การหาคำตอบที่เหมาะสมของคำถามสอบถามจากฐานข้อมูลโดยใช้เมตาเดตาเซอร์ช

DETERMINING OPTIMAL SOLUTIONS FOR NATURAL LANGUAGE DATABASE QUERY USING METADATA SEARCH

ภิทูร วิชกูล
วิบูลย์ เจียมทับทักษิณ

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2546

DETERMINING OPTIMAL SOLUTIONS FOR NATURAL LANGUAGE DATABASE QUERY USING METADATA SEARCH

PITOON WICHGOOL

WIBOON JIAMTHUBTHUGSIN

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHLOR OF SCIENCE

DEPARTMENT OF MATHEMETICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2003

หัวข้อปัญหาพิเศษ	การหาคำตอบที่เหมาะสมของคำถามสอบถามจากฐานข้อมูลโดยใช้เมตาเดตาเซอร์ช DETERMINING OPTIMAL SOLUTIONS FOR NATURAL LANGUAGE DATABASE QUERY USING METADATA SEARCH	
ชื่อนักศึกษา	นายภิทูร วิชกูล	43050398
	นายวิบูลย์ เจียมทับทักษิณ	43050412
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
อาจารย์ที่ปรึกษา	ผศ.ดร.วีระ บุญจริง	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2546

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	ดร.นันทิกา เบญจเทพานันท์	
กรรมการ	อ.สิริลักษณ์ อนันต์สถิตย์สิน	
กรรมการและอาจารย์ที่ปรึกษา	ผศ.ดร.วีระ บุญจริง	

(ผู้ช่วยศาสตราจารย์ ดร.วีระ บุญจริง)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

ลิขสิทธิ์ของภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อปัญหาพิเศษ	การหาคำตอบที่เหมาะสมของคำถามสอบถามจากฐานข้อมูลโดยใช้เมตาเดตาเซอร์ช	
ชื่อนักศึกษา	นายภิทูร วิชกูล	43050398
	นายวิบูลย์ เจียมทับทักษิณ	43050412
ปริญญา	วิทยาศาสตรบัณฑิต	
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2546	
อาจารย์ที่ปรึกษา	ผศ.ดร.วีระ บุญจริง	

บทคัดย่อ

โครงงานนี้มีวัตถุประสงค์เพื่อพัฒนาอัลกอริทึมบรานช์แอนด์บาวน์เพื่อใช้ในการหาคำตอบที่เหมาะสมสำหรับคำถามสอบถามข้อมูลจากฐานข้อมูล อัลกอริทึมทำการค้นหาคำสำคัญจากข้อความคำถาม คำสำคัญเป็นคำต่างๆ ที่ตรงกับโหนดของกราฟเมตาเดตา โหนดของกราฟนี้ประกอบด้วยค่าข้อมูลจากฐานข้อมูล แอตทริบิว รีเลชั่นและ คำที่ผู้ใช้นิยมใช้เรียกโหนดเหล่านี้ เนื่องจากคำสำคัญแต่ละคำในข้อความคำถามอาจชี้ไปยังโหนดหลายโหนดของกราฟ กราฟย่อยที่ใช้แทนคำตอบต่อคำถามฐานข้อมูลหนึ่งจึงมีหลายกราฟย่อย กราฟย่อยที่ดีที่สุดคือกราฟย่อยที่มีเส้นต่อน้อยที่สุด โครงงานเสนออัลกอริทึมในการหากราฟย่อยที่ดีที่สุดต่อคำถามสอบถามข้อมูลโดยไม่ต้องทำการแจกแจงและประเมินกราฟย่อยที่เป็นไปได้ทั้งหมด

Special Project Title	DETERMINING OPTIMAL SOLUTIONS FOR NATURAL LANGUAGE DATABASE QUERY USING METADATA SEARCH	
Students	Mr.Pitoon Wichgool	43050398
	Mr.Wiboon Jiamthubthugsin	43050412
Degree	Bachelor of Science	
Department	Mathematics and Computer Sciences, Faculty of Science	
Programme	Computer Sciences	
Academic Year	2002	
Special Project Advisor	Assoc.Prof. Dr. Veera Boonjing	

ABSTRACT

The purpose of the project is to develop the branch-and-bound algorithm to determine optimal solutions for natural language database queries. The algorithm extracts keywords from text inputs. Keywords are words matched with nodes of metadata graph. Nodes of the graph consist of database values, database attributes, database relations, and their associated user-words. Vertices represent relationships among these nodes. Since each extracted keyword could correspond to many nodes, there could be a number of subgraphs corresponding to a query. The best subgraph is the graph with minimal connections. The project proposes the algorithm to determine such the best solution for the query without exhaustive enumeration-evaluation.

กิตติกรรมประกาศ

ในการจัดทำปัญหาพิเศษเรื่องการหาคำตอบที่เหมาะสมของคำถามสอบถามจากฐานข้อมูลโดยใช้เมตาเดตาเชอร์ชสามารถสำเร็จลุล่วงไปได้ด้วยดี คณะผู้จัดทำขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ดอกเตอร์วีระ บุญจริง อาจารย์ผู้รับผิดชอบปัญหาพิเศษฉบับนี้ที่กรุณาให้คำแนะนำและเป็นที่ปรึกษาในการแก้ปัญหาต่างๆ ที่เกิดขึ้นในระหว่างการทำปัญหาพิเศษ และต้องขอขอบพระคุณคณะกรรมการทุกท่าน ที่ได้สละเวลาอันมีค่ามาเป็นผู้ตรวจสอบความถูกต้อง ของปัญหาพิเศษฉบับนี้

นอกจากนี้คณะผู้จัดทำต้องขอขอบพระคุณบิดา มารดา ที่ได้ให้ความสนับสนุนทางด้านกำลังใจและทุนทรัพย์ จนการทำปัญหาพิเศษครั้งนี้สำเร็จได้ด้วยดีรวมทั้งพี่ๆ เพื่อนๆ และน้องๆ ทุกคน ที่ได้ให้ความช่วยเหลือในการทดสอบโปรแกรม และความช่วยเหลือในด้านต่างๆ เกี่ยวกับปัญหาพิเศษไว้ ณ ที่นี้

คณะผู้จัดทำ

มีนาคม 2547

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบัน มีความต้องการในการถามข้อมูลจากฐานข้อมูลด้วยข้อความอิสระ เช่นเดียวกับการใช้ข้อความเพื่อทำการสืบค้นข้อมูลจากอินเตอร์เน็ต ระบบที่จะสามารถสนองความต้องการดังกล่าวนี้จะต้องมีส่วนประกอบหนึ่งที่ใช้ในการหาคำตอบจากคำสำคัญต่าง ๆ ที่หาได้จากข้อความคำถาม โดยคำสำคัญจะเป็นโหนดของกราฟความหมายของข้อมูลในฐานข้อมูลและเป็นคำที่ผู้ใช้นิยมใช้เรียกโหนดจากกราฟความหมาย ดังนั้นคำสำคัญแต่ละคำอาจชี้ไปที่โหนดหลายโหนดในกราฟความหมาย ทำให้คำตอบ (กราฟย่อย) ของคำถามข้อมูลจากฐานข้อมูลหนึ่งสามารถเป็นไปได้หลายคำตอบ ดังนั้น โครงงานนี้จึงเสนอที่จะหาคำตอบที่ดีที่สุดของข้อความคำถาม

1.2 วัตถุประสงค์

โครงงานนี้มีวัตถุประสงค์เพื่อพัฒนาอัลกอริทึมสำหรับการหาคำตอบที่ดีที่สุดของคำถามข้อมูลจากฐานข้อมูล โดยจะทำการศึกษาและเลือกอัลกอริทึมที่เหมาะสม ทำการพัฒนาระบบ และทดลองกับฐานข้อมูลขนาดเล็ก

1.3 ขอบเขต

โครงงานนี้ใช้แบบจำลองสำหรับการจัดเก็บความหมายของข้อมูลในฐานข้อมูล โดยแบบจำลองนี้จะจัดเก็บข้อมูลสามประเภท คือ โครงสร้างข้อมูล ค่าข้อมูล และคำพ้องรูป โดยข้อมูลทั้งสามประเภทนี้จะเชื่อมโยงกันอยู่ในรูปกราฟ จากนั้นโครงงานนี้จะนำคำสำคัญต่าง ๆ จากข้อความคำถามไปตั้งปัญหาเป็นปัญหาการหาค่าที่เหมาะสมที่สุดภายใต้กราฟความหมาย แล้วใช้อัลกอริทึมการค้นหาคำตอบที่ดีที่สุด

1.4 ขั้นตอนการดำเนินงาน

1. ออกแบบโครงสร้างแบบจำลองการจัดเก็บความหมายของข้อมูลในฐานข้อมูล
2. พัฒนาฐานข้อมูลตัวอย่างและแบบจำลองการจัดเก็บความหมาย
3. ศึกษาวิธีการค้นหาแบบต่าง ๆ เลือกวิธีการค้นหาและปรับแต่งวิธีที่เลือกสำหรับโครงงานนี้
4. พัฒนาโปรแกรมตามวิธีที่เลือก

1.5 ประโยชน์ที่คาดว่าจะได้รับ

โปรแกรมที่ได้จะเป็นส่วนหนึ่งของโปรแกรมที่ใช้อำนวยความสะดวกในการสอบถามข้อมูลจากฐานข้อมูลโดยใช้ข้อความอิสระ ซึ่งระบบการสอบถามข้อมูลจากฐานข้อมูลโดยใช้ข้อความอิสระนี้เหมาะที่จะนำไปเชื่อมกับระบบสอบถามข้อมูลด้วยเสียงพูด เนื่องจากไม่มีข้อจำกัดด้านไวยากรณ์ของภาษาที่ใช้

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

บทนี้เป็นการนำเสนอเทคนิคการแก้ปัญหาโดยการค้นหาซึ่งเป็นเทคนิคที่ใช้ทั่วไปในวงการปัญญาประดิษฐ์ ตอนที่ 2.1 และ 2.2 เป็นการแนะนำหลักการพื้นฐานของการค้นหา ตอนที่ 2.3 นำเสนออัลกอริทึมการค้นหาแบบต่างๆ

2.1 การค้นหา (Search)

