การพัฒนาระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา

Temporal Object-Oriented Database Development

โดย

นาย ชวลิต ฉันชัยสิริเวทย์

รหัส 41067107

อาจารย์ที่ปรึกษา

รศ.ดร.ศุภมิตร จิตตะยโศธร

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2543

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา
นักศึกษา	นาย ชวลิต ฉันชัยสิริเวทย์
อาจารย์ที่ปรึกษา	รศ.ดร.ศุภมิตร จิตตะยโศธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2543

บทคัดย่อ

ในปัจจุบันระบบฐานข้อมูลที่ใช้กันอยู่จะเก็บเฉพาะข้อมูลล่าสุดทำให้เกิดปัญหาในการสืบค้นข้อมูลในอดีต จึงเกิดฐานข้อมูลที่สามารถจัดเก็บและสืบค้นข้อมูลที่เปลี่ยนแปลงไปตามกาลเวลาได้ เรียกว่าฐานข้อมูลอิงเวลา (Temporal Database) แต่ฐานข้อมูลอิงเวลาก็ทำให้เกิดความซับซ้อนของโครงสร้างข้อมูล ทางแก้คือเราจะใช้คุณสมบัติของระบบฐานข้อมูลเชิงวัตถุซึ่งเหมาะแก่การแก้ปัญหาข้อมูลที่มีโครงสร้างซับซ้อนมาแก้ปัญหา และสุดท้ายเราจะได้ระบบฐานข้อมูลใหม่คือ ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา และเราจะทำการศึกษาฐานข้อมูลชนิดนี้ว่ามีการทำงานเป็นอย่างไร รูปแบบเป็นอย่างไร แนวทางในการศึกษาคือจะใช้ฐานข้อมูลเชิงวัตถุมานำเสนอในรูปแบบของ temporal database โดยสร้าง temporal data type ขึ้นเป็นโครงสร้างข้อมูลชนิดอิงเวลา ใส่ไปในฐานข้อมูลเชิงวัตถุ และนำโครงสร้างข้อมูลนั้นไปใช้กับระบบงานตัวอย่าง เพื่อสืบค้นข้อมูลที่เปลี่ยนแปลงไปตามกาลเวลา

Title	Temporal Object-Oriented Development
Student	Mr. Chavalit Chanchaisirivet
Advisor	Assoc.Prof. Dr. Suphamit Chittayasothorn
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2000

ABSTRACT

In current we found that Database System that used everyday keep only updated data but in many system application need to know history of data. Then we want database that can keep and query history of data which data has change according the time or you known to mean the Temporal Database. But Temporal Database is made the complex to data structure. Way to solve is we will use property of Object-Oriented that suitable for solve complex data structure and the last we will have new database system that call Temporal Object-Oriented Database and we will study this database that how to work and what structure it is. We will study by use Object-Oriented to create temporal data type and use it to implement example application for query history of data.

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญภาพ

รูปที่ หน้า

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของระบบ

ในปัจจุบันพบว่าระบบฐานข้อมูลที่ใช้กันอยู่จะเก็บเฉพาะข้อมูลล่าสุดเท่านั้น แต่ในบางระบบงานมีความจำเป็นที่จะใช้ข้อมูลในอดีตด้วย จึงได้มีความต้องการฐานข้อมูลที่สามารถจัดเก็บและสืบค้นข้อมูลที่เปลี่ยนแปลงไปตามกาลเวลาได้ หรือที่เราเรียกว่าฐานข้อมูลเชิงเวลา (Temporal Database) บวกกับการมองข้อมูลในรูปแบบใหม่ที่มองทุกอย่างเป็นวัตถุ (Object) และจากข้อดีต่างๆ ของแนวคิดเชิงวัตถุ (Object-Oriented) เราจะทำการรวมรูปแบบของฐานข้อมูลเชิงวัตถุ (Object-Oriented Database) และฐานข้อมูลเชิงเวลาเพื่อให้เกิดรูปแบบข้อมูล (Data Model) ใหม่ที่เรียกว่า ฐานข้อมูลเชิงวัตถุที่สัมพันธ์กับเวลา (Temporal Object-Oriented Database)

จุดมุ่งหมายของฐานข้อมูลเชิงเวลาคือการรวบรวมข้อมูลทั้งในอดีต ปัจจุบัน และอนาคตเข้าไปในฐานข้อมูลและยังใช้งานฐานข้อมูลได้อย่างง่ายที่สุด ส่วนเป้าหมายของฐานข้อมูลเชิงวัตถุคือการซ่อนข้อมูลในวัตถุ เพื่อปกป้องข้อมูล และการถ่ายทอดคุณสมบัติของวัตถุเพื่อการนำกลับมาใช้ใหม่

ฐานข้อมูลเชิงเวลา (Temporal Database) และ ฐานข้อมูลเชิงวัตถุ (Object-Oriented Database) จะมีบทบาทที่สำคัญในอนาคตอันใกล้นี้และถ้าทั้ง 2 ฐานข้อมูลสามารถรวมกันได้ ก็จะรวมคุณสมบัติและข้อดีต่างๆ ของทั้ง ฐานข้อมูลเชิงเวลาและฐานข้อมูลเชิงวัตถุเข้าไว้ด้วยกัน ทำให้เกิดฐานข้อมูลที่มีประสิทธิภาพสูงสุด

1.2 วัตถุประสงค์ของงานวิจัย

ในการพัฒนาระบบงานวิจัยนี้มีวัตถุประสงค์ดังนี้

1. ศึกษาทฤษฎี และหลักการทำงานของระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา
2. ศึกษาตัวอย่างระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลาในด้านต่างๆ คือ
    a. โครงสร้างทางด้านสถาปัตยกรรม
    b. รูปแบบการใช้งานโปรแกรมประยุกต์กับระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา
    c. วิธีการใช้งานของระบบการจัดการฐานข้อมูลเชิงวัตถุ

3. การสร้างตัวอย่างโปรแกรมประยุกต์ พร้อมทั้งออกแบบโครงสร้างการทำงานให้อยู่ในรูปของระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา

1.3 ทฤษฎี หรือแนวความคิดที่เกี่ยวข้อง

ในปัจจุบันระบบฐานข้อมูลที่ใช้กันอยู่จะเก็บเฉพาะข้อมูลล่าสุดทำให้เกิดปัญหาในการสืบค้นข้อมูลในอดีต จึงเกิดฐานข้อมูลที่สามารถจัดเก็บและสืบค้นข้อมูลที่เปลี่ยนแปลงไปตามกาลเวลาได้ เรียกว่าฐานข้อมูลอิงเวลา (Temporal Database) แต่ฐานข้อมูลอิงเวลาก็ทำให้เกิดความซับซ้อนของโครงสร้างข้อมูล ทางแก้คือเราจะใช้คุณสมบัติของระบบฐานข้อมูลเชิงวัตถุซึ่งเหมาะแก่การแก้ปัญหาข้อมูลที่มีโครงสร้างซับซ้อนมาแก้ปัญหา สุดท้ายเราจะได้ระบบฐานข้อมูลใหม่คือ ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา ที่สามารถแก้ปัญหาการสืบค้นข้อมูลในอดีตและยังสามารถจัดการกับโครงสร้างที่ซับซ้อนของฐานข้อมูลได้อีก

1.4 ขอบเขตของงานวิจัย

ศึกษาระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลาในด้านต่างๆ พร้อมทั้งนำโปรแกรมระบบจัดการฐานข้อมูลมาทำการศึกษาการใช้งานและทำการสร้างโปรแกรมประยุกต์ขึ้นมา โดยทำการออกแบบให้มีโครงสร้างการทำงานสนับสนุนระบบฐานข้อมูลเชิงวัตถุ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. เรียนรู้โครงสร้างการทำงานของระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา
2. สามารถสร้างโปรแกรมประยุกต์ที่ใช้งานกับระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลาได้ เรียนรู้การควบคุมการทำงานระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา

1.6 เนื้อหาในงานวิจัย

เนื้อหางานวิจัยในบทแรกนี้จะกล่าวถึงความสำคัญของระบบงาน และวัตถุประสงค์ในการทำงานวิจัย บทที่ 2 กล่าวถึงระบบฐานข้อมูลเชิงวัตถุ ซึ่งจะกล่าวถึงโครงสร้างต่างๆ ของระบบฐานข้อมูลเชิงวัตถุ ในบทที่ 3 จะกล่าวถึงระบบฐานข้อมูลอิงเวลา และโครงสร้างต่างๆ ของระบบฐานข้อมูลอิงเวลา บทที่ 4 กล่าวถึงหลักการทำงานของระบบจัดการฐานข้อมูลเชิงวัตถุทั่วๆ ไป บทที่ 5 กล่าวถึงหลักการทำงานของระบบจัดการฐานข้อมูลเชิงวัตถุ CACHE บทที่ 6 กล่าวถึงแนวทางในการแก้ไขปัญหาและระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา บทที่ 7 กล่าวถึงโปรแกรมประยุกต์ และวิธีการในการออกแบบระบบฐานข้อมูลตัวอย่าง และในบทที่ 8 จะกล่าวถึงบทสรุป และแนวทางใน

การพัฒนาระบบต่อไป พร้อมทั้งข้อจำกัดต่างๆ ที่ใช้ในการพัฒนาระบบเพื่อให้ได้ระบบที่สมบูรณ์ตามที่สามารถใช้งานได้จริง

บทที่ 2
ฐานข้อมูลเชิงวัตถุ

Object-Oriented Data Model

เนื่องจากข้อจำกัดต่างๆ ที่พบของ Relational Model เช่น ในการที่ไม่รองรับลักษณะข้อมูลที่ซับซ้อนเพิ่มขึ้นเช่น ภาพ และเสียง ทำให้มีการนำแนวความคิดเรื่อง Object-Oriented ซึ่งกำลังเป็นที่นิยมในปัจจุบันว่ามีประสิทธิภาพสูง มารวมกับระบบฐานข้อมูลเพื่อแก้ปัญหาดังกล่าว โดยเราเรียก Data Model ใหม่นี้ว่า Object-Oriented Data Model

โครงสร้างของ Object-Oriented Data Model ก็คือ สิ่งต่างๆ ทุกอย่างไม่ว่าจะเป็นคน สิ่งของ เราสามารถมองมันเป็น Object ได้ และ Object ทุกๆ Object จะมีเพียงหนึ่งเดียว ไม่มี Object ใดซ้ำกันหรือที่เราเรียกว่า Unique Identity และแต่ละ Object ก็จะมีกลุ่มของ Attribute Value กับกลุ่มของ Method เป็นสมาชิกใน Object โดย Object ต่างๆ จะรวมกลุ่มโครงสร้างและคุณสมบัติที่เหมือนกันเข้าเป็น Class ส่วนคุณสมบัติอื่นๆ ของ Object-Oriented Model คือ Inheritance, Encapsulation, Information Hiding และอื่นๆ

Object-Oriented Database Systems

คือการนำวิธีการจัดการข้อมูลเชิงวัตถุมาพิจารณาประกอบร่วมกับความสามารถต่างๆ ของฐานข้อมูล เพื่อนำไปประยุกต์สู่ระบบการจัดการฐานข้อมูลในรูปแบบของวัตถุที่มีความเป็นอิสระต่อกันในการทำงาน

2.1 ส่วนประกอบของ Object-Oriented Data Model

2.1.1 โครงสร้างข้อมูลเชิงวัตถุ ประกอบด้วย

OBJECT

Object คือบางสิ่งบางอย่างซึ่งสามารถเห็น สัมผัสหรือรู้สึกได้ที่ซึ่งเก็บข้อมูล (Data) และคุณสมบัติ (Behavior) ที่เกี่ยวข้องเข้าไว้ด้วยกัน Object อาจจะเป็นได้ทั้ง คน สถานที่ สิ่งของ หรือเหตุการณ์ก็ได้

Object นั้นคล้ายๆ กับ Entity ใน E-R Model เพียงแต่ว่า Object-Oriented นั้นรวมแนวคิดของการ Encapsulating โดยรวมข้อมูลและ Code ที่สัมพันธ์กับ Object นั้น เข้าไปอยู่ด้วยกันใน Object เดียวกัน และการติดต่อสื่อสารกันเองของ Object หรือสื่อสารกับระบบจะติดต่อด้วย Message ดังนั้น Interface ระหว่าง Object และระบบถูกกำหนดโดยกลุ่มของ Message

โดยทั่วไป Object จะประกอบด้วย

- กลุ่มของ Attributes หรือ Data คือข้อมูลที่แทนลักษณะเฉพาะของ Object นั้น เช่น Customer Number, First Name, Home Address, ฯลฯ คล้ายๆ กับ Attribute ใน E-R Model
- กลุ่มของ Method หรือ Behavior คือสิ่งต่างๆที่ Object สามารถกระทำ รวมไปถึงฟังก์ชันซึ่งกระทำกับตัว Attributes ใน Object นั้นเอง โดยเฉพาะเจ้าของ Object เท่านั้นถึงจะมีสิทธิ์ในการแก้ไขข้อมูลใน Attributes ของตัวเอง
- กลุ่มของ Messages ซึ่งคือการส่งข้อมูลติดต่อกันระหว่าง Object โดย Messages เกิดขึ้นเมื่อ Method ใน Object ส่งข้อมูลไปยัง อีก Method ในอีก Object หนึ่ง

CLASSES

เนื่องจากมีหลายๆ Objects ที่คล้ายกันใน Database โดยถ้าสิ่งที่เหมือนกันนั้นคือ การตอบสนองต่อ Message เดียวกัน มี Methods เดียวกัน และมี Attributes ที่ใช้ชื่อเดียวกัน Objects ต่างๆ เหล่านี้ควรจะถูกจัดกลุ่มอยู่ในชนิดเดียวกันเพราะมีคุณสมบัติพื้นฐานเหมือนกัน เราจะเรียกการจัดกลุ่มนั้นว่า Class และจะเรียก Object ว่า Instance ของ Class

โดยที่ Instance จะเป็นส่วนขยายของคลาส และถูกเก็บอยู่ในส่วนของฐานข้อมูล ดังนั้นข้อมูลต่างๆ ที่เป็นตัวกำหนดคลาสจะถูกเข้าถึง และถ่ายทอดผ่านทางวัตถุ Instance และ Class Method

- Instance method คือ โอเปอเรชันที่ถูกประยุกต์ใช้งานกับ Instance แต่ละตัวในคลาสที่กำหนด
- Class method คือ วิธี (Method) ที่ถูกประยุกต์ใช้กับส่วนของคลาสทั้งหมด เช่น TotalEmployees() ที่ใช้คำนวณหาจำนวนพนักงานทั้งหมด

ATTRIBUTES

คือโครงสร้างของฟิลด์ภายในทูเปิลที่ให้สื่อทางด้านความหมายของแต่ละทูเปิล สำหรับแอททริบิวท์จะถูกแปลงไปเป็น Atomic Object หรือกลุ่มของ Atomic Object ซึ่งแอททริบิวท์จะแบ่งออกเป็น 2 ประเภท คือ

- Class Attributes คือ แอททริบิวท์ของคลาสนั้นๆ ที่ถูกเข้าถึงโดยตัวคลาสเอง Instance และ Subclass ของตัวคลาส
- Instance Attributes คือ แอททริบิวท์ที่เข้าถึงโดย Instance ของคลาสที่กำหนดเท่านั้นซึ่งจะคล้ายคลึงกับแอททริบิวท์ในโครงสร้างอื่นๆ โดยที่ Instance Attributes นั้นจะมีค่าตายตัวของมันเอง (Default Instance Attributes) หรือ อาจจะมีค่าสากลที่ประยุกต์ใช้กับแต่ละ Instance ของคลาสได้ (Shared Instance Attributes)

RELATIONSHIP (ความสัมพันธ์ระหว่าง Object)

- **Association** เป็นการอธิบายถึงความสัมพันธ์ระหว่างโครงสร้างโดยทั่วไป โดยอ้างถึงความสัมพันธ์ทั้งสองด้านและมีการระบุชื่อความสัมพันธ์อย่างชัดเจน ความสัมพันธ์แบบนี้ประกอบด้วย Constraint และ Link
- **Aggregation** เป็นรูปแบบความสัมพันธ์อย่างหนึ่งของ Association ในการกำหนด Parts ในส่วนประกอบนั้นโดยมีการแยกระหว่าง Whole กับ Parts ซึ่งจะได้ความสัมพันธ์แบบ Whole-Part หรือ a-part-of หรืออาจเรียกอีกแบบหนึ่งว่ามีความสัมพันธ์แบบ has-a-relationship เป็นผลให้ Super Class มีอิทธิพลต่อการดำรงอยู่ของ Sub Class คือถ้า Super Class ถูกทำลาย Sub Class จะถูกทำลายไปด้วยโดยอัตโนมัติ นอกจากนี้การพิจารณาถึงการแบ่งระดับของ Aggregation นั้นขึ้นอยู่กับความต้องการของระบบ
- **Generalization** เป็นความสัมพันธ์ในการจัดกลุ่มสิ่งที่เหมือนกันระหว่าง Class ของ Object หรือ Class ขึ้นมาเป็นอีก Class ของ Object หนึ่ง โดยพิจารณาจาก Structure หรือ Behavior ในกลุ่มของ Class ของ Object นั้น จะมีลักษณะเป็นแบบทั่วไปไม่ชี้เฉพาะเจาะจง
- **Depends on** เป็นความสัมพันธ์ที่ชี้ให้เห็นถึง Class ของ Object ที่ร้องขอ ขึ้นอยู่กับ Class ของ Object ปลายทาง เช่นใน Class หนึ่งสามารถมี Method ที่ขึ้นอยู่กับ Class อื่นเวลาทำงานจะมีการอ้างถึง Class นั้นด้วย

OPERATIONS

ถูกกำหนดขึ้นมาสำหรับคลาส และสืบทอดผ่านทาง Instance โดยแบ่งออกได้ 2 ทางดังนี้

- **Type Specific vs. Generic**
  - Type Specific Operation คือ โอเปอเรชันที่ถูกประยุกต์ใช้กับ Atomic Value
  - Generic Operation คือ โอเปอเรชันที่ประยุกต์ใช้กับ Composite Values
- **Predefined vs. User-Defined**
  - Predefined Operation คือ โอเปอเรชันที่ถูกติดตั้งโดยนักพัฒนาระบบฐานข้อมูล ประยุกต์ใช้กับ Atomic Value บางชนิด
  - User-Defined Operation คือ โอเปอเรชันที่ถูกนิยามโดยผู้ใช้

2.1.2 ลักษณะพื้นฐานของฐานข้อมูลเชิงวัตถุ ประกอบด้วย

COMPLEX OBJECT

คือ Object ซึ่งประกอบไปด้วยกลุ่มของ Attributes ที่ซึ่ง Attribute นั้นเป็น Object ที่ถูกสร้างขึ้นมาก่อนหน้า คุณสมบัตินี้ช่วยเพิ่มความสามารถในการสร้างรูปแบบ Object ใหม่ๆ ที่ซับซ้อนได้

USER-DEFINDED OPERATIONS

ลักษณะที่แตกต่างของภาษาเชิงวัตถุ คือ การจัดเตรียมให้กับผู้ใช้ในการกำหนดคลาส และความสัมพันธ์ของโอเปอเรชัน ซึ่งลักษณะนี้จะเป็นประโยชน์สำหรับคลาสได้

โครงสร้างการจัดการเชิงวัตถุนั้น โอเปอเรชันสามารถกำหนดโดยผู้ใช้ ซึ่งโอเปอเรชันเหล่านั้นคือ

- วิธีการต่างๆ ที่ซ่อนอยู่ภายใต้วัตถุ
- ผู้ใช้ได้กำหนดโอเปอเรเตอร์ขึ้นมาสำหรับ User-Define Type

OBJECT IDENTITY

คือคุณสมบัติของ Object ที่ใช้แสดงถึงความแตกต่างของแต่ละ Object ออกจากกันโดย Object Identity (OID) ถูกใช้เป็นสัญลักษณ์ที่แสดงถึงลักษณะเฉพาะตัว ของ Object โดยทั่วไปแล้ว Object Identity นั้นจะถูกสร้างขึ้นเองโดยระบบ มีลักษณะอยู่ในรูปของพอยน์เตอร์ ใช้อ้างอิงไปยัง Object ต่างๆ

INHERITANCE

คือการสืบทอดคุณสมบัติต่างๆ จาก Class หนึ่ง ไปยังอีก Class หนึ่ง โดย Class ใหม่ที่ได้จะได้รับการถ่ายทอดคุณสมบัติทั้ง Attributes และ Methods มาจาก Class เดิม แต่จะมี Attributes หรือ Method ที่เพิ่มเติมขึ้นมาอีกจาก Class เดิม โดยเรียก Class ต้นแบบว่า Super Class เรียก Class ใหม่ว่า Sub Class โดยเรียกสัมพันธ์แบบนี้ว่า is-a relationship

- Class และ Instance การขยายความของคลาสจะนำเสนอโดยกลุ่มของ Instance ที่มีคุณสมบัติเดียวกัน นั่นคือ ความสัมพันธ์ของคลาสแม่ (Super Class) และคลาสลูก (Sub Class) จะถูกนำเสนอออกมา โดยผ่านทาง Instance
- Attributes ซึ่งคลาสลูกจะสืบทอดแอททริบิวท์ต่างๆ จากคลาสแม่โดยตรง นั่นคือถ้ามีการเปลี่ยนแปลงใดๆ ที่คลาสแม่แล้ว คลาสลูกจะได้รับการเพิ่มแอททริบิวท์นั้นโดยตรงการสืบทอดแอททริบิวท์ทั้งชื่อ ข้อจำกัด และความสัมพันธ์ต่างๆ ภายในแต่ละแอททริบิวท์ ซึ่งข้อจำกัดที่คลาสลูกได้รับนั้นอาจจะเป็นเพียงส่วนหนึ่งของคลาสแม่ หรือได้รับการถ่ายทอดมาทั้งหมด ซึ่งจะรวมทั้งค่า Default Value ด้วย
- Relationships การสืบทอดคุณสมบัติความสัมพันธ์จะรวมในส่วนของชื่อ ชนิดของความสัมพันธ์ ข้อจำกัดต่างๆ และความสัมพันธ์ที่เกี่ยวข้องกับคลาสอื่นๆ
- Operations โอเปอเรชันจะถูกสืบทอดโดยคลาสลูก ซึ่งจะเกี่ยวข้องกับ
  1. คลาสลูกสามารถเพิ่มโอเปอเรชันเข้าไปได้โดยอัตโนมัติ
  2. โอเปอเรชันของคลาสลูกสามารถเป็นส่วนขยายของคลาสแม่ได้

OBJECT PERSISTENCE

เป็นทั้งคุณสมบัติพื้นฐาน และลักษณะเด่นของระบบฐานข้อมูลเชิงวัตถุ Persistency คือ ความสามารถของวัตถุที่จะสืบอดชีวิตอยู่ภายในระบบในขณะที่ทากรประมวลผลโปรแกรม ดังรูปการต่อไปนี้

- Persistent Object จะไม่ถูกทำลายเมื่อโปรแกรมได้ถูกกำหนดจุดสิ้นสุดไว้ ซึ่งจะเกี่ยวข้องกับ Internal Garbage Collection
- Persistent Object จะประกอบด้วยชื่อที่เป็นสากล ซึ่งสามารถถูกอ้างถึงได้โดยโปรแกรมอื่นๆ

ผู้ใช้สามารถกำหนดคลาสของวัตถุที่เป็น Persistent ในบางภาษาได้โดยการกำหนดในรูปแบบท้องถิ่นได้เท่าเทียมกับการกำหนดที่เป็นสากล

COMPUTATIONAL COMPLETENES

ระบบจะมีการคำนวณที่สมบูรณ์ ถ้าระบบนั้นกำหนดการใช้ภาษาที่อนุญาตให้มีการเรียกใช้ฟังก์ชัน หรืออัลกอริทึมที่โค้ดด้วย Data Manipulation Language (DML) ของระบบ

VERSION

การเปลี่ยนแปลงของวัตถุจากสถานะหนึ่งไปยังสถานะอื่นๆ ซึ่ง Version ก็คือกระบวนการในการเก็บผลลัพธ์ของวัตถุที่มีการเปลี่ยนแปลงในแต่ละสถานะอย่างรวดเร็ว คลาสที่อนุญาตให้มีการทำ Version เรียกว่า Version Class และทุกๆ Instance ที่อยู่ภายใต้คลาสนั้นจะเรียกว่า Version Instance ก็คือ การทำ Version ให้กับคลาสที่มีความสัมพันธ์กัน การกำหนดให้คลาสเป็น Version class จะต้องทำการกำหนดก่อน Version Instance ซึ่งการกำหนด Version Instance จะเกิดภายใต้ 3 สถานะ คือ

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

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

Version Instance ถูกนำเสนอจากสถานะหนึ่งไปอีกสถานะหนึ่ง นั่นคือจาก transient ไปยัง working และไปสู่ Released state และการนำเสนอการใช้งาน Version Instance กระทำได้ 2 วิธีการ ดังนี้

- การอ้างอิง Version Instance โดยตรงโดยการใช้ Object Identifier
- การอ้างอิงกับ Generic Object และเรียกใช้งาน Version ล่าสุด

ENCAPSULATION

คือการรวม Attributes และ Methods ของ Object เข้ามาอยู่ด้วยกัน และทางเดียวที่ Object อื่นจะเข้าถึง Attributes ของ Object ได้ ต้องผ่าน Interface ของ Object ที่เปิดไว้ให้ โดย Interface ของ Object ประกอบไปด้วยกลุ่มของ Operations ซึ่งสามารถเข้าถึง Object นั้นๆ ได้

POLYMORPHISM

คือการที่คลาสหนึ่งๆ สามารถแปรเปลี่ยนไปได้หลายรูปแบบ ขึ้นกับสภาพแวดล้อม หรือสถานการณ์ในขณะนั้น ความสามารถอีกอย่างหนึ่งของ Polymorphism ก็คือ สามารถกำหนดการดำเนินการใหม่ให้กับตัวดำเนินการ หรือแม้แต่ฟังก์ชันได้ เรียกว่าการทำ Overloading คือการเรียกใช้ฟังก์ชัน หรือตัวดำเนินการเดิมให้ไปทำงานอีกอย่างหนึ่งได้

2.2 ข้อดีของรูปแบบข้อมูลเชิงวัตถุ

EXTENSIBILITY

สำหรับ User-defined types และ Overloaded Operation นั้น ส่วนของคลาส และการจัดการความสัมพันธ์จะสามารถปรับเปลี่ยน หรือเพิ่มได้ตามความต้องการ การเปลี่ยนแปลงหรือการเพิ่มนี้จะถูกจัดการให้กับคลาสที่มีความสัมพันธ์กันโดยอัตโนมัติ นั่นคือคลาส และโอเปอเรชันใหม่ๆ สามารถรวมเข้าไปในระบบ โดยปราศจากผลกระทบกับวัตถุอื่นๆ ภายในระบบ

INFORMATION HIDING

การซ่อนข้อมูลวัตถุจะอนุญาตให้ข้อมูล และโอเปอเรชันถูกกำหนดอยู่ภายในโครงสร้างข้อมูลเดี่ยว (Single model) นั่นคือ ข้อมูลจะถูกกำหนดเฉพาะที่ (Local) และถูกซ่อนอยู่ภายในวัตถุเอง เมื่อมีการเชื่อมต่อเกิดขึ้นจะมีการแสดงออกมาโดยอัตโนมัติ

FLEXIBILITY OF TYPE DEFINITION

ความสามารถที่อนุญาตให้ผู้ใช้กำหนดคลาส และโอเปอเรชันใหม่จะให้ความยืดหยุ่น และการควบคุมให้กับผู้ใช้ ซึ่งเป็นประโยชน์กับการประยุกต์ใช้ฐานข้อมูลใหม่ๆ ในการปรับเปลี่ยนข้อมูล

CODE REUSABILITY

จากคุณสมบัติการถ่ายทอดจากคลาสแม่ไปยังคลาสลูก ทำให้ไม่ต้องมีการกำหนดวิธีการและโอเปอเรชันขึ้นใหม่ในแต่ละคลาส ซึ่งจะช่วยลดความซ้ำซ้อน และการทำงานที่ไม่ควบคู่กัน

2.3 ตัวอย่างของฐานข้อมูลเชิงวัตถุ

ในฐานข้อมูลเชิงวัตถุ Complex Object จะถูกสร้างมาจาก Object ที่ง่ายๆ โดยการใช้ Constructor ช่วย สำหรับรูปแบบที่ง่ายที่สุดของ Object คือ Atomic Type เช่น Integer, Character และ Boolean ซึ่งเป็นรูปแบบที่ไม่มีส่วนประกอบย่อยเลย สำหรับรูปแบบ Constructor ของ Complex Object นั้นจะรวม Tuples, Sets, Bags, Lists และ Arrays โดยการสร้าง Complex Object นี้จะเป็นการซ่อนความซับซ้อนของโครงสร้างข้อมูลต่อ Users ใน Complex Object นี้ค่าของ Non-Atomic Objects จะอ้างถึง Objects อื่นๆ โดยตัว Object Identifiers ตัวอย่างข้างล่างจะแสดงถึงตัวอย่างของฐานข้อมูลเชิงวัตถุตามรูปแบบของ ODL ของ ODMG1

```
interface Department ( extent Departments  keys dno, name ): persistent
{
        attribute Short dno;
        attribute String name;
        attribute Instructor head;
        relationship Set<Instructor> instructors
```

```
            inverse Instructor::dept;
        relationship Set<Course> courses_offered
            inverse Course::offered_by:
};

interface Instructor ( extent Instructors  key ssn ): persistent
{
        attribute Short ssn;
        attribute String name;
        attribute Integer salary;
        attribute String rank;
        relationship Department dept
            inverse Department::instructors;
        relationship Set<Cource> teaches
            inverse Course::taught_by:
};
```

บทที่ 3
ฐานข้อมูลอิงเวลา

Temporal Database

โดยปกติข้อมูลที่เก็บอยู่ใน Database จะเป็นข้อมูลล่าสุดที่มีการ Update แล้ว และจะไม่สามารถดูข้อมูลเก่าๆ ก่อน Update ได้เลย เพราะว่าข้อมูลเก่าเหล่านั้นได้ถูกข้อมูลใหม่ Update ทับไป ซึ่งประโยชน์ของการที่จะเข้าไปดูข้อมูลเก่าๆ นั้นเช่น ในทางการแพทย์ต้องการดูข้อมูลประวัติการรักษาพยาบาล การจ่ายยาหรือ ทางด้านธุรกิจก็เช่น ต้องการดูข้อมูลการลงทุนในอดีตในช่วงเวลาต่างๆ หรือดูประวัติการส่งของ ฯลฯ จริงๆ แล้วฐานข้อมูลเชิงสัมพันธ์ในปัจจุบันก็สามารถเก็บประวัติข้อมูลเก่าๆ ได้ แต่ในการออกแบบจะทำให้มีความซ้ำซ้อนของข้อมูลและมีความซับซ้อนในการสืบค้นข้อมูล จากปัญหาดังกล่าวทำให้เกิดความต้องการฐานข้อมูลซึ่งสามารถสืบค้นข้อมูลเก่าๆ ได้ สนับสนุนข้อมูลซึ่งขึ้นอยู่กับเวลาต่างๆ ไม่ว่าจะเป็นอดีตปัจจุบันหรืออนาคต และสามารถที่จะสืบค้นข้อมูลได้อย่างง่ายดายไม่ซับซ้อน ซึ่งเราเรียกฐานข้อมูลประเภทนี้ว่าฐานข้อมูลอิงเวลา (Temporal Database)

