ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

An External Cost-Based Optimizer for Rule-Based DBMS

นายทวีศักดิ์ อินเมฆ

นายวรนนท์ ชำนาญศิริ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

An External Cost-Based Optimizer for Rule-Based DBMS

โดย

นายทวีศักดิ์ อินเมฆ

นายวรนนท์ ชำนาญศิริ

อาจารย์ที่ปรึกษา

รศ.ดร. ศุภมิตร จิตตะยโศธร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ ปีการศึกษา 2547

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

An External Cost-Based Optimizer for Rule-Based DBMS

คณะผู้จัดทำ นาย ทวีศักดิ์ อินเมฆ 45015369

นาย วรนนท์ ชำนาญศิริ 45015378

.....อาจารย์ที่ปรึกษา

(รศ.ดร. ศุภมิตร จิตตะยโศธร)

ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ

นาย ทวีศักดิ์ อินเมฆ
นาย วรนนท์ ชำนาญศิริ
รศ.ดร. ศุภมิตร จิตตะยโสธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

ในปัจจุบันมีระบบจัดการฐานข้อมูลที่เป็นแบบอิงสถิติ (Cost-Based) อยู่จำนวนมาก แต่ผู้ใช้ไม่ทราบถึงการใช้งาน ผู้ใช้ส่วนใหญ่จะทราบเพียงวิธีการของแบบอิงกฎ (Rule-Based) ทำให้สนใจเพียงแค่การเขียนคำสั่ง SQL ให้ดีที่สุด ซึ่งเป็นการใช้งานระบบจัดการฐานข้อมูลไม่เต็มประสิทธิภาพ นอกจากนั้นในการใช้ระบบจัดการฐานข้อมูลแบบอิงสถิติ จำเป็นจะต้องมีการเก็บสถิติเป็นเวลานาน

ปริญญานิพนธ์เล่มนี้จะกล่าวถึงวิธีการ เปลี่ยนแปลงรูปแบบคำสั่ง SQL โดยพิจารณาข้อมูลทางสถิติที่ได้เก็บมาเอง เพื่อให้สามารถใช้งานระบบจัดการฐานข้อมูลแบบอิงสถิติของระบบจัดการฐานข้อมูลได้เต็มประสิทธิภาพ โดยที่ผู้ใช้โปรแกรมไม่จำเป็นต้องมีความรู้ในเรื่องระบบจัดการฐานข้อมูลแบบอิงสถิติและวิธีการเก็บสถิติ

An External Cost-Based Optimizer for Rule-Based DBMS

Taweesak Inmek

Voranon Chumnansiri

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-base optimization, majority of users mostly focus on writing the sql query.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 the 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

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี ด้วยคำแนะนำและความช่วยเหลือจากหลายๆฝ่าย โดยเฉพาะอย่างยิ่ง อาจารย์ ศุภมิตร จิตตะยโสธร อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่คอยดูแลเอาใจใส่ แนะนำ ดูแลให้คำปรึกษาที่ดีเสมอมา

ขอขอบคุณเพื่อนๆทุกคนที่คอยช่วยเหลือให้กำลังใจอยู่ร่วมทุกข์ร่วมสุขกันมาตลอดระยะเวลาที่ทำการศึกษาอยู่ด้วยกัน ณ สถาบันแห่งนี้

และสุดท้ายขอขอบพระคุณบุคคลที่สำคัญที่สุดในชีวิตที่ทำให้ข้าพเจ้ามีวันนี้นั่นคือ บิดา มารดา และบุคคลในครอบครัวอันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดู ให้การอบรมสั่งสอน ข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ และความรักเสมอมา ข้าพเจ้าขอกราบพระคุณอย่างสูงมา ณ ที่นี้

นาย ทวีศักดิ์ อินเมฆ

นาย วรนนท์ ชำนาญศิริ

สารบัญ

	หน้าที่
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VIII
สารบัญรูป	IX
บทที่ 1 บทนำ	1
1.1 ความสำคัญและความเป็นมา	1
1.2 วัตถุประสงค์ของโครงงาน	2
1.3 ที่มาของปัญหา	2
1.4 แนวทางการแก้ไข	2
1.5 ขอบเขตการทำโครงงาน	2
1.6 ผลที่คาดว่าจะได้รับ	3
บทที่ 2 Query Processing	4
2.1 ภาพรวมของการทำ Query Processing	4
2.2 การวัดต้นทุนของ query	5
บทที่ 3 การทำ Optimization	7
3.1 ตัวประมวลผลคำสั่ง SQL (SQL Optimizer)	7
3.1.1 ตัวประมวลผลแบบอิงกฎ (Rule-Based Optimizer)	7
3.1.2 ตัวประมวลผลแบบอิงสถิติ (Cost-Based Optimizer)	8
3.2 การทำ Optimization ของ Oracle	9
3.2.1 การประมวลผลแบบอิงกฎ (Rule-Based Optimization)	10
3.2.2 การประมวลผลแบบอิงสถิติ (Cost-Based Optimization)	11
บทที่ 4 วิธีที่ตัวประมวลผลทำการแปลง SQL สเตตเมนท์	13
4.1 วิธีที่ตัวประมวลผลแบบอิงสถิติแปลง OR ไปเป็น Compound Queries	13
4.2 วิธีที่ตัวประมวลผลแบบอิงสถิติแยก Subqueries	13
4.3 วิธีที่ตัวประมวลผลแบบอิงสถิติรวม Views	14
4.3.1 ความสามารถในการรวม Views	14
4.3.2 การรวม View เชิงซ้อน	15

สารบัญ(ต่อ)

	หน้าที่
4.4 วิธีที่ตัวประมวลผลแบบอิงสถิติวาง Predicates	15
4.4.1 วิธีที่ตัวประมวลผลแบบอิงสถิติใช้งานแอกกริเกรทฟังก์ชั่นกับ View	15
4.4.2 วิธีที่ตัวประมวลผลแบบอิงสถิติประมวลผล Views ในการทำOuter Joins	15
4.5 วิธีที่ตัวประมวลผลแบบอิงสถิติประมวลผล Compound Queries	15
บทที่ 5 Oracle Hints	16
5.1 Optimizer Hints	18
5.1.1 all_rows Hint	18
5.1.2 rule Hint	18
5.1.3 first_rows Hint	19
5.2 Table Join Hints	19
5.2.1 use_hash Hint	19
5.2.2 use_merge Hint	20
5.2.3 use_nl Hint	21
5.2.4 star Hint	21
5.3 Table Anti-Join Hints	23
5.3.1 merge_aj Hint	23
5.3.2 hash_aj Hint	23
5.4 Index Hints	24
5.4.1 index Hint	24
5.4.2 index_join Hint	24
5.4.3 and_equal Hint	24
5.4.4 index_asc Hint	25
5.4.5 no_index Hint	25
5.4.6 index_desc Hint	25
5.4.7 index_combine Hint	25
5.4.8 index_ffs Hint	26
5.4.9 Use_concat Hint	26
5.5 Parallel Hints	26
5.5.1 parallel hint	26
5.5.2 noparallel Hint	27

สารบัญ(ต่อ)

	หน้าที่
5.6 Table Access Hints	27
5.6.1 full Hint	27
5.6.2 ordered Hint	27
บทที่ 6 การปรับปรุงQUERY	28
6.1 Driving Table	28
6.1.1 ตำแหน่งของ Driving Table	28
6.1.2 การเปลี่ยน Driving Table ของ Rule-Based	28
6.2 การปรับปรุงการทำ Catesian product	29
6.3 ลดการเสียเวลาในการหาลำดับการjoinตารางโดยใช้ ORDERED Hint	29
6.4 การใช้งาน USE_CONCAT Hint เพื่อให้มีการทำ Union All	30
บทที่ 7 การทำงานและโครงสร้างของระบบ	31
7.1 ส่วนติดต่อกับผู้ใช้(User interface)	31
7.2 การทำงานของระบบ	31
7.3 โครงสร้างของระบบ	32
7.4 วิธีการในการเก็บสถิติ	33
บทที่ 8 การวิเคราะห์และการออกแบบระบบ	35
8.1 การวิเคราะห์ออกแบบระบบ	36
8.1.1 วิธีการแบบสถิต	36
8.1.2 วิธีการแบบพลวัต	36
8.2 ยูสเคสไดอะแกรม (Use case Diagram)	36
8.3 Sequence Diagram	39
8.3.1 Sequence Diagram ของการทำงานแบบสถิต	39
8.3.2 Sequence Diagram ของการแสดงข้อมูลทางสถิติ	40
8.3.3 Sequence Diagram ของการทำงานแบบพลวัต	41
8.4 Collaboration Diagram	42
8.4.1 Collaboration Diagram ของการทำงานแบบพลวัต	42
8.4.2 Collaboration Diagram ของการดูสถิติ	43
8.4.3 Collaboration Diagram ของการทำงานแบบพลวัต	43
8.5 Class Diagram ของระบบ	44

สารบัญ(ต่อ)

	หน้าที่
บทที่ 9 การทดลองและวิเคราะห์ผล	46
9.1 ขั้นตอนการทดลอง	46
9.2 ส่วนประกอบในการทดลอง	47
9.3 การทดลอง	47
9.3.1 ทดลองการ optimize ด้วยการทำ multiple join	47
9.3.1.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle	48
9.3.1.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle	48
9.3.2 ทดลองการ optimize ด้วยการทำ Catesian product	49
9.3.2.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle	49
9.3.2.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle	50
9.3.3 ทดลองการ optimize ด้วยการทำ Multiple Join ที่มีการทำ Catesian product	50
9.3.3.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle	51
9.3.3.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle	51
บทที่ 10 บทสรุปและข้อเสนอแนะ	52
10.1 สรุปผลการดำเนินงาน	52
10.2 ปัญหา และ อุปสรรค	52
10.3 แนวทางในการพัฒนาต่อ	52
ภาคผนวก ก.	54
บรรณานุกรม	61

สารบัญตาราง

ตารางที่		หน้าที่
3.1	ลำดับเงื่อนไขของตัวประมวลผลแบบอิงกฎ	10
5.1	เงื่อนไขที่ทำให้ hints ไม่ถูกนำไปใช้งาน	17
9.1	แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ optimize การทำ multiple join	47
9.2	แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)	48
9.3	แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)	48
9.4	แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ Optimize การทำ Catesian product	49
9.5	แสดงจำนวน เวลาของการประมวลผลการ optimize catesian product วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)	49
9.6	แสดงจำนวน เวลาของการประมวลผลการ optimize catesian product วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)	50
9.7	แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ Optimize การทำ Multiple Join ที่มีการทำ Catesian product	50
9.8	แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join ที่มีการทำ Catesian product วัดเทียบกันสิบครั้งโดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)	51
9.9	แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join ที่มีการทำ Catesian product วัดเทียบกันสิบครั้งโดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)	51

สารบัญรูป

รูปที่		หน้าที่
3.1	แสดงการทำงานของการประมวลผลคำสั่ง SQLของ Oracle	9
3.2	แสดงส่วนประกอบของตัวประมวลผลแบบอิงสถิติของ Oracle	12
5.1	แสดงการทำ hash join และการใช้งาน RAM	20
5.2	star table join	22
7.1	โครงสร้างของระบบ	32
8.1	แสดง Business Process Modeling ของระบบ	35
8.2	Use case diagram แสดงการทำงานของระบบ	37
8.3	Use case Diagram แสดงการทำงานภายใน	38
8.4	Sequence Diagram แสดงลำดับการทำงานของ การทำงานในแบบสถิต	39
8.5	Sequent Diagram แสดงลำดับการทำงานเมื่อมีการแสดงข้อมูลทางสถิติที่ถูกจัดเก็บไว้	40
8.6	Sequent Diagram แสดงการทำงานของแบบพลวัต	41
8.7	Collaboration Diagram ของการทำงานแบบสถิต	42
8.8	Collaboration Diagram ของการดูสถิติ	43
8.9	Collaboration Diagram ของการทำงานแบบ พลวัต	43
8.10	ภาพ Class Diagram ของทั้งระบบ	44
ก.1	แสดงลักษณะของโปรแกรม	55
ก.2	Dialog Box ที่ใช้กรอกข้อมูลเพื่อเข้าใช้งาน	56
ก.3	แสดงการเข้าใช้งานโปรแกรม	56
ก.4	แสดงการใช้งานในแบบสถิต	57
ก.5	การใช้งานในแบบพลวัต	58
ก.6	แสดงผลการ optimize คำสั่งผ่านโปรแกรมในแบบพลวัต	59
ก.7	แสดงผลลัพธ์ของการเก็บสถิติด้วยโปรแกรม	60

บทที่ 1

บทนำ

1.1 ความสำคัญและความเป็นมา

โลกในยุคปัจจุบันนี้ ข้อมูลข่าวสารต่างสามารถส่งถึงกันได้ง่ายกว่าสมัยก่อนมาก ดังนั้นวิถีชีวิตของเราทุกวันนี้จึงเปลี่ยนแปลงไปมาก ในแต่ละวันเราจะต้องบริโภคข้อมูลข่าวสารเป็นจำนวนมาก ซึ่งข้อมูลข่าวสารที่ผ่านเข้ามาในชีวิตเราเหล่านี้ บางครั้งก็ไม่จำเป็นสำหรับชีวิตเรา บางข่าวสารก็ต้องจำไว้หรือควรจะมีการจัดเก็บเพื่อไว้ใช้ในวันข้างหน้า แต่เนื่องจากข้อมูลในปัจจุบันมีแนวโน้มว่าจะเพิ่มมากขึ้นทุกวัน จนเราไม่สามารถที่จะจดจำได้หมด จึงมีการใช้เทคโนโลยีทั้งทาง Software และ Hardware เข้ามาช่วยในการเก็บข้อมูล อันได้แก่ Computer ที่กำลังมีบทบาทสำคัญมากในปัจจุบัน แต่ Computer อย่างเดียวจะไม่สามารถทำงานได้ฉลาดเพียงพอที่จะจัดการกับข้อมูลที่ปริมาณมากขึ้นทุกวันได้ ดังนั้นจึงมีความจำเป็นต้องมี Software ที่เข้ามาช่วยในเรื่องนี้คือ ระบบจัดการฐานข้อมูล (Database Management System) หรือที่เราเรียกสั้นๆว่า DBMS

ระบบจัดการฐานข้อมูล ในปัจจุบันมีมากมายหลาย ยี่ห้อ ให้เลือกใช้ตามความเหมาะสม ซึ่งความแตกต่างของระบบจัดการฐานข้อมูลเหล่านี้ก็คือ ประสิทธิภาพในด้านต่างๆ เช่น ความเสถียรภาพ การฟื้นฟูระบบ (Recovery) หลังจากที่ระบบล่ม และที่สำคัญคือความเร็วในการจัดเก็บและนำข้อมูลมาแสดงผลที่นับวันจะมีความสำคัญมากขึ้นเพราะข้อมูลข่าวสารในปัจจุบันมีแนวโน้มที่จะเพิ่มขึ้นเรื่อย แต่ระบบจัดการฐานข้อมูลที่มีประสิทธิภาพในด้านนี้สูงที่มักจะเป็นระบบจัดการฐานข้อมูลแบบมีการอ้างอิงการเรียกข้อมูลแต่ละครั้งด้วยการอ้างอิงทางสถิติ (Cost-Based) ซึ่งมีความจำเป็นในการทำการเก็บสถิติ

เนื่องด้วยผู้ใช้งานส่วนมากไม่ทราบถึงความสำคัญของการเก็บสถิติ ทำให้ใช้งานตัวประมวลผลแบบอิงสถิติได้ไม่เต็มที่ อีกทั้งการเก็บข้อมูลแต่ละครั้งจำเป็นต้องใช้เวลานานมาก จึงได้มีการคิดที่จะทำตัวประมวลผลภายนอกแบบอิงสถิติบนระบบจัดการฐานข้อมูลแบบอิงกฎ (Rule-Based DBMS)

ด้วยเหตุนี้จึงเป็นที่มาของ “ตัวประมวลผลภายนอกแบบอิงสถิติ บนระบบจัดการฐานข้อมูลแบบอิงกฎ” (An External Cost-Based Optimizer for Rule-Based DBMS)

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาวิธีการเพิ่มความเร็วในการเรียกค้นข้อมูล
2. เพื่อเพิ่มความสะดวกสบายในการปรับแต่งเพื่อเพิ่มประสิทธิภาพของระบบจัดการฐานข้อมูล โดยที่ผู้ใช้ไม่จำเป็นต้องมีความรู้ในเรื่อง ฐานข้อมูลมาก
3. ลดต้นทุนในการจัดหาระบบจัดการฐานข้อมูลที่เป็นแบบอิงสถิติ (Cost-Based) มาใช้งาน
4. นำความรู้ที่ได้จากการเรียนมาประยุกต์ ใช้งานจริงให้เกิดประโยชน์

1.3 ที่มาของปัญหา

1. ข้อมูลมีจำนวนเพิ่มขึ้นอย่างรวดเร็ว จนเกิดความล่าช้าในการเรียกค้นข้อมูล
2. ผู้ใช้ไม่มีความรู้ถึงวิธีการเรียกค้นข้อมูลเพื่อให้ใช้เวลาน้อยที่สุด
3. ระบบจัดการฐานข้อมูลที่มีสามารถเรียกค้นข้อมูลได้อย่างรวดเร็วมีราคาแพงมาก
4. ระบบจัดการฐานข้อมูลที่มีการทำงานแบบอิงสถิติ ใช้เวลาในการเก็บสถิตินานมากเนื่องจากต้องเก็บทั้ง Database space ทั้งที่จริงแล้วอาจใช้เพียงบางส่วน
5. บางครั้งการใช้งานตัวประมวลผลแบบอิงสถิติ อาจเสียเวลามากกว่าการใช้งานตัวประมวลผลแบบอิงกฎ และบางครั้งตัวประมวลผลแบบอิงกฎอาจสร้างแผนการประมวลผล (execution plan) ที่ดีน้อยกว่าที่ตัวประมวลผลแบบอิงกฎสร้างได้

1.4 แนวทางการแก้ไข

1. ศึกษาการเขียนคำสั่ง SQL ที่จะช่วยเพิ่มความเร็วในการประมวลผลได้เร็วขึ้น
2. เก็บสถิติบางส่วนของ DBMS เพื่อลดเวลาในการเก็บสถิติ ซึ่งทำการเก็บเองโดนไม่ต้องอาศัย DBA
3. สร้าง Software สำหรับนำข้อมูลทางสถิติที่เก็บเองมาใช้ และเปลี่ยนแปลงคำสั่ง SQL เพื่อให้ประมวลผลให้ได้ผลลัพธ์เร็วขึ้น เพื่อช่วยให้ผู้ใช้สามารถใช้งานได้สะดวก โดยที่ผู้ใช้ไม่จำเป็นต้องมีความรู้ในการปรับแต่งคำสั่ง SQL

1.5 ขอบเขตการทำโครงงาน

1. ศึกษาวิธีการจัดเก็บข้อมูลทางสถิติของระบบจัดการฐานข้อมูล
2. ปรับปรุงแก้ไขคำสั่งภาษา SQL โดยพิจารณาจากสถิติที่ได้จากการจัดเก็บเอง
3. ศึกษาการสร้างดัชนี (Index) สำหรับแต่ละตาราง และวิธีการเลือกใช้ดัชนี
4. ศึกษาวิธีการใช้คำสั่งพิเศษของระบบจัดการฐานข้อมูลนั้น
5. เขียนโปรแกรมเพื่อจัดเก็บสถิติเอง และปรับปรุง SQL Statement เอง

1.6 ผลที่คาดว่าจะได้รับ

1. ได้รับความรู้ในการปรับแต่งฐานข้อมูลให้เกิดประสิทธิภาพที่ดีขึ้น
2. ได้โปรแกรมที่ทำหน้าที่เสมือนเป็นผู้ช่วยผู้ดูแลระบบ (DBA) ในการเก็บสถิติ
3. ได้นำความรู้มาประยุกต์ใช้งานจริง
4. ได้ตัวต้นแบบของโปรแกรมที่สามารถนำไปพัฒนาต่อได้

บทที่ 2

Query Processing

2.1 ภาพรวมของการทำ Query Processing

ในการส่ง Query เข้าไปประมวลผลนั้น วิธีการในการที่จะคำนวณหาคำตอบออกมานั้นสามารถทำได้มากมายหลายแบบ ตัวอย่างเช่น ใน SQL นั้น Query หนึ่งๆจะสามารถแปลงเป็น Relational-Algebra ได้หลายแบบ ซึ่ง Relational-Algebra จะเป็นตัวแทนส่วนหนึ่งที่จะบอกถึงวิธีการในการประมวลผล Query นั้นๆ โดยใน Relational-Algebra จะมีโอเปอเรเตอร์ (Operater) ทั้งหมดแปดตัว ดังนี้

1. Select (Restrict)
2. Project
3. Join (Natural Join)
4. Union
5. Intersection
6. Difference
7. Product (Cartesian Product)
8. Divide

โอเปอเรเตอร์ที่สำคัญหลักๆ ได้แก่ Select, Project และ Join ซึ่งตัวอย่างการประมวลผลของ Relational-Algebra โดยพิจารณา Query ต่อไปนี้

SELECT balance

FROM account

WHERE balance < 2500

ซึ่ง Query นี้จะถูกแปลงเป็น Relational-Algebra ได้สองแบบดังต่อไปนี้

- σ balance < 2500 ($\prod$ balance(account))
- $\prod$ balance (σ balance < 2500 (account))

จาก Reletion-Algebra ข้างบนนั้น เครื่องหมาย $\prod$ แทน operator Select และ σ แทนโอเปอเรเตอร์ Project ซึ่งจะเห็นได้ว่าเราสามารถดำเนินการกับ Relational-Algebra โอเปอเรชั่น (Operation) ได้หลายอัลกอริทึมเช่น ถ้าเราจะทำตามแบบที่ใช้ Select ก่อน เราจะต้องค้นหา tuple ทุกตัวใน account เพื่อจะหาว่า tuple ใดที่มี balance น้อยกว่า 2500 แต่ถ้ามี B+-tree index บนแอตทริบิว balance แล้ว เราจะใช้ดัชนี (index) แทนตำแหน่งของ tuple

ในการที่จะอธิบายถึงวิธีการประมวลผล query นั้น จะไม่ใช่เพียงแค่การจัดการกับ Relational-Algebra เอกเพรสชันเท่านั้น แต่ยังจะต้องทำการระบุวิธีที่จะใช้ในการประมวลผลแต่ละโอเปอเรชั่นด้วย ซึ่งการระบุวิธีที่ใช้ในการประมวลผลนั้นอาจเป็นการระบุถึงอัลกอริทึมที่จะใช้กับโอเปอเรชั่นที่ระบุไว้ หรือเป็นการบอกว่าจะใช้ดัชนีตัวใดก็ได้ ซึ่งจะเรียกว่าเรียกว่าการประเมินผลล่วงหน้า (evaluation primitive) และลำดับของการประเมินผลล่วงหน้านั้นยังสามารถนำไปใช้ในการประมวลผล query ได้อีก ซึ่งเรียกว่า แผนการประมวลผล (execution plan)

แผนการประมวลผลที่ต่างกันของ query เดียวกันจะมีค่าต้นทุน (cost) ที่ต่างกันด้วย ซึ่งเป็นหน้าที่ของระบบที่จะต้องสามารถสร้างแผนการประมวลผลที่จะลดต้นทุนในการดำเนินการกับ query ให้มีค่าน้อยที่สุด

