ความก้าวหน้าของฐานข้อมูลแบบกระจาย

Advances in Distributed Databases

นายธีศิษฎ์ เอียดทองใส
นายนรา มีอุปการ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

ความก้าวหน้าของฐานข้อมูลแบบกระจาย

Advances in Distributed Databases

โดย

นายธีศิษฎ์ เอียดทองใส รหัสประจำตัว 46010327

นายนรา มีอุปการ รหัสประจำตัว 46010340

อาจารย์ที่ปรึกษา

รศ.ดร.ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ความก้าวหน้าของฐานข้อมูลแบบกระจาย

Advances in Distributed Databases

ผู้จัดทำ

1. นายธีศิษฎ์ เอียดทองใส รหัสนักศึกษา 46010327
2. นายนรา มีอุปการ รหัสนักศึกษา 46010340

อาจารย์ที่ปรึกษา

(รศ.ดร.ศุภมิตร จิตตะยโศธร)

ความก้าวหน้าของฐานข้อมูลแบบกระจาย

นายธีศิษฏ์ เอียดทองใส 46010327
นายนรา มีอุปการ 46010340
รศ.ดร. ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอข้อมูลที่ได้ทำการศึกษาเกี่ยวกับแนวคิดและสถาปัตยกรรมของฐานข้อมูลแบบกระจาย ความสามารถและขีดจำกัดของซอฟท์แวร์ Oracle 10g ในการทำระบบฐานข้อมูลแบบกระจาย โดยได้เปรียบเทียบกับระดับสถาปัตยกรรมของระบบฐานข้อมูลแบบกระจาย จากที่ได้ทำการศึกษาพบว่าซอฟท์แวร์ Oracle 10g มีส่วนที่สนับสนุนการทำระบบฐานข้อมูลแบบกระจายในบางระดับสถาปัตยกรรมอย่างเต็มที่ แต่ก็ยังไม่ใช่ในระดับสถาปัตยกรรมสูงสุดซึ่งก็คือระดับ Global schema

โครงงานนี้จึงได้ศึกษาและพัฒนาวิธีการเพื่อเพิ่มประสิทธิภาพในการทำฐานข้อมูลแบบกระจายของ Oracle 10g ให้มีความสามารถใกล้เคียงกับแนวความคิดของฐานข้อมูลแบบกระจายมากขึ้น ทั้งนี้ยังได้พัฒนาซอฟท์แวร์เพื่อช่วยในการสร้างคำสั่งที่จำเป็นในการเพิ่มความสามารถ เพื่อให้ผู้ใช้งานสามารถทำฐานข้อมูลแบบกระจายได้ง่ายขึ้น

Advances in Distributed Databases

Teesit Eadthongsai 46010327

Nara Meeauppakarn 46010340

Assoc.Prof.Dr. Suphamit Chittayasothorn Advisor

Acedemic Year 2006

ABSTRACT

This thesis proposes the comparison between the current implemented distributed database system, which used in general, and the distributed database system concept – defining such as the architecture of the distributed system. Focusing on analyzing how much the implemented system reaches the concept. And then trying to solve the problem and limitation of the database management system by finding out the method to increase its capability; the system, used in this thesis, is implemented by Oracle. Moreover, for convenience, an application was developed according to the invented method. This application gives users the easier way to increase the capability of the implemented distributed database system.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้อย่างดี ด้วยคำแนะนำ และคำปรึกษาจาก รศ.ดร.ศุภมิตร จิตตะยโศธร ซึ่งเป็นอาจารย์ผู้ควบคุมปริญญานิพนธ์ ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์จากท่านอาจารย์ และขอขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกเรื่องๆ ทำให้ข้าพเจ้าสามารถทำปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากปริญญานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นายธีศิษฎ์ เอียดทองใส

นายนรา มีอุปการ

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

ในอดีตที่ผ่านมา จากการที่เครื่องคอมพิวเตอร์ยังมีประสิทธิภาพไม่มากนัก ได้เกิดแนวความคิดในการกระจายการทำงาน เพื่อเพิ่มประสิทธิภาพการทำงานเดิมที่มีอยู่ ซึ่งคือการใช้คอมพิวเตอร์หลายๆเครื่องช่วยกันทำงานงานหนึ่งให้สำเร็จ การทำฐานข้อมูลแบบกระจายก็เป็นหนึ่งในความคิดที่จะกระจายการเข้าถึงข้อมูล เพื่อให้ผู้ใช้งานสามารถเข้าถึงข้อมูลที่อยู่ต่างสถานที่กันได้ โดยผ่านระบบเครือข่าย และเพิ่มความน่าเชื่อถือของระบบฐานข้อมูลในแนวความคิดที่ว่า มีโอกาสน้อยมากที่คอมพิวเตอร์ฐานข้อมูลที่เชื่อมต่อกันทั้งหมดจะไม่สามารถทำงานได้พร้อมกัน

และในปัจจุบันข้อมูลถือเป็นสิ่งที่มีความสำคัญมากขึ้น เห็นได้จากการที่บริษัทต่างๆให้ความสำคัญกับการหาข้อมูล เก็บข้อมูลและรักษาข้อมูลที่มีอยู่ ทำให้เทคโนโลยีการเก็บข้อมูลได้พัฒนาไปอย่างรวดเร็ว มีแนวความคิดใหม่ๆออกมาอย่างต่อเนื่อง และสิ่งที่พัฒนามาพร้อมๆกับเทคโนโลยีการเก็บข้อมูลนั้นคือระบบเน็ตเวิร์ค ที่ในปัจจุบันนั้นมีการเชื่อมต่อเครือข่ายที่มีความน่าเชื่อถือมากขึ้น แนวความคิดของการทำฐานข้อมูลแบบกระจายจึงมีความเป็นไปได้มากขึ้น เนื่องจากมีระบบเน็ตเวิร์คที่มีความเร็วรวมทั้งประสิทธิภาพของการทำงานของคอมพิวเตอร์สูงขึ้นมาก และจะเห็นได้ว่าผู้ผลิตซอฟท์แวร์ฐานข้อมูลเริ่มพัฒนาให้ฐานข้อมูลของตนมีส่วนการรองกับการทำฐานข้อมูลแบบกระจายมากขึ้น แต่ด้วยการที่แนวความคิดของฐานข้อมูลแบบกระจายนั้นยังเป็นแนวความคิดที่ได้มีการนำไปปฏิบัติงานจริงไม่มากนัก โครงงานนี้จึงได้ทำการศึกษาความก้าวหน้าของฐานข้อมูลว่ามีความก้าวหน้าในระดับใด สามารถทำงานตามแนวความคิดฐานข้อมูลแบบกระจายได้สมบูรณ์หรือไม่ โดยโครงงานชิ้นนี้ได้เลือกที่จะศึกษาซอฟท์แวร์ฐานข้อมูลของบริษัท Oracle ซึ่ง Oracle ถือเป็นหนึ่งในผู้ผลิตซอฟท์แวร์ฐานข้อมูลที่มีการยอมรับในประสิทธิภาพการทำงาน ที่สามารถรองรับระบบการทำงานขนาดใหญ่ได้ จึงน่าจะมีเทคโนโลยีการทำงานที่สูงกว่าซอฟท์แวร์อื่นๆ

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 เพื่อศึกษาแนวความคิดและสถาปัตยกรรมของฐานข้อมูลแบบกระจาย

1.2.2 เพื่อศึกษาความก้าวหน้าของซอฟท์แวร์ฐานข้อมูลในปัจจุบันว่าสามารถทำตามแนวความคิดของฐานข้อมูลแบบกระจายได้ถึงระดับใด

1.2.3 เพื่อศึกษาข้อจำกัดของซอฟท์แวร์ฐานข้อมูลในปัจจุบันในการทำฐานข้อมูลแบบกระจาย

1.2.4 เพื่อศึกษาหาวิธีและแนวทางการแก้ไขข้อจำกัดของซอฟท์แวร์ฐานข้อมูลในปัจจุบันเพื่อให้ใกล้เคียงแนวความคิดที่มีอยู่มากขึ้น

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1.3.1 ได้รับความรู้ความเข้าใจในแนวความคิดและข้อกำหนดของการทำฐานข้อมูลแบบกระจาย

1.3.2 สามารถนำแนวความคิดของการทำฐานข้อมูลแบบกระจายไปสร้างฐานข้อมูลแบบกระจายให้สามารถใช้งานได้

1.3.3 ได้รับความรู้ความชำนาญในซอฟท์แวร์ฐานข้อมูลของบริษัท Oracle

1.3.4 ได้วิธีการหรือชุดคำสั่งที่สามารถเพิ่มความสามารถในการทำฐานข้อมูลแบบกระจายของ Oracle

1.4 ขอบเขตของโครงงาน

โครงงานชิ้นนี้ได้ทำการศึกษาแนวความคิดการทำฐานข้อมูลแบบกระจาย ซึ่งสามารถแบ่งเป็นระดับๆได้หลายระดับ ได้แก่ Global Schema, Fragmentation Schema, Allocation Schema, Local Mapping Schema โดยได้ทำการศึกษาว่าแต่ละระดับนั้นมีข้อกำหนดอย่างไรบ้าง และหลังจากศึกษาแนวความคิดการทำฐานข้อมูลแบบกระจายแล้วได้ทำการศึกษาต่อว่าความก้าวหน้าของ DBMS ในปัจจุบันนั้นสนับสนุนแนวความคิดดังกล่าวถึงระดับไหน โดยได้เลือกซอฟท์แวร์ฐานข้อมูลของบริษัท Oracle มาทำการศึกษาความก้าวหน้า นอกจากนั้นในโครงงานชิ้นนี้ยังได้ทำการศึกษาและพัฒนาหาวิธีเพิ่มความสามารถให้กับ Oracle ในการทำระบบฐานข้อมูลแบบกระจาย

บทที่ 2

หลักการของระบบฐานข้อมูลแบบกระจาย

2.1 ความหมายของฐานข้อมูลแบบกระจาย

ระบบฐานข้อมูลแบบกระจาย คือ การเก็บรวบรวมข้อมูลที่มีความสัมพันธ์ในทางตรรกะเข้าไว้เป็นระบบเดียวกัน แต่จะกระจายข้อมูลเก็บตามสถานที่ (Site) ต่างๆ เชื่อมต่อกันอยู่บนระบบเครือข่ายคอมพิวเตอร์ แต่ละสถานที่ สามารถประมวลผลได้อย่างเป็นอิสระ และมีการทำโปรแกรมประยุกต์แบบท้องถิ่น (Local Application) นอกจากนั้นจำเป็นต้องมีการทำงานร่วมกับสถานที่อื่นอย่างน้อย 1 โปรแกรมประยุกต์แบบรวม (Global Application)

หลักในการพิจารณาว่าเป็นระบบฐานแบบกระจายหรือไม่มีดังนี้

- **Autonomous site** คือ แต่ละสถานที่สามารถดูแลจัดการฐานข้อมูลได้ด้วยตนเอง ซึ่งคำว่า Autonomous site จะต้องประกอบไปด้วย Autonomous Computer, Autonomous DBMS, Autonomous Database
- **Interconnected via communication networks** คือ แต่ละสถานที่ต้องมีการเชื่อมต่อผ่านทางเครือข่ายคอมพิวเตอร์
- **Local application** แต่ละสถานที่ต้องมีโปรแกรมประยุกต์ที่เรียกใช้ข้อมูลของสถานที่ที่โปรแกรมทำงานอยู่เท่านั้น
- **Global application** แต่ละสถานที่ต้องมีโปรแกรมประยุกต์ที่มีการเรียกใช้ข้อมูลที่เก็บไว้ต่างสถานที่กัน

ปัจจัยสำคัญที่ใช้ตัดสินว่าระบบใดๆ นั้นควรพัฒนาให้เป็นระบบฐานข้อมูลแบบกระจายหรือว่าระบบฐานข้อมูลแบบรวมดีนั้น ขึ้นอยู่กับปริมาณการใช้งานของโปรแกรมประยุกต์แบบท้องถิ่นยิ่งปริมาณการใช้งานมากยิ่งเหมาะสำหรับระบบฐานข้อมูลแบบกระจาย

โดยหลักการของระบบฐานข้อมูลแบบกระจายในมุมมองของผู้ใช้งานนั้น ผู้ใช้งานไม่ควรจะทราบว่าระบบที่ใช้งานอยู่เป็นระบบฐานข้อมูลแบบกระจาย ซึ่ง C.J. Date ได้กำหนดเป็นกฎ 12 ข้อดังต่อไปนี้

- **Local autonomy**

คือการที่ แต่ละสถานที่ของฐานข้อมูลนั้นสามารถดูแลจัดการได้โดยผู้ดูแลแต่ละสถานที่เอง ไม่ต้องมีศูนย์กลางการควบคุมซึ่งทำให้ผู้ดูแลแต่ละเซิร์ฟเวอร์นั้นสามารถดูแล ปรับปรุง แก้ไขให้สามารถใช้งานได้อย่างมีประสิทธิภาพ

- **No reliance on a central site**

คือ ไม่ต้องมีศูนย์กลางการควบคุมดูแลฐานข้อมูล ทุกสถานที่ของฐานข้อมูลนั้นมีความสำคัญเท่ากันซึ่งการทำ Query optimization, Transaction management, Naming service ไม่มีเซิร์ฟเวอร์ใดที่ทำหน้าที่เป็นศูนย์กลาง ทุกๆเซิร์ฟเวอร์จะต้องทำของแต่ละเซิร์ฟเวอร์เอง

- **Continuous operation**

การทำการกระจายฐานข้อมูลไปในหลายๆสถานที่จะช่วยเพิ่มการทำงานที่ต่อเนื่องของระบบ ซึ่งหากมีส่วนใดส่วนหนึ่งไม่สามารถทำงานได้ ระบบส่วนใหญ่ยังทำงานต่อไปได้ การทำให้ระบบทำงานได้อย่างต่อเนื่องนั้น สามารถทำใช้หลายวิธีร่วมกันได้ เช่น การทำ Replication เพื่อให้มีข้อมูลมากกว่าหนึ่งชุดอยู่ในระบบ หากเซิร์ฟเวอร์ที่เก็บข้อมูลชุดหนึ่งเสียหาย ก็ยังมีข้อมูลอีกชุดที่สามารถเรียกดูได้ ซึ่งการทำงานของ Replication จะกล่าวถึงต่อไปในภายหลัง การมีฐานข้อมูลมากกว่าหนึ่งที่ ก็เป็นการช่วยให้งานสามารถดำเนินการต่อเนื่องได้ โดยมีโอกาสน้อยมากที่ทุกๆ ฐานข้อมูล จะไม่สามารถใช้งานพร้อมกันได้หมด ต่างจากฐานข้อมูลแบบมีศูนย์กลางเพียงที่เดียว ถ้าเครื่องเซิร์ฟเวอร์มีปัญหาอาจจะส่งผลให้ต้องหยุดการให้บริการฐานข้อมูลไปด้วย

- **Location independence**

คือ การที่ผู้ใช้งานสามารถเรียกดูข้อมูลที่ต้องการได้โดยไม่จำเป็นต้องทราบว่าข้อมูลนั้นถูกเก็บไว้ที่ ณ ฐานข้อมูลที่ใด รวมทั้งการย้ายสถานที่เก็บข้อมูลนั้นผู้ใช้งานไม่จำเป็นต้องเปลี่ยนแปลงการเรียกใช้งานด้วย

- **Fragmentation independence**

Fragmentation independence คือ การที่ผู้ใช้งานไม่จำเป็นต้องทราบว่าข้อมูลที่เห็นนั้นในระดับกายภาพแล้วเก็บไว้ที่เดียวกันหรือมีการแบ่งแยกข้อมูลไปเก็บไว้ในหลายๆที่ รวมทั้งไม่จำเป็นต้องทราบว่าแต่ละส่วนของตารางที่ได้แบ่งแยกนั้นเก็บอยู่ที่ใดบ้าง ตัวอย่างเช่น ข้อมูลของพนักงานบริษัทซึ่งข้อมูลของพนังงานทั่วไปถูกเก็บไว้ในเซิร์ฟเวอร์ของสำนักงานในประเทศไทย แต่ในขณะที่พนักงานที่มีตำแหน่งเป็นผู้บริหารจะถูกเก็บอยู่ในเซิร์ฟเวอร์ของบริษัทแม่ที่อยู่ในต่างประเทศ เป็นต้น โดยรายละเอียดของการทำ Fragmentation จะอธิบายเพิ่มเติมต่อไป

- **Replication independence**

Replication คือการที่มีข้อมูลมากกว่าหนึ่งชุดถูกเก็บไว้ในระบบ มีประโยชน์คือ สามารถกระจายการเข้าถึงของข้อมูล ทำให้ผู้ใช้งานเข้าถึงข้อมูลได้หลายทาง รวมทั้งยังเป็นการทำให้ระบบทำงานได้อย่างต่อเนื่องคือ หากเซิร์ฟเวอร์หนึ่งมีปัญหาอาจจะยังมีข้อมูลเดียวกันอยู่อีกที่หนึ่ง ทำให้การดำเนินงานของระบบไม่สะดุด ส่วน Replication independence นั้น คือการที่ผู้ใช้งานไม่จำเป็นต้องรู้ว่าข้อมูลที่ใช้งานอยู่นั้นถูกคัดลอกไปไว้ที่ใดบ้าง การปรับปรุง เปลี่ยนแปลง แก้ไขนั้นสามารถทำได้เสมือนว่าข้อมูลนั้นมีอยู่เพียงชุดเดียวในระบบ

- **Distributed query processing**

มีการวิเคราะห์การค้นหาแนวทางการได้มาซึ่งข้อมูลที่ดีที่สุด โดยในฐานข้อมูลแบบกระจายนั้นจำเป็นที่จะต้องคำนึงถึงการประมวลข้อมูลที่ได้ผลลัพธ์ตามต้องการ เนื่องจากต้องมีการส่งข้อมูลข้ามระบบเครือข่าย และถ้าข้อมูลที่ส่งนั้นเป็นข้อมูลที่มีขนาดใหญ่กว่าที่อยู่ที่ฝั่งรับ จะเป็นการทำให้ใช้เวลาในการส่งมากกว่าการส่งข้อมูลฝั่งน้อยกว่าไปประมวลผลอีกฝั่งหนึ่ง

- **Distributed transaction management (update processing)**

มีการจัดการ Transaction ที่มีการทำงานที่ยุ่งเกี่ยวกับฐานข้อมูลหลายๆสถานที่ต้องมีการทำงานอย่างสมบูรณ์คือ ถ้าสามารถทำงานได้ทุกที่ที่มีการแก้ไขต้องได้รับการแก้ไขอย่างถูกต้องหมด แต่ถ้ามีฐานข้อมูลที่ใดไม่สามารถทำงานได้อย่างถูกต้อง จำเป็นต้องมีการ Roll Back กลับเพื่อให้เสมือนว่าการแก้ไขข้อมูลไม่เคยเกิดขึ้น เป็นการรักษาความถูกต้องของข้อมูล

- **Hardware independence**

โดยปกติแล้ว DBMS ที่ติดตั้งนั้นต้องไม่ขึ้นอยู่กับอุปกรณ์ฮาร์ดแวร์ การเลือกฮาร์ดแวร์สำหรับเครื่องเซิร์ฟเวอร์นั้นจะคำนึงถึงปริมาณข้อมูลและผู้ใช้งานเป็นหลัก เพราะยิ่งมีผู้ใช้งานพร้อมกันเป็นจำนวนมากเครื่องที่ให้บริการก็จำเป็นที่จะต้องมีประสิทธิภาพสูงพอที่จะรองรับได้ ไม่เช่นนั้นอาจจะเกิดปัญหาในการบริการได้

- **Operating system independence**

คือการเป็นอิสระจากระบบปฏิบัติการ ซึ่งในปัจจุบันนั้นผู้ผลิตซอร์ฟแวร์ฐานข้อมูลนั้นส่วนใหญ่จะพัฒนาออกมารองรับให้สามารถใช้งานได้ในหลายๆระบบปฏิบัติการ และการทำฐานข้อมูลแบบกระจายนั้นไม่ควรจะขึ้นอยู่กับระบบปฏิบัติการแบบใดแบบหนึ่งเท่านั้น ควรสามารถทำการเชื่อมต่อกันได้แม้ว่าจะเป็นคนละระบบปฏิบัติการรวมถึงมีฮาร์ดแวร์ที่แตกต่างกันด้วย

- **Network independence**

Network independence เป็นการที่ระบบไม่จำเป็นต้องขึ้นอยู่กับลักษณะของการเชื่อมต่อระหว่างกัน ขอเพียงสามารถส่งข้อมูลระหว่างฐานข้อมูลที่อยู่คนละที่กันได้ก็เพียงพอ ไม่จำเป็นต้องมีการเชื่อมต่อที่เหมือนกัน แต่อย่างไรก็ตามควรมีระบบเน็ตเวิร์คที่รองรับปริมาณการเชื่อมต่อได้ไม่น้อยเกินไปเพื่อป้องกันไม่ให้เกิดปัญหาในการทำงาน

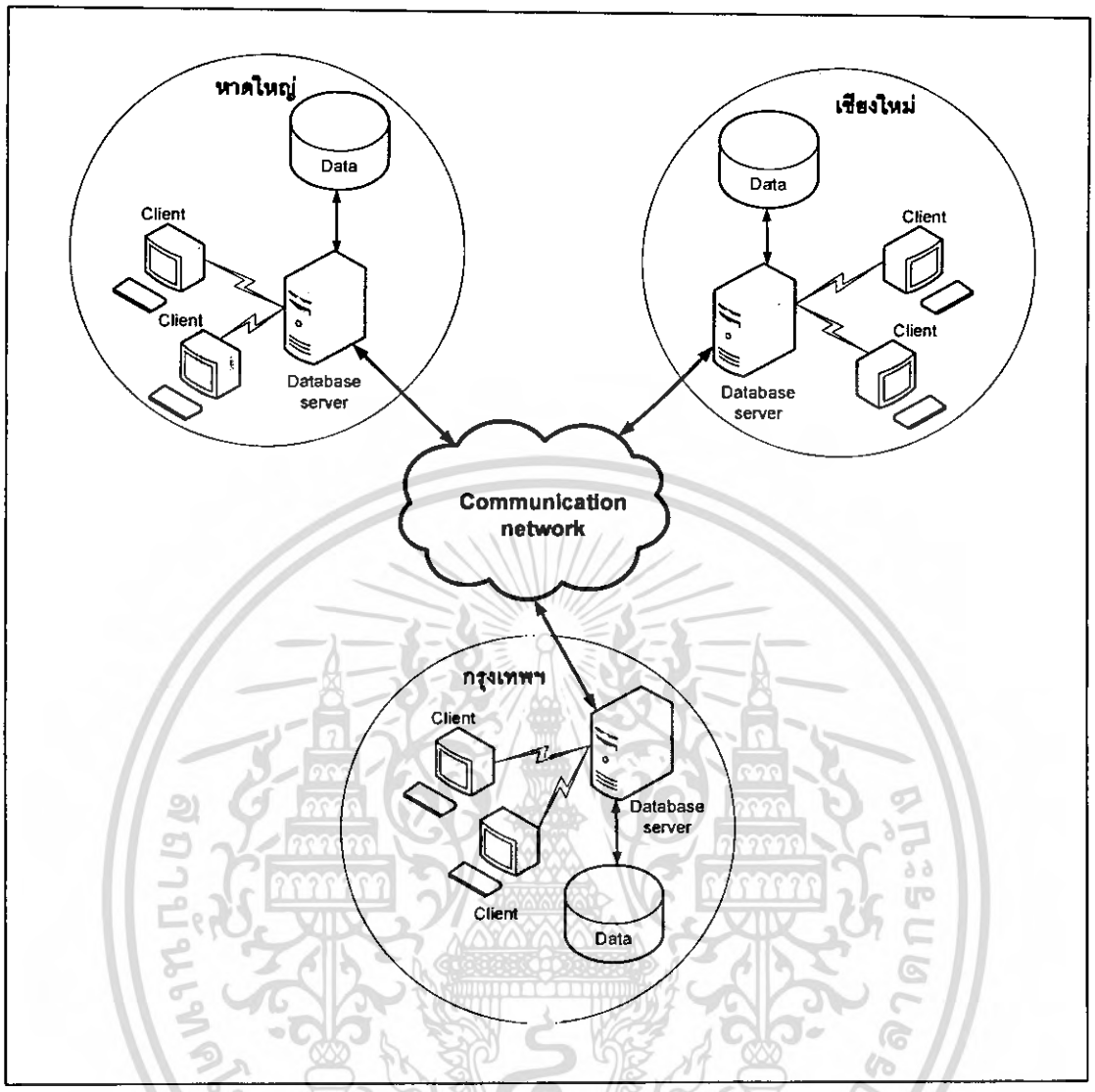
- **DBMS independence**

คือการทำฐานข้อมูลแบบกระจายนั้นไม่ได้จำกัดอยู่เพียงการทำกระจายบนซอร์ฟแวร์ฐานข้อมูลที่ผลิตมาจากผู้ผลิตรายเดียวกัน ควรจะสนับสนุนการทำการกระจายฐานข้อมูลทั้งแบบบนซอร์ฟแวร์ฐานข้อมูลเดียวกัน (Homogeneous) หรือเป็นและแบบใช้หลายๆซอร์ฟแวร์ฐานข้อมูลร่วมกัน (Heterogeneous)

ต่อไปนี้จะเป็นการยกตัวอย่างเพื่อให้เห็นความแตกต่างที่ชัดเจนมากขึ้น ระหว่างระบบฐานข้อมูลแบบกระจาย กับระบบฐานข้อมูลแบบรวม

ตัวอย่างที่ 2.1

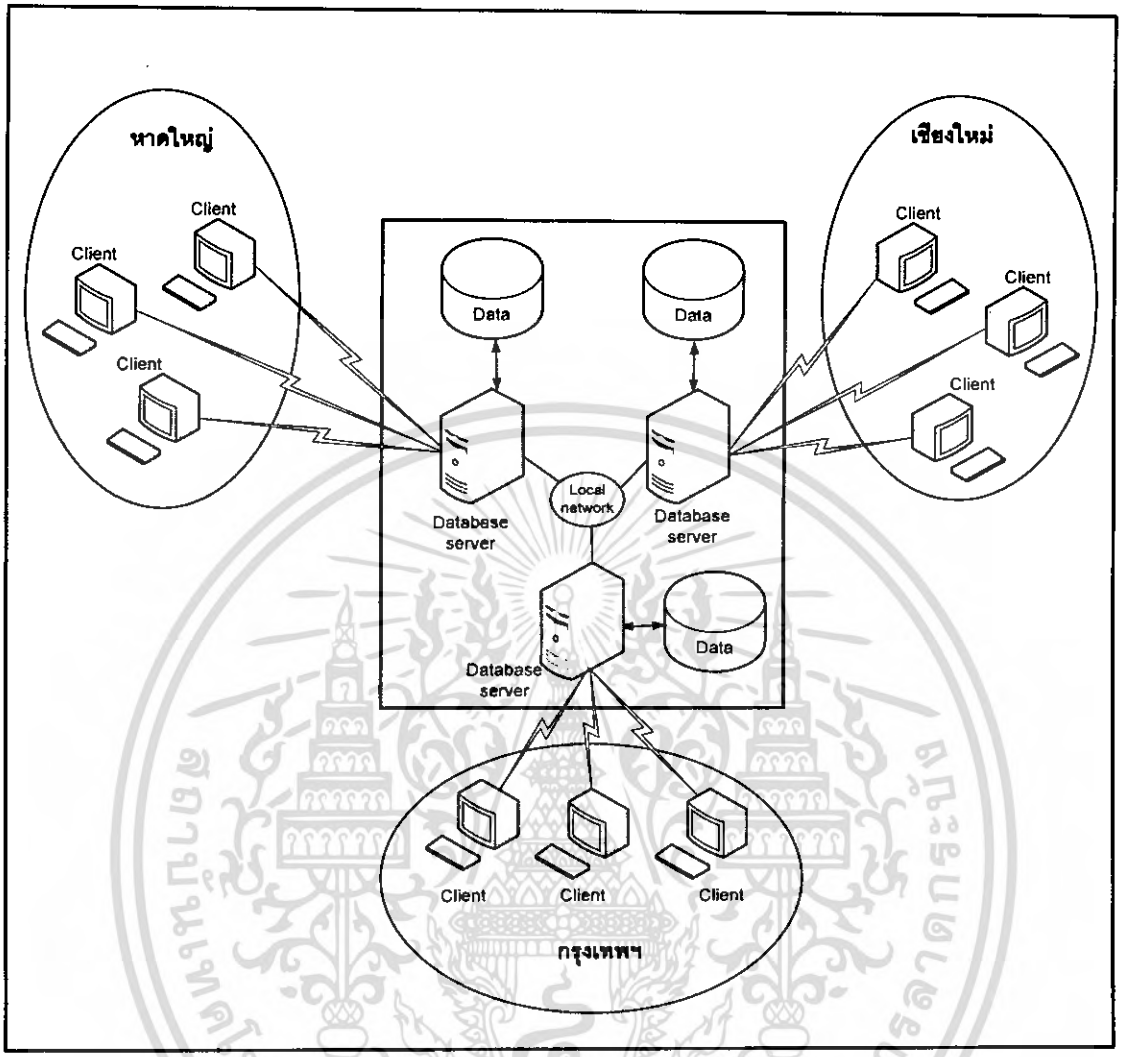
จากรูปที่2.1 แสดงการจำลอง ธนาคาร 3 สาขาที่อยู่ต่างสถานที่กัน แต่ละสาขาจะทำการควบคุมเครื่องไคลเอนต์ และฐานข้อมูลของตัวเอง คอมพิวเตอร์ไคลเอนต์และเซิร์ฟเวอร์ฐานข้อมูลในแต่ละสาขาประกอบกันเรียกว่า "หนึ่งสถานที่" (Site) ของระบบฐานข้อมูลแบบกระจาย คอมพิวเตอร์ในแต่ละสถานที่จะต่อถึงกันด้วยเครือข่ายคอมพิวเตอร์ โดยปรกติแล้ว โปรแกรมประยุกต์จะเรียกใช้ข้อมูลของสาขาที่โปรแกรมทำงานอยู่เท่านั้น เรียกโปรแกรมประยุกต์ชนิดนี้ว่า "โปรแกรมประยุกต์แบบท้องถิ่น" (Local Application) ส่วนโปรแกรมที่มีการเรียกใช้ข้อมูลที่เก็บต่างสถานที่กันนั้นจะเรียกว่า "โปรแกรมประยุกต์แบบรวม" (Global Application) หรือเรียกว่า "โปรแกรมประยุกต์แบบกระจาย" (Distributed Application)



รูปที่ 2.1 แสดงระบบฐานข้อมูลแบบกระจายตามลักษณะภูมิศาสตร์

ตัวอย่างที่ 2 .2

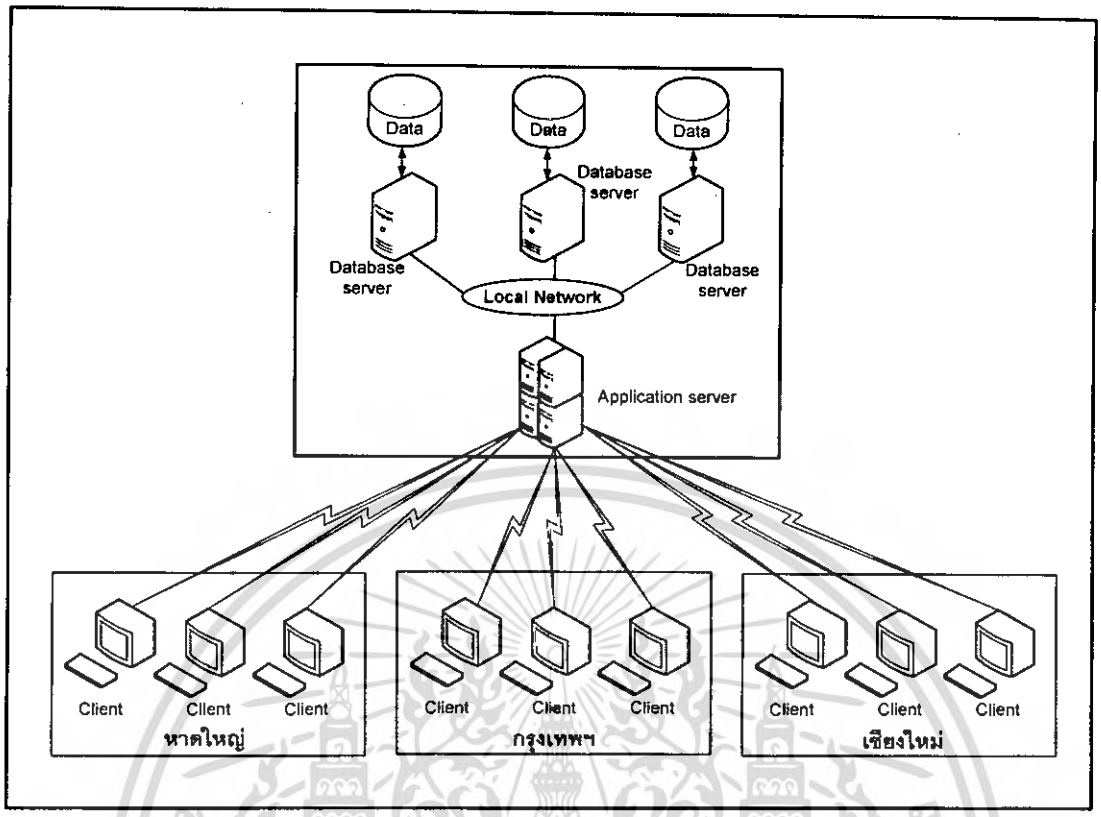
จากกรณีของธนาคารในตัวอย่างที่ 2.1 ถ้าโครงสร้างของระบบฐานข้อมูลถูกเปลี่ยนแปลงให้เป็นดังรูปที่ 2.2 คือย้ายเซิร์ฟเวอร์ของฐานข้อมูลทั้งหมดมาอยู่รวมกันที่ส่วนกลาง เครื่องไคลเอนต์เชื่อมต่อผ่านเครือข่ายคอมพิวเตอร์ จะเห็นได้ว่าเซิร์ฟเวอร์ของฐานข้อมูลและเครื่องไคลเอนต์ รวมกันเรียกว่าหนึ่งสถานที่ (Site) เหมือนเดิม จะเห็นว่าแม้ลักษณะทางกายภาพจะถูกเปลี่ยนแปลงไป แต่สถาปัตยกรรมของระบบ และคุณสมบัติต่างๆ ทางตรรกะยังคงเหมือนเดิม การทำงานของโปรแกรมประยุกต์แบบท้องถิ่น ก็ยังคงประมวลผลที่เครื่องคอมพิวเตอร์สถานที่เดิม รวมทั้งเข้าถึงข้อมูลของสถานที่เดิมอีกด้วย ซึ่งสรุปว่า การแบ่งความเป็นท้องถิ่น ไม่ได้ขึ้นอยู่กับลักษณะทางกายภาพ แต่จะขึ้นอยู่กับสถานที่ (Site) ประมวลผลและข้อมูลที่ใช้



รูปที่ 2.2 แสดงระบบฐานข้อมูลแบบกระจายที่นำเครื่องเซิร์ฟเวอร์มาไว้รวมกันที่ส่วนกลาง

ตัวอย่างที่ 2.3

จากรูปที่ 2.3 เป็นการแสดงโครงสร้างของธนาคาร ทั้ง 3 สาขา ซึ่งโครงสร้างจะเปลี่ยนไปจากตัวอย่างที่ 2.1 และ 2.2 คือ ข้อมูลของธนาคารทั้ง 3 จะเก็บไว้ที่เครื่องเซิร์ฟเวอร์ฐานข้อมูล 3 เครื่อง ซึ่งจะทำหน้าที่จัดการฐานข้อมูล ส่วนโปรแกรมประยุกต์จะประมวลผลบนเครื่องคอมพิวเตอร์เครื่องอื่น ซึ่งจะเรียกใช้ข้อมูลผ่านเซิร์ฟเวอร์ฐานข้อมูล จะเห็นว่าในตัวอย่างนี้ เป็นเพียงระบบฐานข้อมูลแบบรวม ที่ประมวลผลบนตัวประมวลผลหลายตัวเท่านั้น เนื่องจากโปรแกรมประยุกต์ใช้งานของธนาคารทั้ง 3 สาขาประมวลผลบนคอมพิวเตอร์เครื่องเดียวกัน แม้ว่าจะแยกเก็บข้อมูลไว้ต่างที่กันก็ตาม ก็ไม่ได้จัดว่าเป็นระบบฐานข้อมูลแบบกระจาย



รูปที่ 2.3 แสดงระบบประมวลผลที่ไม่ใช่ฐานข้อมูลแบบกระจาย

2.2 ประเภทของ Distributed Database System

จะสามารถแบ่งออกได้เป็น 2 ประเภท ด้วยกันคือ Homogeneous distributed database system และ Heterogeneous distributed database system

Homogeneous distributed database system ทุกสถานที่จะใช้ซอฟท์แวร์ระบบจัดการฐานข้อมูล (DBMS) ชนิดเดียวกัน ซึ่งอาจจะไม่ได้อยู่บนระบบปฏิบัติการเดียวกันก็ได้

Heterogeneous distributed database system มีอย่างน้อย 1 สถานที่ ที่ใช้ซอฟท์แวร์ระบบจัดการฐานข้อมูล (DBMS) ต่างจากสถานที่อื่น ซึ่งหลักในการสร้าง Heterogeneous distributed database system ต้องอาศัยมาตรฐานที่เรียกว่า Gateway protocols ในที่นี้ Gateway protocols จะหมายถึง API (Application Programming Interface) ซึ่งทำหน้าที่เชื่อมต่อระหว่าง DBMS และโปรแกรมต่าง ๆ เช่น ODBCและ JDBC เป็นต้น

2.3 Distributed Database Architecture

สถาปัตยกรรมของฐานข้อมูลแบบกระจายที่แบ่งเป็นระดับต่างๆ แสดงดังรูปนี้ ในทางปฏิบัติแล้วไม่จำเป็นที่จะต้องทำได้ถึงระดับสูงสุดซึ่งก็คือ Global schema แล้วจึงจะถือว่าเป็นระบบฐานข้อมูลแบบกระจาย อาจจะทำได้เพียงในระดับ Allocation schema หรือ Fragmentation schema ก็ได้ และเนื่องจาก Data Model ที่ใช้ในระบบฐานข้อมูลแบบกระจายอาจเป็นได้ทั้ง Relational Model หรือ Object Model เห็นได้จากรูปแสดงสถาปัตยกรรมของระบบฐานข้อมูลแบบกระจาย ซึ่งใน 3 ระดับบนแสดงให้เห็นว่าไม่ขึ้นอยู่กับ Data Model ที่ใช้ แต่โครงงานนี้จะใช้เฉพาะ Relational Model เป็นหลัก เพราะสะดวกในการศึกษาค้นคว้า

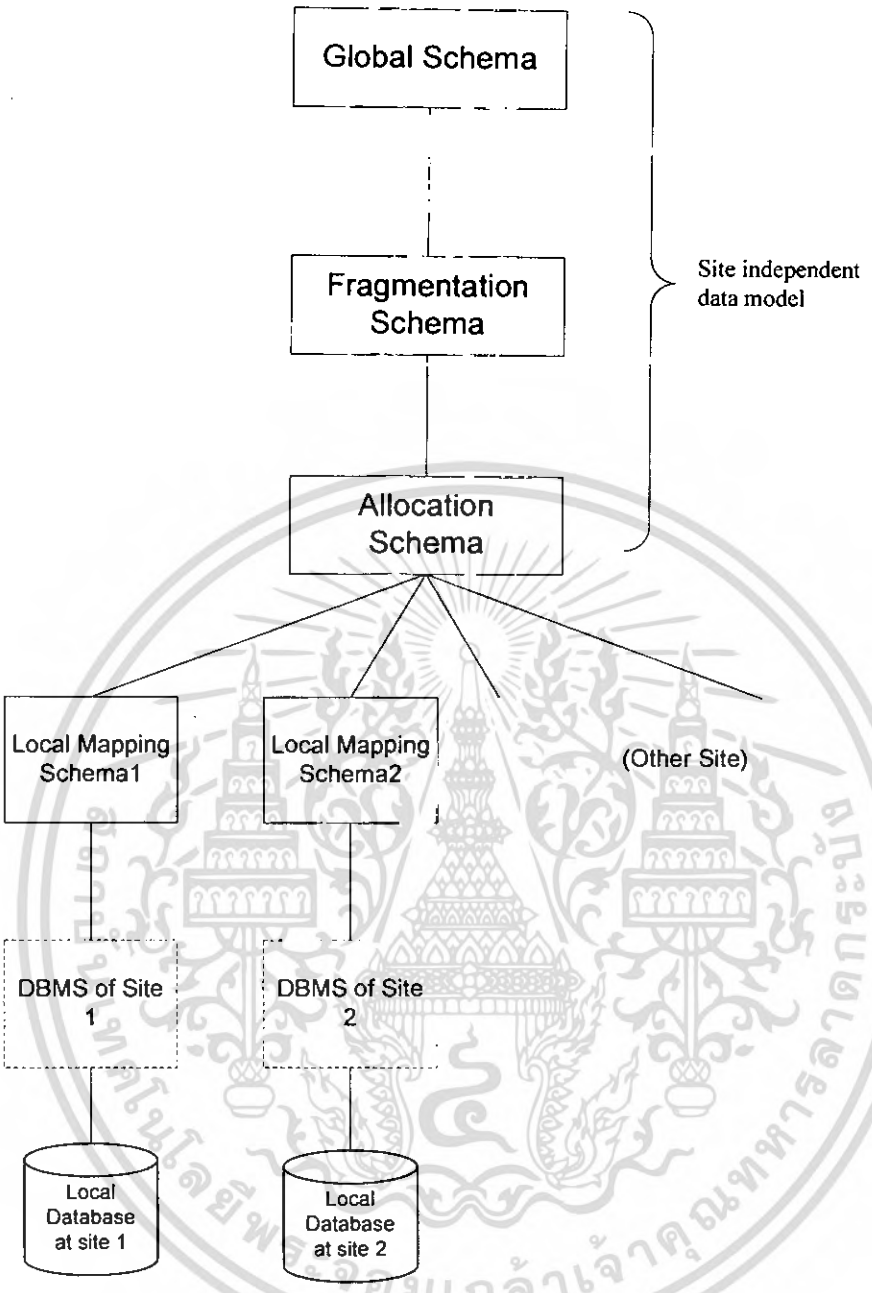
ต่อไปนี้จะเป็นการอธิบายความสามารถของสถาปัตยกรรมในแต่ละระดับ

Global schema

เป็นส่วนที่อยู่บนสุดของสถาปัตยกรรมการอ้างถึงข้อมูลของระบบฐานข้อมูลแบบกระจาย ซึ่ง Global schema จะอ้างถึงข้อมูลที่สถานที่ (Site) ต่างๆ เสมือนกับว่าข้อมูลทั้งหมดไม่มีการกระจายเลย ซึ่งจะเหมือนกับการอ้างถึงข้อมูลในฐานข้อมูลแบบรวม (Centralize database) และ relationที่ถูกอ้างถึงในสถาปัตยกรรมระดับนี้ จะเรียกว่า Global relation

Fragmentation scbema

สถาปัตยกรรมในระดับนี้เป็นการเข้าถึงข้อมูลโดย อ้างถึงบางส่วนของ Global relation ที่ถูกแบ่งออกเป็นส่วนๆ โดยจะเรียกแต่ละส่วนว่า Fragment ซึ่งวิธีการแบ่ง Global relation ออกเป็น fragment นี้ สามารถแบ่งได้หลายวิธี เช่น แบ่งตามแนวนอน (Horizontal Fragmentation) คือแยกทัปเปิล (Tuples) ออกเป็นกลุ่มๆ หรือแบ่งตามแนวตั้ง (Vertical Fragmentation) คือแยกแอทริบิวต์ (Attributes) ออกเป็นกลุ่มๆ และ Fragment ที่แบ่งออกมานี้ สามารถที่จะนำกลับมารวมเข้าเป็นความสัมพันธ์โดยรวมได้ดังเดิม ด้วยการทำ UNION และ JOIN ตามลำดับ อย่างไรก็ตามการทำ Fragmentation นี้ สามารถที่จะทำทั้งสองวิธีผสมกัน (Mixed Fragmentation) ได้ ทั้งนี้จะต้องสามารถนำกลับมารวมเป็นความสัมพันธ์โดยรวมได้ดังเดิม



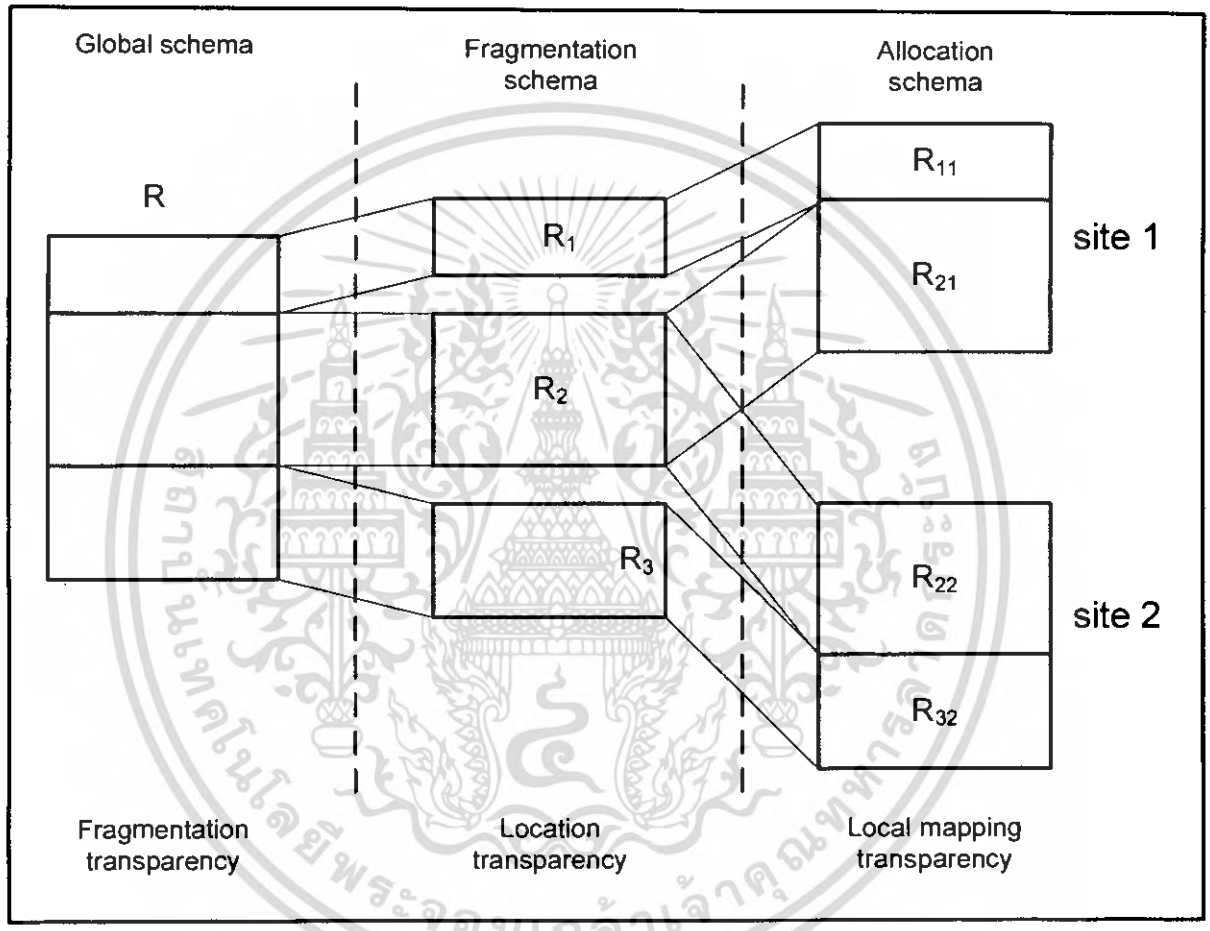
รูปที่ 2.4 แสดงสถาปัตยกรรมระดับต่างๆ ของฐานข้อมูลแบบกระจาย

Allocation schema

สถาปัตยกรรมในระดับนี้จะกล่าวถึง Fragment ที่แบ่งมาจาก Global relation นั้น ว่าจะถูกกำหนดสถานที่ (Site) ที่จะเก็บข้อมูล Fragment ไว้ที่ไหน ซึ่งในแต่ละ Fragment นี้ อาจจะมีสถานที่ (Site) ในเก็บข้อมูลมากกว่าหนึ่งแห่งได้ ทั้งนี้ขึ้นอยู่กับการตัดสินใจของผู้ออกแบบระบบฐานข้อมูล ว่าจะยอมให้มีการซ้ำซ้อนของข้อมูลมากน้อยเพียงใด และสอดคล้องกับการใช้งานหรือไม่

Local mapping schema

ในสถาปัตยกรรมทั้ง 3 ระดับข้างต้นที่ได้กล่าวไปแล้วนั้น ฐานข้อมูลแบบกระจายในแต่ละสถานที่ (Site) จะไม่ขึ้นอยู่กับ Data Model ที่ใช้ เพราะฉะนั้นแล้วในสถาปัตยกรรมในระดับนี้เองจะเป็นระดับที่มีการทำให้ DBMS ในแต่ละสถานที่ (Site) สามารถเข้าใจ Data Model ที่ใช้ในสถาปัตยกรรมระดับที่สูงขึ้นไปได้



