-String Floating Point) หรือ Combination ของแอตทริบิวต์มีความแตกต่างกัน

ตัวอย่างเช่น บริษัท A อาจใช้หมายเลขพนักงานในการ Identify พนักงาน ในขณะที่ บริษัท B อาจใช้หมายเลขประกันสังคมเพื่อจุดประสงค์เดียวกัน หากต้องมีการรวมกันของ 2 บริษัท ทำให้

ต้องมีการเปลี่ยนแปลง Identifier ของบริษัทใดบริษัทหนึ่งอันเนื่องมาจากความไม่ต่อเนื่องกันใน Identity

การที่มีบางออบเจ็กต์ไม่สามารถแสดง Identity ออกมาได้ กล่าวคือ Identifier Key ไม่สามารถจัดหา identity สำหรับทุกๆออบเจ็กต์ในแบบจำลองเชิงสัมพันธ์ แต่ละแอตทริบิวต์หรือสับเซตของแอตทริบิวต์ที่มีความหมาย (meaningful) ไม่สามารถมี Identity เช่น ออบเจ็กต์ EMPLOYEE อาจมีแอตทริบิวต์ของคู่สมรสโดยระบุเป็นชื่อ ต่อมาคู่สมรสนั้นๆ อาจเป็นพนักงานได้ด้วยเช่นกัน ทำให้เกิดความไม่ต่อเนื่องใน Identity สำหรับคู่สมรส

สำหรับ OID แล้วจะมีเพียงรูปแบบเดียวซึ่งจะเหมือนกันในทุกๆออบเจ็กต์กล่าวคือถ้าเป็น Relational แล้ว Identifier ของแต่ละตารางจะไม่เป็นรูปแบบเดียวกัน บางตารางอาจใช้ Meaningful Digit บางตารางอาจใช้ Integer บางตารางอาจเป็น Character ก็ได้ ทำให้ขาด Uniformity ไป

5.3 "Unnatural" joins ทำให้สูญเสีย semantic

การดึงข้อมูลเมื่อใช้ Identifier Key ต้องมีการ Join แทนที่จะสามารถดึงออบเจ็กต์ได้ตรงๆ ซึ่งง่ายกว่า ตัวอย่างเช่น

Employee relation

OfficeWorker(Name, Age, Address, Salary, DeptName)

Department relation

Department(Name, Budget, Location, …)

คำสั่ง SQL ในการดึงชื่อพนักงานทุกคนและ location ที่พนักงานทำงาน

```
SELECT OfficeWorker.Name, Department.Location
FROM OfficeWorker, Department
WHERE OfficeWorker.DeptName = Department.Name
```

สิ่งนี้เป็น Unnatural กล่าวคืออะไรที่ผู้ใช้ต้องการทราบจริงๆ คือ Actual Department Tuple ไม่ใช่ DeptName กล่าวได้ว่าใน Relational System การนอร์มอลไลซ์เซชั่นจะถูกจำกัดชนิดของข้อมูลที่จะอยู่ในตารางและไม่อนุญาตให้กำหนดและจัดการทูเปิล, รีเลชั่น หรือ Complex Object Type อื่นๆ ของแอตทริบิวต์ นั่นคือการทำนอร์มอลไลซ์เซชั่นทำให้สูญเสีย Semantic ที่เชื่อมในระหว่างออบเจ็กต์ในฐานข้อมูล

ใน Relational Language เช่น SQL จะรวมกันกับความสามารถบางอย่างเพิ่มเติมเช่น Foreign Key Constraint เพื่อรักษา Semantic ที่หายไป

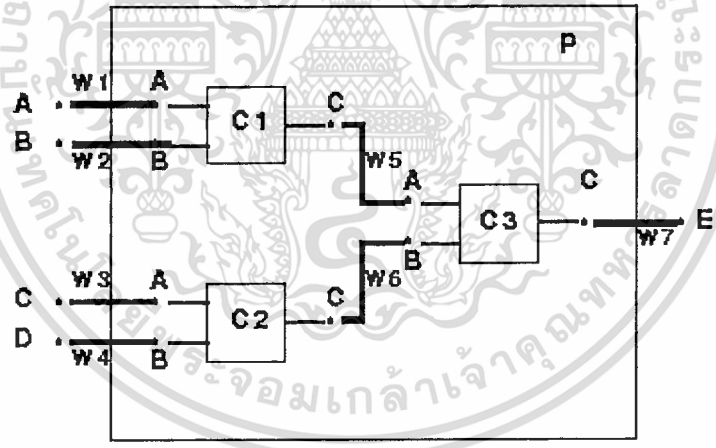
ถ้าจะไม่ให้สูญเสีย Semantic ใน Relational ต้องยอมรับภาระความซ้ำซ้อนกล่าวคือให้รวมทุกๆแอตทริบิวต์ไว้ในตารางเดียวกัน ซึ่งจะทำให้การ Maintain ลำบากและมีโอกาสที่จะทำให้ข้อมูลนั้น Inconsistency ได้

สำหรับ Relational Model ถ้าจะสร้าง Complex Object จะต้องสร้างจากรีเลชั่นมากกว่าหนึ่งรีเลชั่นมา Join กันเพื่อให้เกิด Complex Object ขึ้นมาทำให้เกิดความยุ่งยากและมีความซับซ้อน

นิยาม Complex Object

- เป็นออบเจ็กต์ที่มีคุณสมบัติ large, persistent และ structured โดยที่ "large" หมายถึงขนาดใหญ่มากไม่สามารถ fit ใน Conventional Main Memory ได้, "persistent" หมายถึง ว่าออบเจ็กต์จะคงอยู่จากช่วงเวลาหนึ่งไปยังอีกช่วงเวลาหนึ่ง และ "structured" หมายถึงว่าออบเจ็กต์ถูกสร้างจากหนึ่งหรือหลายๆออบเจ็กต์
- เป็นออบเจ็กต์ที่สร้างขึ้นจาก Atomic Object (เช่น จำนวนเต็ม, จำนวนจริง, บูลีน, สตริง, บิต เป็นต้น) โดยการใช้ Recursive Object Constructor เช่น tuple, set, bags, lists และ arrays

ตัวอย่างในการสร้าง Complex Object โดยใช้ RDBMS



รูปที่ 13 แสดง Complex Object

จากรูปแสดงให้เห็นถึง 4-input AND gate ซึ่งสร้างขึ้นจาก 2-input AND gate สามอัน กับ Relation ซึ่งไม่ได้แสดงถึงรายละเอียดของ gate หรือโครงสร้างภายในของ gate ในฐานข้อมูล

แนวทางในการจัดเก็บรายละเอียดของออบเจ็กต์ดังกล่าวใน RDBMS มี 2 แนวทางหลักๆ ซึ่งวิธีการแรกคือการ Decompose ออบเจ็กต์(อย่างสมบูรณ์) ให้เก็บในอยู่ในรูปของ Tuples หรือ Row โดยที่โครงสร้างที่สมบูรณ์ของออบเจ็กต์สามารถแสดงและ Manipulate ได้โดยตรงด้วย DBMS วิธีการดังกล่าวสามารถแสดงได้ดังนี้

Gate Type	GT	description
	2AND	"C = A&B"
	4AND	"E = A&B&C&D"

Pin Type	GT	PT	I/O
	2AND	A	I
	2AND	B	I
	2AND	C	O
	4AND	A	I
	4AND	B	I
	4AND	C	I
	4AND	D	I
	4AND	E	O

Gate Instance	GT	GI	Parent
	2AND	C1	4AND
	2AND	C2	4AND
	2AND	C3	4AND
	4AND	P	.....

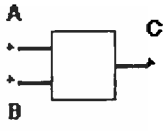
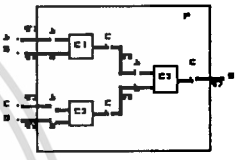
Wire Instance	WI	GT1	GI1	Pin1	GT2	GI2	Pin2	Parent
	W1	4AND	P	A	2AND	C1	A	4AND
	W2	4AND	P	B	2AND	C1	B	4AND
	W3	4AND	P	C	2AND	C2	A	4AND
	W4	4AND	P	D	2AND	C2	B	4AND
	W5	2AND	C1	C	2AND	C3	A	4AND
	W6	2AND	C2	C	2AND	C3	B	4AND
	W7	2AND	C3	C	4AND	P	E	4AND

รูปที่ 14 แสดงการแทน Complex Object ในรูปแบบของตารางในฐานข้อมูลเชิงสัมพันธ์

ในกรณีดังกล่าวนั้นใน Relation ของ Gate Type และ Pin Type ได้อธิบายเกี่ยวกับ Type ทั้ง 2 ชนิดของ gate และ pin ในแต่ละ Gate type ตามลำดับ สำหรับใน Relation ของ Gate Instance นั้นชี้ให้เห็นถึง Instance แต่ละอันของ 4-input AND gate คือการสร้างขึ้นจาก Instance 2-input AND gate 3 instances และใน Relation ของ Wire Instance ได้แสดงให้เห็นว่า 4-input AND gate นั้นประกอบด้วย wire 7 wire (ลวด) ซึ่งแต่ละอันจะเชื่อมด้วย pin 1 คู่ ตัวอย่างเช่น wire W1 เชื่อมต่อกับ Input pin A (จากภายนอก) ของ 4-input AND gate ไปยัง pin A ของ Component แรกที่เป็น 2-input AND gate เป็นต้น

จากตัวอย่างดังกล่าวนั้นจะพบว่ามีความยุ่งยากในการออกแบบให้สมบูรณ์เพื่อที่จะแสดงโครงสร้างของ Complex Object ใน RDBMS ซึ่งจะต้องแยกส่วนต่างๆ ให้อยู่ในรูปแบบของ Tuple ที่มีความสัมพันธ์กันมากมาย(จากตัวอย่างเป็นเพียง Module ง่ายๆ) นอกจากนี้แล้วการ Manipulate กับออบเจ็กต์ที่แสดงในรูปแบบดังกล่าวค่อนข้างมีความซับซ้อนและยุ่งยากมาก

วิธีการอีกวิธีหนึ่งที่ใช้กัน คือการจัดเก็บออบเจ็กต์ทั้งหมดนั้นให้อยู่ในรูปแบบที่เรียกกันว่า BLOB [1] โดยจะเก็บออบเจ็กต์ในรูปของ Binary ทั้งหมด ซึ่ง BLOB นั้นก็เป็นก็เป็น Column หนึ่งใน Relation โดยจะมีการใช้ร่วมกับข้อมูลใน Column อื่นๆ ที่เป็นส่วนอธิบายเสมอ เช่น part number หรือ object type ที่ใช้ในการ Identify ข้อมูลที่เก็บในรูปแบบ BLOB ดังกล่าว ซึ่งวิธีการดังกล่าวแสดงได้ดังนี้

Gate Type	GT	description	layout
	2AND	"C = A&B"	
	4AND	"E = A&B&C&D"	

Relational Representation of a Complex Object Using a BLOB

รูปที่ 15 แสดงการเก็บ Complex Object ใน Relational โดยการใช้ BLO B

จากรูปดังกล่าวใน Relation ของ Gate Type นั้นพบว่าค่าใน Column "layout" จะแสดงให้เห็นในลักษณะ Graphic โดยใน Column ดังกล่าวนี้จะเก็บข้อมูลในลักษณะ BLOB

วิธีการนี้ค่อนข้างยอมรับได้อย่างสมบูรณ์ ถ้ามีความต้องการแค่เพียงแสดง Layout ของข้อมูลนั้นๆ เท่านั้น อย่างไรก็ตามวิธีการนี้ DBMS ไม่สามารถถูกใช้เพื่อทำการ Manipulation ที่เกี่ยวกับโครงสร้างของออบเจ็กต์นั้นๆ ได้ตัวอย่างเช่นเราไม่สามารถที่จะ Identified ถึงตัวที่เป็น Subcomponent ของออบเจ็กต์นั้นได้ หรือไม่สามารถที่จะดึงข้อมูลในส่วนที่เป็น Subcomponent ได้เลยเนื่องจาก BLOB มีลักษณะการจัดเก็บเป็น Binary ทั้งหมด

---

[1] *BLOBS* (Binary Large Objects) เป็นกรณีพิเศษใน RDBMS ที่สนับสนุนให้มีการเก็บ (Store) ลงในฐานข้อมูล และสนับสนุนให้มีการ Retrieve ในส่วนของ Data Type ที่เพิ่มขึ้นมา โดยเฉพาะกับข้อมูลชนิด Multimedia (เช่น Text, Images, Audio clips, Videostreams เป็นต้น) แต่มีข้อจำกัดเนื่องจากไม่สามารถใช้ในการ Query ในส่วน Content ของ BLOB ได้ ตัวอย่างเช่น เราสามารถทำได้เพียงเก็บและดึง Image ได้เท่านั้น แต่ไม่สามารถ Query ฐานข้อมูลสำหรับ Image ที่ match กับสีดังกล่าว เป็นต้น

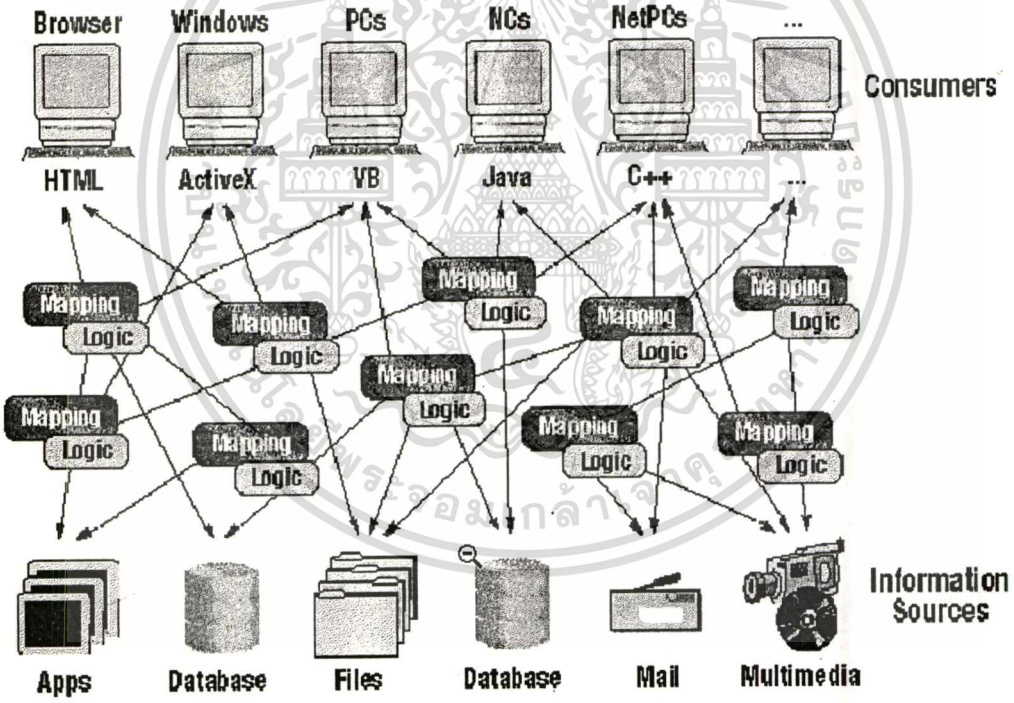
สำหรับ Object-Oriented Model แล้ว Complex Object มักเกิดจากการรวมกันของ Object Constructors เช่น Tuples, Collection Objects ซึ่งได้แก่ อาร์เรย์หรือเซต, และ Object Identity ที่จะอ้างไปยังออบเจ็กต์ที่ใช้ร่วมกัน (Referentially Share Object)

บทที่ 6

แนะนำ Jasmine

6.1 บทนำ

ในปัจจุบัน มีหลายๆ บริษัทที่ต้องทำงานและเกี่ยวข้องกับข้อมูลที่เป็นขนาดใหญ่ทั้ง Text, รูปภาพ, วีดีโอ, เสียง หรือภาพเคลื่อนไหวต่างๆซึ่งงานเหล่านี้ไม่สามารถรองรับได้ด้วยระบบฐานข้อมูลแบบเดิมที่มีใช้กันอยู่ ดังรูปที่ 16



รูปที่ 16 แสดง Today's Enterprise

ในขณะเดียวกันก็ต้องการแอปพลิเคชั่นที่มีคุณสมบัติทางด้านเว็บและระบบไคลเอนต์เซิร์ฟเวอร์ด้วย งานต่างๆเหล่านี้สามารถทำได้โดยการใช้ Jasmine ซึ่งเป็นระบบฐานข้อมูลเชิงวัตถุที่พร้อมไปด้วยคุณสมบัติทางด้านมัลติมีเดีย, Internet/Intranet System เพื่อสร้างแอปพลิเคชั่นที่มีประสิทธิภาพและใช้ค่าใช้จ่ายต่ำ

6.2 คุณสมบัติเด่น

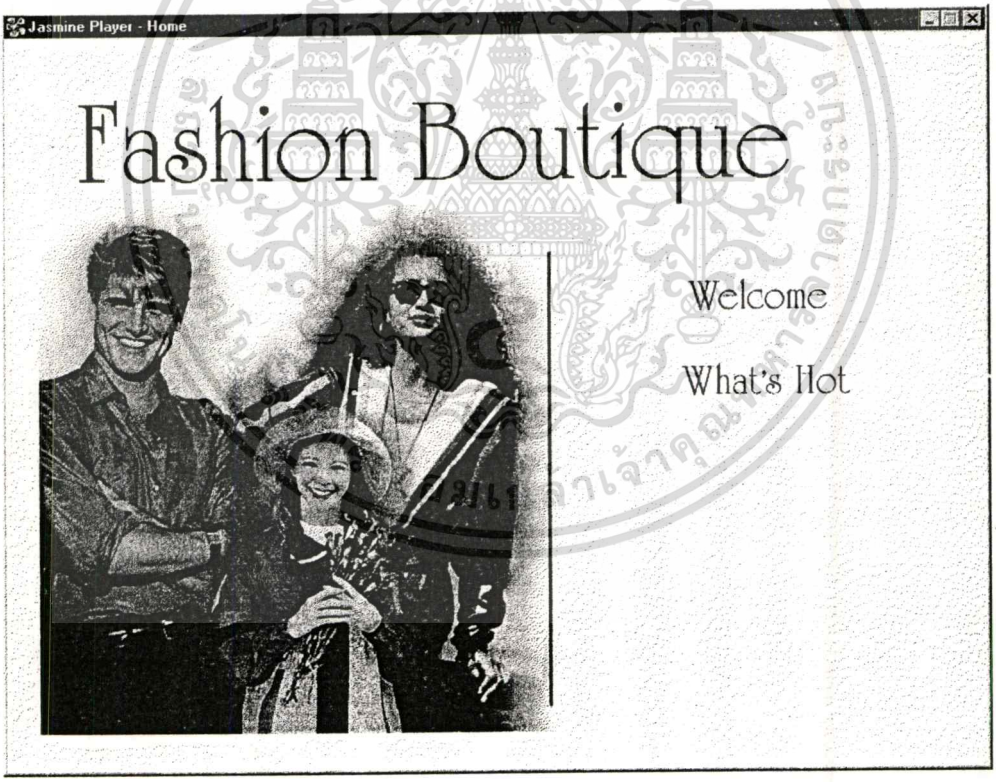
6.2.1 Object-Oriented

Jasmine สนับสนุนแนวคิดเชิงวัตถุ สามารถนำออบเจ็กต์มาใช้ใหม่ได้ทำให้สามารถสร้างแอปพลิเคชั่นได้รวดเร็วและซับซ้อนได้มากขึ้น ใช้เวลาในการพัฒนาไม่มาก

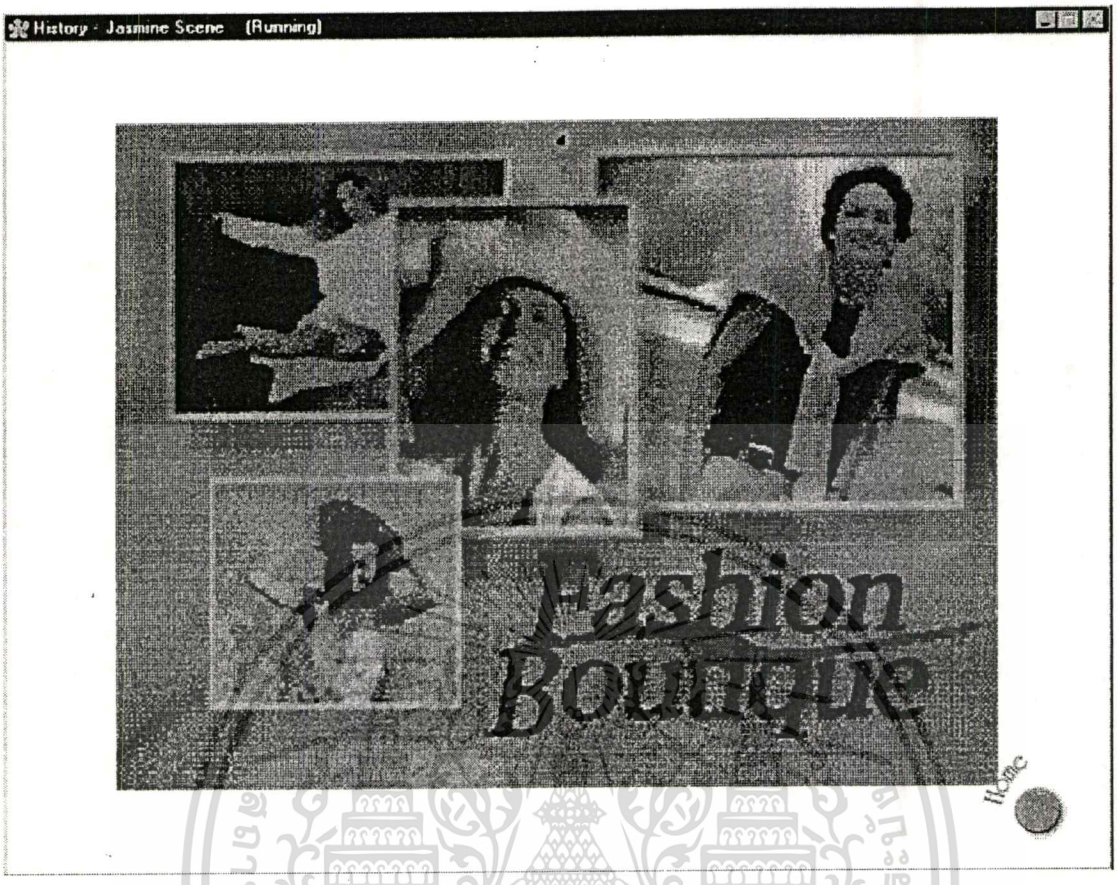
6.2.2 Easy to Use

ในการพัฒนาแอปพลิเคชั่นด้วย Jasmine Studio ซึ่งเป็นระบบการพัฒนาแอปพลิเคชั่นของ Jasmine จะช่วยให้สามารถสร้างหรือประกอบแอปพลิเคชั่นได้ง่ายไม่ต้องทราบถึงรายละเอียดของการโปรแกรมมากนัก Jasmine Studio จัดหาทูลที่ช่วยในการดีบักโปรแกรมและทูลที่ใช้จัดการกับข้อมูลในฐานข้อมูล

Jasmine Studio สามารถพัฒนาโดยเฉพาะแอปพลิเคชั่นด้านมัลติมีเดียได้ง่าย ตัวอย่างดังรูปที่ 17, 18



รูปที่ 17 แสดงตัวอย่างแอปพลิเคชั่นที่พัฒนาโดย Jasmine Studio



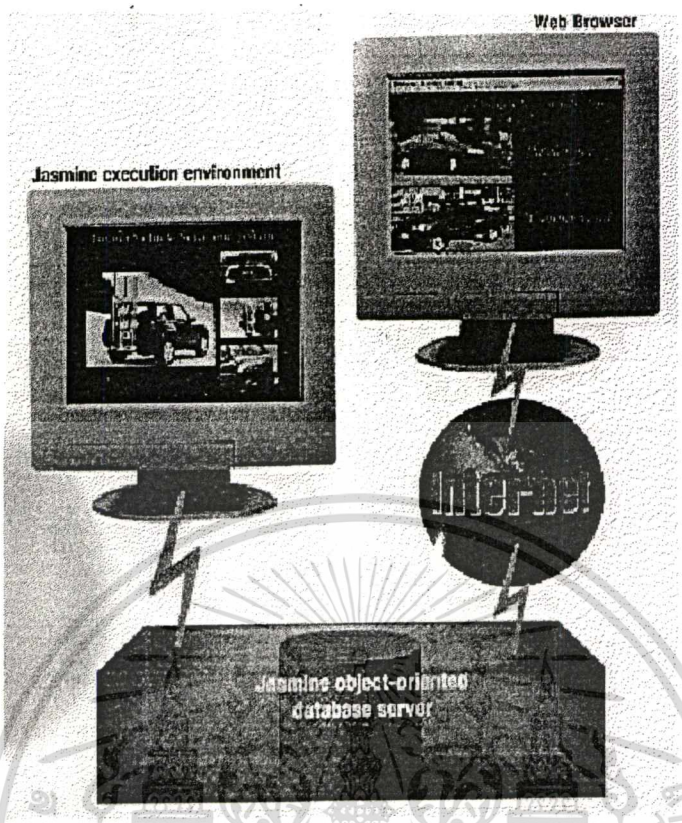
รูปที่ 18 แสดงตัวอย่างของภาพเคลื่อนไหวที่พัฒนาโดยใช้ Jasmine Studio

6.2.3 Multimedia Enabled

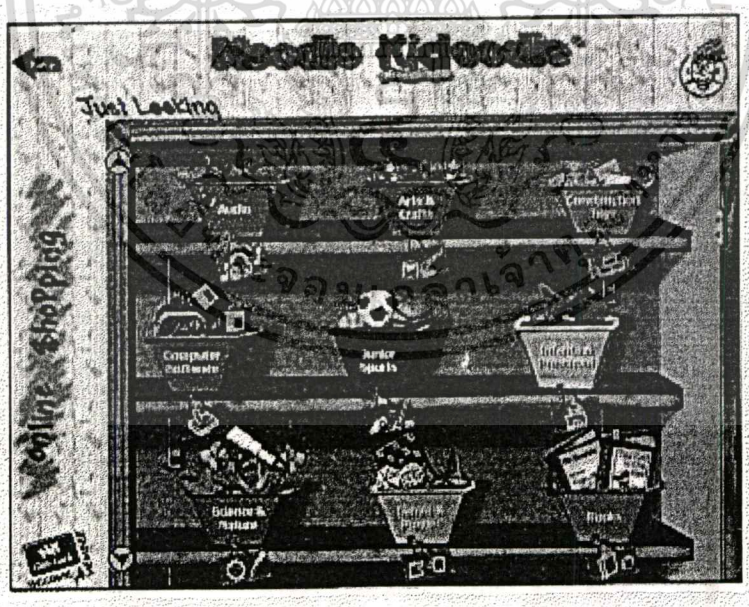
ข้อมูลหรือสารสนเทศที่เป็นลักษณะมัลติมีเดียเป็นแรงผลักดันพื้นฐานที่จะทำให้เกิดการแข่งขันรวมถึงโอกาสทางด้านธุรกิจมากขึ้น ในรูปแบบของกราฟ, รูปภาพ, วีดีโอ และเสียง ทำให้ผู้ใช้เข้าใจได้ง่าย หลายๆบริษัทมองไปถึงธุรกิจใหม่ๆที่จะเกิดขึ้นในอนาคตผ่านทางเครือข่ายอินเตอร์เน็ตและอินทราเน็ต รวมถึงการก้าวเข้าสู่ Electronic Commerce

6.2.4 Internet/Intranet/Extranet Enabled

Jasmine สามารถทำลงเว็บได้อย่างง่าย



รูปที่ 19 แสดงภาพการติดต่อกับ Jasmine Database Server



รูปที่ 20 แสดง Electronic Commerce Application ของ Noodle Kidoodle

6.2.5 Complete and Open

Jasmine สามารถทำงานร่วมกันได้หลายๆ ไคลเอนต์แอปพลิเคชั่นทูลและภาษา กล่าวคือ Jasmine สนับสนุน

- Java Bindings
- ActiveX Control
- Jasmine C API
- HTML Access

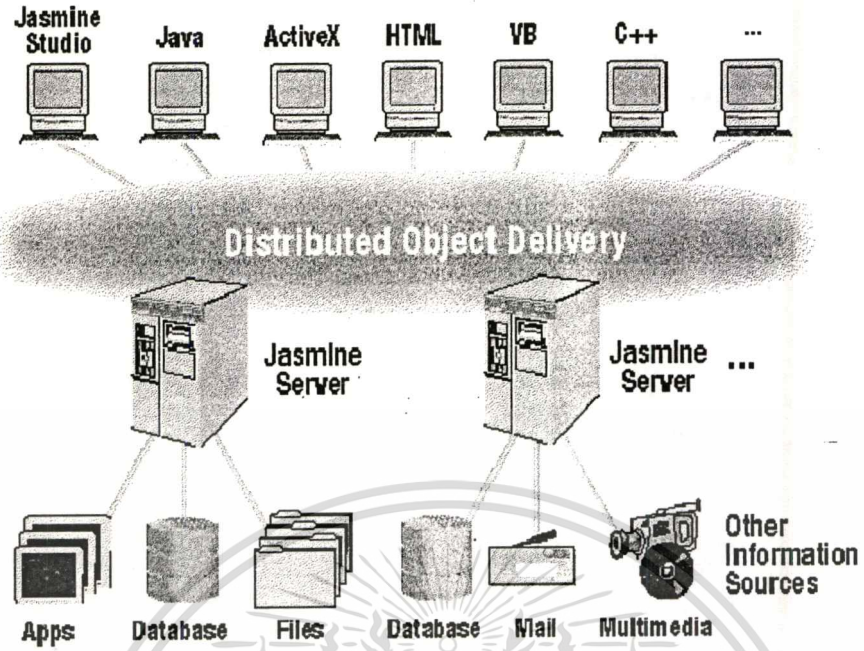
นอกจากนี้ยังทำงานร่วมกับระบบงานฐานข้อมูลเชิงสัมพันธ์ที่มีอยู่ได้โดยมีการทำไลบรารีคลาส SQL ไว้สำหรับการ access กับข้อมูลที่ไม่ได้อยู่บน Jasmine