ในการที่จะ optimize query ใดๆ ตัวประมวลผล query (query optimizer) จะต้องทราบต้นทุนของแต่ละโอเปอเรชั่นก่อน ถึงแม้ว่าค่าที่แน่นอนจะทำการคำนวณออกมาได้ยาก เพราะมันขึ้นอยู่กับตัวแปรอีกมากมาย แต่ก็สามารถที่จะประเมินค่าต้นทุนนั้นอย่างคร่าวๆได้

2.2 การวัดต้นทุนของ query

ต้นทุนของการประมวลผล query นั้น สามารถวัดได้ในรูปแบบของจำนวนทรัพยากรที่ต่างกัน ซึ่งรวมถึง การเข้าถึงดิสก์ (disk), เวลาของ CPU ในการดำเนินการกับ query และต้นทุนในการสื่อสาร โดยเวลาในการประมวลผล query นั้น (รวมเวลาที่ใช้ในการวางแผนการดำเนินการด้วย) จะเกิดจากต้นทุนเหล่านี้รวมกัน จึงสามารถวัดค่าต้นทุนของแต่ละแผนได้

ในระบบฐานข้อมูลที่มีขนาดใหญ่นั้น การเข้าถึงดิสก์ที่ซึ่งวัดจากจำนวนบล็อกที่ส่งผ่านจากดิสก์มักจะเป็นต้นทุนที่สำคัญที่สุด เพราะการเข้าถึงดิสก์จะช้ามากเมื่อเทียบกับการทำงานบนหน่วยความจำ (memory) ยิ่งไปกว่านั้นความเร็วของหน่วยประมวลผลกลางในปัจจุบันนี้เร็วกว่าการเข้าถึงดิสก์มาก ดังนั้นจึงเหมือนกับว่าเวลาที่ใช้กับดิสก์จะมีอิทธิพลมากต่อความเร็วทั้งหมดในการดำเนินการกับ query ดังนั้นส่วนใหญ่จึงมักใช้ต้นทุนของการเข้าถึงดิสก์ในการวัดต้นทุนของแผนการดำเนินการกับ query

ในการวัดค่าต้นทุนที่ใช้จริงจะใช้จำนวนของบล็อกที่ส่งผ่านจากดิสก์ เพื่อให้สามารถคำนวณต้นทุนในการเข้าถึงดิสก์ได้อย่างง่ายๆ ซึ่งสมมุติให้การส่งผ่านของทุกบล็อกมีค่าต้นทุนที่เท่ากัน แต่ถ้าหากต้องการค่าที่มีความแม่นยำมากกว่านี้ จำเป็นต้องแยกค่า sequential I/O เมื่อบล็อกที่ถูกอ่านอยู่ติดกัน และค่า random I/O เมื่อบล็อกที่ถูกอ่านไม่ได้อยู่ติดกัน ออกจากกัน และยังต้องเพิ่มค่าต้นทุนที่ใช้ในการค้นหาเพิ่มเติมสำหรับแต่ละการทำงานของ I/O ของดิสก์ แล้วเรายังต้องแยกการอ่านบล็อกและการเขียนบล็อกออกจากกันด้วย เนื่องจากการเขียนบล็อกลงดิสก์จะใช้เวลามากกว่าการอ่านบล็อกจากดิสก์ แต่ถ้าต้องการให้คำนวณได้แม่นยำมากยิ่งขึ้นไปอีก เราจะต้องคำนึงถึงสิ่งต่างๆต่อไปนี้ด้วย

1. จำนวนของการทำงานเพื่อค้นหา
2. จำนวนของบล็อกที่ถูกอ่าน
3. จำนวนของบล็อกที่ถูกเขียน

จากนั้นทำการบวกค่าเหล่านี้เข้าไปหลังจากที่คูณมันด้วยเวลาเฉลี่ยของการค้นหา, เวลาเฉลี่ยในการส่งผ่านบล็อกที่ถูกอ่าน และค่าเฉลี่ยในการส่งผ่านบล็อกที่ถูกเขียนตามลำดับ แต่ตัวประมวลผล query ในชีวิตจริงจะรวมค่าต้นทุนของ CPU เข้าไปในการคำนวณด้วย

บทที่ 3

การทำ Optimization

3.1 ตัวประมวลผลคำสั่ง SQL (SQL Optimizer)

ในทางอุดมคติแล้ว ตัวประมวลผลคำสั่ง SQL ควรสามารถที่จะทำการสร้างวิธีการในการเรียกค้นข้อมูลที่มีประสิทธิภาพที่สุดได้ ซึ่งหากตัวประมวลผลคำสั่ง SQL มีข้อมูลทั้งหมดของตารางที่ปรากฏอยู่ใน query แล้ว ตัวประมวลผลคำสั่ง SQL ควรจะเลือกใช้เส้นทางที่ดีที่สุดสำหรับการเรียกค้นข้อมูล

แต่ในความเป็นจริงนั้นตัวประมวลผลคำสั่ง SQL ยังต้องการการช่วยเหลือจากมนุษย์อยู่ ซึ่งเป็นความรับผิดชอบของผู้ดูแลจัดการระบบฐานข้อมูล (Database Administrator) ในการเก็บสถิติเพื่อให้ตัวประมวลผล SQL สามารถนำเอาสถิติเหล่านั้นไปใช้ได้ และความซับซ้อนของการประมวลผลคำสั่ง SQL (SQL optimization) ทำให้ยากต่อการที่ผู้ออกแบบระบบฐานข้อมูลจะออกแบบให้ตัวประมวลผลคำสั่ง SQL เลือกเส้นทางที่ดีที่สุดเสมอ ดังนั้นเราจำเป็นต้องมีความรู้ในการ optimize เพื่อที่จะช่วยตัวประมวลผลให้เลือกเส้นทางในการเข้าถึงข้อมูลได้มีประสิทธิภาพดีที่สุด

3.1.1 ตัวประมวลผลแบบอิงกฎ (Rule-Based Optimizer)

ตัวประมวลผลแบบอิงกฎนั้นจะไม่ใช้สถิติใดๆของตาราง แต่ตัวประมวลผลแบบอิงกฎจะใช้ heuristics เพื่อทำการหาเส้นทางการเข้าถึงข้อมูลที่ดีที่สุด โดยที่ตัวประมวลผลแบบอิงกฎจะทำการสร้างรายการของแผนการประมวลผลทั้งหมดที่เป็นไปได้ออกมา

ตัวประมวลผลแบบอิงกฎใช้กรรมวิธีการทำซ้ำ เพื่อทำการสร้างแผนการประมวลผลทำการพิจารณาแต่ละตารางในอนุประโยค FROM (FROM clause) และทุกทางที่เป็นไปได้ในการ Join ตารางนั้นกับตารางอื่นใน query นั้น แต่ละเส้นทางที่เป็นไปได้จะมีระดับชั้น (rank) ของต้นทุนและเส้นทางที่มีต้นทุนน้อยที่สุดจะถูกเลือก ซึ่งเป็นไปตามลำดับต่อไปนี้

สำหรับแต่ละตารางในอนุประโยค WHERE

- รายการของแต่ละแผนการประมวลผลได้ถูกสร้างขึ้นมาในรูปของเส้นทางที่เป็นไปได้ทั้งหมดของตาราง
- ทำการกำหนดค่าระดับชั้นให้แต่ละแผนการประมวลผล
- ตัวประมวลผลแบบอิงกฎเลือกใช้เส้นทางที่มีระดับชั้นต่ำที่สุด

สำหรับแต่ละตารางที่เหลืออยู่ในอนุประโยค WHERE

- ตัวประมวลผลแบบอิงกฎจะทำการหาวิธีการjoinที่เป็นไปได้ทั้งหมดที่สามารถถูกใช้เพื่อทำการ Join ตารางกับเซทผลลัพธ์ (result set) ที่ได้จากข้อที่สามที่ได้กล่าวไปข้างต้น
- เลือกใช้เทคนิคที่มีต้นทุนน้อยที่สุด

ซึ่งจะเห็นได้ว่าวิธีการกำหนดระดับชั้นของตัวประมวลผลแบบอิงกฎนั้นเป็นวิธีการแบบง่ายๆที่อาศัยความสัมพันธ์กันของต้นทุนและระดับชั้นสำหรับแต่ละโอเปอเรชั่น

3.1.2 ตัวประมวลผลแบบอิงสถิติ (Cost-Based Optimizer)

ตัวประมวลผลแบบอิงสถิติถูกสร้างมาใหม่ให้ดีกว่าตัวประมวลผลแบบอิงกฎ เพื่อหาเส้นทางการประมวลผลที่ดีที่สุดสำหรับแต่ละสเตตเมนท์ตัวประมวลผลแบบอิงสถิติใช้ข้อมูลทางสถิติที่เก็บเป็นฐานข้อมูล เช่น ขนาดของตาราง, จำนวนของแถว (row), การกำหนดคีย์ (key), และอื่นๆ ซึ่งค่อนข้างจะดีกว่า การใช้กฎบังคับ

ถ้าหากตารางไม่ได้ถูกวิเคราะห์เพื่อทำการเก็บสถิติแล้ว ตัวประมวลผลแบบอิงสถิติจะตรวจสอบเงื่อนไขตามกฎของตัวประมวลผลแบบอิงกฎเพื่อเลือกวิธีที่ดีที่สุดแทน ในบางครั้งก็อาจเป็นไปได้ที่จะมีการทำงานร่วมกันระหว่างตัวประมวลผลแบบอิงสถิติและตัวประมวลผลแบบอิงกฎเนื่องจากมีบางตารางที่ผ่านการวิเคราะห์แต่บางตารางไม่ได้วิเคราะห์

การทำงานของตัวประมวลผลแบบอิงสถิติสามารถแบ่งแยกย่อยได้ดังนี้

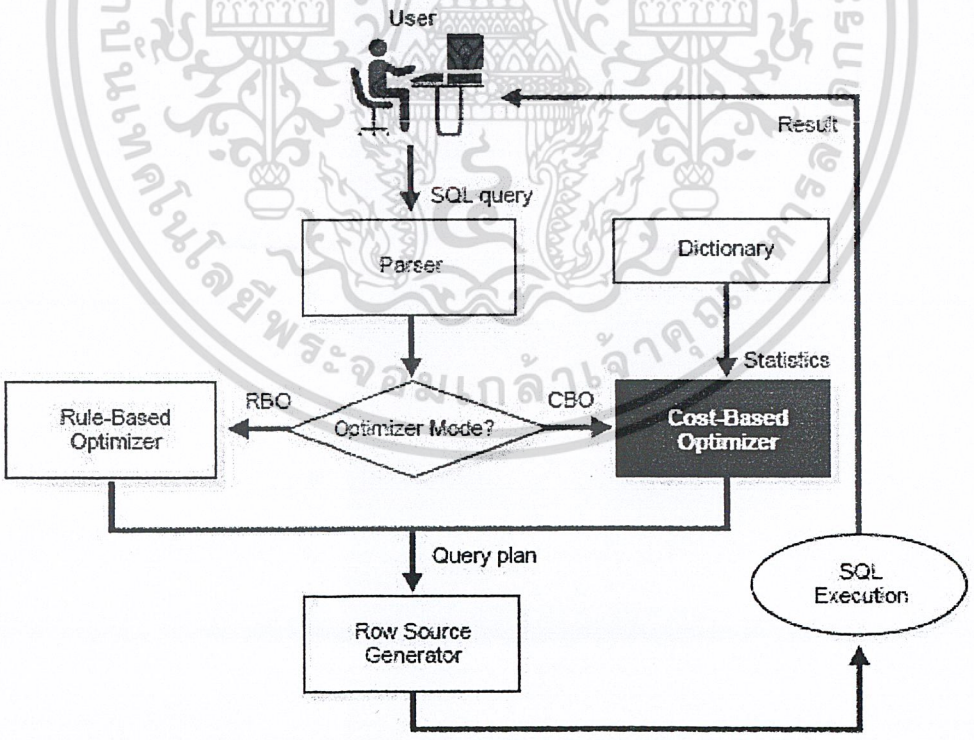
- แตก SQL ออกเป็นส่วนย่อย
- สร้างรายการของทุกทางที่จะเป็นไปได้ในการประมวลผล
- คำนวณต้นทุนของทุกรายการ
- เลือกเส้นทางที่ดีที่สุดโดยเลือกจากทางที่ใช้ต้นทุนน้อยที่สุด

3.2 การทำ Optimization ของ Oracle

เมื่อใดก็ตามที่ทำการประมวลผล SQL สเตตเมนท์นั้น ตัวประมวลผลจะตัดสินใจใช้วิธีที่จะเข้าถึงข้อมูลได้ดีที่สุด ดังที่ได้กล่าวไปแล้วว่าตัวประมวลผลมี2แบบคือ ตัวประมวลผลแบบอิงกฎและตัวประมวลผลแบบอิงสถิติ ซึ่งOracleสนับสนุนการใช้งานตัวประมวลผลทั้ง2แบบ

เพื่อการทำให้สเตตเมนท์ต่างๆดีที่สุดนั้น ตัวประมวลผลจะพิจารณาสิ่งต่างๆ ดังนี้

- รูปแบบไวยากรณ์ที่เราเขียนในสเตตเมนท์นั้นๆ
- เงื่อนไขต่างๆที่กำหนดในอนุประโยค WHERE
- ตารางที่เราจะไปเรียกค้นข้อมูล
- ดัชนีทุกตัวที่ใช้ในการเรียกค้นข้อมูลจากตาราง
- เวอร์ชันของระบบจัดการฐานข้อมูลเชิงสัมพันธ์ Oracle (Oracle Relational Database Management System : Oracle RDBMS)
- วิธีการของตัวประมวลผลที่เลือกใช้ในขณะนั้น
- SQL สเตตเมนท์ฮินท์ (SQL statement hints)
- ทุกสถิติที่สามารถใช้ได้
- ตำแหน่งของตารางในระดับกายภาพ
- ORA (parallel query, async I/O, อื่นๆ)



รูปที่ 3.1 แสดงการทำงานของการประมวลผลคำสั่ง SQL ของ Oracle

3.2.1 การประมวลผลแบบอิงกฎ (Rule-Based Optimization)

ตัวประมวลผลแบบอิงกฎ เป็นตัวประมวลผลแบบแรกของ Oracle ที่ถูกสร้างขึ้นมา ซึ่งจากที่ได้กล่าวไปแล้วว่าตัวประมวลผลแบบอิงกฎนั้นมีการกำหนดระดับชั้นที่สัมพันธ์กันกับต้นทุนที่ใช้ในแต่ละ SQL โอเปอเรชั่น ซึ่งระดับชั้นนี้ถูกตัวประมวลผลแบบอิงกฎใช้ในการเลือกแผนการประมวลผลที่ดีที่สุด

ตัวแปลที่มีผลต่อแผนการประมวลผลของตัวประมวลผลแบบอิงกฎของ Oracle นั้นได้แก่ดัชนีของตาราง, ลำดับของตารางในอนุประโยค FROM และ boolean เอกเพรสชัน (boolean expression) ใน SQL สเตตเมนท์นั้นๆ ซึ่งจะเป็นตัวกำหนดแผนการประมวลผลของ query นั้นๆ

ในตอนแรกนั้นตัวประมวลผลแบบอิงกฎของ Oracle นั้นได้กำหนดเงื่อนไขหรือกฎในการกำหนดระดับชั้นของแผนการประมวลผลเอาไว้ ซึ่งถูกเรียกว่ากฎทอง (golden rules) กฎเหล่าเป็นสิ่งกำหนดตัวประมวลผลถึงวิธีการกำหนดเส้นทางการประมวลผล (execution path) สำหรับสเตตเมนท์เมื่อต้องเลือกดัชนีใดๆจากหลายๆดัชนีและเมื่อต้องทำการตรวจหาทั้งตาราง กฎเหล่านี้แสดงในตารางที่ 3.1

Rank	Condition
1	ROWID fetch
2	Single row by cluster join
3	Single row by hash cluster join
4	Single row by primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite key
9	Single-column index
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort merge join
13	Max or min on indexed column
14	Order by on an indexed column
15	Full-Table scan

ตารางที่ 3.1 ลำดับเงื่อนไขของตัวประมวลผลแบบอิงกฎ

ตัวประมวลผลแบบอิงกฎมีข้อดีตรงความเรียบง่ายและบางครั้งที่สามารถสร้างวิธีการประมวลผลที่ไวกว่าตัวประมวลผลแบบอิงสถิติสร้างขึ้นมา ซึ่งเป็นเหตุให้เราสนใจที่จะนำเอาข้อดีในส่วนนี้มาใช้เพื่อให้เกิดประโยชน์ต่อการใช้งาน Oracle มากยิ่งขึ้นโดยการสังเกตคุณลักษณะของตัวประมวลผลแบบอิงกฎดังต่อไปนี้

- เมื่อเราสามารถใช้ดัชนีในการเข้าถึงข้อมูลในตารางได้ ให้พยายามใช้ดัชนี เนื่องจากการใช้ดัชนีดีกว่าการทำ full table scan หรือการทำ sort merge join (การทำ sert merge join ไม่ต้องการดัชนีในการทำงาน)
- พยายามเลือกใช้ driving table ที่เหมาะสม เนื่องจากมันจะเป็นตารางที่ถูกเลือกใช้เป็นตารางแรกซึ่งควรเป็นตารางที่มีจำนวนจำนวนแถวที่น้อยที่สุด ซึ่งตัวประมวลผลแบบอิงกฎจะนำเอา driving table ไปเป็นตารางแรกในการทำ nested loop join
- พยายามใช้ full table scan เป็นการทำงานท้ายที่สุด เนื่องจากตัวประมวลผลแบบอิงกฎไม่รู้ถึงวิธีการทำ parallel query และการทำ multiblocks read และตัวประมวลผลแบบอิงกฎจะไม่พิจารณาถึงขนาดของตารางนั้นๆ ดังนั้นตัวประมวลผลแบบอิงกฎจึงไม่ชอบการกระทำ full table scan และจะใช้ก็ต่อเมื่อไม่สามารถใช้ดัชนีได้เท่านั้น
- บางครั้งตัวประมวลผลแบบอิงกฎอาจเลือกใช้ดัชนีอื่นๆ ที่ไม่ใช่ดัชนีที่ควรจะเลือกใช้ที่สุดในอุดมคติ เนื่องจากว่าตัวประมวลผลแบบอิงกฎไม่ได้เข้าถึงสถิติที่แสดงถึง selectivity ของคอลัมน์ (column) ที่เป็นดัชนี
- ความเรียบง่ายบางครั้งอาจดีกว่า ซึ่งมีบ่อยครั้งที่ตัวประมวลผลแบบอิงกฎได้ทำการสร้างแผนการประมวลผลที่ดีที่สุดสำหรับบางฐานข้อมูลขึ้นมาได้

3.2.2 การประมวลผลแบบอิงสถิติ (Cost-Based Optimization)

ตัวประมวลผลแบบอิงสถิติของ Oracle ถูกสร้างขึ้นเพื่อที่จะจัดเตรียมทางเลือกที่ดีกว่าการประมวลผลแบบอิงกฎ เพราะ Oracle มองว่าการประมวลผล SQL จะมีประสิทธิภาพมากขึ้นถ้าตัวประมวลผลทราบถึงรายละเอียดของข้อมูลและดัชนีในตารางนั้น ซึ่งข้อมูลนั้นได้แก่ ข้อมูลของตาราง และข้อมูลของดัชนี

ข้อมูลของตารางได้แก่

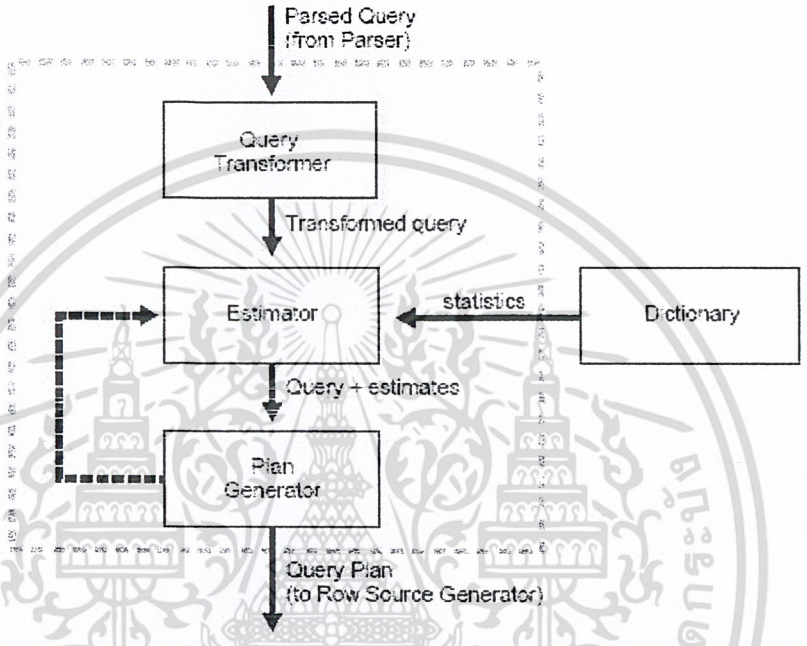
- จำนวนแถวของตารางนั้นๆ
- จำนวนของบล็อกข้อมูลในระดับกายภาพ (physical data blocks)

ข้อมูลของดัชนีได้แก่

- จำนวนค่าที่ไม่ซ้ำกันในดัชนี นั้น
- การกระจายกันของแต่ละค่าในดัชนีนั้น
- วิธีการเลือกใช้ดัชนี
- Index clustering factor

Oracle พยายามสนับสนุนให้ตารางและดัชนีได้ถูกทำการเก็บสถิติใหม่อยู่เสมอ เมื่อมีการเปลี่ยนแปลงข้อมูลที่สำคัญ ในเมื่อแผนการประมวลผลขึ้นอยู่กับข้อมูลทางสถิติแล้วดังนั้นSQLสเตตเมนท์ใดๆอาจจะมีแผนการประมวลผลได้ต่างกันไปในแต่ละครั้งที่ได้ทำการเก็บสถิติใหม่ ทำให้การปรับปรุง SQL สเตตเมนท์เป็นไปได้อย่างยากลำบาก ดังนั้นผู้ใช้งาน Oracle จึงมักจะไม่พอใจกับการที่ต้องหาแผนการประมวลผลของ SQL ที่จะถูกเปลี่ยนแปลงไปนี้ เพื่อไปทำการปรับปรุง SQL สเตตเมนท์นั้นๆ

ส่วนประกอบของตัวประมวลผลแบบอิงสถิติของ Oracle แสดงดังรูปที่ 3.2



รูปที่ 3.2 แสดงส่วนประกอบของตัวประมวลผลแบบอิงสถิติของ Oracle

บทที่ 4

วิธีที่ตัวประมวลผลทำการแปลง SQL สเตตเมนท์

SQL เป็นภาษา query ที่ยืดหยุ่นมาก บ่อยครั้งที่หลายสเตตเมนท์ที่ต่างกันสามารถใช้ในการหาผลลัพธ์เดียวกันได้ ซึ่งบางครั้งตัวประมวลผล (ตัวแปลง Query) จะทำการแปลงสเตตเมนท์หนึ่งไปเป็นอีกสเตตเมนท์หนึ่งที่ได้ผลลัพธ์เดียวกัน ถ้าหากสเตตเมนท์ที่แปลงมาแล้วนี้จะได้รับการประมวลผลได้อย่างมีประสิทธิภาพมากกว่าสเตตเมนท์เดิม