รูปที่ 2.5 แสดงการพิจารณาระดับสถาปัตยกรรมและระดับ Transparency

2.4 Distribution Transparency

จากสถาปัตยกรรมของระบบฐานข้อมูลแบบกระจายในระดับต่างๆ ทำให้สามารถกำหนดระดับ Transparency ได้ดังต่อไปนี้

- **Fragmentation transparency** เป็นระดับ Transparency สูงสุด ซึ่งในระดับนี้ ผู้ใช้งานหรือ โปรแกรมเมอร์ จะเห็นในระดับ Global relation คือ ไม่ทราบว่าแบ่งเป็นกี่ Fragment แต่ละ Fragment ถูกเก็บไว้ที่ไหน รวมทั้งมีการทำซ้ำที่ไหนบ้าง

- **Location transparency** เป็นระดับที่ ผู้ใช้งาน หรือ โปรแกรมเมอร์ จะเห็นในระดับ Fragment ทำงานบน Fragment แต่จะไม่ทราบว่า Fragment นั้นถูกเก็บไว้ที่สถานที่ (Site) ไหน
- **Local mapping transparency** เป็นระดับที่ผู้ใช้งาน หรือโปรแกรมเมอร์ จะทราบว่าเป็น Fragment ที่ใช้งานถูกเก็บไว้ที่สถานที่ (Site) ไหน มีการทำซ้ำที่ไหนบ้าง แต่จะไม่ทราบว่า ที่สถานที่ (Site) นั้น ใช้ DBMS ยี่ห้อไหน Data model แบบใด
- **No transparency** เป็นระดับที่ผู้ใช้งานหรือ โปรแกรมเมอร์ จำเป็นต้องทราบว่าในสถานที่ (Site) นั้นใช้ DBMS ยี่ห้อไหน Data model แบบใด

2.5 Distributed Data Storage

ในการจัดเก็บข้อมูลในระบบฐานข้อมูลแบบกระจายมีอยู่หลายวิธีด้วยกันคือ

- **Fragmentation** เป็นการแบ่ง Relation ออกเป็นหลาย ๆ ส่วน และจัดเก็บแต่ละส่วนไว้ต่างสถานที่กัน
- **Replication** เป็นการทำสำเนาของ Relation ไว้หลาย ๆ สำเนา และแต่ละสำเนาจะถูกเก็บไว้ต่างสถานที่กัน

นอกจากนั้นเรายังสามารถทำ Fragmentationร่วมกับ Replication ได้ คือ ทำการแบ่ง Relation ออกเป็นหลาย ๆ ส่วน และแต่ละส่วนก็จะมีการจัดทำสำเนาไว้ด้วย

2.6 รูปแบบในการทำ Fragmentation

การแบ่งความสัมพันธ์โดยรวม (Global relation) ออกเป็น Fragment นี้ สามารถแบ่งออกได้ 2 วิธีหลักๆ ด้วยกันคือ แบ่งตามแนวนอน (Horizontal Fragmentation) คือแยกทัปเปิล (Tuples) ออกเป็นกลุ่มๆ หรือแบ่งตามแนวตั้ง (Vertical Fragmentation) คือแยกแอทริบิวต์ (Attributes) ออกเป็นกลุ่มๆ ซึ่งในการแบ่งความสัมพันธ์โดยรวม (Global relation) ออกเป็น Fragment ต้องเป็นไปตามกฎเกณฑ์ดังนี้

- **Completeness condition** คือ ทุกข้อมูลของความสัมพันธ์โดยรวม (Global relation) ต้องพบได้ใน Fragment ที่ได้แบ่งไว้ด้วย เช่น จะต้องไม่เกิดกรณีที่มีข้อมูลอยู่ในความสัมพันธ์โดยรวมแต่ไม่อยู่ใน Fragment ใดๆ
- **Reconstruction condition** คือ ต้องสามารถสร้างความสัมพันธ์โดยรวม (Global relation) ขึ้นมาได้จาก Fragment ที่ได้แบ่งเอาไว้ เพราะว่าในระบบฐานข้อมูลแบบกระจายนั้น แต่ละสถานที่ (Site) จะเก็บ Fragment ที่ได้แบ่งเอาไว้ ส่วนความสัมพันธ์

โดยรวม (Global relation) จะต้องถูกสร้างขึ้นใหม่เมื่อมีความจำเป็นจาก Fragment ในแต่ละสถานที่ (Site)

- **Disjointness condition** คือ แต่ละ Fragment ต้องแยกออกจากกัน สำหรับเงื่อนไขนี้ถือเป็นหลักการสำหรับกรณี การแบ่งส่วนย่อยตามแนวนอน (Horizontal Fragmentation) ขณะที่กรณีการแบ่งส่วนย่อยตามแนวตั้ง (Vertical Fragmentation) จะยอมให้มีการละเมิดเงื่อนไขในข้อนี้ได้

สามารถอธิบายรายละเอียดเพิ่มเติมในการแบ่งส่วนย่อย (Fragmentation) ประเภทต่างๆ ได้ดังนี้

2.6.1 การแบ่งตามแนวนอน (Horizontal Fragmentation)

คือ การแยกทัปเปิล (Tuples) ออกเป็นกลุ่มๆ ซึ่งแต่ละกลุ่มสามารถแบ่งได้ตามลักษณะทางภูมิศาสตร์ อย่างเช่น ภาค จังหวัด ประเทศ เป็นต้น ซึ่งในการทำ Fragmentation ตามแนวนอนนี้สามารถทำได้โดยการใช้คำสั่งของภาษา Relation algebra คือใช้ select บน Global relation ตัวอย่างเช่น กำหนดให้ Global relation คือ

SUPPLIER (SNUM, NAME, CITY)

สามารถทำการ Fragmentation ตามแนวนอนได้โดยการ select ข้อมูลใน SUPPLIER ตามค่าแอทริบิวต์ของ CITY ซึ่งก็คือค่า "SF" สำหรับ SUPPLIER1 และ "LA" สำหรับ SUPPLIER2 ได้ดังนี้

SUPPLIER1 = $\mathbf{SL}_{CITY = "SF"}$ SUPPIER

เป็นข้อมูลของ SUPPLIER เฉพาะในเมือง San Francisco

SUPPLIER2 = $\mathbf{SL}_{CITY = "LA"}$ SUPPIER

เป็นข้อมูลของ SUPPLIER เฉพาะในเมือง Los Angeles

กรณี Completeness condition จะเป็นจริงได้ถ้าค่าแอทริบิวต์ของ CITY มีแค่ "SF" และ "LA" เท่านั้น

กรณี Reconstruction condition ในการสร้าง SUPPLIER Global relation ขึ้นมาใหม่ สามารถทำได้โดยการใช้ UNION operation ดังนี้

SUPPLIER = SUPPLIER1 **UN** SUPPLIER2

กรณี Disjointness condition จะเห็นได้ว่าข้อมูลจะถูกแยกตาม CITY คือ San Francisco และ Los Angeles เท่านั้น ต้องไม่มีทัปเปิล (Tuples) ไหนที่ค่า CITY เป็นทั้ง San Francisco และ Los Angeles

2.6.2 การแบ่งตามการสืบทอดแนวนอน (Derived Horizontal Fragmentation)

มีบางกรณีที่การแบ่งตามแนวนอน (Horizontal Fragmentation) ไม่สามารถที่จะแบ่งตามแอทริบิวต์ของตัวเองได้ จำเป็นต้องสืบทอดมาจาก Relation อื่น ยกตัวอย่างเช่นมี Relation ดังนี้

SUPPLY (SNUM, PNUM, DEPTNUM, QUAN)

Relation นี้จะต้องทำการแบ่ง Fragment ตามค่า CITY ของ SUPPLIER relation อย่างไรก็ตาม CITY ไม่ได้เป็นแอทริบิวต์ของ SUPPLY relation แต่เป็นแอทริบิวต์ของ SUPPLIER relation ดังนั้นเราจำเป็นต้องใช้ SEMI-JOIN operation ในการบอกว่าทัปเปิล (Tuples) ไหนใน SUPPLY ที่ตรงกับ SUPPLIER ที่ได้แบ่งตาม CITY จะทำให้ Derived fragmentation ของ SUPPLY สามารถกำหนดได้ดังนี้

$$\text{SUPPLY1} = \text{SUPPLY } \mathbf{SJ}_{\text{SNUM = SNUM}} \text{ SUPPLIER1}$$

$$\text{SUPPLY2} = \text{SUPPLY } \mathbf{SJ}_{\text{SNUM = SNUM}} \text{ SUPPLIER2}$$

ซึ่ง SNUM เป็นค่าของ supplier number ถูกใช้เป็นค่าในการทำ SEMI-JOIN ระหว่าง SUPPLY relation กับ SUPPLIER relation

กรณี Completeness condition มีความจำเป็นที่ต้องไม่มีค่า SNUM ค่าใดที่มีอยู่ใน SUPPLY relation แต่ไม่ได้อยู่ใน SUPPLIER relation จึงจะตรงกับเงื่อนไขนี้

กรณี Reconstruction condition การสร้าง Global relation SUPPLY ขึ้นมาใหม่สามารถใช้ UNION operation ได้เหมือนกับการแบ่งตามแนวนอนของ SUPPLIER relation

กรณี Disjointness condition คือ จะต้องไม่มีทัปเปิล(Tuples)ใดใน SUPPLY relation ที่ตรงกับ 2 Fragment ที่ต่างกัน

2.6.3 การแบ่งตามแนวตั้ง (Vertical Fragmentation)

คือการแบ่งแอทริบิวต์ (Attributes) ของ Global relation ออกเป็นกลุ่มๆ อธิบายได้ง่ายๆ ก็คือการแบ่งตามคอลัมน์นั่นเอง มักจะขึ้นอยู่กับ Application ว่าต้องการใช้แอทริบิวต์หรือคอลัมน์ไหนบ้าง แต่ละ Fragment จะมาจากการใช้ Projection operation เลือกเอาคอลัมน์ที่ต้องการ นำมาจัดกลุ่ม ตัวอย่างสามารถพิจารณาจาก Global relation ดังนี้

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

การแบ่ง Fragment ตามแนวตั้งของ relation นี้ คือ

$$EMP1 = \mathbf{PJ}_{EMPNUM,\ NAME,\ MGRNUM,\ DEPTNUM}\ EMP$$

$$EMP2 = \mathbf{PJ}_{EMPNUM,\ NAME,\ SAL,\ TAX}\ EMP$$

กรณี Completeness condition คือค่าของแอทริบิวต์หรือคอลัมน์ของ Global relation จะต้องมีอยู่อย่างน้อยใน 1 Fragment ที่ได้แบ่งเอาไว้

กรณี Reconstruction condition การสร้าง Global relation EMP ขึ้นมาใหม่จะใช้การ JOIN ระหว่าง Fragment ที่ได้แบ่งเอาไว้ ซึ่งตัวอย่างนี้จะใช้ค่า EMPNUM เป็น key ในการ JOIN

$$EMP = EMP1\ \mathbf{JN}_{EMPNUM\ =\ EMPNUM}\ \mathbf{PJ}_{EMPNUM,\ SAL,\ TAX}\ EMP2$$

จะเห็นได้ว่าเมื่อทำการ JOIN กันแล้ว มีค่าแอทริบิวต์หรือว่าคอลัมน์ที่ซ้ำกันซึ่งก็คือ EMPNUM และ NAME สามารถกำจัดค่าที่ซ้ำออกไปโดยใช้ Projection operation จัดกลุ่มเอาเฉพาะค่าที่ต้องการเท่านั้น

กรณี Disjointness condition เพราะว่าจำเป็นที่ต้องมี Key ที่อยู่ในทุกๆ Fragment เพื่อใช้ในการสร้าง Global relation ขึ้นมาใหม่ เพราะฉะนั้นในการทำ Vertical Fragmentation จึงยอมให้มีการละเมิดกฎในข้อนี้ได้

2.6.4 การแบ่งแบบผสม (Mixed Fragmentation)

นอกจากการทำ Fragmentation แบบต่างๆ ที่ได้กล่าวมาแล้ว ยังสามารถนำการทำ Fragmentation แบบต่างๆ มาผสมกันได้ ตัวอย่างเช่นมี Global relation ดังนี้

EMP (EMPNUM, NAME, SAL, TAX, MGRNUM, DEPTNUM)

สามารถทำ Fragmentation แบบผสมได้โดย การแบ่งตามแนวตั้ง (Vertical fragmentation) ตามด้วย แบ่งตามแนวนอน (Horizontal Fragmentation) โดยใช้ค่า DEPTNUM เป็นตัวแบ่ง จะได้

$$EMP1 = SL_{DEPTNUM \leq_{10}} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

$$EMP2 = SL_{10 < DEPTNUM \leq_{20}} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

$$EMP3 = SL_{DEPTNUM > 20} PJ_{EMPNUM, NAME, MGRNUM, DEPTNUM} EMP$$

$$EMP4 = PJ_{EMPNUM, NAME, SAL, TAX} EMP$$

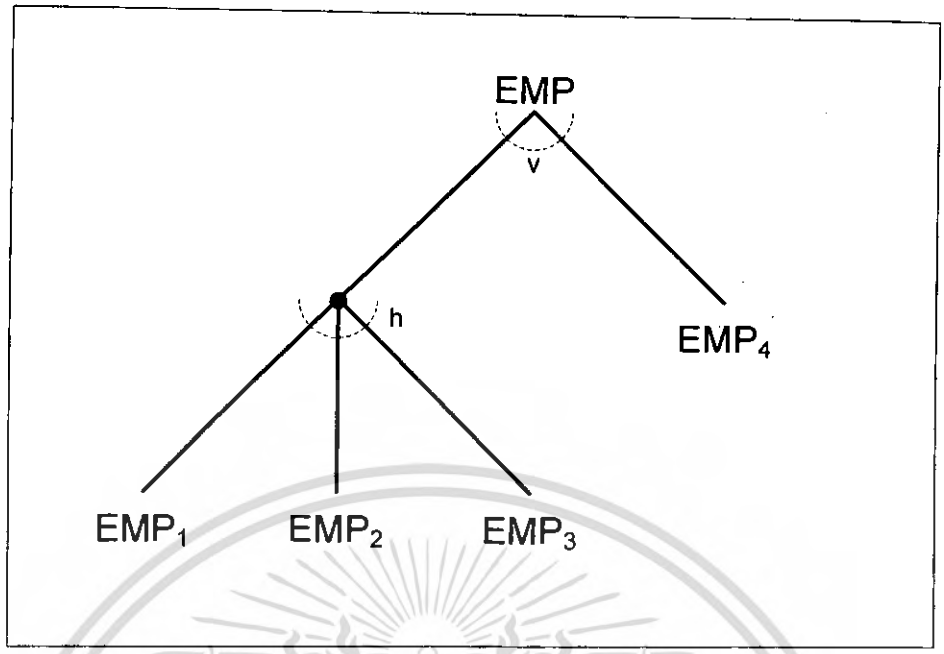
กรณี Completeness condition ใช้การพิจารณาตามหลักของการแบ่งตามแนวนอน (Horizontal fragmentation) และแบ่งตามแนวตั้ง (Vertical fragmentation) ประกอบกัน

กรณี Reconstruction condition สำหรับการสร้างGlobal relation ขึ้นใหม่สามารถทำได้โดย ย้อนกลับขึ้นไปหา Global relation ตัวอย่างนี้จะทำการ UNION fragment ที่แบ่งตามแนวนอนก่อน แล้วจึงทำการ JOIN fragment ที่ทำการแบ่งตามแนวตั้ง แสดงเป็นลำดับคำสั่งได้ดังนี้

$$EMP = UN(EMP1, EMP2, EMP3)\ JN_{EMPNUM = EMPNUM}\ PJ_{EMPNUM, SAL, TAX}\ EMP4$$

กรณี Disjointness condition ก็พิจารณาตามหลักการของของการแบ่งตามแนวนอน (Horizontal fragmentation) และแบ่งตามแนวตั้ง (Vertical fragmentation) ประกอบกัน

การทำ Fragmentation แบบผสมสามารถแสดงได้เป็น Tree ซึ่งจะถูกเรียกว่า Fragmentation tree จากตัวอย่างแสดง Tree ได้ดังนี้



รูปที่ 2.6 แสดง Tree ที่เกิดจากการทำ Fragmentation แบบผสม

2.7 Data Replication

การที่จะบอกว่า Relation r หรือ Fragment ของ Relation r มีการทำ Replication ก็ต่อเมื่อมีการทำสำเนาของ Relation r หรือ Fragment นี้ เก็บไว้ตามสถานที่ต่างๆ โดยที่ไม่จำเป็นจะต้องเก็บไว้ในทุกๆ สถานที่ แต่สำหรับกรณีที่มีการทำสำเนาไว้ที่ทุกสถานที่ จะเรียกว่า Full replication

ซึ่งการทำ Replication นี้มีทั้งข้อดีและข้อเสีย สามารถอธิบายได้ดังต่อไปนี้

- **ข้อดี**

Availability ถ้าสถานที่กำลังทำงานอยู่กับ Relation r เกิดหยุดการทำงานลง ระบบสามารถที่จะทำงานต่อไปได้โดยไปดึงข้อมูลที่สถานที่อื่น โดยไม่ต้องคำนึงถึงสถานที่ ที่หยุดทำงานไป

Increased parallelism ในกรณีที่การดำเนินการกับข้อมูลส่วนใหญ่ของ Relation r เป็นการอ่านข้อมูลจาก Relation เมื่อเรามีข้อมูลของ Relation r อยู่ในหลาย ๆ สถานที่ เราก็สามารถที่จะทำการดึงข้อมูลมาพร้อม กันได้ อีกทั้งจากการที่มีสำเนาของ Relation อยู่หลาย ๆ สถานที่ โอกาสที่จะพบข้อมูลในสถานที่ ที่ต้องการจะทำ Transaction ก็มีโอกาสมากกว่า ดังนั้นด้วยวิธีการนี้จะช่วยลดปริมาณข้อมูลที่จะส่งผ่านระหว่างสถานที่ได้

- **ข้อเสีย**

Increased overhead on update ด้วยวิธีการ Replication นี้ สำเนาข้อมูลของ Relation r ในแต่ละสถานที่จะต้องเหมือนกันทุก ๆ สถานที่ ไม่เช่นนั้นแล้วการประมวลผลกับ Relation r อาจจะเกิดความผิดพลาดขึ้นได้ ดังนั้นเมื่อไรก็ตามที่ Relation r ถูกแก้ไข ระบบจะต้องทำการแก้ไขข้อมูล Relation r ให้ครบทุกสถานที่ที่มี Relation r อยู่ ยกตัวอย่างเช่น ในระบบธนาคารซึ่งมีการทำสำเนาของข้อมูลบัญชีไว้หลาย ๆ สถานที่ เมื่อมีการปรับปรุงยอดบัญชี ก็จะต้องทำการปรับยอดบัญชีให้เท่ากันในทุก ๆ สถานที่

โดยทั่วไปแล้วการทำสำเนาข้อมูลจะช่วยเพิ่มประสิทธิภาพในการอ่านข้อมูล และช่วยเพิ่มความสะดวกในการสืบค้นข้อมูลของ Transaction แบบอ่านอย่างเดียว อย่างไรก็ตาม Transaction แบบที่มีการปรับปรุงข้อมูลจะต้องมีการดำเนินการที่เพิ่มมากขึ้น ซึ่งการควบคุมการปรับปรุงข้อมูลพร้อมๆ กันในหลาย ๆ สถานที่ค่อนข้างจะยุ่งยากกว่าระบบฐานข้อมูลแบบรวม

2.7.1 ชนิดของการทำ Replication

- **Synchronous Replication** คือการที่ข้อมูลที่มีการคัดลอกไว้หลายชุดและกระจายไปอยู่ตามสถานที่ต่างๆนั้น จะเป็นข้อมูลที่มีการแก้ไขหรือเปลี่ยนแปลงพร้อมกันเสมอ คือ เมื่อมีการแก้ไขเปลี่ยนแปลงข้อมูล ณ ที่ใดๆ ต้องมีการแก้ไขข้อมูลชุดอื่นๆด้วย โดยถ้ามีข้อมูลชุดใดชุดหนึ่งไม่สามารถแก้ไขได้ ก็จะถือว่าการแก้ไขไม่สามารถทำได้ การทำ Synchronous Replication นั้นเพื่อเป็นการรักษา Integrity Constraints ซึ่งจะทำให้ไม่ว่าเราดึงข้อมูลมาจากข้อมูลชุดใดๆ จะได้ข้อมูลที่เหมือนกันและข้อมูลที่ใหม่ที่สุดเสมอ แต่อย่างไรก็ตามการทำ Replication แบบนี้นั้น จำเป็นที่จะต้องมีระบบเครือข่ายคอมพิวเตอร์ที่เชื่อถือได้อย่างมาก เพราะว่าการทำแบบนี้เมื่อมีการเพิ่มหรือแก้ไขข้อมูลจำเป็นที่จะต้องแก้ไขข้อมูลทุกชุดเท่าที่มีอยู่ ซึ่งเป็นการเพิ่มภาระหน้าที่ของการทำงานแต่ละงานและทำให้การ COMMIT ช้าลงไปอีกด้วย โดยการทำ Synchronous Replication นั้นจำเป็นที่จะต้องมีการใช้งาน 2-Phrase Commit Protocols (2PC) เข้ามาช่วยในการที่จะ Commit Transaction เพื่อให้สามารถแก้ไขข้อมูลทุกๆที่พร้อมกัน และการทำ Synchronous Replication นั้น ถ้าหากมีเซิร์ฟเวอร์ใดที่เก็บข้อมูลชุดนี้อยู่ด้วย ไม่สามารถทำงานได้นั้น การแก้ไขข้อมูลนั้นไม่สามารถทำได้เลยซึ่งอาจจะเป็นปัญหาได้ ดังนั้นจึงต้องระมัดระวังในการวางแผนการทำ Replication

- **Asynchronous Replication** เป็นการทำ Replication ที่ข้อมูลชุดหลักนั้นมีเพียงชุดเดียว ส่วนข้อมูลที่คัดลอกไปนั้น ทำเพื่อเพิ่มช่องทางการเข้าถึงข้อมูล โดยข้อมูลทั้งสองชุดนั้นอาจไม่เหมือนกันตลอดเวลา เนื่องจากสถานที่ที่เก็บข้อมูลไว้สามารถทำการ Commit โดยอิสระ การทำ

Replication แบบนี้เหมาะกับกรณีข้อมูลไม่ต้องเปลี่ยนแปลงบ่อยนัก แบ่งตามการทำงานได้ 3 ลักษณะด้วยกัน คือ

1 PUSH คือ สถานที่หลัก (Primary site) จะส่ง Transaction ไปยังสถานที่ที่เกี่ยวข้อง (Remote site) โดยที่สถานที่ที่เกี่ยวข้องไม่ต้องทำการร้องขอ อาจจะส่งเป็นข้อมูลที่มีการเปลี่ยนแปลงแล้วก็ได้

2 PULL คือ สถานที่ที่เกี่ยวข้อง (Remote site) จะต้องทำการร้องขอการเปลี่ยนแปลงจากสถานที่หลัก (Primary site) ถ้าไม่มีการร้องขอสถานที่หลักก็ไม่ทำการส่งข้อมูลที่มีการเปลี่ยนแปลง

3 SNAPSHOT คือ สถานที่หลัก (Primary site) จะส่งการเปลี่ยนแปลงไปยังสถานที่ที่เกี่ยวข้อง (Remote site) ตามช่วงเวลาที่ได้กำหนดไว้

2.8 Distributed Transaction

ในการทำงานของ Application ของระบบฐานข้อมูลแบบรวมจะมีการทำ Transaction เป็นส่วนหนึ่งอยู่ด้วยเสมอ เช่นกันในระบบฐานข้อมูลแบบกระจาย เนื่องจากมีการทำงานของ Application ดังนั้นจึงมี Transaction เข้ามาเกี่ยวข้องด้วยเช่นกัน ในระบบฐานข้อมูลแบบกระจายนี้จะมีการทำ Transaction อยู่ 2 ประเภทด้วยกัน คือ

Local transaction จะดำเนินการกับข้อมูลในฐานข้อมูลของตัวเองเท่านั้น

Global transaction จะทำการเข้าถึงข้อมูลและปรับปรุงข้อมูลในหลาย ๆ ฐานข้อมูลตามสถานที่ต่างๆ

สำหรับในกรณีของ Global transaction การรักษาคุณสมบัติ ACID (Atomicity, Consistency, Isolation, Durability) ของการทำ Transaction แบบกระจายจะมีการดำเนินการที่ซับซ้อนมากขึ้น เนื่องจากแต่ละสถานที่จะมีส่วนร่วมในการทำ Transaction ซึ่งความล้มเหลวที่เกิดขึ้นในสถานที่เหล่านี้ หรือความล้มเหลวจากการสื่อสารกันระหว่างสถานที่ อาจทำให้การทำ Transaction ผิดพลาดได้

2.8.1 โครงสร้างของระบบ

ในแต่ละสถานที่จะมีตัวจัดการ Transaction (Local transaction manager) เป็นของตนเอง ซึ่งทำหน้าที่ควบคุมให้การทำ Transactionให้มีคุณสมบัติ ACIDโดยที่ตัวจัดการ Transaction แต่ละตัวก็จะร่วมกันในการทำ Global transaction โดยหลักการแล้ว เมื่อทำการเริ่มต้นการทำงานของ Transaction ณ สถานที่ไหน ตัวจัดการ Transaction ของสถานที่นั้นจะทำหน้าที่เป็นตัวประสานงาน Transaction (Transaction coordinator) ส่วนสถานที่อื่นจะเป็นผู้เกี่ยวข้อง (Participant) อธิบายการทำงานเพิ่มเติมได้ ดังนี้

- **ตัวจัดการ Transaction (Transaction manager)** จะคอยควบคุมดูแลการดำเนินการของ Transaction ต่างๆ ที่มีการเข้าถึงข้อมูลในสถานที่ของตนเอง แต่ละ Transaction เป็นได้ทั้ง Local transaction หรือเป็นส่วนหนึ่งของ Global transaction ก็ได้ อธิบายหน้าที่การทำงานเพิ่มเติมได้ดังนี้

  I. การจัดการกับ Log เพื่อจุดประสงค์ในการฟื้นคืนสภาพข้อมูล (Recovery)

  II. มีส่วนร่วมในการทำ Concurrency control กับสถานที่อื่นๆ เพื่อร่วมทำ Concurrent execution ของ Transaction ในสถานที่ของตนเอง

- **ตัวประสานงาน Transaction (Transaction coordinator)** จะคอยประสานการทำงานของ Transaction ต่าง ๆ ทั้ง Local transaction และ Global transaction อธิบายหน้าที่การทำงานเพิ่มเติมได้ดังนี้

  - เริ่มต้นการทำงานของ Transaction ที่เกี่ยวข้อง
  - แบ่ง Transaction ออกเป็น Transaction ย่อย และส่งไปยังสถานที่ ที่เหมาะสม
  - ประสานการดำเนินการของ Transaction ว่า ผลของการทำ Transaction นั้น ทำสำเร็จในทุก ๆ สถานที่ หรือยกเลิกในทุก ๆ สถานที่หรือไม่ คือ ดูแลการทำ Two-phase commit

2.8.2 Commit Protocols

เพื่อให้การทำ Transaction บนระบบฐานข้อมูลแบบกระจายมีคุณสมบัติ Atomicity คือ Transaction T ต้อง Commit ในทุก ๆ สถานที่ หรือยกเลิกการทำ Transaction ในทุก ๆ สถานที่ Transaction coordinator ของ Transaction T ต้องมีการทำ Commit protocol ซึ่ง Commit protocol ที่มีการใช้งานกันอย่างแพร่หลายก็คือ Two-phase commit protocol (2PC) และ Three-phase commit protocol(3PC) ซึ่งจะช่วยหลีกเลี่ยงข้อเสียของ 2PC แต่การดำเนินการจะมีความซับซ้อนมากกว่า สำหรับในโครงงานนี้จะพูดถึงเฉพาะในกรณีของ Two-phase commit protocol(2PC) เท่านั้น

Two-Phase Commit (2PC)

ให้ T เป็น Transaction เริ่มต้นที่สถานที่ S_i และให้ ตัวประสานงาน transaction ที่สถานที่ S_i เป็น C_i สามารถอธิบายการทำงานเป็น 2 เฟส ดังนี้

Phase 1: Obtaining a decision

ฝั่ง Transaction coordinator

ตัวประสานงาน Transaction Ci จะถาม participants รอบๆ ว่าพร้อมที่จะ commit หรือยังโดย

- C_i ทำการเพิ่มเรคอร์ด <Prepare T> ลงไปใน Log ไฟล์ บน Stable storage
- จากนั้น C_i จะส่งสัญญาณ Prepare T ไปให้กับทุก ๆ สถานที่ ที่ประมวลผล Transaction T

ฝั่ง Participant

เมื่อสถานที่อื่น ๆ ได้รับสัญญาณ ตัวจัดการ Transaction จะต้องตอบกลับไปหาตัวประสานงาน Transaction ว่าจะทำการ Commit transaction ได้หรือไม่

- ถ้าไม่ได้จะเพิ่มเรคอร์ด <No T> ลงไปใน Log ไฟล์และส่งสัญญาณ Abort T กลับไปที่ C_i
- ถ้าได้จะเพิ่มเรคอร์ด <Ready T> ลงไปใน Log ไฟล์ และตัวจัดการ Transaction จะส่งสัญญาณ Ready T กลับไปที่ C_i

Phase 2: Recording the decision

ฝั่ง Transaction coordinator

- C_i จะ Commit transaction T ก็ต่อเมื่อได้รับสัญญาณ Ready T จากทุก ๆ สถานที่ ที่มีส่วนร่วมในการทำ Transaction กรณีนอกเหนือจากนี้ Transaction T จะต้องถูกยกเลิก และจะต้องทำการเพิ่มเรคอร์ด <Commit T> หรือ <Abort T> ลงไปใน Log ไฟล์ ถ้า Transaction T สามารถ Commit ได้ หรือ ทำการยกเลิก Transaction
- ต่อมา C_i ก็จะส่งสัญญาณ Commit T หรือ Abort T ไปให้กับทุกๆ สถานที่ ที่ร่วมกันทำ Transaction

ฝั่ง Participant

- เมื่อแต่ละสถานที่ได้รับสัญญาณ Commit T หรือ Abort T ที่ถูกส่งมาจาก Ci ก็จะทำการบันทึกเรคอร์ดนี้ลงไปใน Log ไฟล์

จะเห็นว่าสัญญาณ Ready T มีความสำคัญมาก เนื่องจากสถานที่ต่าง ๆ ที่ทำ Transaction T จะต้องส่งสัญญาณ Ready T กลับไปให้ตัวประสานงาน Transaction เพื่อบอกกับตัวประสานงาน Transaction ว่าพร้อมที่จะ Commit แล้ว ซึ่งสัญญาณนี้จะมีผลต่อการทำ Commit หรือ Abort ของ

Transaction จากลักษณะดังกล่าวอาจเป็นไปได้ว่าเมื่อสถานที่หนึ่งได้ทำการส่งสัญญาณ Ready T ไปให้ตัวประสานงาน Transaction แล้ว อาจเกิดความล้มเหลวของสถานที่ได้ ซึ่งจากโปรโตคอล 2PC เมื่อตัวประสานงาน Transaction ได้รับสัญญาณ Ready T หรือ Abort T ครบ ก็จะส่งสัญญาณ Commit T หรือ Abort T ไปให้กับ สถานที่ ที่ร่วมทำ Transaction เท่านั้น โดยไม่ได้สนใจว่าสถานที่ต่าง ๆ เหล่านั้นสามารถ Commit หรือ Abort ตามสัญญาณที่ส่งไปได้หรือไม่ ดังนั้นในการดำเนินการโปรโตคอล 2PC จะต้องมีการส่งสัญญาณ Acknowledge T กลับมาที่ตัวประสานงาน Transaction เพื่อเป็นการบอกว่าได้ดำเนินการ ในระยะที่สองเสร็จเรียบร้อยแล้ว เมื่อตัวประสานงาน Transaction ได้รับ Acknowledge T ครบจากทุก ๆ สถานที่แล้ว ก็จะเพิ่มเรคอร์ด <Complete T> เข้าไปใน Log ไฟล์ของ C_i

2.9 Distributed Query Processing (กระบวนการสืบค้นข้อมูลแบบกระจาย)

ในระบบฐานข้อมูลแบบรวม (Centralized databases) ประสิทธิภาพของการสืบค้นข้อมูลจะวัดจากปริมาณของการเข้าถึงข้อมูลในดิสก์ แต่ในระบบฐานข้อมูลแบบกระจายจะต้องพิจารณาเพิ่มเติมอีกคือ

- ค่าใช้จ่ายในการส่งข้อมูลผ่านระบบเครือข่าย
- ประสิทธิภาพของไซต์ที่ประมวลผลแต่ละส่วนของคำสั่งแบบขนาน

ความสัมพันธ์ของค่าใช้จ่ายระหว่างการส่งข้อมูลผ่านระบบเครือข่ายและการส่งข้อมูลจากดิสก์ จะขึ้นอยู่กับประเภทของระบบเครือข่าย และความเร็วของดิสก์ ดังนั้นเราไม่สามารถที่จะระบุลงไปได้เลยว่าจะต้องเสียค่าใช้จ่ายไปกับดิสก์หรือระบบเครือข่ายมากกว่ากัน

2.9.1 Query Transformation

จากตัวอย่าง Query คือ "จงแสดงข้อมูลทุกแถวของของรีเลชัน employee" จะเห็นว่าในระบบฐานข้อมูลแบบรวมจะเป็น Query ที่ง่าย แต่ในระบบฐานข้อมูลแบบกระจาย การประมวลผลQuery นี้จะค่อนข้างยุ่งยากในการประมวลผล เนื่องจากรีเลชัน employee อาจจะถูกทำสำเนาไว้หลาย ๆ ไซต์ หรือถูกแบ่งออกเป็นหลายๆ รีเลชันย่อย หรือถูกทำทั้งสองอย่าง ถ้ารีเลชัน employee ถูกทำสำเนาและไม่ได้ถูกแบ่งออกเป็นรีเลชันย่อย เราก็จะเลือกสำเนาที่มีค่าใช้จ่ายในการส่งข้อมูลน้อยที่สุด อย่างไรก็ตามถ้าสำเนาของรีเลชันมีการแบ่งออกเป็นรีเลชันย่อยด้วย การเลือกก็จะมีความยุ่งยากเพิ่มขึ้น เนื่องจากเราต้องทำการ join หรือ union เพื่อสร้างรีเลชัน employee สำหรับกรณีนี้ ก็สามารถดำเนินการได้หลายวิธี

ตัวอย่างเช่นในระดับ Fragmentation Transparency ผู้ใช้อาจจะเขียนแบบสอบถามเป็น

σ state='New york' (employee)

และเนื่องจาก employee ถูกกำหนดดังนี้

employee1 $\cup$ employee2

สามารถเขียนได้ดังนี้

state='New york' (employee1 $\cup$ employee2)

ถ้าเราทำการOptimize นิพจน์นี้ เราสามารถเขียนเป็นนิพจน์ได้ดังนี้

σ state='New york' (employee1) $\cup$ σ state='New york' (employee2)

ซึ่งทำการแบ่งออกเป็นนิพจน์ย่อย 2 นิพจน์ โดยนิพจน์แรกจะดำเนินการเฉพาะ employee1 ที่ไซต์ New york และนิพจน์ที่สองจะดำเนินการเฉพาะ employee2 ที่ไซต์ Texas

ถ้าเรามีการทำ Optimize ต่อไป โดยพิจารณาที่นิพจน์แรก

σ state='New york' (employee1)

เนื่องจาก employee1 จะมีข้อมูลเฉพาะของ New york เท่านั้น ดังนี้เราสามารถที่จะขจัดการดำเนินการ Selection ออกไปได้ และในนิพจน์ที่สอง

σ state='New york' (employee2)

เราสามารถปรับได้ดังนี้

σ state='New york' (σ state='Texas' (employee))

ผลลัพท์ที่จะเป็นเซตว่าง ดังนั้นเมื่อทำ Qucry Optimize แล้ว ผลลัพธ์จะได้จากการดำเนินการดึงข้อมูลจากไซต์ New york เพียงไซต์เดียว

2.9.2 Simple Join Processing

สิ่งสำคัญในการทำ query-processing คือการเลือกวิธีการ join พิจารณานิพจน์ดังต่อไปนี้

employee $\bowtie$ department $\bowtie$ project

สมมุติว่าทั้งสามรีเลชันไม่ได้ถูกทำสำเนาและไม่ได้ถูกแบ่งเป็นรีเลชันย่อย และ employee ถูกเก็บไว้ที่ไซต์ S1 department เก็บไว้ที่ไซต์ S2 และ project เก็บไว้ที่ไซต์ S3 และกำหนด Si เป็นไซต์ที่จะส่งผลลัพธ์ของการ Query ไปให้ ดังนั้นวิธีการที่เป็นไปได้สำหรับประมวลผล Query นี้คือ

1. ส่งสำเนาของทั้ง 3 รีเลชันไปที่ไซต์ Si และใช้เทคนิคต่างๆ ในการ Query ข้อมูลที่ไซต์ Si

2. ส่งสำเนาของรีเลชัน employee ไปที่ไซต์ S2 และทำการประมวลผล temp1= employee ⋈ department ที่ไซต์ S2 จากนั้นส่ง temp1 จากไซต์ S2 ไปยังไซต์ S3 และประมวลผล temp2 = temp1 ⋈ project และส่ง temp2 ไปยังไซต์ Si

3. ทำในลักษณะคล้าย ๆ กับวิธีการที่สอง แต่สลับไซต์ในการส่งข้อมูล

ไม่มีวิธีไหนที่ดีที่สุด เราต้องพิจารณาระหว่างปริมาณของข้อมูลที่จะต้องส่งระหว่างไซต์ ค่าใช้จ่ายในการส่งผ่านข้อมูลระหว่างสองไซต์ และความเร็วในการประมวลผลของแต่ละไซต์ ซึ่งในวิธีการแรก ถ้าเราส่งข้อมูลทั้งหมดไปที่ไซต์ Si โดยที่รีเลชันเหล่านั้นมีการสร้างอินเด็กซ์ ดังนั้นเราจำเป็นที่จะต้องสร้างอินเด็กซ์เหล่านั้นที่ไซต์ Si ด้วย ซึ่งการสร้างอินเด็กซ์ทำให้มีการประมวลผลเพิ่มขึ้นมา และยังมีการใช้ดิสก์เพิ่มขึ้นอีก อย่างไรก็ตามวิธีการที่สองก็มีข้อเสียคือรีเลชันที่ได้จากการทำประมวลผลมีขนาดใหญ่(employee ⋈ dep ⋈ artment) ซึ่งต้องส่งข้อมูลจากไซต์ S2 ไปที่ไซต์ S3 ซึ่งวิธีการที่สองจะทำให้มีการส่งข้อมูลบนระบบเครือข่ายมากกว่า เมื่อเทียบกับวิธีการที่หนึ่ง

2.9.3 Semijoin Strategy

แนวคิดการทำ Query แบบกระจายโดยใช้วิธีการทำ semijoin มีจุดประสงค์เพื่อลดจำนวนของแถวใน รีเลชันก่อนที่จะทำการส่งให้ไซต์อื่น สมมุติว่าเราต้องการประมวลผลนิพจน์ r1 ⋈ r2 ซึ่ง r1 และ r2 เก็บอยู่ที่ไซต์ S1 และ S2 ตามลำดับ กำหนดให้ R1 และ R2 แทนschema ของ r1 และ r2 ตามลำดับ สมมุติว่าเราต้องการผลลัพธ์ที่ S1 ถ้ามีแถวหลาย ๆ แถวใน r2 ที่ไม่ได้ join กับแถวใด ๆ ใน r1 ดังนั้นการส่งรีเลชัน r2 ทั้งหมดไปที่ไซต์ S1 ก็จะเป็นการส่งที่ทำให้เกิดการ join เกิดผลลัพธ์ที่เกินมาได้ ดังนั้น ก่อนที่เราจะส่งข้อมูลจาก r2 ไปที่ไซต์ S1 ก็น่าจะส่งเฉพาะแถวสามารถ join กับ รีเลชัน r1 ที่ไซต์ S1 เท่านั้น เราสามารถดำเนินการดังกล่าวได้ดังนี้

1. หา temp1 ← Π R1 $\cap$ R2 (r1) ที่ไซต์ S1
2. ส่ง temp1 จากไซต์ S1 ไป S2
3. หา temp2 ← r2 ⋈ temp1 ที่ไซต์ S2

4. ส่ง temp2 จากไซต์ S2 ไป ไซต์ S1

5. ประมวลผล r1 ⋈ temp2 ที่ไซต์ S1

ในขั้นตอนที่ 3 temp2 สามารถหาได้จาก r2 ⋈ Π R1 $\cap$ R2 (r1) และ
ในขั้นตอนที่ 5 ทำ r1 ⋈ r2 ⋈ Π R1 $\cap$ R2 (r1) เราสามารถเขียนนิพจน์ใหม่ได้ดังนี้

(r1 ⋈ Π R1 $\cap$ R2 (r1)) ⋈ r2

เนื่องจาก r1 ⋈ Π R1 $\cap$ R2 (r1)= r1 ดังนั้นนิพจน์นี้จะเท่ากับ r1 ⋈ r2

วิธีการนี้จะมีความเหมาะสมในกรณีที่จำนวนของแถวของ r2 มีจำนวนน้อย วิธีการดำเนินการแบบ semijoin แทนด้วยสัญลักษณ์ ⋈ ดังนั้น semijoin ของ r1 และ r2 เขียนแทนด้วย r1 ⋈ r2 คือ

Π R1(r1 ⋈ r2)

ดังนั้น r1 ⋈ r2 จะเป็นการเลือกแถวของ r1 เพื่อการทำ r1 ⋈ r2 ซึ่งในขั้นตอนที่ 3 จะสามารถเขียนได้ใหม่ดังนี้

temp2 = r2 ⋈ r1

2.9.4 Join Strategies that Exploit Parallelism

พิจารณาการ join ของรีเลชัน 4 รีเลชัน

r1 ⋈ r2 ⋈ r3 ⋈ r4

ซึ่งรีเลชัน ri จะถูกเก็บไว้ที่ไซต์ Si สมมุติว่าเราต้องการผลลัพธ์ที่ไซต์ S1 ซึ่งก็มีอยู่หลายวิธีการที่จะดำเนินการแบบขนาน ยกตัวอย่างเช่น r1 ถูกส่งไปที่ไซต์ S2 และทำ r1 ⋈ r2 ที่ไซต์ S2 ในขณะเดียวกัน r3 ก็ถูกส่งไปที่ไซต์ r4 และทำ r3 ⋈ r4 ที่ไซต์ S4 ไซต์ S2 จะส่งผลลัพธ์ (r1 ⋈ r2) กลับไปที่ไซต์ S1 และไซต์ S4 จะส่งผลลัพธ์ (r3 ⋈ r4) กลับไปที่ไซต์ S1 เมื่อ S1 ได้รับผลลัพธ์กลับจาก S2 และ S4 แล้ว ก็จะทำการ join อีกครั้ง (r3 ⋈ r4) ⋈ (r1 ⋈ r2) ดังนั้นจะเห็นว่าการประมวลผลการ join ที่ S1 สามารถดำเนินการแบบขนานโดยมีการประมวลผลที่ไซต์ S2 และไซต์ S4 ไปพร้อมๆ กัน

2.10 เปรียบเทียบระหว่าง Distributed Database System กับ Centralized Database System

ในการเปรียบเทียบระหว่างระบบฐานข้อมูลแบบกระจายกับระบบฐานข้อมูลแบบรวม จะยึดเอาความสามารถของฐานข้อมูลแบบรวมเป็นหลัก แล้วพิจารณาว่าฐานข้อมูลแบบกระจายมีความสามารถแค่ไหน ซึ่งหัวข้อที่นำมาพิจารณาประกอบไปด้วย Centralized control, Data independence Reduction of redundancy, Complex physical structure for efficient access, Integrity, Recovery, Concurrency control, Privacy และ Security

Centralized control คือ ความสามารถในการควบคุมจัดการข้อมูลโดยรวมทั้งหมด ซึ่งหน้าที่นี้เป็นของ ผู้ดูแลระบบฐานข้อมูล Database administrator (DBA)

ในระบบฐานข้อมูลแบบกระจาย แนวความคิดนี้จะไม่ได้ให้ความสำคัญมากนัก โดยทั่วไปแล้ว ในระบบฐานข้อมูลแบบกระจายจะมีระดับการควบคุมอยู่ โดยที่ผู้ดูแลระบบฐานข้อมูลโดยรวม (Global database administrator) มีหน้าที่ในการดูแลฐานข้อมูลทั้งหมด ส่วนผู้ดูแลระบบฐานข้อมูลระดับท้องถิ่น (Local database administrator) มีหน้าที่ในการดูแลระบบฐานข้อมูล ณ สถานที่ต่างๆ ซึ่งจุดนี้จะต้องเน้นว่าผู้ดูแลระบบส่วนท้องถิ่นต้องมีความสำคัญสูงกว่าผู้ดูแลระบบโดยรวม

Data independence หมายถึง การเปลี่ยนแปลงที่อยู่ของข้อมูลจะต้องไม่ส่งผลกระทบกับโปรแกรม

ในระบบฐานข้อมูลแบบกระจาย Data independence ก็มีความสำคัญเช่นกัน และได้มีข้อกำหนดใหม่ขึ้นมา ซึ่งเรียกว่า Distribution transparency ซึ่งหมายถึง โปรแกรมสามารถถูกเขียนขึ้นมาเหมือนกับว่าไม่ใช่ฐานข้อมูลแบบกระจาย ดังนั้นการเคลื่อนย้ายข้อมูลจะไม่ส่งผลกับความถูกต้องของโปรแกรม อย่างไรก็อาจส่งผลกับความเร็วในการประมวลผลโปรแกรม

Reduction of redundancy ในระบบฐานข้อมูลแบบรวม ความซ้ำซ้อนของข้อมูลจะถูกจำกัดด้วยเหตุผล 2 กรณี ด้วยกันคือ กรณีความไม่สอดคล้องกันของชุดสำเนาของข้อมูล ซึ่งสามารถหลีกเลี่ยงโดยการทำสำเนาเพียงแค่ 1 ชุดเท่านั้น ส่วนอีกกรณีนั้นเพื่อการประหยัดเนื้อที่ที่ใช้เก็บข้อมูล

ในระบบฐานข้อมูลแบบกระจาย ความซ้ำซ้อนเป็นสิ่งที่ต้องการ จากเหตุผลที่ว่า เพื่อทำให้โปรแกรมสามารถเข้าถึงข้อมูลได้เลยเมื่อต้องการ โดยการทำสำเนาของข้อมูลไว้ในทุกๆ สถานที่ และถ้ามีกรณีที่สถานที่ที่เก็บข้อมูลไว้อยู่ไม่สามารถทำงานได้ การทำงานของโปรแกรมก็ยังคงดำเนินต่อไปได้เพราะว่าได้มีการทำซ้ำข้อมูลเอาไว้แล้ว แต่การทำซ้ำของข้อมูลก็มีข้อเสียเช่นกันคือ มีเมื่อมีการเปลี่ยนแปลงของข้อมูลที่สถานที่ใด ต้องตามเปลี่ยนแปลงข้อมูล ที่สถานที่อื่นๆ ด้วย

Complex physical structure and efficient access ในฐานข้อมูลแบบกระจายนั้น เมื่อมีการต้องการประมวลผลข้อมูลที่ซับซ้อนซึ่งมีขนาดใหญ่และอยู่คนละที่กันนั้น เช่นการเรียกดูข้อมูลที่เกิดจากการรวมกันข้องข้อมูลซึ่งต้องมีการวนรอบในการทำงาน และถ้าการวนรอบในการทำงาน

เหล่านี้มีข้อมูลอยู่ต่างสถานที่กัน จะทำให้ต้องมีการเชื่อมต่อเพื่อส่งข้อมูลไปตรวจสอบไปมา ทำให้มีปัญหาในเรื่องเวลาของการประมวลผล ซึ่งจะต่างจากฐานข้อมูลที่อยู่ที่เดียวซึ่งจะเป็นการส่งข้อมูลภายในซึ่งมีความเร็วมากกว่าการส่งข้อมูลข้ามระบบเครือข่ายมาก

Integrity, recovery, and concurrency control ความถูกต้องของข้อมูล การฟื้นฟูข้อมูล และการควบคุมการสภาวะการทำงาน ปัญหาเหล่านี้เกิดมาจากการทำ Transaction ซึ่งจะต้องดูแลให้มีคุณสมบัติ Atomic unit

ในระบบฐานข้อมูลแบบกระจาย ในการดูแลให้การทำ Transaction มีคุณลักษณะ Atomic unit กรณีของความถูกต้องของข้อมูล (Integrity) การควบคุมสภาวะการทำงาน (Concurrency control) รวมทั้งการฟื้นฟูข้อมูล (Recovery) จะมีความยุ่งยากมากกว่าในระบบฐานข้อมูลแบบรวม เพราะจำเป็นที่จะต้องทำการดูแลควบคุมในหลายๆ สถานที่

