การประมวลผลรายการบนระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

TRANSACTION PROCESSING USING MULTIVERSION ENGIN DBMS

นายจีรพัฒน์ ศุภอภินันต์
นายเอกรัตน์ เดนโพธิ์
นายอภิชัย ธวัชชัยดำรงค์
นายสาเรศ ธรรมเจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2551

การประมวลผลรายการบนระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

TRANSACTION PROCESSING USING MULTIVERSION ENGIN DBMS

โดย

นายจีรพัฒน์	ศุภอภินันต์	49015273
นายเอกรัตน์	เคนโพธิ์	49015320
นายอภิชัย	ธวัชชัยดำรงค์	49015315
นายสาเรศ	ธรรมเจริญ	49015308

อาจารย์ที่ปรึกษา

รศ.ดร.ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญานิพนธ์ปีการศึกษา 2551

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประมวลผลรายการบนระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

TRANSACTION PROCESSING USING MULTIVERSION ENGIN DBMS

ผู้จัดทำ

1. นายจีรพัฒน์ ศุภอภินันต์ 49015273
2. นายเอกรัตน์ เคนโพธิ์ 49015320
3. นายอภิชัย ธวัชชัยดำรงค์ 49015315
4. นายสาเรศ ธรรมเจริญ 49015308

.....อาจารย์ที่ปรึกษา

(รศ.ดร.ศุภมิตร จิตตะยโศธร)

การประมวลผลรายการบนระบบจัดการฐานข้อมูลที่ใช้กลไกล มัลติเวอร์ชัน

นายจีรพัฒน์ ศุภอภินันต์ 49015273
นายเอกรัตน์ เคนโพธิ์ 49015320
นายอภิชัย ธวัชชัยดำรงค์ 49015315
นายสาเรศ ธรรมเจริญ 49015308
รศ.ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2551

บทคัดย่อ

เนื่องจากในปัจจุบันได้มีการใช้งานฐานข้อมูลกันอย่างแพร่หลาย ทั้งในด้านธุรกิจ หรือแม้แต่กระทั้งด้านการศึกษา และอื่นๆ อีกมากมาย ระบบจัดการฐานข้อมูลจึงมีความสำคัญอย่างหนึ่ง ดังนั้นเราจึงจำเป็นที่จะต้องรู้ถึงการทำงานของแต่ละระบบการจัดการฐานข้อมูล

โครงงานนี้จึงได้ศึกษาเพื่อให้มีความรู้และความเข้าใจในระบบจัดการฐานข้อมูลรวมไปถึงโครงสร้างการทำงานที่สำคัญของระบบจัดการฐานข้อมูล ที่ใช้กลไกมัลติเวอร์ชัน (ออราเคิล and โพสเกรสเอสคิวแอล) และรวบรวมเครื่องมือที่ระบบจัดการฐานข้อมูลจัดเตรียมไว้ให้เพื่ออำนวยความสะดวกสำหรับผู้ใช้งาน ซึ่งผู้ใช้งานสามารถที่จะนำเครื่องมือที่ระบบจัดการฐานข้อมูลเหล่านี้มีมาให้ไปใช้ประโยชน์ให้มากที่สุด และเกิดประสิทธิภาพสูงสุด และในโครงงานนี้ยังแสดงถึงความสามารถของระบบจัดการฐานข้อมูลทั้งของ ออราเคิล และ โพสเกรสเอสคิวแอล ด้วย

Transaction processing using Multiversion engine DBMS

Mr. Jeeraput	Supaapinun	49015273
Mr. Eagkarat	Kanpo	49015320
Mr. Sares	Tammajabean	49015308
Mr. Apichai	Thawatchaidumrong	49015315

Assoc.Prof. Dr. Suphamit Chittayasothorn Advisor

Academic Year 2008

Abstract

The use of databases in the present day is largely expanded, not only in business but also in education and many others. So, the understanding of the working process of each database management system is very important.

The objectives of this project are to study database management systems, which cover the main part of the working process structure of the database management systems. We concentrate on database management systems which use the multiversion (Oracle and PostgreSQL) mechanism. We collect the technical knowledge from our experiences the general database management systems for other users so that they can use them as easily as possible. This project shows the competence of both Oracle and ProtgreSQL database management systems.

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญตาราง(ต่อ)

หน้า

สารบัญตาราง(ต่อ)

หน้า

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญของปริญญานิพนธ์

ในปัจจุบันได้มีการนำฐานข้อมูลมาใช้อย่างแพร่หลาย ไม่ว่าจะเป็นในเชิงธุรกิจหรือแม้แต่ในระบบการศึกษา ซึ่งผู้ใช้ก็มักจะคาดหวังให้มีระบบจัดการฐานข้อมูลที่มีประสิทธิภาพสูงๆ และใช้งานได้ง่ายดังนั้นระบบจัดการฐานข้อมูลจึงมีส่วนสำคัญมากขึ้นทุกวันแต่ผู้ใช้งานส่วนใหญ่นั้นยังไม่เข้าใจ ถึงการทำงานของ ระบบจัดการฐานข้อมูล มากพอจึงทำให้ต้องทำงานบางอย่างที่ระบบจัดการฐานข้อมูลมีเครื่องเมื่อช่วยให้ทำงานง่ายขึ้นนั้นด้วยตัวเอง โดยเฉพาะเรื่องความปลอดภัยของฐานข้อมูลและบางครั้งงานนั้นอาจเกิดข้อผิดพลาดขึ้น โดยที่ผู้ใช้นั้นไม่ทราบสาเหตุ ที่เกิดขึ้นอย่างแท้จริงนำมาซึ่งการแก้ปัญหาที่ผิดวิธี และอาจทำให้เกิดความเสียหายกับธุรกิจขึ้น เช่น ความเข้าใจในเรื่อง การควบคุมสถานะพร้อมกัน และเรื่องความปลอดภัยของฐานข้อมูล

โครงงานนี้จึงได้ศึกษาเพื่อให้มีความรู้และความเข้าใจในระบบจัดการฐานข้อมูลรวม ไปถึงโครงสร้างการทำงานที่สำคัญของระบบจัดการฐานข้อมูล ที่ใช้กลไก มัลติเวอร์ชัน(Multiversion) และรวมรวมเครื่องมือที่ ระบบจัดการฐานข้อมูล จัดเตรียมไว้ให้เพื่ออำนวยความสะดวกสำหรับผู้ใช้

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1.2.1 เพื่อศึกษาและทำความเข้าใจถึงการทำงานของระบบฐานข้อมูลของ ระบบจัดการฐานข้อมูลที่ใช้กลไกมัลติเวอร์ชัน รวมถึงโครงสร้างโดยทั่วไปของระบบฐานข้อมูล เช่น โครงสร้างฐานข้อมูลและการทำงานของทรานแซกชัน

1.2.2 เพื่อให้ทราบถึงปัจจัยต่าง ๆ ที่มีผลกระทบต่อความถูกต้องของข้อมูลในระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน เช่น ไอโซเลชันเลเวล (Isolation level)

1.2.3 จัดทำโปรแกรมหรือระบบที่ใช้เครื่องมือที่ ระบบจัดการฐานข้อมูลมีให้มากที่สุดเพื่อทดสอบเครื่องมือต่างๆของระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

1.3 ขอบเขตของปริญญานิพนธ์

1.3.1 เข้าใจโครงสร้างและการทำงานโดยทั่วไปของทรานแซกชัน

1.3.2 เข้าใจขั้นตอนการทำงานของระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน จัดการทรานแซกชันและความถูกต้องของข้อมูล

1.3.3 รวบรวมเครื่องมือและวิธีการใช้เครื่องมือต่างๆที่ระบบจัดการฐานข้อมูลจัดเตรียมไว้ให้เกี่ยวกับความปลอดภัย

1.3.4 ปรับแต่งและตั้งค่าระบบจัดการฐานข้อมูลให้นำเครื่องมือที่มีอยู่มาช่วยในการทำงานของโปรแกรมให้มากที่สุด

1.3.5 พัฒนาโปรแกรมหรือระบบเพื่อทดสอบเครื่องมือต่างๆ ที่ระบบจัดการฐานข้อมูลได้จัดเตรียมไว้ให้

1.4 วิธีการดำเนินงาน

1.4.1 ศึกษาการทำงานของทรานแซกชันในระบบจัดการฐานข้อมูล

1.4.2 ศึกษาการทำงานของระบบจัดการฐานข้อมูลในการจัดการความถูกต้องของข้อมูล และการควบคุมสภาวะพร้อมกัน (Concurrency Control)

1.4.3 ศึกษาเครื่องมือที่ระบบจัดการฐานข้อมูลจัดเตรียมไว้ให้

1.4.4 ทำการทดลองเพื่อทดสอบผลกระทบในการทำทรานแซกชันแบบต่างๆ

1.4.5 ทำการทดลองเกี่ยวกับการใช้และตั้งค่าเครื่องมือต่างๆที่ระบบจัดการฐานข้อมูลจัดเตรียมไว้ให้

1.4.6 ศึกษาการเขียนการพัฒนาแอพพลิเคชันโดยใช้ จาวา

1.4.7 ทำการพัฒนาโปรแกรมที่ใช้ทดสอบเครื่องมือที่ระบบจัดการฐานข้อมูลจัดเตรียม

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ได้รับความรู้ ความเข้าใจเกี่ยวกับโครงสร้างทั่วไปของระบบจัดการฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

1.5.2 ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานของทรานแซคชันและการจัดการกับทรานแซคชันของระบบฐานข้อมูลที่ใช้กลไก มัลติเวอร์ชัน

1.5.3 ได้รับความรู้ความเข้าใจเกี่ยวกับเครื่องมือต่างๆที่ระบบจัดการฐานข้อมูล ออราเคิล (Oracle) และโพสเกรสเอสคิวแอล (ProtgreSQL) ได้จัดเตรียมไว้ให้เพื่อให้ผู้ใช้ทำงานได้ง่ายขึ้น

1.5.4 ได้รับความรู้ความเข้าใจเกี่ยวกับการพัฒนาแอพพลิเคชันโดยใช้ จาวา

บทที่ 2

ทฤษฎีการประมวลผลรายการ (Transaction)

2.1 ความหมายของการประมวลผลรายการ

การประมวลผลรายการ (Transaction) คือ กลุ่มของคำสั่งฐานข้อมูลในระดับลอจิคอล (Logical Unit of Work) ที่ยอมให้ละเมิดกฎความถูกต้องของฐานข้อมูล (Integrity rules) เช่น การโอนเงินระหว่างบัญชีที่ต้องไม่มีเงินหายไปจากระบบ เช่น โอนจาก A ไป B จะต้อง ถอนจาก A ก่อน ค่อยฝาก B ซึ่งระหว่างการทำงานจะมีการละเมิดกฎชั่วคราว แต่เมื่อทำการโอนเสร็จแล้วข้อมูลจะต้องตรงตามกฎเหมือนเดิม ซึ่งการประมวลผลรายการจะต้องตรงตามคุณสมบัติของ ACID

2.1.1 คุณสมบัติ ACID

2.1.1.1 อะตอมมิกซิตี

อะตอมมิกซิตี (Atomicity) คือการกระทำจะทำงานแยกจากกันไม่ได้ถ้าเกิดการทำงานไม่สำเร็จก็แสดงว่างานทุกอย่างจะถูกยกเลิกหมด แต่ถ้างานสำเร็จงานก็จะสำเร็จหมด

2.1.1.2 คอนซิสแทนซี

คอนซิสแทนซี (Consistency) คือ ฐานข้อมูลจะต้องอยู่ในสภาพที่ถูกต้องอยู่เสมอ ตราบใดที่โปรแกรมงานของผู้ใช้ไม่มีข้อผิดพลาดในทางธุรกิจการประมวลผลของ ทรานแซคชัน จะต้องไม่ทำให้ข้อมูลในฐานข้อมูลผิดพลาด

2.1.1.3 ไอโซเลชัน

ไอโซเลชัน (Isolation) คือ ไม่ว่าในระบบจะมี ทรานแซคชัน ทำงานอยู่จำนวนเท่าไรก็ตามการทำงานของ ทรานแซคชัน หนึ่งจะต้องทำงานร่วมกับ ทรานแซคชัน อื่นเสมือนว่าแต่ละ ทรานแซคชัน นั้นทำงานเป็นอิสระซึ่งกันและกัน

2.1.1.4 ดูราบิลิตี้

ดูราบิลิตี้ (Durability) คือ เมื่อมี ทรานแซคชัน หนึ่งทำงานเสร็จสิ้นลงแล้วผลของการประมวลของ ทรานแซคชัน จะต้องไม่สูญหาย ถึงแม้ว่าจะมีเหตุขัดข้องหลังจาก ทรานแซคชัน เสร็จสิ้นแล้วก็ตาม

2.2 สถานะของ ทรานแซคชัน

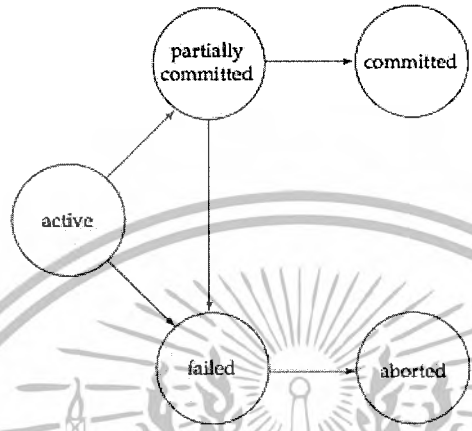
2.2.1 Active เป็นสถานะเริ่มต้นของ ทรานแซคชันหรือขณะที่ทรานแซคชันกำลังทำงาน

2.2.2 Partially คือการ คอมมิต (COMMIT) เมื่อคำสั่งสุดท้ายทำเสร็จแล้ว

2.2.3 Failed เป็นสถานะที่ ทรานแซคชันไม่สามารถทำงานได้เพราะปัญหาต่างๆ

2.2.4 Aborted เป็นสถานะที่ทรานแซคชันถูกยกเลิกและจะถูกโรลแบ็ค(ROLLBACK) กลับไปที่เก่า

2.2.5 Committed เป็นสถานะที่ยืนยันการเปลี่ยนแปลง



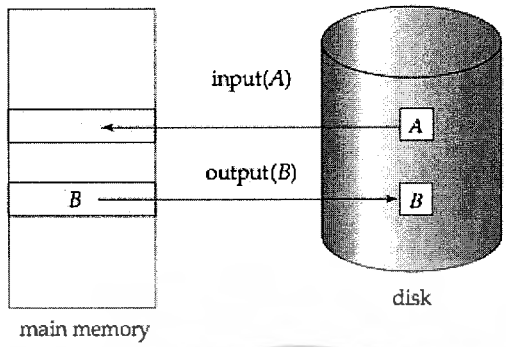
รูปที่ 2.1 ขั้นตอนที่ไดอะแกรมของทรานแซคชัน

ถ้าทรานแซคชันใดๆถูก คอมมิต แล้วจะไม่สามารถถูก โรลแบ็ค กลับไปยังสถานะก่อนหน้าได้เลย และทรานแซคชันจะอยู่ในสถานะ Failed เมื่อทรานแซคชันนั้นไม่สามารถดำเนินการต่อไปได้เนื่องจากปัญหาใดๆ หลังจากทรานแซคชันถูก โรลแบ็ค แล้วจะอยู่ในสถานะ Aborted ซึ่งสามารถดำเนินการต่อได้ 2 วิธีคือ

- เริ่มทรานแซคชันใหม่ (Restart Transaction) ทำเมื่อเกิดปัญหาทาง อาร์ดแวร์ หรือ ซอฟแวร์ ซึ่งไม่ใช่ปัญหาที่เกี่ยวกับลอจิก และเสมือนเป็นการสร้างทรานแซคชันใหม่ขึ้นมา
- ทำลายทรานแซคชัน (Kill Transaction) ทำเมื่อเกิดความผิดพลาดทางลอจิก หรืออาจเป็นเพราะ อินพุตที่ผิดก็ได้

2.3 การกู้คืนทรานแซคชันที่เกิดข้อผิดพลาดระหว่างทำงาน

การกู้คืนทรานแซคชันที่เกิดข้อผิดพลาดระหว่างทำงาน (Transaction Recovery Problem ขั้นตอนที่ Statements)การทำงานของ ดีบีเอ็มเอส (DBMS) นั้นไม่ได้ทำงานกับดิกส์จริงๆแต่จะทำงานกับข้อมูลบนบัฟเฟอร์ที่อยู่ในหน่วยความจำหลัก ซึ่ง ดีบีเอ็มเอส จะดูก่อนว่ามีข้อมูลที่ต้องการในบัฟเฟอร์หรือไม่ ถ้าไม่มีก็จะไปโหลดมาจาก ดีบี สเปรซ มาเก็บไว้บนบัฟเฟอร์ซึ่งทำให้เกิดปัญหา 2 ข้อดังนี้



รูปที่ 2.2 การทำงานระหว่างบัฟเฟอร์กับ ดีบี สเปรซ

2.3.1 มีทรานแซคชันที่ คอมมิต

แล้วแต่การเปลี่ยนแปลงยังไม่ถูกเอาท์พุทจากบัฟเฟอร์ ลง ดีบี สเปรซ ถ้าเกิดความผิดพลาดขึ้นจะทำอย่างไรจึงจะกู้ข้อมูลการเปลี่ยนแปลงที่ คอมมิต แล้วนั้นให้กลับมาอยู่อย่างถาวรบน ดีบี สเปรซ(DB pace) ได้

2.3.2 ที่ทรานแซคชันที่ยังไม่ คอมมิต

แต่การเปลี่ยนแปลงถูกเอาท์พุทลง ดีบี สเปรซ ไปแล้วถ้าเกิดข้อผิดพลาดขึ้นจะทำอย่างไรจึงจะยกเลิกการเปลี่ยนแปลงเหล่านั้นออกจาก ดีบี สเปรซ

2.4 ล็อก – เบส รีโคเวอร์รี

ล็อก – เบส รีโคเวอร์รี (Log - Based Recovery) คือขั้นตอนการการกู้คืนข้อมูลจากไฟล์ล็อกที่เก็บไว้ เมื่อเกิดความผิดพลาดระหว่างการทำงาน

2.4.1 เป็นใช้ล็อกไฟล์

ใช้มาช่วยในการกู้คืนข้อมูลซึ่งเมื่อเริ่มทรานแซคชันจะมีการเขียนข้อมูลลงล็อกไฟล์ ดังนี้

- เมื่อเริ่มต้นทรานแซคชันจะเขียน <Ti,Start>ลงใน ล็อกไฟล์
- ขณะทำทรานแซคชันจะเขียน < Ti , Xi,OV,NV> ลงใน ล็อกไฟล์
- เมื่อทำทรานแซคชันสำเร็จจะเขียน <Ti,COMMIT> ลงใน ล็อกไฟล์
- Ti= Transaction ID, Xi=Row ID, OV= Old Values, NV=New Value

2.4.2 หลังจากเกิดข้อผิดพลาด

หลังจากเกิดข้อผิดพลาด เช่นไฟดับ ก็ให้ รีสตาร์ทระบบปฏิบัติการ และรีสตาร์ทดีบีเอ็มเอส จะมีขั้นตอนการทำงานดังนี้

- ดีบีเอ็มเอส จะสแกน ล็อกไฟล์ ย้อนจากท้ายเพื่อตรวจสอบว่า ทรานแซคชันไหน คอมมิต หรือ ทรานแซกชัน ไหนผิดพลาด (ไม่มี คอมมิต)
- สำหรับแต่ละทรานแซคชันที่ ผิดพลาด ดีบีเอ็มเอส จะนำ Ti ไปเก็บไว้ที่ Undo list Undo list และทรานแซคชันที่ คอมมิต แล้ว ดีบีเอ็มเอส จะนำ Ti ไปเก็บไว้ที่ Redo list
- สำหรับแต่ละทรานแซคชันใน Undo list ดีบีเอ็มเอส จะนำ Old Values ไป Undo ดีบี สเปรซ เป็นการแก้ปัญหา Output ลง ดีบี สเปรซ โดยที่ยังไม่ได้ คอมมิต (ปัญหาข้อ 2.3.2)และ สำหรับแต่ละทรานแซคชันใน Redo list ดีบีเอ็มเอส จะนำ New Values ไป Redo ดีบี สเปรซ เป็นการแก้ปัญหา คอมมิตแล้วแต่ยัง Output ลง ดีบี สเปรซ โดยที่ยังไม่ได้ คอมมิต (ปัญหาข้อ 2.3.1) Log Base Recovery จะต้องปฏิบัติตาม Write ahead protocol คือข้อมูลที่อยู่ใน Log Buffer จะต้องถูก เอาท์พุตไปยัง ล็อกไฟล์ ก่อนที่ข้อมูลที่เกี่ยวข้องกับทรานแซคชันใน DB Buffer จะถูกเอาท์พุตไปลง ดีบี สเปรซ

2.5 การควบคุมภาวะพร้อมกัน

2.5.1 ความหมายของภาวะพร้อมกัน

คำว่า "ภาวะพร้อมกัน (Concurrency)" หมายความว่า การที่มีทรานแซกชันหลายๆ ทรานแซกชันต้องการเรียกใช้ข้อมูลเดียวกันในเวลาเดียวกันจากฐานข้อมูลเพื่อทำงานของแต่ละทรานแซกชันภาวะการทำงานพร้อมกันเกิดจากระบบการทำงานได้ 2 ระบบ คือ การทำงานในระบบหลายโปรแกรม และ การทำงานในระบบการประมวลผลในเวลาเดียวกัน

2.5.1.1 การทำงานในระบบหลายโปรแกรม

การทำงานแบบมัลติโปรแกรมมิ่ง (multiprogramming) เป็นการทำงานของระบบคอมพิวเตอร์ที่ออกแบบเพื่อให้หน่วยประมวลผลผลกลางหรือซีพียู (Central Processing Unit; CPU) ทำงานหลายๆ งานในขณะเดียวกัน ได้ทั้งนี้ด้วยเหตุผลเพื่อให้การใช้งานซีพียูเป็นไปอย่างคุ้มค่าโดยไม่ต้องอยู่ว่าง (idle) เนื่องจากซีพียูจะมีความเร็ว ในการทำงานสูงกว่าอุปกรณ์อื่นๆถ้าหากไม่มีระบบการทำงานแบบมัลติโปรแกรมมิ่งซีพียูต้องทำงานใดงานหนึ่งจนเสร็จจึงจะสามารถทำงานที่ 2, 3 ต่อไปได้ ซึ่งหมายความขณะที่งานนั้นกำลัง ใช้เครื่องพิมพ์หรืออุปกรณ์อื่นๆ ที่ไม่ใช่ซีพียู ซีพียูก็ต้องเสียเวลารอดังนั้นจึงเกิดแนวคิดการประมวลผลแบบ ให้หลายโปรแกรมทำงานพร้อมๆ กันโดยมี

การสลับช่วงการทำงานระหว่าง โปรแกรมเพื่อสลับให้ซีพียูไปทำงานของทรานแซกชันอื่นๆ ซึ่งแต่ละทรานแซกชันที่ต้องการให้ซีพียูทำงานนั้นอาจจะเป็นโปรแกรมเดียวกันหรือต่างโปรแกรมก็ได้ โดยใช้หลักการอินเทอลีฟ มาใช้ในการควบคุมภาวะพร้อมกัน อินเทอลีฟ (interleaved) คือ การที่ทรานแซกชันมากกว่าหนึ่ง

ทรานแซกชัน มีการสลับการทำงานกันในขณะใดขณะหนึ่งโดยที่ระบบจัดการฐานข้อมูลจะต้องควบคุมภาวะพร้อมกัน (Concurrency control) เพื่อให้แต่ละทรานแซกชันมีการทำงานสลับกันไปมา ทั้งนี้ผลลัพธ์ที่ได้จะต้องมีความถูกต้องเสมือนว่าแต่ละทรานแซกชันทำงานเรียงลำดับที่ละทรานแซกชันจนสิ้นสุดงานของแต่ละทรานแซกชันนั้น

2.5.1.2 การประมวลผลในเวลาเดียวกัน

เป็นการทำงานในระบบคอมพิวเตอร์ที่มีซีพียูมากกว่า 1 ซีพียู เพื่อรองรับการทำงานของโปรแกรมใดโปรแกรมหนึ่งได้โดยไม่ต้องสลับทำงานระหว่างหลายทรานแซกชัน ดังนั้นซีพียูแต่ละตัวก็ จะทำงานของโปรแกรมใดโปรแกรมหนึ่งแยกกันไปแต่ละซีพียูจนเสร็จงาน

2.5.2 ปัญหาการควบคุมภาวะพร้อมกัน

การควบคุมภาวะพร้อมกัน(Concurrency Control Problem) ในการใช้งานฐานข้อมูลเป็นสิ่งสำคัญอย่างยิ่ง เพราะหากระบบจัดการฐานข้อมูลไม่มีกลไกดังกล่าวย่อมจะก่อให้เกิดปัญหาในการทำงานดังนี้

2.5.2.1 ปัญหาการสูญหายของข้อมูล (The Lost Update Problem)

เป็นปัญหาที่เกิดจากทรานแซกชันมากกว่าหนึ่งทรานแซกชันต้องการแก้ไขข้อมูลในช่วงเวลาเดียวกันทำให้ผลลัพธ์ที่ได้ไม่ถูกต้อง

ตารางที่ 2.1 ปัญหา Lost Update

ทรานแซคชัน 1	ทรานแซคชัน 2	A=10	ผลลัพธ์ที่ต้องการ
Read (A) A=A+5		10+5=15	ทรานแซคชัน 1 , 10+5=15
	Read (A) A=A-5	10-5=5	ทรานแซคชัน 1 , A=15

ตารางที่ 2.1 ปัญหา Lost Update (ต่อ)

ทรานแซคชัน 1	ทรานแซคชัน 2	A=10	ผลลัพธ์ที่ต้องการ
Write A		15	ทรานแซคชัน 2 ,15-5 =10
	Write A	5	ทรานแซคชัน 2 ,A=10

2.5.2.2 ปัญหาจากการเรียกใช้ข้อมูลชุดเดียวกันของทรานแซกชันที่ไม่คอมมิต

ปัญหาจากการเรียกใช้ข้อมูลชุดเดียวกันของทรานแซกชันที่ยังไม่คอมมิต (Uncommitted Dependency Problem) เป็นปัญหาที่เกิดจากทรานแซกชันมากกว่าหนึ่งทรานแซกชันต้องการเรียกใช้ข้อมูลชุดเดียวกันโดยมีทรานแซกชันหนึ่งยังอยู่ระหว่างการทำงานขณะเดียวกันมี ทรานแซกชันอื่นเรียกใช้ข้อมูลเดียวกัน ซึ่งเป็นข้อมูลที่ยังไม่มีการ คอมมิต หรือ โรลแบ็ก ทำให้ข้อมูลที่ทรานแซกชั้นนั้นเรียกใช้ยังไม่นิ่งเพราะข้อมูลนั้นยังสามารถเปลี่ยนแปลงได้อยู่

ตารางที่ 2.2 แสดงปัญหา Uncommitted Dependency

ทรานแซคชัน A	ทรานแซคชัน B
A=A+10 Write (A)	
	Read (A) A=A-5
ROLLBACK	
	Write (A)???????

2.5.2.3 ปัญหาการเรียกใช้ข้อมูลที่ไม่สอดคล้องกัน

ปัญหาการเรียกใช้ข้อมูลที่ไม่สอดคล้องกัน (Inconsostent Analysis Problem) เป็นปัญหาที่เกิดจากทรานแซกชันมากกว่าหนึ่งทรานแซกชันมีการใช้งานชุดข้อมูลเดียวกันโดยทรานแซกชันหนึ่งใช้ข้อมูลนั้นเพื่อประมวลผลใดๆ ในขณะเดียวกันก็มีทรานแซกชันอื่นแก้ไขข้อมูลชุดเดียวกันทำให้ผลลัพธ์ของทรานแซกชันแรกไม่ถูกต้อง

ตารางที่ 2.3 แสดงปัญหา Inconsistent Analysis

ทรานแซคชัน A	ทรานแซคชัน B
Read R acc1(40) ; บวก = 40	
Read R acc2(50) ; บวก = 90	
	update R acc3(20) ; 30-10
	update R acc1(50) ; 40+10
	COMMIT;
Read R acc3(20) ; บวก = 110	

2.5.2.4 Phantom Phenomenon คล้ายกับข้อ 2.5.2.3 แต่ เปลี่ยนจาก UPDATE เป็น INSERT

2.5.3 ไอโซเลชัน เลเวล

ไอโซเลชัน เลเวล (Isolation level) คือระดับในการความคุมความถูกต้องของข้อมูลซึ่ง ดีบีเอ็มเอส จะเป็นตัวจัดการให้ซึ่งแต่ละระดับก็จะมีความเข้มงวดเรื่องความถูกต้องต่างกันและแก้ปัญหา สภาวะพร้อมกัน (Concurrency) ได้บางข้อในแต่ละระดับ

ตารางที่ 2.4 แสดง ไอโซเลชัน เลเวลที่แก้ปัญหาต่างๆได้

ไอโซเลชัน เลเวล	Dirty read	Uncommit Dependency	Inconsistent Analisis	Phantom Phenomenon
Serializable	√	√	√	√
Repeaตาราง read	√	√	√	
Read committed	√	√		
Read Uncommitted				

2.6 ล็อก–เบส โปรโตคอล

ล็อก–เบส โปรโตคอล(Lock – Based Protocols) เป็นกลไกการทำงานภายในหลักๆ ของดีบีเอ็มเอส ซึ่งมี หลักการที่สำคัญ หลักๆ 2 ข้อ คือlock – based protocol (lock – base Technique) Time stamp – base protocol

2.6.1 ล๊อก – เบส เทคนิค

ล๊อก – เบส เทคนิค (Lock – base Technique) มีปัจจัยที่เกี่ยวข้อง 3 ประการ ได้แก่

2.6.1.1 คำสั่งที่ใช้ในการ ล็อค (Lock primitive)

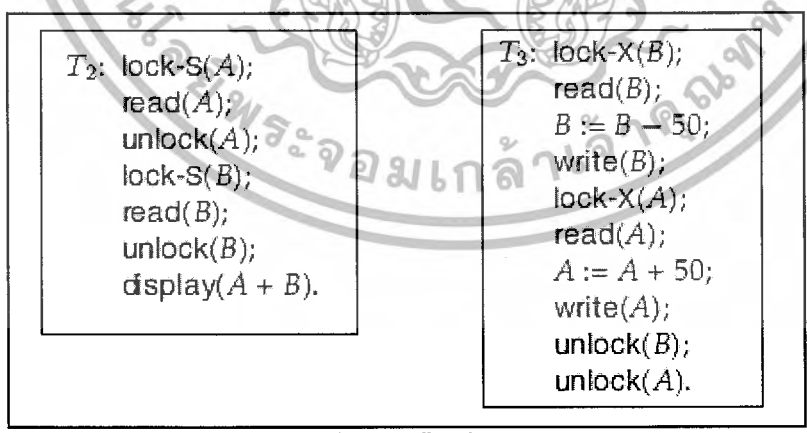
แชร์ล็อก(Shared lock) มีความสามารถ ในการอ่านได้อย่างเดียว (เอสล็อก) และเอ็กคลูซีพ ล็อก (Exclusive lock) มีความสามารถในการอ่านหรือเขียนก็ได้ (เอ็กล็อก)

2.6.1.2 ตารางเปรียบเทียบการทำงานร่วมกันได้ (Compatibility matrix)

ตารางที่ 2.5 สภาวะการทำงานของ SLOCKและ XLOCK

	S	X
S	true	false
X	false	false

- ถ้าแชร์อยู่ คนอื่นขอแชร์ได้ แต่ขอเอ็กล็อกไม่ได้
- ความผิดพลาด คนที่ มาทีหลังต้องรอจนกว่าเราจะปลดล็อค
- ปลด ล็อค กันที่ จุดสิ้นสุดการทำงาน ยิ่งคอมมิตเร็วยิ่งปลด ล็อค เร็ว



รูปที่ 2.3 แสดงตัวอย่างปัญหาที่ยังเกิดขึ้นเมื่อมีการใช้ ล๊อก- เบส เข้ามาช่วย

ถ้า T2 และ T3 ทำพร้อมกันสิ่งที่เกิดขึ้นคือคำตอบผิดค่าที่ T2 อ่านเป็น A เก่า แต่ค่า B เป็นค่าใหม่ที่ T3 ได้ write มาก่อนแล้ว

2.6.1.3 ทู-เฟส ล็อกกิ๊ง โปรโตคอล (The Two-Phase Locking Protocol)

เป็นโปรโตคอลที่นิยมใช้มากที่สุดมี หลักการทำงานดังนี้

- Growing phase: ล็อคได้อย่างเดียวไม่สามารถปลดล็อคได้
- Shrinking phase: ปลดล็อคแล้วไม่สามารถล็อคได้อีก

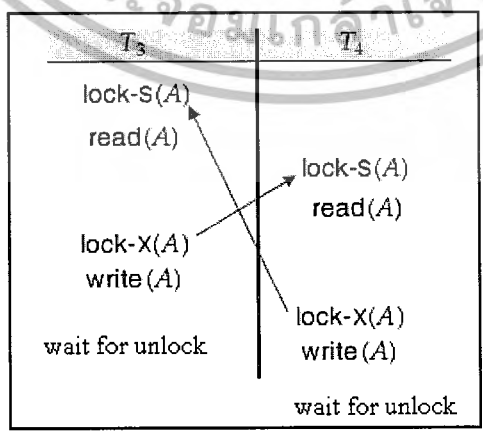
Lock A
Lock B
Lock C
} เฟส 1 (ห้ามปลดล็อค)

Unlock A
Unlock B
Unlock C
} เฟส 2 (ปลดล็อค แล้วจะล็อคไม่ได้อีก)

ซึ่ง ทู-เฟส ล็อกกิ๊ง โปรโตคอล ยังไม่ได้แก้ปัญหาข้อที่ 2.5.2.2 วิธีที่ใช้แก้ ปัญหาด้วยการเพิ่ม Solution ต่อไปนี้เข้าไป

สติ๊ก ทู-เฟส ล็อกกิ๊ง โปรโตคอล (Strict two-phase locking protocol) คือการนำ ทู-เฟส ล็อกกิ๊ง โปรโตคอล บวกด้วย การที่ เอ็กล็อกจะปลดล็อค ที่จุดสิ้นสุดทรานแซคชันเท่านั้น

รีโกรออัล ทู-เฟส ล็อกกิ๊ง โปรโตคอล (Rigorous two-phase locking protocol) คือการนำทู-เฟส ล็อกกิ๊ง โปรโตคอล บวกด้วย การที่ เอ็กล็อกหรือ เอสล็อกสำหรับ เอ็กล็อกจะปลดล็อคได้เฉพาะในจุด สิ้นสุดทรานแซกชันเท่านั้นและ เอสล็อกปลดล็อคเมื่อไรก็ได้แชร์ริ่ง ทั้ง สติ๊ก ทู-เฟส ล็อกกิ๊ง โปรโตคอล และรีโกรออัล ทู-เฟส ล็อกกิ๊ง โปรโตคอล รับประกันคอทฟิกชีเรียลไลซาเบิล (conflict serializable schedule) และคอสเคดเลส สเก็ตดวล (casscadeless schedule) และวิธีนี้เป็นการแก้ไขปัญหาข้อที่ 2.5.2.2 แต่ สร้างปัญหาในข้อที่ 2.5.2.1 คือ ติด เด็ดล็อก (DEADLOCK)



รูปที่ 2.4 แสดงให้เห็นปัญหา DEADLOCK

จากรูปที่ 2.4 จะเห็นได้ว่า ที่ T3 จะขอ เอ็กล็อกไม่ได้ เนื่องจากติด เอสล็อกของ T4 และ T4 จะขอ เอ็กล็อกไม่ได้ เนื่องจากติด เอสล็อก ของ T3 และจะรอฝ่ายตรงข้ามที่จะปลดล็อค ทั้งคู่วิธีแก้ไข คือ ให้ใช้ เอ็กล็อกตั้งแต่ต้น โดยไม่ใช้ เอสล็อกก็จะสามารถแก้ไขปัญหาข้อที่ 2.5.2.1

2.6.2 การล็อคหลายระดับ

การล็อคหลายระดับ (Multiple Granularity)เป็นการล็อค ทั้งตารางเพื่อช่วยในการแก้ปัญหา 2.5.2.3 และ 2.5.2.4 โดย ดีบีเอ็มเอส จะสามารถปรับ ขนาดของข้อมูลที่จะถูกนำมาล็อคโดย อัตโนมัติ นอกจากนี้ วิธีนี้ยังเป็นการช่วยลด over head ในขณะ off peak ด้วย (เป็นการปรับล็อค ในขณะที่ไม่มีคนแชร์) ลักษณะการทำงาน เมื่อต้องการ ล็อคหลายระดับซึ่งมี ขนาดใหญ่ ดีบีเอ็มเอส จะทราบได้อย่างไรว่า ไม่มี Granularity ขนาดเล็กกว่า ไม่มี ทรานแซคชัน อื่นใช้งานอยู่ วิธีการคือ ใช้ Intend lock แผนภูมิต้นไม้ของการล็อคหลายระดับ เข้ามาช่วย ซึ่งมีขั้นตอนการทำงานมีดังนี้

2.6.2.1 อินเท็นชัน แชร์โหมด

อินเท็นชัน แชร์โหมด (IS: intention-shared mode) ถ้าโหนดใดถูก ไอเอสล็อกมี sub tree ใต้โหนดนั้น ถูกเอสล็อก

2.6.2.2 อินเท็นชัน เอ็กคลูซีฟโหมด

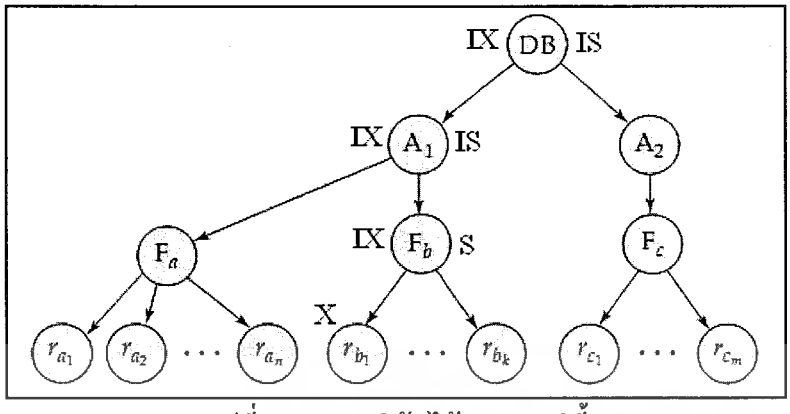
อินเท็นชัน เอ็กคลูซีฟโหมด (IX: intention-exclusive mode) ถ้าโหนดใดถูก I เอ็กล็อกจะมี sub tree ใต้โหนดนั้นถูกเอสล็อกหรือ เอ็กล็อกก็ได้

2.6.2.3 แชร์แอนอินเท็นชันเอ้นคลูซีฟโหมด

แชร์แอนอินเท็นชันเอ้นคลูซีฟโหมด (SIX: shared and intention-exclusive mode) ถ้าโหนดใดถูก เอสไอเอ็กล็อกโหนดนั้นถูก เอสล็อกแน่นอน และใต้ซับทรีนั้นอาจถูก เอ็กล็อก

ตารางที่ 2.6 สภาวะการทำงานของ มัลติเพิ้ล แกรนลาริตี้ (Multiple Granularity)

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false



รูปที่ 2.5 แผนภูมิต้นไม้ แกรนลาริตี้

จากรูปที่ 2.5 ถ้าเราต้องการ อัพเดท ที่ r_{b1} จะต้อง ไอเอ็กล็อกตั้งแต่รูท และทางผ่านทั้งหมด จนมาถึงโรลที่ต้องการจึงใส่ เอ็กล็อกเมื่อเราต้องการบวกทั้งตาราง F_b ก็ต้อง ไอเอสล็อกตั้งแต่ รูทมาเช่นกัน จนถึง F_b

ขั้นตอนที่ 1 : ที่รูทมีการ ไอเอ็กล็อกอยู่ก่อนแล้ว มีการ ไอเอสล็อกตามมา ตรวสอบที่ตาราง สภาวะการ ทำงานของ มัลติเพิ้ล แกรนลาริตี้ ได้ผลว่าไอเอ็กล็อกกับ ไอเอสล็อกมีค่าเป็นทรู

ขั้นตอนที่ 2 : ที่ A_1 Iเอ็กล็อกกับ ไอเอสล็อกมีค่าเป็น ทรู

ขั้นตอนที่ 3 : ที่ F_b Iเอ็กล็อกกับ เอสล็อกมีค่าเป็น ค่าเป็น ฟอล

ดังนั้น การ บวก ทั้ง ตาราง F_b ยังไม่สามารถทำได้

2.7 ไทม์แสตมป์-เบส โปรโตคอล

ไทม์แสตมป์-เบสโปรโตคอล (Time Stamp-Base Protocol) คือวิธีการควบคุมภาวะความพร้อมกัน โดย ระบบจัดการฐานข้อมูลจะระบุลำดับของแต่ละทรานแซกชันในการเข้าทำงาน ดังนั้น การควบคุมภาวะพร้อมกันโดย จะรับประกัน คอลฟิก ซีเรียลไลซาเบิล และเป็นการแก้ไขปัญหาภาวะการทำงานพร้อมกัน ทั้ง 4 ข้อได้โดยที่ไม่ต้องมีการใช้ ล็อก เบส เทคนิค เข้ามาช่วยเลย จึงไม่ต้องกังวลว่าจะเกิดปัญหา เด็ดล็อก ขึ้นซึ่งจะมีการใช้เวลามาช่วยแทน มีวิธีการดังนี้

Ts(Ti) เป็นเวลาที่ ทรานแซกชัน Ti เกิด อาจเป็นเวลาจิง หรือเวลาที่โปรแกรมสร้างขึ้นเองก็ได้ แต่มักใช้ เวลาที่โปรแกรมสร้างขึ้นเองก็ได้ มากกว่าเพราะไม่ยึดติดกับ ระบบปฏิบัติการ ถ้า TS (Ti) เกิดก่อน TS (Tj) แล้ว: TS (Ti) < TS (Tj): (TS (Tj) มีความเป็นเด็กกว่า)

การบันทึกไทม์แสตมป์ (Write timestamp (WTS)) คือ ไทม์แสตป์ ทรานแซกชัน ใด ที่มีอายุน้อยที่สุดแล้ว ทืนทึก Q สำเร็จ (เด็กที่สุดที่ยันทึกสำร็จ)

อ่านค่าเวลาไทม์แสตมป์ (RTS) คือ ไทม์แสตป์ ของ ทรานแซกชัน ใด ที่มีอายุน้อยที่สุดแล้ว อ่าน Q สำเร็จ (เด็กที่สุดที่อ่านสำร็จ)

2.7.1 กฎและขั้นตอนของการทำไทม์แสตป์

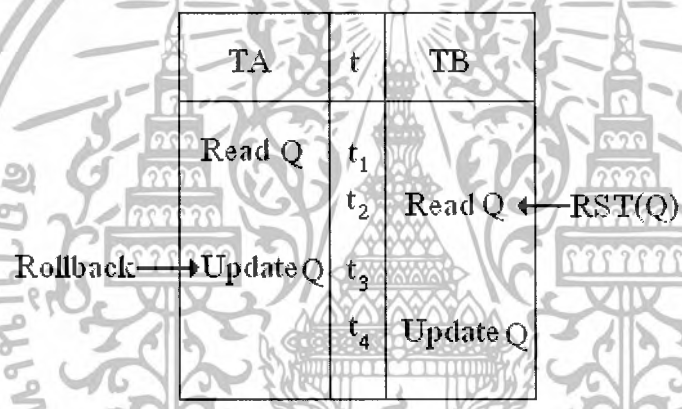
กฎและขั้นตอนของการทำไทม์แสตป์(Time Stamp-Base Qrdering Protocol)มีดังนี้

2.7.1.1 ดูว่าอ่านได้หรือไม่ได้

TS ที่ (Ti) มีค่ามากกว่าเราแล้วบันทึกไปแล้วเราจะไม่สามารถอ่านได้ จะ โรลแบ็ค ถ้ามีใครที่ TS (Ti) มีค่ามากกว่าเราแล้ว อ่านไปแล้ว เรา อ่านได้ หรือตัวเราเอง เป็นคน บันทึกไว้ เรา ก็สามารถอ่าน ได้

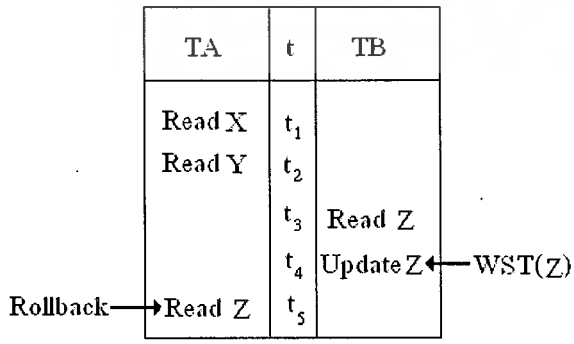
2.7.1.2 ดูว่า บันทึกได้หรือไม่ได้

ถ้ามีใครที่ TS (Ti) มีค่าน้อยกว่า RST(Q) แสดงว่ามีคนที่มาทีหลัง อ่านค่าเก่าเรา ไปแล้ว เราจะไม่สามารถ บันทึกได้ จะถูก โรลแบ็ล เมื่อเราต้องการบันทึกแต่มีคนอ่านตัดหน้าเรา ไปแล้วเราจะไม่สามารถ บันทึกได้ จะ โรลแบ็ก



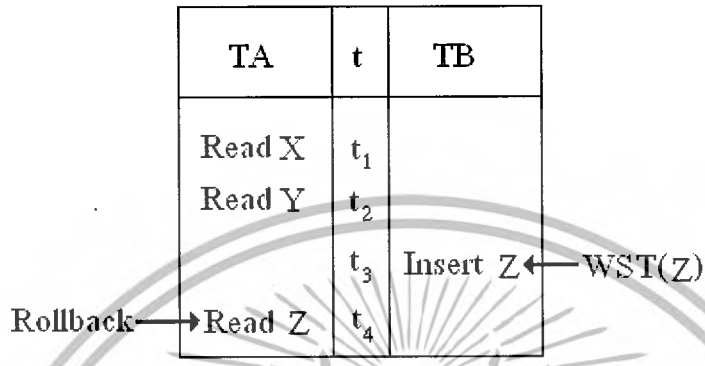
รูปที่ 2.6 แสดงการแก้ไขปัญหาข้อที่ 2.5.2.1

จากรูปที่ 2.6 เมื่อ TA เกิดก่อน TB ดังนั้น TB จึงมีอายุน้อยกว่า TA เริ่มต้นด้วยการ ที่ TA ทำการ อ่าน Q ที่ เวลาที่ t_1 เวลาที่ t_2 TB ได้มา อ่านQ ตัวเดียวกับ แต่เป็น ทรานแซคชัน ที่มี อายุน้อยกว่า จึงมี RST (Q) กำกับไว้ ในเวลาที่ t_3 TA จะทำการอัพเดท Q แต่เมื่อตรวจสอบแล้วพบ RST(Q) เมื่อเห็นเช่นนี้แล้ว TA จะ โรลแบ็ก ทันที



รูปที่ 2.7 แสดงการแก้ไขปัญหาข้อที่ 2.5.2.3

จากรูปที่ 2.7 เห็นได้ว่า TB เป็น ทรานแซคชันที่มีอายุน้อยกว่า TA เมื่อมีการ อ่าน Z และ อัพเดท Z แล้วดังนั้น จะมี WST(Z) กำกับอยู่ที่ TB ด้วย เมื่อ TA มีการ อ่าน Z จะเห็น ค่า WST(Z) ดังนั้น TA จะ โรลแบ็ก ทันที



รูปที่ 2.8 แสดงการแก้ไขปัญหาข้อที่ 2.5.2.4

จากรูปที่ 2.8 จะใช้หลักการเดียวกับรูปที่ 2.7 คือ TB เป็น ทรานแซคชัน ที่มีอายุน้อยที่สุดแล้วเพิ่มข้อมูล Z ทำให้มี WST(Z) กำกับอยู่ที่ TB ด้วย เมื่อ TA มีการ อ่านค่า Z จะเห็น ค่า WST(Z) ดังนั้น TA จะ โรลแบ็ก ทันที

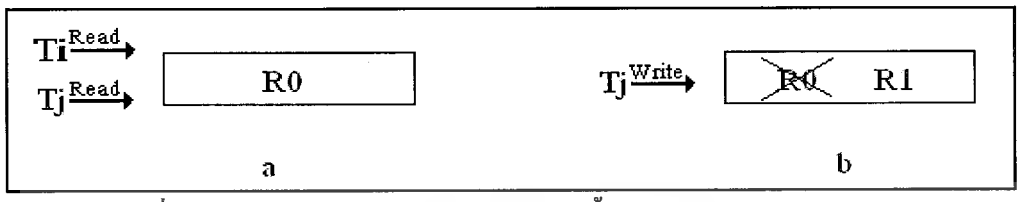
ในกฎและขั้นตอนของการทำไทม์แสตป์ยังไม่ได้รองรับถึงความสามารถในการแก้ไขปัญหาข้อที่ 2.5.2.2 ทำให้ต้องมีการเพิ่มความสามารถอื่นๆเข้าไปอีกเพื่อให้สามารถแก้ไขปัญหาข้อที่ 2.5.2.2 ได้ ซึ่งวิธีการ 2 วิธีที่ให้เลือกใช้คือ

- การเพิ่ม คอมมิต บิต สำหรับแต่ละ ทรานแซคชัน ในกฎและขั้นตอนของการทำไทม์แสตป์ต้องการให้สามารถ อ่านได้ ต้องตรวจสอบเพิ่มว่า ทรานแซคชันบันทึกนั้น คอมมิต แล้วหรือยัง ถ้ายัง ต้องรอ จนกว่าจะ คอมมิต, ถ้า คอมมิต แล้วก็สามารถอ่านได้เลย
- การใช้ เอ็กล็อกมาช่วย ถ้ามีการ อัพเดทให้ติด เอ็กล็อกจนกว่าจะถึงจุด สิ้นสุดการทำงาน ทำตาม กฎและขั้นตอนของการทำไทม์แสตป์ถ้าสามารถ อ่านได้แต่ติด เอ็กล็อกต้องรอก่อน

2.8 มัลติเวอร์ชัน สกีม่า

มัลติเวอร์ชัน สกีม่า (Multi Version Schema) ในปัญหาของกฎและขั้นตอนของการทำไทม์แสตป์ คือ ในการแก้ปัญหาที่ 2.5.2.1, 2.5.2.3, 2.5.2.4 นั้น เมื่อเราเจอปัญหาจะให้ โรลแบ็ก

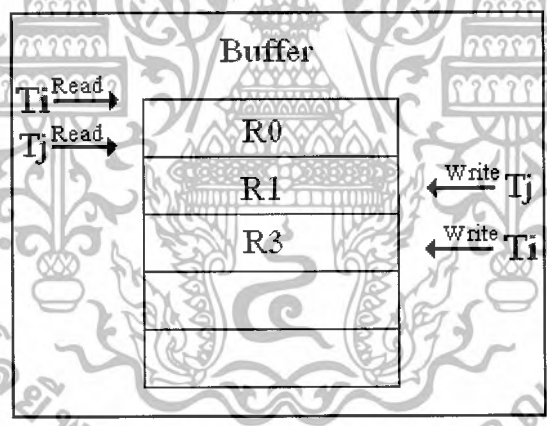
ทันที แต่การ โรลแบ็ก เป็นสิ่งที่ทำให้การทำงานเกิดความล้าช้าแล้วไม่เป็นผลดีเท่าที่ควร จึงมีความคิดที่จะหลีกเลี่ยงการเกิด โรลแบ็ก แนวคิดนี้คือการใช้ มัลติเวอร์ชัน สกรีม่า



รูปที่ 2.9 แสดงถึงวิธีการทำงานของ กฎและขั้นตอนของการทำไทม์แสตป์

โดยปกติแล้ว การ บันทึก มักจะเป็นการบันทึก ของเก่าเช่นจากรูปที่ 2.9 a เมื่อ Ti เป็น ทรานแซคชัน ที่มาก่อนแล้วกำลังอ่าน Ro และ Tj เป็นทรานแซคชันที่เกิดทีหลัง Tj จึงเป็น ทรานแซคชัน ที่เด็กกว่า เมื่อเข้ามาอ่าน Ro และทำงานเสร็จก่อน จึง บันทึก Ro ลงไปเป็น R1 เมื่อ Ti จะมาบันทึก ไม่เห็น Ro แต่เห็น R1 จึงไม่สามารถ บันทึกได้ ทำให้ โรลแบ็ก

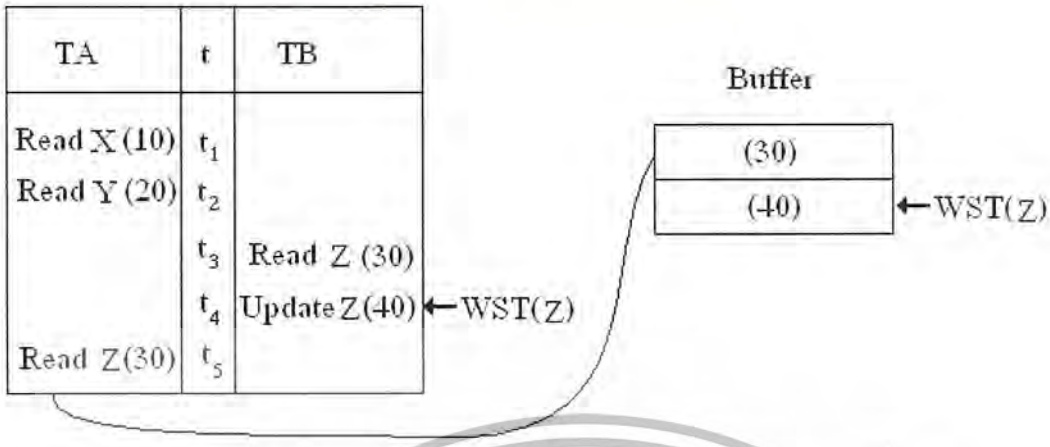
หลักการของ มัลติเวอร์ชันสกรีม่าคือเขียนเวอร์ชันใหม่เสมอไม่ให้ทับของเดิม



รูปที่ 2.10 แสดงวิธีการทำงานของ มัลติเวอร์ชันสกีม่า

จากรูปที่ 2.10 เมื่อ ในช่วงเริ่มการทำ ทรานแซคชัน เมื่อ Ti เกิดก่อน Tj และทั้ง Ti และ Tj อ่านRo ทั้งคู่ เมื่อ Tj ทำงานเสร็จก่อน Ti จึง บันทึกลง บัฟเฟอร์แต่แทนที่จะบันทึก ทับ R0 ก็สร้างเนื้อที่ใน บัฟเฟอร์ใหม่เลยแล้ว บันทึกลงเป็น R1 เมื่อ Ti ทำงานเสร็จ ก็ยังเห็น R0 อยู่ทำให้สามารถบันทึกลงบัฟเฟอร์ได้ เป็น R3

ดังที่กล่าว ตอนต้นหัวข้อมัลติเวอร์ชันสกรีม่า ได้ใช้แก้ไขปัญหาการโรลแบ็ก เมื่อใช้ไทม์แสตมป์ ในการแก้ 2.5.2. 1-2.5.2.4



รูปที่ 2.11 เป็นการใช้ มัลติเวอร์ชันสกีม่า แก้ปัญหาการเกิด โรลแบ็ก

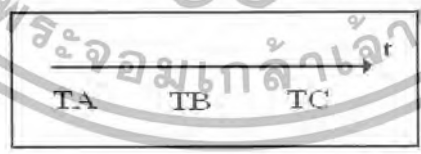
จากการใช้ ไทม์แสตมป์แก้ปัญหาข้อที่ 2.5.2.3 และข้อที่ 2.5.2.4

จากรูปที่ 2.11 จะเห็นได้ว่า TA จะไม่ โรลแบ็ก เพราะว่าไปอ่าน ค่า Z เก่าที่มีอยู่เดิมก่อนที่ TB จะ อัพเดท ลงไป และการจัดการป้องกันการ โรลแบ็ก ของการแก้ปัญหา ข้อที่ 2.5.2.4 สามารถใช้ขั้นตอนนี้แก้ปัญหาข้อที่ 2.5.2.3 จัดการได้เช่นกัน

แต่การใช้ มัลติเวอร์ชันสกีม่า จะไม่ช่วยแก้ปัญหาการเกิด โรลแบ็ก จากการใช้ ไทม์แสตมป์แก้ปัญหาข้อที่ 2.5.2.1 ดูจากรูปที่ 2.6 เช่นถ้าเรากำหนดให้ TA อัพเดทค่าQ เป็น Q+1 แล้ว TA จะไม่สามารถ อัพเดทอยู่ดี เนื่องจากค่า Q ได้อ่านไปแล้ว

เราจะเห็นได้ว่า ปัญหาข้อที่ 2.5.2.1 ถ้าใช้ ล็อค-เบส เทคนิค จะเกิดปัญหา เด็ดล็อกและถ้าเราใช้กฎและขั้นตอนของการทำไทม์แสตป์ก็จะเกิดปัญหา โรลแบ็ก

การเลือกใช้ให้ถูกเวอร์ชันทำได้อย่างไร



รูปที่ 2.12 แสดงแกนเวลาของการเกิดแต่ละ ทรานแซกชัน

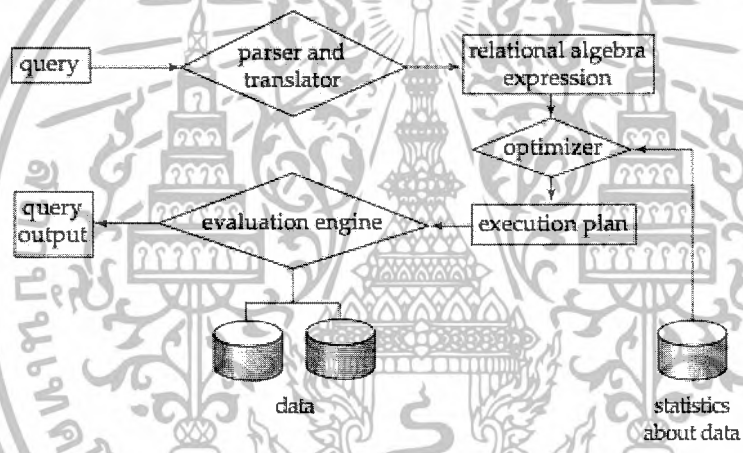
จากตัวอย่างที่ 2.10 ถ้า TC เป็น ทรานแซคชัน ที่เกิดหลังสุดและต้องการจะอ่าน ข้อมูลออกไป ควรเลือกเวอร์ชัน ใด คำตอบคือ เลือกเวอร์ชันที่ TB ได้ บันทึกลงไปเพราะเป็นทราน-แซคชันที่มีอายุน้อยที่สุดรองจากตัวเองลงมา จึงเชื่อได้ว่าเป็นการบันทึกที่ใหม่ที่สุด

ในทางปฏิบัติแล้ว การที่จะเก็บเวอร์ชันต่างไว้ในบัฟเฟอร์เป็นการทำให้เปลืองเนื้อที่ดิสข้อมูลเป็นอย่างมาก ดังนั้นวิธีที่ปฏิบัติกันโดนทั่วไป จะเก็บไว้เพียง 3 เวอร์ชันเท่านั้น และก่อน

หน้าที่จะลบทิ้งไปจะมีการสำรองเก็บไว้ใน Old values (BIJ) ก่อนดังนั้นเมื่อ Ti ต้องการ R0 แต่ในบัฟเฟอร์ไม่มีแล้วใน ออราเคิล และ โพสเกรสเอสคิวแอล เมื่อทรานแซคชันมีการ คอมมิต แล้วจะยังไม่เอาเนื้อที่ใน Recycle Old values (BIJ)มาใช้ใหม่เพราติดค่า R ต่างๆ ที่เก็บไว้และใช้วิธีไล่เนื้อที่ คือการเอาข้อมูลใหม่มาเก็บแทนที่ข้อมูลที่อยู่นานๆ

2.9 การประมวลผลสอบถาม

การประมวลผลสอบถาม(Query Processing) จะทำการประมวลผลเป็นขั้นตอนต่าง ๆ โดยจะนำคำสั่งภาษาสอบถามที่ผู้ใช้ต้องการมาตรวจสอบกับกฎและไวยากรณ์ (rule of grammars) ของภาษาสอบถามนั้น ๆ ระบบการจัดการ ฐานข้อมูลจะใช้กลยุทธ์การประมวลผล (execution strategy) สำหรับ หาผลลัพธ์ของคำสั่งภาษาสอบถาม



รูปที่ 2.1.3 ขั้นตอนการทำงานของการประมวลผลสอบถาม

2.9.1 ขั้นตอนการทำงานของการประมวลผลคำถาม

2.9.1.1 การตรวจเช็คไวยากรณ์ และแปลงคำถาม (Parser and translator)

เมื่อผู้ใช้ ได้ทำการป้อนคำสั่งภาษาสอบถามระบบจัดการฐานข้อมูลโดย ตัวประมวลผลสอบถาม (query processor) จะทำการตรวจเช็คกฎและหลักเกณฑ์ (syntax checking) ของภาษาสอบถามนั้น เมื่อตรวจ เช็คแล้วหากพบว่าคำสั่งภาษาสอบถามที่ป้อนเข้ามายังระบบมีความผิดพลาดไม่เป็นไปตามกฎเกณฑ์ ของภาษาสอบถาม ระบบจัดการฐานข้อมูลโดยตัวประมวลผลสอบถามก็จะส่งข้อความไปยังผู้ใช้ให้ผู้ใช้ปรับแก้ให้ถูกต้อง แต่ถ้าตรวจเช็คแล้วคำสั่งนั้นถูกต้องตามกฎไวยากรณ์ของภาษาสอบถาม จะแปลงทำให้ได้รูปแบบของภาษาสอบถาม (intermediate from of query) เพื่อให้ระบบจัดการฐานข้อมูลสามารถเข้าใจได้ ซึ่งผลที่แปลงจะอยู่

ในรูปของสมการพีชคณิตแบบสัมพันธ์ (relational algebraic expression) เพื่อให้ระบบจัดการฐานข้อมูลเข้าใจและทำตามผลที่ได้นั้น

2.9.1.2 การแปลงการสอบถามให้อยู่ในรูปที่เหมาะสมที่สุด (Query Optimizer)

เมื่อได้สมการพีชคณิตแบบสัมพันธ์แล้วระบบการจัดการฐานข้อมูลจะทำการประมวลผลสมการพีชคณิตแบบสัมพันธ์นั้น เพื่อให้ได้แผนการเข้าถึงข้อมูลในการประมวลผล การที่จะให้ได้แผนการ เข้าถึงข้อมูล (access) นั้นจะมีวิธีการเข้าถึงฐานข้อมูลได้หลายวิธี ระบบการจัดการฐานข้อมูลจะทำ การหาวิธีที่เหมาะสมในการเข้าถึงฐานข้อมูลโดยจะมีเทคนิคต่างๆในการเข้าถึงข้อมูลที่ใช้เวลาน้อยที่สุด หรือเรียกว่าการทำออพติไมเซชัน (optimisation)ซึ่งเทคนิคในการเข้าถึงข้อมูลที่เหมาะสม เหล่านี้ เช่น เทคนิคการซอร์ตและการเมอร์จ (sorting and merge method) เป็นการเรียงลำดับข้อมูลในตารางข้อมูลโดยในระบบจัดการฐานข้อมูลจะมีโปรแกรมสำหรับจัดเรียงลำดับข้อมูลเพื่อให้การประมวล ผลประหยัดเวลาในการประมวลผลลง เทคนิคดีคอมโพสิชัน (decomposition method) เป็นการแตกคำสั่งในภาษาสอบถามให้เป็นโมดูลย่อยๆเพื่อง่ายต่อการประมวลผล หรือเทคนิคโอเปอร์เรเตอร์ กราฟ (operator graph) เป็นการนำสมการพีชคณิตที่ได้จากการตรวจเช็คกฎไวยากรณ์ของภาษา แล้วนำมาหาทางการประมวลผลที่ทำให้การประมวลผลนั้นใช้เวลาน้อย เทคนิคต่างๆ เหล่านี้จะทำ ให้ได้แผนการในการประมวลผลที่เหมาะสมว่าระบบจะต้องประมวลผลอย่างไร เพื่อให้ได้วิธีการ เข้าถึงฐานขอมูลที่มีประสิทธิภาพและใช้เวลาในการประมวลผลน้อยที่สุด

2.9.1.3 การประมวลผลชุดคำสั่ง (Evalution Engin)

เป็นการประมวลผลคำที่ถูกเลือกแล้วเพื่อหาคำตอบและส่งข้อมูลออกไป

2.9.2 Basic Algorithm

เป็นวิธีการค้นหาข้อมูลของออฟติไมเซอร์(อ๊อบติไมเซอร์) แบบต่างๆ ซึ่งมีดังต่อไปนี้

- Primary Index คืออินเดกซ์ ที่ข้อมูลในคอลัมถูกเก็บแบบเรียงลำดับ
- Secondary Index คืออินเดกซ์ที่ข้อมูลในคอลั่มถูกเก็บแบบไม่เรียงลำดับ

2.9.2.1 Linear Search

เป็นการสแกนแต่ละดาต้าบล๊อกเพื่อหาข้อมูลที่ตรงกับเงื่อนไขในคำถาม *Cost estimate = br block transfers* ถ้าเงื่อนไขอยู่บน คีย์แอดทิบิว ค่า Cost สามารถหยุดหาได้เลยเมื่อพบแล้ว *Cost = (br/2) block transfers*

2.9.2.2 Binary Search

จะถูกใช้เมื่อข้อมูลที่เก็บอยู่ถูกเรียงลำดับอยู่แล้วและเงื่อนไขในการค้นหาเป็นเงื่อนไขเท่ากับ

$$Cost\ estimate = \lceil Log(2)\ br \rceil + \lceil nr/V(A,r) \square fr \rceil \qquad \text{(สมการที่ 2.1)}$$

เมื่อ nr คือ จำนวน row ของตาราง

br คือ จำนวน Data Block ทั้งหมดของตาราง

fs คือ Blocking factory

V(A,r) คือจำนวน Distinct Value ของแอดทิบิว A ในตาราง r

2.9.2.3 Primary index on candidate key, equality

จะถูกใช้เมื่อมีการค้นหาใน key ที่เป็น Primary Index และเป็นเงื่อนไขเท่ากับ

Cost estimate HT+1 (สมการที่ 2.2)

เมื่อ HT=ความสูงของ tree

2.9.2.4 Primary index on nonkey, equality

เมื่อมีการค้นหาบน Primary Index ที่ไม่ใช้ Key แอดทิบิว และใช้เงื่อนไขเท่ากับ

$$Cost\ estimate = HT + \lceil row\ return/fr \rceil \quad \text{(สมการที่ 2.3)}$$

2.9.2.5 Equality on search-key of secondary index

เมื่อมีการค้นหาเงื่อนไข ที่เป็นแบบ เท่ากับบน Secondary Index Cost estimate ใช้สูตรการคำนวณเดียวกับ 2.9.2.3 เมื่อผลลัพธ์มี 1 แถวและ ใช้สูตรการคำนวณเดียวกับ 2.9.2.4 เมื่อผลลัพธ์มีหลายแถว

2.10 เอสคิวแอล จอยน์

เอสคิวแอล จอยน์ (SQL join) แบ่งออกได้หลายชนิดดังนี้

2.10.1 ชนิดของการจอยน์

2.10.1.1 Equi-Join

เป็นการจอยน์โดยใช้ เครื่องหมายเท่ากับ (equality operator "=") ในการจอยน์เช่น

```
SELECT e.ename,d.dname
FROM emp e , dept d
WHERE e.deptno = d.deptno ;
```