4.1 วิธีที่ตัวประมวลผลแบบอิงสถิติแปลง OR ไปเป็น Compound Queries

หาก Query มีอนุประโยค WHERE หลายเงื่อนไขถูกรวมไว้โดยโอเปอเรเตอร์ OR ดังนั้นตัวประมวลผลจะแปลงมันให้เป็น compound query เสมือน ที่ใช้เซทโอเปอเรเตอร์ UNION ALL ถ้าหากมันทำให้ query สามารถประมวลผลได้อย่างมีประสิทธิภาพยิ่งขึ้น

- หากแต่ละเงื่อนไขทำให้ใช้ดัชนีในการเข้าถึงแผนการประมวลผลได้แล้ว ดังนั้นตัวประมวลผลจะทำการแปลงสเตตเมนท์ตัวประมวลผลแบบอิงสถิติเลือกแผนการประมวลผลสำหรับหาผลลัพธ์ของสเตตเมนท์ที่เข้าถึงตารางหลายครั้ง โดยใช้ดัชนีที่ต่างกันแล้วนำผลลัพธ์มาไว้ด้วยกัน
- หากเงื่อนไขใดต้องการการทำ full table scan เนื่องจากมันไม่มีดัชนีที่ใช้ได้แล้ว ตัวประมวลผลแบบอิงสถิติจะไม่ทำการแปลงสเตตเมนท์แต่ตัวประมวลผลแบบอิงสถิติเลือกที่จะทำ full table scan เพื่อที่จะประมวลผลสเตตเมนท์และ Oracle ตรวจสอบแต่ละแถวในตารางเพื่อตัดสินใจเปรียบเทียบความต้องการในแต่ละเงื่อนไข
- สำหรับสเตตเมนท์ที่ใช้ตัวประมวลผลแบบอิงสถิติ ตัวประมวลผลจะใช้สถิติเพื่อทำการตัดสินใจถึงการทำการแปลงรูป (transformation) โดยตีค่าและเปรียบเทียบต้นทุนการประมวลผลของสเตตเมนท์ดั้งเดิมและสเตตเมนท์ผลลัพธ์

4.2 วิธีที่ตัวประมวลผลแบบอิงสถิติแยก Subqueries

ตัวประมวลผลจะเลือกที่จะหัวข้อใดหัวข้อหนึ่งจากหัวข้อต่อไปนี้ เพื่อทำการปรับปรุงในส่วนของสเตตเมนท์เชิงซ้อน (complex statement)

- ทำการแปลงส่วนสเตตเมนท์เชิงซ้อนไปเป็น join สเตตเมนท์เสมือน แล้วค่อยทำการปรับปรุงในส่วน join สเตตเมนท์
- ทำการปรับปรุงส่วนสเตตเมนท์เชิงซ้อนเลย

ตัวประมวลผลทำการแปลงสเตตเมนท์เชิงซ้อน ไปเป็น join สเตตเมนท์ก็ต่อเมื่อผลลัพธ์ของ join สเตตเมนท์นั้นรับรองได้ว่าจะส่งผลลัพธ์เดียวกันกับสเตตเมนท์เชิงซ้อน ซึ่งการแปลงรูปนี้อนุญาตให้ Oracle ทำการประมวลผลสเตตเมนท์โดยการนำเอาข้อดีของเทคนิคการประมวลผลวิธีการ join มาใช้

4.3 วิธีที่ตัวประมวลผลแบบอิงสถิติรวม Views

เพื่อทำการรวม query ของ view ไปเป็น query บล็อกที่ถูกอ้างถึงในสเตตเมนท์ที่เข้าถึงนั้น ตัวประมวลผลแทนที่ชื่อของ view ด้วยชื่อของตารางหลักของมันใน query บล็อกและทำการเพิ่มเงื่อนไขใน query ของ view ตรงส่วนของอนุประโยค WHERE เพื่อทำการเข้าถึง query บล็อกในส่วนของอนุประโยค WHERE

การ Optimization นี้เกี่ยวข้องกับการ select-project-join views ที่ซึ่งเป็น views ที่มีเฉพาะการ selections, projections, และ joins ซึ่งหมายความว่า views นี้ไม่มีส่วนประกอบอย่างเช่น เซทโอเปอเรเตอร์, aggregate functions, DISTINCT, GROUP BY, และ CONNECT BY

4.3.1 ความสามารถในการรวม Views

ตัวประมวลผลสามารถรวม view ไปเป็น query บล็อกที่ถูกอ้างถึงได้เมื่อ view มีหนึ่งหรือหลายๆ ตารางหลัก โดยมีเงื่อนไขว่า view ต้องไม่ตรงตามข้อใดข้อหนึ่งต่อไปนี้

- มีเซทโอเปอเรเตอร์ (UNION, UNION ALL, INTERSECT, MINUS)
- มีอนุประโยค CONNECT BY
- มีคอลัมน์เทียม (pseudocolumn) ROWNUM
- มี aggregate functions (AVG, COUNT, MAX, MIN, SUM) อยู่ในรายการ SELECT

เมื่อ View มีส่วนหนึ่งของโครงสร้างดังต่อไปนี้แล้ว จะสามารถถูกรวมไปเป็น query บล็อกเสมือนก็ต่อเมื่อได้รับอนุญาตให้ทำการรวมViewเชิงซ้อน

- มีอนุประโยค GROUP BY
- มีโอเปอเรเตอร์ DISTINCT ในรายการ SELECT

ถ้าหาก Query มีเอกเพรสชัน CURSOR ดังนั้นแล้วล่ะก็จะไม่มีการทำการรวม view แม้แต่กับ view ที่ซึ่งสามารถรวมได้ตามปกติ

4.3.2 การรวม View เชิงซ้อน

ถ้าหาก query ของ view นั้นมีอนุประโยค GROUP BY หรือโอเปอเรเตอร์ DISTINCT อยู่ในรายการ SELECT แล้ว ตัวประมวลผลสามารถที่จะรวม query ของ view ไปเป็นสเตตเมนท์ที่เข้าถึงถ้าหากอนุญาตให้ทำการรวม view เชิงซ้อน การรวม view เชิงซ้อนของ view ที่มีอนุประโยค GROUP BY

การรวม view เชิงซ้อนนั้นยังสามารถใช้ทำการรวมตัว IN subquery ได้ถ้าหาก subquery นั้นเป็น uncorrelated subquery

4.4 วิธีที่ตัวประมวลผลแบบอิงสถิติวาง Predicates

ตัวประมวลผลสามารถทำการแปลงส่วน query บล็อกที่เข้าถึง view ที่ไม่สามารถรวมได้ (nonmergeable view) โดยทำการวาง predicates ของ query บล็อกไว้ใน query ของ view

4.4.1 วิธีที่ตัวประมวลผลแบบอิงสถิติใช้งานแอกกริเกรทฟังก์ชั่นกับ View

ตัวประมวลผลสามารถแปลง query ที่มี aggregate functions (AVG, COUNT, MAX, MIN, SUM) โดยการใช้ประโยชน์จากฟังก์ชั่นไปเป็น query ของ view ได้

4.4.2 วิธีที่ตัวประมวลผลแบบอิงสถิติประมวลผล Views ในการทำ Outer Joins

สำหรับ view ที่อยู่บนด้านขวา (right side) ของ outer join แล้ว ตัวประมวลผลสามารถใช้หนึ่งในสองเมธอด (methods) ต่อไปนี้ ซึ่งก็ขึ้นอยู่กับจำนวนตารางหลักที่ view ได้เข้าถึง

- ถ้าหาก view มีเพียงหนึ่งตารางหลักแล้ว ตัวประมวลผลสามารถใช้การรวม view ได้
- ถ้าหาก view มีหลายตารางหลักแล้ว ดังนั้นตัวประมวลผลสามารถวาง join predicate ไปยัง view ได้

4.5 วิธีที่ตัวประมวลผลแบบอิงสถิติประมวลผล Compound Queries

เพื่อเลือกเส้นทางการประมวลผลสำหรับ Compound query ตัวประมวลผลจะเลือกเส้นทางการประมวลผลสำหรับแต่ละ query ที่เป็นส่วนประกอบของมัน และหลังจากนั้นจะนำผลจากแต่ละส่วนมา union, intersection หรือ minus ขึ้นอยู่กับโอเปอเรเตอร์ที่ใช้ใน compound query

บทที่ 5

Oracle Hints

Hints ได้ถูกนำมาใช้ครั้งแรกใน Oracle 7 เพื่อแก้ไขข้อบกพร่องจากการสร้างตัวประมวลผลแบบอิงสถิติในตอนแรกๆ ซึ่ง Oracle พยายามที่จะลดการใช้ hint ลงไปเมื่อแทนที่โดยการปรับปรุงตัวประมวลผลแบบอิงสถิติให้ดีขึ้น แต่การใช้งาน hint ก็ยังคงมีอยู่ใน Oracle8i ซึ่งแนวความคิดในการนำเอา hint มาใช้นั้นเป็นที่โต้เถียงกันกับผู้ใช้งาน SQL ที่คิดว่าตัวประมวลผลแบบอิงสถิติควรจะฉลาดเพียงพอที่จะเลือกเลือกแผนการประมวลผลที่เหมาะสมได้เอง แต่แล้ว hint ก็กลายมาเป็นสิ่งสำคัญในการปรับปรุงคำสั่ง SQL ในท้ายที่สุด

โดยปกติแล้ว hints ถูกใช้งานในสองกรณีคือ

1. ถูกใช้ในการดัดแปลงแผนการประมวลผลสำหรับ SQL สเตตเมนท์
2. Hints สามารถถูกใช้เป็นทางเลือกในการเปลี่ยนแปลงแผนการประมวลผลให้คงรูปแบบเดิมไว้เสมอได้

การใช้ Hints นั้นอนุญาตให้เราทำการตัดสินใจเลือกในสิ่งที่บ่อยครั้งถูกตัวประมวลผลทำการเลือก ซึ่งหากเราเป็นผู้ออกแบบ application แล้ว สิ่งสำคัญคือเราต้องรู้ข้อมูลที่ซึ่งเป็นสิ่งที่ตัวประมวลผลไม่รู้

ยกตัวอย่างเช่นเราต้องรู้ว่าจะใช้ดัชนีไหนที่เหมาะสมกับ qeury ของเราที่สุด ซึ่งจากข้อมูลเหล่านี้เราอาจจะสามารถเลือกแผนการประมวลผลที่ดีกว่าให้ตัวประมวลผลเลือกเองได้ ซึ่งหากเป็นเช่นนั้น เราสามารถใช้ hints เพื่อบังคับให้เลือกใช้แผนการประมวลผลตามแบบของเราได้

เราสามารถใช้ hints เพื่อกำหนดสิ่งต่างๆต่อไปนี้ได้

- กำหนดวิธีการทำ optimization
- กำหนดเป้าหมายของตัวประมวลผลแบบอิงสถิติ
- ใช้ระบุเส้นทางการเข้าถึง (access path) ของตารางที่ถูกเข้าถึงโดย sql สเตตเมนท์
- ลำดับของการ join ของ join สเตตเมนท์
- วิธีการ join ในของ join สเตตเมนท์

Hints ได้จัดเตรียมกรรมวิธีการที่จะแนะนำให้ตัวประมวลผลเลือกแผนการประมวลผลที่ถูกต้องของ query ซึ่งมีพื้นฐานอยู่บนสิ่งต่างๆต่อไปนี้

- ลำดับของการ join
- วิธีการ join
- เส้นทางการเข้าถึง (access path)
- การทำ parallelization

ทุก Hints ยกเว้น RULE hint เรียกใช้ตัวประมวลผลแบบอิงสถิติ ซึ่งหากเราไม่ได้ทำการเก็บสถิติแล้ว ค่ามาตรฐานจะถูกนำมาใช้แทน

Oracle hints นั้นถูกใส่ไว้ใน comment ซึ่งอาจมีปัญหาของการเขียนผิด syntax และจะทำให้ hints ไม่ถูกนำไปใช้โดยที่ไม่มีข้อความแสดงความผิดปกติขึ้นมา ซึ่งเราควรรู้กฎของการใช้ hints ที่สำคัญทั้งสี่ข้อดังต่อไปนี้

- ตรวจสอบการเขียน syntax ของ hints ให้ถูกต้อง เช่นการเขียน /+* hints */ โดยทั่วไปแล้วหมายถึง - - + hints
- เมื่อมีการใช้ชื่อเสมือนของตาราง (alias) เราจะไม่สามารถใช้ชื่อตารางมาเขียนลงใน hints ได้ ซึ่งเราต้องใช้ชื่อเสมือนของตารางมาเขียนลงไปใน hints แทน ยกตัวอย่างเช่น query ดังต่อไปนี้ select /*+ index(e,dept_idx) */ e.emp_name from employees e;
- อย่าอ้างถึงชื่อ schema ลงใน hints จะไม่ถูกนำไปใช้งาน ดังตัวอย่าง query ต่อไปนี้ select /*+ index(scott.employees,dept_idx) */ emp_name from employees;
- การทำให้ hints นำไปใช้งานได้ ซึ่ง hints จะไม่ถูกนำไปใช้งาน หากรู้ว่าเส้นทางการเข้าถึงนั้นไม่สามารถใช้งานได้ ยกตัวอย่างเช่น การระบุให้ใช้ดัชนีบนตารางที่ไม่มีดัชนี หรือว่าการระบุ parallel hint ในการทำ index range scan ก็จะถูกมองข้ามการใช้ดัชนีไป ซึ่งควรจะเอาใจใส่การเขียน hints เป็นพิเศษ เพราะว่าการเขียน hints ที่ขัดแย้งกับ query ไม่เห็นความขัดแย้งกันชัดเจนเสมอไป ยกตัวอย่างเช่นการเขียนคำสั่ง query ดังที่จะแสดงต่อไปนี้
  select /*+ first_rows */ * from emp order by ename;
  ซึ่ง hints ที่เขียนไปจะไม่ถูกนำไปใช้ เพราะว่าการเขียน first_rows optimizer มันไม่รองรับการเขียนร่วมกันกับอนุประโยค ORDER BY เพราะว่าการทำ order by นั้นต้องทำการเรียงข้อมูลก่อน และจะทำการส่งผลลัพธ์ไปได้ก็ต่อเมื่อทำการเรียงข้อมูลเสร็จแล้ว

ซึ่งจะเห็นได้ว่าการเขียน hints ที่ขัดแย้งกับ query จะทำให้ hints ไม่ถูกนำไปใช้งาน ดังนั้นเราควรรู้วิธีการเขียน hints ที่ถูกต้องและไม่ขัดแย้งกันกับการเขียน query ซึ่งตารางที่ 5.1 แสดงถึงการเขียน hints ที่ขัดแย้งกันกับกรรมวิธีการเข้าถึงข้อมูล (access method)

Hints	กรณีที่ Hints จะไม่ถูกนำไปใช้
cluster	เมื่อใช้กับตารางที่ไม่ได้ทำ cluster
hash	เมื่อใช้กับตารางที่ไม่ได้ทำ cluster
hash_aj	เมื่อไม่มีการเขียน subquery
index	เมื่อไม่มีดัชนีที่ระบุ
index_combine	เมื่อไม่มี bitmap index
merge_aj	เมื่อไม่มีการเขียน subquery

ตารางที่ 5.1 เงื่อนไขที่ทำให้ hints ไม่ถูกนำไปใช้งาน

parallel	เมื่อใช้กับแพลนที่ไม่ได้เข้าถึงตารางแบบ full
push_subq	เมื่อไม่มีการเขียน subquery
star	เมื่อมีดัชนีที่ไม่เหมาะสม (improper index) บน fact table
use_concat	เมื่อไม่มีการเขียน OR conditions หลายๆอันในอนุประโยค WHERE
use_nl	เมื่อไม่มีดัชนีบนตารางที่ระบุ

ตารางที่ 5.1(ต่อ) เงื่อนไขที่ทำให้ hints ไม่ถูกนำไปใช้งาน

5.1 Optimizer Hints

การเขียน optimizer hints ทำเพื่อทำการเลือกวิธีการประมวลผลและจุดมุ่งหมายของการทำ optimize ซึ่งการที่เราเขียน optimizer hints ลงไปเป็นเพียงการบอกให้ Oracle เลือกใช้ตัวประมวลผลเท่านั้น ดังนั้นเรายังสามารถเพิ่ม hints อื่นๆเพิ่มต่อไปได้อีก

5.1.1 all_rows Hint

all_rows hints ใช้บอกให้ใช้ตัวประมวลผลแบบอิงสถิติ โดยมีเป้าหมายอยู่ที่ throughput ที่ดีที่สุด และเลือกใช้งานทรัพยากร (resource) ที่น้อยที่สุด ซึ่งเหมาะกับการที่มีการทำ full table scan แต่ไม่เหมาะกับการใช้งานกับฐานข้อมูลแบบ OLTP (Online Transaction Processing Databases) เมื่อมีการใช้ hints นี้กับฐานข้อมูลแบบอิงกฎ (rule_based databased) แล้วควรแน่ใจว่าสถิติของตารางและดัชนีที่เกี่ยวข้องอยู่ใน query ได้ถูกจัดเก็บแล้ว

5.1.2 rule Hint

การเขียน hints นี้บอกให้ Oracle ใช้งานตัวประมวลผลแบบอิงกฎกับ query นั้น ซึ่ง hints นี้เป็น hints ที่นิยมใช้ในการปรับปรุง SQL สเตตเมนท์ เมื่อเราแน่ใจว่าตัวประมวลผลแบบอิงสถิติจะสร้างแผนการประมวลผลที่แย่กว่าออกมา

การเขียน hints นี้จะทำให้ตัวประมวลผลไม่ใช้งานสถิติของตารางและดัชนี และใช้ heuristic พื้นฐานในการสร้างแผนการประมวลผล ซึ่งใน Oracle8i มีบ่อยครั้งที่ตัวประมวลผลแบบอิงกฎสร้างแผนการประมวลผลที่เร็วกว่าแผนการประมวลผลที่ตัวประมวลผลแบบอิงสถิติสร้างมา

5.1.3 first_rows Hint

การเขียน hints นี้จะบอกให้เลือกใช้ตัวประมวลผลแบบอิงสถิติ โดยมีเป้าหมายอยู่ที่เวลาในการตอบสนองที่เร็วที่สุด เช่นเดียวกับการเขียน all_rows hints คือควรแน่ใจว่าสถิติของตารางและดัชนีที่เกี่ยวข้องอยู่ใน query ได้ถูกจัดเก็บแล้ว

first_rows hint มีประโยชน์ในการทดสอบแผนการประมวลผลสำหรับฐานข้อมูลที่ optimizer mode ไม่แน่นอน เช่นการใช้ optimizer mode เป็น CHOOSE โดยการเลือกใช้ first_rows hint สามารถปรับ SQL ที่ซึ่งตัวประมวลผลแบบอิงสถิติพิจารณาหาวิธีการให้เวลาการตอบสนองที่เร็วที่สุด

5.2 Table Join Hints

Oracle จัดเตรียม hints ที่ใช้สำหรับการกำหนดวิธีการ join table ในหลายๆวิธี ซึ่ง Oracle สามารถทำการ join ได้แก่ nested loop join, sort merge join, hash join และ star join เนื่องจากจะเสียเวลามากเมื่อทำการ join ดังนั้น Oracle จึงมี hints สำหรับการ join ตารางที่ถูกใช้งานบ่อยครั้งในการทดสอบความเร็วในการประมวลผลเทคนิคการ join หลายๆแบบ

5.2.1 use_hash Hint

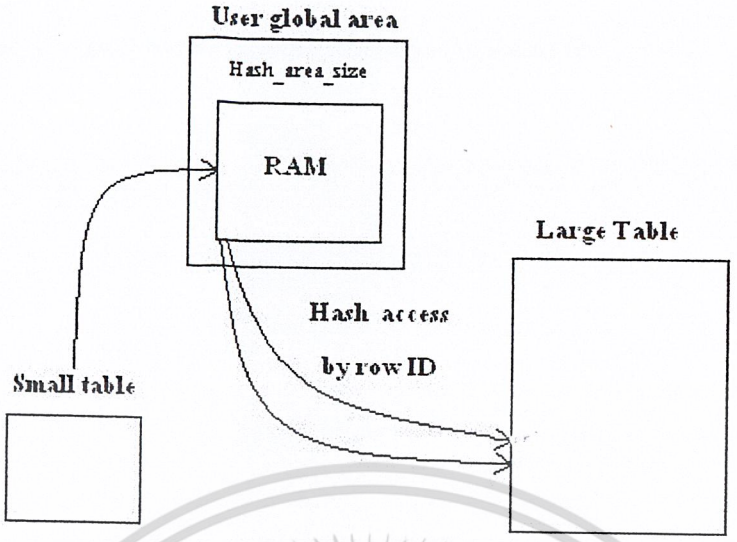
การใช้งาน use_hash hint ทำการร้องขอให้มีการทำ hash join กับตารางที่เรากำหนด จุดสำคัญก็คือ hash join เป็นเทคนิคที่ซึ่ง Oracle ทำการโหลดแถวทั้งหมดจาก driving table ซึ่งในกรณีนี้คือตารางแรกในอนุประโยค FROM ลงไปยังพื้นที่ RAM (Random access Memories) ที่ถูกกำหนดไว้โดยค่าพารามิเตอร์ hash_area_size

Oracle จะใช้เทคนิคของการ hash เพื่อชี้ตำแหน่งแถวในตารางที่สองซึ่งกว้างกว่า ส่วนใหญ่แล้วมักใช้ควบคู่ไปกับ parallel query ในกรณีที่ตารางทั้งสองตารางมีจำนวนแถวมากๆ

ตัวอย่างของการเขียน use_hash hint สามารถเขียนได้ดังนี้

```
SELECT /*+ use_hash(e,b) */ e.emp_name, hiredate, b.comm
FROM employees e,bonus b
WHERE e.emp_name=b.emp_name
```

รูปที่ 5.1 แสดงวิธีการทำ hash join และการใช้งาน RAM ซึ่งจะใช้ตารางที่เล็กเป็น driving table ทำการโหลดลงไปใน hash table ก่อน แล้วค่อยทำการ join กับตารางใหญ่



รูปที่ 5.1 แสดงการทำ hash join และการใช้งาน RAM

5.2.2 use_merge Hint

การใช้งาน use_merge hint ทำการร้องขอให้มีการทำ sort merge join ซึ่งบ่อยครั้งถูกใช้ร่วมไปกับการทำ parallel query เพราะว่าการทำ sort merge join จะทำการ full table scan ทุกครั้ง การทำ sort merge join เหมาะกับการที่ query ได้ผลลัพธ์จำนวนมากๆ เช่นการทำรายงานประจำวันหรือตารางที่ไม่ได้ใช้ดัชนีในการ join

สิ่งสำคัญคือเราควรรู้ว่าการทำ sort merge join นั้นไม่ได้ใช้ดัชนีในการ join ตาราง ซึ่งในส่วนมากแล้ว การเข้าถึงโดยใช้ดัชนีจะเร็วกว่า แต่ว่า sort merge join เหมาะกับตารางที่มีขนาดใหญ่ที่ทำการ join โดยไม่มีอนุประโยค WHERE หรือ query ที่ไม่มีดัชนีให้ใช้ในการ join ตาราง

ตัวอย่างของการเขียน use_merge hint สามารถเขียนได้ดังนี้

```
SELECT /*+ use_merge(e,b) */ e.emp_name, hiredate, b.comm
FROM employees e,bonus b
WHERE e.emp_name=b.emp_name
```

