การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย สำหรับการสอบถามแบบสมาชิก

Encoded Bitmap Index Enhancement using Frequent Itemsets for Membership Queries

จรรยา สายนุ้ย

Janya Sainui

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Prince of Songkla University
2552

ลิขสิทธิ์ของมหาวิทยาลัยสงขลานครินทร์

ชื่อวิทยานิพนธ์ การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย สำหรับการสอบถามแบบสมาชิก
ผู้เขียน นางสาวจรรยา สายนุ้ย
สาขาวิชา วิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

…….....
(ผู้ช่วยศาสตราจารย์ ดร.ศิริรัตน์ วณิชโยบล)

คณะกรรมการสอบ

…….....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.โอม ศรนิล)

…….....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.ศิริรัตน์ วณิชโยบล)

…….....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วิภาดา เวทย์ประสิทธิ์)

…….....กรรมการ
(ดร.ลัดดา ปรีชาวีรกุล)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

.....
(รองศาสตราจารย์ ดร.เกริกชัย ทองหนู)
คณบดีบัณฑิตวิทยาลัย

ชื่อวิทยานิพนธ์ การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย สำหรับการสอบถามแบบสมาชิก

ผู้เขียน นางสาวจรรยา สายนุ้ย

สาขาวิชา วิทยาการคอมพิวเตอร์

ปีการศึกษา 2551

บทคัดย่อ

การทำดัชนีเป็นเทคนิคการเพิ่มความเร็วในการค้นหาข้อมูล โดยไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ และสำหรับระบบคลังข้อมูลแล้วการทำดัชนีที่นิยมใช้คือการทำดัชนีแบบบิตแมป เนื่องจากสามารถดำเนินการระดับบิตอันเป็นการสนับสนุนการทำงานของฮาร์ดแวร์ก่อนเข้าถึงข้อมูลจริง ทำให้การประมวลผลมีความรวดเร็วยิ่งขึ้น

วิทยานิพนธ์นี้ เป็นการนำเสนอขั้นตอนวิธีการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส ซึ่งดัชนีบิตแมปแบบเข้ารหัส เป็นเทคนิคการทำดัชนีแบบบิตแมปที่มีประสิทธิภาพสูงที่สุดในด้านการใช้พื้นที่ในการจัดเก็บ เมื่อเปรียบเทียบกับดัชนีบิตแมปที่เคยมีมา ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน เพื่อเพิ่มประสิทธิภาพด้านเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแบบเข้ารหัส ในวิทยานิพนธ์นี้จึงนำกลุ่มข้อมูลที่ปรากฏบ่อย มาช่วยในการเข้ารหัสเพื่อให้ได้การเข้ารหัสที่ดี โดยใช้หลักการที่ว่า แต่ละค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันบ่อย เมื่อถูกสอบถามด้วยกันอีก จะสามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบให้เหลือน้อยที่สุดได้ จากผลการวิเคราะห์และทดลองเปรียบเทียบระหว่างดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย กับดัชนีบิตแมปที่เคยมีมา พบว่า ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย มีประสิทธิภาพในแง่ Space-Time Trade-off (การแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่กับเวลา) สำหรับการสอบถามแบบสมาชิกดีกว่าดัชนีบิตแมปที่เคยมีมา

Thesis Title	Encoded Bitmap Index Enhancement using Frequent Itemsets for Membership Queries
Author	Miss Janya Sainui
Major Program	Computer Science
Academic Year	2008

ABSTRACT

Indexing techniques based on bitmap representations are well suited to a warehouse system. They significantly improve query processing time by utilizing low-cost Boolean operations and multiple index scans, executing queries by performing simple predicate conditions on the index level before going to the primary data source.

In this thesis, Encoded Bitmap Index enhancement is introduced. Encoded Bitmap Index outperforms the best bitmap indices in term of space requirement, comparing with Simple Bitmap Index, Interval Bitmap Index, Scatter Bitmap Index and Dual Bitmap Index. To improve existing Encoded Bitmap Index, in this thesis, we apply a frequent itemsets mining to find a well-defined encoding scheme, leading to improve query processing time. In other word, we use frequent itemsets mining to find frequent itemsets of indexed attribute values before encoding, leading to reduce bitmap vectors to be accessed. The comparative study shows that the performance of Encoded Bitmap Index using frequent itemsets is better than those found by existing techniques for membership queries from the point of view of space-time trade-off.

สารบัญ

หน้า

สารบัญ .....(6)
รายการตาราง .....(9)
รายการภาพประกอบ.....(11)
บทที่
1 บทนำ 1
1.1 ความเป็นมา 1
1.2 งานวิจัยที่เกี่ยวข้อง 2
1.3 วัตถุประสงค์ 4
1.4 วิธีการดำเนินการวิจัย.....4
1.5 ขอบเขตการวิจัย 5
1.6 ขั้นตอนการดำเนินงาน 5
1.7 ระยะเวลาดำเนินงานและแผนการดำเนินงาน 6
1.8 เครื่องมือและอุปกรณ์ 7
1.9 ประโยชน์ที่คาดว่าจะได้รับ.....7
2 ทฤษฎีที่เกี่ยวข้อง.....8
2.1 คลังข้อมูล 8
2.1.1 ความหมายของคลังข้อมูล.....8
2.1.2 คุณลักษณะของคลังข้อมูล.....8
2.1.3 ความแตกต่างระหว่างฐานข้อมูลปฏิบัติการและคลังข้อมูล 9
2.1.4 สถาปัตยกรรมของคลังข้อมูล (Data Warehouse Architecture).....11
2.1.5 การประยุกต์ใช้เทคโนโลยีคลังข้อมูล.....12
2.2 การสร้างดัชนี(Indexing).....12
2.2.1 ดัชนีแบบ B-Tree 13
2.2.1.1 ข้อดีของดัชนีแบบ B-Tree.....14
2.2.1.2 ข้อจำกัดของดัชนีแบบ B-Tree 14

สารบัญ (ต่อ)

หน้า

2.3 กลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets).....14
2.3.1 นิยามเทอมพื้นฐาน.....14
2.3.2 อัลกอริทึมการหากลุ่มข้อมูลที่ปรากฏบ่อย 15
2.3.2.1 อัลกอริทึม Apriori 15
2.3.2.2 อัลกอริทึม FP-growth.....19
2.3.2.3 อัลกอริทึม Closed 24
2.3.2.4 อัลกอริทึม Index-BitTableFI.....28
3 ดัชนีแบบบิตแมป (Bitmap index).....32
3.1 ดัชนีบิตแมปแบบพื้นฐาน (Simple Bitmap Index) 32
3.2 ดัชนีบิตแมปแบบบีบอัด (Compression Bitmap Index).....34
3.2.1 WAH (Word-aligned Hybrid) Compression 34
3.2.2 RLH (Run-Length Huffman) Compression.....36
3.3 ดัชนีบิตแมปแบบไม่บีบอัด (Compression Bitmap Index).....38
3.3.1 ดัชนีบิตแมปแบบช่วง (Interval Bitmap Index) 39
3.3.2 ดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index).....41
3.3.3 ดัชนีบิตแมปแบบคู่กัน (Dual Bitmap Index).....43
3.3.4 ดัชนีบิตแมปแบบเข้ารหัส (Encoded Bitmap Index) 45
3.4 เปรียบเทียบดัชนีบิตแมปแบบไม่บีบอัดทั้ง 5 ชนิด.....48
4 การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส.....50
4.1 ขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย…….. 50
4.2 การสอบถามข้อมูลแบบค่าเท่ากัน.....62
4.3 การสอบถามข้อมูลแบบสมาชิก.....62
4.4 ข้อเด่นของดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย 64
4.5 ข้อด้อยของดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย 65

สารบัญ (ต่อ)

หน้า

5 การวิเคราะห์และผลการทดลอง 66
5.1 ค่าใช้จ่ายเกี่ยวกับพื้นที่ที่ใช้ในการจัดเก็บดัชนี 66
5.2 ค่าใช้จ่ายเกี่ยวกับเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิก 84
5.3 ค่าใช้จ่ายในแง่ Space-Time Trade-off 89
6 บทสรุปและข้อเสนอแนะ 89
6.1 สรุป 89
6.2 ข้อเสนอแนะและงานในอนาคต 94
บรรณานุกรม 95
ภาคผนวก 98
ภาคผนวก ก 99
ก.1 เตรียมข้อมูลเพื่อใช้ในการทดลอง 99
ก.2 การเขียนโปรแกรมเพื่อทดลองสอบถามแบบสมาชิกบนดัชนีบิตแมป 102
ก.2.1 ขั้นตอนการทำงานหลักของการสอบถามบนดัชนีบิตแมปทั้ง 6 ชนิด 104
ก.2.2 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบพื้นฐาน 105
ก.2.3 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบช่วง 106
ก.2.4 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบเข้ารหัสทั่วไป 109
ก.2.5 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบกระจาย 111
ก.2.6 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบคู่กัน 113
ก.2.7 ขั้นตอนการสอบถามบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 115
ก.2.8 ขั้นตอนวิธีการลดรูปฟังก์ชันการเข้าถึงข้อมูลบนดัชนีบิตแมปแบบเข้ารหัสทั่วไป และดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 117
ภาคผนวก ข 121
ภาคผนวก ค 130
ภาคผนวก ง 139
ประวัติผู้เขียน 145

รายการตาราง

ตาราง		หน้า
2-1	เปรียบเทียบฐานข้อมูลปฏิบัติการและคลังข้อมูล	9
3-1	เปรียบเทียบประสิทธิภาพของดัชนีบิตแมปทั้ง 5 ชนิด	49
4-1	นิยามสัญลักษณ์ที่ใช้ในอัลกอริทึม EncodedBitmapFI	54
5-1	พื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมปทั้ง 6 ชนิด	67
5-2	จำนวนบิตแมปเวกเตอร์ที่ต้องอ่านและจำนวนครั้งในการดำเนินการตรรกะ ระหว่างบิตแมปเวกเตอร์เมื่อมีการสอบถามแบบสมาชิกของดัชนี บิตแมปทั้ง 6 ชนิด	70
5-3	เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150	72
5-4	เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150	73
5-5	เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000	74
5-6	เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000	75
5-7	สรุปลำดับประสิทธิภาพของการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด จากการทดลอง เมื่อ 1 หมายถึงดีที่สุด และ 6 หมายถึงแย่ที่สุด	77
5-8	จำนวนการสอบถาม (เปอร์เซ็นต์) ในแต่ละการทดลองที่ดัชนีบิตแมปแบบเข้ารหัส ที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามน้อยที่สุด	79
5-9	เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150	80
5-10	เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150	81
5-11	เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000	82
5-12	เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000	83

รายการตาราง (ต่อ)

ตาราง **หน้า**

6-1 ลักษณะที่สำคัญของดัชนีบิตแมปทั้ง 6 ชนิด 91

6-2 สรุปประสิทธิภาพที่ต้องการสำหรับการสอบถามแบบสมาชิก และดัชนีบิตแมปที่ควรเลือกใช้ 93

รายการภาพประกอบ

ภาพประกอบ **หน้า**

2-1 สถาปัตยกรรมของคลังข้อมูล 11
2-2 ตัวอย่างดัชนีบิตแมปแบบ B-Tree 13
2-3 อัลกอริทึม Apriori 15
2-4 ฟังก์ชัน Apriori-gen 16
2-5 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Apriori 16
2-6 อัลกอริทึม Create FP-tree 20
2-7 ฐานข้อมูล ซึ่งคอลัมน์ทางขวาสุด แสดง Frequent items ที่เรียงลำดับตามค่าความถี่จากมากไปน้อยของแต่ละทรานแซคชัน 20
2-8 ตัวอย่างการสร้าง FP-tree 21
2-9 อัลกอริทึม FP-growth 22
2-10 Conditional pattern base และ Conditional FP-tree ของแต่ละ Item 23
2-11 Frequent itemsets ทั้งหมด 24
2-12 อัลกอริทึม Closed 25
2-13 ฟังก์ชัน Gen-Closure 25
2-14 ฟังก์ชัน Gen-Generator 26
2-15 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Closed 27
2-16 อัลกอริทึมในการหา Index array 29
2-17 อัลกอริทึม Index-BitTableFI 30
2-18 ตัวอย่างการหา Frequent itemsets ด้วยอัลกอริทึม Index-BitTableFI 31
3-1 ตัวอย่างการทำดัชนีบิตแมปแบบพื้นฐานบนแอทริบิวต์ *X* ของตาราง *T* 33
3-2 ขั้นตอนการบีบอัดแบบ WAH 35
3-3 ดัชนีบิตแบบพื้นฐานแมปบนแอทริบิวต์เพศ 36
3-4 The modified run-length encoding 37
3-5 Huffman tree 37
3-6 Huffman code 37
3-7 บิตแมปที่ได้จากการบีบอัดแบบ RLH 38

รายการภาพประกอบ (ต่อ)

ภาพประกอบ **หน้า**

3-8 ตัวอย่างการทำดัชนีบิตแมปแบบช่วงบนแอทริบิวต์ *X* ของตาราง *T*.....39

3-9 ตัวอย่างการทำดัชนีบิตแมปแบบกระจายบนแอทริบิวต์ *X* ของตาราง *T*.....42

3-10 ตัวอย่างการทำดัชนีบิตแมปแบบคู่กันบนแอทริบิวต์ *X* ของตาราง *T*.....44

3-11 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ *X* ของตาราง *T*.....46

3-12 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ *X* ของตาราง *T* โดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย.....47

3-13 แผนภาพการลงรหัสของดัชนีบิตแมปทั้ง 5 ชนิด เมื่อ *C* = 8.....48

4-1 ขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส.....51

4-2 ตัวอย่างรายการสอบถาม (Workload).....52

4-3 ตัวอย่างตารางค่าของแอทริบิวต์ *X* ที่สกัดจาก Workload.....52

4-4 อัลกอริทึม EncodedBitmapFI.....53

4-5 ตัวอย่างตาราง BitMatrix.....55

4-6 ค่าความถี่ของแต่ละ Item.....55

4-7 ตาราง BitMatrix เมื่อผ่านการเรียงลำดับตามค่าความถี่ของแต่ละ Item.....56

4-8 ตาราง BitMatrix เมื่อเลือก Item ที่มีค่าความถี่น้อยกว่าค่าความถี่ขั้นต่ำออก.....56

4-9 ตาราง BitMatrix เมื่อเลือก {A,C,E,G,H,J K,O} ออก.....60

4-10 กลุ่มข้อมูลที่ปรากฏบ่อยขนาด 2^i.....60

4-11 ตารางเทียบค่า.....62

4-12 เปรียบเทียบฟังก์ชันการเข้าถึงข้อมูลที่ผ่านการลดรูป เมื่อใช้ตารางเทียบค่า 4-11(ก) และ 4-11(ข).....64

5-1 แผนภาพการลงรัหสดัชนีบิตแมปทั้ง 6 ชนิด เมื่อคาร์ดินอร์ลิตี้เท่ากับ 8.....66

5-2 กราฟเปรียบเทียบพื้นที่ที่ใช้ในการสร้างดัชนีทั้ง 6 ชนิด เมื่อแอทริบิวต์ที่นำมาสร้างดัชนีมี 1,000,000 เรคอร์ด (*N* = 1,000,000).....68

5-3 กราฟเปรียบเทียบพื้นที่ที่ใช้ในการสร้างดัชนีทั้ง 4 ชนิด เมื่อแอทริบิวต์ที่นำมาสร้างดัชนีมี 1,000,000 เรคอร์ด (*N* = 1,000,000).....65

รายการภาพประกอบ (ต่อ)

ภาพประกอบ **หน้า**

5-4 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150 72

5-5 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150 73

5-6 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000 74

5-7 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000 75

5-8 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิก บนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย ของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150 80

5-9 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิก บนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย ของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150 81

5-10 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิก บนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย ของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000 82

5-11 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิก บนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย ของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000 83

5-12 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมป และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 150 85

5-13 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมป และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 150 86

รายการภาพประกอบ (ต่อ)

ภาพประกอบ **หน้า**

5-14 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมป
และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด
บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 1,000 87

5-15 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมป
และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด
บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 1,000.....88

บทที่ 1

บทนำ

สำหรับบทนำนี้ จะกล่าวถึง ความเป็นมาของงานวิจัย การตรวจสอบเอกสารและงานวิจัยที่เกี่ยวข้อง วัตถุประสงค์ของการวิจัย วิธีการดำเนินการวิจัย ขอบเขตการวิจัย ขั้นตอนการดำเนินการวิจัย ระยะเวลาดำเนินงานและแผนการดำเนินงาน เครื่องมือและอุปกรณ์ที่ใช้ในการวิจัย และสุดท้าย ประโยชน์ที่คาดว่าจะได้รับจากการดำเนินการวิจัย

1.1 ความเป็นมา

ในระบบคลังข้อมูล การสอบถามข้อมูลเป็นแบบ Online Analytical Processing (OLAP) (Chauhuri and Dayal, 1997; Han and Kamber, 2000) โดยเป็นการสอบถามข้อมูลเพื่อช่วยในการตัดสินใจของผู้บริหาร ซึ่งการสอบถามมีความซับซ้อน (Complex query) และเป็นแบบทันทีทันใด (Adhoc) (Chauhuri and Dayal, 1997; Bontempo and Saracco, 2002) ข้อมูลที่เก็บในคลังข้อมูลเป็นข้อมูลที่มาจากหลาย ๆ แหล่ง และเป็นข้อมูลที่เก็บตั้งแต่อดีตจนถึงปัจจุบัน ข้อมูลที่อยู่ในคลังข้อมูลจึงมีปริมาณมาก ทำให้การประมวลผลในการสอบถามข้อมูลต้องใช้เวลามากขึ้น

การเพิ่มความเร็วในการค้นหาข้อมูลมีหลายวิธี เช่น การประมวลผลแบบคู่ขนาน (Chauhuri and Dayal, 1997; Bontempo and Saracco, 2002) การเพิ่มสมรรถนะเครื่อง (Chan and Bontempo, 1998) หรือการทำดัชนี (Chauhuri and Dayal, 1997; Chan and Bontempo, 1998; Intelligent Enterrice, 1999; Bontempo and Saracco, 2002; Korth and Sudarshan, 2006) ซึ่งการทำดัชนีเป็นเทคนิคการเพิ่มความเร็วที่ไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ สำหรับระบบคลังข้อมูลดัชนีที่นิยมใช้คือ ดัชนีแบบบิตแมป (Bitmap index) (Chan and Bontempo, 1998; Wu and Buchmann, 1998; Chan and Ioannidis, 1999; Stockinger and Wu, 2007) เนื่องจากสามารถดำเนินการระดับบิต (Bit operation) เช่น AND OR XOR และ NOT ระหว่างบิตแมปเวกเตอร์ก่อนดึงข้อมูลจริง ทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น (Chan and Ioannidis, 1998; Chan and Ioannidis, 1999; Bontempo and Saracco, 2002; Stockinger and Wu, 2007) นอกจากนี้การทำดัชนีแบบบิตแมปยังมีประสิทธิภาพในเรื่องของการใช้พื้นที่ในการจัดเก็บดัชนี โดยเฉพาะบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำ (Cardinality) (คาร์ดินอร์ลิตี้ คือ จำนวนค่าที่แตกต่างกันบนคอลัมน์ที่นำมาทำดัชนี) เนื่องจากการทำดัชนีบิตแมปบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้สูง จะใช้พื้นที่ในการจัดเก็บดัชนีมาก งานวิจัยเกี่ยวกับการทำดัชนีบิตแมปจำนวนมากจึงมุ่งประเด็นไปที่การลดขนาดดัชนี

แต่ยังคงประสิทธิภาพในการสอบถามข้อมูล จากการศึกษาเทคนิคการลดขนาดดัชนีแบ่งได้เป็น 2 วิธี คือเทคนิคแบบบีบอัดดัชนี (Compression bitmap index) (Stockinger and Wu, 2007; Elizabeth and Patrick, 2007; Wu *et al*., 2002; Stabno and Wrembe, 2007l) และเทคนิคแบบไม่บีบอัดดัชนี (Uncompression bitmap index) (Wu and Buchmann, 1998; Chan and Y. E. Ioannidis, 1999; Koudas, 2000; Wattanakitrungroj and Vanichayobon, 2006; Vanichayobon *et al*., 2006) สำหรับในงานวิทยานิพนธ์นี้จะมุ่งประเด็นไปที่การทำดัชนีแบบไม่มีการบีบอัดดัชนี เนื่องจากการดำเนินการตรรกะระดับบิตบนบิตแมปที่มีการบีบอัดดัชนีไม่สามารถทำได้โดยตรง

เทคนิคการทำดัชนีบิตแมปแบบเข้ารหัส (Wu and Buchmann, 1998) เป็นเทคนิคการทำดัชนีบิตแมปแบบไม่บีบอัดดัชนีที่ใช้พื้นที่น้อยที่สุด (Wattanakitrungroj and Vanichayobon, 2006; Vanichayobon *et al*., 2006) กล่าวคือมีการใช้ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์ (เมื่อ C คือค่าคาร์ดินอร์ลิตี้) ขณะที่เทคนิคการทำดัชนีบิตแมปแบบพื้นฐานใช้ C บิตแมปเวกเตอร์ แต่ในการสอบถามข้อมูลแบบค่าเท่ากันบนดัชนีบิตแมปแบบเข้ารหัสนั้นต้องตรวจสอบทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) และหากเป็นการสอบถามข้อมูลแบบสมาชิกสามารถทำได้โดยหาคำตอบของแต่ละค่าแล้วนำมาดำเนินการตรรกะ OR ทำให้ใช้เวลาในการสอบถามข้อมูลเพิ่มขึ้น ดังนั้นหากเราสามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามแบบสมาชิกได้ โดยไม่ต้องตรวจสอบทุกบิตแมปเวกเตอร์ของแต่ละค่าที่ต้องการสอบถาม จะทำให้เวลาที่ใช้ในการสอบถามน้อยลง กล่าวคือทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น

หัวใจสำคัญในการทำดัชนีบิตแมปแบบเข้ารหัสคือ การเข้ารหัส (Encoding) ซึ่งการเข้ารหัสที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามให้เหลือน้อยที่สุดถือเป็นการเข้ารหัสในรูปแบบที่ดี ทำให้ประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกเพิ่มขึ้น วิทยานิพนธ์นี้จึงนำเสนอแนวคิดการเพิ่มประสิทธิภาพด้านเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแบบเข้ารหัส โดยใช้กลุ่มข้อมูลที่ปรากฏบ่อยในที่นี้หมายถึงค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันบ่อย มาช่วยในการเข้ารหัสค่าของแอทริบิวต์ที่มีโอกาสถูกสอบถามด้วยกันบ่อย ให้อยู่ในรูปแบบที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบได้เมื่อมีการสอบถามแบบสมาชิกด้วยกัน

1.2 งานวิจัยที่เกี่ยวข้อง

Bitmap Index Design and Evaluation

งานวิจัยนี้ (Chan and Ioannidis, 1998) กล่าวถึงการทำดัชนีบิตแมปแบบพื้นฐาน (Simple Bitmap Index) ซึ่งใช้จำนวนบิตแมปเวกเตอร์เท่ากับจำนวนค่าแอทริบิวต์ที่

นำมาสร้างดัชนี ดังนั้นการทำดัชนีแบบบิตแมปจึงเหมาะกับแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำ แต่เมื่อแอทริบิวต์ที่นำมาสร้างดัชนีมีค่าคาร์ดินอร์ลิตี้สูง ทำให้ต้องใช้พื้นที่ในการสร้างดัชนีมากขึ้น และได้นำเสนอกรอบแนวคิดในการออกแบบดัชนีบิตแมปและขั้นตอนวิธีการสร้างดัชนีบิตแมปที่มีประสิทธิภาพด้านพื้นที่ (Space) ที่ใช้จัดเก็บดัชนี เวลา (Time) ที่ใช้สอบถามข้อมูล การแลกเปลี่ยนระหว่างพื้นที่และเวลา (Space-Time trade off)

An Efficient Bitmap Encoding Scheme for Selection Queries

งานวิจัยนี้ (Chan and Ioannidis, 1999) กล่าวถึงเทคนิคการทำดัชนีบิตแมปแบบแถว (Range Bitmap Index) และนำเสนอเทคนิคการทำดัชนีบิตแมปแบบช่วง (Interval Bitmap Index) ที่มีการใช้พื้นที่ในการจัดเก็บดัชนีเป็นครึ่งหนึ่งของดัชนีบิตแมปแบบพื้นฐาน ซึ่งมีการใช้ $\lceil C/2 \rceil$ บิตแมปเวกเตอร์ แต่ในการสอบถามข้อมูลใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ต่อการสอบถามหนึ่งค่า จึงมีการใช้เวลาในการสอบถามข้อมูลมากกว่าดัชนีบิตแมปแบบพื้นฐาน

Scatter Bitmap: Space-time Efficient Bitmap Indexing for Equality and Membership Queries

งานวิจัยนี้ (Vanichayobon *et al*., 2006) กล่าวถึงเทคนิคการทำดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง และดัชนีบิตแมปแบบเข้ารหัส และนำเสนอเทคนิคการทำดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index) ที่ใช้พื้นที่ในการจัดเก็บดัชนีน้อยกว่าดัชนีบิตแมปแบบช่วง นั่นคือมีการใช้ $\lceil 2\sqrt{C} \rceil$ บิตแมปเวกเตอร์ และในการสอบถามข้อมูลยังใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ต่อการสอบถามหนึ่งค่า

Dual Bitmap Index: Space-Time Efficient Bitmap Index for Equality and Membership Queries

งานวิจัยนี้ (Wattanakitrungroj and Vanichayobon, 2006) กล่าวถึงเทคนิคการทำดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบเข้ารหัส และนำเสนอเทคนิคการทำดัชนีบิตแมปแบบคู่กัน (Dual Bitmap Index) ที่มีการใช้พื้นที่ในการจัดเก็บดัชนีน้อยกว่าดัชนีบิตแมปแบบกระจาย นั่นคือมีการใช้ $\lceil \sqrt{2C + 0.25} + 0.5 \rceil$ บิตแมปเวกเตอร์ และในการสอบถามข้อมูลยังใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ต่อการสอบถามหนึ่งค่า

Encoded Bitmap Indexing for Data Warehouses

งานวิจัยนี้ (Wu and Buchmann, 1998) กล่าวถึงเทคนิคการทำดัชนีบิตแมปแบบพื้นฐาน และนำเสนอเทคนิคการทำดัชนีบิตแมปแบบเข้ารหัส (Encoded Bitmap Index) ที่มีการใช้พื้นที่ในการจัดเก็บดัชนีน้อยกว่าการทำดัชนีบิตแมปแบบอื่น นั่นคือมีการใช้เพียง $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์ สามารถทำดัชนีบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้สูงได้ แต่เวลาที่ใช้

ในการสอบถามข้อมูลมากกว่าดัชนีบิตแมปแบบอื่น เพราะต้องตรวจสอบทุกบิตแมปเวกเตอร์ และมีการตรวจสอบกับตารางเทียบค่าต่อการสอบถามหนึ่งค่า

1.3 วัตถุประสงค์

เพื่อคิดค้นเทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส โดยใช้กลุ่มข้อมูลที่ปรากฏบ่อยสำหรับการสอบถามแบบสมาชิก ที่มีประสิทธิภาพด้านการแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่กับเวลา (Space-Time Trade-off) สำหรับการสอบถามแบบสมาชิกดีกว่าดัชนีบิตแมปที่เคยมีมา

1.4 วิธีการดำเนินการวิจัย

1. ศึกษาแนวคิดที่เกี่ยวข้องกับระบบคลังข้อมูลและดัชนีบิตแมปทั้ง 5 ชนิด ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัส

2. วิเคราะห์และออกแบบเทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย

3. กำหนดรูปแบบการประเมินประสิทธิภาพของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย เปรียบเทียบกับดัชนีบิตแมปทั้ง 5 ชนิด ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัส โดยทำการประเมินประสิทธิภาพทั้งในด้านการใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนีและเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิก (Membership query) ดังขั้นตอนต่อไปนี้

3.1 ทำการศึกษาข้อมูลสำหรับใช้ในการทดสอบ ซึ่งเป็นข้อมูลมาตราฐาน TPC-H Benchmark (Transaction Processing Performance Council, 2006)

3.2 จัดเตรียมข้อมูลทดสอบ โดยเลือกเฉพาะแอทริบิวต์ที่จะนำมาทำดัชนี และเปลี่ยนรูปค่าข้อมูลเป็นจำนวนเต็มที่เรียงต่อเนื่องกัน โดยเริ่มจาก 0

3.3 ออกแบบขั้นตอนวิธีในการสร้างดัชนีบิตแมปทั้ง 6 ชนิด ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน ดัชนีบิตแมปแบบเข้ารหัส และดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

3.4 พัฒนาโปรแกรมเพื่อสร้างดัชนีทั้ง 6 ชนิด ตามขั้นตอนที่ออกแบบไว้ตามข้อ 3.3 โดยใช้ตัวแปลภาษาซี (C Compiler)

4. ออกแบบขั้นตอนวิธีการสอบถามข้อมูลแบบสมาชิกบนดัชนีบิตแมปทั้ง 6 ชนิด ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนี

บิตแมปแบบคู่กัน ดัชนีบิตแมปแบบเข้ารหัส และดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

5. พัฒนาโปรแกรมเพื่อสอบถามข้อมูลแบบสมาชิกบนดัชนีบิตแมปทั้ง 6 ชนิด ตามขั้นตอนที่ออกแบบไว้ในขั้นตอนที่ 4 โดยใช้ตัวแปลภาษาซี

6. ประเมินประสิทธิภาพการใช้พื้นที่ในการจัดเก็บดัชนี (Space) ของดัชนีบิตแมปทั้ง 6 ชนิด ด้วยการเขียนกราฟเปรียบเทียบพื้นที่ที่ใช้ในการจัดเก็บดัชนีแต่ละชนิด

7. ประเมินประสิทธิภาพการใช้เวลาในการสอบถามข้อมูลแบบสมาชิก (Time) บนดัชนีบิตแมปทั้ง 6 ชนิด ด้วยการรันโปรแกรมที่พัฒนาขึ้นตามข้อ 5 แล้วบันทึกเวลาในการสอบถามของดัชนีบิตแมปทั้ง 6 ชนิด

8. ประเมินเกี่ยวกับการแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่กับเวลา (Space-Time Trade-off) สำหรับการสอบถามแบบสมาชิกบนดัชนีบิตแมปแต่ละชนิด

9. วิเคราะห์และสรุปผลการประเมินประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย กับดัชนีเปรียบเทียบทั้ง 5 ชนิด

1.5 ขอบเขตการวิจัย

1. ศึกษาดัชนีบิตแมปที่มีอยู่ ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัส

2. วิเคราะห์และออกแบบเทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิกโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย

3. ประเมินประสิทธิภาพของการใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนี และเวลาในการสอบถามแบบสมาชิก บนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยกับดัชนีเปรียบเทียบ

1.6 ขั้นตอนการดำเนินงาน

1. ศึกษางานวิจัยและเอกสารที่เกี่ยวข้องและกำหนดขอบเขตของปัญหา

2. วิเคราะห์และออกแบบเทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส

3. กำหนดรูปแบบการประเมินประสิทธิภาพ

4. ศึกษาและวิเคราะห์หาเครื่องมือสำหรับประเมินประสิทธิภาพ

5. พัฒนาดัชนีบิตแมปแบบเข้ารหัสตามที่ได้ออกแบบไว้

6. ประเมินประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยกับดัชนีเปรียบเทียบ

7. สรุปและวิเคราะห์ผลการประเมินประสิทธิภาพกับดัชนีเปรียบเทียบ

8. จัดทำเอกสารประกอบการวิจัย

1.7 ระยะเวลาดำเนินงานและแผนการดำเนินงาน

พฤศจิกายน 2550 – กุมภาพันธ์ 2552

ขั้นตอนการดำเนินงาน	เดือน															
	2550		2551												2552	
	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2
1. ศึกษางานวิจัยและเอกสารที่เกี่ยวข้อง																
2. วิเคราะห์และออกแบบเทคนิค																
3. กำหนดรูปแบบการประเมินประสิทธิภาพ																
4. ศึกษาเครื่องมือสำหรับใช้ประเมินประสิทธิภาพ																
5. พัฒนาดัชนีบิตแมปแบบเข้ารหัสตามที่ได้ออกแบบไว้																
6. ประเมินประสิทธิภาพกับดัชนีเปรียบเทียบ																
7. สรุปและวิเคราะห์ผลการประเมิน ประสิทธิภาพกับดัชนีเปรียบเทียบ																
8. จัดทำเอกสารประกอบการวิจัย																

1.8 เครื่องมือและอุปกรณ์

ด้านฮาร์ดแวร์

1. เครื่องคอมพิวเตอร์จำนวน 1 เครื่อง
2. หน่วยความจำ 1 GB
3. ฮาร์ดดิสก์ 150 GB

ด้านซอฟต์แวร์

1. ระบบปฏิบัติการ Linux Red Hat 9.0
2. ตัวแปลภาษาซี (C Compiler) สำหรับใช้ในการพัฒนาโปรแกรม เพื่อวัดประสิทธิภาพของเวลาที่ใช้ในการสอบถามข้อมูล

1.9 ประโยชน์ที่คาดว่าจะได้รับ

ได้เทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส โดยใช้กลุ่มข้อมูลที่ปรากฏบ่อยสำหรับการสอบถามแบบสมาชิก ที่มีประสิทธิภาพด้านการแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่กับเวลา (Space-Time Trade-off) สำหรับการสอบถามแบบสมาชิกดีกว่าดัชนีบิตแมปที่เคยมีมา

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องกับงานวิทยานิพนธ์นี้ ประกอบด้วย คลังข้อมูล (Data warehouse) การสร้างดัชนี (Indexing) เทคนิคการทำดัชนีแบบบิตแมป (Bitmap indexing) และกลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets) สำหรับบทนี้จะกล่าวถึง คลังข้อมูล การสร้างดัชนี และกลุ่มข้อมูลที่ปรากฏบ่อย ส่วนเทคนิคการทำดัชนีแบบบิตแมปจะกล่าวในบทถัดไป

2.1 คลังข้อมูล

2.1.1 ความหมายของคลังข้อมูล

คลังข้อมูล หมายถึง ฐานข้อมูลขนาดใหญ่ขององค์กรหรือหน่วยงานหนึ่งๆ ซึ่งเก็บรวบรวมข้อมูลจากฐานข้อมูลปฏิบัติการ (Operational database) และแหล่งข้อมูลภายนอกองค์กร (External sources) รวมทั้งจากไฟล์ธรรมดาทั่วไป ข้อมูลที่ถูกจัดเก็บในคลังข้อมูลนั้นมีวัตถุประสงค์ในการนำมาใช้งานและมีลักษณะของการจัดเก็บแตกต่างไปจากข้อมูลในฐานข้อมูลระบบงานอื่น โดยข้อมูลในคลังข้อมูลจะถูกนำมาใช้เพื่อการวิเคราะห์ และสนับสนุนการตัดสินใจบริหารงานของผู้บริหาร (Chaudhuri and Dayal, 1997; Han and Kamber, 2000; เบญจมาศ และ ดร.ภัทรชัย, 2002)

2.1.2 คุณลักษณะของคลังข้อมูล

จากความหมายของคลังข้อมูลที่บอกถึงความแตกต่างกันระหว่างคลังข้อมูลกับฐานข้อมูลปฏิบัติการ สามารถสรุปคุณลักษณะของคลังข้อมูลได้ดังนี้ (Han and Kamber, 2000; เบญจมาศ และ ดร.ภัทรชัย, 2002)

1. Subject Oriented หมายถึง คลังข้อมูลถูกออกแบบมาเพื่อมุ่งเน้นไปในเนื้อหาที่สนใจ เช่น ลูกค้า สินค้า การขาย เป็นต้น ไม่ได้เน้นไปที่การทำงานหรือกระบวนการแต่ละอย่างโดยเฉพาะเหมือนอย่างฐานข้อมูลปฏิบัติการ ในส่วนของรายละเอียดข้อมูลที่จัดเก็บในระบบทั้งสองแบบก็จะแตกต่างกันไปตามความต้องการใช้งานด้วยเช่นกัน คลังข้อมูลจะไม่เก็บข้อมูลที่ไม่มีส่วนเกี่ยวข้องกับการประมวลผลเพื่อสนับสนุนการตัดสินใจ การวิเคราะห์ และการทำเหมืองข้อมูล (Data mining)

2. Integrated หมายถึง การรวบรวมข้อมูลจากหลายแหล่งข้อมูลเข้าด้วยกัน ซึ่งข้อมูลอาจมาจากระบบที่แตกต่างกัน รวมทั้งอาจมีการจัดเก็บข้อมูลในรูปแบบที่แตกต่างกัน หรืออาจมีความแตกต่างทางด้านแพล็ตฟอร์ม (Platform) ดังนั้นจึงต้องทำให้ข้อมูลมีความ

สอดคล้องและเป็นมาตรฐานเดียวกัน เช่น กำหนดให้ข้อมูลเดียวกันมีรูปแบบเดียวกันทั้งหมด และมีค่าอยู่ในช่วงเดียวกัน เป็นต้น

3. *Time Variant* หมายถึง ข้อมูลในคลังข้อมูล จะต้องจัดเก็บโดยกำหนดช่วงเวลาเอาไว้ เพราะในการตัดสินใจด้านการบริหารจำเป็นต้องมีข้อมูลเปรียบเทียบในแต่ละช่วงเวลา โดยข้อมูลที่เก็บในคลังข้อมูลจะเป็นข้อมูลในอดีตจนถึงปัจจุบันย้อนหลัง 5-10 ปี

4. *Nonvolatile* หมายถึง ข้อมูลในคลังข้อมูลจะไม่เปลี่ยนแปลงบ่อย โดยจะมีการเข้าถึงข้อมูล 2 แบบ คือ การโหลดข้อมูลและการเข้าถึงข้อมูลเท่านั้น

2.1.3 ความแตกต่างระหว่างฐานข้อมูลปฏิบัติการและคลังข้อมูล

การทำงานของระบบฐานข้อมูลปฏิบัติการ (Operation database) คือ การประมวลผลทรานแซคชันและการประมวลผลแบบออนไลน์ เรียกระบบนี้ว่า OLTP (On-Line Transaction Processing) เป็นระบบที่ครอบคลุมการดำเนินการประจำวันขององค์กร เช่น การจัดซื้อ การขาย การลงทะเบียน การธนาคาร เป็นต้น ส่วนการทำงานของระบบคลังข้อมูล คือ การวิเคราะห์ข้อมูลและสนับสนุนการตัดสินใจของผู้บริหาร ซึ่งเรียกระบบนี้ว่า OLAP (On-Line Analytical Processing) การเปรียบเทียบความแตกต่างระหว่างระบบฐานข้อมูลปฏิบัติการและคลังข้อมูลแบ่งตามลักษณะต่าง ๆ พิจารณาได้ดังตาราง 2-1 (Chaudhuri and Dayal, 1997; Han and Kamber, 2000; เบญจมาศ และ ดร.ภัทรชัย, 2002)

ตาราง 2-1 เปรียบเทียบฐานข้อมูลปฏิบัติการและคลังข้อมูล

ลักษณะ	ฐานข้อมูลปฏิบัติการ	คลังข้อมูล
การประมวลผล	- แบบ OLTP	- แบบ OLAP
การกำหนดทิศทาง	- มุ่งเน้นไปที่ลูกค้าเป็นสำคัญ	- มุ่งเน้นด้านการตลาด - หัวเรื่องในการวิเคราะห์เป็นสำคัญ
ผู้ใช้ระบบ	- เสมียน นักธุรกิจ ผู้เชี่ยวชาญทางด้านฐานข้อมูล	- คนที่ทำงานเกี่ยวกับความรู้ เช่น ผู้จัดการ ผู้บริหาร นักวิเคราะห์

ตาราง 2-1 เปรียบเทียบฐานข้อมูลปฏิบัติการและคลังข้อมูล (ต่อ)

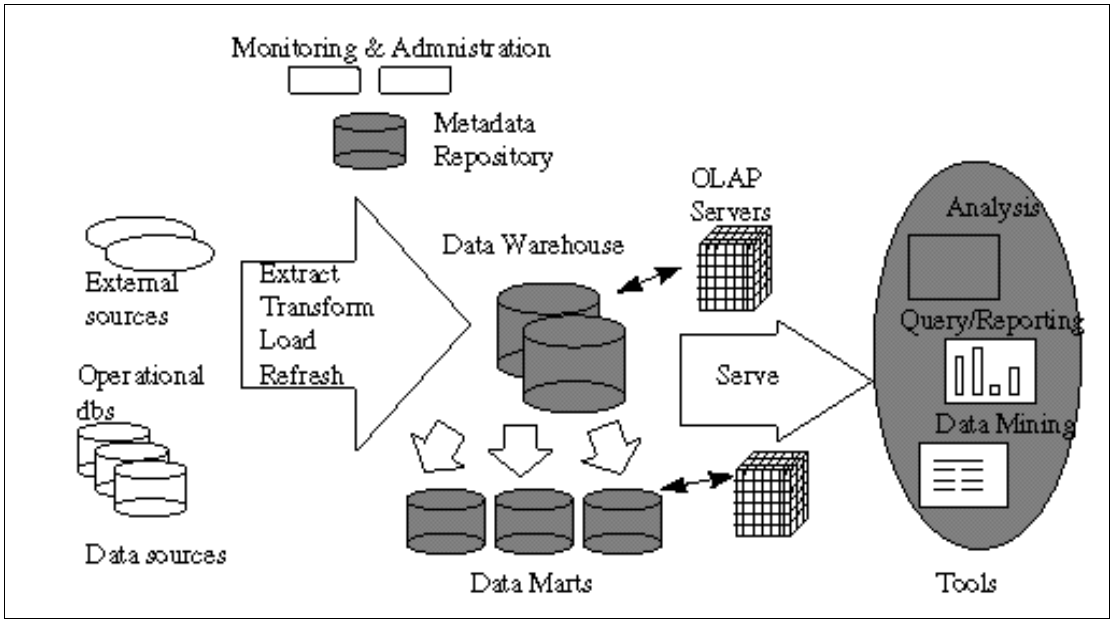
ลักษณะ	ฐานข้อมูลปฏิบัติการ	คลังข้อมูล
ข้อมูล	- เป็นข้อมูลปัจจุบันที่ทันสมัย - ข้อมูลมีการเปลี่ยนแปลงได้ - มุ่งประเด็นนำข้อมูลดิบ (Data) เข้า	- เป็นข้อมูลในอดีตจนถึงปัจจุบันที่มีความถูกต้องแม่นยำในช่วงเวลานาน ๆ - ข้อมูลไม่มีการเปลี่ยนแปลง และมุ่งประเด็นนำข้อมูลข่าวสาร (Information) ออก
การสรุปความ	- ง่าย ๆ ไม่ซับซ้อน - ต้องการรายละเอียดมาก - มีความถูกต้องแม่นยำในขณะที่มีการเข้าถึง	- มีการสรุปข้อมูลในอดีตไว้ให้ใช้ - มีความมั่นคง - สามารถแสดงข้อมูลในอดีตได้อย่างรวดเร็ว
การมอง (View)	- แสดงรายละเอียดโดยที่โครงสร้างของข้อมูลไม่มีการเปลี่ยนแปลง แต่ข้อมูลที่อยู่ข้างในสามารถเปลี่ยนแปลงได้	- มีการสรุปไว้ในลักษณะหลายมิติ ซึ่งโครงสร้างมีความยืดหยุ่นได้
การสอบถาม	- เป็นการสอบถามแบบสั้น ๆ ง่าย ๆ	- เป็นการสอบถามที่ซับซ้อน
การเข้าถึง	- อ่าน/เขียน - จำนวนเรคอร์ดที่ถูกเข้าถึงเป็นสิบ - จำนวนผู้เข้าถึงข้อมูลเป็นพัน	- อ่านเป็นส่วนมาก - จำนวนเรคอร์ดที่ถูกเข้าถึงเป็นล้าน - จำนวนผู้เข้าถึงข้อมูลเป็นร้อย
ขนาดของฐานข้อมูล	- 100 MB ถึง GB	- 100 GB ถึง TB
มาตรวัดประสิทธิภาพ	- ปริมาณทรานแซคชัน	- ปริมาณการสอบถาม - เน้นเวลาในการตอบสนอง

จะเห็นว่า ฐานข้อมูลปฏิบัติการและคลังข้อมูลมีความแตกต่างกันหลายประการ เหตุผลหลักที่ต้องแยกสองระบบนี้ออกจากกันคือ เพื่อเพิ่มความสามารถในการการทำงานของทั้งสองระบบ เนื่องจากความแตกต่างทางด้านโครงสร้างและสิ่งที่บรรจุอยู่ข้างในและ

วัตถุประสงค์ที่แตกต่างกันตามที่กล่าวมา ซึ่งหากเรามีการดำเนินการวิเคราะห์ข้อมูลแบบออนไลน์โดยตรงบนฐานข้อมูล พร้อมกับการดำเนินการประมวลผลทรานแซคชันบนฐานข้อมูลปฏิบัติการแล้ว จะก่อให้เกิดความเสียหายอย่างรุนแรง และเป็นการลดประสิทธิภาพของทั้งระบบ OLTP และ OLAP ด้วย

2.1.4 สถาปัตยกรรมของคลังข้อมูล (Data Warehouse Architecture)

DWA (Data Warehouse Architecture) เป็นโครงสร้างมาตรฐานที่ใช้บ่อย ซึ่งโดยทั่วไปแล้วคลังข้อมูลแต่ละระบบอาจจะมีรูปแบบที่ไม่เหมือนกันได้ เพื่อให้เหมาะสมกับองค์กรนั้นๆ ทั้งนี้ส่วนประกอบต่างๆ ภายใน DWA ที่สำคัญ แสดงดังภาพประกอบ 2-1 ได้แก่ แหล่งข้อมูล หน่วยเก็บข้อมูล OLAP Server และ เครื่องมือสำหรับผู้ใช้ ดังนี้



ภาพประกอบ 2-1 สถาปัตยกรรมของคลังข้อมูล (Chaudhuri and Dayal, 1997)

แหล่งข้อมูล (Data Source) โดยแหล่งข้อมูลที่นำมาใช้ในการสร้างคลังข้อมูลมาจากฐานข้อมูลปฏิบัติการ (Operation Database) และแหล่งข้อมูลภายนอก (External Sources) รวมถึงไฟล์ธรรมดาทั่วไป ซึ่งอาจมีรูปแบบที่แตกต่างกัน

หน่วยเก็บข้อมูล (Data Storage) ประกอบด้วย คลังข้อมูล (Data Warehouse) คลังข้อมูลย่อย (Data Mart) และหน่วยจัดการความรู้ (Metadata Repository) ซึ่งหน่วยจัดการความรู้ทำหน้าที่จัดเก็บข้อมูลเกี่ยวกับเครื่องมือที่ใช้ในการควบคุมระบบคลังข้อมูล เช่น ที่มาของข้อมูล รูปแบบของข้อมูล ข้อจำกัดของข้อมูล เวลาที่ปรับปรุงข้อมูลครั้งล่าสุด เป็นต้น

OLAP Server ทำหน้าที่ในการจัดเตรียมการเข้าถึงข้อมูลเพื่อให้การประมวลผลมีความรวดเร็วยิ่งขึ้น ได้แก่ การทำดัชนี การทำ Materialize View การเปลี่ยนรูปแบบการสอบถามที่ซับซ้อน และการประมวลผลแบบขนานเพื่อลดเวลาในการตอบคำถาม โดยคลังข้อมูลอาจใช้มากกว่า 1 Server

เครื่องมือสำหรับผู้ใช้ (Front-End Tools) ประกอบด้วยเครื่องมือในการวิเคราะห์ข้อมูล (Analysis) เครื่องมือสำหรับสอบถาม (Query) เครื่องมือสำหรับออกรายงาน (Report) และเครื่องมือในการทำเหมืองข้อมูล (Data Mining)

ขั้นตอนการสร้างคลังข้อมูล เริ่มจากการดึงข้อมูล (Extraction) จากแหล่งข้อมูล จากนั้นจะเป็นการทำความสะอาดข้อมูล (Data Cleaning) และเปลี่ยนรูปแบบข้อมูล (Transforming) ให้สอดคล้องตรงกัน เนื่องจากแหล่งข้อมูลที่นำมาสร้างคลังข้อมูลเป็นข้อมูลที่มาจากหลายแหล่งที่แตกต่างกัน ทำให้ข้อมูลมีโครงสร้างและรูปแบบที่แตกต่างกัน และมีโอกาสพบข้อมูลที่ผิดพลาด ดังนั้นจึงต้องมีการทำความสะอาดและเปลี่ยนรูปข้อมูลให้อยู่ในรูปแบบที่ตรงกัน เช่น การทำให้ข้อมูลอยู่ในช่วงที่กำหนด การจัดการกับข้อมูลที่สูญหาย การทำให้ข้อมูลมีความสอดคล้องกัน เป็นต้น จากนั้นจึงโหลดข้อมูล (Load) เข้าสู่คลังข้อมูล สำหรับการทำให้ข้อมูลใหม่ (Refresh) นั้นจะขึ้นอยู่กับการควบคุมของผู้ดูแลคลังข้อมูลซึ่งแตกต่างกันไปตามความต้องการ

2.1.5 การประยุกต์ใช้เทคโนโลยีคลังข้อมูล

ในปัจจุบันพบว่า เทคโนโลยีคลังข้อมูลได้ถูกนำไปประยุกต์ใช้ในด้านต่าง ๆ ดังนี้ (Chaudhuri and Dayal, 1997)

- ด้านการผลิต ได้แก่ การวิเคราะห์รายการสินค้าที่ส่ง การสนับสนุนลูกค้า
- ด้านการค้า ได้แก่ การวิเคราะห์ประวัติลูกค้า การวิเคราะห์รายการสินค้า
- ด้านการเงิน ได้แก่ การวิเคราะห์การอ้างสิทธิ์ (Claim) การวิเคราะห์ความเสี่ยง การตรวจจับการโกง
- ด้านการขนส่ง ได้แก่ การวิเคราะห์และจัดการยานพาหนะ
- ด้านการสื่อสาร ได้แก่ การวิเคราะห์การติดต่อ การตรวจจับการโกง
- ด้านการใช้ประโยชน์ให้คุ้มค่า ได้แก่ การวิเคราะห์การใช้พลังงาน
- ด้านการดูแลสุขภาพ ได้แก่ การวิเคราะห์ผลสุขภาพ

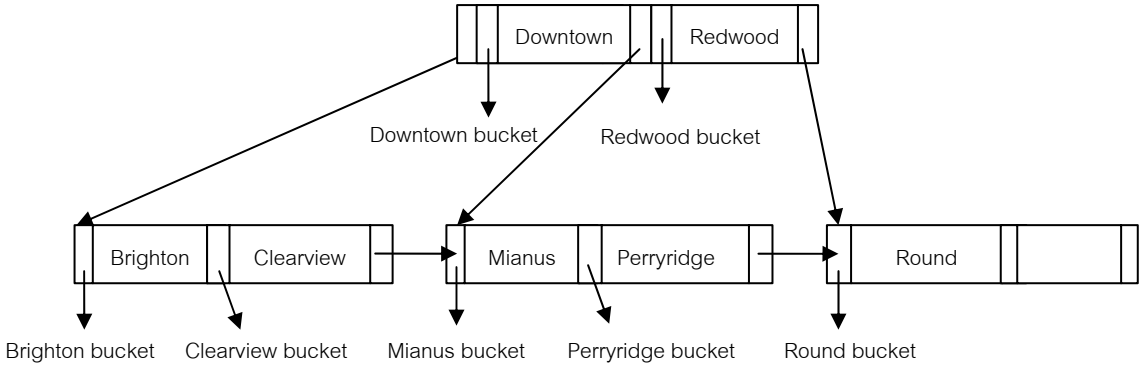
2.2 การสร้างดัชนี (Indexing)

การสร้างดัชนีเป็นเทคนิคการเพิ่มความเร็วในการค้นหาข้อมูลโดยที่ไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ (Intelligent Enterprise, 1999) โดยดัชนีจะถูกเก็บไว้ใน

ไฟล์ที่ประกอบด้วยคีย์ (Key) และที่อยู่ (Address) ที่ทำให้สามารถเข้าถึงเรคอร์ดที่ต้องการได้โดยตรง (Bontempo and Saracco, 2002) ขนาดของดัชนีจะเล็กกว่าข้อมูลจริงที่เก็บไว้มาก จึงเป็นการลดการเข้าถึง I/O ได้มาก เพราะถ้าไม่มีดัชนี ในการเข้าถึงข้อมูลจะต้องอ่านข้อมูลในตารางทั้งหมด เพื่อให้ได้ข้อมูลที่ต้องการ ซึ่งเป็นวิธีที่ช้าและไม่มีประสิทธิภาพ (Chan and Bontempo, 1998; Korth and Sudarshan, 2006) อย่างไรก็ตามการสร้างดัชนีต้องใช้พื้นที่ในการจัดเก็บดัชนี ซึ่งเป็นเรื่องที่ต้องพิจารณาในการจัดสรรทรัพยากรของฐานข้อมูล (Chan and Bontempo, 1998) การทำดัชนีมีหลายวิธี เช่น ดัชนีแบบ B-Tree และดัชนีแบบบิตแมป เป็นต้น ในบทนี้จะกล่าวถึงดัชนีแบบ B-Tree ส่วนดัชนีแบบบิตแมปจะกล่าวถึงในบทต่อไป

2.2.1 ดัชนีแบบ B-Tree

คุณสมบัติทั่วไปของดัชนีแบบ B-Tree คือ โหนดใบทุกโหนดอยู่ในระดับเดียวกัน โหนดภายในที่มี t คีย์ สามารถมีโหนดลูกได้ทั้งหมด t+1 โหนด และอย่างน้อยที่สุดควรมีโหนดลูก (t+1)/2 โหนด (ยกเว้นโหนดรากและโหนดใบ) โดยโหนดรากต้องมีโหนดลูกอย่างน้อย 2 โหนด ทุกคีย์ที่อยู่บนโหนดจะเรียงจากน้อยไปหามากและมีพอยน์เตอร์ระหว่างคีย์ โหนดลูกที่เชื่อมมาจากพอยเตอร์ทางซ้ายของค่าคีย์จะต้องมีค่าน้อยกว่าหรือเท่ากับคีย์นั้น ส่วนโหนดลูกที่เชื่อมมาจากพอยเตอร์ทางขวาของคีย์จะต้องมีค่ามากกว่าคีย์นั้น ความสูงของดัชนีแบบ B-Tree ขึ้นอยู่กับจำนวนคีย์ที่อนุญาตให้มีได้ในแต่ละโหนด ถ้าจำนวนคีย์ทั้งหมดคือ n และแต่ละโหนดมีคีย์ได้ t คีย์ ความสูงของดัชนีแบบ B-Tree ก็จะเท่ากับ $log_t n$ เมื่อมีการค้นหาข้อมูลมาถึงโหนดที่มี t คีย์ ก็จะมีทางเลือกในการค้นหา t+1 เส้นทาง ดังนั้นการค้นหาข้อมูลบนดัชนีแบบ B-Tree จึงใช้เวลาเป็น $O(log_t n)$ ตัวอย่างดัชนีบิตแมปแบบ B-Tree แสดงดังภาพประกอบ 2-2



ภาพประกอบ 2-2 ตัวอย่างดัชนีบิตแมปแบบ B-Tree (Korth and Sudarshan, 2006)

2.2.1.1 ข้อดีของดัชนีแบบ B-Tree

ดัชนีแบบ B-Tree เป็นโครงสร้างข้อมูลที่ออกแบบมาสำหรับการเข้าถึงข้อมูลโดยตรงบนหน่วยความจำสำรอง เช่น แฟ้มข้อมูลและฐานข้อมูลที่เก็บในดิสก์ มีการจัดการเกี่ยวกับ I/O ได้ดี ซึ่งระบบฐานข้อมูลส่วนใหญ่ก็มีการนำดัชนีแบบ B-Tree รวมถึงโครงสร้างอื่นที่อยู่บนพื้นฐานของดัชนีแบบ B-Tree เช่น B+Tree ซึ่งมีการออกแบบให้มีประสิทธิภาพในการสอบถามข้อมูลแบบช่วงได้ดียิ่งขึ้น

2.2.1.2 ข้อจำกัดของดัชนีแบบ B-Tree

ดัชนีแบบ B-Tree ไม่เหมาะสำหรับการสอบถามที่มีความซับซ้อน เนื่องจากจะต้องมีการสร้างคีย์ผสมขึ้นหากเป็นการสอบถามที่ต้องการข้อมูลมากกว่า 1 แอทริบิวต์ ซึ่งมีความยุ่งยากและใช้เวลาในการสอบถามมากขึ้น นอกจากนี้การทำดัชนีแบบ B-Tree ไม่เหมาะกับแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้ต่ำ

2.3 กลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets)

2.3.1 นิยามเทอมพื้นฐาน

- $I = \{i_1, i_2, ..., i_m\}$ หมายถึง เซตของสิ่งที่สนใจ เช่น เซตของตัวอักษรหรือตัวเลข (Literals) หรือแอทริบิวต์ และเรียกแต่ละสมาชิกของ I ว่า Item
- *k-itemsets* หมายถึง เซตของ Items ที่มีจำนวนสมาชิก k Items
- *Database* = $\{ t_1, t_2, ... ,t_n \}$ หมายถึง ฐานข้อมูลซึ่งประกอบด้วยเซตของทรานแซคชัน (Transactions) และแต่ละทรานแซคชัน t_i ประกอบด้วยหมายเลขทรานแซคชัน (*TID*) และเซตของ Item $I_t \subseteq I$
- *TID* หมายถึง หมายเลข/รหัสประจำแต่ละทรานแซคชันที่ไม่ซ้ำกัน (Unique identifier)
- ค่าความถี่ (Support) ของ Itemset A หมายถึง จำนวนทรานแซคชันที่ประกอบด้วย Itemset A ปรากฏอยู่ในฐานข้อมูล เช่น ค่าความถี่ของ Itemset A เท่ากับ 20% หมายถึง ถ้าจำนวนทรานแซคชันทั้งหมดมี 100 ทรานแซคชัน จะมี Itemset A ปรากฏอยู่ 20 ทรานแซคชัน
- ค่าความถี่ขั้นต่ำ (Minimum support) หมายถึง ค่าความถี่ขั้นต่ำที่กำหนดซึ่งจะถูกกำหนดโดยผู้ใช้
- กลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets) หมายถึง กลุ่มข้อมูลที่มีค่าความถี่มากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำที่กำหนด

- Candidate itemsets หมายถึง กลุ่มข้อมูลที่มีโอกาสเป็นกลุ่มข้อมูลที่ปรากฏบ่อย

2.3.2 อัลกอริทึมการหากลุ่มข้อมูลที่ปรากฏบ่อย

การหากลุ่มข้อมูลที่ปรากฏบ่อยเป็นรากฐานและเป็นปัญหาสำคัญในทางด้าน Data mining ในปัจจุบันจึงมีอัลกอริทึมในการหากลุ่มข้อมูลที่ปรากฏบ่อยจำนวนมาก ตัวอย่างอัลกอริทึมที่ได้มีการศึกษา ได้แก่ อัลกอริทึม Apriori อัลกอริทึม FP-growth อัลกอริทึม Closed และ อัลกอริทึม Index-BitTableFI ซึ่งมีรายละเอียดดังนี้