หลักการพื้นฐานของ การค้นหา คือ ถ้าเรารู้การดำเนินการทั้งหมดที่สามารถทำได้เพื่อแก้ปัญหา แต่ไม่สามารถบอกได้ว่าการดำเนินการใดที่จะให้คำตอบ เราสามารถที่จะค้นหาจากความเป็นไปได้ทั้งหมดเพื่อให้ได้มาซึ่งคำตอบ ซึ่งมีหลายเทคนิคที่สามารถนำมาประยุกต์เพื่อหาคำตอบได้ เทคนิคหลักสองเทคนิคได้แก่ การสร้างแล้วทดลอง (Generate and Test) และการค้นหาโดยการสุ่ม (Random Search)

การสร้างและทดสอบ

เป็นวิธีที่ไม่สลับซับซ้อนมีขั้นตอนคือ

1. สร้างคำตอบที่เป็นไปได้
2. ตรวจสอบว่าเป็นคำตอบที่ถูกต้องหรือไม่
   (a) ถ้าใช่หยุด
   (b) ถ้าไม่ใช่ กลับไปทำข้อ 1. ใหม่

ตัวอย่างที่ชัดเจนของวิธีนี้ก็คือโปรแกรมที่พยายามจะเจาะเข้ามาในระบบของคุณ โดยการสร้างรหัสให้มากที่สุดจนกระทั่งสามารถเข้าระบบได้หรือถูกตรวจจับได้

วิธีการนี้ไม่เหมาะสมกับปัญหาที่มีคำตอบจำนวนมากหรือการที่จะสร้างคำตอบที่เป็นไปได้นั้นสิ้นเปลือง มีอันตรายหรือสิ้นเปลืองเวลา

หากเราสามารถทำให้แน่ใจได้ว่าคำตอบเดียวกันจะไม่ถูกสร้างซ้ำมากกว่าหนึ่งครั้งจะทำให้วิธีการนี้มีประสิทธิภาพมากขึ้น ซึ่งเป็นการง่ายหากคำตอบนั้นสามารถเรียงเป็นลำดับได้ แต่หากไม่ได้ก็สามารถที่จะทำการเก็บรายการคำตอบที่เราได้ทดลองไปแล้วไว้ แต่นั่นก็จะทำให้เกิดปัญหา

ในกรณีที่รายการคำตอบมีจำนวนมากซึ่งจะทำให้การตรวจสอบนั้นใช้เวลามาก ในกรณีนี้การยอมเสี่ยงที่จะสร้างคำตอบโดยไม่ตรวจสอบนั้นดูเหมาะสมกว่าที่จะเสียเวลาในการตรวจสอบ

การค้นหาโดยการสุ่ม

เป็นการแก้ปัญหาโดยใช้หลักการ ปริภูมิปัญหา (Problem Space) เราจะสันนิษฐานว่าโปรแกรมสามารถเก็บสถานะที่ได้ประสบไปแล้วในการแก้ปัญหาในรูปของกลุ่มของสถานะได้ โดยเทคนิคนี้มีขั้นตอนดังนี้

1. เริ่มที่สถานะเริ่มต้น
2. ทำซ้ำ
   2.a สุ่มเลือกตัวดำเนินการ
   2.b ถ้าตัวดำเนินการไม่สามารถใช้ประโยชน์ก่อให้เกิดสถานะล้มเหลว ทำต่อโดยใช้สถานะก่อนหน้าที่ขั้นตอน 2.a
   2.c ถ้าได้ผลเป็นสถานะเป้าหมาย หยุด
   2.d ทำต่อโดยใช้สถานะใหม่ที่ขั้นตอน 2.a

การค้นหาโดยการสุ่มมีปัญหาหลักๆ คือ

1. ถ้าโปรแกรมเกิดพิจารณาสถานะเดิมสองครั้ง มันอาจใช้ตัวดำเนินการเหมือนเดิมและกลับไปที่สถานะเดิมอีกครั้งซึ่งอาจทำให้ไม่ได้คำตอบทั้งหมด
สามารถหลีกเลี่ยงได้โดยทำการเก็บเส้นทางของสถานะว่าสถานะใดได้ดำเนินการแล้ว
2. มันไม่สามารถรับรองได้ว่าจะได้คำตอบที่ถูกต้องไม่ว่าจะทำงานนานขนาดไหนก็ตาม
3. เราไม่สามารถทราบได้ว่าเราจะได้คำตอบในเวลาที่เราต้องการต่อให้เรารู้ว่ามีคำตอบแน่นอน

2.2 ปริภูมิการค้นหา

ปริภูมิการค้นหา (Search Space) เป็นคำที่ใช้เมื่อใช้อัลกอริทึมการค้นหาในการแก้ปัญหาโดยมากมักจะแสดงโดยสถานะเฉพาะตัวซึ่งเป็นผลจากสถานะก่อนหน้านั้นใช้ตัวดำเนินการ

ปริภูมิการค้นหาในแต่ละปัญหามักมีจำนวนสถานะมากเสมอ ในกรณีที่มีตัวดำเนินการ 2 ตัวในแต่ละสถานะในปริภูมิ จำนวนสถานะในแต่ละระดับจะเป็น 2^n เมื่อ n คือระดับชั้นของแผนภูมิต้นไม้ (Tree) ซึ่งทำให้ว่าเพียงแค่การแก้ปัญหาทั่วไปยังมีความยากขนาดนี้

ในกรณีที่ปัญหามีตัวดำเนินการมากกว่านี้จำนวนสถานะก็จะมากขึ้น การเพิ่มขึ้นแบบเอ็กซ์โปเนนเชียล (Exponential) นี้เป็นปัญหาในการออกแบบโปรแกรมการค้นหา เนื่องจากปัญญาประดิษฐ์ (AI) เกี่ยวข้องกับการค้นหา ทำให้หลายๆเทคนิคของการเขียนโปรแกรมปัญญาประดิษฐ์เกี่ยวข้องกับการจัดการกับปริภูมิการค้นหาขนาดมหาศาล

2.3 อัลกอริทึมการค้นหา

อัลกอริทึมในการค้นหาที่เราสนใจนั้นสามารถแบ่งออกได้เป็น 2 ประเภทคือ การค้นหาแบบบรูทฟอร์จ (Brute-force Search) และ การค้นหาแบบฮิวริสติก (Heuristic Search) ซึ่งมีรายละเอียดดังต่อไปนี้ (การอธิบายอัลกอริทึมต่างๆจะมีศัพท์คำว่า สร้าง (generate) และ แผ่ (expand) โดยสร้างโหนดจะหมายถึงสร้างโครงสร้างข้อมูลที่สอดคล้องกับโหนดนั้น ส่วนการแผ่โหนดหมายความว่าสร้างโหนดลูกทั้งหมดของโหนดนั้น)

2.3.1 การค้นหาแบบบรูทฟอร์จ

การค้นหาแบบบรูทฟอร์จเป็นการแก้ปัญหาอย่างตรงไปตรงมา ต้องการเพียงคำอธิบายสถานะ, กลุ่มตัวดำเนินการที่ยอมรับได้, สถานะเริ่มต้น, สถานะเป้าหมาย ไม่จำเป็นต้องทราบลักษณะเฉพาะใดๆ ของขอบเขตปัญหา (Domain) อัลกอริทึมการค้นหาทั่วไปส่วนมากมักเป็นประเภทนี้ เทคนิคสำคัญของ การค้นหาแบบบรูทฟอร์จ คือการค้นหาแบบแนวกว้างก่อน (breadth-first), แบบยูนิฟอร์ม (uniform-cost), แบบแนวลึกก่อน (depth-first), แบบแนวลึกก่อนด้วยการทำซ้ำ : ดีเอฟไอดี (depth-first iterative-deepening : DFID) และ แบบสองทิศทาง (bidirectional)

การค้นหาแบบแนวกว้างก่อน มีวิธีการคือจะ แผ่โหนดตามลำดับของระยะทางจากราก โดยจะทำการสร้างทีละระดับชั้นของ แผนภูมิต้นไม้ จนพบคำตอบการสร้างแผนภูมิต้นไม้จะใช้รูปแบบคิว (queue) ของโหนด โดยเริ่มต้นจากประกอบไปด้วยรากเพียงอย่างเดียว และนำโหนดออกจากด้านหัวของคิวเท่านั้นและการเพิ่มโหนดลูกจะเพิ่มไปยังท้ายคิว ถ้าหากเส้นทางทั้งหมดในการค้นหาแบบแนวกว้างก่อน ไม่ได้มีค่าต้นทุน (cost) เดียวกัน จะกลายเป็นแบบยูนิฟอร์ม ซึ่งจะแผ่โหนดเป็นลำดับของ ค่าจากรากแต่ละขั้นตอนที่จะทำคือ จะแผ่โหนด (แทนด้วย n) ที่ต้นทุนของโหนดนั้น (แทนด้วย g(n)) มีค่าน้อยที่สุด โดย g(n) คือ ผลรวมทั้งหมดของต้นทุนของเส้นจากรากไปยังโหนด n โดยโหนดจะถูกเก็บไว้ในคิว อัลกอริทึมของการค้นหาแบบนี้ที่เป็นที่รู้จักกันดี เช่น อัลกอริทึมการหาเส้นทางที่สั้นที่สุดของดิสค์ตรา (Dijkstra's single-source shortest-path)

ข้อเสียที่สำคัญของวิธีแนวกว้างก่อน และยูนิฟอร์มคล้ายกันคือต้องการหน่วยความจำมาก เพราะต้องบันทึกแต่ละระดับของแผนภูมิต้นไม้เพื่อที่จะใช้สร้างในระดับถัดไป จำนวนของหน่วยความจำที่ใช้จะเป็นจำนวนของโหนดที่ถูกเก็บ และการค้นหาแบบแนวกว้างก่อนถูกจำกัดด้วยเนื้อที่ว่างอย่างมากในทางปฏิบัติ ในขณะที่การค้นหาแบบแบบแนวลึกก่อนเป็นตัวช่วยแก้ปัญหาความจำกัดเรื่องเนื้อที่ของการค้นหาแบบแนวกว้างก่อน โดยมีวิธีการคือ จะทำการสร้างโหนดลูกตัวถัดไปที่อยู่ลึกที่สุดที่ยังไม่ถูกแผ่วิธีการนี้จะใช้สิสต์ (list) ของโหนดที่ยังไม่ถูกแผ่ แต่มีข้อแตกต่างกับการค้นหาแบบแนวกว้างก่อน คือ การค้นหาแบบแบบแนวลึกก่อนจะจัดการลิสต์เป็นลักษณะของคิวแบบเข้าก่อนออกก่อน (first-in first-out) ในขณะที่การค้นหาแบบแนวกว้างก่อน จะจัดการลิสต์ในลักษณะแบบสแตค (stack) แบบเข้าหลังออกก่อน (last-in first-out)