5.2.3 use_nl Hint

การใช้งาน use_nl hint ทำการร้องขอให้มีการทำ nested loop join กับตารางที่เรากำหนด ซึ่งไม่เหมือนการ join แบบอื่นๆที่ต้องระบุตารางทั้งสองตาราง ซึ่งการใช้งาน use_hash hint ต้องการเพียงแค่ชื่อตารางที่จะถูกใช้เป็น driving table ซึ่งก็คือตารางแรกในอนุประโยค FROM ในการใช้งานตัวประมวลผลแบบอิงสถิติ การทำ nested loop join เป็นวิธีการ join ที่เก่าแก่ที่สุดและถูกใช้กับการประมวลผลแบบอิงกฎอยู่เสมอ

การใช้งาน use_nl hint นั้นถูกใช้ไม่บ่อยนักในการทำการปรับปรุงคำสั่ง SQL เนื่องจากทั้งตัวประมวลผลแบบอิงกฎและตัวประมวลผลแบบอิงสถิตินั้นมักจะนิยมใช้ nested loop join มากกว่า hash join หรือ sort merge join อยู่แล้ว อย่างไรก็ดีการใช้งาน use_nl hint บางครั้งก็มีประโยชน์ในการเปลี่ยนแปลง driving table โดยไม่ต้องทำการเปลี่ยนลำดับของตารางในอนุประโยค FROM เพราะว่าไม่จำเป็นว่าตารางไหนควรจะเป็น drivier table เสมอไป บางครั้งในการเปรียบเทียบจำนวนแถวของทั้งสองตาราง เราสามารถเพิ่มประสิทธิภาพของการทำ nested loop join โดยการเปลี่ยน driving table สำหรับการ join ได้

ตัวอย่างของการเขียน use_nl hint สามารถเขียนได้ดังนี้

```
SELECT /*+ use_nl(e) */ e.emp_name, hiredate, b.comm
FROM employees e,bonus b
WHERE e.emp_name=b.emp_name
```

5.2.4 star Hint

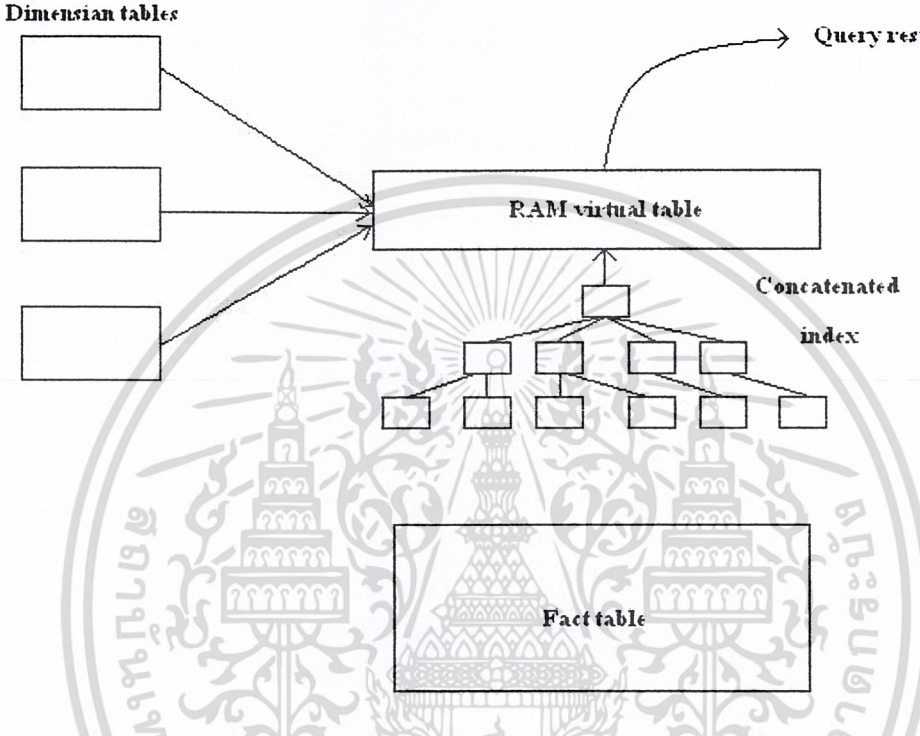
การใช้งาน star hint ทำการร้องขอให้มีการทำ star query plan การจะใช้งานได้จำเป็นต้องมีอย่างน้อยสามตารางใน query และมีดัชนีอยู่ใน fact table ด้วย การใช้งาน star hint นั้นทำงานไวกว่าวิธีการ join อื่นๆที่ได้กล่าวมา โดยการใช้ตารางที่เล็กที่สุดเทียบกับ fact table และทำการ join แต่ละตารางอ้างอิง (reference table) กับ intermediate table อย่างไรก็ตาม การจะใช้งาน star hint นั้นมีหลายเงื่อนไขต่อไปนี้

- ต้องมีอย่างน้อยสามตารางที่ถูก join เข้าด้วยกัน โดยมี fact table ขนาดใหญ่อยู่หนึ่งตารางกับพวกตาราง dimension ที่เล็กกว่า
- ต้องมีดัชนีบน column ของ fact table ซึ่งถูกใช้เป็น join keys ซึ่งใน Oracle8i นั้น bitmap index นั้นถูกนำมาใช้แทน concatenated index
- ต้องแน่ใจว่าทำการ join โดยการทำ nested loop join

การทำ star join นั้นเริ่มจากการที่ Oracle ทำ catesian product ของ dimention tables ทั้งหมดเข้าด้วยกันแล้วเก็บผลลัพธ์ทั้งหมดลงในหน่วยความจำของ Oracle ซึ่งตารางเสมือนนี้จะเก็บทุก columns ของทุกๆ dimention tables ไว้ดังรูปที่ 5.2 ซึ่งprimary key ของตารางเสมือนนี้จะเป็น composite key จาก primary key ของ dimention tables และถ้าหาก keys นี้จับคู่ได้กับ composite index บน fact table จะทำให้

query นี้ process ได้ไวขึ้น ถ้าหากผลรวมของตารางอ้างอิงได้ถูกนำไปใช้แล้ว Oracle จะทำ nested loop join ระหว่าง intermediate table กับ fact table

ความเร็วในการทำ star join นี้จะช่วยลดการใช้งาน physical I/O ลงไปได้ ซึ่งดัชนีจะถูกอ่านเพื่อทำการรวบรวม virtual table ใน memory และ fact table จะไม่ถูกเข้าถึงจนกว่า virtual index จะมีสิ่งที่ต้องการครบ เพื่อทำการเข้าถึงแถวที่ร้องขอโดยตรงผ่าน composite key บน fact table



รูปที่ 5.2 star table join

5.3 Table Anti-Join Hints

การทำ anti-join ได้ถูกใช้เมื่อใน SQL สเตตเมนท์มีอนุประโยค NOT IN หรืออนุประโยค NOT EXISTS ปรากฏอยู่ ซึ่งจะเป็นการเสียเวลาในการที่จะทำการกรองแถวจากตารางหนึ่งที่ซึ่งไม่มีอยู่ในอีกตารางหนึ่ง ซึ่ง Oracle Hints ที่สามารถช่วยทำงานให้เกิดประสิทธิภาพได้คือ anti join hint

5.3.1 merge_aj Hint

การใช้งาน merge_aj hint นั้นถูกใส่เข้าไปใน NOT IN subquery เพื่อทำ anti-join ที่ซึ่ง full-table access ได้ใช้งานแทนที่การเข้าถึงด้วยดัชนี

ซึ่งการใช้งาน merge_aj hint นั้นเป็นวิธีที่ดีกว่าเมื่อใช้กับ NOT IN subquery แต่ต้องเป็นเฉพาะกรณีที่คอลัมน์นั้น ต้องมี NOT NULL constraint ด้วย

ตัวอย่างของการใช้ merge_aj hint แสดงดังตัวอย่างต่อไปนี้ โดยที่คอลัมน์ deptno นั้นมี NOT NULL constraint

```
SELECT dname
FROM dept
WHERE deptno NOT IN ( SELECT /*+ merge_aj */ deptno
                      FROM employees
                      WHERE job = 'SALEMAN') ;
```

5.3.2 hash_aj Hint

การใช้งาน hash_aj hint นั้นถูกใส่เข้าไปใน NOT IN subquery เพื่อทำ anti-join ในกรณีที่ต้องการใช้ hash join ซึ่งการใช้งาน hash_aj hint นั้นใช้กับ NOT IN subquery และต้องเป็นเฉพาะกรณีที่คอลัมน์นั้น ต้องมี NOT NULL constraint ด้วย

ตัวอย่างของการใช้งาน hash_aj hint แสดงดังตัวอย่าง

```
SELECT dname
FROM dept
WHERE deptno NOT IN ( SELECT /*+ hash_aj */ deptno
                      FROM employees
                      WHERE job='SALEMAN');
```

5.4 Index Hints

การใช้งาน index hints เป็นประโยชน์มากในการปรับปรุงคำสั่ง SQL ตัวอย่างเช่นในกรณีที่ตัวประมวลผลคำสั่ง SQL เลือกใช้ดัชนีไม่ถูกต้อง ซึ่งจะเกิดบ่อยครั้งในกรณีที่ใช้งานกับตัวประมวลผลแบบอิงกฎ แต่ก็มีบางกรณีที่เกิดขึ้นกับตัวประมวลผลแบบอิงสถิติด้วยเช่นกัน

5.4.1 index Hint

index hint ถูกใช้โดยการระบุชื่อตารางอย่างชัดเจน ที่ซึ่งตัวประมวลผลคำสั่ง SQL จะเลือกดัชนีที่ดีที่สุดของตาราง หรือระบุชื่อตารางและชื่อดัชนีในกรณีที่ต้องการให้ตัวประมวลผลคำสั่ง SQL ใช้ดัชนีที่เรากำหนด

ซึ่งการใช้งาน index hint นั้นต้องเป็นไปตามกฎดังนี้

- ถ้าหากชื่อตารางหรือชื่อของดัชนีสะกดผิดแล้ว hint จะไม่ถูกเรียกใช้
- ชื่อของตารางต้องอยู่ใน hint ด้วย จะใส่แต่ชื่อดัชนีไม่ได้
- หากเราทำการตั้งชื่อเสมือนของตารางแล้ว ใน index hint เราต้องใช้ชื่อเสมือนของตารางด้วย จะใช้ชื่อตารางไม่ได้
- ชื่อดัชนีนั้นเป็นเพียงทางเลือก ซึ่งจะใส่หรือไม่ใส่ก็ได้ หากไม่ได้กำหนดตัวประมวลผลคำสั่ง SQL จะทำการเลือกดัชนีที่คิดว่าดีที่สุดให้

สิ่งสำคัญคือเราควรกำหนดทั้งชื่อตารางและดัชนีลงไปด้วย เนื่องจากมันมีโอกาสไม่มากที่สถิติของตัวประมวลผลแบบอิงกฎจะเปลี่ยนแปลงไปแล้วทำให้เลือกใช้ดัชนีที่ต่างจากเดิม

5.4.2 index_join Hint

index_join Hint นั้นสั่งให้ตัวประมวลผลคำสั่ง SQL เลือกใช้ index join เป็นเส้นทางการเข้าถึงข้อมูลซึ่งจะได้ผลที่ดีกว่าเมื่อมันมีดัชนีทุกๆคอลัมน์ที่เราต้องใช้ในการประมวลผล query นั้น

5.4.3 and_equal Hint

and_equal Hint ถูกใช้เมื่อตารางมีหลาย nonunique single column index และต้องการดัชนีหลายตัวในการประมวลผล query นั้น ซึ่ง and_equal Hint จะทำการเชื่อมดัชนีและแยกสร้างดัชนีถ้าหากมันเป็น concatenation index เดียว

and_equal Hint ต้องการการระบุชื่อตารางกับชื่อดัชนีอย่างน้อยสองดัชนี และต้องไม่เกินจำนวนห้าดัชนี

การใช้งาน and_equal Hint แสดงดังตัวอย่าง ซึ่งสมมติว่ามี nonunique index บน job และ mgr

```
SELECT /*+ and_equal(employees, job_idx, mgr_idx) */ emp_name, job,deptno, mgr
FROM employees
WHERE job = 'SALESMAN'
AND mgr = 7698 ;
```

ซึ่งการเขียน hint เพิ่มเข้าไปนี้จะทำให้ตัวประมวลผลคำสั่ง SQL ทำการindex range scan บนทั้งสองดัชนี

5.4.4 index_asc Hint

index_asc hint ร้องขอให้มีการใช้ ascending index บนการทำ range scan ซึ่งโดยปกติในการทำ index range scan ก็กระทำเช่นนี้อยู่แล้ว ดังนั้นเราจึงไม่จำเป็นต้องทำการกำหนด hint ชนิดนี้

5.4.5 no_index Hint

no_index hint ทำการบังคับให้ตัวประมวลผลไม่ใช้ดัชนีที่มีอยู่ในการเข้าถึงข้อมูล ซึ่งจะใช้มากในกรณีที่รู้ว่าการทำ parallel full table scan จะไวกว่าการทำ index range scan ซึ่ง hint นี้มีความหมายเท่ากับ full hints ซึ่งมีประโยชน์อย่างมากในการทำการปรับปรุงคำสั่ง SQL

5.4.6 index_desc Hint

index_asc hint ร้องขอให้มีการใช้ descending index บนการทำ range scan ซึ่งจะมีประสิทธิภาพที่ดีขึ้นในกรณีที่ทำการคำนวณหาค่าสูงสุดของคอลัมน์ที่ใช้ build-in function max

ตัวอย่างของการใช้งาน index_desc hint แสดงดังต่อไปนี้

```
SELECT /*+ index_desc(employees, sal_idx) */ emp_name, max(salary)
FROM employees
GROUP BY emp_name;
```

5.4.7 index_combine Hint

index_combine hint ถูกใช้เพื่อบังคับให้ทำ bitmap access path บนตารางนั้น ถ้าหากไม่ได้ระบุชื่อดัชนีลงใน hint แล้ว ตัวประมวลผลคำสั่ง SQL จะเลือก boolean combination ใดๆก็ตามของ bitmap index ที่มีต้นทุนต่ำที่สุดสำหรับตารางนั้น index_combine hint บอกให้ตัวประมวลผลคำสั่ง SQL ทำการ intersection ROWID จาก bitmap index ทั้งคู่

การที่เราทำการรวม bitmap index นั้นจะช่วยลดเวลาในการประมวลผลของ query ที่มีขนาดตารางใหญ่ๆลงไปได้มาก

5.4.8 index_ffs Hint

เมื่อใช้ index_ffs hint ตัวประมวลผลคำสั่ง SQL จะทำการแสกนทุกๆบล็อกในดัชนี โดยใช้การอ่านแบบ multiblocks และทำการเข้าถึงข้อมูลแบบไม่มีลำดับ ซึ่งสามารถใช้ร่วมกับ parallel hint เพื่อเพิ่มความเร็วได้ ซึ่ง index fast full scan นั้นแตกต่างกันกับ full-index scan

การใช้งาน index_ffs เป็นประโยชน์ในกรณีที่ตารางมีขนาดใหญ่มากๆ เมื่อจะทำการสร้างดัชนีใหม่นั้นต้องการพื้นที่ดิสก์จำนวนมาก ในกรณีที่ตารางมีขนาดใหญ่มากๆและไม่มี high-level index ที่ต้องใช้ในการหาคอลัมน์นั้น fast full-index scan จะไวกว่าการทำ full table scan เสมอ

5.4.9 Use_concat Hint

การกำหนด use_concat hint นั้นจะทำการร้องขอให้ใช้แผนการประมวลผล union all กับทุกๆ OR condition ใน query แล้วทำการเขียน query นั้นขึ้นมาใหม่ในรูปแบบของ multiple query การใช้งาน use_concat นิยมใช้เมื่อSQL query นั้นมีหลาย OR condition ในอนุประโยค WHERE

การใช้งาน use_concat hint นั้นจะทำให้การประมวลผลแบ่งเป็น query ย่อยๆ ทำการประมวลผลบนคอลัมน์เดียวแล้วนำมาทำการ concatenation เพื่อทำการ union all ผลลัพธ์ของquery ย่อยๆเข้าด้วยกัน เช่นการ access ผ่าน B-tree index บนคอลัมน์เดียวของตาราง แล้วนำเอาผลลัพธ์ที่ได้มาทำการ concatenation กัน จะดีกว่าการทำ full table scan บนหลายๆคอลัมน์

5.5 Parallel Hints

parallel hints สามารถช่วยเพิ่มประสิทธิภาพของการทำ full table scan ได้ ซึ่งต้องการการใช้งานกับเครื่อง server ที่มีหน่วยประมวลผลกลาง (Central Processing unit : CPU) หลายๆตัว เพื่อทำการกำหนดจำนวนการทำ parallel

5.5.1 parallel hint

สำหรับการทำ full table scan นั้นการเขียน parallel hint ต้องระบุชื่อตารางที่จะทำการประมวลผลใน parallel mode และจำนวนของการทำ parallel query ระบุไว้ด้วย มันจำเป็นด้วยว่าต้องทำการใส่ full hint เพื่อให้แน่ใจว่ามีการทำ full table scan เนื่องจากว่า parallel hint จะไม่ถูกเรียกใช้งานหากไม่มีการทำ full table scan

ตัวอย่างต่อไปนี้แสดงถึงการใช้งาน parallel hint กับเครื่อง server ฐานข้อมูลที่มีหน่วยประมวลผลกลาง36 ตัว

```
SELECT /*+ full parallel( employees,35) */ emp_name
FROM employees;
```

5.5.2 noparallel Hint

noparallel hint ใช้ในกรณีที่ไม่ต้องการให้มีการทำ parallel processing ในการทำ full table scan ซึ่งอาจจะมีในกรณีที่ทำ full table scan ตารางขนาดเล็ก ซึ่งโดยปกติแล้วตัวประมวลผลแบบอิงสถิติจะคิดว่าตารางมีขนาดเล็ก และจะไม่ใช้งาน parallel query อยู่แล้ว

5.6 Table Access Hints

Table access hint ใช้เพื่อระบุวิธีการเข้าถึงข้อมูลของตาราง (table access method) ซึ่งสามารถใช้เพื่อบอกขั้นตอนการเข้าถึงตารางหลายๆตารางใน query ได้

5.6.1 full Hint

full hint ใช้เพื่อให้ทำการ full table scan โดยที่ไม่ใช้งานดัชนีที่มีอยู่ของตาราง ซึ่งถูกใช้ร่วมกับ parallel hint บ่อยครั้ง เราใช้งาน full hint เมื่อทราบว่าการทำ index range scan จะเสียเวลากว่าการทำ full table scan และการทำ parallel query จะช่วยเพิ่มประสิทธิภาพของการ query ได้

5.6.2 ordered Hint

ในตัวประมวลผลแบบอิงสถิตินั้น การใช้งาน ordered hint ทำการบอกให้ตารางได้ถูก join ตามลำดับที่เขียนในอนุประโยค FROM ที่ซึ่งตารางแรกในอนุประโยค FROM ใช้ระบุเป็น driving table

Ordered hint นั้นได้ถูกใช้ร่วมกันกับ hints อื่นๆบ่อยครั้ง เพื่อให้แน่ใจว่าตารางทั้งหลายได้ถูก join ตามลำดับที่เหมาะสม เช่นถ้าเราทำ query ที่ทำการ join ตารางทั้งหมดห้าตาราง เราต้องการให้มีการทำ hash join ในตารางที่เรากำหนด และทำ nested loop join กับตารางที่เหลือ ซึ่งการใช้ ordered hint นั้นถูกใช้อย่างกว้างขวางในการปรับปรุง data warehouse query ที่มีการ join ตารางมากกว่าสี่ตารางเข้าด้วยกัน

บทที่ 6

การปรับปรุงQUERY

6.1 Driving Table

ตารางที่เป็น driving table นั้น จะเป็นตารางแรกที่ถูก access เพื่อหาผลลัพธ์แล้วนำไปใช้กับตารางอื่นๆที่เหลือ ดังนั้นการเลือก driving table ที่ถูกต้องย่อมมีส่วนช่วยปรับปรุงคำสั่ง sql statement ได้

6.1.1 ตำแหน่งของ Driving Table

ตำแหน่งของ driving table ในการประมวลผลแบบอิงกฎและการประมวลผลแบบอิงสถิตินั้นสลับกันอยู่ สำหรับตัวประมวลผลแบบอิงกฎนั้น ตารางที่อยู่ท้ายสุดของอนุประโยค FROM จะเป็น driving table และควรเป็นตารางที่มี rows น้อยที่สุดด้วย ซึ่งตัวประมวลผลแบบอิงกฎใช้ driving table นี้เป็นตารางแรกเมื่อทำ nested loop join operations แต่ในตัวประมวลผลแบบอิงสถิตินั้น driving table จะเป็นตารางแรกในอนุประโยค FROM ส่วนในกรณีที่มีการกำหนด ORDERED hint ระบุไว้นั้น Oracle จะใช้ตารางแรกในอนุประโยค FROM เป็น driving table

6.1.2 การเปลี่ยน Driving Table ของ Rule-Based

ในตัวประมวลผลแบบอิงกฎของ Oracle นั้น ลำดับของชื่อตารางในอนุประโยค FROM เป็นตัวกำหนด driving table ซึ่ง driving table นั้นมีความสำคัญมาก เพราะว่ามันจะถูกใช้เป็นตารางแรก จากนั้น rows ของตารางที่สองจะถูกรวมเข้าไปกับเซทผลลัพธ์ (result set) ของตารางแรก ซึ่ง driving table ไม่จำเป็นที่จะต้องเป็นตารางที่มีจำนวน rows ที่น้อยที่สุดเสมอไป โดยที่เราต้องทำการดูค่า rows return โดยเปรียบเทียบจากเงื่อนไขในอนุประโยค WHERE ด้วย ซึ่งหมายความว่า driving table ควรเป็นตารางที่ return rows น้อยที่สุดเมื่อดูเทียบจากอนุประโยค WHERE ด้วย

ปัญหาที่แท้จริงของการกำหนดลำดับของ table ในอนุประโยค FROM เพื่อทำการกำหนด driving table ก็คือในกรณีของตัวประมวลผลแบบอิงกฎนั้น ตารางจะถูกอ่านจากขวาไปซ้าย ซึ่งจะหมายความว่าตารางที่อยู่ท้ายสุดของอนุประโยค FROM จะเป็น driving table และควรเป็นตารางที่ return rows น้อยที่สุดด้วย

6.2 การปรับปรุงการทำ Catesian product

เราสามารถปรับปรุงการทำ catesian product operation ให้เร็วขึ้นได้ โดยการเลือก driving table ให้เหมาะสม โดยที่ driving table ควรเป็นตารางที่ให้ผลลัพธ์จำนวน rows น้อยที่สุดใน query นั้น ซึ่งลำดับของ driving table นั้น Oracle มีอยู่2แบบซึ่งแตกต่างกันตาม optimizer ซึ่งในตัวประมวลผลแบบอิงกฎนั้น ตารางสุดท้ายในอนุประโยค from จะถูกใช้เป็น driving table ในทางกลับกันแล้ว ใน cost-based optimizer นั้น driving table จะเป็นตารางแรกที่ระบุไว้ใน from clause ซึ่งหากมี query ที่เป็น catesian product อยู่แล้ว เราสามารถทำการปรับปรุงให้มีการ query ได้เร็วขึ้นได้โดยการเปลี่ยนตารางที่ได้ผลลัพธ์จำนวน rows น้อยที่สุดมาเป็น driving table ซึ่งตำแหน่งของ driving table เราต้องกำหนดตาม optimizer นั้นๆให้ถูกต้อง ซึ่งเราจะทำการเพิ่ม RULE hint เข้าไปเพื่อให้เราสามารถกำหนดตารางที่ได้ผลลัพธ์ rows ที่น้อยที่สุดไว้เป็นตารางสุดท้ายใน from clause ตลอดได้โดยไม่ต้องกังวลว่าขณะนั้นใช้ optimizer แบบใดอยู่ได้ เนื่องจาก RULE hint จะทำการบังคับให้oracleทำการเลือกใช้ rule-based optimizer