2.10.1.2 Non-equi join

เป็นการใช้โอปาเรเตอร์อื่นที่ไม่ใช่เครื่องหมายเท่ากับ ในการจอยน์เช่น < . >, BETWEEN

```
SELECT e.ename,e.job,e.sal,sg.grade
FROM emp e , salgrade sg
WHERE e.sal BETWEEN sg.losal AND sg.HISAL ;
```

2.10.1.3 Inner join หรือ Simple join

การจอยน์แบบนี้ จะรีเทิร์นแถวออกมาตามเงื่อนไขใน WHERE และ ถ้าคอลั่มในเงื่อนไขแถวไหนมีค่าเป็น NULL ก็จะไม่รีเทิร์น แถวนั้นออกมา

2.10.1.4 Natural Join

ก็คือ Equi-Join ที่ไม่ได้ระบุว่าจะเอาคอลั่มอะไรมาเป็นตัวเท่ากับ ระหว่างกันซึ่งออราเคิล จะไปดูว่าคอลั่มไหนที่เป็นคอลั่มชื่อเหมือนกันของตารางที่จะมา จอยน์ มันก็จะจับคอลั่มนั้นแหละมาเป็นตัวเท่ากับในการจอยน์ เช่น

```
SELECT e.ename,d.dname
FROM emp e NATURAL JOIN dept d ;
```

2.10.1.5 Outer join

เป็นการจอยน์ ที่ขยายมาจาก inner join โดยจะสามารถแสดงแถวที่คอลั่มในเงื่อนไขการจอยน์ เป็น NULL สามารถแบ่ง outer join ออกเป็นประเภทย่อยๆ ตามการแสดงแถวที่ NULL ได้ คือจะแสดงแถวในตารางที่อยู่ทางซ้ายมือของเงื่อนไข WHERE ออกมาทั้งหมด แม้ว่าตารางทางขวามือของ WHERE จะไม่ตรงกับเงื่อนไขก็ตาม ตารางทางขวามือก็จะรีเทิร์น ค่าเป็น NULL ออกมาให้

```
SELECT d.department_id,e.last_name
FROM departments d , employees e
WHERE d.department_id = e.department_id(+)
ORDER by d.department_id ;
```

2.10.1.6 Right outer join

จะแสดงแถวในตารางที่อยู่ทางขวามือของเงื่อนไข WHERE ออกมาทั้งหมดแม้ว่าตารางทางซ้ายมือของ WHERE จะไม่ตรงกับเงื่อนไข แต่ตารางทางซ้ายมือก็จะรีเทิร์น ค่าเป็น NULL ออกมาให้

```
SELECT d.department_id,e.last_name
FROM departments d , employees e
WHERE d.department_id(+) = e.department_id
ORDER by d.department_id ;
```

2.10.1.7 Full outer join

จะแสดงแถวในตารางที่อยู่ทั้งทางซ้ายและขวาของเงื่อนไข WHERE ออกมาทั้งหมด ตารางไหนไม่ตรงกับเงื่อนไข ก็จะรีเทิร์น ค่าออกมาเป็น NULL

```
SELECT d.department_id,e.last_name
FROM departments d FULL OUTER JOIN employees e
ON d.department_id = e.department_id
ORDER by d.department_id ;
```

หมายเหตุ การใช้เครื่องหมาย (+) ตามหลังตารางทีเป็นฟากส่วนที่หายไปนั้น (ฟากที่ไม่ใช่ outer join) เป็นรูปแบบแบบเก่า คลาสสิค ตามมาตรฐาน SQL ANISI/86 ซึ่งจะยังไม่มี FULL OUTER JOIN แต่รูปแบบ แบบใหม่ตามมาตรฐาน SQL ANSI/92 จะมี FULL OUTER JOIN แล้ว และรูปแบบ นี้จะไม่ใช้ (+)

2.10.1.8 Self Join

ก็คือการจอยน์ ภายในตารางเดียวกัน เนื่องจากการออกแบบตารางนั้นมี คอลั่มอย่างน้อย 2 คอลั่มมีความสัมพันธ์ซึ่งกันและกัน เราสามารถจอยน์ ตัวเองได้โดยการเอาตารางเดียวกันนี้แหละ ทำให้มันดูเป็น 2 ตาราง โดยตั้งชื่อ alias (ชื่อเล่น) มันจึงมองเหมือนเป็นตาราง 2 ตาราง มาจอยน์ กันแบบปกติ

2.10.2 การเลือก แบบแผน(Plans) สำหรับการจอยน์

2.10.2.1 อ๊อบติไมเซอร์(Optimizer)

จะค้นเลือกทางที่มีให้ผลลัพธ์ออกมามากที่สุด 1 โรล ก่อน เช่น UNIQUE หรือ PRIMARY KEY Constraints ก่อน จากนั้นก็จะเอา ตาราง นั้นมาว่างก่อนในลำดับของการ จอยน์ จากนั้นก็ทำการตั้งค่าจอยน์ของ ตาราง

2.10.2.2 สำหรับการ จอยน์แบบ outer join

นั้น ตาราง ที่มีเงื่อนไขใน outer join นั้นจำเป็นต้องทำที่หลัง ตาราง อื่นที่อยู่ใน เงื่อนไขการ จอยน์ อ๊อบติไมเซอร์ จะไม่พิจารณาการจอยน์ ที่ ฝ่าฝืนกฎนี้ เช่น ตาราง ที่อยู่ใน sub query จะต้องถูกทำก่อน ตาราง ที่อยู่ในเงื่อนไขข้างนอก

2.10.2.3 cost ของ nested loop join

นั้นขึ้นอยู่กับ จำนวนของโรล ของ ตาราง ที่อยู่ด้านนอก และสอดคล้องกับแต่ละ โรลที่อยู่ใน ตาราง ด้านในที่อยู่ใน memory ซึ่ง อ๊อบติไมเซอร์ จะใช้สถิติที่เก็บอยู่ในพจนานุกรม ข้อมูลมาเป็นตัวประมาณค่า cost

2.10.2.4 cost ของ sort merge join

นั้นขึ้นอยู่กับ cost ของการอ่านข้อมูลขึ้นมา เรียงลำดับ ในเมมโมรี่

2.10.2.5 cost ของ hast join

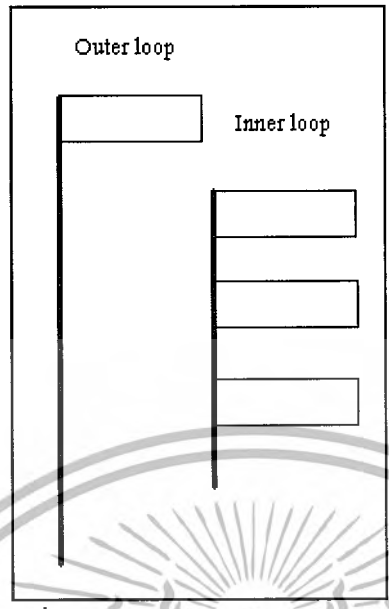
นั้นขึ้นอยู่กับการสร้าง hash ตาราง ฝั่งใดฝั่งหนึ่งที่นำมา จอยน์

2.10.3 Join Algorithm

2.10.3.1 Nested loop join

จะมีประสิทธิภาพดีเมื่อ มีข้อมูล ในการจอยน์ น้อยๆ และถ้าเงื่อนไขในการจอยน์ มี เส้นทางที่ดีในการเข้าถึง อีกตาราง ซึ่งมีขั้นตอนการทำงานดังนี้

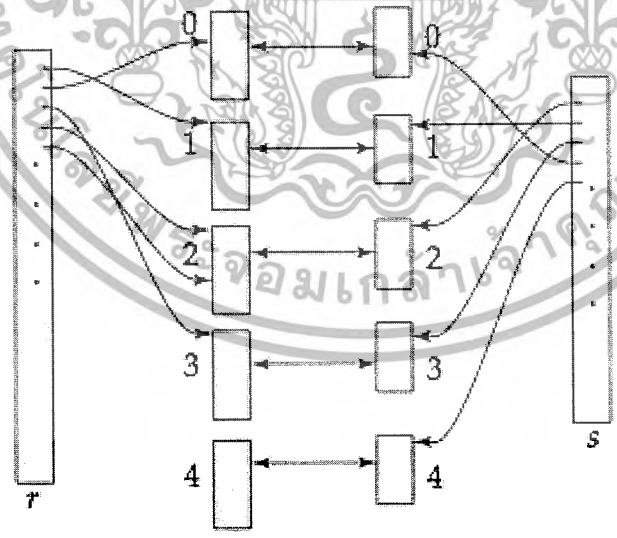
- อ๊อบติไมเซอร์ จะทำการกำหนด ตารางด้านนอก
- ตารางอื่นจะถูกกำหนดให้เป็น ตารางด้านใน
- แต่ละโรลในตารางด้านนอก จะถูกเปรียบเทียบกับ ทุกโรล ในตาราง ด้านใน ซึ่งใน execution plan จะแสดงดังนี้
  - NESTED LOOPS
  - outer_loop
  - inner_loop



รูปที่ 2.14 จำลองการทำ Nested loop

2.10.3.2 Hash join

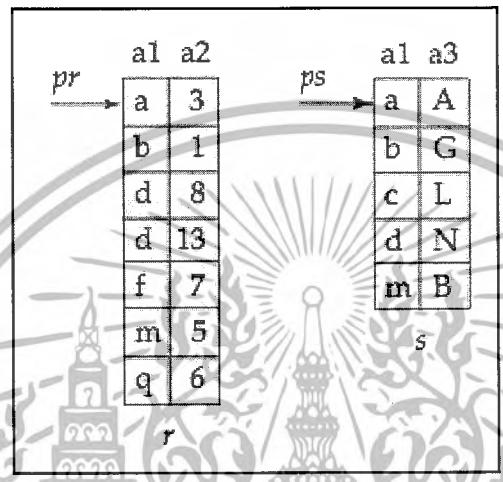
จะถูกใช้เมื่อมีการ จอยน์ ข้อมูลจำนวนมาก อ๊อบติไมเซอร์ จะใช้ตารางที่เล็กที่สุดในตารางที่ต้องการ จอยน์ มาสร้าง hash ตาราง ไว้ใน เมมโมรี่ จากนั้นทำการ สแกน ตารางที่ใหญ่กว่า ซึ่งการทำงานนั้นจะเอาข้อมูลใน คอลั่ม ที่ต้องการ join กันมาผ่าน Hash function ทั้งสองข้าง



รูปที่ 2.15 จำลองการทำ hash join

2.10.3.3 Sort Merge Join

จะมีประสิทธิภาพเมื่อ ข้อมูลที่ต้องการจอยน์นั้นมีการ เรียงลำดับอยู่แล้ว แต่ถ้าข้อมูลยังไม่มีการ sort อ๊อบติไมเซอร์ ก็จะทำการเรียงลำดับ ใน temporally เทเบิลสเปรซ ก่อน และใช้กับเงื่อไขที่ไม่ใช่ เท่ากับ เช่น <, <=, >, >= Sort Merge Join จะทำงานกับข้อมูลจำนวนมากได้ดีกว่า Nested loop join



รูปที่ 2.16 จำลองการทำ Sort merge join

2.10.3.4 Cartesian Joins

จะถูกใช้เมื่อมีการ join table แบบไม่มีเงื่อนไขในการ join ซึ่งจะนำเอา row ของทั้ง 2 ตารางที่ join กันมาทำ Cartesian Product กัน

2.11 กฎของการควบคุมความถูกต้องของข้อมูล

กฎของการควบคุมความถูกต้องของข้อมูล (Integrity Constraint)มีหลักอยู่ 5 อย่างนี้คือ

2.11.1 Not NULL

การที่กำหนดให้ คอลั่ม มีค่าที่เป็น NULL ได้หรือไม่ ถ้ากำหนดเป็น Not NULL จะไม่สามารถทำการ เพิ่มข้อมูลข้อมูลลงใน คอลั่ม นั้นได้ โดยถ้าไม่กำหนดค่าเริ่มต้น จะเป็น NULL คือเก็บค่า NULL ได้

ตัวอย่าง

```
CREATE TABLE ri_not_null(
A NUMBER NOT NULL,
B NUMBER      NULL,
C NUMBER );
```

2.11.2 Unique Key

การที่กำหนดให้ คอลั่ม จะไม่มีค่าที่ซ้ำกันหรือเหมือนกัน ถ้าพยายามเพิ่มข้อมูลข้อมูลที่มีค่าซ้ำกันจะไม่สามารถทำการเพิ่มข้อมูลข้อมูลนั้นได้ แต่ถ้าหากไม่มีการกำหนดให้ คอลั่ม นั้นเป็น NOT NULL ค่า NULL ก็จะเพิ่มข้อมูลลงใน คอลั่ม นั้นได้โดยไม่ถือว่าเป็นการซ้ำและจะเพิ่มข้อมูลค่า NULL กี่ค่าก็ได้ใน คอลั่ม นั้น

ตัวอย่าง

```
CREATE TABLE ri_unique(
A NUMBER UNIQUE,
B NUMBER);
```

หากต้องการที่จะทำให้เพิ่มข้อมูลค่าที่ซ้ำ กันได้ทำได้ดังนี้

```
ALTER TABLE ri_unique DROP CONSTRAINT SYS_C001463
```

ในทางตรงข้ามหากต้องการที่จะทำให้ คอลั่ม เป็น Unique ก็ทำได้ดังนี้

```
ALTER TABLE ri_unique ADD CONSTRAINT uq_ri_b UNIQUE(b);
```

2.11.3 Primary Key

การที่ คอลั่ม นั้นมีคุณสมบัติที่เป็น NOT NULL กับ Unique key

2.11.4 Foreign Key

การที่ คอลั่ม นั้นจะทำอ้างถึง Primary Key ของอีกตารางหนึ่ง โดยที่ คอลั่ม นั้นจะต้องมีค่าเดียวกันกับค่าที่จะอ้างถึงของอีกตารางหนึ่ง เรียกอีกอย่างหนึ่งว่า Referential Integrity Constraint

2.11.5 Check

การที่ทำการกำหนดค่าต่ำสุดหรือสูงสุดที่จะนำมา insert ลงไปใน คอลั่ม นั้นได้

ตัวอย่าง

```
CREATE TABLE ri_check(
A NUMBER CHECK(A BETWEEN 0 AND 100),
B NUMBER
);
```

2.12 การประมวลผลแบบกระจาย

การประมวลผลแบบกระจาย (Distributed) โดยหลักการมีดังนี้

2.12.1 Distributed Data Storage

2.12.1.1 การแยกข้อมูล(Data Fragmentation)

วิธีการเก็บข้อมูลแบบนี้รีเลชัน r จะถูกแบ่งออกเป็นรีเลชันย่อย ๆ $r_1,r_2,\ldots,r_n$ ซึ่งรีเลชันเหล่านี้เมื่อนำกลับมารวมกันจะได้ผลลัพธ์เป็นรีเลชัน r เหมือนเดิม ซึ่งในการแบ่งรีเลชันมีอยู่ 2 วิธีคือ การแบ่งรีเลชันตามแนวนอน(horizontal fragmentation) และ การแบ่งรีเลชันตามแนวตั้ง vertical fragmentation) โดยการแบ่งรีเลชันตามแนวนอน จะแบ่งแต่ละทูเพิลของรีเลชัน r ไปเป็นเป็นหลาย ๆ ส่วน ส่วนการแบ่งรีเลชันตามแนวตั้งจะเป็นการแยกรีเลชัน(decomposition)โดยการแตกโครงสร้างของรีเลชัน r ออกเป็นหลาย ๆ รีเลชันย่อย ในที่นี้จะยกตัวอย่างรีเลชัน Employee ซึ่งมีโครงสร้างของรีเลชันดังนี้

Employee = (empid, name, state, age, salary)

รีเลชัน Emloyee แสดงได้ดังรูปที่ 2.17

Empid	Name	State	Age	Salary
43125	Nancy	New York	28	4500
43126	Andrew	Texas	26	4250
43129	Janet	New York	32	5000
43135	Margaret	New York	47	8500
43136	Steven	Texas	36	6500
43139	Michael	Texas	24	4000
43140	Robert	Texas	29	4500

รูปที่ 2.17 ตัวอย่างรีเลชัน Employ

2.12.1.2 Horizontal Fragmentation

รีเลชัน r ถูกแบ่งออกเป็นรีเลชันย่อย ๆ r_1, r_2, …, r_n โดยแต่ละทูเปิลของรีเลชันจะต้องถูกแบ่งออกไปอยู่ใน รีเลชันย่อย ๆ เหล่านั้น

รีเลชันย่อยแต่ละอันจะเป็นส่วนหนึ่งของรีเลชัน r เราใช้สัญลักษณ์ P_i เพื่อแทนรีเลชันย่อย r_i ดังนี้

$$r_i = s\, P_i(r) \qquad \text{(สมการที่ 2.4)}$$

ดังนั้น เราสามารถสร้างรีเลชัน r ได้ใหม่โดยใช้วิธีการรวม(Union)ทุก รีเลชันย่อย ได้ดังนี้

$$r = r_1 \cup r_2 \cup \ldots \cup r_n \qquad \text{(สมการที่ 2.5)}$$

จากตัวอย่าง สมมุติว่า r ถูกแบ่งออกเป็นรีเลชันย่อย ๆ n รีเลชัน แต่ละรีเลชันย่อยประกอบด้วยข้อมูลที่เป็นของแต่ละรัฐ จากตัวอย่าง ถ้าในระบบมีรัฐอยู่ 2 รัฐ คือ New york และ Texas จะทำการแบ่งรีเลชันออกเป็น 2 รีเลชัน คือ

$$employee_1 = s_{state='New\ york'}\ (employee) \qquad \text{(สมการที่ 2.6)}$$

$$employee_2 = s_{state='Texas'}\ (employee) \qquad \text{(สมการที่ 2.7)}$$

ทั้ง 2 รีเลชันย่อย แสดงได้ดังรูปที่ 2.18 โดยที่รีเลชันย่อย $employee_1$ จะถูกเก็บไว้ที่ไซต์ New york และรีเลชันย่อย $employee_2$ จะถูกเก็บไว้ที่ไซต์ Texas

$employee_1$

Empid	Name	State	Age	Salary
43125	Nancy	New York	28	4500
43129	Janet	New York	32	5000
43135	Margaret	New York	47	8500

$employee_2$

Empid	Name	State	Age	Salary
43126	Andrew	Texas	26	4250
43136	Steven	Texas	36	6500
43139	Michael	Texas	24	4000
43140	Robert	Texas	29	4500

รูปที่ 2.18 แสดงการแบ่งรีเลชันตามแนวนอนของรีเลชัน empl

2.12.1.3 Vertical Fragmentation

การแบ่งรีเลชันตามแนวตั้งจะเป็นการแยกแอตทริบิวต์ของรีเลชัน r ออกเป็นรีเลชันย่อยๆ ซึ่งแต่ละรีเลชันย่อยจะประกอบไปด้วยแอตทริบิวต์ของรีเลชัน r นั่นคือ ถ้าเราให้ R เป็นสกีมาของรีเลชัน r และสับเซตของแอตทริบิวต์ $R_1, R_2, \ldots, R_n$ ของ R คือ

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

แต่ละรีเลชันย่อย r_i ของ r คือ

$$r_i = P\, R_i(r) \qquad \text{(สมการที่ 2.8)}$$

รีเลชันย่อยที่ได้นำกลับมาสร้างเป็นรีเลชัน r ได้ใหม่โดยการทำ natural join

$$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \ldots \bowtie r_4 \qquad \text{(สมการที่ 2.9)}$$

ในการทำ natural join เพื่อที่จะให้ผลลัพธ์ของการจอยน์ ได้เป็นรีเลชัน r เหมือนเดิม สามารถทำได้โดยการใส่แอตทริบิวท์ที่ทำหน้าที่เป็นคีย์หลักของ R เข้าไปในแต่ละ R_i โดยทั่วไปแล้วก็สามารถใช้ซูเปอร์คีย์แทนได้เช่นกัน เพื่อความสะดวกเราจะทำการเพิ่มแอตทริบิวต์พิเศษเข้าไปในรีเลชัน จะเรียกว่า tid โดยที่ค่าของแอตทริบิวต์ tid จะถูกกำหนดให้มีค่าไม่ซ้ำกันเพื่อใช้ในการจำแนกทูเปิลแต่ละทูเปิลออกจากกัน ดังนั้นแอตทริบิวต์ tid จะทำหน้าที่เป็นคีย์คู่แข่งของรีเลชัน r และจะถูกนำไปรวมเข้ากับแต่ละรีเลชันย่อยที่แยกออกมา

พิจารณารีเลชัน employee ทำการเพิ่มแอตทริบิวต์ tid ดังรูปที่ 2.19

Tid	Empid	Name	State	Age	Salary
1	43125	Nancy	New York	28	4500
2	43126	Andrew	Texas	26	4250
3	43129	Janet	New York	32	5000
4	43135	Margaret	New York	47	8500
5	43136	Steven	Texas	36	6500
6	43139	Michael	Texas	24	4000
7	43140	Robert	Texas	29	4500

รูปที่ 2.19 แสดงรีเลชัน employee และแอตทริบิวต์ tid

แสดงการแยกรีเลชันรีเลชัน employee และแอตทริบิวต์ tid ออกเป็น 2 รีเลชันย่อย

employee-schema-1= (Tid, empid, state) (สมการที่ 2.10)

employee-schema-2 = (Tid, name, age, salary) (สมการที่ 2.11)

รีเลชันย่อยทั้ง 2 ในรูปที่ 2.20 เป็นผลมาจากการดำเนินการดังนี้

$employee_1$ = P employee-schema-1 (employee) (สมการที่ 2.12)

$employee_2$ = P employee-schema-2 (employee) (สมการที่ 2.13)

$Employee_1$

Tid	Empid	State
1	43125	New York
2	43126	Texas
3	43129	New York
4	43135	New York
5	43136	Texas
6	43139	Texas
7	43140	Texas

$Employee_2$

Tid	Name	Age	Salary
1	Nancy	28	4500
2	Andrew	26	4250
3	Janet	32	5000
4	Margaret	47	8500
5	Steven	36	6500
6	Michael	24	4000
7	Robert	29	4500

รูปที่ 2.20แสดงการแบ่งรีเลชันตามแนวตั้งของรีเลชัน employee

ดังนั้นเราสามารถสร้างรีเลชัน employee จากรีเลชันย่อยได้จาก

P employee-schema ($employee_1 \bowtie employee_2$) (สมการที่ 2.14)

นิพจน์ $employee_1 \bowtie employee_2$ เป็นรูปแบบพิเศษของการทำ natural join โดยจะใช้แอตทริบิวต์ tid ในการ join ระหว่างรีเลชัน แม้ว่าแอตทริบิวต์ tid จะมีประโยชน์ในการแบ่งรีเลชันตามแนวตั้ง แต่ผู้ใช้จะไม่สามารถมองเห็นแอตทริบิวต์นี้ เนื่องจากเป็นส่วนที่ระบบสร้างขึ้นภายในตัวระบบเองเพื่อจัดการกับรีเลชันย่อยเท่านั้น

2.12.2 Mixed Fragmentation

การแบ่งแบบผสมนี้ รีเลชัน r จะถูกแบ่งออกเป็นรีเลชันย่อย $r_1, r_2, \ldots, r_n$ โดยแต่ละรีเลชันเหล่านั้นอาจจะถูกแบ่งนอนหรือแนวตั้งก็ได้จากรีเลชัน r และรีเลชันย่อยแต่ละอันก็สามารถถูกแบ่งตามแนวนอนหรือแนวตั้งต่อไปได้อีก

พิจารณาจากรูปที่ 13.4 รีเลชัน employee ถูกแบ่งตามแนวตั้งเป็นรีเลชันย่อย $employee_1$ และ $employee_2$ จากนั้นเราสามารถที่จะแบ่งรีเลชันย่อย $employee_1$ ตามแนวนอนได้อีก เป็น 2 รีเลชันย่อย ดังนี้

$$employee_{11} = s_{state="New\ york"}(employee_1) \quad \text{(สมการที่ 2.15)}$$

$$employee_{12} = s_{state=="Texas"}(employee_1) \quad \text{(สมการที่ 2.16)}$$

ดังนั้นรีเลชัน r จึงถูกแบ่งออกเป็น 3 รีเลชันย่อย คือ $employee_{11}$, $employee_{12}$ และ $employee_2$ ซึ่งแต่ละ รีเลชันย่อยอาจจะถูกจัดเก็บไว้ต่างไซต์กัน

2.12.3 คอมมิต โปรโตคอล

เพื่อให้การทำทรานแซกชันบนระบบฐานข้อมูลแบบกระจายมีคุณสมบัติ Atomicity คือทรานแซกชัน T ต้อง คอมมิต ในทุก ๆ ไซต์ หรือยกเลิกการทำทรานแซกชันในทุก ๆ ไซต์ ตัวประสานงานทรานแซกชันของทรานแซกชัน T ต้องมีการทำ คอมมิต โปรโตคอล

วิธีการของ คอมมิต โปรโตคอลมีการใช้งานกันอย่างแพร่หลายคือ ทู-เฟตคอมมิตโปรโตคอล(two-phase commit protocol(2PC)) และ ทรี-เฟตคอมมิตโปรโตคอล(three-phase commit protocol(3PC)) ซึ่งจะช่วยหลีกเลี่ยงข้อเสียของ 2PC แต่การดำเนินการจะมีความซับซ้อนมากกว่า

2.12.3.1 ทรี-เฟตคอมมิตโปรโตคอล

ให้ T เป็นทรานแซกชันเริ่มต้นที่ไซต์ S_i และให้ ตัวประสานงานทรานแซกชันที่ไซต์ S_i เป็น C_i

2.12.3.2 คอมมิต โปรโตคอล

เมื่อทรานแซกชัน T ได้ทำการประมวลผลเสร็จสิ้น นั่นคือ ทุก ๆ ไซต์ที่ทรานแซกชัน T ได้ประมวลผลได้แจ้งกลับมาที่ C_i ว่า ได้ทำทรานแซกชัน T เสร็จเรียบร้อยแล้ว ตัวประสานงานทรานแซกชัน C_i จะเริ่มทำโปรโตคอล 2PC

Phase 1

C_i เพิ่มเรคอร์ด <prepare T> ลงไปใน log ไฟล์บน stable storage จากนั้นก็จะส่งสัญญาณ prepare T ไปให้กับทุก ๆ ไซต์ที่ประมวลผลทรานแซกชัน T เมื่อไซต์อื่น ๆ ได้รับสัญญาน ตัวจัดการทรานแซกชันจะต้องตอบกลับไปหาตัวประสานงานทรานแซกชันว่าจะทำการ คอมมิต ทรานแซกชันได้หรือไม่ ถ้าไม่ได้จะเพิ่มเรคอร์ด <no T> ลงไปใน log ไฟล์และส่งสัญญาณ abort T กลับไปที่ C_i ถ้าได้จะเพิ่มเรคอร์ด <ready T> ลงไปใน log ไฟล์ และตัวจัดการทรานแซกชันจะส่งสัญญาณ ready T กลับไปที่ C_i

Phase 2

ทรานแซกชัน T ทรานแซกชัน T จะ คอมมิต ได้ก็ต่อเมื่อได้รับสัญญาณ ready T จากทุก ๆ ไซต์ที่มีส่วนร่วมในการทำทรานแซกชัน นอกเหนือจากนั้นทรานแซกชัน T จะต้องถูกยกเลิก ซึ่งก็จะต้องทำการเพิ่มเรคอร์ด <commit T> หรือ <abort T> ลงไปใน log ไฟล์ ถ้าทรานแซกชัน T สามารถ คอมมิต ได้ หรือ ยกเลิกการทำทรานแซกชัน จากนั้น ตัวประสานงานทรานแซกชันก็จะส่งสัญญาณ คอมมิต T หรือ abort T ไปให้กับทุกๆ ไซต์ที่ร่วมกันทำทรานแซกชัน เมื่อแต่ละไซต์ได้รับสัญญาณก็จะทำการบันทึกเรคอร์ดนี้ลงไปใน log ไฟล์จะเห็นว่าสัญญาณ ready T มีความสำคัญมาก เนื่องจากไซต์ต่าง ๆ ที่ทำทรานแซกชัน T จะต้องส่งสัญญาณ ready T กลับไปให้ตัวประสานงานทรานแซกชัน เพื่อบอกกับตัวประสานงานทรานแซกชันว่าพร้อมที่จะ คอมมิต แล้ว ซึ่งสัญญาณนี้จะมีผลต่อการทำ คอมมิต หรือ abort ของทรานแซกชัน จากลักษณะดังกล่าวอาจเป็นไปได้ว่าเมื่อไซต์หนึ่งได้ทำการส่งสัญญาณ ready T ไปให้ตัวประสานงานทรานแซกชันแล้ว อาจเกิดความล้มเหลวของไซต์ได้ ซึ่งจากโปรโตคอล 2PC เมื่อตัวประสานงานทรานแซกชันได้รับสัญญาณ ready T หรือ abort T ครบ ก็จะส่งสัญญาณ คอมมิต T หรือ abort T ไปให้กับไซต์ที่ร่วมทำทรานแซกชันเท่านั้น โดยไม่ได้สนใจว่าไซต์ต่าง ๆ เหล่านั้นสามารถ คอมมิต หรือ abort ตามสัญญาณที่ส่งไป ได้หรือไม่ ดังนั้นในการดำเนินการโปรโตคอล 2PC จะมีการส่งสัญญาณ acknowledge T กลับมาที่ตัวประสานงานทรานแซกชันเพื่อเป็นการบอกว่าได้ดำเนินการ ในระยะที่สองเสร็จเรียบร้อยแล้ว เมื่อตัวประสานงานทรานแซกชันได้รับ acknowledge T ครบจากทุก ๆ ไซต์ ก็จะเพิ่มเรคอร์ด <complete T> เข้าไปใน log ไฟล์

2.12.4 Handling of Failures

ความล้มเหลวในการทำทรานแซกชันมีอยู่หลายกรณี ซึ่งจะมีการจัดการกับความล้มเหลวต่าง ๆ เหล่านั้นดังนี้

2.12.4.1 Failure of participating site

ถ้าตัวประสานงานทรานแซกชันตรวจพบว่า มีไซต์หนึ่งเกิดล้มเหลวขึ้นมา ก็ต้องพิจารณาว่า ไซต์นั้นเกิดความล้มเหลาก่อนการส่งสัญญาณ ready T หรือไม่ ถ้าเกิดความล้มเหลวก่อน ตัวประสานงานทรานแซกชันจะถือว่าไซต์นั้นส่งสัญญาณ abort T กลับมาให้ แต่ถ้าไซต์นั้นล้มเหลวหลังจากที่ได้ส่งสัญญาณ การดำเนินการโปรโตคอล 2PC ก็จะดำเนินการไปตามปกติโดยไม่สนใจไซต์ที่ล้มเหลว เมื่อไซต์ที่ร่วมทำทรานแซกชัน S_k ทำการฟื้นคืนสภาพข้อมูลจากความล้มเหลวของไซต์จะต้องกมีการตรวจสอบ log ไฟล์เพื่อกำหนดว่าจะต้องทำอย่างไรกับทรานแซกชันที่ยังทำไม่เสร็จกำหนดให้ T เป็นทรานแซกชันที่ยังทำไม่เสร็จ เราจะต้องพิจารณากรณีต่าง ๆ ต่อไปนี้

ถ้าใน log ไฟล์ มีเรคอร์ด <commit T> ให้ทำการ redo(T)

ถ้าใน log ไฟล์ มีเรคอร์ด <abort T> ให้ทำการ undo(T)

ถ้าใน log ไฟล์ มีเรคอร์ด <ready T> ในกรณีนี้ต้องติดต่อกับ Ci ว่าได้ส่งสัญญาณอะไรมาให้ ถ้าพบว่าทรานแซกชันได้ทำการ คอมมิต ทรานแซกชัน T ก็จะทำการ redo(T) และจะทำ undo(T) ถ้าพบว่าได้ทำการ abort ทรานแซกชัน แต่ถ้าไม่สามารถติดต่อกับ C_i ได้ ไซต์ S_k ก็จะต้องพยายามตรวจสอบว่าทรานแซกชันได้ทำ คอมมิต หรือ abort จากไซต์อื่น ๆ ทุกไซต์ โดยการส่งสัญญาณ query-status T ไปให้กับทุก ๆ ไซต์ เมื่อไซต์ได้รับสัญญาณ query-status T ก็จะทำการตรวจสอบกับ log ไฟล์ว่าได้ทำ คอมมิต หรือ abort กับ ทรานแซกชัน T

log ไฟล์ ไม่ปรากฏเรคอร์ด(abort, commit, ready) อะไรเลยที่เกี่ยวข้องกับทรานแซกชัน T หมายความว่าไซต์ได้เกิดความล้มเหลวก่อนที่จะส่งสัญญาณ ready T กลับไปให้ตัวประสานงานทรานแซกชัน ทำให้ทรานแซกชันต้อง abort ดังนั้น S_k ต้องทำ undo(T)

2.12.4.2 Failure of coordinator

ถ้าตัวประสานงานทรานแซกชันล้มเหลวในระหว่างที่กำลังทำทรานแซกชันไซต์ที่ร่วมกันทำทรานแซกชัน ต้องคอยให้ตัวประสานงานทรานแซกชันกลับมาเริ่มต้นทำงานใหม่แล้วจึงค่อยดำเนินการต่อ

ถ้าใน log ไฟล์มีเรคอร์ด <commit T> ดังนั้นทรานแซกชัน ต้องทำการ คอมมิต

ถ้าใน log ไฟล์มีเรคอร์ด <abort T> ดังนั้นทรานแซกชัน ต้องถูก abort

ถ้าในบางไซต์ไม่มีเรคอร์ด <ready T> ใน log ไฟล์ ดังนั้นจะไม่สามารถทำการคอมมิต ได้ เนื่องจากไม่ได้ส่งสัญญาณ ready T กลับไปให้ตัวประสานงานทรานแซกชัน ดังนั้นตัวประสานงานทรานแซกชันก็จะต้องทำการ abort ทรานแซกชัน อยู่แล้ว ซึ่งในกรณีนี้ไม่จำเป็นต้องรอให้ C_i กลับมาเริ่มต้นทำงานใหม่อีกครั้ง สามารถที่จะ abort ได้ทันที

ถ้าในทุก ๆ ไซต์มีเรคอร์ด <ready T> แต่ไม่มีอะไรต่อจากนั้นเช่น abort หรือ คอมมิต ทำให้ไม่สามารถตัดสินใจได้ว่าจะต้องทำอะไร ดังนั้นไซต์ต่างๆ ต้องคอย C_i กลับให้มาเริ่มต้น

ทำงานใหม่ แล้วค่อยดำเนินทรานแซกชันต่อไป และในขณะที่คอยก็ต้องใช้ทรัพยากรของระบบด้วย

บทที่ 3

ความสามารถของดีบีเอ็มเอส กลไกมัลติเวอร์ชัน (ออราเคิล11จี)

3.1 ออราเคิล ทรานแซกชัน

ในออราเคิล นั้นจะเริ่มต้นทรานแซกชันเมื่อเริ่มประมวลผลคำสั่ง DDL, DML โดยอัตโนมัติและจะทำการเขียนค่าเก่าของข้อมูลที่จะเปลี่ยนแปลงลงใน Undo เทเบิลสเปรซ เพื่อใช้ในการ โรลแบ็ก แต่ละทรานแซกชันจะสิ้นสุดก็ต่อเมื่อ

- ได้รับคำสั่ง คอมมิต หรือ โรลแบ็ก
- ถ้ายูสเซอร์ใช้คำสั่งประเภท DDL เช่น CREATE, RENAME,ALTER โดยที่ภายในทรานแซกชันนั้นมีคำสั่งประเภท DML อยู่ด้วยก็จะทำการ คอมมิต ทรานแซกชันก่อนแล้วค่อย คอมมิต DDL อีกครั้ง
- ยูสเซอร์ตัดการเชื่อมต่อกับฐานข้อมูลทรานแซกชันปัจจุบันจะ คอมมิต
- ยูสเซอร์โพรเซสหยุดทำงานแบบผิดปกติทรานแซกชันจะ โรลแบ็ค

3.1.1 คอมมิต ทรานแซกชัน

คอมมิต คือการยืนยันการเปลี่ยนแปลงที่เกิดขึ้นก่อนที่จะ คอมมิต มีขึ้นตอนการทำงานดังนี้

- ออราเคิล จะรับประกันข้อมูลเก่าเผื่อทรานแซกชันเกิดความผิดพลาดทำให้เกิดโรลแบ็ก โดยจะเขียนข้อมูลเก่าลงใน Undo เทเบิลสเปรซ (BIJ) โดยใช้ unique system change number (SCN)
- ออราเคิล จะรับประกันข้อมูลใหม่โดยจะใช้ Redo log buffer ที่อยู่ใน SGA เก็บข้อมูลใหม่ที่มีการเปลี่ยนแปลงและจะเขียนข้อมูล unique system change number (SCN) ใน redo log buffer ลงใน redo ล็อกไฟล์ ก่อนทรานแซกชัน คอมมิต โดยจะมี log writer process (LGWR) เป็นตัวจัดการ
- ออราเคิล ดาต้าเบส จะทำการปลด lock ต่างๆที่ใช้ในทรานแซกชันนั้น
- การเปลี่ยนแปลงจะถูกทำภายใน ดาต้าเบส Buffer ของ SGA ซึ่งการเปลี่ยนแปลงอาจถูก output ลง Disk ก่อนหรือหลังทรานแซกชัน คอมมิต ก็ได้

3.1.2 โรลแบ็ก ทรานแซกชัน

การ โรลแบ็ก ของ ออราเคิล มีขั้นตอนการทำงานดังนี้

- ออราเคิล ดาต้าเบส จะทำการยกเลิกการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นภายในทรานแซกชัน โดยการไปอ่านค่าเก่ามาจาก Undo เทเบิลสเปรซ

- ออราเคิล ดาต้าเบส จะทำการปลด lock ต่างๆที่ใช้ในทรานแซกชันนั้น การที่จะทำทรานแซกชันแบบที่ละเมิดกฎ Constraint ได้นั้นตอนสร้างตารางเราเปิดคุณสมบัติของ DEFERRABLE

ในการประกาศ Constraint ก่อนซึ่งการเปิดคุณสมบัติ DEFERRABLE นั้นจะทำให้เราสามารถที่จะละเมิด Constraint ของทรานแซกชันได้เช่น

```
ALTER TABLE "STORE"."CAR" ADD(CONSTRAINT "CAR_EMP_FK" FOREIGN KEY("EMPLOYEE_ID")
REFERENCES "STORE"."EMPLOYEES"("EMPLOYEE_ID") DEFERRABLE validate);
```

รูปที่ 3.1 ตัวอย่างคำสั่งประกาศ Foreign key Constraint แบบ DEFERRABLE

```
SET CONSTRAINT CAR_EMP_FK DEFERRED;
BEGIN
 INSERT
  INTO store.car
  VALUES('KOT908',   '6',   'D-Max',   'Pickup',   'Isuzu');
  INSERT
  INTO store.employees
  VALUES('6',   '1',   'David',   'Wloe',   'Salesperson',   '500000');
  COMMIT WORK;
END;
```

รูปที่ 3.2 ตัวอย่างการใช้ DEFERRED

และถ้าเราเปิดคุณสมบัติ INITIALLY DEFERRED ในการประกาศ Constraint เราไม่จำเป็นต้องพิมพ์คำสั่ง SET CONSTRAINT เพราะการเปิดคุณสมบัติ INITIALLY DEFERRED เป็นการกำหนดให้ ทรานแซกชันที่กระทำกับ Constraint นั้นๆสามารถละเมิดกฎได้โดย Default กรณีที่เกิด ความผิดพลาด ขึ้นภายในทรานแซกชันก็จะ โรลแบ็ก กลับไปก่อนที่จะมีการเปลี่ยนแปลงข้อมูล

3.1.3 SAVEPOINT ในทรานแซกชัน

SAVEPOINT คือจุดที่บอกให้มีการจดจำ การ คอมมิต ไว้ในกรณีที่ทรานแซกชันนั้นมีหลายคำสั่ง ในกรณีที่เมื่อเกิดข้อผิดพลาดเราไม่ต้องการให้ โรลแบ็ก ทั้งหมด เมื่อเกิดข้อผิดพลาดก็จะ โรลแบ็ก ไปยัง SAVEPOINT ล่าสุด ซึ่งการใช้ SAVEPOINT นี้จะเรียกว่า ทรานแซกชันภายใน (Nested Transaction)

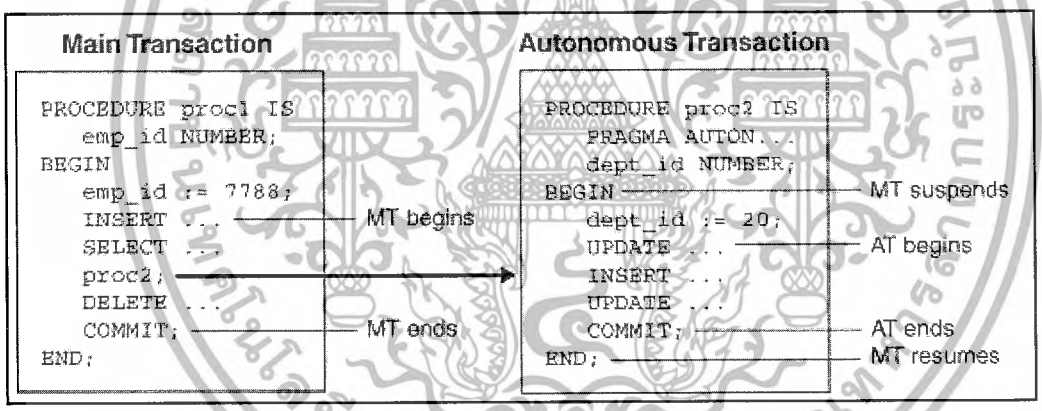
ถ้าหากเรามีการทำเครื่องหมายทั้งหมด 5 SAVEPOINT ต่อมามีการ Roll back ที่ 3 ของ SAVEPOINT จะทำให้ SAVEPOINT ตำแหน่งที่ 4 และ 5 ถูกลบออกไปคำสั่ง คอมมิต และ โรลแบ็ก จะถูกลบออกทุก SAVEPOINT โดยปกติหมายเลขของการทำงานของ SAVEPOINT ในแต่

ละผู้ใช้ทำมีได้เพียง 5 SAVEPOINT เท่านั้นซึ่งการทำงานหนึ่งของ SAVEPOINT ต้องมีการทำเครื่องหมายหลังสุดท้ายของการ คอมมิต และ โรลแบ็ก และ DBA สามารถตั้งจำกัดค่า โดยการไปเพิ่มค่าในตัวแปรที่ชื่อ SAVEPOINTS หากมี SAVEPOINT 2 อันที่มีชื่อเหมือนกัน จะทำให้เกิดการลบ SAVEPOINT อันใดอันหนึ่งอย่างทันที

ถ้าไม่มีการใช้ SAVEPOINT คำสั่งทั้งหมดในทรานแซกชันจะถูกยกเลิกเสมือนว่าไม่มีการกระทำคำสั่งเกิดขึ้นในกรณีที่เกิดข้อผิดพลาดขึ้นภายในทรานแซกชัน

3.1.4 ทรานแซกชันอิสระ(Autonomous Transaction)

คือ ทรานแซกชันภายใน อีกแบบหนึ่งที่ไม่ขึ้นกับทรานแซกชันหลัก ซึ่งไม่มีการแชร์ล็อกหรือทรัพยากรใดๆ และถ้าทรานแซกชันอิสระถูก คอมมิต แล้ว สามารถมองเห็นการเปลี่ยนแปลงได้จากทรานแซกชันหลักหรือทรานแซกชันอิสระ ถัดไป (กรณีที่ในทรานแซกชันหลัก มีหลายทรานแซกชันอิสระ แต่ต้องตั้งค่าไอโซเลชัน เลเวลเป็น READ COMMITTED แต่ถ้า ไอโซเลชัน เลเวลเป็น SERIALIZABLE การเปลี่ยนแปลงในทรานแซกชันอิสระจะไม่ถูกมองเห็นจนกว่าทรานแซกชันหลักจะ คอมมิต



รูปที่ 3.3 ตัวอย่าง ทรานแซกชันอิสระ

3.1.4.1 การประกาศ ทรานแซกชันอิสระ สามารถประกาศได้โดยใช้คำสั่งของ PL/SQL PRAGMA AUTONOMOUS_TRANSACTION ได้ 4 แบบ

- ประกาศที่ Anonymous PL/SQL block
- ประกาศที่ subprogram หรือ function ย่อย
- ประกาศที่ method ของ SQL object type
- ประกาศที่ ดาต้าเบส Trigger

ตัวอย่าง การประกาศ ทรานแซกชันอิสระ แบบ Standalone Procedure

Autonomous Transaction

```
CREATE PROCEDURE lower_salary (emp_id NUMBER, amount NUMBER) AS PRAGMA
AUTONOMOUS_TRANSACTION;
BEGIN
 UPDATE employees SET salary =
  salary - amount WHERE employee_id = emp_id;
 COMMIT;
END lower_salary;
```

3.1.4.2 การทำงานของทรานแซกชันอิสระ จะเริ่มทำงานเมื่อเราสั่งให้ทำงาน การทำงานของทรานแซกชันหลักจะหยุดทำงานชั่วคราวและของทรานแซกชันจะเริ่มทำงานต่อเมื่อจบการทำงานโดยการจบการทำงาน จะต้องมีการ คอมมิต หรือ โรลแบ็ก ซึ่งการ คอมมิต หรือ โรลแบ็ก จะเป็นการจบทรานแซกชันอิสระ แต่จะยังไม่จบ การทำงานซะทีเดียว เพราะเมื่อจบทรานแซกชันอิสระ แล้วอาจจะมีคำสั่ง SQL อื่นต่อหรือมีทรานแซกชันอิสระ อื่นต่อใน การทำงานเดียวกัน เซฟพอย(SAVEPOINT) ในทรานแซกชันอิสระสามารถมีชื่อเดียวกันกับ เซฟพอยในทรานแซกชันหลักได้ แต่เราไม่สามารถ โรลแบ็ก จากทรานแซกชันอิสระไปยัง SAVEPOINT ของทรานแซกชันหลักได้

3.1.5 เปรียบเทียบทรานแซกชันอิสระ กับทรานแซกชันภายใน

เปรียบเทียบทรานแซกชันอิสระ กับทรานแซกชันภายใน (Comparison of Autonomous Transaction and Nested Transaction)

- ทรานแซกชันอิสระ จะเริ่มทำงานโดยการถูกเรียกจากทรานแซกชันอื่น ไม่ใช่ทรานแซกชันภายใน
- ทรานแซกชันอิสระ จะไม่แชร์ทรัพยากร (resources) ในการทำทรานแซกชัน เช่น การ ล็อก(LOCK) กับทรานแซกชันหลัก
- ทรานแซกชันอิสระ จะไม่ขึ้นกับทรานแซกชันหลัก เช่น ถ้า ทรานแซกชันหลัก โรลแบ็ก ทรานแซกชันภายใน จะ โรลแบ็ก แต่ทรานแซกชันอิสระ จะไม่ โรลแบ็ก
- เมื่อเกิดการ คอมมิต แล้วผลของการเปลี่ยนแปลงจะถูกมองเห็นโดย ทรานแซกชันอื่นได้ (ทรานแซกชันภายในเมื่อเกิดการ คอมมิต แล้วจะไม่สามารถมองเห็นผลการเปลี่ยนแปลงจากทรานแซกชันอื่นได้จนกว่าทรานแซกชันหลักจะ คอมมิต)

3.2 ทดสอบคุณสมบัติ ACID ของทรานแซกชัน

3.2.1 Atomicity

การทำ ทรานแซกชัน แบบทั่วไปที่ไม่มีการทำทรานแซกชันภายใน (Nested transaction) จะสอดคล้องกับคุณสมบัติของ Atomicity เพราะเมื่อเกิดการ คอมมิต ก็จะเสมือนทำเสร็จหมดและ ถ้า โรลแบ็ก ก็จะเสมือนไม่ได้มีการทำงานใดๆเกิดขึ้นรวมถึงการใช้คำสั่ง SAVEPOINT (Nested transaction) ด้วยเพราะถือว่าเป็นการทำทรานแซกชัน ย่อยๆ แต่ละทรานแซกชันย่อยก็มีคุณสมบัติ Atomicity ด้วย

```
BEGIN
  INSERT
  INTO store.employees
  VALUES('6' ,    '1',    'David',    'Wloe',    'Salesperson',    '500000');
  INSERT
  INTO store.car
  VALUES('[illegible]',    '6',    'D-Max',    'Pickup',    'Isuzu');
  INSERT
  INTO store.employees
  VALUES('7',    '1',    'Vann',    'Cole',    'Salesperson',    '500000');
  INSERT
  INTO store.employees
  VALUES('8',    '1',    'David',    'Ziman',    'Salesperson',    '500000');
  SAVEPOINT [illegible];
  INSERT
  INTO store.employees
  VALUES('9' ,    '1',    'Mark',    'Rodney' ,    'Salesperson',    '500000');
  INSERT
  INTO store.employees
  VALUES('10',    '1',    'Susan',    'Parker',    'Salesperson',    '500000');
  INSERT
  INTO store.employees
  VALUES('10',    '1',    'Susan',    'Parker',    'Salesperson',    '500000');
  EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
     ROLLBACK ;
END;
```

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME	TITLE	SALARY
1	(null)	James	Smith	CEO	800000
2	1	Ron	Johnson	Sales Manager	600000
3	2	Fred	Hobbs	Salesperson	150000
4	2	Susan	Jones	Salesperson	500000
6	1	David	Wloe	Salesperson	500000
7	1	Vann	Cole	Salesperson	500000
8	1	David	Ziman	Salesperson	500000

รูปที่ 3.4 ตัวอย่างคำสั่ง และ ตารางผลลัพธ์หลังทำทรานแซกชัน

3.2.2 Consistency

ในการทำทรานแซกชันแต่ละครั้งไม่ว่าจะเป็นทรานแซกชันธรรมดาหรือทรานแซกชันภายใน เมื่อจบทรานแซกชันแล้วจะมีการตรวจสอบความถูกต้องของข้อมูล (Constrain) อีกครั้ง ถ้าไม่ถูกต้องตามที่ได้ประกาศไว้ ออราเคิล ก็จะทำ ทรานแซกชัน โรลแบ็ก และแสดง **ความผิดพลาด**

```
SET CONSTRAINT car_emp_fk DEFERRED;
BEGIN
  INSERT
  INTO store.car
  VALUES('KOT908',   '6',   'D-Max',   'Pickup',   'Isuzu');
  SAVEPOINT insert_car;
  INSERT
  INTO store.employees
  VALUES('6',   '1',   'David',   'wole',   'Salesperson',   '500000');
  INSERT
  INTO store.employees
  VALUES('6',   '1',   'David',   'wole',   'Salesperson',   '500000');

EXCEPTION
WHEN dup_val_on_index THEN
  ROLLBACK TO insert_car;
COMMIT;
END;
```

```
Error report:
ORA-02091: โรลแบ็คการทำรายการแล้ว
ORA-02291: มีการละเมิดข้อกำหนดเพื่อตรวจสอบความถูกต้อง (STORE.CAR_EMP_FK) ไม่พบคีย์หลัก
ORA-06512: ที่ บรรทัด 16
ORA-00001: ละเมิดข้อจำกัดเฉพาะ (STORE.EMPLOYEES_PK)
02091. 00000 - "transaction rolled back"
*Cause:    Also see error 2092. If the transaction is aborted at a remote
           site then you will only see 2091; if aborted at host then you will
           see 2092 and 2091.
*Action:   Add rollback segment and retry the transaction.
```

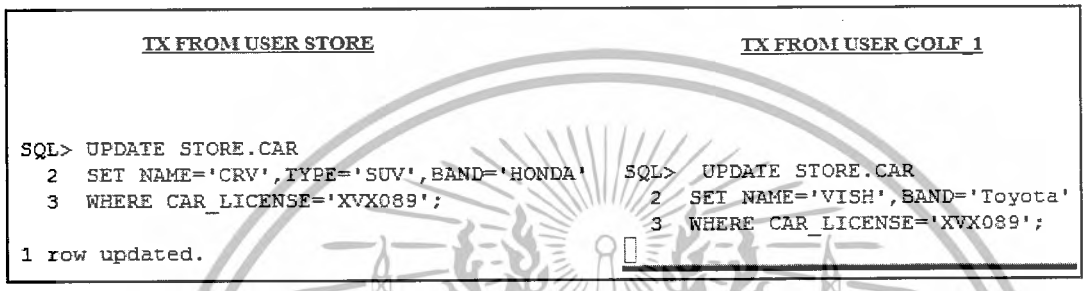
รูปที่ 3.5 ตัวอย่าง หรือ ทรานแซกชัน ภายใน ที่ละเมิด Consistency ไม่ได้

จากตัวอย่างรูปที่ 3.5 เป็นการ INSERT CAR ก่อนค่อย INSERT พนักงานผู้ดูแลตามโดยมี SAVEPOINT ขั้นกลางจากนั้น INSERT ข้อมูลซ้ำเพื่อทำให้เกิด exception แล้ว โรลแบ็ก ไปที่ SAVEPOINT เมื่อจบทรานแซกชันก็เสมือนการ INSERT CAR ที่ไม่มีพนักงานดูแลจึงทำให้ผิดกฎความถูกต้อง (Constrain) ที่ประกาศไว้ทำให้เกิด ความผิดพลาด

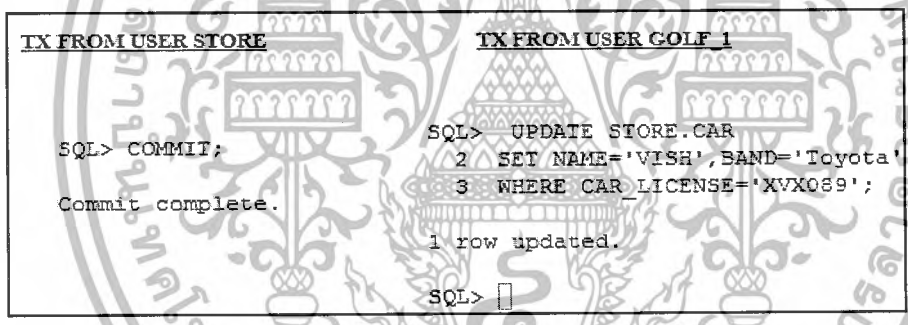
กรณีที่เรามีการ คิวรี่ ยาวๆ หรือข้อมูลที่ คิวรี่ มีจำนวนมากและอาจใช้เวลานาน เช่นการสรุปยอดขายประจำวัน อาจมีการเปลี่ยนแปลงข้อมูลระหว่างที่มีการ คิวรี่ วิธีการคง ความถูกต้องซึ่งใน ออราเคิล จะใช้คำสั่ง SET TRANSACTION READ ONLY; เพื่อป้องกันการเปลี่ยนแปลงของข้อมูล ที่ถูกกระทำโดยทรานแซกชันอื่นระหว่างที่ประมวลผลทรานแซกชัน ปัจจุบันอยู่

3.2.3 Isolation

ในกรณีที่แต่ละทรานแซกชันกระทำกับข้อมูลคนละโรล (อาจอยู่ใน table เดียวกันก็ได้) ไม่ว่าทรานแซกชัน 1 หรือทรานแซกชัน 2 อันไหนทำก่อนก็จะได้ผลลัพธ์ ที่เหมือนกัน แต่ถ้ามีการแก้ไขข้อมูลที่เป็นโรล เดียวกัน ทรานแซกชัน ที่เรียกใช้โรล นั้นทีหลังจะไม่สามารถใช้ได้จนกว่าทรานแซกชันที่เรียกใช้ก่อนจะจบทรานแซกชันเพราะ ใน ออราเคิล จะมีการ ROW-LOCK และใน ออราเคิล ไม่ยอมให้มีการทำ Dirty read



รูปที่ 3.6 ก ตัวอย่าง ทรานแซกชัน 2 ไม่สามารถทำงานต่อได้



รูปที่ 3.6ข ตัวอย่าง เมื่อ ทรานแซกชัน 1 คอมมิต ทรานแซกชัน 2 จึงจะสามารถทำงานต่อได้

3.2.4 Durability

จะทดลองโดยการวน loop insert ข้อมูล 20000 แถวเมื่อ คอมมิต แล้วจะทำการ reset OS ทันที เพื่อทดลองผลคือขณะ boot OS ใหม่จะมีการ recovery journal เมื่อทำการ startup DB ขึ้นมาใหม่ก็พบว่าข้อมูล 20k row ถูก INSERT ลงตารางแล้ว

```
BEGIN

  FOR i IN 1..20000 LOOP
   INSERT INTO temp VALUES(i, 'to be added later');
  END LOOP;
  COMMIT;
END;
```

```
Checking filesystems
/: clean, 5968/512512 files, 92805/512064 blocks
/u02: recovering journal
/u02: clean, 36/1264224 files, 581398/1263102 blocks
/usr: recovering journal
/usr: clean, 134715/767232 files, 701970/767103 blocks
/u01: recovering journal
/u01: Clearing orphaned inode 229406 (uid=501, gid=501, mode=0100640, size=0)
/u01: clean, 44789/1409024 files, 1103681/1407695 blocks
/boot: recovering journal
/boot: clean, 33/255232 files, 13769/255024 blocks
```

รูปที่ 3.7 ตัวอย่างคำสั่งวน INSERT ข้อมูล และ ข้อความขณะ Boot OS ใหม่

3.3 ทดสอบคุณสมบัติ ACID ของทรานแซกชันอิสระ

3.3.1 Atomicity

ทรานแซกชันอิสระ นั้นสอดคล้องกับคุณสมบัติ Atomicity เพราะทรานแซกชันอิสระ ก็คือทรานแซกชันภายใน รูปแบบหนึ่งซึ่งถือว่าเป็นการทำงานแบบทรานแซกชันย่อย การทำงานของทรานแซกชันหลักกับทรานแซกชันอิสระ นั้นไม่ขึ้นต่อกันแม้ว่า ทรานแซกชันหลักจะเกิดการโรลแบ็ก แต่ผลลัพธ์ของทรานแซกชันอิสระ ที่ คอมมิต แล้วจะยังคงอยู่จากการทดลองสร้าง Procedure ที่มีทรานแซกชันอิสระ อยู่ขึ้นมาและใช้ทรานแซกชันหลักเรียกทรานแซกชันอิสระ เมื่อโรลแบ็ก ก็มีผลลัพธ์ของทรานแซกชันอิสระ ที่ คอมมิต แล้วอยู่ในตาราง T

```
CREATE OR REPLACE PROCEDURE insert_into_t AS
pragma autonomous_transaction;
BEGIN
  INSERT  INTO t  VALUES(99);
  COMMIT;
END;
/
```

```
SQL> select * from t;

no rows selected
```

```
BEGIN
  INSERT  INTO t  VALUES(-1);
  insert_into_t;
  ROLLBACK;
END;
/
```

```
SQL> r
  1* select * from t

         X
----------
        99
```

รูปที่ 3.8 ตัวอย่าง Autonomous ทรานแซกชัน

3.3.2 Consistency

ใน ทรานแซกชันอิสระ จะมีการตรวจสอบความถูกต้องข้องข้อมูลในตัวเอง หลังจาก คอมมิต ทรานแซกชันอิสระ แล้วข้อมูลยังละเมิดกฎอยู่จะเกิด ความผิดพลาด ขึ้นและจะ โรลแบ็ก ทรานแซกชันหลักด้วยแต่ถ้าใช้คำสั่ง โรลแบ็ก ทรานแซกชันอิสระ ทรานแซกชันหลักจะไม่ถูก โรลแบ็ก

```
CREATE OR REPLACE PROCEDURE insert_car_only AS
pragma autonomous_transaction;
BEGIN
  INSERT  INTO STORE.CAR
  VALUES('AAA111','7','385i','Zidan','BMW');
  COMMIT;
END;
/

PROCEDURE insert_car_only Compiled.
```

```
BEGIN

  INSERT
  INTO store.car
  VALUES('ZZZ666',  '5',  'Civic',  'Zidan',  'HONDA');

  insert_car_only;

  INSERT
  INTO store.employees
  VALUES('5',  '1',  'Jame',  'Borne',  'Salesperson',  '5555');
  INSERT
  INTO store.employees
  VALUES('7',  '1',  'Michle',  'Jackson',  'Salesperson',  '5555');
  COMMIT;
END;
Error report:
ORA-02091: โรลแบ็คการทำรายการแล้ว
ORA-02291: มีการละเมิดข้อกำหนดเพื่อตรวจสอบความถูกต้อง (STORE.CAR_EMP_FK)  ไม่พบคีย์หลัก
ORA-06512: ที่ "STORE.INSERT_CAR_ONLY",  บรรทัด 6
ORA-06512: ที่  บรรทัด 7
02091. 00000 -  "transaction rolled back"
```

รูปที่ 3.9 ตัวอย่าง ทรานแซกชันอิสระ (Autonomous transaction) ที่เกิด ความผิดพลาด

```
BEGIN
  INSERT
  INTO store.car
  VALUES('ASD',   'TOYOTA',   '7');
  ROLLBACK;
END;
/

BEGIN
  INSERT
  INTO store.car
  VALUES('TTT',   'HONDA',   '9');

  insert_car_only;

  INSERT
  INTO store.employees
  VALUES('9',   '1',   'CRUIES',   'MEDY',   'XXXXX',   '5555');
  INSERT
  INTO store.employees
  VALUES('7',   '1',   'HULK',   'SHINO',   'XXXXX',   '7777');
  COMMIT;
END;
```

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME	TITLE	SALARY
1	(null)	James	Smith	CEO	800000
2	1	Ron	Johnson	Sal...	600000
3	2	Fred	Hobbs	Sal...	150000
4	2	Susan	Jones	Sal...	500000
9	1	CRUIES	MEDY	XXXXX	5555
7	1	HULK	SHINO	XXXXX	7777

รูปที่ 3.10 ตัวอย่าง การ โรลแบ็ก ใน ทรานแซกชันอิสระ (Autonomous transaction) แล้วทรานแซกชันหลักทำงานต่อได้

3.3.3 Isolation

ในกรณีที่แต่ละทรานแซกชันกระทำกับข้อมูลคนละ row (อาจอยู่ใน table เดียวกันก็ได้) ไม่ว่าทรานแซกชัน 1 หรือทรานแซกชัน 2 อันไหนทำก่อนก็จะได้ผลลัพธ์ที่เหมือนกันแต่ถ้าในทรานแซกชันหลักมีการเรียกใช้ข้อมูลที่ถูกทรานแซกชันอิสระ (Autonomous transaction) คอมมิต แล้วได้ (ต้อง SET TRANSACTION ISOLATION LEVEL READ COMMITTED) แต่เราสามารถกำหนดให้ทรานแซกชันหลักให้ไม่สามารถอ่านค่าใหม่ได้ (ต้อง SET TRANSACTION ISOLATION LEVEL SERIALIZABLE)

```
SQL> set transaction isolation level read committed;

Transaction set.

CREATE OR REPLACE PROCEDURE insert_car_only AS
pragma autonomous_transaction;
BEGIN
  INSERT
  INTO store.car
  VALUES('AAAlll',   '7',   '385i',   'Zidan',   'BMW');
  INSERT
  INTO store.employees
  VALUES('7',   '1',   'Michle',   'Jackson',   'Salesperson',   '5555');
  COMMIT;
END;
/
```

```
SQL> r
  1  DECLARE
  2  v_detail  VARCHAR2(10);
  3
  4  BEGIN
  5  insert_car_only;
  6  SELECT CAR_LICENSE
  7  INTO v_detail
  8  FROM car;
  9  DBMS_OUTPUT.PUT_LINE(v_detail);
 10* END;
AAA111

PL/SQL procedure successfully completed.
```

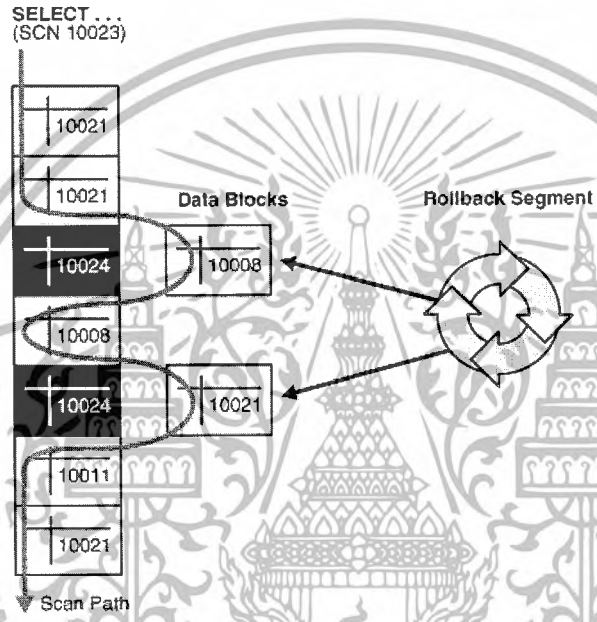
รูปที่ 3.11 ตัวอย่าง ไอโซเลชัน เลเวล READ COMMITTED

```
SQL> set transaction isolation level serializable;

Transaction set.

SQL> DECLARE
  2  v_detail  VARCHAR2(10);
  3  BEGIN
  4  insert_car_only;
  5  SELECT CAR_LICENSE INTO v_detail
  6  FROM car;
  7  DBMS_OUTPUT.PUT_LINE(v_detail);
  8  COMMIT;
  9  END;
 10  /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 5
```

รูปที่ 3.12 ตัวอย่าง ไอโซเลชัน เลเวล SERIALZABLE

3.4 มัลติเวอร์ชัน คอลเคอเรนซี่ คอลโทรล ในผลิตภัณฑ์ออราเคิล

มัลติเวอร์ชัน คอลเคอเรนซี่ คอลโทรล ในผลิตภัณฑ์ออราเคิล (Multiversion Concurrency Control Oracle)จะจัดการบริหารข้อที่ยังไม่มีการ คอมมิต (บีไอเจ) หรือ ข้อมูลที่พึงมีการ คอมมิต ในโรลแบ็กเซ็กเม้น เมื่อมีการประมวลผลข้อมูล ออราเคิล จะมีการใส่ ซีเควนเชงนัมเบอร์ (System change number (SCN)) ตาม ไทม์สแตมป์โปรโตคอล (Time stamp protocol) เมื่อส่งคิวรี่เข้ามาก็ จะอ่านเฉพาะข้อมูลที่ มี SCN น้อยกว่าหรือ เท่ากับ SCN ของ คิวรี่ ดังรูปที่ 38



รูปที่ 3.13 การใช้ซีเควนเชงนัมเบอร์ (SCN) ของ ออราเคิล

Statement-Level Read Consistency Oracle Database ควบคุม Statement-Level Read Consistency เสมอคือการรับรองว่าผลลัพธ์จากการ คิวรี่ จะเป็นข้อมูลขณะที่เริ่มต้น คิวรี่ เพราะฉะนั้น คิวรี่ จะไม่เป็นการเปลี่ยนแปลงใดๆในระหว่างที่กำลังทำทรานแซกชันการประมวลผล คิวรี่ จะประมวลผลข้อมูลที่ถูก คอมมิต ก่อนเริ่ม คิวรี่ เท่านั้น

Transaction-Level Read Consistency Oracle Database จะมี Transaction-Level Read Consistency เป็น Option เพิ่มเติมเมื่อทรานแซกชันทำงานอยู่ใน Serializable mode คือข้อมูลทั้งหมดที่มีการเข้าถึงเมื่อเริ่มทรานแซกชันจะไม่เห็นการเปลี่ยนแปลงใดๆของข้อมูลจบกว่าจะจบทรานแซกชัน