Privacy and security ในระบบฐานข้อมูลแบบรวม ผู้แลระบบฐานข้อมูลจะมีสิทธิในการควบคุมจัดการระบบฐานข้อมูลได้ทั้งหมด ทำให้มั่นใจได้ว่าเฉพาะผู้ที่ได้รับอนุญาตเท่านั้นที่สามารถเข้าถึงข้อมูลได้ อย่างไรก็ตามหากเป็นระบบฐานข้อมูลแบบรวมที่ไม่มีการควบคุมที่ดีพออาจจะไม่ปลอดภัยจากผู้ไม่ประสงค์ดี

ส่วนในระบบฐานข้อมูลแบบกระจาย สำหรับผู้ดูแลระบบฐานระดับท้องถิ่นจะประสบปัญหาเดียวกับกรณีของระบบฐานข้อมูลแบบรวม อย่างไรก็ตามข้อกำหนดที่พิเศษเพิ่มเติมของระบบฐานข้อมูลแบบกระจาย ข้อแรกคือ ผู้ดูแลระบบฐานข้อมูลระดับท้องถิ่นจะมีสิทธิเหนือกว่าผู้ดูแลระบบฐานข้อมูลโดยรวมในการควบคุมจัดการฐานข้อมูล ณ สถานที่นั้นๆ ข้อสองก็คือ ปัญหาเรื่องความปลอดภัย เพราะว่าระบบฐานข้อมูลแบบกระจายมีการเชื่อมต่อผ่านเครือข่ายคอมพิวเตอร์ซึ่งอาจเป็นจุดอ่อนในเรื่องความปลอดภัยของข้อมูล

2.11 เหตุผลที่เลือก Distributed Database System

1. งานฐานข้อมูลบางรูปแบบมีข้อมูลกระจายอยู่ตามแหล่งต่างๆ หน่วยงานบางแห่ง อาจจะมีสาขาอยู่ในหลายจังหวัด ตัวอย่างเช่น ธนาคารซึ่งมีสาขาอยู่ทั่วประเทศ ข้อมูลของแต่ละสาขาจะถูกเก็บไว้ที่สาขานั้นๆ แต่ในบางครั้งผู้บริหารหน่วยงานที่ส่วนกลางต้องการข้อมูลสรุป ซึ่งต้องใช้ข้อมูลจากหลายๆ สาขาทั่วประเทศ ซึ่งในกรณีนี้ ระบบจัดการฐานข้อมูลควรจะมีความสามารถในการดึงข้อมูลจากสาขาต่างๆ ได้ด้วย

2. เพิ่มความน่าเชื่อถือ เนื่องจากในระบบฐานข้อมูลแบบกระจายมีการจัดเก็บข้อมูลไว้ในหลาย ๆ ที่ นั่นคือเมื่อมีที่ใด ที่หนึ่งเกิดความล้มเหลวขึ้นทำให้ไม่สามารถเข้าถึงข้อมูลในสถานที่นั้นได้ ระบบสามารถที่จะไปหาข้อมูลจากสถานที่อื่นได้ ซึ่งแตกต่างจากระบบแบบรวม ถ้าเกิดความล้มเหลวขึ้น จะทำให้ไม่สามารถใช้งานได้เลย

3. เพื่อควบคุมการเข้าใช้ข้อมูล ทั้งนี้เนื่องจากข้อมูลกระจัดกระจายอยู่ตามจุดต่างๆ ทำให้ผู้ดูแลระบบสามารถกำหนดได้ว่า ผู้ใช้จากจุดใดสามารถเข้าใช้ข้อมูลจากจุดใดได้บ้าง และได้มากระดับใดด้วย

4. เพิ่มประสิทธิภาพการทำงาน เมื่อมีการกระจายข้อมูลตามจุดต่าง ๆ จะส่งผลให้ฐานข้อมูลย่อยแต่ละจุดมีขนาดเล็กลง ซึ่งทำให้เมื่อมีการสอบถามข้อมูลในแต่ละฐานข้อมูลย่อย การค้นหาข้อมูลย่อมทำได้รวดเร็วขึ้น นอกจากนี้ ผู้ใช้ก็กระจายอยู่ตามจุดต่าง ๆ แต่ละจุด จึงสามารถทำงานขนานกันได้เลย ซึ่งไม่เหมือนกับระบบรวมที่จะต้องส่งทุก ๆ คำสั่งไปที่เดียวกันหมด และทำให้ต้องรอลำดับในการทำงาน เพราะไม่สามารถทำงานพร้อม ๆ กันได้

บทที่ 3

ความสามารถของ Oracle ในการทำฐานข้อมูลแบบกระจาย

จากแนวความคิดของ Distributed Database ซึ่งได้มีการคิดค้นมาเป็นเวลาหลายปี และในปัจจุบัน ผู้พัฒนาซอฟท์แวร์ฐานข้อมูล (Database Management System) หลายบริษัทได้เริ่มให้ความสนใจในการพัฒนาฐานข้อมูลที่สนับสนุนแนวความคิดของฐานข้อมูลแบบกระจาย โดยในโครงงานชิ้นนี้ได้เลือกที่จะศึกษาความก้าวหน้าของการทำฐานข้อมูลแบบกระจาย ที่พัฒนาขึ้นโดยบริษัท Oracle เนื่องมาจากเป็นซอฟท์แวร์ฐานข้อมูล ที่มีความสามารถซึ่งเป็นที่ยอมรับกันในอุตสาหกรรมที่เกี่ยวเนื่องกับฐานข้อมูลและมีการพัฒนาเทคโนโลยีอย่างต่อเนื่อง

ในโครงงานนี้ได้ทำการศึกษาความก้าวหน้าของซอฟท์แวร์ฐานข้อมูลที่พัฒนาโดยบริษัท Oracle ซึ่งในปัจจุบันได้พัฒนามาถึงเวอร์ชัน 10 โดยซอฟท์แวร์ที่นำมาทำการศึกษาทดลองคือ Oracle 10g Enterprise Edition Release 2 เพื่อให้สามารถใช้งานบริการต่างๆที่ Oracle พัฒนาขึ้นได้อย่างเต็มที่ ซึ่ง Oracle ได้รองรับการทำฐานข้อมูลแบบกระจายได้ในระดับหนึ่ง โดยจะอธิบายด้วยการเปรียบเทียบแนวความคิดของฐานข้อมูลแบบกระจายตามโครงสร้างที่ได้อธิบายไว้ข้างต้นเทียบกับบริการที่ Oracle ได้พัฒนาออกรองรับไว้

ลักษณะการทำฐานข้อมูลของแบบกระจายของ Oracle

จากแนวความคิดของฐานข้อมูลแบบกระจายซึ่งได้อธิบายไว้ในบทที่ 2 ในส่วนนี้จะแสดงให้เห็นถึงความสามารถของ Oracle ที่จะทำให้แนวคิดในระดับต่างๆของสถาปัตยกรรมที่ออกแบบไว้ สามารถใช้งานได้จริง แต่เนื่องจากในโครงงานชิ้นนี้นั้นได้ทำการทดลองติดตั้งฐานข้อมูลแบบกระจายโดยใช้ซอฟท์แวร์ของ Oracle ทั้งหมดและใช้รูปแบบข้อมูลแบบเดียวกัน คือ Relational Database จึงทำให้ไม่สามารถทดลองในส่วนของการทำการกระจายแบบ Heterogeneous และจะทำการทดลองเริ่มทดสอบที่ระดับ Local mapping Transparency ได้เลย

3.1 การเตรียมการเพื่อเชื่อมต่อฐานข้อมูล

ในการทำฐานข้อมูลแบบกระจายนั้น ถ้าเป็นการทำโดยการใช้ซอฟท์แวร์ของ Oracle เพียงอย่างเดียว ก่อนที่จะทำให้ฐานข้อมูลแต่ละที่สามารถเชื่อมต่อกันได้นั้น จะมีตั้งค่าการใช้งานชื่อของฐานข้อมูลในการอ้างอิงซึ่ง Oracle แนะนำให้มีการตั้งค่าชื่อของเซิร์ฟเวอร์ที่ฐานข้อมูลตั้งอยู่ด้วยเพื่อใช้ร่วมกับชื่อของฐานข้อมูลทำให้สามารถอ้างอิงได้ง่ายรวมทั้งการนำชื่อของฐานข้อมูลมาต่อ

รวมกับชื่อของเซิร์ฟเวอร์ทำให้ชื่อของฐานข้อมูลนั้นไม่ซ้ำกันทั้งระบบ สามารถอ้างอิงได้แม้ว่าจะมีชื่อฐานข้อมูลเดียวกันแต่อยู่คนละสถานที่ซึ่ง Oracle ได้เรียกชื่อนี้ว่า Global Name

ตัวอย่างเช่น มีฐานข้อมูลชื่อ ORCL โดยถ้าเซิร์ฟเวอร์ที่เก็บฐานข้อมูลนี้มีชื่อที่สามารถอ้างอิงได้จากระบบเครือข่ายคือ UNGRAD15.CE.KMITL.AC.TH เมื่อมีการตั้งค่าชื่อเซิร์ฟเวอร์เรียบร้อยแล้ว ชื่อที่เป็น Global ของฐานข้อมูลนี้คือ ORCL.UNGRAD15.CE.KMITL.AC.TH

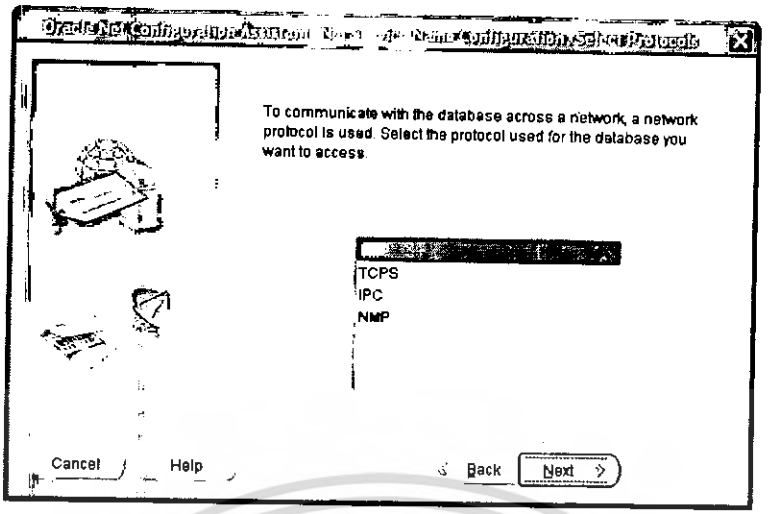
3.2 การทำ Local Mapping Transparency

เมื่อมีชื่อของฐานข้อมูลที่ตั้งค่าอย่างถูกต้องแล้ว การทำให้ฐานข้อมูลแต่ละที่เชื่อมต่อกันได้จะใช้บริการที่ Oracle พัฒนาขึ้น คือ Database Link ซึ่งก่อนการสร้าง Database Link นั้นจำเป็นที่จะต้องมีการตั้งค่าอีกเล็กน้อย คือการตั้งค่าสำหรับกำหนดการเชื่อมต่อสำหรับแต่ละสถานที่เพื่อกำหนดให้โปรแกรมทราบว่าเมื่อจะเชื่อมต่อไปยังสถานที่นั้นๆจะต้องใช้โปรโตคอลใด พอร์ทใด และมีข้อกำหนดอย่างไรบ้าง เพราะแต่ละสถานที่นั้นอาจจะมีการใช้งานการเชื่อมต่อเครือข่ายแตกต่างกัน

เมื่อมีการตั้งค่าการเชื่อมต่อของฐานข้อมูลที่เราจะเชื่อมต่อด้วยเรียบร้อยแล้ว ขั้นต่อมาเป็นการสร้าง Database Link ซึ่งสามารถสร้างได้หลายวิธีเช่น การใช้คำสั่งภาษา SQL ในการสร้าง หรือสามารถสร้างได้โดยการใช้ Oracle Enterprise Manager Console ซึ่งเป็นซอร์ฟแวร์ที่ทาง Oracle พัฒนาขึ้นเพื่อช่วยในการตั้งค่าต่างๆของระบบโดยไม่ต้องพิมพ์คำสั่งเอง ตัวอย่างการสร้าง Database Link โดยใช้คำสั่งภาษา SQL

```
CREATE DATABASE LINK ORCL.UNGRAD06.CE.KMITL.AC.TH USING
'UNGRAD06.CE.KMITL.AC.TH';
```

มีความหมายคือทำการสร้างการเชื่อมต่อฐานข้อมูลไปยังเซิร์ฟเวอร์ชื่อ ungrad06.ce.kmitl.ac.th โดยใช้ชื่อของการเชื่อมต่อที่เราได้สร้างขึ้นไว้ก่อนหน้านี้



รูปที่ 3.1 หน้าต่างสำหรับการตั้งค่า Name Service

การสร้างและใช้งาน Database Link นั้นมีการรักษาความปลอดภัยเข้ามาเกี่ยวข้อง และ Database Link ที่สร้างขึ้นนั้นต้องมีการกำหนดชนิดของการเชื่อมต่อ สามารถกำหนดได้ว่าจะให้ผู้อื่นสามารถใช้งาน Database Link นี้ได้หรือไม่ รวมทั้งการใช้งาน Database Link นั้นยังต้องมีการกำหนดลักษณะของการตรวจสอบการเข้าใช้งาน ณ ฐานข้อมูลปลายทางด้วย โดยสามารถกำหนดได้ว่าจะให้เข้าใช้งานโดยใช้ชื่อผู้ใช้งานและรหัสชุดเดียวกับที่เชื่อมต่ออยู่ หรือเข้าใช้งานโดยชื่อผู้ใช้อื่น

รูปที่ 3.2 การใช้ Enterprise Manager Console ช่วยสร้าง Database Link

แต่ถ้ามีการทำฐานข้อมูลแบบกระจายโดยมีซอฟท์แวร์จากผู้ผลิตอื่นเชื่อมต่ออยู่ด้วย Oracle ได้มีการพัฒนาซอฟท์แวร์ที่ต้องนำมาติดตั้งเพิ่มเติม คือ Oracle Transparent Gateway เพื่อทำหน้าที่แปลงข้อมูลที่ส่งระหว่างฐานข้อมูลของ Oracle ให้ซอฟท์แวร์ฐานข้อมูลจากผู้ผลิตรายอื่นสามารถ

ทำงานร่วมกันได้ และยังมีซอฟท์แวร์ Generic Connectivity ที่ช่วยในการเชื่อมต่อไปยังซอฟท์แวร์ฐานข้อมูลอื่นผ่านทาง ODBC

3.3 Location Transparency

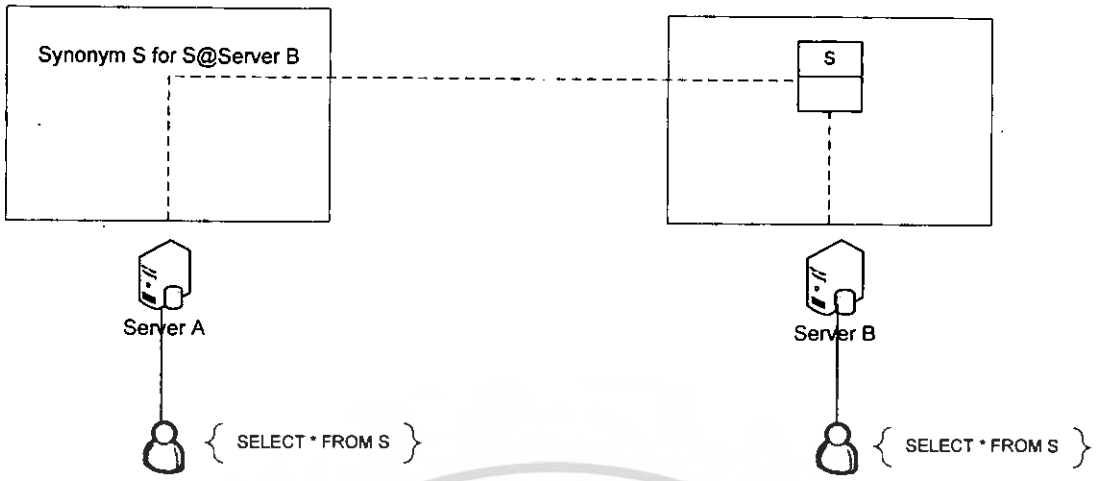
3.3.1 Synonym

คือ การที่ผู้ใช้งานสามารถเรียกข้อมูลที่ต้องการได้โดยไม่จำเป็นต้องทราบว่าข้อมูลนั้นถูกเก็บไว้ที่สถานที่ใด ใน Oracle สามารถทำ Location Transparency ได้หลายวิธีเช่น Synonym, View โดยวิธีที่นำมาทดลองนั้นคือ ทำการสร้าง Synonym ซึ่ง Synonym ที่สร้างขึ้นจะทำงานเหมือนกับการสร้างชื่อเสมือนขึ้นมาให้ข้อมูล เพื่อเป็นการให้ผู้ใช้งานเรียกใช้ชื่อเสมือนที่สร้างขึ้นแทนชื่อเดิมซึ่งยาวและยังต้องมีการระบุที่อยู่ของข้อมูลด้วย โดยเมื่อสร้าง Synonym ขึ้นมาให้ผู้ใช้งานเรียกใช้นั้น ทำให้ผู้ใช้งานไม่จำเป็นต้องรู้ว่าแท้จริงแล้วข้อมูลถูกเก็บไว้ที่ใด

รูปแบบของคำสั่งภาษา SQL ในการสร้าง Synonym ของ Oracle คือ

```
CREATE [PUBLIC] synonym_name
FOR [schema.]object_name[@database_link_name];
```

ตัวอย่างการทำ Location Transparency เช่น จากรูปที่ 3.5 มีผู้ใช้งานต้องการจะ Select ข้อมูลจากตาราง S ซึ่งอยู่ที่ Server B ซึ่งผู้ใช้งานที่เชื่อมต่ออยู่กับ Server B สามารถใช้คำสั่งSQL "SELECT * FROM B" ได้โดยตรง แต่ถ้าหากผู้ใช้งานที่เชื่อมต่ออยู่กับ Server A ต้องใช้คำสั่ง "SELECT * FROM S@Server B" ซึ่งไม่ถือว่าเป็น Location Transparency แต่เมื่อทำการสร้าง Synonym S เพื่อแทนที่ S@Server B ทำให้ ผู้ใช้งานที่เชื่อมต่ออยู่กับ Server A สามารถใช้คำสั่ง "SELECT * FROM B" ได้เช่นเดียวกัน ซึ่งจะเห็นได้ว่าหลังจากการสร้าง Synonym แล้ว ผู้ใช้งานทั้งสองที่สามารถใช้คำสั่ง SQL เดียวกันได้เพื่อเรียกข้อมูลจากตาราง S โดยไม่จำเป็นต้องทราบว่าตาราง S นั้นมีข้อมูลอยู่ที่ Server ใด



รูปที่ 3.3 ตัวอย่างการใช้ Synonym ในการทำ Location Transparency

3.3.2 การทำ Replication

Oracle ได้สนับสนุนการทำReplication ทั้งแบบ Synchronous Replication และ Asynchronous Replication ซึ่งทั้งสองแบบนั้นสามารถทำโดยการใช้งานบริการของ Oracle ที่ชื่อว่า Oracle Database Advanced Replication.

- **Synchronous Replication**

จากแนวความคิดการทำ Replication นั้นใน Oracle นั้นสามารถทำ Synchronous ได้อย่างสมบูรณ์โดยการใช้งาน Advance Replication ซึ่งสามารถตั้งค่าการทำงานได้ทั้งโดยคำสั่ง SQL และการใช้ Enterprise Manager Console ซึ่งเป็น Java Application ที่มี Graphic user interface ช่วยในการตั้งค่าทำให้สามารถตั้งค่าได้ง่ายขึ้น

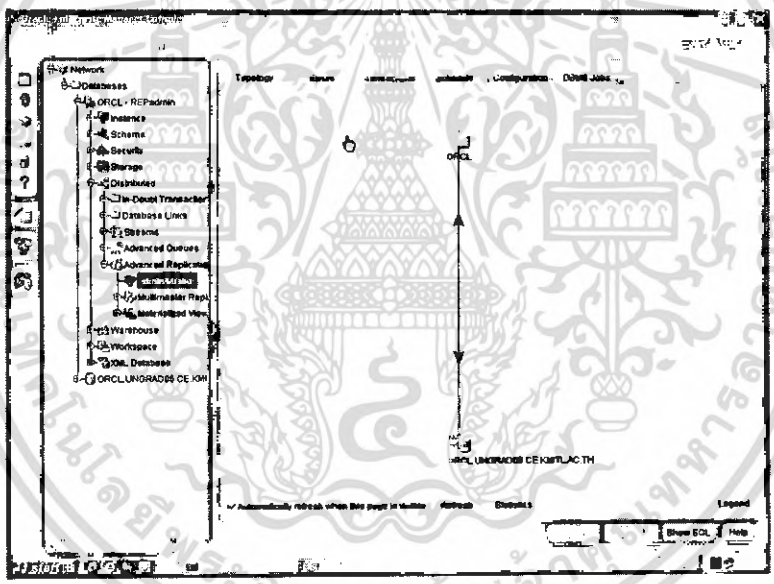
ในขั้นตอนการสร้างการ Replicate ข้อมูลนั้น Oracle จะมีการสร้าง Master Site เป็นการกำหนดเพื่อบอกว่าข้อมูลที่อยู่ในเซอร์เวอร์นี้นั้นเป็นข้อมูลหลักซึ่งสามารถเรียกดูและปรับปรุงแก้ไขได้ ใน Oracle นั้นได้จัดแบ่งการทำ Replication เป็นสามชนิดหลักๆคือ

1. Multimaster Replication คือทำการเชื่อมต่อกลุ่มของ Master Site เข้าด้วยกันในลักษณะของ peer-to-peer เพื่อทำการ Replication
2. Materialized View Replication เป็นการทำ Replication โดยหลีกเลี่ยงการใช้งาน 2 Phase Commit โดยจะไม่จำเป็นต้องมีการแก้ไขข้อมูลทันทีเมื่อข้อมูลชุดอื่นๆเปลี่ยนแปลง แต่จะ

ทำการปรับปรุงข้อมูลเป็นระยะๆแทน ซึ่งจะกล่าวต่อไปในส่วนของการทำ Asynchronous Replication

3. <u>แบบผสม</u> คือการนำ Multimaster และ Materialized View มาผสมกันเพื่อเพิ่มประสิทธิภาพในการทำงาน

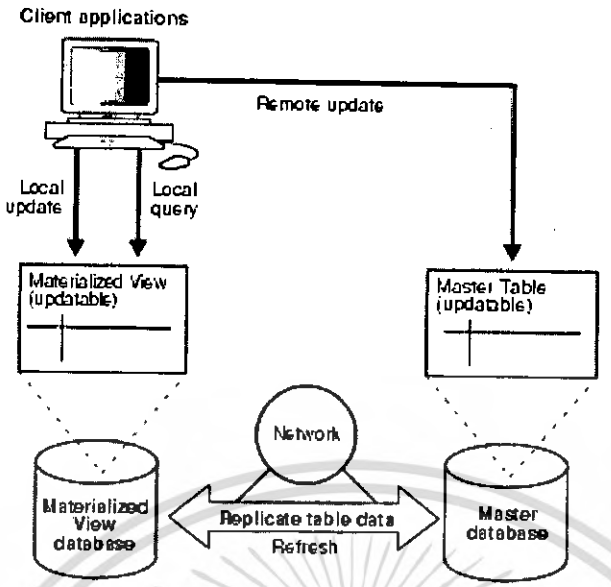
และการทำ Synchronous Replicate ใน Oracle จะใช้ Multimaster Replication ในการทำงาน โดยเมื่อกำหนดให้เซิร์ฟเวอร์ที่มีข้อมูลซึ่งต้องการทำ Replicate ทุกๆเซิร์ฟเวอร์เป็น Master Site แล้ว ขั้นต่อมาเป็นการสร้าง Replication Group เพื่อทำการกำหนดว่าจะให้เซิร์ฟเวอร์ใดทำการ Replicate ข้อมูลใดบ้าง ซึ่งจากการทดลอง Oracle สามารถทำ Synchronous Replicate ได้อย่างสมบูรณ์ โดยเมื่อมีการแก้ไขข้อมูลในที่ใดที่หนึ่ง ซอฟท์แวร์ฐานข้อมูลจะไปแก้ไขข้อมูลที่อื่นให้ด้วย และเมื่อมีการแก้ไขข้อมูลในขณะที่มีเซิร์ฟเวอร์ที่เก็บข้อมูลชุดเดียวกันแต่ไม่สามารถทำงานได้ ปรากฏว่าไม่สามารถทำการแก้ไขข้อมูลได้ ตามข้อกำหนดของ 2 Phase Commit Protocol



รูปที่ 3.4 หน้าต่างแสดงการเชื่อมต่อเมื่อทำ Synchronous Replicate สำเร็จแล้ว

- **Asynchronous Replication**

ปัจจุบัน Oracle ได้สนับสนุนการทำ Asynchronous ไว้ในหลายๆบริการได้แก่ Snapshot, Materialized View



รูปที่ 3.5 ตัวอย่างการทำงานของ Materialized View แบบแก้ไขได้ ของ Oracle

ใน Oracle นั้นสามารถสร้าง Materialized View ได้สองแบบคือ แบบที่สามารถอ่านได้อย่างเดียวและแบบที่สามารถแก้ไขข้อมูลได้ด้วย การใช้ Materialized View แบบแก้ไขได้นั้นสะดวกคือสามารถแก้ไขข้อมูลได้แม้กระทั้งไม่สามารถติดต่อกับ Master Site อื่นๆได้ แต่ต้องมีการกำหนดเงื่อนไข เพิ่มเติมในกรณีที่มีการแก้ไขข้อมูลสองที่พร้อมๆกันว่าจะให้ใช้เงื่อนไขใดเพื่อพิจารณาข้อมูล การตรวจสอบความเปลี่ยนแปลงกับข้อมูลนั้น Oracle ได้มีการรองรับการปรับปรุงข้อมูลสามแบบ คือ

- Complete Refresh เป็นการปรับปรุงข้อมูลโดยการคัดลอกข้อมูลและโครงสร้างของข้อมูลจากต้นฉบับใหม่ทั้งหมด
- Fast Refresh เป็นการปรับปรุงข้อมูลโดยตรวจสอบความเปลี่ยนแปลงจากการปรับปรุงครั้งหลังสุด รวมทั้งถ้าเป็น View ชนิดแก้ไขได้ ก็จะทำการส่งการเปลี่ยนแปลงที่เกิดที่ View นี้ไปยังข้อมูลต้นฉบับด้วย ซึ่งเป็นรูปแบบที่มีประโยชน์ในการทำ Replication อย่างมาก
- Force Refresh เป็นการปรับปรุงข้อมูลเมื่อการปรับปรุงข้อมูลแบบ Fast Refresh ไม่สามารถทำงานได้ การทำ Force Refresh นั้นเป็นการสั่งให้ทำการปรับปรุงข้อมูลทันที เพื่อแก้ปัญหาต่างๆที่เกิดข้อผิดพลาดในการปรับปรุงข้อมูล

การตั้งค่าการปรับปรุงข้อมูลระหว่างฐานข้อมูลโดยใช้ Materialized View นั้นต้องมีการสร้าง Refresh Group เพื่อบ่งบอกว่าเซิร์ฟเวอร์ที่อยู่ในกลุ่มนั้นจะทำการปรับปรุงข้อมูลให้เหมือนกันทุกๆระยะเวลาเท่าใดและทำการปรับปรุงข้อมูลโดยใช้วิธีใด

ตัวอย่างการตั้งค่าเวลาที่ใช้ในการปรับปรุงข้อมูล เช่น SYSDATE + 1/24 มีความหมายคือให้มีการปรับปรุงข้อมูลให้เหมือนต้นฉบับที่ Master Site ทุกๆ 1 ชั่วโมง

ตัวอย่างคำสั่งภาษา SQL ในการสร้าง Materialized View แบบพื้นฐานชนิดที่สามารถแก้ไขข้อมูลได้

```
CREATE MATERIALIZED VIEW hr.employee_depts AS
 SELECT DISTINCT department_id FROM hr.employees@orc1.world
        ORDER BY department_id;
```

3.4 Global Query Optimization

ในการประมวลผลคำสั่งที่ใช้ในการเรียกดูข้อมูลนั้น โดยปกติแล้วซอฟท์แวร์ฐานข้อมูลของ Oracle จะทำการเลือกวิธีที่ดีที่สุดโดยใช้ Cost-Based Optimization มาทำงานเพื่อให้ได้ข้อมูลรวดเร็วที่สุด แต่ทั้งนี้ต้องมีเจ้าหน้าที่ดูแล ในการเก็บสถิติและรวบรวมข้อมูล เพื่อให้ซอฟท์แวร์สามารถหาวิธีที่ดีที่สุดได้อย่างถูกต้อง

สำหรับในระบบฐานข้อมูลแบบกระจาย การทำงานของระบบการหาวิธีเข้าถึงข้อมูลที่ดีที่สุดโดยใช้ Cost-Based Optimization นั้น Oracle จะมีการทำงานโดยใช้การคำสั่งให้อยู่ในลักษณะของ Collocated Inline View ซึ่งจะทำการเปลี่ยนคำสั่งที่ผู้ใช้งานให้เป็นคำสั่งย่อยๆ เพื่อส่งไปยังฐานข้อมูลเครื่องอื่นๆที่เป็นเจ้าของข้อมูล ให้ทำการประมวลผลข้อมูลให้ได้ข้อมูลในส่วนที่ต้องการเพื่อลดขนาดการส่งข้อมูลผ่านเครือข่าย แล้วจึงทำการส่งกลับมายังเครื่องที่แจกคำสั่ง หรือเครื่องใดเครื่องหนึ่งโดยจะใช้คำสั่ง DRIVING_SITE Hint แล้วจึงทำการประมวลผลข้อมูลทั้งหมดแล้วส่งผลที่ได้ให้ผู้ใช้งานต่อไป

ต่อไปนี้เป็นวิธีในการตั้งค่าให้ซอฟท์แวร์ฐานข้อมูลของ Oracle สามารถใช้ Cost-Based Optimization ได้ซึ่งประกอบด้วย 2 ขั้นตอนด้วยกัน คือ

1. ตั้งค่า OPTIMIZER_MODE ใน initialization parameter ให้เป็น CHOOSE หรือ COST
2. ทำการสร้างสถิติให้กับตารางที่จะใช้งานโดยการใช้คำสั่ง ANALYZE เช่น ถ้าต้องการสร้างสถิติให้กับตาราง EMP จะสามารถใช้คำสั่งดังนี้

```
ANALYZE TABLE EMP COMPUTE STATISTICS;
```

นอกจากนั้นยังสามารถแสดงวิธีการเข้าถึงข้อมูลของ Oracle ได้ โดยสามารถใช้ ISQL* PLUS ซึ่งมีขั้นตอน คือ

1. ทำการโหลดไฟล์ SQL ไฟล์ที่ชื่อว่า utlxplan.sql จากโฟล์เดอร์ $ORACLE_HOME/rdbms/admin เพื่อทำการสร้างตารางไว้เก็บข้อมูล
2. ใช้คำสั่ง EXPLAIN PLAN FOR นำหน้าชุดคำสั่งที่ต้องการแสดงวิธีการเข้าถึงข้อมูล แล้วใช้คำสั่งต่อไปนี้แสดงผลออกมา

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

หรืออาจจะกำหนดให้ ISQL* PLUS แสดงวิธีการเข้าถึงข้อมูลโดยอัตโนมัติโดยใช้คำสั่งต่อไปนี้ จากนั้นสามารถพิมพ์คำสั่งใน ISQL* PLUS ได้เลย

```
SET AUTOTRACE ON;
```

ตัวอย่างต่อไปนี้เป็นการแสดงการทำ Global Query Optimization ของ Oracle 10g โดยที่จะอธิบายโดยใช้การแสดงวิธีการเข้าถึงข้อมูลใน ISQL* PLUS

โดยใช้คำสั่งสำหรับทำการทดสอบ คือ

```
SELECT * FROM EMP_V;
```

รูปที่ 3.6 กำหนดคำสั่งที่ต้องการแสดงวิธีการเข้าถึงข้อมูลใน ISQL * PLUS

ซึ่ง EMP_V เป็น View ที่สร้างมาจากการทำ Vertical Fragmentation โดยใช้การ Join ตาราง EMP_ONE และ EMP_TWO เข้าด้วยกัน ซึ่งจากการแสดงวิธีการเข้าถึงข้อมูล จะเห็นว่ามีการแยก Remote SQL ออกมาด้วย

```
แผนการรัน
----------------------------------------

Plan hash value: 502959965

-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     | Inst   |IN-OUT|
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |    18 |   810 |     6   (0)| 00:00:01 |        |      |
|   1 |  NESTED LOOPS                 |             |    18 |   810 |     6   (0)| 00:00:01 |        |      |
|   2 |   REMOTE                      | EMP_TWO     |    18 |   450 |     3   (0)| 00:00:01 |   ORCL | R->S |
|   3 |   TABLE ACCESS BY INDEX ROWID | EMP_ONE     |     1 |    20 |     1   (0)| 00:00:01 |        |      |
|*  4 |    INDEX UNIQUE SCAN          | PK_EMP_ONE  |     1 |       |     0   (0)| 00:00:01 |        |      |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("E1"."EMPNO"="E2"."EMPNO")

Remote SQL Information (identified by operation id):
----------------------------------------------------

   2 - SELECT "EMPNO","HIREDATE","SAL","COMM" FROM "SCOTT"."EMP_TWO" "E2" (accessing
       'ORCL.UNGRAD15.CE.KMITL.AC.TH' )
```

รูปที่ 3.7 แสดงวิธีการเข้าถึงข้อมูลโดยที่มีการแยกเป็น Remote SQL

สรุปแล้วก็คือ วิธีที่ Oracle ใช้นั้นถือเป็นวิธีที่มีประสิทธิภาพระดับหนึ่งเท่านั้น เนื่องมาจากซอฟท์แวร์ไม่ได้ทำการเก็บรวบรวมข้อมูลและประมวลผลว่าควรส่งข้อมูลใดไปประมวลผลที่ใดก่อนหรือรวบรวมข้อมูลที่ใดเพื่อให้ได้ประสิทธิภาพดีที่สุด เพราะการส่งข้อมูลทั้งหมดมารวบรวมที่เครื่องใดเครื่องหนึ่งโดยไม่มีการวิเคราะห์ถึงประสิทธิภาพเลยนั้น ทำให้วิธีที่ได้อาจจะไม่ใช่วิธีที่ดีที่สุด และอาจจะเกิดการส่งข้อมูลขนาดใหญ่ผ่านระบบเครือข่ายซึ่งทำให้ใช้เวลาในการรวบรวมข้อมูลนานขึ้น

3.5 Referential Integrity Constraints

ความสามารถของ Oracle ในการควบคุมความถูกต้องของข้อมูลนั้น ถ้าเป็นฐานข้อมูลแบบกระจาย Oracle ไม่สามารถมี Referential Integrity Constraints ระหว่างฐานข้อมูลที่อยู่คนละที่ ได้ แต่ในทางปฏิบัติ สามารถสร้าง Trigger ขึ้นมาเพื่อช่วยในการคงความถูกต้องของข้อมูลให้คงไว้ ซึ่งจะบอกถึงวิธีการสร้างและใช้งานในบทต่อไป

3.6 สรุปความสามารถของ Oracle ในการทำฐานข้อมูลแบบกระจาย

จากการศึกษาความสามารถของ Oracle ในการทำฐานข้อมูลแบบกระจาย เปรียบเทียบกับแนวคิดของสถาปัตยกรรมในระดับต่างๆ พบว่า

- ระดับ Local mapping Transparency Oracle จะใช้ Database link ในการทำ
- ระดับ Location Transparency Oracle จะใช้ Synonym, View และ Advance Replication ในการทำ

ในระดับ Fragmentation Transparency นั้น Oracle ยังไม่รองรับโดยตรง ต้องมีการใช้ View ในการรวบรวมข้อมูล และต้องพัฒนาโปรแกรมย่อย Trigger ช่วยในการทำ ซึ่งจะอธิบายรายละเอียดเพิ่มเติมในบทถัดไป

สำหรับการทำ Global Query Optimization วิธีที่ Oracle ใช้ก็คือ จะทำเปลี่ยนคำสั่งของผู้ใช้งานให้เป็นคำสั่งย่อยๆ เพื่อส่งไปยังฐานข้อมูลเครื่องอื่นๆที่เป็นเจ้าของข้อมูล ให้ทำการประมวลผลข้อมูลแล้วทำการส่งกลับมายังเครื่องที่แจกคำสั่ง หรือเครื่องใดเครื่องหนึ่ง เพื่อทำการประมวลผลข้อมูลทั้งหมดแล้วส่งผลที่ได้ให้ผู้ใช้งานต่อไปใช้ถือเป็นวิธีที่มีประสิทธิภาพระดับหนึ่งเท่านั้น เนื่องมาจากซอฟท์แวร์ Oracle ไม่ได้ทำการเก็บรวบรวมข้อมูลและประมวลผลว่าควรส่งข้อมูลใดไปประมวลผลที่ใดก่อนหรือรวบรวมข้อมูลที่ใดเพื่อให้ได้ประสิทธิภาพดีที่สุด

และสำหรับการกำหนด Referential Integrity Constraints ระหว่างฐานข้อมูลที่อยู่คนละที่ Oracle ก็ไม่ได้สนับสนุนโดยตรง จำเป็นที่จะต้องเขียนโปรแกรมย่อย Trigger มาควบคุมเอง

บทที่ 4

การพัฒนาเพื่อเพิ่มประสิทธิภาพในการทำฐานข้อมูลแบบกระจายของ Oracle

4.1 สิ่งที่จะพัฒนาเพื่อเพิ่มความสามารถให้กับ Oracle ในการทำฐานข้อมูลแบบกระจาย

จากการศึกษาความสามารถของ Oracle ในการทำฐานข้อมูลแบบกระจาย เปรียบเทียบกับแนวคิดของสถาปัตยกรรมในระดับต่างๆ พบว่า Oracle สนับสนุนในการทำฐานข้อมูลแบบกระจายได้ในระดับ Local mapping Transparency และระดับ Location Transparency เท่านั้น

สำหรับระดับ Fragmentation Transparency ใน Oracle นั้นยังไม่รองรับโดยตรง ต้องมีการใช้ View ในการรวบรวมข้อมูล และต้องพัฒนาโปรแกรมย่อย Trigger ช่วยในการทำ ซึ่งการทำ Fragmentation Transparency จะเป็นหนึ่งในหัวข้อที่จะทำการพัฒนาเพื่อเพิ่มความสามารถให้กับ Oracle ในการทำฐานข้อมูลแบบกระจาย

อีกหัวข้อหนึ่งที่จะทำการพัฒนาเพื่อเพิ่มความสามารถให้กับ Oracle ในการทำฐานข้อมูลแบบกระจาย ก็คือการกำหนด Referential Integrity Constraints ระหว่างฐานข้อมูลที่อยู่คนละที่ โดยการเขียนโปรแกรมย่อย Trigger มาควบคุมเอง ซึ่งสามารถกำหนดให้กับตัว Global Table ที่มาจากการทำ Fragmentation Transparency ได้ด้วยเช่นกัน

4.2 การทำ Fragmentation Transparency

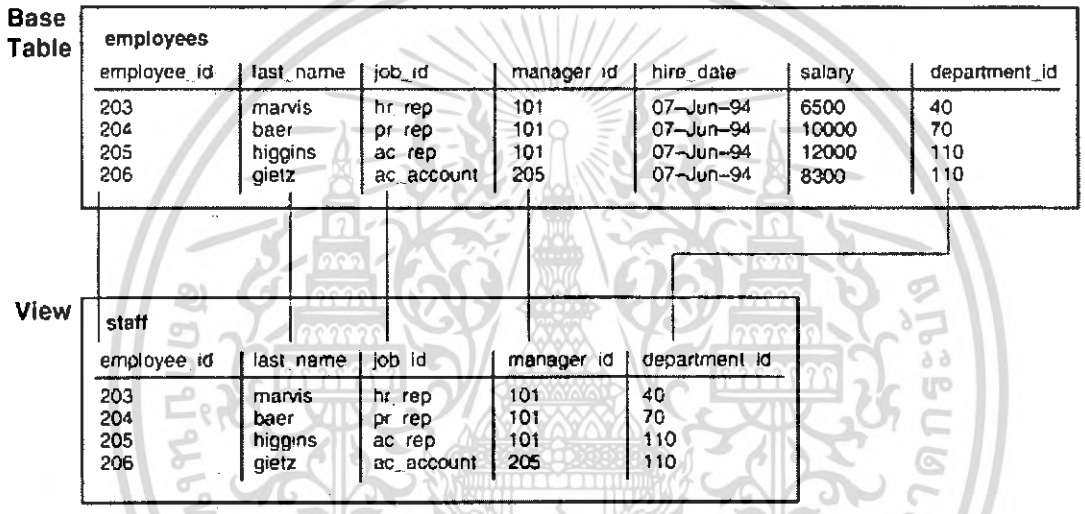
Fragmentation Transparency คือ การที่ผู้ใช้งานไม่จำเป็นต้องทราบว่าตารางของข้อมูลที่เห็นนั้นในระดับ Physicals แล้วเก็บไว้ที่เดียวกันหรือมีการแบ่งแยกข้อมูลไปเก็บไว้ในหลายๆที่ รวมทั้งไม่จำเป็นต้องทราบว่าแต่ละส่วนของตารางที่ได้แบ่งแยกนั้นเก็บอยู่ที่ใดบ้าง ซึ่งถือเป็น Transparency ระดับสูงสุดหรือเทียบเท่ากับ Global Schema ตัวอย่างเช่น ข้อมูลของพนักงานบริษัท ซึ่งข้อมูลของพนังงานทั่วไปถูกเก็บไว้ในเซิร์ฟเวอร์ของสำนักงานในประเทศไทย แต่ในขณะที่พนักงานที่มีตำแหน่งเป็นผู้บริหารจะถูกเก็บอยู่ในเซิร์ฟเวอร์ของบริษัทแม่ที่อยู่ในต่างประเทศ เป็นต้น

ใน Oracle นั้นยังไม่รองรับการทำ Fragmentation Transparency โดยตรง ต้องมีการสร้าง View ขึ้นมาครอบข้อมูลที่ถูกเก็บไว้ตามที่ต่างๆ แทน แล้วจึงให้ผู้ใช้งานที่ต้องการเรียกดูข้อมูลเรียกดูข้อมูลผ่าน View ที่สร้างขึ้นมาแทน ซึ่งรายละเอียดของ View จะมีดังนี้

4.2.1 VIEW

เปรียบเสมือนตารางในฐานข้อมูล ซึ่งข้อมูลใน View อาจจะได้มาจากการ query จาก 1 ตาราง หรือหลายตาราง หรืออาจจะเป็นการ query มาจาก View ก็ได้ ในแง่ของผู้ใช้ ข้อมูลใน View จะปรากฏเหมือนตารางจริง แต่สิ่งที่แตกต่างกันระหว่าง View กับตารางก็คือ View จะไม่มีการเก็บข้อมูลลงในฐานข้อมูล เพราะข้อมูลที่ผู้ใช้เห็นจะได้มาจากการ query

ตัวอย่าง ตาราง employees จะมีคอลัมน์ดังที่ได้แสดงในรูป ถ้าต้องการกำหนดให้ ผู้ใช้งานบางคนสามารถมองเห็นข้อมูลเฉพาะคอลัมน์ที่จำเป็นเท่านั้น สามารถที่จะใช้การสร้าง View ที่เลือกเฉพาะคอลัมน์ที่ต้องการ จากตาราง employees เพื่อซ่อนข้อมูลที่ไม่ต้องการให้ผู้ใช้ทุกคนเข้าถึงได้



รูปที่ 4.1 แสดงตัวอย่างของ View

เพราะว่า View นั้นได้สืบทอดลักษณะมาจากตาราง ดังนั้น View จึงมีความสามารถหลายๆ อย่าง ที่ตารางสามารถทำได้ เช่น สามารถที่จะทำการ INSERT UPDATE DELETE ได้ ถ้า View นั้นถูกสร้างขึ้นมาอยู่ในเงื่อนไขที่ทำให้สามารถใช้คำสั่งดังกล่าวได้ และจากการทำคำสั่ง INSERT UPDATE DELETE กับ View การเปลี่ยนแปลงจะเกิดขึ้นกับตารางที่ View ดึงข้อมูลมา และการทำคำสั่งดังกล่าวต้องเป็นไปตาม integrity constraint และ Trigger บนตารางนั้นด้วย

ลักษณะคำสั่งในการสร้าง View คือ

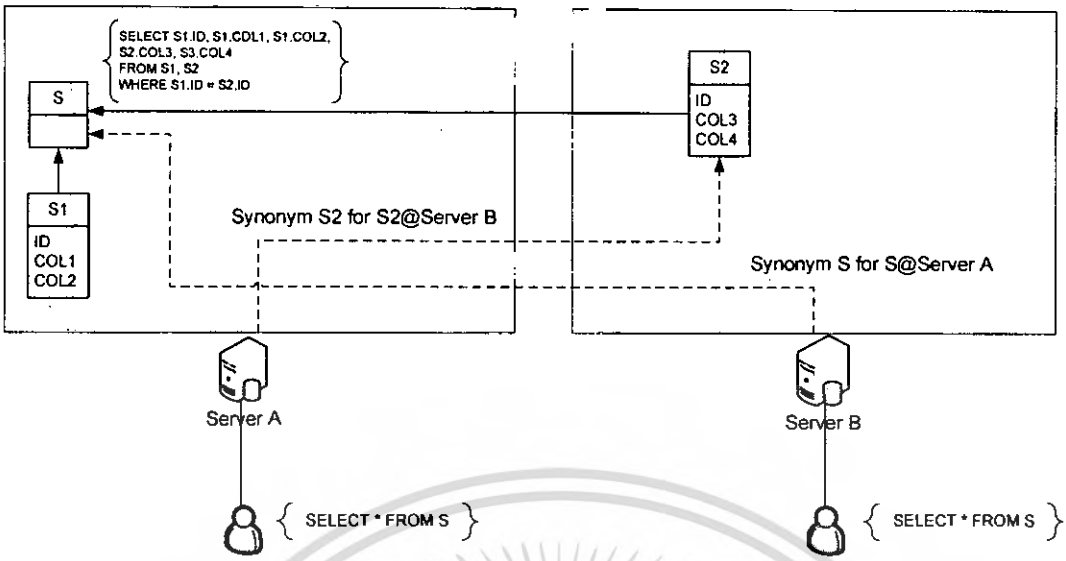
```
CREATE [OR REPLACE] VIEW <view_name> [<column name(s)>] AS
<Query [WITH CHECK OPTION [constraint <name>]]>
```

เนื่องจากสามารถจะใช้คำสั่ง INSERT UPDATE DELETE ได้กับเฉพาะ View ที่สร้างมาจากเงื่อนไขที่ถูกต้องเท่านั้น ต่อไปจะเป็นคุณสมบัติของ View ที่สามารถทำการเปลี่ยนแปลงข้อมูลได้

- ต้องไม่มีการใช้คำสั่ง DISTINCT ซึ่งก็คือ ข้อมูลแถวที่ซ้ำกันจะต้องไม่ถูกลบออกไป
- ในคำสั่ง SELECT ต้องมีการกำหนดชื่อคอลัมน์ที่ไม่ใช่ค่าคงที่ ที่ไม่มีนิพจน์ทางคณิตศาสตร์ ที่ไม่มีฟังก์ชัน Aggreate (เช่น SUM, AVG, MIN เป็นต้น) และที่ไม่ซ้ำกัน
- ในประโยค FROM ต้องระบุเฉพาะตารางเดียวเท่านั้น ดังนั้นจะไม่มีการเชื่อมตาราง (JOIN) และต้องไม่มีการใช้ตัวกระทำเซต UNION, INTERSECT และ MINUS
- ประโยค WHERE ต้องไม่มีการ Query ย่อยอีก ซึ่งอ้างอิงกับตารางในประโยค FROM ต้องไม่มีการการใช้ประโยค GROUP BY หรือ HAVING ในการกำหนด Query
- นอกจากนั้นแล้ว ข้อมูลที่ได้ทำการ INSERT หรือ UPDATE ต้องไม่ละเมิด Integrity constraint ของตารางหลัก

จากคุณสมบัติดังกล่าวจะเห็นได้ว่า View ที่สร้างมาจากการ JOIN จะไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้ ถ้าต้องการให้ View ที่สร้างมาจากการ JOIN สามารถทำการเปลี่ยนแปลงข้อมูลได้ คุณสมบัติของ View นั้นต้องตรงตามเงื่อนไขต่อไปนี้

- คำสั่ง INSERT UPDATE DELETE ต้องมีผลเฉพาะตารางเดียวเท่านั้น
- สำหรับคำสั่ง INSERT และ UPDATE, View จะต้องไม่มีคำสั่ง WITH CHECK OPTION อยู่ด้วย และข้อมูลที่ทำการ INSERT เข้าไปต้องมีลักษณะของตาราง key-preserved ซึ่งตาราง key-preserved คือตารางที่มี Primary key และ Unique key แล้วทำให้ View ที่สร้างขึ้นนั้นยังคงลักษณะของ Primary key และ Unique key ไว้ได้
- สำหรับคำสั่ง DELETE ถ้าเป็นการ join ที่มาจากตาราง key-preserved มากกว่า 1 ตาราง ฐานข้อมูล Oracle จะทำการลบตารางแรกที่ประกาศไว้ที่ประโยค FROM ไม่ว่า View นั้นจะถูกสร้างขึ้นมาด้วยคำสั่ง WITH CHECK OPTION หรือไม่