Temporal Database ในปัจจุบัน

ในทศวรรษนี้มีงานวิจัยในด้าน Temporal Databases ที่ก่อให้เกิดการพัฒนาของ Temporal Query Language อย่างแพร่หลายโดยมีพื้นฐานบน Relational Languages เช่น TQUEL หรือ Temporal Extensions ของ SQL และตัวที่มีชื่อเสียงที่สุดคือ TSQL2 และมีอีกมากเช่น ATSQL2 และในปัจจุบันได้มีการเสนอ Temporal Extension ของ SQL3 ที่ได้ ISO และ ANSI Standardization Committees-SQL/Temporal

ทุกข้อเสนอในปัจจุบันยอมรับการใช้ Timestamping ใส่ลงไปใน Tuples ทั่วไป ๆ ณ เวลาในขณะนั้น ซึ่งสนับสนุนความต้องการต่อการใช้ข้อมูลจำนวนมากใน Tuple นั้นๆ และ ควรมี Repeated (ซ้ำซ้อนได้) สำหรับทุก ๆ ช่วงเวลาที่ซึ่งจะถูกแสดงตามความจริง ใน Tuple ที่ถูกจองไว้แทนที่เป็นการเข้ารหัสของชุดของช่วงเวลานั้น (โดยปกติเรียกว่า *Periods* ของ Validity)

3.1 เวลาในฐานข้อมูล

ตามธรรมชาติแล้วเวลามีลักษณะต่อเนื่อง อย่างไรก็ตามในการพิจารณาเวลาในทางฐานข้อมูลโดยทั่วไปสามารถแบ่งประเภทของเวลาออกได้เป็น 2 ประเภทคือ เวลาที่ต่อเนื่อง (Continuous Time) และเวลาที่ไม่ต่อเนื่อง (Discrete Time) เวลาที่ต่อเนื่องนี้มีลักษณะคล้ายกับเลขจำนวนจริง ในขณะที่เวลาที่ไม่ต่อเนื่องนั้นจะคล้ายกับเลขจำนวนเต็มซึ่งเป็น Subset ของเลขจำนวนจริง ซึ่งเวลาทั้งสองมีลักษณะเป็นลำดับเส้นตรงคือ ณ เวลาใดๆ ในสองจุดของเวลาที่แตกต่างกันเช่น เวลา t1 และ t2 นั้นอาจเป็นไปได้ว่า t1 มาก่อน t2 หรือ t2 มาก่อน t1 โดยที่แต่ละจุดของเวลามีขนาดที่เท่าๆกัน และเป็นลำดับโดยให้แทนด้วยตัวอักษร T นั่นคือ T = {0,1,2...now....} โดยที่ 0 แทนจุดเริ่มต้นของเวลา และ now เป็นค่าพิเศษที่ใช้แทนเวลาในปัจจุบันซึ่งค่าของ now จะเพิ่มขึ้นตามเวลาที่ผ่านไป โดยที่จุดของเวลาใดๆ หลังจาก now ถือว่าเป็นเวลาในอนาคตทั้งสิ้น และช่วงเวลา(Interval) คือจุดของเวลาซึ่งอยู่ต่อเนื่องกันและถูกจำกัดด้วยจุดเริ่มต้นและจุดสิ้นสุดของเวลา ในช่วงเวลาปิด [b, c] จะประกอบด้วยทุกจุดของเวลาจาก b ไปจนถึง c และในลักษณะของช่วงเวลากึ่งเปิด [b, c) ก็เช่นเดียวกันคือประกอบด้วยทุกจุดของเวลาจาก b ไปจนถึง c แต่ไม่รวมจุดของเวลา c โดยที่ subset ของ T จะเรียกว่า Temporal Element เป็นผลลัพธ์ของการ Union ของช่วงเวลาต่างๆ ดังกล่าวนั่นเอง ช่วงเวลาหรือ Temporal Element ใดๆ ซึ่งประกอบด้วย now จะไม่คงที่คือจะมีขนาดที่ขยายออกเมื่อค่าของ now เพิ่มขึ้นตามเวลา ซึ่งใน Temporal Data Model นั้นได้นิยามเวลาไว้ดังนี้

3.1.1 ประเภทของเวลาใน Temporal Database

ใน Temporal Database จะนิยามเวลาเป็น 3 ชนิดคือ

- Valid Time คือเวลาซึ่งข้อมูลต่างๆ ได้ถูกบันทึกตามเวลาจริงๆ ในโลก
- Transaction Time คือเวลาซึ่งข้อมูลต่างๆ ถูกจัดเก็บลงใน Database
- User define Time คือเวลาซึ่ง User เป็นผู้กำหนดขึ้นเอง แล้วแต่การใช้งานของ User ซึ่งเวลาประเภทนี้มีอยู่แล้วใน Database ปัจจุบัน

ซึ่ง Valid Time และ Transaction Time นี้ไม่มีความสัมพันธ์กันแต่อย่างไร ในการใช้งานนั้นระบบฐานข้อมูลใดๆ อาจให้การสนับสนุนการใช้งานเวลาประเภทใดประเภทหนึ่ง หรือทั้งสองประเภทร่วมกันได้ ซึ่งจากประเภทของเวลาดังกล่าวทำให้เกิดการจำแนกประเภทของฐานข้อมูลที่แตกต่างกันตามลักษณะการสนับสนุนการใช้งานเวลาในลักษณะต่างๆ ดังกล่าวข้างต้น

3.1.2 ประเภทของฐานข้อมูลใน Temporal Database

ในการใช้งานระบบฐานข้อมูลจะเลือกเวลาประเภทใดก็ได้ และจากประเภทต่างๆ ของเวลาทำให้เกิดประเภทของฐานข้อมูลที่แตกต่างกัน ดังนี้

- Snapshot Database คือฐานข้อมูลซึ่งเก็บเฉพาะข้อมูล ณ เวลาใดเวลาหนึ่งเท่านั้น ฐานข้อมูลประเภทนี้ไม่มี Valid Time และ Transaction Time สำหรับฐานข้อมูลเชิงสัมพันธ์เป็น Snapshot Database
- Valid-Time Database คือฐานข้อมูลซึ่งเก็บประวัติของข้อมูลพร้อมทั้งเวลาที่เกิดขึ้นแบบ Valid-Time
- Transaction-Time Database คือฐานข้อมูลซึ่งเก็บข้อมูลพร้อมทั้งเวลาแบบ Transaction-Time ซึ่งฐานข้อมูลประเภทนี้เราสามารถ Rollback กลับไปยังเวลาใดๆ ได้อย่างง่ายดาย เพราะเวลาที่ใช้เก็บเป็นเวลาของฐานข้อมูลอยู่แล้ว
- Bitemporal Database เป็นฐานข้อมูลที่เก็บทั้ง Valid-Time และ Transaction-Time และจะมีคุณสมบัติต่างๆ ของทั้ง Valid-Time และ Transaction-Time รวมกัน

3.2 การขยายความสามารถของฐานข้อมูลเชิงสัมพันธ์เพื่อใช้งานฐานข้อมูลอิงเวลา

เนื่องจากในปัจจุบันระบบจัดการฐานข้อมูลที่ได้รับการยอมรับอย่างกว้างขวาง และได้รับการนำไปประยุกต์ใช้งานอย่างแพร่หลายคือ ฐานข้อมูลเชิงสัมพันธ์ ดังนั้นจึงเป็นการง่ายที่จะทำความเข้าใจกับทฤษฎีต่างๆ และการใช้งานฐานข้อมูลทางด้านเวลา ผ่านทางฐานข้อมูลเชิงสัมพันธ์

Temporal Database เป็นเพียงแนวคิดในการนำข้อมูลมาสัมพันธ์กับเวลา ไม่มี Data Model เป็นของตัวเอง โดยทั่วไปจะใช้ Data Model ของฐานข้อมูลเชิงสัมพันธ์เป็นตัวแทนเพื่อใช้งาน Temporal Database ซึ่งจะเป็นเสมือนว่า Temporal Database เป็นส่วนขยายความสามารถของฐานข้อมูลเชิงสัมพันธ์ในส่วนของการจัดการกับเวลา และ Temporal Database ก็ยังสามารถนำไปใช้ขยายความสามารถให้กับ Data Model อื่นๆ ได้อีก เช่น นำไปขยายความสามารถของฐานข้อมูลเชิงวัตถุ

3.2.1 ประเภทของ Temporal Database

ในการขยายความสามารถในการจัดการกับเวลามีอยู่ 2 แนวความคิดคือ การนำ Timestamp ไปผูกติดกับข้อมูลในแต่ละ Tuple (Tuple Timestamping) กับการนำ Timestamp ไปผูกติดกับ Attribute ของ Schema (Attribute Timestamping) ในกรณีแรกความสัมพันธ์จะได้รับการเพิ่มคอลัมน์ของเวลาเพื่อใช้เก็บช่วงเวลาในการอ้างอิงข้อมูลในแต่ละแถวการเพิ่มนี้จะเพิ่มเพียงคอลัมน์เดียวโดยใช้เป็นช่วงเวลาหรือสองคอลัมน์โดยใช้เป็นเวลาเริ่มต้นกับเวลาสิ้นสุดก็ได้ ข้อดีคือ ความสัมพันธ์นั้นยังคงอยู่ในรูปแบบ 1NF ซึ่งทำให้สามารถใช้กับฐานข้อมูลเชิงสัมพันธ์ได้อย่างมีประสิทธิภาพ ดังตารางที่ 2.1

Name	Title	Salary	Time_Interval
Smith	Engineer	35K	9/97 - 9/98
Smith	Engineer	48K	9/98 - now
Brown	Analyst	32K	8/96 - 6/97
Brown	DBA	40K	6/97 - 1/99
Brown	DBA	46K	1/99 - now
Jones	CEO	80K	1/98 - now

ตารางที่ 2.1 Temporal Representations (Tuple Timestamp)

ส่วนในกรณีที่สองการเพิ่ม Timestamp เข้าไปยังคอลัมน์ทำให้ค่า Domain ของ Attribute รวมกับเซ็ตของเวลา และทำให้เป็น Non 1NF ซึ่งมีความซับซ้อนและยุ่งยากในการจัดการ แต่สามารถระบุเวลาของแต่ละ Attribute ได้ ดังตารางที่ 2.2

Name	Title Time_Interval	Salary Time_Interval
Smith	Engineer 9/97 - now	35K 9/97 – 9/98
		48K 9/98 - now
Brown	Analyst 8/96 – 6/97	32K 8/96 – 6/97
	DBA 6/97 – now	40K 6/97 – 1/99
		46K 1/99 - now
Jones	CEO 1/98 – now	80K 1/98 - now

ตารางที่ 2.2 Temporal Representations (Attribute Timestamp)

โดยรูปแบบนี้จะเหมาะแก่การนำไปใช้กับ Data Model ของฐานข้อมูลเชิงวัตถุ

เวลาใน Temporal Database ที่แสดงนี้เราไม่สนใจว่าเป็นเวลาชนิดใด Valid Time หรือ Transaction Time เพราะเราสามารถกำหนดที่ Abstract Data Type ในขั้นตอนเริ่มต้นของการสร้างได้ และไม่ว่าจะเป็นเวลาชนิดใดก็ไม่มีผลกระทบต่อ Data Model

จากทั้ง 2 แนวความคิดนี้ ทำให้ Temporal Database ถูกแบ่งออกเป็น 2 แนวทาง คือ

1. **Object Versioning (Object-Oriented Model)** หรือ Tuple Timestamping (Relational Model) แต่ละ Object หรือ Tuple นั้นจะมีช่วงระยะเวลากำหนดซึ่งครอบคลุมทั้ง Object หรือ Tuple นั้นๆ
2. **Attribute Versioning (Object-Oriented Model)** หรือ Attribute Timestamping (Relational Model) แต่ละ Attribute จะถูกกำกับด้วยเวลา (value, interval)

3.2.2 ตัวอย่าง Applications ของ Temporal Database

- ด้านการศึกษา เช่น การออกหนังสือรับรองการศึกษาในเทอมก่อนและเทอมปัจจุบัน หรือออกเกรดสำหรับเทอมก่อน
- ด้านบัญชี เช่น ใบเสร็จอะไรที่มีการออกและออกให้เมื่อไร, บัญชีที่ค้างชำระและเงินหมุนเวียนหมดอายุเมื่อไร

- งบประมาณ เช่น งบประมาณก่อนและหลังโปรเจค ว่างบประมาณในปีต่างๆ หรือเดือนต่างๆ เป็นเช่นไร
- ด้าน Data Warehouse เช่น การวิเคราะห์แนวโน้มของประวัติต่าง ๆ สำหรับการทำระบบช่วยตัดสิน
- และอื่น ๆ

ทั้งหมดของตัวอย่างจะเป็นการยากที่จะทำเป็น Application ถ้าไม่ใช้การบริหารข้อมูลด้วย Temporal

Application ดังกล่าวนั้นจะได้ประโยชน์จากการใช้ Temporal ที่สนับสนุนใน DBMS ที่จะทำให้การพัฒนา Application นั้นมีประสิทธิภาพมากขึ้น และเพิ่มศักยภาพของการทำงาน

3.2.3 ข้อดีของการใช้ฐานข้อมูลอิงเวลา

(เทียบกับกรณีที่ใช้ฐานข้อมูลเชิงสัมพันธ์มาทำงานด้าน Temporal)

- โครงสร้างของ Semantics จะง่ายขึ้นมากเมื่อนำเสนอด้วย Temporal Query Language
- ผู้ใช้สามารถเขียน Temporal Queries ในการประกาศตัวแปรที่เข้าใจได้ง่าย
- มันสนับสนุนการให้ความหมายของการ Duplicate, Semantics และ Aggregation
- Queries ที่ใช้จะเป็นภาษาที่ทำให้ประสิทธิภาพบนการใช้ Temporal Relations ดีขึ้น และทำให้มีการเขียน Code ที่สั้นขึ้น ประโยชน์ที่ตามมาก็คือ ง่ายต่อการเข้าใจ ง่ายต่อการตรวจสอบความถูกต้องและง่ายต่อการบำรุงรักษา

บทที่ 4
ระบบจัดการฐานข้อมูลเชิงวัตถุ

OODBMS (Object-Oriented Database Management Systems) ทั่วๆ ไปได้พยายามนำเอาแนวความคิดของการโปรแกรมเชิงวัตถุไปใช้ และเพิ่มคุณลักษณะบางอย่างในการทำงานที่จำเป็นเพื่อสนับสนุนข้อมูลขนาดใหญ่, การใช้งานร่วมกัน รวมทั้งการคงอยู่ของออบเจ็กต์ที่เก็บ (Persistent Object Stores) ซึ่งคุณลักษณะดังกล่าวยังรวมถึงการสนับสนุนการร้องขอแบบ Set-Oriented หรือการสอบถามข้อมูล (Query) ประสิทธิภาพในการประมวลผลบนโครงสร้างของ Secondary storage ขนาดใหญ่ การควบคุมความถูกต้อง (Concurrency Control) และการเรียกข้อมูลกลับ (Recovery)

4.1 วิวัฒนาการของ OODBMS

ในการพัฒนา Software ระบบฐานข้อมูลในอดีต ส่วนมากการเก็บข้อมูลนั้นอ้างอิงอยู่กับระบบฐานข้อมูลแบบ RDBMS ซึ่งเริ่มมีการพัฒนามาตั้งแต่ปี ค.ศ. 1980 และการพัฒนาทางด้าน Programming ของระบบฐานข้อมูลได้เริ่มเปลี่ยนมาเป็น Object - Oriented Programming Languages ซึ่งเป็นการพัฒนา Software ในรูปแบบของ Object Model แทนรูปแบบเดิม ซึ่งเป็นการพัฒนาในรูปแบบของ Relational Model และจากจุดนี้เองทำให้เกิดการเปลี่ยนแปลงขึ้น และนำไปสู่การพัฒนา Software ระบบฐานข้อมูลแบบใหม่ขึ้นมา คือ ระบบการจัดการฐานข้อมูลในรูปแบบของ Objects (Object - Oriented Database Management System) ซึ่งสนับสนุนในเรื่องของการเก็บข้อมูลในลักษณะของ Objects และสนับสนุน Object - Oriented Programming Languages โดยตรง

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

เนื่องจากความเจริญก้าวหน้าทางด้านเทคโนโลยีในการพัฒนาระบบ Software โดยเฉพาะทางด้าน Programming มีการพัฒนามาใช้ Object - Oriented Programming Language ทำให้เกิดปัญหาเกี่ยวกับการเก็บข้อมูลลงในระบบฐานข้อมูลในรูปแบบของ Relational เพราะว่า Relational Database เป็นฐานข้อมูลในรูปแบบของ Relational Model ซึ่งมีการเก็บข้อมูลต่างๆอยู่ในรูปแบบของตารางที่มีความสัมพันธ์กัน ส่วนการพัฒนาโดยใช้ Object - Oriented Programming เป็นการมองลักษณะของข้อมูลในรูปแบบของ Object Model โดยที่แต่ละ Object มีความสัมพันธ์กันโดย

Reference ทำให้มีความลำบากและยุ่งยากในการเก็บข้อมูลในลักษณะของ Objects ลงในฐานข้อมูลที่มีลักษณะเป็น Relational

ดังนั้นระบบการจัดการฐานข้อมูลในรูปแบบของ Object (OODBMS) ซึ่งเป็นระบบการจัดการฐานข้อมูลแบบ Object Model และให้การสนับสนุนการพัฒนา Software ระบบฐานข้อมูลในลักษณะของ Object - Oriented Programming จึงเป็นสิ่งที่ดีและเหมาะสมกว่าระบบฐานข้อมูลแบบ Relational

4.2 สิ่งที่ OODBMS ให้การสนับสนุน

ระบบการจัดการฐานข้อมูลแบบ Objects (OODBMS) ที่แท้จริง ควรให้การสนับสนุนคุณสมบัติพื้นฐานสำหรับ Objects ดังต่อไปนี้

ENCAPSULATION

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

INHERITANCE

Inheritance คือ การที่ Class หนึ่งมีการสืบทอดคุณสมบัติทุกอย่างมาจากอีก Class หนึ่ง ซึ่ง Class ที่สืบทอดคุณสมบัติมานั้นสามารถมีคุณสมบัติอื่นเพิ่มเติมนอกเหนือจาก Class ที่ถูกสืบทอดคุณสมบัติได้

POLYMORPHISM

Polymorphism คือ การที่ Objects ที่ต่างกันสามารถแสดงพฤติกรรมตอบสนองที่ต่างกันเมื่อได้รับ Message ชนิดเดียวกันเช่น ในกรณีของการแสดงข้อมูลของ Objects 2 Objects ซึ่งทั้ง 2 Objects ต่างก็ได้รับ Message ชนิดเดียวกัน คือ display() ซึ่ง Object หนึ่งอาจจะเป็นการแสดงข้อมูลในรูปแบบของรายละเอียดของข้อมูล (Text) ในขณะที่อีก Object หนึ่งอาจจะเป็นการแสดงข้อมูลในรูปแบบของรูปภาพ (Graphic)

Example *การจัดเก็บข้อมูลของ Student และ Employee ลงในฐานข้อมูล*

```
public class Student {
        public class Employee {
                // Attribute declaration// Attribute declaration
                ::
                Student s = new Student();Employee e = new Employee();
                // Set attribute value to object// Set attribute value to object
                s.set_name(name); e.set_name(name);
                ::
                s.save(); e.save();
        }
}
```

จากตัวอย่างจะเห็นว่าทั้ง Student class และ Employee class ต่างก็ทำการจัดเก็บข้อมูลโดยเรียก Method เดียวกัน คือ save() เพียงแต่เป็นการจัดเก็บข้อมูลของ Object คนละ class กัน

OBJECT IDENTITY

Object แต่ละ Object ใน OODBMS จะมี Logical Object Identifier (LOID) เป็นเลขที่บอก ID ของ Object นั้น โดยที่ LOID จะไม่ซ้ำกันเลย และจะไม่มีการนำกลับมาใช้ใหม่ถึงแม้ว่า Object นั้นจะถูกลบออกไปจากฐานข้อมูลแล้วก็ตาม

ในทางกลับกันเมื่อมีการสร้าง Object ใหม่ขึ้นมา ก็จะมีการกำหนด LOID ให้กับ Object นั้นด้วยโดยที่ LOID นี้จะไม่มีการเปลี่ยนแปลง ถึงแม้ว่าจะมีการนำ Object นั้นไปเก็บยังฐานข้อมูลตัวอื่นก็ตาม

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

1. ส่วนแรกเป็นตัวเลข 16-bit ที่แสดงถึงเลขของ Database ที่ไม่ซ้ำกัน

2. 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

REFERENCES AMONG OBJECTS

OODBMS มีการอ้างอิงถึง Objects ต่างๆ โดยใช้ Links ซึ่ง Link ของ Object ใน Database แทนด้วย LOID ของ Object

BASIC DATABASE OPERATIONS

OODBMS มี Operations พื้นฐานที่สามารถทำได้ดังนี้

1. Storing Objects

Storing Object คือ การจัดเก็บข้อมูล Object ลงในฐานข้อมูล ซึ่งเมื่อเปรียบเทียบกันระหว่าง Relational Database (RDBMS) กับ Object Database(ODBMS) แล้ว ODBMS จะทำการจัดเก็บข้อมูลได้ง่ายกว่าและไม่ค่อยยุ่งยาก

ในการเก็บข้อมูลลงใน RDBMS นั้นไม่สามารถเก็บลงไปได้โดยตรง ต้องมีการแปลงข้อมูลจาก Object ให้อยู่ในรูปของ Tables ก่อน โดยการผ่านชั้น Mapping Layer ก่อน แล้วจึงทำการเก็บลงในฐานข้อมูล ซึ่งค่อนข้างจะยุ่งยาก

ในทางตรงกันข้ามสำหรับ ODBMS นั้นสามารถทำการเก็บข้อมูลในรูปของ Object ลงไปได้โดยตรงไม่ต้องทำการ Mapping ก่อน

Example Create Person Object and save to Database

```
public class Create Person {
        public static void main(String [] args) {
                if (args.length != 3) {
                        System.out.println("Usage  : java  CreatePerson  <database>
                        <name> <age>");
                        System.exit(1);
                }
                String database = args[0];
                String name = args[1];
                Int age = Integer.parseInt(atgs[2]);
                Properties prop = new Properties();
                prop.put("database", database);
                Capability capability = new NewSessionCapability();
```

```
            TransSession session = new TransSession(prop, capability);
            Person person = new Person(name, age);
            session.endSession();
        }
}
```

2. Changing Objects

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

3. Deleting Objects

Deleting Object คือ การลบ Objects ออกจากระบบฐานข้อมูล ซึ่งทำได้โดยการค้นหา Object ที่ต้องการจากระบบฐานข้อมูล เมื่อพบแล้วก็ทำการลบ Object นั้นตามปกติ เช่น ถ้าต้องการลบ Object ของ Person ออกจากระบบฐานข้อมูลสามารถทำได้ดังนี้

Example Delete Person name = 'John'

```
public class DeletePerson {
        public static void main(String [] args) {
            if (args.length !=2) {
                System.out.println("Usage : Java DeletePerson <database>
                <name>");
                System.exit(1);
            }
            String database = args[0];
            String name = args[1];
            Properties prop = new Properties();
            prop.put("database", database);
```

```
        Capability capability = new NewSessionCapability();
        TransSession session new TransSession(prop, capability);
        VQLQuery query = new VQLQuery(session, "select selfoid from
        Person where name = 'John'");
        Enumeration e = query.execute();
        if(!e.hasMoreElements()) {
                System.out.println("No Person Objects were found");
        } else {
                while (e.hasMoreElemments()) {
                        Person person = (Person) e.nextElement();
                        session.deleteObject(person);
                }
        }
        session.endSession();
    }
}
```

4. Query

ในการค้นหาข้อมูลจากฐานข้อมูล ทั้ง RDBMS และ ODBMS ใช้ Query ในการค้นหาข้อมูล ซึ่ง ODBMS จะได้เปรียบ RDBMS ในด้านต่อไปนี้

- **ความเร็วในการค้นหาข้อมูล**

เช่น ในกรณีที่ทำการค้นหาข้อมูลซึ่งมีความเกี่ยวข้องสัมพันธ์กับข้อมูลในตารางอื่นอีกหลายตาราง ซึ่งทำให้ข้อมูลค่อนข้างจะซับซ้อน สำหรับ RDBMS ทำให้ต้องเสียเวลามากในการค้นหา ส่วน ODBMS จะใช้เวลาน้อยกว่า เนื่องจากใน ODBMS Objects สามารถอ้างถึงกันโดยใช้ Link ดังที่ได้กล่าวมาแล้ว เพราะฉะนั้นในการค้นหาข้อมูลก็เพียงแต่ไปที่ Object ที่ต้องการ เมื่อได้แล้วก็สามารถดูจาก Link ของ Object นั้น เพื่อที่ไปหา Objectที่ต้องการได้ จึงทำให้เร็วกว่า RDBMS ในการค้นหาข้อมูล

- **ODBMS ช่วยลดค่าใช้จ่ายในเรื่องของการ Joining**

เนื่องจาก Objects ใน ODBMS 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 ซึ่งยากแก่การจัดการใน RDBMS เพราะฉะนั้นจึงต้องมีการสร้างใหม่ขึ้นมาอีกหนึ่งตาราง(Student/Course) เพื่อช่วยในการจัดการกับข้อมูลที่มีความสัมพันธ์ในลักษณะนี้ เพื่อที่จะทำให้ข้อมูลอยู่ในรูปแบบที่ง่ายขึ้น

ส่วนใน ODBMS เนื่องจากการเก็บข้อมูลอยู่ในรูปแบบของ Objects อยู่แล้ว และอาศัย Links (LOIDs) ในการอ้างถึง Objects ต่างๆ เพราะฉะนั้นจึงง่ายต่อการค้นหาและจัดเก็บข้อมูล

4.3 หลักเกณฑ์ในการพิจารณาระบบที่มีพื้นฐานอยู่บน OODBMS

เมื่อทำการเปรียบเทียบกับระบบฐานข้อมูลที่ใช้กันในปัจจุบัน หรือ RDBMS นั้นจะมีข้อแตกต่างกับ OODBMS อย่างน้อยดังนี้

- OODBMS สนับสนุนการกำหนดโครงสร้างข้อมูล และการทำงานโดยผู้ใช้เอง
- OODBMS สนับสนุนแนวความคิดเกี่ยวกับ Object Identity ซึ่งหมายความว่าออบเจ็กต์จะมีการชี้ หรือระบุอย่างอิสระทั้งภายในออบเจ็กต์เอง หรือระหว่างออบเจ็กต์ก็ตาม
- OODBMS สนับสนุนความสัมพันธ์ของออบเจ็กต์ทางตรง (Direct Object Relationships) ซึ่งหมายความว่าออบเจ็กต์ที่มีความสัมพันธ์กันสามารถแอ็กเซส (Access) ผ่านทางการกระทำที่เกิดขึ้นภายในออบเจ็กต์ที่สัมพันธ์กันดังกล่าว

พื้นฐานที่อยู่บน Object-Oriented DBMS จะมีหลักเกณฑ์ที่ใช้ในการพิจารณา 2 หลักเกณฑ์ด้วยกันคือ 1. ระบบต้องมีรูปแบบเชิงวัตถุ และ 2. ระบบต้องเป็น DBMS

จากหลักเกณฑ์ดังกล่าวสามารถอธิบายเพิ่มเติมได้โดย

หลักเกณฑ์ที่ 1. สามารถแปลงได้เป็น 8 คุณลักษณะดังนี้ คือ Complex Object, Object Identity, Encapsulation, Types or Classes, Inheritance, Overriding Combined with Late Binding, Extensibility และ Computation Completeness

หลักเกณฑ์ที่ 2. สามารถแปลงได้เป็น 5 คุณลักษณะดังนี้คือ Persistence, Secondary Storage Management, Concurrency, Recovery และ Ad Hoc Query Facility

Complex Objects and Object Identity

OODBMS ต้องมีความสามารถในการยอมให้ผู้ใช้สามารถสร้างออบเจ็กต์ที่มีลักษณะซับซ้อนได้ ซึ่งเรียกว่า Complex Object ซึ่งจะรวมหรืออ้างอิงออบเจ็กต์อื่นๆ เพื่อสร้างโปรแกรมเกี่ยวกับออบเจ็กต์ เช่น เอกสารที่มีความซับซ้อนซ่อนอยู่ภายในโครงสร้าง นอกจากนี้แล้ว OODBMS ควรที่จะสนับสนุนคอนสตักเตอร์ (Constructor) ที่จะใช้สร้าง Complex Object

Complex Object เป็นออบเจ็กต์ที่สร้างจากออบเจ็กต์ง่ายๆ โดยการใช้คอนสตักเตอร์ช่วยในการสร้าง สำหรับรูปแบบที่ง่ายที่สุดของออบเจ็กต์ (บางครั้งเรียกว่า Atomic type ซึ่งเป็นรูปแบบที่ไม่มีส่วนประกอบย่อยเลย) เช่น Integer, Character หรือ Boolean สำหรับรูปแบบของคอนสตักเตอร์ของ Complex Object นั้นจะรวมทูเปิล (Tuples), เซต (Sets), แบ็ก (Bags) (เป็นเซตที่อาจจะมีการซ้ำกันของสมาชิก), ลิสต์ (Lists) และ อาร์เรย์ (arrays) การกำหนดกลุ่มของคอนสตักเตอร์นั้นอย่างน้อย OODBMS ควรจะประกอบด้วยคอนสตักเตอร์ดังนี้คือ เซต, ลิสต์ และ ทูเปิล

ออบเจ็กต์นั้นจะถูกกล่าวถึงโดยการใช้ Object Identity อย่างเป็นอิสระซึ่งจะตรงข้ามกับทูเปิลใน RDBMS ที่ต้องมีการเข้าถึงโดยการผ่านค่าของคีย์ที่เป็นส่วนหนึ่งของทูเปิลนั้นๆ โดยทั่วไปแล้ว Object Identity นั้นจะถูกสร้างขึ้นเองโดยระบบ ซึ่งจะมีลักษณะอยู่ในรูปของพอยน์เตอร์ เชิงวัตถุ (Object-Oriented Pointers) หรือบางครั้งอาจจะเรียกว่า Surrogates ลักษณะการจัดการดังกล่าวนี้สอดคล้องกับเรื่องของความเป็นหนึ่งเดียว (Uniqueness) และเรื่องของ Referential Integrity ที่จะถูกดูแล และจัดการในการใช้งานด้วย

หลักสำคัญในการใช้ Object Identifier ในการสร้าง Complex Object ซึ่งจะช่วยชี้ หรืออ้างอิงไปยังส่วนประกอบอื่นๆของ Complex Object สำหรับตัวอย่างจากการอธิบายออบเจ็กต์เอกสารข้างต้นนั้น Identifier ที่ถูกใช้เพื่อแทนเอกสาร Identifier ที่เป็นรายละเอียดของการใช้ในการแสดงถึงแนวคิดของการใช้โครงสร้างร่วมกัน ซึ่งจะมีความหมายถึงออบเจ็กต์ที่แตกต่างกัน 2 ออบเจ็กต์มีออบเจ็กต์ที่เป็นส่วนประกอบย่อย (Subcomponents) เหมือนกัน

Extensibility Using Abstract Types or Classes

OODBMS ต้องสนับสนุนรูปแบบของ Type หรือ Class ซึ่ง Type ก็คือการอธิบายถึงลักษณะทั่วไปของกลุ่มของออบเจ็กต์ซึ่งมีคุณลักษณะเหมือนกัน ในระบบที่มีพื้นฐานแบบ Type ทั่วๆไปนั้น Type จะถูกใช้ในส่วนต้นๆ ของการคอมไพล์ และไม่ได้เป็นคลาสแรกของออบเจ็กต์

