ตัวประมวลผลภายนอกแบบอิงสถิติ
บนระบบจัดการฐานข้อมูลแบบอิงกฎ
(An External Cost-based Optimizer for Rule-based DBMS)

โดย

นางสาว ลวัณกร โชคอัมพรทรัพย์ 44010403

นางสาว ลัลธิมา ภูษณะพงษ์ 44010404

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

AN EXTERNAL COST-BASED OPTIMIZER FOR RULE-BASED DBMS

ผู้จัดทำ

1. นางสาว ลวัณกร โชคอัมพรทรัพย์ รหัสประจำตัว 44010403
2. นางสาว ลัลธิมา ภูษณะพงษ์ รหัสประจำตัว 44010404

อาจารย์ที่ปรึกษา

(รศ.ดร.ศุภมิตร จิตตะยโศธร)

ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

นางสาว ลวัณกร โชคอัมพรทรัพย์ 44040403
นางสาว ลัลธิมา ภูษณะพงษ์ 44010404
รศ.ดร.ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

ในปัจจุบันมีระบบจัดการฐานข้อมูลที่เป็นแบบอิงสถิติอยู่เป็นจำนวนมาก แต่ผู้ใช้มักจะไม่ทราบถึงวิธีการใช้งาน เนื่องจากผู้ใช้ส่วนใหญ่จะทราบเพียงวิธีในการจัดการฐานข้อมูลแบบอิงกฎเท่านั้น จึงทำให้ผู้ใช้มักจะสนใจเพียงแค่การเขียนคำสั่ง sql ให้ดีที่สุด ซึ่งนับเป็นการใช้งานระบบจัดการฐานข้อมูลที่ไม่เต็มประสิทธิภาพ นอกจากนั้นในการใช้ระบบจัดการฐานข้อมูลแบบอิงสถิตินั้น ยังต้องมีการเก็บสถิติเป็นเวลานาน และยังต้องทำการอัพเดตค่าสถิตินั้นอยู่เสมอ ซึ่งอาจเป็นอีกสาเหตุหนึ่งที่ทำให้ผู้ใช้ไม่สนใจที่จะใช้ระบบจัดการฐานข้อมูลแบบอิงสถิติ

ปริญญานิพนธ์เล่มนี้จะกล่าวถึงวิธีการเปลี่ยนแปลงรูปแบบคำสั่ง sql โดยพิจารณาข้อมูลทางสถิติที่ได้เก็บมาเอง เพื่อให้ผู้ใช้สามารถใช้งานระบบจัดการฐานข้อมูลแบบอิงสถิติได้อย่างเต็มประสิทธิภาพ โดยที่ผู้ใช้โปรแกรมไม่จำเป็นต้องมีความรู้ในเรื่องของระบบจัดการฐานข้อมูลแบบอิงสถิติ และวิธีในการเก็บสถิติ

An External Cost-based Optimizer for Rule-based DBMS

Lawankorn Chokumpornsub

Lulthima Phoosanabhongs

Assoc.Prof.Dr. Suphamit chittayasothon Advisor

Abstract

There are currently several cost-based optimizer for DBMS developed, but most of the users do not know how to fully utilize it. Only with the knowledge about the rule-based optimization, majority of users mostly focus on how to write the sql query to get the answers they want. That is utilization without gaining the performance of optimization. Furthermore, statistical information is required for cost-based optimizer, which may cause the cost-based optimization to be overlooked from users.

This paper introduces the methodology to change the pattern of sql query. The statistical information collected is applied in cost-based optimization so as to fully utilize the optimizer for users, who do not have knowledge of cost-based optimization and collecting the statistical information.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สามารถทำสำเร็จลุล่วงไปด้วยดีก็ด้วยความดูแลจากหลายๆ ฝ่าย โดยเฉพาะอาจารย์ที่ปรึกษา รศ.ดร.ศุภมิตร จิตตะยโศธร ที่คอยให้คำแนะนำให้ข้อมูลต่างๆ ที่เป็นประโยชน์ต่อการทำงานเป็นอย่างมาก

ขอบคุณภาควิชาวิศวกรรมศาสตร์คอมพิวเตอร์ที่เอื้อเฟื้อสถานที่ และเป็นแหล่งรวมข้อมูลต่างๆ ที่เป็นประโยชน์ในการทำงานวิจัย

ทางคณะผู้จัดทำขอมอบวิทยานิพนธ์ฉบับนี้เพื่อเป็นประโยชน์แก่ผู้ที่สนใจต่อไปในอนาคต

นางสาว ลวัณกร โชคอัมพรทรัพย์
นางสาว ลัลธิมา ภูษณะพงษ์

สารบัญ

หน้าที่

สารบัญตาราง

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เนื่องจากในปัจจุบันในการดำเนินการต่างๆนั้น จะมีการใช้งานและการจัดเก็บข้อมูลอยู่เป็นจำนวนมาก ซึ่งไม่ใช่เรื่องง่ายในการที่จะให้มนุษย์เป็นผู้คอยจัดการกับข้อมูลต่างๆเหล่านั้น เพราะอาจทำให้เกิดการผิดพลาดได้ง่าย ทั้งในเรื่องของความถูกต้องของข้อมูล และความรวดเร็วในการทำงาน ดังนั้นจึงเกิดความต้องการในการนำระบบจัดการฐานข้อมูล(Database management system - DBMS) มาใช้งาน เพื่อให้การจัดเก็บ, การเรียกใช้ และการจัดการข้อมูลต่างๆนั้นเป็นไปอย่างมีประสิทธิภาพมากยิ่งขึ้น

ในการติดต่อกับระบบจัดการฐานข้อมูลนั้น ต้องใช้ภาษา sql ในการติดต่อ ซึ่งการที่จะ query ให้ได้ผลลัพธ์ออกมาตามที่ต้องการนั้น อาจเขียนแทนด้วยภาษา sql ได้ในหลายๆทาง โดยที่แต่ละทางนั้น ถึงแม้จะทำให้ผลลัพธ์ที่ได้ออกมานั้นเหมือนกัน แต่ในความเป็นจริงแล้ว การประมวลผลแต่ละคำสั่งนั้น อาจจะต่างกันได้ ซึ่งจะทำให้ประสิทธิภาพของแต่ละทางนั้นไม่เท่ากัน ซึ่งการประมวลผลคำสั่งนั้น จะขึ้นอยู่กับกฎต่างๆที่ใช้ในแต่ละระบบจัดการฐานข้อมูล ซึ่งการประมวลผลเช่นนี้ จะเป็นไปตามการประมวลผลแบบอิงกฎ

ปัจจุบันได้มีการพัฒนาการประมวลผลให้เป็นแบบอิงสถิติมากขึ้น โดยทำการเก็บค่าสถิติต่างๆของข้อมูลที่ใช้งาน และนำค่าสถิตินั้นมาคำนวณหาเส้นทางที่มีค่า cost น้อยที่สุด เพื่อนำเส้นทางนั้นมาทำการ query ให้ได้ผลลัพธ์ตามที่ต้องการ และมีประสิทธิภาพมากที่สุด แต่ในปัจจุบันนั้นยังไม่มีระบบจัดการฐานข้อมูลใดที่สามารถพัฒนาการประมวลผลให้เป็นแบบอิงสถิติได้ถึง 100% ดังนั้นในระบบจัดการฐานข้อมูลส่วนใหญ่จึงใช้การประมวลผลทั้ง 2 แบบผสมควบคู่กันไปในการจัดการฐานข้อมูล

จากที่ได้กล่าวมาข้างต้น รวมกับการที่ผู้ใช้งานส่วนใหญ่ในปัจจุบันนั้น ยังไม่มีความเข้าใจในพื้นฐานของการจัดการข้อมูล และไม่ทราบถึงกฎการใช้งานต่างๆของระบบจัดการฐานข้อมูลอย่างเพียงพอ ทำให้การทำงานกับระบบจัดการฐานข้อมูลส่วนใหญ่ ไม่เป็นไปอย่างเต็มประสิทธิภาพ ซึ่งจะทำให้ผู้ใช้เสียประโยชน์หรืออาจจะทำให้งานต่างๆ เกิดความล่าช้าและเสียหายได้

ดังนั้นโครงงานวิจัยนี้จำทำการเลือกระบบจัดการฐานข้อมูลขึ้นมาหนึ่งตัว เพื่อศึกษาถึงการทำงานและกฎต่างๆที่มีอยู่ในระบบจัดการฐานข้อมูลนั้น และทำการพัฒนาโปรแกรมต้นแบบเพื่อหา query ที่เป็นไปตามกฎที่มีอยู่ของระบบจัดการฐานข้อมูลมากขึ้น นอกจากนั้นยังมีการทดลองเก็บสถิติ เพื่อนำมาใช้ในการพิจารณาหา query ทำให้การทำงานนั้นเป็นแบบอิงสถิติมากขึ้น ซึ่งทำให้ผู้ใช้ที่ต้องการใช้งานระบบฐานข้อมูลนั้นไม่จำเป็นที่จะต้องทราบเกี่ยวกับกฎของฐานข้อมูลนั้นๆ ซึ่งในโครงงานวิจัยนี้ได้เลือกระบบจัดการฐานข้อมูล คือ MySQL ซึ่งจะได้กล่าวถึงต่อไป

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพิ่มความเร็วในการเรียกค้นข้อมูล

1.2.2 ลดต้นทุนในการจัดหาระบบจัดการฐานข้อมูลที่เป็นแบบสถิติมาใช้งาน

1.2.3 เพิ่มความสะดวกสบายให้กับผู้ใช้เนื่องจากผู้ใช้ไม่จำเป็นที่จะต้องมีความรู้ในเรื่องฐานข้อมูลมากนัก

1.2.4 ศึกษาถึงวิธีการประมวลผลของระบบฐานข้อมูล MySQL

1.2.5 ศึกษาถึงวิธีในการเก็บสถิติของฐานข้อมูล เพื่อนำมาใช้ในการพัฒนาประสิทธิภาพของระบบจัดการฐานข้อมูลได้

1.2.6 สามารถนำความรู้ต่างๆที่ได้เรียน มาประยุกต์ให้งานจริงได้

1.3 ขอบเขตของงานวิจัย

โครงงานวิจัยนี้จะสร้างโปรแกรมที่ทำงานเป็นตัวประมวลผลภายนอกให้มีการประมวลผลตรงตามกฎของระบบจัดการฐานข้อมูล MySQL มากขึ้น และจำลองการประมวลผลแบบอิงสถิติ โดยการทำงานของโปรแกรมนั้นจะอยู่ตรงกลางระหว่างระบบจัดการฐานข้อมูล และผู้ใช้ โดยที่ผู้ใช้จะต้องติดต่อกับฐานข้อมูลผ่านทางโปรแกรมนี้เท่านั้น

นอกจากนั้นในโครงงานวิจัยนี้ยังถือว่าเป็นโครการที่ทดลองสร้าง เพื่อศึกษาการทำงาน โดยสมมติว่าผู้ใช้นั้นมีความรู้พื้นฐานในภาษา sql มาแล้ว และมีข้อจำกัดของข้อมูลบางอย่าง เช่น สมมติว่าผู้ไม่สามารถรองรับการใช้งานที่ข้อมูลมีปริมาณมากๆได้ ค่าตัวแปรต่างๆของระบบจัดการฐานข้อมูลจะใช้ค่าที่เป็นค่า default ของ MySQL เป็นต้น

1.4 วิธีการดำเนินงาน

งานวิจัยในโครงงานนี้จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องกับงานวิจัย ซึ่งมีเรื่องหลักๆ คือ ทฤษฎีการประมวลผลฐานข้อมูลแบบกฎ และแบบอิงสถิติ ซึ่งมีรายละเอียดในบทที่ 2 จากนั้นทำการศึกษาถึงโครงสร้างของระบบจัดการฐานข้อมูล MySQL ซึ่งมีรายละเอียดในบทที่ 3 จากนั้นนำเอาความรู้ที่ได้ศึกษามาแล้วทั้งหมด มาออกแบบสถาปัตยกรรมของระบบ ซึ่งมีรายละเอียดในบทที่ 4 จากนั้นทำการพัฒนาโปรแกรมตัวต้นแบบขึ้นมาตามที่ได้ทำการออกแบบไว้ ซึ่งรายละเอียดของการทำงาน และเทคนิคต่างๆที่ใช้จะอยู่ในเนื้อหาปริญญานิพนธ์บทที่ 5 สำหรับในบทที่ 6 นั้นจะเป็นวิธีการทดลอง และผลต่างๆที่ได้จากการทดลอง รวมถึงการสรุปผลการทดลอง และบทที่ 7 นั้นจะเป็นสรุปการดำเนินงานทั้งหมดที่ได้ทำมา และแนวทางในการพัฒนางานวิจัยนี้เพิ่มเติม

บทที่ 2
ทฤษฎีพื้นฐาน

2.1 การดำเนินการกับ query

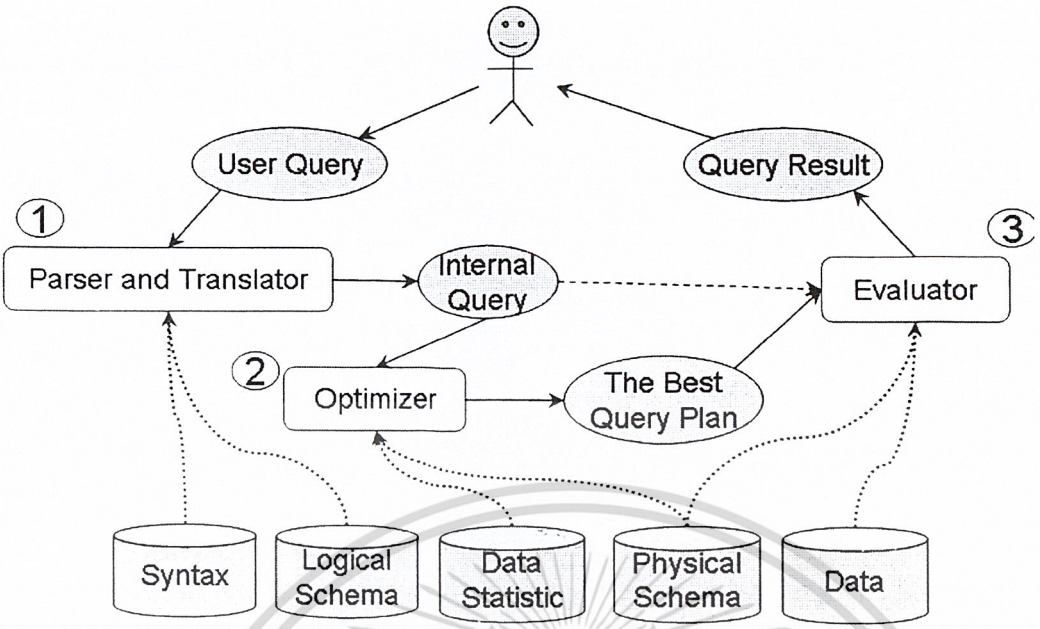
2.1.1 แนวคิดในการดำเนินการกับ query

การดำเนินการกับ query (query processing) จะอ้างอิงกิจกรรมที่เกี่ยวข้องกับการคัดข้อมูลจากฐานข้อมูล ซึ่งกิจกรรมเหล่านี้จะรวมถึงการแปลง query ต่างๆ ในภาษาระดับสูงของฐานข้อมูล (high-level database language) ให้กลายเป็น expression ที่จะสามารถถูกนำไปใช้ได้ในระดับกายภาพ (physical level) ของระบบไฟล์, การทำ query optimization ของ query ที่ถูกแปลงแล้ว และการดำเนินการกับ query นั้นจริงๆ

2.1.2 ขั้นตอนของการดำเนินการกับ query

- วิเคราะห์ query ที่เข้ามา (Parsing) และแปลง query นั้น (Translation)
  - ตัววิเคราะห์คำ (parser) จะตรวจสอบ syntax ของ query และตรวจสอบว่ารีเลชั่น (relation) และแอตทริบิวต์ (attribute) นั้นมีอยู่หรือไม่
  - ตัวแปลงคำ (translator) จะทำการเปลี่ยน query ของผู้ใช้ให้เป็นภาษาที่สามารถใช้ได้กับระดับภายใน (internal representation)
  - ในขั้นตอนนี้ต้องใช้ logical schema ของฐานข้อมูลในการตรวจสอบรีเลชั่น
- Optimization
  - optimizer จะทำการหาเส้นทางที่ดีที่สุดในการดำเนินการกับ query นั้น
  - ในขั้นตอนนี้จะต้องใช้ physical schema ของฐานข้อมูล และสถิติของข้อมูลที่เก็บอยู่ในฐานข้อมูลนั้น
  - DBMS อาจจะข้ามขั้นตอนนี้ไป
- ดำเนินการกับ query
  - ตัวดำเนินการกับ query (query execution engine) จะทำการประมวลผล query นั้น แล้วส่งผลลัพธ์กลับไปให้ผู้ใช้
  - ในขั้นตอนนี้จะต้องใช้ทั้ง database schema และ database content

โดยภาพรวมของการดำเนินการกับ query นั้น จะสรุปได้ดังรูปที่ 2-1



รูปที่ 2-1 แสดงการทำงานของการดำเนินการกับ query

2.1.3 จุดมุ่งหมายในการดำเนินการกับ query คือ

- เพื่อประมวลผล query และส่งผลลัพธ์ให้กับผู้ขอให้ได้เร็วที่สุดเท่าที่จะเป็นไปได้
- ในขั้นตอนที่ 1 "วิเคราะห์ และแปลง query" จะไม่ใช้เวลามากนัก และมีความซับซ้อนน้อย
- ในขั้นตอนที่ 3 "ประมวลผล" จะเป็นขั้นตอนที่ใช้เวลานานที่สุดในการดำเนินการ แต่ขั้นตอนที่ 2 "optimization" จะสามารถช่วยในการหาเส้นทางที่เร็วที่สุดในการดำเนินการตามขั้นตอนที่ 3
- ในขั้นตอนที่ 2 "optimization" จะสามารถกระทำได้ 2 ลักษณะ (และสามารถกระทำร่วมกันได้) ดังนี้
    - Cost-based Query Optimization : โดยจะทำการประมาณค่า cost สำหรับแต่ละแผนการดำเนินการกับ query แล้วเลือกแผนที่ใช้ cost น้อยที่สุด
    - Rule-based Query Optimization : เป็นการใช้กฎที่ได้ประกาศเอาไว้ก่อนแล้วในการดำเนินการกับ query ให้มีแผนที่ดีที่สุด

2.1.4 การหาแผนการดำเนินการกับ query

เมื่อเราส่ง query เข้าไปแล้ว ส่วนใหญ่แล้วจะมีวิธีการมากมายในการที่จะคำนวณหาคำตอบออกมา ตัวอย่างเช่น ใน SQL นั้น query หนึ่งๆ จะสามารถแสดงออกมาได้หลายแบบ เพราะมันสามารถแปลเป็น relational-algebra ได้หลายแบบ ซึ่ง relational-algebra จะเป็นตัวแทนส่วนหนึ่งที่จะบอกถึงวิธีการในการประเมินผล query นั้นๆ โดยตัวดำเนินการ (operation) ที่ใช้ใน relational-algebra จะมีดังนี้

1.) Select (Restrict) : จะเลือกเฉพาะแถวที่สอดคล้องกับเงื่อนไข โดยใช้สัญลักษณ์ $\prod$

2.) Project : จะเลือกเฉพาะคอลัมน์ที่ต้องการ โดยใช้สัญลักษณ์ σ

3.) Join (Natural Join) : จะจับคู่ค่าที่เท่ากันของ common attribute จากนั้น common attribute จะปรากฏที่เอาต์พุตเพียงครั้งเดียว ใช้สัญลักษณ์ $\bowtie$

4.) Union

5.) Intersection

6.) Difference

7.) Product (Cartesian Product)

8.) Divide

โดยในวิทยานิพนธ์นี้จะสนใจเฉพาะ Select, Project และ Join เท่านั้น ต่อไปนี้จะเป็นตัวอย่างการประมวลผลของ relational-algebra โดยพิจารณา query ต่อไปนี้

SELECT balance

FROM account

WHERE balance < 2500

โดย query นี้จสามารถแปลเป็น relational-algebra expression ได้ดังนี้

- $\sigma_{balance<2500}(\prod_{balance}(account))$
- $\prod_{balance}(\sigma_{balance<2500}(account))$

นอกจากนั้น เรายังสามารถดำเนินการกับตัวดำเนินการของ relational-algebra นี้ได้ในอีกหลายอัลกอริทึม ตัวอย่างเช่น ถ้าเราจะทำตามแบบที่ใช้ Select ก่อน เราจะต้องค้นหา tuple ทุกตัวใน account เพื่อจะหาว่า tuple ใดที่มี balance น้อยกว่า 2500 แต่ถ้าดัชนีที่เป็น B^+-tree นั้นสามารถใช้ได้บนแอตทริบิวต์ balance แล้ว เราจะใช้ดัชนีแทนตำแหน่งของ tuple

ในการที่จะบอกได้ถึงวิธีการประมวลผล query นั้น จะไม่ใช่เพียงแค่การจัดการกับ relational-algebra expression เท่านั้น แต่ยังจะต้องทำหมายเหตุประกอบที่เป็นคำสั่งว่าจะใช้วิธีใดในการประมวลผลแต่ละตัวดำเนินการด้วย ซึ่งการทำหมายเหตุประกอบ (hint) นั้นอาจจะเป็นการระบุถึงอัลกอริทึมที่จะใช้กับตัวดำเนินการที่ระบุไว้ หรือเป็นการบอกว่าจะใช้ดัชนีตัวใดก็ได้ โดยการทำหมายเหตุประกอบนี้จะเรียกว่า "การประเมินผลล่วงหน้า (evaluation primitive)" และลำดับของการประเมินผลล่วงหน้านั้นยังสามารถนำไปใช้ในการประมวลผล query ได้อีก ซึ่งจะเรียกว่า "แผนการดำเนินการ (execution plan)"

แผนการประมวลผลที่ต่างกันของ query เดียวกันจะมีค่า cost ที่ต่างกันด้วย และเราคงไม่คาดหวังให้ผู้ใช้เขียน query ในแบบที่เราได้แนะนำแล้วว่าจะได้แผนการดำเนินการที่ดีที่สุด ดังนั้นจึงกลายเป็นหน้าที่ของระบบที่จะต้องสามารถสร้างแผนการประมวลผลที่จะลด cost ในการดำเนินการกับ query ให้เหลือน้อยที่สุด

ในการที่จะประมวลผล query ใดๆ ได้นั้น query optimizer จะต้องทราบค่า cost ของแต่ละตัวดำเนินการก่อน ถึงแม้ว่าค่าที่แน่นอนจะคำนวณออกมาได้ยาก เพราะต้องขึ้นอยู่กับตัวแปรอีกมากมาย แต่มันก็สามารถที่จะประเมินค่า cost นั้นได้อย่างคร่าวๆ

2.1.5 การวัด cost ของ query

cost ของการประมวลผล query นั้น สามารถวัดได้ในเทอมของจำนวนทรัพยากรที่ต่างกัน ซึ่งรวมถึงการเข้าถึงดิสก์ (disk), เวลาของ CPU ในการดำเนินการกับ query และ cost ในการสื่อสาร โดยเวลาในการประมวลผล query นั้น (ยังรวมถึงเวลาที่ใช้ในการวางแผนการดำเนินการด้วย) จะเกิดจาก cost เหล่านี้รวมกัน จึงสามารถวัดค่า cost ของแต่ละแผนได้

ในระบบฐานข้อมูลที่มีขนาดใหญ่นั้น การเข้าถึงดิสก์ (ซึ่งวัดจากจำนวนบล็อกที่ส่งผ่านจากดิสก์) มักจะเป็น cost ที่สำคัญที่สุด เพราะการเข้าถึงดิสก์จะช้ามากเมื่อเทียบกับการทำงานบนหน่วยความจำ (memory) ยิ่งไปกว่านั้นความเร็วของ CPU ในปัจจุบันนี้เร็วกว่าการเข้าถึงดิสก์มาก ดังนั้นจึงเหมือนกับว่าเวลาที่ใช้กับดิสก์จะมีอิทธิพลมากต่อความเร็วทั้งหมดในการดำเนินการกับ query เพราะฉะนั้นส่วนใหญ่จึงมักใช้ cost ของการเข้าถึงดิสก์ในการวัด cost ของแผนการดำเนินการกับ query

เราใช้จำนวนของบล็อกที่ส่งผ่านจากดิสก์ในการวัด cost ที่ใช้จริง เพื่อให้สามารถคำนวณ cost ในการเข้าถึงดิสก์ได้อย่างง่ายๆ เราจะสมมุติให้การส่งผ่านของทุกบล็อกมีค่า cost เท่ากัน แต่ถ้าเราต้องการค่าที่มีความแม่นยำมากกว่านี้ เราจะต้องแยกค่า sequential I/O, เมื่อบล็อกที่ถูกอ่านอยู่ติดกัน, และค่า random I/O, เมื่อบล็อกที่ถูกอ่านไม่ได้อยู่ติดกัน, ออกจากกัน และยังต้องเพิ่มค่า cost ที่ใช้ในการค้นหาเพิ่มเติมสำหรับแต่ละการทำงานของ I/O ของดิสก์ แล้วเรายังต้องแยกการอ่านบล็อกและการเขียนบล็อกออกจากกันด้วย เนื่องจากการเขียนบล็อกลงดิสก์จะใช้เวลามากกว่าการอ่านบล็อกจากดิสก์ แต่ถ้าต้องการให้คำนวณได้แม่นยำมากยิ่งขึ้นไปอีก เราจะต้องคำนึงถึงสิ่งต่อไปนี้ด้วย

1. จำนวนของการทำงานเพื่อค้นหา
2. จำนวนของบล็อกที่ถูกอ่าน
3. จำนวนของบล็อกที่ถูกเขียน

จากนั้นบวกค่าเหล่านี้เข้าไปหลังจากคูณมันด้วยเวลาเฉลี่ยของการค้นหา, เวลาเฉลี่ยในการส่งผ่านบล็อกที่ถูกอ่าน และค่าเฉลี่ยในการส่งผ่านบล็อกที่ถูกเขียนตามลำดับ แต่ query optimizer ในชีวิตจริงจะรวมค่า cost ของ CPU เข้าไปในการคำนวณด้วย

2.1.6 อัลกอริทึมที่ใช้เมื่อมีการ JOIN

ในส่วนนี้จะพูดถึงอัลกอริทึมต่างๆ ที่ใช้ในการคำนวณเมื่อมีการ join แล้วจะทำการวิเคราะห์ถึง cost ที่เกิดขึ้นจากอัลกอริทึมเหล่านั้น โดยอัลกอริทึมจะมีดังนี้

1.) Nested-Loop Join

2.) Block Nested-Loop Join

3.) Indexed Nested-Loop Join

4.) Merge Join

5.) Hash Join

6.) Complex Join

แต่ภายในวิทยานิพนธ์นี้จะทำการอ้างอิงเพียง 2 อัลกอริทึม นั่นคือ Nested-Loop Join และ Indexed Nested-Loop Join จึงจะขออธิบายเพียง 2 อัลกอริทึมนี้เท่านั้น

2.6.1.1 Nested-Loop Join

เป็นอัลกอริทึมพื้นฐานในการดำเนินการกับ join ระหว่าง 2 relation ใดๆ ($r \bowtie s$) โดยอัลกอริทึมจะเป็นดังนี้

for each tuple t_r in r **do begin**

  for each tuple t_s in s **do begin**

    test pair(t_r, t_s) to see if they satisfy the join condition

    if they do, add $t_r.t_s$ to the result.

  end

end

โดยรีเลชั่น r จะเรียกว่า "รีเลชั่นตัวนอก (outer relation)" และรีเลชั่น s จะเรียกว่า "รีเลชั่นตัวใน (inner relation)" ของการ join เพราะลูป (loop) ของ r จะหุ้มลูปของ s ไว้ โดยในอัลกอริทึมนี้จะใช้สัญลักษณ์ $t_r.t_s$,เมื่อ t_r และ t_s เป็น tuple, โดยเป็นการแสดงถึง tuple ที่จะถูกสร้างโดยการรวมกันของค่าในแอตทริบิวต์ของ tuple t_r และ t_s

อัลกอริทึมนี้จะไม่ต้องการดัชนี และยังสามารถถูกใช้โดยไม่จำเป็นต้องคำนึงถึงเงื่อนไขในการ join ซึ่งจะเหมือนกับอัลกอริทึมที่เป็นการสแกนไฟล์ธรรมดา แต่จะเพิ่มการคำนวณ natural join โดยตรงเข้าไป เพราะ natural join สามารถใช้กับการ join ในแบบที่จะกำจัดแอตทริบิวต์ที่ซ้ำกันจากการทำ Project ซึ่งสิ่งเดียวที่ต้องเปลี่ยนแปลง คือ ขั้นตอนพิเศษในการลบแอตทริบิวต์ที่ซ้ำกันจาก tuple $t_r.t_s$ ก่อนที่จะเพิ่มเข้าไปในผลลัพธ์

อัลกอริทึมนี้จะใช้ cost สูง เพราะต้องทำการตรวจสอบทุกๆ คู่ของ tuple ในรีเลชั่นทั้งสอง เมื่อพิจารณา cost ของอัลกอริทึมนี้ จะได้จำนวนคู่ของ tuple เท่ากับ $n_r * n_s$, เมื่อ n_r คือ จำนวน tuple ของ r และ n_s คือจำนวน tuple ของ s, โดยแต่ละเรคอร์ด (record) ของ r เราจะต้องทำการสแกนทั้งหมดของ s ซึ่งในกรณีที่แย่ที่สุดนั้น บัฟเฟอร์จะสามารถรองรับได้เพียงหนึ่งบล็อก (block) ของรีเลชั่นนั้น แล้วจะพบว่าต้องเข้าถึงบล็อกเป็นจำนวน $n_r * b_r + b_s$ ครั้ง, เมื่อ b_r คือ จำนวนบล็อกที่บรรจุ tuple ของ r และ b_s คือ จำนวนบล็อกที่บรรจุ tuple ของ s, ส่วนในกรณีที่ดีที่สุดนั้น คือ จะมีที่ว่างในหน่วยความจำมากเพียงพอที่จะเก็บทั้งสองรีเลชั่นพร้อมกัน ซึ่งทำให้ในแต่ละบล็อกถูกอ่านเพียงครั้งเดียว ดังนั้นจะทำการเข้าถึงบล็อกเพียง $b_r + b_s$ ครั้งเท่านั้น

แต่ถ้ามีเพียงรีเลชั่นเดียวเท่านั้นที่สามารถเก็บได้พอดีในหน่วยความจำ การนำรีเลชั่นตัวในไปเก็บในหน่วยความจำจะให้ประสิทธิภาพที่ดีกว่า เพราะมันจะถูกอ่านเพียงครั้งเดียวเท่านั้น ดังนั้น ถ้า s มีขนาดเล็กพอที่เก็บในหน่วยความจำ อัลกอริทึมนี้จะต้องเข้าถึงบล็อกเพียง $b_r + b_s$ เท่านั้น ซึ่งเป็น cost ที่เท่ากับกรณีที่สามารถเก็บทั้งสองรีเลชั่นไว้ภายในหน่วยความจำ