2.3.2.1 อัลกอริทึม Apriori

Apriori (Agrawal *et al*., 1993; Agrawal and Srikant, 1994) เป็นอัลกอริทึมในการหารูปแบบความสัมพันธ์ที่ถูกพัฒนาขึ้นในงานวิจัยรุ่นแรก ๆ ซึ่งเป็นอัลกอริทึมพื้นฐานในการศึกษาและพัฒนาอัลกอริทึมอื่น ๆ อีกมากมาย ขั้นตอนวิธีของ Apriori เป็นวิธีแบบ Generation-and-test โดยจะเริ่มต้นที่การหา Frequent 1-itemsets จากการอ่านฐานข้อมูลรอบแรก และหา Frequent itemsets ในระดับถัด ๆ ไปด้วยการสร้าง Candidate itemsets ที่มีความยาว k+1 จากการรวมกันของ Frequent itemsets ขนาด k และทำการตัด Candidate itemsets ที่มีค่าความถี่น้อยกว่าค่าความถี่ขั้นต่ำออก โดยใช้คุณสมบัติพื้นฐานซึ่งมีหลักการอยู่ว่า ถ้า k-itemset ใด ๆ ไม่เป็น Frequent itemsets แล้ว Itemsets ขนาด k+1 ที่เป็น Superset ของ Itemsets นั้น จะไม่เป็น Frequent itemsets ด้วย (Agrawal and Srikant; 1994)

(1) L_1 = {large 1-itemsets};
(2) for (k = 2; $L_{k-1} \neq \phi$; k++) do begin
(3) C_k = Apriori-gen (L_{k-1}); // New Candidates
(4) forall transactions $t \in D$ do begin
(5) C_t = subset(C_k,t); // Candidates contained in t
(6) forall candidates $c \in C_t$ do
(7) c.count++;
(8) end
(9) L_k = {$c \in C_k$ | c.count $\geq$ minsup}
(10) end
(11) Answer = $\cup_k L_k$;

ภาพประกอบ 2-3 อัลกอริทึม Apriori (Agrawal and Srikant, 1994)

จากอัลกอริทึม Apriori ดังภาพประกอบ 2-3 สามารถหา L_1 (Frequent 1-itemsets) ได้จากการอ่านฐานข้อมูลครั้งที่หนึ่งและทำการนับค่าความถี่ของแต่ละ Item ถ้า Item ใดมีค่าความถี่มากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำก็จะเป็นสมาชิกของ L_1 หลังจากนั้นก็จะทำการหา L_2 , L_3 , L_4 , ..., L_k เมื่อ k คือขนาดของ Itemsets ที่ใหญ่ที่สุดที่ผ่านค่าความถี่ขั้นต่ำ ในการหา L_k (Frequent k-itemsets) สามารถทำได้โดยการสร้าง C_k (Candidate k-itemsets) จากการรวมกันของ L_{k-1} ด้วยฟังก์ชัน Apriori-gen ดังภาพประกอบ 2-4 และทำการนับค่าความถี่ของ C_k ที่สร้างได้ แล้วเลือก C_k ที่มีค่าความถี่มากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำให้เป็นสมาชิกของ L_k

(1) Insert into C_k
(2) Select $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_{k-1}$, $q.\text{item}_{k-1}$
(3) From L_{k-1} p, L_{k-1} q
(4) Where $p.\text{item}_1 = q.\text{item}_1$, ..., $p.\text{item}_{k-2} = q.\text{item}_{k-2}$, $p.\text{item}_{k-1} < q.\text{item}_{k-1}$

ภาพประกอบ 2-4 ฟังก์ชัน Apriori-gen (Agrawal and Srikant, 1994)

การทำงานของฟังก์ชัน Apriori-gen โดย C_k เกิดจากการรวมกันของ L_{k-1} มีเงื่อนไขว่าสมาชิกทุกตัวต้องมีสมาชิกเหมือนกันยกเว้นตัวสุดท้าย และสมาชิกจะต้องมีการเรียงลำดับด้วย เช่น ให้ L_3 = {{123}, {124}, {134}, {135}, {234}} จะได้ C_4 = {{1234}, {1345}} เป็นต้น

ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Apriori แสดงดังภาพประกอบ 2-5 เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2 (50%)

Database

TID	**Items**
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

→

C_1

Itemsets	**Support**
1	2
2	3
3	3
4	1
5	3

(ก) หาค่าความถี่ของแต่ละ Item

ภาพประกอบ 2-5 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Apriori

C_1

Itemsets	Support
1	2
2	3
3	3
4	1
5	3

→

L_1

Itemsets	Support
1	2
2	3
3	3
5	3

(ข) หา Frequent 1-itemsets

L_1

Itemsets	Support
1	2
2	3
3	3
5	3

→

C_2

Itemsets	Support
1, 2	1
1, 3	2
1, 5	1
2, 3	2
2, 5	3
3, 5	2

(ค) สร้าง Candidate 2-itemsets และหาค่าความถี่

C_2

Itemsets	Support
1, 2	1
1, 3	2
1, 5	1
2, 3	2
2, 5	3
3, 5	2

→

L_2

Itemsets	Support
1, 3	2
2, 3	2
2, 5	3
3, 5	2

(ง) หา Frequent 2-itemsets

ภาพประกอบ 2-5 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Apriori (ต่อ)

L_2

Itemsets	Support
1, 3	2
2, 3	2
2, 5	3
3, 5	2

→

C_3

Itemsets	Support
2, 3, 5	2

(จ) สร้าง Candidate 3-itemsets และหาค่าความถี่

C_3

Itemsets	Support
2, 3, 5	2

→

L_3

Itemsets	Support
2, 3, 5	2

(ฉ) หา Frequent 3-itemsets

Frequent itemsets	Support
1	2
2	3
3	3
5	3
1, 3	2
2, 3	2
2, 5	3
3, 5	2
2, 3, 5	2

(ช) Frequent itemsets ทั้งหมด ($L_1 \cup L_2 \cup L_3$)

ภาพประกอบ 2-5 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Apriori (ต่อ)

ข้อด้อยของอัลกอริทึม Apriori คือ การหาค่าความถี่ของแต่ละ Candidate itemsets จะต้องทำการอ่านฐานข้อมูลทุกครั้ง เพื่อให้ได้ Frequent itemsetsในระดับถัดไป ดังนั้นจะเห็นได้ว่าต้องอ่านฐานข้อมูลจำนวน ***k*** ครั้ง (เมื่อ ***k*** คือขนาดของ Itemsets ที่ใหญ่ที่สุดที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ) เพื่อตรวจสอบค่าสนับสนุนของ

Candidate itemsets โดยเฉพาะเมื่อฐานข้อมูลมีความหนาแน่นของข้อมูลสูง หรือมีจำนวน Frequent itemsets มาก

2.3.2.2 อัลกอริทึม FP-growth

อัลกอริทึม FP-growth (Han *et al.*, 2000; Goethals, 2003; Song, 2006) เป็นอัลกอริทึมที่ใช้โครงสร้างข้อมูลแบบ FP-tree (Frequent Patterns Tree) ในการหา Frequent itemsets โดยไม่ต้องสร้าง Candidate itemsets เหมือนอัลกอริทึม Apriori ที่ต้องสร้าง Candidate itemsets จำนวนมาก ซึ่งทำให้เกิดปัญหาคอขวด (Bottleneck) เนื่องจากต้องอ่านฐานข้อมูลหลายครั้ง ซึ่งมีหลักการในการเพิ่มประสิทธิภาพ 3 เทคนิคหลักเพื่อช่วยแก้ปัญหาคือ (1) การจัดเก็บ Itemsets ในรูปแบบ FP-tree ซึ่งมีเพียง Item ที่ผ่านค่าความถี่ขั้นต่ำ และโหนดที่มีค่าความถี่มาก จะมีโอกาสเป็นโหนดร่วมมากกว่าโหนดที่มีค่าความถี่น้อย เพื่อหลีกเลี่ยงการอ่านฐานข้อมูลซ้ำหลายรอบ (2) ไม่มีการสร้าง Candidate itemsets (3) ใช้เทคนิคการแบ่งส่วนข้อมูล (Partitioning-based) หรือ Divide-and-conquer method โดยการหา Frequent itemsets ที่มีความยาวมาก ๆ จากการนำ Frequent itemsets ที่สั้นกว่ามาเชื่อมต่อกัน เพื่อรองรับการขยายตัวของฐานข้อมูล เทคนิคเหล่านี้จะช่วยลดเวลาในการหา Frequent itemsets ได้มาก

การหา Frequent itemsets ด้วยอัลกอริทึม FP-growth ประกอบด้วย 2 ขั้นตอนคือ

1) สร้าง FP-tree จากฐานข้อมูลที่กำหนด ด้วยอัลกอริทึม Create FP-tree ดังภาพประกอบ 2-6 โดยมีการอ่านฐานข้อมูล 2 ครั้ง ดังนี้

1.1) อ่านฐานข้อมูลครั้งที่ 1 เพื่อนับค่าความถี่ของแต่ละ Item ในฐานข้อมูล หลังจากนั้นทำการตัด Items ที่ไม่ผ่านค่าความถี่ขั้นต่ำออก แล้วนำ Items ที่เหลือมาเรียงลำดับตามค่าความถี่ของแต่ละ Items จากมากไปน้อย ได้ผลลัพธ์เป็น Head Table ดังภาพประกอบ 2-8(ก)

1.2) อ่านฐานข้อมูลครั้งที่ 2 เพื่อสร้าง FP-tree โดยการสร้างโหนดรากของ FP-tree และกำหนดค่าให้เป็น “null” สำหรับแต่ละทรานแซคชันในฐานข้อมูลให้เลือก Items ที่ผ่านค่าความถี่ขั้นต่ำ (Frequent items) และทำการเรียงลำดับตามค่าความถี่จากมากไปน้อย ก่อนการเพิ่มเข้าไปในต้นไม้ด้วยฟังก์ชัน InsertTree() ของอัลกอริทึม Create FP-tree ตัวอย่างการสร้าง FP-tree เช่น จากฐานข้อมูลภาพประกอบ 2-7 สามารถแสดงขั้นตอนการสร้าง FP-tree ได้ดังภาพประกอบ 2-8 เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2 (50%) และตัวเลขหลังเครื่องหมาย “:” หมายถึง ค่าความถี่ของแต่ละโหนด

Algorithm : Create FP-tree

Input : A database Database and a minimum support threshold ξ

Output : FP-tree

(1) Procedure **CreateFP-tree**

(2) scan Database once to collect the frequent items and their support them sort in support descending and create the header table

(3) FP-tree is null

(4) for each transaction t_i in Database

(5) select and sort the frequent items in t_i according to the order in the header table

(6) call InsertTree(FP-tree, t_i)

(7) end for

(8) Procedure **InsertTree**(*root, tran*)

(9) for each item k_i in *tran*

(10) if root has a child *N* that *N*.item_name = k_i

(11) increment *N*'s count by 1

(12) *root* = *N*

(13) else

(14) create the new node k_i is the child of root

(15) link the header table to node

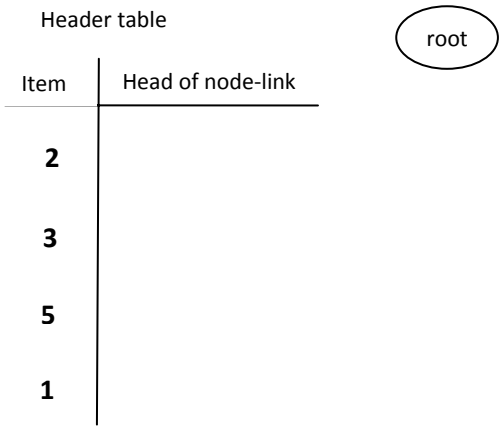
(16) end if

(17) end for

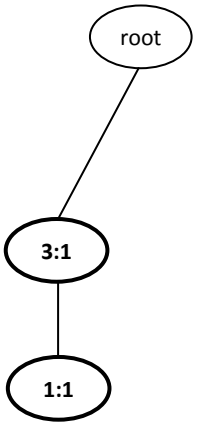
ภาพประกอบ 2-6 อัลกอริทึม Create FP-tree (Han *et al*., 2000)

TID	**Item**	**(Orders) Frequent Items**
100	1 3 4	3 1
200	2 3 5	2 3 5
300	1 2 3 5	2 3 5 1
400	2 5	2 5

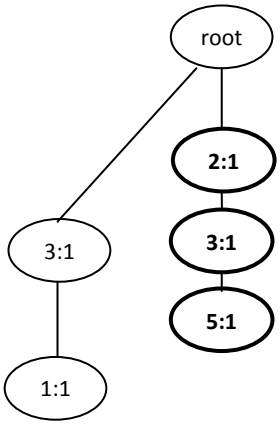
ภาพประกอบ 2-7 ฐานข้อมูล ซึ่งคอลัมน์ทางขวาสุด แสดง Frequent items ที่เรียงลำดับตามค่าความถี่จากมากไปน้อยของแต่ละทรานแซคชัน



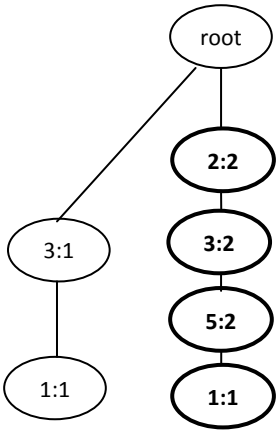
(ก) Header Table และ โหนดราก



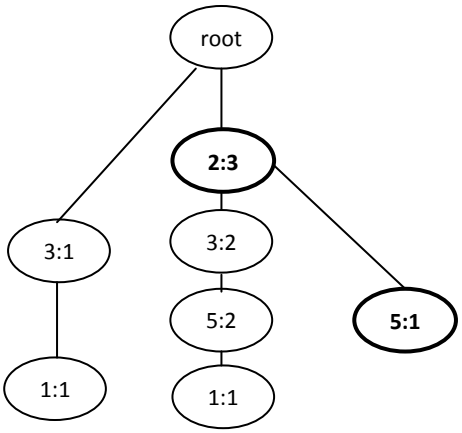
(ข) เพิ่ม Items {3,1} จาก TID 100



(ค) เพิ่ม Items {2,3,5} จาก TID 200

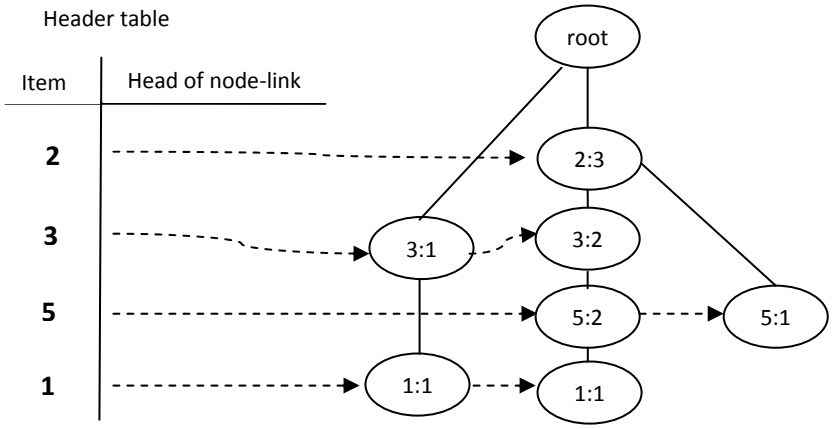


(ง) เพิ่ม Items {2,3,5,1} จาก TID 300



(จ) เพิ่ม Items {2,5} จาก TID 400

ภาพประกอบ 2-8 ตัวอย่างการสร้าง FP-tree



(ฉ) ต้นไม้ FP-tee ที่สมบูรณ์

ภาพประกอบ 2-8 ตัวอย่างการสร้าง FP-tree (ต่อ)

2) หา Frequent itemsets จาก FP-tree ที่สร้างได้ด้วยอัลกอริทึม FP-growth ดังภาพประกอบ 2-9

Algorithm: FP-growth

Input : FP-tree and a minimum support threshold ξ

Output : The set of all frequent itemsets

(1) Procedure **FP-growth**(Tree, α)
(2) if *Tree* contains a single path *P*
(3) for each combination of nodes (denoted as β) in the part *P*
(4) Generate itemset $\beta \cup \alpha$ with support = minimum support of nodes in β
(5) end for
(6) else
(7) for each a_i in the header table of *Tree*
(8) Generate itemsets $\beta = a_i \cup \alpha$ with support = a_i.support
(9) Construct β's conditional pattern base and conditional FP-tree β
(10) end for
(11) if *Tree* $\beta \neq \phi$
(12) Call FP-growth(Tree β , β)
(13) end if
(14) end if

ภาพประกอบ 2-9 อัลกอริทึม FP-growth (Han *et al.,* 2000)

จากอัลกอริทึม FP-growth สัญลักษณ์ ξ หมายถึง ค่าความถี่ขั้นต่ำที่กำหนด α หมายถึง Itemsets ในฐานข้อมูล และ β หมายถึง Itemsets ใน Conditional pattern base ของ α (α's conditional pattern base) โดยการหา Frequent itemsets จะเริ่มพิจารณาจาก Frequent item ที่อยู่ด้านล่างของ Header table แต่ละ Item มีการสร้าง Conditional pattern base และ Conditional FP-tree เมื่อ Conditional pattern base หมายถึงเซตของ Item ที่เกิดพร้อม Item นั้นในแต่ละเส้นทาง และกำหนดให้ทุก Item มีค่าความถี่เท่ากับค่าความถี่ของ Item นั้น เช่นจาก FP-tree ดังภาพประกอบ 2-8(ฉ) เมื่อพิจารณา Item 1 จะเห็นว่ามี 2 เส้นทางที่มี Item 1 ปรากฏอยู่ ได้แก่ (3:1, 1:1) และ (2:3, 3:2, 5:2, 1:1) เมื่อตัวเลขที่อยู่ด้านหลังเครื่องหมาย ":" หมายถึงค่าความถี่ เส้นทางแรกแสดงให้เห็นว่า (3, 1) ปรากฏ 1 ครั้งในฐานข้อมูล และเส้นทางที่สองจะเห็นว่า (2, 3, 5, 1) ปรากฏอยู่ 1 ครั้ง (เกิดพร้อม Item 1 เพียง 1 ครั้ง) ดังนั้นจะได้เซตของ Item ที่เกิดพร้อม Item 1 คือ {(3:1), (2:1, 3:1, 5:1)} ซึ่งเรียกว่า 1' Conditional pattern base จากนั้นสร้าง FP-tree บน Conditional pattern base นี้ เรียกว่า Conditional FP-tree ซึ่งเกิดจากการนำค่าความถี่ของแต่ละ Item ในทุกเส้นทางมารวมกันและเลือกเฉพาะ Item ที่ผ่านค่าความถี่ขั้นต่ำ จาก Conditional FP-tree นำไปสร้าง Frequent itemsets ต่อไป ดังนั้นจาก FP-tree ดังภาพประกอบ 2-8(ฉ) จะได้ Conditional pattern base และ Conditional FP-tree ของแต่ละ Item แสดงดังภาพประกอบ 2-10 และสามารถหา Frequent itemsets ทั้งหมดได้ดังภาพประกอบ 2-11

item	Conditional pattern base	Conditional FP-tree
1	{(3:1), (2:1, 3:1, 5:1)}	{(3:2)}| 1
5	{(2:2, 3:2), (2:1)}	{(2:3, 3:2)}| 5
3	{(2:2)}	{(2:2)}| 3
2	ϕ	ϕ

ภาพประกอบ 2-10 Conditional pattern base และ Conditional FP-tree ของแต่ละ Item

ข้อด้อยของอัลกอริทึม FP-growth คือ ไม่เหมาะกับฐานข้อมูลที่มีขนาดใหญ่และมีการกระจายสูง เพราะจะทำให้โครงสร้าง FP-tree ต้องการพื้นที่หน่วยความจำมาก จึงมีงานวิจัยที่ได้คิดค้นเทคนิคการลดขนาดของ FP-tree เช่น H-mine (Pei *et al.*, 2001) เพื่อให้สามารถทำงานบนหน่วยความจำที่จำกัดได้ดียิ่งขึ้น

Frequent Itemsets	Support
1	2
2	3
3	3
5	3
1, 3	2
2, 3	2
2, 5	3
3, 5	2
2, 3, 5	2

ภาพประกอบ 2-11 Frequent itemsets ทั้งหมด

2.3.2.3 อัลกอริทึม Closed

อัลกอริทึม Closed (Pasquier *et al*., 1999) เป็นอัลกอริทึมที่พัฒนามาเพื่อแก้ปัญหาของอัลกอริทึม Apriori ในกรณีที่ต้องมีการอ่านฐานข้อมูลหลายครั้ง โดยมีหลักการอยู่ว่า ถ้า Closed itemsets ขนาด *k* ใด ๆ ไม่เป็น Frequent itemsets แล้ว Closed Itemsets ขนาด *k*+1 ที่เป็น Superset ของ Closed itemsets นั้นจะไม่เป็น Frequent itemsets ด้วย หรือกล่าวอีกนัยหนึ่งว่า ถ้า Closed itemset ที่เป็น Superset เป็น Frequent itemsets แล้ว Closed itemsets ที่เป็นสับเซตจะเป็น Frequent itemsets ด้วย และ เซตของ Frequent itemsets ที่ยาวที่สุดจะเท่ากับเซตของ Frequent closed itemsets ที่ยาวที่สุด โดยการเลือกใช้โครงสร้างข้อมูลที่เหมาะสมเพื่อลดการใช้หน่วยความจำหลักในการจัดเก็บข้อมูลให้น้อยลง ขั้นตอนการทำงานของอัลกอริทึม Closed แสดงดังภาพประกอบ 2-12 2-13 และ 2-14

(1) Generators in $FCC_1 \leftarrow$ {1-itemsets};
(2) For ($i \leftarrow 1$; FCC_i.generator $\neq \phi$; i++) do begin
(3) Closures in $FCC_i \leftarrow \phi$;
(4) Supports in $FCC_i \leftarrow 0$;
(5) $FCC_i \leftarrow$ Gen-Closure(FCC_i) // Produces generators closures
(6) Forall candidate closed itemsets $c \in FCC_i$ do begin
(7) If (c.support $\geq$ minsupport) then
(8) $FC_i \leftarrow FC_i \cup \{c\}$;
(9) End
(10) $FCC_{i+1} \leftarrow$ Gen-Generator(FC_i); // Creates generators of iteration i+1
(11) End
(12) Answer $FC \leftarrow \cup_{j=1}^{j=i-1} \{FC_j$.closure, FC_j.support};

ภาพประกอบ 2-12 อัลกอริทึม Closed (Pasquier and Bastide, 1999)

(1) forall objects $o \in O$ do begin
(2) $G_o \leftarrow$ Subset(FCC_i.generator,f({o}) ; // Gen. that are subsets of f({0}))
(3) forall generators $p \in G_o$ do begin
(4) if (p. closure = ϕ) then
(5) p. closure $\leftarrow$ f({o});
(6) else p. closure $\leftarrow$ p. closure ∩ f({o});
(7) p.support++;
(8) end
(9) end
(10) Answer $\leftarrow \cup \{c \in FCC_i \mid c$.closure $\neq \phi\}$;

ภาพประกอบ 2-13 ฟังก์ชัน Gen-Closure (Pasquier and Bastide, 1999)

```
(1)  insert into FCC_{i+1}.generator
(2)  select p.item_1, p.item_2, …, p.item_i, q.item_i
(3)  from FC_i.generator p, FC_i.generator q
(4)  where p.item_1 = q.item_1, …, p.item_{i-1} = q.item_{i-1}, p.item_i < q.item_i
(5)  for all generators p ∈ FCC_{i+1}.generator do begin
(6)        for all i-subsets s of p do begin
(7)              if ( s ∉ FC_i.generator then
(8)                   delete p from FCC_{i+1}.generator;
(9)        end
(10) end
(11) for all generators p ∈ FCC_{i+1}.generator do begin
(12)       S_p ← Subset( FC_i.generator , p);  // Generators that are subsets of p
(13)       for all s ∈ S_p do begin
(14)             if (p ⊆ s.closure) then
(15)                  delete p from FCC_{i+1}.generator;
(16)       end
(17) end
(18) Answer ← ∪{ c ∈ FCC_{i+1}};
```

ภาพประกอบ 2-14 ฟังก์ชัน Gen-Generator (Pasquier and Bastide, 1999)

จากอัลกอริทึม Closed สัญลักษณ์ FCC_k หมายถึงเซตของ Generator ที่มีความยาว k และ FC_k หมายถึงเซต Frequent closed itemsets ของ Generator ที่มีความยาว k การทำงานของฟังก์ชัน Gen-Generator คล้ายกับการสร้าง Candidate itemsets ในอัลกอริทึม Apriori โดย FCC_k เกิดจากการรวมกันของ FCC_{k-1} โดยมีเงื่อนไขเพิ่มเติมว่า FCC_k ใด ๆ ต้องไม่เป็นสับเซตของ FC_{k-1} และสมาชิกทุกตัวต้องเป็นสมาชิกใน FC_{k-1} ด้วย

ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Closed แสดงดังภาพประกอบ 2-15 เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2 (50%)

TID	Item
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

(ก) Database

FCC_1

Generator	Closure	Support
1	{1,3}	2
2	{2,5}	3
3	{3}	3
4	{4}	1
5	{2,5}	3

(ข) หา Closure ของแต่ละ Generator และค่าความถี่ของแต่ละ Closure

FC_1

Generator	Closure	Support
1	{1,3}	2
2	{2,5}	3
3	{3}	3
5	{2,5}	3

(ค) เลือกแถวที่ไม่ผ่านค่าความถี่ขั้นต่ำออก

FFC_2

Generator	Closure	Support
1,2	{1,2,3,5}	1
1,5	{1,2,3,5}	1
2,3	{2,3,5}	2
3,5	{2,3,5}	2

(ง) หา Closure ของแต่ละ Generator และค่าความถี่ของแต่ละ Closure

FC_2

Generator	Closure	Support
2,3	{2,3,5}	2
3,5	{2,3,5}	2

(จ) เลือกแถวที่ไม่ผ่านค่าความถี่ขั้นต่ำออก

Frequent Itemsets	Support
3	3
1,3	2
2,5	3
2,3,5	2

(ฉ) Frequent closed itemsets ทั้งหมด

ภาพประกอบ 2-15 ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Closed

ข้อด้อยของอัลกอริทึม Closed คือ Frequent itemsets ที่หาได้คือ Frequent closed Itemsets หรือกลุ่มข้อมูลที่มีขนาดใหญ่ที่สุดเท่านั้น ไม่สามารถหาสับเซตอื่นๆ (Subset) ได้

2.3.2.4 อัลกอริทึม Index-BitTableFI

อัลกอริทึม Index-BitTableFI (Song *et al*., 2008) เป็นอัลกอริทึมที่มีการเพิ่มประสิทธิภาพโดยการลดจำนวนของการสร้าง Candidate itemsets ลดความต้องการหน่วยความจำและลดเวลาที่ใช้ในการทำงาน ซึ่งสามารถทำงานได้อย่างมีประสิทธิภาพกับชุดข้อมูลที่มีการกำหนดค่าความถี่ขั้นต่ำน้อย ๆ และมีความหนาแน่นของชุดข้อมูลสูง โดยการเก็บข้อมูลในรูปแบบบิตแมปเวกเตอร์ เรียกว่าตาราง BitTable จึงมีการอ่านฐานข้อมูลเพียงครั้งเดียวเพื่อสร้างตาราง BitTable ตัวอย่างตาราง BitTable แสดงดังภาพประกอบ 2-18(ข) การหา Frequent itemsets เริ่มจากการหา Frequent 1-itemsets จากการนับจำนวนบิต 1 ของแต่ละ Item และเลือกเฉพาะ Item ที่มีค่าความถี่มากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำที่กำหนด จากนั้นหา Index array ของแต่ละ Frequent item โดยใช้อัลกอริทึมดังภาพประกอบ 2-16 และนำ Index array ที่ได้ไปหา Frequent *k*-itemsets ต่อไป โดยอัลกอริทึมการหา Frequent itemsets แสดงดังภาพประกอบ 2-1

Algorithm : Computing index array

Input : database, minimum support

Output : index array

(1) Scan database once. Delete infrequent itemsets;

(2) Sort frequent single items in supports ascending order as $a_1, a_2, \ldots, a_m$

(3) for each element *index[j]* of index array do

(4) $index[j].item = a_j$;

(5) Represent the database with BitTable;

(6) for each element *index[j]* in index array do

(7) $index[j].subsume = \phi$;

(8) $candidate = \cap_{t \in g(index[j].item)} t$;

(9) for each $i > j$ do

(10) if the value of the *i*-th bit in *candidate* is set then

(11) $index[j].subsume \leftarrow index[j].item$;

(12) end if

(13) end for

(14) end for

(15) Write out the index array;

ภาพประกอบ 2-16 อัลกอริทึมในการหา Index array (Song *et al.,* 2008)

การหา Index array ของแต่ละ Item สามารถหาได้โดยการอินเตอร์เซค ($\cap$) ระหว่างบิตแมปเวกเตอร์ของทรานแซคชันที่ Item นั้นปรากฏอยู่ตั้งแต่ตำแหน่งของ Item นั้นไปทางขวา เช่น จากตาราง BitTable ดังภาพประกอบ 2-18(ข) หา Index array ของ Item 2 ได้จากการอินเตอร์เซคระหว่างบิตแมปเวกเตอร์ของ *TID* 200 300 และ 400 ตั้งแต่ Item 2 ไปทางขวาดังนี้ $111 \cap 111 \cap 101$ ได้ผลลัพธ์เป็น 101 ซึ่งบิต 1 ปรากฏอยู่ ณ ตำแหน่งของ Item 2 และ Item 5 ดังนั้น (2,5) เป็น Index array ของ Item 2

ตัวอย่างการหากลุ่มข้อมูลที่ปรากฏบ่อยด้วยอัลกอริทึม Index-BitTableFI แสดงดังภาพประกอบ 2-18 เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 2 (40%) โดยจากฐานข้อมูลดังภาพประกอบ 2-18(ก) เริ่มจากการอ่านฐานข้อมูลหนึ่งครั้งเพื่อสร้างตาราง BitTable แล้วหาค่าความถี่ของแต่ละ Item และเลือก Item ที่ไม่ผ่านค่าความถี่ขั้นต่ำออก จากนั้นเรียงลำดับ Items ในตาราง BitTable ตามค่าความถี่จากน้อยไปมากได้ผลลัพธ์ดังภาพประกอบ 2-18(ข) จากนั้นหา Index array ของแต่ละ Item ได้ผลลัพธ์ดังภาพประกอบ 2-18(ค) สุดท้ายหา Frequent itemsets ด้วยอัลกอริทึม Index-BitTableFI ได้ผลลัพธ์ดังภาพประกอบ 2-18(ง)

Algorithm : Index-BitTableFI Algorithm

Input : index array, minimum support

Output : frequent itemsets

```
(1)  for each element index[j] of index array do
(2)      Write Out index[j].item and its support;
(3)      if index[j].subsume == ϕ then
(4)          If (sup(index[j].item) > min_sup) then
(5)              Depth_First(index[j].item));
(6)          end if
(7)      else
(8)          for each element s-item ⊆ index[j].subsume do
(9)              Write Out index[j].item ∪ s-item and its support;
(10)         end for
(11)         if (sup(index[j]).item) > min_sup) then
(12)             tail ← t(index[j].item)\items in index[j].subsume;
(13)             Depth_First(index[j].item,tail);
(14)             for each element s-item ⊆ index[j].subsume do
(15)                 Depth_First(index[j].item ∪ s-item, tail);
(16)             End for
(17)         End if
(18)     End else
(19)     End for
(20)     Procedure Depth_First(itemsets, tail)
(21)     if tail == ϕ then return;
(22)     for each i ∈ tail do
(23)         f – itemset ←itemset ∪ i;
(24)         if sup(f – itemset) ≥min_sup then
(25)             Write Out f – itemset and its support;
(26)             tail ←tail \ i;
(27)             Dept_First(f – itemset, tail);
(28)         end if
(29)     end for
```

ภาพประกอบ 2-17 อัลกอริทึม Index-BitTableFI (Song *et al.,* 2008)

TID	Item
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

(ก) Database

TID	1:2	2:3	3:3	5:3
100	1	0	1	0
200	0	1	1	1
300	1	1	1	1
400	0	1	0	1

(ข) ตาราง BitTable

item	index array
1	(1,3)
2	(2,5)
3	$(3,\phi)$
5	$(5,\phi)$

(ค) Index array

Frequent Itemsets	Support
1	2
2	3
3	3
5	3
1, 3	2
2, 3	2
2, 5	3
3, 5	2
2, 3, 5	2

(ง) Frequent itemsets ทั้งหมด

ภาพประกอบ 2-18 ตัวอย่างการหา Frequent itemsets ด้วยอัลกอริทึม Index-BitTableFI

ข้อด้อยของอัลกอริทึม Index-BitTableFI คือ เมื่อมีจำนวน Items ในฐานข้อมูลมากหรือมีจำนวน Items ในแต่ละทรานแซคชันมาก จะทำให้เสียเวลาในการสร้างตาราง BitTable และในกรณีที่จำนวน Item ในฐานข้อมูลมีมากแต่ความหนาแน่นในแต่ทรานแซคชันมีน้อยจะทำให้เสียเนื้อที่ในหน่วยความจำเป็นจำนวนมาก

นอกจากอัลกอริทึมตามที่กล่าวมายังมีอัลกอริทึมอื่น ๆ อีกมากมาย ที่ได้ทำการศึกษาเพื่อหาขั้นตอนวิธีที่มีประสิทธิภาพสำหรับงานวิจัยนี้ เช่น อัลกอริทึม Transaction Mapping (Song and Sanguthevar, 2006) อัลกอริทึม Eclat (Zaki and Gouda, 2003; Goethals, 2003) อัลกอริทึม dEclat (Zaki and Gouda, 2003; Goethals, 2003) อัลกอริทึม Bit-AssocRule (Lin *et al*., 2003) อัลกอริทึม Boolean (Ying and Leu, 1999) อัลกอริทึม Bitmap Based (Gardarrin *et al*., 1998) เป็นต้น โดยแต่ละวิธีพยายามออกแบบเพื่อลดเวลาในการประมวลผลและยังคงประสิทธิภาพในการทำงานเพื่อหากลุ่มข้อมูลที่ต้องการ

บทที่ 3

ดัชนีแบบบิตแมป (Bitmap index)

ดัชนีแบบบิตแมปเป็นเทคนิควิธีที่ช่วยในการเพิ่มความเร็วในการค้นหาข้อมูล เนื่องจากสามารถดำเนินการระดับบิตเช่น AND OR NOT และ XOR ระหว่างบิตแมปเวกเตอร์ก่อนดึงข้อมูลจริง ทำให้มีประสิทธิภาพในการค้นหาข้อมูลเพราะสนับสนุนการทำงานของฮาร์ดแวร์ นอกจากนี้การทำดัชนีบิตแมปยังมีประสิทธิภาพในเรื่องพื้นที่ในการจัดเก็บดัชนี โดยเฉพาะกับแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำ ที่ผ่านมาได้มีการนำเสนอเทคนิคการทำดัชนีบิตแมปแบบต่าง ๆ ที่น่าสนใจ ซึ่งมีรูปแบบการลงรหัส (Encoding scheme) ที่แตกต่างกัน ในบทนี้จะกล่าวถึงเทคนิคการทำดัชนีบิตแมปแบบต่าง ๆ ดังนี้

- ดัชนีบิตแมปแบบพื้นฐาน (Simple Bitmap Index)
- ดัชนีบิตแมปแบบบีบอัด (Compression Bitmap Index) ประกอบด้วย
  - การบีบอัดแบบ Word-aligned Hybrid
  - การบีบอัดแบบ Run-Length Huffman
- ดัชนีบิตแมปแบบไม่บีบอัด (Uncompression Bitmap Index) ประกอบด้วย
  - ดัชนีบิตแมปแบบช่วง (Interval Bitmap Index)
  - ดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index)
  - ดัชนีบิตแมปแบบคู่กัน (Dual Bitmap Index)
  - ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded Bitmap Index)

3.1 ดัชนีบิตแมปแบบพื้นฐาน (Simple Bitmap Index)

การทำดัชนีบิตแมปแบบพื้นฐาน (Wu and Buchmann, 1998; Chan and Ioannidis, 1999; Wu *et al*., 2002; Vanichayobon *et al*., 2006; Wattanakitrungroj and Vanichayobon, 2006; Stockinger and Wu, 2007; Elizabeth and Patrick , 2007; Stabno and Wrembel, 2007; Koudas, 2000) บนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ C ประกอบด้วย C บิตแมปเวกเตอร์ คือ S^0, S^1, S^2, ..., S^{C-1} นั่นคือ ดัชนีบิตแมปแบบพื้นฐานใช้ 1 บิตแมปเวกเตอร์ต่อ 1 ค่าข้อมูล โดยที่ S^j แทนบิตแมปเวกเตอร์ที่มีค่าของแอทริบิวต์เท่ากับ j ซึ่งในการแทนค่าจะกำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ S^j มีค่าเป็น 1 ก็ต่อเมื่อแถวที่ i มีค่าของแอทริบิวต์เท่ากับ j เมื่อ j เป็นสมาชิกของแอทริบิวต์ที่นำมาทำดัชนี ส่วนบิตอื่นให้มีค่าเป็น 0 ตัวอย่างการทำดัชนีบิตแมปแบบพื้นฐานดังภาพประกอบ 3-1 เป็นการทำดัชนีบิตแมปแบบ

พื้นฐานบนแอทริบิวต์ *X* ของตาราง *T* โดยแอทริบิวต์ *X* ประกอบด้วยสมาชิก คือ {0,1,2,3,4,5,6,7} นั่นคือมีคาร์ดินอร์ลิตี้เท่ากับ 8 จึงมีการใช้ 8 บิตแมปเวกเตอร์ ได้แก่บิตแมปเวกเตอร์ S^0, S^1, S^2, S^3, S^4, S^5, S^6, S^7

RID	...	*X*	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
.		.	
.		.	
.		.	
10000		4	

(ก) ตาราง *T*

S^0	S^1	S^2	S^3	S^4	S^5	S^6	S^7
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
0	0	0	0	1	0	0	0

(ข) ดัชนีบิตแมปแบบพื้นฐาน

ภาพประกอบ 3-1 ตัวอย่างการทำดัชนีบิตแมปแบบพื้นฐานบนแอทริบิวต์ *X* ของตาราง *T*

สำหรับการสอบถามข้อมูลแบบค่าเท่ากัน (Equality query) ของดัชนีบิตแมปแบบพื้นฐานทำได้โดยตรวจสอบบิตแมปเวกเตอร์ที่แทนค่านั้น ๆ ซึ่งมีรูปแบบดังนี้

$$\text{“X = } j\text{”} = S^j$$

ตัวอย่างการสอบถามข้อมูลที่มีค่าแอทริบิวต์ “*X*=3” สามารถตรวจสอบได้จากบิตแมปเวกเตอร์ S^3 จากภาพประกอบ 3-1 จะได้ว่าบิตที่ 7 และบิตที่ 8 ของบิตแมปเวกเตอร์ S^3 มีค่าเป็น 1 แสดงว่าแถวที่ 7 และ 8 มีค่าของแอทริบิวต์ “*X*=3” สำหรับการสอบถามข้อมูลแบบสมาชิก (Membership query) ทำได้โดยดำเนินการตรรกะ OR ระหว่างบิตแมปเวกเตอร์ของค่าที่ต้องการสอบถาม เช่นเมื่อต้องการสอบถามตามเงื่อนไข “*X* in {0, 3, 5, 6}” ผลลัพธ์ของการสอบถามคือ $S^0 + S^3 + S^5 + S^6$ (ในที่นี้ใช้สัญลักษณ์ + แทน OR) นั่นคือ แถวที่ 3, 5, 6, 7, 8 และ 9 เป็นคำตอบของการสอบถาม

ข้อเด่นของดัชนีบิตแมปแบบพื้นฐาน

ดัชนีบิตแมปแบบพื้นฐานจะทำงานได้ดี เมื่อแอทริบิวต์ที่นำมาทำดัชนีมีค่าคาร์ดินอร์ลิตี้ต่ำ เช่น แอทริบิวต์เพศ ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ 2 คือ เพศชายและเพศหญิง เพราะมีการใช้เพียง 2 บิตแมปเวกเตอร์ในการสร้างดัชนี ทำให้ลดพื้นที่ในการจัดเก็บดัชนีได้มาก

ข้อด้อยของดัชนีบิตแมปแบบพื้นฐาน

เมื่อคาร์ดินอร์ลิตี้ของแอทริบิวต์ที่นำมาทำดัชนีสูงขึ้น จำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนีบิตแมปแบบพื้นฐานจะมากขึ้นด้วย และยังใช้ประโยชน์จากบิตแมปเวกเตอร์ไม่เต็มที่ ทำให้สิ้นเปลืองพื้นที่ที่ใช้ในการจัดเก็บดัชนี

ด้วยเหตุนี้งานวิจัยเกี่ยวกับการทำดัชนีบิตแมปจำนวนมากจึงมุ่งประเด็นไปที่การลดขนาดดัชนี เพื่อให้สามารถทำดัชนีบิตแมปบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้สูงได้ และยังคงประสิทธิภาพในการสอบถามข้อมูล จากการศึกษาเทคนิคการลดขนาดดัชนีแบ่งได้เป็น 2 วิธี คือ เทคนิคที่มีการบีบอัดดัชนี (Compression bitmap index) (Stockinger and Wu, 2007; Elizabeth and Patrick, 2007; Wu *et al*., 2002; Stabno and Wrembe, 2007) และเทคนิคที่ไม่มีการบีบอัดดัชนี (Uncompression bitmap index) (Wu and Buchmann, 1998; Chan and Ioannidis, 1999; Koudas, 2000; Wattanakitrungroj and Vanichayobon, 2006; Vanichayobon *et al*., 2006) สำหรับในงานวิจัยนี้จะมุ่งประเด็นไปที่การทำดัชนีที่ไม่มีการบีบอัด เนื่องจากการดำเนินการตรรกะระดับบิตบนบิตแมปที่มีการบีบอัดไม่สามารถทำได้โดยตรง

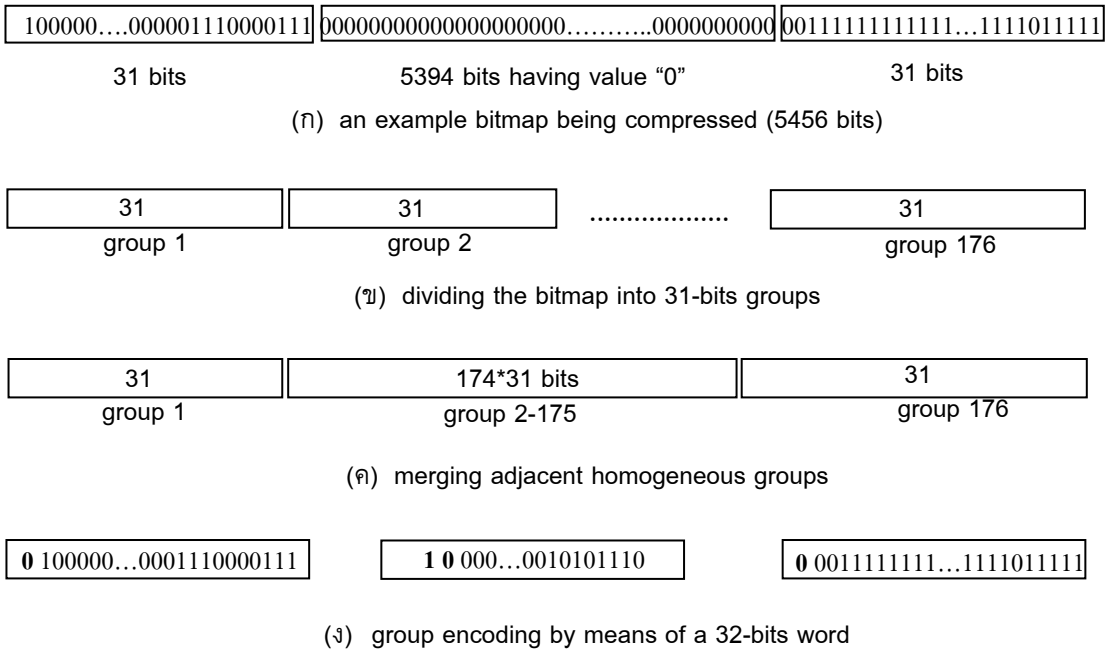
3.2 ดัชนีบิตแมปแบบบีบอัด (Compression Bitmap Index)

การบีบอัดดัชนีบิตแมปคือ การบีบอัดแต่ละบิตแมปเวกเตอร์ของดัชนีบิตแมปแบบพื้นฐาน เทคนิคการบีบอัดที่ได้ทำการศึกษาได้แก่ การบีบอัดแบบ WAH และ RLH มีรายละเอียดดังนี้

3.2.1 WAH (Word-aligned Hybrid) Compression

เทคนิคการบีบอัดบิตแมปแบบ WAH (Stockinger and Wu, 2007; Elizabeth and Patrick, 2007; Wu *et al*., 2002; Stabno and Wrembe, 2007) มีแนวคิดในการบีบอัดคือจะทำการบีบอัดบิตที่มีค่าเหมือนกันและอยู่ติดกันในแต่ละบิตแมปเวกเตอร์ ไม่ว่าจะเป็นบิต 0 หรือบิต 1 หลักการบีบอัดแบบ WAH มีวิธีการ 3 ขั้นตอน (Stabno and Wrembe, 2007) ดังนี้ ขั้นตอนที่หนึ่งเป็นการแบ่งบิตแมปออกเป็นกลุ่ม (Group) กลุ่มละ 31 บิต ดังตัวอย่างบิตแมปเวกเตอร์ในภาพประกอบ 3-2(ก) สามารถแบ่งออกได้เป็น 176 กลุ่มดังภาพประกอบ 3-2(ข) ในขั้นตอนที่สองจะเป็นการรวมกลุ่มของกลุ่มที่มีค่าบิตเหมือนกันและอยู่ติดกันให้เหลือเพียงกลุ่ม

เดียวดังแสดงในภาพประกอบ 3-2(ค) โดยกลุ่มที่ 1 ข้อมูลมีค่าบิตแตกต่างกัน (มีทั้งบิต 0 และบิต 1) ภายในกลุ่ม กลุ่มที่ 2 ถึง 175 มีค่าบิตเหมือนกันคือบิต 0 จึงมีการรวมให้เป็นกลุ่มเดียวที่มีจำนวนบิตเท่ากับ 174*31 บิต ส่วนกลุ่มที่ 3 เหมือนกับข้อมูลกลุ่มที่ 1 คือ มีค่าบิตแตกต่างกัน ขั้นตอนที่สามจะเป็นการบีบอัดให้แต่ละกลุ่มมี 32 บิต (1 เวิร์ด) โดยมีหลักการคือ สำหรับกลุ่มที่มีค่าบิตแตกต่างกันให้เติมบิต 0 ไว้ทางบิตซ้ายสุด สำหรับกลุ่มข้อมูลที่เกิดจากการรวมกันหลายกลุ่ม เนื่องจากมีค่าบิตเหมือนกันให้เติมบิต 1 ไว้ทางบิตซ้ายสุด และบิตถัดมาเป็นค่าของบิตที่ถูกบีบอัดคือ เป็นบิต 1 เมื่อบิตที่ถูกบีบอัดคือบิต 1 และเป็นบิต 0 เมื่อบิตที่ถูกบีบอัดคือบิต 0 ส่วน 30 บิตที่เหลือใช้เพื่อแทนจำนวนกลุ่มที่ถูกบีบอัดเข้าด้วยกัน ดังภาพประกอบ 3-2(ง) เป็นผลลัพธ์สุดท้ายที่เกิดจากการบีบอัดแบบ WAH



ภาพประกอบ 3-2 ขั้นตอนการบีบอัดแบบ WAH (Stabno and Wrembe, 2007)

สำหรับการสอบถามข้อมูลนั้น มีการดำเนินการเช่นเดียวกับดัชนีบิตแมปแบบพื้นฐาน แต่จะต้องทำให้บิตแมปเวกเตอร์ที่ถูกบีบอัดอยู่นั้น กลับคืนสู่สภาพเหมือนกับตอนที่ยังไม่ได้มีการบีบอัด (Decompression) จึงสามารถดำเนินการต่าง ๆ ได้

ข้อเด่นของดัชนีบิตแมปบีบอัดแบบ WAH

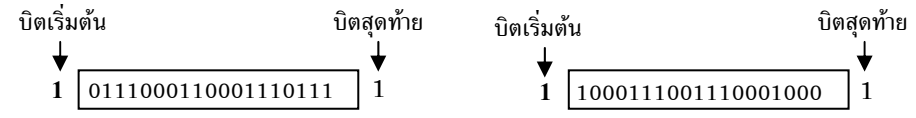
ใช้พื้นที่ในการจัดเก็บดัชนีและใช้เวลาในการสอบถามข้อมูลน้อยกว่าดัชนีบิตแมปแบบพื้นฐาน เมื่อในแต่ละบิตแมปเวกเตอร์มีบิตที่เหมือนกันอยู่ติดกันจำนวนมากอย่างภาพประกอบ 3-2(ก) เนื่องจากขนาดดัชนีลดลง และมีการใช้ I/O น้อยลง นั่นคือเหมาะที่จะบีบอัดบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้สูง

ข้อด้อยของดัชนีบิตแมปบีบอัดแบบ WAH

การดำเนินการตรรกะบนดัชนีบิตแมปที่มีการบีบอัดไม่สามารถทำได้โดยตรง นอกจากนี้หากแต่ละบิตแมปมีการกระจายของบิต 0 และ บิต 1 อย่างสม่ำเสมอ นั่นคือแต่ละเวิร์ดมีค่าบิตแตกต่างกัน มีเวิร์ดจำนวนน้อยที่มีค่าบิตเหมือนกันและอยู่ติดกันอย่างต่อเนื่อง ประสิทธิภาพของการบีบอัดจะน้อยลง จากการทดลองในงานวิจัยของ (Stabno and Wrembe, 2007) พบว่าไม่เหมาะจะบีบอัดบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้น้อยกว่า 20

3.2.2 RLH (Run-Length Huffman) Compression

เทคนิคการบีบอัดดัชนีบิตแมปแบบ RLH (Stabno and Wrembe, 2007) มีวิธีการคือ จากดัชนีบิตแมปแบบพื้นฐาน female และ male ดังภาพประกอบ 3-3 ขั้นตอนที่หนึ่งแปลงบิตแมปเวกเตอร์โดยการหาจำนวนบิต 0 ที่อยู่ระหว่างบิต 1 สองบิต เมื่อกำหนดให้บิตเริ่มต้นและบิตสุดท้ายของบิตแมปเวกเตอร์มีค่าเป็นบิต 1 ตัวอย่างเช่น บิตแมปเวกเตอร์ female ดังภาพประกอบ 3-3(ก) จะเห็นว่าบิตเริ่มต้นกับบิต 1 บิตแรกของบิตแมปเวกเตอร์ female มีจำนวนบิต 0 ระหว่างบิต 1 ทั้งสองอยู่ 1 บิต และบิต 1 ตัวแรกกับบิต 1 ตัวที่สองมีจำนวนบิต 0 อยู่ระหว่างบิต 1 ทั้งสองอยู่ 0 บิต เป็นต้น ดังนั้นจากบิตแมปเวกเตอร์ดังภาพประกอบ 3-3 จะได้ผลลัพธ์ของการแปลงบิตแมปเวกเตอร์เป็นจำนวนบิต 0 ที่อยู่ระหว่างบิต 1 ของทั้งสองบิตแมปเวกเตอร์ดังภาพประกอบ 3-4(ก) จากนั้นนับความถี่ของแต่ละค่าทั้งหมด ซึ่งจากภาพประกอบ 3-4(ก) ได้ผลลัพธ์ดังภาพประกอบ 3-4(ข)



(ก) bitmap for sex = ‘female’ (ข) bitmap for sex = ‘male’

ภาพประกอบ 3-3 ดัชนีบิตแบบพื้นฐานแมปบนแอทริบิวต์เพศ

female	male
1	0
0	3
0	0
3	0
0	2
3	0
0	0
0	3
1	3
0	
0	
0	

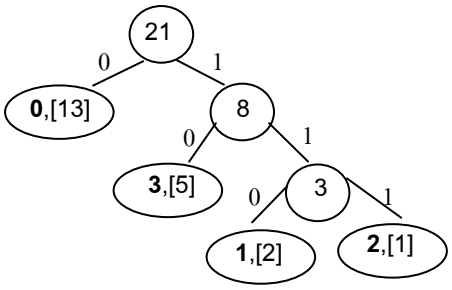
(ก) the modified run-length

distance	frequency
0	13
3	5
1	2
2	1

(ข) frequencies of distances

ภาพประกอบ 3-4 The modified run-length encoding (Stabno and Wrembe, 2007)

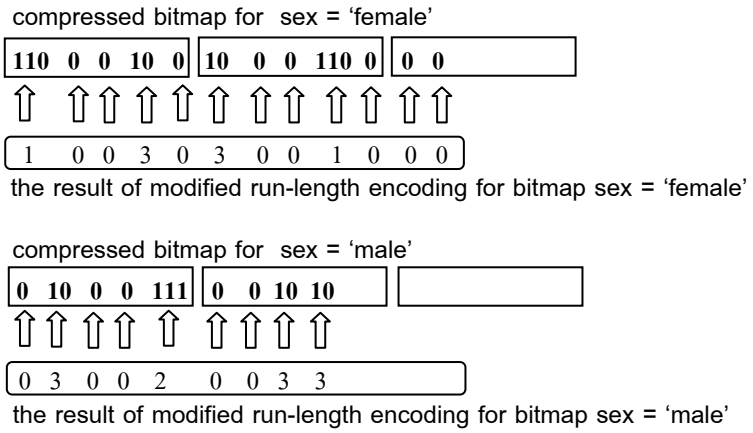
ขั้นตอนถัดไปเป็นการสร้าง Huffman tree เพื่อหา Huffman code ของแต่ละค่า เช่น จากภาพประกอบ 3-4(ข) จะถูกเข้ารหัสโดยใช้ Huffman algorithm ได้ Huffman tree แสดงดังภาพประกอบ 3-5 และจาก Huffman tree ได้ Huffman code ดังภาพประกอบ 3-6 ขั้นตอนสุดท้ายเป็นการนำ Huffman code ไปแทนที่แต่ละค่าของผลลัพธ์ที่ได้ในขั้นตอนที่หนึ่ง ดังนั้นจะได้ผลลัพธ์สุดท้ายของการบีบอัดแบบ RLH ดังภาพประกอบ 3-7



ภาพประกอบ 3-5 Huffman tree
(Stabno and Wrembe, 2007)

symbol	symbol code
0	0
3	10
1	110
2	111

ภาพประกอบ 3-6 Huffman code
(Stabno and Wrembe, 2007)



ภาพประกอบ 3-7 บิตแมปที่ได้จากการบีบอัดแบบ RLH (Stabno and Wrembe, 2007)

สำหรับการสอบถามข้อมูลนั้น มีการดำเนินการเช่นเดียวกับดัชนีบิตแมปแบบพื้นฐาน แต่จะต้องทำให้บิตแมปเวกเตอร์ที่ถูกบีบอัดอยู่นั้น กลับคืนสู่สภาพเหมือนกับตอนที่ยังไม่ได้มีการบีบอัด จึงสามารถดำเนินการต่าง ๆ ได้

ข้อเด่นของดัชนีบิตแมปแบบ RLH

เหมาะกับแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้สูง สามารถลดพื้นที่ในการจัดเก็บดัชนีได้มากกว่าดัชนีบิตแมปแบบพื้นฐาน และดัชนีบิตแมปที่มีการบีบอัดแบบ WAH และใช้เวลาในการสอบถามข้อมูลน้อยกว่าดัชนีบิตแมปแบบพื้นฐานและการบีบอัดแบบ WAH เนื่องจากมีการใช้ I/O น้อยกว่า โดยประสิทธิภาพของการบีบอัดแบบ RLH จะขึ้นอยู่กับขนาด Huffman tree ที่ถูกสร้างขึ้น ถ้า Huffman tree มีขนาดเล็กสามารถโหลดเข้าสู่หน่วยความจำหลักได้ในครั้งเดียว จะทำให้การบีบอัดแบบ RLH มีประสิทธิภาพมากขึ้น

ข้อด้อยของดัชนีบิตแมปแบบ RLH

ไม่เหมาะกับแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำ เนื่องจากจะทำให้บิตแมปมีความหนาแน่น กล่าวคือแต่ละบิตแมปเวกเตอร์มีบิต 1 จำนวนมาก ทำให้จำนวนบิต 0 ที่อยู่ระหว่างบิต 1 สองบิตมีจำนวนที่แตกต่างกันหลายค่า ส่งผลให้บิตแมปที่ผ่านการบีบอัดมีขนาดใหญ่ ซึ่งจากการทดลองใน (Stabno and Wrembe, 2007) พบว่าไม่เหมาะจะบีบอัดบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้น้อยว่า 5 และในการสอบถามข้อมูลไม่สามารถดำเนินการตรรกะระดับบิตบนดัชนีที่มีการบีบอัดได้โดยตรง

3.3 ดัชนีบิตแมปแบบไม่บีบอัด (Uncompression Bitmap Index)

สำหรับดัชนีบิตแมปแบบไม่มีการบีบอัดดัชนี เป็นการลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนี และยังคงประสิทธิภาพในการดำเนินการตรรกะระดับบิต ซึ่งเป็น

คุณสมบัติเด่นของการทำดัชนีแบบบิตแมป เทคนิคการลดจำนวนบิตแมปเวกเตอร์ที่ได้ศึกษาได้แก่ ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัส โดยมีรายละเอียดดังนี้

3.3.1 ดัชนีบิตแมปแบบช่วง (Interval Bitmap Index)

การทำดัชนีบิตแมปแบบช่วง (Chan and Ioannidis, 1999) บนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ C จะประกอบด้วย $\lceil C/2 \rceil$ บิตแมปเวกเตอร์ คือ $I^0, I^1, I^2, ..., I^{\lceil C/2 \rceil - 1}$ นั่นคือสามารถลดจำนวนบิตแมปเวกเตอร์ได้เป็นครึ่งหนึ่งของดัชนีบิตแมปแบบพื้นฐาน โดยที่ I^j แทนบิตแมปเวกเตอร์ที่มีค่าของแอทริบิวต์อยู่ในช่วง [*j,j+m*] เมื่อ $m = \lfloor C/2 \rfloor - 1$ ในการแทนค่าจะกำหนดให้บิตที่ *i* ของบิตแมปเวกเตอร์ I^j มีค่าเป็น 1 ก็ต่อเมื่อ แถวที่ *i* มีค่าของแอทริบิวต์ที่นำมาทำดัชนีบิตแมปอยู่ในช่วง [*j,j+m*] ส่วนบิตอื่น ๆ จะให้ค่าเป็น 0 ตัวอย่างการทำดัชนีบิตแมปแบบช่วงดังภาพประกอบ 3-8 เป็นการทำดัชนีบิตแมปแบบช่วงบนแอทริบิวต์ X โดยแอทริบิวต์ X ประกอบด้วยสมาชิกคือ {0,1,2,3,4,5,6,7} จึงต้องใช้ 4 บิตแมปเวกเตอร์ ได้แก่บิตแมปเวกเตอร์ I^0, I^1, I^2, I^3 และ $m = \lfloor 8/2 \rfloor - 1 = 3$ แต่ละบิตแมปเวกเตอร์แทนค่าได้ดังนี้ $I^0 = [0,3], I^1 = [1,4], I^2 = [2,5]$ และ $I^3 = [3,6]$ ซึ่งมีรูปแบบการลงรหัสดังภาพประกอบ 3-13 (ข)

RID	...	X	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
.		.	
.		.	
.		.	
10000		4	

