การวิเคราะห์แผนการสืบค้นเพื่อประเมินประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ที่มีต่อคำสั่งเอสคิวแอลแบบซีเล็กชัน

Analysis of the Query Plan for Assessing Optimizer Performance of SQL Select Statemant

ชยาพร แก่นสาร์[1]

Chayaporn Kaensar[1]

บทคัดย่อ

ภาษาเอสคิวแอล (Structured Query Languages) เป็นองค์ประกอบหนึ่งที่สำคัญในโปรแกรมหรือระบบงานต่างๆ เพราะช่วยเข้าถึงและจัดการข้อมูลภายในฐานข้อมูลได้ ดังนั้นความเร็วหรือประสิทธิภาพการทำงานของแอพพลิเคชันส่วนหนึ่ง จึงเกี่ยวข้องกับรูปแบบคำสั่งภาษาเอสคิวแอลโดยตรง ฉะนั้นในบทความนี้จึงต้องการนำเสนอการวิเคราะห์ และประเมินถึงประสิทธิภาพการทำงานของดีบีเอ็มเอสที่มีต่อคำสั่งเอสคิวแอล โดยใช้แผนการสืบค้น (Query Plan) ซึ่งจะแบ่งการทำงานออกเป็น 4 ขั้นตอนหลักคือ 1) จัดเตรียมเครื่องมือและข้อมูลการทดสอบ โดยจะใช้ออราเคิล เวอร์ชัน 11.1 และ SQL Developer เวอร์ชัน 1.5.5 2) ออกแบบคำสั่งเอสคิวแอล 3) ประมวลผลคำสั่งเอสคิวแอลและสร้างแผนการสืบค้น และ 4) วิเคราะห์และสรุปผลการทำงานตามลำดับ ซึ่งผลการวิเคราะห์และทดสอบการทำงานพบว่ารูปแบบการเข้าถึงข้อมูล วิธีการลิงค์ตาราง การใช้ฟังก์ชันภายในคำสั่ง การใช้ประโยคสืบค้นย่อยแบบวนซ้ำ หรือการใช้ CASE และ UNION นั้นจะส่งผลต่อประสิทธิภาพการเข้าถึงข้อมูลของออราเคิล โดยประโยชน์ที่คาดว่าจะได้รับจากการทดสอบดังกล่าวคือจะช่วยให้นักพัฒนาระบบหรือผู้ใช้ที่สืบค้นข้อมูลด้วยภาษาเอสคิวแอลได้เข้าใจถึงปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพการทำงานของดีบีเอ็มเอส รวมทั้งสามารถออกแบบและสร้างคำสั่งเอสคิวแอลเพื่อสืบค้นข้อมูลให้ทำงานได้อย่างเหมาะสมทั้งในด้านของเวลาและการใช้งานทรัพยากรในระบบ

คำสำคัญ: การปรับปรุงการทำงานในคำสั่งเอสคิวแอล การปรับปรุงการทำงานของฐานข้อมูล การสร้างแผนการสืบค้น ออราเคิลเวอร์ชัน 11g

Abstract

SQL (Structured Query Language) is an important element of software applications because it is a database computer declarative language designed for managing data in the Database System. Therefore, SQL influences system performance directly. This article presents an analysis and evaluation of the Oracle database performance by using the query plan. This work has been divided into 4 steps. First, an oracle

[1] อาจารย์ ภาควิชาคณิตศาสตร์ สถิติและคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยอุบลราชธานี
Tel. 0-4535-3401 Ext. 4518, E-mail: scchayka@ubu.ac.th

รับเมื่อ 5 กรกฎาคม 2554 ตอบรับเมื่อ 16 มีนาคม 2555

database environment is set up and a database object is created using Oracle version 11g (Release 1) and SQL Developer version 1.5.5 is used as the main tool for SQL execution. Secondly, an SQL select statement was designed according to the query plan subject in different ways. Thirdly, the SQL command was executed and a query plan was generated. Finally, we discuss and conclude the results. In summary, we found that using a query plan based on access path, join method, and an SQL pattern such as the oracle function, correlated sub query, and the CASE and UNION function will influence system performance. Further, the finding of this article may help developers understand how to perform proper SQL commands in order to improve database performance and minimize resource utilization.

Keywords: SQL Tuning, Database Performance Improvement, Query Plan Generation, Oracle 11g

1. บทนำ

ปัจจุบันพบว่ามีการใช้ข้อมูลเพื่อสนับสนุนการตัดสินใจภายในองค์กรธุรกิจ เช่น การสร้างรายงานเพื่อสรุปสารสนเทศที่เป็นประโยชน์แก่ผู้บริหาร หรือการค้นหาและจัดการข้อมูลในระบบนั้นมีแนวโน้มที่เพิ่มขึ้นอย่างต่อเนื่องและรวดเร็ว ซึ่งเรื่องดังกล่าวจะส่งผลกระทบโดยตรงต่อความเร็วในการประมวลผลต่างๆ ในระบบงาน เนื่องจากโดยส่วนใหญ่นักพัฒนาระบบมักจะฝังโค้ดภาษาเอสคิวแอล (Structured Query Language) ลงในระบบหรือแอพพลิเคชันเพื่อใช้เข้าถึงและจัดการกับข้อมูลต่างๆ ภายในฐานข้อมูลได้ ซึ่งหากส่วนคำสั่งเหล่านั้นไม่มีประสิทธิภาพหรือใช้เวลาในการประมวลผลที่นานแล้วก็จะทำให้แอพพลิเคชันหรือระบบงานที่เกี่ยวข้องใช้เวลาในการประมวลผลที่นานขึ้นด้วย

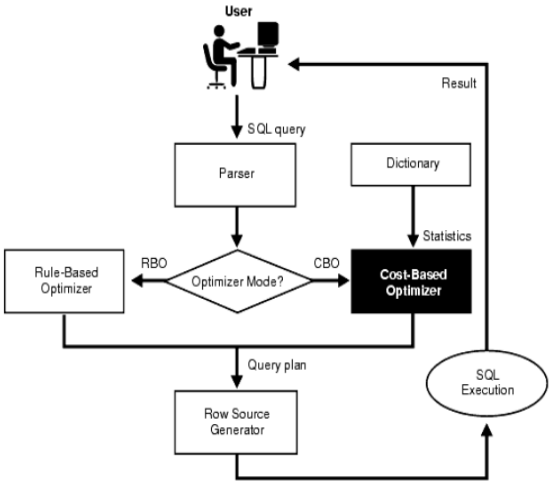
ดังนั้นจากประเด็นดังกล่าว แนวคิดการปรับปรุงการทำงานในคำสั่งเอสคิวแอล (SQL Tuning) ซึ่งหมายถึงกระบวนการที่ใช้ตรวสอบและปรับปรุงโครงสร้างคำสั่งเอสคิวแอลเพื่อให้แอพพลิเคชันหรือการทำงานต่างๆ นั้นสามารถจัดการกับข้อมูลในฐานข้อมูลได้อย่างมีประสิทธิภาพทั้งในด้านเวลาและการใช้ทรัพยากร [1] จึงเป็นเรื่องสำคัญมากและได้ถูกศึกษาและพัฒนาในงานวิจัยต่างๆ มาอย่างต่อเนื่องมากมาย ดังจะเห็นได้จาก [2]-[6]

ฉะนั้นในบทความนี้จึงต้องการนำเสนอถึงวิธีการวิเคราะห์และประเมินถึงประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ (Optimizer) ซึ่งเป็นคอมโพเน็นต์สำคัญในดีบีเอ็มเอสที่ทำหน้าที่ประมวลผลคำสั่งเอสคิวแอลต่างๆ โดยใช้เทคนิคหรือข้อกำหนดต่างๆ ที่กำหนดขึ้นมา [7] โดยวิธีการสำคัญหนึ่งคือการวิเคราะห์จากแผนการสืบค้น (Query Plan) ซึ่งเป็นข้อมูลสำคัญที่ใช้เก็บลำดับการประมวลผลคำสั่งเอสคิวแอลของอ๊อพติไมเซอร์ซึ่งจะช่วยให้ผู้ใช้ที่ต้องการสืบค้นข้อมูลด้วยภาษาเอสคิวแอลได้ทราบและเข้าใจถึงปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพการทำงานของระบบจัดการฐานข้อมูล ทั้งในด้านเวลาและการใช้งานทรัพยากรในระบบ โดยได้จัดแบ่งการนำเสนอบทความออกเป็น 4 ส่วนคือ แนวคิดการประมวลผลคำสั่งเอสคิวแอล สภาพแวดล้อมที่ใช้ทดสอบระบบ ผลการทดลอง อภิปรายและสรุปผลตามลำดับ

2. แนวคิดการประมวลผลคำสั่งเอสคิวแอล

โดยทั่วไปผู้ใช้จะเปลี่ยนแปลง หรือแก้ไขข้อมูลต่างๆ ภายในฐานข้อมูลผ่านซอฟต์แวร์ที่เรียกว่าระบบจัดการฐานข้อมูลหรือดีบีเอ็มเอส (DBMS: Database Management System) โดยใช้ภาษาเอสคิวแอล ซึ่งเป็นภาษามาตรฐานที่ถูกออกแบบมาเพื่อจัดการข้อมูลผ่านระบบจัดการฐานข้อมูลเชิงสัมพันธ์ (RDBMS)