รูปที่ 4.2 ตัวอย่างการทำ Fragmentation Transparency โดยการใช้ View

การสร้าง View ที่ทำหน้าที่รวมข้อมูลจากที่ต่างๆ มาไว้ด้วยกันนั้นทำโดยการใช้คำสั่งในการรวมข้อมูลจากภาษา SQL ที่ใช้เรียกดูข้อมูล ซึ่งจะได้ View ตามการแบ่ง Fragmentation ดังหัวข้อถัดไป

View ที่ได้การแบ่ง Fragmentation ทั้ง 3 แบบ

1. View ที่เกิดจากการแบ่งข้อมูลตามแนวคอลัมน์ (Vertical Fragmentation)

การสร้าง View แบบนี้สามารถทำได้โดยการใช้คำสั่ง JOIN แบบ FULL OUTER JOIN เพราะจะได้ข้อมูลครบในทุกๆ Fragment และคอลัมน์ที่จะถูกใช้เป็นเงื่อนไขในการ JOIN ก็คือคอลัมน์ที่เป็น primary key

2. View ที่เกิดจากการแบ่งข้อมูลในลักษณะของการแบ่งแถว (Horizontal Fragmentation)

การสร้าง View แบบนี้สามารถทำได้โดยการใช้คำสั่ง UNION โดยที่คอลัมน์ทุกคอลัมน์ของตารางที่นำมา UNION ต้องตรงกันทุกคอลัมน์

3. View ที่เกิดจากการแบ่งข้อมูลแบบผสม (Mixed Fragmentation)

การสร้าง View แบบนี้สามารถทำได้โดยการใช้คำสั่ง UNION และ JOIN แบบ FULL OUTER JOIN ร่วมกัน ในการสร้าง View ซ้อน View อีกขั้นหนึ่ง

ซึ่งหัวข้อต่อไปจะเป็นการวิธีการทำ Fragmentation Transparency จาก View ทั้ง 3 ประเภท

วิธีการทำ Fragmentation Transparency จาก View ที่ได้จากการทำ Fragmentation

1. การทำ Fragmentation Transparency ที่เกิดจาก View ที่ได้จากการทำ Vertical Fragmentation

กำหนดให้มีฐานข้อมูลอยู่ 2 ที่ คือ ungrad06.ce.kmitl.ac.th และ ungrad15.ce.kmitl.ac.th และต้องการสร้าง View ชื่อ EMP_JOIN ที่เกิดจากการแบ่งข้อมูลตามแนวคอลัมน์ จาก 2 ตารางที่อยู่ต่าง Site คือ EMP_JOIN1 เป็นของ ungrad06.ce.kmitl.ac.th และ EMP_JOIN2 เป็นของ ungrad15.ce.kmitl.ac.th และกำหนดให้ทำการสร้าง View อยู่ที่ ungrad06.ce.kmitl.ac.th

ขั้นตอนการสร้าง View ที่ได้จากการทำ Vertical Fragmentation

1. สร้าง SYNONYM ที่เครื่อง ungrad06.ce.kmitl.ac.th ไปยังตาราง EMP_JOIN2 ของเครื่อง ungrad15.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_JOIN2 FOR
SCOTT.EMP_JOIN2@ORCL.UNGRAD15.CE.KMITL.AC.TH;
```

2. สร้าง VIEW EMP_JOIN โดยทำการ JOIN แบบ FULL OUTER JOIN ด้วย primary key ซึ่งก็คือ EMPNO จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE VIEW EMP_JOIN (EMPNO, ENAME, JOB, MGR, HIREDATE,
SAL, COMM, DEPTNO) AS
(SELECT E1.EMPNO, E1.ENAME, E1.JOB,E1.MGR,E2.HIREDATE, E2.SAL, E2.COMM,
E1.DEPTNO
FROM EMP_JOIN1 E1 FULL OUTER JOIN EMP_JOIN2 E2
ON E1.EMPNO = E2.EMPNO);
```

3. สร้าง SYNONYM ที่เครื่อง ungrad15.ce.kmitl.ac.th แทน VIEW EMP_JOIN ที่ ungrad06.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_JOIN FOR
SCOTT.EMP_JOIN@ORCL.UNGRAD06.CE.KMITL.AC.TH;
```

2. การทำ Fragmentation Transparency ที่เกิดจาก View ที่ได้จากการทำ Horizontal Fragmentation

กำหนดให้มีฐานข้อมูลอยู่ 2 ที่ คือ ungrad06.ce.kmitl.ac.th และ ungrad15.ce.kmitl.ac.th และต้องการสร้าง View ชื่อ EMP_UNION ที่เกิดจากการลักษณะของการแบ่งแถว จาก 2 ตารางที่อยู่ต่าง Site คือ EMP_UNION1 เป็นของ ungrad06.ce.kmitl.ac.th และ EMP_UNION2 เป็นของ ungrad15.ce.kmitl.ac.th และกำหนดให้ทำการสร้าง View อยู่ที่ ungrad06.ce.kmitl.ac.th

ขั้นตอนการสร้าง View ที่ได้จากการทำ Horizontal Fragmentation

1. สร้าง SYNONYM ที่เครื่อง ungrad06.ce.kmitl.ac.th ไปยังตาราง EMP_UNION2 ของเครื่อง ungrad15.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_UNION2 FOR
SCOTT.EMP_UNION2@ORCL.UNGRAD15.CE.KMITL.AC.TH;
```

2. สร้าง VIEW EMP_UNION โดยทำการ UNION ตารางทั้ง 2 ซึ่งจะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE VIEW EMP_UNION (EMPNO, ENAME, JOB, MGR, HIREDATE,
SAL, COMM, DEPTNO) AS
(SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO FROM
EMP_UNION1)
UNION
(SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO FROM
EMP_UNION2);
```

3. สร้าง SYNONYM ที่ ungrad15.ce.kmitl.ac.th แทน VIEW EMP_UNION ที่ ungrad06.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_UNION FOR
SCOTT.EMP_UNION@ORCL.UNGRAD06.CE.KMITL.AC.TH;
```

3. การทำ Fragmentation Transparency ที่เกิดจาก View ที่ได้จากการทำ Mixed Fragmentation

กำหนดให้มีฐานข้อมูลอยู่ 2 ที่ คือ ungrad06.ce.kmitl.ac.th และ ungrad15.ce.kmitl.ac.th และมีตารางอยู่ 3 ตารางคือ EMP_MIXED1_UNION1 EMP_MIXED1_UNION2 และ EMP_MIXED2 โดยที่ EMP_MIXED1_UNION1 และ EMP_MIXED2 จะอยู่ที่ ungrad06.ce.kmitl.ac.th ส่วน EMP_MIXED1_UNION2 จะอยู่ที่ ungrad15.ce.kmitl.ac.th

จะทำ Mixed Fragmentation โดยที่จะสร้าง View ชื่อ EMP_MIXED1 ที่ได้มาจากการทำ Horizontal Fragmentation จาก EMP_MIXED1_UNION1 กับ EMP_MIXED1_UNION2

จากนั้นนำ View ที่ชื่อ EMP_MIXED1 มาทำ Vertical Fragmentation กับตาราง EMP_MIXED2 อีกครั้งหนึ่ง และกำหนดให้สร้าง View ที่ ungrad06.ce.kmitl.ac.th

ขั้นตอนการสร้าง View ที่ได้จากการทำ Mixed Fragmentation

1. สร้าง SYNONYM ที่เครื่อง ungrad06.ce.kmitl.ac.th ไปยังตาราง EMP_MIXED1_UNION2 ของเครื่อง ungrad15.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_MIXED1_UNION2 FOR
SCOTT. EMP_MIXED1_UNION2 @ORCL.UNGRAD15.CE.KMITL.AC.TH;
```

2. สร้าง VIEW EMP_MIXED1 โดยทำการ UNION ตารางทั้ง 2 ซึ่งจะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE VIEW EMP_MIXED1 (EMPNO, ENAME, JOB, MGR, DEPTNO) AS
(SELECT EMPNO, ENAME, JOB, MGR, DEPTNO FROM EMP_MIXED1_UNION1
UNION
SELECT EMPNO, ENAME, JOB, MGR, DEPTNO FROM EMP_MIXED1_UNION2
);
```

3. สร้าง SYNONYM ที่ ungrad15.ce.kmitl.ac.th แทน VIEW EMP_MIXED1 ที่ ungrad06.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_MIXED1 FOR SCOTT.
EMP_MIXED1@ORCL.UNGRAD06.CE.KMITL.AC.TH;
```

4. สร้าง View EMP_MIXED จาก EMP_MIXED1 และ EMP_MIXED2 ซึ่งเป็นการทำ Fragmentation แบบ Vertical Fragmentation ซึ่งจะใช้การ JOIN แบบ FULL OUTER JOIN ในการสร้าง จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE VIEW EMP_MIXED (EMPNO, ENAME, JOB, MGR, HIREDATE,
SAL, COMM, DEPTNO) AS
(SELECT E1.EMPNO, E1.ENAME, E1.JOB,E1.MGR, E2.HIREDATE, E2.SAL, E2.COMM,
E1.DEPTNO
FROM EMP_MIXED1 E1 FULL OUTER JOIN EMP_MIXED2 E2
ON E1.EMPNO = E2.EMPNO);
```

5. สร้าง SYNONYM ที่ ungrad15.ce.kmitl.ac.th แทน VIEW EMP_MIXED ที่ ungrad06.ce.kmitl.ac.th จะได้คำสั่ง SQL ดังนี้

```
CREATE OR REPLACE SYNONYM EMP_MIXED FOR
SCOTT. EMP_MIXED@ORCL.UNGRAD06.CE.KMITL.AC.TH;
```

ในการสร้าง View ทั้งจากการ JOIN แบบ FULL OUTER JOIN และ UNION นั้น จะไม่สามารถทำการเปลี่ยนแปลงข้อมูลได้ คือ ไม่สามารถ INSERT UPDATE และ DELETE โดยปัญหาดังกล่าวนี้ สามารถแก้ได้โดยการเขียนโปรแกรมย่อย Trigger มาควบคุม ซึ่งรายละเอียดของ Trigger จะทำการอธิบายในหัวข้อถัดไป

4.2.2 Trigger

Trigger คือ โปรแกรมย่อย ซึ่งถูกเขียนโดยใช้ภาษา PL/SQL หรือภาษา JAVA ที่สร้างขึ้นเพื่อตอบสนองต่อเหตุการณ์บางเหตุการณ์ ตัวอย่างเช่น เมื่อมีคำสั่งที่ต้องการเปลี่ยนแปลงค่าใน table หรือ view, เมื่อฐานข้อมูลมีความผิดพลาดเกิดขึ้น หรือ เมื่อผู้ใช้งานทำการเข้าใช้งานระบบฐานข้อมูล เป็นต้น

สามารถแบ่งประเภทเหตุการณ์ที่อาจกำหนด Trigger ไว้ได้ดังนี้

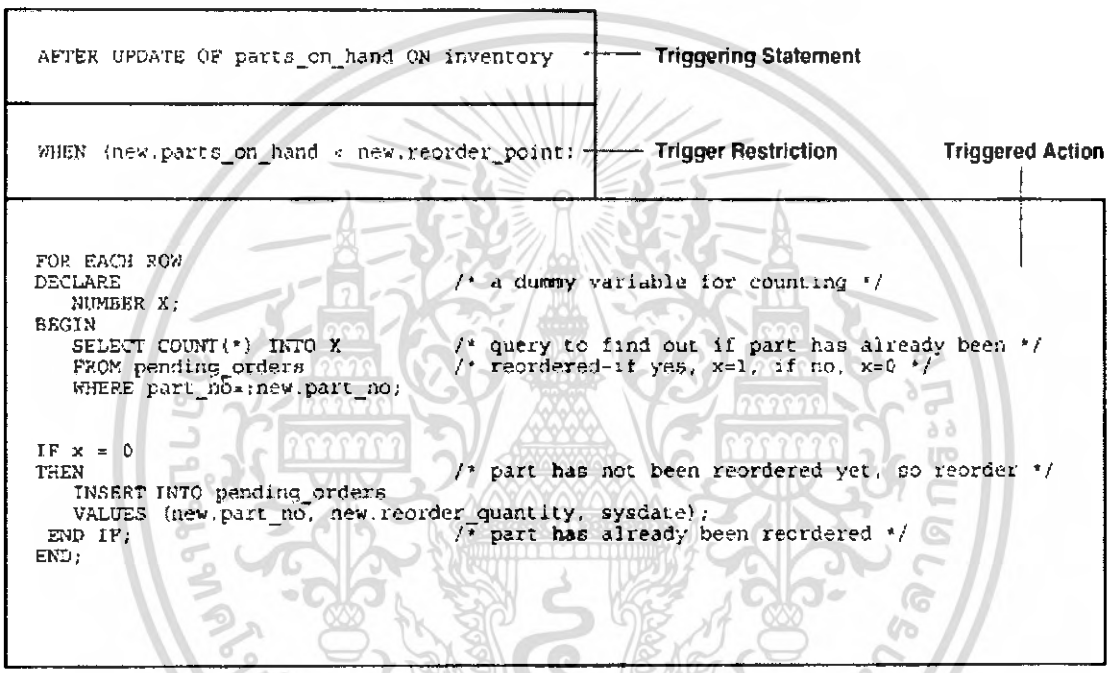
- เมื่อมีคำสั่ง DML statements กับ table หรือ view อันได้แก่ INSERT, UPDATE และ DELETE
- เมื่อมีคำสั่ง DDL statements เช่น CREATE, ALTER, DROP เป็นต้น
- เมื่อเกิด System event เช่น มีการ startup, shutdown ระบบฐานข้อมูล หรือมี error message เกิดขึ้น

- เมื่อเกิด User event เช่น ผู้เข้าใช้งานทำการ logon หรือ logoff

โครงสร้างพื้นฐานของ Trigger ประกอบด้วย 3 ส่วนด้วยกันคือ

- ส่วนที่เป็นเหตุการณ์ หรือคำสั่งที่ทำให้ Trigger ทำงาน (Triggering event or statement)
- ส่วนที่เป็นเงื่อนไขของ Trigger (Trigger restriction)
- ส่วนที่เป็นการทำคำสั่งของ Trigger (Trigger action)

REORDER Trigger



รูปที่ 4.3 ตัวอย่างโครงสร้างพื้นฐานของ Trigger

ส่วนที่เป็นเหตุการณ์ หรือคำสั่งที่ทำให้ Trigger ทำงาน (Triggering event or statement)

Trigger event หรือ Trigger statement คือ คำสั่ง SQL, เหตุการณ์ที่เกิดจากระบบฐานข้อมูล, เหตุการณ์ที่เกิดจากผู้ใช้งาน ที่มีผลทำให้ Trigger มีการทำงาน ยกตัวอย่างได้ดังต่อไปนี้

- คำสั่ง INSERT, UPDATE, DELETE ที่กระทำกับตาราง (หรือวิว ในบางกรณี)
- คำสั่ง CREATE, ALTER, DROP ที่กระทำกับ SCHEMA
- ระบบฐานข้อมูลทำการ startup หรือ ทำการ shutdown
- มี error message เกิดขึ้น
- ผู้ใช้งานทำการ logon หรือ logoff

ตัวอย่างที่เป็น Trigger statement

```
... UPDATE OF parts_on_hand ON inventory ...
```

ส่วนที่เป็นเงื่อนไขของ Trigger (Trigger restriction)

ส่วนที่เป็นเงื่อนไขของ Trigger (Trigger restriction) จะเป็นส่วนคอยตรวจสอบว่าจะทำคำสั่งใน Trigger action หรือไม่ โดยที่คำสั่งใน Trigger action จะทำงานก็ต่อเมื่อเงื่อนไขเป็นจริงเท่านั้น

ตัวอย่างการกำหนด Trigger restriction

```
new.parts_on_hand < new.reorder_point
```

ซึ่งเงื่อนไขจะเป็นจริงก็ต่อเมื่อ ค่า reoder_point มากกว่าค่า parts_on_hand

ส่วนที่เป็นการทำคำสั่งของ Trigger (Trigger action)

ส่วนนี้จะเป็นส่วนที่เป็นคำสั่ง SQL ที่จะทำงานเมื่อมี Triggering statement ขึ้น หรือเมื่อเงื่อนไขใน Trigger restriction เป็นจริง

นอกจากนั้น Trigger action ยังสามารถ

- บรรจุคำสั่ง SQL, PL/SQL หรือคำสั่งภาษา Java ได้
- กำหนดตัวแปร, ค่าคงที่, เคอร์เซอร์ และส่วนตรวจจับข้อผิดพลาด ที่เป็นภาษา PL/SQL ได้
- เรียกโปรแกรมย่อยให้ทำงานได้

และถ้าเป็น Row Trigger หากมีคำสั่งใน Trigger action ที่ต้องเข้าถึงค่าในคอลัมน์ใดๆ สามารถที่จะใช้ new แทนค่าข้อมูลที่เป็นค่าใหม่ และ old แทนค่าข้อมูลที่เป็นค่าเก่าได้

ประเภทของ Trigger

Trigger แบ่งออกได้เป็น 4 กลุ่มด้วยกัน คือ

- Row Trigger และ Statement Trigger
- BEFORE และ AFTER Trigger
- INSTEAD OF Trigger
- Trigger ที่เกิดจาก System Event และ User Event

Row Trigger และ Statement Trigger

Row Trigger

Trigger ประเภทนี้จะกระทำกับข้อมูลทีละแถว เช่นถ้าต้องการ Update ตารางชื่อ books ที่มี 1000 แถว และใช้คำสั่งต่อไปนี้ในส่วนที่เป็นการกระทำของ Trigger

```
UPDATE books SET title = UPPER (title);
```

ซึ่งคำสั่งนี้จะต้องกระทำกับข้อมูล 1000 แถว และหากได้กำหนด Trigger เป็นแบบ Row Trigger จะทำให้ Trigger ทำงานทั้งหมด 1000 ครั้งด้วย

Statement Trigger

Trigger ประเภทนี้จะกระทำกับข้อมูลทุกแถว เช่นถ้ามีคำสั่ง Delete ที่มีผลกับข้อมูลหลายๆ แถว Trigger ประเภทนี้จะทำงานเพียงครั้งเดียวเท่านั้นก็สามารถ Delete ข้อมูลได้ครบ

BEFORE และ AFTER Trigger

ในการสร้าง Trigger สามารถที่จะกำหนดได้ว่าจะให้ Trigger ทำงานก่อนหรือหลัง Triggering Statement โดยการใช้คำสั่ง BEFORE หรือ AFTER และทั้ง 2 แบบ ใช้ได้ทั้ง Row Trigger และ Statement Trigger

BEFORE และ AFTER Trigger ที่สร้างขึ้นเพื่อตอบสนองต่อคำสั่ง DML statements จะสามารถกำหนดได้เฉพาะกับตารางเท่านั้น สำหรับคำสั่ง DDL statement จะกำหนดได้เฉพาะตัวฐานข้อมูลและ schema ไม่สามารถกำหนดให้ทำงานกับตารางได้

จากกลุ่มของTrigger ทั้งRow Trigger, Statement Trigger และ BEFORE, AFTER Trigger สามารถที่จะจำแนกกรณีต่างๆ ได้ดังนี้

- BEFORE statement trigger

  Trigger action จะทำงานก่อนที่จะทำคำสั่งที่เป็น Triggering statement

- BEFORE row trigger

  Trigger action จะทำงานก่อนที่จะมีการเปลี่ยนแปลงแต่ละแถวที่เกิดจากคำสั่งที่เป็น Triggering statement สามารถใช้ในการตรวจสอบ Integrity Constraint ได้ โดยที่ส่วนที่เป็นเงื่อนไขของ Trigger ต้องไม่ถูกละเมิดด้วย

- AFTER statement trigger

  Trigger action จะทำงานหลังจากทำคำสั่งที่เป็น Triggering statement

- AFTER row trigger

  Trigger action จะทำงานหลังจากมีการเปลี่ยนแปลงข้อมูลแต่ละแถวซึ่งมาจากคำสั่งที่เป็น Triggering statement สามารถใช้ในการตรวจสอบ Integrity Constraint ได้ โดยที่ส่วนที่เป็นเงื่อนไขของ Trigger ต้องไม่ถูกละเมิดด้วย

นอกจากนั้นยังสามารถสร้าง Trigger หลายๆ Trigger ซึ่งทำงานกับคำสั่งเดียวบนตาราง ใดๆ ได้ ตัวอย่างเช่น สามารถกำหนด BEFORE statement Trigger ไว้ 2 ลักษณะการทำงาน สำหรับคำสั่ง UPDATE บนตาราง employee

INSTEAD OF Trigger

Instead of Trigger เป็น Trigger ประเภทหนึ่ง ซึ่งทำให้สามารถ Insert, Update, delete ผ่าน View ที่ไม่สามารถทำการ Update ข้อมูลได้ อย่างเช่น view ที่เกิดจาก Union หรือ จากการ Join ใน บางกรณี Trigger ประเภทนี้จะไม่เหมือนแบบอื่นคือ จะมีการตรวจสอบว่ามีคำสั่ง INSERT, UPDATE, DELETE ผ่าน View หรือไม่ ถ้ามีก็เข้าไปทำตามสั่งที่เขียนไว้ใน Trigger body แทน เพราะฉะนั้นเราต้องทำการเขียนคำสั่ง INSERT, UPDATE, DELETE ดารางที่ต้องการเอง ซึ่ง ลักษณะคำสั่งในการสร้าง Instead of Trigger คือ

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
INSTEAD OF {INSERT OR UPDATE OR DELETE}
ON <view_name>
[REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]]
[FOR EACH ROW [WHEN (<trigger_condition>)]]
<trigger_body>
```

Trigger ที่เกิดจาก System Event และ User Event

Trigger ที่เกิดจาก System event สามารถกำหนดไว้ที่ระดับฐานข้อมูล หรือระดับ SCHEMA ได้ สำหรับ Trigger ที่เกิดจาก DDL statement หรือจากการที่ User ทำการ logon และ logoff นั้นสามารถกำหนดได้ที่ระดับฐานข้อมูลหรือระดับ SCHEMA และ Trigger ที่เกิดจากคำสั่ง DML เช่น INSERT UPDATE DELETE สามารถกำหนดไว้ที่ระดับตารางหรือวิว
ถ้าTrigger ใดกำหนดไว้ที่ระดับฐานข้อมูลจะมีผลกับทุกผู้ใช้งานและ ถ้า Trigger ใดที่กำหนดไว้ที่ ระดับ SCHEMA หรือดารางก็จะมีผลกับ SCHEMA หรือตารางที่ถูกกำหนดไว้เท่านั้น

System Event

Trigger ที่เกิดจาก System event สามารถกำหนดไว้ที่ระดับฐานข้อมูล หรือระดับ SCHEMA ได้ โดยที่ถ้าเป็น Trigger ที่สร้างมาจากกรณีที่ฐานข้อมูลทำการ startup หรือ shutdown จะสามารถกำหนดไว้ได้ที่ระดับฐานข้อมูล ส่วน Trigger ที่สร้างมากรณีที่เกิดข้อผิดพลาดขึ้นจะ สามารถกำหนดไว้ได้ที่ระดับฐานข้อมูลและระดับ SCHEMA

- STARTUP triggers จะทำงานเมื่อฐานข้อมูลถูก open โดย instance

- SHUTDOWN triggers จะทำงานก่อนที่จะทำการ shutdown ตัว instance ถ้าเป็นการ shutdown ที่ไม่ปกติ Trigger ชนิดนี้จะไม่ทำงาน
- SERVERERROR triggers จะทำงานเมื่อเกิดข้อผิดพลาดที่กำหนดไว้ขึ้น และอาจเป็นกรณีข้อผิดพลาดใดๆ ที่ไม่ได้ระบุไว้ก็ได้
- DB_ROLE_CHANGE triggers จะทำงานเมื่อ role transition เกิดขึ้นในส่วนปรับแต่งค่าของ Data Guard

ลักษณะคำสั่งของการสร้าง System Event Trigger

```
CREATE [OR REPLACE] TRIGGER trigger name
{BEFORE | AFTER} {System Event} ON {DATABASE | SCHEMA}
DECLARE
Variable declarations
BEGIN
... some code...
END;
```

User Event

User event ทำให้ Trigger ทำงานได้เมื่อผู้ใช้งานทำการ logon หรือว่า logoff หรือมีคำสั่งที่เป็น DDL เช่น CREATE DROP หรือคำสั่ง DML อย่างเช่น INSERT UPDATE DELETE

Trigger ที่เกิดจาก LOGON Event และ LOGOFF Event

Logon และ longoff Triger สามารถกำหนดไว้ได้ที่ระดับฐานข้อมูล และระดับ schema

- LOGON trigger จะทำงานเมื่อผู้ใช้ทำการ logon อย่างสมบูรณ์
- LOGOFF trigger จะทำงานเมื่อผู้ใช้เริ่มทำการ logoff

ลักษณะคำสั่งของการสร้าง LOGON Event, LOGOFF Event Trigger

```
CREATE [OR REPLACE] TRIGGER trigger name
{BEFORE | AFTER} {LOGON Event, LOGOFF Event} ON {DATABASE | SCHEMA}
DECLARE
Variable declarations
BEGIN
... some code...
END;
```

Triggers ที่เกิดจาก DDL Statement

DDL trigger สามารถที่จะกำหนดไว้ที่ระดับฐานข้อมูล หรือที่ระดับ schema ซึ่งได้แก่

- BEFORE CREATE และ AFTER CREATE trigger จะทำงานเมื่อมี schema ถูกสร้างขึ้น
- BEFORE ALTER และ AFTER ALTER trigger จะทำงานเมื่อ schema มีการแก้ไขเกิดขึ้น

BEFORE DROP และ AFTER DROP trigger จะทำงานเมื่อ schema ถูกทำลาย

ลักษณะคำสั่งของการสร้าง DDL Statement Trigger

```
CREATE [OR REPLACE] TRIGGER trigger name
{BEFORE | AFTER} {DDL event} ON {DATABASE | SCHEMA}
[WHEN (...)]
DECLARE
Variable declarations
BEGIN
... some code...
END;
```

Triggers ที่เกิดจาก DML Statement

Triggers ที่เกิดจาก DML Statement จะสามารถกำหนดไว้ได้ที่ระดับ table และอาจจะเป็นได้ทั้ง BEFORE หรือ AFTER trigger ซึ่งจะมีการทำงานกับแต่ละแถวตามคำสั่ง DML ที่ได้กำหนดไว้ และสำหรับค่าของคอลัมน์ในแต่ละแถวสามารถใช้ :OLD ตามด้วยชื่อคอลัมน์เอาไว้อ้างถึงข้อมูลค่าเก่าได้ ส่วน :NEW ตามด้วยชื่อคอลัมน์จะเอาไว้อ้างถึงข้อมูลค่าใหม่

Triggers ที่เกิดจาก DML Statement ได้แก่

- BEFORE INSERT และ AFTER INSERT trigger จะทำงานกับแต่ละแถวที่ถูก INSERT ลงในตาราง
- BEFORE UPDATE และ AFTER UPDATE trigger จะทำงานกับแต่ละแถวที่ถูก UPDATE ในตาราง
- BEFORE DELETE และ AFTER DELETE trigger จะทำงานกับแต่ละแถวที่ถูก DELELTE ออกจากตาราง

ลักษณะคำสั่งของการสร้าง DML Statement Trigger

```
CREATE [OR REPLACE] TRIGGER trigger name
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE | UPDATE OF column list} ON table name
[FOR EACH ROW]
[WHEN (...)]
[DECLARE ... ]
BEGIN
... executable statements ...
[EXCEPTION ... ]
END [trigger name];
```

การเขียน INSTEAD OF TRIGGER ช่วยให้สามารถ INSERT, UPDATE, DELETE กับ View ได้

เพราะว่า View ที่สร้างขึ้นมายังเป็นการทำ Fragmentation Transpareney ที่ไม่สมบูรณ์ คือ ผู้ใช้งานจะทำได้เพียง SELECT ข้อมูลเท่านั้น ไม่สามารถทำการ INSERT, UPDATE, DELETE กับ View ได้ ซึ่งต้องมีการเขียน Instead of Trigger เพื่อช่วยในการ INSERT UPDATE DELETE ข้อมูลบน View สามารถแบ่งออกได้เป็น 3 กรณีด้วยกันคือ

1. Instead of Trigger กับView ที่เกิดจากการแบ่งข้อมูลตามแนวคอลัมน์ (Vertical Fragmentation)

สมมุติว่าได้ทำการสร้าง View ชื่อ EMP_JOIN ที่เกิดจากการแบ่งข้อมูลตามแนวคอลัมน์จาก 2 ตาราง คือ EMP_JOIN1 ที่ ungrad06.ce.kmitl.ac.th, EMP_JOIN2 ที่ ungrad15.ce.kmitl.ac.th และ Synonym สำหรับ EMP_JOIN ที่ ungrad15.ce.kmitl.ac.th ไว้แล้วจากนั้นต้องเขียน Trigger ไว้ที่ ungrad06.ce.kmitl.ac.th ดังต่อไปนี้

- กรณีที่เป็นการ INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER IOT_EMP_JOIN_INSERT INSTEAD OF INSERT ON
EMP_JOIN
FOR EACH ROW
BEGIN
 BEGIN
  INSERT INTO EMP_JOIN1 (EMPNO, ENAME, JOB, MGR, DEPTNO)
  VALUES (:NEW.EMPNO, :NEW.ENAME, :NEW.JOB, :NEW.MGR
, :NEW.DEPTNO);
```

```
END;
BEGIN
   INSERT INTO EMP_JOIN2 (EMPNO, ENAME, HIREDATE, SAL, COMM)
VALUES (:NEW.EMPNO, :NEW.ENAME, :NEW.HIREDATE, :NEW.SAL
, :NEW.COMM);
 END;
  EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
--  do not insert if a duplicate EMP exists
   RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF UPDATE

```
CREATE OR REPLACE TRIGGER IOT_EMP_JOIN_UPDATE INSTEAD OF UPDATE ON
EMP_JOIN
FOR EACH ROW
BEGIN
-- Updates to EMP table.
BEGIN
IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL
AND :old.ename !=        :new.ename) OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.job IS NULL AND :new.job IS NOT NULL) OR
(:old.job IS NOT NULL AND :new.job IS NOT NULL AND :old.job != :new.job) OR
(:old.job IS NOT NULL AND :new.job IS NULL) OR
(:old.mgr IS NULL AND :new.mgr IS NOT NULL) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NOT NULL AND :old.mgr != :new.mgr) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NULL) OR
(:old.deptno IS NULL AND :new.deptno IS NOT NULL) OR
```

```
(:old.deptno IS NOT NULL AND :new.deptno IS NOT NULL AND :old.deptno != :new.deptno)
OR
(:old.deptno IS NOT NULL AND :new.deptno IS NULL)
   THEN
    UPDATE EMP_JOIN1
    SET empno = :new.empno
    ,ENAME = :new.ename
    ,JOB = :new.job
    ,MGR = :new.mgr
    ,DEPTNO = :new.deptno
    WHERE EMPNO = :old.empno;
   END IF;
END;

BEGIN
   IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL
AND :old.ename !=       :new.ename) OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.sal IS NULL AND :new.sal IS NOT NULL) OR
(:old.sal IS NOT NULL AND :new.sal IS NOT NULL AND :old.sal != :new.sal) OR
(:old.sal IS NOT NULL AND :new.sal IS NULL) OR
(:old.comm IS NULL AND :new.comm IS NOT NULL) OR
(:old.comm IS NOT NULL AND :new.comm IS NOT NULL AND :old.comm != :new.comm)
OR (:old.comm IS NOT NULL AND :new.comm IS NULL)
   THEN
    UPDATE EMP_MIXED2
    SET empno = :new.empno
    ,ENAME = :new.ename
    ,SAL = :new.sal
    ,COMM = :new.comm
```

```
    WHERE EMPNO = :old.empno;
  END IF;
 END;

 EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF DELETE

```
CREATE OR REPLACE TRIGGER IOT_EMP_JOIN_DELETE INSTEAD OF DELETE ON
EMP_JOIN
FOR EACH ROW
BEGIN
 --delete from emp_one table
 DELETE FROM EMP_JOIN1
WHERE EMPNO = :OLD.EMPNO;
 --delete from emp_two table
 DELETE FROM EMP_JOIN2
WHERE EMPNO = :OLD.EMPNO;
END;
```

2. Instead of Trigger กับ View ที่เกิดจากการ แบ่งข้อมูลในลักษณะของการแบ่งแถว (Horizontal Fragmentation)

สมมุติว่าได้ทำการสร้าง View ชื่อ EMP_UNION ที่เกิดจากการแบ่งข้อมูลตามแถว จาก 2 ตาราง คือ EMP_UNION1 ที่ ungrad06.ce.kmitl.ac.th,.UNION2 ที่ ungrad15.ce.kmitl.ac.th และ Synonym สำหรับ EMP_JOIN ที่ ungrad15.ce.kmitl.ac.th ไว้แล้ว ต้องเขียน Trigger ไว้ที่ ungrad06.ce.kmitl.ac.th ดังต่อไปนี้

- กรณีที่เป็นการ INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER IOT_EMP_UNION_INSERT INSTEAD OF INSERT ON
EMP_UNION
FOR EACH ROW
BEGIN
  IF :new.DEPTNO <= 30 THEN
    BEGIN
INSERT INTO EMP_UNION1 (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM,
DEPTNO)
VALUES(:new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.hiredate
,:new.SAL, :new.COMM, :new.DEPTNO);
    END;
  END IF;

  IF :new.DEPTNO >= 30 THEN
    BEGIN
INSERT INTO EMP_UNION2
(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)
VALUES(:new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.hiredate
, :new.SAL, :new.COMM, :new.DEPTNO);
    END;
  END IF;
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF UPDATE

```
CREATE OR REPLACE
TRIGGER IOT_EMP_UNION_UPDATE INSTEAD OF UPDATE ON EMP_UNION
FOR EACH ROW
DECLARE
rowcnt     number;
```

```
BEGIN
IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL AND :old.ename != :new.ename)
OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.job IS NULL AND :new.job IS NOT NULL) OR
(:old.job IS NOT NULL AND :new.job IS NOT NULL AND :old.job != :new.job) OR (:old.job
IS NOT NULL AND :new.job IS NULL)  OR
(:old.sal IS NULL AND :new.sal IS NOT NULL) OR
(:old.sal IS NOT NULL AND :new.sal IS NOT NULL AND :old.sal != :new.sal) OR (:old.sal IS
NOT NULL AND :new.sal IS NULL)      OR
(:old.hiredate IS NULL AND :new.hiredate IS NOT NULL) OR
(:old.hiredate IS NOT NULL AND :new.hiredate IS NOT NULL
AND :old.hiredate != :new.hiredate) OR
(:old.hiredate IS NOT NULL AND :new.hiredate IS NULL)          OR
(:old.mgr IS NULL AND :new.mgr IS NOT NULL) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NOT NULL AND :old.mgr != :new.mgr) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NULL)          OR
(:old.deptno IS NULL AND :new.deptno IS NOT NULL) OR
(:old.deptno IS NOT NULL AND :new.deptno IS NOT NULL AND :old.deptno != :new.deptno)
OR
(:old.deptno IS NOT NULL AND :new.deptno IS NULL)
THEN
  BEGIN
    SELECT COUNT(*) INTO rowcnt FROM EMP_UNION1 WHERE
EMPNO = :OLD.EMPNO;
    IF rowcnt = 1 THEN
        DELETE EMP_UNION1 WHERE EMPNO = :OLD.EMPNO;
    END IF;

    SELECT COUNT(*) INTO rowcnt FROM EMP_UNION2 WHERE
```

```
EMPNO = :OLD.EMPNO;
      IF rowcnt = 1 THEN
       DELETE EMP_UNION2 WHERE EMPNO = :OLD.EMPNO;
      END IF;

    IF :new.deptno <= 30 THEN
 INSERT INTO EMP_UNION1 (empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES(:new.empno, :new.ename, :new.job, :new.mgr, :new.hiredate, :new.sal, :new.comm, :n
ew.deptno);
    END IF;

    IF :new.deptno >= 30 THEN
INSERT INTO EMP_UNION2 (empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES(:new.empno, :new.ename, :new.job, :new.mgr, :new.hiredate, :new.sal, :new.comm, :n
ew.deptno);
    END IF;
   END;
END IF;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
   RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF DELETE

```
CREATE OR REPLACE TRIGGER IOT_EMP_UNION_DELETE INSTEAD OF DELETE ON
EMP_UNION
FOR EACH ROW
DECLARE
  cntrow number;
BEGIN
     SELECT COUNT(*) INTO cntrow FROM EMP_UNION1 WHERE
EMPNO = :OLD.EMPNO;
```

```
    IF cntrow = 1 THEN
        DELETE EMP_UNION1 WHERE EMPNO = :OLD.EMPNO;
    END IF;

    SELECT COUNT(*) INTO cntrow FROM EMP_UNION2 WHERE
EMPNO = :OLD.EMPNO;
    IF cntrow = 1 THEN
        DELETE EMP_UNION2 WHERE EMPNO = :OLD.EMPNO;
    END IF;
END;
```

3. Instead of Trigger กับ View ที่เกิดจากการแบ่งข้อมูลแบบผสม (Mixed Fragmentation)

สมมุติว่าได้สร้าง View ชื่อ EMP_MIXED1 ที่ได้มาจากการทำ Horizontal Fragmentation จาก EMP_MIXED1_UNION1 กับ EMP_MIXED1_UNION2

และสมมุติว่าสร้าง View ที่ชื่อ EMP_MIXED ที่ได้มาจากการทำ Vertical Fragmentation ของ View ชื่อ EMP_MIXED1 กับตาราง EMP_MIXED2 แล้ว รวมทั้งสมมุติว่าได้สร้าง Synonym ของ View ทั้ง 2 Synonym ที่ ungrad15.ce.kmitl.ac.th แล้ว ดังนั้นต้องทำการสร้าง Trigger ที่ ungrad06.ce.kmitl.ac.th ดังนี้

3.1 INSTEAD OF TRIGGER ของ View EMP_MIXED1 (Horizontal Fragmentation)

- กรณี INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER IOT_ EMP_MIXED1_UNION_INSERT INSTEAD OF
INSERT ON EMP_MIXED1
FOR EACH ROW
BEGIN
  IF :new.DEPTNO <= 30 THEN
    BEGIN
INSERT INTO EMP_MIXED1_UNION1 (EMPNO, ENAME, JOB, MGR
, DEPTNO)
VALUES(:new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO);
    END;
  END IF;
```

```
  IF :new.DEPTNO >= 30 THEN
    BEGIN
INSERT INTO EMP_MIXED1_UNION2 (EMPNO, ENAME, JOB, MGR
, DEPTNO)
VALUES (:new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO);
    END;
  END IF;
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF UPDATE

```
CREATE OR REPLACE TRIGGER IOT_ EMP_MIXED1_UPDATE INSTEAD OF UPDATE
ON EMP_MIXED1
FOR EACH ROW
DECLARE
rowcnt      number;
BEGIN
IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL AND :old.ename != :new.ename)
OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.job IS NULL AND :new.job IS NOT NULL) OR
(:old.job IS NOT NULL AND :new.job IS NOT NULL AND :old.job != :new.job) OR (:old.job
IS NOT NULL AND :new.job IS NULL)  OR
 (:old.mgr IS NULL AND :new.mgr IS NOT NULL) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NOT NULL AND :old.mgr != :new.mgr) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NULL)          OR
(:old.deptno IS NULL AND :new.deptno IS NOT NULL) OR
```

```
(:old.deptno IS NOT NULL AND :new.deptno IS NOT NULL AND :old.deptno != :new.deptno)
OR
(:old.deptno IS NOT NULL AND :new.deptno IS NULL)
THEN
  BEGIN
    SELECT COUNT(*) INTO rowcnt FROM EMP_MIXED_UNION1 WHERE
EMPNO = :OLD.EMPNO;
    IF rowcnt = 1 THEN
        DELETE EMP_MIXED1_UNION1 WHERE EMPNO = :OLD.EMPNO;
    END IF;

    SELECT COUNT(*) INTO rowcnt FROM EMP_MIXED_UNION2 WHERE
EMPNO = :OLD.EMPNO;
    IF rowcnt = 1 THEN
     DELETE EMP_MIXED1_UNION2 WHERE EMPNO = :OLD.EMPNO;
    END IF;

   IF :new.deptno <= 30 THEN
 INSERT INTO EMP_MIXED1_UNION1 (empno, ename, job, mgr, deptno)
VALUES(:new.empno, :new.ename, :new.job, :new.mgr, :new.deptno);
   END IF;
   IF :new.deptno >= 30 THEN
INSERT INTO EMP_MIXED1_UNION2 (empno, ename, job, mgr, deptno)
VALUES(:new.empno, :new.ename, :new.job, :new.mgr, :new.deptno);
END IF;
  END;
END IF;
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF DELETE

```
CREATE OR REPLACE TRIGGER IOT_EMP_MIXED1_DELETE INSTEAD OF DELETE
ON EMP_MIXED1
FOR EACH ROW
DECLARE
  cntrow number;
BEGIN
     SELECT COUNT(*) INTO cntrow FROM EMP_MIXED1_UNION1 WHERE
EMPNO = :OLD.EMPNO;
     IF cntrow = 1 THEN
          DELETE EMP_MIXED1_UNION1 WHERE EMPNO = :OLD.EMPNO;
     END IF;

     SELECT COUNT(*) INTO cntrow FROM EMP_UNION2 WHERE
EMPNO = :OLD.EMPNO;
     IF cntrow = 1 THEN
          DELETE EMP_MIXED1_UNION2 WHERE EMPNO = :OLD.EMPNO;
     END IF;
END;
```

3.2 INSTEAD OF TRIGGER ของ View EMP_MIXED (Vertical Fragmentation)

- กรณี INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER IOT_EMP_MIXED_INSERT INSTEAD OF INSERT ON
EMP_MIXED
FOR EACH ROW
BEGIN
 BEGIN
  INSERT INTO EMP_MIXED1 (EMPNO, ENAME, JOB, MGR, DEPTNO)
  VALUES (:NEW.EMPNO, :NEW.ENAME, :NEW.JOB, :NEW.MGR
, :NEW.DEPTNO);
END;
BEGIN
```

```
  INSERT INTO EMP_MIXED2 (EMPNO, ENAME, HIREDATE, SAL, COMM)
VALUES (:NEW.EMPNO, :NEW.ENAME, :NEW.HIREDATE, :NEW.SAL
, :NEW.COMM);
 END;
 EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF UPDATE

```
CREATE OR REPLACE TRIGGER IOT_EMP_MIXED_UPDATE INSTEAD OF UPDATE
ON EMP_MIXED
FOR EACH ROW
BEGIN
BEGIN
IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL
AND :old.ename !=       :new.ename) OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.job IS NULL AND :new.job IS NOT NULL) OR
(:old.job IS NOT NULL AND :new.job IS NOT NULL AND :old.job != :new.job) OR
(:old.job IS NOT NULL AND :new.job IS NULL) OR
(:old.mgr IS NULL AND :new.mgr IS NOT NULL) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NOT NULL AND :old.mgr != :new.mgr) OR
(:old.mgr IS NOT NULL AND :new.mgr IS NULL) OR
(:old.deptno IS NULL AND :new.deptno IS NOT NULL) OR
(:old.deptno IS NOT NULL AND :new.deptno IS NOT NULL AND :old.deptno != :new.deptno)
OR
(:old.deptno IS NOT NULL AND :new.deptno IS NULL)
  THEN
   UPDATE EMP_MIXED1
```

```
    SET empno = :new.empno
    ,ENAME = :new.ename
    ,JOB = :new.job
    ,MGR = :new.mgr
    ,DEPTNO = :new.deptno
    WHERE EMPNO = :old.empno;
  END IF;
END;

BEGIN
  IF :old.empno != :new.empno OR
(:old.ename IS NULL AND :new.ename IS NOT NULL) OR
(:old.ename IS NOT NULL AND :new.ename IS NOT NULL
AND :old.ename !=       :new.ename) OR
(:old.ename IS NOT NULL AND :new.ename IS NULL) OR
(:old.sal IS NULL AND :new.sal IS NOT NULL) OR
(:old.sal IS NOT NULL AND :new.sal IS NOT NULL AND :old.sal != :new.sal) OR
(:old.sal IS NOT NULL AND :new.sal IS NULL) OR
(:old.comm IS NULL AND :new.comm IS NOT NULL) OR
(:old.comm IS NOT NULL AND :new.comm IS NOT NULL AND :old.comm != :new.comm)
OR (:old.comm IS NOT NULL AND :new.comm IS NULL)
  THEN
    UPDATE EMP_MIXED2
    SET empno = :new.empno
    ,ENAME = :new.ename
    ,SAL = :new.sal
    ,COMM = :new.comm
    WHERE EMPNO = :old.empno;
  END IF;
 END;

  EXCEPTION
```

```
 WHEN DUP_VAL_ON_INDEX THEN
  RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END;
```

- กรณีที่เป็น INSTEAD OF DELETE

```
CREATE OR REPLACE TRIGGER IOT_EMP_MIXED_DELETE INSTEAD OF DELETE ON
EMP_MIXED
FOR EACH ROW
BEGIN
 DELETE FROM EMP_MIXED1
WHERE EMPNO = :OLD.EMPNO;
 DELETE FROM EMP_MIXED2
WHERE EMPNO = :OLD.EMPNO;
END;
```

4.3 การเพิ่มความสามารถในการทำ Referential Integrity Constraint

จากขีดความสามารถของ Oracle ในบทที่ 3 ซึ่ง Oracle ไม่สามารถมี Referential Integrity Constraint ระหว่างฐานข้อมูลที่อยู่ต่าง Site กันได้ ในโครงงานชิ้นนี้จึงได้ศึกษาและพัฒนาเพื่อให้สามารถใช้งาน Referential Integrity Constraint ระหว่าง Site ได้

ซึ่งสามารถทำได้โดยการใช้งาน Trigger ซึ่งในการบังคับให้มีการควบคุมการถูกต้องของข้อมูลตรงนี้นั้น แบ่งได้เป็น สองส่วนคือ

- **ส่วนการตรวจสอบข้อมูลก่อนที่จะเพิ่มลงตารางที่มี Foreign Key**

ในส่วนนี้จะเป็นการตรวจสอบตารางที่มี Foreign Key อยู่ว่า ข้อมูลที่กำลังจะเพิ่มลงในคอลัมน์นี้นั้นตรงกับข้อมูลที่เป็น Primary Key อยู่ในอีกตารางหรือไม่ หากไม่มีก็ไม่สามารถเพิ่มข้อมูล Row นั้นลงในตารางได้ การทำงานตรงนี้นั้นสามารถใช้ Trigger มาช่วยทำงานได้

การตรวจสอบข้อมูลในตารางที่ Foreign Key อ้างอิงถึงนั้น ทำได้โดยการเรียกข้อมูลจากตารางนั้นมาดูว่า ข้อมูลที่กำลังจะเพิ่มลงใน Column ที่เป็น Foreign Key นั้นมีอยู่ใน Primary Key หรือไม่ ศึกษาได้จากตัวอย่าง ซึ่งทำการสร้าง Trigger ที่จะทำการตรวจสอบ Foreign Key ใน Column DEPTNO ของตาราง EMP ที่อ้างไปถึง Primary Key ของตาราง DEPT โดย Trigger นี้จะเป็นการทำงานก่อนการเพิ่มข้อมูล (Before Trigger)

```
CREATE OR REPLACE TRIGGER EMP_DEPT_CHECK BEFORE INSERT OR UPDATE OF
DEPTNO ON EMP
FOR EACH ROW WHEN (new.DEPTNO IS NOT NULL)
DECLARE
        Dummy  NUMBER;
        KInvalid EXCEPTION;
        KValid  EXCEPTION;
        Mutating_table    EXCEPTION;
        PRAGMA    EXCEPTION_INIT  (Mutating_table, -4901);
        CURSOR Dummy_cursor   (Dn NUMBER) IS SELECT DEPTNO FROM
DEPT@ORCL.UNGRAD15.CE.KMITL.AC.TH WHERE DEPTNO = Dn FOR UPDATE OF
DEPTNO ;
BEGIN
        OPEN Dummy_cursor ( :new.DEPTNO);
        FETCH Dummy_cursor INTO Dummy;
        IF Dummy_cursor%NOTFOUND THEN
                RAISE KInvalid;
        ELSE
                RAISE KValid;
        END IF;
        CLOSE Dummy_cursor;
EXCEPTION
        WHEN KInvalid THEN
                CLOSE Dummy_cursor;
                Raise_application_error(-20000, ' || '"Invalid Parent PK Key"' || ');
        WHEN KValid THEN
                CLOSE Dummy_cursor;
        WHEN Mutating_table THEN
                NULL;
END;
```