6.2.6 Portable and Extensible

Jasmine ทำให้ผู้พัฒนาสามารถสร้างแอปพลิเคชั่นเพียงครั้งหนึ่งและนำไปใช้กับ Internet, Intranet และ Extranet ได้อย่างง่าย แอปพลิเคชั่นที่สร้างจาก Jasmine สามารถเข้าถึงได้จากที่ใดก็ได้โดยผ่านเว็บ

6.3 สถาปัตยกรรม

Jasmine Application ถูกกระจาย สามารถเอ็กซีคิวต์บนเครื่องไคลเอนต์ซึ่งอาจเป็นแบบ Standalone หรือภายในเว็บบราวเซอร์ก็ได้ และติดต่อกับ Database Server ซึ่งจะเอ็กซีคิวต์ Business Logic ต่างๆ และจัดเก็บสำหรับออบเจ็กต์



รูปที่ 21 แสดง Simplified Jasmine Architecture

- มี Database Engine ที่มีประสิทธิภาพสูงในการเก็บข้อมูล, การบริหารจัดการ (Administration), และการเข้าถึงนิยามของคลาส และออบเจ็กต์พร้อมๆ กัน
- ฐานข้อมูลเชิงวัตถุสนับสนุนโครงสร้างข้อมูลที่มีความซับซ้อนและข้อมูลที่มีขนาดใหญ่ซึ่งเป็นสิ่งที่ต้องการของแอปพลิเคชั่นมัลติมีเดียในปัจจุบัน
- Jasmine สนับสนุนการใช้ C, C++ และ Javaในการเขียน Method
- มีไลบรารีสนับสนุนด้านมัลติมีเดีย, SQL Access และคลาสที่เป็นยูทิลิตี้อื่นๆ
- สนับสนุน ActiveX

6.4 Object-Oriented Database Engine

6.4.1 Robust Database

ฐานข้อมูลเชิงวัตถุของ Jasmine มีความ Robust ซึ่งมีความจำเป็นสำหรับแอปพลิเคชั่นบนอินเตอร์เน็ตโดยเฉพาะสำหรับ Electronic Commerce นอกจากนี้ยังมีการจัดการด้าน Integrity, ความปลอดภัย, รวมถึงการจัดการกับงานทรานแซคชั่น

6.4.2 มีฟังก์ชั่นของออบเจ็กต์โอเรียนเต็ดครบถ้วน เช่น

- Multiple inheritance
- Instance properties และ class properties
- Instance methods และ class methods
- Complex data structure

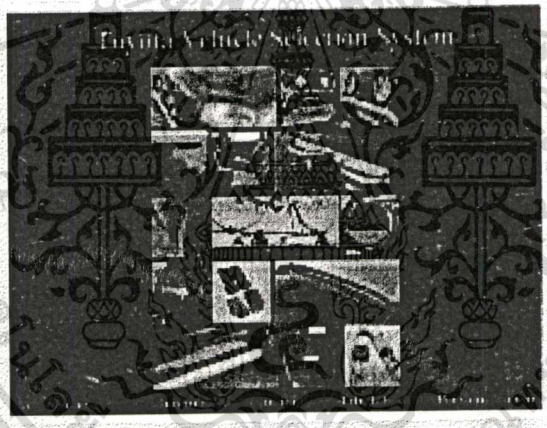
6.4.3 การรวมกับฐานข้อมูลอื่น

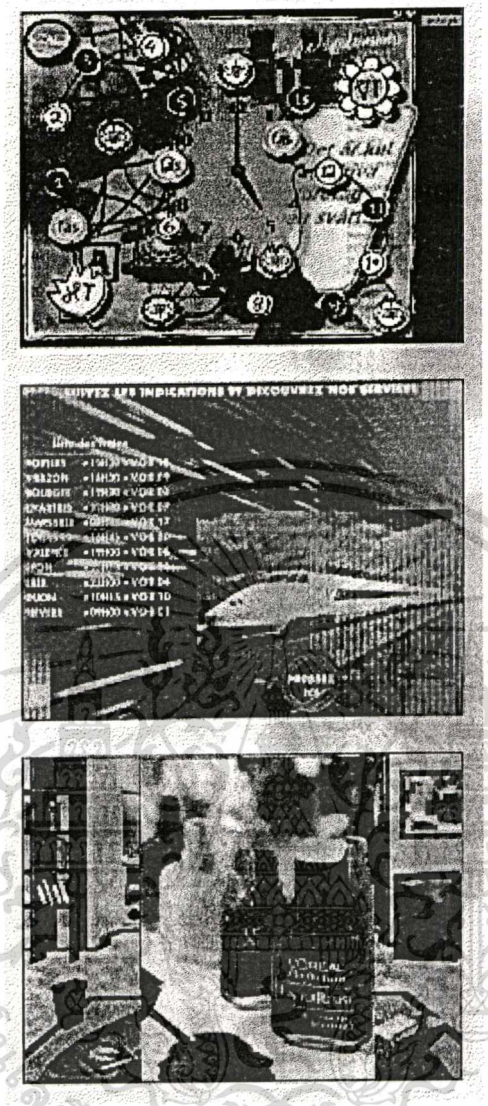
Jasmine สนับสนุนระบบเชิงสัมพันธ์ เช่น Ingres II, Oracle, Sybase, Informix และ SQLServer รวมถึงข้อมูลที่อยู่บนเมนเฟรมได้แก่ CA-IDMS, CA-Datacom, VSAM และ DB2 โดยจะแทนข้อมูลเหล่านั้นเป็นเหมือนออบเจ็กต์ การรวมกันนี้ method ของ Jasmine objects จะอ้างถึงข้อมูลที่จะใช้ในแอปพลิเคชั่นทางธุรกิจที่ใช้อยู่อย่างมีประสิทธิภาพเหมือนไม่มีรอยต่อ

6.4.4 Flexible APIs

Jasmine จัดหา APIs สำหรับการเข้าถึงจาก C และ C++ ส่วน Java bindings ใช้สำหรับการเข้าถึงฐานข้อมูลจาก Java application และ applet นอกจากนี้ ActiveX controls มีไว้สำหรับ Visual Basic หรือ เครื่องมือในการพัฒนาอื่นๆที่สนับสนุน ActiveX สามารถติดต่อกับฐานข้อมูลได้

6.5 ตัวอย่างแสดงแอปพลิเคชั่นที่พัฒนาด้วย Jasmine





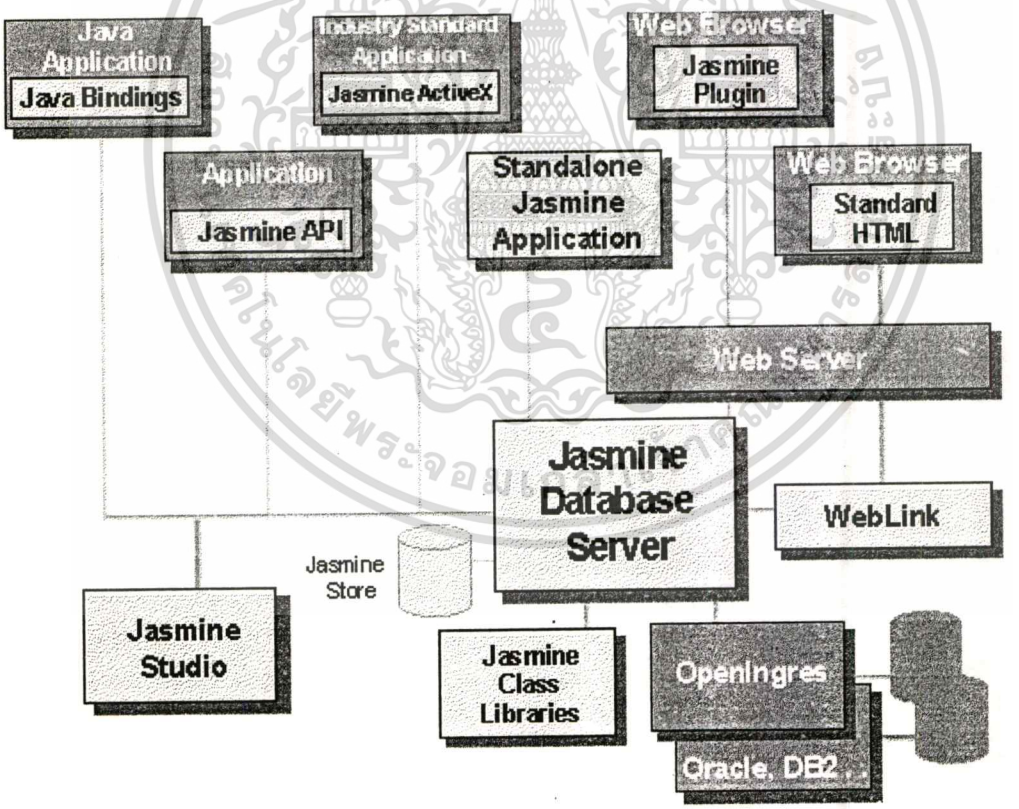
รูปที่ 22 แสดงตัวอย่างแอปพลิเคชั่นที่พัฒนาด้วย Jasmine

บทที่ 7

การพัฒนาแอปพลิเคชั่นโดยใช้ Jasmine

7.1 บทนำ

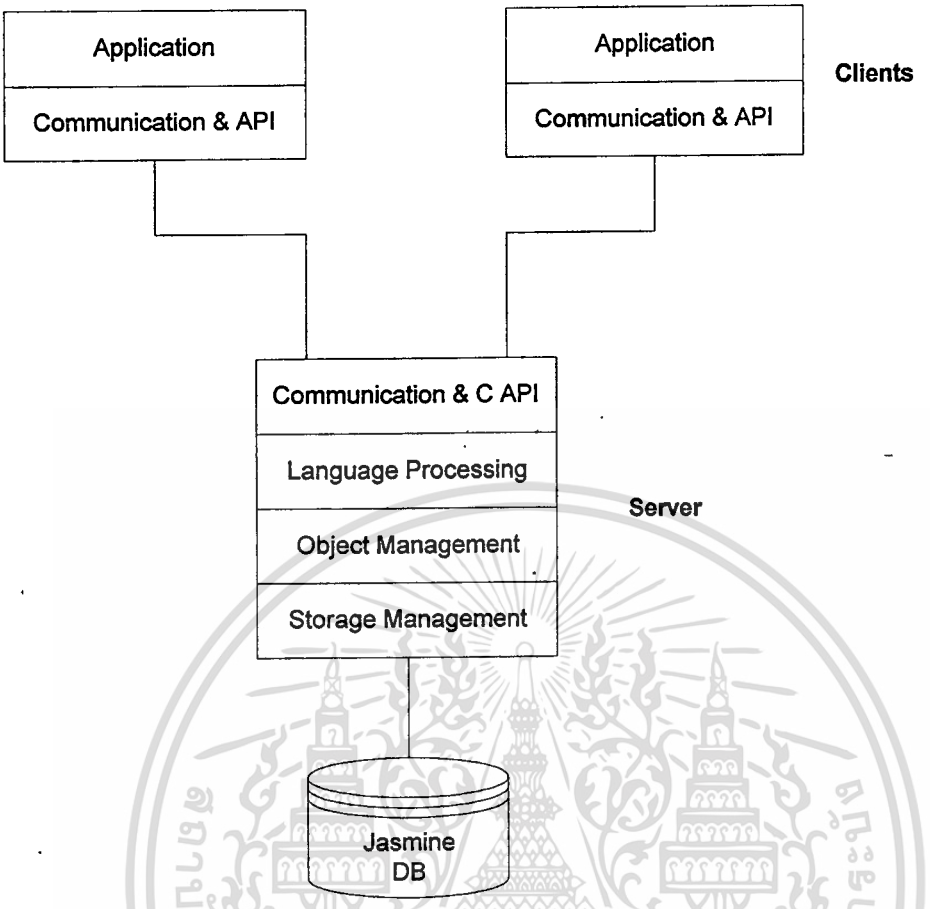
Jasmine เป็นระบบการจัดการฐานข้อมูลเชิงวัตถุ โดยมีสถาปัตยกรรมเป็นแบบไคลเอนต์เซิร์ฟเวอร์ ที่ถูกออกแบบมาเฉพาะเพื่อตอบสนองความต้องการของผู้ใช้เพื่อพัฒนาแอปพลิเคชั่นทางด้าน Internet และ Intranet เนื่องจากมีการจัดการกับข้อมูลขนาดใหญ่ เช่น multimedia data ได้เป็นอย่างดี องค์ประกอบของ Jasmine ดังรูปที่ 23



รูปที่ 23 แสดงองค์ประกอบของ Jasmine

- Jasmine Store เป็นที่เก็บ physical data ของฐานข้อมูล Jasmine

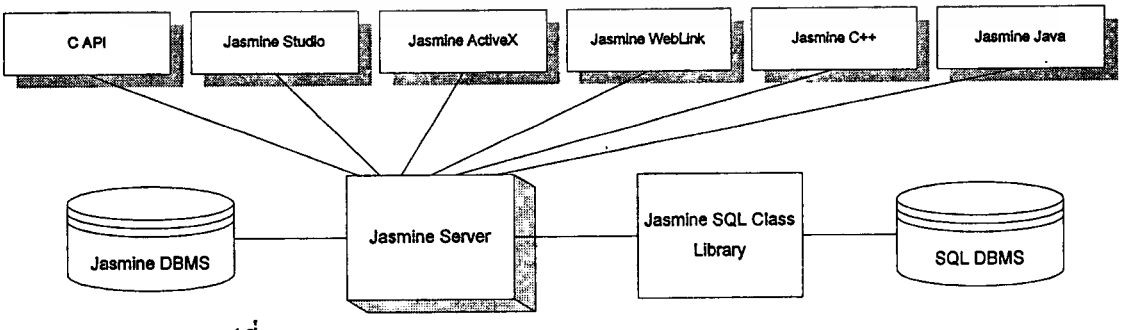
- Jasmine Database Server เป็นเซิร์ฟเวอร์ที่เก็บข้อมูลมัลติมีเดียสำหรับใช้ในแอปพลิเคชั่น รวมถึงมีหน้าที่จัดการกับข้อมูลและ method ในฐานข้อมูล
- Jasmine Client เป็น environment ที่ใช้รันแอปพลิเคชั่น Jasmine ซึ่งต้องมีการติดต่อกันกับ Jasmine Database Server
- Jasmine Class Libraries เป็นกลุ่มของ Class Families ที่ Jasmine จัดหาไว้ให้สำหรับ multimedia data
- Jasmine Studio เป็นทูลที่ใช้ในการพัฒนาแอปพลิเคชั่น (GUI-based)
- Jasmine API เป็นเซตของ function call ที่ใช้จัดการกับฐานข้อมูล Jasmine ผ่านภาษา C, C++ และภาษาโปรแกรมอื่นๆ
- Java Bindings เป็นอินเตอร์เฟสระหว่าง Jasmine กับ Java ทำให้ภาษา Java สามารถเข้าถึงฐานข้อมูล Jasmine ได้
- Jasmine ActiveX เป็นตัวอำนวยความสะดวกซึ่งทำงานร่วมกันกับระบบปฏิบัติการในการทำให้สามารถเข้าถึงฐานข้อมูล Jasmine โดยใช้ Visual Basic หรือตัวพัฒนาโปรแกรมอื่นๆที่สนับสนุน ActiveX
- Jasmine Plug-In เป็น plug-in ในเว็บบราวเซอร์ที่ช่วยให้สามารถรัน Jasmine Application และเข้าถึง Jasmine Database โดยผ่าน World Wide Web
- WebLink เป็น HTML renderer ที่จัดการการเข้าถึงออบเจ็กต์ Jasmine จาก HTML มาตรฐานบน World Wide Web โดยไม่อาศัย Jasmine plug-in
- Relational Data Access เป็นเซตของ Jasmine Classes ที่ทำให้ Jasmine Application สามารถเข้าถึงและจัดการกับข้อมูลบนระบบฐานข้อมูลเชิงสัมพันธ์ได้



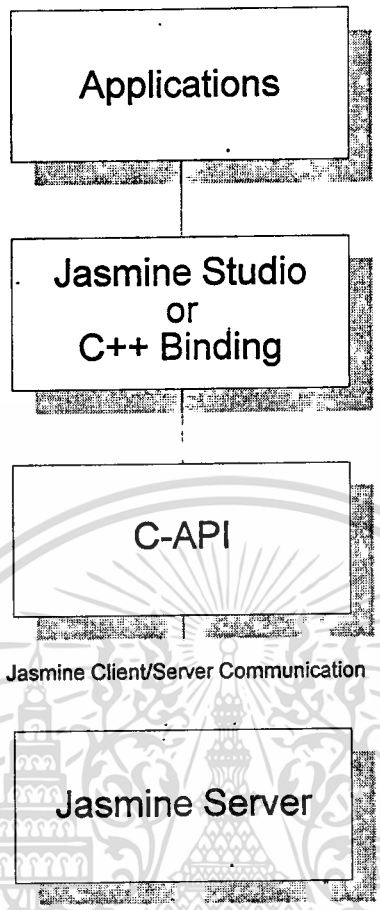
รูปที่ 24 แสดงสถาปัตยกรรมของ Jasmine

สถาปัตยกรรมของ Jasmine จะเป็นไคลเอนต์เซิร์ฟเวอร์แบบกระจาย Complex Method ที่เกี่ยวกับ Business Logic ต่างๆ จะถูกเอ็กซีคิวต์บนเซิร์ฟเวอร์ เป็นการลดความจำเป็นในการถ่ายข้อมูลขนาดใหญ่ไปยัง Client Application

สำหรับ Jasmine Client จะรันแอปพลิเคชั่นเป็นแบบ Stand-alone หรือ plug-in ไปยังเว็บบราวเซอร์ เช่น Netscape Navigator หรือ Microsoft Internet Explorer



รูปที่ 25 แสดง Jasmine application Development Environment



รูปที่ 26 แสดงสถาปัตยกรรมของ Jasmine applications โดยใช้ Jasmine Studio หรือ C++ Binding

Jasmine เป็น Pure Object Database ซึ่งสนับสนุน

- User-defined data type
- Single Inheritance และ Multiple Inheritance
- Encapsulation
- Instance properties และ class properties
- Instance methods และ class method
- Polymorphism

นอกจากนี้ Jasmine ยังสนับสนุนโครงสร้างข้อมูลที่มีความซับซ้อนและข้อมูลที่มีขนาดใหญ่ ได้แก่ข้อมูลทางด้านมัลติมีเดีย โดย Jasmine ได้สร้างเป็น Multimedia Class Library เพื่อสนับสนุนข้อมูลประเภทนี้โดยเฉพาะ ทั้งรูปภาพ (Images), ภาพเคลื่อนไหว (Frame Animation Sequences), Audio และ Video File และ Rich Text Format (RTF) files

นอกจากนี้ Jasmine ยังมีระบบการจัดการระหว่างไคลเอนต์และเซิร์ฟเวอร์, Concurrency Control, Transaction management, Recovery, Access Control และ Database Administration

การพัฒนา Jasmine Application ทำได้สองวิธีคือ

- Jasmine C API
- Jasmine Studio

7.2 Jasmine C API

Jasmine C API เป็นการสร้างแอปพลิเคชั่นโดยใช้ภาษา C/C++ หรืออาจเป็นทูลอื่นๆที่สนับสนุน external Dynamic Link Libraries (DLLs) โดยแอปพลิเคชั่นสร้างด้วย C API นี้ Main Program จะเขียนจาก C หรือ C++ ซึ่งจะใช้ API function calls ในการเข้าถึงและจัดการกับ Jasmine database โดยใช้ ODQL ซึ่งเป็นภาษาโปรแกรมที่ใช้จัดการกับฐานข้อมูลเชิงวัตถุที่ Jasmine จัดไว้ให้ ซึ่งอาจจะใช้ ODQL โดยตรง, Embed ODQL ในโปรแกรมภาษา C หรือ C++ หรือเอ็กซีคิวต์ ODQL โดย C API

7.2.1 Preliminary Information ในโปรแกรม

C Header Files	`#include <windows.h>` `#include <stdio.h>`

C API Header File	`extern "C" {`
	`#include <codqlhdr.h>`
	`}`

Error Checking Routine	ตัวอย่างข้างล่างนี้เป็นตัวอย่างของรูทีนการตรวจสอบความผิดพลาด ถ้ามีความผิดพลาดเกิดขึ้น message box จะแสดงโค้ดผิดพลาดออกมา
	
	`void ShowErrorMessage (odbSessH sh)`
	`{`
	`long ret_len;`
	`char error_message[1000];`
	`long error_code;`
	
	`odbGetError(sh, &error_code, error_message, sizeof(error_message) - 1, &ret_len);`
	`MessageBox(NULL, error_message, "C-API ERROR", MB_OK);`
	`}`

Entry Point for Starting the Program	`int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)` `{`

Variable Declaration	โค้ดข้างล่างนี้กำหนดค่าสำหรับตัวแปร ODQL ที่จะใช้ในตัวอย่างโปรแกรมบูติกนี้ โครงสร้าง odbData (odbData Structure) จะแปลง host language values ไปยัง ODQL ที่เทียบเท่ากันและในทางตรงข้ามแปลงจาก ODQL ไปเป็น host language values ด้วย
	`odbSessH sh;`
	`odbStatus status;`
	`odbData data[10];`
	`odbData data2;`
	`odbScanId scanId;`
	`int j;`
	`char* name;`
	`char fn[128];`
	`char msg[256];`
	`int n_ret;`

7.2.2 การติดต่อกับ Jasmine Database

โค้ดต่อไปนี้จะเป็นการสร้าง Session กับ Jasmine Server และกำหนดไฟล์สภาวะแวดล้อมของไคลเอนต์ที่ใช้

```
status = odbSesStart(&sh,
"usmlsxaa::JASMINE/JASMINE",NULL,NULL,"C:\\Jasmine\\
Jasmine\\envfile\\filename.env");
```

- odbSesStart() และ odbSesEnd() Functions

ฟังก์ชัน odbSesStart() จะเป็นฟังก์ชันแรกที่ถูกเรียกเสมอ ก่อนออกจากแอปพลิเคชั่นควรปิด session โดยการใช้ฟังก์ชัน odbSesEnd()

- Session Handle

ตัวแปรโกลบอล sh จะหมายถึง session handle ได้มาค่าจากฟังก์ชัน odbSesStart() ทุกๆฟังก์ชันจำเป็นต้องติดต่อกับ Jasmine server ต้องมี session handle เป็นอาร์กิวเมนต์ โดยใช้เป็นตัวแปรโกลบอล

7.2.3 การตรวจสอบการ Return Code และแสดง Error Message

โค้ดต่อไปนี้แสดง Error Checking โดยใช้ฟังก์ชัน odbExecODQL() เพื่อเข้าถึงโดยตรงกับ Jasmine database สำหรับการเอ็กซีคิวต์ ODQL statement แล้วจึงเริ่มทรานแซคชั่นด้วย Transaction.start

```
status = odbExecODQL(sh, "Transaction.start();", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
```

7.2.4 การเซตดีฟอลต์คลาสแฟมิลี่ (Setting the Default Class Family)

โค้ดต่อไปนี้เป็นการกำหนด Default class family (CAStore)

```
status = odbExecODQL(sh, "defaultCF CAStore;", 0, 0);
if (status != ODB_NORMAL) {
ShowErrorMessage(sh);
return 0;
}
```

7.2.5 การประกาศตัวแปร

โค้ดต่อไปนี้ "T" แทน system class named Tuple ซึ่งถูกใช้เพื่อสร้างค่าจะเรียกว่า *tuples* และมีคอมโพเนนต์ซึ่งจะเรียกว่า *fields*

```
status = odbExecODQL(sh, "T[String name, mediaCF::Bitmap
photo] set ts;", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
```

7.2.6 การเอ็กซีคิวต์คิวรี (Executing the Query)

โค้ดคิวรีข้างล่างนี้เป็นการหาสินค้าที่ได้รับนิยมซึ่งมีค่ามากว่าศูนย์ ผลลัพธ์ของการคิวรีจะถูกแทนในเซตของ *ts*

```
status = odbExecODQL(sh, "ts = [x.name, x.photo] from
Top x where x.hot > 0;", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
```

7.2.7 Allocation Space for Fetching Data

ฟังก์ชัน odbAllocData() กำหนดโครงสร้างข้อมูลในตัวอย่างนี้จะกำหนดเนื้อที่สำหรับ 10 items (ผลของการคิวรี)

```
for (j = 0; j < 10; ++j) {
        data[j] = odbAllocData();
    }
```

7.2.8 การเปิดเซตของผลลัพธ์ (Opening the Result Set)

โค้ดนี้เป็นการเปิด scan บน *ts* ซึ่งเป็น set named

```
status = odbScanOpen(sh, "ts", &scanId, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
```

7.2.9 การปิดเซตของผลลัพธ์ (Closing the Result Set)

โค้ดนี้เป็นการปิด scan บน *ts* ซึ่งเป็น set named

```
status = odbScanClose(sh, scanId);
```

7.2.10 การสิ้นสุดทรานแซคชั่น

```
status = odbExecODQL(sh, "Transaction.end();", 0, 0);

    for (j = 0; j < n_ret; ++j) {
        odbFreeData(data[j]);
    }

    return 1;
}
```

7.2.11 ตัวอย่างโปรแกรมที่สร้างด้วย C API

```
#include <windows.h>
#include <stdio.h>

extern "C" {
#include <codqlhdr.h>
}

void ShowErrorMessage (odbSessH sh)
{
    long            ret_len;
    char            error_message[1000];
    long            error_code;

    odbGetError(sh, &error_code, error_message, sizeof
(error_message) - 1, &ret_len);
    MessageBox(NULL, error_message, "C-API ERROR",
MB_OK);
}
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
    odbSessH  sh;
    odbStatus status;
    odbData         data[10];
    odbData         data2;
    odbScanId       scanId;
    int             j;
    char*           name;
    char            fn[128];
    char            msg[256];
    int             n_ret;
```

```
    status = odbSesStart(&sh,
"usmlsxaa::JASMINE/JASMINE",NULL,NULL,"C:\\Jasmine\\Jasm
ine\\envfile\\filename.env");

    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
     status = odbExecODQL(sh, "Transaction.start();", 0,
0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
    status = odbExecODQL(sh, "defaultCF CAStore;", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
    status = odbExecODQL(sh, "T[String name,
mediaCF::Bitmap photo] set ts;", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
    status = odbExecODQL(sh, "ts = [x.name, x.photo] from
Top x where x.hot > 0;", 0, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }

    for (j = 0; j < 10; ++j) {
        data[j] = odbAllocData();
    }
status = odbScanOpen(sh, "ts", &scanId, 0);
    if (status != ODB_NORMAL) {
        ShowErrorMessage(sh);
        return 0;
    }
    status = odbScanFetch(sh, scanId, 10, &n_ret, data);
    if (status != ODB_NORMAL && status !=
ODB_SCAN_END) {
        ShowErrorMessage(sh);
        return 0;
    }
    for (j = 0; j < n_ret; ++j) {
        name = odbGetStr(odbFieldAt(data[j], 0));

        data2 = odbFieldAt(data[j], 1);
        wsprintf(fn, "test%d.bmp", j + 1);

        status = odbGetMMDataToFile(sh, data2, fn);
        if (status != ODB_NORMAL) {
            ShowErrorMessage(sh);
            return 0;
        }
    wsprintf(msg, "File %s has been created for photo
property of %s.", fn, name);
        MessageBox(NULL, msg, "Info", MB_OK);
    }

    status = odbScanClose(sh, scanId);

    status = odbExecODQL(sh, "Transaction.end();", 0, 0);

    for (j = 0; j < n_ret; ++j) {
        odbFreeData(data[j]);
    }

    return 1;
}
```

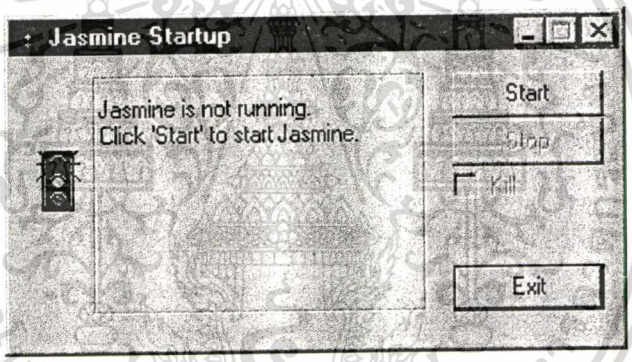
7.3 Jasmine Studio

Jasmine Studio จะเป็นทูลที่ใช้ในการออกแบบ, ทำโปรโตไทพ์, Debug, การ Deploy แอปพลิเคชั่นลงเว็บ และจัดการกับฐานข้อมูล คลาสต่างๆ (Database Administration) ในที่นี้จะขออธิบายแต่การใช้ Jasmine Studio ในการสร้างแอปพลิเคชั่น

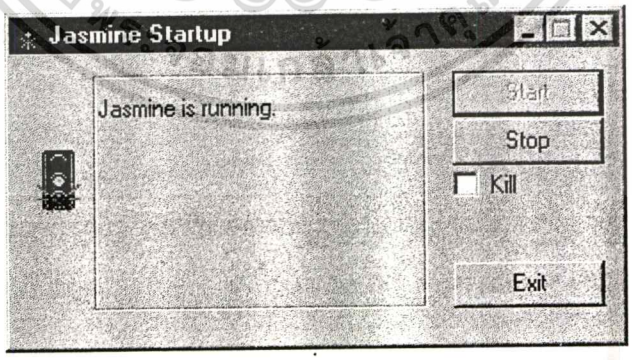
Server Requirement

- Pentium Processor
- RAM 32 MB (64 MB preferred)
- เนื้อที่ว่างใน HDD 200 MB (300 MB preferred)
- Microsoft Windows NT 4.0
- Microsoft Visual C++ Compiler 4.2 or higher