ตารางที่ 3.1 ไอโซเลชัน เลเวล ที่มีในออราเคิล

ไอโซเลชัน เลเวล	Description
Read committed	เป็น default level ของ ออราเคิล แต่ละ คิวรี่ จะเห็นเฉพาะข้อมูลที่มีการ คอมมิต ก่อนหน้า คิวรี่ เท่านั้น (ไม่ใช่ ทรานแซกชัน)
Serializable	ทรานแซกชัน จะมองเห็นเฉพาะข้อมูลที่มีการ คอมมิต ก่อนที่จะเริ่ม ทรานแซกชัน และ การอ่านข้อมูลใหม่ที่พึ่ง insert เข้าไปขณะกำลังทำทรานแซกชัน (Phantom Read)
Read Only	ทรานแซกชัน นั้นจะไม่สามารถ INSERT , DELETE , UPDATE ได้ และจะอ่านได้เฉพาะข้อมูลที่มีการ คอมมิต ก่อน เริ่ม ทรานแซกชัน

ซึ่งสามารถใช้คำสั่งดังต่อไปนี้

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
```

ออราเคิล จะมีการแจ้ง ความผิดพลาด เมื่อมีการพยายามแก้ไขข้อมูลที่ คอมมิต หลังจากที่เริ่มSERIALIZABLE ทรานแซกชันไปแล้ว ดังรูปที่ 3.14 เมื่อเกิด ความผิดพลาด เช่นนี้ขึ้นมีแนววางในการจัดการ ความผิดพลาด ดังนี้

- คอมมิต โดยไม่ทำคำสั่งอัพเดท นั้น
- โรลแบ็ก แล้วทำทรานแซกชันใหม่ต่อจาก SAVEPOINT
- อันดู ทรานแซกชัน

3.4.1 การ LOCK ของ ออราเคิล โดย ค่าเริ่มต้น ของ ออราเคิล นั้นจะเป็นการล็อก ระดับโรลเพื่อรองรับการทำงานพร้อมๆกันได้หลายทรานแซกชันซึ่งจะมีการปลดล็อกก็ต่อเมื่อคอมมิต หรืออันดูทรานแซกชัน เท่านั้น

ตารางที่ 3.2 แสดงการ LOCK ในแบบต่าง

Lock	Description
DML locks (data locks)	คือการล็อก ข้อมูล เช่น ตาราง,โรล
DDL locks (dictionary locks)	คือการ ล็อกพจนานุกรมข้อมูล, ข้อมูลโครงสร้างของตาราง
Internal locks and latches	คือการล็อก โครงสร้างภายในของฐานข้อมูล เช่น Datafile

3.4.1.1 DML Locks

คือการรับรองความถูกต้องของข้อมูลเมื่อมีการเข้าถึงข้อมูลพร้อมๆกันหลาย user Row Locks (TX) เมื่อรวมการทำงานแบบ Row-level locks และกลไกแบบ มัลติเวอร์ชัน เข้าด้วยกันแล้วมีข้อดีดังนี้

- ผู้อ่านข้อมูลไม่ต้องรอผู้เขียนข้อมูล ที่ทำงานกับโรล เดียวกัน
- ผู้เขียนข้อมูลไม่ต้องรอผู้อ่านข้อมูล ที่ทำงานกับ โรล เดียวกัน ยกเว้น SELECT FOR UPDATE
- ผู้เขียนข้อมูล จะรอผู้เขียนข้อมูลอื่น ก็ต่อเมื่อ ต้องการ update row และเวลาเดียวกัน เท่านั้น

3.4.1.2 Table Locks (TM)

จะเป็นการล็อก เพื่อป้องกันคำสั่ง DDL ในขณะที่มีการทำงานคำสั่ง DML อยู่เพื่อไม่ให้มีการเปลี่ยนแปลงโครงสร้างของตารางหรือ view ในขณะที่ทรานแซกชันกำลังทำงาน ซึ่งจะเกิดขึ้นเมื่อมีคำสั่ง INSERT, UPDATE, DELETE, SELECT FOR UPDATE และ LOCK TABLE กับตารางนั้น Table lock นั้นสามารถมีโหมดการล็อก ได้หลาย mode ดังนี้

row share (RS)

row exclusive (RX)

share (S)

share row exclusive (SRX)

exclusive (X)

**Y คือ ทรานแซกชัน อื่นสามารถรอ เพื่อทำการ Locks แบบนั้นๆ ได้*

SQL Statement	Mode of Table Lock	Lock Modes Permitted? RS	RX	S	SRX	X
SELECT...FROM *table*...	none	Y	Y	Y	Y	Y
INSERT INTO *table* ...	RX	Y	Y	N	N	N
UPDATE *table* ...	RX	Y*	Y*	N	N	N
DELETE FROM *table* ...	RX	Y*	Y*	N	N	N
SELECT ... FRDM *table* FOR UPDATE DF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE *table* IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE *table* IN ROW EXCLUSIVE MODE	RX	Y	Y	N	N	N
LOCK TABLE *table* IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE *table* IN EXCLUSIVE MODE	X	N	N	N	N	N

รูปที่ 3.14 แสดง โหมดของ table locks ของแต่ละคำสั่ง

จากรูปที่ 3.14 จะเห็นว่าจะมีการ X-LOCK Table เพียงคำสั่งเดียวเพื่อให้เพื่อให้รองรับการทำงานพร้อมกัน

3.5 ทดสอบ ไอโซเลชัน เลเวล กับปัญหา Concurrency Control 4 ข้อ

ไอโซเลชัน เลเวลที่ เป็นค่าริ่มต้น ของ ออราเคิล คือ READ COMMITTED

3.5.1 Lost update Problem

ถ้าเริ่มต้นซึ่งกรณีแบบนี้จะเป็นการใช้คำสั่ง SELECT ..... FOR UPDATE เพื่อทำการอ่านค่ามาเพื่อ อัพเดท ใน ออราเคิล นั้นจะใช้ Strict 2 - phase locking protocol ในการจัดการโดยจะเอ็กล็อก ระดับของโรล แล้ว อาร์เอส ล็อก ระดับของตารางเมื่อทรานแซกชัน อื่นใช้คำสั่ง SELECT FOR ... UPDATE ก็จะต้องรอให้ทรานแซกชันที่ ล็อก อยู่ คอมมิต หรือ โรลแบ็ก ก่อนจึงจะทำงานได้

ตารางที่ 3.3 ผลการทดลอง Lost update problem

TX A	TX B	XQTY=10
SELECT * FROM store.x WHERE x#=1 FOR UPDATE;		10
	SELECT * FROM store.x WHERE x#=1 FOR UPDATE; << wait >>	
UPDATE store.x SET xqty=xqty+5 WHERE x#=1;		
COMMIT;	<< ready >>	15
	UPDATE store.x SET xqty=xqty-5 WHERE x#=1;	
	COMMIT;	10

3.5.2 Uncommitted Dependency Problem

การจัดการกับ ปัญหา Uncommitted Dependency นั้น ออราเคิล ใช้ ทู-เฟส สติกล๊อกกิ๊ง โปรโตคอลเพื่อ เอ็กล็อก row ที่มีการ อัพเดทและยังไม่ได้ คอมมิต ซึ่งทรานแซกชันอื่นจะไม่สามารถขอ เอ็กล๊อก ได้คือถ้าจะ อัพเดทหรือ SELECT ..... FOR UPDATE จะต้องรอให้ทรานแซกชันที่กำลัง ล็อกอยู่ปลดล็อก ก่อน แต่สามารถ SELECT ได้ โดย ออราเคิล จะไปดึงเอา เวอร์ชัน เก่าของข้อมูลมาให้ (ข้อมูลก่อนที่ จะเริ่มทรานแซกชัน) โดยกลไก มัลติเวอร์ชัน

ตารางที่ 3.4 ตัวอย่างการแก้ปัญหา Uncommitted Dependency ของ ออราเคิล

TX A	TX B	TX C
UPDATE store.x SET xqty=10 WHERE x#=1;		
	SELECT * FROM store.x WHERE x#=1 FOR UPDATE; << wait >>	SELECT * FROM store.x WHERE x#=1; << XQTY = 0 >> ใช้ข้อมูล version เก่า
ROLLBACK;	Display <<< XQTY = 0 >>	
	COMMIT;	COMMIT;

จากตารางที่ 3.4 จะเห็นว่า ถ้าใช้คำสั่ง SELECT.. FOR UPDATE ไปใช้นั้นจะเกิดการรอ ซึ่งจะช่วยให้ได้ข้อมูลที่ถูกต้อง แต่ถ้าใช้คำสั่ง SELECT ก็จะ ได้ข้อมูลเวอร์ชัน เก่าซึ่งจะไม่ขึ้นกับการ คอมมิต หรือ โรลแบ็ก ของทรานแซกชันที่กำลัง ล็อกโรล อยู่

3.5.3 Inconsistent Analysis Problem

ตารางที่ 3.5 ตัวอย่าง ไอโซเลชัน เลเวล READ COMMITTED แก้ปัญหาข้อ 3, 4 ไม่ได้

Tx A	Tx B
SELECT xqty FROM store.x WHERE x#=1; << XQTY = 0 >>	
	UPDATE store.x SET xqty=10 WHERE x#=1;
	COMMIT;

ตารางที่ 3.5 ตัวอย่าง ไอโซเลชัน เลเวล READ COMMITTED แก้ปัญหาข้อ 3, 4 ไม่ได้ (ต่อ)

Tx A	Tx B
SELECT xqty FROM store.x WHERE x#=1; << XQTY = 10 >>	

3.5.4 Phantom Phenomenon

คล้ายกับข้อ 3 แต่ เปลี่ยนจาก UPDATE เป็น INSERTใน ไอโซเลชัน เลเวล READ COMMITTED นั้นไม่สามารถจัดการปัญหา ข้อ 3 และ 4 ได้เพราะ READ COMMITTED นั้นยอมให้อ่านข้อมูลที่ทีการ คอมมิต จากทรานแซกชันอื่นในระหว่างที่ทรานแซกชันกำลังทำงาน ดังนั้นจึงต้องให้ ไอโซเลชัน เลเวล SERIALIZABLE เพื่อแก้ปัญหาข้อ 3, 4 ซึ่ง ออราเคิล จะใช้กลไก มัลติเวอร์ชัน มาช่วยแก้ปัญหา เมื่อเริ่มทรานแซกชันคำสั่งภายในทรานแซกชันนั้นก็จะอ่านข้อมูลเวอร์ชัน ที่มี SCN น้อยกว่า SCN ตอนเริ่ม ทรานแซกชันเท่านั้น ซึ่งเมื่อมีทรานแซกชันอื่นแก้ไขข้อมูล และ คอมมิต ภายหลัง row นั้นจะมี SCN ที่มากกว่า เมื่อทำการเปลี่ยน ไอโซเลชัน เลเวล แล้วทำการทดลองคำสั่งเดิมซ้ำได้ผลดังนี้

ตารางที่ 3.6 ตัวอย่าง การแก้ปัญหา ข้อ 3, 4 ของ ออราเคิล

Tx A	Tx B
SELECT xqty FROM store.x WHERE x#=1; << XQTY = 0 >>	
	UPDATE store.x SET xqty=10 WHERE x#=1;
	COMMIT;
SELECT xqty FROM store.x WHERE x#=1; << XQTY = 0 >>	

ตารางที่ 3.6ตัวอย่าง การแก้ปัญหา ข้อ 3, 4 ของ ออราเคิล (ต่อ)

Tx A	Tx B
	SELECT xqty FROM store.x WHERE x#=1; << XQTY = 10 >>
	INSERT INTO store.x VALUES ('7','20');
	COMMIT;
SELECT xqty FROM store.x WHERE x#=7; <<no rows selected >>	
COMMIT;	
SELECT xqty FROM store.x WHERE x#=7; <<XQTY=20>>	

สรุป ออราเคิล จะใช้ ทู-เฟส สติก ล็อกกิ้ง โปรโตคอล ในการแก้ปัญหา ข้อ 1 และ 2 ใช้ กลไก มัลติเวอร์ชัน ในการแก้ปัญหาข้อ 3 และ 4 ในการเรียกคำสั่ง SELECT นั้น จะไม่เกิดการล็อก ขึ้นและจะไม่มีสถานะคอยเกิดขึ้นเพราะสามารถอ่านข้อมูลจากเวอร์ชัน เก่าได้ แต่จะเกิดเอ็กล็อก ระดับของโรล เมื่อมีคำสั่ง INSERT, DELETE, UPDATE, SELECT.. FOR UPDATE ISOLATION LEVEL READ COMMITTED แก้ปัญหาข้อ 1, 2 ได้ แต่ไม่สามารถแก้ปัญหาข้อ 3, 4 ได้แต่ ไอโซเลชัน เลเวล SERIALIZABLE นั้นแก้ปัญหาได้ทุกข้อ

3.6 การตรวจสอบประสิทธิภาพคำสั่ง เอสคิวแอล

ใน ออราเคิล เวลารันคำสั่งเอสคิวแอล ต่าง ๆ เราสามารถทำการตรวจสอบได้ว่าคำสั่ง เอสคิวแอล ที่เราใช้นั้น มีประสิทธิภาพแค่ไหน โดยใช้คำสั่งเอสคิวแอล ในรูปแบบ EXPLAIN PLAN FOR <คำสั่ง เอสคิวแอล > โดย <คำสั่ง เอสคิวแอล > คือ คำสั่ง เอสคิวแอล ที่เราต้องการดูประสิทธิภาพ แต่ก่อนที่จะสามารถใช้คำสั่งข้างบนได้ เราจะต้องสร้างตารางสำหรับเก็บผลลัพธ์

ด้านบนก่อน โดยในโฟลเดอร์<ORACLE_HOME>\rdbms\admin จะมีไฟล์ชื่อว่า “utlxplan.sql” ซึ่งเก็บคำสั่ง เอสคิวแอล สำหรับสร้างตารางที่ชื่อว่า “PLAN_TABLE” เอาไว้ ให้เรารันคำสั่ง เอสคิวแอล นั้นเพื่อสร้างตาราง PLAN_TABLE ใน ดาต้าเบส ที่เราต้องการใช้กับคำสั่งเอสคิวแอล ที่เราจะดูประสิทธิภาพ จากนั้นเมื่อสร้างตารางแล้ว เวลารันคำสั่ง EXPLAIN PLAN ด้านบน จะไม่ได้ผลลัพธ์อะไร (เพราะผลลัพธ์จะถูกเก็บไว้ในตารางPLAN_TABLE) ซึ่งเวลาเราจะดูผลลัพธ์ประสิทธิภาพ ให้ใช้คำสั่งเอสคิวแอลที่อยู่ในไฟล์<ORACLE_HOME>\ora92\rdbms\admin\utlxplp.sql หรือ utlxpls.sql ในการดู หรืออาจจะใช้ SELECT * FROM PLAN_TABLE ORDER BY TIMESTAMP DESC สำหรับดูเลยก็ได้ โดยดูเฉพาะคำสั่งบน ๆ (เพราะทุกครั้งที่รัน EXPLAIN FOR จะเป็นการเพิ่มข้อมูลลงตารางนี้ ทำให้มีข้อมูลของคำสั่ง เอสคิวแอล เก่าๆ อยู่ด้วย) ในกรณีที่ต้องการเก็บผลลัพธ์ไว้ในตารางอื่นที่ไม่ใช่ PLAN_TABLE ก็สามารถทำได้ โดยสร้างตาราง PLAN_TABLE ในชื่ออื่นเอาไว้ แล้วเวลาสั่ง EXPLAIN PLAN FOR ให้ใช้คำสั่งเป็น EXPLAIN PLAN INTO <ชื่อตาราง> FOR <คำสั่ง เอสคิวแอล >

ตัวอย่างเช่น

```
SQL> CONN sys/password AS SYSDBA
Connected
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql
SQL> GRANT ALL ON sys.plan_table TO public;
SQL> CREATE PUBLIC SYNONYM plan_table FOR sys.plan_table;
SQL> EXPLAIN PLAN FOR
  2  SELECT *
  3  FROM   emp e, dept d
  4  WHERE  e.deptno = d.deptno
  5  AND    e.ename  = 'SMITH';
Explained.
```

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxpls.sql

Plan Table
--------------------------------------------------------------------------------
| Operation                 | Name     | Rows | Bytes | Cost | Pstart| Pstop |
--------------------------------------------------------------------------------
| SELECT STATEMENT          |          |      |       |      |       |       |
|  NESTED LOOPS             |          |      |       |      |       |       |
|   TABLE ACCESS FULL       |EMP       |      |       |      |       |       |
|   TABLE ACCESS BY INDEX RO|DEPT      |      |       |      |       |       |
|    INDEX UNIQUE SCAN      |PK_DEPT   |      |       |      |       |       |
--------------------------------------------------------------------------------

8 rows selected.
```

รูปที่ 3.15 แสดงการ EXPLAIN

OPTIMIZER_MODE Initialization Parameter เป็นตัวบอกว่าจะให้ อ๊อปติไมเซอร์ หาเส้นทางที่ดีที่สุดสำหรับผลลัพธ์แบบใด โดยมีค่าดังนี้

- ALL_ROWS คือหาเส้นทางที่ดีที่สุดสำหรับผลลัพธ์ทั้งหมด โดยการใช้สถิติมาคำนวณ
- FIRST_ROWS_n คือ การหาเส้นทางที่ดีที่สุดสำหรับ n row แรก ซึ่ง n คือ 1,10,100,1000 โดย

การใช้สถิติมาคำนวณ FIRST_ROWS คือการใช้สิ่งที่เกี่ยวข้องทั้งหมดในการหาเส้นทางมาคำนวนโดยใช้ Heuristics สามารถเปลี่ยนค่าได้โดยใช้คำสั่ง ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_1; ถ้าในตารางใดไม่มีการเก็บสถิติ (INDEX) อ๊อบติไมเซอร์ ก็จะใช้ข้อมูลภายในระบบเช่น จำนวนการจอง Data block ของตารางนั้นๆ เพื่อเป็นค่าประมาณที่นำมาใช้เป็นสถิติ

3.7 เส้นทางการเข้าถึงข้อมูล (Access Paths for the Query Optimizer)

คือวิธีการต่างๆในการค้นหาขอมูลในฐานข้อมูล ซึ่งมีวิธีการต่างๆ ดังนี้

3.7.1 Full Table Scan

คือการอ่าน โรลทั้งหมดใน ตารางแล้วนำข้อมูลในแต่ละโรลมากรองให้ได้โรลที่ตรง ตามเงื่อนไขเมื่อมีการ สแกนทั้งตราง ออราเคิล จะทำการ จะทำการอ่านแต่ละบล๊อกมาเป็นลำดับ ซึ่งสามารถกำหนดได้ว่าจะให้อ่านทีละกี่บล๊อกโดยกำหนดที่ DB_FILE_MULTIBLOCK_READ_COUNT ซึ่งการสแกนทั้งตารางจะเกิดขึ้นในกรณีต่างๆ ดังต่อไปนี้

- ไม่มี index

- เมื่ออ๊อปติไมเซอร์คิดว่า คิวรี่ นี้ เข้าถึงข้อมูลขนาดใหญ่ใน ตารางก็จะใช้การ การสแกนข้อมูลทั้งตารางราง
- เมื่อตารางมีขนาดเล็กกว่าจำนวนบล๊อก DB_FILE_MULTIBLOCK_READ_COUNT

3.7.2 RowID Scan

RowID จะเป็นตัวระบุตำแหน่งที่อยู่ของโรล ในบล๊อกข้อมูลซึ่งจะเร็วมากในการเข้าถึงข้อมูลโรลเดียว เพราะรู้ได้ทันทีว่า โรลนั้นถูกเก็บอยู่ที่ไหนในทางกายภาพ

3.7.3 Index Scan

คือการ สแกน Index ของ คอลั่ม ต่างๆ ที่เกี่ยงข้องกับ คิวรี่

3.7.3.1 Index Unique Scans

คือการ สแกน Index ที่แน่ในว่ามีผลลัพธ์แค่ 1 โรลซึ่งจะใจคับการสแกนคอลั่ม ที่มี Primary key, Unique Constraint ซึ่ง ออราเคิลจะเก็บ Index แบบ B-Tree

3.7.3.2 Index Range Scans

คือกรณีทั่วไปที่เกิดขึ้นในการค้นหาข้อมูล โดยที่ผลลัพธ์อาจอยู่ จะถูกเรียงน้อยไปมาก ซึ่งอ๊อปติไมเซอร์จะใช้ Index Rang Scan เมื่อเห็นว่าเงื่อนไขนั้นให้ผลลัพธ์ หลาย Row เช่น

คอลั่ม = '10'
คอลั่ม > 10
คอลั่ม like 'ABC%'

3.7.3.3 Index Range Scans Descending

คล้ายกับ Index Range Scan แต่จะมีการเรียงผลลัพธ์จากมากไปหาน้อยโดยทั่วไปแล้วการ เรียงลำดับข้อมูลจะเรียงจากน้อยไปมาก ซึ่งจะใช้ Index Range Scan แต่ Index Range Scans Descending จะใช้ในกรณีที่ต้องการผลลัพธ์ แบบมากไปน้อย

3.7.3.4 Index Skip Scans

คือการใช้ Subindex ของ คอลั่ม นั้นๆ ซึ่งเป็นการเพิ่มประสิทธิภาพในการสแกน เช่นมีตาราง emp(sex,id) ซึ่งมี index (sex,emp_id) ซึ่งจะมี Logical subindex คือ M,F โดยในตารางมีข้อมูลตัวอย่างดังนี้

('F',98)

('F',100)

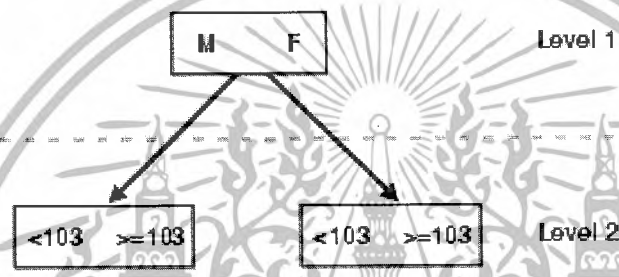
('F',102)

('F',104)

('M',101)

('M',103)

('M',105)



รูปที่ 3.16 Logical Subindex

*SELECT **

FROM employees

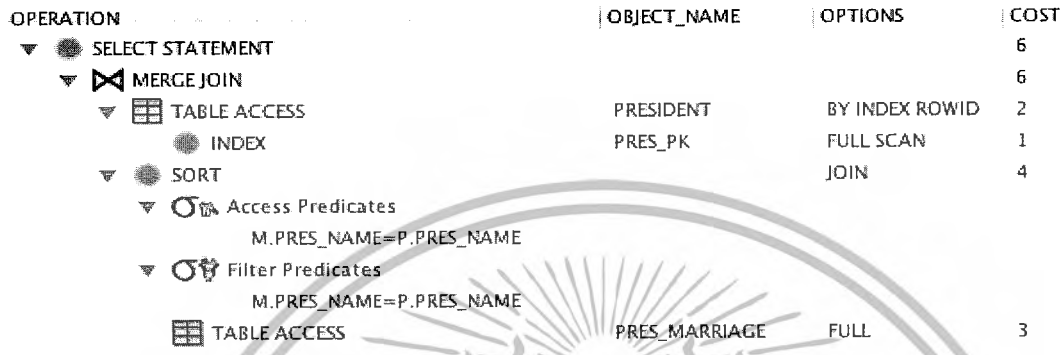
WHERE employee_id = 101;

เมื่อได้รับ คิวรี่ นี้อ็อปติไมเซอร์จะไม้ Scan emp_id ก่อน แต่จะ Scan ค่าในฝั่งของ M ก่อนแล้วค่อยตามด้วย F

3.8 การทดลอง คิวรีออปติไมเซอร์

ผลกระทบจากการ Projection การเลือก ผลลัพธ์ของคอลั่ม นั้นมีผลกระทบกับประสิทธิภาพในการ คิวรี่พอสมควรเพราะ ถ้าเราจอยน์โดยใช้คอลั่มที่มี unique index อยู่แล้วนั้น ออปติไมเซอร์ไม่จำเป็นต้อง Access ตาราง แต่ถ้ามีการ select คอลั่ม ที่ไม่มี unique index นั้น ออปติไมเซอร์ต้องไปสแกน ตารางเพื่อ Access ตารางเพื่อหาคำตอบทำให้เกิด cost เพิ่มขึ้น

```
SELECT p.pres_name, p.state_born, m.nr_children
FROM president p, pres_marriage m
WHERE m.pres_name = p.pres_name;
```



รูปที่ 3.17 การ SELECT คอลั่ม ที่ไม่มี Unique Index

จากรูปที่ 3.17 จะเห็นว่ามี คอลั่ม p.state_born ที่ไม่มี index ทำให้เกิด TABLE ACCESS ที่ PRESIDENT ขึ้นและมี COST เป็น 6

```
SELECT p.pres_name, m.nr_children
FROM president p, pres_marriage m
WHERE m.pres_name = p.pres_name;
```

รูปที่ 3.18แสดงการ SELECT คอลั่ม ที่มี Unique Index

จากรูปที่ 3.18 จะเห็นว่า เมื่อลบ คอลั่ม p.state_born ออก ทำให้ลด TABLE ACCESS ที่ PRESIDENT ลงได้โดยจะเป็น INDEX SCAN เท่านั้น ดังนั้นจึงช่วยลด COST ลงได้ และเนื่องจากการจอยน์ นี้เป็นการนำข้อมูลส่วนใหญ่ของตารางมาทำการจอยน์ออปติไมเซอร์ จึงใช้ nested loop ในการจอยน์

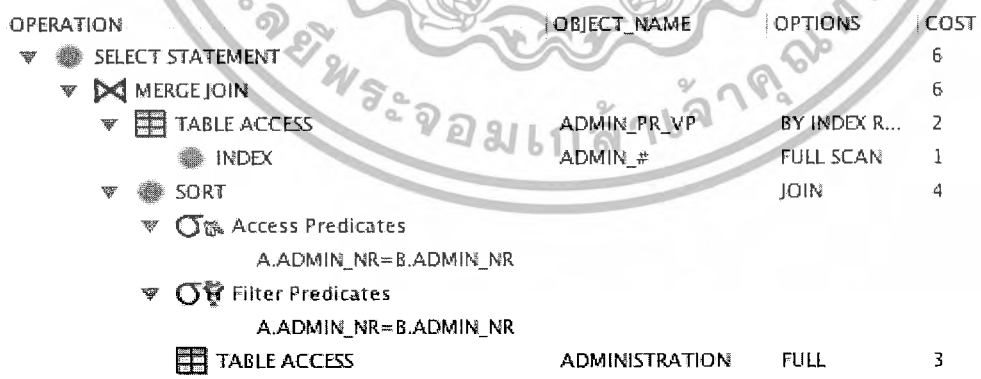
การเปลี่ยน plan เมื่อมีการใช้ index การจัดการเก็บสถิตินั้นสีความสำคัญต่อความประสิทธิภาพในการตอบคำถามของ อ๊อบติไมเซอร์ พอสมควรดังนั้นเมื่อมีการใช้ คอลั่มไหนในการ จอยน์บ่อยๆนั้นก็ควรมีการเก็บ Index ยกตัวอย่างกรณีต่างๆดังนี้

ตัวอย่างที่ 1

```
SELECT a.pres_name , b.year_inaugurated
FROM admin_pr_vp a, administration b
WHERE a.admin_nr = b.admin_nr;
```

รูปที่ 3.19 ตัวอย่าง คิวรี่ ที่ทำให้เกิด hash join

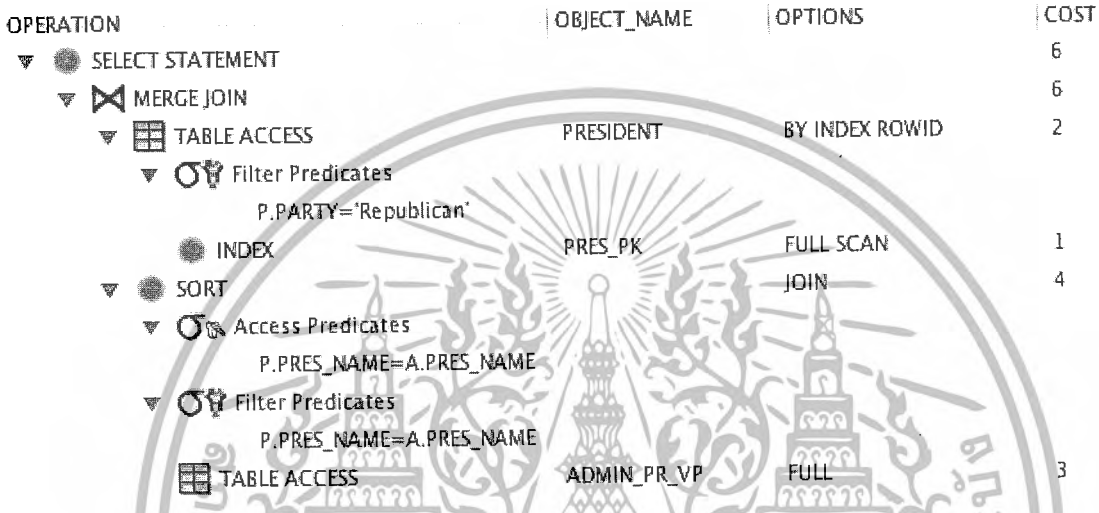
เมื่อลองทำการเพิ่ม primary index ให้กับ คอลั่ม ADMIN_PR_VR.ADMIN_NR และใช้คิวรี่ เดิม ทำให้ออปติไมเซอร์เปลี่ยนไปใช้เส้นทางอื่น แล้วเลือกใช้การจอยน์ แบบ merge join ทำให้ cost ลดลง และเนื่องจาก ข้อมูลมีจำนวนไม่มาก จึงทำให้ cost ต่างกัน นิดเดียว



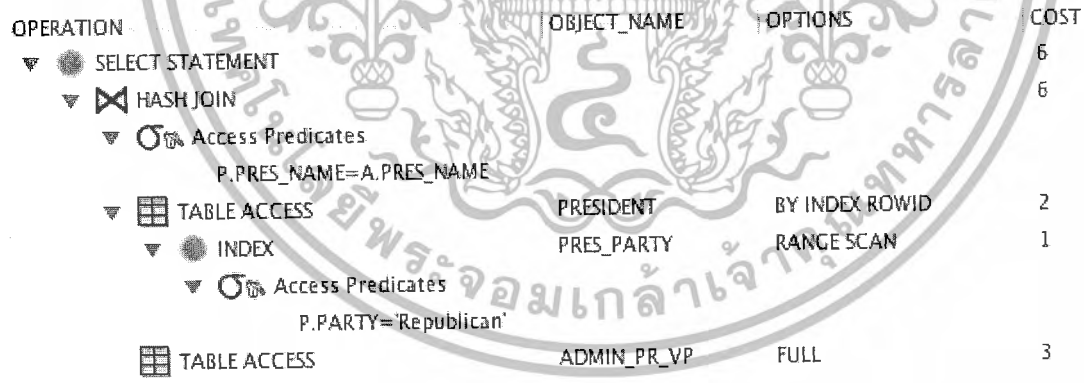
รูปที่ 3.20 ตัวอย่าง plan ที่ใช้ merge join

ตัวอย่างที่ 2

```
SELECT p.pres_name , a.vice_pres_name
FROM president p, admin_pr_vp a
WHERE p.pres_name = a.pres_name
AND p.party ='Republican';
```



รูปที่ 3.21 Plan เมื่อไม่มีการเพิ่ม index



รูปที่ 3.22 Plan เมื่อมีการเพิ่ม index คอลัม party

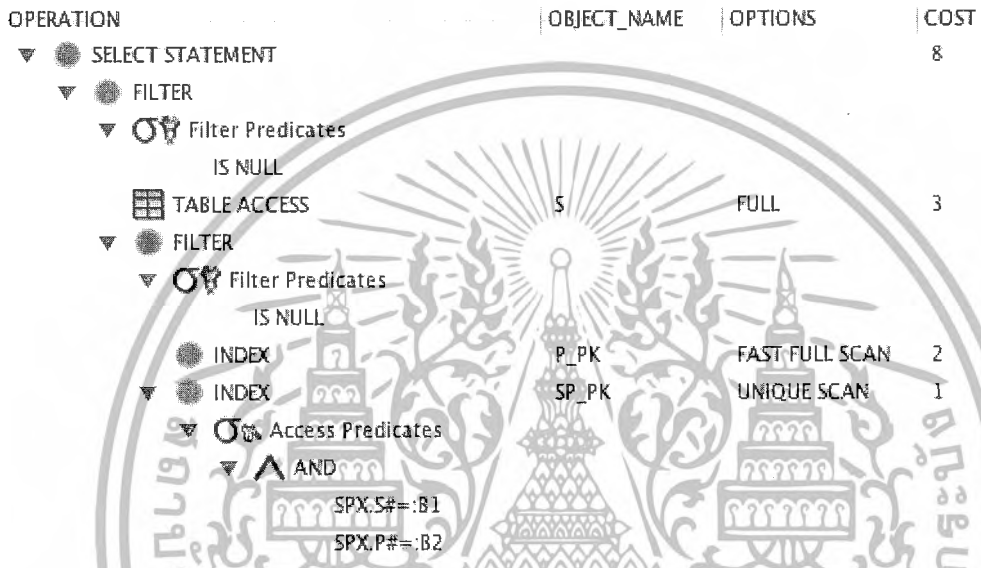
เมื่อมีการเพิ่ม index ที่ คอลัม party แล้วนั้นจะเห็นว่า การ scan index ของ PRES_PK เปลี่ยนจาก full scan เป็น range scan และไม่มีการ sort ข้อมูลก่อน join แต่จะเห็นว่า cost ไม่ต่างกันเลย เนื่องจากข้อมูลในตารางมีจำนวนไม่มากจึงไม่เห็นความแตกต่าง

ตัวอย่างที่ plan ของ Correlated Subqueries

SELECT sx.sname FROM s sx WHERE NOT EXISTS

*(SELECT * FROM p px WHERE NOT EXISTS*

*(SELECT * FROM sp spx WHERE spx.s#=sx.s# AND spx.p#=px.p#));*



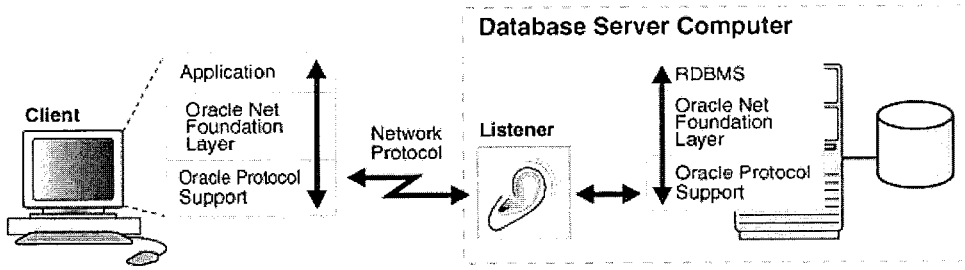
รูปที่ 3.23 ตัวอย่าง Plan ของ Correlated Subqueries

จากตัวอย่างจะเห็นว่า ในการ จอยน์ใช้ในสุดของ index sp_pk นั้นมีการใช้ ตัวแปร B1, B2 มาใช้ในการจอยน์ ซึ่งต่างจาก คิวรี่ ทั่วไปที่ ใช้ คอลั่ม ในการจอยน์

3.9 แนวความคิดการเชื่อมต่อ

แนวความคิดการเชื่อมต่อ (Connectivity Concepts) ใน ออราเคิล จะมีการติดต่อเข้าใช้งานของผู้ใช้อยู่ 2 แบบ คือ Dedicate Server และ Shared Server ซึ่งทั้งสองแบบจะให้ผลลัพธ์กับยูสเซอร์ที่เหมือนกัน

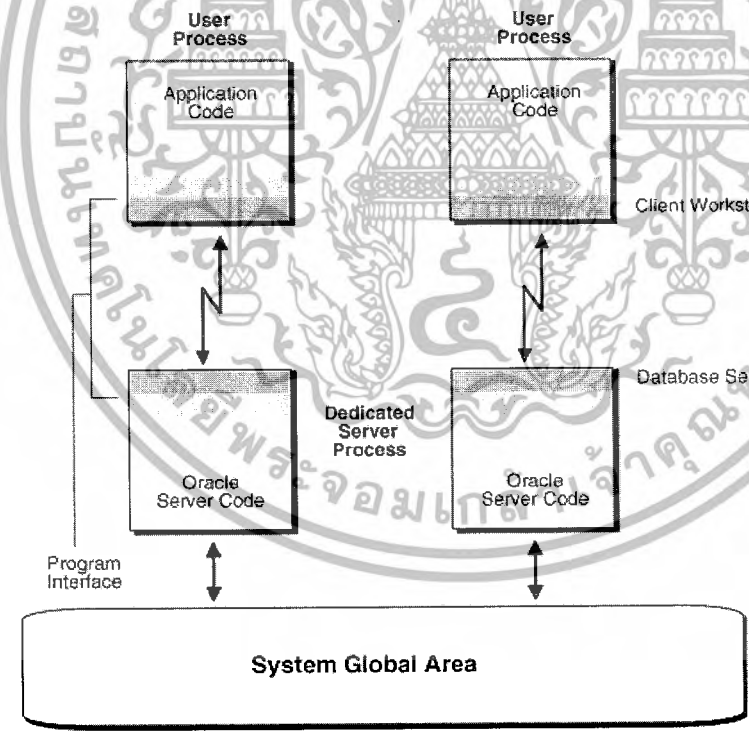
ใน ออราเคิล จะมีเซอร์วิส ที่รอรับการติดต่อจากไคลเอนต์เรียกว่า LISTENER (Default port 1521) ซึ่งการติดต่อออราเคิลจะต้องทำผ่าน LISTENER และมี NAMING SERVICE ซึ่งจะเป็น เซอวิสที่จัดการกับ การติดต่อนั้นๆ โดยที่ หนึ่ง LISTENER สามารถมีได้หลาย NAMING SERVICE ซึ่งแต่ละ NAMING SERVICE สามารถมีการทำงานที่ต่างกัน เช่น เซอร์วิส A เป็นแบบ Dedicate Server และ เซอร์วิส B เป็น แบบแชร์เซิร์ฟเวอร์ก็ได้



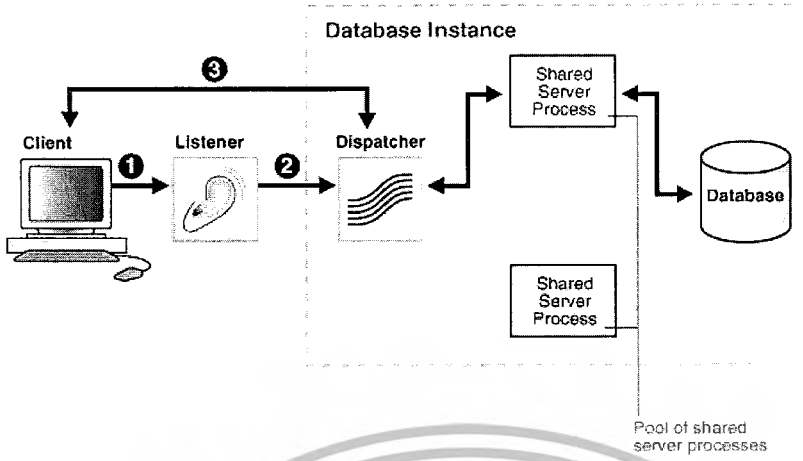
รูปที่ 3.24 ออราเคิล LISTENER

3.9.1 ดีดิเคท เซิร์ฟเวอร์ (Dedicated Server)

การทำงานระหว่าง เซิร์ฟเวอร์โพรเซส และ ยูสเซอร์โพรเซสจะเป็นแบบหนึ่งต่อหนึ่ง คือ ออราเคิล จะสร้าง เซิร์ฟเวอร์โพรเซส 1 โพรเซส สำหรับจัดการกับ ยูสเซอร์โพรเซส 1 โพรเซส ถ้ามี ยูสเซอร์โพรเซส เข้ามาอีก ก็จะสร้าง เซิร์ฟเวอร์โพรเซส มาจัดการอีก ซึ่งแบบ ดีดิเคทเซิร์ฟเวอร์นี้จะเป็น ค่าเริ่มต้นของ ออราเคิล



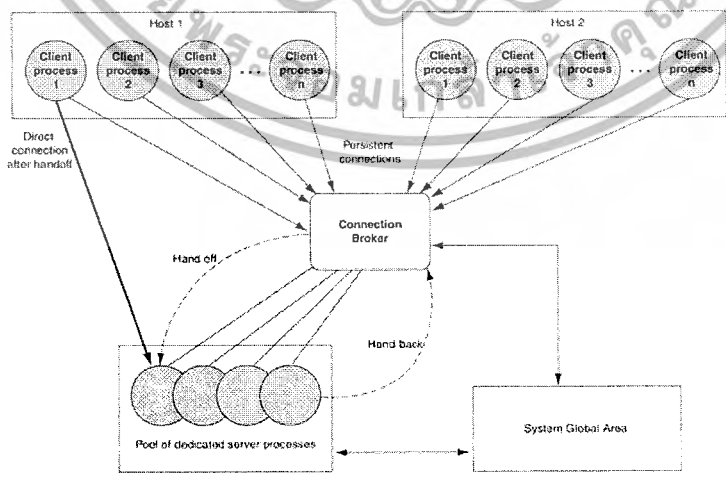
รูปที่ 3.25 Dedicate Server



รูปที่ 3.26 การทำานของ LISTENER กับ Dedicate Server

ซึ่งแบบ ดีดิเคทเซิร์ฟเวอร์ นี้ในกรณีที่มีการติดต่อ เข้ามาเยอะๆจะทำให้ CPU ทำงานหนัง และใช้พื้นที่ของหน่วยความจำหลัก เยอะ และเมื่อการเชื่อต่อถูกตัดขาด ก็จะมีโพรเซสค้างอยู่ที่ให้เปลือง ทรัพยากร เพื่อให้สามารถรองรับการติดต่อ เยอะๆ โดยมีโพรเซส ที่จำกัน ออราเคิล จึงมีคอนเนคชัน พลูลิ่ง

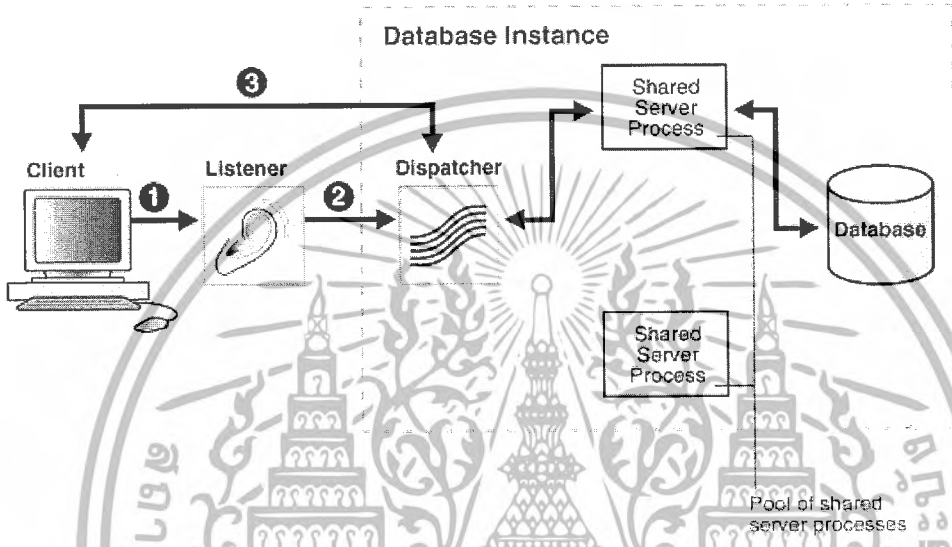
คอนเนคชัน พลูลิ่ง คือ ดีดิเคทเซิร์ฟเวอร์อีกแบบหนึ่ง ซึ่งจะมีโพรเซส ที่รอรับ การติดต่อจำนวนจำกันโดยจะสระของโพรเซส ที่ได้รับการเชื่อมต่อแล้ว (pool of dedicate server process) กับสระของโพรเซสที่ไม่มีการเชื่อมต่อ (Connection broker) เมื่อ ดีดิเคมเซิร์ฟเวอร์โพรเซส ไม่มีการทำงานหรือไม่มีคำสั่งเข้ามาช่วงระยะเวลาหนึ่ง หรือ ถูกตัดการเชื่อมต่อไปแล้ว ก็จะนำโพรเซส นั้นมาไว้ใน Connection broker เพื่อรอรับการติดต่อใหม่ และเมื่อมีการติดต่อใหม่เข้ามา ก็จะนำเอา โพรเซส ใน connection broker ไปรับ ยูสเซอร์โพรเซส แล้วย้าย โพรเซส นั้นไปยัง pool of dedicate server process



รูปที่ 3.27 การทำงานของ คอนเนคชัน พลูลิ่ง

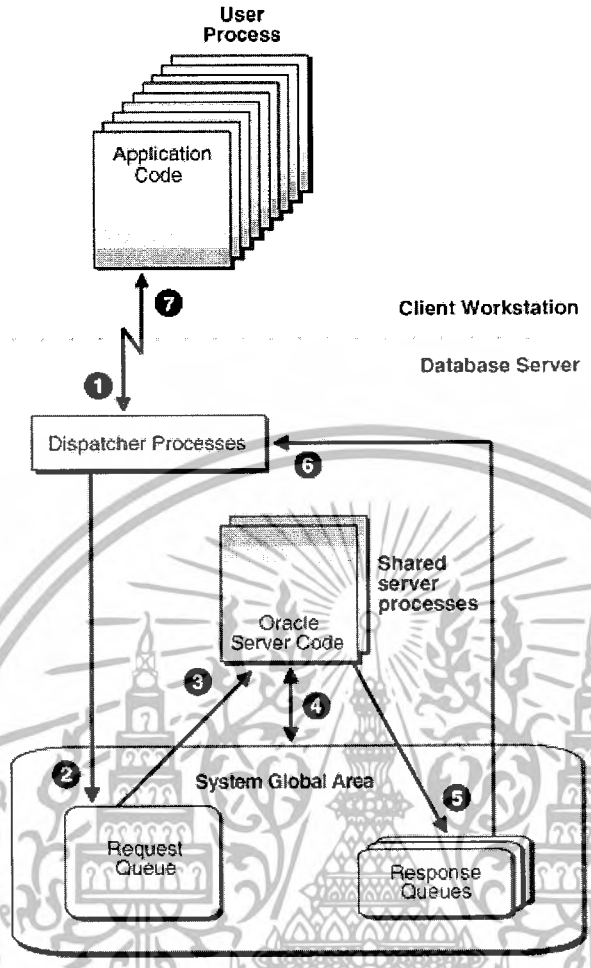
3.9.2 แชร์ เซิร์ฟเวอร์ (Shared server)

ออราเคิล จะสร้าง เซิร์ฟเวอร์โพรเซส 1 โพรเซส สำหรับจัดการกับ ยูสเซอร์โพรเซส หลาย ๆ โพรเซส ซึ่งแบบ แชร์เซิร์ฟเวอร์นั้น จะมีตัว จัดลำดับให้ การร้องขอเรียกว่า ดีแพทเชอร์ (ดีแพทเชอร์) หลังจาก ยูสเซอร์โพรเซสติดต่อมายังเซอร์วิส ที่ทำงานแบบ แชร์แซอร์เวอร์ผ่าน ลิสเทนเนอร์ (LISTENER) แล้ว จะมีขึ้นตอนการทำงานดังนี้



รูปที่ 3.28 การทำงานของ ลิสเทนเนอร์ กับ แชร์เซิร์ฟเวอร์

1. ยูสเซอร์ สั่ง คิวรี่ มายัง ดีแพทเชอร์
2. ดีแพทเชอร์สั่ง คิวรี่ ไปเก็บไว้ในคิวการร้องขอ
3. คิวการร้องขอสั่ง คิวรี่ ไปให้ เซิร์ฟเวอร์โพรเซส
4. เซิร์ฟเวอร์โพรเซสทำการหาผลลัพธ์ให้ คิวรี่
5. เซิร์ฟเวอร์โพรเซสส่งผลลัพธ์ไปเก็บไว้ยัง คิวการร้องขอ
6. คิวการร้องขอส่งผลลัพธ์ไปให้ดีแพทเชอร์
7. ดีแพทเชอร์ส่งผลลัพธ์ให้ ยูสเซอร์

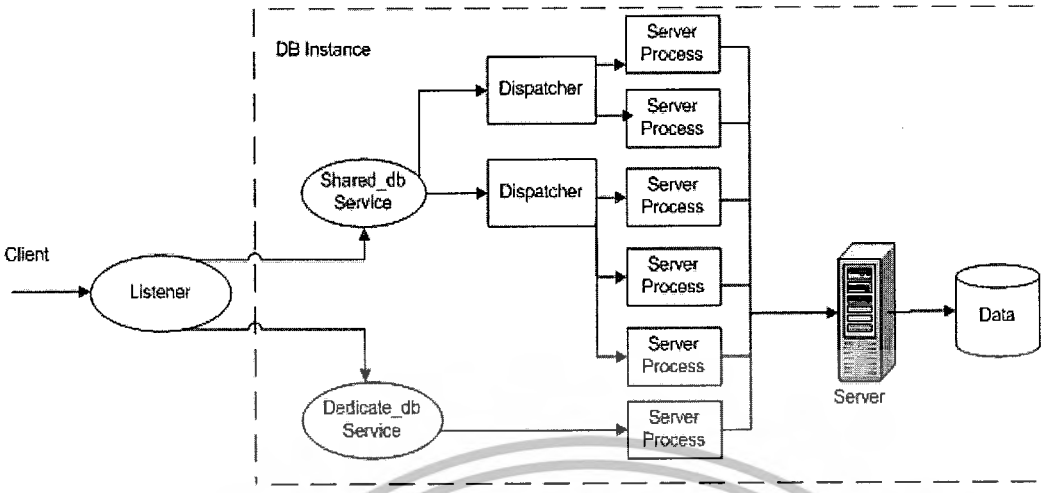


รูปที่ 3.29 การทำงานของ แชร์เซิร์ฟเวอร์

ในกรณีที่มี การติดต่อของยูสเซอร์โพรเซส เข้ามาเยอะๆ เกินกว่าที่ ดีแพทเชอร์ จะรองรับได้ (สามารถกำหนดได้ว่า หนึ่ง ดีแพทเชอร์ สามารถรองรับ โพรเซส ได้กี่โพรเซส) ออราเคิล ก็จะใช้ดีดิเคทเซิร์ฟเวอร์ กับโพรเซสที่ดีแพทเชอร์ ไม่สามารถรองรับได้

3.10 การทดลอง การติดต่อผ่านเครื่อข่าย

การทดลอง การติดต่อผ่านเครื่อข่าย (Net Connection) ในการทดลองนี้จะสร้าง เนมมิ่งเซอร์วิส (Naming Service) ขึ้นมา 2 เซอร์วิส มีชื่ออว่า Dedicate_db (เป็นแบบ ดีดิเคท เซอรเวอร์คอลเน็คชันพลูลิ่ง) และ Shared_db เป็นแบบ (แชร์เซิร์ฟเวอร์) โดยมี ลิสเทนเนอร์ชื่อ LISTENER ดังรูป



รูปที่ 3.30 แบบจำลองการ ติดตั้ง Naming Service

การสร้าง LISTENER ตอนที่เราสร้าง DB Instance นั้นจะบังคับให้เราสร้าง LISTENER ขึ้นมาหนึ่งตัวอยู่แล้วแต่เราสามารถ แก้ไขหรือสร้างเพิ่มได้ โดยใช้ โปรแกรม $ netce (เป็น GUI) หรือ โดยการแก้ไขไฟล์ *$ORACLE_HOME/network/admin/listener.ora* โดยมีรายละเอียดดังนี้

```
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (GLOBAL_DBNAME = shared_db)
   (SID_NAME = oracle)
   (ORACLE_HOME = /u01/app/oracle/product/11.1.0/db_1)
  )
  (SID_DESC =
   (GLOBAL_DBNAME = dedicate_db)
   (SID_NAME = oracle)
   (ORACLE_HOME = /u01/app/oracle/product/11.1.0/db_1)
  )
 )
```

GLOBAL_DBNAME	คือ	ชื่อของ เนมมิ่งเซอร์วิส ที่ระบุให้ ลิสเทนเนอร์นี้รู้จัก
SID_NAME	คือ	ชื่อของ DB Instance ที่ เนมมิ่งเซอร์วิส นั้น
ORACLE_HOME	คือ	พาธ ของ ORACLE_HOME

เราสามารถใช้คำสั่ง *$ lsnrctl start* เพื่อเริ่มการทำงานของลิสเทนเนอร์ ในกรณีที่เปิดระบบปฏิบัติการใหม่และใช้คำสั่ง *$ lsnrctl service* เพื่อดูรายละเอียดของลิสเทนเนอร์ว่ามีเซอร์วิสใดอยู่บ้าง

การสร้างเนมมิ่งเซอร์วิสการสร้าง เนมมิ่งเซอร์วิสนั้นสามารถสร้างได้โดยใช้ program *$ netca* หรือไปแก้ไขไฟล์ ที่ชื่อ *$ORACLE_HOME/network/admin/tnsnames.ora*

```
ORACLE =
(DESCRIPTION =
(ADDRESS = (PROTOCOL = TCP)(HOST = localhost.localdomain)
(PORT = 1521))
(CONNECT_DATA =
(SERVER = DEDICATED)
(SERVICE_NAME = oracle) )
(
DEDICATE_DB =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)
(PORT = 1521))
(LOAD_BALANCE = yes)
(
(CONNECT_DATA =
(SERVER = POOLED)
(SERVICE_NAME = oracle) )
(
SHARED_DB =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = localhost.localdomain)
```

(PORT = 1521))

(LOAD_BALANCE = yes))

(CONNECT_DATA =

(SID = oracle)

(SERVER = SHARED)))

จะเห็นว่า มีอยู่ 3 เซอวิสซึ่ง เซอร์วิสออราเคิล นั้นเป็นที่เป็นแบบ เซอร์วิสเริ่มต้นถูกสร้างขึ้นตอนที่เราสร้าง DB Instance โดยตัวแปร *SERVER=[.]* เป็นตัวที่ระบุว่าเซอร์วิสใดทำงานแบบใด ซึ่ง จะเห็นว่า DEDICATE_DB มี SERVICE=POOLED หมายถึงเป็นแบบ คอนเนคชัน พลูลิ่ง

3.10.1 การปรับแต่งแชร์เซิร์ฟเวอร์

นั้นสามารถเปิดใช้งาน โดยกำหนดค่าในพารามิเตอร์ (โดยการแก้ พารามิเตอร์ไฟล์หรือใช้คำสั่ง ALTER SYSTEM SET PARAMETER = x) ดังนี้

3.10.1.1 SHARED_SERVERS

ถ้าเป็น 0 หมายความว่า ปิดการ แชร์เซิร์ฟเวอร์ (ถึงแม้เรา จะกำหนดใน เนมมิ่งเซอร์วิสแล้วก็ตาม แต่ถ้ามีค่าเป็น 0 และ ไม่มี ดิสแพทเชอร์มารองรับเซอร์วิสนั้นจะทำงานเป็นแบบ ดีดิเคท เซิร์ฟเวอร์(Dedicate server) ต้องเป็นค่าที่มากกว่า 0 ซึ่งจะหมายถึง จำนวน Process เริ่มต้นของแชร์ เซิร์ฟเวอร์

3.10.1.2. MAX_SHARED_SERVERS คือจำนวน แชร์ เซิร์ฟเวอร์โพสเซสที่มีได้มากที่สุด ถ้ามี การติดต่อเข้ามามากเกินที่กำหนดก็จะกลายเป็นแบบ ดีดิเคท เซิร์ฟเวอร์

3.10.1.3. MAX_DEPATCHERS คือจำนวน ดิสแพทเชอร์ ที่มีได้มากที่สุด

3.10.1.4. DEPATCHER รายละเอียดของตัวแปรนี้จะไม่ใช่ตัวเลข แต่จะมีพารามิเตอร์ ย่อยต่างๆ ที่จำเป็นดังนี้ ดังนี้

- PROTOCOL (PRO or PROT) คือ โปรโตคอล ที่ใช้ในการติดต่อ
- DEPATCHERS (DIS or DISP) คือ จำนวน ดิสแพทเชอร์เริ่มต้น
- SESSIONS (SES or SESS) คือจำนวนเซสชัน ที่แต่ละดิสแพทเชอร์รองรับได้
- CONNECTIONS (CON or CONN) คือ จำนวน การติดต่อที่แต่ละดิสแพทเชอร์ รองรับได้
- SERVICE (SER or SERV) คือ ชื่อเซอร์วิส ที่ดิสแพทเชอร์ ให้บริการ

ในการทดลองมีการกำหนดค่าต่างๆ ดังนี้

- SHARED_SERVERS=5
- MAX_SHARED_SERVERS=20
- MAX_DEPATCHERS=10
- DEPATCHER=(protocol=TCP)(disp=5)(con=200)(sess=200)(serv=shared_db)

3.10.2 Configuring Database Resident connection pooling

การเปิดใช้งานคอนเนคชัน พลูลิ่ง นั้นทำได้โดยใช้คำสั่ง EXECUTE DBMS_CONNECTION_POOL.START_POOL(); ในการเปิดใช้งาน และ สามารถแก้ไขพารามิเตอร์ต่างๆ โดยใช้คำสั่ง EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','PARA','*'); ซึ่งพารามิตอร์ ต่างๆที่จำเป็นมีดังนี้

3.10.2.1 MINSIZE คือจำนวนโพรเซสต่ำสุดใน pool (ค่าเริ่มต้นเท่ากับ 4)

3.10.2.2 MAXSIZE คือจำนวนโพรเซสมากที่สุดใน pool (ค่าเริ่มต้นเท่ากับ 40)

3.10.2.3 INCRSIZE คือจำนวนโพรเซสที่เพิ่มขึ้นในแต่ละครั้งเมื่อ process ไม่เพียงพอให้บริการ user (ค่าเริ่มต้นเท่ากับ 3)

3.10.2.4 INACTIVITY_TIMEOUT คือ เวลาในการรอเพื่อนำ โพรเซสที่ไม่มีการทำงาน กลับมาใช้ และยังสามารถดูหน้าจอการทำงานของ pool ได้โดยมี วิว ต่างๆ ดังนี้

- DBA_CPOOL_INFO เก็บรายละเอียดในการกำหนดค่าต่างๆของ pool
- V$CPOOL_STATS เก็บสถิติการใช้งานต่างๆของ pool

ในการทดลองมีการกำหนดค่าต่างๆดังนี้

MINSIZE = 10

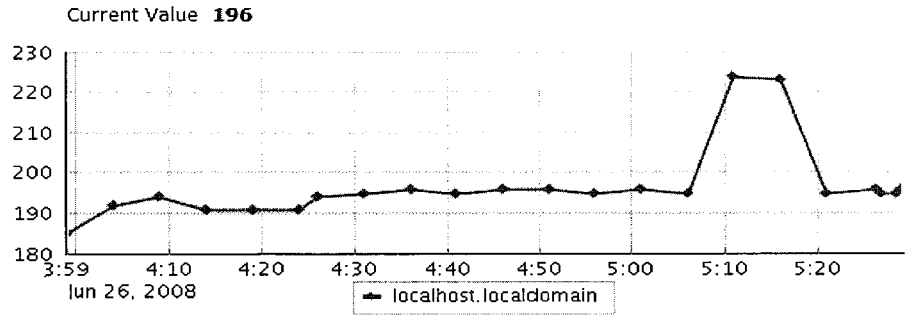
MAXSIZE = 50

INCRSIZE = 2

INACTIVITY_TIMEOUT = 60

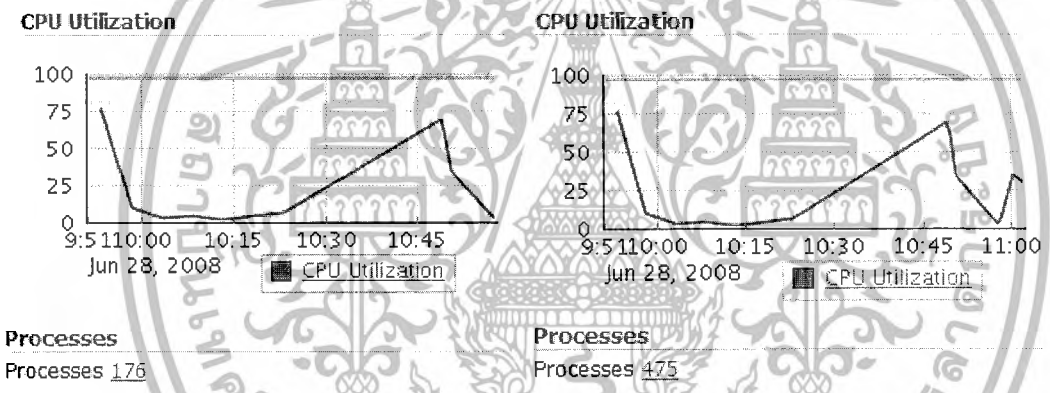
3.10.3 การทดลองการติดต่อแบบต่างๆ

ดูจำนวนโพสเซส ที่เพิ่มขึ้น ในการติดต่อแบบ แชร์เซิร์ฟเวอร์ และ ดีดิเคทเซิร์ฟเวอร์

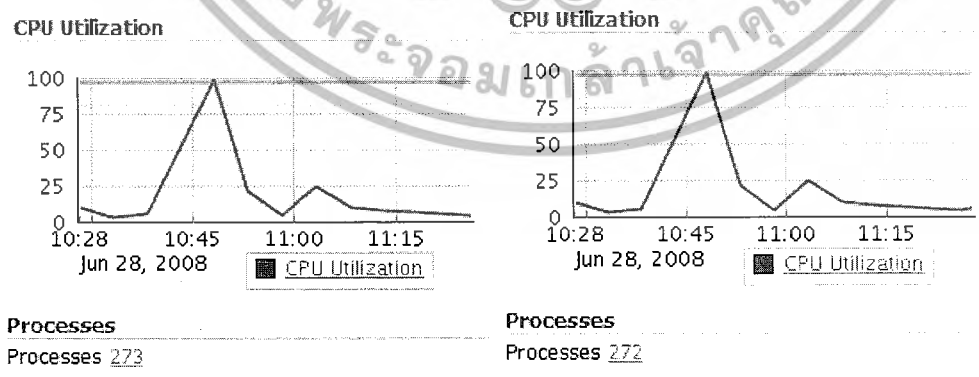


รูปที่ 3.31 ตัวอย่างเมื่อ การติดต่อ แบบแชร์เซิร์ฟเวอร์ 24 ยูสเซอร์ จาก โปรแกรม sqlDeveloper

จากตัวอย่างจะเห็นได้ว่า จำนวนโ ไม่ได้เพิ่มขึ้น เพราะว่าการการติดต่อแบบแชร์เซิร์ฟเวอร์จะมี ดีแพทเชอร์ ไว้คอยรองรับหลายๆ ยูสเซอร์ แต่เมื่อลองติดต่อด้วยเซอร์วิส ที่เป็นแบบดีดิเคท เซิร์ฟเวอร์

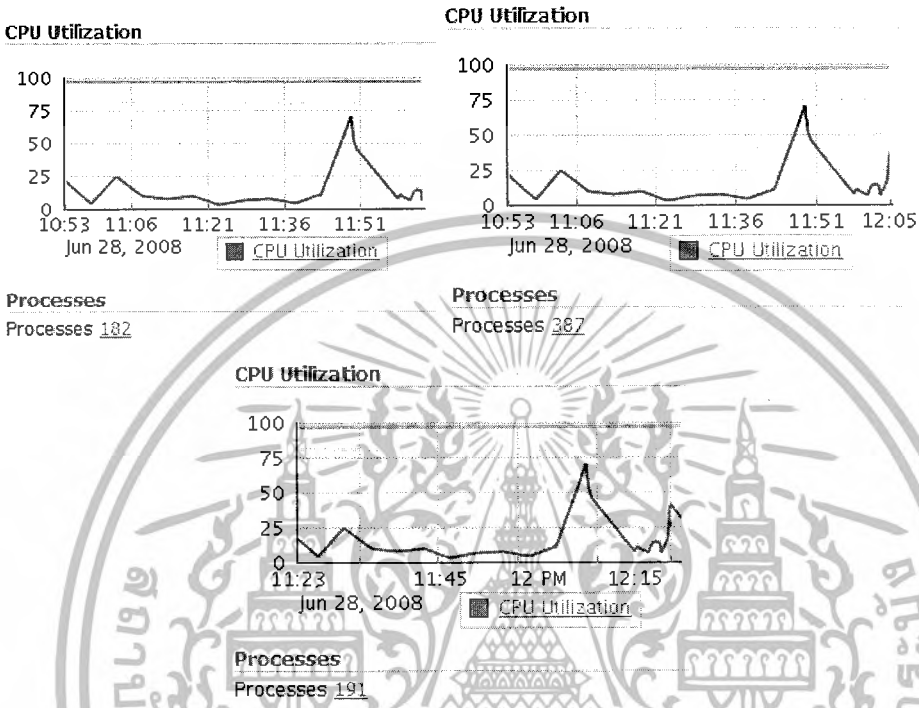


รูปที่ 3.32 ตัวอย่าง มี300 การติดต่อแบบดีดิเคท เซิร์ฟเวอร์ จะเห็น ว่า cpu ทำงาน เพิ่มมากขึ้น



รูปที่ 3.33 ตัวอย่าง 300 การติดต่อแบบแชร์เซิร์ฟเวอร์

จะเห็นว่า cpu ทำงานเพิ่มขึ้นเพียงเล็กน้อยที่เริ่มต้น มีโพสเซส 273 โพสเซส เพราะว่าหลังจากการเชื่อมต่อแบบดีดิเคทนั้น ยังคงมี อินแอคทีฟโพสเซส เหลืออยู่แก้ปัญหาโดยการใช้ คอนเนคชัน พลูลิ่ง



รูปที่ 3.34 ตัวอย่าง 300 คอนเนคชัน พลูลิ่งจะเห็นว่าโพรเซส ที่ ค้างอยู่ลดลง

สรุปการเปรียบเทียบเซิร์ฟเวอร์แบบต่างๆเซอร์วิสแบบ แชร์เซิร์ฟเวอร์นั้นจะกินทรัพยากรของเซอร์ น้อยกว่าแบบดีดิเคทเซิร์ฟเวอร์เพราะ แชร์เซิร์ฟเวอร์นั้น จะมี โพสเซสน้อยทำให้ไม่เปลือง address spaces ของระบบปฏิบัติการ และเหมาะกับงานที่มีการเชื่อต่อฐานข้อมูลเข้ามาจำนวนมากแต่ทำงานเล็กๆ และใช้เวลาน้อยในการทำงานแต่ถึงอย่างไรก็ตามเราก็จำเป็นต้องมีเซอร์วิสแบบ ดีดิเคทเซิร์ฟเวอร์เพราะเราจำเป็นต้องใช้ ดีดิเคทเซิร์ฟเวอร์เซอร์วิส ในการทำ แบคอัพ และสำหรับรองรับการใช้งานที่ที่งานเป็นเวลานานๆและมี คิวรี่ จำนวนมากๆ และมีไว้เพื่อรองรับการเชื่อต่อเยอะๆจนเกินขอบเขตของ แชร์เซิร์ฟเวอร์เซอร์วิส เพื่อให้ การเชื่อมต่อที่เกินมานั้นสามารถเข้าทำงานได้ โดยไม่ต้องรอให้มีช่องทางการเชื่อมต่อที่ว่าง

3.11 การกำหนดสิทธิการใช้งาน

การกำหนดสิทธิการใช้งาน (System Privileges) คือสิทธิต่างๆ ในการทำงานกับระบบฐานข้อมูล ออราเคิล เช่น สิทธิในการติดต่อเข้ามายังฐานข้อมูล สิทธิในการสร้างตารางอินเด็กซ์หรือออบเจ็กต์ สิทธิในการสร้าง ดีบีสเปรซ และสิทธิในการใช้งาน เทเบิลสเปรซ เป็นต้น ผู้ใช้ที่มีสิทธิใน ซิสเทมพริวิเลจ (System Privileges) สูงสุดในฐานข้อมูลคือ ผู้ใช้ SYS, SYSTEM และยูสเซอร์ DBA ดังนั้นจึงไม่ควรที่จะให้ผู้ใช้อื่นรู้รหัสผ่านของผู้ใช้เหล่านี้ได้