- **ส่วนการตรวจสอบก่อนทำการ แก้ไข เปลี่ยนแปลง ตารางที่มี Primary**

สำหรับตารางที่ Primary Key ถูกใช้เป็น Foreign Key ในอีกตารางหนึ่งนั้นก่อนที่จะมีการลบหรือแก้ไขข้อมูลในตารางนั้น ระบบจะต้องกระทำบางอย่างเพื่อความถูกต้องของข้อมูล ซึ่งขึ้นอยู่กับผู้ใช้กำหนด มี 3 กรณี คือ

I. Cascade จะเป็นการให้ระบบตามไปแก้ไขข้อมูลในตารางที่มี Foreign Key กับตารางนี้ ซึ่งถ้ามีการตั้งค่าให้ Update หรือ Delete แบบ Cascade ระบบจะตามไปแก้ไขหรือลบในตารางที่มี Foreign Key อยู่ให้ด้วย

II. Restrict เป็นการห้าม ลบหรือแก้ไข ข้อมูล Row ในตารางที่ Primary Key ถูกอ้างจาก Foreign Key ในตารางอื่นๆ

III. Set Null ในแบบนี้ ระบบจะไปทำการแก้ไขค่าในตาราง Foreign Key ให้เป็นค่า Null เมื่อ Row ในตาราง Primary Key ถูกลบหรือแก้ไข

ในทางปฏิบัตินั้น การตั้งค่าการทำงานของตาราง Primary Key นั้นสามารถตั้งค่าให้แตกต่างกันได้ระหว่างการ Update และ Delete ซึ่งทั้งสองการทำงานนี้สามารถเลือกได้สามแบบคือ Cascade, Restrict, Set Null

ต่อไปนี้เป็นตัวอย่างของการเขียน Trigger เพื่อให้สามารถคงความถูกต้องของข้อมูลได้

I. การ Update แบบ Restrict

```
CREATE OR REPLACE TRIGGER DEPT_EMP_UPDATE_RESTRICT BEFORE UPDATE
OF DEPTNO ON DEPT
FOR EACH ROW
        DECLARE Dummy   NUMBER;
        present    EXCEPTION;
        not_present EXCEPTION;
        CURSOR Dummy_cursor (Dn NUMBER) IS SELECT DEPTNO FROM
                EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH WHERE DEPTNO = Dn;
BEGIN
        OPEN Dummy_cursor (:old.DEPTNO);
        FETCH Dummy_cursor INTO Dummy;
        IF Dummy_cursor%FOUND THEN
                RAISE present;
        ELSE
                RAISE not_present;
```

```
        END IF;
        CLOSE Dummy_cursor;
EXCEPTION
        WHEN present THEN
                CLOSE Dummy_cursor;
                Raise_application_error(-20001, ' || '" FK Present in
                        EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH '" || ');
        WHEN not_prescnt THEN
                CLOSE Dummy_cursor;
END;
```

II. การ Update แบบ Cascade

ในการ Update แบบ Cascade นั้นจะมีความยุ่งยากมากกว่าส่วนอื่นๆ เนื่องจากต้องมีการไปแก้ไขค่าในตารางที่มี Foreign Key อ้างอิงมาด้วย ซึ่งต้องมีกระบวนการที่รอบคอบเพื่อป้องกันการเปลี่ยนแปลงค่าเดิมที่ไปมีผลกระทบกับข้อมูลอื่นๆที่ไม่เกี่ยวข้อง รวมทั้งการป้องกันการแก้ไขค่าผิดตำแหน่ง และการ Update แบบนี้จำเป็นต้องมีการเพิ่ม Column Update_id (Number) เพื่อไว้อ้างอิงในการแก้ไขค่าด้วย ซึ่งประกอบด้วยคำสั่งต่อไปนี้

คำสั่งสำหรับเพิ่ม Column Update_id ในตาราง Foreign Key

```
ALTER TABLE EMP ADD Update_id Number
```

คำสั่งสำหรับสร้าง Sequence เพื่อใช้ในการอ้างอิงสำหรับการ Update Primary Key ในแต่ละครั้ง

```
CREATE SEQUENCE DEPT_EMP_Update_sequence
INCREMENT BY 1 MAXVALUE 5000
CYCLE
```

คำสั่งเพื่อสร้างตัวแปรแบบ Number ไว้เก็บค่า Sequence ที่ใช้ล่าสุด

```
CREATE OR REPLACE PACKAGE DEPT_UpdateIntegritypackage AS
        Updateseq NUMBER;
END DEPT_UpdateIntegritypackage;
```

คำสั่งสร้าง Trigger เพื่อเรียกค่าตัวเลขจาก Sequence Number มามาเก็บไว้ในตัวแปรเพื่อนำไปอ้างอิง

```
CREATE OR REPLACE TRIGGER DEPT_EMP_UPDATE_CASCADE1  BEFORE UPDATE
OF DEPTNO ON DEPT
DECLARE
        Dummy NUMBER;BEGIN
        SELECT DEPT_EMP_Update_sequence.NEXTVAL INTO Dummy FROM dual;
        DEPT_UpdateIntegritypackage.Updateseq := Dummy;
END;
```

คำสั่งเพื่อสร้าง Trigger เพื่อทำการเอาตัวเลขที่เก็บไว้ ใส่ลงใน Column Update_id ของ Row ที่ทำการแก้ไข เพื่อไม่ให้มีการแก้ไขทับซ้อนกัน

```
CREATE OR REPLACE TRIGGER DEPT_EMP_UPDATE_CASCADE2  BEFORE UPDATE
OF DEPTNO ON DEPT
FOR EACH ROW
BEGIN
 IF UPDATING THEN
  UPDATE EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
    SET DEPTNO = :new.DEPTNO,
   Update_id = DEPT_UpdateIntegritypackage.Updateseq   --from 1st
   WHERE DEPTNO = :old.DEPTNO
   AND Update_id IS NULL;
 END IF;
END;
```

คำสั่งเพื่อสร้าง Trigger ที่ทำหน้าที่ลบค่าใน Column Update_id ทั้งหมด

```
CREATE OR REPLACE TRIGGER DEPT_EMP_UPDATE_CASCADE3 AFTER UPDATE
OF DEPTNO ON DEPT
BEGIN  UPDATE  EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
 SET Update_id = NULL
```

```
 WHERE  Update_id = DEPT_UpdateIntegritypackage.Updateseq;   --from 1st
END;
```

III. การ Update แบบ Set Null

```
CREATE OR REPLACE TRIGGER DEPT_EMP_UPDATE_SETNULL AFTER UPDATE OF
DEPTNO ON DEPT
FOR EACH ROW
BEGIN
        IF UPDATING AND :OLD.DEPTNO != :NEW.DEPTNO THEN
                UPDATE EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
                        SET DEPTNO = NULL WHERE DEPTNO = :old.DEPTNO;
        END IF;
END;
```

IV. การ Delete แบบ Restrict

```
CREATE OR REPLACE TRIGGER DEPT_EMP_DEL_RESTRICT BEFORE DELETE ON
DEPT
FOR EACH ROW
        DECLARE Dummy    NUMBER;
         present     EXCEPTION;
        not_present EXCEPTION;
        CURSOR Dummy_cursor (Dn NUMBER) IS SELECT DEPTNO FROM
                EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH WHERE DEPTNO = Dn;
BEGIN
        OPEN Dummy_cursor (:old.DEPTNO);
        FETCH Dummy_cursor INTO Dummy;
        IF Dummy_cursor%FOUND THEN
                RAISE present;
        ELSE
                RAISE not_present;
```

```
        END IF;
        CLOSE Dummy_cursor;
EXCEPTION
        WHEN present THEN
                CLOSE Dummy_cursor;
                Raise_application_error(-20001, ' || '"FK Present in
                        EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH '" || ');
        WHEN not_present THEN
                CLOSE Dummy_cursor;
END;
```

V. การ Delete แบบ Cascade

```
CREATE OR REPLACE TRIGGER DEPT_EMP_DEL__CASCADE AFTER DELETE ON
DEPT
FOR EACH ROW
BEGIN
        DELETE FROM EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
                WHERE EMP.DEPTNO = :old.DEPTNO;
END;
```

VI. การ Delete แบบ Set Null

```
CREATE OR REPLACE TRIGGER DEPT_EMP_DEL_SETNULL AFTER DELETE ON
DEPT
FOR EACH ROW
BEGIN
        IF DELETING THEN
                UPDATE EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
                        SET DEPTNO = NULL WHERE EMP.DEPTNO = :old.DEPTNO;
        END IF;
END;
```

4.4 การพัฒนาซอฟท์แวร์เพื่อช่วยสร้าง Global Table และ Referential Integrity Constraint

นอกจากการศึกษาค้นคว้าเพื่อพัฒนาให้ซอฟท์แวร์ฐานข้อมูลมีความสามารถในการทำ Distributed Database มากขึ้น ในโครงงานชิ้นนี้ยังได้พัฒนาซอฟท์แวร์ขึ้นมา คือ DDBHelper ซึ่งมีจุดประสงค์ คือ เพื่อช่วยผู้ใช้งานในการสร้างคำสั่งสำหรับการสร้าง Global Table และช่วยสร้างคำสั่งสำหรับสร้าง Trigger เพื่อบังคับให้มี Referential Integrity Constraint ระหว่างฐานข้อมูลที่อยู่คนละ Site ได้ โดยใช้รูปแบบของ Trigger ตามที่ได้ศึกษามาจากหัวข้อก่อนหน้านี้

- **ความสามารถของซอฟท์แวร์ DDB Helper**

  I. สามารถสร้างคำสั่งทั้งหมดที่จำเป็นในการสร้าง Global Table

  II. สามารถมี Fragment ได้ทั้งแบบ Vertical และ Horizontal รวมทั้งสามารถมี Fragment ย่อย ภายใต้ Fragment ใดๆได้

  III. สามารถเชื่อมต่อไปยังฐานข้อมูลของ Oracle ใดที่มีบัญชีผู้ใช้งานได้

  IV. สามารถเลือกตารางที่จะประกอบกันเป็น Fragment จากฐานข้อมูลใดๆที่มี Database Link เชื่อมต่ออยู่กับเครื่องที่ผู้ใช้งานเชื่อมต่ออยู่ได้

  V. กำหนดเงื่อนไขสำหรับการสร้าง Fragment แบบ Horizontal ได้

  VI. สามารถกำหนด Foreign Key และมี Referential Integrity Constraint ใน Global Table กับตารางใดๆได้

  VII. ตรวจสอบความถูกต้องของตารางที่อยู่ภายใต้ Fragment ให้มีรูปแบบถูกต้องตามชนิดของ Fragment นั้นๆ

  VIII. ตรวจสอบชื่อของ Global Table ว่าไม่ซ้ำกับชื่อใดๆในฐานข้อมูลใดๆที่เชื่อมต่ออยู่ รวมทั้งสร้าง Database Link ให้ทุกเครื่องที่เชื่อมต่ออยู่สามารถเรียกใช้งานได้โดยใช้ชื่อเดียวกัน

  IX. สามารถสร้าง Referential Integrity Constraint ระหว่าง Column ในตารางใดๆได้

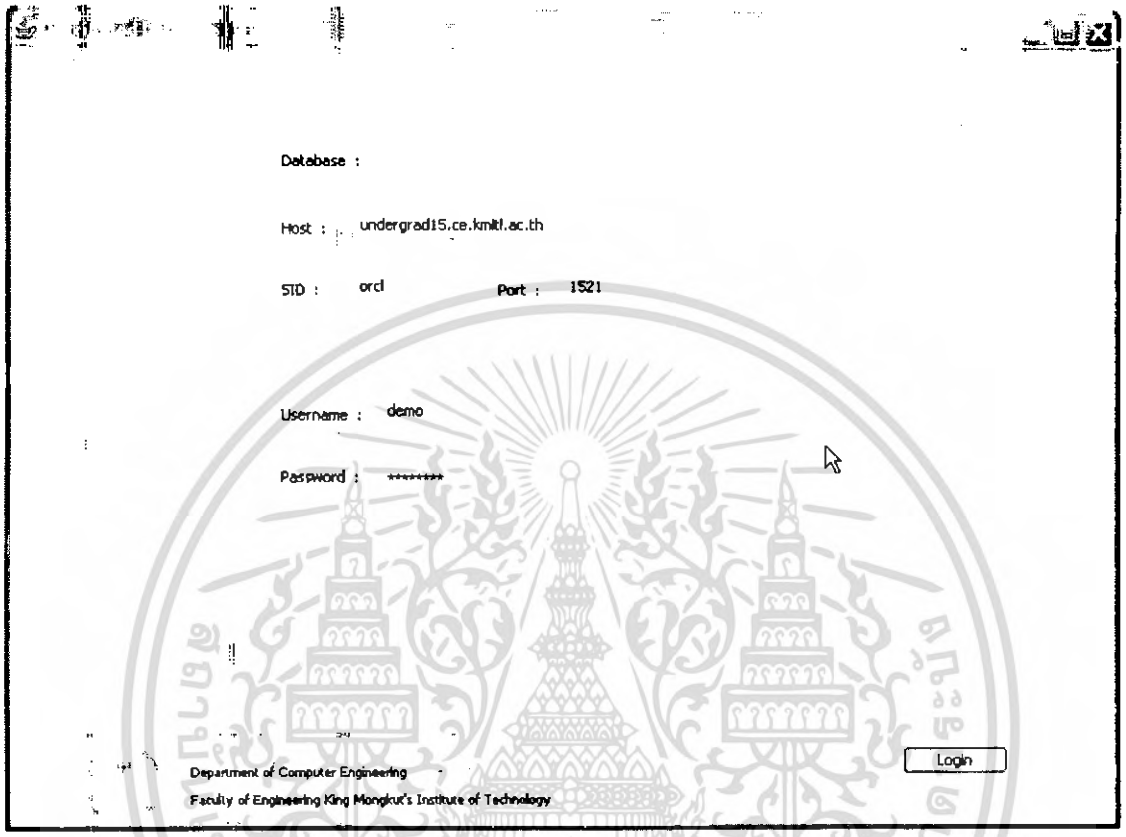
- **ขั้นตอนและวิธีการทำงานของซอฟท์แวร์ DDBHelper**

  I. **การ Login เพื่อเชื่อมต่อไปยังฐานข้อมูล**

ซึ่งซอฟท์แวร์ชิ้นนี้ได้พัฒนาขึ้นโดยรองรับให้สามารถใช้งานได้กับฐานข้อมูลใดๆ ดังนั้นจึงต้องมีการระบุค่าเพื่อเชื่อมต่อไปยังฐานข้อมูล ซึ่งได้แก่

- Host : IP Address หรือ Domain ของเครื่องเซิร์ฟเวอร์ฐานข้อมูลที่ทำการเชื่อมต่อ
- SID : ชื่อฐานข้อมูลของ Oracle ที่ต้องการใช้งาน

- Port : Port ที่รองรับสำหรับฐานข้อมูลนั้นๆ
- Username : ชื่อบัญชีผู้ใช้งานที่มีอยู่ในระบบที่ต้องการเชื่อมต่อ
- Password : รหัสผ่านที่ใช้ในการยืนยันตน



รูปที่ 4.4 หน้าต่างสำหรับ Log in เข้าใช้งานฐานข้อมูลผ่านซอฟท์แวร์ที่สร้างขึ้น

เมื่อผู้ใช้งานกรอกข้อมูลเพื่อเข้าใช้งานและกดปุ่ม Login ซอฟท์แวร์จะทำการเชื่อมต่อไปยังฐานข้อมูลที่กำหนดแล้วเริ่มสร้างการเชื่อมต่อเตรียมไว้สำหรับใช้งานต่อไป ซึ่งในซอฟท์แวร์ชิ้นนี้ได้ใช้ Driver สำหรับการเชื่อมต่อไปยังฐานข้อมูล Oracle คือ JDBC เนื่องจากไม่ต้องมีการติดตั้งเข้ากับเครื่องที่ใช้งาน เพื่อให้สามารถใช้งานซอฟท์แวร์นี้ในเครื่องใดๆได้

การสร้างการเชื่อมต่อไปยังฐานข้อมูลโดยใช้ JDBC นั้นมีขั้นตอน คือ

- ทำการดิดตั้ง Import โมดูลสำหรับใช้งานการติดต่อของ Oracle และ SQL โดยใช้คำสั่ง

```
import oracle.jdbc.OracleDriver;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;
```

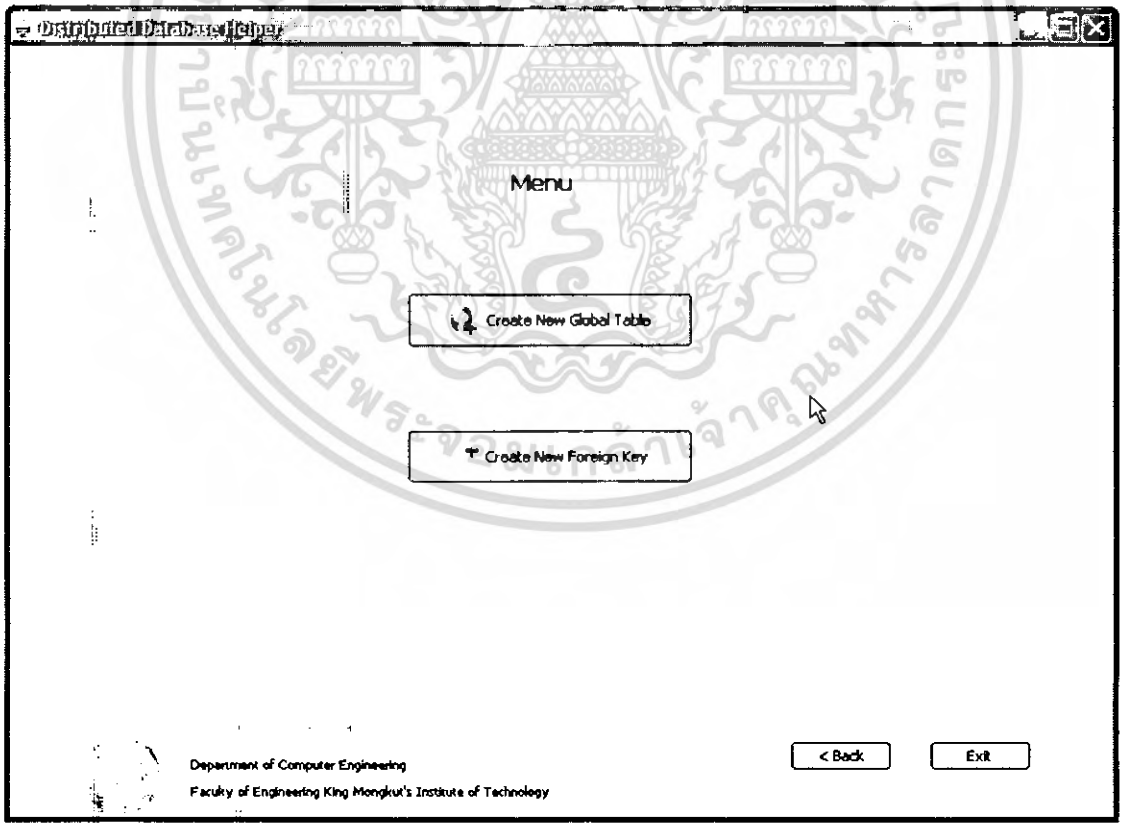
- สร้างการเชื่อมต่อโดยใช้คำสั่ง

```
Connection conn = null;
DriverManager.registerDriver(new OracleDriver());
conn = DriverManager.getConnection("jdbc:oracle:thin:@" + hostname + ":" + port +
                    ":" + sid, username, password);
```

เมื่อเชื่อมต่อสำเร็จแล้วก็สามารถนำ Object conn ไปใช้งานเพื่อเรียกดูข้อมูลต่อไป

II. Menu สำหรับเลือกการใช้งาน

ในหน้าต่างนี้จะให้ผู้ใช้งานเลือกว่าจะเข้าใช้งานส่วนใด ระหว่างการสร้าง Global Table หรือ การสร้าง Referential Integrity Constraint ระหว่างฐานข้อมูล



รูปที่ 4.5 แสดงเมนูการใช้งานของซอฟท์แวร์

III. การสร้างและตั้งค่า Fragment

เมื่อเลือกเข้าใช้งานส่วนสร้าง Global Table ซึ่งจะมีให้ตั้งค่าดังต่อไปนี้คือ

- การสร้าง Fragment ใหม่ เป็นการกำหนดชื่อของ Fragment และ กำหนดว่า Fragment ที่จะสร้างขึ้นนั้นอยู่ภายใต้ Fragment ใดหรือไม่
- การกำหนดชนิดของ Fragment ที่มี เพื่อให้สามารถตรวจสอบตารางที่อยู่ภายใต้ Fragment นั้นๆให้มีรูปแบบเดียวกัน และสร้างคำสั่งได้ถูกต้องตามรูปแบบของ Fragment นั้นๆ
- การระบุตารางให้กับ Fragment เป็นการเลือกตารางที่มีอยู่แล้วทั้งในฐานข้อมูลที่เชื่อมต่ออยู่และฐานข้อมูลอื่นๆที่มี Database Link เชื่อมต่อไป โดยซอฟท์แวร์จะทำการตรวจสอบโครงสร้างของตารางที่จะกำหนดด้วยว่าตรงกับชนิดของ Fragment นั้นๆหรือไม่

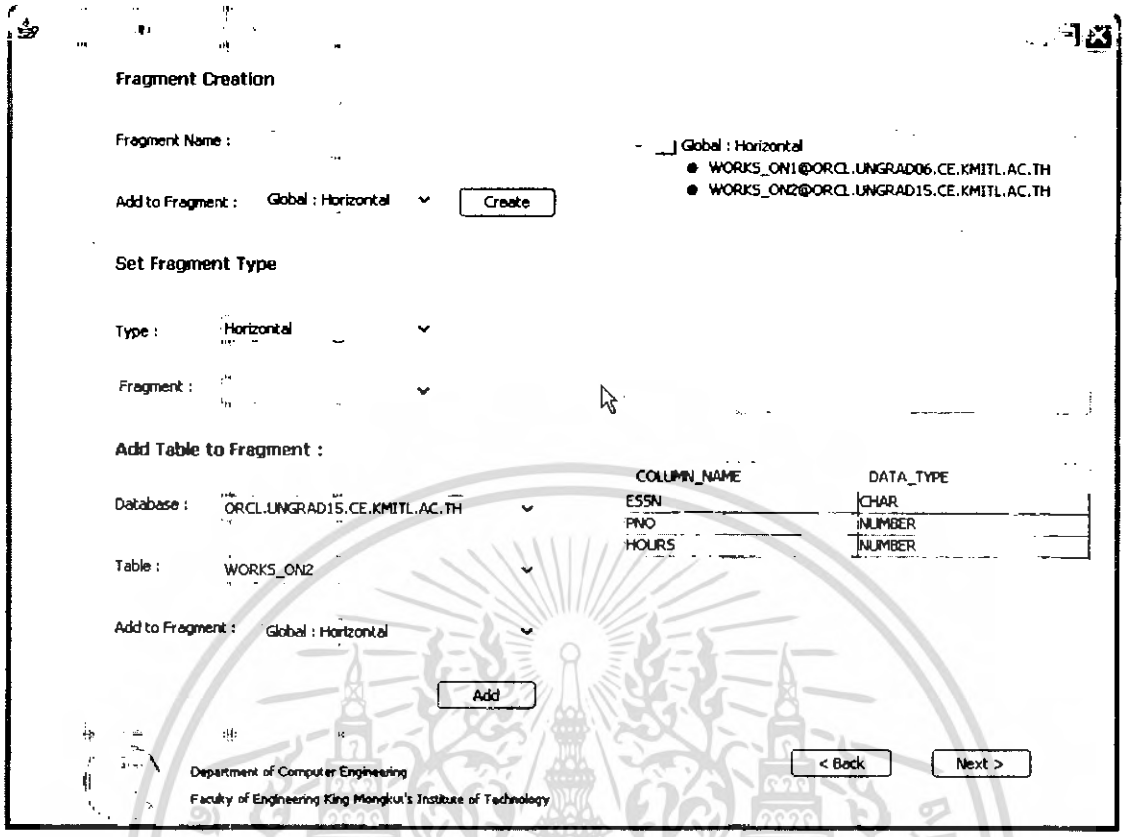
- ในหน้าสำหรับสร้างและ ตั้งค่า Fragment นั้น จะมีการแสดงข้อมูลของตารางที่มีอยู่ ซึ่งจะใช้คำสั่งดังต่อไปนี้ คือ

- ในการแสดงข้อมูลของ Database Link ที่มีอยู่ในฐานข้อมูลนั้น สามารถเรียกดูได้จาก Data Dictionary ของ Oracle โดยใช้คำสั่ง

```
SELECT * FROM ALL_DB_LINKS
```

- การตารางทั้งหมดที่ผู้ใช้งานที่ Login นั้นๆสามารถมีสิทธิเข้าถึง คือ

```
SELECT TABLE_NAME from USER_TABLES[@ site]
```



รูปที่ 4.6 แสดงหน้าต่างสำหรับสร้างและตั้งค่า Fragment

- การแสดง Column ทั้งหมดของ Table นั้นๆ จาก Data Dictionary ใช้

```
SELECT COLUMN_NAME , DATA_TYPE FROM USER_TAB_COLUMNS[@site] WHERE
TABLE_NAME = '[table name]'
```

IV. การกำหนดเงื่อนไขสำหรับ Horizontal Fragment

ถ้า Global Table ที่สร้างขึ้นเป็นการ Fragment แบบ Horizontal หรือ ประกอบด้วย Fragment แบบ Horizontal จะมีหน้าต่างขึ้นมาสำหรับการตั้งค่าเงื่อนไขในการแบ่งข้อมูล ว่าจะใช้เงื่อนไขใดในการแบ่งข้อมูล โดยจะแสดงข้อมูลของตารางว่ามี Column อะไรบ้าง

ในหน้านี้มีการเรียกดู Primary Key ของ Table ต่างๆ ซึ่งจะสามารถดูได้ผ่านคำสั่งดังนี้

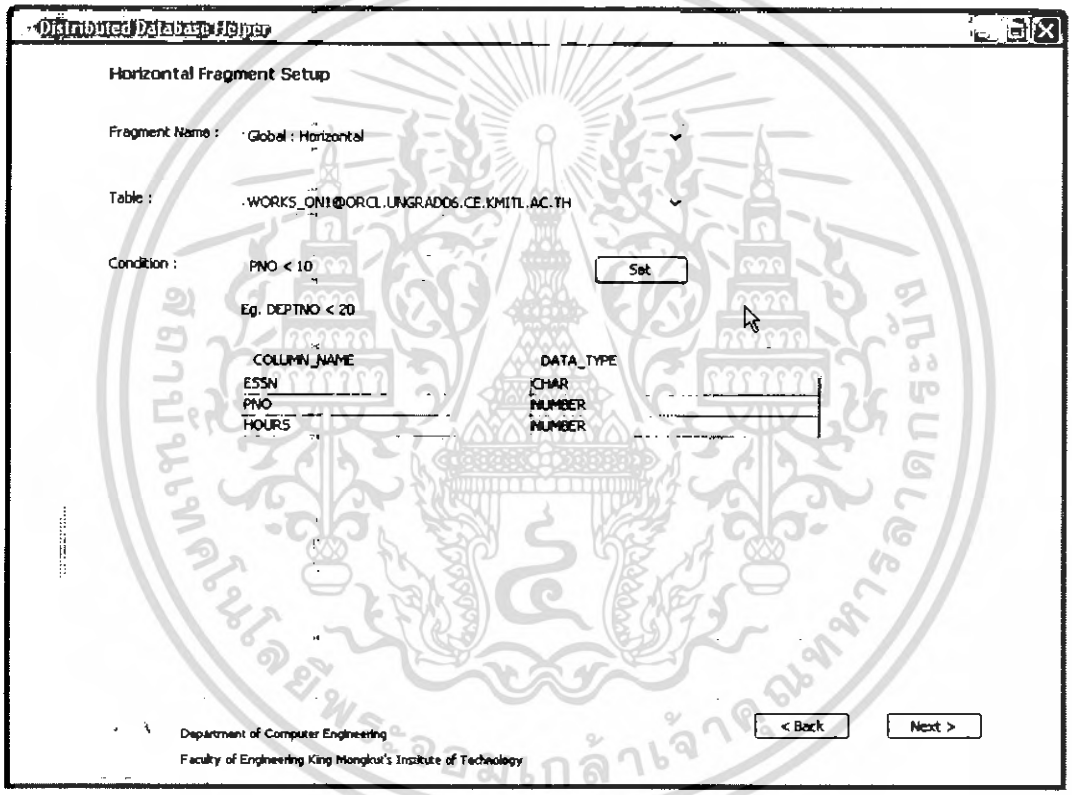
```
SELECT COLUMN_NAME FROM ALL_CONS_COLUMNS[@site] WHERE TABLE_NAME
= '[table name]' AND CONSTRAINT_NAME IN ( SELECT CONSTRAINT_NAME FROM
ALL_CONSTRAINTS[@site] WHERE TABLE_NAME = '[table name]' AND
CONSTRAINT_TYPE = 'P' )
```

V. การตั้งค่าชื่อของ Global Table

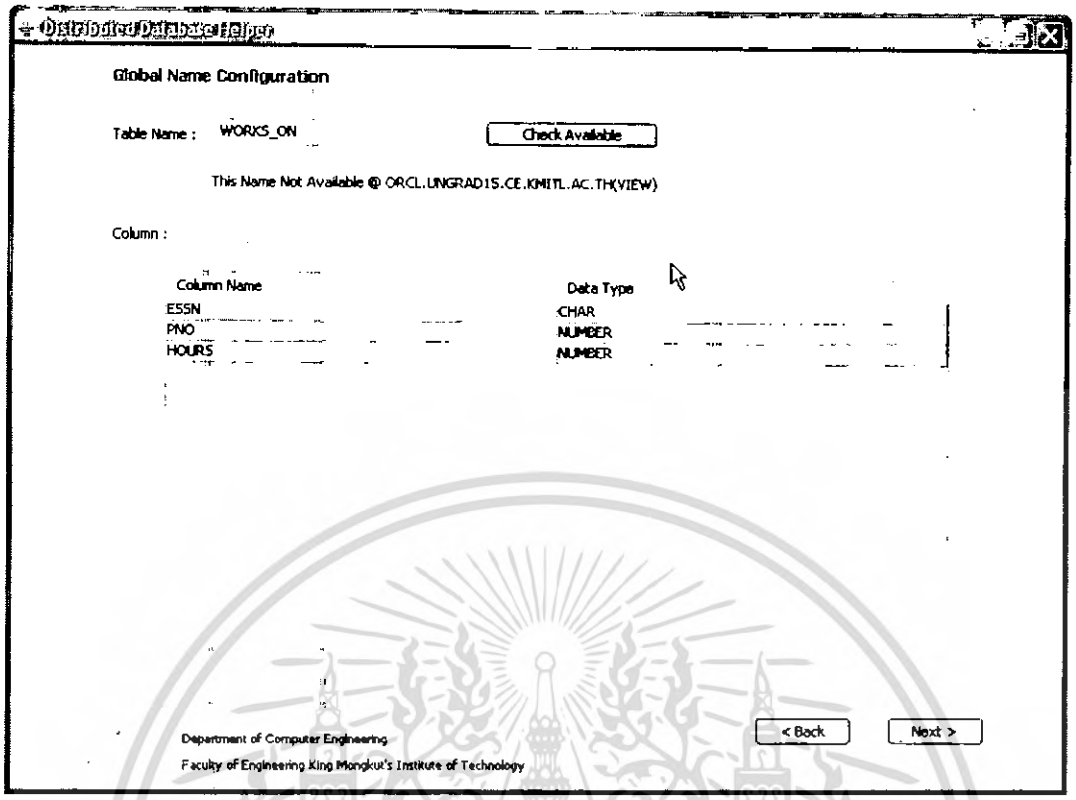
ส่วนถัดมาเป็นการตั้งชื่อของ Global Table โดยระบบจะทำการตรวจสอบให้ว่าชื่อที่ตั้งนั้นจะต้องไม่ซ้ำกับชื่อ Table หรือ View ในฐานข้อมูลใดๆที่เชื่อมต่ออยู่ รวมทั้งถ้ามีการซ้ำเกิดขึ้น ก็จะทำการแสดงรายละเอียดว่าซ้ำกับตารางหรือ View ที่ฐานข้อมูลเครื่องใด

ในการตรวจสอบ ว่าชื่อที่เพิ่มเข้าไปนั้นซ้ำกับตาราง หรือ View ใดหรือไม่ ต้องมีการเรียกชื่อของ ตารางและ View มาตรวจสอบ สามารถทำได้โดยใช้คำสั่ง SQL คือ

```
SELECT TABLE_NAME FROM USER_TABLES[@site]
และ
SELECT VIEW_NAME FROM USER_VIEWS [@site]
```



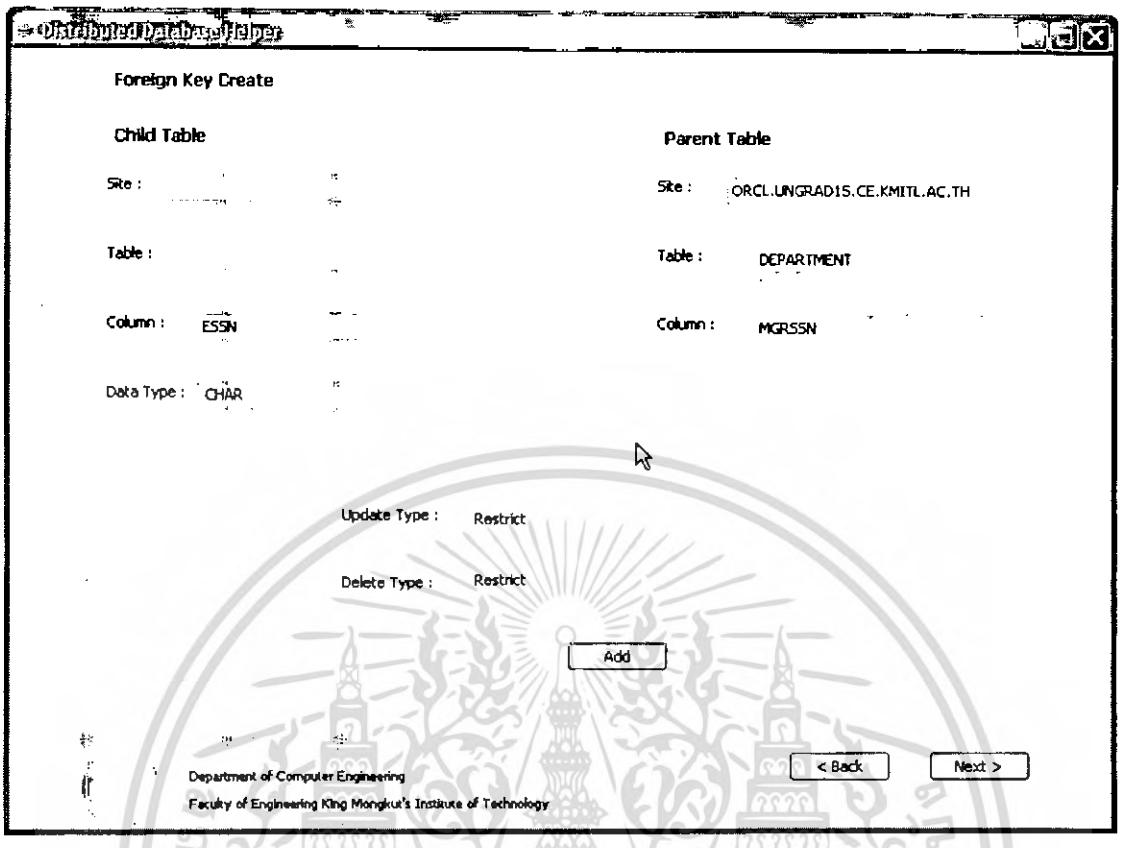
รูปที่ 4.7 แสดงหน้าต่างสำหรับตั้งค่าเงื่อนไขในการแบ่งข้อมูล ของ Horizontal Fragment



รูปที่ 4.8 แสดงหน้าต่างสำหรับตั้งค่าชื่อ Global Table

VI. การกำหนด Foreign Key

เมื่อทำการตั้งค่าชื่อของ Global Table เรียบร้อยแล้วขั้นต่อมาเป็นการกำหนด Foreign Key ให้กับ Global Table ซึ่งสามารถเลือก Column จากตารางและฐานข้อมูลใดก็ได้ที่เชื่อมต่ออยู่และซอฟท์แวร์จะทำการแสดงผลเฉพาะ Column ที่มีรูปแบบเดียวกับ Column ใน Global Table ที่เลือกไว้



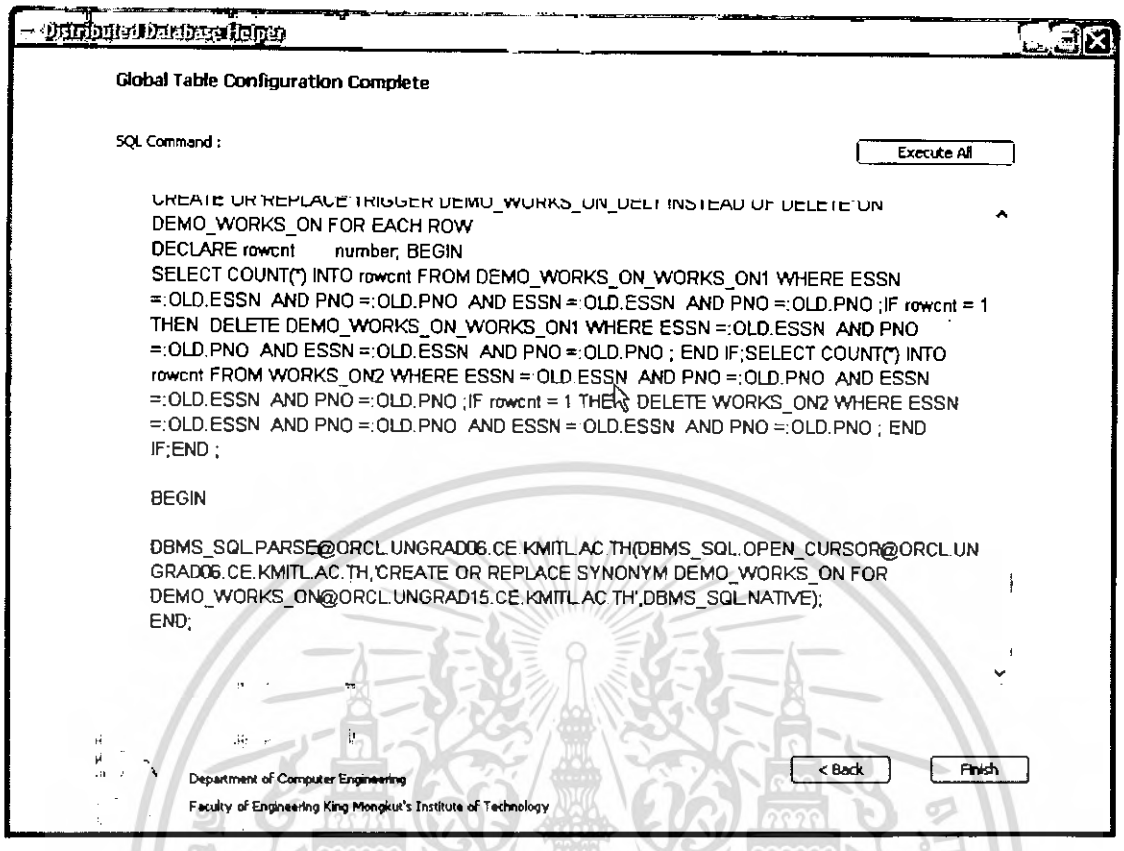
รูปที่ 4.9 แสดงหน้าต่างสำหรับตั้งค่า Foreign Key

VII. ผลลัพธ์โปรแกรม

เมื่อตั้งค่าสำหรับการสร้างตาราง Global เรียบร้อยแล้ว ระบบจะทำการสร้างคำสั่ง SQL ที่ต้องใช้ให้ ซึ่งผู้ใช้งานสามารถส่งคำสั่งทั้งหมดไปยังฐานข้อมูลได้โดยกดปุ่ม Execute All

ในการทำงานขั้นตอนนี้ ส่วนใหญ่จะเป็นคำสั่งแบบ Data Definition Language คือคำสั่งที่เกี่ยวกับโครงสร้างข้อมูล ซึ่ง Oracle ไม่อนุญาตให้ใช้คำสั่งเหล่านี้ผ่าน Database Link จึงจำเป็นที่จำต้องส่งคำสั่งผ่านไปยังฐานข้อมูลนั้นๆ โดยวิธีอื่น ซึ่งซอฟท์แวร์ชิ้นนี้ ได้ทำการส่งคำสั่งเหล่านี้โดยใช้ Procedure DBMS_SQL ซึ่งเป็น Procedure ที่ทาง Oracle มีไว้ให้ใช้ โดยมีตัวอย่างการใช้งานดังนี้

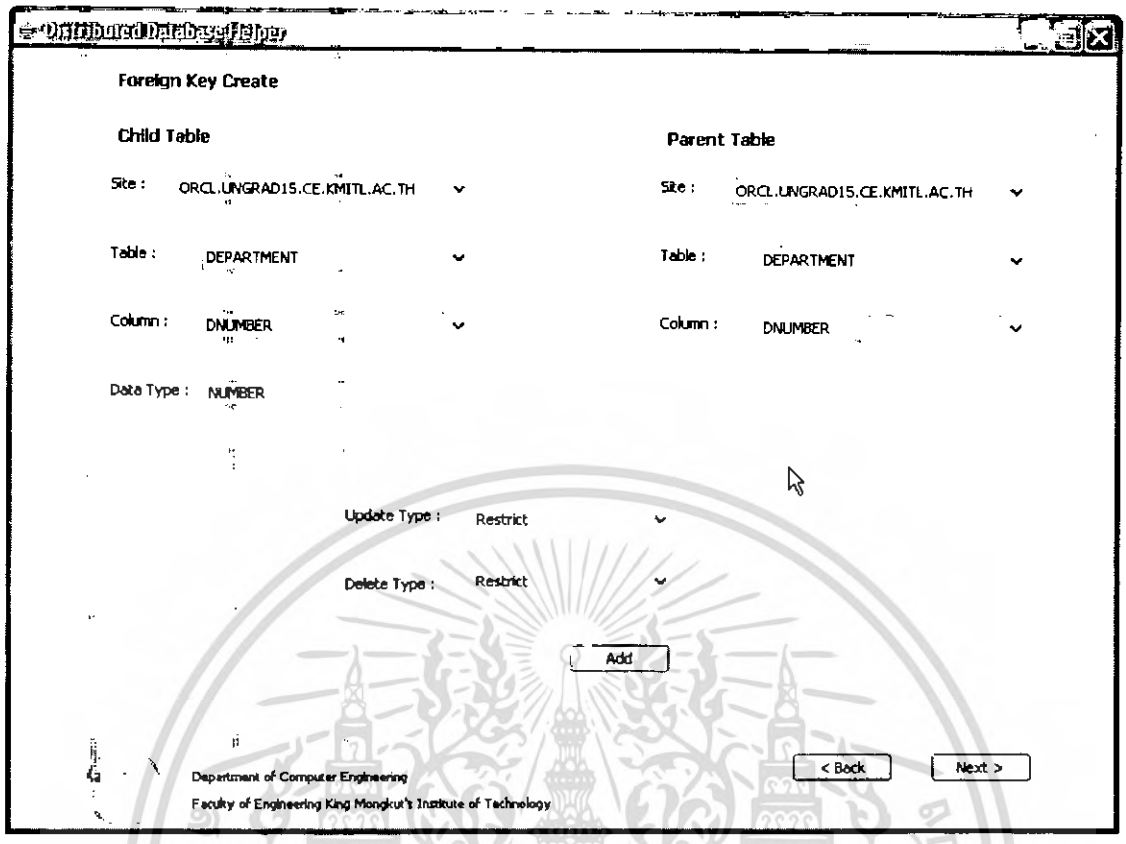
```
BEGIN
 DBMS_SQL.PARSE@[Database Link] (DBMS_SQL.OPEN_CURSOR@[Database
Link],'[คำสั่งภาษา SQL ]',DBMS_SQL.NATIVE);
END;
```



รูปที่ 4.10 แสดงหน้าต่างแสดงผลลัพธ์ของซอฟท์แวร์ซึ่งเป็นคำสั่ง SQL

VIII. ส่วนการสร้าง Foreign Key และบังคับ Referential Integrity Constraint

เป็นอีกเมนูให้เลือกใช้เพื่อการสร้าง Foreign Key และบังคับความถูกต้องของ Referential Integrity Constraint สำหรับตารางที่มีอยู่แล้วในระบบ ซึ่งหลังจากเลือก ตารางและ Column ที่ต้องการทำแล้ว ซอฟท์แวร์จะทำการสร้างคำสั่ง SQL ที่ต้องใช้มาให้ เพื่อนำไปสร้างเพื่อใช้งานต่อไป



รูปที่ 4.11 แสดงหน้าต่างสำหรับตั้งค่า Foreign Key สำหรับตารางใดๆในระบบ

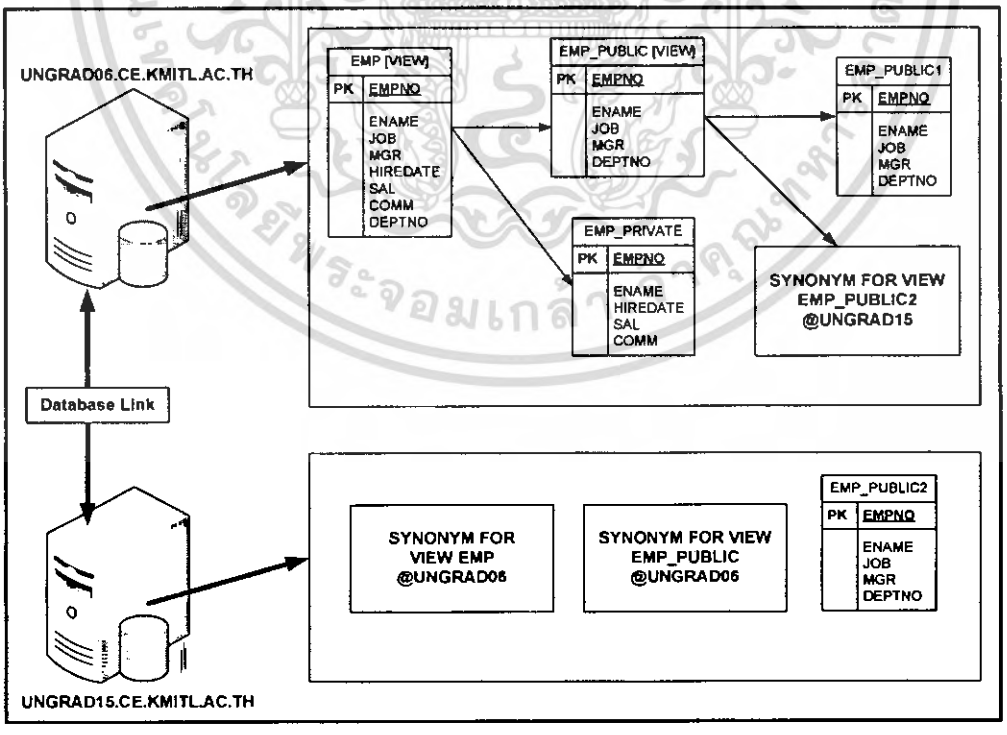
บทที่ 5

การทดลองและผลการทดลอง

สิ่งที่ได้ทำการศึกษาและพัฒนาเพื่อเพิ่มความสามารถให้กับการทำฐานข้อมูลแบบกระจายให้กับ Oracle ประกอบไปด้วยหัวข้อต่อไปนี้

- การทำให้ Oracle สามารถทำ Fragmentation Transparency ได้ โดยการใช้ Instead of Trigger ร่วมกับ View
- การทำให้สามารถกำหนด Referential Integrity Constraint กับฐานข้อมูลที่อยู่ต่างที่กัน โดยใช้ DML Trigger