ก่อนเริ่มการใช้ Jasmine Studio ต้องมีการ start Jasmine services ก่อนดังรูปที่ 27, 28



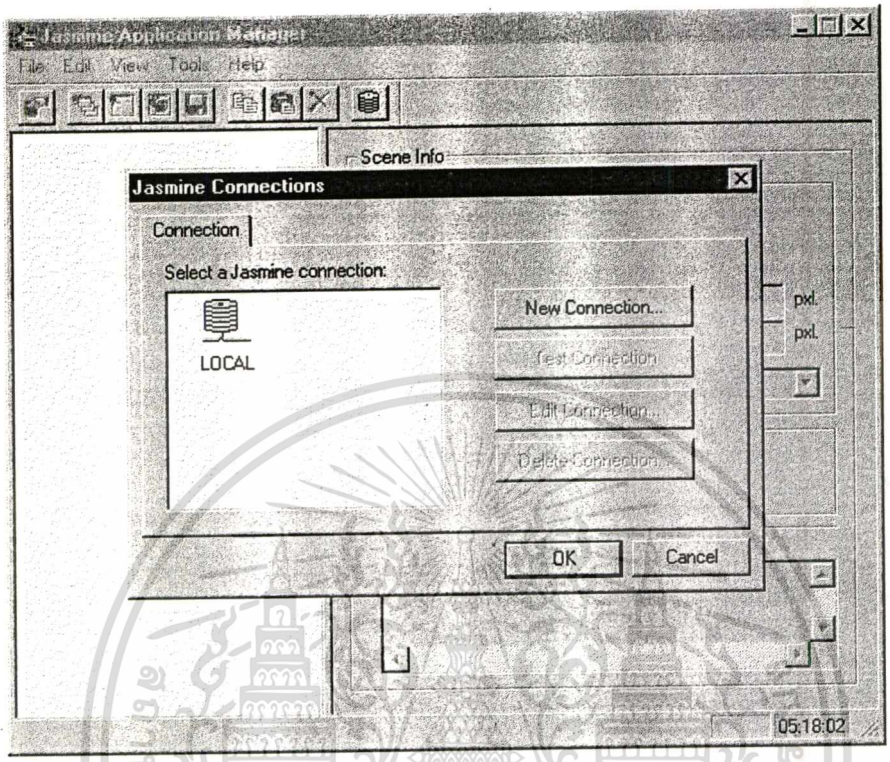
รูปที่ 27 แสดงหน้าต่างก่อน Start Jasmine Services



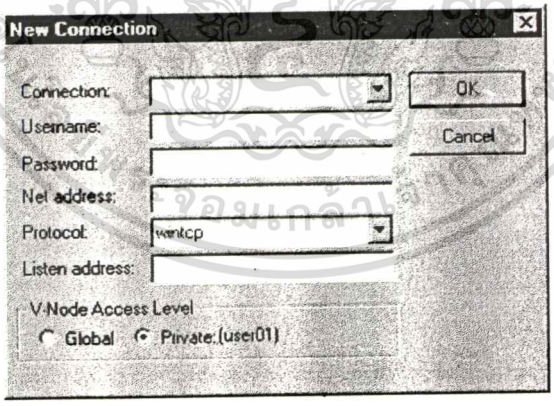
รูปที่ 28 แสดงหน้าต่างเมื่อ Start Jasmine Services เรียบร้อยแล้ว

เมื่อ Start services แล้วจึงเรียกโปรแกรม Jasmine Studio ขึ้นมาซึ่งต้องมีการ Connect กับ Jasmine Database Server ก่อน โดยเลือก Server ที่ต้องการเชื่อมต่อหรืออาจสร้าง Connection ขึ้น

ใหม่ก็ได้ โดยการเลือก New Connection (Default ที่ได้มาหลังจากการ Install จะได้ Connection LOCAL ซึ่งไม่ควรลบ Connection นี้ไป) ดังรูปที่ 29

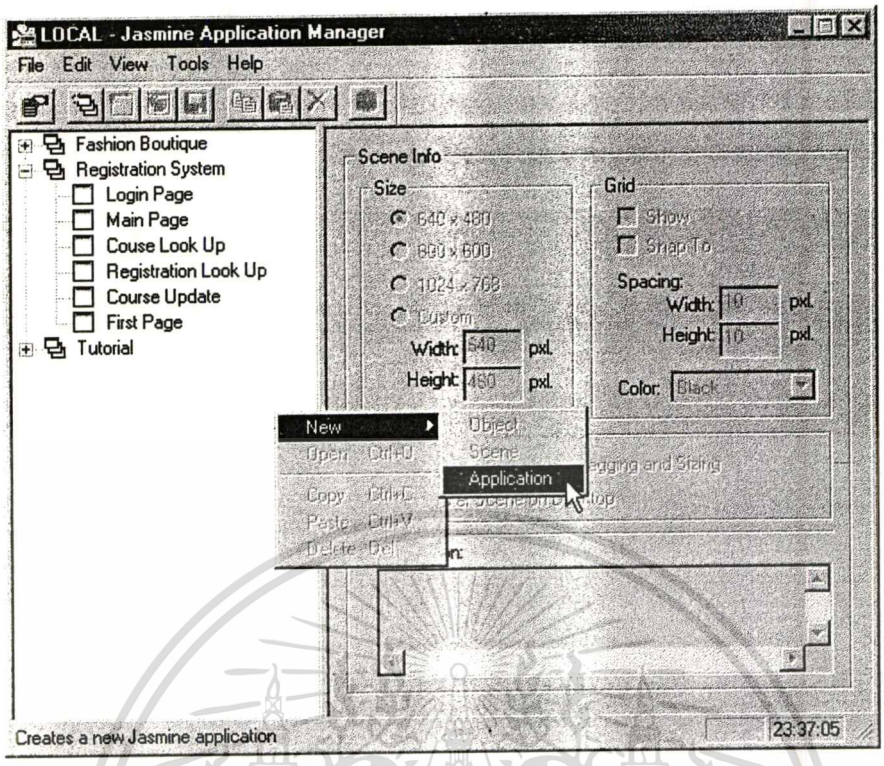


รูปที่ 29 แสดงหน้าต่างของ Jasmine Connections



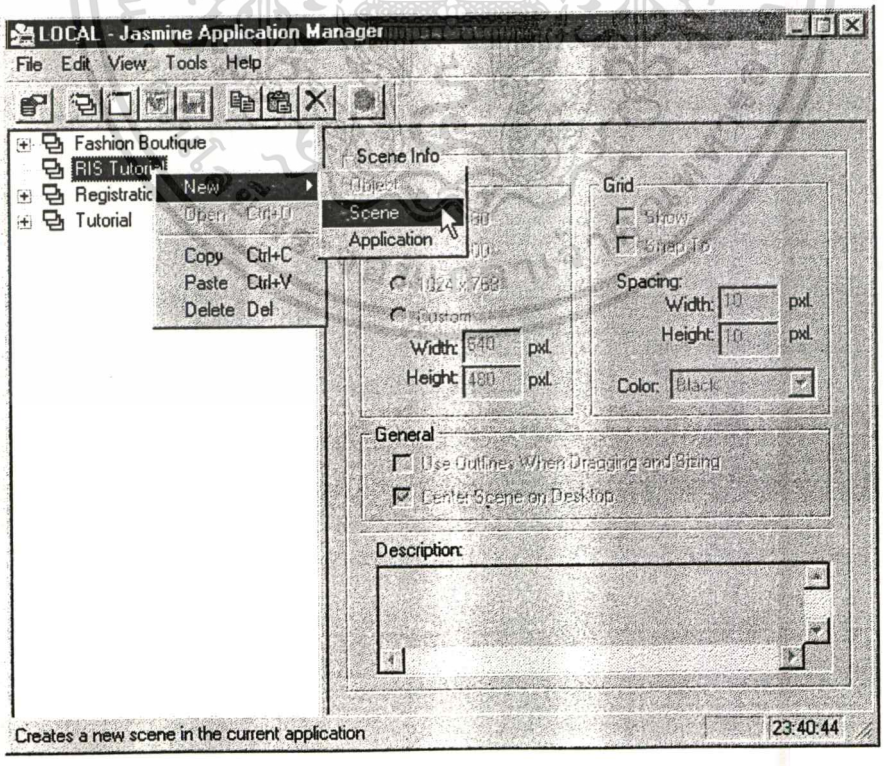
รูปที่ 30 แสดงหน้าต่าง New Connection

เมื่อเลือก Server ได้แล้วจะเข้าสู่หน้าต่าง Jasmine Application Manager เริ่มสร้างแอปพลิเคชั่นจากเมนู File/New/Application หรือใช้เมาส์คลิ้กขวาจะเป็นเมนู ดังรูป 31 ในที่นี้จะแสดงการสร้างแอปพลิเคชั่น RIS Tutorial ซึ่งเป็นแอปพลิเคชั่นที่ใช้เป็นโปรโตไทพ์สำหรับงานโครงการพัฒนาระบบซึ่งปรากฏในภาคผนวกท้ายเล่ม



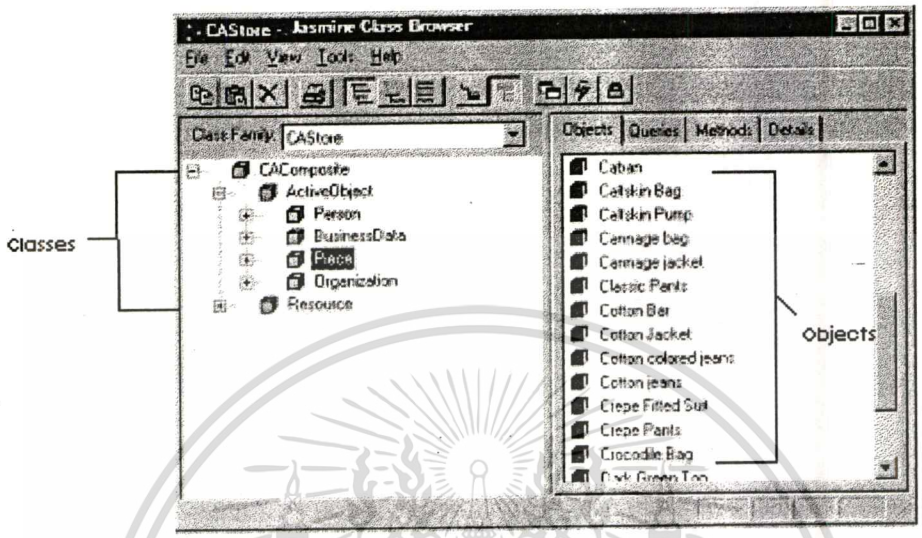
รูปที่ 31 แสดงการสร้างแอปพลิเคชั่นใหม่จากหน้าต่าง Jasmine Application Manager

จากนั้นจะสร้าง Scene ใหม่ดังรูปที่ 32

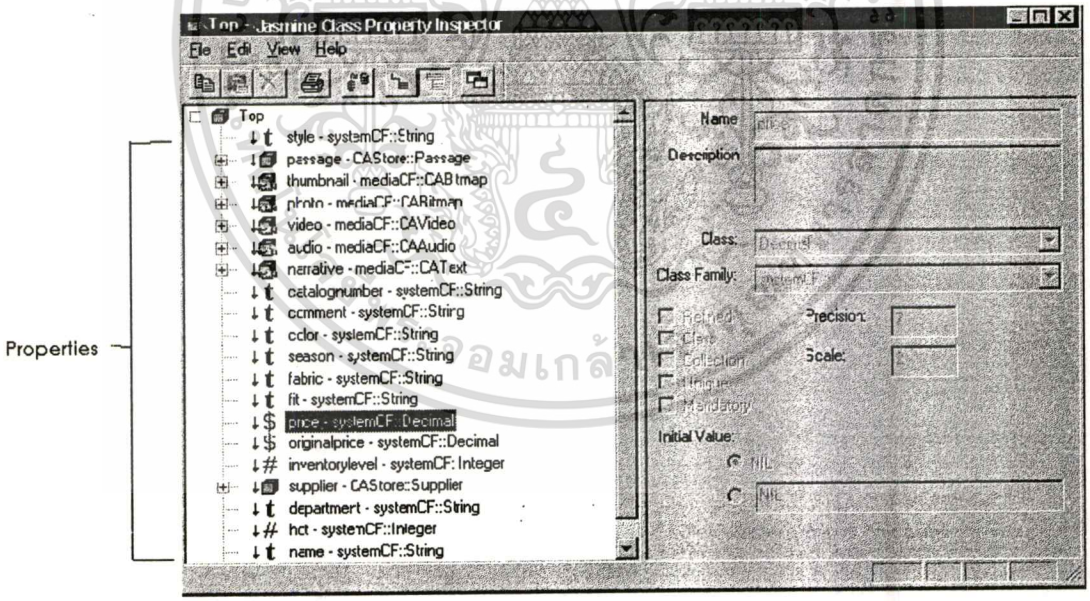


รูปที่ 32 แสดงการสร้าง Scene ใหม่จากหน้าต่าง Jasmine Application Manager

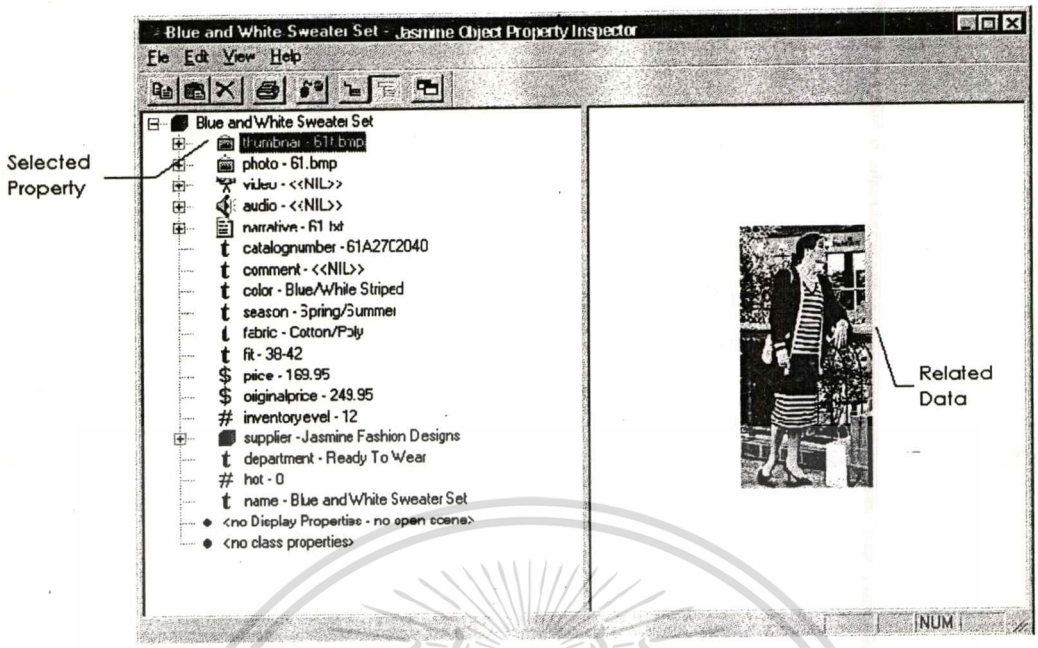
เมื่อต้องการสร้างคลาสหรือออบเจ็กต์ใหม่สามารถทำได้โดยเลือก File/Database Administration จะได้หน้าต่าง Jasmine Class Browser ดังรูปที่ 33 หน้าต่างนี้สามารถสร้างคลาสและออบเจ็กต์รวมถึงอินสแตนส์ต่างๆพร้อมกำหนดการ Query, Method และ รายละเอียดต่างๆได้



รูปที่ 33 แสดงหน้าต่าง Jasmine Class Browser



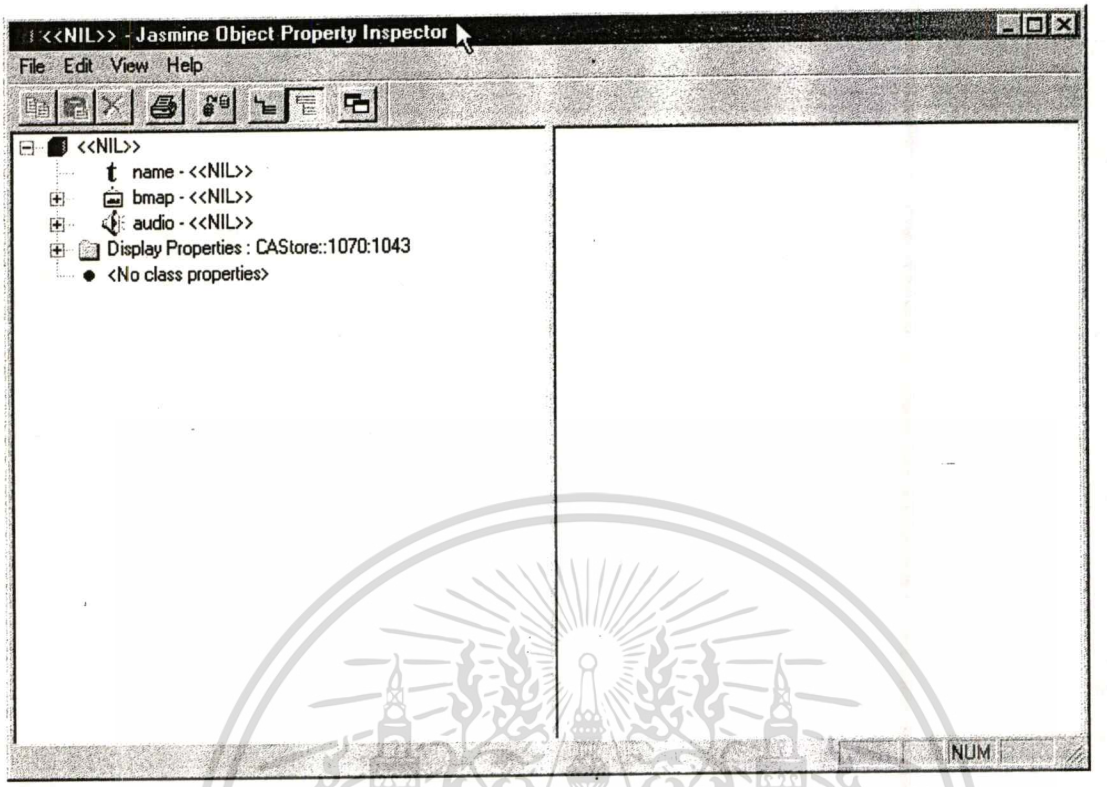
รูปที่ 34 แสดงหน้าต่าง Jasmine Class Property Inspector



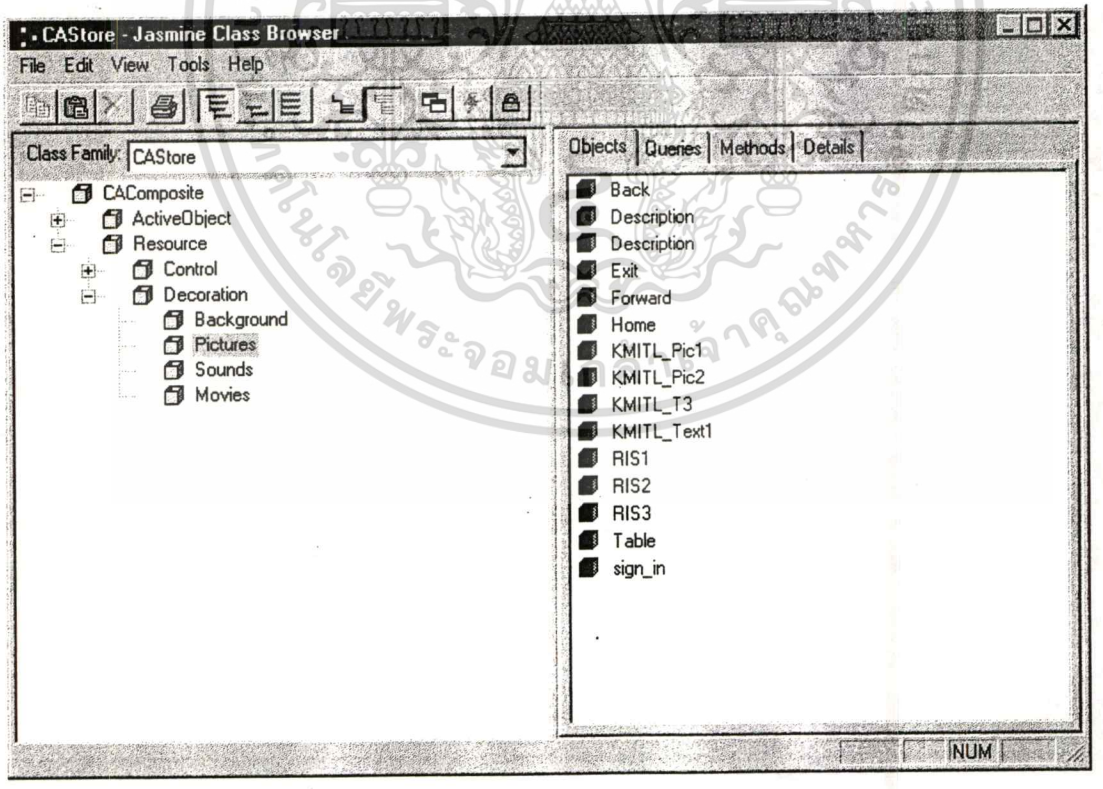
รูปที่ 35 แสดงหน้าต่าง Jasmine Object Property Inspector

สำหรับวิธีการพัฒนาแอปพลิเคชั่นโดยใช้ Jasmine Studio ในที่นี้จะขอยกตัวอย่างการทำงานของ Scene Registration Look Up ซึ่งเป็นโมดูลหลักอันหนึ่งในโปรโตไทพ์ที่ปรากฏในภาคผนวก โดยที่ Scene นี้จะเป็นส่วนของการดูรายละเอียดการลงทะเบียนของนักศึกษาที่ได้มีการลงทะเบียนไปแล้ว

เริ่มจากการหาจัดเตรียมรูปภาพต่างๆที่จะใช้ประกอบในแอปพลิเคชั่น โดยการสร้างออบเจ็กต์ที่เป็นรูปภาพเหล่านั้น Properties ของออบเจ็กต์รูปภาพเป็นดังรูปที่ 36 วิธีการสร้างคือให้ลากไฟล์ที่เป็น Image Format ใส่ลงใน bmap properties ส่วน properties อื่นๆให้ใส่ตามความเหมาะสม เมื่อสร้างออบเจ็กต์เรียบร้อยแล้วจะปรากฏออบเจ็กต์นั้นดังรูปที่ 37

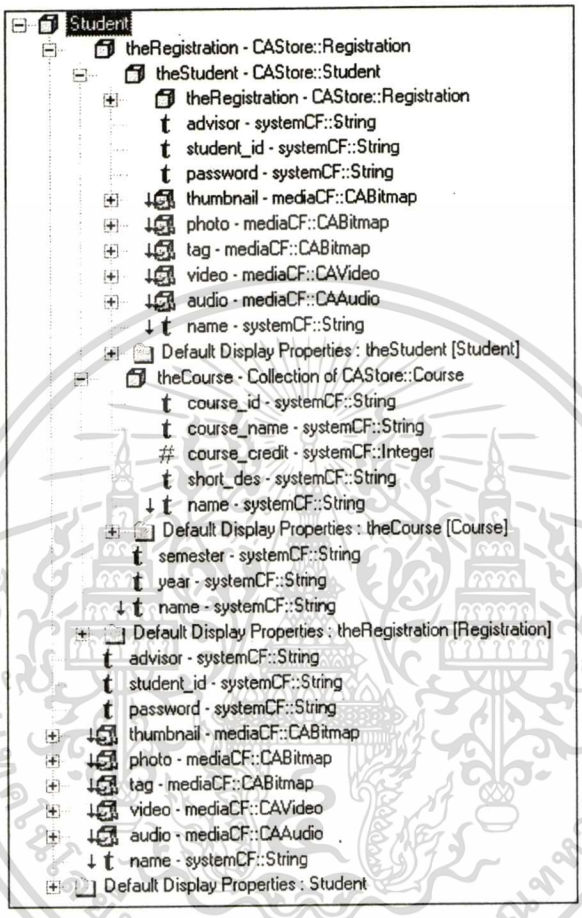


รูปที่ 36 แสดง Properties ของออบเจ็กต์รูปภาพ

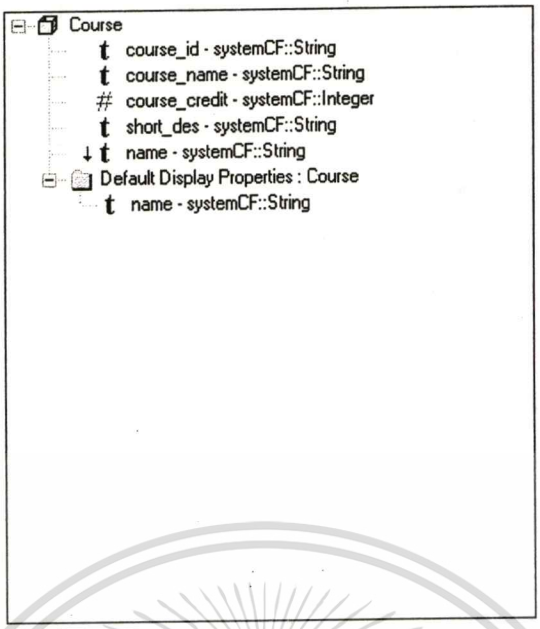


รูปที่ 37 แสดงออบเจ็กต์ของคลาส Pictures ที่สร้างไว้ในระบบ

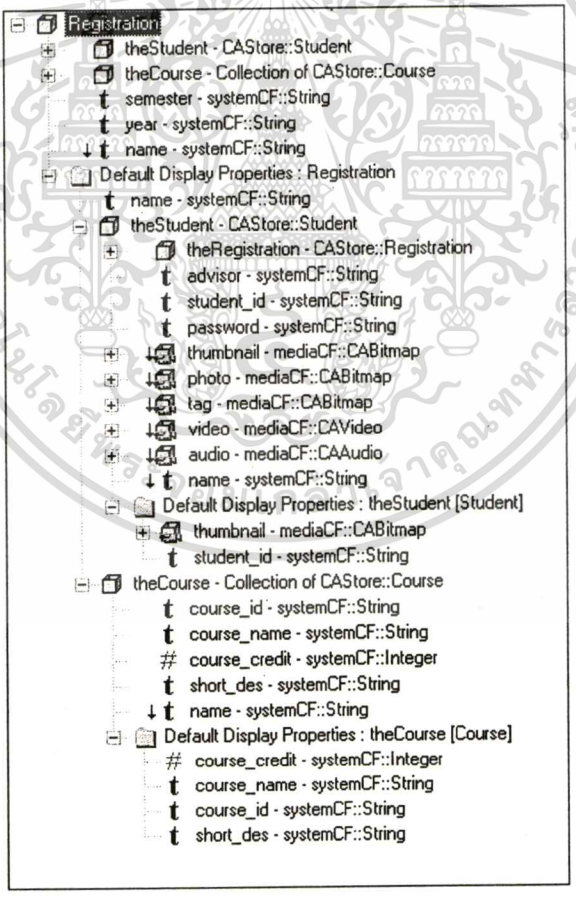
สำหรับโปรโตไทพ์ Registration Information System จะมีคลาสหลักๆที่ใช้อยู่ 3 คลาสได้แก่ Student, Course, และ Registration ซึ่งแต่ละคลาสมี Properties ดังแสดงในรูปที่ 38, 39, และ 40 ตามลำดับ โดยการสร้างคลาสจะสร้างจาก Database Administration ซึ่งอยู่ในเมนู File