2.6.1.2 Indexed Nested-Loop Join

ในการทำอัลกอริทึมแบบ Nested-Loop Join นั้น ถ้ามีการใช้ดัชนีบนแอตทริบิวต์ที่อยู่ภายในลูปของการ join แล้ว การค้นหาด้วยดัชนีนั้นจะสามารถใช้แทนการสแกนทั้งไฟล์ได้ โดยแต่ละ tuple t_r ของรีเลชั่นตัวนอก r นั้น ดัชนีจะถูกใช้ในการค้นหา tuple ใน s ที่ตรงกับเงื่อนไขของการ join

การ join แบบนี้จะเรียกว่า "Index Nested-Loop Join" ซึ่งสามารถใช้ได้เมื่อมีดัชนี หรือมีดัชนีชั่วคราวที่ถูกสร้างเพื่อประมวลผล join

cost ของการ join นั้นสามารถคำนวณได้ ดังนี้ แต่ละ tuple ในรีเลชั่นตัวนอก r, การค้นหาจะเกิดขึ้นบนดัชนีสำหรับ s แล้ว tuple ที่สัมพันธ์กันจะถูกนำออกมา ซึ่งในกรณีที่แย่ที่สุด คือ บัฟเฟอร์จะมีที่ว่างไว้เก็บเพจ (page) ของ r ได้เพียงเพจเดียว และเก็บเพจของดัชนีได้เพียงเพจเดียวเช่นกัน ซึ่งจะทำให้ต้องทำการเข้าถึงดิสก์ เพื่ออ่านรีเลชั่น r เป็นจำนวน b_r ครั้ง เมื่อ b_r คือ จำนวนบล็อกที่เก็บเรคอร์ด r และแต่ละ tuple ใน r เราจะทำการค้นหาด้วยดัชนีบน s ดังนั้น เราจะสามารถคำนวณ cost ได้ดังนี้ $b_r + n_r * c$ เมื่อ n_r คือ จำนวนเรคอร์ดของ r และ c คือ cost ในการเลือกบน s โดยใช้เงื่อนไขของการ join

จากสูตรการคำนวณ พบว่า ถ้ามีดัชนีอยู่บนทั้งสองรีเลชั่นแล้ว โดยปกติ การใช้ relation ที่มีจำนวน tuple น้อยกว่าเป็นรีเลชั่นตัวนอกนั้น จะมีประสิทธิภาพมากกว่า

2.2 Query Optimization

เป็นกระบวนการในการเลือกแผนการประมวลผล query ที่มีประสิทธิภาพที่สุดจากแผนที่เป็นไปได้จำนวนมาก โดยมักใช้กับ query ที่มีความซับซ้อน เนื่องจากเราไม่คาดหวังว่าผู้ใช้จะทราบวิธีการ query ที่มีประสิทธิภาพที่สุด ดังนั้นเราจึงคาดหวังให้ระบบสามารถสร้างแผนในการประมวลผล query ที่ใช้ cost น้อยที่สุดได้ ซึ่ง query optimization จะมาจัดการในจุดนี้

มุมมองหนึ่งในการประมวลผลจะเกิดขึ้นในระดับของ relational-algebra ซึ่งระบบจะพยายามหา expression ที่เท่ากันกับ expression ที่ได้มา แต่สามารถดำเนินการได้มีประสิทธิภาพมากกว่า แต่ในอีกมุมมองหนึ่งจะเป็นการเลือกวิธีการในการดำเนินการกับ query เช่น เลือกอัลกอริทึมมาใช้ดำเนินการกับ ตัวดำเนินการ, เลือกดัชนีที่จะใช้ เป็นต้น

ความแตกต่างของ cost (ในแง่ของเวลาในการประมวลผล) ระหว่างวิธีการที่ดีกับวิธีการที่ไม่ดีนั้นมักจะค่อนข้างมาก ดังนั้นมันจึงคุ้มค่าที่จะใช้เวลาที่มากมายนั้นในการดำเนินการกับ query แทน

โดยการทำ query optimization นั้น จะสามารถกระทำได้ 2 ทาง ดังนี้

2.2.1 การประมวลผล query แบบอิงกฎ (Rule-based query optimization)

เป็นการนำกฎที่ได้กำหนดไว้แล้วมาใช้สร้างแผนในการ query เพื่อที่จะให้ได้แผนที่ดีกว่า :

- สามารถทำ Project ได้ทุกเวลาในทันทีที่เป็นไปได้เพื่อลดขนาดของรีเลชั่น เช่น ขนาดของ แถว (row)
- ควรทำ Select ให้เร็วที่สุดเท่าที่จะเป็นไปได้ เพื่อลดขนาดของรีเลชั่น เช่น จำนวนของแถว
- ทำการ join รีเลชั่นที่เล็กเป็นอันดับแรก จากนั้นจึงค่อยทำการ join กับรีเลชั่นขนาดใหญ่

2.2.2 การประมวลผล query แบบอิงสถิติ (Cost-based query optimization)

ในแบบอิงสถิตินั้น ตัวประมวลผลจะตัดสินใจว่าแผนในการดำเนินการใดที่มีประสิทธิภาพที่สุด โดยจะพิจารณาจากเส้นทางที่เป็นไปได้ และปัจจัยด้านข้อมูลที่เกี่ยวกับสถิติสำหรับ schema object (ตารางหรือดัชนี) ที่จะถูกเข้าถึงจากสเตตเมนต์ (statement) sql นั้น ซึ่งโดยปกติแล้วจุดมุ่งหมายในการทำแบบอิงสถิตินั้นคือ เพื่อให้มี throughput ที่ดีที่สุด หรือให้มีการใช้ทรัพยากรที่จำเป็นน้อยที่สุด

โดยหลักการแล้ววิธีแบบอิงสถิติจะทำตามขั้นตอนต่อไปนี้

1. ตัวประมวลผลจะสร้างแผนการดำเนินการที่เป็นไปได้ทั้งหมดของสเตตเมนต์ sql นั้น โดยจะมีพื้นฐานอยู่บนเส้นทางที่ใช้เข้าถึง (access path) และหมายเหตุประกอบ (hint)
2. ตัวประมวลผลจะประมาณค่า cost ของแต่ละแผนการดำเนินการ โดยจะใช้สถิติที่เก็บอยู่ใน data dictionary สำหรับการกระจายของข้อมูล และการเก็บลักษณะของตาราง, ดัชนี และส่วนที่จะถูกเข้าถึงจากสเตตเมนต์นั้น โดยค่า cost นั้น จะเป็นค่าที่ประมาณจากสัดส่วนของทรัพยากรมันที่คาดว่าจะใช้ในการดำเนินการกับสเตตเมนต์นั้นเมื่อทำตามแผนที่วางไว้ โดยตัวประมวลผลจะคำนวณค่า cost ของแต่ละวิธีการเข้าถึงที่เป็นไปได้ และลำดับการ join โดยยึดหลักในการประมาณทรัพยากรของคอมพิวเตอร์ ซึ่งรวมถึงเวลาของ I/O, เวลาของ CPU และหน่วยความจำที่จะใช้ดำเนินการกับสเตตเมนต์ด้วยแผนนั้นๆ
3. ตัวประมวลผลจะเปรียบเทียบค่า cost ของแต่ละแผนดำเนินการ แล้วเลือกแผนที่มีค่า cost น้อยที่สุด

บทที่ 3

MySQL

3.1 เครื่องมือที่ใช้เก็บรักษาข้อมูลของ MySQL และประเภทของตาราง

เครื่องมือที่ใช้เก็บรักษาข้อมูลของ MySQL มี 2 อย่างคือ แบบที่จัดการตารางแบบทรานแซคชั่น-เซฟ (transaction-safe) และตารางแบบนัน-ทรานแซคชั่น-เซฟ (non-transaction-safe)

ตารางแบบทรานแซคชั่น-เซฟ มีข้อได้เปรียบตารางแบบนัน-ทรานแซคชั่น-เซฟ ดังนี้

- ปลอดภัยกว่า เพราะเมื่อเกิดการพัง (crash) หรือฮาร์ดแวร์(hardware) เกิดปัญหา จะสามารถกู้ข้อมูลกลับคืนมาได้ โดยการทำการกู้ข้อมูลแบบอัตโนมัติ หรือ จากการนำข้อมูลสำรอง (backup) มาบวกกับล็อกของทรานแซคชั่น (transaction log)
- สามารถรวมหลายคำสั่ง และยอมรับมันทั้งหมดได้พร้อมๆกัน โดยใช้คำสั่งยืนยัน (commit) (ถ้า การยืนยันโดยอัตโนมัตินั้นไม่อนุญาติ)
- สามารถทำการปฏิบัติคำสั่ง ROLLBACK เพื่อเพิกเฉยต่อการเปลี่ยนแปลงที่เกิดขึ้นได้ (ถ้า การยืนยันโดยอัตโนมัตินั้นไม่อนุญาติ)
- ถ้าการอัพเดต(update) ล้มเหลวแล้ว ทุกๆการเปลี่ยนแปลงจะถูกคืนค่ากลับ(restore)
- สามารถเตรียมการเรื่อง concurrency ได้ดีกว่าสำหรับตารางที่อัพเดตหลายๆอันพร้อมกัน

ตารางแบบนัน-ทรานแซคชั่น-เซฟนั้นมีประโยชน์หลายอย่างในตัวเอง เพราะไม่เปลืองค่าใช้จ่ายในการทำทรานแซคชั่น

- เร็วกว่า
- ร้องขอพื้นที่ในดิสก์น้อยกว่า
- ร้องขอพื้นที่ในหน่วยความจำในการทำการอัพเดตน้อยกว่า

คุณสามารถรวมตารางที่เป็นแบบทรานแซคชั่น-เซฟ และแบบนัน-ทรานแซคชั่น-เซฟได้ในคำสั่งที่เหมือนกัน เพื่อนำเอาสิ่งที่ดีที่สุดของทั้ง 2 แบบนี้ออกมา อย่างไรก็ตามถ้าในทรานแซคชั่นซึ่งมีการตั้งการยืนยันโดยอัตโนมัติแบบไม่อนุญาตินั้น การเปลี่ยนแปลงที่เกิดกับตารางนัน-ทรานแซคชั่น-เซฟนั้นจะยังคงถูกยืนยันแบบฉับพลันและไม่สามารถทำการ roll back ได้

โดยในวิทยานิพนธ์นี้จะสนใจเฉพาะเครื่องมือที่ใช้เก็บรักษาข้อมูลชนิด InnoDB เท่านั้น จึงขออธิบายเพียง InnoDB เท่านั้น

3.2 เครื่องมือที่ใช้ในการเก็บข้อมูลชนิด InnoDB

3.2.1 คำอธิบายเบื้องต้นเกี่ยวกับ InnoDB

InnoDB เป็นเครื่องมือที่ใช้ในการเก็บข้อมูลที่จะจัดการเกี่ยวกับทรานแซคชั่นให้กับ MySQL โดยจะมีความสามารถในการ commit, roll back และกู้ข้อมูลคืนได้เมื่อเกิด crash โดย InnoDB จะทำการล็อค (lock) ในระดับของแถว และยังจัดการในวิธีเดียวกับ Oracle ในเรื่องของการอ่านโดยไม่มีการล็อค (non-locking read) ในสเตตเมนต์ SELECT ซึ่งลักษณะนี้จะช่วยเพิ่มประสิทธิภาพในการทำงานพร้อมๆ กันของผู้ใช้หลายคน และไม่จำเป็นต้องเพิ่มล็อคใน InnoDB เพราะการล็อคในระดับแถวของ InnoDB นั้นจะใช้พื้นที่น้อยมาก นอกจากนั้น InnoDB ยังสนับสนุน FOREIGN KEY constraint และใน SQL query นั้น เราก็ยังสามารถใช้ตารางแบบ InnoDB ร่วมกับตารางแบบอื่นของ MYSQL ได้อย่างอิสระแม้จะอยู่ใน query เดียวกัน

InnoDB นั้น ได้ถูกออกแบบมาเพื่อให้สามารถดำเนินการกับข้อมูลที่มีขนาดใหญ่ๆ ได้อย่างมีประสิทธิภาพสูงสุด โดยประสิทธิภาพของ CPU อาจจะไม่ตรงกันกับเครื่องมือของฐานข้อมูล (database engine) อื่นๆ ที่เกี่ยวกับความสัมพันธ์โดยอิงดิสก์ (disk-based relational)

เมื่อใช้กับเซิร์ฟเวอร์ของ MySQL อย่างสมบูรณ์แล้ว InnoDB จะดูแลบัฟเฟอร์ของตัวเอง สำหรับใช้ในการแคช (cache) ข้อมูล และดัชนีในหน่วยความจำหลัก โดย InnoDB จะเก็บตารางและดัชนีของมันเองใน tablespace ซึ่งอาจจะประกอบด้วยไฟล์หลายไฟล์ และนี่คืออีกส่วนหนึ่งที่แตกต่างจากตารางแบบอื่น เช่น MyISAM ที่จะเก็บแต่ตารางในไฟล์ที่แยกกัน

3.2.2 คอนฟิกูเรชันของ InnoDB

ใน MySQL 4.0 ขึ้นไป ตารางแบบ InnoDB จะเป็นตัวเลือกอัตโนมัติ โดยทรัพยากรที่เป็นอิงดิสก์ (disk-based) ที่สำคัญ และถูกจัดการโดย InnoDB คือไฟล์ข้อมูล tablespace (tablespace data file) และไฟล์ล็อก (log file) ของมันเอง

ถ้าเราไม่ได้ระบุทางเลือก (option) ในคอนฟิกูเรชันใดๆ ใน InnoDB เลย ใน MySQL 4.0 ขึ้นไปจะทำการสร้างไฟล์ข้อมูล (data file) ขนาด 10MB ที่สามารถขยายขนาดได้อัตโนมัติ ชื่อว่า "ibdata1" และไฟลล็อกขนาด 5MB 2 ตัว ชื่อว่า "ib_logfile0" และ "ib_logfile1" ขึ้นมาภายในไดเรกทอรีของข้อมูล (data directory) ของ MySQL (ใน MySQL 4.0.0 และ MySQL 4.0.1 นั้น จะสร้าง data file ขนาด 64MB ที่ไม่สามารถขยายขนาดได้โดยอัตโนมัติ ส่วน MySQL 3.23 นั้น InnoDB จะไม่เริ่มทำงาน จนกว่าจะจัดการในส่วนทางเลือกของคอนฟิกูเรชัน)

3.2.3 โครงสร้างของตารางและดัชนี

MySQL จะทำการเก็บข้อมูล data dictionary ของตารางไว้ในไฟล์ '.frm' ที่อยู่ในไดเรกทอรีของฐานข้อมูล แต่ทุกๆ ตารางของ InnoDB นั้นจะมีการบันทึกข้อมูลของตัวเองลงใน InnoDB internal data dictionary ภายใน tablespace เมื่อ MySQL ทำการยกเลิก (drop) ตารางหรือฐานข้อมูลนั้น มันจำเป็นที่จะต้องลบทั้งไฟล์ '.frm' และส่วนที่เกี่ยวข้องกันภายใน data dictionary ของ InnoDB ด้วย

ทุกๆตารางของ InnoDB จะมีดัชนีพิเศษเรียกว่า "ดัชนีคลัสเตอร์ (clustered index)" ซึ่งมีข้อมูลของแถวเก็บอยู่ ถ้ามีการระบุ PRIMARY KEY ไว้บนตารางแล้ว ดัชนีของ PRIMARY KEY จะเป็นดัชนี คลัสเตอร์ แต่ถ้าไม่ได้ระบุไว้ MySQL จะหยิบดัชนีตัวแรกที่ไม่ซ้ำกัน (unique) ซึ่งภายในคอลัมน์นั้นจะไม่มีค่า null อยู่เลยมาเป็น PRIMARY KEY และจะใช้มันเป็นดัชนีคลัสเตอร์ แต่ถ้าไม่มีดัชนีในตารางนั้น InnoDB จะทำการสร้างดัชนคลัสเตอร์ในที่ซึ่งแถวนั้นๆ จะเรียงตาม ID ของแถวที่ InnoDB กำหนดให้แก่แถวในตาราง

การเข้าถึงแถวในตารางโดยผ่านดัชนีคลัสเตอร์จะเร็ว เพราะข้อมูลของแถวจะอยู่บนเพจเดียวกัน เมื่อมีดัชนีค้นหาก่อน ซึ่งถ้าตารางมีขนาดใหญ่ การใช้โครงสร้างของดัชนีคลัสเตอร์จะช่วยประหยัดดิสก์ I/O เมื่อเทียบกับวิธีที่เคยใช้มา

ใน InnoDB นั้นเรคอร์ดในดัชนีที่ไม่ใช่ดัชนีคลัสเตอร์ (non-clustered index) ซึ่งเรียกได้อีกชื่อว่า secondary index จะเก็บค่า primary key ของแถว และ InnoDB จะใช้ค่านี้เพื่อทำการค้นหาแถวจากดัชนี คลัสเตอร์ ข้อสังเกต คือ ถ้า primary key มีขนาดยาวแล้ว secondary index จะใช้พื้นที่มากขึ้น

3.2.4 โครงสร้างทางกายภาพของดัชนี

ทุกๆ ดัชนีใน InnoDB เป็น B^+-trees ซึ่งดัชนีของเรคอร์ดจะถูกเก็บไว้ในเพจล่างสุดของ tree (leaf pages) ค่าขนาดของเพจที่เป็นค่าตัวเลือกอัตโนมัติ คือ 16 KB เมื่อมีเรคอร์ดใหม่ถูกเพิ่มเข้าไป (insert) InnoDB จะพยายามกันพื้นที่ ไว้ 1/6 ของเพจไว้สำหรับการในอนาคต และการอัพเดตเรคอร์ดของดัชนี

ถ้าเรคอร์ดของดัชนีถูกเพิ่ม (insert) ตามลำดับแล้ว ผลลัพธ์ของเพจของดัชนีจะมีค่าเต็มประมาณ 15/16 ของ page แต่ถ้าเรคอร์ดถูกเพิ่มแบบไม่เรียงลำดับแล้ว เพจจะเป็น 1/2 จนถึง 15/16 และถ้า fillfactor ของเพจของดัชนีนั้น ตกลงมาต่ำกว่า 1/2 แล้ว InnoDB จะพยายามย่อ tree ของดัชนี เพื่อให้มีเพจอิสระมากขึ้น (free page)

3.2.5 การบัฟเฟอร์ของการเพิ่ม (insert buffering)

ในสถานการณ์ปกติ แอพพลิเคชันของฐานข้อมูลนั้น จะมี primary key เป็นตัวระบุ (identifier) ที่ไม่ซ้ำกัน และแถวใหม่จะถูกเพิ่มเข้าไปเป็นลำดับจากน้อยไปมากใน primary key และด้วยเหตุนี้เองทำให้ ดัชนีคลัสเตอร์ไม่ต้องอ่านค่าจากดิสก์แบบสุ่ม (random) ในทางตรงกันข้าม secondary index ปกติแล้วอาจจะมีค่าซ้ำกันเป็น และการเพิ่มเข้าไปที่ secondary index จะเกิดขึ้นเป็นลำดับแบบสุ่มที่มีความเกี่ยวข้องกัน (relatively random order) ซึ่งเป็นสาเหตุให้เกิดดิสก์ I/O แบบสุ่มเป็นจำนวนมาก ซึ่ง InnoDB ยังไม่มีวิธีพิเศษใดในการจัดการกับเรื่องนี้

ถ้าเรคอร์ดของดัชนีถูกเพิ่มไปยัง secondary index ที่มีค่าซ้ำแล้ว InnoDB จะทำการตรวจสอบดูว่าเพจของ secondary index นั้นอยู่ในกลุ่มบัฟเฟอร์ (buffer pool) หรือยัง ถ้าอยู่แล้วก็จะเพิ่มโดยตรงไปที่เพจของดัชนีนั้นเลย แต่ถ้ายังไม่เจอเพจของดัชนีนั้นในกลุ่มบัฟเฟอร์แล้ว InnoDB จะเพิ่มเรคอร์ดไปยังโครงสร้างในการเพิ่มบัฟเฟอร์ (insert buffer structure) แบบพิเศษ แล้วบัฟเฟอร์ที่เพิ่มเข้าไปจะถูกเก็บให้มีขนาดเล็กพอที่จะเก็บทั้งหมดของมันไว้ในกลุ่มบัฟเฟอร์ได้ ซึ่งทำให้สามารถเพิ่มได้อย่างรวดเร็ว

ในสมัยก่อนบัฟเฟอร์ที่เพิ่มเข้าไปจะถูกรวมลงใน secondary index tree ของฐานข้อมูล และเป็นไปได้บ่อยครั้งที่จะทำการรวมเพิ่มหลายๆ อันไปยังเพจเดียวกันของ tree ของดัชนี เพื่อประหยัดดิสก์ I/O และได้มีการพิสูจน์มาแล้วว่า การเพิ่มบัฟเฟอร์จะสามารถเพิ่มความเร็วในการเพิ่มข้อมูลลงในตารางได้มากถึง 15 เท่า

3.2.6 Adaptive Hash Index

ถ้าตารางนั้นวางได้เกือบทั้งหมดในหน่วยความจำแล้ว วิธีที่เร็วที่สุดในการทำ query คือ การใช้ดัชนีแบบแฮช (hash index) ซึ่ง InnoDB จะมีกลไกแบบอัตโนมัติ ที่จะเฝ้าสังเกตการมองหาดัชนี (index search) ไปเป็นดัชนีที่ระบุ (index defined) สำหรับตาราง ถ้า InnoDB สังเกตเห็นว่า query น่าจะได้ประโยชน์จากการสร้างดัชนีแบบแฮชแล้ว มันจะทำให้โดยอัตโนมัติ

สังเกตว่าดัชนีแบบแฮชนั้นถูกสร้างโดยมีพื้นฐานมาจากดัชนีแบบ B-tree ซึ่งมีอยู่บนตารางเสมอ โดย InnoDB สามารถสร้างดัชนีแบบแฮชบนคำนำหน้า (prefix) ของความยาวของ key ใดๆ ที่นิยามไว้สำหรับ B-tree ขึ้นอยู่กับรูปแบบของการค้นหาที่ InnoDB จะกระทำกับดัชนีที่เป็นแบบ B-tree โดยสำหรับดัชนีแบบแฮชนั้น สามารถใช้เป็นแค่ส่วนหนึ่งได้ เนื่องจากมันไม่ได้ต้องการให้ดัชนีแบบ B-tree ทั้งหมดจะต้องถูกแคช (cache) ลงในกลุ่มบัฟเฟอร์ ซึ่ง InnoDB จะสร้างดัชนีแบบแฮชตามความต้องการสำหรับเพจเหล่านั้นของดัชนีที่ถูกเข้าถึงบ่อยๆ

3.2.7 โครงสร้างของเรคอร์ดทางกายภาพ

เรคอร์ดในตาราง InnoDB นั้นจะมีลักษณะดังต่อไปนี้

3.2.7.1 แต่ละเรคอร์ดของดัชนีจะบรรจุส่วนหัว (header) ขนาด 6 bytes ซึ่งส่วนหัวจะถูกใช้ เพื่อเชื่อมต่อเรคอร์ดที่ติดกันเข้าด้วยกัน

3.2.7.2 เรคอร์ดในดัชนีคลัสเตอร์ จะมีการบรรจุฟิลด์สำหรับคอลัมน์ของการกำหนดของผู้ใช้ (user-defined) ทั้งหมดไว้ นอกจากนั้น ยังมีฟิลด์ขนาด 6 bytes สำหรับ ID ของทรานแซคชั่น และฟิลด์ขนาด 7 bytes สำหรับพอยเตอร์ของ roll (roll pointer)

3.2.7.3 ถ้าตารางนั้นไม่มีการกำหนด primary key เอาไว้ แต่ละเรคอร์ดของดัชนีคลัสเตอร์จะบรรจุค่า ID ซึ่งเป็นฟิลด์ขนาด 6 bytes

3.2.7.4 แต่ละเรคอร์ดของ secondary index จะบรรจุฟิลด์ทั้งหมดที่กำหนดไว้สำหรับ key ของดัชนีคลัสเตอร์เอาไว้ด้วย

3.2.7.5 เรคอร์ดจะบรรจุพอยเตอร์ที่ชี้ไปยังแต่ละฟิลด์ของเรอคร์ดเอาไว้ด้วย ถ้าความยาวทั้งหมดของฟิลด์ใน 1 เรคอร์ดนั้นน้อยกว่า 128 bytes แล้วพอยเตอร์จะมีขนาด 1 byte แต่ถ้าเป็นแบบอื่นจะมีขนาด 2 bytes

3.2.7.6 ภายใน InnoDB จะเก็บคอลัมน์แบบตัวอักษรที่มีขนาดตายตัว เช่น CHAR(10) ในรูปแบบความยามที่แน่นอน (fixed-length) และ InnoDB จะตัดพื้นที่ส่วนท้ายจากคอลัมน์ที่เป็น VARCHAR สังเกตว่า MySQL อาจจะทำการเปลี่ยนคอลัมน์จาก CHAR เป็น VARCHAR ให้ภายใน

3.2.7.7 SQL ที่มีค่าเป็น NULL นั้นจะจองพื้นที่ 0 byte ถ้าเก็บอยู่ในคอลัมน์ที่มีความยาวแปรผันได้ สำหรับคอลัมน์ที่มีความยาวตายตัวนั้น มันจะจองพื้นที่สำหรับคอลัมน์ไว้ตายตัว

3.2.8 ข้อจำกัดของตาราง InnoDB

3.2.8.1 ตาราง 1 ตาราง มีคอลัมน์ได้ไม่เกิน 1,000 คอลัมน์

3.2.8.2 ความยาวมากที่สุดของแถวยกเว้นคอลัมน์แบบ BLOB และ TEXT นั้นจะมีค่าน้อยกว่าค่าเพจของฐานข้อมูลอยู่เล็กน้อย นั่นคือ ความยาวมากที่สุดของแถวนั้นจะมีค่าประมาณ 8,000 bytes

3.2.8.3 ในบางระบบปฏิบัติการนั้น ขนาดไฟล์ของข้อมูล (data files) ต้องน้อยกว่า 2 GB

3.2.8.4 ขนาดของไฟล์ล็อกของ InnoDB นั้น เมื่อรวมกันแล้วต้องมีค่าไม่เกิน 4 GB

3.2.8.5 ขนาดของ tablespace ที่น้อยที่สุดคือ 10 MB และขนาดของ tablespace ที่มากที่สุดคือ 64 TB (เพจของฐานข้อมูล 4 ล้านเพจ) และยังเป็นขนาดที่มากที่สุดของ 1 ตารางด้วย

3.2.8.6 InnoDB ไม่สนับสนุนดัชนีแบบ FULLTEXT

3.2.8.7 บนระบบปฏิบัติการแบบ Windows นั้น InnoDB จะเก็บชื่อของฐานข้อมูล และตารางไว้ภายในโดยใช้อักษรเป็นตัวพิมพ์เล็กเสมอ

3.2.8.8 คำเตือน คือ ห้ามแปลงตารางของระบบของ MySQL ที่อยู่ในฐานข้อมูลชื่อ mysql จากตารางประเภท MyISAM ไปเป็นตารางประเภท InnoDB เด็ดขาด ถ้าทำเช่นนั้นแล้ว MySQL จะไม่ทำการเริ่มใหม่จนกว่าคุณจะทำการนำตารางเก่าของระบบกลับมาจากการแบ็กอัพ หรือทำการสร้างมันขึ้นมาใหม่ด้วย สคริปต์ของ mysql_install_db

3.2.8.9 InnoDB จะไม่เก็บข้อมูลการนับจำนวนแถวไว้ภายในตาราง ในการประมวลผลคำสั่ง SELECT COUNT(*) FROM T นั้น InnoDB จะต้องทำการสแกนดัชนีของตาราง ซึ่งจะต้องใช้เวลาในการทำบ้างถ้าหากตารางนั้นไม่ได้อยู่บนกลุ่ม (buffer pool) ทั้งหมด ถ้าตารางของคุณไม่มีการเปลี่ยนแปลงบ่อยมากนักการใช้แคช query ของ MySQL (MySQL query cache) ถือเป็นทางออกที่ดี สำหรับคำสั่ง SHOW TABLE STATUS นั้นสามารถใช้มาประมาณจำนวนแถวอย่างคร่าวๆ ได้

3.2.8.10 สำหรับคอลัมน์ที่เป็น AUTO_INCREMENT คุณต้องกำหนดดัชนีให้กับตารางเสมอ และดัชนีนั้นต้องมีแค่คอลัมน์ที่เป็น AUTO_INCREMENT เท่านั้น ซึ่งถ้าเป็นตารางแบบ MyISAM นั้น คอลัมน์ที่เป็น AUTO_INCREMENT นั้น อาจจะเป็นส่วนหนึ่งของดัชนีแบบหลายคอลัมน์ก็ได้

3.2.8.11 เมื่อคุณทำการเริ่มเซิร์ฟเวอร์ของ MySQL ใหม่นั้น InnoDB อาจจะใช้ค่าเก่าสำหรับคอลัมน์ที่เป็น AUTO_INCREMENT

3.2.8.12 เมื่อคอลัมน์ที่เป็น AUTO_INCREMENT นั้นรันไปจนเกินค่าของมันแล้ว InnoDB จะทำการแปลงค่า BIGINT ไปเป็น 9223372036854775808 และ BIGINT UNSIGNED ไปเป็น 1 แต่อย่างไรก็ตาม ค่าของ BIGINT นั้นมี 64 บิต ดังนั้นสังเกตว่าถ้าคุณทำการเพิ่มข้อมูลจำนวนหนึ่งล้านแถวต่อวินาทีแล้ว ยังต้องใช้เวลาทั้งหมดหนึ่งล้านปี ก่อนที่จะถึงค่าขอบเขตบนของ BIGINT

3.2.8.13 การทำคำสั่ง DELETE FROM tbl_name นั้น จะไม่สร้างตารางขึ้นมาใหม่ แต่จะทำการลบข้อมูลทุกแถวของตารางแทน ซึ่งจะทำการลบทีละแถว (one by one)

3.2.8.14 คำสั่ง TRUNCATE tbl_name นั้นใน InnoDB จะทำเหมือนคำสั่ง DELETE FROM tbl_name และจะไม่ทำการรีเซต (reset)ค่าเคาน์เตอร์ (counter)

3.2.8.15 คำสั่ง SHOW TABLE STATUS นั้นไม่ได้ให้ค่าสถิติที่แท้จริงของตารางแบบ InnoDB ยกเว้นค่าขนาดทางกายภาพ (physical) ที่ถูกจองโดยตาราง การนับจำนวนแถวจะเป็นเพียงการประเมินค่าแบบหยาบๆ เพื่อใช้ในการประมวณผลคำสั่ง SQL

3.2.8.16 คำสั่ง INSERT DELAYED นั้นไม่สนับสนุนสำหรับตารางแบบ InnoDB

3.2.8.17 ค่าขนาดเพจของฐานข้อมูล (database page) ที่เป็นตัวเลือกอัตโนมัตินั้น ใน InnoDB มีขนาด 16 KB คุณสามารถกำหนดค่าให้มันได้ตั้งแต่ 8 KB ถึง 64 KB โดยการรีคอมไพล์โค้ด คุณต้องทำการอัพเดตค่าของ UNIV_PAGE_SIZE และ UNIV_PAGE_SIZE_SHIFT ในไฟล์ (source file) ที่มีชื่อว่า 'univ.i'