โดยคอมโพเน็นต์ในดีบีเอ็มเอสที่นับเป็นกลไกสำคัญในการจัดการงานดังกล่าวได้นั้นเรียกว่าอ๊อพติไมเซอร์ (Optimizer) ซึ่งสามารถจัดการและประมวลผลคำสั่งสืบค้นและคำสั่งจัดการข้อมูลต่างๆ ได้ โดยมีขั้นตอนในการประมวลผลคำสั่งเอสคิวแอลดังนี้คือ [8]-[10] (รูปที่ 1)



รูปที่ 1 การทำงานของอ๊อพติไมเซอร์ในการประมวลผลคำสั่ง

1. เมื่อระบบได้รับคำสั่งเอสคิวแอลจากผู้ใช้ระบบจะส่งชุดคำสั่งดังกล่าวไปยังคอมโพเน็นต์ที่เรียกว่า Parser ให้ทำการวิเคราะห์รูปแบบและความถูกต้องของประโยค (Syntax Checking) รวมถึงตรวจสอบถึงความถูกต้องของอ็อบเจ็คหรือตารางต่างๆ ที่มีการอ้างถึงในระบบฐานข้อมูล ซึ่งผลลัพธ์จะได้รูปแบบคำสั่งเอสคิวแอลที่ผ่านการตรวจสอบตามโครงสร้างที่ประโยคที่ถูกต้องแล้ว (Verified SQL Query)

2. จากนั้นจะส่งผลคำสั่งที่ผ่านการตรวจสอบในข้อ 1 ไปยังอ๊อพติไมเซอร์ เพื่อวิเคราะห์และสร้างแผนการสืบค้น โดยมี 2 รูปแบบคือแบบ RBO (Ruled Based Optimizer) ซึ่งใช้กฎความสัมพันธ์เพื่อสร้างการประมวลผลคำสั่ง หรือสร้างอินเด็กซ์ให้กับข้อมูล ในขณะที่ CBO (Cost Based Optimizer) จะใช้ข้อมูลสถิติเข้ามาช่วยประเมินและกำหนดแผนการสืบค้นก่อนประมวลผลหรือรันคำสั่งจริง เช่น จำนวนแถวข้อมูลหรือไบต์ข้อมูลที่เข้าถึง เป็นต้น [11],[12]

3. จากนั้นเมื่อ Row Source Generator ได้รับแผนการสืบค้นแล้ว ก็จะสร้างความสัมพันธ์และลำดับการทำงานในรูปแบบของทรี (Tree Structure) ก่อนส่งโครงสร้างดังกล่าวไปยัง SQL Executor เพื่อประมวลผลคำสั่งตามแผนดังกล่าวก่อนส่งผลลัพธ์ไปแสดงผลยังผู้ใช้ต่อไป

2.1 ความหมายและความสำคัญของแผนการสืบค้น (Definition and Benefits of Query Plan)

จากขั้นตอนการทำงานในรูปที่ 1 พบว่าการสร้างแผนการสืบค้นนั้น นับเป็นขั้นตอนหนึ่งที่สำคัญเนื่องจากเป็นส่วนกำหนดลำดับการทำงานของคำสั่งและโอเปอเรชันให้กับอ๊อพติไมเซอร์เพื่อดำเนินการรันคำสั่งนั้นๆ

โดยแผนการสืบค้น (Query Plan หรือ Execution Plan) หมายถึงแผนการทำงานที่เก็บลำดับการทำงานของคำสั่งเอสคิวแอล เช่น select, update, insert และ delete โดยอ๊อพติไมเซอร์จะประมวลผลคำสั่งตามข้อมูลต่างๆ ที่ระบุในแผนดังกล่าว เช่น ลำดับการทำงาน การเข้าถึงข้อมูล วิธีการลิงค์ข้อมูล หรือรูปแบบการใช้โอเปอเรเตอร์ต่างๆ เป็นต้น ซึ่งข้อดีคือจะทำให้ผู้ใช้ได้ทราบถึงรูปแบบและการจัดการของดีบีเอ็มเอสในระดับภายใน นั่นคือหากทราบการทำงานของแผนสืบค้น ก็จะสามารถทราบได้ว่ามีโครงสร้างคำสั่ง ลำดับการทำงาน หรือภาระการโหลดงานที่มากหรือน้อยเกิดขึ้นที่โอเปอเรชันใด ซึ่งก็ช่วยให้สามารถปรับแต่งโครงสร้างคำสั่งเอสคิวแอลที่เหมาะสมได้ ดังรูปที่ 2

```
-------------------------------------------------
| Id  | Operation                    | Name     |
-------------------------------------------------
|   0 | SELECT STATEMENT             |          |
| * 1 |  FILTER                      |          |
|   2 |   NESTED LOOPS               |          |
|   3 |    TABLE ACCESS FULL         | EMP      |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPT    |
| * 5 |     INDEX UNIQUE SCAN        | PK_DEPT  |
| * 6 |   TABLE ACCESS FULL          | SALGRADE |
-------------------------------------------------

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

   1 - filter( NOT EXISTS
    (SELECT 0 FROM "SALGRADE" "SALGRADE" WHERE
     "HISAL">=:B1 AND "LOSAL"<=:B2))
   5 - access("DEPT"."DEPTNO"="EMP"."DEPTNO")
   6 - filter("HISAL">=:B1 AND "LOSAL"<=:B2)
```

รูปที่ 2 ตัวอย่างแผนการสืบค้น [1]

โดยแผนการสืบค้นประกอบด้วยข้อมูลพื้นฐานต่างๆ ดังนี้คือ

1. Operation เป็นรายการทำงานหรือคำสั่งย่อยของโอเปอร์ชันต่างๆ ที่อ๊อพติไมเซอร์จะประมวลผล โดยจะเริ่มรันจากโหนดลูกหรือ Child Node ก่อน (โอเปอเรชันที่อยู่ด้านขวามือสุด) จากนั้นรันโหนดแม่หรือ Parent Node (โอเปอเรชันที่อยู่ใกล้ที่สุดของโหนดลูกนั้น) แต่หากมีโอเปอเรชันที่ระดับเท่ากัน ก็ให้เริ่มรันที่รหัสด้านบนก่อน เช่น จากรูปที่ 2 จะมีลำดับการทำงานคือ 5, 3, 4, 2, 6 และ 1 ตามลำดับ [1]

2. Name เป็นชื่อดาต้าเบสอ็อบเจ็คที่มีการอ้างถึง เช่น ตาราง หรืออินเด็กซ์ที่มีการใช้งาน

3. Rows, Bytes เป็นจำนวนแถว/ไบต์ ที่มีการอ่านหรือโหลดข้อมูลเข้าในระบบ

4. Cost เป็นค่าน้ำหนักที่ระบบสูญเสียไปในการประมวลผลคำสั่งเอสคิวแอลนั้น ซึ่งจะไม่มีหน่วยการวัด โดยอ๊อพติไมเซอร์จะคำนวณค่าดังกล่าวจากการใช้งานซีพียู (CPU Cost) และค่าการติดต่อใช้งานกับดิสก์หรือส่วนแสดงผล (I/O Cost) ดังแสดงการคำนวณในสมการดัง [11], [12]

5. %CPU เป็นค่าอัตราส่วนที่อ๊อพติไมเซอร์ใช้ประมวลผลและทำงานสำหรับโอเปอเรชันนั้นๆ

6. Time เป็นเวลาที่ใช้ในการประมวลผลคำสั่ง มีหน่วยเป็น Millisecond (ms)

โดยขั้นตอนการวิเคราะห์แผนการสืบค้นนั้น จะเริ่มด้วยการสร้างคำสั่งเอสคิวแอล ดังตัวอย่างคำสั่ง (1) ซึ่งประกอบด้วยส่วนการลิงค์ตาราง (Join Table) และการใช้ประโยคสืบค้นย่อย (Sub Query) จากนั้นจึงสร้างแผนการสืบค้น โดยรันคำสั่งผ่านโปรแกรม SQL Developer ซึ่งจะได้ผลลัพธ์ดังรูปที่ 2

```
select  ename,job,sal,dname
from  emp,dept
where dept.deptno = emp.deptno and not exists
      ( select *                                    (1)
      from salgrade
      where emp.sal between losal and hisal );
```