รูปที่ 38 แสดง Properties ของคลาส Student

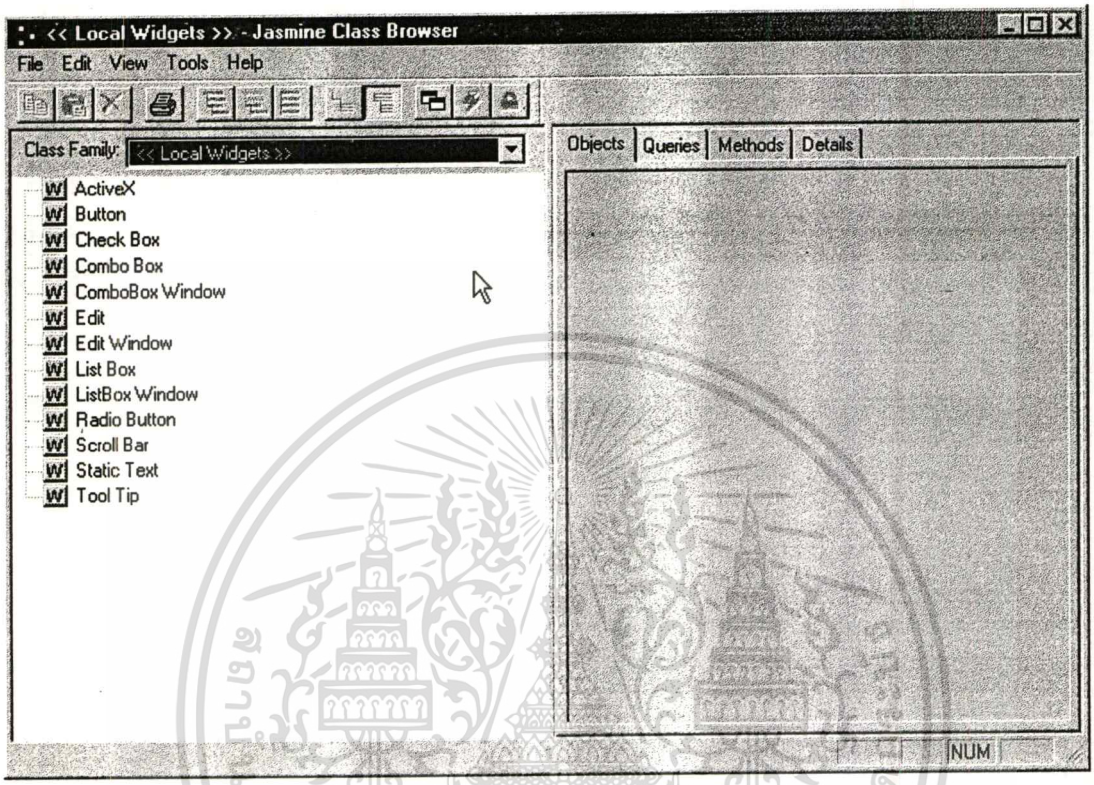


รูปที่ 39 แสดง Properties ของคลาส Course



รูปที่ 40 แสดง Properties ของคลาส Registration

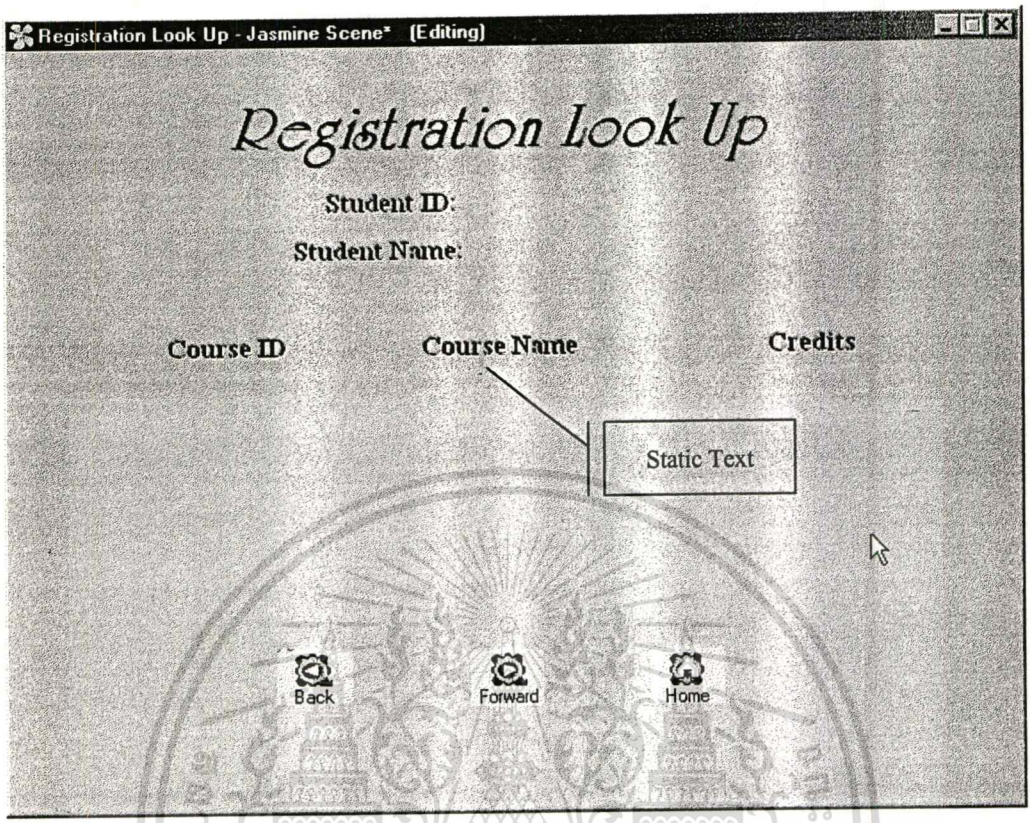
เมื่อได้ออบเจ็กต์รูปภาพต่างๆแล้ว ต่อไปจะทำการตกแต่ง Scene ในที่นี้ได้แก่พวก Text และ Edit Box ต่างๆ ซึ่งออบเจ็กต์เหล่านี้อยู่ใน Class Family ที่ชื่อ <<Local Widgets>> ดังรูปที่ 41



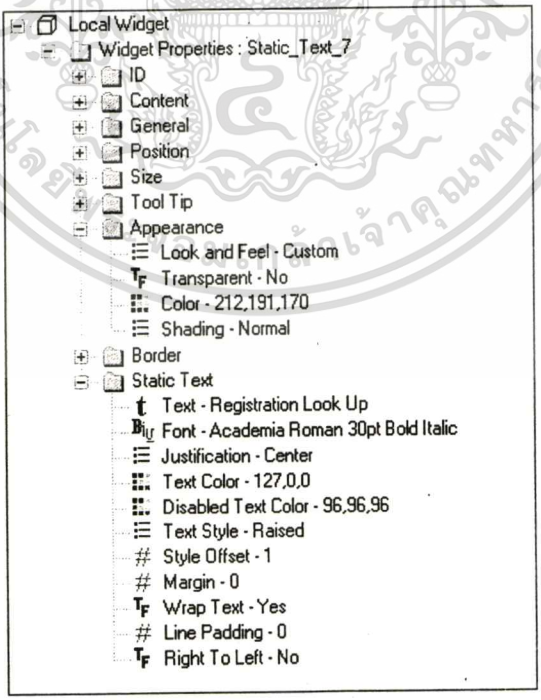
รูปที่ 41 แสดงออบเจ็กต์ที่อยู่ใน Class Family: <<Local Widgets>>

ให้ลากออบเจ็กต์ต่างๆที่ต้องการลงบน Scene แล้วจัดรูปแบบ Font, สี และอื่นๆ พร้อมทั้งลากออบเจ็กต์ที่เป็นรูปภาพที่ได้สร้างไว้แล้วมาวางบน Scene ดังรูปที่ 42

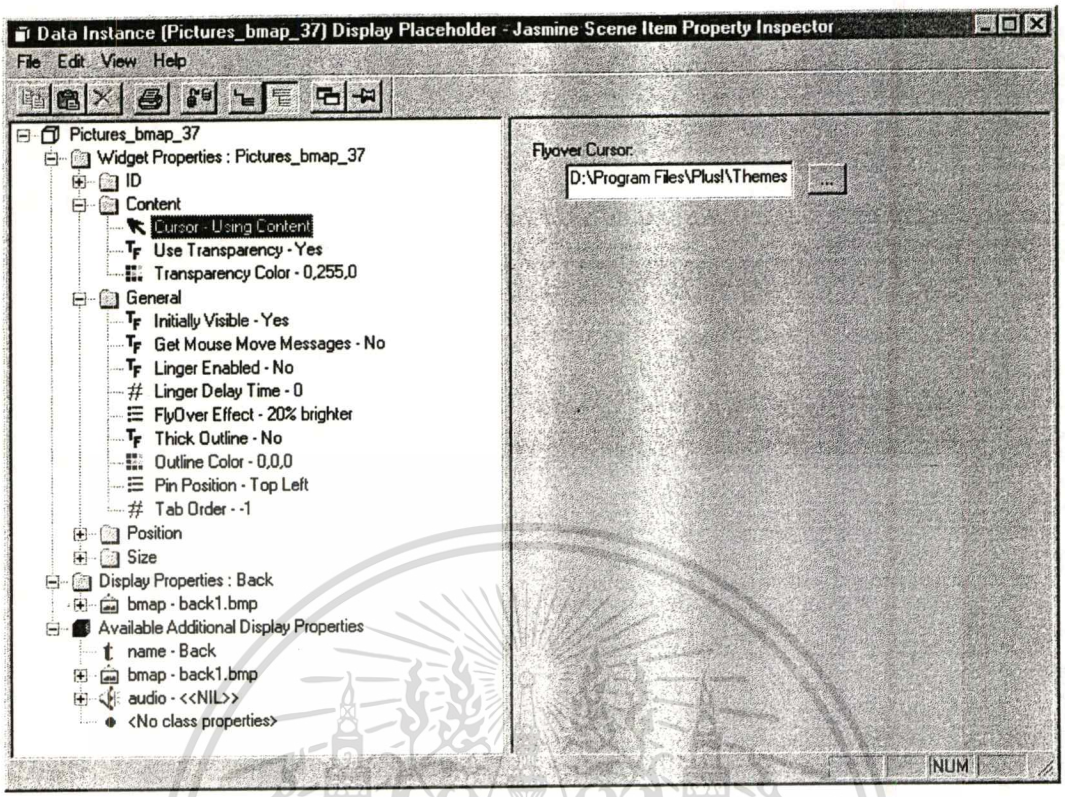
สำหรับการตกแต่งออบเจ็กต์ต่างๆให้ทำการเลือกออบเจ็กต์ที่ต้องการจัดรูปแบบก่อนแล้วคลิ้กเมาส์ขวาเลือก Property Inspector ซึ่งจะได้ Property ดังรูปที่ 43, 44



รูปที่ 42 แสดงโครงของหน้าจอ Registration Look Up

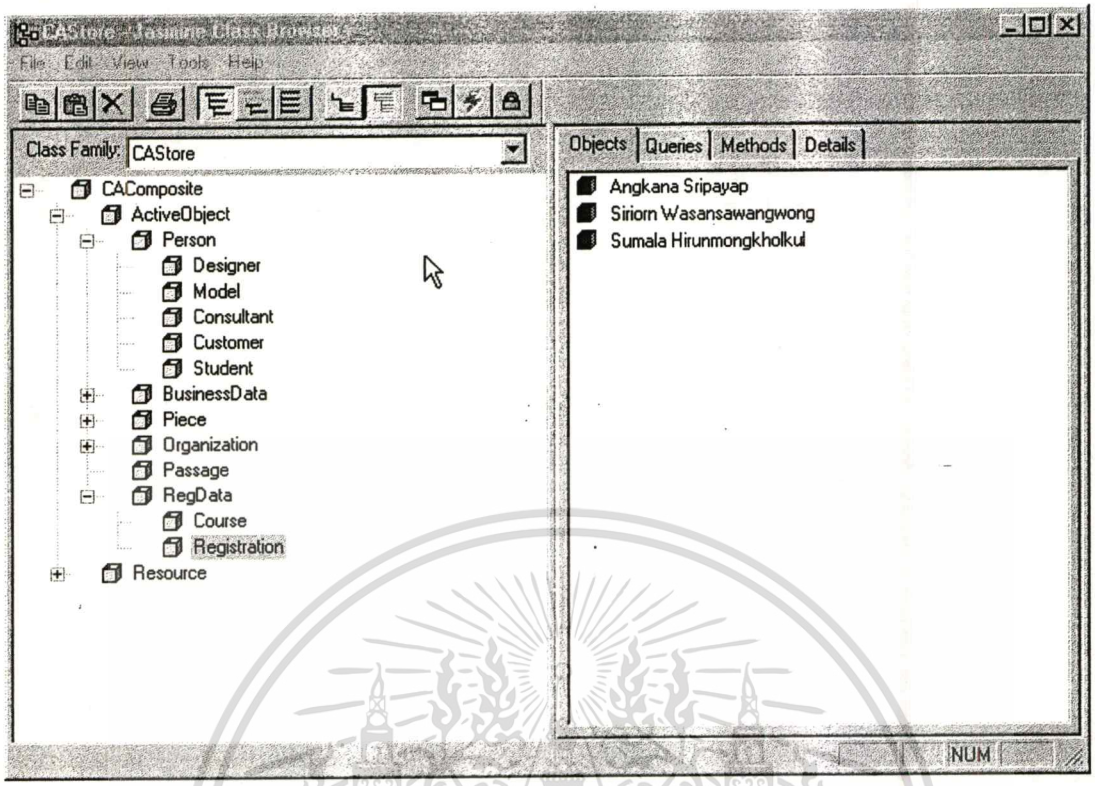


รูปที่ 43 แสดง Properties ของออบเจ็กต์ Static_Text_6 ที่อยู่ใน Class Family: <<Local Widget>>



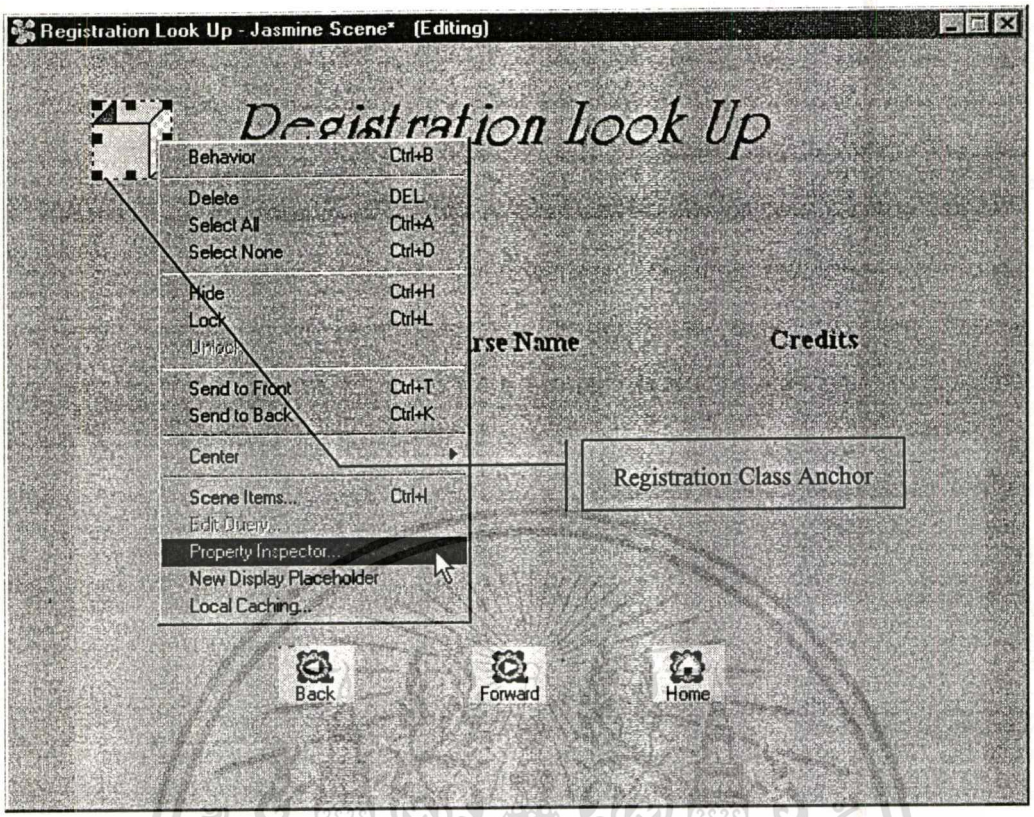
รูปที่ 44 แสดงหน้าต่าง Data Instance ของออบเจ็กต์รูปภาพ Back

Expand เครื่องหมาย + ของหน้าต่าง CAStore - Jasmine Class Browser จนพบ Registration ดังรูปที่ 45 ให้ลากคลาส Registration มาวางไว้บน Scene ซึ่งจะปรากฏรหัสของนักศึกษาขึ้นมา 1 ฟิลด์ (ฟิลด์ที่ปรากฏนี้เป็นฟิลด์ที่ได้จากการกำหนด Default Display ของออบเจ็กต์ตั้งแต่ตอนสร้างออบเจ็กต์)

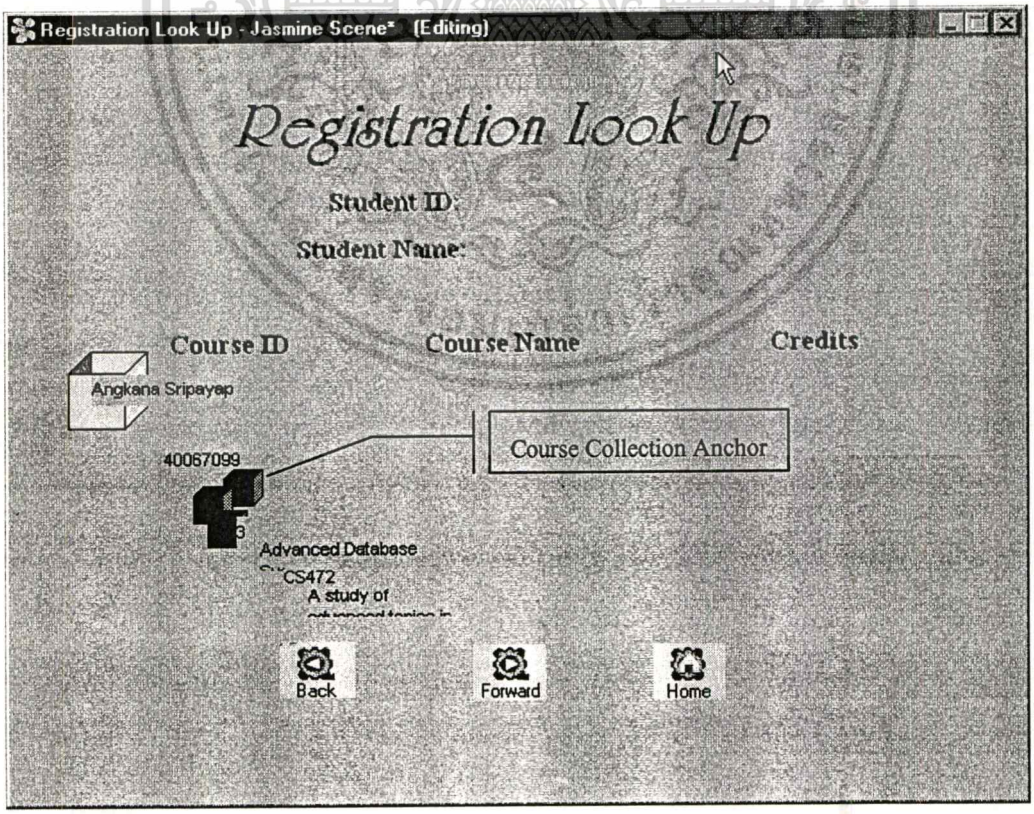


รูปที่ 45 แสดงหน้าต่าง CAStore - Jasmine Class Browser

คลิ้กขวาที่ Registration Class Anchor เลือก Property Inspector ดังแสดงในรูปที่ 46 จะปรากฏ Properties ดังรูปที่ 40 ให้ลาก theCourse ซึ่งเป็น Collection ของ CAStore::Course ออกมาไว้บน Scene จะปรากฏ Course Collection Anchor (หากต้องการให้แอตทริบิวต์แสดงบน Scene โดยที่แอตทริบิวต์นั้นมิได้กำหนดเป็น default display ของคลาสนั้น ก็สามารถทำได้โดยการลากแอตทริบิวต์ที่ต้องการออกมาวางบน Scene ได้เช่นเดียวกัน) ดังรูปที่ 47



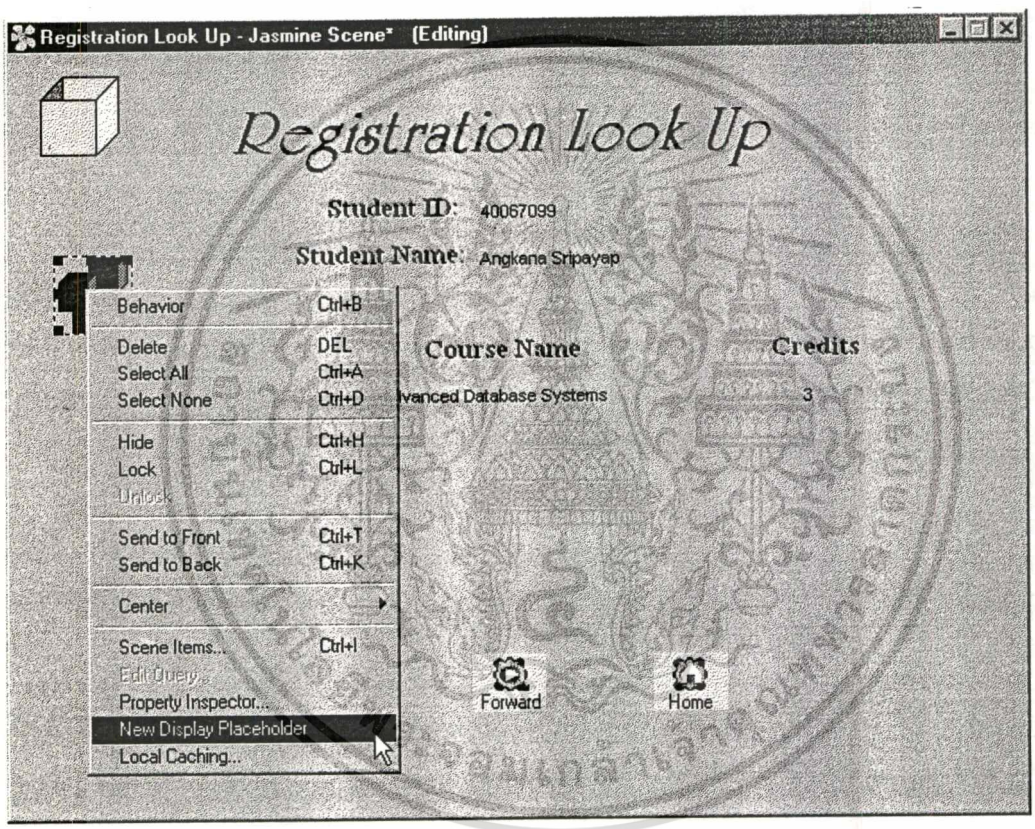
รูปที่ 46 แสดง Registration Class Anchor



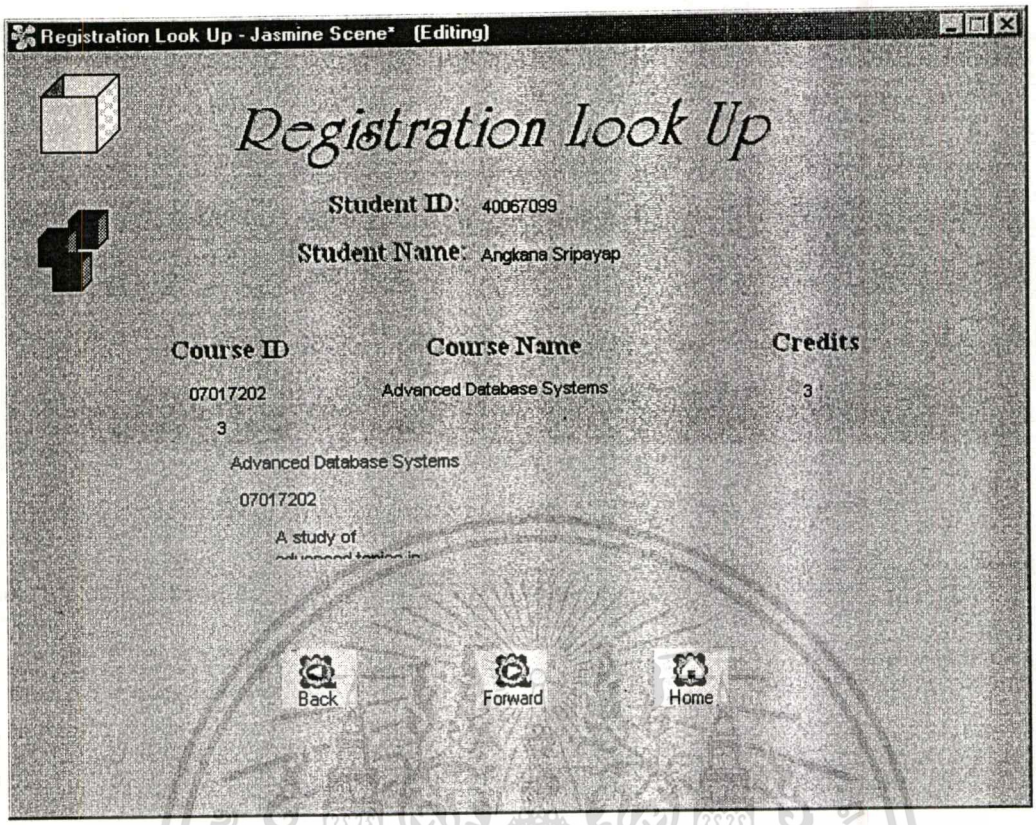
รูปที่ 47 แสดง Course Collection Anchor

เมื่อได้แอตทริบิวต์ต่างๆที่ต้องการแสดงบน Scene เรียบร้อยแล้วก็จัดแอตทริบิวต์เหล่านั้น พร้อมทั้งกำหนดรูปแบบ Font, สี และอื่นๆ

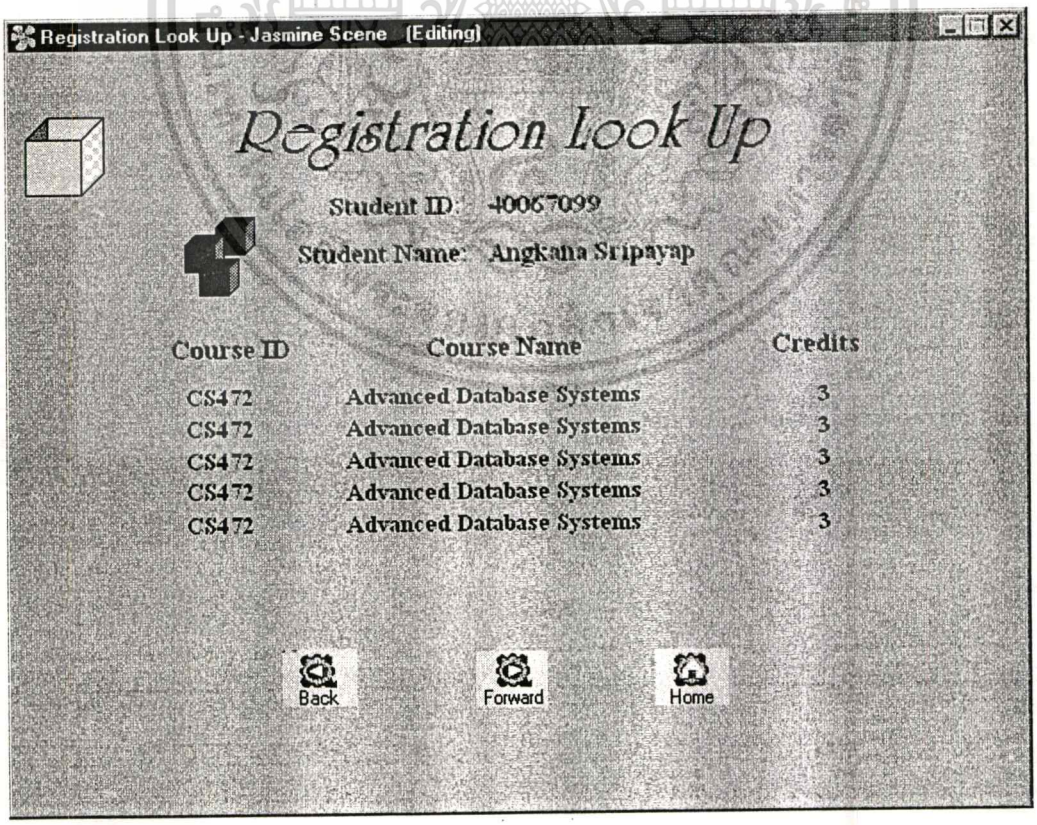
เนื่องจากรายวิชาที่มีการลงเบียนเรียนของนักศึกษาจะประกอบด้วยหลายรายวิชา (เป็น Collection) ดังนั้นหากต้องการแสดงรายวิชาเหล่านั้นมากกว่าหนึ่งวิชาให้เลือกที่ Course Collection Anchor คลิ้กเมาส์ขวาเลือก New Display Placeholder ดังแสดงในรูปที่ 48 หลังจากเลือก New Display Placeholder แล้วจะปรากฏดังรูปที่ 49 ในที่นี้จะกระทำการเลือก New Display Placeholder ทั้งหมด 4 ครั้ง แล้วจึงจัดรูปแบบให้เป็นระเบียบตามรูปที่ 50



รูปที่ 48 แสดงการเลือก New Display Placeholder

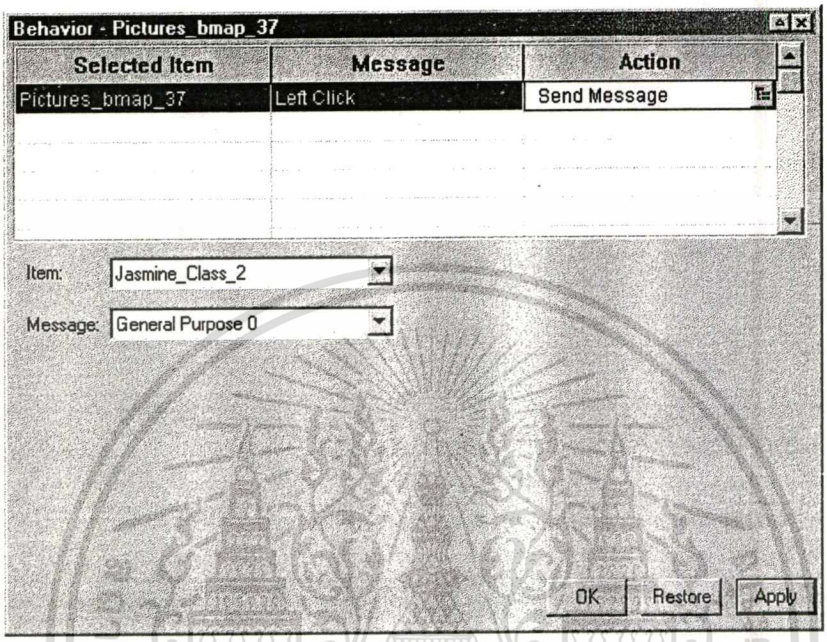


รูปที่ 49 แสดงหน้าจอที่ปรากฏหลังการเลือก New Display Placeholder



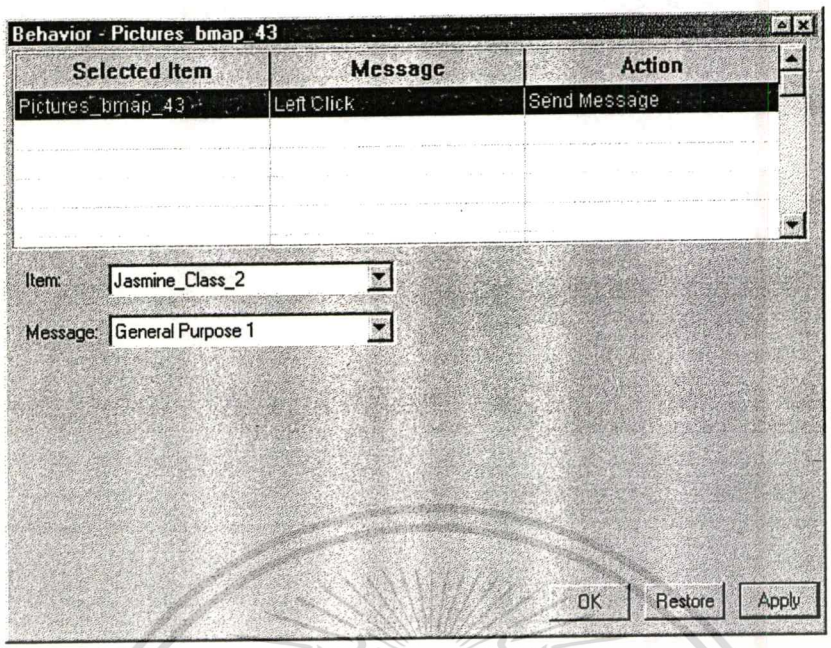
รูปที่ 50 แสดงหน้าจอที่ได้รับการจัดรูปแบบเรียบร้อยแล้ว