หรือกล่าวได้ว่า Type ไม่ได้เป็นออบเจ็กต์นั่นเอง ทั้งนี้รูปแบบดังกล่าวจะไม่สามารถแก้ไขเปลี่ยนแปลงในช่วง Run Time ได้ สำหรับ Class นั้นมีลักษณะที่คล้ายคลึงกับ Type แต่สามารถทำงานในขณะ Run Time ได้มากกว่าในระบบที่มีพื้นฐานแบบ Class ทั่วๆไปนั้น Class เป็นออบเจ็กต์ในตัวมันเองซึ่งจะคอยอำนวยความสะดวกในการสร้างออบเจ็กต์ใหม่ เช่นการส่งเมสเซจ "new" ไปยังคลาสของออบเจ็กต์ซึ่งการสร้างออบเจ็กต์ขึ้นมานี้จะเรียกว่า Extension ซึ่งเป็นกลุ่มของอินสแตนท์ของคลาสนั้นๆในฐานข้อมูล

การสนับสนุนแนวความคิดใดความคิดหนึ่งข้างต้นนั้นมีความจำเป็นมาก และเป็นมาตรฐานของ OODBMS ในการกำหนด หรือระบุออบเจ็กต์ขึ้นมาใช้งานในระบบฐานข้อมูล และในเรื่องของความสามารถในเรื่อง Extensible นั้นจะเกี่ยวกับการที่ผู้ใช้กำหนดรูปแบบขึ้นมาใหม่ ซึ่งความต้องการดังกล่าวนี้มีการสนับสนุนโดยการใช้ Abstract Data Type หรือ ADT ในการช่วยสร้างออบเจ็กต์ใหม่ขึ้นมา

การใช้ ADT นั้นจะพาดพิงถึงแนวคิดในเรื่อง Encapsulation ส่วนที่เป็น Object State หรือแอตทริบิวต์และส่วนที่เป็นพฤติกรรม (Operation) ไว้ตามหลักการในระบบเชิงวัตถุ ในการติดต่อกับออบเจ็กต์นั้นจะติดต่อผ่านส่วนที่เป็น External Interface เท่านั้น สำหรับส่วนที่เป็น Internal Interface นั้นไม่สามรถที่จะทำการติดต่อได้โดยตรง ซึ่งลักษณะดังกล่าวนี้จะช่วยในเรื่องของความยืดหยุ่นโดยรายละเอียดภายในของออบเจ็กต์นั้นสามารถเปลี่ยนแปลงโดยไม่มีผลกระทบต่อออบเจ็กต์อื่นๆที่ติดต่อเข้ามา

Class หรือ Type Hierarchies

ODBMS ต้องสนับสนุนหลักการ Inheritance ซึ่งเป็นการที่คลาสหนึ่งมีการสืบทอดคุณสมบัติทุกอย่างมาจากอีกคลาสหนึ่งซึ่งคลาสที่สืบทอดคุณสมบัติมานั้นสามารถมีคุณสมบัติอื่นเพิ่มเติมนอกเหนือจากคลาสที่ถูกสืบทอดคุณสมบัติได้ ซึ่งหลักการ Inheritance นั้นเป็นแนวความคิดที่ดีและแสดงถึงรายละเอียดที่น่าสนใจในการกำหนดข้อจำกัดในการสืบทอดนั้นๆ ตัวอย่างเช่น การกำหนดคลาส Teenager ที่เป็นสับคลาสของคลาส Person ที่มีอายุระหว่าง 13 ถึง 19 ปี

ในกรณีของ Multiple Inheritance นั้นเป็นกรณีทั่วๆไปของการสืบทอดที่ยินยอมให้ออบเจ็กต์หนึ่งมีการสืบทอดคุณลักษณะ และการกระทำจากคลาสที่มากกว่า 1 คลาส ตัวอย่างเช่น คลาส Employee และ คลาส Student ถูกกำหนดให้เป็น สับคลาสของคลาส Person ในการสร้างคลาสใหม่ เช่น การกำหนดคลาส Student-Employee ซึ่งออบเจ็กต์ในคลาสดังกล่าวจะแทนคนที่ทำงานไปด้วย เรียนไปด้วย โดยการสืบทอดคุณลักษณะ และการกระทำจากคลาส Student และคลาส Employee

จากหลักการ Inheritance จะพบว่ามีข้อดีที่ชัดเจนมากในเรื่องของการนำกลับมาใช้ใหม่ (Reusability) เพราะว่าทุกๆ โปรแกรมในแต่ละระดับของ Hierarchy สามารถใช้หลักเกณฑ์ดังกล่าวในการสืบทอดคุณลักษณะ และการกระทำจากคลาสที่มีอยู่แล้วได้

Overriding, Overloading and Late Binding

OODBMS ต้องสนับสนุนโอเปอร์เรชัน Overloading และโอเปอร์เรชัน Overriding ซึ่งบางครั้งจะมีการอ้างอิงกับหลักการของ Polymorphism ซึ่งยอมให้มีการใช้ชื่อของโอเปอร์เรชันเหมือนกัน โดยมีลักษณะการทำงานที่แตกต่างกัน ตัวอย่างเช่น การยอมให้มีการประยุกต์ใช้โอเปอร์เรชัน Display กับกลุ่มของออบเจ็กต์ที่เป็นกราฟิก หรือภาพประเภทต่างๆ ที่มีรูปแบบการจัดเก็บแตกต่างกัน โดยมีชื่อของโอเปอร์เรชันเหมือนกัน (ใช้หลักการ Inheritance ในการสืบทอดคุณลักษณะ และการกระทำ) แต่กระทำกับประเภทของข้อมูลที่แตกต่างกันออกไป โดยใช้กลไก Run-Time Binding หรือ Late Binding ในการจัดการเกี่ยวกับการใช้โอเปอร์เรชันดังกล่าวให้ถูกต้องตามประเภทของข้อมูลที่เรียกใช้โอเปอร์เรชันนั้นๆ

Persistence and Secondary Storage Management

Persistence คือการยอมให้มีการคงอยู่ของออบเจ็กต์ แม้ว่าการประมวลผลนั้นจะเสร็จสิ้นลงแล้ว การสนับสนุนการคงอยู่ของออบเจ็กต์เป็นความต้องการที่ชัดเจนของ OODBMS เพื่อความถูกต้องของระบบฐานข้อมูลในการอ้างอิงจากออบเจ็กต์อื่นในภายหลัง และ OODBMS ยังต้องการระบบที่จัดการกับฐานข้อมูลขนาดใหญ่ซึ่ง OODBMS จะต้องมีการสนับสนุนกลไกการจัดการเกี่ยวกับ Secondary Storage เช่น การจัดการ Index, การทำ Clustering, การทำ Buffering, การเข้าถึงจากเส้นทาง (Path) ที่เลือก และการทำ Query Optimization ซึ่งการจัดการดังกล่าวนี้ ผู้ใช้ไม่สามารถทราบได้เลยว่ามีการจัดการอย่างไร

ความสามารถดังกล่าวเป็นส่วนสำคัญในการสนับสนุนให้ระบบฐานข้อมูลสามารถเก็บข้อมูลประเภทมัลติมีเดียได้ เช่น โปรแกรมประยุกต์ที่จัดการเกี่ยวกับเอกสารซึ่งอาจจะมีรูปแบบการจัดเก็บเป็นภาพที่เรียกว่า "Clip Art" ในระดับสูงนั้นการจัดการเกี่ยวกับ Secondary Storage จะเป็นส่วนสำคัญในการเพิ่มประสิทธิภาพการทำงานบนข้อมูลประเภทมัลติมีเดีย เช่น เสียง, ภาพ และตัวอักษร (ที่มีจำนวนข้อมูลมาก) ให้มีการทำงานที่ดีขึ้น

Concurrency and Recovery

OODBMS ต้องสนับสนุนการทำงานเกี่ยวกับการประมวลผลทรานส์แซ็กชั่น (Transaction Processing) และกลไกในการควบคุมความถูกต้อง (Concurrency Control) ซึ่งมีความจำเป็นในเรื่องประสิทธิภาพ และความถูกต้องของการประมวลผลจากการร้องขอที่เข้ามาในขณะนั้นๆ กลไกที่สำคัญอีกส่วนหนึ่ง คือกลไกการเรียกคืน (Recovery) ซึ่งกลไกการเรียกคืนจะช่วยในการสนับสนุนการทำงานของกลไกการควบคุมความถูกต้อง

ในการประมวลผลทรานส์แอ็กชัน(Transaction), การควบคุมความถูกต้อง และกลไกการเรียกคืน ต้องสนับสนุนการทำงานในการติดต่อกับรูปแบบข้อมูลที่เป็นมัลติมีเดีย, ข้อมูลชั้นสูง หรือข้อมูลที่มีความซับซ้อนมากโดยจะใช้ข้อดีของการกำหนดเองโดยผู้ใช้ (User-Define) ซึ่งสามารถกำหนดรูปแบบของข้อมูลได้เองในการพัฒนารูปแบบของความถูกต้อง ตัวอย่าง เช่น ในการที่ผู้ใช้กำหนดชนิดของข้อมูลแบบ Queue ที่มีการทำงานคือ Enqueue() และ Dequeue() ถ้ากำหนดให้ Q เป็นออบเจ็กต์ของชนิดข้อมูล Queue ดังนั้น Equeue(Q, X) จะแทนสมาชิก X ที่เป็นข้อมูลเข้าในส่วนท้ายของคิว และ Dequeue(Q, Y) จะแทนการลบสมาชิก Y ที่เป็นข้อมูลออกในส่วนท้ายของคิว Enqueue() และ Dequeue() สามารถพิจารณาถึงกลไกการควบคุมความถูกต้องโดยการเขียนฟังก์ชันการทำงาน เช่นมีการ Lock ค่าของ Q ในการทำงานแต่ละครั้ง ซึ่งการทำงานของทรานส์แอ็กชัน ทั้ง Enqueue() และ Dequeue() ที่กระทำบน Q จะต้องรอให้ทรานส์แอ็กชันอื่นๆ เสร็จสมบูรณ์ก่อน

Ad hoc Query Facility

ในส่วนของ OODBMS ต้องสนับสนุนการใช้งานในเชิงวัตถุที่สอดคล้องกับภาษาที่ใช้ในการสอบถามข้อมูล เช่นเดียวกับภาษาที่ใช้งานในระบบฐานข้อมูลเชิงสัมพันธ์ อย่างไรก็ตามสิ่งที่อำนวยความสะดวกในการสอบถามควรจะมี คือ

- ภาษาระดับสูง (High-Level) คือ สามารถเข้าใจได้ง่าย
- ประสิทธิภาพ (Efficient) คือ ในส่วนของ Interface มีการสนับสนุนมาจากการทำ Query Optimize
- ความเป็นอิสระของโปรแกรมประยุกต์ (Application-Independent) คือ ความสามารในการประยุกต์ใช้กับ ฐานข้อมูลอื่นๆที่เป็นไปได้

ในส่วนนี้มีมาตรฐานที่รองรับคือ SQL3 และ ODMG-93 ซึ่งใน SQL3 นั้นจะเป็นการสนับสนุนการทำงานบน ORDBMS มีลักษณะการทำงานที่เรียกว่า "SQL-Like" และสำหรับมาตรฐานของ ODMG-93 นั้นจะมีการกำหนดมาตรฐานของ OQL ซึ่งเป็นภาษาในการสอบถามข้อมูลที่สนับสนุนการทำงานบน OODBMS

4.4 ข้อดีของระบบการจัดการฐานข้อมูลเชิงวัตถุ

ในส่วนนี้จะเป็นการกล่าวถึงข้อดีของระบบการจัดการฐานข้อมูลเชิงวัตถุที่นำเอาแนวคิดเชิงวัตถุมาประยุกต์ใช้ในระบบฐานข้อมูล ซึ่งข้อดีต่างๆ ที่จะกล่าวถึงในส่วนนี้เป็นเพียงบางส่วนขององค์ประกอบที่สนับสนุนในระบบการจัดการฐานข้อมูลเชิงวัตถุ

Extensibility

สำหรับ User-Defined Type และ Overloading Operation นั้นส่วนของคลาส และการจัดการความสัมพันธ์จะสามารถปรับเปลี่ยน หรือเพิ่มเติมได้ตามความต้องการของผู้ใช้ การเปลี่ยนแปลง หรือการเพิ่มนี้จะถูกจัดการให้กับคลาสที่มีความสัมพันธ์กันโดยอัตโนมัติ นั่นคือ คลาส และ Operation ใหม่ๆ สามารถรวมเข้าไปในระบบ โดยปราศจากผลกระทบกับวัตถุอื่นๆภายในระบบ

Information Hiding

การซ่อนข้อมูลวัตถุจะอนุญาตให้ข้อมูล และ Operation ถูกกำหนดอยู่ในโครงสร้างของมูลเดี่ยว (Single Model) นั่นคือ ข้อมูลจะถูกกำหนดเฉพาะที่ (Local) และถูกซ่อนอยู่ในวัตถุเอง เมื่อมีการเชื่อมต่อเกิดขึ้นจะมีการแสดงออกมาโดยอัตโนมัติ

Flexibility of Type Definition

ความสามารถที่อนุญาตให้ผู้ใช้กำหนดคลาส และ Operation ใหม่จะให้ความยืดหยุ่นในการจัดการกับผู้ใช้ ซึ่งเป็นประโยชน์กับการประยุกต์ใช้ฐานข้อมูลใหม่ๆ ในการปรับเปลี่ยนข้อมูล

Code Reusability

จากคุณสมบัติการสืบทอดจากคลาสแม่ไปยังคลาสลูก ทำให้ไม่ต้องมีการกำหนดวิธีการและ Operation ขึ้นใหม่ในแต่ละคลาส ซึ่งจะช่วยลดความซ้ำซ้อน และการทำงานที่ไม่ควบคู่กัน

บทที่ 5

แนะนำระบบจัดการฐานข้อมูลเชิงวัตถุ CACHE

CACHE เป็น ระบบจัดการฐานข้อมูลเชิงวัตถุ ที่พัฒนาโดย บริษัท InterSystem Corporation ในปี 1999 เดิมที CACHE เป็นระบบจัดการฐานข้อมูลเชิงสัมพันธ์ (RDBMS) แต่ในภายหลังได้เพิ่มระบบการทำงานให้สนับสนุนระบบฐานข้อมูลเชิงวัตถุ และได้มีการพัฒนาเรื่อยมาจนกระทั่ง สามารถรองรับการทำงานเชิงวัตถุได้อย่างมีประสิทธิภาพ และง่ายแก่การพัฒนา

5.1 แนะนำระบบฐานข้อมูลเชิงวัตถุ CACHE ในขั้นต้น

CACHE มีความสามารถในการรองรับการทำงานได้หลากหลาย Platform ดูได้จากตาราง

Operating System	Hardware
Compaq Tru64 UNIX 4.0F and 5.0 (Digital UNIX)	Alpha computers
Digital OpenVMS 7.1 and 7.2	Alpha computers
DG/UX 4.2	AViioN(Intel) computers
HP/UX 11.0	Hewlett-Packard computers
IBM AIX 4.2.2,4.3.3	IBM PowerPC computer
IBM AIX 5.3.2	IBM RS/6000 computer
Red Hat Linux 6.1	Intel-based computers
SCO UNIX 5.0, 5.0.2, 5.0.4, and 5.0.5	Intel-based computers
SCO Unix Ware 7.1	Intel-based computers
Sun Solaris 2.7	Intel-based and SPARC-based computers
Windows NT 4.0 (with Service Pack 4, 5, 6 or, 6A)	Intel-based and Alpha-based computers
Windows 2000	Intel-based computers
Windows 95 and 98	Intel-based computers

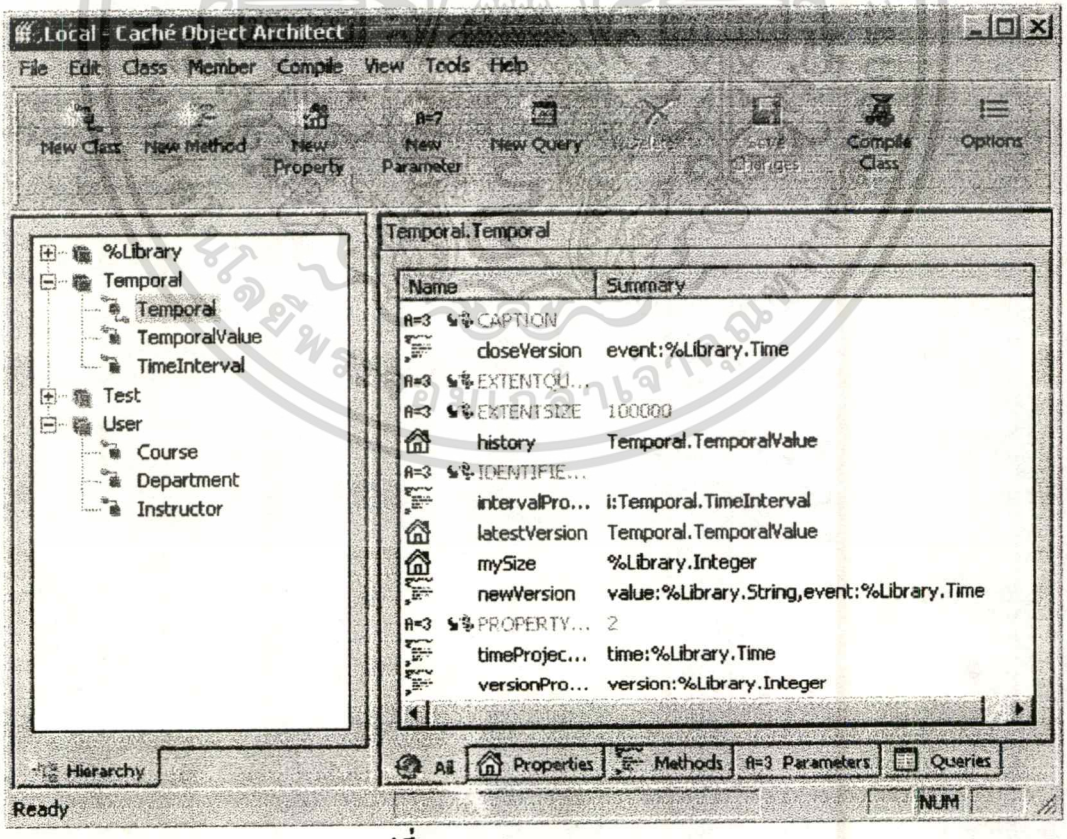
ตาราง 5.1 แสดง Platform ที่ CACHE สนับสนุน

ในงานวิจัยครั้งนี้ทำบน Platform Intel-based และระบบปฏิบัติการ Windows 2000 ในการพัฒนาระบบงาน

5.2 Component ต่างๆ ใน CACHE

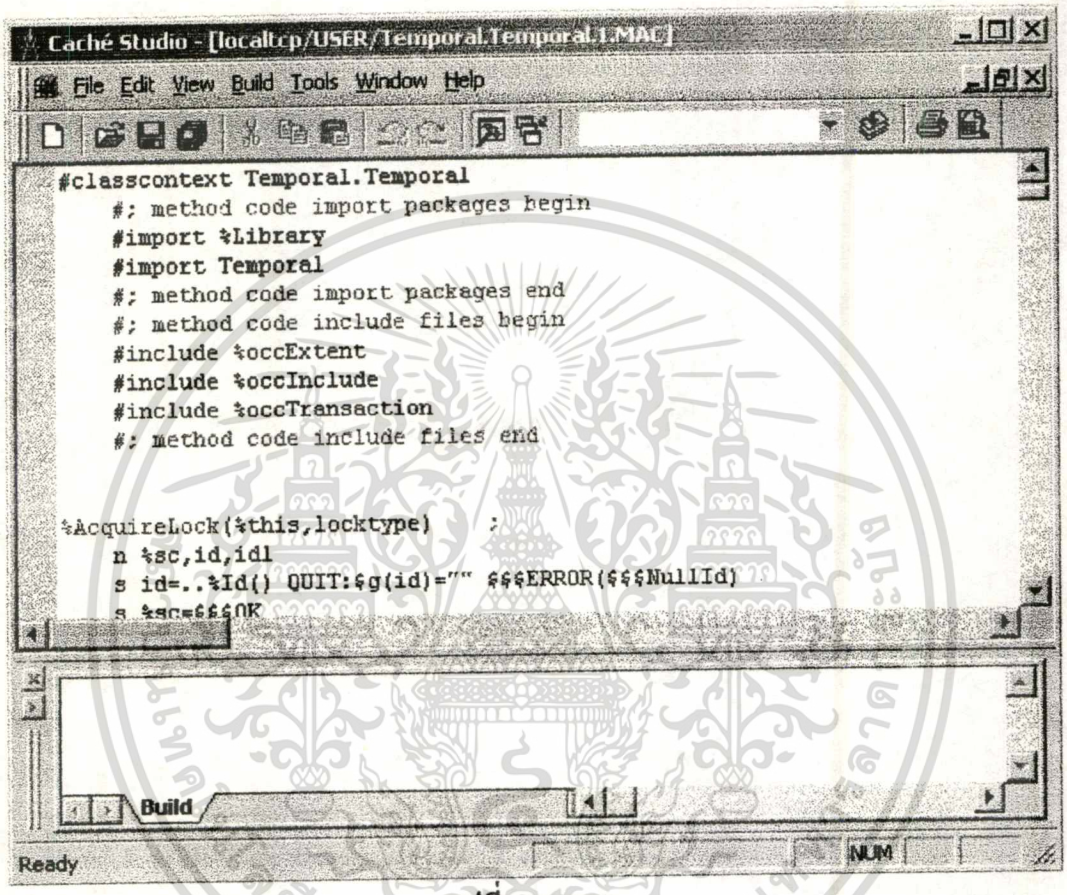
หลังจากที่เราติดตั้งระบบฐานข้อมูล CACHE แล้ว จะเห็นว่ามี Tools ต่างๆ ที่ติดตั้งมาด้วยเพื่อช่วยในการพัฒนา ดังนี้

- **Object Architecture** : เป็นเครื่องมือที่ใช้ในการสร้าง Classes ต่างๆ ที่จะนำไปใช้งานในระบบ รวมทั้ง Property, Method, Parameter, Relationship และ Query ของ Class นั้นๆ จากนั้นจะทำการ Compile Class ที่สร้างเพื่อที่ Class นั้นจะเข้าไปสู่ Database ของ CACHE และพร้อมที่จะใช้งาน และจาก Classes ที่สร้างเราสามารถ Export ออกไปเป็นภาษาต่างๆ เช่น Java, ActiveX เพื่อที่จะใช้เป็น Interface ในการพัฒนาระบบงาน



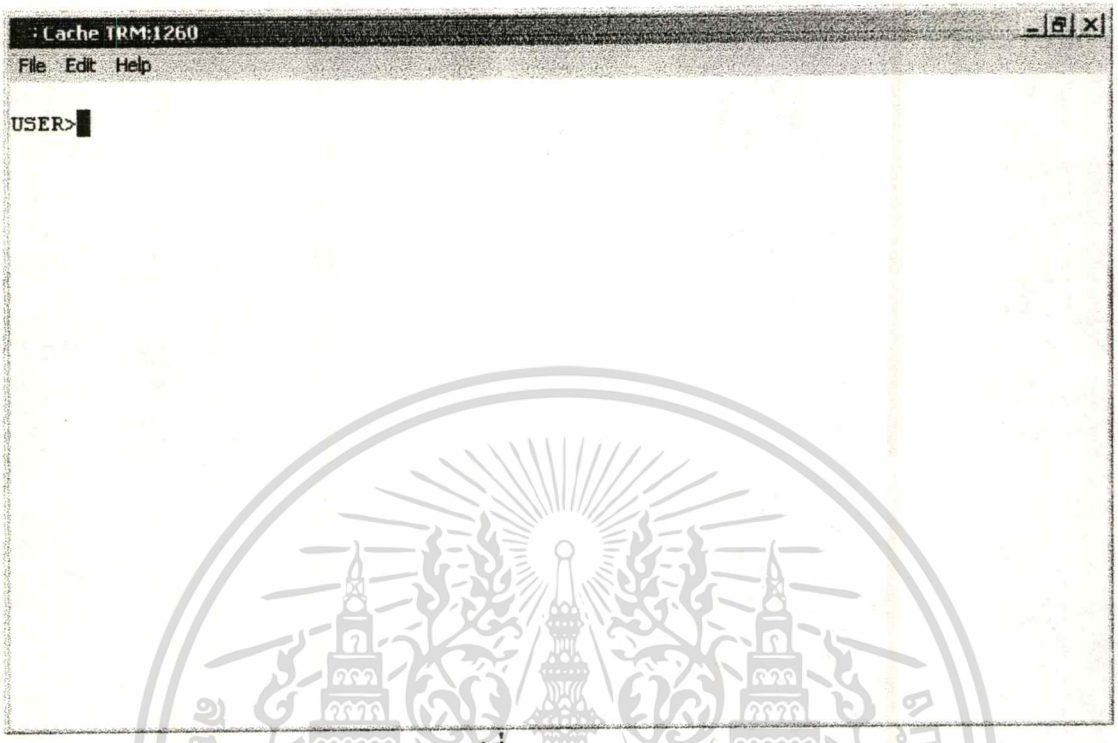
รูปที่ 5.1 Object Architecture

- **Studio** : ใช้สำหรับเขียน ObjectScript สำหรับ CACHE เป็น Tool ที่ใช้ Script ของ CACHE ในการทำงาน ซึ่งในงานวิจัยนี้เราไม่ได้ใช้งาน



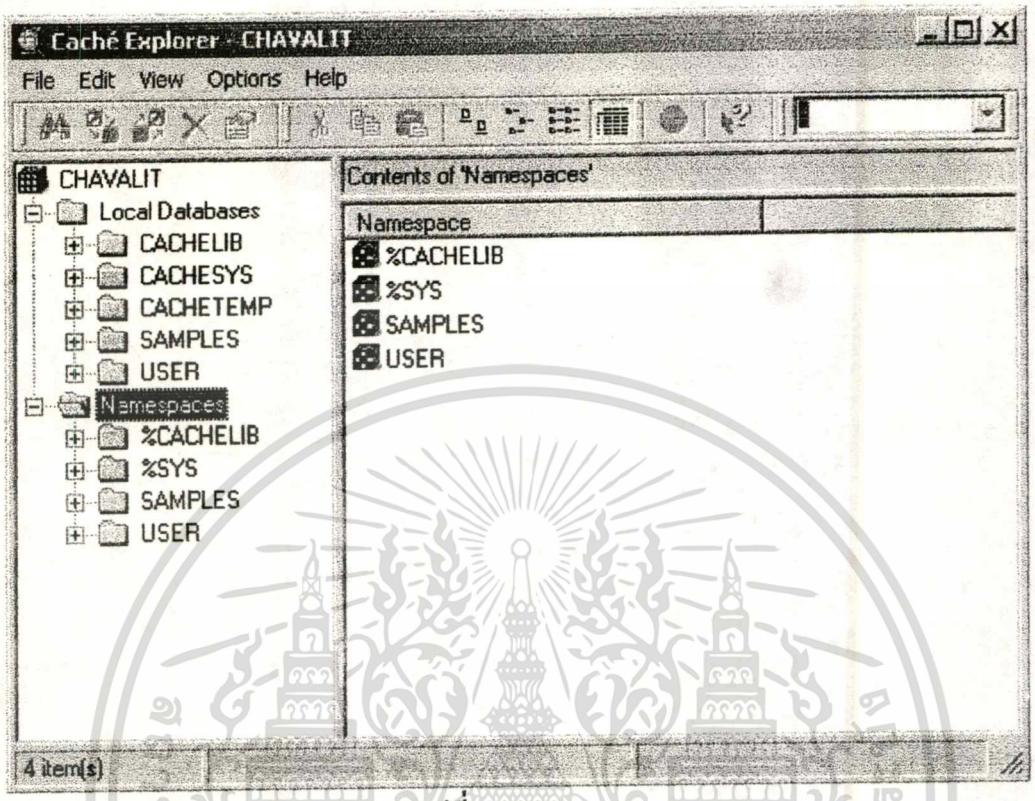
รูปที่ 5.2 Studio

- **Terminal** : เป็นเหมือน Command Line ใช้ในการเรียก Script ต่างๆ ให้ทำงาน



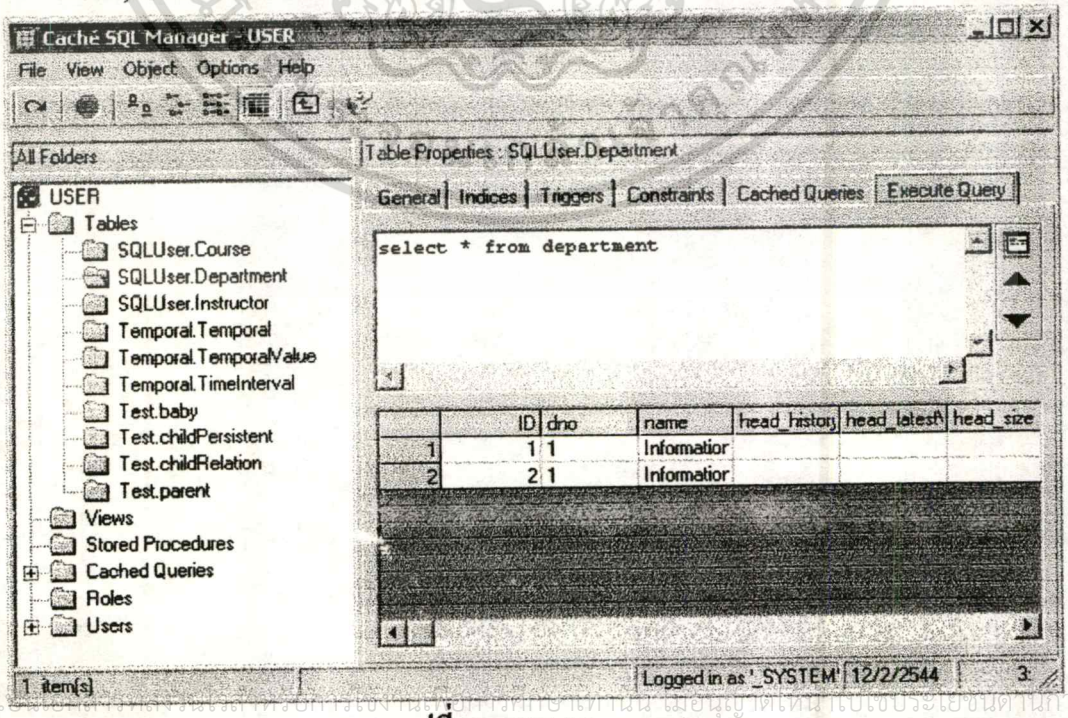
รูปที่ 5.3 Terminal

- **Explorer** : มีลักษณะต่าง ๆ ดังนี้
  1. ในระดับ Routines : สามารถที่จะทำการค้นหาและทำการแทนที่ (Find/Replace) และทำการ Compile functionality ได้ เสมือนกับความสามารถในการที่จะกำหนด Mode ของ Language และ การแสดงจำนวน n บรรทัดแรก
  2. ในระดับ Global : สามารถที่จะทำการค้นหาและพิมพ์ได้เสมือนกับการ cut-and-paste operations ใน The Global Viewer
  3. ในระดับ Classes : สามารถทำการพิมพ์ได้