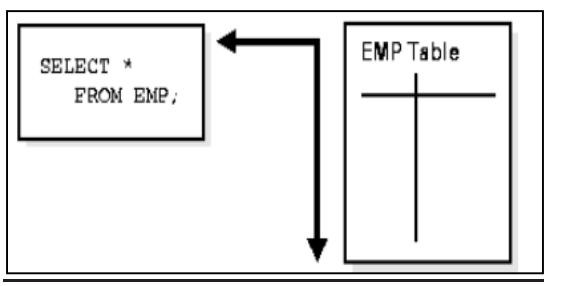
โดยจากรูปที่ 2 จะพบว่าการศึกษาหรือวิเคราะห์ข้อมูลจากแผนการสืบค้นนั้นจะช่วยให้ผู้ใช้เข้าใจกระบวนการทำงานของอ๊อพติไมเซอร์ที่มีต่อการประมวลผลคำสั่งเอสคิวแอลได้ ซึ่งผู้ใช้สามารถนำหลักการเหล่านี้ไปออกแบบและสร้างเอสคิวแอลที่เหมาะสม เพื่อลดการใช้งานทรัพยากรระบบ และเพิ่มความเร็วในการทำงานได้มากขึ้น โดยในบทความนี้จะแบ่งการนำเสนอปัจจัยที่มีผลต่อการสร้างแผนการสืบค้น ซึ่งส่งผลต่อการทำงานของอ๊อพติไมเซอร์ เช่น รูปแบบการเข้าถึงข้อมูล และวิธีการลิงค์ข้อมูลระหว่างตาราง เป็นต้น [11], [14]

2.2 รูปแบบการเข้าถึงข้อมูล (Access Path)

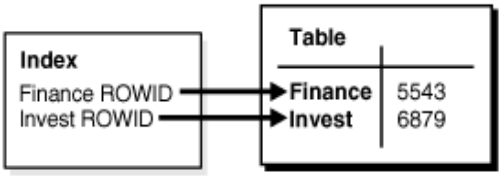
เป็นวิธีหรือรูปแบบการเข้าถึงข้อมูลในฐานข้อมูล โดยเบื้องต้นสามารถแบ่งได้ 2 รูปแบบหลักคือ [9], [13], 14]

2.2.1 แบบสแกนทั้งหมด (Full Table Scan หรือ FTS) จะตรวจสอบและอ่านข้อมูลทั้งหมดในทุกแถวของตาราง จากนั้นจึงทำการเลือกข้อมูลที่ไม่เกี่ยวข้อง (Filter Out) ออกจากบัฟเฟอร์หรือหน่วยความจำชั่วคราว ซึ่งจากรูปที่ 3 เป็นการแสดงการอ่านข้อมูลในทุกแถวของตารางเมื่อมีการค้นหาแบบแสดงข้อมูลทั้งหมดด้วยคำสั่ง “select * from emp”

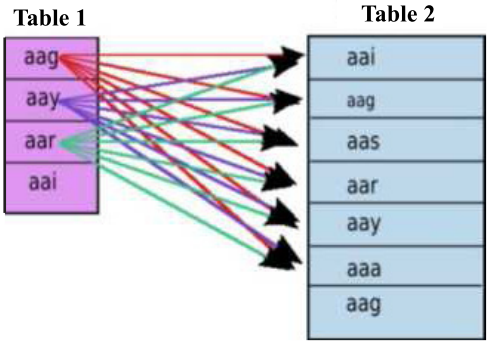
2.2.2 แบบสแกนโดยใช้อินเด็กซ์ (Index Lookup Scan) เป็นวิธีค้นหาข้อมูลด้วยอินเด็กซ์ที่ได้ถูกระบุหรือสร้างไว้ในคอลัมน์ (Indexed Column) ในตอนที่มีการสร้างหรือระบุข้อกำหนด (Constraint) ภายในตารางไว้แล้ว โดยจะประกอบด้วยข้อมูลอินเด็กซ์ และที่อยู่ข้อมูล



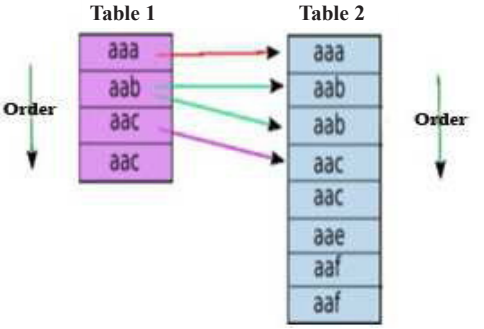
รูปที่ 3 รูปแบบสแกนข้อมูลทั้งหมด (Full Table Scan)



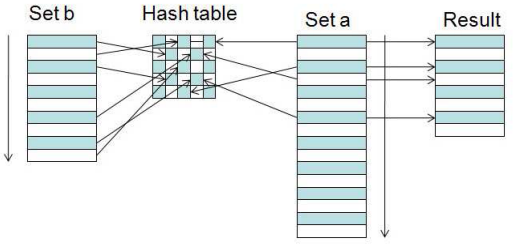
รูปที่ 4 รูปแบบใช้อินเด็กซ์ (Index Lookup Scan)



รูปที่ 5 การ Join ตารางแบบ Nested Loop



รูปที่ 6 การ Join ตารางแบบ Merge Join



รูปที่ 7 การ Join ตารางแบบ Hash Join

หรือรหัสแถว (Row ID) ซึ่งเป็นค่าข้อมูลภายในคอลัมน์นั้นๆ ดังนั้นเมื่อออราเคิลมีการใช้อินเด็กซ์ในคอลัมน์ดังกล่าว ก็จะอ่านค่าข้อมูลที่อยู่ในอินเด็กซ์โดยตรง แทนการอ่านข้อมูลจากตารางในดิสก์ ซึ่งข้อดีคือสามารถทำงานได้เร็วขึ้น เนื่องจากลดโอเปอเรชันในการติดต่อกับดิสก์ในระบบ โดยรูปที่ 4 เป็นการลิงค์ (Mapping) ข้อมูลระหว่างโครงสร้างข้อมูลแบบอินเด็กซ์ไปยังคอลัมน์ในตารางที่ถูกกำหนดหรือสร้างเป็นอินเด็กซ์ (Index Column) โดยใช้ข้อมูล Row ID ซึ่งเป็นค่าที่แสดงลำดับหมายเลขของแถวข้อมูลลิงค์ไปยังตารางที่จะค้นหาข้อมูล ซึ่งจะทำให้การค้นหาทำได้รวดเร็วขึ้น

2.3 การลิงค์ข้อมูลระหว่างตาราง (Join Method)

เป็นวิธีการเชื่อมหรือลิงค์ข้อมูล (Join) ระหว่างตาราง ซึ่งสามารถจำแนกรูปแบบการลิงค์ได้ 3 วิธีดังนี้คือ

2.3.1 Nested Loop Join เป็นวิธีการลิงค์ข้อมูลโดยจะตรวจสอบข้อมูลทีละแถวในแต่ละตาราง กล่าวคือ อ๊อพติไมเซอร์จะตรวจสอบข้อมูลในตารางด้านนอก (Outer Table) ก่อน จากนั้นจึงนำข้อมูลแต่ละแถวมาตรวจสอบในทุกข้อมูลของตารางด้านใน (Inner Table) ดังแสดงในรูปที่ 5

2.3.2 Merge Join (Sort Merge Join) มีการทำงานคล้ายแบบแรก แต่ต่างกันที่ก่อนเริ่มการทำงาน จะมีการจัดเรียงลำดับข้อมูล (Sort) ในแต่ละตารางก่อนทำการลิงค์ดังรูปที่ 6

ข้อดีคือจะช่วยลดเวลาในการลิงค์ตารางได้ เนื่องจากหากไม่พบข้อมูลหรือเงื่อนไขในการลิงค์ตารางแล้ว ระบบก็จะหยุดการทำงานในตารางนั้นๆ ทันที ดังรูปที่ 6 ซึ่งจะเห็นว่า Table 1 และ 2 มีการจัดเรียงลำดับข้อมูลก่อนทำการลิงค์ตาราง

2.3.3 Hash Join เป็นการลิงค์ตารางซึ่งจะทำผ่านตารางแฮช (Hash Table) ที่ระบบได้สร้างขึ้นชั่วคราว โดยข้อมูลในตารางแฮชจะประกอบด้วยคีย์ข้อมูลที่ใช้ลิงค์ระหว่างตาราง และที่อยู่ของดาต้าบล็อค (Data Block) ที่ใช้เก็บค่าข้อมูลนั้น ซึ่งวิธีนี้จะช่วยให้ตารางหลักเข้าถึงข้อมูลในตารางย่อยได้เร็วขึ้น จากรูปที่ 7 พบว่ามีการสร้างตารางแฮชขึ้นมาชั่วคราว เพื่อจัดเก็บคีย์คอลัมน์ที่ใช้ลิงค์ระหว่างตารางเอาไว้ ก่อนใช้คีย์ในตารางแฮชเข้าถึงค่า

ข้อมูลที่เกี่ยวข้องต่อไป แต่ในบางกรณีอ๊อพติไมเซอร์อาจจะต้องตรวจสอบข้อมูลในตารางเดิมด้วย เพราะการสร้างตารางแฮชนั้น อาจจะมีค่าข้อมูลที่ซ้ำหรือชนกัน ดังนั้นจึงเป็นความยากและซับซ้อนมาก ที่จะออกแบบอัลกอริทึมเพื่อสร้างตารางแฮชที่ง่ายและทำงานได้อย่างรวดเร็ว