3.3 การประมวลผลของ MySQL (MySQL Optimization)

3.3.1 อธิบายเบื้องต้นถึงสาเหตุที่ต้องมีการประมวลผล

ปัญหาคอขวดของระบบคือ

1. การค้นหาดิสก์ (Disk seeks) ระบบต้องการเวลาเพื่อจะหาชิ้นข้อมูลในดิสก์ สำหรับดิสก์รุ่นใหม่ เวลาเฉลี่ยที่ใช้สำหรับกรณีนี้ปกติแล้วจะน้อยกว่า 10 ms ดังนั้นตามทฤษฎีเราสามารถทำการหาข้อมูลได้ 100 ครั้งใน 1 วินาที การพัฒนาในเรื่องนี้จะเป็นไปอย่างช้าๆ ในดิสก์รุ่นใหม่ และเป็นการยากที่จะประมวลผล (optimize) ในตารางเดี่ยว (single table) วิธีที่จะประมวลผลเวลาในการหาข้อมูลคือ ต้องกระจายข้อมูลไปอยู่บนดิสก์มากกว่า 1 ดิสก์

2. การอ่านและเขียนบนดิสก์ (Disk reading and writing) เมื่อดิสก์อยู่ในตำแหน่งที่ถูกต้องแล้ว และเราต้องการอ่านข้อมูลขึ้นมา สำหรับดิสก์รุ่นใหม่ 1 อัน จะมีปริมาณงานที่ทำในช่วงเวลาหนึ่ง (throughput) อย่างน้อย 10 – 20 MB/s ซึ่งกรณีนี้จะง่ายต่อการประมวลผลมากกว่าการหาข้อมูล เพราะเราสามารถอ่านข้อมูลแบบขนานได้จากหลายๆ ดิสก์

3. วัฏจักรของซีพียู (CPU cycles) เมื่อเรามีข้อมูลในหน่วยความจำหลัก และเราต้องการใช้งานมันเพื่อที่จะให้ได้ผลลัพธ์ออกมานั้น การมีตารางขนาดเล็กหลายตาราง เมื่อเทียบกับปริมาณของหน่วยความจำจะเป็นปัจจัยพื้นฐานที่จำกัดวัฏจักรของซีพียู แต่กับตารางเล็กๆนั้น จะไม่มีปัญหาเรื่องความเร็ว

4. ช่วงความถี่ของหน่วยความจำ (Memory bandwidth) เมื่อซีพียูต้องการข้อมูลมากกว่าที่จะสามารถนำมาใส่ไว้ในแคชของซีพียู (CPU cache) ได้นั้นช่วงความถี่ของหน่วยความจำหลักจะกลายมาเป็นปัญหาคอขวด ปัญหานี้ไม่ใช่ปัญหาคอขวดของระบบทั่วไป แต่เป็นปัญหาหนึ่งที่ควรจะระวัง

3.3.2 ข้อจำกัดของการออกแบบใน MySQL

MySQL สามารถทำงานได้ทั้งกับตารางแบบแทรนแซคชั่น (transactional) และนันแทรนแซคชั่น (non-transactional) โดยการจะให้มันสามารถทำงานได้อย่างราบรื่นกับตารางแบบนันแทรนแซคชั่นซึ่งจะไม่สามารถทำการคืนค่า (roll back) เมื่อเกิดข้อผิดพลาดนั้น MySQL จะต้องทำตามกฎต่อไปนี้

- ทุกคอลัมน์ (column) ต้องมีค่าตัวเลือกอัตโนมัติ (default)

- ถ้ามีการใส่ค่าที่ผิดในคอลัมน์ (เช่น ใส่ตัวเลขที่มีค่ามากเกินไปลงในคอลัมน์ที่ใช้เก็บข้อมูลประเภทตัวเลข) MySQL จะทำการตั้งค่าในคอลัมน์ให้เป็นค่าที่ดีที่สุดที่เป็นไปได้ แทนที่จะฟ้องความผิดพลาด (error) โดยสำหรับค่าตัวเลขนั้น 0 คือค่าที่น้อยที่สุดหรือเป็นค่าที่มากที่สุดที่จะเป็นไปได้ สำหรับชุดอักษร (string) นั้นจะเป็นชุดอักษรว่าง (empty string) หรือชุดอักษรที่ยาวที่สุดที่คอลัมน์นั้นสามารถเก็บได้ก็ได้

- ทุกเอกซ์เพรสชั่น (expression) ที่ใช้ในการคำนวณ จะทำการคืนค่าที่สามารถใช้ได้แทนที่จะฟ้องออกมาว่าผิดพลาด ตัวอย่างเช่น 1/0 จะคืนค่าออกมาว่า NULL

ความหมายโดยนัยของกฎเหล่านี้ คือ ไม่ควรใช้ MySQL ในการเช็คข้อมูลที่อยู่ในคอลัมน์ แต่ควรจะทำการเช็คค่าภายในแอพพลิเคชั่น (application) ก่อนที่จะเก็บมันเข้าไปในฐานข้อมูล

3.3.3 การประมวลผลคำสั่ง SELECT และคำสั่งอื่นๆ

ก่อนอื่น สิ่งที่ควรรู้ คือ ปัจจัยหนึ่งที่จะมีผลกระทบต่อทุกสเตตเมนต์ (statement) คือ ยิ่งเพิ่มความยุ่งยากให้กับการติดตั้งการอนุญาต (permission) ก็จะยิ่งเพิ่มค่าใช้จ่าย (overhead) ขึ้นเท่านั้น

การใช้การอนุญาตแบบพื้นฐานนั้น เมื่อคุณส่งคำสั่ง GRANT จะเป็นการอนุญาตให้ MySQL ลด overhead ในการตรวจสอบการอนุญาต ในเวลาที่ลูกข่าย (client) ทำการดำเนินการ (execute) สเตตเมนต์ ตัวอย่างเช่น ถ้าคุณไม่อนุญาตให้มีสิทธิพิเศษสำหรับระดับตาราง (table-level) หรือระดับคอลัมน์ (column-level) ใดๆ แล้วเซิร์ฟเวอร์ (server) ก็ไม่จำเป็นต้องเช็คข้อมูลภายในของตาราง tables_priv และตาราง column_priv ในทำนองเดียวกัน ถ้าไม่มีการจำกัดทรัพยากร (resource) ของ account ใดๆ แล้วเซิร์ฟเวอร์ก็ไม่จำเป็นต้องทำการนับ ทรัพยากรซึ่งถ้ามีขนาดการ query ที่สูงมาก เราอาจใช้เวลาให้คุ้มค่าโดยการใช้โครงสร้างที่มีการอนุญาตแบบพื้นฐาน เพื่อลดค่าใช้จ่าย ในการตรวจสอบการอนุญาต

3.3.3.1 คำสั่ง EXPLAIN (ใช้ในการหาข้อมูลเกี่ยวกับคำสั่ง SELECT)

EXPLAIN tbl_name

หรือ

EXPLAIN SELECT select_options

EXPLAIN สามารถถูกใช้เป็นคำที่ความหมายเหมือนกับคำสั่ง DESCRIBE หรือเป็นวิธีการในการหาข้อมูลเกี่ยวกับว่า MySQL จะดำเนินการ (execute) SELECT อย่างไร

- syntax ของ EXPLAIN tbl_name จะเป็นคำที่มีความหมายเหมือนกับ DESCRIBE tbl_name หรือ SHOW COLUMNS FROM tbl_name
- เมื่อเราใช้ EXPLAIN ก่อน SELECT, MySQL จะอธิบายว่า มันจัดการกับ SELECT อย่างไร และให้ข้อมูลเกี่ยวกับว่ารวมตาราง (join) กันอย่างไร ในลำดับใด

ด้วยความช่วยเหลือของ EXPLAIN ช่วยให้ทราบว่าเมื่อไรที่จะต้องเพิ่มดัชนี (index) ให้กับตาราง เพื่อให้ SELECT ใช้ดัชนีในการหาเร็คคอร์ด (record) ได้เร็วขึ้น

ควรทำ ANALYZE TABLE บ่อยๆ เพื่ออัพเดต (update) สถิติของตาราง เช่น cardinality ของคีย์ (key) ซึ่งจะส่งผลถึงตัวเลือกที่ตัวประมวลผลจะเลือก

คุณยังจะทราบว่าตัวประมวลผลนั้น รวมตารางในลำดับที่เหมาะสมหรือไม่ การบังคับให้ตัวประมวลผลใช้ลำดับในการรวมตารางตามลำดับที่ตารางถูกตั้งชื่อใน SELECT นั้น เริ่มต้นด้วย SELECT STRAIGHT_JOIN แทนที่เริ่มต้นด้วย SELECT ธรรมดา

EXPLAIN จะส่งแถวของข้อมูลสำหรับแต่ละตารางที่ใช้ใน SELECT กลับมา ตารางนั้นจะถูกลิสต์ (list) ตามเอาท์พุต (output) ตามลำดับที่ MySQL อ่านในขณะที่จัดการ query โดย MySQL ตั้งใจให้ทุกๆ การการรวมตารางนั้น ใช้กระบวนการ single-sweep multi-join ซึ่งหมายถึงว่า MySQL จะอ่าน 1 แถว จากตารางที่ 1 จากนั้นจะหาแถวที่เข้าคู่กันในตารางที่ 2, ตารางที่ 3 และเรื่อยๆ ไป แล้วเมื่อทุกๆ ตารางได้รับการจัดการเรียบร้อยแล้ว มันจะส่งเอาท์พุตออกมาเป็นคอลัมน์ที่ได้เลือกไว้แล้ว และย้อนกลับผ่านลิสต์ของตารางจนกระทั่งพบตารางที่มีแถวที่เหมาะสมอีก แล้วแถวถัดไปจะถูกอ่านจากตารางนี้ และกระบวนการก็จะดำเนินต่อไปกับตารางถัดไป

แต่ละแถวที่เป็นเอาท์พุตจากคำสั่ง EXPLAIN จะให้ข้อมูลเกี่ยวกับตาราง 1 ตาราง และแต่ละแถวจะประกอบด้วยคอลัมน์ต่อไปนี้:

id ตัวระบุถึง SELECT, เป็นเลขตามลำดับของ SELECT ภายใต้ query

select_type

ชนิดของ SELECT, ซึ่งอาจเป็นได้ดังต่อไปนี้:

SIMPLE	Simple SELECT (ไม่ใช้ UNION หรือ subquery)
PRIMARY	SELECT ตัวนอกสุด
UNION	SELECT ตัวที่ 2 หรือหลังจากนั้นใน UNION
DEPENDENT UNION	SELECT ตัวที่ 2 หรือหลังจากนั้นใน UNION, แต่ขึ้นอยู่กับ subquery ตัวนอก
SUBQUERY	SELECT ตัวแรกใน subquery
DEPENDENT SUBQUERY	SELECT ตัวแรกใน subquery แต่ขึ้นอยู่กับ subquery ตัวนอก
DERIVED	เกิดจากตาราง SELECT (subquery ในประโยค FROM)

table ตารางที่แถวของเอาท์พุตอ้างถึง

type ชนิดของการรวมตาราง โดยชนิดการรวมที่ต่างกันจะถูกลิสต์ไว้ที่นี้ ตามลำดับจากชนิดที่ดีที่สุดไปแย่ที่สุด:

system	ตารางมีเพียงแถวเดียว (= ตารางของระบบ) โดยเป็นกรณีพิเศษของการรวมตารางชนิด const

const	ตารางจะมีอย่างมากเพียงหนึ่งแถวเท่านั้นที่เข้ากัน (1 matching row) ซึ่งจะถูกอ่านตอนเริ่มต้นของ query และเนื่องจากมีเพียงแถวเดียว ค่าจากคอลัมน์ในแถวนี้จึงถือว่าคงที่สำหรับตัวประมวลผลอื่นๆ โดยตาราง const จะเร็วมาก เพราะมันจะถูกอ่านเพียงครั้งเดียวเท่านั้น const จะถูกใช้เมื่อเปรียบเทียบทุกๆ ส่วนของ PRIMARY หรือดัชนีที่ไม่ซ้ำ (UNIQUE index) กับค่าคงที่
eq_ref	แถว 1 แถวจะถูกอ่านจากตารางนี้สำหรับแต่ละการรวมกันของแถวจากตารางก่อนหน้า นอกเหนือจากชนิด const วิธีนี้เป็นชนิดการรวมตารางที่ดีที่สุด โดยจะถูกใช้เมื่อทุกส่วนของดัชนีถูกใช้โดยการรวม และดัชนีนั้นเป็น PRIMARY KEY หรือดัชนีที่ไม่ซ้ำ eq_ref สามารถใช้เป็นดัชนีคอลัมน์ที่ถูกเปรียบเทียบกับการใช้เครื่องหมาย "=" โดยค่าที่ใช้เปรียบเทียบอาจเป็นค่าคงที่ หรือเอกซ์เพรสชั่นที่ใช้คอลัมน์จากตารางที่ถูกอ่านก่อนตารางนี้
ref	ทุกแถวที่มีค่าดัชนีที่เข้ากัน (matching index) จะถูกอ่านจากตารางนี้สำหรับแต่ละการรวมกันของแถวจากตารางก่อนหน้า โดย ref จะถูกใช้เวลาที่การรวมนั้น ใช้เฉพาะ leftmost prefix ของคีย์ หรือเวลาที่คีย์นั้นไม่ใช่ PRIMARY KEY หรือคีย์ที่ไม่ซ้ำ (หรือให้เข้าใจง่าย คือ ถ้าการรวมตารางนั้นไม่สามารถเลือกแถวเดี่ยว (single row) ที่ยึดถือค่าคีย์) ถ้าคีย์นั้นใช้จับคู่เพียงไม่กี่แถว วิธีนี้จะเป็นวิธีที่ดี ref สามารถใช้เป็นดัชนีคอลัมน์ที่ถูกเปรียบเทียบกับการใช้เครื่องหมาย "="
ref_or_null	การรวมตารางแบบนี้จะเหมือนกับ ref แต่มีส่วนเพิ่มเติม คือ MySQL จะทำการค้นหาเพิ่มเติม (extra search) สำหรับแถวที่มีค่า NULL
Index_merge	การรวมตารางแบบนี้แสดงให้เห็นว่าใช้ Index Merge Optimization ซึ่งในกรณีนี้คอลัมน์คีย์จะมีลิสต์การใช้ของดัชนี และ key_len จะมีลิสต์ของส่วนของคีย์ที่ยาวที่สุดสำหรับการใช้ดัชนี
Unique_subquery	แบบนี้จะใช้แทนที่ ref สำหรับบาง IN subquery ของฟอร์มต่อไปนี้: Value IN (SELECT primary_key FROM single_table WHERE some_expr) unique_subquery นั้น เป็นเพียงฟังก์ชั่นในการค้นหาดัชนีที่แทนที่ subquery อย่างสมบูรณ์เพื่อประสิทธิภาพที่ดีกว่า
index_subquery	การรวมตารางแบบนี้จะคล้ายกับแบบ unique_subquery โดยใช้แทนที่ IN subquery แต่จะใช้สำหรับดัชนีที่มีซ้ำกัน (non-unique index) ใน subquery ของฟอร์มต่อไปนี้: Value IN (SELECT key_column FROM single_table WHERE some_expr)

range มีเพียงแถวที่อยู่ภายในขอบเขตที่เลือกไว้เท่านั้นที่จะถูกเรียกมาใช้งาน โดยใช้ดัชนีในการเลือกแถวในกรณีนี้คอลัมน์คีย์จะเป็นตัวบ่งชี้ว่าดัชนีใดถูกใช้
key_len จะมีส่วนของคีย์ที่ยาวที่สุดที่ถูกใช้ และคอลัมน์ ref จะเป็น NULL สำหรับการรวมตารางแบบนี้
range สามารถถูกใช้เวลาที่คอมลัมน์คีย์นั้น ทำการเปรียบเทียบกับค่าคงที่ด้วยเครื่องหมาย =, <>, >, >=, <, <=, IS NULL, <=>, BETWEEN, หรือ IN:

Index การรวมตารางแบบนี้จะเหมือนกับ ALL ยกเว้นที่จะมีเพียง tree ของดัชนีเท่านั้นที่จะถูกสแกน (scan) ซึ่งโดยปกติแล้ว จะเร็วกว่า ALL เพราะไฟล์ของดัชนี (index file) มักจะมีขนาดเล็กกว่าไฟล์ข้อมูล (data file)
MySQL สามารถใช้การรวมตารางแบบนี้ ในเวลาที่ query ใช้เพียงคอลัมน์ที่เป็นส่วนหนึ่งของดัชนี (single index)

ALL จะทำการสแกนทั้งตารางสำหรับแต่ละการรวมกันของแถวจากตารางก่อนหน้า ซึ่งโดยปกติแล้ว มันจะไม่ดีถ้าตารางนั้นเป็นตารางแรกที่ไม่ได้ระบุว่าเป็น const และจะแย่มากๆ ในกรณีอื่น แต่เราสามารถหลีกเลี่ยง ALL โดยการเพิ่มดัชนีที่อนุญาตให้นำแถวมาเก็บไว้จากตารางที่ยึดถือค่าคงที่ หรือค่าของคอลัมน์จากตารางก่อนหน้า

possible_keys

คอลัมน์ possible_keys ใช้แสดงว่าดัชนีใดที่ MySQL ใช้ในการหาแถวของตารางนี้ ข้อสังเกต คือ คอลัมน์นี้จะเป็นอิสระจากลำดับของตารางที่แสดงในเอาท์พุตของ EXPLAIN ซึ่งหมายความว่าบางคีย์ใน possible_keys ที่อาจไม่สามารถใช้ได้ในการใช้งานด้วยลำดับของตารางที่ถูกสร้างขึ้น
ถ้าคอลัมน์นี้เป็น NULL คือไม่มีดัชนีที่สำคัญ (relevant index) ซึ่งในกรณีนี้ อาจสามารถพัฒนาประสิทธิภาพของ query โดยการทดลองใส่ WHERE เพื่อดูว่ามันอ้างถึงคอลัมน์ใด (หรือคอลัมน์ที่จะเหมาะสำหรับการทำดัชนี) ซึ่งถ้าเป็นอย่างนั้น ให้สร้างดัชนีที่เหมาะสม (appropriate index) ขึ้นมา 1 ตัว และตรวจสอบ query ด้วยคำสั่ง EXPLAIN อีกครั้ง
เพื่อดูว่าตารางมีดัชนีอะไร ให้ใช้ SHOW INDEX FROM tbl_name

key คอลัมน์คีย์ ใช้แสดงคีย์ (ดัชนี) ที่ MySQL ตัดสินใจจะใช้จริงๆ ถ้าคีย์เป็น NULL คือ ไม่มีดัชนีใดถูกเลือก ในการบังคับให้ MySQL ใช้หรือไม่ใช้ดัชนีที่ถูกลิสต์ในคอลัมน์ possible_key ให้ใช้คำสั่ง FORCE iNDEX, USE INDEX หรือ IGNORE INDEX ใน query
ในตาราง MyISAM และ BDB การใช้คำสั่ง ANALYZE TABLE จะช่วยให้ตัวประมวลผลเลือก ดัชนีที่ดีกว่า สำหรับตาราง MyISAM คำสั่ง run myisamchk –analyze จะทำเช่นเดียวกัน

key_len

คอลัมน์ key_len จะแสดงความยาวของคีย์ ที่ MySQL ตัดสินใจใช้ โดยถ้าคอลัมน์คีย์เป็น NULL แล้ว คอลัมน์ key_len ก็จะเป็น NULL ด้วย มีข้อสังเกต คือ ค่าของ key_len จะช่วยให้เราสามารถตัดสินใจว่าจะมี multiple-part key ที่ MySQL จะใช้จริงๆ กี่ส่วน

ref คอลัมน์ ref ใช้แสดงว่าคอลัมน์หรือค่าคงที่ใดที่ถูกใช้ร่วมกับคีย์ในการเลือกแถวจากตาราง

rows คอลัมน์ rows ใช้แสดงจำนวนของแถวที่ MySQL เชื่อว่าต้องตรวจดูเพื่อจะดำเนินการ query

Extra คอลัมน์นี้จะเก็บข้อมูลเพิ่มเติมเกี่ยวกับว่า MySQL จะทำอย่างไรกับ query โดยชุดอักษร (text string) ที่อาจปรากฏในคอลัมน์จะเป็นได้ดังต่อไปนี้:

Distinct MySQL จะหยุดค้นหาแถวเพิ่มเติมจากการรวมแถวปัจจุบัน หลังจากพบแถวที่เข้ากันเป็นแถวแรก

Not exists MySQL สามารถทำการประมวลผลคำสั่ง LEFT JOIN บน query และจะไม่ตรวจดูแถวจากตารางนี้เพิ่มจากการรวมกันของแถวก่อนหน้า หลังจากมันพบแถว หนึ่งที่เข้ากันกับกับกฎของ LEFT JOIN

Range checked for each record (index map: #)

MySQL พบว่าไม่มีดัชนีดีๆ ที่ควรใช้ ดังนั้นแทนที่การรวมแถวในตารางก่อนๆ มันจะทำการตรวจสอบเพื่อตัดสินใจว่าจะใช้ดัชนีใด (ถ้ามี) และใช้ดัชนีนั้นในการนำแถวจากตารางมาเก็บไว้ การทำวิธีนี้ไม่เร็วนัก แต่ก็ยังเร็วกว่าการรวมตารางโดยไม่ใช้ดัชนีเลย

Using filesort MySQL ต้องทำส่งผ่านแบบพิเศษ (extra pass) เพื่อค้นหาว่าจะนำแถวมาเรียงลำดับอย่างไร โดยการเรียงลำดับจะทำได้ด้วยการตรวจดูทุกๆ แถวตามชนิดของการรวมตาราง และเก็บคีย์ที่ใช้ในการเรียงลำดับ (sort key) และพ้อยเตอร์ (pointer) ที่ชี้ไปที่แถวของทุกๆ แถวที่เข้ากันกับประโยค WHERE แล้วคีย์ก็จะถูกเรียงลำดับ และแถวก็จะเรียงลำดับด้วย

Using index คอลัมน์ข้อมูลจะถูกนำมาจากตารางโดยใช้เพียงข้อมูลใน tree ของดัชนี โดยไม่ต้องทำการค้นหาเพิ่มเติมเพื่ออ่านแถวที่ถูกใช้อยู่ (actual row) วิธีการนี้สามารถถูกใช้เวลา query ใช้เพียงคอลัมน์ที่เป็นส่วนหนึ่งของดัชนีเดี่ยว (single index)

Using temporary ในการทำ query นั้น MySQL จะต้องสร้างตารางชั่วคราวเพื่อใช้เก็บผลลัพธ์ แบบอย่างนี้จะเกิดขึ้นเมื่อ query มี GROUP BY และ ORDER BY ที่คอลัมน์ลิสต์แตกต่างกัน

Using where ประโยคของ WHERE จะถูกใช้เพื่อจำกัดว่าแถวใดที่เข้ากันกับตารางถัดไป หรือส่งให้กับลูกข่าย ถ้าเราไม่ชี้ชัดว่าตั้งใจนำหรือตรวจดูทุกแถวของตาราง อาจเกิดความ

ผิดพลาดขึ้นใน query ถ้าค่าของ Extra ไม่ใช่ Using WHERE และการรวมตารางเป็นแบบ ALL หรือ INDEX

ถ้าต้องการสร้างให้ query มีความเร็วมากที่สุดที่จะเป็นไปได้ เราควรระวังค่าของ Extra ที่เป็น Using filesort และ Using temporary

เราสามารถบอกได้ว่าการรวมตารางนั้นดีอย่างไร โดยการนำค่าของผลิตภัณฑ์ (product) ในคอลัมน์ rows ของเอาท์พุตจาก EXPLAIN ซึ่งควรบอกอย่างคร่าวๆ ได้ว่ามีกี่แถวที่ MySQL ต้องตรวจดูเพื่อดำเนินการ query และถ้าเราจำกัด query ด้วย max_join_size แล้วผลิตภัณฑ์นี้จะถูกใช้ในการตัดสินใจว่า multiple-table ใดที่ SELECT จะดำเนินการ

3.3.3.2 การประมาณความเร็วของการทำ query

ในกรณีส่วนใหญ่ เราสามารถประมาณประสิทธิภาพ (performance) ด้วยการนับจำนวนครั้งที่ใช้ในการค้นหาดิสก์ โดยสำหรับตารางเล็กๆ เรามักจะพบแถวหนึ่งในการค้นหาดิสก์หนึ่งๆ (เพราะบางทีดัชนี (index) อาจถูกแคช) และสำหรับตารางใหญ่ๆ เราสามารถประมาณโดยใช้ดัชนีแบบ B-tree ซึ่งเราต้องทำการค้นหาหลายๆ ครั้งเพื่อหาแถว: log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1

ใน MySQL, บล็อกของดัชนี (index block) มักจะมีขนาด 1024 ไบต์ และดัชนีข้อมูล (data pointer) มักจะมีขนาด 4 ไบต์ สำหรับตาราง 500,000 แถว ที่มีความยาวดัชนีเท่ากับ 3 ไบต์ (เลขจำนวนเต็ม(integer) ขนาดกลาง) จากสูตรแสดงให้เห็นว่า log(500,000) / log(1024 / 3 * 2 / (3+4)) + 1 = 4 seeks

ดัชนีนี้ต้องการความจุประมาณ 500,000 * 7 * 3 / 2 = 5.2 MB (สมมุติ typical index buffer fill ration เป็น 2/3) ดังนั้นอาจต้องมีดัชนีจำนวนมากในหน่วยความจำ และต้องการเพียง 1 หรือ 2 ครั้งในการเรียก (call) ในการอ่านข้อมูลเพื่อหาแถว

ในการเขียน อย่างไรก็ตาม เราต้องการ 4 คำร้องเพื่อค้นหา (seek request) เพื่อหาว่าจะนำดัชนีใหม่ไปไว้ที่ใด และโดยปกติแล้วใช้ 2 ในการค้นหาเพื่ออัพเดตดัชนี และเขียนแถว

ข้อสังเกต คือ จากที่อธิบายไว้ข้างต้นไม่ได้หมายความว่า ประสิทธิภาพของแทพพลิเคชั่นจะแย่ลงอย่างช้าๆ จาก log N! แต่ตราบใดที่ทุกสิ่งถูกแคชโดย OS หรือ SQL server แล้ว สิ่งต่างๆ จะช้าขึ้นเมื่อตารางมีขนาดใหญ่ขึ้น และหลังจากข้อมูลมีขนาดใหญ่เกินไปที่จะถูกแคชแล้ว สิ่งต่างๆ จะเริ่มช้ามากขึ้นๆ จนกระทั่งแอพพลิเคชั่นจะถูกผูกมัดด้วยการค้นหาดิสก์ (ที่เพิ่มโดย log N!) ซึ่งสามารถหลีกเลี่ยงโดยการเพิ่มขนาดแคชของคีย์ (key cache) ตามข้อมูลที่เพิ่มขึ้น ในตาราง MyISAM, ขนาดแคชของคีย์จะถูกควบคุมด้วยตัวแปรของระบบ key_buffer_size

3.3.3.3 ความเร็วในการทำ SELECT

โดยทั่วๆ ไป เมื่อต้องการสร้างให้คำถาม SELECT.....WHERE เร็วขึ้น สิ่งแรกที่ต้องทำ คือ ตรวจสอบว่าเราสามารถเพิ่มดัชนีเข้าไปได้หรือไม่ เพราะทุกๆ การอ้างถึงระหว่างตารางที่ต่างกันควรจะถูกทำด้วย ดัชนี โดยเราสามารถใช้คำสั่ง EXPLAIN ในการตัดสินใจว่าดัชนีใดที่ควรใช้สำหรับทำ SELECT

คำแนะนำทั่วๆ ไปในการเพิ่มความเร็ว query บนตาราง MyISAM:

- เพื่อช่วยให้ MySQL ทำการประมวล query ได้ดีขึ้น ให้ใช้คำสั่ง ANALYZE TABLE หรือ run myisamchk--analyze บนตารางหลังจากที่โหลด (load) ข้อมูลเรียบร้อยแล้ว ซึ่งวิธีการนี้จะอัพเดตค่าแต่ละส่วนของดัชนีที่แสดงจำนวนเฉลี่ยของแถวที่มีค่าเหมือนกัน (สำหรับดัชนีที่ไม่ซ้ำ (unique index) ค่านี้จะเป็น 1) MySQL จะใช้วิธีนี้ในการเลือกดัชนี เวลาที่รวม (join) 2 ตารางเข้าด้วยกันโดยยึดถือ non-constant expression โดยสามารถตรวจสอบผลลัพธ์จากการวิเคราะห์ตารางด้วยคำสั่ง SHOW INDEX FROM tbl_name และทดสอบค่า Cardinality และสามารถให้มันแสดงข้อมูลการกระจายของดัชนีด้วยคำสั่ง myisamxhk--escription--verbose
- ในการเรียงลำดับ (sort) ดัชนี และข้อมูลตามดัชนี เราจะใช้คำสั่ง myisamchk--sort-index--sort-records=1 (ถ้าต้องการให้เรียงลำดับตามดัชนีที่ 1) วิธีนี้เป็นวิธีที่ดีในการทำให้ query เร็วขึ้นเมื่อมีดัชนีที่ไม่ซ้ำในเร็คคอร์ด (record) ทั้งหมดที่ต้องการอ่านซึ่งเรียงตามดัชนี ข้อสังเกต คือ ครั้งแรกที่ทำการเรียงลำดับในตารางที่มีขนาดใหญ่ด้วยวิธีนี้ อาจจะใช้เวลาค่อนข้างนาน

3.3.3.4 MySQL ประมวลผลคำสั่ง WHERE อย่างไร

ในส่วนนี้จะอธิบายถึงการประมวลผลที่สร้างขึ้นเพื่อการจัดการกับ WHERE โดยในตัวอย่างจะใช้ SELECT แต่การประมวลผลแบบนี้สามารถใช้ WHERE กับคำสั่ง DELETE และ UPDATE ก็ได้

ข้อสังเกต ในการทำงานบนตัวประมวลผล MySQL จะเป็นแบบต่อเนื่อง ดังนั้นในส่วนนี้จึงยังไม่สมบูรณ์ เนื่องจาก MySQL จะทำการประมวลผลหลายครั้งมาก จึงแสดงให้ดูได้เพียงบางส่วนเท่านั้น

การประมวลผลบางอย่างที่ทำโดย MySQL แสดงได้ต่อไปนี้:

- เอาเครื่องหมายวงเล็บที่ไม่จำเป็นบางส่วนออกไป:

  ((a AND b) AND c OR (((a AND b) AND (c AND d))))

  -> (a AND b AND c) OR (a AND b AND c AND d)
- แทนค่าคงที่:

  (a<b AND b=c) AND a=5

  -> b>5 AND b=c AND a=5
- เอาเงื่อนไขของค่าคงที่ออกไป (จำเป็นเพราะต้องแทนค่าคงที่):

  (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)

  -> B=5 OR B=6
- expression มี่เป็นค่าคงที่ซึ่งถูกใช้โดยดัชนีจะถูกประเมินผลเพียงครั้งเดียว

