การจัดการระบบสารสนเทศเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์

Managing Temporal Information on Object Relational Database

นายธีรวุธ สันติสุธีรวุฒิ
นายประยงค์ ทานนท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

การจัดการระบบสารสนเทศเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์

Managing Temporal Information on Object Relational Database

โดย

นายธีรวุธ สันติสุธีรวุฒิ

นายประยงค์ ทานนท์

อาจารย์ที่ปรึกษา

อาจารย์ บัณฑิต พัสยา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การจัดการระบบสารสนเทศเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์

Managing Temporal Information on Object Relational Database

ผู้จัดทำ

1. นายธีรวุธ สันติสุธีรวุฒิ รหัสประจำตัว 41013533
2. นายประยงค์ ทานนท์ รหัสประจำตัว 41013539

อาจารย์ที่ปรึกษา

(อ. บัณฑิต พัสยา)

การจัดการระบบสารสนเทศเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์

นายธีรวุธ สันติสุธีรวุฒิ 41013533
นายประยงค์ ทานนท์ 41013539
อ. บัณฑิต พัสยา อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในระบบฐานข้อมูลเชิงสัมพันธ์ การเก็บข้อมูลจะเก็บเฉพาะข้อมูลที่สถานะ (State) ล่าสุดเท่านั้น เมื่อมีการแก้ไขเปลี่ยนแปลงข้อมูล ข้อมูลที่มีการจัดเก็บอยู่ในฐานข้อมูลจะถูกทำการลบและหายไป ทำให้ไม่มีการจัดเก็บค่าข้อมูลเก่าไว้เลย แต่ในความเป็นจริงแล้ว โปรแกรมประยุกต์ส่วนใหญ่มักจะมีการเรียกค้นค่าข้อมูลหลาย ๆ สถานะ เช่น ระบบเก็บประวัตินักศึกษา ระบบเก็บประวัติคนไข้ เป็นต้น ซึ่งระบบฐานข้อมูลที่มีอยู่ไม่สามารถตอบสนองความต้องการนี้ได้

ฐานข้อมูลเชิงเวลา เป็นระบบฐานข้อมูลที่สามารถจัดเก็บและเรียกค้นข้อมูลเชิงเวลาได้โดยระบบฐานข้อมูลนี้จะมีการจัดการเวลาต่าง ๆ ของข้อมูล ซึ่งก่อนหน้านี้ได้มีผู้ทำการวิจัยเกี่ยวกับฐานข้อมูลเชิงเวลามาแล้ว ซึ่งได้นำเวลาไปแปะไว้ที่ (Tuple) ของตารางเท่านั้น แต่เราจะไม่ทราบเลยว่าคอลัมน์ไหนได้เปลี่ยนไปบ้าง จึงเป็นที่มาของงานวิจัยนี้ที่ได้นำเวลามาแปะไว้ที่คอลัมน์โดยใช้ รหัสเฉพาะ Object Identification (Oid) ซึ่งมีอยู่ในฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object Relational Database) เข้ามาช่วยแก้ปัญหาคีย์ของข้อมูลในตารางที่มีการเปลี่ยนแปลง

ผู้วิจัยได้ทำการสร้างโปรแกรมประยุกต์เชิงเวลาที่มีการแปะเวลาไว้ที่คอลัมน์ โดยได้ใช้ข้อมูลชนิด Serial ซึ่งมีอยู่ใน Informix เพราะมีคุณลักษณะที่คล้ายกับ Oid เข้ามาช่วยในการแก้ปัญหาครั้งนี้ และได้นำส่วนของภาษา Stored Procedure Language (SPL) เข้ามาประมวลผลเพื่อจัดการกับข้อมูลเชิงเวลา เพราะว่าระบบการจัดการฐานข้อมูลที่ใช้อยู่ในปัจจุบันยังไม่ได้มีส่วนของการจัดการข้อมูลเชิงเวลาอย่างเต็มที่

Managing Temporal Information on Object Relational Database

Teerawut Suntisuteerawut

Prayong Tanon

Bundit Patsaya Advisor

Abstract

In General, Relational Database System will keep only the latest state of data. Whenever data is kept in database has been changed all converted to a new state, the old one will be deleted and completely disappeared. As such, and old data will be lost and hard to be regained. In fact most application programs, used in nowadays, often query several state of data, such as, students all patients ' Registered information system of which and existing database system can not support the excessive data effectively.

Temporal Database is a database system that is keep capable to keep an query temporal data more efficiently by managing and reprocessing data 's time which previously used to be researched by some one by time stamping at the tuple only but with this way, we can not know which column of time has been changed. That is the point where we start to research an outcome by stamping time at object identification (OID) which is already settled in the object relational database in order to solve problem of the key in tuple which state has been changed.

Therefor, we have created new temporal application program which stamps time at column of tuple by using serial data type which is in Informix because it is similar to OID so as to help solve mentioned problem and by using Stored Procedure Language (SPL) to execute for manage temporal data because at present, Database Management System (DBMS) can still not support temporal data fully dependency.

กิตติกรรมประกาศ

ปริญญานิพนธ์เล่มนี้คงไม่สำเร็จลุล่วงไปด้วยดีแน่ ถ้าหากไม่ได้รับความช่วยเหลือและร่วมมือจากบุคคลหลาย ๆ ฝ่ายด้วยกัน ซึ่งอันดับแรกก็คือท่านอาจารย์ บัณฑิต พัสยา อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้คำแนะนำ เอาใจใส่และคอยช่วยเหลือเป็นอย่างดีมาโดยตลอดการทำโปรเจ็กต์ ซึ่งผู้จัดทำรู้สึกมีความภาคภูมิใจเป็นอย่างมาก ที่ทำให้เราได้โอกาสที่ดีในการเข้ามาทำ TORDB ต้องขอขอบพระคุณในความกรุณาเป็นอย่างยิ่ง

ขอขอบคุณบริษัท Informix Thailand ที่เอื้อเฟื้อให้เอกสารและเครื่องมือต่าง ๆ เช่น Document Online, Data Director Object และ INFORMIX 2000 ในการทำวิจัยจนสำเร็จลุล่วงไปด้วยดี ขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้สนับสนุนสถานที่และทรัพยากรต่าง ๆ ในการทำงานวิจัยนี้ทั้งคำปรึกษาคำแนะนำที่ดีจากอาจารย์ทุก ๆ ท่านด้วยโดยเฉพาะ อ. ADEK ขอขอบคุณเพื่อน ๆ ในห้อง P (โดยเฉพาะที่อยู่ห้อง NETWORK) ที่รักและคอยกวนใจในทุก ๆ โอกาส และ น้อง ๆ ห้อง D (ที่อยู่ OLALA) ที่คอยกระตุ้นคอยถามถึงโปรเจ็กต์ที่ทำถึงไหนแล้ว ขอบคุณจริง ๆ ขอบคุณคณาจารย์ในพระจอมเกล้าลาดกระบังทุกท่านที่ประสิทธิ์ประสาทวิชาความรู้มาให้แก่เรา และท้ายที่สุดที่มิอาจจะลืมได้ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ทำให้เรามีวันนี้ได้

สุดท้ายขอขอบคุณสิ่งศักสิทธิ์ทั้งหลายในสากลโลกที่คลบันดาลให้ข้าพเจ้าเกิดมาเป็นลูกของท่าน ก็คือ บิดา มารดา อันที่เคารพรักยิ่ง ซึ่งท่านได้คอยสนับสนุน อบรมเลี้ยงดู ให้กำลังใจพร้อมทั้งให้โอกาสและการศึกษาที่ดีแก่ข้าพเจ้ามาอย่างเต็มที่ ข้าพเจ้าขอระลึกในพระคุณอันยิ่งใหญ่นี้ และกราบขอบพระคุณมา ณ ที่นี้ด้วย

ธีรวุธ สันติสุธีรวุฒิ
ประยงค์ ทานนท์

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

หน้า

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันงานการพัฒนาทางด้านการจัดการฐานข้อมูลนั้นมีความก้าวหน้ามาก ซึ่งมีนักวิจัยจำนวนมากได้ทำการวิจัยและนำเสนอโมเดลต่าง ๆ หลากหลายวิธีเพื่อนำมาตอบสนองการจัดเก็บข้อมูลหลากหลายชนิดต่าง ๆ กัน เดิมทีเดียวฐานข้อมูลที่มีการใช้อยู่ในปัจจุบันส่วนใหญ่มักจะถูกออกแบบมาใช้ในการตอบคำถามกับข้อมูล State เดียวเท่านั้น เมื่อข้อมูลมีการเปลี่ยนแปลงก็จะทำการเขียนทับข้อมูลใหม่ลงในข้อมูลเดิม จึงทำให้ไม่สามารถเรียกค้นข้อมูลใน State เก่า ๆ ได้ ซึ่งในบางคำถามอาจทำให้ได้คำตอบที่ไม่ตรงกับข้อมูลจริงออกมาเพราะว่าข้อมูลใน State เก่าได้ถูกเขียนทับไปแล้ว จึงได้มีการนำเสนอ Temporal Data Model มาทำการแก้ปัญหาเหล่านั้นซึ่ง Temporal Data Model ก็เป็นโมเดลที่มีการสนับสนุนการตอบคำถามของข้อมูลในอดีตและปัจจุบัน โดยโมเดลนี้จะมีการนำส่วนของเวลาเข้าไปแปะไว้ที่ข้อมูล

ฐานข้อมูลเชิงเวลา (Temporal Database) ก็เป็นหนึ่งในการวิจัยและพัฒนาโดยมีหลักการแนวคิดที่จะนำเอาเวลาเข้ามาเกี่ยวข้องในการจัดเก็บลงไปในฐานข้อมูล และทฤษฎีการจัดเก็บข้อมูลแบบเอาเวลาเข้ามาจัดเก็บนั้นได้มีคนทำวิจัยไว้แล้วแต่ได้ทำเพียงแค่ส่วนของการแปะเวลา (Time Stamping) ไว้ที่ Tuples ของตารางเท่านั้นซึ่งยังอาจจะไม่เพียงพอกับการตอบคำถามได้อย่างละเอียด เช่น ถ้ามีการเปลี่ยนแปลงข้อมูลภายใน Row นั้น จะไม่ทราบเลยว่าคอลัมน์ (Column) ไหนได้มีเปลี่ยนแปลงโดยในการแปะเวลาไว้ที่ Tuple นั้นจะทราบแต่เพียงว่า Row นั้นได้มีการเปลี่ยนแปลงแล้วจึงทำการแปะเวลาเข้าไปที่ Tuple ของตารางนั้น และทำการสร้าง Row ใหม่เพื่อนใส่ Fact ของเวลาใหม่ที่เป็นจริงเข้าไปใน Tuple ทำให้เกิด Row ที่เป็น Fact ใหม่ของข้อมูลในตารางอย่างกระจัดกระจาย และยังพบความยุ่งยากในการตอบคำถามของการแปะเวลาไว้ที่ Tuple คือจะต้องทำการ Split ตารางและ Join ตารางต่อมาทำให้เกิดปัญหาในเรื่องของ Overhead ในการทำการ Split และ Join ตารางในงานวิจัยนี้จึงได้มีการนำเสนอการแปะเวลา ไว้ในทุก ๆ คอลัมน์ที่จำเป็นในการตอบคำถามเพื่อให้ทราบว่าคอลัมน์ไหนเปลี่ยนไปบ้างและ Fact ใหม่ที่เกิดขึ้นจะถูกเก็บอยู่ใน Tuple เดียวกันหมดอย่างเป็นระบบ และการแปะเวลาไว้ที่ คอลัมน์นั้นเราได้นำแนวคิดของออบเจ็กต์ (Object) มาใช้ในการจัดเก็บข้อมูลโดยทำเป็น ฐานข้อมูลเชิงวัตถุสัมพันธ์ที่ขึ้นกับเวลา (Temporal Object Relational Database) ซึ่งสามารถเก็บค่าของข้อมูลได้หลายค่า (Multiple Value) เช่น ลิสต์ (List) หรือ เซต (Set) อยู่ภายในคอลัมน์ เดียวกันใน Tuple ของตารางนั้นได้ โดยได้มีการขยายความสามารถเพิ่มขึ้นตามคุณสมบัติของ Object Oriented Concept คือมีการ Inheritance Encapsulation Polymorphism และเราจะนำหลักการของค่ารหัสเฉพาะ (Object Identification : Oid) เข้ามาใช้เพื่อช่วยในการชี้บอกว่าข้อมูลในแต่ละ Tuple ที่เหมือนกันเป็นข้อมูลคนละตัวกัน

1.1 ความสำคัญและที่มา

เนื่องด้วยการจัดเก็บข้อมูลลงไปในฐานข้อมูลปัจจุบันส่วนใหญ่นั้น ถูกออกแบบมาเพื่อเก็บข้อมูลสถานะ (State) ล่าสุด เมื่อมีการเปลี่ยนแปลงข้อมูล ข้อมูลเดิมจะถูกลบทิ้งไปจากฐานข้อมูลแม้ว่าฐานข้อมูลทั่ว ๆ ไปจะสามารถรองรับบางแอพพลิเคชั่น (Application) ได้ดี แต่บางครั้งก็ไม่เพียงพอกับแอพพลิเคชั่นที่ต้องการข้อมูลหลาย ๆ State ด้วยสิ่งที่ต้องการคือฐานข้อมูลที่สนับสนุนการตอบคำถาม (Query) ข้อมูลที่เปลี่ยนแปลงตามเวลาอย่างเต็มที่ ฐานข้อมูลที่เก็บข้อมูลหลาย ๆ State เรียกว่าฐานข้อมูลเชิงเวลา (Temporal Database) โดยฐานข้อมูลเชิงเวลานี้จะจัดการกับเวลาของข้อมูลให้อย่างอัตโนมัติ

Temporal Data Model ก็เป็นโมเดลที่มีการสนับสนุนการตอบคำถามของข้อมูลหลาย ๆ State โมเดลนี้จะมีการนำส่วนของเวลาเข้าไปแปะไว้ที่ข้อมูล โดยจะมีการนำเสนอการแปะเวลาไว้ที่ Tuple ของตาราง แต่การแปะไว้ที่ Tuple ของตารางนั้นเราไม่สามารถทราบได้เลยว่าคอลัมน์ไหนเปลี่ยนไปบ้าง จึงได้มีการนำเสนอการนำเวลามาแปะไว้ที่คอลัมน์ เพื่อแก้ปัญหาดังกล่าวข้างต้น ปัญหาที่ตามมาของการแปะเวลาไว้ที่คอลัมน์คือเราไม่ทราบว่าข้อมูลในแต่ละ Tuple นั้นหากมีการเปลี่ยนข้อมูลแล้วข้อมูลที่ทำการเปลี่ยนแปลงเกิดไปซ้ำหรือเหมือนกันกับข้อมูลใน Tuple อื่นเราจะไม่ทราบอีกเช่นกันว่าข้อมูลที่ซ้ำกันนั้นเป็นตัวเดียวกันหรือไม่ จึงได้มีการนำเอาหลักการของค่ารหัสเฉพาะ ซึ่งจะมีอยู่ในฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object Relational Database) เข้ามาช่วยในการแก้ปัญหานี้

1.2 วัตถุประสงค์

1.2.1 เพื่อนำหลักการของฐานข้อมูลเชิงวัตถุสัมพันธ์ มาประยุกต์ใช้กับงานที่เกี่ยวข้องกับข้อมูลเชิงเวลา (Temporal Data)

1.2.2 เพื่อนำหลักการของการ Normalization ถึงในระดับ 3NF โดยใช้หลักการของฐานข้อมูลเชิงเวลา

1.2.3 เพื่อนำหลักการของค่ารหัสเฉพาะ (Object Identification : Oid) มาแก้ปัญหาที่เกิดขึ้นกับข้อมูลที่แปรผันตามเวลาโดยใช้ ค่าข้อมูลชนิด Serial ซึ่งมีใน Informix เข้ามาช่วยในการแก้ปัญหา

1.2.4 เพื่อนำทฤษฎีที่ได้ศึกษามาทั้งหมด มาทำการทดลองสร้างโปรแกรมประยุกต์ที่เป็นไปตามทฤษฎีของฐานข้อมูลเชิงเวลา

1.3 ขอบเขตของงานวิจัย

โครงงานนี้เป็นโครงงาน ที่มีลักษณะเป็นงานวิจัยโดยเป็นการศึกษาถึงการออกแบบฐานข้อมูลเชิงเวลาและการใช้ ค่ารหัสเฉพาะ โดยนำค่าข้อมูลชนิด Serial ซึ่งมีอยู่ใน Informix มาแก้ปัญหาต่าง ๆ ที่เกิดขึ้นกับข้อมูลเชิงเวลา และจะนำหลักการเหล่านั้นมาประยุกต์ตามทฤษฎีฐานข้อมูลเชิงเวลา เป็นระบบสารสนเทศเชิงเวลาที่มี ความสามารถในการเรียกค้นข้อมูลใน State เก่า ๆ ข้อมูลใน State ล่าสุด โดยจะทราบว่าแอตทริบิวต์ (Attribute) ในคอลัมน์ไหนเปลี่ยนแปลงไปบ้างอย่างไร

1.4 วิธีการดำเนินงาน

1.4.1 ศึกษาทฤษฎีของฐานข้อมูลเชิงเวลา

1.4.2 ศึกษาถึงการ Normalization ในระดับ 1 Temporal Normal Form ไป จนถึง 3 Temporal Normal Form โดยจะทำการศึกษาถึง Temporal Function Dependency (TFD) ของตาราง

1.4.3 ศึกษาถึงปัญหาของฐานข้อมูลเชิงสัมพันธ์ที่ขึ้นกับเวลา (Temporal Relational Database)

1.4.4 แก้ปัญหาโดยนำหลักการของฐานข้อมูลเชิงวัตถุสัมพันธ์ มาใช้ และได้นำคอนเซ็ปต์ ของค่ารหัสเฉพาะ เข้ามาช่วยในการแก้ปัญหา

1.4.5 ทำการสร้างโปรแกรมประยุกต์เชิงเวลา โดยสร้างระบบสารสนเทศเชิงเวลา โดยอยู่ในโมเดล ของฐานข้อมูลเชิงวัตถุสัมพันธ์ที่ขึ้นกับเวลา

1.5 อุปกรณ์ที่ใช้ในการดำเนิน

1.5.1 Pentium 200 MMX Ram 96 Megabyte Hardisk 3.2 Gigabyte

1.5.2 Informix Dynamic Server 2000 (IDS2000)

1.5.3 Data Director Object

1.5.4 Informix Document Online

1.5.5 Window Nt Server 4.0 Service Pack 6

1.5.6 Visual Basic

1.5.7 Application Programs อื่น ๆ

บทที่ 2
ฐานข้อมูลเชิงเวลา
(Temporal Database)

2.1 ฐานข้อมูลเชิงเวลา

ในฐานข้อมูลนั้นโดยหลักจะต้องเก็บ Fact ที่เป็นจริงไว้ในฐานข้อมูลเสมอ State เก่า ๆ ของข้อมูลนั้นจะไม่ถูกเก็บไว้ในฐานข้อมูล เกิดว่ามีเหตุการณ์ในการเปลี่ยนแปลง Update ข้อมูล (Data) ในแต่ละ Tuple เกิดขึ้น โดยหลักการเดิมก็จะทำการแก้ไขข้อมูลใหม่ที่ทำการเปลี่ยนแปลง โดยจะเขียนข้อมูล ใหม่นั้นทับลงไปใน Row ที่ทำการเปลี่ยนแปลงทำให้ข้อมูลเดิมเกิดการเปลี่ยนแปลง เรียกได้ว่า Fact เดิมไม่ได้เป็นจริงแล้วเกิดว่าต้องการไปเรียกดูข้อมูลเดิมเราก็ไม่สามารถไปหาข้อมูลนั้นได้เพราะข้อมูลใหม่ได้ถูกเขียนทับไปแล้วเช่นถ้าเกิดว่า นายประยงค์ตอนเข้ามาเรียนปี 1 ได้ลงทะเบียนรหัสวิชา 001 เป็นวิชา คณิตศาสตร์ เกิดว่าพอตอนปี 3 ได้มีการเปลี่ยนรหัสวิชา 001 ไปเป็นภาษาอังกฤษเมื่อเราทำการเรียกค้นข้อมูลมาดูก็จะทำให้เราไม่ได้รับข้อมูลที่เคยเป็นจริงในอดีต ก็คือ รหัสวิชา 001 ที่เคยลงทะเบียนเป็น คณิตศาสตร์ ได้หายไป ซึ่งรหัส 001 ได้กลายเป็น ภาษาอังกฤษ ทำให้เราได้ข้อมูลที่ไม่ถูกต้องเนื่องจากแต่เดิมนายประยงค์ได้ เคยผ่านการเรียนวิชา คณิตศาสตร์ มาแล้วกลายเป็นว่าข้อมูลใหม่นั้นนายประยงค์ไม่ได้เรียนวิชาคณิตศาสตร์ โดยเราได้มีวิธีการนำเสนอแนวทางในการแก้ปัญหาโดยนำเสนอหลักการของฐานข้อมูลเชิงเวลา โดยจะทำการเพิ่มแอตทริบิวต์ ของเวลาเข้ามา 2 คอลัมน์ โดยเราจะทำการเพิ่มคอลัมน์ Timestart และ Timeend เพื่อบอกถึงช่วงเวลาเริ่มต้นที่ข้อมูลนั้นได้เป็นจริงและช่วงเวลาสิ้นสุด ของข้อมูล ตามลำดับ หลักการของฐานข้อมูลเชิงเวลาคือมีการเพิ่มส่วนของเวลาติดไปกับส่วนของข้อมูล โดยจะมีการเพิ่มคอลัมน์ ขึ้นมา 2 คอลัมน์ ก็คือ Timestart และ Timeend ซึ่งเรียกว่าการทำ Timestamp โดยจะทำการบันทึกเวลาที่ทำการเปลี่ยนแปลงฐานข้อมูลใน Tuple ของตารางนั้น ตั้งแต่ระยะเริ่มต้นเราจะบันทึกใน Timestat แรก และ Timeend ก็ทำการบันทึกเวลาที่สิ้นสุดก็คือเวลาที่เกิดการเปลี่ยนแปลงนั่นเอง ถ้าเกิดว่ามีการเปลี่ยนแปลงข้อมูลใน Tuple นั้นเพียงแค่แอตทริบิวต์ เดียวเราต้อง Timestamp ข้อมูลแล้วสร้าง Tuple ใหม่เพื่อใส่ข้อมูลที่เราได้ทำการเปลี่ยนแปลงและบันทึกเวลาใหม่ที่ทำการเปลี่ยนแปลงและก็ Timestamp ส่วนเวลาเริ่มต้นและเวลาสิ้นสุดของการเปลี่ยนแปลงลงไปใน Timestart และ Timeend หากส่วนเวลาสิ้นสุดเรายังไม่ทราบเราก็จะกำหนดเป็น Forever

ในส่วนของเวลาที่เกี่ยวข้องกับฐานข้อมูลเชิงเวลา นั้นเราจำแนกเป็น 3 อย่างคือ Valid Time Transaction Time และ User Define Time โดย

Valid Time เป็นเวลาที่ทำการบันทึกไปในฐานข้อมูลนั้นเหมือนกันแต่จะเป็นช่วงเวลาที่ฐานข้อมูลที่บันทึกนั้นเป็นจริง เช่น ถ้าเมื่อก่อนมีการบันทึกว่ายอดเขา เอฟเวอเรสมีความสูง 500 กิโลเมตร เราก็จะมีการบันทึกข้อมูลตารางโดยเก็บค่าเริ่มต้นไว้ใน Transaction Time และใน Valid Time จะเก็บเพื่อยืนยันช่วงเวลาที่เป็นจริง ในอนาคตอีก 10 ปี ข้างหน้าอาจจะมีการวัดความสูงของยอดเขา เอฟเวอเรสใหม่ ค่าอาจจะไม่เท่าเดิมแล้วเช่น อาจเกิดจากเทคโนโลยีที่ใช้การวัดที่ละเอียดขึ้น แม่นยำหรือ อื่นๆ เราต้องการบันทึก Transaction time ใหม่โดยการสร้าง Tuple ใหม่ขึ้น เพราะ Transaction Time เราไม่สามารถไปแก้เวลาที่ค้นพบข้อมูลได้เมื่อสร้าง Tuple ใหม่ก็บันทึก Transaction Time คือเวลาที่ค้นพบ แล้วบันทึก Valid Time คือเวลาที่เป็นจริง

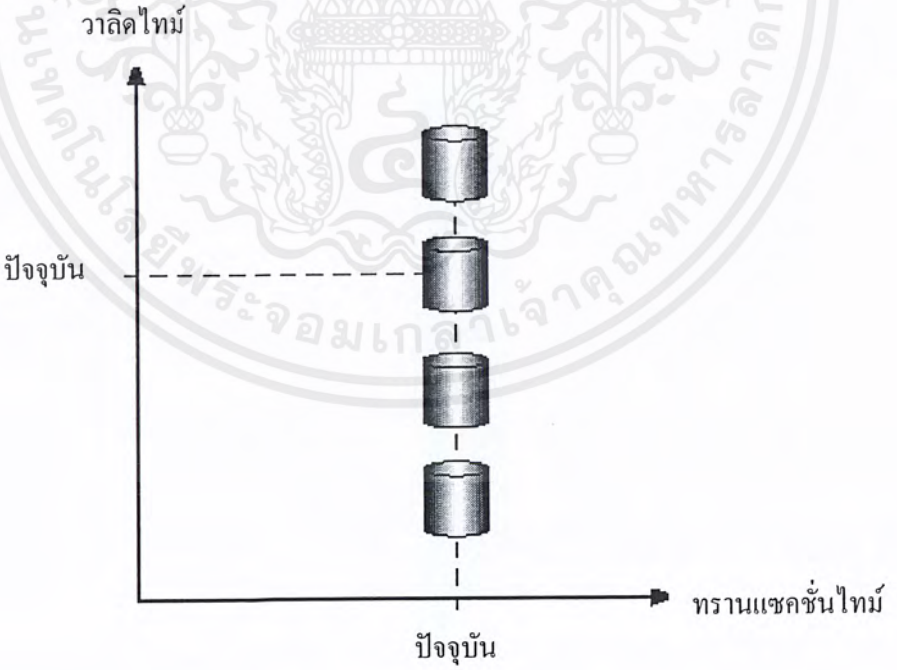
Transaction Time คือ เวลาที่ได้มีการบันทึกลงในฐานข้อมูล คือบอกว่าข้อมูลได้รับการจัดการเมื่อใด เช่น Insert เมื่อใด Update เมื่อใด เป็นต้น

User-define time ก็คือเวลาที่เราเป็นคนกำหนดขึ้นมาเองซึ่งเป็นช่วงเวลาที่จะไม่เปลี่ยนแปลง (เป็นจริงเสมอ) เช่น วันเกิด วันหยุดนักขัตฤกษ์ วันขึ้นปีใหม่ เป็นต้น

ประเภทของฐานข้อมูลเชิงเวลา

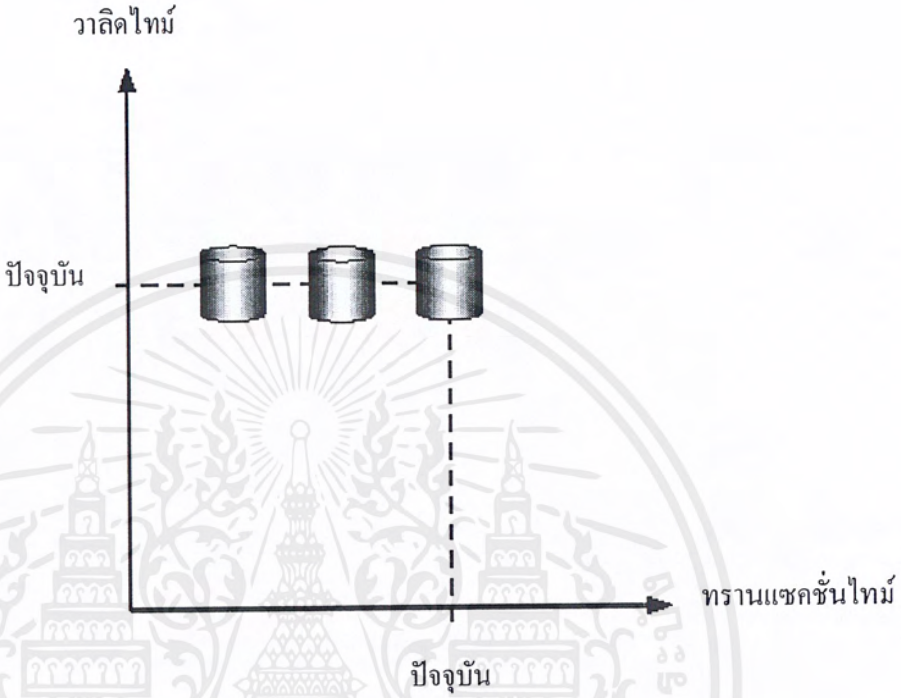
Valid Time และ Transaction Time ก่อให้เกิดฐานข้อมูลเชิงเวลาหลายลักษณะดังต่อไปนี้

1 ฐานข้อมูลวาลิดไทม์ (Valid-Time Database) หรือฐานข้อมูลประวัติ (Historical Database) เก็บเฉพาะวาลิดไทม์ดังรูปที่ 2-1



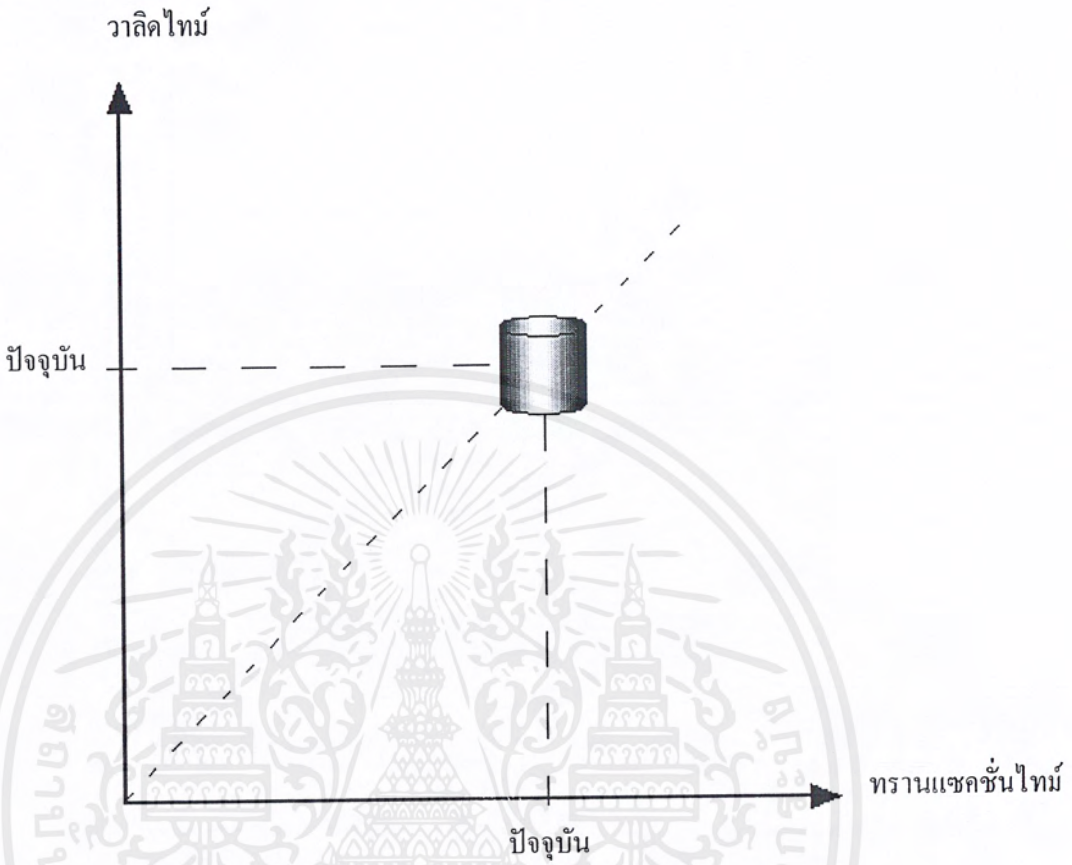
รูปที่ 2-1 ฐานข้อมูลวาลิดไทม์

2 ฐานข้อมูลทรานแซคชั่นไทม์ (Transaction-Time Database) หรือฐานข้อมูลย้อนกลับ (Rollback Database) เก็บเฉพาะทรานแซคชั่นไทม์รูปที่ 2-2



รูปที่ 2-2 ฐานข้อมูลทรานแซคชั่นไทม์

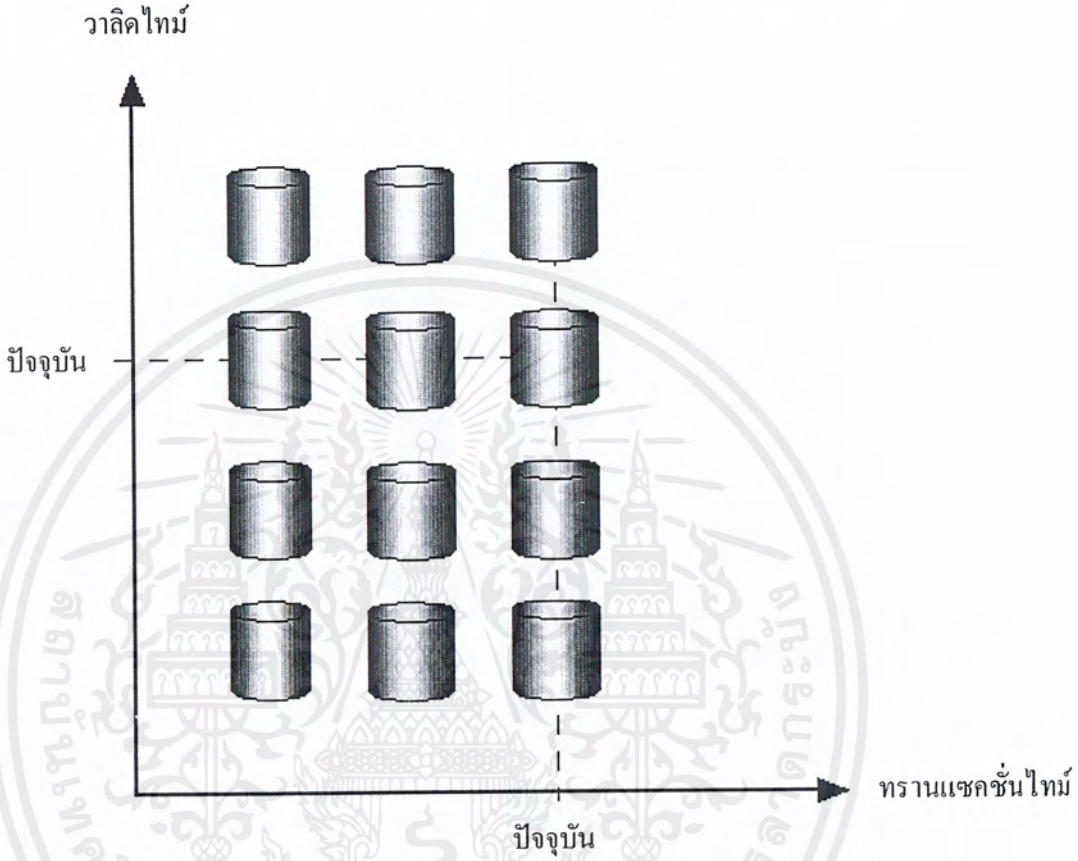
3. ฐานข้อมูลไบเทมโพรอล (Bitemporal Database) เก็บทั้งวาลิดไทม์ และ ทรานแซคชั่นไทม์ ในฐานข้อมูลเชิงพาณิชย์จะเก็บข้อมูลที่เป็นจริงในปัจจุบันเท่านั้น ฐานข้อมูลแบบนี้มักจะเรียกว่า ฐานข้อมูลสแน็ปช็อท (Snapshot Database) โดยฐานข้อมูลสแน็ปช็อทในแบบของวาลิดไทม์และทรานแซคชั่นไทม์แสดงดังรูปที่ 2-3



รูปที่ 2-3 ฐานข้อมูลสแน็ปช็อท

และ ฐานข้อมูลไบเทมโพรอล อาจจะเป็นได้ทั้งตารางสแน็ปช็อท (Snapshot Table) คือ เก็บเฉพาะข้อมูลปัจจุบัน หรือตารางวาลิดไทม์ คือเก็บว่าเมื่อใดที่ข้อมูลเป็นจริงหรือ ตารางทรานแซคชั่นไทม์ (Transaction-Time Table) คือ เก็บว่าข้อมูลถูกบันทึกในฐานข้อมูลเมื่อใด หรือตารางไบเทมโพรอล (Bitemporal Table) คือเก็บทั้งวาลิดไทม์และทรานแซคชั่นไทม์ โดยมีลักษณะของฐานข้อมูลดังรูปที่ 2-4

ฐานข้อมูลไบเทมโพรอลในแบบของวาลิดไทม์และทรานแซคชั่นไทม์แสดงดังรูปที่ 2-4



รูปที่ 2-4 ฐานข้อมูลไบเทมโพรอล

ระบบจัดการฐานข้อมูลเชิงเวลา

ระบบจัดการฐานข้อมูลเชิงเวลาควรจะสนับสนุน Temporal Data Definition Language, Temporal Data Manipulation Language, Temporal Query Language และ Temporal Constraint

แม้ว่าระบบจัดการฐานข้อมูลบางระบบจะสนับสนุนชนิดของข้อมูลที่เป็นวันและเวลา แต่ก็ไม่สามารถเรียกได้ว่าเป็นระบบจัดการฐานข้อมูลเชิงเวลา เนื่องจากถ้าต้องการข้อมูลอดีต ผู้ใช้จะต้องจัดการเองทั้งหมด โดยปราศจากการสนับสนุนใดๆจากระบบจัดการฐานข้อมูล

2.2 การออกแบบฐานข้อมูลเชิงเวลา (Temporal Databases Design)

โดยทั่วไปในการออกแบบฐานข้อมูล ที่ดีจะมีจุดประสงค์เพื่อที่จะกำจัดการซ้ำซ้อนของข้อมูลและการแทรก (Insertion) และ การลบข้อมูล (Deletion) ของข้อมูลที่ไม่ถูกต้องโดยที่ในการออกแบบฐานข้อมูลเชิงเวลานั้นจะต้องมีความเข้าใจ Temporal type ซึ่งหมายถึง ชนิดของข้อมูลที่เกี่ยวกับเวลานั้นเองและจะต้องมีความเข้าใจเกี่ยวกับ ฟังก์ชันการขึ้นต่อกัน (Temporal Functional Dependencies: TFDs) โดยที่ TFD ก็คือ การเพิ่มส่วนของเวลาของฟังก์ชันการขึ้นต่อกัน (Functional: FD) โดยจะเขียนในรูปแบบ $X \rightarrow_{\mu} Y$ ซึ่งมีความหมายว่า X มีฟังก์ชันการขึ้นอยู่กับ Y ภายใต้ Temporal type μ ซึ่งจะกล่าวต่อไป

2.3 การนอร์มัลไลซ์ (Normalization)

2.3.1 การนอร์มัลไลซ์แบบปกติ

ในการออกแบบฐานข้อมูลแบบรีเลชันนอล (Relational Database) การนอร์มัลไลซ์เป็นการออกแบบฐานข้อมูลที่เป็นมาตรฐานที่สุด โดยมีจุดประสงค์ของการออกแบบเพื่อลดความซ้ำซ้อนของความสัมพันธ์ของข้อมูลให้เหลือน้อยที่สุด (Minimum Redundancy) ซึ่งตามมาตรฐานปกติจะมี 3 ระดับคือ

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)

โดยที่รีเลชันใดที่ยังไม่สอดคล้องตามรูปแบบนอร์มัล (Normal Form) ทั้งสามก็จะต้องมีแยกรีเลชันนั้น ๆ ออกเป็นรีเลชันย่อย ๆ (Decomposition Method) ต่อไป คือ

- 4NF (Forth Normal Form)
- 5NF (Fifth Normal Form)

หากรีเลชันใดมีมาตรฐานถึงรูปแบบนอร์มัลระดับที่ 5 (5NF) แล้วก็จะมั่นใจได้ว่ารีเลชันนั้นจะไม่มีความซ้ำซ้อนของความสัมพันธ์ของข้อมูลอย่างแน่นอน นอกจากนี้แล้วยังมีการออกแบบรูปแบบนอร์มัลเพิ่มเติมระหว่างรูปแบบนอร์มัลที่ 3 (3NF) และรูปแบบนอร์มัลที่ 4 (4NF) โดย Boyce และ Codd ซึ่งมีชื่อว่า Boyce Codd Normal Form (BCNF) อีกด้วย

รูปแบบนอร์มัลระดับที่ 1 (First Normal Form)

การปรับรีเลชันให้อยู่ในรูปแบบนอร์มัลระดับที่ 1 คือ การปรับจากรีเลชันที่ไม่นอร์มัลไลซ์(Unnormalize Relation) เช่น รีเลชันที่มีข้อมูลของแอตทริบิวต์บางคอลัมน์มากกว่า 1 ค่า (มีแอตทริบิวต์ที่มีข้อมูลเป็น Repeating group)

นิยาม รีเลชันจะอยู่ในรูปแบบนอร์มัลระดับที่ 1 (1NF) ก็ต่อเมื่อโดเมนของแต่ล่ะแอตทริบิวต์ประกอบไปด้วยข้อมูลที่เป็นหน่วยย่อที่สุด

สิ่งที่ได้จากการปรับรีเลชันให้อยู่ในรูปแบบระดับที่ 1 ก็คือ รีเลชันยังคงมีความซ้ำซ้อนของความสัมพันธ์ระหว่างข้อมูลอยู่มากมาย เพราะนิยามของรูปแบบนอร์มัลระดับที่ 1 นี้กำหนดเพียงเพราะว่าแต่ละแอตทริบิวต์ของรีเลชันจะมีโดเมนที่มีสมาชิกเป็นหน่วยย่อยที่สุดเท่านั้น มิได้ลดความซ้ำซ้อนของความสัมพันธ์ระหว่างข้อมูลแต่ประการใด

รูปแบบนอร์มัลระดับที่ 2 (Second Normal Form)

นิยาม รีเลชันจะอยู่ในรูปแบบระดับที่ 2 (2NF) ก็ต่อเมื่อรีเลชันนั้นอยู่ในรูปแบบนอร์มัลระดับที่ 1 แล้วและทุกแอตทริบิวต์ที่ไม่เป็นส่วนหนึ่งของคีย์หลัก (Non-key attribute) จะต้องขึ้นอยู่กับคีย์หลักของรีเลชันนั้นอย่างเต็มที่ (Full Dependency) สิ่งที่ได้จากการปรับให้อยู่ในรูปแบบนอร์มัลระดับที่ 2 คือ ข้อมูลของบางแอตทริบิวต์ที่ไม่ใช่คีย์หลักอาจมีความสัมพันธ์กันเองที่ไม่มีความหมายกับคีย์หลักเลย ซึ่งความสัมพันธ์ดังกล่าวนี้ถือว่าเป็นความซ้ำซอนประการหนึ่งของรีเลชัน ๆ ที่จะต้องทำการลดด้วยรูปแบบนอร์มัลในระดับต่อไป

รูปแบบนอร์มัลระดับที่ 3 (Third Normal Form)

นิยาม รีเลชันจะอยู่ในรูปแบบนอร์มัลระดับที่ 3 (3NF) ก็ต่อเมื่อรีเลชันนั้นอยู่ในรูปแบบนอร์มัลระดับที่ 2 แล้วและทุกแอตทริบิวต์ที่ไม่เป็นส่วนใดส่วนหนึ่งของคีย์หลักจะต้องไม่เป็นฟังก์ชันที่ขึ้นต่อกันเอง หรือ ไม่มี Transitive FD ระหว่าง Primary key กับ Non-key attribute นั้นเอง โดยปกติแล้ว สิ่งที่ได้จากการที่รีเลชันอยู่ในรูปแบบนอร์มัลระดับที่ 3 คือ รีเลชันจะไม่มีความซ้ำซ้อนอีกต่อไปโดยที่จะสอดคล้องกับรูปแบบนอร์มัลระดับที่ 4 และที่ 5 ด้วย แต่รีเลชันบางลักษณะที่จะต้องทำให้อยู่ในรูปแบบนอร์มัลระดับที่ 4 และ 5 ต่อไปอีก รีเลชันดังกล่าวจะมีลักษณะดังต่อไปนี้

- เป็นรีเลชันที่มีหลายคีย์คู่แข่ง (Multiple Candidate key)
- เป็นรีเลชันที่มีคีย์คู่แข่งที่เกิดจากการรวมกันของคีย์ย่อย ๆ (Candidate key เป็น Combine key)
- เป็นรีเลชันที่มีการเหลื่อมซ้อนกัน (Overlap กัน)

สำหรับการทำนอร์มัลไลซ์ในการ ออกแบบฐานข้อมูลเชิงเวลานั้นจะต้องเข้าใจถึงหลักการออกแบบของฐานข้อมูลรีเลชันก่อน แล้วค่อยพิจารณาส่วนของเวลาที่เข้ามาเกี่ยวข้องโดยที่จะต้องเข้าใจถึงหลักการของ Temporal Normal Form ซึ่งเป็นสิ่งที่สำคัญในการออกแบบฐานข้อมูลเชิงเวลา (Temporal Database Design)

2.3.2 การนอร์มัลไลซ์แบบ Temporal

First Temporal Normal Form (1TNF)

Definition "A time slice at point t has to result in a standard 1NF-relation"

นิยาม รีเลชันจะต้องอยู่ในมาตรฐานของรูปแบบนอร์มัลที่ 1 ก่อน และในรีเลชันนั้นจะต้องมี Valid-time จะแทนโดย Snapshot Rrelation ในแต่ละแอตทริบิวต์หรือ Tuple Timestamping

สำหรับ Valid-time ที่จะใช้ในการทำ Timestamp นั้นจะประกอบด้วยแอตทริบิวต์ T_s ซึ่งเป็นการเริ่มของ valid time และแอตทริบิวต์ T_e ซึ่งเป็นการสิ้นสุดของ valid time สำหรับตัวอย่างตารางที่เป็น 1TNF แสดงดังรูปข้างล่าง

EMP	Dept	Ts	Te
Bill	Shipping	1	4
Bill	Loading	5	10

ตารางที่ 2-1 แสดงตารางที่เป็น 1TNF

เป็นตารางแสดงถึงการเก็บสถานะของลูกจ้างชื่อ Bill ที่อยู่ใน Department ณ ช่วงเวลาตั้งแต่ 1 ถึง 10 สำหรับการพิจารณารีเลชันที่เป็น **Second Temporal Normal Form (2TNF)** และ **Third Temporal Normal Form (3TNF)** ก็จะคล้ายกับการปรับให้อยู่ในมาตรฐานของการทำรูปแบบนอร์มัลของรีเลชันปรกติแต่การพิจารณาแทนที่จะใช้ฟังก์ชันการขึ้นต่อกัน (Functional Dependency: FD) ก็จะใช้ ฟังก์ชันการขึ้นต่อกันเชิงเวลา (Temporal Functional Dependency: TFD)

2.3.3 ฟังก์ชันการขึ้นต่อกัน (Functional Dependency)

2.3.3.1 ฟังก์ชันการขึ้นต่อกันแบบปกติ (Conventional Functional Dependency)

ฟังก์ชันการขึ้นต่อกัน เป็นข้อกำหนดที่ช่วยให้เห็นความสัมพันธ์ของแอตทริบิวต์ต่าง ๆ ที่อยู่ในรีเลชันเพราะเป็นไปได้ที่แอตทริบิวต์ต่าง ๆ ที่อยู่ในเอนติตี้เดียวกันเหล่านี้มีความสัมพันธ์กันเอง โดยที่ความสัมพันธ์นี้ อาจเกี่ยวข้องหรือไม่เกี่ยวข้องกับความสัมพันธ์ที่มันมีต่อคีย์หลักของเอนติตี้นั้นก็เป็นได้ ซึ่งการที่แอตทริบิวต์เหล่านั้นมีความสัมพันธ์กันเองจะเป็นสิ่งที่เราต้อง

พิจารณาแยกออกเป็นรีเลชันย่อย ๆ เพราะแอตทริบิวต์แต่รีเลชันก็ควรมีความสัมพันธ์กับคีย์หลักของรีเลชันตนเองเท่านั้น

กำหนดรีเลชัน R ถ้ามีแอตทริบิวต์ Y ของ R เป็นฟังก์ชันที่ขึ้นต่อแอตทริบิวต์ X ของรีเลชัน เราสามารถเขียนได้ด้วยสัญลักษณ์

$$R.X \rightarrow R.Y$$

อ่านว่า R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y

หรือ R.X มีฟังก์ชันในการเลือก R.Y

หรือ R.Y ขึ้นอยู่กับ R.X

นิยาม R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y ก็ต่อเมื่อ ทุกค่าข้อมูลของแอตทริบิวต์ X ใน R มีค่าข้อมูลของแอตทริบิวต์ Y ใน R ได้เพียงค่าเดียวเสมอ โดยที่แอตทริบิวต์ X และ Y อาจจะเป็นคีย์ประกอบ (Composite key) ก็ได้

รีเลชัน R

X	Y
S1	P1
S1	P2
S2	P2
S3	P4

ตารางที่ 2-2 แสดง R.X มีฟังก์ชันการขึ้นอยู่กับ R.Y

นิยาม R.Y มีฟังก์ชันการขึ้นอยู่กับ R.X อย่างเต็มที่ก็ต่อเมื่อ R.X มีฟังก์ชันการขึ้นอยู่กับ R.X และไม่ขึ้นอยู่กับข้อมูลเพียงบางส่วนของ R.X โดยที่แอตทริบิวต์ของ X และ Y อาจจะเป็นคีย์ประกอบก็ได้

รีเลชัน R

X

X1	X2	Y
S1	P1	Red
S1	P2	Blue
S2	P2	Black
S3	P4	Green

ตารางที่ 2-3 รูปแสดง R.Y มีฟังก์ชันการขึ้นอยู่กับ R.X อย่างเต็มที่

2.3.3.2 ฟังก์ชันการขึ้นต่อกันเชิงเวลา (TFD: Temporal Functional dependency)

นิยาม ให้ X และ Y เป็นเซตของแอตทริบิวต์ที่จำกัด และ μ เป็น ชนิดของ Temporal ซึ่ง μ (i) $\neq \phi$ สำหรับค่า i ดังนั้น $X \rightarrow_{\mu} Y$ เรียกว่า Temporal Functional Dependency (TFD)

Temporal types (μ) เป็นสิ่งที่แสดงถึงความสัมพันธ์เชิงเวลา (Time Relation) ซึ่ง ความสัมพันธ์เชิงเวลาจะเป็นปัจจัยของ Binary Relation ที่อยู่บนเซตของเวลา (time points) สำหรับ Temporal types อาจจะหมายถึง ขนาดของช่วงเวลา (time granularity) ก็ได้เช่น วัน (day), เดือน (month), สัปดาห์ (week) และ ปี (year)

พิจารณาตัวอย่างของตารางรีเลชัน "Employee" ซึ่งประกอบไปด้วยแอตทริบิวต์เกี่ยวกับ "Employee" ดังนี้ Name Dept และ Sal ซึ่งก็คือ ชื่อของลูกจ้างแต่ล่ะคน แผนก และเงินเดือนของลูกจ้างตามลำดับ โดย Name จะต้องเป็นชื่อที่ไม่ซ้ำ (Uniquely) สามารถแสดงได้ดังนี้

EMP

Name	Dept	Sal

โดยมีข้อบังคับ(constraints)ไว้ว่า

C1: *An employee cannot have two distinct salaries within a month.*

ในระยะเวลา 1 เดือน ลูกจ้างไม่สามารถรับเงินเดือนได้มากกว่า 1 ครั้ง

C2: *An employee cannot change department within a year.*

ในระยะเวลา 1 ปี ลูกจ้างจะไม่สามารถเปลี่ยนแผนกได้

สามารถเขียน FD (Functional Dependency)ได้ดังนี้

$$EMP: Name \longrightarrow Sal$$

$$EMP: Name \longrightarrow Dept$$

ลูกจ้างจะได้รับเงินเดือนโดยในแต่ละเดือนสามารถรับเงินได้เพียง ครั้งเดียวดังนั้นสามารถเขียนใหม่ที่เป็น TFD (Temporal Functional Dependency) ได้ดังนี้

$$EMP: Name \longrightarrow_{Month} Sal$$

$$EMP: Name \longrightarrow_{year} Sal$$

2.4 การแก้ปัญหาที่เกิดขึ้นกับข้อมูลที่แปรผันตามเวลา

ในส่วนนี้จะเสนอว่าทำไม Object Identification ถึงมีประโยชน์ในฐานข้อมูลเชิงเวลาโดยทั่วไปและเงื่อนไขความถูกต้องของฐานข้อมูลเชิงเวลา โดยเฉพาะ TFDs ถูกนำเสนอให้ใช้ในเงื่อนไขความถูกต้อง (Integrity Constraints)

พิจารณาถึง Histrorical Relation ของ EMP ที่อยู่บน schema {Name, Dept, Sal, Valid} ซึ่งเป็นการเก็บข้อมูลประวัติของลูกจ้างจะมี Valid-time ที่มี Timestamping ที่แอตทริบิวต์ด้วย เดิมทีจะมีแอตทริบิวต์ Name Dept และ Sal เป็นแอตทริบิวต์เดิม สามารถสร้างแอตทริบิวต์ในแต่ละแถว (Tuple) ได้ดังนี้

{Name: x, Dept : y, Sal : z, Valid : [d1 – d2] }

หมายความว่าจากช่วงวันที่ d1 จนกระทั่งถึงวันที่ d2 ลูกจ้างชื่อ x ทำงานใน Department ชื่อ y และได้รับเงินเดือนเท่ากับ z การใช้ “month/day” แทนวันที่ time slice ของ EMP ที่วันที่ d แสดงถึงความสัมพันธ์ snapshot (snapshot relation) ที่อยู่บน {Name, Dept, Sal} ซึ่งประกอบด้วย

{Name: x, Dept : y, Sal : z} เมื่อใดก็ตามตาราง EMP มีค่า แถว เป็น

{Name: x, Dept : y, Sal : z, Valid : [d1-d2]}

สำหรับค่า d จะอยู่ระหว่าง d1 และ d2 ถ้าค่า t อยู่ใน timeslice ของ EMP ที่วันที่เท่ากับ d ดังนั้นเราสามารถพูดได้ว่า t เป็น valiid-time ในตาราง EMP ณ วันเวลาเท่ากับ d โดยที่เราจะแนะนำถึงเงื่อนไขความถูกต้องของตาราง (Integrity Constraint) ขั้นแรก Name ของลูกจ้างจะต้องไม่ซ้ำกัน (Uniquely identified) งานที่จะทำจะต้องอยู่ใน Department เดียวกันเท่านั้น และรายได้เงินเดือนจะรับเพียง 1 ครั้งต่อเดือน ดังนั้นเราจะให้ Name เป็น Primary key ในแต่ละ timeslice ซึ่งเราจะกำหนดเงื่อนไขความถูกต้องดังนี้

C1 ลูกจ้างไม่สามารถรับเงินเดือนได้มากกว่า 1 ครั้ง

C2 ลูกจ้างไม่สามารถเปลี่ยน department ได้ในเวลา 1 ปี

ดังนั้นจากเงื่อนไขข้างต้นจะสามารถเขียนTemporal FDs (TFDs) ได้ดังนี้

$$\text{Name} \rightarrow_{\text{month}} \text{Sal} \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$\text{Name} \rightarrow_{\text{year}} \text{Dept} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

จากข้างต้นสำหรับวันที่ทั้งหมด d1 และ d2 จะอยู่ในช่วงเวลา 1 ปี ถ้า {Name : x, Dept : y ,Sal : z} เป็นช่วงเวลาจริง (valid) ในตาราง Emp ณ วันที่เท่ากับ d1 และ {Name : x, Dept : y', Sal : z'} ณ วันที่ d2 ดังนั้นจะได้ $y = y'$ ด้วย

EMP-G

Name	Dept	Sal
[1/1-5/3] Tom [5/4-12/31] Thomas	[1/1-12/31] MIS	[1/1-8/31] 120 [9/1-12/31] 130
[1/1-12/31] Jim	[1/1-5/3] Candy [5/4-12/31]Sweets	[1/1-12/31] 110
[5/4-12/31] Tom	[5/4-12/31]Sweets	[5/4-12/31] 105

ตารางที่ 2-4 Prototypical temporally grouped relation

จากรูป 2-6 แสดงถึง ลูกจ้างที่อยู่ในตารางเก็บประวัติลูกจ้าง เป็นกลุ่มซึ่งเป็น Non-first-normal-form (N1NF) tuple จะเป็นการนำ Valid time ไปแปะทุกแอตทริบิวต์โดยจะแทนเป็น "เดือน/วัน" สมมติเป็นปี 1999 ในตาราง EMP-G จะเก็บข้อมูลของลูกจ้าง 3 คน โดยในวันที่ 4 พฤษภาคม[5/4] คนที่ชื่อว่า "Tom" เปลี่ยนชื่อเป็น "Thomas" เมื่อมีลูกจ้างคนใหม่ที่ชื่อ "Tom" เข้ามาใหม่ซึ่งบังเอิญชื่อไปซ้ำคนเดิมตามเงื่อนไขข้อบังคับ C1 และ C2 ทำให้เกิดข้อผิดพลาดตามกฎคือ