3. วัสดุ อุปกรณ์และวิธีการวิจัย

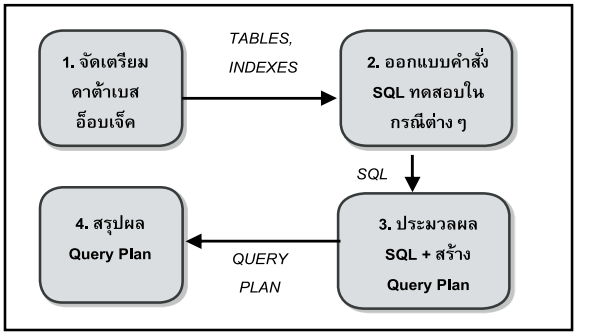
ในการเตรียมสภาพแวดล้อมการทำงานของอ๊อพติไมเซอร์นี้ จะใช้ดีบีเอ็มเอสออราเคิลเวอร์ชัน 11.1g (Release 1) และ SQL Developer เวอร์ชัน 1.5.5 เข้ามาใช้ในการทดสอบ เนื่องจากออราเคิลนั้นเป็นซอฟต์แวร์จัดการฐานข้อมูลที่มีความน่าเชื่อถือ และมีประสิทธิภาพการทำงานที่ดี ดังจะเห็นได้จากการสำรวจของสำนักข่าวรอยเตอร์ในปี 2010 ที่ระบุว่าเป็นดีบีเอ็มเอสที่มีผู้ใช้งานทั่วโลกมากที่สุด [15] ในขณะที่โปรแกรม SQL Developer ซึ่งเป็นเครื่องมือที่มีส่วนติดต่อกับผู้ใช้เป็นแบบกราฟิก (Graphic User Interface) ที่ช่วยผู้ใช้จัดการหรือแก้ไขข้อมูลในฐานข้อมูล ทั้งยังมีฟังก์ชันหรือบริการต่างๆ ที่ช่วยให้ทำงานได้ง่ายและสะดวกมากขึ้น เช่น ส่วนการจัดการและเรียกดูข้อมูลดาต้าเบสอ็อบเจ็ค เช่น ตาราง อินเด็กซ์ รวมทั้งส่วนการรันและแสดงผลคำสั่งหรือส่วนการสร้างสคริปต์และโค้ดต่างๆ เป็นต้น

โดยในการจัดเตรียมสภาพแวดล้อมสำหรับทดสอบระบบนั้น จะแบ่งการทำงานออกเป็น 4 ขั้นตอนหลัก ดังแสดงในรูปที่ 8

1. จัดเตรียมดาต้าเบสอ็อบเจ็ค เป็นขั้นตอนการจัดเตรียมตาราง และอินเด็กซ์เพื่อใช้ทดสอบคำสั่งเอสคิวแอล โดยจะใช้ตัวอย่างข้อมูลที่ติดตั้งมาพร้อมกับออราเคิล ของ Schema SCOTT, OE และ SH ซึ่งเป็นข้อมูลเกี่ยวกับส่วนพนักงาน แผนก ประวัติการทำงาน การจัดการสินค้า ราคาสินค้า และการสั่งซื้อสินค้า ซึ่งประกอบด้วย 6 ตารางหลักคือ

- **EMP2 (<u>EMPNO</u>**, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO) เก็บข้อมูลทั่วไปของพนักงานจำนวน 1.83 ล้านแถว
- **DEPT2 (<u>DEPTNO</u>**, DNAME, LOC) เก็บข้อมูลแผนกต่างๆ ในองค์กร จำนวน 1.04 ล้านแถว
- **JOB_HISTORY(<u>EMPID</u>**, JOB) เก็บข้อมูลประวัติการทำงานพนักงาน จำนวน 5 แสนแถว
- **COSTS (<u>PROD_ID</u>**, TIME_ID, PROMO_ID, CHANCEL_ID, UNIT_COST, UNIT_PRI) เก็บข้อมูลราคาและรายการขนส่งสินค้า จำนวน 0.82 แสนแถว
- **PRODUCTS(<u>PROD_ID</u>**, PROD_NAME, PRO_DESC, SUP_ID, PROD_LIST_PRICE, …., PROD_TOT) เก็บรายละเอียดสินค้า จำนวน 0.74 แสนแถว
- **ORDERS(<u>ORD_ID</u>**, ORD_TOTAL, ORD_DATE, CUST_NAME) เก็บข้อมูลรายการสั่งซื้อ จำนวน 1 ล้านแถว

โดยกำหนดให้มีคีย์หลัก (Primary Key) และคีย์นอก (Foreign Key) เพื่อลิงค์ความสัมพันธ์ระหว่างตาราง โดยแต่ละตารางจะถูกสร้างขึ้นด้วยคำสั่งเพิ่มข้อมูล (insert) โดยใช้ข้อมูลจากตารางเดิมมารันเพิ่มด้วยคำสั่ง “insert into …” ลงในตารางด้วยอัตราส่วนที่เพิ่มขึ้นและกระจายค่าอย่างเท่ากัน



รูปที่ 8 ขั้นตอนการทดสอบระบบ

2. ออกแบบคำสั่ง เป็นการสร้างคำสั่งเอสคิวแอลในรูปแบบและกรณีต่างๆ ที่แตกต่างกัน โดยในการทดสอบนี้จะทดสอบเฉพาะในส่วนคำสั่ง SELECT เท่านั้น เพราะโครงสร้างดังกล่าวสามารถสร้างโอเปอเรชันที่ซับซ้อนและหลากหลายได้มากกว่ารูปแบบ INSERT UPDATE หรือ DELETE เช่น การสร้างการเข้าถึงข้อมูลแบบมีเงื่อนไข (WH-Clause) การจัดเรียงข้อมูล (Sorting) การลิงค์

ข้อมูลระหว่างตาราง การสร้าคำสั่งสืบค้นย่อย (Sub Query) หรือการใช้งาน Aggregation Function เป็นต้น นอกจากนี้ คำสั่ง SELECT ยังเกี่ยวข้องกับการอ่านข้อมูลเพียงอย่างเดียว ส่วนผลการทำงานในคำสั่งอื่นๆ จะมีรูปแบบหรือปัจจัยอื่นร่วมด้วย เช่น นโยบายการอ่านเขียนข้อมูล (Read/Write Policy) การติดต่อกับดิสก์ (I/O Operation) หรือการล้างข้อมูลในบัฟเฟอร์ (Flushing) เป็นต้น ซึ่งผลจากขั้นตอนนี้จะได้คำสั่งเอสคิวแอลในรูปแบบต่างๆ โดยถูกกำหนดขึ้นตามรูปแบบและประเด็นที่เกี่ยวข้องกับหลักการเพิ่มประสิทธิภาพการทำงานของคำสั่งเอสคิวแอล [7]

3. ประมวลผลคำสั่ง ในขั้นตอนนี้จะทำการประมวลผลคำสั่งเอสคิวแอลโดยใช้รูปแบบ *"explain plan for sql statement"* ซึ่งผลจากการรันคำสั่งดังกล่าว จะทำให้ระบบสร้างแผนการสืบค้นขึ้นมา จากนั้นก็จะทำการตรวจสอบและแสดงข้อมูลในตารางดังกล่าวด้วยคำสั่ง *"select * from table(dbms_xplan.display)"*

4. สรุปผล เป็นขั้นตอนสุดท้าย โดยจะสรุปและประเมินผลการทำงานที่อ๊อพติไมเซอร์ใช้ในการวิเคราะห์และรันคำสั่งเอสคิวแอลในขั้นตอนแรก

4. ผลการทดลอง

ในการทดสอบประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ที่มีต่อคำสั่งเอสคิวแอลนั้น มีวัตถุประสงค์เพื่อตรวจสอบว่าเมื่อรันคำสั่งเอสคิวแอลด้วยรูปแบบหรือวิธีลิงค์ข้อมูลที่แตกต่างกันแล้ว อ๊อพติไมเซอร์จะสร้างแผนการสืบค้นและมีประสิทธิภาพการทำงานที่ต่างกันหรือไม่ จากนั้นจึงจะเปรียบเทียบและวิเคราะห์ผลที่เกิดขึ้น ซึ่งจะแบ่งการทดสอบออกเป็น 2 ด้านดังนี้คือ

4.1 การทดสอบการเข้าถึงข้อมูล

การทดสอบนี้ มีวัตถุประสงค์เพื่อศึกษาว่าอ๊อพติไมเซอร์จะมีปัจจัยในการพิจารณาเลือกรูปแบบการเข้าถึงข้อมูลต่างๆ อย่างไร และรูปแบบการเข้าถึงข้อมูลที่แตกต่างกันนั้น จะมีประสิทธิภาพการทำงานต่างกันอย่างไร โดยจะแบ่งกรณีทดสอบออกเป็น 2 ด้านคือแบบระบุและไม่ระบุเงื่อนไขในคำสั่ง SELECT ดังนี้คือ [7], [11]

4.1.1 การค้นหาข้อมูลทั้งหมด โดยทดสอบเพื่อค้นหาว่าอ๊อพติไมเซอร์จะเลือกใช้การเข้าถึงข้อมูลรูปแบบใด เมื่อมีการรันคำสั่งเข้าถึงข้อมูลทั้งหมดในตาราง ด้วย *"select * from table_name"* ซึ่งผลลัพธ์จากการทดสอบสามารถสรุปผลในตาราง 1

