การประมวลผลชุดคำสั่ง SQL โดยอ๊อพติไมเซอร์ของระบบจัดการฐานข้อมูล

สำเริง จันทร์ลมูล

ผู้เชี่ยวชาญระบบจัดการฐานข้อมูล วิทยากร และที่ปรึกษาอิสระ

Emails: samreung@truemail.co.th

บทคัดย่อ

โดยทั่วไปนักพัฒนาโปรแกรมประยุกต์และผู้บริหารระบบฐานข้อมูลควรมีความรู้ความเข้าใจขั้นตอนการประมวลผลชุดคำสั่ง SQL (Structured Query Language) บ้างพอสมควรแล้วจึงเขียนชุดคำสั่งให้สอดคล้องกับการประมวลผลของชุดคำสั่งแต่ละประเภท ซึ่งจะทำให้ระบบจัดการฐานข้อมูลไม่ต้องเสียทรัพยากรและเวลามากเกินความจำเป็น บทความนี้ให้ความรู้และความเข้าใจขั้นตอนการประมวลผลชุดคำสั่งโดยอ๊อพติไมเซอร์ของระบบจัดการฐานข้อมูลเชิงสัมพันธ์ และจะแสดงตัวอย่างชุดคำสั่งหลายชุดที่ให้ผลลัพธ์เหมือนกันแต่มีการประมวลผลต่างกันซึ่งจะให้ประสิทธิภาพต่างกันด้วย การเลือกชุดคำสั่งที่มีประสิทธิภาพดีขึ้น การรวบรวมและวิเคราะห์ค่าสถิติเชิงประสิทธิภาพ (Performance Statistics) ให้ระบบจัดการฐานข้อมูลนำไปใช้ได้อย่างมีประสิทธิภาพ รวมทั้งคำแนะนำสำหรับนักพัฒนาโปรแกรมประยุกต์และผู้ดูแลระบบฐานข้อมูลให้เขียนชุดคำสั่งให้สอดคล้องกับการทำงานหรือสิ่งที่อ๊อพติไมเซอร์ต้องทำเพื่อลดขั้นตอนการทำงานของอ๊อพติไมเซอร์ และช่วยให้อ๊อพติไมเซอร์ทำงานตามต้องการในกรณีที่มันอาจทำงานได้ไม่ดีหรือในสถานการณ์จำเพาะหรือพิเศษซึ่งไม่สามารถใช้วิธีการทั่วไปให้มีประสิทธิภาพได้

คำสำคัญ – optimizer; performance statistics; SQL processing; execution plan; parsing; fetching

Abstract

In general, application software developers and database system administrators should have some knowledge and understanding on steps in SQL commands processing. Such knowledge, when used for writing a command set, can avoid unnecessary excessive resource usage. This article explains the steps in SQL command processing by an optimizer inside a relational database management system. In addition, the article gives examples of how different command sets with the same query result differ in terms of performance. By taking into account how the optimizer operates, guidelines on writing an efficient command set are provided. Such guidelines are useful for improving processing performance in situations where a general approach for performance optimization is not applicable.

Keywords – optimizer; performance statistics; SQL processing; execution plan; parsing; fetching

1. บทนำ

บทความนี้ให้ความรู้และความเข้าใจขั้นตอนการประมวลผลชุดคำสั่ง โดยอ๊อพติไมเซอร์ของระบบจัดการฐานข้อมูลเชิงสัมพันธ์ สำหรับนักพัฒนาโปรแกรมประยุกต์และผู้บริหารระบบฐานข้อมูลให้เข้าใจบทบาทความสำคัญ และความสัมพันธ์ระหว่างอ๊อพติไมเซอร์กับชุดคำสั่ง เพื่อให้เขียนชุดคำสั่งได้สอดคล้องกับการทำงานของอ๊อพติไมเซอร์ อันจะทำให้ไม่ต้องเสียทรัพยากรและเสียเวลาในการประมวลผลมากเกินความจำเป็น และบทความนี้จะแสดงให้เห็นถึงชุดคำสั่งหลายชุดที่ให้ผลลัพธ์เหมือนกัน แต่อาจมีประสิทธิภาพต่างกัน การเลือกชุดคำสั่งที่มีประสิทธิภาพดีขึ้น การรวบรวมและวิเคราะห์ค่าสถิติเชิงประสิทธิภาพให้ระบบจัดการฐานข้อมูลนำไปใช้ได้อย่างมีประสิทธิภาพ เนื่องจากอ๊อพติไมเซอร์อาจเลือกใช้วิธีการทำงานที่มีประสิทธิภาพต่ำในกรณีที่ไม่มีสถิติเชิงประสิทธิภาพหรือมีแต่ไม่ดีพอหรือเก่าไปหรือโครงสร้างข้อมูลไม่ดีพอหรือระบบจัดการฐานข้อมูลไม่สามารถรู้ล่วงหน้าในสถานการณ์พิเศษ

กลไกการทำงานของอ๊อพติไมเซอร์ และตัวอย่างชุดคำสั่งในเอกสารนี้มีเหมือนกันในระบบจัดการฐานข้อมูลออราเคิล (Oracle) และ ไอบีเอ็ม ดีบีทู (IBM DB2) [1-4] กลไกและชุดคำสั่งเหล่านี้ยังพบได้ในระบบจัดการฐานข้อมูลอื่นๆเป็นส่วนใหญ่

2. สถาปัตยกรรมของระบบจัดการฐานข้อมูลเชิงสัมพันธ์เบื้องต้น

ระบบจัดการฐานข้อมูลเชิงสัมพันธ์มีองค์ประกอบที่สำคัญ 2 ส่วน คือกลุ่มโปรเซสเบื้องหลังกับหน่วยความจำที่ใช้ร่วมกัน (Background Processes and Shared Memory) และกลุ่มแฟ้มข้อมูลที่จัดเก็บข้อมูลกับสารสนเทศที่ใช้ควบคุมการใช้งานระบบฐานข้อมูล (Database Files) [5-6]

2.1. กลุ่มโปรเซสเบื้องหลังกับหน่วยความจำที่ใช้ร่วมกัน

โปรเซสเบื้องหลังอำนวยความสะดวกในการใช้งานโดยทำหน้าที่บันทึกชุดคำสั่งที่ได้รับลงบนหน่วยความจำและแฟ้มรายการเปลี่ยนแปลง อ่านข้อมูลจากแฟ้มข้อมูลที่ต้องการเข้าสู่ความจำ บันทึกข้อมูลใหม่ที่เปลี่ยนแปลงลงบนความจำที่ใช้ร่วมกันแล้วบันทึกลงแฟ้มข้อมูล นอกจากนี้ยังควบคุมไม่ให้มีความขัดแย้งกันในกรณีที่ต้องใช้ข้อมูลตัวเดียวกัน (Concurrency Control)

2.2. กลุ่มแฟ้มข้อมูลที่ใช้จัดเก็บข้อมูลและสารสนเทศที่ใช้ควบคุมการใช้งานระบบฐานข้อมูล

กลุ่มแฟ้มข้อมูลนี้ยังแบ่งออกเป็นอีก 3 ประเภท คือ ประเภทแรกเป็นแฟ้มข้อมูลที่เก็บโครงสร้างทางกายภาพและทางตรรกะของฐานข้อมูล (Control Files) ประเภทที่สองเป็นแฟ้มที่ใช้จัดเก็บข้อมูลของระบบงานและสารสนเทศที่ใช้ควบคุมระบบฐานข้อมูล (Data Files) ประเภทที่สามเป็นแฟ้มข้อมูลที่จัดเก็บรายการเปลี่ยนแปลง (Redo Log Files) ในฐานข้อมูลเพื่อใช้กู้คืนรายการแก้ไขข้อมูลและกู้คืนระบบฐานข้อมูล