{Name: Tom, Dept: Mis, Sal:120} เป็นจริง ณ วันที่ 3 เดือนพฤษภาคม[5/3]

{Name: Tom, Dept: Sweets, Sal:105} เป็นจริงในวันภายหลัง

แต่กระนั้นจะทำให้ผิดจากเงื่อนไขข้อบังคับ 2 ข้อ

C1 เห็นได้ชัดเลยว่าใน 1 เดือน จะต้องมีคนชื่อ Tom รับเงินเดือนเกิน 1 ครั้ง

C2 จะเป็นการละเมิดเงื่อนไขข้อบังคับคือ ลูกจ้างชื่อ "Jim" จะมีการเปลี่ยน Department คือเปลี่ยนจาก Department ชื่อ "Candy" ไป "Sweets" เมื่อ วันที่ 4 เดือนพฤษภาคม[5/4]

DEPT_G

Name
[1/1-12/31] MIS
[1/1-5/3] Candy
[5/4-12/31] Sweets

DEPT_G′

Name
[1/1-12/31] MIS
[1/1-12/31] Candy
[1/1-12/31] Sweets

ตารางที่ 2-5 แสดงถึงการแยก Department

จากตารางรูปที่ 2-9 Dept_G Relation ทางด้านบนแสดงถึง Department "Candy" เปลี่ยนชื่อเป็น Sweets ในวันที่ 4 เดือนพฤษภาคม[5/4] ส่วนตารางความสัมพันธ์ DEPT-G′ รูปแสดงทางด้านล่างจะแสดง Candy และ Sweets ที่แยก Department แต่จะเกิดปัญหาจากคำถามว่า

"ในลูกจ้างแต่ละคนให้ list จำนวนของ Department"

ในสถานการณ์ที่เหล่านี้ จากตาราง EMP-G ไม่สามารถตอบคำถามได้เลยจะเกิดปัญหาแน่

EMP_0

λ	Name	Dept	Sal
[1/1-12/31] *oid3*	[1/1-5/3] Tom [5/4-12/31Thomas	[1/1-12/31] *oid1*	[1/1-8/31] 120 [9/1-12/31]130
[1/1-12/31] *oid4*	[1/1-12/31] Jim	[1/1-12/31] *oid2*	[1/1-12/31] 130
[5/4-12/31] *oid5*	[5/4-12/31] Tom	[5/4-12/31] *oid2*	[5/4-12/31] 150

DEPT_0

λ	Name
[1/1-12/31] *oid1*	[1/1-12/31] MIS
[1/1-12/31] *oid2*	[1/1-5/3] Candy [5/4-12/31] Sweets

ตารางที่ 2-6 แสดงถึง Temporally Grouped Relation

จะใช้ *Object-identifier (OID) ใน NINF tuples* โดยจะมีแอตทริบิวต์พิเศษ λ จะใช้แทน Object-identity และมีเงื่อนไขข้อบังคับ

EMP_0 : x $\rightarrow_{month}$ Sal .....(3)

EMP_0 : x $\rightarrow_{year}$ Dept .....(4)

EMP_0 Relation จะอ้างถึง Departments โดยใช้ Oid ซึ่งเป็นผลต่อเนื่องจากการพิสูจน์ C2 จาก EMP_0 Relation ทำให้ทราบว่า Jim ทำงานเพียง Departments เดียวเท่านั้น

EMP_0

λ	Name	Dept	Sal	Valid
Oid3	Tom	Oid1	120	[1/1-5/3]
Oid3	Thomas	Oid1	120	[5/4-8/31]
Oid3	Thomas	Oid1	130	[9/1-12/31]
Oid4	Jim	Oid2	110	[1/1-12/31]
Oid5	Tom	Oid2	105	[5/4-12/31]

DEPT_0

λ	Name	Valid
Oid1	MIS	[1/1-12/31]
Oid2	Candy	[1/1-5/3]
Oid2	Sweets	[5/4-12/31]

ตารางที่ 2-7 แสดง Temporally Ungrouped Relation

ซึ่งเป็นผลจากการแยกกลุ่มของ Relation รูปที่ 2-7 จะเป็นการนำ Valid time ไปแปะเป็น Snapshot Relation ซึ่งทำให้ไม่มีการสูญหายของข้อมูล การแทน Temporally Relation เป็นลำดับด้วย Snapshot จะทำให้มีข้อดีเป็นประโยชน์ช่วยให้ง่ายต่อการตอบคำถามอันเนื่องมาจากสามารถตรวจสอบย้อนหลังได้จากค่า Object-identifiers (Oid) เพราะค่ามันไม่เปลี่ยน

บทที่ 3
ฐานข้อมูลเชิงวัตถุสัมพันธ์
(Object Relational Database)

3.1 แนวคิดเชิงวัตถุ (Object-Oriented)

แนวคิดเชิงวัตถุเป็นการรวมโค้ด (Code) และตัวแปรข้อมูล (Data) ไว้ด้วยกันหรือเป็นการปกป้องข้อมูลไว้ภายใต้โค้ด (Encapsulate) ซึ่งเรียกว่าออบเจ็กต์ (Object) แต่ละออบเจ็กต์จะเชื่อมต่อกันผ่านทางเมสเซจ (Message) ซึ่งเมสเซจคือการส่งการร้องขอ (Request) ระหว่างออบเจ็กต์โดยไม่คำนึงถึงรายละเอียดในการอิมพลีเมนต์ (Implementation) หรือโครงสร้างภายใน (Internal Structure) ของออบเจ็กต์นั้น ออบเจ็กต์จะตอบสนองการติดต่อจากภายนอกที่เป็นเมสเซจเท่านั้น ทำให้การแก้ไขหรือปรับปรุงออบเจ็กต์หนึ่งไม่มีผลต่อ ออบเจ็กต์อื่น ๆ อีกทั้งเรายังสามารถปกป้องข้อมูลภายในออบเจ็กต์จากการใช้ที่ผิด ๆ อีกด้วย ซึ่งเป็นข้อดีที่สำคัญอย่างหนึ่งของแนวคิดเชิงวัตถุ

ออบเจ็กต์สามารถถ่ายทอดคุณสมบัติ (Property) และการดำเนินการ (Operation) ของออบเจ็กต์ได้โดย การทำการถ่ายทอดคุณสมบัติ ซึ่งจะทำให้มีการใช้คุณสมบัติและฟังก์ชันร่วมกัน จัดความสัมพันธ์ให้อยู่ในรูป ซูเปอร์คลาส (Superclass) และ สับคลาส (Subclass) โดยสับคลาสสามารถรับการถ่ายทอดคุณสมบัติจากซูเปอร์คลาส ๆ เดียวหรือหลายคลาสก็ได้ อีกทั้งออบเจ็กต์ยังมีความสามารถในการทำโพลีมอร์ฟิซึม คือความสามารถในการอ้างถึงอินสแตนท์หลายชนิดหรือคลาส ณ ขณะกำลังทำงาน โดยสามารถสร้างรูทีน (Routine) ที่มีชื่อเดียวกันแต่ต่างอาร์กิวเมนท์ (Argument) เพื่อให้ง่ายต่อการใช้งาน (One Interface Multiple Method) เรียกว่ารูทีนโอเวอร์โหลดดิง (Routine Overloadind) โดยดาต้าเบสเซิร์ฟเวอร์ (Database Server) จะทำการตัดสินใจว่าจะเรียกใช้รูทีนใดดูจากรูทีน ซิกเนเจอร์ (Routine Signature)

สรุปคุณสมบัติที่ออบเจ็กต์พึงมี

- เป็นการมองภาพของสิ่งที่เราสนใจ มีลักษณะเฉพาะของแต่ละออบเจ็กต์ซึ่งเรารู้ว่าออบเจ็กต์นี้ทำอะไรและจะต้องเรียกผ่านแมสเสจใด
- มีขอบเขตที่ชัดเจน กำหนดการปกป้องข้อมูลว่าใครจะใช้ได้ใครใช้ไม่ได้
- มีวิธีการติดต่อระหว่างกันชัดเจน
- ปกป้องข้อมูลที่มีอยู่ภายในตามประเภทของข้อมูลที่มีอยู่สามแบบ คือ Public , Private และ Protected
- สามารถถ่ายทอดคุณสมบัติของออบเจ็กต์หนึ่งไปยังอีกออบเจ็กต์ได้
- มีความสามารถต่าง ๆ เช่น มีความสามารถในการทำ Overloading , Late Binding เป็นต้น

- Abstract Data Type (ADT)

แนวความคิดพื้นฐานของแอ็บสแทกต์ดาต้าไทป์ คือการแยกภาพการมองของผู้ใช้ออกจากรายละเอียดในการสร้าง ADT (Implement ADT) มีบทบาทต่อโปรแกรมประยุกต์ทางฐานข้อมูลโดยใช้ขยายความสามารถจัดการกับชนิดข้อมูลให้มีความซับซ้อนมากยิ่งขึ้น

3.2 ฐานข้อมูลเชิงวัตถุสัมพันธ์ (Object Relational Database: ORDB)

ระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ คือ ฐานข้อมูลชนิดที่นำแนวความคิดแบบ ออบเจ็กต์โอเรียนเต็ด (Object-Oriented Database) มาผสมผสานกับระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เพื่อทำให้ระบบฐานข้อมูลเชิงสัมพันธ์มีความสามารถทางด้านออปเจ็กต์ได้ เช่นการปกป้องข้อมูล การสืบทอดคุณสมบัติ และ โพลีมอร์ฟิซึม เป็นต้น โดยระบบฐานข้อมูลเชิงวัตถุสัมพันธ์จะมองข้อมูลและทำการจัดเก็บข้อมูลเป็นแบบตาราง โดยมีการจัดการฐานข้อมูลแบบเชิงสัมพันธ์ แต่การติดต่อกับผู้ใช้งานได้นำระบบของออบเจ็กต์โอเรียนเต็ดมาใช้งาน ซึ่งระบบออปเจ็กต์โอเรียนเต็ดจะมีการสนับสนุนการพัฒนาและมีการดูแลระบบฐานข้อมูลขนาดใหญ่ ๆ ได้ดีกว่า

ฐานข้อมูลเชิงวัตถุสัมพันธ์เสมือนการเพิ่มชั้นของออปเจ็กต์โอเรียนเต็ดครอบบนชั้นของระบบฐานข้อมูลสัมพันธ์ที่มีอยู่ และ ฐานข้อมูลเชิงวัตถุสัมพันธ์เป็นฐานข้อมูลเชิงสัมพันธ์ที่สามารถมีแอตทริบิวต์เป็นแอตทริบิวต์ที่สามารถแบ่งแยกย่อยได้อีก โดยภาพรวมแล้วเรายังมองเห็นฐานข้อมูลเป็นฐานข้อมูลเชิงสัมพันธ์ ในคอลัมน์สามารถเป็นได้ทั้ง ออบเจ็กต์หรือเป็นแอตทริบิวต์ที่มีหลายค่า เช่น ลิสต์ หรือเซต โดยทั่วไปโมเดลที่เป็นฐานข้อมูลเชิงวัตถุสัมพันธ์สามารถใช้งานเสมือนเป็น ฐานข้อมูลเชิงสัมพันธ์ ได้

3.3 คุณสมบัติของฐานข้อมูลเชิงวัตถุสัมพันธ์

1. มีความสามารถเดิมของฐานข้อมูลเชิงสัมพันธ์
2. สนับสนุนการสร้างออบเจ็กต์ที่มีความซับซ้อน
3. มีความยืดหยุ่นสูงยอมให้สร้างชนิดข้อมูล ฟังก์ชันและตัวดำเนินการ (Operator) ใหม่ได้
4. สามารถสร้างแอตทริบิวต์เป็นออบเจ็กต์หรือเป็นแอตทริบิวต์ที่สามารถแบ่งแยกย่อยได้อีกได้
5. มีคุณสมบัติแนวคิดเชิงวัตถุ เช่น ปกป้องข้อมูล การสืบทอดคุณสมบัติ และ โพลิมอร์ฟิซึม

3.4 การเปรียบเทียบระหว่างฐานข้อมูลเชิงสัมพันธ์กับฐานข้อมูลเชิงวัตถุสัมพันธ์

3.4.1 ชนิดข้อมูล

ในระบบฐานข้อมูลเชิงสัมพันธ์นั้น แอตทริบิวต์ของรีเลชันไม่สามารถแยกออก เป็นส่วน ๆ ได้อีกคือต้องเป็นแอตทริบิวต์ที่ไม่สามารถแบ่งแยกย่อย (Atomic Value Attribute)ได้อีก และมีชนิดของข้อมูลเป็นชนิดที่ค่อนข้างง่าย ได้แก่

- Integer
- Floating-point number
- Character string, fixed or variable length
- Date and time, time interval
- Numeric and decimal

การดำเนินการต่าง ๆ บนรีเลชันจำกัดอยู่ที่การเรียกดูข้อมูลและการแก้ไข และมีความสามารถจำกัดในการจัดการกับข้อมูลที่เป็นไบนารี่ (Binary) เช่น BLOB (Binary Large Object) จากขีดความสามารถที่จำกัดดังกล่าวส่งผลให้ต้องมีการพัฒนาโมเดลอื่นที่สามารถจัดการข้อมูลที่ซับซ้อนได้

ฐานข้อมูลเชิงวัตถุสัมพันธ์อนุญาตให้มีชนิดของข้อมูลเป็นแอตทริบิวต์ที่สามารถแบ่งแยกย่อยได้อีกหรือเป็นออบเจ็กต์ได้ดังนั้นชนิดของข้อมูลในฐานข้อมูลเชิงวัตถุสัมพันธ์สามารถเป็นข้อมูลที่มีความซับซ้อนกว่าในรีเลชันเช่นข้อมูลที่เป็นอิมเมจ (Image) และข้อมูลชนิดที่ใช้ในงานด้านมัลติมีเดียเป็นต้น ชนิดของข้อมูลที่เพิ่มเข้ามาในฐานข้อมูลเชิงสัมพันธ์ เช่น

- Row Type
- คอลเลคชั่นไทป์ (Collection Type)
- แอ็บสแทรกต์ดาต้าไทป์ (Abstract Data Type)

3.4.2 การควบคุมความถูกต้องของข้อมูล (Data Integrity)

ความสามารถในการทำการควบคุมความถูกต้อง (Integrity Constraint) ของฐานข้อมูลเชิงสัมพันธ์ยึดตามมาตรฐานของ SQL92 ในขณะที่ฐานข้อมูลเชิงวัตถุสัมพันธ์มีความสามารถตามที่กำหนดไว้ในมาตรฐาน SQL3 ดังนี้

- การกำหนดคีย์หลัก (Primary Key)
- Referential Integrity Constraint การกำหนด Foreign Key
- Attribute Constraint การกำหนด Not Null Constraint, Attribute-Based Check, Domain Constraint
- Global Constraint การกำหนด Tuple-Based Check และ Assertion
- Trigger เป็นแอ็กทีฟคอมโพเน็นต์ (Active Component) ที่จะเข้ามาทำแอ็กชัน (Action) เกิดเหตุการณ์ (Event) ตามที่กำหนดไว้

3.4.3 ภาษาที่ใช้

ฐานข้อมูลเชิงสัมพันธ์ใช้มาตรฐาน SQL92 ในการจัดการกับฐานข้อมูล ในขณะที่ฐานข้อมูลเชิงวัตถุสัมพันธ์ใช้ SQL3 ซึ่งเป็น SQL มาตรฐานใหม่ซึ่งเพิ่มความสามารถของแนวคิดเชิงวัตถุเข้าไป ในการสร้าง ADT และเพิ่มความสามารถของการจัดการออบเจ็กต์เข้าไปเช่น การปกป้องข้อมูล, การถ่ายทอดคุณสมบัติ และ โพลีมอร์ฟิซึม รวมถึงการเพิ่มการควบคุมความถูกต้อง ได้แก่ แอสเสิร์ทชั่น (Assertion) และ ทริกเกอร์ (Trigger)

ชนิดของ Statement ที่เพิ่มเติมใน SQL3

- ❑ New Statement เป็นสเตทเมนท์ที่ใช้ในการสร้าง ADT
- ❑ Destroy Statement เป็นสเตทเมนท์ที่ใช้ในการ Destroy ADT
- ❑ Assignment Statement เป็นสเตทเมนท์ที่นำ Result SQL ของ Value ไปเก็บไว้ใน Local Variable
- ❑ Call Statement เป็นสเตทเมนท์ที่ใช้เรียก SQL Procedure
- ❑ Return Statement เป็นสเตทเมนท์ที่ใช้ในการ Return ค่าจาก SQL Function

	Relational Database Management System (RDBMS)	Object Relational Database Management System (ORDBMS)
ชนิดข้อมูล Data Type	Simple Datatype มีชนิดของข้อมูลที่ค่อนข้างง่ายได้แก่ Integer Floating-point number Character string, fixed or variable length, Date and time, time interval และ Numeric and decimal	Complex Datatype เป็นข้อมูลที่มีความซับซ้อนมากกว่าแบบ รีเลชัน ซึ่งจะมีชนิดของข้อมูลเป็น แบบ Row type, Collection type (list, set, multiset) และ Abstract data type
ภาษา	ใช้มาตรฐาน SQL 92 ในการเก็บข้อมูล	ใช้มาตรฐาน SQL3 ในการเก็บข้อมูล ซึ่งเพิ่มความสามารถและแนวคิดเชิงวัตถุเข้าไป

ตารางที่ 3-1 สรุปตาราง เปรียบเทียบระหว่าง RDBMS และ ORDBMS

3.5 อินฟอร์มิกซ์ ไดนามิก เซิร์ฟเวอร์ (Informix Dynamic Server)

3.5.1 อะไรคือไดนามิกเซิร์ฟเวอร์

อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ เป็นดาต้าเบสเซิร์ฟเวอร์ ที่เป็นซอฟต์แวร์แพ็กเกจ (Software package) ที่จัดการการเข้าถึง (access) ฐานข้อมูลตั้งแต่ 1 หรือมากกว่านั้นสำหรับไคลเอนต์แอพพลิเคชั่น (Client applications) เดียวหรือหลาย ๆ ไคลเอนต์แอพพลิเคชั่น

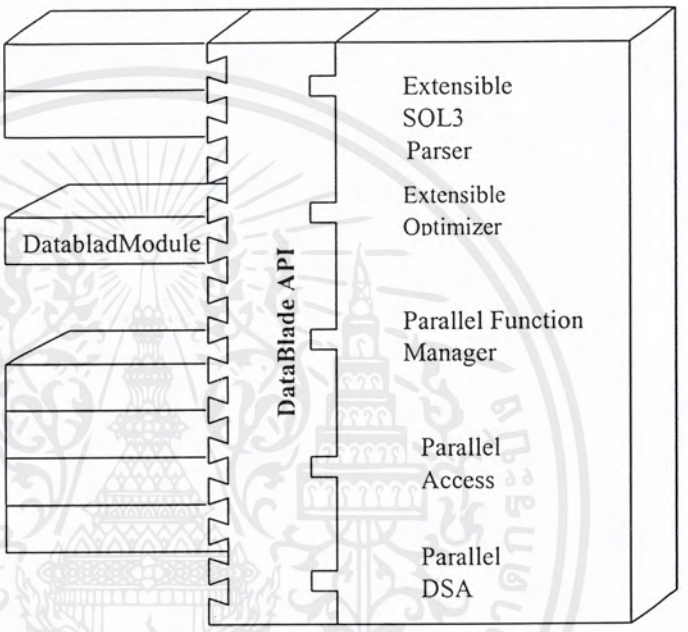
ลักษณะเฉพาะของไดนามิกเซิร์ฟเวอร์ จะเป็นมัลติเทรด (Multithreaded) คือจะมีการแบ่งงานออกเป็นงานย่อย ๆ เพื่อช่วยในการประมวลผลในออบเจ็กต์รีเลชันนัลดาต้าเบสเซิร์ฟเวอร์ (object-relational database server) ซึ่งจะคอยจัดการกับข้อมูลที่เก็บใน rows และ columns โดยจะใช้ ระบบซิงเคิลโพรเซสเซอร์ (Single processor) หรือ ระบบเอสเอ็มพี (Symmetric multiprocessing :SMP) และ สถาปัตยกรรมของดีเอสเอ (Dynamic Scalable Architecture) ซึ่งฐานข้อมูลจะได้ความสามารถขยายระบบได้อย่างต่อเนื่อง (Scalability) ความสามารถในการจัดการ และประสิทธิภาพที่เพิ่มขึ้นตามมา

อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ เป็นระบบฐานข้อมูลเชิงวัตถุสัมพันธ์ (ORDBMS) ที่มีการรวมความสามารถของระบบฐานข้อมูลเชิงวัตถุ (OODBMS) และระบบฐานข้อมูลเชิงสัมพันธ์ (RDBMS) โดยภาษาที่ใช้กับฐานข้อมูลเชิงวัตถุสัมพันธ์ได้มีการนำภาษา SQL3 มาใช้ และยังสามารถจัดการกับข้อมูลที่มีความซับซ้อนเช่น ภาพ 2 มิติ 3 มิติ (2D/3D Images), ข้อมูลเสียง (Sound), ข้อมูลวิดีโอ (Video), เอกสารทางอิเล็กทรอนิกส์ (Electronic Documents), เอกสาร

เอชทีเอ็มแอล (HTML Pages) เป็นต้น ซึ่งอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์จะใช้คาต้าเบลด ในการจัดการข้อมูลนั้นได้อย่างมีประสิทธิภาพ

3.6 สถาปัตยกรรมของอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์

ในอินฟอร์มิกซ์ไดนามิกส์เซิร์ฟเวอร์ ข้อมูลเกี่ยวกับชนิดของข้อมูล (Data type), ฟังก์ชัน และการ เข้าถึงเมทธอดจะเก็บไว้ในตารางแคตตาล็อก (Catalog Table) มากกว่าที่จะเก็บฮาร์ดโค้ด (Hard-coded) ไว้ในเซิร์ฟเวอร์



รูปที่ 3-1 สถาปัตยกรรมของอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์

สนับสนุนการทำงานแบบไคลเอนท์/เซิร์ฟเวอร์ (Client/Server) การติดต่อระหว่างโปรแกรมประยุกต์ไคลเอนท์กับคาต้าเบสเซิร์ฟเวอร์ โดยฝั่งเซิร์ฟเวอร์จะประกอบไปด้วย

- **Parser** ทำหน้าที่อ่านและตรวจสอบไวยากรณ์ โดยฟังก์ชันที่ผู้ใช้สร้างขึ้นมา (User-defined Function) และชนิดข้อมูลจะถูกบันทึกลงในแคตาล็อกเทเบิล (Catalog Table) และจะถูกตรวจสอบโดยพาร์เซอร์
- **Optimizer** ทำหน้าที่เลือกทางที่ดีที่สุดเพื่อการเข้าถึง (Access) ข้อมูลที่ถูกคิวรี (Query)
- **Function Manager** ทำหน้าที่หาและจัดการ User-defined Function เพราะว่าฟังก์ชันที่ผู้ใช้สร้างขึ้นมาจะถูกเก็บไว้ในแคตาล็อกเทเบิล
- **Access Method** ที่เหมาะสมจะถูกเลือกมาสำหรับข้อมูลที่กำลังถูกแก้ไข
- **Data Manager** ทำหน้าที่เคลื่อนย้ายข้อมูลเข้าออกจากดิสก์

3.7 จุดเด่นของโครงสร้างสถาปัตยกรรมของอินฟอร์มิกซ์ไดนามิกส์ เซิร์ฟเวอร์

3.7.1 การออกแบบมีประสิทธิภาพสูง (Performance)

การออกแบบอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ ใช้โครงสร้างเดียวกับระบบฐานข้อมูลแบบขนานแบบดีเอสเอ ซึ่งออกแบบให้สามารถใช้ได้ทั้งระบบคอมพิวเตอร์แบบโปรเซสเซอร์เดียวและแบบหลายโปรเซสเซอร์ และเป็นระบบที่สามารถใช้ประสิทธิภาพของฮาร์ดแวร์ได้อย่างมีประสิทธิภาพ และ อินฟอร์มิกซ์ไดนามิกส์เซิร์ฟเวอร์ยังสามารถปรับแต่งให้การทำงานของระบบที่บางครั้งอาจเป็นโพรเซสที่ซับซ้อนให้สามารถจัดการกับข้อมูลที่เกี่ยวข้องอย่างใกล้ชิด ทำให้การพัฒนาโปรแกรมทำได้ง่ายโดยไม่จำเป็นต้องค้นหาข้อมูลที่ซับซ้อนเอง

3.7.2 การออกแบบให้สามารถขยายได้ (Extensible)

ในระบบการจัดการฐานข้อมูลโดยทั่วไปจะจัดการกับข้อมูลที่เป็นตัวเลข ตัวอักษร แต่อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ออกแบบให้สามารถใช้งานอย่างไม่มีขีดจำกัดโดยสามารถที่จะใช้งานกับข้อมูลประเภทใดก็ได้ เช่น ภาพ วิดีโอ เสียง แกนเวลา สองมิติ สามมิติ ภาพ ข้อความ และข้อมูลที่ใช้กับเว็บ

3.7.3 ออกแบบให้เชื่อมต่อกันอย่างดี (Integrated)

การออกแบบที่รวบรวมข้อดีของระบบจัดการฐานข้อมูลแบบขนานและโครงสร้างข้อมูลแบบ ออบเจ็กต์โอเรียนเต็ดที่ทำให้ระบบสามารถขยายขีดความสามารถและนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ การรวมกันนี้ทำให้เป็นระบบจัดการฐานข้อมูลที่ทั้งสามารถขยายได้และเพิ่มความสามารถของระบบได้เป็นอย่างดี การที่สามารถรวมอยู่ในระบบเดียวกันได้นี้ทำให้ได้ประสิทธิภาพสูงสุด และลดความซ้ำซ้อนของระบบจัดการฐานข้อมูลและระบบงานได้ และยังคงไว้ด้วยความปลอดภัยและความเชื่อถือได้ของระบบ

3.7.4 ออกแบบที่นวัตกรรมใหม่ (Innovation)

การออกแบบอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ที่คำนึงถึงความเป็นระบบเปิด (Open System) ระบบที่ขยายได้โดยไม่มีขีดจำกัด (Unlimit Extensibility) และเพิ่มประสิทธิผลให้กับโปรแกรมเมอร์ทำให้สามารถสร้างระบบที่เป็นสากลต่าง ๆ เช่น มาตรฐาน SQL3 การเชื่อมต่อกับเน็ตสเคป หรือไมโครซอฟต์บราวซ์เซอร์ หรือการใช้งานในระบบขนาดเล็กจนถึงขนาดใหญ่ได้อย่างมีประสิทธิภาพ

3.8 ชนิดข้อมูลในอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์

ในระบบฐานข้อมูลของอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ ได้มีการแบ่งออกเป็น 3 ประเภทหลัก ๆ คือ

- ชนิดข้อมูลพื้นฐาน (Built-in data type) เป็นชนิดข้อมูลที่มีมากับอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ เช่น Character, Numeric, Time และ Large Object
- ชนิดข้อมูลที่ขยายเพิ่มขึ้นมา (Extended data types) ซึ่งจะแยกย่อยออกเป็น 2 ชนิด คือ
  1. ชนิดข้อมูลที่สร้างขึ้นใหม่ (User-defined types) เป็นชนิดข้อมูลที่ระบบฐานข้อมูลอนุญาตให้ ผู้ใช้สามารถที่จะสร้างชนิดข้อมูลขึ้นมาใช้เอง ซึ่งแบ่งเป็น 2 ชนิดย่อยอีก คือ

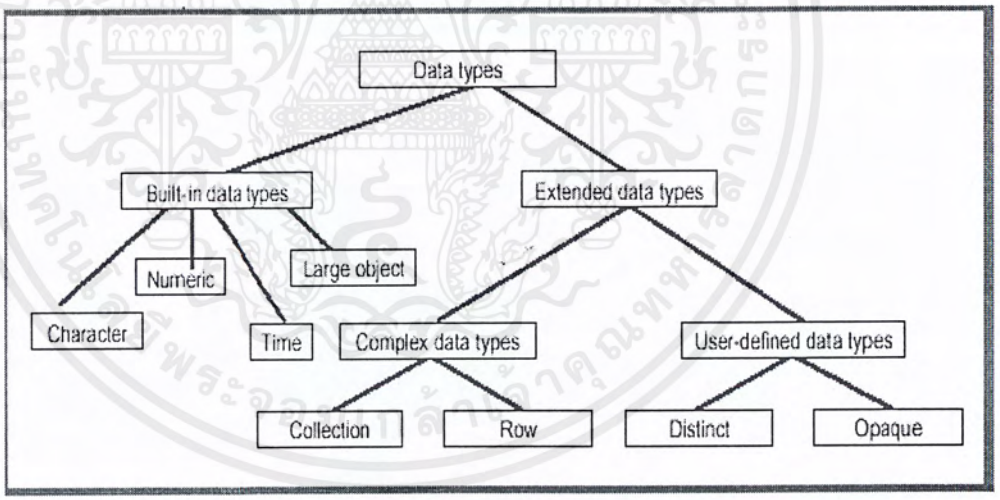
     1.1 Opaque type

     1.2 Distinct type
  2. ข้อมูลแบบที่ซับซ้อน (Complex data types) เป็นชนิดของข้อมูลที่มีการกำหนดไว้ในมาตร

     ฐาน SQL3 ซึ่งแบ่งเป็น 2 ชนิดย่อย คือ

     2.1 Collection type (List, Set, Multiset)

     2.2 Row types



รูป ที่ 3-2 แสดงถึง Data types

3.8.1 ข้อมูลชนิดพื้นฐาน (Built-in Data types)

เป็นชนิดข้อมูลที่มีลักษณะดังนี้

- เป็นชนิดข้อมูลพื้นฐานซึ่งดาต้าเบสเซิร์ฟเวอร์จัดให้มีไว้
- ไม่สามารถแบ่งย่อยออกเป็นส่วนเล็ก ๆ ต่อไปได้อีก (Atomic)
- บิวท์อินดาต้าไทพ์ของอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ มีดังนี้

Data types		Informix Dynamic Server
Character		CHAR,CHARACTER VARYING (VARCHAR), LVARCHAR
Numeric	Exact numeric	DECIMAL,MONEY,SMALLINT,INTEGER, INT8,SERIAL,SERIAL8
	Approximate numeric	SMALLFLOAT ,FLOAT
Large object	Simple large object	TEXT,BYTE
	Smart large object	CLOB,BLOB
Time		DATE,DATETIME,INTERVAL
Miscellaneous		BOOLEAN

ตารางที่ 3-2 แสดงตาราง Built-in data type ของ Informix Dynamic Server

3.8.2 ชนิดข้อมูลที่สร้างใหม่ (User defined type)

เป็นชนิดข้อมูลที่ผู้ใช้สร้างขึ้นมาใหม่นอกเหนือจากที่ทางอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ มีให้เพื่อความยืดหยุ่นในการเก็บข้อมูลและการจัดการโดยทางอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ ได้แบ่งข้อมูลชนิดนี้ออกเป็น 2 ชนิด คือ

3.8.2.1 Opaque type

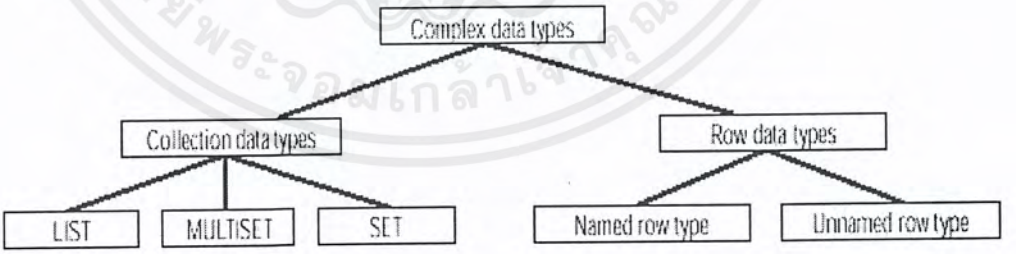
เป็นชนิดข้อมูลที่มีลักษณะซ่อนเร้นข้อมูลภายในทั้งหมด (Fully Encapsulated) ซึ่งเป็นข้อมูลที่ไม่สามารถจะแบ่งแยกย่อยได้ (Atomic Data type) ที่เราสามารถที่จะกำหนดหรือสร้างไว้ในระบบฐานข้อมูลโดยที่ระบบฐานข้อมูลจะไม่รู้ถึงลักษณะของโครงสร้างภายในของข้อมูลชนิดนี้ว่าเป็นแบบใดเหมือนกับข้อมูลที่มากับระบบฐานข้อมูล (Built-in Data type) ดังนั้นจึงจำเป็นที่จะต้องมีฟังก์ชันต่าง ๆ ที่คอยจัดการและดำเนินการกับ Opaque type Opaque type ใช้เก็บข้อมูลขนาดใหญ่ เช่น ข้อความ เสียง วิดีโอ เป็นต้น Opaque type จะสนับสนุนออบเจ็กต์โอเรียนเต็ดของข้อมูลไพรเวท (Private data)

3.8.2.2 Distinct type

เป็นชนิดของข้อมูลที่โครงสร้างภายในเหมือนกับชนิดข้อมูลที่มีอยู่แล้วโดยที่จะมีชื่อและฟังก์ชันที่ถูกสร้างมาแตกต่างไปจากข้อมูลที่เป็นต้นแบบ (Source type) ตัวอย่างเช่นเราสร้างข้อมูลชนิด decnum ที่มีลักษณะของโครงสร้างภายในเหมือนกับข้อมูลชนิด real ดังนั้นฟังก์ชันทั้งหมดที่ใช้กับ real จะสามารถที่จะทำงานกับ decnum ได้ แต่อย่างไรก็ตามเราไม่สามารถที่จะทำการบวก ลบ หรือเปรียบเทียบข้อมูลระหว่าง decnum กับ real ได้ จำเป็นจะต้องทำผ่านฟังก์ชันที่มีหน้าที่แปลงชนิดข้อมูล(Casting function เป็นฟังก์ชันที่ทำการ Cast (เป็น mechanism ที่ทำการแปลงค่าจากข้อมูลชนิดหนึ่งไปเป็นข้อมูลอีกชนิดหนึ่ง) ซึ่งเราสามารถที่จะสร้างขึ้นมาเองได้นอกเหนือจากที่ระบบฐานข้อมูลมีให้) เพื่อทำการแปลงชนิดของข้อมูลก่อน

3.8.2.3 Complex types

เป็นข้อมูลที่ผู้ใช้สร้างที่สามารถจะประกอบไปด้วยข้อมูลชนิดต่าง ๆ ซึ่งคุณสมบัติที่สำคัญของชนิดข้อมูลนี้ก็คือการที่ง่ายที่จะเข้าถึงแต่ละส่วน (Component) ของข้อมูลที่ไม่เหมือนกับข้อมูลชนิด Built-in หรือแบบ Opaque ที่มีลักษณะซ่อนเร้นกล่าวคือถ้าเราจะเข้าถึงข้อมูลของ Opaque ก็จะต้องทำผ่านฟังก์ชันที่กำหนดไว้กับ Opaque



รูปที่ 3-3 Complex types ที่มีอยู่ใน Informix Dynamic Server

จากรูปที่ 3-3 จะเห็นว่าข้อมูลแบบ Complex ประกอบด้วย

- **ข้อมูลแบบ Collection**: เป็นกลุ่มของข้อมูลที่มีชนิดข้อมูลเดียวกัน โดยอนุญาตให้เราสามารถที่จะเก็บและจัดการกับ Collection ของข้อมูลภายใน Row เดียว ข้อมูลแบบ Collection ประกอบ 2 ส่วนคือ Type constructor และ Element type โดยที่

  1. Type constructor ประกอบด้วย set, multiset และ list
     - SET : เป็นข้อมูลแบบ Collection ที่ Elements ไม่มีลำดับและไม่สามารถที่จะมีข้อมูลซ้ำกันได้ภายใน Set ซึ่งในการกำหนด Set จะต้องใช้ Set constructor
     - LIST : เป็นข้อมูลแบบ Collection ที่ Elements มีลำดับและ Element สามารถที่จะซ้ำกันได้ ในการกำหนด List จะต้องใช้ List constructor
     - MULTISET : เป็นข้อมูลแบบ Collection ที่ไม่มีลำดับของ Element และอนุญาตให้ข้อมูลซ้ำกันได้ ในการกำหนด Multiset จะต้องใช้ Multiset constructor
  2. Element type เป็นการระบุชนิดของข้อมูลของ Element ซึ่ง Element สามารถที่จะมีชนิดใดก็ ได้นอกจาก SERIAL และ SERIAL8

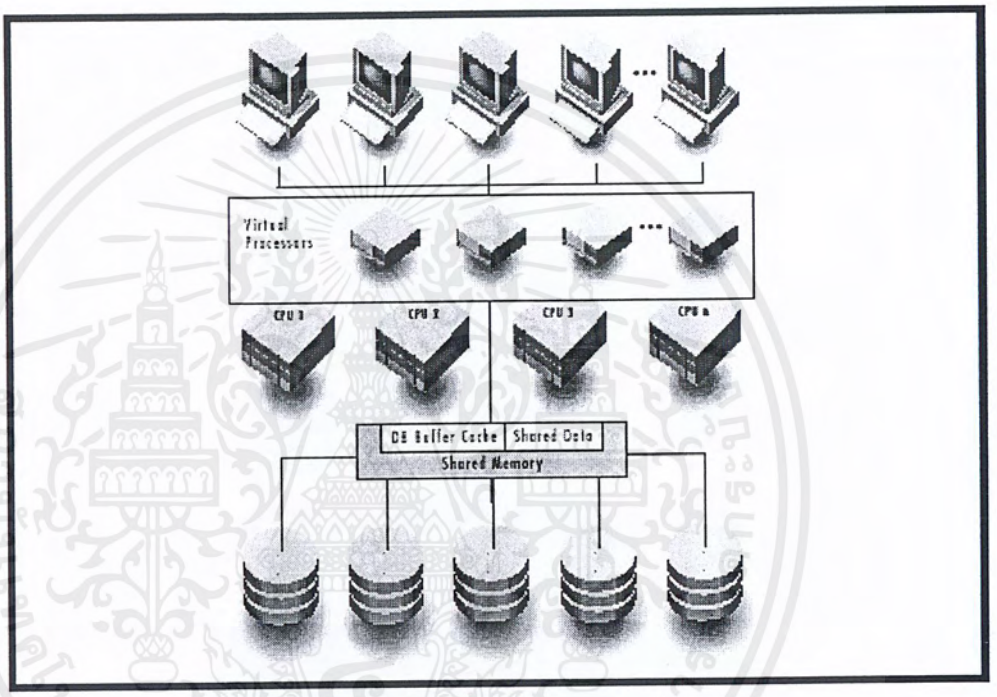
- **Row types** เป็นชนิดข้อมูลที่มีลักษณะคล้ายกับข้อมูลแบบโครงสร้าง (Structure) ในภาษา C โดยภายในจะประกอบด้วยฟิลด์ (Fields) แต่ละฟิลด์จะมีชื่อและชนิดข้อมูลเป็นของตัวเองซึ่งฟิลด์ของ Row type ก็เปรียบได้กันคอลัมน์ของตารางนั้นเองซึ่งจะเห็นว่า Row type ของอินฟอร์มิกซ์ไดนามิกส์เซิร์ฟเวอร์ ก็เป็นชนิดของข้อมูลชนิดเดียวกับ Row type ที่มีอยู่มาตรฐานของ SQL3 โดยที่ Row type ใน อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ นั้นมีอยู่ด้วย 2 แบบคือ named row type และ Unnamed row type

  1. Named row type : เป็นกลุ่มของฟิลด์ ที่การกำหนดภายใต้ชื่อชื่อหนึ่ง โดยฟิลด์นี้หมายถึงคอมโพเน็นต์ของ Row type โดยในการสร้าง named row type จะต้องมีชื่อไม่ซ้ำกับ named row type ที่มีในฐานข้อมูลอยู่ก่อนแล้ว เราจะมีการสร้าง named row type ก็ต่อเมื่อเราต้องการที่จะเข้าถึงคอมโพเนนท์ของข้อมูลได้เช่นเราอาจจะสร้าง named row type ที่เก็บ address ซึ่งสามารถที่จะเข้าถึงข้อมูลแต่ละคอมโพเน็นต์ ของ address ได้เช่น ถนน เมือง จังหวัด เป็นต้น
  2. Unnamed row type : เป็นกลุ่มของฟิลด์ที่กำหนดโดยใช้โครงสร้างของมันเองซึ่งไม่เหมือนกับ named row type เราสามารถที่จะกำหนด unnamed row type ให้กับตารางและใช้ในการกำหนดชนิดให้กับคอลัมน์ได้

3.9 อินฟอร์มิกซ์ดาต้าเบสเซิร์ฟเวอร์

ภายในสถาปัตยกรรมนั้นจะประกอบไปด้วยการทำ

- Shared momory
- Disk
- Virtual processor



รูป 3-4 อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์

จากรูป 3-4 อินฟอร์มิกไดนามิกเซิร์ฟเวอร์ประกอบด้วย ความสามารถในการปรับแต่งค่าจากกรรมวิธีการเข้ามารวมกันของดาต้าเบสเซิร์ฟเวอร์ ซึ่งเรียกว่าโพรเซสเซอร์เสมือน (Virtual processors) โดยสามารถตอบสนองความต้องการร้องขอจากหลาย ๆ ไคลเอนต์ได้

ที่ฝั่งไคลเอนท์จะสร้างลอจิกคอนเน็กชัน (Logical Connection) กับเซิร์ฟเวอร์โดยมีชนิดของการ เชื่อมต่อ ดังนี้

- **Local Connection** ได้แก่ Shared Memory และ Stream Pipe
- **Network Connection** ได้แก่ Computer-to-Computer และ Local Loopback

Interface	Network Protocol
Socket	TCP/IP
TLI	TCP/IP
TLI	SPX/IPX

ตารางที่ 3-3 ตาราง ชนิดของอินเทอร์เฟซกับโพรโตคอลที่ไดนามิกเซิร์ฟเวอร์สนับสนุน

3.9.1 ส่วนประกอบอื่น ๆ ที่จำเป็นต่อการใช้งานต่าง ๆ ดังนี้

Connectivity File: SQLhosts เป็นไฟล์ที่เก็บข้อมูลที่ไคลเอนท์ใช้ในการเชื่อมต่อกับเซิร์ฟเวอร์ ชื่อของดาต้าเบสเซิร์ฟเวอร์, ชนิดของอินเตอร์เฟซและโพรโตคอล, ชื่อของโฮสต์, Servicename Field, Options Field

Onconfig Parameters สำหรับการเชื่อมต่อ เป็นคอนฟิกูเรชั่นไฟล์ (Configuration File) อยู่ใน $INFORMIXDIR/etc Directory ระหว่างการติดตั้งเก็บข้อมูลที่ใช้ในการกำหนดค่าเริ่มต้น (Initial Setting) และ พารามิเตอร์ในการคอนฟิก (Configuration Parameter)

Utility Enhancement ตัวอย่างของยูทิลิตี้ที่ใช้งาน

- Oninit Initialize Disk Space ต้องล็อกอิน (Log In) เป็นรูท (Root) หรือยูสเซอร์อินฟอร์มิกซ์ (User Informix) เพื่อที่จะเรียกใช้ oninit
- Onmode ต้องล็อกอินเป็นรูท หรือ ยูสเซอร์อินฟอร์มิกซ์ onmode จะทำการเปลี่ยนโหมดในการทำงานของไดนามิกเซิร์ฟเวอร์ (Dynamic Server Operating Mode)
- Onspace ต้องล็อกอินเป็นรูท หรือ ยูสเซอร์อินฟอร์มิกซ์ โดยจะต้องทำการกำหนด Create/Drop Sbspace หรือ Dbspace, Add/Drop Chunk onmode กับ onspace ต้องกำหนดก่อนที่จะทำยูทิลิตี้อื่นๆ

3.9.2 SetNet32

ด้วย SetNet32 Utility สามารถกำหนดหรือแก้ไข Environment Variable และ Network Parameter ขณะรันไทม์ (Runtime) Setnet32 เป็นยูทิลิตี้ที่ติดตั้งบนไคลเอนท์พีซี, Setnet32 Utility มี 4 ส่วนที่สำคัญคือ

- Environment Enable ใช้กำหนด Environment Variables
- Server Information ใช้กำหนดข้อมูลของดาต้าเบสเซิร์ฟเวอร์
- Host Information ใช้กำหนดคอนฟิกของโฮสต์แมชชีน (Host Machine)
- About Setnet32 ใช้แสดงข้อมูล Setnet32 Utility

3.9.3 Schema Knowledge

เป็นทูลที่แสดงข้อมูลต่าง ๆ ในรูปแบบของกราฟิก (Graphical Knowledge Tool) ใช้แสดง เมต้าดาต้า (Meta Data) ของฐานข้อมูล ไม่สามารถใช้แก้ไขข้อมูลใด ๆ บนฐานข้อมูลได้ เราสามารถใช้ Schema Knowledge ดูเมต้าดาต้าต่อไปนี้ได้

- Database
- Routine
- Table และ View
- Row Type
- Inheritance View

ก่อนจะเข้ามาใช้ต้องมีการล็อกอิน ก่อนโดยจะต้องทำการใส่ ชื่อเซิร์ฟเวอร์,ชื่อฐานข้อมูล, ชื่อผู้ใช้ และรหัสผ่าน

3.9.4 SQLEditor

เป็นทูลที่แสดงผลเป็นกราฟิก (Graphical Tool) ซึ่งใช้ทำการติดต่อกับไดนามิกเซิร์ฟเวอร์ และทำการคิวรีข้อมูลจากฐานข้อมูล (Query Database Information) และดูผลของการคิวรีของ SQL Statement ต้องมีการกำหนดค่าต่างๆใน SetNet32

3.9.5 DBDK: DataBlade Developer Kit

เป็นทูลที่ใช้พัฒนาดาต้าเบลดโมดูล(Datablade Module) มีรายละเอียดอยู่ในส่วนถัดไป

3.9.6 Iconnect

เป็น ODBC ของไดนามิกเซิร์ฟเวอร์

3.10 คุณสมบัติของไดนามิกเซิร์ฟเวอร์ (Dynamic Server)

- มีความสามารถในการทำ Server Instance Manager โดยตัวนี้จะเป็นตัวจัดการเกี่ยวกับการ Create Database และทำการ Delete Database โดย ไดนามิกเซิร์ฟเวอร์นั้น สามารถสร้าง ดาต้าเบสเซิร์ฟเวอร์ ขึ้นมาได้หลายตัว แต่ ยูนิเวอร์แซลเซิร์ฟเวอร์ สามารถมีดาต้าเบสเซิร์ฟเวอร์ได้เพียง เซิร์ฟเวอร์ เดียว
- มีความสามารถสูง (High Performance) สนับสนุนการทำงานที่เป็น Raw Disk Management , Memory Management (Dynamic Sharing Memory, Buffering Transaction) , Dynamic Thread Allocation และ Parallelization
- มีความสามารถในการจัดการ Logging-Recovery เพื่อทำการควบคุมความถูกต้องของฐานข้อมูล (Integrity Consistency) เมื่อมีเดียชำรุด (Media Failed)
- สนับสนุนการทำ Fast Recovery, Mirroring, Data Replication
- Distributed Database สนับสนุนสถาปัตยกรรมในแบบกระจาย (Distributed)
- ไดนามิกเซิร์ฟเวอร์สนับสนุนการใช้งานหลายภาษา (Global language) โดยมี Global Language Support Function (GLS) สนับสนุนการทำงานหลายภาษา

3.11 ค่ารหัสเฉพาะ (Object Identification: Oid)

ค่ารหัสเฉพาะ Object Identification (Oid) คือจะเป็นค่าที่มีลักษณะเฉพาะที่มีลักษณะเป็นค่า ๆ เดียวไม่ซ้ำซ้อน เป็นเลขจำนวนเต็มบวก และมีคุณลักษณะในการเพิ่มค่าเองขึ้นโดยอัตโนมัติได้โดยจะเพิ่มจากน้อยไปมากซึ่งตัว DBMS จะเป็นตัวกำเนิด (Generate) ให้เองโดยอัตโนมัติ ซึ่งค่า Oid ที่ออกไปแล้วนั้นไม่สามารถนำกลับมาใช้ใหม่ได้ ค่า Oid นั้นจะมีอยู่ในฐานข้อมูลเชิงวัตถุสัมพันธ์ และใน Informix ก็คือ Serial Data type นั่นเอง

เนื่องจากฐานข้อมูลแบบเดิมเป็น ฐานข้อมูลเชิงสัมพันธ์ซึ่งมี Primary key เป็น External Identify เมื่อมีการเปลี่ยนแปลง key ทำให้ Identify เปลี่ยนแปลงได้ แต่สำหรับฐานข้อมูล Informix ซึ่งเป็นฐานข้อมูลเชิงวัตถุสัมพันธ์ สามารถใช้ Internal Identify เป็น key เมื่อมีการเปลี่ยนแปลงข้อมูลใด ๆ ไม่มีผลต่อ Identify ทำให้สืบค้นข้อมูลได้ง่ายกว่าวิธีเดิม
โดย ชนิดของข้อมูลของ Informix ที่เป็น Internal Identify ได้แก่ ข้อมูลชนิด Serial(n) ซึ่งมีคอลัมน์ ที่เป็น Serial มีคุณลักษณะดังต่อไปนี้

- เป็นตัวเลขจำนวนเต็มบวก
- ไม่ซ้ำซ้อน (UNIQUE)
- ป้อนค่าให้เองโดยอัตโนมัติ ซึ่งเรียงลำดับจากน้อยไปมาก

ข้อจำกัดของ คอลัมน์ ที่เป็น Serial มีดังต่อไปนี้

1. ในตารางหนึ่ง ๆ สามารถมี คอลัมน์ ที่เป็น Serial ได้เพียง คอลัมน์ เดียว
2. ชนิดของข้อมูลที่เป็น Serial ไม่ได้เป็นคอลัมน์ที่ไม่ซ้ำ (Unique column) โดยอัตโนมัติ ต้องทำการกำหนด unique index ไปยัง คอลัมน์ นั้น เพื่อป้องกันการซ้ำกันของ Serial number ถ้าสร้างตารางโดยใช้ Interactive Schema Editor (เช่น SQL Editor ของ Informix) จะมีการกำหนด unique index ไปยัง Serial column นั้น โดยอัตโนมัติ

3.11.1 การกำหนดค่าเริ่มต้นให้กับชนิดของข้อมูล Serial

ค่าเริ่มต้นของข้อมูลชนิด Serial นั้นสามารถกำหนดโดยผู้ใช้เองได้ เช่น ถ้าให้ค่าเริ่มต้นที่ 100 หมายความว่าตัว DBMS ของ Informix จะทำการนับค่าที่เริ่มต้นจริงเป็น 101 ถ้าเราไม่กำหนดค่าเริ่มต้นให้กับโปรแกรม แต่ถ้าเราต้องการให้ตัว DBMS เป็นตัว Count ให้อัตโนมัติ จะต้องให้ค่าเวลา insert เป็น 0 ซึ่งค่าเริ่มต้นของ คอลัมน์ จะมีค่าเริ่มที่ 1, 2, ...ไปเรื่อยๆจากนั้นก็จะเพิ่มค่าขึ้นไปเรื่อย ๆ ทีละ 1 เมื่อมีการ insert โดยค่าสูงสุดของ ข้อมูลชนิด Serial มีค่าเท่ากับ 2,147,483,647 ถ้ามากกว่านี้จะเกิดความผิดพลาด (Syntax Error) หากมีการกำหนดค่าให้กับ Serial Column (โดยใช้การ Insert) โดยที่ค่านั้นไม่ซ้ำกับค่าเดิมที่อยู่ใน Serial Column ของตารางดาต้าเบสเซิร์ฟเวอร์ จะกำหนดค่าถัดไปในลำดับให้กับ Serial Column ของแถวนั้น

3.11.2 การใช้ Serial กับ Integer

ดาต้าเบสเซิร์ฟเวอร์ จะมองว่าข้อมูลชนิด Serial เป็นข้อมูล Integer ชนิดพิเศษ ดังนั้น การกระทำทางคณิตศาสตร์ที่ข้อมูลชนิด Integer สามารถทำได้ (+, -, *, /) ก็จะสามารถทำได้ในข้อมูลชนิด Serial เช่นเดียวกัน

3.11.3 ข้อมูลชนิด Serial8(n)

ถ้าหากค่าใน Serial Column มีค่ามากกว่า 2,147,483,647 หากใช้ข้อมูลชนิด Serial จะเกิดความผิดพลาด (Syntax Error) ซึ่งจะต้องใช้ข้อมูลชนิด Serial8 โดยจะมีการเก็บเป็นข้อมูลขนาด 8 ไบต์ มีค่าสูงสุดเท่ากับ 2^{63}-1 หรือเท่ากับ 9,223,372,036,854,775,807 โดยมีข้อจำกัดเหมือนกับข้อมูลชนิด Serial

3.12 หลักการนำข้อมูลชนิด Serial มาใช้งาน

ค่าข้อมูลชนิด Serial นั้น เป็นค่าที่เมื่อนำมาใช้ประกาศตัวแปรเราจะใช้เป็น Primary Key เพราะค่า Serial เมื่อรันแล้วจะเริ่ม นับเพิ่มไปทีละ 1 ไป เรื่อย ๆ และ ค่าข้อมูล serial ที่รันออกไปแล้วจะไม่สามารถนำกลับมาใช้ได้อีก ถ้าค่าไหนทำการลบออกจาก Tuple ของตารางนั้นแล้วก็จะหายไปเลย เพราะค่าตัวแปร Serial จะรันไป เรื่อย ๆ ค่าข้อมูลชนิด Serial นั้น เราจะเริ่มต้นที่เท่าไรก็ได้ อยู่ที่ ผู้เขียนโปรแกรมเป็นคนกำหนด แต่การเริ่ม Insert ข้อมูล ลง List นั้น จะต้องให้ค่า เริ่มต้น ของตัวแปร Serial เริ่มจาก 0 เท่านั้น

ตัวอย่างการใช้งานข้อมูลชนิด Serial

```
CREATE TABLE STUDENT
(Std_serial        SerialI(100),
StdID List(Row(StdID_t CHAR(8), TimeStart DATE,TimeEnd DATE) NOT NULL),
FirstName List(Row(FirstName_t CHAR(15), TimeStart DATE,TimeEnd DATE) NOT
NULL),
LastName List(Row(LastName_t CHAR(15), TimeStart DATE,TimeEnd DATE) NOT
NULL)
);

INSERT INTO STUDENT VALUES
(0,
"list{row('41013533','1/20/1999',NULL)}",
"list{row('JIM','1/02/1999','1/25/1999'),row('JOHN','1/25/1999',NULL)}",
"list{row('HENRY','1/20/1999','1/25/1999'),row('DOHATHY','1/5/1999',NULL)}"
);
```

บทที่ 4
ฐานข้อมูลเชิงวัตถุสัมพันธ์ที่ขึ้นอยู่กับเวลา
(Temporal Object Relational Database)

4.1 ฐานข้อมูลเชิงสัมพันธ์ที่ขึ้นอยู่กับเวลา (Temporal Relational Database)

ฐานข้อมูลเชิงเวลาที่มีการแปะเวลา (Time stamping) ไว้ที่ Tuple ของตาราง นั้น จะมีลักษณะการนำเสนอที่เป็น Temporal Relational Database (TRDB) คือ มีลักษณะการนำเสนอที่เป็น Flat table แต่ได้เพิ่มส่วนของการแปะเวลาไว้ที่แอตทริบิวต์ ส่วนท้ายของตาราง โดยจะมีการสร้างคอลัมน์ใหม่ขึ้นมา 2 คอลัมน์ คือเป็น คอลัมน์ของเวลาเริ่มต้น Valid time start (Vs) และคอลัมน์ของเวลาสิ้นสุด Valid time end (Ve) เพื่อบันทึกว่า Fact ของข้อมูลใน Tuple นั้นเคยเป็นจริงในช่วงเวลาใด จึงทำให้สามารถเรียกค้นค่าข้อมูลใน State เก่า ๆ ได้ ดังรูปที่ 4-1

รหัสนักศึกษา	ชื่อ	นามสกุล	ภาควิชา	เวลาเริ่มต้น	เวลาสิ้นสุด
41013524	จิราเจตน์	อ่อนสา	คอมพิวเตอร์	03/01/1985	01/01/1991
41013524	จิรายุ	อ่อนสา	คอมพิวเตอร์	01/01/1991	12/31/993
41013524	จิรายุ	อ่อนสา	โทรคมนาคม	12/31/993	Now
41013539	ประยงค์	ทานนท์	คอมพิวเตอร์	02/01/1988	10/31/1995
41013533	ธีรวุธ	สันติสุธีรวุฒิ	คอมพิวเตอร์	04/01/1991	Now
41013549	วิเชษฐ์	เจอตี้	คอมพิวเตอร์	06/01/1988	Now

ตารางที่ 4-1 แสดงตารางในระบบฐานข้อมูลเชิงเวลาที่ทำการแปะเวลาไว้ที่ Tuple ของตาราง