(ก) ตาราง T

I^0	I^1	I^2	I^3
1	1	0	0
1	1	1	0
1	0	0	0
0	0	0	0
1	0	0	0
0	0	0	1
1	1	1	1
1	1	1	1
0	0	1	1
.	.	.	.
.	.	.	.
.	.	.	.
0	1	1	1

(ข) ดัชนีบิตแมปแบบช่วง

ภาพประกอบ 3-8 ตัวอย่างการทำดัชนีบิตแมปแบบช่วงบนแอทริบิวต์ X ของตาราง

สำหรับการสอบถามข้อมูลแบบค่าเท่ากันของการทำดัชนีบิตแมปแบบช่วง สามารถดำเนินการได้ดังต่อไปนี้

$$\text{“X=}v\text{”}= \begin{cases} I^0 & \text{if } v = 0,\ m = 0, \\ \overline{I^0} & \text{if } v = 1, C = 2, \\ I^1 & \text{if } v = 1, C = 3, \\ I^v \wedge I^{\overline{v+1}} & \text{if } v < m, \\ I^v \wedge I^0 & \text{if } v = m,\ m > 0, \\ I^{v-m} \wedge I^{\overline{v-m-1}} & \text{if } m < v < C\text{-}1,\ m > 0, \\ I^{\overline{\left\lceil \frac{C}{2} \right\rceil - 1}} \vee I^0 & \text{if } v = C\text{ -}1 \end{cases}$$

สำหรับการสอบถามข้อมูลแบบสมาชิก ทำได้โดยการหาบิตแมปเวกเตอร์ผลลัพธ์ของแต่ละค่าแล้วนำมาดำเนินการตรรกะ OR เช่น เมื่อต้องการสอบถามตามเงื่อนไข “*X* in {0, 3, 5, 6}” ผลลัพธ์ของการสอบถามคือ $(I^0 \wedge \overline{I^1}) + (I^3 \wedge I^0) + (I^2 \wedge I^1) + (I^3 \wedge I^2)$ นั่นคือ แถวที่ 3, 5, 6, 7, 8 และ 9 เป็นคำตอบของการสอบถาม

ข้อเด่นของดัชนีบิตแมปแบบช่วง

ดัชนีบิตแมปแบบช่วงมีประสิทธิภาพในด้านการใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนีมากกว่าดัชนีบิตแมปแบบพื้นฐาน คือ สามารถลดพื้นที่ได้ครึ่งหนึ่งของดัชนีบิตแมปแบบพื้นฐาน และในการสอบถามข้อมูลแบบค่าเท่ากันใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ และมีโอกาสตรวจสอบเพียง 1 บิตแมปเวกเตอร์ เมื่อ *C* น้อยกว่าหรือเท่ากับ 3

ข้อด้อยของดัชนีบิตแมปแบบช่วง

ในการสอบถามข้อมูลแบบค่าเท่ากัน และแบบสมาชิกของดัชนีบิตแมปแบบช่วงจะด้อยประสิทธิภาพกว่าดัชนีบิตแมปแบบพื้นฐาน เมื่อ *C* มากกว่า 3 เนื่องจากการสอบถามแต่ละค่าของดัชนีบิตแมปแบบช่วงใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ แต่การสอบถามแต่ละค่าของดัชนีบิตแมปแบบพื้นฐานตรวจสอบเพียง 1 บิตแมปเวกเตอร์เท่านั้น

3.3.2 ดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index)

การทำดัชนีบิตแมปแบบกระจาย (Vanichayobon *et al*., 2006) บนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ C ประกอบด้วย $\lceil 2\sqrt{C} \rceil$ บิตแมปเวกเตอร์ ซึ่งแบ่งเป็น 2 กลุ่มคือ บิตแมปเวกเตอร์กลุ่ม Z มี $\lceil C/(m-1) \rceil + 1$ บิตแมปเวกเตอร์ และบิตแมปเวกเตอร์กลุ่ม L มี $m-2$ บิตแมปเวกเตอร์ เมื่อ $m = \lceil \sqrt{C} \rceil + 1$ โดยแต่ละค่าของแอทริบิวต์จะถูกแทนด้วยบิตแมปเวกเตอร์ที่อยู่ในกลุ่ม Z และกลุ่ม L หรืออาจแทนด้วยบิตแมปเวกเตอร์ที่อยู่ในกลุ่ม Z เพียงอย่างเดียว

การสร้างบิตแมปเวกเตอร์กลุ่ม Z และ กลุ่ม L สามารถทำได้ดังนี้ เมื่อกำหนดให้ C คือ ค่าคาร์ดินอร์ลิตี้ที่นำมาสร้างดัชนี และ m คือ จำนวนสมาชิกภายในแต่ละกลุ่ม Z โดยที่ $m > 3$

การสร้างบิตแมปเวกเตอร์กลุ่ม Z

1. สร้างบิตแมปเวกเตอร์ Z^0 โดยกำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ Z^0 มีค่าเป็น 1 ก็ต่อเมื่อแถวที่ i มีค่าของแอทริบิวต์เท่ากับ 0 ส่วนบิตอื่น ๆ ให้มีค่าเป็น 0

2. สร้างบิตแมปเวกเตอร์ Z^j โดยกำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ Z^j มีค่าเป็น 1 ก็ต่อเมื่อ $j = \lfloor v/(m-1) \rfloor + 1$ โดยที่แถวที่ i มีค่าของแอทริบิวต์เท่ากับ v ส่วนบิตอื่น ๆ ให้มีค่าเป็น 0

และถ้า m-1 หาร v ลงตัวแล้วจะกำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ Z^{j-1} มีค่าเป็น 1 ด้วย ส่วนบิตที่เหลือจะให้ค่าเป็น 0

การสร้างบิตแมปเวกเตอร์กลุ่ม L

1. กำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ L^k มีค่าเป็น 1 ก็ต่อเมื่อ แถวที่ i มีค่าของแอทริบิวต์ที่นำมาทำดัชนีเป็น v โดยที่ $k = v \bmod(m\text{-}1)$, $v \in \{0,1,2,...,C-1\}$ และ $k \neq 0$

ตัวอย่างการทำดัชนีบิตแมปแบบกระจายดังภาพประกอบ 3-9 เป็นการทำดัชนีบิตแมปแบบช่วงบนแอทริบิวต์ X โดยที่แอทริบิวต์ประกอบด้วยสมาชิกคือ {0,1,2,3,4,5,6,7} ที่มีคาร์ดินอร์ลิตี้เท่ากับ 8 ดังนั้นมีการใช้ 6 บิตแมปเวกเตอร์ในการสร้างดัชนี ประกอบด้วยบิตแมปกลุ่ม Z มี 4 บิตแมปเวกเตอร์ ได้แก่ Z^0, Z^1, Z^2 , Z^3 และบิตแมปกลุ่ม L มี 2 บิตแมปเวกเตอร์ ได้แก่ L^1, L^2 ซึ่งมีรูปแบบการลงรหัสดังภาพประกอบ 3-13(ค)

RID	...	*X*	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
. . .		. . .	
10000		4	

(ก) ตาราง *T*

Z^0	Z^1	Z^2	Z^3	L^1	L^2
0	1	0	0	1	0
0	1	0	0	0	1
1	1	0	0	0	0
0	0	0	1	1	0
1	1	0	0	0	0
0	0	1	1	0	0
0	1	1	0	0	0
0	1	1	0	0	0
0	0	1	0	0	1
. . .	. . .	. . .	. . .	. . .	. . .
0	0	1	0	1	0

(ข) ดัชนีบิตแมปแบบกระจาย

ภาพประกอบ 3-9 ตัวอย่างการทำดัชนีบิตแมปแบบกระจายบนแอทริบิวต์ *X* ของตาราง *T*

สำหรับการสอบข้อมูลแบบค่าเท่ากันของดัชนีบิตแมปแบบกระจายพิจารณาได้ดังนี้

$$\text{“X = } v\text{”} = \begin{cases} Z^{\overbrace{v}^{m-1}} \Lambda Z^{\overbrace{v}^{m-1}+1} & \text{if } v \bmod(m\text{-}1) = 0, \\ Z^{\overbrace{v}^{m-1}+1} \Lambda L^{v \bmod(m-1)} & \text{otherwise} \end{cases}$$

สำหรับการสอบถามข้อมูลแบบสมาชิก ทำได้โดยการหาบิตแมปเวกเตอร์ผลลัพธ์ของแต่ละค่ามาดำเนินการตรรกะ OR เช่น เมื่อต้องการสอบถามตามเงื่อนไข “*X* in {0, 3, 5, 6}” ผลลัพธ์ของการสอบถามคือ $(Z^0 \wedge Z^1) + (Z^1 \wedge Z^2) + (Z^2 \wedge L^2) + (Z^2 \wedge Z^3)$ นั่นคือ แถวที่ 3, 5, 6, 7, 8 และ 9 เป็นคำตอบของการสอบถาม

ข้อเด่นของดัชนีบิตแมปแบบกระจาย

การทำดัชนีบิตแมปแบบกระจายใช้พื้นที่ในการจัดเก็บดัชนีน้อยกว่าดัชนีบิตแมปแบบช่วงเมื่อ *C* มีค่ามากกว่าหรือเท่า 20 และในการสอบถามข้อมูลแบบค่าเท่ากันของดัชนีบิตแมปแบบกระจายใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์เหมือนดัชนี

บิตแมปแบบช่วง และสำหรับการสอบถามแบบค่าสมาชิกมีโอกาสตรวจสอบเพียง 1 บิตแมปเวกเตอร์เมื่อสมาชิกที่ต้องการค้นหาอยู่ภายในกลุ่ม *Z* หรือ *L* เดียวกัน

ข้อด้อยของดัชนีบิตแมปแบบกระจาย

เช่นเดียวกับดัชนีบิตแมปแบบช่วง กล่าวคือในการสอบถามข้อมูลแบบค่าเท่ากันและแบบสมาชิกของดัชนีบิตแมปแบบกระจายด้อยประสิทธิภาพกว่าดัชนีบิตแมปแบบพื้นฐาน เนื่องจากการสอบถามแต่ละค่าของดัชนีบิตแมปแบบกระจายใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ แต่การสอบถามแต่ละค่าของดัชนีบิตแมปแบบพื้นฐานตรวจสอบเพียง 1 บิตแมปเวกเตอร์เท่านั้น

3.3.3 ดัชนีบิตแมปแบบคู่กัน

จากเทคนิคการทำดัชนีบิตแมปแบบกระจาย จะเห็นว่ายังสามารถลดพื้นที่ได้อีก จึงมีการนำเสนอการทำดัชนีบิตแมปแบบคู่กัน (Wattanakitrungroj and Vanichayobon; 2006) ในการทำดัชนีบิตแมปแบบคู่กันบนแอทริบิวที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ *C* ประกอบด้วย $\lceil \sqrt{2C + 0.25} + 0.5 \rceil$ บิตแมปเวกเตอร์ คือ D^0, D^1, D^2,..., $D^{\lceil \sqrt{2C+0.25}+0.5 \rceil -1}$ ในการแทนค่าข้อมูลของแอทริบิวต์บนบิตแมปเวกเตอร์ จะกำหนดให้บิตที่ *i* ของบิตแมปเวกเตอร์ D^j มีค่าเป็น 1 ก็ต่อเมื่อ $j = r$ หรือ $j = r - 1 - \left(v - \frac{(n-r)(n-r-1)}{2} \right)$ และเป็น 0 ในกรณีอื่น โดยที่ $r = \lceil \sqrt{2(hiC - v) + 0.24} - 0.5 \rceil$ เมื่อ v คือ ค่าข้อมูลในแถวที่ *i* ของแอทริบิวต์ที่เลือกมาทำดัชนีบิตแมปแบบคู่กัน และ $hiC = \binom{n}{2}$ เมื่อ n คือ จำนวนบิตแมปเวกเตอร์ ตัวอย่างการทำดัชนีบิตแมปแบบคู่กันดังภาพประกอบ 3-10 ซึ่งเป็นการทำดัชนีบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ 8 จึงต้องใช้ 5 บิตแมปเวกเตอร์ ได้แก่บิตแมปเวกเตอร์ D^0, D^1, D^2, D^3, D^4 ซึ่งมีรูปแบบการลงรหัสดังภาพประกอบ 3-13(ง)

RID	...	*X*	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
. . .		. . .	
10000		4	

(ก) ตาราง T

D^0	D^1	D^2	D^3	D^4
0	0	1	0	1
0	1	0	0	1
0	0	0	1	1
0	1	1	0	0
0	0	0	1	1
1	0	0	1	0
1	0	0	0	1
1	0	0	0	1
0	1	0	1	0
. . .	. . .	. . .	. . .	. . .
0	0	1	1	0

(ข) ดัชนีบิตแมปแบบคู่กัน

ภาพประกอบ 3-10 ตัวอย่างการทำดัชนีบิตแมปแบบคู่กันบนแอทริบิวต์ *X* ของตาราง *T*

สำหรับการสอบข้อมูลแบบค่าเท่ากันของการทำดัชนีบิตแมปแบบคู่กันพิจารณาได้ดังนี้

$$\text{“X = } v\text{”} = D^{r} \wedge D^{r-1-\left(v-\frac{(n-r)(n-r-1)}{2}\right)}$$

โดยที่ $r = \left\lceil \sqrt{2(hiC - v) + 0.24} - 0.5 \right\rceil$ และ $hiC = \binom{n}{2}$

สำหรับการสอบถามข้อมูลแบบสมาชิก ทำได้โดยการหาบิตแมปเวกเตอร์ผลลัพธ์ของแต่ละค่าแล้วนำมาดำเนินการตรรกะ OR เช่น เมื่อต้องการสอบถามตามเงื่อนไข “X = {0, 3, 5, 6}” ผลลัพธ์ของการสอบถามคือ $(D^3 \wedge D^4)$ + $(D^0 \wedge D^4)$ + $(D^1 \wedge D^3)$ + $(D^0 \wedge D^3)$ นั่นคือ แถวที่ 3, 5, 6, 7, 8 และ 9 เป็นคำตอบของการสอบถาม

ข้อเด่นของดัชนีบิตแมปแบบคู่กัน

การทำดัชนีบิตแมปแบบคู่กันใช้พื้นที่ในการจัดเก็บดัชนีน้อยกว่าดัชนีบิตแมปแบบช่วงและแบบกระจาย และในการสอบถามข้อมูลแบบค่าเท่ากันของดัชนีบิตแมปแบบคู่ยังคงใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์เหมือนดัชนีบิตแมปแบบช่วงและแบบกระจาย

ข้อด้อยของดัชนีบิตแมปแบบคู่กัน

เช่นเดียวกับดัชนีบิตแมปแบบช่วงและแบบกระจาย กล่าวคือในการสอบถามข้อมูลแบบค่าเท่ากันและแบบสมาชิกของดัชนีบิตแมปแบบคู่กันด้อยประสิทธิภาพกว่าดัชนีบิตแมปแบบพื้นฐาน เนื่องจากการสอบถามแต่ละค่าของดัชนีบิตแมปแบบคู่กันใช้การดำเนินการตรรกะระหว่าง 2 บิตแมปเวกเตอร์ แต่การสอบถามแต่ละค่าของดัชนีบิตแมปแบบพื้นฐานตรวจสอบเพียง 1 บิตแมปเวกเตอร์เท่านั้น

3.3.4 ดัชนีบิตแมปแบบเข้ารหัส

การทำดัชนีบิตแมปแบบเข้ารหัส (Wu and Buchmann, 1998) บนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ C ประกอบด้วย $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์ คือ $E^{\lceil \log_2 C \rceil -1},...,E^2,E^1,E^0$ และตารางเทียบค่า (Mapping table) โดยที่แต่ละค่าของแอทริบิวต์จะถูกเข้ารหัสในรูปแบบ $E^{\lceil \log_2 C \rceil -1}...E^2E^1E^0$ เมื่อ $E^i \in \{B_i, \quad B_i^{'}\}$ โดยให้ $B_i = 1$, $B_i^{'} = 0$ และ $i = 0, 1, 2, ..., \lceil \log_2 C \rceil -1$

ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสดังภาพประกอบ 3-11 เป็นการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T โดยแอทริบิวต์ X ประกอบด้วยสมาชิกคือ {0,1,2,3,4,5,6,7} จะเห็นได้ว่ามีค่าคาร์ดินอร์ลิตี้เท่ากับ 8 จึงมีการใช้ 3 บิตแมปเวกเตอร์ ($\lceil \log_2 8 \rceil = 3$) ได้แก่ บิตแมปเวกเตอร์ E^2,E^1 และ E^0 ซึ่งมีรูปแบบการลงรหัสดังภาพประกอบ 3-13(จ) (โดยจุดสีเข้มแทนค่าบิต 1 และจุดสีอ่อนแทนค่าบิต 0) ในการแทนค่าของแอทริบิวต์แต่ละค่า เช่น จากตารางการเทียบค่า 0 ถูกเข้ารหัสด้วย 000 หรือ $B_2^{'}B_1^{'}B_0^{'}$

RID	...	*X*	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
.		.	
.		.	
.		.	
10000		4	

E^2	E^1	E^0
0	0	1
0	1	0
0	0	0
1	1	1
0	0	0
1	1	0
0	1	1
0	1	1
1	0	1
.	.	.
.	.	.
.	.	.
1	0	0

Mapping table

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

(ก) ตาราง *T* (ข) ดัชนีบิตแมปแบบเข้ารหัสและตารางเทียบค่า

ภาพประกอบ 3-11 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ *X* ของตาราง *T*

สำหรับการสอบถามข้อมูลแบบค่าเท่ากันต้องตรวจสอบกับตารางการเทียบค่าก่อนว่าแต่ละค่าที่ต้องการสอบถามถูกเข้ารหัสในรูปแบบใด แล้วจึงนำไปตรวจสอบกับบิตแมปเวกเตอร์ว่าตรงกับแถวใด จะได้แถวนั้นมีค่าของแอทริบิวต์ตรงกับที่ต้องการ ตัวอย่างเช่น ต้องการสอบถาม “*X* = 0” จากตาราง *T* ดังภาพประกอบ 3-11 ต้องตรวจสอบค่า 0 จากตารางเทียบค่าจะได้ว่าค่า 0 ถูกเข้ารหัสด้วย 000 หรือ $B_2^{'}B_1^{'}B_0^{'}$ จากนั้นจึงนำไปตรวจสอบกับบิตแมปแบบเวกเตอร์ จะได้ว่าแถวที่ 3 และ 5 มีบิตแมปเวกเตอร์ $E^2 = 0$ $E^1 = 0$ และ $E^0 = 0$ ดังนั้นแถวที่ 3 และแถวที่ 5 เป็นคำตอบของการสอบถามนี้

สำหรับการสอบถามข้อมูลแบบสมาชิก เริ่มจากการหาฟังก์ชันการเข้าถึงข้อมูลของการสอบถาม โดยการนำรูปแบบการเข้ารหัสของแต่ละสมาชิก มาดำเนินการตรรกะ OR จากนั้นจะทำการลดรูปฟังก์ชันการเข้าถึงข้อมูลที่หาได้ เพื่อลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบ พิจารณาตัวอย่าง เมื่อต้องการสอบถามข้อมูลแบบสมาชิกตามเงื่อนไข “X in {0,3,5,6}” โดยใช้ตารางการเทียบค่าจากภาพประกอบ 3-11 จะได้ฟังก์ชันการเข้าถึงข้อมูลเป็น $B_2^{'}B_1^{'}B_0^{'} + B_2^{'}B_1B_0 + B_2B_1^{'}B_0 + B_2B_1B_0^{'}$ จะเห็นว่าไม่สามารถลดรูปฟังก์ชันการเข้าถึงข้อมูลได้ จึงต้องตรวจสอบทุกบิตแมปเวกเตอร์ของแต่ละค่า อย่างไรก็ตาม ถ้าเรามีตารางการเทียบค่าดังภาพประกอบ 3-12 หากเปรียบเทียบกับการสอบถามที่เหมือนกัน จะได้ฟังก์ชันการเข้าถึงข้อมูลเป็น $B_2^{'}B_1^{'}B_0^{'} + B_2^{'}B_1^{'}B_0 + B_2^{'}B_1B_0 + B_2^{'}B_1B_0^{'}$ ซึ่งสามารถลดรูปฟังก์ชันการเข้าถึงข้อมูล

ได้เป็น $B_2^{'}(B_1^{'}(B_0^{'}+B_0)+B_1(B_0+B_0^{'}))=B_2^{'}$ เพราะฉะนั้นตรวจสอบเพียงบิตแมปเวกเตอร์ $B_2^{'}$ นั่นคือตรวจสอบบิตแมปเวกเตอร์ E^2 ที่มีค่าบิตเท่ากับ 0 ก็สามารถได้คำตอบของการสอบถาม

RID	...	X	...
1		1	
2		2	
3		0	
4		7	
5		0	
6		6	
7		3	
8		3	
9		5	
.		.	
.		.	
.		.	
10000		4	

(ก) ตาราง T

E^2	E^1	E^0
1	0	0
1	0	1
0	0	0
1	1	0
0	0	0
0	1	0
0	0	1
0	0	1
0	1	1
.	.	.
.	.	.
.	.	.
1	1	1

Mapping table

0	000
1	100
2	101
3	001
4	111
5	011
6	010
7	110

(ข) ดัชนีบิตแมปแบบเข้ารหัสและตารางเทียบค่า

ภาพประกอบ 3-12 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ *X* ของตาราง *T* โดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย

ข้อเด่นของดัชนีบิตแมปแบบเข้ารหัส

ดัชนีบิตแมปแบบเข้ารหัสเป็นเทคนิคการทำดัชนีบิตแมปที่มีการใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนีน้อยที่สุด (มีประสิทธิภาพสูงที่สุดในการใช้พื้นที่) เนื่องจากดัชนีบิตแมปแบบเข้ารหัสใช้จำนวนบิตแมปเวกเตอร์ในการสร้างดัชนีน้อยกว่าดัชนีบิตแมปแบบพื้นฐาน แบบช่วง แบบกระจาย และแบบคู่กัน สรุปดังตารางที่ 3-1 โดยที่รูปแบบการลงรหัสของดัชนีบิตแมปแต่ละแบบเป็นดังภาพประกอบ 3-13 นอกจากนี้หากดัชนีบิตแมปแบบเข้ารหัสมีการเข้ารหัสในรูปแบบที่ดี (Well-defined encoding) จะทำให้มีประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกด้วย

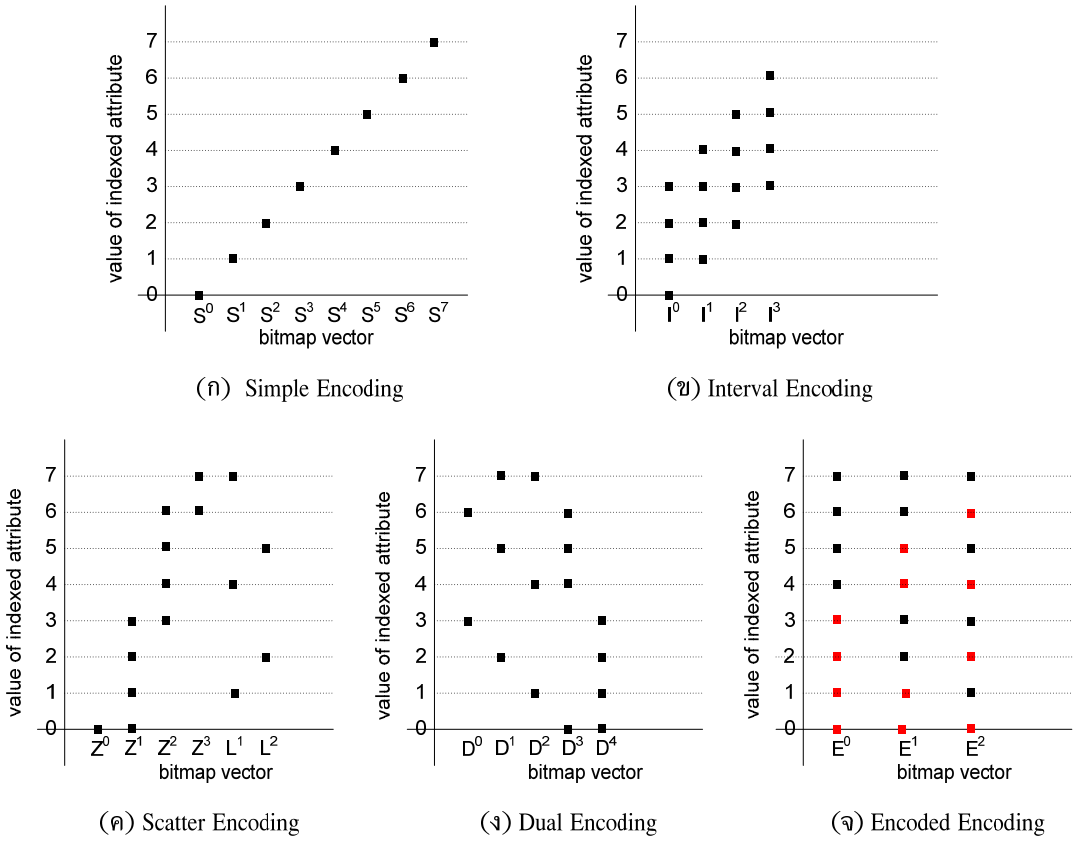
ข้อด้อยของดัชนีบิตแมปแบบเข้ารหัส

ดัชนีบิตแมปแบบเข้ารหัสไม่เหมาะกับการสอบถามข้อมูลแบบค่าเท่ากัน เนื่องจากมีการเปรียบเทียบกับทุกบิตของบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) และตรวจสอบกับตารางเทียบค่า และหากการเข้ารหัสมีรูปแบบที่ไม่ดี การสอบถามข้อมูลแบบ

สมาชิกจะไม่มีประสิทธิภาพด้วย ดังนั้นหากเรามีการเข้ารหัสในรูปแบบที่ดี จะสามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามข้อมูลได้ โดยไม่ต้องตรวจสอบทุกบิตแมปเวกเตอร์ในแต่ละค่าของสมาชิก จะทำให้เวลาที่ใช้ในการสอบถามน้อยลง นั่นคือจะทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น

3.4 เปรียบเทียบดัชนีบิตแมปแบบไม่บีบอัดทั้ง 5 ชนิด

จากที่ได้กล่าวมา ดัชนีบิตแมปแต่ละชนิดมีรูปแบบการลงรหัสและการสอบถามที่แตกต่างกัน จึงมีข้อเด่นและข้อด้อยแตกต่างกันออกไป พิจารณารูปแบบการลงรหัสที่แตกต่างกันของดัชนีบิตแมปแต่ละชนิดดังภาพประกอบ 3-13 จากรูปแบบการลงรหัสและการสอบถามแบบสมาชิกของดัชนีบิตแมปแต่ละชนิด สามารถสรุปความต้องการพื้นที่ที่ใช้ในการจัดเก็บดัชนีและเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแต่ละชนิด แสดงดังตาราง 3-1 โดยพื้นที่ที่ใช้ในการจัดเก็บดัชนี พิจารณาจากจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนี และเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิก พิจารณาจากจำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบและจำนวนครั้งในการดำเนินการตรรกะ



ภาพประกอบ 3 -13 แผนภาพการลงรหัสของดัชนีบิตแมปทั้ง 5 ชนิด เมื่อ $C = 8$

ตาราง 3-1 เปรียบเทียบประสิทธิภาพของดัชนีบิตแมปทั้ง 5 ชนิด (เมื่อ C คือ คาร์ดินอร์ลิตี้ และ k คือ จำนวนสมาชิกที่ถูกสอบถาม)

ดัชนีบิตแมป	พื้นที่	เวลา	
	จำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนี	จำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบ: จำนวนครั้งในการดำเนินการตรรกะ	
		การสอบถามแบบค่าเท่ากัน	การสอบถามแบบสมาชิก
แบบพื้นฐาน	C	1:0	k: k-1 (OR)
แบบช่วง	$\lceil C/2 \rceil$	2:2 (1AND,1NOT)	$2k$: $3k$-1 (k AND, k-1OR, k NOT)
แบบกระจาย	$\lceil 2\sqrt{C} \rceil$	2:1 (1AND)	$2k$: $2k$-1(k AND, k-1OR)
แบบคู่กัน	$\lceil \sqrt{2C + 0.25} + 0.5 \rceil$	2:1 (1AND)	$2k$: $2k$-1(k AND, k-1OR)
แบบเข้ารหัส	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$: use mapping table	$k\lceil \log_2 C \rceil$: use mapping table (k-1 OR)

จากตาราง 3-1 เมื่อพิจารณาพื้นที่ที่ใช้ในการจัดเก็บดัชนี จะเห็นว่าดัชนีบิตแมปแบบเข้ารหัสใช้พื้นที่ในการจัดเก็บดัชนีน้อยที่สุด คือ ใช้ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์ ส่วนดัชนีบิตแมปที่ใช้พื้นที่ในการจัดเก็บมากที่สุด คือ ดัชนีบิตแมปแบบพื้นฐาน ซึ่งใช้ C บิตแมปเวกเตอร์ และเมื่อพิจารณาเวลาที่ใช้ในการสอบถามแบบสมาชิก จะเห็นว่าดัชนีบิตแมปแบบพื้นฐานใช้เวลาน้อยที่สุด ส่วนดัชนีบิตแมปแบบเข้ารหัสใช้เวลามากที่สุด

จากเทคนิคการทำดัชนีบิตแมปแบบไม่มีการบีบอัดดัชนีตามที่กล่าวมาจะเห็นว่าดัชนีบิตแมปแบบเข้ารหัสมีประสิทธิภาพสูงที่สุดในการใช้พื้นที่ในการจัดเก็บดัชนี และหากมีการเข้ารหัสในรูปแบบที่ดีจะทำให้ประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกเพิ่มขึ้นด้วย ซึ่งการเข้ารหัสแต่ละค่าของแอทริบิวต์ในตารางเทียบค่าเป็นหัวใจของการลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามข้อมูลแบบสมาชิก ดังนั้นหากเราสามารถเข้ารหัสค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อย ให้อยู่ในรูปแบบที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามข้อมูลได้ ก็จะเป็นการเพิ่มประสิทธิภาพในการสอบถามข้อมูลกล่าวคือสามารถลดเวลาในการค้นหาข้อมูลได้มาก ซึ่งในการหาค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อยนั้น เราสามารถนำเอาเทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets mining) มาช่วยในการจัดลำดับค่าของแอทริบิวต์ก่อนเข้ารหัสเพื่อสร้างตารางเทียบค่าที่นำไปสู่การลดรูปบิตแมปเวกเตอร์เมื่อมีการสอบถามแบบสมาชิก

บทที่ 4

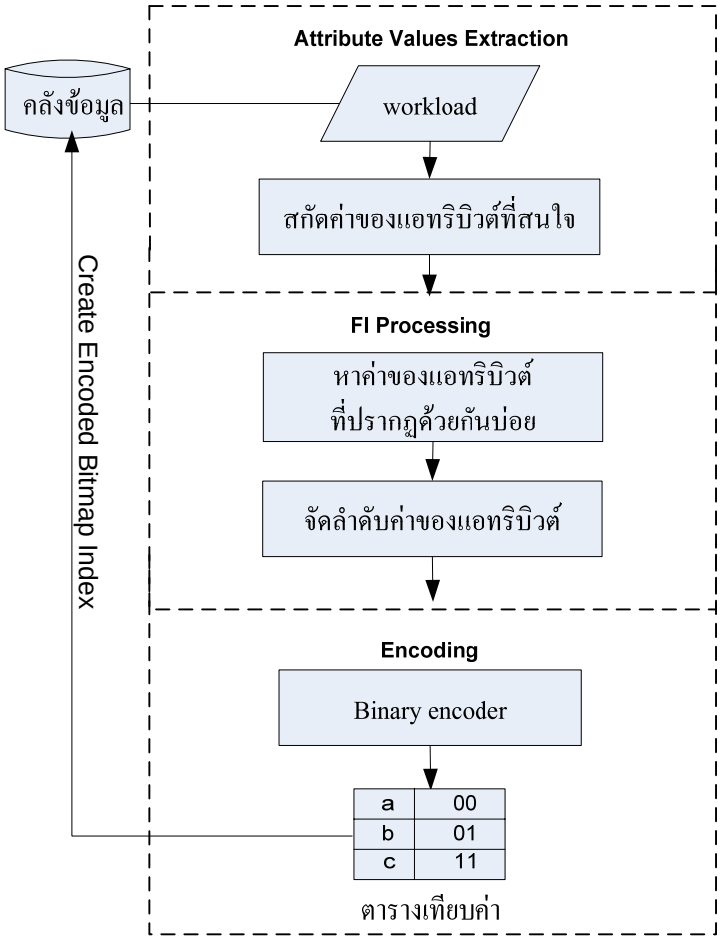
การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส

หัวใจสำคัญของดัชนีบิตแมปแบบเข้ารหัสคือ การเข้ารหัสแต่ละค่าของแอทริบิวต์ ซึ่งการเข้ารหัสที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามข้อมูลให้เหลือน้อยที่สุดถือเป็นการเข้ารหัสที่ดี ทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น กล่าวคือสามารถลดเวลาที่ใช้ในการสอบถามข้อมูลได้

วิทยานิพนธ์นี้จึงนำกลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets) มาช่วยในการเข้ารหัส โดยนำเทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets mining) มาใช้ในการหาค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อยจากชุดข้อมูลการสอบถามในอดีต (Workload) เพื่อเข้ารหัสค่าของแอทริบิวต์ที่หาได้ให้สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามแบบสมาชิกด้วยกัน

4.1 ขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย

ขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อยประกอบด้วย 3 ขั้นตอนหลัก คือ ขั้นตอนที่ 1 Attribute Values Extraction ขั้นตอนที่ 2 FI Processing และขั้นตอนที่ 3 Encoding แสดงดังภาพประกอบ 4-1 โดยแต่ละขั้นตอนมีรายละเอียดดังต่อไปนี้



ภาพประกอบ 4-1 ขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัส

ขั้นตอนที่ 1 : Attribute Values Extraction

Attribute Values Extraction เป็นขั้นตอนการสกัดค่าของแอทริบิวต์บนแอทริบิวต์ที่สนใจจะนำมาสร้างดัชนีจากการสอบถามทั้งหมด (Workload) โดยการพิจารณาค่าของแอทริบิวต์ที่อยู่หลังเงื่อนไข Where หรือ Having ได้ผลลัพธ์เป็นตารางค่าของแอทริบิวต์ ในตารางค่าของ แอทริบิวต์แสดงให้เห็นว่าแต่ละทรานแซคชัน หรือแต่ละการสอบถามมีการสอบถามค่าของ แอทริบิวต์บนแอทริบิวต์ที่นำมาสร้างดัชนีค่าใดบ้าง ภาพประกอบ 4-2 แสดงตัวอย่าง Workload และภาพประกอบ 4-3 แสดงตัวอย่างตารางค่าของแอทริบิวต์ *X* ที่สกัดจาก Workload ในภาพประกอบ 4-2 เมื่อกำหนดให้แอทริบิวต์ *X* ประกอบด้วยสมาชิก 16 ตัว คือ {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P}

```
Q1 : SELECT * FROM  T WHERE  X  IN (A,C,E,G,O,H,J,K,P)
Q2 : SELECT * FROM  T WHERE  X  IN (B,D,F,I)
Q3 : SELECT * FROM  T WHERE  X  IN (A,C,E,G,O,H,J,K,M,N)
Q4 : SELECT * FROM  T WHERE  X  IN (A,C,E,G,O,H,J,K)
Q5 : SELECT * FROM  T WHERE  X  IN (B,D,F,I ,M,N)
```

ภาพประกอบ 4-2 ตัวอย่างรายการสอบถาม (Workload)

Query	ค่าของแอทริบิวต์ X
Q1	A,C,E,G,O,H,J,K,P
Q2	B,D,F,I
Q3	A,C,E,G,O,H,J,K,M,N
Q4	A,C,E,G,O,H,J,K
Q5	B,D,F,I ,M,N

ภาพประกอบ 4-3 ตัวอย่างตารางค่าของแอทริบิวต์ *X* ที่สกัดจาก Workload

ขั้นตอนที่ 2 : FI Processing

FI (Frequent Itemsets) Processing เป็นขั้นตอนการหากลุ่มข้อมูลที่ปรากฏบ่อย ในที่นี้หมายถึงค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันบ่อย เพื่อนำไปสู่การเข้ารหัสสำหรับสร้างตารางเทียบค่าที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามแบบสมาชิก ขั้นตอนนี้มี 2 ขั้นตอนย่อยคือ การหากลุ่มข้อมูลที่ปรากฏบ่อยและการจัดลำดับค่าของแอทริบิวต์

ขั้นตอนที่ 2.1 : การหากลุ่มข้อมูลที่ปรากฏบ่อย

ขั้นตอนนี้เป็นการหาค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อยจากตารางค่าของแอทริบิวต์ที่ถูกสร้างในขั้นตอนที่ 1 โดยใช้อัลกอริทึม EncodedBitmapFI ดังภาพประกอบ 4-4 และนิยามสัญลักษณ์ที่ใช้ในขั้นตอนวิธีนี้ แสดงได้ดังตารางที่ 4-1

อัลกอริทึม : EncodedBitmapFI

ข้อมูลเข้า : ตารางค่าของแอทริบิวต์และค่าความถี่ขั้นต่ำ (Minimum Support)

ข้อมูลออก: ค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อย (F_k, $k = 2^i$, $i = 1, 2, \ldots, \lceil \log_2 C \rceil - 1$)

1) อ่านตารางค่าของแอทริบิวต์หนึ่งครั้งเพื่อสร้างตาราง BitMatrix
2) หาความถี่ (Support) ของแต่ละ Items โดยการนับจำนวนบิตที่มีค่า 1 ของแต่ละ Item
3) เรียงลำดับ Items ในตาราง BitMatrix ตามค่าความถี่จากน้อยไปมาก
4) เลือกเฉพาะ Items ที่มีค่าความถี่มากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำที่กำหนด
5) For ($k = 2^{\lceil \log C \rceil - 1}$; $k \geq 2$; $k = k/2$)
6) While($N \geq k$)
7) $pointer = N - k$;
8) While ($pointer \geq 0$)
9) $Delegate_k.item_{pointer}$= Gen_Delegate($Item_{pointer}$);
10) If ($Delegate_k.item_{pointer} \neq \phi$)
11) $F_k = F_k \cup Delegate_k.item_{pointer}$;
12) Remove $Delegate_k.item_{pointer}$ from BitMatrix table;
13) $N = N-k$;
14) Exit Loop;
15) Else
16) If ($Support.item_{pointer} > MinSup$)
17) $C_k.item_{pointer}$ = Gen_Candidate($item_{pointer}$,$tail$)
18) If ($C_k.item_{pointer} \neq \phi$)
19) $F_k = F_k \cup C_k.item_{pointer}$;
20) Remove $C_k.item_{pointer}$ from BitMatrix table;
21) $N = N-k$;
22) Exit Loop;
23) End if
24) End if
25) End if
26) $pointer = pointer - 1$;
27) End while
28) End while
29) End for
30) Return F_k

ภาพประกอบ 4-4 อัลกอริทึม EncodedBitmapFI

ตารางที่ 4-1 นิยามสัญลักษณ์ที่ใช้ในอัลกอริทึม EncodedBitmapFI

Item	ค่าของแอทริบิวต์
Itemsets	ค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อย
pointer	ตำแหน่งของ Item ที่กำลังพิจารณา โดย $0 \le pointer < N$
$Item_{pointer}$	Item ที่กำลังพิจารณา
N	จำนวน Items ในตาราง BitMatrix ณ ปัจจุบัน
$Delegate_k.item_{pointer}$	เซตของ Items ตั้งแต่ $Item_{pointer}$ เป็นต้นไปที่ปรากฏพร้อม $Item_{pointer}$ และอยู่ทางขวาของ $Item_{pointer}$ ในตาราง BitMatrix ณ ปัจจุบัน
C_k	Candidate itemsets ขนาด *k*
F_k	Frequent itemsets ขนาด *k*
tail	Items ทั้งหมดที่อยู่ทางขวาของ $Item_{pointer}$ ในตาราง BitMatrix ณ ปัจจุบัน
$Support.item_{pointer}$	ค่าความถี่ของ $Item_{pointer}$
MinSup	ค่าความถี่ขั้นต่ำที่กำหนด (Minimum Support)

ขั้นตอนการทำงานของอัลกอริทึม EncodedBitmapFI จากภาพประกอบ 4-4 มีรายละเอียดของแต่ละบรรทัดดังนี้

บรรทัดที่ 1 อ่านตารางค่าของแอทริบิวต์หนึ่งครั้งเพื่อสร้างตาราง BitMatrix ตัวอย่างเช่น จากตารางค่าของแอทริบิวต์ดังภาพประกอบ 4-3 สามารถสร้างตาราง BitMatrix ได้ดังภาพประกอบ 4-5 เมื่อตาราง BitMatrix มีขนาด *R* x *C* โดยที่ *R* คือ จำนวนการสอบถามทั้งหมดใน Workload และ *C* คือ ค่าคาร์ดินอร์ลิตี้ของแอทริบิวต์ที่นำมาสร้างดัชนี ซึ่งแต่ละแถวแสดงให้เห็นว่าการสอบถามข้อมูล *Qi* มีการสอบถามค่าของแอทริบิวต์ใดบ้าง และในแต่ละคอลัมน์แสดงให้เห็นว่ามี $Item_j$ ถูกสอบถามในการสอบถามใดบ้าง โดย BitMatrix[*i,j*] มีค่าเท่ากับ 1 หมายความว่า *Qi* มีการสอบถาม $Item_j$ (เมื่อ $1 \le i \le R$ และ $0 \le j \le C-1$) ตัวอย่างเช่น จากภาพประกอบ 4-5 BitMatrix[3,0] มีค่าเท่ากับ 1 หมายความว่า *Q3* มีการสอบถาม Item A และ BitMatrix[3,1] มีค่าเท่ากับ 0 หมายความว่า *Q3* ไม่มีการสอบถาม Item B ประโยชน์ของการสร้างตาราง BitMatrix คือ การอ่านตารางค่าของแอทริบิวต์เพียงครั้งเดียวในการหาค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อยขนาด *k* ถ้าใช้เทคนิคที่เคยมีมา เช่น อัลกอริทึม Apriori จะต้องเสียเวลาอ่านตารางค่าของแอทริบิวต์ *k* ครั้ง หรืออัลกอริทึม FP-growth จะต้องเสียเวลาอ่านตารางค่าของแอทริบิวต์ 2 ครั้ง และสามารถดำเนินการตรรกะระดับบิตในการหากลุ่มข้อมูลที่ปรากฏบ่อย ซึ่งเป็นการเพิ่มประสิทธิภาพในการประมวลผลอีกด้วย

Query	ค่าของแอทริบิวต์ *X*															
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
Q1	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	1
Q2	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0
Q3	1	0	1	0	1	0	1	1	0	1	1	0	1	1	1	0
Q4	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	0
Q5	0	1	0	1	0	1	0	0	1	0	0	0	1	1	0	0

ภาพประกอบ 4-5 ตัวอย่างตาราง BitMatrix

บรรทัดที่ 2 หาค่าความถี่ของแต่ละ Item โดยการนับจำนวนบิต 1 ของแต่ละ Item จากตาราง BitMatrix ตัวอย่างเช่น จากภาพประกอบ 4-5 Item A มีบิต 1 ทั้งหมด 3 บิต ค่าความถี่ของ Item A จึงเท่ากับ 3 และสามารถหาค่าความถี่ของ Item อื่น ๆ แสดงได้ดังภาพประกอบ 4-6

Item	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
support	3	2	3	2	3	2	3	3	2	3	3	0	2	2	3	1

ภาพประกอบ 4-6 ค่าความถี่ของแต่ละ Item

บรรทัดที่ 3 เรียงลำดับ Items ในตาราง BitMatrix ตามค่าความถี่จากน้อยไปมาก สำหรับ Items ที่มีค่าความถี่เท่ากันให้เรียงตามลำดับตัวอักษร ตัวอย่างเช่น จากตาราง BitMatrix ดังภาพประกอบ 4-5 เมื่อเรียงลำดับ Items ตามความถี่ของแต่ละ Item จากน้อยไปมากจะได้ผลลัพธ์แสดงดังภาพประกอบ 4-7 เมื่อตัวเลขที่อยู่หลังเครื่องหมาย “:” หมายถึงค่าความถี่ของ Item นั้น

Query	ค่าของแอทริบิวต์ X														
	P:1	B:2	D:2	F:2	I:2	M:2	N:2	A:3	C:3	E:3	G:3	H:3	J:3	K:3	O:3
Q1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Q2	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Q3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Q4	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Q5	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0

ภาพประกอบ 4-7 ตาราง BitMatrix เมื่อผ่านการเรียงลำดับตามค่าความถี่ของแต่ละ Item

บรรทัดที่ 4 เลือก Items ที่มีค่าความถี่น้อยกว่าค่าความถี่ขั้นต่ำที่กำหนดออกจากตาราง BitMatrix ตัวอย่างเช่น เมื่อกำหนดให้ค่าความถี่ขั้นต่ำเท่ากับ 40% จากภาพประกอบ 4-7 จะได้ผลลัพธ์ดังแสดงในภาพประกอบ 4-8 จะเห็นว่าค่า Item P ถูกเลือกออก เนื่องจากมีค่าความถี่เท่ากับ 1 หรือ 20% ซึ่งน้อยกว่าค่าความถี่ขั้นต่ำที่กำหนด

Query	ค่าของแอทริบิวต์ *X*												
	B:2	D:2	F:2	I:2	M:2	N:2	A:3	C:3	E:3	G:3	H:3	J:3	K:3	O:3
Q1	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Q2	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Q3	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Q4	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Q5	1	1	1	1	1	1	0	0	0	0	0	0	0	0

ภาพประกอบ 4-8 ตาราง BitMatrix เมื่อเลือก Item ที่มีค่าความถี่น้อยกว่าค่าความถี่ขั้นต่ำออก

บรรทัดที่ 5-29 เป็นการหากลุ่มข้อมูลที่ปรากฏบ่อยขนาด 2^i เมื่อ i = 1, 2, ...,$\lceil \log_2 C \rceil$-1 นั่นคือ จากตัวอย่างข้างต้นแอทริบิวต์ที่นำมาสร้างดัชนีมีค่า C เท่ากับ 16 ดังนั้นกลุ่มข้อมูลที่ต้องการคือ กลุ่มข้อมูลขนาด 8, 4 และ 2 ตามลำดับ โดยกำหนดค่า k เริ่มต้นเท่ากับ $2^{\lceil \log_2 C \rceil - 1}$ นั่นคือ ต้องการหากลุ่มข้อมูลขนาด $2^{\lceil \log_2 C \rceil - 1}$ เป็นลำดับแรก

บรรทัดที่ 6 เป็นการตรวจสอบจำนวน Items ในตาราง BitMatrix ณ ปัจจุบัน หรือค่า *N* ว่ามีค่ามากกว่าหรือเท่ากับ *k* หรือไม่ นั่นคือเป็นการตรวจสอบว่า จากตาราง BitMatrix ณ ปัจจุบัน มีโอกาสหากลุ่มข้อมูลขนาด *k* ที่ต้องการได้หรือไม่ ซึ่งกรณี *N* มีค่ามากกว่าหรือเท่ากับ *k* แสดงว่ามีโอกาสหากลุ่มข้อมูลขนาด *k* ที่ต้องการได้ แต่ถ้า *N* มีค่าน้อยกว่า *k* แสดงว่าไม่มีโอกาสหากลุ่มข้อมูลขนาด *k* ที่ต้องการได้ จะต้องเปลี่ยนค่า *k* เป็น *k/2* แล้วพิจารณาใหม่

บรรทัดที่ 7 เป็นการหาตำแหน่งของ Item ที่จะเริ่มพิจารณาเพื่อหา F_k โดยกำหนดให้ตำแหน่ง (*pointer*) ของ Item ในตาราง BitMatrix มีค่าตั้งแต่ 0 ถึง *N*-1 เมื่อตำแหน่งเริ่มต้นที่จะถูกพิจารณาเพื่อหา F_k ที่ต้องการ คือ ตำแหน่งของ Item ที่มีค่าความถี่มากที่สุดที่มีโอกาสหา F_k ได้ นั่นคือ Item ณ ตำแหน่ง *N-k* ในตาราง BitMatrix

บรรทัดที่ 8 เป็นการตรวจสอบค่า *pointer* ว่าอยู่ในขอบเขตที่ถูกต้องหรือไม่ โดยกำหนดให้ $0 \le pointer < N$

บรรทัดที่ 9 เป็นการหา *Delegate* ขนาด *k* ของ Item ที่กำลังพิจารณา ($Item_{pointer}$) โดยเรียกใช้ฟังก์ชัน Gen_Delegate()

การทำงานของฟังก์ชัน Gen_Delegate() เป็นการดำเนินการตรรกะ AND ระหว่างบิตแมปเวกเตอร์ของทุกการสอบถาม (ทุกทรานแซคชัน) ในตาราง BitMatrix ที่ Item ที่กำลังพิจารณาปรากฏอยู่ โดยเริ่มดำเนินการตรรกะ AND ตั้งแต่ตำแหน่ง Item ที่กำลังพิจารณาเป็นต้นไปทางขวา ผลลัพธ์ของการดำเนินการตรรกะ AND ถ้าบิตของ Item ใดมีค่าเป็น 1 หมายความว่า Item นั้นเป็นสมาชิกใน $Delegate_k.item_{pointer}$ กรณีที่จำนวนสมาชิกใน $Delegate_k.item_{pointer}$ มีมากกว่าหรือเท่ากับ *k* ให้เลือกเพียง *k* Item แต่กรณีที่จำนวนสมาชิกใน *Delegate* มีน้อยกว่า *k* ให้ $Delegate_k.item_{pointer}$ มีค่าเท่ากับเซตว่าง ($Delegate_k.item_{pointer} \neq \phi$)

บรรทัดที่ 10 เป็นการตรวจสอบว่า $Delegate_k.item_{pointer}$ มีค่าเป็นเซตว่างหรือไม่ นั่นคือ เป็นการตรวจสอบว่าสามารถหา *Delegate* ขนาด *k* ของ Item ที่กำลังพิจารณาได้หรือไม่

บรรทัดที่ 11 เป็นการนำ $Delegate_k.item_{pointer}$ ให้เป็นสมาชิกของกลุ่มข้อมูลที่ต้องการ ($F_k = F_k \cup Delegate_k.item_{pointer}$) ซึ่งแสดงว่าเราสามารถหากลุ่มข้อมูลที่ต้องการได้

บรรทัดที่ 12 เป็นการเลือก Items ที่เป็นสมาชิกใน $Delegate_k.item_{pointer}$ ออกจากตาราง BitMatrix ณ ปัจจุบัน เนื่องจากเราต้องการหากลุ่มข้อมูลที่มีค่าสมาชิกไม่ซ้ำกัน ดังนั้นการเลือก Items ที่เป็นสมาชิกของกลุ่มข้อมูลที่หาได้แล้วออกจากตาราง BitMatrix จะทำให้กลุ่มข้อมูลใหม่ที่หาได้มีสมาชิกไม่ซ้ำกับกลุ่มข้อมูลที่เคยหาได้ และยังเป็นการลดเวลาที่ใช้ในการหากลุ่มข้อมูลถัดไป เพราะมี Item ที่นำมาพิจารณาน้อยลง ขนาดของตาราง BitMatrix ก็ลดลงด้วย

บรรทัดที่ 13 เป็นการกำหนดจำนวน Items ที่มีอยู่ในตาราง BitMatrix ณ ปัจจุบันใหม่ เนื่องจากมีการเลือก Items ออกจากตาราง BitMatrix (จากคำสั่งในบรรทัดที่ 12)

บรรทัดที่ 14 แสดงถึงการออกจากลูป โดยการกระโดดไปทำงาน ณ บรรทัดที่ 6 นั่นคือเป็นการกระโดดไปเพื่อเริ่มหากลุ่มข้อมูลถัดไป

บรรทัดที่ 15 เป็นทางเลือกกรณี $Delegate_k.item_{pointer}$ มีค่าเท่ากับเซตว่าง

บรรทัดที่ 16 เป็นการตรวจสอบค่าความถี่ของ Item ที่กำลังพิจารณาว่ามีค่ามากกว่าค่าความถี่ขั้นต่ำหรือไม่ เนื่องจากถ้าค่าความถี่ของ Item ที่กำลังพิจารณามีค่าเท่ากับค่าความถี่ขั้นต่ำ เราไม่จำเป็นต้องทำงานในส่วนของฟังก์ชัน Gen_Candidate() ซึ่งการทำงานในส่วนฟังก์ชัน Gen_Delegate() ก็เพียงพอที่จะบอกได้ว่าไม่สามารถหากลุ่มข้อมูลขนาด ***k*** ที่ต้องการจาก Item ที่กำลังพิจารณานี้ได้ การทำงานส่วนนี้จึงเป็นการช่วยลดเวลาการประมวลผลได้

บรรทัดที่ 17 เป็นการหา Candidate itemsets ขนาด ***k*** ของ Item ที่กำลังพิจารณา ($C_k.Item_{pointer}$) โดยเรียกใช้ฟังก์ชัน Gen_Candidate()

การทำงานของฟังก์ชัน Gen_Candidate() เป็นการดำเนินการตรรกะ AND ระหว่าง ***k*** บิตแมปเวกเตอร์ ได้แก่บิตแมปเวกเตอร์ของ Item ที่กำลังพิจารณากับ Items ที่อยู่ทางขวาของ Item ที่กำลังพิจารณาอีก ***k***-1 บิตแมปเวกเตอร์ และตรวจสอบค่าความถี่จากบิตแมปเวกเตอร์ผลลัพธ์ที่ได้โดยการนับจำนวนบิต 1 กรณีที่มีค่าความถี่น้อยกว่าค่าความถี่ขั้นต่ำ ให้ $C_k.Item_{pointer}$ มีค่าเท่ากับเซตว่าง การทำงานของฟังก์ชันนี้จะสิ้นสุดเมื่อ $C_k.Item_{pointer}$ มีค่าไม่เท่ากับเซตว่าง หรือพิจารณาจนครบทุกเซตที่สามารถเป็นไปได้

บรรทัดที่ 18 เป็นการตรวจสอบว่า $C_k.Item_{pointer}$ มีค่าไม่เท่ากับเซตว่างหรือไม่ นั่นคือเป็นการตรวจสอบว่าสามารถหา Candidate itemsets ขนาด ***k*** ของ Item ที่กำลังพิจารณาได้หรือไม่

บรรทัดที่ 19 เป็นการนำ $C_k.Item_{pointer}$ ให้เป็นสมาชิกของกลุ่มข้อมูลที่ต้องการ ($F_k = F_k \cup C_k.Item_{pointer}$) ซึ่งแสดงว่าเราสามารถหากลุ่มข้อมูลที่ต้องการได้

บรรทัดที่ 20 เป็นการเลือก Items ที่เป็นสมาชิกใน $C_k.Item_{pointer}$ ออกจากตาราง BitMatrix ณ ปัจจุบัน เช่นเดียวกับการทำงานในบรรทัดที่ 12

บรรทัดที่ 21 เป็นการกำหนดจำนวน Items ที่มีอยู่ในตาราง BitMatrix ณ ปัจจุบันใหม่ เนื่องจากมีการเลือก Items ออกจากตาราง BitMatrix (จากคำสั่งในบรรทัดที่ 20) เช่นเดียวกับการทำงานในบรรทัดที่ 13

บรรทัดที่ 22 แสดงถึงการออกจากลูป โดยการกระโดดไปทำงาน ณ บรรทัดที่ 6 นั่นคือเป็นการกระโดดไปเพื่อเริ่มหากลุ่มข้อมูลถัดไป

บรรทัดที่ 26 เป็นการเลื่อน *pointer* ไปทางซ้ายเพื่อพิจารณา Item ถัดไป เนื่องจากไม่สามารถหากลุ่มข้อมูลที่ต้องการได้จาก Item ที่กำลังพิจารณา

การทำงานของอัลกอริทึม EncodedBitmapFI จะหยุดดำเนินการก็ต่อเมื่อไม่มี Item เพียงพอที่จะหา F_2 ได้อีก หรือพิจารณาจนครบทุก F_k ที่ต้องการแล้ว โดยผลลัพธ์ที่ได้เมื่อสิ้นสุดการทำงานคือ กลุ่มข้อมูลขนาด 2^i ที่แต่กลุ่มข้อมูลมีค่าสมาชิกไม่ซ้ำกัน

ตัวอย่างการหากลุ่มข้อมูลตามขั้นตอนวิธีข้างต้น โดยใช้ตาราง BitMatrix จากภาพประกอบ 4-8 เนื่องจากคาร์ดินอร์ลิตี้ของแอทริบิวต์ X ที่นำมาสร้างดัชนีมีค่าเท่ากับ 16 (C = 16) ดังนั้นจะมีการทำงานทั้งหมด 3 ($\lceil \log_2 16 \rceil$-1 = 3) รอบดังนี้ ในรอบที่ 1 เป็นการหากลุ่มข้อมูลขนาด 8 Items (F_8) เริ่มด้วยการตรวจสอบจำนวน Items ที่มีอยู่ในตาราง BitMatrix พบว่า มีจำนวน Items ทั้งหมดเท่ากับ 14 (N=14) ซึ่งมากกว่า 8 แสดงว่ามี Items เพียงพอที่จะหา F_8 จากนั้นหาตำแหน่งของ Item ที่จะเริ่มพิจารณาเพื่อหา F_8 จากภาพประกอบ 4-8 จะได้ว่า Item B อยู่ตำแหน่งที่ 0 และ Item D อยู่ตำแหน่งที่ 1 เป็นต้น ซึ่งตำแหน่งเริ่มต้นที่จะพิจารณา คือ Item ที่มีค่าความถี่มากที่สุดที่มีโอกาสสร้าง F_8 ได้ นั่นคือ Item ที่อยู่ ณ ตำแหน่ง $N - k$ ดังนั้นจากตาราง BitMatrix ภาพประกอบ 4-8 จะได้ว่า Item ตำแหน่งที่ 6 (14-8) ซึ่งคือ Item A เป็นตำแหน่งเริ่มต้นที่จะพิจารณาเพื่อหา F_8 จากนั้นหา *Delegate* ของ Item A จากภาพประกอบ 4-8 Item A ปรากฏอยู่ใน $Q1$, $Q3$ และ $Q4$ จึงต้องดำเนินการตรรกะ AND ระหว่าง $Q1$, $Q3$ และ $Q4$ ตั้งแต่บิตของ Item A ถึง Item O จะได้บิตแมปเวกเตอร์ของ $Q1$ คือ 11111111 บิตแมปเวกเตอร์ของ $Q3$ คือ 11111111 และบิตแมปเวกเตอร์ของ $Q4$ คือ 11111111 ดังนั้นผลลัพธ์ที่ได้คือ 11111111 AND 11111111 AND 11111111 = 11111111 จากผลลัพธ์ของการดำเนินการตรรกะ AND ถ้าบิตของ Item ใดมีค่าเป็น 1 จะได้ว่าเป็นสมาชิกใน *Delegate* นั้น ดังนั้น *Delegate* ของ Item A คือ {A,C,E,G,H,J,K,O} ซึ่งมีจำนวนสมาชิกเท่ากับ 8 จึงถือว่าเป็นกลุ่มข้อมูลขนาด 8 (Frequent 8-itemsets) ที่ต้องการ ดังนั้นจะได้ F_8 = {A,C,E,G,H,J K,O} จากนั้นเลือก Items เหล่านี้ออกจากตาราง BitMatrix จะเหลือ Items ในตาราง BitMatrix จำนวน 6 Items แสดงดังภาพประกอบ 4-9 ซึ่งไม่สามารถหา F_8 ได้อีก