ตารางที่ 1. ตาราง DEPT

DNO	DNAME
10	Management
20	Marketing
30	Sales
40	Production

ตาราง DEPT มี 2 คอลัมน์ คือ คอลัมน์ dno บันทึกหมายเลขฝ่ายขององค์กรและเป็นคีย์หลัก (Primary Key) และคอลัมน์ dname บันทึกชื่อของฝ่ายต่างๆ

ตารางที่ 2. ตาราง EMP

ENO	ENAME	JOB	DNO
1	Anne	CEO	10
2	Peter	Manager	10
3	Smith	Manager	20
4	Alex	Manager	30
5	Ben	Clerk	40
6	Scott	Manager	40
7	Clark	Worker	40
8	Frank	Clerk	30
9	Peter	Worker	40
10	Robert	Worker	40

ตาราง EMP บันทึกข้อมูลเกี่ยวกับพนักงาน ประกอบด้วยคอลัมน์ eno บันทึกหมายเลขประจำตัวพนักงานและเป็นคีย์หลัก คอลัมน์ ename บันทึกชื่อพนักงาน คอลัมน์ job บันทึกตำแหน่งงาน และคอลัมน์ dno บันทึกหมายเลขฝ่ายที่พนักงานสังกัดและเป็นคีย์นอก (Foreign Key) ซึ่งเชื่อมโยงความสัมพันธ์ไปยังตาราง DEPT

3. อ๊อพติไมเซอร์

อ๊อพติไมเซอร์เป็นส่วนหนึ่งของระบบจัดการฐานข้อมูลซึ่งมีโหมดในการทำงานให้เลือกโหมดใดโหมดหนึ่งใน 2 โหมด คือ Rule-Based ซึ่งทำตามแผนการสืบค้น (Execution Plan) ที่เป็นไปตามกฎที่วางไว้ล่วงหน้า กับ Cost-Based ซึ่งใช้สถิติเชิงประสิทธิภาพในการเลือกแผนการสืบค้น โดยการสร้างแผนการสืบค้นที่เป็นไปได้หลายแผน คำนวณค่าใช้จ่ายของแต่ละแผนในรูปของเวลาที่ซีพียู (CPU) ใช้ในการประมวลผล ปริมาณหน่วยความจำ (RAM) ที่ต้องใช้ จำนวนครั้งและขนาดของข้อมูลที่ต้องอ่านหรือบันทึกบนดิสก์ (Disk Access) ในการคำนวณค่าใช้จ่ายนี้จำเป็นต้องใช้สถิติเชิงประสิทธิภาพในการใช้งานฐานข้อมูลที่ผ่านมา จากนั้นเลือกแผนการสืบค้นที่มีค่าใช้จ่ายต่ำสุดไปดำเนินการประมวลผล โดยทั่วไปโหมด Cost-Based ทำงานมีประสิทธิภาพดีกว่าโหมด Rule-Based แต่ถ้าไม่มีสถิติเชิงประสิทธิภาพก็จะกลับไปเป็นโหมด Rule-Based

3.1. การรวบรวมและวิเคราะห์สถิติเชิงประสิทธิภาพ

ค่าสถิติเชิงประสิทธิภาพคือข้อมูลที่เกี่ยวกับตารางข้อมูลหรืออินเด็กซ์ ค่าสถิติเชิงประสิทธิภาพของแต่ละตารางประกอบด้วยจำนวนแถวทั้งหมด จำนวนบล็อกหรือเพจที่มีข้อมูลและไม่มีข้อมูล ความยาวเฉลี่ยของแถวข้อมูล เป็นต้น ส่วนค่าสถิติเชิงประสิทธิภาพของอินเด็กซ์ประกอบด้วย จำนวนค่าของอินเด็กซ์ที่แตกต่างกันทั้งหมด จำนวนบล็อกหรือเพจที่ใช้จัดเก็บอินเด็กซ์ จำนวนชั้นหรือความสูงของอินเด็กซ์ ชนิดของอินเด็กซ์ เป็นต้น

อ๊อพติไมเซอร์จะใช้ค่าของสถิติเชิงประสิทธิภาพในการคำนวณค่าใช้จ่ายที่จะต้องใช้ในการประมวลผลทุกชุดคำสั่งที่ให้ผลของการประมวลผลเหมือนกัน จากนั้นจะเลือกชุดคำสั่งที่มีผลการคำนวณค่าใช้จ่ายที่จะใช้ต่ำที่สุดไปประมวลผลต่อไป

วิธีการรวบรวมและวิเคราะห์สถิติเชิงประสิทธิภาพมี 2 วิธีคือ ใช้ชุดคำสั่ง และการใช้โมดูลหรือแพ็คเก็ตที่ระบบจัดการฐานข้อมูลมีให้ ตัวอย่างการรวบรวมและวิเคราะห์สถิติการใช้ตาราง emp และอินเด็กซ์ emp_dno_idx ตามลำดับมีดังนี้

```
ANALYZE TABLE emp ESTIMATE
STATISTICS;

ANALYZE INDEX emp_dno_idx COMPUTE
STATISTICS;
```

3.2. Selectivity

เป็นค่าที่ Cost-based Optimizer ใช้ในการประเมินว่าจะเลือกแผนดำเนินการใดที่ให้ได้ประสิทธิภาพที่ดี

โดยมีความสัมพันธ์กับข้อมูลดังนี้ [1,2]

Selectivity = 1/จำนวนค่าของข้อมูลที่แตกต่างกัน

เช่น ค่า Selectivity ของคอลัมน์ ename ในตาราง EMP คือ 1/9 ไม่ใช่ 1/10 เพราะมีชื่อ Peter ซ้ำกัน 2 ชื่อ ทำให้จำนวนค่าที่แตกต่างกันของชื่อมีจำนวน 9 ชื่อ ตัวอย่างของชุดคำสั่งที่ใช้หาค่า Selectivity คือ

```
1/(SELECT DISTINCT(ename) FROM
emp);
```

โดยทั่วไประบบจัดการฐานข้อมูลมักจะใช้อินเด็กซ์ถ้าค่าของ Selectivity ไม่เกินขนาดหนึ่ง เช่น Selectivity < 0.04 [3]

ตัวอย่างเช่น ถ้าสร้างอินเด็กซ์บนคอลัมน์ ename แล้วมีชุดคำสั่งเรียกดูข้อมูลเป็น

```
SELECT *
FROM emp
WHERE ename = 'Robert';
```

เมื่ออ๊อพติไมเซอร์ได้รับชุดคำสั่งข้างต้นแล้วมันจะคำนวณค่า Selectivity ของคอลัมน์ ename กรณีนี้คือ 1/9 ซึ่ง > 0.04 ระบบจัดการฐานข้อมูลอาจเลือกที่จะไม่ใช้อินเด็กซ์แต่จะอ่านข้อมูลทั้งตารางที่เรียกว่า Full Table Scan และถ้าเป็นเช่นนี้จริงแล้วการลบอินเด็กซ์ที่สร้างบนคอลัมน์นี้ทิ้งไปอาจทำให้ประสิทธิภาพโดยรวมของระบบดีขึ้นได้