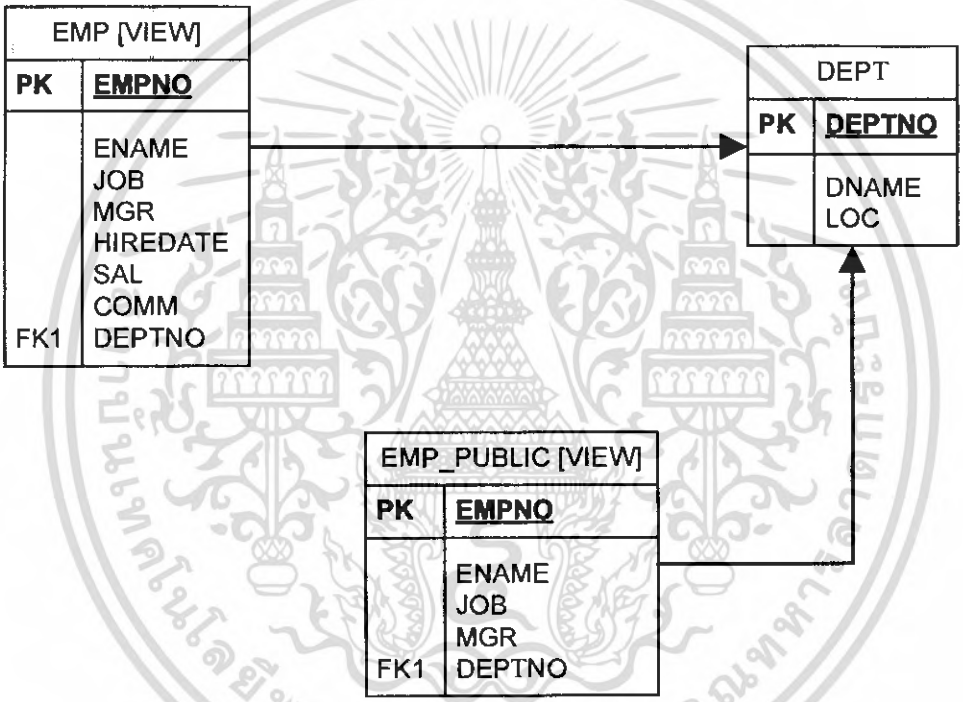
จากนั้นเพื่อช่วยผู้ใช้งานในการสร้างคำสั่งสำหรับการสร้าง Global Table และช่วยสร้างคำสั่งสำหรับสร้าง Trigger เพื่อบังคับให้มี Referential Integrity Constraint ระหว่างฐานข้อมูลที่อยู่คนละ Site ได้ ในโครงงานนี้จึงได้ทำการพัฒนาซอฟท์แวร์ขึ้นมา คือ DDBHelper และในบทนี้ก็จะเป็นการทดลองใช้ซอฟท์แวร์ DDBHelper กับตัวอย่างข้อมูลตัวอย่าง ซึ่งตัวอย่างที่จะทำการทดลองสามารถแสดงได้ดังรูปต่อไปนี้



รูปที่ 5.1 แสดงตัวอย่างการออกแบบเพื่อทดสอบฐานข้อมูลแบบกระจาย

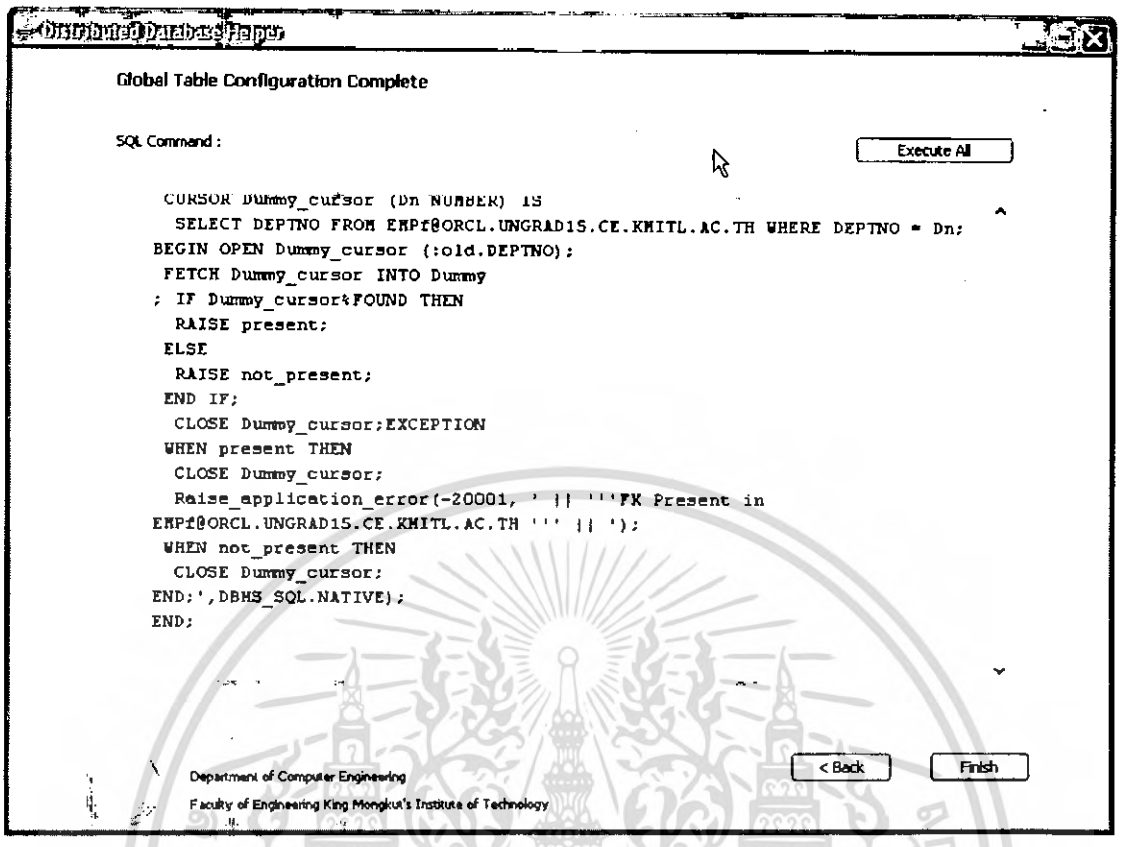
จากรูปเป็นการทดลองทำ Fragmentation Transparency จาก View ที่เป็น Mixed Fragmentation โดยที่ View ที่ชื่อ EMP_PUBLIC จะได้มาจากการทำ Horizontal Fragmentation จากตาราง EMP_PUBLIC1 ที่ ungrad06.ce.kmitl.ac.th และ EMP_PUBLIC2 ที่ ungrad15.ce.kmitl.ac.th

และ View ที่ชื่อ EMP จะได้มาจากการทำ Vertical Fragmentation จาก View EMP_PUBLIC และตาราง EMP_PRIVATE และเมื่อเราได้ Global Table จะต้องทำการกำหนด Referential integrity constraint ดังรูปด่อไปนี้



รูปที่ 5.2 แสดงตัวอย่าง Referential Integrity Constraints

จากรูปจะสามารถทำได้โดยใช้ซอฟท์แวร์ DDBHelper ในการสร้างระบบฐานข้อมูลแบบกระจาย รวมทั้งการกำหนด Refcrential integrity constraint ระหว่าง Global Table (View) กับ ตาราง หรือ ตาราง กับ ตาราง ที่ฐานข้อมูลที่ใช้งานอยู่ต่างที่กัน โดยการ UPDATE แบบ Cascade และ DELETE แบบ Restrict ได้คำสั่งดังต่อไปนี้



รูปที่ 5.3 แสดงผลลัพธ์ของซอฟท์แวร์เมื่อใช้ออกแบบตามการออกแบบที่ออกแบบไว้

คำสั่งเพิ่ม Column UPDATE_ID เพื่อใช้ทำ Referential Integrity Constraint UPDATE แบบ Cascade

```
BEGIN
DBMS_SQL.PARSE@ORCL.UNGRAD06.CE.KMITL.AC.TH(DBMS_SQL.OPEN_CURSOR
@ORCL.UNGRAD06.CE.KMITL.AC.TH,'ALTER TABLE EMP_PRIVATE ADD Update_id
Number',DBMS_SQL.NATIVE);
END;
```

คำสั่ง สร้าง Synonym สำหรับ ตาราง EMP_PUBLIC 1

```
CREATE OR REPLACE SYNONYM EMP_EMP_PUBLIC_EMP_PUBLIC1 FOR
EMP_PUBLIC1@ORCL.UNGRAD06.CE.KMITL.AC.TH
```

คำสั่ง สร้าง VIEW EMP_EMP_PUBLIC

```
CREATE OR REPLACE VIEW EMP_EMP_PUBLIC (EMPNO, ENAME, JOB, MGR, DEPTNO ) AS SELECT EMPNO, ENAME, JOB, MGR, DEPTNO FROM EMP_EMP_PUBLIC_EMP_PUBLIC1 UNION SELECT EMPNO, ENAME, JOB, MGR, DEPTNO FROM EMP_PUBLIC2
```

คำสั่งสำหรับสร้าง Trigger รองรับการ Insert บน VIEW EMP_EMP_PUBLIC

```
CREATE OR REPLACE TRIGGER EMP_EMP_PUBLIC_INST INSTEAD OF INSERT ON EMP_EMP_PUBLIC REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW BEGIN
IF :NEW.DEPTNO < 30 THEN BEGIN INSERT INTO EMP_EMP_PUBLIC_EMP_PUBLIC1 ( EMPNO, ENAME, JOB, MGR, DEPTNO ) VALUES ( :new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO ) ;
 END ;

 END IF ;
IF :NEW.DEPTNO >= 30 THEN BEGIN INSERT INTO EMP_PUBLIC2 ( EMPNO, ENAME, JOB, MGR, DEPTNO ) VALUES ( :new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO ) ;
 END ;

 END IF ;
 EXCEPTION WHEN DUP_VAL_ON_INDEX THEN RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
END ;
```

คำสั่งสำหรับสร้าง Trigger รองรับการ update บน VIEW EMP_EMP_PUBLIC

```
CREATE OR REPLACE TRIGGER EMP_EMP_PUBLIC_UPDT INSTEAD OF UPDATE ON EMP_EMP_PUBLIC FOR EACH ROW
DECLARE rowcnt      number; BEGIN
```

```
IF :old.EMPNO != :new.EMPNO OR (:old.ENAME IS NULL AND :new.ENAME IS NOT
NULL) OR(:old.ENAME IS NOT NULL AND :new.ENAME IS NOT NULL
AND :old.ENAME != :new.ENAME) OR (:old.ENAME IS NOT NULL AND :new.ENAME IS
NULL) OR (:old.JOB IS NULL AND :new.JOB IS NOT NULL) OR(:old.JOB IS NOT NULL
AND :new.JOB IS NOT NULL AND :old.JOB != :new.JOB) OR (:old.JOB IS NOT NULL
AND :new.JOB IS NULL) OR (:old.MGR IS NULL AND :new.MGR IS NOT NULL)
OR(:old.MGR IS NOT NULL AND :new.MGR IS NOT NULL AND :old.MGR != :new.MGR)
OR (:old.MGR IS NOT NULL AND :new.MGR IS NULL) OR (:old.DEPTNO IS NULL
AND :new.DEPTNO IS NOT NULL) OR(:old.DEPTNO IS NOT NULL AND :new.DEPTNO IS
NOT NULL AND :old.DEPTNO != :new.DEPTNO) OR (:old.DEPTNO IS NOT NULL
AND :new.DEPTNO IS NULL) THEN BEGIN
SELECT COUNT(*) INTO rowcnt FROM EMP_EMP_PUBLIC_EMP_PUBLIC1 WHERE
EMPNO =:OLD.EMPNO ;IF rowcnt = 1 THEN DELETE
EMP_EMP_PUBLIC_EMP_PUBLIC1 WHERE EMPNO =:OLD.EMPNO ; END IF; SELECT
COUNT(*) INTO rowcnt FROM EMP_PUBLIC2 WHERE EMPNO =:OLD.EMPNO ;IF rowcnt
= 1 THEN DELETE EMP_PUBLIC2 WHERE EMPNO =:OLD.EMPNO ; END IF;
IF :NEW.DEPTNO < 30 THEN BEGIN INSERT INTO EMP_EMP_PUBLIC_EMP_PUBLIC1
( EMPNO, ENAME, JOB, MGR, DEPTNO ) VALUES
( :new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO ) ;
 END ;

 END IF ;
IF :NEW.DEPTNO >= 30 THEN BEGIN INSERT INTO EMP_PUBLIC2 ( EMPNO, ENAME,
JOB, MGR, DEPTNO ) VALUES
( :new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO ) ;
 END ;

 END IF ;

END;
END IF; EXCEPTION WHEN DUP_VAL_ON_INDEX THEN
RAISE_APPLICATION_ERROR(-20000, 'unique constraint violated');
```

```
END ;
```

คำสั่งสำหรับสร้าง Trigger รองรับการ deleteบน VIEW EMP_EMP_PUBLIC

```
CREATE OR REPLACE TRIGGER EMP_EMP_PUBLIC_DELT INSTEAD OF DELETE ON
EMP_EMP_PUBLIC FOR EACH ROW
DECLARE rowcnt      number; BEGIN
SELECT COUNT(*) INTO rowcnt FROM EMP_EMP_PUBLIC_EMP_PUBLIC1 WHERE
EMPNO =:OLD.EMPNO ;IF rowcnt = 1 THEN DELETE
EMP_EMP_PUBLIC_EMP_PUBLIC1 WHERE EMPNO =:OLD.EMPNO ; END IF;SELECT
COUNT(*) INTO rowcnt FROM EMP_PUBLIC2 WHERE EMPNO =:OLD.EMPNO ;IF rowcnt
= 1 THEN  DELETE EMP_PUBLIC2 WHERE EMPNO =:OLD.EMPNO ; END IF;END ;
```

คำสั่งสำหรับสร้าง Synonym ตาราง EMP_PRIVATE

```
CREATE OR REPLACE SYNONYM EMP_EMP_PRIVATE
 FOR EMP_PRIVATE@ORCL.UNGRAD06.CE.KMITL.AC.TH
```

คำสั่งสำหรับสร้าง Global Table (View)

```
CREATE OR REPLACE VIEW EMP (EMPNO, ENAME, JOB, MGR, DEPTNO, HIREDATE,
SAL, COMM, UPDATE_ID  ) AS
SELECT EMP_EMP_PUBLIC.EMPNO, EMP_EMP_PUBLIC.ENAME,
EMP_EMP_PUBLIC.JOB, EMP_EMP_PUBLIC.MGR, EMP_EMP_PUBLIC.DEPTNO,
EMP_EMP_PRIVATE.HIREDATE, EMP_EMP_PRIVATE.SAL,
EMP_EMP_PRIVATE.COMM, EMP_EMP_PRIVATE.UPDATE_ID FROM
EMP_EMP_PUBLIC FULL OUTER JOIN EMP_EMP_PRIVATE ON
EMP_EMP_PUBLIC.EMPNO = EMP_EMP_PRIVATE.EMPNO
```

คำสั่งสำหรับสร้าง Trigger รองรับการ Insert บน EMP

```
CREATE OR REPLACE TRIGGER EMP_INST INSTEAD OF INSERT ON EMP
FOR EACH ROW
 BEGIN
 BEGIN
```

```
INSERT INTO EMP_EMP_PUBLIC (EMPNO, ENAME, JOB, MGR, DEPTNO ) VALUES
( :new.EMPNO, :new.ENAME, :new.JOB, :new.MGR, :new.DEPTNO ) ;
END ;
BEGIN
INSERT INTO EMP_EMP_PRIVATE ( EMPNO, ENAME, HIREDATE, SAL, COMM,
UPDATE_ID ) VALUES
( :new.EMPNO, :new.ENAME, :new.HIREDATE, :new.SAL, :new.COMM, :new.UPDATE_ID
) ;
END ;
EXCEPTION WHEN DUP_VAL_ON_INDEX THEN RAISE_APPLICATION_ERROR(-
20000, 'unique constraint violated');
END ;
```

คำสั่งสำหรับสร้าง Trigger รองรับการ update บน EMP

```
CREATE OR REPLACE TRIGGER EMP_UPDT INSTEAD OF UPDATE ON EMP FOR
EACH ROW
BEGIN
BEGIN
IF :old.EMPNO != :new.EMPNO OR (:old.ENAME IS NULL AND :new.ENAME IS NOT
NULL) OR(:old.ENAME IS NOT NULL AND :new.ENAME IS NOT NULL
AND :old.ENAME != :new.ENAME) OR (:old.ENAME IS NOT NULL AND :new.ENAME IS
NULL) OR (:old.JOB IS NULL AND :new.JOB IS NOT NULL) OR(:old.JOB IS NOT NULL
AND :new.JOB IS NOT NULL AND :old.JOB != :new.JOB) OR (:old.JOB IS NOT NULL
AND :new.JOB IS NULL) OR (:old.MGR IS NULL AND :new.MGR IS NOT NULL)
OR(:old.MGR IS NOT NULL AND :new.MGR IS NOT NULL AND :old.MGR != :new.MGR)
OR (:old.MGR IS NOT NULL AND :new.MGR IS NULL) OR (:old.DEPTNO IS NULL
AND :new.DEPTNO IS NOT NULL) OR(:old.DEPTNO IS NOT NULL AND :new.DEPTNO IS
NOT NULL AND :old.DEPTNO != :new.DEPTNO) OR (:old.DEPTNO IS NOT NULL
AND :new.DEPTNO IS NULL) THEN
UPDATE EMP_EMP_PUBLIC SET EMPNO = :new.EMPNO, ENAME = :new.ENAME, JOB
= :new.JOB, MGR = :new.MGR, DEPTNO = :new.DEPTNO WHERE EMPNO =:old.EMPNO ;
```

```
END IF;


END ;
BEGIN
IF :old.EMPNO != :new.EMPNO OR (:old.ENAME IS NULL AND :new.ENAME IS NOT
NULL) OR(:old.ENAME IS NOT NULL AND :new.ENAME IS NOT NULL
AND :old.ENAME != :new.ENAME) OR (:old.ENAME IS NOT NULL AND :new.ENAME IS
NULL)  OR (:old.HIREDATE IS NULL AND :new.HIREDATE IS NOT NULL)
OR(:old.HIREDATE IS NOT NULL AND :new.HIREDATE IS NOT NULL
AND :old.HIREDATE != :new.HIREDATE) OR (:old.HIREDATE IS NOT NULL
AND :new.HIREDATE IS NULL)  OR (:old.SAL IS NULL AND :new.SAL IS NOT NULL)
OR(:old.SAL IS NOT NULL AND :new.SAL IS NOT NULL AND :old.SAL != :new.SAL) OR
(:old.SAL IS NOT NULL AND :new.SAL IS NULL)  OR (:old.COMM IS NULL
AND :new.COMM IS NOT NULL) OR(:old.COMM IS NOT NULL AND :new.COMM IS NOT
NULL AND :old.COMM != :new.COMM) OR (:old.COMM IS NOT NULL AND :new.COMM
IS NULL)  OR (:old.UPDATE_ID IS NULL AND :new.UPDATE_ID IS NOT NULL)
OR(:old.UPDATE_ID IS NOT NULL AND :new.UPDATE_ID IS NOT NULL
AND :old.UPDATE_ID != :new.UPDATE_ID) OR (:old.UPDATE_ID IS NOT NULL
AND :new.UPDATE_ID IS NULL)  THEN UPDATE EMP_EMP_PRIVATE SET EMPNO
= :new.EMPNO, ENAME = :new.ENAME, HIREDATE = :new.HIREDATE, SAL = :new.SAL,
COMM = :new.COMM, UPDATE_ID = :new.UPDATE_ID WHERE EMPNO =:old.EMPNO ;


END IF;


END ;
END;
```

คำสั่งสำหรับสร้าง Trigger รองรับการ delete บน EMP

```
CREATE OR REPLACE TRIGGER EMP_DELT INSTEAD OF DELETE ON EMP FOR
EACH ROW
 BEGIN
BEGIN
```

```
DELETE FROM EMP_EMP_PUBLIC WHERE EMPNO =:old.EMPNO ;
END ;
BEGIN
DELETE FROM EMP_EMP_PRIVATE WHERE EMPNO =:old.EMPNO ;
END ;
END ;
```

คำสั่งสำหรับสร้าง Synonym ตาราง EMP บนเครื่องอื่นๆ

```
BEGIN
DBMS_SQL.PARSE@ORCL.UNGRAD06.CE.KMITL.AC.TH(DBMS_SQL.OPEN_CURSOR
@ORCL.UNGRAD06.CE.KMITL.AC.TH,'CREATE OR REPLACE SYNONYM EMP FOR
EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH',DBMS_SQL.NATIVE);
END;
```

คำสั่งสำหรับตรวจสอบ Primary Key สำหรับ Referential Integrity Constraint

```
BEGIN
DBMS_SQL.PARSE@ORCL.UNGRAD06.CE.KMITL.AC.TH(DBMS_SQL.OPEN_CURSOR
@ORCL.UNGRAD06.CE.KMITL.AC.TH,'CREATE OR REPLACE TRIGGER
EMP_PUBLIC1_DEPT_CHECK
BEFORE INSERT OR UPDATE OF DEPTNO ON EMP_PUBLIC1
FOR EACH ROW WHEN (new.DEPTNO IS NOT NULL)
DECLARE
 Dummy  NUMBER;
 KInvalid EXCEPTION;
 KValid  EXCEPTION;
 Mutating_table   EXCEPTION;
 PRAGMA   EXCEPTION_INIT  (Mutating_table, -4901);
 CURSOR Dummy_cursor   (Dn NUMBER) IS
  SELECT   DEPTNO   FROM   DEPT@ORCL.UNGRAD06.CE.KMITL.AC.TH   WHERE
DEPTNO = Dn FOR UPDATE OF DEPTNO ;
BEGIN
```

```
 OPEN Dummy_cursor ( :new.DEPTNO);
 FETCH Dummy_cursor INTO Dummy;
  IF Dummy_cursor%NOTFOUND THEN
    RAISE KInvalid;
  ELSE
    RAISE KValid;
  END IF;
  CLOSE Dummy_cursor;
EXCEPTION
  WHEN KInvalid THEN
    CLOSE Dummy_cursor;
    Raise_application_error(-20000, ' || '"Invalid Parent PK Key"' || ');
  WHEN KValid THEN
    CLOSE Dummy_cursor;
  WHEN Mutating_table THEN
    NULL;
END;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสำหรับตรวจสอบ Primary Key สำหรับ Referential Integrity Constraint

```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE TRIGGER
EMP_PUBLIC2_DEPT_CHECK
BEFORE INSERT OR UPDATE OF DEPTNO ON EMP_PUBLIC2
FOR EACH ROW WHEN (new.DEPTNO IS NOT NULL)
DECLARE
 Dummy  NUMBER;
 KInvalid EXCEPTION;
 KValid  EXCEPTION;
 Mutating_table   EXCEPTION;
 PRAGMA   EXCEPTION_INIT  (Mutating_table, -4901);
 CURSOR Dummy_cursor   (Dn NUMBER) IS
```

```
 SELECT DEPTNO FROM DEPT@ORCL.UNGRAD06.CE.KMITL.AC.TH WHERE
DEPTNO = Dn FOR UPDATE OF DEPTNO ;
BEGIN
 OPEN Dummy_cursor ( :new.DEPTNO);
 FETCH Dummy_cursor INTO Dummy;
  IF Dummy_cursor%NOTFOUND THEN
    RAISE KInvalid;
  ELSE
    RAISE KValid;
  END IF;
  CLOSE Dummy_cursor;
EXCEPTION
  WHEN KInvalid THEN
    CLOSE Dummy_cursor;
    Raise_application_error(-20000, ' || '"Invalid Parent PK Key"' || ');
  WHEN KValid THEN
    CLOSE Dummy_cursor;
  WHEN Mutating_table THEN
    NULL;
END;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสำหรับสร้าง sequence สำหรับ Update Cascade

```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE SEQUENCE
DEPT_EMP_Update_sequence
INCREMENT BY 1 MAXVALUE 5000
CYCLE
',DBMS_SQL.NATIVE);
END;
```

คำสั่งสำหรับสร้าง Package เพื่อเก็บข้อมูลสำหรับการ Update แบบ Cascade

```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE PACKAGE
DEPT_UpdateIntegritypackage AS
  Updateseq NUMBER;
END DEPT_UpdateIntegritypackage;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสร้าง Trigger สำหรับ Update cascade ชุดที่ 1

```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE TRIGGER
DEPT_EMP_UPDATE_CASCADE1 BEFORE UPDATE OF DEPTNO ON DEPT
DECLARE
 Dummy NUMBER;BEGIN
 SELECT DEPT_EMP_Update_sequence.NEXTVAL
  INTO Dummy
  FROM dual;
   DEPT_UpdateIntegritypackage.Updateseq := Dummy;
END;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสร้าง Trigger สำหรับ Update cascade ชุดที่ 2

```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE TRIGGER
DEPT_EMP_UPDATE_CASCADE2 BEFORE UPDATE OF DEPTNO ON DEPT
FOR EACH ROW
BEGIN
 IF UPDATING THEN
  UPDATE EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
    SET DEPTNO = :new.DEPTNO,
   Update_id = DEPT_UpdateIntegritypackage.Updateseq   --from 1st
```

```
  WHERE DEPTNO = :old.DEPTNO
  AND Update_id IS NULL;
 END IF;
END;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสร้าง Trigger สำหรับ Update cascade ชุดที่ 3

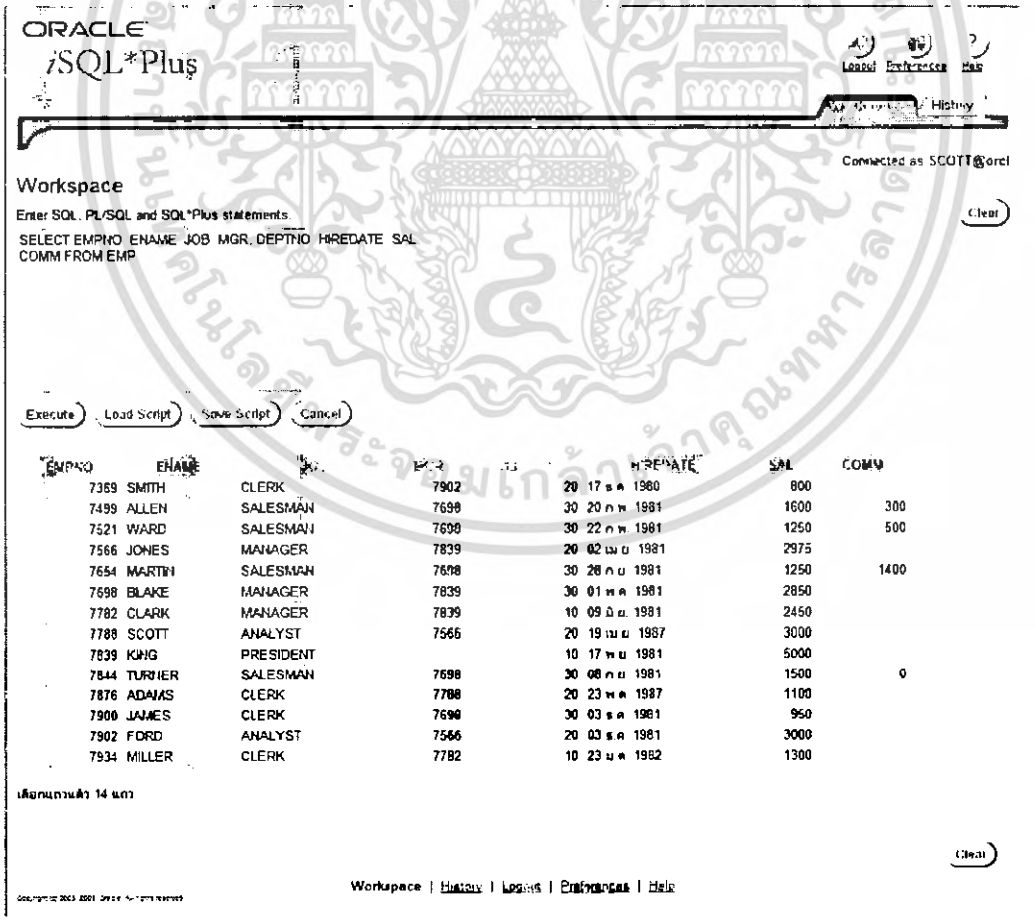
```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE TRIGGER
DEPT_EMP_UPDATE_CASCADE3 AFTER UPDATE OF DEPTNO ON DEPT
BEGIN UPDATE EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH
 SET Update_id = NULL
  WHERE Update_id = DEPT_UpdateIntegritypackage.Updateseq;  --from 1st
END;',DBMS_SQL.NATIVE);
END;
```

คำสั่งสร้าง Trigger สำหรับ Delete แบบ Restrict

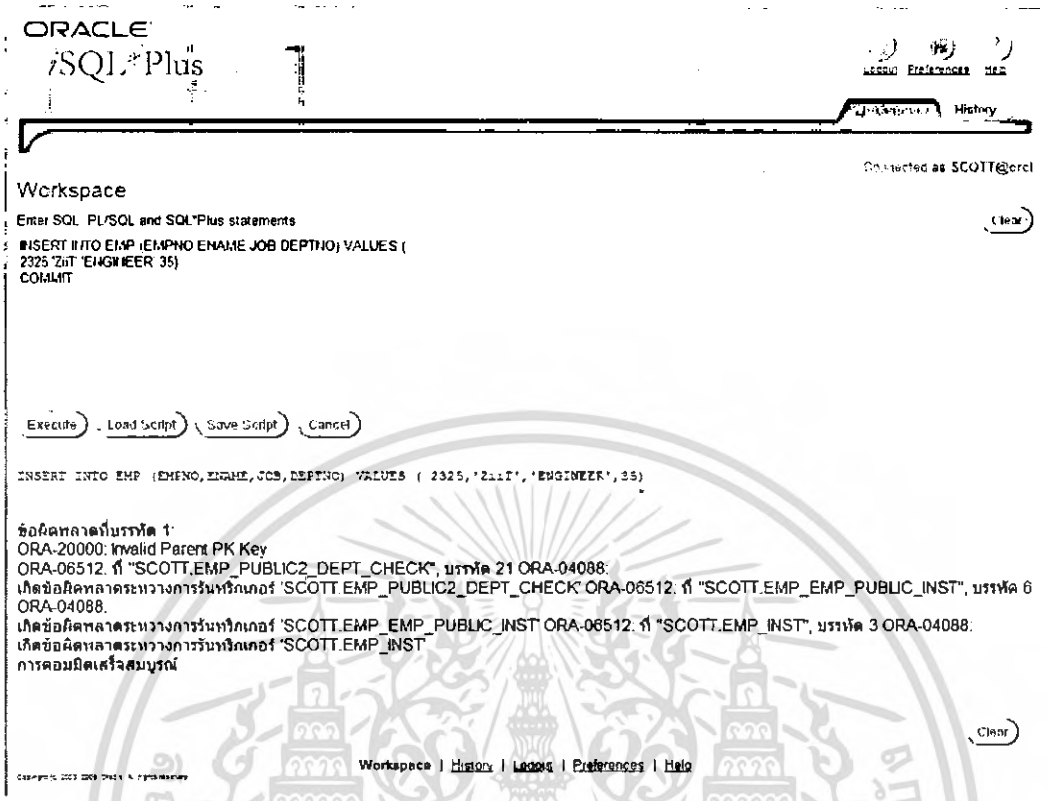
```
BEGIN
 DBMS_SQL.PARSE(DBMS_SQL.OPEN_CURSOR,'CREATE OR REPLACE TRIGGER
DEPT_EMP_DEL_RESTRICT BEFORE DELETE ON DEPT
FOR EACH ROW
DECLARE Dummy   NUMBER;
 present    EXCEPTION;
 not_present EXCEPTION;
 CURSOR Dummy_cursor (Dn NUMBER) IS
  SELECT DEPTNO FROM EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH WHERE DEPTNO
= Dn;
BEGIN OPEN Dummy_cursor (:old.DEPTNO);
 FETCH Dummy_cursor INTO Dummy
; IF Dummy_cursor%FOUND THEN
  RAISE present;
```