ตัวอย่างเช่น ผู้ใช้ชื่อ Retail จะสร้างตารางได้จะต้องได้รับสิทธิ CREATE TABLE ก่อน เป็นต้น ดังนั้นเราควรมีการกำหนดในฐานข้อมูลว่าผู้ใช้จะใช้งาน ซิสเทมพริวิเลจ ได้ขนาดไหนในดาต้าดิกชันนารี่(ดาต้า ดิกชันนารี่) ซึ่งเราสามาที่จะทำการกำหนดค่านี้ได้ด้วยการตั้งค่าให้กับตัวแปรที่ชื่อ O7_DICTIONARY_ACCESSIBILITY โดยการที่เข้าไปตั้งค่าในไฟล์ที่ชื่อ initSID.ora หรือ การที่ใช้คำสั่งเอสคิวแอล โดยที่เราล๊อกออนบน SQL*Plus เป็น SYS หรือ SYSDBA โดยใช้คำสั่ง ALTER SYSTEM SET O7_DICTIONARY_ACCESSIBILITY=FALSE SCOPE=SPFILE

กฎต่างๆ ที่มีใน SYS มีดังนี้

- SELECT_CATALOG_ROLE คือการที่ให้สิทธิผู้ใช้ในการ Select ดูข้อมูลใน ดาต้าดิกชันนารี
- EXECUTE_CATALOG_ROLE คือการที่ให้สิทธิผู้ใช้ในการ เอ็กซีคิวข้อมูลที่เป็น แพ็กเก็ตและ Procedure ในดาต้าดิกชันนารี่ได้
- DELETE_CATALOG_ROLE คือการที่ให้สิทธิผู้ใช้ในการ ลบเร็คคอร์ดจาก System audit table (AUD$) ได้

3.11.1 การให้สิทธิและยกเลิกสิทธิ ซิสเท็มพริวิเลจ (Granting And Revoking System Privileges) เราสามารถที่จะให้มอบสิทธิให้แก่ผู้ใช้อื่น หากเรามีสิทธิใน Role นั้นอยู่ ในขณะเดียวกันเราก็สามารถที่จะกำหนดซิสเท็มพริวิเล็จให้กับผู้ใช้คนอื่น แต่โดยทั่วไปแล้วสิทธิเหล่านี้น่าจะเป็นของ แอดมินิสเตรเตอร์ หรือ ยูสเซอร์ ทั่วไปไม่ควรทำได้

วิธีการ ให้สิทธิ(Grant) หรือ ยกเลิกสิทธิ (Revoke) มีด้วยกัน 2 วิธีคือ

- การใช้คำสั่งเอสคิวแอล
- การที่ให้ ออราเคิล จัดการให้เอง

คนที่สามารถให้สิทธิกับ ซิสเท็มพริวิเลจ กับผู้ใช้คนอื่นต้องเป็นชนิดดังนี้คือ

- ยูสเซอร์ ที่เป็น ADMIN OPTION
- ยูสเซอร์ ที่เป็น GRANT ANY PRIVILEGES

3.11.2 ความหมายของ ANY และ PUPLIC หากเราใช้คำว่า ANY จะทำให้คุณมีสิทธิในการกำหนด พริวิเลจทั้งหมดของออบเจ็ก ที่มีอยู่ในฐานข้อมูล หากเราใช้คำว่า PUPLIC จะทำให้ผู้ใช้ทุกคนที่อยู่ในฐานข้อมูลนั้นได้รับสิทธินั้นๆ

3.11.3 คำสั่งที่ใช้ในการให้สิทธิ ซิสเท็มพริวิเลจ

- ***GANT*** *privilege_name*
- ***ON*** *object_name*
- ***TO*** *{user_name | PUBLIC | role_name}*
- *[WITH GRANT OPTION];*

privilege_name คือสิทธิที่จะให้กับผู้ใช้นั้น

object_name คือชื่อของอ็อบเจ็กต์ เช่น Table, View, Stored Proc Sequence นั้น

user_name คือชื่อของผู้ใช้ที่จะได้รับสิทธินั้น หรือ PUBLIC | role_name ใช้แทนกับผู้ใช้ทุกๆ คนในฐานข้อมูล

[WITH GRANT OPTION] คือการที่เป็นสิ่งที่กำหนดว่าผู้ใช้นั้นสามารถที่จะให้สิทธิคนอื่นได้หรือไม่

3.11.4 คำสั่งที่ใช้ในการยกเลิกสิทธิซิสเท็มพริวิเลจ

- ***REVOKE*** *privilege_name*
- ***ON*** *object_name*
- ***FROM*** *{c| PUBLIC | role_name}*

privilege_name คือสิทธิที่จะยกเลิกการให้กับผู้ใช้นั้น

object_name คือชื่อของอ็อบเจ็กต์ เช่น Table, View, Stored Proc Sequence นั้น

object_name คือชื่อของผู้ใช้ที่จะได้รับสิทธินั้นหรือ PUBLIC | role_name ใช้แทนกับผู้ใช้ทุกๆ คนในฐานข้อมูล

ในการที่ REVOKE กับผู้ใช้คนใดแล้วจะไม่มีผลไปกระทบต่อผู้ใช้ที่ถูกผู้ใช้คนนั้น GRANT ให้ และผู้ที่ควรจะมีสิทธิในการ GRANT หรือ REVOKE ก็น่าจะเป็นผู้ใช้ที่เป็น DBA, SYS หรือ SYSTEM

3.12 อ็อบเจ็กต์ พริวิเลจ

อ็อปเจ็กต์ พริวิเลจ (Object Privileges)คือสิทธิในการทำงานกับออบเจ็กต์ต่างๆ ในฐานข้อมูล เช่น สิทธิในการอ่านหรือเปลี่ยนแปลงข้อมูลในตาราง View และ Sequence ของผู้ใช้เป็นต้น โดยปกติออบเจ็กต์ที่สร้างขึ้นภายใต้ผู้ใช้ใด ผู้ใช้คนนั้นจะมีสิทธิทุกอย่างกับออบเจ็กต์ที่อยู่ภายใต้ผู้ใช้นั้น ผู้ใช้อื่นๆ ไม่สามารถ จะทำงานกับออบเจ็กต์นั้นๆ ได้หากไม่ได้รับอนุญาติจากผู้ใช้ที่เป็นเจ้าของก่อน เช่น ถ้าผู้ใช้ชื่อ James ต้องการอ่านข้อมูลจากตาราง Retail.Employee เขาจะต้องได้รับสิทธิในการ SELECT ตาราง Empleyee จากผู้ใช้ Retail ก่อน เป็นต้น

3.12.1 อ็อบเจ็กต์ พริวิเลจ

ที่ควรทราบมีดังนี้

- SELECT object คือสิทธิในการ Select ข้อมูลของออบเจ็กต์ที่กำหนด
- INSERT object คือสิทธิในการ Insert ข้อมูลของออบเจ็กต์ที่กำหนด
- DELETE object คือสิทธิในการ Delete ข้อมูลของออบเจ็กต์ที่กำหนด
- UPDATE object คือสิทธิในการ Update ข้อมูลของออบเจ็กต์ที่กำหนด
- ALTER object คือสิทธิในการเปลี่ยนแปลงออบเจ็กต์ที่กำหนด
- INDEX object คือสิทธิในการสร้างอินเด็กซ์บนออบเจ็กต์ที่กำหนด
- REFERENCE object คือสิทธิในการอ้างอิงถึงออบเจ็กต์ที่กำหนด

3.12.2 คำสั่งในการให้สิทธิ อ็อบเจ็กต์ พริวิเลจ

- ***GRANT*** *Object_Privileges [(คอลัม name)]*
- ***ON*** *[owner.]object_name*
- ***TO*** *gratee [WITH GRANT OPTION]*

อ็อบเจ็กต์ พริวิเลจ คือสิทธิที่ให้กับผู้ใช้นั้นที่เป็น อ็อปเจ็กต์ พริวิเลจ [(กำหนดชื่อ คอลัม ในตารางที่จะให้สิทธิ)]

[owner.]object_name คือการกำหนดชื่อเจ้าของและชื่อออบเจ็กต์ ถ้าในกรณีที่ไม่ใช่เจ้าของออบเจ็กต์ต้องกำหนดชื่อเจ้าของออบเจ็กต์ด้วย

[WITH GRANT OPTION] คือการที่เป็นสิ่งที่กำหนดว่าผู้ใช้นั้นสามารถที่จะให้สิทธิคนอื่นได้หรือไม่

3.12.3 คำสั่งในการยกเลิกสิทธิ อ็อบเจ็กต์ พริวิเลจ

- ***REVOKE*** *Object_Privileges [(คอลัม name)]*

- ***ON*** *[owner.]object_name*

- ***FROM*** *gratee [CASCADE CONSTRAINT]*

อ็อบเจ็กต์ พริวิเลจ คือสิทธิที่ยกเลิกให้กับผู้ใช้นั้นที่เป็น อ็อปเจ็กต์ พริวิเลจ [(กำหนดชื่อคอลั่ม ในตารางที่จะให้สิทธิ)]

[owner.]object_name คือการกำหนดชื่อเจ้าของและชื่อออบเจ็กต์ ถ้าในกรณีที่ไม่ใช่เจ้าของออบเจ็กต์ต้องกำหนดชื่อเจ้าของออบเจ็กต์ด้วย

[CASCADE CONSTRAINT] คือการที่กำหนดว่าให้ลบคอนสเทรนออกไปด้วยเมื่อมีการ Revoke Reference Privileges

3.13 การจัดการกับ Role

Role ที่มีมาพร้อมกับการสร้างฐานข้อมูลที่สำคัญมีดังนี้

- **CONNECT** คือ Role ที่รวม ซิสเท็มพริวิเลจ ที่จำเป็นในการที่กำหนดให้ผู้ใช้สามารถล็อกอินเข้ามาทำงานในฐานข้อมูลได้ โดยประกอบด้วย ซิสเท็มพริวิเลจ ดังนี้ ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CRATE SESSION, CREATE SYNONYM, CREATE TABLE CREATE VIEW
- **RESOURCE** คือ Role ที่รวม ซิสเท็มพริวิเลจ ในการสร้างออบเจ็กต์ต่างๆ เพิ่มเติมจาการเชื่อมต่อโดยประกอบด้วย ซิสเท็มพริวิเลจดังนี้ CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
- **DBA** คือ Role ที่รวม ซิสเท็มพริวิเลจ ทั้งหมดที่ผู้ดูแลระบบฐานข้อมูลจำเป็นต้องใช้งาน ซึ่งพริวิเลจต่างๆ ใน Role นี้ได้กำหนดออปชัน with Admin Option ด้วยจึงสามารถที่จะ กำหนด สิทธิ ต่อผู้ใช้คนอื่นได้

3.13.1 การสร้าง Role

CREATE ROLE *role_name [NOT IDENTIFIED | IDENTIFIED BY password | IDENTIFIED EXTERNALLY];*

role_name คือชื่อของ Role ที่เราจะสร้าง

NOT IDENTIFIED คือกำหนดว่าสามารถเข้าใช้งาน Role นี้โดยไม่ต้องมีรหัสผ่าน

IDENTIFIED BY password คือกำหนดว่าการเข้าใช้งาน Role นี้ต้องมีรหัสผ่านในการทำงานตามที่กำหนดไว้ด้วย พาสเวิร์ด

IDENTIFIED EXTERNALLY คือกำหนดว่าการเข้าใช้งาน Role นี้ต้องมีรหัสผ่านในการทำงาน โดยกำหนดรหัสผ่านจากระบบปฏิบัติการ Externally

ตัวอย่าง CREATE ROLE RETAIL_ADMIN_ROLE;

CREATE ROLE RETAIL_QUERY_ROLE;

3.13.2 การลบ Role

DROP ROLE *role_name;*

role_name คือชื่อของ Role ที่ต้องการลบออกจากฐานข้อมูล

หากมีการลบ Role นั้นออกจากฐานข้อมูล จะมีผลทำให้ผู้ใช้ที่ได้รับ Role นั้นถูกยกเลิกสิทธิของ Role ไปโดยอัตโนมัติทำให้สิทธิที่ผู้ใช้เคยได้รับผ่าน Role นั้นใช้ไม่ได้อีกต่อไป

3.13.3 การให้สิทธิแก่ ROLE

GRANT *role_name* ***TO*** *grantee [WITH ADMIN OPTION];*

role_name คือชื่อของ Role ที่ต้องการ Grant

grantee คือ ชื่อผู้ใช้ที่ต้องการให้สิทธิ

[WITH GRANT OPTION] คือการที่เป็นสิ่งที่กำหนดว่าผู้ใช้นั้นสามารถที่จะให้สิทธิคนอื่นได้หรือไม่

3.13.4 การยกเลิกสิทธิ ROLE

REVOKE *role_name* ***FROM*** *gratee;*

role_name คือชื่อของ Role ที่ต้องการยกเลิก

grantee คือชื่อผู้ใช้ที่ต้องการยกเลิกสิทธิ

ตัวอย่าง REVOKET RETAIL_ADMIN_ROLE FROM JAMES;

3.14 ความปลอดภัยที่เป็นต่าเริ่มต้นตอนที่ติดตั้ง ออราเคิล ดาต้าเบส

- User Accounts เมื่อคุณทำการสร้างผู้ใช้ คุณสามารถจัดการกับผู้ใช้เหล่านี้ได้ โดยการกำหนด Password Profile ให้กับผู้ใช้เพื่อความปลอดภัยที่ดีกว่าได้
- Authentication methods ใน ออราเคิล ดาต้าเบส จะมีการตรวจสอบ ยืนยันตัวบุคคลกับผู้ใช้และ ผู้ดูแลระบบ ตัวอย่างเช่น การที่ทำการตรวจสอบจาก ออราเคิล ดาต้าเบส, ระบบปฏิบัติการและในเน็ตเวิค
- Privileges and Role การกำหนดสิทธิ ของผู้ใช้ ในการเข้าใช้ข้อมูล

- Application Security ในครั้งแรกที่จะทำการสร้างฐานข้อมูล แน่นอนว่าต้องกำหนดความปลอดภัยที่เหมาะสม
- User session information using application context Application context จะมีชื่อกับค่าที่เป็นตัวบ่งบอก Session information นั้น คุณสามารถที่จะทำการเรียกคืน Session information นั้นจากผู้ใช้ได้ เช่น User name หรือ terminal และการจำกัดการเข้าใช้งาน ดาต้าเบส และ Application ที่ผู้ใช้มีอยู่ใน Session นั้นๆ
- Database access on the row and คอลั่ม level using Virtual Private Database
- Encryption เป็นการป้องกันข้อมูลในเน็คเวิค ที่จะมีการปลอมแปลงข้อมูล โดยจะใช้ DBMS_CRYPTO และ DBMS_SQLHASH PL/SQL
- Auditing database activities คุณสามารถที่จะทำการตรวจสอบการกระทำเท่าที่ตกลงกันไว้ เช่น การตรวจสอบทุกคำสั่ง SQL, SQL Privileges, schema object, และ Network activity

3.14.1 ความปลอดภัยเกี่ยวกับผู้ใช้

ในฐานข้อมูลแต่ละฐานข้อมูลนั้นจะมีการเข้ามาใช้งานของผู้ใช้หลายๆ คนซึ่งในการจัดการกับความปลอดภัยของผู้ใช้ที่เข้ามาใช้งานนั้น เราต้องมีการสร้างผู้ใช้โดยที่คุณสามารถที่จะกำหนดสิทธิของผู้ใช้แต่ละคนได้ว่าสามารถทำอะไรกับข้อมูลเหล่านั้นได้บ้าง โดยสิทธิเหล่านี้เราเรียกว่า Privileges and Role

ในการสร้างผู้ใช้ใหม่เราจะใช้คำสั่ง CREATE USER โดยที่คุณต้องมีสิทธิในการใช้คำสั่งนี้ เพราะเป็นสิทธิที่สำคัญที่น่าจะมีเพียงแค่ผู้ใช้ที่เป็นผู้ดูแลฐานข้อมูล เท่านั้น

ตัวอย่าง

```
CREATE USER jward
IDENTIFIED BY password
DEFAULT เทเบิลสเปรซ data_ts
QUOTA 100M ON test_ts
QUOTA 500K ON data_ts
TEMPORARY เทเบิลสเปรซ temp_ts
PROFILE clerk;
GRANT CREATE SESSION TO jward;
```

ผู้ใช้จะมีรหัสที่เป็นเฉพาะและมี ค่าเริ่มต้นของการสร้างตารางและมี เทมโพรีเมื่อมีการสร้างตารางเป็น temp_ts และมีการกำหนด โคสต้า(QUOTA) ในตารางปล่าว และกำหนดสิทธิ ที่ทำการ แกรนให้กับผู้ใช้อย่างน้อยต้องมี CONNECT และ CREATE SESSION

และยูสเซอร์เนมของแต่ละผู้ใช้ที่มีในแต่ละฐานข้อมูลต้องไม่ซ้ำกันและชื่อกับ Role ต้องห้ามมีชื่อเหมือนกันด้วย ในสกีม่าแต่ละสกีม่า จะมีชื่ออ็อบเจ็กต์ ที่อยู่ภายในต้องมีชื่อที่เหมือนกันด้วย

และ ยูสเซอรแต่ละคนควรที่จะมีค่าเริ่มต้นให้กับ เทเบิลสเปรซ ในเวลาที่ผู้ใช้สร้างอ็อบเจ็กต์ หรือไม่ได้กำหนด เทเบิลสเปรซ ออราเคิล ดาต้าเบส จะทำการเก็บอ็อบเจ็กต์ เหล่านี้ในค่าเริ่มต้นของ เทเบิลสเปรซ ของแต่ละผู้ใช้ โดยที่ เทเบิลสเปรซ ของทุกผู้ใช้นั้นจะอยู่ใน ซิสเท็มเทเบิลสเปรซ แต่ผู้ใช้จะไม่สามารถที่จะสร้างอ็อบเจ็กต์ หรือมีสิทธิในการทำกับอ็อบเจ็กต์ ในขณะที่เป็น ค่าเริ่มต้นของเทเบิลสเปรซ หากผู้ใช้ต้องการที่จะสร้างอ็อบเจ็กต์ ชนิดใด ผู้ใช้ควร ที่จะกำหนด ค่าเริ่มต้นของเทเบิลสเปรซ ของแต่ละผู้ใช้ให้เป็นเฉพาะคือยูสเซอร์ เทเบิลสเปรซ

และคุณสามารถที่จะใช้คำสั่ง CREATE เทเบิลสเปรซ ในการสร้าง เทเบิลสเปรซ และหากต้องการเปลี่ยนแปลงก็ใช้คำสั่ง ALTER เทเบิลสเปรซ ได้ และสามารถที่จะทำพร้อมกับการ CREATE USER ตัวอย่างเช่น

```
CREATE USER jward
IDENTIFIED BY password
DEFAULT เทเบิลสเปรซ data_ts
QUOTA 100M ON test_ts
QUOTA 500K ON data_ts
TEMPORARY เทเบิลสเปรซ temp_ts
PROFILE clerk;
```

นอกจากนี้แล้วเราควรที่จะทำการกำหนดโควต้า ของผู้ใช้ที่จะใช้ในแต่ละ เทเบิลสเปรซ ด้วย โดย default แล้วผู้ใช้จะไม่มี โควต้าเทเบิลสเปรซ ในฐานข้อมูล หากผู้ใช้เหล่านั้นมีสิทธิในการสร้างอ็อบเจ็กต์ คุณต้องมีการกำหนด โควต้าเทเบิลสเปรซ ให้กับผู้ใช้นั้นด้วย เช่น ในตัวอย่างข้างต้นในตัวหนังสือสีแดง และคุณสามารถที่จะทำการ ยกเลิกสิทธิในการสร้าง อ็อบเจ็กต์ ของผู้ใช้ได้โดยการใช้คำสั่ง ALTER USER โดยการเปลี่ยนโควต้าเทเบิลสเปรซ ขอผู้ใช้ให้เท่ากับ 0 โดยที่อ็อบเจ็กต์ เดิมยังคงเหลืออยู่ แต่ไม่สามารถที่จะทำการสร้าง อ็อบเจ็กต์ ใหม่ได้

นอกจากนี้แล้วคุณยังสามารถที่จะมอบสิทธิให้กับผู้ใช้ ในการมีสิทธิในเทเบิลสเปรซแบบ อันลิมิตเต็ด เทเบิลสเปรซ ได้ โดยที่ผู้ใช้สามารถที่จะใช้โควต้าเทเบิลสเปรซ ได้โดยไม่มีการจำกัด ด้วยแต่จะมีข้อดีข้อเสียอยู่ ข้อดีคือ คุณสามารถที่จะให้สิทธิให้ผู้ใช้สามารถที่จะเข้าใช้งานฐานข้อมูลได้ด้วยคำสั่งเดียว ข้อเสียคือ คุณไม่สามารถที่จะเลือก Revoke เทเบิลสเปรซ ของผู้ใช้ที่เป็นแบบ อันลิมิตเต็ดเทเบิลสเปรซ ได้ แต่คุณสามารถที่จะทำได้ก็ต่อเมื่อคุณทำการยกเลิกสิทธิ ของผู้ใช้นั้น

หลังจากที่เราทำการกำหนด โควต้า เทเบิลสเปรซ แล้วนั้น ต่อไปเราจะทำการกำหนดเทมโพรารีเทเบิลสเปรซ ของผู้ใช้ซึ่งเราควรที่จะกำหนดเทมโพรารีเพื่อที่จะทำการเก็บคำสั่งเอสคิวแอลของผู้ใช้ที่ทำการเอ็กคิว โดยที่เราสามารถที่จะสร้างเทมโพรารีโดยใช้คำสั่ง CREATE TEMPORATY TABLESPACE ได้ แต่หากไม่มีการระบุเทมโพรารีให้กับผู้ใช้ที่แน่นอนได้ ออราเคิล ดาต้าเบส จะทำการกำหนดให้เองและจะเปลี่ยนแปลงได้โดยใช้คำสั่ง ALTER DATABASE หรือจะทำการกำหนดต้องที่ CREATE TABLE ก็ได้เช่น

```
CREATE USER jward
IDENTIFIED BY password
DEFAULT tablespace data_ts
QUOTA 100M ON test_ts
QUOTA 500K ON data_ts
TEMPORARY tablespace temp_ts
PROFILE clerk;
```

และเรายังต้องทำการกำหนด โปรไฟล์ (PROFILE) ให้กับผู้ใช้ในตอนที่เรา สร้างตารางด้วย ซึ่งหากเราไม่ทำการกำหนดโปรไฟล์ ให้กับ ยูสเซอร์ออราเคิล ดาต้าเบส จะทำการกำหนดให้เอง จากตัวอย่างข้างบนเป็นการกำหนดโปรไฟล์ ให้กับผู้ใช้ชื่อ jward เป็น โปรไฟล์ clerk;

จากนั้นจึงทำการตั้งค่า Role ของผู้ใช้ซึ่งเราไม่สามารถที่จะทำการกำหนดในตอน สร้างตาราง ได้ เราต้องใช้คำสั่งในการตั้งค่า คือ

GRANT USER jward clerk_role; เป็นการกำหนด Role ให้กับผู้ใช้

ALTER USER jward DEFAULT ROLE clerk_role; เป็นการเปลี่ยนแปลง Role ของผู้ใช้ ผู้ใช้จะได้รับสิทธิต่างๆ หลังจากการ GRANT USER jward clerk_role; เท่านั้น

และเราสามารถที่จะทำการเปลี่ยนแปลงสิทธิหรืออะไรต่างๆ ของผู้ใช้ได้แต่เราต้องมีสิทธิในการใช้คำสั่ง ALTER USER ด้วยแต่การทำเช่นนี้ควรมีเพียงผู้ดูแลฐานข้อมูลเท่านั้นที่สามารถทำได้ตัวอย่างเช่น

```
ALTER USER avyrros
IDENTIFIED EXTERNALLY
DEFAULT เทเบิลสเปรซ data_ts
TEMPORARY เทเบิลสเปรซ temp_ts
QUOTA 100M ON data_ts
QUOTA 0 ON test_ts
PROFILE clerk;
```

และเราสามารถที่จะทำการเปลี่ยนรหัสของผู้ใช้ได้โดยการใช้คำสั่ง SQL PASSWORD ได้เช่น

```
PASSWORD andy
Changing password for andy
New password: password
Retype new password: password
```

แต่ผู้ใช้จะไม่มีสิทธิในการติดต่อกับฐานข้อมูลและสร้างเซสชัน และสนับสนุนให้มีการเปลี่ยนแปลงรหัสของผู้ใช้บ่อยๆเพื่อจะได้มีความปลอดภัยของรหัสแต่เราจะมีอีกวิธีหนึ่งในการเปลี่ยนแปลง รหัสของผู้ใช้คือ

ALTER USER andy

IDENTIFIED BY *password*

ในการเปลี่ยนแปลงนี้เราต้องให้ผู้ดูแลฐานข้อมูลเป็นคนเปลี่ยนให้ ดังนั้นเราควรที่จะให้ผู้ดูแลฐานข้อมูล ใช้คำสั่ง PASSWORD เพราะเวลาเราใส่ผู้ดูแลฐานข้อมูลนั้น จะไม่มีการแสดงออกมาบนหน้าจอ แต่หากใช้คำสั่ง ALTER USER นั้น จะแสดงผู้ดูแลฐานข้อมูลใหม่ที่เราทำการเปลี่ยนนั้นออกมา

3.14.2 การพิสูจน์ตัวตนในการเข้าใช้งานฐานข้อมูล

การพิสูจน์ตัวตนเป็นความปลอดภัยอีกอย่างหนึ่งที่จะช่วยให้เราจัดการกับการเข้าใช้งานฐานข้อมูล ซึ่งผู้ที่สามารถที่จะเข้าใช้งานฐานข้อมูลได้นั้น ต้องเป็นผู้ใช้ที่มีอยู่ในฐานข้อมูลเท่านั้น และสามารถที่จะมีสิทธิการใช้งานข้อมูลได้ตามสิทธิที่ถูกกำหนดให้ต้อง CREATE USER เท่านั้น

ออราเคิล ดาต้าเบส มีฟังชันก์ในการป้องกันรหัสการเข้าใช้งานดังนี้คือ

- Password encryption คือการห่อหุ้มรหัสของการส่งจาก ไคลเอนมายังเซิร์ฟเวอร์ และเซิร์ฟเวอร์ไปยังเซิร์ฟเวอร์ ก่อนที่จะทำการส่งไปในเน็ดเวิค นั้นๆ
- Password complexity checking คือการป้องกันการบุกรุกของผู้ที่ไม่หวังดีต่อระบบโดยการที่คาดเดารหัสเพื่อล็อกอินเข้าสู้ระบบ
- Preventing passwords from being broken คือการที่มีผู้ใช้ทำการ Login ด้วย รหัสที่ไม่ถูกต้อง ออราเคิล ดาต้าเบส จะทำการ delays หลังจากการ Login ผิดครั้งที่ 3 เพื่อป้องกันการใช้ IP Address หรือ เครื่อง client ที่ต่างกันในการพยายาม Login เข้าสู้ระบบ ซึ่งการ Delay จะเพิ่มขึ้นที่ละนิดแต่จะไม่เกิน 10 วินาที
- Enforced case sensitivity for passwords คือการที่มีการตรวจรหัสที่เป็นทั้งตัวเล็กและตัวใหญ่ โดยจะต้องเช็คให้ถูกทุกตัว
- Passwords hashed using the Secure Hash Algorithm (SHA) cryptographic hash function SHA-1 คือการพิสูจน์ตัวตนของผู้ใช้โดยการตรวจสอบรหัสซึ่งอยู่ภายใต้การตรวจสอบรหัสที่เป็นตัวเล็กตัวใหญ่และจำนวนของรหัสที่มีได้ 160 bit ซึ่งการใช้ SHA-1 จะทำให้ได้ความปลอดภัยที่ดีกว่า

3.14.3 หลักในการตั้งรหัสผ่าน

ในการตั้งรหัสนั้นจะมีตัวอักษรไม่เกิน 30 ตัว และเราสามารถที่จะใช้คำสั่งเอสคิวเอล ในตอนที่

CREATE USER หรือ ALTER USER เช่น

CREATE USER psmith IDENTIFIED BY *password*;

GRANT CREATE SESSION TO psmith IDENTIFIED BY *password*;

CREATE DATABASE USER psmith IDENTIFIED BY *password*;

CREATE DATABASE LINK AUTHENTICATED BY psmith IDENTIFIED BY *password*;

เวลาเราสร้าง ดาต้าเบส ใน ออราเคิล ดาต้าเบส 11จี ส่วนมากแล้วจะถูก ล๊อครหัสถ้าเราจะทำการเปลี่ยนแปลง เราจะมีแอคเคาน์ ที่มีรหัสเป็นค่าเริ่มต้นที่มีมาตอนที่เราสร้างฐานข้อมูลด้วยคือ HR, OE, และ SCOTT

เพื่อทำให้มีความปลอดภัยดีขึ้น จึงควรที่จะเปลี่ยนรหัสเพราะถ้าเป็นค่าเริ่มต้น นั้นจะทำให้ง่ายต่อการบุกรุกของผู้ที่ไม่ประสงค์ดีและหากต้องการหาว่าแอคเคาน์ ใดที่เป็นล็อกหรือ อันล็อกที่ใช้รหัสที่เป็นออราเคิลกำหนดให้สามารถดูได้โดยการใช้คำสั่ง เอสคิวแอลดังนี้คือ

```
CONNECT / AS SYSDBA
Enter password: password
Connected.
SELECT d.username, u.account_status
FROM DBA_USERS_WITH_DEFPWD d, dba_users u
WHERE d.username = u.username
ORDER BY 2,1;
```

ผลของคำสั่งที่ได้

```
USERNAME  ACCOUNT_STATUS
--------- ------------------------------
SCOTT     EXPIRED & LOCKED
```

และเราสามารถที่จะทำการเปลี่ยนแปลงค่า ค่าเริ่มต้นของ พารามิเตอร์ ใน โปรไฟล์ได้ ซึ่งค่าพารามิเตอร์ต่างมีดังนี้คือ

ตารางที่ 3.7 ออฟชันที่ใช้ในการกำหนดความปลอดภัยในโปรไฟล์

Parameter	Default Setting	Description
FAILED_LOGIN_ATTEMPTS	10	การ ตั้ง ค่าสูงสุดของเวลาที่ ยูสเซอร์ พยายามที่จะล็อกอิน เข้าสู้ระบบและเกิดการผิดพลาดก่อนที่จะทำการล็อกก่อน
PASSWORD_GRACE_TIME	7	การตั้งค่า จำนวนวันที่ยูสเซอร์ทำการเปลี่ยนรหัสใหม่ ก่อนที่จะหมดอายุ
PASSWORD_LIFE_TIME	180	การ ตั้งค่า จำนวนวันที่สามารถใช้งานรหัสได้
PASSWORD_LOCK_TIME	1	การ ตั้งค่า จำนวนวันที่จะสามารถทำการล็อกอิน ได้อีกครั้งหลังจากการโดนล็อกเนื่องจากการ ล็อกอินผิดพลาดเกิน 3 ครั้ง
PASSWORD_REUSE_MAX	UNLIMITED	การตั้งค่า จำนวนวันก่อนที่รหัสจะถูกกลับมาใช้ได้อีกครั้ง
PASSWORD_REUSE_TIME	UNLIMITED	การ ตั้งค่า จำนวนเวลาที่จะเปลี่ยนรหัสก่อนที่รหัสที่ถูกต้องจะถูกนำกลับมาใช้งาน

ซึ่งเราสามารถที่จะทำการ ตั้งค่าต่างๆ เหล่านี้ได้ โดยการใช้คำสั่งเอสคิวแอล ดังนี้

```
ALTER PROFILE prof
FAILED_LOGIN_ATTEMPTS 10
PASSWORD_LOCK_TIME 1;
```

หากผู้ใช้พยายามที่จะล็อกอิน เข้าสู้ระบบโดยที่ใส่ รหัสที่ผิดนั้น ออราเคิล ดาต้าเบส จะทำการ ล็อกรหัสอัตโนมัติ ซึ่งถ้าต้องการ ปลดล็อกต้องรอจนกว่าจะครบกำหนดการ ปลดล็อกของค่าที่ต้องไว้หรือไม่ก็ไปให้ผู้ดูแลฐานข้อมูลเป็นคนปลดล็อกให้ และหากต้องการกำหนดค่าของพารามิเตอร์ ต่างนั้นสามารถที่จะทำได้คือ

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 10
 PASSWORD_LOCK_TIME 30;
ALTER USER johndoe PROFILE prof;
ALTER USER johndoe ACCOUNT UNLOCK;
ALTER USER susan ACCOUNT LOCK;
```

และการที่ป้องกันไม่ให้ผู้ใช้นำรหัสเก่ากลับมาใช้ได้อีกนั้นเราสามารถที่จะทำการกำหนดโดยการ ตั้งค่าพารามิเตอร์ 2 ตัวคือ PASSWORD_REUSE_TIME แล PASSWORD_REUSE_MAX

ตารางที่ 3.8 แสดงออฟชันในการในรหัสผ่านเก่ามาใช้งานอีกครั้ง

Parameter Name	**Description and Use**
PASSWORD_REUSE_TIME	การกำหนดจำนวนวันจากการใช้งานครั้งล่าสุดก่อนที่จะให้ใช้รหัสไม่ทำการกำหนด ไม่จำกัด
PASSWORD_REUSE_MAX	การกำหนดจำนวนการ ตั้งค่ารหัสใหม่ ก่อนที่จะทำการนำมาใช้ใหม่ ไม่ทำการกำหนดเป็นการทำงานที่ไม่จำกัด

ในการกำหนดค่าของ พารามิเตอร์ 2 ตัวนี้หากเราทำการกำหนดค่าที่ไม่ใช่ อันลิมิเต็ดให้กับพารามิเตอร์ ตัวใดตัวหนึ่งแล้ว ผู้ใช้จะไม่มีทางที่จะใช้รหัสเก่าได้ แต่หากเราทำการกำหนดให้กับทั้งคู่เป็นอันลิมิตเต็ด จะทำให้ผู้ใช้สามารถที่จะนำรหัสเก่ากลับมาใช้เมื่อไรก็ได้

การกำหนด เวลาการใช้งานของรหัสนั้นเราสามารถทำการกำหนดได้ดังนี้คือ

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 4
 PASSWORD_LOCK_TIME 30
 PASSWORD_LIFE_TIME 180;
ALTER USER johndoe PROFILE prof;
```

คือการกำหนดว่าผู้ใช้ชื่อ johndoe สามารถใช้งานรหัสได้ 180 วัน และก่อนที่รหัสใกล้จะหมดอายุการใช้งานนั้นทุกครั้งที่ ยูสเซอร์ล็อกอิน จะมีข้อความเตือนให้ผู้ใช้ทำการเปลี่ยนรหัสแต่หาก ผู้ใช้ไม่ทำการเปลี่ยนรหัสใหม่ภายในเวลาที่มีข้อความเตือน ออราเคิล ดาต้าเบส จะทำการขึ้นเตือนมาให้ใส่รหัสใหม่ทุกครั้งและจะเข้าใช้งานได้จนกว่าจะใส่รหัสใหม่ตัวอย่างเช่น

```
CREATE PROFILE prof LIMIT
FAILED_LOGIN_ATTEMPTS 4
PASSWORD_LOCK_TIME 30
PASSWORD_LIFE_TIME 90
PASSWORD_GRACE_TIME 3;
ALTER USER johndoe PROFILE prof;
```

คือการกำหนดวันที่ผู้ใช้จะได้รับข้อความเตือนหลังจากหมดอายุการใช้งานของรหัสแล้ว

นอกจากนี้ ออราเคิล ดาต้าเบส ยังมีการเช็คความซับซ้อนของรหัสที่ผู้ใช้ทำการตั้งไว้เพื่อการล็อกอินเข้าสู่ระบบด้วย โดยจะมี PL/SQL script UTLPWDMG.SQL เป็นตัวทำการตรวจสอบเวลาที่ ยูสเซอร์ทำการตั้งรหัสหรือการเปลี่ยนรหัสซึ่งจะทำการตรวจสอบดังนี้คือ

- รหัสจะต้องมีตัวอักษรอย่างน้อย 8 ตัวขึ้นไปและจะไม่เกิน 30 ตัว
- รหัสจะต้องไม่เหมือนกับ ยูสเซอร์เนม และจะไม่มีตัวเลขต่อท้าย
- รหัสจะต้องไม่เหมือนกันเซิร์ฟเวอร์เนม และจะไม่มีตัวเลขต่อท้ายตัวอย่างรหัสที่ไม่ควรตั้ง เช่น welcome1, database1, account1, user1234, password1, oracle, oracle123, computer1, abcdefg1, or change_on_install
- รหัสต้องประกอบไปด้วยตัวเลขอย่างน้อย 1 ตัวและตัวอักษร 1 ตัว
- รหัสต้องแตกต่างจากกันกับตัวก่อนหน้าอย่างน้อย 3 ตัว

3.14.4 ข้อได้เปรียบของการใช้ ดาต้าเบสการยืนยันตัว

3.14.4.1 ผู้ใช้และทุกการ ยืนยันตัวจะถูกควบคุมโดยฐานข้อมูล ซึ่งฐานข้อมูลจะไม่มีการเชื่อถืออะไรจากภายนอกของฐานข้อมูลที่ไม่มีในฐานข้อมูล

3.14.4.2 ออราเคิล ดาต้าเบส จะมีวิธีการที่จะจัดการกับรหัสเพื่อที่จะทำให้เกิดความปลอดภัยมากยิ่งขึ้น

3.14.4.3 ทำให้ผู้ดูแลฐานข้อมูลง่ายในการจัดการในเวลาที่มี ยูสเซอร์ติดต่อเข้ามาน้อย

3.15 Configuring Auditing

Auditing คือ การตรวจสอบและการบันทึกการกระทำของผู้ใช้กับฐานข้อมูล เราสามารถที่จะตรวจสอบได้เป็นรายบุคคล เช่น การใช้คำสั่ง เอสคิวแอลหรือ การที่รวมทั้ง ยูสเซอร์เนม, โปรแกรม, เวลา เป็นต้น ดังนั้นเราสามารถที่จะสร้างความปลอดภัยโดยการใช้ trigger auditing เวลาที่เข้าใช้งานหรือการเปลี่ยนแปลงของอ็อบเจ็กต์ ในฐานข้อมูล

3.15.1 ทำไมต้องมีการ ออดิต (Auditing)

3.15.1.1 เพื่อทำการบันทึกการกระทำต่อสกรีม่า, ตาราง, หรือ row

3.15.1.2 ทำการป้องกันผู้ใช้ที่พยายามทำในสิ่งที่ตนไม่มีสิทธิหรือไม่หวังดี

3.15.1.3 เพื่อทำการสืบสวนการกระทำที่น่าสงสัย เช่น ถ้าผู้ใช้ทำการลบ ข้อมูลจาก ตารางดังนั้น ผู้ดูแลฐานข้อมูลต้องทำการออดิต ทุกการเชื่อมต่อเข้ามาในฐานข้อมูลและการ delete row จากตารางในฐานข้อมูลทั้งที่ทำสำเร็จและทำไม่สำเร็จ

3.15.1.4 การทำการแจ้งเกี่ยวกับการกระทำที่ผู้ใช้ไม่มีสิทธิ เช่น ผู้ใช้ไม่มีสิทธิ ที่จะเปลี่ยนหรือลบข้อมูล หรือผู้ใช้น่าจะมีสิทธิที่ผู้ใช้สมควรที่จะได้รับ ซึ่งสามารถที่จะให้เราทำการให้สิทธิกับผู้ใช้ให้ถูกต้องและสมควร

3.15.1.5 การรวบรวมและตรวจสอบข้อมูลเกี่ยวกับการกระทำที่มีกับฐานข้อมูล เช่น การที่ผู้ดูแลฐานข้อมูลทำการเก็บข้อมูลสถิติเกี่ยวกับการ แก้ไขตาราง, จำนวนของ logical I/O ที่มีการทำงานหรือเวลาที่มีจำนวนผู้ใช้มีการใช้งานสูงสุด

3.15.1.6 การตรวจสอบปัญหาโดยการ ยืนยันตัวหรือ แอคเซส คอลโทรลเช่น เราสามารถที่จะสร้างวิธีการ ออดิต ซึ่งนั้นเราจะได้แน่ใจว่าจะไม่เกิดการบันทึกออดิต เพราะข้อมูลจะถูกป้องกันคนอื่น อย่างไรก็ตาม ถ้าวิธีการบันทึกออดิตในขณะนั้นเราจะรู้ถึงการควบคุมความปลอดภัยที่ไม่ควร

เมื่อเราทำการออกแบบออดิตที่จะ focused หรือ broad เมื่อทำการ enabling audit ของเราจะปฏิบัติดังนี้

- คำสั่งที่ประมวลผล สำเร็จ, คำสั่งที่ ประมวลผลไม่สำเร็จ หรือทั้ง 2 กรณี
- คำสั่งของแต่ละผู้ใช้ในเซสชัน หรือ เวลาหนึ่งที่คำสั่งประมวลผล
- การทำงานของทุกผู้ใช้หรือแต่ละผู้ใช้

3.15.2 ชนิดของออดิต

3.15.2.1 Auditing SQL Statements

คือการตรวจสอบทุกชนิดของคำสั่งแอลคิวแอล คำสั่งที่ ออดิตทำการตรวจสอบแต่ละชนิดที่เกี่ยวข้องกับการกระทำของแต่ละอ็อปชัน เช่น AUDIT TABLE การคอยติดตามแต่ละคำสั่ง DDL ที่ไม่ถูกสนใจจาก table เราสามารถที่จะกำหนดคำสั่งของผู้ใช้ที่จะ ออดิต หรือทุกผู้ใช้ในฐานข้อมูลก็ได้

3.15.2.2 Auditing Privileges

คือการที่ audit ทำการตรวจสอบเกี่ยวกับ ซิสเท็มพริวิเลจ ที่ซึ่งใช้ตรงกับ การกระทำ เช่น AUDIT CREATE TABLE ซึ่งพริวเลจ ออดิตจะมีการ focused มากกว่า สเตทเมนออดิต โดยที่จะทำการออดิต แต่ละชนิดของการกระทำ เราสามารถที่จะกำหนด พริวิเลจออดิตติ้ง ที่จะทำการ ออดิตในแต่ละผู้ใช้หรือทุกผู้ใช้ในฐานข้อมูลก็ได้

3.15.2.3 Auditing Schema Objects

คือการระบุคำสั่งที่จะออดิต ในแต่ละ สกีม่าอ็อบเจ็คต์เช่น AUDIT SELECT ON employee จะเห็นได้ว่า สกีม่าอ็อบเจ็คออดิตติ้ง จะระบุแต่ละชนิดของคำสั่งบนแต่ละ สกีม่า อ็อบเจค เช่น SELECT และจะถูกในไปใช้กับทุกผู้ใช้ในฐานข้อมูล

3.15.2.4 Auditing SQL Statements and Privileges in a MultitierEnvironment

คือการเอา ออดิต ไปใช้ประโยชน์ในการเป็นตัวกลางของโปรแกรม ในการตรวจสอบจากไคลเอน

3.15.2.5 Auditing Network Activity

คือการออดิต ข้อผิดพลาดที่ไม่ได้คาดคิดใน เน็ตเวิค โปรโตคอลหรือข้อผิดพลาดภายในเน็ตเวิคเลเยอร์

3.15.2.6 Using Fine-Grained Auditing to Monitor Specitfic Activities

คือการออดิต ระดับที่ ละเอียดมากที่สุด การเข้าถึงข้อมูลและการกระทำที่อยู่บนคอนเทนใช้ บูลีนเป็นตัววัด เช่น value > 1,000,000 เป็นการเข้าถึงหรือเปลี่ยนแปลงใน คอลั่ม

3.15.3 การสร้างการบันทึกการกระทำของ Audit

เมื่อมีการบันทึกการกระทำต่างๆที่ผู้ใช้กระทำนั้น สิ่งที่บันทึกจะไปเก็บดาต้าดิกชันนารี่เทเบิลเรียกว่า ดาต้าเบสออดิเทรล (Database audit trail) หรือเอาไปเก็บไว้ในไฟล์ของ โอเปอร์เรชันซิสเท็ม เรียกว่า โอเปอร์เรชัน ซิสเท็ม ออดิตเทรล (Operation System audit trail)
โดยทั่วไป auditing แบ่งเป็น 3 ชนิดคือ

3.15.3.1 Standard auditing จะใช้กับการตรวจสอบกับคำสั่ง เอสคิวแอล, พริวเลจ ,สกีม่า,อ็อบเจ็คต์,เน็ตเวิต และ มัลติทรี่ แอคติวิตี้

3.15.3.2 Fine-grained auditing จะใช้กับ fine-grained ที่จะดูในการกระทำที่น่าสงสัย เช่น การกระทำบนฐานข้อมูลหรือเวลาที่มีการกระทำ เป็นต้น

3.15.3.3 Administrator auditing จะใช้กับ administrator ว่ามีการทำอะไรของ administrator บ้างในฐานข้อมูล

3.15.4 การบริหารจัดการ ดาต้าเบส Audti Trail

เมื่อมีการบันทึกการกระทำต่างๆ ที่เกิดขึ้นในฐานข้อมูล ข้อมูลที่ทำการบันทึกจะนำไปเก็บในตาราง SYS.AUD$ และ SYS.FGA_LOG$ ในโครงสร้างของ SYS แต่ละ ดาต้าเบส ดาต้า ดิกชันนารี่ โดยจะทำการ ออดิต ทั้งสเตดาร์ดและ fine-grained ซึ่งเราสามารถที่จะดูข้อมูลเพิ่มในของแต่ละตารางได้ เช่น DBA_AUDIT_TRAIL และความต่างของออดิต ระหว่าง ดาต้าเบส ออดิตเทรล กับโอปอเรชัน ซิสเท็ม ออดิต เทรล จะแสดงดังในตางรางนี้

ตารางที่ 3.9 ความต่างของออดิตระหว่าง ดาต้าเบสออดิตเทรลกับโอปอเรชัน ซิสเท็ม ออดิต เทรล

Data Populated in Database Audit Trail	In Operating System Audit Trail?
(*) Bind values used for the SQL statement, if any	Footnote 1
(*) SQL text (the SQL text that triggered the auditing)	Footnote 1
Completion code of the operation	Yes
Database user name (DATABASE USER)	Yes
Date and time stamp in UTC (Coordinated Universal Time) format	No
Distinguished name	Yes
Global User unique ID	No
Instance number	No
Name of the schema object accessed	Yes
Operating system login user name (CLIENT USER)	Yes
Operation performed or attempted (ACTION)	Yes
Process number	Footnote 2

ตารางที่ 3.9 ความต่างของออดิตระหว่าง ดาต้าเบส ออดิตเทรล กับ โอปอเรชัน ซิสเท็ม ออดิต เทรล(ต่อ)

Data Populated in Database Audit Trail	In Operating System Audit Trail?
Proxy Session audit ID	No
SCN (system change number) for the SQL statements	No
Session identifier	Yes
System privileges used (PRIVILEGE)	Yes
Terminal identifier	Yes
ทรานแซกชัน ID	No

วิธีการเปลี่ยน ออดิตติ่ให้ไปเก็บในตาราง SYS.AUD$ ทำได้ดังต่อไปนี้

3.15.4.1 สร้างผู้ใช้ที่เป็นของ Tutorial

a) ล๊อกอิน ที่เป็น ยูสเซอร์ SYS โดยมีสิทธิเป็น AS SYSDBA

```
sqlplus "SYS/AS SYSDBA"
Enter password: password
Connected.
```

b) สร้างผู้ใช้

```
GRANT CREATE SESSION TO smith IDENTIFIED BY password;
```

c) มอบสิทธิให้กับผู้ใช้ชื่อ smith

```
GRANT SELECT, INSERT, UPDATE, DELETE ON AUD$ TO smith;
GRANT SELECT ON DBA_AUDIT_TRAIL TO smith;
```

d) ทำการพิมพ์คอมมานตามลำดับข้างล่างนี้

```
col username format a10
col action_name format a13
```

```
col owner format a7
col obj_name format a10
```

3.15.4.2 ทำการเปิด ออดิต และ TRUNCATE ให้กับตาราง SYS.AUD$ โดยใช้คำสั่งดังนี้

```
AUDIT SELECT ON AUD$ BY ACCESS;
TRUNCATE TABLE AUD$;
```

3.15.4.3 ออดิต การปฏิบัติและการกระทำของผู้ใช้

a) ติดต่อโดยผู้ใช้ชื่อ smith

```
CONNECT smith
Enter password: password
Connected.
```

b) ใช้คำสั่งดังนี้

```
SELECT COUNT(*) FROM SYS.AUD$;
COUNT(*)
---------
1
```

c) ใช้คำสั่ง SELECT ดังนี้

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME FROM
        DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);


USERNAME   ACTION_NAME   OWNER   OBJ_NAME
---------- ------------ ------- ----------
SMITH      SELECT       SYS     AUD$
```

ซึ่งเป็นการแสดงให้เห็นการใช้คำสั่ง SELECT ของผู้ใช้ชื่อ smith บน DBA_AUDIT _TRAIL ซึ่งจะแสดงสิ่งที่ตาราง SYS.AUD$ เก็บอยู่

d) ทำการใช้คำสั่ง UPDATE บนตาราง SYS.AUD$ ดังนี้

UPDATE SYS.AUD$ SET USERID = 0;

3 rows updated.

e) เมื่อเราทำซ้ำในตอนที่ SELECT ที่ตอนที่ผ่านมาแล้วจะได้ผลลัพธ์ที่เปลี่ยนไปดังนี้

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME
FROM  DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);
USERNAME   ACTION_NAME   OWNER   OBJ_NAME
---------- ------------- ------- ----------
0          SELECT        SYS     AUD$
0          SELECT        SYS     AUD$
SMITH      UPDATE        SYS     AUD$
```

เราจะเห็นว่าตาราง SYS.AUD$ จะทำการบันทึกการกระทำของผู้ใช้ชื่อ smith

f) ทำการ Delete rows จากตาราง SYS.AUD$

DELETE FROM SYS.AUD$;

4 rows deleted.

g) แล้วทำการ SELECT ซ้ำอีกรอบจะได้ผลที่เปลี่ยนไปดังนี้

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME
FROM  DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);

USERNAME   ACTION_NAME   OWNER   OBJ_NAME
```

```
---------- ------------- ------- ----------
SMITH      UPDATE        SYS     AUD$
SMITH      DELETE        SYS     AUD$
```

3.15.4.4 ทำการเอาส่วนประกอบต่างของ Tutorial ออก

a) ติดต่อโดยผู้ใช้ชื่อ SYS AS SYSDBA

```
CONNECT SYS/AS SYSDBA
Enter password: password
```

b) ทำการ Remove auditing ออกจากตาราง SYS.AUD$

```
NOAUDIT SELECT, INSERT, UPDATE, DELETE ON AUD$;
```

c) Drop user smith

```
DROP USER smith;
```

3.15.5 การใช้ Default Audting for Security-Relevant SQL Statement and Privileges

เวลาที่เราสร้างฐานข้อมูลขึ้นมาใหม่หรือมีการปรับเปลี่ยนฐานข้อมูลเก่า เราสามารถที่ใช้ DBCA ในการเปิดหรือปิด default security setting ซึ่ง ออราเคิล จะเปิดใช้ให้เป็น default เวลาเราเปิด default security setting Oracle audits จะทำการ audit security-relevant SQL statements and privileges และจะทำการกำหนด AUDIT_TRAIL ให้เป็น DB ด้วย Oracle Database audits คำสั่ง AUDIT ROLE SQL ที่เป็น Default ดังนี้

ALTER ANY PROCEDURE	CREATE ANY JOB	DROP ANY TABLE
ALTER ANY TABLE	CREATE ANY LIBRARY	DROP PROFILE
ALTER DATABASE	CREATE ANY PROCEDURE	DROP USER
ALTER PROFILE	CREATE ANY TABLE	EXEMPT ACCESS POLICY
AUDIT ROLE BY ACCESS	CREATE EXTERNAL JOB	GRANT ANY OBJECT PRIVILEGE

ALTER SYSTEM	CREATE PUBLIC DATABASE LINK	GRANT ANY PRIVILEGE
ALTER USER	CREATE SESSION	GRANT ANY ROLE
AUDIT SYSTEM	CREATE USER	
AUDIT SYSTEM BY ACCESS	DROP ANY PROCEDURE	

ออราเคิล ดาต้าเบส ทำการ audit ทุก Privilegs และทุกคำสั่งที่อยู่ข้างบนหากเราใช้คำสั่ง BY ACCESS

หากเรารู้สึกว่าการทำ ออดิตจะมีผลกระทบต่อโปรแกรม ของเรา เราก็สามารถที่จะทำการปิดการออดิต ได้โดยใช้ DBCA และสิ่งหนึ่งที่ควรจะรู้ไว้อีกอย่างคือการออดิตจะมีผลต่อประสิทธิภาพของระบบเราด้วย

3.16 การใช้ สเตนดาร์ด ออดิตติ้ง เพื่อทำการตรวจสอบการกระทำทั่วๆ ไป

หากเราทำการเปิดใช้ สเตนดาร์ด ออดิตติ่ง แล้ว เวลาทำงานจริงๆ จะทำการ ออดิต คำสั่ง เอสคิวเอล, สกีม่า อ็อบเจ็ค และเน็ตเวิค โดยที่เราจะใช้คำสั่ง เอสคิวแอล คือ ออดิต ในการเปิดการออดิต และ โนออดิต ในการปิดการ ออดิต หรืออาจจะใช้ Enterprise Manager Database Control ก็ได้

3.16.1 การเปิดหรือปิดการทำงาน ของ สเตนดาร์ดออดิตเทรล

การอื่นต้องทำการกำหนดตัวแปร AUDIT_TRAIL ก่อนและต้องทำโดยผู้ที่เป็นผู้ดูแลฐานข้อมูล การดูว่าค่าของตัวแปร AUDIT_TRAIL นั้นมีค่าเป็นอะไรใช้คำสั่งดังนี้

```
SHOW PARAMETERS AUDIT_TRAIL

NAME                 TYPE      VALUE
-------------------------------- ----------- -------
audit_trail          string    DB
```

และในการเปิดการใช้งานสเตนดาร์ดดังนี้

```
sqlplus "SYS/AS SYSDBA"
Enter password: password
Connected.
SQL> ALTER SYSTEM SET AUDIT_TRAIL=DB, EXTENDED SCOPE=SPFILE;
System altered.
SQL> CONNECT SYS/AS SYSOPER
Enter password: password
Connected.
SQL> SHUTDOWN;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP;
ORACLE instance started.
```

ตัวอย่างเช่น

```
AUDIT SELECT TABLE, UPDATE TABLE, DELETE TABLE
BY jward, jane;
```

ในทางกลับกันถ้าต้องการยกเลิกการ ออดิตก็ทำได้ดังนี้

```
NOAUDIT SELECT TABLE, UPDATE TABLE, DELETE TABLE
BY jward, jane;
```

และยังมี อ๊อบชันอื่นอีกดังนี้

BY SESSION/BY ACCESS

BY SESSION จะทำการออดิตในตอนแรกที่กระทำโดยตรงกับเซสชัน

BY ACCESS จะทำการออดิต ทุกๆ การกระทำและทุกๆ เวลาที่มีการกระทำด้วย

ตัวอย่าง เช่น

```
AUDIT SELECT TABLE, UPDATE TABLE, DELETE TABLE
BY SESSION;
```

และนอกจากนี้แล้วเรายังทำการออดิตในส่วนของคำสั่งต่างๆ ที่ทำสำเร็จและไม่สำเร็จได้ด้วยโดยใช้คำสั่งดังนี้คือ

WHENEVER SUCCESSFUL คือการทำการ ออดิตคำสั่งที่กระทำสำเร็จ

WHENEVER NOT SUCCESSFUL คือการทำการ ออดิตคำสั่งที่กระทำไม่สำเร็จ

ตัวอย่าง เช่น

```
AUDIT SELECT UPDATE TABLE, DELETE TABLE
WHENEVER NOT SUCCESSFUL;
```

และหลังจากที่เราทำการเปิดใช้งานสเตนดาร์ดออดิตเทรล นั้นบางครั้งเราอาจจะต้องการลบการบันทึกต่างๆ ที่เกิดขึ้นเพื่อทำการเคลียให้มีพื้นที่ว่าง เราสามารถที่จะทำการลบการบันทึกเหล่านี้ได้โดยใช้คำสั่งดังนี้

```
DELETE FROM SYS.AUD$;
```

หรือหากต้องการลบที่ระบุตารางก็ได้ดังนี้

```
DELETE FROM SYS.AUD$
WHERE obj$name='EMP';
```

3.17 การ auditing SQL Statements

ในการออดิต คำสั่งของเอสคิวแอล ที่เกี่ยวข้องกับกลุ่มของ ดาต้าเบส สทรักเจอร์หรือ สกีม่าอ็อบเจ็คต์ แต่จะไม่ระบุเฉพาะว่าเป็น ดาต้าเบส สทรักเจอร์หรือ สกีม่าอ็อบเจ็คต์ ไหน

ชนิดของคำสั่งของเอสคิวแอล มี 2 ประเภทคือ

- DDL สเตทเม้น ตัวอย่างเช่น AUDIT TABLE จะทำการออดิตทุกคำสั่ง CREATE และ DROP TABLE
- DMLสเตทเม้น ตัวอย่างเช่น AUDIT SELECT TABLE จะทำการออดิต ทุกคำสั่งที่ SELECT … FROM TABLE/VIEW

การเปิดใช้งานของ SQL Statement Auditing นั้นเราจะใช้คำสั่ง AUDIT ในการเปิดใช้งานแต่เรา จะต้องมีสิทธิ AUDIT SYSTEM จึงจะสามารถที่จะเปิดใช้งานได้ แต่สิทธินี้น่าจะทำได้เพียงคนเดียวคือผู้ที่เป็น Administrator ตัวอย่างเช่น

AUDIT DROP TABLE;

และถ้าหากเราทำการออดิตคำสั่งที่เป็น SESSION หรือ NOT EXISTS นั้นจะทำให้เกิดผลดังนี้คือ ถ้าหากเราทำการ ออดิต โดย SESSION จะไม่ทำการออดิตทุกๆ คำสั่ง แต่จะทำการออดิตในตอนที่ติดต่อเข้ามาตอนแรกและจะทำการสะสมคำสั่งที่มีการกระทำต่อฐานข้อมูลนั้นๆ จนกว่าจะทำการปิดการติดต่อ จากนั้นก็จะทำการเอาคำสั่งที่สะสมมาบันทึกอีกที และจะทำการออดิตทุกคำสั่งที่สำเร็จและไม่สำเร็จด้วย แต่ถ้าหากเป็นการทำ ออดิต โดย NOT EXISTS จะทำการออดิตทุกคำสั่งของเอสคิวแอล ที่ผิดเพราะว่าไม่มีอ็อบเจ็คต์ นั้นอยู่ ตัวอย่างเช่น

```
AUDIT SESSION;
AUDIT SESSION
BY jward, swilliams;
```

และ AUDIT NOT EXISTS;

และในส่วนของการปิดการ ออดิต ก็ทำได้โดยใช้คำสั่ง NOAUDIT ตัวอย่างเช่น

```
NOAUDIT session;
NOAUDIT session BY preston, sebastian;
NOAUDIT DELETE ANY TABLE;
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
EXECUTE PROCEDURE;
```

หรือหากต้องการที่จะทำการปิดทุกการ ออดิตก็สามารถที่จะใช้คำสั่งเดียวคือ

NOAUDIT ALL;

3.18 การออดิตติ้ง พริวิเล็จ

ในการทำการออดิตติ้ง พริวิเล็จ(audit privileges) นั้นเราจะเห็นว่าจะคล้ายกับการออดิตเอสคิวแอล สเตทเม้น แต่จะแตกต่างกันตรงที่ผู้ที่จะถูกทำการ ออดิต พริวิเล็จได้นั้นต้องมีสิทธิในการเป็นเจ้าของหรือสิทธิ ที่จะกระทำต่อ สกีม่า อ็อบเจ็คต์ นั้นๆ และอีกอย่างที่แตกต่างก็คือตอนที่

ทำการ ออดิตพริวิเลจ จะทำการออดิตเฉพาะคำสั่งนั้นจะไม่ทำการ ออดิต หมด เช่น CREATE TABLE จะทำการออดิตเฉพาะคำสั่ง CREATE TABLE เท่านั้น

ในการเปิดการ ออดิตติ่ง นั้นจะใช้คำสั่งที่เหมือนกับสิทธิต่างที่มีใน ซิสเท็มพริวิเลจเช่น DELETE ANY TABLE ก็จะใช้คำสั่ง DELETE ANY TABLE ตัวอย่างเช่น

```
AUDIT DELETE ANY TABLE;
```

และนอกจากนี้แล้วยังสามารถที่ใช้คำสั่งที่เป็น BY ACCESS, BY SESSION, WHENEVER SUCCESSFUL และ WHENEVER NOT SUCCESSFUL ได้อีกด้วยเช่น

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE, EXECUTE PROCEDURE
      BY ACCESS
      WHENEVER NOT SUCCESSFUL;
```

ในการปิดการออดิตติ่ง ก็ใช้คำสั่ง NOAUDTI ..... ออดิตเอสคิวแอลสเตดเม้น

3.19 การออดิตติ๊ง สกีม่า อ็อบเจ็คต์

การ ออดิตติ๊งสกีม่า อ็อบเจ็คต์ (Auditing Schema Objects)นั้นจะทำการ ออดิต ทุกๆ คำสั่งที่เป็น SELECT และ DML ที่มีสิทธิใน สกีม่าอ็อปเจ็คต์ นั้น เช่นคำสั่ง SELECT หรือ DELETE ที่ส่งไปยังตาราง

เราสามารถที่จะ ออดิต คำสั่งที่อ้างถึง ตาราง(tables),views,sequences,standalone stored procedures หรือฟังก์ชันและแพ็กเก็ต แต่จะไม่ใช่ procedures ที่อยู่ในแพ็กเก็ต และเราไม่สามารถที่จะทำการ ออดิตคำสั่งที่จะอ้างถึง คลัสเตอร์, ดาต้าเบสลิ้ง,อินเด็ก, หรือ synonyms อย่างไรก็ตามเราสามารถที่จะทำการเข้าถึง สกีม่าอ็อปเจ็ค ได้โดยการใช้คำสั่งที่อยู่บนตารางนั้นได้
ตารางที่แสดงการกระทำที่มีใน ออราเคิล ดาต้าเบส 11จี

ตารางที่ 3.10 การกระทำที่มีใน ออราเคิล ดาต้าเบส 11จี

Object or Element	**Auditable Action**
Mining Model	ALTER, AUDIT, COMMENT, GRANT, RENAME, SELECT
OLAP Primary Dimension	ALTER, AUDIT, DELETE, INSERT, SELECT, CREATE

ตารางที่ 3.10 การกระทำที่มีใน ออราเคิล ดาต้าเบส 11จี(ต่อ)

Object or Element	**Auditable Action**
OLAP Cube	ALTER, AUDIT, DELETE, SELECT, UPDATE, CREATE
OLAP Measure Folder	AUDIT, DELETE, INDEX, SELECT, CREATE
OLAP InterAction	AUDIT, UPDATE, CREATE
Edition	ALTER, AUDIT, COMMENT, GRANT

ตารางที่ 3.11 ตารางแสดงออฟชัน(Options) ที่มีใน ออราเคิล ดาต้าเบส 11จี

stem	Auditable Action
Edition	CREATE ANY EDITION, DROP ANY EDITION, ALTER ANY EDITION, COMMENT EDITION, GRANT EDITION, USE EDITION
Primary Dimension	CREATE PRIMARY DIMENSION, ALTER ANY PRIMARY DIMENSION, CREATE ANY PRIMARY DIMENSION, DELETE ANY PRIMARY DIMENSION, DROP ANY PRIMARY DIMENSION, INSERT ANY PRIMARY DIMENSION, SELECT ANY PRIMARY DIMENSION, UPDATE ANY PRIMARY DIMENSION
Cube	CREATE CUBE, ALTER ANY CUBE, CREATE ANY CUBE, DROP ANY CUBE, SELECT ANY CUBE, UPDATE ANY CUBE
Measure Folder	CREATE MEASURE FOLDER, CREATE ANY MEASURE FOLDER, DELETE ANY MEASURE FOLDER, DROP ANY MEASURE FOLDER, INSERT ANY MEASURE FOLDER
Interaction	CREATE INTERACTION, CREATE ANY INTERACTION, DROP ANY INTERACTION, UPDATE ANY INTERACTION

ตัวอย่างการเปิดใช้งาน ออดิตสกีม่าอ็อปเจ็คต์

เช่นถ้าต้องการให้ออดิต SELECT, INSERT และ DELETE บน object ที่ชื่อ dept โดยที่มี user jward เป็นเจ้าของและโดย BY ACCESS ที่ทำสำเร็จ จะเขียนคำสั่งได้ดังนี้

```
AUDIT SELECT, INSERT, DELETE
ON jward.dept
BY ACCESS
WHENEVER SUCCESSFUL;
```

หากต้องการให้ ออดิตอ็อบเจ็คต์ ที่เป็น default และ ออดิตคำสั่ง SELECT มีออฟชันที่เป็นคำสั่งที่ audit เอาเฉพาะที่ทำไม่สำเร็จ โดย BY SESSION จะเขียนคำสั่งได้ดังนี้

```
AUDIT SELECT
ON DEFAULT
WHENEVER NOT SUCCESSFUL;
```

ตัวอย่างการปิดการใช้งาน

ในตอนที่ปิดการใช้งาน ออดิตต้องเปิดทำอย่างไรก็ให้เหมือนกับตอนปิด แตกต่างกันตรงที่ใช้คำสั่งเป็น NOAUDIT เช่น

```
NOAUDIT DELETE
ON emp;
NOAUDIT SELECT, INSERT, DELETE
ON jward.dept;
```

หรือหากต้องการปิดทั้งหมดก็ทำได้ดังนี้

```
NOAUDIT ALL
ON emp;
```

และถ้าต้องการที่จะทำการปิดทั้งหมดในดีฟอร์ท(default) ก็ทำได้ดังนี้

```
NOAUDIT ALL
ON DEFAULT;
```

บทที่ 4
ความสามารถของ ดีบีเอ็มเอส ที่ใช้กลไก มัลติเวอร์ชัน (โพสเกรสเอสคิวแอล)

4.1 โพสเกรสเอสคิวแอล ทรานแซคชัน

ทรานแซคชันคือ ชุดของคำสั่งภาษาฐานข้อมูลในระดับ โลจิคอล ซึ่งยอมให้มีการละเมิดกฎความถูกต้องเป็นการภายใน ชั่วคราว ในหนึ่งทรานแซกชันอาจจะประกอบไปด้วยชุดของคำสั่งเอสคิวแอล หนึ่งคำสั่งหรือมากกว่าหนึ่งคำสั่ง ก็ได้ การทำงานของทรานแซกชันถ้าทำงานสำเร็จ (COMMIT) ก็ต้องสำเร็จทั้งหมด แต่ถ้าไม่สำเร็จ (โรลแบ็ก) ก็ไม่สำเร็จด้วยกันทั้งหมด

4.1.1 COMMIT ทรานแซกชัน

โพสเกรสเอสคิวแอล ใช้คำสั่ง COMMIT, COMMIT WORK, COMMIT TRANSACTION ในการยืนยันการเปลี่ยนแปลงของทรานแซคชั่นโดยการทำงานของคำสั่ง คอมมิต เมื่อมีการเปลี่ยนแปลงหลังจากการทำทรานแซคชันแล้วโดยการทำงานของทรานแซคชันจะถูกบันทึกไว้ในไฟล์ pg_clog ซึ่งทำหน้าที่ในการเก็บสถานะของทรานแซกชัน เช่น Committed or Aborted