3.3. Cardinality

เป็นจำนวนแถวที่อ๊อพติไมเซอร์คำนวณล่วงหน้าว่าจะได้ข้อมูลประมาณกี่แถว จากนั้นจึงจะพิจารณาว่าจะใช้วิธีการใดในการอ่านข้อมูลนี้ เช่น อ่านข้อมูลทั้งหมด หรืออ่านบางบล็อกหรือบางเพจ ตัวอย่างหากต้องการข้อมูลบางส่วน เช่นประมาณ 5% แต่ถ้าผลการคำนวณได้ค่า Cardinality เป็น 20% ของจำนวนแถวทั้งหมดในตารางก็จะทำให้อ๊อพติไมเซอร์ต้องอ่านข้อมูลออกมาทั้งหมดแล้วมากรองออกในภายหลังก็จะทำให้เสียเวลาในการอ่านข้อมูลมากเกินความจำเป็น ตามความสัมพันธ์ต่อไปนี้ [1,2]

Cardinality = Selectivity * จำนวนแถวทั้งหมดในตาราง

4. ชุดคำสั่ง

เราสามารถแบ่งชุดคำสั่งออกเป็น 3 กลุ่ม คือ กลุ่มที่ใช้สร้าง ปรับเปลี่ยนวัตถุ และลบวัตถุออกจากฐานข้อมูล (Data Definition Language หรือ DDL) กลุ่มที่ใช้จัดการเรียกดูหรือเปลี่ยนแปลงแก้ไขข้อมูล (Data Manipulation Language หรือ DML) กลุ่มที่ใช้ควบคุมสิทธิ์การใช้งานและควบคุมรายการเปลี่ยนแปลงข้อมูล (Data Control Language หรือ DCL)

ชุดคำสั่งในแต่ละกลุ่มมีการประมวลผลไม่เหมือนกัน ปัจจัยที่มีผลต่อประสิทธิภาพก็ไม่เหมือนกัน บทความนี้จะเน้นไปที่การทำความเข้าใจขั้นตอนการทำงานของอ๊อพติไมเซอร์เมื่อได้รับชุดคำสั่งในกลุ่มที่ใช้จัดการข้อมูล เช่น SELECT, INSERT, UPDATE และ DELETE เป็นต้น เนื่องจากมีการใช้มากและประสิทธิภาพของระบบโดยรวมจะขึ้นกับชุดคำสั่งเหล่านี้มากกว่าชุดคำสั่งอย่างอื่น ในหัวข้อนี้ จะได้แสดงตัวอย่างการใช้งานชุดคำสั่งดังต่อไปนี้

ตัวอย่างที่ 1 ชุดคำสั่งที่ใช้สร้างตารางข้อมูล dept

```
CREATE TABLE dept
(dno NUMBER(2) PRIMARY KEY,
 dname CHAR(30));
```

ตัวอย่างที่ 2 ชุดคำสั่งที่ใช้สร้างตาราง emp

```
CREATE TABLE emp
(eno NUMBER(2) PRIMARY KEY,
 ename CHAR(30) NOT NULL,
 job CHAR(10),
 dno NUMBER(2),
 FOREIGN KEY (dno) REFERENCES
dept(dno));
```

ตัวอย่างที่ 3 ชุดคำสั่งที่บันทึกข้อมูลเข้าตาราง dept และชุดคำสั่งที่ใช้จบทรานแซคชัน

```
INSERT INTO dept VALUES(10,
'Management');
INSERT INTO dept VALUES(20,
'Marketing');
INSERT INTO dept VALUES(30,
'Sales');
INSERT INTO dept VALUES(40,
'Production');
COMMIT;
```

ตัวอย่างที่ 4 ชุดคำสั่งที่บันทึกข้อมูลเข้าตาราง emp และชุดคำสั่งที่ใช้จบทรานแซคชัน

```
INSERT INTO emp VALUES(1, 'Anne',
'CEO', 10);
INSERT INTO emp
VALUES(2, 'Peter', 'Manager', 10);
INSERT INTO emp
VALUES(3, 'Smith', 'Manager', 20);
INSERT INTO emp
VALUES(4, 'Alex', 'Manager', 30);
INSERT INTO emp
VALUES(5, 'Ben', 'Clerk', 40);
INSERT INTO emp
VALUES(6, 'Scott', 'Manager', 40);
INSERT INTO emp
VALUES(7, 'Clark', 'Worker', 40);
INSERT INTO emp
VALUES(8, 'Frank', 'Clerk', 30);
INSERT INTO emp
VALUES(9, 'Peter', 'Worker', 40);
INSERT INTO emp
VALUES(10, 'Robert', 'Worker', 40);
COMMIT;
```

ตัวอย่างที่ 5 ชุดคำสั่งที่ใช้แสดงข้อมูลทั้งหมดในตาราง dept และ emp

```
SELECT * FROM dept;
SELECT * FROM emp;
```

ตัวอย่างที่ 6 ชุดคำสั่งที่ใช้เรียกดูพนักงานที่สังกัดฝ่าย Production

```
SELECT dno, ename, job
FROM emp
WHERE dno = 40;
```

ตัวอย่างที่ 7 ชุดคำสั่งที่ใช้เปลี่ยนแปลงข้อมูลในตาราง emp โดยย้ายให้พนักงานที่มีหมายเลขประจำตัว 8 ไปสังกัดฝ่าย Marketing

```
UPDATE emp
SET dno = 20
WHERE eno = 8;
```

ตัวอย่างที่ 8 ชุดคำสั่งที่ใช้ลบข้อมูลออกจากตาราง ลบข้อมูลของพนักงานที่ชื่อ Clark ออกจากฐานข้อมูล

```
DELETE FROM emp
WHERE eno = 7;
```

ตัวอย่างที่ 9 ชุดคำสั่งที่ยกเลิกการลบข้อมูลในตัวอย่างที่ 8

```
ROLLBACK;
```

5. การประมวลผลชุดคำสั่ง

เมื่อผู้ใช้งานส่งคำสั่งไปยังฐานข้อมูลโดยเครื่องมือทางซอฟท์แวร์หรือโปรแกรมใช้งาน ระบบจัดการฐานข้อมูลจะนำคำสั่งไปประมวลผลตามขั้นตอนดังนี้ คือ Parsing, Binding, Execution, และ Fetching ต่อไปนี้เป็นชุดคำสั่งที่ใช้เป็นตัวอย่างในการอธิบายในหัวข้อนี้คือ

```
SELECT ename, job
FROM emp
WHERE dno = 10;
```

5.1. Parsing

ขั้นตอนนี้อ๊อพติไมเซอร์จะตรวจสอบว่าชุดคำสั่งนี้เคยส่งมาประมวลผลและยังเก็บชุดคำสั่งนี้ไว้หรือไม่ ถ้ายังเก็บอยู่อ๊อพติไมเซอร์อาจจะไม่ต้องทำ Parsing โดยข้ามไปทำขั้นตอนต่อไป แต่ถ้าไม่พบชุดคำสั่งนี้ในระบบแล้วอ๊อพติไมเซอร์จะตรวจสอบความถูกต้องของชุดคำสั่งกับรูปแบบชุดคำสั่งที่ถูกต้องใน Data Dictionary หรือ Catalog จากนั้นจะตรวจสอบว่ามีตาราง emp ที่อ้างในชุดคำสั่งนี้หรือไม่ และผู้ที่ส่งชุดคำสั่งนั้นมามีสิทธิ์ใช้ตาราง emp นี้หรือไม่ หากไม่มีปัญหาก็จะเลือกแผนการสืบค้นที่เตรียมไว้แล้วถ้าเป็น Rule-Based Optimizer แต่ถ้าเป็น Cost-Based Optimizer ก็จะสร้างแผนดำเนินการหลายแผนที่เป็นไปได้และคำนวณค่าใช้จ่ายของแต่ละแผนจากสถิติเชิงประสิทธิภาพที่มีอยู่ในขณะนั้น แล้วเลือกแผนการสืบค้นที่มีค่าใช่จ่ายต่ำสุดเตรียมไว้เพื่อประมวลผลในขั้นต่อไป