- COUNT(*) ของตารางเดี่ยว โดยไม่มี WHERE จะถูกนำมาโดยตรงจากตารางข้อมูลสำหรับ MyISAM และ HEAP ซึ่งวิธีเดียวกันนี้จะใช้สำหรับเอกซ์เพรสชั่นใดๆ ที่ NOT NULL เมื่อใช้กับตารางเพียงตารางเดียว
- มีการตรวจสอบเบื้องต้นของเอกซ์เพรสชั่นคงที่ (constant expression) ที่ผิด โดย MySQL จะพบว่า SELECT นั้นเป็นไปไม่ได้ และไม่ส่งแถวใดๆ คืนมา
- HAVING จะถูกรวมเข้ากับ WHERE ถ้าไม่ใช้ GROUP BY หรือฟังก์ชั่นกรุ๊ป (group function) ใดๆ เช่นCOUNT(), MIN() เป็นต้น
- ในการรวมแต่ละตาราง WHERE อย่างง่ายจะถูกสร้างขึ้นเพื่อให้การประเมินผล WHERE ของตารางเร็วขึ้น และจะข้ามเร็คคอร์ดทันทีที่เป็นไปได้
- ทุกๆ ตารางคงที่ (constant table) จะถูกอ่านเป็นอันดับแรกก่อนตารางอื่นๆ ใน query โดยตารางคงที่จะเป็นดังต่อไปนี้:
    - ตารางเปล่า หรือตารางที่มีเพียงแถวเดียว
    - ตารางที่ใช้กับ WHERE บน PRIMARY KEY หรือดัชนีไม่ซ้ำ (UNIQUE index) ซึ่งทุกส่วนของดัชนีจะถูกเปรียบเทียบกับเอกซ์เพรสชั่นที่เป็นค่าคงที่ และถูกกำหนดว่า NOT NULL
- หลังจากได้ลองรวมตารางในทุกทางที่เป็นไปได้แล้ว พบว่า การรวมกันที่ดีที่สุดสำหรับการรวมตาราง คือ ถ้าทุกคอลัมน์ใน ORDER BY และ GROUP BY มาจากตารางเดียวกันแล้ว ตารางนั้นจะถูก รวมก่อน
- ถ้ามี ORDER BY และ GROUP BY ที่ต่างกัน หรือถ้า ORDER BY หรือ GROUP BY ที่ประกอบด้วยคอลัมน์จากตารางที่ไม่ใช่ตารางแรกในคิวของการรวมตารางแล้ว จะมีการสร้างตารางชั่วคราวขึ้น
- ถ้าใช้คำสั่ง SQL_SMALL_RESULT, MySQL จะใช้ in-memory ของตารางชั่วคราว
- แต่ละดัชนีของตารางจะถูก query และจะใช้ดัชนีที่ดีที่สุด นอกเสียจากตัวประมวลผลจะเชื่อว่าการสแกนตารางจะมีประสิทธิภาพมากกว่า โดยในอดีตจะใช้การสแกนที่ยึดหลักการว่าดัชนีดีที่สุดจะกว้างมากกว่า 30% ของตาราง แต่ในปัจจุบันนี้ ตัวประมวลผลจะมีความซับซ้อนมากขึ้น และยึดหลักการประมาณด้วยปัจจัยเพิ่มเติมต่างๆ เช่น ขนาดตาราง, จำนวนแถว และขนาดบล็อก I/O ดังนั้นเปอร์เซนต์ที่กำหนดไว้จึงไม่ได้ใช้ในการตัดสินใจว่าจะเลือกการใช้ดัชนีหรือการสแกนอีกแล้ว
- ในบางกรณี MySQL สามารถอ่านแถวจากดัชนี โดยที่ไม่ต้องติดต่อกับไฟล์ข้อมูล (data file) เลย เช่น ถ้าทุกคอลัมน์ที่ถูกใช้จากดัชนีเป็นตัวเลข จะใช้เพียงแค่ tree ของดัชนี (index tree) ในการทำ query
- ก่อนที่แต่ละเร็คคอร์ดจะเป็นเอาท์พุต ในส่วนที่ไม่เข้ากันกับ HAVING จะถูกข้ามไป

3.3.3.5 MySQL ประมวลผลคำสั่ง OR อย่างไร

กระบวนการรวมดัชนีเข้าด้วยกัน (Index Merge) จะถูกใช้ในการนำแถวออกมาด้วยวิธีการสแกนแบบ ref, ref_or_null หรือ range แล้วรวมผลลัพธ์ที่ได้เป็นหนึ่งเดียว กระบวนการนี้จะถูกใช้งานเมื่อเงื่อนไขของตารางเป็นเงื่อนไขที่แยกออกมาจากเงื่อนไขสำหรับ ref, ref_or_null หรือ range ซึ่งสามารถถูกใช้กับคีย์ที่ต่างกัน

ในเอาท์พุตของคำสั่ง EXPLAIN กระบวนการนี้จะปรากฏเป็น index_merge ในคอลัมน์ชนิด (type) ในกรณีนี้คอลัมน์คีย์จะเก็บลิสต์ของดัชนีที่ถูกใช้ไปแล้ว และ key_len จะเก็บลิสต์ของส่วนของคีย์ที่ยาวที่สุดสำหรับดัชนีเหล่านั้น

3.3.3.6 MySQL ประมวลผลคำสั่ง IS NULL อย่างไร

MySQL จะทำการประมวลผลของ col_name IS NULL ด้วยวิธีเดียวกับที่ทำของ col_name = constant_value ตัวอย่างต่อไปนี้แสดงว่า MySQL สามารถใช้ดัชนีและขอบเขต (range) ในการหา NULL ด้วย IS NULL

ถ้า WHERE รวมเงื่อนไข col_name IS NULL สำหรับคอลัมน์ที่ประกาศว่าเป็น NOT NULL แล้วเอกซ์เพรสชั่นนั้นจะไม่ถูกประมวลผล โดยกรณีที่จะไม่เกิดการประมวลผลแบบนี้ คือ เมื่อคอลัมน์นั้นอาจสร้าง NULL ออกมา เช่น ถ้ามาจากตารางฝั่งขวาของ LEFT JOIN

ref_or_null ทำงานโดยเริ่มแรกจะอ่านคีย์อ้างอิง (reference key) จากนั้นจะแยกกันหาแถวที่มีค่าคีย์เป็น NULL

ข้อสังเกต คือ การประมวลผลสามารถรับมือกับ IS NULL ได้เพียงระดับเดียว

3.3.3.7 MySQL ประมวลผลคำสั่ง DISTINCT อย่างไร

การใช้ DISTINCT ร่วมกับ ORDER BY จะต้องการตารางชั่วคราวในหลายๆ กรณี

ข้อสังเกต คือ เนื่องจาก DISTINCT อาจใช้ GROUP BY จึงควรระวังว่า MySQL ทำงานกับ คอลัมน์ใน ORDER BY หรือ HAVING ที่ไม่ใช่ส่วนหนึ่งของคอลัมน์ที่เลือกไว้อย่างไร

เวลาใช้ LIMIT row_count ร่วมกับ DISTINCT, MySQL จะหยุดสแกนตารางที่ไม่ได้ใช้ทันทีที่มันพบตัวที่เข้ากันเป็นตัวแรก

3.3.3.8 MySQL ประมวลผลคำสั่ง LEFT JOIN และ RIGHT JOIN อย่างไร

A LEFT JOIN B join_condition ถูกนำไปใช้ใน MySQL ดังนี้:

- ตาราง B ถูกตั้งให้ขึ้นอยู่กับตาราง A และทุกๆ ตารางที่ A ขึ้นตรงต่อ
- ตาราง A ถูกตั้งให้ขึ้นอยู่กับทุกตาราง (ยกเว้น B) ที่ถูกใช้ในเงื่อนไขของ LEFT JOIN
- เงื่อนไขของ LEFT JOIN จะใช้ในการตัดสินใจว่าจะเอาแถวจากตาราง B อย่างไร (หรือกล่าวอีกอย่างว่าเงื่อนไขใดๆ ใน WHERE จะไม่ถูกใช้)

- ทุกๆ การรวมตารางตามมาตรฐานของการประมวลผลจะถูกทำก่อน โดยมีข้อยกเว้น คือ ตารางจะต้องถูกอ่านหลังจากทุกตารางที่มันขึ้นตรงต่อเสมอ แต่ถ้ามีการขึ้นตรงต่อกันเป็นวงกลม MySQL จะฟ้องว่าผิดพลาด
- ทุกๆ WHERE มาตรฐานของตัวประมวลผลจะถูกทำ
- ถ้ามีแถวใน A ที่เข้ากันกับ WHERE แต่ไม่มีแถวใน B ที่เข้ากันกับเงื่อนไข ON แล้ว จะมีการสร้างแถวพิเศษใน B (extra B row) โดยทุกๆ คอลัมน์จะเป็น NULL
- ถ้าใช้ LEFT JOIN ในการค้นหาแถวที่ไม่ปรากฏในบางตาราง และมีการทดสอบต่อไปนี้: col_name IS NULL ในส่วนของ WHERE โดย col_name เป็นคอลัมน์ที่ประกาศไว้ว่า NOT NULL แล้ว MySQL จะหยุดหาแถวเพิ่ม (สำหรับการรวมคีย์เฉพาะ) หลังจากมันพบแถวหนึ่งที่เข้ากันกับเงื่อนไขของ LEFT JOIN

  RIGHT JOIN จะถูกนำไปใช้คล้ายๆ กับ LEFT JOIN แต่บทบาทของตารางจะย้อนกลับ

ตัวประมวลผลในการรวมตารางจะคำนวณลำดับของตารางที่ควรถูกรวม โดยลำดับการอ่านตารางที่ถูกบังคับด้วย LEFT JOIN และ STRAIGTH_JOIN จะเป็นตัวช่วยให้ตัวประมวลผลในการรวมตารางสามารถทำงานได้เร็วขึ้นมากๆ เพราะจะมีตารางถูกเปลี่ยนลำดับให้ตรวจสอบน้อยกว่า

MySQL จะทำการประมวลผล LEFT JOIN ดังต่อไปนี้: ถ้าเงื่อนไขของ WHERE สำหรับการสร้างแถว NULL เป็น false เสมอแล้ว LEFT JOIN จะถูกเปลี่ยนเป็นการรวมตารางธรรมดา

3.3.3.9 MySQL ประมวลผลคำสั่ง ORDER BY อย่างไร

ในบางกรณี MySQL สามารถใช้ดัชนีในการตอบคำถาม ORDER BY หรือ GROUP BY โดยที่ไม่ต้องทำการเรียงลำดับข้อมูลเพิ่มเลย

สามารถใช้ดัชนีได้แม้ในกรณีที่ดัชนีนั้นไม่ตรงกันกับใน ORDER BY ซะทีเดียว ตราบใดที่ทุกส่วนของดัชนีที่ไม่ได้ใช้ และทุกส่วนพิเศษ (extra) เป็นคอลัมน์ที่ใช้กับ ORDER BY เป็นค่าคงที่ใน WHERE

ในบางกรณี MySQL จะไม่สามารถใช้ดัชนีในการตอบปัญหา ORDER BY ถึงแม้มันจะยังใช้ดัชนีในการค้นหาแถวที่เข้ากันกับ WHERE ซึ่งกรณีเช่นนั้น ได้แก่ :

- ใช้ ORDER BY บนคีย์ที่ต่างกัน:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- ใช้ ORDER BY บนคีย์ที่ไม่เปลี่ยนค่า:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- ใช้ ASC และ DESC ปนกัน:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- คีย์ที่ใช้ในการนำแถวมาไม่ใช่อันเดียวกับที่ใช้ใน ORDER BY:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- มีการรวมหลายๆ ตาราง และคอลัมน์ใน ORDER BY ไม่ใช่ทั้งหมดจากตารางที่ไม่ใช่ค่าคงที่ตารางแรกที่ถูกใช้ในการนำแถวขึ้นมา
- มีเอกซ์เพรสชั่นที่เป็น ORDER BY และ GROUP BY ที่ต่างกัน
- ชนิดของดัชนีของตารางไม่ได้เก็บแถวตามลำดับ เช่น ดัชนีแฮช (HASH index) ในตาราง HEAP

ซึ่งกรณีเหล่านั้น ทำให้ MySQL ต้องเรียงลำดับผลลัพธ์ โดยมีขั้นตอนดังต่อไปนี้:

1. อ่านทุกแถวตามคีย์ หรือใช้การสแกนตาราง โดยแถวที่ไม่เข้ากันกับ WHERE จะถูกข้ามไป
2. เก็บค่าคีย์ที่ใช้ในการเรียงลำดับ (sort key) ในบัฟเฟอร์ (buffer)
3. เมื่อบัฟเฟอร์เริ่มเต็ม บัฟเฟอร์จะดำเนินการเรียงลำดับอย่างเร็ว (qsort) และเก็บผลลัพธ์ที่ได้ในไฟล์ชั่วคราว จากนั้นจะเก็บพ้อยเตอร์ (pointer) ที่ชี้ไปยังบล็อกที่เรียงลำดับแล้ว (ถ้าทุกแถวเก็บในบัฟเฟอร์ได้พอดี จะไม่ต้องสร้างไฟล์ชั่วคราว)
4. ทำซ้ำขั้นตอนที่ผ่านมาจนกว่าทุกแถวจะถูกอ่าน
5. ทำ multi-merge ให้มากกว่าขอบเขตของ MERGEBUFF(7) กับบล็อกหนึ่งๆ ในไฟล์ชั่วคราวอีกไฟล์ และทำซ้ำเรื่อยๆ จนกว่าทุกบล็อกจากไฟล์แรกจะอยู่ในไฟล์ที่สอง
6. ทำตามขั้นตอนที่ผ่านมาจนกระทั่งมีบล็อกที่เหลือน้อยกว่า MERGEBUFF2 (15)
7. ใน multi-merge ครั้งสุดท้าย จะมีเพียงพ้อยเตอร์ที่ชี้ไปที่แถว (ส่วนสุดท้ายของคีย์เรียงลำดับ) เท่านั้นที่ถูกเขียนในไฟล์ผลลัพธ์ (result file)
8. อ่านแถวตามลำดับที่เรียงไว้ โดยใช้พ้อยเตอร์ที่ชี้ไปที่แถวในไฟล์ผลลัพธ์ ในการประมวลผลนั้น ทำโดยการอ่านบล็อกขนาดใหญ่ของพ้อยเตอร์ที่ชี้ไปที่แถว, เรียงลำดับมัน, และใช้มันในการอ่านแถวในลำดับที่เรียงไว้ลงในบัฟเฟอร์ของแถว (row buffer) โดยขนาดของบัฟเฟอร์จะเป็นค่าของตัวแปรของระบบ(read_rnd_buffer_size)

ถ้าต้องการเพิ่มความเร็วของ ORDER BY, อันดับแรกต้องดูว่าจะสามารถให้ MySQL ใช้ดัชนีแทนที่จะใช้การเรียงลำดับเพิ่มเติม แต่ถ้าเป็นไปไม่ได้ อาจลองใช้วิธีการต่อไปนี้:

- เพิ่มขนาดของตัวแปร sort_buffer_size
- เพิ่มขนาดของตัวแปร read_rnd_buffer_size
- เปลี่ยน tmpdir ให้ชี้ไปที่ไฟล์ระบบ (dedicated filesystem) ที่กำหนดไว้ว่ามีที่ว่างจำนวนมาก

ตามปกติแล้ว MySQL จะเรียงลำดับข้อมูลของทุก GROUP BY col1, co2, ... ราวกับว่าได้มีการกำหนด ORDER BY col1, col2, ... โดยถ้ามีการสั่ง ORDER BY ที่ใช้ลิสต์ของคอลัมน์เดียวกันอย่างแน่ชัด MySQL จะไม่ทำการประมวลผลซึ่งจะไม่เกิดโทษในเรื่องของความเร็ว (speed penalty) แต่อย่างไรก็ตาม มันก็ยังทำการเรียงลำดับอยู่ แต่ถ้า query มี GROUP BY แต่เราต้องการหลีกเลี่ยงค่าใช้จ่ายที่เกิดจากการเรียงลำดับผลลัพธ์ เราสามารถทำได้โดยสั่ง ORDER BY NULL

3.3.3.10 MySQL ประมวลผลคำสั่ง LIMIT อย่างไร

ในบางกรณี MySQL จะรับมือกับ query ที่ใช้ LIMIT row_count แต่ไม่มี HAVING ด้วยวิธีที่ต่างออกไป:

- ถ้าเลือกเพียงไม่กี่แถวด้วยคำสั่ง LIMIT แล้ว MySQL จะใช้ดัชนีเฉพะในบางกรณี แต่ปกติแล้วมักจะใช้การสแกนทั้งตารางมากกว่า
- ถ้าใช้คำสั่ง LIMIT row_count ร่วมกับ ORDER BY แล้ว MySQL จะเลิกการเรียงลำดับทันทีที่มันพบ row_count แรกแทนที่จะทำการเรียงลำดับทั้งตาราง
- เมื่อใช้คำสั่ง LIMIT row_count ร่วมกับ DISTINCT แล้ว MySQL จะหยุดทันทีที่มันพบแถวที่ไม่ซ้ำของ row_count
- ในบางกรณีจะสามารถแก้ปัญหา GROUP BY โดยการอ่านคีย์ตามลำดับ (หรือทำการเรียงลำดับคีย์) จากนั้นทำการคำนวณผลสรุปออกมาจนกระทั่งค่าคีย์เกิดการเปลี่ยนแปลง ซึ่งในกรณีนี้ LIMIT row_count จะไม่คำนวณค่า GROUP BY ที่ไม่จำเป็น
- ทันทีที่ MySQL ได้ส่งจำนวนแถวที่ต้องการไปยังลูกข่าย มันจะทำการยกเลิก query นอกเสียจากว่ามีการใช้ SQL_CALC_FOUND_ROWS
- LIMIT 0 จะคืนค่าเซตว่างมาทันที ซึ่งมีประโยชน์ในการตรวจสอบ query หรือเมื่อต้องการชนิดของคอลัมน์ในคอลัมน์ของผลลัพธ์
- เมื่อเซิร์ฟเวอร์ใช้ตารางชั่วคราวในการแก้ปัญหา query, LIMIT row_count จะใช้ในการคำนวณหาขนาดของที่ว่าง (space) ที่ต้องการ

3.3.3.11 สามารถหลีกเลี่ยงการสแกนตารางได้อย่างไร

MySQL จะใช้การสแกนตารางเมื่อ:

- ตารางเล็กมากจนคิดว่าใช้การสแกนตารางจะเร็วกว่าไปหาคีย์ ซึ่งจะเป็นกรณีปกติของตารางที่มีน้อยกว่า 10 แถว หรือมีความยาวของแถวน้อย
- ไม่มีข้อจำกัดที่ใช้ได้ใน ON หรือ WHERE สำหรับคอลัมน์ของดัชนี
- ใช้การเปรียบเทียบคอลัมน์ของดัชนีกับค่าคงที่ และ MySQL คำนวณ (ยึดหลัก index tree) แล้วว่าค่าคงที่นั้นครอบคลุมส่วนของตารางที่ใหญ่มากจนใช้การสแกนตารางน่าจะเร็วกว่า
- ใช้คีย์ที่มีค่า cardinality ต่ำ (มีหลายแถวที่เข้ากันกับค่าของคีย์) ผ่านทางอีกคอลัมน์หนึ่ง ซึ่งในกรณีนี้ MySQL จะสมมุติว่าการใช้คีย์นั้น อาจต้องทำการหาคีย์หลายครั้ง และการสแกนทั้งตารางน่าจะเร็วกว่า

ในตารางขนาดเล็ก การใช้การสแกนตารางน่าจะเหมาะสมแล้ว แต่สำหรับตารางขนาดใหญ่ ควรลองใช้ เทคนิคต่อไปนี้เพื่อหลีกเลี่ยงการสแกนตารางโดยไม่เหมาะสม:

- ใช้คำสั่ง ANALYZE TABLE tbl_name เพื่ออัพเดตการกระจายของคีย์สำหรับการสแกนตาราง
  ใช้คำสั่ง FORCE INDEX สำหรับการสแกนตารางเพื่อบอก MySQL ว่าการสแกนตารางนั้นจะเปลืองมากกว่าการใช้ดัชนีที่กำหนดให้
- ใช้คำสั่ง SET max_seeks_for_key=1000 เพื่อบอกตัวประมวลผลว่าไม่ให้การสแกนคีย์ใดๆ ใช้มากกว่า 1000 ครั้ง

3.3.3.12 ความเร็วในการทำคำสั่ง INSERT

ในการใส่ข้อมูล (insert) ลงเร็คคอร์ดจะถูกตัดสินโดยปัจจัยต่อไปนี้ โดยเลขด้านหลังจะระบุอัตราส่วนโดยประมาณ:

- การติดต่อ (3)
- การส่ง query ไปยังเซิร์ฟเวอร์ (2)
- วิเคราะห์ query (2)
- ใส่ข้อมูลเร็คคอร์ด (1*ขนาดของเร็คคอร์ด)
- ใส่ข้อมูลดัชนี (1*จำนวนดัชนี)
- ปิดการติดต่อ (1)
  โดยสิ่งที่ไม่ได้นำเข้ามาพิจารณาด้วย คือ ค่าใช้จ่ายตอนเริ่มต้นในการเปิดตาราง ซึ่งจะถูกทำหนึ่งครั้งสำหรับแต่ละ query ที่ทำงานพร้อมๆ กัน
  ขนาดของตารางจะทำให้การใส่ข้อมูลดัชนีช้าลงตาม log N (สมมุติให้ดัชนีเป็น B-tree) โดยเราจะสามารถใช้กระบวนการต่อไปนี้ในการเพิ่มความเร็วของการใส่ข้อมูล
- ถ้ามีการใส่ข้อมูลหลายแถวจากลูกข่ายเดียวกันในเวลาเดียวกัน ให้ใช้สเตตเมนต์ของคำสั่ง INSERT ที่มีหลายลิสต์ของ VALUE เพื่อใส่ข้อมูลในหลายๆ แถวในเวลาเดียวกัน ซึ่งจะเร็วกว่าการใช้สเตตเมนต์ของคำสั่ง INSERT ที่จะแยกใส่ข้อมูลทีละแถวมาก (บางกรณีอาจเร็วกว่าหลายเท่าทีเดียว) แต่ถ้ามีการใส่ข้อมูลในตารางที่ไม่ว่าง เราอาจปรับตัวแปร bulk_insert_buffer_size เพื่อให้เร็วขึ้นไปอีกได้
- ถ้ามีการใส่ข้อมูลหลายๆ แถวจากต่างลูกข่ายกัน เราสามารถเพิ่มความเร็วโดยการใช้คำสั่ง INSERT DELAYED
- ในตาราง MyISAM เราสามารถใส่ข้อมูลหลายๆ แถวในเวลาเดียวกับที่ SELECT กำลังทำงานอยู่ เมื่อไม่มีแถวที่ถูกลบ (deleted row) อยู่ในตาราง
- เมื่อทำการโหลดตารางจากไฟล์ข้อความ (text file) ให้ใช้คำสั่ง LOAD DATA INFILE ซึ่งจะเร็วกว่าการใช้คำสั่ง INSERT หลายๆ ครั้งถึง 20 เท่าทีเดียว
- ในงานบางอย่าง เราสามารถทำให้ LOAD DATA INFILE มีความเร็วมากขึ้นไปอีกเมื่อตารางนั้นมีหลายดัชนี โดยให้ทำตามขั้นตอนต่อไปนี้:
  1. สร้างตารางด้วยคำสั่ง CREATE TABLE (เป็นทางเลือก)
  2. สั่งดำเนินการ (execute) FLUSH TABLES หรือคำสั่ง mysqladmin flush-tables

3. ใช้ myisamchk –keys-used=0 –rq/path/to/db/tbl_name ซึ่งจะเอาทุกการทำงานของดัชนีออกจากตาราง
4. ใส่ข้อมูลลงไปในตารางด้วยคำสั่ง LOAD DATA INFILE ซึ่งจะไม่มีการอัพเดตดัชนีใดๆ ดังนั้นจึงเร็วมาก
5. ถ้าจะให้ตารางเป็นตารางที่ใช้อ่านอย่างเดียว ให้ใช้ myisampack เพื่อทำให้มันเล็กลง
6. สร้างดัชนีอีกครั้งด้วยคำสั่ง myisamchk –r-q/path/to/db/tbl_name ซึ่งจะสร้าง tree ของดัชนีขึ้นในหน่วยความจำก่อนที่จะทำการเขียนลงดิสก์ โดยการทำเช่นนี้จะช่วยให้เร็วขึ้นมาก เพราะจะหลีกเลี่ยงการค้นหาในดิสก์ และ tree ของดัชนีจะมีความสมดุลอีกด้วย
7. สั่งคำเนินการ FLUSH TABLES หรือคำสั่ง mysqladmin flush-tables

ข้อสังเกต การใช้ LOAD DATA INFILE จะทำการประมวลผลเบื้องต้นเมื่อใส่ข้อมูลลงในตาราง MyISAM ที่ว่าง โดยความแตกต่างหลัก คือ เราสามารถใช้ myisamchk ในการจองพื้นที่ของหน่วยความจำชั่วคราว (temporary memory) ในการสร้างดัชนีได้มากกว่าที่เราต้องการให้เซิร์ฟเวอร์จองให้ดัชนีที่ถูกสร้างขึ้นใหม่ตอนที่คำเนินการ LOAD DATA INFILE มากๆ

- สามารถเพิ่มความเร็วของการคำเนินการ INSERT ที่มีหลายเสตตเมนต์ (multi-statement) โดยการล็อค (lock) ตาราง

  ประโยชน์ของวิธีการนี้จะเกิดจากบัฟเฟอร์ของดัชนีจะถูกนำลงดิสก์เพียงครั้งเดียว หลังจากที่ทุกๆการทำ INSERT สำเร็จแล้ว ซึ่งโดยปกติแล้ว จะมีบัฟเฟอร์ของดัชนีจำนวนมากที่ถูกนำลงดิสก์ตามความแตกต่างของสเตตเมนต์ใน INSERT แต่การล็อคภายนอกเช่นนี้จะไม่เกิดประโยชน์ถ้าเป็นการใส่ข้อมูลทุกแถวด้วยสเตตเมนต์เดียว

  สำหรับตารางแบบแทรนแซคชั่น ควรใช้คำสั่ง BEGIN/COMMIT แทนคำสั่ง LOCK TABLES

  คำสั่ง INSERT, UPDATE และ DELETE นั้น ปกติจะเร็วมากใน MySQL แต่เรายังสามารถเพิ่มประสิทธิภาพโดยรวมได้ด้วยการใส่ล็อคเข้าไปในทุกอย่างที่ต้องใช้การใส่ข้อมูหรือการอัพเดต มากกว่า 5 ครั้งในหนึ่งแถว แต่ถ้าเรามีการใส่ข้อมูลในแถวหนึ่งๆ หลายครั้ง ควรใช้คำสั่ง LOCK TABLES หลังจากนั้นสักพักให้ตามด้วยคำสั่ง UNLOCK TABLES หนึ่งครั้ง (ประมาณ 1000 แถว) เพื่ออนุญาตให้ thread อื่นสามารถเข้าถึงตารางได้ ซึ่งวิธีการยังคงประสิทธิภาพที่ดีเอาไว้

  แต่อย่างไรก็ตาม INSERT ก็ยังคงช้ากว่าการใช้ LOAD DATA INFILE ถึงแม้จะใช้วิธีการเพิ่มความเร็วแล้วก็ตาม
- สามารถเพิ่มความเร็วของตาราง MyISAM สำหรับทั้ง LOAD DATA INFILE และ INSERT ด้วยการเพิ่มขนาดแคชของคีย์โดยการเพิ่มตัวแปร key_buffer_size ซึ่งเป็นตัวแปรของระบบ

3.3.3.13 ความเร็วในการทำคำสั่ง UPDATE

การอัพเดตจะถูกประมวลผลเช่นเดียวกับการทำ SELECT แต่จะเพิ่มค่าใช้จ่าย (overhead) ในการเขียนเข้าไปอีก โดยความเร็วของการเขียนจะขึ้นอยู่กับจำนวนของข้อมูลที่กำลังถูกอัพเดต และจำนวนของดัชนีที่ถูกอัพเดต ซึ่งดัชนีที่ไม่มีการเปลี่ยนแปลงจะไม่ถูกอัพเดต

การเพิ่มความเร็วในการอัพเดต คือ ทำให้การอัพเดตล่าช้า (delay) แล้วทำหลายอัพเดตในแถวหนึ่งๆ ทีหลัง ซึ่งถ้าใช้การล็อคตาราง การทำหลายอัพเดตในแถวหนึ่งๆ จะเร็วกว่าการทำทีละอัน

3.3.3.14 ความเร็วในการทำคำสั่ง DELETE

เวลาในการลบแต่ละเร็คคอร์ดจะเป็นสัดส่วนโดยตรงกับจำนวนของดัชนี ซึ่งถ้าจะทำให้การลบเร็คคอร์ดเร็วขึ้น จะต้องเพิ่มขนาดแคชของคีย์

ถ้าต้องการลบทุกๆแถวในตารางให้ใช้คำสั่ง TRUNCATE TABLE tbl_name ซึ่งจะเร็วกว่าการใช้คำสั่ง DELETE FROM tbl_name

3.4 เทคนิคพิเศษในการประมวลผล

ผู้พัฒนา MySQL ได้พยายามเพิ่มประสิทธิภาพของ optimizer นั่นคือ การทำให้มันฉลาดและเร็วขึ้นเสมอในทุกๆ เวอร์ชั่น โดยแนวทางในการพัฒนานั้น จะได้มาจากกระแสตอบรับของผู้ใช้ในชีวิตจริง และในการที่จะตัดสินใจได้ดีนั้น MySQL จะพยายามตอบคำถามที่สำคัญเหล่านี้

- มีการใช้ดัชนีซึ่งจะทำให้หาแถวที่ต้องการได้เร็วขึ้นหรือไม่
- ดัชนีใดที่ดีที่สุด ถ้ามีหลายตารางที่เข้ามาเกี่ยวข้อง แล้วดัชนีใดที่ดีที่สุดกับแต่ละตาราง
- มีตารางใดที่ขึ้นตรงกับตารางใดในการ join หรือไม่
- ลำดับในการ join ตารางแบบใดที่เหมาะสมที่สุด

ซึ่งสิ่งที่สำคัญอีกสิ่งก็คือ MySQL ต้องตัดสินใจให้เร็วที่สุดเท่าที่เป็นไปได้ มิฉะนั้น มันอาจจะใช้เวลาในการตัดสินใจว่าจะดำเนินการอย่างไรนานกว่าเวลาที่ใช้ในการดำเนินการกับ query นั้นจริงๆ เสียอีก

3.4.1 การวัด cost ของ query

เราจะวัด cost ของ query ได้อย่างไร

- cost ของ query จะเกิดจาก
  - CPU time : ในการประมวลผลโปรแกรม
  - I/O time : ในการอ่านข้อมูล และดัชนีจากดิสก์
  - Network time : ในการส่งข้อมูลผ่านเน็ตเวิร์ก (network)
- ในระบบแบบส่วนกลาง หรือในระบบ client/server นั้น ข้อมูลมักจะอยู่บนเซิร์ฟเวอร์เดียว
  - ดังนั้น network time จึงเป็นค่าที่ไม่สัมพันธ์ หรือเป็นค่าคงที่
  - CPU time จะมีค่าน้อยมากเมื่อเทียบกับค่าของ I/O time
  - ดังนั้น เราจะสนใจไปที่การลด I/O time ให้มีค่าน้อยที่สุด