ตารางวาลิดไทม์ (Valid-time table) ข้างต้นเก็บประวัติของของนักศึกษาจะเห็นว่าระหว่างวันที่ 1 เดือน มีนาคม ปี ค.ศ. 1985 ถึงวันที่ 1 เดือน มกราคม ปี ค.ศ. 1991 นักศึกษาชื่อจิราเจตน์ยังคงใช้ชื่อว่าจิราเจตน์ อยู่ในภาควิชาคอมพิวเตอร์จากนั้นได้ทำการเปลี่ยนชื่อไปเป็น จิรายุโดยยังคงอยู่ที่ภาควิชาเดิมคือคอมพิวเตอร์ และในวันที่ 31 เดือน ธันวาคม ปี ค.ศ. 1993 จิรายุได้ทำการย้ายภาควิชาจากคอมพิวเตอร์ไปเป็นโทรคมนาคม สังเกตว่าหากมีการเปลี่ยนแปลงข้อมูลภายใน Tuple แม้แต่เพียงแอตทริบิวต์เดียวเราจะต้องทำการ Create Row ใหม่ขึ้นมา แล้วทำการแปะเวลาเริ่มต้นและเวลาสิ้นสุดเข้าไปที่ Tuple ของตารางเพื่อเป็นการบอกว่าข้อมูลที่ได้ถูกเปลี่ยนแปลงนั้นเคยเป็นจริง (Valid) เมื่อใด และในส่วนของข้อมูลที่เป็นข้อมูลปัจจุบัน (Current) นั้นเราใส่เวลาสิ้นสุดไว้ให้เป็น Now เพราะเนื่องจากเรายังไม่ทราบเวลาสิ้นสุด และยังเป็นการบอกว่าข้อมูลใน Tuple นั้นเป็นข้อมูลใน State ล่าสุด และในส่วนของ Row ที่มีการใส่ทั้งเวลาเริ่ม

ต้นและเวลาสิ้นสุดเอาไว้นั้น จะสื่อถึงว่าข้อมูลนั้นได้เคย Valid มาแล้วในช่วงเวลานั้น เช่นที่ ประยงค์ ทานนท์ มีเวลาที่ Valid ตั้งแต่วันที่ 1 เดือน กุมภาพันธ์ ปี ค.ศ. 1988 ถึง วันที่ 31 เดือน ตุลาคม ปี ค.ศ. 1998 อาจจะหมายความว่า ประยงค์เคยเป็นนักศึกษาในช่วงเวลาดังกล่าวข้างต้น ซึ่งปัจจุบัน ประยงค์ อาจจะลาออกจากการเป็นนักศึกษาแล้วจึงได้มีการแปะเวลาล่าสุดเข้าไว้ที่ Tuple ของตาราง เนื่องหลักการจากฐานข้อมูลเชิงเวลาจะไม่การลบข้อมูลออกไปจากฐานข้อมูล จึงทำให้สามารถเรียกค้นค่าข้อมูลใน State เก่า ๆ ได้ ซึ่งเป็นข้อดีอย่างหนึ่งของฐานข้อมูลเชิงเวลา แต่เราอาจจะเสียเรื่อง Space ไปเพราะว่าฐานข้อมูลเราจะมีขนาดใหญ่ขึ้นเรื่อย ๆ และการแปะเวลาไว้ที่ Tuple นั้นจะไม่ทราบเลยว่าคอลัมน์ไหนได้มีการเปลี่ยนแปลง

4.2 ฐานข้อมูลเชิงวัตถุสัมพันธ์ที่ขึ้นกับเวลา (Temporal Object Relational Database)

โมเดลของระบบฐานข้อมูลที่มีใช้กันอยู่ในปัจจุบันเป็นฐานข้อมูลเชิงสัมพันธ์ (Relational Database) แต่ฐานข้อมูลเชิงสัมพันธ์มีข้อจำกัดหลายด้าน เช่นการใช้งานกับชนิดข้อมูล จะมีแต่ชนิดข้อมูลที่ง่ายๆ เช่น Integer Character และString ซึ่งไม่อาจรองรับการใช้งานฐานข้อมูลที่มีความซับซ้อนได้ และก็ไม่สามารถจัดการกับฐานข้อมูลที่เป็นออบเจ็กต์ ได้เพราะฐานข้อมูลเชิงสัมพันธ์ แสดงข้อมูลได้เพียงตาราง จึงได้นำเสนอแนวความคิดของออบเจ็กต์ มาใช้เป็น Object Relational ฐานข้อมูลเชิงวัตถุสัมพันธ์ได้แนวความคิดมาจากโมเดลฐานข้อมูลเชิงสัมพันธ์ Relational Database และแนวความคิดเชิงวัตถุ Object Oriented Concept ซึ่งทำให้ฐานข้อมูลเชิงวัตถุสัมพันธ์ มีความสามารถเพิ่มมากขึ้นคือทำให้การจัดการข้อมูลได้มีประสิทธิภาพ มีความสามารถในการเข้าถึงภาษา SQL และความสามารถในแนวคิดเชิงวัตถุซึ่งได้แก่ การปกป้องข้อมูล Encapsulation การถ่ายทอดคุณลักษณะ Inheritance และ โพลีมอร์ฟีซึม Polymorphism โดยฐานข้อมูลเชิงวัตถุสัมพันธ์ ยังคงมองเห็นข้อมูลเป็นตารางในแบบเดียวกับฐานข้อมูลเชิงสัมพันธ์ และยังสามารถมีแอตทริบิวต์ที่สามารถแยกย่อยได้ (Non–atomic attribute) ซึ่ง Object Relational Database ยังคงใช้งานเสมือน Relational Database ได้

จากคุณลักษณะของฐานข้อมูลเชิงวัตถุสัมพันธ์ ที่มีแนวคิดเชิงวัตถุมาใช้โดยจะมองแต่ละคอลัมน์เป็นออบเจ็กต์ หรือเป็นคอลัมน์ที่เป็นแอตทริบิวต์ที่มีหลายค่า (Collection Data Type) โดยอาจจะมีอยู่แล้วหรือเป็นข้อมูลที่สร้างขึ้น เช่น อาจเป็น Row type ทำให้มีความสามารถในการเก็บข้อมูล ที่มีการเปลี่ยนแปลงตามเวลาไว้ในคอลัมน์เดียวได้ เสมือนหนึ่งว่าเอาแอตทริบิวต์ของช่วงเวลา (Timestamp) ไปเกาะติดกับแอตทริบิวต์ของข้อมูลนั้น ในการนำเวลาไปแปะข้อมูลไว้ที่ คอลัมน์นั้นเราจะต้องนึกถึงปัญหา External Identifier เนื่องจากมีการเปลี่ยน Key แล้วจะทำให้ identify เปลี่ยนแปลงได้

ปัญหา External Identifier เนื่องจากเปลี่ยนคีย์

1. ปัญหาเรื่อง Identifier จะพบว่าถ้าใช้ข้อมูลเชิงเวลาในฐานข้อมูลเชิงสัมพันธ์นั้น จะมีปัญหาเรื่อง Primary Key และการอ้างถึง Foreign Key จะทำให้ยุ่งยากเพราะต้องรวมเอา Valid time เข้ากับ Primary Key ด้วยเนื่องจากในฐานข้อมูลเชิงเวลาเก็บทุก ๆ ค่า ความจริงที่เปลี่ยนแปลงไปเมื่อเวลาเปลี่ยนไป

2. เนื่องจากใน First Normal Form ของฐานข้อมูลเชิงเวลาจะกำหนดให้เพียงแอตทริเดียวเท่านั้นที่เปลี่ยนแปลงตาม Valid time ได้ แต่ในการใช้งานจริง ๆ นั้นจำเป็นต้องเปลี่ยนแปลงมากกว่า 1 แอตทริบิวต์ในตาราง เช่นในทะเบียนนักศึกษา อาจจะมีการเปลี่ยนชื่อ เปลี่ยน นามสกุล หรือ ย้ายที่อยู่เกิดขึ้นก็ได้

จากปัญหาทั้ง 2 ข้อที่กล่าวมาข้างต้น สามารถแก้ได้โดยใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์

- เรื่อง Identifier ในระบบการจัดการฐานข้อมูล (DBMS) ของ Informix จะมี Oid ซึ่งเป็นข้อมูลที่เป็นเอกลักษณ์ (Unique) ภายในตาราง ซึ่งนำมาใช้เป็น Identifier ได้
- ส่วนปัญหา First Normal Form ในฐานข้อมูลเชิงเวลาแก้ได้โดย เอา Valid time ไปเกาะติดแต่ละแอตทริบิวต์ที่สามารถเปลี่ยนแปลงได้ตามเวลา โดยจะกำหนดให้ แต่ละคอลัมน์เดิมเป็นกลุ่มของข้อมูลที่มี Valid time กำกับไว้ดังรูปที่ 4-2

EMP	Dept	Ts	Te
Bill	Shipping	1	4
Bill	Loading	5	10

ตารางที่ 4-2 รูปแสดงการเอาตารางที่มี Valid-time มาแปะไว้ที่แอตริบิวต์

จากข้อจำกัดของ TRDB ถ้าในกรณีที่มีการถามว่าข้อมูลใน คอลัมน์ ไหนเปลี่ยนและเปลี่ยนแปลงเวลาไหนบ้างการแปะเวลาไว้ที่ Tuple ไม่สามารถตอบคำถามนี้ได้เลย หรือในกรณีที่มีการเปลี่ยนแปลงข้อมูลทีละหลาย ๆ คอลัมน์พร้อม ๆ กัน ก็ไม่สามารถรู้ได้เช่นกัน ในส่วนของโคลงสร้างข้อมูลในการจัดเก็บนั้นแบบรีเลชันนัลธรรมดา ซึ่งยังไม่สนับสนุนการจัดเก็บข้อมูลที่ซับซ้อนได้ คือไม่สามารถที่จะจัดเก็บข้อมูลในลักษณะคอลัมน์ที่เป็นแอตทริบิวต์ที่มีหลายค่า (Collection Data Type) เช่น เซต (Set) หรือ ลิสต์ (List) เป็นต้น จึงได้มีการนำเสนอ การจัดเก็บแบบใหม่เป็น Temporal Object Relational Database (TORDB) โดยจะดึงเอาคุณสมบัติที่เพิ่มเติมมาของออบเจ็กต์มาใช้ ดังที่ได้กล่าวมาแล้วในข้างต้น และในบทที่ 3 จากตารางรูปที่ 4-1 เราสามารถนำมาเสนอเป็น ตารางใหม่ที่เป็น TORDB ได้ดังตารางที่ 4-3

Oid	รหัสนักศึกษา	ชื่อ	นามสกุล	ภาควิชา
512	41013524 [03/01/1985] –Now	จิราเจตน์ [03/01/1985]-[01/01/1991] จิรายุ [01/01/1991] – Now	อ่อนสา [03/01/1985] – Now	คอมพิวเตอร์ [01/01/1991]-[12/31/1993] โทรคมนาคม [12/31/1993] – Now
513	41013539 [02/01/1988] – [10/31/1995]	ประยงค์ [02/01/1988]-[10/31/1995]	ทานนท์ [02/01/1988] - [10/31/1995]	คอมพิวเตอร์ [02/01/1988] - [10/31/1995]
514	41013533 [04/01/1991]-Now	ธีรวุธ [04/01/1991] – Now	สันติสุธีรวุฒิ [04/01/1991] – Now	คอมพิวเตอร์ [04/01/1991] – Now
515	41013549 [06/01/1988]-Now	วิเชษฐ์ [06/01/1988] – Now	เจือตี๋ [06/01/1988] –Now	คอมพิวเตอร์ [06/01/1988] – Now

ตารางที่ 4-3 ตัวอย่างตารางในระบบฐานข้อมูลเชิงเวลาที่ทำการแปะเวลาไว้ที่ Column

จากตารางที่ 4-3 เราจะมองใน 1 Tuple เป็น Fact เพียง Fact เดียว ข้อมูลที่มีการเปลี่ยนแปลงจะเก็บไว้ในแต่ละแอตทริบิวต์นั้นๆ โดยภายในแต่ละแอตทริบิวต์นั้นจะมีลักษณะเป็นออบเจ็กต์และภายในออบเจ็กต์นั้นจะมีลักษณะเป็นเซลล์ (Cell) ที่เก็บค่าของข้อมูลที่เปลี่ยนไปตามเวลาโดยมี Valid time ไปเกาะกับข้อมูลที่เปลี่ยนแปลงภายในเซลล์นั้น ตามหลักการของฐานข้อมูลเชิงเวลา

ในการแปะเวลาไว้ที่คอลัมน์ นั้นจะช่วยแก้ปัญหาคำถามข้างต้นได้ คือจะทราบว่าคอลัมน์ไหนมีอะไรเปลี่ยนไปบ้างและได้ทำการเปลี่ยนแปลงไปเวลาใด และลักษณะการเอาเวลาไปแปะกับข้อมูลในคอลัมน์ จะมีลักษณะข้อมูลที่มีโครงสร้างมากว่าการแปะเวลาไว้ที่ Tuple คือ Fact ใหม่ที่เกิดขึ้นมาจะอยู่ภายใน Tuple เดียวกันทั้งหมด จะไม่เหมือนกับแบบการแปะเวลาไว้ที่ Tuple คือ แบบ Tuple ถ้ามี Fact ใหม่จะทำการสร้าง Row ใหม่ขึ้นมาเรื่อย ๆ และในการแปะเวลาไว้ที่คอลัมน์นั้นจะมีการจัดเก็บในกลุ่มของแอตทริบิวต์ของข้อมูลเป็นลักษณะออบเจ็กต์ ที่ซ้อน ๆ

กัน ซึ่งเราเรียกว่า คอลเล็คชั่นไทป์ โดยจะมีการจัดเก็บเป็นแบบ เซต มัลติเซต หรือ ลิสต์ โดยการประกาศ Type แบบนี้เพื่อรองรับการจัดเก็บข้อมูลที่ซับซ้อนได้ ลักษณะการจัดเก็บข้อมูลแบบเป็นเซต นั้น ข้อมูลที่อยู่ใน คอลัมน์นั้นเวลาที่เราทำการ Insert ข้อมูลที่ซ้ำกันลงไปในฐานข้อมูลจะถูก Reject โดย DBMS เป็นตัว Reject ให้ทันที แต่ถ้าการจัดเก็บแบบเป็นลิสต์นั้น ลักษณะการจัดเก็บจะสามารถมีข้อมูลที่ซ้ำกันได้แต่ข้อมูลที่ซ้ำกันนี้จะเป็นคนละตัวกับข้อมูลตัวแรกที่มีอยู่ในฐานข้อมูลเพราะข้อมูลแบบลิสต์จะมีลักษณะซ้ำได้และมีลำดับกำกับ สำหรับในโครงงานนี้ได้นำเสนอการจัดเก็บข้อมูลเป็นแบบลิสต์

4.3 การนำ Oid มาแก้ปัญหา

การนำข้อมูลเชิงเวลามาเก็บไว้ในแต่ละคอลัมน์นั้นในการใช้งานจริง ๆ นั้นมีความเป็นไปได้ที่จะมีการเปลี่ยนแปลงมากกว่า 1 แอตทริบิวต์ในตาราง ซึ่งเป็นปัญหาที่ได้กล่าวมาทั้งหมดจากข้างต้น จึงได้นำค่ารหัสเฉพาะหรือ Object Identification (Oid) เข้ามาทำระบุเพื่อยืนยันว่า ข้อมูลที่เหมือนกันในแต่ละ Tuple ไม่ใช่ข้อมูลตัวเดียวกันเพราะเรามีตัว Oid เข้ามาชี้บอกไว้อยู่แล้ว ซึ่งใน DBMS ของ Informix ก็คือตัว Serial นั้นเอง

ค่า Oid นั้นจะรันมาจากตัว DBMS โดย System Generator ค่านี้ให้ซึ่งค่า Oid นี้จะไปเกาะอยู่ตาม Tuple ของตารางซึ่งจะมีค่าไม่เหมือนกัน เนื่องด้วยในการใช้งานจริงนั้นค่าข้อมูลชนิด Oid นั้นสามารถนำมาใช้งานได้ยากเนื่องจาก ตัว DBMS ที่ Generator Oid ออกมานั้นมีค่าที่ไม่แน่นอนและยังกำหนดค่าเริ่มต้นไม่ได้ เราจึงต้องนำข้อมูลชนิด Serial ที่มีอยู่ใน Informix ซึ่งมีคุณลักษณะที่คล้าย Oid มาใช้แทน และสิ่งที่ค่าข้อมูลชนิด Serial มีความแตกต่างจาก Oid ก็คือโปรแกรมเมอร์เป็นคนกำหนดค่าเริ่มต้นกับค่าข้อมูลชนิด Serial ได้ จึงมีความยืดหยุ่นสูงที่จะนำมาใช้เป็น Primary Key ในการเขียนโปรแกรม

Oid	รหัสนักศึกษา	ชื่อ	นามสกุล	ภาควิชา
513	41013539 [03/01/1985] – Now	ประยงค์ [03/01/1985]-[01/01/1991] อัศวิน [01/01/1991] – Now	ทานนท์ [03/01/1985] – Now	คอมพิวเตอร์ [01/01/1991]-[12/31/1993] โทรคมนาคม [12/31/1993] – Now
514	41013533 [04/01/1991]-Now	ธีรวุธ [04/01/1991] – [10/30/1992] อัศวิน [10/30/1992]-Now	สันติสุธีรวุฒิ [04/01/1991] – Now	คอมพิวเตอร์ [04/01/1991] – Now

ตารางที่4-4 ตารางที่แสดงถึงการใช้ Oid

จากตารางข้างต้นจะเห็นว่า ใน Tuple แรกนายประยงค์ได้เปลี่ยนชื่อเป็น อัศวิน และใน Tuple ที่สอง นายธีรวุธ ได้เปลี่ยนชื่อไปเป็นอัศวินเช่นกัน ถ้าไม่มีการนำ Oid มาใช้ เวลาที่ทำการ Query อัศวิน ก็จะได้อัศวินออกมาทั้งสองคนออกมา ซึ่งจริงๆ แล้ว ทั้งสอง คนนี้เป็นคนละคนกัน จึงจำเป็นที่จะต้องเอา Oid เข้ามาช่วยโดยใช้ข้อมูลชนิด Serial มาเป็น Primary Key ในการเขียนโปรแกรม

ตัวอย่างโปรแกรมจากตารางที่ 4-4

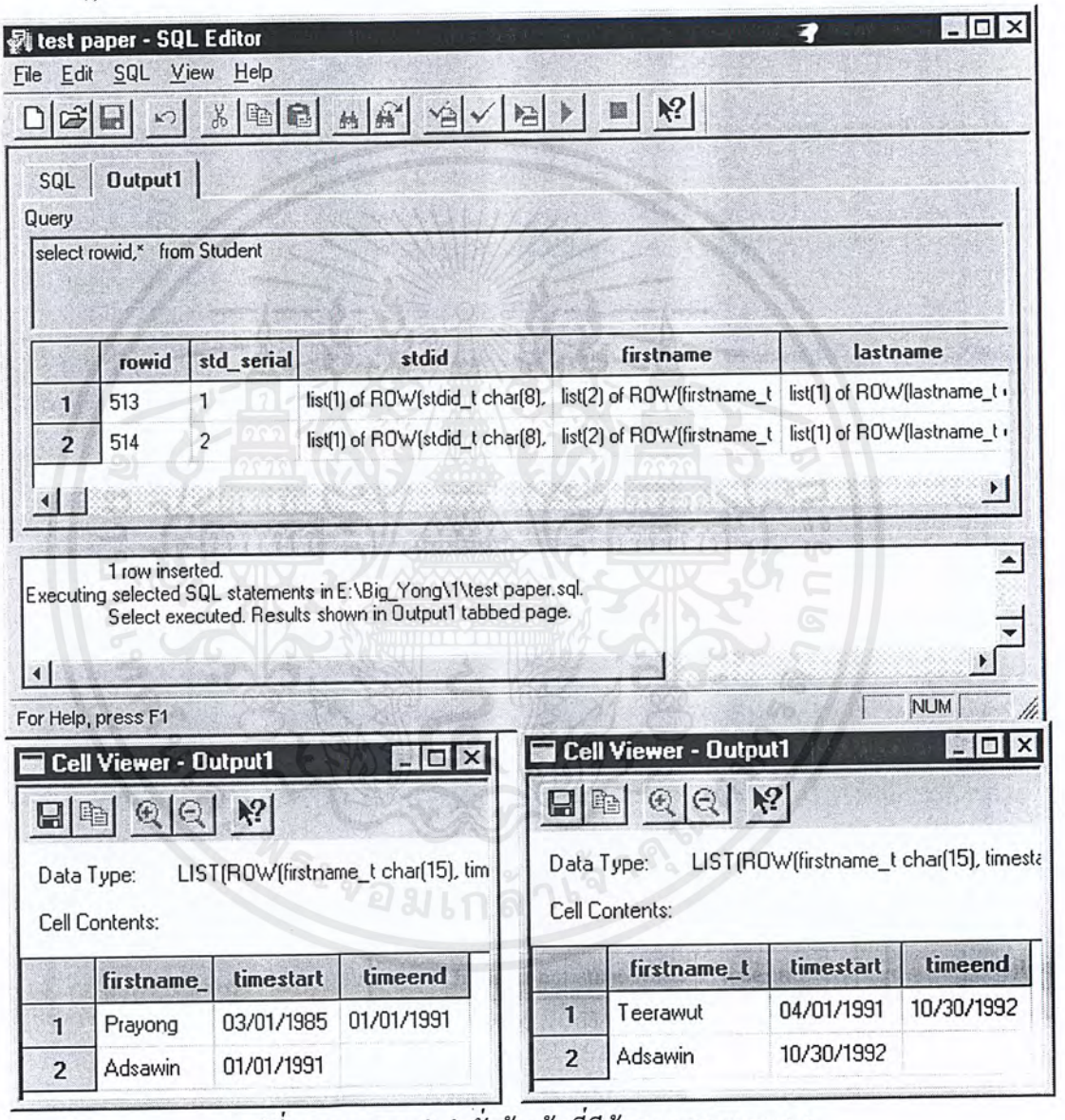
```
CREATE TABLE STUDENT

( Std_serial        Serial Primary key ,
StdID               List(Row(StdID_t  CHAR(8), TimeStart  DATE, TimeEnd    DATE) NOT
NULL),
FirstName           List(Row(FirstName_t CHAR(15), TimeStart    DATE, TimeEnd    DATE)
NOT NULL),
LastName            List(Row(LastName_t  CHAR(15), TimeStart    DATE, TimeEnd    DATE)
NOT NULL),
Major               List(Row(Major_t     CHAR(15),  TimeStart   DATE, TimeEnd    DATE)
NOT NULL)
);

insert into student  values

(           0,
"list{row('41013539','3/1/1985',Null)}",
"list{row('Prayong','3/1/1985','1/1/1991'),row('Adsawin','1/1/1991',Null)}",
 "list{row('Tanon','3/1/1985',Null)}",
"list{row('Computer','1/1/1991','12/31/1993'),row('Telecom','1/1/1993',Null)}"
 );
```

```
insert into student  values
( 0,
"list{row('41013533','4/1/1991',Null)}",
"list{row('Teerawut','4/1/1991','10/30/1992'),row('Adsawin','10/30/1992',Null)}",
"list{row('Suntisuteerawut','4/1/1991',Null)}",
"list{row('Computer','4/1/1991',Null)}"
);
```



รูปที่4-1 แสดงการทำคำสั่งข้างต้นที่มีข้อมูลตามตาราง 4-4

บทที่ 5

ภาษา SPL (Stored Procedure Language)

Stored Procedure Language หรือ SPL เป็นภาษาที่มีความสามารถของภาษา SQL และได้เพิ่มความสามารถในส่วนการควบคุมการทำงาน เช่น การทำลูป การทำทางเลือก เข้าไป โดย SPL เป็นรูทีนที่ฝังอยู่กับระบบจัดการฐานข้อมูล (Database Management System : DBMS) ข้อดีของการมีรูทีนคือสามารถรวบรวมคำสั่งภาษา SQL หลายๆคำสั่งไว้ด้วยการเรียกใช้รูทีนเพียงครั้งเดียว (Encapasulate Multiple SQL Statement) และ ขยายขีดความสามารถของบิวท์อินฟังก์ชั่นที่มีอยู่เดิม (Extend Function Built-In Data Type)

คุณสมบัติที่สำคัญอีกประการของ SPL คือความสามารถในการเข้าไปจัดการแต่ละสมาชิกของคอลเลคชั่นไทป์

5.1 การสร้าง SPL รูทีน

สามารถสร้าง SPL รูทีนได้ 2 ประเภท คือ โพรซีเดอร์ซึ่งไม่มีการส่งคืนค่าข้อมูลและฟังก์ชั่นซึ่งจะต้องมีการกำหนดการส่งคืนค่าข้อมูล การสร้าง SPL รูทีนนั้นสามารถสร้างให้มีชื่อซ้ำกันได้เนื่องจากอินฟอร์มิกซ์สนับสนุนความสามารถในการทำรูทีนโอเวอร์โหลดดิง (Routine Overloading) แต่ๆละรูทีนต้องมีซิกเนเจอร์ต่างกัน โดยเราสามารถกำหนดความแตกต่างของรูทีนได้หลายวิธีดังนี้

- ชนิดของรูทีนต่างกัน
- ชื่อของรูทีนต่างกัน
- จำนวนพารามิเตอร์
- ชนิดของพารามิเตอร์ของรูทีน
- ลำดับของพารามิเตอร์

5.2 กำหนดชื่อ พารามิเตอร์ และชนิดข้อมูลที่จะทำการส่งค่าคืน

เป็นการกำหนดชนิดของรูทีนที่จะทำการสร้างโดยใช้คีย์เวิร์ดเป็นตัวระบุชนิด ถ้ามีการส่งพารามิเตอร์จะต้องทำการกำหนดชื่อและชนิดของพารามิเตอร์ และในกรณีของฟังก์ชั่นจะต้องมีการกำหนดชนิดของข้อมูลที่จะส่งคืนด้วย ดังตัวอย่าง

```
CREATE PROCEDURE New_price(percent REAL)
        .
        .
END PROCEDURE
```

หรือ

```
CREATE FUNCTION find_group(id INT)
    RETURNING INT,REAL;
```

.

.

END FUNCTION

สำหรับการกำหนดพารามิเตอร์เราไม่สามารถกำหนดพารามิเตอร์เป็น Text หรือ Byte โดยตรงได้ ต้องใช้คีย์เวิร์ด REFERENCES ในการอ้างถึงพารามิเตอร์ที่เป็นข้อมูลชนิด Text หรือ Byte ดังกล่าว ตามตัวอย่าง

CREATE PROCEDURE Proc1(lo_text **REFERENCES TEXT**)

ใน SPL จะมีการสร้างบล็อค(Block) การทำงานภายในโดยใช้คีย์เวิร์ด BEGIN และ END ส่วนการกำหนดคอมเมนท์ (Comment) ทำได้โดยการใช้ { } หรือ -- ดังตัวอย่าง

```
CREATE FUNCTION block_demo()          -- SPL Function
RETURNING INT;
DEFINE distance INT;
LET distance = 37;
{ Show how to create implicit statement block}
BEGIN                                 -- Begin implicitstatement block
DEFINE distance INT;
LET distance = 2;
END                                   -- End implicitstatement block
RETURN distance;
END FUNCTION;
```

5.3 การประกาศค่าตัวแปรและการกำหนดค่าให้ SPL

การประกาศค่าตัวแปรทำได้โดยใช้คีย์เวิร์ด DEFINE ใน SPL รูทีนนั้น สามารถกำหนดตัวแปรขึ้นมาใช้งานได้ 2 ลักษณะ คือ ตัวแปรโกลบอล (Global Variable) และตัวแปรโลคอล (Local Variable) โดยตัวแปรโกลบอลจะสามารถใช้งานร่วมกันได้ในหลายรูทีนเพราะจะทำการจองเนื้อที่บนหน่วยความจำไว้และจะต้องทำการกำหนดค่าดีฟอลท์(Default)ไว้ แต่ตัวแปรโลคอลจะสามารถใช้งานได้เพียงบน SPL นั้นๆ จะถูกเคลียร์ค่าทุกครั้งที่การใช้งานรูทีนนั้นสิ้นสุดลง และไม่สามารถกำหนดค่าของข้อมูลเป็นดีฟอลท์ (Default) ได้ ตัวอย่างการประกาศตัวแปรโลคอลชนิดต่างๆ

```
DEFINE today DATETIME YEAR TO DAY;      -- Built in type
DEFINE b REFERENCES BYTE;               -- Simple large object
DEFINE a COLLECTION;                    -- Collection type
DEFINE c SET                            -- Collection Type
CREATE ROW TYPE zip_t                   -- Named row type
(z_code CHAR(5),                        -- Unnamed row type
z_suffix CHAR(4));
DEFINE manager ROW ( nameVARCHAR(30),
                     department VARCHAR(30),
                     salaryINTEGER
```

การประกาศตัวแปรโกลบอลจะต้องมีคีย์เวิร์ด GLOBAL และการกำหนดค่า DEFAULT เพิ่มเข้ามาดังตัวอย่าง

DEFINE GLOBAL gvar INT **DEFAULT** 2;

การกำหนดค่าให้กับตัวแปร 4 ลักษณะดังนี้

1. LET เป็นการกำหนดตัวแปรให้กับค่าใดๆซึ่งจะอยู่ในรูปของ Expression

```
LET a = 5;
LET b = 6; LET c = 10;
LET a,b = 10,c+d;
LET a,b = (SELECT cola,colb FROM tab1 WHERE cola=10);
LET d = func1(x,y);
LET a = ROW ('A Street', 'Nowhere', 'AA',
ROW(NULL, NULL))::address_t
```

2. SELECT…INTO.. เป็นการกำหนดค่าตัวแปรโดยทำการ Select มาจากฐานข้อมูล แล้วทำการเก็บค่าลงไปในตัวแปร
3. EXECUTE FUNCTION…INTO… เป็นการกำหนดตัวแปรโดยใช้ค่าที่ได้จากฟังก์ชัน เมื่อทำการเอ็กซีคิวฟังก์ชันค่าที่รีเทิร์นกลับมาจะเก็บไว้ในตัวแปร
4. CALL…RETURNING… เป็นการกำหนดตัวแปรโดยการคืนค่าจากโพรซีเดอร์

ตัวอย่างการใช้การกำหนดค่าวิธีอื่นๆ

```
SELECT fname, lname INTO a, b FROM customer
WHERE customer_num = 101

EXECUTE FUNCTION read_address('Smith')
INTO p_fname, p_lname, p_add, p_city, p_state, p_zip;

CALL read_address('Smith')
RETURNING p_fname, p_lname, p_add, p_city, p_state, p_zip;
```

5.4 Foreach และ Cursor

ใน SPL มีการใช้เคอร์เซอร์ (Cursor) เนื่องมาจากการกำหนดค่าในตัวแปรตัวหนึ่ง เมื่อมีการคืนค่ากลับขึ้นมาอาจมีค่ามากกว่าหนึ่งค่าถูกส่งมา จึงมีคำสั่ง Foreach มาจัดการกับแต่ละ Row ที่ถูกส่งคืนมา เช่น ในคำสั่ง Select,Execute และ Call ซึ่งในการใช้งาน Foreach และเคอร์เซอร์นี้ สามารถย่อยใช้ได้อีกภายใน Foreach นั้น SPL รูทีนซึ่งรีเทิร์นกลุ่มของ Row จะเรียกว่า เคอร์เซอร์รูทีน (Cursor Routine)

ตัวอย่างการใช้งาน Foreach

```
CREATE_PROCEDURE increase_by_pct( pct INTEGER )
    DEFINE s INTEGER;
    FOREACH sal_cursor FOR
            SELECT salary INTO s FROM employee
                    WHERE salary > 35000
            LET s = s + s * ( pct/100 );
            UPDATE employee SET salary = s
                    WHERE CURRENT OF sal_cursor;
    END FOREACH
END PROCEDURE;
```

การใช้ Foreach ในการใช้งานคอลเลคชั่นไทป์ทำได้โดย Select เอาคอลัมน์ที่เป็นคอลเลคชั่นไทป์ขึ้นมาเก็บในตัวแปรซึ่งตัวแปรนั้นจะเปรียบเสมือนเทเบิลที่เป็นสมาชิกของคอลเลคชั่นไทป์แล้วใช้ Foreach เข้าไปจัดการกับแต่ละสมาชิกได้

5.5 Flow Control

Flow Control ใน SPL มี 2 ชนิดคือ

- IF-ELIF-ELSE ใช้ในการตรวจเงื่อนไขแล้วแยกทำแต่ละส่วนตามเงื่อนไข ตัวอย่างการใช้ IF-ELIF-ELSE

```
CREATE FUNCTION str_compare( str1 CHAR(20), str2 CHAR(20))
        RETURNING INTEGEF ,
        DEFINE result INTEGER;
        IF str1 > str2 THEN
                Result = 1;
        ELIF str2 > str1 THEN
                result = -1;
        ELSE
                result = 0;
        END IF
        RETURN result;
    END FUNCTION;
```

- WHILE และ FOR ใช้ในการทำลูป ทำบล็อค (Block) ของสเตดเมนท์โดยจะมีเงื่อนไขในการตรวจสอบเพื่อออกจากลูป ตัวอย่างการใช้งานลูป WHILE และ FOR

```
CREATE PROCEDURE test_rows( num INT )
        DEFINE i INTEGER;
        LET i = 1;
        WHILE i < num
                INSERT INTO table1 (numbers) VALUES (i);
                LET i = i + 1;
        END WHILE;
    END PROCEDURE;

    FOR i = 1 TO 10
        IF i = 5 THEN
        CONTINUE FOR;
```

```
.
.
        ELIF i = 8 THEN
        EXIT FOR;
        END IF;
    END FOR;
```

การรีเทิร์นค่าของ SPL มีได้ทั้งคืนมาค่าเดียวและคืนมาหลายค่าดังตัวอย่าง

```
CREATE FUNCTION increase_by_pct(amt DECIMAL, pct DECIMAL)
        RETURNING DECIMAL;
        DEFINE result DECIMAL;
        LET result = amt + amt * (pct/100);
        RETURN result;
END FUNCTION;

CREATE FUNCTION b_date_2( num INTEGER )
RETURNING VARCHAR(30), DATE;
        DEFINE n VARCHAR(30);
        DEFINE b DATE;
        FOREACH cursor1 FOR
                SELECT name, bdate INTO n, b FROM person
                        WHERE emp_no > num;
                RETURN n, b WITH RESUME;
        END FOREACH
END FUNCTION;
```

ในกรณีที่การคืนค่ามีมากกว่าหนึ่งเช่นดังตัวอย่างต้องทำการประกาศตัวแปรไว้ และถ้าไม่มี Row Return ค่าของตัวแปรที่ถูกส่งคืนกลับมาจะเป็น Null

5.6 การจัดการกับ Collection Type

การจัดการกับข้อมูลที่เป็นคอลเลคชั่นมีขั้นตอนดังนี้ ประกาศตัวแปรคอลเลคชั่นมประกาศตัวแปรที่จะทำการเก็บสมาชิกแต่ละตัวในคอลเลคชั่น แล้วทำการ Select คอลเลคชั่นจากฐานข้อมูล ดังตัวอย่าง

```
DEFINE P_coll SET( INTEGER NOT NULL );
DEFINE P INTEGER;
SELECT primes INTO P_coll FROM numbers
        WHERE id = 220;
```

เมื่อเราทำการสร้างตัวแปรที่เป็นตัวเก็บค่าที่รีเทิร์นจากคอลเลคชั่น ตัวแปรนั้นจะเสมือนกับเป็นเทเบิลหนึ่ การจะ Insert ค่าให้กับตัวแปรคอลเลคชั่นนั้นทำได้ดังนี้

```
INSERT INTO TABLE (<Collection Variable>) VALUES (<Value>)
```

```
INSERT INTO TABLE (P_coll)  VALUES (3) ;
```

การ Insert ข้อมูลที่เป็นคอลเลคชั่นนั้นทำได้ 2 วิธีคือใช้เคอร์เซอร์และไม่ใช้เคอร์เซอร์ การ Insert แบบไม่ใช่เคอร์เซอร์จะทำได้เมื่อคอลเลคชั่นที่เราใช้ต้องเป็นเซตหรือมัลติเซตเท่านั้นเพราะลำดับของสมาชิกไม่มีความสำคัญ ดังตัวอย่าง

```
CREATE PROCEDURE new_emp( emp VARCHAR(30), mgr VARCHAR(30) )
    DEFINE r SET(VARCHAR(30) NOT NULL);
    SELECT direct_reports INTO r FROM manager
        WHERE mgr_name = mgr;
    INSERT INTO TABLE (r) VALUES(emp);
    UPDATE manager SET direct_reports = r
        WHERE mgr_name = mgr;
END PROCEDURE;
```

เมื่อเราทำการแก้ไขข้อมูลในตัวแปรคอลเลคชั่นต้องมีการแก้ไขค่าดังกล่าวลงไปบนฐานข้อมูล ดังตัวอย่าง

```
CREATE PROCEDURE shapes()
    DEFINE vertexes SET(point NOT NULL);
    DEFINE pnt point;
        SELECT definition INTO vertexes FROM polygons
            WHERE id = 207;
        FOREACH cursor1 FOR
```

```
                SELECT * INTO pnt FROM TABLE(vertexes)
                IF pnt = '(3,4)' THEN
                        DELETE FROM TABLE(vertexes)
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                ELSE
                CONTINUE FOREACH;
                END IF;
        END FOREACH
        UPDATE polygons SET definition = vertexes
                WHERE id = 207;
END PROCEDURE;
```

การจะเข้าไป Select ข้อมูลที่เป็นคอลเลคชั่นต้องเริ่มด้วยการเคอร์เซอร์โดยใช้คีย์เวิร์ด FOREACH

```
        FOREACH cursor1 FOR
                SELECT * INTO pnt FROM TABLE(vertexes)
                        .
                        .
        END FOREACH
```

การอัพเดทสมาชิกในคอลเลคชั่นทำได้โดยการดึงข้อมูลมาเก็บในตัวแปรคอลเลคชั่น แล้วทำการประกาศเคอร์เซอร์ขึ้นมา ดังตัวอย่าง

```
DEFINE s SET(INTEGER NOT NULL);
DEFINE n INTEGER;
SELECT numbers INTO s FROM orders
        WHERE order_num = 10;
FOREACH cursor1 FOR
        SELECT * INTO n FROM TABLE(s)
        IF ( n == 500 ) THEN
                UPDATE TABLE(s)(x)
                        SET x = 400 WHERE CURRENT OF cursor1;
                EXIT FOREACH;
```

```
            ELSE
                    CONTINUE FOREACH;
            END IF;
        END FOREACH
```

เมื่อเราต้องการจะอัพเดทข้อมูลคอลเลคชั่นในฐานข้อมูลโดยใช้ตัวแปรคอลเลคชั่นสามารถทำได้ดังตัวอย่าง

```
CREATE PROCEDURE new_report(mgr VARCHAR(30),
        old VARCHAR(30), new VARCHAR(30) )
        DEFINE s SET (VARCHAR(30) NOT NULL);
        DEFINE n VARCHAR(30);
        SELECT direct_reports INTO s FROM manager
                WHERE mgr_name = mgr;
        FOREACH cursor1 FOR
                SELECT * INTO n FROM TABLE(s)
                IF ( n == old ) THEN
                        UPDATE TABLE(s)(x)
                                SET x = new WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                ELSE
                        CONTINUE FOREACH;
                END IF;
        END FOREACH
        UPDATE manager SET mgr_name = s
                WHERE mgr_name = mgr;
END PROCEDURE;
```

5.7 การเรียกใช้รูทีน

สามารถเรียกใช้ได้หลายวิธีดังนี้

- คำสั่ง EXECUTE PROCEDURE หรือ EXECUTE FUNCTION ใน DB-Access
- เรียกจาก SPL รูทีนอื่นหรือจากรูทีนภายนอก ด้วยคำสั่ง CALL
- เรียกผ่าน SQL สเตตเมนท์

ตัวอย่างการเรียกใช้รูทีนแบบต่างๆ

```
EXECUTE PROCEDURE update_orders();
EXECUTE FUNCTION scale_rectangles( 1 07, 1.9 )
        INTO new;
CALL area(rectv.length, rectv.width) RETURNING a;
SELECT increase_by_pct(price, 20) INTO p
        FROM inventory WHERE prod_num = num;
```

ในการเรียกค้นข้อมูลที่เป็น LIST นั้นทำได้โดยเรียกค้นข้อมูลในคอลัมน์ที่เก็บกลุ่มของข้อมูลที่เป็นลิสต์จะต้องใช้วิธีการเขียนภาษา SPL ซึ่งจะมีประโยชน์มาก โดยเราจะต้องเขียนโค้ดของภาษา SPL นี้ฝังไว้ที่ Server ซึ่งเราจะต้องเขียนภาษานี้จัดการกับข้อมูลที่เป็นลิสต์เอง ใน Informix นั้นสามารถที่จะทำคอลัมน์เป็นกลุ่มของข้อมูลเช่นเป็นลิสต์ เป็นเซตซึ่งต่อไปนี้จะเสนอวิธีการเรียกค้นจากลิสต์ได้ในบทที่ 6 คือภาษา SPL ที่จัดการกับข้อมูลที่เป็น List ถัดไป

บทที่ 6
ภาษา SPL ที่จัดการกับข้อมูลที่เป็น List

ทั้ง List และ Set เป็นข้อมูลแบบ collection ที่มีความแตกต่างของกลุ่มของสมาชิก(Element) ภายในคือ

List ลักษณะของสมาชิกจะซ้ำกันได้และสมาชิกแต่ละตัวจะมีลำดับเช่น LIST{5,1,7,31,19,5,13}

Set ลักษณะของสมาชิกจะไม่ซ้ำและสมาชิกแต่ละตัวไม่มีลำดับเช่น SET {5,7,31,19,13}

แต่ในการทำวิจัยนี้เราได้เลือกเอาการเก็บข้อมูลแบบ List เพราะการเก็บข้อมูลเป็นกลุ่มแบบ List นี้เราสามารถที่จะทำการ insert ข้อมูลเข้าไปในกลุ่มข้อมูลเข้าไปในตำแหน่งใดๆ ก็ได้ที่อยู่ใน List และมีความยืดหยุ่นกว่าการเก็บข้อมูลที่เป็นแบบ set เพราะสมาชิกแต่ล่ะตัวมีลำดับเป็นดังบ่งชี้อยู่ ดังตัวอย่าง การเก็บประวัติการเปลี่ยนรหัส ชื่อ นามสกุล และที่อยู่ของนักศึกษาในตาราง STUDENT โดยใช้ Std_serial เป็น Primary key ซึ่งเป็นตัวแปรข้อมูลชนิด Serial ซึ่งมีใน Informix โดยค่า serial จะมีค่าเริ่มต้นที่ 100 เป็นต้นไป ตามคุณสมบัติของข้อมูลชนิด serial

ตาราง STUDENT

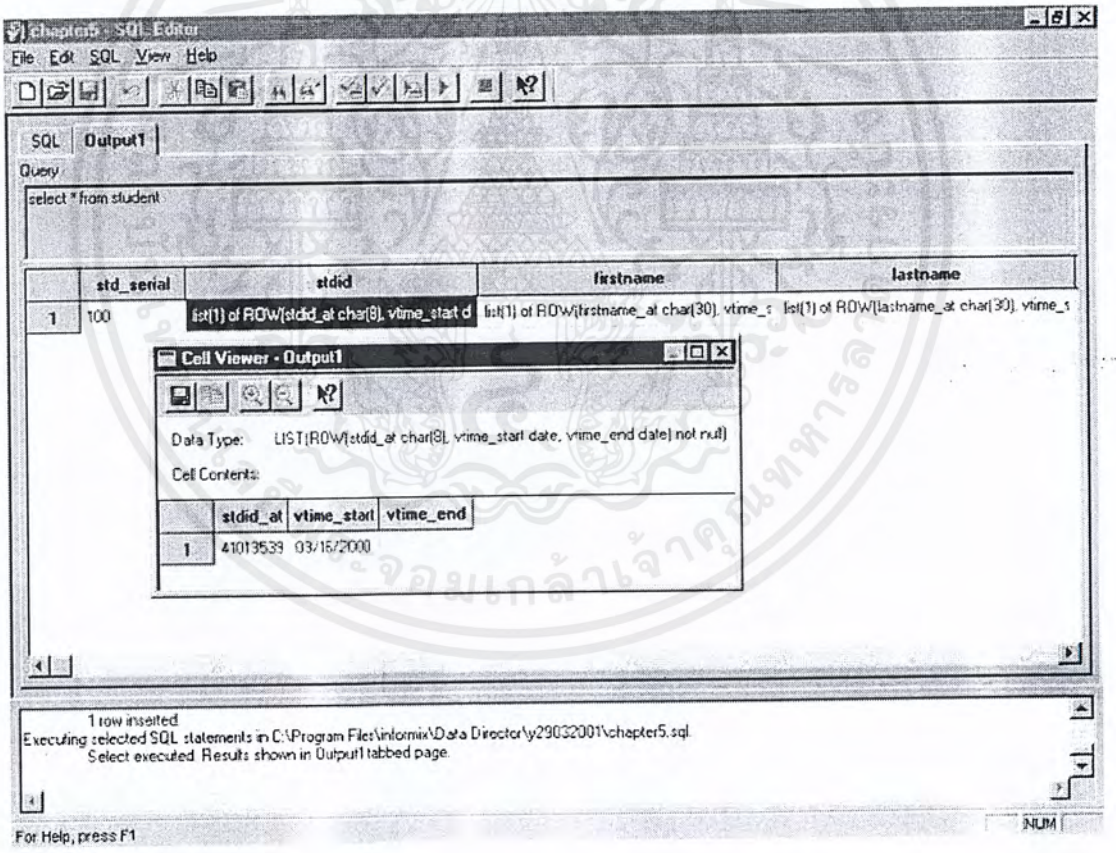
```
CREATE TABLE STUDENT
(       Std_serial      Serial(100)  Primary key ,
        StdID           List(Row(StdID_at   CHAR(8) ,Vtime_Start   DATE ,Vtime_End   DATE)
                        NOT NULL),
        FirstName       List(Row(FirstName_at CHAR(30) ,Vtime_Start DATE ,Vtime_End
                        DATE) NOT NULL),
        LastName        List(Row( LastName_at     CHAR(30) ,Vtime_Start     DATE ,Vtime_End
                        DATE) NOT NULL),
        Address         List(Row( Address_at      CHAR(70)  ,Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL)
);
```

จากตาราง STUDENT ข้างบนค่าของทุกแอตทริบิวต์ยกเว้น Primary key จะเก็บเป็นแบบ List ของ Row หมายถึงในแต่ล่ะ แอตทริบิวต์จะเก็บเป็นกลุ่มของข้อมูลเป็น List มีลำดับกำกับซึ่งจะมีค่าของเวลาเริ่ม (Vtime_Start = Valid time start) และ ค่าของเวลาสิ้นสุด (Vtime_End = Valid time end) แปะอยู่ในแต่ล่ะแอตทริบิวต์เพื่อบอกถึงการเปลี่ยนแปลงของแอตทริบิวต์นั้น

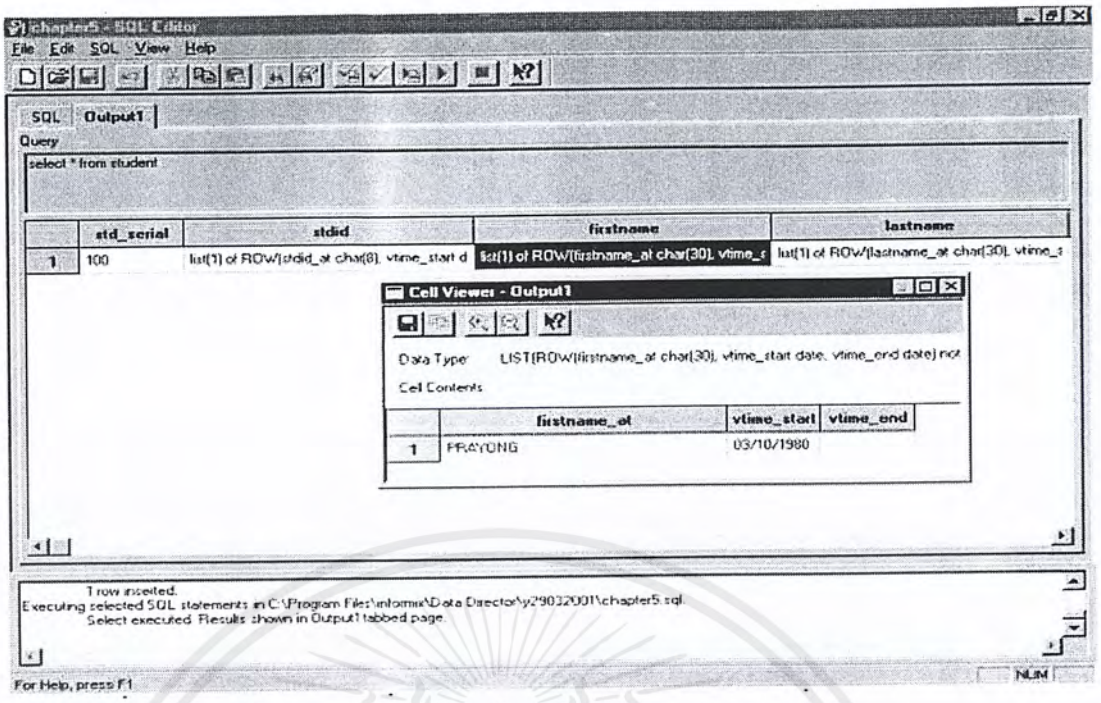
6.1 การ Insert ข้อมูลที่เก็บเป็น List ใช้คำสั่งดังนี้

```
insert into student  values
(          0,
           "list{row('41013539','03/16/2000',Null)}",
           "list{row('PRAYONG','03/10/1980',Null)}",
           "list{row('TANON','03/02/1980',Null)}",
           "list{row('BANGKOK','02/02/2000',Null)}"
);
```

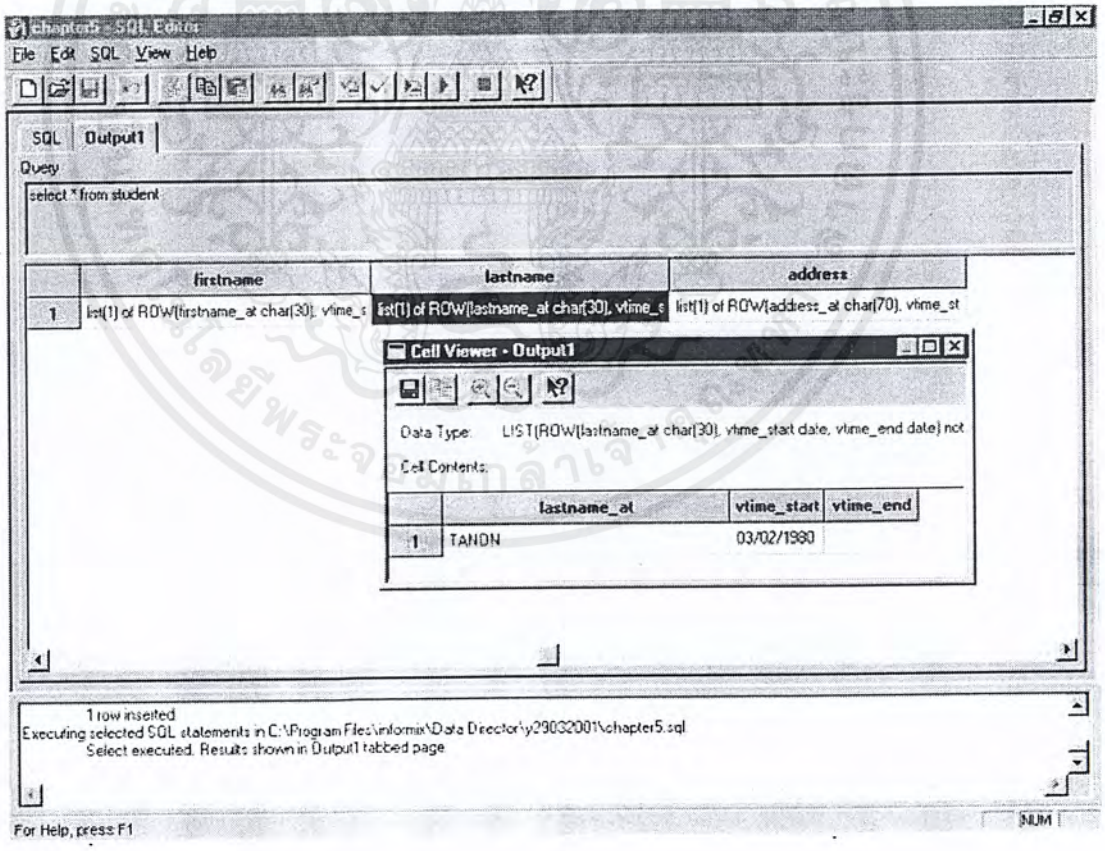
ในการ Insert ข้อมูล Std_serial นั้นเราให้ค่า 0 หมายความว่าตัวระบบจะ ให้ค่าเป็นอัตโนมัติเริ่มต้นที่ 100 (ขึ้นอยู่กับที่เราให้ค่าตอนสร้างตารางในที่นี้ คือ Std_serial Serial(100) Primary key)



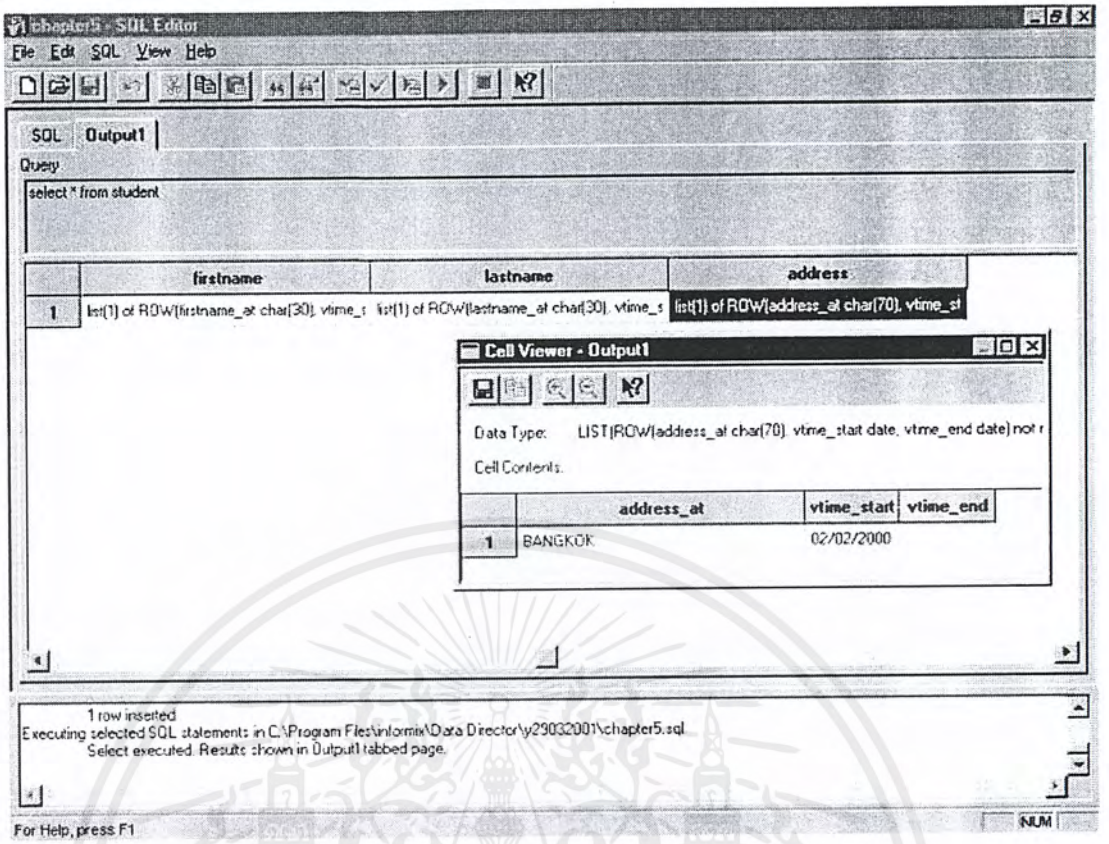
รูปที่ 6-1 แสดงการเก็บข้อมูลรหัสนักศึกษาในตารางหลัง Insert ข้อมูลที่เก็บเป็น List



รูปที่ 6-2 แสดงการเก็บข้อมูลชื่อนักศึกษาในตารางหลัง Insert ข้อมูลที่เก็บเป็น List