ตารางที่ 1 เปรียบเทียบผลลัพธ์ที่เกิดขึ้นใน Query Plan เมื่อมีการรันคำสั่ง "select * from emp2"

จำนวนข้อมูล (แถว)	รูปแบบการเข้าถึงข้อมูลที่อ๊อพติไมเซอร์เลือกใช้	Cost (%CPU)	Time (ms)
1,830,000	Table Access Full	1	00:00:37
28,000	Table Access Full	0	00:00:01

จากตารางที่ 1 พบว่าในตารางขนาดใหญ่แบบแรกซึ่งมีจำนวนข้อมูล 1.83 ล้านแถวนั้น อ๊อพติไมเซอร์จะเลือกการเข้าถึงข้อมูลแบบ Full Table Scan ซึ่งใช้ Cost และ Time ในการประมวลผลไปทั้งสิ้น 1% และ 37 ms ตามลำดับ ในขณะที่ตารางขนาดเล็กแบบที่ 2 ซึ่งมีการลดปริมาณข้อมูลลงอย่างเท่ากันให้เหลือเพียง 28,000 แถว พบว่าอ๊อพติไมเซอร์ยังคงเลือกการเข้าถึงข้อมูลรูปแบบเดิม นั่นคือ Full Table Scan แต่มี Cost และ Time ที่ลดลงคือเพียง 1 ms

ดังนั้นจึงสรุปได้ว่าการค้นหาข้อมูลทั้งหมดในตารางซึ่งมีปริมาณที่มากหรือน้อย จะไม่มีผลต่อการเลือกรูปแบบการเข้าถึงข้อมูลของอ๊อพติไมเซอร์ เนื่องจากระบบจะเลือกแบบ Full Table Scan เพราะอ๊อพติไมเซอร์สามารถเข้าถึงและอ่านข้อมูลในดาต้าบล็อคได้ง่ายและสะดวกกว่าวิธีการอื่นๆ [7], [13], [14]

4.1.2 การค้นหาข้อมูลโดยใช้เงื่อนไข ซึ่งมีวัตถุประสงค์เพื่อทดสอบว่าหากมีการเข้าถึงข้อมูลเพียงบางส่วนหรือเพิ่มส่วนการกำหนดเงื่อนไขในคำสั่งเอสคิวแอลแล้วนั้น อ๊อพติไมเซอร์จะมีรูปแบบการเข้าถึงข้อมูลในระบบอย่างไร โดยเบื้องต้นจะทำการสร้างคำสั่งเอสคิวแอลโดยทดสอบด้วยตาราง EMP2 ดังคำสั่งที่ (2)

```
explain plan for
select empno, ename, deptno, sal
from emp2
where deptno > 7900
```
(2)

จากนั้นเพิ่มการทดสอบคำสั่งที่ (2) ด้วยการสร้างอินเด็กซ์ (Index) บนคอลัมน์ DEPTNO เพื่อรองรับการค้นหาโดยผ่านเงื่อนไข WH-Clause DEPTNO ในตาราง EMP2 ซึ่งสามารถสรุปผลการรันคำสั่งและผลลัพธ์ใน Query Plan ดังตารางที่ 2

ตารางที่ 2 เปรียบเทียบผลลัพธ์ที่เกิดขึ้นใน Query Plan เมื่อมีการรันคำสั่งที่ 2 แบบใช้และไม่ใช้อินเด็กซ์

รูปแบบ	คำสั่งที่ 2 ไม่ใช้ Index	คำสั่งที่ 2 ใช้ Index บนคอลัมน์ Deptno
รูปแบบการเข้าถึงข้อมูลที่อ็อพติไมเซอร์เลือกใช้	Table Access Full	Index Range Scan
Cost (%CPU)	2	0
Time (ms)	00:00:37	00:00:01

โดยสรุปคำสั่งในการค้นหาข้อมูลทั้งแบบที่กำหนดและไม่กำหนดเงื่อนไขนั้น อ็อพติไมเซอร์จะเลือกใช้การเข้าถึงแบบ Full Table Scan ก่อน แต่หากตารางนั้นมีการกำหนดอินเด็กซ์บนคอลัมน์ที่มีเงื่อนไข WH-Cause ร่วมด้วย อ็อพติไมเซอร์จะใช้รูปแบบการเข้าถึงแบบ Index Lookup Scan แทน

4.2 ด้านการลิงค์ข้อมูลระหว่างตาราง

เป็นการวิเคราะห์ว่ารูปแบบการลิงค์ข้อมูลระหว่างตารางนั้น จะมีผลต่อการทำงานของอ็อพติไมเซอร์อย่างไร โดยการทดสอบนั้น จะใช้คำสั่งลิงค์ตารางระหว่าง EMP2 และ DEPT2 ผ่านคีย์นอก DEPTNO ดังแสดงในคำสั่งรูปแบบ (3)

```
select *
from emp2 e, dept2 d
where e.deptno = d.deptno
```
(3)

นอกจากนี้ยังเพิ่มการทดสอบลิงค์ข้อมูลระหว่างตารางของอ็อพติไมเซอร์ในแบบต่างๆ เช่น Nested Loop, Merge Join และ Hash Join ด้วยคำสั่ง “HINT” ซึ่งเป็นรูปแบบที่ให้ผู้ใช้แนะนำการประมวลผลคำสั่งสืบค้นแก่อ็อพติไมเซอร์ได้ ด้วยการระบุคีย์เวิร์ดดังกล่าว โดยเขียนตามหลังคำสั่ง SELECT และใช้อักขระ /*+ แล้วตามด้วยคำแนะนำ เช่น USE_NL, USE_MERGE หรือ USE_HASH และปิดด้ายด้วยอักขระ */ ตามลำดับ ดังแสดงในตัวอย่าง (4) เช่น การใช้ HINT “USE_NL” เพื่อระบุให้อ็อพติไมเซอร์ลิงค์ตาราง EMP2 และ DEPT2 แบบ Nested Loop หรือใช้ HINT “USE_HASH (table1, table2) เพื่อแนะนำให้อ็อพติไมเซอร์ลิงค์ข้อมูลด้วยตารางแบบแฮช เป็นต้น

```
explain plan for
select /*+ use_nl (emp2 e) */ e.*, d.*
from emp2 e, dept2 d
where e.deptno = d.deptno
```
(4)

ผลการทดสอบในตารางที่ 3 ซึ่งลิงค์ตารางทั้ง 3 รูปแบบด้วยคีย์เวิร์ด HINT นั้น พบว่าอ็อพติไมเซอร์จะใช้เวลาในการลิงค์ตารางแบบ Hash Join น้อยที่สุด ในขณะที่รูปแบบ Nested Loop จะใช้เวลามากที่สุด

ตารางที่ 3 เปรียบเทียบผลลัพธ์ใน Query Plan โดยลิงค์ตารางแบบ Nested Loop, Merge Join และ Hash Join

รูปแบบ	รูปแบบการลิงค์ตารางของอ็อพติไมเซอร์		
	Nested Loop	**Merge Join**	**Hash Join**
รูปแบบการลิงค์ตาราง	Nested Loop	Merge Join และ Sort Join	Hash Join
การใช้งาน CPU (%)	2	100	100
Time (ms)	999:59:59	07:56:39	07:52:38

ในทางกลับกัน หากมีการเพิ่มโอเปอร์ชันในการจัดเรียงข้อมูลด้วยคำสั่ง “*order by deptno*” ในคำสั่ง (4) พบว่าการลิงค์แบบ Merge Join จะให้ผลการทำงานที่เร็ว

และใช้ทรัพยากรในระบบที่น้อยที่สุด ในขณะที่อ๊อพติไมเซอร์จะหลีกเลี่ยงการลิงค์ข้อมูลระหว่างตารางในแบบ Hash Join และ Nested Loop เนื่องจากทั้งสองรูปแบบนั้นจะใช้เวลาและทรัพยากรในระบบส่วนหนึ่งเพื่อจัดเรียงข้อมูลในตารางก่อนซึ่งมีอยู่เป็นจำนวนมากดังสรุปผลในตารางที่ 4

ตารางที่ 4 เปรียบเทียบผลลัพธ์ใน Query Plan โดยลิงค์ตารางแบบ Nested Loop, Merge Join และ Hash Join และใช้เงื่อนไขจัดเรียงข้อมูล *"order by deptno"*

รูปแบบ	รูปแบบการลิงค์ตารางของอ็อพติไมเซอร์		
	Nested Loop	**Merge Join**	**Hash Join**
รูปแบบการลิงค์ตาราง (เรียงตามลำดับการทำงาน)	Nested Loop และ Index Scan	Merge Join และ Sort Join	Sort Order และ Hash Join
CPU Cost หน่วย (%CPU)	2	100	100
Time (ms)	999:59:59	07:56:39	999:59:59