3.4.2 การประมวลผลโดยใช้คำสั่ง IN แทนที่คำสั่ง OR

ถ้าเงื่อนไขภายใน WHERE เป็นคำสั่ง OR ทั้งหมด และกระทำกับคอลัมน์เพียงคอลัมน์เดียว เช่น

SELECT * FROM categories WHERE categoryID = 100 OR categoryID = 120 OR categoryID = 130

เราสามารถ optimize โดยการแปลงคำสั่ง OR เหล่านั้น เป็น IN (...) แทนได้ ซึ่งจากการทดลองพบว่า MySQL จะใช้เวลาในการดำเนินการกับคำสั่ง IN (...) น้อยกว่า และจากตัวอย่างด้านบนเราสามารถแปลงเป็นคำสั่ง IN (...) ได้ดังนี้

SELECT * FROM categories WHERE categoryID IN (100,120,130)

เมื่อเราสั่ง EXPLAIN query ทั้ง 2 จะพบว่าได้เอาต์พุตเหมือนกัน เนื่องจากโดยการทำงานแล้วมันจะเหมือนกัน แต่ปรากฏว่าเมื่อทดลองแล้ว เราจึงทราบว่าในกรณีนี้ MySQL จะทำการแปลงคำสั่ง OR หลายๆ ตัวให้เป็น IN(...) เพียงตัวเดียว อย่างไรก็ตาม เราได้ใช้จำนวนของ ID ที่เพิ่มขึ้นเรื่อยๆ ดังนั้นคำสั่ง query จึงสั้นลงเมื่อใช้ IN (...) เพราะยิ่งมีคำสั่ง query ที่เล็ก นั่นหมายความว่า จะมีการเสียทรัพยากรในการวิเคราะห์น้อยเท่านั้น ซึ่งส่งผลให้เร็วยิ่งขึ้น

ตัวอย่างของคำสั่ง query ที่ใช้ multi-OR และ IN(...) และเวลาที่ใช้ดำเนินการกับ query นั้น

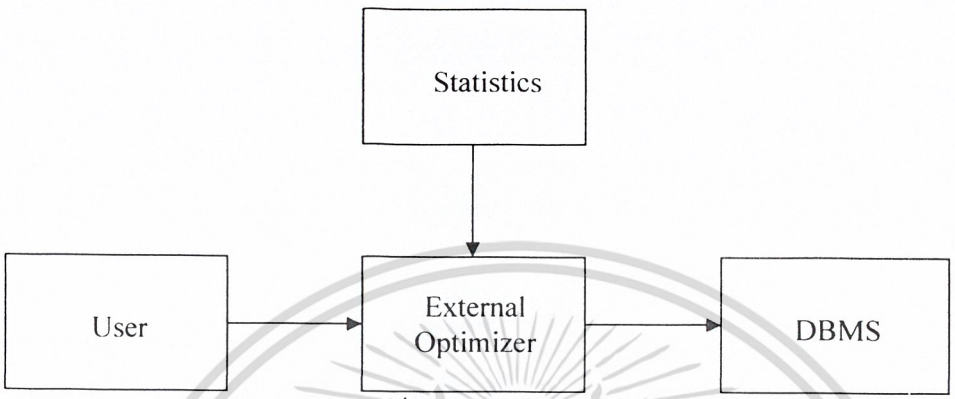
```
mysql> SELECT Url FROM Headline WHERE Id IN(153513, 10231599,1396322);
+-------------------------------------------------+
| Url                                             |
+-------------------------------------------------+
| http://biz.yahoo.com/bond/010117/bf.ht          |
| http://biz.yahoo.com/e/021101/yhoo10-q.ht       |
| http://biz.yahoo.com/bw/030331/315850_1.ht      |
+-------------------------------------------------+
3 rows in set (0.00 sec)

mysql>  SELECT Url FROM Headline WHERE Id = 1531513 OR Id = 10231599 OR
        Id = 13962322;
+-------------------------------------------------+
| Url                                             |
+-------------------------------------------------+
| http://biz.yahoo.com/bond/010117/bf.ht          |
| http://biz.yahoo.com/e/021101/yhoo10-q.ht       |
| http://biz.yahoo.com/bw/030331/315850_1.ht      |
+-------------------------------------------------+
3 rows in set (0.03 sec)
```

บทที่ 4

การออกแบบระบบ

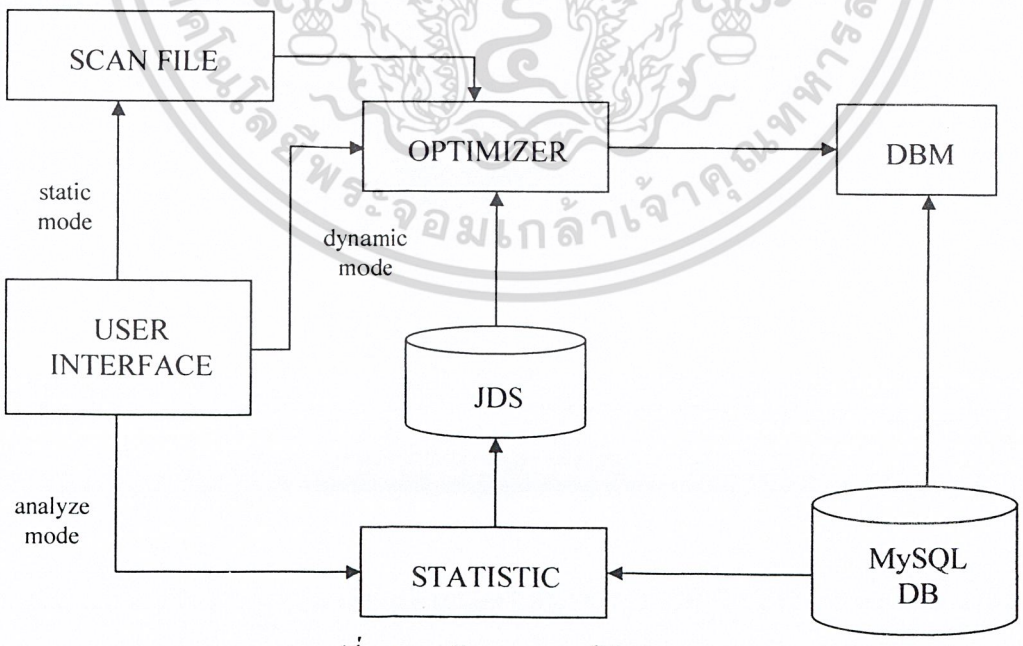
4.1 ภาพรวมของระบบ



รูปที่ 4-1 ภาพรวมของระบบ

ทำการพัฒนาโปรแกรมขึ้นมาอยู่ตั้นกลางระหว่าง ผู้ใช้และระบบจัดการฐานข้อมูล ซึ่งโปรแกรมจะรับคำสั่ง sql จากผู้ใช้ หรือนำคำสั่งมาจากไฟล์โค้ดของโปรแกรมอื่นๆ มาทำการประมวล หาคำสั่ง sql ที่คิดว่าดีที่สุด โดยจะนำข้อมูลทางสถิติที่เก็บไว้มาทำการพิจารณาควบคู่กันไปด้วย หลังจากได้คำสั่ง sql ออกมาแล้ว จะทำการส่งคำสั่ง sql นี้ไปรันบนระบบจัดการฐานข้อมูล เพื่อหาผลลัพธ์ตามที่ผู้ใช้ต้องการออกมา

4.2 โครงสร้างของโปรแกรม



รูปที่ 4-2 สถาปัตยกรรมของโปรแกรม

การทำงานของโปรแกรมนี้คือ หลังจากที่ผู้ใช้ได้ทำการเชื่อมต่อกับระบบจัดการฐานข้อมูลผ่านทางโปรแกรมนี้เรียบร้อยแล้ว ผู้ใช้จะสามารถใช้โปรแกรมทำงานได้ 2 ฟังก์ชัน คือ ส่วน optimizer และส่วน statistics โดยส่วน optimizer นั้นจะรับแบ่งเป็น 2 แบบคือ แบบ static และ dynamic ซึ่งแบบ dynamic นั้น โปรแกรมจะรับ query มาจากหน้าจออินเทอร์เฟซ แล้วส่ง query ที่ได้จากหน้าจอนั้นเข้าไปยังส่วนของ optimizer ส่วนแบบ static นั้นโปรแกรมจะนำไฟล์โค้ดของโปรแกรมอื่นๆ มาทำการสแกนเพื่อหา query ที่อยู่ในโปรแกรมนั้นๆ แล้วส่ง query ที่ได้ไปให้ส่วน optimizer เพื่อทำการประมวลผลตามทฤษฎีที่ได้ศึกษา และทำการทดลองมาในขั้นต้น โดยใช้ข้อมูลทางสถิติที่ได้ทำการเก็บไว้ก่อนหน้านี้มาพิจารณา จากนั้นจะได้ผลลัพธ์ออกมาเป็น query ที่เป็นไปตามกฎ และจะทำการแสดง query ออกมาทางหน้าจออินเทอร์เฟซ พร้อมทั้งส่ง query เข้าไปรันในระบบจัดการฐานข้อมูลที่ได้ทำการเชื่อมต่อไว้แล้วในตอนแรก เพื่อทำการหาผลลัพธ์ของ query ออกมา และแสดงผลลัพธ์นั้นทางหน้าจออินเทอร์เฟซ อีกส่วนหนึ่งคือ ส่วน statistic ซึ่งจะเป็นส่วนที่ทำการเก็บสถิติจากฐานข้อมูลของผู้ใช้ ซึ่งค่าที่เก็บไว้จะไม่มีการอัพเดต จนกว่าจะมีการเข้ามาใช้งานโปรแกรมในส่วนนี้

4.3 ส่วนต่างๆของโปรแกรม

มีส่วนที่เกี่ยวข้องในโปรแกรม 4 ส่วน ดังรูป 9-2 ดังนี้

4.3.1 USER INTERFACE

ส่วนนี้จะเป็นส่วนที่มีการติดต่อกับผู้ใช้ ซึ่งผู้ใช้จะใช้งานโปรแกรมผ่านอินเทอร์เฟซในส่วนนี้ โดยทำการใส่ input ให้กับระบบ และ output ที่ระบบได้ออกมา ก็จะมาแสดงอยู่ในส่วนอินเทอร์เฟซนี้ด้วยเช่นกัน

4.3.2 SCAN FILES

จะนำไฟล์โค้ดต่างๆมาทำการสแกนหาคำสั่ง sql ที่มีอยู่ในโค้ดนั้น เพื่อส่งไปให้ส่วนของ optimizer ทำการประมวลผลต่อไป

4.3.3 OPTIMTZER

ส่วนนี้จะรับคำสั่ง sql เข้ามาจากหน้าจออินเทอร์เฟซ หรือจากการสแกนไฟล์โค้ด เพื่อทำการประมวลผลหาคำสั่ง sql ที่ดีที่สุดออกมา ซึ่งจะทำการพิจารณาโดยอ้างอิงมาจากข้อมูลสถิติที่เก็บอยู่ในฐานข้อมูล (JDS) เมื่อทำการประมวลผลเรียบร้อยแล้ว จะส่งคำสั่ง sql ที่ได้เข้าไปรันที่ระบบจัดการฐานข้อมูลเพื่อหาผลลัพธ์ที่ต้องการ

4.3.4 STATISTIC

ส่วนนี้จะทำการเก็บสถิติของตารางที่สนใจ โดยจะทำการดึงข้อมูลจากฐานข้อมูลที่ใช้งานอยู่จริง มาคำนวณหาค่าสถิติที่สนใจและทำการเก็บค่าลงในฐานข้อมูล (JDS)

4.3.5 DBMS

เป็นระบบจัดการฐานข้อมูลที่โปรแกรมทำการติดต่อ เพื่อหาผลลัพธ์ของ query และเก็บข้อมูลตารางของผู้ใช้

4.4 INPUT ของระบบ

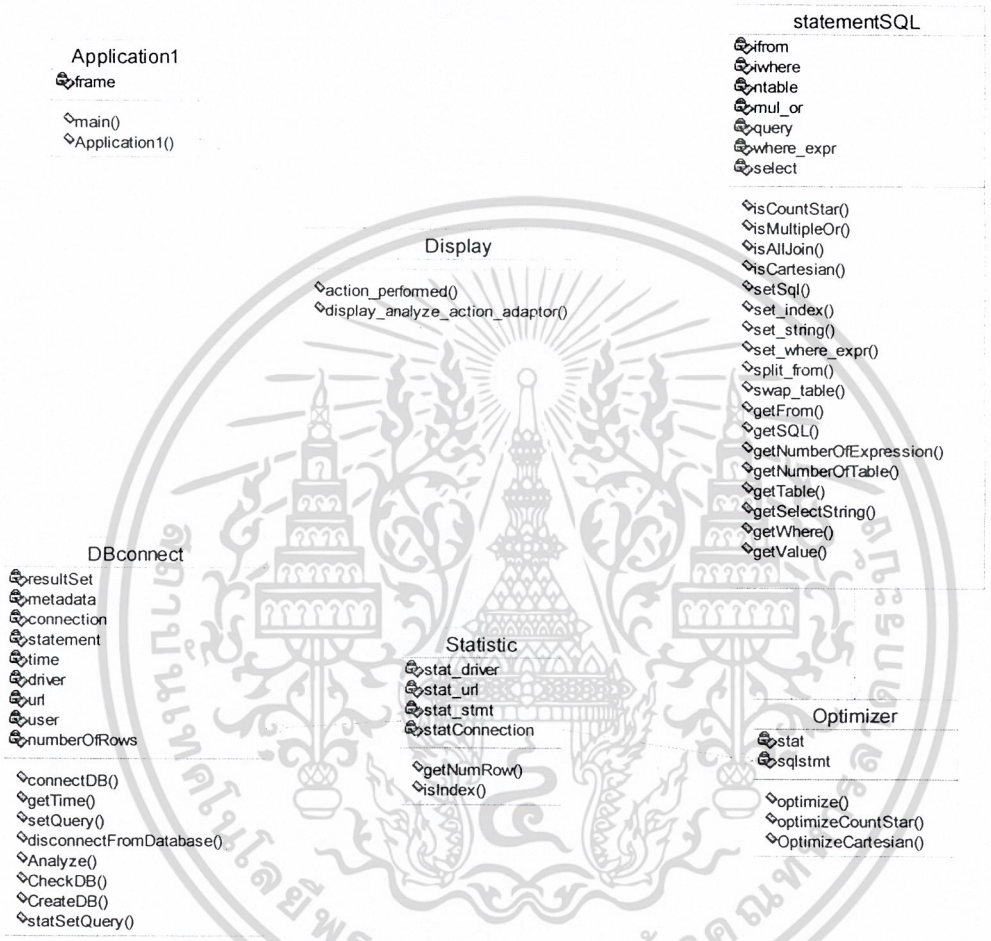
1. user name, password เป็น input ในขั้นตอนของการ connect กับฐานข้อมูล ซึ่งโปรแกรมจะทำการรับค่าจาก dialog เข้ามาแล้วทำการติดต่อกับฐานข้อมูลตาม user และ password ที่ได้รับมา
2. คำสั่ง sql ที่ผู้ใช้ทำการกรอกมาทางหน้าจออินเทอร์เฟซ
3. ไฟล์โค้ดของโปรแกรมอื่นๆที่จะนำมาทำการสแกน

4.5 OUTPUT ของระบบ

1. คำสั่ง sql ที่ผ่านการ optimize แล้ว โดยจะส่งไปรันที่ระบบจัดการฐานข้อมูล พร้อมกับแสดงขึ้นมาในหน้าจออินเทอร์เฟซ
2. เวลาที่ใช้ในการค้นหาผลลัพธ์ของ query ที่ทำการประมวลผลแล้ว ซึ่งจะแสดงมาทางหน้าจออินเทอร์เฟซ
3. จำนวน row ของผลลัพธ์ที่ได้ออกมาจากการส่ง query ไปรันในระบบจัดการฐานข้อมูล
4. แสดงผลลัพธ์ทั้งหมดของการ query ทางหน้าจออินเทอร์เฟซ
5. ไฟล์โค้ดที่ผ่านการ optimize คำสั่ง sql แล้ว

บทที่ 5
การพัฒนาโปรแกรม

5.1 การออกแบบโปรแกรม



รูปที่ 5-1 แสดง Class Diagram ของโปรแกรม

หลังจากที่ทำการออกแบบโครงสร้างของโปรแกรมได้แล้ว จากนั้นจะเป็นขั้นตอนของการพัฒนาโปรแกรมตามโครงสร้างที่ได้มีการออกแบบไว้ โดยโครงงานวิจัยนี้จะใช้ภาษา Java ในการพัฒนา เนื่องจากเป็นภาษาที่ใช้การเขียนโปรแกรมในเชิงวัตถุ (Object Oriented Programming) ที่มีความยืดหยุ่นสูง และมี API ที่สนับสนุนการทำงานหลายแบบเพียงพอต่อการพัฒนาระบบ จึงทำให้การพัฒนาโปรแกรมทำได้ง่าย

โปรแกรมที่ทำการพัฒนาขึ้นมานั้น จะมีการสร้างคลาส (class) ต่างๆตามหน้าที่การทำงานที่แตกต่างกันอย่างชัดเจน ซึ่งประกอบด้วยคลาสจำนวน 6 คลาส ดังที่แสดงในรูปที่ 5-1 โดยแต่ละคลาสจะมีการทำงานดังนี้

5.1.1 คลาส Application1

เป็นคลาสแอพพลิเคชันของ JAVA ซึ่งเมื่อมีการรันโปรแกรม คลาสนี้จะถูกเรียกขึ้นมาด้วย method main() โดยภายในคลาสนี้ได้มีการประกาศ object ของคลาส Display ไว้ ดังนั้นเมื่อมีการรันโปรแกรม คลาสนี้จะไปเรียกคลาส Display ขึ้นมาทำงาน

5.1.2 คลาส Display

เป็นคลาสที่ใช้อินเทอร์เฟซ (interface) กับผู้ใช้โปรแกรม โดยในคลาสนี้จะรับ action ที่เกิดจากผู้ใช้ ถ้าผู้ใช้คลิ๊กที่ปุ่ม CONNECT หรือที่ปุ่ม EXIT เพื่อติดต่อกับฐานข้อมูล และยกเลิกการติดต่อกับฐานข้อมูลตามลำดับแล้ว คลาสนี้จะเรียกคลาส DBconnect ขึ้นมาทำงาน และจะไปเรียกคลาส Statistic เพื่ออัพเดตค่าสถิติที่เก็บไว้ และถ้าผู้ใช้คลิ๊กที่ปุ่ม ANALYZE คลาสนี้ก็จะไปเรียกคลาส Statistic เช่นกัน แต่ถ้าผู้ใช้ใส่คำสั่ง query ที่ต้องการให้มีการดำเนินการ แล้วคลิ๊กที่ปุ่ม OK คลาสนี้จะไปเรียกคลาส Optimizer ขึ้นมาทำงาน

5.1.3 คลาส DBconnect

เป็นคลาสที่ใช้ในการติดต่อกับฐานข้อมูล รวมถึงการอัพเดตสถิติต่างๆ ที่เก็บไว้ในส่วน JDS โดยจะมีฟังก์ชั่นหลักๆ ดังนี้

5.1.3.1 connectDB()

เป็นฟังก์ชั่นที่จะเปิดการเชื่อมต่อกับฐานข้อมูล โดยจะมีการรับอินพุต (input) เป็น username และ password จากผู้ใช้ ซึ่งถ้าอินพุตทั้งสองที่ผู้ใช้กรอกเป็นข้อมูลที่ถูกต้องแล้ว มันจะไปเรียกคลาส Statistic เพื่ออัพเดตค่าสถิติที่เก็บไว้

5.1.3.2 getTime()

เป็นฟังก์ชั่นที่ใช้ส่งค่าของเวลาที่ใช้ในการดำเนินการกับ query ออกไปที่คลาส Display เพื่อนำค่าที่ได้ไปแสดงบนอินเทอร์เฟซของโปรแกรม

5.1.3.3 setQuery()

เป็นฟังก์ชั่นที่ใช้ส่ง query ไปที่ฐานข้อมูลเพื่อดำเนินการ แล้วนำผลลัพธ์ออกมาแสดง

5.1.3.4 disconnectFromDatabase()

เป็นฟังก์ชั่นที่ใช้ในการยกเลิกการเชื่อมต่อกับฐานข้อมูล

5.1.3.5 CheckDB()

เป็นฟังก์ชั่นที่ใช้ในการตรวจสอบว่ามีการเปลี่ยนแปลงตารางในฐานข้อมูลเกิดขึ้นหรือไม่ ซึ่งถ้ามีการเปลี่ยนแปลงแล้ว จะไปเรียกฟังก์ชั่น CreateDB() ขึ้นมาทำงาน แต่ถ้าไม่มีการเปลี่ยนแปลง จะไปเรียก Analyze() มาทำงานแทน

5.1.3.6 CreateDB()

เป็นฟังก์ชั่นที่จะลบข้อมูลต่างๆ ในตารางสถิติที่ได้เก็บไว้ในส่วนของ JDS จากนั้นจะสร้างโครงสร้างของตารางขึ้นมาใหม่ตามฐานข้อมูลหลังจากการเปลี่ยนแปลงใดๆ แล้วจะไปเรียกฟังก์ชั่น Analyze()

5.1.3.7 Analyze()

เป็นฟังก์ชั่นที่ใช้ในการอัพเดตสถิติในส่วนของ JDS ตามข้อมูล ณ ขณะนั้น เพื่อให้เป็นสถิติล่าสุด ซึ่งจะนำไปใช้ในการ optimize ของโปรแกรม

5.1.4 คลาส Optimizer

เป็นคลาสที่ใช้ optimize คำสั่ง query ให้มีประสิทธิภาพมากขึ้น จากนั้นจะส่ง query ใหม่ที่ผ่านการ optimize แล้วไปดำเนินการที่ฐานข้อมูล โดยผ่านคลาส Display ซึ่งจะไปเรียกคลาส DBconnect อีกต่อหนึ่ง การทำงานของฟังก์ชั่นต่างๆ ในคลาสนี้สามารถแสดงได้ดังรูปที่ 5-2 โดยฟังก์ชั่นหลักๆ ในคลาสนี้จะมีดังนี้

5.1.4.1 optimize()

เป็นฟังก์ชั่นที่จะรับค่าอินพุตเป็น String ซึ่งก็คือ query ที่ผู้ใช้โปรแกรมกรอกเข้ามาทางหน้าจออินเทอร์เฟซนั่นเอง โดยเมื่อได้รับอินพุตมาแล้ว ฟังก์ชั่นนี้จะไปเรียกฟังก์ชั่นต่างๆ ของคลาส statementSQL, ซึ่งจะกล่าวต่อไปในเรื่องของคลาส statementSQL, แล้วทำการตรวจสอบว่า query ที่ส่งเข้ามานั้น ตรงกันกับรูปแบบ query ที่ต้องการการ optimization หรือไม่

5.1.4.2 optimizeCountStar()

เป็นฟังก์ชั่นที่ใช้ในการ optimize query ที่ตรงกับกรณีของฟังก์ชั่น isCountStar() ของคลาส statementSQL โดยการ optimize นั้น จะทำด้วยการสลับตารางของ FROM ให้นำตารางที่มีขนาดใหญ่ที่สุดเป็นตารางแรกในการ join แล้วจากนั้นก็เรียงลำดับลงไปเรื่อยๆ จนถึงตารางที่มีขนาดเล็กที่สุด

5.1.4.3 optimizeMultipleOR()

เป็นฟังก์ชั่นที่ใช้ optimize query ที่ตรงกับกรณีของฟังก์ชั่น isMultipleOR() และ isAllJoin() ของคลาส statementSQL และ isIndex ของคลาส Statistic โดยการ optimize จะทำด้วยการเปลี่ยน query ที่เข้ามาจาก query ที่มีเงื่อนไขภายใน WHERE เชื่อมกันด้วย OR ทั้งหมด และกระทำอยู่บนคอลัมน์เดียวกันทั้งหมด เป็น query ที่จะใช้คำสั่ง IN(...) แทน

5.1.4.4 optimizeCartesian()

เป็นฟังก์ชั่นที่ใช้ optimize query ที่ตรงกับกรณีของฟังก์ชั่น isCatesian ของคลาส statementSQL โดยการ optimize จะทำด้วยการสลับตารางของ FROM ให้นำตารางที่มีขนาดเล็กที่สุดเป็นตารางแรกในการ join แล้วจากนั้นก็เรียงลำดับขึ้นไปเรื่อยๆ จนถึงตารางที่มีขนาดใหญ่ที่สุด

5.1.5 คลาส statementSQL

เป็นคลาสที่จะถูกเรียกจากคลาส Optimizer โดยเปรียบเหมือนกับเป็นคลาสที่ใช้ในการตรวจสอบว่า query ที่เข้ามานั้นจะตรงกับกรณีใดของ optimization โดยฟังก์ชั่นภายในคลาสนี้จะมีดังนี้

5.1.5.1 setsql()

เป็นฟังก์ชั่นที่รับค่าอินพุตเป็น query จากคลาส Optimizer ซึ่งเมื่อรับเข้ามาแล้ว ฟังก์ชั่นนี้จะไปเรียกฟังก์ชั่น set_index(), set_string(), set_where_expr() และ splitFrom() ตามลำดับ

รูปที่ 5-2 แสดง Flow Chart ของคลาส optimization()

5.1.5.2 set_index()

เป็นฟังก์ชั่นที่ใช้ในการระบุว่า คำสั่ง FROM, WHERE, GROUP BY, HAVING และ ORDER BY นั้นอยู่ในตำแหน่งที่เท่าไรของ query ที่รับมา

5.1.5.3 set_string()

เป็นฟังก์ชั่นที่ใช้ในการตัดแบ่ง query ออกเป็นส่วนๆ โดยใช้ตำแน่งที่ได้จากการทำฟังก์ชั่น set_index() เป็นตัวอ้างอิงในการตัด query นั้นๆ

5.1.5.4 set_where_expr()

เป็นฟังก์ชั่นที่ใช้ในการตัดแบ่งเงื่อนไขภายใน WHERE ออกเป็นส่วนๆ โดยใช้คำเชื่อมต่างๆ เป็นตัวอ้างอิงในการตัด โดยในวิทยานิพนธ์นี้ได้ใช้คำสั่ง AND, OR และ BETWEEN ในการตัด

5.1.5.5 splitFrom()

เป็นฟังก์ชั่นที่ใช้ตัดแบ่ง query ที่อยู่หลัง FROM แต่อยู่หน้า WHERE เพื่อจะหาว่ามีจำนวนตารางที่เกี่ยวข้องกี่ตาราง และคือตารางใดบ้าง เพื่อจะได้นำไปเทียบกับสถิติที่เก็บไว้

5.1.5.6 isMultipleOR()

เป็นฟังก์ชั่นที่ใช้ในการตรวจสอบว่าเงื่อนไขภายใน WHERE นั้น ทุก expression ถูกเชื่อมกันด้วย OR ใช่หรือไม่

5.1.5.7 isAllJoin()

เป็นฟังก์ชั่นที่ใช้ในการหาว่าเงื่อนไขภายใน WHERE นั้น กระทำอยู่บนคอลัมน์เพียงคอลัมน์เดียวใช่หรือไม่

5.1.5.8 isCountStar()

เป็นฟังก์ชั่นที่ใช้ในการตรวจสอบว่าเป็นการ SELECT COUNT(*) และมีตารางที่เกี่ยวข้องกันมากกว่า 1 ตาราง และไม่มีเงื่อนไขภายใน WHERE ใช่หรือไม่

5.1.5.9 isCatesian()

เป็นฟังก์ชั่นที่ใช้ในการตรวจสอบว่าเป็นการ SELECT ที่ไม่มีเงื่อนไขภายใน WHERE และมีตารางเกี่ยวข้องมากกว่า 1 ตารางใช่หรือไม่

5.1.5.10 swapTable()

เป็นฟังก์ชั่นที่ใช้ในการสลับตาราง โดยจะนำตารางที่มีขนาดใหญ่กว่ามาทำการ join เป็นอันดับแรก แล้วเรียงลำดับลงไปเรื่อยๆ จนถึงตารางที่มีขนาดเล็กที่สุด

5.1.5.11getFrom()

เป็นฟังก์ชั่นที่จะส่งคืนเป็น String ของ query ที่อยู่หลัง FROM แต่อยู่หน้า WHERE

5.1.5.12 getNumberOfExpression()

เป็นฟังก์ชั่นที่จะส่งคืนเป็นค่าจำนวนของ expression ของเงื่อนไข WHERE หลังจากที่ถูกตัดแบ่งแล้ว

5.1.5.13 getNumberOfTable()

เป็นฟังก์ชั่นที่จะส่งคืนเป็นค่าจำนวนของตารางที่เกี่ยวข้อง

5.1.5.14 getTable()

เป็นฟังก์ชั่นที่ส่งคืนค่าอาเรย์ (array) ของชื่อของตารางที่เกี่ยวข้อง

5.1.5.15 getWhere()

เป็นฟังก์ชั่นที่จะส่งคืนเป็น String ที่เป็นเงื่อนไขภายใน WHERE

5.1.5.16 getSelectString()

เป็นฟังก์ชั่นที่จะส่งคืนเป็น String ที่เป็นด้านหลังของ SELECT แต่อยู่ด้านหน้าของ FROM

5.1.6 คลาส Statistical

เป็นคลาสที่ใช้ในการอัพเดตสถิติที่เก็บไว้ และใช้ติดต่อกับส่วน JDS ที่ใช้เก็บสถิติไว้ โดยจะมีฟังก์ชั่นหลัก ดังนี้

5.1.6.1 GetNumRow()

เป็นฟังก์ชั่นที่ใช้ส่งค่าของจำนวนแถวของแต่ละตารางซึ่งได้เก็บเป็นสถิติไว้อยู่แล้ว กลับไปยังจุดที่เรียกใช้

5.1.6.2 isIndex()

เป็นฟังก์ชั่นที่ใช้ในการตรวจสอบว่าคอลัมน์ที่เป็นอินพุตของฟังก์ชั่นนี้เป็นดัชนีของตารางหรือไม่ โดนจะไปตรวจสอบในส่วนสถิติที่ได้เก็บไว้ใน JDSs

บทที่ 6

ผลการทดลอง

6.1 ซอฟต์แวร์ที่ใช้ในการทดลอง

6.1.1 mysql-4.1.7-win

6.1.2 mysql-connector-java-3.0.14-production

6.1.3 MySQL-Front

6.1.4 mysql-administrator-1.0.19-win

6.1.5 mysql-query-browser-1.1.6-win

6.1.6 Borland JBuilder

6.2 ค่าตัวแปรต่างๆใน MySQL

การทดลองที่ทำนี้จะทำบนพื้นฐานค่า default ของ MySQL ทั้งหมด ซึ่งมีค่าหลักๆดังต่อไปนี้

6.2.1 ตัวแปรทั่วไป (General Parameters)

Key buffer : 7 M

Sort buffer size : 256 K

6.2.2 ตัวแปรของตารางประเภท InnoDB (InnoDB Parameters)

Buffer Pool Size : 10 M

Add.mem Pool Size : 2 M

Log File Size : 10 M

Log Buffer Size : 1 M

6.2.3 ตัวแปรเกี่ยวกับประสิทธิภาพ (Performance)