4.1.2 โรลแบ็ก ทรานแซกชัน

โพสเกรสเอสคิวแอล ใช้คำสั่ง ROLLBACK, ROLL BACK WORK, ROLLBACK ทรานแซกชัน ในการทำการย้อนกลับของทรานแซคชัน โดยการทำงานของคำสั่ง ROLLBACK เมื่อทรานแซคชันเกิดข้อผิดพลาด PostgrSQL จะยกเลิกค่าการทำทรานแซคชันแล้วไปอ่านค่าจาก pg_xlog ซึ่งทำหน้า WAL

4.1.3 SAVEPOINT ทรานแซกชัน

โพสเกรสเอสคิวแอล ใช้คำสั่ง SAVEPOINT savepoint-name เป็นจุดเมื่อมีการทำ ทรานแซกชันไประยะหนึ่งแล้ว ถ้าหากเกิดข้อผิดพลาดเกิดขึ้นระหว่างการทำทรานแซกชันระบบจะ โรลแบ็ก มายังตำแหน่งที่มีการทำ SAVEPOINT เอาไว้

4.2 ทดสอบ คุณสมบัติ ACID ของ ทรานแซกชัน

4.2.1 Atomicity

จากคุณสมบัติของ Atomicity ถ้าการทำงานสำเร็จก็ต้องสำเร็จหมด แต่ถ้าไม่สำเร็จก็ต้องไม่สำเร็จทั้งหมดกรณีสำเร็จก็ต้องสำเร็จหมด

did	dname
10	COMPUTER
20	PLANNING

eid	fname	lname	did
000001	SMITH	CLERK	10
000002	ALLEN	SALESMAN	20
000003	JONES	MANAGER	20

รูปที่ 4.1 ตัวอย่างตารางทดสอบ Atomicity

จากตัวอย่างในการทดสอบคุณสมบัติ Atomicity จะนำตารางมาทำการทดสอบ โดยในวิธีการทดสอบจะทำการเพิ่มข้อมูลใหม่เข้าไปทั้ง 2 ตารางพร้อมกันแล้วดูผลการทดสอบว่าถ้า คอมมิต แล้วจะมีข้อมูลในตารางทั้ง 2 หรือไม่ถ้ามีข้อมูลในตารางทั้ง 2 แสดงว่าเป็นไปตามคุณสมบัติของ Atomicity คือถ้าสำเร็จก็สำเร็จหมด

```
mystore=# BEGIN;
BEGIN
mystore=# INSERT INTO dept VALUES('30','LOGISTICS');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('40','MARKETING');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000004','BLACK','KILLER','10');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000005','BOLD','RED','30');
INSERT 0 1
mystore=# COMMIT;
COMMIT
```

รูปที่ 4.2 แสดงการเพิ่มข้อมูลเพื่อทดสอบ Atomicity

จากรูปเป็นการเพิ่มข้อมูลลงใน 2 ตาราง ดูผลการทดสอบหลังจากที่ทรานแซคชันคอมมิต

did	dname
10	COMPUTER
20	PLANNING
30	LOGISTICS
40	MARKETING

eid	fname	lname	did
000001	SMITH	CLERK	10
000002	ALLEN	SALESMAN	20
000003	JONES	MANAGER	20
000004	BLACK	KILLER	10
000005	BOLD	RED	30

รูปที่ 4.3 ผลลัพธ์ของคุณสมบัติ Atomicity คือการสำเร็จต้องสำเร็จหมด

กรณีถ้าไม่สำเร็จก็ไม่สำเร็จหมด

\

did	dname
10	COMPUTER
20	PLANNING
30	LOGISTICS
40	MARKETING

eid	fname	lname	did
000001	SMITH	CLERK	10
000002	ALLEN	SALESMAN	20
000003	JONES	MANAGER	20
000004	BLACK	KILLER	10
000005	BOLD	RED	30

รูปที่ 4.4 ตัวอย่างตารางที่นำมาพิสูจน์คุณสมบัติ Atomicity

การทดสอบในกรณีที่ทรานแซคชันทำงานไม่สำเร็จจะใช้ 2 ตารางข้างต้นนี้โดยจะทำการเพิ่มข้อมูลลงไปใน 2 ตารางพร้อมกันและดูผลการทดสอบว่าทำไมทรานแซคชันนี้ถึงทำงานไม่สำเร็จ

```
mystore=# BEGIN;
BEGIN
mystore=# INSERT INTO emp VALUES('000006','SCREW','BLACK','40');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000007','VHDL','RED','30');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000008','DREAM','BLUE','30');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('50','ACCOUNT');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('60','SALES');
INSERT 0 1
mystore=# ROLLBACK;
ROLLBACK
```

รูปที่ 4.5 แสดงการเพิ่มข้อมูลลงในตาราง

จากรูปดังกล่าวเป็นการใช้คำสั่งเพื่อเพิ่มข้อมูลเข้าไปใน 2 ตารางจะสังเกตุว่าเมื่อทรานแซคชันทำงานไม่สำเร็จจะต้องเป็นไปตามคุณสมบัติของ Atomicity คือ ถ้าไม่สำเร็จก็ไม่สำเร็จหมด

```
 did |   dname
-----+-----------
 10  | COMPUTER
 20  | PLANNING
 30  | LOGISTICS
 40  | MARKETING
```

```
  eid   | fname |  lname   | did
--------+-------+----------+-----
 000001 | SMITH | CLERK    | 10
 000002 | ALLEN | SALESMAN | 20
 000003 | JONES | MANAGER  | 20
 000004 | BLACK | KILLER   | 10
 000005 | BOLD  | RED      | 30
```

รูปที่ 4.6 ไม่มีข้อมูลเพิ่มเข้าไปในตาราง

จากรูปเมื่อทรานแซคชันทำงานไม่สำเร็จผลที่เกิดขึ้นก็คือจะไม่มีข้อมูลที่ถูกเพิ่มเข้าไปในตารางทั้ง 2 ถ้าไม่มีข้อมูลแสดงว่าเป็นไปตามคุณสมบัติของ Atomicity

4.2.2 Consistency

ฐานข้อมูลจะต้องอยู่ในสภาพที่ถูกต้องอยู่เสมอ ไม่ว่าทรานแซคชันจะสำเร็จหรือล้มเหลวตาม การประมวลผลของทรานแซคชันจะต้องทำให้ข้อมูลคงความถูกต้องเสมอ

ตัวอย่างการทดสอบคุณสมบัติ Consistency จะทำการเพิ่มข้อมูลใหม่ในตาราง emp ซึ่งจะทำให้ผิดกฎของ Constraint ดังนั้นจะทำให้ทรานแซคชัน โรลแบ็ก เมื่อทรานแซคชัน โรลแบ็ก แล้วก็ไปตรวจสอบว่าข้อมูลในตารางถูกต้องตามคุณสมบัติ Consistency หรือไม่

did	dname
10	COMPUTER
20	PLANNING
30	LOGISTICS
40	MARKETING

eid	fname	lname	did
000001	SMITH	CLERK	10
000002	ALLEN	SALESMAN	20
000003	JONES	MANAGER	20
000004	BLACK	KILLER	10
000005	BOLD	RED	30

รูปที่ 4.7 แสดงตารางที่นำมาทดสอบคุณสมบัติ Consistency

คำสั่งที่เพิ่มข้อมูลลงในตารางแล้วทำให้ทรานแซคชัน โรลแบ็ก

```
mystore=# BEGIN;
BEGIN
mystore=# INSERT INTO emp VALUES('000006','SCREW','BLACK','40');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000007','VHDL','RED','30');
INSERT 0 1
mystore=# INSERT INTO emp VALUES('000008','DREAM','BLUE','50');
ERROR:  insert or update on table "emp" violates foreign key constraint "emp_did_fkey"
DETAIL:  Key (did)=(50) is not present in table "dept".
mystore=# COMMIT;
ROLLBACK
mystore=#
```

รูปที่ 4.8 ตารางแสดงการเพิ่มข้อมูลเข้าไปในฐานข้อมูล

หลังทำคำสั่งดังกล่าวจะส่งผลทำให้ทรานแซคชัน โรลแบ็ก และทำให้การเพิ่มข้อมูลลงไปในตารางไปเป็นผลเหมือนไม่มีการเพิ่มข้อมูล

did	dname
10	COMPUTER
20	PLANNING
30	LOGISTICS
40	MARKETING

eid	fname	lname	did
000001	SMITH	CLERK	10
000002	ALLEN	SALESMAN	20
000003	JONES	MANAGER	20
000004	BLACK	KILLER	10
000005	BOLD	RED	30

รูปที่ 4.9 ตารางแสดงการทำงานหลังจากที่ทรานแซคชัน คอมมิต

หลังจากที่ทรานแซคชัน โรลแบ็ก จะทำให้ในทั้งสองตารางไม่มีข้อมูลที่เพิ่มเข้าไปใหม่

4.2.3 Isolation

ไม่ว่าในระบบจะมีทรานแซคชัน ทำงานอยู่จำนวนเท่าไรก็ตามการทำงานของทรานแซคชัน หนึ่งจะต้องทำงานร่วมกับทรานแซคชันอื่นเสมือนว่าแต่ละทรานแซคชัน นั้นทำงานเป็นอิสระซึ่งกันและกัน ในที่นี้จะทำการทดสอบเมื่อมีทรานแซคชัน 2 ทรานแซคชันการทำงานทั้งสองจะเป็นอย่างไร โดยจะให้ทรานแซคชันทั้งสองทำการอ่านข้อมูลเดียวกันแล้วสังเกต ดูการทำงานว่าจะเป็นไป ตามคุณสมบัติ ไอโซเลชัน หรือไม่

```
mystore=# BEGIN;
BEGIN
mystore=# SELECT * FROM dept;
 did |   dname
-----+-----------
 10  | COMPUTER
 20  | PLANNING
 30  | LOGISTICS
 40  | MARKETING
(4 rows)

mystore=# UPDATE dept SET dname='COMPUTER' WHERE did='30';
UPDATE 1
mystore=#
```

รูปที่ 4.10 แสดงการแก้ไขข้อมูลของทรานเซ็คชัน TA

จากรูป ทรานแซคชัน TA เริ่มทรานแซคชันด้วยคำสั่ง BEGIN จากนั้นใช้คำสั่ง SELECT เพื่อ เรียกดูข้อมูลในตาราง และต่อมาก็ทำการแก้ไขข้อมูลในตารางด้วยจะแก้ไขรายชื่อแผนกจาก LOGISTICS เป็น COMPUTER ต่อมามีทรานแซคชันหนึ่งมาทำการแก้ไขข้อมูลเดียวกันนี้ทรานแซคชัน TB

```
mystore=# BEGIN;
BEGIN
mystore=# SELECT * FROM dept;
 did |   dname
-----+-----------
 10  | COMPUTER
 20  | PLANNING
 30  | LOGISTICS
 40  | MARKETING
(4 rows)

mystore=# UPDATE dept SET dname='SALES' WHERE did='30';
```

รูปที่ 4.11 แสดงการแก้ไขข้อมูลของทรานแซคชัน TB

จากรูปมีอีกทรานแซคชันที่มาอ่านข้อมูลเดียวกันและจะทำการแก้ไขข้อมูลเดียวกันด้วย ดังนั้นทรานแซคชันที่มาทำงานที่หลังจะต้องรอให้ทรานแซคชันแรก คอมมิต หรือ โรลแบ็ก ก่อน ทรานแซคชันที่มาทำที่หลังถึงจะสามารถทำงานต่อไปได้

```
mystore=# UPDATE dept
UPDATE 1
mystore=# COMMIT;
COMMIT
mystore=#
```

TA : COMMIT

```
-----+-----------
 10  | COMPUTER
 20  | PLANNING
 30  | LOGISTICS
 40  | MARKETING
(4 rows)

mystore=# UPDATE dept SET dname='SALES' WHERE did='30';
UPDATE 1
mystore=#
```

TB จึงสามารถทำงานได้

รูปที่ 4.12 ตารางแสดงข้อมูลหลังจากที่ทดสอบทรานแซคชัน

จากตัวอย่างจะเห็นว่า เมื่อทรานแซคชัน TA คอมมิต ทรานแซคชัน TB จึงจะสามารถทำงานต่อด้วย

4.2.4 Durability

เมื่อมี ทรานแซคชัน หนึ่งทำงานเสร็จสิ้นลงแล้ว ผลของการประมวลของ ทรานแซคชันจะต้องไม่สูญหาย ถึงแม้ว่าจะมีเหตุขัดข้องหลังจากทรานแซคชัน เสร็จสิ้นแล้วก็ตาม เช่น ระบบล้ม คอมพิวเตอร์หยุดทำงาน ระบบฐานข้อมูลจะต้องรักษาสถานะของฐานข้อมูลที่ถูกต้องไว้ การทดสอบคุณ Durability จะทำการเพิ่มข้อมูลในตารางและ คอมมิต แล้วทำการรีเซ็ตระบบพอระบบ boot ขึ้นมาใหม่แล้วมาดูผลการทดสอบว่าเป็นไปตามคุณสมบัติหรือไม่

```
BEGIN
mystore=# SELECT * FROM dept;
 did |   dname
-----+-----------
 10  | COMPUTER
 20  | PLANNING
 40  | MARKETING
 30  | COMPUTER
(4 rows)

mystore=# INSERT INTO dept VALUES('50','DEPART 1');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('60','DEPART 2');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('70','DEPART 3');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('71','DEPART 4');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('72','DEPART 5');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('73','DEPART 6');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('74','DEPART 7');
INSERT 0 1
mystore=# INSERT INTO dept VALUES('75','DEPART 8');
INSERT 0 1
```

รูปที่ 4.13 แสดงการเพิ่มข้อมูลเพื่อทดสอบคุณสมบัติ Durability

จากรูปจะเห็นว่ามีการเพิ่มข้อมูลลงไปในตารางแล้วทำการรีเซ็ตระบบพอระบบบูธขึ้นมาแล้วมาดูผลการทำงานของทรานแซคชันว่าหลังจากที่รีเซ็ตระบบแล้วนั้นข้อมูลจะยังมีอยู่ในตารางหรือไม่ ถ้าข้อมูลมีแสดงว่าเป็นไปตามคุณสมบัติ Durability

```
mystore=# SELECT * FROM dept;
 did |    dname
-----+-------------
 10  | COMPUTER
 20  | PLANNING
 40  | MARKETING
 30  | COMPUTER
 50  | DEPART 1
 60  | DEPART 2
 70  | DEPART 3
 71  | DEPART 4
 72  | DEPART 5
 73  | DEPART 6
 74  | DEPART 7
 75  | DEPART 8
 76  | DEPART 9
 77  | DEPART 10
 78  | DEPART 11
 79  | DEPART 12
 80  | DEPART 13
(17 rows)
```

รูปที่ 4.14 แสดงคุณสมบัติของ Durability

ผลจากการทดลองจะเห็นว่ามีข้อมูลในตารางครบตามที่เราได้เพิ่มเข้าไปในตารางแสดงว่าเป็นไปตามคุณสมบัติของ Durability

4.3 มัลติเวอร์ชันคอลเคอเรนซี่คอลโทรล ในผลิตภัณฑ์โพสเกรสเอสคิวแอล

มัลติเวอร์ชันคอลเคอเรนซี่คอลโทรล ในผลิตภัณฑ์โพสเกรสเอสคิวแอล(Multiversion Concurrency Control PostgreSQL) ใน โพสเกรสเอสคิวแอล จะมีการกำหนดมาตฐานของ ไอโซเลชัน เลเวลไว้ 4 มาตรฐานด้วยกัน แต่ว่า โพสเกรสเอสคิวแอล จะอนุญาตให้ใช้ ไอโซเลชัน เลเวลได้ 2 แบบเท่านั้นคือ Read committed กับ Serializable เมื่อเลือกระดับ ไอโซเลชัน เลเวล เป็น Read uncommitted จะเท่ากับเป็นการเลือกRead committed และเช่นเดียวกันเมื่อเลือกระดับของ ไอโซเลชัน เลเวลเป็น Repeatable read จะเท่ากับเป็นการเลือก Serializable ดังนั้น การทำงานของ ไอโซเลชัน เลเวลจะต้องมีการทำงานที่ถูกต้องโดยจะต้องเข้มงวดในเรื่องของความถูกต้องดังนั้นใน โพสเกรสเอสคิวแอล จึงได้อนุญาติให้ใช้ได้แค่ 2 มาตรฐานเท่านั้นดัง

ตารางที่ 4.1 ไอโซเลชัน เลเวลที่มีในโพสเกรสเอสคิวแอล

ANSI/ISO ไอโซเลชัน	Dirty read	Unrepeatable read	Phantom read
Read committed	Not possible	Possible	Possible
Serializable	Not possible	Not possible	Not possible

สำหรับคำสั่งในการตั้งค่าทรานแซกชันไอโซเลชัน เลเวลของโพสเกรสเอสคิวแอล มีดังนี้

```
SET TRANSACTION ISOLATION LEVEL {READ COMMITTED | SERIALIZABLE}
SET SESSION CHARACTERSTICS AS TRANSACTION ISOLATION LEVEL {READ COMMITTED | SERIALIZABLE }
```

4.3.1 ทดสอบ ไอโซเลชัน เลเวลกับบัญหา Concurrency Control 4 ข้อ

ในการทดสอบปัญหา 4 ข้อของ Concurrency Control ใน โพสเกรสเอสคิวแอล จะมีค่าไอโซเลชัน เลเวลพื้นฐาน เป็น READ COMMITTED

4.3.1.1 Lost update Problem

เป็นปัญหาที่เกิดขึ้นเมื่อมี 2 ทรานแซกชันหรือมากกว่า มาทำการอ่านข้อมูลเดียวกันเพื่อนำไปทำการแก้ไขข้อมูล โดยที่ไม่รู้ว่ามีผู้ใช้คนอื่นได้นำข้อมูลชุดเดียวกันนี้ไปแก้ไข

เหมือนกัน จนกระทั้งทรานแซกชันแรกได้ทำการแก้ไขข้อมูลแล้วทำการบันทึกเรียบร้อย และในเวลาต่อมา ทรานแซกชันที่ 2 ก็ได้ทำการแก้ไขข้อมูลเดียวกันนั้นส่งผลทำให้ข้อมูลที่ ทรานแซกชันแรกสูญหายโดยไม่โดยที่เจ้าของไม่รู้ตัว ดังนั้นการแก้ไขข้อมูลของ ทรานแซกชันที่ 2 จะสามารถทำได้ก็ต่อเมื่อให้ ทรานแซกชันที่ 1 ทำงานเสร็จเรียบร้อยแล้วเท่านั้น

ตารางที่ 4.2 ปัญหา The Lost Update Problem

TA	Time	TB
SELECT * FROM dept;	t1	
	t2	SELECT * FROM dept;
UPDATE dept SET dname ='MARKETING';WHERE deptno ='30';	t3	
	t4	UPDATE dept SET dname ='MARKETING';WHERE deptno ='30';

การทดสอบปัญหา The Lost Update Problem ใน โพสเกรสเอสคิวแอล สามารถแก้ไขปัญหานี้โดยใช้ สติ๊กกู-เฟสล็อกกิ้ง โปรโตคอล ในการแก้ปัญหาโดย เมื่อมีการเริ่มทรานแซกชันโดยการใช้คำสั่ง BEGIN การควบคุมการทำงานในสภาวะพร้อมกันนี้ โพสเกรสเอสคิวแอล จะทำการล็อกแบบ Exclusive Locks ก่อนจากนั้นเมื่อมีการใช้คำสั่ง SELECT….FOR UPDATE จะเพิ่มการล็อกเป็น Locks Row หรือ RowSharedLock การล็อก Row จะทำการล็อกจนกว่าทรานแซกชันที่เริ่มจะ คอมมิต หรือว่า โรลแบ็ก ทรานแซกชันอื่นจึงจะสามารถทำงานได้

การทดสอบปัญหา The Lost Update Problem

ตารางที่ 4.3 การแก้ปัญหาThe Lost Update Problem

ทรานแซกชัน A	ทรานแซกชัน B
SELECT * FROM dept FOR UPDATE;	
	SELECT * FROM dept FOR UPDATE; <<wait>>

ตารางที่ 4.3 การแก้ปัญหาThe Lost Update Problem(ต่อ)

ทรานแซกชัน A	ทรานแซกชัน B
UPDATE edb SET dname ='MARKETING' WHERE deptno='50';	
COMMIT;	<<ready>>
	UPDATE edb SET dname ='PLANING' WHERE deptno='50'; <<wait>>
	COMMIT;

4.3.1.2 Uncommitted Dependency Problem

เป็นปัญหาที่เกิดขึ้นเมื่อทรานแซกชัน Aสามารถเข้าไปอ่านข้อมูล ซึ่งอยู่ระหว่างการแก้ไขข้อมูลของ ทรานแซกชัน B ซึ่ง ทรานแซกชัน B ยังทำงานไม่เสร็จสมบูรณ์หรือว่ายังไม่คอมมิต และปัญหาที่เกิดขึ้นก็คือถ้าเกิดว่าเมื่อ ทรานแซกชัน B เกิดข้อผิดพลาดขึ้น เช่น ไฟดับ ดังนั้น ทรานแซกชัน B จึงทำการ โรลแบ็ก ส่งผลทำให้ ทรานแซกชัน A ได้รับข้อมูลที่ผิดพลาดทันทีเนื่องจากว่าข้อมูลที่ ทรานแซกชัน A อ่านมาจาก ทรานแซกชัน B เสมือนว่าไม่เคยเกิดขึ้นมาก่อนเลย

ตารางที่ 4.4 ปัญหา Uncommitted Dependency Problem

ทรานแซกชัน A	Time	ทรานแซกชัน B
-	**t1**	**Write(R)**
Read(R)	**t2**	
	t3	**ROLLBACK**
Write(R) เกิดปัญหา	**t4**	

การแก้ไขปัญหานี้ใน โพสเกรสเอสคิวแอล ใช้วิธีการ ใช้ สติ๊ก ทู-เฟส ล็อกกิ๊งโปรโตคอล เมื่อทรานแซกชัน A เริ่มทรานแซกชันก็จะมีการล็อกแบบ เอ็กล็อกเมื่อทรานแซคชัน A ใช้คำสั่ง

SELECT ... FOR UPDATE ก็จะเป็นการเพิ่มการล็อกเกิดขึ้นโดยการล็อกนั้นจะมีการล็อกแบบ Accessharedlock และ RowSharedLock ถ้าทรานแซกชันอื่นจะเข้ามาแก้ไขข้อมูลจะต้องรอให้ทรานแซกชัน A ทำงานจนเสร็จก่อนหรือว่า คอมมิต หรือ โรลแบ็ก ก่อนทรานแซกชันอื่นถึงจะสามารถมาทำงานได้

ตารางที่ 4.5 การแก้ปัญหา Uncommitted Dependency Problem

ทรานแซกชัน A	ทรานแซกชัน B	ทรานแซกชัน C
UPDATE dept SET dname='PLANNING' WHERE deptno = '50';		
	SELECT * FROM dept WHERE deptno='50' FOR UPDATE; << wait >>	SELECT * FROM dept WHERE deptno='50' << dename = COMPUTER >> ใช้ข้อมูลเวอร์ชัน เก่า
ROLLBACK;		
	UPDATE dept SET dname='PLANNING' WHERE deptno = '50';	
	COMMIT;	COMMIT;

จากตัวอย่างนี้จะเป็นการแก้ไขรายชื่อแผนกคือทรานแซคชัน A จะแก้ไขชื่อแผนกจากแผนก COMPUTER เป็นแผนก PLANNING ต่อมาทรานแซคชัน B จะทำการแก้ไขชื่อแผนกเหมือนกันโดยใช้คำสั่ง SELECT ... FOR UPDATE ทรานแซคชัน B จะต้องรอให้ทรานแซกชัน A ทำงานจนเสร็จก่อน (คอมมิต หรือ โรลแบ็ก) จึงจะสามารถทำงานต่อไป ส่วนทรานแซกชัน C ต้องการเรียกดูข้อมูลในตารางทรานแซกชัน C จะเห็นข้อมูลเก่าที่ยังไม่มีการแก้ไขใด ๆ

4.3.1.3 Inconsistent Analysis Problem

เป็นปัญหาที่เกิดขึ้นเมื่อการทำงานของ ทรานแซกชัน A มีการอ่านข้อมูลเดียวกันอยู่หลาย ๆ รอบ ซึ่งยังไม่มีการ คอมมิต จากนั้นมี ทรานแซกชัน B สามารถเข้ามาแก้ไขข้อมูล

เดียวกันนี้ได้ดังนั้นข้อมูลที่ ทรานแซกชัน A อ่านได้ไปหลังจาก ทรานแซกชัน B ได้แก้ไขไปแล้ว เป็นข้อมูลที่ขัดแย้งกับข้อมูลที่อ่านในครั้งแรก

ตารางที่ 4.6 ปัญหา Inconsistent Analysis Problem

Acc1	Acc2	Acc3
40	50	30

Transaction A	Time	Transaction B
Fetch Acc1 (40)	t1	-
Fetch Acc1 (90)	t2	-
-	t3	Fetch Acc3
-	t4	Update Acc3 30->20
-	t5	Fetch Acc1
-	t6	Update Acc1 : 40->50
-	t7	COMMIT
Fetch Acc3: sum = 110	t8	

จากตัวอย่างจะเห็นว่า ข้อมูลในบัญชี Acc1 = 40, Acc2 = 50, Acc3 = 30 ทรานแซกชัน A ได้ทำการอ่านข้อมูลใน Acc1 ไปหาผลรวมซึ่งได้ค่าเป็น 40 ณ ช่วงเวลา t1 จากนั้น ทรานแซกชัน A ได้ทำการอ่านข้อมูลใน Acc2 ไปหาผลรวมซึ่งผลรวมเป็น 90 (Acc1 + Acc2) ณ ช่วงเวลา t2 แต่ทรานแซกชัน 2 ได้ทำการอ่านข้อมูลของ Acc3 ณ ช่วงเวลา t3 และหลังจากนั้น ทรานแซกชัน B ได้ทำการแก้ไขข้อมูลจาก 30 เป็น 20 ณ ช่วงเวลา t4 จากนั้นทำการอ่านข้อมูลของ Acc1 ณ ช่วงเวลา t5 และนำข้อมูลที่อ่านได้ไปทำการแก้ไขจาก 40 เป็น 50 ณ ช่วงเวลา t6 และหลังจากนั้น ทรานแซกชัน 2 จึงทำการ คอมมิต ณ ช่วงเวลา t7 ซึ่งการทำงานของ ทรานแซกชัน B ไม่น่าจะมีปัญหาใด ๆ ที่ส่งผลกระทบกับ ทรานแซกชัน A เนื่องจาก ทรานแซกชัน B ทำการแก้ไขข้อมูลจาก Acc3 ไป Acc1 เท่านั้น ปัญหาที่เกิดขึ้นก็คือว่า ในช่วงเวลา t8 ทรานแซกชัน A ได้ทำการอ่านข้อมูลของ Acc3 ซึ่งตอนนี้ข้อมูลใน Acc3 ได้มีการเปลี่ยนแปลงไปแล้วจาก 30 เป็น 20 ดังนั้นผลรวมในช่วงเวลา t8 จึงเป็นค่าจึงเกิดข้อมูลพลาดเป็น 110 แทนที่ข้อมูลจะมีค่าเป็น 120 ซึ่งเป็นผลรวมที่ถูกต้อง

ตารางที่ 4.7 ปัญหา Inconsistent Analysis Problem

TX A	TX B
Read R acc1(40) ; sum = 40	
Read R acc2(50) ; sum = 90	
	update R acc3(20) ; 30-10
	update R acc1(50) ; 40+10
	COMMIT;
Read R acc3(20) ; sum = 110	

ตารางที่ 4.8 ไอโซเลชัน เลเวล READ COMMITTED แก้ปัญหาข้อ 3, 4 ไม่ได้

Tx A	Tx B
SELECT dname FROM ebd WHERE did='00002'; << dname = 0 >>	
	UPDATE edb SET dname='COMPUTER' WHERE did='50';
	COMMIT;
SELECT dename FROM deb WHERE did='00003'; << dname = 10 >>	

4.3.1.4 Phantom Phenomenon

เป็นปัญหาที่เกิดขึ้นเมื่อทรานแซกชัน A ต้องการอ่านข้อมูลในขณะที่ทรานแซกชันยังไม่ คอมมิตแต่ว่าในขณะเดียวกัน ทรานแซกชัน B สามารถที่จะเพิ่มข้อมูลได้ ดังนั้นข้อมูลที่ ทรานแซกชัน A อ่านไปนั้นจะมีข้อมูลซึ่งไม่เคยมีมาก่อนเข้าไปร่วมในการวิเคราะห์ด้วย สิ่งที่ควรเป็นก็คือว่าทรานแซกชัน B จะสามารถเพิ่มข้อมูลใหม่ได้ก็ต่อเมื่อต้องรอให้ ทรานแซกชัน A ทำงานเสร็จสิ้นแล้วเท่านั้นหรือว่าคอมมิตแล้วเท่านั้น

ตารางที่ 4.9 ตัวอย่าง การแก้ปัญหา ข้อ 3 , 4 ของ โพสเกรสเอสคิวแอล

Tx A	Tx B
SELECT money FROM Accounts WHERE acc# ='00003';<< dbane = 0 >>	
	UPDATE Accounts SET money =10 WHERE acc# ='00003';
	COMMIT;
SELECT money FROM Accounts WHERE acc# ='00003';<< money = 0 >>	
	SELECT money FROM Accounts WHERE acc# 1;<< money = 10 >>
	INSERT INTO Accounts VALUES ('7','20');
	COMMIT;
SELECT money FROM Accounts WHERE acc# =7;<<no rows selected >>	
COMMIT;	
sELECT money FROM Accounts WHERE acc# #=7; << money =20>>	

สรุปในการแก้ปัญหาที่เกิดจากการที่มีหลายทรานแซคชันทำงานพร้อมกันในช่วงเวลาเดียวกันโดยปัญหาที่เกิดขึ้นจะแตกต่างกันไปและจะมีวิธีการแก้ปัญหาที่ต่างกันไปด้วย ใน โพสเกรสเอสคิวแอล จะแก้ปัญหาของการทำงานพร้อมกันในช่วงเวลาเดียวกันดังต่อไปนี้

ปัญหา	วิธีการแก้ปัญหา
The Lost Update Problem	ใช้ Strict 2 Phase Locking Protocol
The Uncommitted Dependency Problem	
The Inconsistency Analysis Problem	ใช้ Multiversion
The Phantom Phenomenon	

4.4 อ็อบติไมเซอร์เซชันในโพสเกรสเอสคิวแอล

3.4.1 Sequential Scans

การทดสอบ ซีเควนเชียลสแกน(Sequential Scans) จะใช้ตาราง accounts เนื่องจากว่าตาราง accounts จะมีจำนวน row มากถึง 500,000 row ซึ่งน่าจะเหมาะกับการทดสอบการทำงานของ อ็อบติไมเซอร์ ซึ่งถ้าเรากำหนดคำถามเพื่อที่จะให้ อ็อบติไมเซอร์ หาผลลัพธ์ออกมานั้น อ็อบติไมเซอร์จะใช้เวลาเท่าไหร่ในการทำงานและทำไมถึงใช้วิธีการแบบ ซีเควนเชียลสแกน

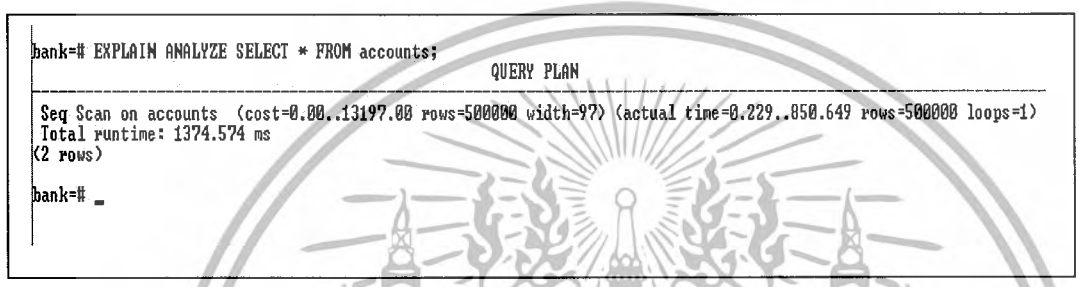
```
bank=# EXPLAIN SELECT * FROM accounts;
                          QUERY PLAN
-------------------------------------------------------------
 Seq Scan on accounts  (cost=0.00..13197.00 rows=500000 width=97)
(1 row)

bank=#
```

รูปที่ 4.15 Sequential Scan

จากตัวอย่าง การใช้คำสั่ง EXPLAIN SELECT * FROM accounts; ผลลัพธ์ที่ได้มานั้นจะเป็นลักษณะของตัวเลขซึ่งตัวเลขเหล่านี้จะบอกถึงประสิทธิ์ภาพในการทำงานของ อ็อบติไมเซอร์ว่าใช้เวลาเท่าไหร่ในการเข้าไปค้นหาข้อมูล จากตัวอย่างจะเห็นว่าเวลาที่ใช้ไปในการเข้าไปค้นหาข้อมูลในระบบ physical จริง ๆ ใช้ไปเท่าไหร่ โดยค่าที่เป็นผลลัพธ์ซึ่งค่าต่าง ๆ นี้จะเป็นค่าเฉลี่ยเท่านั้นโดยจะอธิบายได้ดังนี้

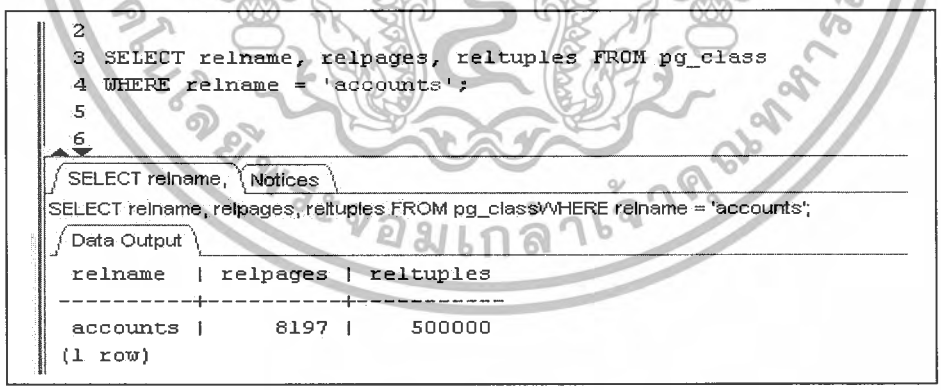
- Start up cost เป็นค่าที่ใช้ไปในการค้นหาข้อมูล
- Total cost เป็นค่ารวมของการทำงานทั้งหมด
- Row จำนวน row ที่เก็บในตารางสถิตินำมาคำนวณด้วย
- Width เป็นขนาดของตัวอักษรมีหน่วยเป็น byte

ผลลัพธ์ที่ได้มานั้นยังไม่ใช่จริงที่ อ๊อบติไมเซอร์ทำงานซึ่งค่าที่จริงจะต้องใช้คำสั่ง ANALYZE ร่วมด้วยซึ่งจะได้เห็นในตัวอย่างต่อไปต่อไปเป็นตัวอย่างการทำงานของอ๊อบติไมเซอร์โดยใช้คำสั่ง ANALYZE ร่วมด้วยเพื่อคำนวณหาเวลาที่ใช้ในการทำงานจริง



รูปที่ 4.16 การใช้คำสั่ง ANALYZE ใน Sequential Scan

จากตัวอย่างจะเห็นว่าถ้ามีการเพิ่มคำสั่ง ANALYZE เข้าไปด้วยผลลัพธ์ที่ออกมานั้นจะเป็นค่าที่แท้จริง พร้อมกับคำนวณเวลารวมทั้งหมดในการทำงานคือ total runtime: 1374.574 ms การคำนวณผลลัพธ์ออกมานั้นจะนำค่าต่าง ๆ มาจากตารางที่ทำหน้าที่เก็บสถิติโดยวิธีการคำนวณมีดังนี้



รูปที่ 4.17 ตารางเก็บสถิติเพื่อนำค่าไปคำนวณ

ในการคำนวณของ อ๊อบติไมเซอร์นั้นจะนำค่าที่เกี่ยวข้องต่าง ๆดังนี้ จากตัวอย่าง อ๊อบติไมเซอร์ จะทำการอ่าน read_page มาจากตารางคือ 8187 และอ่านจำนวน row = 500,000 ดังนั้นจึงสามารถคำนวณเวลาเฉลี่ยออกมาได้ดังนี้ Total cost = (read_page * seg_page_cost) + (rows scans * cpu_tuples) ซึ่งจะได้ (8197 * 1.0) + (500,000 * 0.01) = 13197

3.4.2 Index Scans

การสร้าง index นั้นเราสามารถสร้างพร้อมกับตอนเราสร้างตารางหรือสามารถที่จะสร้างที่หลังก็ได้ แต่เมื่อเราทำการสร้าง index แล้ว index จะนำตำแหน่งของแถวข้อมูลใน คอลั่ม ที่ทำ index ไปเก็บไว้ และเมื่อมีการเพิ่มข้อมูลลงไปในตารางก็เท่ากับว่าจะเพิ่มข้อมูลใน index ด้วย

```
bank=# EXPLAIN SELECT * FROM accounts WHERE aid = 100;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using accounts_pkey on accounts  (cost=0.00..8.32 rows=1 width=97)
   Index Cond: (aid = 100)
(2 rows)

bank=# _
```

รูปที่ 4.18 แสดงการผลการทำงานของ อ๊อบติไมเซอร์ แบบ Index scans

จากรูปจะเห็นว่า การทำงานของ อ๊อบติไมเซอร์จะทำ index scan เนื่องจากว่าในตาราง account ได้มีการสร้าง index โดย คอลั่ม ที่ทำการสร้าง index ชื่อ aid ดังนั้น อ๊อบติไมเซอร์ จึงทำการค้นหาข้อมูลโดยวิธีการ index scan จะเห็นว่าการค้นหาของ อ๊อบติไมเซอร์ แบบ Sequential scan กับแบบ Index scan นั้นเวลาที่ใช้ในการค้นหาจะต่างกันมาก ซึ่งเวลาของการใช้ index scan จะทำงานเร็วกว่ามาโดยจะสังเกตุจากตารางที่นำมาทดสอบมีจำนวนเร็คคอร์ด 500,000 row เท่ากัน

4.4.3 Bitmap Scans

Bitmap Index scans ที่จัดเก็บข้อมูลในรูปแบบ Bitmap คือ การเก็บข้อมูลจะอยู่ในรูป 1 และ 0 เท่านั้นจึงมีความเร็วในการทำงานสูง ข้อมูลที่เหมาะกับการสร้าง Bitmap index นั้นควรมีข้อมูลที่มีค่าของข้อมูลซ้ำกันมากเป็นช่วงมาก ๆ และมีการเปลี่ยนแปลงน้อย Bitmap Index เหมาะอย่างยิ่งกับการใช้ร่วมกันคอลัมน์ที่ มี caridnality ต่ำๆ เช่น 0.2% - 1% เป็นต้น อินเด็กซ์ที่ถูกสร้างขึ้นจะประกอบด้วยบิทต่างๆ เพื่อให้แทนค่าในคอลัมน์ที่ถูกทำอินเด็กซ์ของแต่ละแถว ตัวอย่างเช่น

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100;

                                QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on tenk1  (cost=2.37..232.35 rows=106 width=244)
   Recheck Cond: (unique1 < 100)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..2.37 rows=106 width=0)
```

รูปที่ 4.19 แสดงการผลการทำงานของ อ๊อบติไมเซอร์ แบบ Bitmap scans

จากตัวอย่างจะมีขั้นตอนการทำงาน 2 ขั้นตอนคือ ขั้นตอนแรกจะทำการค้นหาข้อมูลโดยการค้นหาตำแหน่ง index ของในแต่ละแถวเพื่อให้ตรงตามเงื่อนไข จากนั้นก็ทำงานอ่านในแต่ละ row ซึ่งการอ่านค่าในแต่ละ row ทำให้สิ้นเปลือง cost ค่อนข้างมากเพราะว่าจะเป็นการทำอ๊อบติไมเซอร์แบบ sequential scan ทำให้การอ่านให้แต่ละ disk_pages ต้องเสียเวลาในการ scan ดังนั้นการทำเลือกวิธีการโดย bitmap scan นั้นจะไปทำการอ้างถึงชื่อของโหนดซึ่งเป็นวิธีการโดยในตารางนั้นจะต้องมีการ sorting แล้วเท่านั้น

4.4.4 Nested loop Join

เป็นวิธีการนำ 2 ตารางมาเชื่องโยงเข้าด้วยกันซึ่งตารางที่อยู่ด้านนอกเราจะเรียกว่า outer join ส่วนตารางที่อยู่ด้านในจะเรียกว่า inner join

```
bank=# EXPLAIN SELECT h.mtime, h.delta, b.bid, a.aid FROM history h, branches b, accounts a
bank-# WHERE h.bid = b.bid AND h.aid = a.aid;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=1.11..200.93 rows=25 width=20)
   ->  Hash Join  (cost=1.11..25.40 rows=25 width=20)
         Hash Cond: (h.bid = b.bid)
         ->  Seq Scan on history h  (cost=0.00..20.20 rows=1020 width=20)
         ->  Hash  (cost=1.05..1.05 rows=5 width=4)
               ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Index Scan using accounts_pkey on accounts a  (cost=0.00..7.01 rows=1 width=4)
         Index Cond: (a.aid = h.aid)
(8 rows)

bank=# _
```

รูปที่ 4.20 แสดงการผลการทำงานของ อ๊อบติไมเซอร์ แบบ Nested Loop

จากตัวอย่าง ขั้นแรกจะทำการค้นหาข้อมูลโดยใช้ index scan ซึ่งจะใช้คีย์ในตาราง account ต่อมาจะเป็นการ sequential scan ตาราง branches และตาราง history และนำเข้าตารางทั้งสองนี้ไป Hash Join จากนั้นก็ทำ Nested loop ไปลำดับสุดท้าย การที่ อ๊อบติไมเซอร์ จะเลือกเส้นทางแบบ nested loop นั้นจะดูว่าข้อมูลที่นำมา join จะเป็นการนำข้อมูลส่วนใหญ่ของตารางดังนั้น อ๊อบติไมเซอร์ จึงทำการเลือกวิธี Nested Loop ในการ คิวรี่ plan

4.4.5 Merge Join

ตัวอย่างนี้เป็นการนำเอาตาราง branches และตาราง accounts มาทำการ Merge Join เข้าด้วยกันและดูการทำงานของ อ๊อบติไมเซอร์ ในการเลือกเส้นทางที่จะได้มาซึ่งผลลัพธ์จากการทำงาน

```
EXPLAIN SELECT /*+ USE_MERGE(a) */ b.bid, a.aid, abalance FROM branches b, accounts a WHERE b.bid = a.bi

                                  QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=69143.62..76650.97 rows=500488 width=12)
   Merge Cond: (b.bid = a.bid)
   ->  Sort  (cost=1.11..1.12 rows=5 width=4)
         Sort Key: b.bid
         ->  Seq Scan on branches b  (cost=0.00..1.05 rows=5 width=4)
   ->  Sort  (cost=69142.52..70393.74 rows=500488 width=12)
         Sort Key: a.bid
         ->  Seq Scan on accounts a  (cost=0.00..13209.88 rows=500488 width=12)
(8 rows)
```

รูปที่ 4.21 แสดงการผลการทำงานของ อ๊อบติไมเซอร์ แบบ Merge Join

จากรูปจะเห็นว่า ในการทำงานขั้นแรกจะเริ่ม scan ในตาราง accounts ด้วย sequent scan ก่อนจากนั้นจึงทำการ sort ซึ่งในการทำงานแต่ละครั้งจะสังเกตว่าจะมี cost ที่เสียไป จากนั้นก็จะไป scan ในตาราง branches และก็ทำการ sort ข้อมูล สุดท้ายจะเป็นวิธี Merge Join ซึ่งการทำงานในลักษณะนี้จะใช้พารามิเตอร์ USE_MERGE ในการกำหนดให้ทำการค้นหาในลักษณะของ Merge Join ซึ่งค่า cost ต่าง ๆ ที่ใช้ไปจะนำเอาค่าที่อยู่ในตาราง pg_class เข้ามาร่วมในการคำนวณหา total cost ซึ่งเราสามารถดูได้ว่าตารางนี้มีจำนวน disk page และจำนวนทั้งหมดของ row เท่าไหร่ ดูได้โดยการใช้คำสั่ง SELECT relname, relpages, reltuples FROM pg_class;

4.4.6 Hash Join

ตัวอย่างการทำงานที่มีความซับซ้อนมากขึ้น ตัวอย่างการ Hash Join จะเป็นการนำเอาตาราง 2 ตารางมาทำการเชื่อมโยงกัน

```
le=# EXPLAIN SELECT * FROM customer, orderinfo
le-# WHERE customer.customer_id = orderinfo.customer_id;
                        QUERY PLAN
------------------------------------------------------------------------
Join  (cost=1.34..31.64 rows=105 width=226)
h Cond: (orderinfo.customer_id = customer.customer_id)
  Seq Scan on orderinfo  (cost=0.00..24.00 rows=1400 width=28)
  Hash  (cost=1.15..1.15 rows=15 width=198)
    ->  Seq Scan on customer  (cost=0.00..1.15 rows=15 width=198)
s)

le=#
```

รูปที่ 4.22 แสดงการผลการทำงานของ อ๊อบติไมเซอร์ แบบ Hash Join

4.5 ลักษณะโครงสร้าง

ระบบที่ใช้ โพสเกรสเอสคิวแอล จะติดตั้ง โพสเกรสเอสคิวแอล ไว้ที่เครื่องเซอรเวอร์ ซึ่งเป็นที่เก็บ ฐานข้อมูล ด้วย และยังสามารถ ติดตั้ง โพสเกรสเอสคิวแอล ได้มากกว่า 1 ชุดในเซิร์ฟเวอร์ เครื่องเดียว

ผู้ดูแลระบบ โพสเกรสเอสคิวแอล จะใช้ชื่อว่า *postgres* ซึ่งเป็นผู้ดูแลทั้ง ตัวโปรแกรม และ ฐานข้อมูล ซึ่งสามารถทำงานกับบางคำสั่งเฉพาะ เพื่อจัดการ ฐานข้อมูล และ ผู้ใช้บริการ (user) ซึ่ง ผู้ดูแลระบบ ฐานข้อมูล (postgres) จะคล้ายการทำงานของซุปเปอร์ยูสเซอร์ ในระบบ ยูนิกส์ หน้าที่ของ *postgres* สามารถ สร้างชื่อ ผู้ใช้งาน และกำหนดสิทธิและระดับการใช้งานต่างๆได้

โพสเกรสเอสคิวแอล ใช้รูปแบบการทำงาน แบบ ไคลเอน /เซิร์ฟเวอร์ ซึ่งในการทำงานจะประกอบด้วย 3 โพรเซทำงานร่วมกัน คือ

a) โพสมาสเตอร์(Postmaster) เป็น supervisory daemon process ซึ่งจัดการติดต่อระหว่าง ฟรอนต์เอน(Frontend) กับแบ็คเอนโพรเซส(Backend process) ในการ

จจองแชร์บัฟเฟอร์ , จัดการค่าเริ่มต้นต่างๆในระหว่างเริ่มทำงาน และเก็บบันทึกการเข้าใช้ระบบและความผิดพลาดต่างๆที่เกิดขึ้น

b) โพสเกรส (Postgres) เป็น แบ็คเอนโพรเซสเพื่อจัดการฐานข้อมูลถือว่าส่วนนี้เป็นโพรเซส ที่ทำงานจริงๆ เช่น ทำงานตาม คิวรี่ โดย โพสมาสเตอร์จะสั่งให้สร้าง แบ็คเอนโพรเซสสำหรับทุกๆ การเชื่อมต่อกับ ฟรอนต์เอน ดังนั้น Postgres นี้จะทำงานที่ server

c) ฟรอนต์เอน(Frontend) เป็นแอพพลิเคชัน ซึ่งจะทำงานที่เครื่อง ไคลเอน และจะส่งคำสั่งการเชื่อมต่อ หรือคำสั่งต่างๆ มาที่ โพสมาสเตอร์แล้ว โพสมาสเตอร์จึงส่งต่อการทำงานไปที่ โพสเกรซ

4.5.1 หลักการทำงานของ โพสเกรสเอสคิวแอล

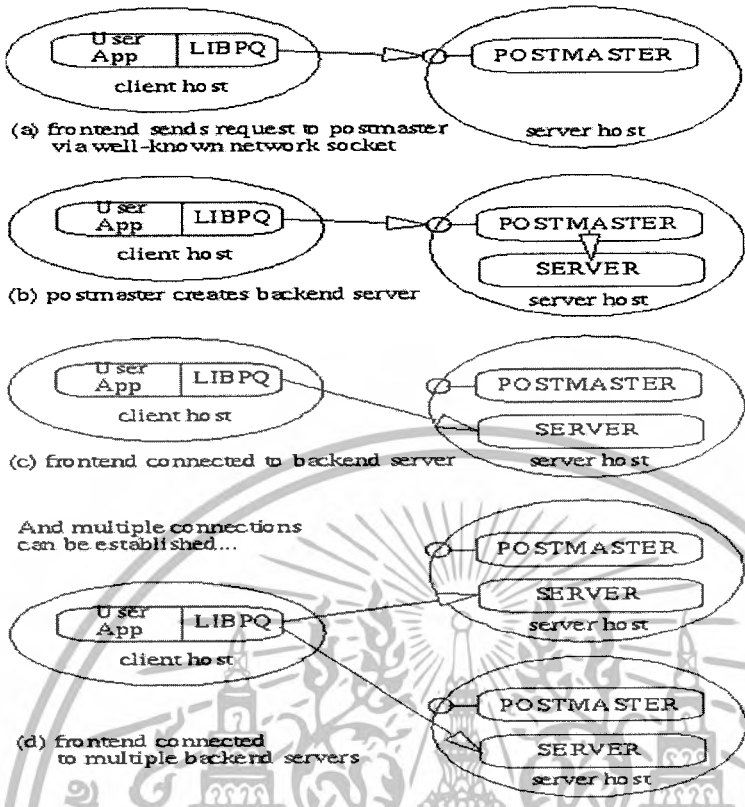
การทำงานจะแบ่งโพรเซสที่ทำงานดังที่กล่าวมาแล้ว คือ

- ในส่วนของ Supervisory daemon process คือ โพสมาสเตอร์
- ในส่วนของ User's Frontend application เช่น โปรแกรม psql หรือ CGI-Perl
- และในส่วน Backend database servers คือ โพสเกรส

เมื่อโปรแกรมทาง ฟรอนต์เอนต้องการข้อมูล หรือทำงานกับฐานข้อมูลโดยเรียกผ่านทาง library libpq ซึ่ง library libpq นี้ จะส่ง requests ผ่านทางเน็คเวิค ไปยัง โพสมาสเตอร์ เมื่อ โพสมาสเตอร์ ได้รับ request ดังกล่าวทาง โพสมาสเตอร์ จะสร้าง แบ็คเอน โพรเซส ขึ้นที่เซิร์ฟเวอร์ เพื่อติดต่อกับ ฟรอนต์เอน แทน การทำงานนั้นจะเกิดขึ้นระหว่าง ฟรอนต์เอนกับ แบ็คเอนโดยไม่ผ่านโพสมาสเตอร์ อีก และ โพสมาสเตอร์ ก็ทำงานต่อไป คือรอรับการร้องขอ อื่นๆต่อไป

Library libpq จะให้ หนึ่งฟรอนต์เอนสามารถติดต่อได้หลาย แบ็คเอน โพรเซส แต่การทำงานยังเป็นแบบ เทรดเดียว เนื่องจาก library libpq ยังไม่สามารถทำ มัลติเทรดได้ ได้

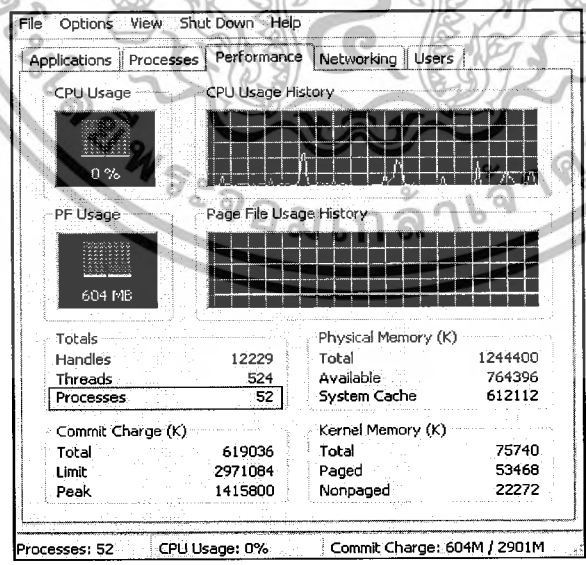
ตามหลักการที่กล่าวมา ดังนั้น โพสมาสเตอร์กับ แบ็คเอนจะต้องทำงานอยู่ที่ เครื่องเดียวกัน คือฐานข้อมูลเซิร์ฟเวอร์แต่ ฟรอนต์เอน จะทำงานที่เครื่องใดก็ได้



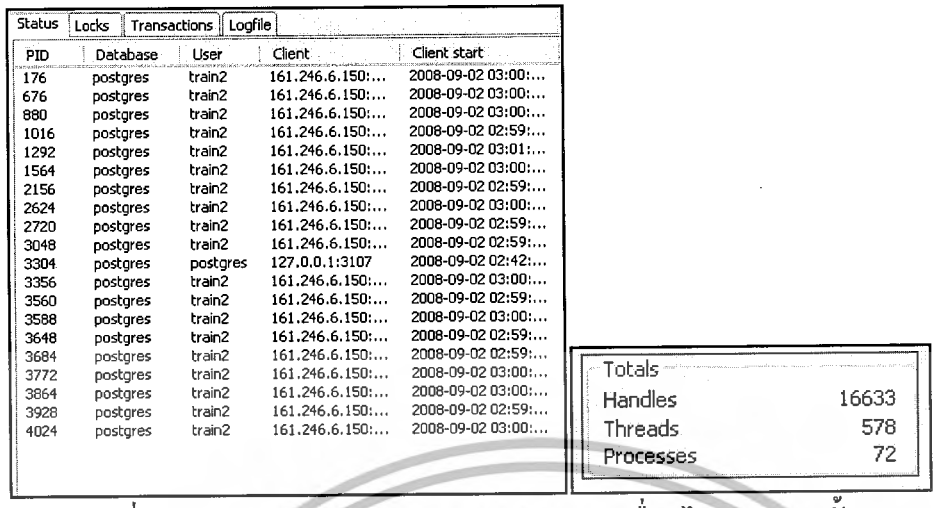
s

รูปที่ 4.23 การทำงานของโครงสร้าง Postmaser

ตัวอย่างการทดลองเรื่องการเชื่อมต่อ 1 ยูสเซอร์ ต่อ 1 โพรเซส



รูปที่ 4.24 เริ่มต้นเมื่อยังไม่มีการ ล๊อกอิน ผู้ใช้งาน ใน โพสเกรสเอสคิวแอล จะมีโพรเซสที่ ทำงานอยู่ในวินโดว์ ทั้งหมด 52 โพรเซส



รูปที่ 4.25 แสดงการ ล๊อกอินผู้ใช้งาน train2 จากเครื่อง ไคลเอน 19 ครั้ง

ดังที่เห็นว่ามี ผู้ใช้งาน train2 ปรากฏ 19 ครั้งรวมผู้ใช้งาน โพสเกรสเอสคิวแอล อีก 1 ครั้ง รวมทั้ง หมดมีการ ล็อกอิน 20 ผู้ใช้งาน

เมื่อมาดู โพรเซส ที่ทำงานอยู่บนเครื่องผู้ใช้งานจะมี โพรเซสเพิ่มขึ้นนมาจากตอนที่ยังไม่ได้ล็อกอินเลย 20 โพรเซสพอดี

สรุปว่า เมื่อมีการร้องขอ ติดต่อ ผู้ใช้งานกับเครื่องเซิร์ฟเวอร์ แล้ว ภายในเครื่องเซิร์ฟเวอร์นั้น จะมีการสร้างโพรเซส รองรับทุกๆ การเชื่อมต่อโดยเป็นอัตราส่วน 1 ผู้ใช้งาน 1 โพรเซส ไม่ว่าจะเป็นการร้องขอการเชื่อมต่อ จากเครื่อง ไคลเอนหรือภายในเครื่อง เซิร์ฟเวอร์ เองก็ตาม

4.5.2 การกำหนดสิทธิการใช้งาน (Privilege)

Database Users คือ ชื่อผู้ใช้ที่มีสิทธิเข้าไปจัดการฐานข้อมูล เช่น ตาราง ตามสิทธิที่ได้รับ สร้างและลบ ดาต้าเบสยูสเซอร์ สามารถทำได้ 2 วิธี (Login ด้วย postgres)

4.5.2.1 จากภาษา เอสคิวแอล(ใช้โปรแกรม psql)

สามารถใช้อักษรตัวเล็ก หรือ ตัวใหญ่ หรือ ตัวเล็กผสมตัวใหญ่ ได้ผลลัพธ์เดียวกัน ในทางปฏิบัติภาษา เอสคิวแอล จะใช้ตัวใหญ่และที่ไม่ใช่ภาษา เอสคิวแอล เช่นชื่อฐานข้อมูล ตาราง คอลัมน์ จะใช้ตัวเล็กหรือขึ้นตัวใหญ่ตามด้วยตัวเล็ก เพื่อให้ง่ายต่อการอ่านคำสั่งหรือแก้ไขคำสั่ง

4.5.2.2 สร้างชื่อผู้ใช้

CREATE USER ชื่อผู้ใช้ [WITH [SYSID uid] [PASSWORD 'รหัสผ่าน']]
[CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER]
[IN GROUP ชื่อกลุ่ม [, ...]][VALID UNTIL 'abstime']

```
CREATE USER train1 WITH PASSWORD '123456';
CREATE USER train2 CREATEDB;
CREATE USER train3 CREATEUSER;
CREATE USER train4 CREATEDB CREATEUSER;
CREATE USER train5 WITH PASSWORD '123456' CREATEDB CREATEUSER;
CREATE USER train6 CREATEDB CREATEUSER PASSWORD '123456';
```

รูปที่ 4.26 แสดงการสร้างยูสเซอร์

	usename name	usesysid oid	usecreatedb boolean	usesuper boolean	usecatupd boolean	passwd text	valuntil abstime	useconfig text[]
1	ipointer	10	t	t	t	md5bf4d0		
2	train1	16588	f	f	f	md527be(		
3	train2	16589	t	f	f			
4	train3	16590	f	t	t			
5	train4	16591	t	t	t			
6	train5	16592	t	t	t	md5d3561		
7	train6	16593	t	t	t	md583eb9		

รูปที่ 4.27 แสดงการสร้างผู้ใช้งานพร้อมกำหนดสิทธิการใช้งาน

ถ้ากำหนด CREATEDB CREATEUSER แสดงว่าผู้ใช้นั้นเป็น Superuser (train4) ถ้าพิมพ์ WITH , WITH PASSWORD ต้องอยู่หลังชื่อผู้ใช้ (train5) ถ้าไม่พิมพ์ WITH , PASSWORD จะอยู่หลังชื่อผู้ใช้ ตรงไหนก็ได้(train6)

เนื่องจาก iponter เป็น admin user จึงมีความสามารถ ทุกอย่างที่สามารถทำได้สิทธิของแต่ละผู้ใช้งานที่ได้รับเป็นไปที่ได้กำหนดไว้ให้ในตอนที่ create user แล้ว

4.5.2.3 แก้ไขคุณสมบัติผู้ใช้

Syntax

ALTER USER ชื่อผู้ใช้ [WITH PASSWORD 'รหัสผ่านใหม่']

[CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER]

[VALID UNTIL 'abstime']

```
ALTER USER train6 NOCREATEDB NOCREATEUSER  PASSWORD '654321';
SELECT * FROM pg_shadow;
```

	usename name	usesysid oid	usecreatedb boolean	usesuper boolean	usecatupd boolean	passwd text	valuntil abstime	useconfig text[]
1	ipointer	10	t	t	t	md5bf4d0		
2	train1	16588	f	f	f	md527be(		
3	train2	16589	t	f	f			
4	train3	16590	f	t	t			
5	train4	16591	t	t	t			
6	train5	16592	t	t	t	md5d3561		
7	train6	16594	f	f	f	md5278c7		

รูปที่ 4.28 แสดงชื่อผู้ใช้เมื่อแก้ไขคุณสมบัติผู้ใช้งาน train6

สังเกตคุณสมบัติของ ผู้ใช้งาน train6 คือ จะไม่สามารถจัดการอะไรกับฐานข้อมูลได้เลย

4.5.2.4 ลบชื่อผู้ใช้

ชื่อผู้ใช้ จากคำสั่ง DROP USER train6;

```
drop user train6;
SELECT * FROM pg_shadow;
```

	usename name	usesysid oid	usecreatedb boolean	usesuper boolean	usecatupd boolean	passwd text	valuntil abstime	useconfig text[]
1	ipointer	10	t	t	t	md5bf4d0		
2	train1	16588	f	f	f	md527be(		
3	train2	16589	t	f	f			
4	train3	16590	f	t	t			
5	train4	16591	t	t	t			
6	train5	16592	t	t	t	md5d3561		

รูปที่ 4.29 สังเกตว่า ผู้ใช้งาน train6 หายไป

Group คือ ชื่อกลุ่มที่รวบรวมชื่อผู้ใช้ที่ทำหน้าที่เหมือนกัน และให้ง่ายต่อการกำหนดสิทธิ (Privilege) เช่นกำหนดสิทธิ select ให้กลุ่ม สมาชิกที่อยู่ในกลุ่มจะได้สิทธิ select เหมือนกัน ไม่ต้องเสียเวลาในการกำหนดสิทธิให้กับผู้ใช้แต่ละคน

CREATE GROUP ชื่อกลุ่ม [WITH [SYSID gid] [USER ชื่อผู้ใช้ [, ...]]]

4.5.2.5 คำสั่งดูรายชื่อกลุ่ม

SELECT * FROM pg_group;

```
CREATE GROUP guser USER train1;
CREATE GROUP gadmin USER train2,train3,train4;
CREATE GROUP gtest;
SELECT * FROM pg_group;
```

	groname name	grosysid oid	grolist oid[]
1	guser	16595	{16588}
2	gadmin	16596	{16589,16590,16591}
3	gtest	16597	{}

รูปที่ 4.30 แสดงการสร้างกลุ่มการใช้งาน

4.5.2.6 เพิ่มสมาชิก

ALTER GROUP ชื่อกลุ่ม ADD USER ชื่อผู้ใช้

```
ALTER GROUP gtest ADD USER train5;
SELECT * FROM pg_group;
```

	groname name	grosysid oid	grolist oid[]
1	guser	16595	{16588}
2	gadmin	16596	{16589,16590,16591}
3	gtest	16597	{16592}

รูปที่ 4.31 แสดงการเพิ่มชื่อผู้ใช้งานเข้าในกลุ่ม

4.5.2.7 ลบสมาชิก

ALTER GROUP ชื่อกลุ่ม DROP USER ชื่อผู้ใช้, ...