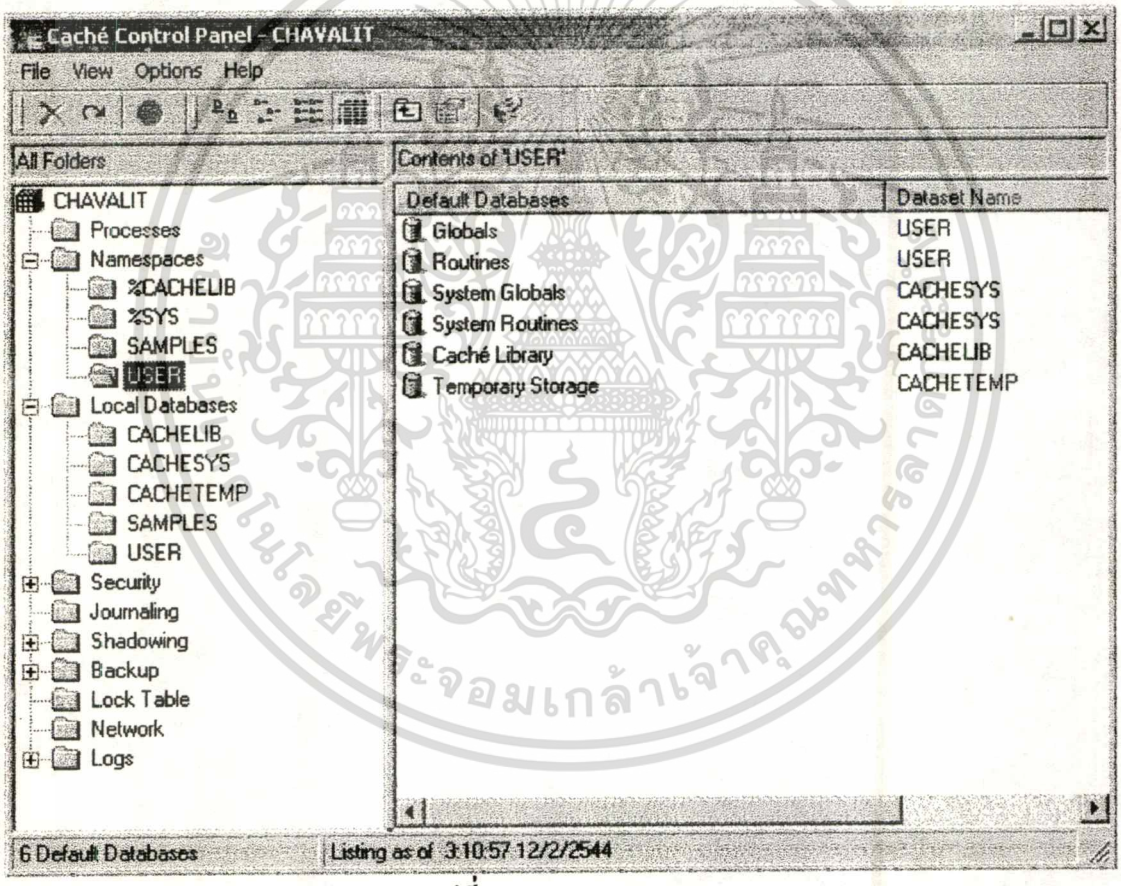
รูปที่ 5.4 Explorer

- **SQL Manager** : ใช้ในการ Query ดูข้อมูลต่างๆ ใน Classes ที่อยู่ใน CACHE รวมทั้ง Views, Stored Procedures และการกำหนด Security ให้กับ User ในการ Access ข้อมูล



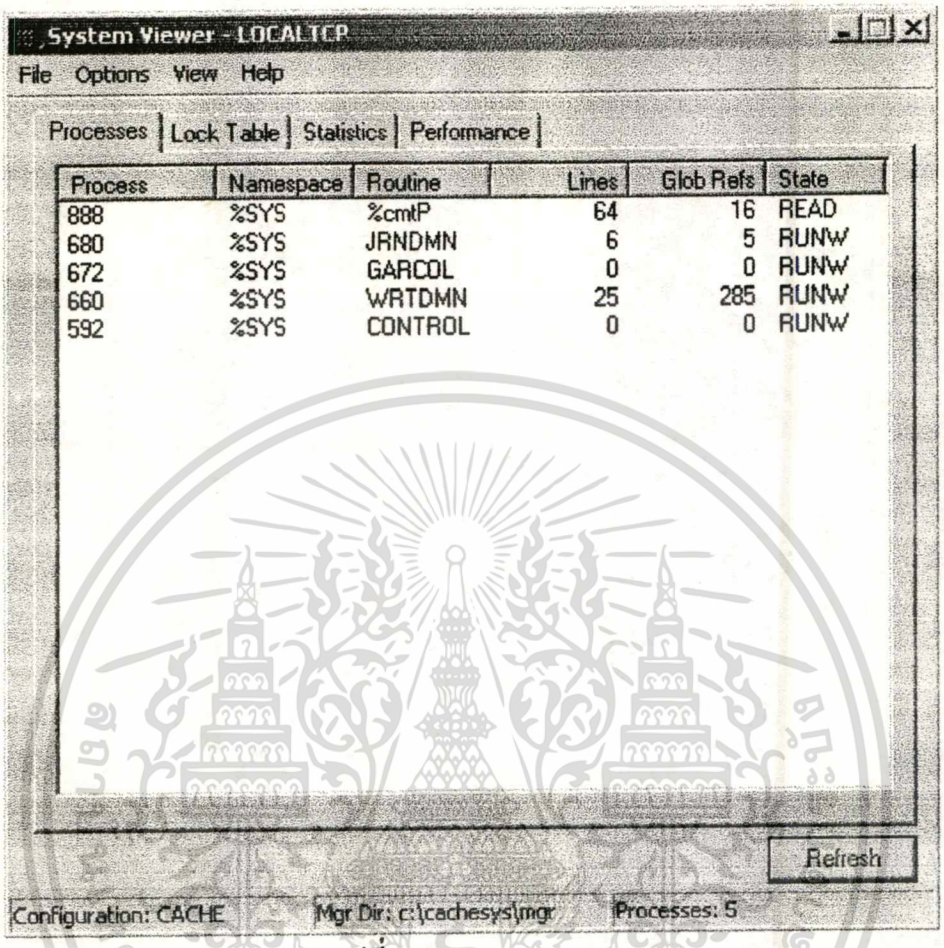
รูปที่ 5.5 SQL Manager

- **Control Panel** : ประกอบด้วย Section ต่างๆ ดังนี้
  1. The Namespaces section : สามารถให้เราทำการลบ Namespaces
  2. The Local Databases section : ประกอบด้วย Delete option รวมทั้ง แสดงเวลาล่าสุด และทำการตรวจสอบความถูกต้องของข้อมูล (Integrity Check)
  3. Print section : ส่วนของการพิมพ์
  4. ส่วนที่แสดงข้อมูลต่างๆ ของการประมวลผล



รูปที่ 5.6 Control Panel

- **System Viewer** : ใช้ดู Performance และ Processes ต่างๆ รวมทั้งการ Lock Table และ Statistics ของ CACHE

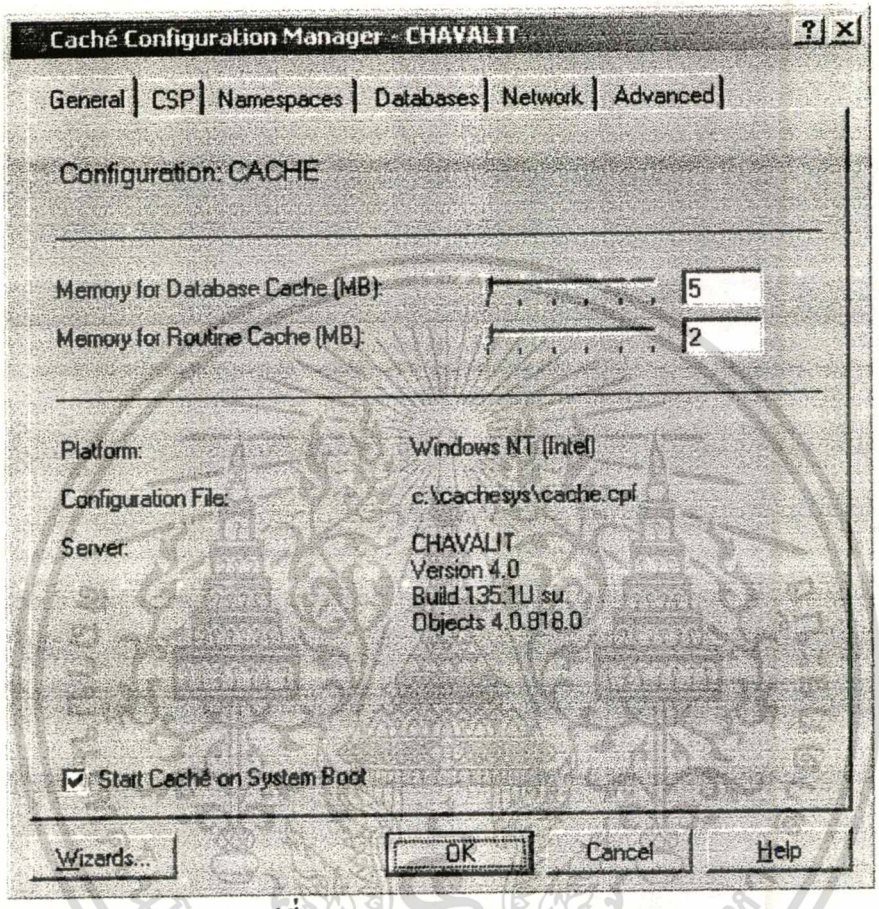


รูปที่ 5.7 System Viewer

- **Configuration Manager** : ทำหน้าที่ในการควบคุมจัดการการทำงานของระบบฐานข้อมูล ซึ่งสามารถจัดแบ่งการจัดการเป็น Section ซึ่งเราสามารถกำหนดค่าต่างๆ ได้ ดังนี้

1. *SQL section* : SQL system-wide settings
2. *License section* : WAN (Wide-area Network) License Allocation
3. *General section* : กำหนดจำนวนงาน (Job) ในตอนเริ่มต้น และกำหนดเวลาที่จะ Time out
4. *Process section* : สามารถเพิ่มหน่วยความจำได้ถึง 16 MB ต่อ Process
5. *ViewPoint section* : ดูข้อมูลการประมวลผลของ Server
6. *Network section* : DTM collation, เราสามารถกำหนด DTM-NetBIOS Server เพื่อที่จะใช้ DTM Compatible Collation type for DTM global
7. *Journal section* : สามารถกำหนดจำนวนวันก่อนที่จะทำการเคลียร์ข้อผิดพลาดต่างๆ (Error) ออกจากระบบ

8. *Miscellaneous section* : ทำหน้าที่ในการสร้าง Log File เพื่อใช้ในการ Rollback



รูปที่ 5.8 Configuration Manager

5.3 การพัฒนาระบบงานโดยใช้ CACHE

ในส่วนนี้จะนำเสนอการพัฒนาระบบโดยใช้ CACHE เป็นเครื่องมือในการพัฒนา เช่นเดียวกันกับการพัฒนา Application ส่วนใหญ่ซึ่งจะประกอบด้วยส่วนต่างๆ ดังนี้

- **Data** – ส่วนของข้อมูล
- **Logic** – ส่วนของ Business Rule
- **User Interface** – ส่วนที่ติดต่อกับผู้ใช้
- **Output** – ส่วนของข้อมูลที่ได้มาหลังจากผ่านการประมวลผลแล้ว

CACHE ใช้หลักการพัฒนาเชิงวัตถุ โดยจะนำเอา Data และ Logic มารวมกันเป็น Object โดยจะมีการกำหนด Property เป็นส่วนของ Data และกำหนด Method เป็นส่วนของ Logic เก็บไว้ใน Class โดยเมื่อทำการเปรียบเทียบกับระบบฐานข้อมูลทั่วๆ ไปจะสามารถสร้างเป็นตารางเปรียบเทียบได้ดังนี้

Cache	Traditional Relational	
Rich Properties	Simple Columns	- ในเทคโนโลยีแบบ Relational การนิยามข้อมูลจะนิยามเป็น Column ซึ่งต้องเป็นข้อมูลอย่างธรรมดา และเป็น Single-valued data types - Cache สามารถสร้างลักษณะของข้อมูลให้สามารถรองรับการทำงานให้ดียิ่งขึ้น โดยสามาถที่จะนิยามลักษณะแบบใหม่ที่จะเพิ่ม Object หนึ่งๆ เข้าไว้ในอีก Object หนึ่งได้ หรือที่เรียกว่า Complex Object
Collections	Extra Tables	- Cache แต่ละรายการสามารถที่จะประกอบด้วยหลายๆ Item ได้ โดยการใช้ Collection - Relational Database เราต้องทำการแบ่ง Table แล้วทำการ Join Table เพื่อที่จะรองรับการทำงานแบบ Complex Object ได้
Methods	Loose Logic	- ในระบบฐานข้อมูลเชิงสัมพันธ์ไม่มีส่วนของการกำหนด Logic

ตาราง 5.2 แสดงการเปรียบเทียบ Cache กับ Traditional

ในส่วน User Interface เมื่อเราสร้าง Class ได้แล้วเราสามารถที่จะนำ Tools ต่างๆ เข้ามาช่วงสร้าง Application ได้หลากหลายเช่น Visual Basic หรือ Delphi หรือ Tools ของ CACHE เองที่ใช้ Cache Object Script ในการพัฒนา ในกรณีที่ Application ที่เราพัฒนาทำงานในลักษณะ Windows และสามารถใช้ Java Tools ในการพัฒนา Application ที่มีการทำงานในลักษณะ Browser ในส่วนของ Output ในกาที่เราต้องการข้อมูลจากระบบเราสามารถที่จะใช้ SQL ได้ทำนองเดียวกันกับที่ใช้งานกับ Relation โดยที่เราจะเขียน SQL เองหรือสามารถใช้ Tools อื่นๆ ช่วยในการ Generate ก็ได้เช่น Crystal Report, Business Object หรือ MS Access

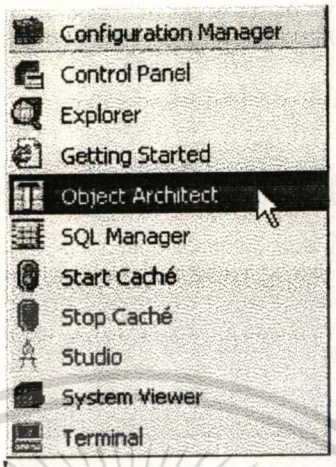
5.4 ขั้นตอนการพัฒนาระบบงานโดย CACHE

ขั้นตอนในการพัฒนาระบบโดยใช้ CACHE จะเริ่มจากเราต้องมี Class Diagram ที่ออกแบบมาก่อนแล้ว ซึ่งเราสามารถที่จะใช้ Rational Rose เป็นเครื่องมือช่วยในการออกแบบได้ หลังจากที่เราได้ Object Model ที่ออกแบบแล้ว เราจะทำการสร้างระบบงานโดยมีขั้นตอนดังต่อไปนี้

ขั้นที่ 1 Starting CACHE : ทำการ Start CACHE โดยการ Click ขวาที่ Menu Bar ของ Windows ดังภาพ เพื่อทำการ Start CACHE Engine

รูปที่ 5.9 การ Start Cache

ขั้นที่ 2 Object Architect : หลังจากที่เราเปิด CACHE แล้ว ขั้นตอนต่อมาเราจะทำการสร้าง Class โดยใช้ Object Architecture โดยทำตามรูปภาพ

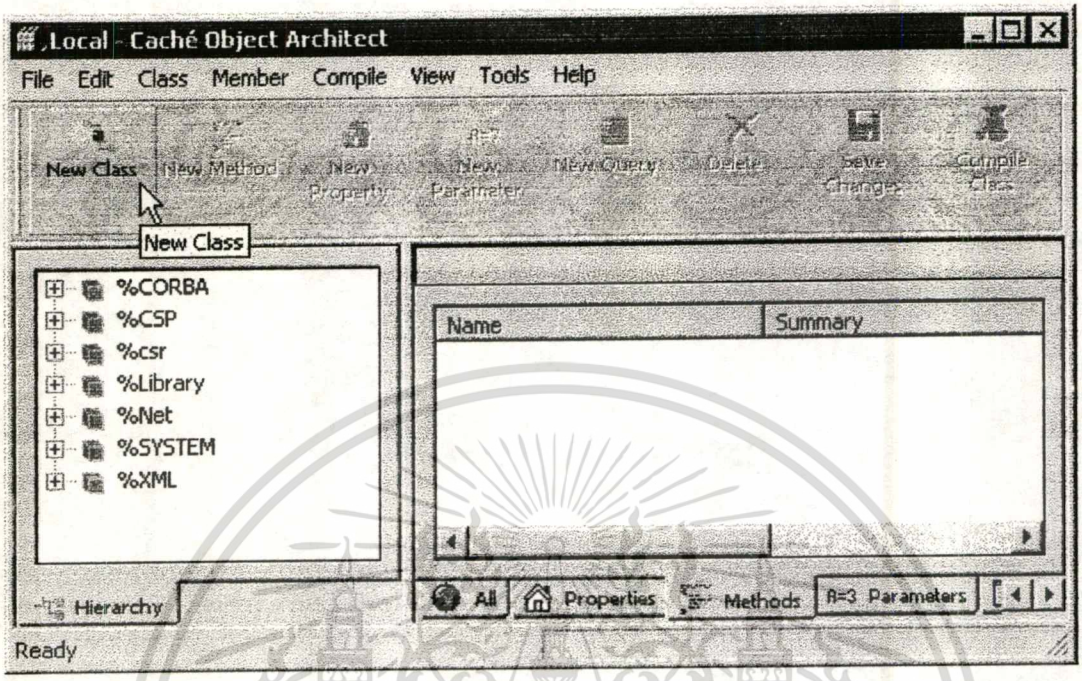


รูปที่ 5.10 การเข้าสู่ Object Architect

ขั้นที่ 3 Connection Architect : หลังจากที่เราเข้าสู่ Object Architect แล้วจะปรากฏหน้าจอดังภาพ เป็นการให้เราเลือกว่าจะทำการ Connect กับ Cache Dictionary Located ที่ใดซึ่งเราสามารถกำหนดเองได้โดยการระบุ IP Address ของเครื่องที่เราต้องการจะ Connect ในที่นี้เราเลือก architecture Connect ชื่อ SAMPLES ซึ่งมี Location อยู่ที่เรื่องของเราเอง (IP Address: 127.0.0.1)

รูปที่ 5.11 Architect connect

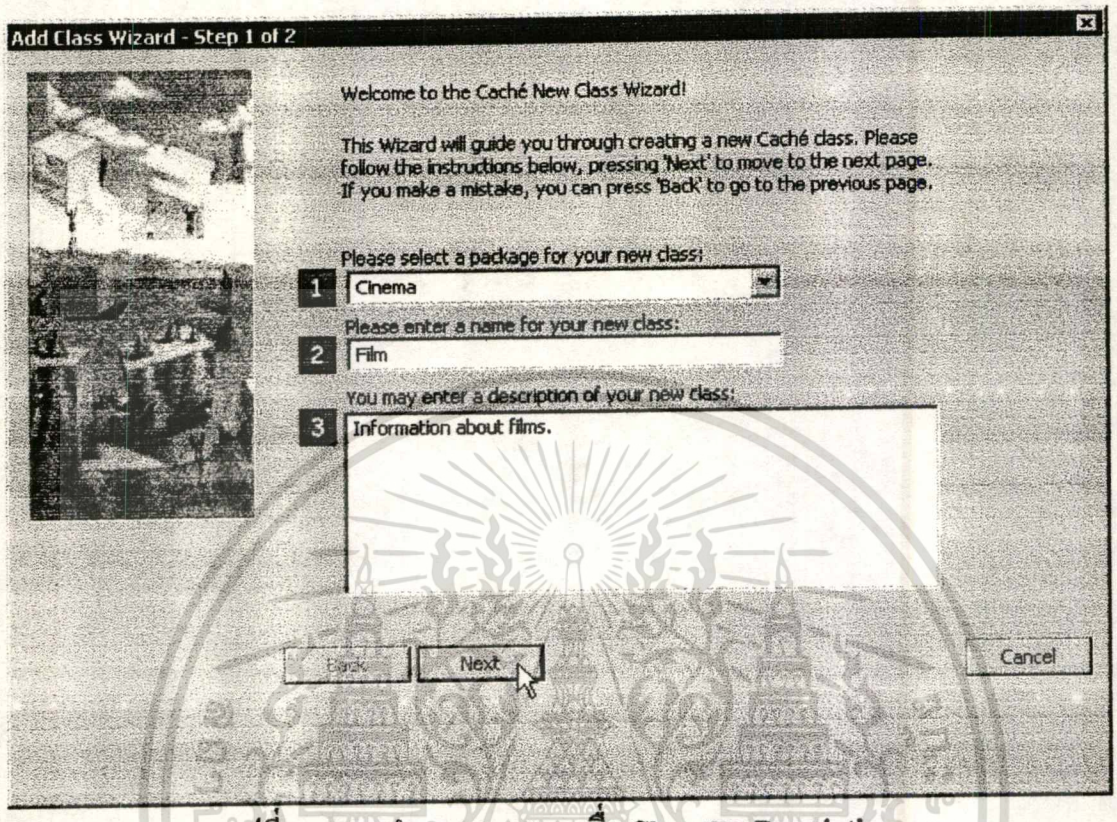
ขั้นที่ 4 การสร้าง Class : หลังจากที่เรา Connect เข้าสู่ Object Architecture แล้วจะปรากฏหน้าจอดังภาพ



รูปที่ 5.12 Object Architect

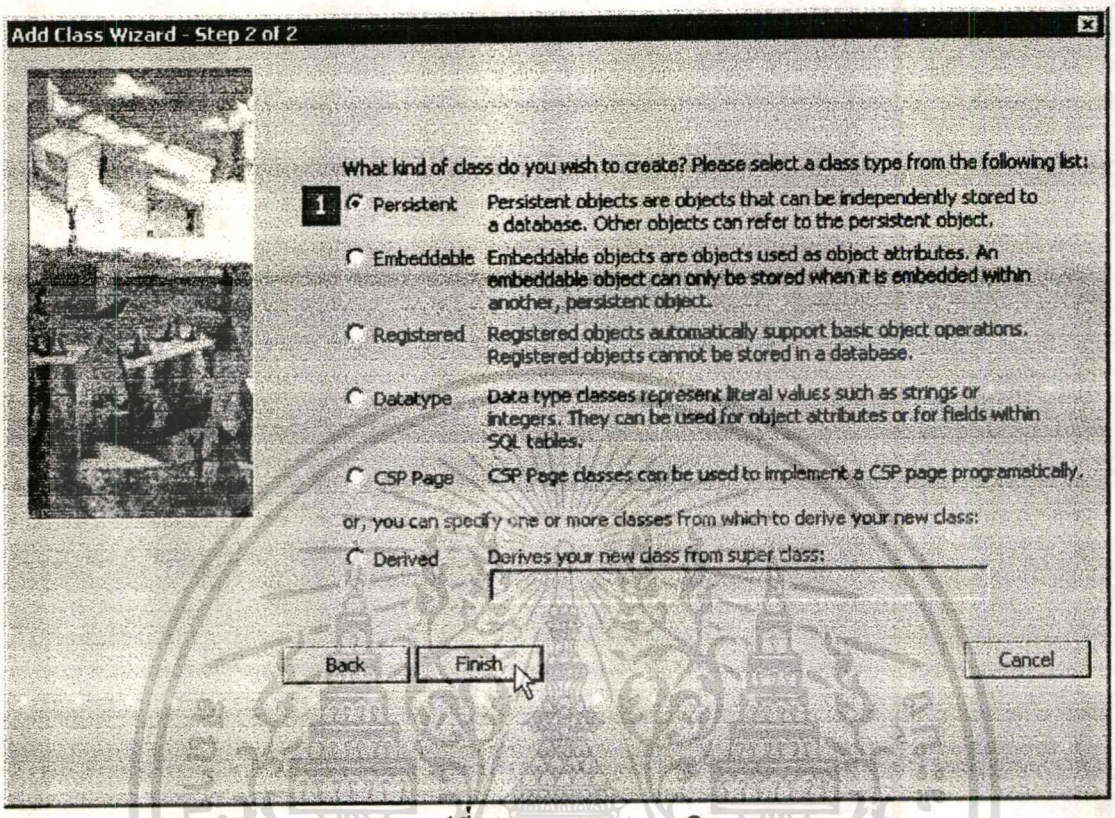
จากภาพเราต้องการที่จะสร้าง Class ใหม่ ทำได้โดยการ Click ที่ปุ่ม New Class จากนั้นก็จะเข้าสู่การสร้าง Class โดย CACHE จะมี Wizard เป็นตัวช่วยอำนวยความสะดวกในการสร้างทำให้เข้าใจได้ง่าย โดยจะมีขั้นตอนย่อยดังนี้

1.1 กำหนดชื่อ Class



รูปที่ 5.13 การกำหนด Package ชื่อ Class และ Description

1.2 กำหนดชนิดของ Class

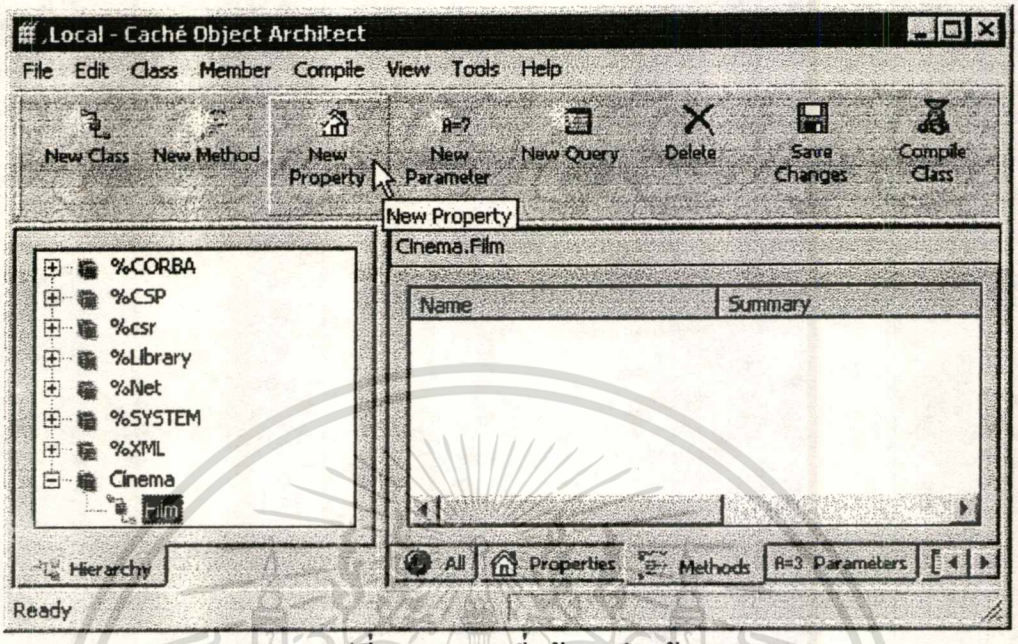


รูปที่ 5.14 การกำหนดชนิดของ Class

การกำหนดชนิดของ Class แบ่งได้เป็น 4 ประเภทดังนี้

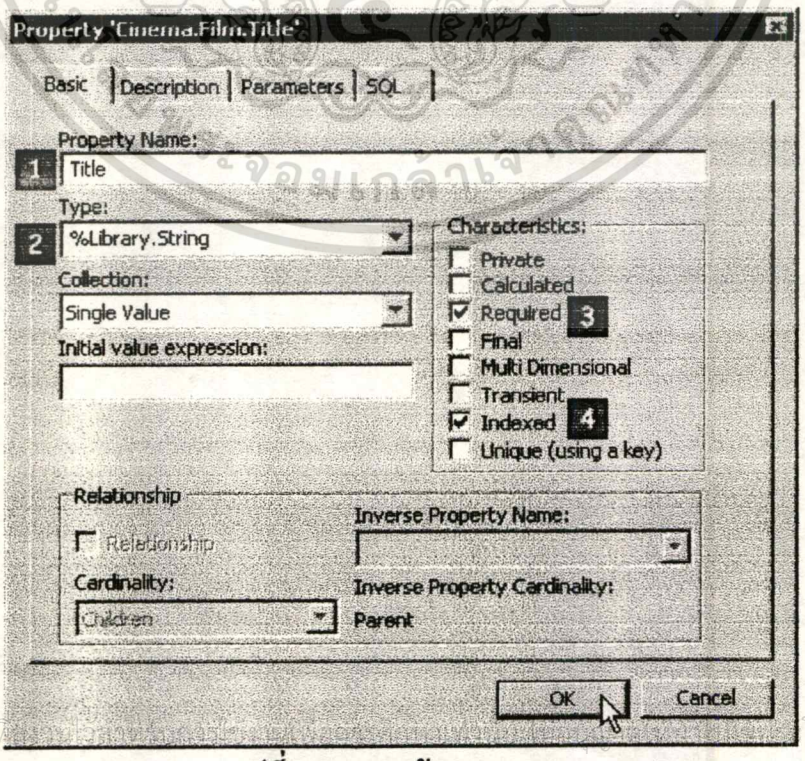
1. Persistent – เป็น Object ที่เป็นอิสระ ไม่ขึ้นกับ Object ใดๆ
2. Embeddable – เป็น Object ที่ไม่สามารถอยู่อย่างโดดเดี่ยวได้ คือ ต้องประกอบอยู่ใน Object เท่านั้น
3. Registered – เป็น Object ที่ไม่สามารถเก็บในฐานข้อมูลได้ เป็น Object ที่ใช้ในการ Support การทำงานขึ้นพื้นฐานของ Object อื่นๆ
4. Datatype – จะอยู่ในรูปแบบของ Literal Value
5. Derived – เป็น Object ที่ Inherit มาจาก Super Class

ในตัวอย่างเป็นการสร้าง Class ชื่อ Instructor โดยเป็น Persistent เมื่อทุกอย่างเสร็จแล้วจะได้ผลดังภาพ จากนั้นจะเป็นการสร้าง Property เป็นขั้นตอนต่อไป



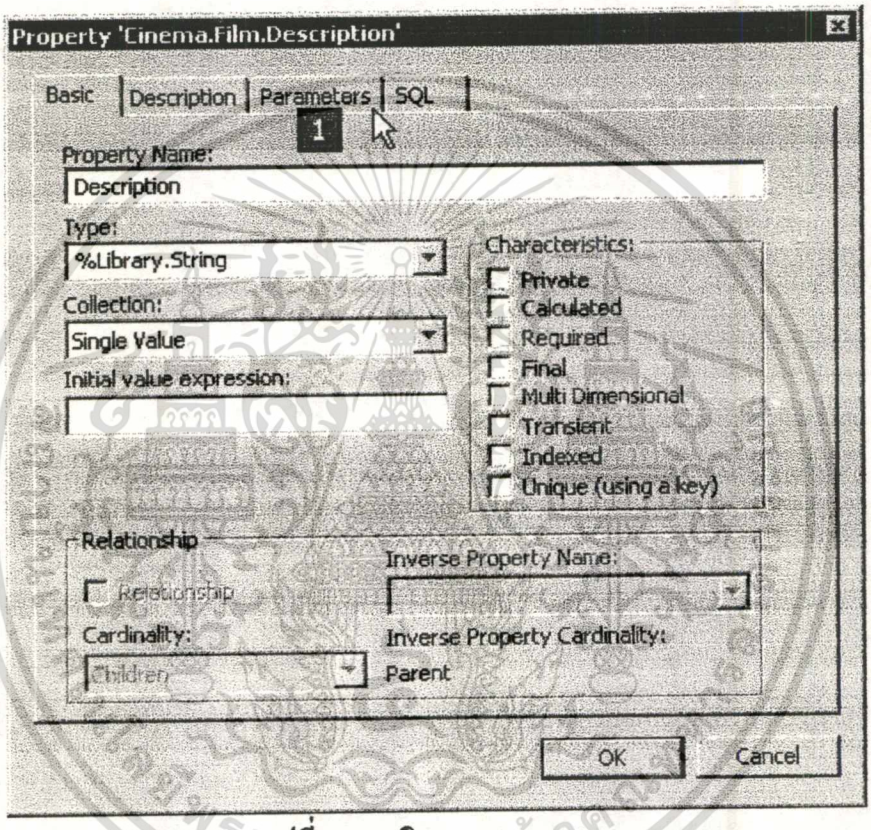
รูปที่ 5.15 Class ที่สร้างเสร็จแล้ว

ขั้นที่ 5 การสร้าง Property : หลักจากที่เราได้สร้าง Class แล้ว ขั้นต่อไปคือการกำหนด Property ให้กับ Class ที่เราสร้าง โดยจากภาพข้างบนเมื่อ Click ที่ New Property แล้วจะเข้าสู่ Wizard การสร้าง Property ดังภาพ