Query cache limit : 1 M

Minimal size of result : 4096 K

Cache Size : 0 K

Cache type : Cache all queries except SELECT SQL_NO_CACHE

6.3 ตารางที่ใช้ในการทดลอง

6.3.1 ตารางที่ใช้ทดลอง และ index ของตาราง

1. ตาราง categories : 150 rows

   PRIMARY : CategoryID

2. ตาราง customers : 3,800 rows

   PRIMARY : CustomerID

3. ตาราง employees : 560 rows

   PRIMARY : EmployeeID

4. ตาราง order_details : 355,000 rows

   PRIMARY : OrderID, ProductID

   fi0 : OrderID

   fi1 : ProductID

5. ตาราง orders : 100,000 rows

   PRIMARY : OrderID

   fi0 : CustomerID

   fi1 : EmployeeID

   fi2 : ShipVia

6. ตาราง products : 12,400 rows

   PRIMARY : ProductID

   fi0 : CategoryID

   fi1 : SupplierID

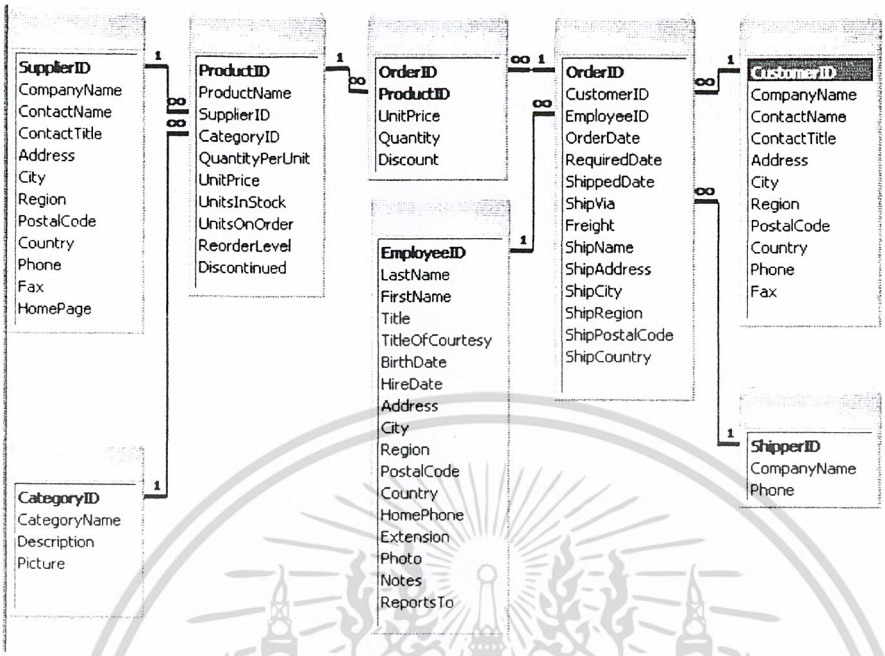
7. ตาราง shippers : 50 rows

   PRIMARY : ShipperID

8. ตาราง suppliers : 1,200 rows

   PRIMARY : SupplierID

6.3.2 ความสัมพันธ์ของตาราง



รูปที่ 6-1 ความสัมพันธ์ของตารางในการทดลอง

6.4 ผลการทดลองกับตารางประเภท InnoDB

6.4.1 การทดลองที่ 1

6.4.1.1 สมมติฐาน

การส่งคำสั่ง sql ที่มีการ query จำนวนแถว ของผลลัพธ์ที่มีการ join กันของตารางตั้งแต่ 2 ตารางขึ้นไป โดยไม่มีเงื่อนไขใน where ใน query เข้าไปรันที่ระบบจัดการฐานข้อมูล โดยให้ตารางที่มีจำนวน row ทั้งหมดน้อยที่สุด อยู่ทางด้านขวามือสุดแล้ว จะทำให้การระบบจัดการฐานข้อมูลทำการประมวลผลได้รวดเร็วยิ่งขึ้น

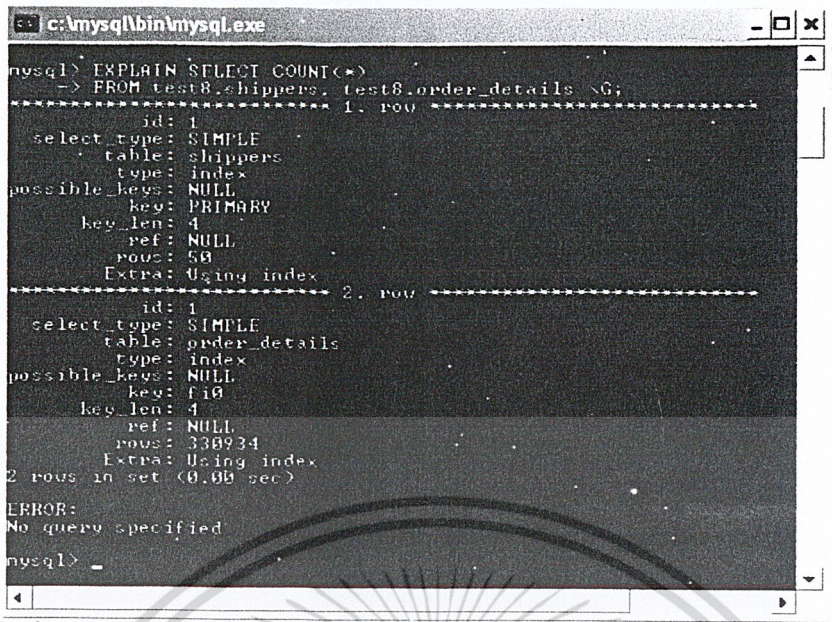
6.4.1.2 การทดลองแผนในการประมวลผล

ทำโดยการใช้คำสั่ง EXPLAIN

คำสั่งแรก คือ

EXPLAIN SELECT COUNT(*) FROM test8.shippers, test8.order_details;

ได้ผลลัพธ์ดังรูปที่ 6-2

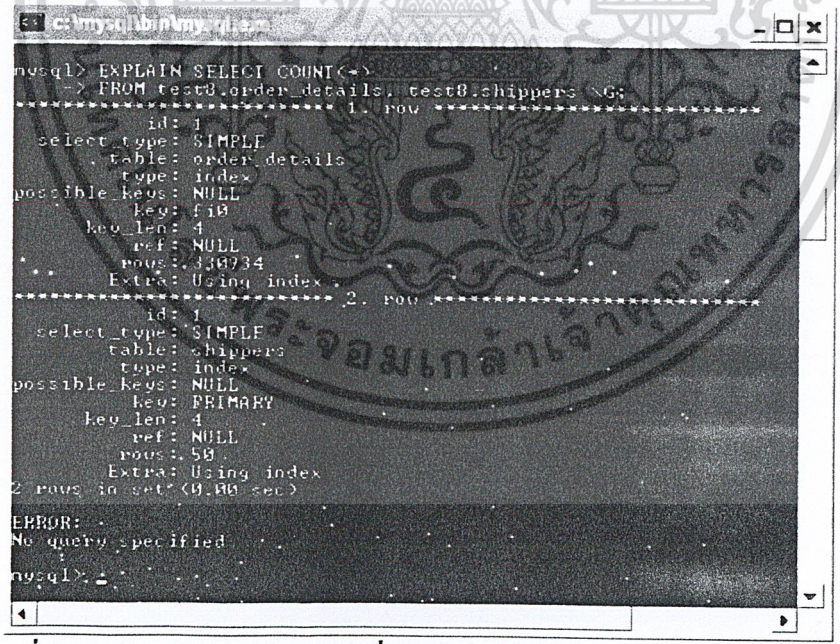


รูปที่ 6-2 แสดงการใช้ EXPLAIN กับคำสั่ง COUNT(*) โดยให้ตาราง shippers ขึ้นก่อน

คำสั่งที่สอง คือ

EXPLAIN SELECT COUNT(*) FROM test8.order_details, test8.shippers;

ได้ผลลัพธ์ดังรูปที่ 6-3



รูปที่ 6-3แสดงการใช้ EXPLAIN กับคำสั่ง COUNT โดยให้ตาราง order_details ขึ้นก่อน

จากการพิจารณาผลลัพธ์ที่ได้ทั้ง 2 อันตามรูปด้านบนแล้ว จะเห็นได้ว่า MySQL จะใช้ดัชนีมาช่วยในการjoin ตาราง และจะทำการ join ตาราง ตามลำดับที่อยู่ใน FROM

6.4.1.3 การจับเวลาในการประมวลผลก่อน ANAYZE TABLE

โดยส่งคำสั่งต่อไปนี้เข้าไปรันที่ระบบจัดการฐานข้อมูล

```
SELECT COUNT(*)
FROM test8.shippers, test8.order_details;
```

และ

```
SELECT COUNT(*)
FROM test8.orders_details, test8.shippers;
```

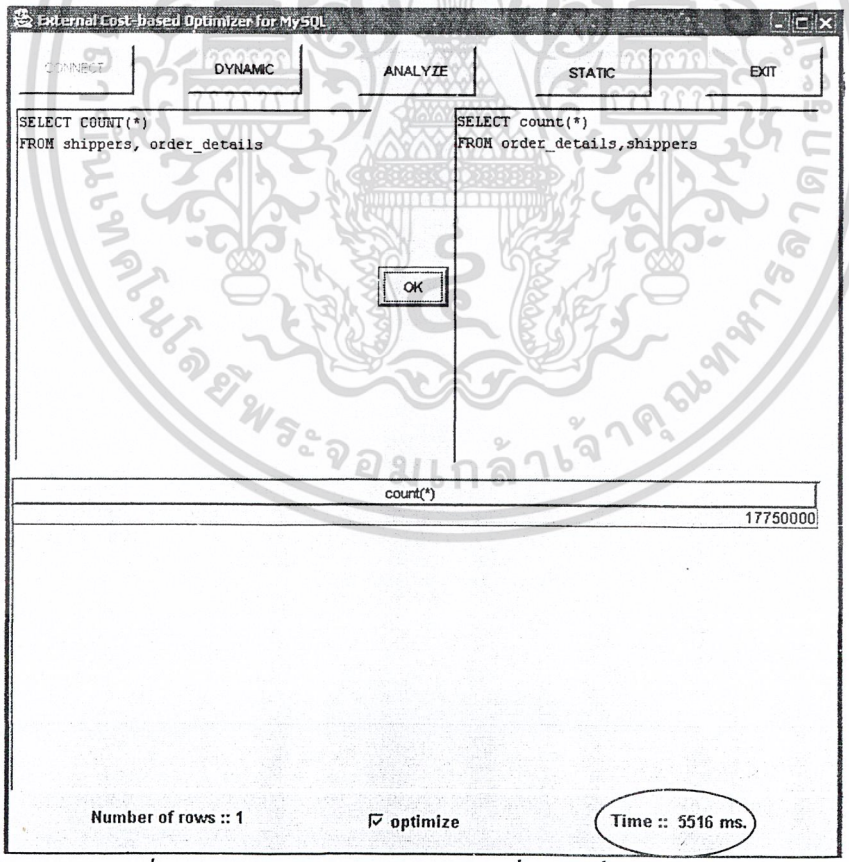
เมื่อทำการจับเวลาในการรันคำสั่งทั้ง 2 แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 19.52 19.61 19.64 19.50 24.45 (หน่วยเป็นวินาที)

คำสั่งสอง : 5.34 5.45 5.30 5.45 5.75 (หน่วยเป็นวินาที)

6.4.1.4 การทดลองกับโปรแกรมก่อน ANAYZE TABLE

จะได้ผลลัพธ์ดังรูปที่ 6-4



รูปที่ 6-4 แสดงการทำงานของโปรแกรมเมื่อส่งคำสั่ง COUNT(*)

6.4.1.5 การจับเวลาในการประมวลผลหลังจาก ANAYZE TABLE

โดยส่งคำสั่งต่อไปนี้เข้าไปรันที่ระบบจัดการฐานข้อมูล

```
SELECT COUNT(*)
FROM test8.shippers, test8.order_details;
```

และ

```
SELECT COUNT(*)
FROM test8.orders_details, test8.shippers;
```

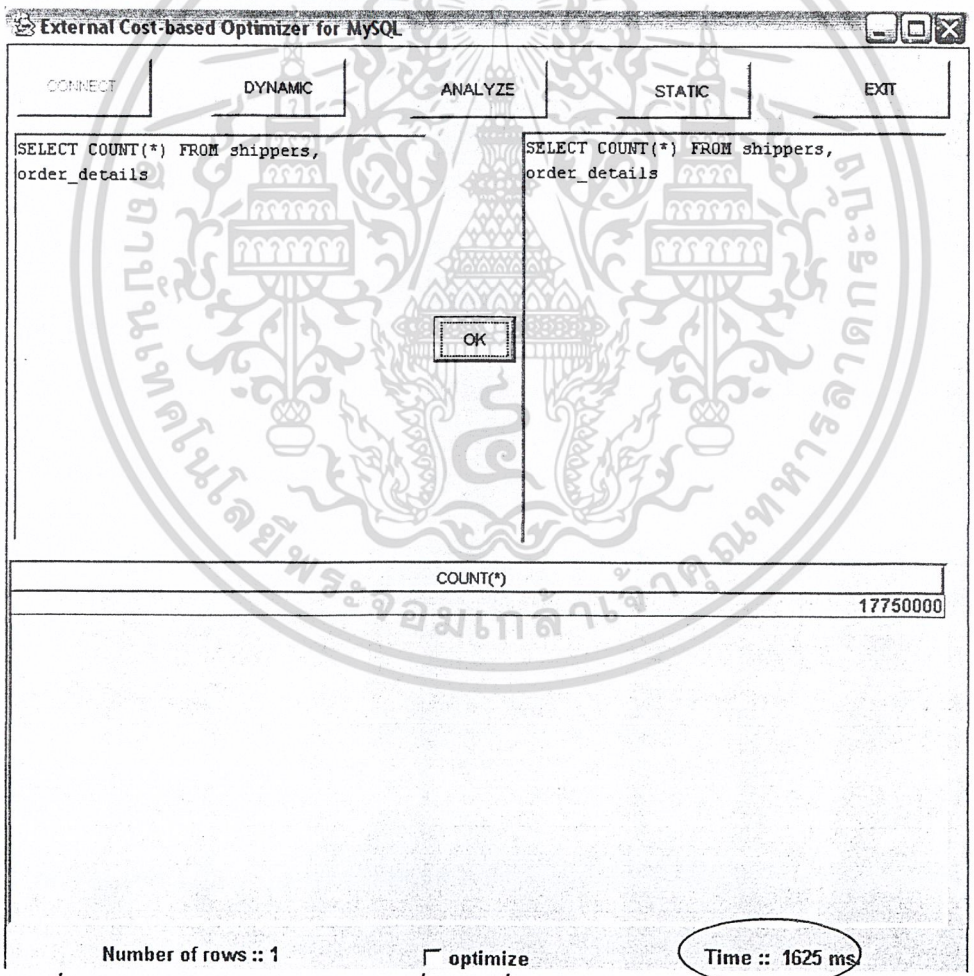
เมื่อทำการจับเวลาในการรันคำสั่งทั้ง 2 แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.50 1.52 1.52 1.55 1.55 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.41 1.41 1.39 1.41 1.39 (หน่วยเป็นวินาที)

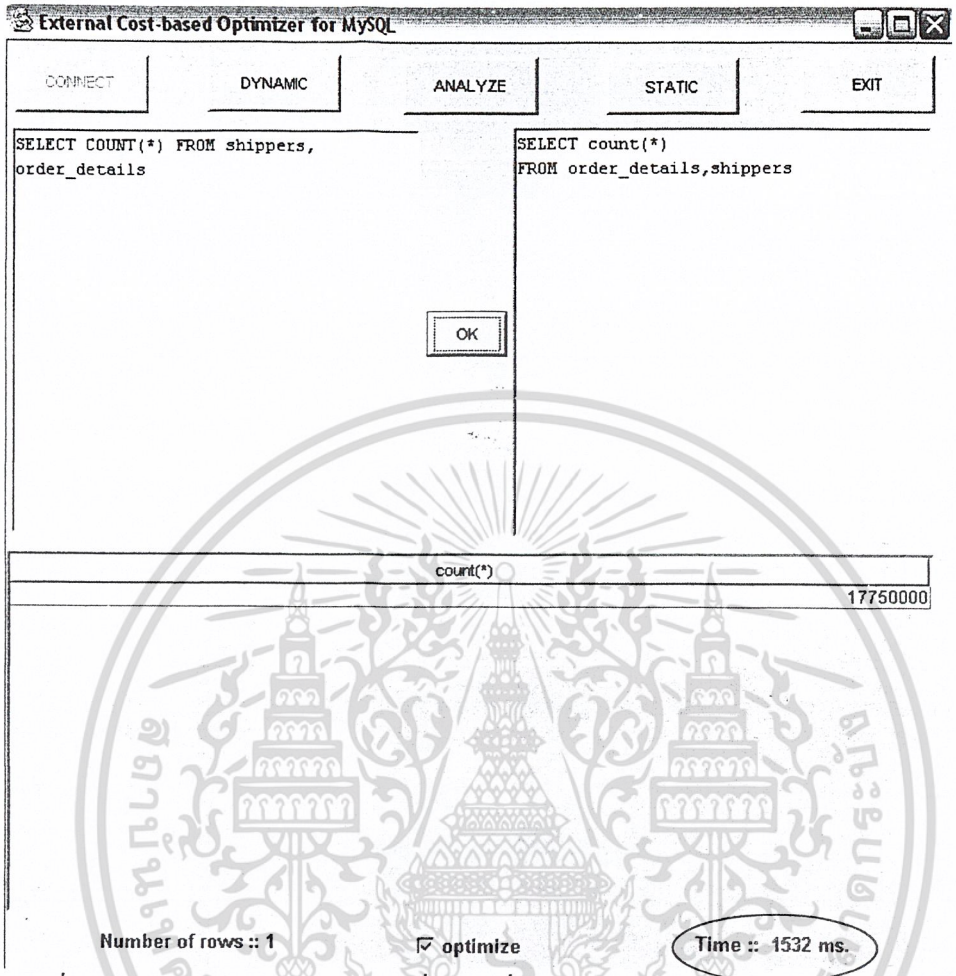
6.4.1.6 การทดลองกับโปรแกรมหลังจาก ANAYZE TABLE

จะได้ผลลัพธ์ของคำสั่งที่ยังไม่ผ่านการ optimize แล้ว ดังรูปที่ 6-5



รูปที่ 6-5 แสดงการทำงานของโปรแกรมเมื่อส่งคำสั่ง COUNT(*) หลังจาก ANALYZE TABLEF โดยไม่ผ่านการ optimize

จะได้ผลลัพธ์ดังรูปที่ 6-6



รูปที่ 6-6 แสดงการทำงานของโปรแกรมเมื่อส่งคำสั่ง COUNT(*) หลังจาก ANALYZE TABLE

6.4.2 การทดลองที่ 2

6.4.2.1 สมมติฐาน

การส่งคำสั่ง sql ที่มีการ query ที่ใช้ข้อมูลจากตารางเดียว และใน where นั้นมีการใช้คำสั่ง OR เพื่อหาค่าที่อยู่ในคอลัมน์เดียวกันหลายๆอัน ซึ่งคอลัมน์นี้ไม่ได้มีการกำหนด index เอาไว้ โดยเงื่อนไขใน OR นั้นเป็นการใช้ "=" จะสามารถใช้คำสั่ง IN เข้ามาแทนคำสั่ง OR ได้ และจะทำให้การประมวลผลรวดเร็วยิ่งขึ้น

6.4.2.2 การทดลองแผนในการประมวลผล

ทำโดยการใช้คำสั่ง EXPLAIN

คำสั่งแรก คือ

EXPLAIN SELECT * FROM test8.products

WHERE Unitsinstock = 132 OR Unitsinstock = 197 OR Unitsinstock = 140

OR UnitsInstock = 208 OR UnitsInstock = 1104 OR UnitsInstock = 58

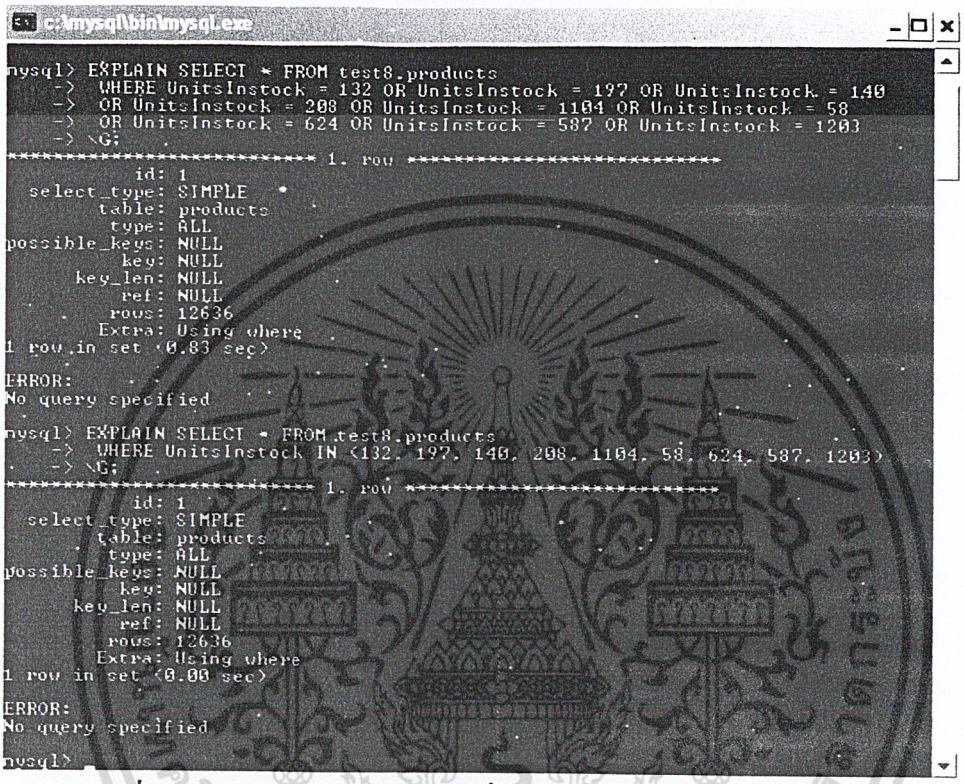
OR UnitsInstock = 624 OR UnitsInstock = 587 OR UnitsInstock = 1203;

และ

EXPLAIN SELECT * FROM test8.products

WHERE UnitsInstock IN (132, 197, 140, 208, 1104, 58, 624, 587, 1203);

ได้ผลลัพธ์ที่เหมือนกันดังรูปที่ 6-7



รูปที่ 6-7 แสดงการใช้ EXPLAIN กับคำสั่ง IN และ OR บนคอลัมน์ที่ไม่ใช่คีย์

6.4.2.3 การจับเวลาในการประมวลผลก่อน ANALYZE TABLE

โดยส่งคำสั่งต่อไปนี้เข้าไปรันที่ระบบจัดการฐานข้อมูล และใช้ข้อมูลในตาราง products ซึ่งอยู่ในฐานข้อมูลชื่อ test (190,000 rows)

SELECT * FROM test.products

WHERE UnitsInstock = 132 OR UnitsInstock = 197 OR UnitsInstock = 140

OR UnitsInstock = 208 OR UnitsInstock = 1104 OR UnitsInstock = 58

OR UnitsInstock = 624 OR UnitsInstock = 587 OR UnitsInstock = 1203;

และ

SELECT * FROM test.products

WHERE UnitsInstock IN (132, 197, 140, 208, 1104, 58, 624, 587, 1203);

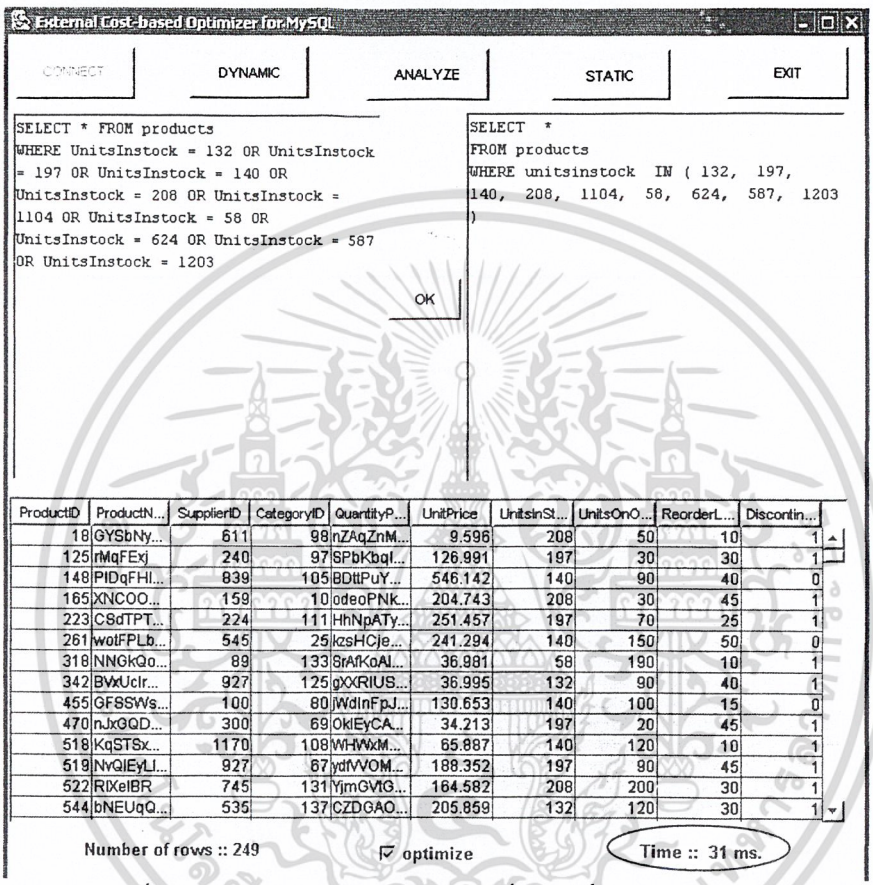
เมื่อทำการจับเวลาในการรันคำสั่งทั้ง 2 แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 3.28 0.89 0.86 1.08 0.91 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.77 0.81 0.80 0.77 0.78 (หน่วยเป็นวินาที)

6.4.2.4 การทดลองกับโปรแกรมก่อน ANALYZE TABLE

ได้ผลลัพธ์ดังรูปที่ 6-8



รูปที่ 6-8 แสดงการทำงานของโปรแกรมเมื่อใช้คำสั่ง IN(...)แทน OR

6.4.2.5 การจับเวลาในการประมวลผลหลังจาก ANAYZE TABLE

โดยส่งคำสั่งต่อไปนี้เข้าไปรันที่ระบบจัดการฐานข้อมูล

SELECT * FROM products WHERE UnitsInstock = 132

OR UnitsInstock = 197 OR UnitsInstock = 140 OR UnitsInstock = 208 OR UnitsInstock = 1104

OR UnitsInstock = 58 OR UnitsInstock = 624 OR UnitsInstock = 725 OR UnitsInstock = 750

OR UnitsInstock = 802 OR UnitsInstock = 830 OR UnitsInstock = 856 OR UnitsInstock = 894

OR UnitsInstock = 936 OR UnitsInstock = 1009 OR UnitsInstock = 1120 OR UnitsInstock = 1136

และ

SELECT * FROM products

WHERE UnitsInstock in (132,197,140,208,1104,58,624,725,750,802,830,856,894,

936,1009,1120,1136)

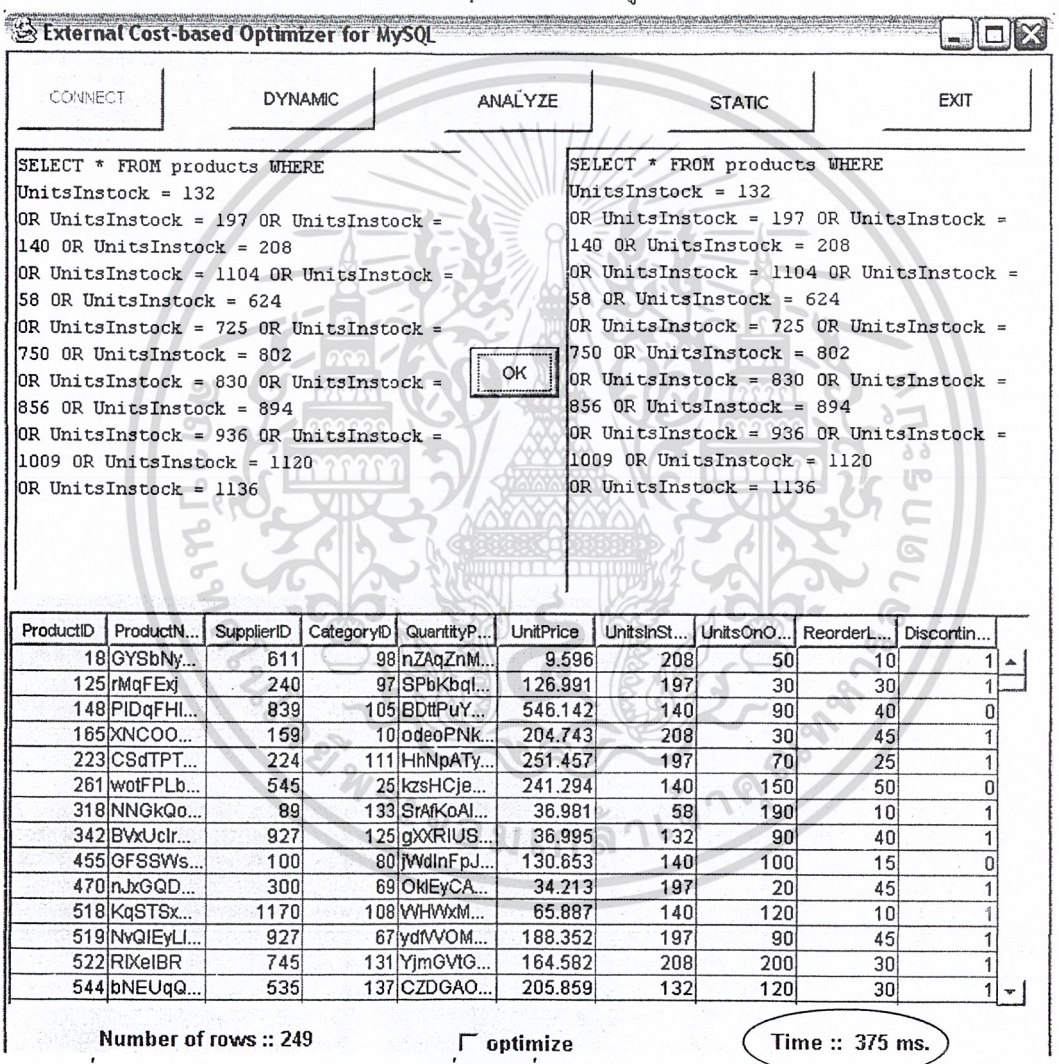
เมื่อทำการจับเวลาในการรันคำสั่งทั้ง 2 แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.17 0.05 0.03 0.05 0.03 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.02 0.02 0.03 0.02 (หน่วยเป็นวินาที)