6.3 ลดการเสียเวลาในการหาลำดับการjoinตารางโดยใช้ORDERED Hint

ใน cost-based optimizer นั้น ORDERED hint จะทำตารางถูก join ตามลำดับที่ระบุไว้ใน from clause ซึ่งตารางแรกใน from clause จะถูกใช้เป็น driving table

ORDERED hint ส่วนมากจะถูกใช้ร่วมกับ hint อื่นเพื่อให้แน่ใจว่าตารางทั้งหมดได้ถูกใช้ join อย่างต้องตามลำดับ ซึ่ง ordered hint จะถูกใช้มากในการtuning data warehouse queries ที่มีการ join ตารางมากกว่า4ตารางเข้าด้วยกัน

เมื่อมีการ join หลายๆตารางเช่น7ตารางหรือมากกว่านั้นขึ้นไป บ่อยครั้งใช้เวลาในการส่ง SQL มากกว่า30นาทีขึ้นไป เนื่องจากว่า Oracle จะคำนวณหาลำดับของการ join ที่เป็นไปได้ทั้งหมด ยกตัวอย่างเช่นเมื่อมีการ join ตาราง8ตารางเข้าด้วยกันใน query แล้ว Oracle จะทำการหาลำดับของการ join ที่เป็นไปได้ทั้งหมดซึ่งก็คือ8!หรือ40,320ทางที่เป็นไปได้ซึ่งจะเป็นการเสียเวลามากในการที่จะมาหาลำดับการ join ที่ดีที่สุดจากวิธีการทั้งหมดที่เป็นไปได้ เราจึงใช้ ORDERED hint เพื่อทำการ bypass การเสียเวลาในส่วนที่ทำการหาลำดับของการ join ก่อนที่จะทำการส่งคำสั่ง SQL ซึ่งหากเรารู้ลำดับของการ join ที่ดีที่สุดอยู่แล้ว เราสามารถใช้ ORDERED hint เขียนเพิ่มไปใน query เพื่อลดเวลาลงไปได้

ดังนั้นเมื่อผู้ใช้ทำการ query คำสั่งSQLที่มีการ join หลายๆตาราง เราสามารถทำการเรียงลำดับของ table ใน from clause เพื่อให้ driving table เป็นตารางที่ให้ผลลัพธ์จำนวน rows ที่น้อยที่สุด ซึ่งหากเราทำการใช้ ORDERED hint แล้ว driving table จะเป็นตารางแรกใน from clause (ตามรูปแบบของ cost-based optimizer) ซึ่งหมายความว่าทำการสลับเอาตารางที่ให้ผลลัพธ์จำนวน rows ที่น้อยที่สุดมาเป็นตารางแรกใน from clause นั่นเอง แล้วทำการเพิ่ม ORDERED hint เข้าไปใน query จะทำให้เราลดเวลาที่เสียไปจากการหาลำดับการ join ที่ดีที่สุดออกไปได้ และยังทำให้เรากำหนด driving table ได้ถูกต้องเพื่อให้สามารถ query ได้เร็วขึ้นอีกด้วย

6.4 การใช้งาน USE_CONCAT Hint เพื่อให้มีการทำ Union All

การกำหนด use_concat hint นั้นจะทำการร้องขอให้ใช้แผนการประมวลผล union all กับทุกๆ OR condition ใน query แล้วทำการเขียน query นั้นขึ้นมาใหม่ในรูปแบบของ multiple query การใช้งาน use_concat นิยมใช้เมื่อSQL query นั้นมีหลาย OR condition ในอนุประโยค WHERE

การใช้งาน use_concat hint นั้นจะทำให้การประมวลผลแบ่งเป็น query ย่อยๆ ทำการประมวลผลบนคอลัมน์เดียวแล้วนำมาทำการ concatenation เพื่อทำการ union all ผลลัพธ์ของquery ย่อยๆเข้าด้วยกัน เช่นการ access ผ่าน B-tree index บนคอลัมน์เดียวของตาราง แล้วนำเอาผลลัพธ์ที่ได้มาทำการ concatenation กัน จะดีกว่าการทำ full table scan บนหลายๆคอลัมน์

กรณีที่USE_CONCAT hintจะถูกมองข้ามไป คือกรณีที่ ไม่มี OR condition หลายๆ condition ใน where clause ซึ่งหากเราตรวจสอบได้ว่ามีการการทำ OR condition หลาย condition อนุประโยค WHERE แล้ว และทุก condition สามารถถูก access ผ่าน index ได้ เราสามารถใช้ USE_CONCAT hint เพื่อให้มีการทำแผนการประมวลผล union all ได้

บทที่ 7

การทำงานและโครงสร้างของระบบ

7.1 ส่วนติดต่อกับผู้ใช้(User interface)

หน้าตาของโปรแกรมจะออกแบบโดยเน้นความง่ายในการใช้งานและง่ายต่อการทำความเข้าใจ ประกอบไปด้วยสามส่วนหลักดังนี้

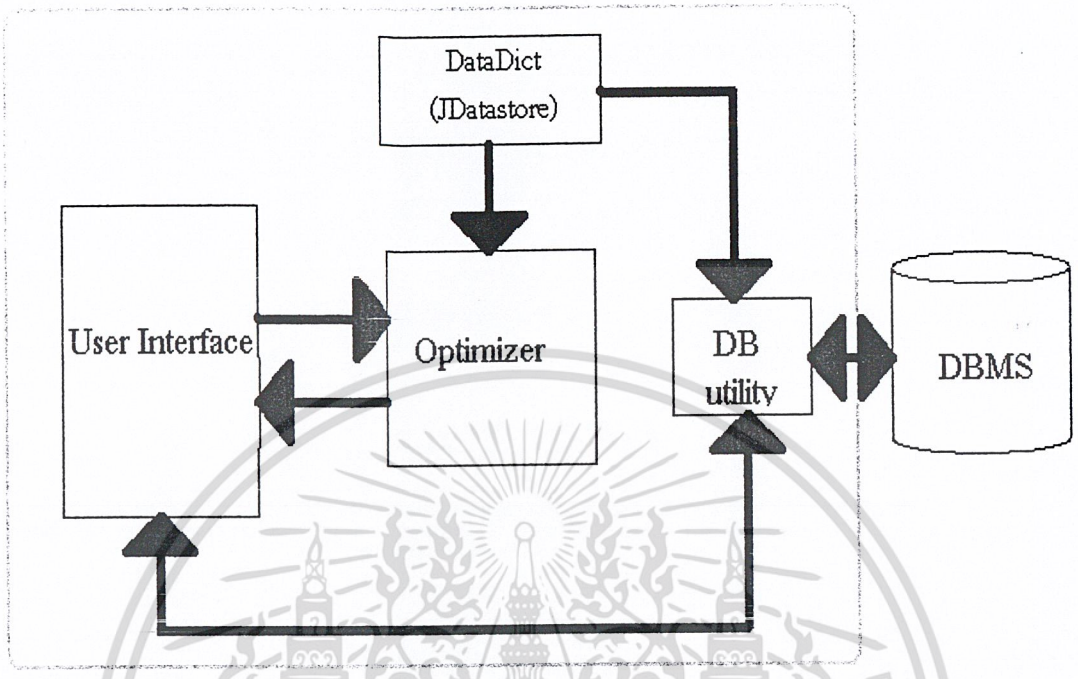
1. ส่วนปุ่มควบคุมต่างๆ
   1.1. ปุ่ม connect สำหรับ connect เข้ากับ DBMS เมื่อกดปุ่มนี้จะมี Dialog ขึ้นมาให้ป้อน user และ password
   1.2. ปุ่ม dynamic มีไว้สำหรับป้อนคำสั่ง sql ที่ต้องการลงไป
   1.3. ปุ่ม analyze ใช้สำหรับเก็บแก้ไข้ข้อมูลทางสถิติ และเก็บสถิติเพิ่มสำหรับตารางที่ยังไม่ถูกเก็บสถิติ
   1.4. ปุ่ม exit ใช้สำหรับออกจากโปรแกรม
2. ส่วนแสดงคำสั่ง sql จะมีไว้เพื่อเปรียบเทียบระหว่างคำสั่งทั้งที่ optimize แล้ว และยังไม่ optimize ทั้งสองหน้าจอนี้ จะแสดงทั้งสองคำสั่ง
3. ส่วนแสดงผลลัพธ์ของการ query และแสดงข้อมูลทางสถิติที่ได้จัดเก็บมา

7.2 การทำงานของระบบ

เริ่มต้นเมื่อผู้เปิดโปรแกรมขึ้นมาใช้ จะต้องกดปุ่ม connect ก่อน และจะเกิดการทำงานต่างๆดังนี้

1. โปรแกรมจะตรวจสอบดูว่ามีการเก็บสถิติของ ผู้ใช้ที่กำลังใช้ระบบรึเปล่า
2. ถ้าไม่มีสถิติของผู้ใช้ที่กำลังใช้ระบบ โปรแกรมจะทำการเก็บสถิติ เฉพาะข้อมูลของผู้ใช้ที่ใช้ระบบในขณะนั้น โดยจะไปดูข้อมูลใน Data Dict ของ DBMS ว่าผู้ใช้ระบบในขณะนั้น มีตารางอะไรบ้างและนับว่าตารางเหล่านั้นมีจำนวนกี่แถว, แต่ละตารางมีคอลัมน์อะไรบ้าง, แต่ละคอลัมน์มีดัชนีรึเปล่า, แต่ละตารางมีค่าสูงสุดและต่ำสุดเป็นอะไร, มีค่าที่เป็นไปได้ทั้งหมดกี่ค่า และมีชนิดของข้อมูลเป็นอะไร
3. เมื่อผู้ใช้ป้อนคำสั่ง SQL เข้าไปโปรแกรมจะนำคำสั่งนั้นมาแก้ไข โดยพิจารณาจากข้อมูลทางสถิติที่ได้เก็บมาในข้อสอง
4. เมื่อแก้ไขคำสั่งเสร็จแล้วก็จะส่งคำสั่งนั้นต่อไปให้ DBMS เพื่อประมวลต่อ
5. หากผู้ใช้กดปุ่ม analyze ระบบจะทำการลบข้อมูลทางสถิติของทั้งหมดของผู้ใช้คนนั้นแล้วทำการเก็บใหม่ทั้งหมด

7.3 โครงสร้างของระบบ



รูปที่ 7.1โครงสร้างของระบบ

ส่วนประกอบหลักๆ ของระบบมี 4 ส่วนดังนี้

1. User Interface เป็นส่วนที่ใช้รับคำสั่ง SQL จากผู้ใช้และแสดงผลของคำสั่ง นอกจากนั้นยังใช้รับคำสั่งอื่นๆจากผู้ใช้ด้วย
2. Optimizer เป็นส่วนที่นำคำสั่ง SQL ของผู้ใช้มาแก้ไข โดยพิจารณาจากสถิติที่ได้เก็บไว้ใน JdataStore แล้วส่งไปประมวลผลที่ DBMS ต่อ
3. Datadict เป็นส่วนที่เก็บข้อมูลทางสถิติ เพื่อไว้ให้แก่ Optimizer สำหรับใช้พิจารณาประกอบในการแก้ไขคำสั่ง SQL และหากผู้ใช้กดปุ่ม analyze ใน User Interface ในส่วนนี้จะทำการเก็บสถิติใหม่ทันที
4. DB utility เป็นส่วนที่ระบบใช้ติดต่อกับ DBMS ทั้งในการส่งคำสั่ง SQL ไปประมวลผลและรับผลลัพธ์จากการประมวลผล

7.4 วิธีการในการเก็บสถิติ

วิธีการจัดเก็บข้อมูลทางสถิติของระบบจะเก็บในรูปแบบของ Relational database โดยใช้ Jdata Store เป็นตัวเก็บ ซึ่ง Jdata Store จะเป็น Embeded Relational database ที่ทำหน้าที่เหมือน DBMS ขนาดเล็ก การออกแบบฐานข้อมูลสำหรับจัดเก็บข้อมูลทางสถิติ จะออกแบบโดยแบ่งเป็น 2 ตารางดังนี้

1. ตาราง TABLEROW มีคอลัมน์ต่างๆดังนี้
   1.1. Username ใช้เก็บชื่อของผู้ที่ใช้ระบบ ค่านี้จะได้มาจากการป้อนของผู้ใช้ในการเข้ามาใช้งานในแต่ละครั้ง
   1.2. Password ใช้เก็บรหัสของผู้ที่ใช้ระบบ ค่านี้จะได้มาจากการป้อนของผู้ใช้ในการเข้ามาใช้งานในแต่ละครั้ง
   1.3. Tablename ใช้เก็บชื่อตาราง ค่านี้จะได้มาจากการดูใน Data Dictionary ของ ORACLE ว่าผู้ใช้ระบบในขณะนั้นมีตารางอะไรบ้าง
   1.4. Row ใช้ในการเก็บจำนวนแถวทั้งหมดของตารางนั้น ซึ่งค่านี้ได้มาจากการส่งคำสั่ง " select count(*) from ชื่อตาราง" ไปประมวลผลแล้วนำผลลัพธ์ที่ได้กลับมาบันทึกค่า
2. ตาราง COLUMNSTATE มีคอลัมน์ต่างๆดังนี้
   2.1. Username ใช้เก็บชื่อของผู้ที่ใช้ระบบ ค่านี้จะได้มาจากการป้อนของผู้ใช้ในการเข้ามาใช้งานในแต่ละครั้ง
   2.2. Password ใช้เก็บรหัสของผู้ที่ใช้ระบบ ค่านี้จะได้มาจากการป้อนของผู้ใช้ในการเข้ามาใช้งานในแต่ละครั้ง
   2.3. Tablename ใช้เก็บชื่อตาราง ค่านี้จะได้มาจากการดูใน Data Dictionary ของ ORACLE ว่าผู้ใช้ระบบในขณะนั้นมีตารางอะไรบ้าง
   2.4. COLLUMNNAME ใช้เก็บชื่อ columnname ของตาราง ค่านี้จะได้มาจาก method getColumnName(เลขจำนวนเต็มบวกระบุตำแหน่งคอลัมน์) ซึ่งเป็น method ของคลาส ResultSetMetaData
   2.5. DISTINCTVALUE ใช้เก็บจำนวนค่าที่เป็นไปได้ทั้งหมดในคอลัมน์นั้นๆ ค่านี้ได้มาจากการส่งคำสั่ง "select count(distinct ชื่อcolumn from ชื่อตาราง" ไปประมวลผลแล้วนำผลลัพธ์ที่ได้กลับมาบันทึกค่า
   2.6. NULLVALUE ใช้เก็บจำนวนค่าว่าง ในคอลัมน์นั้นๆ ค่านี้ได้มาจากการส่งคำสั่ง "select count(*) from ชื่อตาราง where ชื่อcolumn is null " ไปประมวลผลแล้วนำผลลัพธ์ที่ได้กลับมาบันทึกค่า
   2.7. MINVALUE ใช้เก็บค่าที่น้อยสุดในคอลัมน์นั้นๆ หากชนิดของข้อมูลในคอลัมน์นั้นเป็น varchar ก็จะพิจารณาโดยดูจากลำดับตัวอักษร ซึ่งค่าในคอลัมน์นี้ได้มาจากการส่งคำสั่ง "select min(ชื่อcolumn) from ชื่อตาราง" ไปประมวลผลแล้วนำผลลัพธ์ที่ได้กลับมาบันทึกค่า

2.8. MAXVALUE ใช้เก็บค่าที่มากสุดในคอลัมน์นั้นๆ หากชนิดของข้อมูลในคอลัมน์นั้นเป็น varchar ก็จะพิจารณาโดยดูจากลำดับตัวอักษร ค่านี้ได้มาจากการส่งคำสั่ง "select max(ชื่อcolumn) from ชื่อตาราง" ไปประมวลผลแล้วนำผลลัพธ์ที่ได้กลับมาบันทึกค่า

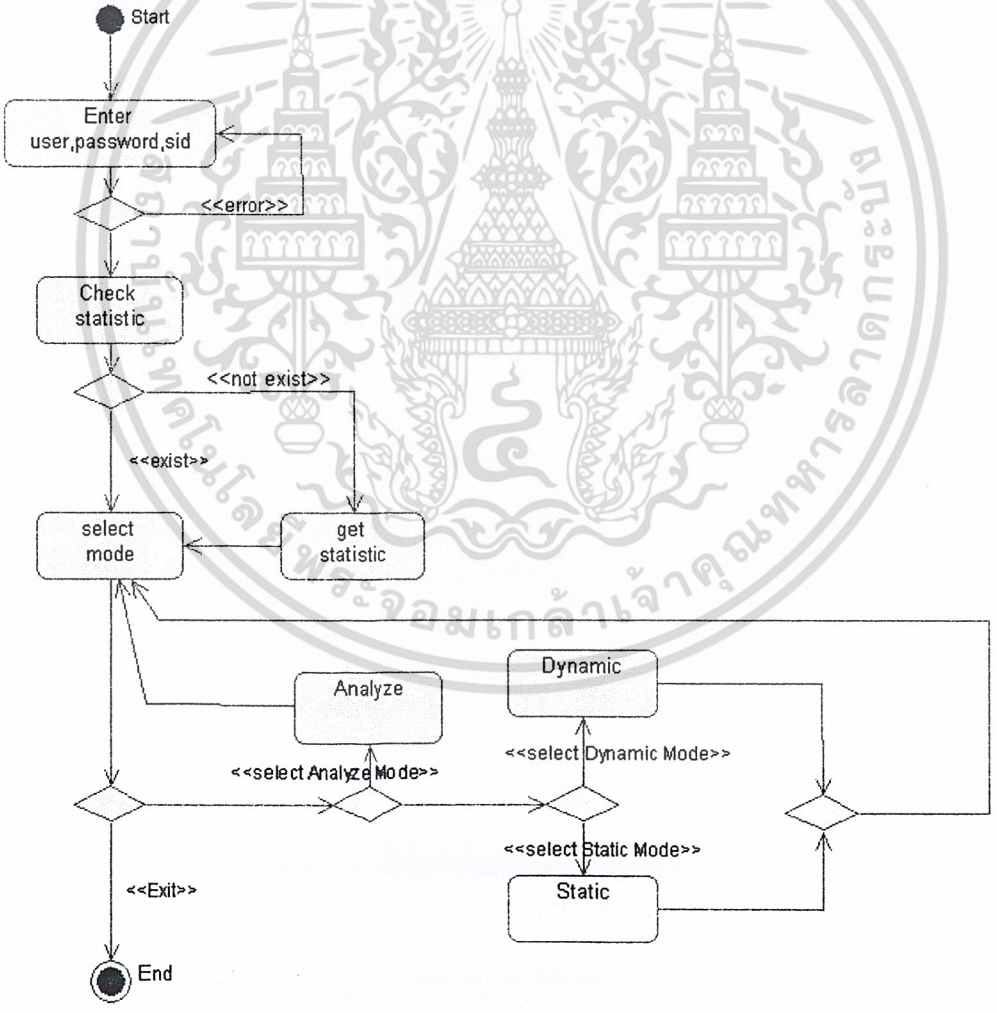
2.9. DATATYPE ใช้เก็บชนิดของข้อมูลในคอลัมน์นั้นๆ เช่น varchar,number ค่านี้ได้มาจาก method getColumnTypeName(เลขจำนวนเต็มบวกระบุตำแหน่งคอลัมน์) ซึ่งเป็น method ของคลาส ResultSetMetaData

2.10 ISINDEX ใช้เพื่อตรวจสอบดูว่าคอลัมน์นั้นๆ มีการสร้างดัชนีหรือไม่ ค่านี้ได้มาจากการดูค่าใน Data Dictionary ของ ORACLE

บทที่ 8

การวิเคราะห์และการออกแบบระบบ

การพัฒนาโปรแกรมที่ดีนั้นจะต้องเริ่มต้นพัฒนาระบบอย่างเป็นขั้นเป็นตอน และมีหลักการ คือ ขั้นตอนการวิเคราะห์และออกแบบจะต้องชัดเจน ทั้งนี้การศึกษาและการใช้กระบวนการพัฒนาเชิงวัตถุต่างๆ จำเป็นต้องมีความเข้าใจแนวคิดเชิงวัตถุอย่างแม่นยำ ในระบบซอฟต์แวร์ก็เช่นกัน ผู้ที่พัฒนาระบบซอฟต์แวร์ที่ต้องการใช้ภาษาในการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programing : OOP) ก็ควรจะต้องมีการวิเคราะห์และการออกแบบระบบซอฟต์แวร์มากมายหลายวิธี โดยเฉพาะกระบวนการที่มีความเกี่ยวข้องกับภาษา UML (Unified Modeling Language)



รูปที่ 8.1 แสดง Business Process Modeling ของระบบ

8.1 การวิเคราะห์ออกแบบระบบ

เนื่องจากตัวประมวลผลภายนอกแบบอิงสถิติ เป็นซอฟต์แวร์ที่ทำหน้าที่เสมือนเป็นตัวแทนผู้ดูแลระบบ มีหน้าที่ในการ ปรับแต่ง ระบบให้มีประสิทธิภาพที่ดีขึ้น โดยที่ผู้ใช้ไม่จำเป็นต้องมีความรู้ความเชี่ยวชาญด้านฐานข้อมูลมาก โปรแกรมจึงถูกออกแบบเพื่อซ่อนความซับซ้อนวุ่นวายต่างๆ เหล่านี้ โดยในสายตาของ ผู้ใช้จะไม่มีความจำเป็นต้องยุ่งเกี่ยวกับเรื่องต่างๆ เหล่านี้ หน้าที่การทำงานของโปรแกรมจะถูกแบ่งเป็นสองวิธีการ หลักๆ คือ วิธีการแบบสถิต (Static mode) และ วิธีการแบบพลวัต (Dynamic mode)

8.1.1 วิธีการแบบสถิต

เป็นการนำคำสั่ง SQL ที่มีอยู่แล้วใน ไฟล์ซอร์ดโค้ด (source code) มาทำการแก้ไขเปลี่ยนแปลง โดยจะวิเคราะห์จากข้อมูลทางสถิติที่เกี่ยวข้องกับโปรแกรมนั้นๆ ข้อมูลทางสถิติเหล่านี้จะได้มาในขณะนั้นทันทีว่าควรจะแก้ไขให้คำสั่ง SQL ออกมาในแบบใดเพื่อให้ใช้เวลาในการประมวลผลน้อยที่สุด

การทำงานในวิธีการแบบสถิตนี้ผู้ใช้จำเป็นจะต้องมีไฟล์ซอร์ดโค้ดที่จะมาทำการพัฒนา และเมื่อไฟล์ซอร์ดโค้ดผ่านการพัฒนาแล้ว ผู้ใช้จะต้องนำไฟล์ซอร์ดนั้นไป คอมไพล์ (compile) อีกที่จึงจะสามารถใช้ได้ แต่เนื่องจากข้อมูลในฐานข้อมูลมีการเปลี่ยนแปลงอยู่ตลอดเวลาจึงส่งผลให้ข้อมูลทางสถิติของฐานข้อมูลนั้นๆ เปลี่ยนแปลงตามไปด้วย