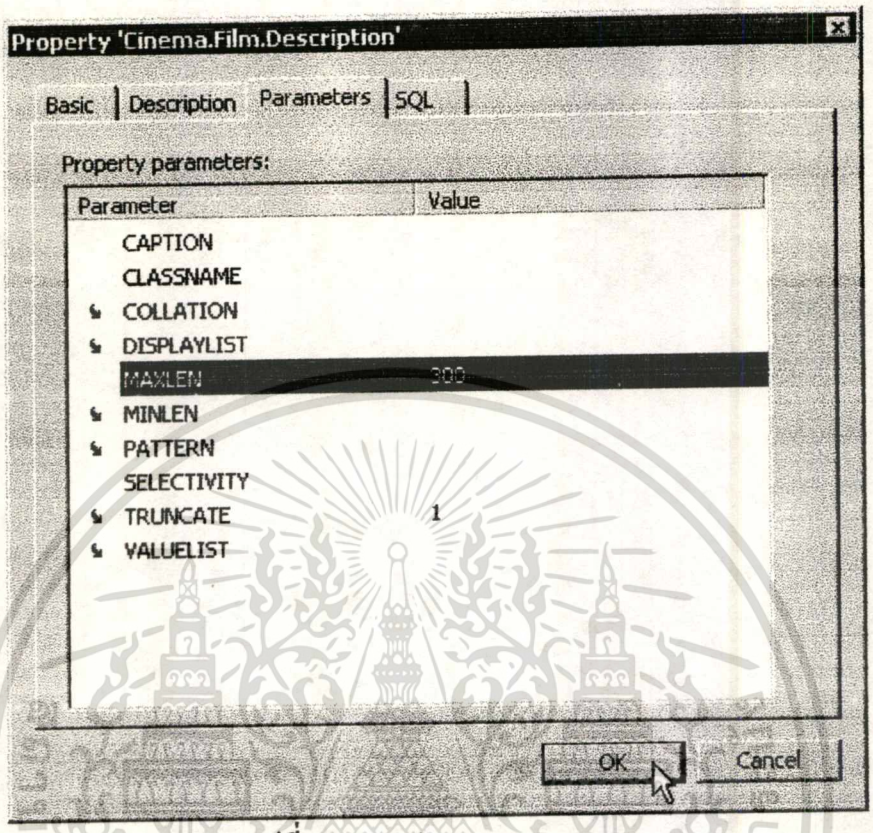
รูปที่ 5.16 การสร้าง Property

จากภาพที่ 5.16 เป็นการสร้าง Property โดยกำหนด Property Name และกำหนดชนิดของ Property นั้น โดย CACHE มีชนิดของ Property ซึ่งสามารถรองรับการทำงานได้แทบทุกประเภท นอกจากนี้ยังมีส่วนของ Collection ซึ่งเราสามารถเลือกได้ว่า Property นั้นๆ จะมีค่าเป็นแบบใดดังภาพ



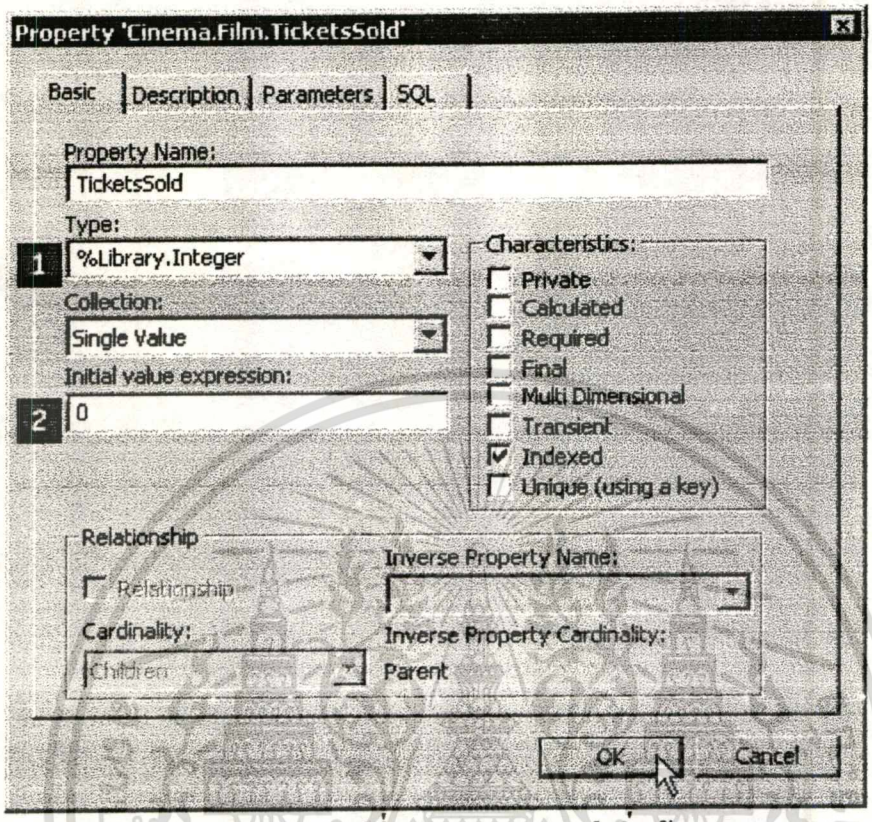
รูปที่ 5.17 ชนิดของ Collection

นอกจากนี้เรายังสามารถกำหนดลักษณะของ Characteristic ได้อีกด้วย และเมื่อเราเลือกไปที่ Tab ชื่อ Parameter เราสามารถกำหนดค่าต่างๆ ได้ เช่นค่ามากที่สุด (MAXVAL) ค่าน้อยที่สุด (MINVAL) ดังภาพ



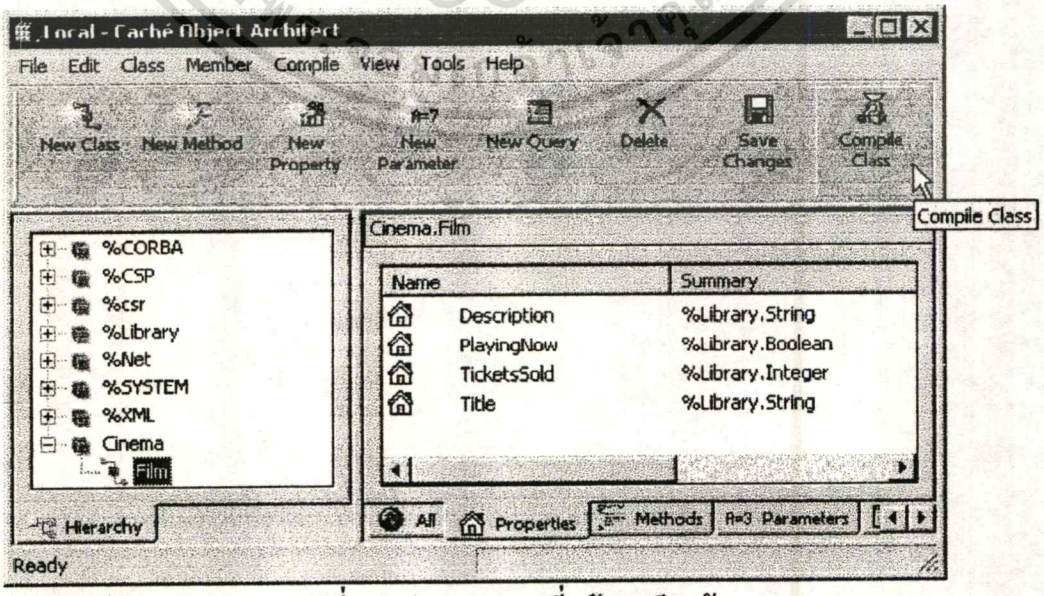
รูปที่ 5.18 การกำหนด Parameter

จากภาพเป็นการกำหนดค่ามากที่สุด ค่าต่ำที่สุด และค่า Scale นอกจากนี้เรายังสามารถที่จะกำหนดค่าเริ่มต้น (Initial Value) ได้โดยทำได้ดังภาพดังนี้



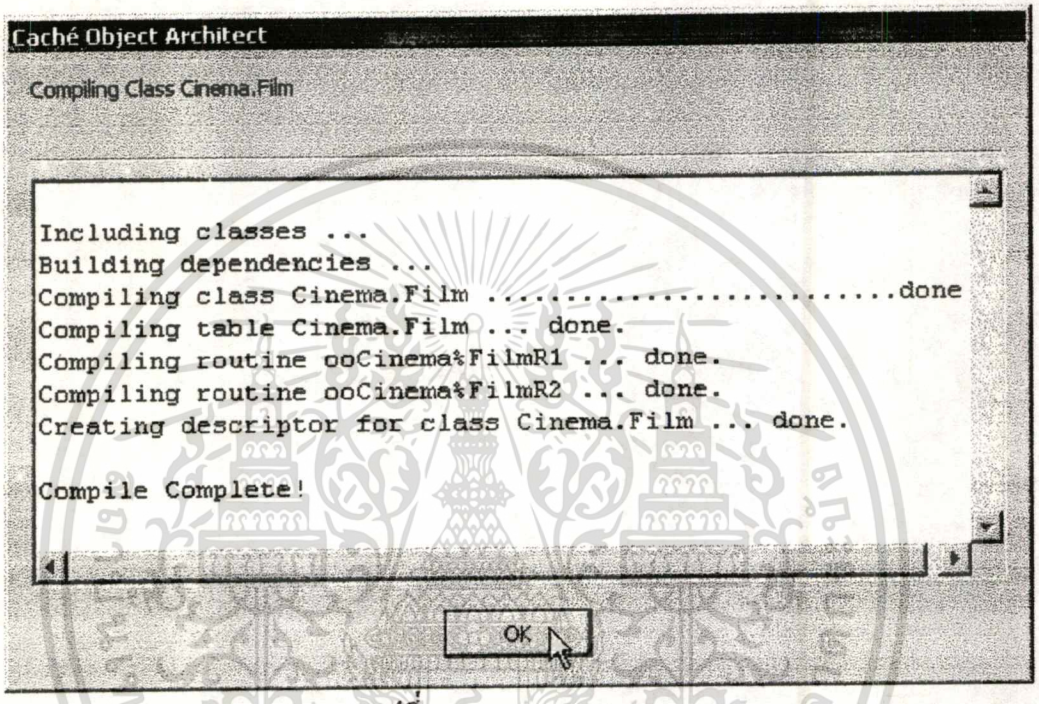
รูปที่ 5.19 การกำหนดค่าเริ่มต้น

ขั้นที่ 6 การ Compile Class : หลังจากที่เราได้สร้างและกำหนดค่าต่างๆ ของ Property ของ Class ทั้งหมดตามที่เราได้ออกแบบไว้แล้ว เราจะต้องทำการ Compile Class นั้นๆ เพื่อที่จะนำไปใช้งานดังภาพ



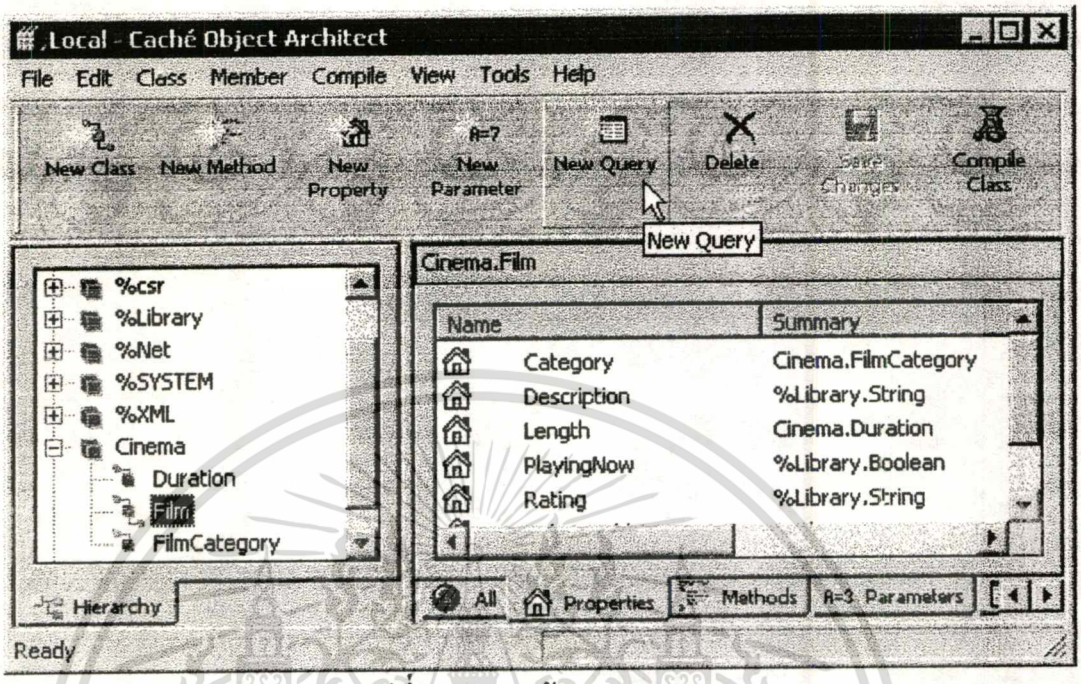
รูปที่ 5.20 Class Film ที่สร้างเสร็จแล้ว

เมื่อเราได้ Class ตามที่ต้องการแล้ว ต้องทำการ Compile โดยนำ Mouse ไป Click ที่ "Compile Class" ดังภาพข้างบน จากนั้นจะได้ผลลัพธ์ดังนี้



รูปที่ 5.21 Compile class

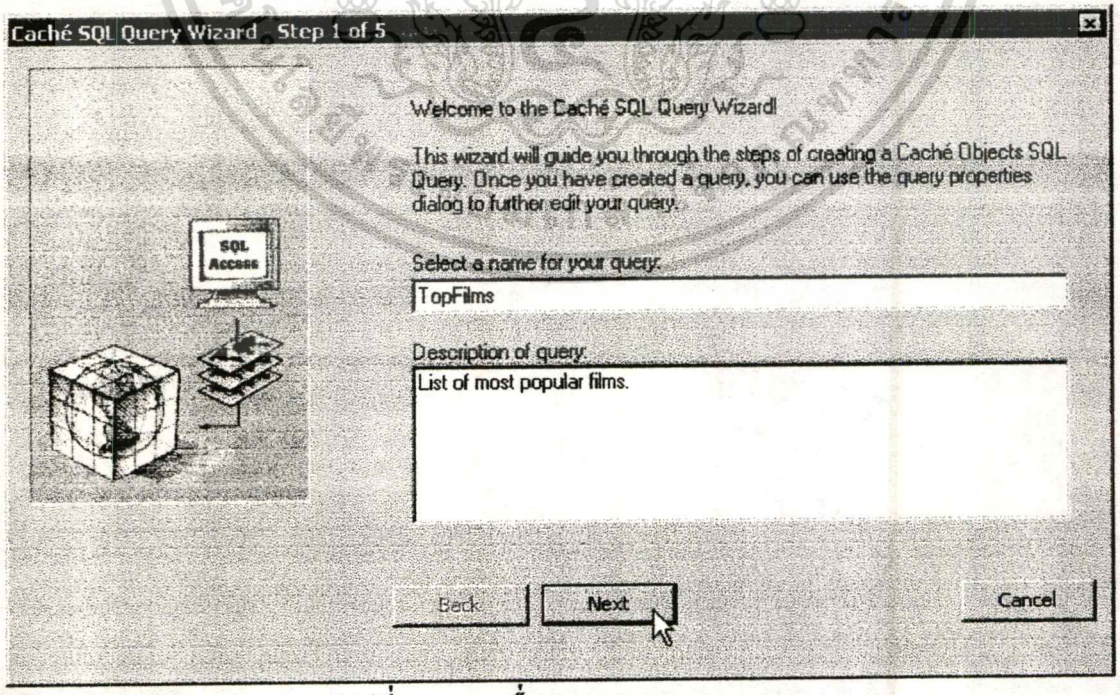
ขั้นที่ 7 การสร้าง Query เบื้องต้น : หลังจากที่เราได้สร้าง Class แล้ว เราสามารถที่จะใช้ CACHE ในการสร้าง Query ได้ โดยนำ Mouse ไป Click ที่ "New Query" ดังภาพ



รูปที่ 5.22 การสร้าง Query

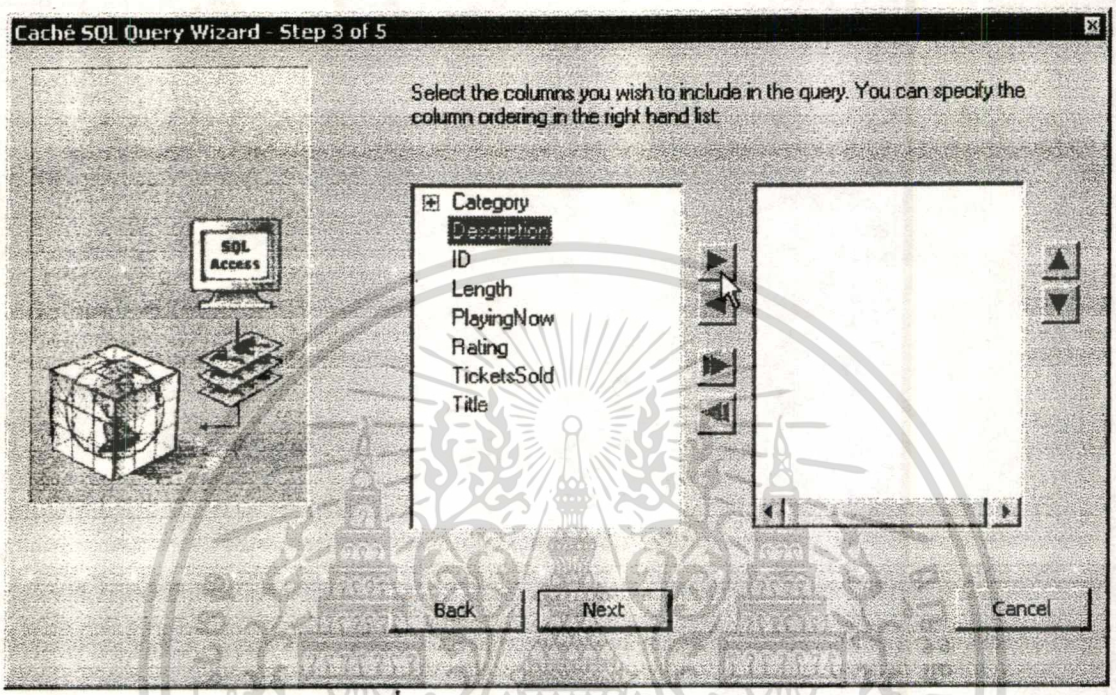
CACHE จะมี Wizard ในการสร้าง Query โดยจะมีขั้นตอนดังนี้

ขั้นที่ 7.1 กำหนดชื่อและ Description



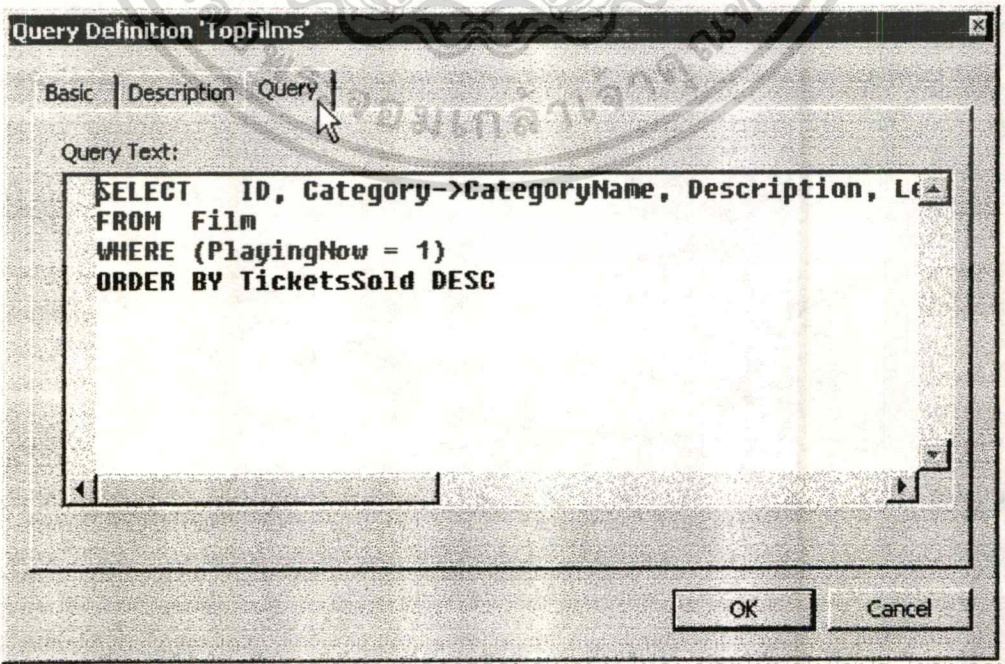
รูปที่ 5.23 ระบุชื่อและ Description New Query

ขั้นที่ 7.2 เลือก Property ที่จะใช้ในการทำ Query



รูปที่ 5.24 เลือก Property ในการสร้าง Query

ขั้นที่ 7.3 เมื่อเราทำการเลือก Property เสร็จแล้ว สามารถที่จะดู SQL Statement ที่เราสร้างไว้ได้ดังภาพ



รูปที่ 5.25 SQL ที่ได้จาก Wizard

เมื่อสร้าง Query เสร็จแล้ว ถ้าต้องการที่จะใช้งาน Query ที่สร้างนั้น เราต้องทำการ Compile Class ใหม่อีกครั้งตามขั้นตอนที่ได้กล่าวมาแล้วข้างต้น

บทที่ 6
แนวทางในการแก้ไขปัญหา

ในปัจจุบันระบบฐานข้อมูลที่ใช้กันอยู่จะเก็บเฉพาะข้อมูลล่าสุดทำให้เกิดปัญหาในการสืบค้นข้อมูลในอดีต จึงเกิดฐานข้อมูลที่สามารถจัดเก็บและสืบค้นข้อมูลที่เปลี่ยนแปลงไปตามกาลเวลาได้ เรียกว่าฐานข้อมูลอิงเวลา (Temporal Database) แต่ฐานข้อมูลอิงเวลาก็ทำให้เกิดความซับซ้อนของโครงสร้างข้อมูล ทางแก้คือเราจะใช้คุณสมบัติของระบบฐานข้อมูลเชิงวัตถุซึ่งเหมาะแก่การแก้ปัญหาข้อมูลที่มีโครงสร้างซับซ้อนมาแก้ปัญหา สุดท้ายเราจะได้ระบบฐานข้อมูลใหม่คือ ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา ที่สามารถแก้ปัญหาการสืบค้นข้อมูลในอดีตและยังสามารถจัดการกับโครงสร้างที่ซับซ้อนของฐานข้อมูลได้อีก

6.1 ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา

Temporal Object-Oriented Database

เหตุผลหลักที่หลายๆ Commercial DBMS ในปัจจุบันยังไม่ค่อยสนับสนุนแนวคิดของ Temporal ก็เพราะข้อมูลจะมีปริมาณมากและซับซ้อนเกินไป และตัว Temporal Extensions จะไปเปลี่ยนแปลง Data Model ที่เป็นพื้นฐานเดิม ทำให้การออกแบบลำบากขึ้นหรือการ Query ก็ยังไม่มีความสะดวกเพียงพอ และทำให้ใช้ประสิทธิภาพของ DBMS ได้ไม่เต็มที่ จากแนวคิดของ Temporal นั้นต้องการโครงสร้างข้อมูลที่ซับซ้อน และเราพบว่าฐานข้อมูลเชิงวัตถุ (Object-Oriented Database) นั้นถูกพัฒนาขึ้นมาเพื่อสนับสนุน Complex Object เราสามารถนำ Complex Object นั้นมาสร้างเป็นโครงสร้าง Temporal ที่ซับซ้อนได้โดยไม่กระทบต่อ Data Model ของฐานข้อมูลเชิงวัตถุ ดังนั้นฐานข้อมูลเชิงวัตถุคือทางเลือกที่ดีในการนำไปพัฒนาฐานข้อมูลเชิงเวลา (Temporal Database)

มีคนเสนอผลงานของ Temporal Relational Database มากกว่า Temporal Object-Oriented Database ก็เพราะว่า โดยพื้นฐานแล้ว Temporal Database ต้องการภาษาที่มีโครงสร้างพิเศษสามารถประยุกต์ใช้ในการจัดการกับ Operation ด้านเวลา ซึ่งก่อนหน้านี้ OODB Systems ยังมีจุดอ่อนในเรื่องของภาษาที่ใช้จัดการกับข้อมูล โดยใช้การเข้าถึง Object โดยผ่านพอยน์เตอร์เป็นหลัก ไม่มีภาษาที่ใช้ง่ายๆ ซึ่งทำให้การพัฒนาTemporal บน Object-Oriented Database ก็ยากไปด้วย และแม้ว่า Data Model ของ Object-Oriented จะเหมาะแก่การแปลงมาเป็น Temporal ก็ตาม อีกทั้งการ

สนับสนุนภาษาของ Temporal นั้นจะต้องรวม Temporal Features เข้าไปโดยตรงกับภาษาที่ใช้ เราเชื่อว่าความบกพร่องของภาษาที่ใช้ใน OODB ระบบเก่า เป็นผลให้เกิดความบกพร่องของภาษา Temporal ที่ใช้ใน OODB ในอดีตด้วย แต่มันจะไม่เป็นแบบนั้นแล้วเพราะได้มีการสร้างภาษาสำหรับ OODB ที่มีการจำลอง Interface ให้เหมือนกับภาษา SQL ส่วนในเรื่องความไม่เป็นมาตรฐานของ OODB ก็กำลังกลายเป็นเรื่องเก่าแล้วเพราะเมื่อไม่นานมานี้ ODMG ได้เป็นผู้กำหนดมาตรฐานของ OODB และได้เผยแพร่มาตรฐานนั้นออกมาคือ ODMG-93 Release 1.2 ซึ่งสนับสนุนภาษาในการ Query Object ในระดับสูง เราเรียกว่า OQL ที่สำคัญคือผู้ผลิต OODB ทั้งหลายมีข้อตกลงกันในการสนับสนุน OQL Interface เข้าสู่ระบบของตัวเอง ในที่นี้เราจะกำหนด TOQL ขึ้นมาซึ่งเป็นส่วนขยายจาก OQL แต่เราจะเพิ่มฟังก์ชันในการจัดการกับข้อมูล Historical เข้าไปด้วย ดังนั้นการ Query โดยปราศจากส่วนที่เราเพิ่มเติมเข้าไปก็จะเป็น OQL Query ธรรมดา

หัวข้อที่จะต้องพิจารณาเมื่อจะออกแบบ Temporal OODB ก็คือเราจะบวกมิติของเวลาเข้าไปกับส่วนไหนของ object เรามี 3 ทางเลือกที่จะต้องพิจารณาก็คือ Object-Versioning คือการรวมข้อมูล Historical เข้าไปกับตัว Object เลย, Attribute-Versioning คือการรวมข้อมูล Historical เข้าไปกับ Attribute ของ Object และ Type-Versioning คือการรวมข้อมูล Historical ไปยัง Type ของ Object หรือ Value โดย Type-Versioning เป็นแบบที่เหมาะสมที่สุดในการนำมาใช้กับ Temporal OODB เพราะสนับสนุนโดยตรงกับ Attribute ที่เป็น Complex Values ตัวอย่างเช่น Attribute ที่เป็น Set ของ Tuples เราอาจจะกำหนด Version (Time) ให้กับตัว Tuples แทนที่จะเป็นทั้ง Set ในการ Implement Type-Versioning จะกำหนดข้อมูลที่ต้องการให้เป็น Historical ให้เป็น Type Object โดยนิยามเป็น Constructor ได้ เช่น Temporal<T> ซึ่งเป็นการเพิ่มระดับของ Type T ให้เป็น Temporal ซึ่งจะบรรจุ Historical ของ Version ทั้งหมดของ T

Query Language ของ Temporal OODB จะต้องสนับสนุนหลายๆ Type ที่ต่างกันของ Temporal Query รวมทั้ง Temporal Projection และ Coincidence Queries

หลักการของ Type-Versioning คือการไปนิยามตัว Data Type ให้เป็นชนิด Temporal ซึ่งวิธีการนี้คือการทำ Parameterize Type

6.1.1 Parameterize Type (Parametric Polymorphism)

Parameterize Type หรือ Parametric Polymorphism ก็คือการนำ Parameter ที่ผ่านมาทางชื่อ Class มาเป็น Data Type ของข้อมูลภายใน Class ซึ่ง Parameter ที่ผ่านมานั้นจะต้องเป็นชนิดของ Data Type หรือชนิดของ Class ที่สามารถ Define ได้

สมมติเรามี Class Stack ซึ่งจะเอาไว้ใช้เก็บข้อมูลลง Stack แต่มีปัญหาที่ว่าข้อมูลที่เก็บลง Stack นี้มีหลากหลายชนิด ซึ่งทำให้เราต้องสร้าง Stack ขึ้นมาหลายๆ ชุดตามจำนวนชนิด Data Type ของข้อมูลที่ต้องการเก็บ ทำให้มีความยุ่งยากและความซ้ำซ้อนเนื่องมาจาก Class ที่เพิ่มขึ้น วิธีแก้ปัญหานี้คือการนำ Parameterize Type มาใช้ดังตัวอย่างนี้

```
public class Stack<T>
{
        T data;

        void push(T data);
        T pop();
}
```

T คือ Data Type ที่ผ่านมาทาง Parameter อาจจะเป็น Integer, String, Float หรือ Class ใดๆ ก็ได้ ส่วน Stack คือ Class ที่ใช้งาน Parameterize Type

ตัวอย่างการเรียกใช้งาน Class ที่ Define ด้วย Parameterize Type

```
public class CallStack
{
        Stack<int>      s1 = new Stack<int>();
        int             i[] = { 1, 2, 3, 4, 5 };
        Stack<String>   s2 = new Stack<String>();
        String          str[] = { "one", "two", "three" }

        s1.push(i[1]);
        System.out.println((String)s1.pop());
        s2.push(str[1]);
        System.out.println(s2.pop());
}
```

6.1.2 Data Model และ Query Language

มาตรฐานของ ODMG'93 ได้จัดเตรียมภาษาสำหรับการนิยาม Object Classes และภาษาสำหรับการจัดการกับข้อมูล (Data Manipulation Language) คือ ODL และได้เตรียมภาษา OQL สำหรับการ Query ข้อมูล แนวทางของเราที่จะใช้คือสร้าง TODL ซึ่งเป็นส่วนขยายของ ODL โดยเพิ่ม Type ในส่วนที่จัดการกับมิติของเวลาเข้าไปด้วย กับการ Query ก็เช่นกันเราใช้ TOQL ซึ่งเป็นส่วนขยายของ OQL อีกที