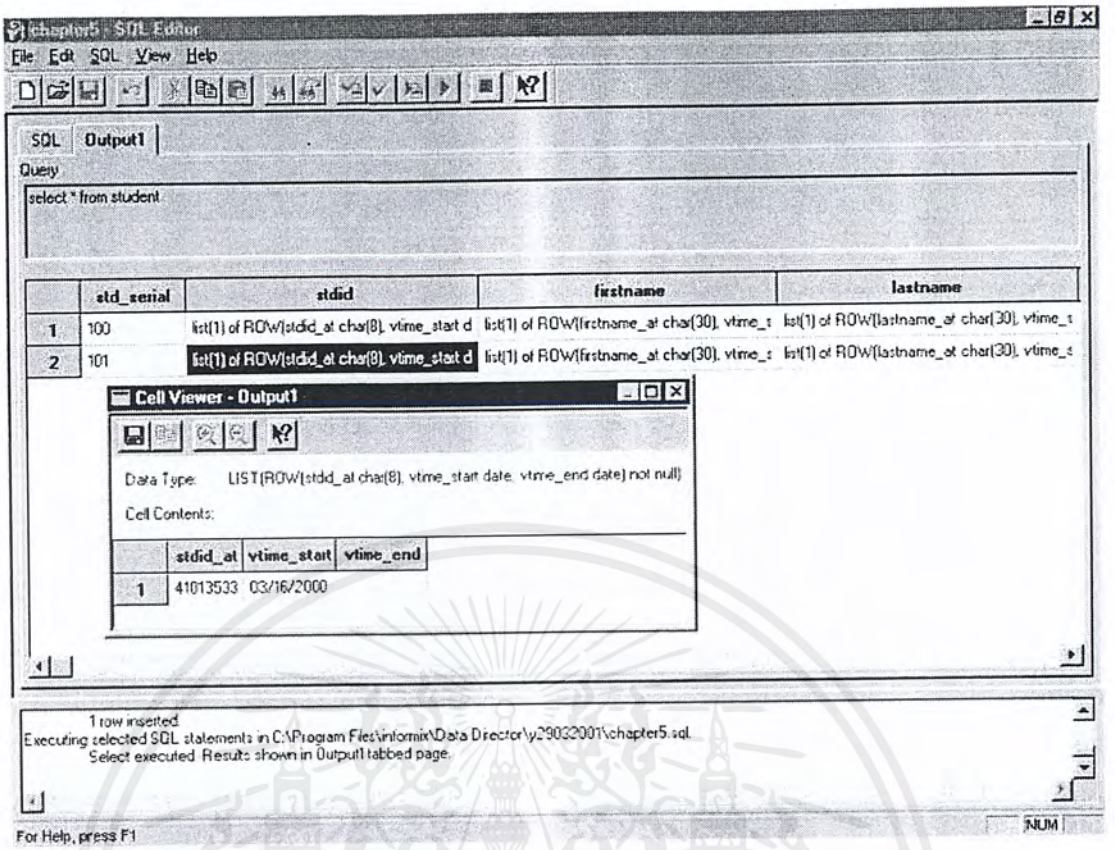
รูปที่ 6-3 แสดงการเก็บข้อมูลนามสกุลนักศึกษาในตารางหลัง Insert ข้อมูลที่เก็บเป็น List



รูปที่ 6-4 แสดงการเก็บข้อมูลที่อยู่นักศึกษาในตารางหลัง Insert ข้อมูลที่เก็บเป็น List

ถ้า Insert เพิ่มข้อมูลลงในตารางจะใช้คำสั่ง ดังนี้

```
insert into student  values
(       0,
        "list{row('41013533','03/16/2000',Null)}",
        "list{row('Teerawut','03/10/1980',Null)}",
        "list{row('Suntisuteerawut','03/02/1980',Null)}",
        "list{row('Nontharburi','02/02/2000',Null)}"
);
```



รูปที่ 6-5 แสดงการเก็บข้อมูลนักศึกษาในตารางหลัง Insert ข้อมูลที่สองที่เก็บเป็น List

จากรูปที่ 6-5 จะเห็นได้ว่าในตารางจะมี tuple เพิ่มมาอีกหนึ่ง tuple จะเห็นได้ว่าช่อง std_serial นั้นจะเพิ่มขึ้นเป็น 101 ซึ่งถ้ามีการ Insert ข้อมูลเพิ่มมาอีกค่า ก็จะมี tuple ใหม่อย่างนี้ขึ้นมาอีก

6.2 การ Delete และ Update ข้อมูลที่เก็บเป็น List

ก็ทำคล้ายกันโดยเราจะยกตัวอย่างการ Update ข้อมูลที่เป็นลิสต์จะต้องสร้างภาษา SPL โดยในคำสั่งข้างล่างจะเป็น โพรซีเจอร์ UpdateStudent การเปลี่ยนชื่อของนักศึกษาได้ดังนี้

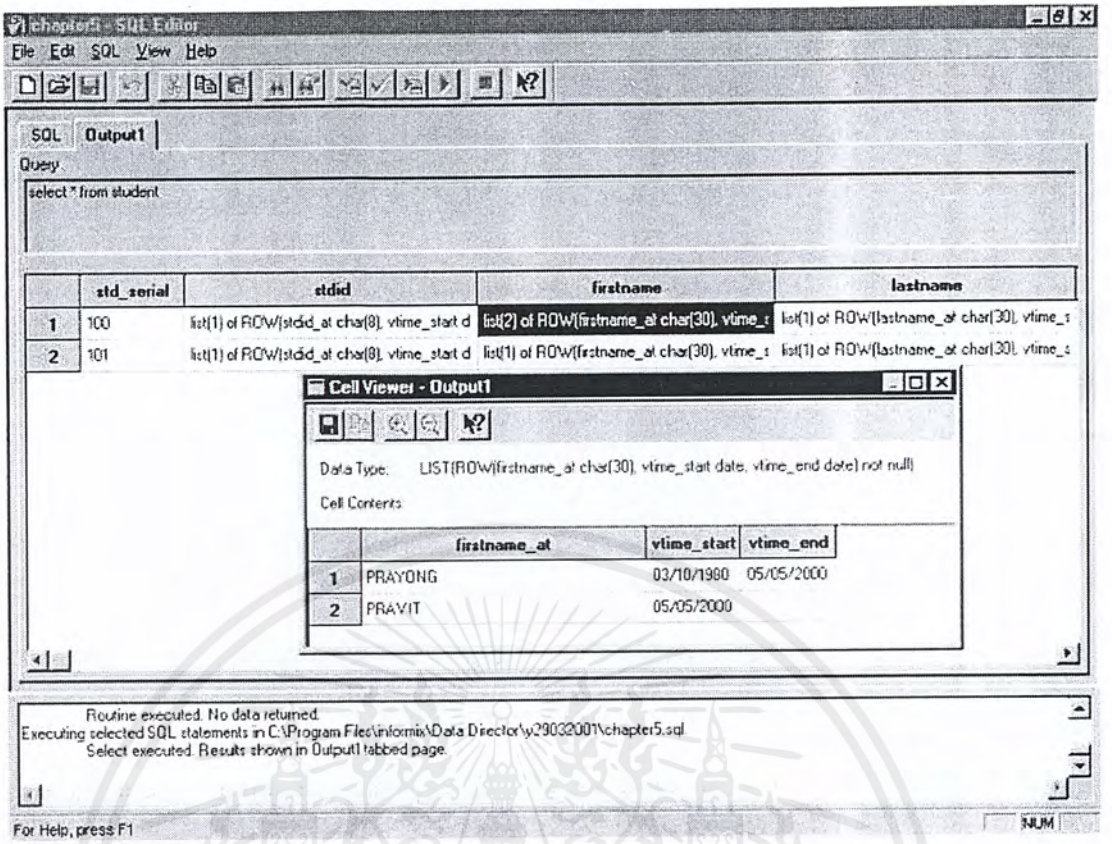
```
CREATE PROCEDURE UpdateStudent(data1 VARCHAR (8) ,ts DATE ,tp DATE,id INTEGER);
DEFINE  n smallint;
DEFINE thelist  LIST(ROW(name1 VARCHAR(8),vts1 DATE ,vte1 DATE) NOT NULL);
DEFINE therow     ROW(name2 VARCHAR (8),vts2 DATE ,vte2 DATE);
        SELECT firstname  INTO thelist  FROM Student
                WHERE std_serial = id;
        SELECT CARDINALITY(firstname) INTO n  FROM  student
                WHERE  std_serial =id;
        FOREACH cursor1 FOR
```

```
            SELECT * INTO therow  FROM TABLE(thelist)
            IF (therow.vte2 is NULL) THEN
                  let therow.vte2 =ts;
                  UPDATE TABLE(thelist)(x)  SET   x= therow
                        WHERE CURRENT OF cursor1;
                  EXIT FOREACH;
            END IF;
      END FOREACH
      LET n=n+1;
INSERT AT n  INTO TABLE(thelist) VALUES(Row(data1,ts,Null ) );
UPDATE Student SET  firstname= thelist
            WHERE  std_serial=id;
END PROCEDURE;
```

การทำงานของโพรซีเจอร์ UpdateStudent จะนับจำนวนข้อมูลในแอตทริบิวต์ Firstname ในตาราง STUDENT ที่เก็บเป็น List โดยจะใช้ฟังก์ชัน CARDINALITY ในการนับจำนวนข้อมูลที่อยู่ใน List แล้วค่อยทำการ Insert ข้อมูลเข้าไปที่ตำแหน่งถัดไป ในการจัดการ collection type ได้โดย select เอาคอลัมน์ที่เป็น collection type มาเก็บไว้ในตัวแปรที่เป็นตัวแปรที่เปรียบเสมือน Table ที่เป็นสมาชิกของ collection type แล้วค่อยใช้ Foreach เข้าไปจัดการกับสมาชิกแต่ล่ะตัวได้

- การเรียกใช้รูทีน UpdateStudent ได้ดังนี้

```
execute Procedure UpdateStudent('PRAVIT','05/05/2000',Null,'100');
```



รูปที่ 6-6 แสดงการเก็บข้อมูลชื่อนักศึกษาในตารางหลัง Update ชื่อนักศึกษาที่เก็บเป็น List

6.3 การค้นหาข้อมูลที่อยู่ใน List

ต่อไปนี้เป็นตัวอย่างการค้นหาข้อมูลชื่อปัจจุบันของนักศึกษาที่อยู่ใน List โดยการทำงานของโพรซีเจอร์ Findname จะค้นหาข้อมูลในแอตทริบิวต์ Firstname ที่เก็บเป็น List ในการจัดการ collection type ได้โดย select เอาคอลัมน์ที่เป็น collection type มาเก็บไว้ในตัวแปรที่เป็นตัวแปรที่เปรียบเสมือน Table ที่เป็นสมาชิกของ collection type แล้วค่อยใช้ Foreach เข้าไปจัดการกับสมาชิกแต่ละตัวได้แล้วค่อยตรวจสอบข้อมูลที่ Vtime_end ว่าเป็น null หรือไม่ ถ้าเป็น Null แสดงว่าข้อมูลนั้นเป็นข้อมูลชื่อปัจจุบันของนักศึกษา

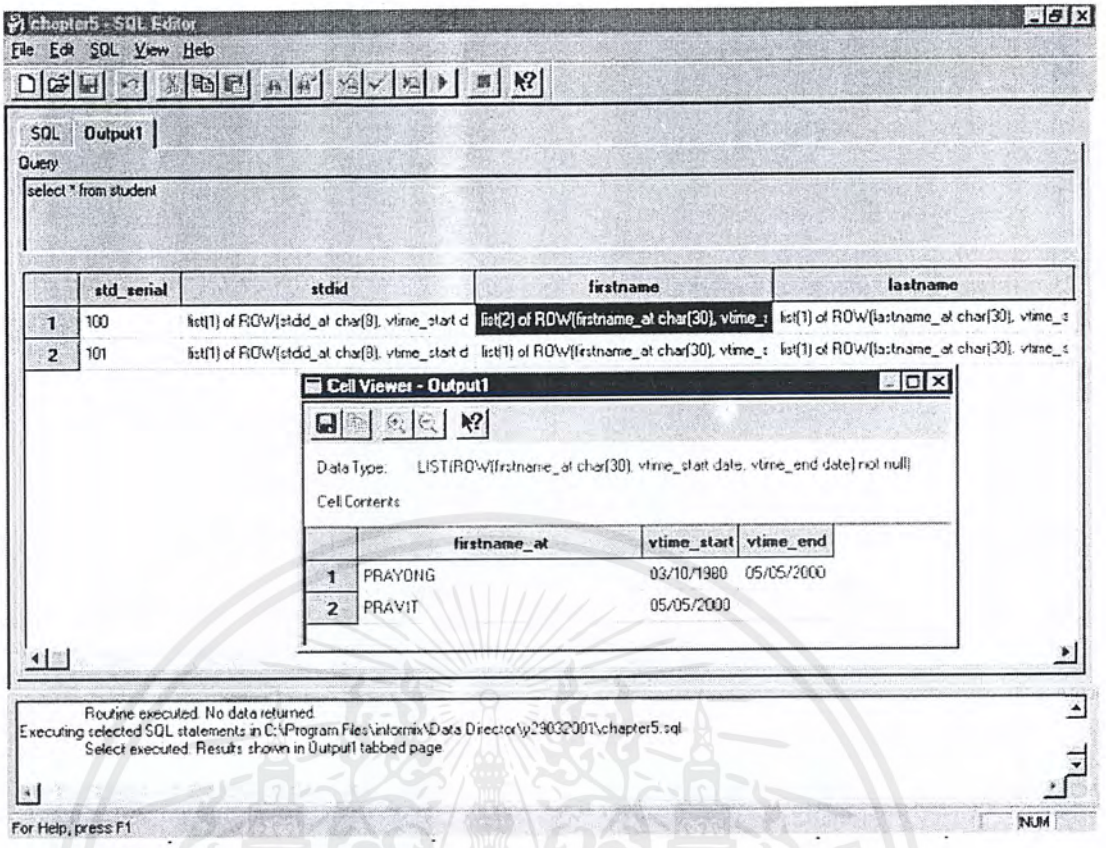
ผลลัพธ์จะนำไป insert ในตาราง result แล้วค่อย select เอาผลลัพธ์จากตารางอีกที ต่อไปนี้เป็นโพรซีเจอร์ค้นหาชื่อปัจจุบันของนักศึกษา

```
Create Procedure Findname(x_serial  integer);
define  name char(20);
define  n smallint;
define thelist  list(row(name1 varchar(30),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(30),vts2 date ,vte2 date);
        drop table result;
```

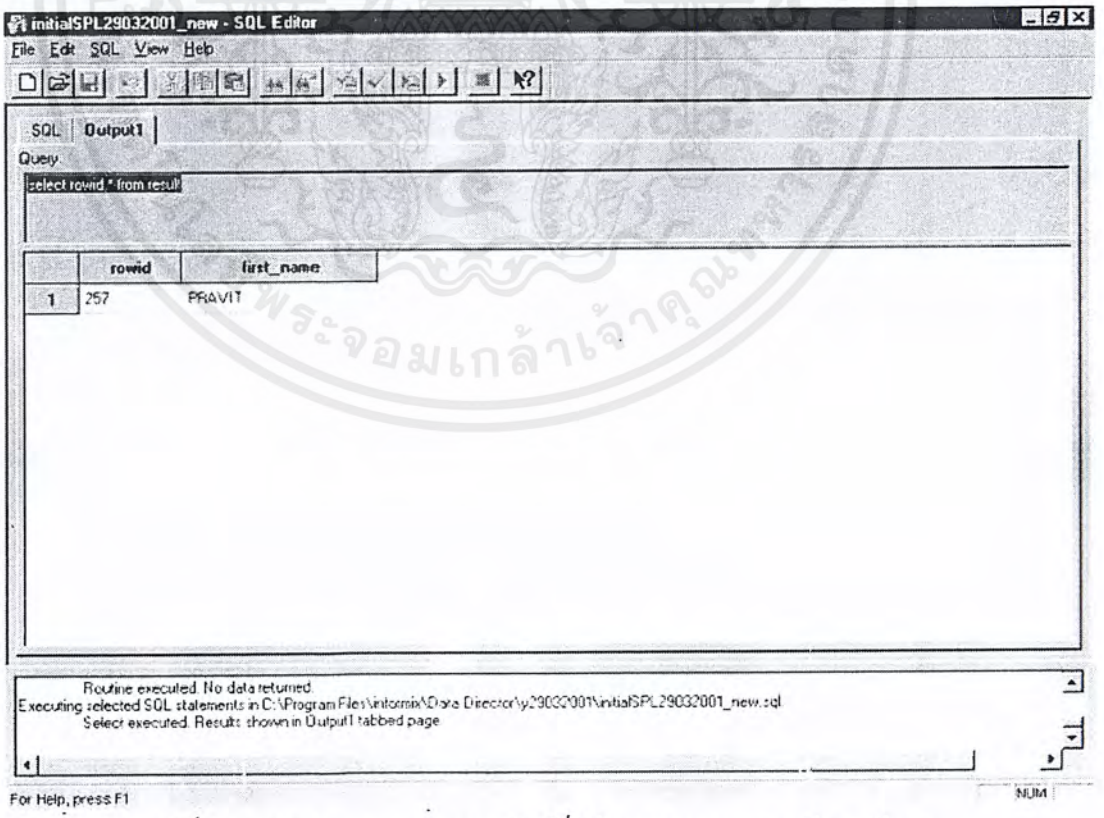
```
create table result(First_name char(20) );
SELECT Firstname INTO thelist  FROM Student
WHERE std_serial = x_serial;
Foreach cursor1 for
        select * into therow from table(thelist)
        if  (therow.vte2 is Null ) then
                LET name=therow.name2;
                insert into  result values (name);
        Exit Foreach;
        End if;
End Foreach
END PROCEDURE;
```

- การเรียกใช้รูทีน Findname ได้ดังนี้

```
execute Procedure findname(100);
```



รูปที่ 6-7 แสดงการเก็บข้อมูลชื่อนักศึกษาในตาราง ก่อนการค้นหา ชื่อนักศึกษาที่เก็บเป็น List



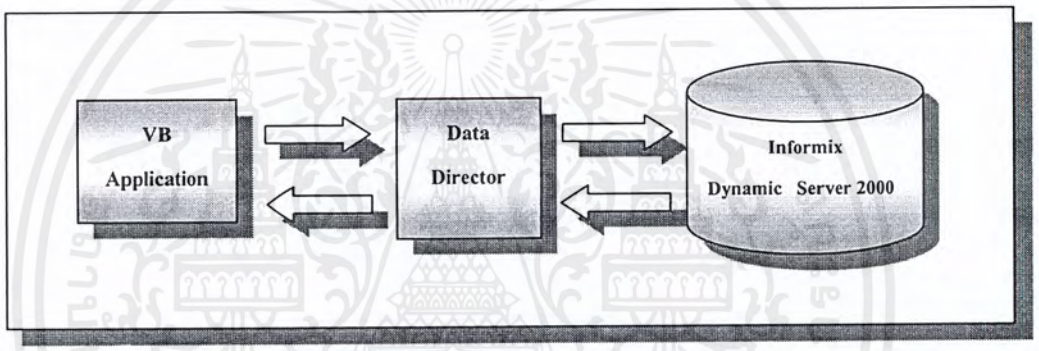
รูปที่ 6-8 แสดงการ select เอาผลลัพธ์ที่ได้จากตาราง result หลังการค้นหา

บทที่ 7
ตัวอย่างระบบสารสนเทศนักศึกษาเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์

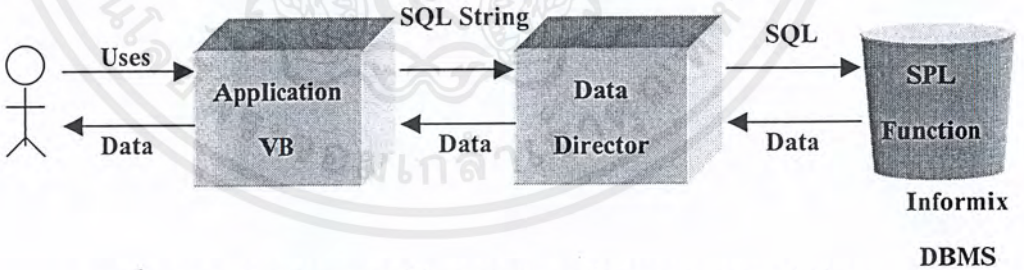
ในบทนี้จะกล่าวถึงการนำสิ่งต่างๆ ที่ได้ศึกษามาทำการสร้างโปรแกรมประยุกต์ซึ่งได้เลือกระบบสารสนเทศนักศึกษาซึ่งเป็นระบบสืบค้นข้อมูลนักศึกษามาเป็นกรณีศึกษา

7.1 สถาปัตยกรรมของระบบ

ในการพัฒนาโปรแกรมประยุกต์ระบบสารสนเทศนักศึกษามีโครงสร้างดังแสดงในรูปที่ 7-1 จากรูปโปรแกรมประยุกต์จะใช้วิชวลเบสิกในการพัฒนาและทำการติดต่อกับฐานข้อมูลโดยใช้ดาต้าไดเร็กเตอร์ซึ่งใช้หลักการของดาต้าไดเร็กเตอร์ออบเจ็คต์ (DDO) ในการติดต่อ ซึ่งดาต้าไดเร็กเตอร์นี้เป็นเครื่องมือที่ทางอินฟอร์มิกซ์ ได้จัดเตรียมไว้ให้ใช้ในการพัฒนาโปรแกรมประยุกต์ที่จะต้องทำงานติดต่อกับฐานข้อมูลอินฟอร์มิกซ์



รูปที่ 7-1 สถาปัตยกรรมของโปรแกรมประยุกต์ระบบสารสนเทศนักศึกษา



รูปที่ 7-2 แสดงการทำงานของโปรแกรมประยุกต์ระบบสารสนเทศนักศึกษา

จากรูปที่ 7-2 เป็นรูปแสดงการทำงานของโปรแกรมประยุกต์ระบบสารสนเทศนักศึกษาเริ่มจาก ผู้ใช้ (User) ใช้งานโปรแกรมผ่าน โปรแกรมประยุกต์ที่เขียนโดย Visual Basic หลังจากนั้นก็จะส่งคำสั่ง SQL ที่เป็นสตริงผ่านตัว Data Director ซึ่งตัว Data Director จะติดต่อฐานข้อมูลอินฟอร์มิกซ์ ให้เองโดยจะเรียกใช้ SPL ฟังก์ชันที่เขียนฝังไว้ที่เซิร์ฟเวอร์ หลังจากประมวลผลข้อมูลเสร็จก็จะส่งข้อมูลกลับมายัง Data Director และ Application VB แล้วก็ส่งต่อมาให้ผู้ใช้ตามลำดับ

7.2 อุปกรณ์ที่ใช้ Requirement และ Specification ของ Appliction

สำหรับโปรแกรมประยุกต์ที่ทำหน้าที่ในการสืบค้นข้อมูลนักศึกษาโดยสนับสนุนการทำงานแบบไคลเอนท์/เซิร์ฟเวอร์ โดยฝั่งไคลเอนท์จะพัฒนาโดยใช้วิชวลเบสิก (Visual Basic) และทางฝั่งดาต้าเบสเซิร์ฟเวอร์ใช้อินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ซึ่งการจัดการกับฐานข้อมูลจะพัฒนาโดยใช้โมเดลฐานข้อมูลเชิงวัตถุสัมพันธ์ แบ่งการทำงานออกดังนี้

7.2.1 เซิร์ฟเวอร์

โปรแกรมประยุกต์นี้จะทำการอ่านข้อมูลเพื่อจัดการทุกอย่างจากฝั่งเซิร์ฟเวอร์ โดยเราจะทำการสร้างเอสพีแอลรูทีน (SPL Routine) ซึ่งเขียนโดยใช้สตรอโปรซีเยอร์ของอินฟอร์มิกซ์ (Informix Stored Procedure Language) ซึ่งเป็นภาษาที่มีความสามารถของเอสคิวแอล (SQL) และเพิ่มความสามารถในการทำการคอนโทรลการทำงานลงไป (Flow Control) เช่น การทำลูป (Looping) หรือ การทำทางเลือก (Branching) เป็นตัวการในการตรวจสอบและจัดการกับข้อมูลให้กับไคลเอนท์

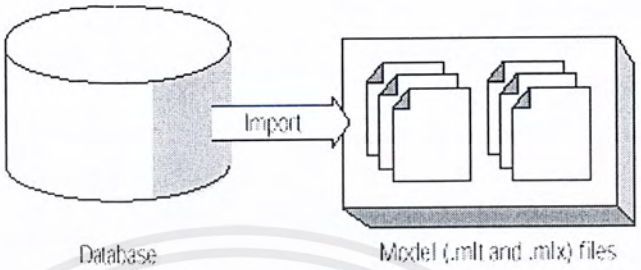
7.2.2 Data Director

การติดต่อระหว่างโปรแกรมประยุกต์กับฐานข้อมูลจะติดต่อผ่านดาต้าไดเรคเตอร์ (Data Director) ซึ่งจะเป็นตัวจัดการการติดต่อให้ ดาต้าไดเรคเตอร์เป็นทูลที่ทางอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์มีมาให้สำหรับการติดต่อระหว่างฐานข้อมูลกับวิชวลเบสิกโดยเฉพาะ ดาต้าไดเรคเตอร์จะช่วยให้การติดต่อกับฐานข้อมูลทำได้โดยง่าย ลดการเขียนโค้ดลงไปได้มาก อีกทั้งยังใช้งานง่ายเพียงลากแล้วปล่อยเม้าส์ลงบนคอนโทรลของวิชวลเบสิก (Drag-Drop) ก็สามารถนำข้อมูลจากฐานข้อมูลขึ้นมาแสดงผลได้

การติดต่อระหว่างโปรแกรมประยุกต์กับฐานข้อมูลจะติดต่อผ่านดาต้าไดเรคเตอร์ (Data Director) ดาต้าไดเรคเตอร์เป็นทูลที่ทางอินฟอร์มิกซ์ไดนามิกเซิร์ฟเวอร์ มีมาให้สำหรับการติดต่อระหว่างฐานข้อมูลกับวิชวลเบสิกโดยเฉพาะ การใช้งานดาต้าไดเร็คเตอร์จะมี 2 วิธี คือ การใช้งานแบบกราฟิกซึ่งเมื่อทำการติดตั้งดาต้าไดเร็คเตอร์ ตัวดาต้าไดเร็คเตอร์ คอมโพเนนท์ต่างๆจะถูกนำเข้าไปไว้ในวิชวลเบสิกทุกครั้งที่เรียกใช้จะปรากฏดาต้าไดเร็คเตอร์ขึ้นมาเสมอซึ่งจะช่วยให้เราสามารถนำข้อมูลขึ้นมาแสดงบนฟอร์มได้ง่ายโดยไม่ต้องมีการเขียนโค้ดใด ๆ เลย อีกวิธีเป็นการเรียกใช้ API ของดาต้าไดเร็คเตอร์ในการติดต่อกับฐานข้อมูล วิธีนี้จะต้องใช้กับการโปรแกรมมิ่งเท่านั้น

การทำงานของดาต้าไดเร็คเตอร์จะทำการอิมพอร์ตโมเดลของฐานข้อมูลเข้ามาซึ่งจะทำให้โปรแกรมประยุกต์รู้โครงสร้างภายในฐานข้อมูล โดยในการสร้างโปรแกรมประยุกต์เพื่อทำการติดต่อฐานข้อมูลหนึ่งจะต้องใช้งานดาต้าไดเร็คเตอร์ออบเจกต์ (Data Director Objects) หรือ ดีดีโอ (DDO) ซึ่งเราจะต้องทำการสร้างออบเจกต์ oEngine ให้กับทุก ๆ โปรแกรมประยุกต์ oEngine จะทำหน้าที่เป็นรูท (Root) ให้กับทุก ๆ ดีดีโอ แล้วทำการสร้างดาต้ากรุ๊ป (Datagroup) ซึ่งเป็นตัวที่จัดการรวมข้อมูลไว้ด้วยกัน โดยทำการสร้างออบเจกต์ oDatagroup ขึ้นมา

การใช้งานข้อมูลในฐานข้อมูล หากเราต้องการทำการคิวรีข้อมูลต้องทำการสร้างเวอร์ชวลเทเบิล (Virtual Table) เพื่อทำการเก็บผลที่ได้จากการคิวรีเราสามารถใส่คำสั่ง SQL ได้โดยตรง โดยการสร้างเวอร์ชวลเทเบิลจะต้องทำการสร้างออบเจกต์ oTable สำหรับการอิมพอร์ตโมเดลแสดงได้ดังรูปที่ 7-3



รูปที่ 7-3 การอิมพอร์ตโมเดล

7.3 สภาพแวดล้อมของระบบ

7.3.1 โปรแกรมที่ใช้ในการติดตั้งเพื่อทำงานกับฐานข้อมูล Informix 2000

Server

- Informix Dynamic Server.2000 Version 9.21.TC3

Client

- Database Administrator
- Informix Connect
- Informix Client Software Developer kits

โดยที่เครื่อง จะมีความต้องการของระบบ (System Requirements) ดังนี้

Requirement	**Windows NT**
CPU	Pentium processor or higher (200 megahertz recommended)
Windows Version	Windows NT, Version 4.0 or later (Windows NT Service Pack 3 required)
RAM	มีขนาด 64 megabytes เป็นอย่างน้อย
Hard-disk	ต้องมีพื้นที่ 200 megabytes เป็นอย่างน้อยและใช้พื้นที่ขนาด 20 megabytes สำหรับ เก็บ dbspaces Fat ต้องเป็น (NTFS)

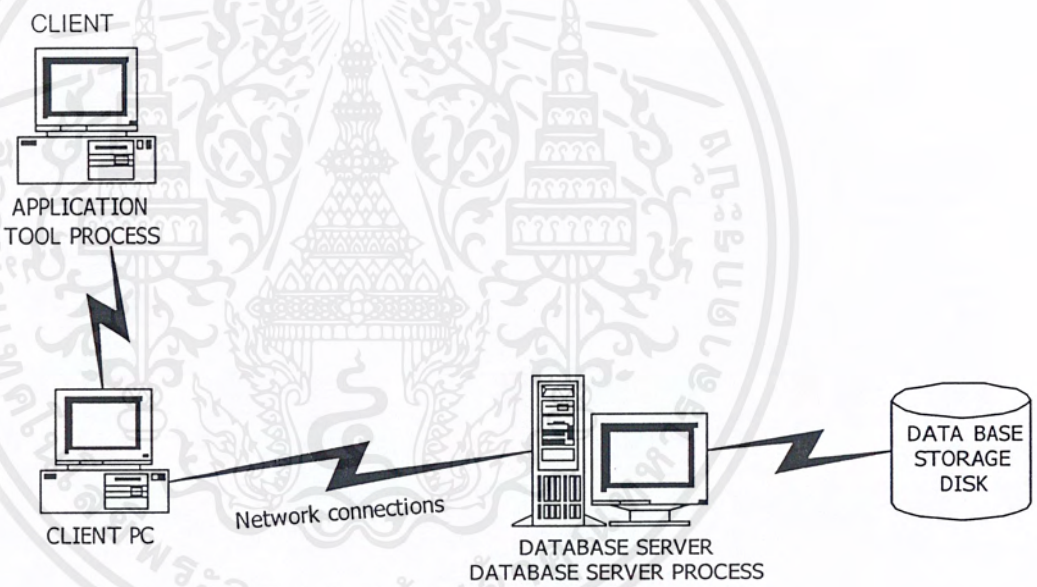
ตารางที่ 7-1 ความต้องการของระบบเพื่อทำงานกับฐานข้อมูล Informix 2000

<u>หมายเหตุ</u> ในการติดตั้งต้องเตรียม serial-number key ให้พร้อมด้วย

7.3.2 สถาปัตยกรรมของการเชื่อมต่อ Client และ Server

Client ของ Informix จะใช้สถาปัตยกรรมแบบ 2 กระบวนการ (Two – process Architecture) ซึ่งเหมาะสมสำหรับการเชื่อมต่อเครือข่ายแบบ Client/Server ดังแสดงตามรูปที่ 7-4

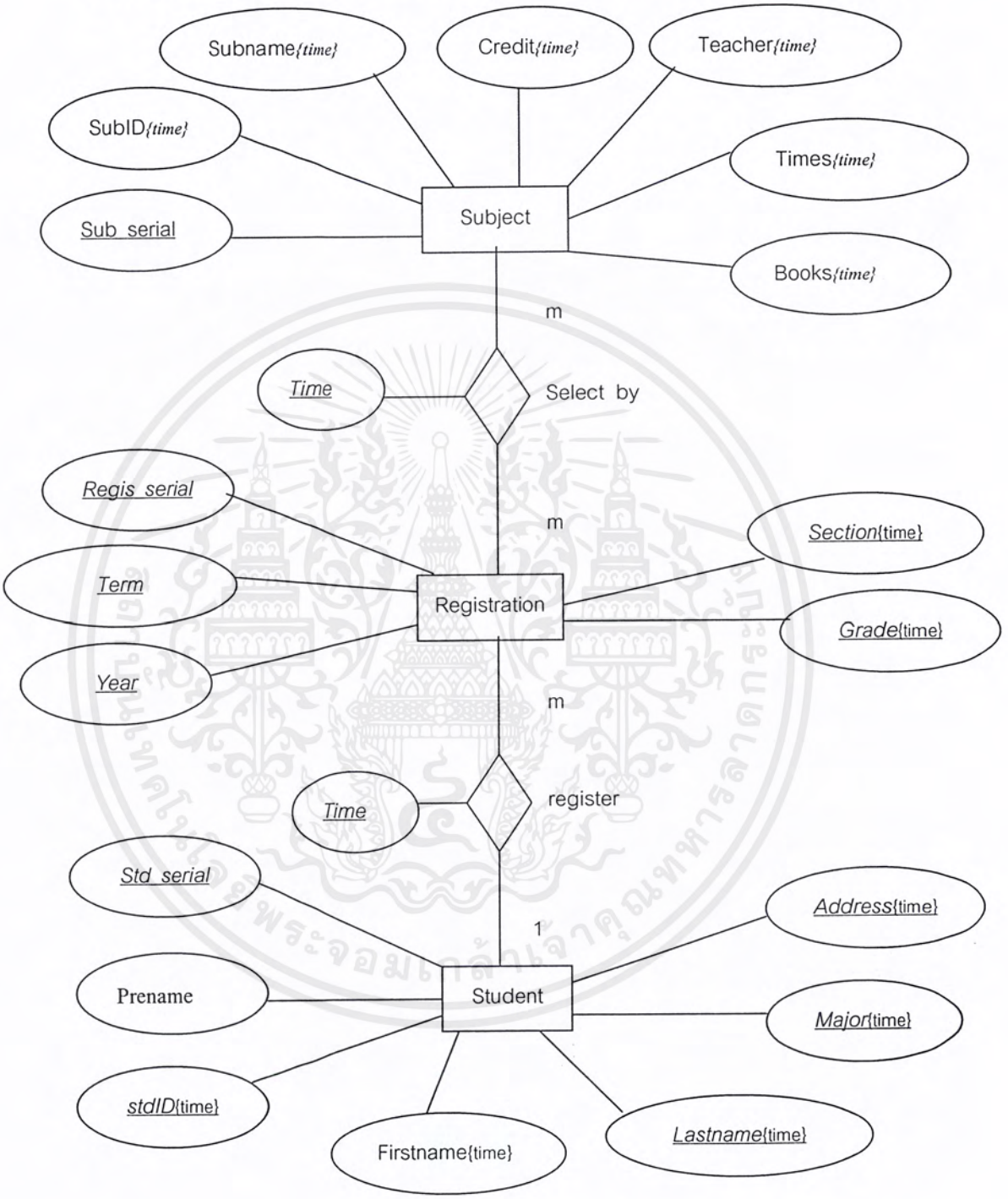
โดย Database Application จะทำงานเป็นโพรเซสเดียวบน PC ที่เป็น workstation และมีการสื่อสารผ่านเครือข่ายแยกกับโพรเซสของ Informix Database Server ที่ Host Machine



รูปที่ 7-4 สถาปัตยกรรมของการเชื่อมต่อ Client/Server

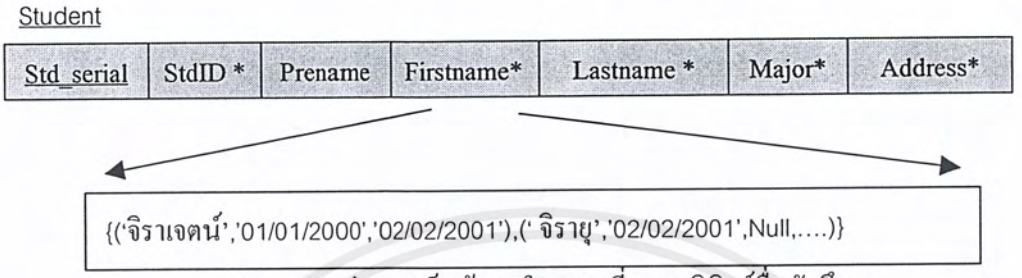
7.4 การออกแบบระบบ

7.4.1 สามารถเขียน ERT(Entity Relationship Time) Model ของระบบสืบค้นข้อมูลนักศึกษา

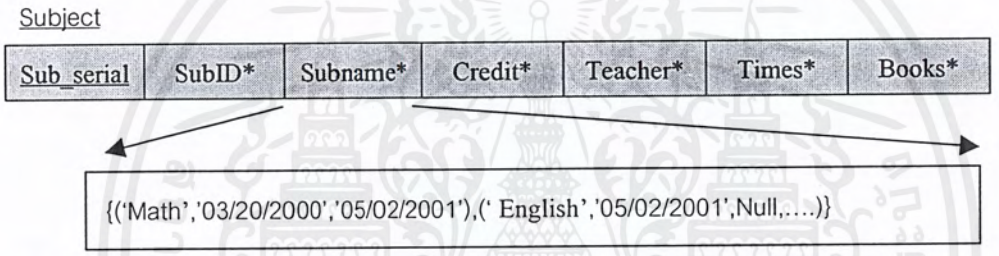


รูปที่ 7-5 ERT Model ของระบบสืบค้นข้อมูลนักศึกษา

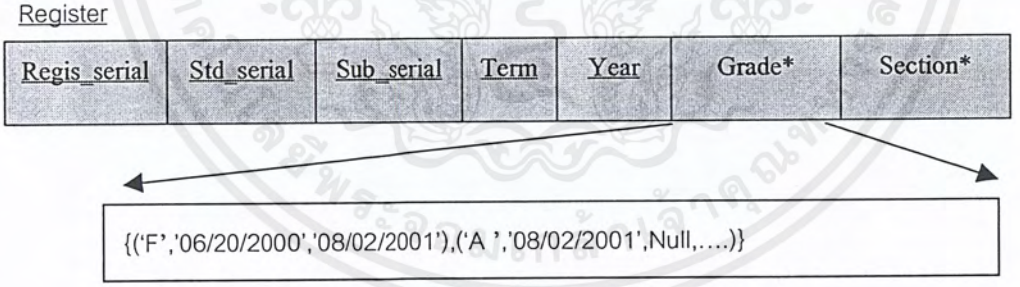
7.5 ตารางต่างๆที่ใช้ในระบบสืบค้นข้อมูลนักศึกษา



ตัวอย่างการเก็บข้อมูลใน List ที่แอตทริบิวต์ชื่อนักศึกษา



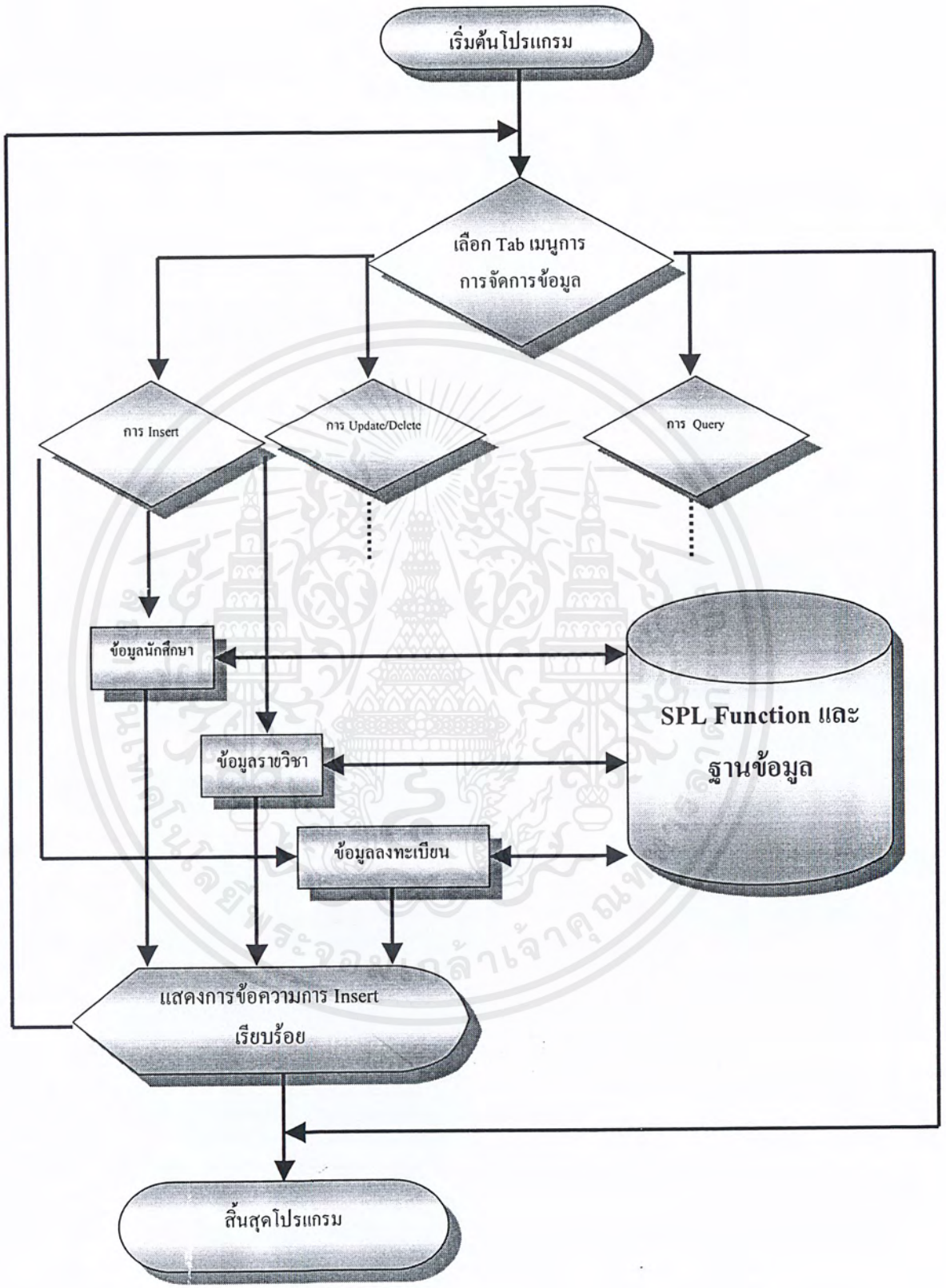
ตัวอย่างการเก็บข้อมูลใน List ที่แอตทริบิวต์ชื่อของรายวิชา



ตัวอย่างการเก็บข้อมูลใน List ที่แอตทริบิวต์ Grade

หมายเหตุ " เครื่องหมาย * หมายถึง แอตทริบิวต์นั้นมีการนำเวลาไปแปะไว้"

รูปที่ 7-6 จากตารางข้างบนเราจะกำหนดให้คอลัมน์เก็บเป็นกลุ่มของข้อมูลที่มี Valid time กำกับ



รูปที่ 7-7 แสดงโฟล์วชาร์ทการทำงานของระบบสืบค้นข้อมูลนักศึกษา

7.6 การออกแบบการจัดการกับดาต้าเบสเซิร์ฟเวอร์

การทำงานที่ฝั่งดาต้าเบสเซิร์ฟเวอร์นอกจากการจัดเก็บข้อมูลแล้วที่ฝั่งดาต้าเบสเซิร์ฟเวอร์จะมีการฝังรูทีนให้ฝั่งไคลเอนท์มาเรียกใช้ซึ่งรูทีนดังกล่าวเขียนโดยใช้ภาษา Stored Procedure ของอินฟอร์มิกซ์ซึ่งได้กล่าวมาแล้วในบทที่ผ่านมา โดยที่ฝั่งเซิร์ฟเวอร์จะประกอบไปด้วยรูทีนต่างๆดังนี้

	Procedure Findname(x_serial integer)
ชื่อรูทีน	Findname
หน้าที่	ทำการค้นหาข้อมูลชื่อของนักศึกษา เพื่อที่จะได้ชื่อนักศึกษาล่าสุด
ถูกเรียกใช้เมื่อ	ต้องการค้นหารายชื่อนักศึกษาปัจจุบันที่มี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการค้นหารายชื่อนักศึกษาปัจจุบันที่มี Std_serial เท่ากับ x_serial ในกรณีที่นักศึกษามีการเปลี่ยนชื่อมาแล้ว

	Function CurrentStudent(x_serial integer)
ชื่อรูทีน	CurrentStudent
หน้าที่	ทำการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบัน
ถูกเรียกใช้เมื่อ	ต้องการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา ประกอบไปด้วย รหัส คำนำหน้า ชื่อ นามสกุล สาขาวิชา ที่อยู่

	Function CurrentStudentQuery(x_serial integer)
ชื่อรูทีน	CurrentStudentQuery
หน้าที่	ทำการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบัน
ถูกเรียกใช้เมื่อ	ในส่วนของหน้าจอการ Query ข้อมูลนักศึกษาต้องการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา ประกอบไปด้วย รหัส คำนำหน้า ชื่อ นามสกุล สาขาวิชา ที่อยู่ โดยจะให้ค่ากลับมาอยู่ใน ตาราง Result

	Function CurrentSubject(x_serial integer)
ชื่อรูทีน	CurrentSubject
หน้าที่	ทำการค้นหาข้อมูลของรายวิชาทั้งหมดที่เป็นปัจจุบัน
ถูกเรียกใช้เมื่อ	ต้องการค้นหาข้อมูลของรายวิชาทั้งหมดที่เป็นปัจจุบันและมี Sub_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการค้นหาข้อมูลของรายวิชาทั้งหมดที่เป็นปัจจุบันและมี Sub_serial เท่ากับ x_serial ของนักศึกษา ประกอบไปด้วย รหัสรายวิชา ชื่อวิชา หน่วยกิจ อาจารย์ผู้สอน เวลาที่สอน หนังสือที่ใช้ โดยจะให้ค่ากลับมาอยู่ใน ตาราง Result

	Function CurrentSubjectQuery(x_serial integer)
ชื่อรูทีน	CurrentSubjectQuery
หน้าที่	ทำการค้นหาข้อมูลของรายวิชาทั้งหมดที่เป็นปัจจุบัน
ถูกเรียกใช้เมื่อ	ในส่วนของหน้าจอการ Query ข้อมูลนักศึกษาต้องการค้นหาข้อมูลของนักศึกษาทั้งหมดที่เป็นปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการค้นหาข้อมูลของรายวิชาทั้งหมดที่เป็นปัจจุบันและมี Sub_serial เท่ากับ x_serial ของนักศึกษา ประกอบไปด้วย รหัสรายวิชา ชื่อวิชา หน่วยกิจ อาจารย์ผู้สอน เวลาที่สอน หนังสือที่ใช้ โดยจะให้ค่ากลับมาอยู่ใน ตาราง Result

	Procedure UpdateStdID(data1 varchar(8) ,ts date ,tp date,x_serial varchar(5))
ชื่อรูทีน	UpdateStdID
หน้าที่	ทำการแก้ไขข้อมูลรหัสนักศึกษา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลรหัสนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการการแก้ไขข้อมูลรหัสนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษาโดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateFirstname(data1 varchar(30) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateFirstname
หน้าที่	ทำการแก้ไขข้อมูลชื่อนักศึกษา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลชื่อนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการการแก้ไขข้อมูลชื่อนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateLastname(data1 varchar(30) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateFirstname
หน้าที่	ทำการแก้ไขข้อมูลชื่อนักศึกษา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลชื่อนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการการแก้ไขข้อมูลชื่อนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateMajor(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateMajor
หน้าที่	ทำการแก้ไขข้อมูลสาขาวิชา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลสาขาวิชาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการการแก้ไขข้อมูลชื่อนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateAddress(data1 varchar(50) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateAddress
หน้าที่	ทำการแก้ไขข้อมูลที่อยู่ของนักศึกษา

ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลที่อยู่ของนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการการแก้ไขข้อมูลที่อยู่ของนักศึกษาให้เป็นข้อมูลปัจจุบันและมี Std_serial เท่ากับ x_serial ของนักศึกษา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateSubID(data1 varchar(30) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateSubID
หน้าที่	ทำการแก้ไขข้อมูลรหัสรายวิชา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลรหัสรายวิชาให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลรหัสรายวิชาให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชาโดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateSubname(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateSubname
หน้าที่	ทำการแก้ไขข้อมูลชื่อวิชา
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลชื่อวิชาให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลชื่อวิชาให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชาโดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateCredit(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateCredit
หน้าที่	ทำการแก้ไขข้อมูลหน่วยกิจ
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลหน่วยกิจให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลหน่วยกิจให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateTeacher(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateTeacher
หน้าที่	ทำการแก้ไขข้อมูลอาจารย์ผู้สอน
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลอาจารย์ผู้สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลอาจารย์ผู้สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateTimes(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateTimes
หน้าที่	ทำการแก้ไขข้อมูลเวลาที่สอน
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลเวลาที่สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลเวลาที่สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Procedure UpdateBooks(data1 varchar(40) ,ts date ,tp date,x_serial integer)
ชื่อรูทีน	UpdateBooks
หน้าที่	ทำการแก้ไขข้อมูลหนังสือที่ใช้สอน
ถูกเรียกใช้เมื่อ	ต้องการทำการแก้ไขข้อมูลหนังสือที่ใช้สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการการแก้ไขข้อมูลหนังสือที่ใช้สอนให้เป็นข้อมูลปัจจุบันและมี Sub_serial เท่ากับ x_serial ของรายวิชา โดย data1 เป็นข้อมูลใหม่ที่จะแก้ไข ts เป็นเวลาเริ่มต้น tp เป็นเวลาสิ้นสุด

	Function GetStdID(x_serial integer)
ชื่อรูทีน	GetStdID
หน้าที่	ทำการเอาข้อมูลรหัสของนักศึกษาออกมาจาก List ที่อยู่ในตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลรหัสของนักศึกษาออกมาจาก List และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการเอาข้อมูลรหัสของนักศึกษาออกมาจาก List มาเก็บไว้ในตาราง Listresult

	Function GetFirstname(x_serial integer)
ชื่อรูทีน	GetFirstname
หน้าที่	ทำการเอาข้อมูลชื่อของนักศึกษาออกมาจาก List ที่อยู่ในตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลชื่อของนักศึกษาออกมาจาก List และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการเอาข้อมูลชื่อของนักศึกษาออกมาจาก List มาเก็บไว้ในตาราง Listresult

	Function GetLastname(x_serial integer)
ชื่อรูทีน	GetLastname
หน้าที่	ทำการเอาข้อมูลนามสกุลของนักศึกษาออกมาจาก List ที่อยู่ในตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลนามสกุลของนักศึกษาออกมาจาก List และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการเอาข้อมูลนามสกุลของนักศึกษาออกมาจาก List มาเก็บไว้ในตาราง Listresult

	Function GetMajor(x_serial integer)
ชื่อรูทีน	GetMajor
หน้าที่	ทำการเอาข้อมูลสาขาวิชาของนักศึกษาออกมาจาก List ที่อยู่ในตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลสาขาวิชาของนักศึกษาออกมาจาก List และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการเอาข้อมูลสาขาวิชาของนักศึกษาออกมาจาก List มาเก็บไว้ในตาราง Listresult

	Function GetAddress(x_serial integer)
ชื่อรูทีน	GetAddress
หน้าที่	ทำการเอาข้อมูลที่อยู่ของนักศึกษาออกมาจาก List ที่อยู่ในตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลที่อยู่ของนักศึกษาออกมาจาก List และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการเอาข้อมูลที่อยู่ของนักศึกษาออกมาจาก List มาเก็บไว้ในตาราง Listresult

	Function GetSubID(x_serial integer)
ชื่อรูทีน	GetSubID
หน้าที่	ทำการเอาข้อมูลรหัสรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลรหัสรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลรหัสรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Function GetSubname(x_serial integer)
ชื่อรูทีน	GetSubname
หน้าที่	ทำการเอาข้อมูลชื่อรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลชื่อรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลชื่อรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Function GetCredit(x_serial integer)
ชื่อรูทีน	GetCredit
หน้าที่	ทำการเอาข้อมูลหน่วยกิจของรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลหน่วยกิจของรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลหน่วยกิจของรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Function GetTeacher(x_serial integer)
ชื่อรูทีน	GetTeacher
หน้าที่	ทำการเอาข้อมูลอาจารย์ผู้สอนของรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลอาจารย์ผู้สอนของรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลอาจารย์ผู้สอนของรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Function GetTimes(x_serial integer)
ชื่อรูทีน	GetTimes
หน้าที่	ทำการเอาข้อมูลเวลาที่สอนของรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลเวลาที่สอนของรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลเวลาที่สอนของรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Function GetBooks(x_serial integer)
ชื่อรูทีน	GetBooks
หน้าที่	ทำการเอาข้อมูลหนังสือที่ใช้สอนของรายวิชาออกมาจาก List ที่อยู่ในตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาข้อมูลหนังสือที่ใช้สอนของรายวิชาออกมาจาก List และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการเอาข้อมูลหนังสือที่ใช้สอนของรายวิชาที่มีการเปลี่ยนแปลงออกมาจาก List ที่อยู่ในแอตทริบิวต์มาเก็บไว้ในตาราง Listresult

	Procedure DeleteStudent(x_serial integer,vte date)
ชื่อรูทีน	DeleteStudent
หน้าที่	ทำการลบข้อมูลนักศึกษาออกจากตาราง Student
ถูกเรียกใช้เมื่อ	ต้องการเอาลบข้อมูลนักศึกษาออกจากตาราง Student และมี Std_serial เท่ากับ x_serial ของนักศึกษา
รายละเอียด	จะทำการลบข้อมูลนักศึกษาออกจากตาราง Student โดยการลบก็คือการ Update เวลาสิ้นสุดให้ มีค่าเท่ากับ vte

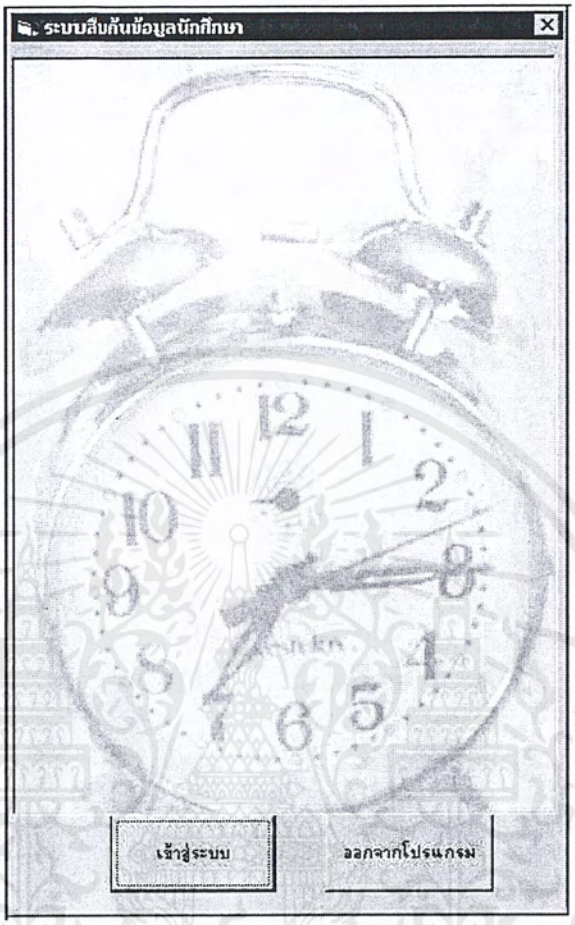
	Procedure DeleteSubject(x_serial integer,vte date)
ชื่อรูทีน	DeleteSubject
หน้าที่	ทำการลบข้อมูลรายวิชาออกจากตาราง Subject
ถูกเรียกใช้เมื่อ	ต้องการเอาลบข้อมูลรายวิชาออกจากตาราง Subject และมี Sub_serial เท่ากับ x_serial ของรายวิชา
รายละเอียด	จะทำการลบข้อมูลรายวิชาออกจากตาราง Student โดยการลบก็คือการ Update เวลาสิ้นสุดให้ มีค่าเท่ากับ vte

	Procedure UpdateGrade(data1 varchar(40) ,ts date ,tp date,xstd_serial integer,xsub_serial integer,term_serial integer,Year_serial integer)
ชื่อรูทีน	UpdateGrade
หน้าที่	ทำการแก้ไขข้อมูลเกรดในตาราง Register
ถูกเรียกใช้เมื่อ	ต้องการแก้ไขข้อมูลเกรดในตาราง Register และมี Std_serial เท่ากับ xstd_serial ,Sub_serial เท่ากับ xsub_serial
รายละเอียด	จะทำการแก้ไขข้อมูลเกรดในตาราง Register โดยค่า data1 จะเป็นข้อมูลที่ต้องการแก้ไข และค่า ts เป็นเวลาเริ่มต้นของค่านี้ ส่วน tp เป็นเวลาที่สิ้นสุดของข้อมูลเดิม

	Procedure UpdateSection(data1 varchar(40) ,ts date ,tp date,xstd_serial integer,xsub_serial integer,term_serial integer,Year_serial integer)
ชื่อรูทีน	UpdateSection
หน้าที่	ทำการแก้ไขข้อมูล Section ในตาราง Register
ถูกเรียกใช้เมื่อ	ต้องการแก้ไขข้อมูล Sectionในตาราง Register และมี Std_serial เท่ากับ xstd_serial ,Sub_serial เท่ากับ xsub_serial
รายละเอียด	จะทำการแก้ไขข้อมูล Sectionในตาราง Register โดยค่า data1 จะเป็นข้อมูลที่ต้องการแก้ไข และค่า ts เป็นเวลาเริ่มต้นของค่านี้ ส่วน tp เป็นเวลาที่สิ้นสุดของข้อมูลเดิม