เมื่อจัดรูปแบบหน้าจอเรียบร้อยแล้วต่อไปจะเป็นการกำหนด Behavior ให้กับออบเจ็กต์ โดยเริ่มจากออบเจ็กต์ที่เป็นรูปภาพ Back โดยการเลือกออบเจ็กต์รูปภาพ Back คลิ้กเมาส์ขวาเลือก Behavior กำหนด Behavior ของรูปภาพ Back ดังรูปที่ 51



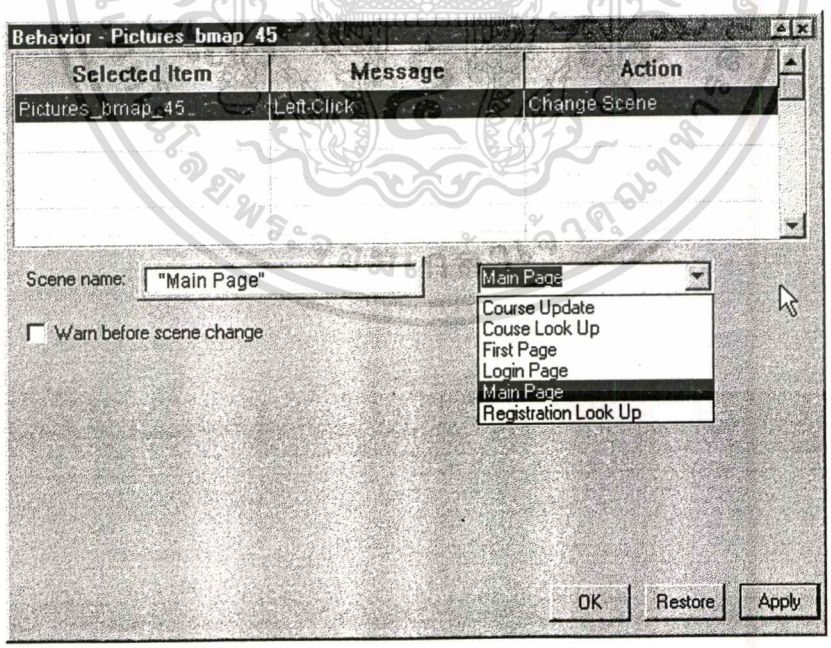
รูปที่ 51 แสดงการกำหนด Behavior ให้กับออบเจ็กต์รูปภาพ Back

จากรูปที่ 51 ช่อง Message ให้เลือก Mouse/Left Click ส่วน Action ให้เลือก Message Actions/Send Message สำหรับช่อง Item: ให้เลือก Jasmine_Class_2 (ตัวเลขนี้ขึ้นอยู่กับตอนพัฒนา) และ Message: เลือก General Purpose 0



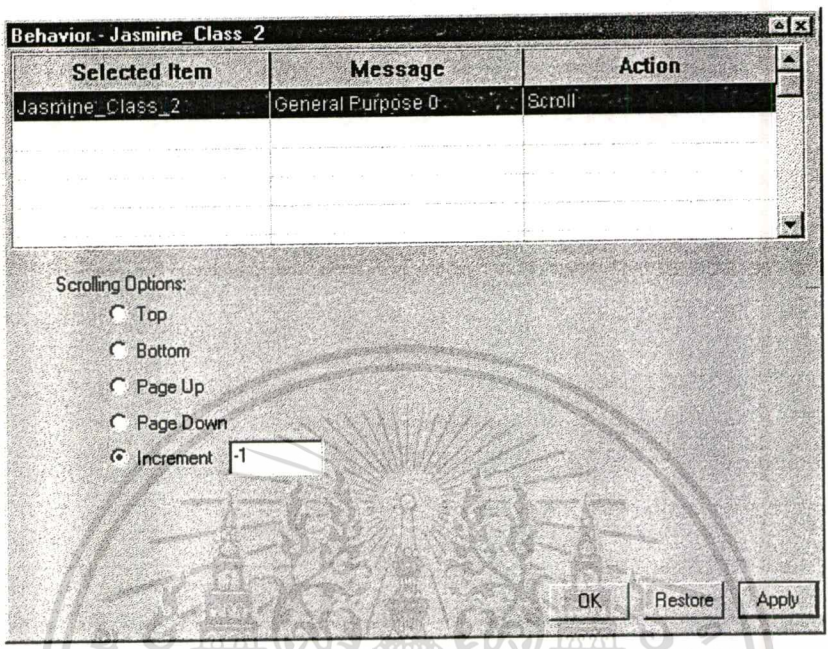
รูปที่ 52 แสดงการกำหนด Behavior ให้กับออบเจ็กต์รูปภาพ Next

จากรูปที่ 52 ช่อง Message ให้เลือก Mouse/Left Click ส่วน Action ให้เลือก Message Actions/Send Message สำหรับช่อง Item: ให้เลือก Jasmine_Class_2 (ตัวเลขนี้ขึ้นอยู่กับตอนพัฒนา) และ Message: เลือก General Purpose 1



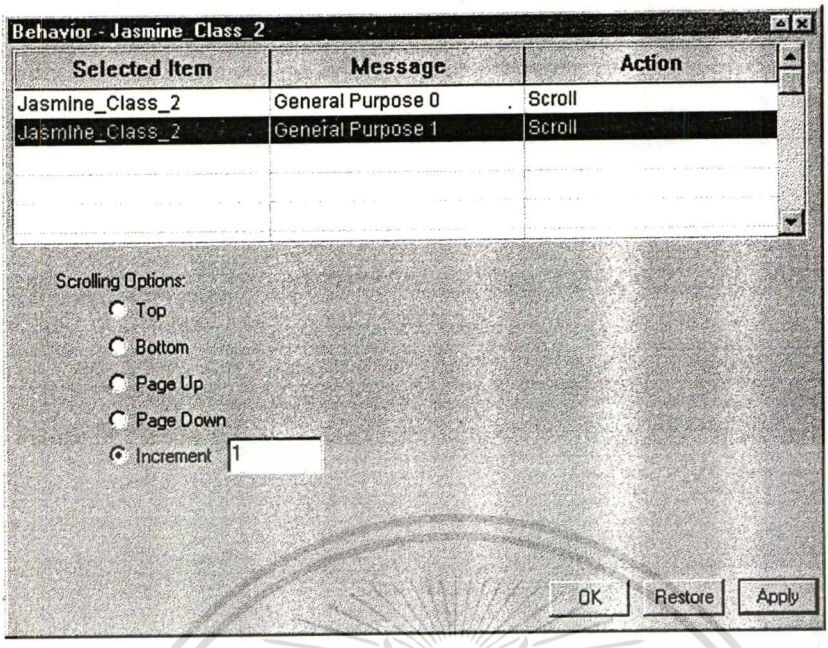
รูปที่ 53 แสดงการกำหนด Behavior ให้กับออบเจ็กต์รูปภาพ Home

จากรูปที่ 53 ช่อง Message ให้เลือก Mouse/Left Click ส่วน Action ให้เลือก Scene Actions/Change Scene แล้วเลือก Scene ที่ต้องการเปลี่ยนไป ในที่นี้ให้กลับไปที่ Scene Main Page



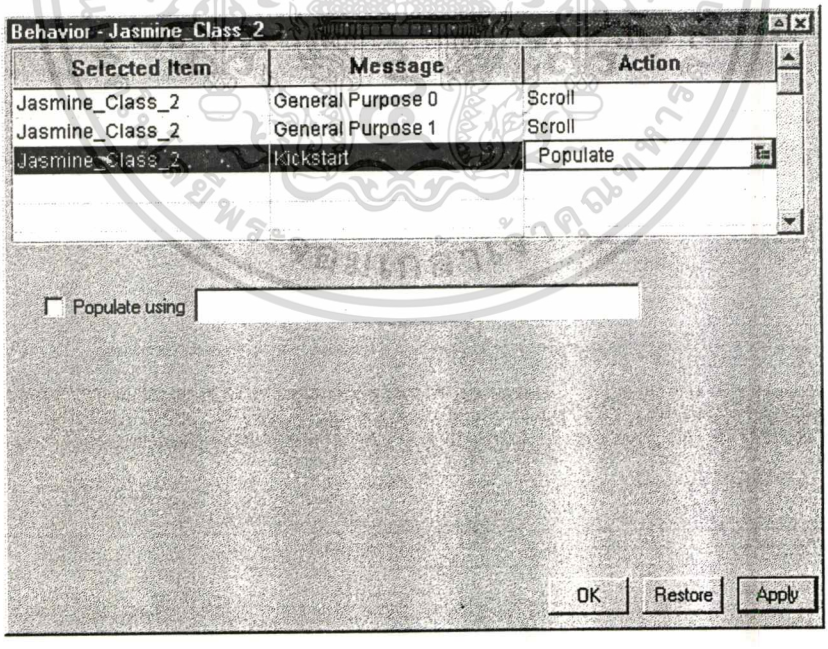
รูปที่ 54 แสดงการกำหนด Behavior ให้กับออบเจ็กต์ Registration Class Anchor เมื่อได้รับ Message: General Purpose 0 ซึ่งส่งโดยออบเจ็กต์รูปภาพ Back

จากรูปที่ 54 ช่อง Message ให้เลือก General Purpose/General Purpose 0 ส่วน Action ให้เลือก Jasmine Actions/Scroll และเลือก Scrolling Options: เป็น Increment –1



รูปที่ 55 แสดงการกำหนด Behavior ให้กับออบเจ็กต์ Registration Class Anchor เมื่อได้รับ Message: General Purpose 1 ซึ่งส่งโดยออบเจ็กต์รูปภาพ Next

จากรูปที่ 55 ช่อง Message ให้เลือก General Purpose/General Purpose 1 ส่วน Action ให้เลือก Jasmine Actions/Scroll และเลือก Scrolling Options: เป็น Increment 1



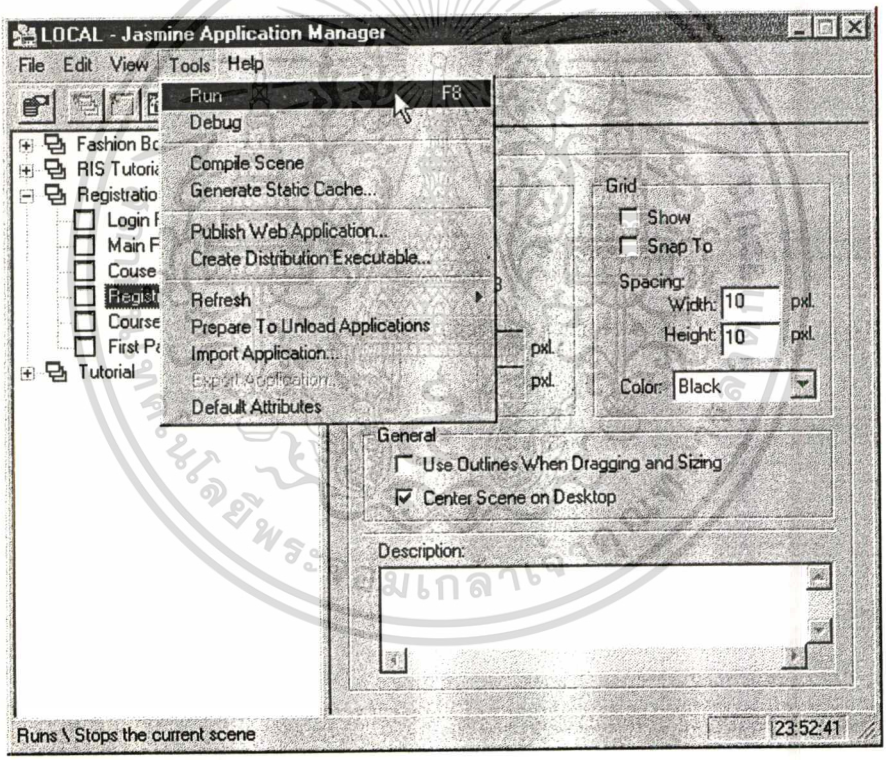
รูปที่ 56 แสดงการกำหนด Behavior ให้กับออบเจ็กต์ Registration Class Anchor เมื่อเริ่มต้นรัน Scene

จากรูปที่ 56 ช่อง Message ให้เลือก Begin/End แล้วเลือก Kickstart ส่วน Action ให้เลือก Jasmine Actions/Populate

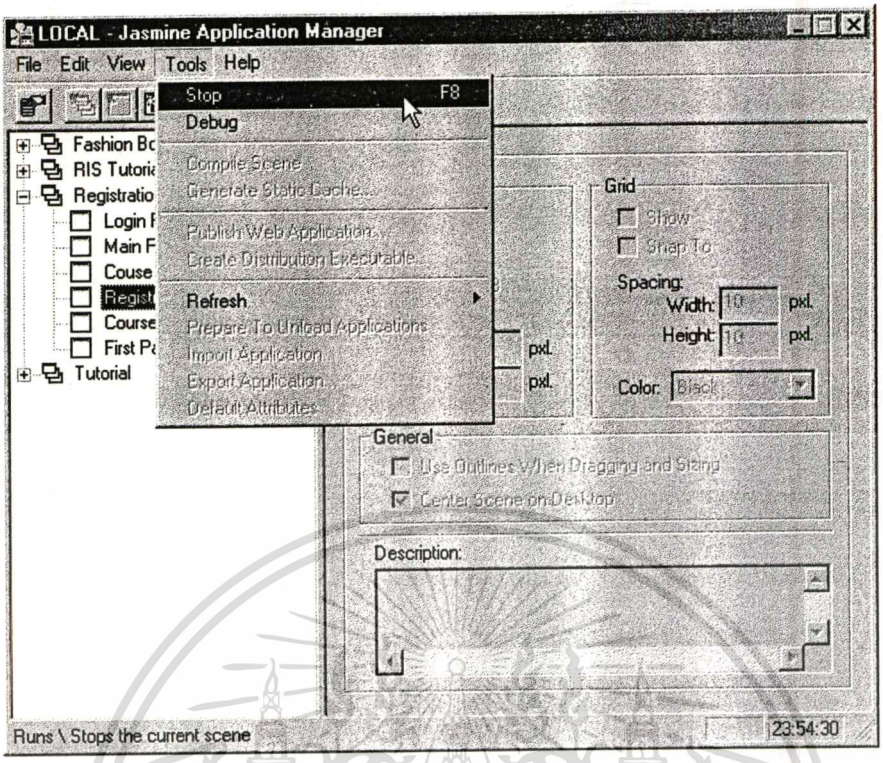
สำหรับ Scene อื่นๆ ในที่นี้จะไม่ขอกล่าวถึงขั้นตอนในการทำ แต่ผู้ที่สนใจสามารถศึกษาได้จาก Concepts, GS, ActiveX, C API ซึ่งเป็นไฟล์เอกสารการใช้งานต่างๆของ Jasmine รวมทั้ง Online Help ได้จาก CD หรือสามารถหาข้อมูลเพิ่มเติมได้ที่ http://www.cai.com

7.4 การรันและการสร้างเอ็กซีคิวต์ไฟล์ด้วย Jasmine Studio

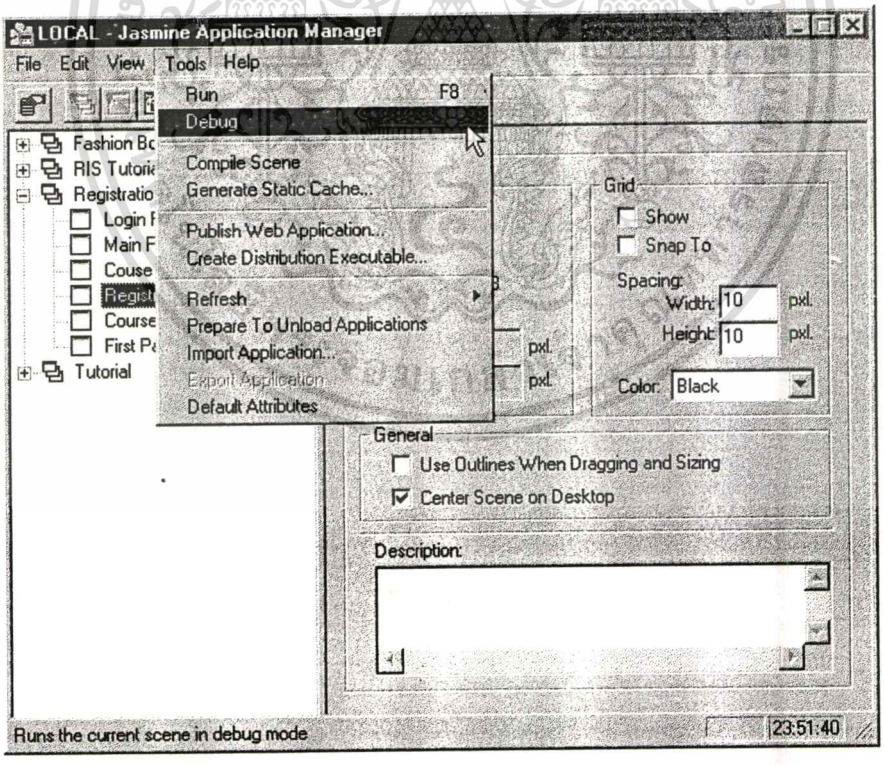
เลือก Scene ที่ต้องการคอมไพล์และรัน เลือกเมนู Tools/Run ดังแสดงตามรูปที่ 57 เมื่อต้องการหยุดการรัน ให้เลือกเมนู Tools/Stop ดังรูปที่ 58 (ถ้าต้องดีบักโปรแกรมให้เลือก Tools/Debug จะได้หน้าต่างดังรูปที่ 59 และ 60)



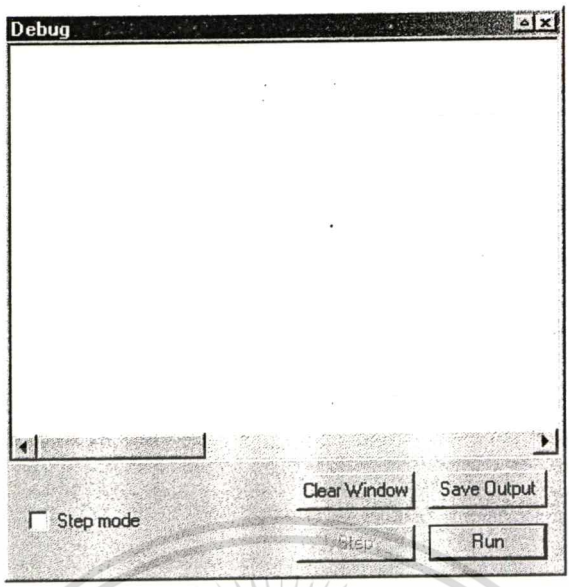
รูปที่ 57 แสดงการรัน Scene



รูปที่ 58 แสดงการหยุดการรัน Scene

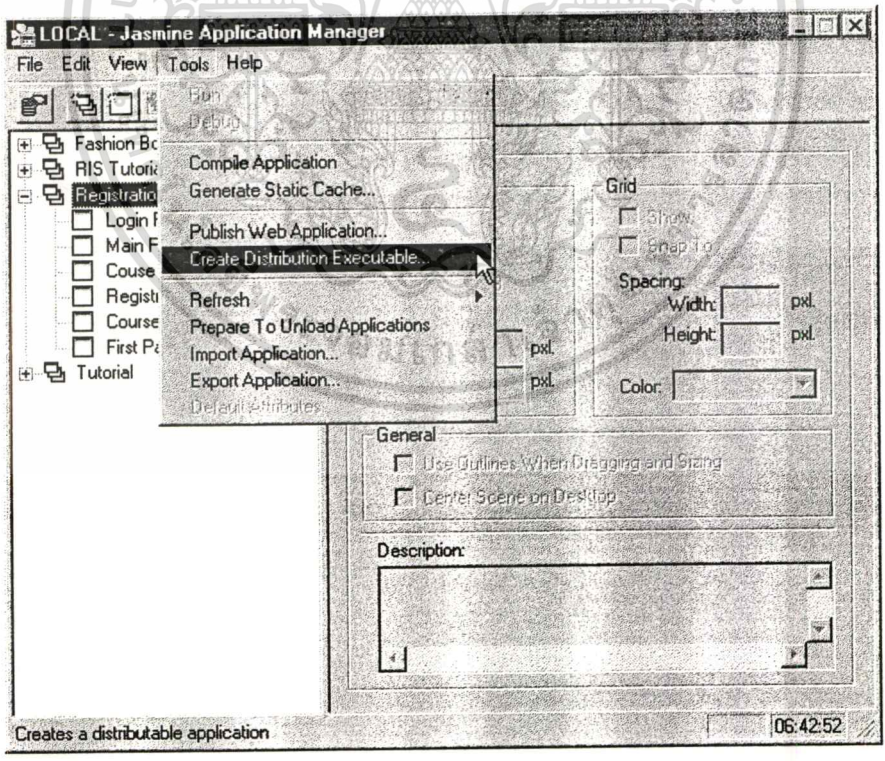


รูปที่ 59 แสดงหน้าต่างการเลือกทำงาน Debug



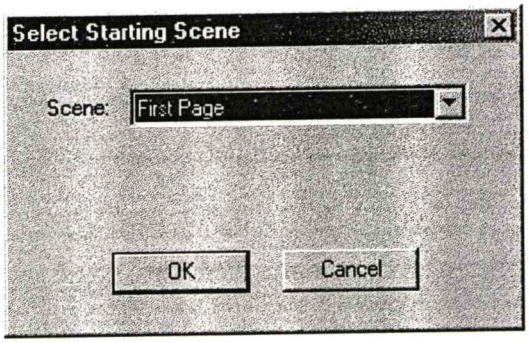
รูปที่ 60 แสดงหน้าต่าง Debug

สำหรับการสร้างเอ็กซีคิวต์ไฟล์มีขั้นตอนการทำคือ เลือกเมนู Tools/Create Distribution Executable ดังรูปที่ 61



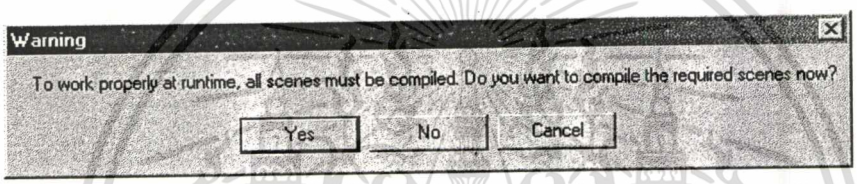
รูปที่ 61 แสดงการสร้างเอ็กซีคิวต์ไฟล์

จากนั้นเลือก Scene ที่จะเป็น Scene เริ่มต้นของแอปพลิเคชั่นดังรูปที่ 62



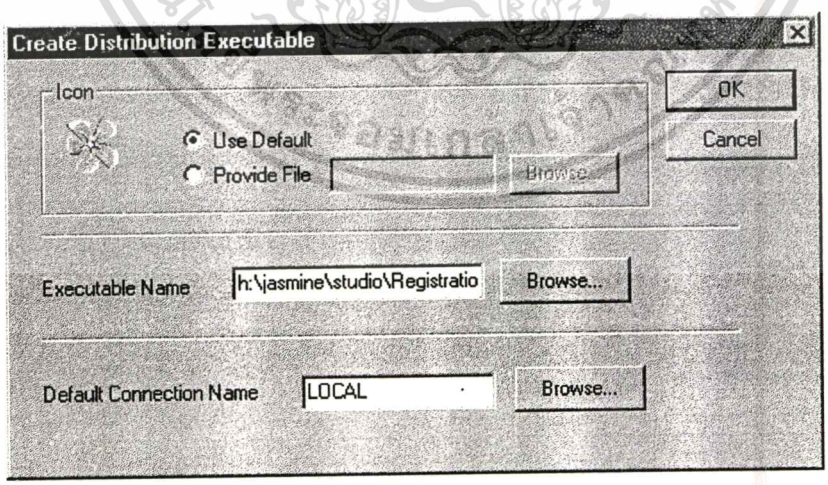
รูปที่ 62 แสดงหน้าต่าง Select Starting Scene

เลือก First Page เป็น Scene เริ่มต้นกดปุ่ม OK จะปรากฏหน้าต่าง Warning เพื่อรอการยืนยันการคอมไพล์ Scene ดังรูปที่ 63

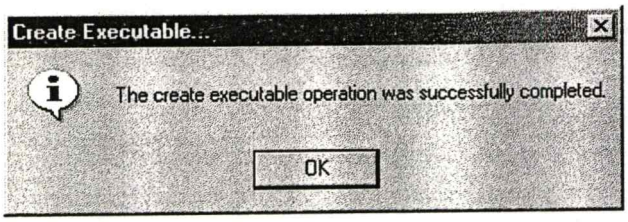


รูปที่ 63 แสดงหน้าต่าง Warning เพื่อรอการยืนยันการคอมไพล์

เมื่อกด Yes ยืนยันการคอมไพล์จะปรากฏหน้าต่าง Create Distribution Executable ดังรูปที่ 64 ใส่ชื่อเอ็กซีคิวต์ไฟล์และการเชื่อมต่อ เมื่อกด OK จะปรากฏหน้าต่างดังรูปที่ 65 เพื่อบอกว่าการสร้างเอ็กซีคิวต์ไฟล์ได้ทำเสร็จเรียบร้อยแล้ว



รูปที่ 64 แสดง Create Distribution Executable



รูปที่ 65 แสดงหน้าต่างยืนยันการสร้างเอ็กซีคิวต์ไฟล์เป็นที่เรียบร้อย

บทที่ 8

แนวทางในการแก้ไขปัญหา

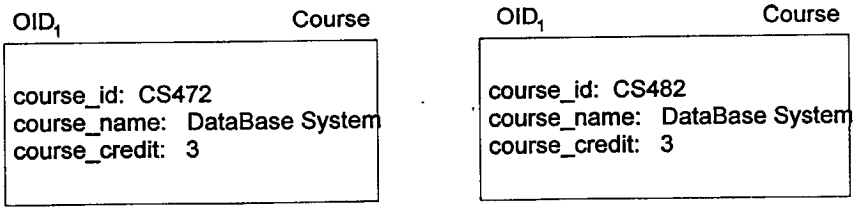
การใช้ค่าหรือกลุ่มของค่าของแอตทริบิวต์ในการแสดงเอกลักษณ์ของรีเลชั่น (Identifier Key) ที่ใช้ในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์จะพบกับปัญหาต่างๆ ดังที่ได้นำเสนอไว้ในบทที่ 5 สำหรับการศึกษาในบทนี้จะเป็นแนวทางในการแก้ไขปัญหาดังกล่าวด้วยแนวทางของออบเจ็กต์โอเรียนเต็ดและระบบจัดการฐานข้อมูลเชิงวัตถุ Jasmine

8.1 แนวทางในการแก้ไขปัญหา Identifier Key ตามแนวทางของระบบการจัดการฐานข้อมูลเชิงวัตถุ

8.1.1 แนวทางในการแก้ไขปัญหา Modifying Identifier Keys (ปัญหาที่ 5.1)

สำหรับระบบการจัดการฐานข้อมูลเชิงวัตถุ ซึ่งสนับสนุน Object Identity Concept (จากบทที่ 4) จะมี Object Identifier (OID) ซึ่งระบบเป็นผู้สร้างหรือกำหนดให้เองตั้งแต่เริ่มสร้างออบเจ็กต์นั้นขึ้นมา และเนื่องจากคุณสมบัติของ OID ซึ่งไม่ขึ้นกับค่าหรือ content ของออบเจ็กต์ ทำให้การเปลี่ยนแปลงค่าของแอตทริบิวต์ต่างๆภายในออบเจ็กต์ไม่มีผลต่อการอ้างถึงอีกออบเจ็กต์หนึ่ง การเปลี่ยนแปลง Content ในออบเจ็กต์จะกระทำผ่าน Method ของออบเจ็กต์นั้นๆโดยไม่ส่งผลถึง OID ที่ใช้ในการอ้างอิงถึงออบเจ็กต์ดังกล่าว ทำให้ระบบการจัดการฐานข้อมูลเชิงวัตถุไม่จำเป็นต้องมีการดูแลเรื่อง Referential Integrity เหมือนในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์

ตัวอย่างเช่นหากมีการเปลี่ยนแปลงรหัสวิชา CS472 ให้เป็น CS 482 แต่รายละเอียดอื่นๆยังคงเหมือนเดิม ดังรูปที่ 66 ออบเจ็กต์ที่อ้างอิงถึงออบเจ็กต์ของวิชานี้ ก็ยังคงสามารถอ้างถึงออบเจ็กต์ตัวนี้ด้วย OID_1 ได้เช่นเดิม