ในรอบที่ 2 เป็นการหากลุ่มข้อมูลขนาด 4 (F_4)โดยจะเริ่มพิจารณาที่ Item F (ตำแหน่งที่ 6-4 = 2 ของ Items ในตาราง BitMatrix จากภาพประกอบ 4-9) หา *Delegate* ของ Item F ได้จาก 1100 AND 1111 (ระหว่างบิตแมปเวกเตอร์ของ $Q2$ และบิตแมปเวกเตอร์ของ $Q5$ ตั้งแต่บิตของ F เป็นต้นไป) ได้ผลลัพธ์เป็น 1100 ดังนั้น *Delegate* ของ Item F คือ {F,I} ซึ่งมีจำนวนสมาชิกเท่ากับ 2 แสดงว่าไม่สามารถหา F_4 จาก *Delegate* ของ Item F ได้ และเนื่องจากค่าความถี่ของ Item F มีค่าเท่ากับค่าความถี่ขั้นต่ำ จึงไม่มีโอกาสค้นพบ F_4 จากการสร้าง C_4 ได้ ดังนั้นจึงเลื่อนค่า *pointer* เพื่อพิจารณา Item ถัดไปทางซ้ายคือ Item D จะได้ *Delegate* ของ Item D คือ {D,F,I} ซึ่งมีจำนวนสมาชิกเท่ากับ 3 แสดงว่าไม่สามารถหา F_4 จาก *Delegate* ของ Item D ได้ และไม่มีโอกาสค้นพบ F_4 จากการหา C_4 ได้เนื่องจากค่าความถี่ของ

Item D มีค่าเท่ากับค่าความถี่ขั้นต่ำ จึงต้องพิจารณา Item ถัดไปทางซ้ายของ Item D ซึ่งก็คือ Item B จะได้ว่า *Delegate* ของ Item B คือ {B,D,F,I} ซึ่งมีจำนวนสมาชิกเท่ากับ 4 แสดงว่าสามารถหา F_4 จาก *Delegate* ของ Item B ได้ ดังนั้นจะได้ F_4 = {B,D,F,I} จากนั้นเลือก Item เหล่านี้ออกจากตาราง BitMatrix จะเหลือ Item ในตาราง BitMatrix จำนวน 2 Items คือ M และ N ซึ่งไม่สามารถหา F_4 ได้อีก

ในรอบที่ 3 เป็นการหา F_2 โดยพิจารณาในทำนองเดียวกันจะได้ F_2 = {M,N} ดังนั้นจากตัวอย่างที่กล่าวมาจะได้กลุ่มข้อมูลที่ต้องการตามขั้นตอนวิธี EncodedBitmapFI ดังภาพประกอบ 4-10

Query	**ค่าของแอทริบิวต์ *X***					
	B:2	**D:2**	**F:2**	**I:2**	**M:2**	**N:2**
*Q*1	0	0	0	0	0	0
*Q*2	1	1	1	1	0	0
*Q*3	0	0	0	0	1	1
*Q*4	0	0	0	0	0	0
*Q*5	1	1	1	1	1	1

ภาพประกอบ 4-9 ตาราง BitMatrix เมื่อเลือก {A,C,E,G,H,J K,O} ออก

จำนวนสมาชิก (*k*)	**Itemsets**	**Support (%)**
8	A,C,E,G,H,J,K,O	60
4	B,D,F,I	40
2	M,N	40

ภาพประกอบ 4-10 กลุ่มข้อมูลที่ปรากฏบ่อยขนาด 2^i

ขั้นตอนที่ 2.2 : จัดลำดับค่าของแอทริบิวต์

การจัดลำดับค่าของแอทริบิวต์เป็นการจัดเรียงค่าของแอทริบิวต์แต่ละค่าให้อยู่ในตำแหน่งที่เหมาะสมก่อนการเข้ารหัสในขั้นตอนที่ 3 โดยนำสมาชิกในกลุ่มข้อมูลที่หาได้แต่ละกลุ่มมาจัดเรียงให้เป็นลำดับที่ต่อเนื่องกัน โดยจัดเรียงตามจำนวนสมาชิกของแต่ละกลุ่มข้อมูลจากมากไปน้อย จากนั้นนำค่าของแอทริบิวต์ที่เหลือ (ค่าของแอทริบิวต์ที่ไม่ได้เป็นสมาชิกของกลุ่มข้อมูลใด) มาเรียงต่อจากข้อมูลที่จัดเรียงอยู่แล้ว ตัวอย่างเช่น จากกลุ่มข้อมูลที่หาได้ดังภาพประกอบ 4-10 จะได้ว่ากลุ่มข้อมูลที่มีสามาชิกมากที่สุดคือ {A,C,E,G,H,O,J,K} และกลุ่ม

ข้อมูลถัดไปคือ {B,D,F,I} และ {M,N} ตามลำดับ ดังนั้นจะได้ผลลัพธ์ของการจัดลำดับค่าของแอทริบิวต์เป็น A, C, E, G, H, J, K, O, B, D, F, I, M, N และเมื่อนำค่าของแอทริบิวต์ที่เหลือคือ L และ P มาเรียงต่อ จะได้ผลลัพธ์สุดท้ายของการจัดลำดับค่าของแอทริบิวต์เป็น A, C, E, G, H, J, K, O, B, D, F, I, M, N, L, P ตามลำดับ

ขั้นตอนที่ 3 : Encoding

Encoding เป็นขั้นตอนการเข้ารหัสค่าของแอทริบิวต์ตามลำดับที่ถูกจัดเรียงตั้งแต่ข้อมูลตัวแรกจนถึงตัวสุดท้าย โดยกำหนดให้รูปแบบการลงรหัสเริ่มจาก 0 ถึง C-1 ในรูปแบบของเลขฐานสอง (Binary encoding) ซึ่งผลลัพธ์ที่ได้จากขั้นตอนนี้จะเป็นตารางเทียบค่าของดัชนีบิตแมปแบบเข้ารหัส ที่นำไปสู่การสร้างดัชนีบิตแมปแบบเข้ารหัสที่มีประสิทธิภาพสำหรับการสอบถามแบบสมาชิกต่อไป ดังนั้นจากตัวอย่างค่าของแอทริบิวต์ที่ถูกจัดเรียงลำดับไว้ในขั้นตอนที่ 2.2 ซึ่งมีคาร์นอร์ลิตี้เท่ากับ 16 แสดงว่าต้องใช้ 4 บิตแมปเวกเตอร์ในการสร้างดัชนี ได้แก่ บิตแมปเวกเตอร์ E^3, E^2, E^1 และ E^0 โดยที่แต่ละค่าถูกเข้ารหัสในรูปแบบ $E^3E^2E^1E^0$ เมื่อ $E^i \in \{B_i^{'} = 0, B_i = 1\}$, i = 0, 1, 2, ..., $\lceil \log_2 C \rceil$-1 นำค่าของแอทริบิวต์มาเข้ารหัสตามลำดับตั้งแต่ค่าแรก (คือ A) จนถึงค่าสุดท้าย (คือ P) ได้ผลลัพธ์แสดงดังภาพประกอบ 4-11(ข) เปรียบเทียบกับตารางเทียบค่าที่ถูกสร้างโดยไม่ได้ใช้กลุ่มข้อมูลที่ปรากฏบ่อยแสดงดังภาพประกอบ 4-11(ก)

A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000
J	1001
K	1010
L	1011
M	1100
N	1101
O	1110
P	1111

(ก) ไม่ใช้กลุ่มข้อมูลที่ปรากบ่อย

A	0000
C	0001
E	0010
G	0011
H	0100
J	0101
K	0110
O	0111
B	1000
D	1001
F	1010
I	1011
M	1100
N	1101
L	1110
P	1111

(ข) ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

ภาพประกอบ 4-11 ตารางเทียบค่า

4.2 การสอบถามข้อมูลแบบค่าเท่ากัน

การสอบถามข้อมูลแบบค่าเท่ากันมีรูปแบบคือ “X=v” หมายถึง การสอบถามว่าบนแอทริบิวต์ X มีเรคอร์ดใดบ้างที่มีค่าเท่ากับ v สำหรับการสอบถามข้อมูลแบบค่าเท่ากันบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีวิธีการเช่นเดียวกับดัชนีบิตแมปแบบเข้ารหัสทั่วไป โดยการตรวจสอบกับตารางเทียบค่าก่อนว่าแต่ละค่าที่ต้องการสอบถามถูกเข้ารหัสในรูปแบบใด แล้วจึงนำไปตรวจสอบกับบิตแมปเวกเตอร์ว่าตรงกับแถวใด ก็จะได้แถวนั้นมีค่าของแอทริบิวต์ตรงกับที่ต้องการ

4.3 การสอบถามข้อมูลแบบสมาชิก

การสอบถามข้อมูลแบบสมาชิกมีรูปแบบคือ “X in $\{v_1,v_2, ...,v_n\}$” หมายถึง การสอบถามว่าบนแอทริบิวต์ X มีเรคอร์ดใดบ้างที่มีค่าเท่ากับ v_i เมื่อ i = 1,2, ...,n สำหรับการสอบถามข้อมูลแบบสมาชิกบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยประกอบด้วยการอ่านตารางเทียบค่าว่าแต่ละค่าที่ต้องการสอบถามถูกเข้ารหัสในรูปแบบใด การ

ดำเนินการตรรกะ OR ระหว่างรูปแบบการเข้ารหัสของแต่ละค่า การลดรูปฟังก์ชันการเข้าถึงข้อมูล การอ่านบิตแมปเวกเตอร์และดำเนินการตรรกะตามฟังก์ชันการเข้าถึงข้อมูลที่ผ่านการลดรูปแล้ว

ตัวอย่างเช่น เมื่อต้องการสอบถามข้อมูลตามเงื่อนไขการสอบถาม Q1 ในภาพประกอบ 4-2 โดยใช้ตารางเทียบค่าจากภาพประกอบ 4-11(ข) สามารถดำเนินการได้ดังนี้

1) อ่านตารางเทียบค่าเพื่อตรวจสอบรูปแบบการเข้ารหัสของแต่ละค่าที่ต้องการสอบถามจะได้

A ถูกเข้ารหัสเป็น $B_3'B_2'B_1'B_0'$

C ถูกเข้ารหัสเป็น $B_3'B_2'B_1'B_0$

E ถูกเข้ารหัสเป็น $B_3'B_2'B_1B_0'$

G ถูกเข้ารหัสเป็น $B_3'B_2'B_1B_0$

H ถูกเข้ารหัสเป็น $B_3'B_2B_1'B_0'$

J ถูกเข้ารหัสเป็น $B_3'B_2B_1'B_0$

K ถูกเข้ารหัสเป็น $B_3'B_2B_1'B_0'$

O ถูกเข้ารหัสเป็น $B_3'B_2B_1B_0$

P ถูกเข้ารหัสเป็น $B_3B_2B_1B_0$

2) ดำเนินการตรรกะ OR (ในที่นี้ใช้สัญลักษณ์ + แทน OR) ระหว่างรูปแบบการเข้ารหัสของแต่ละค่า จะได้ฟังก์ชันการเข้าถึงข้อมูลเป็นดังนี้

$$B_3'B_2'B_1'B_0' + B_3'B_2'B_1'B_0 + B_3'B_2'B_1B_0' + B_3'B_2'B_1B_0 + B_3'B_2B_1'B_0' + B_3'B_2B_1'B_0 + B_3'B_2B_1'B_0' + B_3'B_2B_1B_0 + B_3B_2B_1B_0$$

3) ดำเนินการลดรูปฟังก์ชันการเข้าถึงข้อมูล จะได้ผลลัพธ์ดังนี้

$$B_3' + B_2B_1B_0$$

จากผลลัพธ์แสดงว่า เราต้องตรวจสอบบิตแมปเวกเตอร์ E_3 = 0, E_2 = 1, E_1 = 1 และ E_0 = 1 และมีการดำเนินการตรรกะ OR เพื่อหาคำตอบของการสอบถามนี้

ต่อไปเป็นการเปรียบเทียบผลลัพธ์ของการลดรูปฟังก์ชันการเข้าถึงข้อมูลที่ได้จากตารางเทียบค่าดังภาพประกอบ 4-11(ก) ซึ่งเป็นตารางเทียบค่าที่ถูกสร้างขึ้นโดยไม่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย เปรียบเทียบกับตารางเทียบค่าดังภาพประกอบ 4-11(ข) ซึ่งเป็นตาราง

เทียบค่าที่ถูกสร้างขึ้นโดยใช้กลุ่มข้อมูลที่ปรากฏบ่อย โดยใช้การสอบถามจากภาพประกอบ 4-2 จะได้ผลลัพธ์ของฟังก์ชันการเข้าถึงข้อมูลที่ผ่านการลดรูปของแต่ละการสอบถาม แสดงดังภาพประกอบ 4-12

Query	ตารางเทียบค่า 4-11(ก) (ไม่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย)	ตารางเทียบค่า 4-11(ข) (ใช้กลุ่มข้อมูลที่ปรากฏบ่อย)
Q1	$B_3'B_0' + B_1B_0' + B_2B_1 + B_3B_2'B_1'B_0$	$B_3' + B_2B_1B_0$
Q2	$B_3'B_2'B_0 + B_3'B_1'B_0 + B_3B_2'B_1'B_0'$	B_3B_2'
Q3	$B_3'B_0' + B_1B_0' + B_3B_2B_1' + B_3'B_2B_1$	$B_3' + B_3B_2B_1'$
Q4	$B_3'B_0' + B_1B_0' + B_3'B_2'B_1 + B_3B_2'B_1'B_0$	B_3'
Q5	$B_3'B_2'B_0 + B_3'B_1'B_0 + B_3B_2B_1'$	$B_3B_2' + B_3B_2B_1'$

ภาพประกอบ 4-12 เปรียบเทียบฟังก์ชันการเข้าถึงข้อมูลที่ผ่านการลดรูป
เมื่อใช้ตารางเทียบค่า 4-11(ก) และ 4-11(ข)

จากภาพประกอบ 4-12 จะเห็นว่าตารางเทียบค่า 4-11(ข) สามารถลดรูปฟังก์ชันหรือลดจำนวนบิตแมปเวกเตอร์และจำนวนครั้งในการดำเนินการตรรกะได้มากกว่าตารางเทียบค่า 4-11(ก) กล่าวคือการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกดีกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไปที่ไม่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

4.4 ข้อเด่นของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย เป็นเทคนิคการทำดัชนีบิตแมปที่มีการใช้พื้นที่ในการจัดเก็บดัชนีน้อยที่สุด (มีประสิทธิภาพมากที่สุดในการใช้พื้นที่) เช่นเดียวกับดัชนีบิตแมปแบบเข้ารหัสทั่วไป เนื่องจากดัชนีบิตแมปแบบเข้ารหัสใช้จำนวนบิตแมปเวกเตอร์ในการสร้างดัชนีน้อยกว่าดัชนีบิตแมปแบบพื้นฐาน แบบช่วง แบบกระจาย และแบบคู่กัน และมีประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิก โดยเฉพาะบนค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อย ๆ เนื่องจากได้มีการเข้ารหัสแต่ละค่าของแอทริบิวต์ให้อยู่ในรูปแบบที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบให้เหลือน้อยที่สุดเมื่อมีการสอบถามข้อมูลแบบสมาชิกด้วยกันอีก

4.5 ข้อด้อยของดัชนีบิตแมปแบบเข้าที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย ยังด้อยประสิทธิภาพในเรื่องการสอบถามข้อมูลแบบค่าเท่ากันเช่นเดียวกับการทำดัชนีบิตแมปแบบเข้ารหัสทั่วไป เนื่องจากต้องตรวจสอบทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) และมีการเปรียบเทียบกับตารางเทียบค่า นอกจากนี้สำหรับเงื่อนไขการสอบถามที่ไม่เคยถูกสอบถามมาก่อน อาจจะไม่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบให้เหลือน้อยที่สุดได้ เนื่องจากรูปแบบการลงรหัสแต่ละค่าของ แอริทริบิวต์มุ่งให้ความสำคัญที่ค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อยในอดีต

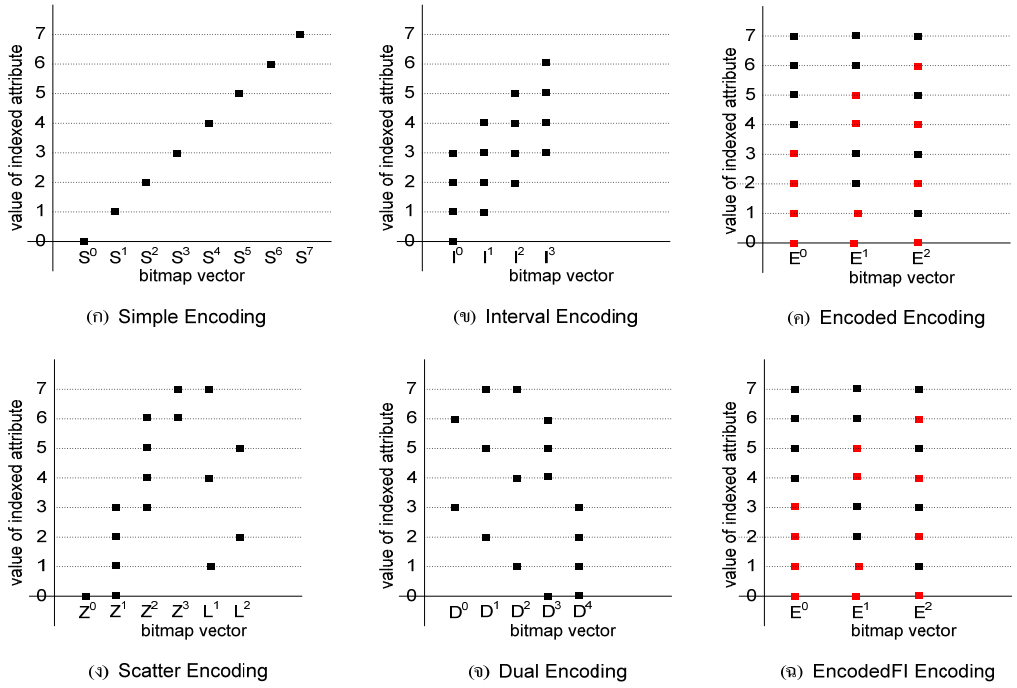
บทที่ 5

การวิเคราะห์และผลการทดลอง

สำหรับบทนี้จะกล่าวถึงการวิเคราะห์เปรียบเทียบค่าใช้จ่ายเกี่ยวกับพื้นที่ที่ใช้ในการจัดเก็บดัชนี ค่าใช้จ่ายเกี่ยวกับเวลาที่ใช้ในการสอบถามแบบสมาชิก และประสิทธิภาพในแง่ Space-Time Trade-off (การแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่และเวลา) ของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI Bitmap Index) ที่คิดค้นขึ้นและดัชนีบิตแมปแบบไม่มีการบีบอัดดัชนีที่เคยมีมา 5 ชนิด ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน (Simple Bitmap Index) ดัชนีบิตแมปแบบช่วง (Interval Bitmap Index) ดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index) ดัชนีบิตแมปแบบคู่กัน (Dual Bitmap Index) และดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded Bitmap Index)

5.1 ค่าใช้จ่ายเกี่ยวกับพื้นที่ที่ใช้ในการจัดเก็บดัชนี

เนื่องจากดัชนีบิตแมปแต่ละชนิดมีรูปแบบการลงรหัสที่ไม่เหมือนกัน จึงทำให้ประสิทธิภาพด้านพื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมปแตกต่างกันด้วย ตัวอย่างรูปแบบการลงรหัสของดัชนีบิตแมปทั้ง 6 ชนิด แสดงดังภาพประกอบ 5-1



(ก) Simple Encoding (ข) Interval Encoding (ค) Encoded Encoding

(ง) Scatter Encoding (จ) Dual Encoding (ฉ) EncodedFI Encoding

ภาพประกอบ 5-1 แผนภาพการลงรหัสดัชนีบิตแมปทั้ง 6 ชนิด เมื่อคาร์ดินอร์ลิตี้เท่ากับ 8

จากภาพประกอบ 5-1 เมื่อกำหนดคาร์ดินอร์ลิตี้เท่ากับ 8 ประกอบด้วยสมาชิก คือ {0,1,2,3,4,5,6,7} ดัชนีบิตแมปแบบพื้นฐานใช้ 8 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(ก) ดัชนีบิตแมปแบบช่วงใช้ 4 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(ข) ดัชนีบิตแมปแบบเข้ารหัสทั่วไปใช้ 3 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(ค) ดัชนีบิตแมปแบบกระจายใช้ 6 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(ง) ดัชนีบิตแมปแบบคู่กันใช้ 5 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(จ) และดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้ 3 บิตแมปเวกเตอร์ดังภาพประกอบ 5-1(ฉ) โดยจุดสีเข้มแสดงให้เห็นว่า แต่ละบิตแมปเวกเตอร์ใช้แทนค่าใดบ้าง เช่น จากรูปแบบการลงรหัสของดัชนีบิตแมปแบบพื้นฐานดังภาพประกอบ 5-1(ก) บิตแมปเวกเตอร์ S^0 ใช้แทนค่า 0 หรือจากรูปแบบการลงรหัสของดัชนีบิตแมปแบบช่วงดังภาพประกอบ 5-1(ข) บิตแมปเวกเตอร์ I^0 แทนค่าที่อยู่ในช่วง [0,3] เป็นต้น

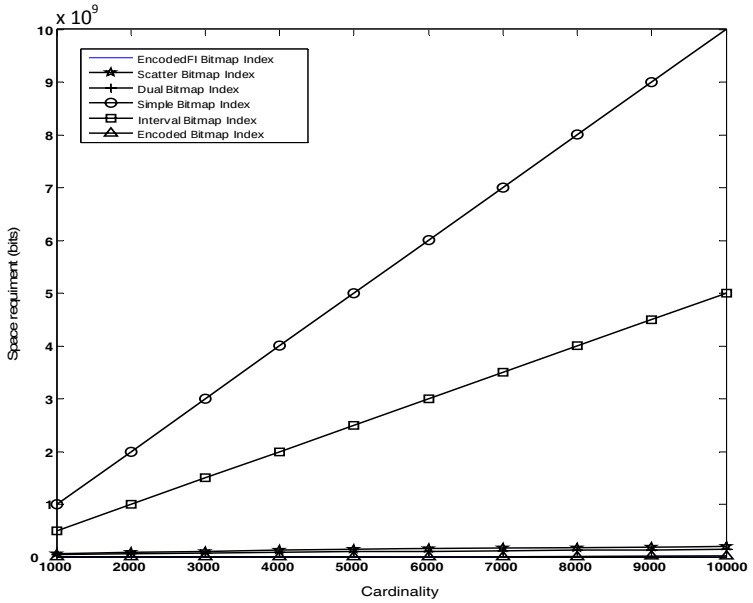
พิจารณาการเปรียบเทียบพื้นที่ที่ใช้ในการจัดเก็บดัชนีของดัชนีบิตแมปทั้ง 6 ชนิด ในรูปแบบทั่วไป ดังตาราง 5-1

ตาราง 5-1 พื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมปทั้ง 6 ชนิด เมื่อ C คือ คาร์ดินอร์ลิตี้ และ N คือ จำนวนเรคอร์ด

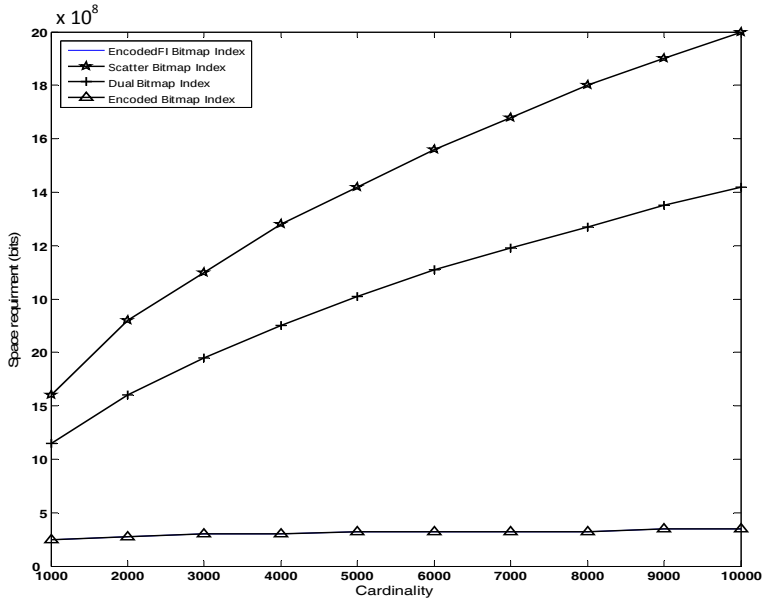
ดัชนีบิตแมป	พื้นที่ที่ใช้ในการจัดเก็บดัชนี (บิต)
ดัชนีบิตแมปแบบพื้นฐาน	CN
ดัชนีบิตแมปแบบช่วง	$\lceil C/2 \rceil N$
ดัชนีบิตแมปแบบกระจาย	$\lceil 2\sqrt{C} \rceil N$
ดัชนีบิตแมปแบบคู่กัน	$\lceil \sqrt{2C + 0.25} + 0.5 \rceil N$
ดัชนีบิตแมปแบบเข้ารหัสทั่วไป	$\lceil \log_2 C \rceil N$
ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย	$\lceil \log_2 C \rceil N$

จากตาราง 5-1 แสดงให้เห็นว่า การทำดัชนีบิตแมปบนแอทริบิวต์ที่มี N เรคอร์ด พื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมปแต่ละชนิดจะแปรผันตรงกับคาร์ดินอร์ลิตี้ (C) ของแอทริบิวต์ที่นำมาทำดัชนี

ต่อไปจะเปรียบเทียบการใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนีบิตแมปแต่ละชนิด โดยพิจารณาจากกราฟดังภาพประกอบ 5-2



ภาพประกอบ 5-2 กราฟเปรียบเทียบพื้นที่ที่ใช้ในการสร้างดัชนีทั้ง 6 ชนิด
เมื่อแอทริบิวต์ที่นำมาสร้างดัชนีมี 1,000,000 เรคอร์ด (N = 1,000,000)



ภาพประกอบ 5-3 กราฟเปรียบเทียบพื้นที่ที่ใช้ในการสร้างดัชนี 4 ชนิด
เมื่อแอทริบิวต์ที่นำมาสร้างดัชนีมี 1,000,000 เรคอร์ด (N = 1,000,000)

พิจารณาภาพประกอบ 5-2 แสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการสร้างดัชนีกับคาร์ดินอร์ลิตี้ของแอทริบิวต์ที่เลือกมาสร้างดัชนีบิตแมปทั้ง 6 ชนิด ซึ่งคาร์ดินอร์ลิตี้มีค่า

ตั้งแต่ 1,000 ถึง 10,000 จะเห็นว่าเราสามารถแบ่งกลุ่มดัชนีตามการใช้พื้นที่จัดเก็บดัชนีได้เป็น 2 กลุ่ม ได้แก่

- กลุ่มที่ใช้พื้นที่มาก ได้แก่ ดัชนีบิตแมปแบบพื้นฐานและดัชนีบิตแมปแบบช่วง โดยดัชนีบิตแมปแบบช่วงใช้พื้นที่น้อยกว่าดัชนีบิตแมปแบบพื้นฐาน
- กลุ่มที่ใช้พื้นที่น้อย ได้แก่ ดัชนีบิตแมปแบบเข้ารหัสทั่วไป ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน

พิจารณาตามภาพประกอบ 5-3 จะเห็นว่าดัชนีบิตแมปที่ใช้พื้นที่น้อยที่สุดคือ ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยซึ่งมีการใช้พื้นที่เท่ากัน รองลงมาคือ ดัชนีบิตแมปแบบคู่กันและดัชนีบิตแมปแบบกระจายตามลำดับ

โดยสรุป ดัชนีบิตแมปที่มีประสิทธิภาพสูงที่สุดด้านการใช้พื้นที่ในการจัดเก็บดัชนีคือ ดัชนีบิตแมปแบบเข้ารหัสทั่วไป และดัชนีบิตแมปแบบเข้ารหัสที่กลุ่มข้อมูลที่ปรากฏบ่อย ซึ่งดัชนีทั้ง 2 ชนิดมีการใช้พื้นที่เท่ากัน

5.2 ค่าใช้จ่ายเกี่ยวกับเวลาที่ใช้ในการสอบถามแบบสมาชิก

เวลาที่ใช้ในการสอบถามข้อมูลประกอบด้วย เวลาที่ใช้ในการอ่านบิตแมปเวกเตอร์ ถ้าต้องอ่านหลายบิตแมปเวกเตอร์ก็จะทำให้ต้องใช้เวลามาก และเวลาที่ใช้ในการดำเนินการตรรกะระหว่างบิตแมปเวกเตอร์ ถ้าต้องดำเนินการตรรกะหลายครั้งย่อมต้องใช้เวลามากเช่นกัน พิจารณาจำนวนบิตแมปเวกเตอร์ที่ต้องอ่านและจำนวนครั้งในการดำเนินการตรรกะระหว่างบิตแมปเวกเตอร์เมื่อมีการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด แสดงดังตาราง 5-2

ตาราง 5-2 จำนวนบิตแมปเวกเตอร์ที่ต้องอ่านและจำนวนครั้งในการดำเนินการตรรกะระหว่างบิตแมปเวกเตอร์ เมื่อมีการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด เมื่อ k คือ จำนวนสมาชิกที่ต้องการสอบถาม

ดัชนีบิตแมป	จำนวนบิตแมปเวกเตอร์ที่อ่าน	จำนวนครั้งในการดำเนินการตรรกะระหว่างบิตแมปเวกเตอร์
ดัชนีบิตแมปแบบพื้นฐาน	k	k-1 (OR)
ดัชนีบิตแมปแบบช่วง	2k	3k-1 (k AND, k-1OR, k NOT)
ดัชนีบิตแมปแบบกระจาย	2k	2k-1(k AND, k-1OR)
ดัชนีบิตแมปแบบคู่กัน	2k	2k-1(k AND, k-1OR)
ดัชนีบิตแมปแบบเข้ารหัสทั่วไป	$\lceil \log_2 C \rceil k$	use mapping table, (k-1 OR)
ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย	1	0

จากตาราง 5-2 จะเห็นว่า ในการสอบถามแบบสมาชิก k ค่า ดัชนีบิตแมปแบบพื้นฐาน มีจำนวนบิตแมปเวกเตอร์ที่ต้องอ่าน k บิตแมปเวกเตอร์ และดำเนินการตรรกะ OR จำนวน k-1 ครั้ง ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน มีจำนวนบิตแมปเวกเตอร์ที่ต้องอ่าน 2k บิตแมปเวกเตอร์ และจำนวนครั้งในการดำเนินการตรรกะ 3k-1 (k AND, k-1OR, k NOT) ครั้ง สำหรับดัชนีบิตแมปแบบช่วง และ 2k-1(k AND, k-1 OR) ครั้ง สำหรับดัชนีบิตแมปแบบกระจายและดัชนีบิตแมปแบบคู่กัน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปคู่กันจึงใช้เวลาในการสอบถามแบบสมาชิกใกล้เคียงกันแต่ใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน ส่วนดัชนีบิตแมปแบบเข้ารหัสทั่วไป มีจำนวนบิตแมปเวกเตอร์ที่ต้องอ่าน $\lceil \log_2 C \rceil k$ บิตแมปเวกเตอร์ และดำเนินการตรรกะ OR จำนวน k-1 ครั้ง นอกจากนี้ยังมีการอ่านตารางเทียบค่า ดัชนีบิตแมปแบบเข้ารหัสทั่วไปจึงใช้เวลาในการสอบถามแบบสมาชิกมากที่สุด สำหรับดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย ในกรณีที่ดีที่สุดตรวจสอบเพียง 1 บิตแมปเวกเตอร์ และไม่มีการดำเนินการตรรกะ อย่างไรก็ตาม ในบางกรณีดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยอาจจะใช้เวลาในการสอบถามแบบสมาชิกมากกว่าดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน ซึ่งขึ้นอยู่กับชุดข้อมูลสอบถามในอดีต แต่จะใช้เวลาในการสอบถามแบบสมาชิกไม่มากกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป

โดยสรุปดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามแบบสมาชิกน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป และอาจมีโอกาสใช้เวลาน้อยกว่า

ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน นอกจากนี้ยังมีโอกาสใช้เวลาน้อยกว่าดัชนีบิตแมปแบบพื้นฐานด้วย โดยจะแสดงให้เห็นจากผลการทดลอง

โดยทำการทดลองบนเครื่องคอมพิวเตอร์รุ่น Intel(R) Pentium(R) 4 ที่มีหน่วยประมวลผลกลางขนาด 3.06 GHz หน่วยความจำ 1 GB ระบบปฏิบัติการ Linux Red Hat 9.0 และใช้ภาษาซีในการเขียนโปรแกรม

ข้อมูลที่ใช้ในการทดลองเป็นชุดข้อมูลมาตรฐานจาก TPC-H Benchmark (Transaction Processing Performance Council, 2006) โดยแอทริบิวต์ที่เลือกมาสร้างดัชนีมีคาร์ดินอร์ลิตี้เท่ากับ 150 และ 1,000 มีจำนวนเรคอร์ดเท่ากับ 1,000,000 เรคอร์ด สำหรับชุดการสอบถามแบบสมาชิก ในแต่ละคาร์ดินอร์ลิตี้จะทำการสุ่มการสอบถามแบบสมาชิกจำนวน 2 ชุดการสอบถาม แต่ละชุดการสอบถามมีจำนวน 100 การสอบถาม (ตัวอย่างชุดการสอบถามดูภาคผนวก ก) โดยหาค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อยจากชุดการสอบถามดังกล่าวก่อนสร้างดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย ซึ่งในการหากลุ่มข้อมูลที่ปรากฏบ่อยจะกำหนดค่าความถี่ขั้นต่ำ 4 ค่า ได้แก่ 10% 20% 30% และ 40%

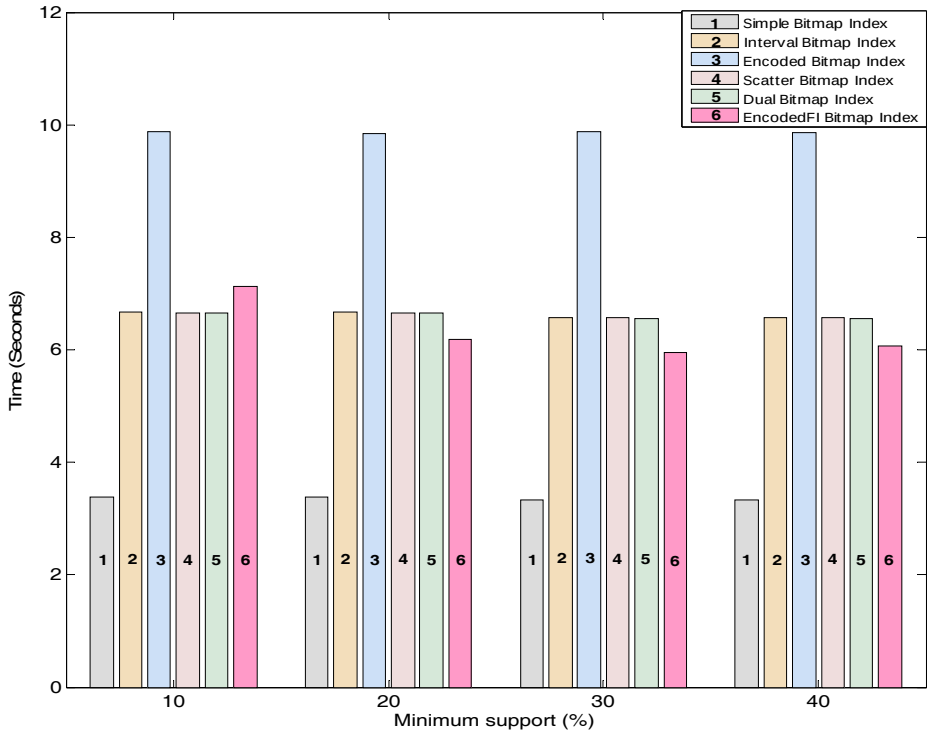
ในการทดลองกำหนดให้เวลาที่ใช้ในการสอบถามแบบสมาชิกบนดัชนีบิตแมปแต่ละชนิดประกอบด้วย การคำนวณเพื่อหาบิตแมปเวกเตอร์ที่ต้องตรวจสอบ การอ่านบิตแมปเวกเตอร์ที่ต้องตรวจสอบ การดำเนินตรรกะระหว่างบิตแมปเวกเตอร์ และการบันทึกคำตอบลงในไฟล์ข้อมูล สำหรับดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยจะรวมเวลาที่ใช้ในการอ่านตารางเทียบค่า และเวลาที่ใช้ในการคำนวณเพื่อลดรูปฟังก์ชันในการเข้าถึงข้อมูลด้วย

การบันทึกผลการทดลอง ดำเนินการโดยประมวลผลโปรแกรมเพื่อสอบถามแบบสมาชิกจากชุดการสอบถาม แล้วบันทึกเวลาที่ใช้แต่ละการสอบถาม จนครบทุกการสอบถาม จากนั้นหาค่าเฉลี่ยของเวลาที่ใช้ในแต่ละการสอบถาม ซึ่งดำเนินการซ้ำทั้งหมด 3 รอบ ดังผลการทดลองต่อไปนี้

ตาราง 5-3 เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	3.379906	6.665000	9.874623	6.660943	6.657925	7.118868
20	3.380000	6.670755	9.837642	6.656415	6.655189	6.180849
30	3.331038	6.567453	9.869057	6.567736	6.561038	5.952547
40	3.331792	6.566887	9.855849	6.562547	6.558868	6.064057

จากตาราง 5-3 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-4

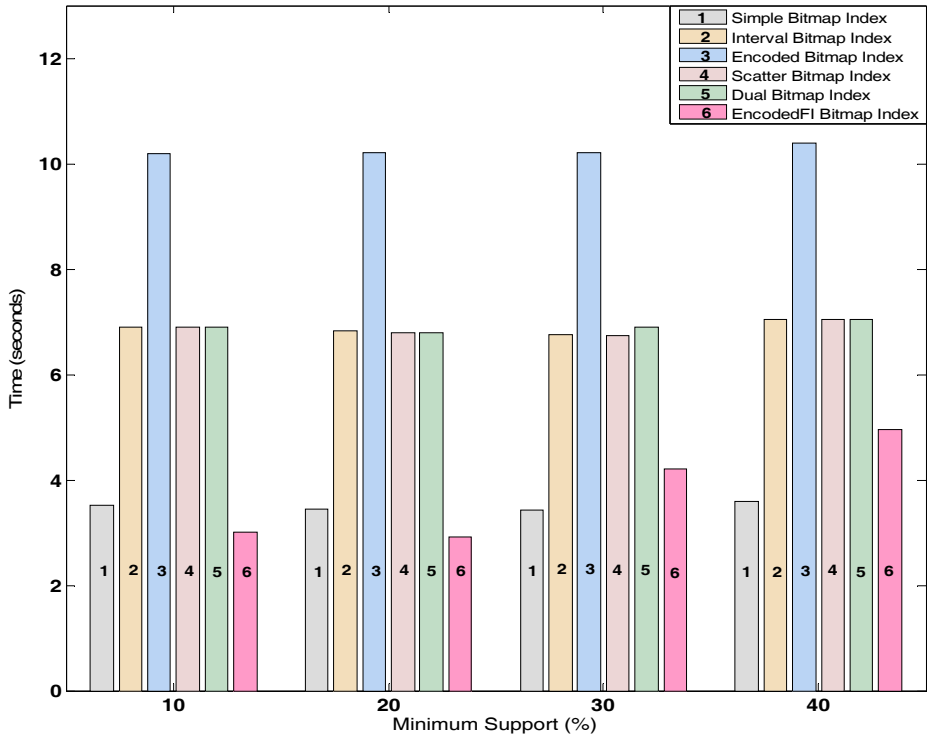


ภาพประกอบ 5-4 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ตาราง 5-4 เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	3.5093	6.9039	10.1979	6.8967	6.901	3.0122
20	3.4504	6.8245	10.2155	6.7982	6.7962	2.9253
30	3.4248	6.7529	10.214	6.7415	6.9048	4.2031
40	3.5859	7.0457	10.3953	7.0451	7.0413	4.9615

จากตาราง 5-4 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-5

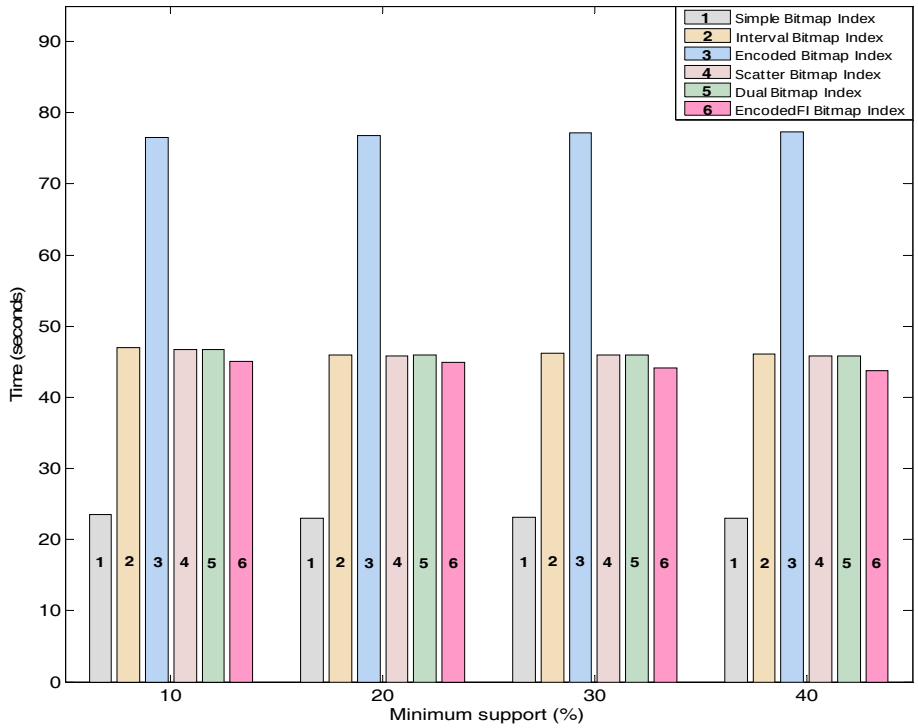


ภาพประกอบ 5-5 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ตาราง 5-5 เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	23.497212	46.983558	76.527404	46.742308	46.714423	45.089519
20	23.023077	45.882115	76.809712	45.751635	45.933077	44.968654
30	23.122981	46.170673	77.178558	45.935481	45.928750	44.071923
40	23.022807	46.106308	77.275769	45.834077	45.825330	43.772212

จากตาราง 5-5 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-6

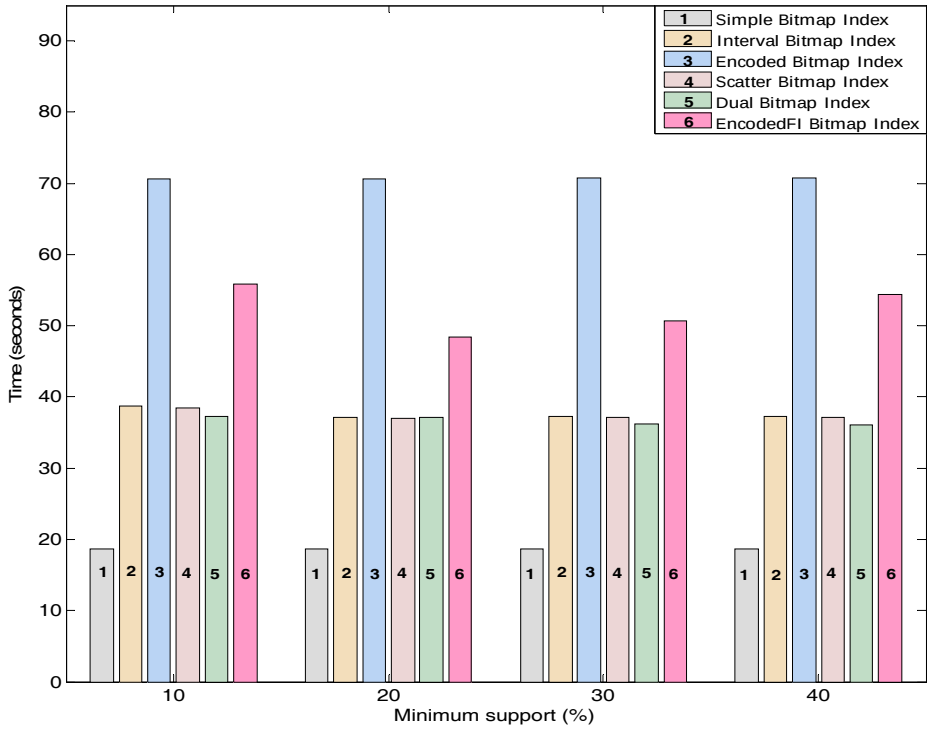


ภาพประกอบ 5-6 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ตาราง 5-6 เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	18.734020	38.774412	70.602843	38.458333	37.222353	55.844804
20	18.611373	37.108725	70.682941	37.013235	37.081176	48.478529
30	18.687059	37.242745	70.811569	37.147549	36.257451	50.628137
40	18.630521	37.207543	70.801669	37.074872	36.042431	54.388137

จากตาราง 5-6 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-7



ภาพประกอบ 5-7 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

สำหรับผลการทดลองบนชุดการสอบถามที่ 1 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 150 แสดงดังตาราง 5-3 และภาพประกอบ 5-4 โดยกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยของการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 4 ค่า ได้แก่ 10% 20% 30% และ 40% จะเห็นว่า เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 10% ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาในการสอบถามเฉลี่ยน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป แต่ยังใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน แต่เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 20% 30% และ 40% ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาน้อยกว่าทั้งดัชนีบิตแมปแบบเข้ารหัสทั่วไป ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน แต่ยังใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน

สำหรับผลการทดลองบนชุดการสอบถามที่ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 150 แสดงดังตาราง 5-4 และภาพประกอบ 5-5 โดยกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยของการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 4 ค่า ได้แก่ 10% 20% 30% และ 40% จะเห็นว่า เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 10% และ 20% ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาในการสอบถามเฉลี่ยน้อยที่สุด นั่นคือใช้เวลาน้อยกว่าทั้งดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัสทั่วไป แต่เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 30% และ 40% ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาในการสอบถามเฉลี่ยน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน แต่ใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน

สำหรับผลการทดลองบนชุดการสอบถามที่ 1 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 1,000 แสดงดังตาราง 5-5 และภาพประกอบ 5-6 โดยกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยของการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 4 ค่า ได้แก่ 10% 20% 30% และ 40% จะเห็นว่า ในแต่ละค่าความถี่ขั้นต่ำที่กำหนดให้ผลลัพธ์ที่ใกล้เคียงกัน นั่นคือดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาน้อยกว่าดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัสทั่วไป แต่ใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน

สำหรับผลการทดลองบนชุดการสอบถามที่ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 1,000 แสดงดังตาราง 5-6 และภาพประกอบ 5-7 โดยกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยของการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย 4 ค่า ได้แก่ 10% 20% 30% และ 40% จะเห็นว่า ในแต่ละค่าความถี่ขั้นต่ำที่กำหนดให้ผลลัพธ์ที่ใกล้เคียงกัน นั่นคือดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากกฏบ่อยใช้เวลาน้อยกว่าดัชนี

บิตแมปแบบเข้ารหัสทั่วไป แต่ใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่

จากผลการทดลองสามารถสรุปลำดับของประสิทธิภาพในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิดได้ดังตาราง 5-7 โดย 1 หมายถึง มีประสิทธิภาพดีที่สุด และ 6 หมายถึง แย่ที่สุด

ตาราง 5-7 สรุปลำดับประสิทธิภาพของการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด ที่ได้จากการทดลอง เมื่อ 1 หมายถึง ดีที่สุด และ 6 หมายถึง แย่ที่สุด

คาร์ดินอร์ลิตี้	ชุดการสอบถามที่	ค่าความถี่ขั้นต่ำ (%)	แบบพื้นฐาน	แบบช่วง	แบบเข้ารหัส	แบบกระจาย	แบบคู่กัน	แบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย
150	1	10	1	4	6	3	2	5
		20	1	5	6	4	3	2
		30	1	4	6	5	3	2
		40	1	5	6	4	3	2
	2	10	2	5	6	3	4	1
		20	2	5	6	4	3	1
		30	1	5	6	3	4	2
		40	1	5	6	4	3	2
1000	1	10	1	5	6	4	3	2
		20	1	4	6	3	5	2
		30	1	5	6	4	3	2
		40	1	5	6	4	3	2
	2	10	1	5	6	4	3	5
		20	1	5	6	3	4	5
		30	1	4	6	3	5	5
		40	1	5	6	4	3	5

จากตาราง 5-7 แสดงให้เห็นว่าดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีโอกาสใช้เวลาในการสอบถามแบบสมาชิกน้อยที่สุด (มีประสิทธิภาพในการสอบถามแบบสมาชิกดีที่สุด) ดังผลการทดลองในชุดการสอบถามที่ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 150 เมื่อกำหนดค่าความถี่ขั้นต่ำเท่ากับ 10% และ 20% และจากผลการทดลองในชุดการสอบถามที่ 1 และ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 150 และชุดการสอบถาม

ที่ 1 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 1,000 แสดงให้เห็นว่าโดยส่วนใหญ่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน ส่วนผลการทดลองจากชุดการสอบถามที่ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 1,000 แสดงให้เห็นว่า ถึงแม้ว่าในบางชุดการการสอบถามดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลามากกว่าดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน แต่จะใช้เวลาไม่มากกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป ทั้งนี้ประสิทธิภาพของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยขึ้นอยู่กับชุดการสอบถามในอดีต โดยชุดการสอบถามที่ทำให้ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพ คือ ชุดการสอบถามที่มีการสอบถามค่าของแอทริบิวต์ในเงื่อนไขเดียวกันหลายค่าและมีการสอบถามด้วยกันหลาย ๆ ครั้ง และสำหรับชุดการสอบถามที่ค่าของแอทริบิวต์ที่เลือกมาสร้างดัชนีมีรูปแบบการสอบถามที่ไม่ค่อยมีความสัมพันธ์กัน ไม่สามารถหาค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันได้ หรือค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันมีจำนวนสมาชิกน้อยเมื่อเทียบกับคาร์ดินอร์ลิตี้ของแอทริบิวต์นั้น ประสิทธิภาพของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยก็จะน้อย ดังเช่นผลการทดลองในชุดการสอบถามที่ 2 ของแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 1,000

จากผลการทดลองที่กล่าวมาเป็นการแสดงเวลาเฉลี่ยที่ใช้ในการสอบถามแบบสมาชิกของแต่ละการสอบถามจากการสอบถามทั้งหมด ซึ่งในแต่ละชุดการสอบถามอาจมีทั้งการสอบถามที่มีค่าของแอทริบิวต์ที่ไม่ได้เป็นกลุ่มข้อมูลที่ปรากฏบ่อย และการสอบถามที่มีค่าของแอทริบิวต์ที่มีกลุ่มข้อมูลที่ปรากฏบ่อย เมื่อพิจารณาแต่ละการสอบถามในแต่ละชุดการสอบถามพบว่าสำหรับการสอบถามที่มีค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อยนั้น ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยจะใช้เวลาในการสอบถามน้อยที่สุด โดยในแต่ละการทดลองมีจำนวนการสอบถามที่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามน้อยที่สุดแสดงดังตาราง 5-8 ข้อมูลในตารางแสดงให้เห็นว่าจากจำนวนการสอบถามทั้งหมด 100 การสอบถาม มีจำนวนการสอบถามที่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามแบบสมาชิกน้อยที่สุดอยู่กี่การสอบถาม เช่น จากการทดลองบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้เท่ากับ 150 เมื่อใช้ชุดการสอบถามที่ 1 และกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยเท่ากับ 10% มีจำนวนการสอบถามที่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาน้อยที่สุดอยู่ 19% หมายความว่าจากการสอบถามทั้งหมด 100 การสอบถาม มี 19 การสอบถามที่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาน้อยที่สุด โดยเวลาที่ใช้ในแต่ละการสอบถามเฉลี่ยแสดงดังตาราง 5-9, 5-10, 5-11 และ 5-12

ตาราง 5-8 จำนวนการสอบถาม (เปอร์เซ็นต์) ในแต่ละการทดลองที่ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามน้อยที่สุด

ค่าความถี่ขั้นต่ำ (%)	คาร์นอร์ลิตี้ 150		คาร์ดินอร์ลิตี้ 1,000	
	ชุดการสอบถามที่ 1	ชุดการสอบถามที่ 2	ชุดการสอบถามที่ 1	ชุดการสอบถามที่ 2
10	19%	63%	9%	4%
20	37%	63%	9%	4%
30	19%	61%	25%	0
40	19%	42%	25%	0

ตาราง 5-9, 5-10, 5-11 และ 5-12 และกราฟดังภาพประกอบ 5-8, 5-9, 5-10 และ 5-11 เป็นผลการทดลองบนการสอบถามที่มีค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันบ่อย แสดงให้เห็นว่า ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพด้านเวลาที่ใช้ในการสอบถามแบบค่าสมาชิกดีที่สุด

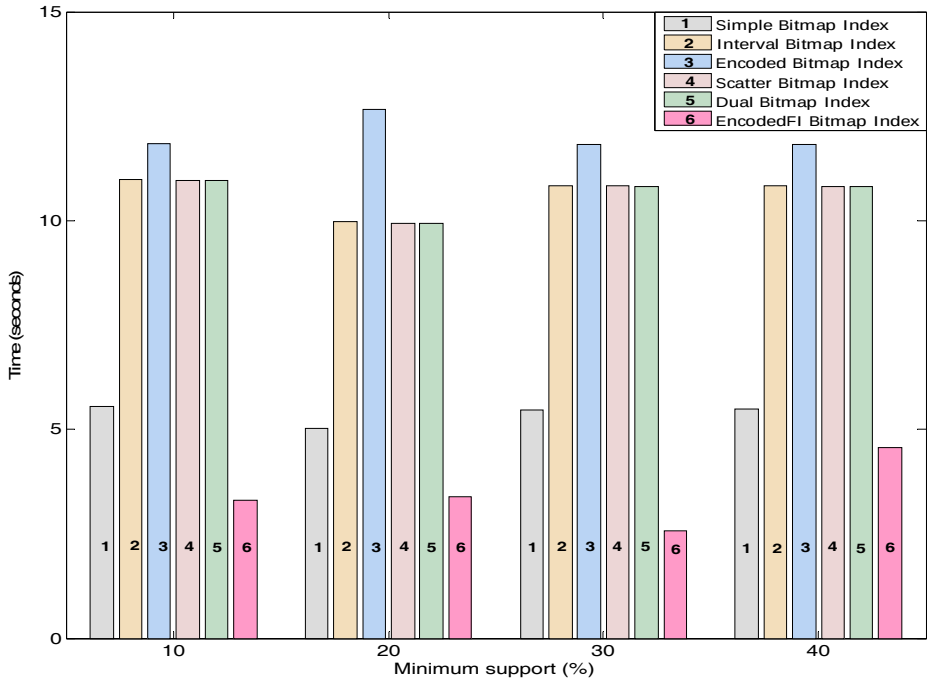
จากผลการทดลองทั้งหมดที่กล่าวมา จะเห็นว่า ประสิทธิภาพด้านเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยขึ้นอยู่กับชุดการสอบถามในอดีตและกลุ่มข้อมูลที่ปรากฏบ่อยที่หาได้ ซึ่งกลุ่มข้อมูลที่ปรากฏบ่อยที่หาได้จะขึ้นอยู่กับค่าความถี่ขั้นต่ำที่กำหนดด้วย ซึ่งพิจารณาได้จากผลการทดลองทั้ง 4 ชุดการสอบถาม เมื่อกำหนดค่าความถี่ขั้นต่ำในการหากลุ่มข้อมูลที่ปรากฏบ่อยต่างกัน เวลาที่ใช้ของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยต่างกันด้วย โดยการกำหนดค่าความถี่ขั้นต่ำให้มีค่าสูงจะทำให้พบกลุ่มข้อมูลที่ปรากฏบ่อยน้อย หรือบางครั้งอาจไม่พบกลุ่มข้อมูลที่ปรากฏบ่อยเลย ส่วนการกำหนดค่าความถี่ขั้นต่ำให้มีค่าน้อย จะทำให้พบกลุ่มข้อมูลที่ปรากฏบ่อยได้มากขึ้น แต่การกำหนดค่าความถี่ขั้นต่ำที่น้อยเกินไปอาจทำให้พบกลุ่มข้อมูลที่นำมาสร้างดัชนีบิตแมปแบบเข้ารหัสมีประสิทธิภาพเพิ่มขึ้นไม่มากนัก ดังนั้นการกำหนดค่าความถี่ขั้นต่ำที่เหมาะสมจะส่งผลให้สามารถเพิ่มประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัสได้มากยิ่งขึ้น

สรุปโดยภาพรวมจากการทดลองแสดงให้เห็นว่า ดัชนีบิตแมปแบบเข้ารหัสทั่วไปใช้เวลาในการสอบถามแบบสมาชิกมากที่สุด ส่วนดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กันใช้เวลาใกล้เคียงกัน สำหรับดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป และมีโอกาสใช้เวลาน้อยกว่าดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่ ทั้งนี้ขึ้นอยู่กับชุดการสอบถามที่เคยมีมาในอดีต

ตาราง 5-9 เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	5.550000	10.990000	11.840000	10.960000	10.970000	3.300000
20	5.030000	9.970000	12.666667	9.940000	9.933333	3.386667
30	5.470000	10.830000	11.830000	10.830000	10.820000	2.580000
40	5.480000	10.830000	11.820000	10.820000	10.820000	4.560000

จากตาราง 5-9 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-8

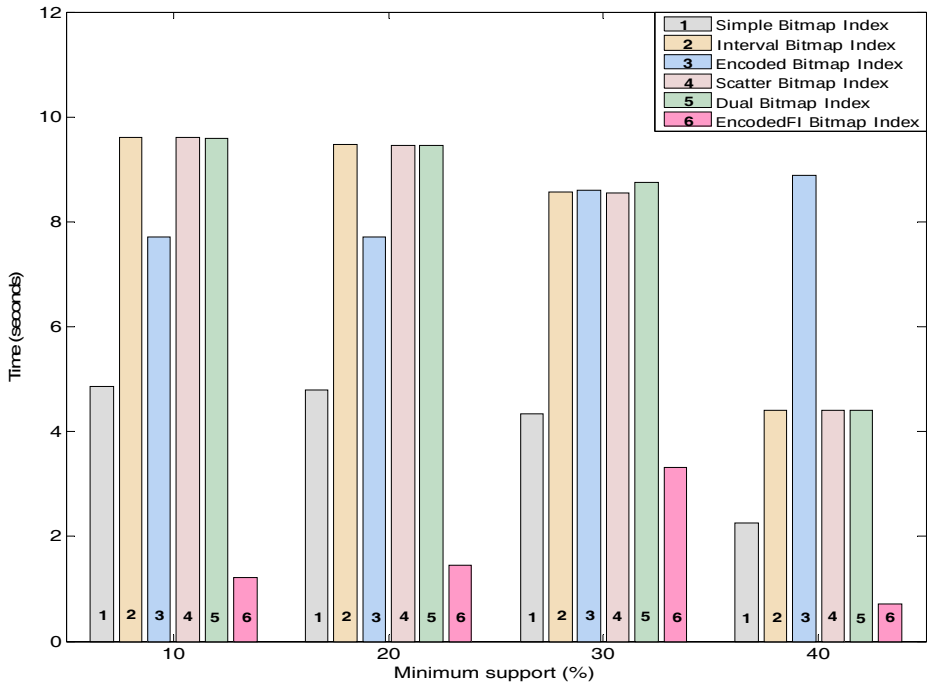


ภาพประกอบ 5-8 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อยของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ตาราง 5-10 เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อยบนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถาม (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	4.866667	9.603333	7.710000	9.606667	9.600000	1.216667
20	4.786667	9.480000	7.720000	9.453333	9.450000	1.460000
30	4.336667	8.573333	8.593333	8.556667	8.750000	3.310000
40	2.260000	4.410000	8.890000	4.410000	4.410000	0.720000