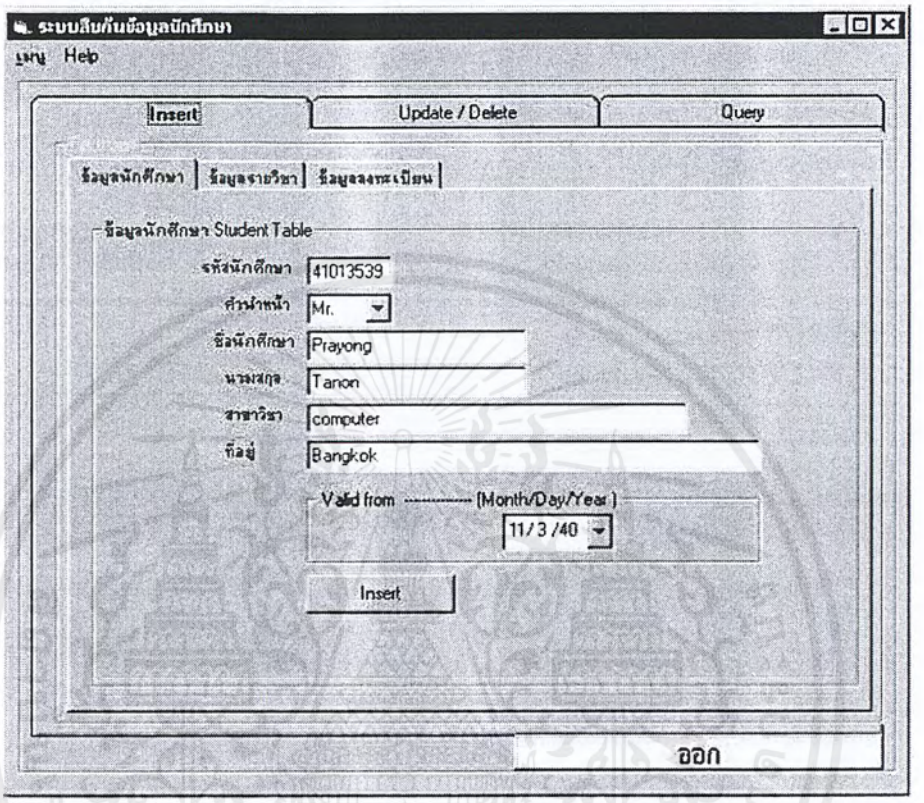
7.7 ตัวอย่างโปรแกรมระบบสืบค้นข้อมูลนักศึกษา

เมื่อเปิดโปรแกรมเข้าสู่ระบบสืบค้นข้อมูลนักศึกษาก็จะปรากฏดังรูปที่ 7-8



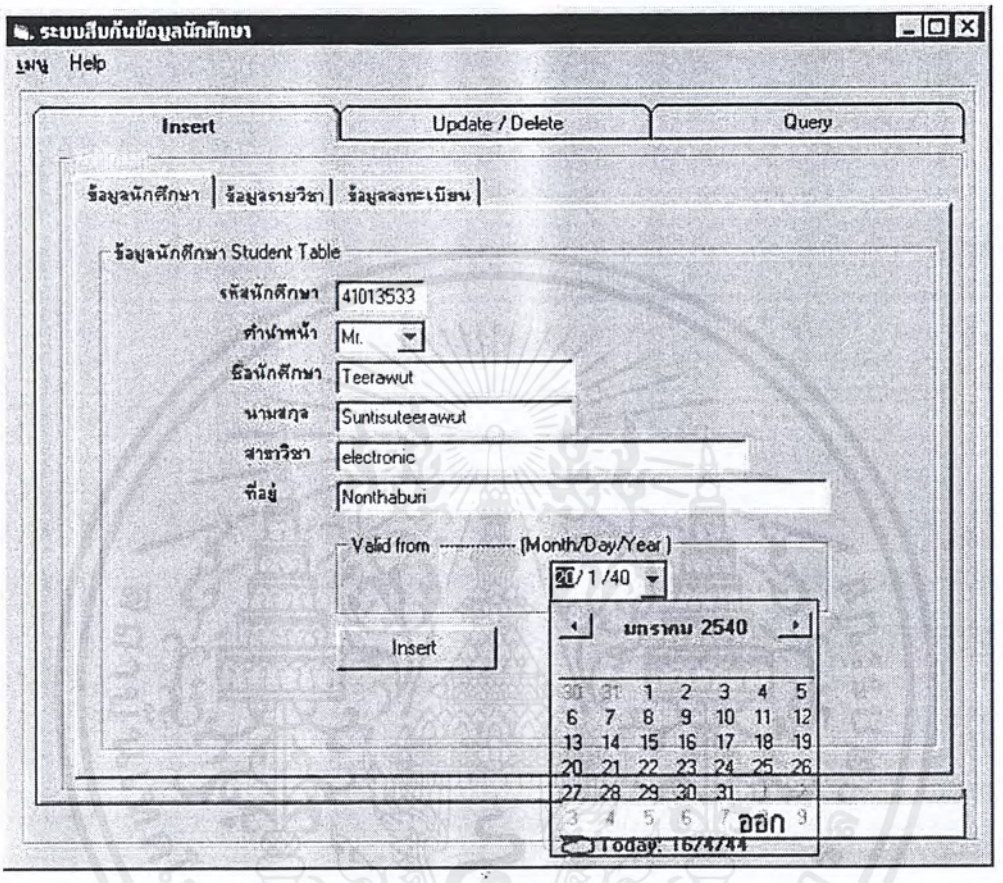
รูปที่ 7-8 แสดงโปรแกรมระบบสืบค้นข้อมูลนักศึกษา

หลังจากเข้าโปรแกรมแล้วก็จะประกอบด้วยฟังก์ชันการทำงาน 3 ส่วนด้วยกันคือส่วนของการ Insert Update/Delete และ Query แสดงดังรูปที่ 7-9



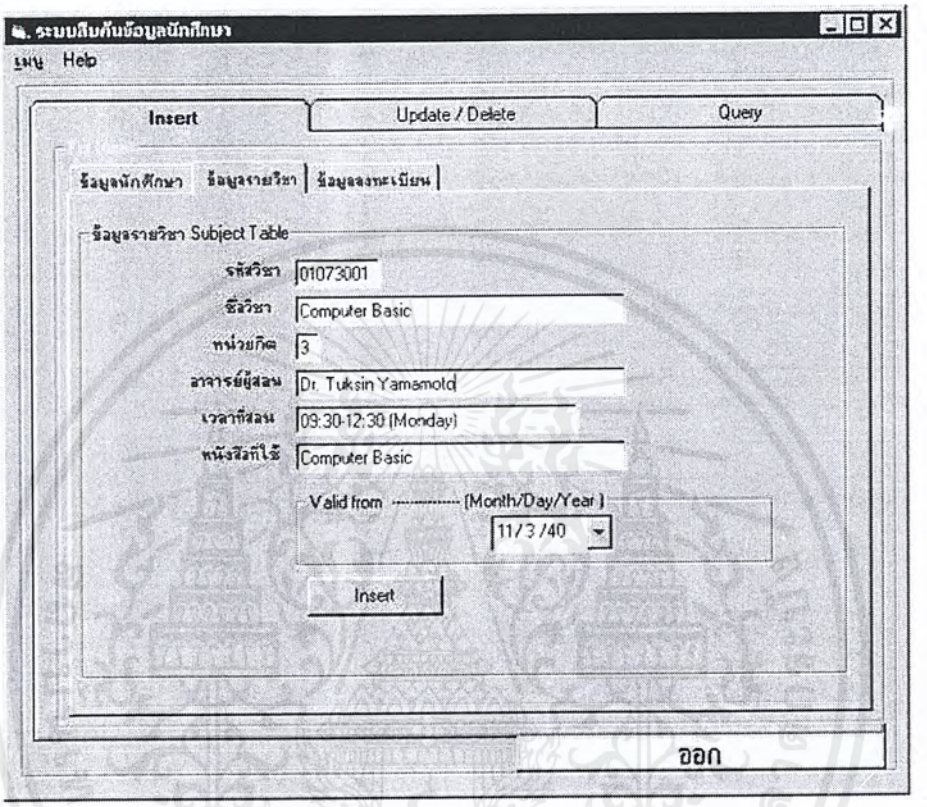
รูปที่ 7-9 แสดงโปรแกรมระบบสืบค้นข้อมูลนักศึกษา insert ข้อมูลนักศึกษา

หลังจากใส่ข้อมูลนักศึกษาแล้วก็ทำการเลือกเวลาที่ช่องของ Valid from ซึ่งเป็นการกำหนดค่าของเวลาที่ข้อมูลนั้นเป็นจริง แล้วก็คลิกที่ปุ่ม Insert จะแสดงดังรูปที่ 7-10



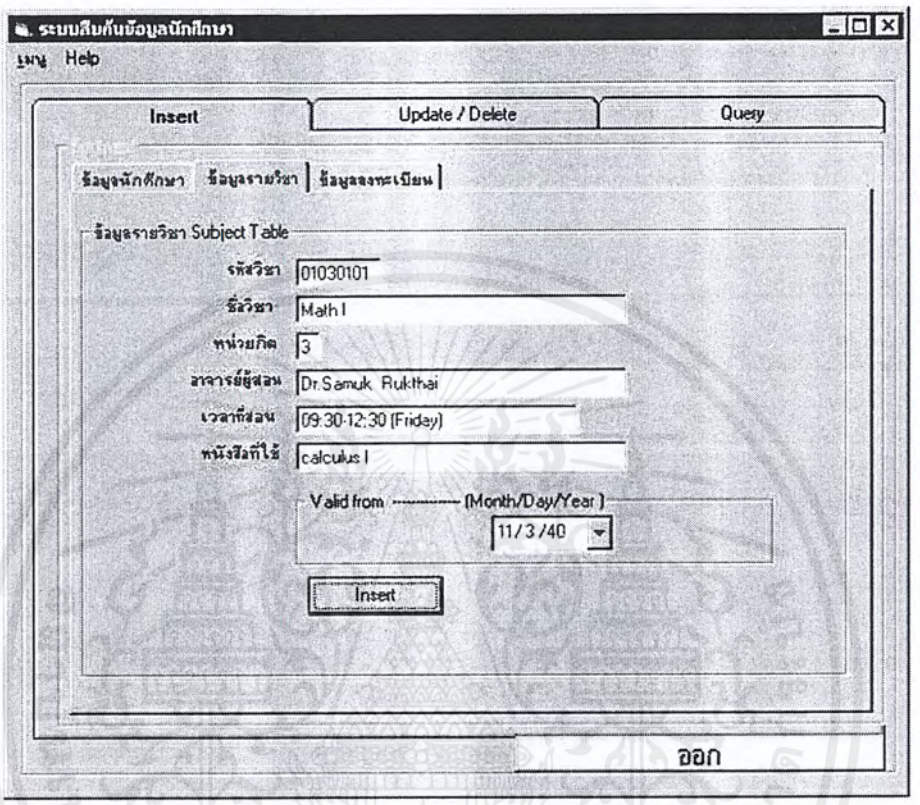
รูปที่ 7-10 แสดงโปรแกรมระบบสืบค้นข้อมูลนักศึกษา ขณะเลือกเวลา

เมื่อจะ Insert ข้อมูลรายวิชาสามารถทำได้โดยการคลิกที่ tab ของข้อมูลรายวิชาแล้วก็ป้อนข้อมูลรายวิชาลงในช่องว่าง โดยจะทำคล้ายกับการ Insert ข้อมูลนักศึกษา จะแสดงได้ดังรูปที่ 7-11



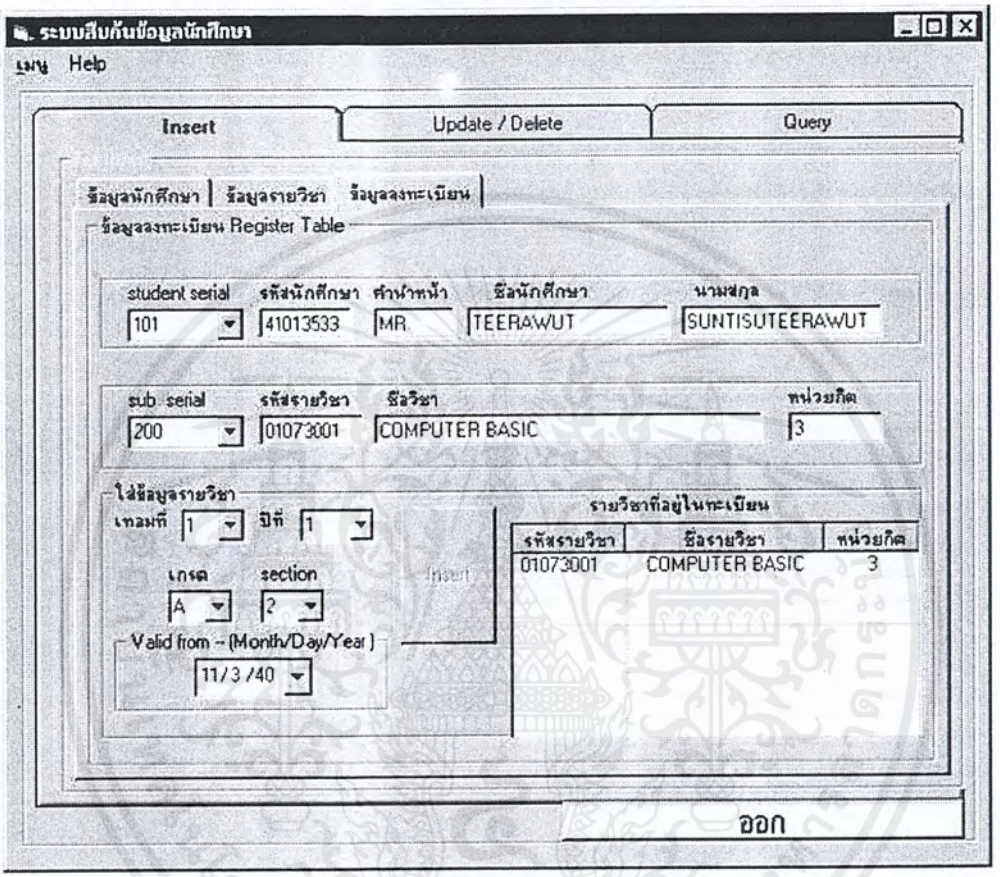
รูปที่ 7-11 แสดงหน้าจอการ Insert ข้อมูลรายวิชาที่หนึ่ง

จากรูปที่ 7-12 เป็นรูปแสดงการป้อนข้อมูลรายวิชาเป็นวิชาที่สองซึ่งเป็นวิชาใหม่ ชื่อ Math แสดงได้ดังรูป



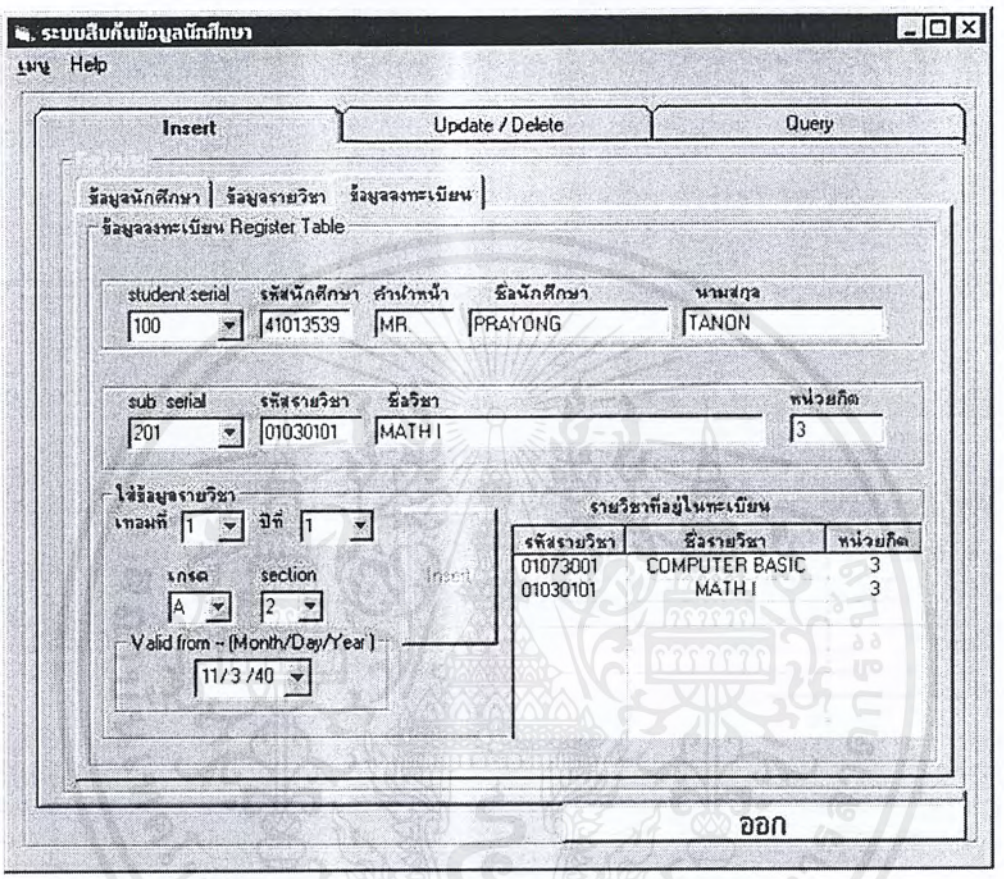
รูปที่ 7-12 แสดงหน้าจอของการ insert ข้อมูลรายวิชาที่สอง

ในการ Insert ข้อมูลการลงทะเบียนของนักศึกษาชื่อ TEERAWUT ลงทะเบียนวิชา COMPUTER BASIC โดยสามารถคลิก Mouse เลือก serial ของนักศึกษาได้ที่ช่อง student_serial ในทำนองเดียวกันก็สามารถเลือกรายวิชาที่จะลงได้โดยการคลิก Mouse เลือกที่ sub_serial จะแสดงดังรูปที่ 7-13



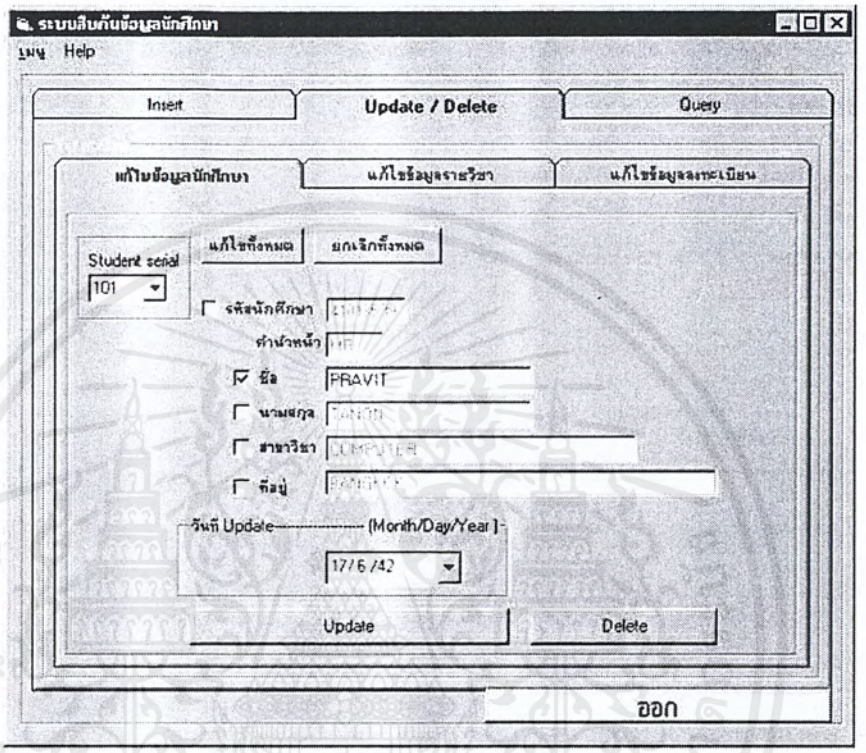
รูปที่ 7-13 แสดงหน้าจอการ insert ข้อมูลลงทะเบียนคนที่หนึ่ง

เมื่อเราต้องการ Insert ข้อมูลการลงทะเบียนของนักศึกษาคนใหม่ก็สามารถทำได้โดยการคลิก Mouse เปลี่ยนที่ student_serial จะแสดงดังรูปที่ 7-14



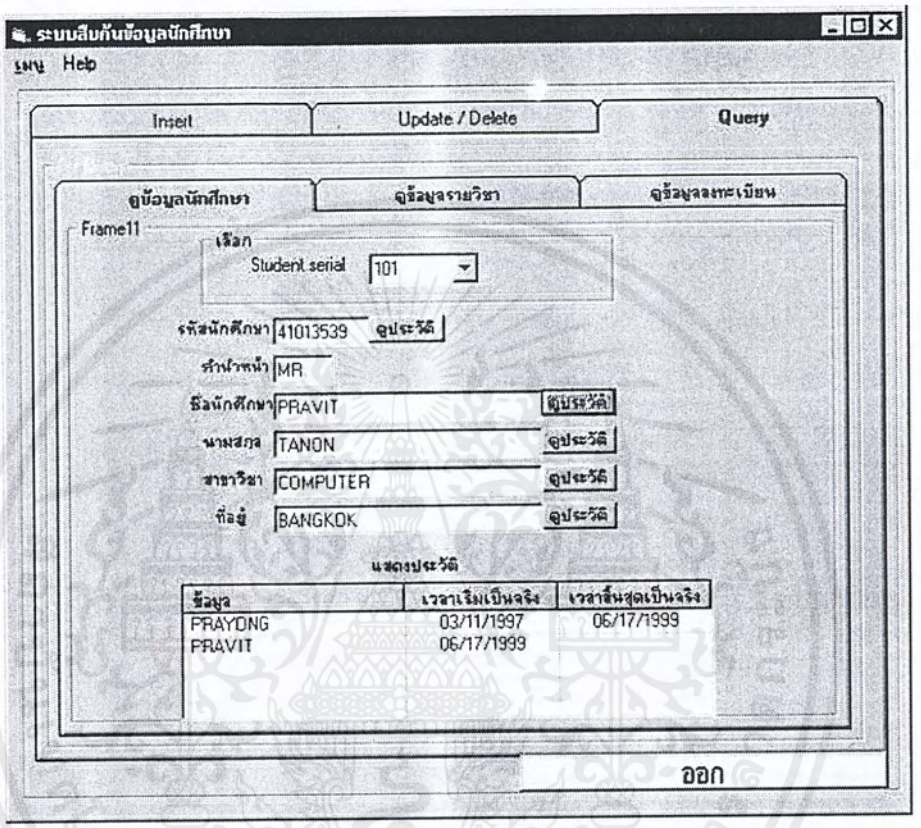
รูปที่ 7-14 แสดงหน้าจอการ insert ข้อมูลลงทะเบียนคนที่สอง

เมื่อข้อมูลนักศึกษามีการเปลี่ยนแปลงเช่น ชื่อเปลี่ยน สามารถเลือกแก้ไขข้อมูลของนักศึกษา ได้โดยคลิกเลือกตรงช่องที่จะแก้ไข จากรูปจะเปลี่ยนชื่อจาก PRAYONG เป็น PRAVIT จะแสดงได้รูปที่ 7-15



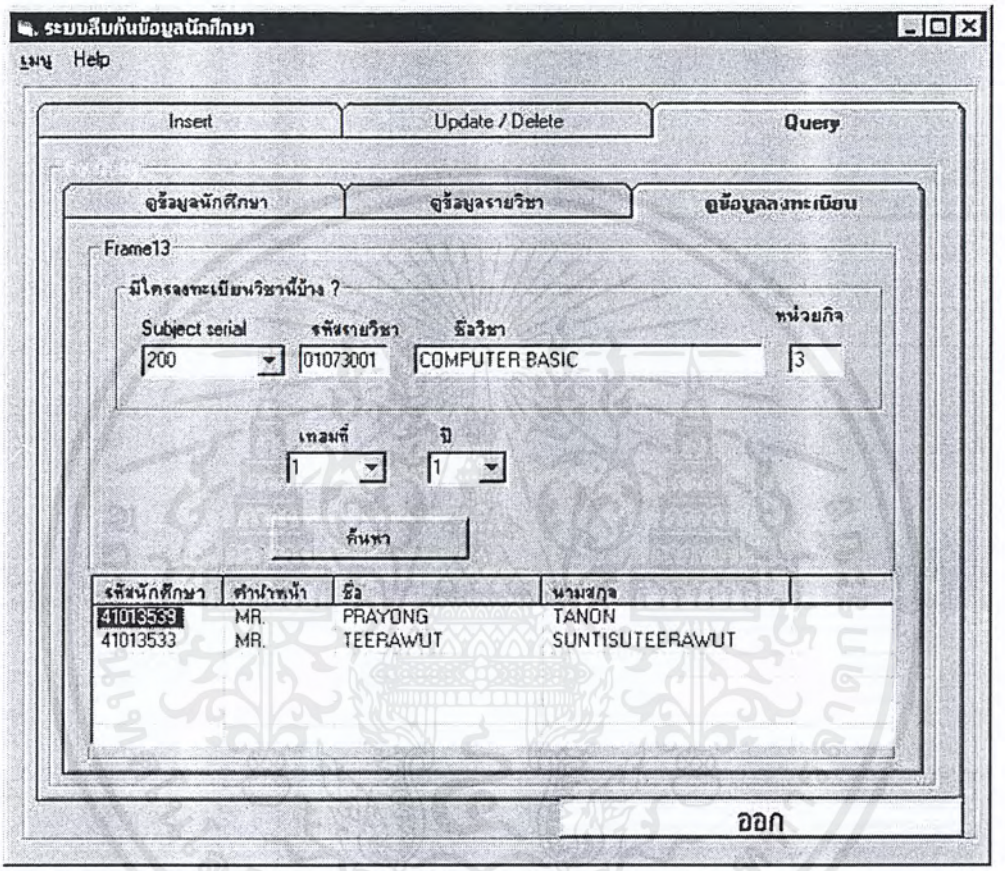
รูปที่ 7-15 แสดงหน้าจอการ Update ชื่อนักศึกษา

รูปที่ 7-16 เมื่อจะ Query ดูข้อมูลของนักศึกษาเช่นดู ประวัติการเปลี่ยนชื่อ ก็สามารถคลิกดูได้ที่ปุ่มดูประวัติ ส่วนการ Query ดูประวัติของข้อมูลรายวิชาก็สามารถทำได้เช่นเดียวกัน



รูปที่ 7-16 แสดงหน้าจอการ Query ดูประวัติการเปลี่ยนชื่อของนักศึกษา

สำหรับการ Query ดูข้อมูลการลงทะเบียนก็จะเป็นตัวอย่างคำถามว่า มีใครลงทะเบียนวิชานี้บ้าง โดยการคลิก Subject serial เพื่อเลือกรายวิชา จะสามารถแสดงได้ดังรูปที่ 7-17



รูปที่ 7-17 แสดงหน้าจอการ Query ดูข้อมูลลงทะเบียน

บทที่ 8
สรุปและวิจารณ์

8.1 สรุปผลการวิจัย

จากการศึกษาและการทำวิจัยโครงงานนี้ ได้พบว่าการนำเสนอการแปะเวลาไว้ที่ Tuple ของตารางนั้น มีข้อดีในเรื่องของภาษาที่ใช้เขียนจะไม่ยุ่งยากซับซ้อน แต่ข้อเสียก็คือเราไม่สามารถทราบได้เลยว่าคอลัมน์ไหนมีการเปลี่ยนแปลงข้อมูลไปบ้างในโครงงานนี้จึงได้นำเสนอการแปะเวลาไว้ที่คอลัมน์เพื่อแก้ปัญหาดังกล่าวข้างต้น โดยเราได้นำหลักการของฐานข้อมูลเชิงวัตถุสัมพันธ์มาใช้กับข้อมูลเชิงเวลา เนื่องจากคุณลักษณะของฐานข้อมูลที่มีการนำเสนอแนวคิดเชิงวัตถุมาใช้ โดยจะมองแต่ละคอลัมน์เป็นออบเจ็กต์ หรือเป็นคอลัมน์ที่เป็นแอตทริบิวต์ที่มีหลายค่า (Collection Data Type) ซึ่งอาจกำหนดให้เป็น Set Multiset หรือ List ทำให้มีความสามารถในการเก็บข้อมูลที่มีการเปลี่ยนแปลงตามเวลาไว้ในคอลัมน์เดียวได้ เสมือนหนึ่งว่าเอาแอตทริบิวต์ของช่วงเวลา (Timestamp)ไปเกาะติดกับแอตทริบิวต์ของข้อมูลนั้น

แต่ข้อเสียของแปะเวลาไว้ที่คอลัมน์คือจะมีปัญหาเรื่องภาษาที่ใช้ในการประมวลผลข้อมูลเชิงเวลานั้นยังมีความยุ่งยากอยู่เพราะ DBMS นั้นยังไม่สนับสนุนข้อมูลเชิงเวลาอย่างเต็มที่ ผู้เขียนโปรแกรมจะต้องจัดการสร้างภาษาที่มาสนับสนุนการจัดการกับข้อมูลเชิงเวลาเอง โดยต้องใช้ภาษา Stored Procedural Language (SPL) มาทำการประมวลผลที่ไว้ที่ Database Server เพื่อช่วยในการทำ Insert Delete Update Query กับข้อมูลเชิงเวลาได้อย่างถูกต้องตามหลักการ

จากการศึกษาโครงงานนี้ได้ทำการสร้างโปรแกรมประยุกต์กับข้อมูลเชิงเวลาบนฐานข้อมูลเชิงวัตถุสัมพันธ์ และได้นำเวลามาแปะเวลาไว้ที่คอลัมน์ โดยใช้ Infomix Dynamic Server 2000 มาเป็นตัวจัดการกับข้อมูลเชิงเวลาพบว่า ข้อมูลชนิด Serial นั้นมีคุณลักษณะที่เหมือนกับรหัสเฉพาะ Object Identification (Oid) ที่มีอยู่ใน Object Relational Database และได้นำข้อมูลชนิด Serial มาใช้เป็น Primary Key เพราะว่าข้อมูลชนิด Serial นั้นมีคุณลักษณะที่ Unique เนื่องจากข้อมูลชนิด Serial จะมีค่าเพียงค่าเดียวเดียวเท่านั้น หากว่าทำการลบค่าของข้อมูลชนิด Serial นั้นออกไปจาก Tuple นั้นแล้วก็ไม่สามารถนำค่าข้อมูลชนิด Serial ที่ทำการลบไปแล้วมาใช้อีกได้ เพราะค่าข้อมูลชนิด Serial จะทำงานโดยการเพิ่มค่าตัวเองขึ้นทีละ 1 จึงเป็นคุณสมบัติที่เด่นที่มีการทำงานคล้ายคลึงกับ Oid และนำมาประยุกต์ใช้ตามทฤษฎี Temporal Database ได้อย่างมีประสิทธิภาพ

8.2 แนวทางในการพัฒนาต่อ

สำหรับแนวทางการพัฒนาต่อนั้นผู้ที่จะพัฒนาต่อควรจะศึกษาหลักทฤษฎีที่เกี่ยวข้องให้ดีเสียก่อน โดยเฉพาะในภาษา SQL ใน Informix นั้นไม่ว่าจะเป็นการ Insert Update Delete การนำเอาเวลาไปแปะไว้ที่คอลัมน์ในภาษา SQL ควรจะเขียนอย่างไร เป็นต้นรวมถึงภาษา Stored Procedural Language (SPL) ที่ฝังไว้ที่ Database Server มีการวิธีการเขียนการเรียกใช้งานอย่างไร

โปรแกรมโดยใช้ Visual Basic ติดต่อกับฐานข้อมูลที่ใช้ดาต้าไดเร็กเตอร์เป็นตัวเชื่อมต่อควรจะมีการใช้ดาต้าไดเร็กเตอร์ที่สนับสนุนภาษาไทยเพื่อความสมบูรณ์ในการพัฒนาโปรแกรมสำหรับคนไทย เนื่องจากในปัจจุบันนี้การพัฒนาโปรแกรมที่ใช้งานกันบนอินเตอร์เน็ตมีความนิยมและแพร่หลายเป็นอย่างมาก ดังนั้นแนวทางในการพัฒนาต่อไปควรจะทำให้แอพพลิเคชันสามารถใช้งานบนอินเตอร์เน็ตได้

ในด้านภาษาที่ใช้จัดการกับกลุ่มของข้อมูลเช่น Set, List, Multiset นั้นควรจะพัฒนาให้มีประสิทธิภาพมากกว่านี้เช่นอาจจะสร้างเป็นฟังก์ชั่นหรือเป็นโมดูลที่ทำงานให้อัตโนมัติเพื่อลดความสลับซับซ้อนลงจะทำให้งานวิจัยทางด้านนี้ได้ประสิทธิภาพมาก

ภาคผนวก

ภาคผนวก ก

การติดตั้ง Informix Dynamic Server.2000 บน Windows NT

1. การเตรียมการติดตั้ง

ในการติดตั้งบน Windows NT ก่อนอื่นผู้ที่ทำการติดตั้งจะต้องอยู่ใน Group ของ Administrators ก่อนซึ่งมีความสามารถในการติดตั้ง database server โดยความต้องการของระบบมีดังนี้

- Windows NT, version 4.0 มี Service Pack 3 หรือมากกว่า
- TCP/IP
- ขนาดของหน่วยความจำ(RAM) และ swap files รวมอย่างน้อย 64 เมกะไบต์
- FAT ของฮาร์ดดิสก์ที่ลง Windows NT ต้องเป็น NTFS
- มีหน่วยความจำอย่างน้อย 16 เมกะไบต์ ยิ่งมากยิ่งดี
- พื้นที่ของฮาร์ดดิสก์อย่างน้อย 140 เมกะไบต์(จะใช้ 30 เมกะไบต์สำหรับ root dbspace และ ใช้ 20 เมกะไบต์ สำหรับ additional dbspace)

ในขณะทำการติดตั้งสามารถที่จะกำหนดขนาดของพื้นที่ของดิสก์ตามความต้องการได้

2. ลำดับของการติดตั้ง

การติดตั้งจะต้องทำการติดตั้ง database server ก่อนค่อยติดตั้ง Products อื่นๆซึ่งมีลำดับดังต่อไปนี้

2.1 Install Dynamic Server

2.2 Install Informix tools

2.3 Install DataBlade modules

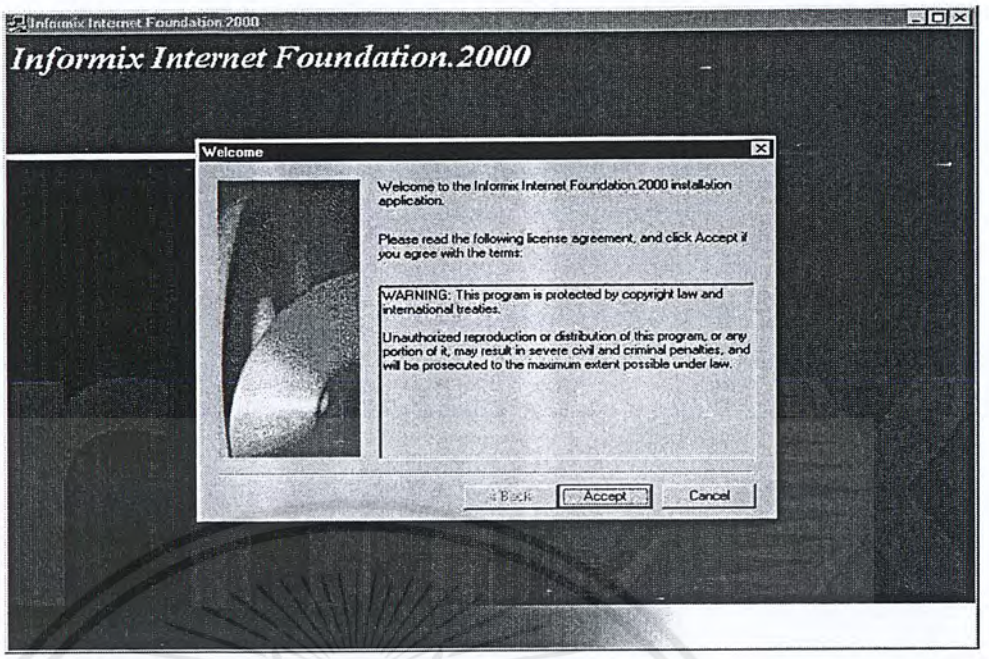
2.4 Install client products เช่น Install Client SDK หรือ Informix Connect

ในการติดตั้งหลาย Product ถ้าติดตั้ง Product หนึ่งเสร็จแล้วเมื่อคุณเริ่มติดตั้ง Product ตัวต่อไป คุณไม่ควรที่จะ load files จาก Informix Product ตัวอื่นจากเครื่องคอมพิวเตอร์จนกว่าการติดตั้งปัจจุบันจะเสร็จสมบูรณ์

3. ขั้นตอนการติดตั้ง

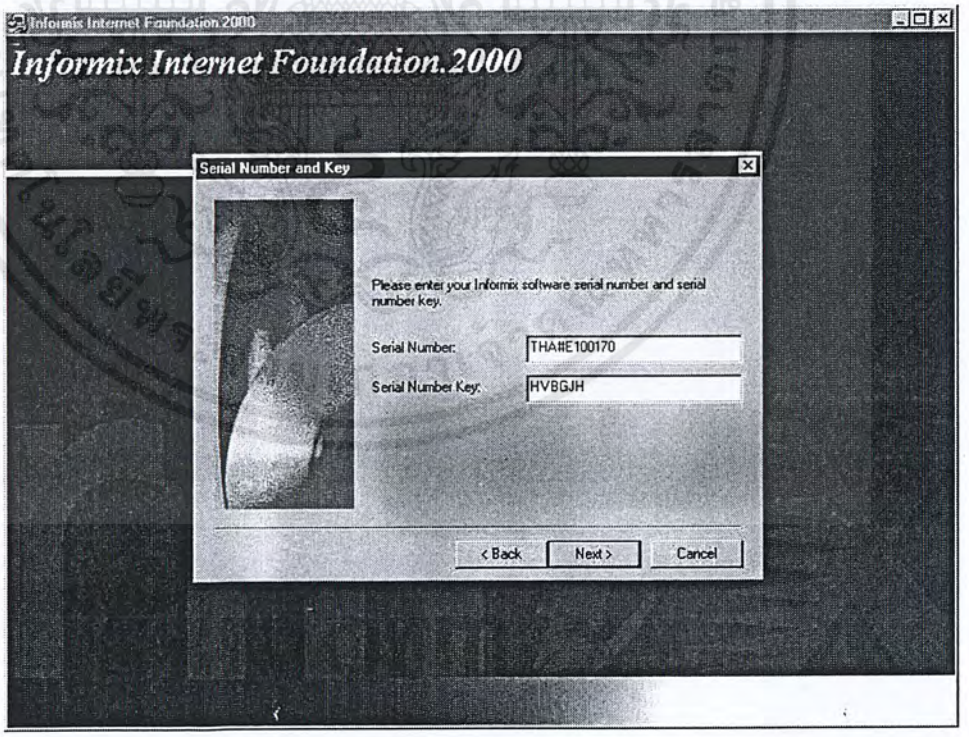
3.1 ผู้ติดตั้งต้องอยู่ในกลุ่มของ Administration Group

3.2 Run Setup.exe เพื่อทำการติดตั้ง



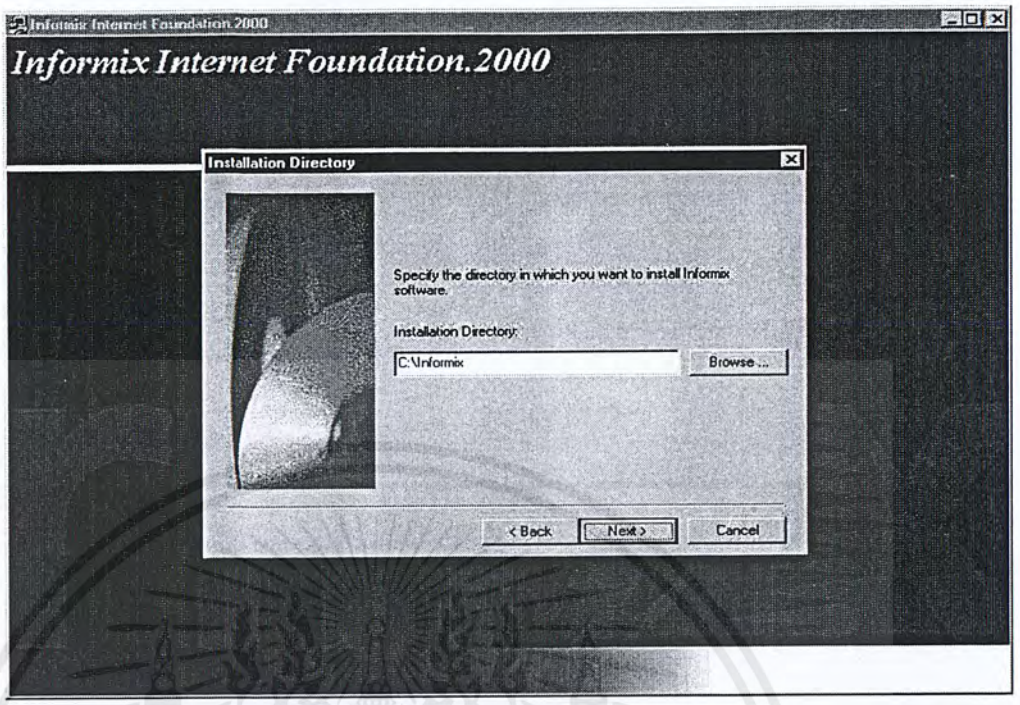
รูปที่ ก-1 แสดงการติดตั้ง Informix Dynamic Server.2000

3.3 ใส่ Serial Number และ Serial Number Key



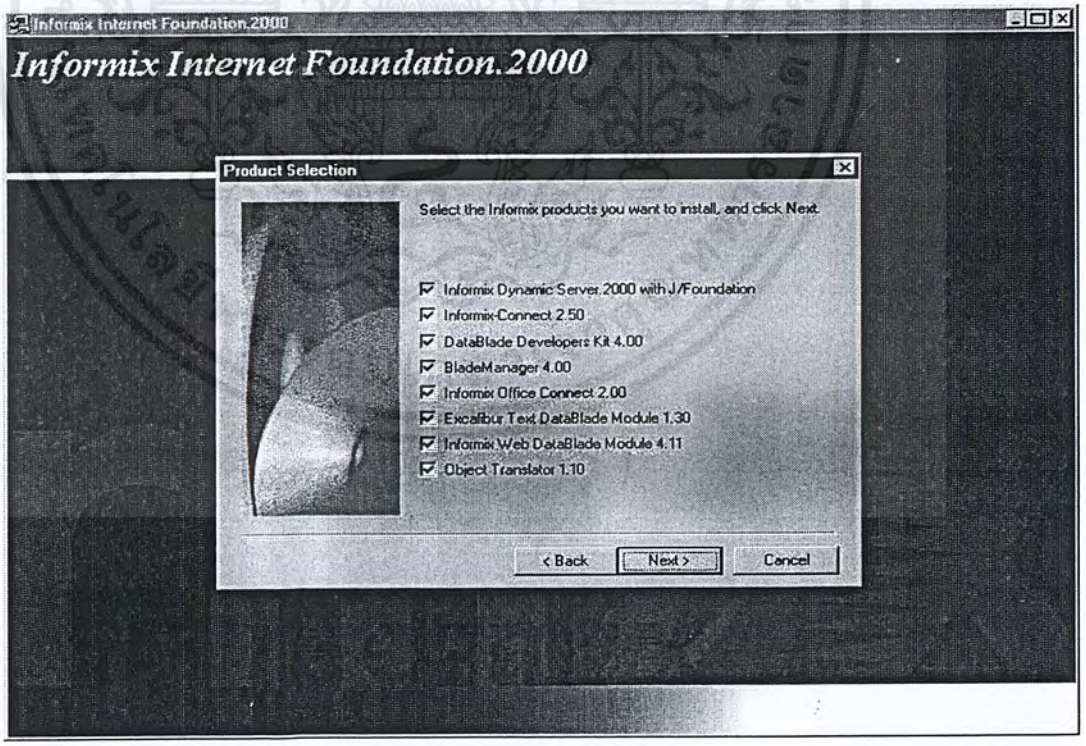
รูปที่ ก-2 แสดงการใส่ Serial Number และ Serial Number Key

3.4 เลือกตำแหน่งดิสก์ที่จะทำการติดตั้ง



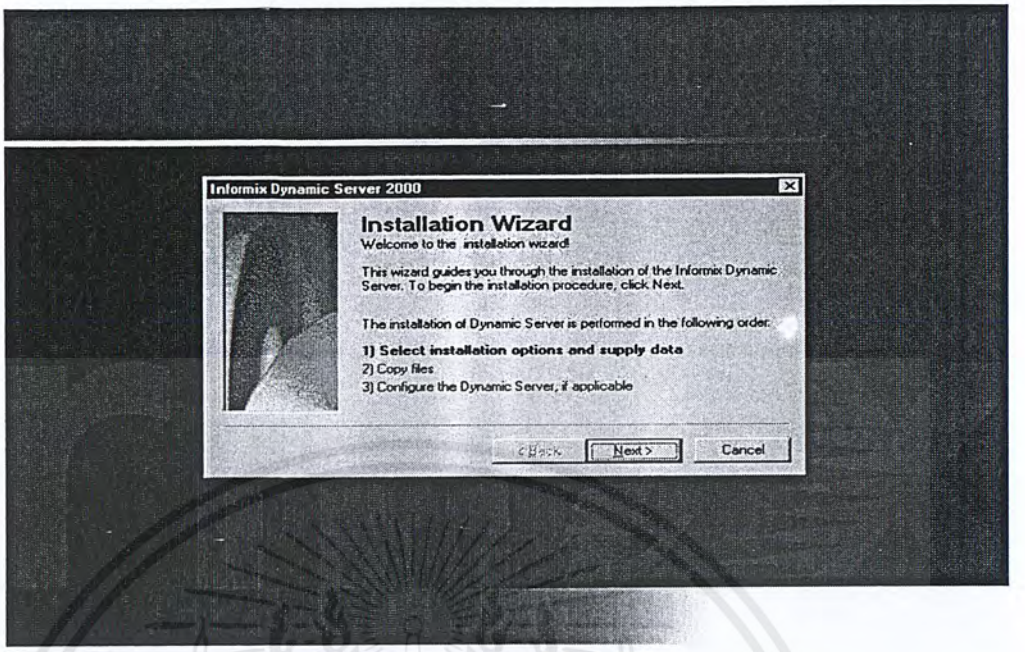
รูปที่ ก-3 แสดงการเลือกตำแหน่งดิสก์ที่จะทำการติดตั้ง

3.5 เลือก Informix Products ที่จะติดตั้ง



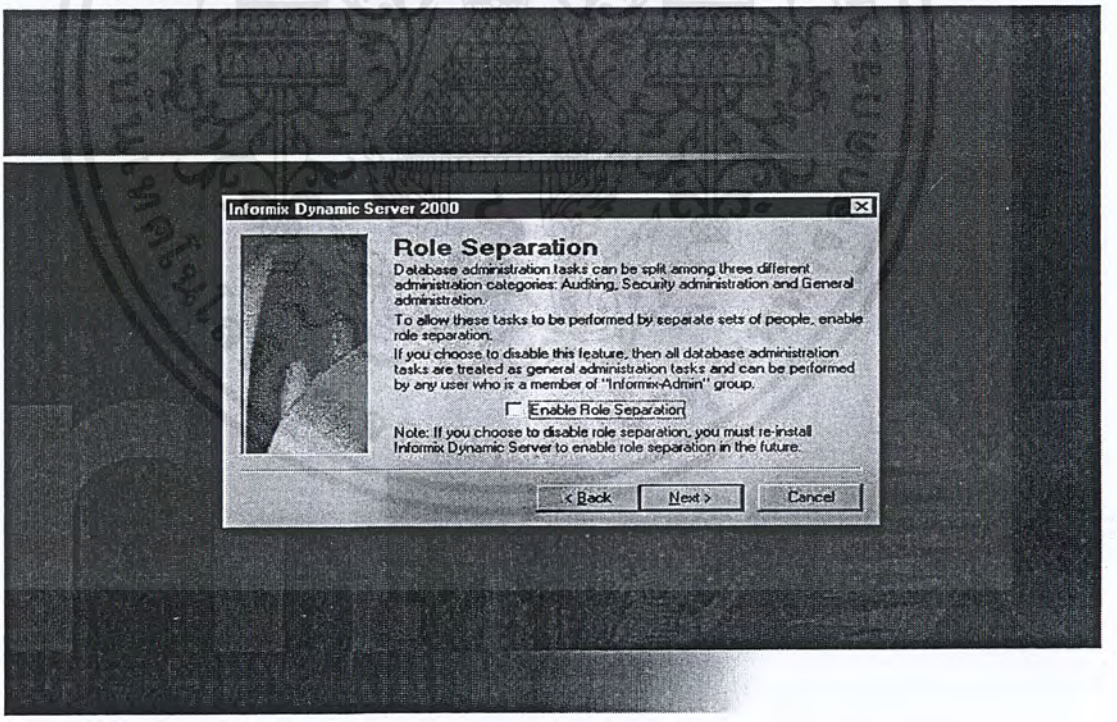
รูปที่ ก-4 แสดงการเลือก Informix Products ที่ต้องการติดตั้ง

3.6 กด Next เพื่อทำการ Install



รูปที่ ก-5 แสดงการยืนยันการ install

3.7 เลือกการกำหนดบทบาท ของ DB-Admin



รูปที่ ก-6 แสดงการการกำหนดบทบาท ของ DB-Admin

เลือกการกำหนดบทบาท ของ DB-Admin ว่าจะแยกหน้าที่กันในแต่ละงาน หรือ รวมกันใช้เพียง Account เดียว หากเลือกที่จะแยกบทบาทในการดูแลระบบจะต้องปฏิบัติตามขั้นตอนต่อไปนี้เพิ่มเติม

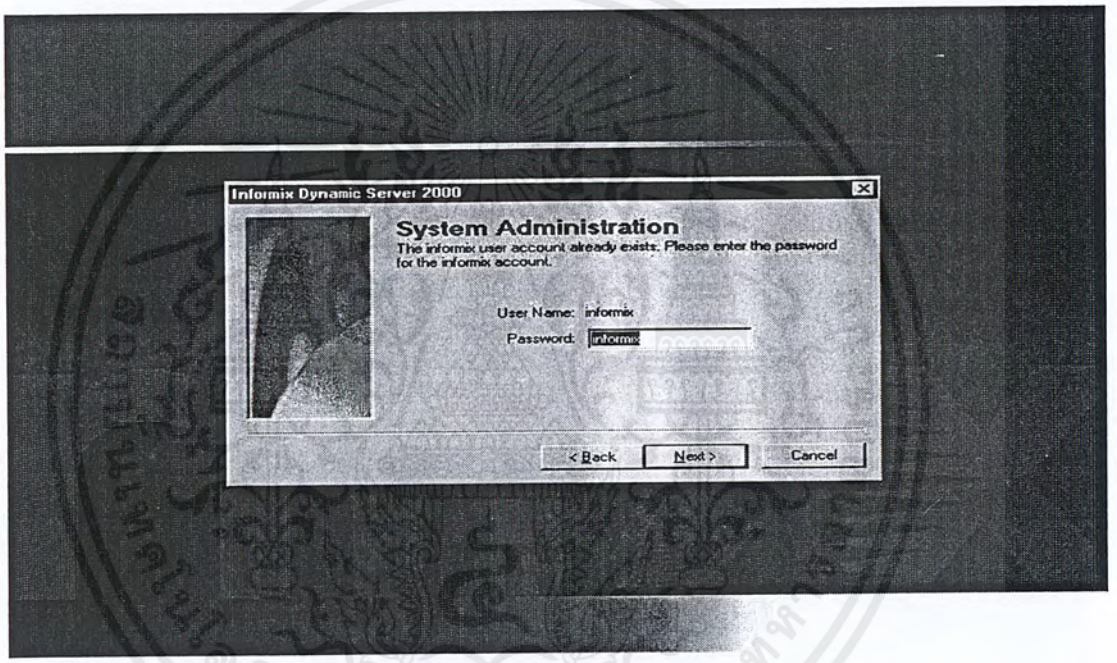
3.7.1 จัดประเภทงานของผู้ดูแลระบบ เช่น งานตรวจสอบระบบ, งานรักษาความปลอดภัยให้กับ ฐานข้อมูล

3.7.2 ระบุชื่อผู้ตรวจสอบระบบ

3.7.3 ระบุชื่อผู้คอยรักษาความปลอดภัยให้กับฐานข้อมูล

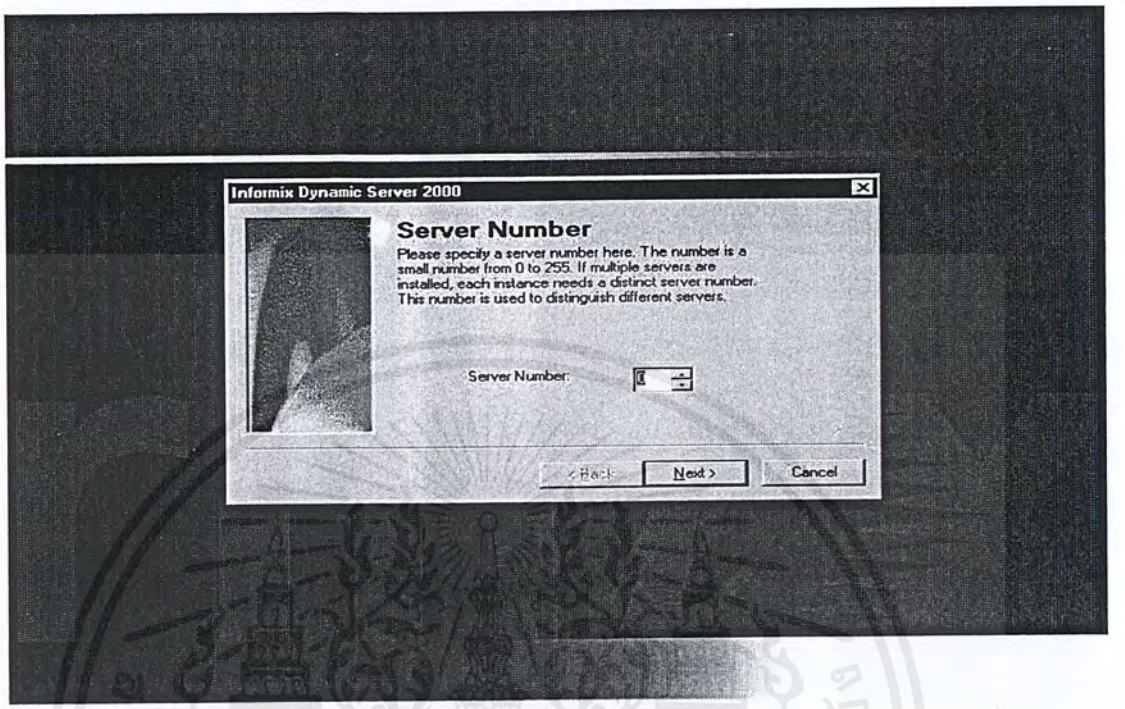
3.8 การตั้ง Password

ในการตั้ง Password โดยค่า default จะเป็น "informix" ขึ้นอยู่กับผู้ติดตั้งว่าจะตั้ง Password อะไร