5.2. Binding

ขั้นตอนนี้เป็นการนำค่าของข้อมูลไปแทนที่ตัวแปร (ถ้ามี) ในชุดคำสั่งเพื่อเตรียมประมวลผลในขั้นต่อไป

5.3. Execution

อ๊อพติไมเซอร์ตรวจสอบว่ามีข้อมูลที่ต้องการเรียกหรือเปลี่ยนแปลงในหน่วยความจำหรือไม่ ถ้ามีข้อมูลก็ไม่ต้องทำอะไรสำหรับชุดคำสั่ง SELECT หรือเปลี่ยนแปลงแก้ไขข้อมูลที่หน่วยความจำนี้สำหรับชุดคำสั่ง INSERT, UPDATE และ DELETE แต่ถ้าไม่มีข้อมูลก็จะอ่านจากดิสก์หรือเทปหรือสื่ออื่นๆเข้าไปในหน่วยความจำที่ใช้ร่วมกันแล้วส่งกลับหรือเปลี่ยนแปลงข้อมูลในหน่วยความจำนี้

จากตัวอย่างชุดคำสั่งข้างต้น อ๊อพติไมเซอร์จะตรวจสอบดูว่ามีข้อมูลในตาราง emp ที่มีค่าของ dno = 10 ในหน่วยความจำของระบบจัดการฐานข้อมูลหรือไม่ ถ้ามีก็ไม่ต้องทำอะไร แต่ถ้าไม่มีมันจะอ่านข้อมูลบนสื่อที่ใช้จัดเก็บข้อมูลในบล็อคหรือเพจที่มีข้อมูลแถวนี้ไปไว้บนหน่วยความจำของระบบจัดการฐานข้อมูล แล้วเลือกเอาเฉพาะแถวที่มีค่าของ dno = 10 เท่านั้น เตรียมส่งให้โปรเซสที่ส่งชุดคำสั่งมาประมวลผล

ชุดคำสั่ง SELECT มีจำนวนคำสั่งย่อยทั้งหมด 6 คำสั่ง โดยแต่ละชุดคำสั่งต้องประกอบด้วยคำสั่งย่อยอย่างน้อย 2 คำสั่งคือ SELECT และ FROM โดยชุดคำสั่งเต็มมีดังนี้

```
SELECT columns and/or expressions
FROM tables and/or views and/or synonyms/alias
[WHERE row conditions]
[GROUP BY columns and/or expressions]
[HAVING group conditions]
[ORDER BY selected columns and/or expressions;]
```

ลำดับการประมวลผลโดยทั่วไปคือ FROM, WHERE, GROUP BY , HAVING, SELECT และ ORDER BY ตามลำดับ

5.4. Fetching

อ๊อพติไมเซอร์ส่งข้อมูลที่ประมวลผลได้จากขั้นตอน Execution กลับไปให้โปรแกรมที่ส่งชุดคำสั่ง SELECT มา

เท่านั้น แต่จะไม่มีการทำขั้นตอนนี้สำหรับชุดคำสั่ง INSERT, UPDATE และ DELETE

6. สาเหตุที่ทำให้การประมวลผลชุดคำสั่งไม่มีประสิทธิภาพ

สาเหตุที่ทำให้การประมวลผลชุดคำสั่งไม่มีประสิทธิภาพมีหลายประการด้วยกันดังนี้

6.1. ไม่มีสถิติเชิงประสิทธิภาพหรือมีแต่เก่าเกินไป

กรณีที่สร้างตารางแล้วป้อนข้อมูลในช่วงต้นของการพัฒนาระบบงานมักจะไม่ได้ทำการรวบรวมและวิเคราะห์สถิติเชิงประสิทธิภาพไว้จึงทำให้การคำนวณ และเลือกแผนดำเนินการไม่สอดคล้องกับชุดคำสั่งเท่าที่ควรและเป็นผลให้ ผลที่ได้จากอ๊อพติไมเซอร์ไม่สอดคล้องกับข้อมูลที่มีอยู่จริงในฐานข้อมูล ซึ่งจะส่งผลประสิทธิภาพในการดึงข้อมูล **ฉนั้นจึงควรรวบรวมและวิเคราะห์สถิติเชิงประสิทธิภาพสม่ำเสมออย่างน้อยเดือนละครั้ง เป็นต้น**

6.2. ขาดโครงสร้างบางอย่างที่เหมาะสม เช่น อินเด็กซ์

กรณีที่มีการเรียกข้อมูลออกมาหลายตารางพร้อมกันนั้น ถ้านำคอลัมน์ที่ใช้เชื่อมโยงข้อมูลระหว่างตาราง (Joining Column) มาสร้างอินเด็กซ์ที่เหมาะสมก็อาจทำให้อ๊อพติไมเซอร์ทำงานน้อยลงทำให้การประมวลผลโดยรวมมีประสิทธิภาพมากขึ้นได้

สมมติว่ามีชุดคำสั่งออกรายงานดังข้างล่าง

```
SELECT emp.ename, dept.dname
FROM emp, dept
WHERE emp.dno = dept.dno;
```

เราสามารถเพิ่มประสิทธิภาพการทำงานของอ๊อพติไมเซอร์โดยการนำคอลัมน์ dno ของตาราง emp มาสร้าง Balanced-Tree Index ดังนี้

```
CREATE INDEX emp_dno_idx ON emp(dno);
```

จากนั้นสั่งประมวลผลชุดคำสั่ง SELECT ข้างบนอีกครั้งอาจพบว่ามีประสิทธิภาพดีขึ้น

6.3. ชุดคำสั่งไม่เหมาะสมกับงาน ชุดคำสั่งเขียนไม่ดี หรือแผนการสืบค้นไม่มีประสิทธิภาพ

6.3.1. ชุดคำสั่งไม่เหมาะสมกับงานหรือชุดคำสั่งเขียนไม่ดี

บางครั้งรายงานชิ้นหนึ่งอาจเขียนชุดคำสั่งได้หลายชุด นักพัฒนาโปรแกรมประยุกต์และผู้บริหารระบบฐานข้อมูลต้องศึกษาเปรียบเทียบประสิทธิภาพของชุดคำสั่งต่างๆ แล้วเลือกรูปแบบที่เหมาะสมในแต่ละสถานการณ์

สมมติว่าผู้ใช้งานต้องการข้อมูลเฉพาะบางแถวเท่านั้น เช่น ต้องการข้อมูลของพนักงานที่เป็นผู้จัดการฝ่าย Sales แต่ถ้าเขียนชุดคำสั่งให้แสดงรายชื่อผู้จัดการฝ่ายทุกคนออกมาแล้วให้ผู้ใช้งานเลือกเอาเองก็จะทำให้อ๊อพติไมเซอร์ทำงานเกินความจำเป็นดังชุดคำสั่งข้างล่าง

```
SELECT ename, dno
FROM emp
WHERE job = 'Manager';
```