ดูตัวอย่างการนิยาม Database Schema ของมหาวิทยาลัยแห่งหนึ่ง ในรูปแบบของ TODL

```
interface Department ( extent Departments  keys dno, name ): persistent
{
        attribute Short dno;
        attribute String name;
        attribute Temporal<Instructor> head;
        temporal relationship Set<Instructor> instructors
                inverse Instructor::dept;
        temporal relationship Set<Cource> courses_offered
                inverse Course::offered_by:
};

interface Instructor ( extent Instructors  key ssn ): persistent
{
        attribute Short ssn;
        attribute String name;
        attribute Temporal<Integer> salary;
        attribute Temporal<String> rank;
        temporal relationship Department dept
                inverse Department::instructors;
        temporal relationship Set<Cource> teaches
        inverse Course::taught_by:
};

interface Course ( extent Courses  keys code, name ): persistent
{
        attribute Short code;
        attribute String name;
        temporal relationship Department offered_by
```

```
        inverse Department::courses_offered;
    temporal relationship Instructor taught_by
        inverse Instructor::teaches;
    temporal relationship Set<Cource> in_prerequisite_for
        inverse has_prerequisites;
    temporal relationship Set<Cource> has_prerequisites
        inverse is_prerequisite_for;
};
```

จะเห็นว่า Parameterized Type หรือ Constructor ของ Temporal <T> คือการบวกมิติของเวลาเข้าไปยัง Type T ตัวอย่าง Type ของ Attribute Head ใน Class Department คือ Temporal<Instructor> หมายถึงมีการเก็บข้อมูลในอดีตของ Heads ใน Department ทั้งหมด ส่วน Type **relationship** ใน ODL สามารถนิยามได้เป็น one-to-one, one-to-many, และ many-to-many ซึ่งเป็นความสัมพันธ์ระหว่าง Object Type ที่เกิดขึ้น ตัวอย่างเช่น Relationships ของ Instructors ใน Class Department คือ Set ของ Instructors ทั้งหมด ที่ซึ่งทำงานใน Department และ Inverse Relationship คือ dept ที่อยู่ใน Class Instructor ความสัมพันธ์นี้คือ one-to-many relationship ระหว่าง Instructors และ Department โดยทั่วไป Relationship ระหว่าง Object สามารถแบ่งเป็น Temporal และ Non-Temporal ได้ Keyword ที่ใช้คือ temporal ใช้นำหน้าคำว่า relationship ถูกใช้เพื่อสร้างความสัมพันธ์ที่ขึ้นอยู่กับเวลา (เพื่อทำให้ Relationship นั้นกลายเป็น temporal relationship) Object ซึ่งเป็น temporal relationship กันเรียกว่า Temporal Entities จาก Schema ในตัวอย่างไม่มี Non-Temporal Relationship เลย ตัวอย่างของ Relationship ที่ไม่ใช่ Temporal ก็คือ ความสัมพันธ์ระหว่างพ่อกับลูก ซึ่งจะพบว่าเป็นความสัมพันธ์ที่ไม่เปลี่ยนตามกาลเวลา ส่วนตัวอย่างของ Temporal Relationship คือความสัมพันธ์ระหว่าง Instructors และ Courses ซึ่งมีการสร้าง Entities ของ Temporal ร่วมกัน เช่น Temporal Relationship ระหว่าง taught_by : Instructor และ teaches : Set<Course> กลายมาเป็นความสัมพันธ์ระหว่าง taught_by : Temporal <Instructor> และ teaches : Temporal<Set<Course>>

Query Language TOQL เป็น Super Set ของ OQL โดยใช้รูปแบบ Syntax ของ OQL แต่มีการเพิ่มของคำสั่งเพื่อใช้จัดการกับข้อมูลที่เป็น historical ถ้าสัญลักษณ์ที่ถูกเพิ่มขึ้นมาไม่ได้ถูกใช้งาน การ Query ใน TOQL ก็จะเป็นการ Query ใน OQL ธรรมดา ดังตัวอย่างข้างล่าง

Query 1) Give me the name and the rank of all instructors who teach at least one CSE course.

```
select i.name, i.rank
from i in Instructors,
        c in i.teaches
where c.offered_by.name = 'CSE'
```

ตัวอย่างนี้ไม่ได้พิจารณาข้อมูลเกี่ยวกับ Historical ของ Object เลย โดยมันจะ Retrieve ข้อมูลล่าสุดของ Object ออกมา (ที่เวลาปัจจุบัน)

เราได้นิยาม Class TODL สำหรับใช้กับข้อมูล Historical คือ

Time_Interval คือ Class ของช่วงเวลา

Temporal< T > คือ Class ที่ใช้แทน History ของ T

Class Time_Interval มี 2 Public Attributed คือ start และ end ทั้งคู่มี Type เป็น **Time** ของ TOQL ซึ่งเป็นตัวชี้จุดเริ่มต้นและสิ้นสุดของช่วงเวลา โดย Type Time interval สามารถแสดงได้ดังนี้

Operation	Type
[*time*]	**Time_Interval**
[*time1*, *time2*]	**Time_Interval**

ตารางที่ 6.1 แสดงโครงสร้างของ Time_Interval

โดย Type ของ *time*, *time1* และ *time2* คือ **Time** ช่วงเวลา [*time*] ระบุแค่เวลา ณ ขณะใดขณะหนึ่ง ในขณะที่ช่วงเวลา [*time1*,*time2*] ระบุตำแหน่งของเวลาระหว่าง *time1* และ *time2* TOQL สนับสนุนคำสั่งที่ใช้ในการเปรียบเทียบสองช่วงเวลา a และ b เช่น a **before** b มีค่าเท่ากับ a.end < b.start

Temporal<T> เป็น Parameterized Class ที่รวมมิติของเวลาเข้าไปยัง Type T มี 4 Operation ใน TOQL ที่ใช้เพื่อจัดการกับ Objects Temporal <T>

Operation	Type	Explanation
e@version	**T**	Version projection
e : *time*	**T**	Time projection
e : *interval*	**Temporal<T>**	Interval projection
?v	**Time_Interval**	The time interval of *v*
#v	**Integer**	The version of *v*

ตารางที่ 6.2 แสดง Operation ในการจัดการ Temporal <T>

ซึ่ง

- *time* คือเครื่องหมายของชนิดข้อมูล **Time**
- *e* คือเครื่องหมายแทน Type **Temporal<T>**
- *version* คือเครื่องหมายของชนิดข้อมูล Integer ซึ่งขึ้นอยู่กับ Parameter last (ซึ่งบอกถึง Version สุดท้ายของ Object)
- *interval* คือเครื่องหมายของชนิดข้อมูล **Time_Interval**
- *v* คือ ตัวแปร **temporal**

<u>ตัวอย่าง</u> กำหนดให้ Smith เป็น Object ซึ่งแทน Instructor ด้วยชื่อ Smith

- **Version Projection** คือ Smith.dept@n จะ Return Version ที่ n ของ dept ซึ่งคือจำนวนของ Department ที่ซึ่ง Smith เคยทำงานมาแล้ว โดย n=0 คือ Version แรก และ n=**last** คือ Version หลังสุด ถ้าไม่มี Version และ Expression นี้จะถูกยกเลิกในขณะที่ Runtime

- **Time Projection** คือ Smith.dept:1/1/90 Return ค่าของ Type Department ซึ่งแทน Department ซึ่ง Smith ได้ทำงานในวันที่ 1/1/90 ถ้า Smith ไม่ได้ทำงานใน Department ที่เวลานั้น Expression นี้ก็จะถูกยกเลิกในขณะ Runtime เช่นกัน เหมือนกับ Smith.dept:**now** จะ Return Department ปัจจุบันของ Smith ค่า Default ถ้าไม่มีการระบุ Time Constraint คือ Smith.dept จะเท่ากับ Smith.dept:**now** ดังนั้น Smith.dept:**now** ไม่ค่อยจำเป็นเหมือนกับ Smith.dept@last ถ้า Smith นั้นไม่ได้ทำงานในแผนกนั้น ณ เวลาปัจจุบัน กรณีแรก Expression จะถูกยกเลิก ในขณะที่กรณี 2 จะ Return ค่า Department สุดท้ายที่เขาอยู่

เพื่อหลีกเลี่ยงการยกเลิกเราอาจจะใช้ Interval Projections แทนที่ Version ของ Time Projection ตัวอย่าง

- **Interval Projection** ของ Smith.dept:[1/1/98, 1/1/90] จะ Return ค่าของ Type Temporal< Department > ที่บรรจุ History ของ Department ซึ่ง Smith ทำงานระหว่าง 1/1/88 ถึง 1/1/90 ค่าข้อมูลในอดีตนี้บางทีอาจจะว่างเปล่า คล้ายกับ Interval Projection Smith.dept:[1/1/90] Return ค่าของ Type Temporal< Department > ซึ่งอาจจะเป็นค่าที่ว่างเปล่าของ Department ถ้า Smith ไม่ได้ทำงานในแผนกนี้ในเวลานั้น ในการ Retrieve ค่าทั้งหมดของ History ของ Department ที่ Smith อยู่ คือ Smith.dept:[**start, now**], โดย **start** คือจุดเริ่มต้นของเวลา

พิจารณาตัวอย่างการ Query โดยใช้ Temporal Projection

Query 2) Give me the name and the rank of all person who were CSE instructors between 1/1/88 and 1/1/90

```
select i.name, r
from i in Instructors,
        d in i.dept:[1/1/88,1/1/90],
        r in i.rank:?d
where d.name = 'CSE'
```

Interval Projection i.dept:[1/1/88,1/1/90] จะคืนค่า History ของ Department ซึ่ง Instructor i นั้นทำงานอยู่ ตัวแปร d เรียกว่า Temporal Variable ค่าของ d เรียกว่า Temporal Element และสามารถดูอีกสามส่วนที่เกี่ยวข้องคือ Time_interval, Version Number และ Department เมื่อ d ถูกกล่าวถึงใน Query มันจะอ้าง Department จริงๆ เมื่อ ?d ถูกกล่าวถึง มันจะอ้างอิงถึงช่วงเวลาของ d และเมื่อ #d ถูกกล่าวถึงมันจะอ้างถึง Version Number ของ d ตรง Interval Projection i.rank?d ใช้ช่วงเวลาของ d ในการ Retriever Rank ของ i ที่คาบเวลานั้น สิ่งต่างๆ เหล่านี้คือแนวทางที่จะแสดงใน TOQL ตัวอย่างต่อไปนี้ใช้ now แทน Current Time เพื่อ Retriever History ระหว่างเวลา

Query 3) List the salary history of all instructors who were employed sometime during the last 4 years.

```
select i.name, s
from i in Instructors,
     s in i.salary:[now-4years,now],
```

6.1.3 Data Type ของ Temporal

เราได้ทำการ Define Classes สำหรับ Time_Interval และ Temporal< T > ดังข้างล่าง

```
Interface Time_Interval
{
    attribute Time start;
    attribute Time end;
    Integer duration();
    Time_Interval intersect(in Time_Interval te);
    Boolean equal(in Time_Interval te);
    Boolean overlaps(in Time_Interval te);
    Boolean before(in Time_Interval te);
    Boolean contains(in Time_Interval te);
};

parameterized type Temporal < T >
{
    Structure { Time_Interval valid_time, Integer version, T value } Temporal_value;
```

```
attribute List< Temporal_value > history;
attribute Integer size;
attribute Temporal_value latest_version;
Temporal_value version_projection (in Short version)
        raises(no_such_version);
Temporal_value time_projection (in Time time)
        raises(not_defined_at_that_time);
Temporal< T > interval_projection(in Time_Interval I);
new_version(in T value, in Time event);
close_version(in Time event);
};
```

- ก่อนอื่นเรากำหนด Type (Structure) Temporal_value ขึ้นมาก่อนซึ่งประกอบด้วยค่า 3 ค่าตามตัวอย่าง รวมทั้ง History ของ Object ของ Type T
- ส่วนต่อไปคือ List ของ Temporal_value ในส่วนหัวของ List คือ Version สุดท้ายในขณะที่ Element สุดท้ายของ List คือ Version เริ่มต้น (Version Number 0) ตรงส่วน Version ท้ายสุดสามารถเลือกได้ว่าเปิด (ถ้า validtime end=now) หรือปิด ซึ่งเป็นตัวชี้ว่า Object ไม่มีอยู่ ณ เวลาปัจจุบัน
- Attribute size คือรายการ Size Attribute
- latest_version คือ Pointer ชี้ไปยัง Version สุดท้าย ถ้า Version สุดท้ายถูกเปิด นอกนั้นเป็น null ส่วนนี้ถูกใช้เพื่อการเข้าถึง Object อย่างรวดเร็ว ณ เวลาปัจจุบัน

โครงสร้างสัญลักษณ์ TOQL นำมาแปลงกลับเป็น OQL expression ได้ดังนี้

[*time*]	⇨	**struct(start:***time*, **end:***time*)
[*time1*,*time2*]	⇨	**struct(start:***time1*, **end:***time2*)
e@version	⇨	*e*.version_projection(version).value
e:time	⇨	*e*.time_projection(time).value
e:interval	⇨	*e*.interval_projection(interval).history
?v	⇨	*e*.valid_time
#v	⇨	*e*.version

ถ้า Temporal Variable *v* ถูกกล่าวถึงใน Query (ในเทอมอื่นนอกเหนือจาก ?*v* หรือ #*v*) มันจะถูกแปลงเป็น *v*.value และ projection e.A ซึ่ง Expression *e* ของ Type Temporal<T> ถูกแปลงเป็น e:**now**.A

สำหรับตัวอย่าง Query1 และ Query2 ถูกแปลงได้ดังนี้

```
select i.name, i.rank.latest_version.value
from i in Instructors,
        c in i.teaches.latest_version.value
where c.offered_by.latest_version.value.name = 'CSE'

select i.name, r.value
from i in Instructors,
        d in i.dept.interval_projection(struct(start:1/1/88,end:1/1/90)).history,
        r in i.rank.interval_projection(d.valid_time).history
where d.value.name = 'CSE'
```

การเปลี่ยนแปลง Element *e* ของ Type Temporal<T> กับค่าใหม่ *v* ของ Type T ถูกเรียกว่า *e*.new_version(*v*,*t*), ซึ่ง *t* คือเวลาเมื่อ Update มีผล Operation จะปิด Version สุดท้ายของ *e* (คือมันจะแทนค่าของ *e*.latest_version.end จาก **now** ไปยัง *t*) และจะ Insert Version ใหม่ด้วย Valid Time [*t*, **now**] ที่ส่วนบนของ list (Attribute latest_version จะเปลี่ยนตามด้วย) การสร้าง Object ใหม่ของ Type Temporal<T> กับสถานะเริ่มต้น *v* ของ Type T ที่เวลา *t* จะสำเร็จโดยเรียก *e*.new_verison(*v*,*t*) ตามหลักแล้วการลบ *e* ที่เวลา *t* จะปิด Version สุดท้ายของ *e* โดยใช้ *e*.close_version(*t*)

6.2 แนวทางในการ Implement ระบบ

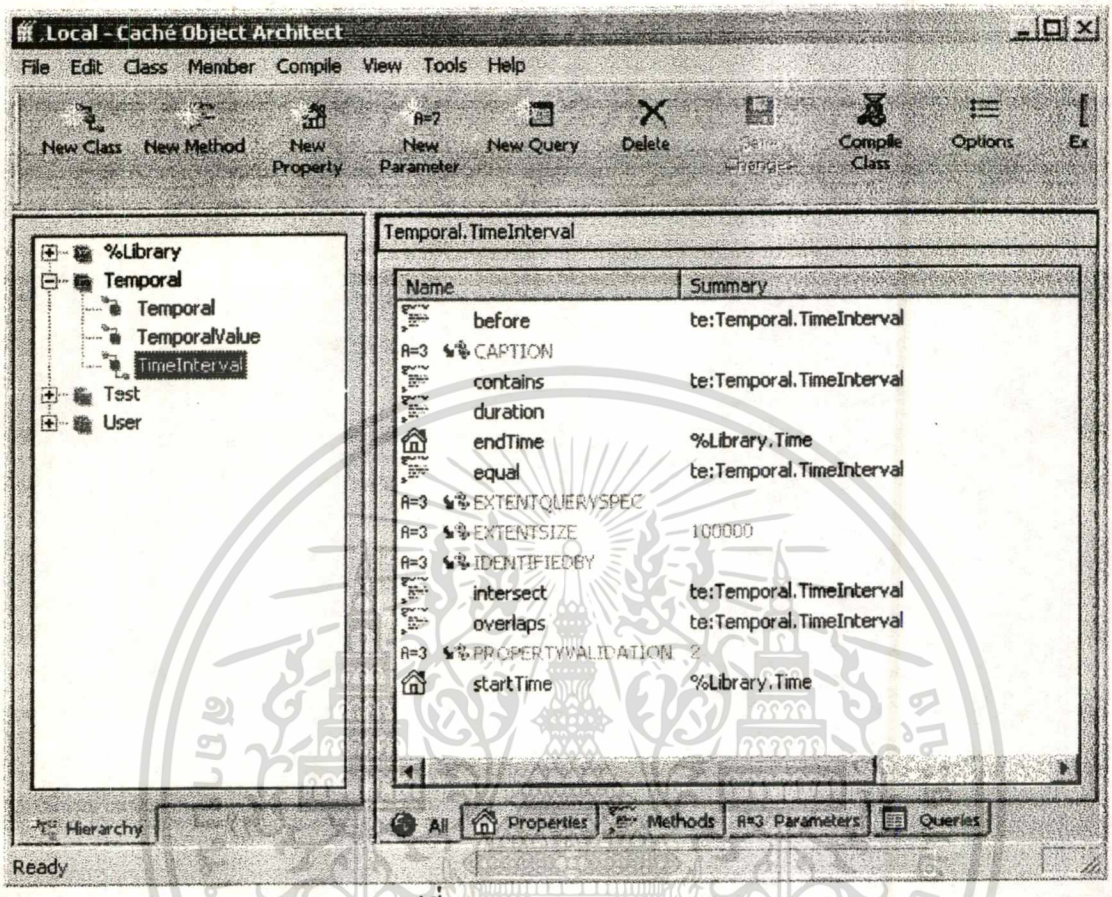
- สร้าง Temporal Class ใน Cache
- สร้าง Persistent Class ทั่วๆ ไปของ database โดยมีการนำ Temporal Class มาใช้ด้วย
- เขียนโปรแกรมด้วยภาษา Java ติดต่อกับ CACHE (ดูในบทถัดไป)
- สร้าง User Interface (ดูในบทถัดไป)

6.3 สร้างระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลาด้วย CACHE

เราจะทำการสร้างระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา และสร้าง Temporal Class ที่เราได้คิดไว้ ด้วย CACHE

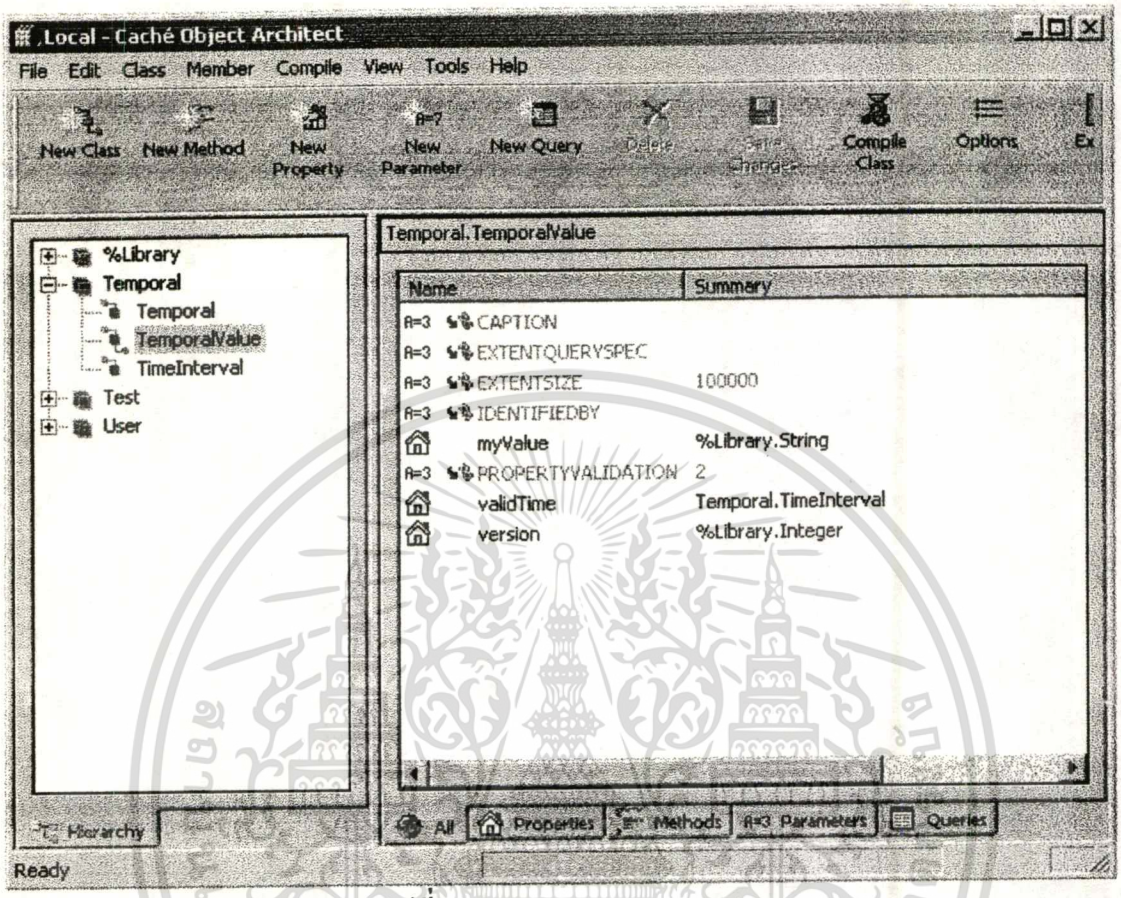
โดยในขั้นตอนแรกเราจะทำการสร้าง Class ที่ใช้สำหรับจัดการกับชนิดข้อมูลที่เป็น Temporal ดังนี้

1. สร้าง Class -> TimeInterval
   - โดยเลือกชนิดของ Class เป็น Embeddable (Embedded เพื่อที่ว่า Class นี้จะถูกฝังลงใน Class Parent ของมัน ซึ่งจะทำให้การทำงานเร็วขึ้นเพราะไม่ต้องสร้าง Link แบบ Relational)
   - เพิ่ม Properties ต่างๆ ตามที่ได้ออกแบบไว้
     i. Time **startTime**
     ii. Time **endTime**
   - เพิ่ม Methods ต่างๆ ตามที่ได้ออกแบบไว้
     i. Integer **duration**();
     ii. TimeInterval **intersect** (TimeInterval te);
     iii. Boolean **equal** (TimeInterval te);
     iv. Boolean **overlaps** (TimeInterval te);
     v. Boolean **before** (TimeInterval te);
     vi. Boolean **contains** (TimeInterval te);



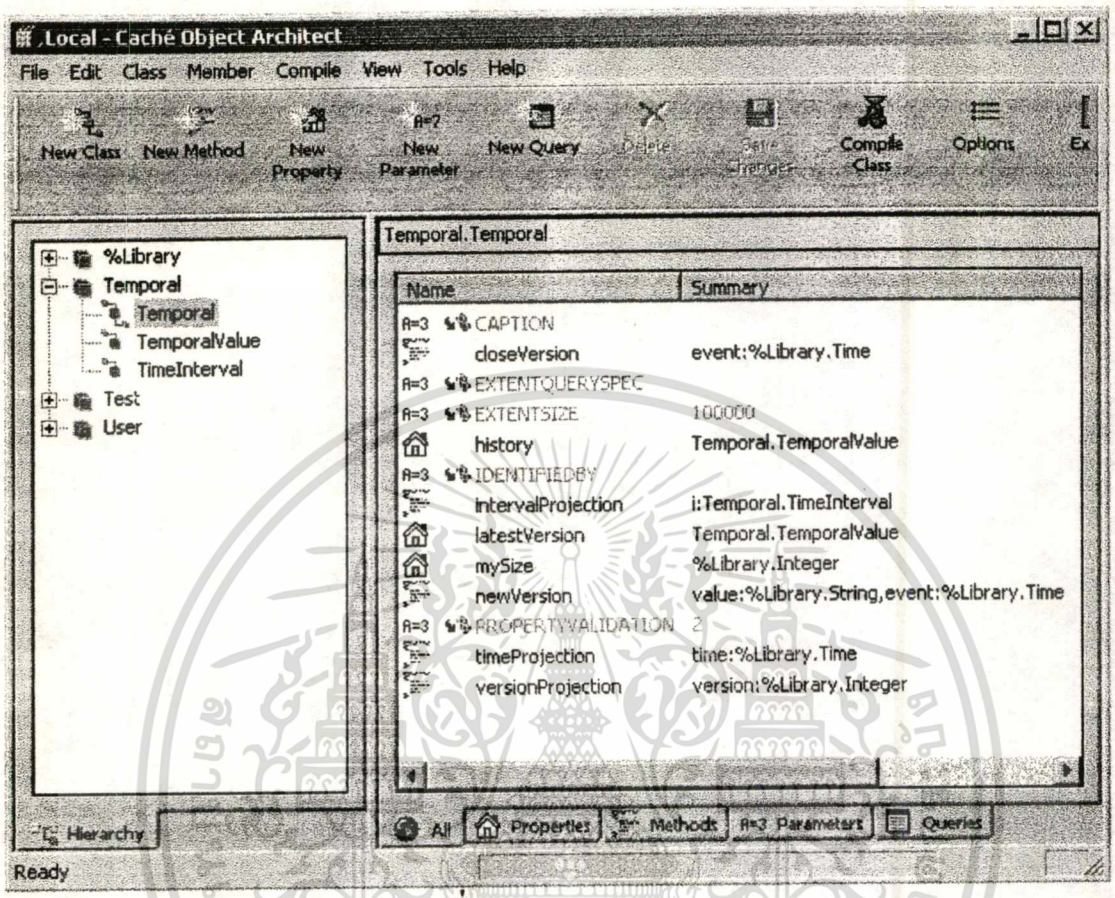
รูปที่ 6.1 Class TimeInterval

2. สร้าง Class -> TemporalValue
   - โดยเลือกชนิดของ Class เป็น Embeddable
   - เพิ่ม Properties
     i. String **myValue** (จริงๆ แล้วควรจะเป็น Parameterized Type T แต่ใน Cache ไม่มีการ Implement Parameterized Type)
     ii. TimeInterval **validTime**
     iii. Integer **version**
   - Class นี้ไม่มี Method



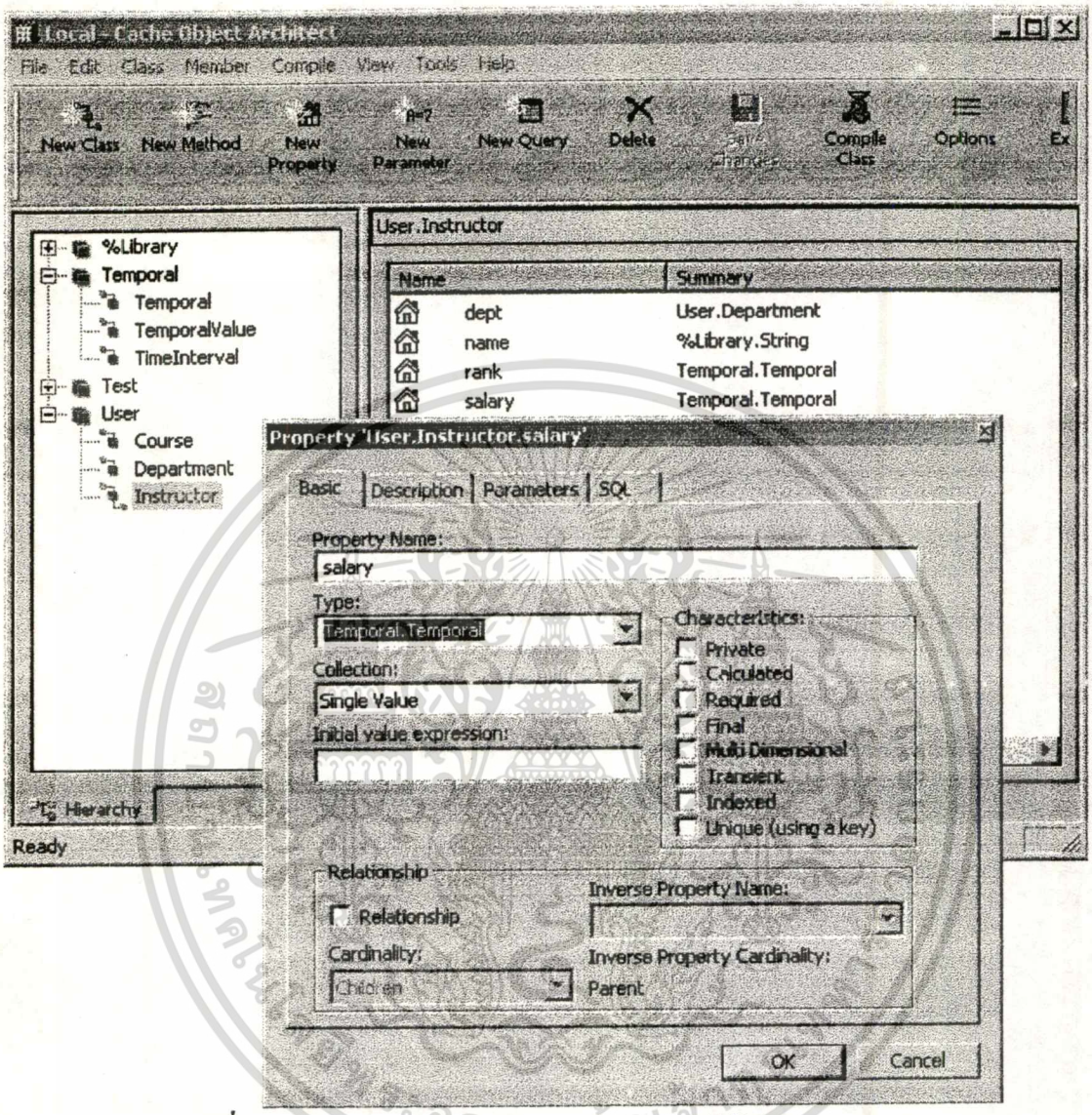
รูปที่ 6.2 Class TemporalValue

3. สร้าง Class -> Temporal
   - โดยเลือกชนิดของ Class เป็น Embeddable
   - เพิ่ม Properties
     i. TemporalValue **history**
     ii. TemporalValue **latest Version**
     iii. Integer **my Size**
   - เพิ่ม Methods
     i. TemporalValue **version Projection** (Short version)
     ii. TemporalValue **time Projection** (Time)
     iii. Temporal **interval Projection** (TimeInterval i)
     iv. Void **new Version** (String value, Time event)
     v. Void **close Version** (Time event)