รูปที่ 66 แสดงออบเจ็กต์ที่มีการเปลี่ยนแปลง attribute value แต่ OID คงเดิม

สำหรับระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ ถ้ากำหนดให้ Course_ID เป็น identifier key ของรีเลชั่น COURSE ดังรูปที่ 67 ถ้ารหัสวิชา CS472 ถูกเปลี่ยนเป็น CS482 จะมีผลทำให้ต้องมีแก้ไขค่าแอตทริบิวต์ของรีเลชั่นอื่นๆที่ได้อ้างถึง CS472 ในลักษณะของ foreign key ตัวอย่างเช่น จากรูปที่ 68 รีเลชั่น STD_COURSE_REG ซึ่งมี (Student_ID, Course_ID, Reg_Semester) เป็น identifier key และมี Course_ID ทำหน้าที่เป็น foreign key เพื่ออ้างอิงไปยัง Course_ID ซึ่งเป็นคีย์หลักที่อยู่ในรีเลชั่น COURSE หากมีการเปลี่ยนแปลงรหัสวิชาดังกล่าวเกิดขึ้น Course_ID ที่อยู่ในรีเลชั่น STD_COURSE_REG ต้องได้รับการแก้ไข โดยจะมีการเปลี่ยนแปลงแก้ไขในทุกๆ ทูเปิลของรีเลชั่น STD_COURSE_REG ที่มี Course_ID = CS472 ให้เป็น CS482 ทั้งหมด

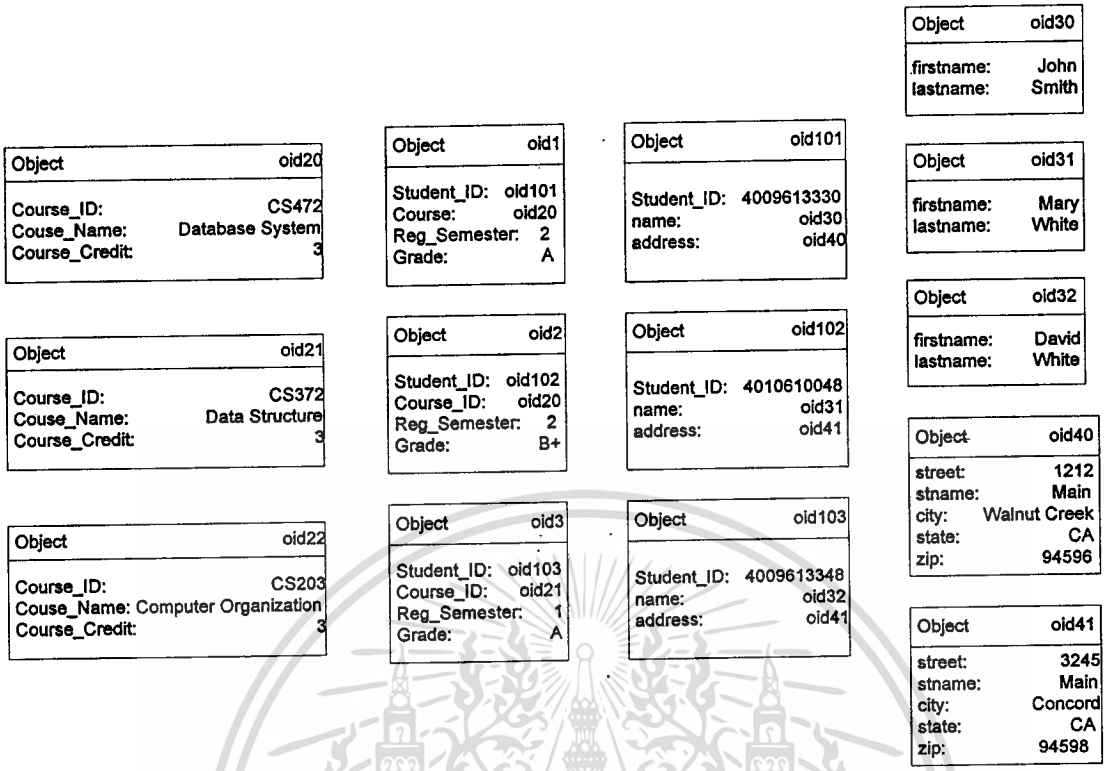
สำหรับระบบการจัดการฐานข้อมูลเชิงวัตถุจะมีการเก็บข้อมูลเป็นแบบออบเจ็กต์ ดังรูปที่ 69 เมื่อมีการเปลี่ยนแปลงจะสามารถกระทำได้ภายในตัวออบเจ็กต์นั้น โดยที่ OID ไม่มีการเปลี่ยนแปลง ทำให้ในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ทุกๆระบบจำเป็นต้องมี Referential Integrity Rule เพื่อควบคุมความถูกต้องของข้อมูล และเป็นสิ่งที่ต้องมี ถ้าไม่มีจะทำให้ข้อมูลที่อยู่ในระบบมีความผิดพลาดได้

Course_ID	Course_Name	Course_Credit
CS472	Database System	3
CS203	Computer Organization	3
CS372	Data Structure	3

รูปที่ 67 แสดงรีเลชั่น COURSE

Student_ID	Course_ID	Reg_Semester	Grade
4009613330	CS472	2	A
4010610048	CS472	2	B+
4009613348	CS372	1	A

รูปที่ 68 แสดงรีเลชั่น STD_COURSE_REG



รูปที่ 69 แสดงการเก็บข้อมูลแบบออบเจ็กต์โอเรียนเต็ด

8.1.2 แนวทางในการแก้ไขปัญหา Nonuniformity (ปัญหาที่ 5.2)

ปัญหานี้ ระบบการจัดการฐานข้อมูลเชิงวัตถุจะใช้ Object Identifier (OID) ในการแสดงเอกลักษณ์ของแต่ละออบเจ็กต์ ซึ่งมีคุณสมบัติเป็น unique identifier ที่ระบบเป็นผู้สร้างขึ้นเอง ผู้พัฒนาระบบไม่จำเป็นต้องสร้างหรือกำหนดเองเหมือนกับในระบบเชิงสัมพันธ์ ทำให้เกิดความสะดวกในการพัฒนาแอปพลิเคชั่นเนื่องจากโปรแกรมเมอร์ไม่ต้องเกี่ยวข้องกับการคัดเลือกคีย์ที่เหมาะสมสำหรับหลายๆคลาสของออบเจ็กต์ (ใน Relational Model หนึ่งรีเลชั่นอาจมี candidate key มากมาย ซึ่งโปรแกรมเมอร์ต้องเลือกว่าตัวใดเหมาะสมที่สุดที่จะนำมาใช้เป็น primary key)

OID เป็น แนวคิดในระดับ Logical บางครั้งเรียกว่า Logical Object Identifier (LOID) ใช้ในการแสดงเอกลักษณ์ของแต่ละออบเจ็กต์ โดยที่การอิมพลีเมนต์ OID อาจทำได้หลายรูปแบบซึ่งจะมีระดับความเป็น physical มากน้อยเพียงไรขึ้นอยู่กับแต่ละ OO-DBMS และ OID จะมีขนาดเป็น fixed-length และมีรูปแบบเดียวกันทั้งฐานข้อมูลทำให้ไม่เกิดปัญหา Nonuniformity ดังเช่นที่พบใน RDBMS กล่าวคือ บางรีเลชั่นอาจใช้ identifier key เป็น Integer, characters หรือ meaningful digit เป็นต้น ทำให้ขาด uniformity ไป

ตัวอย่างของ LOID (ใน Versant, Object - Oriented Database Management System) ประกอบด้วยตัวเลข 64-bit ซึ่งแบ่งเป็น 2 ส่วน คือ

- ส่วนแรกเป็นตัวเลข 16-bit ที่แสดงถึงเลขของ Database ที่ไม่ซ้ำกัน
- ส่วนที่สองเป็นตัวเลข 48-bit ที่แสดงถึงเลข ID ของ Object นั้น โดยใน 48-bit นี้จะแสดงแยกเป็น 2 ส่วน คือ ส่วนแรกเป็น 16-bit (ส่วนมากเป็น 0) ส่วนที่สองเป็นเลข 32-bit ซึ่งแสดง Object ID

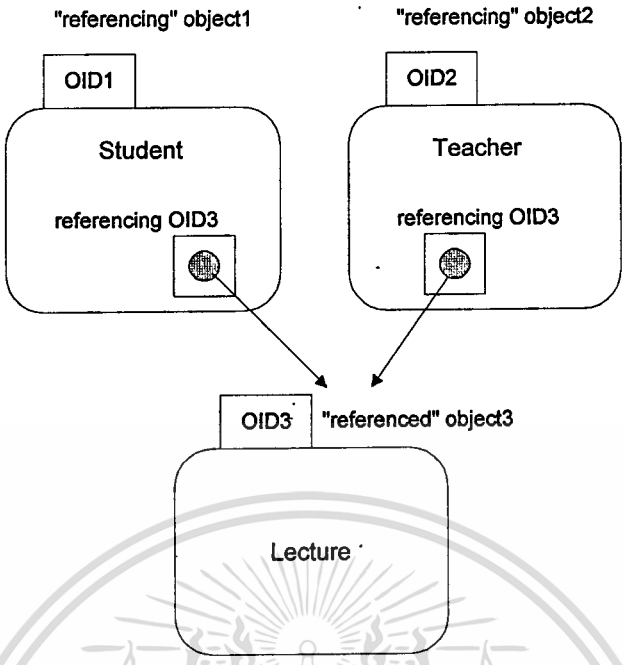
8.1.3 แนวทางในการแก้ไขปัญหา "Unnatural" Join (ปัญหาที่ 5.3)

เนื่องจากชนิดของข้อมูลในระบบการจัดการฐานข้อมูลเชิงสัมพันธ์เป็นแบบ Atomic Type หากในแอตทริบิวต์ใดไม่เป็น Atomic จะต้องทำให้เป็น Atomic เสียก่อน อันเป็นที่มาของการทำนอร์มอลไลซ์เซชั่น เพื่อลดความซ้ำซ้อน และด้วยเหตุนี้ทำให้บางครั้งต้องมีการกระจายข้อมูลของออบเจ็กต์เดียวกันออกไปไว้กับออบเจ็กต์อื่นๆ ทำให้ขาด Semantic ไป

การ Join กันทำให้เกิดความยุ่งยากในการสร้างออบเจ็กต์ที่มีความซับซ้อน สำหรับระบบการจัดการฐานข้อมูลเชิงวัตถุที่สนับสนุนแนวคิดของ Object Identity จะสามารถสร้าง Complex Object ได้ง่ายไม่ต้องมีการ Join กันเพราะจะใช้คอนสตรักเตอร์ในการสร้างออบเจ็กต์ที่มีความซับซ้อนเหล่านี้แทนการ Join

Complex-Object Model ถูกแสดงออกมาอย่างชัดเจนในการใช้ออบเจ็กต์ร่วมกันระหว่างออบเจ็กต์อื่นๆที่ต่างกัน โดยจะใช้ OID ในการอ้างถึงออบเจ็กต์ที่เกี่ยวข้องกัน ทำให้เกิดความสามารถในการใช้ออบเจ็กต์ร่วมกัน (Shared Object)

ตัวอย่าง ฐานข้อมูลโรงเรียนโดยใช้ OID ในการสร้าง Complex Object



รูปที่ 70 แสดงแนวคิดในการสร้าง Complex Object โดยใช้ OID

จากรูปที่ 70 กำหนดให้ฐานข้อมูลของโรงเรียนมีลักษณะดังนี้

```
Student ::=   [ name:       string,
              grade:        int,
              adviser:      Teacher,
              lectures:     {Lecture} ]
Teacher ::=   [ name:       string,
              phone:        string,
              lectures:     {Lecture} ]

Lecture ::=   [ name:       string,
              teacher:      Teacher,
              students:     {Student},
              credit:       int ]
```

ระบบฐานข้อมูลโรงเรียนประกอบด้วยออบเจ็กต์ 3 ชนิด ได้แก่ Teacher, Lecture และ Student โดยที่ Student Type มีแอตทริบิวต์ได้แก่ name, grade, adviser และ lectures โดยที่

name และ grade เป็นชนิด String และ Integer ตามลำดับ ซึ่งเป็น Atomic Type ส่วนแอตทริบิวต์ adviser เป็น OID ของทูเปิลออบเจ็กต์ของ Teacher Type ซึ่งหมายถึงว่าออบเจ็กต์ Student อ้างถึงออบเจ็กต์ Teacher และ แอตทริบิวต์ lecturers เป็น OID ของเซตออบเจ็กต์ซึ่งมีสมาชิกเป็น OID ของทูเปิลออบเจ็กต์ของ Lecture Type ซึ่งหมายความว่าออบเจ็กต์ Student อ้างถึงเซตของออบเจ็กต์ Lecture สำหรับ Teacher และ Lecture types สามารถอธิบายได้ในทำนองเดียวกัน ตัวอย่างของอินสแตนส์ของ Complex Object สำหรับออบเจ็กต์ทั้งสามชนิดนี้เป็นดังนี้

(Student, s1, ["Tom", 3, t1, {l1, l3}])

(Student, s2, ["Mike", 3, t2, {l1, l2, l3}])

(Student, s3, ["Mary", 4, t1, {l1, l2}])

...

(Teacher, t1, ["Lorie", "812-2111", {l1, l2}])

(Teacher, t2, ["Smith", "814-4111", {l3}])

...

(Lecture, l1, ["DBMS", t1, {s1, s2, s3}, 3])

(Lecture, l2, ["OS", t1, {s2, s3}, 3])

(Lecture, l3, ["Language", t2, {s1, s2}, 3])

...

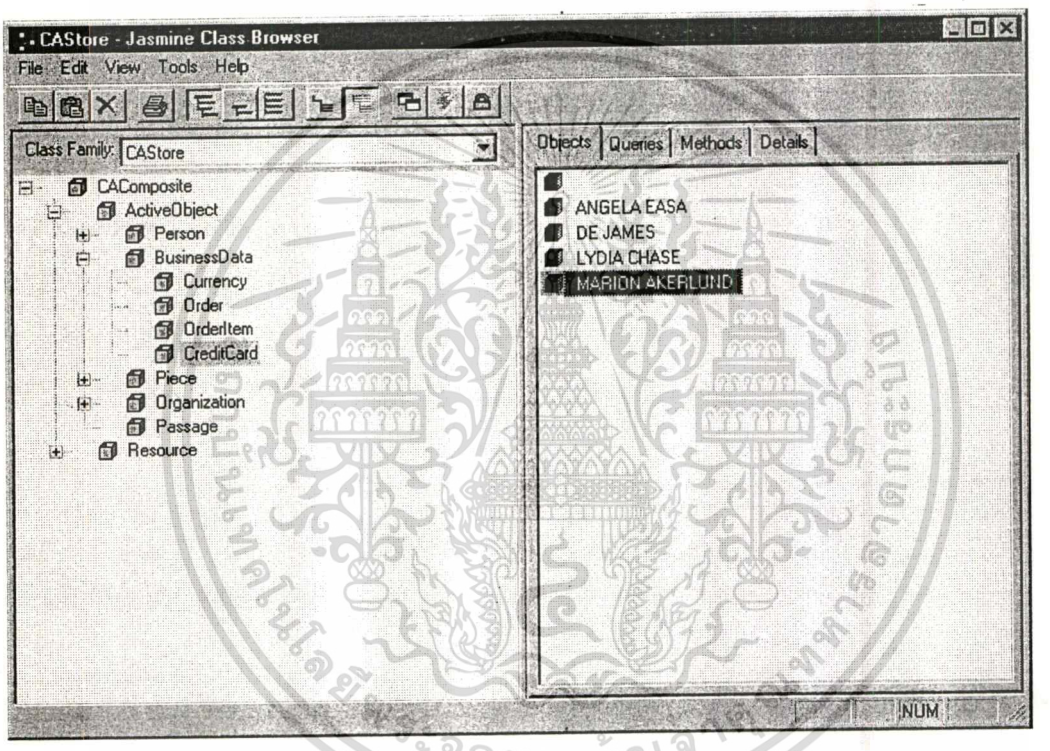
จากตัวอย่างอินสแตนส์ข้างต้น (Student, s1, ["Tom", 3, t1, {l1, l3}]) Student เป็นการกำหนดว่าออบเจ็กต์นี้เป็น Student Type และ s1 เป็น OID ที่กำหนดให้กับออบเจ็กต์นี้ ส่วน ["Tom", 3, t1, {l1, l3}] เป็น Tuple Value ของออบเจ็กต์ ซึ่งมี 4 แอตทริบิวต์ "Tom" เป็นค่าสตริงสำหรับแอตทริบิวต์ name, 3 เป็นค่าจำนวนเต็มสำหรับแอตทริบิวต์ grade, t1 จะอ้างถึงออบเจ็กต์ Teacher ซึ่ง OID เป็น t1 และ l1,l3 เป็น set value สำหรับแอตทริบิวต์ lectures ซึ่งกำลังอ้างสองออบเจ็กต์ซึ่งเป็น Lecture Type โดยที่ OID ของสองออบเจ็กต์นั้นคือ l1 และ l3

8.2 แนวทางในการแก้ไขปัญหา Identifier Key โดย Jasmine (OO-DBMS)

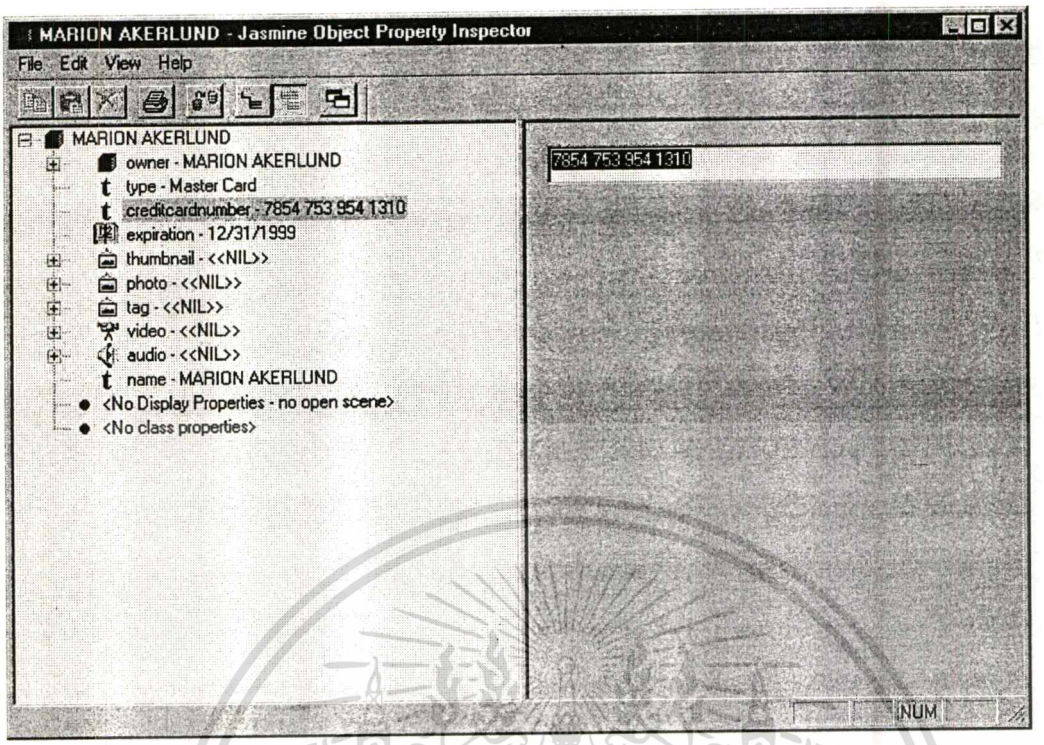
8.2.1 แนวทางในการแก้ไขปัญหา Modifying Identifier Keys (ปัญหาที่ 5.1)

ระบบจะสร้าง OID ขึ้นมาทันทีที่สร้างออบเจ็กต์หรืออินสแตนส์ขึ้นมา โดยที่แต่ละออบเจ็กต์จะมีคุณสมบัติหรือแอตทริบิวต์ต่างๆถ่ายทอดมาจากคลาส เราสามารถเปลี่ยนแปลงค่าของแอตทริบิวต์ที่ปกติเป็นจะใช้เป็น Identifier Key ในระบบฐานข้อมูลเชิงสัมพันธ์ได้ เพราะการเปลี่ยนแปลงค่าในส่วนนี้เป็นเพียงการเปลี่ยนค่าของ Property มิใช่เป็นการเปลี่ยนแปลง Identifier

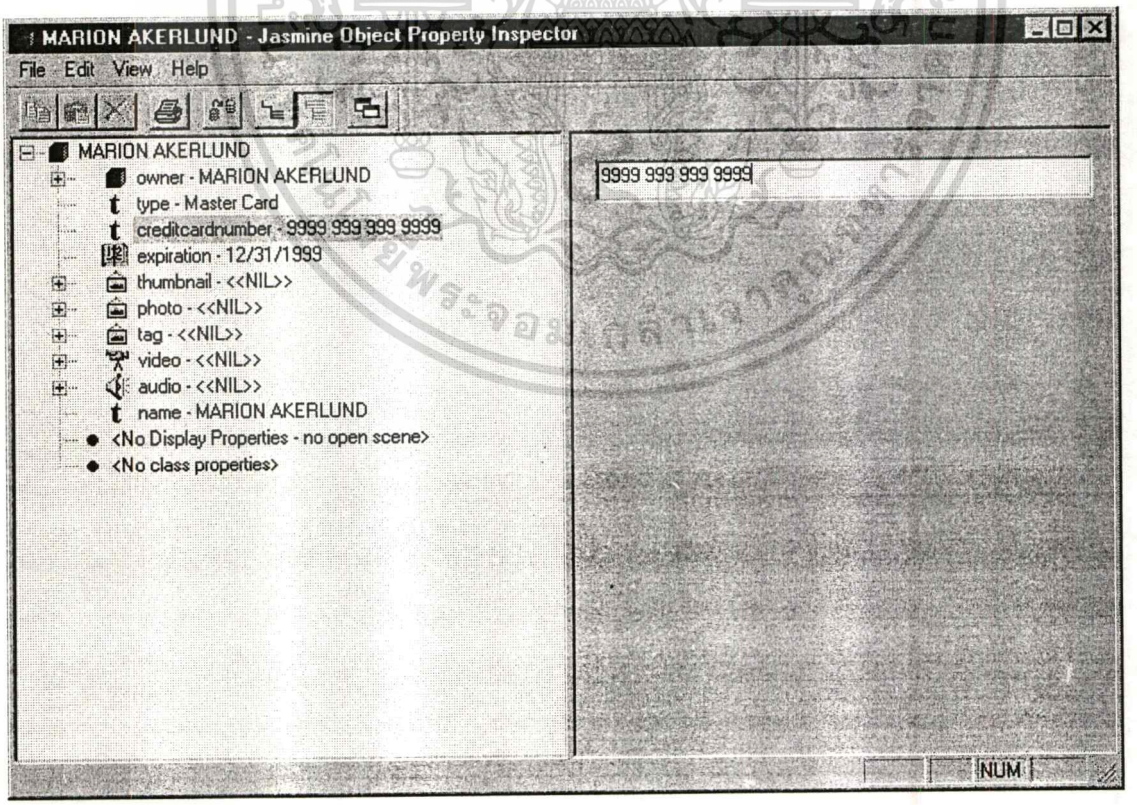
ตัวอย่างเช่น ออบเจ็กต์ CreditCard ดังรูปที่ 71 จะมีอินสแตนส์ของออบเจ็กต์ CreditCard อยู่ 4 อินสแตนส์ แอตทริบิวต์หรือ Property ของออบเจ็กต์ CreditCard เป็นดังรูปที่ 72 ซึ่งจะมีแอตทริบิวต์ที่ชื่อ creditcardnumber ปรากฏอยู่ โดยทั่วไปแล้วถ้าเป็นระบบฐานข้อมูลเชิงสัมพันธ์มักใช้แอตทริบิวต์เป็น Identifier Key เมื่อมีการแก้ไขจะต้องแก้ไขให้ครบทุกๆรีเลชั่นที่อ้างถึง แต่สำหรับ Jasmine หากแก้ไขค่าข้อมูลในแอตทริบิวต์นี้จาก 7854 753 954 1310 เป็น 9999 999 999 9999 ดังรูปที่ 73 จะสามารถทำได้โดยไม่กระทบต่อออบเจ็กต์อื่นๆ เพราะเป็นเพียงการแก้ Property ภายในออบเจ็กต์ของ MARION AKERLUND เท่านั้น มิได้เปลี่ยนแปลง OID



รูปที่ 71 แสดงอินสแตนส์ของออบเจ็กต์ CreditCard



รูปที่ 72 แสดง Property ของ ออบเจ็กต์ MARION AKERLUND



รูปที่ 73 แสดง Property ของออบเจ็กต์ MARION AKERLUND ที่เปลี่ยนค่าหมายเลขบัตรเครดิต

8.2.2 แนวทางในการแก้ไขปัญหา Nonuniformity (ปัญหาที่ 5.2)

ใน Jasmine OID จะเป็น Logical ซึ่งไม่ได้ประกอบขึ้นมากจาก physical addresses ใน secondary memory แต่ OID ของ Jasmine จะประกอบด้วย database id-no, class id-no, และ instance id-no ทำให้ขจัดปัญหา Nonuniformity ไปได้

สำหรับ Jasmine จะจัดหามาโครที่ใช้ใน Embedded ODQL (Object Database Query Language) ซึ่งเป็นภาษาที่ Jasmine เตรียมไว้สำหรับการโปรแกรม ซึ่งมาโครเหล่านี้ได้เคยกำหนดโครงสร้างโดยใช้ host language เพื่อที่จะใช้ในการดึงข้อมูลต่างๆจาก Jasmine มาโครที่เกี่ยวข้องกับ Object identifier ได้แก่ ODB_OBJECT_ID macro มาโครนี้จะเป็นการดึง Object identifier เพื่อใช้ในการอ้างถึง class objects และ instance objects ซึ่งมีโครงสร้างดังนี้

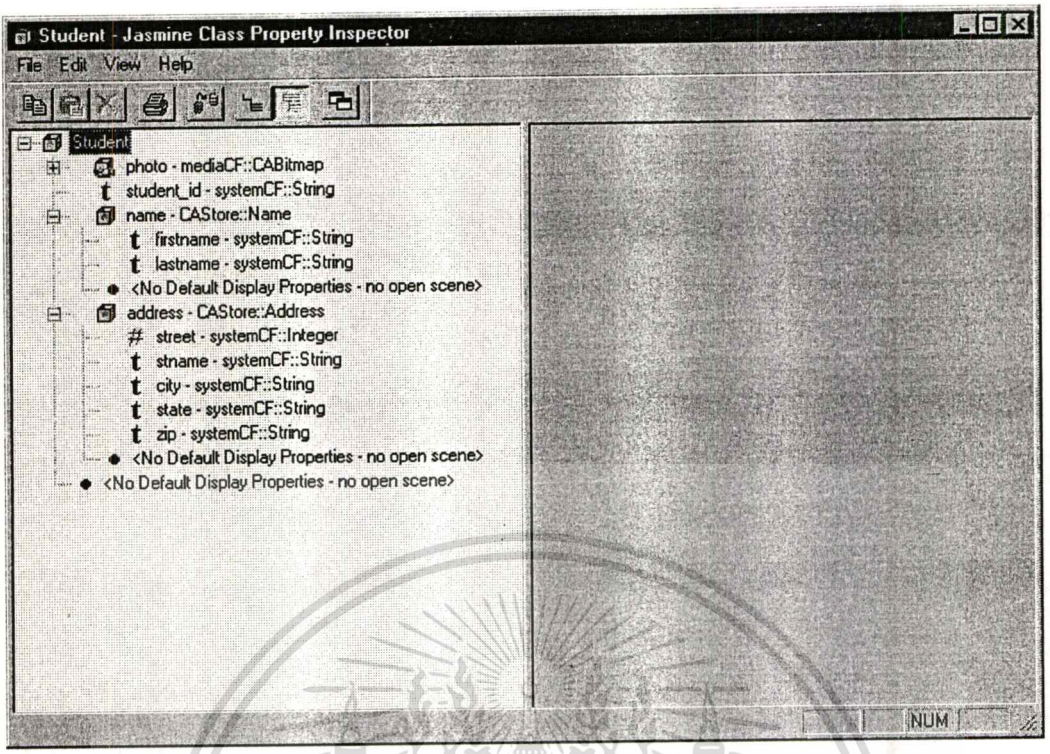
```
typedef struct odb_object_id{
        unsigned char cfname[34];       /*class family name*/
        unsigned short cno;             /*class number*/
        unsigned long ino;              /*instance number*/
} ODB_OBJECT_ID
```

8.2.3 แนวทางในการแก้ไขปัญหา "Unnatural" Join (ปัญหาที่ 5.3)