ด้วยเหตุนี้ในการทำงานแบบสถิตจึงมีความจำเป็นที่จะต้องพัฒนาสถิติอยู่เรื่อยๆตามความเหมาะสม

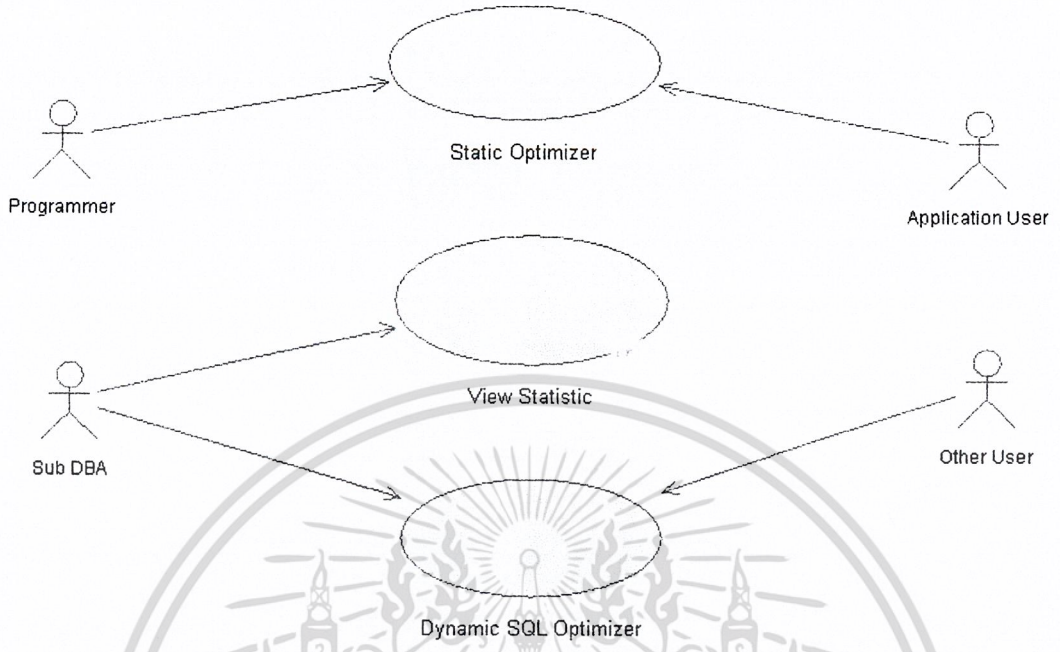
8.1.2 วิธีการแบบพลวัต

เป็นการนำคำสั่ง SQL ที่รับมาในขณะนั้น มาทำการแก้ไขปรับปรุงทันที โดยวิเคราะห์จากข้อมูลทางสถิติของตารางที่คำสั่งนั้นเกี่ยวข้องด้วย ซึ่งข้อมูลทางสถิติเหล่านี้ไม่จะเป็นจะต้องมีการปรับปรุงตลอดเวลาเนื่องจากหากมีการปรับปรุงทุกครั้งที่ประมวลผลคำสั่งจะทำให้การทำงานเป็นไปได้ช้ากว่าเดิม ดังนั้นจะใช้ข้อมูลทางสถิติที่มีอยู่แล้ว และจะมีการปรับปรุงบ้างเป็นบางครั้ง

ในการทำงานแบบพลวัตนี้ ในขณะที่ผู้ใช้ป้อนคำสั่งในขณะนั้น ก็จะเกิดการแก้ไขคำสั่งทันทีที่ผู้ใช้ทำการป้อนคำสั่งและหากคำสั่งที่ผู้ใช้ป้อนคำสั่งผิดรูปแบบของคำสั่ง SQL ก็จะไม่เกิดเปลี่ยนแปลงคำสั่งที่ป้อนเข้ามาแต่อย่างใด

8.2 ยูสเคสไดอะแกรม (Use case Diagram)

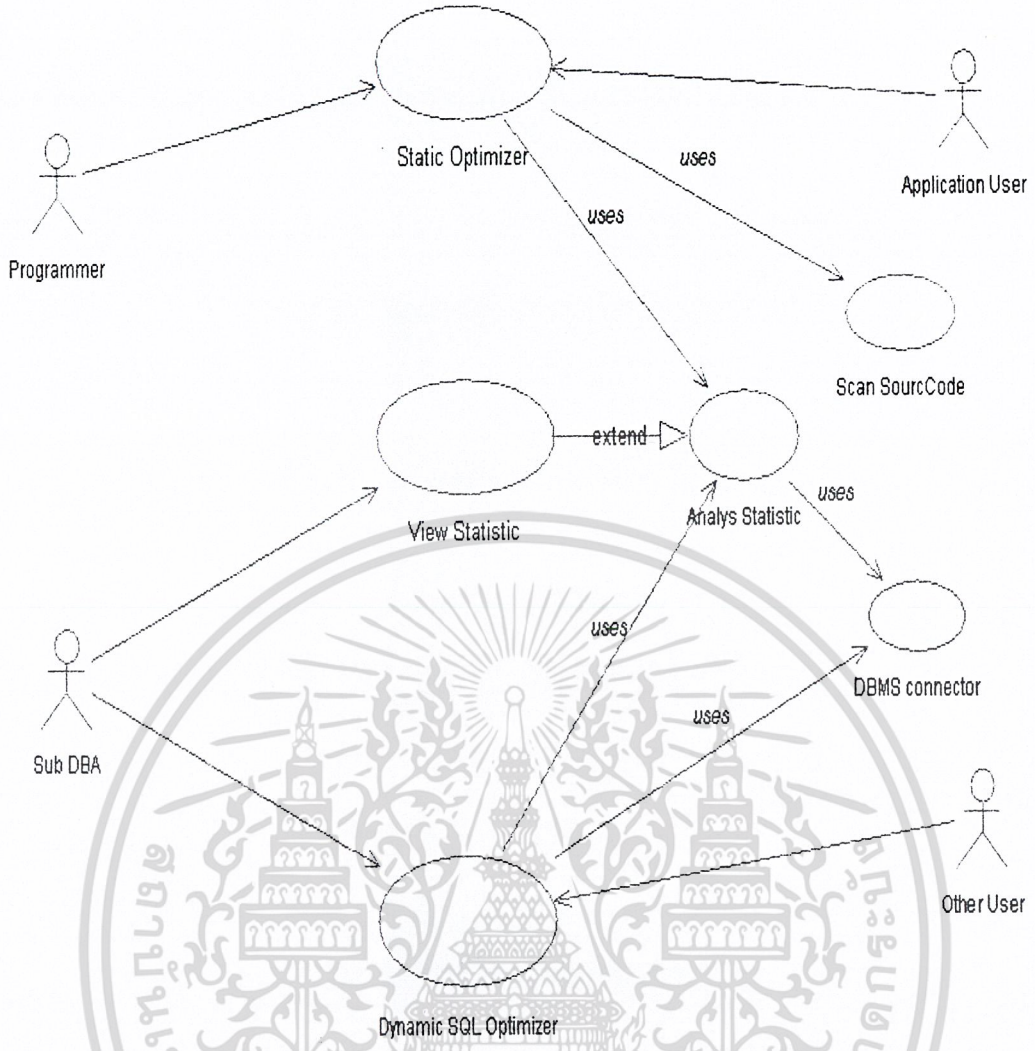
รูปที่ 8.2 แสดงยูสเคสไดอะแกรมของระบบ



รูปที่ 8.2 Use case diagram แสดงการทำงานของระบบ

จากรูปจะเห็นว่ามีผู้ใช้ระบบทั้งหมด 4 ประเภทซึ่งจะมาใช้โปรแกรมด้วยวัตถุประสงค์ที่ต่างกัน ดังนี้

- ผู้พัฒนาโปรแกรม (Programmer): ผู้พัฒนาโปรแกรมจะนำโปรแกรมที่เขียนเสร็จแล้วมาทำการพัฒนาไฟล์ซอร์ดโค้ดของตัวเองเพื่อให้ คำสั่งSQLที่ฝังตัวอยู่ในไฟล์ซอร์ดโค้ด (Static SQL) ของตัวเองนั้นได้รับการปรับปรุงแก้ไขก่อนจะถูกคอมไพล์ เพื่อให้โปรแกรมทำงานได้เร็วที่สุดหลังจากถูกคอมไพล์ไปแล้ว
- ผู้ใช้โปรแกรม (Application User): เป็นผู้ที่ใช้งานโปรแกรมทั่วไป แต่ไฟล์ซอร์ดโค้ดของโปรแกรมที่กำลังใช้งานอยู่และต้องการไฟล์ซอร์ดโค้ดของตัวเองมาแก้ไขใหม่ เพื่อให้คำสั่ง SQL ในโปรแกรมนั้นๆทำงานได้เหมาะสมกับข้อมูลทางสถิติของฐานข้อมูลในขณะนั้น โดยอาจจะนำไฟล์ซอร์ดโค้ดมา Optimize เป็นประจำตามความเหมาะสม
- ผู้ดูแลระบบย่อย (Sub DBA): เป็นผู้ที่ทำหน้าที่ใช้งานโปรแกรมเพื่อปรับประสิทธิภาพของโปรแกรมโดยไม่จำเป็นต้องมีความรู้ด้านฐานข้อมูลมากนัก
- ผู้ใช้ทั่วไป (Other User): เป็นผู้ที่มาใช้ระบบเพื่อต้องการจะดูข้อมูลทั่วๆไปในฐานข้อมูล



รูปที่ 8.3 Use case Diagram แสดงการทำงานภายใน

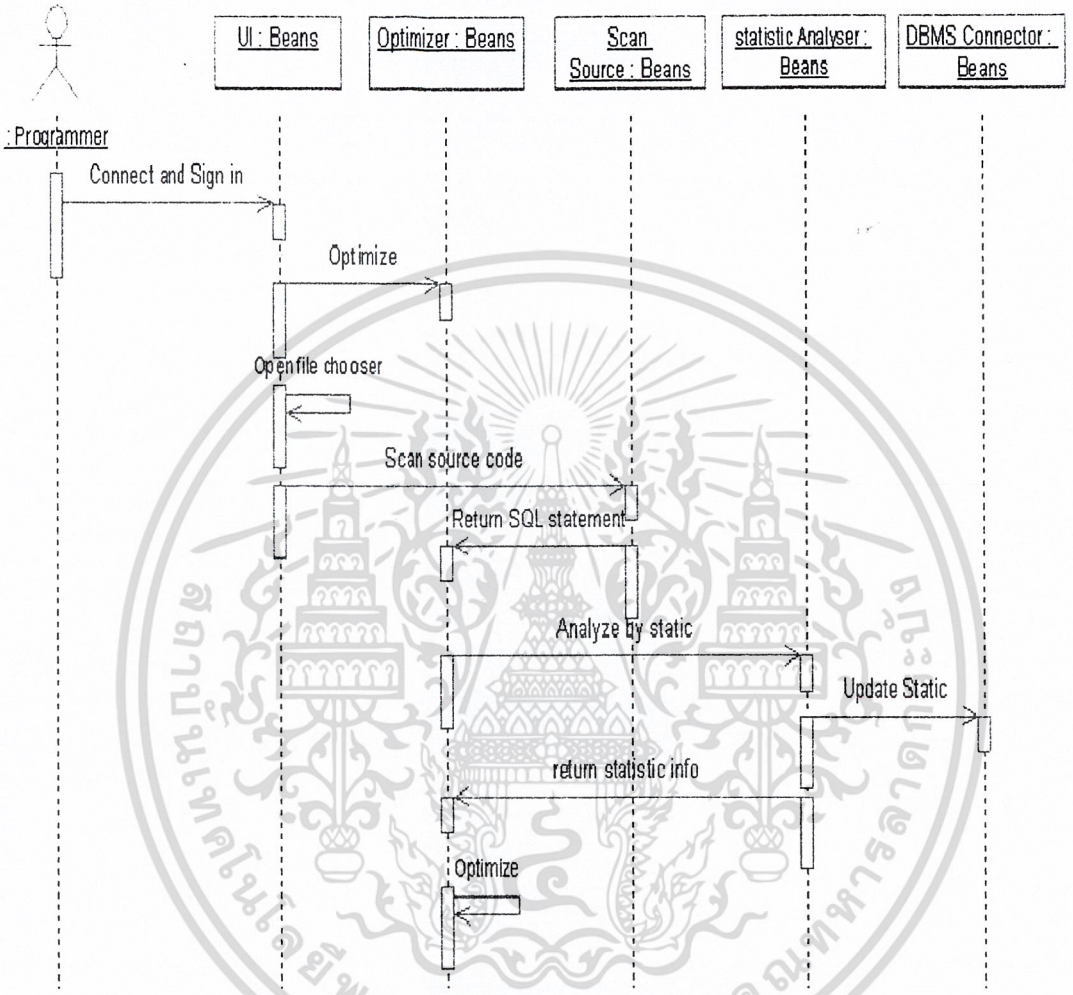
รูปที่ 8.3 เป็นยูสเคสไดอะแกรมแสดงการทำงาน ที่ลึกกว่ารูปที่ 8.2 จากรูปจะเห็นว่าการทำงานของทั้งวิธีการแบบสถิตและวิธีการแบบพลวัตมีการใช้ข้อมูลทางสถิติร่วมกัน และของมูลทางสถิติเหล่านี้จะได้มาจากการเก็บสถิติจริงผ่านทาง DBMS connector

ตัว DBMS connector นอกจากจะใช้ในการนำข้อมูลทางสถิติมาให้กับ Analysis statistic แล้ว ยังมีหน้าที่อีกอย่างคือ ส่งคำสั่ง SQL จากการทำงานแบบพลวัตที่ผ่านการ Optimize แล้วไปประมวลผลที่ระบบจัดการฐานข้อมูล

คำสั่ง SQL ที่ทำงานในแบบสถิตและพลวัตนั้นต่างกัน โดยแบบสถิตนั้นจะได้มาจากการค้นหาคำสั่งจากในไฟล์ซอร์ดโค้ด ส่วนแบบพลวัตจะได้มาจากการป้อนของผู้ใช้โดยตรง

8.3 Sequence Diagram

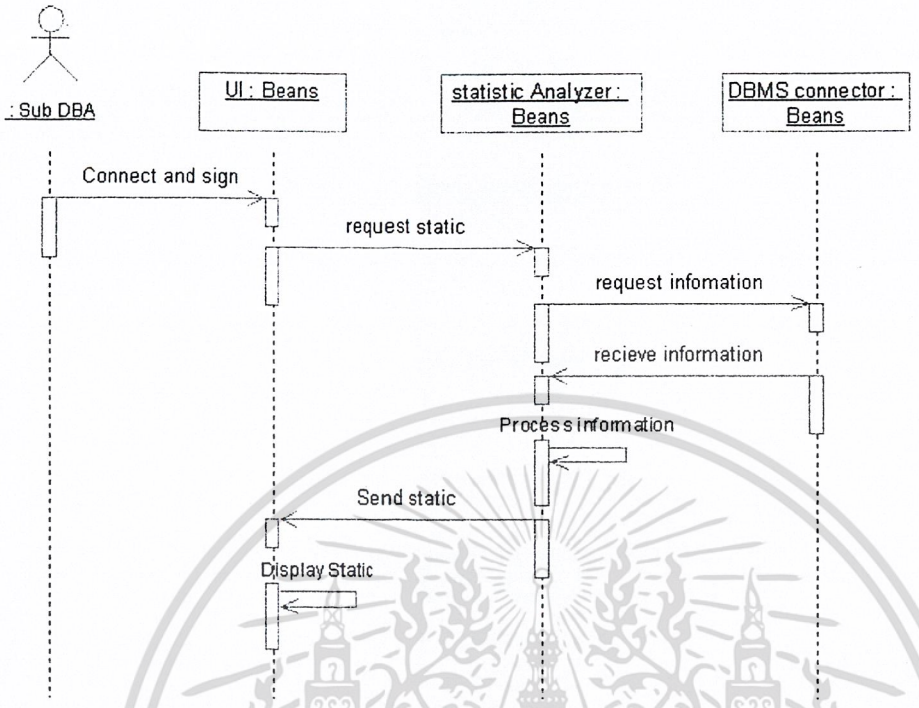
8.3.1 Sequence Diagram ของการทำงานแบบสถิต



รูปที่ 8.4 Sequence Diagram แสดงลำดับการทำงานของ การทำงานในแบบสถิต

หลังจากที่ผู้ใช้ได้เข้ามาในระบบแล้ว จะต้องทำการเลือกไฟล์ที่เป็นไฟล์ซอร์ดโค้ด (Source code) ที่ต้องการจะพัฒนา หลังจากนั้นจะทำการค้นหาในไฟล์เพื่อหาคำสั่ง SQL เมื่อได้คำสั่ง SQL มาก็จะส่งต่อให้กับตัว optimizer เพื่อทำการแก้ไข โดยพิจารณาจากข้อมูลทางสถิติ แล้วบันทึกคำสั่ง SQL ที่ผ่านการพัฒนาแล้วแทนคำสั่งเดิม

8.3.2 Sequence Diagram ของการแสดงข้อมูลทางสถิติ

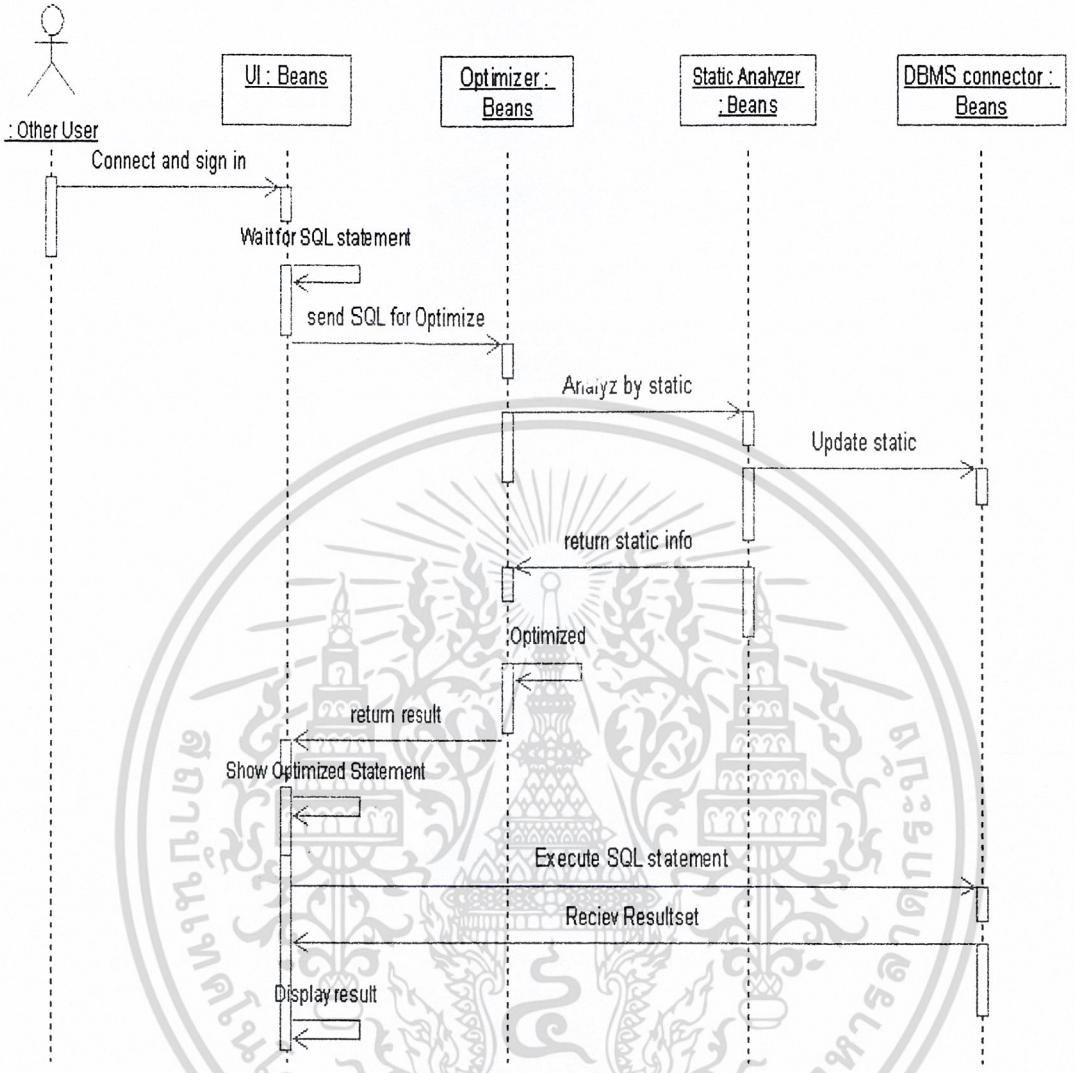


รูปที่ 8.5 Sequent Diagram แสดงลำดับการทำงานเมื่อมีการแสดงข้อมูลทางสถิติที่ถูกจัดเก็บไว้

จากรูปที่ 8.5 เมื่อผู้ใช้เข้ามาในระบบแล้วต้องการดูข้อมูลที่จัดเก็บอยู่ ตัว statistic analyzer จะไปนำข้อมูลจริงมาโดยผ่านทาง DBMS connector เมื่อได้ข้อมูลมาก็จะประมวลผลข้อมูลเพื่อจัดเก็บในรูปแบบที่กำหนดแล้วจึงแสดงออกจอภาพ

จากการทำงานจะเห็นว่าตัว การดูข้อมูลทางสถิตินั้นนอกจากจะเป็นการ แสดงข้อมูลที่จัดเก็บแล้วยังเป็นการปรับปรุงข้อมูลไปในตัวด้วย

8.3.3 Sequence Diagram ของการทำงานแบบพลวัต



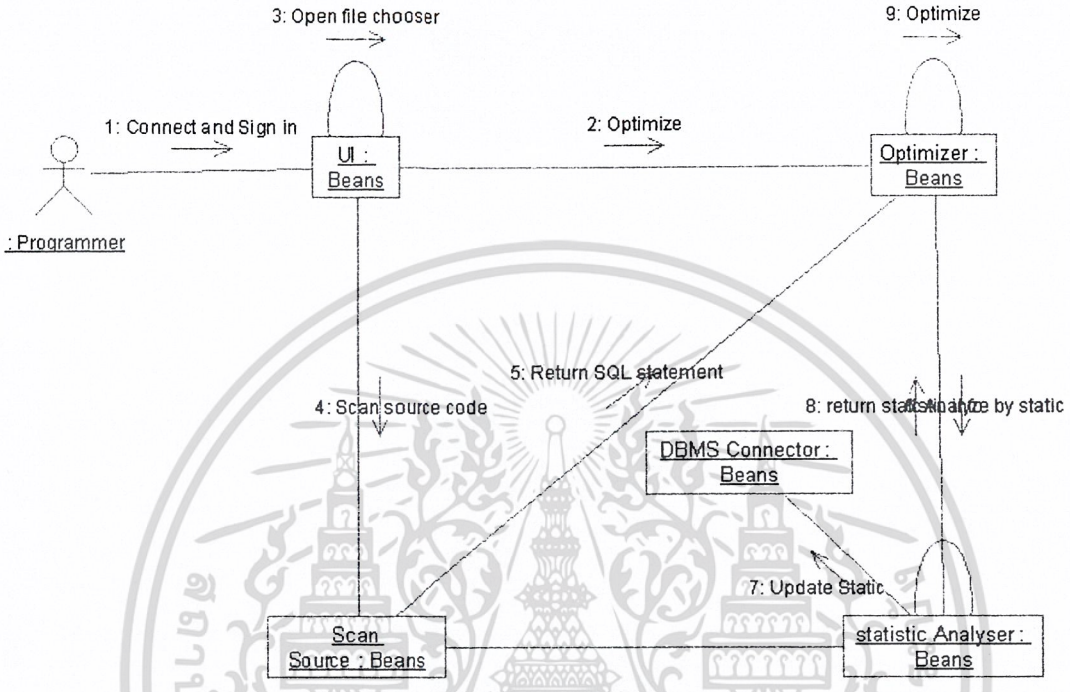
รูปที่ 8.6 Sequent Diagram แสดงการทำงานของแบบพลวัต