```
ALTER GROUP gadmin DROP USER train4;
SELECT * FROM pg_group;
```

	groname name	grosysid oid	grolist oid[]
1	guser	16595	{16588}
2	gtest	16597	{16592}
3	gadmin	16596	{16589,16590}

รูปที่ 4.32 แสดงการลบรายชื่อสมาชิกออกจากกลุ่ม

4.5.2.8 ลบกลุ่ม

DROP GROUP ชื่อกลุ่ม;

```
DROP GROUP gtest;
SELECT * FROM pg_group;
```

	groname name	grosysid oid	grolist oid[]
1	guser	16595	{16588}
2	gadmin	16596	{16589,16590}

รูปที่ 4.33 แสดงการลบกลุ่มออกจากฐานข้อมูล

Privilege คือ สิทธิที่ผู้ใช้หรือกลุ่มสามารถจัดการฐานข้อมูลตามสิทธิที่ได้รับ ฐานข้อมูลที่ถูกสร้างด้วยชื่อผู้ใช้ใด ชื่อผู้ใช้คนนั้นจะเป็นเจ้าของหรือมีสิทธิในฐานข้อมูลนั้น ผู้ใช้คนอื่นไม่มีสิทธิ ยกเว้นจะกำหนดสิทธิให้

4.5.2.9 การกำหนดสิทธิ

GRANT privilege [, ...] ON object[, ...] TO{ PUBLIC|GROUPชื่อกลุ่ม|ชื่อผู้ใช้ }

ระบุ PUBLIC หมายถึง ทุกกลุ่มทุกผู้ใช้มีสิทธิ

ระบุ GROUP ชื่อกลุ่ม หมายถึง สมาชิกในกลุ่มมีสิทธิ

ระบุ ชื่อผู้ใช้ หมายถึง ชื่อผู้ใช้มีสิทธิ

จากคำสั่งขั้นต้น เห็นได้ว่าจุดที่สามารถกำหนดสิทธิได้ 2 ที่คือที่ ในส่วนของ Group และในส่วนของผู้ใช้งานเอง

4.5.2.10 การยกเลิกสิทธิ

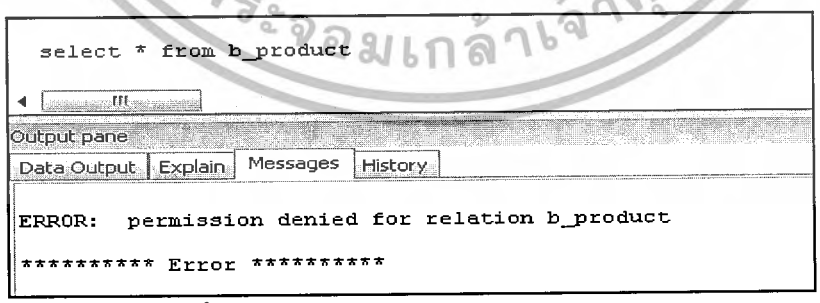
เมื่อเราต้องการยกเลิกสิทธิในความสามารถจัดการฐานข้อมูลก็ทำได้เช่นเดียวกัน จากคำสั่งต่อไปนี้

REVOKE privilege [, ...] ON object [, ...]
FROM { PUBLIC | GROUP ชื่อกลุ่ม | ชื่อผู้ใช้ }

ตารางการกำหนดสิทธิ เมื่อกำหนดให้ train6 อยู่ใน GROUP gadmin ทำการยกเลิกสิทธิ ในการขอดู Teble b_product ทั้งในส่วนของ GROUP และ USER ที่ ipoiter ซึ่งเป็น Admin

```
REVOKE SELECT ON b_product FROM GROUP gadmin ;
REVOKE SELECT ON b_product FROM train6;
```

รูปที่ 4.34 แสดงถึงการยกเลิกสิทธิการเข้าเรียกดูตาราง b_product ของ กลุ่ม gadmin และผู้ใช้ train6



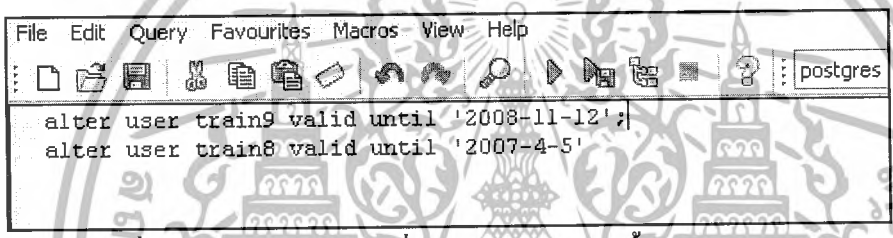
รูปที่ 4.35 แสดงให้เมื่อผู้ใช้ train6 ขอเรียกผู้ตาราง b_product ไม่สามารถทำได้

เมื่อทดลองกำหนดสิทธิการใช้งานในรูปแบบต่างๆ จะสามารถสรุปได้ตามตารางด้านล่างนี้

ตารางที่ 4.10 การกำหนดสิทธิให้กับ ผู้ใช้และกลุ่มของผู้ใช้ในรูปแบบต่างๆ

GROUP	USER	ความสามารถการจัดการฐานข้อมูล
ไม่ให้	ไม่ให้	**ไม่มีสิทธิ** ในการจัดการฐานข้อมูล
ไม่ให้	ให้	**มีสิทธิ** ในการจัดการฐานข้อมูล
ให้	ไม่ให้	**มีสิทธิ** ในการจัดการฐานข้อมูล
ให้	ให้	**มีสิทธิ** ในการจัดการฐานข้อมูล

สรุปในเรื่อง การกำหนดสิทธิแล้ว จะยึดถือการให้สิทธิเป็นหลัก คือ ไม่ว่าจะเป็นGROUP หรือ ผู้ใช้งาน ถ้าได้มีการให้สิทธิไว้ที่ใดที่หนึ่งแล้ว ถือว่า ผู้ใช้งาน นั้นมีสิทธิในการจัดการฐานข้อมูล



รูปที่ 4.36 แสดงการแก้ไขเปลี่ยนแปลงกำหนดวันสิ้นสุดการใช้งาน

จากคำสั่ง เอสคิวแอล ข้างต้น ได้กำหนดวันหมดอายุของ ผู้ใช้งาน train8 และ train9 คือ

Train8 เวลาหมดอายุ วันที่ 5 เดือน เมษายน ปี 2007

Train9 เวลาหมดอายุ วันที่ 12 เดือน พฤษจิกายน ปี 2008

เมื่อเรียกคำสั่ง select * from pg_shadow จะแสดงคุณสมบัติของผู้ใช้งาน ดังนี้

	usename name	usesysid oid	usecr boole	usesu boole	usec bool	passi text	valuntil abstime	useconfig text[]
9	train7	16410	f	f	t	md57t	infinity	
10	train9	16456	t	f	f	md5fc	2008-11-12 00:00:00+07	
11	train8	16411	f	f	f	md50(	2007-04-05 00:00:00+07	

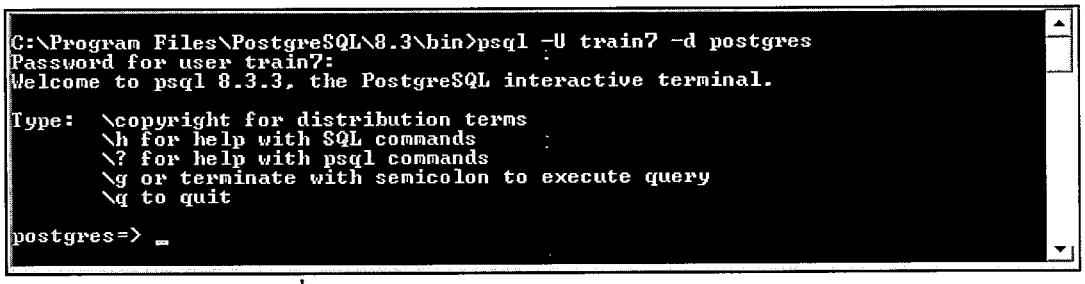
รูปที่ 4.37 แสดงให้เห็นถึงวันสิ้นสุดการใช้งาน ของผู้ใช้ train8 และ train9

เห็นได้ว่า Train7 ไม่ได้มีการตั้งเวลาวันหมดอายุ จะสามารถใช้งานได้เตลอดไป

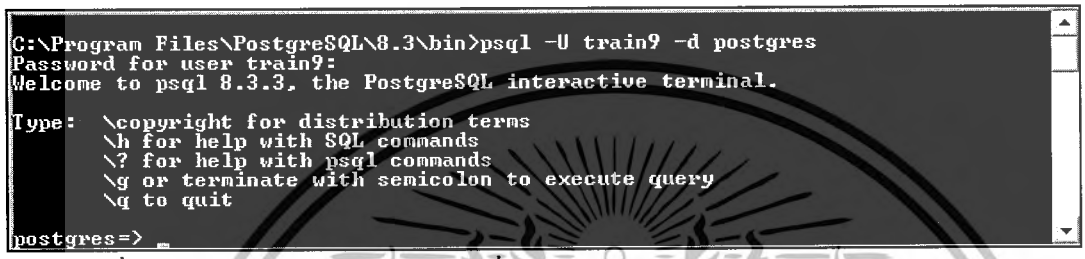
ที่ Train9 มีการตั้งเวลา แต่ยังไม่ถึงวันหมดอายุ

ที่ Train8 มีการตั้งเวลา และหมดอายุลงแล้ว

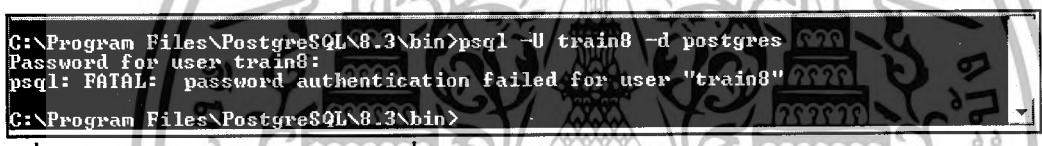
ทดลองการ ล็อกอินผู้ใช้งาน ต่างๆ



รูปที่ 4.38 ผู้ใช้งาน train7 มีอายุไม่จำกัด สามารถ ล็อกอินได้



รูปที่ 4.39 ผู้ใช้งาน train9 มีอายุถึงวันที่ 2008-11-12 ยังไม่หมดอายุ สามารถล็อกอินได้



รูปที่ 4.40 ผู้ใช้งาน train8 มีอายุถึงวันที่ 2007-04-05 หมดอายุการใช้งานแล้วไม่สามารถ ล็อกอิน ได้

4.6 การเชื่อมต่อและการยืนยันตัวบุคคล

การเชื่อมต่อและการยืนยันตัวบุคคล(Connection and authentication) การตรวจสอบสิทธิของเครื่องลูกข่ายในการเข้าใช้งาน โพสเกรสเอสคิวแอลเซิร์ฟเวอร์ จะถูกควบคุมโดยข้อความที่ระบุในไฟล์ pg_hba.conf (HBA ย่อมาจาก host-based authentication.) ที่อยู่ในไดเรคทอรี่ Drive:\Program Files\PostgreSQL\8.3\data ซึ่งไฟล์ pg_hba.conf นี้จะถูกติดตั้งเมื่อทำการสร้างดาต้าเบส คลัสเตอร์หรือมีการรันไฟล์ initdb.exe

รูปแบบทั้วไปของไฟล์ pg_hbb.conf คือกลุ่มของข้อความแยกแยะตามบรรทัดหนึ่งของข้อความจะต้องจบลงในหนึ่งบรรทัด บรรทัดว่างจะถูกข้ามไปโดยไม่มีการประมวลผล และข้อความที่อยู่หลังเครื่องหมาย # จะเป็นคำอธิบายและไม่ถูกประมวลผลเช่นกัน แต่ละบรรทัดประกอบหลายข้อความที่ถูกแยกแยะออกจากกันด้วยช่องว่าง ในแต่ละข้อความมีช่องว่างได้ แต่ต้องคร่อมด้วยเครื่องหมาย “ “

แต่ละบรรทัดจะรบุ ชนิดของการเชื่อมต่อ ช่วงหมายเลขไอพี (ip) ของเครื่องลูกข่าย ชื่อของฐานข้อมูลที่จะเข้าใช้งาน ชื่อ ผู้ใช้งาน และวิธีการตรวจสอบ กรณีมีหลายบบรทัดที่เข้าเงื่อนไข

บรรทัดแลกที่สอดคล้องเงื่อนไขจะถูกนำมาใช้สิทธิ ถ้าไม่ถูกต้องการเชื่อมต่อนั้นจถูกยอกเลิก โดยไม่สนใจบบรทัดอื่นที่กำหนดสิทธิแตกต่างกัน

รูปแบบของแต่ละบรรทัดสามารถเขียนได้ 5 รูปแบบ ตามตัวอย่าง

local database user authentication-method [authentication-option]

host database user CIDR-address authentication-method [authentication-option]

host database user IP-address IP-mask authentication-method [authentication-option]

hostssl database user IP-address IP-mask authentication-method[authentication-option]

hostnossl database user IP-address IP-mask authentication-method [authentication-option]

แต่ละข้อความจะมีความหมายดังนี้

Local เป็นการเชื่อมต่อในรูปแบบที่เครื่องบลูกข่าย (client) และฐานข้อมูลอยู่ในเครื่องเดียวกัน

Host เป็นการเชื่อมต่อของลูกข่ายโดยใช้ tcp/ip

Database กำหนดว่าจะเข้าใช้ฐานข้อมูลแบบไหน โดยระบุชื่อฐานข้อมูลสามารถใช้ค่าต่างๆ เหล่านี้

- all เพื่อระบุทุกฐานข้อมูล
- ชื่อฐานข้อมูล
- ถ้ากำหนดหลายฐานข้อมูลให้คั่นด้วยเครื่องหมาย , (comma)

ผู้ใช้งาน ระบุชื่อที่จะเข้าใช้งานฐานข้อมูล สามารถใช้คำเล่านี้แทนได้

- all เพื่อระบุทุกฐานข้อมูล
- ถ้ากำหนดหลาย ผู้ใช้งาน ให้คั่นแต่ละ ผู้ใช้งาน ด้วยเครื่องหมาย , (comma)
- ถ้าระบุเป็นชื่อของกลุ่ม ผู้ใช้งาน ให้นำชื่อกลุ่มด้วยเครื่องหมาย

CIDR – address (Classless inter-Domain Routing)กำหนดที่เครื่องไคลเอน ระบุหมายเลขไอพีของเครื่องไคลเอนเอง ที่จะขออนุญาติเข้าใช้งานรูปแบบก็คือ หมายเลข ไอพีที่อยู่ในรูปจุดทศนิยมตามด้วยหมายเลข CIDR เช่น 255.255.255.0 = 24 ดังตัวอย่าง 161.246.6.147/24

IP – address, IP – markเป็นทางเลือกแทนการระบุหมายเลข ip ของ client แบบ CIDR โดยวิธีนี้ จะระบุเป็นหมายเลข ip และ subnet เช่น 161.246.6.147 255.255.255.0

Authentication – method ระบุการตรวจสอบตัวตน ของ ผู้ใช้งาน ที่เชื่อมต่อเข้ามา สามารถใช้ตัวเลือกเหล่านี้ได้

- Trust ไม่มีเงื่อนไขในการตรวจสอบ คือยืนยอมให้ผู้ใช้งานทุกสามารถเข้าใช้งานได้ทุกฐานข้อมูลโดยไม่ต้องใส่ รหัสผ่าน

- Reject ปฏิเสธการเชื่อมต่อโดยไม่มีเงื่อนไขมีประโยชน์ ในการกรองบางเครื่องเพื่อไม่ให้เข้าใช้งาน server
- Md5 เครื่อง client ที่เข้าใช้งานต้องใส่รหัสผ่านในรูปแบบที่เข้ารหัสแบบ MD5 ทำให้ยากต่อการถอดรหัสเมื่อถูกดักข้อจับมูล
- Crypt คล้าย MD5 แต่รหัสผ่านจะถูส่งในรูปแบบที่เก่ากว่าสำหรับเครื่องที่มีเวอร์ชันต่ำกว่า 7.2
- Password คล้าย MD5 แต่รหัสผ่านจะถูกส่งในรูปแบบที่เข้าใจได้
- Krb4 ใช้ Kerberos เวอร์ชัน 4 ในการตรวจสอบตัวตนของ ผู้ใช้งาน สำหรับการเชื่อมต่อแบบ tcp/ip เท่านั้น
- Krb5 ใช้ Kerberos เวอร์ชัน 5 ในการตรวจสอบตัวตนของ ผู้ใช้งาน สำหรับการเชื่อมต่อแบบ tcp/ip เท่านั้น

Authentication – option ใส่อ็อปชันขึ้นกับว่า การเลือกการพิสูจน์ตัวตนแบบไหน

การกำหนดลำดับบรรทัดของข้อความ ก็มีความสำคัญต่อการตรวจสอบไฟล์ pg_hba.conf จะถูกตรวจสอบตามลำดับ โดยบรรทัดที่อยู่ก่อนจะถูกพิจารณาก่อนตามที่กล่าวไว้แล้วดังนั้นในกรณีที่เป็นการเชื่อมต่อฐานข้อมูลเดียวกันควรกำหนดให้บรรทัดต้นๆแรกมีเงื่อนไขการเชื่อมต่อที่ชัดเจนแต่มีการพิสูจน์ตัวตนที่ไม่ค่อยเข้มงวดมากนัก ในขณะที่ บรรทัดถัดไปมีการกำหนดเงื่อนไขที่ชัดเจนกว่าการระบุตัวตนที่เข้มงวดมากขึ้น เช่น ใช้ trust สำหรับ local connection แต่ใช้รหัสผ่านสำหรับ remote connection

ไฟล์ pg_hba.conf จะถูกอ่านในช่วง เริ่มการเรียกใช้โปรแกรมและเมื่อโพรเซส หลักของเซิร์ฟเวอร์หลัก ได้รับสัญญาณ เชื่อมต่อถ้ามีการแก้ไข ไฟล์นี้ในช่วงที่ระบบทำงานอยู่ จะต้องทำการสั่ง edb_postmaster ทำการอ่าน ไฟล์ใหม่ (โดยใช้pg_ctl reload หรือ kill - HUP)

Type	Database	User	IP-Address	Method	Option
☑ host	all	all	127.0.0.1/32	trust	
☐					

รูปที่ 4.41 กำหนดให้ เชื่อมต่อ โดยใช้ tcp/ip ภายในเครื่องเดียวกันและยินยอมให้ทุกผู้ใช้งานสามารถเข้าได้ ทุกฐานข้อมูลและไม่ต้องมีการ ใส่รหัสผ่าน

```
C:\Program Files\PostgreSQL\8.3\bin>psql -U postgres -d postgres
Welcome to psql 8.3.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=#
```

รูปที่ 4.42 ทดสอบโดย ใช้ผู้ใช้งาน postgres และ เข้าใช้งานฐานข้อมูล postgres

Type	Database	User	IP-Address	Method	Option
☑ host	all	all	127.0.0.1 255.255.255.255	password	
☐					

รูปที่ 4.43 เหมือนรูปที่ 16 แต่เปลี่ยนเป็นการใช้ ซับเน็ต มาร์คและมีการยืนยันตัวตนโดยใช้ password

```
C:\Program Files\PostgreSQL\8.3\bin>psql -U postgres -d postgres
Password for user postgres:
Welcome to psql 8.3.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=#
```

รูปที่ 4.44 การทดสอบโดย ใช้ ผู้ใช้งาน postgres และฐานข้อมูล postgres จะเห็นว่ามีรหัสผ่าน เพิ่มขึ้นมาให้ยืนยันตัว

Type	Database	User	IP-Address	Method	Option
☑ host	postgres	train0	127.0.0.1 255.255.255.255	md5	
☐					

รูปที่ 4.45 เป็นการกำหนดให้เชื่อมต่อได้เฉพาะผู้ใช้งาน train0 และเข้าใช้ฐานข้อมูลเท่านั้นแล้วมีการกำหนดการยืนยันตัว แต่มีการเข้ารหัสไว้เพื่อป้องกันการ ดักจับข้อมูล

```
C:\Program Files\PostgreSQL\8.3\bin>psql -U train1 -d postgres
psql: FATAL:  no pg_hba.conf entry for host "127.0.0.1", user "train1", database
 "postgres", SSL off

C:\Program Files\PostgreSQL\8.3\bin>psql -U train0 -d mediawiki
psql: FATAL:  no pg_hba.conf entry for host "127.0.0.1", user "train0", database
 "mediawiki", SSL off

C:\Program Files\PostgreSQL\8.3\bin>psql -U train0 -d postgres
Password for user train0:
Welcome to psql 8.3.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=#
```

รูปที่ 4.46 แสดงตัวอย่างการเข้าใช้งาน

ครั้งที่ 1 ผู้ใช้งาน train1 ผิด ฐานข้อมูล postgres ถูก ไม่สามารถเชื่อมต่อได้

ครั้งที่ 2 ผู้ใช้งาน train1 ถูก ฐานข้อมูล mediawiki ผิดไม่สามารถเชื่อมต่อได้

ครั้งที่ 3 ผู้ใช้งาน train0 ถูก ฐานข้อมูล postgres ถูก สามารถ เชื่อมต่อได้

การใช้เครื่อง client connect ด้วย tcp/ip

ขั้นตอนที่ 1 เข้าไปที่ file postgresql.conf แล้วค้นหาคำว่า listen_addresses

แก้จาก listen_addresses = 'localhost'

เป็น listen_addresses = '*'

เพื่ออนุญาติให้เครื่องอื่นสามารถเข้าใช้งานฐานข้อมูลบนเซิร์ฟเวอร์ได้

ขั้นตอนที่ 2 เข้าไปแก้ ip-address ให้ยอมรับ ip-address เครื่อง client ได้

ขณะที่เครื่อง client มี ip- address คือ 161.246.6.150/24 คือ การเช็คบิตของ ip-address ที่เข้ามา connect ในที่นี้คือ ยอมรับให้ ip-adddress ตั้งแต่ 161.246.6.1 ถึง 161.246.6.254 ให้สามารถ connect server ได

Type	Database	User	IP-Address	Method	Option
☑ host	all	all	161.246.6.150/24	md5	
☐					

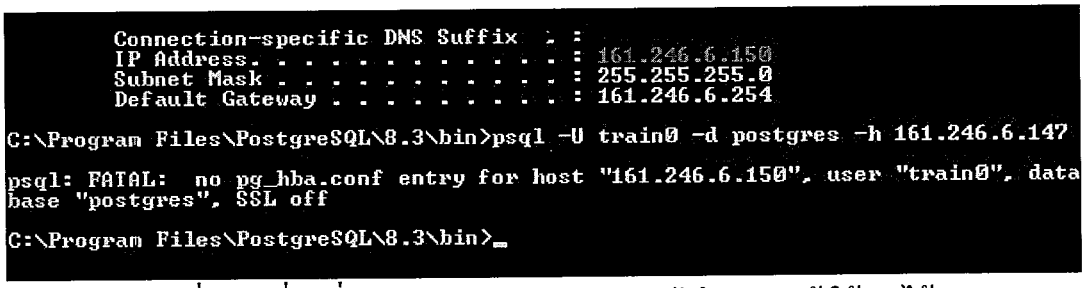
รูปที่ 4.47 การกำหนดให้เครื่อง server ยอมรับผู้ใช้งานเป็นบาง ip address

```
C:\Program Files\PostgreSQL\8.3\bin>psql -U train0 -d postgres -h 161.246.6.147
Password for user train0:
Welcome to psql 8.3.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=#
```

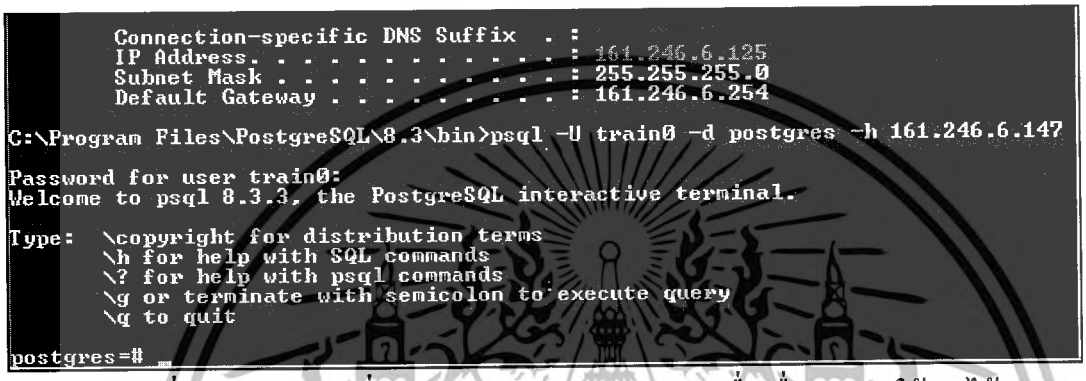
รูปที่ 4.48 เป็นการ connect จากเครื่อง client ip-address 161.246.6.150 และเข้ามายังเครื่อง server ซึ่งเป็น ip-address 161.246.6.147

Type	Database	User	IP-Address	Method	Option
☑ host	all	all	161.246.6.150/32	reject	
☑ host	all	all	0.0.0.0/0	md5	
☐					

รูปที่ 4.49 เป็นการมีการยอมรับทุก client ทุกๆ ip-address ยกเว้นip-address เดียว นั้นคือ 161.246.6.150



รูปที่ 4.50 เมื่อเครื่อง ip address 161.246.6.150 ไม่สามารถเข้าใช้งานได้

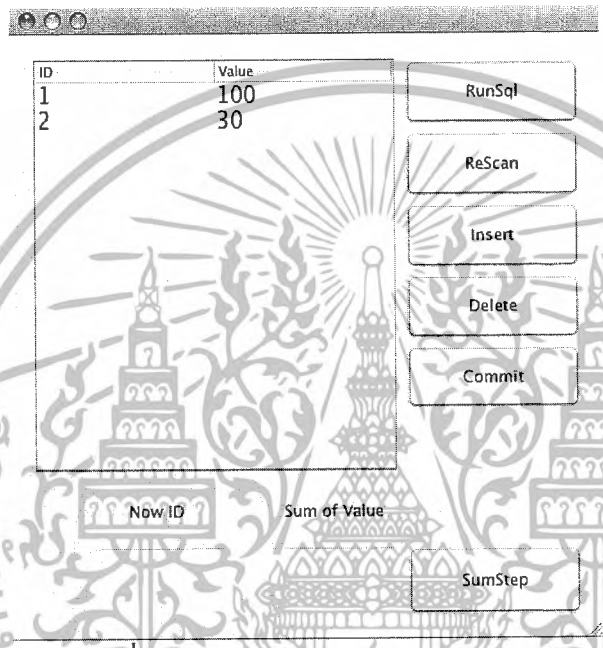


รูปที่ 4.51 นอกจากเครื่อง ip address 161.246.6.150 เครื่องอื่นสามารถใช้งานได้หมด

บทที่ 5

การทดลองและผลการทดลอง

โดยใช้เคอร์เซอร์ซึ่งจะทำการทดลองโดยเขียนโปรแกรมโดยใช้ภาษาจาวา และเรียกใช้เคอร์เซอร์ในการ insert, delete, update ซึ่งมีหน้าตาโปรแกรมดังนี้



รูปที่ 5.1 หน้าตาโปรแกรมในการทดสอบ

RunSql ทำการส่งคำสั่ง เอสคิวแอล ไป ดิวรี่ ใหม่

ReScan ทำการนำข้อมูลที่อยู่ในเคอร์เซอร์ทั้งหมดมาแสดงผล

Insert ทำการเพิ่มข้อมูลเข้าใน เคอร์เซอร์ (ไม่ได้ใช้คำสั่ง INSERT ใน เอสคิวแอล)

Delete ทำการลบข้อมูลแถวที่เคอร์เซอร์ชี้อยู่

คอมมิต สั่ง คอมมิต ผ่าน เคอร์เซอร์ ในกรณีที่ insert หรือ delete

SumStep ทำการหาผลรวมของค่าในคอลัมน์ทั้งหมด โดยการกด 1 ครั้งจะเป็นการ บวกเพิ่มทีละแถว โดยที่ NowID จะเป็นตัวบอกว่าขณะนี้เคอร์เซอร์ชี้อยู่ที่ไหน และ Sum of Value คือผลรวมตั้งแต่แถวแรกจนถึงแถวที่เคอร์เซอร์ชี้อยู่

5.1 การทดลองที่ 1 ผลิตภัณฑ์ออราเคิล

การใช้โปรแกรมทดสอบไอโซเลชันเลเวลกับผลิตภัณฑ์ ออราเคิล ในการ แก้ไขข้อมูลผ่านเคอร์เซอร์ จะไม่มีผลกับ order by คือ ลำดับของ row จะไม่เปลี่ยนแปลง แม้ คอลั่ม ที่สั่ง order by นั้น ลำดับจะเปลี่ยนไป

เคอร์เซอร์จะไม่เห็นการเปลี่ยนแปลงเมื่อผลลัพธ์ของ เอสคิวแอล มาจากหลายตาราง method refeshrow () จะใช้ไม่ได้ เกิด exception unsupported

5.1.1 ทรานแซกชัน ไอโซเลชันเลเวล Read-committed

5.1.1.1 Lost update problem

ตารางที่ 5.1 การทดสอบทรานแซกชันไอโซเลชันเลเวล Read-Committed กับการแก้ไขปัญหา Lost update problem กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
ID Value 1 10 2 20 3 30		เรียก โปรแกรมทดสอบ ของ ทรานแซกชัน A ขึ้นมาแสดง
	ID Value 1 10 2 20 3 30	เรียกโปรแกรมทดสอบ ของ ทรานแซกชัน B ขึ้นมาแสดง
ID Value 1 100 2 20 3 30		แก้ไขข้อมูลแถวที่ 1 ของ TxA ให้มีค่า =100 โดยยังไม่ คอมมิต
	ID Value 1 500 2 20 3 30	แก้ไขข้อมูลแถวที่ 1 ของ TxB ให้มีค่า =500 โดยยังไม่ คอมมิต เกิดการ รอ โปรแกรมใน TxB ไม่สามารถทำงานต่อได้

ตารางที่ 5.1 การทดสอบทรานแซกชั่ไอโซเลชัน เลเวล Read-Committed กับการแก้ไข ปัญหา Lost update problem กับผลิตภัณฑ์ ออราเคิล(ต่อ)

Tx A	Tx B	Status
		สั่งให้ Tx A คอมมิต โปรแกรมใน TxB สามารถทำงานต่อได้
		สั่งให้ Tx B คอมมิต
ID Value 1 500 2 20 3 30	ID Value 1 500 2 20 3 30	ผลลัพธ์ของทั้ง TxA และ TxB หลังจากกดปุ่ม rescan ปรากฏว่าได้ 500 เท่ากัน

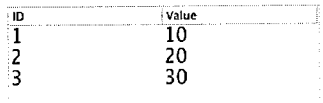
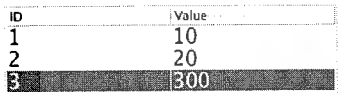
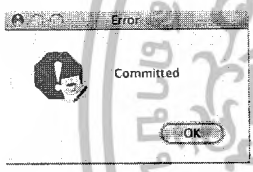
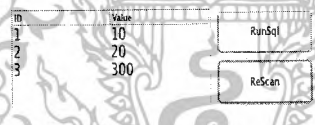
จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์ สามารถแก้ปัญหา lost update ได้ เพราะเมื่อมีการแก้ไขข้อมูลใน row เดียวกัน tx ที่เริ่มทำงานทีหลังจะต้องรอจนกว่า tx ที่ทำงานก่อนคอมมิต จึงจะแก้ไข ข้อมูลใน row เดียวกันนั้นได้

5.1.1.2 Uncommit dependency

ตารางที่ 5.2 การทดสอบทรานแซกชันไอโซเลชันเลเวล Read-Committed กับการแก้ไขปัญหา Uncommit dependency กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
ID Value 1 10 2 20 3 30		เรียก program ทดสอบ ของทรานแซกชัน A ขึ้นมาแสดง

ตารางที่ 5.2 การทดสอบทรานแซกชันไอโซเลชันเลเวล Read-Committed กับการแก้ไขปัญหา Uncommit dependency กับผลิตภัณฑ์ ออราเคิล(ต่อ)

Tx A	Tx B	Status
		เรียกโปรแกรม ทดสอบของ ทรานแซกชัน B ขึ้นมาแสดง
		แก้ไขข้อมูลใน row ที่ 3 ของ Tx A ให้ ค่า=300
		กดปุ่ม ReScan เพื่อ scan ดูข้อมูลในเคอร์เซอร์ อีกครั้ง จะเห็นว่าข้อมูลไม่มีการเปลี่ยนแปลง
		สั่ง Tx A คอมมิต
		กดปุ่ม ReScan เพื่อ scan ดูข้อมูลในเคอร์เซอร์อีกครั้ง จะเห็นว่าข้อมูลเปลี่ยนไป

จากการทดลองจะเห็นว่า การทำงานผ่าน เคอร์เซอร์ สามารถแก้ปัญหา uncommit dependency ได้ เพราะข้อมูลที่ถูกแก้ไขผ่านเคอร์เซอร์จะไม่ถูกมองเห็นจาก ทรานแซกชัน อื่น หรือ เคอร์เซอร์ อื่น จนกว่าจะมีการ คอมมิต

5.1.1.3 Inconsistent analysis problem

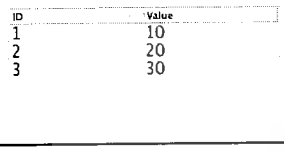
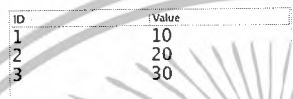
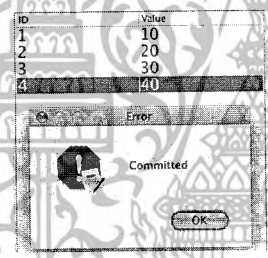
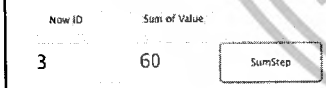
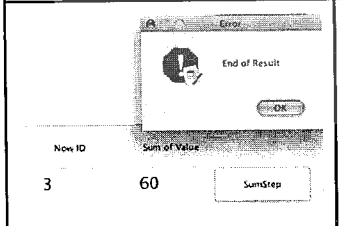
ตารางที่ 5.3 การทดสอบทรานแซกชันไอโซเลชันเลเวล Read-Committed กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
ID Value 1 10 2 20 3 30		เรียกโปรแกรมทดสอบ ของทรานแซกชัน A ขึ้นมาแสดง
	ID Value 1 10 2 20 3 30	เรียกโปรแกรม ทดสอบ ของทรานแซกชัน B ขึ้นมาแสดง
Now ID Sum of Value 1 10 SumStep		กดปุ่ม SumStep เพื่อทำการหาผลรวมของค่า ทีละ row
	ID Value 1 30 2 20 3 10 Error Committed OK	ทำการแก้ไขข้อมูลใน Tx B โดยจำลองการ โอนเงินจาก ID 3 ไป ID 4 จำนวน 20 และ คอมมิต
Now ID Sum of Value 2 30 SumStep		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์ คือ 30(10+20)
Now ID Sum of Value 3 40 SumStep		กดปุ่ม SumStep ครั้งที่ 3 ผลลัพธ์ คือ 40(10+20+10) ซึ่งเป็นผลลัพธ์ที่ผิดเพราะ ควรจะเป็น (10+20+30)=60

จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์ ไม่สามารถแก้ปัญหา Inconsistent analysis ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 10+20+30=60 ซึ่งไม่ควรเห็นการเปลี่ยนแปลงของข้อมูลขณะที่มีการวิเคราะห์ข้อมูลนั้นๆ

5.1.1.4 Phantom Phenomenom

ตารางที่ 5.4 การทดสอบทรานแซกชันไอโซเลชันเลเวล Read-Committed กับการแก้ไขปัญหา Phantom Phenomenom กับผลิตภัณฑ์ ออราเคิล

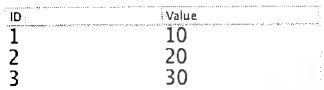
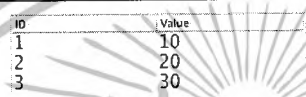
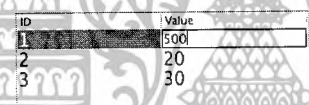
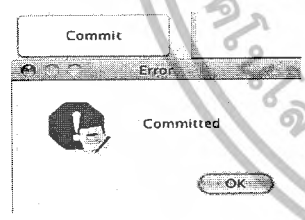
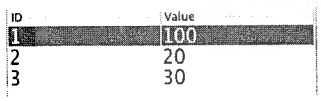
Tx A	Tx B	Status
		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
		กดปุ่ม SumStep เพื่อทำการหาผลรวมของ value ทีละ row
		ทำการเพิ่มข้อมูล ID 4 มี Value = 40 ลงใน Tx B
		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์คือ 30(10+20)
		กดปุ่ม SumStep ครั้งที่ 3 ผลลัพธ์คือ 60(10+20+30)
		กดปุ่ม SumStep ครั้งที่ 4 มีข้อความแจ้งว่า ไม่มีข้อมูลแล้ว หมายความว่า ใน TxA ไม่เห็นการเพิ่มข้อมูลใน TxB

จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์สามารถแก้ปัญหา phantom phenomenom ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 10+20+30=60 ซึ่งไม่ควรเห็นข้อมูลที่เพิ่มเข้ามาหลังจากเริ่ม ทรานแซกชัน แล้ว

5.1.2 ทรานแซกชัน ไอโซเลชันเลเวล Serializable

5.1.2.1. Lost update problem

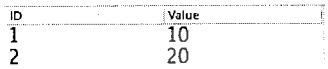
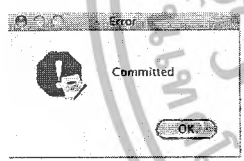
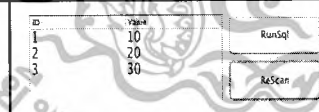
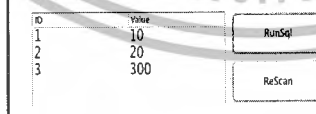
ตารางที่ 5.5 การทดสอบทรานแซกชันไอโซเลชันเลเวล Serializable กับการแก้ไขปัญหา Lost update problem กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
		เรียกโปรแกรม ทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรม ทดสอบ ของ TxB ขึ้นมาแสดง
		แก้ไขข้อมูลแถวที่ 1 ของ TxA ให้มี ค่า =100 โดยยังไม่ คอมมิต
		แก้ไขข้อมูลแถวที่ 1 ของ TxB ให้มีค่า =500 โดยยังไม่ คอมมิต เกิดการ รอโปรแกรมใน TxB ไม่สามารถทำงานต่อได้
		สั่งให้ Tx A คอมมิต แต่จะเกิด error can't serilize access ขึ้นที่ Tx B และ TxB จะถูก โรลแบ็ก
		ผลลัพธ์ของทั้ง TxA และ TxB หลังจาดกดปุ่ม rescan ปรากฏว่า Tx A เป็นข้อมูลใหม่ แต่ Tx B เป็นข้อมูลเก่า

จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์ สามารถแก้ปัญหา lost update ได้ แต่จะเป็นการเกิด ความผิดพลาด แทน เพราะลำดับการทำงานของปัญหา lost update นั้นไม่ config serializable schedule

5.1.2.2. Uncommit dependency

ตารางที่ 5.6 การทดสอบทรานแซกชันไอโซเลชันเลเวล Serializable กับการแก้ไขปัญหา Uncommit dependency กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
		เรียกโปรแกรม ทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
		แก้ไขข้อมูลใน row ที่ 3 ของ TxA ให้ ค่า=300
		กดปุ่ม ReScan เพื่อแสกน ดูข้อมูลในเคอร์เซอร์ อีกครั้ง จะเห็นว่าข้อมูลไม่มีการเปลี่ยนแปลง
		สั่ง Tx A คอมมิต
		กดปุ่ม ReScan เพื่อสแกน ดูข้อมูลในเคอร์เซอร์อีกครั้ง จะเห็นว่าข้อมูลยังคงเหมือนเดิม
		หลังจากกด คอมมิต แล้ว ส่งคำสั่งเอสคิวแอล เข้าไปใหม่จะเห็นว่าข้อมูลมีการเปลี่ยนแปลง

จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์ สามารถแก้ปัญหา Uncommit dependency ได้ เพราะข้อมูลที่ถูกแก้ไขผ่านเคอร์เซอร์จะไม่ถูกมองเห็นจาด ทรานแซกชัน อื่น หรือเคอร์เซอร์อื่น จนกว่าจะมีการ คิวรี่ คำสั่ง เอสคิวแอล ใหม่

5.1.2.3 Inconsistent analysis problem

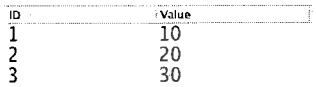
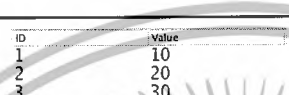
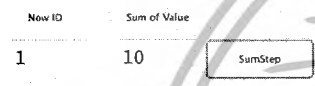
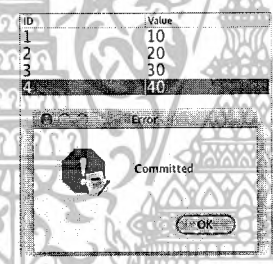
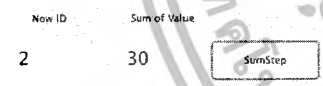
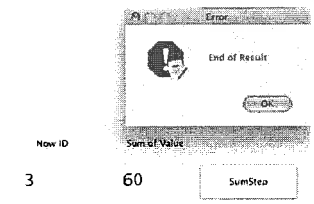
ตารางที่ 5.7 การทดสอบทรานแซกชันไอโซเลชันเลเวล Serializable กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ออราเคิล

Tx A	Tx B	Status
ID Value 1 10 2 20 3 30		เรียกโปรแกรม ทดสอบ ของ TxA ขึ้นมาแสดง
	ID Value 1 10 2 20 3 30	เรียกโปรแกรม ทดสอบ ของ TxB ขึ้นมาแสดง
Now ID Sum of Value 1 10 SumStep		กดปุ่ม SumStep เพื่อทำการหาผลรวมของค่าทีละ row
	ID Value 1 30 2 20 3 10 Error Committed OK	ทำการแก้ไขข้อมูลใน Tx B โดยจำลองการ โอนเงินจาก ID 3 ไป ID 4 จำนวน 20 และ คอมมิต
Now ID Sum of Value 2 30 SumStep		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์คือ 30(10+20)
Now ID Sum of Value 3 60 SumStep		กดปุ่ม SumStep ครั้งที่ 3 ผลลัพธ์คือ 40(10+20+30) ซึ่งเป็นผลลัพธ์ที่ควรจะเป็น

จากการทดลองจะเห็นว่า การทำงานผ่านเคอร์เซอร์สามารถแก้ปัญหา Inconsistent analysis ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 10+20+30=60 และไม่เห็นการเปลี่ยนแปลงของข้อมูลขณะที่มีการ analysis ข้อมูลนั้นๆ

5.1.2.4 Phantom Phenomenom

ตารางที่ 5.8 การทดสอบทรานแซกชันไอโซเลชันเลเวล Serializable กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ ออราเคิล

Tx A	Tx B	Status
		เรียกโปรแกรม ทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
		กดปุ่ม SumStep เพื่อทำการหาผลรวมของ value ทีละ row
		ทำการเพิ่มข้อมูล ID 4 มีค่า= 40 ลงใน Tx B
		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์คือ 30(10+20)
		กดปุ่ม SumStep ครั้งที่ 3 ผลลัพธ์คือ 60(10+20+30)
		กดปุ่ม SumStep ครั้งที่ 4 มีข้อความแจ้งว่า ไม่มีข้อมูลแล้ว หมายความว่า ใน TxA ไม่เห็นการเพิ่มข้อมูลใน TxB

จากการทดลองพบว่า การทำงานผ่าน สามารถแก้ปัญหา Phantom phenomenom ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 10+20+30=60 ซึ่งไม่ควรเห็นข้อมูลที่เพิ่มเข้ามาหลังจากเริ่ม ทรานแซกชัน แล้ว

สรุปการจัดการเรื่อง concurrency control ของออราเคิล 11จี ใน ไอโซเลชันเลเวล serizable เหมือนกันทั้งในการทำงานทรานแซกชันระดับคำสังเอสคิวแอลและทรานแซกชัน ระดับเคอร์เซอร์ แต่ในไอโซเลชันเลเวล readcommitted นั้นจะแตกต่างกันตรงที่ทรานแซกชันระดับคำสังเอสคิวแอล จะเห็นข้อมูลที่ถูกเพิ่มเข้ามาหลังเริ่มทรานแซกชันแต่ทรานแซกชันระดับเคอร์เซอร์นั้นจะไม่เห็นข้อมูลที่เพิ่มเข้ามาหลังจากเริ่มทรานแซกชัน ทั้งนี้เพราะ JDBC ไม่สามารถมองเห็นข้อมูลที่อยู่นอกเหนือเซ็ต ของผลลัพธ์ ของคำสั่ง เอสคิวแอล ที่ รีซอลต์เซ็ต ชี้อยู่ตอนเริ่ม ทรานแซกชันได้และถ้าตั้งค่าไอโซเลชันเลเวลใน interactive mode จะไม่มีผลกับการทำงานผ่านเคอร์เซอร์ซึ่งถ้าไม่มีการ ตั้งค่า Concurenncy ในเคอร์เซอร์ก็จะมี ค่าเริ่มต้นเป็น READ_COMMITTED

ตารางที่ 5.9 ตารางสรุปการใช้โปรแกรมทดสอบเคอร์เซอร์

Can see	Forward-only	Scroll-insensitive	Scroll-sensitive
Own inserts	NO	NO	NO
Own updates	YES	YES	YES
Own deletes	NO	YES	YES
Other's inserts	NO	NO	NO
Other's updates	NO	NO	YES
Other's deletes	NO	NO	NO

5.2 การทดลองที่ 2 ประยุกต์ใช้งานโปรแกรมกับไอโซเลชันเลเวลกับออราเคิล

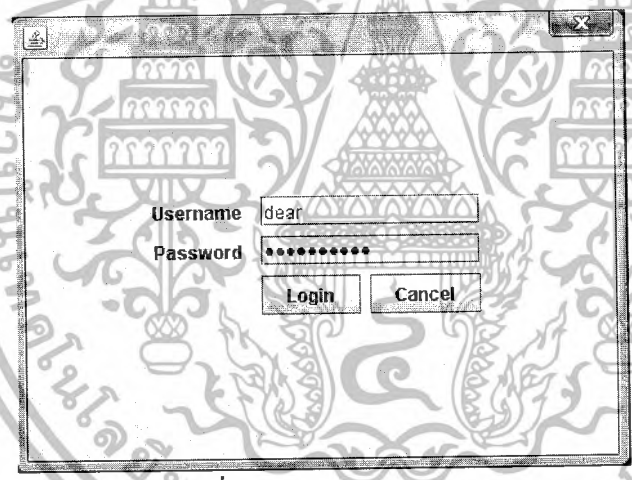
ประยุกต์ใช้งานโปรแกรมกับไอโซเลชันเลเวล(READ_COMMITTED และ Serailizable ของกับ ออราเคิล เป็นการนำหลักการของ ไอโซเลชันเลเวล ที่ได้ศึกษามาในตอนต้นซึ่งเป็นของ ออราเคิล 11จี มาทำการเขียนโปรแกรมเพื่อทดสอบว่าตรงกับหลักการที่ได้ศึกษามาเหรอเปล่า โดยการเขียนโปรแกรมที่เกี่ยวกับการแสดงการอัพเดทของตารางสายการบิน โดยการทดลองเซ็ต ไอโซเลชัน เลเวล เป็นทั้งSerialization กับ Read commited ซึ่งจะได้เห็นมามีผลแตกต่างกันอย่างไร แล้วตรงตามหลักการหรือไม่

5.2.1 โปรแกรมแสดงสายการบินของสนามบินการ

โปรแกรมแสดงสายการบินของสนามบินการ กับการใช้ตั้งค่า ทรานแซกชัน ไอโซเลชัน เลเวล READ-COMMITTED ของผลิตภัณฑ์ ออราเคิล ความสามารถของโปรแกรมมีดังนี้

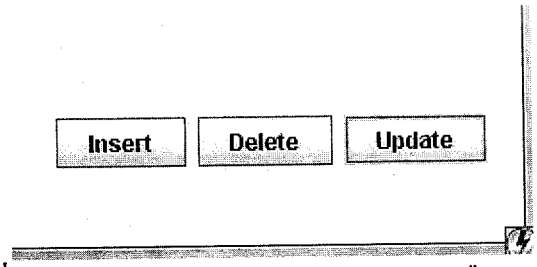
- สามารถที่จะให้ ยูสเซอร์เข้ามาทำการ Insert, Delete และ Update ได้
- มีการแสดงตารางให้เห็นถึงการเปลี่ยนแปลงของตารางสายการบิน หากมีการอัพเดทเกิดขึ้น
- หากเราทำการเซ็ตไอโซเลชันเลเวล เป็น Serialization จะไม่สามารถเห็นการเปลี่ยนแปลงของตารางสายการบินได้
- หากเราทำการเซ็ตไอโซเลชันเลเวลเป็น Read Commited จะสามารถเห็นการเปลี่ยนแปลงของตารางสายการบินได้

5.2.2 การทดสอบโปรแกรมแสดงสายการบินของสนามบิน



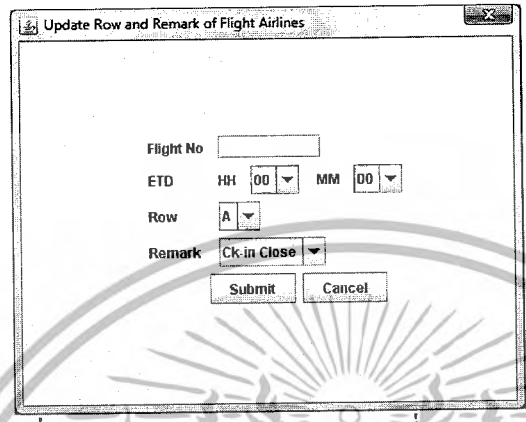
รูปที่ 5.2 หน้าต่างให้ Login

1. ทำการล็อกอินเข้าสู่ระบบ



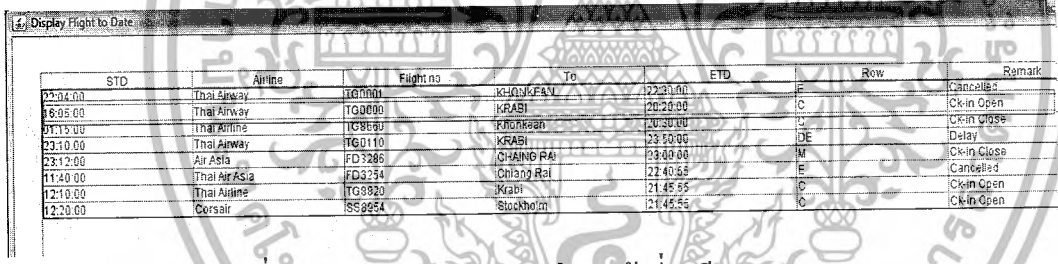
รูปที่ 5.3 หน้าต่างให้เลือกการ Insert, Delete หรือ Update

2. จากนั้นก็เลือกในส่วนที่เราต้องการทำซึ่งจะเป็น Insert, Delete หรือ Update แต่ในการทดลองนี้เลือก Update



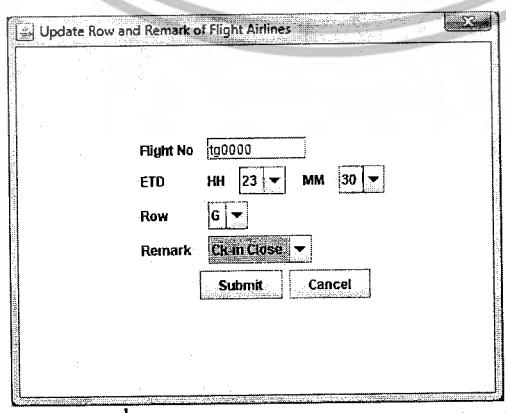
รูปที่ 5.4 หน้าต่างให้กรอกรายละเอียดที่จะ Update

3. จากนั้นจะให้กรอกรายละเอียดที่จะทำการ Update ดังรูปที่ 5.4



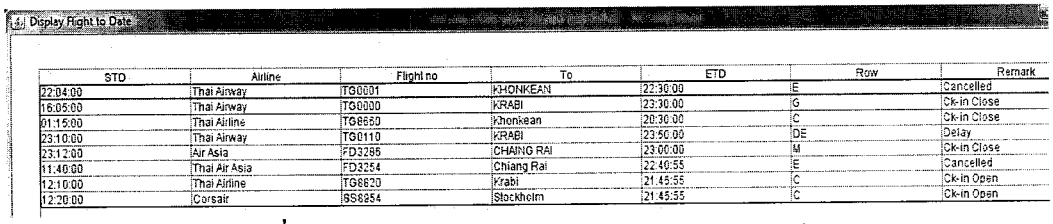
รูปที่ 5.5 หน้าต่างการแสดงผลก่อนหน้าที่จะมีการ Update

4. นี้จะเป็นส่วนของการแสดงผลก่อนหน้าที่ยังไม่มีการ Update



รูปที่ 5.6 หน้าต่างที่ทำการกรอกรายละเอียดเสร็จเรียบร้อยแล้ว

5. จากนั้นก็ทำการกรอกรายละเอียดที่ต้องการ Update ดังรูปข้างบน



STD	Airline	Flight no	To	ETD	Row	Remark
22:04:00	Thai Airway	TG0001	KHONKEAN	22:30:00	E	Cancelled
16:05:00	Thai Airway	TG0000	KRABI	23:30:00	G	Ck-in Close
01:15:00	Thai Airline	TG6650	Khonkean	20:30:00	C	Ck-in Close
23:10:00	Thai Airway	TG0110	KRABI	23:50:00	DE	Delay
23:12:00	Air Asia	FD3286	CHAING RAI	23:00:00	M	Ck-in Close
11:40:00	Thai Air Asia	FD3254	Chiang Rai	22:40:55	E	Cancelled
12:10:00	Thai Airline	TG6620	Krabi	21:45:55	C	Ck-in Open
12:20:00	Corsair	SS8854	Stockholm	21:45:55	C	Ck-in Open

รูปที่ 5.7 หน้าต่างแสดงผลหลังจากมีการ Update แล้ว

6. จะเห็นว่ามีการเปลี่ยนแปลงเกิดขึ้นในคอลัมน์ที่ชื่อ ETD, Row และ Remark

5.2.3 โปรแกรมจองตั๋วรถไฟ

โปรแกรมจองตั๋วรถไฟมีการตั้งค่า ทรานแซกชัน ไอโซเลชันเลเวลเป็น Serializable ติดต่อฐานข้อมูล ออราเคิล กับ เป็นการประยุคใช้ ทรานแซกชัน ไอโซเลชัน เลเวล Serializable ไปใช้เพื่อช่วยควบคุมความถูกต้องของข้อมูลในโปรแกรมประเภท Multiuser และเครื่องมือเกี่ยวกับการรักษาความปลอดภัยของข้อมูลไปใช้ในโปรแกรม ซึ่ง โปรแกรมที่พัฒนาขึ้นมานั้นเป็น โปรแกรมขายตั๋วรถไฟซึ่งสามารถใช้ได้ในทุกๆสถานีและจองที่นั่งที่ว่างระหว่างทางได้

ความสามารถของโปรแกรม

- มีการจำกัดระดับการเข้าใช้งานของผู้ใช้ ที่เป็นคนขายตั๋ว อย่างเดียว และ ผู้ใช้ที่สามารถกำหนดเวลาและเว้นทางเดินรถได้
- สามารถจองที่นั่งเดียวกัน เส้นทางเดียวกัน แต่สถานีต้นทาง และ ปลายทางจะต้องไม่เหลื่อมกัน
- ในกรณีที่มีการจองที่นั่งที่เดียวกัน เส้นทางเดียวกัน และ สถานีต้นทาง และสถานีปลายทาง เหลื่อมกันจะมีการแจ้งเตือนให้ทราบ

5.2.4 เครื่องมือของ ดีบีเอ็มเอส ที่นำมาช่วยในการพัฒนาโปรแกรม

ทริกเกอร์ (Trigger) ในขณะที่ผู้ใช้กำหนดเส้นทางเดินรถจะใช้ทริกเกอร์เพิ่มข้อมูลที่นั่งของแต่ละโบกี้ โดยอัตโนมัติ โดยที่ไม่ต้องเขียนโปรแกรมจัดการเอง

```
create or replace
TRIGGER "AUTOCREATESEAT" AFTER
INSERT ON trainDB.train_dep FOR EACH ROW DECLARE i NUMBER := 1;
bogie_rec bogie%ROWTYPE;
n NUMBER;
```

```
x NUMBER;
s_pk NUMBER;
CURSOR c1  IS   SELECT * FROM trainDB.bogie td WHERE td.t_id=:new.tr_id;
CURSOR c2  IS   SELECT * FROM railroad ;
BEGIN
OPEN c1;
LOOP
FETCH c1 INTO bogie_rec;
EXIT
WHEN c1%NOTFOUND;
   n    :=bogie_rec.percent_price;
   x    := bogie_rec.seat_count;
FOR i IN 1 .. x
LOOP
   s_pk:= seatpk;
INSERT  INTO seat  ( DEP_DATE,T_ID,CHK, B_ID,SEAT_NO,seat_id)
VALUES (:new.dep_date ,:new.tr_id,'n',bogie_rec.b_id,i,s_pk );
        createstation(s_pk,:new.destination,:new.origin,:new.flag);
        END LOOP;
        END LOOP;
CLOSE c1;
END;
```

ใช้ SEQUENCES + ทริกเกอร์ ในการ INSERT ข้อมูลที่เป็นประเภท หมายเลขอัตโนมัติ เช่น หมายเลขตั๋ว

```
CREATE SEQUENCE "SEQ_TICKET"  MINVALUE 1 MAXVALUE
99999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20
ORDER  CYCLE ;
CREATE OR REPLACE TRIGGER "AUTOTICKETID" before
INSERT ON TICKET FOR EACH row DECLARE seq_val NUMBER;
BEGIN
```

```
SELECT SEQ_TICKET.nextval INTO seq_val FROM dual;
:new.TIC_ID := seq_val;
END;
```

กำหนดให้ผู้ใช้ ใช้ยูสเซอร์ของ ดีบีเอ็มเอส เพื่อจำกัดความปลอดภัยในการเข้าถึงข้อมูลโดยที่ไม่ต้องเขียนโปรแกรมจัดการ และกำหนดสิทธิ์ให้แก่ยูสเซอร์ โดยการใช้ กฎและการกำหนดสิทธิของ ดีบีเอ็มเอส เช่น

```
CREATE ROLE "TRAIN_SEALER" NOT IDENTIFIED
GRANT SELECT ON "TRAINDB"."BOGIE" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."RAILROAD" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."SEAT" TO "TRAIN_SEALER"
GRANT UPDATE ON "TRAINDB"."SEAT" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."STATION" TO "TRAIN_SEALER"
GRANT UPDATE ON "TRAINDB"."STATION" TO "TRAIN_SEALER"
GRANT DELETE ON "TRAINDB"."TICKET" TO "TRAIN_SEALER"
GRANT INSERT ON "TRAINDB"."TICKET" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."TICKET" TO "TRAIN_SEALER"
GRANT UPDATE ON "TRAINDB"."TICKET" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."TRAIN" TO "TRAIN_SEALER"
GRANT SELECT ON "TRAINDB"."TRAIN_DEP" TO "TRAIN_SEALER"
GRANT TRAIN_SEALER TO "USER"
```

จะเป็นว่ายูสเซอร์ ไม่ได้รับสิทธิ์ให้ INSERT หรือ UPDATE ตาราง TRAIN_DEF เพราะยูสเซอร์เป็นเพียงแค่คนขายตั๋วไม่สามารถกำหนดเส้นทางเดินรถได้

การล็อกอินเข้าใช้งานของยูสเซอร์ซึ่ง โดยทั่วไปแล้วผู้พัฒนามักจะสร้างตารางที่เก็บยูสเซอร์ และ รหัสเข้าใช้งานขึ้นมาแต่ในการทดสอบนี้จะระบุ ยูสเซอร์ และ รหัสเข้าใช้งานไปพร้อมกับการเชื่อมต่อฐานข้อมูลเลย โดยสามารถเขียนโค๊ด ได้ดังนี้

```
try {
        Class.forName(getDriver());
        conn = DriverManager.getConnection(getConnection(), user, pass);
```

```
} catch (Exception e) {
   throw new SQLException("User/Password ผิด ไม่สามารถเข้าใช้งานได้");
}
```

ถ้าไม่สามารถล๊อกอิน ได้โปรแกรมก็จะแจ้งให้ผู้ใช้ทราบ ซึ่งในการพัฒนาเพียงแค่นำความผิดพลาด ที่ได้ไปแสดงผลให้ผู้ใช้ดูเท่านั้น

การทดสอบสิทธิ์การเข้าใช้งานส่วนใดบ้างนั้นทำได้โดยการใช้คำสั่งที่ละเมิดสิทธิ์ที่มี เช่น

```
public void hasPrivilage() throws SQLException {
    db_connect.setAutoCommit(false);
    String testPrivilage = "INSERT INTO TRAINDB.STATION (SEAT_ID,
STATION, AVALIBLE) VALUES ('69777', 'ลำพูน', 'n')";
    prepare = db_connect.prepareStatement(testPrivilage);
    prepare.execute();
    db_connect.rollback();
  }
```

จากตัวอย่างโค๊ด ถ้าล็อกอิน เป็นยูสเซอร์ ธรรมดา โปรแกรม ก็จะแจ้งเตือนในส่วนนี้ แต่ถ้า ล็อกอินด้วยยูสเซอร์ที่มีสิทธิ์กำหนดเส้นทางเดินรถ ก็จะไม่มีแจ้งเตือนเกิดขึ้น และสามารถใช้งานในส่วนต่อไปได้

การใช้ ไอโซเลชัน เลเวล Readcommitted และ Serializable มาช่วยในการควบคุมความถูกต้องของข้อมูล ในการทดสอบนั้น จะใช้เซสชัน ที่มี ไอโซเลชัน เลเวล Readcommitted เป็น เซสชันที่ใช้ในการแสดงผลที่ยูสเซอร์ A ว่าที่นั่งไหนว่าง หรือถูกจองไปแล้ว ซึ่งถ้า ยูสเซอร์ B อื่นจองตั๋วในโบกี้เดียวกันกับที่ยูสเซอร์ A กำลังจองอยู่เสร็จแล้วก็จะแสดงผลให้ ยูสเซอร์ A เห็น ซึ่งจะใช้วิธี สร้างเทรด ขึ้นมาในส่วยการแสดงผลทำการวน สแกนเคอร์เซอร์ที่อ้างอิงถึงที่นั่งต่างๆในโบกี้

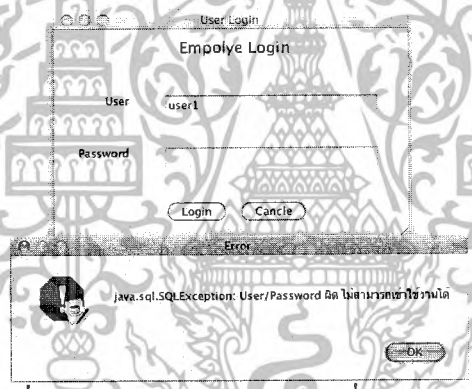
```
Class.forName(conn.getDriver());
Connection read_conn = DriverManager.getConnection(conn.getConnection(),
conn.getUser(), conn.getPass());
seat_view_thread = new ThreadSeat(jP_Seat, vec_jtb, read_conn, sql);
seat_view_thread.start();
```

ถ้ายูสเซอร์เลือกที่นั่งก็จะใช้เซสชันที่มีไอโซเลชันเลเวล Serializable ทำหน้าที่ UPDATE ข้อมูลในตารางที่นั่งของแต่ละโบกี้ซึ่งจะยังไม่คอมมิต จนกว่ายูสเซอร์จะกดตกลงซื้อตั๋วทั้งหมดที่เลือกไว้ซึ่งในระหว่างนี้อาจมียูสเซอร์อื่นตกลงซื้อตั๋วพร้อมๆกันซึ่งอาจทำให้เกินความผิดพลาดได้ ตั๋วที่นั่งซ้ำกันในเส้นทางเดียวกัน ได้แต่ในการทดสอบนี้จะไม่เกิดขึ้นเพราะในกรณีแบบนี้ไอโซเลชันเลเวล Serializable จะแจ้ง error can't serializable access ขึ้นซึ่งช่วยป้องกันปัญหาดังกล่าว