ข้อดีของการค้นหาแบบแบบแนวลึกก่อนคือต้องการเนื้อที่เป็นแบบเชิงเส้นซึ่งคือ ความลึก (depth) ในการค้นหาไม่ใช่แบบเอ็กซ์โปเนนเชียลเหมือนกับของการค้นหาแบบแบบแนวลึกก่อน เนื่องจากอัลกอริทึมนี้เก็บสแตคของโหนดที่อยู่บนเส้นทางจากรากไปยังโหนด ณ ปัจจุบัน การค้นหาแบบแบบแนวลึกก่อนนี้จะสร้างโหนดกลุ่มเดียวกันกับกลุ่มของโหนดที่ถูกสร้างโดยการค้นหาแบบแนวกว้างก่อน ทำให้ในทางปฏิบัติการค้นหาแบบแบบแนวลึกก่อนจะถูกจำกัดด้วยเวลาที่ใช้มากกว่าเรื่องเนื้อที่ นอกจากนี้การค้นหาแบบแบบแนวลึกก่อนอาจทำงานไม่รู้จบในลักษณะเป็นแผนภูมิต้นไม้แบบไม่สิ้นสุด

สำหรับกราฟที่มีวงการค้นหาแบบแนวกว้างก่อน จะมีประสิทธิภาพมากกว่าแบบแบบแนวลึกก่อนเนื่องจากการค้นหาแบบแนวกว้างก่อน สามารถตรวจสอบการซ้ำกันของโหนดได้ในขณะที่แบบแบบแนวลึกก่อน ไม่สามารถทำได้ ด้วยเหตุนี้ความซับซ้อนของแบบแนวกว้างก่อน จะมากขึ้นตามจำนวนโหนดที่ความลึกนั้นเท่านั้น ขณะที่ความซับซ้อนของการค้นหาแบบแบบแนวลึกก่อนจะขึ้นอยู่กับจำนวนเส้นทางของความยาวหนึ่ง ๆ

ในขณะที่การค้นหาแบบแนวลึกก่อนด้วยการทำซ้ำนั้นจะเป็นการรวมคุณสมบัติที่ดีของการค้นหาแบบแนวกว้างก่อน และแบบแนวลึกก่อนเข้าไว้ด้วยกัน วิธีการแบบแนวลึกก่อนด้วยการทำซ้ำ คือ จะกระทำการค้นหาแบบแบบแนวลึกก่อนสำหรับความลึกที่ 1 แล้วทำแบบการค้นหาแนวลึกก่อนสำหรับความลึกที่ 2 และทำการค้นหาแบบแบบแนวลึกก่อนเช่นนี้ไปเรื่อย ๆ สำหรับความลึกต่อ ๆ มา จนพบคำตอบ เนื่องจากจะไม่สร้างโหนดต่อไปจนกว่าโหนดอื่นที่อยู่ระดับตื้นกว่าจะถูกสร้างทำให้สามารถรับรองคำตอบแรกที่พบจะสามารถเป็นตามแบบเส้นทางที่สั้นที่สุด นอกจากนี้จุดใด ๆ ที่ถูกประมวลผลโดยการการค้นหาแบบแบบแนวลึกก่อนจะเก็บเพียงสแตคของโหนดไว้เท่านั้นถึงแม้ว่าการค้นหาแบบแนวลึกก่อนด้วยการทำซ้ำจะเสียเวลาในลักษณะการทำซ้ำ

จนกว่าจะพบคำตอบ แต่การทำซ้ำที่เพิ่มเข้ามานี้ก็ไม่ได้มีผลมากนักซึ่งทำให้การค้นหาแบบแนวลึกก่อนด้วยการทำซ้ำมีความเหมาะสมทั้งในแง่ของเวลาและการใช้เนื้อที่เมื่อเทียบกับอัลกอริทึมการหาเส้นทางที่สั้นที่สุดแบบบรูทฟอร์จบนแผนภูมิต้นไม้

นอกจากนี้การค้นหาแบบบรูทฟอร์จยังมีการค้นหาแบบสองทิศทางซึ่งเป็นอัลกอริทึมที่ต้องรู้สภาพเป้าหมายที่แน่นอนแทนที่จะเป็นการทดสอบเงื่อนไขของเป้าหมาย มีหลักการคือทำการค้นหาแบบไปข้างหน้าจากสภาพเริ่มต้นไปพร้อม ๆ กับทำการค้นหาแบบถอยกลับจากสภาพเป้าหมาย ทำจนกระทั่งการค้นหาทั้ง 2 ทิศทางนี้มาเจอกัน เส้นทางของคำตอบที่สมบูรณ์จะเป็นเส้นทางจากสภาพเริ่มต้นรวมกับเส้นทางย้อนกลับจากสภาพเป้าหมาย

2.3.2 การค้นหาแบบฮิวริสติก

เป็นการพัฒนาวิธีการค้นหาโดยใช้ในปัญหาที่มีขอบเขตใหญ่ โดยนำความรู้เกี่ยวกับลักษณะของขอบเขตซึ่งจะสามารถนำมาช่วยเพิ่มประสิทธิภาพในการค้นหา ในปัญญาประดิษฐ์ การค้นหาแบบฮิวริสติกมี2 ความหมายคือในเชิงทั่ว ๆ ไป และในเชิงศึกษาเฉพาะเรื่อง ในความรู้สึกทั่ว ๆ ไป คำว่าฮิวริสติกถูกใช้เพื่อให้คำแนะนำที่มีประสิทธิภาพแต่ไม่ได้รับรองว่าจะทำได้ทุกกรณี ในสิ่งตีพิมพ์ต่าง ๆ ที่เกี่ยวกับการค้นหาแบบฮิวริสติกคำว่าฮิวริสติกจะมีเกี่ยวข้องกับกรณีพิเศษของฟังก์ชั่นประมาณค่าแบบฮิวริสติกซึ่งเป็นส่วนสำคัญของการค้นหาแบบฮิวริสติกอัลกอริทึมของการค้นหาแบบฮิวริสติกเช่น การค้นหาแบบเพียวฮิวริสติก (Pure Heuristic Search), เอ-สตาร์ (A*), บรานช์แอนด์บาวน์แบบแนวลึกก่อน : ดีเอฟบีเอ็นบี (Depth-First Branch-and-Bound :DFBnB) และวิธีอื่นๆแล้วแต่ต้องใช้ฟังก์ชั่นการประมาณค่าแบบฮิวริสติกทั้งสิ้น

ฟังก์ชั่นประมาณค่าแบบฮิวริสติก(Heuristic Evaluation Functions)

ในปัญหาการหาเส้นทางของพนักงานขาย (single-agent path-finding) จะใช้ฟังก์ชั่นประมาณค่าแบบ ฮิวริสติกในการประมาณค่าต้นทุนของเส้นทางที่เหมาะสมระหว่างสภาพสองสภาพ เช่น ระยะทางแบบยุคลิด (Euclidean) หรือที่เรียกว่าระยะทางของการบิน จะเป็นตัวประมาณของระยะทางการสัญจรระหว่างที่สองที่ และฮิวริสติกฟังก์ชั่นของปริศนาแผ่นเลื่อน (sliding-tile puzzle) เช่น ระยะทางแบบแมนฮัตตัน (Manhattan) ถูกคำนวณโดยการนับจำนวนของการเคลื่อนแผ่นตามแนวตารางจากสภาพเป้าหมายและทำการรวมค่านี้ของแต่ละแผ่นเข้าด้วยกัน สำหรับสภาพเป้าหมายคงที่ฮิวริสติกฟังก์ชั่น จะเป็นฟังก์ชั่นของโหนด แทนด้วย h(n) ซึ่งใช้ในการประมาณค่าระยะทางจากโหนด n ไปยังสภาพเป้าหมาย

กุญแจสำคัญของฟังก์ชั่นประมาณค่าแบบฮิวริสติกคือการประมาณค่าของต้นทุนจริงและสามารถคำนวณได้ง่าย เช่นระยะทางแบบยุคลิดระหว่างจุด 2 จุดสามารถถูกคำนวณได้ในระยะเวลาที่คงที่ ระยะทางแบบแมนฮัตตันระหว่างสภาพ 2 ตัวสามารถถูกคำนวณได้เป็นสัดส่วนของเวลาต่อจำนวนของแผ่น นอกจากนี้ ลักษณะของฮิวริสติกฟังก์ชั่นส่วนใหญ่จะเป็นการยืดหยุ่นของปัญหาต้นแบบ คือ เป็นขอบเขตล่างของต้นทุนจริง ค่าจะมีลักษณะที่สามารถยอมรับได้ เช่น ระยะทางของการบินเป็นขอบเขตล่างของระยะทางบนถนนระหว่าง 2 จุด เนื่องจากเส้นทางที่สั้นที่สุดของจุด 2 จุดคือเส้นตรง ระยะทางแบบแมนฮัตตัน คือขอบเขตล่างของค่าจริงของจำนวนการเคลื่อนที่จำนวนเพื่อที่จะหาคำตอบของปริศนาแผ่นเลื่อน เนื่องจากทุก ๆ แผ่นจะต้องถูกเคลื่อนอย่างน้อยที่สุดเป็นจำนวนครั้งของระยะในหน่วยตารางจากตำแหน่งเป้าหมาย

อัลกอริทึมพื้นฐานที่สุดของการค้นหาแบบฮิวริสติกคือการค้นหาแบบเพียวฮิวริสติกซึ่งมีหลักการคือจะทำการแผ่โหนดตามลำดับของค่าของฮิวริสติกฟังก์ชั่นของโหนด โดยจะมีการจัดการลิสต์แบบปิดของโหนดที่ได้ถูกแผ่แล้ว และ ลิสต์แบบเปิดของโหนดที่ได้ถูกสร้างแล้วแต่ยังไม่ถูกแผ่ วิธีการคือ เริ่มด้วยสภาพเริ่มต้นในลิสต์แบบเปิด, ในแต่ละรอบโหนดหนึ่งโหนดในลิสต์แบบเปิดที่มีค่า h(n) น้อยที่สุดจะถูกแผ่แล้วทำการสร้างโหนดลูกของมันทั้งหมดแล้วใส่ในลิสต์แบบปิด หลังจากนั้นโหนดลูกก็จะมีการใช้ฮิวริสติกฟังก์ชั่นนี้เช่นเดียวกัน โดยโหนดลูกเหล่านี้จะถูกใส่ในลิสต์แบบเปิดตามลำดับของค่าฮิวริสติกของมัน ขั้นตอนนี้จะทำเช่นนี้ไปเรื่อย ๆ จนการแผ่ไปถึงสภาพเป้าหมาย