จากตาราง 5-10 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-9

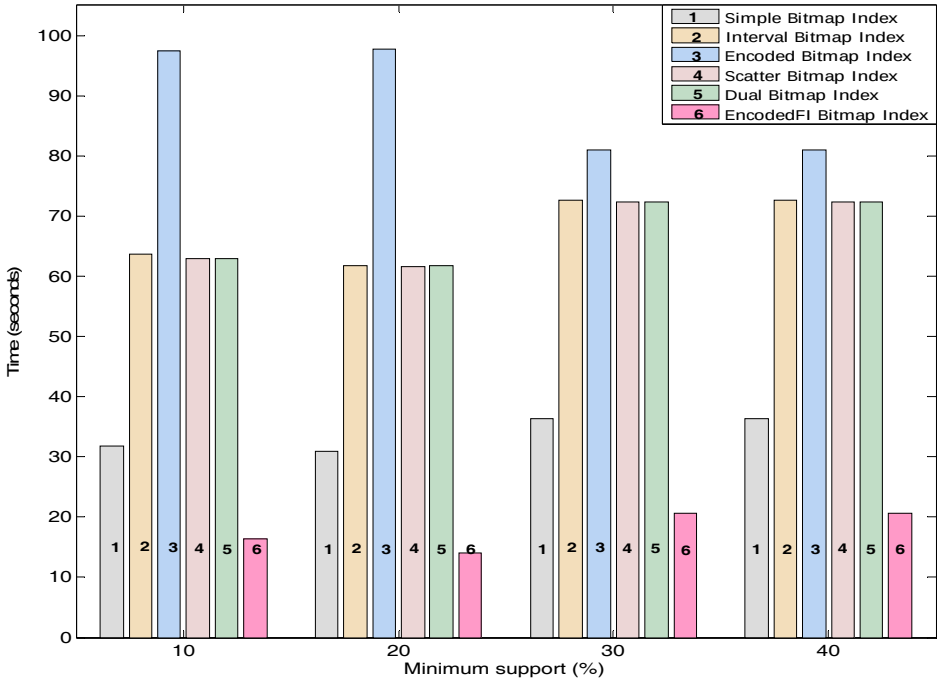


ภาพประกอบ 5-9 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อยของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 150

ตาราง 5-11 เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ค่าความถี่ขั้นต่ำ (%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	31.730000	63.590000	97.380000	62.890000	62.870000	16.320000
20	30.950000	61.790000	97.710000	61.600000	61.780000	14.010000
30	36.344000	72.608400	80.976000	72.269600	72.286800	20.600000
40	36.344000	72.608400	80.976000	72.269600	72.286800	20.600000

จากตาราง 5-11 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-10

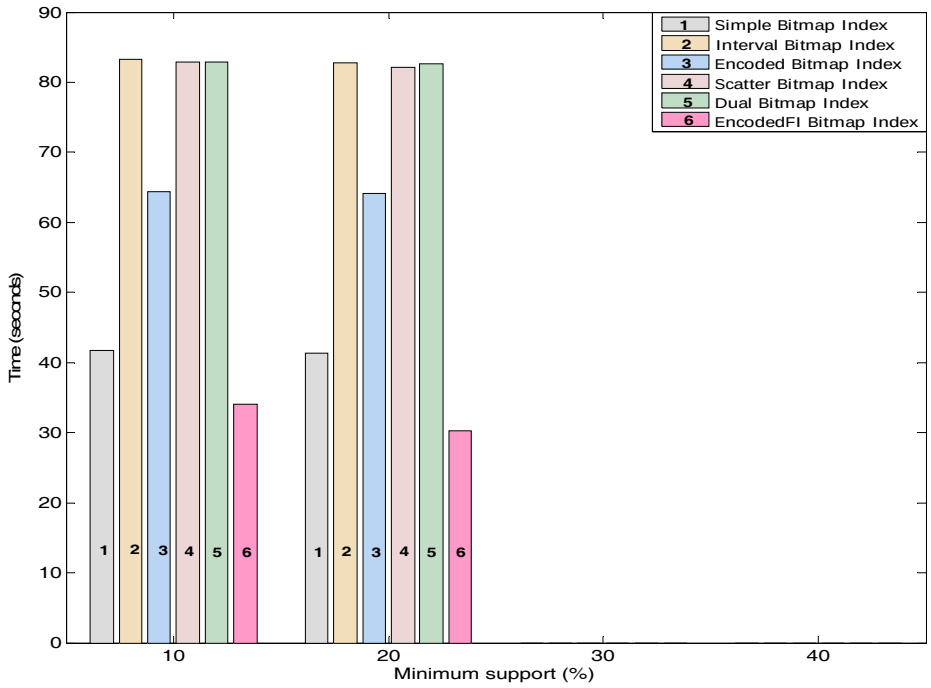


ภาพประกอบ 5-10 กราฟเปรียบเทียบเวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อยของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ตาราง 5-12 เวลาที่ใช้ในการสอบถามแบบสมาชิกบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

ค่าความถี่ขั้นต่ำ(%)	เวลาที่ใช้ในการสอบถามเฉลี่ย (วินาที)					
	ดัชนีบิตแมปแบบพื้นฐาน (Simple)	ดัชนีบิตแมปแบบช่วง (Interval)	ดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encoded)	ดัชนีบิตแมปแบบกระจาย (Scatter)	ดัชนีบิตแมปแบบคู่กัน (Dual)	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (EncodedFI)
10	41.680000	83.300000	64.430000	82.900000	82.930000	34.070000
20	41.380000	82.710000	64.100000	82.160000	82.690000	30.280000
30	-	-	-	-	-	-
40	-	-	-	-	-	-

จากตาราง 5-12 นำเวลาที่ใช้ในการสอบถามเฉลี่ยแต่ละการสอบถามมาเขียนกราฟได้ดังภาพประกอบ 5-11



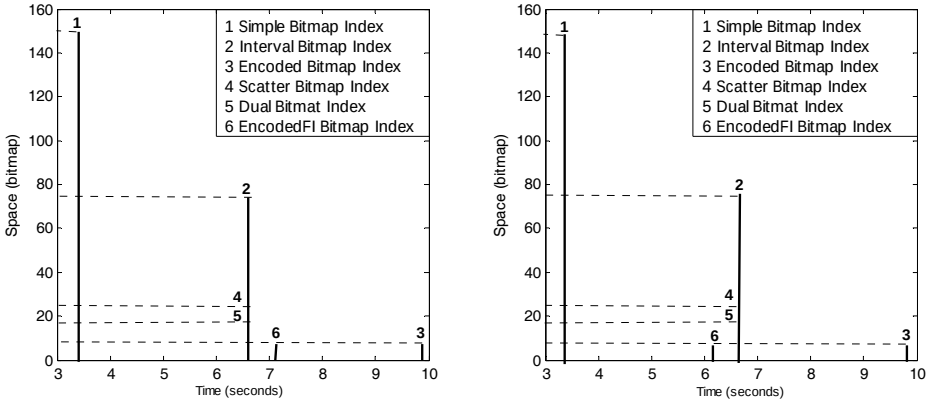
ภาพประกอบ 5-11 กราฟแสดงเวลาที่ใช้ในการสอบถามแบบสมาชิกเฉลี่ยแต่ละกาสอบถามบนการสอบถามที่มีกลุ่มข้อมูลที่ปรากฏบ่อย เปรียบเทียบกับดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้มีค่าเท่ากับ 1,000

5.3 ประสิทธิภาพในแง่ Space-Time Trade-off

จากหัวข้อ 5.1 และ 5.2 จะเห็นว่าดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยยังคงประสิทธิภาพในแง่พื้นที่ นั่นคือยังคงใช้พื้นที่เท่ากับดัชนีบิตแมปแบบเข้ารหัสทั่วไปซึ่งน้อยที่สุด และสำหรับการสอบถามแบบสมาชิกนั้นมีประสิทธิภาพดีกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป และมีโอกาสมีประสิทธิภาพดีกว่าดัชนีบิตแมปแบบอื่น ขึ้นอยู่กับชุดการสอบถามที่เคยมีมาในอดีต แต่สำหรับการวิเคราะห์ในแง่ Space-Time Trade-off ของดัชนีบิตแมปแต่ละชนิด โดยพิจารณาจากพื้นที่ใต้กราฟที่แสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการสร้างดัชนีบิตแมปกับเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแต่ละชนิดดังภาพประกอบ 5-12, 5-13, 5-14 และ 5-15

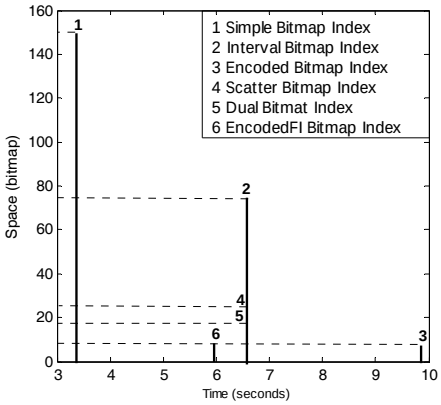
จากภาพประกอบ 5-12, 5-13, 5-14 และ 5-15 เมื่อพิจารณาพื้นที่ใต้กราฟของดัชนีบิตแมปแต่ละชนิด พบว่า พื้นที่ใต้กราฟของดัชนีบิตแมปแบบพื้นฐานมีลักษณะเป็นรูปสี่เหลี่ยมที่มีความสูงมาก เนื่องจากใช้พื้นที่ในการจัดเก็บดัชนีมาก จึงทำให้พื้นที่ใต้กราฟมาก ส่วนพื้นที่ใต้กราฟของดัชนีบิตแมปแบบเข้ารหัสทั่วไปมีลักษณะเป็นรูปสี่เหลี่ยมที่มีฐานกว้างมาก เนื่องจากใช้เวลาในการสอบถามแบบสมาชิกมาก สำหรับดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน ใช้เวลาในการสอบถามแบบสมาชิกใกล้เคียงกัน จึงทำให้มีฐานกว้างใกล้เคียงกันมาก แต่เมื่อพิจารณาจากความสูงของกราฟ พบว่าดัชนีบิตแมปแบบช่วงมีความสูงมากกว่าดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบกระจายมีความสูงมากกว่าดัชนีบิตแมปแบบคู่กัน ทั้งนี้เนื่องจากดัชนีบิตแมปแบบช่วงใช้พื้นที่ในการจัดเก็บดัชนีมากกว่าดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบกระจายใช้พื้นที่ในการจัดเก็บดัชนีมากกว่าดัชนีบิตแมปแบบคู่กัน จึงได้ว่าพื้นที่ใต้กราฟของดัชนีบิตแมปทั้ง 3 ชนิด เรียงจากมากไปหาน้อยคือ พื้นที่ใต้กราฟของดัชนีบิตแมปแบบช่วง พื้นที่ใต้กราฟของดัชนีบิตแมปแบบกระจาย และพื้นที่ใต้กราฟของดัชนีบิตแมปแบบคู่กัน สำหรับพื้นที่ใต้กราฟของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยจะมีความสูงเท่ากับดัชนีบิตแมปแบบเข้ารหัสทั่วไป เนื่องจากใช้พื้นที่ในการจัดเก็บดัชนีเท่ากัน แต่สำหรับฐานของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีความกว้างน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป เนื่องจากดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยใช้เวลาในการสอบถามข้อมูลแบบสมาชิกน้อยกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป นอกจากนี้จากภาพประกอบ 5-12, 5-13 และ 5-14 ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีความกว้างของฐานน้อยกว่าดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน และจากภาพประกอบ 5-13(ก) และ 5-13(ข) ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีความกว้างของฐานน้อยกว่าดัชนีบิตแมปแบบพื้นฐานด้วย และเมื่อเปรียบเทียบพื้นที่ใต้กราฟของดัชนีบิตแมปทั้ง 6 ชนิด จากภาพประกอบ 5-12, 5-13, 5-14 และ 5-15 จะเห็นว่า พื้นที่ใต้

กราฟของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยน้อยที่สุด หมายความว่าดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพในแง่ Space-Time Trade-off ดีที่สุด

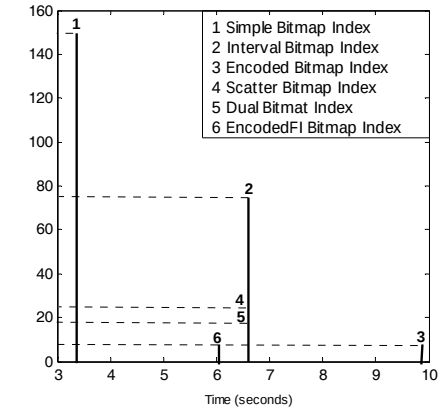


(ก) Minimum support = 10%

(ข) Minimum support = 20%

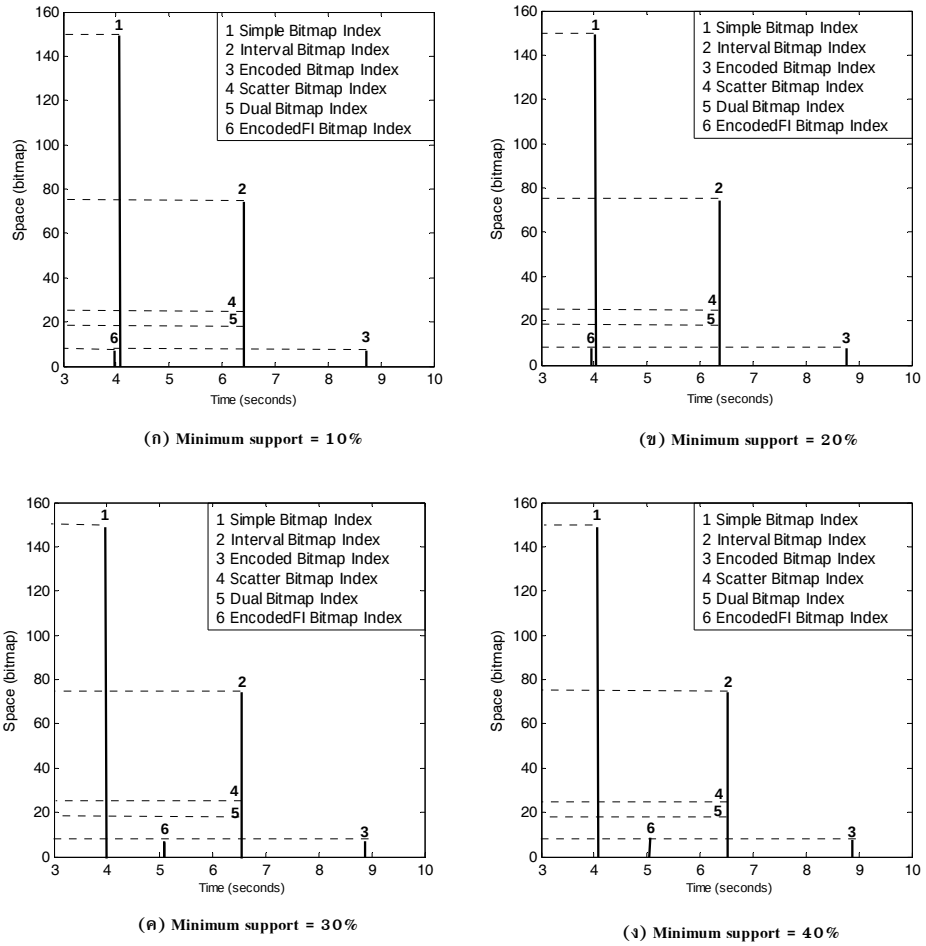


(ค) Minimum support = 30%



(ง) Minimum support = 40%

ภาพประกอบ 5-12 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนี และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 150



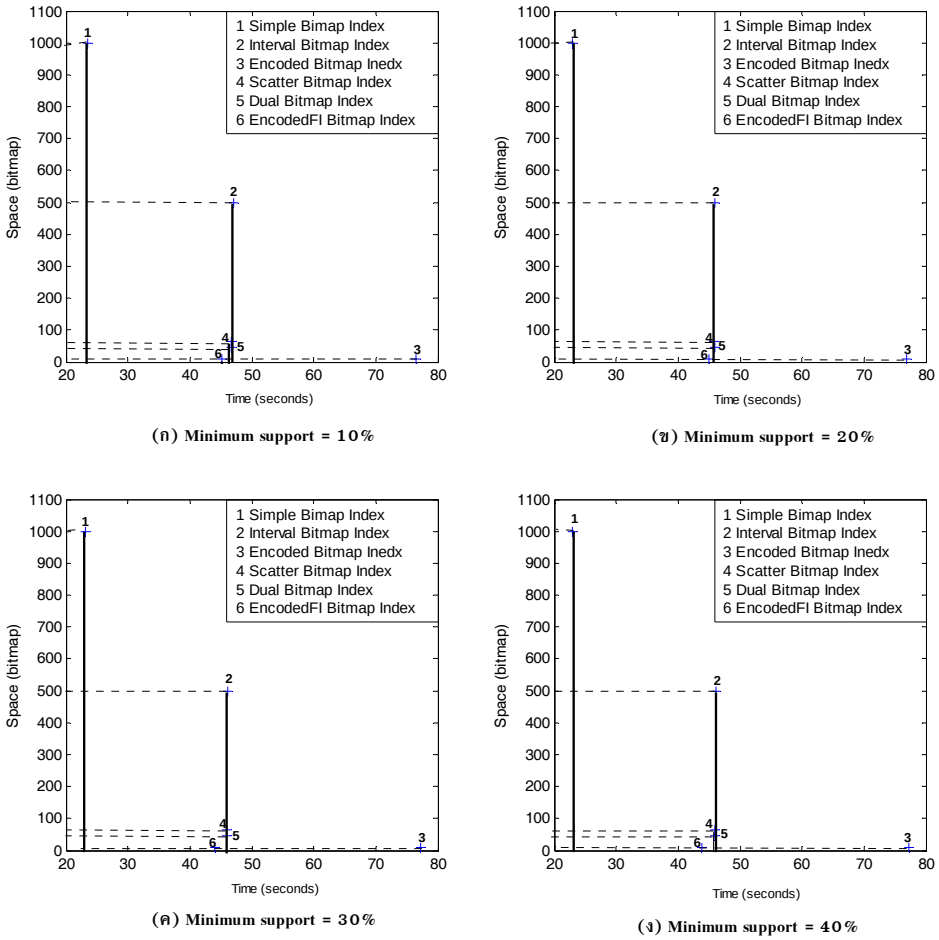
(ก) Minimum support = 10%

(ข) Minimum support = 20%

(ค) Minimum support = 30%

(ง) Minimum support = 40%

ภาพประกอบ 5-13 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนี
และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด
บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 150



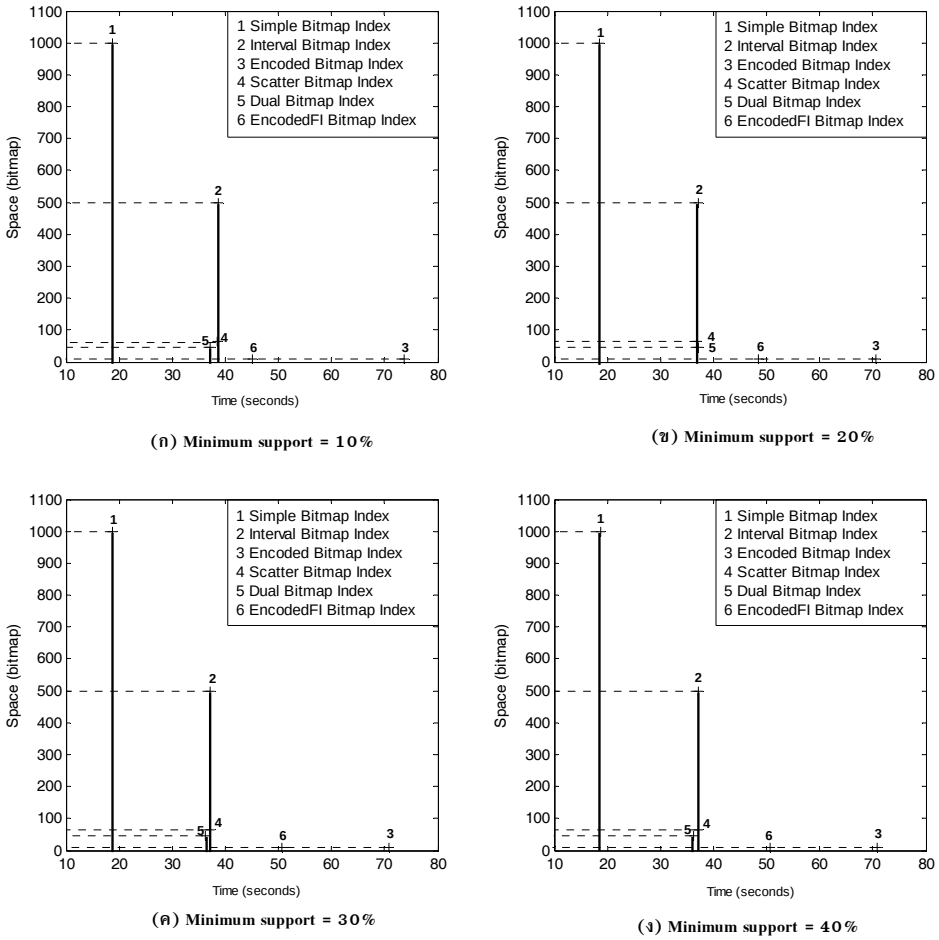
(ก) Minimum support = 10%

(ข) Minimum support = 20%

(ค) Minimum support = 30%

(ง) Minimum support = 40%

ภาพประกอบ 5-14 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนี และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด บนชุดการสอบถามที่ 1 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 1,000



(ก) Minimum support = 10%

(ข) Minimum support = 20%

(ค) Minimum support = 30%

(ง) Minimum support = 40%

ภาพประกอบ 5-15 กราฟแสดงความสัมพันธ์ระหว่างพื้นที่ที่ใช้ในการจัดเก็บดัชนี
และเวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปทั้ง 6 ชนิด
บนชุดการสอบถามที่ 2 เมื่อคาร์ดินอร์ลิตี้เท่ากับ 1,000

บทที่ 6

บทสรุปและข้อเสนอแนะ

งานวิทยานิพนธ์นี้ ได้นำเสนอเทคนิคการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิก ด้วยการนำกลุ่มข้อมูลที่ปรากฏบ่อย (Frequent itemsets) มาช่วยในการสร้างดัชนีบิตแมปแบบเข้ารหัส โดยขั้นตอนการเพิ่มประสิทธิภาพประกอบด้วย 3 ขั้นตอนหลักคือ ขั้นตอนที่ 1 Attribute Values Extraction (การสกัดค่าของแอทริบิวต์ที่ต้องการสร้างดัชนีจาก Workload) ขั้นตอนที่ 2 FI Processing (การหากลุ่มข้อมูลที่ปรากฏบ่อย) และขั้นตอนที่ 3 Encoding (การเข้ารหัส) ผลลัพธ์ที่ได้คือ สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามแบบสมาชิกให้เหลือน้อยที่สุดได้ ทำให้ประสิทธิภาพในแง่เวลาที่ใช้ในการสอบถามแบบสมาชิกเพิ่มขึ้น ซึ่งงานวิทยานิพนธ์นี้ ได้รับการตีพิมพ์ผลงานวิจัยในระดับประเทศและนานาชาติ ดังนี้

เรื่อง การใช้กฎความสัมพันธ์เพื่อเพิ่มประสิทธิภาพของการทำดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิก ตีพิมพ์ใน The 5th Joint Conference on Computer Science and Software Engineering (JCSSE 2008) จัดที่จังหวัดกาญจนบุรี ประเทศไทย วันที่ 7-9 พฤษภาคม 2551 ดังแสดงในภาคผนวก ข

เรื่อง การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามข้อมูลแบบสมาชิกโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อย ตีพิมพ์ใน The 12th Computer Science and Engineering Conference (NCSEC 2008) จัดที่จังหวัดชลบุรี ประเทศไทย วันที่ 20-21 พฤศจิกายน 2551 ดังแสดงในภาคผนวก ค

เรื่อง Optimizing Encoded Bitmap Index using Frequent Itemsets Mining ตีพิมพ์ใน International Conference on Computer and Electrical Engineering (ICCEE 2008) จัดที่จังหวัดภูเก็ต ประเทศไทย วันที่ 21-22 ธันวาคม 2551 ดังแสดงในภาคผนวก ง

6.1 บทสรุป

เทคนิคการทำดัชนีแบบบิตแมปเป็นเทคนิคที่ได้รับความนิยมในระบบสนับสนุนการตัดสินใจของผู้บริหาร เนื่องจากเป็นวิธีการทำดัชนีที่อยู่บนพื้นฐานของบิตแมป จึงสนับสนุนการทำงานของฮาร์ดแวร์ ทำให้การประมวลผลมีความรวดเร็ว เหมาะกับระบบสนับสนุนการตัดสินใจของผู้บริหารที่มีลักษณะการสอบถามที่ซับซ้อนและเป็นแบบทันทีทันใด โดยที่ผ่านมามีการพัฒนาขั้นตอนการสร้างดัชนีบิตแมปต่อ ๆ กันมา เพื่อให้สอดคล้องกับความต้องการและให้มีประสิทธิภาพยิ่งขึ้น

ในอดีตที่ผ่านมามีเทคนิคการทำดัชนีบิตแมปที่น่าสนใจ จากการศึกษาสามารถแบ่งเทคนิคการทำดัชนีบิตแมปได้เป็น 2 กลุ่ม ได้แก่ การทำดัชนีบิตแมปแบบบีบอัด และการทำดัชนีบิตแมปแบบไม่บีบอัด สำหรับงานวิทยานิพนธ์นี้ สนใจเทคนิคการทำดัชนีบิตแมปแบบไม่บีบอัด เทคนิคการทำดัชนีบิตแมปแบบไม่บีบอัดที่น่าสนใจ ได้แก่ ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบเข้ารหัส ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน ซึ่งดัชนีบิตแมปแต่ละชนิดมีวิธีการสร้างดัชนีและการสอบถามที่ไม่เหมือนกัน จึงทำให้ประสิทธิภาพในแต่ละด้านแตกต่างกันออกไป คือ ดัชนีบิตแมปแบบพื้นฐานมีประสิทธิภาพในด้านเวลาที่ใช้ในการสอบถามดีที่สุด แต่ประสิทธิภาพในด้านพื้นที่ที่ใช้ในการจัดเก็บดัชนีแย่ที่สุด ส่วนดัชนีบิตแมปแบบเข้ารหัสมีประสิทธิภาพในด้านพื้นที่ที่ใช้ในการจัดเก็บดัชนีดีที่สุด แต่ประสิทธิภาพในด้านเวลาที่ใช้ในการสอบถามข้อมูลแย่ที่สุด ส่วนดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กัน มีประสิทธิภาพในด้านเวลาที่ใช้ในการสอบถามใกล้เคียงกัน ส่วนประสิทธิภาพด้านพื้นที่ที่ใช้ในการจัดเก็บข้อมูล ดัชนีบิตแมปแบบคู่กันใช้พื้นที่น้อยกว่าดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบกระจายใช้พื้นที่น้อยกว่าดัชนีบิตแมปแบบช่วง

ในงานวิทยานิพนธ์นี้ ได้นำเสนอขั้นตอนการเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิก โดยใช้หลักการที่ว่า แต่ละค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อย ๆ เมื่อมีการสอบถามด้วยกันอีก จะสามารถลดจำนวนบิตแมปเวกเตอร์ให้เหลือน้อยที่สุดได้ ทำให้ประสิทธิภาพในแง่เวลาที่ใช้ในการสอบถามแบบสมาชิกเพิ่มขึ้น ซึ่งสามารถพิจารณาการเปรียบเทียบลักษณะที่สำคัญของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยที่นำเสนอกับดัชนีบิตแมปที่เคยมีมา ได้ดังตาราง 6-1

ตาราง 6-1 ลักษณะที่สำคัญของดัชนีบิตแมปทั้ง 6 ชนิด (C คือ จำนวนค่าที่เป็นไปได้ของแอทริบิวต์ที่เลือกมาสร้างดัชนี)

ลักษณะที่สำคัญ	ดัชนีบิตแมป					
	แบบพื้นฐาน	**แบบช่วง**	**แบบกระจาย**	**แบบเข้ารหัสทั่วไป**	**แบบคู่กัน**	**แบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย**
จำนวนบิตแมปเวกเตอร์ที่ใช้ในการแทนแต่ละค่าของแอทริบิวต์	1	$\leq \lfloor C/2 \rfloor$	2	$\lceil \log_2 C \rceil$	2	$\lceil \log_2 C \rceil$
จำนวนค่าของแอทริบิวต์ที่ถูกแทนด้วยแต่ละบิตแมปเวกเตอร์	1	$\lfloor C/2 \rfloor$	$\leq \lceil \sqrt{C} \rceil + 1$	C	$\left\lceil \sqrt{2C + \frac{1}{4}} + \frac{1}{2} \right\rceil - 1$	C
จำนวนบิตแมปเวกเตอร์ที่ใช้ในการแทนค่าของแอทริบิวต์ที่มีคาร์ดินอลิตี้ C (พื้นที่)	C	$\lceil C/2 \rceil$	$\lceil 2\sqrt{C} \rceil$	$\lceil \log_2 C \rceil$	$\left\lceil \sqrt{2C + \frac{1}{4}} + \frac{1}{2} \right\rceil$	$\lceil \log_2 C \rceil$
จำนวนบิตแมปเวกเตอร์ที่อ่าน : จำนวนครั้งในการดำเนินการตรรกะสำหรับการสอบถามแบบค่าสมาชิก (เวลา)	k : k-1 (OR)	2k : 3k-1 (k AND, k-1OR, k NOT)	2k : 2k-1 (k AND, k-1OR)	k $\lceil \log_2 C \rceil$:use mapping table, k-1 (OR)	2k : 2k-1 (k AND, k-1 OR)	1:0

จากตาราง 6-1 จะเห็นว่า ดัชนีบิตแมปแต่ละชนิดมีลักษณะบางประการที่แตกต่างกัน ส่งผลให้มีข้อดีและข้อจำกัดที่แตกต่างกัน เมื่อพิจารณาการสอบถามแบบสมาชิก พบว่าดัชนีบิตแมปแต่ละชนิดมีลักษณะ ดังนี้

- ดัชนีบิตแมปแบบพื้นฐาน แต่ละบิตแมปเวกเตอร์ใช้แทนค่าแอทริบิวต์ได้เพียงค่าเดียว ดังนั้นในการสอบถามแบบสมาชิก k ค่า จะต้องตรวจสอบ k บิตแมปเวกเตอร์ และดำเนินการตรรกะ OR จำนวน k-1 ครั้ง
- ดัชนีบิตแมปแบบช่วง ใช้จำนวนบิตแมปเวกเตอร์ในการแทนค่าแต่ละแอทริบิวต์มากเกินไป เพราะในการสอบถามแต่ละค่ามีการอ่านเพียง 2 บิตแมปเวกเตอร์เท่านั้น จึงควรใช้เพียง 2 บิตแมปเวกเตอร์ในการแทนแต่ละค่าของแอทริบิวต์ ในการสอบถามแบบสมาชิก k ค่า จะต้องตรวจสอบ $2k$ บิตแมปเวกเตอร์ ดำเนินตรรกะ AND จำนวน k ครั้ง ดำเนินตรรกะ NOT จำนวน k ครั้ง และดำเนินการตรรกะ OR จำนวน k-1 ครั้ง จึงใช้เวลาในการสอบถามแบบสมาชิกมากกว่าดัชนีบิตแมปแบบพื้นฐาน
- ดัชนีบิตแมปแบบเข้ารหัสทั่วไป ใช้จำนวนบิตแมปเวกเตอร์ในการแทนแต่ละค่าทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) ทำให้ในการสอบถามแต่ละค่าต้องตรวจสอบทุกบิตแมปเวกเตอร์เปรียบเทียบกับตารางเทียบค่า ในการสอบถามแบบสมาชิก k ค่า จึงต้องตรวจสอบ $\lceil \log_2 C \rceil k$ บิตแมปเวกเตอร์เปรียบเทียบกับตารางเทียบค่า และดำเนินการตรรกะ OR จำนวน k-1 ครั้ง จึงใช้เวลาในการสอบถามแบบสมาชิกมากที่สุด
- ดัชนีบิตแมปแบบกระจาย มีบางบิตแมปเวกเตอร์ถูกใช้อย่างไม่คุ้มค่า โดยเฉพาะบิตแมปเวกเตอร์ Z^0 ที่ใช้แทนค่าของแอทริบิวต์ได้เพียงบิตเดียวเท่านั้น ซึ่งในแต่ละค่าของแอทริบิวต์ใช้ 2 บิตแมปเวกเตอร์ในการแทนค่า ดังนั้นในการสอบถามแบบสมาชิก k ค่า จึงต้องตรวจสอบ $2k$ บิตแมปเวกเตอร์ และมีการดำเนินการตรรกะ AND จำนวน k ครั้ง และดำเนินตรรกะ OR จำนวน k-1 ครั้ง จึงใช้เวลาในการสอบถามแบบสมาชิกใกล้เคียงกับดัชนีบิตแมปแบบช่วง
- ดัชนีบิตแมปแบบคู่กัน ใช้จำนวนบิตแมปเวกเตอร์ได้อย่างคุ้มค่าขึ้น แต่ยังคงใช้ 2 บิตแมปเวกเตอร์ในการแทนแต่ละค่าของแอทริบิวต์ ซึ่งยังคงใช้จำนวนบิตแมปเวกเตอร์มากกว่าดัชนีบิตแมปแบบพื้นฐาน เนื่องจากดัชนีบิตแมปแบบพื้นฐานใช้เพียง 1 บิตแมปเวกเตอร์ในการแทนแต่ละค่า ดังนั้นในการสอบถามแบบสมาชิก k ค่า จึงต้องตรวจสอบ $2k$ บิตแมปเวกเตอร์ และมีการดำเนินการตรรกะ AND จำนวน k ครั้ง ดำเนินตรรกะ OR จำนวน k-1 ครั้ง จึงใช้เวลาในการสอบถามแบบสมาชิกใกล้เคียงกับดัชนีบิตแมปแบบกระจายและดัชนีบิตแมปแบบช่วง
- ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย ใช้จำนวนบิตแมปเวกเตอร์ในการแทนแต่ละค่าทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) เหมือนดัชนีบิตแมปแบบเข้ารหัสทั่วไป ในการสอบถามแต่ละค่าจึงต้องตรวจสอบทุกบิตแมปเวกเตอร์เปรียบเทียบกับตารางเทียบค่า ในการสอบถามแบบค่าสมาชิก k ค่า จะต้องตรวจสอบ

$\lceil \log_2 C \rceil k$ บิตแมปเวกเตอร์เปรียบเทียบกับตารางเทียบค่า แต่สำหรับค่าของแอทริบิวต์ที่เคยถูกสอบถามด้วยกัน จะสามารถลดจำนวนบิตแมปเวกเตอร์ให้เหลือน้อยที่สุดได้ กรณีที่ดีที่สุดตรวจสอบเพียงหนึ่งบิตแมปเวกเตอร์และไม่มีการดำเนินการตรรกะ จึงมีโอกาสใช้เวลาในการสอบถามแบบสมาชิกดีที่สุด แต่กรณีเป็นการสอบถามค่าของแอทริบิวต์ที่ไม่เคยถูกสอบถามด้วยกัน จะไม่สามารถลดจำนวนบิตแมปเวกเตอร์ให้เหลือน้อยที่สุดได้ ทั้งนี้เวลาที่ใช้ในการสอบถามแบบสมาชิกของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย จึงขึ้นอยู่กับการสอบถามที่เคยมีมาในอดีต

ในการคิดค้นเทคนิคการสร้างดัชนีบิตแมปมีวัตถุประสงค์ในการนำไปใช้และความต้องการที่แตกต่างกัน ได้แก่ รูปแบบการสอบถามที่ต้องการ เช่น การสอบถามแบบค่าเท่ากัน การสอบถามแบบสมาชิก และการสอบถามแบบเป็นช่วง เป็นต้น หรือความต้องการเกี่ยวกับประสิทธิภาพในแง่ต่าง ๆ เช่น ประสิทธิภาพในแง่เวลา ประสิทธิภาพในแง่พื้นที่ และประสิทธิภาพในแง่ Space-Time Trade-off สำหรับวิทยานิพนธ์นี้ ระบุรูปแบบการสอบถามเป็นการสอบถามแบบสมาชิก จึงขอสรุปประสิทธิภาพที่ต้องการสำหรับการสอบถามแบบสมาชิกและดัชนีบิตแมปที่ควรเลือกใช้ ดังตาราง 6-2

ตาราง 6-2 สรุปประสิทธิภาพที่ต้องการสำหรับการสอบถามแบบสมาชิกและดัชนีบิตแมปที่ควรเลือกใช้

ประสิทธิภาพที่ต้องการ	ดัชนีบิตแมปที่ควรเลือกใช้
ประสิทธิภาพในแง่ของพื้นที่	ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย
ประสิทธิภาพในแง่ของเวลา	ดัชนีบิตแมปแบบพื้นฐานหรือดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย (ขึ้นอยู่กับชุดข้อมูลสอบถามในอดีต)
ประสิทธิภาพในแง่ Space-Time Trade-off	ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

จากการวิเคราะห์และผลการทดลองเปรียบเทียบดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยกับดัชนีบิตแมปที่เคยมีมา พบว่า ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยยังคงใช้พื้นที่ที่ใช้ในการจัดเก็บดัชนีเท่ากับดัชนีบิตแมปแบบเข้ารหัสทั่วไป ซึ่งน้อยกว่าดัชนีบิตแมปแบบอื่น ๆ เหมาะสำหรับการสร้างดัชนีบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้สูง และต้องการประสิทธิภาพในแง่พื้นที่ โดยดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพในแง่เวลาที่ใช้ในการสอบถามแบบสมาชิกดีกว่าดัชนีบิตแมปแบบเข้ารหัสทั่วไป

และมีโอกาสดีกว่าดัชนีบิตแมปแบบอื่นขึ้นอยู่กับชุดข้อมูลสอบถามในอดีต ซึ่งในกรณีที่ดีที่สุด ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยมีประสิทธิภาพในแง่เวลาที่ใช้ในการสอบถามแบบสมาชิกดีที่สุด นอกจากนี้เมื่อพิจารณาประสิทธิภาพในแง่ Space-Time Trade-off ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยยังมีประสิทธิภาพดีที่สุด จากที่กล่าวมาจะได้ว่า เมื่อต้องการทำดัชนีบนแอทริบิวต์ที่มีคาร์ดินอร์ลิตี้สูง และต้องการประสิทธิภาพทั้งในแง่พื้นที่ เวลา และ Space-Time Trade-off ควรเลือกสร้างดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

6.2 ข้อเสนอแนะและงานในอนาคต

1. ในการหากลุ่มข้อมูลที่ปรากฏบ่อย นอกจากจะใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อยที่นำเสนอแล้ว ยังสามารถนำเทคนิคการทำเหมืองข้อมูลแบบอื่นมาใช้ได้ เช่น Clustering

2. เนื่องจากประสิทธิภาพของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยที่นำเสนอขึ้นอยู่กับชุดการสอบถามในอดีต ดังนั้นชุดการสอบถามที่นำมาใช้ควรสอดคล้องกับเป้าหมายที่ต้องการ

อย่างไรก็ตามเมื่อมีการสอบถามแบบค่าเท่ากัน ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อยยังคงใช้เวลามากที่สุด เนื่องจากต้องตรวจสอบทุกบิตแมปเวกเตอร์และมีการเปรียบเทียบกับตารางเทียบค่า ดังนั้นสำหรับงานในอนาคตจะเป็นการหาวิธีในการสร้างดัชนีบิตแมปแบบเข้ารหัสให้มีประสิทธิภาพด้านเวลาที่ใช้ในการสอบถามแบบค่าเท่ากัน

บรรณานุกรม

เบญจมาศ เต็มอุดม และ ดร.ภัทรชัย ลลิตโรจน์วงศ์. 2002. การพัฒนาระบบคลังข้อมูล. บทความวิชาการ สาร NECTEC. พฤศจิกายน-ธันวาคม.

C. J. Bontempo and C. M. Saracco. 1998. Accelerating Indexed Searching. http://www.dbpd.com /vault/bontempo.htm (accessed 16/09/07).

C. J. Bontempo and C. M. Saracco. 2002. Coping with Complex Queries in Data Warehouse. InfoDB Volume 10 Number 5. http://www.evaltech.com/wpapers/complexqueries.pdf (accessed 19/03/08).

C. Y. Chan and Y. E. Ioannidis. 1998. Bitmap Index Design and Evaluation. Proceeding of the 1998 ACM SIGMOD international conference on Management of data. pp. 355-366.

C. Y. Chan and Y. E. Ioannidis. 1999. An Efficient Bitmap Encoding Scheme for Selection Queries. Proceedings of the 1999 ACM SIGMOD international conference on management of data. pp. 215-226.

G. Gardarrin, P. Pucheral and F. Wu. 1998. Bitmap Based Algorithms For Mining Association Rules. 14th Bases de Donnes Avances (BDA'98).

H. F. Korth and S. Sudarshan. 2006. Indexing and Hashing. In: Database System Concepts fifth Edition. pp. 489-502.

Intelligent Enterprise. 1999. Indexing Goes a NewDirection. www.wintercorp.com/rwintercolumns/ie_9901.html (accessed 16/09/07)

J. Han, J. Pei and Y. Yin. 2000. Mining Frequent Patterns without Candidate Generation. 2000 ACM SIGMOD Intl. Conference on management of data. pp. 1-12.

J. Han and M. Kamber. 2000. Data Warehouse and OLAP Technology for Data Mining. In: Data Mining: Concepts and Techniques. Chapter 2.

J. Han and M. Kamber. 2000. Mining Association Rules in Large Database. In: Data Mining: Concepts and Techniques. Chapter 6.

J. Pei, J. Han, H. Lu, S. Nishio, S. Tang and D. Yang. 2001. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Database. Proceeding of IEEE ICOM. pp 441-448.

K. Stockinger and K. Wu. 2007. Bitmap Indices for Data Warehouse. In Data Warehouse and OLAP.

K. Wu, E. J. Otoo and A.Shoshani. 2002. Compressing Bitmap Indexes for Faster Search Operation. International Conference on Scientific and Statistical Database Management (SSDBM).

MC. Wu and A. Buchmann. 1998. Encoded Bitmap Indexing for Data Warehouses. Proceeding of the Fourteenth International Conference on Data Engineering. pp. 220-230.

M. J. Zaki and K. Gouda. 2003. Fast Vertical Mining Using Diffsets. Proceeding Ninth ACM SIGKDD Int'l conf. Knowledge Discovery and data Mining. pp. 326-335.

M. Song and S. Sanguthevar. 2006. A Transaction Mapping Algorithm for Frequent Itemsets Mining. IEEE TRANSACTIONS ON KNOWKEDGE AND DATA ENGINEERING. pp. 472-481.

M. Stabno and R. Wrembel. 2007. RLH:Bitmap Compression Technique Based on Run-Length and Huffman Encoding. Proceedings of the ACM tenth international workshop on Data warehousing and OLAP. pp. 14-48.

N. Koudas. 2000. Space Efficient Bitmap Indexing. Proceedings of the ninth international conference on Information and knowledge management. pp. 194-201.

N. Pasquier, Y. Bastide, R. taouil and L. Lakhal. 1999. Efficient mining of association rules using closed itemset lattices. Information System Volume 24 Number 1. pp. 25-46.

N. Wattanakitrungroj and S. Vanichayobon. 2006. Dual Bitmap Index: Space-Time Efficient Bitmap Index for Equality and Membership Queries. International Symposium on Communications and Information Technologies (ISCIT'06). pp. 568-573.

O'Neil. Elizabeth and Patrick. 2007. Bitmap Index Design Choices and Their Performance Implications. 11th International Database Engineering and Applications Symposium. pp. 72-84.

R. Agrawal, T. Imielinski and A. Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD international conference on Management of data SIGMOD.

R. Agrawal and R. Srikant. 1994. Fast Algorithm for mining association rules in large database. Proceeding of the 20th VLDB Conference Santiago. pp. 487-499.

S. Chaudhuri and U. Dayal. 1997. An Overview of Data Warehousing and OLAP Technology. ACM SIGMOD RECORD Volume 26. pp. 65-74.

S. Vanichayobon, J. Manfuekphan and L. Gruenwald. 2006. Scatter Bitmap: Space-time Efficient Bitmap Indexing for Equality and Membership Queries. Proceeding of IEEE International Conferences on cybernetics and Intelligent Systems. pp. 1-6.

S. Ying and Y. Leu. 1999. An Effective Boolean Algorithm for Mining Association Rules in Large Databases. Database Systems for Advanced Applications, 1999. Proceedings., 6th International Conference. pp. 179-186.

Transaction Processing Performance Council (TPC). 2006. TPC-H: An Ad-hoc, Decision Support Benchmark. Standard Specicication Revision 2.1.0. http://wwww.tpc.org/tpch/ default.asp (accessed 19/03/08).

T. Y. Lin, X. Hu and E.Louie. 2003. A Fast Association Rule Algorithm Based On Bitmap and Granular Computing. Fuzzy Systems, 2003. FUZZ '03. The 12th IEEE International Conference. Vol 1. pp. 678- 683.

W. Song, B. Yang and Z. Xu. 2008. Index-BitTableFI: An improved algorithm for mining frequent itemsets. Knowledge-Based System. Vol 21. Issue 6. pp. 507-513.

ภาคผนวก

ภาคผนวก ก

ก.1 การเตรียมข้อมูลเพื่อใช้ในการทดลอง

ข้อมูลที่ใช้ในการทดลองเป็นข้อมูลมาตรฐานจาก TPC-H Benchmark (Transaction Processing Performance Council, 2006) ซึ่งเป็นตัววัดเปรียบเทียบประสิทธิภาพการประมวลผลการสอบถามที่ซับซ้อนในระบบสนับสนุนการตัดสินใจ (Decision support)

ข้อมูลที่ใช้ในการทดลองมี 2 ชุด คือ

ข้อมูลชุดที่ 1 แอทริบิวต์ TYPE บน PART Table
มี 1,000,000 แถว คาร์ดินอร์ลิตี้เท่ากับ 150 (C = 150)

ชุดข้อมูลที่ 2 แอทริบิวต์ CLERK บน ORDER Table
มี 1,000,000 แถว คาร์ดินอร์ตี้เท่ากับ 1000 (C=1000)

สำหรับแต่ละแอทริบิวต์ที่เลือกมาสร้างดัชนีบิตแมป จะต้องมีการเตรียมข้อมูลตามขั้นตอนต่อไปนี้

1) เลือกเฉพาะแอทริบิวต์ที่จะนำมาสร้างดัชนีโดยใช้คำสั่ง

awx -F \| '{Print $number}' input_file > output_file

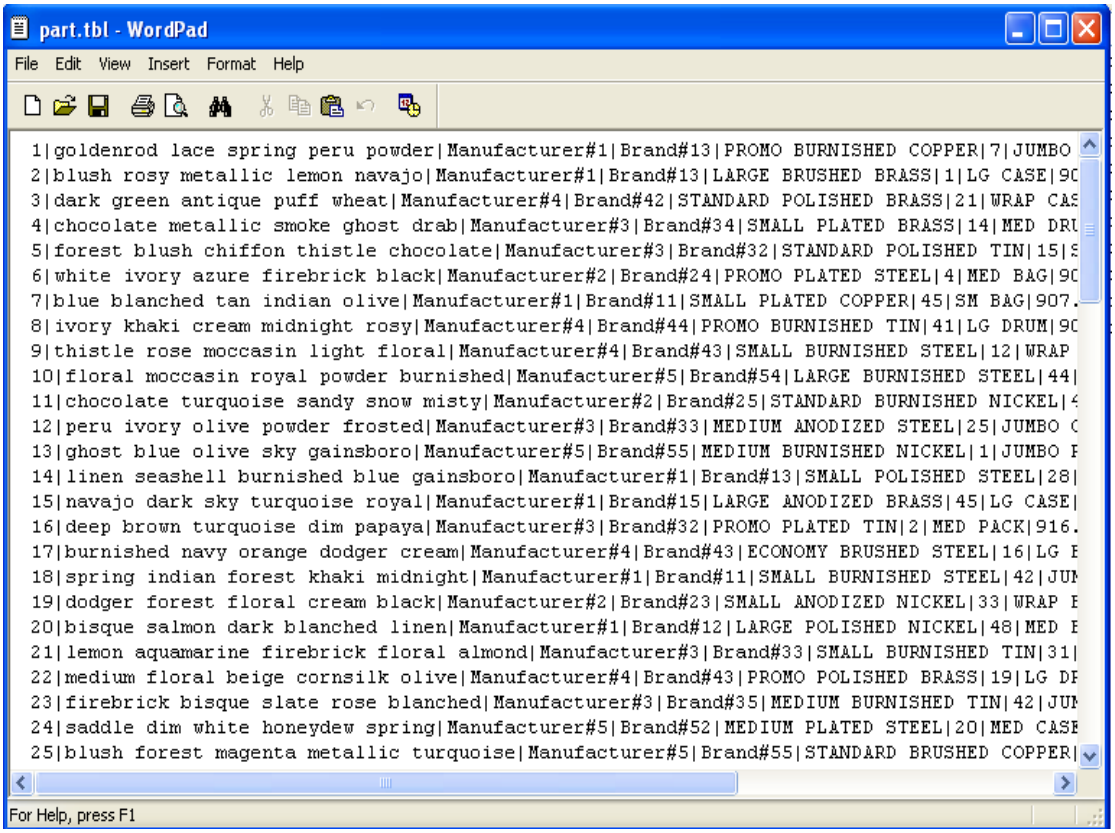
โดย | คือ อักขระที่ใช้คั่นแต่ละค่าของแอทริบิวต์, number คือ แอทริบิวต์ที่ต้องการ, input_file คือ ชื่อไฟล์ตารางที่มีแอทริบิวต์ที่ต้องการ และ output_file คือ ชื่อไฟล์ที่ใช้เก็บข้อมูลของแอทริบิวต์ที่เลือก

ตัวอย่างเช่น awx -F \| '{Print $5}' part.tbl > attr_type.txt เป็นการเลือกแอทริบิวต์ที่ 5 จากตาราง part เก็บผลลัพธ์ไว้ใน attr_type.txt ตัวอย่างไฟล์ part.tbl แสดงดังภาพประกอบ ก-1 และตัวอย่างไฟล์ attr_type.txt แสดงดังภาพประกอบ ก-2 (ไฟล์ด้านซ้าย)

2) เปลี่ยนรูปค่าของแอทริบิวต์ให้เป็นจำนวนเต็มที่ต่อเนื่องกัน เริ่มตั้งแต่ค่า 0 ดังนั้น ค่าแอทริบิวต์ที่ถูกเปลี่ยนค่าแล้วคือ 0, 1, 2, ..., C-1 โดยการสร้าง file.awk ซึ่งเขียนเป็นชุดคำสั่งสำหรับเปลี่ยนค่าในไฟล์แล้วเก็บไว้ในอีกไฟล์หนึ่ง

ตัวอย่างชุดคำสั่งในไฟล์ file.awk

```
/TOCHIBA/ {print 0}          (เปลี่ยนคำว่า TOCHIBA เป็น 0)
/PANASONIC/ {print 1}(เปลี่ยนคำว่า PANASONIC เป็น 1)
/SONY/ {print 2}             (เปลี่ยนคำว่า SONY เป็น 2)
```



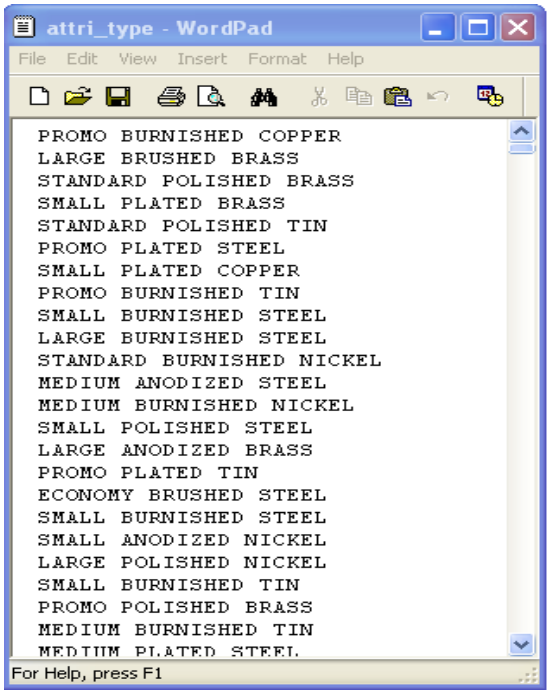
ภาพประกอบ ก-1 ตัวอย่างข้อมูลในไฟล์ part table (part.tbl)

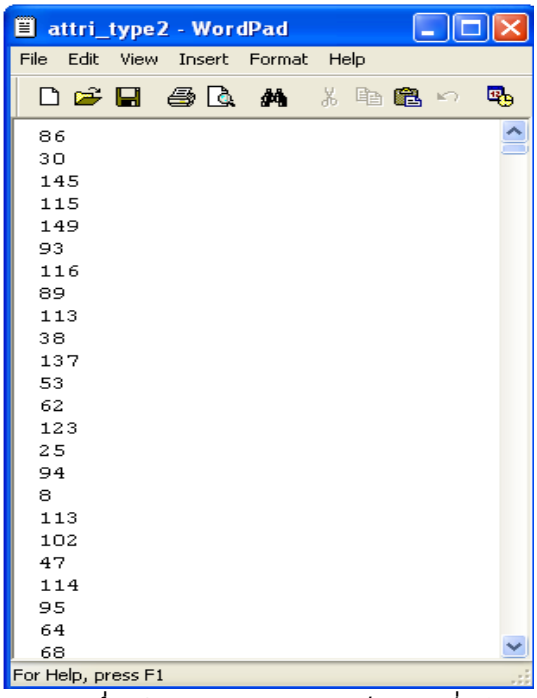
จากนั้นทำให้ชุดคำสั่งใน file.awk ทำงานโดยใช้คำสั่ง

awk -f file.awk input_file > output_file

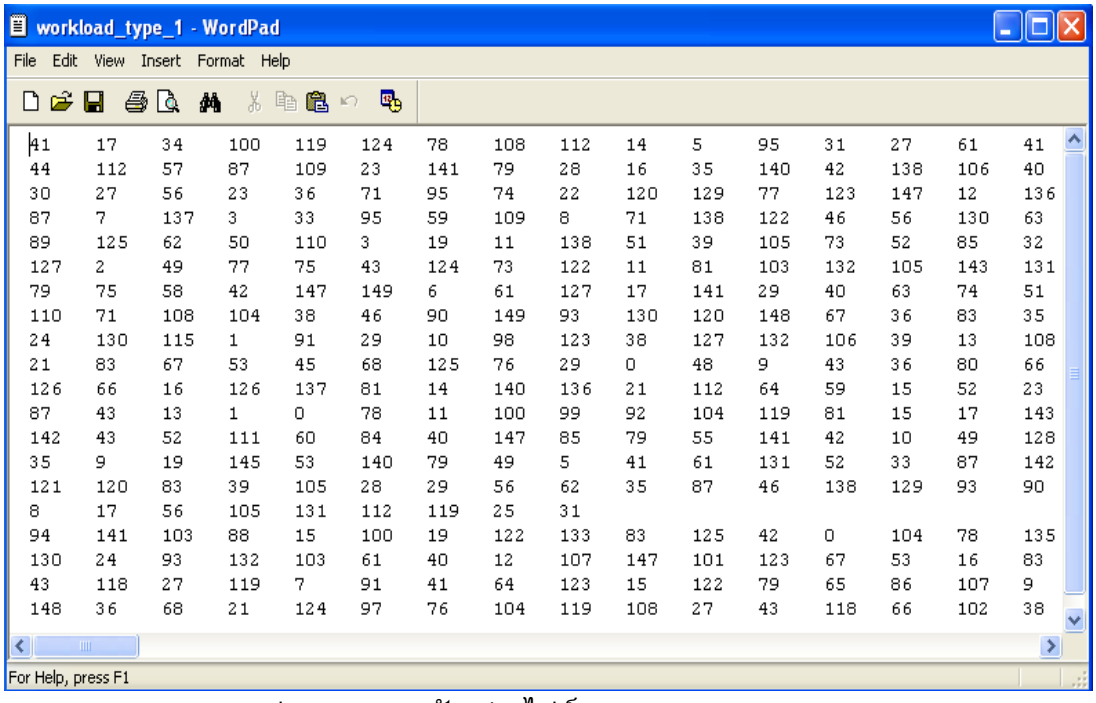
โดยที่ file.awk คือ ไฟล์ชุดคำสั่ง input_file คือ ชื่อไฟล์ที่ใช้เก็บข้อมูลของแอทริบิวต์ที่เลือก และ output_file คือ ชื่อไฟล์ที่ใช้เก็บข้อมูลของแอทริบิวต์ที่เลือกที่มีการเปลี่ยนรูปเป็นจำนวนเต็มที่ต่อเนื่องกัน ตัวอย่างไฟล์ output.file แสดงดังภาพประกอบ ก-2 (ไฟล์ด้านขวา)

สำหรับชุดการสอบถามที่นำมาใช้ในการทดลอง ในแต่ละแอทริบิวต์ที่เลือกมาสร้างดัชนีจะทำการสุ่มชุดการสอบถามจำนวน 2 ชุดการสอบถาม โดยแต่ละชุดการสอบถามมีจำนวน 100 ทรานแซคชัน แต่ละทรานแซคชันจะทำการสุ่มจำนวนสมาชิก (จำนวน Items ในแต่ละทรานแซคชัน) และค่าแอทริบิวต์แต่ละค่า (เป็นจำนวนเต็ม) ที่ถูกสอบถามด้วยกัน ตัวอย่างชุดการสอบถามแสดงดังภาพประกอบ ก-3 โดยตัวเลขตัวแรกในแต่ละทรานแซคชันหมายถึงจำนวนสมาชิกที่ถูกสอบด้วยกันในแต่ละทรานแซคชัน และตัวเลขที่เหลือเป็นค่าที่ถูกสอบถาม





ภาพประกอบ ก-2 (ด้านซ้าย) ตัวอย่างไฟล์ attr_type.txt ซึ่งเก็บค่าของแอทริบิวต์ type ที่สกัดได้จาก part.tbl และ (ด้านขวา) ไฟล์ที่ได้จากการเปลี่ยนรูปค่าของแอทริบิวต์ type เป็นจำนวนเต็ม



ภาพประกอบ ก-3 ตัวอย่างไฟล์ชุดการสอบถาม (Workload)

ก.2 การเขียนโปรแกรมเพื่อทดลองสอบถามแบบค่าสมาชิกบนดัชนีบิตแมป

ในการเขียนโปรแกรมเพื่อทดลองเป็นการเขียนโปรแกรมด้วยตัวแปลภาษาซี (C Compiler) โดยมีการกำหนดค่าที่นิยาม ตัวแปร ฟังก์ชัน และไฟล์ที่สำคัญดังต่อไปนี้