สามารถทำให้ผู้ใช้งานได้ข้อมูลตรงตามความต้องการและยังเป็นการเพิ่มประสิทธิภาพให้การประมวลผลชุดคำสั่งข้างบนด้วยการเพิ่มเงื่อนไขให้ตรงความต้องการ กรณีนี้ให้เพิ่มเงื่อนไข dno = 30 ลงไป ถ้ามีอินเด็กซ์บนคอลัมน์ dno ของตาราง emp อยู่แล้วอ๊อพติไมเซอร์อาจใช้อินเด็กซ์นั้นก็จะทำให้การทำงานมีประสิทธิภาพดียิ่งขึ้น ดังนี้

```
SELECT ename, dno
FROM emp
WHERE job = 'Manager'
   AND dno = 30;
```

6.3.2. แผนการสืบค้นไม่มีประสิทธิภาพ

บางกรณีชุดคำสั่งหนึ่งอาจมีแผนการสืบค้นมากกว่าหนึ่งแผน ค่าใช้จ่ายของแต่ละแผนที่อ๊อพติไมเซอร์ประเมินได้ก่อนการทำงานอาจไม่ตรงตามที่ควรจะเป็น ณ ขณะนั้น ซึ่งจะทำให้อ๊อพติไมเซอร์ไม่ได้เลือกแผนสืบค้นที่ดีที่สุดก็ได้ ฉนั้นนักพัฒนาโปรแกรมประยุกต์และผู้บริหารฐานข้อมูลอาจต้องทดสอบประสิทธิภาพของบางชุดคำสั่งที่ใช้เวลาประมวลผลนานด้วยการสั่งให้อ๊อพติไมเซอร์ทำตามแผนการสืบค้นที่เป็นไปได้ทีละแผน แล้วนำค่าประสิทธิภาพของทุกแผนการสืบค้นมาเปรียบเทียบกัน จากนั้นเลือกแผนการสืบค้นที่มีประสิทธิภาพดีที่สุดไปใช้งานจริง

6.4. ทรัพยากรมีไม่เพียงพอ

เมื่อเวลาผ่านไปสักระยะจะพบว่าชุดคำสั่งเดิมเริ่มทำงานช้าลง ทั้งนี้อาจมีปัจจัยหลายอย่างประกอบกัน เช่น มีปริมาณข้อมูลมากขึ้น มีเนื้อที่ดิสก์ที่จัดเก็บข้อมูลเหลือน้อยลง มีจำนวนผู้ใช้งานมากขึ้น มีจำนวนโปรแกรมใช้งานมากขึ้น จำนวนผู้ใช้งานในเครือข่ายมีมากขึ้น เป็นต้น ผู้ที่เกี่ยวข้องต้องหมั่นตรวจสอบประสิทธิภาพของระบบอย่างสม่ำเสมอและทำการปรับแต่งระบบฐานข้อมูล ระบบปฏิบัติการคอมพิวเตอร์ ระบบเครือข่าย และโปรแกรมประยุกต์ให้มีประสิทธิภาพมากขึ้นเป็นระยะ เช่น ไตรมาสละครั้ง เป็นต้น

6.5 การครอบครองข้อมูลหรือทรัพยากร (Data and Resource Locking) ของแต่ละทรานแซคชันนานเกินไป

กรณีที่ผู้ใช้งานใช้เวลาในการป้อนข้อมูลหรือแก้ไขข้อมูลนานเกินสมควรก็อาจเป็นสาเหตุทำให้ระบบโดยรวมทำงานช้าลงได้เพราะการป้อนข้อมูลหรือการแก้ไขข้อมูลนั้นมักต้องล็อคข้อมูลก่อนป้อนหรือแก้ไขทำให้ผู้อื่นที่เข้ามาทีหลังไม่สามารถเรียกใช้ข้อมูลนั้นได้จนกว่าผู้ล็อคปล่อยข้อมูลออกมาด้วยการจบรายการทรานแซคชันก่อน สามารถลดการรอคอยข้อมูลกันได้โดยนักพัฒนาโปรแกรมประยุกต์ต้องจบรายการทรานแซคชันด้วยการใส่ชุดคำสั่ง COMMIT ทันทีเมื่อจบรายการทรานแซคชันในโปรแกรม และขอความร่วมมือผู้ใช้งานโปรแกรมประยุกต์ไม่ควรใช้เวลาในการป้อนหรือแก้ไขข้อมูลนานเกินสมควรของแต่ละโปรแกรม และออกจากโปรแกรมหรือเลิกใช้โปรแกรมทันทีที่ไม่ได้ใช้งานก็จะทำให้ระบบจัดการฐานข้อมูลไม่ต้องเสียทรัพยากรเกินความจำเป็น ประสิทธิภาพของการใช้งานระบบเครือข่ายก็จะดีขึ้นด้วย

6.6. สาเหตุอื่นๆ

สาเหตุอื่น ๆ มี อาทิ การออกแบบฐานข้อมูลที่ไม่สอดคล้องกับการนำไปใช้งานก็เป็นสาเหตุหนึ่งของประสิทธิภาพที่ไม่ดี เช่น บางครั้งจำเป็นต้องนำข้อมูลจากหลายตารางมารวมกันเป็นตารางเดียวเพื่อลดการเชื่อมโยงข้อมูลในการออกรายงานเชิงวิเคราะห์ (Business Intelligence) ซึ่งจะทำให้การออกรายงานรวดเร็วมากขึ้น แต่ไม่ได้ทำการรวมตาราง เป็นต้น

7. แหล่งข้อมูลที่อ๊อพติไมเซอร์ใช้ประมวลผลชุดคำสั่ง

จำนวนแหล่งข้อมูล (Number of Resources) ที่อ๊อพติไมเซอร์จะใช้ประมวลผลชุดคำสั่งนั้นขึ้นกับลักษณะของชุดคำสั่ง โดยอ๊อพติไมเซอร์อาจแปลงชุดคำสั่งที่ได้รับเป็นชุดคำสั่งอื่นที่ทำงานแล้วได้ผลเหมือนกันแต่มีประสิทธิภาพมากกว่า ดังนี้

7.1. Unary Operations

แบบนี้มีแหล่งข้อมูลป้อนเข้าเพียงแหล่งเดียวเท่านั้น อ๊อพติไมเซอร์จะพิจารณาใช้วิธีการเรียกใช้ข้อมูล (Access Path) ที่มีประสิทธิภาพ โดยทั่วไปวิธีเรียกใช้ข้อมูลจะมีประสิทธิภาพจากมากไปหาน้อยดังนี้คือ Index Unique

Scan, Index Range Scan, Index Fast Full Scan, Index Full Scan, Bitmap Index และ Full Table Scan แต่มีข้อยกเว้นบางประการ เช่น ถ้าข้อมูลทุกแถวใช้เนื้อที่จัดเก็บไม่เกิน 1 Page หรือ 1 Block แล้วการอ่านแบบ Full Table Scan จะมีประสิทธิภาพดีกว่า Index Scan และถ้าต้องอ่านข้อมูลส่วนใหญ่หรือทั้งตารางแล้วการอ่านแบบ Full Table Scan ก็จะดีกว่า Index Scan เป็นต้น