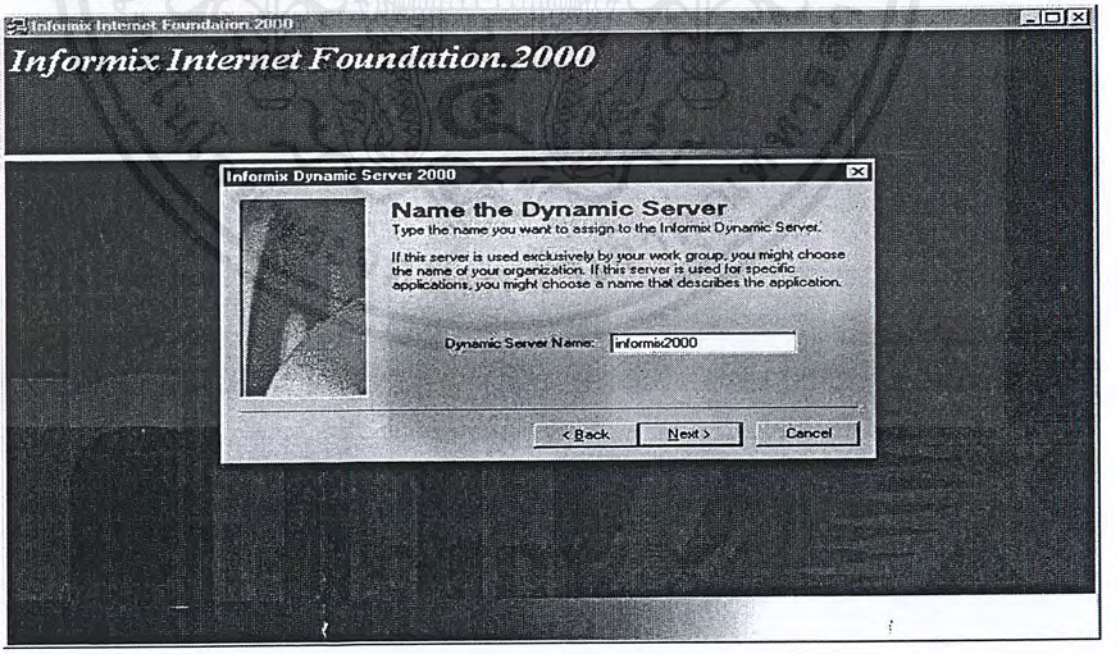
รูปที่ ก-7 แสดงการการตั้ง Password

3.9 การตั้ง หมายเลขของ Server โดยหมายเลขของ Server จะมีตั้งแต่ 0 ไปถึง 255



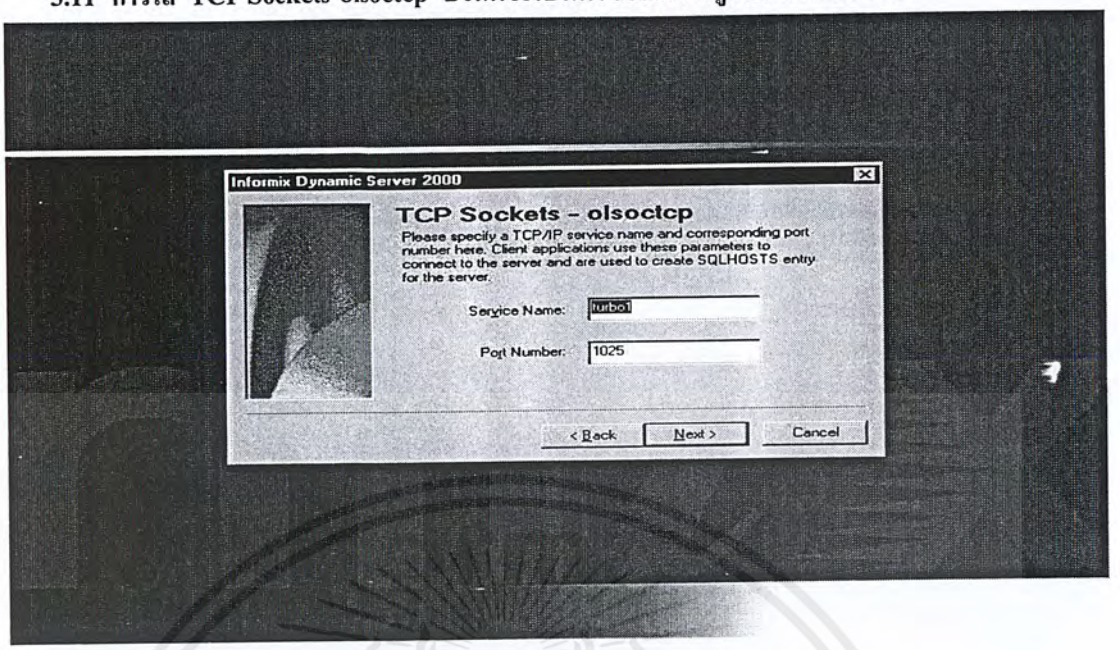
รูปที่ ก-8 แสดงการการตั้ง หมายเลขของ Server

3.10 การตั้งชื่อ Server เราสามารถกำหนดเองได้



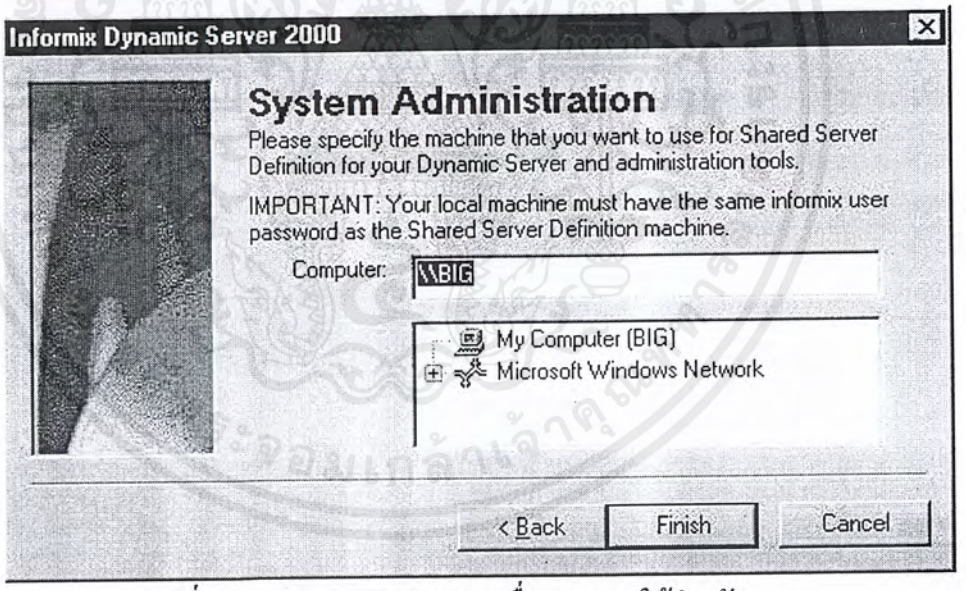
รูปที่ ก-9 แสดงการการตั้งชื่อของ Server

3.11 การใส่ TCP Sockets-olsoctcp ปรกติจะเป็นค่า default อยู่แล้วกด Next เลย



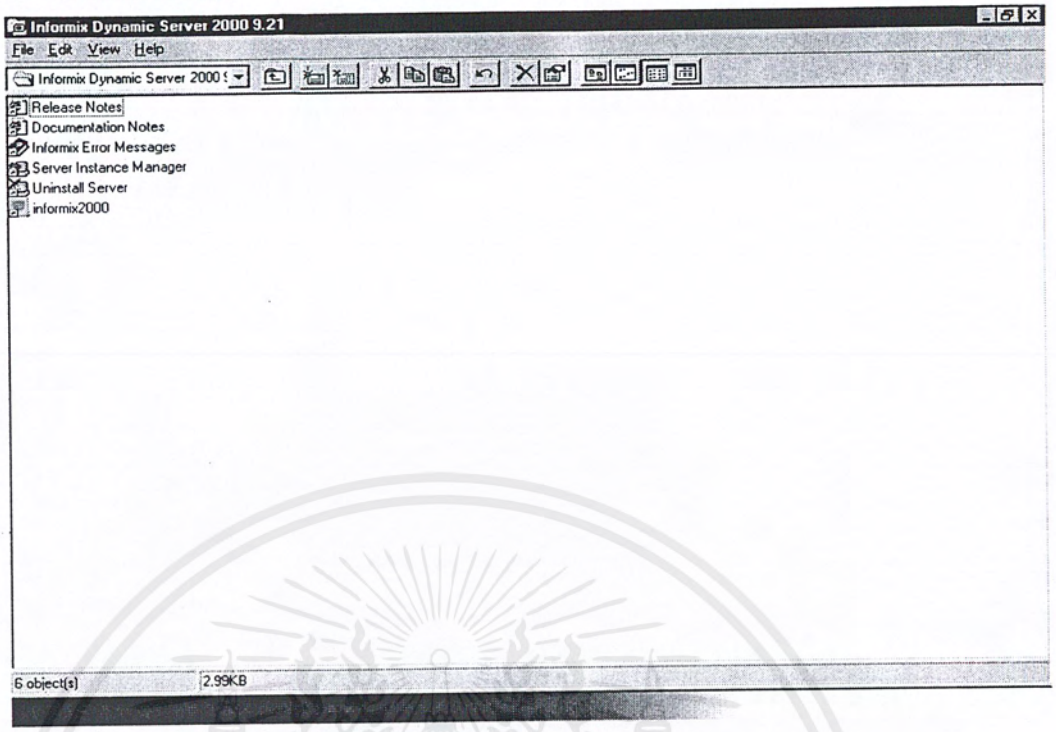
รูปที่ ก-10 แสดงการการใส่ TCP Sockets-olsoctcp

3.12 การกำหนดเครื่อง Server ให้สำหรับ user



รูปที่ ก-11 แสดงการการกำหนดเครื่อง Server ให้สำหรับ user

3.13 เสร็จสิ้นการติดตั้ง



รูปที่ ก-12 เสร็จสิ้นการติดตั้ง

ภาคผนวก ข

- การเซตให้ระบบฐานข้อมูลเริ่มทำงาน
- การ Instance Server
- การเซต Informix Setnet32
- การใช้งาน SQL editor

1. การเซตให้ระบบฐานข้อมูลเริ่มทำงาน

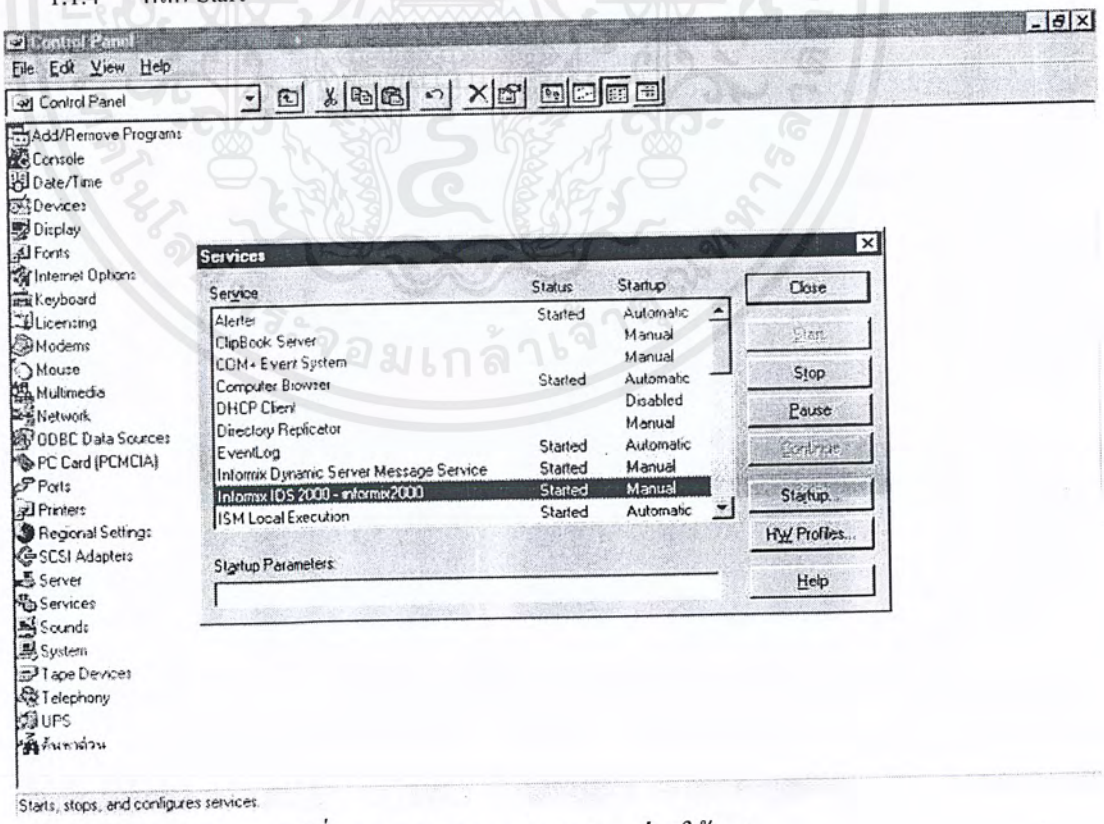
1.1 ขั้นตอนการเซตให้ database server ทำงาน

1.1.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

1.1.2 ดับเบิลคลิกที่ไอคอน Services

1.1.3 เลือก Informix IDS 2000-informix2000 จากกล่องรายการ service

1.1.4 คลิก Start



รูปที่ ข-1 แสดงการ start services ก่อนใช้งาน

1.2 หากเป็นการเซตให้ server ทำงานเป็นครั้งแรกต้องทำตามขั้นตอนดังนี้

1.2.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

1.2.2 ดับเบิลคลิกที่ไอคอน Services

1.2.3 เลือก Informix IDS 2000-informix2000 จากกล่องรายการ service

1.2.4 ใส่ข้อความ -iy ลงในกล่อง Startup Parameters

1.2.5 คลิก Start

1.3 ขั้นตอนการเซตให้ server ทำงานโดยอัตโนมัติ

1.3.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

1.3.2 ดับเบิลคลิกที่ไอคอน Services

1.3.3 เลือก Informix IDS 2000-informix2000 จากกล่องรายการ service

1.3.4 คลิก Start และ Automatic ใน service dialog box

1.3.5 จากนั้นคลิก OK

1.3.6 ตรวจสอบว่าไม่มีข้อความใดๆ ในกล่อง Startup Parameters

1.3.7 คลิก Start

1.4 ขั้นตอนการเซตให้ระบบฐานข้อมูลหยุดทำงาน

1.4.1 คลิกที่ไอคอน Control Panel ใน Start menu ของ Windows NT

1.4.2 ดับเบิลคลิกที่ไอคอน Services

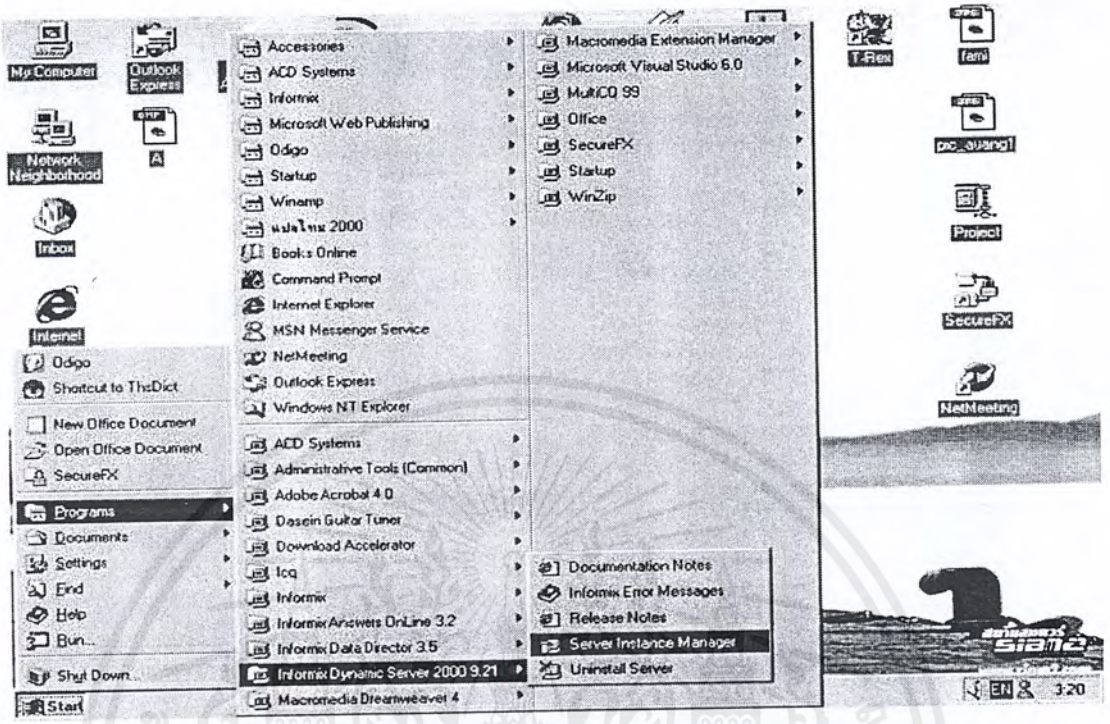
1.4.3 เลือก Informix IDS 2000-informix2000 จากกล่องรายการ service

1.4.4 คลิก Stop

2. การ Instance Server

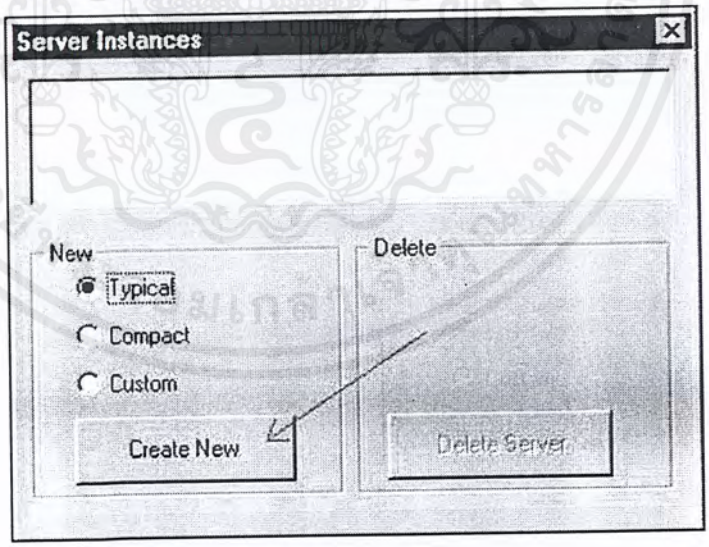
เป็นการสร้าง Server ขึ้นมาอีกตัวซึ่งเป็นความสามารถของ Informix Dynamic Server

2.1 คลิกที่ Start -> Programs -> Informix Dynamic Server 2000 9.21 -> Server Instance Manager



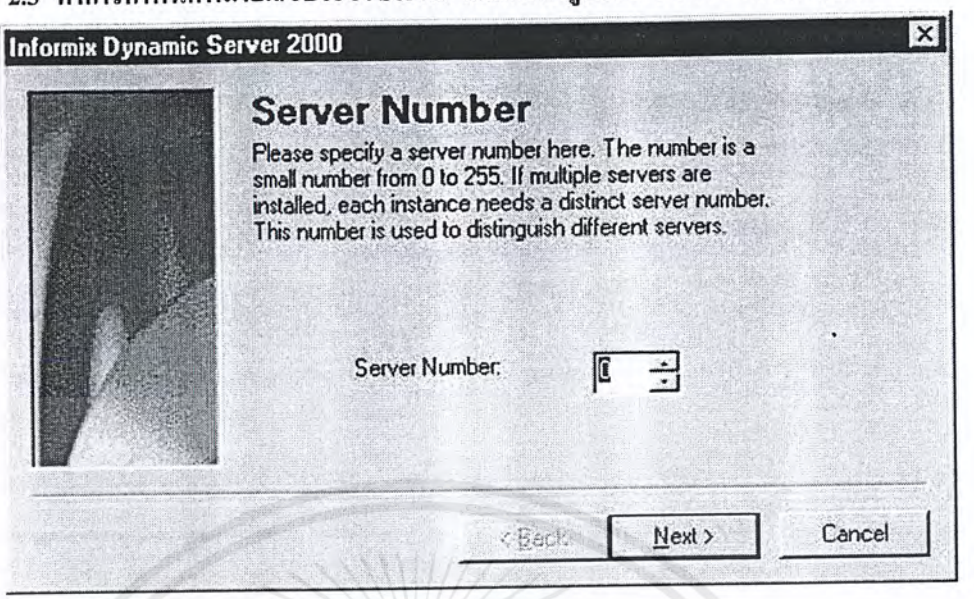
รูปที่ ข-2 แสดงการ start services ก่อนใช้งาน

2.2 หลังจากนั้นก็ เลือกไปที่ Typical แล้วก็คลิกที่ Create New เพื่อสร้าง Server



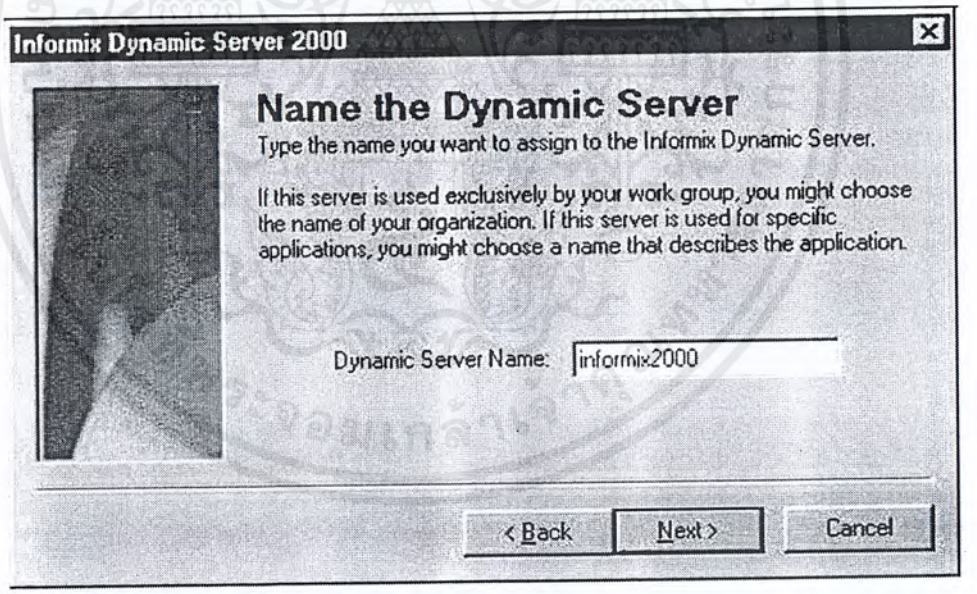
รูปที่ ข-3 แสดงการหน้าจอของ Server Instances

2.3 ทำการกำหนดหมายเลขประจำ Server ที่แสดงดังรูปที่ ข-4 กำหนดให้ค่าเท่ากับ 0



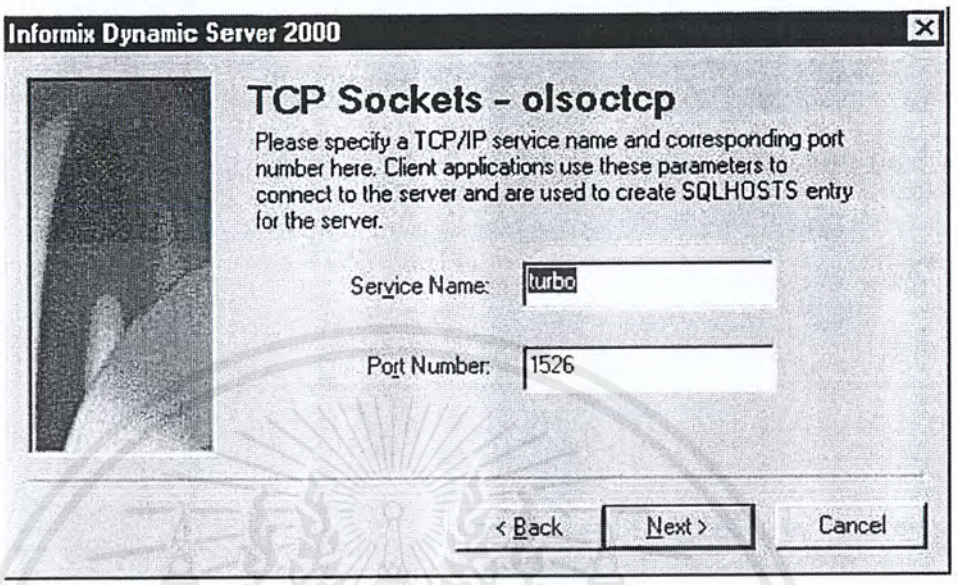
รูปที่ ข-4 แสดงการกำหนดหมายเลขประจำ Server

2.4 การตั้งชื่อ Dynamic Server โดยเราเป็นคนตั้งชื่อเอง



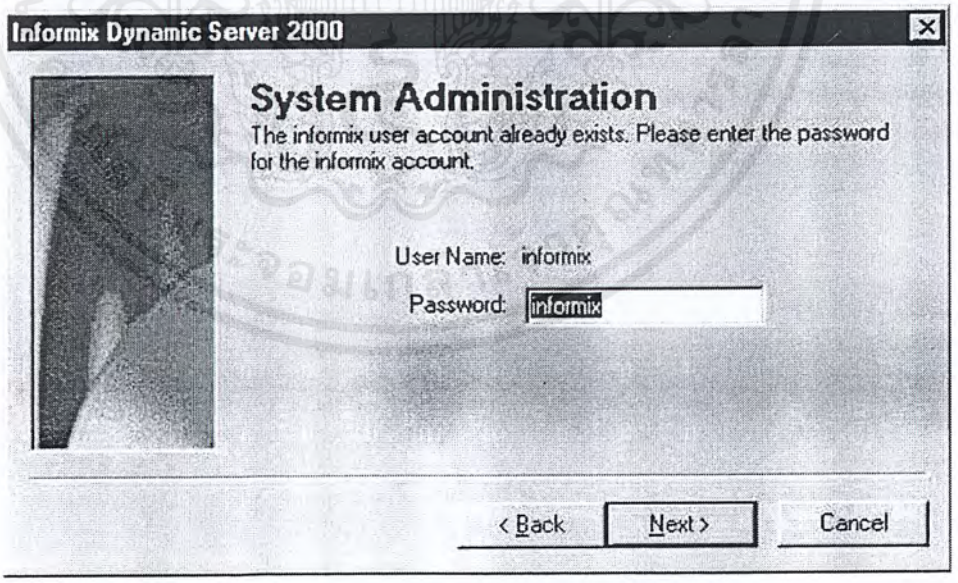
รูปที่ ข-5 แสดงการกำหนดตั้งชื่อ Dynamic Server

2.5 การเซต TCP Sockets – olsoctcp โดยปรกติจะขึ้น default ให้แล้ว คลิก Next ได้เลย



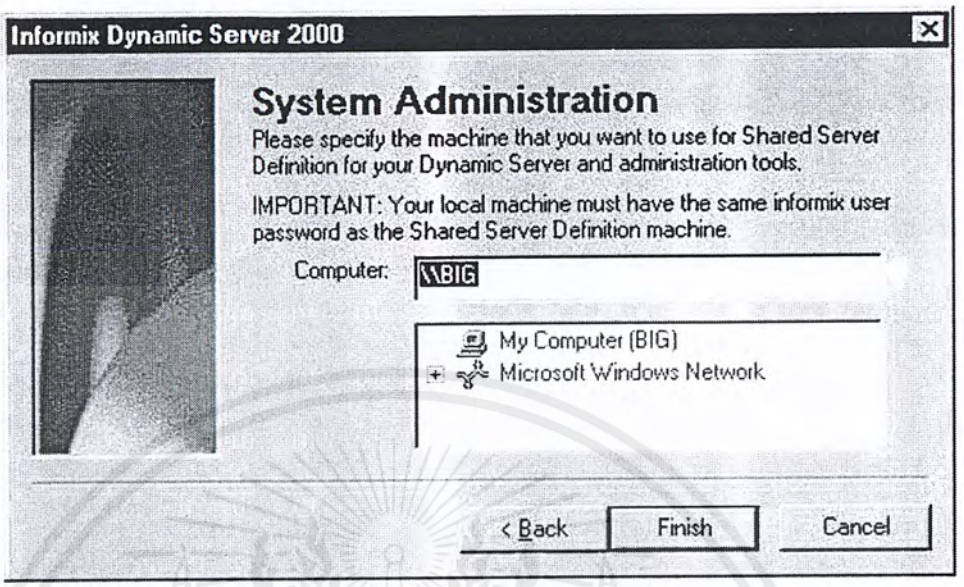
รูปที่ ข-6 แสดงการเซต TCP Sockets - olsoctcp

2.6 การตั้ง User Name และ Password โดย User Name จะเป็น "informix" จะเป็นค่า default ให้แล้ว เราจะต้องตั้ง Password เองจะแสดงดังรูป



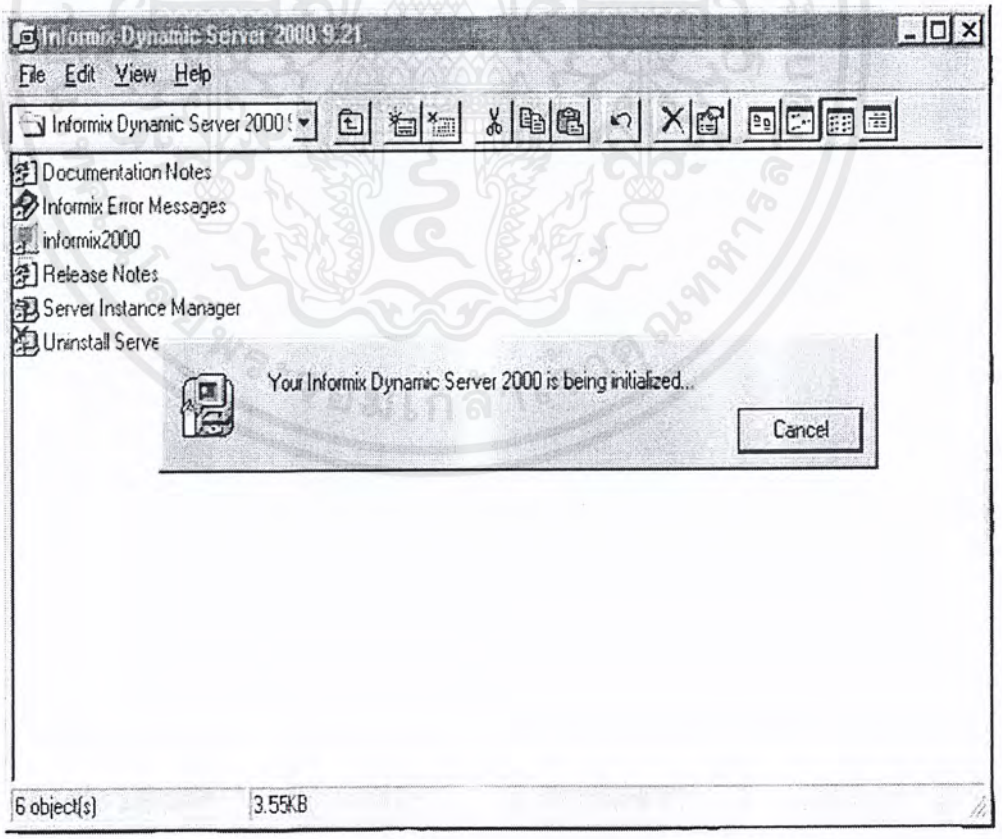
รูปที่ ข-7 แสดงการตั้ง User Name และ Password

2.7 แสดงการกำหนดเครื่องที่เป็น Server ดังรูป



รูปที่ ข-8 แสดงการกำหนดเครื่องที่เป็น Server

2.8 แสดงการขั้นตอนสุดท้ายของการ Instance คือการ initialize Server informix2000 ดังรูป

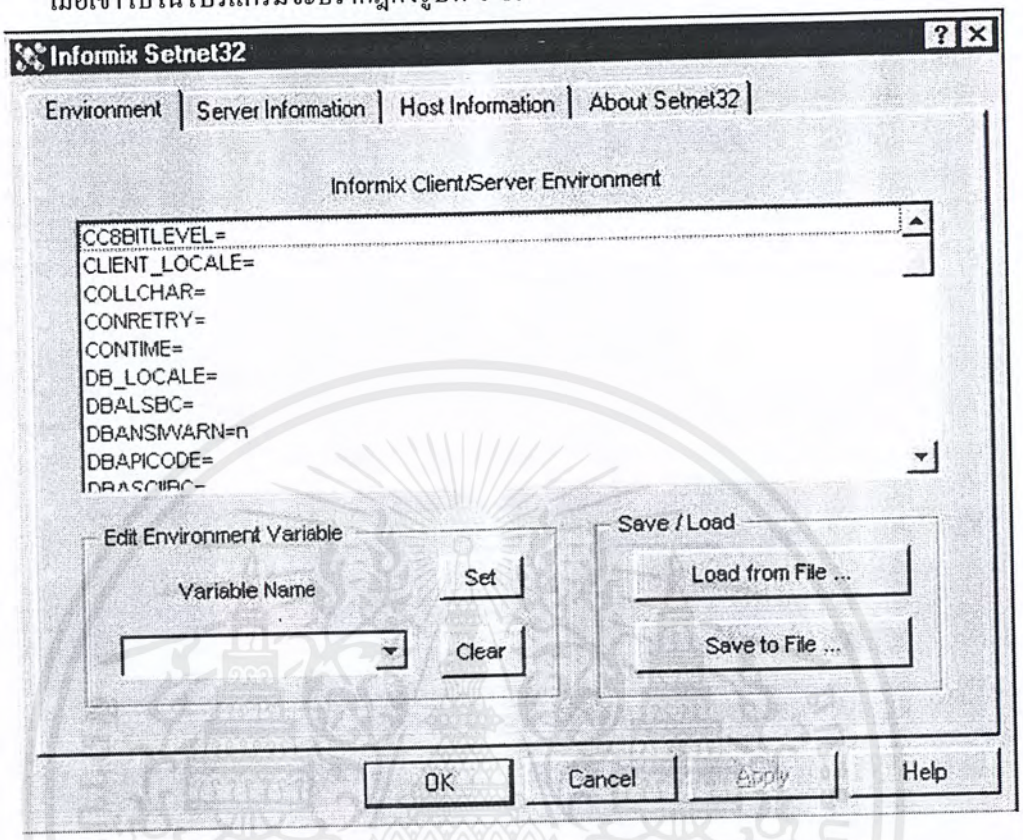


รูปที่ ข-9 แสดงการ initialize เป็นขั้นตอนสุดท้ายของการ Instance Server

3. การเซต Informix Setnet32

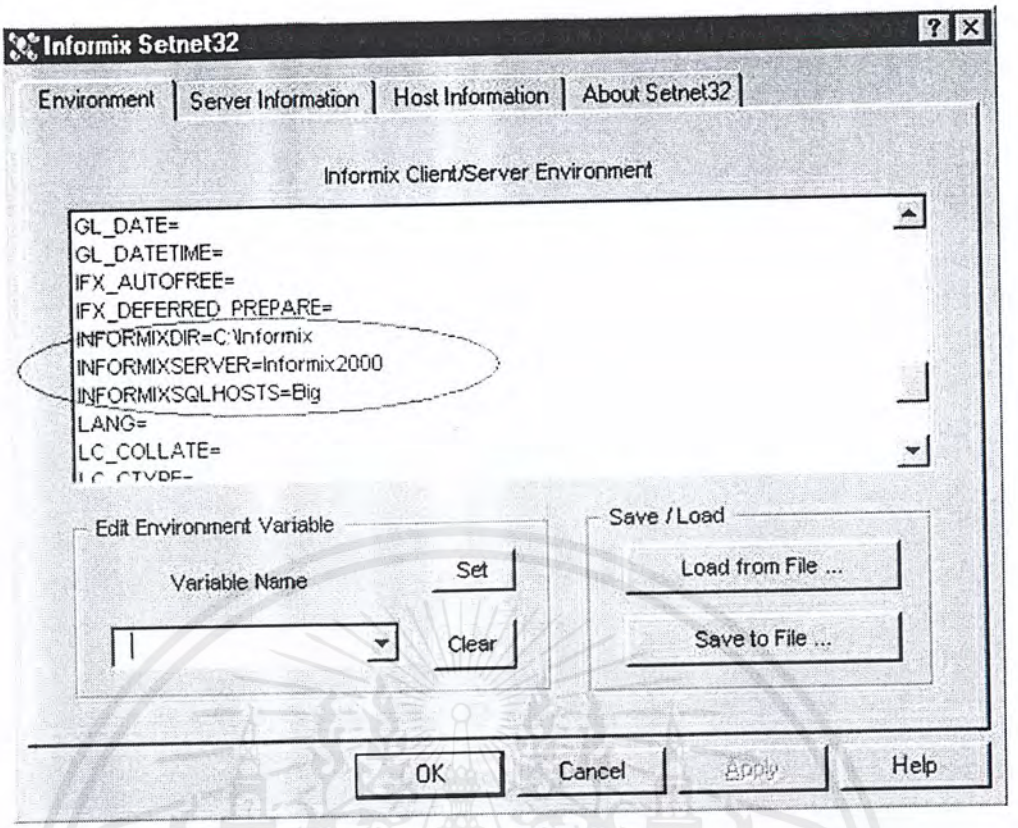
สามารถ เข้าไปที่ Start Menu -> Programs -> Informix -> Informix Setnet32

เมื่อเข้าไปในโปรแกรมจะปรากฏดังรูปที่ ข-10

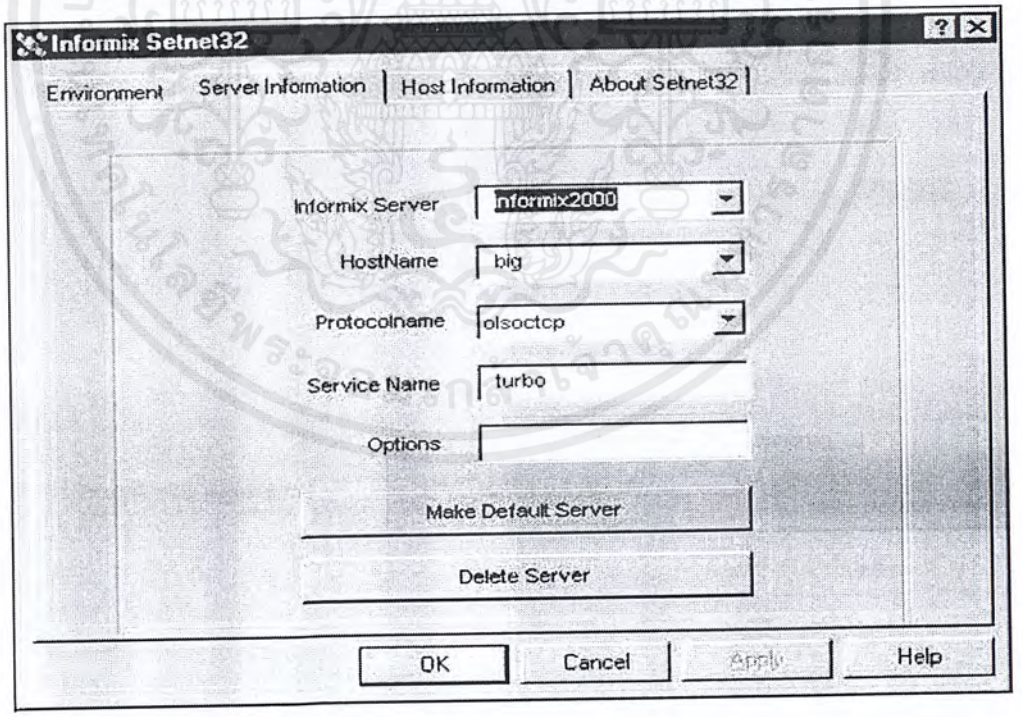


รูปที่ ข-10 แสดงหน้าจอของ Informix Setnet32

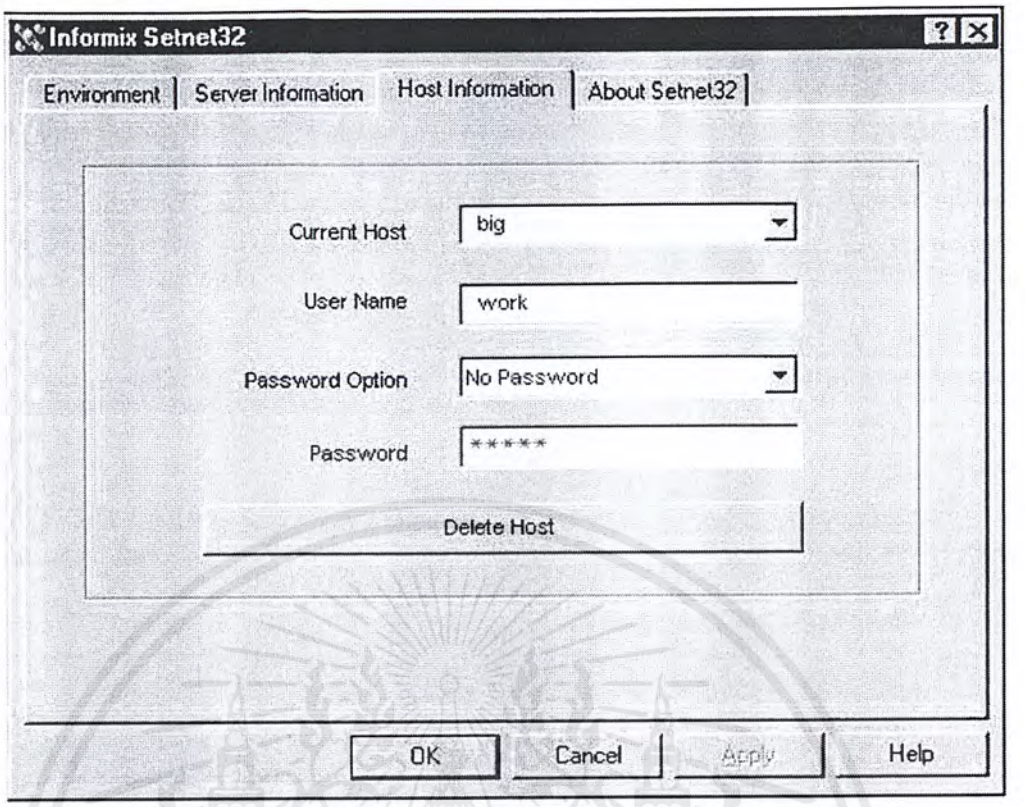
- ในส่วนของ Environment
- ที่ช่อง INFORMIXSERVER = ซึ่งให้ใส่ชื่อของ Database Server ลงไป ไม่ใช่ชื่อเครื่องเซิฟเวอร์
- ในส่วนของ Server Information
- ในช่อง Host name ให้ใส่ชื่อของเครื่องเซิฟเวอร์ลงไป
- ในช่อง Protocalname ให้เลือกค่า onsoctcp
- ในช่อง Service Name ให้ใส่ชื่อของ Service ที่ได้ตั้งไว้ เช่น turbo
- ในส่วนของ Host Information
- ในช่อง User Name ให้ใส่ชื่อผู้ที่จะใช้ฐานข้อมูลลงไป



รูปที่ ข-11 แสดงการ Set Informix Setnet32 ในส่วนของ Environment

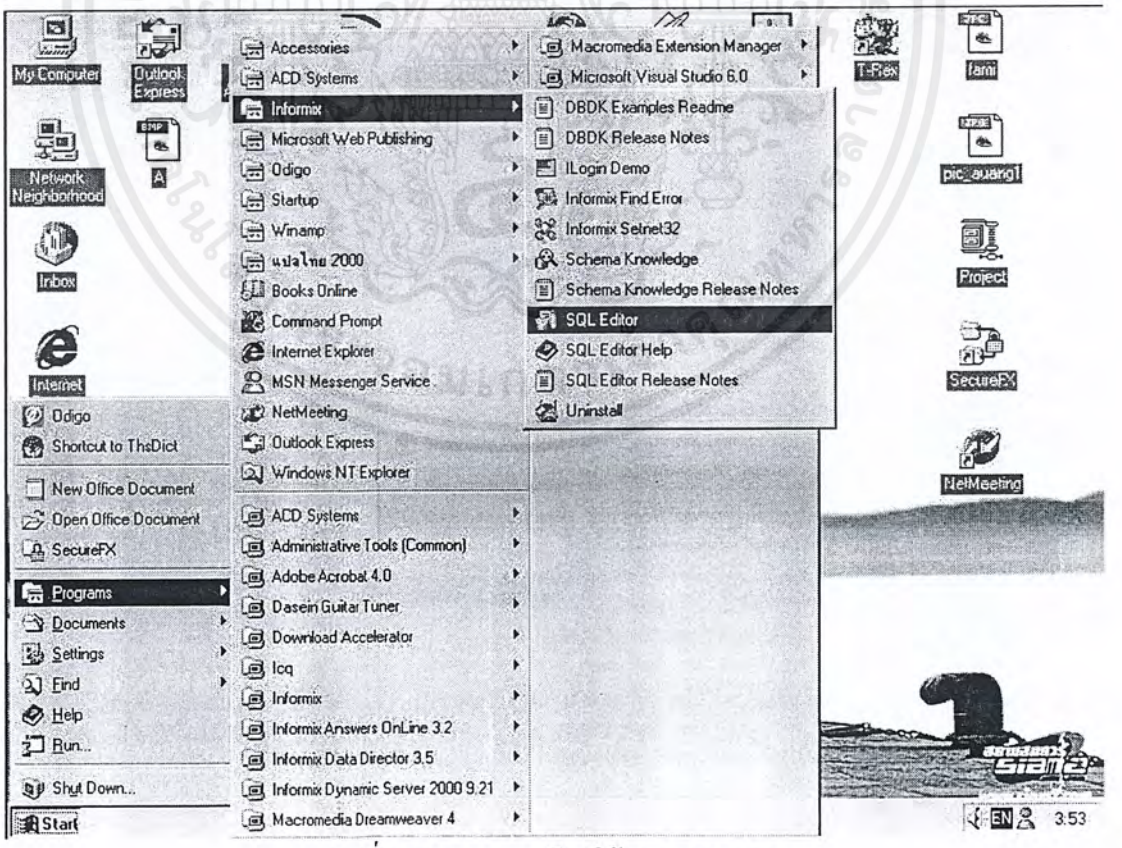


รูปที่ ข-12 แสดงการ Set Informix Setnet32 ในส่วนของ Server Information



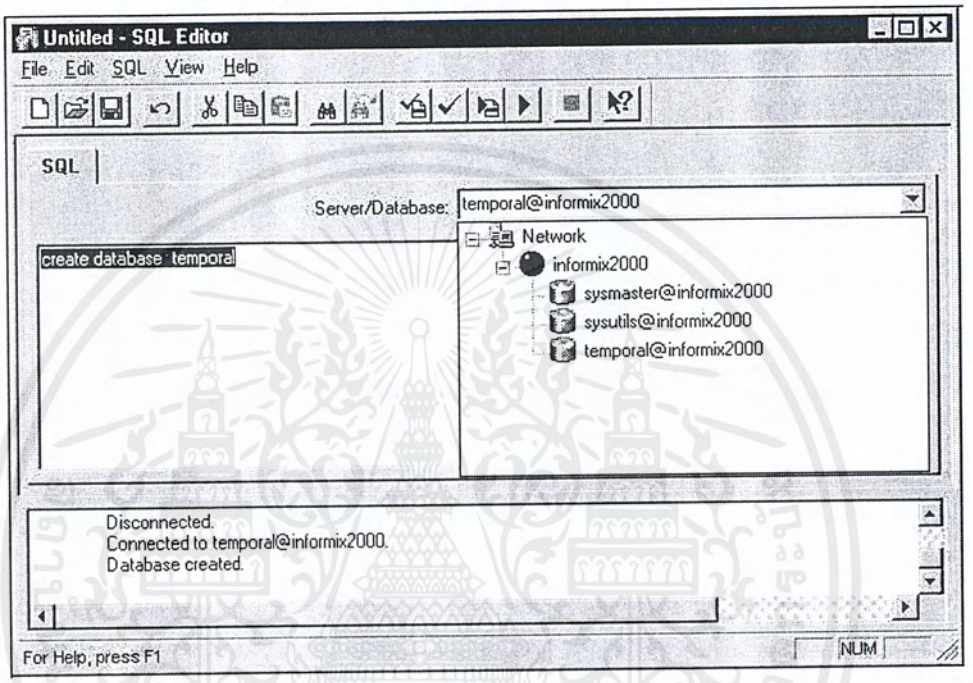
รูปที่ ข-13 แสดงการ Set Informix Setnet32 ในส่วนของ Host Information

4. การใช้งาน SQL Editor



รูปที่ ข-14 แสดงการเรียกใช้งาน SQL Editor

หลังจากเข้าโปรแกรม SQL Editor ก็จะปรากฏดังรูปที่ ข-15 ขั้นแรกต้องทำการติดต่อ Server และ Database ก่อน ถ้ายังไม่ได้สร้าง database ก็สามารถทำได้โดยใช้คำสั่ง SQL ดังรูปนั้นจะสร้าง Database name ชื่อ temporal



รูปที่ ข-15 แสดงการติดต่อกับ Server และ Database

ภาคผนวก ค

หลักการดาต้าไดเรคเตอร์ออบเจ็กต์ Data Director Objects

1. ดาต้าไดเรคเตอร์ออบเจ็กต์คืออะไร

ดาต้าไดเรคเตอร์ออบเจ็กต์เป็นชุดของการโปรแกรมอ๊อบเจกต์ต่าง ๆ ที่ซ่อนการทำงานต่าง ๆ ของฐานข้อมูลซึ่งจะมีความสัมพันธ์กับดาต้าไดเรคเตอร์จียูไอ (Data Director GUI)

2. ทำไมจึงใช้ดาต้าไดเรคเตอร์ออบเจ็กต์

2.1 ขยายการทำแอพพลิเคชันต้นแบบ

เราไม่ต้องสร้างออบเจ็กต์ทั้งหมดในการเขียนโปรแกรม ถ้าเราได้สร้างออบเจ็กต์เหล่านั้นผ่านทางยูสเซอร์อินเทอร์เฟส (User interface) แล้ว เราสามารถอ้างถึงออบเจ็กต์เหล่านั้นอย่างง่าย ๆ ได้ ในการพัฒนาแอพพลิเคชัน โดยส่วนมากจะเริ่มต้นด้วยการใช้ดาต้าไดเรคเตอร์จียูไอ พัฒนาแอพพลิเคชันต้นแบบและพัฒนาต้นแบบนั้นโดยการใช้ดีดีโอ(DDO)

ครั้งแรกที่แอพพลิเคชันทำงานได้และได้ผ่านการทดสอบแล้ว เราสามารถเพิ่มการทำงานที่ซับซ้อนได้โดยการใช้ดีดีโอ ฟอร์มการติดต่อกับดีดีโอ (DDO interface form) การทำงานพื้นฐานจะสร้างในจียูไอ (GUI) ดังนั้นการขยายโค้ดจะหมายถึงการเพิ่มประสิทธิภาพมากกว่าการเขียนใหม่ เราจะใช้จียูไอในการสร้างโค้ดออบเจ็กต์แบบที่มองเห็นได้ (Visual object) ต่าง ๆ ที่จำเป็นลงบนฟอร์ม และใช้ดีดีโอในการเขียนโค้ดในส่วนของการติดต่อกับฐานข้อมูล (Database) ทั้งหมด

2.2 ประสิทธิภาพ

ในบางกรณีประสิทธิภาพของแอพพลิเคชันสามารถปรับปรุงได้โดยการกำจัดการลิ้งค์แบบแดร็กและดร็อป (Drag-and-drop) ตัวอย่างเช่น สมมุติว่าแอพพลิเคชันของเรามีฟอร์มหนึ่งที่ผู้ใช้ต้องการใช้งานเพียงหนึ่งครั้งในหนึ่งเดือน ถ้าเราใช้วิธีแดร็กและดร็อปในการสร้างการติดต่อระหว่าง Visual object บนฟอร์มกับ คอลัมน์ ต่าง ๆ ในฐานข้อมูล การเชื่อมต่อครั้งแรกจะเกิดเมื่อเราเริ่มใช้แอพพลิเคชันซึ่งอาจจะใช้เวลาไม่มาก ในการให้ข้อมูลที่ใช้ไม่บ่อย ถ้าเราใช้ดีดีโอแทน โค้ดจะถูกประมวลผลเมื่อผู้ใช้ได้ใช้ฟอร์มนั้นซึ่งจะไม่ได้ประมวลผลทุกครั้งเมื่อเราเริ่มใช้ แอพพลิเคชัน

2.3 สามารถติดต่อกับซอฟต์แวร์ประเภทอื่น ๆ ได้

การเขียนโปรแกรมด้วยดีดีโอจะอยู่บนพื้นฐานของคอมโพเนนท์ออบเจ็กต์โมเดล (Component Object Model (COM)) ซึ่งทำให้โค้ดที่เป็นดีดีโอสามารถทำงานกับซอฟต์แวร์ที่สนับสนุนคอม (COM) ได้ ตัวอย่างเช่น เราสามารถเขียนโค้ดที่เป็นดีดีโอในโปรแกรมของเอกเซลล์ (Excel Application) ทำงานกับเอกสารที่ใช้ข้อมูลจากฐานข้อมูลอินฟอร์มิกซ์

2.4 ความชื่นชอบของผู้พัฒนาโปแกรม

เหมาะสมกับผู้พัฒนาโปรแกรมที่ชอบเขียนโค้ดเอง เพราะว่าการฟังก์ชันทำงานของ Data Director ทั้งหมดสามารถใช้งานโดยการเขียนโค้ดผ่านดีดีโอได้

2.5 การจัดการกับข้อมูลชนิดพิเศษสำหรับข้อมูลที่ไม่มีรูปแบบ

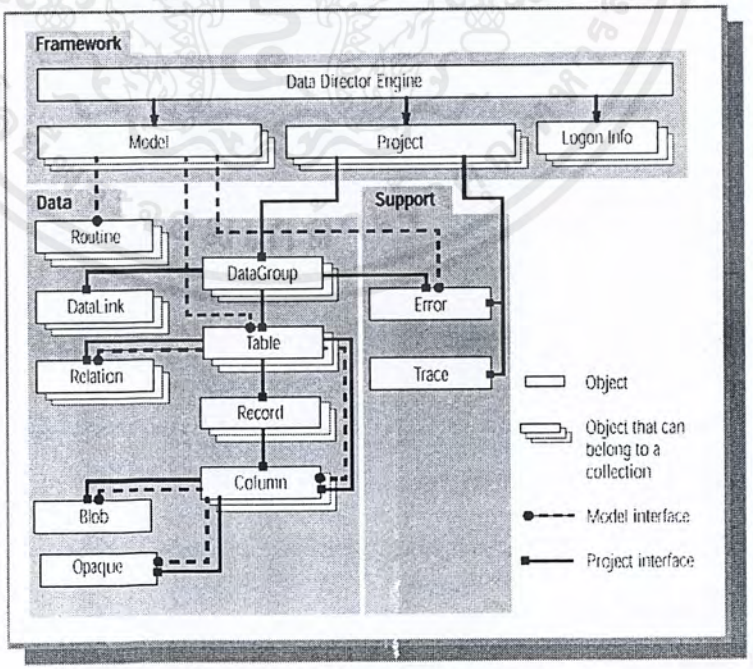
ถ้าข้อมูลที่เก็บในโอเปค (Opaque) นั้นไม่ใช่ข้อมูลที่เป็นตัวอักษร เราต้องใช้ดีดีโอในการแสดงค่าของข้อมูลชนิดโอเปคในแอพพลิเคชันของเรา ข้อมูลที่เก็บในโอเปคส่วนมากเป็นข้อมูลที่ไม่มีรูปแบบแน่นอน เช่น เสียง วิดีโอ และสี เราต้องใช้ดีดีโอในการทำงานที่แตกต่างกันกับข้อมูล เช่น การอ่านและการเขียนจากบัฟเฟอร์หรือไฟล์

2.6 มีความยืดหยุ่นเป็นอย่างมากในขณะทำงาน

การใช้ดีดีโออนุญาติให้แอพพลิเคชันเปลี่ยนโมเดลขณะรันโปรแกรมได้ แต่เมื่อเราใช้แอพพลิเคชันที่เป็นจียูไอขณะรันโปรแกรม ต้องใช้โมเดลที่สร้างในขณะออกแบบโปรแกรมเท่านั้น

3 โครงสร้างลำดับขั้นของดาต้าไดเรกเตอร์ออบเจ็กต์

ไลบรารี่(Library)ของดีดีโอประกอบไปด้วยออบเจ็กต์ต่างๆ ที่จะเข้าถึงออบเจ็กต์อื่น ๆ ในโครงสร้างลำดับขั้นทาง Logical ออบเจ็กต์ที่เป็นเจ้าของออบเจ็กต์อื่น ๆ เรียกออบเจ็กต์นั้นว่า **ออบเจ็กต์พ่อแม่** (Parent object) ส่วนออบเจ็กต์ที่ถูกควบคุมเรียกว่า **ออบเจ็กต์ลูก** (Child object) ออบเจ็กต์แต่ละออบเจ็กต์สามารถมีพ่อแม่ได้มากกว่า 1 ออบเจ็กต์ตัวอย่างเช่น ออบเจ็กต์โมเดล (Model object), ออบเจ็กต์โปรเจค (Project object) และออบเจ็กต์ดาต้ากรุ๊ป (DataGroup object) เป็นพ่อแม่ของออบเจ็กต์ความผิดพลาด (Error object)



รูปที่ ค-1 โครงสร้างลำดับขั้นของดีดีโอ

4 ออบเจ็กต์และคอลเลกชัน (Objects and Collections)

คอลเลกชันหนึ่ง ๆ ประกอบไปด้วยหลาย ๆ ออบเจ็กต์เช่น ตาราง, โปรเจค และโมเดล ตัวอย่างเช่น ออบเจ็กต์ดาต้ากรุ๊ป สามารถเป็นสมาชิกของดาต้ากรุ๊ปคอลเลกชัน (DataGroup collection) ซึ่งดาต้ากรุ๊ปคอลเลกชันนี้ ประกอบด้วยหลาย ๆ ออบเจ็กต์ดาต้ากรุ๊ป คอลเลกชันหนึ่ง ๆ จะมีกลุ่มของคุณสมบัติและเมทธอดเฉพาะและแตกต่างจากสมาชิกต่าง ๆ ใน คอลเลกชัน ดังนั้นคอลเลกชันต่าง ๆ จะไม่มีเมทธอดและคุณสมบัติที่เหมือนกัน

คุณสมบัติของคอลเลกชัน

- สามารถแก้ไขหรือดึงข้อมูลแต่ละออบเจ็กต์ในคอลเลกชันได้
- สามารถดึงแต่ละออบเจ็กต์ใน คอลเลกชันได้
- สามารถนับจำนวนของแต่ละออบเจ็กต์ในคอลเลกชันได้

ออบเจ็กต์ที่สามารถเป็น คอลเลกชัน ได้ ได้แก่ เอนจิ้น (Engine), โปรเจค, โมเดล, ดาต้ากรุ๊ป, ตาราง, ความสัมพันธ์, คำสั่ง (Routines) และเรคคอด คอลเลกชันต่าง ๆ เป็นเจ้าของแต่ละออบเจ็กต์ที่รวมกันเป็นคอลเลกชัน ตัวอย่างเช่น ตารางคอลเลกชันเป็นเจ้าของออบเจ็กต์ตาราง และเรคคอดคอลเลกชันเป็นเจ้าของออบเจ็กต์เรคคอด