- **การกำหนดค่าคงที่ (define)**

CARDINALITY 150 เป็นการกำหนดค่าให้กับคาร์ดินอร์ลิตี้ ในที่นี้เท่ากับ 150

NUM_RECORD 1000000 เป็นการกำหนดจำนวนแถวข้อมูล ในที่นี้เท่ากับ 1,000,000

NUM_QUERY 100 เป็นการกำหนดจำนวนการสอบถาม ในที่นี้เท่ากับ 100

- **ตัวแปร (variable)**

number ใช้สำหรับเก็บจำนวนสมาชิกในแต่ละการสอบถาม (0 < number < C)

value ใช้สำหรับเก็บค่าที่ต้องการสอบถาม (0<= value < C)

vector ใช้สำหรับเก็บค่าลำดับของบิตแมปเวกเตอร์ที่ต้องการอ่านมาเพื่อดำเนินการตรรกะ

bitsvector ใช้สำหรับเก็บค่าบิตแมปเวกเตอร์ที่อ่านได้จากตารางดัชนี

ansvector ใช้สำหรับเก็บค่าบิตแมปเวกเตอร์ผลลัพธ์จากการดำเนินการตรรกะ ก่อนนำไปเขียนลงไฟล์คำตอบของการสอบถาม

- **ฟังก์ชัน (function)**

QuerySimple(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบพื้นฐาน

QueryInterval(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบช่วง

QueryEncoded(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบเข้ารหัส

QueryScatter(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบกระจาย

QueryDual(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบคู่กัน

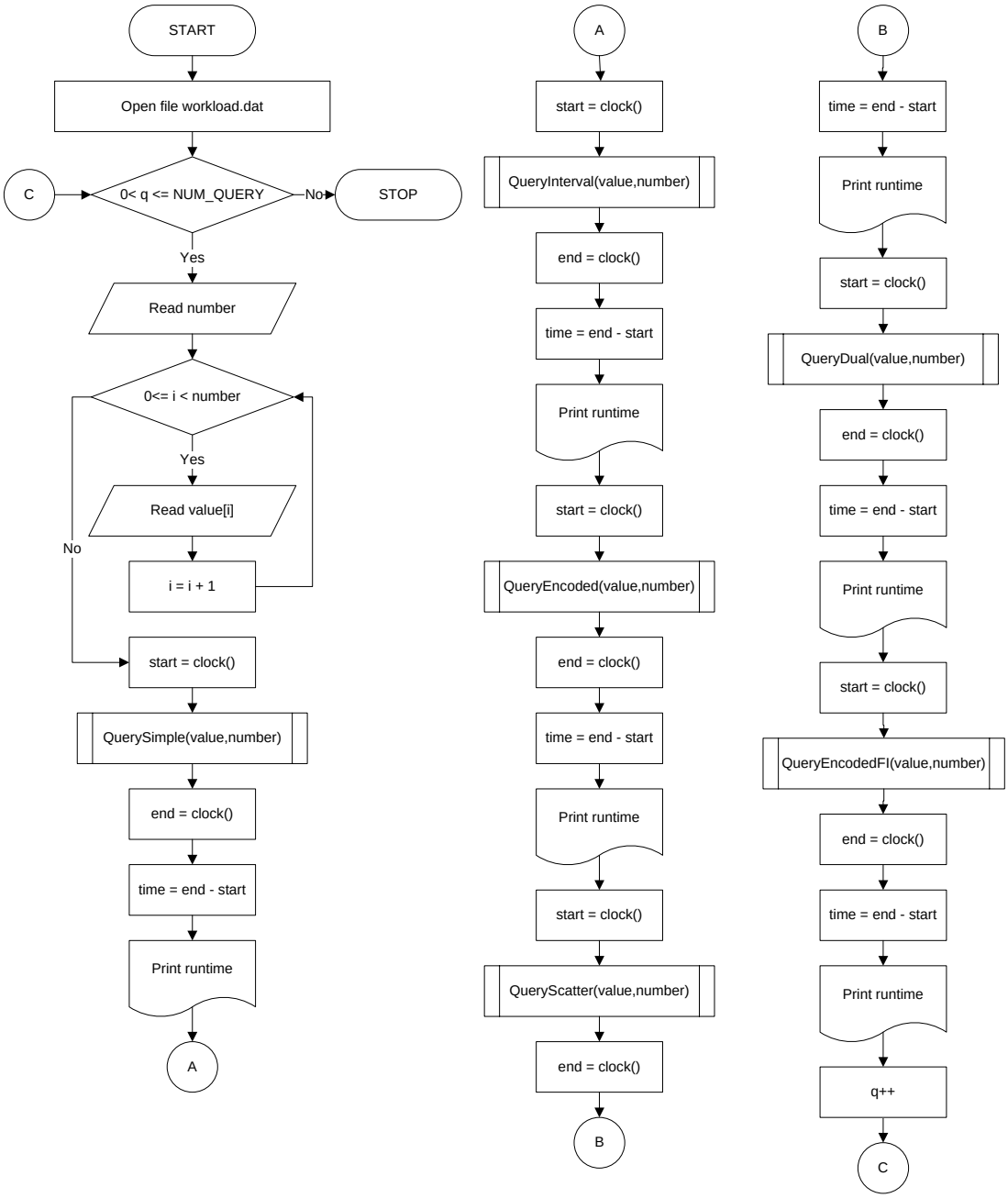
QueryEncodedFl(value,number) ฟังก์ชันดำเนินการสอบถามบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

WordPerVector(NUM_RECORD) ฟังก์ชันดำเนินการหาจำนวนเวิร์ด (Word) ของหนึ่งบิตแมปเวกเตอร์ ซึ่งในที่นี้ให้ 1 เวิร์ดมีค่าเท่ากับ 32 (เท่ากับขนาดของ Integer ในตัวแปลภาษาซีที่ใช้

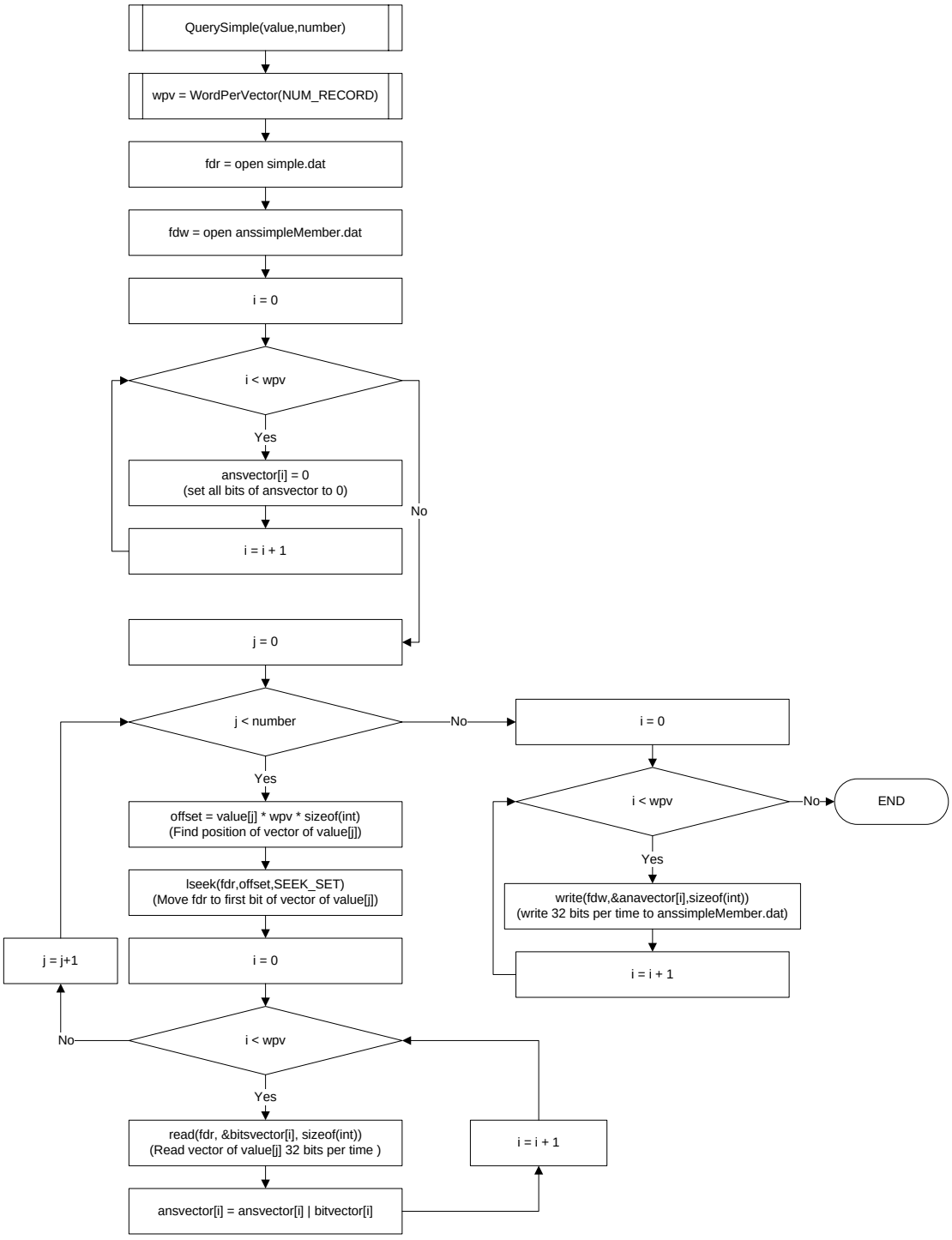
- **ไฟล์ (file)**

simple.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบพื้นฐาน
interval.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบช่วง
encoded.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบเข้ารหัส
scatter.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบกระจาย
dual.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบคู่กัน
encodedFI.dat	เป็นไฟล์ที่เก็บดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย
anssimpleMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบพื้นฐาน
ansintervalMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบช่วง
ansencodedMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบเข้ารหัส
ansscatterMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบกระจาย
ansdualMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบคู่กัน
ansencodedFIMember.dat	เป็นไฟล์ที่เก็บคำตอบของการสอบถามบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

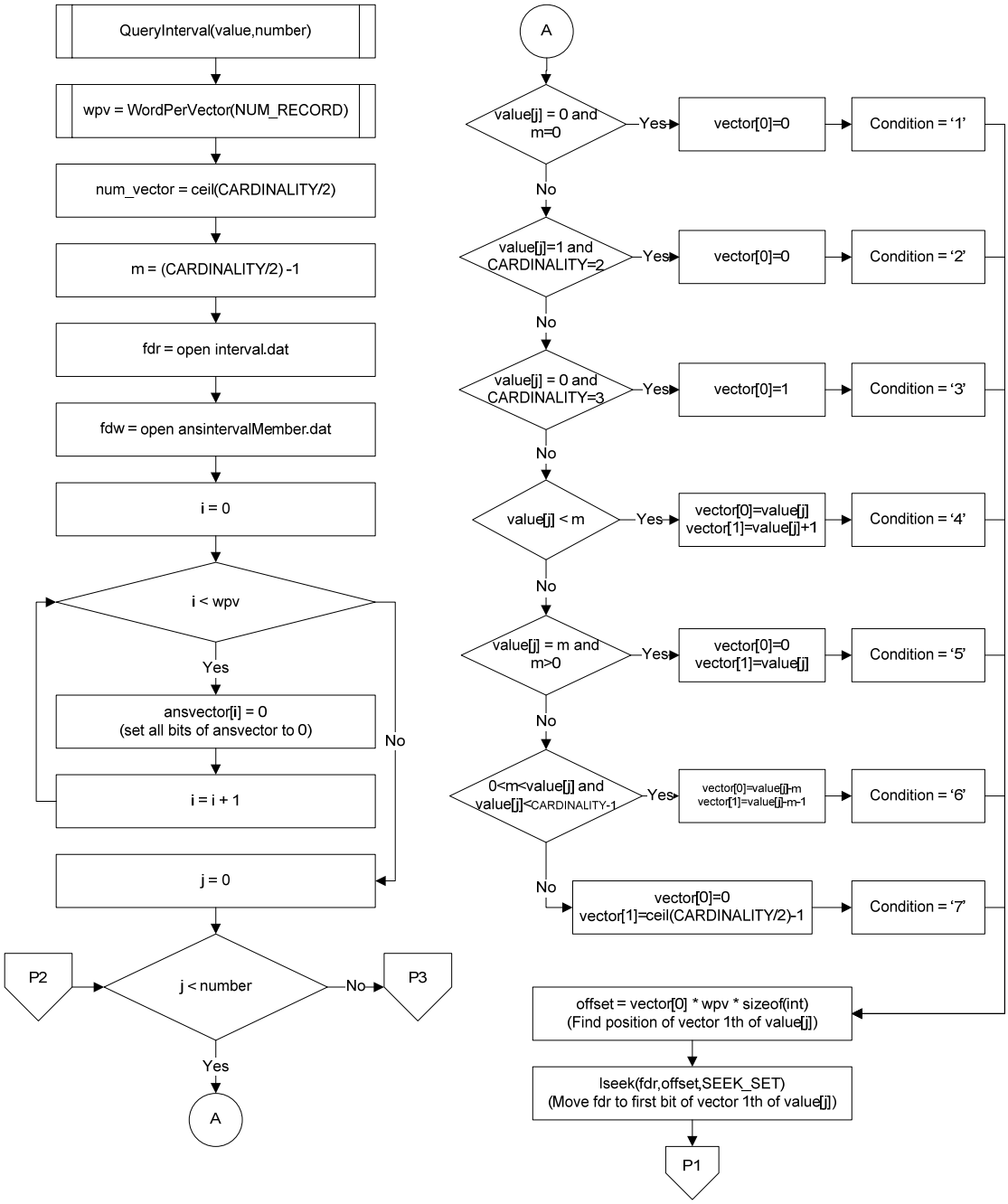
ก.2.1 ขั้นตอนวิธีทำงานหลักของการสอบถามแบบสมาชิกบนดัชนีบิตแมปทั้ง 6 ชนิด

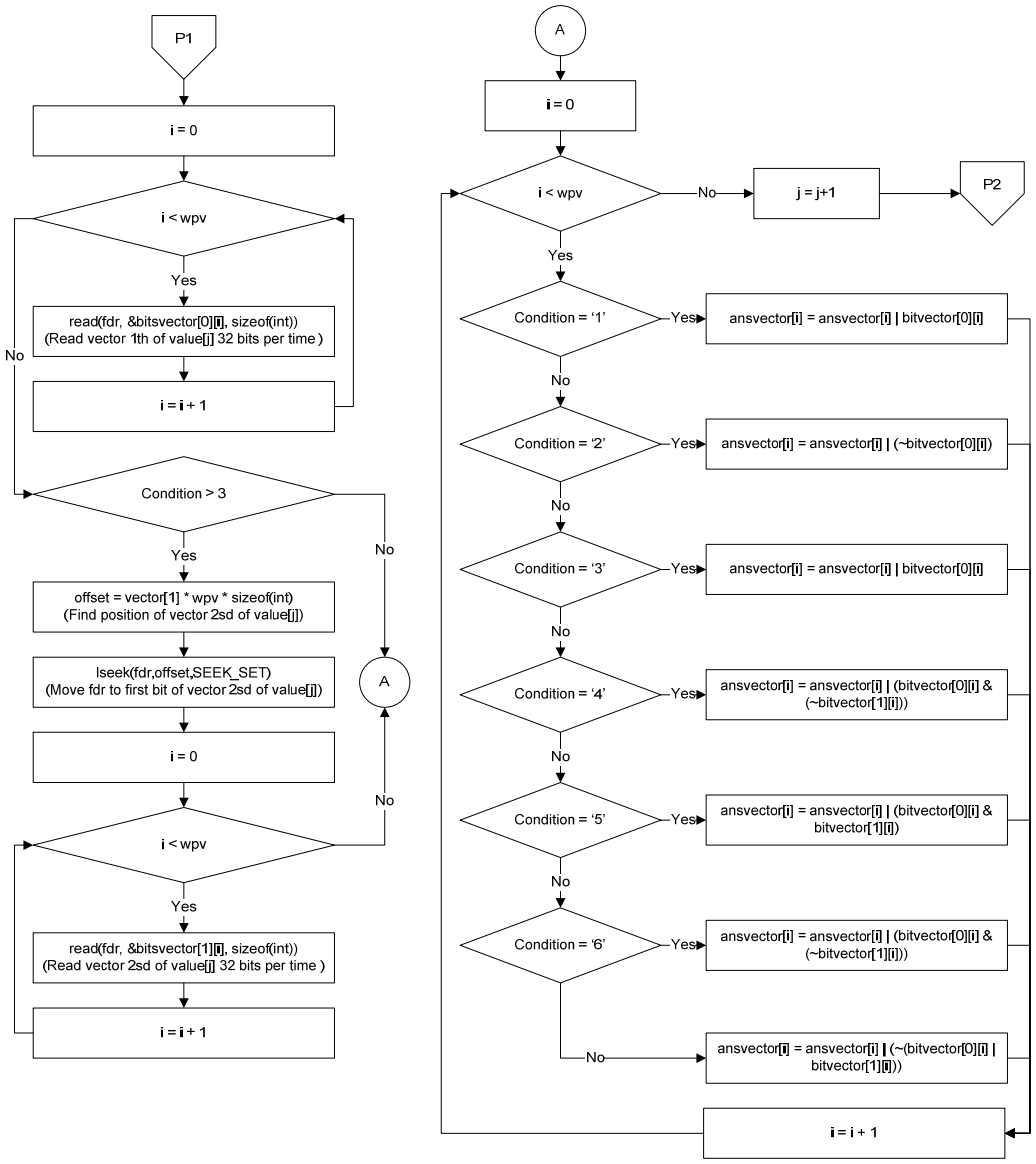


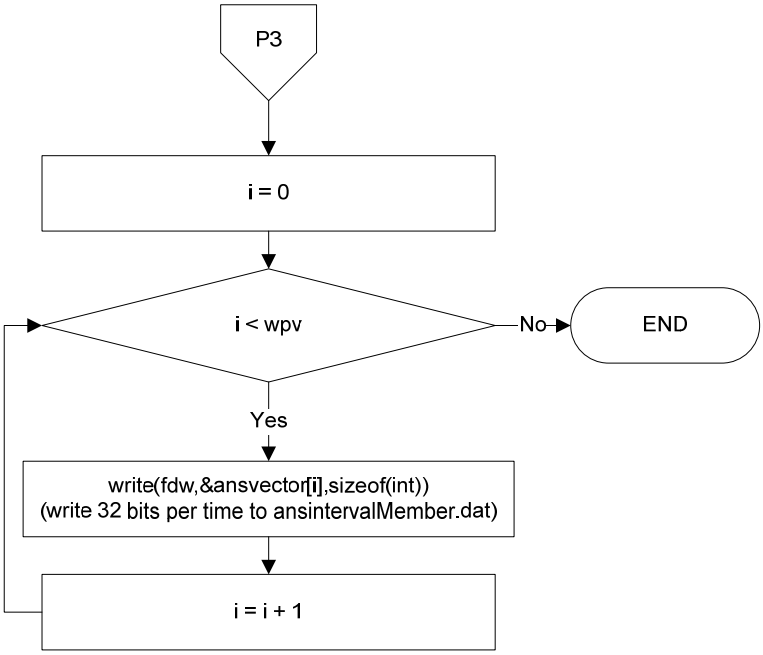
ก.2.2 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบพื้นฐาน



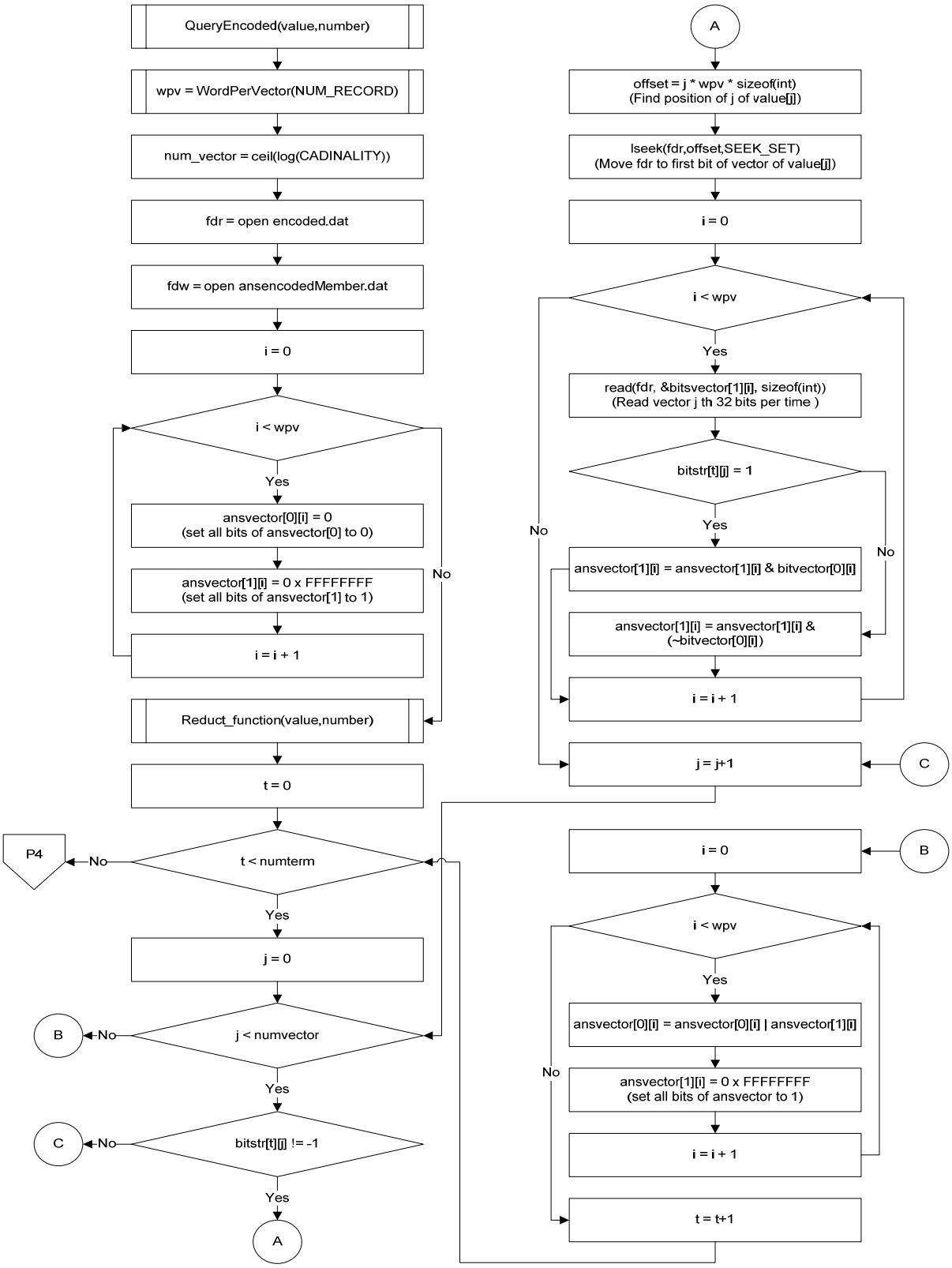
ก.2.3 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบช่วง

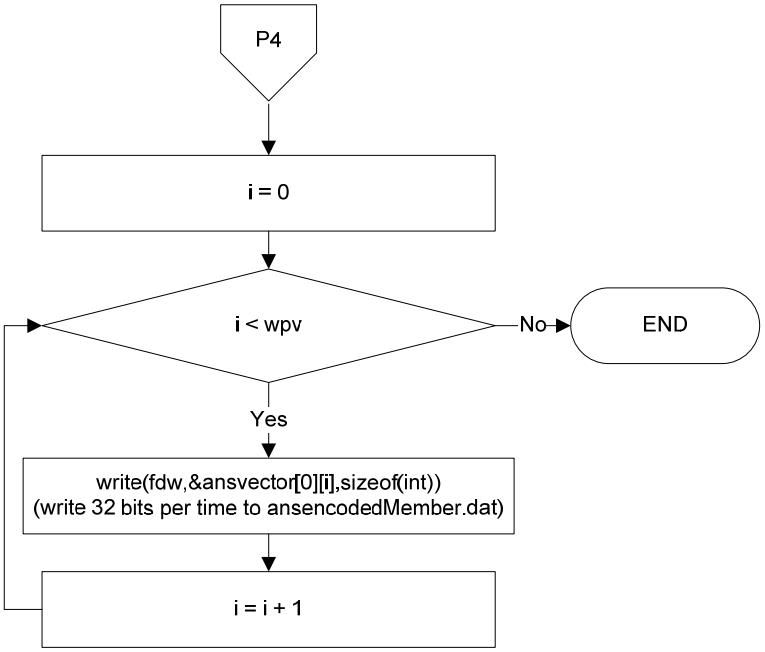



P1
i = 0
i < wpv
Yes
No
read(fdr, &bitsvector[0][i], sizeof(int))
(Read vector 1th of value[j] 32 bits per time)
i = i + 1
Condition > 3
Yes
No
offset = vector[1] * wpv * sizeof(int)
(Find position of vector 2sd of value[j])
lseek(fdr,offset,SEEK_SET)
(Move fdr to first bit of vector 2sd of value[j])
i = 0
i < wpv
Yes
No
read(fdr, &bitsvector[1][i], sizeof(int))
(Read vector 2sd of value[j] 32 bits per time)
i = i + 1
A
A
i = 0
i < wpv
No
j = j+1
P2
Yes
Condition = '1'
Yes
ansvector[i] = ansvector[i] | bitvector[0][i]
No
Condition = '2'
Yes
ansvector[i] = ansvector[i] | (~bitvector[0][i])
No
Condition = '3'
Yes
ansvector[i] = ansvector[i] | bitvector[0][i]
No
Condition = '4'
Yes
ansvector[i] = ansvector[i] | (bitvector[0][i] & (~bitvector[1][i]))
No
Condition = '5'
Yes
ansvector[i] = ansvector[i] | (bitvector[0][i] & bitvector[1][i])
No
Condition = '6'
Yes
ansvector[i] = ansvector[i] | (bitvector[0][i] & (~bitvector[1][i]))
No
ansvector[i] = ansvector[i] | (~(bitvector[0][i] | bitvector[1][i]))
i = i + 1

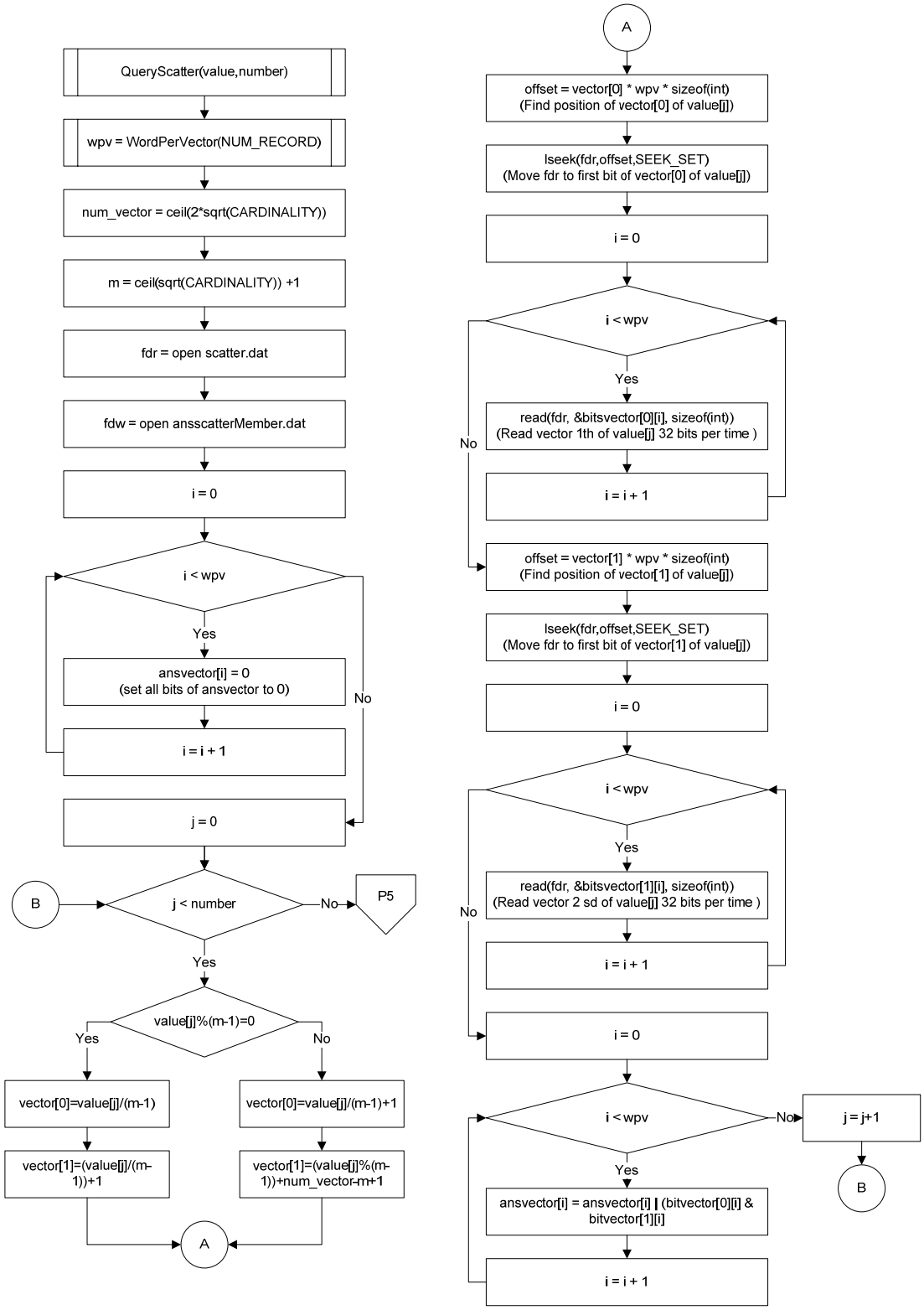

P3
i = 0
i < wpv
No
END
Yes
write(fdw,&ansvector[i],sizeof(int))
(write 32 bits per time to ansintervalMember.dat)
i = i + 1

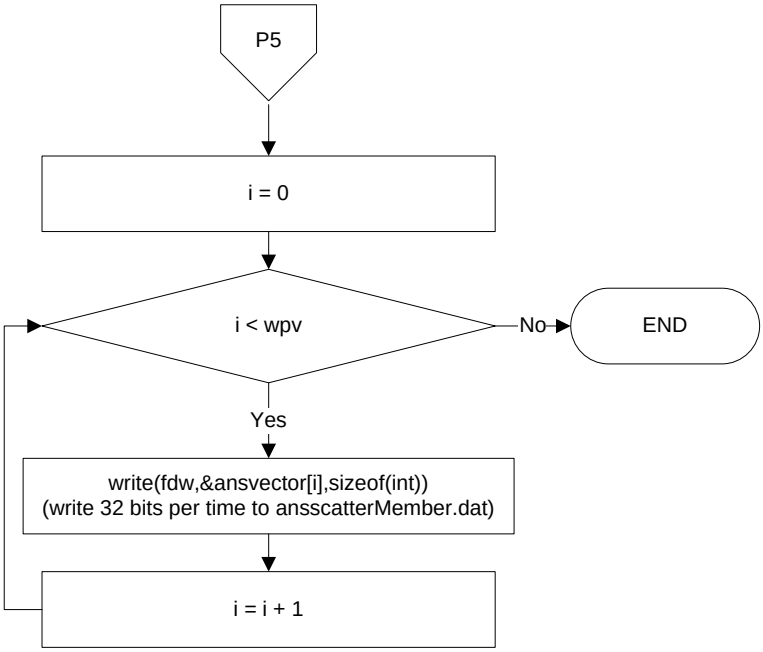
ก.2.4 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบเข้ารหัสทั่วไป



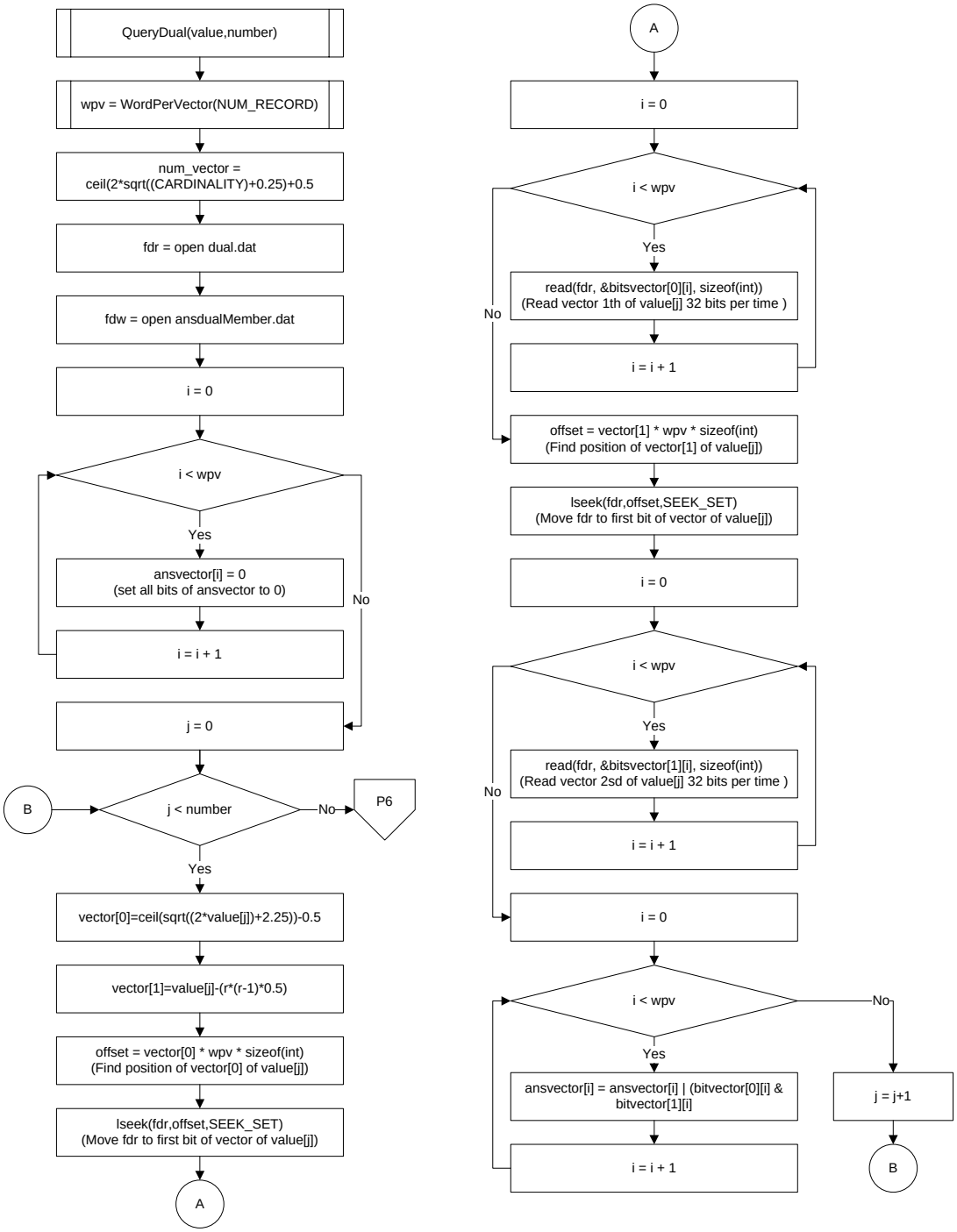

P4
i = 0
i < wpv
No
END
Yes
write(fdw,&ansvector[0][i],sizeof(int))
(write 32 bits per time to ansencodedMember.dat)
i = i + 1

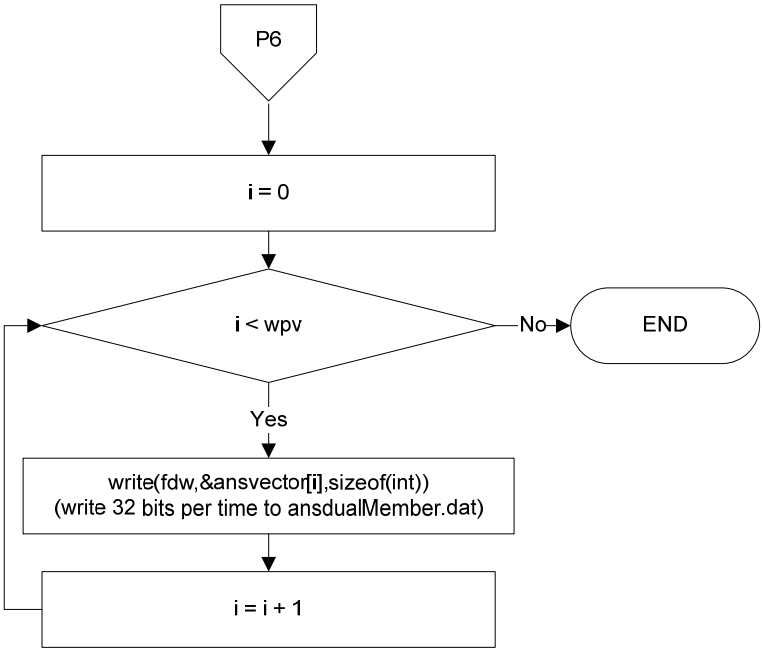
ก.2.5 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบกระจาย



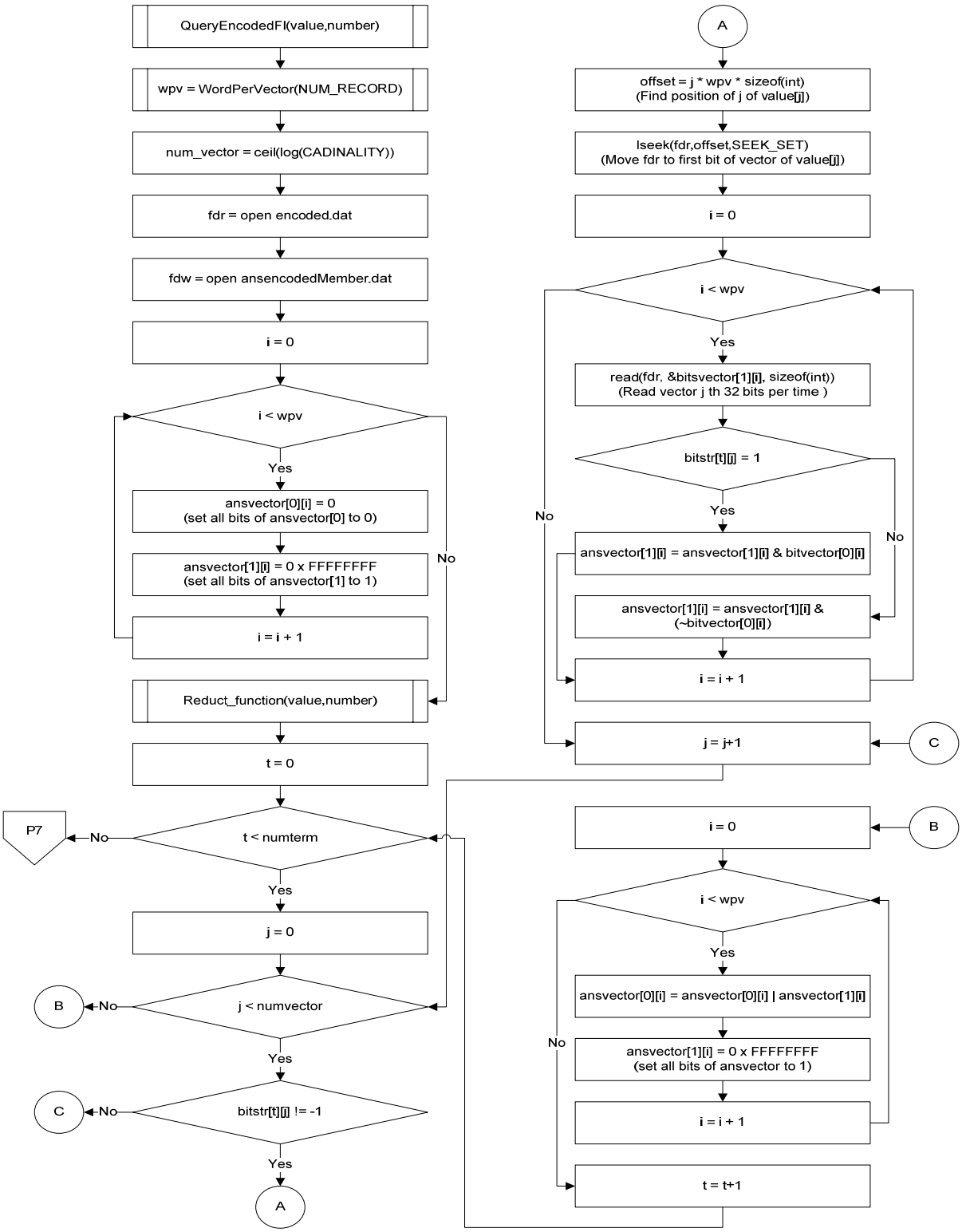

P5
i = 0
i < wpv
No
END
Yes
write(fdw,&ansvector[i],sizeof(int))
(write 32 bits per time to ansscatterMember.dat)
i = i + 1

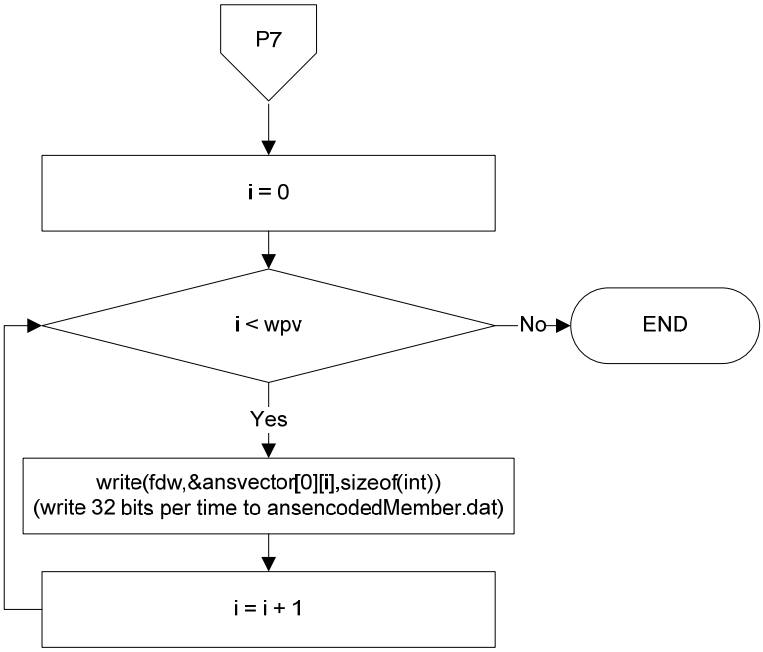
ก.2.6 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบคู่กัน



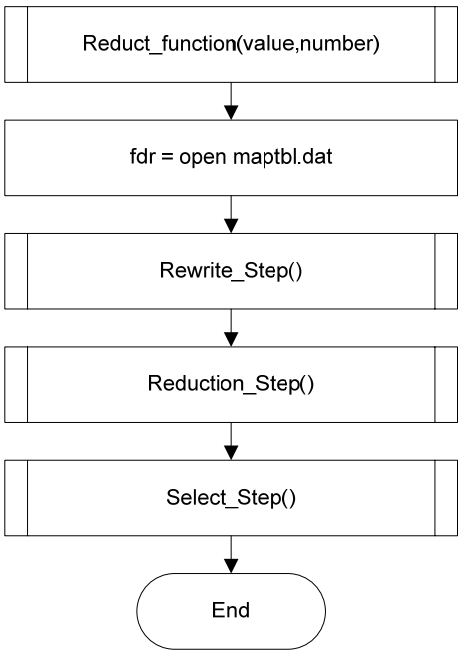

P6
i = 0
i < wpv
No
END
Yes
write(fdw,&ansvector[i],sizeof(int))
(write 32 bits per time to ansdualMember.dat)
i = i + 1

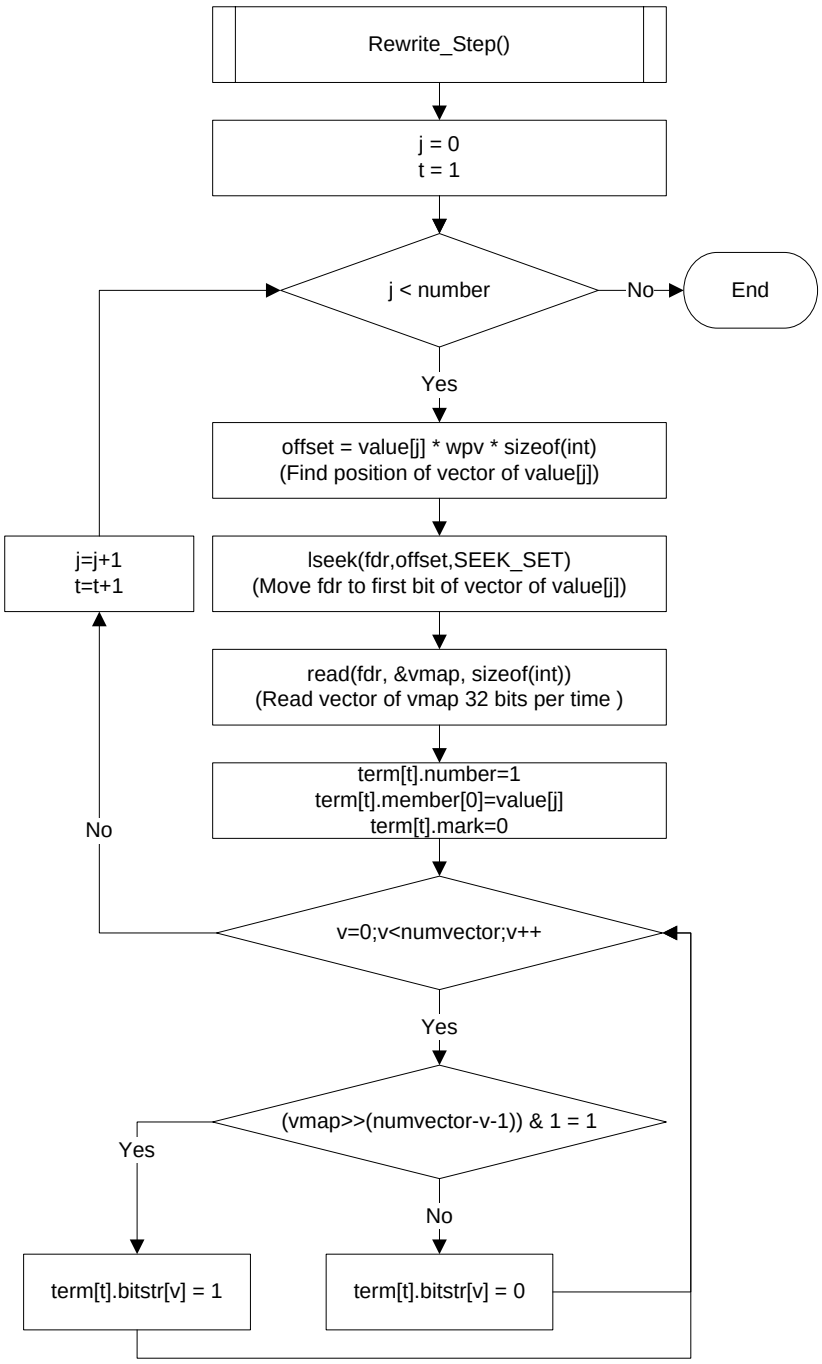
ก.2.7 ขั้นตอนวิธีการสอบถามแบบสมาชิกบนดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย

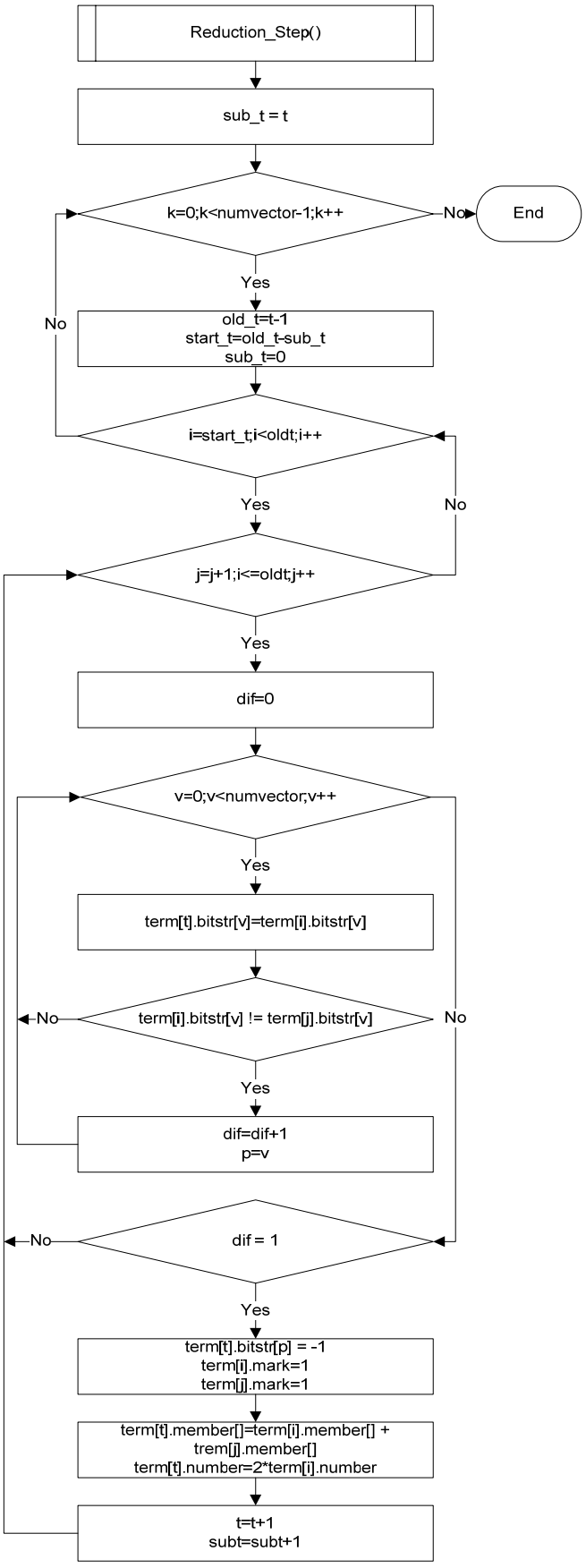


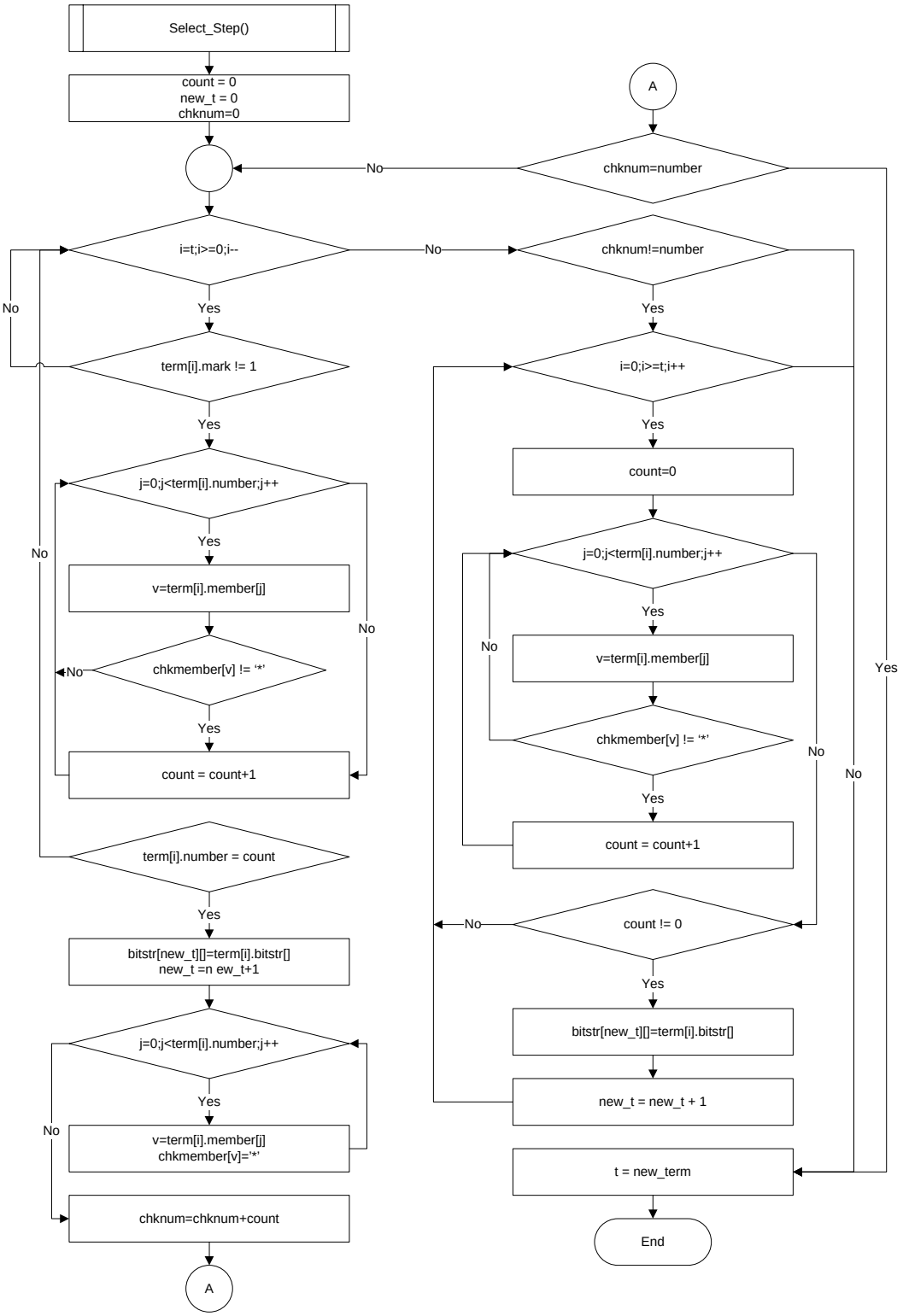

P7
i = 0
i < wpv
No
END
Yes
write(fdw,&ansvector[0][i],sizeof(int))
(write 32 bits per time to ansencodedMember.dat)
i = i + 1

ก.2.8 ขั้นตอนวิธีการลดรูปฟังก์ชันการเข้าถึงข้อมูลบนดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กลุ่มข้อมูลที่ปรากฏบ่อย




Rewrite_Step()
j = 0
t = 1
j < number
No
End
Yes
offset = value[j] * wpv * sizeof(int)
(Find position of vector of value[j])
j=j+1
t=t+1
lseek(fdr,offset,SEEK_SET)
(Move fdr to first bit of vector of value[j])
read(fdr, &vmap, sizeof(int))
(Read vector of vmap 32 bits per time)
term[t].number=1
term[t].member[0]=value[j]
term[t].mark=0
No
v=0;v<numvector;v++
Yes
(vmap>>(numvector-v-1)) & 1 = 1
Yes
No
term[t].bitstr[v] = 1
term[t].bitstr[v] = 0


Reduction_Step()
sub_t = t
k=0;k<numvector-1;k++
No
End
Yes
No
old_t=t-1
start_t=old_t-sub_t
sub_t=0
i=start_t;i<oldt;i++
Yes
No
j=j+1;i<=oldt;j++
Yes
dif=0
v=0;v<numvector;v++
Yes
term[t].bitstr[v]=term[i].bitstr[v]
No
term[i].bitstr[v] != term[j].bitstr[v]
No
Yes
dif=dif+1
p=v
No
dif = 1
Yes
term[t].bitstr[p] = -1
term[i].mark=1
term[j].mark=1
term[t].member[]=term[i].member[] +
trem[j].member[]
term[t].number=2*term[i].number
t=t+1
subt=subt+1


Select_Step()
count = 0
new_t = 0
chknum=0
i=t;i>=0;i--
No
Yes
term[i].mark != 1
Yes
No
j=0;j<term[i].number;j++
Yes
No
v=term[i].member[j]
chkmember[v] != '*'
No
Yes
count = count+1
No
term[i].number = count
Yes
bitstr[new_t][]=term[i].bitstr[]
new_t =n ew_t+1
j=0;j<term[i].number;j++
Yes
No
v=term[i].member[j]
chkmember[v]='*'
chknum=chknum+count
A
A
chknum=number
No
Yes
chknum!=number
Yes
i=0;i>=t;i++
Yes
count=0
j=0;j<term[i].number;j++
Yes
No
v=term[i].member[j]
chkmember[v] != '*'
No
Yes
count = count+1
No
count != 0
Yes
bitstr[new_t][]=term[i].bitstr[]
new_t = new_t + 1
t = new_term
End
Yes
No

ภาคผนวก ข

ผลงานวิจัยที่ได้รับการตีพิมพ์

เรื่อง	การใช้กฎความสัมพันธ์เพื่อเพิ่มประสิทธิภาพของการทำดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิก
งานประชุมวิชาการ	The 5th Joint Conference on Computer Science and Software Engineering (JCSSE 2008)
สถานที่	จังหวัดกาญจนบุรี ประเทศไทย
วันที่	ระหว่างวันที่ 7-9 พฤษภาคม 2551

การใช้กฎความสัมพันธ์เพื่อเพิ่มประสิทธิภาพของการทำดัชนีบิตแมปแบบเข้ารหัส สำหรับการสอบถามแบบสมาชิก

Using Association Rules to Optimize Encoded Bitmap Index for Membership Queries

จรรยา สายนุ้ย, นิวรรณ วัฒนกิจรุ่งโรจน์ และ ศิริรัตน์ วณิชโยบล

*i*STAR LAB ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์

อ.หาดใหญ่ จ.สงขลา 90112

Email: s5010220020@psu.ac.th

บทคัดย่อ

ในระบบคลังข้อมูล การทำดัชนีแบบบิตแมปเป็นเทคนิคหนึ่งที่จะช่วยให้การสอบถามข้อมูลมีประสิทธิภาพ โดยไม่ต้องเสียค่าใช้จ่ายในการเพิ่มฮาร์ดแวร์ใด ๆ ทั้งสิ้น ซึ่งจะช่วยให้การประมวลผลมีความรวดเร็วมากยิ่งขึ้น เนื่องจากสามารถดำเนินการระดับบิตระหว่างบิตแมปเวกเตอร์ก่อนดึงข้อมูลจริง เพื่อเพิ่มประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัส ในงานวิจัยนี้จึงนำเอากฎความสัมพันธ์ซึ่งเป็นเทคนิคหนึ่งในการทำเหมืองข้อมูลมาช่วยในการเข้ารหัสเพื่อให้ได้การเข้ารหัสที่ดีและนำไปสู่การเพิ่มประสิทธิภาพในการสอบถามข้อมูล จากการเปรียบเทียบประสิทธิภาพของเทคนิคการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัส กับเทคนิคการทำดัชนีบิตแมปแบบอื่นที่เคยมีมา เราพบว่าเทคนิคการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์มีประสิทธิภาพในการใช้พื้นที่และเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิกดีกว่า

คำสำคัญ : bitmap index, association rules, frequent itemsets

Abstract

Indexing techniques based on bitmap representations are well suited to a warehouse system. They significantly improve query processing time by utilizing low-cost Boolean operations and multiple index scans, executing queries by performing simple predicate conditions on the index level before going to the primary data source. To optimize existing Encoded Bitmap Index, in this paper, we apply a data mining technique called association rule to find a well-defined encoding scheme, leading to improve query processing time. Our comparative study show that in the best case the performance of optimizing Encoded Bitmap Index using Association Rules is better than those found by existing techniques for membership queries from the point of view of space-time trade-off.