```
ELSE
 RAISE not_present;
END IF;
 CLOSE Dummy_cursor;EXCEPTION
WHEN present THEN
 CLOSE Dummy_cursor;
 Raise_application_error(-20001, ' || '"FK Present in
EMP@ORCL.UNGRAD15.CE.KMITL.AC.TH '" || ');
 WHEN not_present THEN
 CLOSE Dummy_cursor;
END;',DBMS_SQL.NATIVE);
END;
```

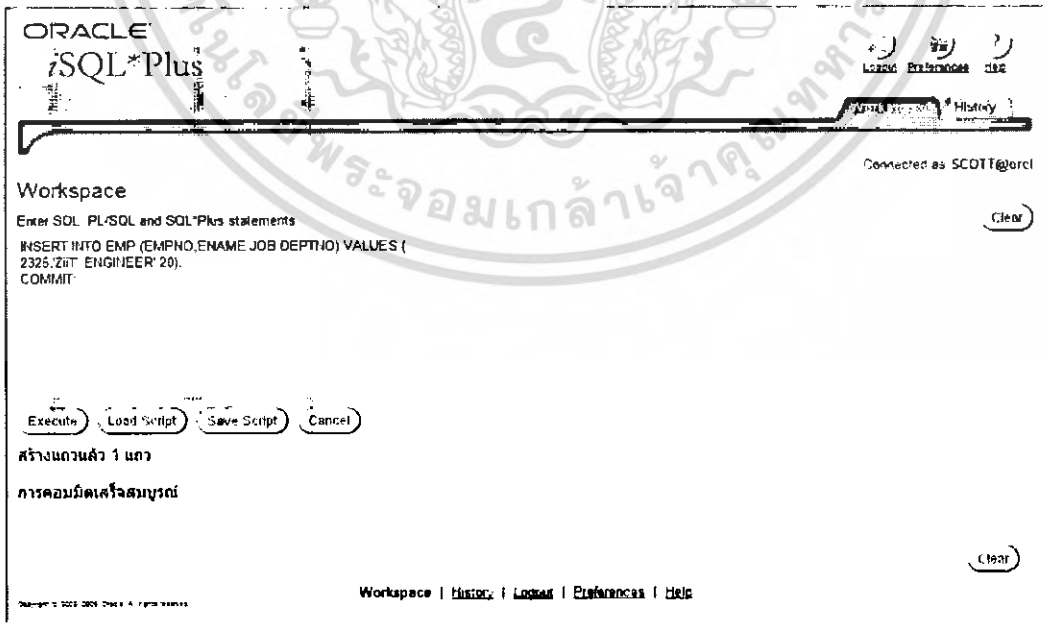
เมื่อกดปุ่ม Execute All ได้ผลลัพธ์บนฐานข้อมูลดังนี้



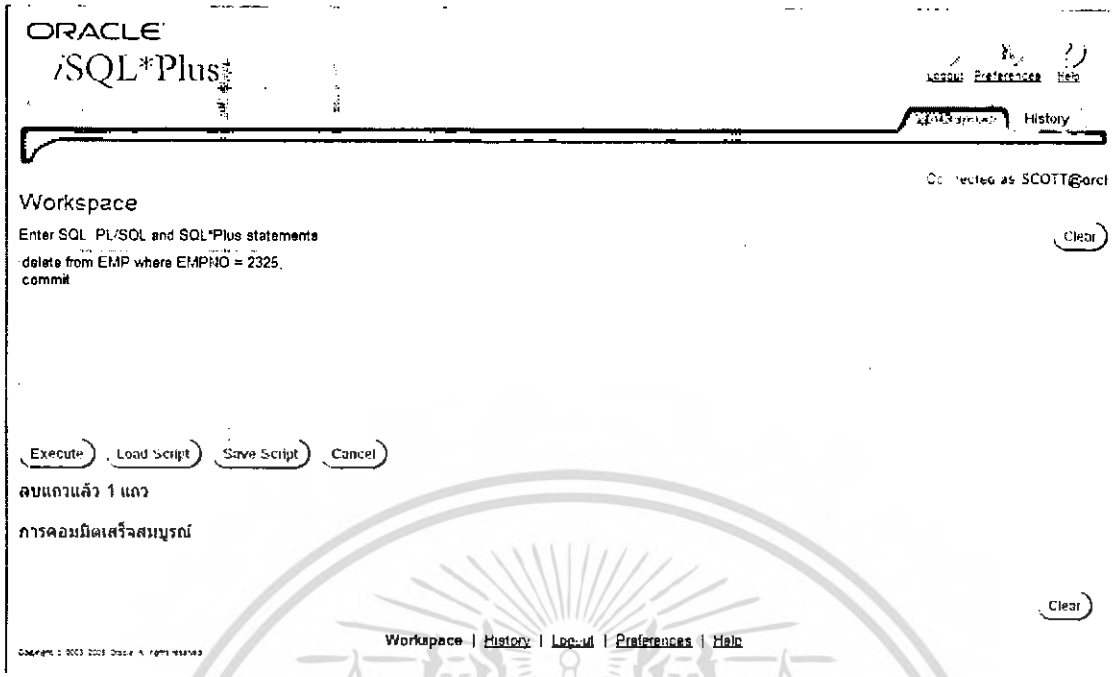
รูปที่ 5.4 แสดงตัวอย่างการใช้คำสั่ง SELECT เพื่อเรียกดูข้อมูลจาก Global Table



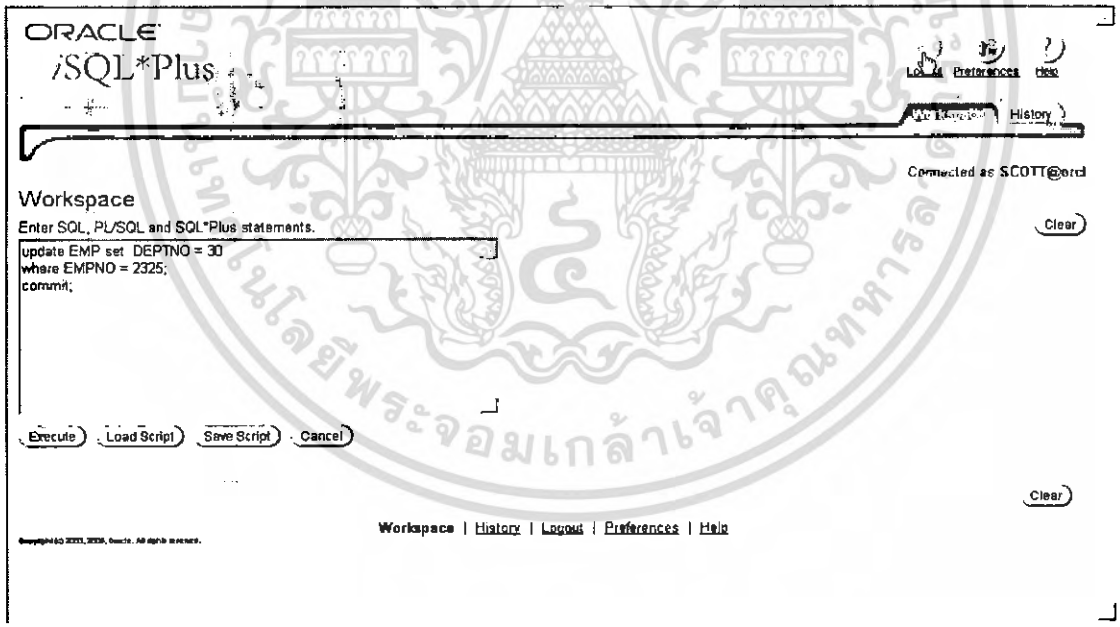
รูปที่ 5.5 แสดงตัวอย่างการตรวจสอบความถูกต้องของ Referential Integrity Constraint บน Global Table



รูปที่ 5.6 แสดงตัวอย่างการ INSERT ข้อมูลบน Global Table



รูปที่ 5.7 แสดงตัวอย่างการ DELETE ข้อมูลบน Global Table



รูปที่ 5.8 แสดงตัวอย่างการ UPDATE ข้อมูลบน Global Table

บทที่ 6

บทวิจารณ์และสรุป

6.1 บทวิจารณ์และสรุปผล

จากการศึกษาพบว่า การทำฐานข้อมูลแบบกระจายของ Oracle 10g Enterprise Edition นั้น พบว่า Oracle ได้สนับสนุนการทำงานได้ในระดับหนึ่ง แต่ก็ยังมีข้อจำกัดอยู่หลายประการที่ยังไม่สามารถทำได้ รวมทั้งยังมีอีกหลายหัวข้อซึ่งยังไม่ได้ทำการศึกษา เช่น การควบคุมดูแลบัญชีผู้เข้าใช้งานที่สามารถเข้าใช้งานในฐานข้อมูลได้หลายๆที่ การเพิ่มประสิทธิภาพของ Global Query Optimization และ การดูแลการออกแบบฐานข้อมูลแบบกระจาย เป็นต้น ซึ่งอาจทำให้พบข้อจำกัดที่เพิ่มขึ้น และในปัจจุบันความสนใจในการพัฒนาระบบตามแนวความคิดในการทำฐานข้อมูลแบบกระจายได้ลดน้อยลง ซึ่งถ้าจะมีการใช้งานฐานข้อมูลแบบกระจายนั้น ส่วนใหญ่จะเป็นการนำไปใช้เพียงบางความสามารถเท่านั้น เช่นการทำ Location Transparency การทำ Asynchronous Replication เป็นต้น

และจากการพัฒนาซอฟท์แวร์ขึ้นมาเพื่อช่วยในการสร้างคำสั่งนั้น พบว่ายังสามารถใช้สร้างคำสั่ง SQL สำหรับรูปแบบ Global Table ที่ไม่ซับซ้อนมากนัก ซึ่งถ้า Global Table ที่สร้างขึ้น มีความซับซ้อนมากจำเป็นที่จำต้องออกแบบและพัฒนาคำสั่งขึ้นมาโดยเฉพาะสำหรับ Table นั้นๆ เพื่อให้ใช้งานได้อย่างถูกต้อง

6.2 ปัญหาอุปสรรคและแนวทางแก้ไข

6.2.1 เนื่องจากปัจจุบันความสนใจในการใช้งานฐานข้อมูล ค่อนข้างจะเน้นไปที่ระบบ Centralized ซึ่งทำให้การหาข้อมูลเพื่อศึกษาขีดจำกัดหรือพัฒนาวิธีการที่จะเพิ่มความสามารถของ Oracle นั้นค่อยข้างน้อย จึงต้องใช้เวลาค่อนข้างมากในการหาความสามารถของ Oracle ที่มีอยู่และหาวิธีการพัฒนาให้ดีขึ้น

6.2.2 ความแตกต่างของฐานข้อมูลแบบกระจายกับฐานข้อมูลปกติมีอยู่มากพอสมควร และในเวลาจำกัด จึงไม่สามารถทดสอบความสามารถของ Oracle ได้ทั้งหมด จึงทำการเลือกจุดทดสอบที่น่าสนใจและสามารถพัฒนาความสามารถได้ภายในเวลาที่เหมาะสม

6.2.3 บางครั้งเนื่องจากเกิดความผิดพลาดของระบบ หรือการติดตั้งระบบฐานข้อมูลในตอนเริ่มแรก ทำให้ต้องเสียเวลาในการค้นหาวิธีแก้ไขเป็นเวลานาน

6.3 แนวทางการพัฒนาต่อ

6.3.1 ศึกษาแนวทางในการจัดการบัญชีผู้ใช้งานสำหรับฐานข้อมูลแบบกระจาย เพื่อเพิ่มความปลอดภัยในการทำงาน

6.3.2 ศึกษาหาวิธีการออกแบบฐานข้อมูลแบบกระจายอย่างไรให้มีประสิทธิภาพสูงที่สุด

6.3.3 ศึกษาหาวิธีพัฒนา Global Query Optimization ให้มีความสามารถเพิ่มมากขึ้น

6.3.4 พัฒนาซอฟท์แวร์ที่สร้างขึ้นให้มีความสามารถรองรับ Design ที่มีความซับซ้อนขึ้น

บรรณานุกรม

เอกสารอ้างอิงที่เป็นหนังสือ

[1] C.J. Date 2004, "An Introduction to Database Systems" Addison-Wesley

[2] Abraham Silberschatz, Henry F. Korth, S. Sudarshan 2006, "Database System Concepts" New York, McGraw-Hill

[3] Stefano Ceri,Giuseppe Pelagatti 1986, "Distributed Databases Principles & Systems" Singapore, McGraw-Hill

[4] George Coulouris, Jean Dollimore and Tim Kindberg 1994, "Distributed systems : concepts and design" New York , Addison-Wesley

เอกสารอ้างอิงที่เป็นปริญญานิพนธ์

[5] สมชิด ศิริผลหลาย "ระบบประมวลผลคำถามสำหรับฐานข้อมูลแบบกระจายในโครงข่ายคอมพิวเตอร์ TCP/IP" วิทยานิพนธ์ วิศวกรรมศาสตรมหาบัณฑิต วิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2536

[6] เอก โชติชวาลวงศ์ "การประมวลผลคำถามแบบกระจายบนระบบจัดการฐานข้อมูล" ปริญญานิพนธ์ วิศวกรรมศาสตรบัณฑิต วิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2536

เอกสารอ้างอิงที่เป็น Web-site

[7] Oracle Faq. "Oracle Advanced Replication - Oracle FAQ" [Online].
Available : http://orafaq.com/wiki/Oracle_Advanced_Replication. 2006.

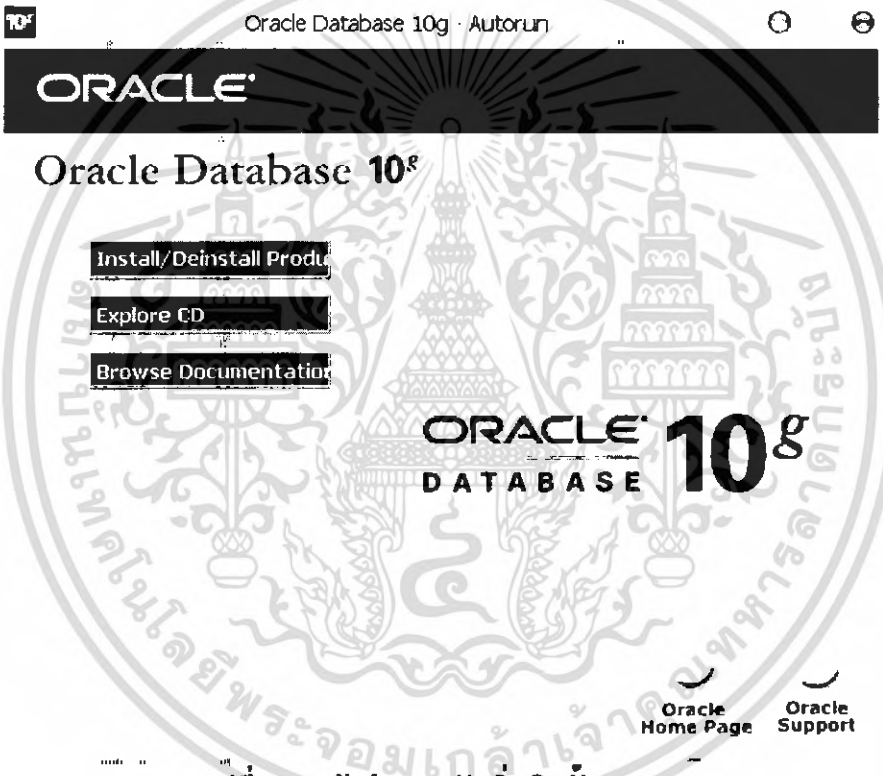
[8] Oracle Corporation. "Oracle Database Online Documentation 10g Release 2 (10.2)" [Online].
Available : http://www.oracle.com/pls/db102/homepage. 2006.

ภาคผนวก ก.

คู่มือการติดตั้งซอฟท์แวร์

1. การติดตั้ง Oracle Database Server 10g release 2 แบบ Basic

1.1 run ไฟล์ "setup.exe" เพื่อ start Oracle Universal In-staller หรือ รอให้ Autorun ทำงานซึ่งจะแสดงหน้าต่างดังภาพ ให้เลือก Install/Deinstall Product



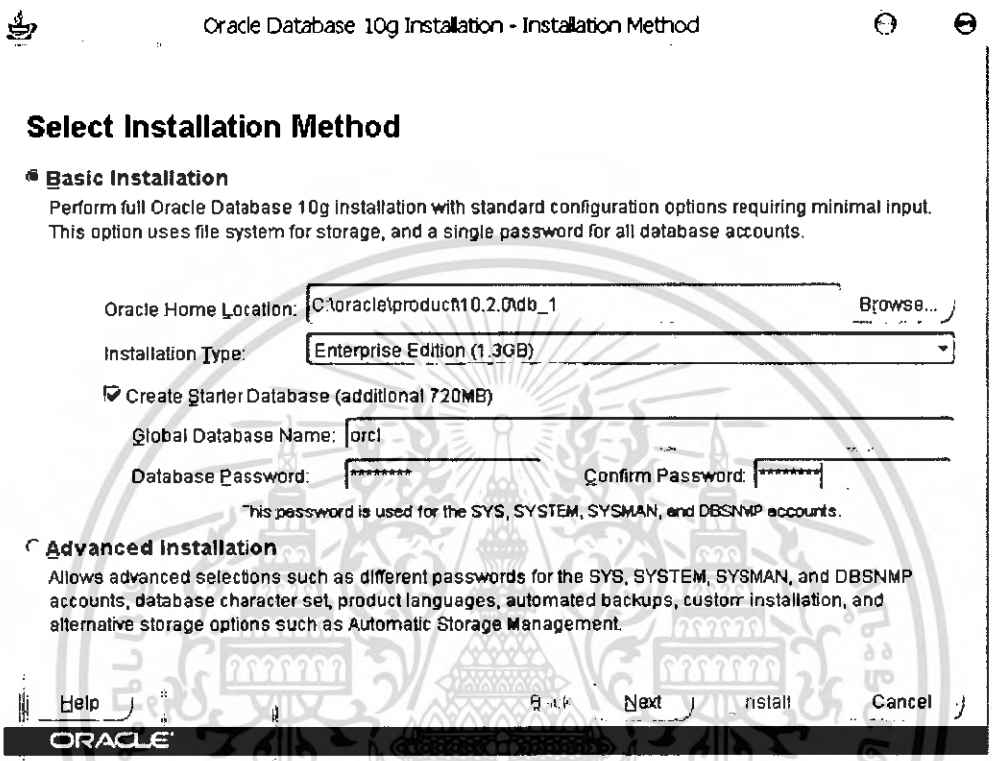
รูปที่ ก.1 หน้าต่างสำหรับเริ่มติดตั้ง Oracle

1.2 เข้าสู่ขั้นตอนการเลือก mode ของการ install ให้เลือก Basic Installation และในช่องของ Database Password และ Confirm Password ให้ใส่ Password ที่มีความยาวระหว่าง 4 ถึง 30 ตัวอักษร โดยสามารถมีสัญลักษณ์ต่อไปนี้ได้ คือ underscore(_), dollar($), pound sign(#) และ

- ห้ามขึ้นต้นด้วยตัวเลข
- ห้ามใส่ password ที่เหมือนกับ user name
- ห้ามใช้คำสงวนของ Oracle
- ห้ามใช้คำว่า change_on_install, sysman, dbsnmp เป็น password

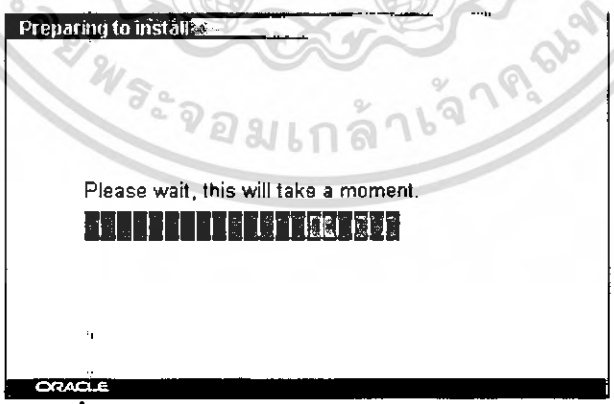
หมายเหตุ ในที่นี้ให้ใส่คำว่า password ทั้งในช่อง Database Password และ Confirm Password)

แล้ว click ปุ่ม Next



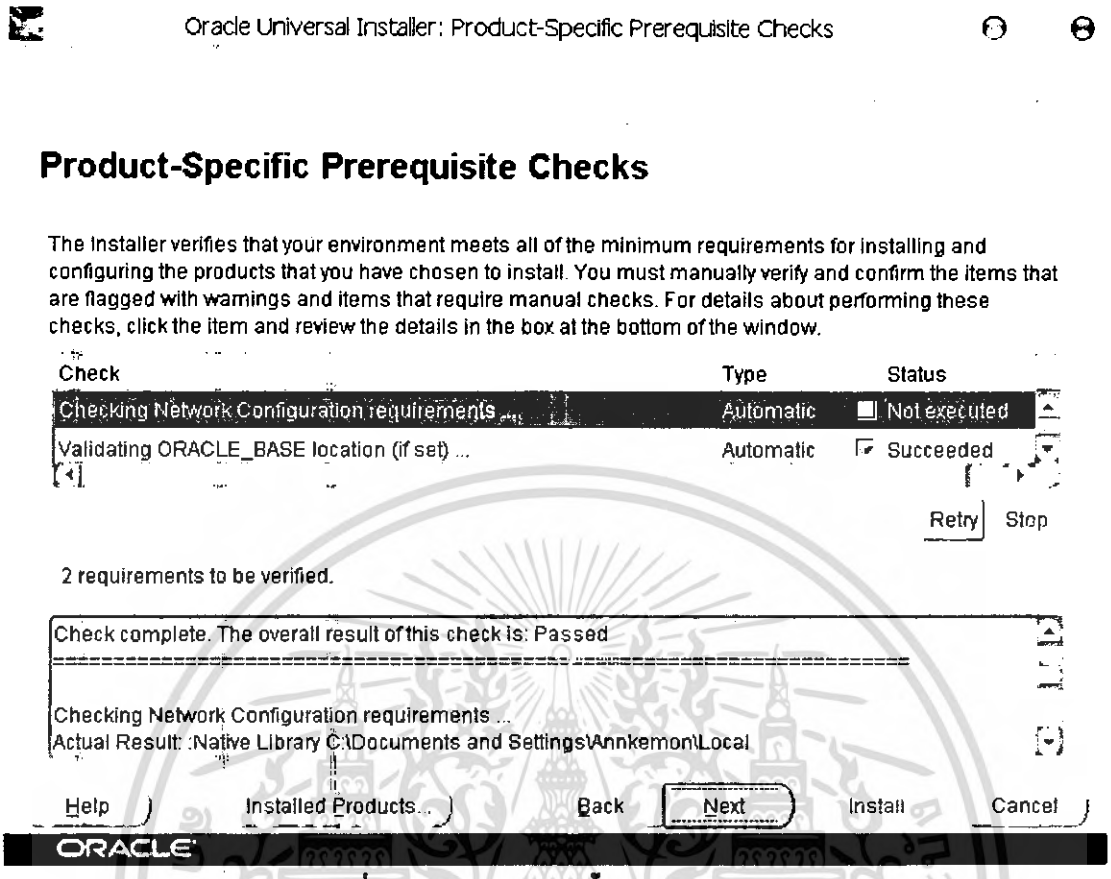
รูปที่ ก.2 หน้าต่างสำหรับเลือกรูปแบบ และ Path สำหรับติดตั้ง

หลังจากนั้นระบบจะเตรียมการ Install ดังรูป



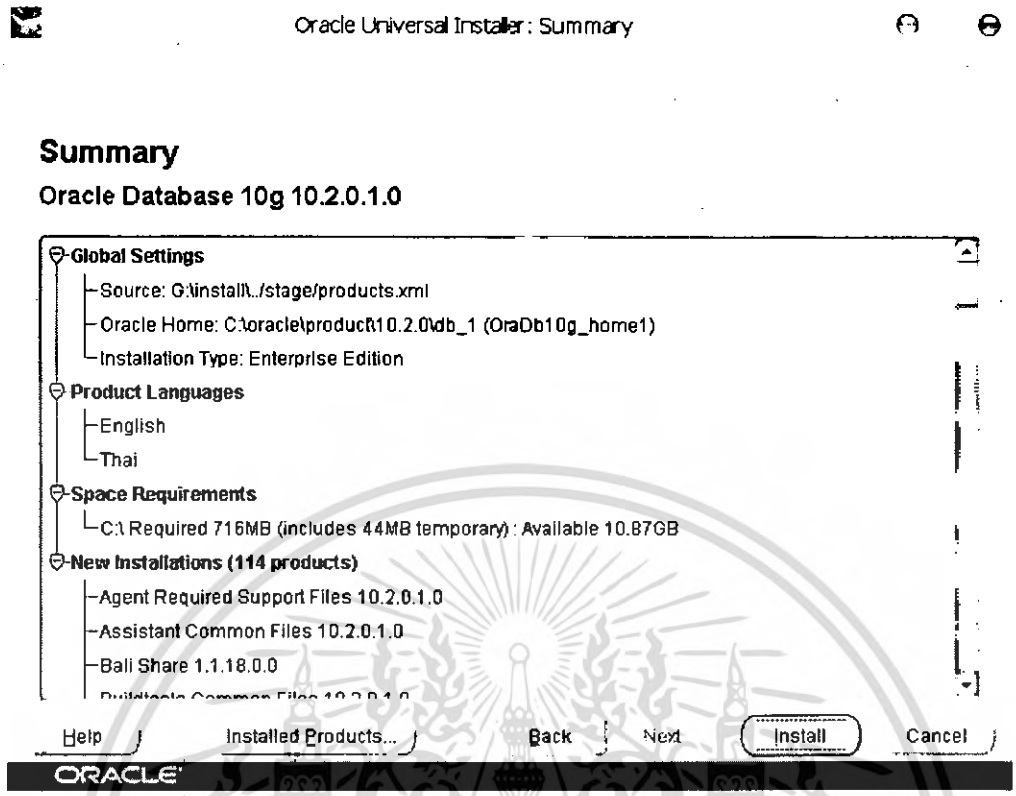
รูปที่ ก.3 หน้าต่างขณะ Oracle กำลังโหลดข้อมูล

1.3 เมื่อแสดงหน้าต่าง Product-Specific Prerequisite Checks ให้ click ปุ่ม next



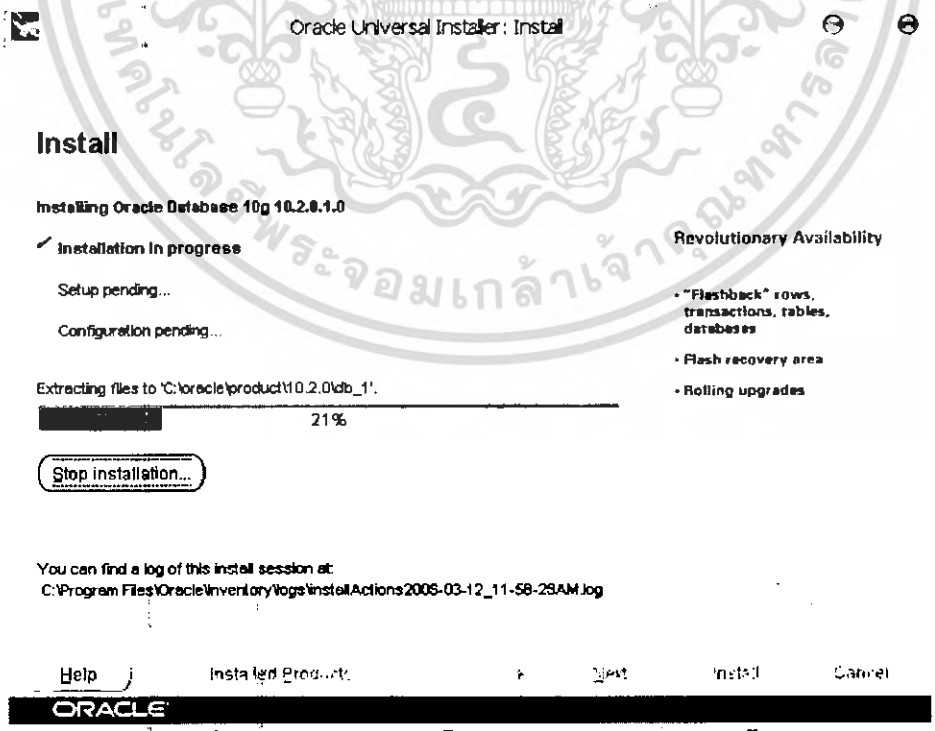
รูปที่ ก.4 หน้าต่างแสดงขั้นตอนการทำงาน

1.4 จากหน้าต่างสรุปผลให้เลือก Install



รูปที่ ก.5 หน้าต่างแสดงการตั้งค่าทั้งหมด พร้อมติดตั้ง

ขณะ install จะแสดงหน้าต่าง Install ดังรูป

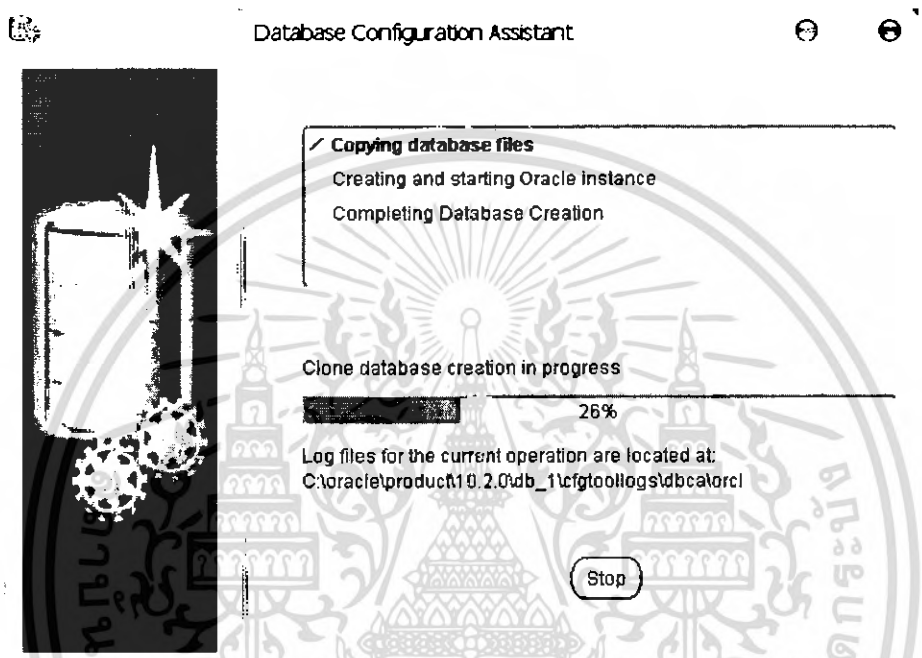


รูปที่ ก.6 หน้าต่างแสดงขั้นตอนการทำงานขณะติดตั้ง

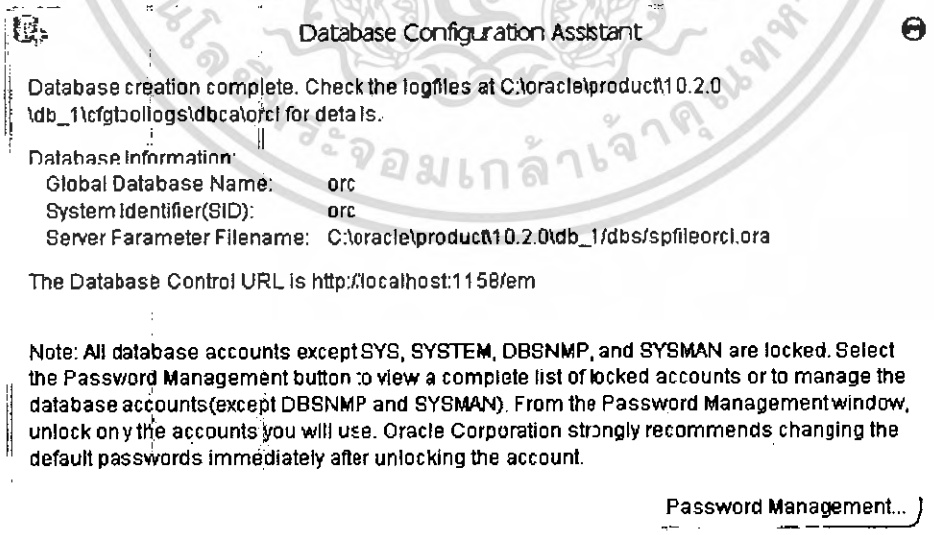
หลังจากนั้นจะแสดงหน้าต่าง Oracle Universal Installer : Configuration Assistants (ซึ่ง ผู้ติดตั้งไม่ต้องทำอะไร)

ต่อมาจะแสดงหน้าต่าง Database Configuration Assistant (ซึ่งผู้ติดตั้งไม่ต้องทำอะไร) ดังรูป

1.5 ในขั้นตอนต่อมาเป็นการแสดงรายละเอียดผลของการ install ให้ click ปุ่ม OK

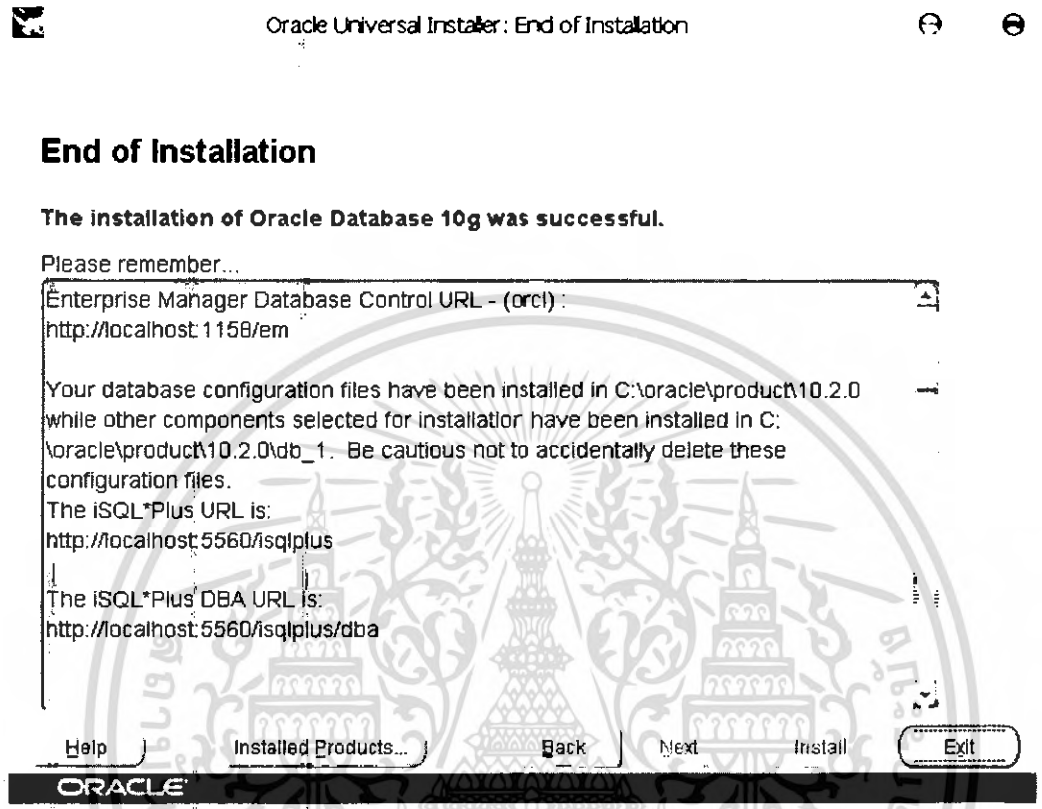


รูปที่ ก.7 หน้าต่างแสดงการทำงานขณะสร้างฐานข้อมูล



รูปที่ ก.8 หน้าต่างเมื่อเสร็จสิ้นการสร้างฐานข้อมูล

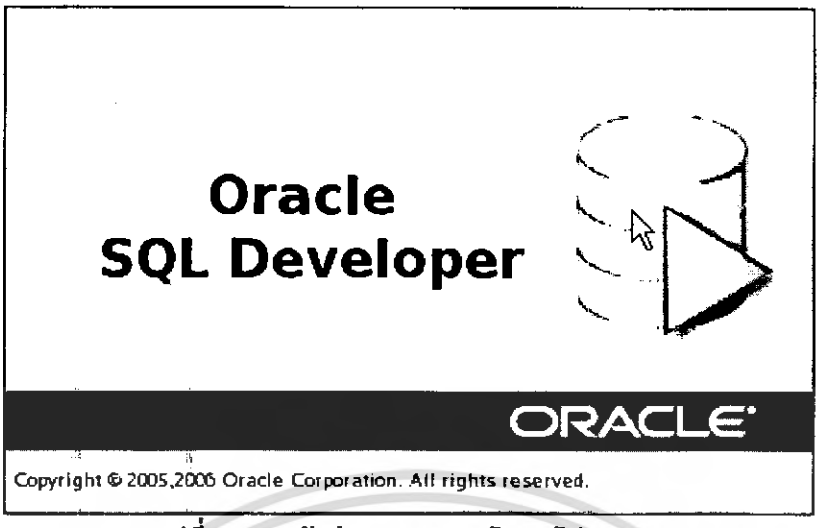
1.6 ในขั้นตอนสุดท้ายของการ install ให้ click ปุ่ม Exit



รูปที่ ก.9 หน้าต่างแสดงเสร็จสิ้นการติดตั้ง

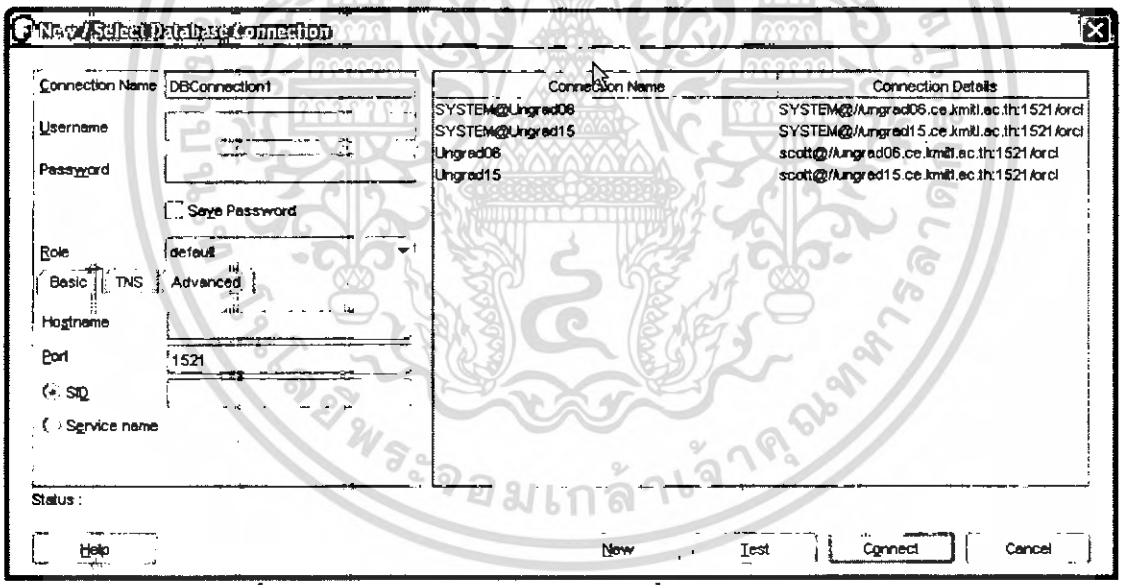
2. การติดตั้ง SQL Developer

SQL Developer เป็น ซอฟท์แวร์ของทาง Oracle ที่ช่วยในการใช้งานฐานข้อมูลได้ง่ายขึ้น ซึ่งพัฒนามาจาก Java ทำให้ไม่จำเป็นต้องมีการ Install เพียงดาวน์โหลดไฟล์จาก www.oracle.com มาแล้วทำการ Extract ไฟล์ที่ได้มา จากนั้นก็สั่งให้ไฟล์ sqldeveloper.exe ทำงานก็สามารถใช้งานซอฟท์แวร์ได้

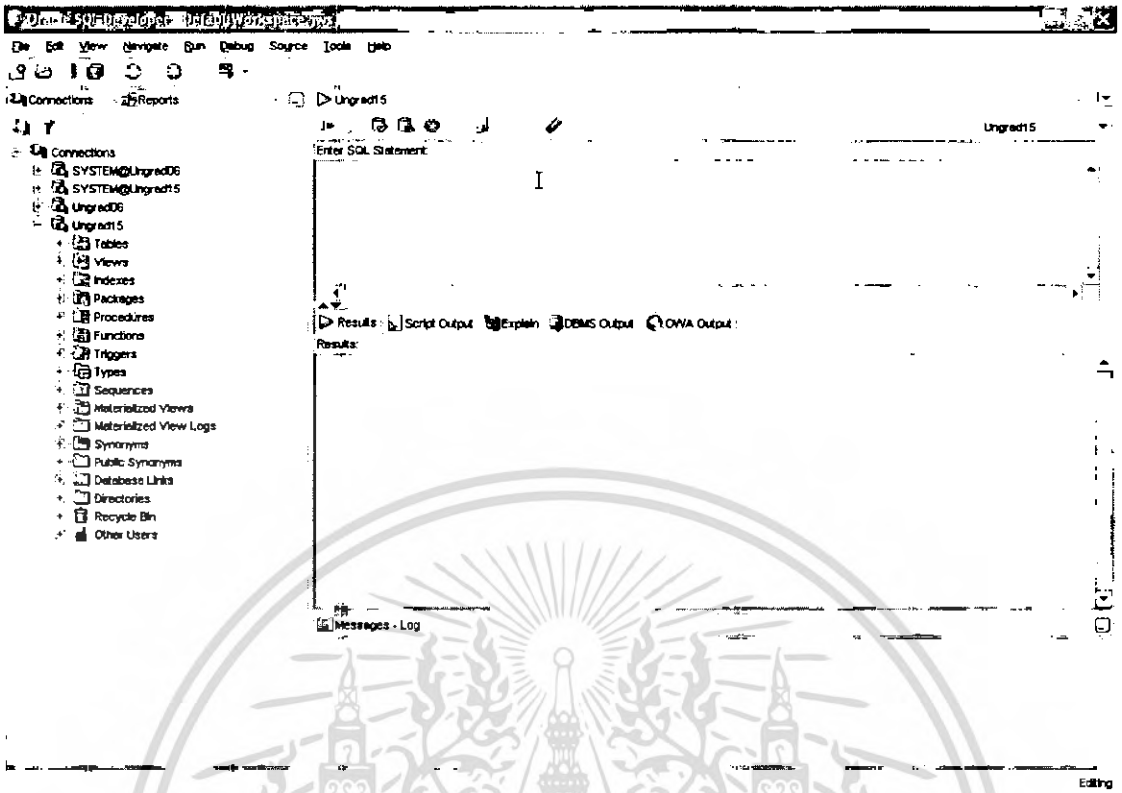


รูปที่ ก.10 หน้าต่างแสดงขณะโหลดโปรแกรม

ซึ่งในการใช้งานครั้งแรกจะเป็นต้องมีการตั้งค่าการเชื่อมต่อเพื่อให้ซอฟท์แวร์สามารถเชื่อมต่อไปยังฐานข้อมูลที่ต้องการได้อย่างถูกต้อง



รูปที่ ก.11 หน้าต่างแสดงการสร้างการเชื่อมต่อไปยังฐานข้อมูล



รูปที่ ก.12 หน้าต่างพร้อมสำหรับใช้งาน

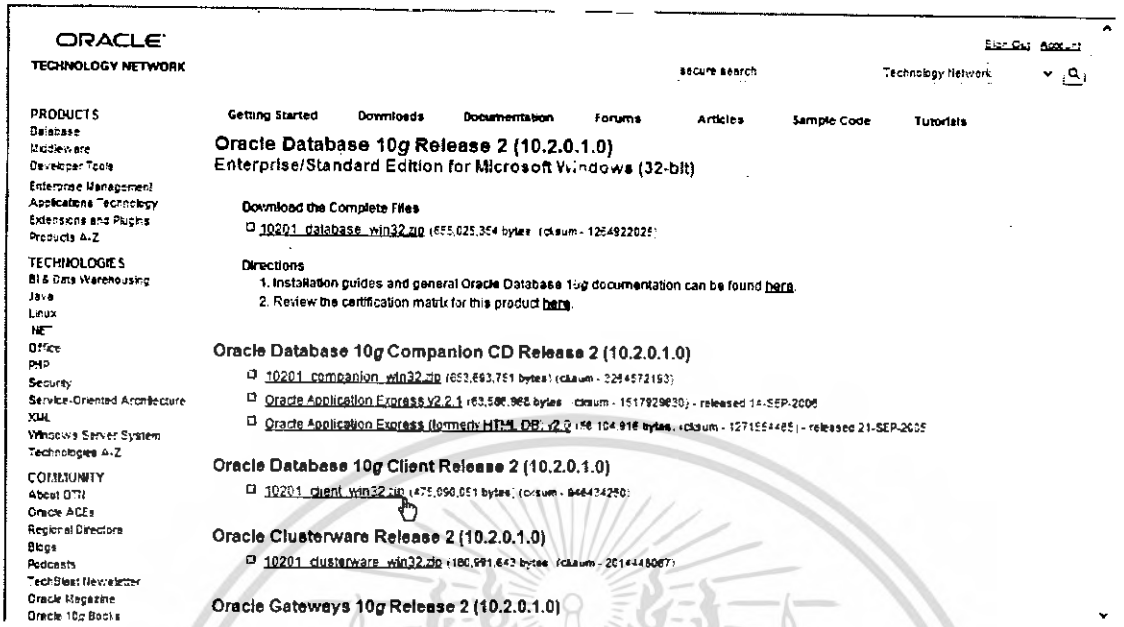
3. การติดตั้งซอฟท์แวร์ Oracle Database Client

ความต้องการของระบบที่จะทำการติดตั้ง

Requirement	**Minimum Value**
Physical memory (RAM)	256 MB minimum, 512 MB recommended
Virtual memory	Double the amount of RAM
Hard disk space	Total ranges from 216–738 MB.
Video adapter	256 colors
Processor	550 MHz minimum

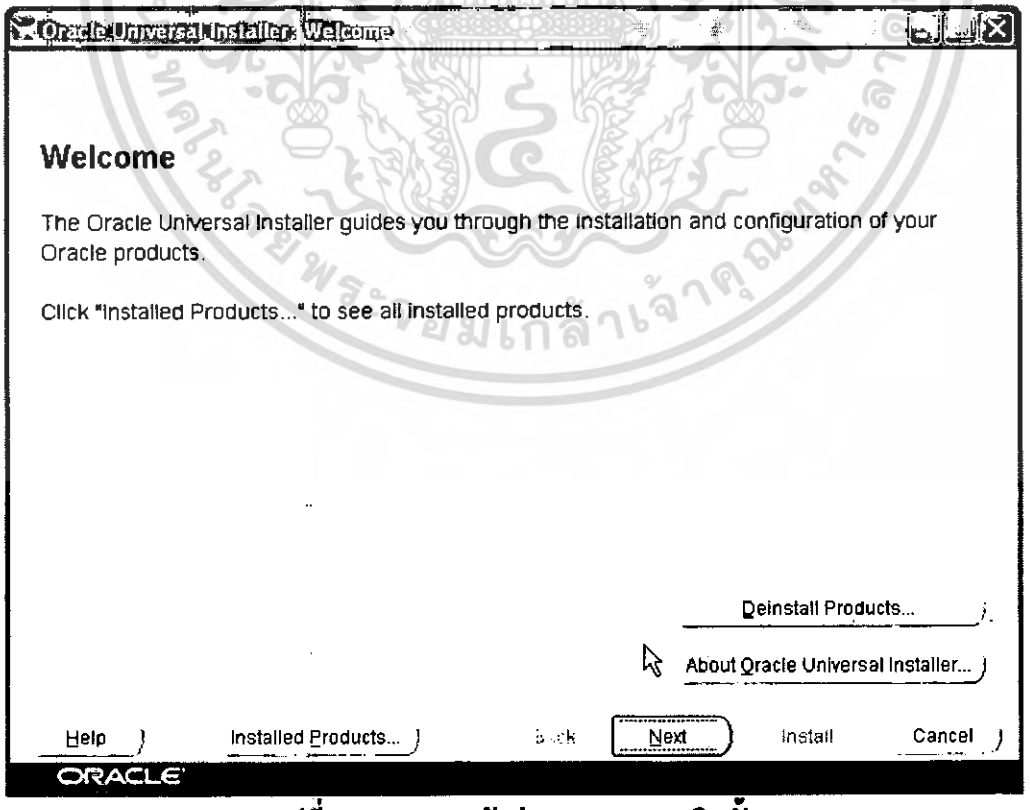
1. ดาวโหลดไฟล์ที่ต้องใช้ในการติดตั้งจาก หน้าเว็บไซต์

   http://www.oracle.com/technology/software/ เลือก Database 10g Enterprise/Standard Editions และทำการเลือก Oracle Database 10g Client Release 2 (10.2.0.1.0)



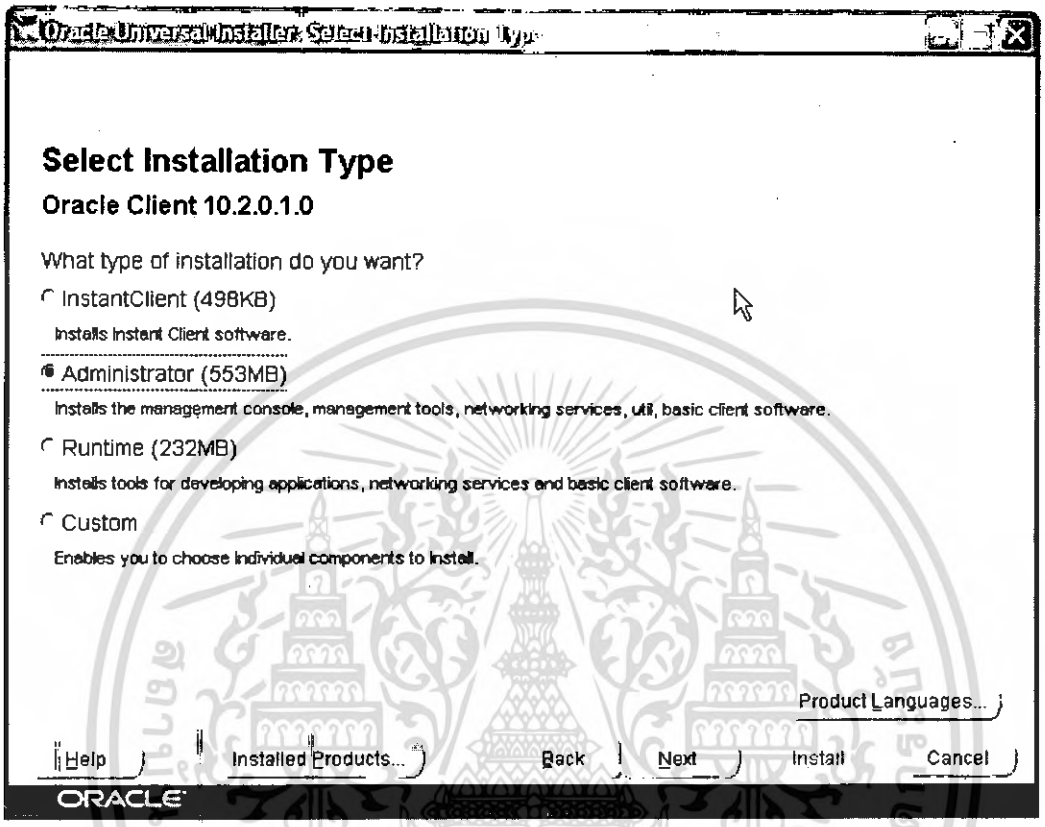
รูปที่ ก.13 แสดงหน้าเว็บสำหรับดาวน์โหลด Client

2. เมื่อดาวโหลดไฟล์เรียบร้อยแล้วทำการคลายการบีบอัดไฟล์ ไว้ในโฟลเดอร์เพื่อให้พร้อมสำหรับการติดตั้ง และ ดับเบิลคลิกไฟล์ Setup.exe



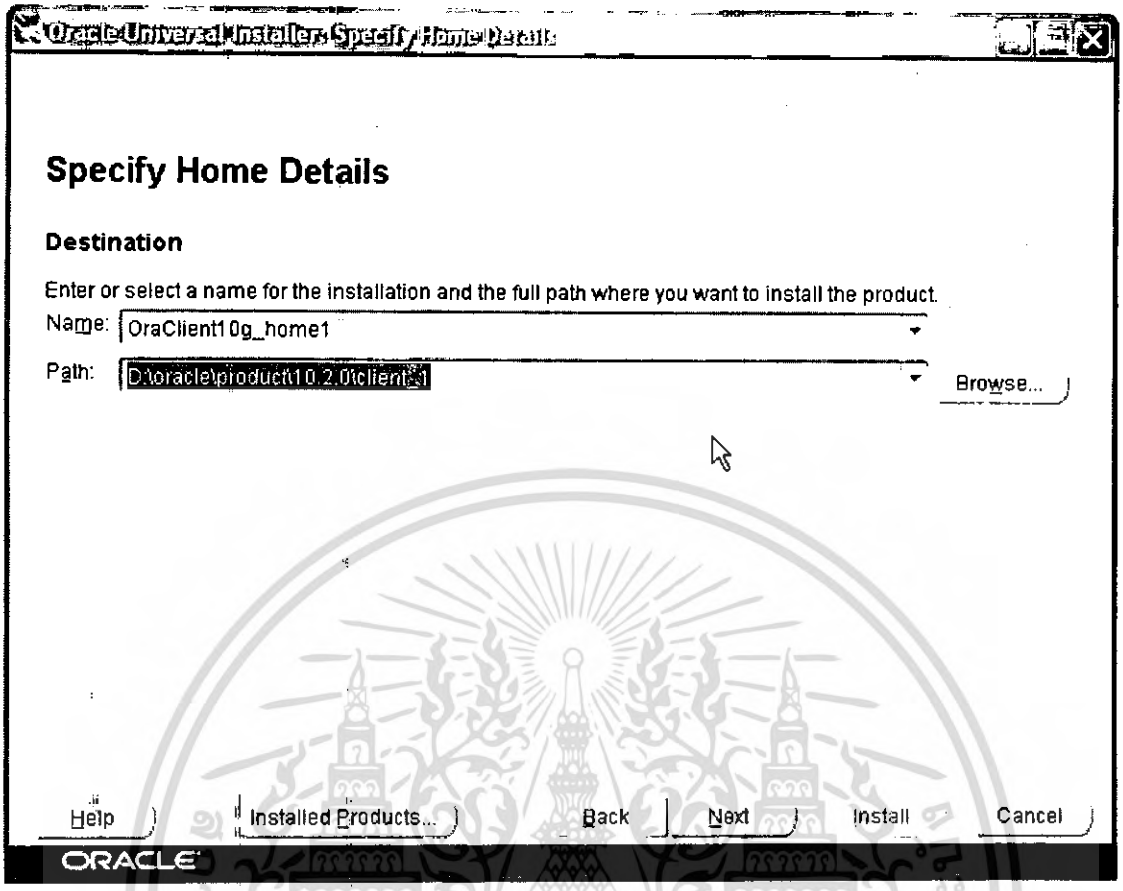
รูปที่ ก.14 แสดงหน้าต่างแรกของการติดตั้ง

3. ในหน้าต่างแรกของการติดตั้ง กด Next
4. เมื่อโปรแกรมติดตั้งทำงานเสร็จเรียบร้อย ให้เลือกรูปแบบการติดตั้งเป็น Administrator



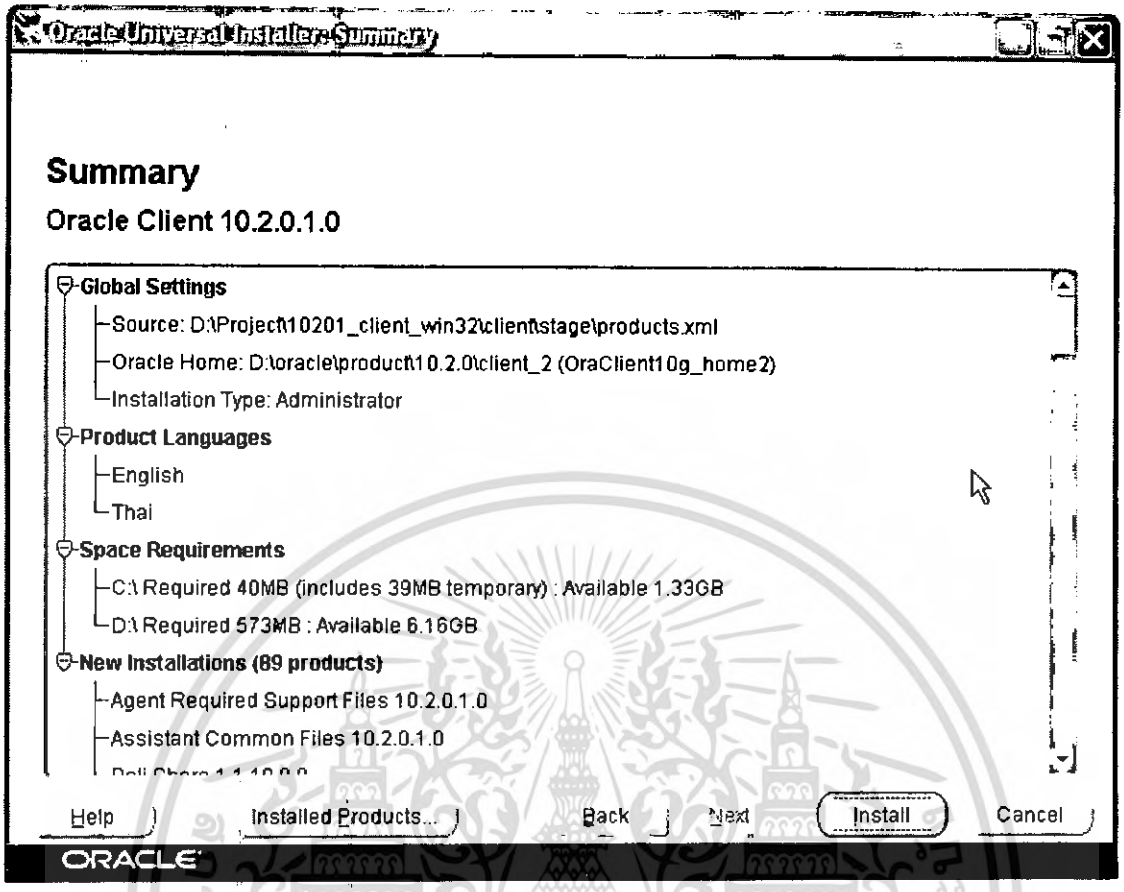
รูปที่ ก.15 แสดงการเลือกรูปแบบการติดตั้งเป็น Administrator

5. ขั้นต่อมาเป็นการเลือกสถานที่ที่ต้องการติดตั้งซอฟท์แวร์ เมื่อตั้งค่าเสร็จกด Next



รูปที่ ก.16 แสดงหน้าต่างสำหรับตั้งค่าการติดตั้ง

6. หน้าต่างต่อมาเป็นหน้าต่างสำหรับตรวจสอบความต้องการของระบบที่กำลังจะติดตั้ง ให้กด Next
7. เป็นหน้าต่างแสดงรายงานสุดท้ายก่อนเริ่มติดตั้ง ให้กด Install



รูปที่ ก.17 แสดงสรุปรายละเอียดก่อนติดตั้งจริง

8. เมื่อทำการติดตั้งเรียบร้อยแล้ว ซอฟท์แวร์จะทำการแนะนำให้ตั้งค่าสำหรับการเชื่อมต่อกับฐานข้อมูล (Oracle Net Configuration Assistant) ซึ่งถ้ามีการตั้งค่าไว้ตอนลงฐานข้อมูลแล้วก็ไม่จะเป็นต้องตั้งค่าส่วนนี้อีก

ภาคผนวก ข.

คู่มือการทำระบบฐานข้อมูลแบบกระจาย

1. ขั้นตอนการทำระบบฐานข้อมูลแบบกระจายด้วย Oracle 10g

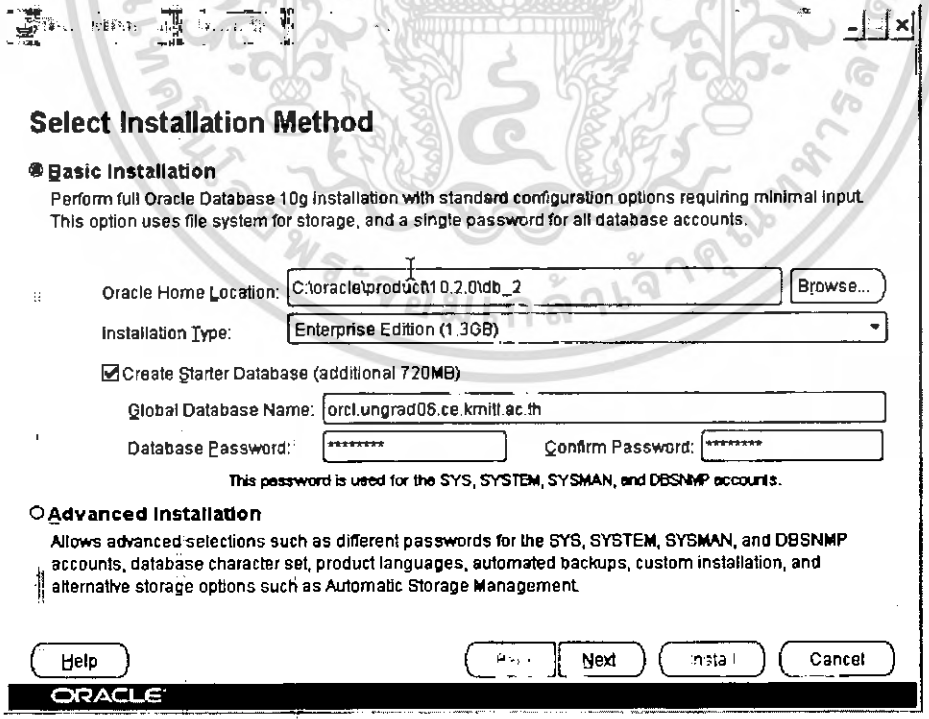
กำหนดให้มีเซิร์ฟเวอร์อยู่ 2 เครื่อง คือ UNGRAD06.CE.KMITL.AC.TH และ UNGRAD15.CE.KMITL.AC.TH ถ้าจะทำให้เป็นระบบฐานข้อมูลแบบกระจาย มีขั้นตอนดังนี้

1. กรณีที่ใช้ซอฟท์แวร์ของ Oracle ทั้งหมด การที่จะทำให้ฐานข้อมูลแต่ละที่สามารถเชื่อมต่อกันได้นั้น ขั้นตอนแรกต้องมีการตั้งค่าการใช้งานชื่อของฐานข้อมูลร่วมกับชื่อของเซิร์ฟเวอร์ที่ฐานข้อมูลตั้งอยู่ด้วย ซึ่งเรียกว่า Global Name ซึ่งสามารถทำการกำหนดได้ในขั้นตอนของการติดตั้งซอฟท์แวร์ Oracle

   จากข้อกำหนด จะให้ฐานข้อมูล ชื่อ ORCL เมื่อรวมกับชื่อเซิร์ฟเวอร์จะได้ Global Name คือ

   ORCL.UNGRAD06.CE.KMITL.AC.TH และ

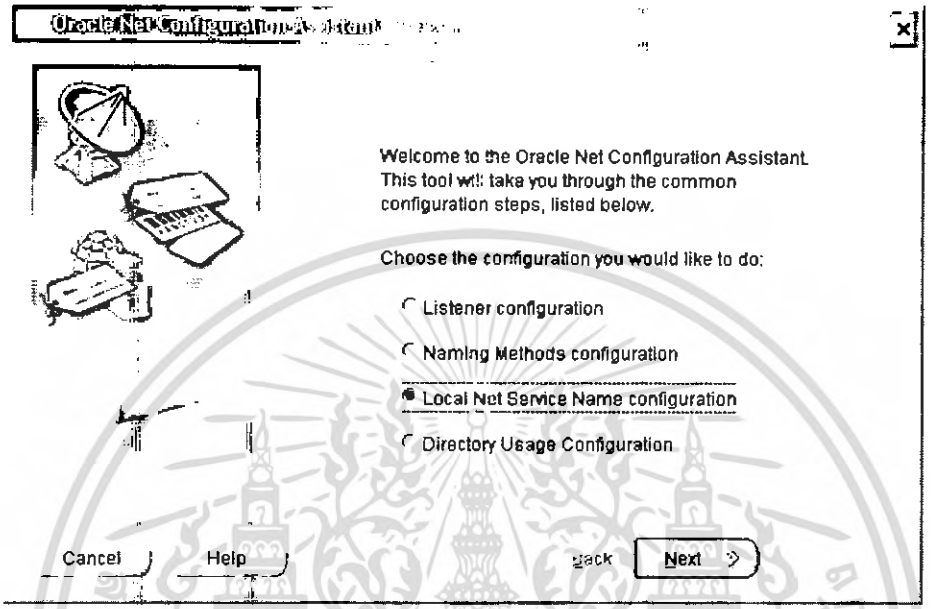
   ORCL.UNGRAD15.CE.KMITL.AC.TH



รูปที่ ข.1 การตั้งค่า Global Name ของเซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH

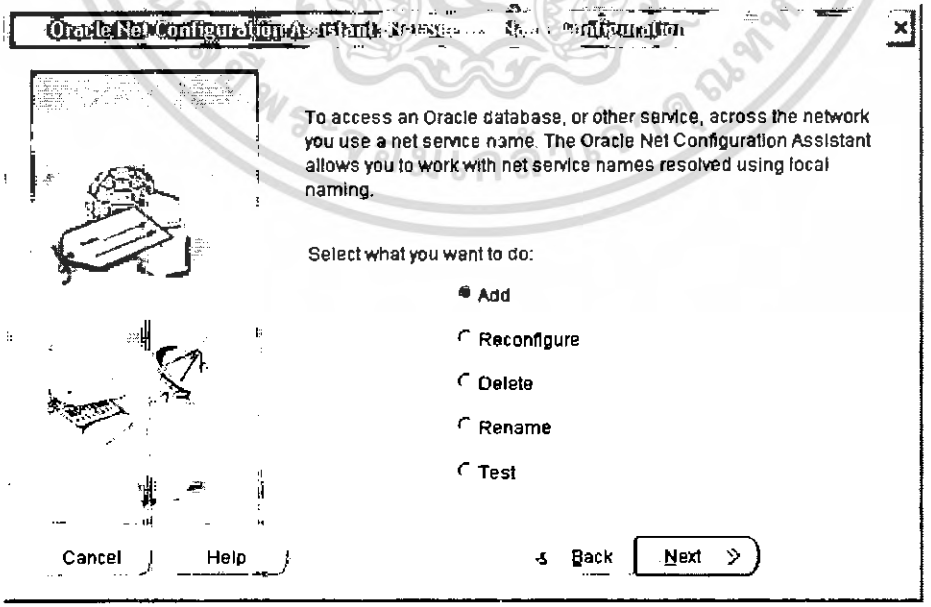
2. ทำการสร้างการเซอร์วิสในการเชื่อมต่อ (Net Service) ใช้ในการสร้าง Database Link เพื่อทำให้เซิร์ฟเวอร์สามารถเชื่อมต่อกันได้ โดยจะใช้ซอฟท์แวร์เสริมของ Oracle ซึ่งก็คือ Net Configuration Assistant สร้างขึ้นโดยมีขั้นตอนดังนี้

2.1 เลือกหัวข้อ Local Net Service Name configuration เพื่อสร้าง Net Service



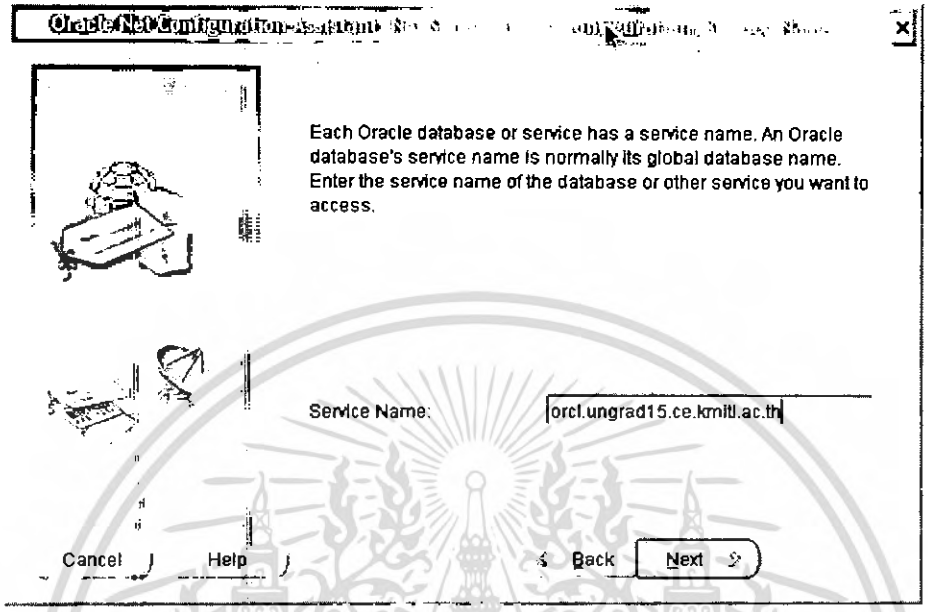
รูปที่ ข.2 การเลือกหัวข้อที่จะปรับแต่ง Net Service

2.2 เลือกหัวข้อ Add เพื่อทำการสร้าง Net Service ขึ้นมาใหม่



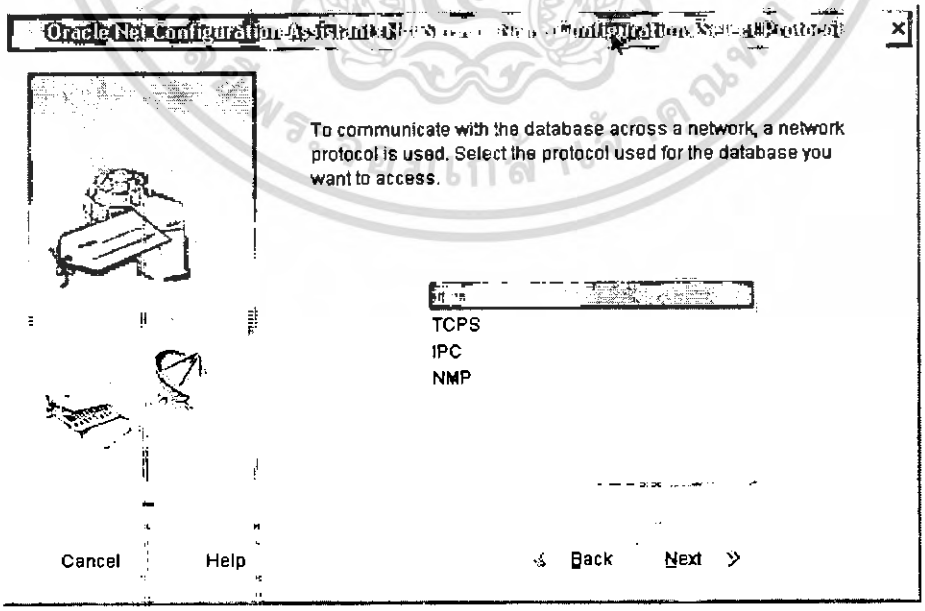
รูปที่ ข.3 การเลือกหัวข้อ Add เพื่อสร้าง Net Service ใหม่

2.3 ใส่ชื่อ Service Name ซึ่งต้องระบุชื่อฐานข้อมูลตามด้วยชื่อของเซิร์ฟเวอร์ที่จะทำการเชื่อมต่อด้วย เช่น ORCL.UNGRAD15.CE.KMITL.AC.TH



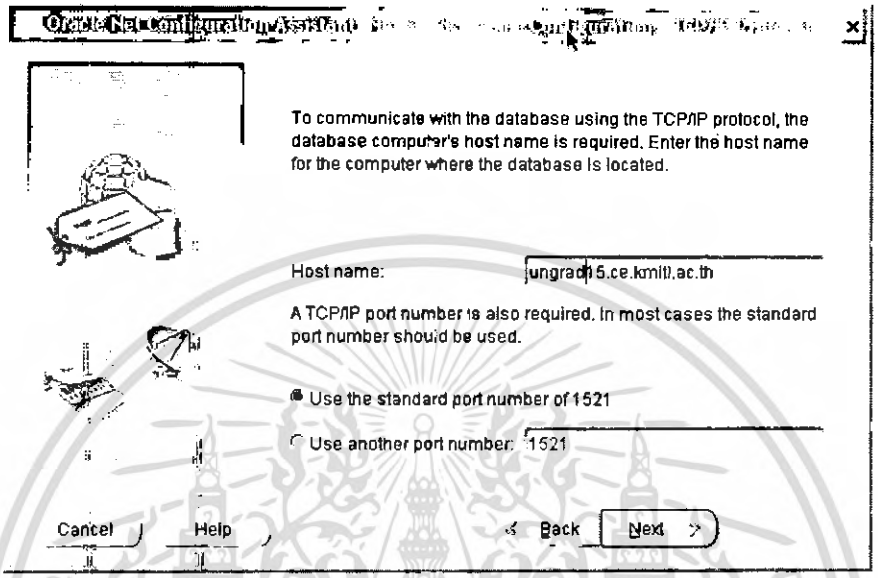
รูปที่ ข.4 การตั้งค่า Service Name

2.4 เลือก Network Protocol ที่ใช้ในการเชื่อมต่อระหว่างฐานข้อมูลระหว่างเซิร์ฟเวอร์ ในที่นี้เลือกแบบ TCP



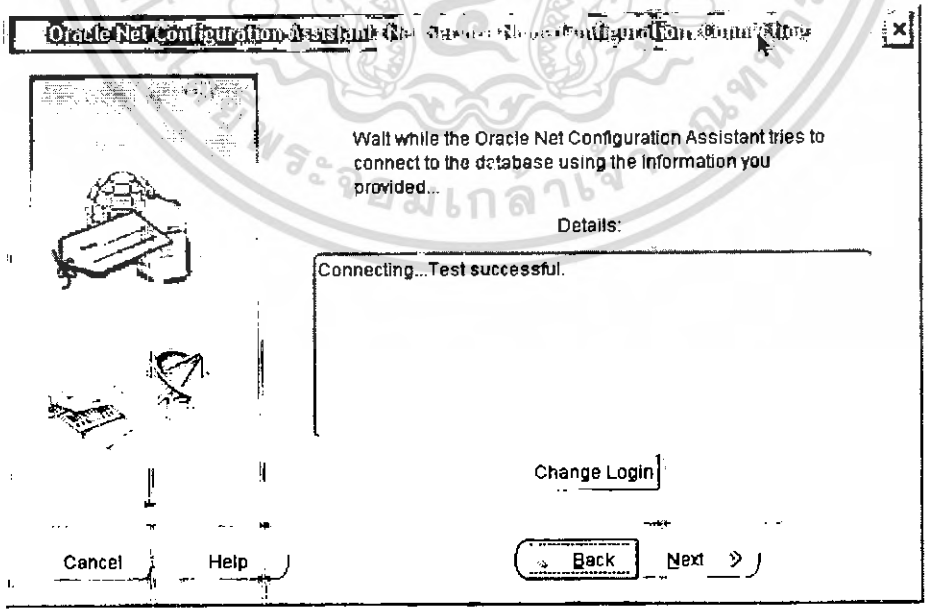
รูปที่ ข.5 การเลือก Network Protocol

2.5 ใส่ชื่อเซิร์ฟเวอร์ที่จะทำการเชื่อมต่อด้วย พร้อมทั้งระบุพอร์ตที่ใช้เชื่อมต่อ ซึ่งเซิร์ฟเวอร์ที่จะทำการเชื่อมต่อด้วยคือ UNGRAD15.CE.KMITL.AC.TH และพอร์ตที่ใช้คือ 1521 ซึ่งเป็นพอร์ตมาตรฐาน



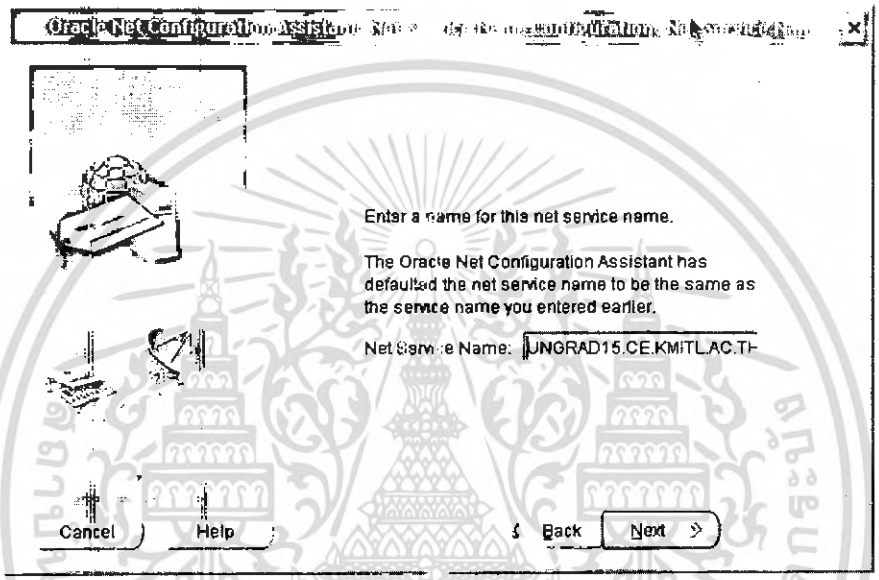
รูปที่ ข.6 การกำหนดค่า Host name และ Port

2.6 ทำการทดสอบการเชื่อมต่อว่าค่าที่ได้กำหนดไว้สามารถทำการเชื่อมต่อได้หรือไม่



รูปที่ ข.7 การทดสอบการเชื่อมต่อ

2.7 กำหนด Net Service Name คือ UNGRAD15.CE.KMITL.AC.TH เพื่อใช้ในการสร้าง Database Link ต่อไป ซึ่งจะได้ Net Service ที่ใช้เชื่อมต่อจากเซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH ไปยัง UNGRAD15.CE.KMITL.AC.TH และต้องสร้าง Net Service อีกตัวหนึ่งเพื่อใช้เชื่อมต่อจากเซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH ไปยัง UNGRAD15.CE.KMITL.AC.TH ด้วย โดยใช้วิธีในลักษณะเดียวกันนี้



รูปที่ ข.8 การกำหนด Net Service Name

3. ต่อไปเป็นขั้นตอนการสร้าง Database link ซึ่งสามารถสร้างได้หลายวิธีเช่น การใช้คำสั่งภาษา SQL ในการสร้าง หรือสามารถสร้างได้โดยการใช้ซอร์ฟแวร์เสริมของ Oracle ที่ชื่อว่า Oracle Enterprise Manager Console เพื่อช่วยในการตั้งค่าต่างๆของระบบโดยไม่ต้องพิมพ์คำสั่งเอง

ตัวอย่างการสร้าง Database Link โดยใช้คำสั่งภาษา SQL โดย Database Link ตัวอย่างนี้จะเป็นการสร้างไว้ที่เซิร์ฟเวอร์ UNGRAD15.CE.KMITL.AC.TH เพื่อทำให้สามารถเชื่อมต่อกับเซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH ได้ โดยที่

ชื่อของ Database link คือ ORCL.UNGRAD06.CE.KMITL.AC.TH

และใช้ Service Name คือ UNGRAD06.CE.KMITL.AC.TH ที่ได้สร้างเอาไว้แล้ว