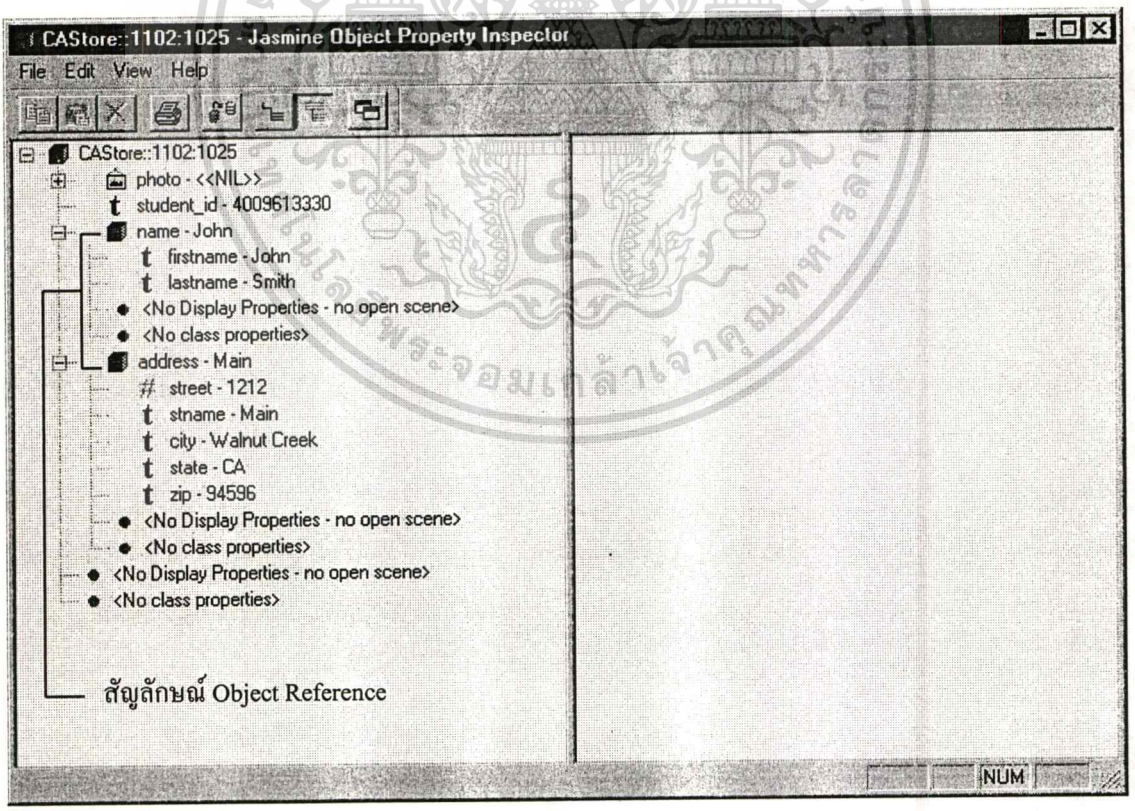
Jasmine จะใช้ OID ในการอ้างถึงออบเจ็กต์ต่างๆที่มิใช่เป็น Atomic Type กล่าวคือในแอตทริบิวต์ที่มีชนิดข้อมูลเป็นแบบผู้ใช้กำหนดเอง (User-Define Type) เช่น คลาสของนักศึกษา, คลาสที่อยู่ เป็นต้น หรือเป็นข้อมูลประเภทมัลติมีเดีย การอ้างถึงข้อมูลในลักษณะนี้จะอ้างอิงโดยผ่าน OID ดังตัวอย่างต่อไปนี้ กำหนดให้คลาส Student ซึ่งมีแอตทริบิวต์หลักๆคือ

student_id	มีชนิดข้อมูลเป็น String ซึ่งเป็น Atomic Type
name	เป็นออบเจ็กต์ของคลาส Name (คลาส Name ประกอบไปด้วย firstname และ lastname)
address	เป็นออบเจ็กต์ของคลาส Address (คลาส Address ประกอบไปด้วย street, stname, city, state, zip)

ส่วนแอตทริบิวต์อื่นๆที่ปรากฏเป็นแอตทริบิวต์ที่ได้รับการถ่ายทอดมากจาก Superclass ของ Student ดังรูปที่ 74



รูปที่ 74 แสดง Property ของออบเจ็กต์ Student



รูปที่ 75 แสดง Object Reference

จากรูปที่ 75 แสดงสัญลักษณ์ในการอ้างถึงอีกออบเจ็กต์หนึ่งในที่นี้ John และ Main เป็นชื่อที่ใช้ในการอ้างถึงอีกออบเจ็กต์หนึ่ง (การใช้ชื่อแบบนี้เพื่อให้ผู้พัฒนาเข้าใจได้ง่าย แต่เวลาระบบทำงานจริงๆจะมีการแปลงชื่อนี้ให้เป็น OID เสียก่อน) ซึ่งทำให้การสร้างออบเจ็กต์ที่มีความซับซ้อนไม่ต้องอาศัยการ Join กันของออบเจ็กต์เหมือนในระบบฐานข้อมูลเชิงสัมพันธ์แต่อาศัย OID ในการอ้างไปยังอีกออบเจ็กต์

บทที่ 9

สรุป

การใช้ค่าของคีย์เพื่อแสดงถึงเอกลักษณ์ของออบเจ็กต์ในระบบฐานข้อมูลเชิงสัมพันธ์ ไม่สามารถตอบสนองงานสารสนเทศที่เกี่ยวข้องกับข้อมูลที่มีความซับซ้อน ได้แก่ออบเจ็กต์ที่มีลักษณะเป็นเซต, อาร์เรย์ หรือแม้แต่ข้อมูลประเภทมัลติมีเดีย เช่น รูปภาพ, เสียง, วีดีโอ กล่าวคือ สำหรับระบบฐานข้อมูลเชิงสัมพันธ์แล้วการสร้างออบเจ็กต์ที่มีความซับซ้อนจะเกิดจากการ Join กันผ่านคีย์ ทำให้เกิดความไม่สะดวกและยุ่งยากในการทำ และการที่ต้องมีการ Join กันของรีเลชั่นทำให้เกิดความสูญเสีย Semantic ของออบเจ็กต์นั้นไป

แนวคิดเชิงออบเจ็กต์ทำให้สามารถเก็บหรือสร้างข้อมูลที่มีความซับซ้อนได้สะดวกมากกว่าโมเดลเชิงสัมพันธ์ ไม่ทำให้สูญเสีย Semantic ของออบเจ็กต์ อีกทั้งยังไม่ก่อปัญหาให้กับระบบงานที่มีการเปลี่ยนแปลงค่าของแอตทริบิวต์ เนื่องจากในระบบฐานข้อมูลเชิงวัตถุจะไม่ได้ใช้ค่าหรือกลุ่มของแอตทริบิวต์ใดในการแสดงเอกลักษณ์ของออบเจ็กต์อย่างเช่นในระบบฐานข้อมูลเชิงสัมพันธ์ แต่จะใช้สิ่งที่เรียกว่า Object Identifier (OID) ซึ่งมีคุณสมบัติพิเศษคือ Unique และเป็นสิ่งที่ระบบเป็นผู้จัดสรรให้ โดยที่ผู้พัฒนาไม่ต้องกำหนดเองเหมือนกับระบบฐานข้อมูลเชิงสัมพันธ์ที่ต้องกำหนดว่าแอตทริบิวต์ใดเหมาะสมที่จะใช้เป็นคีย์เพื่อแสดงเอกลักษณ์ของออบเจ็กต์

คุณสมบัติที่สำคัญอีกประการของ OID คือ ไม่ว่าค่าของแอตทริบิวต์จะเปลี่ยนไปเป็นอย่างไร ความสัมพันธ์ระหว่างออบเจ็กต์ก็ยังคงอยู่ กล่าวคือด้วยคุณสมบัติของ OID ทำให้ไม่ต้อง maintain referential integrity เหมือนกับในระบบฐานข้อมูลเชิงสัมพันธ์

สำหรับ OID ที่ใช้ใน Jasmine จะเป็น Logical OID ซึ่งขนาดเป็น fix length ซึ่งมีความเป็น Uniformity เพราะทุกๆออบเจ็กต์จะมี OID ที่เป็นรูปแบบเดียวกัน การรวมกันของออบเจ็กต์สามารถกระทำกันได้ง่ายกว่าการรวมออบเจ็กต์ที่มี Identifier Key ต่างรูปแบบกัน

บรรณานุกรม

Alfons, Kemper and Guido Moerkotte. Object-Oriented Database Management : Applications in Engineering and Computer Science. Englewood Cliffs, NJ. : Prentice Hall, 1994.

Bancilhon, Françis., Claude Delobel and Paris Kanellakis. Building an Object-Oriented Database System : The Story of O_2. San Mateo, CA. : Morgan Kaufmann, 1992.

Batanov, Dentcho N. Fundamentals of Object-Oriented Analysis, Design and Programming. Bangkok, 1996.

Bertino, Elisa and Lorenzo Martino. Object-Oriented Database Systems : Concepts and Architectures. Workingham, England : Addison-Wesley, 1993.

Computer Associates International and FUJITSU. Concept Guide. [CD-ROM]. Abstract from Jasmine[TM] Version 1.2 Developer Edition: CONCEPT Directory: MP47870798, 1996.

Computer Associates International and FUJITSU. Getting Started. [CD-ROM]. Abstract from Jasmine[TM] Version 1.2 Developer Edition: GS Directory: MP47870798, 1996.

Computer Associates International and FUJITSU. Installation and Operation. [CD-ROM]. Abstract from Jasmine[TM] Version 1.2 Developer Edition: INSTALL Directory: MP47870798, 1996.

Computer Associates International and FUJITSU. Tutorial. [CD-ROM]. Abstract from Jasmine[TM] Version 1.2 Developer Edition: TUTORIAL Directory: MP47870798, 1996.

Computer Associates International and FUJITSU. Using Jasmine Guide. [CD-ROM]. Abstract from Jasmine[TM] Version 1.2 Developer Edition: Cajasdoc File: MP47870798, 1996.

Date C.J. An Introduction to Database System. 6th ed. U.S.A.:Addison-Wesley, 1993.

Ishikawa, Hiroshi and others. "The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System." ACM Transactions on Database Systems. 18 (March 1993):1-50.

Ishikawa, Hiroshi and others. "An Object-Oriented Database System Jasmine: Implementation, Application, and Extension." IEEE Transactions on Knowledge and Data Engineering. 8 (April 1996):285-304.

Kazuhiko, Kato and Masuda Takashi. "Persistent Caching: An Implementation Technique for Complex Objects with Object Identity." IEEE Transactions on Software Engineering. 18 (July 1992):631-645.

Khoshafian, Setrag N. Object-Oriented Databases. New York: John Wiley and Sons, 1993.

Khoshafian, Setrag and Brad A. Baker. Multimedia and Imaging Databases. San Francisco, CA. : Morgan Kaufmann, 1996.

Khoshafian, Setrag N.., and George P. Copeland. "Object identity." Proceedings of the 1st OOPSLA Conference (Portland, Ore., 1986). ACM, New York, 1986, 406-416.

Manola, Frank. An Evaluation of Object-Oriented DBMS Developments. Waltham, MA. : GTE Laboratories Incorporated, 1994.

Steve McClure. "Object Database VS. Object-Relational Databases." IDC Bulletin#14821E. August 1997.

Zdonik, Stanley B. and David Maier. Readings in Object-Oriented Database Systems. San Mateo, CA. : Morgan Kaufmann, 1990.

ภาคผนวก

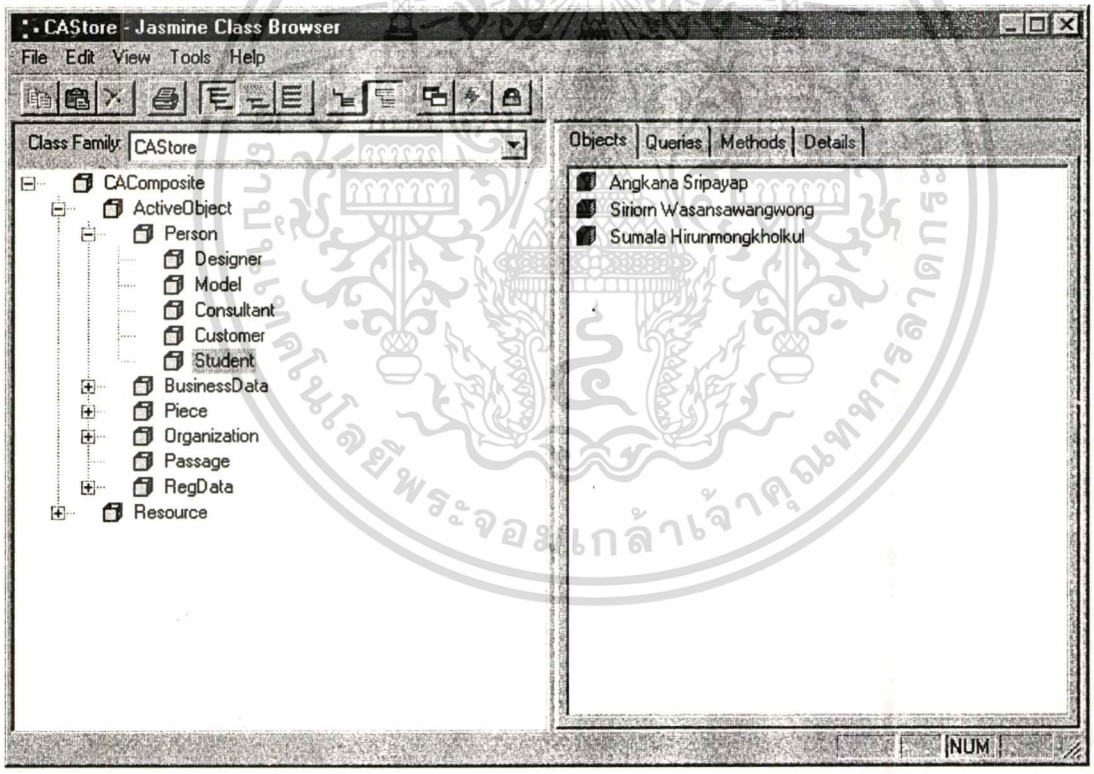
ต้นแบบ Registration Information System

ต้นแบบ Registration Information System

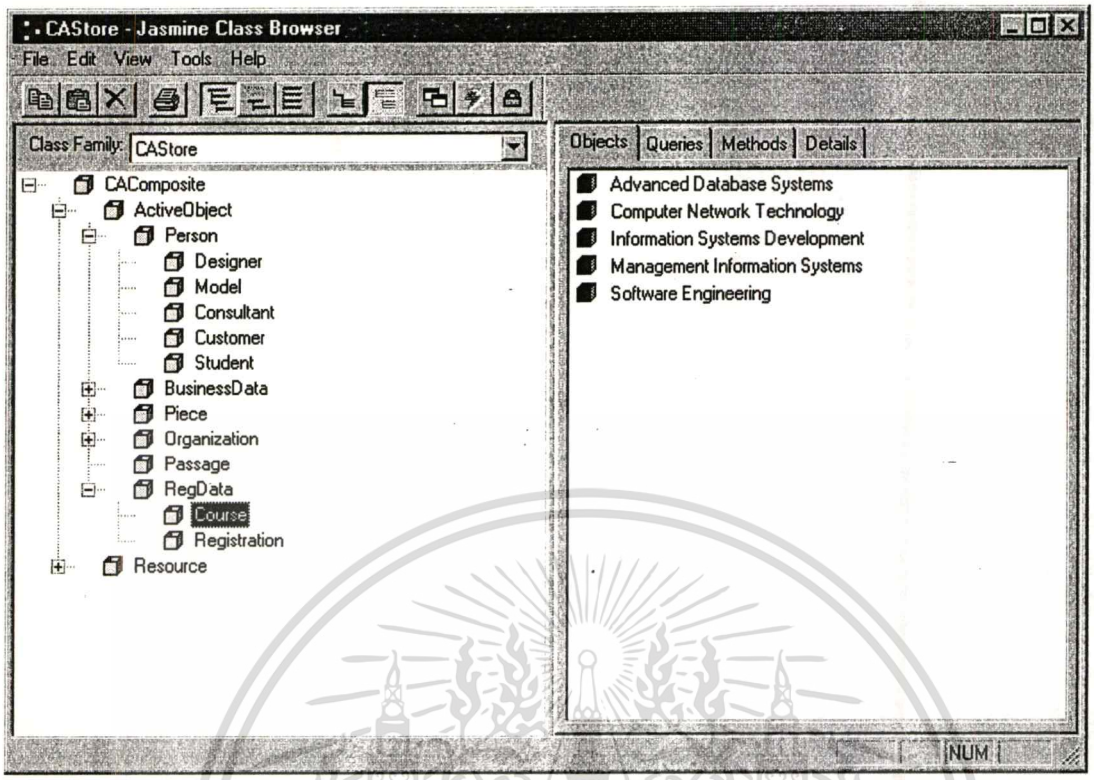
ต้นแบบ Registration Information System เป็นระบบที่มีการทำงานหลักๆอยู่ 3 อย่างคือ

1. Course Look Up เป็นส่วนของการดูรายละเอียดเกี่ยวกับวิชาที่มีอยู่ในระบบฐานข้อมูลทั้งหมด
2. Registration Look Up เป็นส่วนของการดูรายละเอียดการลงทะเบียนของนักศึกษาที่ได้มีการลงทะเบียนไปแล้ว
3. Course Update เป็นส่วนของการแก้ไขเปลี่ยนแปลงข้อมูลที่เกี่ยวกับวิชาที่มีอยู่ในระบบฐานข้อมูล

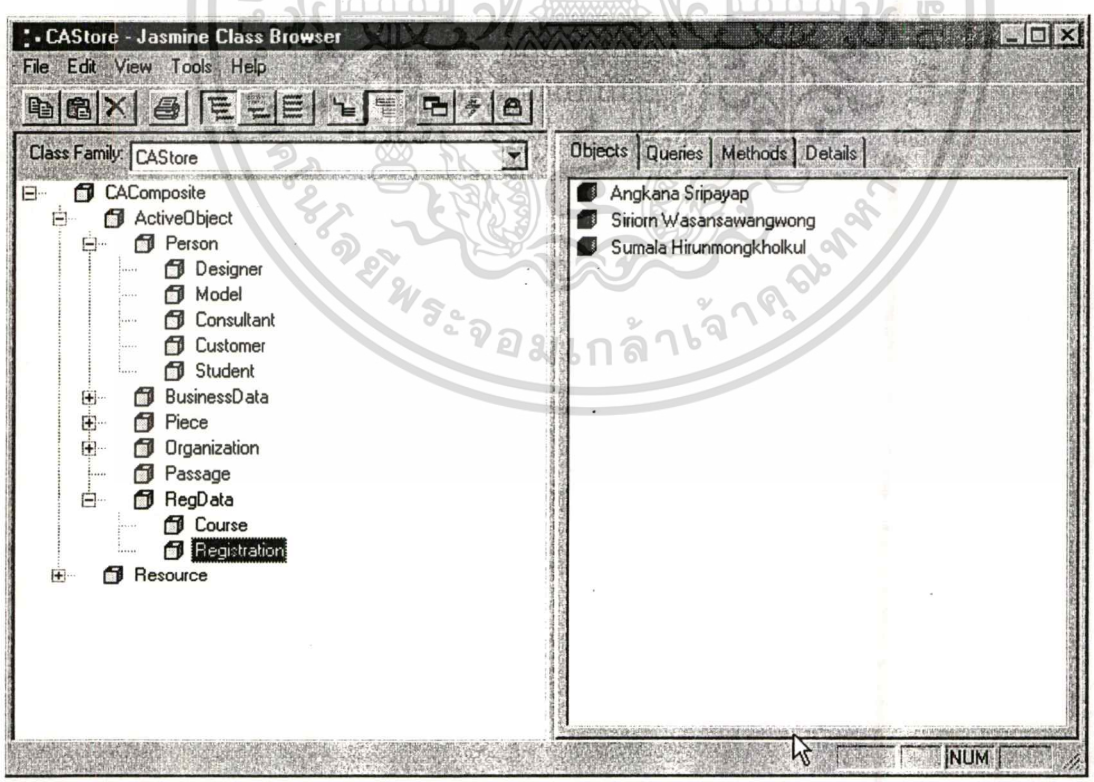
ออบเจ็กต์ที่เป็นข้อมูลเริ่มต้นในการรันแอปพลิเคชั่นที่มีอยู่ใน Jasmine Database ของคลาส Student, Course และ Registration เป็นดังรูปที่ 76, 77 และ 78 ตามลำดับ



รูปที่ 76 แสดงออบเจ็กต์ของคลาส Student

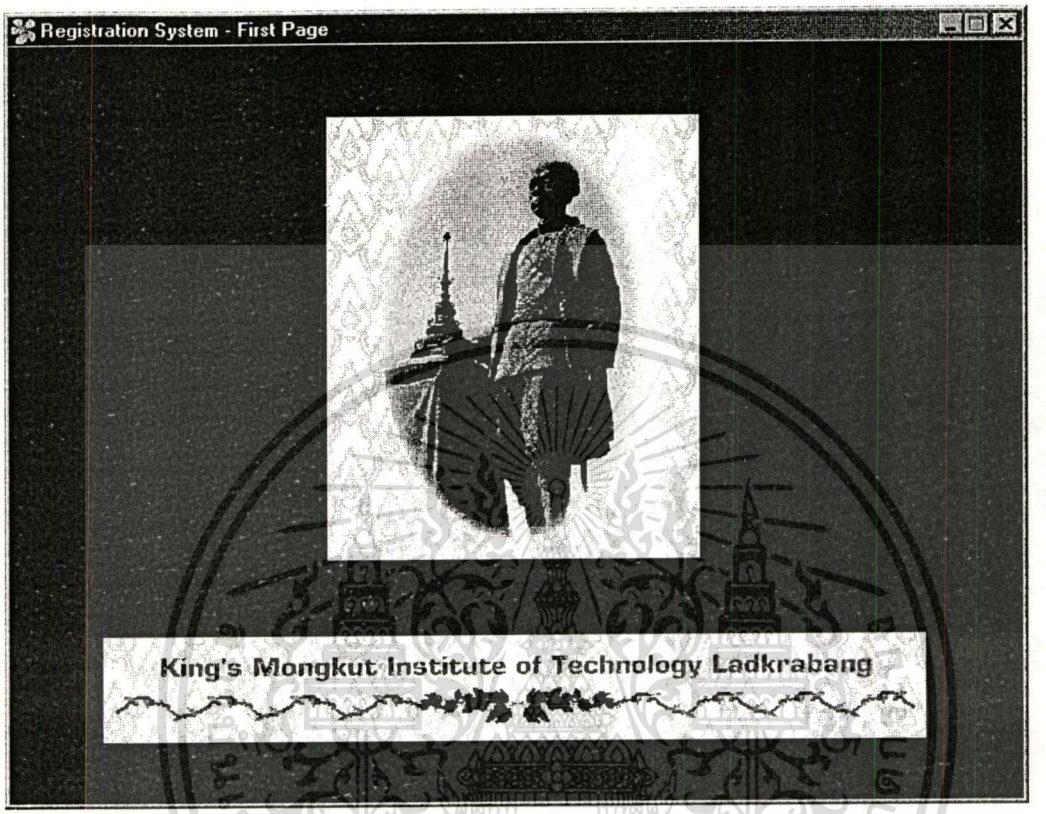


รูปที่ 77 แสดงออบเจ็กต์ของคลาส Course



รูปที่ 78 แสดงออบเจ็กต์ของคลาส Registration

เมื่อรันแอปพลิเคชั่น Registration Information System จะปรากฏหน้าจอ First Page ดังรูปที่ 79



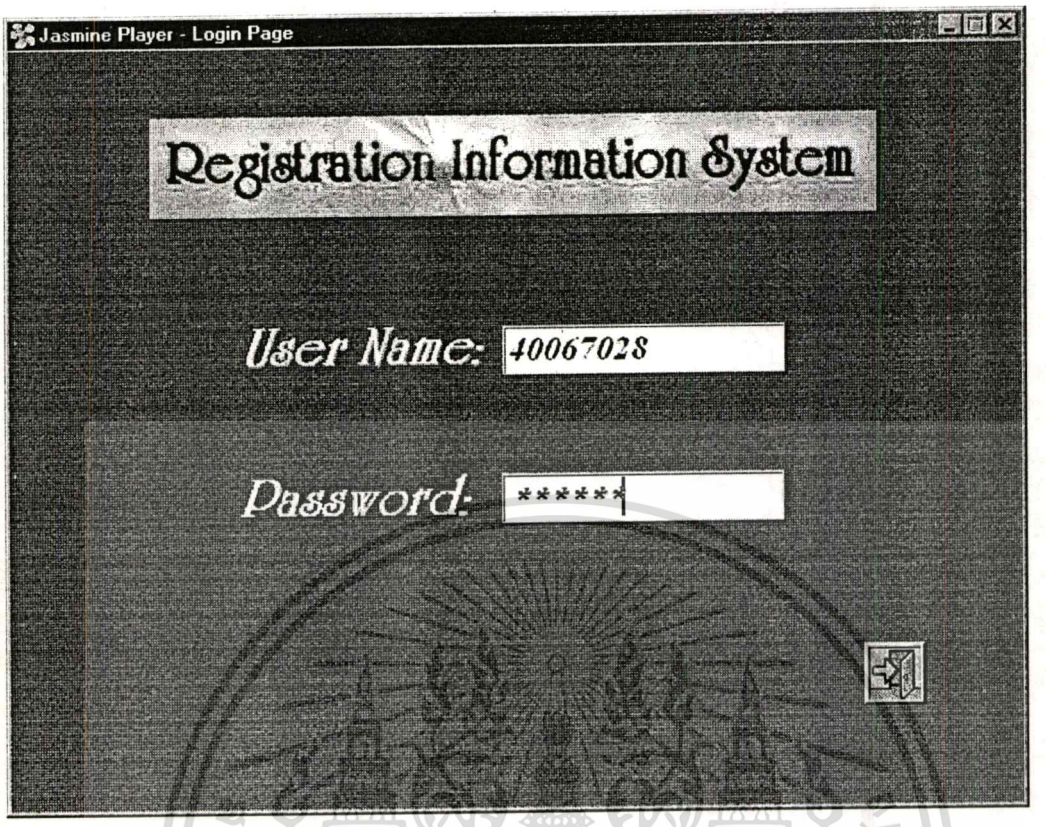
รูปที่ 79 แสดงหน้าจอ First Page

เมื่อ Click ที่รูปหรือป้ายสถาบันฯจะเข้าสู่หน้าจอ Login Page เพื่อให้ผู้ใช้ใส่ Username และ Password ดังรูปที่ 80 ในที่นี้ใส่

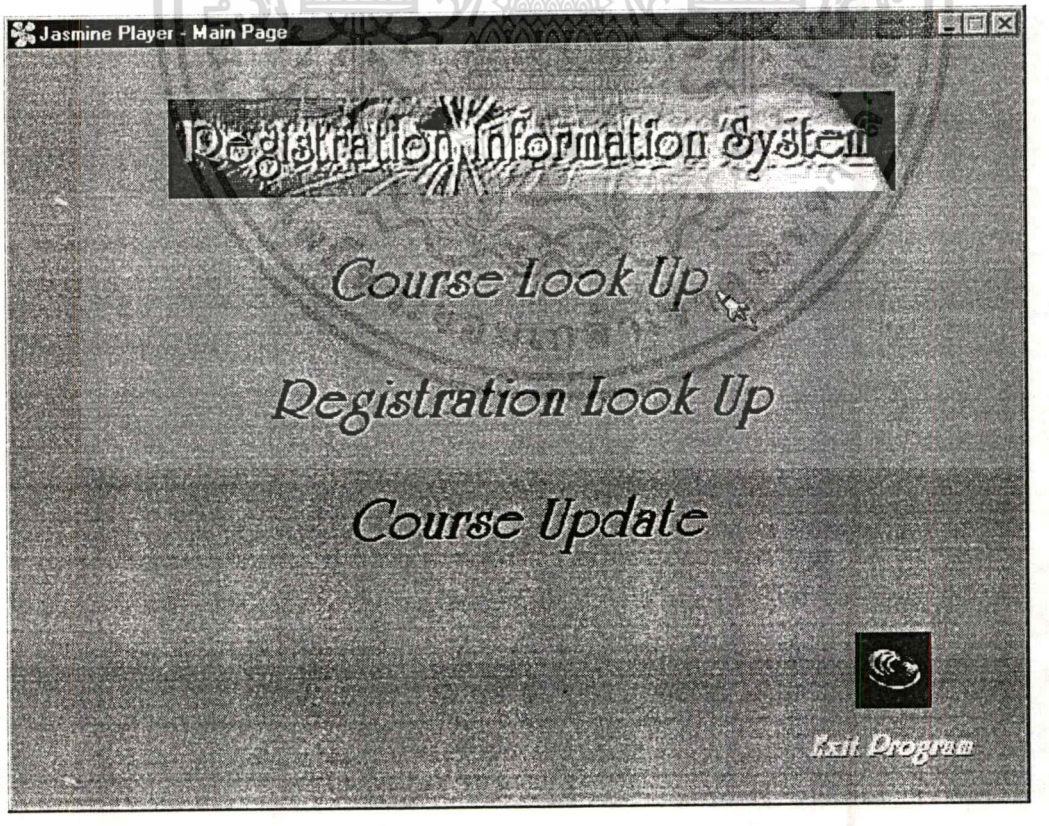
Username: 40067028

Password: Sumala

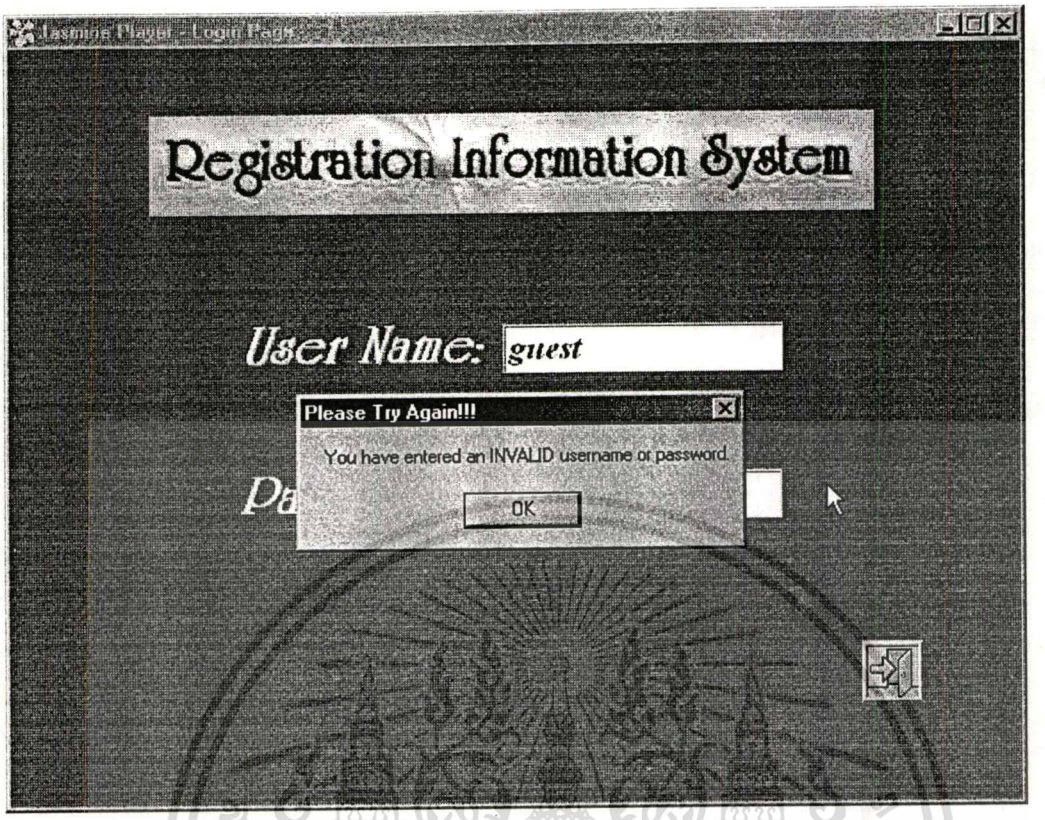
แล้ว Click ที่  เพื่อเข้าสู่ระบบซึ่งจะปรากฏหน้าจอ Main Page ซึ่งเป็นหน้าจอทำงานหลักของระบบดังรูปที่ 81 หากใส่รหัสผู้ใช้หรือรหัสผ่านไม่ถูกต้องจะปรากฏหน้าจอ Invalid Login ดังรูปที่ 82