หลังจากที่ ผู้ใช้เข้ามาในระบบแล้วระบบจะรอให้ผู้ใช้ ป้อนคำสั่ง เมื่อผู้ใช้ป้อนเสร็จระบบก็จะส่งคำสั่งนี้ให้กับตัว Optimizer ทันที เมื่อ Optimizer ได้รับคำสั่งก็จะทำการ Optimize จากข้อมูลที่มีอยู่ และจะทำการเปลี่ยนแปลงข้อมูลบ้างเป็นบางครั้ง

หลังจำที่ตัว Optimizer เปลี่ยนแปลง คำสั่งแล้ว ก็จะส่งคำสั่งนั้นไป ประมวลผลผ่านทาง DBMS connector และจะส่งผลลัพธ์กลับมาในรูปของเซทผลลัพธ์เพื่อแสดงออกที่จอภาพ

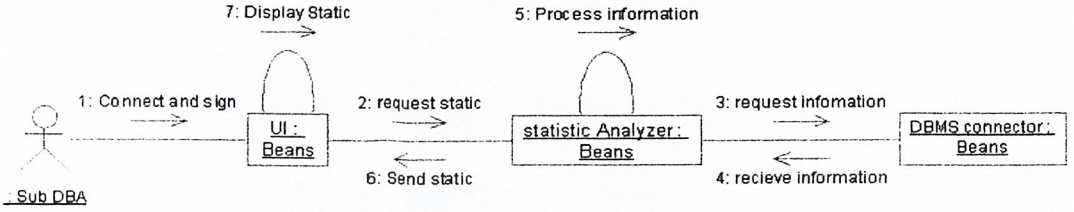
8.4 Collaboration Diagram

8.4.1 Collaboration Diagram ของการทำงานแบบพลวัต



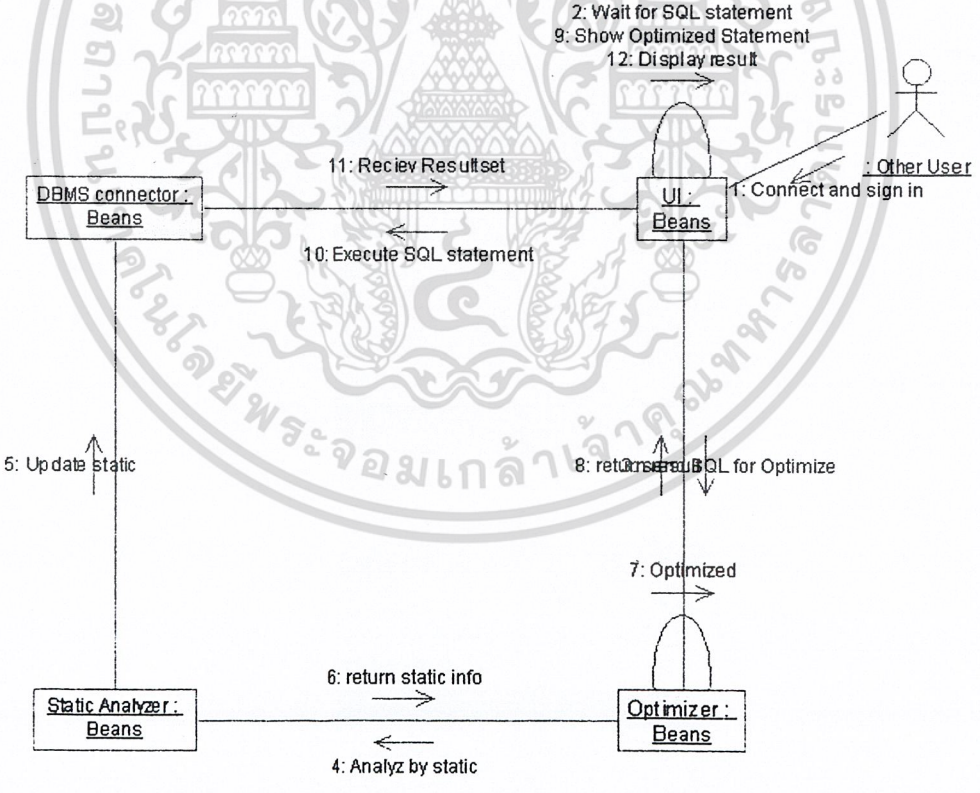
รูปที่ 8.7 Collaboration Diagram ของการทำงานแบบสถิต

8.4.2 Collaboration Diagram ของการดูสถิติ



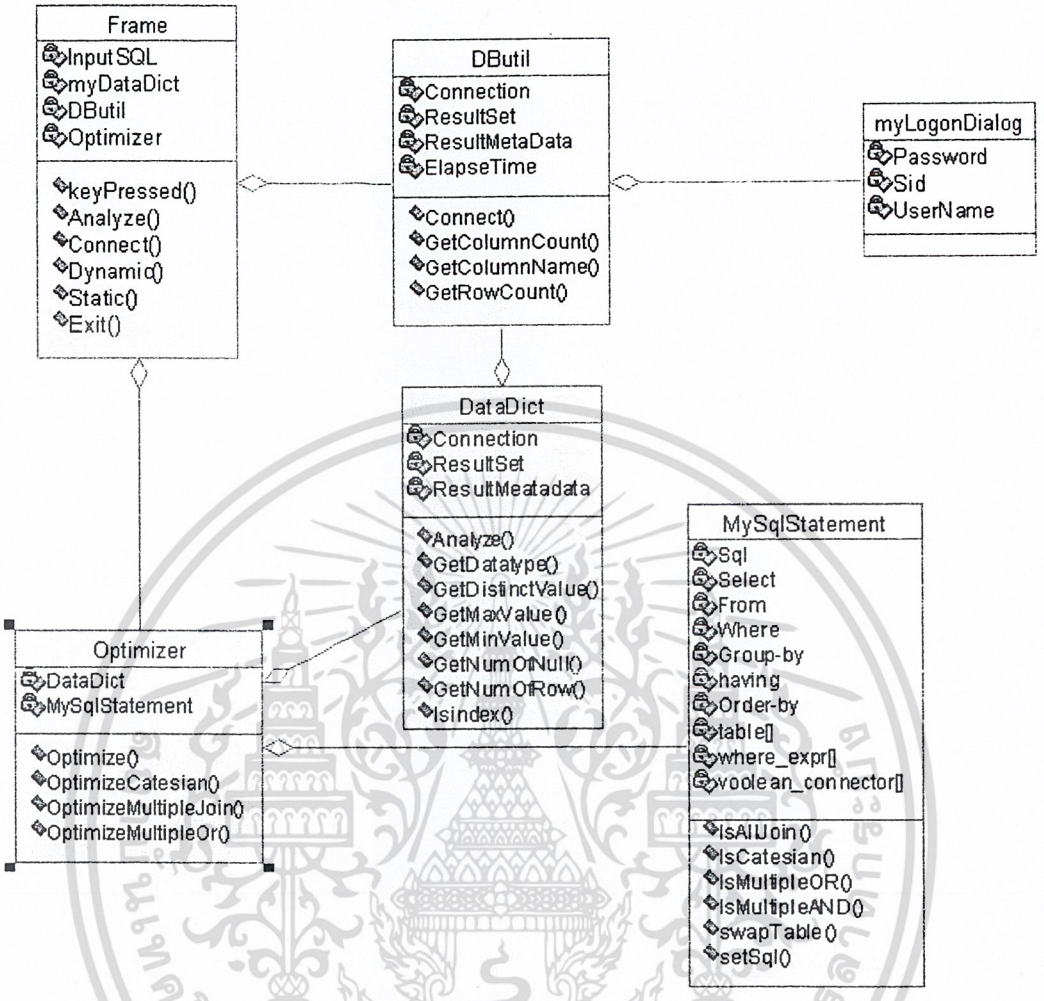
รูปที่ 8.8 Collaboration Diagram ของการดูสถิติ

8.4.3 Collaboration Diagram ของการทำงานแบบพลวัต



รูปที่ 8.9 Collaboration Diagram ของการทำงานแบบ พลวัต

8.5 Class Diagram ของระบบ



รูปที่ 8.10 ภาพ Class Diagram ของทั้งระบบ

จากรูปที่ 8.10 จะเห็นว่าประกอบไปด้วยคลาส หลักๆ ทั้งหมด 6 คลาสดังนี้

1. คลาส frame ซึ่งถือว่าเป็น คลาสหลักของโปรแกรม ทำหน้าที่เป็นทั้งส่วนติดต่อกับผู้ใช้และส่วนควบคุมโปรแกรม
2. คลาส Optimizer ทำหน้าที่รับคำสั่ง SQL มาจากคลาส frame หลังจากนั้นจะเรียก method Optimize เพื่อส่งคำสั่งที่รับมาให้กับ MySqlStatement ต่อ เพื่อพิจารณาดูว่าคำสั่งที่รับมานั้นเข้ากรณีไหน เพื่อที่จะได้รู้ว่าจะเปลี่ยนแปลงยังไงต่อไป หลังจากนั้นจะดูข้อมูลทางสถิติจากคลาส DataDict เพื่อวิเคราะห์ว่าจะเปลี่ยนแปลงรูปแบบคำสั่งยังไงเพื่อให้ประมวลผลเร็วสุดแล้วจึงทำการเปลี่ยนแปลงรูปแบบคำสั่ง

3. คลาส DataDict เป็นคลาส ที่ทำหน้าที่ เก็บข้อมูลทางสถิติของผู้ใช้ทุกคนที่เข้ามาใช้ระบบ ซึ่งผู้ใช้แต่ละคนที่กำลังใช้ระบบในขณะนั้นจะสามารถเห็นได้เฉพาะข้อมูลทางสถิติของตัวเองเท่านั้น ขอมูลทางสถิติเหล่านี้จะถูกเก็บลงบน JdataStore ในการเก็บข้อมูลแต่ละครั้งนั้น คลาส DataDict จะไปสำรวจข้อมูลผ่านคลาส DButil
4. คลาส DButil เป็นคลาสที่ทำหน้าที่เชื่อมต่อและปิดการเชื่อมต่อกับ DBMS คลาสนี้จะรับคำสั่ง SQL มาจากส่วนต่างของระบบและส่งคำสั่งเหล่านี้ไปให้ DBMS ประมวลผล และรับผลลัพธ์กลับมาและส่งกลับไปให้ส่วนที่ส่งคำสั่ง SQL มา นอกจากนั้นในระหว่างที่คำสั่ง SQL ถูกส่งไปประมวลผลที่ DBMS คลาสนี้จะจับเวลาที่ใช้ในการประมวลไปด้วย เพื่อในไปแสดงให้ผู้ใช้เห็น
5. คลาส MySqlStatement เป็นคลาสที่ทำหน้าที่ตรวจสอบ คำสั่ง SQL ที่รับมาจากคลาส Optimizer ว่าเป็นเป็นคำสั่งประเภทไหน เช่น คำสั่งนั้นมีการทำ Catesian รึเปล่า เป็นต้น
6. คลาส MyLogonDialog ทำหน้าที่รับ ชื่อ,รหัส,และ sid เพื่อเข้าสู่การใช้ระบบ หากป้อนผิด ก็จะให้ป้อนใหม่ไปเรื่อยๆ

บทที่ 9

การทดลองและวิเคราะห์ผล

9.1 ขั้นตอนการทดลอง

ทำการทดลองโดยการทดสอบการพิมพ์คำสั่ง query ผ่านตัวต้นแบบของโปรแกรมแล้วทำการเปรียบเทียบความเร็วระหว่างที่ทำการ optimize และไม่ได้ทำการ optimize โดยทำการวัดเวลาในการส่งคำสั่ง SQL ไปประมวลผลจนได้ผลลัพธ์ออกมา

- เขียนโปรแกรมทำการเพิ่มข้อมูล (insert) ไว้ใช้ทดสอบ ประสิทธิภาพของการปรับปรุงคำสั่ง SQL ด้วยโปรแกรมต้นแบบ
- ทำการหา SQL สเตตเมนท์ขึ้นมาทดสอบ
- ทำการประมวลผลคำสั่งระหว่างคำสั่งที่ ยังไม่แก้ไขกับแก้ไขแล้ว
- จับเวลาที่ใช้ในการประมวลผลระหว่างคำสั่งที่ ยังไม่แก้ไขกับแก้ไขแล้ว

9.2 ส่วนประกอบในการทดลอง

การทดลองนี้ทำการทดลองกับ Oracle8i โดยโปรแกรมตัวต้นแบบพัฒนาโดยใช้ Jbuilder X

9.3 การทดลอง

ทำการทดลองโดยการทำการเก็บสถิติของผู้ใช้ก่อน เพื่อให้โปรแกรมสามารถใช้สถิติที่เก็บเองมาในการ optimize คำสั่งได้

9.3.1 ทดลองการ optimize ด้วยการทำ multiple join

ทำการทดลองกับ query ดังต่อไปนี้

```
SELECT count(s.supplierid)
FROM suppliers s,products p,categories c,order_details od
WHERE s.supplierid=p.supplierid
and c.categoryid=p.productid
and p.unitprice<od.unitprice
```

ซึ่งจากสถิติที่เก็บเองแล้ว มีข้อมูลตามตารางที่ 9.1 ดังนี้

Table	Rows
categories	50
suppliers	200
products	1000
order_details	4995

ตารางที่ 9.1 แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ optimize การทำ multiple join

ซึ่งโปรแกรมจะทำการแปลง query จากสถิติในตารางที่ 9.1 เป็นดังต่อไปนี้

```
SELECT /*+ ordered */
 count(s.supplierid)
FROM categories c ,suppliers s ,products p ,order_details od
WHERE s.supplierid=p.supplierid
AND c.categoryid=p.productid
AND p.unitprice<od.unitprice
```

9.3.1.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Rule-Based Optimizer (ไม่ได้เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.2

Original query	5234	5217	5219	5250	5141	5203	5172	5219	5250	5140
Optimized query	62	78	94	78	78	78	109	110	78	78

ตารางที่ 9.2 แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)

9.3.1.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Cost-Based Optimizer (เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.3

Original query	79	79	78	63	63	78	78	63	62	63
Optimized query	78	63	62	62	63	78	63	63	78	79

ตารางที่ 9.3 แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)

จะเห็นว่าเมื่อทำการสั่งให้ Oracle ทำการเก็บสถิติแล้วเวลาที่ได้ไม่ต่างกันมากนัก เนื่องจาก Oracle มีสถิติ ทำให้สามารถใช้ Cost-Based Optimizer ได้ และ query ทั้งสองก็มีแผนการประมวลผลที่เหมือนกันด้วย

9.3.2 ทดลองการ optimize ด้วยการทำ Catesian product

ทำการทดลองกับ query ดังต่อไปนี้

```
SELECT count(*)
FROM shippers,order_details
```

ซึ่งจากสถิติที่เก็บเองแล้ว มีข้อมูลตามตารางที่ 9.4 ดังนี้

Table	Rows
shippers	80
order_details	4998

ตารางที่ 9.4 แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ Optimize การทำ Catesian product

ซึ่งโปรแกรมจะทำการแปลง query จากสถิติที่ได้เป็นดังต่อไปนี้

```
SELECT /*+ rule */
 count(*)
FROM order_details,shippers
```

9.3.2.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Rule-Based Optimizer (ไม่ได้เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.5

Original query	63	78	63	63	63	62	78	78	62	62
Optimized query	31	47	43	31	31	31	47	31	31	47

ตารางที่ 9.5 แสดงจำนวน เวลาของการประมวลผลการ optimize catesian product วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)

9.3.2.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Cost-Based Optimizer (เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.6

Original query	94	94	94	78	94	109	94	78	94	79
Optimized query	78	63	47	31	31	31	31	31	31	32

ตารางที่ 9.6 แสดงจำนวน เวลาของการประมวลผลการ optimize catesia... product วัดเทียบกันสิบครั้ง โดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)

9.3.3 ทดลองการ optimize ด้วยการทำ Multiple Join ที่มีการทำ Catesian product

ทำการทดลองกับ query ดังต่อไปนี้

```
SELECT count(s.supplierid)
FROM suppliers s,products p,categories c,order_details od
WHERE s.supplierid=p.supplierid
and c.categoryid=p.productid
```

ซึ่งจากสถิติที่เก็บเองแล้ว มีข้อมูลตามตารางที่ 9.7 ดังนี้

Table	Rows
categories	50
suppliers	200
products	1000
order_details	4998

ตารางที่ 9.7 แสดงสถิติที่เก็บเองและนำมาใช้ได้ของตารางที่ใช้ในการทดลองการ Optimize การทำ multiple join ที่มีการทำ Catesian product

ซึ่งจากสถิติที่ได้ โปรแกรมจะทำการแปลง query เป็นดังต่อไปนี้

```
SELECT /*+ rule */
 count(s.supplierid)
FROM order_details od,products p ,suppliers s ,categories c
WHERE s.supplierid=p.supplierid
AND c.categoryid=p.productid
```

9.3.3.1 ทำการทดลองโดยไม่ได้เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Rule-Based Optimizer (ไม่ได้เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.8

Original query	1328	1422	1375	1375	1406	1360	1359	1375	1390	1563
Optimized query	31	31	32	47	47	31	47	47	47	47

ตารางที่ 9.8 แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join ที่มีการทำ Catesian product วัดเทียบกันสิบครั้งโดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Rule-Based Optimizer)

9.3.3.2 ทำการทดลองโดยที่เก็บสถิติให้ Oracle

ทำการทดลองโดยการ query โดยที่ Oracle ใช้ Cost-Based Optimizer (เก็บสถิติให้ Oracle) จากการวัดผลลัพธ์การ query ทั้งสองแบบ จะได้ผลลัพธ์ตามตารางที่ 9.9

Original query	78	78	47	62	47	46	62	78	47	46
Optimized query	31	31	32	31	31	46	32	31	32	31

ตารางที่ 9.9 แสดงจำนวน เวลาของการประมวลผลการ optimize multiple join ที่มีการทำ Catesian product วัดเทียบกันสิบครั้งโดยที่หน่วยเวลาเป็นมิลลิวินาที (Oracle ใช้ Cost-Based Optimizer)

บทที่ 10

บทสรุปและข้อเสนอแนะ

10.1 สรุปผลการดำเนินงาน

จากการศึกษาและทำการทดลอง ทำให้สามารถทราบถึงกฎทฤษฎีของการปรับแต่ง query สำหรับ Oracle และสามารถนำทฤษฎีบางส่วนมาใช้ทดลองและเขียนโปรแกรมได้ ทำให้ได้ตัวโปรแกรมต้นแบบที่พัฒนาขึ้นมาและนำมาทดลองใช้งานได้ ซึ่งโปรแกรมสามารถทำงานได้ตามตรงตามที่ทำการทดลองมา

10.2 ปัญหา และ อุปสรรค

ปัญหาหนึ่งของการทำการทดลองคือไม่ได้ทำการทดลองบนเครื่อง server จริงๆ ซึ่งการทดลองยังคงเป็นบนเครื่องคอมพิวเตอร์ส่วนตัวอยู่ ซึ่งทำให้ไม่สามารถทดลองบางคำสั่งและไม่มีผลการทดลองที่นำมาใช้เขียนโปรแกรมได้ เช่นการใช้ parallel query เป็นต้น

ในบางครั้งการทดลองก็เป็นไปได้ยากเนื่องจากเห็นผลไม่ชัดเจน ทำให้ไม่สามารถนำผลการทดลองมาใช้งานได้ และการทดลองกับฐานข้อมูลที่มีขนาดใหญ่มากๆก็ยังคงเป็นปัญหาอยู่ เนื่องจากผลการทดลองที่ใช้มาเขียนโปรแกรมเป็นผลการทดลองจากเครื่องคอมพิวเตอร์ส่วนตัวเท่านั้น

สิ่งสำคัญคือบางครั้งข้อมูลที่ได้มาจากแหล่งข้อมูลที่ดูน่าเชื่อถืออาจไม่สามารถใช้งานได้จริงเสมอไป ดังนั้นจึงควรระมัดระวังโดยควรทำการทดลองจนเห็นผลที่แน่นอนก่อนนำมาใช้งานจริง

10.3 แนวทางในการพัฒนาต่อ

โปรแกรมที่ได้พัฒนาขึ้นมานั้น เป็นเพียงตัวต้นแบบของตัวประมวลผลภายนอกแบบอิงสถิติเท่านั้น การทำงานในโปรแกรมนั้นยังมีการวิเคราะห์และสามารถปรับปรุง query ได้เพียงบางส่วนและบางกรณีเท่านั้นเท่านั้น ซึ่งการปรับปรุง query ยังคงยึดติดอยู่กับ Oracle8i ซึ่งหากใช้กับเวอร์ชันอื่นอาจจะปรับแต่งได้ไม่ถูกต้องได้

สำหรับสถิติที่ทำการเก็บเพื่อนำมาใช้งานในโปรแกรมนั้น ได้ทำการเก็บเฉพาะสถิติบางค่าที่คาดว่าจะมีการใช้งานเท่านั้น และสถิติบางค่ายังไม่ถูกนำมาใช้งานอย่างเต็มที่เนื่องจากการทำงานของโปรแกรมยังคงมีมีการคำนวณเพื่อเปรียบเทียบค่าต้นทุนที่ซับซ้อนมาก

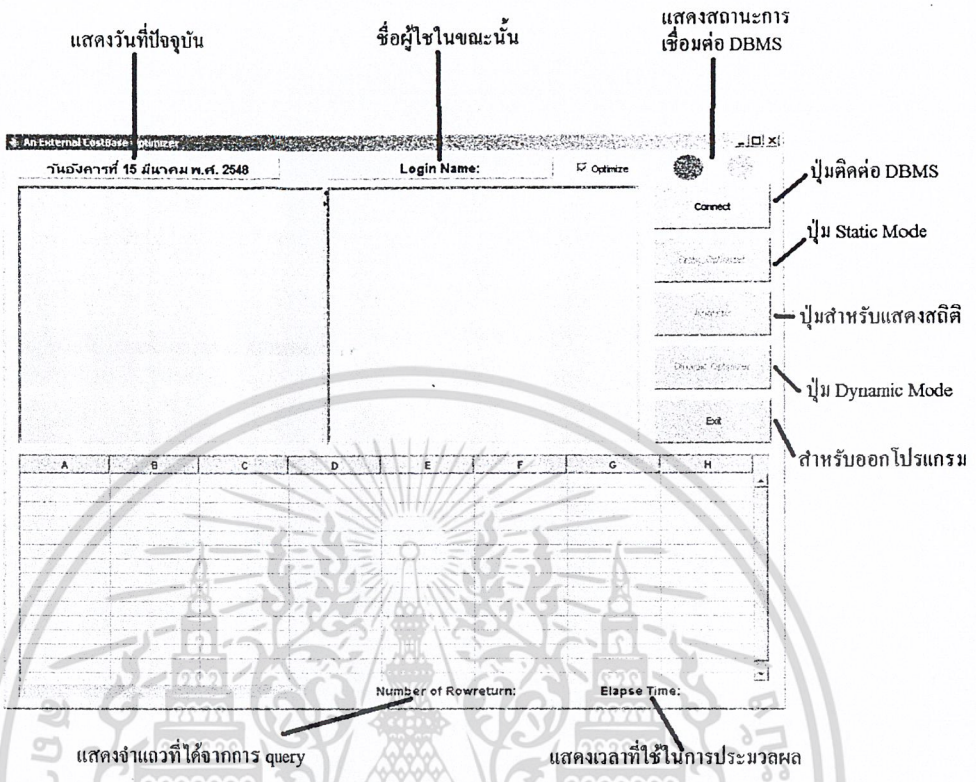
ดังนั้น จึงควรทำการพัฒนาระบบต่อไป โดยการการศึกษาและทำการทดลองการประมวลผลของ query ที่ซับซ้อนมากขึ้น เช่น การทำ subquery เพื่อหาอัลกอริทึมในการตรวจสอบและปรับแต่ง query ได้อย่างมีประสิทธิภาพยิ่งขึ้น หรืออาจทำการศึกษาวิธีการปรับแต่งในเวอร์ชันอื่นๆเพื่อเป็นตัวเลือกให้ผู้ใช้ใน

การใช้งานได้ ซึ่งหากทำการศึกษาข้อมูลใหม่ๆแล้วต้องการสถิติที่ไม่มีอยู่เดิมก็ควรทำการพัฒนาฐานข้อมูลที่เก็บสถิติให้มีการเก็บค่าต่างๆที่จำเป็นได้ และนำเอาสถิติที่เก็บมานี้ไปทำการคำนวณหาค่าต้นทุนของการประมวลผลที่แม่นยำยิ่งขึ้น เพื่อจะได้เลือกใช้เส้นทางการประมวลผลที่ดีที่สุดได้

ภาคผนวก ก.