```
CREATE DATABASE LINK ORCL.UNGRAD06.CE.KMITL.AC.TH USING
'UNGRAD06.CE.KMITL.AC.TH';
```

ตัวอย่างคำสั่ง SQL ในการสร้าง Database Link เพื่อใช้เชื่อมต่อฐานข้อมูลที่เซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH ไปยัง UNGRAD15.CE.KMITL.AC.TH โดยที่

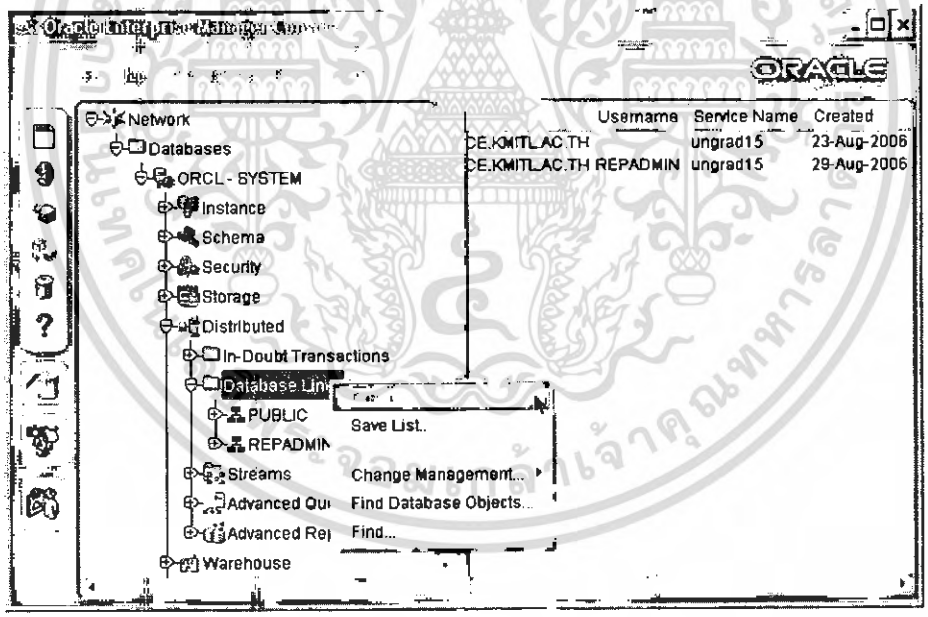
ชื่อของ Database link คือ ORCL.UNGRAD15.CE.KMITL.AC.TH

และใช้ Service Name คือ UNGRAD15.CE.KMITL.AC.TH ที่ได้สร้างเอาไว้แล้ว

```
CREATE DATABASE LINK ORCL.UNGRAD15.CE.KMITL.AC.TH USING
'UNGRAD15.CE.KMITL.AC.TH';
```

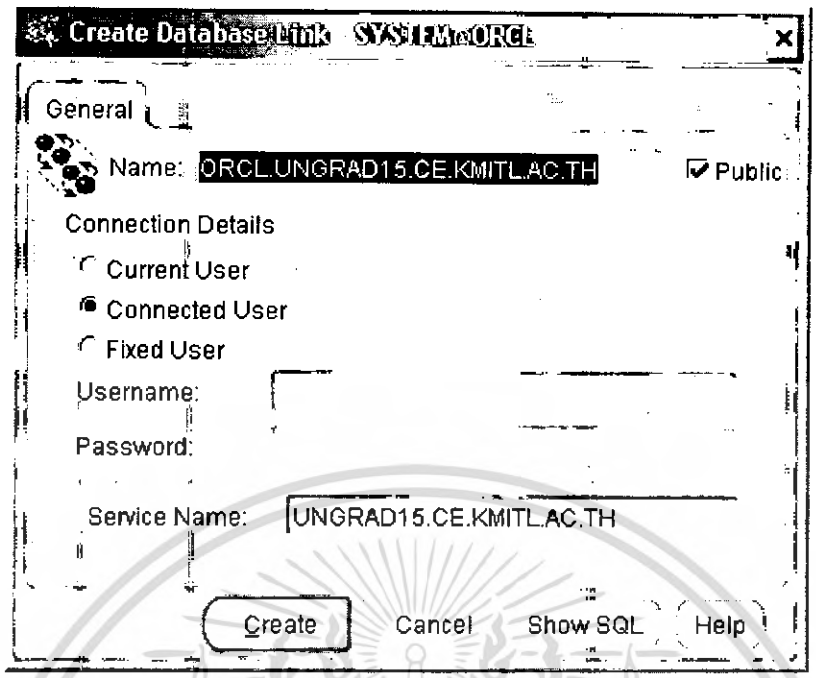
หรืออาจจะใช้ซอฟท์แวร์ Oracle Enterprise Manager Console ซึ่งเป็นส่วนหนึ่งของซอฟท์แวร์ Oracle Database Client ช่วยสร้าง Database link โดยที่เราไม่ต้องพิมพ์คำสั่งเอง ซึ่งมีขั้นตอนดังนี้

1. เข้าไปที่ส่วนที่เป็น Database Links ภายในหัวข้อ Distributed ซึ่งอยู่ในฐานข้อมูลที่เราได้กำหนดไว้ แล้วกด mouse ปุ่มขวาเลือกเมนู Create แสดงได้ดังรูป



รูปที่ ข.9 การเลือกเมนู Create เพื่อสร้าง Database Link

2. เมื่อขึ้นหน้าต่าง Create Database Link ก็ให้กำหนดค่าที่ต้องการลงไป โดยในตัวอย่างนี้จะเป็นการสร้าง Database Link ที่เป็นแบบ Public, Connection เป็น Connected User และให้เป็นการเชื่อมต่อฐานข้อมูลจากเซิร์ฟเวอร์ UNGRAD06.CE.KMITL.AC.TH ไปยัง UNGRAD15.CE.KMITL.AC.TH ซึ่งจะแสดงค่าต่างๆ ได้ดังรูป



รูปที่ ข.10 กำหนดค่าต่างๆ ในการสร้าง Database Link

พอถึงขั้นตอนนี้ก็จะได้ระบบฐานข้อมูลแบบกระจายโดยที่การอ้างถึงข้อมูลยังต้องทำการอ้างชื่อเซิร์ฟเวอร์ด้วย สำหรับในการทำระบบฐานข้อมูลให้ได้ตามระดับสถาปัตยกรรมในระดับต่างๆ นั้นก็ได้บอกวิธีการทำไว้แล้วในบทที่ 3

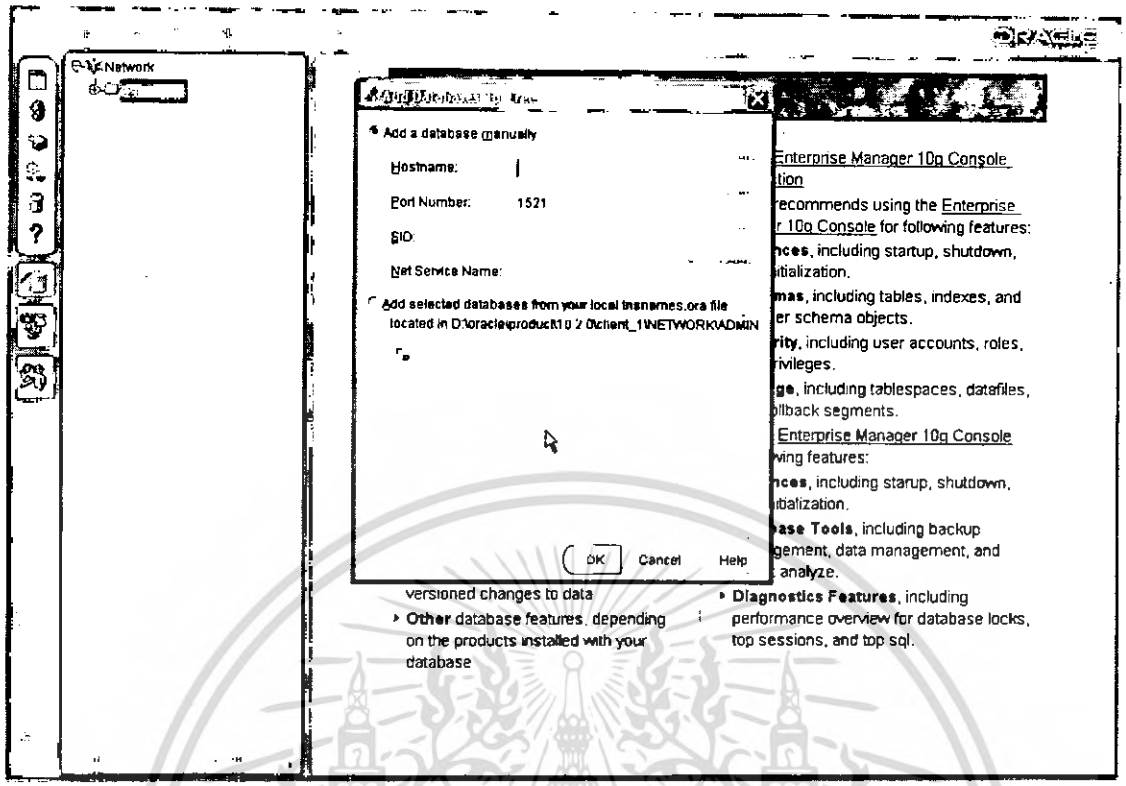
2. การทำ Replication โดยใช้ Oracle Enterprise Manager Console

1. ก่อนทำการตั้งค่าการทำ Replication นั้น Oracle แนะนำให้สร้าง User ขึ้นมาดูการทำ Replication โดยเฉพาะ ซึ่งทำให้สามารถควบคุมดูแลได้ง่ายขึ้น ซึ่งจะต้องทำการสร้าง User นี้ในทุกๆฐานข้อมูล โดยสามารถดูตัวอย่างคำสั่งที่ใช้ในการสร้าง User สำหรับดูแล Replication ได้จากด้านล่าง

```
CONNECT / AS SYSDBA

-- Ensure the databse global name is correctly set
ALTER DATABASE RENAME global_name TO site1.world;

-- Create public DB links to all replication sites
```

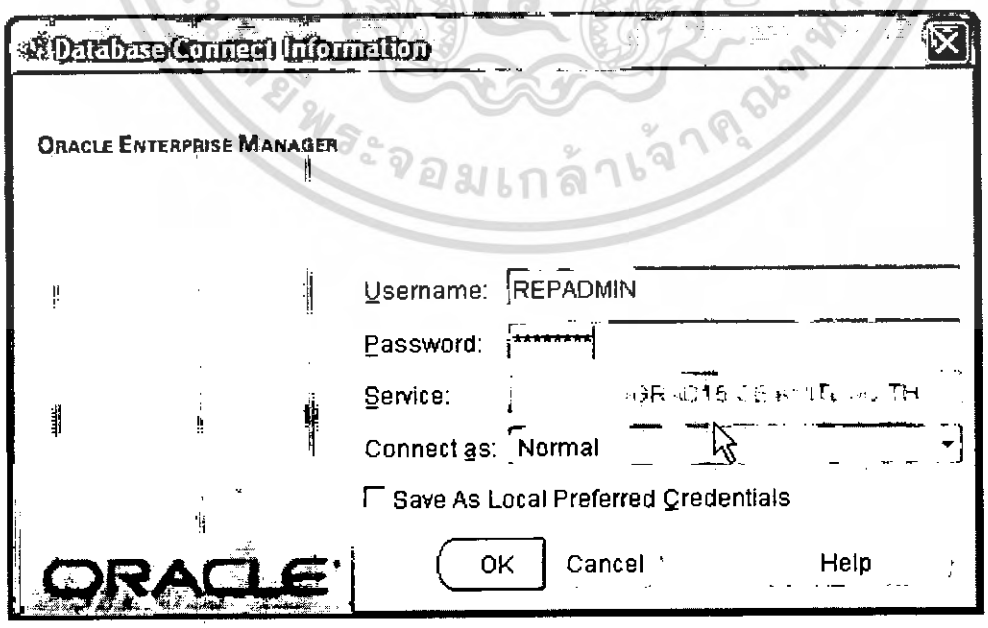
```
CREATE PUBLIC DATABASE LINK site2.world USING 'site2.world';

-- Create replication administrator / propagator / receiver
CREATE USER repadmin IDENTIFIED BY repadmin
 DEFAULT   TABLESPACE users
 TEMPORARY TABLESPACE temp
 QUOTA UNLIMITED ON users;

-- Grant privs to the propagator, to propagate changes to remote sites
EXECUTE Dbms_Defer_Sys.Register_Propagator(username=>'REPADMIN');

-- Grant privs to the receiver to apply deferred transactions
GRANT EXECUTE ANY PROCEDURE TO repadmin;

-- Authorize the administrator to administer replication groups and schemas
EXECUTE Dbms_Repcat_Admin.Grant_Admin_Any_Repgroup('REPADMIN');
EXECUTE Dbms_Repcat_Admin.Grant_Admin_Any_Schema (username => 'REPADMIN');


-- Authorize the administrator to lock and comment tables
GRANT LOCK ANY TABLE TO repadmin;
GRANT COMMENT ANY TABLE TO repadmin;
```

2. หลังจากสร้างผู้ใช้งานที่เป็นผู้ดูแลการทำ Replication แล้ว ขั้นต่อมาเปิดโปรแกรม Oracle Enterprise Manager Console ซึ่งได้หลังจากลง Oracle Client แบบ Administrator ซึ่งถ้าเป็นการใช้งานครั้งแรก ต้องกำหนดฐานข้อมูลที่ให้ซอฟท์แวร์ติดต่อได้โดยการคลิกขวาที่ database เลือก Add Database To Tree ... แล้วกำหนดฐานข้อมูลที่ต้องการ



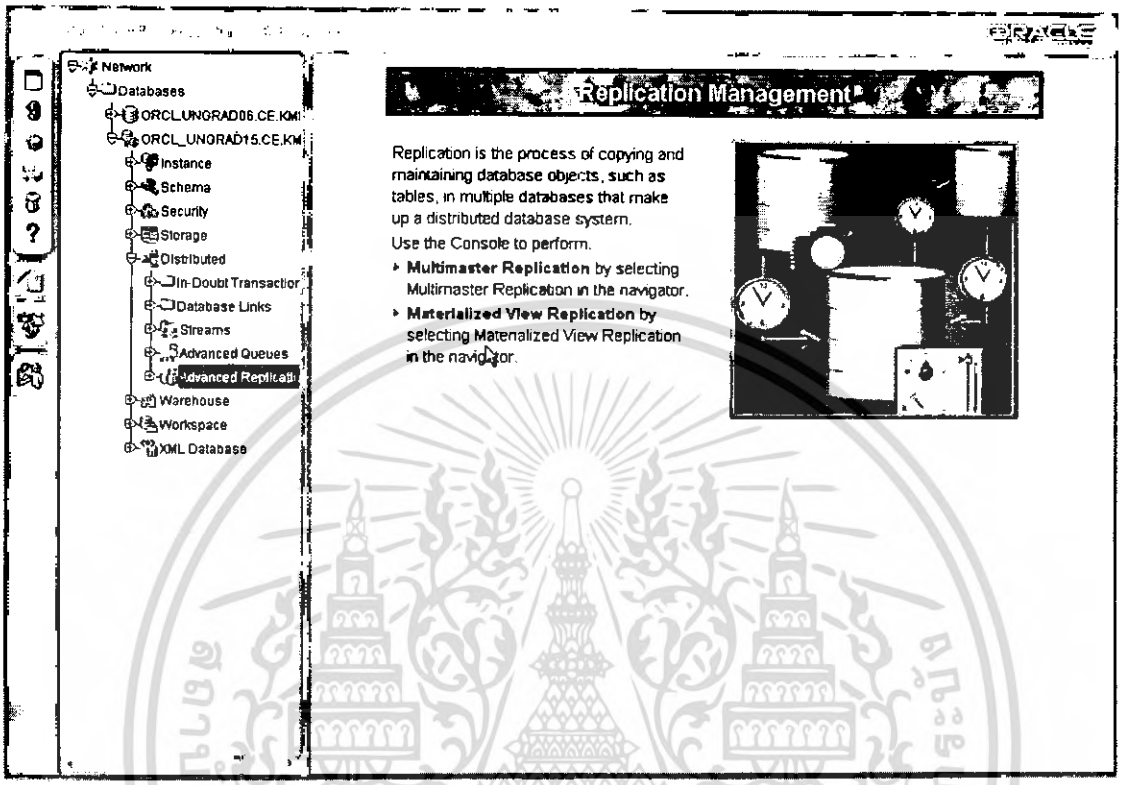
รูปที่ ข.11 แสดงหน้าต่างการตั้งค่าฐานข้อมูลที่ต้องการจัดการ

3. เมื่อตั้งค่าฐานข้อมูลแล้ว ทำการ Login เข้าสู่ฐานข้อมูล โดยใช้ User ที่สร้างมาดูแล Replication โดยเฉพาะ



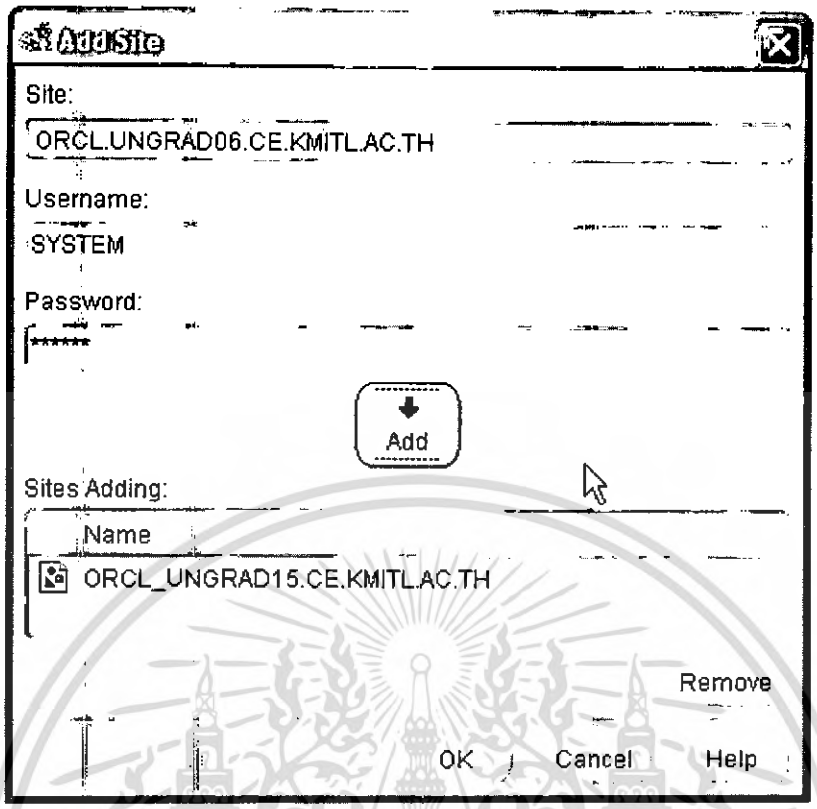
รูปที่ ข.12 แสดงการใส่ข้อมูลเพื่อ Login เข้าสู่ฐานข้อมูล

4. เลือก Advanced Replication จากนั้นคลิกขวาที่ Multimaster Replication เลือก Set Master Sites



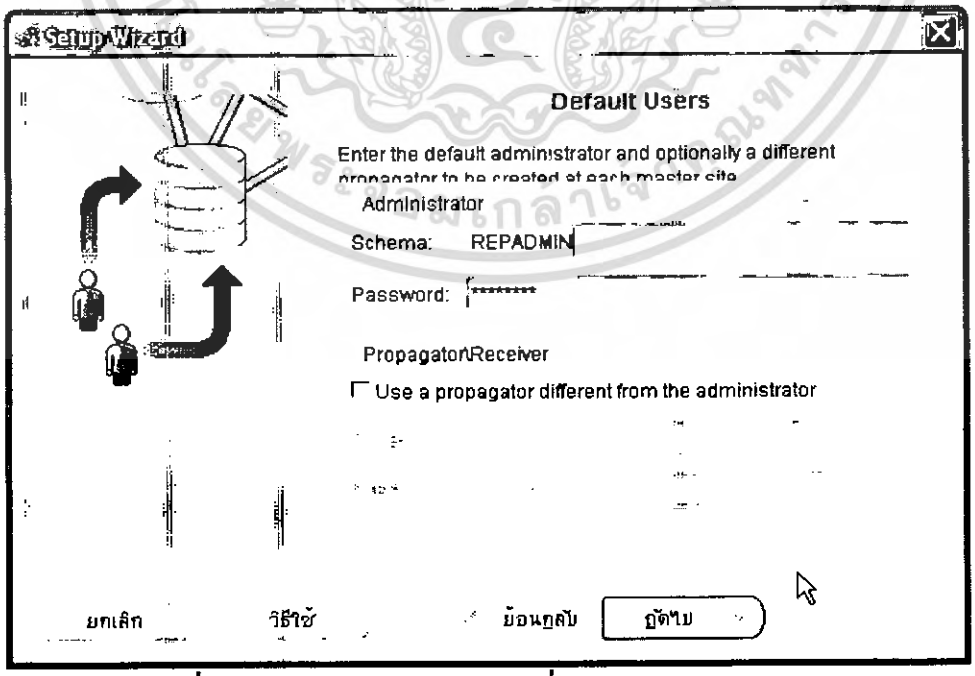
รูปที่ ข.13 แสดงการเลือกใช้งานส่วน Advanced Replication

5. กดปุ่ม Add เพื่อสร้าง Master Site
6. เลือก Site ที่ต้องการ พร้อมใส่ Password สำหรับเชื่อมต่อเข้ากับฐานข้อมูลนั้นแบบ Admin



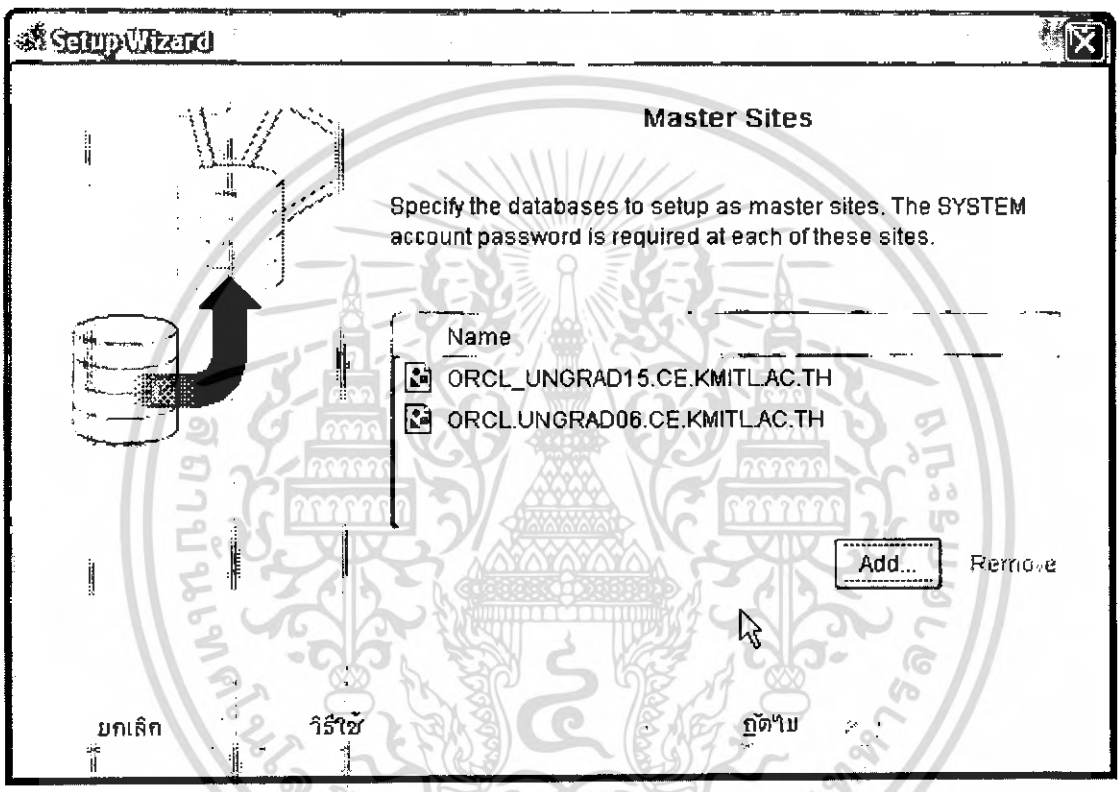
รูปที่ ข.14 แสดงการเลือก Master Site ที่จะเชื่อมต่อกัน

7. เมื่อกำหนด Site เรียบร้อยแล้วขั้นต่อมาจะเป็นการกำหนดผู้จัดการ Replication ซึ่งใช้ User เดียวกับที่สร้างไว้



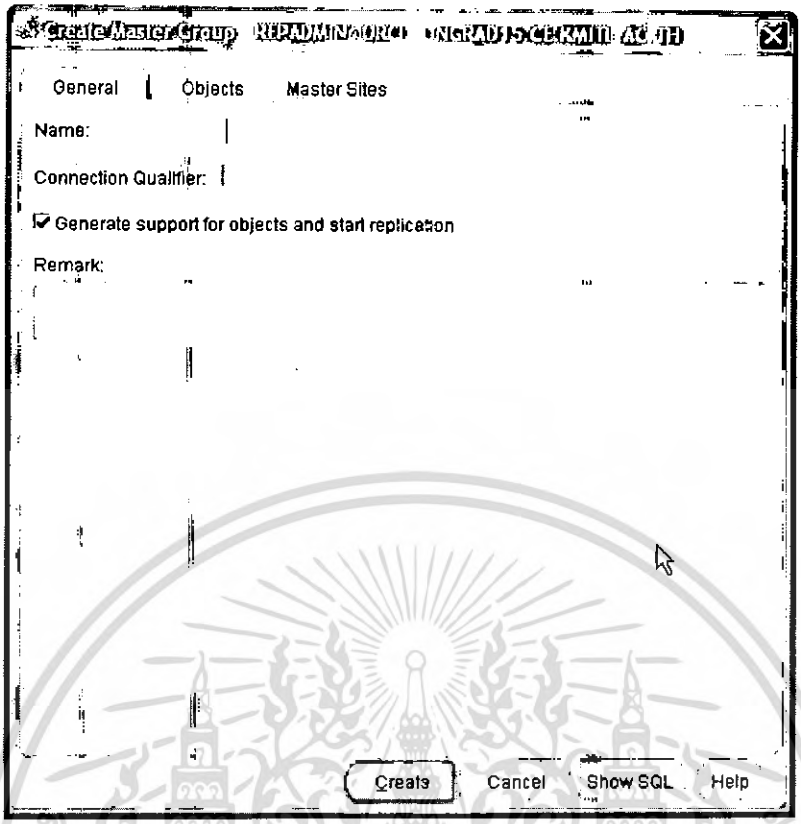
รูปที่ ข.15 แสดงการกำหนด User ที่ดูแลการทำ Replication

8. หน้าต่อมาสำหรับถ้าต้องการสร้าง Schema ใหม่เพื่อทำ Replication ถ้าไม่ต้องการ กดปุ่ม ถัดไป
9. ขั้นต่อมาเป็นการกำหนดเวลาในการเชื่อมต่อ link ซึ่งการทำ Synchronous นั้นใช้ค่าเดิม กดปุ่ม ถัดไป
10. กดปุ่มถัดไปอีกครั้ง จะพบหน้าต่างเพื่อให้สามารถเปลี่ยนแปลงค่าของแต่ละ Master Site ได้



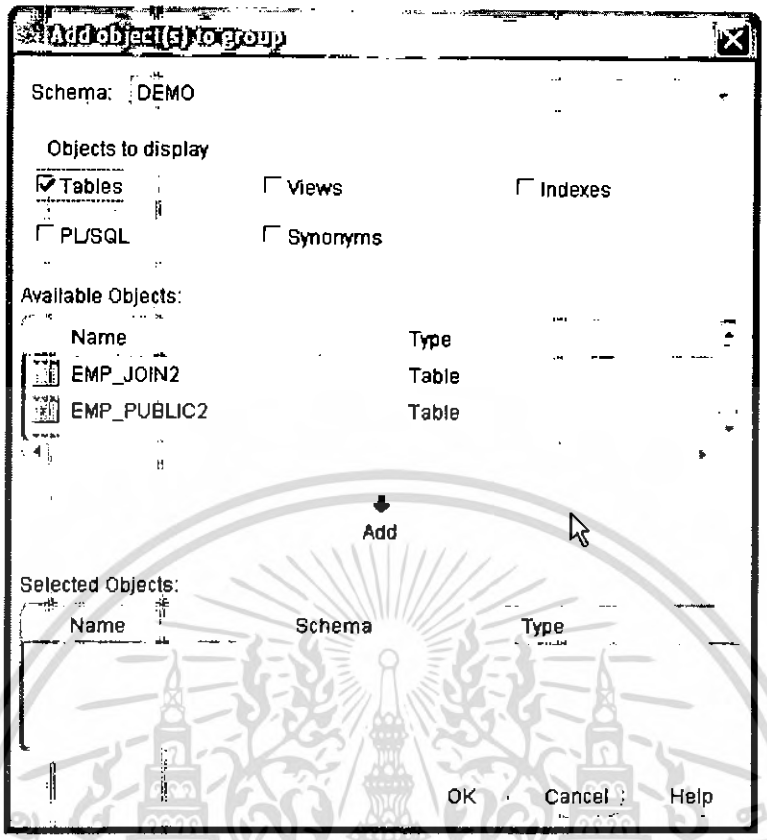
รูปที่ ข.16 แสดงหน้าต่างแสดง Master Site ที่กำหนด

11. เสร็จสิ้นการสร้าง Master Site ต่อไปเป็นการกำหนดสิ่งที่จะทำการ Replicate
12. คลิกขวาที่ Master Group ใน Tree ทางด้านซ้าย เลือก Create



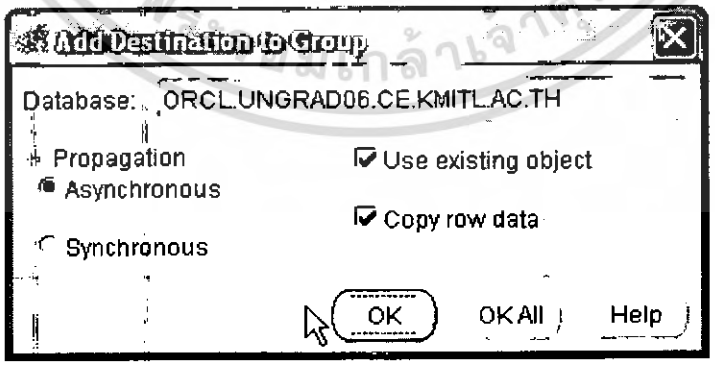
รูปที่ ข.17 แสดงหน้าต่างแรกสำหรับสร้าง Replicate Group

13. จะพบหน้าต่างสำหรับกำหนดชื่อ Group ของการ Replicate กำหนดชื่อเรียบร้อยแล้ว เลือกไป Tab Object



รูปที่ ข.18 แสดงหน้าต่างแรกสำหรับเลือกสิ่งที่ต้องการทำ Replicate

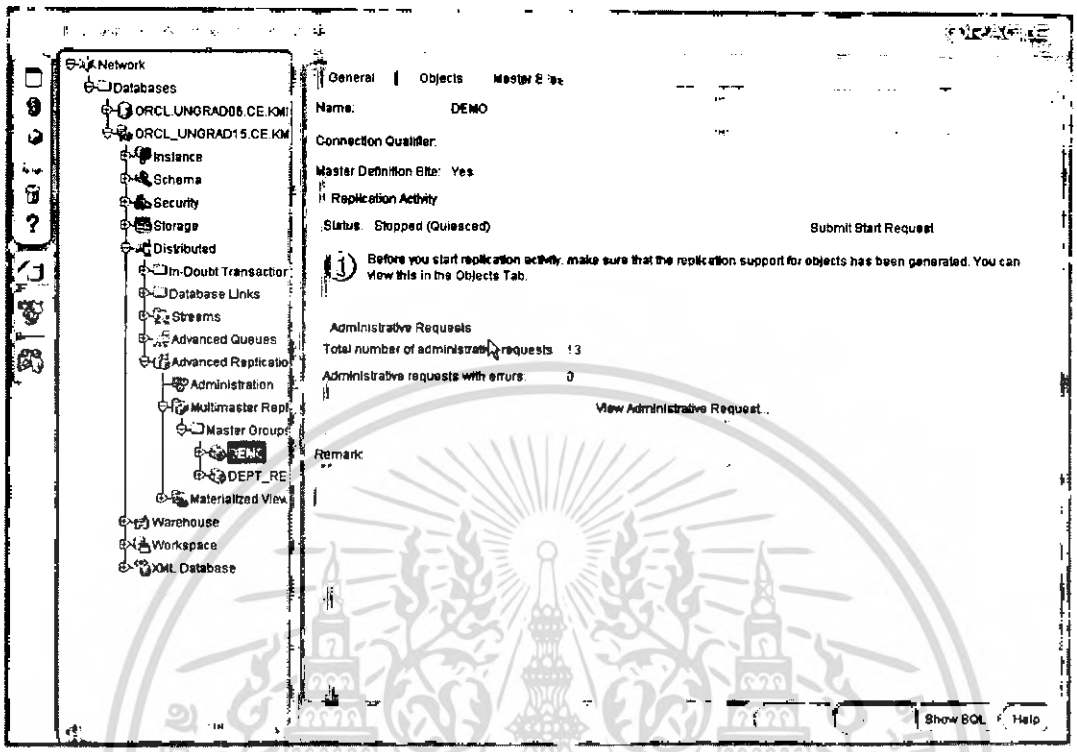
14. ทำการเลือก Schema และสิ่งที่ต้องการทำ Replication
15. ต่อมาเลื่อนไป Tab Master Site เพื่อกำหนด Site ที่ต้องการให้สิ่งที่ Replicate ที่กำลังสร้างอยู่ไปมีข้อมูลชุดนี้อยู่ด้วย



รูปที่ ข.19 แสดงรูปแบบการ Replicate สำหรับ Master Site นั้นๆ

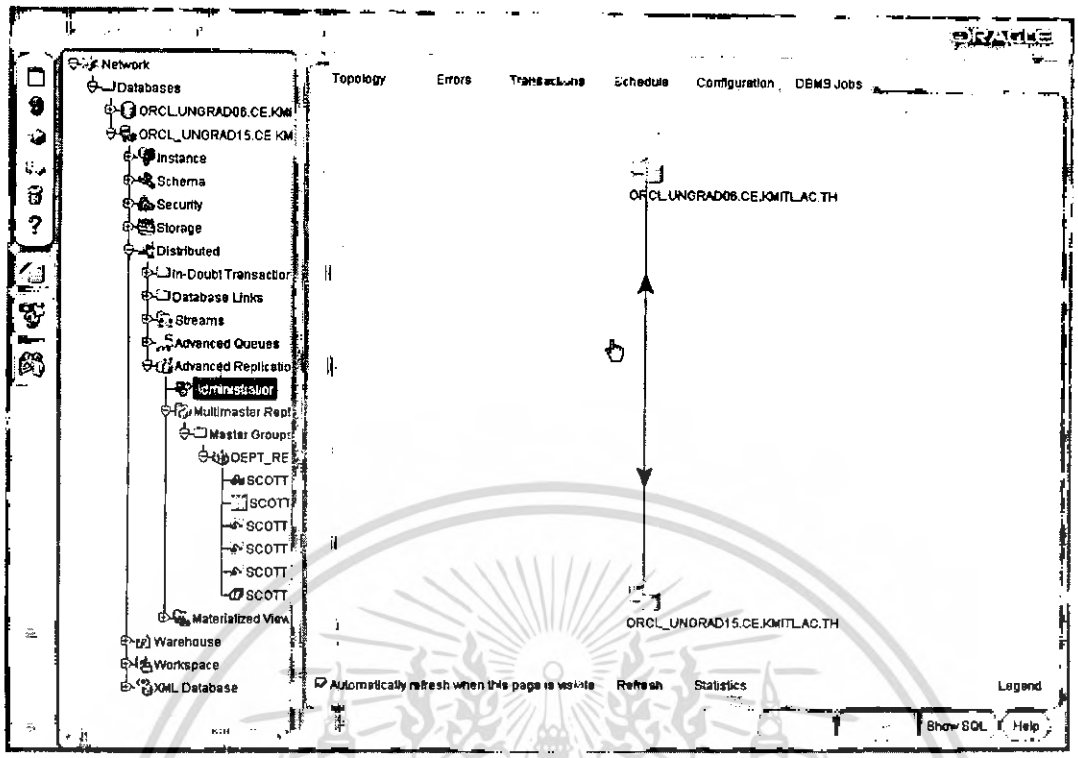
16. เมื่อกด Ok จะมี หน้าต่างให้เลือกว่าต้องการทำ Replicate แบบใด

17. กด Create เป็นการสร้างการ Replication



รูปที่ ข.20 แสดงหน้าต่างควบคุม Replicate Group ที่สร้างขึ้น

18. เมื่อปรากฏ Replication Group ที่สร้างขึ้น ให้เลือก และกด Submit Start Request
19. รอจนค่า Total number of administrative requests : เป็น 0
20. เสร็จสิ้นการทำ Replication



รูปที่ ข.21 แสดงผลลัพธ์ของการทำ Replication