```
connection = conn.getConnObject();
connection.setTransactionIsolation(connection.TRANSACTION_SERIALIZABLE);
connection.setAutoCommit(false);
```

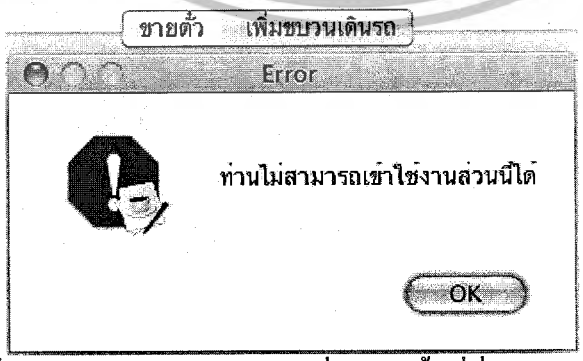
5.2.4 การทดสอบโปรแกรมจองตั๋วรถไฟ

a) หน้าจอการ Login เมื่อเกิดการ Login โดยใช้ User หรือ Password ที่ผิด



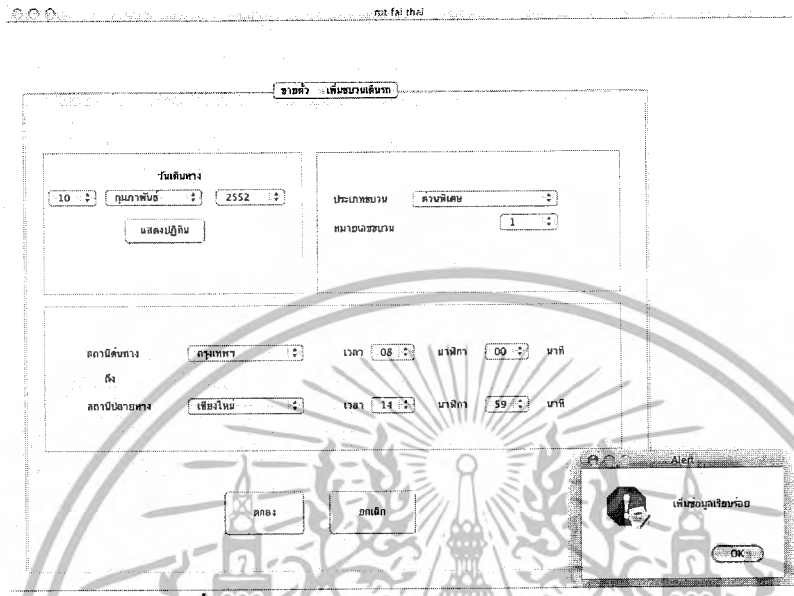
รูปที่ 5.8 หน้าจอ ความผิดพลาด เมื่อล็อกอินผิด

b) การทดลองเข้าใช้งานในส่วนที่ ยูสเซอร์นั้นไม่มีสิทธิ์เข้าใช้



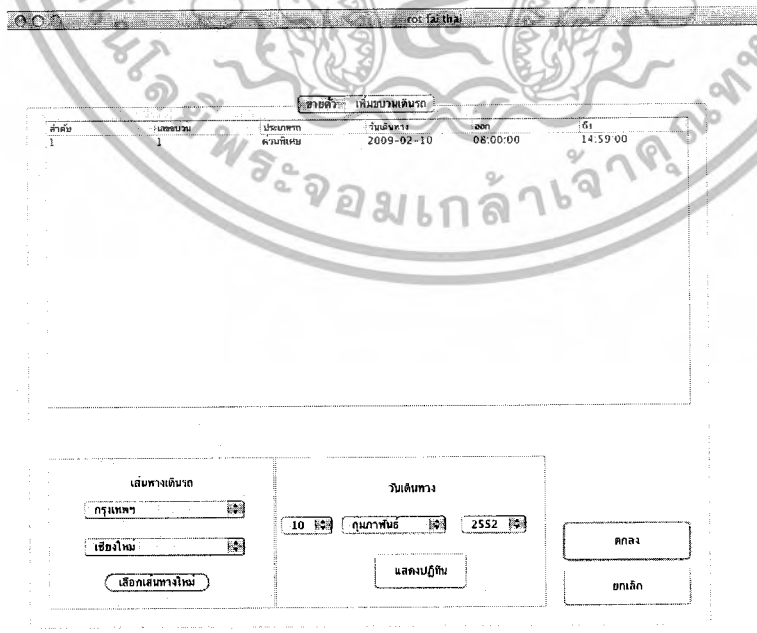
รูปที่ 5.9 หน้าจอ ความผิดพลาด เมื่อเข้าใช้พื้นที่ที่ไม่ได้รับสิทธิ์

สำหรับยูสเซอร์ ที่มีสิทธิ์ใช้งานก็จะเห็นหน้าต่างการ กำหนดเส้นทางเดินรถ ว่ารถขบวนไหน วิ่งจากสถานีไหน ไปยังสถานีไหน และเวลาในการเดินทาง



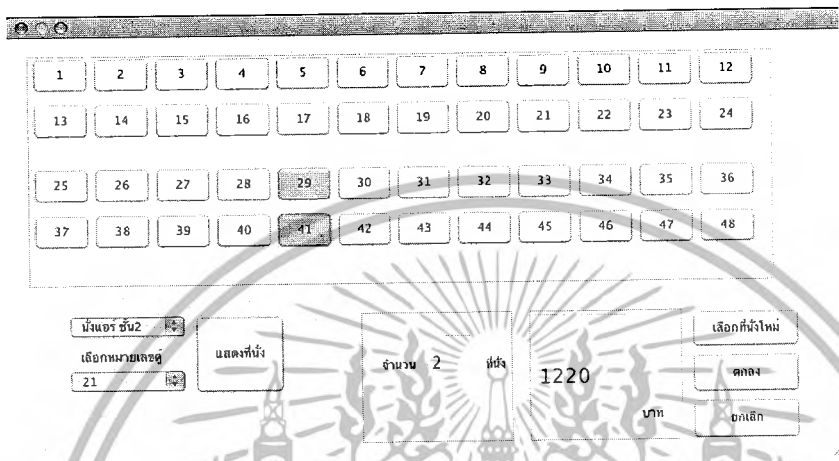
รูปที่ 5.10 หน้าต่างการกำหนดเส้นทางเดินรถ

c) การใช้งานในส่วนของการจองตั๋วรถ โดยทำการเลือกสถานีต้นทาง สถานีปลายทาง และวันที่เดินทาง โปรแกรมก็จะแสดงขบวนรถที่ตรงตามต้องการมาให้



รูปที่ 5.11 หน้าต่างแสดงขบวนรถเที่ยวที่ตรงกับความต้องการ

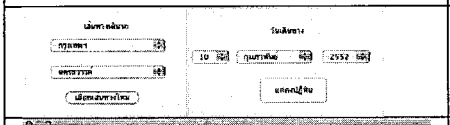
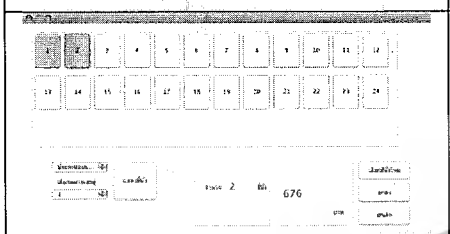
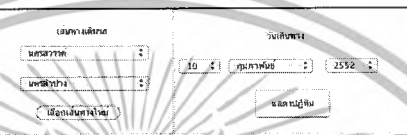
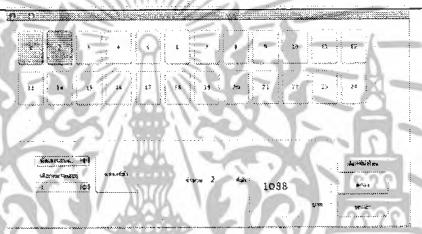
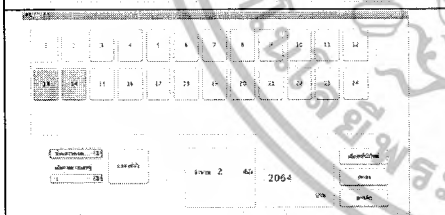
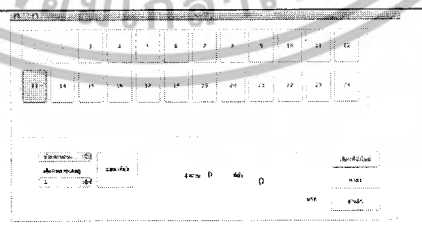
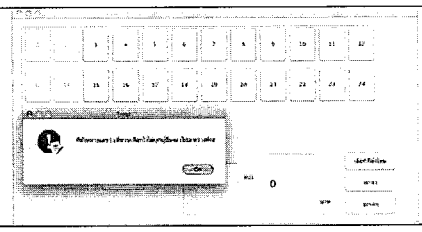
และเมื่อกดคลิก ที่ขบวนรถ ก็จะปรากฏหน้าต่างให้เลือกโบกี้และที่นั่งที่ต้องการจองซึ่งในการเลือกที่นั่งแต่ละครั้งจะทำให้เกิดอิเว็น และเซสชันที่มี ไอโซเลชันเลเวล serializable จะทำการ UPDATE ข้อมูลของที่นั่งนั้นๆ แต่ยังไม่ คอมมิต จนกว่าจะกดตกลง



รูปที่ 5.12 หน้าต่างการเลือกที่นั่ง

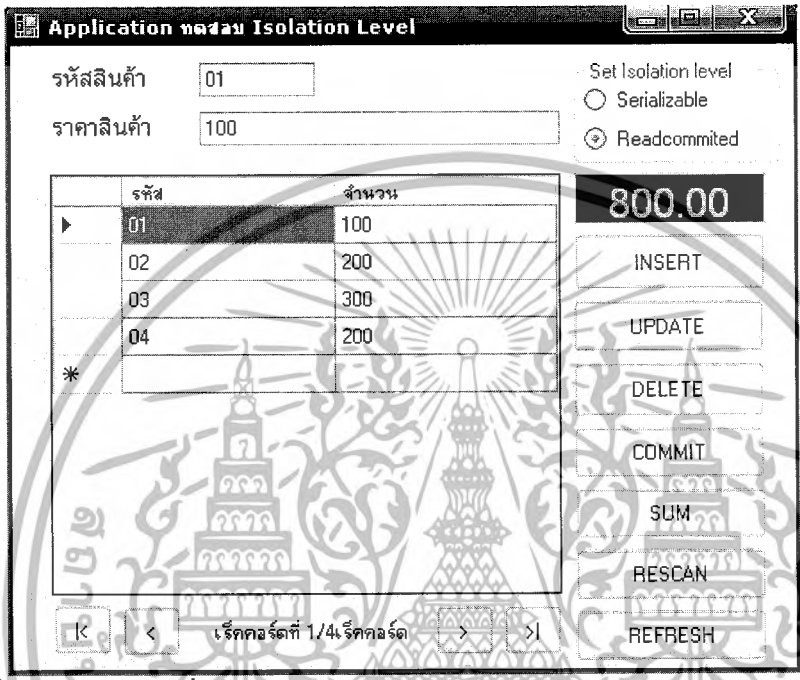
d) การทดสอบการจองที่นั่งเดียวกัน ในเส้นทางเดียวกัน พร้อมกัน ซึ่งเหตุการณ์แบบนี้จะเป็นปัญหา Lost Update ในมุมมองของ Concurrency control

ตารางที่ 5.10 การทดลองทำ ทรานแซกชัน 2 ทรานแซกชัน พร้อมๆ กัน

User A จองที่ กรุงเทพฯ	User B จองที่นครสวรรค์	สถานะ
		
		ยูสเซอร์ A คอมมิต
		
		ยูสเซอร์ B สามารถเลือกที่นั่งเดิมได้เพราะสถานีเดินทางไม่เหลื่อมกับยูสเซอร์ A
		
		
		ยูสเซอร์ B จะรอให้ ยูสเซอร์ A คอมมิต หรือ โรลแบ็ก ก่อน
User A กดตกลง (Commit)		แจ้งให้ ยูสเซอร์ B ทราบว่า มีคนจองก่อนแล้ว

5.3 การทดลองที่ 3 การใช้โปรแกรมทดสอบ ไอโซเลชัน เลเวล กับผลิตภัณฑ์ โพสเกรส เอสคิวแอล

ทำการทดลองโดยเขียนโปรแกรมโดยใช้ภาษา วิชวลเบสิกดอทเน็ต (vb.net) และเรียกใช้เคอร์เซอร์ในการ insert,delete,update ซึ่งมีหน้าตา โปรแกรมดังนี้



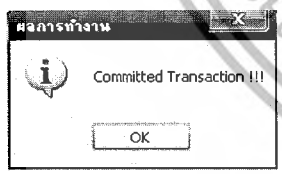
รูปที่ 5.13 โปรแกรมที่ใช้ในการทดสอบ ไอโซเลชัน เลเวล ของ โพสเกรสเอสคิวแอล

Set Isolation level	เป็นการเซ็ตค่า ไอโซเลชัน เลเวล ที่ต้องการในการทำการทดลอง
Insert	ทำการ Insert ข้อเป็น เอสคิวแอล ลง ในฐานข้อมูล
Update	ทำการ Update ข้อเป็นเอสคิวแอลลงในฐานข้อมูลในrow ที่เคอร์เซอร์ชี้อยู่
Delete	ทำการลบข้อมูลแถวที่ เคอร์เซอร์ชี้อยู่
Commit	ทำการยืนยันข้อมูล ที่ต้องการให้ลงฐานข้อมูลทันที
Sum	ทำการหาผลรวมของค่าทั้งหมด โดยการกด 1 ครั้งจะเป็นการ บวกเพิ่มทีละแถว โดยที่จะเป็นตัวบอกว่าขณะนี้ เคอร์เซอร์ชี้อยู่ที่ไหน และ มีการแสดงผลรวมตั้งแต่แถวแรกจนถึงแถวที่ เคอร์เซอร์ ชี้อยู่
Rescan	ทำการโหลดข้อมูลจากฐานข้อมูลขึ้นมาใหม่
Refresh	กลับไปสู่ค่าเริ่มต้นของฐานข้อมูลที่ได้โหลดมา
Cursor	เป็นการเลือก Row ที่ต้องการเพื่อทำการเปลี่ยนแปลงข้อมูล

5.3.1 ทรานแซกชันไอโซเลชันเลเวล Read-Committed

5.3.1.1 Lost update problem

ตารางที่ 5.11 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Lost update problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 400		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
	รหัส จำนวน 01 100 02 200 03 300 04 400	เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 100		แก้ไขข้อมูลแถวที่รหัส04 ของ TxA ให้มี จำนวน = 100 โดยยังไม่ คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 200	แก้ไขข้อมูลที่รหัส 04 ของ TxB ให้จำนวน = 200 โดยยังไม่ คอมมิต เกิดการ รอโปรแกรมใน TxB ไม่สามารถทำงานต่อได้
		สั่งให้ Tx A คอมมิตโปรแกรมใน TxB สามารถทำงานต่อได้
		สั่งให้ Tx B คอมมิต
รหัส จำนวน 01 100 02 200 03 300 04 200	รหัส จำนวน 01 100 02 200 03 300 04 200	ผลลัพธ์ของทั้ง TxA และ TxB หลังจาดกดปุ่ม rescan ปรากฏว่าได้ 200 เท่ากัน

จากการทดลอง นี้จะเห็นได้ว่า การตั้งค่าไอโซเลชัน เลเวล เป็น Read-Committed สามารถแก้ไขปัญหา lost update problem ได้ เนื่องจาก TxA เป็น ทรานแซกชัน ที่มาก่อน จะทำกการ Lock Row 04 ไว้ TxB จะเข้ามาทำการแก้ไขไม่ได้ จนกว่า TxA จะทำงานเสร็จ จึงทำให้ไม่มีการรบกวนการทำงานกัน

5.3.1.2 Uncommit dependency

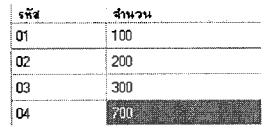
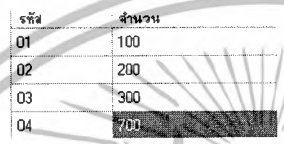
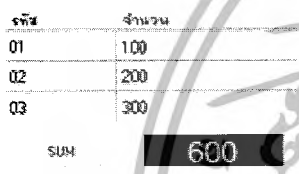
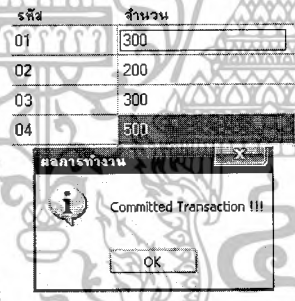
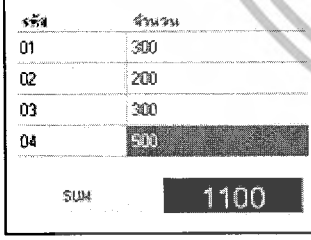
ตารางที่ 5.12 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา uncommit dependency กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 200		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 700		แก้ไขข้อมูลที่ รหัส 04 ของ Tx A ให้ จำนวน = 700
	รหัส จำนวน 01 100 02 200 03 300 04 200	จากนั้นเรียกโปรแกรมทดสอบ ของ TxBขึ้นมาแสดงจะเห็นว่า ที่รหัส 04 ยังคงมี จำนวนเท่ากับ 200 อยู่
ผลการทำงาน Committed Transaction !!! OK		สั่ง Tx A คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 700	กดปุ่ม ReScan เพื่อสแกนดูข้อมูลใน อีกครั้ง จะเห็นว่าข้อมูลใน TxA เปลี่ยนไป

จากการทดลองจะเห็นว่า การตั้งค่าไอโซเลชัน เลเวล เป็น Read-Committed สามาแก้ไขปัญหา uncommit dependency ได้ สังเกตได้จากเมื่อ TxA ทำการ Update แล้วแต่ยังไม่ คอมมิต ที่ TxB ก็จะยังไม่สามารถเห็นการเปลี่ยนแปลงที่ Row นั้น จนกว่า TxA จะทำการยืนยัน คอมมิต ลงฐานข้อมูล TxB ก็จะสามารถมองเห็นการแปลงแปลง

5.3.1.3 Inconsistent analysis problem

ตารางที่ 5.13 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
		กดปุ่ม Sum 2 ครั้ง เพื่อทำการหาผลรวมของ คอลัม จำนวน จะได้ (100+200+300) = 600
		ทำการแก้ไขข้อมูลใน Tx B โดยจำลองการ โอนเงินจาก รหัส04 ไป รหัส 01 จำนวน 200 และ คอมมิต
		กดปุ่ม Sum ครั้งที่ 3 ผลลัพธ์ คือ 900+500 = 1100

จากการทดลองนี้ จะเห็นได้ว่า การคำนวนหาผลรวมของจำนวนเงินที่มีในระบบทั้งหมดผิดไป จการทำงานที่ถูกต้องคือ 100+200+300+700 ผลรวมที่ได้ คือ 1,300 แต่ในการทดลองนี้ ได้มีการโอนย้ายจำนวนเงินภายในฐานข้อมูลเดียวกันเองขณะที่มีการหาผลรวม เมื่อมา Sum ครั้งสุดท้ายที่รหัส 04 จึงได้100+200+300+500 = 1100 ดังนั้น ไอโซเลชัน เลเวล Read-Committed ไม่สามารถแก้ไขปัญหาเรื่อง inconsistent analysis problem ได้

5.3.1.4 Phantom Phenomenom

ตารางที่ 5.14 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Phantom Phenomenom กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

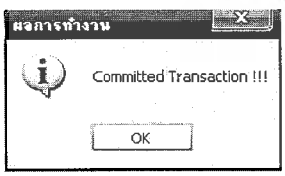
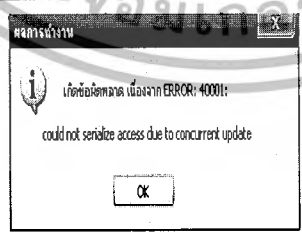
Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 700		เรียกโปรแกรมทดสอบของ TxAขึ้นมาแสดง
	รหัส จำนวน 01 100 02 200 03 300 04 700	เรียกโปรแกรมทดสอบของ TxBขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 SUM 600		กดปุ่ม SumStep เพื่อทำการหาผลรวมของ value ทีละ row
	รหัส จำนวน 01 400 02 200 03 300 04 700 05 300	ทำการเพิ่มข้อมูล ID 4 มี Value = 40 ลงในTxB
รหัส จำนวน 01 400 02 200 03 300 04 700 05 300 SUM 1600		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์ คือ 30(10+20)

จากการทดลองจะเห็นว่า การทดลอง ไม่สามารถแก้ปัญหา phantom phenomenom ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 100+200+300+700+300 = 1600 ซึ่งไม่ควรเห็นข้อมูลที่เพิ่มเข้ามาหลังจากเริ่ม ทรานแซกชัน แล้ว

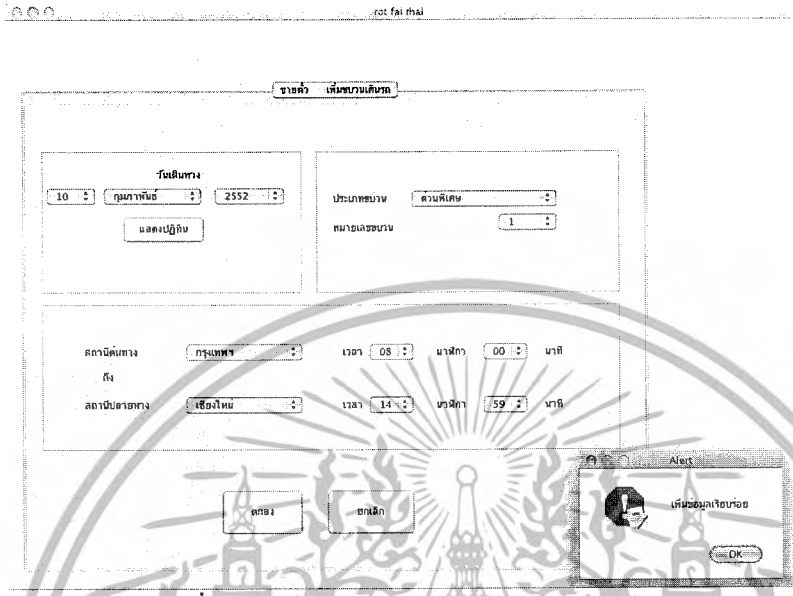
5.3.2 ทรานแซกชันไอโซเลชันเลเวล Serializable

5.3.2.1 lost update problem

ตารางที่ 5.15 การทดสอบ ทรานแซกชัน Isolation Level Serializable กับการแก้ไขปัญหา lost update problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

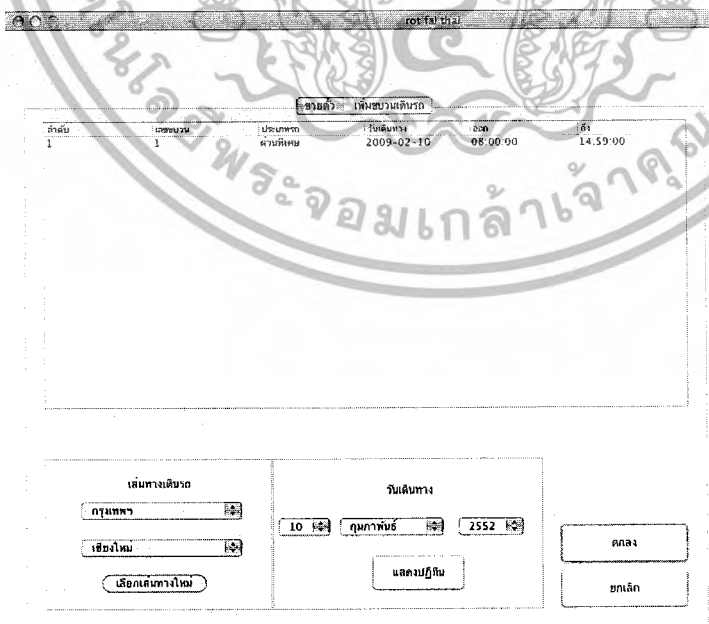
Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 400		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
	รหัส จำนวน 01 100 02 200 03 300 04 400	เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 100		แก้ไขข้อมูลที่รหัส 04 ของ TxA ให้แก้ไข จำนวน = 100 โดยยังไม่ คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 200	แก้ไขข้อมูลที่รหัส 04 ของ TxA ให้แก้ไข จำนวน = 100 โดยยังไม่คอมมิต เกิดการ รอโปรแกรมใน TxB ไม่สามารถทำงานต่อได้
		สั่งให้ Tx A คอมมิต แต่จะเกิด error can't serilize access ขึ้นที่ Tx B และ TxB จะถูก โรลแบ็ก
รหัส จำนวน 01 100 02 200 03 300 04 100	รหัส จำนวน 01 100 02 200 03 300 04 100	เมื่อทำการ Rescan ทั้ง 2 ทรานแซกชัน จะพบว่า ข้อมูลท่รหัส 04 มีจำนวน = 100 ตามที่ TxA สามารถ updateได้

สำหรับยูสเซอร์ ที่มีสิทธิ์ใช้งานก็จะเห็นหน้าต่างการ กำหนดเส้นทางเดินรถ ว่ารถขบวนไหน วิ่งจากสถานีไหน ไปยังสถานีไหน และเวลาในการเดินทาง



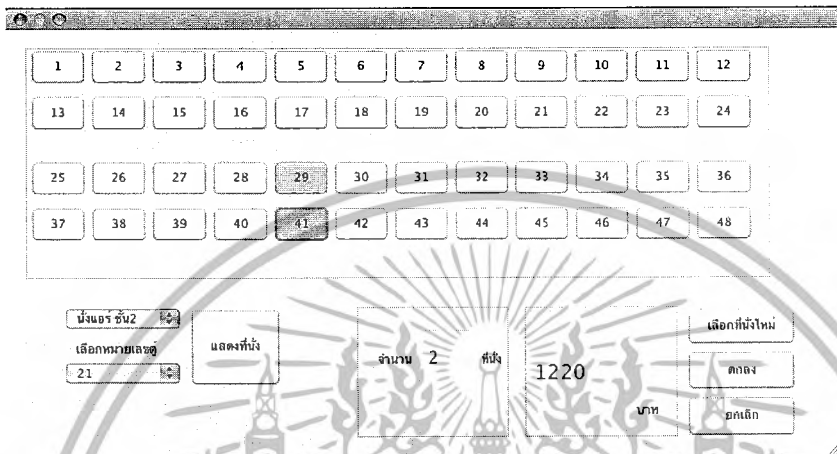
รูปที่ 5.10 หน้าต่างการกำหนดเส้นทางเดินรถ

c) การใช้งานในส่วนของการจองตั๋วรถ โดยทำการเลือกสถานีต้นทาง สถานีปลายทาง และวันที่เดินทางโปรแกรมก็จะแสดงขบวนรถที่ตรงตามต้องการมาให้



รูปที่ 5.11 หน้าต่างแสดงขบวนรถเที่ยวที่ตรงกับความต้องการ

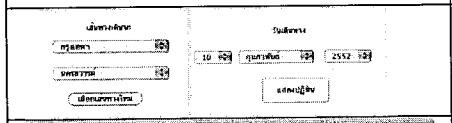
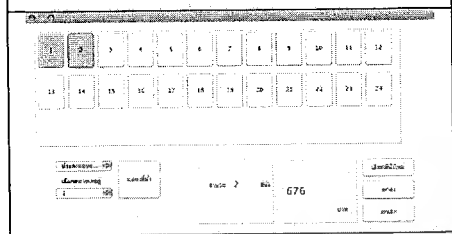
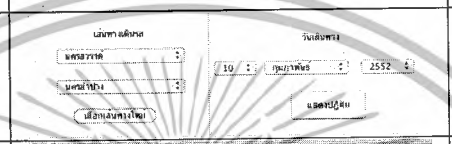
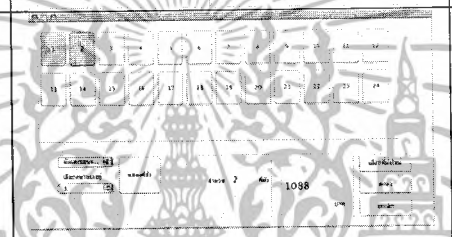
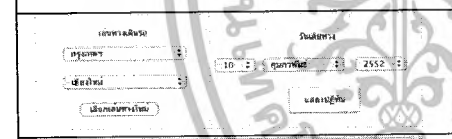
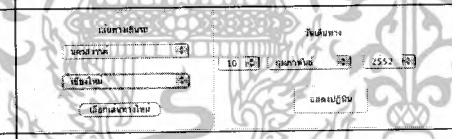
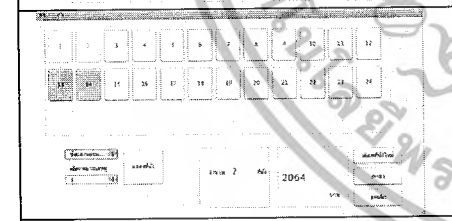
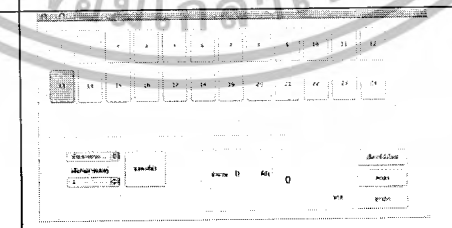
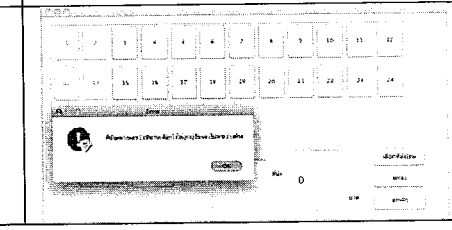
และเมื่อกดคลิก ที่ขบวนรถ ก็จะปรากฏหน้าต่างให้เลือกโบกี้และที่นั่งที่ต้องการจองซึ่งในการเลือกที่นั่งแต่ละครั้งจะทำให้เกิดอิเว็น และเซสชันที่มี ไอโซเลชันเลเวล serializable จะทำการ UPDATE ข้อมูลของที่นั่งนั้นๆ แต่ยังไม่ คอมมิต จนกว่าจะกดตกลง



รูปที่ 5.12 หน้าต่างการเลือกที่นั่ง

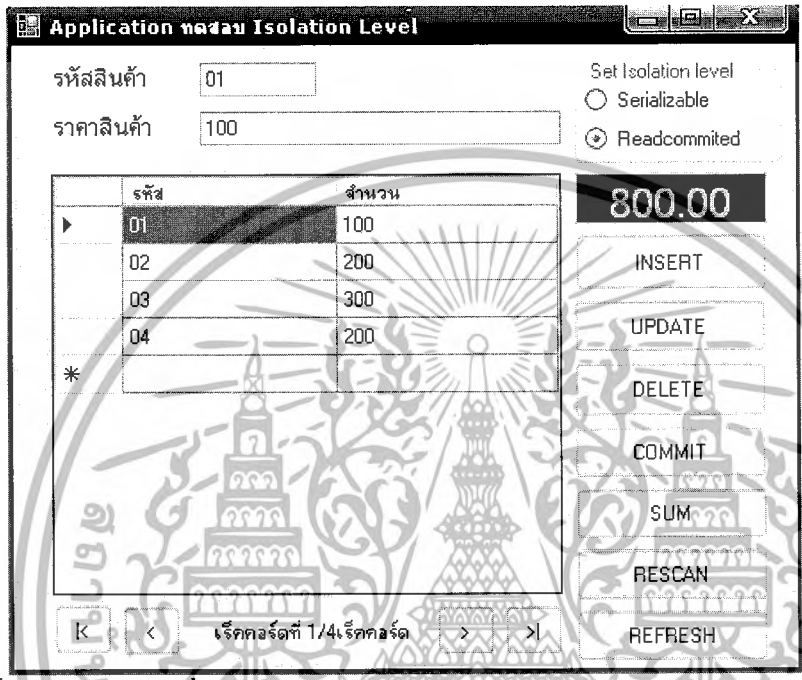
d) การทดสอบการจองที่นั่งเดียวกัน ในเส้นทางเดียวกัน พร้อมกัน ซึ่งเหตุการณ์แบบนี้จะเป็นปัญหา Lost Update ในมุมมองของ Concurrency control

ตารางที่ 5.10 การทดลองทำ ทรานแซกชัน 2 ทรานแซกชัน พร้อมๆ กัน

User A จองที่ กรุงเทพฯ	User B จองที่นครสวรรค์	สถานะ
		
		ยูสเซอร์ A คอมมิต
		
		ยูสเซอร์ B สามารถเลือกที่นั่งเดิมได้เพราะสถานีเดินทางไม่เหลื่อมกับยูสเซอร์ A
		
		
		ยูสเซอร์ B จะรอให้ ยูสเซอร์ A คอมมิต หรือ โรลแบ็ก ก่อน
User A กดตกลง (Commit)		แจ้งให้ ยูสเซอร์ B ทราบว่า มีคนจองก่อนแล้ว

5.3 การทดลองที่ 3 การใช้โปรแกรมทดสอบ ไอโซเลชัน เลเวล กับผลิตภัณฑ์ โพสเกรส เอสคิวแอล

ทำการทดลองโดยเขียนโปรแกรมโดยใช้ภาษา วิชวลเบสิกดอทเน็ต (vb.net) และเรียกใช้เคอร์เซอร์ในการ insert,delete,update ซึ่งมีหน้าตา โปรแกรมดังนี้



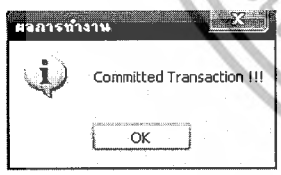
รูปที่ 5.13 โปรแกรมที่ใช้ในการทดสอบ ไอโซเลชัน เลเวล ของ โพสเกรสเอสคิวแอล

Set Isolation level	เป็นการเซ็ตค่า ไอโซเลชัน เลเวล ที่ต้องการในการทำการทดลอง
Insert	ทำการ Insert ข้อเป็น เอสคิวแอล ลง ในฐานข้อมูล
Update	ทำการ Update ข้อเป็นเอสคิวแอลลงในฐานข้อมูลในrow ที่เคอร์เซอร์ชี้อยู่
Delete	ทำการลบข้อมูลแถวที่ เคอร์เซอร์ชี้อยู่
Commit	ทำการยืนยันข้อมูล ที่ต้องการให้ลงฐานข้อมูลทันที
Sum	ทำการหาผลรวมของค่าทั้งหมด โดยการกด 1 ครั้งจะเป็นการ บวกเพิ่มทีละแถว โดยที่จะเป็นตัวบอกว่าขณะนี้ เคอร์เซอร์ชี้อยู่ที่ไหน และ มีการแสดงผลรวมตั้งแต่แถวแรกจนถึงแถวที่ เคอร์เซอร์ ชี้อยู่
Rescan	ทำการโหลดข้อมูลจากฐานข้อมูลขึ้นมาใหม่
Refresh	กลับไปสู่ค่าเริ่มต้นของฐานข้อมูลที่ได้โหลดมา
Cursor	เป็นการเลือก Row ที่ต้องการเพื่อทำการเปลี่ยนแปลงข้อมูล

5.3.1 ทรานแซกชันไอโซเลชันเลเวล Read-Committed

5.3.1.1 Lost update problem

ตารางที่ 5.11 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Lost update problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 400		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
	รหัส จำนวน 01 100 02 200 03 300 04 400	เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 100		แก้ไขข้อมูลแถวที่รหัส04 ของ TxA ให้มี จำนวน = 100 โดยยังไม่ คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 200	แก้ไขข้อมูลที่รหัส 04 ของ TxB ให้จำนวน = 200 โดยยังไม่ คอมมิต เกิดการ รอโปรแกรมใน TxB ไม่สามารถทำงานต่อได้
		สั่งให้ Tx A คอมมิตโปรแกรมใน TxB สามารถทำงานต่อได้
		สั่งให้ Tx B คอมมิต
รหัส จำนวน 01 100 02 200 03 300 04 200	รหัส จำนวน 01 100 02 200 03 300 04 200	ผลลัพธ์ของทั้ง TxA และ TxB หลังจาดกดปุ่ม rescan ปรากฏว่าได้ 200 เท่ากัน

จากการทดลอง นี้จะเห็นได้ว่า การตั้งค่าไอโซเลชัน เลเวล เป็น Read-Committed สามารถแก้ไขปัญหา lost update problem ได้ เนื่องจาก TxA เป็น ทรานแซกชัน ที่มาก่อน จะทำกการ Lock Row 04 ไว้ TxB จะเข้ามาทำการแก้ไขไม่ได้ จนกว่า TxA จะทำงานเสร็จ จึงทำให้ไม่มีการรบกวนการทำงานกัน

5.3.1.2 Uncommit dependency

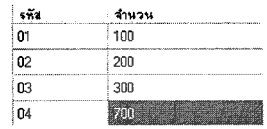
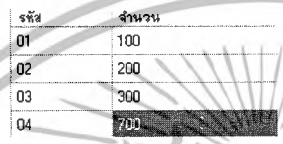
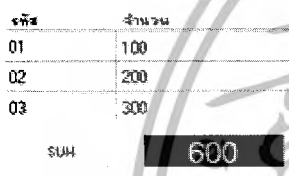
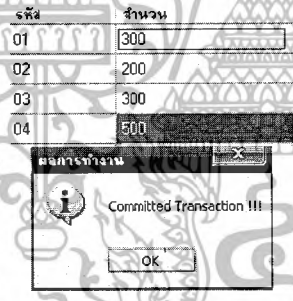
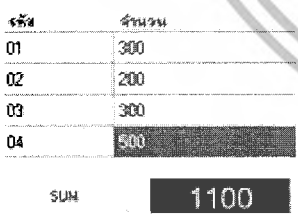
ตารางที่ 5.12 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา uncommit dependency กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 200		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 700		แก้ไขข้อมูลที่ รหัส 04 ของ Tx A ให้ จำนวน = 700
	รหัส จำนวน 01 100 02 200 03 300 04 200	จากนั้นเรียกโปรแกรมทดสอบ ของ TxBขึ้นมาแสดงจะเห็นว่า ที่รหัส 04 ยังคงมี จำนวนเท่ากับ 200 อยู่
ผลการทำงาน Committed Transaction !!! OK		สั่ง Tx A คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 700	กดปุ่ม ReScan เพื่อสแกนดูข้อมูลใน อีกครั้ง จะเห็นว่าข้อมูลใน TxA เปลี่ยนไป

จากการทดลองจะเห็นว่า การตั้งค่าไอโซเลชัน เลเวล เป็น Read-Committed สามาแก้ไขปัญหา uncommit dependency ได้ สังเกตได้จากเมื่อ TxA ทำการ Update แล้วแต่ยังไม่ คอมมิต ที่ TxB ก็จะยังไม่สามารถเห็นการเปลี่ยนแปลงที่ Row นั้น จนกว่า TxA จะทำการยืนยัน คอมมิต ลงฐานข้อมูล TxB ก็จะสามารถมองเห็นการแปลงแปลง

5.3.1.3 Inconsistent analysis problem

ตารางที่ 5.13 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
		กดปุ่ม Sum 2 ครั้ง เพื่อทำการหาผลรวมของ คอลั่ม จำนวน จะได้ (100+200+300) = 600
		ทำการแก้ไขข้อมูลใน Tx B โดยจำลองการ โอนเงินจาก รหัส04 ไป รหัส 01 จำนวน 200 และ คอมมิต
		กดปุ่ม Sum ครั้งที่ 3 ผลลัพธ์ คือ 900+500 = 1100

จากการทดลองนี้ จะเห็นได้ว่า การคำนวนหาผลรวมของจำนวนเงินที่มีในระบบทั้งหมดผิดไป จการทำงานที่ถูกต้องคือ 100+200+300+700 ผลรวมที่ได้ คือ 1,300 แต่ในการทดลองนี้ ได้มีการโอนย้ายจำนวนเงินภายในฐานข้อมูลเดียวกันเองขณะที่มีการหาผลรวม เมื่อมา Sum ครั้งสุดท้ายที่รหัส 04 จึงได้100+200+300+500 = 1100 ดังนั้น ไอโซเลชัน เลเวล Read-Committed ไม่สามารถแก้ไขปัญหาเรื่อง inconsistent analysis problem ได้

5.3.1.4 Phantom Phenomenom

ตารางที่ 5.14 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Phantom Phenomenom กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

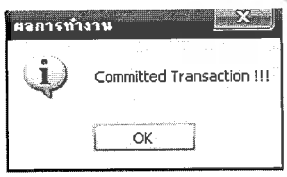
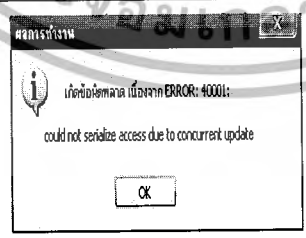
Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 700		เรียกโปรแกรมทดสอบของ TxAขึ้นมาแสดง
	รหัส จำนวน 01 100 02 200 03 300 04 700	เรียกโปรแกรมทดสอบของ TxBขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 SUM 600		กดปุ่ม SumStep เพื่อทำการหาผลรวมของ value ทีละ row
	รหัส จำนวน 01 400 02 200 03 300 04 700 05 300	ทำการเพิ่มข้อมูล ID 4 มี Value = 40 ลงในTxB
รหัส จำนวน 01 400 02 200 03 300 04 700 05 300 SUM 1600		กดปุ่ม SumStep ครั้งที่ 2 ผลลัพธ์ คือ 30(10+20)

จากการทดลองจะเห็นว่า การทดลอง ไม่สามารถแก้ปัญหา phantom phenomenom ได้ เพราะผลลัพธ์ที่ควรจะได้ คือ 100+200+300+700+300 = 1600 ซึ่งไม่ควรเห็นข้อมูลที่เพิ่มเข้ามาหลังจากเริ่ม ทรานแซกชัน แล้ว

5.3.2 ทรานแซกชันไอโซเลชันเลเวล Serializable

5.3.2.1 lost update problem

ตารางที่ 5.15 การทดสอบ ทรานแซกชัน Isolation Level Serializable กับการแก้ไขปัญหา lost update problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 400		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
	รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 400	เรียกโปรแกรมทดสอบ ของ TxB ขึ้นมาแสดง
รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 100		แก้ไขข้อมูลที่รหัส 04 ของ TxA ให้แก้ไข จำนวน = 100 โดยยังไม่ คอมมิต
	รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 200	แก้ไขข้อมูลที่รหัส 04 ของ TxA ให้แก้ไข จำนวน = 100 โดยยังไม่ คอมมิต เกิดการ รอโปรแกรมใน TxB ไม่สามารถทำงานต่อได้
		สั่งให้ Tx A คอมมิต แต่จะเกิด error can't serilize access ขึ้นที่ Tx B และ TxB จะถูก โรลแบ็ก
รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 100	รหัส / จำนวน 01 / 100 02 / 200 03 / 300 04 / 100	เมื่อทำการ Rescan ทั้ง 2 ทรานแซกชัน จะพบว่า ข้อมูลท่รหัส 04 มีจำนวน = 100 ตามที่ TxA สามารถ updateได้

จากการทดลองจะเห็นว่า เมื่อ ตั้งค่าไอโซเลชัน เลเวล เป็น Serailizable แล้ว สามารถแก้ไขปัญหา lost update ได้ แต่จะเป็นการเกิด ความผิดพลาด แทน เพราะลำดับการทำงานของปัญหา lost update นั้นไม่ config serializable schedule

5.3.2.2 Uncommit dependency

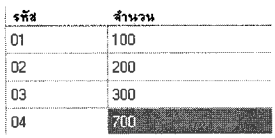
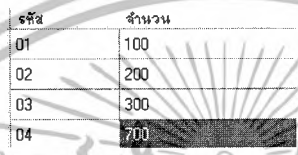
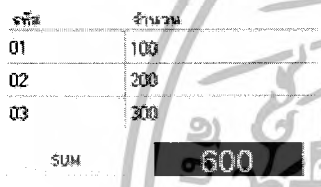
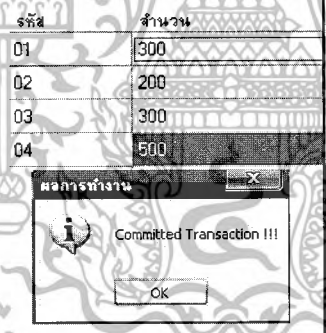
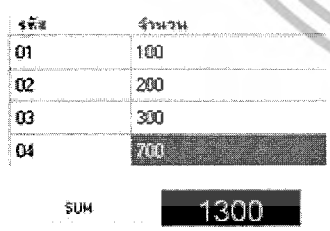
ตารางที่ 5.16 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Uncommit dependency กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
รหัส จำนวน 01 100 02 200 03 300 04 200		เรียกโปรแกรมทดสอบ ของทรานแซกชัน A ขึ้นมาแสดง
รหัส จำนวน 01 100 02 200 03 300 04 700		แก้ไขข้อมูลที่ รหัส 04 ของ Tx A ให้ จำนวน = 700
	รหัส จำนวน 01 100 02 200 03 300 04 200	จากนั้นเรียกโปรแกรมทดสอบของ ทรานแซกชัน B ขึ้นมาแสดงจะเห็นว่า ที่รหัส 04 ยังคงมี จำนวนเท่ากับ 200 อยู่
ผลการทำงาน Committed Transaction !!! OK		สั่ง Tx A คอมมิต
	รหัส จำนวน 01 100 02 200 03 300 04 700	กดปุ่ม ReScan เพื่อ scan ดูข้อมูลในอีกครั้ง จะเห็นว่าข้อมูลใน TxA เปลี่ยนไป

จากการทดลองจะเห็นว่า การ Set ไอโซเลชัน เลเวล เป็น Serailizable สามาแก้ไขปัญหา uncommit dependency ได้เหมือนกัน Read-Committed ทุกประการ

5.3.2.3 Inconsistent analysis problem

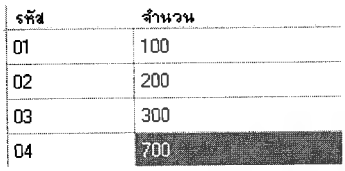
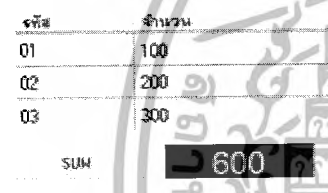
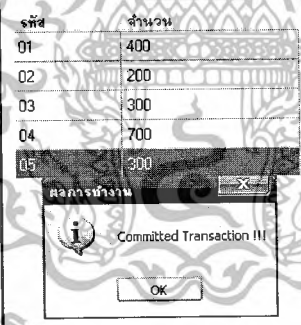
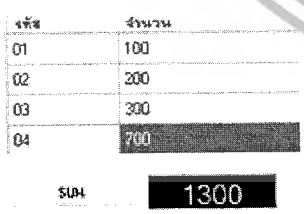
ตารางที่ 5.17 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Inconsistent analysis problem กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
		เรียกโปรแกรมทดสอบ ของ TxA ขึ้นมาแสดง
		เรียก โปรแกรมทดสอบ ของ ทรานแซกชัน B ขึ้นมาแสดง
		กดปุ่ม Sum 2 ครั้ง เพื่อทำการหาผลรวมของ คอลั่ม จำนวน จะได้ (100+200+300) = 600
		ทำการแก้ไขข้อมูลใน Tx B โดยจำลองการ โอนเงินจาก รหัส04 ไป รหัส 01 จำนวน 200 และ คอมมิต
		กดปุ่ม Sum ครั้งที่ 3 ผลลัพธ์ คือ 900+700 = 1300

จากการทดลองจะเห็นได้ว่า เมื่อเรา ตั้งค่าไอโซเลชัน เลเวล เป็น Serializable ที่ TxB ได้มีการ Update เข้ามาแล้ว คอมมิต ก็จะพบว่า ที่ TxA จะยังไม่สามารถมองเห็นการเปลี่ยนแปลงที่เกิดขึ้นในตารางได้ จึงสามารถ แก้ไขปัญหา inconsistent analysis problem ได้

5.3.2.4 Phantom Phenomenom

ตารางที่ 5.18 การทดสอบ ทรานแซกชัน ไอโซเลชัน เลเวล Read-Committed กับการแก้ไขปัญหา Phantom Phenomenom กับผลิตภัณฑ์ โพสเกรสเอสคิวแอล

Tx A	Tx B	Status
		เรียกโปรแกรมทดสอบ ของทรานแซกชัน A ขึ้นมาแสดง
		เรียกโปรแกรมทดสอบ ของทรานแซกชัน B ขึ้นมาแสดง
		กดปุ่ม Sum 2 ครั้ง เพื่อทำการหาผลรวมที่ 100+200+300 = 600
		ทำการเพิ่มข้อมูลลงไป คือ รหัส 05 มีจำนวนเงิน 300 และ คอมมิต เพื่อยืนยันข้อมูล
		กดปุ่ม Sum อีก จะเห็นว่า TxA มองไม่เห็นสิ่งที่ Txb เพิ่มลงมาในตราง ผลลัพธ์ คือ 100+200+300+700 = 1300

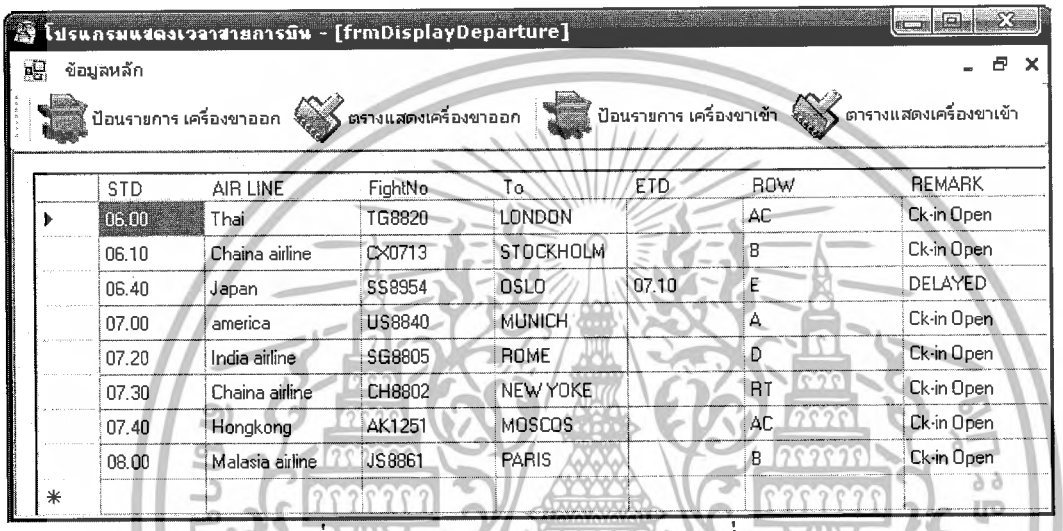
จากการทดลองจะเห็นได้ว่า เมื่อเราตั้งค่าไอโซเลชัน เลเวล เป็น Serializable ที่ TxB ได้มีการ Insert ข้อมูลลงในฐานข้อมูลแล้ว คอมมิต ก็จะพบว่า ที่ TxA จะยังไม่สามารถมองเห็นการเปลี่ยนแปลงที่เกิดขึ้นในตารางได้ จึงสามารถ แก้ไขปัญหา Phantom Phenomenom ได้

5.4 การทดลองที่ 4 ประยุกต์ใช้งานโปรแกรมกับ ไอโซเลชันเลเวล โพสเกรสเอสคิวแอล

ประยุกต์ใช้งานโปรแกรมกับ ไอโซเลชันเลเวล โพสเกรสเอสคิวแอล(Read committed และ Serailizable)

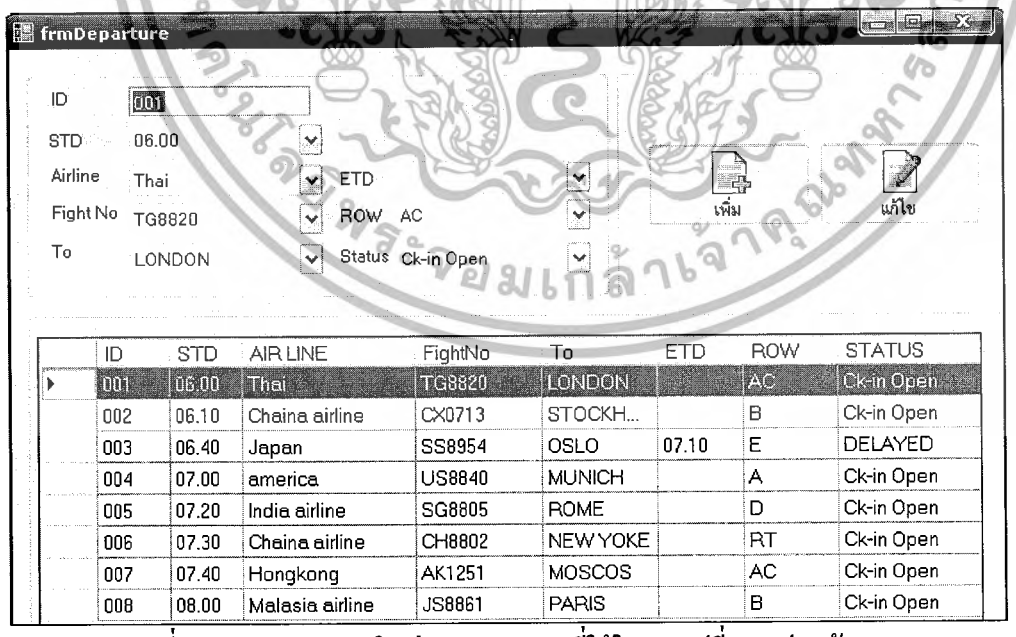
5.4.1 โปรแกรม แสดงสายการบินของสนามบินการ

โปรแกรม แสดงสายการบินของสนามบินการ กับการใช้ตั้งค่า ทรานแซกชันไอโซเลชันเลเวล READ-COMMITTED ของผลิตภัณฑ์ ออราเคิล



	STD	AIR LINE	FightNo	To	ETD	ROW	REMARK
▶	06.00	Thai	TG8820	LONDON		AC	Ck-in Open
	06.10	Chaina airline	CX0713	STOCKHOLM		B	Ck-in Open
	06.40	Japan	SS8954	OSLO	07.10	E	DELAYED
	07.00	america	US8840	MUNICH		A	Ck-in Open
	07.20	India airline	SG8805	ROME		D	Ck-in Open
	07.30	Chaina airline	CH8802	NEW YOKE		RT	Ck-in Open
	07.40	Hongkong	AK1251	MOSCOS		AC	Ck-in Open
	08.00	Malasia airline	JS8861	PARIS		B	Ck-in Open
*							

รูปที่ 5.14 ตารางแสดงสายการบินขาออกที่ออก



	ID	STD	AIR LINE	FightNo	To	ETD	ROW	STATUS
▶	001	06.00	Thai	TG8820	LONDON		AC	Ck-in Open
	002	06.10	Chaina airline	CX0713	STOCKH...		B	Ck-in Open
	003	06.40	Japan	SS8954	OSLO	07.10	E	DELAYED
	004	07.00	america	US8840	MUNICH		A	Ck-in Open
	005	07.20	India airline	SG8805	ROME		D	Ck-in Open
	006	07.30	Chaina airline	CH8802	NEW YOKE		RT	Ck-in Open
	007	07.40	Hongkong	AK1251	MOSCOS		AC	Ck-in Open
	008	08.00	Malasia airline	JS8861	PARIS		B	Ck-in Open

รูปที่5.15 Application ในส่วนของ User ที่ใช้ในการเปลี่ยนแปลงข้อมูล

ด้วยคุณสมบัติ ไอโซเลชัน เลเวล Read Committed ดังที่กล่าวมาข้างต้น เป็น เหมาะกับการใช้งานของโปรแกรมจำลอง ฐานข้อมูลสนามบิน มากที่สุด การทดลองนี้จึงเน้นไปที่ การตั้งค่าไอโซเลชันเลเวล เป็น Read Committed จากความต้องการของโปรแกรมจำลอง ฐานข้อมูลสนามบินนั้นที่สำคัญคือ

- ตารางแสดงสายการบินขาเข้า และขาออก ต้องมีคุณสมบัติเป็น Real-time คือ เมื่อยูสเซอร์ การเพิ่มข้อมูลหรือแก้ไขข้อมูลของสายการบิน ต้องมีการแสดงของมูลนั้นขึ้นที่ โปรแกรมทันที
- การใช้งานต้องเป็นมัลติยูสเซอร์คือ ต้องรองรับการใช้งานของยูสเซอร์ หลายคนในเวลาพร้อมกันและทุกยูสเซอร์ ที่มีการเปลี่ยนแปลงฐานข้อมูล ข้อมูลนั้นต้องลงในฐานข้อมูลจริงๆ และไม่ขัดกับหลักการของฐานข้อมูล

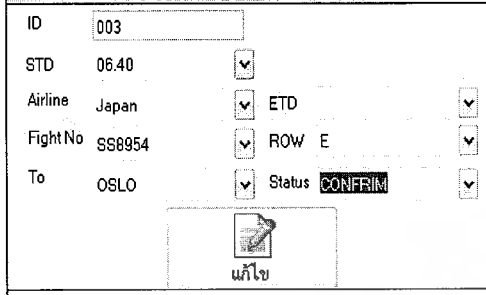
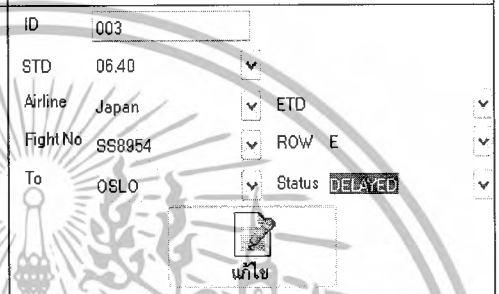
5.4.2 การทดสอบโปรแกรมแสดงสายการบินของสนามบิน

เริ่มต้นการทดลอง สถานะของตารางแสดงรายการสายการบินที่จะลงท่าสนามบินมีดังนี้

STD	AIR LINE	FightNo	To	ETD	ROW	REMARK
06.00	Thai	TG8820	LONDON		AC	Ck-in Open
06.10	Chaina airline	CX0713	STOCKHOLM		B	Ck-in Open
06.40	Japan	SS8954	OSLO		E	Cancelled

รูปที่5.16 แสดงตารางสายการบินขาออก

ตารางที่ 5.19 แสดงช่วงเวลาการทดลอง การใช้การแบบ Multi user

Application ที่1	Time	Application ที่ 2
	T1	
	T2	
	T3	

เริ่มต้นการทดลอง ที่ T1 จำลองว่ามียูสเซอร์คนแรก เปิด Application เพื่อทำการ Update ข้อมูลในของเครื่องบินขาเข้า สายการบิน Japan ที่เวลา 06.40 เปลี่ยนแปลง คอลั่ม REMARK เดิมเป็น Cancelled ให้เป็น CONFRIM จากนั้นกดปุ่มแก้ไข และยื่นยันการแก้ไขข้อมูลโปรแกรมจะทำการแก้ไขข้อมูล แต่จะยังไม่ คอมมิต จากนั้น

T2 มียูสเซอร์อีก1 คนต้องการกแก้ไขข้อมูลที่ Row เดียวกันคือ สายการบิน Japan ที่เวลา 06.40 เปลี่ยนแปลง คอลั่ม REMARK เดิมเป็น Cancelled ให้เป็น DELAYED กดปุ่มแก้ไขข้อมูล และยื่นยันการแก้ไขข้อมูล เมื่อเสร็จขั้นตอนนี้แล้วโปรแกรม ที่ 2 นี้ จะค้างชั่วขณะ เพราะว่าโปรแกรมที่ 1 ได้ทำการแก้ไขข้อมูลที่ Row นี้อยู่ก่อนและยังไม่ได้ คอมมิต จึงล็อก Row นี้ค้างไว้ โปรแกรมที่ 2 จึงต้องรอจนกว่า โปรแกรมที่ 1 จะ คอมมิต หรือ โรลแบ็ก จึงจะสามารถทำงานต่อไปได้

T3 Application ที่ 1 คอมมิต

STD	AIR LINE	FightNo	To	ETD	ROW	REMARK
06.00	Thai	TG8820	LONDON		AC	Ck-in Open
06.10	Chaina airline	CX0713	STOCKHOLM		B	Ck-in Open
06.40	Japan	SS8954	OSLO		E	CONFRIM

รูปที่ 5.17 แสดงการเปลี่ยนแปลงตารางสายการบินเมื่อมีโปรแกรมที่ 1 มีการ Update และคอมมิต

ตารางที่ 5.19 แสดงช่วงเวลาการทดลอง การใช้การแบบ Multi user **(ต่อ)**

Application ที่1	Time	Application ที่ 2
	T4	

T4 หลังจากที่ โปรแกรมที่ 1 ได้ คอมมิต เรียบร้อยไปแล้วโปรแกรม ที่ 2 จะสามารถทำงานต่อไปได้ เนื่องจาก Row ที่ถูกล็อก โดยโปรแกรมที่ 1 ได้ ปลดล็อก แล้ว ก็จะทำการแก้ไขข้อมูลในคอลั่ม REMARK ให้เป็น DELAYED แล้วทำการ คอมมิต ข้อมูลนี้ก็จะถูกเก็บลงฐานข้อมูลทันที

STD	AIR LINE	FightNo	To	ETD	ROW	REMARK
06.00	Thai	TG8820	LONDON		AC	Ck-in Open
06.10	Chaina airline	CX0713	STOCKHOLM		B	Ck-in Open
06.40	Japan	SS8954	OSLO		E	DELAYED

รูปที่5.18 แสดงการเปลี่ยนแปลงตารางสายการบินเมื่อมีโปรแกรมที่2 มีการ Update และคอมมิต

ที่โปรแกรมแสดงตารางสายการบินขาเข้า จะเห็นได้ว่ามีการเปลี่ยนแปลข้อมูลที่ สายการบิน Japan ที่เวลา 06.40 ในคอลั่ม REMARK เป็น CONFRIM จริง

5.4 .3 โปรแกรมจองตั๋วโรงภาพยนตร์

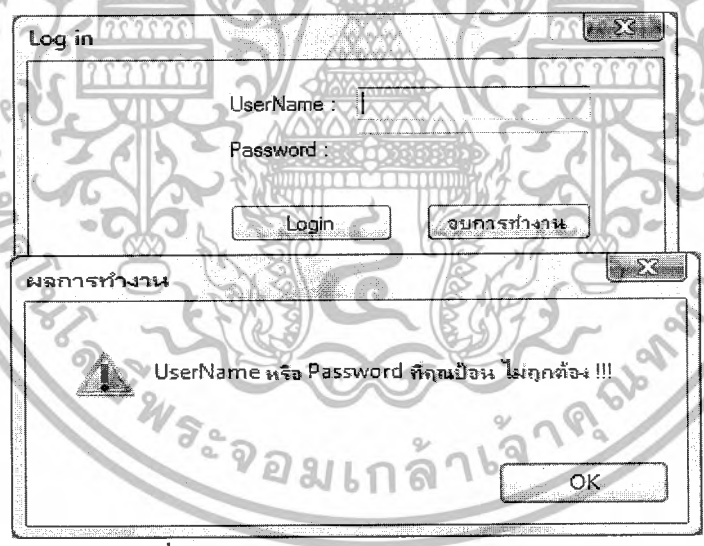
เป็นการทดลองการทำงานของ ทรานแซกชัน ไอโซเลชัน เลเวล Serializable ของผลิตภัณฑ์ โพสเกรสเอสคิวแอลซึ่ง โปรแกรมที่ใช้ทดสอบคือ ระบบจองตั๋วภาพยนตร์ ซึ่งระบบจองตั๋วภาพยนตร์จะสามารถรองรับการทำงานแบบผู้ ใช้หลายคนและมีระบบรักษาความปลอดภัยโดยการกำหนดสิทธิ์ต่าง ๆ ในการเข้าใช้งานโปรแกรมแต่ ละคนทำให้สามารถควบคุมความถูกต้องของข้อมูลไว้ได้

5.4.2.1 ความสามารถของโปรแกรม

- โปรแกรมสามารถตรวจสอบการเข้าใช้งานของ User ได้โดยถ้าไม่สามารถเข้าใช้งานโปรแกรมได้ก็จะมีการแจ้งเตือนให้ทราบ
- ถ้ามีการจองที่นั่งเดียวกันพร้อมกันโปรแกรมจะแจ้งเตือนให้ทราบ
- สามารถเพิ่มรายชื่อภาพยนตร์ผ่านทางหน้าโปรแกรมได้

5.4.3 การทดสอบโปรแกรมจองตั๋วโรงภาพยนตร์

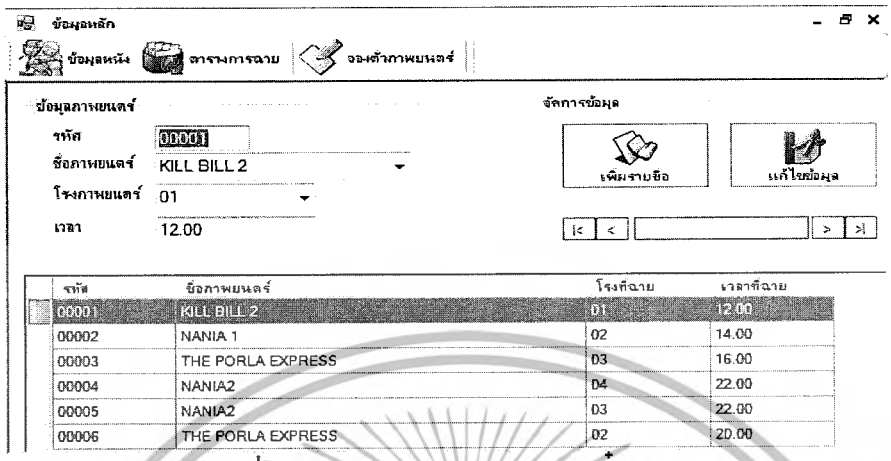
5.4.3.1 เมื่อใช้งานโปแกรมครั้งแรกโปรแกรมจะให้ใส่ ยูสเซอร์และรหัสผ่าน



รูปที่5.19 แสดงการยืนยันตัวก่อนเข้าใช้งาน

เมื่อมีการเข้าใช้งานโปรแกรมโดยที่ใส่ยูสเซอร์และรหัสผ่านผิดโปรแกรมจะแจ้งเตือนให้ทราบ

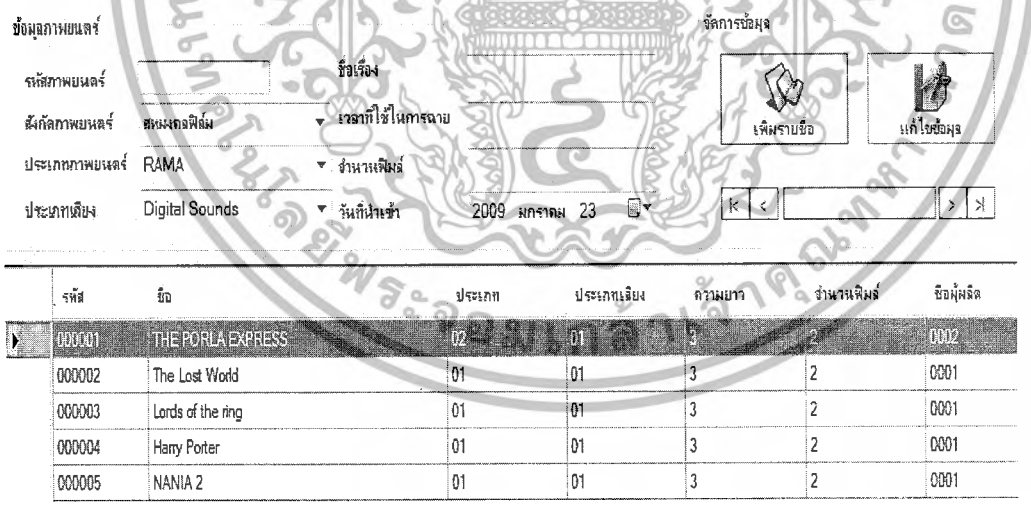
5.4.3.2 เมื่อมีการเข้าใช้งานได้แล้วโปรแกรมจะเข้าสู่หน้าจอดังนี้



รูปที่5.20 แสดงหน้าหน้าจอการจองตั๋วหนัง

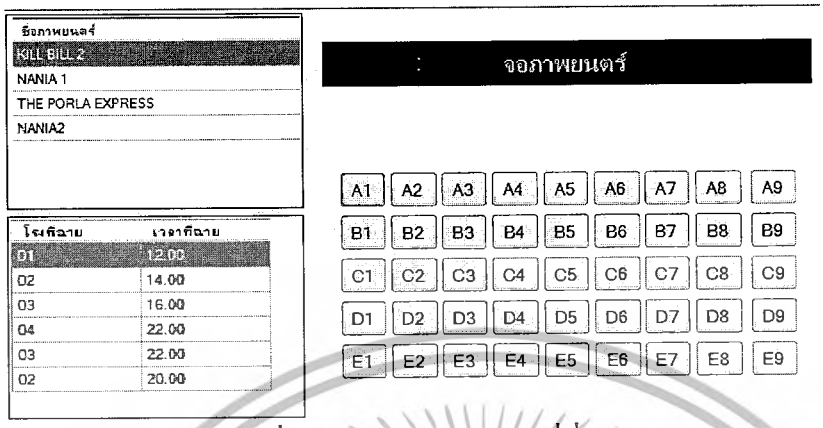
จากรูปจะเป็นหน้าจอหลักที่ทำหน้าที่ในการเพิ่มข้อมูลการฉายหนังและถ้าต้องการจองตั๋วภาพยนตร์ก็คลิกที่ปุ่มจองตั๋วภาพยนตร์โปรแกรมก็จะเปิดหน้าจอมาดังรูปต่อไปนี้

5.4.3.3 แสดงหน้าจอการทำงานของโปแกรมจองตั๋วภาพยนตร์



รูปที่ 5.21 หน้าจอแสดงการเพิ่มรายชื่อภาพยนตร์

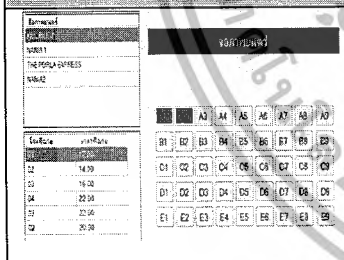
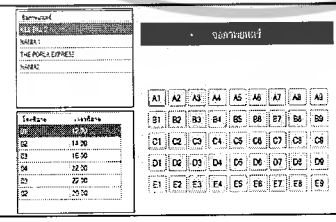
5.4.3.4 เมื่อมีการคลิกที่จองตั๋วภาพยนตร์แล้วโปรแกรมก็จะเปิดหน้าการจองขึ้นมา



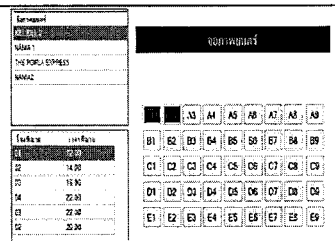
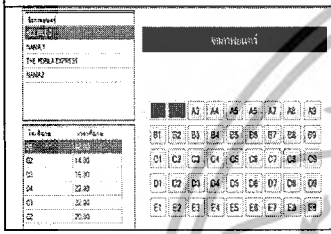
รูปที่ 5.22 แสดงหน้าจอเลือกที่นั่ง

จากรูปเมื่อทำการเลือกที่ชื่อภาพยนตร์โปรแกรมก็จะแสดงเวลาพร้อมกับโรงภาพยนตร์ที่ฉายหลังจากนั้นก็ทำการจองที่นั่งตามหมายเลขที่นั่งซึ่งถ้ามีการจองที่นั่งพร้อมกันโปรแกรมจะแจ้งให้ทราบว่าที่นั่งมีการจองแล้วดังตารางต่อไปนี้

ตารางที่ 5.20 การทดสอบการใช้งานแบบ มัลติยูสเซอร์ ที่ใช้งานในเวลาเดียวกัน

User A	User B	สถานะ
		ยูสเซอร์ A เปิดโปรแกรมจองตั๋วหนังขึ้นมาและทำการเลือกที่นั่งแต่ยังไม่มีการ คอมมิต
		ยูสเซอร์ B เปิดโปรแกรมขึ้นมาโดยที่ยังไม่ได้เลือกที่นั่ง

ตารางที่ 5.20 การทดสอบการใช้งานแบบ Multi user ที่ใช้งานในเวลาเดียวกัน(ต่อ)

		ยูสเซอร์ B ทำการเลือกที่นั่งเดียวกันที่ ยูสเซอร์ A ได้เลือกไว้
		หลังจากที่ User B ทำการเลือกที่นั่งเดียวกันโปรแกรมจะแจ้งให้ทราบ
		ยูสเซอร์ A ทำการ คอมมิต และจะทำให้ยูสเซอร์ B ยกเลิกไป

บทที่ 6
บทวิจารณ์และสรุป

6.1 บทวิจารณ์และสรุปผล

จากการศึกษาได้พบว่าระบบจัดการฐานข้อมูล(ดีบีเอ็มเอส) แบบมัลติเวอร์ชันเอ็นจิ้น นั้น มีการรองรับคุณสมบัติของทรานแซกชัน ทุกประการ ไม่ว่าจะเป็นการทำทรานแซกชันบนเฐานข้อมูลเดี่ยว หรือฐานข้อมูลแบบกระจาย และจุดเด่นของ ระบบจัดการฐานข้อมูลแบบมัลติเวอร์ชันนั้น คือ การอ่านข้อมูล (SELECT) สามารถอ่านตลอดเวลาโดยที่ไม่ต้องรอ และโปรแกรมที่พัฒนาขึ้นนั้น สามารถนำเอาความสามารถของฐานข้อมูลและคุณสมบัติของทรานแซกชัน ไปใช้ช่วยลดภาระด้านการเขียน Code ได้ในระดับหนึ่ง

6.1.1 ความสามารถของ ออราเคิล 11จี

ออราเคิล 11จี นั้นมีเครื่องความสามารถมากมายรวมถึงมีเครื่องมือที่ช่วยใน การรักษาความปลอดภัยของฐานข้อมูลที่ดีเพราะมี ซิสเท็มพริวิเลจมากกว่า 100 พริวิเลจ ซึ่งทำให้สามารถจัดการระดับการใช้งานได้อย่างละเอียดพอสมควร และยังสามารองรับการทำ ฐานข้อมูลแบบขนาน (Parallel Database) ออราเคิล 11จี รองรับการทำงานแบบแชร์แคช(Shared cache) เท่านั้น และในแต่ละอินเสตน ต้องอ่านดิกส์จากที่เดียวกัน ซึ่งข้อนข้างเป็นข้อจำกัด แต่ก็สามารถรองรับการขยายตัวของ เซิร์ฟเวอร์ได้ดี เพราะสามารถเพิ่มและลบ โหนดในคลัสเตอร์ ได้ และ ออราเคิล 11จี ยังมีระบบจัดการกับการเชื่อมต่อผู้ใช้ได้ดี เพราะรองรับทั้ง ดีดิเคทเซิร์ฟเวอร์,แชร์เซิร์ฟเวอร์,พลูลิ่ง

6.1.2 ความสามารถของโพสเกรส

โพสเกรสเอสคิวแอล 8.3 เป็นฐานข้อมูลที่สามารถหามาใช้ได้ฟรี ไม่เสียค่าสิขสิทธิ ที่มีความสามารถที่มีความสามารถไม่ว่าจะเป็น รองรับมาตรฐาน เอสคิวแอล 99 การทำ sub select Performance สามารถทำงานโดยเขียนคำสั่งฝั่งใน ฐานข้อมูล โดยที่เป็นลักษณะของ ทริกเกอร์ หรือ พีแอล/เอสคิวแอล , ควบคุมการทำงานในลักษณะที่มีผู้ใช้งานหลายคนด้วย Concurrency Control ,มีความสามารถรองรับการทำฐานข้อมูลแบบขนาน (Parallel Database) ด้วยกริดเอสคิวแอล(Gridsql) อีกทั้งยังมีลักษณะการดูแลความปลอดภัยในฐานข้อมูลด้วยการกำหนดสิทธิการเข้าถึงฐานข้อมูล

6.2 ปัญหาอุปสรรคและแนวทางการแก้ไข

6.2.1 ในการทำฐานข้อมูลแบบกระจายนั้นไม่สามารถควบคุมความถูกต้องของข้อมูลในระดับโกลบอลได้ซึ่งทำได้เพียงในระดับโลคอล แนวทางการแก้ไขคือ การใช้เครื่องมือต่างๆของ ออราเคิล 11จี มาช่วยในการจัดการ เช่น Trigger,view,Synonym แต่ก็สามารถทำได้ยากในระบบฐานข้อมูลที่ซับซ้อน

6.2.2 ในการทดลองทำฐานข้อมูลแบบขนานนั้น ออราเคิล11จี มีข้อจำกันคือ เครื่องเซิร์ฟเวอร์ ที่ทำงานร่วมกันต้องเป็นแพลตฟอร์มเดียวกัน และรองรับแบบแชร์แคชเท่านั้น และยังต้องการอุปกรณ์ Network Attached Storage (NAS) เพื่อให้แต่ละโหนดในระบบอ่านข้อมูลจากดิสเดียวกัน จึงทำให้ไม่มีทรัพยากรเพียงพอในการทดลองจึงต้องใช้ซอฟแวร์ VMWare มาช่วยในการทดลอง โดยการใช้ Network File System(NSF) แทน NAS แต่ก็ไม่สามารถทำให้เห็นผลลัพธ์ ด้าน Performance, Availability เพราะ ออราเคิล 11จี นั้นต้องการทรัพยากรเยอะ แล้วทั้งหมดล้วนจำลองอยู่ในเครื่อง PC ซึ่งมีประสิทธิภาพไม่เพียงพอ และการทำ NFS นั้น เมื่อเครื่องที่เป็น NFS Server ดับไปเครื่องอื่นๆที่เป็น NFS Client ก็ไม่สามารถทำงานต่อได้

6.2.3 ในการพัฒนาโปรแกรมพบข้อผิดพลาดของ ออราเคิล11จี เมื่อใช้ ไอโซเลชัน เลเวล Serializable ที่ไม่ตรงตามทฤษฎีคือ เมื่อมีการ Update ข้อมูลคนละ Row ซึ่งคอสั่ง Update นั้นใช้การ คิวรี่ และ join เงื่อไขในการ Update มาจาก 2 ตาราง ทำให้เกิด Error Can'Serializable Access ขึ้น

6.2.4 ออราเคิล 11จี เป็น Softwareขนาดใหญ่มีเครื่องมือและความสามารถอยู่มาก จึงทำให้ไม่สามารถศึกษาและทดสอบได้ทั้งหมดภายในระยะเวลาที่จำกัดจึงทำการเลือกจุดทดสอบที่น่าสนใจและสามารถพัฒนาความสามารถได้ภายในเวลาที่เหมาะสม

6.2.5 ในการเขียนโปรแกรมติดต่อฐานข้อมูลโดยใช้ โพสเกรสเอสคิวแอล นั้นจะมีปัญหาในเรื่องของ Driver ในการติดต่อฐานข้อมูลทำให้ยากในการพัฒนาอีกทั้งยังโพสเกรสเอสคิวแอล เป็นลักษณะ Open Source ดังนั้นระบบปฏิบัติการที่รองรับจะเป็น Open Source เช่นเดียวกันทำให้การพัฒนาแอพพลิเคชันบนวินโดว์ทำได้ยากเพราะว่าอุปกรณ์หรือ Driver ที่ใช้ในการติดต่อกับฐานข้อมูล โพสเกรสเอสคิวแอล ค่อนข้างน้อยแต่อย่างไรก็ตามก็ยังสามารถพัฒนาแอพพลิเคชันบนวินโดว์ได้ โดยมี Driver ที่รองรับการทำงานของ PostgrSQL อยู่ด้วยกันสองประเภทคือ ODBC และ Npgsql ทำให้สามารถพัฒนาต่อไปได้

6.2.6 ด้วยความที่ โพสเกรสเอสคิวแอล เป็น Open Source ดังนั้นความนิยมในการใช้งานมีไม่มาก เอกสารและคู่มือการใช้งานจึงหาได้ยากมาก ทำให้การทดลองความสามารถของ PosrgreSQL ดำเนินการได้ไม่รวดเร็วเท่าที่ควร

6.3 แนวทางการพัฒนาต่อ

6.3.1 ศึกษาการปรับแต่งประสิทธิภาพของฐานข้อมูล(Performance Tunning)

6.3.2 ศึกษาการติดตั้งฐานข้อมูลแบบขนาน (Parallel Database) หรือ (ออราเคิล Real Application) เชื่อมต่อกับ Disk Arrary หรือ Disk แบบต่างๆ

6.3.3 ศึกษาการออกแบบฐานข้อมูลในระดับฟิสิคอล (Physical Data Design) เพื่อเพิ่มประสิทธิ์ภาพในการทำงานสูงขึ้น และ ศึกษาการสำรองข้อมูล (Backup)

6.3.4 พัฒนาโปรแกรมที่มีขนาดใหญ่ ที่ใช้คุณสมบัติของทรานแซกชัน และ นำเครื่องมือหรือความสามารถที่มีในระบบจัดการฐานข้อมูลไปใช้ในการัฒนาโปรแกรมเหมาะสมกับโปรแกรม หรือ ระบบนั้นๆ

บรรณานุกรม

ญาณี กาชัย. 2546. **จัดการระบบฐานข้อมูลอย่างมืออาชีพ Oracle DBA.** นนทบุรี : อินโฟเพรส.

สัจจะ จรัสรุ่งรวีวร และมณีโชติ, (ผู้รวบรวม) 2564. **จัดการระบบฐานข้อมูลอย่างมืออาชีพ Oracle DBA.** นนทบุรี : อินโฟเพรส

Silberschatz, A. Henry.F. Korth. and Sudarshan, S. 2006. **DATABASE SYSTEM CONCEPTS.** Singapore : McGRAW-HILL.S

Immanuel, 2008. Oracle Database 11g Release 1 (11.1) Documentation. [Online]. Available : http://www.oracle.com/technology/documentation/database11gr1.html

Available : http://www.oracleskill.com

วิสุทธิ์ แซ่ตั้ง , (2546) , Open Source DBMS PostgreSQL,กรุงเทพฯ : สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น)

Matthew, N., Stones, R.(2005). Beginning Databases with PostgreSQL: From Novice to Professional, Second Edition. New York: Apress 1-59059-478-9

Blum, R.,(2005) PostgreSQL 8 for Windows. New York : McGraw-Hill 0-07-150949-6

PostgreSQL Global Development Group, (1996-2008), PostgreSQL 8.3.1 Documentation ,University of California

ภาคผนวก ก

การติดตั้งออราเคิล Real Application Clusters

1. ติดตั้ง OS ออราเคิล Enterprise 5 ทั้ง 2 เครื่อง
2. ตั้งค่า network ทั้ง 2 เครื่อง ดังนี้
   a. ORAC1

      Hostname : orac1.localdomain

      IP eth0 : 192.168.x.x

      Gateway eth0 : 192.168.x.x

      IP eth1 : 10.0.0.x

   b. ORAC2

      Hostname : orac1.localdomain

      IP eth0 : 192.168.x.x

      Gateway eth0 : 192.168.x.x

      IP eth1 : 10.0.0.x

3. ทำการติดตั้ง package เพิ่มเติมทั้ง 2 เครื่องจากแผ่น CD