นอกจากนี้ ยังเพิ่มการทดสอบเพื่อหาความสัมพันธ์ในการลิงค์ข้อมูลระหว่างตารางในรูปแบบต่าง ๆ กับปริมาณข้อมูลที่แตกต่างกันด้วย เพื่อวิเคราะห์ถึงประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ ทั้งในแง่การเปรียบเทียบด้านเวลา และการเปรียบเทียบด้านทรัพยากรที่อ๊อพติไมเซอร์ใช้ประมวลผลคำสั่งในระบบ ดังสรุปในตารางที่ 5-6

ตารางที่ 5 แสดงเวลา (Milliseconds) ที่ใช้ในการประมวลผลลิงค์ตารางเมื่อเทียบกับแถวข้อมูลที่เพิ่มขึ้น

ขนาดข้อมูล (แถว)	เวลาที่ใช้ในการประมวลผล (ms)		
	Nested Loop	**Merge Join**	**Hash Join**
1,000	7.50	12.32	7.50
10,000	72.40	24	16
100,000	10009	377.33	240.01
500,000	10,099.10	560	310
1,000,000	10,256.37	852.39	397.98

ตารางที่ 6 แสดงค่าการใช้งาน %CPU ที่คำนวณจากค่าของ I/O Operation และ CPU Cost ที่ใช้ลิงค์ตารางเมื่อเทียบกับจำนวนแถวข้อมูลที่เพิ่มมากขึ้น

ขนาดข้อมูล (แถว)	ร้อยละของการใช้งาน CPU (100%)		
	Nested Loop	**Merge Join**	**Hash Join**
1,000	0.0001	0.0001	0.0001
10,000	0.0027	0.0010	0.0003
100,000	2.16	3.57	0.82
500,000	78.53	3.82	2.47
1,000,000	96.41	5.06	2.84

ซึ่งผลการทดสอบในตาราง 5-6 พบว่าเมื่อปริมาณข้อมูลเพิ่มมากขึ้น การทำงานแบบ Nested Loop จะใช้เวลาและทรัพยากรมากที่สุด ในขณะที่แบบ Merge Join และ Hash Join ใช้เวลาและทรัพยากรโดยเฉลี่ยที่น้อยและไม่แตกต่างกัน

4.3 การทดสอบรูปแบบคำสั่งสืบค้นต่าง ๆ

การทดสอบรูปแบบคำสั่งสืบค้นต่าง ๆ [7], [14], [16] ในการเพิ่มประสิทธิภาพการประมวลผลคำสั่งเอสคิวแอลนั้น นอกจากรูปแบบการเข้าถึงข้อมูล การลิงค์ตารางแล้ว ส่วนหนึ่งยังขึ้นอยู่กับโครงสร้างและการออกแบบคำสั่งให้เหมาะสมด้วย และแม้ว่าหน้าที่ดังกล่าวจะเป็นของผู้ใช้โดยตรง แต่ว่าอ๊อพติไมเซอร์ก็มีเทคนิคหรือเงื่อนไขเบื้องต้นในการพิจารณาอยู่ ดังนั้นหากผู้ใช้ทราบถึงกลไกดังกล่าว ก็จะช่วยให้สามารถปรับแต่งคำสั่งได้เหมาะสม และหลีกเลี่ยงรูปแบบที่อาจจะก่อให้เกิดโหลดภายในอ๊อพติไมเซอร์ลงได้ โดยขอแยกเป็นกรณีต่าง ๆ ที่พบบ่อยหรือเป็นความผิดทั่วไปดังนี้คือ

4.3.1 การใช้คีย์หลักร่วมกับฟังก์ชัน โดยทั่วไปการกำหนดคอลัมน์ใด ๆ ให้เป็นคีย์หลักในตารางนั้น อ๊อพติไมเซอร์จะสร้างอินเด็กซ์สำหรับคอลัมน์นั้นให้โดยอัตโนมัติเพื่อการสืบค้นที่เร็วขึ้น แต่หากผู้ใช้มีการเปลี่ยนชนิดข้อมูลของคีย์หลักด้วยฟังก์ชันแปลงข้อมูลใด ๆ จะส่งผลให้อ๊อพติไมเซอร์ไม่สามารถใช้อินเด็กซ์ในคอลัมน์ได้ตามปกติ เพราะชนิดข้อมูลในอินเด็กซ์และค่าในคำสั่งไม่

สัมพันธ์กัน โดยกำหนดคำสั่ง (5) แทนการใช้งานฟังก์ชันแปลงข้อมูลร่วมกับคีย์หลัก empno และ คำสั่ง (6) แทนตัวอย่างการใช้งานคีย์หลักแบบปกติ (ไม่ใช้ร่วมกับฟังก์ชัน) ตามลำดับ

```
explain plan for
select count(*)
from   job_history jh, emp2 e
where  substr(to_char(e.empno),1) =          (5)
       substr(to_char(jh.employee_id),1)
```

```
explain plan for
select count(*)                              (6)
from  job_history jh, emp2 e
where  e.empno = jh.employee_id;
```

ตารางที่ 7 เปรียบเทียบผลลัพธ์ที่เกิดขึ้นใน Query Plan ของการใช้ฟังก์ชันในคีย์หลักของคำสั่ง (5) และ (6)

รูปแบบ	คำสั่งที่ (5) ใช้ฟังก์ชันในคีย์หลัก	คำสั่งที่ (6) ไม่ใช้ฟังก์ชันในคีย์หลัก
การทำงานของอ็อพติไมเซอร์	Full Table Scan	Index Scan
ร้อยละการใช้งาน CPU (%)	2	2
Time (ms)	00:00:57	00:00:21

จากตารางที่ 7 คำสั่ง (5) มีการใช้ฟังก์ชันแปลงชนิดข้อมูลใน empno ซึ่งเป็นคีย์หลัก ซึ่งผลดังกล่าวทำให้อ็อพติไมเซอร์จะค้นหาข้อมูลแบบ Full Table Scan โดยที่ไม่ใช้อินเด็กซ์ในคีย์หลัก เพราะชนิดข้อมูลใน empno ซึ่งเป็นอินเด็กซ์ได้ถูกเปลี่ยนค่าไป เนื่องจากมีการใช้ฟังก์ชันแปลงข้อมูล ซึ่งจะทำให้เวลาที่ใช้นั้นมากกว่าการเข้าถึงข้อมูลด้วยวิธีการใช้ Index Scan ในคำสั่ง (6)

4.3.2 การเปรียบเทียบชนิดข้อมูลที่ไม่สอดคล้องกัน เนื่องจากในการกำหนดหรือเปรียบเทียบเงื่อนไขที่มีชนิดข้อมูลไม่ตรงกันนั้น อ็อพติไมเซอร์จะแปลงข้อมูลให้มีชนิดที่ตรงและสัมพันธ์กันให้กับผู้ใช้เองโดยอัตโนมัติ (Implicit Converting) โดยที่ไม่แจ้งข้อผิดพลาด (Compile Error) ใดๆ เมื่อรันคำสั่ง แต่การทำงานดังกล่าวจะทำให้อ็อพติไมเซอร์ใช้เวลาตรวจสอบและแปลงชนิดข้อมูลที่มากขึ้น จากคำสั่ง (7) แทนการเปรียบเทียบชนิดข้อมูลที่ไม่ตรงกันคือชนิดสตริง (order_id_char) และตัวเลข (1205) ในขณะที่คำสั่ง (8) เป็นการกำหนดเงื่อนไขเพื่อเปรียบเทียบชนิดข้อมูลที่ตรงและสัมพันธ์กัน

```
explain plan for
SELECT count(*) FROM orders                  (7)
WHERE order_id_char = 1205;
```

```
explain plan for
select count(*) from orders                  (8)
where order_id_char = '1205';
```

ตารางที่ 8 เปรียบเทียบผลลัพธ์ที่เกิดขึ้นใน Query Plan เมื่อมีการกำหนดชนิดข้อมูลในคำสั่ง (7) และ (8)

รูปแบบ	คำสั่งที่ (7) ระบุเงื่อนไขเพื่อเปรียบเทียบข้อมูลที่ไม่ตรงกัน	คำสั่งที่ (8) ระบุเงื่อนไขเพื่อเปรียบเทียบข้อมูลที่ตรงกัน
การทำงานของอ็อพติไมเซอร์	Full Table Scan	Index Scan
ร้อยละการใช้งาน CPU (%)	1	0
Time (ms)	00:01:47	00:00:01

จากตารางที่ 8 กำหนดให้คำสั่งที่ (7) เปรียบเทียบชนิดข้อมูลต่างประเภทกันคือ varchar2 และ number ซึ่งในกรณีนี้อ็อพติไมเซอร์จะแปลงชนิดข้อมูลในคอลัมน์นี้ให้เป็นตัวเลขโดยอัตโนมัติ ทำให้ผู้ใช้สามารถรันคำสั่งดังกล่าวได้ตามปกติโดยไม่เกิดข้อผิดพลาดใดๆ แต่การค้นหาที่เกิดขึ้นนั้นจะทำให้อ็อพติไมเซอร์ไม่สามารถค้นหาโดยใช้อินเด็กซ์ได้ เนื่องจากคีย์คอลัมน์ได้ถูกเปลี่ยนชนิดข้อมูลจาก varchar2 ให้เป็น number ด้วยฟังก์ชัน TO_NUMBER แล้ว ซึ่งทำให้อ็อพติไมเซอร์จะเข้าถึงข้อมูลแบบสแกนทั้งหมด และใช้เวลาหรือทรัพยากรที่มากตามลำดับด้วย ในขณะที่หากเปรียบเทียบบนชนิดข้อมูลที่สัมพันธ์กันดังคำสั่งที่ (8)