รูปที่ 6.3 Class Temporal

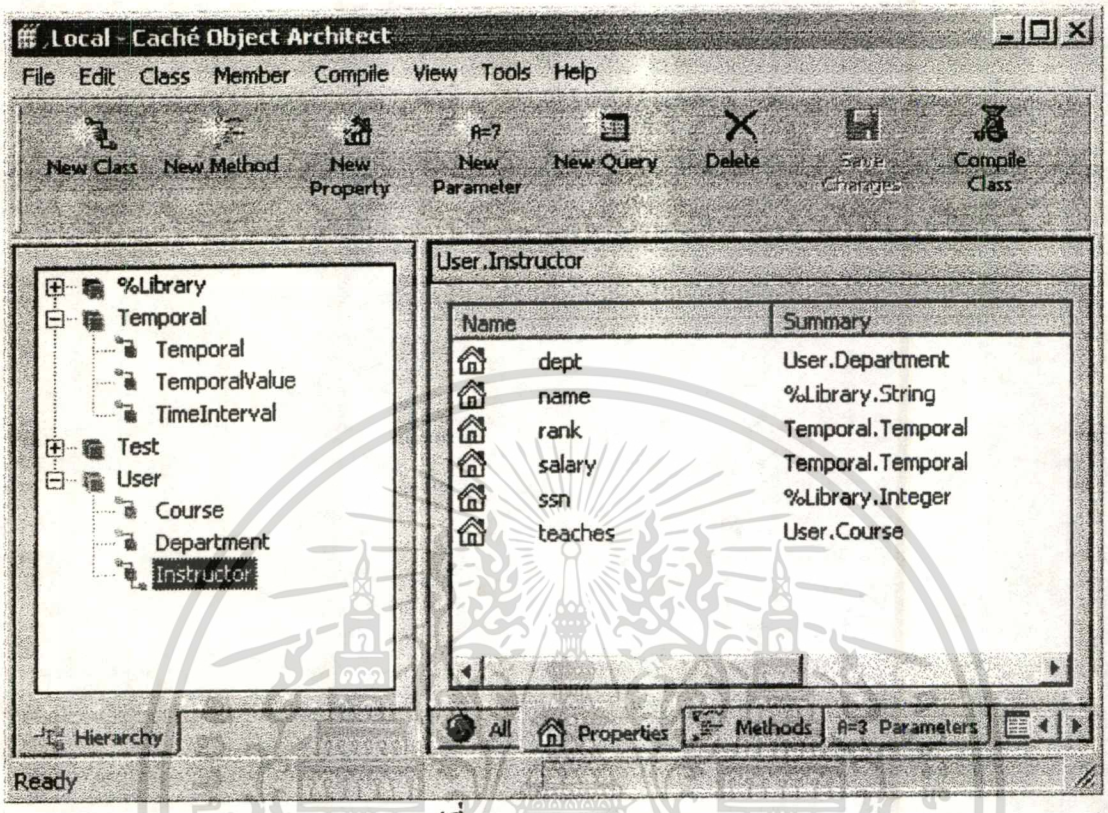
เมื่อได้ Class Temporal พื้นฐานแล้ว หากเราจะใช้ Type Temporal ใน Class ใดๆ ก็เพียงแต่กำหนด Type ของ Property นั้นให้เป็น Temporal ดังรูป



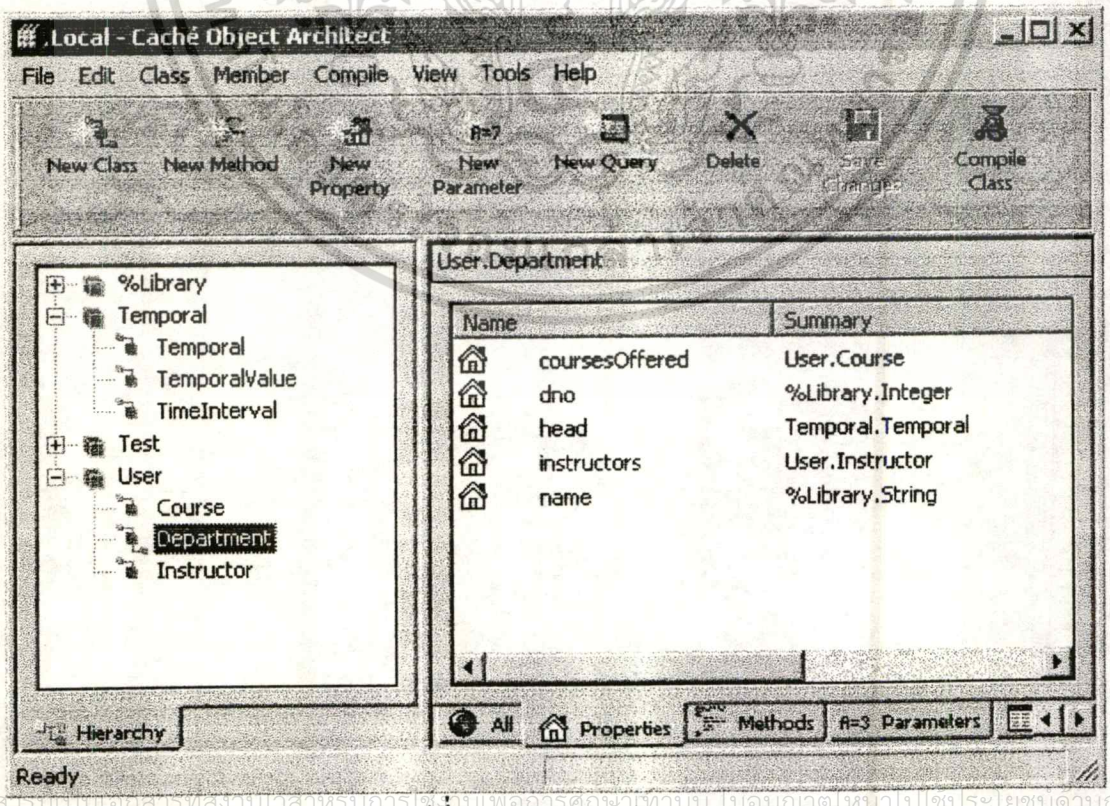
รูปที่ 6.4 แสดงการกำหนด Type ของ Property ให้เป็น Temporal

ในขั้นตอนต่อไป คือนำ Temporal Type นี้ไปใส่รวมกับระบบตัวอย่างที่เราสร้างขึ้นเพื่อทำการทดลอง (ตัวอย่างจากหัวข้อที่แล้ว 6.1) โดยระบบที่เราสร้างนั้นมี Class อยู่ 3 Class ด้วยกันคือ Instructor, Department, และ Course

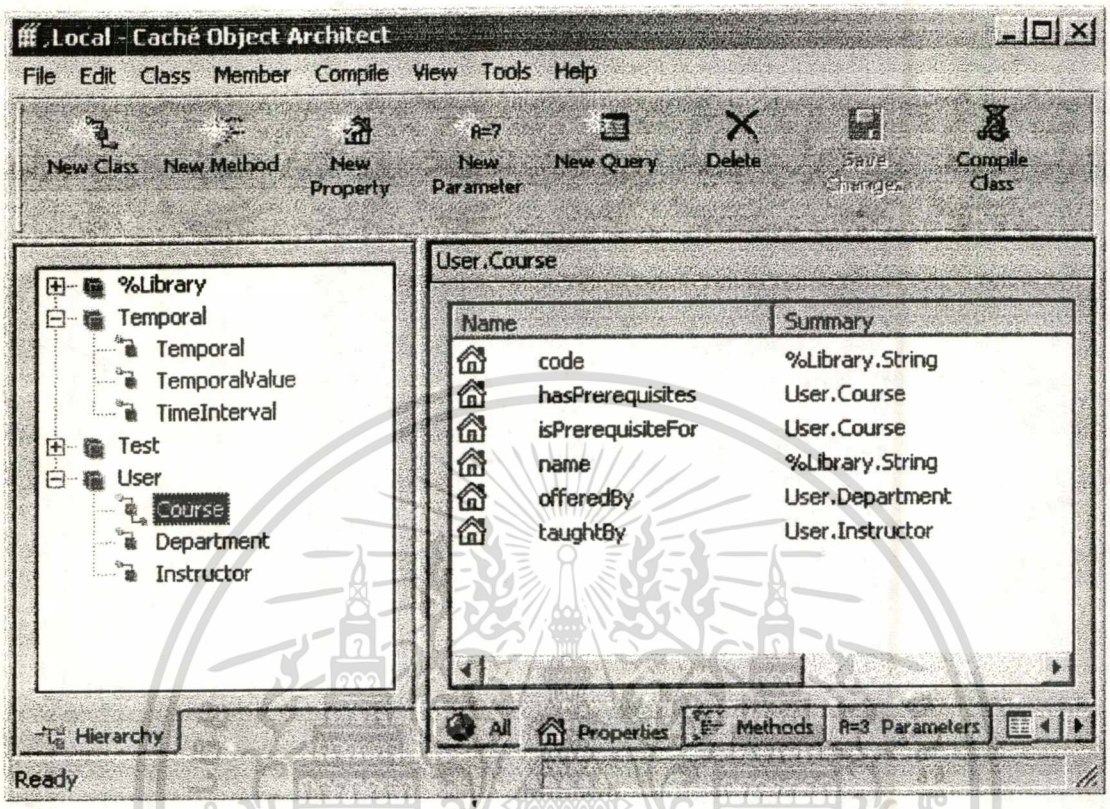
เมื่อนำมาสร้างด้วย Object Architect ของ Cache จะได้ดังนี้



รูปที่ 6.5 Class Instructor



รูปที่ 6.6 Class Department



รูปที่ 6.7 Class Course

บทที่ 7
ตัวอย่างโปรแกรมประยุกต์

เนื่องจากโครงการพัฒนาระบบงานนี้เป็นการศึกษาเกี่ยวกับด้าน Object-Oriented Database ดังนั้นภาษาที่นำมา Implement ก็ควรจะเป็นภาษาทางด้าน Object-Oriented เหมือนกัน ซึ่งภาษาในด้าน Object-Oriented ที่ได้รับความนิยมมากที่สุดในขณะนี้คงไม่พ้นภาษา Java เพราะฉะนั้น เราจึงใช้ภาษา Java ในการ Implement ระบบ อีกทั้งการทำงานระหว่าง Object-Oriented กับ Object-Oriented จะทำให้ Performance ดีกว่า กระ Map ไปมาระหว่าง Object-Oriented กับ Procedure-Oriented

7.1 พัฒนาโปรแกรมประยุกต์สำหรับระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลาด้วยภาษา Java

มี 2 ทางเลือกในการสร้าง Java Application กับ CACHE คือ

1. ใช้ SQL – ให้ Java application access กับ CACHE Database โดยผ่าน JDBC หรือ ODBC
2. ไม่ใช้ SQL – ก็คือ ใช้การเข้าถึง Object โดยตรงจาก Java Object

(ในการพัฒนาระบบนี้เราจะใช้วิธีที่ 2 เป็นส่วนใหญ่ เพราะวิธีแรกจะเกิด Overhead ที่การทำงานของ SQL JDBC)

จาก Classes ต่างๆ ที่เราสร้างใน Object Architect ของ CACHE สามารถ Export ออกมาเป็น Code ภาษา Java ได้ เพื่อที่จะนำไป Implement ต่อ ซึ่งเราก็ทำดังนั้น และเราจะได้ Classes ที่ Export ออกมาดังนี้

Class	Properties	Methods
Temporal	m_history m_latestVersion m_mySize	Temporal(ObjectServer) _close() gethistory() sethistory(ListOfObjects) getlatestVersion() setlatestVersion(TemporalValue)

Class	Properties	Methods
		getmySize() setmySize(int)
TemporalValue	m_myValue m_validTime m_version	TemporalValue(ObjectServer) _close() _initClass(Hashtable) getmyValue() setmyValue(String) getvalidTime() setvalidTime(TimeInterval) getversion() setversion(int)
TimeInterval	m_endTime m_startTime	TimeInterval(ObjectServer) _close() _initClass(Hashtable) getendTime() setendTime(Time) getstratTime() setstartTime(Time)

ตารางที่ 7.1 แสดง Classes, Properties และ Method ของ Class Temporal ในภาษา Java

Class ที่ Export ออกมา จะเอาไว้ใช้เป็น Interface ในการติดต่อกับ Database โดยทำการดึงค่าด้วย Method get และ Save ค่าด้วย Method set โดยเราจะเขียนเฉพาะส่วนหลักหรือ Business Logic กับ GUI เท่านั้น

ตัวอย่าง การนำ Temporal Class ที่ได้ไป Implement กับ Class Instructor

AddInstructor.java

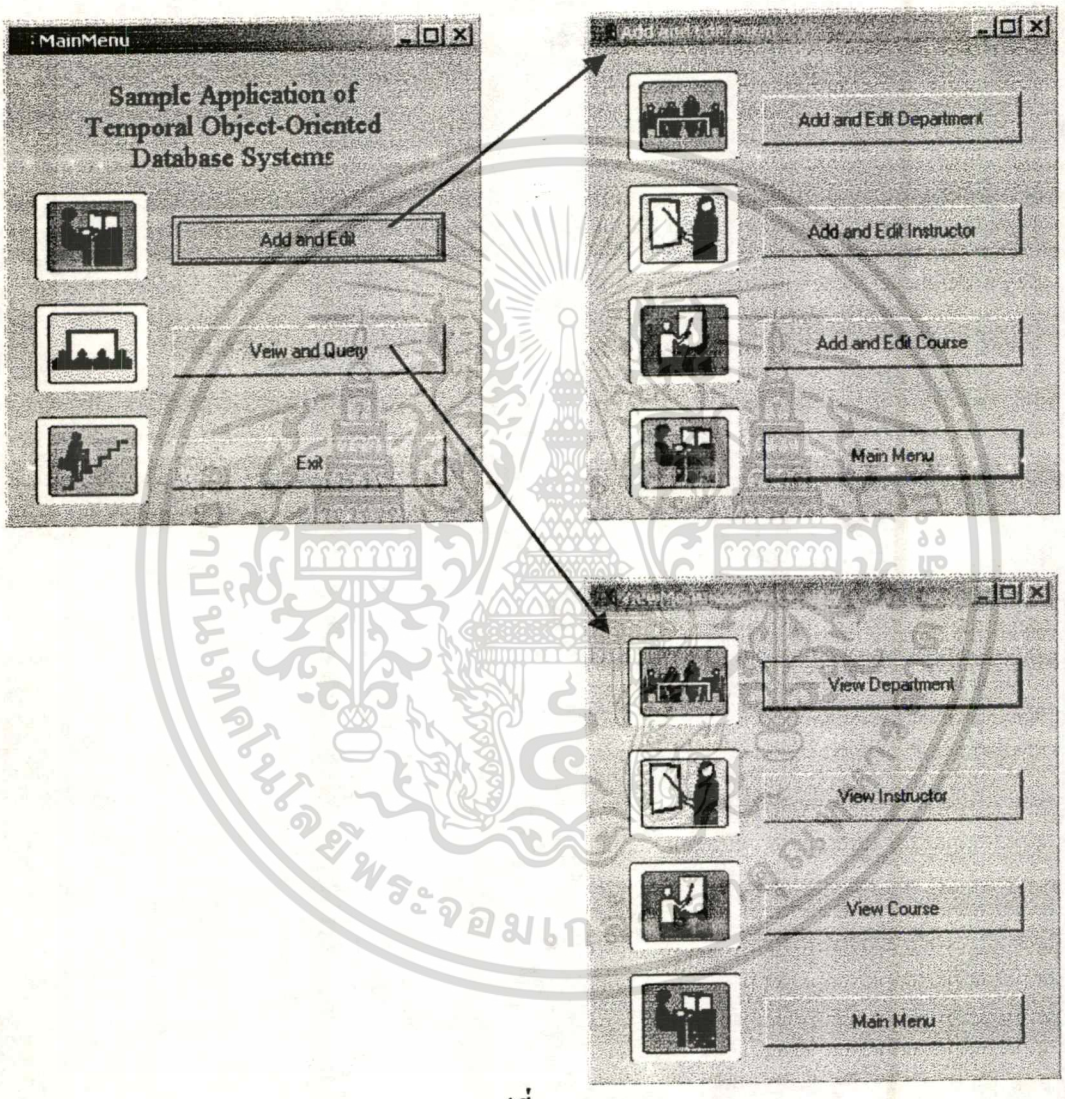
```
import COM.intersys.util.*;
import COM.intersys.objects.*;

public class AddInstructor {
        public static void main( String[] args ) {
                ObjectFactory                   factory = null;
```

```
Instructor                  instructor = null;
Temporal.TemporalValue      latestVer = null, oldVer = null;
Temporal.TimeInterval       timeIntv = null;
ListOfObjects               history = null;

try {
        factory = new ObjectFactory( "cn_iptcp:127.0.0.1[1972]:USER" );
        System.out.println( "Connected." );

        //instructor = new Instructor( factory, new SysList() );
        instructor = new Instructor( factory, "1" );

        instructor.setname("Chavalit Chanchaisirivet");
        instructor.setssn(1111111);

        System.out.println( "Instructor" );
        System.out.println( "Name: " + instructor.getname() );
        System.out.println( "SSN: " + instructor.getssn() );

        Department department = new Department(factory, "1");
        instructor.setdept(department);
        System.out.println( "Department: " + instructor.getdept().getname());
        department._close();

        // ประกาศตัวแปร salary เป็นชนิด Temporal
        Temporal.Temporal salary = instructor.getsalary();
        salary.setsize(1);
        latestVer = new Temporal.TemporalValue(factory);
        latestVer.setversion(2);
        latestVer.setvalue(String.valueOf(30000));
        salary.setlatestVersion(latestVer);

        history = new ListOfObjects(factory);
        oldVer = new Temporal.TemporalValue(factory);
        oldVer.setversion(1);
        oldVer.setvalue(String.valueOf(25000));
        history._insert(oldVer);
        salary.sethistory(history);

        System.out.println("Show Salary" );
        System.out.println("  Size: " + salary.getsize() );
        System.out.println("  Salary:   " + salary.getlatestVersion().getvalue() );
        System.out.println("  LatestVersion: " +
        salary.getlatestVersion().getversion() );
        System.out.println("  HistoryVersion");
        history = salary.gethistory();
        for (int i=1; i <= history._count(); i++) {
                latestVer = (Temporal.TemporalValue)history._getAt(i);
                System.out.println("    Element #" + Integer.toString(i) + " ->
                version:" + latestVer.getversion());
        }

        Temporal.Temporal rank = instructor.getrank();
        rank.setsize(1);
        latestVer = new Temporal.TemporalValue(factory);
        latestVer.setversion(2);
        latestVer.setvalue("rank 5");
        rank.setlatestVersion(latestVer);
        history = new ListOfObjects(factory);
        oldVer = new Temporal.TemporalValue(factory);
```

```
                oldVer.setversion(1);
                oldVer.setvalue("rank 4");
                history._insert(oldVer);
                rank.sethistory(history);

                System.out.println( "Show Rank" );
                System.out.println( "  Size: " + rank.getsize() );
                System.out.println( "  Rank:   " + rank.getlatestVersion().getvalue() );
                System.out.println( "  LatestVersion:   " +
                rank.getlatestVersion().getversion() );
                System.out.println( "  HistoryVersion");
                history = rank.gethistory();
                for (int i=1; i <= history._count(); i++) {
                        latestVer = (Temporal.TemporalValue)history._getAt(i);
                        System.out.println("    Element #" + Integer.toString(i) + " ->
                        version:" + latestVer.getversion());
                }

                instructor._save();
                latestVer._close();
                oldVer._close();
                history._close();
                salary._close();
                rank._close();
                instructor._close();
                factory.close();

        } catch (Exception ex) {
                System.out.println( "Caught exception: " + ex.getClass().getName()
                                + ": " + ex.getMessage() );
        }
    }
}
```

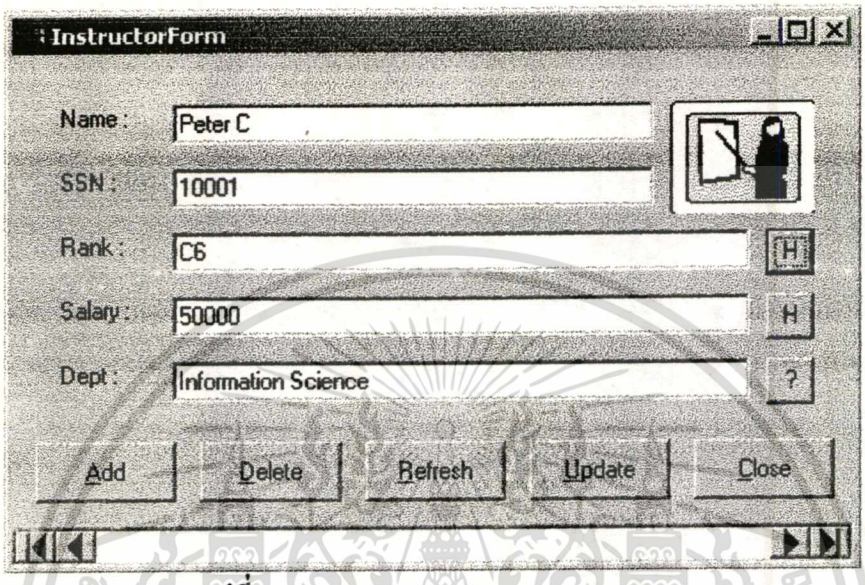
7.2 ตัวอย่างหน้าจอโปรแกรม

Menu

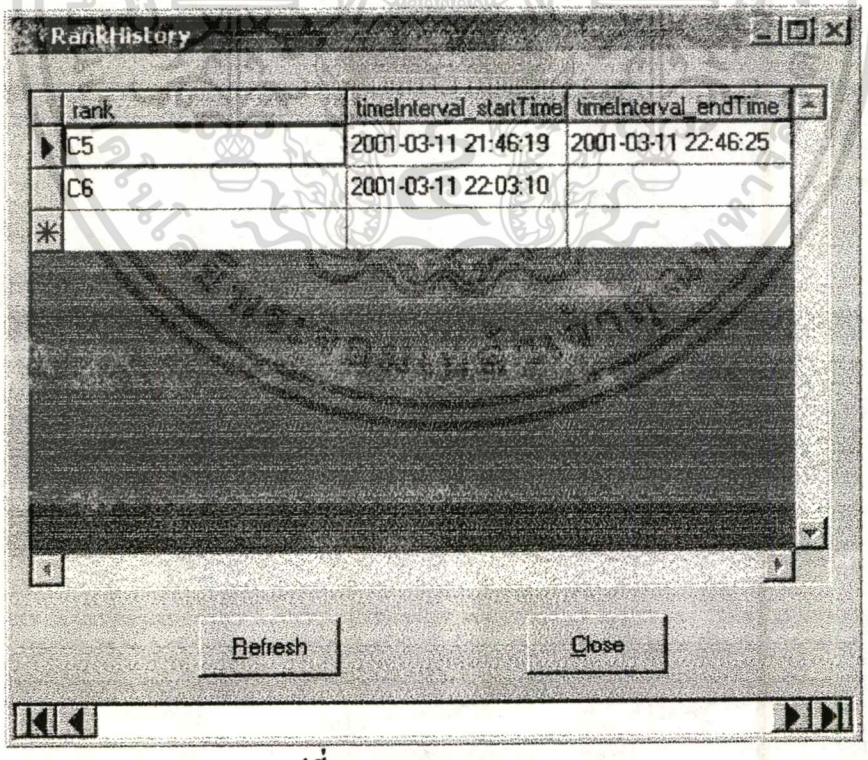


รูปที่ 7.1 Menu

Add and Edit

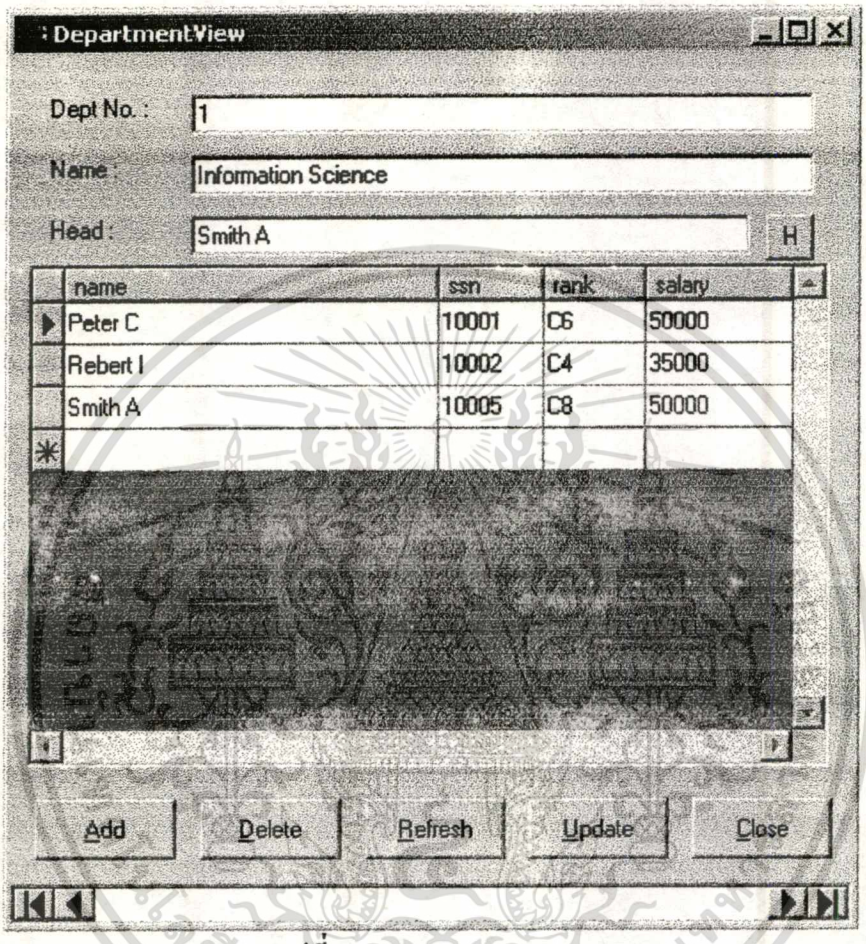


รูปที่ 7.2 Add and Edit Instructor From



รูปที่ 7.3 List Rank History

View and Query



รูปที่ 7.4 Department View

7.3 ปัญหาที่พบ

- สำหรับโปรแกรมประยุกต์ที่พัฒนาด้วย Java บนระบบฐานข้อมูล CACHE นั้นยังมีปัญหาทั้ง Java และ CACHE ยังไม่สนับสนุน การประกาศชนิดข้อมูลเป็น Parameterized type ทำให้ไม่สามารถกำหนด Property ของข้อมูลให้เป็น Temporal ของ ชนิดข้อมูล Object ได้โดยตรง แต่สามารถกำหนดเป็น Temporal ของชนิดข้อมูลพื้นฐานเช่น Integer, String ได้ ทำให้เกิดความยุ่งยากมากขึ้นในการ Implement
- CACHE ยังไม่สนับสนุนการ Query ข้อมูลที่มี Data Type เป็น List (collection) ซึ่งเป็น data type หนึ่งของ Object-Oriented ซึ่งในโปรเจคนี้คือ Property History ทำให้ไม่สามารถ select ค่าขึ้นมาดูได้
- CACHE สนับสนุนการ Query Embedded Object แค่ชั้นเดียว ไม่สามารถ Query ข้อมูลที่เป็น Embedded หลายๆ ชั้นได้เช่น

```
select i.name, i.rank.latest_version.value
from i in Instructors,
        c in i.teaches.latest_version.value
where c.offered_by.latest_version.value.name = 'CSE'
```

ตรงนี้ค่า value เป็น Embedded Object ใน latest_version และ latest_version เป็น Embedded Object ใน rank ซึ่ง CACHE ยังไม่สามารถ Query ได้

7.4 แนวทางในการแก้ปัญหา

(สำหรับปัญหาที่ CACHE ไม่มีการใช้งาน Parameterize Type)

- กำหนด Data Type T เป็น String และใช้วิธี Serialize object เพื่อแปลงทุกๆ Data Type เป็น String ก่อน Save ค่าลง Database และเช่นกัน ในขั้นตอนการดึงค่าจาก Database ก็ทำการแปลงค่า String กลับมาเป็นค่านั้นๆ เพื่อนำไปใช้งานต่อ
- สร้าง Class Temporal ตามชนิดตัวแปรที่ใช้งาน เช่น TemporalString, TemporalInteger, TemporalInstructor,...
- ใช้ List<T> แทน Temporal<T> เช่นแทนที่เราจะกำหนดค่า

  Temporal<Instructor> head; ก็ใช้เป็น

  List<Instructor> head; และ List<timeInterval> headTime;

บทที่ 8
บทสรุปและผลการดำเนินงาน

8.1 บทสรุป

การนำระบบฐานข้อมูลเชิงวัตถุและระบบฐานข้อมูลอิงเวลามารวมกันสามารถแก้ปัญหาการสืบค้นข้อมูลในอดีตได้ และยังมีความซับซ้อนน้อยกว่า การใช้ระบบฐานข้อมูลอิงเวลาเพียงอย่างเดียว โดยระบบฐานข้อมูลที่รวมคุณสมบัติทั้ง 2 อย่างนี้เรียกว่า ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลา (Temporal Object-Oriented Database) ซึ่งมีหลักการทำงาน เหมือนกับระบบฐานข้อมูลเชิงวัตถุ เพียงแต่ว่าจะมี Data Type พิเศษขึ้นมาอีกชนิดหนึ่ง คือ Temporal ซึ่งเอาไว้ใช้สำหรับ ครอบ Fields (Properties) ของข้อมูลที่เราต้องการให้เก็บบันทึก ทำให้ได้ทั้งค่าของข้อมูลและค่าเวลา ณ. เวลานั้นๆ การสืบค้นเราก็ทำได้โดยระบุเวลาหรือช่วงเวลาไปกับฟิลด์ที่เราต้องการค้นหา

8.2 แนวทางในการพัฒนาในอนาคต

ระบบฐานข้อมูล CACHE นั้นสนับสนุนการทำงานของฐานข้อมูลเชิงวัตถุก็จริง แต่ว่ายังไม่สนับสนุนครบทุกอย่าง จึงทำให้บาง Feature ที่เราต้องการยังไม่สามารถทำได้ เช่น การกำหนด Data type ของชนิดข้อมูลเป็น Parameterized type (Template) ก็ยังทำไม่ได้ ซึ่งทำให้เราไม่สามารถกำหนด Property ของข้อมูลให้เป็นชนิด Temporal รวมกับ Data type ชนิดอื่นได้ เช่น Temporal<String>, Temporal<Instructor> ทำให้เราต้องเลี่ยงไปใช้วิธีอื่นซึ่งยุ่งยากซับซ้อนและช้ากว่า เช่นการทำ Multiple Inheritance, Object Serialize, หรือการใช้ List<> เพื่อให้ได้ Temporal Type ซึ่งถ้าในอนาคตเหล่า Object-Oriented Database นั้นสนับสนุนการสร้างคลาสแบบ Parameterized type ก็จะทำให้ระบบฐานข้อมูลเชิงวัตถุชนิดอิงเวลานั้นสมบูรณ์แบบมากขึ้น

บรรณานุกรม

Bertino E. and Martino L. 1993. Object-Oriented Database Systems Concepts and Architectures. : Addison-Wesley.

Bertino E., Ferrari E., Guerrini G., and Merlo I. 1998. "Extending the ODMG Object Model with Time." 41-66. In E. Jul, editor, Proceedings of the Twelfth European Conference on Object-Oriented Programming (ECOOP). Brussels, Belgium : Springer Verlag.

Elmasri R. and Fegaras L. 1998. "A Temporal Object Query Language." In Proceedings of the Fifth International Workshop on Temporal Representation and Reasoning, Sanibel Island, Florida.

Elmasri R., Kouramajian V., and Fernando S. 1993. "Temporal Database Moding: An Object-Oriented Approach." 574-585. In Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM), Washington, DC.

R.G.G Cattell. 1996. The Object Database Standard. ODMG-93, Release 1.2 : Morgan Kaufmann Publishers.