```
# From Enterprise Linux 5 Disk 1
cd /media/cdrom/Server
rpm -Uvh binutils-2.*
rpm -Uvh elfutils-libelf-0.*
rpm -Uvh glibc-2.*
rpm -Uvh glibc-common-2.*
rpm -Uvh libaio-0.*
rpm -Uvh libgcc-4.*
rpm -Uvh libstdc++-4.*
rpm -Uvh make-3.*
cd /
```

```
eject
# From Enterprise Linux 5 Disk 2
cd /media/cdrom/Server
rpm -Uvh compat-libstdc++-33*
rpm -Uvh elfutils-libelf-devel-*
rpm -Uvh glibc-headers*
rpm -Uvh glibc-devel-2.*
rpm -Uvh libgomp*
rpm -Uvh gcc-4.*
rpm -Uvh gcc-c++-4.*
rpm -Uvh libaio-devel-0.*
rpm -Uvh libstdc++-devel-4.*
rpm -Uvh unixODBC-2.*
rpm -Uvh unixODBC-devel-2.*
cd /
eject
# From Enterprise Linux 5 Disk 3
cd /media/cdrom/Server
rpm -Uvh sysstat-7.*
cd /
eject
```

4. Login root user ที่เครื่อง ORAC1 แล้วแก้ไข File /etc/hosts ดังนี้

```
127.0.0.1       localhost.localdomain   localhost
# Public
192.168.2.101   orac1.localdomain       orac1
192.168.2.102   orac2.localdomain       orac2
#Private
192.168.0.101   orac1-pv.localdomain    orac1-pv
```

192.168.0.102 orac2-pv.localdomain orac2-pv

#Virtual

192.168.2.111 orac1-vip.localdomain orac1-vip

192.168.2.112 orac2-vip.localdomain orac2-vip

#NAS

192.168.2.101 nas1.localdomain nas1

5. เพิ่ม ข้อความต่อไปนี้ลงใน /etc/sysctl.conf

kernel.shmmni = 4096

semaphores: semmsl, semmns, semopm, semmni

kernel.sem = 250 32000 100 128

net.ipv4.ip_local_port_range = 1024 65000

net.core.rmem_default=4194304

net.core.rmem_max=4194304

net.core.wmem_default=262144

net.core.wmem_max=262144

Additional and amended parameters suggested by Kevin Closson

#net.core.rmem_default = 524288

#net.core.wmem_default = 524288

#net.core.rmem_max = 16777216

#net.core.wmem_max = 16777216

net.ipv4.ipfrag_high_thresh=524288

net.ipv4.ipfrag_low_thresh=393216

net.ipv4.tcp_rmem=4096 524288 16777216

net.ipv4.tcp_wmem=4096 524288 16777216

net.ipv4.tcp_timestamps=0

net.ipv4.tcp_sack=0

net.ipv4.tcp_window_scaling=1

net.core.optmem_max=524287

net.core.netdev_max_backlog=2500

sunrpc.tcp_slot_table_entries=128

sunrpc.udp_slot_table_entries=128

net.ipv4.tcp_mem=16384 16384 16384

และ Run คำสั่ง /sbin/sysctl –p เพื่อเปลี่ยนแปลง kernel parameter

6. เพิ่ม คำสั่งต่อไปนี้ลงใน /etc/security/limits.conf

oracle soft nproc 2047

oracle hard nproc 16384

oracle soft nofile 1024

oracle hard nofile 65536

7. เพิ่มคำสั่งต่อไปนี้ลงใน /etc/pam.d/login (ถ้ายังไม่มี)

session required /lib/security/pam_limits.so

session required pam_limits.so

8. ทำการสร้าง Group และ User โดยใช้คำสั่งต่อไปนี้

groupadd oinstall

groupadd dba

groupadd oper

groupadd asmadmin

useradd -u 500 -g oinstall -G dba,oper,asmadmin oracle

passwd oracle

9. ทำการสร้าง SSH ในแต่ละ node (Login “oracle” user) ทำการสร้าง Public key และ Private Key

su - oracle

mkdir ~/.ssh

chmod 700 ~/.ssh

/usr/bin/ssh-keygen -t rsa # Accept the default settings.

10. Login “oracle” User ที่เครื่อง ORAC1 ทำการ copy “Authorized_keys” จาก ORAC1 ไปยัง ORAC2

```
su - oracle
cd ~/.ssh
cat id_rsa.pub >> authorized_keys
scp authorized_keys orac2:/home/oracle/.ssh/
```

Login “oracle” User ที่เครื่อง ORAC2 แล้ว Run คำสั่งดังนี้

```
su – oracle
cd ~/.ssh
cat id_rsa.pub >> authorized_keys
scp authorized_keys orac1:/home/oracle/.ssh/
```

11. เพื่อให้ SSH ทำงานบนเครื่องที่เป็นสมาชิก ออราเคิล RAC ทำงานได้ ให้ใช้คำสั่งดังนี้ ในทุกๆโหนด

```
ssh orac1 date
ssh orac2 date
ssh orac1.localdomain date
ssh roac2.localdomain date
exec /usr/bin/ssh-agent $SHELL
/usr/bin/ssh-add
```

12. Login “oracle” User แล้วทำการเพิ่มข้อความต่อไปนี้ ลงใน ~/.bash_profile ในทุกๆโหนดโดยเปลี่ยน ORACLE_SID และ ORACLE_HOSTNAME ตามโหนดนั้นๆ

```
# Oracle Settings
TMP=/tmp; export TMP
TMPDIR=$TMP; export TMPDIR

ORACLE_HOSTNAME=orac1.localdomain; export
ORACLE_HOSTNAME
```

```
ORACLE_BASE=/u01/app/oracle; export ORACLE_BASE
ORACLE_HOME=$ORACLE_BASE/product/11.1.0/db_1; export
ORACLE_HOME
ORACLE_SID=ORAC1; export ORACLE_SID
ORACLE_TERM=xterm; export ORACLE_TERM
PATH=/usr/sbin:$PATH; export PATH
PATH=$ORACLE_HOME/bin:$PATH; export PATH

LD_LIBRARY_PATH=$ORACLE_HOME/lib:/lib:/usr/lib; export
LD_LIBRARY_PATH
CLASSPATH=$ORACLE_HOME/JRE:$ORACLE_HOME/jlib:$OR
ACLE_HOME/rdbms/jlib; export CLASSPATH

if [ $USER = "oracle" ]; then
 if [ $SHELL = "/bin/ksh" ]; then
  ulimit -p 16384
  ulimit -n 65536
 else
  ulimit -u 16384 -n 65536
 fi
fi
```

13. สร้าง Shared Disk โดยใช้ Network File System(NFS) โดยในตัวอย่างนี้จะให้เครื่อง ORAC1 เป็น Server และเครื่อง ORAC2 เป็น Client โดยใช้คำสั่งต่อไปนี้

บนเครื่อง ORAC1

```
mkdir /shared_config
mkdir /shared_crs
mkdir /shared_home
mkdir /shared_dat
```

เพิ่มข้อความต่อไปนี้ใน /etc/exports

/shared_config *(rw,sync,no_wdelay,insecure_locks,no_root_squash)

/shared_crs

*(rw,sync,no_wdelay,insecure_locks,no_root_squash)

/shared_home

*(rw,sync,no_wdelay,insecure_locks,no_root_squash)

/shared_data

*(rw,sync,no_wdelay,insecure_locks,no_root_squash)

Run คำสั่งต่อไปนี้เพื่อ export NFS shares

chkconfig nfs on service nfs restart

14. สร้าง Directory สำหรับ Oracle Software ทั้ง 2 node (ORAC1,ORAC2)

mkdir -p /u01/app/crs/product/11.1.0/crs

mkdir -p /u01/app/oracle/product/11.1.0/db_1

mkdir -p /u01/oradata

mkdir -p /u01/shared_config

chown -R oracle:oinstall /u01/app /u01/app/oracle /u01/oradata

/u01/shared_config

chmod -R 775 /u01/app /u01/app/oracle /u01/oradata

/u01/shared_config

15. เพิ่มข้อความต่อ ไปนี้ ใน /etc/fstab เพื่อ auto mount NFS

nas1:/shared_config /u01/shared_config nfs

rw,bg,hard,nointr,tcp,vers=3,timeo=300,rsize=32768,wsize=32768,acti

meo=0 0 0

nas1:/shared_crs /u01/app/crs/product/11.1.0/crs nfs

rw,bg,hard,nointr,tcp,vers=3,timeo=300,rsize=32768,wsize=32768,acti

meo=0 0 0

nas1:/shared_home /u01/app/oracle/product/11.1.0/db_1 nfs

rw,bg,hard,nointr,tcp,vers=3,timeo=300,rsize=32768,wsize=32768,acti

meo=0 0 0

nas1:/shared_data /u01/oradata nfs

rw,bg,hard,nointr,tcp,vers=3,timeo=300,rsize=32768,wsize=32768,acti

meo=0 0 0

Mount NFS shares ทั้ง 2 เครื่อง โดยใช้คำสั่ง งนี้

mount /u01/shared_config

mount /u01/app/crs/product/11.1.0/c

mount /u01/app/oracle/product/11.1.0/db_1

mount /u01/oradata

16. ทำการสร้าง shared OCR และ Voting Disk file โดยใช้คำสั่งต่อไปนี้

touch /u01/shared_config/ocr_configuration

touch /u01/shared_config/voting_disk

17. ทำการเปลี่ยน permission ใน shared directory

chown -R oracle:oinstall /u01/shared_config

chown -R oracle:oinstall /u01/app/crs/product/11.1.0/crs

chown -R oracle:oinstall /u01/app/oracle/product/11.1.0/db_1

chown -R oracle:oinstall /u01/oradata

18. Run Program "runcluvfy.sh" เพื่อตรวจสอบการ configuration ของแต่ละ node ว่าสามารถติดตั้ง clusterware ได้มั้ย ด้วยคำสั่งต่อไปนี้

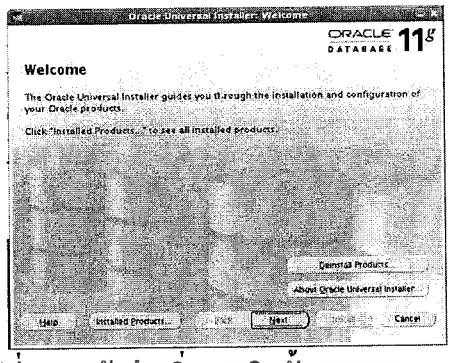
/clusterware/runcluvfy.sh stage -pre crsinst -n orac1,orac2 –verbose

19. ทำการ install clusterware บนเครื่อง ORAC1

Run คำสั่ง

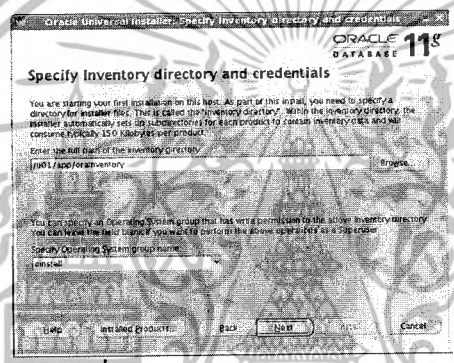
cd clusterware

./runInstaller



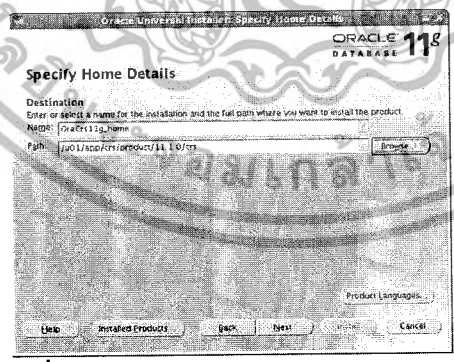
รูปที่ ก.1 หน้าต่างเริ่มการติดตั้ง Clusterware

20. เลือก Default inventory directory



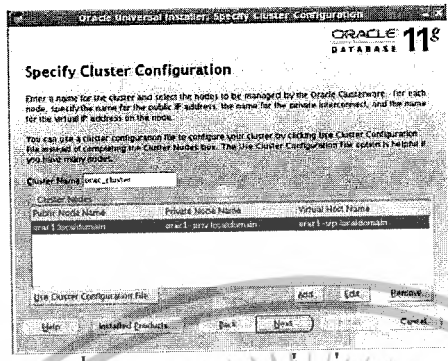
รูปที่ ก.2 เลือก default inventory

21. เลือก path ในการ install cluserware



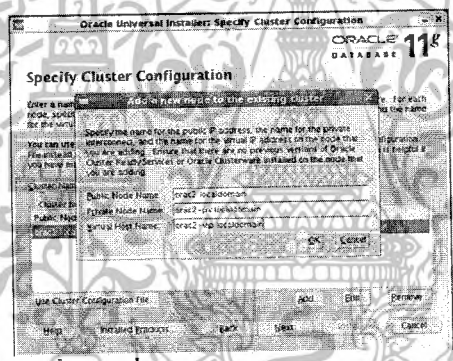
รูปที่ ก.3 path ในการ install clusterware

22. หน้าต่าง registeration node ของ clusterware ขั้นตอนนี้จะเห็นว่ามี node ORAC1 เพียง node เดียว กดปุ่ม add เพื่อเพิ่ม node orac2



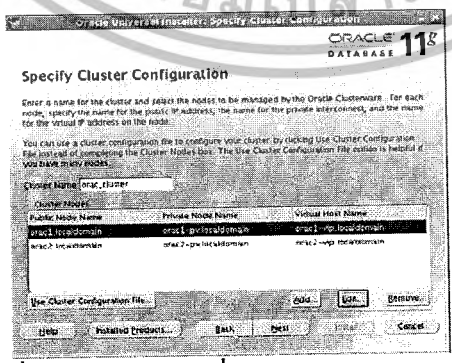
รูปที่ ก.4 กดปุ่ม Add เพื่อเพิ่ม node

23. ทำการใส่ข้อมูลของ node ORAC2 ลงในช่อง แล้วกด ok



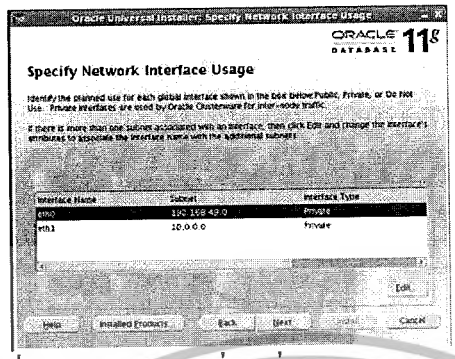
รูปที่ ก.5 เพิ่ม ข้อมูลของ node ORAC2

24. จำนวน node ครบที่ต้องการ กดปุ่ม next เพื่อไปขั้นตอนต่อไป

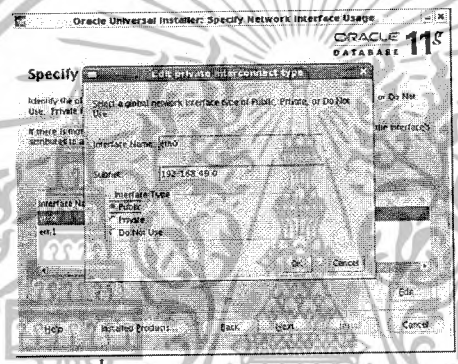


รูปที่ ก.6 แสดง node ที่ regis กับ clusterware

25. จะเห็นว่า interface type ของ eth0 ยังเป็น private อยู่ ทำการเปลี่ยนเป็น public

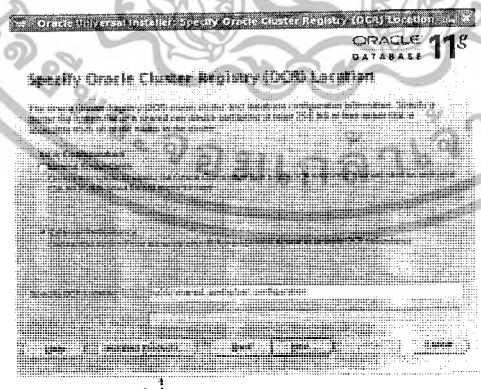


รูปที่ ก.7 กดปุ่ม edit เพื่อเปลี่ยน interface type



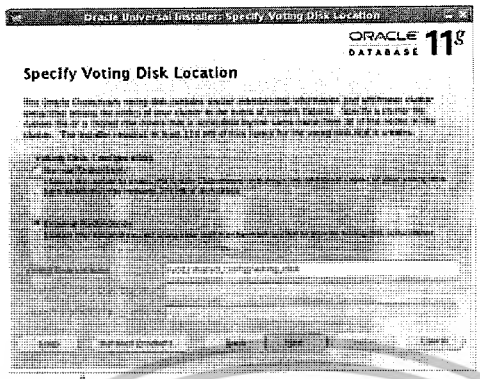
รูปที่ ก.8 เลือก Public แล้วกด ok

26. ระบุ path ของ Oracle Cluster Registry(OCR) ตามที่ได้สร้างไว้ใน ข้อ 16



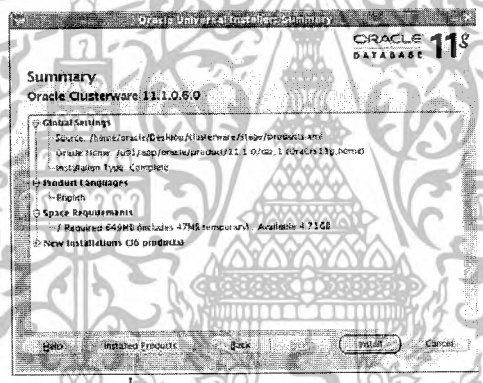
รูปที่ ก.9 ระบุ OCR Path

27. ระบุ Path ของ Voting Disk ตามที่สร้างไว้ในข้อ 16



รูปที่ ก.10 ระบุ path ของ voting disk

28. Summary Screen เพื่อตรวจสอบ Configuration ต่างๆ ก่อนติดตั้ง

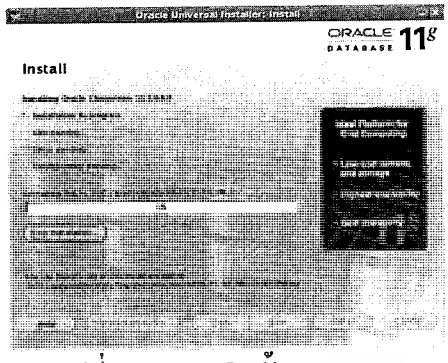


รูปที่ ก.11 Summary screen

29. รอให้ program ติดตั้งเสร็จ ขณะที่ กำลังติดตั้ง program จะมีคำเตือนให้ไป run file ในทั้ง 2 node ตาม path นี้

/u01/app/oraInventory/orainstRoot.sh

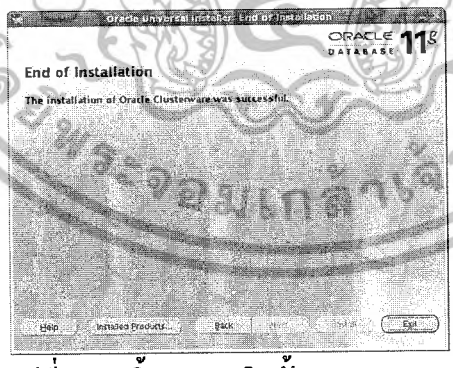
/u01/app/crs/product/11.1.0/crs./root.sh



รูปที่ ก.12 ขณะติดตั้ง program

30. หน้าต่าง Configuration Assistants ช่วยในการตั้งค่าต่างๆ และ start service โดยอัตโนมัติ และจบการติดตั้ง clusterware

รูปที่ ก.13 หน้าต่าง Configuration Assistance



รูปที่ ก.14 สิ้นสุดการติดตั้ง clusterware

ภาคผนวก ข

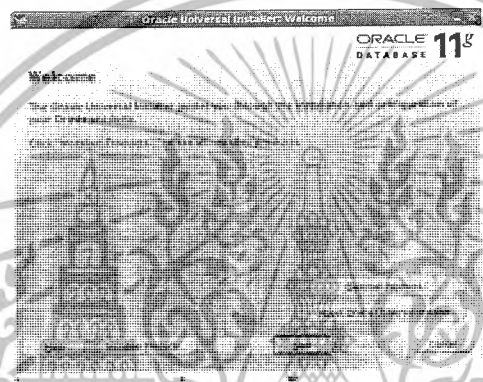
การติดตั้ง ออราเคิล Software

1. Run คำสั่ง

   cd database

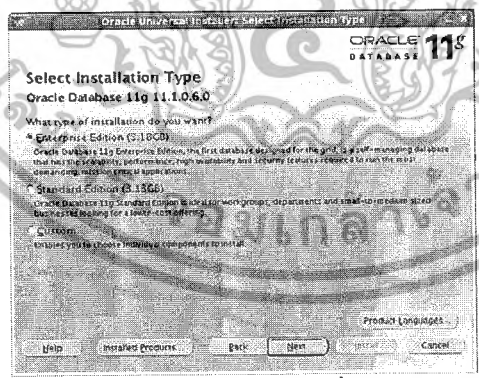
   ./runInstaller

   เพื่อเข้าสู่หน้าต่าง Install



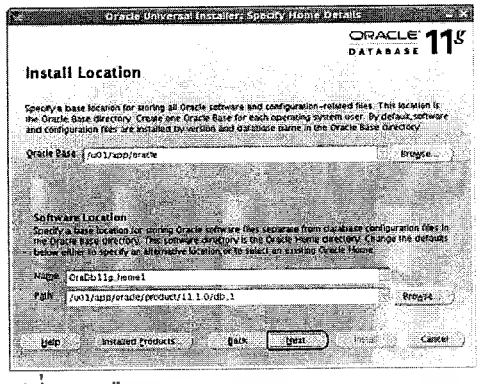
รูปที่ ข.1 หน้าต่างเริ่มการติดตั้ง ออราเคิล software

2. เลือก Enterprise Edition เพื่อให้รองรับ Cluster



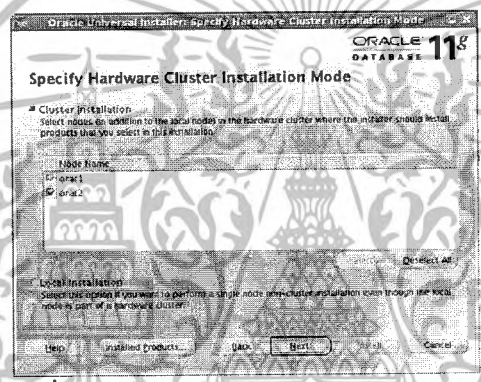
รูปที่ ข.2 เลือก Enterprise Edition เพื่อให้รองรับ Cluster

3. เลือก Path ที่จะ install และ path ของ ORACLE_HOME ตามที่ได้ config ไว้ใน ~/.bash_profile



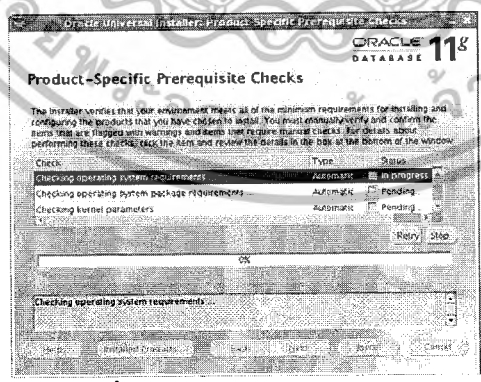
รูปที่ ข.3 เลือก path ของ ORACLE_HOME

4. เลือก node ที่จะให้ทำงานร่วมกันเป็น cluster



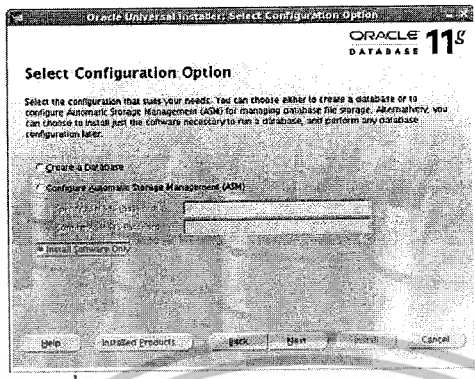
รูปที่ ข.4 เลือก node ORAC1 และ ORAC2

5. Program ทำการเช็คว่า Configuration ต่างๆ ถูกต้องและสามารถติดตั้งได้หรือไม่



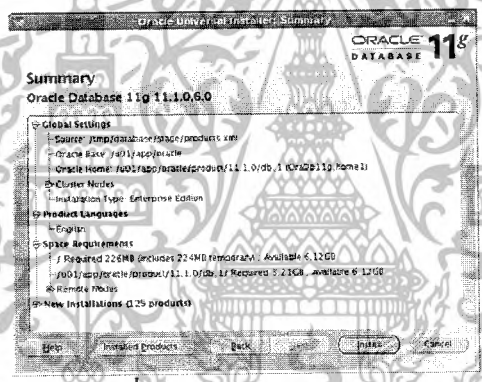
รูปที่ ข.4 Prereauisite Checks

6. เลือก Install Software Only คือการติดตั้งโดยที่ยังไม่มี Database

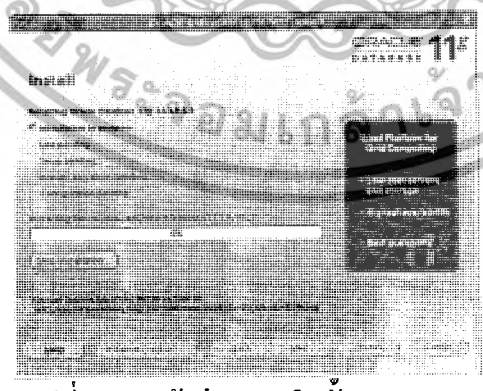


รูปที่ ข. 5 เลือก Install Software Only

7. เข้าสู่หน้า Summary Screen และ กด Install เพื่อติดตั้ง



รูปที่ ข.6 Summary Screen



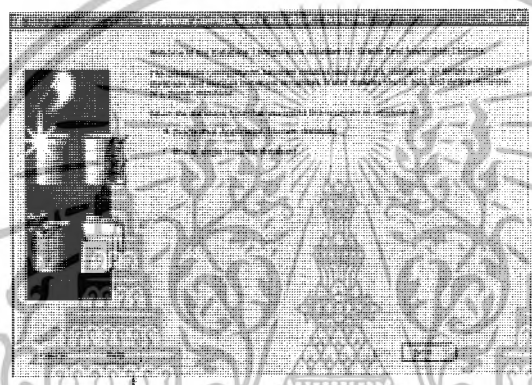
รูปที่ ข.7 หน้าต่างขณะติดตั้ง program

ภาคผนวก ค

การติดตั้ง Database โดยใช้ Program Database Configuration Assistant (DBCA)

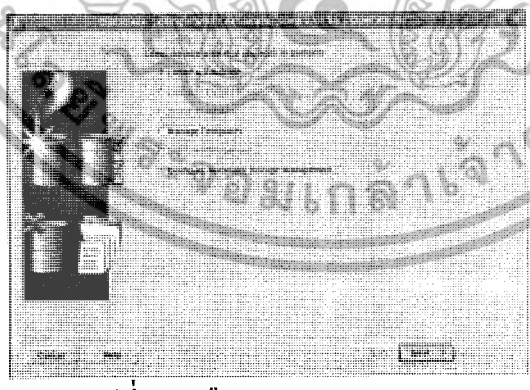
1. Run program โดยใช้คำสั่ง dbca

   เลือก ออราเคิล Real Application Cluster database เพื่อให้ฐานข้อมูลทำงานแบบ Cluster



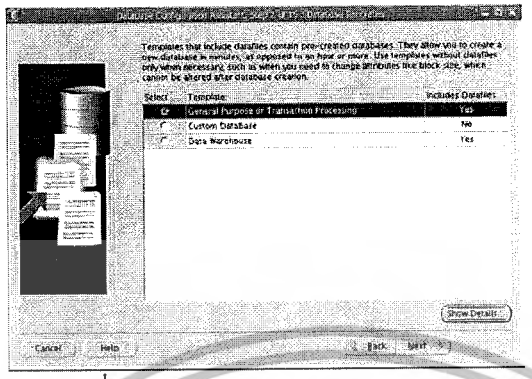
รูปที่ ค.1 หน้าต่าง Program dbca

2. เลือก Create a Database เพื่อสร้าง Database Instance แล้วกด next



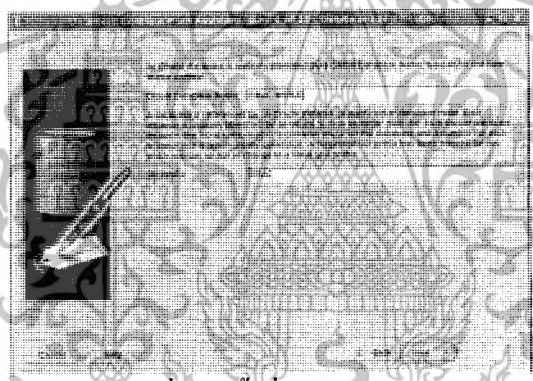
รูปที่ ค.2 เลือก Create Database

3. เลือก General Purpose or Transaction Processing



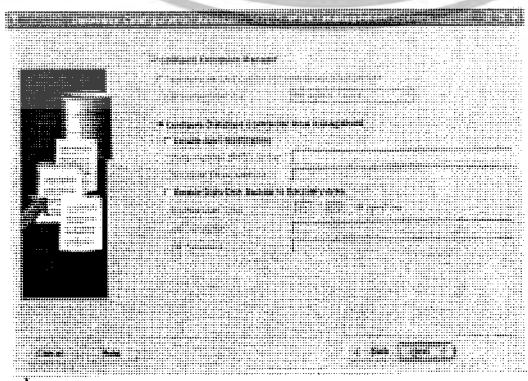
รูปที่ ค.3 เลือกประเภทของ ฐานข้อมูล

4. ตั้งชื่อฐานข้อมูล และ GLOBAL NAME



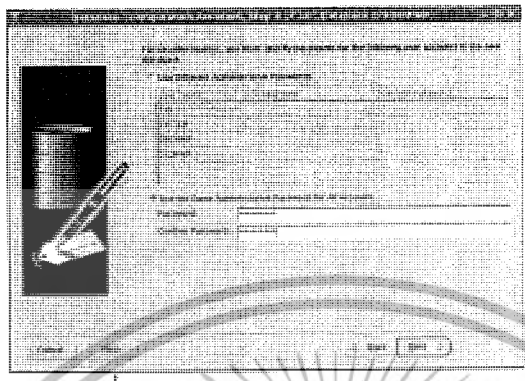
รูปที่ ค.4 ตั้งชื่อ ฐานข้อมูล

5. เลือก Configuration Enterprise Manager เพื่อสร้าง Enterprise Manager ควบคุม Database



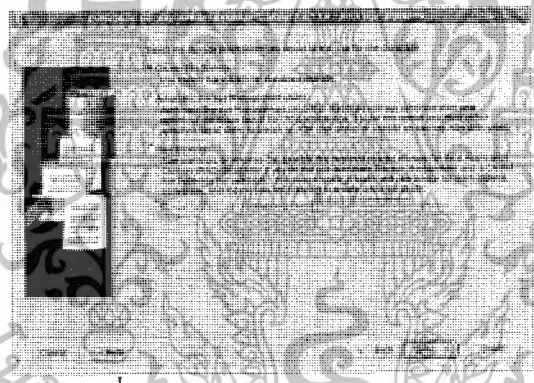
รูปที่ ค.5 เลือก Configuration Enterprise Manager

6. กำหนดรหัสผ่านให้แต่ละ User หรือ ใช้รหัสผ่านเดียว กับทั้ง 4 User



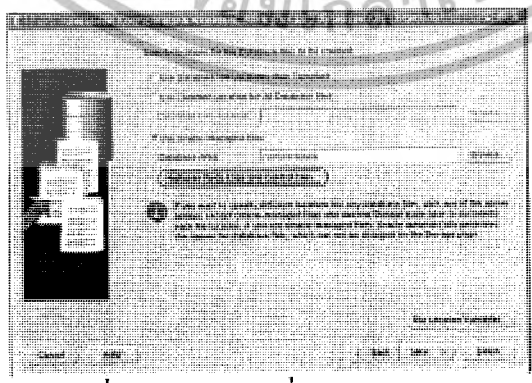
รูปที่ ค.6 กำหนดรหัสผ่านให้ User .

7. เลือก Cluster File System เพื่อให้ ฐานข้อมูลจัดเก็บข้อมูลแบบ Cluster



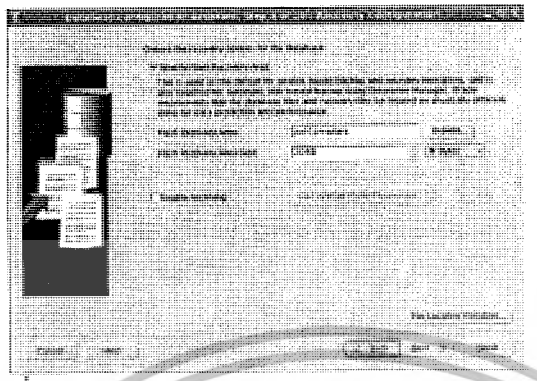
รูปที่ ค.7 เลือก Cluster File System

8. ระบุ Path ที่จะให้เก็บ Data File



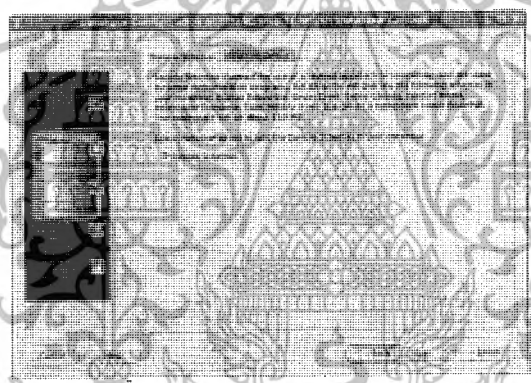
รูปที่ ค.8 ระบุ Path ที่ใช้เก็บ Data File

9. กำหนด Path ของ Flash recovery Area และ Enable Archive mode



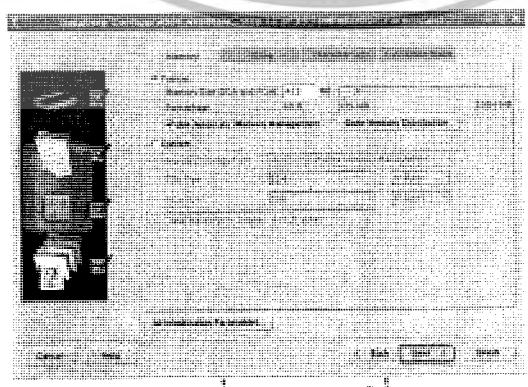
รูปที่ ค. 9 กำหนด Path ให้กับ Flash Recovery Area

10. เลือก Sample Schema หรือ ไม่เลือกก็ได้ แล้วกด Next

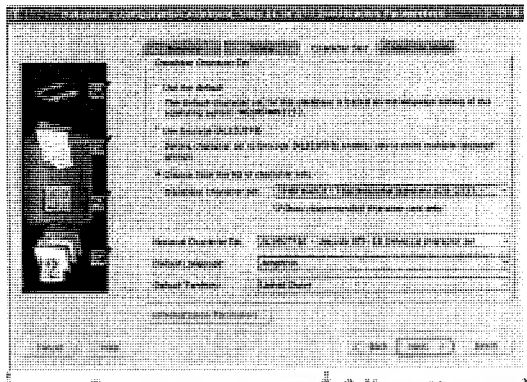


รูปที่ ค.10 เลือก Sample Schema

11. กำหนดขนาดหน่วยความจำ SGA และ Character set เป็น TH8TISACSII เพื่อให้รองรับ ภาษาไทย



รูปที่ ค.11 กำหนดขนาดของ SGA ซึ่ง Default อยู่ที่ 40% ของ Memory ทั้งหมด



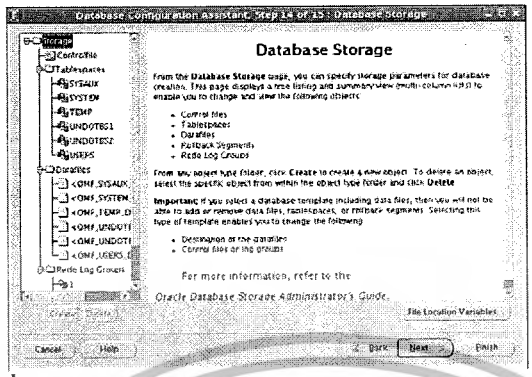
รูปที่ ค.12 เลือก Character set เพื่อให้ รองรับ ภาษาไทย

12. เลือก Enhanced security setting และ Enable Automatic maintenance task

รูปที่ ค.13 เลือก Enhanced security setting

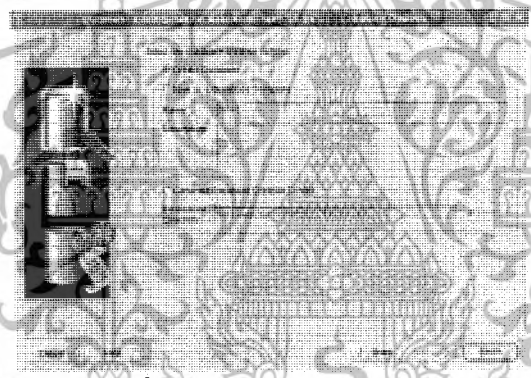
รูปที่ ค.14 เลือก Enable Automatic maintenance task

13. หน้าต่างให้เลือกที่เก็บ redo log, control file เอง



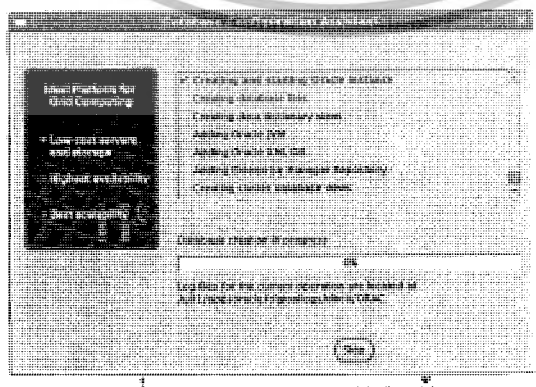
รูปที่ ค.15 หน้าต่างแสดงรายละเอียดต่างๆของ storage

14. เลือก Create Database

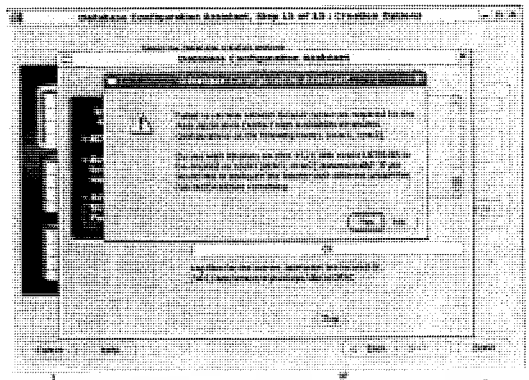


รูปที่ ค.16 เลือก Create Database

15. เข้าสู่หน้าต่างการติดตั้ง และ program จะแจ้งให้เรา คิดตั้ง LISTENER ก่อน โดยใช้ Program netca ให้ click yes เพื่อให้ program ติดตั้งต่อไป



รูปที่ ค.17 Program กำลังติดตั้ง



รูปที่ 18 Program เตือนให้ติดตั้ง Listener ก่อน

การติดตั้ง LISTENER โดยการแก้ไข Configuration File

1. แก้ไข File "$ORACLE_HOME/network/admin/listener.ora" ดังนี้

```
LISTENER_ORAC2 =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = orac2-vip)(PORT =
1521)(IP = FIRST))
   )
   (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.x.x)(PORT =
1521)(IP = FIRST))
   )
   (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
   )
  )
 )

LISTENER_ORAC1 =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
```

```
    (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = orac1-vip)(PORT =
1521)(IP = FIRST))
    )
    (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.x.x)(PORT =
1521)(IP = FIRST))
    )
    (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
    )
   )
  )
```

2. แก้ไข File "$ORACLE_HOME/network/admin/tnsnames.ora" ดังนี้

```
ORAC =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = orac2-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = orac1-vip)(PORT = 1521))
  (LOAD_BALANCE = yes)
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME =ORAC.WORLD)
  )
 )

LISTENERS_ORAC =
 (ADDRESS_LIST =
  (ADDRESS = (PROTOCOL = TCP)(HOST = orac2-vip)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = orac1-vip)(PORT = 1521))
 )

ORAC2 =
```

```
(DESCRIPTION =
 (ADDRESS = (PROTOCOL = TCP)(HOST = orac2-vip)(PORT = 1521))
 (CONNECT_DATA =
  (SERVER = DEDICATED)
  (SERVICE_NAME = ORAC.WORLD)
  (INSTANCE_NAME = RAC2)
 )
)

ORAC1 =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = orac1-vip)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = ORAC.WORLD)
   (INSTANCE_NAME = RAC1)
  )
 )
```

เมื่อติดตั้งเสร็จแล้วเราสามารถ ตรวจสอบว่า มี instance ใด ทำงานอยู่บ้างโดยใช้คำสั่ง

```
$ srvctl config database -d ORAC
orac1 ORAC1 /u01/app/oracle/product/11.1.0/db_1
orac2 ORAC2 /u01/app/oracle/product/11.1.0/db_1
$

$ srvctl status database -d ORAC
Instance ORAC1 is running on node orac1
Instance ORAC2 is running on node orac2
        $
```

หรือเข้า Program sqlplus ดังนี้

```
[oracle@orac1 ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.1.0.6.0 - Production on Thu Jan 22 04:37:36 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options

SQL> SELECT * FROM v$active_instances;

INST_NUMBER INST_NAME
----------- ------------------------------------------------------------
          1 orac1.localdomain:ORAC1
          2 orac2.localdomain:ORAC2
```

รูปที่ ค.19 แสดงการใช้คำสั่งตรวจสอบสถานะ การทำงานของแต่ละ instance

หรือ ใช้ Enterprise Manager เพื่อ ดูสถานะการทำงานต่างๆ

รูปที่ ค.20 แสดงการทำงานของแต่ละ instance ใน program Enterprise Manager

ภาคผนวก ง

การติดตั้งฐานข้อมูล PostgreSQL บนระบบปฏิบัติการ Windows

ทดสอบกับระบบปฏิบัติการ Windows XP 32 bit และ Windows Vista 64 bit

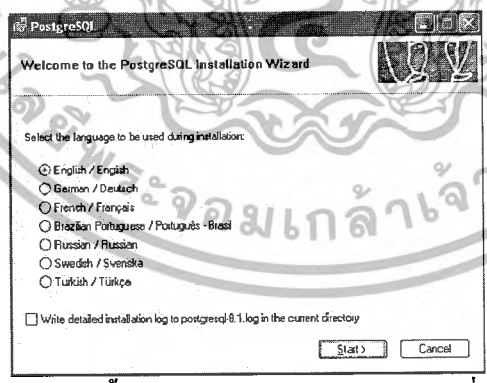
ขั้นตอนการติดตั้ง ขั้นตอนการติดตั้งจะเป็นวิซาร์ด เหมือนกันโปรแกรมอื่นๆที่ติดตั้งภายใต้ระบบปฏิบัติกาวินโดว์ เวอร์ชันที่จะทำการติดตั้งเป็นเวอร์ชันล่าสุดขณะทำการทดสอบ คือเวอร์ชัน 8.3.1 โดยไฟล์ติดตั้งหลังจากดาวน์โหลดมาแล้ว จะประกอบด้วย

postgresql-8.3.1 -int. ไฟล์ข้อมูลติดตั้งหลัก

postgresql-8.3.1 ไฟล์ติดตั้งให้สั่งรันไฟล์ตัวนี้หากต้องการติดตั้งใหม่

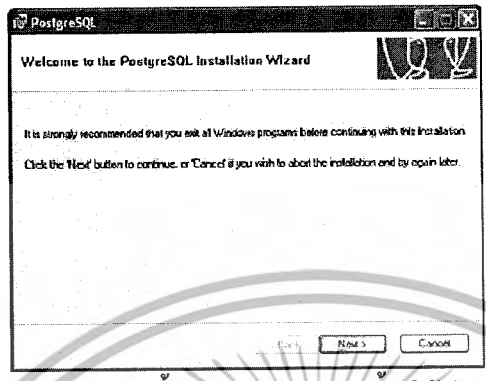
ในขั้นตอนนี้จะทำการติดตั้งใหม่โดยเครื่องที่ยังไม่เคยติดตั้ง postgresql มาก่อน ให้ดับเบิ้ลคลิ๊กที่ไฟล์ postgresql-8.3.1

1. เลือกภาษา เป็นการเลือกภาษาที่วิซาร์ดใช้แสดงในการติดตั้งเท่านั้น ไม่เกี่ยวข้องกับภาษาที่ใช้แสดงในตัวโปรแกรมหลังการติดตั้ง ที่ด้านล่างของหน้าต่างผู้ติดตั้งสามารถเลือกเพื่อให้โปรแกรมติดตั้ง สร้างไฟล์เก็บข้อมูลการติดตั้งไว้ได้ รวมถึงข้อมูลและ password ของ service user และ database superuserเมื่อเลือกครบแล้วให้กด Start



รูปที่ ง.1 การติดตั้งโปรแกรม หน้าแรก เลือกภาษาที่ต้องการ

2. หน้าต่างคำแนะนำเบื้องต้น แนะนำให้ปิดโปรแกรมอื่นก่อนทำการติดตั้ง เพื่อลดปัญหาการขัดแย้งกัน แนะนำให้ทำตาม หลังจากนั้นให้กด Next



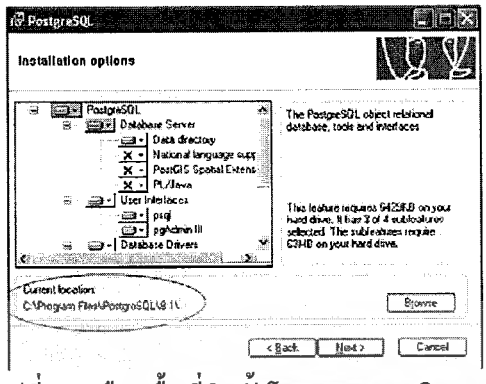
รูปที่ ง.2 คำแนะนำเบื้องต้น เกี่ยวกับการตั้งค่าโปรแกรมม

3. หน้าต่างข้อความต้อนรับ ให้กด Next

รูปที่ ง.3 หน้าต่างข้อความต้อนรับของโปรแปรม

4 .เลือกส่วนประกอบต่างๆของโปรแกรมที่จะทำการติดตั้ง ส่วนประกอบของ Database Server สามารถติดตั้งได้เพียงบน NT based platforms เท่านั้น เนื่องจาก PostgreSQL ต้องการใช้ลักษณะเฉพาะบางอยางของระบบไฟล์ NTFS หากไปติดตั้งบนพาร์ติชันชนิดอื่น ผู้ติดตั้งจะต้องไปรันไฟล์ initdb.exe เพื่อทำการ initial database หลังทำการติดตั้งโปรแกรมเสร็จ

ผู้ติดตั้งสามารถเลือกตำแหน่งติดตั้งตัวโปรแกรมและส่วนประกอบได้โดยอิสระ โดยคลิกเลือกส่วนประกอบแล้วกด browse เพื่อกำหนดตำแหน่งติดตั้ง หรือเลือกทั้งโปรแกรมให้ติดตั้งในที่เดียวกันโดยเลือกที่ PostgreSQL แล้วกด browse เพื่อเลือกตำแหน่งติดตั้ง ในที่นี้เราจะใช้ค่าที่โปรแกรมกำหนดให้เพื่อไม่ให้ยุ่งยากในการจดจำ เสร็จแล้วกด Next เพื่อดำเนินการต่อไป

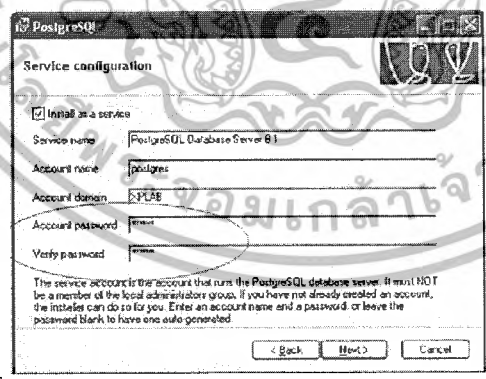


รูปที่ ง.4 เลือกพื้นที่ติดตั้งโพสเกรส เอสคิวแอล

5. ติดตั้งให้ทำงานเป็นเซอร์วิสหน้าต่างนี้จะเป็นการกำหนดให้PostgreSQLติดตั้งเป็น service บนwindows ซึ่งจะมีข้อดีคือ ตัวโปรแกรมจะสามารถทำงานขึ้นมาได้โดยอัตโนมัติ และไม่ต้อง logon windows เพื่อเข้าไป start โปรแกรม ทั้งนี้จะต้องมีการกำหนด account บน windows ที่จะใช้เพื่อ start service ด้วย

กำหนดค่าต่างๆเสร็จแล้วกด Next เพื่อดำเนินการต่อไป

ในการกำหนด Account name “postgres” ถ้าเป็น การติดตั้งครั้งแรก ในเครื่องที่ใช้งานไม่มี user ชื่อ postgres มาก่อน จะมีหน้าต่างขึ้นมาให้ ยืนยันการเพิ่มuser ให้ตอบ Yes และจากนั้นจะมีหน้าต่างขึ้นมาแจ้งว่า รหัสผ่าน(password) ที่ตั้งขึ้นง่ายไปโปรแกรมจะสร้างรหัสผ่าน(password)ให้ใหม่ ให้ตอบ No



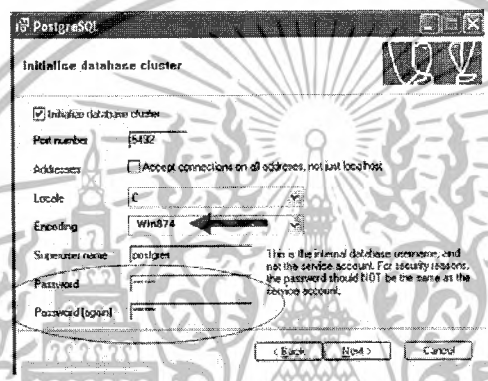
รูปที่ ง.5 เป็น การสร้าง Account name ของ windows

6. Initdb หน้าต่างนี้จะไม่แสดงขึ้นมาหากไม่ได้เลือกติดตั้งให้ทำงานเป็น service (ขั้นตอนที่ 6)ในขั้นตอนนี้เป็นการเลือกว่าต้องการจะสร้างพื้นที่เพื่อจัดเก็บฐานข้อมูล (database cluster) หรือไม่ ถ้าต้องการให้เลือก character set และ encoding และกำหนด ข้อมูลในการ login ดาต้าเบสของ superuser สำหรับ Port number ที่

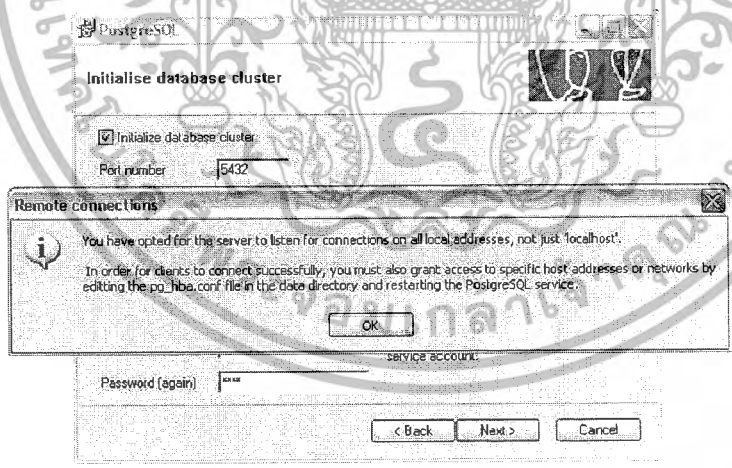
จะให้ server ทำงานสามารถกำหนดเป็นหมายเลขอะไรก็ได้แต่ต้องไม่ซ้ำกับที่ windows เปิดใช้อยู่และ option Accept connections on all IP addresses, not just localhost จะเป็นการกำหนดให้เครื่อง client อื่นสามารถเข้าใช้งาน database บน serverได้

อย่างไรก็ตาม ผู้ติดตั้งจะยังต้องทำการเปิดอนุญาตให้เครื่อง client เข้าใช้งาน server ด้วยตนเองในไฟล์ pg_hba.conf ดังรูป

ถ้าผู้ติดตั้งไม่เลือกที่จะทำการ initialize database ในขั้นตอนนี้ ก็สามารถดำเนินการได้หลังจากติดตั้งโปรแกรมสำเร็จ โดย run ไฟล์ initdb.exe



รูปที่ ง.6 การสร้าง Account name ของผู้ใช้งานและวิธีการเข้ารหัส การเป็นข้อมูล



รูปที่ ง.7 การสร้าง Account name ของผู้ใช้งาน

กำหนดภาษาชุดคำสั่ง (Procedural languages)

เลือกภาษาชุดคำสั่ง(procedural languages) ที่ต้องการใช้ใน template1 จริงๆแล้วทุกชุดภาษาคำสั่ง PL จะถูกติดตั้งลงไปทั้งหมด ในขั้นตอนนี้จึงเป็นเพียงการเปิดใช้งานว่าจะให้ทำงานด้วยชุด PL ชุดใดเป็นค่าเริ่มต้น

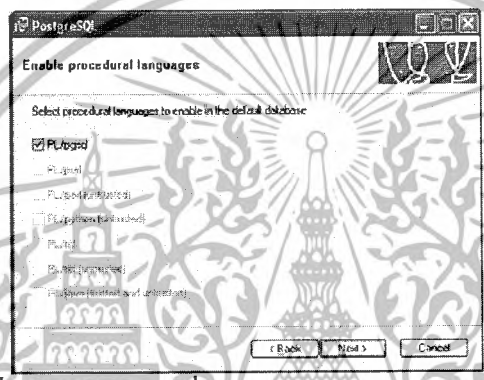
หน้าต่างนี้จะแสดงขึ้นมาเฉพาะที่มีการเลือกติดตั้ง PostgreSQL เป็น service และเลือกที่จะกำหนดค่าเริ่มต้นของ database cluster ผู้ติดตั้งสามารถเลือกได้เฉพาะภาษาที่มี runtime ที่ถูกต้องติดตั้งอยู่เท่านั้น ยกตัวอย่างเช่น

PL/perl จะต้องมี ActiveState Perl 5.8 ติดตั้งอยู่

PL/python จะต้องมี Python 2.3 ติดตั้งอยู่

PL/tcl จะต้องมี ActiveState Tcl 8.4 ติดตั้งอยู่

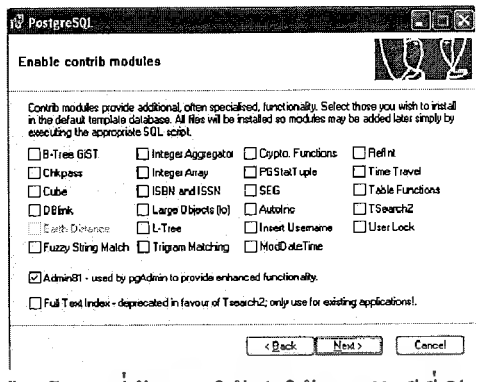
PL/java จะต้องม Sun Java Runtime Environment ติดตั้งอยู่



รูปที่ ง.8 เลือก ภาษาที่ต้องการใช้งานใน PostgreSQL

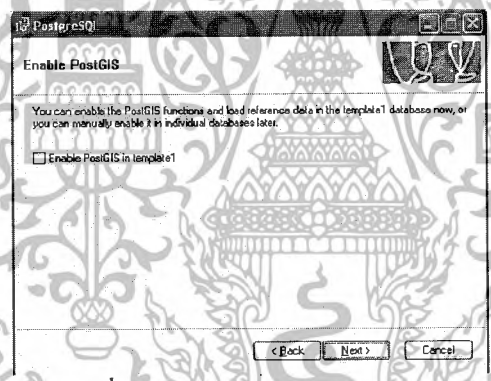
7. เลือก Contrib modules เลือก contrib modules ที่จะใช้งานใน template1 ทุกโมดูลจะถูกติดตั้งตามปกติ แต่โมดูลที่ถูกเลือกในขั้นตอนนี้เท่านั้นจะถูกกำหนดเป็นค่าเริ่มต้นของฐานข้อมูลหน้าต่างนี้จะแสดงขึ้นมาเฉพาะที่มีการเลือกติดตั้ง PostgreSQL เป็น service และเลือกที่จะกำหนดค่าเริ่มต้นของ database cluster

ข้อสังเกตุ : โมดูล Admin81 จะถูกติดตั้งเป็นค่าเริ่มต้น เพราะว่า pgAdmin จำเป็นต้องใช้เพื่อให้เพิ่มเติมความสามารถในการให้บริการ เนื่องจากโมดูลนี้ถูกกำหนดใน template ดังนั้นจึงถูกติดเป็นค่าเริ่มให้กับฐานข้อมูลที่ถูกสร้างขึ้นมาใหม่ทุกตัว ถ้าผู้ติดตั้งไม่ต้องการสามารถเลือกไม่ติดตั้งได้ แต่จะไม่ได้ใชบริการเพิ่มเติมที่โมดูลนี้มีอยู่เสร็จแล้วกด Next เพื่อดำเนินการต่อไป



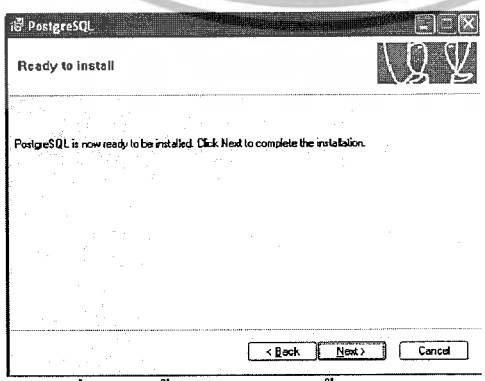
รูปที่ ง.9 เลือกโมดูลที่ต้องการให้เปิดใช้งาน ทันทีที่เปิดโปรแกรม

เปิดใช้งาน PostGIS เลือกว่าจะเปิดใช้งาน PostGIS ใน template1 หรือไม่ มีผลกับทุกฐานข้อมูลที่ถูกสร้างขึ้นมาใหม่ ถ้าไม่เลือกตรงนี้แล้วต้องการใช้งานในภายหลัง สามารถไปเลือกใช้ในแต่ละฐานข้อมูลได้ในภายหลัง เสร็จแล้วกด Next เพื่อดำเนินการต่อไป หน้าต่างนี้จะแสดงขึ้นมาเฉพาะที่มีการเลือกติดตั้งPostGISSpatial Extensions ในขั้นตอนที่ 4



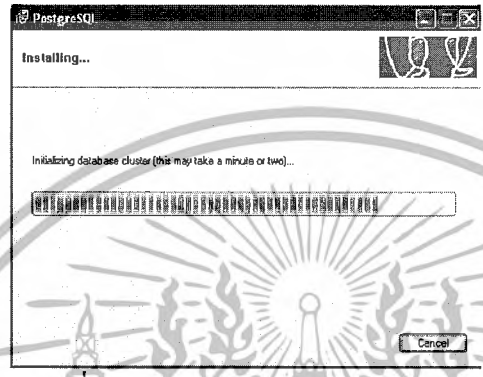
รูปที่ ง.10 การเปิดใช้งาน PostGIS

8. คลิ๊ก Next เพื่อติดตั้งโปรแกรม



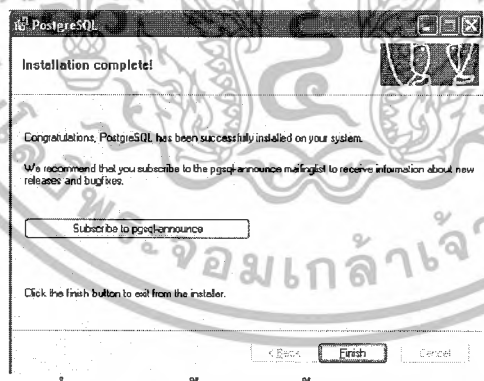
รูปที่ ง. 11 ขั้นตอนการติดตั้ง PostGIS

9. แสดงความก้าวหน้าในการติดตั้ง จะแสดงแถบความคืบหน้าในการติดตั้ง ใน windows บางเวอร์ชันเช่น Windows XP รุ่นก่อนService Pack2 และ Windows 2003 รุ่นก่อน Service Pack1 จะมีหน้าต่าง Command Prompt เปิดขึ้นมาในขั้นตอน "Initializing database cluster" อย่าปิดหน้าต่างนี้ให้ปล่อยทิ้งไว้ ซึ่งมันจะถูกปิดไปเองหลังจากการติดตั้งเสร็จสิ้น



รูปที่ ง.12 การ install กำลังดำเนินการ

10. จบการติดตั้ง ผู้ติดตั้งสามารถไปสมัครเป็นสมาชิกเพื่อรับข่าวสารความเคลื่อนไหวเกี่ยวกับ หลังจากนี้ถ้าผู้ติดตั้งต้องการเพิ่มหรือถอดถอนความสามารถบางอย่างให้ใช้ Add/remove programs จาก Control Panel.



รูปที่ ง.13 เสร็จสิ้นการติดตั้ง PostgreSQL