จะทำให้อ๊อพติไมเซอร์ไม่ต้องใช้เวลาแปลงข้อมูล และยังเข้าถึงข้อมูลด้วยวิธีอินเด็กซ์ได้โดยตรง ซึ่งก็จะทำให้ใช้เวลาและทรัพยากรในระบบน้อยลงไปด้วย

ดังนั้นจึงสรุปได้ว่าผู้ใช้ควรสร้างเงื่อนไขซึ่งมีชนิดข้อมูลที่สัมพันธ์กันในรูปแบบ Explicit Function มากกว่าการสร้างเงื่อนไขในแบบ Implicit Functions ที่ต้องรอให้อ๊อพติไมเซอร์ใช้เวลาและทรัพยากรส่วนหนึ่งตรวจสอบและแปลงข้อมูลให้ก่อนประมวลผล เพราะหากปริมาณข้อมูลมีจำนวนมาก ก็จะทำให้ระบบช้า หรือหากคอลัมน์นั้นทำหน้าที่เป็นคีย์หลักหรือเป็นอินเด็กซ์ในตารางก็จะทำให้คอลัมน์นั้นไม่สามารถเข้าถึงข้อมูลด้วยวิธี Index Scan ได้ตามปกติ ซึ่งก็จะส่งผลต่อเวลาและทรัพยากรที่ต้องใช้มากขึ้นตามลำดับ

4.3.3 การทำงานกับประโยคสืบค้นย่อยแบบวนซ้ำทุกแถว (Correlated Sub Query) ซึ่งรูปแบบดังกล่าวจะทำให้อ๊อพติไมเซอร์ต้องประมวลผลคำสั่งสืบค้นย่อย (Sub Query) ในทุกรอบหรือทุกแถวข้อมูลในตารางนั้น ซึ่งเป็นการสิ้นเปลืองเวลาและทรัพยากรระบบเป็นอย่างมาก

โดยกำหนดให้คำสั่ง (9) แทน Correlated Sub Query ซึ่งพบว่าใน WH-Clause มีคำสั่งสืบค้นย่อยซ้อนอยู่ ในขณะที่คำสั่ง (10) กำหนดคล้ายกัน แต่เปลี่ยนลำดับของคำสั่งสืบค้นย่อยไปที่ตำแหน่ง FROM จากนั้นตรวจสอบผลการสร้างอ๊อพติไมเซอร์

```
explain plan for
select count(*)                                    (9)
from   products p
where  prod_list_price  <  1.15 *
       (select avg(unit_cost)
       from  costs c
       where c.prod_id = p.prod_id  )

explain plan for
select count(*)
from  products p,
      (select prod_id, avg(unit_cost ) ac          (10)
      from costs group by prod_id ) c
where  p.prod_id = c.prod_id and
       p.prod_list_price < 1.15 * c.ac;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |    26 |   181K  (2)| 00:36:16 |       |       |
|   1 |  SORT AGGREGATE        |          |     1 |    26 |            |          |       |       |
|*  2 |   HASH JOIN            |          |  3904 |   198K|   181K  (2)| 00:36:16 |       |       |
|   3 |    TABLE ACCESS FULL   | PRODUCTS | 78071 |  1982K|   992   (1)| 00:00:12 |       |       |
|   4 |    SORT AGGREGATE      |          |     1 |     9 |            |          |       |       |
|   5 |     PARTITION RANGE ALL|          |  1140 | 10260 |    74   (2)| 00:00:01 |     1 |    28 |
|*  6 |      TABLE ACCESS FULL | COSTS    |  1140 | 10260 |    74   (2)| 00:00:01 |     1 |    28 |
-------------------------------------------------------------------------------------------

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

  2 - filter("PROD_LIST_PRICE"<1.15* (SELECT AVG("UNIT_COST") FROM "COSTS" "C" WHERE
              "C"."PROD_ID"=:B1))
  6 - filter("C"."PROD_ID"=:B1)
```

รูปที่ 9 Query Plan เมื่อรันคำสั่ง Correlated Sub Query

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |    52 |  1071   (1)| 00:00:13 |       |       |
|   1 |  SORT AGGREGATE        |          |     1 |    52 |            |          |       |       |
|*  2 |   HASH JOIN            |          |  3904 |   198K|  1071   (1)| 00:00:13 |       |       |
|   3 |    VIEW                |          |    72 |  1872 |    78   (7)| 00:00:01 |       |       |
|   4 |     HASH GROUP BY      |          |    72 |   648 |    78   (7)| 00:00:01 |       |       |
|   5 |      PARTITION RANGE ALL|         | 82112 |   721K|    74   (2)| 00:00:01 |     1 |    28 |
|   6 |       TABLE ACCESS FULL| COSTS    | 82112 |   721K|    74   (2)| 00:00:01 |     1 |    28 |
|   7 |    TABLE ACCESS FULL   | PRODUCTS | 78071 |  1982K|   992   (1)| 00:00:12 |       |       |
-------------------------------------------------------------------------------------------

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

  2 - access("P"."PROD_ID"="C"."PROD_ID")
      filter("P"."PROD_LIST_PRICE"<1.15*"C"."AC")
```

รูปที่ 10 Query Plan เมื่อปรับคำสั่งแบบ Non-Correlated

จากรูปที่ 9 พบว่าผลการสร้างแผนการสืบค้น โดยรันด้วยคำสั่ง (9) จะมีการใช้เวลาและทรัพยากรที่มากกว่า (10) โดยจะเริ่มทำงานที่ Id 6, 5, 3, 4, 2 และ 1 ซึ่งทำให้อ๊อพติไมเซอร์จะรันคำสั่งสืบค้นย่อยเพื่อหา Aggregation Functions ในทุกรอบของการประมวลผลแบบสแกนทั้งหมด ซึ่งก็จะส่งผลให้การใช้ทรัพยากรใน Id 3-4 มีจำนวนที่มากกว่าปกติ และส่งผลโดยรวมต่อการทำงานทั้งหมดซึ่งมีค่ามากตามไปด้วย

ในขณะที่รูปที่ 10 จะเริ่มการทำงานที่ Id 6, 5 และ 4 ตามลำดับ ซึ่งการรันดังกล่าวเป็นการหา Aggregation Function “AVG” ก่อน เพราะมีการสแกนข้อมูลทั้งหมดและใช้ฟังก์ชัน Group By ใน Id 4 เพื่อคำนวณผลลัพธ์ จากนั้นจึงเก็บค่าไว้ใน View (Id 3) ก่อนทำการสแกนข้อมูลทั้งหมดใน Id 7 ซึ่งจะทำให้มีการรันคำสั่ง Aggregation Functions เพียงครั้งเดียว เพราะอ๊อพติไมเซอร์จะใช้ View

เก็บผลลัพธ์ของฟังก์ชันในการตรวจสอบและค้นหาข้อมูลที่สอดคล้องกับเงื่อนไขตามที่กำหนดในคำสั่งต่อไป

ดังนั้นผู้ใช้ควรสร้างคำสั่งประเภท Non-Correlated Query ที่ใช้หลักการจัดเรียงลำดับโอเปอเรชัน และรูปแบบคำสั่งเข้ามาช่วยลดจำนวนการประมวลผลข้อมูลของคำสั่งสืบค้นได้ดีกว่าคำสั่งประเภท Correlated Query

4.3.4 การใช้ UNION และ UNION ALL โดยการใช้รูปแบบ Union นั้นจะให้ผลการทำงานที่ช้ากว่าแบบ Union All เนื่องจากคำสั่ง Union อ๊อพติไมเซอร์จะตัดคีย์ที่มีชนิดข้อมูลและมีคุณสมบัติร่วมกันออกก่อน พร้อมจัดเรียงลำดับข้อมูลให้ด้วย ซึ่งการทำงานดังกล่าว จะทำให้ระบบประมวลผลได้ช้าลง

```
explain plan for
select count(*)
from (select deptno from emp2 union          (11)
      select deptno from dept2);

explain plan for
select count(*)
from (select deptno from emp2 union all      (12)
      select deptno from dept2);
```

ตารางที่ 9 เปรียบเทียบผลลัพธ์ใน Query Plan ของการใช้ UNION และ UNION ALL ในคำสั่ง (11) - (12)

รูปแบบ	คำสั่งที่ (11) ใช้ UNION	คำสั่งที่ (12) ใช้ UNION ALL
การทำงานของอ๊อพติไมเซอร์	Sort และ UNION ALL	UNION ALL
ร้อยละการใช้งาน CPU (%)	34	1
Time (ms)	00:02:22	00:00:49

จากตารางที่ 9 ในคำสั่งที่ 11 ซึ่งใช้รูปแบบ UNION พบว่าอ๊อพติไมเซอร์ใช้เวลาและทรัพยากรส่วนหนึ่งในการจัดเรียง (Sort) ข้อมูลด้วย ในขณะที่การใช้งานรูปแบบ UNION ALL ในคำสั่ง 12 นั้น จะไม่พบโอเปอเรชันดังกล่าวในแผนการสืบค้นเลย ซึ่งก็จะช่วยลดเวลาในการประมวลผลคำสั่งลงได้

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  3040   (1)| 00:00:37 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| EMP2 |  1835K|  7168K|  3040   (1)| 00:00:37 |
-------------------------------------------------------------------------------
```