7.2. Binary Operations

แบบนี้มีแหล่งข้อมูลป้อนเข้าจำนวน 2 แหล่ง อ๊อพติไมเซอร์จะพิจารณาใช้เทคนิคที่เรียกว่า Join ซึ่งมีหลายวิธี โดยทั่วไปประสิทธิภาพของ Join จากมากไปน้อยคือ Hash Join, Sort Merge Join และ Nested Loop Join ตามลำดับ และ Nested Loop Join ใช้แทน Join ชนิดอื่นได้ทุกชนิด

7.3. N-ary Operations

แบบนี้มีแหล่งข้อมูลป้อนเข้ามากกว่า 2 แหล่ง อ๊อพติไมเซอร์จะพิจารณาใช้ Relational Operators เช่น Union, Union All, Intersect และ Minus

8. การปรับปรุงและเพิ่มประสิทธิภาพชุดคำสั่ง

ต่อไปนี้เป็นการปรับปรุงและเพิ่มประสิทธิภาพชุดคำสั่ง เราควรเขียนชุดคำสั่งให้มีประสิทธิภาพตามที่อ๊อพติไมเซอร์ต้องการ และควรทดสอบชุดคำสั่งหลายๆชุดคำสั่งที่ให้ผลเหมือนกันแต่เลือกเอาชุดคำสั่งที่มีประสิทธิภาพดีที่สุดไปใช้งาน ระบบจัดการฐานข้อมูลจะมีวิธีและเครื่องมือติดตามและตรวจสอบประสิทธิภาพอยู่แล้ว ให้ไปศึกษาเพิ่มเติมจากคู่มือของผู้ผลิต

8.1. Full Table Scan

วิธีนี้จะใช้ก็ต่อเมื่อต้องอ่านข้อมูลจำนวนมาก อาจจะเป็นส่วนใหญ่หรือทั้งตาราง ซึ่งจะให้ประสิทธิภาพที่ดีกว่าใช้อินเด็กซ์ หรือไม่มีอินเด็กซ์ที่เหมาะสม หรือจำนวนแถวที่จะได้มีมากกว่า 4% ของตาราง [3] หรือตารางข้อมูลที่ต้องการมีจำนวนแถวน้อยมากซึ่งอาจมีขนาดทุกแถวรวมกันใช้เนื้อที่จัดเก็บไม่เกิน 1 Page หรือ 1 Block (โดยทั่วไป 1 Page หรือ 1 Block อาจมีขนาด 2K – 32K bytes) เป็นต้น ตัวอย่างชุดคำสั่งที่อ๊อพติไมเซอร์อาจใช้วิธี Full Table Scan มีดังนี้

ตัวอย่างที่ 1 ไม่มีการระบุเงื่อนไขในการเรียกข้อมูล อ๊อพติไมเซอร์จะอ่านข้อมูลโดยวิธี Full Table Scan

```
SELECT *
FROM emp;
```

ตัวอย่างที่ 2 มีเงื่อนไขในการค้นหาแต่จำนวนแถวของข้อมูลที่ได้คิดเป็นสัดส่วนที่มากของตารางข้อมูล เช่น จำนวนแถวที่จะได้เกินกว่า 80% ของข้อมูลทั้งตาราง อ๊อพติไมเซอร์จะอ่านข้อมูลโดยวิธี Full Table Scan

```
SELECT *
FROM emp
WHERE dno IN (10, 20, 40);
```

ตัวอย่างที่ 3 ถ้าจำนวนแถวที่จะได้ตามเงื่อนไขมีมากกว่า 4% ของตารางแล้วอ๊อพติไมเซอร์อาจอ่านข้อมูลโดยใช้วิธี Full Table Scan แทน Index Scan แม้ว่าคอลัมน์ในเงื่อนไขนั้นถูกทำเป็นอินเด็กซ์แล้วก็ตาม

```
SELECT *
FROM emp
WHERE dno = 40;
```

8.2. Index Unique Scan

อ๊อพติไมเซอร์จะใช้ Index Unique Scan ถ้าคอลัมน์ใน WHERE เป็นคีย์หลัก หรือ Unique Key ซึ่งเป็นวิธีที่เร็วมาก ฉนั้นถ้าต้องการค้นหาข้อมูลให้เร็วก็ควรใช้คอลัมน์ที่คีย์หลักหรือ Unique Key เป็นเงื่อนไขในการค้นหาข้อมูล

ตัวอย่างที่ 1 คอลัมน์ eno เป็นคีย์หลักและถูกสร้างเป็น Unique Index ของตาราง EMP ชุดคำสั่งข้างล่างจะใช้ Index Unique Scan

```
SELECT *
FROM EMP
WHERE eno = 10;
```

8.3. Index Range Scan

อ๊อพติไมเซอร์จะใช้ Index Range Scan ถ้าคอลัมน์ใน WHERE ถูกทำอินเด็กซ์และจำนวนแถวที่จะได้ตามเงื่อนไขมีจำนวนไม่เกิน 4% ของตารางข้อมูล

ตัวอย่างที่ 1 สมมุติว่าตาราง EMP มีจำนวนแถวทั้งหมด 10,000 แถว ค่า Selectivity ของคอลัมน์ ename = 1/10000 และสร้างอินเด็กซ์บนคอลัมน์ ename แล้วชุดคำสั่งข้างล่างจะใช้ Index Range Scan

```
CREATE INDEX emp_ename_idx ON
emp(ename);

SELECT *
FROM emp
WHERE ename IN ('Smith', 'Anne',
'Peter');
```

8.4. Index Full Scan

วิธีนี้อ๊อพติไมเซอร์จะอ่านข้อมูลจากอินเด็กซ์ทั้งหมดทุก Page หรือ ทุก Block ในคราวเดียวกัน แล้วไปอ่านจากตารางข้อมูลตาม Address ที่ได้จากอินเด็กซ์ โดยจะให้ประสิทธิภาพดีกว่าการอ่านจากอินเด็กซ์ปกติซึ่งอ่านบาง Page หรือ บาง Block แต่อ่านครั้งละ 1 Page หรือ 1 Block โดยวิธีนี้มีเงื่อนไขว่าต้องมีคอลัมน์ใน WHERE ถูกทำเป็นอินเด็กซ์และคอลัมน์นี้ต้องมีค่าข้อมูลเสมอ (NOT NULL)

ตัวอย่างที่ 1 สมมุติว่าคอลัมน์ dno ในตาราง EMP ถูกสร้างอินเด็กซ์ และตารางนี้มีข้อมูล 10,000 แถว ชุดคำสั่งข้างล่างอาจใช้วิธี Index Full Scan

```
CREATE INDEX emp_dno_idx ON
emp(dno);

SELECT *
FROM emp
WHERE job in ('Manager', 'Clerk')
and dno IS NOT NULL;
```

8.5. Index Fast Full Scan

วิธีนี้อ๊อพติไมเซอร์อ่านอินเด็กซ์อย่างเดียวโดยไม่อ่านตารางข้อมูลโดยมีเงื่อนไขว่าทุกคอลัมน์ที่อยู่ใน SELECT Clause ต้องถูกนำไปสร้างอินเด็กซ์รวมกับคอลัมน์ที่อยู่ใน WHERE Clause และคอลัมน์เหล่านั้นมีการเปลี่ยนแปลงหรือแก้ไขข้อมูลน้อยด้วย