รูปที่ 80 แสดงหน้าจอ Login Page



รูปที่ 81 แสดงหน้าจอ Main Page



รูปที่ 82 แสดงหน้าจอ Invalid Login

ถ้าต้องการดูรายละเอียดเกี่ยวกับวิชาต่างๆที่มีอยู่ในฐานข้อมูลให้ Click เมนู Course Look Up จะปรากฏหน้าจอดังรูปที่ 83 และถ้า Click ปุ่ม

- 

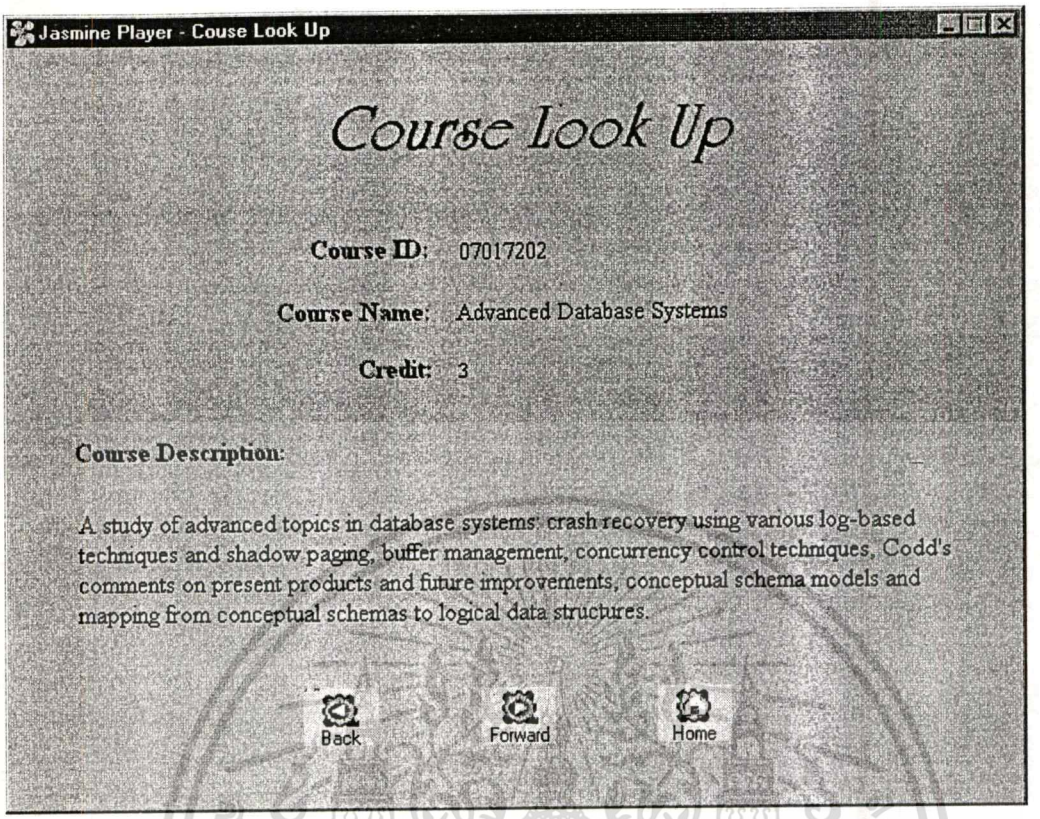
เมื่อต้องการดูรายละเอียดเกี่ยวกับวิชาที่อยู่ก่อนหน้ารายวิชานี้

- 

เมื่อต้องการดูรายละเอียดเกี่ยวกับวิชาที่อยู่ถัดไปจากรายวิชานี้

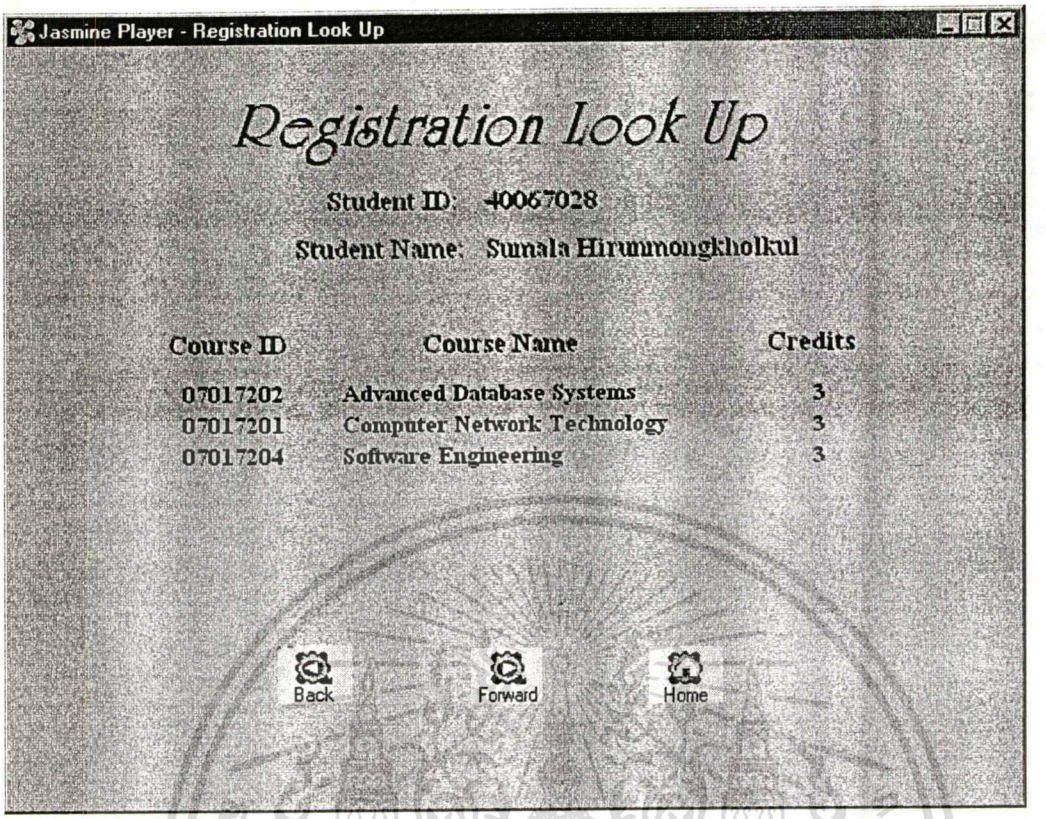
- 

เมื่อต้องการกลับไปที่หน้าจอทำงานหลัก



รูปที่ 83 แสดงหน้าจอ Course Look Up

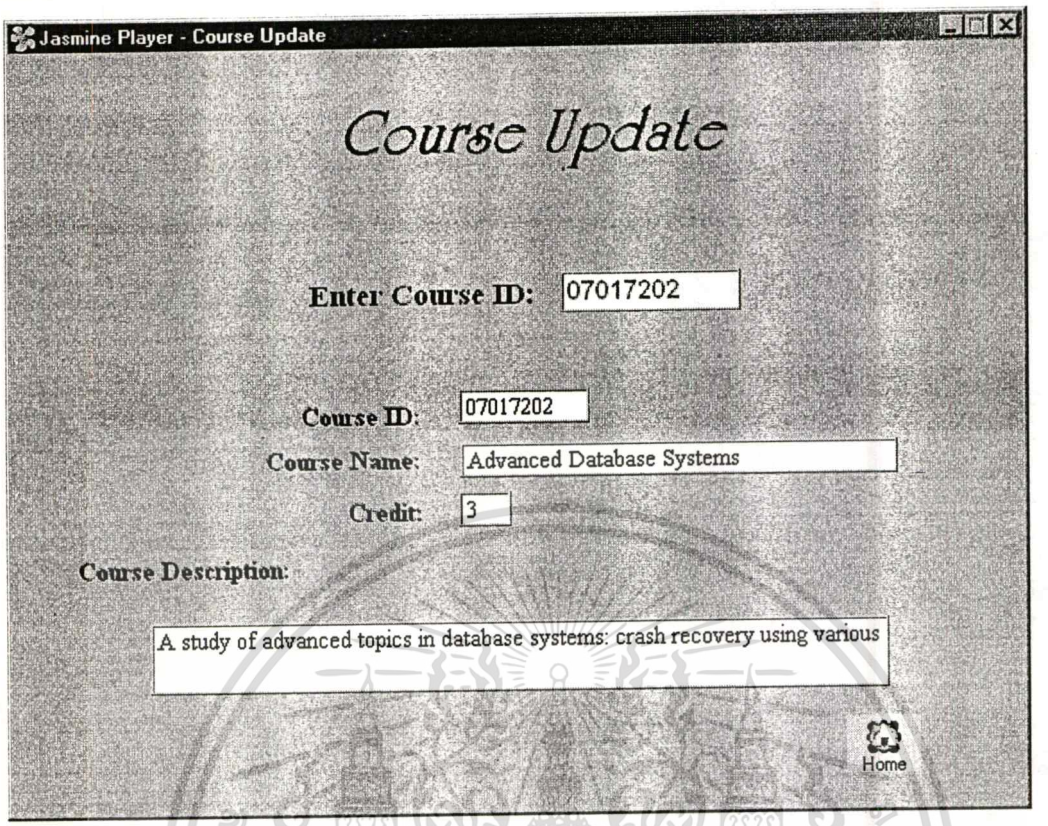
จากหน้าจอทำงานหลัก ถ้าต้องการดูรายละเอียดเกี่ยวกับการลงทะเบียนของนักศึกษาที่ได้มีการลงทะเบียนไปแล้วให้ Click ที่ Registration Look Up จะปรากฏหน้าจอดังรูปที่ 84



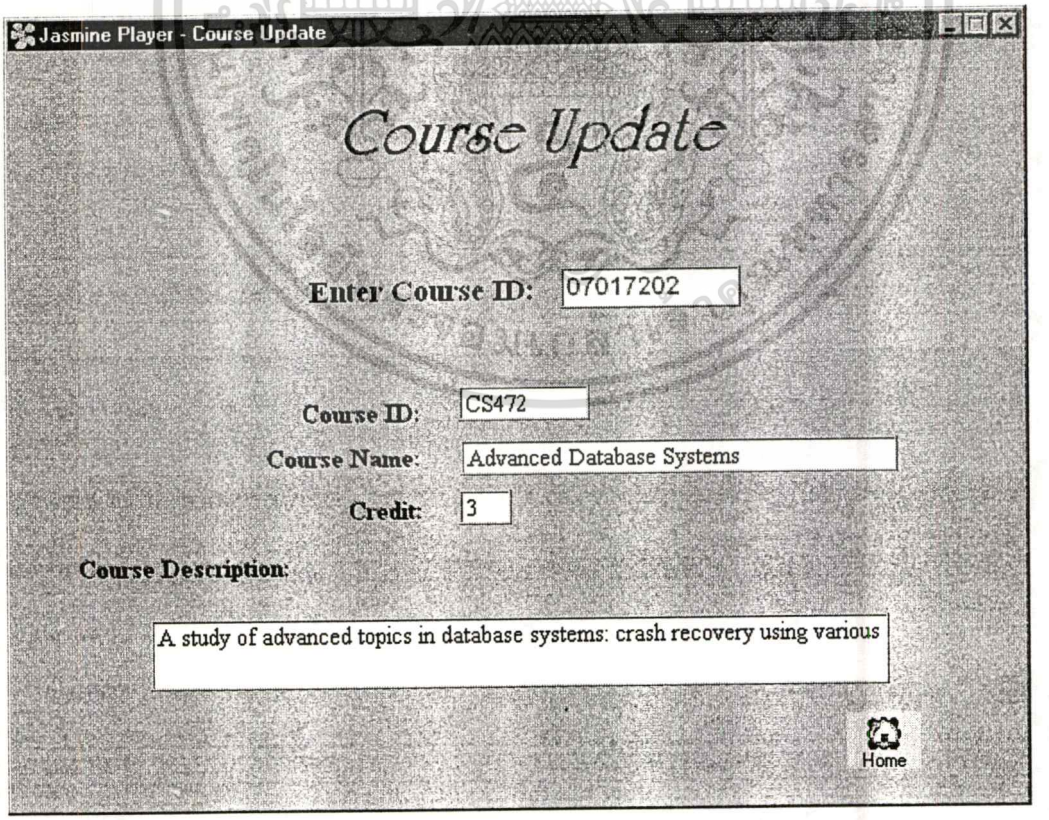
รูปที่ 84 แสดงหน้าจอ Registration Look Up

จากหน้าจอทำงานหลัก เป็นส่วนของการแก้ไขเปลี่ยนแปลงข้อมูลที่เกี่ยวกับวิชาที่มีอยู่ในระบบฐานข้อมูล ให้ Click ที่ Course Update ในช่อง Enter Course ID: ให้ใส่รหัสวิชาที่ต้องการ Update ในที่นี้ต้องการ Update วิชา Advanced Database Systems ซึ่งมีรหัสวิชาเป็น 07017202 ระบบนี้จะดึงข้อมูลของรหัสวิชานี้ออกมาแสดงเพื่อให้ผู้ใช้แก้ไขข้อมูลได้ ดังรูปที่ 85

ในที่นี้ต้องการ Update รหัสวิชาจาก 07017202 เป็น CS472 ดังรูปที่ 86

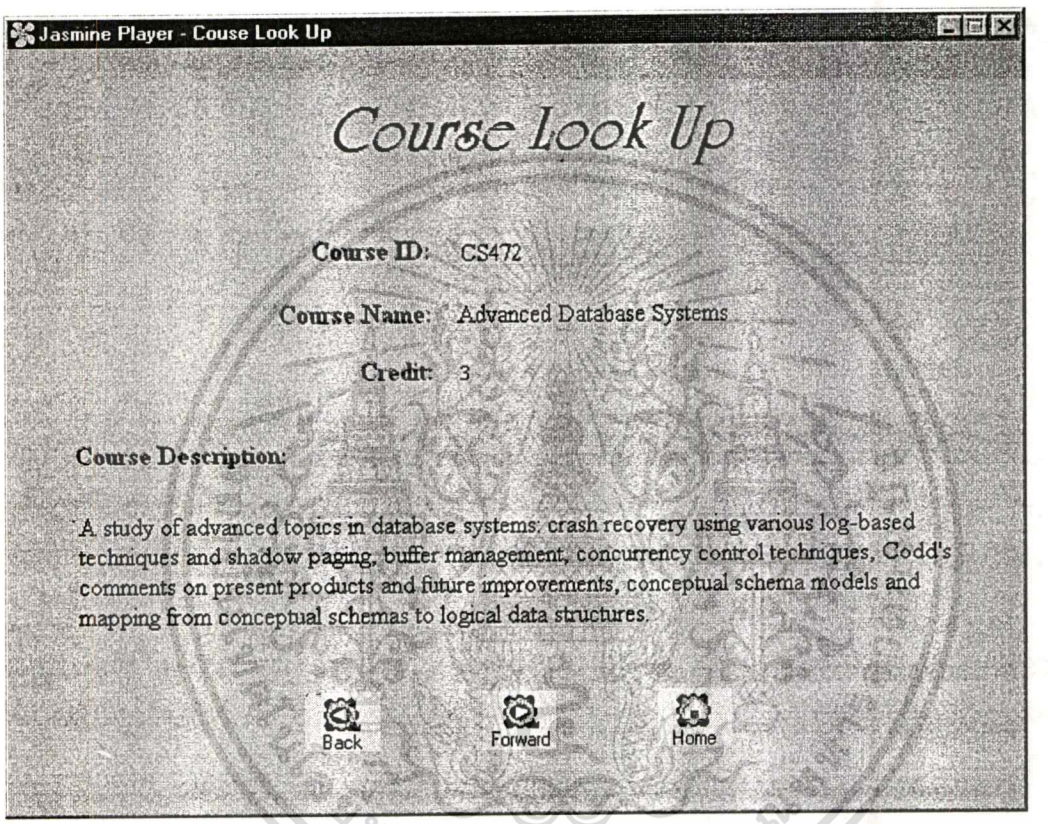


รูปที่ 85 แสดงหน้าจอ Course Update

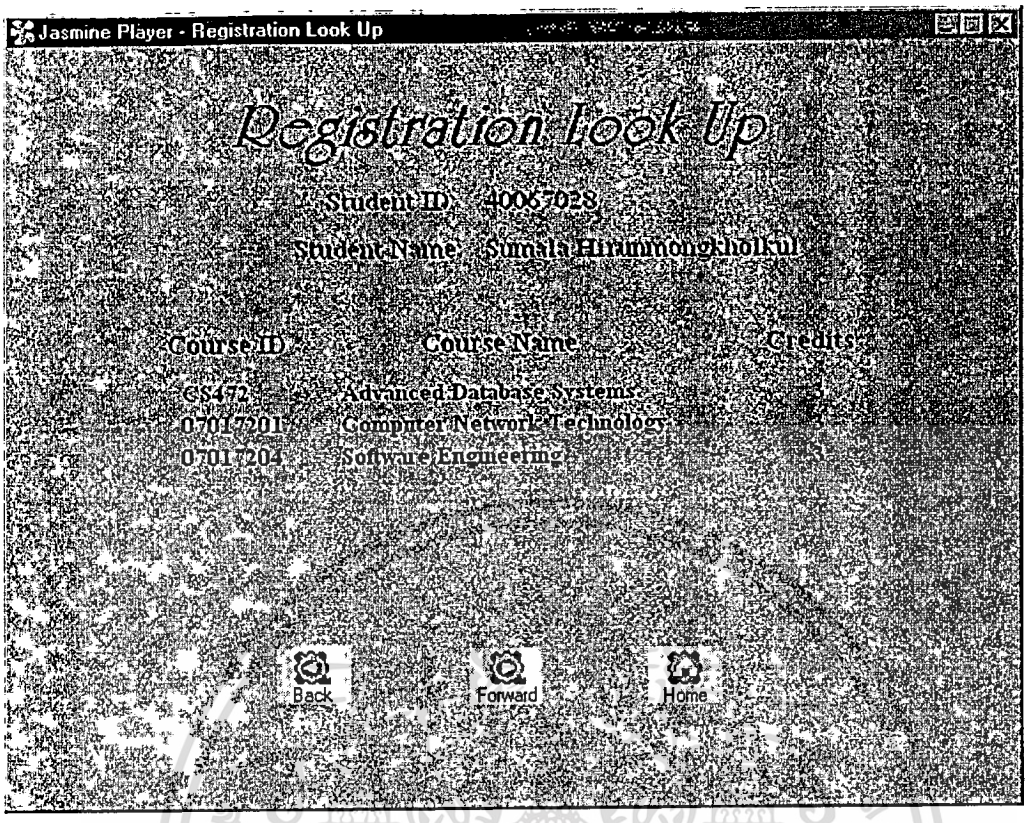


รูปที่ 86 แสดงหน้าจอ Course Update จากรหัสวิชา 07017202 ไปเป็น CS472

เมื่อมีการแก้รหัสวิชาซึ่งโดยปกติแล้วแอตทริบิวต์นี้มักถูกใช้เป็น Identifier Key ในระบบฐานข้อมูลเชิงสัมพันธ์ซึ่งทำให้ต้องมีการ maintain referential integrity แต่สำหรับ Jasmine ไม่ต้อง maintain เพราะแอตทริบิวต์นี้ไม่ได้ถูกใช้เป็น Identifier Key เหมือนในระบบฐานข้อมูลเชิงสัมพันธ์ แต่ยังสามารถรักษาความถูกต้องไว้ได้กล่าวคือทุกๆออบเจ็กต์ที่อ้างอิงถึงออบเจ็กต์นี้ต้องได้รับการเปลี่ยนแปลงจากรหัสวิชา 07017202 เป็น CS472 ดังแสดงในรูปที่ 87 และ 88

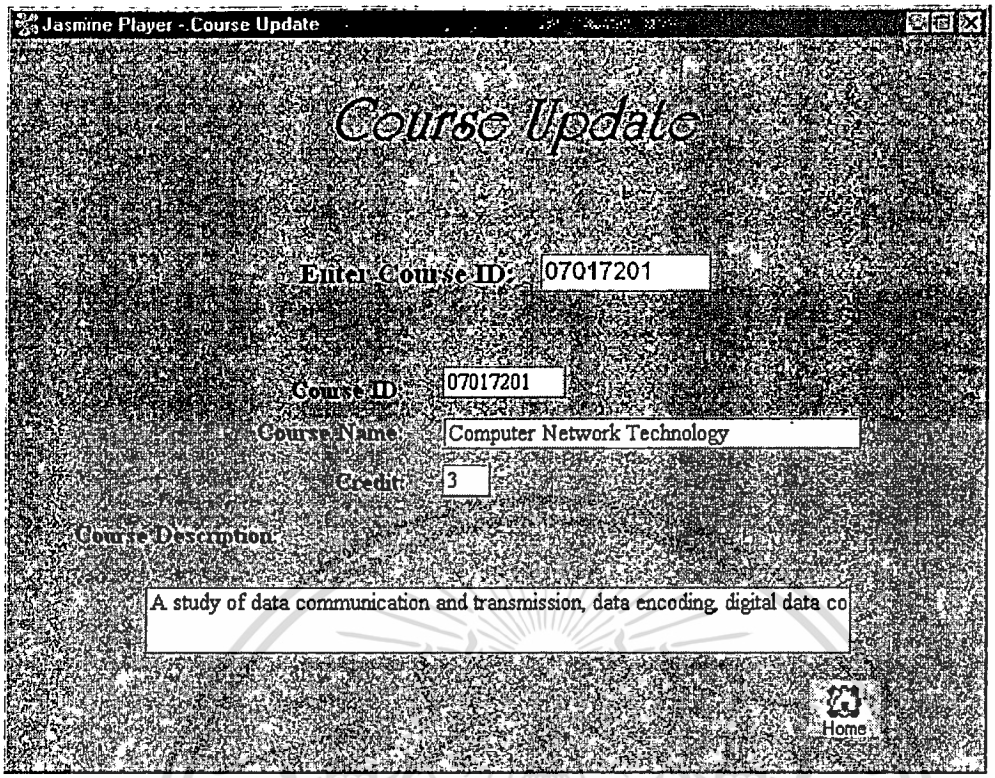


รูปที่ 87 แสดงหน้าจอ Course Look Up หลังจาก Update รหัสวิชา 07017202 ไปเป็น CS472

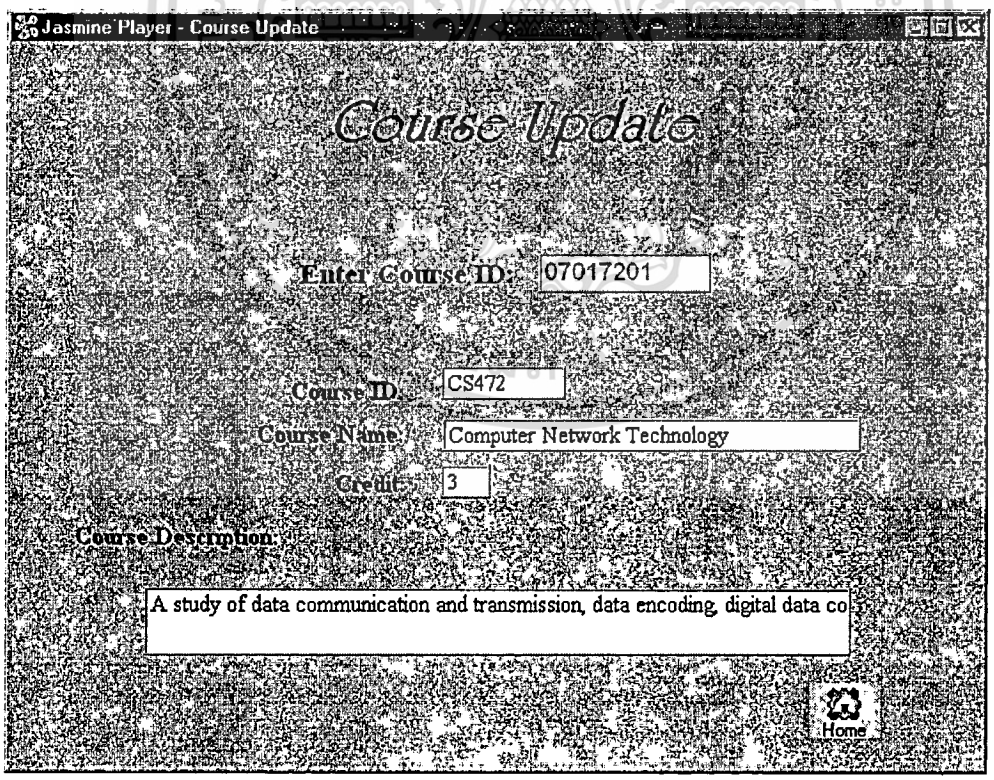


รูปที่ 88 แสดงหน้าจอ Registration Look Up หลังจาก Update รหัสวิชา 07017202 ไปเป็น CS472

นอกจากนี้ระบบสามารถรองรับเหตุการณ์ที่มีรหัสวิชาซ้ำกันได้อีกด้วย (ถ้าในระบบฐานข้อมูลเชิงสัมพันธ์ที่ใช้รหัสวิชาเป็น Identifier Key จะทำไม่ได้) ตัวอย่างเช่น ถ้าเปลี่ยนรหัสวิชา 07017201 ซึ่งเป็นวิชา Computer Network Technology ซึ่งมีรายละเอียดดังรูปที่ 89 ให้เป็นรหัส CS472 ซึ่งตรงกับวิชา Advanced Database Systems ก็สามารถทำได้ดังรูปที่ 90

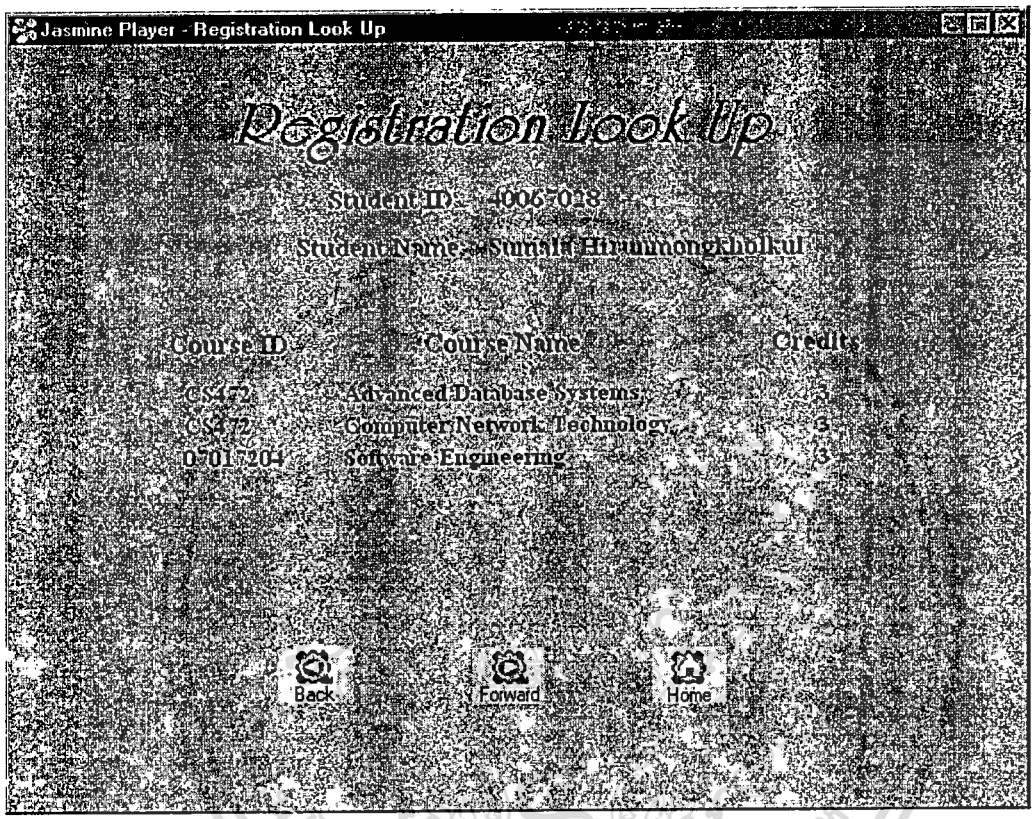


รูปที่ 89 แสดงหน้าจอ Course Update ของรหัสวิชา 07017201 ก่อนการ Update



รูปที่ 90 แสดงหน้าจอ Course Update ของรหัสวิชา 07017201 หลัง Update ไปเป็น CS472

หลังจากการ Update วิชา Computer Network Technology จากรหัสวิชา 07017201 ไปเป็น CS472 ออบเจ็กต์ที่อ้างถึงออบเจ็กต์นี้จะถูกเปลี่ยนแปลงไปโดยยังคงความถูกต้องไว้อยู่ ดังแสดงในรูปที่ 79



รูปที่ 91 แสดงหน้าจอ Registration Look Up หลังจาก Update รหัสวิชา 07017201 ไปเป็น CS472

ประวัติผู้เขียน

ชื่อผู้เขียน	นางสาวสุมาลา หิรัญมงคลกุล
วันเดือนปีเกิด	30 กันยายน 2518
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษาระดับปริญญาตรี	วท.บ. (คณิตศาสตร์ประยุกต์)
สถานที่สำเร็จการศึกษา	คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีที่สำเร็จการศึกษาในระดับปริญญาตรี	2539