ในกราฟที่มีวงเส้นทางที่ไปยังโหนด ๆ หนึ่งอาจจะมีได้หลายเส้นทางและเส้นทางแรกที่พบอาจจะไม่ใช่เส้นทางที่สั้นที่สุดก็ได้ เมื่อเส้นทางที่สั้นกว่าถูกพบเป็นโหนดแบบเปิดก็จะทำการเก็บเส้นทางนั้นไว้และเส้นทางเดิมที่ยาวกว่าก็จะถูกยกเลิกไป แต่เส้นทางที่สั้นกว่าถูกพบเป็นโหนดแบบปิดแล้วโหนดนั้นก็จะเป็นแบบเปิดแล้วเส้นทางที่สั้นกว่าก็จะถูกโยงกับมันแทน อุปสรรคที่สำคัญของการค้นหาแบบเพียวฮิวริสติก คือ ไม่สนใจต้นทุนของเส้นทางที่ไปยังโหนด n ทำให้อาจไม่ได้คำตอบที่เหมาะสม

ขณะที่อัลกอริทึมเช่น เอ-สตาร์นั้นจะรวมคุณสมบัติของการค้นหาแบบยูนิฟอร์มเข้ากับเพียวฮิวริสติก โดยทั้งเพียวฮิวริสติกและเอ-สตาร์ต่างก็เป็นรูปแบบของการค้นหาที่เรียกว่าเบสท์เฟริสท์ (best-first)

โดยที่ลักษณะของเบสท์เฟริสท์ คือในแต่ละรอบโหนดที่ดีที่สุดที่จะถูกแผ่จะใช้ฟังก์ชั่นของ ต้นทุนในการพิจารณาซึ่งอัลกอริทึมแต่ละตัวจะแตกต่างกันเพียงฟังก์ชั่นต้นทุนเท่านั้น โดย

ต้นทุน ของเพียวฮิวริสติก คือ h(n) และของเอ-สตาร์ คือ f(n)=g(n)+h(n) โดยที่ g(n) คือต้นทุนของเส้นทางจากสถานะเริ่มต้นไปยังโหนด n ส่วน h(n) คือ ค่าประมาณแบบฮิวริสติกของต้นทุนของเส้นทางจากโหนด n ไปยังเป้าหมาย ด้วยเหตุนี้เราจึงใช้ f(n) ในการประมาณค่าต้นทุนรวมที่น้อยที่สุดของเส้นทางคำตอบซึ่งเดินทางผ่านโหนด n โหนดที่จะถูกแผ่จะเป็นโหนดที่มีค่า f น้อยที่สุด ถ้ามีโหนดหลายตัวที่มีค่า f เท่ากันเราอาจจะใช้เลือกโหนดตัวที่มีค่า h ต่ำกว่าเป็นเกณฑ์ในการพิจารณาก็ได้ อัลกอริทึมนี้จะจบเมื่อแผ่โหนดเป้าหมาย

เอ-สตาร์จะมีความสามารถในการหาเส้นทางที่เหมาะสมได้ดีถ้าฟังก์ชั่น h(n) มีความเหมาะสมนั่นคือไม่สามารถค่ามากกว่าต้นทุนจริงจนเกินไป ยกตัวอย่างเช่น ระยะทางการบินจะไม่ประมาณค่าเกินมากไปสำหรับระยะทางสัญจร ปัญหาของทั้งเพียวฮิวริสติกและเอ-สตาร์ คือความต้องการด้านหน่วยความจำเพราะจะต้องเก็บลิสต์แบบเปิดทั้งหมด ทำให้เอ-สตาร์มีความจำกัดด้านเนื้อที่ทำให้เอ-สตาร์ไม่เหมาะที่จะนำมาใช้มากกว่าการค้นหาแบบแนวกว้างก่อน ณ ปัจจุบัน

ได้มีการหาวิธีแก้ปัญหาเรื่องข้อจำกัดเรื่องหน่วยความจำในการค้นหาแบบแนวกว้างก่อนของเอ-สตาร์โดยไม่สูญเสียความสามารถในการหาคำตอบที่เหมาะสมซึ่งก็คือ เอ-สตาร์แบบแนวลึกก่อนด้วยการทำซ้ำ : ไอดีเอ-สตาร์ (Iterative-Deepening-A* : IDA*) โดยมีวิธีคือในแต่ละการทำซ้ำของอัลกอริทึมจะทำการค้นหาแบบแบบแนวลึกก่อนที่มีการเก็บทางเดิน (track) ของต้นทุนซึ่งก็คือ f(n) = g(n)+h(n) ของแต่ละโหนดที่ถูกสร้างทันทีที่โหนดที่ถูกสร้างมีต้นทุนมากกว่าขีดจำกัด (threshold) ของรอบการทำซ้ำนั้น เส้นทางเส้นนั้นจะถูกตัดออกไปและจะทำการย้อนกลับเส้นทางการค้นหา, ขีดจำกัดของต้นทุนจะถูกกำหนดตอนเริ่มต้นให้เป็นค่าประมาณแบบฮิวริสติกของสภาพเริ่มต้น และในแต่ละรอบการทำซ้ำต่อ ๆ มา ค่าขีดจำกัดนี้ถูกเพิ่มให้เป็นค่าต้นทุนทั้งหมดของโหนดที่มีค่าน้อยที่สุดซึ่งได้ถูกตัดทิ้งออกไปในการทำซ้ำรอบที่แล้ว อัลกอริทึมจะจบการทำงานเมื่อสภาพเป้าหมายถูกพบและต้นทุนทั้งหมดไม่เกินขีดจำกัดตัวปัจจุบัน

เนื่องจากไอดีเอ-สตาร์ทำการค้นหาเป็นลำดับของการค้นหาแบบแบบแนวลึกก่อนจึงต้องการหน่วยความจำเป็นแบบเชิงเส้น นั่นก็คือความลึกในการค้นหาที่มากที่สุด นอกจากนี้ถ้าฟังก์ชั่นฮิวริสติกมีความเหมาะสมแล้ว ไอดีเอ-สตาร์ก็จะหาคำตอบที่เหมาะสมได้ นอกจากนี้ ไอดีเอ-สตาร์มีคุณสมบัติที่ดีเหมือนกับดีเอฟไอดีที่ได้อธิบายเหตุผลไปแล้ว นั่นคือไอดีเอ-สตาร์จะแผ่โหนดได้เป็นตัวเดียวกับของเอ-สตาร์ในแผนภูมิต้นไม้ และได้รวมข้อดีของเอ-สตาร์อีกด้วยคือ ไอดีเอ-สตาร์มีความเหมาะสมทั้งในด้านเวลาและเนื้อที่มากกว่าอัลกอริทึมการค้นหาแบบฮิวริสติกอื่น ๆ ในการหาคำตอบที่เหมาะสมในแผนภูมิต้นไม้ และไอดีเอ-สตาร์ยังสามารถถูกสร้างได้ง่ายกว่าและทำงานเร็วกว่าเอ-สตาร์เพราะไม่ต้องจัดการลิสต์แบบเปิดและแบบปิด ที่เป็นงานส่วนเกิน

ไอดีเอ-สตาร์นั้นดูจะเป็นอัลกอริทึมที่เหมาะสมที่สุดในปัญหาที่แผนภูมิต้นไม้ค้นหาเป็นแบบไม่จำกัดซึ่งยากที่จะสร้างตัวแก้ปัญหาที่มีค่าน้อยๆ แต่ในกรณีที่ปัญหาที่แผนภูมิต้นไม้ในการค้นหามี ความลึกเป็นแบบจำกัดขอบเขตซึ่งง่ายในการสร้างตัวแก้ปัญหาที่ดีกล่าวคือ ความลึกในการค้นหาที่มากที่สุดสามารถรู้ได้ล่วงหน้า เช่นปัญหาการเดินทางของพนักงานขาย (travelling salesman problem : TSP) ที่กำหนดให้ทำการเยี่ยมแต่ละเมืองทั้งหมดและกลับไปสู่เมืองที่ทำการเริ่มต้นให้ได้ระยะทางที่สั้นที่สุด ปริภูมิปัญหา ตามธรรมชาติสำหรับปัญหานี้ส่วนใหญ่ประกอบด้วยแผนภูมิต้นไม้ที่มีรากแทนเมืองที่เริ่มต้น โหนดที่อยู่ในระดับหนึ่งแทนเมืองทั้งหมดที่สามารถไปเยี่ยมได้ในการเดินทางครั้งแรก โหนดในระดับสองก็จะแทนเมืองทั้งหมดที่สามารถเยี่ยมได้ในการเดินทางครั้งที่สอง เป็นต้น ในแผนภูมิต้นไม้แบบนี้ ความลึกที่มากที่สุดที่เป็นไปได้คือจำนวนของเมือง และคำตอบที่ต้องการก็จะเกิดขึ้นในความลึกนี้ ส่วนด้านเรื่องการใช้เนื้อที่การค้นหานี้ที่มีรูปแบบของแบบแนวลึกก่อนสามารถช่วยในการรับรองได้ว่าจะสามารถหาคำตอบที่เหมาะสมโดยที่ใช้เนื้อที่เป็นเพียงแบบเชิงเส้นเท่านั้นซึ่งก็คือจำนวนของเมือง ในกรณีแบบนี้อัลกอริทึมแบบดีเอฟบีเอ็นบีเป็นอัลกอริทึมที่เหมาะสม โดยแนวคิดของดีเอฟบีเอ็นบีที่ทำให้การค้นหาเป็นไปได้อย่างมีประสิทธิภาพมากขึ้นกว่าคือการเก็บทางเดินของคำตอบที่มีค่าต้นทุนน้อยที่สุด เนื่องมาจากต้นทุนของการท่องเที่ยวแล้วเพียงบางเมืองคือผลรวมของต้นทุนของเส้นที่ได้ท่องเที่ยวมาแล้ว เมื่อใดก็ตามที่พบการท่องเที่ยวแล้วเพียงบางเมืองที่ต้นทุนเท่ากับหรือมากกว่าต้นทุนของการเดินทางที่ดีที่สุดกิ่งที่แสดงการท่องเที่ยวแล้วเพียงบางเมืองนี้ก็จะถูกตัดออกไป เนื่องจากโหนดลูกหลาน (descendent) จะต้องมีต้นทุนที่เท่ากันหรือมากกว่า เมื่อใดที่พบการเดินทางครบที่มีต้นทุนน้อยกว่าก็จะทำการปรับปรุงการเดินทางที่ดีที่สุด นอกจากนี้การใช้ฟังก์ชั่นฮิวริสติกที่เหมาะสม เช่น ใช้ต้นทุนของแผนภูมิต้นไม้แบบทอดข้ามน้อยสุด (minimum spanning) ของเมืองที่ยังไม่ได้ไปที่เหลืออยู่เพิ่มเข้าไปในต้นทุนของการท่องเที่ยวแล้วเพียงบางเมือง ก็สามารถช่วยในการตัดได้ เนื่องมาจากทำการลำดับโหนดลูกทั้งหมดของโหนดนั้นตามผลรวมต้นทุนจากน้อยไปมาก จึงสามารถพบคำตอบที่มีต้นทุนน้อยที่สุดอย่างรวดเร็วและยังช่วยเพิ่มประสิทธิภาพในการตัดอีกด้วย