ตัวอย่างที่ 1 นำทุกคอลัมน์ที่ปรากฏใน SELECT Clause ไปรวมกับคอลัมน์ที่ปรากฏใน WHERE Clause แล้วสร้างอินเด็กซ์ อ๊อพติไมเซอร์จะใช้วิธี Index Fast Full Scan ในการประมวลผลชุดคำสั่ง SELECT

```
CREATE INDEX emp_n_j_idx
ON emp(ename, job);

SELECT ename, job
FROM emp
WHERE ename = 'Clark';
```

8.6. Bitmap Index

อินเด็กซ์ชนิดนี้จะมีจำนวนชิ้นของอินเด็กซ์ (Segment) เท่ากับจำนวนค่าที่แตกต่างกันของคอลัมน์ที่ถูกนำมาสร้าง Bitmap Index และแต่ละชิ้นของอินเด็กซ์ประกอบด้วยค่าของข้อมูลในคอลัมน์ที่ทำอินเด็กซ์ 1 ค่าและค่าบิตประจำตำแหน่งของแถวข้อมูลที่บ่งบอกว่าแถวนั้นมีค่าของคอลัมน์ที่ถูกทำเป็น Bitmap Index เป็นค่าเดียวกันกับค่าของอินเด็กซ์ชิ้นนั้นหรือไม่ หมายความว่าแต่ละชิ้นของอินเด็กซ์จะมีจำนวนบิตเท่ากับจำนวนแถวทั้งหมดของตารางนั้น และแต่ละชิ้นของอินเด็กซ์จะบอกว่ามีแถวใดบ้างที่มีข้อมูลของคอลัมน์ที่ถูกทำเป็น Bitmap Index ตรงกับค่าของอินเด็กซ์ชิ้นนั้นๆ

อินเด็กซ์ชนิดนี้เหมาะที่จะสร้างบนคอลัมน์ที่มีจำนวนค่าข้อมูลที่แตกต่างกันน้อยแต่มีจำนวนแถวมากไม่น้อยกว่าแสนแถวจึงจะมีประสิทธิภาพดีและคุ้มค่า เช่น สมมุติว่าตาราง EMP มีจำนวนแถว 200,000 แถว และในคอลัมน์ dno ซึ่งเป็นหมายเลขฝ่ายมีจำนวนแค่ 4 ฝ่าย นั่นแสดงว่าโดยเฉลี่ยแล้วแต่ละฝ่ายมีคนสังกัดเท่ากับ 200,000/4 = 50,000 คนต่อฝ่าย แบบนี้เหมาะที่จะสร้าง Bitmap Index มากกว่าอินเด็กซ์ธรรมดา (B Tree and B*Tree)

ตัวอย่างที่ 1 สมมุติว่ามีข้อมูล 200,000 แถวในตาราง EMP และสร้าง Bitmap Index บนคอลัมน์ dno ซึ่งมีจำนวนค่าข้อมูลที่แตกต่างกันเพียง 4 ค่า และใช้เป็นเงื่อนไขในชุดคำสั่ง SELECT จากนั้นประมวลผลชุดคำสั่ง SELECT แบบนี้อ๊อพติไมเซอร์อาจใช้ Bitmap Index ในการประมวลผลซึ่งให้ประสิทธิภาพดีกว่าการใช้อินเด็กซ์ธรรมดา

```
CREATE BITMAP INDEX emp_dno_bit_idx
ON emp(dno);

SELECT *
FROM emp
WHERE dno = 30;
```

8.7. Inlist Iterator

อ๊อพติไมเซอร์จะแปลงโอเปอร์เรเตอร์ IN ไปเป็นโอเปอร์เรเตอร์ OR ก่อนที่จะประมวลผลเพราะให้ประสิทธิภาพที่ดีกว่า

ตัวอย่างที่ 1 ส่งชุดคำสั่งที่ใช้โอเปอร์เรเตอร์ IN

```
SELECT *
FROM emp
WHERE dno IN (10, 30);
```

อ๊อพติไมเซอร์จะแปลงเป็นชุดคำสั่งต่อไปนี้แทน

```
SELECT *
FROM emp
WHERE dno = 10 OR dno = 30;
```

8.8. Nested Loop Join

เป็นเทคนิคพื้นฐานที่นำข้อมูลจาก 2 แหล่งมาจับคู่กันและโดยทั่วไปถือว่าเป็นเทคนิคของการจับคู่ที่มีประสิทธิภาพต่ำที่สุดแต่ใช้แทนเทคนิคการจับคู่แบบอื่นได้ทุกแบบ

การทำงานเริ่มจากการอ่านข้อมูลตารางที่หนึ่งแบบ Full Table Scan แล้วนำข้อมูลแถวแรกที่ได้ไปจับคู่กับทีละแถวในตารางที่สองจนหมดก็จะได้ข้อมูลระหว่างทางชุดที่หนึ่งออกมา แล้วกลับไปที่ตารางแรกนำข้อมูลแถวที่สองไปจับคู่กับทีละแถวในตารางที่สองจนหมดก็จะได้ข้อมูลระหว่างทางชุดที่สองออกมา ทำเช่นนี้เรื่อยไปจนตารางทั้งสองจับคู่กันหมดทุกแถว

ปกติเรามักจะเลือกตารางที่มีจำนวนแถวน้อยไปจับคู่กับตารางที่มีจำนวนแถวมากกว่า เราต้องทราบด้วยว่าระบบจัดการฐานข้อมูลที่เราใช้งานนั้นจับคู่ตารางจากซ้ายไปขวาหรือขวาไปซ้าย (รายชื่อตารางใน FROM Clause) เช่น Oracle Database ใช้ตารางขวาสุดไปจับคู่กับตารางที่อยู่ทางซ้ายและนำผลที่ได้ไปจับคู่กับตารางทางซ้ายที่อยู่ถัดไป (ถ้ามี) ใน FROM Clause กรณีนี้ควรระบุชื่อตารางที่จะให้จำนวนแถวออกมาน้อยที่สุดอยู่ขวามือสุด ส่วน

ตารางที่จะให้จำนวนแถวมากกว่าก็ระบุทางซ้ายใน FROM Clause เป็นต้น การกำหนดลำดับชื่อตารางนี้มีความสำคัญมาก หากสลับกันก็จะมีประสิทธิภาพต่างกัน

Nested Loop Join เหมาะสำหรับการจับคู่ตารางที่มีจำนวนแถวน้อยๆ แต่ถ้ามีจำนวนแถวมากก็ควรใช้การจับคู่วิธีอื่น

ตัวอย่างที่ 1 แสดงรายชื่อพนักงานและชื่อของหน่วยงานที่สังกัดบนรายงาน โดยแสดงเฉพาะผู้ที่มีชื่อขึ้นต้นด้วยอักษร S สังเกตจะพบว่า ตาราง DEPT มีจำนวนแถวน้อยกว่าจำนวนแถวของตาราง EMP