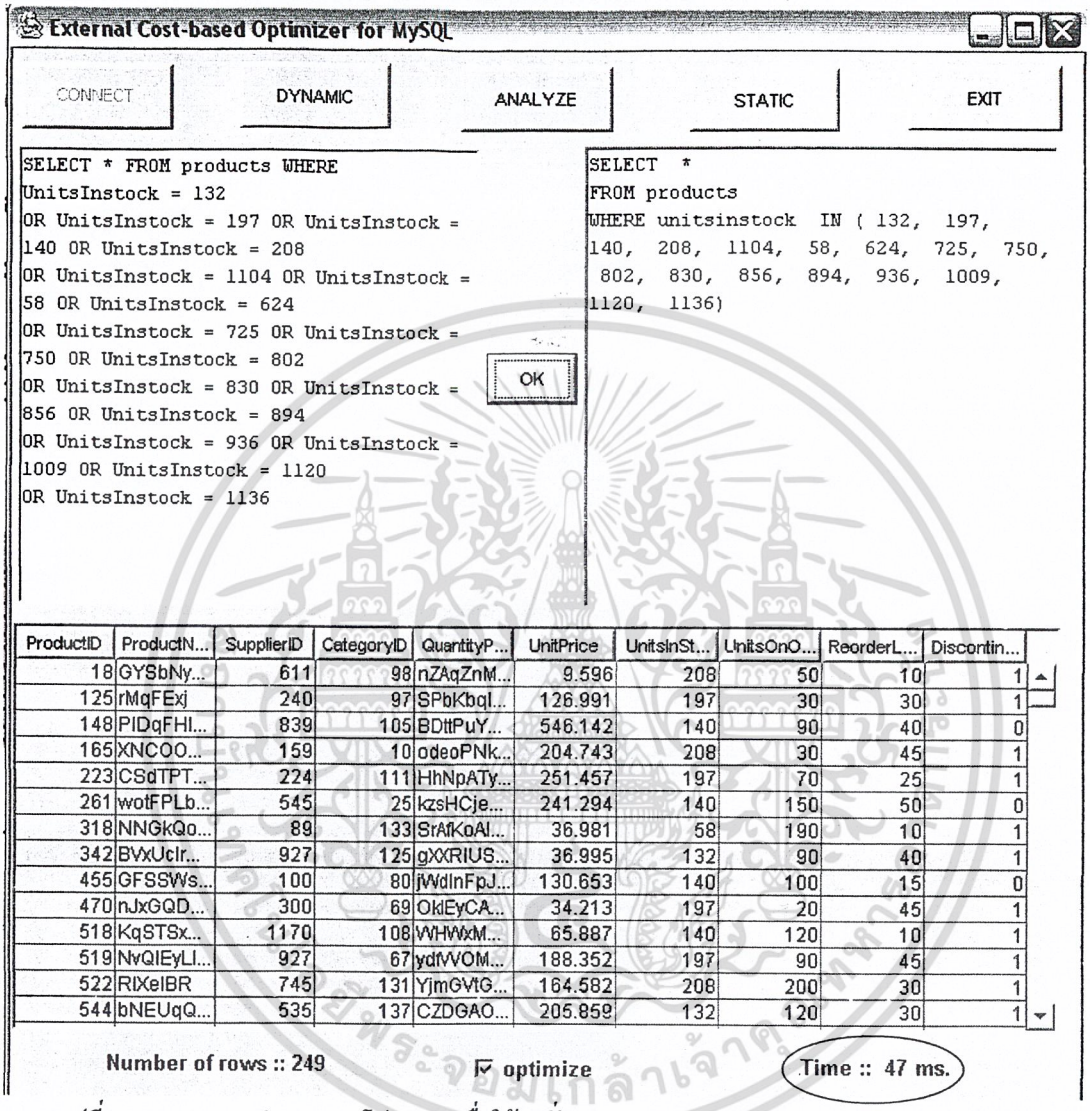
6.4.2.6 การทดลองกับโปรแกรมหลังจาก ANAYZE TABLE

จะได้ผลลัพธ์ของคำสั่งที่ยังไม่ผ่านการ optimize แล้ว ดังรูปที่ 6-9



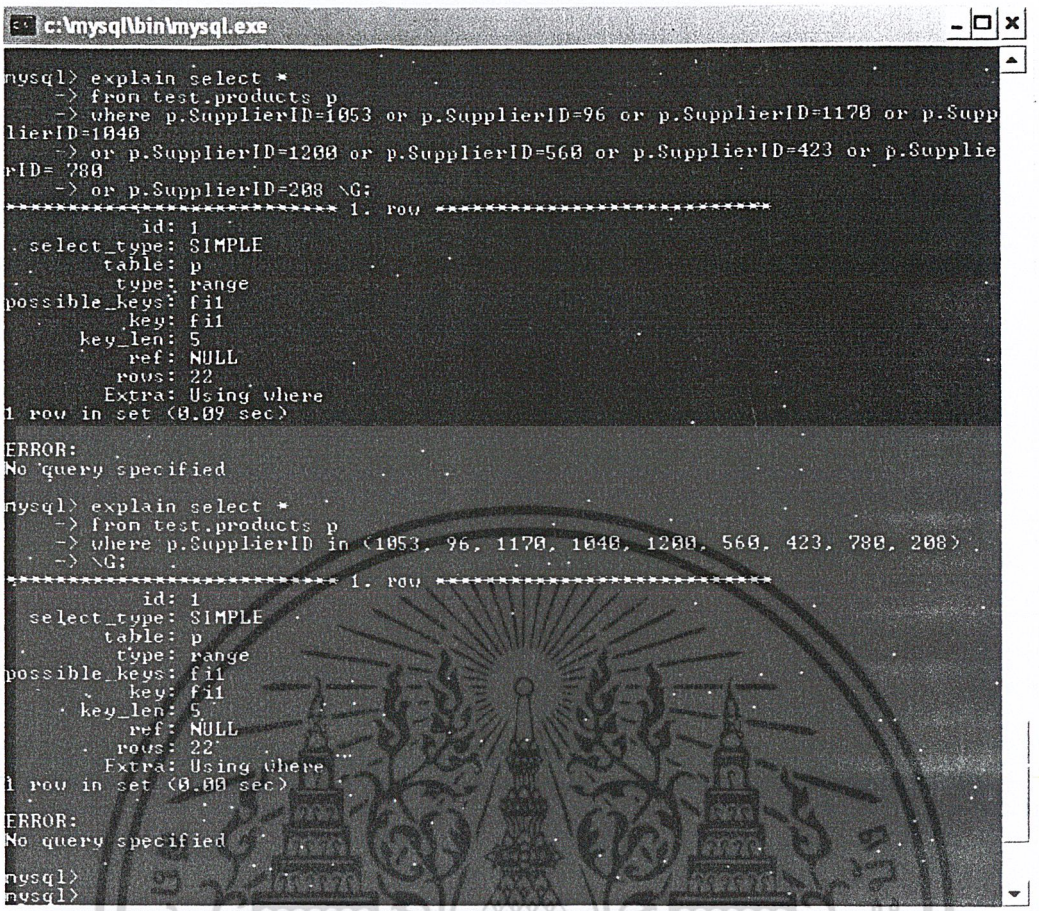
รูปที่ 6-9 แสดงการทำงานของโปรแกรมเมื่อใช้คำสั่ง IN(...)แทน OR หลังจาก ANALYZE TABLE โดยไม่มีการ optimize

และจะได้ผลลัพธ์ของการรันรูปแบบคำสั่งที่ผ่านการ optimize แล้ว ดังรูปที่ 6-10



รูปที่ 6-10 แสดงการทำงานของโปรแกรมเมื่อใช้คำสั่ง IN(...)แทน OR หลังจาก ANALYZE TABLE

สำหรับการใช้คำสั่ง OR บนคอลัมน์ที่มีการกำหนด index นั้น เมื่อทำการ EXPLAIN ดูจะได้ข้อมูลดังนี้



รูปที่ 6-11 แสดงคำสั่ง EXPLAIN การใช้ IN(...) และ OR บนคอลัมน์ที่มีดัชนี

ซึ่งในเวลาที่จับได้จากการรันคำสั่งทั้ง 2 นี้ไม่สามารถนำมาวิเคราะห์ได้ เนื่องจาก เวลาที่ใช้นั้นน้อยมากจนไม่เห็นความแตกต่าง (0.00 วินาที)

6.4.3 การทดลองที่ 3

6.4.3.1 สมมติฐาน

การส่งคำสั่ง sql ที่มีการรวมตารางตั้งแต่ 2 ตารางขึ้นไป ซึ่งใน where ไม่มีเงื่อนไขใดๆนั้น (การรวมตารางแบบคาร์ทีเชียน) MySQL จะทำการรวมตารางโดยเริ่มรวมจากตารางที่มีจำนวน row น้อยไปหาจำนวน row ที่มีจำนวนมาก ดังนั้นจึงมีการนำคำสั่ง STRAIGHT_JOIN เข้ามาบังคับลำดับในการรวมตารางให้มีลำดับจากน้อยไปมาก ทำให้ MySQL ไม่ต้องทำการประมวลผลในการหาแผนการในการรวมตาราง ซึ่งในการทดลองนี้ยังไม่มีการลองรันจริงจึงยังไม่มีการเปรียบเทียบเวลาของคำสั่งก่อน และหลังการ optimize

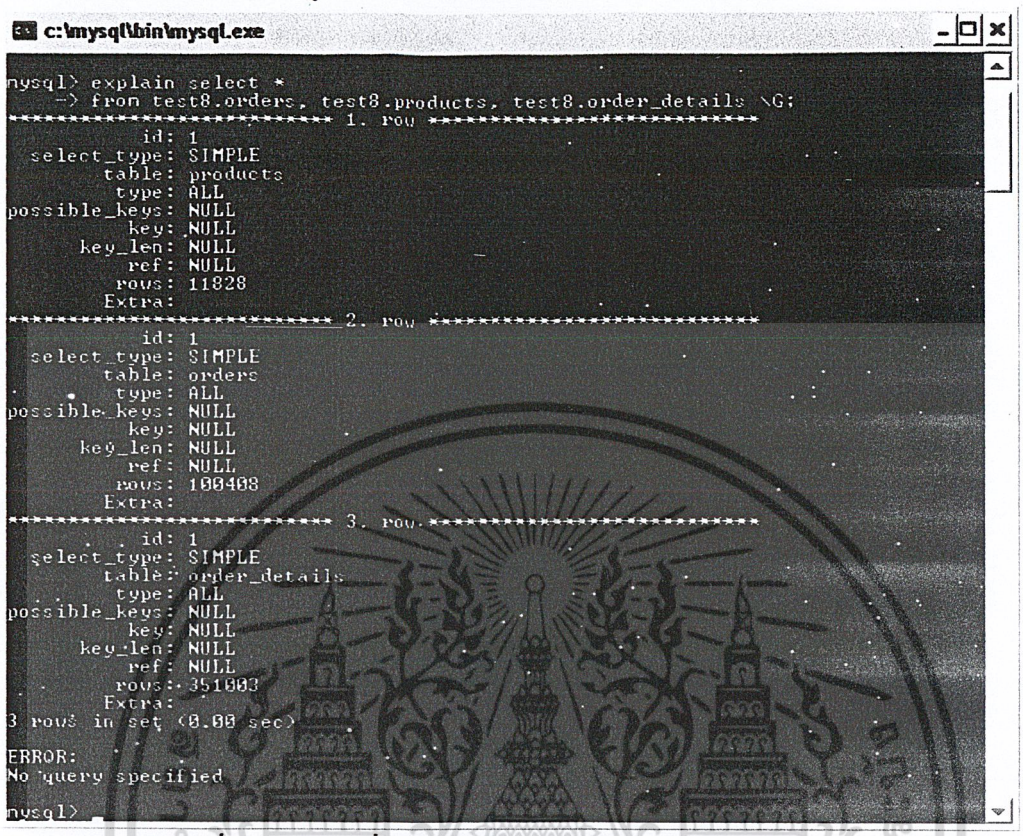
6.4.3.2 การทดลองแผนในการประมวลผล

ทำโดยการใช้คำสั่ง EXPLAIN ดังนี้คือ

EXPLAIN SELECT *

FROM test8.orders, test8.products, test8.order_details;

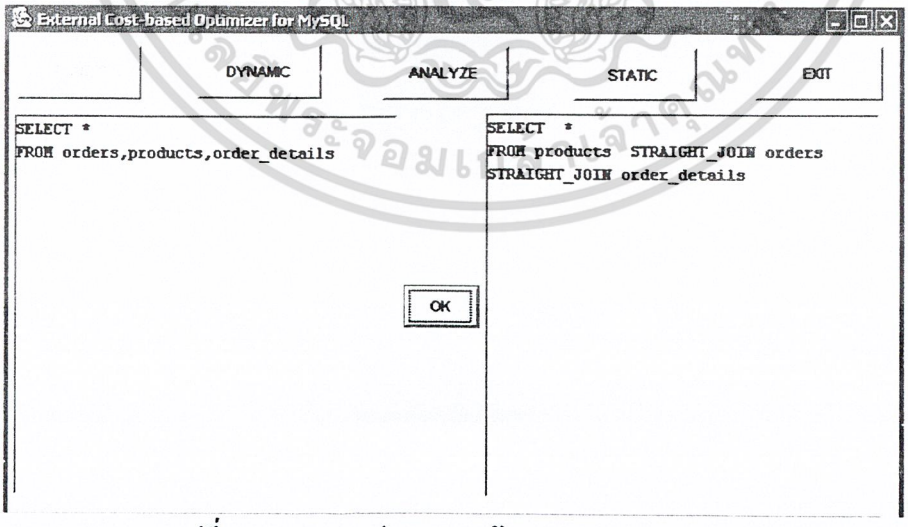
ได้ผลลัพธ์ที่เหมือนกันดังรูปที่ 6-12



รูปที่ 6-12 แสดงคำสั่ง EXPLAIN กับการรวมตารางแบบคาร์ทีเซียน

6.4.3.3 การทดลองกับโปรแกรม

ได้ผลลัพธ์ดังรูปที่ 6-13



รูปที่ 6-13 แสดงการประมวลผลด้วย STRAIGHT_JOIN

6.5 ผลการทดลองกับตารางประเภทอื่นๆ ของ MySQL

เป็นการทดลองโดยใช้ฐานข้อมูลเดียวกัน และวิธีการเดียวกันทั้งหมด แต่ได้ส่งคำสั่ง ALTER TABLE ซึ่ง syntax ของคำสั่งเป็นดังนี้

ALTER TABLE table_name TYPE = table_type

เช่น ALTER TABLE orders TYPE = MyISAM จะเป็นการเปลี่ยนตารางชื่อ orders ให้เป็นตาราง ประเภท MyISAM ซึ่งการเปลี่ยนประเภทของตารางให้เป็นประเภทต่างๆ ก็เพื่อทดลองหาประเภทตารางที่เหมาะสมที่สุดกับ query แต่ละแบบ ซึ่งถ้าสามารถเลือกประเภทของตารางให้เหมาะสมกับ query ที่ใช้เป็นประจำแล้ว จะช่วยให้เวลาในการประมวลผลคำสั่งรวดเร็วยิ่งขึ้น

6.5.1 การทดลองในกรณีของ COUNT(*)

เป็นกรณีที่มีการส่งคำสั่ง sql เพื่อ query หาจำนวนแถวของผลลัพธ์ที่มีการ join กันของตารางตั้งแต่ 2 ตารางขึ้นไป โดยไม่มีเงื่อนไขใน where เข้าไปรันที่ระบบจัดการฐานข้อมูล ซึ่งในการทดลองนี้ จะใช้ query ดังนี้

SELECT COUNT(*) FROM test8.shippers, test8.order_details;

และ

SELECT COUNT(*) FROM test8.order_details, test8.shippers;

ซึ่งคำสั่งที่ 2 จะอยู่ในรูปแบบที่ผ่านการ optimize เรียบร้อยแล้ว โดยจะได้ผลการทดลองดังต่อไปนี้

6.5.1.1 ผลการทดลองกับ MyISAM

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.0223 0.0300 0.0179 0.0017 0.0216 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.0018 0.0218 0.0169 0.0017 0.0017 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.0763 0.0064 0.0243 0.0138 0.0261 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.0226 0.0362 0.0017 0.0016 0.0310 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก คือ

EXPLAIN SELECT COUNT(*) FROM test8.shippers, test8.order_details;

จะได้ผลลัพธ์ คือ 'SIMPLE', '', '', '', '', '', 'Select tables optimized away'

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง คือ

EXPLAIN SELECT COUNT(*) FROM test8.order_details, test8.shippers;

จะได้ผลลัพธ์ คือ 'SIMPLE', '', '', '', '', '', 'Select tables optimized away' ซึ่งจะเหมือนกับผลลัพธ์ของคำสั่งแรก เนื่องจากในตารางประเภท MyISAM นั้น การประมวลผล COUNT(*) ของตาราง โดยไม่มี WHERE จะถูกนำมาโดยตรงจากตารางข้อมูลของ MyISAM เลย ดังนั้นจึงไม่ต้องมีการ optimize หรือการทำงานกับฐานข้อมูลจริงๆ

6.5.1.2 ผลการทดลองกับ ISAM

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.63 1.56 1.53 1.58 1.58 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.44 1.49 1.45 1.52 1.45 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 2.34 2.33 2.25 2.25 2.27 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.61 1.59 1.56 1.58 1.73 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

พบว่ายังคงมีแผนในการประมวลผลเหมือนกับตารางประเภท InnoDB

6.5.1.3 ผลการทดลองกับ BDB

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.59 1.70 1.63 1.64 1.61 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.44 1.42 1.44 1.44 1.45 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.84 1.67 1.67 1.69 1.78 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.53 1.45 1.49 1.44 1.20 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

พบว่ายังคงมีแผนในการประมวลผลเหมือนกับตารางประเภท InnoDB

6.5.1.4 ผลการทดลองกับ NDB

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.92 1.81 1.89 1.86 1.89 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.58 1.58 1.58 1.58 1.56 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 1.67 1.69 1.69 1.72 1.63 (หน่วยเป็นวินาที)

คำสั่งสอง : 1.52 1.53 1.45 1.52 1.52 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

พบว่ายังคงมีแผนในการประมวลผลเหมือนกับตารางประเภท InnoDB

6.5.2 การทดลองในกรณีของ IN(...) แทนที่ OR

เป็นกรณีที่มีการส่งคำสั่ง sql ที่มีการ query ที่ใช้ข้อมูลจากตารางเดียว และใน where นั้นมีการใช้คำสั่ง OR เพื่อหาค่าที่อยู่ในคอลัมน์เดียวกันหลายๆอัน ซึ่งคอลัมน์นี้ไม่ได้มีการกำหนด index เอาไว้ เข้าไปรันที่ระบบจัดการฐานข้อมูล ซึ่งในการทดลองนี้ จะใช้ query ดังนี้

```
SELECT * FROM test8.products
WHERE UnitsInstock = 132 OR UnitsInstock = 197 OR UnitsInstock = 140
OR UnitsInstock = 208 OR UnitsInstock = 1104 OR UnitsInstock = 58
OR UnitsInstock = 624 OR UnitsInstock = 587 OR UnitsInstock = 1203;
```

และ

```
SELECT * FROM test8.products
WHERE UnitsInstock IN (132, 197, 140, 208, 1104, 58, 624, 587, 1203);
```

ซึ่งคำสั่งที่ 2 จะอยู่ในรูปแบบที่ผ่านการ optimize เรียบร้อยแล้ว โดยจะได้ผลการทดลองดังต่อไปนี้

6.5.2.1 ผลการทดลองกับ MyISAM

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.06 0.05 0.06 0.05 0.05 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.01 0.03 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.05 0.05 0.03 0.05 0.05 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.02 0.03 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับทั้งสองคำสั่งจะ ได้ผลลัพธ์เหมือนกัน และเหมือนกับตารางประเภท InnoDB ด้วย นั่นคือ

1, 'SIMPLE', 'products', 'ALL', '', '', , '', 12400, 'Using where'

6.5.2.2 ผลการทดลองกับ ISAM

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.05 0.03 0.05 0.05 0.03 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.01 0.03 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.03 0.03 0.05 0.05 0.05 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.03 0.02 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับทั้งสองคำสั่งจะได้ผลลัพธ์เหมือนกัน และเหมือนกับตารางประเภท InnoDB ด้วย นั่นคือ

1, 'SIMPLE', 'products', 'ALL', '', '', '', 12400, 'Using where'

6.5.2.3 ผลการทดลองกับ BDB

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก :0.17 0.03 0.03 0.0188 0.0147 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.0204 0.0160 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.05 0.05 0.05 0.05 0.05 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.01 0.02 0.03 0.03 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับทั้งสองคำสั่งจะได้ผลลัพธ์เหมือนกัน และเหมือนกับตารางประเภท InnoDB ด้วย นั่นคือ

1, 'SIMPLE', 'products', 'ALL', '', '', '', 12400, 'Using where'

6.5.2.4 ผลการทดลองกับ NDB

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองก่อนสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.19 0.03 0.03 0.03 0.03 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.02 0.03 (หน่วยเป็นวินาที)

เมื่อทำการจับเวลาในการรันคำสั่งทั้งสองหลังจากสั่ง ANALYZE TABLE แล้ว ได้ค่าออกมาดังนี้

คำสั่งแรก : 0.03 0.05 0.05 0.05 0.03 (หน่วยเป็นวินาที)

คำสั่งสอง : 0.03 0.03 0.03 0.03 0.03 (หน่วยเป็นวินาที)

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'order_details', 'index', '', 'fi0' , 4, '', 355176, 'Using index'

1, 'SIMPLE', 'shippers' , 'index', '', 'PRIMARY', 4, '', 50 , 'Using index'

พบว่ายังคงมีแผนในการประมวลผลเหมือนกับตารางประเภท InnoDB

6.5.3 การทดลองในกรณีของ Catesian Product

เป็นกรณีที่มีการส่งคำสั่ง sql ที่มีการรวมตารางตั้งแต่ 2 ตารางขึ้นไป ซึ่งใน where ไม่มีเงื่อนไขใดๆ เลย (การรวมตารางแบบคาร์ทีเชียน) ซึ่งในการทดลองนี้ จะใช้ query ดังนี้

SELECT * FROM test8.orders, test8.products, test8.order_details;

และ

SELECT * FROM test8.orders, test8.order_details, test8.products;

โดยตาราง products, orders และ order_details มีจำนวนแถวเท่ากับ 12400, 100000 และ 355176 แถว ตามลำดับ โดยจะได้ผลการทดลองดังต่อไปนี้

6.5.3.1 ผลการทดลองกับ MyISAM, ISAM, BDB และ NDB

จากการทดลองกับตารางประเภทต่างๆ ของ MYSQL กับกรณีนี้นั้น จะได้แผนการในการประมวลผลเดียวกัน นั่นคือ

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งแรก จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'products' , 'ALL', '', '', 12400 , ''

1, 'SIMPLE', 'orders' , 'ALL', '', '', 100000, ''

1, 'SIMPLE', 'order_details', 'ALL', '', '', 355000, ''

เมื่อใช้คำสั่ง EXPLAIN กับคำสั่งที่สอง จะได้ผลลัพธ์ คือ

1, 'SIMPLE', 'products' , 'ALL', '', '', 12400 , ''

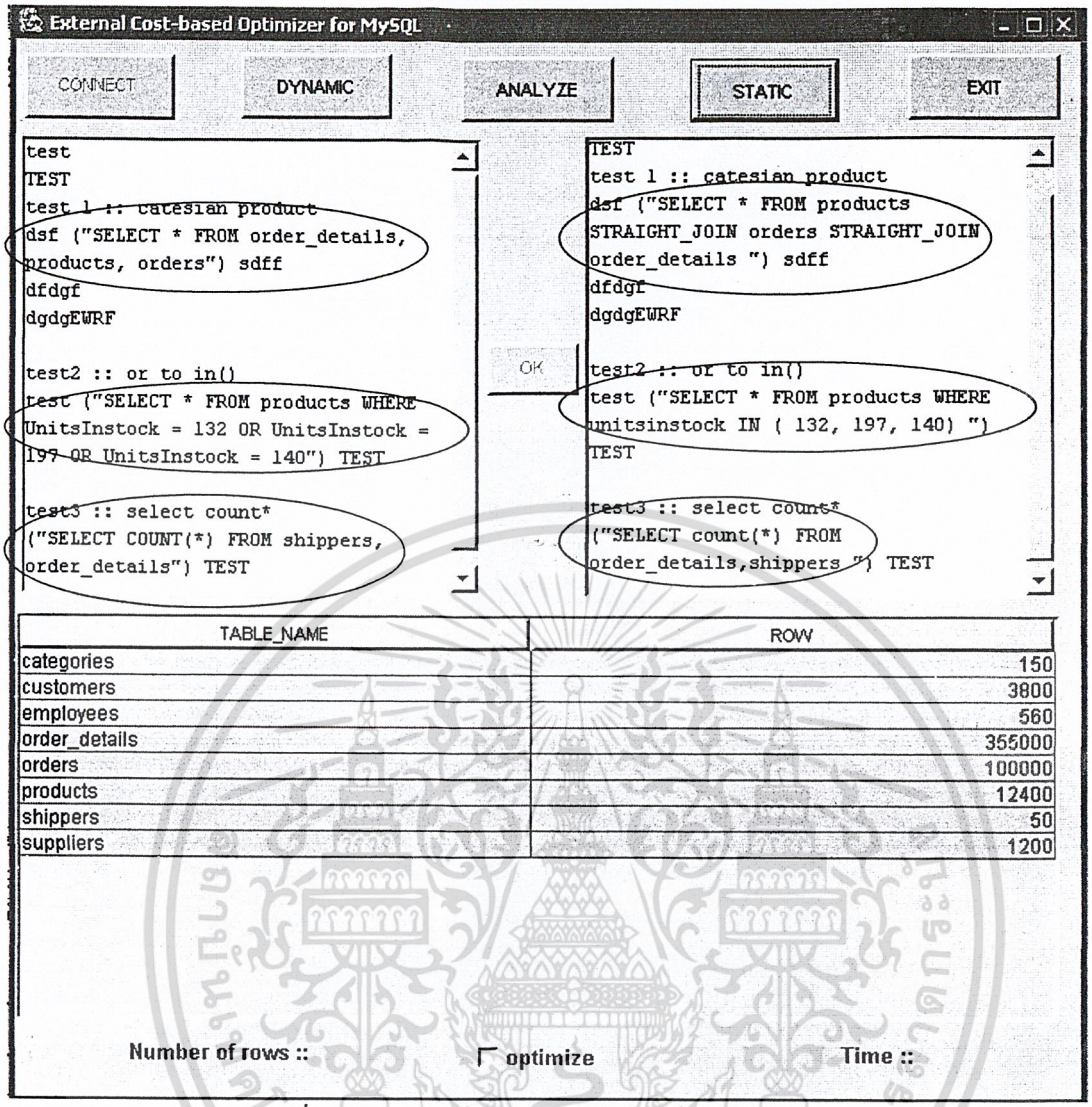
1, 'SIMPLE', 'orders' , 'ALL', '', '', 100000, ''

1, 'SIMPLE', 'order_details', 'ALL', '', '', 355000, ''

พบว่ายังคงมีแผนในการประมวลผลเหมือนกันทั้งสองคำสั่ง และเหมือนกันกับตารางประเภท InnoDB ด้วย

6.6 โหมดในการสแกนไฟล์

เป็นการนำการประมวลผลทั้งสามวิธีที่ได้ทดลองไปแล้วมาใช้ในการประมวลผลไฟล์ของผู้ใช้ โดยโปรแกรมจะสแกนไฟล์ของผู้ใช้ เพื่อหาคำสั่ง sql ที่จะ query หาผลลัพธ์จากฐานข้อมูล แล้วทำการประมวลผลตามวิธีทั้งสาม โดยผลลัพธ์ในการทำงานของโปรแกรมจะเป็นดังรูป 6-12 โดยฟิลด์ข้อความฝั่งซ้ายจะแสดงข้อความในไฟล์ของผู้ใช้ก่อนการประมวลผล และฟิลด์ข้อความฝั่งขวาจะแสดงข้อความในไฟล์หลังจากทำการประมวลผลแล้ว



รูปที่ 6-14 แสดงการประมวลผลด้วยวิธีสแกนไฟล์ของผู้ใช้

6.7 สรุปผลการทดลอง

6.7.1 สำหรับตารางประเภท InnoDB, MyISAM, ISAM, BDB และ NDB

1. เมื่อต้องการส่งคำสั่ง sql ที่มีการ query ที่ใช้ข้อมูลจากตารางเดียว และใน where นั้นมีการใช้คำสั่ง OR เพื่อหาค่าที่อยู่ในคอลัมน์เดียวกันหลายๆอัน ซึ่งคอลัมน์นี้ไม่ได้มีการกำหนด index เอาไว้ โดยเงื่อนไขใน OR นั้นเป็นการใช้ "=" จะสามารถใช้คำสั่ง IN เข้ามาแทนคำสั่ง OR ได้ และจะทำให้การประมวลผลรวดเร็วยิ่งขึ้น เพราะมีความยาวของ query น้อยเท่าไร ก็ยิ่งเป็นการลดเวลาในการวิเคราะห์ และแปลประโยคในส่วนของ Parser และ Translator มากขึ้นเท่านั้น ซึ่งช่วยลดเวลาโดยรวมในการประมวลผลได้

2. เมื่อต้องการส่งคำสั่ง sql ที่มีการรวมตารางตั้งแต่ 2 ตารางขึ้นไป ซึ่งใน where ไม่มีเงื่อนไขใดๆนั้น (การรวมตารางแบบคาร์ทีเชียน) MySQL จะทำการรวมตารางโดยเริ่มรวมจากตารางที่มีจำนวน row น้อยไปหาจำนวน row ที่มีจำนวนมาก

6.7.2 สำหรับตารางประเภท InnoDB, ISAM, BDB และ NDB

เมื่อต้องการคำสั่ง sql ที่มีการ query จำนวนแถวของผลลัพธ์ หรือ COUNT(*) ที่มีการ join กันของตารางตั้งแต่ 2 ตารางขึ้นไป โดยไม่มีเงื่อนไขใน where เข้าไปรันที่ระบบจัดการฐานข้อมูล โดยให้ตารางที่มีจำนวน row ทั้งหมดน้อยที่สุดอยู่ทางด้านขวามือสุดแล้ว จะทำให้การระบบจัดการฐานข้อมูลทำการประมวลผลได้รวดเร็วยิ่งขึ้น

6.7.3 สำหรับตารางประเภท MyISAM

เมื่อต้องการ query หาแถวของผลลัพธ์ หรือ COUNT(*) โดยไม่มีเงื่อนไขใน WHERE แล้ว ตารางประเภทนี้จะนำค่าจำนวนแถวมาโดยตรงจากตารางข้อมูลของ MyISAM ทำให้ไม่ต้องเข้าถึงในส่วนของฐานข้อมูลจริงๆ ซึ่งทำให้ MySQL สามารถประมวลผลได้รวดเร็วขึ้นมาก