สิ่งที่น่าสนใจอีกก็คือ ไอดีเอ-สตาร์และดีเอฟดีเอ็นบีจะแสดงถึงพฤติกรรมที่มีลักษณะเป็นคู่กัน เช่น ทั้งสองวิธีจะรับรองได้ว่าจะหาคำตอบที่เหมาะสมโดยใช้เพียงเนื้อที่แบบเชิงเส้น (ถ้าฟังก์ชั่นต้นทุนมีความเหมาะสม) ในไอดีเอ-สตาร์ค่าขีดจำกัดของต้นทุนจะเป็นขอบเขตล่างของต้นทุนของคำตอบที่เหมาะสม และในแต่ละรอบของการทำซ้ำค่านี้จะถูกเพิ่มจนเป็นต้นทุนที่เหมาะสมในดีเอฟบีเอ็นบีค่าต้นทุนของคำตอบที่ดีที่สุดที่ถูกพบจะเป็นขอบเขตบนของต้นทุนของคำตอบที่เหมาะสมและจะมีค่าลดลงจนเป็นต้นทุนที่เหมาะสม นอกจากนี้ไอดีเอ-สตาร์จะไม่แผ่โหนดใด ๆ ที่ต้นทุนเกินกว่าต้นทุนที่เหมาะสมแต่งานส่วนเกินคือการแผ่โหนดเดิมมากกว่าหนึ่ง

ครั้ง ในขณะที่ดีเอฟบีเอ็นบีจะไม่แผ่โหนดเดิมเกินหนึ่งครั้งแต่งานส่วนเกินคือการแผ่บางโหนดที่มีต้นทุนมากกว่าต้นทุนที่เหมาะสม

นอกจากที่กล่าวไปแล้วยังมี อัลกอริทึมฮิวริสติกพาธ (heuristic path) ซึ่งมีเพื่อแก้ปัญหาขนาดใหญ่โดยยอมเสียความถูกต้องแม่นยำเนื่องมาจากในทางปฏิบัติความซับซ้อนในการหาคำตอบที่เหมาะสมโดยปกติจะเป็นเอ็กซ์โปเนนเชียล เนื้อหาที่เกี่ยวกับส่วนนี้คืออัลกอริทึมฮิวริสติกพาธ : เอชพีเอ (HPA) โดยเอชพีเอเป็นอัลกอริทึมการค้นหาแบบเบสท์เฟริสท์ เลขคุณสมบัติของโหนด n คือ f(n)=(1-w)*g(n)+w*h(n) ค่า w ที่แตกต่างกันสามารถทำให้เกิดอัลกอริทึมในรูปแบบต่าง ๆ เช่น ถ้า w=0 จะเป็นการค้นหาแบบยูนิฟอร์ม , w=1/2 จะเป็น A* และ w=1 จะเป็นการค้นหาแบบเพียวฮิวริสติก การเพิ่มค่าของ w มากกว่า 1/2 จะช่วยลดการคำนวณแต่เพิ่มให้ต้นทุนของคำตอบเป็นเลขที่มากขึ้น กลับกันถ้ามีค่าน้อย ๆ จะให้ค่าต้นทุนที่เหมาะสมแต่ใช้เนื้อที่ขนาดใหญ่ในการคำนวณ

เราสามารถการแก้ไขข้อจำกัดด้านหน่วยความจำของอัลกอริทึมการค้นหาแบบฮิวริสติกพาธให้หมดไปได้โดยการแทนที่การค้นหาแบบเบสท์เฟริสท์ด้วยไอดีเอ-สตาร์ที่ใช้ฟังก์ชั่นประมาณค่าเดียวกัน อย่างไรก็ดีสำหรับ w(1/2) จะทำให้ไอดีเอ-สตาร์ไม่เป็นการค้นหาแบบเบสท์เฟริสท์ อีกต่อไป เนื่องจากต้นทุนของโหนดลูกอาจจะน้อยกว่าโหนดแม่ ทำให้การแผ่โหนดไม่เป็นลำดับตามแบบเบสท์เฟริสท์อัลกอริทึมทางเลือกของปัญหานี้คือการค้นหาเบสท์เฟริสท์แบบเวียนเกิด : อาร์บีเอฟเอส (recursive best-first : RBFS) โดยอาร์บีเอฟเอสเป็นการค้นหาแบบเบสท์เฟริสท์ที่ทำงานโดยใช้เนื้อที่เป็นแบบเชิงเส้นซึ่งก็คือ ความลึกในการค้นหาที่มากที่สุด โดยไม่คำนึงถึงฟังก์ชั่นต้นทุนที่ใช้ในการใช้ฟังก์ชั่นต้นทุนที่เหมาะสม อาร์บีเอฟเอสจะสร้างโหนดเป็นจำนวนน้อยกว่าไอดีเอ-สตาร์และทำงานได้ดีกว่าไอดีเอ-สตาร์ ยกเว้นในเรื่องที่มันจะเพิ่มต้นทุนต่อการสร้างโหนด

วิธีการทำงานคือทำการจัดการในสแตคแบบเวียนเกิด (recursion) ที่เก็บเส้นทางทั้งหมดไปยังโหนดปัจจุบันที่กำลังจะถูกแผ่ (รวมทั้งโหลดลูกทั้งหมดที่ติดกับโหนดพี่น้องที่ติดกัน (immediate sibling) ของมันเหล่านี้ ในทุกโหนดที่อยู่ในเส้นทางนั้นด้วย) ตามค่าต้นทุนของโหนดที่ดีที่สุดโดยอยู่ในแผนภูมิต้นไม้ย่อย (subtree) ซึ่งได้ถูกสำรวจแล้วที่อยู่ใต้ในแต่ละโหนดพี่น้องเมื่อใดก็ตามที่ค่าของโหนดนั้นเกินกว่าโหนดอื่น ๆ ที่อยู่ในบางส่วนของแผนภูมิต้นไม้ที่ถูกแผ่แล้ว อัลกอริทึมจะเก็บสำรองโหนดบรรพบุรุษ (ancestor, เป็นโหนดแม่ของโหนดนั้นและโหนดแม่ของโหนดแม่นั้นไปเรื่อยจนถึงราก) และดำเนินการค้นหาลงไปในเส้นทางใหม่ต่อไปเรื่อย ๆ ซึ่งจะรู้สึกได้ว่าอัลกอริทึมจะจัดการค่าขีดจำกัดที่แยกกันสำหรับแต่ละแผนภูมิต้นไม้ย่อยที่แตกออกไปจากเส้นทางการค้นหาปัจจุบัน

บทที่ 3
ระบบการสอบถามข้อมูลจากฐานข้อมูลด้วยข้อความอิสระ

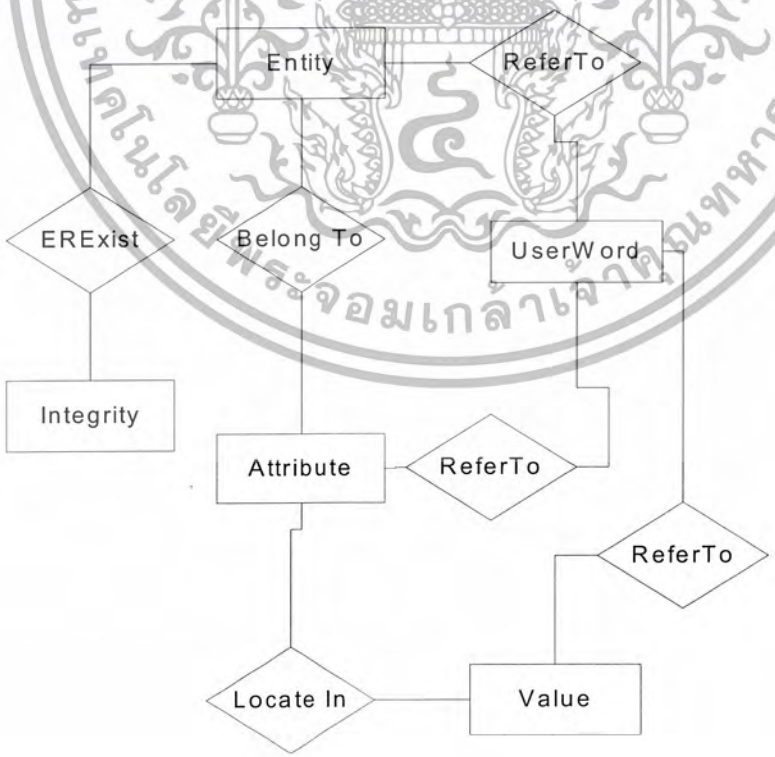
บทนี้เป็นการนำเสนอระบบการสอบถามข้อมูลจากฐานข้อมูลด้วยข้อความอิสระ โดยจะนำเสนอถึงส่วนประกอบของระบบ (ตอนที่ 3.1) , ฐานข้อมูล (ตอนที่ 3.2) และอัลกอริทึมการค้นหาคำตอบที่ดีที่สุด (ตอนที่ 3.3)