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.dno = d.dno
      AND ename LIKE 'S%';
```

การประมวลผลจะเริ่มจากนำแถวแรกของตาราง DEPT ไปจับคู่กับทีละแถวในตาราง EMP ที่พนักงานมีชื่อที่ขึ้นต้นด้วยอักษร S จากนั้นก็นำแถวที่สองของตาราง DEPT ไปจับคู่กับทีละแถวของตาราง EMP ที่พนักงานมีชื่อที่ขึ้นต้นด้วยอักษร S แล้วทำซ้ำเช่นนี้จนนำแถวสุดท้ายของตาราง DEPT ไปจับคู่กับทีละแถวของตาราง EMP ที่พนักงานมีชื่อที่ขึ้นต้นด้วยอักษร S

8.9. Sort Merge Join

วิธีนี้ดีสำหรับการจับคู่ตารางที่มีขนาดไม่ใหญ่มากนักและเครื่องหมายที่ใช้เป็นเงื่อนไขไม่ใช่เครื่องหมายเท่ากับ (Inequality Condition) เช่น <, <=, >, >= เป็นต้น โดยจะเรียงข้อมูลในแต่ละตารางตามคอลัมน์ที่ใช้ในการจับคู่ทีละตาราง แล้วนำผลที่ได้มาจับคู่กัน

ตัวอย่างที่ 1 แสดงชื่อและหน่วยงานที่สังกัดของผู้ที่มีชื่อขึ้นต้นด้วยอักษรที่มีลำดับหลังอักษร F

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.dno = d.dno
      AND e.ename > 'F';
```

ถ้าอ๊อพติไมเซอร์ใช้วิธี Sort Merge Join มันจะกรองเอาเฉพาะแถวที่มีชื่อที่ขึ้นต้นด้วยอักษรที่ตามหลังอักษร F ในตาราง EMP แล้วเรียงข้อมูลที่ได้ตามค่าของคอลัมน์ dno และเรียงข้อมูลตาราง DEPT ตามค่าของคอลัมน์ dno แล้วนำผลที่ได้ทั้งสองชุดมาจับคู่กันโดยใช้ค่าของคอลัมน์ dno ในการจับคู่ เริ่มจากนำแถวแรกของชุดแรกไปหาแถวของชุดที่สองที่มีค่า dno ตรงกันออกมาทีละแถวจนค่าของ dno ไม่ตรงกันจึงหยุดจับคู่ จากนั้นกลับไปที่ชุดแรกที่ค่าของ dno เป็นค่าใหม่ถัดไป นำไปจับคู่กับแถวที่มีค่า dno ตรงกันในชุดที่สองทีละแถวจนไม่พบค่า dno ที่ตรงกันก็หยุด วนทำเช่นนี้เรื่อยไปจนหมดทุกค่าของ dno ก็จะได้ผลลัพธ์ออกมา

8.10. Hash Join

วิธีนี้ใช้สำหรับการจับคู่ตารางที่มีขนาดใหญ่และเงื่อนไขในการจับคู่ใช้เครื่องหมายเท่ากับ (Equijoin) เท่านั้น เริ่มจากนำตารางที่มีขนาดเล็กที่สุดมาแบ่งซอยเป็นตารางย่อยๆในหน่วยความจำ และแบ่งซอยตารางที่สองเป็นตารางย่อยๆเข้าไปไว้ในหน่วยความจำเช่นกัน จากนั้นนำตารางย่อยแรกของตารางแรกไปจับคู่กับทีละตารางย่อยของตารางที่สองจนหมด แล้วกลับไปนำตารางย่อยที่สองของตารางแรกไปจับคู่กับทีละตารางย่อยของตารางที่สองจนหมด วนทำเช่นนี้เรื่อยไปจนครบทุกตารางย่อยของตารางแรก

ตัวอย่างที่ 1 แสดงชื่อและหน่วยงานที่สังกัดของผู้ที่มีชื่อขึ้นต้นด้วยอักษร S โดยสมมุติว่าตาราง EMP มีข้อมูลไม่น้อยกว่า 100,000 แถว

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.dno = d.dno
      AND e.ename LIKE 'S%';
```

ถ้าอ๊อพติไมเซอร์ใช้ Hash Join มันจะแบ่งซอยตาราง DEPT เป็นตารางย่อยๆไว้ในหน่วยความจำ และกรองเอาเฉพาะแถวที่ค่าในคอลัมน์ ename ของตาราง EMP ขึ้นต้นด้วยอักษร S แล้วแบ่งซอยผลที่ได้ออกเป็นตารางย่อยๆในหน่วยความจำเช่นกัน จากนั้นนำตารางย่อยแรกของตาราง DEPT ไปจับคู่กับทีละตารางย่อยที่กรองได้จากตาราง EMP จนหมด เสร็จแล้วกลับไปนำตารางย่อยที่สองของตาราง DEPT ไปจับคู่กับทีละตารางย่อยที่กรองได้จากตาราง EMP จนหมด วนทำเช่นนี้เรื่อยไปจนครบ

9. สรุป

บทความนี้อธิบายขั้นตอนการประมวลผลชุดคำสั่ง SQL โดยอ๊อพติไมเซอร์ของระบบจัดการฐานข้อมูลเชิงสัมพันธ์ รวมทั้งคำแนะนำสำหรับนักพัฒนาโปรแกรมประยุกต์และผู้ดูแลระบบฐานข้อมูลให้เขียนชุดคำสั่งให้สอดคล้องกับการทำงานเพื่อลดขั้นตอนการทำงานของอ๊อพติไมเซอร์ เมื่อเข้าใจการทำงานของอ๊อพติไมเซอร์แล้วก็จะทำให้สามารถเขียนชุดคำสั่ง SQL ได้หลายวิธีตามสภาพแวดล้อมที่แตกต่างกัน นอกจากนี้เราควรรวบรวมสถิติเชิงประสิทธิภาพเป็นประจำให้กับตารางและอินเด็กซ์ที่มีการเปลี่ยนแปลงแก้ไขข้อมูลบ่อยๆ อย่างน้อยทุก 1-3 เดือน พยายามทำความเข้าใจธรรมชาติของตาราง วิว และอินเด็กซ์ที่เราใช้ว่ามีขนาดใหญ่เพียงใด มีการเปลี่ยนแปลงแก้ไขข้อมูลบ่อยไหม ควรมีอินเด็กซ์หรือไม่ชนิดใด จะจับคู่ข้อมูลในตารางด้วยวิธีใด สุดท้ายควรเขียนและทดสอบชุดคำสั่งในรูปแบบที่อ๊อพติไมเซอร์ต้องการมากที่สุดตามสถานการณ์ต่างๆ ด้วยเครื่องมือทางซอฟท์แวร์หรือโปรแกรมพิเศษที่ระบบจัดการฐานข้อมูลมีให้ใช้ เปรียบเทียบประสิทธิภาพที่ได้ของชุดคำสั่งที่ให้ผลเหมือนกันแล้วเลือกเอาชุดคำสั่งที่ให้ประสิทธิภาพที่ดีที่สุดไปใช้งาน

เอกสารอ้างอิง

[1] Jean-Franzois verier, *Oracle Database 11g: SQL Tuning Workshop*, 1st ed. California: Oracle Corporation, 2008.

[2] *Oracle Database 11g: Performance Tuning*, 1st ed. California: Oracle Corporation, 2008.

[3] *Oracle8i Designing and Tuning for Performance*, Oracle Corporation, 1999.

[4] *DB2 for Linux, UNIX, and Windows Performance Tuning and Monitoring Workshop*, 1st ed. New York: IBM, 2006.

[5] C. J. Date, *An Introduction to Database Systems*, 8th ed. Boston: Pearson Education, 2004.

[6] Carlos Coronel, Steven Morris, Peter Rob, *Database Systems: Design, Implementation, and Management*, 10th ed. Boston: Cengage Learning, 2012.

[7] Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, 6th ed. Boston: Addison-Wesley, 2011.