1. บทนำ

ในระบบคลังข้อมูลการสอบถามข้อมูลหรือสารสนเทศจะเป็นแบบ Online Analytical Processing (OLAP) [11] ซึ่งจะมีความซับซ้อน (Complex query) และเป็นแบบทันทีทันใด (Ad hoc) [4] กล่าวคือจะไม่ทราบล่วงหน้าว่าผู้ใช้จะสอบถามอะไรบ้าง โดยจะเป็นการสอบถามข้อมูลเพื่อช่วยในการตัดสินใจของผู้บริหาร ซึ่งข้อมูลที่เก็บในคลังข้อมูลเป็นข้อมูลที่มาจากหลาย ๆ แหล่ง และเป็นข้อมูลที่เก็บตั้งแต่อดีตจนถึงปัจจุบัน ข้อมูลที่อยู่ในคลังข้อมูลจึงมีจำนวนมาก ทำให้การค้นหาข้อมูลต้องใช้เวลามากขึ้น

การเพิ่มความเร็วในการค้นหาข้อมูลมีหลายวิธี เช่น การสร้าง Summary table การประมวลผลแบบคู่ขนาน และการทำดัชนี [2,3,7,8,9,11] ซึ่งการทำดัชนีเป็นวิธีที่ไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ โดยการทำดัชนีก็มีหลายแบบ เช่น B+ tree, Hash และ บิตแมป สำหรับระบบคลังข้อมูลนั้นดัชนีที่นิยมใช้คือ ดัชนีแบบบิตแมป (Bitmap indexing) เนื่องจากสามารถดำเนินการระดับบิต (Bit operation) เช่น AND , OR, XOR และ NOT ระหว่างบิตแมปเวกเตอร์ก่อนดึงข้อมูลจริง ทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น [2,3,7,8,9,12,13] นอกจากนี้การทำดัชนีแบบบิตแมปยังมีประสิทธิภาพในเรื่องของการใช้พื้นที่ในการจัดเก็บดัชนี โดยเฉพาะแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำ (คาร์ดินอร์ลิตี้ คือ จำนวนค่าที่แตกต่างกันในคอลัมน์ที่นำมาทำดัชนี) อย่างไรก็ตามมีงานวิจัยที่ได้คิดค้นเทคนิคในการทำดัชนีแบบบิตแมปกับข้อมูลที่มีค่าคาร์ดินอร์ลิตี้สูง เช่น การทำดัชนีบิตแมปแบบช่วง (Interval Bitmap Index) [3], การทำดัชนีบิตแมปแบบกระจาย (Scatter Bitmap Index) [12] และการทำดัชนีบิตแมปแบบคู่กัน (Dual Bitmap Index) [13] ซึ่งนอกจากจะพัฒนาประสิทธิภาพของพื้นที่แล้วยังพัฒนาในเรื่องของเวลาที่ใช้ในการสอบถามข้อมูลอีกด้วย และในงานวิจัย [8] ได้แนะนำการทำดัชนีบิตแมปแบบเข้ารหัส ซึ่งเป็นการทำดัชนีบิตแมปที่มีประสิทธิภาพในการใช้พื้นที่ในการจัดเก็บดัชนีสูงที่สุด (ใช้ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) สามารถทำดัชนีกับข้อมูลที่มีค่าคาร์ดินอร์ลิตี้สูงได้ แต่ในการสอบถามข้อมูลแบบค่าเท่ากันนั้นจะต้องมีการเปรียบเทียบกับทุกบิตของบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) ว่าตรงกันหรือไม่ เพราะฉะนั้นหากเป็นการสอบถามข้อมูลแบบสมาชิกก็จะต้องตรวจสอบทุกบิตสำหรับแต่ละค่าที่ต้องการสอบถามแล้วนำผลลัพธ์ที่ได้มาดำเนินการตรรกะ OR ทำให้ใช้เวลาในการสอบถามข้อมูลเพิ่มขึ้น ดังนั้นหากเราสามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามข้อมูลได้ โดยไม่ต้องตรวจสอบทุกบิตแมปเวกเตอร์ในแต่ละค่าของสมาชิกก็จะทำให้เวลาที่ใช้ในการสอบถามน้อยลง นั่นคือจะทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น

เนื่องจากหัวใจสำคัญในการทำดัชนีบิตแมปแบบเข้ารหัสคือการเข้ารหัส ซึ่งการเข้ารหัสที่ดีจะทำให้ประสิทธิภาพในการสอบถามข้อมูลสูงขึ้นด้วย (ใช้เวลาสอบถามน้อยลง) โดยการเข้ารหัสที่ดีในที่นี้หมายถึง การเข้ารหัสที่สามารถจะลดจำนวนบิตที่ใช้ในการสอบถามข้อมูลให้เหลือน้อยที่สุด ในงานวิจัยนี้จึงนำเอากฎความสัมพันธ์ซึ่งเป็นเทคนิคหนึ่งในการทำเหมืองข้อมูลมาช่วยในการหาความสัมพันธ์ของข้อมูลและจัดกลุ่มข้อมูลที่จะเข้ารหัส โดยเข้ารหัสกลุ่มข้อมูลที่ถูกสอบถามด้วยกันบ่อย ๆ ให้สามารถลดจำนวนบิตลงได้เมื่อมีการสอบถามข้อมูลด้วยกัน

เอกสารนี้ประกอบด้วย 6 ส่วน โดยส่วนที่ 2 จะกล่าวถึงดัชนีบิตแมปแบบเข้ารหัส ส่วนที่ 3 จะเป็นการพูดถึงกฎความสัมพันธ์ ส่วนที่ 4 เป็นการนำเสนอการเพิ่มประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกของการทำดัชนีบิตแมปแบบเข้ารหัสโดยใช้กฎความสัมพันธ์ ส่วนที่ 5 เป็นการวิเคราะห์ประสิทธิภาพของดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัส และ ส่วนที่ 6 จะเป็นบทสรุปและงานที่จะทำต่อในอนาคต

2. ดัชนีบิตแมปแบบเข้ารหัส (Encoded Bitmap Index)

Wu และ Buchmann ได้เสนอเทคนิคการทำดัชนีบิตแมปแบบเข้ารหัส [8] ซึ่งในการแทนค่าของแอทริบิวต์ที่เลือกมาทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ที่มีค่าคาร์ดินอร์-ลิตี้เท่ากับ C ประกอบด้วย $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์คือ $E_{\lceil \log_2 C \rceil -1}$,..., E_2, E_1, E_0 และตารางเทียบค่า (Mapping Table) โดยที่แต่ละค่าของแอทริบิวต์จะถูกแทนด้วย $E_{\lceil \log_2 C \rceil -1} \ldots E_2 E_1 E_0$ เมื่อ $E_i \in \{B_i, B_i^{'}\}$ โดยให้ $B_i = 1, B_i^{'} = 0$, $i = 0,1,2,\ldots, \lceil \log_2 C \rceil -1$

ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสดังรูปที่ 1 ซึ่งเป็นการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T โดยแอทริบิวต์ X ประกอบด้วยสมาชิก (Domain) คือ {A,B,C,D,E,F,G,H} จะเห็นได้ว่าคาร์ดินอร์ลิตี้เท่ากับ 8 จึงมีการใช้ 3 บิตแมปเวกเตอร์ ($\lceil \log_2 8 \rceil = 3$) ในการแทนค่าแต่ละค่า เช่น จากตารางการเทียบค่า A แทนด้วย $B_2^{'} B_1^{'} B_0^{'}$ (000) และในการสอบถามข้อมูลแบบค่าเท่ากันจะต้องตรวจสอบค่าของแอทริบิวต์ที่ต้องการสอบถามกับตารางการเทียบค่าก่อนว่าแต่ละบิตแมปเวกเตอร์เก็บค่าใด แล้วจึงนำไปตรวจสอบกับ

Table T

RID	…	X	…
1		B	
2		C	
3		A	
4		H	
5		A	
6		G	
7		D	
8		D	
9		F	
.		.	
.		.	
.		.	
10000		E	

a. (ตาราง T)

E_2	E_1	E_0
0	0	1
0	1	0
0	0	0
1	1	1
0	0	0
1	1	0
0	1	1
0	1	1
1	0	1
.	.	.
.	.	.
.	.	.
1	0	0

b. (ดัชนีบิตแมปแบบเข้ารหัสและตารางเทียบค่า)

Mapping table

A	000
B	001
C	010
D	011
E	100
F	101
G	110
H	111

รูปที่ 1 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T

ดัชนีบิตแมปแบบเข้ารหัสที่สร้างไว้ว่าตรงกับบิตใด ก็จะได้แถวนั้นมีค่าของแอทริบิวต์ตรงกับที่ต้องการ ตัวอย่างเช่น ต้องการสอบถาม X = A จากตาราง T รูปที่ 1 จะต้องตรวจสอบค่า A จากตารางเทียบค่าจะได้ว่าค่า A แทนด้วย $B_2'B_1'B_0'$ คือ 000 จากนั้นจึงนำไปตรวจสอบกับดัชนีบิตแมปแบบเข้ารหัสที่สร้างไว้ จะได้ว่าแถวที่ 3 และ 5 มีบิตแมปทั้ง 3 เป็น 0 ดังนั้นจะได้แถวที่ 3 และ 5 เป็นคำตอบของการสอบถามนี้

สำหรับการสอบถามข้อมูลแบบสมาชิกทำได้โดยหาคำตอบของแต่ละสมาชิก แล้วนำผลลัพธ์ที่ได้มาดำเนินการตรรกะ OR (ในที่นี้ใช้สัญลักษณ์ + แทน OR) พิจารณาตัวอย่าง เมื่อต้องการสอบถามข้อมูลแบบสมาชิก “X in {A,D,F,G}” โดยใช้ตารางการเทียบค่าจากรูปที่ 1 จะได้ฟังก์ชันในการสอบถามข้อมูลเป็น $B_2'B_1'B_0' + B_2'B_1B_0 + B_2B_1'B_0 + B_2B_1B_0'$ จะเห็นว่าเราต้องตรวจสอบทุกบิตแมปเวกเตอร์ของแต่ละค่าจึงจะได้คำตอบของการสอบถาม อย่างไรก็ตาม ถ้าเรามีตารางการเทียบค่าแบบในรูปที่ 2 หากเปรียบเทียบกับการสอบถามที่เหมือนกันจะได้ฟังก์ชันในการสอบถามข้อมูล “X in {A,D,F,G}” เป็น $B_2'B_1'B_0' + B_2'B_1'B_0 + B_2'B_1B_0 + B_2'B_1B_0'$ แต่สามารถลดจำนวนบิตให้เหลือเพียง 1 บิตแมปเวกเตอร์ได้ โดยสามารถลดรูปฟังก์ชันการสอบถามข้อมูล[8] ได้เป็น $B_2'(B_1'(B_0' + B_0) + B_1(B_0 + B_0')) = B_2'$ เพราะฉะนั้นตรวจสอบเพียงบิตแมปเวกเตอร์ B_2' ก็สามารถได้คำตอบของการสอบถาม

จากที่กล่าวมาจะเห็นว่า หากเราสามารถเข้ารหัสกลุ่มสมาชิกที่มีการสอบถามด้วยกันบ่อย ๆ ให้อยู่ในรูปแบบที่สามารถลดจำนวนบิตที่ใช้ในการสอบถามข้อมูลได้ ก็จะเป็นการเพิ่มประสิทธิภาพในการสอบถามข้อมูล กล่าวคือสามารถลดจำนวนเวลาในการค้นหาข้อมูลได้มาก ซึ่งในการหากลุ่มข้อมูลที่มีการสอบถามด้วยกันบ่อย ๆ นั้น เราสามารถนำเอากฎความสัมพันธ์ (Association rule) ซึ่งเป็นเทคนิคหนึ่งในการทำเหมืองข้อมูลมาช่วยในการจัดกลุ่มสมาชิกก่อนสร้างตารางเทียบค่าที่นำไปสู่การลดรูปบิตแมปเวกเตอร์เมื่อมีการสอบถามในการทำดัชนีบิตแมปแบบเข้ารหัส

3. กฎความสัมพันธ์ (Association Rule)

กฎความสัมพันธ์[10] คือ การหารูปแบบความสัมพันธ์ระหว่างกลุ่มของรายการหรือสิ่งของ (Items) ในฐานข้อมูลรายการทรานแซคชัน ซึ่งอยู่ภายใต้ค่าสนับสนุน (Support) และค่าความเชื่อมั่น (Confidence)

กระบวนการหากฎความสัมพันธ์ ประกอบด้วย 2 ขั้นตอนคือ[6]

1. การหากลุ่มข้อมูลที่ถูกสอบถามด้วยกันบ่อย ๆ (Frequent itemsets) ที่มีค่าความถี่หรือค่าสนับสนุนมากกว่าหรือเท่ากับค่าความถี่ขั้นต่ำ (Minimum Support)
2. การสร้างกฎความสัมพันธ์ จาก Frequent itemsets ที่หาได้จากขั้นตอนที่ 1 และจะยอมรับกฎความสัมพันธ์ที่สร้างขึ้น ก็ต่อเมื่อกฎนั้นมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ (Minimum Confidence)

Table T

RID	…	X	…
1		B	
2		C	
3		A	
4		H	
5		A	
6		G	
7		D	
8		D	
9		F	
.		.	
.		.	
.		.	
10000		E	

a. (ตาราง T)

E_2	E_1	E_0
1	0	0
1	0	1
0	0	0
1	1	0
0	0	0
0	1	0
0	0	1
0	0	1
0	1	1
.	.	.
.	.	.
.	.	.
1	1	1

Mapping table

A	000
B	100
C	101
D	001
E	111
F	011
G	010
H	110

b. (ดัชนีบิตแมปแบบเข้ารหัสและตารางเทียบค่า)

รูปที่ 2 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T โดยใช้กฎความสัมพันธ์

ในงานวิจัยนี้จะมุ่งเน้นไปที่ขั้นตอนที่ 1 คือการหากลุ่มสมาชิก (Itemsets) ที่ถูกสอบถามด้วยกันบ่อย ๆ เพื่อช่วยในการจัดกลุ่มข้อมูลของแอทริบิวต์ที่เราสนใจที่จะทำดัชนีบิตแมปแบบเข้ารหัส ตัวอย่างเช่น จากตัวอย่างการสอบถามข้อมูลแบบสมาชิกข้างต้น เมื่อทำผ่านขั้นตอนที่ 1 เราจะได้กลุ่มสมาชิกที่ถูกสอบถามด้วยกันบ่อย คือ {A,D,F,G}

4. การเพิ่มประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัส(Optimizing Encoded Bitmap Index)

กระบวนการเพิ่มประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกของการทำดัชนีบิตแมปแบบเข้ารหัสประกอบด้วย 2 ขั้นตอนคือ ขั้นตอนที่ 1 คือการหา Frequent Itemsets เป็นการนำกฎความสัมพันธ์มาช่วยในการจัดกลุ่มสมาชิก และขั้นตอนที่ 2 คือการเข้ารหัส [14]

4.1 การหา Frequent Itemsets

ในส่วนนี้จะเป็นการกล่าวถึงการใช้กฎความสัมพันธ์เพื่อหา Frequent itemsets จากฐานข้อมูลทรานแซคชัน โดยผู้วิจัยได้ทำการศึกษาอัลกอริทึม Apriori [1] และอัลกอริทึม FP-growth [6] และนำเสนอการหา Frequent itemsets สำหรับการทำดัชนีบิตแมปแบบเข้ารหัสที่เหมาะสมกับงานวิจัยนี้

4.1.1 อัลกอริทึม Apriori

อัลกอริทึม Apriori เป็นอัลกอริทึมในการหารูปแบบความสัมพันธ์ที่ถูกพัฒนาขึ้นในระยะแรก ๆ ซึ่งเป็นอัลกอริทึมพื้นฐานในการศึกษาและพัฒนาอัลกอริทึมอื่น ๆ อีกมากมาย

ขั้นตอนวิธีของ Apriori จะเริ่มต้นที่การหา Frequent 1-itemsets จากการอ่านฐานข้อมูลรอบแรก และหา Frequent itemsets ในระดับถัด ๆ ไปด้วยการสร้าง Candidate itemsets ที่มีความยาว k+1 จากการรวมกันของ Frequent k-temsets และทำการตัด Candidate itemsets ที่ไม่มีโอกาสเป็น Frequent itemsets ออก โดยใช้คุณสมบัติพื้นฐานซึ่งมีหลักการอยู่ว่า “ถ้า Itemset ขนาด k ใด ๆ ไม่เป็น Frequent แล้ว Itemsets ขนาด k+1 ที่เป็น Superset ของ Itemsets นั้นจะไม่เป็น Frequent ด้วย” [1] ซึ่งในการหาค่าความถี่ของแต่ละ Candidate itemsets จะต้องทำการอ่านฐานข้อมูลทุกครั้งเพื่อให้ได้ Frequent ในระดับถัดไป ดังนั้นจะเห็นได้ว่าจะต้องอ่านฐานข้อมูลจำนวน k ครั้ง (เมื่อ k คือขนาดของ Itemsets ที่ใหญ่ที่สุดที่มากกว่าค่า Minimum support) เพื่อตรวจสอบค่า Support ของ Candidat itemsets

4.1.2 อัลกอริทึม FP-growth

อัลกอริทึม FP-growth จะใช้โครงสร้างข้อมูล FP-tree เพื่อใช้ในการหา Frequent patterns โดยไม่ต้องสร้าง Candidate itemsets เหมือนอัลกอริทึม Apriori ที่ต้องสร้าง Candidate itemsets จำนวนมาก และอัลกอริทึม FP-growth อ่านฐานข้อมูลเพียง 2 ครั้ง

การหา Frequent itemsets ด้วยอัลกอริทึม FP-growth ประกอบด้วย 2 ขั้นตอนคือ ขั้นตอนที่หนึ่งจะเป็นการสร้าง FP-tree จากฐานข้อมูลที่กำหนด และขั้นตอนที่สองจะเป็น

การหา Frequent itemsets จาก FP-tree ที่สร้างได้ แม้ว่าอัลกอริทึม FP-growth จะอ่านฐานข้อมูลเพียง 2 ครั้ง แต่จะต้องทำการเรียงลำดับ Itemในแต่ละทรานแซคชันที่อ่านจากฐานข้อมูลเพื่อนำไปสร้าง FP-tree

4.1.3 การหา Frequent itemsets สำหรับการทำดัชนีบิตแมปแบบเข้ารหัส

จากการศึกษาอัลกอริทึมที่ใช้ในการหา Frequent itemsets ตามที่กล่าวมา จะเห็นว่ายังไม่มีประสิทธิภาพสำหรับงานวิจัยนี้ เพราะในการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ เราสนใจเฉพาะ Frequent itemsets ที่มีขนาด 2^i เมื่อ i = 0, 1, …, $\lceil \log_2 C \rceil - 1$ และในการสร้าง Itemsets ขนาด 2^{i+1} ใด ๆ จะถูกสร้างจากการรวมกันของ Frequent itemsets ขนาด 2^i ที่มีค่าสมาชิกไม่เหมือนกัน ดังนั้นเราขอนำเสนออัลกอริทึมในการหา Frequent itemsets สำหรับการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ โดยมีขั้นตอนวิธีดังนี้

1. อ่านฐานข้อมูลฐานแซคชันบนแอทริบิว X ที่สกัดจาก workload (รูปที่ 3) 1 ครั้งเพื่อสร้างตารางเมตริกดังรูปที่ 4a
2. หา Frequent 1-itemset จากตารางเมตริกดังรูปที่ 4a โดยการนับจำนวนบิต(Support) 1 ในแต่ละค่า(Item) และเลือกค่าที่มีค่า Support มากกว่าหรือเท่ากับ Minimum support ที่กำหนด (รูปที่ 4b)
3. for (i = 1; i < = $\lceil \log_2 C \rceil - 1$; i++)
   3.1 สร้าง Candidate itemsets ขนาด 2^i ได้จากการนำ Frequent itemsets ขนาด 2^{i-1} นำมาดำเนินการตรรกะ AND โดยมีเงื่อนไขว่า Itemsets ที่นำมารวมกันจะต้องมีสมาชิกไม่เหมือนกัน แล้วทำการนับ Support ของบิต 1 จากผลลัพธ์ที่ได้
   3.2 เลือก Support > = Minimum support

จากขั้นตอนวิธีที่กล่าวมา จะเห็นว่ามีการอ่านฐานข้อมูลเพียงครั้งเดียว และใช้การดำเนินการ Count และการดำเนินการตรรกะ AND มาช่วยในการหา Frequent itemsets ที่ต้องการ ทำให้ใช้เวลาในการหา Frequent itemsets ที่ต้องการได้น้อยกว่าอัลกอริทึม Apriori และ FP-growth และจากเงื่อนไขในการสร้าง Candidate itemsets ยังทำให้ลดจำนวน Frequent itemsets ที่เราไม่สนใจได้มาก จากตัวอย่างฐานข้อมูลทรานแซคชันดังรูปที่ 3 สามารถสร้างตารางเมตริกได้ดังรูปที่ 4a โดยคอลัมน์เป็นการสอบถามและแถวแสดงค่าที่ปรากฏในแต่ละการสอบถาม ซึ่งจะนำเสนอในรูปของบิตเมื่อค่าของแอทริบิวต์ X ปรากฏอยู่ในการสอบถามใด ให้ค่าของแอทริบิวต์ของการสอบถามนั้นมีค่าเป็น 1 ถ้าไม่ปรากฏก็ให้ค่าเป็น 0 ตัวอย่างเช่น Q1 มีการสอบถามตามเงื่อนไข X IN (A,C,E,G,O,H,J,K,P) ดังนั้น ค่า A,C,E,G,O,H,J,K และ P ของ Q1 ในตารางเมตริกจะแทนค่าด้วย 1 และค่าอื่น ๆ จะเป็น 0 และจากตารางเมตริกรูปที่ 4a จะได้ Frequent itemsets ที่หาได้ตามขั้นตอนวิธีนี้ดังรูปที่ 5

4.2 การเข้ารหัส

จากแอทริบิวต์ที่เราสนใจจะนำมาทำดัชนีบิตแมปแบบเข้ารหัส เราได้ใช้เทคนิคการทำเหมืองข้อมูลคือการหาความสัมพันธ์ของข้อมูลเพื่อหากลุ่มสมาชิกที่ถูกสอบถามด้วยกันบ่อย ๆ จากกลุ่มสมาชิกที่หาได้ดังรูปที่ 5 เราสามารถนำมาเข้ารหัสที่เพิ่มประสิทธิภาพในการสอบถามข้อมูลดังรูปที่ 6b

รูปที่ 6a เป็นการเข้ารหัสที่ไม่ได้คำนึงถึงข้อมูลที่ถูกสอบถามด้วยกันบ่อย ๆ (ไม่มีการใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัส) ส่วนรูปที่ 6b เป็นการเข้ารหัสที่คำนึงถึงข้อมูลที่ถูกสอบถามด้วยกันบ่อย ๆ(ใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัส) โดยมีขั้นตอนวิธีการเข้ารหัสดังต่อไปนี้

กำหนดให้ C คือ ค่าคาร์ดินอร์ลิตี้ของแอทริบิวต์ที่นำมาสร้างดัชนี

1. เรียงลำดับ Frequent Itemsets ที่หาได้ตามจำนวนสมาชิกและค่า Support จากมากไปน้อย
2. เลือก Frequent Itemsets ที่มีสมาชิกเท่ากับ $\frac{C}{2^i}$ เมื่อ i = 1, 2, …, $\lceil \log_2 C \rceil - 1$ ตามลำดับ ที่มีสมาชิกไม่เหมือนกันมาเรียงเป็นลำดับที่ต่อเนื่องกัน
3. นำข้อมูลที่เหลือมาเรียงลำดับต่อจากข้อมูลที่เรียงอยู่ และเริ่มเข้ารหัสเรียงตามลำดับตั้งแต่ข้อมูลตัวแรกถึงตัวสุดท้าย โดยกำหนดให้รูปแบบของการลงรหัสเริ่มจากค่า 0 จนถึงค่า $C-1$ ในรูปของเลขฐานสอง

Q1 : SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K,P)

Q2 : SELECT * FROM T WHERE X IN (B,D,F,I)

Q3 :SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K,M,N)

Q4 : SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K)

Q5: SELECT * FROM T WHERE X IN (B,D,F,I ,M,N)

a. (workload)

Query	ค่าของแอทริบิวต์ X
Q1	A,C,E,G,O,H,J,K,P
Q2	B,D,F,I
Q3	A,C,E,G,O,H,J,K,M,N
Q4	A,C,E,G,O,H,J,K
Q5	B,D,F,I ,M,N

b. (ฐานข้อมูลทรานแซคชัน)

รูปที่ 3 ตัวอย่างฐานข้อมูลทรานแซคชันบนแอทริบิวต์ X ที่สกัดจาก workload

ตัวอย่างเช่นจากกลุ่มข้อมูลที่หาได้ดังรูปที่ 5 ทำการเลือกกลุ่มข้อมูลตามลำดับที่ได้เรียงลำดับไว้ จะได้กลุ่มข้อมูล ACEGHOJK เป็นอับดับแรก หลังจากนั้นเลือกกลุ่มข้อมูลถัดไปที่มีสมาชิกไม่ซ้ำกับกลุ่มข้อมูลที่ถูกเลือกแล้ว จะได้เป็นกลุ่มข้อมูล BDFI และถัดไปคือ MN แล้วจึงนำกลุ่มข้อมูลที่ได้มาเรียงลำดับตามลำดับที่ถูกเลือกได้เป็น ACEGHOJKBDFIMN จากนั้นนำค่าข้อมูลที่เหลือมาเรียงต่อท้าย ดังนั้นจะได้ผลลัพธ์สุดท้ายในการจัดเรียงลำดับของค่าข้อมูลเป็น ACEGHOJKBDFIMNLP และทำการเข้ารหัสตามลำดับที่ถูกจัดเรียง ซึ่งจะได้ผลลัพธ์ดังรูปที่ 6b

เมื่อเราต้องการสอบถามข้อมูลตามเงื่อนไข “X IN { A,C,E,G,O,H,J,K }” โดยใช้ตารางเทียบค่า 6b จะได้ฟังก์ชันในการสอบถามข้อมูลคือ $B_3'B_2'B_1'B_0'$ + $B_3'B_2'B_1'B_0$ + $B_3'B_2'B_1\,B_0'$ + $B_3'B_2'B_1\,B_0$ + $B_3'B_2B_1'B_0'$ + $B_3'B_2B_1'B_0$ + $B_3'B_2B_1\,B_0'$ + $B_3'B_2B_1\,B_0$ ซึ่งสามารถลดรูปฟังก์ชันการสอบถามข้อมูลได้เป็น $B_3'[B_2'(B_1'(B_0'+B_0)\ +B_1(B_0'+B_0))+ B_2(B_1'(B_0'+B_0)+B_1(B_0'+B_0))] = B_3'$ ดังนั้นตรวจสอบเพียงบิตแมป B_3' ก็สามารถได้คำตอบของการสอบถามนี้ แต่หากเราใช้ตารางเทียบค่า 6a ซึ่งไม่มีการใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัส ในการสอบถามเงื่อนไขเดียวกันจะได้ฟังก์ชันในการสอบถามข้อมูลคือ $B_3'B_2'B_1'B_0'+B_3'B_2'B_1\,B_0'+$ $B_3'B_2B_1'B_0'+B_3'B_2B_1\,B_0'+B_3B_2B_1\,B_0'+B_3'B_2B_1\,B_0+$ $B_3B_2'B_1'B_0+$ $B_3B_2'B_1\,B_0'$ จะเห็นว่าไม่สามารถลดรูปฟังก์ชันได้ ดังนั้นจะต้องตรวจเกือบทุกบิตแมปเวกเตอร์ของแต่ละค่าข้อมูล นั่นหมายความว่าเวลาที่ใช้ในการสอบข้อมูลเพิ่มขึ้น

Query	ค่าของแอทริบิวต์ X															
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Q1	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	1
Q2	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0
Q3	1	0	1	0	1	0	1	1	0	1	1	0	1	1	1	0
Q4	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	0
Q5	0	1	0	1	0	1	0	0	1	0	0	0	1	1	0	0

a. (ตัวอย่างการสกัดค่า (item) ที่ถูกสอบถามในแต่ละการสอบถาม)

Item	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
support	3	2	3	2	3	2	3	3	2	3	3	0	2	2	3	1

b. (support ของแต่ละ item)

รูปที่ 4 ตัวอย่างตารางเมตริกและค่า support ของ 1-itemsets

5. การวิเคราะห์ประสิทธิภาพ

ในหัวข้อนี้จะแสดงให้เห็นประสิทธิภาพในการใช้พื้นทีและเวลาที่ใช้ในการสอบถามข้อมูลของการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ (Optimizing Encoded Bitmap Index using Association Rules) โดยจะทำการเปรียบเทียบกับการทำดัชนีบิตแมปแบบต่าง ๆ คือ ดัชนีบิตแมปแบบพื้นฐาน (Simple) ดัชนีบิตแมปแบบช่วง (Interval) ดัชนีบิตแมปแบบกระจาย (Scatter) ดัชนีบิตแมปแบบคู่กัน (Dual) และดัชนีบิตแมปแบบเข้ารหัสทั่วไป (Encode)

การวิเคราะห์ประสิทธิภาพในการใช้พื้นที่จะพิจารณาจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนีดังตารางที่ 1 สรุปได้ดังนี้

1. ดัชนีบิตแมปแบบพื้นฐานใช้จำนวนบิตแมปเวกเตอร์มากที่สุด (ใช้พื้นที่มากสุด) คือ C บิตแมปเวกเตอร์
2. ดัชนีบิตแมปแบบช่วงใช้จำนวนบิตแมปเวกเตอร์เป็นครึ่งหนึ่งของดัชนีบิตแมปแบบพื้นฐาน

itemsets	support(%)
ACEGHOJK	60
ACEG, ACEH, ACEJ, ACEK, ACEO, ACGH, ACGJ, ACGK, ACGO, ACHJ, ACHK, ACHO, ACJK, ACJO, ACKO, AEGH, AEGJ, AEGK, AEGO, AEHJ, AEHK, AEHO, AEJK, AEJO, AEKO, AGHJ, AGHK, AGHO, AGJK, AGJO, AGKO, AHJK,AHJO, AHKO, AJKO, CEGH, CEGJ, CEGK, CEGO, CEHJ, CEHK, CEHO, CEJK, CEJO, CEKO, CGHJ, CGHK, CGHO, CGJK, CGJO, CGKO, CHJK, CHJO, CHKO, CJKO, EGHJ, EGHK, EGHO, EGJK, EGJO, EGKO, EHJK, EHJO, EHKO, EJKO, GHJK, GHJO, GHKO, HJKO, HOJK	60
BDFI	40
AC, AE, AG, AH, AJ, AK, AO, CE, CG, CH, CJ, CK, CO, EG, EH, EJ, EK, EO, GH, GJ, GK, GO, HJ, HK, HO, JK, JO, KO	60
BD, BF, BI, DF, DI, FI, **MN**	40

รูปที่ 5 Frequent itemsets (กำหนดค่า support = 40%)

3. ดัชนีบิตแมปแบบกระจายใช้จำนวนบิตแมปเวกเตอร์น้อยกว่าดัชนีบิตแมปแบบช่วงแต่มากกว่าดัชนีบิตแมปแบบคู่กัน
4. ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ ใช้จำนวนบิตแมปเวกเตอร์น้อยที่สุด (ใช้พื้นที่น้อยสุด) คือ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์

ดังนั้น ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์มีประสิทธิภาพในการใช้พื้นที่ดีที่สุด

สำหรับการวิเคราะห์ประสิทธิภาพเรื่องเวลา เราจะพิจารณาจากจำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบและจำนวนครั้งในการดำเนินการตรรกะ โดยแยกวิเคราะห์ออกเป็น 2 กรณีคือการสอบถามข้อมูลแบบค่าเท่ากันและการสอบถามข้อมูลแบบสมาชิก

กรณีที่ 1 การสอบถามข้อมูลแบบค่าเท่ากัน พิจารณาจากตาราง 1 สรุปได้ดังนี้

1. ดัชนีบิตแมปแบบพื้นฐานมีการตรวจสอบ 1 บิตแมปเวกเตอร์ โดยไม่ต้องดำเนินการตรรกะ จึงใช้เวลาน้อยที่สุด
2. ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์มีการตรวจสอบทุกบิตแมปเวกเตอร์ แล้วนำมาเปรียบเทียบกับตารางเทียบค่า จึงใช้เวลามากที่สุด

A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000
J	1001
K	1010
L	1011
M	1100
N	1101
O	1110
P	1111

a. (ไม่ใช้กฎความสัมพันธ์)

A	0000
C	0001
E	0010
G	0011
H	0100
O	0101
J	0110
K	0111
B	1000
D	1001
F	1010
I	1011
M	1100
N	1101
L	1110
P	1111

b. (ใช้กฎความสัมพันธ์)

รูปที่ 6 Mapping table

3. ดัชนีบิตแมปแบบกระจายและดัชนีบิตแมปแบบคู่กันมีการตรวจสอบ 2 บิตแมปเวกเตอร์ และดำเนินการตรรกะ 1 ครั้ง จึงใช้เวลาในการสอบถามมากกว่าดัชนีบิตแมปแบบพื้นฐานแต่น้อยกว่าดัชนีบิตแมปแบบช่วง

ดังนั้น ดัชนีบิตแมปแบบพื้นฐานมีประสิทธิภาพในการสอบถามข้อมูลแบบค่าเท่ากันดีที่สุด

กรณีที่ 2 การสอบถามข้อมูลแบบสมาชิก พิจารณาจากตาราง 1 สรุปได้ดังนี้

1. ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัสทั่วไปทำได้โดยการหาคำตอบของแต่ละค่าสมาชิก แล้วนำผลลัพธ์ที่ได้มาดำเนินการตรรกะ OR
2. ดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ตรวจสอบจำนวนบิตแมปเวกเตอร์เพียง 1 บิตแมปเวกเตอร์ ดังนั้นดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ มีประสิทธิภาพในแง่ของการสอบถามข้อมูลแบบสมาชิกดีที่สุด

แม้ว่าในการสอบถามข้อมูลแบบสมาชิก ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กันมีโอกาสที่จะตรวจสอบบิตแมปเวกเตอร์แค่ 1 บิตแมปเวกเตอร์เหมือนกับดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์แต่เมื่อเทียบกับการใช้พื้นที่ในการจัดเก็บดัชนีแล้วดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์ใช้พื้นที่ในการจัดเก็บน้อยกว่ามาก

ตาราง 1 แสดงการเปรียบเทียบประสิทธิภาพของดัชนีบิตแมปแบบต่าง ๆ (เมื่อ C คือคาร์ดินอร์ลิตี้และ k คือจำนวนสมาชิกที่ถูกสอบถาม)

ดัชนีบิตแมป	พื้นที่	เวลา	
	จำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนีบิตแมป	จำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบ:จำนวนครั้งในการดำเนินการตรรกะ	
		การสอบถามแบบค่าเท่ากัน	การสอบถามแบบสมาชิก
Simple	C	1:0	k: k-1 (OR)
Interval	$\left\lceil \frac{C}{2} \right\rceil$	2:2 (1AND,1NOT)	2k : 3k-1 (k AND,k-1OR,k NOT)
Scatter	$2\lceil \sqrt{C} \rceil$	2:1 (1AND)	2k: 2k-1(k AND, k-1OR)
Dual	$\lceil \sqrt{2C+0.25}+0.5 \rceil$	2:1 (1AND)	2k: 2k-1(k AND, k-1OR)
Encode	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	$k\lceil \log_2 C \rceil$:use mapping table, k-1 OR
Optimizing Encode	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	1:0

6. สรุปและงานที่จะทำในอนาคต

ดัชนีบิตแมปแบบเข้ารหัส (ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้กฎความสัมพันธ์) เป็นการทำดัชนีที่มีประสิทธิภาพในการใช้พื้นที่ในการจัดเก็บดัชนีสูงที่สุด (ใช้พื้นที่น้อย) สามารถทำดัชนีกับแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้สูงได้ ซึ่งสิ่งสำคัญที่สุดในการทำดัชนีบิตแมปแบบเข้ารหัสคือการเข้ารหัสที่ดี (well-defined encoding) การใช้กฎความสัมพันธ์มาช่วยในการเข้ารหัสในการทำดัชนีบิตแมปแบบเข้ารหัสจะทำให้ประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกมีประสิทธิภาพเพิ่มขึ้น เพราะในการสอบถามข้อมูลจะตรวจสอบเพียง 1 บิตแมปเวกเตอร์เท่านั้น

งานวิจัยที่น่าสนใจที่จะทำต่อในอนาคตคือ การใช้เทคนิคการทำเหมืองข้อมูลแบบต่าง ๆ มาช่วยในการกรองข้อมูลเพื่อเพิ่มประสิทธิภาพในการทำดัชนีบิตแมปแบบเข้ารหัส และการศึกษาการหาขั้นตอนวิธีของการทำดัชนีบิตแมปแบบเข้ารหัสเพื่อเพิ่มประสิทธิภาพในการสอบถามแบบค่าเท่ากัน

7. เอกสารอ้างอิง

[1] R. Agrawal and R. Srikant, “Fast Algorithm for mining association rules in large database”,Proceeding of the 20th VLDB Conference Santiago,Chile, 1994, pp.487-499.

[2] C.Y. Chan and Y. E. Ioannidis, “Bitmap Index Design and Evaluation” , Proceeding of the 1998 ACM SIGMOD international conference on Management of data , 1998, pp.355-366.

[3] C.Y. Chan and Y. E. Ioannidis , “An Efficient Bitmap Encoding Scheme for Selection Queries”, Proceedings of the 1999 ACM SIGMOD international conference on management of data, 1999, pp.215-226.

[4] Charles J. Bontempo,C.M. Saracco, “Coping with Complex Queries in Data Warehouse”,InfoDB Volume 10 Number 5, 2002.

[5] Intelligent Enterprise, Indexing Goes a NewDirection, www.wintercorp.com/rwintercolumn s/ie_9901.html, 1999.

[6] J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, 2000 ACM SIGMOD Intl. Conference on management of data, 2000, pp.1-12.

[7] K. Stockinger and K. Wu , “Bitmap Indices for Data Warehouse” ,In Data Warehouse and OLAP , 2007.

[8] MC. Wu and A. Buchmann, ,”Encoded Bitmap Indexing for Data Warehouses”, Proceeding of the Fourteenth International Conference on Data Engineering, Washington Dc, 1998, pp.220-230.

[9] O’Neil. Elizabeth and Patrick, “Bitmap Index Design Choices and Their Performance Implications” 11th International Database Engineering and Applications Symposium, 2007, pp.72-84.

[10]R.Agrawal, T.Imielinski and A.Swami, “Mining Association Rules between Sets of Items in Large Databases”, Proceedings of the 1993 ACM SIGMOD international conference on Management of data SIGMOD, 1993.

[11]S. Chaudhuri and U.Dayal, “An Overview of Data Warehousing and OLAP Technology”, ACM SIGMOD RECORD, 1997.

[12]S.Vanichayabon, J. Manfuekphan and L.Gruenwald, “Scatter Bitmap: Space-time Efficient Bitmap Indexing for Equality and Membership Queries”, Proceeding of IEEE International Conferences on cybernetics and Intelligent Systems , 2006, pp.1-6.

[13]N. Wattanakitrungroj, S. Vanichayabon, “Dual Bitmap Index: Space-Time Efficient Bitmap Index for Equality and Membership Queries”,Communications and Information Technologies, 2006, pp.568-573.

[14]J. Sainui and S. Vanichayobon, “Optimizing Encoded Bitmap Index Using Association Rules” Technical Report, Computer Science Department, Faculty of Science, Prince of Songkla University, Thailand, 2007.

ภาคผนวก ค

ผลงานวิจัยที่ได้รับการตีพิมพ์

เรื่อง	การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามข้อมูลแบบสมาชิกโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อย
งานประชุมวิชาการ	The 12th Computer Science and Engineering Conference (NCSEC 2008)
สถานที่	จังหวัดชลบุรี ประเทศไทย
วันที่	ระหว่างวันที่ 20-21 พฤศจิกายน 2551

การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามข้อมูลแบบสมาชิก โดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

Optimizing Encoded Bitmap Index for Membership Query by Using Frequent Itemsets Mining

จรรยา สายนุ้ย นิวรรณ วัฒนกิจรุ่งโรจน์ และ ศิริรัตน์ วณิชโยบล

ห้องปฏิบัติการวิจัยเทคโนโลยีระบบสารสนเทศและการประยุกต์ ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

มหาวิทยาลัยสงขลานครินทร์ อ.หาดใหญ่ จ.สงขลา 90112 E-mail: s5010220020@psu.ac.th

บทคัดย่อ

การทำดัชนีเป็นเทคนิคการเพิ่มความเร็วในการค้นหาข้อมูล โดยที่ไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ และสำหรับในระบบคลังข้อมูลแล้วการทำดัชนีที่นิยมใช้คือการทำดัชนีแบบบิตแมป เนื่องจากสามารถดำเนินการระดับบิตก่อนดึงข้อมูลจริง ทำให้การประมวลผลมีความรวดเร็วยิ่งขึ้น การทำดัชนีบิตแมปแบบเข้ารหัสเป็นเทคนิคการทำดัชนีแบบบิตแมปที่มีประสิทธิภาพในการใช้พื้นที่ในการจัดเก็บดัชนีสูงที่สุด เพื่อเพิ่มประสิทธิภาพเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิกของการทำดัชนีบิตแมปแบบเข้ารหัส ในงานวิจัยนี้จึงนำเอาเทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย มาช่วยในการเข้ารหัสเพื่อให้ได้การเข้ารหัสที่ดี และจากการวิเคราะห์ประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยกับเทคนิคการทำดัชนีบิตแมปแบบอื่น พบว่าการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย มีประสิทธิภาพในเรื่องพื้นที่และเวลาที่ใช้ในการสอบถามข้อมูลแบบสมาชิกดีกว่าเทคนิคการทำดัชนีบิตแมปแบบอื่นที่เคยมีมา

คำสำคัญ: ดัชนีบิตแมป, เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

Abstract

Indexing techniques based on bitmap representations are well suited to a warehouse system. They significantly improve query processing time by utilizing low-cost Boolean operations and multiple index scans, executing queries by performing simple predicate conditions on the index level before going to the primary data source. To optimize existing Encoded Bitmap Index, in this paper, we apply a data mining technique called frequent itemsets mining to find a well-defined encoding scheme, leading to improve query processing time. Our comparative study show that in the best case the performance of optimizing Encoded Bitmap Index using frequent itemsets mining is better than those found by existing techniques for membership queries from the point of view of space-time trade-off.

Keywords: bitmap index, frequent itemsets mining

1. คำนำ

ในสภาพแวดล้อมของระบบคลังข้อมูล การสอบถามข้อมูลจะเป็นแบบ Online Analytical Processing (OLAP) [1] โดยจะเป็นการสอบถามเพื่อช่วยในการตัดสินใจของผู้บริหาร ซึ่งการสอบถามจะมีความซับซ้อน (Complex query) และเป็นแบบทันทีทันใด (Ad hoc) แต่เนื่องด้วยข้อมูลที่เก็บอยู่ในระบบคลังข้อมูลนั้นมีปริมาณมาก ทำให้การประมวลผลในการสอบถามข้อมูลต้องใช้เวลามากขึ้น การลดเวลาในการประมวลผลสามารถทำได้หลายวิธี เช่น การประมวลผลแบบคู่ขนาน การเพิ่มสมรรถนะเครื่อง หรือการทำดัชนี [1-6] ซึ่งการทำดัชนีเป็นเทคนิคในการเพิ่มความเร็วที่ไม่ต้องเสียค่าใช้จ่ายใด ๆ ในการเพิ่มฮาร์ดแวร์ และสำหรับในระบบคลังข้อมูลแล้วดัชนีที่นิยมใช้คือการทำดัชนีแบบบิตแมป เนื่องจากการทำดัชนีแบบบิตแมปสามารถดำเนินการตรรกะระดับบิตเช่น AND, OR, และ NOT ระหว่างบิตแมปเวกเตอร์ก่อนดึงข้อมูลจริง ทำให้เวลาที่ใช้ในการสอบถามข้อมูลมีความรวดเร็วยิ่งขึ้น [1-8] นอกจากนี้การทำดัชนีแบบบิตแมปยังมีประสิทธิภาพในเรื่องพื้นที่ที่ใช้ในการจัดเก็บดัชนี โดยเฉพาะกับแอทริบิวที่มีค่าคาร์ดินอร์ลิตี้ต่ำ (เมื่อค่าคาร์ดินอร์ลิตี้คือจำนวนค่าที่แตกต่างกันบนคอลัมน์ที่นำมาทำดัชนี)

การทำดัชนีบิตแมปแบบพื้นฐาน [2] ในการแทนค่าข้อมูลจะใช้ 1 บิตแมปเวกเตอร์แทน 1 ค่าข้อมูล เช่นจากรูปที่ 1 เป็นการทำดัชนีบิตแมปแบบพื้นฐานบนแอทริบิวต์ X ของตาราง T โดยแอทริบิวต์ X ประกอบด้วยสมาชิก (Domain) คือ {A,B,C,D,E,F,G,H} ซึ่งมีค่าคาร์ดินอร์ลิตี้เท่ากับ 8 จึงมีการใช้ 8 บิตแมปเวกเตอร์ ได้แก่ S_A, S_B, S_C, ..., S_H สำหรับการแทนค่าในแต่ละบิตแมปเวกเตอร์นั้น จะกำหนดให้บิตที่ i ของบิตแมปเวกเตอร์ S_j มีค่าเป็น 1 เมื่อแถวที่ i ของตารางมีค่าเป็น j โดยที่ j เป็นสมาชิกของ X ส่วนบิตอื่น ๆ จะให้มีค่าเป็น 0 เช่นจากรูปที่ 1 บิตแมปเวกเตอร์ S_Aมีบิตที่ 3 และ 5 มีค่าเป็น 1 เนื่องจากแถวที่ 3 และแถวที่ 5 มีค่า

RID	...	X	...
1		B	
2		C	
3		A	
4		H	
5		A	
6		G	
7		D	
8		D	
9		F	
. . .		. . .	
10000		E	

a. ตาราง T

S_A	S_B	S_C	S_D	S_E	S_F	S_G	S_H
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
0	0	0	0	1	0	0	0

b. ดัชนีบิตแมปแบบพื้นฐาน

รูปที่ 1 ตัวอย่างการทำดัชนีบิตแมปแบบพื้นฐานบนแอทริบิวต์ X ของตาราง T

ของแอทริบิวต์ X = A ดังนั้นเมื่อต้องการสอบถามข้อมูลตามเงื่อนไข X = A สามารถหาได้โดยตรวจสอบบิตแมปเวกเตอร์ S_A และสำหรับการสอบถามข้อมูลแบบสมาชิกเช่น “X in {A,B,C,D}” สามารถหาคำตอบได้โดยดำเนินการตรรกะ OR ระหว่างบิตแมปเวกเตอร์ S_A, S_B, S_C และ S_D จะได้แถวที่ 1, 2, 3, 5, 7 และ 8 เป็นคำตอบของการสอบถาม โดยทั่วไปแล้วการทำดัชนีบิตแมปแบบพื้นฐานเหมาะสำหรับการทำดัชนีบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ต่ำเช่น แอทริบิวต์เพศ ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ 2 คือ เพศชายกับเพศหญิง (มีการใช้เพียง 2 บิตแมปเวกเตอร์ในการสร้างดัชนี) แต่ถ้าต้องการทำดัชนีแบบบิตแมปบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้ที่สูงขึ้น จำนวนบิตแมปเวกเตอร์ที่ใช้จะมากขึ้นด้วย ทำให้สิ้นเปลืองพื้นที่ที่ใช้ในการจัดเก็บดัชนี ด้วยเหตุนี้งานวิจัยเกี่ยวกับการทำดัชนีแบบบิตแมปจำนวนมากจึงมุ่งประเด็นไปที่การลดขนาดดัชนี จากการศึกษาเทคนิคการลดขนาดดัชนีแบ่งได้เป็น 2 วิธี คือเทคนิคที่มีการบีบอัดดัชนี (Compression Bitmap) [6,9,10] และเทคนิคที่ไม่มีการบีบอัดดัชนี (Uncompression Bitmap) [3,5,7,8,11] สำหรับในงานวิจัยนี้จะมุ่งประเด็นไปที่การทำดัชนีที่ไม่มีการบีบอัดดัชนี เนื่องจากการดำเนินการตรรกะระดับบิตบนบิตแมปที่มีการบีบอัดดัชนีจะทำไม่ได้โดยตรง

เทคนิคการทำดัชนีบิตแมปแบบเข้ารหัส [5] เป็นเทคนิคการทำดัชนีบิตแมปแบบไม่มีการบีบอัดดัชนีที่ใช้พื้นที่น้อยที่สุด [7,8] กล่าวคือมีการใช้ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์ (เมื่อ C คือค่าคาร์ดินอร์ลิตี้) ขณะที่เทคนิคการทำดัชนีบิตแมปแบบพื้นฐานใช้ C บิตแมปเวกเตอร์ แต่ในการสอบถามข้อมูลแบบค่าเท่ากันบนดัชนีบิตแมปแบบเข้ารหัสนั้นจะต้องตรวจสอบทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) และหากเป็นการสอบถามข้อมูลแบบสมาชิกสามารถทำได้โดยหาคำตอบของแต่ละค่าแล้วนำมาดำเนินการตรรกะ OR ทำให้ใช้เวลาในการสอบถามข้อมูลเพิ่มขึ้น ดังนั้นหากเราสามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามข้อมูลได้ โดยไม่ต้องตรวจสอบทุกบิตแมปเวกเตอร์ของแต่ละค่าที่ต้องการสอบถาม จะทำให้เวลาที่ใช้ในการสอบถามน้อยลง กล่าวคือจะทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น

หัวใจสำคัญในการทำดัชนีบิตแมปแบบเข้ารหัสคือ การเข้ารหัส ซึ่งการเข้ารหัสที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามให้เหลือน้อยที่สุดถือเป็นการเข้ารหัสที่ดี ทำให้ประสิทธิภาพในการสอบถามข้อมูลเพิ่มขึ้น ในงานวิจัยนี้จึงนำเทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยมาช่วยในการหากลุ่มค่าของแอทริบิวต์ที่มีโอกาสถูกสอบถามด้วยกันบ่อย ๆ เพื่อที่จะเข้ารหัสกลุ่มค่าของแอทริบิวต์ที่มีโอกาสถูกสอบถามด้วยกันบ่อยนี้ให้สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามด้วยกัน

เอกสารนี้ประกอบด้วย 5 ส่วน โดยส่วนที่ 2 จะกล่าวถึงการทำดัชนีบิตแมปแบบเข้ารหัสและเทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย ส่วนที่ 3 จะเป็นการนำเสนอเทคนิคการเพิ่มประสิทธิภาพการสอบถามข้อมูลแบบสมาชิกของดัชนีบิตแมปแบบเข้ารหัสโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย ส่วนที่ 4 จะเป็นการวิเคราะห์ประสิทธิภาพ และส่วนสุดท้ายจะเป็นบทสรุป

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ดัชนีบิตแมปแบบเข้ารหัส

Wu และ Buchmann ได้เสนอเทคนิคการทำดัชนีบิตแมปแบบเข้ารหัส [5] ซึ่งในการแทนค่าของแอทริบิวต์ที่เลือกมาทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้เท่ากับ C ประกอบด้วย $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์คือ $E_{\lceil \log_2 C \rceil - 1}, \ldots, E_2, E_1, E_0$ และตารางเทียบค่า (Mapping Table) โดยที่แต่ละค่าของแอทริบิวต์จะถูกเข้ารหัสด้วย $E_{\lceil \log_2 C \rceil - 1} \ldots E_2 E_1 E_0$ เมื่อ $E_i \in \{ B_i , B_i' \}$ โดยให้ $B_i = 1$, $B_i' = 0$, $i = 0, 1, 2, \ldots, \lceil \log_2 C \rceil - 1$

ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสดังรูปที่ 2 เป็นการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T ซึ่งมีการใช้เพียง $\lceil \log_2 8 \rceil = 3$ บิตแมปเวกเตอร์ ได้แก่บิตแมปเวกเตอร์ E_2, E_1 และ E_0 ในการแทนค่าของแอทริบิวต์แต่ละค่า เช่นจากตารางเทียบค่า A ถูกเข้ารหัสด้วย $B_2' B_1' B_0'$ หรือ 000 ในการสอบถามข้อมูลแบบค่าเท่ากันจะต้องตรวจสอบกับตารางเทียบค่าก่อนว่าแต่ละค่าที่ต้องการสอบถามถูกเข้ารหัสในรูปแบบใด แล้วจึงนำไปตรวจสอบกับบิตแมปเวกเตอร์ว่าตรงกับแถวใด ก็จะได้แถวนั้นมีค่าของแอทริบิวต์ตรงกับที่ต้องการ ตัวอย่างเช่นต้องการสอบถาม X = A จากตาราง T ในรูปที่ 2 จะต้องตรวจสอบค่า A จากตารางเทียบค่าจะได้ว่าค่า A ถูกเข้ารหัสด้วย $B_2' B_1' B_0'$ หรือ 000 จากนั้นจึงนำไปตรวจสอบกับบิตแมปแบบเวกเตอร์ จะได้ว่าแถวที่ 3 และ 5 มีบิตแมปเวกเตอร์ $E_2 = 0$, $E_1 = 0$ และ $E_0 = 0$ ดังนั้นแถวที่ 3 และแถวที่ 5 เป็นคำตอบของการสอบถามนี้

สำหรับการสอบถามข้อมูลแบบสมาชิกทำได้โดยหาคำตอบของแต่ละสมาชิก แล้วนำมาดำเนินการตรรกะ OR (ในที่นี้ใช้สัญลักษณ์ + แทน OR) พิจารณาตัวอย่าง เมื่อต้องการสอบถามข้อมูลแบบสมาชิก “X

RID	...	X	...
1		B	
2		C	
3		A	
4		H	
5		A	
6		G	
7		D	
8		D	
9		F	
.		.	
.		.	
.		.	
10000		E	

a. ตาราง T

E_2	E_1	E_0
0	0	1
0	1	0
0	0	0
1	1	1
0	0	0
1	1	0
0	1	1
0	1	1
1	0	1
.	.	.
.	.	.
.	.	.
1	0	0

Mapping table

A	000
B	001
C	010
D	011
E	100
F	101
G	110
H	111

b. ดัชนีบิตแมปแบบเข้ารหัสและตาราเทียบค่า

รูปที่ 2 ตัวอย่างการทำดัชนีบิตแมปแบบเข้ารหัสบนแอทริบิวต์ X ของตาราง T

in {A,D,F,G}" โดยใช้ตารางการเทียบค่าจากรูปที่ 2 จะได้ฟังก์ชันในการสอบถามข้อมูลเป็น $B_2'B_1'B_0' + B_2'B_1B_0 + B_2B_1'B_0 + B_2B_1B_0'$ จะเห็นว่าต้องตรวจสอบทุกบิตแมปเวกเตอร์ของแต่ละค่าจึงจะได้คำตอบของการสอบถาม อย่างไรก็ตาม ถ้าเป็นการสอบถามข้อมูลตามเงื่อนไข "X in {A,B,C,D}" จะได้ฟังก์ชันในการสอบถามข้อมูลเป็น $B_2'B_1'B_0' + B_2'B_1'B_0 + B_2'B_1B_0' + B_2'B_1B_0$ ซึ่งสามารถลดรูปฟังก์ชันการสอบถามข้อมูล [5] ได้เป็น $B_2'(B_1'(B_0' + B_0) + B_1(B_0' + B_0)) = B_2'$ เพราะฉะนั้นตรวจสอบเพียงบิตแมปเวกเตอร์ B_2' ก็สามารถได้คำตอบของการสอบถาม

จากที่กล่าวมาจะเห็นว่า การเข้ารหัสแต่ละค่าของแอทริบิวต์ในตารางเทียบค่าเป็นหัวใจของการลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามข้อมูลแบบค่าสมาชิก ดังนั้นหากเราสามารถเข้ารหัสกลุ่มค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อย ๆ ให้อยู่ในรูปแบบที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสอบถามข้อมูลได้ ก็จะเป็นการเพิ่มประสิทธิภาพในการสอบถามข้อมูล กล่าวคือสามารถลดเวลาในการค้นหาข้อมูลได้มาก ซึ่งในการหากลุ่มค่าของแอทริบิวต์ที่มีการสอบถามด้วยกันบ่อย ๆ นั้น สามารถนำเทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย (Frequent Itemsets Mining) มาช่วยในการจัดกลุ่มค่าของแอทริบิวก่อนสร้างตารางเทียบค่าที่นำไปสู่การลดรูปบิตแมปเวกเตอร์เมื่อมีการสอบถามข้อมูลด้วยกัน

2.2 เทคนิคการหากลุ่มข้อมูลที่ปรากกฏด้วยกันบ่อย

เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยเป็นขั้นตอนหนึ่งในการหากฎความสัมพันธ์ [12] เมื่อกฎความสัมพันธ์คือการหารูปแบบความสัมพันธ์ระหว่างกลุ่มของข้อมูลหรือสิ่งของ (Items) ในฐานข้อมูลรายการทรานแซคชัน ซึ่งอยู่ภายใต้ค่าสนับสนุน (Support) และค่าความเชื่อมั่น (Confidence) กระบวนการหากฎความสัมพันธ์ประกอบด้วย 2 ขั้นตอนคือ [14]

1. การหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย ๆ (Frequent Itemsets) ในฐานข้อมูลรายการทรานแซคชัน ที่มีค่าสนับสนุนมากกว่าหรือเท่ากับค่าสนับสนุนขั้นต่ำ (Minimum Support) เมื่อค่าสนับสนุนของกลุ่มข้อมูล Y คือจำนวนครั้งที่กลุ่มข้อมูล Y ปรากฏในฐานข้อมูล

2. การสร้างกฎความสัมพันธ์ จาก Frequent Itemsets ที่หาได้จากขั้นตอนที่ 1 และจะยอมรับกฎความสัมพันธ์ที่สร้างขึ้น ก็ต่อเมื่อกฎนั้นมีค่าความเชื่อมั่นมากกว่าหรือเท่ากับค่าความเชื่อมั่นขั้นต่ำ

ในงานวิจัยนี้จะมุ่งเน้นไปที่ขั้นตอนที่ 1 คือการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย เพื่อช่วยในการจัดกลุ่มค่าของแอทริบิวต์ก่อนสร้างตารางเทียบค่าบนแอทริบิวต์ที่สนใจจะทำดัชนีบิตแมปแบบเข้ารหัส โดยคณะผู้วิจัยได้ทำการศึกษาอัลกอริทึมสำหรับการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย ได้แก่อัลกอริทึม Apriori [13] อัลกอริทึม FP-growth [14] อัลกอริทึม Closed [15] และอัลกอริทึม Index-BitTableFI [16] เพื่อหาขั้นตอนวิธีที่ดีที่จะนำมาใช้ในงานวิจัยนี้ จากการศึกษาอัลกอริทึมตามที่กล่าวมา พบว่ายังไม่มีประสิทธิภาพสำหรับงานวิจัยนี้ เพราะในการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย งานวิจัยนี้สนใจเฉพาะกลุ่มข้อมูลที่มีขนาด 2^i เมื่อ $i = 1, 2, \ldots, \lceil \log_2 C \rceil - 1$ ที่แต่ละกลุ่มมีสมาชิกไม่ซ้ำกัน ดังนั้นคณะผู้วิจัยจึงนำเสนออัลกอริทึม EncodedBitmapFIM ในหัวข้อ 3.1.2 สำหรับการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