3.1 ส่วนประกอบของระบบ

ระบบประกอบด้วยส่วนประกอบหลัก ๆ 2 ส่วน คือ ส่วนของฐานข้อมูลซึ่งทำหน้าที่เก็บข้อมูลที่จำเป็นสำหรับการค้นหาคำตอบและส่วนของโปรแกรมที่ทำหน้าที่ค้นหาคำตอบที่ดีที่สุด โดย การออกแบบส่วนของฐานข้อมูลจะถูกแสดงในตอนที่ 3.2 และ การออกแบบส่วนของโปรแกรมและถูกแสดงในตอนที่ 3.3

3.2 ฐานข้อมูล

3.2.1 แบบจำลองความหมายของฐานข้อมูล

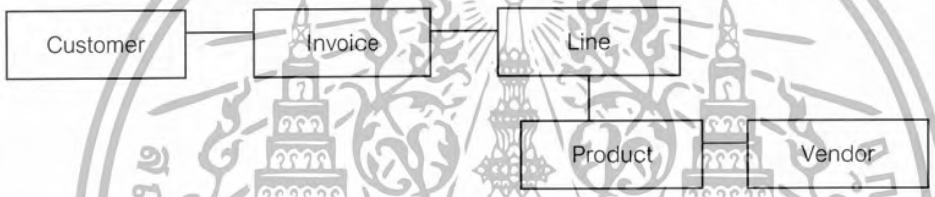


รูปที่ 3.1 โครงสร้างแบบจำลองความหมายของฐานข้อมูล

โครงสร้างของแบบจำลองความหมายของฐานข้อมูลนี้ได้ออกแบบเพื่อใช้จัดเก็บข้อมูลของฐานข้อมูลสามประเภทได้แก่ โครงสร้างของฐานข้อมูล, ค่าข้อมูล และคำที่ผู้ใช้ใช้ในการอ้างถึงส่วนต่าง ๆ ของโครงสร้างฐานข้อมูล โครงสร้างนี้แสดงในรูปที่ 3.1

3.2.2 ฐานข้อมูลตัวอย่าง

ฐานข้อมูลตัวอย่างที่นำมาใช้เป็นตัวอย่างในปัญหาพิเศษนี้ เป็นระบบการซื้อขายสินค้า โดยโครงสร้างของฐานข้อมูลตัวอย่างจะถูกแสดงในรูปที่ 3.2



รูปที่ 3.2 โครงสร้างฐานข้อมูลตัวอย่าง

3.3 อัลกอริทึมการค้นหาคำตอบที่ดีที่สุด

3.3.1 นิยาม

1) กราฟฐานข้อมูล G (Database G) – เป็นกราฟ <V,E> ซึ่ง V เป็นกลุ่มของจุด มี 3 ชนิดคือ เอนติตี้ (entities), แอตทริบิว (attributes) และ ค่าข้อมูล (values) ส่วน E คือ กลุ่มของข้อบังคับของการเชื่อมต่อ ตัวอย่างดังรูป 3.3

2) คำถามข้อมูลด้วยภาษาธรรมชาติ Q (natural language query Q) – สายของตัวอักษรที่แบ่งคำโดยช่องว่าง เช่น “I want to know everything about customer of our company name Smith”

3) คำสำคัญที่จดจำได้ (recognized keywords) - คำสำคัญที่จดจำได้ t_i ของ Q คือส่วนของ Q ที่ตรงกับจุดในกราฟ G หรือคำของผู้ใช้ (userword) เช่น คำว่า smith

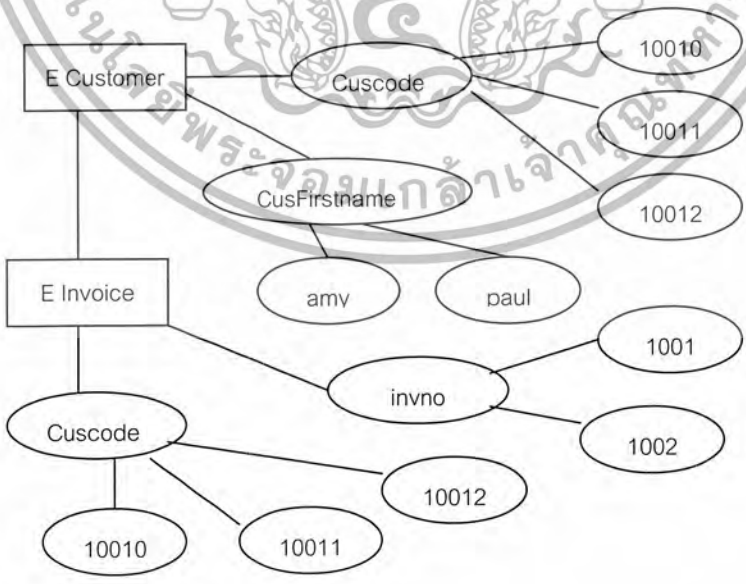
4) กลุ่มของจุดที่จดจำได้ (recognized vertex sets) - กลุ่มของจุดที่จดจำได้ t_i (Vt_i) คือกลุ่มของจุดของกราฟ G ที่จดจำได้หรืออ้างถึงโดย t_i เช่น {Value CusLastname_3|smith, Value CustLastname_10|smith } เป็นกุลุ่มของจุดที่จดจำได้ของ smith

5) รูปลักษณ์ของคำถามข้อมูล QI (query images) – ถ้ามีคำสำคัญที่จดจำได้จำนวน n โดยที่ $n<>0$ QI ในกราฟ G คือ กลุ่มของคำสำคัญที่จดจำได้ $v_i \in Vt_i$ โดยที่ i=1,…,n เช่น { E customer, V CusLastname_3|smith}

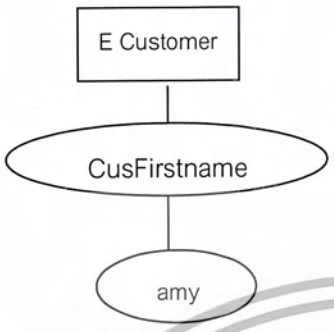
6) เส้นทางความหมาย sp (semantic path) – ถ้ามีกลุ่มของจุดที่จดจำได้ คือ v_i คือ กลุ่มของจุดที่น้อยที่สุดใน G ที่มี v_i โดยตรงตามเงื่อนไขดังนี้ คือ มี จุดเอนติตี้และจุดของมันเองเชื่อมต่อกัน โดยจุดของมันเองนอกเหนือจาก v_i จะเป็นจุดที่ถูกตีความหมายโดยการแปลความหมาย (semantics) ของ vi ตัวอย่างดังรูป 3.4

7) กราฟที่เป็นไปได้ (feasible graph) – ถ้ามี QI ที่ประกอบด้วยจุดที่จดจำได้จำนวน n ซึ่ง $n\neq0$ คือ กลุ่มของ sp_i ซึ่ง i = 1, …, n และ sp_i คือเส้นทางความหมายหนึ่งที่ถูกตีความหมายโดยจุดที่จดจำได้ v_i ของ QI ตัวอย่างดังรูป 3.5

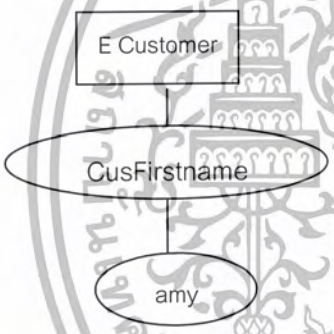
8) กราฟคำถาม (query graph) – เป็นกราฟที่ประกอบไปด้วย <V,E> โดย V คือ กลุ่มของจุดที่น้อยที่สุดใน G ที่ประกอบไปด้วย vi ที่เป็นเซตย่อยของ Vt_i ซึ่งตรงตามเงื่อนไขต่อไปนี้ได้แก่ มีจุด เอนติตี้ และจุดเหล่านั้นต้องเชื่อมต่อกัน และ E คือเส้นที่เชื่อมต่อ ตัวอย่างดังรูป 3.6



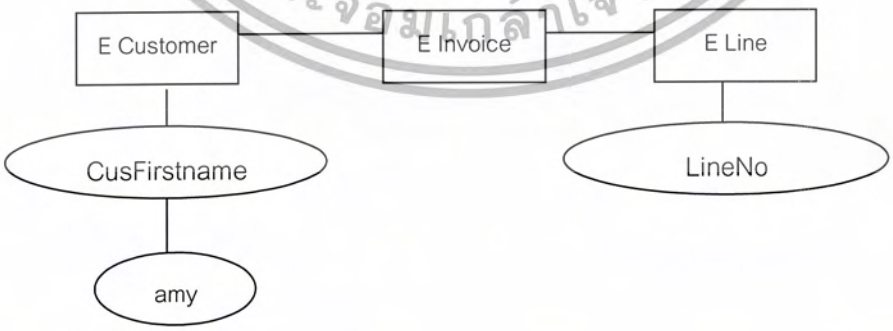
รูปที่ 3.3 ตัวอย่างบางส่วนของกราฟฐานข้อมูล



รูปที่ 3.4 ตัวอย่างเส้นทางความหมาย



รูปที่ 3.5 ตัวอย่างกราฟที่เป็นไปได้



รูปที่ 3.6 ตัวอย่างกราฟคำถาม

3.3.2 การตั้งปัญหา

เนื่องจากคำถามข้อมูลแต่ละคำถามอาจจะประกอบด้วยคำสำคัญหลายคำและคำสำคัญแต่ละคำอาจชี้ไปที่ส่วนประกอบของกราฟฐานข้อมูลได้หลายส่วน ดังนั้นแต่ละคำถามอาจประกอบด้วยหลาย QI และแต่ละ QI สามารถสร้างเป็นกราฟคำถามได้หลายกราฟคำถาม

ปัญหาคือกราฟคำถามใดจะเป็นกราฟคำถามที่ดีที่สุด ภายใต้ข้อสมมุติว่าแนวความคิดที่ใช้แทนความหมายของคำตอบจะเป็นแนวความคิดที่อยู่ใกล้กันในกราฟฐานข้อมูล ดังนั้นคำตอบที่ดีที่สุดสำหรับคำถามจะเป็นคำตอบกราฟคำถามที่มีเส้นต่อน้อยที่สุดหรือกราฟคำถามที่มีต้นทุนต่ำที่สุด