5 ชนิดของออบเจ็กต์

ดาต้าไดเรคเตอร์ออบเจ็กต์ แบ่งเป็น 3 ประเภทดังนี้

- ออบเจ็กต์เฟรมเวิร์ค (Framework objects)
- ออบเจ็กต์ข้อมูล (Data objects)
- ออบเจ็กต์สนันสนุน (Support objects)

5.1 ออบเจ็กต์เฟรมเวิร์ก (Framework Objects)

แต่ละแอพพลิเคชันที่เราสร้างด้วยดาต้าไดเรคเตอร์และดีดีโอจะมีการจัดการและโครงสร้างแบบนามธรรม (Logical) ฐานข้อมูลที่เราทำงานด้วยนั้นก็มีโครงสร้างแบบนามธรรมของมันเอง ออบเจ็กต์เฟรมเวิร์คจะแสดงลักษณะโครงสร้างเหล่านี้และมีการจัดการเข้าถึงออบเจ็กต์อื่น ๆ ที่สร้างขึ้นโดยแอพพลิเคชัน และจัดเตรียมข้อมูลที่แอพพลิเคชันของเราจะเข้าถึง

5.1.1 เอนจิ้น (Engine)

เอนจิ้น เป็นออบเจ็กต์ต้น (Root Object) เราจะต้องสร้างออบเจ็กต์นี้ก่อนที่จะสร้างดาต้าไดเรคเตอร์ออบเจ็กต์อื่น ๆ โดยออบเจ็กต์เอนจิ้น (Engine object) นี้เราสามารถควบคุมโปรเจค และออบเจ็กต์โมเดลได้ เราสามารถสร้าง โปรเจค และโมเดลได้หลายอันในออบเจ็กต์เอนจิ้นเพียงอันเดียว เนื่องจากในการทำงานของวิชวลเบสิกหรือดาต้าไดเรคเตอร์ จะอนุญาตให้มีเอนจิ้นที่รัน ได้เพียงเอนจิ้นเดียว

สิ่งที่เราสามารถกระทำกับ เอนจิ้นได้ มีดังนี้

- สร้างออบเจ็กต์โปรเจค
- สร้างออบเจ็กต์โมเดล

- สร้างออบเจ็กต์ล็อคอินโฟ

ออบเจ็กต์ อื่น ๆ ไม่สามารถเป็นเจ้าของ ออบเจ็กต์เอนจิ้น ได้ ออบเจ็กต์เอนจิ้น เป็นเจ้าของโปรเจค , โมเดลและออบเจ็กต์ล็อคอินโฟ

5.1.1.1 โปรเจค (Project)

ออบเจ็กต์โปรเจค จะจัดการกลุ่มของดาต้ากรุ๊ป ตาราง คอลัมน์ และออบเจ็คความสัมพันธ์ (Relation object) เหมือนออบเจ็กต์อื่น ๆ ที่ใช้ในแอพพลิเคชันของเรา ออบเจ็กต์โปรเจคจะอยู่ในรูปของไฟล์โปรเจคของดาต้าไดเรคเตอร์ (.ddx)

สิ่งที่เราสามารถกระทำกับออบเจ็กต์โปรเจค ได้มีดังนี้

- สร้างออบเจ็กต์ดาต้ากรุ๊ป
- เซตให้มีการทำงาน หรือไม่มีการทำงานของการตรวจจับความผิดพลาด (Handing Error) ขณะรันโปรแกรม

ออบเจ็กต์เอนจิ้น เป็นเจ้าของออบเจ็กต์โปรเจค ออบเจ็กต์โปรเจค เป็นเจ้าของออบเจ็กต์ดาต้ากรุ๊ป ออบเจ็กต์ความผิดพลาด และ ออบเจ็กต์เทรส (Trace object)

5.1.1.2 โมเดล (Model)

ออบเจ็กต์โมเดล จะทำหน้าที่เข้าถึงและปรับปรุง (Update) ข้อมูลของโมเดลของดาต้าไดเรคเตอร์ (Data Director Model) ซึ่งอยู่ในรูปของไฟล์โมเดล (.mlt หรือ .mlx)

สิ่งที่เราสามารถกระทำกับออบเจ็กต์โมเดล ได้มีดังนี้

- อิมพอร์ตโครงสร้างของฐานข้อมูลไปยังโมเดล
- คัดลอกและจัดเก็บโมเดล
- สร้างออบเจ็กต์ตาราง
- สร้าง LogonInfo object

ออบเจ็กต์เอนจิ้น เป็นเจ้าของออบเจ็กต์โมเดล ออบเจ็กต์โมเดล เป็นเจ้าของ ออบเจ็กต์ตาราง, ออบเจ็กต์รูทีน และ ออบเจ็กต์ความผิดพลาด

5.1.1.3 ล็อคออนอินโฟ (LogonInfo)

ล็อคออนอินโฟ จะทำการระบุข้อมูลเกี่ยวกับ ชนิดของฐานข้อมูล, ชื่อผู้ใช้, รหัสผ่าน, หมายเลขเซิร์ฟเวอร์ และ ชื่อฐานข้อมูลเมื่อเรากำลังล็อคอินไปยังฐานข้อมูล

สิ่งที่เราสามารถกระทำกับออบเจ็กต์ล็อคออนอินโฟ ได้มีดังนี้

- ล็อคออน (Log on) ไปยังฐานข้อมูล
- อิมพอร์ตฐานข้อมูลไปยังโมเดล

ออบเจ็กต์เอนจิ้น เป็นเจ้าของออบเจ็กต์ล็อคออนอินโฟ ออบเจ็กต์ล็อคออนอินโฟ ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

5.2 ออบเจ็กต์ข้อมูล (Data Objects)

แต่ละฐานข้อมูลที่เราทำงานด้วยจะประกอบด้วยส่วนต่าง ๆ ที่แน่นอน เช่น ตาราง, คอลัมน์ หรือ แถว ออบเจ็กต์ข้อมูล ซึ่งทำให้เราสามารถทำงานกับข้อมูลต่าง ๆ ของฐานข้อมูลเหล่านี้ได้

5.2.1 ดาต้ากรุ๊ป (DataGroup)

ออบเจ็กต์ดาต้ากรุ๊ป มีความสอดคล้องกับดาต้าไดเรคเตอร์ ดาต้ากรุ๊ป เป็นกลุ่มของความสัมพันธ์ของ ตาราง ในฐานข้อมูลและดาต้าลิงค์

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์ดาต้ากรุ๊ป ได้มีดังนี้

- สร้างออบเจ็กต์ล็อคออนอินโฟ
- สร้างออบเจ็กต์ดาต้าลิงค์
- สร้างออบเจ็กต์รูทีน (Routine object)
- ล็อคอินและล็อคออฟจากฐานข้อมูล
- ตั้งค่าออบชั่นเกี่ยวกับการรายงานความผิดผลาด
- ประมวลผลคำสั่งเอสคิวแอล (SQL)
- ตั้งค่าออบชั่นของระดับการล็อคเรคคอด (Record_Level Locking)

ออบเจ็กต์โปรเจค เป็นเจ้าของออบเจ็กต์ดาต้ากรุ๊ป ออบเจ็กต์ดาต้ากรุ๊ป เป็นเจ้าของออบเจ็กต์ดาต้าลิงค์ ออบเจ็กต์ตาราง และ ออบเจ็กต์ความผิดพลาด

5.2.2 ดาต้าลิงค์(DataLink)

ออบเจ็กต์ดาต้าลิงค์ มีความสอดคล้องกับดาต้าลิงค์ของดาต้าไดเรคเตอร์ ดาต้าลิงค์ เป็นการติดต่อระหว่างคอลัมน์ของฐานข้อมูลกับคอนโทรลของวิชวลเบสิก ดาต้าลิงค์ นี้จะเป็นตัวกำหนดว่าข้อมูลใดจะปรากฏอยู่ในคอนโทรลบน ฟอร์มของแอพพลิเคชัน

สิ่งที่เราสามารถกระทำกับออบเจ็กต์ดาต้าลิงค์ ได้มีดังนี้

- คัดลอกดาต้าลิงค์
- ตั้งค่าคุณสมบัติของ ดาต้าลิงค์
- ระบุว่าดาต้าลิงค์ เป็นดาต้าลิงค์จริง หรือเป็นดาต้าลิงค์เสมือน (Virtual DataLink)

ออบเจ็กต์ดาต้ากรุ๊ป เป็นเจ้าของ ออบเจ็กต์ดาต้าลิงค์ ออบเจ็กต์ดาต้าลิงค์ ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

5.2.3 ตาราง

ออบเจ็กต์ตาราง มีความสอดคล้องกับตารางในฐานข้อมูล ตารางในฐานข้อมูลประกอบไปด้วย คอลัมน์ และแถว

สิ่งที่เราสามารถกระทำกับออบเจ็กต์ตาราง ได้มีดังนี้

- สร้างออบเจ็กต์ความสัมพันธ์
- สร้างออบเจ็กต์คอลัมน์

- สร้างออบเจ็กต์เรคคอด
- สามารถเดินทางโดยผ่านเรคคอดในเซตของผลลัพธ์ได้
- สามารถเดินทางโดยเรคคอดที่ได้ทำการบุ๊คมาร์ค ได้
- กำหนดและประมวลผลฟังก์ชันของสตรอโปรซีเยอร์ (Stored procedure Function) และ รูทีนที่ผู้ใช้สร้างขึ้นใหม่ (User-define routines) ได้
- ตั้งค่าตัวแปรในคำสั่งเอสคิวแอล

ดาต้ากรุ๊ป และ ออบเจ็กต์โมเดล เป็นเจ้าของ ออบเจ็กต์ตาราง ออบเจ็กต์ตาราง เป็นเจ้าของ คอลัมน์ และ Relation ได้หลาย ๆ อัน ถ้า ตาราง เป็นส่วนหนึ่งของ ดาต้ากรุ๊ป มันจะเป็นเจ้าของ เรคคอดคอลเลคชัน ได้หลาย ๆ อันด้วย

5.2.4 คอลัมน์

ออบเจ็กต์คอลัมน์ มีความสอดคล้องกับ คอลัมน์ ในฐานข้อมูล คอลัมน์ ในฐานข้อมูลเป็นหน่วยในการจัดการขั้นพื้นฐานของฐานข้อมูล คอลัมน์ จะประกอบด้วยข้อมูลปกติทั่ว ๆ ไป เช่น นามสกุล ชื่อที่อยู่ และ หมายเลขโทรศัพท์

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์คอลัมน์ ได้มีดังนี้

- เข้าถึงออบเจ็กต์ที่เป็นข้อมูลแบบไบนารี่ขนาดใหญ่ (Binary large object (BLOB)) จากฐานข้อมูลได้
- เข้าถึงชนิดข้อมูลที่ผู้ใช้สร้างใหม่ (โอเปค) จากฐานข้อมูลได้
- กำหนดการดึงและแก้ไขข้อมูลและแคชชิ่ง (Caching) สำหรับข้อมูลแบบบลอบ (BLOB) ได้
- กำหนดชนิดข้อมูลในคอลัมน์ , ขนาด และ เมทธอดการเรียงลำดับ (Sorting method) ได้
- กำหนดคีย์หลัก (Primary key) ให้ คอลัมน์ ได้

ตาราง และ ออบเจ็กต์เรคคอด เป็นเจ้าของ ออบเจ็กต์คอลัมน์ ออบเจ็กต์คอลัมน์ เป็นเจ้าของ ออบเจ็กต์บลอบ และ ออบเจ็กต์โอเปค

5.2.5 เรคคอด (Record)

ออบเจ็กต์เรคคอด มีความสอดคล้องกับเรคคอด ในฐานข้อมูล เรคคอด ในฐานข้อมูลจะบรรจุข้อมูลเฉพาะซึ่งถูกกำหนดโดย คอลัมน์ ให้แยกกันเป็นส่วน ๆ

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์เรคคอด ได้มีดังนี้

- คัดลอกเรคคอด
- ล้างค่าแฟรก (Flag) ต่าง ๆ จากเรคคอด
- รีเซตค่าของ เรคคอด ให้กลับไปอย่างเดิม
- การล็อคเรคคอด ขณะที่กำลังถูกเปลี่ยนแปลง

ออบเจ็กต์ตาราง เป็นเจ้าของ ออบเจ็กต์เรคคอด

ออบเจ็กต์เรคคอด เป็นเจ้าของ ออบเจ็กต์คอลัมน์

5.2.6 ความสัมพันธ์ (Relation)

ออบเจ็กต์ความสัมพันธ์ ใช้อธิบายความสัมพันธ์ระหว่างตาราง 2 ตาราง

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์ความสัมพันธ์ ได้มีดังนี้

- ชี้ไปที่จุดเริ่มต้นและจุดหมายใน ตาราง ที่เราต้องการสร้างความสัมพันธ์ (Relationship)
- ชี้ไปที่จุดเริ่มต้นและจุดหมายใน คอลัมน์ ที่เราต้องการสร้างความสัมพันธ์
- เพิ่มจุดเริ่มต้นและจุดหมายของ คอลัมน์ ให้เป็นความสัมพันธ์
- ชี้ชนิดของความสัมพันธ์

ออบเจ็กต์ตาราง เป็นเจ้าของ ออบเจ็กต์ความสัมพันธ์ ออบเจ็กต์ความสัมพันธ์ ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

5.2.7 บลอบ (Blob)

ออบเจ็กต์บลอบ ใช้แสดงออบเจ็กต์ที่เป็นข้อมูลแบบไบนารี่ขนาดใหญ่ โดยปกติแล้วค่าที่เก็บในบลอบ จะใช้เก็บข้อมูลที่มีขนาดใหญ่ เช่น รูปภาพ ซึ่งเราต้องการที่จะเข้าถึงข้อมูลนั้นเป็นหน่วย ๆ เดียว บลอบ มีความสัมพันธ์กับคอลัมน์และเรคคอด ขณะที่เราเคลื่อนย้ายระหว่างเรคคอดกับเรคคอด ออบเจ็กต์บลอบ จะชี้ไปยังค่าบลอบสำหรับ เรคคอด นั้น ๆ

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์บลอบ ได้มีดังนี้

- ดึงข้อมูลบลอบได้
- กำหนดค่าข้อมูลไปยังออบเจ็กต์บลอบ ได้
- บันทึกและโหลดออบเจ็กต์บลอบ ไปมาระหว่างไฟล์
- นำข้อมูลแบบบลอบออกมา

ออบเจ็กต์คอลัมน์ เป็นเจ้าของออบเจ็กต์บลอบ ออบเจ็กต์บลอบ ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

5.2.8 โอเปค (Opaque)

ออบเจ็กต์โอเปค แสดงชนิดข้อมูลที่ผู้ใช้สร้างขึ้นมาในอินฟอร์มิกซ์ที่เป็นออปชันของข้อมูลของอินฟอร์มิกซ์ ชนิดข้อมูลที่ผู้ใช้สร้างขึ้นมา เป็นชนิดข้อมูลที่สร้างโดยผู้เขียนโปรแกรมเพื่อใช้แสดงข้อมูลชนิดพิเศษ เช่น Spatial หรือ ข้อมูลเกี่ยวกับเวลา (Time-series)

ออบเจ็กต์โอเปค มีความสัมพันธ์กับคอลัมน์ และเรคคอด ถ้าเราเคลื่อนย้ายข้อมูลระหว่างเรคคอด กับเรคคอด ออบเจ็กต์โอเปค จะมีความสัมพันธ์กับค่าข้อมูลสำหรับแต่ละเรคคอดที่เกิดใหม่

สิ่งที่เราสามารถกระทำกับออบเจ็กต์โอเปค ได้มีดังนี้

- แก้ไขข้อมูลที่ผู้ใช้กำหนดขึ้นมา
- มีการบันทึกและดึงข้อมูลออบเจ็กต์โอเปคไปมาระหว่างบัฟเฟอร์ของแอพพลิเคชัน
- มีการบันทึกและดึงข้อมูลออบเจ็กต์โอเปคไปมาระหว่างไฟล์ต่าง ๆ
- ทดสอบว่าออบเจ็กต์โอเปคมีความยาวจำกัดหรือไม่

- ทดสอบว่าออบเจ็กต์โอเปคไม่มีค่าหรือไม่
- เปรียบเทียบออบเจ็กต์ 2 ออบเจ็กต์ที่เห็น ถ้าเรามีข้อมูลแบบโอเปคที่เหมือนกัน

ออบเจ็กต์คอลัมน์ เป็นเจ้าของ ออบเจ็กต์โอเปค ออบเจ็กต์โอเปค ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

สิ่งสำคัญ : ถ้า ออบเจ็กต์โอเปค ถูกสร้างจากผลของการเรียกใช้รูทีนของเซิร์ฟเวอร์ผ่านทางออบเจ็กต์รูทีน ทำให้ ออบเจ็กต์ดาต้ากรุ๊ป เป็นเจ้าของ ออบเจ็กต์โอเปค แทนที่ ออบเจ็กต์คอลัมน์

5.2.9 รูทีน (Routine)

ออบเจ็กต์รูทีน ใช้แสดงฟังก์ชันทุก ๆ ชนิดของผู้ใช้หรือฟังก์ชันที่ระบบกำหนดหรือสตรอโปรซีเยอร์ในฐานข้อมูล การใช้ ออบเจ็กต์รูทีน เราสามารถเพิ่มสตรอโปรซีเยอร์และรูทีนที่ผู้ใช้สร้างขึ้น จากฐานข้อมูลไปยังออบเจ็กต์โมเดล ของเราได้ ออบเจ็กต์โมเดล เป็นเจ้าของ ออบเจ็กต์รูทีน

สิ่งสำคัญ : เราจะต้องได้รับออบเจ็กต์รูทีน จากออบเจ็กต์ดาต้ากรุ๊ป ไม่ใช่ได้มาจากออบเจ็กต์โมเดล ในการเรียกใช้รูทีน ออบเจ็กต์รูทีน ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

5.3 ออบเจ็กต์ช่วยเหลือ (Support Object)

แต่ละโปรเจคที่เราทำงานด้วยโดยใช้ ดาต้าไดเรคเตอร์ ต้องการการแสดงผลฟังก์ชันเฮ้าคีบปิ้ง (Housekeeping) ที่แน่นอน เช่น การเก็บเส้นทางของการเกิดความผิดพลาดและการเก็บที่อยู่ของไฟล์ของออบเจ็กต์ช่วยเหลือ ช่วยในการทำงานฟังก์ชันเหล่านี้

5.3.1 ความผิดพลาด

- ออบเจ็กต์ความผิดพลาด ใช้บอก ชนิด, เวลา และข้อความของความผิดพลาด
- โปรเจค, โมเดลและออบเจ็กต์ดาต้ากรุ๊ปเป็นเจ้าของ ออบเจ็กต์ความผิดพลาด
- ออบเจ็กต์ความผิดพลาด ไม่สามารถเป็นเจ้าของ Object อื่น ๆ ได้

5.3.2 การติดตาม (Trace)

ออบเจ็กต์ติดตาม (Trace object) ตรงกับไฟล์ติดตามของดาต้าไดเรคเตอร์ ไฟล์ติดตาม เป็นล็อคไฟล์ ที่ใช้เก็บคำสั่งที่ดาต้าไดเรคเตอร์ทำการประมวลผล

สิ่งที่เราสามารถกระทำกับ ออบเจ็กต์ติดตาม ได้มีดังนี้

- เปิดหรือปิดการติดตาม
- เพิ่มข้อมูลไปยังไฟล์ติดตาม

ออบเจ็กต์โปรเจค เป็นเจ้าของออบเจ็กต์ติดตาม ออบเจ็กต์ติดตาม ไม่สามารถเป็นเจ้าของออบเจ็กต์อื่น ๆ ได้

6 การใช้ ดาต้าไดเรคเตอร์ออบเจ็กต์

ในส่วนนี้จะอธิบายวิธีการใช้การติดต่อกับดีดีโอ (DDO interface) ในการโปรแกรมในหัวข้อที่สำคัญดังนี้

- การใช้คุณสมบัติ และเมทธอด
- การเริ่มต้นการทำงานดีดีโอ
- การล็คออนไปยังฐานข้อมูล
- การทำงานกับข้อมูล

ตามแบบการตั้งชื่อของอินซแทนซ์ของออบเจ็กต์ในวิชวลเบสิก ในโค้ดตัวอย่างจะมีอักษร o อยู่ด้านหน้าชื่อตัวแปรเหล่านั้นในส่วนนี้ ตัวอย่าง ออบเจ็กต์ดาต้ากรุ๊ป ถูกอ้างถึงด้วยชื่อ oDataGroup ในโค้ดตัวอย่าง

6.1 การใช้ คุณสมบัติ และ เมทธอด

ข้อดีของดีดีโอ คือการซ่อนการเข้าถึงข้อมูลในเมทธอดและคุณสมบัติของออบเจ็กต์เราสามารถใช้เมทธอดและ คุณสมบัติ เหล่านี้ได้ง่าย ๆ โดยการอ้างถึงสิ่งเหล่านั้น

ครั้งหนึ่งที่เราได้สร้างออบเจ็กต์พื้นฐานโดยใช้เมทธอดของดีดีโอและคุณสมบัติของดีดีโอ จะเป็นส่วนที่ติดตามไปในโค้ดของเรา แต่ละดาต้าไดเรคเตอร์ออบเจ็กต์มีเมทธอดและคุณสมบัติ ที่เราสามารถจัดการหรือเรียกขึ้นมาใช้งานได้

6.1.1 การใช้ คุณสมบัติ

การใช้คุณสมบัติของออบเจ็กต์เราจะใช้ชื่อออบเจ็กต์และชื่อคุณสมบัติ เป็นโครงสร้างของรูปแบบคำสั่ง โดยมีรูปแบบทั่ว ๆ ไปดังนี้ คุณสมบัติ จะไม่มีตัวแปรแต่สามารถตั้งค่าที่เหมาะสมโดยใช้ตามรูปแบบคำสั่งตามนี้

```
ObjectName.PropertyName = Value
```

6.1.2 การใช้เมทธอด

การใช้เมทธอดของออบเจ็กต์เราจะใช้ชื่อออบเจ็กต์ ชื่อเมทธอดและตัวแปรของเมทธอดเป็นโครงสร้างของรูปแบบคำสั่ง โดยมีรูปแบบทั่ว ๆ ไป ดังนี้

```
ObjectName.MethodName(Parameter, [Parameter]…)
```

ถ้าเมทธอดไม่มีตัวแปร เราสามารถตัดตัวแปรออกได้ดังนี้

```
ObjectName.MethodName
```

6.1.3 ประโยชน์ของคุณสมบัติและเมทธอด

มีคุณสมบัติและเมทธอดมากมายที่หาได้ในแต่ละออบเจ็กต์แต่มีเมทธอดและคุณสมบัติ บางอันที่เราใช้ประโยชน์เป็นพิเศษได้แก่

- คุณสมบัติของพ่อแม่ (Parent property) แต่ละดาต้าไดเรคเตอร์ออบเจ็กต์มี คุณสมบัติของพ่อแม่ ทำให้เราเคลื่อนย้ายลำดับขั้นของดีดีโอโดยไม่ต้องมีการอ้างถึงที่แน่นอน ตัวอย่าง

  ```
  ObjectSource.[[parentObjectName]…].ObjectDestination.PropertyMethodName
  ```

  คุณสมบัติของพ่อแม่ สามารถเรียกไปได้จนกระทั่งถึงส่วนบนสุดของดีดีโอ(ออบเจ็กต์เอนจิ้น) สำหรับแอพพลิเคชันปัจจุบัน
- คุณสมบัติของแอพพลิเคชันของแต่ละดาต้าไดเรคเตอร์ออบเจ็กต์มีคุณสมบัติของแอพพลิเคชัน ซึ่งใช้ในการส่งค่าการอ้างอิงไปยัง ออบเจ็กต์เอนจิ้น สำหรับแอพพลิเคชันปัจจุบัน การ

อ้างถึงแบบนี้ทำให้เราเขียนโปรแกรมโดยติดต่อโดยตรงกับ ออบเจ็กต์เอนจิ้น โดยไม่ต้องเรียกหลาย ๆ ครั้งผ่านทาง คุณสมบัติของพ่อแม่

- คืนค่าจำนวนตารางหรือคอลัมน์เพื่อใช้นับจำนวนของตารางในดาต้ากรุ๊ป โดยใช้โค้ด ตามนี้

```
oDataGroup.Tables.Count
```

เพื่อใช้นับจำนวนของ คอลัมน์ ในตารางใช้โค้ดต่อไปนี้

```
oDataGroup.Tables("NameOfTable").Columns.Count
```

- คืนค่าที่เป็นชื่อของตาราง ใช้ในการบอกชื่อของตารางจากคอเลคชันของดาต้ากรุ๊ป โดยใช้โค้ดต่อไปนี้

```
oDataGroup.Tables(1).Name
```

เมทธอด **Clone** ใช้ในการสร้างออบเจ็กต์ที่เหมือนกับออบเจ็กต์ปัจจุบันรวมถึงส่วนประกอบทุกอย่างของออบเจ็กต์นั้น ๆ เมทธอดโคลน มีอยู่ในโมเดล และออบเจ็กต์ล็อคออนอินโฟ

6.2 การเริ่มต้นการทำงานกับดีดีโอ

ก่อนที่เราสามารถจะใช้ ดาต้าไดเรคเตอร์ออบเจ็กต์ได้ เราจะต้องสร้างอินซแทนซ์ของออบเจ็กต์ก่อน มีขั้นตอน 2 ขั้นตอนในการเริ่มต้นใช้ดีดีโอ

1. สร้าง ออบเจ็กต์เอนจิ้น
2. สร้าง ออบเจ็กต์โปรเจค

6.2.1 ขั้นตอนที่ 1 สร้าง ออบเจ็กต์เอนจิ้น

ขั้นแรกในการใช้งานดีดีโอคือการสร้าง ออบเจ็กต์เอนจิ้น ขั้นตอนนี้เป็นขั้นตอนที่จำเป็นเพราะว่าออบเจ็กต์เอนจิ้น เป็นออบเจ็กต์ระดับบนสุดในโครงสร้างลำดับขั้นของดีดีโอ ดาต้าไดเรคเตอร์ออบเจ็กต์อื่น ๆ ทั้งหมดจะต้องถูกติดต่อผ่านออบเจ็กต์เอนจิ้น

ในการสร้าง ออบเจ็กต์เอนจิ้น ใช้รูปแบบคำสั่งดังนี้

```
Dim oEngine As New ddoEngine
Set oEngine = CreateObject ("DataDirector.Engine")
Set oEngine = New ddoEngine
```

โค้ดนี้ได้ประกาศตัวแปร oEngine และกำหนดให้เป็นอินซแทนซ์ของดีดีโอออบเจ็กต์เอนจิ้น

6.2.2 ขั้นตอนที่ 2 สร้าง ออบเจ็กต์โปรเจค

หลังจากเราสร้างออบเจ็กต์เอนจิ้นแล้ว ต่อมาเราก็สร้างออบเจ็กต์โปรเจค ขั้นตอนนี้เป็นขั้นตอนที่จำเป็นเพราะว่าทุก ๆ ออบเจ็กต์ในโครงสร้างลำดับขั้นของดีดีโอจะต้องติดต่อผ่านออบเจ็กต์โปรเจค ยกเว้นออบเจ็กต์โมเดล และออบเจ็กต์ล็อคออนอินโฟ

ใช้ เมทธอด **CreateProject** ของออบเจ็กต์เอนจิ้น ในการสร้างออบเจ็กต์โปรเจค ดังนี้

```
Dim oProject As ddoProject
Set oProjcet = oEngine.CreateProject
```

```
oProject.Name = "Virtual Library"
```

โค้ดนี้มีการสร้าง ออบเจ็กต์โปรเจค ที่มีชื่อว่า Virtual Library

สิ่งสำคัญ: แต่ละครั้งที่แอพพลิเคชันเรียก **CreateProject** ดาต้าไดเรคเตอร์ จะสร้างออบเจ็กต์โปรเจค ใหม่ โดยจะไม่อ้างถึงโปรเจคแรกที่ได้สร้างมาก่อน ดาต้าไดเรคเตอร์จะเริ่มต้นใช้งานแต่ละโปรเจค ใหม่ในเทรด (Thread) ของมันเองเพื่อความคล่องตัว

6.2.3 ขั้นตอนที่ 3 การสร้างแแบเจ็กต์ อื่น ๆ

หลังจากเราสร้างดาต้าไดเรคเตอร์ออบเจ็กต์ขั้นพื้นฐานมาแล้ว เราสามารถสร้างออบเจ็กต์ต่าง ๆ ในลำดับขั้นของดีดีโอได้ แต่ละออบเจ็กต์ที่ถูกสร้างจะใช้เมทธอดจากออบเจ็กต์พ่อแม่ของมัน ตัวอย่างเช่น เราสร้าง ออบเจ็กต์ดาต้ากรุ๊ป โดยใช้เมทธอด **CreateDataGroup** ของออบเจ็กต์โปรเจค ซึ่งเป็นพ่อแม่ของ ออบเจ็กต์ดาต้ากรุ๊ป ทำนองเดียวกันเราสร้างดาต้าลิงค์ โดยใช้เมทธอด **CreateLink** ของ ออบเจ็กต์ดาต้ากรุ๊ป

6.2.3.1 การสร้างออบเจ็กต์โมเดล

หลังจากเราสร้างออบเจ็กต์เอนจิ้น แล้วเราสามารถสร้างออบเจ็กต์โมเดล เพื่อใช้เข้าถึงตารางและคอลัมน์ในฐานข้อมูลของเราได้ ตัวอย่างเช่น เราใช้ข้อมูลที่เก็บในโมเดลเพื่อสร้างดาต้าลิงค์ระหว่างคอลัมน์ ในฐานข้อมูลของเรากับคอนโทรลในวิชวลเบสิก

เราใช้เมทธอด **CreateModel** ของออบเจ็กต์เอนจิ้น ในการสร้างออบเจ็กต์โมเดล ดังแสดงในตัวอย่างต่อไปนี้

```
Dim oModel As ddoModel
Set oModel = oEngine.CreateModel ("c:\Program Files\Microsoft Visual
Basic\library.mlt")
```

โค้ดนี้ใช้สร้างออบเจ็กต์โมเดล บนไฟล์โมเดลที่มีอยู่แล้วที่ชื่อ library.mlt ไฟล์โมเดล นี้ถูกสร้างโดยการอิมพอร์ตโมเดล (Model Import Wizard) โดยผ่านทางการติดต่อผ่านผู้ใช้ของดาต้าไดเรคเตอร์

6.2.3.2 การสร้างออบเจ็กต์ดาต้ากรุ๊ป

หลังจากเราสร้างออบเจ็กต์โปรเจค เราสามารถสร้าง ออบเจ็กต์ดาต้ากรุ๊ป เพื่อเข้าถึงและจัดการกับข้อมูลในฐานข้อมูลของเราขณะ Runtime ได้

เราใช้เมทธอด **CreateDataGroup** ของออบเจ็กต์โปรเจค ในการสร้างออบเจ็กต์ดาต้ากรุ๊ป ดังแสดงในตัวอย่างด้านล่างนี้

```
Dim oDataGroup As ddoDataGroup
Set oDataGroup = oProject.CreateDataGroup ("book", "DGBook", "c:\Program
Files\Microsoft Visual Basic\library.mlt")
```

ตัวอย่างโค้ดนี้จะเป็นการสร้าง ออบเจ็กต์ดาต้ากรุ๊ป ที่ชื่อ DGBook ซึ่งอยู่บนไฟล์โมเดล ชื่อ library.mlt มันเป็นรายละเอียดเกี่ยวกับตาราง Book ขณะเป็น ตารางหลัก สำหรับ ดาต้ากรุ๊ป นี้

6.3 การ ล็อคออนไปยังฐานข้อมูล

ในส่วนนี้จะแสดงโค้ดที่สาธิตการใช้ออบเจ็กต์ล็อคออนอินโฟ และการล็อคออนไปยังฐานข้อมูลโดยผ่านทางโมเดลหรือออบเจ็กต์ดาต้ากรุ๊ป สำหรับออบเจ็กต์โมเดล เราจะใช้ข้อมูลชนิดพิเศษในการล็อคออนเพื่อนำโครงสร้างของฐานข้อมูลไปเก็บในโมเดลขณะรันโปรแกร เราจะใช้ข้อมูลชนิดพิเศษในการล็อคออนสำหรับ ออบเจ็กต์ดาต้ากรุ๊ป เพื่อเข้าถึงและจัดการกับข้อมูลในฐานข้อมูลของเราผ่านทางดาต้ากรุ๊ป

วิธีต่าง ๆในการ ล็อคออน ไปยังฐานข้อมูล

- ตั้งค่าตัวแปรล็อคออน โดยการสร้างออบเจ็กต์ล็อคออนอินโฟ
- ใช้วิธีอิมพอร์ตของ ออบเจ็กต์โมเดล
- ใช้วิธีล็อคออนของ ออบเจ็กต์ดาต้ากรุ๊ป หรือไม่ก็โดยการใช้ออบเจ็กต์ล็อคออนอินโฟ หรือโดยการส่งค่าตัวแปรเฉพาะโดยตรงใน ล็อคออน

6.3.1 การสร้างออบเจ็กต์ล็อคออนอินโฟ

ในการอธิบายข้อมูลในการ ล็อคออน สำหรับฐานข้อมูลทำได้โดยใช้เมทธอด **CreateLogonInfo** ของ โมเดล หรือ ออบเจ็กต์ดาต้ากรุ๊ป เพื่อตั้งค่าของออบเจ็กต์ล็อคออนอินโฟ

สิ่งสำคัญเล็ก ๆ น้อยที่เราต้องเก็บเอาไว้ในใจเพื่อพิจารณาเมื่อเราใช้ออบเจ็กต์ล็อคออนอินโฟ

- ถ้าเราล็อคออนไปยังฐานข้อมูลโดยการใช้ออบเจ็กต์ดาต้ากรุ๊ป และไม่ได้ส่งข้อมูลเกี่ยวกับการล็อคออนหรือไม่ได้ส่งค่าตัวแปรทั้งหมดสำหรับออบเจ็กต์ล็อคออนอินโฟ ทำให้ดาต้าไดเรคเตอร์ ใช้ตัวแปรที่เหมือนกันสำหรับดาต้ากรุ๊ป ที่เราเคยใช้ในออบเจ็กต์ล็อคออนอินโฟตอนนำโมเดลเข้ามา
- ถ้าเราใช้ค่าข้อมูลที่ตั้งมาจากโมเดลออบเจ็กต์ล็อคออนอินโฟ เมื่อทำการล็อคออนไปยังดาต้ากรุ๊ป ทำให้ ดาต้าไดเรคเตอร์ไม่ได้จัดเตรียมรหัสผ่านใช้เป็นค่าตั้งต้น
- เพียงครั้งเดียวที่เราได้ส่งข้อมูลในการล็อคออนให้ดาต้ากรุ๊ป ดาต้าไดเรคเตอร์จะใช้ค่านี้ทุกครั้งเป็นค่าตั้งต้นสำหรับการล็อคออนครั้งต่อไป เราสามารถตั้งค่าตั้งต้นได้ใหม่โดยการส่งตัวแปรค่าใหม่ไปยังล็อคออนของดาต้ากรุ๊ปเมื่อมันต้องการ

ในการสร้างออบเจ็กต์ล็อคออนอินโฟ ทำได้โดยใช้เมทธอด**CreateLogonInfo** ของออบเจ็กต์โมเดล หรือไม่ก็ของ ออบเจ็กต์ดาต้ากรุ๊ป

ตัวแปรของออบเจ็กต์ล็อคออนอินโฟ มีดังต่อไปนี้

- varUser
- varPassword
- varServer
- varDatabase

- varDBMSName (ในส่วนนี้จะอ้างถึงตัวไดรเวอร์ของฐานข้อมูลและทุกครั้งจะมีค่าเป็น ODBC)
- varDatasource
- varName

ถ้าเราใช้ตัวแปรทั้งหมดของออบเจ็กต์ล็อคออนอินโฟ เราจะได้รูปแบบของคำสั่งดังต่อไปนี้

```
Dim oLogonInfo As ddoLogonInfo
Set oLogonInfo = oDataGroup.CreateLogonInfo("User", "Password", "Server", "Database", "ODBC", "DataSource")
```

6.3.1.1 การตั้งค่าออบเจ็กต์ล็อคออนอินโฟ สำหรับ ดาต้ากรุ๊ป

ตัวอย่างต่อไปนี้แสดงว่าจะสร้างออบเจ็กต์ล็อคออนอินโฟ อย่างไรสำหรับดาต้ากรุ๊ป ตัวอย่างนี้สมมติว่าเราได้สร้างออบเจ็กต์ต่าง ๆ ต่อไปนี้แล้วคือ

- ออบเจ็กต์เอนจิ้น (DataDirector.Engine)
- ออบเจ็กต์โปรเจค (sports.ddx)
- ออบเจ็กต์ดาต้ากรุ๊ป (DGCustomer)

สำหรับจุดประสงค์ของตัวอย่างนี้คือ การส่งค่าชื่อของยูสเซอร์ (admin), รหัสผ่าน (admin), ฐานข้อมูล (sports), ชื่อของดีบีเอ็มเอส (DBMS) (ODBC) และดาต้าซอร์ส (Data source) (sports) เมื่อเราล็อคออนไปยังฐานข้อมูลขณะรันโปรแกรมโดยดาต้าไดเรคเตอร์ จะล็อคออนโดยไม่มีการแสดงข้อมูลการ ล็อคออนให้ผู้ใช้เห็น

```
' Create a LogonInfo object for the DataGroup
Dim oLogonInfo As ddoLogonInfo
Set oLogonInfo = oDataGroup.CreateLogonInfo("admin", "admin", "sports", "ODBC", "sports")
```

6.3.1.2 การใช้พับบลิกฟังก์ชัน (Public)สำหรับชื่อของยูสเซอร์ และรหัสผ่าน

เราสามารถอธิบายพับบลิกฟังก์ชัน ในการตั้งค่าชื่อยูสเซอร์ และรหัสผ่าน ในออบเจ็กต์ล็อคออนอินโฟ ได้ดังนี้

```
Public Function SetLogonInfo(szName As String, szPassword As String, oLogonInfo As Object) As Boolean
```

ตัวอย่างต่อไปนี้ใช้พับบลิกฟังก์ชันใน **SetLogonInfo** มันจะเช็คว่าได้มีการสร้างออบเจ็กต์ล็อคออนอินโฟไว้หรือยัง ถ้ายังมันจะสร้างขึ้นมาอันหนึ่ง ถ้าออบเจ็กต์ล็อคออนอินโฟมีอยู่แล้วโค้ดนี้จะตั้งค่าตัวแปร ชื่อยูสเซอร์ และรหัสผ่าน ไปยังออบเจ็กต์ที่มีอยู่แล้ว

```
Public Function SetLogonInfo(szName As String, szPassword As String, oLogonInfo As Object) As Boolean
On Error GoTo Error
```

```
If oLogonInfo Is Nothing Then
        Set oLogonInfo = oDataGroup.CreateLogonInfo(szName, szPassword,
        "sports", "ODBC", "sports")
Else
        oLogonInfo.User = szName
        oLogonInfo.Password = szPassword
End If
        SetLogonInfo = True
Exit Function
        Error:
        SetLogonInfo = False
End Function
```

6.3.2 การ ล็อคออนจริง (Actual)

เพื่อแสดงการล็อคออนจริงด้วยการใช้เมทธอดอิมพอร์ต สำหรับออบเจ็กต์โมเดลหรือเมทธอดล็อคออน สำหรับออบเจ็กต์ดาต้ากรุ๊ป ดังที่ได้อธิบายไว้แล้วในส่วนนี้

6.3.2.1 การเข้าถึงโครงสร้างฐานข้อมูล

หลังจากเราได้สร้างออบเจ็กต์ล็อคออนอินโฟ แล้วเราใช้เมทธอดอิมพอร์ตของออบเจ็กต์โมเดลเพื่อนำโครงสร้างของฐานข้อมูลเข้ามาในไฟล์โมเดล เมื่อเรานำเข้าโครงสร้างฐานข้อมูลไปที่โมเดล ตัวดาต้าไดเรคเตอร์ จะแสดงการล็อคออนเข้าฐานข้อมูล ,อิมพอร์ตฐานข้อมูลและล็อคออฟโดยอัตโนมัติ

```
oModel.Import
```

โค้ดนี้เอาตัวแปรที่อยู่ในออบเจ็กต์ล็อคออนอินโฟ และใช้ข้อมูลเหล่านั้นในการล็อคออนไปยังฐานข้อมูลโดยการใช้เมทธอด **Import** จากโมเดล ซึ่งก็คือ oModel

6.3.2.2 การ ล็อคออน ไปยังดาต้ากรุ๊ปโดยตรง

หลังจากเราสร้างออบเจ็กต์ล็อคออนอินโฟ เราใช้เมทธอด **Logon** ของ ออบเจ็กต์ดาต้ากรุ๊ป ในการ ล็อคออน ไปยังฐานข้อมูล

```
oDataGroup.Logon
```

โค้ดนี้เอาตัวแปรที่อยู่ในออบเจ็กต์ล็อคออนอินโฟ และใช้ข้อมูลเหล่านั้นในการล็อคออนไปยังฐานข้อมูลโดยการใช้เมทธอด **Logon** จากดาต้ากรุ๊ปซึ่งก็คือ oModel

6.3.2.3 การ ล็อคออน โดยไม่ใช้ออบเจ็กต์ล็อคออนอินโฟ

เราสามารถล็อคออน ไปยังฐานข้อมูลได้โดยไม่ใช้ออบเจ็กต์ล็อคออนอินโฟ ได้โดยส่งตัวแปรเฉพาะในการล็อคออน โดยการใช้ล็อคออน ของออบเจ็กต์ดาต้ากรุ๊ป ดังตัวอย่างด้านล่างนี้

```
oDataGroup.Logon "MyName", "MyPassword"
```

การทำงานนี้จะ ล็อคออน ไปยังฐานข้อมูลด้วย *MyName* และ *MyPassword* เพื่อตั้งค่า ล็อคออนของผู้ใช้และ รหัสผ่าน เราสามารถตั้งค่าเซิร์ฟเวอร์, ฐานข้อมูล, ชื่อดีบีเอ็มเอส และ ดาต้าซอร์สถ้าเราต้องการ สังเกตได้ว่าวิธีการนี้วิธีการนี้จะไม่ใช้ตัวแปร (0) เป็นส่วนหนึ่งของรูปแบบคำสั่ง

6.3.2.4 การ ล็อคออน โดยไม่ใช้ข้อมูลใด ๆ ของ โมเดล

บางครั้งวิธีนี้ก็จำเป็นในการใช้ล็อคออนดาต้าไดเรคเตอร์ โดยไม่ใช้ข้อมูลของโมเดล ตัวอย่างเช่น เราจะทำสิ่งนี้ถ้าเราพัฒนาแอพพลิเคชันโดยใช้ฐานข้อมูลอันหนึ่งอยู่ แต่เปลี่ยนแอพพลิเคชันไปยังฐานข้อมูลอื่น ๆ ในการจะทำสิ่งนี้ให้ตั้งค่าคุณสมบัติของ ล็อคออนโดยใช้โมเดลของดาต้ากรุ๊ป (DataGroup LogonUsingModel) ให้เป็น False

6.4 การทำงานกับข้อมูล

ในส่วนนี้จะแสดงตัวอย่างโค้ดที่แสดงว่าจะทำงานกับข้อมูลในการเขียนโปรแกรมอย่างไร รวมถึงการสร้างดสต้สลิงค์,คิวรีข้อมูล, แสดงข้อมูล, และการใช้ข้อมูลโอเปค

6.4.1 การสร้างดาต้าลิงค์โดยใช้ดีดีโอ

ดาต้าลิงค์ เป็นการติดต่อที่เราสร้างขึ้นมาระหว่าง คอลัมน์ ในฐานข้อมูลกับส่วนที่ใช้คอนโทรลของวิชวลเบสิก เราสามารถสร้างลิงค์เสมือน ซึ่งใช้ชี้เฉพาะ คอลัมน์ ของฐานข้อมูลแต่ไม่ได้ชี้ไปที่คอนโทรลของวิชวลเบสิก ลิงค์เสมือนทำให้เราเรียกค้นข้อมูลจากโปรแกรมโดยไม่ต้องมีการแสดงมันในแอพพลิเคชันขณะทำงาน

เมื่อเราสร้างดาต้าลิงค์ เราจะต้องตั้งเส้นทางของข้อมูลของแต่ละดาต้าลิงค์ไปยังตารางและคอลัมน์ เฉพาะที่ซึ่งดาต้าลิงค์ ทำการเชื่อมไปหา เส้นทางของข้อมูลจะต้องมีลักษณะเฉพาะกับตารางหลักขณะที่เป็นรูทดังนั้นลิงค์ที่ไปยัง คอลัมน์ แล้วสามารถถูกหาที่อยู่ได้โดยดาต้าไดเรคเตอร์ ตามด้วยเส้นทางที่เริ่มต้นที่ ตารางหลัก

อย่างไรก็ตามเพื่อสร้างดาต้าลิงค์ ไปยังคอลัมน์ ที่ไม่สามารถไปถึงจากตารางหลักได้ คอลัมน์เหล่านั้นจะถูกเรียกค้นเมื่อ ดาต้ากรุ๊ป ถูกคิวรีแต่ไม่มี มาสเตอร์-ดีเทล (Master-detail) เกิดขึ้นพร้อมกัน

ใช้เมทธอด **CreateLink** ของออบเจ็กต์ดาต้ากรุ๊ป เพื่อสร้างดาต้าลิงค์ สำหรับรูปแบบคำสั่งของดาต้ากรุ๊ป ของดาต้าลิงค์ เป็นดังนี้

```
Set DataLink = DataGroup.CreateLink(varColumn As Variant, [varControl1 As
Variant]) As Object
```

ตัวแปร varControl เป็นออปชันถ้าเราต้องการสร้างลิงค์เสมือนที่ไม่เป็นลิงค์จริง ๆ ไปยังคอนโทรลชองวิชวลเบสิก สำหรับตัวแปร varControl เราสามารถส่งค่าในรูปของชื่อเฉพาะของคอนโทรลของวิชวลเบสิก (form1.text1) ถ้าเราต้องการติดต่อ คอลัมน์ ของฐานข้อมูลไปยังตัวคอนโทรล

เราสามารถสร้าง ดาต้าลิงค์ เพื่อออบเจ็กต์ที่ไม่ใช่คอม (Non-visual COM object) ซึ่งดาต้าลิงค์ชนิดนี้มีประสิทธิภาพมากกว่าดาต้าลิงค์เสมือน เพราะว่าทำให้แอพพลิเคชันของเราไม่นานเกินไปเมื่อมีการร้องของข้อมูลทุก ๆ ครั้งที่มีการเปลี่ยนแปลง

6.4.2 การระบุเส้นทางข้อมูล

มีทาง 3 ที่เราสามารถระบุเส้นทางข้อมูลสำหรับดาต้าลิงค์

- รายละเอียดของตารางเริ่มต้นและคอลัมน์
- รายละเอียดเส้นทางบางส่วน (Partial path) และให้ ดาต้าไดเรคเตอร์ แสดงความสัมพันธ์ที่แน่นอนซึ่งอยู่บนพื้นฐานของความสัมพันธ์ในโมเดล
- รายละเอียดเส้นทางทั้งหมดเริ่มต้นที่ ตารางหลัก

คำแนะนำ : ถ้าโค้ดของเรามีเส้นทางข้อมูลที่ไม่ครบหรือมีหลายเส้นทางสำหรับดาต้าลิงค์ จะทำให้ ดาต้าไดเรคเตอร์ ฟ้องความผิดพลาดขึ้นมา

โค้ดตัวอย่างในส่วนนี้สาธิตว่าจะสร้างดาต้าลิงค์และดาต้าลิงค์เสมือนที่สมบูรณ์ได้อย่างไร บางตัวอย่างอาจแสดงวิธีการที่ต่างกันที่เราสามารถตั้งเส้นทางของข้อมูลที่ต่างกันของดาต้าลิงค์ ได้ ตัวอย่างโค้ดนี้สมมติว่า ตารางหลัก สำหรับ oDataGroup คือตารางลูกค้าและ คอนโทรลของวิชวลเบสิกที่เรากำลังจะสร้าง ดาต้าลิงค์ ไปชื่อว่า form1.text1

6.4.2.1 การระบุตารางเริ่มต้นและ คอลัมน์

โค้ดด้านล่างนี้สร้างดาต้าลิงค์โดยตรงไปยังคอลัมน์ของ customer_num ในตารางลูกค้า

```
Set olnkCustID = oDataGroup.CreateLink
("customer.customer_num", form1.text1)
```

6.4.2.2 การระบุ Partial Path

โค้ดด้านล่างนี้ระบุเส้นทางของข้อมูลและให้ดาต้าไดเรคเตอร์อ้างถึงความสัมพันธ์

```
Set olnkOrderID =
oDataGroup.CreateLink("customer\order.order_num", form1.text1)
```

6.4.2.3 การระบุเส้นทางทั้งหมด

โค้ดด้านล่างนี้อธิบายเส้นทางทั้งหมดอย่างชัดจน

```
Set olnkCustID = oDataGroup.CreateLink
("customer.customer_num\orders.customer_num" , form1.text1)
```

6.4.2.4 การสร้าง ลิงค์เสมือน

โค้ดด้านล่างนี้ไม่ได้ตั้งค่าตัวแปร varControl และสร้างฮาร์ฟลิงค์ไปยังคอลัมน์ customer_num ในตารางลูกค้า

```
Set olnkCustID=
oDataGroup.CreateLink("customer.customer_num")
```

โค้ดด้านล่างนี้ไม่ได้ตั้งค่าตัวแปร varControl และสร้างฮาฟลิงค์ (Half-link) ไปยังคอลัมน์ order_num ในตาราง order โดยใช้การแสดงเส้นทางแบบสมบูรณ์ดังนี้

```
Set olnkOrderID =
oDataGroup.CreateLink("customer\orders.order_num")
```

6.4.3 การ เรียกค้นข้อมูล

การ เรียกค้น ข้อมูลนำไปสู่การไปเอาข้อมูลจากฐานข้อมูลมาโดยปกติเพื่อแสดงในแอพพลิเคชันบนวินโดวส์ด้วยดีดีโอข้อมูลที่ถูกเรียกค้น จะถูกกระทำผ่านตาราง และออบเจ็กต์ดาต้ากรุ๊ป ซึ่งมีระยะที่กว้างเพื่อเรียกค้นข้อมูลโดยใช้ดีดีโอรวมถึงเมื่อแอพพลิเคชันเป็นการทำงานในโหมด **Query by example** (QBE) และเมื่อการ เรียกค้น โดยใช้คำสั่งในการทำงาน (Executable statementป

6.4.3.1 การ เรียกค้น ตาราง

เราสามารถแก้ไขข้อมูลที่ถูก เรียกค้น สำหรับ คอลัมน์ จาก ตาราง, เรคคอด หรือออบเจ็กต์ดาต้าลิงค์

ตัวอย่างต่อไปนี้สาธิตการ เรียกค้น ข้อมูลว่าทำงานอย่างไรจากตารางลูกค้าในออบเจ็กต์ oDataGroup

```
Set QueryTable = oDataGroup.Tables("customer")
QueryTable.QueryData
```

6.4.3.2 การ เรียกค้น ข้อมูลในโหมดคิวบีอี (QBE)

ในการ เรียกค้น แบบคิวบีอีต้องทำตามขั้นตอนดังนี้

1. บอกให้ดาต้าไดเรคเตอร์ รู้ว่าเราต้องการจัดการคิวบีอีโดยใช้การตั้งค่าต่อไปนี้

   ```
   DataGroup.StartQBE = True
   ```

   สถานะปัจจุบันของดาต้ากรุ๊ป จะถูกรักษาไว้และออบเจ็กต์ตาราง จะถูกเตรียมสำหรับ การทำงานในโหมดคิวบีอี
2. ตั้งค่าเงื่อนไขของคิวบีอี ในคอลัมน์ที่เกี่ยวข้องกับคิวบีอี เราสามารถทำสิ่งนี้ได้โดยการตั้งค่าข้อมูลในคอลัมน์ ดังนี้

   ```
   oDataGroup.StartQBE = True
   oCustomers.Columns("Name").Value='A%'
   ```
3. จัดการกับ DataGroup.QueryData หรือ DataGroup.Tables(index).QueryData

ผลของคิวบีอีจะมีในดาต้ากรุ๊ปของเรา สังเกตว่าหลังจากจัดการกับคำสั่ง คิวบีอี ทำให้คุณสมบัติของโหมดคิวบีอีบน ดาต้ากรุ๊ป ถูกตั้งค่าให้เป็นฟอล์ลโดยอัตโนมัติ

ตัวอย่างต่อไปนี้สาธิตว่าจะทำอย่างไรในการใช้โหมดคิวบีอีในการเรียกค้นข้อมูล ตัวอย่างนี้ใช้พับบลิกฟังก์ชันที่ชื่อ **QueryCustomer** ชื่อของตารางที่ใช้ในการเรียกค้นถูกส่งเป็นค่าตัวแปรชื่อ szTableName

```
‘ Query a customer whose name is passed in as szName
Public Function QueryCustomer(szTableName as string) as
Object
Dim oTable as object
```

```
On error resume next
' Let Data Director know that you are starting QBE query
oDataGroup.StartQBE = True
' Set the customer surname into the LAST NAME column
oDataGroup.Tables("customer").Columns("lname").Value =
szCustName
' Execute the query
Set QueryCustomer = oDataGroup.Tables("customer")
QueryCustomer.QueryData
End Function
```

6.4.3.3 การแทนที่ค่าของ คอลัมน์ ด้วยผลการคำนวณ

ตัวอย่างโค้ดต่อไปนี้ใช้แทนที่ค่าในคอลัมน์ที่มีอยู่แล้วชื่อ price ด้วยการคำนวณค่าจากคอลัมน์ที่ชื่อ calc

```
Set price = oDataGroup.CreateLink("items.price")
Set calc = oDataGroup.CreateLink("items.calc")
calc.expression = calc.value * 1.05
oDataGroup.QueryData
price.value = calc.value
```

6.4.3.4 การ เรียกค้นด้วยคำสั่งในการเอ็กซ์ซีคิว (Executable Statements)

ตารางเสมือนอนุญาติให้เราส่งค่าคำสั่งเอสคิวแอลหรือคำสั่งที่เรากำหนดขึ้นหรือคำสั่งในการเอ็กซ์ซีคิว ได้โดยตรงโดยผ่านดาต้าไดเรคเตอร์เพื่อเรียกค้น ฐานข้อมูลของเราใช้ ตารางเสมือน เพื่อจัดการกับคำสั่งเอสคิวแอล เช่น Ad hoc query, Subquery, Group-by clause และสตรอโปรซเยอร์

ใช้คุณสมบัติ **QueryCommand** ของออบเจ็กต์ตารางเพื่อรวบรวมคำสั่งในการเรียกค้นที่สัมพันธ์กับตารางเสมือนและใช้เมทธอด **ExecuteCommand** ของออบเจ็กต์ตาราง ในการจัดการกับคำสั่งในการเรียกค้น หลังจากเราสร้าง ตารางเสมือน เราสามารถเพิ่มคุณสมบัติที่หาได้ไปยังออบเจ็กต์ตาราง ได้เพื่อสร้างและจัดการคำสั่งอินเซอร์ท (INSERT), อัพเดต (UPDATE) และดีลีต (DELETE) ถ้าต้องการข้อมูลมากขึ้นเกี่ยวกับ ตารางเสมือน ให้ไปดูที่ดาต้ากรุ๊ป ตัวอย่างข้างล่างนี้จะเป็นการสร้างตารางเสมือน ที่มีชื่อว่า **oVTable** มีการกำหนดคำสั่งในการรันในคุณสมบัติ **QueryCommand** ส่งคำสั่งผ่านเมทธอด **SetParam** และหลังจากนั้นก็ทำการคิวรี

```
' Query a customer whose name is passed in as szName
Function QueryCustomer(szName as String) as Object
Dim oVTable as object
' Instantiate virtual table
```

```
Set oVTable = oDataGroup.CreateVirtualTable()
' Define SQL query statement
oVTable.QueryCommand = "SELECT * from customers where name = ?"
' Set name parameter
oVTable.SetParam QUERY_CMD, 1, szName
' Execute the query
oVTable.ExecuteQueryCommand
End Function
```

6.4.3.5 การคิวรีกอลัมน์ที่เป็นโรว์ไทป์ (Row Type)

โรว์ไทป์ จะเก็บข้อมูลจากหลาย ๆ คอลัมน์เป็น 1 คอลัมน์ ตัวอย่างเช่น ข้อมูลชนิดที่อยู่จะประกอบไปด้วยถนน,เมือง ,รัฐ และอื่น ๆ ตัวอย่างข้างล่างนี้จะเป็นเลือกข้อมูลจากตาราง **Employee** zip code จะถูกส่งผ่านเป็นพารามิเตอร์ของฟังก์ชัน ในฐานข้อมูล zip code จะเก็บในคอลัมน์ชื่อ **code** ภายในโรว์ไทป์ ที่ชื่อ **zip** ซึ่งตัวมันเองยังจะเป็นคอลัมน์หนึ่งของโรว์ไทป์ที่ชื่อ **address**

```
Set oVTable = oDataGroup.CreateVirtualTable()
oVTable.QueryCommand = "SELECT * from employee where address.zip.code = ?"
oVTable.SetParam QUERY_CMD, 1, code
oVTable.ExecuteQueryCommand
' Print the value stored in the first column of the virtual table:
Debug.print ovTable.columns(1).value
```

<u>หมายเหตุ</u> ตัวอย่างนี้สมมติว่าไดร์เวอร์ฐานข้อมูลโอดีบีซี (ODBC database driver) รองรับการส่งผ่านพารามิเตอร์