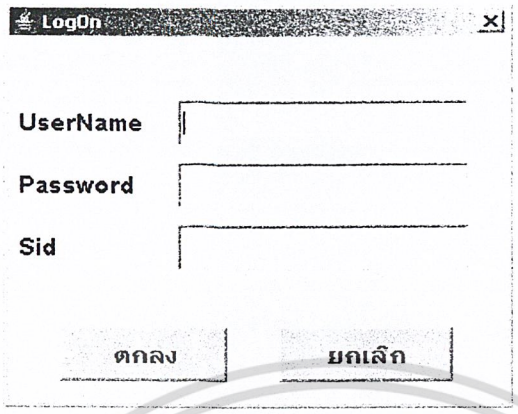
คู่มือการใช้โปรแกรม

เมื่อผู้ใช้ทำการเปิดโปรแกรมขึ้นมาจะมีลักษณะดังรูปที่ ก.1



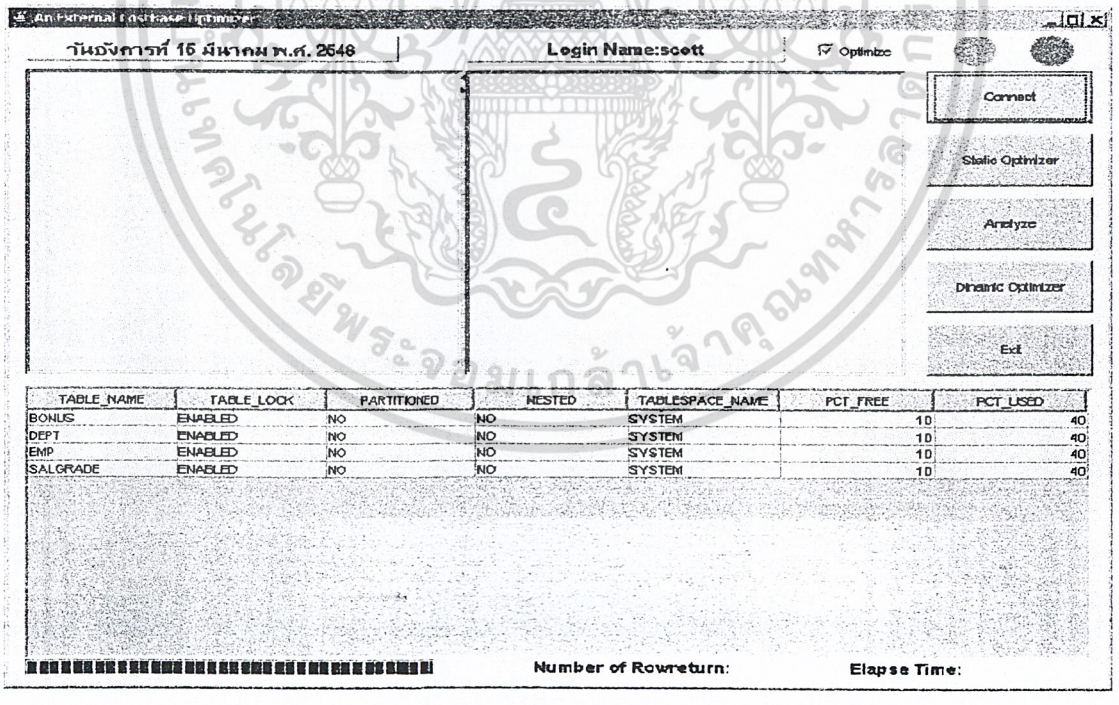
รูปที่ ก.1 แสดงลักษณะของโปรแกรม

1. ขั้นตอนแรกผู้ใช้ต้องทำการเข้าสู่ระบบ โดยการกดที่ปุ่ม Connect เมื่อกดปุ่ม Connect จะมี Dialog Box ขึ้นมาเพื่อให้ป้อน user, password, sid ดังรูปที่ ก.2



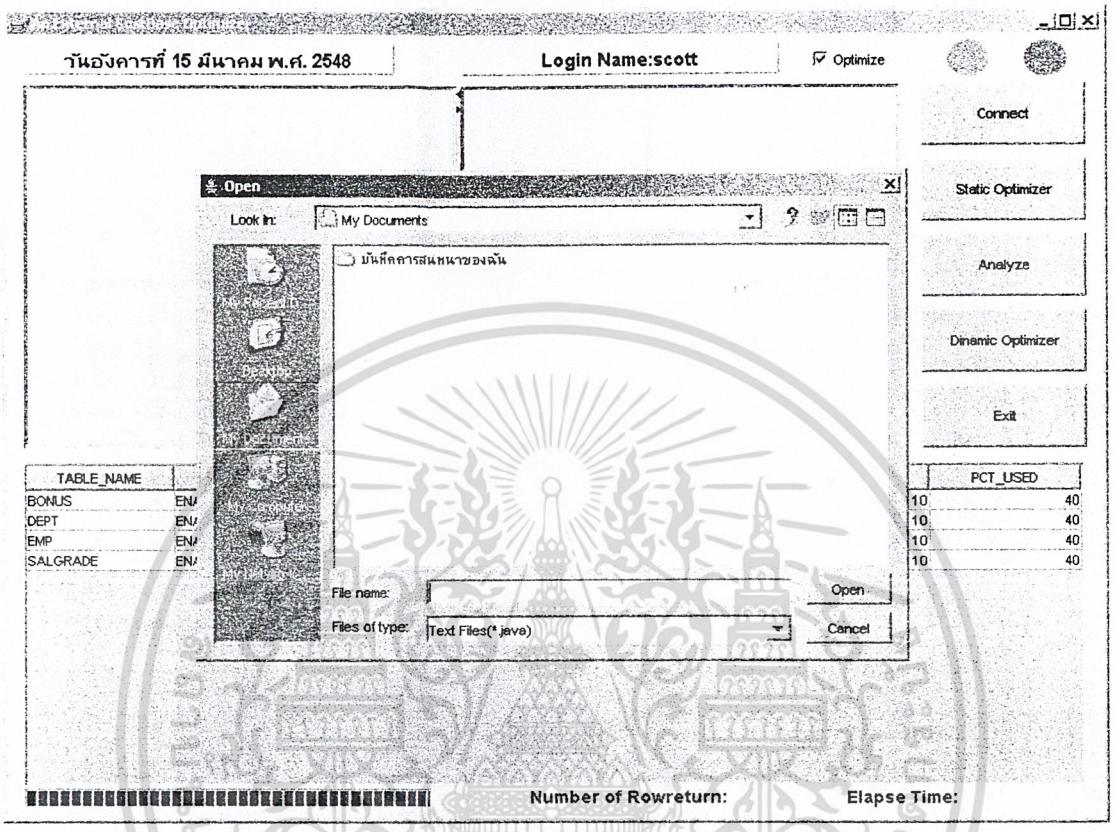
รูปที่ ก.2 Dialog Box ที่ใช้กรอกข้อมูลเพื่อเข้าใช้งาน

2. หากป้อนข้อมูลครบถ้วนและถูกต้องไฟสถานะ การเชื่อมต่อจะเป็นสีเขียว และจะแสดงตารางทั้งหมดของผู้ใช้คนนี้ดังรูปที่ ก.3



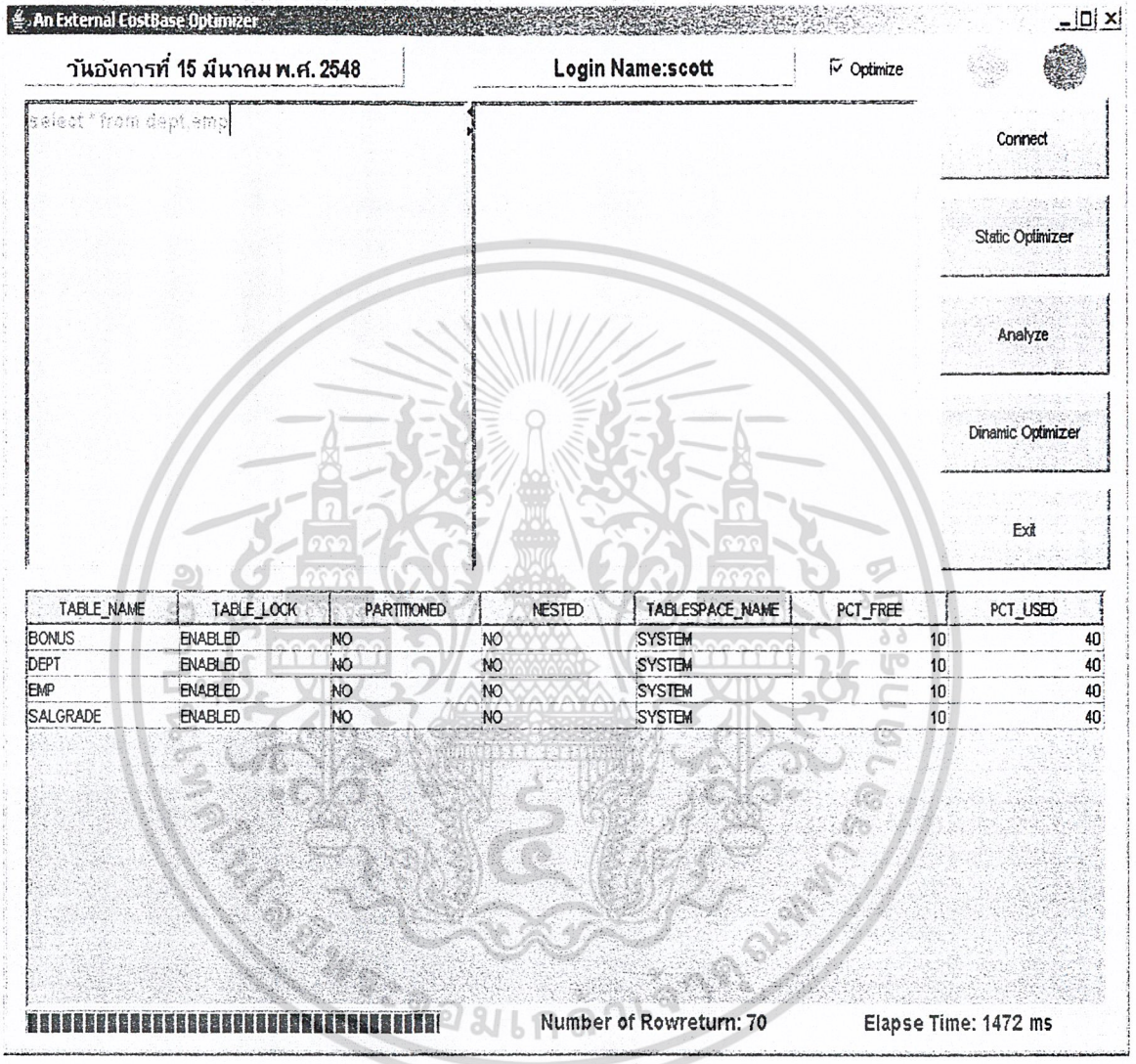
รูปที่ ก.3 แสดงการเข้าใช้งานโปรแกรม

3. ปุ่ม Static Optimizer ใช้ในการ optimize ซอร์ดโค้ด ซึ่งจะมี Dialog ขึ้นมาสำหรับให้เลือกไฟล์สำหรับแก้ไขดังรูปที่ ก.4 ซึ่ง Dialog นี้จะแสดงเฉพาะ file ที่นามสกุล .java เพื่อสำหรับหาเฉพาะ file ที่เขียนด้วยภาษาจาวา

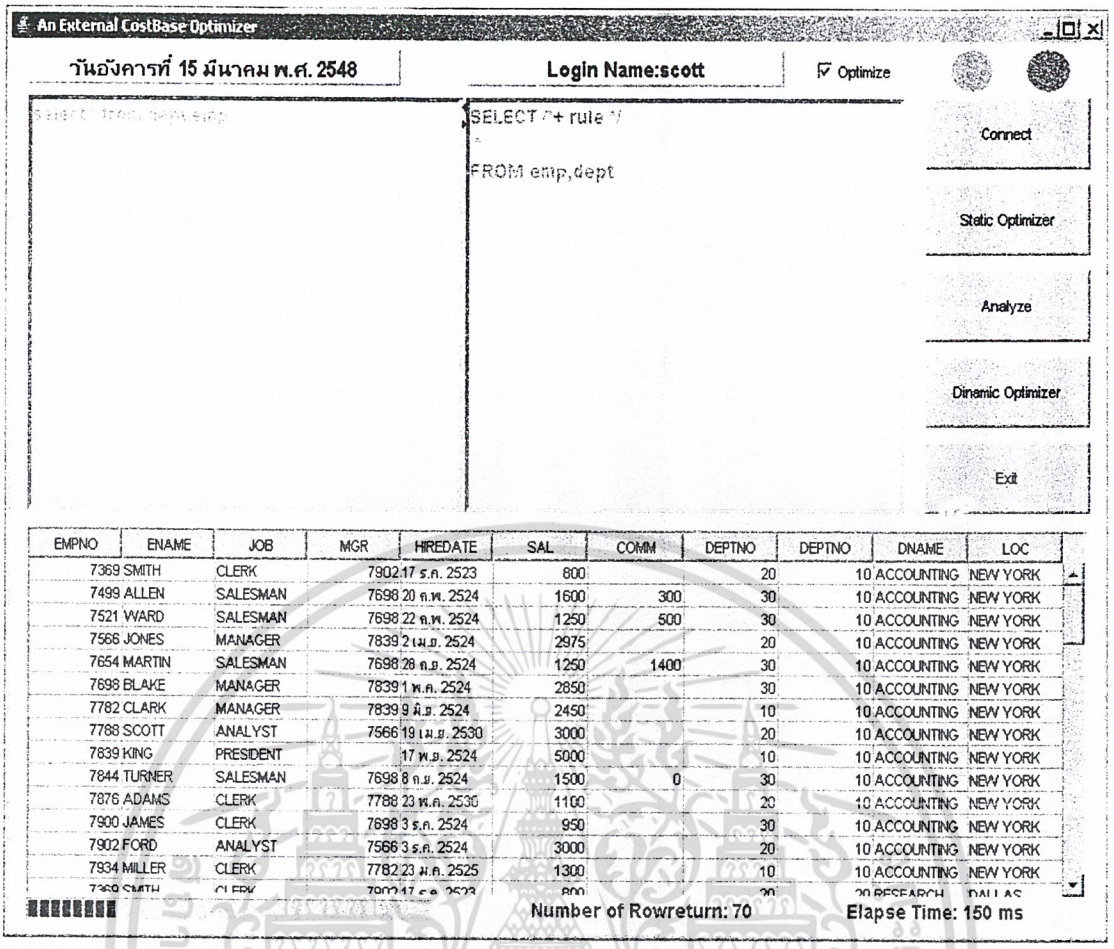


รูปที่ ก.4 แสดงการใช้งานในแบบสถิต

4. ปุ่ม Dynamic Optimizer มีไว้สำหรับทำการ query ผ่านโปรแกรม ซึ่งเราจะสามารถ query ผ่านโปรแกรมในส่วนที่กำหนดไว้ จากรูปจะเห็นว่าสามารถพิมพ์คำสั่ง sql ลงไปในหน้าต่างทางซ้ายได้ เมื่อพิมพ์เสร็จ ก็ให้กดปุ่ม F5 ก็จะมีคำสั่งที่ถูก Optimize ปรากฏขึ้นมาทางขวาและแสดงผลลัพธ์ของคำสั่ง จำนวนแถว รวมไปถึงเวลาที่ใช้ในการประมวลผล ดังรูปที่ ก.5



รูปที่ ก.5 การใช้งานในแบบพลวัต

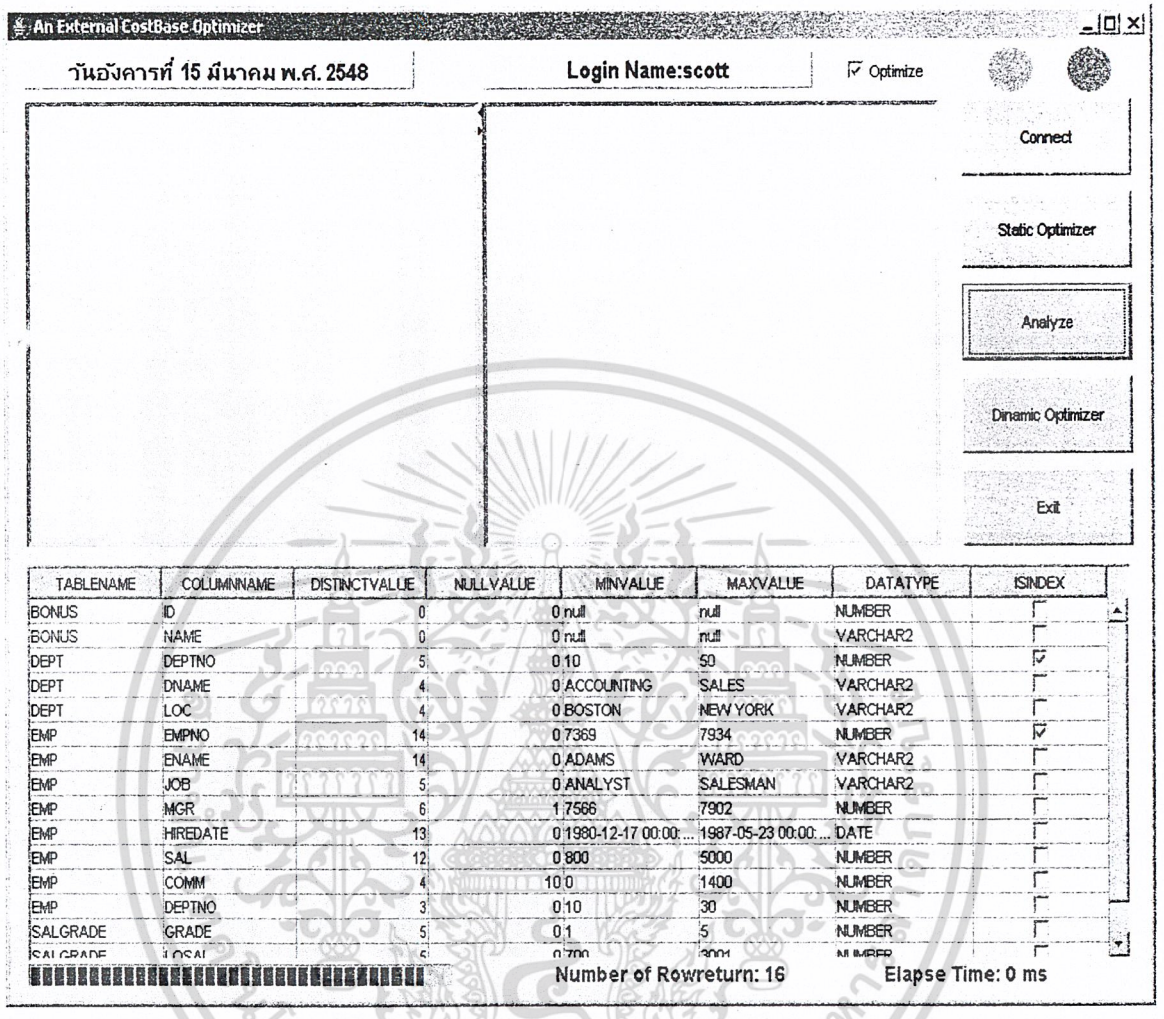


รูปที่ ก.6 แสดงผลการ optimize คำสั่งผ่านโปรแกรมในแบบพลวัต

จากรูปที่ ก.6 จะเห็นว่าทางด้านขวามีคำสั่ง SQL อีกคำสั่งที่ให้ผลลัพธ์เหมือนกับทางด้านซ้าย แต่ถูกปรับปรุงเพื่อให้ประมวลผลได้เร็วขึ้น ส่วนทางด้านล่างจะเป็นการแสดงผลลัพธ์ของคำสั่ง นอกจากนั้นยังมีการแสดง จำนวนแถวทั้งหมดของผลลัพธ์ และ เวลาที่ทั้งหมดที่ใช้ในการประมวลผล

หากเราเอา check box ตรง Optimize ออก จะทำให้ส่งสั่ง query ของผู้ใช้ไปโดยตรง โดยไม่ผ่านการ Optimize ของโปรแกรม

5. สำหรับปุ่ม Analyze จะไว้ใช้สำหรับวิเคราะห์ข้อมูลเพื่อทำการเก็บสถิติของผู้ที่ใช้ระบบอยู่ เมื่อกดปุ่มนี้จะมีข้อมูลทางสถิติ แสดงขึ้นมาทางด้านล่าง



รูปที่ ก. 7 แสดงผลลัพธ์ของการเก็บสถิติด้วยโปรแกรม

จากรูปที่ ก.7 จะเห็นข้อมูลทางสถิติของผู้ใช้ที่กำลังใช้ระบบอยู่ ซึ่งเวลาที่ใช้ในการวิเคราะห์สถิติขึ้นอยู่กับจำนวนข้อมูลของผู้ใช้

6. ปุ่ม Exit มีไว้เพื่อออกจากโปรแกรม พร้อมทั้งทำการปิด connection ที่เชื่อมต่ออยู่

บรรณานุกรม

[1] Mark Gurry. **"Oracle SQL Tuning Pocket Reference."** [Online].Available : http://www.oreilly.de/catalog/orsqltunpr/chapter/. January 2002.

[2] Connie Dialeris Green. "Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2).", Part No. A96533-02, October 2002.

[3] Ruth Baylis. "Oracle9i Database Administrator's Guide, Release 2 (9.2).", Part No. A96521-01, March 2002.

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

[5] Donald K. Burleson. **"Oracle Hight-Performance SQL Tunning."**, McGraw-Hill, 2001