ดังนั้น ปัญหาการหาคำตอบที่เหมาะสมที่สุดสำหรับคำถามคือ ปัญหาการค้นหากราฟย่อยที่มีต้นทุนต่ำสุดในกราฟฐานข้อมูล ภายใต้ข้อจำกัดคือ กราฟฐานข้อมูล

3.3.3 อัลกอริทึมการค้นหา

โปรแกรมจะรับค่าข้อมูลเข้าเป็นคำถามข้อมูลด้วยภาษาธรรมชาติ โดยแยกเป็นคำซึ่งแบ่งโดยช่องว่าง

ขั้นที่ 1: ค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้
ขั้นที่ 2: กำหนด QI จากกลุ่มของคำสำคัญที่จดจำได้
ขั้นที่ 3: ค้นหากราฟคำถามค้นหาที่มีต้นทุนต่ำที่สุดโดยใช้อัลกอริทึมบรานช์แอนด์บาวน์
ขั้นที่ 4: สร้างคำสั่งเอสคิวแอล (SQL)
โดยแต่ละขั้นมีรายละเอียดดังนี้

ขั้นที่ 1: ค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้ - หาคำที่ตรงกับจุดในกราฟ G หรือในคำของผู้ใช้พร้อมทั้งกลุ่มของจุดกราฟที่ตรงกับคำสำคัญ โดยมีอัลกอริทึมดังนี้

ขั้นที่ 1: แบ่งข้อความออกเป็นคำ โดยแต่ละคำจะแบ่งโดยใช้ช่องว่าง
ขั้นที่ 2:สำหรับแต่ละคำที่ได้ จะทำการค้นหาในฐานข้อมูลเมตาเดตาว่ามีจุดที่จดจำได้หรือไม่
ขั้นที่ 3: ถ้ามีจะถือว่าคำนั้นเป็นคำสำคัญที่จดจำได้ จะทำการเก็บคำสำคัญที่จดจำได้นี้พร้อมกับจุดที่จดจำได้ของมัน
ขั้นที่ 4: กลับไปทำขั้นที่ 2 จนกระทั่งทำครบหมดทุกคำ

ขั้นที่ 2: กำหนด QI จากกลุ่มของคำสำคัญที่จดจำได้ โดย QI ทั้งหมดที่เป็นไปได้หาได้จากอัลกอริทึมดังนี้

ขั้นที่ 1 : คำนวณหาจำนวนของรูปลักษณ์ของคำถามข้อมูล (n) โดยนำจำนวนของจุดที่จดจำได้ในแต่ละคำสำคัญที่จดจำได้มาคูณกัน

ขั้นที่ 2: หา QI โดยให้ QI ทั้งหมดจะเป็นกลุ่มของ QI โดยสมาชิกตัวแรกใน QI จะตรงกับจุดที่จดจำได้จุดหนึ่งในคำสำคัญที่จดจำได้ตัวแรก และสมาชิกตัวที่สองใน QI จะตรงกับจุดที่จดจำได้จุดหนึ่งในคำสำคัญที่จดจำได้ตัวที่สอง และจะเป็นเช่นนี้สำหรับสมาชิกตัวต่อ ๆ ไป

ขั้นที่ 3 : สำหรับคำสำคัญที่จดจำได้ตัวแรก ให้ ng เป็นจำนวนจุดที่จดจำได้ของคำสำคัญที่จดจำได้ตัวแรก(nv1) และ ne จะเป็น n/ng

ขั้นที่ 4: นำจุดที่ 1 เพิ่มลงไปใน QI ที่ 1 ถึง ne นั่นคือใส่จุดใน QI เรียงกันตามลำดับจำนวน ne ครั้ง สำหรับจุดต่อ ๆ ไปก็เช่นกันคือใส่จุดเป็นจำนวน ne ครั้ง ทำเช่นนี้จนจุดครบ ng จุด ในกรณีที่จุดหมดแล้วแต่ยังต้องทำเนื่องจากไม่ครบ ng ครั้งให้ย้อนกลับไปนำ จุดที่ 1 มาใส่ใหม่

ขั้นที่ 5 : ให้ ng เป็น ng จากขั้นที่แล้วคูณด้วยจำนวนจุดที่จดจำได้ของคำสำคัญที่จดจำได้ตัวที่สอง(nv2) และ ne จะเป็น ne/nv2 กลับทำตามขั้นที่ 4 ทำจนกระทั่งครบทุกคำ

ขั้นที่ 3: ค้นหากราฟคำถามค้นหาที่มีต้นทุนต่ำที่สุดโดยใช้อัลกอริทึมบรานช์แอนด์บาวน์ – ปัญหาของบรานช์แอนด์บาวน์คือเพื่อกำหนดกราฟคำถามที่ดีที่สุดจากกราฟคำถามที่เป็นไปได้ทั้งหมด โดยกราฟคำถามที่ดีที่สุดคือกราฟที่มีการเดินทางในโครงสร้างของฐานข้อมูลน้อยที่สุด เนื่องจากกราฟคำถามเป็นทรี ต้นทุนของมันคือจำนวนของกลุ่มของจุดในกราฟคำถามลบด้วย 1 เราจะทำการสร้างฟังก์ชั่นประมาณค่า LB เพื่อหาค่าขอบล่างสำหรับจุดของ QI

ในขั้นนี้จะทำการค้นหากราฟคำถามที่มีต้นทุนต่ำที่สุดจากกลุ่มของ QI ที่ได้มาจากขั้นตอนที่แล้ว โดยจะใช้วิธีการค้นหาแบบเบสท์เฟริสท์ โดยใช้โครงสร้างข้อมูลแบบฮีพ (min priority queue) โดยข้อมูลตัวแรกแสดงถึงจุดที่มีความหวังมากที่สุด (มีค่าขอบล่างต่ำที่สุด) เพื่อสำรวจต่อไป รากของทรีและจุดระหว่างกลางจะถูกใส่เพิ่มและดึงออกระหว่างการทำบรานช์แอนด์บาวน์เมื่อข้อมูลตัวแรกของฮีพถูกนำมาใช้ มันจะถูกดึงออกจากฮีพ รากและจุดระหว่างกลางพร้อมกับค่าขอบล่างของมันจะถูกใส่เข้าไปในตำแหน่งที่เหมาะสมในฮีพโดยขึ้นกับค่าขอบล่างของมัน

ขบวนการแผ่กิ่งก้านสาขา (branching) ของในบรานช์แอนด์บาวน์จะดำเนินการดังนี้ ถ้าจุดปัจจุบัน (นำจุดมาจากฮีพ) เป็นรากมันจะกำหนด QI และค่าขอบล่าง ถ้าจุดปัจจุบันเป็น QI มันจะ

กำหนดกราฟที่เป็นไปได้ และค่าขอบล่าง ถ้าจุดปัจจุบันเป็นกราฟที่เป็นไปได้ มันจะกำหนด กราฟคำถามและค่าของมัน ในการกำหนดกราฟคำถามจากกราฟที่เป็นไปได้ มันนำอัลกอริทึมการหาเส้นทางที่สั้นที่สุด (Shortest path)

การคำนวณหาค่าขอบล่างของกราฟที่เป็นไปได้ นั้นจะเป็นการนับจุด เนื่องจากข้อสมมุติที่ว่าเอนติตี้ ทั้งหมดของกราฟที่เป็นไปได้ เชื่อมต่อกัน แต่ข้อสมมุตินี้ไม่สามารถใช้หาค่าขอบล่างของรูปลักษณ์ของคำถามข้อมูลได้ โดยเราจะใช้วิธีคำนวณที่เกี่ยวข้องกับการระบุตำแหน่งและการนับดังอัลกอริทึมดังนี้

ขั้นที่ 1 :นับจำนวนจุดเอนติตี้ของ QI

ขั้นที่ 2 : นับจำนวนแอตทริบิวใน QI สำหรับแต่ละแอตทริบิวใน QI นับจำนวน เอนติตี้ ของแอตทริบิวซึ่งไม่ใช่เอนติตี้ ที่ได้นับไปแล้ว

ขั้นที่ 3 :แต่ละค่าข้อมูล ใน QI นับจำนวนเอนติตี้ที่แอตทริบิวของมันมีค่าข้อมูล นี้ โดยเอนติตี้นี้ไม่ซ้ำกับตัวที่นับไปแล้วและให้นับแอตทริบิวที่ไม่ได้ถูกนับไปแล้วเหล่านี้ด้วย

ขั้นที่ 4 : ค่าขอบเขตล่างของ QI คือ ค่าที่ได้นับจากขั้นตอนที่ 1 ถึง 3 มาบวกกันแล้วลบด้วย 1

ขั้นที่ 4: สร้างคำสั่งเอสคิวแอล

3.3.4 ตัวอย่างการทำงาน

กรณีที่ 1

คำถามข้อมูลด้วยภาษาธรรมชาติ : I want to know code of customer name smith

ขั้นที่ 1 : ค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้ผลลัพธ์ดังตารางที่ 3.1

คำสำคัญที่จดจำได้	กลุ่มของจุดที่จดจำได้
Code	A CusAreaCode A CusCode A PCode A VAreacode

	A VCode
Customer	E Customer
Name	A CusFirstName A CusLastName A VName
Smith	V smith