รูปที่ 11 Query Plan ในการใช้ CASE

4.3.5 การใช้ CASE เพื่อควบคุมการทำงาน ข้อดีคือช่วยลดจำนวนการประมวลผลในหลายๆ คำสั่งได้ โดยใช้การรวมคำสั่งและรูปแบบเงื่อนไขที่สัมพันธ์กันไว้ในโครงสร้างเดียวกันด้วยตัวดำเนินการ CASE ซึ่งจะช่วยให้การประมวลผลหลายคำสั่งหรือหลายเงื่อนไขทำได้ในคราวเดียวกัน ทั้งยังช่วยลดเวลาการติดต่อระบบและการใช้งานทรัพยากรลงด้วย

```
from emp2;
select count (case when sal < 2000
      then 1 else null end) count1,          (13)
       count (case when sal between 2001
       and 4000 then 1 else null end) count2,
       count (case when sal > 4000
       then 1 else null end) count3
explain plan for
```

จากรูปที่ 11 พบว่าผลการรันประโยคเงื่อนไขต่างๆ ด้วยตัวดำเนินการ CASE นั้น สามารถประมวลผลข้อมูลได้ในคราวเดียวกัน ในขณะที่หากรันคำสั่งแยกกันในแต่ละ CASE นั้น จะใช้เวลาทำงานที่มากกว่า เนื่องจากต้องรันคำสั่ง (ครั้ง) ตามกรณีที่กำหนดในเงื่อนไขทั้งหมด (ประมาณ 37 ms * 3 กรณี)

5. อภิปรายผลและสรุป

จากผลการทดสอบ สามารถสรุปผลประเมินประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ที่มีต่อคำสั่งเอสคิวแอลได้ดังนี้คือ

1. อ๊อพติไมเซอร์จะเข้าถึงข้อมูลทั้งหมดในรูปแบบ

Full Table Scan เนื่องจากจะใช้เวลาอ่านบล็อกข้อมูลในดิสก์เพียงครั้งเดียว ซึ่งช่วยลดการติดต่อกับ I/O Disk ได้ ในขณะที่หากมีการระบุเงื่อนไขในคำสั่ง WH-Clause นั้น อ็อพติไมเซอร์จะค้นหาด้วยวิธี Index Scan (กรณีที่มีอินเด็กซ์ในคอลัมน์) ซึ่งจะใช้เวลาที่รวดเร็วกว่าแบบ Full Table Scan (กรณีที่ไม่มีอินเด็กซ์ในคอลัมน์) แต่อย่างไรก็ตามการสร้างอินเด็กซ์บนคอลัมน์ใดๆ นั้น ควรคำนึงถึงปัจจัยแวดล้อมอื่นๆ ร่วมด้วย เช่น ความถี่ในการเข้าถึงข้อมูล ประเภทการใช้งาน หรือปริมาณข้อมูลในระบบ เนื่องจากหากมีโอเปอเรชันเพิ่ม (Insert) ลบ (Delete) หรือแก้ไขข้อมูล (Update) เกิดขึ้น ระบบจะต้องใช้เวลาและทรัพยากรส่วนหนึ่งเพื่อปรับปรุงข้อมูลในอินเด็กซ์นั้นด้วย [14]

2. ในกรณีที่มีการสืบค้นข้อมูลในตารางขนาดใหญ่ ผู้ใช้ควรกำหนดให้อ๊อพติไมเซอร์ลิงค์ข้อมูลด้วยการกำหนดอ็อพชันHash Join ผ่านคำสั่ง /* Hint "use_hash" */ เพราะจะช่วยให้ระบบประมวลผลรวดเร็วยิ่งขึ้น เนื่องจากการลิงค์ตารางแบบ Hash Join นั้นจะมีประสิทธิภาพการทำงานที่ดีกว่าแบบอื่นๆ เนื่องจากมีการใช้ตารางแฮชเข้ามาช่วยในการหาข้อมูลร่วมด้วย

3. การเพิ่มประสิทธิภาพการประมวลผลคำสั่งเอสคิวแอลนั้น ยังขึ้นอยู่กับโครงสร้างและรูปแบบคำสั่งด้วย เช่น ควรหลีกเลี่ยงการใช้คีย์หลักร่วมกับฟังก์ชัน เพราะจะทำให้การค้นหาข้อมูลทำได้ช้ากว่าปกติ เนื่องจากอ๊อพติไมเซอร์จะค้นหาด้วยวิธี Full Table Scan แทนการค้นหาโดยใช้อินเด็กซ์ในแบบ Index Scan หรือผู้ใช้ควรหลีกเลี่ยงคำสั่งที่มีการใช้งานแบบ Correlated Sub Query UNION และ Implicit Casting เพราะจะทำให้อ๊อพติไมเซอร์ใช้ทรัพยากรและเวลาประมวลผลนานขึ้น

4. ในอนาคตอาจมีการเปรียบเทียบการทำงานของอ๊อพติไมเซอร์ในดีบีเอ็มเอสชนิดต่างๆ ซึ่งจะช่วยให้ผู้ใช้ทราบถึงประสิทธิภาพหรือข้อจำกัดของแต่ละระบบได้ ซึ่งก็จะเป็นประโยชน์ในแง่การเลือกใช้ดีบีเอ็มเอสให้เหมาะสมต่อไป หรือจะนำหลักการเพิ่มประสิทธิภาพการทำงานของอ๊อพติไมเซอร์ที่มีต่อคำสั่งเอสคิวแอลแบบซีเล็กชัน มาต่อยอดและนำเสนอกรอบแนวคิดเพื่อพัฒนาระบบแนะนำคำสั่งเอสคิวแอลที่เหมาะสมแบบอัตโนมัติดังที่ระบุในงานวิจัย [5], [16]

เอกสารอ้างอิง

[1] Oracle Corporation (2007). *"Oracle Database Performance Tuning Guide, 11g Release 1 (11.1)"* [Online]. Available: http://download.oracle.com/docs/cd/B28359_01/server.111/b28275/toc.htm

[2] R. Mukkamala and R. Lin, "*Improving Database Performance through Query Standardization*" in *Proceedings. Energy and Information Technologies in the Southeast*, vol. 3 pp. 1321-1325, 1989.

[3] V.K.S. Nerella et al., "*Exploring Query Optimization in Programming Codes by Reducing Run-Time Execution*" in *Computer Software and Applications Conference*, pp. 407-412. 2010.

[4] P. Belknap et at., "*Self-Tuning for SQL Performance in Oracle DB 11g*," *The 25th International Conference on Data Engineering*, pp. 1321-1324, 2009.

[5] Li Dandan et al., "*SQL Query Optimization Methods of Relational Database System*," in *the 2nd International Conference on Computer Engineering and Applications*., vol.1 pp. 557-560, 2010.

[6] P. Khaitan et al., "*Improved query plans for unnesting nested SQL queries*," in *The 9th International Conference on Computer Science and its Applications*, pp.1-6. 2009.

[7] M. Jarke and J. Koch, "*Query Optimization in DB," Computing Survey*, vol. 16, no.3, pp 111–152, June 1984.

[8] A. Silberschatz, H.F. Korth and S. Sudarshan: *Database System Concept*, The McGraw Hill Companies Inc. 2006, pp. 569-602.

[9] Oracle Corporation (2010). *The Query Optimizer*. [Online]. Available: http://download.oracle.

com/docs/ cd/B28359_01/server.111/b28274 /optimops.htm.

[10] Oracle Corporation (2010). *Introduction to the Optimizer*. [Online]. Available: http://docs. oracle.com/cd/B10550_01/server.920/a96533 /optimops.htm

[11] Richard Niemiec, *Oracle Database 10g Performance Tuning Tips & Techniques*, The McGraw Hill Companies Inc. 2007, pp. 254-250.

[12] B. Peasland, *Understanding Oracle cost-based optimizer (CBO) and ruled-based optimizer (RBO)*, SGT, Inc, 2008.

[13] Oracle Corporation (2010). *Using Explain Plan.* [Online]. Available: http://download.oracle. com/docs/ cd/B10500_01/server.920/a96533/ ex_plan.htm.

[14] K. Dias et al., "*Automatic Performance Diagnosis and Tuning in Oracle*", in *the Proceedings of the CIDR Conference*, 2005.

[15] Reuter (2010). *Oracle Is #1 in the RDBMS Market for 2010* [Online]. Available: http://in.reuters. com/article/ 2011/04/07/idUS174473+07 -Apr-2011+MW20110407

[16] B. Dageville et al., "*Automatic SQL Tuning in Oracle 10g*," in *the Proceeding of the 30th VLDB, Canada*, pp. 1098–1109, 2004.