3. การเพิ่มประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัส

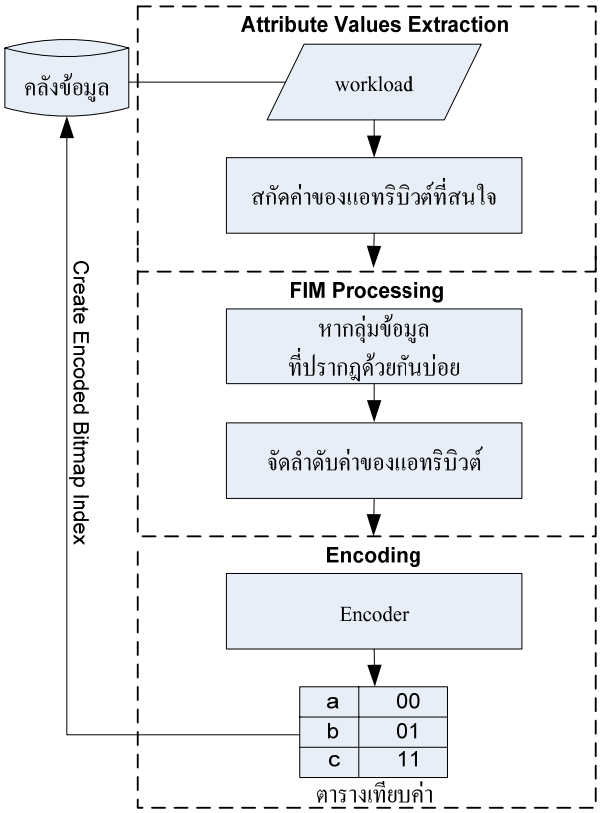
กระบวนการเพิ่มประสิทธิภาพของการทำดัชนีบิตแมปแบบเข้ารหัสแสดงดังรูปที่ 3 ประกอบด้วย 3 ส่วนหลัก คือ Attribute Values Extraction, FIM Processing และ Encoding ดังมีรายละเอียดต่อไปนี้

3.1 Attribute Values Extraction

Attribute Values Extraction เป็นขั้นตอนการสกัดค่าของแอทริบิวต์ที่สนใจจะนำมาทำดัชนีจากการสอบถามทั้งหมด (Workload) เพื่อสร้างตารางค่าของแอทริบิวต์ รูปที่ 5 แสดงตัวอย่าง Workload และค่าของแอทริบิวต์ X ที่สกัดได้ โดยแอทริบิวต์ X ประกอบด้วยสมาชิก 16 ตัว คือ{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P} ซึ่งจะใช้ในการอธิบายกระบวนการต่าง ๆ ที่เกิดขึ้นในขั้นตอนถัดไป

3.2 FIM Processing

FIM (Frequent Itemsets Mining) Processing เป็นขั้นตอนการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย ในที่นี้หมายถึงค่าของแอทริบิวต์ที่ถูกสอบถามด้วยกันบ่อยและจัดลำดับค่าของแอทริบิวต์ เพื่อนำไปสู่การสร้างตารางเทียบค่าที่สามารถลดจำนวนบิตแมปเวกเตอร์ที่ต้องตรวจสอบเมื่อมีการสอบถามข้อมูลแบบค่าสมาชิก ขั้นตอนนี้มี 2 กระบวนการคือ การหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยและการจัดลำดับค่าของแอทริบิวต์



รูปที่ 3 ขั้นตอนการเพิ่มประสิทธิภาพการทำดัชนีบิตแมปแบบเข้ารหัส

3.2.1 การหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

งานวิจัยนี้ได้นำเสนออัลกอริทึม EncodedBitmapFIM สำหรับการหากลุ่มค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อยสำหรับการทำดัชนีบิตแมปแบบเข้ารหัสดังรูปที่ 4 และตารางที่ 1 แสดงนิยามสัญลักษณ์ที่ใช้ในอัลกอริทึมนี้ ในที่นี้จะใช้ Item แทนค่าของแอทริบิวต์ และ Itemsets แทนกลุ่มค่าของแอทริบิวต์

ตารางที่ 1 นิยามสัญลักษณ์ที่ใช้ในอัลกอริทึม EncodedBitmapFIM

pointer	ตำแหน่งของ Item ที่กำลังพิจารณา โดย $0 \le pointer < N$
$Item_{pointer}$	Item ที่กำลังพิจารณา
N	จำนวน Items ในตาราง BitMatrix ณ ปัจจุบัน
Delegate	เซตของ Items ตั้งแต่ $Item_{pointer}$ เป็นต้นไปที่เกิดพร้อมกับ $Item_{pointer}$
C_k	Candidate Itemsets ขนาด k
F_k	Frequent Itemsets ขนาด k
tail	Items ทั้งหมดที่อยู่ทางขวาและไม่ได้เป็นสมาชิกของ Delegate ของ $Item_{pointer}$

อัลกอริทึม EncodedBitmapFIM : การหากลุ่มค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อย

1. อ่านตารางค่าของแอทริบิวต์หนึ่งครั้งเพื่อสร้างตาราง BitMatrix
2. หา Support ของแต่ละ Items โดยการนับจำนวนบิตที่มีค่า 1 ของแต่ละ Items
3. เรียงลำดับ Items ในตาราง BitMatrix ตาม Support จากน้อยไปมากและเลือกเฉพาะ Items ที่มี Support มากกว่าหรือเท่ากับ Minimum Support ที่กำหนด
4. For ($k = 2^{\lceil \log_2 C \rceil - 1}$; $k \ge 2$; $k = k/2$)
5. While($N \ge k$)
6. pointer = $N - k$;
7. While (pointer ≥ 0)
8. $Delegate.item_{pointer}$ = $Gen\_Delegate(Item_{pointer})$;
9. If (จำนวนสมาชิกของ $Delegate.item_{pointer} = k$)
10. $F_k = F_k \cup Delegate.item_{pointer}$;
11. Remove $Delegate.Item_{pointer}$ from BitMatrix table;
12. N = N-k ; Exit Loop;
13. Else
14. If ((N-pointer) > k และ $Support.item_{pointer} > MinSup$)
15. $C_k = Gen\_Candidate(Delegate.item_{pointer}, tail)$
16. If ($Support.C_k \ge MinSup$)
17. $F_k = F_k \cup C_k$;
18. Remove C_k from BitMatrix table;
19. N = N-k ; Exit Loop;
20. End if
21. End if
22. End if
23. pointer = pointer – 1;
24. End while
25. End while
26. End for
27. Return F_k และ Support ของแต่ละ F_k

รูปที่ 4 อัลกอริทึม EncodedBitmapFIM

อัลกอริทึม EncodedBitmapFIM ประกอบด้วย 27 ขั้นตอน กล่าวคือขั้นตอนที่ 1-3 เป็นการเตรียมข้อมูลสำหรับการคำนวณหา Frequent k-itemsets (F_k) ในขั้นตอนที่ 4-26 ซึ่งมีการเรียกใช้ฟังก์ชัน Gen_Delegate (ขั้นตอนที่ 8) ในการสร้าง Delegate และใช้ฟังก์ชัน Gen_Candidate (ขั้นตอนที่ 15) ในการสร้าง Candidate Itemsets การทำงานของฟังก์ชัน Gen_Delegate เป็นการดำเนินการตรรกะ AND ในแนวนอนระหว่างทุกการสอบถาม (Q) ที่ Item ที่กำลังพิจารณาปรากฏอยู่

โดยเริ่มดำเนินการตรรกะ AND ตั้งแต่ตำแหน่ง Item ที่กำลังพิจารณาเป็นต้นไป กรณีที่จำนวนสมาชิกของ Delegate ของ Item ที่กำลังพิจารณาเท่ากับ k ถือเป็นกลุ่มค่าของแอทริบิวต์ที่สนใจ (ขั้นตอนที่ 9-12) แต่ถ้าไม่เท่ากับ k และอยู่ภายใต้เงื่อนไขในขั้นตอนที่ 14 ส่วนการทำงานของฟังก์ชัน Gen_Candidate จะเริ่มสร้าง C_k จากการรวมกันของ Delegate กับ tail ของ Item ที่กำลังพิจารณาและดำเนินการตรรกะ AND ในแนวตั้งระหว่าง Item ใน C_k เพื่อหาค่า Support ของ C_k ถ้าค่า Support ของ C_k มากกว่าหรือเท่ากับค่า Minimum Support ที่กำหนด จะถือว่าเป็นกลุ่มค่าของแอทริบิวต์ที่สนใจ เพื่อให้เห็นภาพของการทำงานของอัลกอริทึม EncodedBitmapFIM ตัวอย่างในรูปที่ 5 จะถูกใช้ในการอธิบายการทำงานของอัลกอริทึมดังรายละเอียดดังนี้

ขั้นตอนที่ 1 เป็นการสร้างตาราง BitMatrix (รูปที่ 6a) จากตารางค่าของแอทริบิวต์ (รูปที่ 5b) กำหนดให้ BitMatrix มีขนาด $R \times C$ โดยที่ R คือจำนวนการสอบถามทั้งหมดใน Workload และ C คือค่าคาร์ดินอร์ลิตี้ของแอทริบิวต์ ซึ่งในแต่ละแถวแสดงให้เห็นว่าการสอบถามข้อมูล Q_i มีการสอบถามค่าของแอทริบิวต์ใดบ้างและในแต่ละคอลัมน์แสดงให้เห็นว่ามีค่าของ $Item_j$ ถูกสอบถามในการสอบถามใดบ้าง BitMatrix[i,j] มีค่าเท่ากับ 1 หมายความว่า Q_i มีการสอบถามค่าของ $Item_j$ ตัวอย่างเช่น BitMatrix[3,0] มีค่าเท่ากับ 1 หมายความว่า Q_3 มีการสอบถามค่าของ Item A และ BitMatrix[3,1] มีค่าเท่ากับ 0 หมายความว่า Q_3 ไม่มีการสอบถามค่าของ Item B ประโยชน์ของการสร้างตาราง BitMatrix คือ การอ่านตารางค่าของแอทริบิวต์เพียงครั้งเดียวในการหากลุ่มค่าของแอทริบิวต์ที่ปรากฏด้วยกันบ่อยขนาด k ถ้าใช้เทคนิคที่เคยมีมาเช่น อัลกอริทึม Apriori จะต้องเสียเวลาอ่านตารางค่าของแอทริบิวต์ k ครั้ง หรืออัลกอริทึม FP-growth จะต้องเสียเวลาอ่านตารางค่าของแอทริบิวต์ 2 ครั้ง เป็นต้น ในขั้นตอนที่ 2 เป็นการหาค่า Support ของแต่ละ Item ดังแสดงในรูปที่ 6b ขั้นตอนที่ 3 เรียงลับดับ Item ในตาราง BitMatrix จาก Item ที่มีค่า Support น้อยไปหามาก แล้วเลือกเฉพาะ Item ที่มีค่า Support มากกว่าหรือเท่ากับค่า Minimum Support เมื่อกำหนดให้ค่า Minimum Support เท่ากับ 40% จะได้ผลลัพธ์ดังรูปที่ 7 โดยตัวเลขที่อยู่หลังค่า Item แต่ละค่าคือค่า Support

ขั้นตอนที่ 4 - 26 เป็นขั้นตอนการหา Frequent k-itemsets (F_k) เมื่อ $k = 2^i$, $i = 1, 2, \ldots, \lceil \log_2 C \rceil - 1$ จาก Delegate ของแต่ละ Item โดยมีการทำงานทั้งหมด $\lceil \log_2 C \rceil - 1$ รอบ และเมื่อสิ้นสุดกระบวนการในขั้นตอนที่ 26 แล้วแต่ละ F_k ที่ได้จะต้องมีสมาชิกไม่ซ้ำกัน ดังนั้นเพื่อลดการสร้าง Candidate Itemsets ที่ไม่จำเป็น เมื่อหา F_k ใด ๆ ได้แล้วจะไม่นำ Item ที่เป็นสมาชิกของ F_k ที่หาได้มาก่อนหน้าไปพิจารณาในรอบถัดไป

จากตัวอย่างในรูปที่ 5 เนื่องจากคาร์ดินอร์ลิตี้ของแอทริบิวต์ X ที่นำมาทำดัชนีมีค่าเท่ากับ 16 ดังนั้นในรอบที่ 1 ของขั้นตอนที่ 4 จะเป็นการหา F_8 ขั้นตอนที่ 5 เป็นการตรวจสอบว่าจำนวน Item ที่มีอยู่ในตาราง BitMatrix มีจำนวนเพียงพอที่จะสร้าง F_8 หรือไม่ กล่าวคือจะเป็นการตรวจสอบจำนวน Item ในตาราง BitMatrix ในรูปที่ 7 ว่ามีจำนวน

$Q1$: SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K,P)

$Q2$: SELECT * FROM T WHERE X IN (B,D,F,I)

$Q3$: SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K,M,N)

$Q4$: SELECT * FROM T WHERE X IN (A,C,E,G,O,H,J,K)

$Q5$: SELECT * FROM T WHERE X IN (B,D,F,I ,M,N)

a. workload

Query	ค่าของแอทริบิวต์ X
$Q1$	A,C,E,G,O,H,J,K,P
$Q2$	B,D,F,I
$Q3$	A,C,E,G,O,H,J,K,M,N
$Q4$	A,C,E,G,O,H,J,K
$Q5$	B,D,F,I ,M,N

b. ตารางค่าของแอทริบิวต์

รูปที่ 5 ตัวอย่างตารางค่าของแอทริบิวต์ X ที่สกัดจาก workload

Query	ค่าของแอทริบิวต์ X															
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
$Q1$	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	1
$Q2$	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0
$Q3$	1	0	1	0	1	0	1	1	0	1	1	0	1	1	1	0
$Q4$	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	0
$Q5$	0	1	0	1	0	1	0	0	1	0	0	0	1	1	0	0

a. ตาราง BitMatrix

Item	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
support	3	2	3	2	3	2	3	3	2	3	3	0	2	2	3	1

b. Support ของแต่ละ Item

รูปที่ 6 ตัวอย่างตาราง BitMatrix และค่า Support ของ 1-itemsets

Query	ค่าของแอทริบิวต์ X													
	B:2	D:2	F:2	I:2	M:2	N:2	A:3	C:3	E:3	G:3	H:3	J:3	K:3	O:3
$Q1$	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$Q2$	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$Q3$	0	0	0	0	1	1	1	1	1	1	1	1	1	1
$Q4$	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$Q5$	1	1	1	1	1	1	0	0	0	0	0	0	0	0

รูปที่ 7 ตาราง BitMatrix ของ Items ที่ผ่าน Minimum Support ที่มีการเรียงลำดับ Items ตามค่า Support จากน้อยไปมาก

Query	ค่าของแอทริบิวต์ X					
	B:2	D:2	F:2	I:2	M:2	N:2
$Q1$	0	0	0	0	0	0
$Q2$	1	1	1	1	0	0
$Q3$	0	0	0	0	1	1
$Q4$	0	0	0	0	0	0
$Q5$	1	1	1	1	1	1

รูปที่ 8 ตาราง BitMatrix เมื่อเลือก Frequent 8-itemset ออก

มากกว่าหรือเท่ากับ 8 หรือไม่ จากตาราง BitMatrix รูปที่ 7 มีจำนวน Item ทั้งหมดเท่ากับ 14 (N=14 ซึ่งมากกว่า 8) ขั้นตอนที่ 6 เป็นการหาตำแหน่งของ Item ที่จะเริ่มพิจารณาเพื่อหา F_8 โดยกำหนดให้ตำแหน่ง (pointer) มีค่าตั้งแต่ 0 ถึง N-1 จากรูปที่ 7 Item B อยู่ตำแหน่งที่ 0, Item D อยู่ตำแหน่งที่ 1 เป็นต้น ดังนั้นตำแหน่งที่ 6 (14-8) ซึ่งมีค่าเท่ากับ A เป็นตำแหน่งเริ่มต้นที่จะพิจารณาเพื่อหา F_8 ขั้นตอนที่ 8 เป็นการหา Delegate ของ Item A จากรูปที่ 7 Item A ปรากฏอยู่ใน $Q1$, $Q3$ และ $Q4$ เราจึงดำเนินการตรรกะ AND ระหว่าง $Q1$, $Q3$ และ $Q4$ โดยเริ่มตั้งแต่ Item A ถึง Item O จะได้เป็น 11111111 AND 1111111111 AND 11111111 ผลลัพธ์ที่ได้คือ 11111111 จากผลลัพธ์ของการดำเนินการตรรกะ AND ถ้าบิตของ Item ใดมีค่าเป็น 1 จะได้ว่าเป็นสมาชิกใน Delegate นั้น ดังนั้น Delegate ของ Item A คือ {A,C,E,G,H,J,K,O} และจำนวนสมาชิกของ Delegate ของ Item A มีค่าเท่ากับ 8 จึงถือว่าเป็น Frequent 8-itemsets ดังนั้นจะได้ F_8 = {A,C,E,G,H,J K,O} จากนั้นเลือก Item เหล่านี้ออกจากตาราง BitMatrix (ขั้นตอนที่ 11-12) จะเหลือ Item ในตาราง BitMatrix จำนวน 6 Items ซึ่งไม่สามารถหา F_8 ได้อีก

ในรอบที่ 2 จะเป็นการหา F_4 โดยจะเริ่มพิจารณาที่ Item F (ตำแหน่งที่ 6-4 = 2 ของ Items ในตาราง BitMatrix รูปที่ 8) สามารถหา Delegate ของ Item F ได้จาก 1100 AND 1111 (ระหว่าง $Q2$ และ $Q5$ ตั้งแต่บิตของ F เป็นต้นไป) ได้ผลลัพธ์เป็น 1100 ดังนั้น Delegate ของ Item F คือ {F,I} ซึ่งมีสมาชิกเท่ากับ 2 แสดงว่าไม่สามารถหา F_4 จาก Delegate ของ Item F ได้ และไม่มีโอกาสสร้าง C_4 ในขั้นตอน 15 เนื่องจากไม่เข้าเงื่อนไขในขั้นตอนที่ 14 ดังนั้นจึงเลื่อนค่า pointer เพื่อพิจารณา Item ถัดไปทางซ้าย (ขั้นตอน 23) คือ Item D จะได้ Delegate ของ Item D คือ {D,F,I} ซึ่งมีสมาชิกเท่ากับ 3 แสดงว่าไม่สามารถหา F_4 จาก Delegate ของ Item D ได้ และไม่มีโอกาสสร้าง C_4 ในขั้นตอน 15 เนื่องจากไม่เข้าเงื่อนไขในขั้นตอนที่ 14 จึงต้องพิจารณา Item ถัดไปทางซ้ายของ Item D ซึ่งก็คือ Item B จะได้ว่า Delegate ของ Item B คือ {B,D,F,I} ซึ่งมีสมาชิกเท่ากับ 4 สามารถหา F_4 จาก Delegate ของ Item B ได้ ดังนั้นจะได้ F_4 = {B,D,F,I} จากนั้นเลือก Item เหล่านี้ออกจากตาราง BitMatrix (ขั้นตอนที่ 11-12) จะเหลือ Item ในตาราง BitMatrix จำนวน 2 Items ซึ่งไม่สามารถหา F_4 ได้อีก ในรอบที่ 3 เป็นการหา F_2 โดยพิจารณาในทำนองเดียวกันจะได้ F_2 = {M,N} ดังนั้นจากตัวอย่างที่กล่าวมาจะสามารถหา Frequent k-itemsets ตามอัลกอริทึม EncodedBitmapFIM ได้ดังรูปที่ 9

จำนวนสมาชิก (k)	Itemsets	support(%)
8	A,C,E,G,H,O,J,K	60
4	B,D,F,I	40
2	M,N	40

รูปที่ 9 กลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

3.2.2 จัดลำดับค่าของแอทริบิวต์

จากกลุ่มค่าของแอทริบิวต์ที่หาได้นำแต่ละกลุ่มมาจัดเรียงให้เป็นลำดับที่ต่อเนื่องกัน โดยจัดเรียงตามจำนวนสมาชิกของแต่ละกลุ่มค่าของแอทริบิวต์จากมากไปน้อย จากนั้นนำข้อมูลที่เหลือมาเรียงต่อจากข้อมูลที่จัดเรียงอยู่แล้ว ตัวอย่างเช่นจากกลุ่มข้อมูลที่หาได้ดังรูปที่ 9 นำมาจัดลำดับได้เป็น A,C,E,G,H,O,J,K,B,D,F,I,M,N และเมื่อนำข้อมูลที่เหลือมาเรียงต่อ จะได้ผลลัพธ์สุดท้ายของการจัดลำดับค่าของแอทริบิวต์เป็น A,C,E,G,H,O,J,K,B,D,F,I,M,N,L,P

3.3 Encoding

Encoding เป็นขั้นตอนการเข้ารหัสค่าของแอทริบิวต์ตามลำดับที่ถูกจัดลำดับตั้งแต่ข้อมูลตัวแรกจนถึงตัวสุดท้าย โดยกำหนดให้รูปแบบการลงรหัสเริ่มจาก 0 ถึง C-1 ในรูปของเลขฐานสอง ซึ่งผลลัพธ์ที่ได้จากขั้นตอนนี้จะเป็นตารางเทียบค่าของการทำดัชนีบิตแมปแบบเข้ารหัส ที่นำไปสู่การสร้างดัชนีบิตแมปแบบเข้ารหัสที่มีประสิทธิภาพสำหรับการสอบถามข้อมูลแบบสมาชิกต่อไป ดังนั้นจากตัวอย่างที่กล่าวมาจะได้ตารางเทียบค่าที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยในการเพิ่มประสิทธิภาพดังรูปที่ 10b เปรียบเทียบกับตารางเทียบค่าที่ไม่ผ่านกระบวนการเพิ่มประสิทธิภาพดังรูปที่ 10a

A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000
J	1001
K	1010
L	1011
M	1100
N	1101
O	1110
P	1111

a. ไม่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

A	0000
C	0001
E	0010
G	0011
H	0100
J	0101
K	0110
O	0111
B	1000
D	1001
F	1010
I	1011
M	1100
N	1101
L	1110
P	1111

b. ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย

รูปที่ 10 Mapping table

4. การวิเคราะห์ประสิทธิภาพ

ในการวิเคราะห์ประสิทธิภาพจะทำการเปรียบเทียบประสิทธิภาพในเรื่องพื้นที่ที่ใช้ในการจัดเก็บดัชนีและเวลาที่ใช้ในการสอบถามข้อมูลระหว่างการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยกับเทคนิคการทำดัชนีแบบบิตแมปแบบต่าง ๆ ได้แก่ การทำดัชนีบิตแมปแบบพื้นฐาน [2] ดัชนีบิตแมปแบบช่วง [3] ดัชนีบิตแมปแบบกระจาย [8] ดัชนีบิตแมปแบบคู่กัน [7] และดัชนีบิตแมปแบบเข้ารหัสทั่วไป [5]

4.1 การวิเคราะห์พื้นที่ที่ใช้ในการจัดเก็บดัชนี

การวิเคราะห์ประสิทธิภาพเรื่องพื้นที่จะพิจารณาจากจำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนีดังตารางที่ 2 สรุปได้ว่า ดัชนีบิตแมปแบบพื้นฐานใช้พื้นที่ในการจัดเก็บดัชนีมากที่สุดคือ C บิตแมปเวกเตอร์ ส่วนดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยใช้พื้นที่ในการจัดเก็บดัชนีน้อยที่สุด คือ $\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์

4.2 การวิเคราะห์เวลาที่ใช้ในการสอบถามข้อมูล

สำหรับการวิเคราะห์ประสิทธิภาพเรื่องเวลาที่ใช้ในการสอบถามข้อมูลจะพิจารณาจากจำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบและจำนวนครั้งในการดำเนินการตรรกะ โดยแยกวิเคราะห์เป็น 2 กรณีคือ การสอบถามข้อมูลแบบค่าเท่ากันและการสอบถามข้อมูลแบบค่าสมาชิก

4.2.1 การสอบถามข้อมูลแบบค่าเท่ากัน

จากตารางที่ 2 สรุปได้ว่าดัชนีบิตแมปแบบพื้นฐานใช้เวลาในสอบถามแบบค่าท่ากันน้อยที่สุด เนื่องจากตรวจสอบเพียง 1 บิตแมปเวกเตอร์และไม่มีการดำเนินการตรรกะ ส่วนดัชนีบิตแมปแบบเข้ารหัสทั่วไปและที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยใช้เวลาในการสอบถามข้อมูลแบบค่าเท่ากันมากที่สุด เนื่องจากต้องตรวจสอบทุกบิตแมปเวกเตอร์ ($\lceil \log_2 C \rceil$ บิตแมปเวกเตอร์) และมีการเปรียบเทียบกับตารางเทียบค่า ส่วนดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย และดัชนีบิตแมปแบบคู่กันตรวจสอบ 2 บิตแมปเวกเตอร์และมีการดำเนินการตรรกะ จึงใช้เวลาในการสอบถามข้อมูลแบบค่าเท่ากันมากกว่าดัชนีบิตแมปแบบพื้นฐานแต่ใช้เวลาน้อยกว่าดัชนีบิตแมปแบบเข้ารหัส (ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย)

4.2.2 การสอบถามข้อมูลแบบค่าสมาชิก

การสอบถามข้อมูลแบบค่าสมาชิก ดัชนีบิตแมปแบบพื้นฐาน ดัชนีบิตแมปแบบช่วง ดัชนีบิตแมปแบบกระจาย ดัชนีบิตแมปแบบคู่กัน และดัชนีบิตแมปแบบเข้ารหัสทั่วไปสามารถทำได้โดยหาคำตอบของแต่ละค่าแล้วนำมาดำเนินการตรรกะ OR กล่าวคือถ้ามีการสอบถามข้อมูลแบบค่าสมาชิก k ตัว แล้วจำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบจะเพิ่มขึ้นเป็น k เท่า และจำนวนครั้งในการดำเนินการตรรกะเพิ่มขึ้นมากกว่า k เท่าดังตารางที่ 2 ส่วนการทำดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย กรณีที่สมาชิกที่ต้องการสอบถามมีจำนวนสมาชิกเท่ากับ $2^{\lceil \log_2 C \rceil - i}$, i = 1, 2, ..., $\lceil \log_2 C \rceil$-1 มีโอกาสตรวจสอบเพียง i บิตแมปเวกเตอร์และไม่มีการดำเนินการตรรกะ กรณีที่ดีที่สุดตรวจสอบเพียง 1 บิตแมปเวกเตอร์

ตัวอย่างการสอบถามข้อมูลแบบสมาชิกโดยใช้ตารางเทียบค่ารูป 10a ซึ่งเป็นตารางเทียบค่าที่สร้างขึ้นโดยไม่ได้ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย และตารางเทียบค่ารูป 10b ซึ่งเป็นตารางเทียบค่าที่สร้างขึ้นโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย เมื่อต้องการสอบถามข้อมูลตามเงื่อนไข X in {A,C,E,G,H,J,K,O} โดยใช้ตารางเทียบค่ารูป 10a จะได้ฟังก์ชันในการสอบถามข้อมูลเป็น $B_3'B_0' + B_3'B_2B_1B_0 + B_3B_2'B_1'B_0 + B_3B_2'B_1B_0' + B_3B_2B_1B_0'$ หรือเมื่อต้องการสอบถามข้อมูลตามเงื่อนไข X in {B,D,F,I} จะได้ฟังก์ชันในการสอบถามข้อมูลเป็น $B_3'B_2'B_0 + B_3'B_1'B_0 + B_3B_2'B_1'B_0'$ แต่หากเป็นการสอบถามที่เหมือนกันโดยใช้ตารางเทียบค่ารูป 10b สามารถตรวจสอบเพียงบิตแมปเวกเตอร์ B_3' และ B_3B_2' ตามลำดับ ดังนั้นสำหรับกลุ่มข้อมูลที่มีการสอบถามด้วยกันบ่อยการทำดัชนีบิตแมปแบบเข้ารหัสโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยมีประสิทธิภาพกว่าเทคนิคการทำดัชนีบิตแมปแบบอื่นที่เคยมีมา

ตารางที่ 2 แสดงการเปรียบเทียบประสิทธิภาพของดัชนีบิตแมปแบบต่าง ๆ (เมื่อ C คือคาร์ดินอร์ลิตี้และ k คือจำนวนสมาชิกที่ถูกสอบถาม)

ดัชนีบิตแมป	พื้นที่	เวลา	
	จำนวนบิตแมปเวกเตอร์ที่ใช้ในการสร้างดัชนีบิตแมป	จำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบ:จำนวนครั้งในการดำเนินการตรรกะ	
		การสอบถามค่าเท่ากัน	การสอบถามแบบสมาชิก
แบบพื้นฐาน [2]	C	1:0	k: k-1 (OR)
แบบช่วง [3]	$\left\lceil \frac{C}{2} \right\rceil$	2:2 (1AND,1NOT)	2k : 3k-1 (k AND,k-1OR,k NOT)
แบบกระจาย [8]	$2\lceil \sqrt{C} \rceil$	2:1 (1AND)	2k: 2k-1(k AND, k-1OR)
แบบคู่กัน [7]	$\lceil \sqrt{2C+0.25}+0.5 \rceil$	2:1 (1AND)	2k: 2k-1(k AND, k-1OR)
แบบเข้ารหัสทั่วไป [5]	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	k$\lceil \log_2 C \rceil$:use mapping table, k-1 OR
แบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	1:0

5. สรุป

ดัชนีบิตแมปแบบเข้ารหัส (ดัชนีบิตแมปแบบเข้ารหัสทั่วไปและดัชนีบิตแมปแบบเข้ารหัสที่ใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อย) เป็นการทำดัชนีที่มีประสิทธิภาพในการใช้พื้นที่ในการจัดเก็บดัชนีสูงที่สุด สามารถทำดัชนีบนแอทริบิวต์ที่มีค่าคาร์ดินอร์ลิตี้สูงได้ สิ่งสำคัญที่สุดในการทำดัชนีบิตแมปแบบเข้ารหัสคือการเข้ารหัสที่ดี (Well-defined Encoding) ในงานวิจัยนี้จึงได้นำเอาเทคนิคการหากลุ่มข้อมูลที่ปรากฏด้วยกันบ่อยมาช่วยในการเข้ารหัสในการทำดัชนีบิตแมปแบบเข้ารหัส ซึ่งพบว่าจะทำให้ประสิทธิภาพในการสอบถามข้อมูลแบบสมาชิกมีประสิทธิภาพเพิ่มขึ้น เพราะสามารถลดจำนวนบิตแมปเวกเตอร์ที่ถูกตรวจสอบได้ โดยมีโอกาสตรวจสอบเพียง 1 บิตแมปเวกเตอร์เท่านั้น

เอกสารอ้างอิง

[1] S. Chaudhuri and U. Dayal, “An Overview of Data Warehousing and OLAP Technology”, *ACM SIGMOD RECORD*, Vol. 26, pp. 65-74, 1997.

[2] C. Y. Chan and Y. E. Ioannidis, “Bitmap Index Design and Evaluation” ,*Proceeding of the 1998 ACM SIGMOD international conference on Management of data* , pp.355-366,1998.

[3] C. Y. Chan and Y. E. Ioannidis , “An Efficient Bitmap Encoding Scheme for Selection Queries”, *Proceedings of the 1999 ACM SIGMOD international conference on management of data*, pp.215-226, 1999.

[4] K. Stockinger and K. Wu , “Bitmap Indices for Data Warehouse” ,*In Data Warehouse and OLAP* , 2007.

[5] MC. Wu and A. Buchmann, ,”Encoded Bitmap Indexing for Data Warehouses”, *Proceeding of the Fourteenth International Conference on Data Engineering*, pp.220-230,1998.

[6] O’Neil. Elizabeth and Patrick, “Bitmap Index Design Choices and Their Performance Implications” *11th International Database Engineering and Applications Symposium*, pp.72-84,2007.

[7] N. Wattanakitrungroj, S. Vanichayabon, “Dual Bitmap Index: Space-Time Efficient Bitmap Index for Equality and Membership Queries”, *International Symposium* on *Communications and Information Technologies (ISCIT'06)*, pp.568-573, 2006.

[8] S. Vanichayabon, J. Manfuekphan and L. Gruenwald, “Scatter Bitmap: Space-time Efficient Bitmap Indexing for Equality and Membership Queries”, *Proceeding of IEEE International Conferences on cybernetics and Intelligent Systems* , pp.1-6,2006.

[9] K. Wu, E.J. Otoo and A.Shoshani, “Compressing Bitmap Indexes for Faster Search Operation”, *International Conference on Scientific and Statistical Database Management (SSDBM)*, 2002.

[10] M. Stabno and R. Wrembel, “RLH:Bitmap Compression Technique Based on Run-Length and Huffman Encoding”, *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, pp. 14-48,2007.

[11] N. Koudas, “Space Efficient Bitmap Indexing”, *Proceedings of the ninth international conference on Information and knowledge management*, pp. 194-201, 2000.

[12] R. Agrawal, T. Imielinski and A. Swami, “Mining Association Rules between Sets of Items in Large Databases”, *Proceedings of the 1993 ACM SIGMOD international conference on Management of data SIGMOD*, 1993.

[13] R. Agrawal and R. Srikant, “Fast Algorithm for mining association rules in large database”, *Proceeding of the 20th VLDB Conference Santiago*, pp.487-499, 1994.

[14] J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, *2000 ACM SIGMOD Intl. Conference on management of data*, pp.1-12, 2000.

[15] N.Pasquier, Y.Bastide, R.taouil and L.Lakhal, “Efficient mining of association rules using closed itemset lattices”, *Information System*, Vol. 24, No. 1, pp. 25-46, 1999.

[16] W. Song, B. Yang and Z. Xu, “Index-BitTableFI:An improved algorithm for mining frequent itemsets”, *Knowledge-Based System*, 2008.

จรรยา สายนุ้ย นักศึกษาปริญญาโท ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ มีความสนใจทางด้าน Database and Information Retrieval, Data Mining.

นิวรรณ วัฒนกิจรุ่งโรจน์ อาจารย์ประจำภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ มีความสนใจทางด้าน Data Mining, Information Retrieval and Image Processing.

ศิริรัตน์ วณิชโยบล อาจารย์ประจำภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ มีความสนใจทางด้าน Database, Data Warehouse, Data Mining.

ภาคผนวก ง

ผลงานวิจัยที่ได้รับการตีพิมพ์

เรื่อง	Optimizing Encoded Bitmap Index using Frequent Itemsets Mining
งานประชุมวิชาการ	International Conference on Computer and Electrical Engineering (ICCEE 2008)
สถานที่	จังหวัดภูเก็ต ประเทศไทย
วันที่	ระหว่างวันที่ 21-22 ธันวาคม 2551

2008 International Conference on Computer and Electrical Engineering

Optimizing Encoded Bitmap Index using Frequent Itemsets Mining

Janya Sainui, Sirirut Vanichayobon and Niwan Wattanakitrungroj
Department of Computer Science
Faculty of Science, Prince of Songkla University
Songkhla, Thailand
e-mail : s5010220020@psu.ac.th, sirirut.v@psu.ac.th, niwan.w@psu.ac.th

***Abstract*— Indexing techniques based on bitmap representations are well suited to a warehouse system. They significantly improve query processing time by utilizing low-cost Boolean operations and multiple index scans, executing queries by performing simple predicate conditions on the index level before going to the primary data source. To optimize existing Encoded Bitmap Index, in this paper, we apply a data mining technique called frequent itemsets mining to find a well-defined encoding scheme, leading to improve query processing time. Our comparative study show that in the best case the performance of optimizing Encoded Bitmap Index using frequent itemsets mining is better than those found by existing techniques for membership queries from the point of view of space-time trade-off.**

Keywords-bitmap index; frequent itemsets mining

I. INTRODUCTION

The data warehouse (DW) is access through an Online Analytical Processing (OLAP) application [1]. The information stored in a DW is obtained from many different sources. Such sources might include Online Transaction Processing (OLTP) or legacy operational systems over a long period of time. Requests for information from DW usually are complex and ad-hoc queries. Such complex queries could take several hours or day to process because the queries have to search through a large amount of data.

There are many solutions to speed up query processing such as summary table, indices, and parallel machines. Indexing is the method to achieve without adding additional hardware [1]. There are many indexing techniques such as B^+-tree, R-tree, and bitmap index. Bitmap index is useful in processing complex queries in decision support systems [2,3,4]. It is simple to represent, uses less space and is more CPU-efficient than row ids when the number of distinct values of the indexed attribute is low. The index improves complex query performance by applying low-cost Boolean operations such as AND, OR and NOT in the selection predicate on multiple indices, thereby reducing the search space before going to the primary source data.

Simple Bitmap Index [2-8] was first introduced and implemented in the Model 204 DBMS. It consists of C (the number of distinct values of the indexed attribute) bitmap vectors each of which is created to represent each distinct value of indexed attribute. A bit i in a bitmap vector, representing value v, is set to 1 if the record i in the indexed table contain v. To answer a query, the bitmap vectors of the values specified in the predicate condition are read into memory. If there are more than one bitmap vectors read, Boolean operations will be performed on them before accessing data. Simple Bitmap Index is efficient in both space and query processing time for an attribute with a low cardinality. However, the sparsity problem occurs if the Simple Bitmap Index is built on high cardinality attribute, which then requires more space and query processing time to build and answer a query. To overcome this size problem, much of the research on bitmap indices has focused on reducing index size. Two following approaches have been developed: bitmap index compression [3,5,8], and bitmap index uncompression [2,4,6,7], aiming to reduce space requirements as well as improve query performance. In this paper, we focus on the bitmap index uncompression because Boolean operations on compressed bitmap index are often slower than on uncompressed bitmap index.

Encoded Bitmap Index [4] outperforms the best bitmap indices in term of space requirement. However, its response time for equality queries is the worst. This is because it needs to spend time finding the encoded representation of the attribute value v from the mapping table, comparing $\lceil \log_2 C \rceil$ bitmap vectors. Furthermore, to answer a membership queries of the form "X in $\{v_1, v_2, \ldots, v_n\}$", it has to find the encoded representation of each attribute value v_i and perform Boolean operation OR n-1 times, requiring more response time. Therefore, in order to minimize the number of bitmap vectors which need to be accessed in answering a membership query, a well-defined encoding scheme is an important key of Encoded Bitmap Index. In this paper, we apply a data mining technique call frequent itemsets mining to find the well-defined encoding scheme, leading to improve queries processing time.

The rest of the paper is organized as follow. In Section 2 we review Encoded Bitmap Index and frequent itemsets mining. In Section 3 we present the optimizing Encoded Bitmap Index using frequent itemsets mining. In Section 4 we discuss our comparative study of five bitmap indexing techniques (i.e., Simple, Interval, Scatter, Dual, and Encoded Bitmap Index) against Encoded Bitmap Index using frequent itemsets mining. Finally, in Section 5 we discuss conclusion.

II. RELATED WORK

A. Encoded Bitmap Index

An Encoded Bitmap Index [5] was introduced by Wu

978-0-7695-3504-3/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCEE.2008.150

IEEE computer society

Authorized licensed use limited to: Prince of Songkla University. Downloaded on January 13, 2009 at 03:26 from IEEE Xplore. Restrictions apply.

and Buchman in 1998. It consists of $\lceil \log_2 C \rceil$ bitmap vectors, says $\{ E_{\lceil \log_2 C \rceil - 1}, \ldots, E_2, E_1, E_0 \}$, and a mapping table. Each distinct value of the indexed attribute is encoded with $\lceil \log_2 C \rceil$ bits (as Huffman encoding). Each of which is stored in a bitmap vector. The mapping table stores the mapping between each attribute value and its encoded representation. Fig. 1 shows an example of the Encoded Bitmap Index on attribute X with C=8. The Index is the collection of 3 bitmap vectors, $\{ E_2, E_1, E_0 \}$, where 'a' is encoded as 000, 'b' as 001, and etc. For those tuples with X = 'a', we set corresponding positions in all bitmap vectors as $E_2 = 0$, $E_1 = 0$, and $E_0 = 0$; for those tuples with X = 'b', we set corresponding positions in all bitmap vectors as $E_2 = 0$, $E_1 = 0$, and $E_0 = 1$; and so on. To answer a membership query, the encoded representation of each attribute value is found. Then, the Boolean operation OR is performed on them to get the result. For example, by using the mapping table as shown in Fig. 1, if we have a query with the selection condition X in (a,d,f,g) and let $E_i \in \{ B_i, B_i' \}$; $B_i = 1$, $B_i' = 0$, the retrieval function to answer this query will be $B_2'B_1'B_0' + B_2'B_1B_0 + B_2B_1'B_0 + B_2B_1B_0'$, which cannot be further reduced. However, if we build a query with the selection condition X in (a,b,c,d), the retrieval functions will be $B_2'B_1'B_0' + B_2'B_1'B_0 + B_2'B_1B_0' + B_2'B_1B_0$, which can be reduced to B_2' (only one bitmap vector is needed to be accessed).

B. Frequent Itemsets Mining

Frequent itemsets mining is the most fundamental and essential problem in mining association rules. A association rule is an association relationship of items in transaction database. It consists of two phases. In the first phase, the set of all frequent itemsets that satisfy the minimum support is discovered. In the second phase, the strong association rules are generated from the frequent itemsets found in the first phase. In this paper, we apply frequent itemsets mining to find the set of frequent attribute values in the workload of queries.

There are many algorithms to find frequent itemsets. For example, Apriori [9] is the best known previous algorithm that uses the frequent itemsets on a current level to construct candidate itemsets on the next level and requires multiple database scans. FP-growth [10] is the well-known algorithm that uses the FP-tree data structure to create all frequent itemsets without generating candidate itemsets and requires two database scans. Closed algorithm [11] finds only the largest closed frequent itemsets. Index-BitTableFI [12] is the algorithm that uses BitTable structure to reduce the cost of candidate generations and to support counting. However, in this paper, we want to create 2^i-frequent itemsets, where i = 1, 2, …, $\lceil \log_2 C \rceil$ -1. Therefore, we propose the efficient algorithm to find frequent 2^i-itemsets, which is explained in Section 3.

RID	...	X	...
1		b	
2		c	
3		a	
4		h	
5		a	
6		g	
7		d	
8		d	
9		f	
10		e	

(a) Table T

E_2	E_1	E_0
0	0	1
0	1	0
0	0	0
1	1	1
0	0	0
1	1	0
0	1	1
0	1	1
1	0	1
1	0	0

Mapping table

a	000
b	001
c	010
d	011
e	100
f	101
g	110
h	111

(b) Encoded Bitmap Index

Figure 1. Example of Encoded Bitmap Index (C=8).

III. Optimizing Encoded Bitmap Index

In this section, we present the process to optimize Encoded Bitmap Index. This process consists of three phases: Attribute Values Extraction, FIM Processing, and Binary Encoding.

A. Attribute Values Extraction

In this phase, the attribute values are extracted from the workload of queries to create attribute value table. Fig. 2(a) shows an example of the workload and Fig. 2(b) shows the attribute value table, which contains values of attribute X extracted from the workload, where the number of distinct attribute values of attribute X is 16 (C=16), and $X \in$ {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p}.

B. FIM Processing

There are two steps in FIM Processing. In the first step, the EncodedBitmapFIM algorithm is presented to discover the frequent itemsets of the attribute values of the indexed attribute in queries. Then, frequent attribute values are ordered in the second step.

1) EncodedBitmapFIM Algorithm : Table I describes the notation used in the EncodedBitmapFIM algorithm and Fig. 3 shows the detail of this algorithm.

EncodedBitmapFIM algorithm consists of 27 steps. In steps 1-3, one database (i.e., attribute value table) scan is needed to create an initial BitMatrix table. The BitMatrix table is a R x N table, where R is the number of all queries in the workload and N is the number of the frequent 1-itemsets. Each column represents the items of the indexed attribute in query Q_i and each row represents a query Q_i having the items. BitMatrix[i,j] takes on bit 1 if the i^{th} query contains j^{th} item. Then the number of bit 1's of each item is counted to get its support value and sorted in ascending order. The initial BitMatrix table is built after removing the items that have the support value below minimum support. Fig. 4 shows the initial BitMatrix table, assuming that the minimum support is equal to 2. The number after ":" indicates the support value.

Steps 4-26 generates F_k, where k=2^i, i = 1, 2, …, $\lceil \log_2 C \rceil$ -1. Step 5 checks a number of items (N) in the BitMatrix table; if N ≥ k (where k is the size of F_k), the number of items in the BitMatrix table is enough to create

Authorized licensed use limited to: Prince of Songkla University. Downloaded on January 13, 2009 at 03:26 from IEEE Xplore. Restrictions apply.

$Q1$: select * from T where X in (a,c,e,g,o,h,j,k,p)
$Q2$: select * from T where X in (b,d,f,i)
$Q3$: select * from T where X in (a,c,e,g,o,h,j,k,m,n)
$Q4$: select * from T where X in (a,c,e,g,o,h,j,k)
$Q5$: select * from T where X in (b,d,f,i,m,n)

(a) Workload

Query	Attribute Values
$Q1$	a,c,e,g,o,h,j,k,p
$Q2$	b,d,f,i
$Q3$	a,c,e,g,o,h,j,k,m,n
$Q4$	a,c,e,g,o,h,j,k
$Q5$	b,d,f,i,m,n

(b) Attribute value table

Figure 2. Example of workload and values of attribute *X* extracted from the workload.

TABLE I.
NOTATION USED IN ENCODEDBITMAPFIM ALGORITHM

Item	Attribute value.
Itemsets	The set of attribute values.
pointer	Starting position, $0 \le$ pointer $< N$.
$Item_{pointer}$	Item at starting position
N	The number of item in BitMatrix table
Delegate. $Item_{pointer}$	The set of $Item_{pointer}$ and items that occurs at the same time with $Item_{pointer}$ and located after $Item_{pointer}$.
C_k	Candidate *k*-itemsets.
F_k	Frequent *k*-itemsets.
tail	All items after starting position except Delegate.

EncodedBitmapFIM Algoritm : Find frequent itemsets of attribute values

```
1.Scan attribute value table once to create initial BitMatrix table.
2.Count a number of bit 1's of each items to get its support value.
3.Sort support value (single itemsets) in ascending order and
  remove infrequent itemsets.
4.For (k = 2^(⌈log2 C⌉)−1 ; k ≥ 2; k = k/2)
5.   While (N ≥ k )
6.    pointer = N – k ;
7.       While (pointer ≥ 0)
8.          Delegate.item_pointer= Gen_Delegate(Item_pointer);
9.          If (a number of item in Delegate.item_pointer = k )
10.            F_k = F_k ∪ Delegate.item_pointer ;
11.            Remove Delegate.Item_pointer from BitMatrix table
12.            N = N-k;  Exit Loop;
13.         Else
14.            If ((N-pointer) > k and Support.item_pointer > MinSup)
15.               C_k = Gen_Candidate(Delegate.item_pointer ,tail)
16.               If (Support.C_k ≥ MinSup)
17.                  F_k = F_k ∪ C_k ;
18.                  Remove C_k from BitMatrix table;
19.                  N = N-k;  Exit Loop;
20.               End if
21.            End if
22.         End if
23.         pointer = pointer – 1;
24.      End while
25.  End while
26.End for
27.Return F_k and Support of each F_k
```

Figure 3. EncodedBitmapFIM Algorithm

F_k. Step 6 finds the starting position (i.e., pointer) among items to create F_k. Step 8 generates Delegate of $item_{pointer}$ by using Boolean operation AND on all bitmap vectors of queries containing $item_{pointer}$. For example, in Fig. 4, let $item_{pointer}$ be item 'a' which appears in $Q1$, $Q3$ and $Q4$. Starting at position item 'a' is located, from left to right, bitmap vector $Q1$ is 11111111, bitmap vector $Q3$ is 11111111, and bitmap vector $Q4$ is 11111111. After applying Boolean operation AND on these three bitmap vectors, the result is 11111111. The first bit 1's corresponds to item 'a', and the next bit 1's corresponds to items 'c', 'e', 'g', 'h', 'j', 'k' and 'o', respectively. Thus, the set of Delegate of item 'a' is {a,c,e,g,h,j,k,o}. Step 9 checks whether the number of items in Delegate is equal to *k* or not. If it is, Delegate is F_k. Then, all items in Delegate are removed from the BitMatrix table (step 10-12). If it is not and the condition in step 14 is true, C_k is generated in step 15 by Gen_Candidate function. The function generates C_k as described in [12]. After C_k is used to generate F_k in step 17, it is removed from the Bitmatrix table in step 18.

To show the EncodedBitmapFIM algorithm, we use the BitMatrix table in Fig. 4 to describe the steps 4-26, where C=16. In the first main loop, F_8= {a, c, e, g ,h ,j ,k ,o} with support=3 is generated. We then remove the items in F_8 from the BitMatrix table. Therefore, the remaining items in the BitMatrix table is {b,d,f,i,m,n}. In the second main loop, F_4={b,d,f,i} with support=2 is generated. After removing items in F_4 from the BitMatrix table, the remaining items in the BitMatrix table is {m,n}. The final main loop generates F_2 which is {m,n} with support=2. Thus, the result of all F_k is {a,c,e,g,h,j,k,o}:3, {b,d,f,i}:2, {m,n}:2 (the number after ":" indicate the support value).

The advantage of EncodedBitmapFIM algorithm requires only one database scan, and needs only the Boolean operation AND and COUNT. It reduce frequency-checking and generates a few candidate itemsets.

2) Ordering Attribute Values : After discovering the set of frequent attribute values, we line them in a descending order of frequent itemsets followed by the remaining items (i.e., infrequent attribute values). For example, the set of frequent attribute values, in the descending order, found in the first step is a, c, e, g, h, j, k, o, b, d, f, i, m, n. The infrequent attribute values are l and p. Therefore, the final result of this step is ordered as a, c, e, g, h, j, k, o, b, d, f, i, m, n, l, p.

C. Binary Encoding

In the last phase, we encode each of distinct attribute values using binary encoding. The result in this phase is a well-defined encoding scheme (i.e., mapping table) that leads to improve performance of response time in answering membership queries. By using the result of ordering attribute values in FIM Processing phase, the mapping table

Authorized licensed use limited to: Prince of Songkla University. Downloaded on January 13, 2009 at 03:26 from IEEE Xplore. Restrictions apply.

Query	Attribute value of X													
	b:2	**d:2**	**f:2**	**i:2**	**m:2**	**n:2**	**a:3**	**c:3**	**e:3**	**g:3**	**h:3**	**j:3**	**k:3**	**o:3**
$Q1$	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$Q2$	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$Q3$	0	0	0	0	1	1	1	1	1	1	1	1	1	1
$Q4$	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$Q5$	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 4. The initial BitMatrix table

is created. Fig. 5 shows the mapping table created by using frequent itemsets mining (Fig. 5(b)) comparing with the mapping table created without using frequent itemsets mining (Fig. 5(a)).

a	0000
b	0001
c	0010
d	0011
e	0100
f	0101
g	0110
h	0111
i	1000
j	1001
k	1010
l	1011
m	1100
n	1101
o	1110
p	1111

(a) the mapping table without using frequent itemsets mining.

a	0000
c	0001
e	0010
g	0011
h	0100
j	0101
k	0110
o	0111
b	1000
d	1001
f	1010
i	1011
m	1100
n	1101
l	1110
p	1111

(b) the mapping table using frequent itemsets mining.

Figure 5. The mapping table

IV. PERFORMANCE STUDY

This section presents the results of comparing the space-time performance of six bitmap indexing techniques (Simple, Interval, Scatter, Dual, Encoded, and our Proposed Encoded Bitmap Index). Table II shows space requirements, and a number of operations used to answer an equality query and membership query of each bitmap indexing technique. Encoded Bitmap Index (both traditional Encoded Bitmap Index and Encoded Bitmap Index using frequent itemsets mining) outperforms all other bitmap indices in term of space requirement and Simple Bitmap Index is worst. For answering equality query, Simple Bitmap Index takes 1 bitmap vector and no Boolean operation. Interval, Scatter and Dual Bitmap Index take 2 bitmap vectors and 2, 1, and 1 Boolean operation, respectively. Traditional Encoded Bitmap Index and Encoded Bitmap Index using frequent itemsets mining take $\lceil \log_2 C \rceil$ bitmap vectors and use mapping tables. For answering membership query, Simple, Interval, Scatter, Dual, and traditional Encoded Bitmap Index create a bitmap vector representing each distinct attribute value. Then the Boolean operation ORs is used to get the final result. Encoded Bitmap Index using frequent itemsets mining may take i bitmap vectors if the number of distinct attribute values in the membership query is $2^{\lceil \log_2 C \rceil - i}$, $i = 1, 2, \ldots, \lceil \log_2 C \rceil - 1$.

We choose query $Q2$ and $Q4$ in Fig. 2(a) to show the performance different between the traditional Encoded Bitmap Index and Encoded Bitmap Index using frequent itemsets mining. $Q2$ has the selection condition X in (b, d, f, i) and $Q4$ has the selection condition X in (a, c, e, g, o, h, j, k). By using the mapping table in Fig. 5(a), the retrieval function to answer $Q2$ and $Q4$ will be $B_3'B_2'B_0 + B_3'B_1'B_0 + B_3B_2'B_1'B_0'$ and $B_3'B_0' + B_3'B_2B_1B_0 + B_3B_2'B_1'B_0 + B_3B_2'B_1B_0' + B_3B_2B_1B_0'$, respectively. However, the retrieval function will be B_3B_2' for $Q2$ and will be B_3' for $Q4$, if we use the mapping table in Fig. 5(b).

V. CONCLUSIONS

The ability to extract data to answer complex, iterative, and ad-hoc queries quickly is a critical issue for data warehouse application. Various methods of bitmap indexing

TABLE II. A COMPARATIVE STUDY OF FIVE BITMAP INDICES

Bitmap Index	**Space**	**Time**	
	Number of bitmap vectors used to represent an attribute value	*Number of bitmap vectors scanned : Number of low-cost Boolean operations*	
		Equality query	*Membership query*
Simple [3]	C	1:0	k: k-1 (OR)
Interval [2]	$\left\lceil \frac{C}{2} \right\rceil$	2:2 (1AND,1NOT)	$2k$: $3k$-1 (k AND,k-1OR,k NOT)
Scatter [7]	$2\lceil \sqrt{C} \rceil$	2:1 (1AND)	$2k$: $2k$-1(k AND, k-1OR)
Dual [6]	$\lceil \sqrt{2C+0.25}+0.5 \rceil$	2:1 (1AND)	$2k$: $2k$-1(k AND, k-1OR)
Traditional Encoded [4]	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	$k\lceil \log_2 C \rceil$:use mapping table, k-1 OR
Encoded using FIM	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$:use mapping table	1:0

C is cardinality, k is a number of attribute values to answer in a membership query

Authorized licensed use limited to: Prince of Songkla University. Downloaded on January 13, 2009 at 03:26 from IEEE Xplore. Restrictions apply.

have proven time-efficient for answering data warehouse queries because they perform fast binary operation the index level before retrieving base data. This significantly improves query processing time. Several variant of bitmap indices have been introduced reduce storage requirement and speed up performance.

Encoded Bitmap Index (both traditional Encoded Bitmap Index and Encoded Bitmap Index using frequent itemsets mining) outperforms all other bitmap indices in term of space requirement. Therefore, it can be built on an attribute with high cardinality. An important key of Encoded Bitmap Index is a well-defined encoding scheme. In this paper, we apply a frequent itemsets mining to find frequent itemsets of indexed attribute values before encoding, leading to reduce bitmap vectors to be accessed. Our comparative study shows that, in the best case, the performance of Optimizing Encoded Bitmap Index using frequent itemsets mining is better than those found by existing techniques for a membership query from the point of view of space-time trade-off.

REFERENCES

[1] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD RECORD, Vol. 26, pp. 65-74, 1997.

[2] C. Y. Chan and Y. E. Ioannidis , "An Efficient Bitmap Encoding Scheme for Selection Queries", Proceedings of the 1999 ACM SIGMOD international conference on management of data, pp.215-226, 1999.

[3] K. Stockinger and K. Wu , "Bitmap Indices for Data Warehouse" ,In Data Warehouse and OLAP , 2007.

[4] M. Wu and A. Buchmann, ,"Encoded Bitmap Indexing for Data Warehouses", Proceeding of the Fourteenth International Conference on Data Engineering, pp.220-230,1998.

[5] E. O'Neil and P. O'Neil, "Bitmap Index Design Choices and Their Performance Implications" 11th International Database Engineering and Applications Symposium, pp.72-84,2007.

[6] N. Wattanakitrungroj, S. Vanichayabon, "Dual Bitmap Index: Space-Time Efficient Bitmap Index for Equality and Membership Queries", International Symposium on Communications and Information Technologies (ISCIT'06), pp.568-573, 2006.

[7] S. Vanichayabon, J. Manfuekphan and L. Gruenwald, "Scatter Bitmap: Space-time Efficient Bitmap Indexing for Equality and Membership Queries", Proceeding of IEEE International Conferences on cybernetics and Intelligent Systems , pp.1-6, 2006.

[8] K. Wu, E. J. Otoo and A. Shoshani, "Compressing Bitmap Indexes for Faster Search Operation", International Conference on Scientific and Statistical Database Management (SSDBM), 2002.

[9] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", Proceedings of the 1993 ACM SIGMOD international conference on Management of data SIGMOD, 1993.

[10] H. J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation", 2000 ACM SIGMOD Intl. Conference on management of data, pp.1-12, 2000.

[11] N. Pasquier, Y. Bastide, R. taouil and L. Lakhal, "Efficient mining of association rules using closed itemset lattices", Information System, Vol. 24, No. 1, pp. 25-46, 1999.

[12] W. Song, B. Yang and Z. Xu, "Index-BitTableFI:An improved algorithm for mining frequent itemsets", Knowledge-Based System, 2008.

Authorized licensed use limited to: Prince of Songkla University. Downloaded on January 13, 2009 at 03:26 from IEEE Xplore. Restrictions apply.

ประวัติผู้เขียน

ชื่อ สกุล นางสาวจรรยา สายนุ้ย

รหัสประจำตัวนักศึกษา 5010220020

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วท.บ. (วิทยาการคอมพิวเตอร์)	มหาวิทยาลัยสงขลานครินทร์	2548

ทุนการศึกษา (ที่ได้รับในระหว่างการศึกษา)

ทุนมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร พ.ศ. 2550 จากมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร

การตีพิมพ์เผยแพร่ผลงาน

จรรยา สายนุ้ย ศิริรัตน์ วณิชโยบล และนิวรรณ วัฒนกิจรุ่งโรจน์. 2551. การใช้กฎความสัมพันธ์เพื่อเพิ่มประสิทธิภาพของการทำดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามแบบสมาชิก. The 5th Joint Conference on Computer Science and Software Engineering (JCSSE 2008). กาญจนบุรี ประเทศไทย. หน้า 317-324.

จรรยา สายนุ้ย ศิริรัตน์ วณิชโยบล และนิวรรณ วัฒนกิจรุ่งโรจน์. 2551. การเพิ่มประสิทธิภาพดัชนีบิตแมปแบบเข้ารหัสสำหรับการสอบถามข้อมูลแบบสมาชิกโดยใช้เทคนิคการหากลุ่มข้อมูลที่ปรากฏบ่อย. The 12th Computer Science and Engineering Conference (NCSEC 2008). ชลบุรี ประเทศไทย. หน้า 560-567.

J. Sainui, S. Vanichayobon and N. Wattanakitrungroj. 2008. Optimizing Encoded Bitmap Index using Frequent Itemsets Mining. Proceeding of 2008 International Conference on Computer and Electrical Engineering (ICCEE 2008). Phuket Thailand. pp. 511-515.