บทที่ 7
บทสรุปและวิจารณ์

7.1 สรุปผลการดำเนินงาน

จากการศึกษาและได้ทำการทดลองต่างๆ ทำให้ทราบถึงกฎพื้นฐานในการทำงานของ optimizer ของ MySQL และได้นำกฎเหล่านั้นมาทำการพัฒนาต่อเป็น Optimizer ภายนอกที่จะทำการเปลี่ยนแปลง query ของผู้ใช้ให้อยู่ในรูปแบบที่เหมาะสมก่อนที่จะส่งไปให้กับ DBMS เพื่อให้ MySQL ใช้เวลากับการประมวลผล query ของผู้ใช้ให้น้อยที่สุด ซึ่งจะส่งผลให้ผู้ใช้สามารถทำงานต่างๆ เกี่ยวกับฐานข้อมูลได้รวดเร็วยิ่งขึ้น โดยโปรแกรมที่พัฒนาขึ้นนั้นสามารถรับคำสั่ง sql ใดๆ จากผู้ใช้ แล้วจะนำไปตรวจสอบว่าตรงกับกฎที่กำหนดไว้หรือไม่ ซึ่งถ้าตรงกับกฎแล้ว จะทำการเปลี่ยนรูปแบบของคำสั่งให้เหมาะสม แล้วจะส่งคำสั่งนั้นเข้าไปทำงานในระบบจัดการฐานข้อมูล เพื่อหาผลลัพธ์ของ query นั้น นอกจากนั้นยังสามารถสแกนไฟล์ของผู้ใช้เพื่อหาคำสั่ง sql แล้วเปลี่ยนคำสั่ง sql ในไฟล์นั้นๆ ให้โดยที่ไม่กระทบกระเทือนส่วนอื่นของข้อมูลในไฟล์ ดังนั้นเมื่อผู้ใช้นำไฟล์ที่ได้มีการเปลี่ยนแปลง sql ให้เหมาะสมแล้วไปใช้งาน ก็จะเป็นการช่วยลดเวลาในส่วนที่ต้องติดต่อกับฐานข้อมูลให้ด้วย

7.2 แนวทางในการพัฒนาต่อ

จากโปรแกรมที่ได้พัฒนาขึ้นมานั้น เปรียบเหมือนเป็นตัวต้นแบบของตัวประมวลผลภายนอกเท่านั้น ซึ่งการประมวลผลต่างๆในโปรแกรมนั้น ทำการวิเคราะห์ได้เพียงบาง query เท่านั้น และเป็น query ที่ยังไม่มีความซับซ้อนมากนัก และการประมวลผลหา query ออกมานั้น ยังอิงอยู่กับกฎของระบบจัดการฐานข้อมูลทั้งสิ้น สำหรับส่วนการเก็บสถิตินั้น ยังมีการเก็บค่าสถิติเพียงบางค่า ที่จำเป็นต้องใช้กับการประมวลผลในโปรแกรมนี้เท่านั้น ซึ่งค่าที่เก็บมานั้นนำมาใช้เพียงแค่เปรียบเทียบ และทำการพิจารณาเท่านั้น ยังไม่มีการคำนวณทางสถิติที่ซับซ้อน

เพราะฉะนั้น แนวทางการพัฒนาต่อไปของระบบนี้ คือ การศึกษาและทำการทดลองการประมวลผลของ query ที่ซับซ้อนมากขึ้น เช่น การทำ subquery เป็นต้น เพื่อนำมาวิเคราะห์และทำการปรับปรุงวิธีการในส่วนการประมวลผลให้มีประสิทธิภาพมากขึ้น สำหรับส่วนของการเก็บสถิตินั้น ถ้ามีการพัฒนาต่อไป ควรจะมีการเก็บค่าต่างๆที่จำเป็นในการประมวลผลแบบอิงสถิติให้มากกว่านี้ เพื่อนำมาคำนวณค่า cost ที่ใช้ในการทำ query และพิจารณาเลือกเส้นทางที่ดีที่สุดออกมาได้อย่างมีประสิทธิภาพ นอกจากนั้นยังสามารถพัฒนาในส่วนการนำโปรแกรมไปใช้งานได้อีก โดยทำให้เป็นแบบ embedded ซึ่งจะสะดวกสบายต่อผู้ใช้ และตรงกับลักษณะในการทำงานจริงของผู้ใช้มากกว่า

ภาคผนวก ก.
การติดตั้งระบบจัดการฐานข้อมูล MySQL

1. ดาวน์โหลดแพ็กเกจของ MySQL server จาก http://dev.mysql.com/downloads/
2. ทำการ extract แพ็กเกจของ MySQL server ออกมา
3. ติดตั้งโดยดับเบิลคลิ๊กที่ไฟล์ setup.exe หรือ ดับเบิลคลิ๊กที่ไฟล์ .msi
4. ทำตามขั้นตอนที่ขึ้นมาเรื่อยๆ จนจบการติดตั้ง

ภาคผนวก ข.
การติดต่อกับฐานข้อมูลโดยใช้ JDBC

Java Database Connectivity(JDBC)

เป็น API ที่ช่วยอำนวยความสะดวกในการติดต่อกับฐานข้อมูลต่างๆ อาจกล่าวได้ว่า JDBC นั้นเป็นกลุ่มของคลาสที่ใช้ในการติดต่อกับฐานข้อมูล ดังนั้นเมื่อต้องการติดต่อฐานข้อมูลด้วยภาษา Java จึงต้องมีการเรียก ไดร์ฟเวอร์ของ JDBC ด้วย

ในระบบจัดการฐานข้อมูลแต่ละตัวนั้น จะมีไดร์ฟเวอร์ของตัวเอง ดังนั้นหากต้องการติดต่อกับระบบจัดการฐานข้อมูลตัวใด จะต้องติดตั้ง JDBC ไดร์ฟเวอร์ ของระบบจัดการฐานข้อมูลนั้นลงไปด้วย

การติดต่อกับฐานข้อมูลโดยใช้ JDBC

สามารถแบ่งออกได้เป็น 4 ขั้นตอน ดังนี้

1. โหลดคลาสไดร์ฟเวอร์ของ JDBC

การโหลดไดร์ฟเวอร์ของ JDBC ในโปรแกรม JBuilder นั้นจะต้องเข้าไปเพิ่ม library โดยทำตามขั้นตอนดังนี้

1.1 คลิ๊กที่ tool -> Configure Libraries แล้วจะมี dialog ขึ้นมาให้

1.2 คลิ๊กที่ปุ่ม New

1.3 เมื่อมี dialog ขึ้นมา ให้ทำการตั้งชื่อ แล้วคลิ๊กที่ปุ่ม Add

1.4 ทำการระบุ path ที่เก็บไฟล์ไดร์ฟเวอร์นั้นอยู่ เมื่อทำการเลือกแล้วให้คลิ๊กปุ่ม OK ก็จะจบขั้นตอน

2. เปิดการเชื่อมต่อไปยังฐานข้อมูล

ส่วนนี้จะอยู่ในโค้ดของโปรแกรม โดยจะต้องระบุถึงตัวไดร์ฟเวอร์ของฐานข้อมูล และระบุตำแหน่งของฐานข้อมูล โดยใช้ 2 คำสั่ง ดังตัวอย่างต่อไปนี้

```
Class.forName(MysqlDriver);
connection = DriverManager.getConnection(url,user,password);
```

โดยมีตัวแปรต่างๆดังนี้

2.1 MysqlDriver จะเป็นสตริงที่ใช้กำหนดตัวไดร์ฟเวอร์ของฐานข้อมูลที่ต้องการติดต่อ สำหรับ MySQL นั้น MysqlDriver = “com.mysql.jdbc.Driver”

2.2 url เป็นสตริงที่ใช้แสดงตำแหน่งของฐานข้อมูลที่ต้องการติดต่อ สำหรับตัวอย่างนี้ url = “jdbc:mysql://127.0.0.1:3306/test” โดย test นั้นคือชื่อของฐานข้อมูลที่ทำการติดต่อ

2.3 user เป็นสตริงที่ระบุชื่อของผู้ใช้

2.4 password เป็นสตริงที่ระบุรหัสผ่านของผู้ใช้

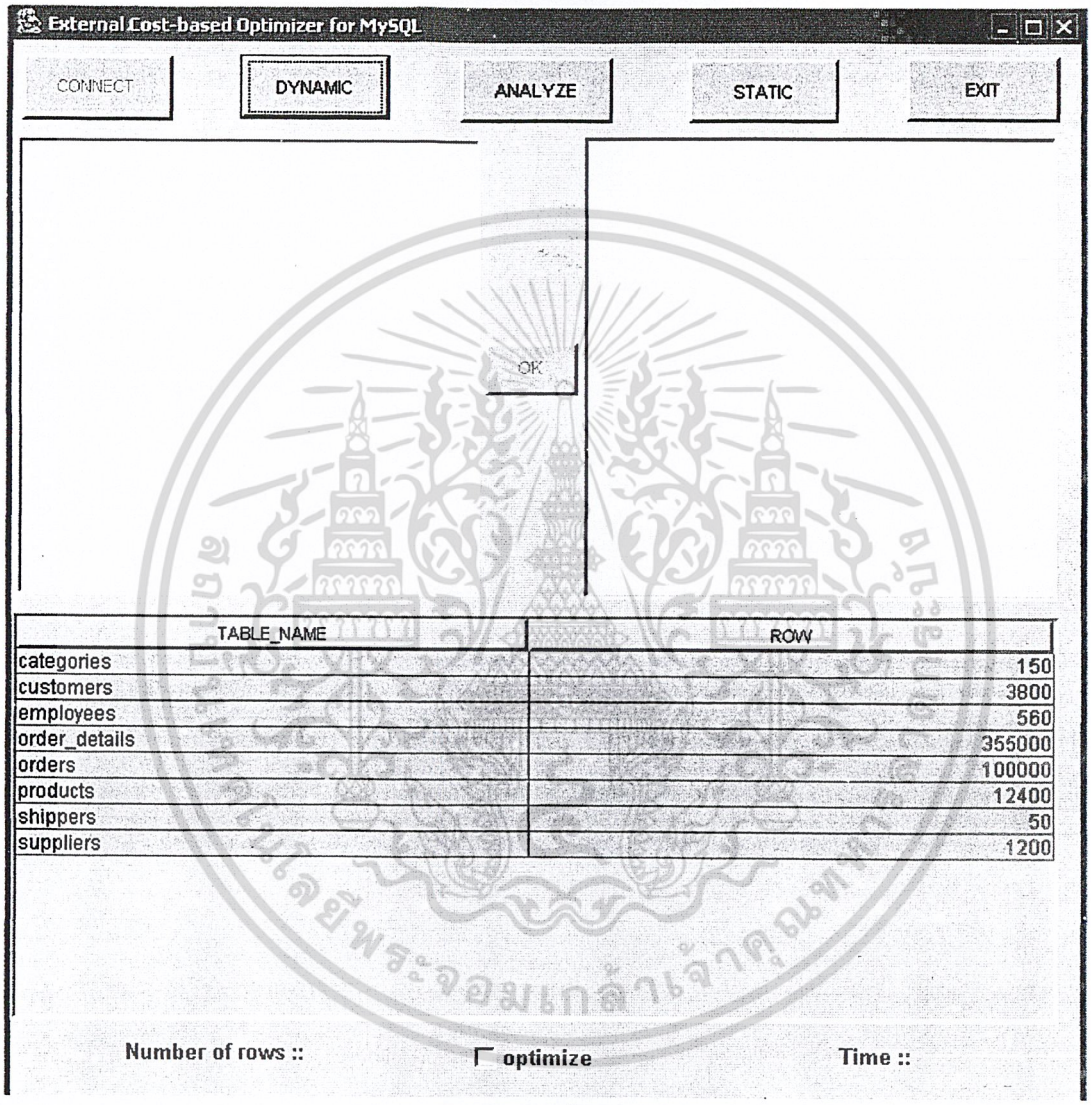
3. ติดต่อกับฐานข้อมูลโดยใช้คำสั่ง SQL

4. จัดการเกี่ยวกับผลลัพธ์ที่ได้มาจากคำสั่ง SQL

ภาคผนวก ค.

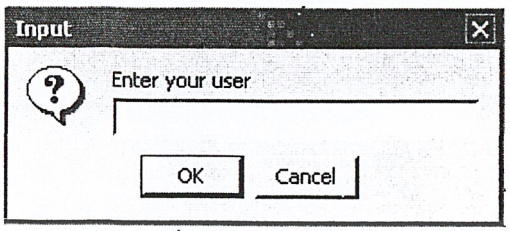
คู่มือการใช้งานตัวประมวลผลภายนอกแบบอิงสถิติบน MySQL

1. เมื่อทำการรันโปรแกรม ผู้ใช้โปนแกรมจะเห็นหน้าจออินเทอร์เฟซ ดังรูป



รูปแสดงหน้าจออินเทอร์เฟซของโปรแกรม

2. ผู้ใช้ต้องทำการติดต่อกับฐานข้อมูล โดยทำการคลิ๊กที่ปุ่ม CONNECT จากนั้นจะมี dialog ขึ้นมา ซึ่งผู้ใช้ต้องทำการกรอกข้อมูล username และ password เพื่อเป็นการตรวจสอบบุคคลที่จะเข้าถึงในส่วนของฐานข้อมูลนั้น



รูปแสดง dialog ที่ให้ผู้ใช้โปรแกรมกรอก username

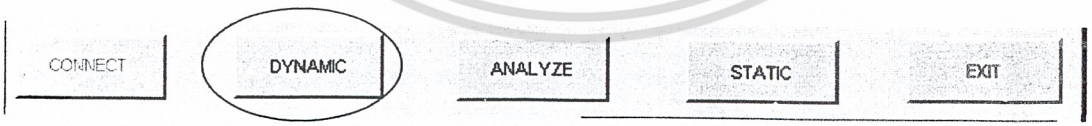
ซึ่งเมื่อทำการ log in เรียบร้อยแล้ว ตารางด้านล่างของอินเทอร์เฟซจะแสดงจำนวนแถวของแต่ละตารางในฐานข้อมูลที่กำลังติดต่ออยู่ ดังรูป

TABLE_NAME	ROW
categories	150
customers	3800
employees	560
orders	35500
order_details	100000
products	12400
shippers	50
suppliers	1200

รูปแสดงการแสดงจำนวนแถวของตารางในฐานข้อมูล

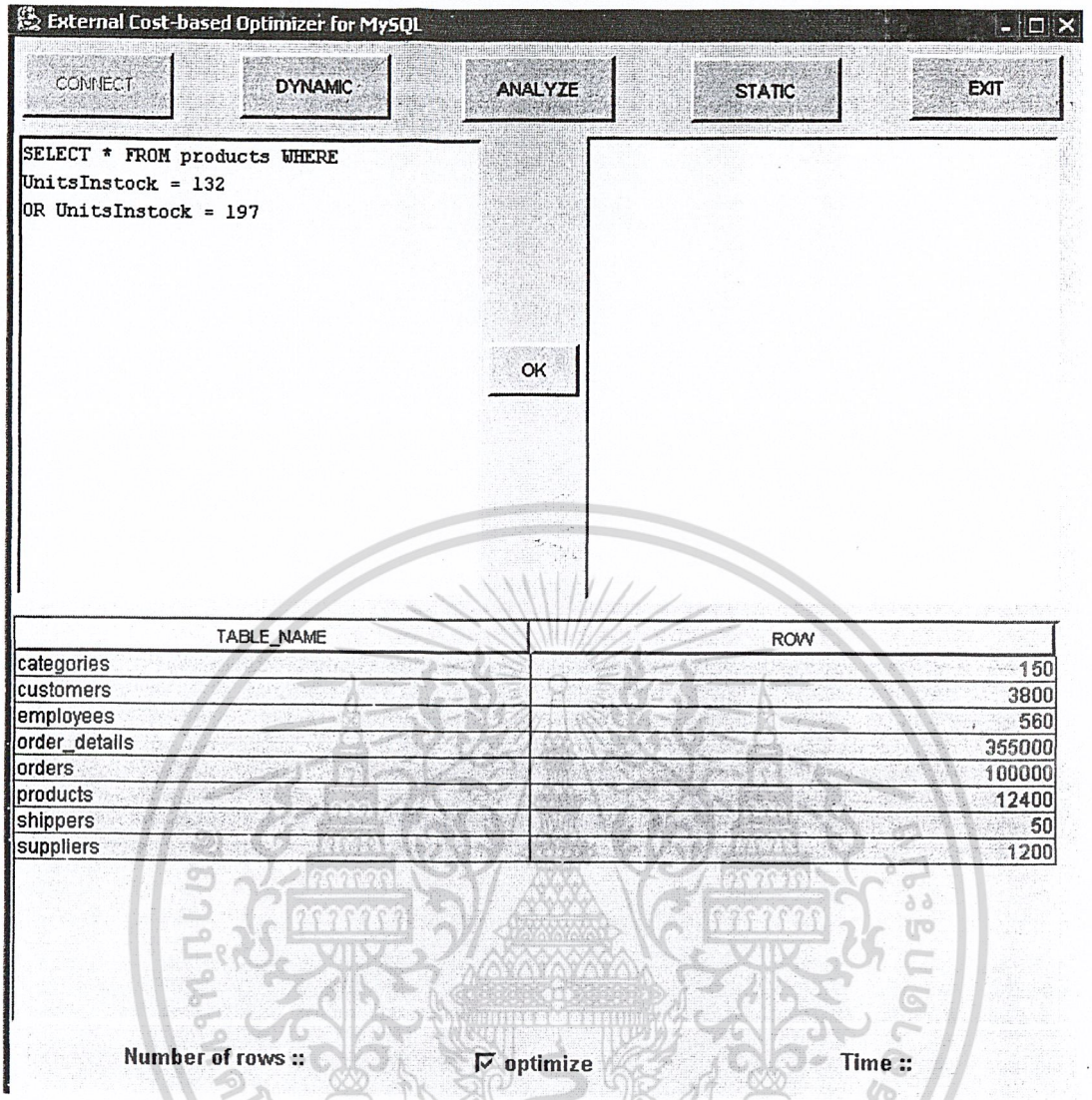
3. ผู้ใช้จะต้องเลือกโหมดในการทำงานก่อนโดยสามารถเลือกได้ 3 โหมด ดังนี้

   3.1 Dynamic Mode : เป็นโหมดที่จะให้ผู้ใช้กรอกคำสั่ง sql เอง จากนั้นโปรแกรมจะทำการประมวลผล และส่งผลลัพธ์ออกมาให้ทางตารางด้านล่าง โดยผู้ใช้ต้องคลิ๊กที่ปุ่ม DYNAMIC ดังรูป



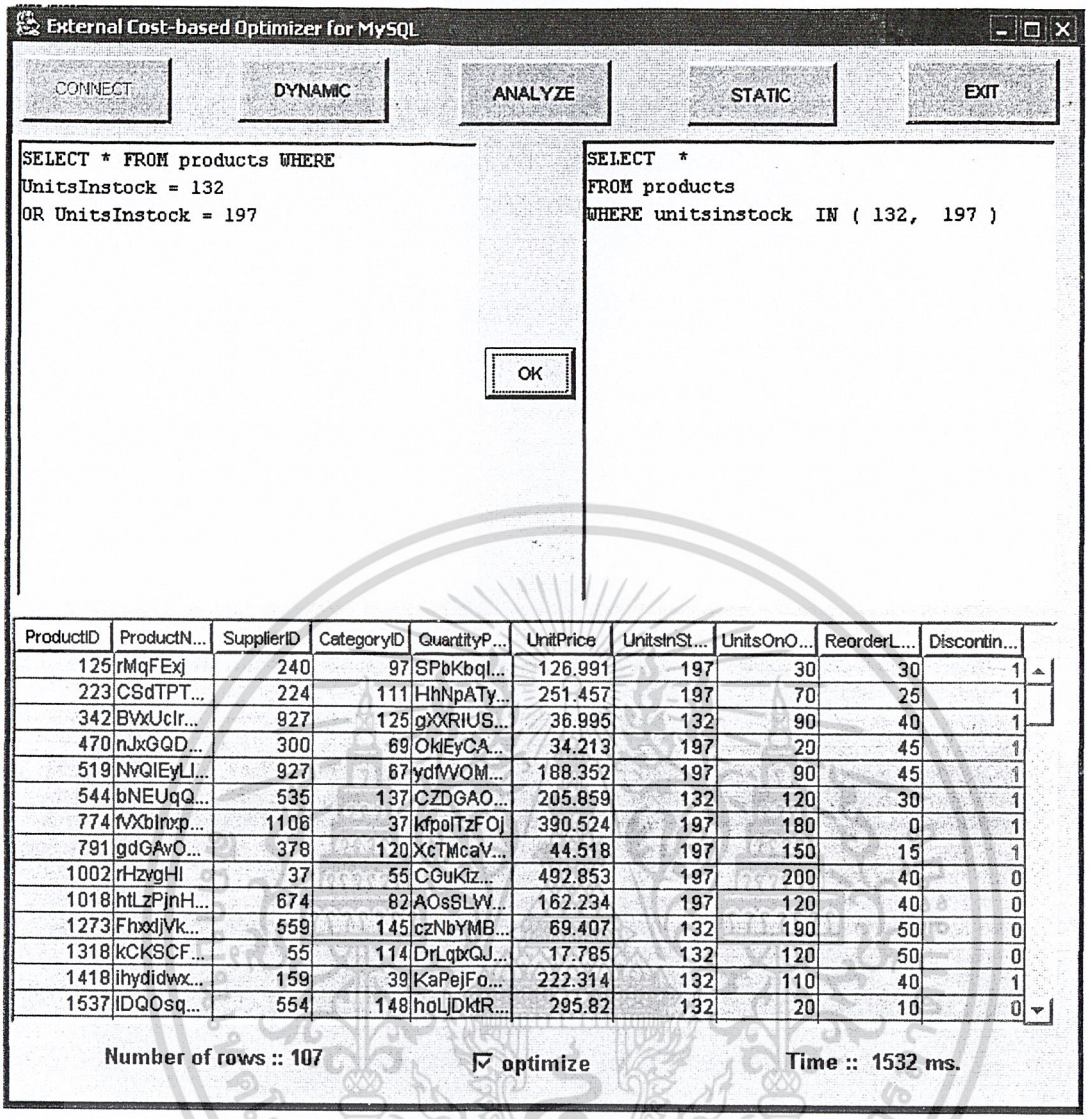
รูปแสดงตำแหน่งของปุ่ม DYNAMIC

จากนั้นให้ผู้ใช้เขียน query ที่ต้องการลงในฟิลด์ข้อความ (text field) ฝั่งซ้าย และถ้าต้องการให้โปรแกรมทำการประมวลคำสั่ง sql ที่เข้ามาให้อยู่ในรูปแบบที่เหมาะสมนั้น ผู้ใช้จะต้องทำเครื่องหมายที่ช่อง optimize ด้วย ดังรูป



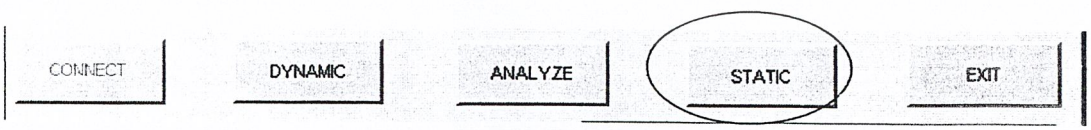
รูปแสดงการป้อนคำสั่ง query

จากนั้นผู้ใช้ต้องคลิ๊กที่ปุ่ม OK จากนั้นโปรแกรมทำการประมวลผล (optimize) query ที่ผู้ใช้ได้เขียนไว้ แล้วจะแสดง query ใหม่ที่ผ่านการ optimize แล้วใน text field ฝั่งขวา และนำผลลัพธ์ที่ได้จากการดำเนินการกับ query นั้นมาแสดงที่ตารางด้านล่าง นอกจากนี้ ที่บริเวณด้านล่างของหน้าจออินเทอร์เฟซยังแสดงจำนวนแถวที่เป็นผลลัพธ์ของ query และเวลาของการดำเนินการกับ query ดังรูป



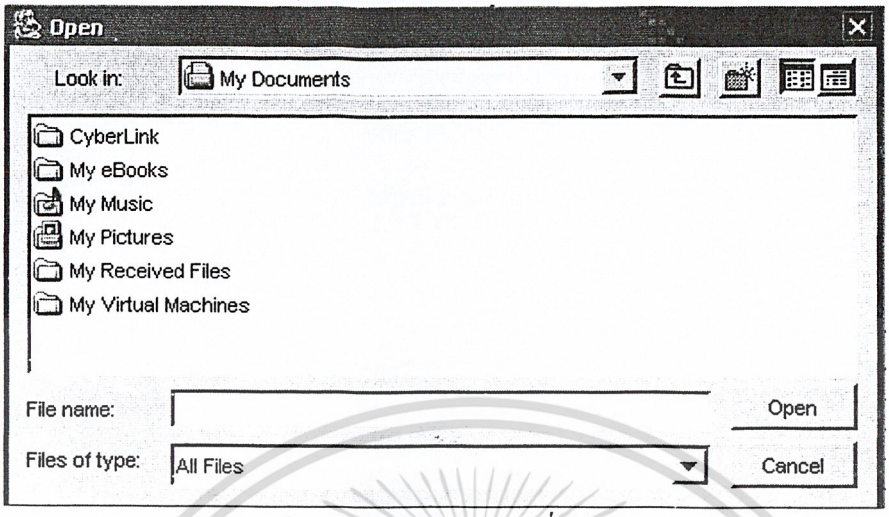
รูปแสดงผลลัพธ์ของโปรแกรมในการประมวลผล query

3.2 Static Mode : เป็นโหมดที่ใช้ในการสแกนไฟล์ เพื่อหาคำสั่ง sql ในโปรแกรมของผู้ใช้ ซึ่งเมื่อโปรแกรมนี้พบ query แล้ว จะทำการแปลง query นั้นของผู้ใช้ให้อยู่แบบที่เหมาะสมโดยอัตโนมัติ ซึ่งการทำงานในโหมดนี้ ผู้ใช้ต้องคลิ๊กที่ปุ่ม STATIC ดังรูป



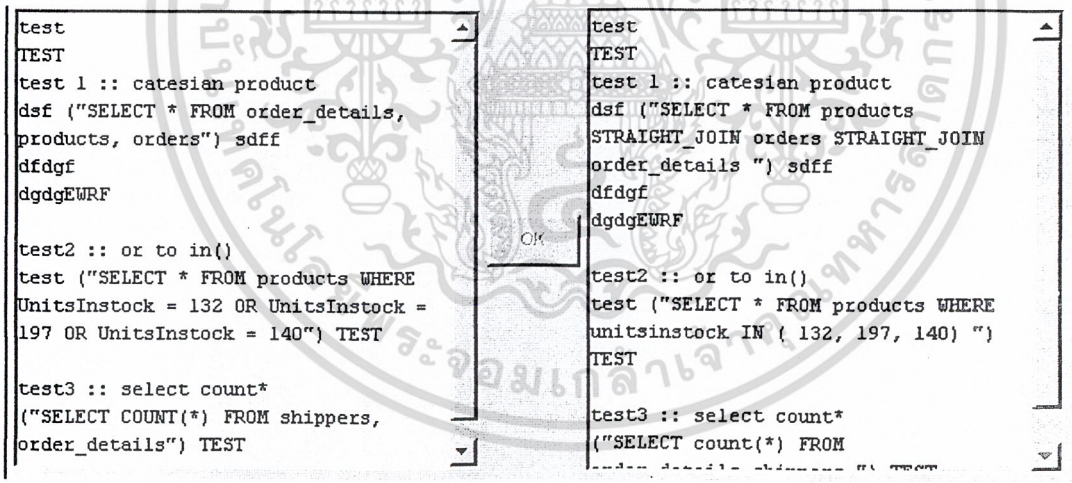
รูปแสดงตำแหน่งของปุ่ม STATIC

จากนั้นให้ผู้ใช้เลือกไฟล์ที่ต้องการ ดังภาพ



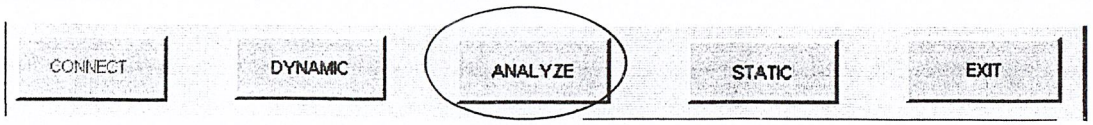
รูปแสดงการให้ผู้ใช้เลือกไฟล์ที่ต้องการ

แล้วโปรแกรมนี้จะทำการเปลี่ยนคำสั่ง sql เหล่านั้นให้เอง โดยฟิลด์ข้อความฝั่งซ้ายจะแสดงข้อความในไฟล์ของผู้ใช้ก่อนการประมวลผล และฟิลด์ข้อความฝั่งขวาจะแสดงข้อความในไฟล์หลังจากทำการประมวลผลแล้ว ดังรูป



รูปแสดงการประมวลผลใน Static Mode

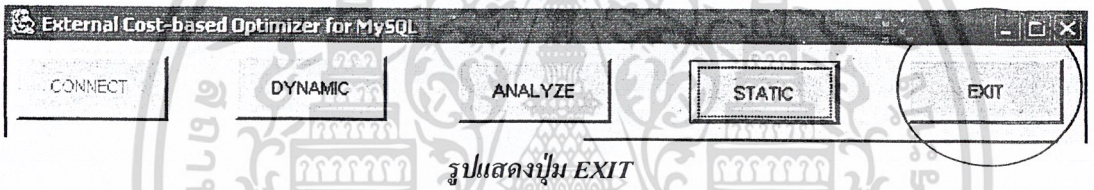
3.3 Analyze Mode : เป็นโหมดที่ใช้ในการอัพเดตค่าสถิติที่เก็บไว้โดยโปรแกรม ซึ่งการทำงานในโหมดนี้ ผู้ใช้ต้องคลิ๊กที่ปุ่ม ANALYZE ดังรูป



รูปแสดงปุ่ม ANALYZE

จากนั้นโปรแกรมจะทำการอัพเดตค่าสถิติที่เก็บไว้ให้เป็นไปตามข้อมูลล่าสุดในฐานข้อมูล ซึ่งเมื่อการอัพเดตค่าเสร็จสิ้นแล้ว ตารางด้านล่างของอินเทอร์เฟซจะแสดงจำนวนแถวของแต่ละตารางในฐานข้อมูลที่กำลังติดต่ออยู่ ตารางด้านล่างจะแสดงผลลัพธ์เป็นจำนวน

4. เมื่อผู้ใช้ต้องการจบการทำงานของโปรแกรม ให้ผู้ใช้คลิ๊กที่ปุ่ม EXIT หรือที่ปุ่มกากบาทด้านบน แล้วโปรแกรมจะทำการยกเลิกการติดต่อกับฐานข้อมูลให้เองโดยอัตโนมัติ จากนั้นโปรแกรมจะจบการทำงาน



รูปแสดงปุ่ม EXIT

บรรณานุกรม

[1] **"MySQL Reference Manual"** http://dev.mysql.com/doc/mysql/en/index.html

[2] **"MySQL Optimization"** http://jeremy.zawodny.com/mysql/mysql-optimization.html

[3] **"Oracle8i Concepts : The Optimizer"**
http://www.csee.umbc.edu/help/oracle8/server.815/a67781/c20a_opt.htm

[4] Abraham Silberschatz, Henry F. Korth and S. Sudarshan : **"Database System Concepts, Fourth Edition."**, McGraw-Hill, 2002

[5] Derek J. Balling and Jeremy Zawodny : **"High Performance MySQL"**, O'Reilly, 2004