6.4.3.6 การตรวจสอบผลลัพธ์

หลังจากที่ได้คิวรีตารางหรือตารางเสมือน สามารถที่จะดูผลลัพธ์ได้ซึ่งตัวอย่างข้างล่างนี้ใช้ตารางเสมือน ที่ชื่อ **ovTable**

ประการแรก อาจจะแสดงข้อความ ถ้าไม่มีข้อมูลของการคิวรี เช่น

```
If ovTable is Nothing Then
MsgBox "No results were returned."
End If
```

ประการที่สอง อาจจะอ่านผลลัพธ์ขึ้นมาและแสดงผลเรคคอดนั้น ๆ

```
If Not ovTable is Nothing then
For i = 0 to ovTable.Columns.Count - 1
Debug.Print ovTable.Columns (i + 1).Value
ovTable.NextRecord
Next
End If
```

6.4.4 การแสดงผลข้อมูล

ตัวอย่างข้างล่างนี้แสดงผลของการคิวรีข้อมูลของลูกค้าในแอพพลิเคชัน

```
‘ Display a customer record
Private Sub DisplayCustomer(Record As Object)
tbCust_Id = Record.Columns(“customer_num”).Value
tbName = Record.Columns(“lname”).Value
tbContact = Record.Columns(“fname”).Value
tbAddress1 = Record.Columns(“address1”).Value
tbAddress2 = Record.Columns(“address2”).Value
tbCity = Record.Columns(“city”).Value
tbState = Record.Columns(“state”).Value
tbZip = Record.Columns(“postal_code”).Value
tbPhone = Record.Columns(“phone”).Value
End Sub
```

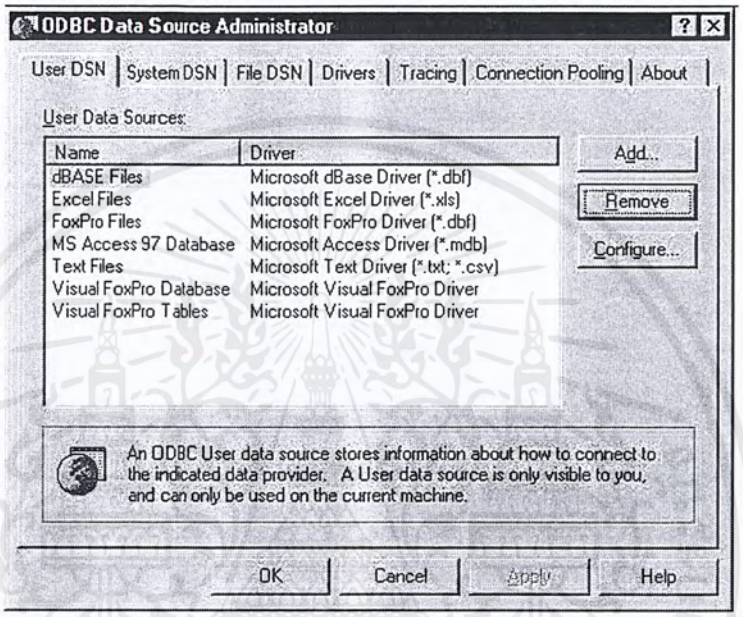
ภาคผนวก ง

การสร้างการติดต่อระหว่าง Visual Basic กับฐานข้อมูลโดยผ่าน Data Director

การสร้างการติดต่อระหว่าง Visual Basic กับฐานข้อมูลโดยผ่าน Data Director นั้นมีขั้นตอนดังนี้

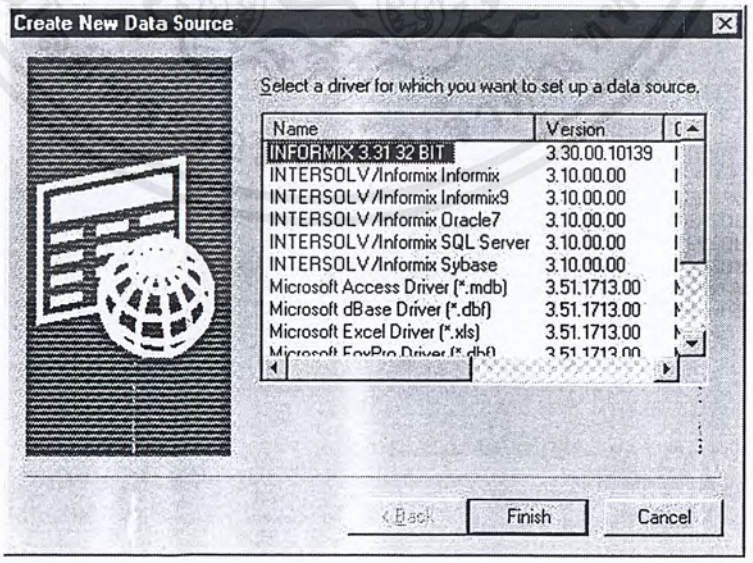
1. **สร้าง Data Source ใน ODBC Driver** มีรายละเอียดดังนี้

1.1 ไปที่ Start Menu -> Setting -> Control Panel เลือก ODBC จะปรากฏดังรูปที่ ง-1



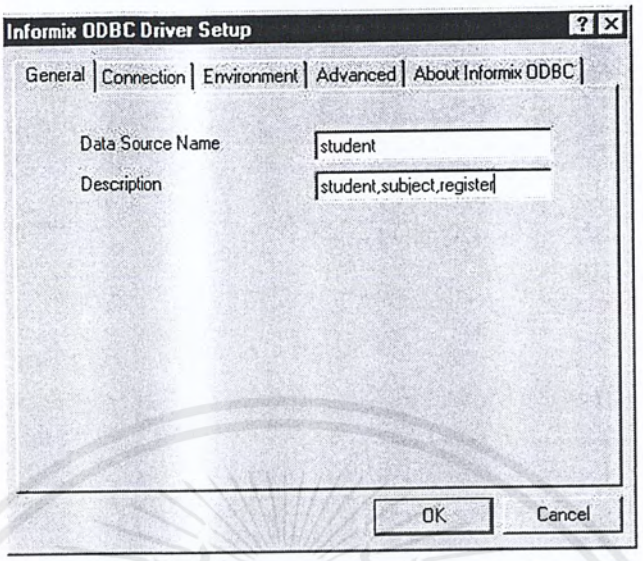
รูปที่ ง-1 ODBC Data Source Administrator

1.2 สร้าง Data Source โดยการ คลิก Add... จะได้ดังรูปที่ ง-2



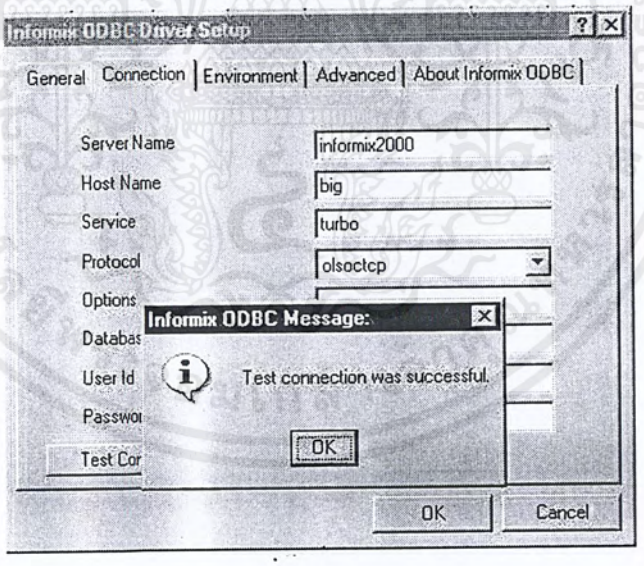
รูปที่ ง-2 Create New Data Source

1.3 ใส่ชื่อของ Data Source Name ที่ต้องการ แล้วคลิกไปที่ Tab Connection จะได้ดังรูปที่ ง-3



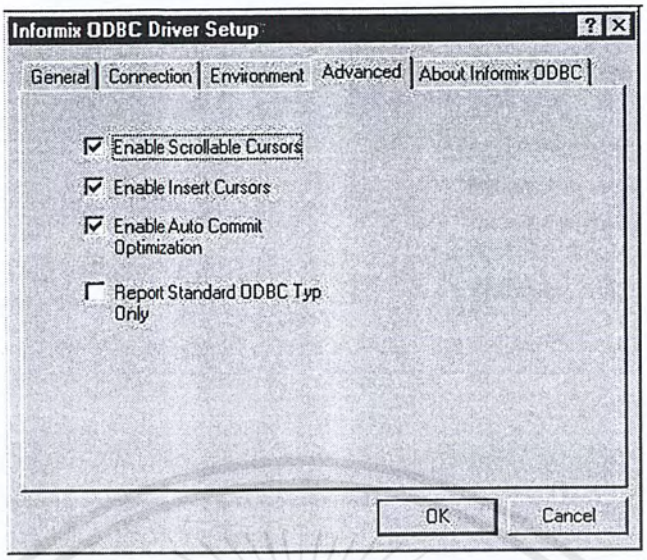
รูปที่ ง-3 การตั้งชื่อ Data Source Name

1.4 ใส่ชื่อ Server Name Host Name Database Name แล้วคลิกปุ่ม Test Connection ได้ดังรูปที่ ง-4



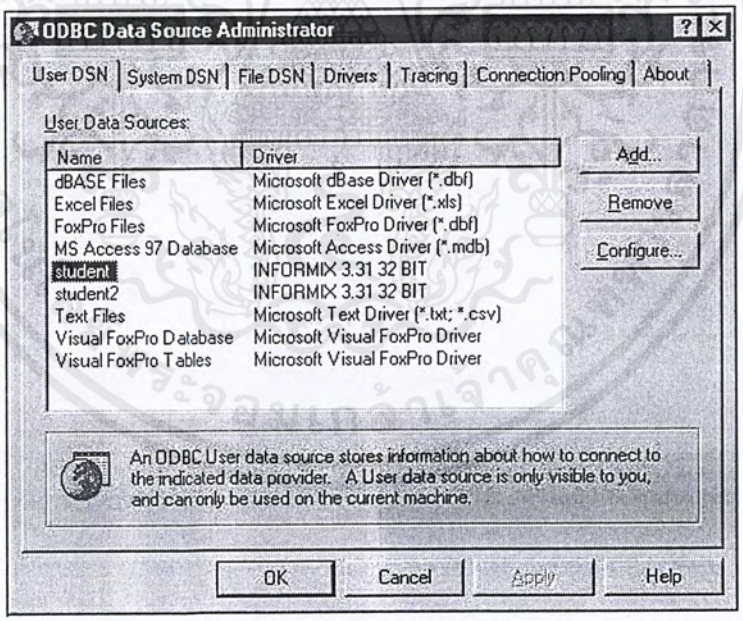
รูปที่ ง-4 Informix ODBC Driver setup

1.5 เลือก tab Advanced และเลือก Enable จะได้ดังรูปที่ ง-5



รูปที่ ง-5 Informix ODBC Driver setup การเลือก tab Advanced

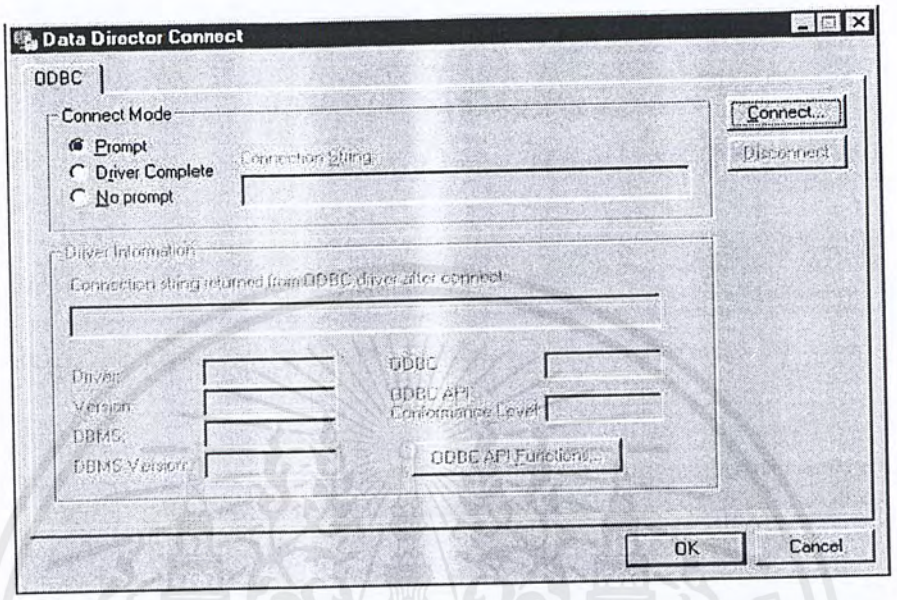
1.6 ภายหลังจากกาคลิกปุ่ม OK ก็ถือว่าเสร็จแล้วจะได้ดังรูปที่ ง-6



รูปที่ ง-6 แสดงการเสร็จสิ้นการสร้าง Student Data Sources Name(DSN)

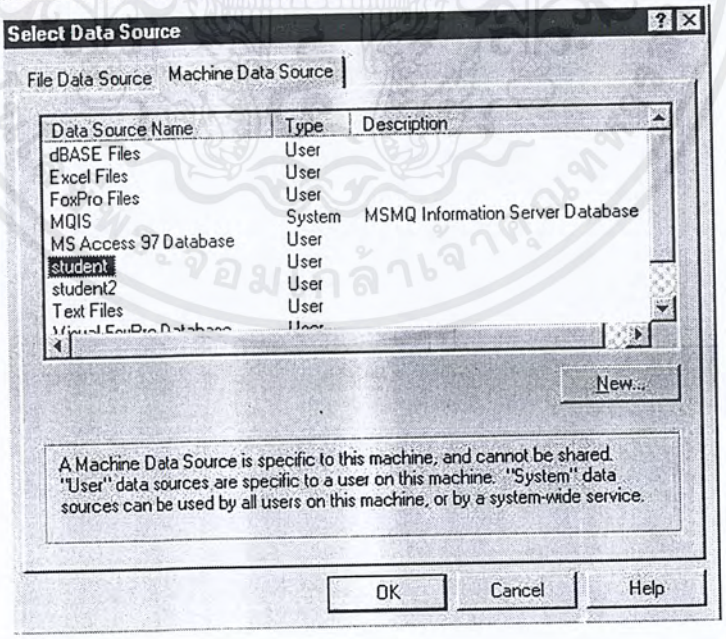
2. ทดสอบการเชื่อมต่อระหว่าง Data Director กับ ฐานข้อมูลโดยใช้ Data Director Connect (อยู่ในชุด Software ของ Data Director)

2.1 ไปที่ Start Menu -> Program -> Data Director 3.5 -> Data Director Connect จะปรากฏดังรูปที่ ง-7



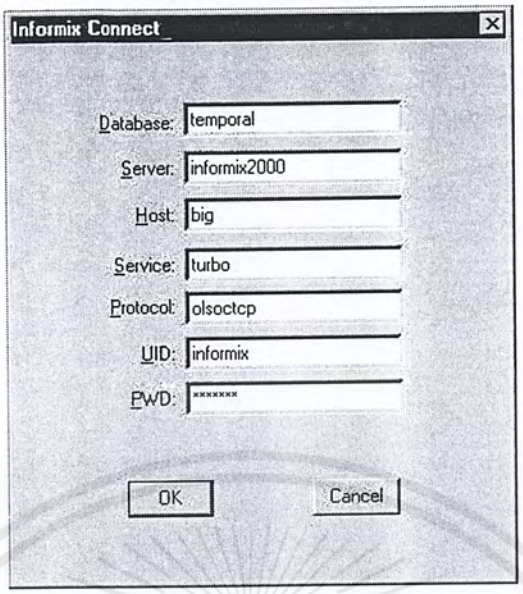
รูปที่ ง-7 แสดงการเข้าสู่การทดสอบ Data Director Connect

2.2 เลือก Machine Data Source ชื่อ student ตามที่ได้สร้างไว้ จะได้ดังรูปที่ ง-8



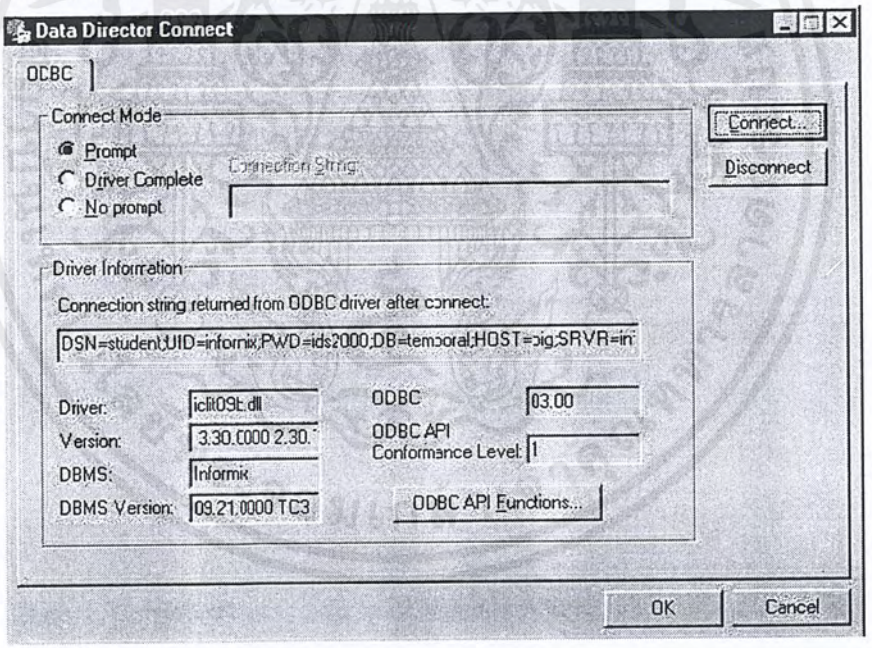
รูปที่ ง-8 แสดงการเลือก Machine Data Source ชื่อ student ตามที่ได้สร้างไว้

2.3 ใส่ UID PWD ที่ได้ตั้งไว้เพื่อที่จะ Connect ได้ดังรูปที่ ง-9



รูปที่ ง-9 แสดงการใส่ UID PWD ที่ได้ตั้งไว้เพื่อที่จะ Connect

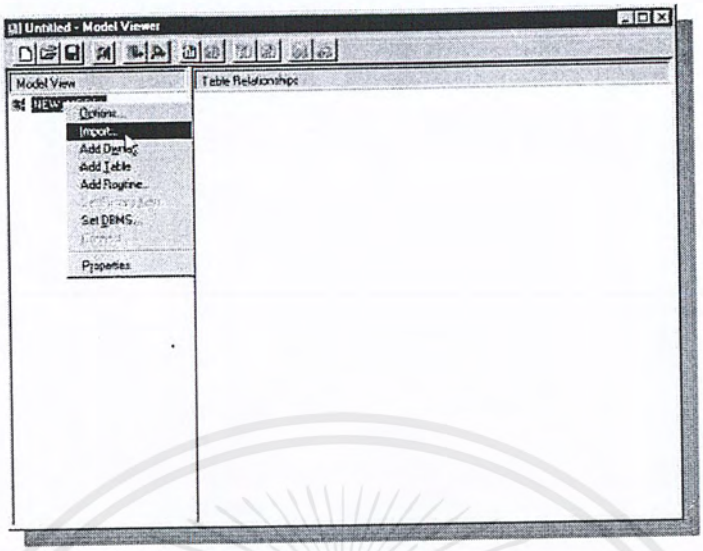
2.4 ถ้าการติดต่อสำเร็จจะได้ดังรูปที่ ง-10



รูปที่ ง-10 แสดงการหลังจากการทดสอบ Connect Data Director Connect สำเร็จ

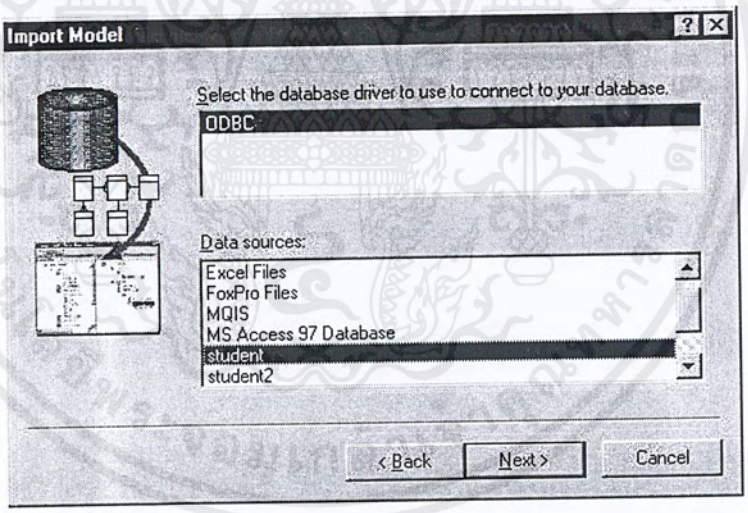
3. **สร้าง Model ของฐานข้อมูล โดยใช้ Data Director**

3.1 เปิดโปรแกรม Visual Basic -> View -> Model Viewer แล้วคลิกเมาส์ขวาจะได้ดังรูปที่ ง-11



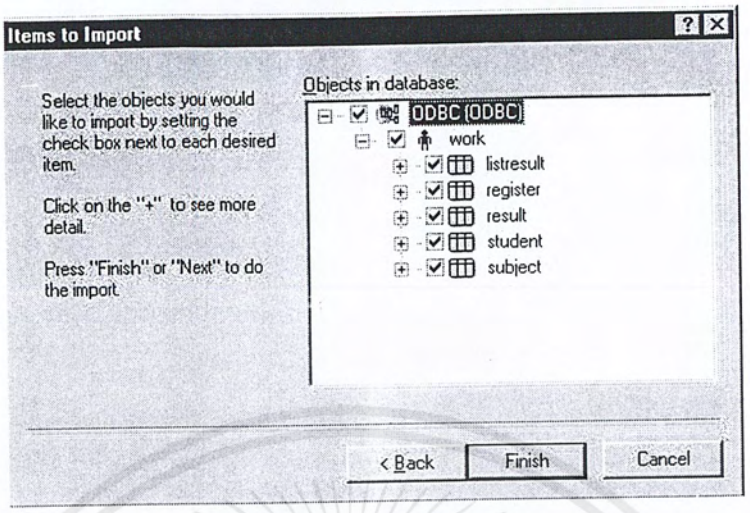
รูปที่ ง-11 Model Viewer

3.2 เลือก Import เพื่อดึงเอาตารางในฐานข้อมูลเข้ามาใน Visual Basic จะปรากฏดังรูปที่ ง-12



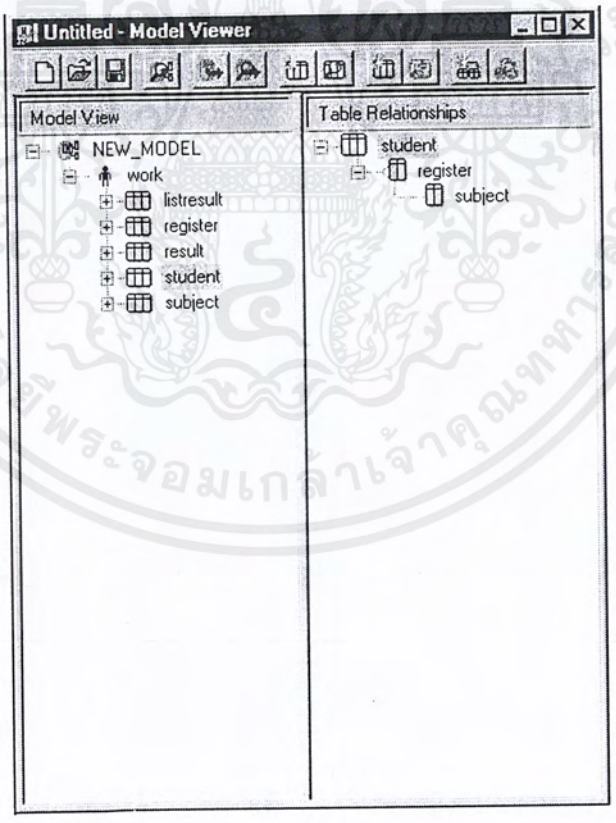
รูปที่ ง-12 Import Model

3.3 เลือก Data Source ที่ต้องการจะใช้ แล้วคลิก Finish จะได้ดังรูปที่ ง-13



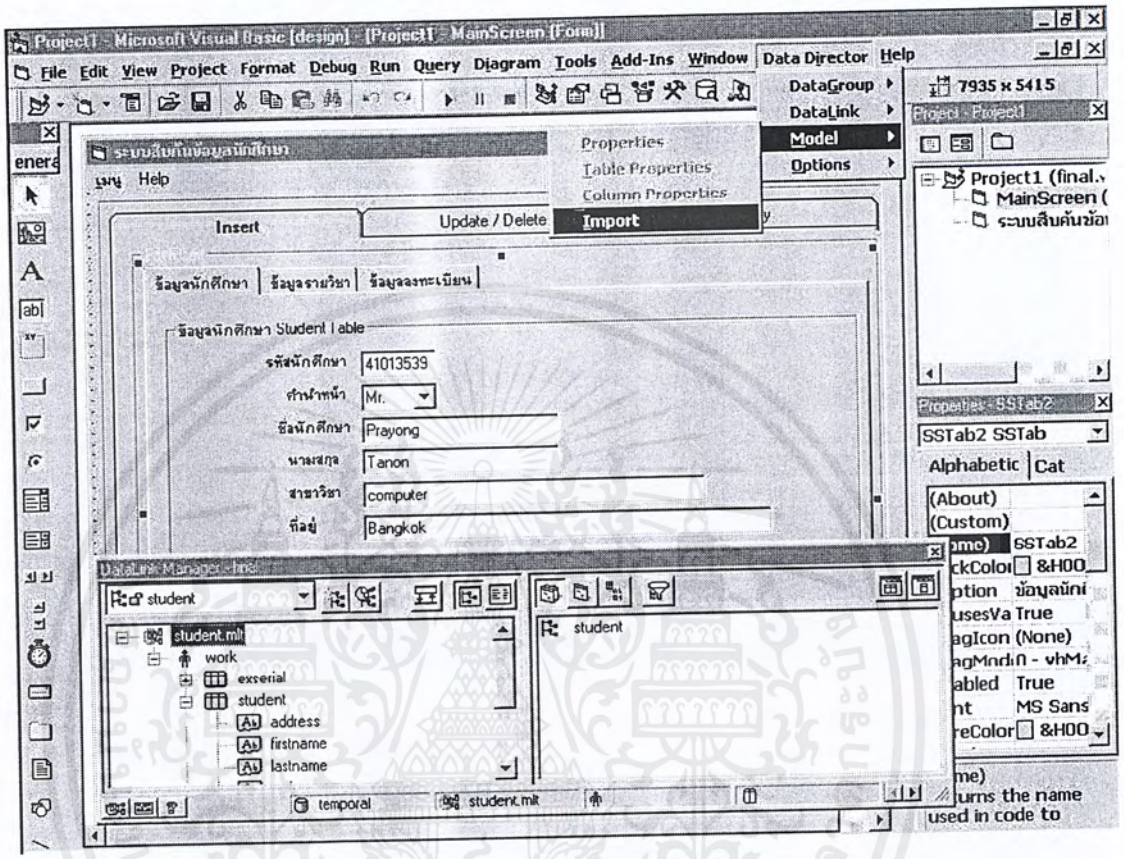
รูปที่ ง-13 Items to Import

3.4 ตารางแล้วคลิก Finish จะปรากฏดังรูปที่ ง-14



รูปที่ ง-14 Verify Inferred Relationships

3.5 ขณะนี้เราได้นำรูปแบบของตารางต่างๆ ในฐานข้อมูลเข้ามาใน Visual Basic แล้ว ในการติดต่อกับฐานข้อมูลเราจะติดต่อผ่าน Model นี้โดยที่เราไม่ต้องติดต่อกับฐานข้อมูลจริง Model จะติดต่อกับฐานข้อมูลให้เราเอง



รูปที่ ง-15 แสดงการ Import Model ใน Visual Basic 6

ภาคผนวก จ

Code ของภาษา SPL ที่ Run ฝังไว้ที่ Server

1. สร้าง Database และ ตารางๆ ที่ต้องใช้ได้ดังนี้

```
CREATE DATABASE   temporal
CREATE TABLE STUDENT
(       Std_serial      Serial(100)  Primary key ,
        StdID           List(Row( StdID_at   CHAR(8)  ,Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL),
        PreName         CHAR(5),
        FirstName       List(Row( FirstName_at   CHAR(30)  ,Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL),
        LastName        List(Row( LastName_at   CHAR(30)  ,Vtime_Start   DATE ,Vtime_End
                        DATE) NOT NULL),
        Major           List(Row( Major_at       CHAR(50)  ,Vtime_Start  DATE ,Vtime_End
                        DATE) NOT NULL ),
        Address         List(Row( Address_at     CHAR(70)  ,Vtime_Start  DATE ,Vtime_End
                        DATE) NOT NULL)
);
CREATE TABLE  result(Name_at char(30),vts date,vte date);
CREATE TAB ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

CREATE TABLE SUBJECT
(       Sub_serial      Serial(200)  Primary key ,
        SubID           List(Row(SubID_at       CHAR(8),    Vtime_Start    DATE  ,Vtime_End
                        DATE) NOT NULL),
        SubName         List(Row(SubName_at  CHAR(40),   Vtime_Start    DATE  ,Vtime_End
                        DATE) NOT NULL),
```

```
        Credit          List(Row(Credit_at      CHAR(1),    Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL),
        Teacher         List(Row(Teacher_at     CHAR(40),  Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL),
        Times           List(Row(Times_at       CHAR(40),   Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL),
        Books           List(Row(Books_at       CHAR(50),   Vtime_Start    DATE ,Vtime_End
                        DATE) NOT NULL)
);
CREATE TABLE REGISTER
(       Regis_serial    Serial (300),
        Std_serial      Integer ,
        Sub_serial      Integer ,
        Term            Integer,
        Year            Integer,
        Grade           List(Row(Grade_at       CHAR(2),   Vtime_Start    DATE,Vtime_End
                        DATE) NOT NULL),
        Section         List(Row(Section_at     CHAR(8),   Vtime_Start    DATE,Vtime_End
                        DATE) NOT NULL),
        Primary key(Regis_serial,Std_serial,Sub_serial,Term,Year)
);
```

2. CODE ของ SPL ที่เป็น PROCEDURE และ FUNTION ได้ดังนี้

```
Create Procedure Findname(x_serial  integer);  --x_serial ='100'
define  name char(20);
define  n smallint;
define thelist  list(row(name1 varchar(30),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(30),vts2 date ,vte2 date);
        drop table result;
        create table result(First_name char(20) );
        SELECT Firstname INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
            select * into therow from table(thelist)
            if  (therow.vte2 is Null ) then
                LET name=therow.name2;
                insert into  result values (name);
            Exit Foreach;
            End if;
        End Foreach
END PROCEDURE;

Create Function  CurrentStudent(x_serial  integer)  --x_serial ='100'
RETURNING  varchar(8) ,varchar(5) ,varchar(30) ,varchar(30) ,varchar(50),varchar(70);
DEFINE Output_StdID  varchar(8);
DEFINE Output_Prename varchar(5);
DEFINE Output_Firstname varchar(30);
DEFINE Output_Lastname varchar(30);
DEFINE Output_Major varchar(50);
DEFINE Output_Address varchar(70);
```

```
define  name char(30);
define  n smallint;
define check  integer;
define data  varchar(70);
define thelist1  collection;
define thelist2  collection;
define therow1    row(name1 varchar(70),vts1 date ,vte1 date);
define therow2    row(name2 varchar(70),vts2 date ,vte2 date);
Drop table result;
Create table result(StdID varchar(8),Prename varchar(5),First_name char(20),Output_Lastname varchar
(20),Output_Major varchar(20),Output_Address varchar(40));
--StdID
        SELECT StdID INTO thelist1  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_StdID=therow1.name1;
                        LET check=1;
                else  LET data=therow1.name1;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_StdID=data;
        End if;
--Prename
        LET Output_Prename=(SELECT Prename   FROM  Student
                        WHERE std_serial = x_serial);
```

```
--Firstname
        SELECT Firstname INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Firstname=therow2.name2;
                        LET check=1;
                else  LET data=therow2.name2;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_Firstname=data;
        End if;
--Lastname
SELECT Lastname INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Lastname=therow2.name2;
                        LET check=1;
                else  LET data=therow2.name2;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
```

```
        LET Output_Lastname=data;
        End if;
--Major
SELECT Major  INTO thelist1  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Major=therow1.name1;
                        LET check=1;
                else  LET data=therow1.name1;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_Major=data;
        End if;
--Address
        SELECT Address INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Address=therow2.name2;
                        LET check=1;
                else  LET data=therow2.name2;
                Exit Foreach;
                End if;
        End Foreach
```

```
        IF  check=0 then
        LET Output_Address=data;
        End if;
insert into result values(Output_StdID,Output_Prename,Output_Firstname ,Output_Lastname,Output_Major
,Output_Address);
RETURN Output_StdID,Output_Prename,Output_Firstname ,Output_Lastname,Output_Major
,Output_Address with resume;
END FUNCTION;

Create Function  CurrentStudentQuery(x_serial  integer)  --x_serial ='100'
RETURNING  varchar(8) ,varchar(5) ,varchar(30) ,varchar(30) ,varchar(50),varchar(70);
DEFINE Output_StdID  varchar(8);
DEFINE Output_Prename varchar(5);
DEFINE Output_Firstname varchar(30);
DEFINE Output_Lastname varchar(30);
DEFINE Output_Major varchar(50);
DEFINE Output_Address varchar(70);

define  name char(30);
define  n smallint;
define check  integer;
define data  varchar(70);
define thelist1  collection;
define thelist2  collection;
define therow1    row(name1 varchar(70),vts1 date ,vte1 date);
define therow2    row(name2 varchar(70),vts2 date ,vte2 date);
Drop table result;
Create table result(StdID varchar(8),Prename varchar(5),First_name char(20),Output_Lastname varchar
(20),Output_Major varchar(20),Output_Address varchar(40));
--StdID
        SELECT StdID INTO thelist1  FROM Student
```

```
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_StdID=therow1.name1;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach

--Prename
        LET Output_Prename=(SELECT Prename   FROM  Student
                        WHERE std_serial = x_serial);


--Firstname
        SELECT Firstname INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Firstname=therow2.name2;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach


--Lastname
```

```
SELECT Lastname INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Lastname=therow2.name2;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Major
SELECT Major  INTO thelist1  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Major=therow1.name1;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Address
        SELECT Address INTO thelist2  FROM Student
        WHERE std_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Address=therow2.name2;
```

```
                    LET check=1;
                Exit Foreach;
                End if;
        End Foreach
insert into result values(Output_StdID,Output_Prename,Output_Firstname ,Output_Lastname,Output_Major
,Output_Address);
RETURN Output_StdID,Output_Prename,Output_Firstname ,Output_Lastname,Output_Major
,Output_Address with resume;
END FUNCTION;

Create Function  CurrentSubject(x_serial  integer)   --x_serial ='100'
RETURNING  varchar(8) ,varchar(40) ,varchar(1) ,varchar(40) ,varchar(40),varchar(50);
DEFINE Output_SubID  varchar(8);
DEFINE Output_Subname varchar(40);
DEFINE Output_Credit  varchar(1);
DEFINE Output_Teacher  varchar(40);
DEFINE Output_Times  varchar(40);
DEFINE Output_Books varchar(50);
define check  integer;
define data  varchar(70);
define  name char(30);
define  n smallint;
define thelist1  collection;
define thelist2  collection;
define therow1    row(name1 varchar(50),vts1 date ,vte1 date);
define therow2    row(name2 varchar(50),vts2 date ,vte2 date);
Drop table result;
Create table result(SubID varchar(8),Subname varchar(40),Credit varchar(1),Teacher varchar(40),Times
varchar(40),Books varchar(50));

--SubID
```

```
SELECT SubID INTO thelist2  FROM Subject
WHERE sub_serial = x_serial;
LET check = 0;
Foreach cursor1 for
        select * into therow2 from table(thelist2)
        if  (therow2.vte2 is Null ) then
                LET Output_SubID=therow2.name2;
                LET check=1;
        else  LET data=therow2.name2;
        Exit Foreach;
        End if;
End Foreach
IF  check=0 then
LET Output_SubID=data;
End if;
--Subname
SELECT Subname INTO thelist1  FROM Subject
WHERE sub_serial = x_serial;
LET check = 0;
Foreach cursor1 for
        select * into therow1 from table(thelist1)
        if  (therow1.vte1 is Null ) then
                LET Output_Subname=therow1.name1;
                LET check=1;
        else  LET Output_Subname=therow1.name1;
        Exit Foreach;
        End if;
End Foreach
IF  check=0 then
LET Output_Subname=data;
End if;
```

```
--Credit
SELECT Credit INTO thelist1  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1  from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Credit=therow1.name1;
                        LET check=1;
                else  LET data=therow1.name1;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_Credit=data;
        End if;
 --Teacher
 SELECT Teacher INTO thelist2  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Teacher=therow2.name2;
                        LET check=1;
                else  LET data=therow2.name2;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_Teacher=data;
```

```
        End if;
--Times
SELECT Times INTO thelist2  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Times=therow2.name2;
                        LET check=1;
                else  LET data=therow2.name2;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
        LET Output_Times=data;
        End if;
 --Books
 SELECT Books INTO thelist1  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Books=therow1.name1;
                        LET check=1;
                else  LET data=therow1.name1;
                Exit Foreach;
                End if;
        End Foreach
        IF  check=0 then
```

```
        LET Output_Books=data;
        End if;
Insert into result values(Output_SubID,Output_Subname,Output_Credit,Output_Teacher,Output_Times
,Output_Books);
RETURN Output_SubID,Output_Subname,Output_Credit ,Output_Teacher,Output_Times ,Output_Books
with resume;
END FUNCTION;

Create Function  CurrentSubjectQuery(x_serial  integer)  --x_serial ='100'
RETURNING  varchar(8) ,varchar(40) ,varchar(1) ,varchar(40) ,varchar(40),varchar(50);
DEFINE Output_SubID  varchar(8);
DEFINE Output_Subname varchar(40);
DEFINE Output_Credit  varchar(1);
DEFINE Output_Teacher  varchar(40);
DEFINE Output_Times  varchar(40);
DEFINE Output_Books varchar(50);
define check  integer;
define data  varchar(70);
define  name char(30);
define  n smallint;
define thelist1  collection;
define thelist2  collection;
define therow1    row(name1 varchar(50),vts1 date ,vte1 date);
define therow2    row(name2 varchar(50),vts2 date ,vte2 date);
Drop table result;
Create table result(SubID varchar(8),Subname varchar(40),Credit varchar(1),Teacher varchar(40),Times
varchar(40),Books varchar(50));
--SubID
        SELECT SubID INTO thelist2  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
```

```
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_SubID=therow2.name2;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Subname
        SELECT Subname INTO thelist1  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1 from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Subname=therow1.name1;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Credit
SELECT Credit INTO thelist1  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow1  from table(thelist1)
                if  (therow1.vte1 is Null ) then
                        LET Output_Credit=therow1.name1;
                        LET check=1;
                Exit Foreach;
                End if;
```

```
        End Foreach

--Teacher
SELECT Teacher INTO thelist2  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Teacher=therow2.name2;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Times
SELECT Times INTO thelist2  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
                select * into therow2 from table(thelist2)
                if  (therow2.vte2 is Null ) then
                        LET Output_Times=therow2.name2;
                        LET check=1;
                Exit Foreach;
                End if;
        End Foreach
--Books
SELECT Books INTO thelist1  FROM Subject
        WHERE sub_serial = x_serial;
        LET check = 0;
        Foreach cursor1 for
```

```
            select * into therow1 from table(thelist1)
            if  (therow1.vte1 is Null ) then
                    LET Output_Books=therow1.name1;
                    LET check=1;
            Exit Foreach;
            End if;
        End Foreach
Insert into result values(Output_SubID,Output_Subname,Output_Credit,Output_Teacher,Output_Times
,Output_Books);
RETURN Output_SubID,Output_Subname,Output_Credit ,Output_Teacher,Output_Times ,Output_Books
with resume;
END FUNCTION;

Create Procedure UpdateStdID(data1 varchar(8) ,ts date ,tp date,x_serial  varchar(5));
define  n smallint;
define thelist  list(row(name1 varchar(8),vts1 date ,vte1 date) not null);
define therow    row(name2 varchar(8),vts2 date ,vte2 date);
        SELECT StdID  INTO thelist  FROM Student
            WHERE std_serial = x_serial;

        SELECT CARDINALITY(StdID)  INTO n  FROM  student
            WHERE  std_serial =x_serial;
        FOREACH cursor1 FOR
            SELECT * INTO therow  FROM TABLE(thelist)
            IF (therow.vte2 is NULL) THEN
                LET  therow.vte2 =ts;
                UPDATE TABLE(thelist)(x)  SET   x= therow
                        WHERE CURRENT OF cursor1;
                EXIT FOREACH;
            END IF;
        END FOREACH
```

```
        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Student SET  StdID= thelist
                WHERE  std_serial=x_serial;
END PROCEDURE;

Create Procedure UpdateFirstname(data1 varchar(30) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(30),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(30),vts2 date ,vte2 date);
        SELECT Firstname  INTO thelist  FROM Student
                WHERE std_serial = x_serial;

        SELECT CARDINALITY(Firstname)  INTO n  FROM  student
                WHERE  std_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Student SET  Firstname= thelist
                WHERE  std_serial=x_serial;
END PROCEDURE;
```

```
Create Procedure UpdateLastname(data1 varchar(30) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(30),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(30),vts2 date ,vte2 date);
        SELECT Lastname INTO thelist  FROM Student
                WHERE std_serial = x_serial;

        SELECT CARDINALITY(Lastname)  INTO n  FROM  student
                WHERE  std_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Student SET  Lastname= thelist
                WHERE  std_serial=x_serial;
END PROCEDURE;

--Major
Create Procedure UpdateMajor(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
```

```
        SELECT Major INTO thelist  FROM Student
                WHERE std_serial = x_serial;

        SELECT CARDINALITY(Major)  INTO n  FROM  student
                WHERE  std_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Student SET Major= thelist
                WHERE  std_serial=x_serial;
END PROCEDURE;

--Address
Create Procedure UpdateAddress(data1 varchar(50) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(50),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(50),vts2 date ,vte2 date);
        SELECT Address INTO thelist  FROM Student
                WHERE std_serial = x_serial;

        SELECT CARDINALITY(Address)  INTO n  FROM  student
                WHERE  std_serial =x_serial;
```

```
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Student SET Address= thelist
                WHERE  std_serial=x_serial;
END PROCEDURE;

Create Procedure UpdateSubID(data1 varchar(30) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(30),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(30),vts2 date ,vte2 date);
        SELECT SubID  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

        SELECT CARDINALITY(SubID)  INTO n  FROM Subject
                WHERE  sub_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
```

```
                    EXIT FOREACH;
                END IF;
            END FOREACH

            LET n=n+1;
    Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
    UPDATE Subject SET SubID= thelist
                WHERE  sub_serial=x_serial;
    END PROCEDURE;

    Create Procedure UpdateSubname(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
    define  n smallint;
    define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
    define therow     row(name2 varchar(40),vts2 date ,vte2 date);
            SELECT Subname  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

            SELECT CARDINALITY(Subname)  INTO n  FROM Subject
                WHERE  sub_serial =x_serial;
            FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                    LET  therow.vte2 =ts;
                    UPDATE TABLE(thelist)(x)  SET   x= therow
                        WHERE CURRENT OF cursor1;
                    EXIT FOREACH;
                END IF;
            END FOREACH

            LET n=n+1;
     Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
```

```
UPDATE Subject SET Subname= thelist
                WHERE  sub_serial=x_serial;
END PROCEDURE;

Create Procedure UpdateCredit(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
        SELECT Credit  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

        SELECT CARDINALITY(Credit)  INTO n  FROM Subject
                WHERE  sub_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Subject SET Credit= thelist
                WHERE  sub_serial=x_serial;
END PROCEDURE;
```

```
- Teacher
Create Procedure UpdateTeacher(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
        SELECT Teacher  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

        SELECT CARDINALITY(Teacher)  INTO n  FROM Subject
                WHERE  sub_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Subject SET Teacher= thelist
                WHERE  sub_serial=x_serial;
END PROCEDURE;

-- Times
Create Procedure UpdateTimes(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
```

```
        SELECT Times  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

        SELECT CARDINALITY(Times)  INTO n  FROM Subject
                WHERE  sub_serial =x_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Subject SET Times= thelist
                WHERE  sub_serial=x_serial;
END PROCEDURE;


-- Books
Create Procedure UpdateBooks(data1 varchar(40) ,ts date ,tp date,x_serial  integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
        SELECT Books  INTO thelist  FROM Subject
                WHERE sub_serial = x_serial;

        SELECT CARDINALITY(Books)  INTO n  FROM Subject
```

```
            WHERE  sub_serial =x_serial;
        FOREACH cursor1 FOR
            SELECT * INTO therow  FROM TABLE(thelist)
            IF (therow.vte2 is NULL) THEN
                LET  therow.vte2 =ts;
                UPDATE TABLE(thelist)(x)  SET   x= therow
                    WHERE CURRENT OF cursor1;
                EXIT FOREACH;
            END IF;
        END FOREACH

        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Subject SET Books= thelist
            WHERE  sub_serial=x_serial;
END PROCEDURE;

Create Function  GetStdID(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);

Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT StdID  INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
            select * into therow from table(thelist)
```

```
                Insert into Listresult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from Listresult);

RETURN  n;
END FUNCTION ;

Create Function  GetFirstname(x_serial  integer)  --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Firstname  INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
            select * into therow from table(thelist)
              Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetLastname(x_serial  integer)  --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
```

```
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Lastname  INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetMajor(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Major   INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
```

```
END FUNCTION ;

Create Function  GetAddress(x_serial  integer)  --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Address   INTO thelist  FROM Student
        WHERE std_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetSubID(x_serial  integer)  --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT SubID   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;
```

```
        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetSubname(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Subname   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetCredit(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
```

```
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Credit   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetTeacher(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Teacher   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
```

```
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetTimes(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);

        SELECT Times   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Function  GetBooks(x_serial  integer)   --x_serial ='100'
RETURNING int;
DEFINE n  int;
define thelist  collection;
define therow    row(at  varchar(70),vts date ,vte date);
Drop table ListResult;
Create table ListResult(data varchar(70),Vtime_Start date,Vtime_End date);
```

```
        SELECT Books   INTO thelist  FROM Subject
        WHERE sub_serial = x_serial;

        Foreach cursor1 for
                select * into therow from table(thelist)
                Insert into ListResult values(therow.at,therow.vts,therow.vte);
        End Foreach
LET n=(select count(*) from ListResult);
RETURN n ;
END FUNCTION ;

Create Procedure DeleteStudent(x_serial integer,vte  date);
define  n smallint;
define thelist1  collection;
define thelist2  collection;
define thelist3  collection;
define thelist4  collection;
define thelist5  collection;
define therow1    row(name1 varchar(50),vts1 date ,vte1 date);
define therow2    row(name2 varchar(50),vts2 date ,vte2 date);
define therow3    row(name3 varchar(50),vts3 date ,vte3 date);
define therow4    row(name4 varchar(50),vts4 date ,vte4 date);
define therow5    row(name5 varchar(50),vts5 date ,vte5 date);
--stdID
        SELECT StdID  INTO thelist1  FROM Student
                WHERE std_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow1  FROM TABLE(thelist1)
                IF (therow1.vte1 is NULL) THEN
                        LET  therow1.vte1 =vte;
```

```
                    UPDATE TABLE(thelist1)(x)  SET   x= therow1
                            WHERE CURRENT OF cursor1;
                    EXIT FOREACH;
                END IF;
            END FOREACH

UPDATE Student SET  StdID= thelist1
                WHERE  std_serial=x_serial;

--Firstname
SELECT Firstname  INTO thelist2  FROM Student
                WHERE std_serial = x_serial;

            FOREACH cursor1 FOR
                SELECT * INTO therow2  FROM TABLE(thelist2)
                IF (therow2.vte2 is NULL) THEN
                    LET  therow2.vte2 =vte;
                    UPDATE TABLE(thelist2)(x)  SET   x= therow2
                            WHERE CURRENT OF cursor1;
                    EXIT FOREACH;
                END IF;
            END FOREACH

UPDATE Student SET Firstname = thelist2
                WHERE  std_serial=x_serial;

--Lastname
SELECT Lastname  INTO thelist3  FROM Student
                WHERE std_serial = x_serial;

            FOREACH cursor1 FOR
```

```
            SELECT * INTO therow3  FROM TABLE(thelist3)
            IF (therow3.vte3 is NULL) THEN
                  LET  therow3.vte3 =vte;
                  UPDATE TABLE(thelist3)(x)  SET   x= therow3
                        WHERE CURRENT OF cursor1;
                  EXIT FOREACH;
            END IF;
      END FOREACH

UPDATE Student SET Lastname = thelist3
            WHERE  std_serial=x_serial;


--Major
SELECT Major  INTO thelist4  FROM Student
            WHERE std_serial = x_serial;

      FOREACH cursor1 FOR
            SELECT * INTO therow4  FROM TABLE(thelist4)
            IF (therow4.vte4  is NULL) THEN
                  LET  therow4.vte4 =vte;
                  UPDATE TABLE(thelist4)(x)  SET   x= therow4
                        WHERE CURRENT OF cursor1;
                  EXIT FOREACH;
            END IF;
      END FOREACH


UPDATE Student SET Major = thelist4
            WHERE  std_serial=x_serial;
```

```
--Address
SELECT Address  INTO thelist5  FROM Student
                WHERE std_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow5  FROM TABLE(thelist5)
                IF (therow5.vte5  is NULL) THEN
                        LET  therow5.vte5 =vte;
                        UPDATE TABLE(thelist5)(x)  SET   x= therow5
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

UPDATE Student SET Address = thelist5
                WHERE  std_serial=x_serial;

END PROCEDURE;

Create Procedure DeleteSubject(x_serial integer,vte  date);
define  n smallint;
define thelist1  collection;
define thelist2  collection;
define thelist3  collection;
define thelist4  collection;
define thelist5  collection;
define therow1    row(name1 varchar(50),vts1 date ,vte1 date);
define therow2    row(name2 varchar(50),vts2 date ,vte2 date);
define therow3    row(name3 varchar(50),vts3 date ,vte3 date);
define therow4    row(name4 varchar(50),vts4 date ,vte4 date);
define therow5    row(name5 varchar(50),vts5 date ,vte5 date);
```

```
--subID
        SELECT SubID  INTO thelist1  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow1  FROM TABLE(thelist1)
                IF (therow1.vte1 is NULL) THEN
                        LET  therow1.vte1 =vte;
                        UPDATE TABLE(thelist1)(x)  SET   x= therow1
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

UPDATE Subject SET  SubID= thelist1
                WHERE  sub_serial=x_serial;

--Subname
SELECT Subname  INTO thelist2  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow2  FROM TABLE(thelist2)
                IF (therow2.vte2 is NULL) THEN
                        LET  therow2.vte2 =vte;
                        UPDATE TABLE(thelist2)(x)  SET   x= therow2
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH
```

```
UPDATE Subject SET Subname = thelist2
                WHERE  sub_serial=x_serial;

--Credit
SELECT Credit  INTO thelist3  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow3  FROM TABLE(thelist3)
                IF (therow3.vte3 is NULL) THEN
                        LET  therow3.vte3 =vte;
                        UPDATE TABLE(thelist3)(x)  SET   x= therow3
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH

UPDATE Subject  SET Credit = thelist3
                WHERE  sub_serial=x_serial;


--Teacher
SELECT Teacher  INTO thelist4  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow4  FROM TABLE(thelist4)
                IF (therow4.vte4  is NULL) THEN
                        LET  therow4.vte4 =vte;
                        UPDATE TABLE(thelist4)(x)  SET   x= therow4
                                WHERE CURRENT OF cursor1;
```

```
                    EXIT FOREACH;
                END IF;
            END FOREACH

UPDATE Subject  SET Teacher = thelist4
                WHERE  sub_serial=x_serial;


--Times
SELECT Times  INTO thelist5  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow5  FROM TABLE(thelist5)
                IF (therow5.vte5  is NULL) THEN
                    LET  therow5.vte5 =vte;
                    UPDATE TABLE(thelist5)(x)  SET   x= therow5
                        WHERE CURRENT OF cursor1;
                    EXIT FOREACH;
                END IF;
        END FOREACH

UPDATE Subject  SET Times = thelist5
                WHERE  sub_serial=x_serial;


--Books
SELECT Books  INTO thelist5  FROM Subject
                WHERE sub_serial = x_serial;

        FOREACH cursor1 FOR
                SELECT * INTO therow5  FROM TABLE(thelist5)
```

```
            IF (therow5.vte5 is NULL) THEN
                LET therow5.vte5 =vte;
                UPDATE TABLE(thelist5)(x) SET x= therow5
                    WHERE CURRENT OF cursor1;
                EXIT FOREACH;
            END IF;
        END FOREACH

UPDATE Subject SET Books = thelist5
            WHERE sub_serial=x_serial;

END PROCEDURE;

Create Procedure UpdateGrade(data1 varchar(40) ,ts date ,tp date,xstd_serial integer,xsub_serial
integer,term_serial integer,Year_serial integer);
define n smallint;
define thelist list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow   row(name2 varchar(40),vts2 date ,vte2 date);
        SELECT Grade INTO thelist FROM Register
            WHERE Std_serial =xstd_serial and sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;

        SELECT CARDINALITY(Grade) INTO n FROM Register
            WHERE Std_serial =xstd_serial and sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;
        FOREACH cursor1 FOR
            SELECT * INTO therow FROM TABLE(thelist)
            IF (therow.vte2 is NULL) THEN
                LET therow.vte2 =ts;
                UPDATE TABLE(thelist)(x) SET x= therow
                    WHERE CURRENT OF cursor1;
```

```
                    EXIT FOREACH;
                END IF;
        END FOREACH
        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Register SET Grade= thelist
                WHERE Std_serial =xstd_serial and  sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;
END PROCEDURE;

Create Procedure UpdateSection(data1 varchar(40) ,ts date ,tp date,xstd_serial  integer,xsub_serial
integer,term_serial  integer,Year_serial integer);
define  n smallint;
define thelist  list(row(name1 varchar(40),vts1 date ,vte1 date) not null);
define therow     row(name2 varchar(40),vts2 date ,vte2 date);
        SELECT Section  INTO thelist  FROM Register
                WHERE Std_serial =xstd_serial and  sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;

        SELECT CARDINALITY(Section)  INTO n  FROM Register
                WHERE  Std_serial =xstd_serial and  sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;
        FOREACH cursor1 FOR
                SELECT * INTO therow  FROM TABLE(thelist)
                IF (therow.vte2 is NULL) THEN
                        LET  therow.vte2 =ts;
                        UPDATE TABLE(thelist)(x)  SET   x= therow
                                WHERE CURRENT OF cursor1;
                        EXIT FOREACH;
                END IF;
        END FOREACH
```

```
        LET n=n+1;
Insert AT n  INTO TABLE(thelist) Values(Row(data1,ts,Null ) );
UPDATE Register SET Section= thelist
                WHERE Std_serial =xstd_serial and  sub_serial = xsub_serial and Term=term_serial and
Year = Year_serial;
END PROCEDURE;
```

บรรณานุกรม

[1] Andreas Steiner's Publications PhD Thesis : *"A Generalisation Approach to Temporal Data Models and their Implementations"*, Departement Informatik, ETH Zurich, November 1997

[2] Cheristian S. Jensen, Richard T. Snodgrass, and Michael D. Soo : "*Extending Existing Dependency Theory to Temporal Databases*"

[3] Date C.J. (1986) : "*An introduction to Database System Vol. 1, 4 Edition.*", Addison Wesley, Massachusatts, 1986.

[4] Iqbal A. Goralwalla, Abdullah U. Tansel and M. Tamer Ozsu : "*Experimenting with Temporal Relational Databases*", Laboratory for Database Systems Research, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

[5] Informix Inc., "Informix Manual"

[6] Jef Wijsen, Vrije Universiteit Brussel : "Temporal FDs on Complex Objects"

[7] กิตติ ภักดีวัฒนะกุล และ จำลอง ครูอุตสาหะ: "*Visual Basic 6 ฉบับโปรแกรมเมอร์*",พิมพ์ครั้ง 2, เคทีพี คอมพ์ แอนด์ คอนซัลท์,กรุงเทพฯ 1999

[8] เกียรติณรงค์ ทองประเสริฐ และ ปรัชญา ศรีเสาวชาติ, "ระบบฐานข้อมูลมัลติมีเดีย" ; ปริญญานิพนธ์, 2542

[9] วุฒิชัย รัฐปฐมวงศ์ และ ศะรัณ เที่ยงสกุล, "ฐานข้อมูลเชิงเวลา" ; ปริญญานิพนธ์, 2542

[10] พิมพ์ผกา อังสวานนท์ และ ภุชงค์ ตั้งเจตน์, "การจัดเก็บวงจรดิจิตอลโดยใช้ฐานข้อมูลเชิงวัตถุสัมพันธ์" ; ปริญญานิพนธ์, 2541

[11] สัจจะ จรัสรุ่งรวีวร : "*คู่มือการสร้างแอพพลิเคชันด้วย Visual Basic 6 Basic & Advanced*",พิมพ์ครั้ง 2 , อินโฟเพรส, กรุงเทพฯ 1999