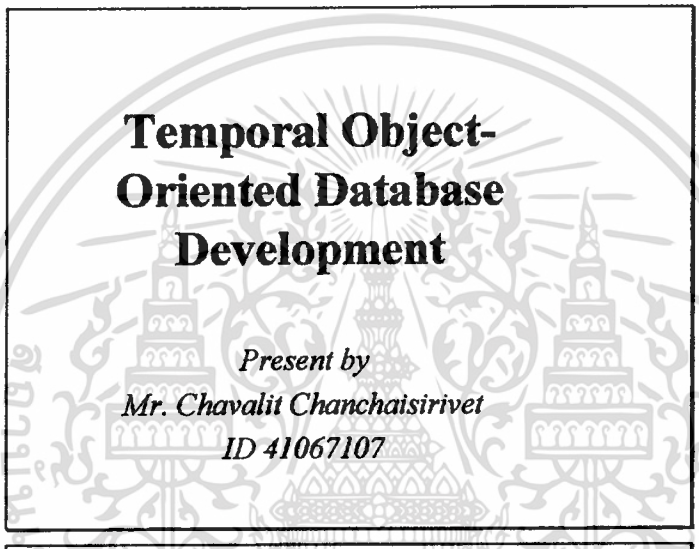
Snodgrass R. 1995. "Temporal Object-Oriented Databases: A Critical Comparison." 238-408. In W. Kim, editor, Modern Database Systems: The Object Model, Interoperability, and Beyond : Addison-Wesley.

ภาคผนวก

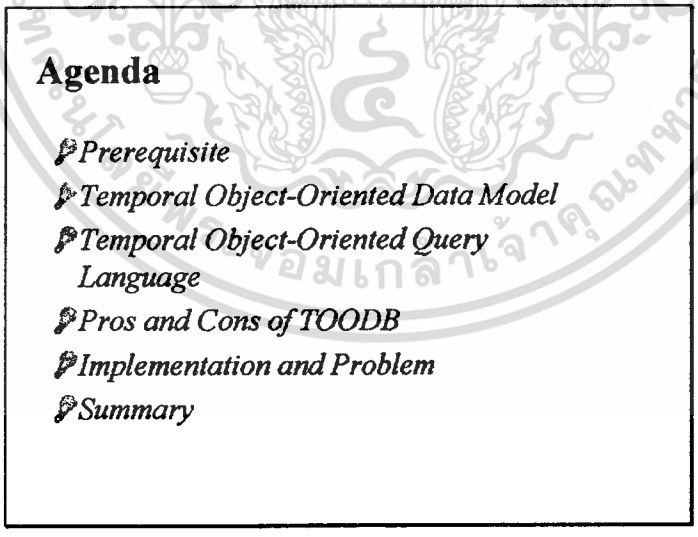
ภาคผนวก ก

Temporal Object-Oriented Database Development Present Slide

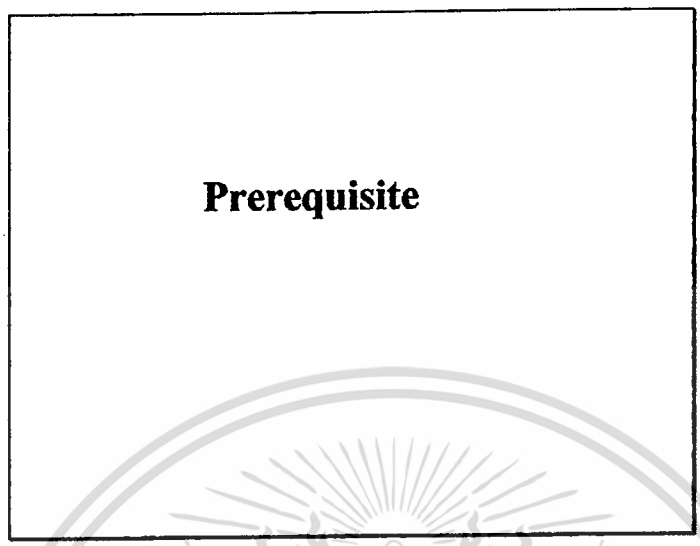
Slide 1



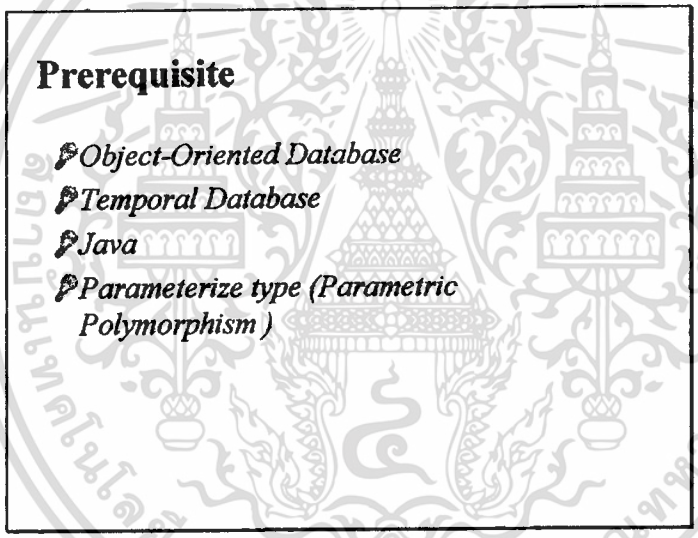
Slide 2



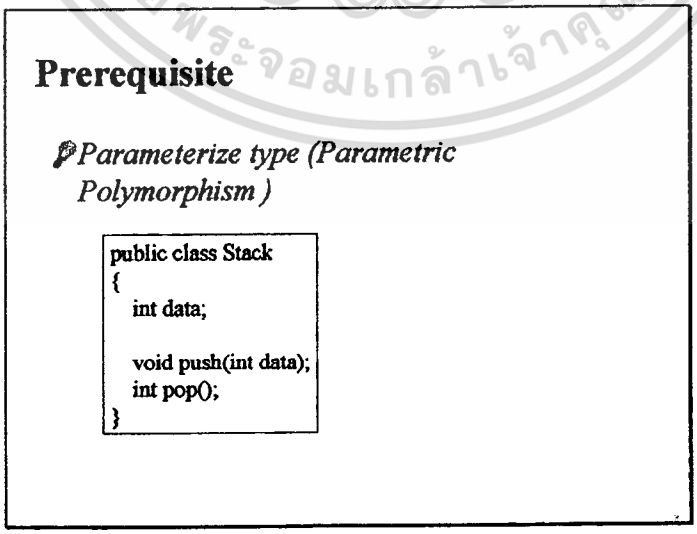
Slide 3



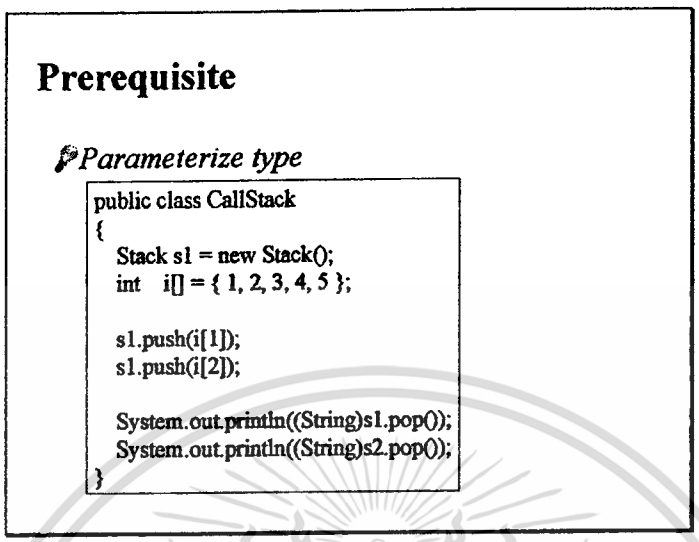
Slide 4



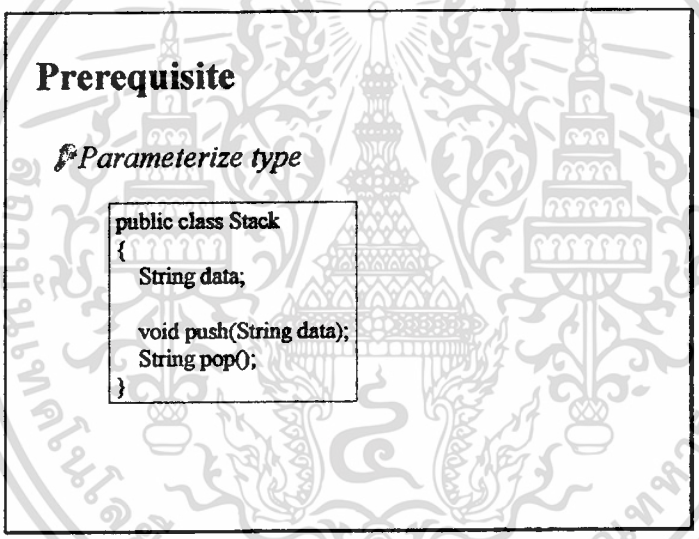
Slide 5



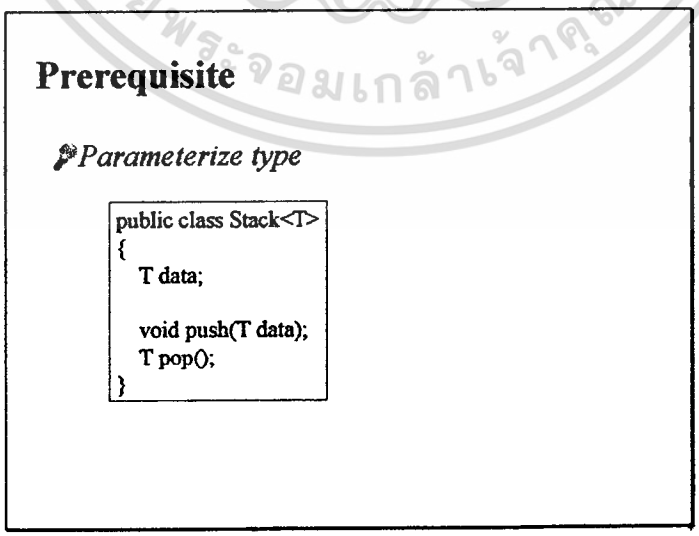
Slide 6



Slide 7



Slide 8



Slide 9

Prerequisite

- *Parameterize type*

```
public class CallStack
{
   Stack<int>  s1 = new Stack<int>();
   int         i[] = { 1, 2, 3, 4, 5 };
   Stack<String> s2 = new Stack<String>();
   String      str[] = { "one", "two", "tree" }

   s1.push(i[1]);
   System.out.println((String)s1.pop());
   s2.push(str[1]);
   System.out.println(s2.pop());
}
```

Slide 10

Temporal Object-Oriented Data Model

Slide 11

Temporal Object-Oriented Data Model

- *Data structure และ Data model เหมือนกับ Object-oriented database*
- *Temporal เป็นส่วนขยายความสามารถของ OODB ให้เพิ่มคุณสมบัติของ Temporal data*
- *เพิ่ม class Temporal<T> (เพื่อเป็น data type ในการระบุ properties ที่ต้องการให้เป็น temporal)*

Slide 12

Temporal Object-Oriented Data Model

- *Temporal Properties*
  - *Syntax : Temporal<T> propertity*

```
public class Instructor
{
   int ssn;
   String name;
   Temporal<float> salary;
   Temporal<String> rank;;
}
```

```
public class Department
{
   int dno;
   String name;
   Temporal<Instructor> head;
}
```

Slide 13

Temporal Object-Oriented Data Model

- *Temporal Extension on OODB*

```
public class Department
{
   int dno;
   String name;
   Temporal<Instructor> head;
   transient Set<Instructor> instructors;
   transient Set<Course> coursesOffered;
}
```

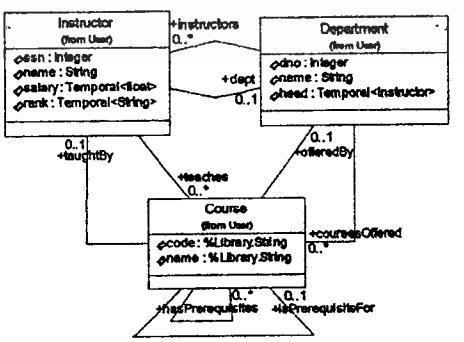
```
public class Instructor
{
   int ssn;
   String name;
   Temporal<float> salary;
   Temporal<String> rank;
   transient Department dept;
   transient Set<Course> teaches;
}
```

```
public class Course
{
   String code;
   String name;
   transient Department offeredBy;
   transient Instructor taughtBy;
   transient Set<Course> isPrerequisiteFor;
   transient set<Course> hasPrerequisites;
}
```

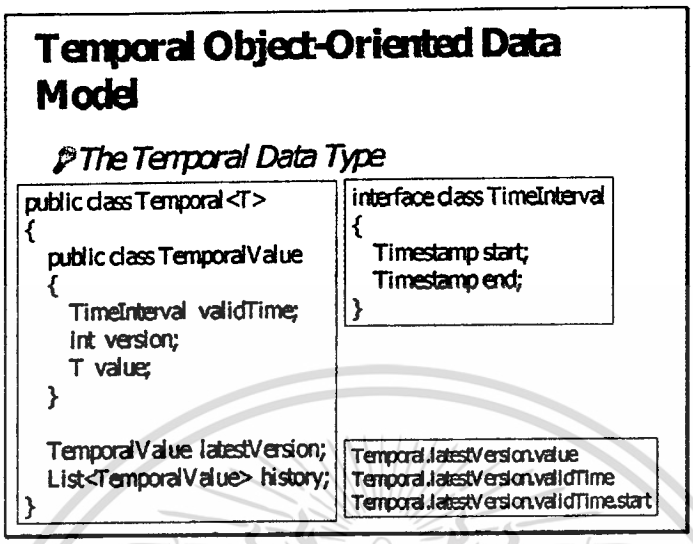
Slide 14

Temporal Object-Oriented Data Model

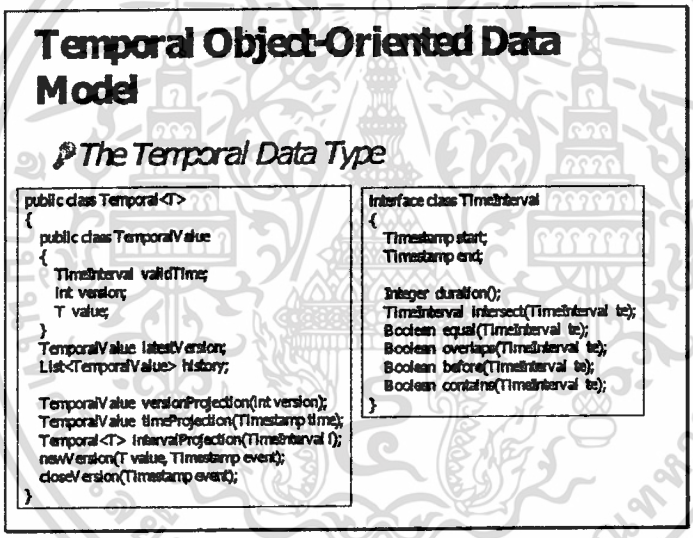
- *Temporal Extension on OODB*



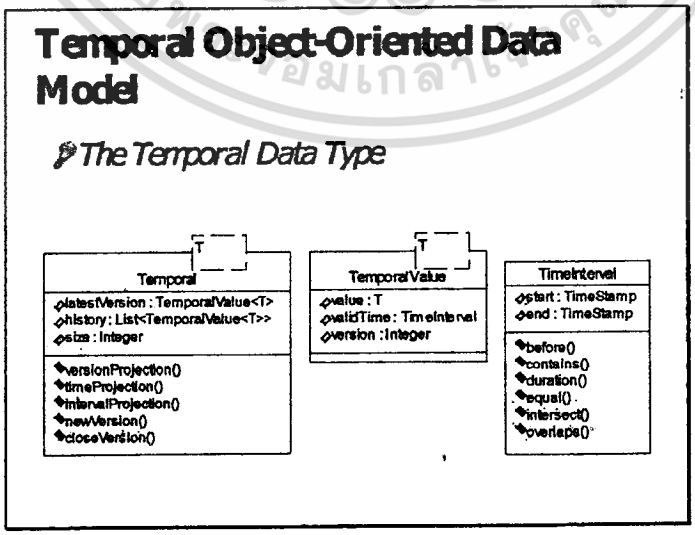
Slide 15



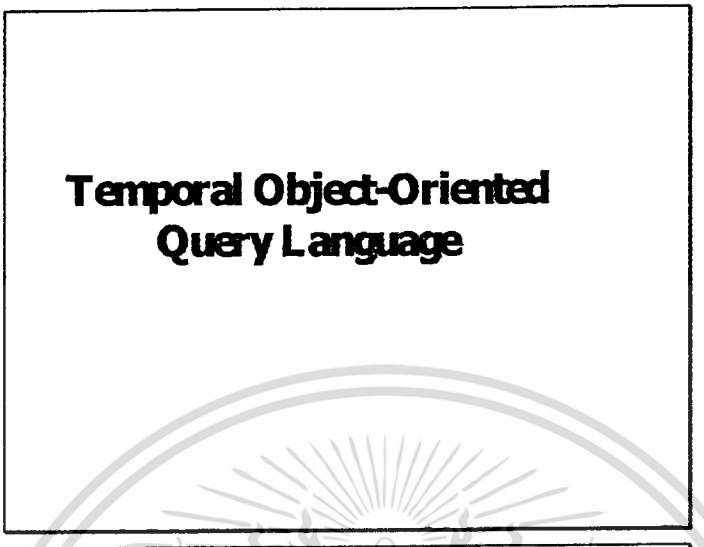
Slide 16



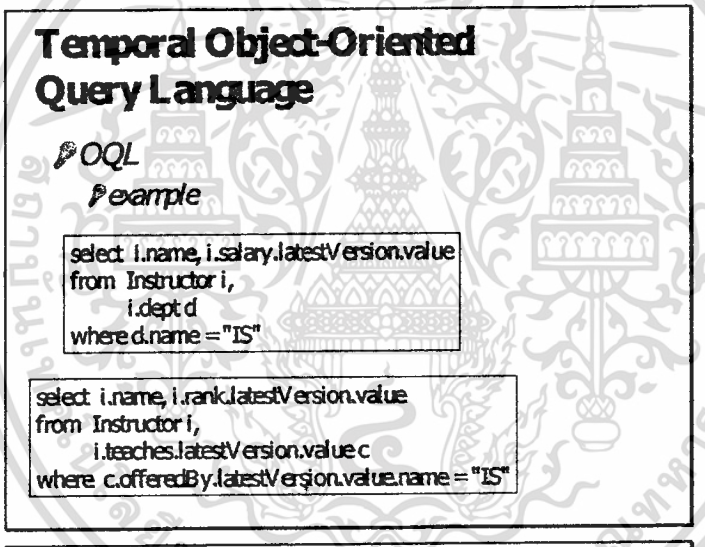
Slide 17



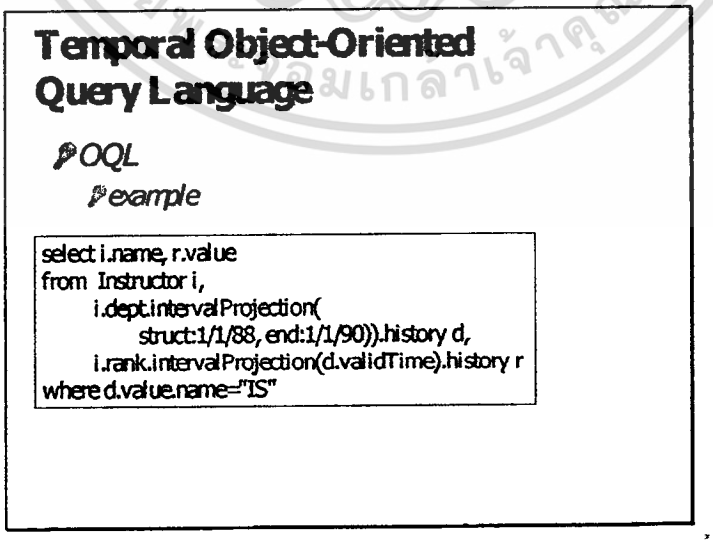
Slide 18



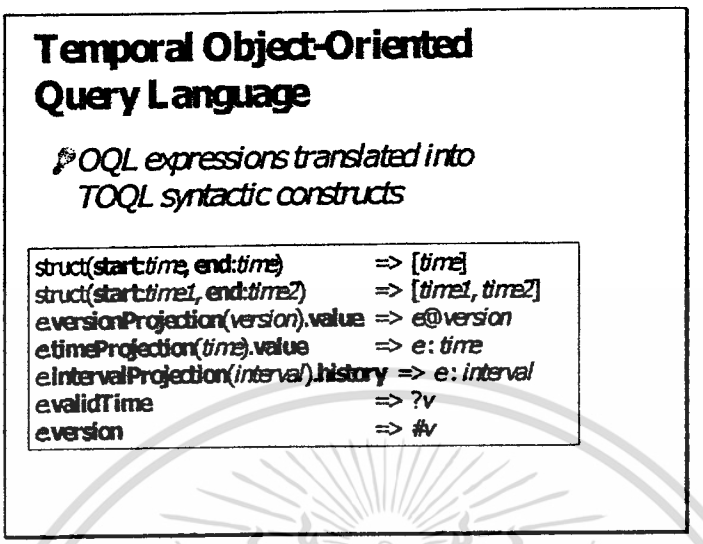
Slide 19



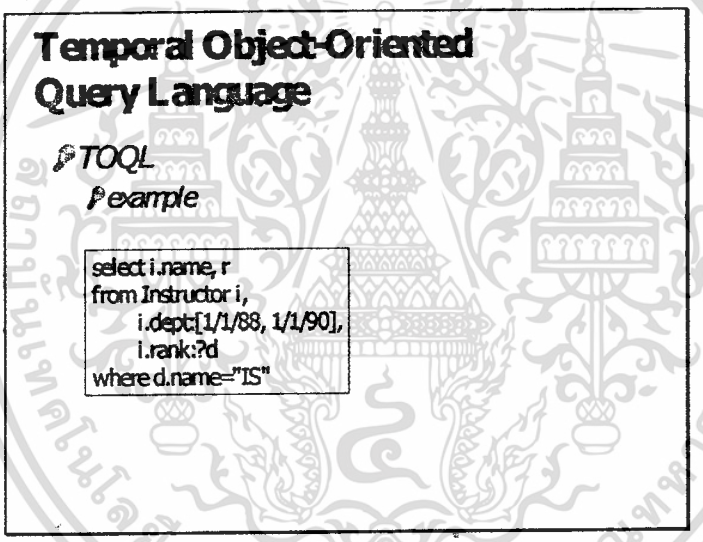
Slide 20



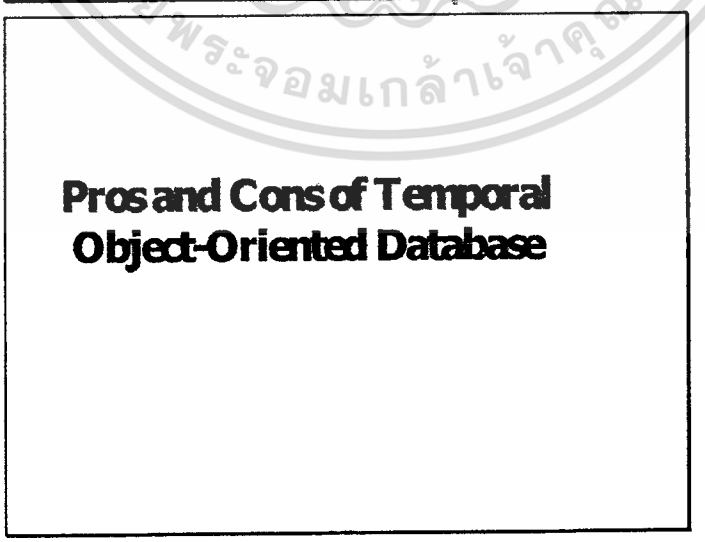
Slide 21



Slide 22



Slide 23



Slide 24

Pros and Cons of TOODB

- Temporal Extension ของ Relational DB ไม่สามารถใช้ attribute timestamping ได้ เพราะจะเข้าไปกระทบกับ data structure ของ relational database
- Temporal Extension ของ Object-oriented DB ใช้ attribute timestamping ได้

Slide 25

Pros and Cons of TOODB

- Temporal extension on relational DB

Tuple timestamping

Smith	c5	35k	9/97 - 9/98
Smith	c6	48k	9/98 - now
Brown	c5	32k	8/96 - 6/97

Attribute timestamping

Smith	c6 (9/97 – now)	35K (9/97 – 9/98)
		48K (9/98 – now)
Brown	c4 (8/96 – 6/97)	32K (8/96 – 6/97)
	c5 (6/97 – now)	40K (6/97 – 1/99)
		45K (1/99 – now)
Jane	c8 (1/98 – now)	80K (1/98 – now)

Slide 26

Pros and Cons of TOODB

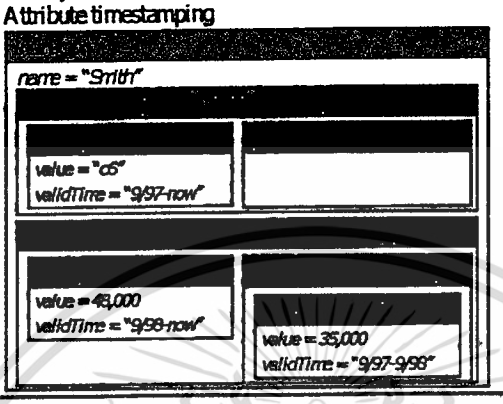
- วิธีแก้ปัญหาของ tuple timestamping ใน R Relational DB คือ แตก table ออกเป็น table ย่อยๆ
- แต่เกิดปัญหาตามมาคือ ช้า เพราะต้อง join และเขียน SQL ยากขึ้น

Slide 27

Pros and Cons of TOODB

- Temporal extension on OODB

Attribute timestamping



Slide 28

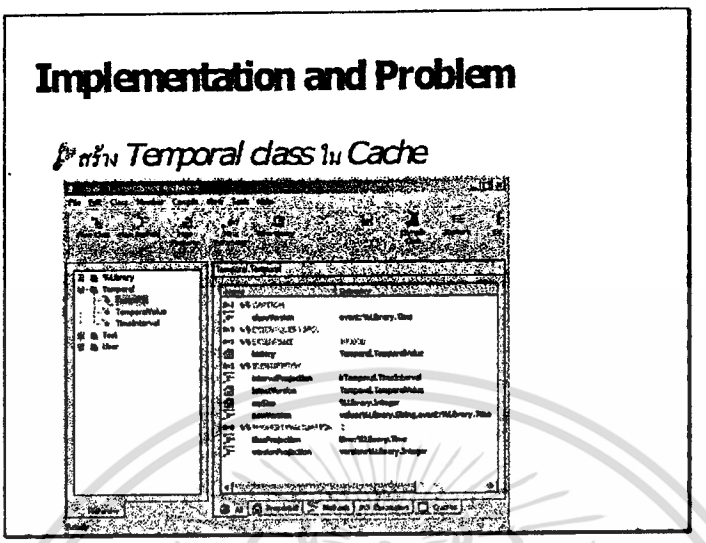
Implementation and Problem

Slide 29

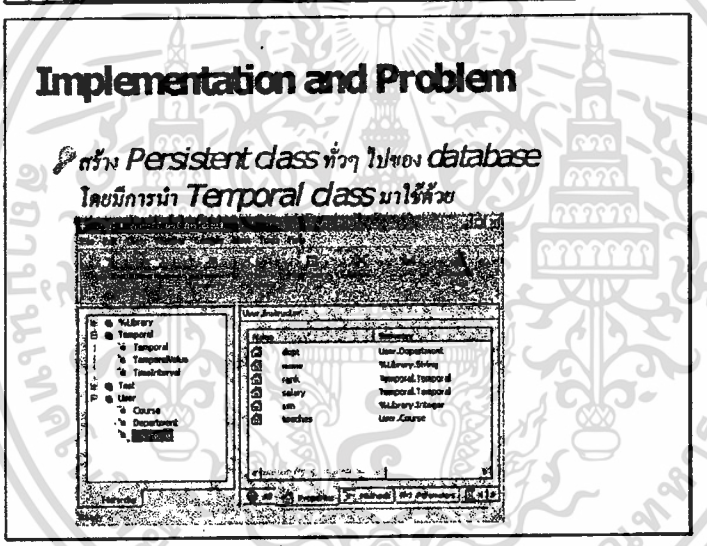
Implementation and Problem

- Implementation
  - สร้าง Temporal class ใน Cache
  - สร้าง Persistent class ทั่วๆ ไปของ database โดยมีการนำ Temporal class มาใช้ด้วย
  - เขียนโปรแกรมด้วยภาษา Java ติดต่อกับ Cache
  - สร้าง User Interface

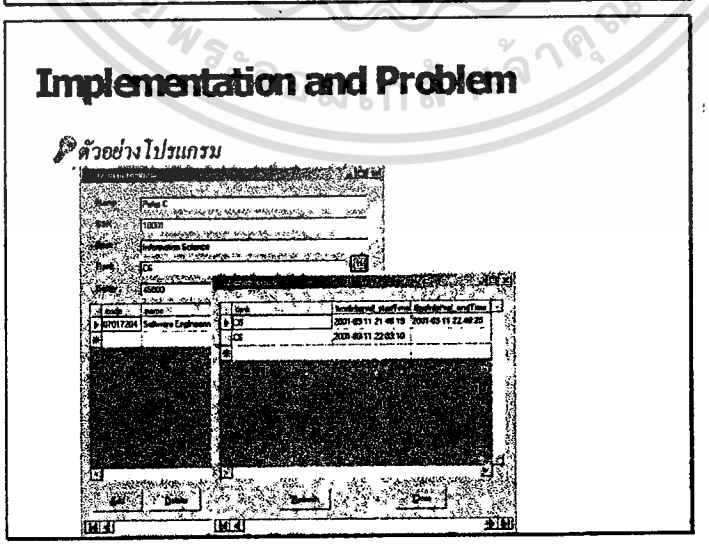
Slide 30



Slide 31



Slide 32



Slide 33

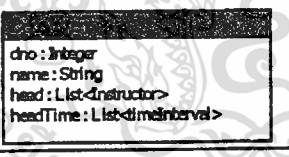
Implementation and Problem

- Problem
  - Cache ไม่ support parameterize type
  - Cache ไม่สามารถเก็บ data type ที่ไม่แน่นอนได้
  - Relationship ใน Cache ยังมีแค่ one to many
  - Cache ไม่สามารถ query สิ่งที่เป็น object ได้ เช่น
    - embedded object มากกว่า 1 ชั้น
    - property ที่เป็น list (collection)

Slide 34

Implementation and Problem

- Solution
  - กำหนด data type T เป็น String และใช้วิธี Serialize object เพื่อแปลงทุกๆ data type เป็น String
  - สร้าง class Temporal ตามชนิดตัวแปรที่ใช้งาน เช่น TemporalString, TemporalInteger, TemporalInstructor,...
  - ใช้ List<T> แทน Temporal<T>



Slide 35

Summary

Slide 36

Summary

- Cache ยังไม่สามารถนำมา Implement Temporal object-oriented database ตามที่ได้ออกแบบไว้ได้ ต้องใช้วิธีอื่นแก้ปัญหาไปก่อน
- ปัญหาหลักคือตัว Cache ยังไม่สนับสนุน หลายๆ feature ของ Object-oriented

ประวัติผู้เขียน

ชื่อผู้เขียน	นาย ชวลิต ฉันชัยสิริเวทย์
วันเดือนปีเกิด	11 มีนาคม 2519
สถานที่เกิด	จังหวัดกรุงเทพมหานคร
วุฒิการศึกษาระดับปริญญาตรี	วท.บ. (คณิตศาสตร์ประยุกต์)
สถานที่สำเร็จการศึกษา	คณะวิทยาศาสตร์
	สถาบันเทคโนโลยีพระจอมเกล้าฯ ลาดกระบัง
ปีที่สำเร็จการศึกษา	ปีการศึกษา 2540
ประสบการณ์การทำงาน	2540 Product Support Engineer & Programmer
	บริษัท CDG Systems Ltd.
	2543 Developer บริษัท Livebrand (Thailand) Ltd.
อาชีพปัจจุบัน	Developer บริษัท Livebrand (Thailand) Ltd.