ตารางที่ 5.1 คำสำคัญที่จดจำได้และกลุ่มของจุดที่จดจำได้สำหรับกรณีที่ 1

ขั้นที่ 2: กำหนด QI จากกลุ่มของคำสำคัญที่จดจำได้ - ได้ QI ดังนี้

QI 1 : {A CusAreaCode, E Customer, A CusFirstName, V Smith}

QI 2 : {A CusAreaCode, E Customer, A CusLastName, V Smith}

QI 3 : {A CusAreaCode, E Customer, A Vname, V Smith}

QI 4 : {A CusCode, E Customer, A CusFirstname, V Smith}

QI 5 : {A CusCode, E Customer, A CusLastname, V Smith}

QI 6 : {A CusCode, E Customer, A Vname, V Smith}

QI 7 : {A PCode, E Customer, A CusFirstName, V smith}

QI 8 : {A PCode, E Customer, A CusLastName, V smith}

QI 9 : {A PCode, E Customer, A VName, V smith}

QI 10 : {A VAreacode, E Customer, A CusFrstname, V smith}

QI 11 : {A VAreacode, E Customer, A CusLastname, V smith}

QI 12 : {A VAreacode, E Customer, A VName, V smith}

QI 13 : {A VCode, E Customer, A CusFisrtaname, V smith}

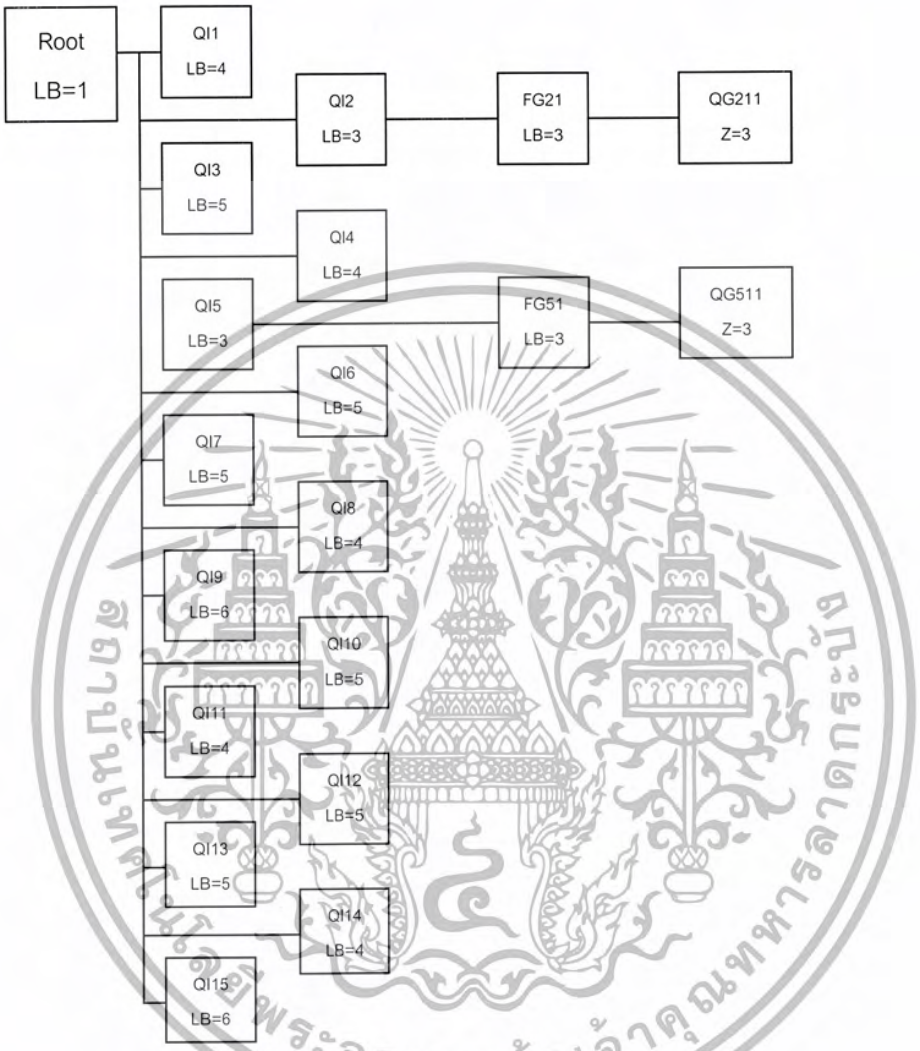
QI 14 : {A VCode, E Customer, A CusLastname, V smith}

QI 15 : {A VCode, E Customer, A VName, V smith}

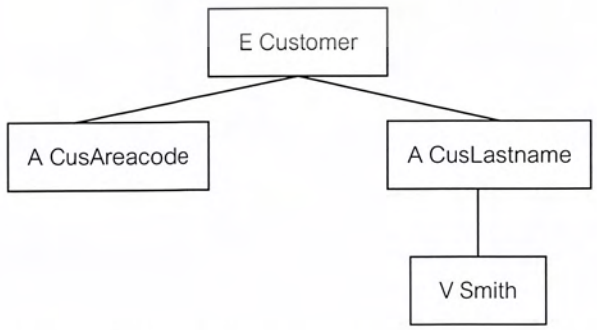
ขั้นที่ 3

ค้นหากราฟคำถามที่ดีที่สุดโดยใช้อัลกอริทึมการค้นหาแบบบรานช์แอนด์บาวน์กราฟการค้นหาของกรณีที่ 1 ถูกแสดงดังรูปที่ 3.7 โดยจากเริ่มที่ราก ได้รูปลักษณ์ของคำถามข้อมูลที่เป็นไปได้ คือ QI1-QI15 ใน QI เหล่านี้ QI ที่มีค่าขอบเขตล่างน้อยที่สุดคือ QI2 ซึ่งมีกราฟที่เป็นไปได้ คือ FG21 ดังรูปที่ 3.8 หลักจากนั้นเนื่องจาก QI5 มีค่าขอบเขตล่างเท่ากับ FG11 คือ 3 ซึ่งน้อยที่สุดจึงทำ QI5 ได้กราฟที่เป็นไปได้คือ FG51 ดังรูปที่ 3.9 จากนั้น FG21 นี้มีค่าขอบเขตล่างน้อยที่สุดคือ 3 จึงทำ FG21 ได้กราฟคำถาม QG211 ซึ่งมีต้นทุนคือ 3 ดังรูป 3.10 และ FG51 นี้ก็มีค่าขอบเขตล่างน้อย

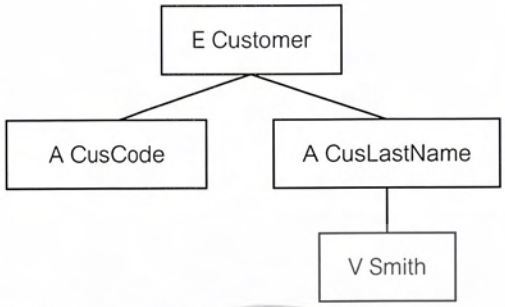
ที่สุดด้วยคือ 3 จึงทำ FG51 ได้กราฟคำถาม QG511 ซึ่งมีต้นทุนคือ 3 ดังรูป 3.11 ไม่มีจุดอื่นอีกที่จะให้ต้นทุนต่ำกว่านี้ ดังนั้น QG211 จึงเป็นคำตอบ



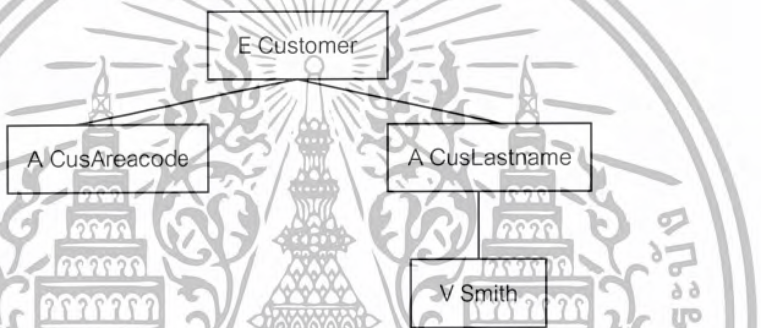
รูปที่ 3.7 กราฟการค้นหาสำหรับกรณีที่ 1



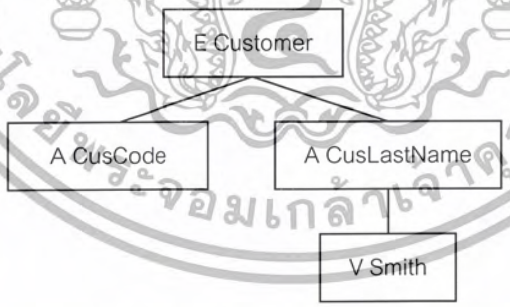
รูปที่ 3.8 กราฟที่เป็นไปได้ FG21 ของ QI2



รูปที่ 3.9 กราฟที่เป็นไปได้ FG51 ของ QI5



รูปที่ 3.10 กราฟคำถาม QG211 ของ FG21



รูปที่ 3.11 กราฟคำถาม QG511 ของ FG51

ขั้นที่ 4 : สร้างคำสั่งเอสคิวแอล

select Customer.*, Customer.CusAreacode, Customer.CusLastname from Customer where (Customer.CusLastname='smith')

กรณีที่ 2

คำถามข้อมูลด้วยภาษาธรรมชาติ : what date does amy buy product ?

ขั้นที่ 1 : ค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้ผลลัพธ์ดังตารางที่ 3.2

คำสำคัญที่จดจำได้	กลุ่มของจุดที่จดจำได้
Date	A InvDate
	A PInDate
Amy	V Amy
Product	E Product

ตารางที่ 3.2 คำสำคัญที่จดจำได้และกลุ่มของจุดที่จดจำได้สำหรับกรณีที่ 2

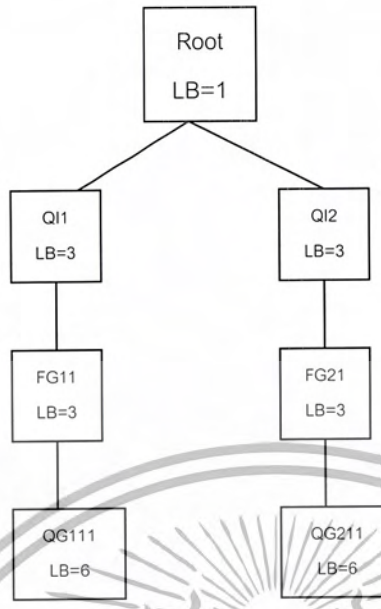
ขั้นที่ 2: กำหนด QI จากกลุ่มของคำสำคัญที่จดจำได้ - ได้ QI ดังนี้

QI1 : {A InvDate, V amy, E Product }

QI2 : { A PInDate, V amy, E Product }

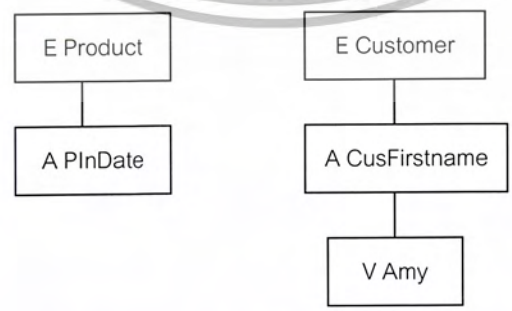
ขั้นที่ 3

ค้นหากราฟคำถามที่ดีที่สุดโดยใช้อัลกอริทึมการค้นหาแบบบรานช์แอนด์บาวน์ กราฟการค้นหาของกรณีที่ 1 ถูกแสดงดังรูปที่ 3.12 โดยจากเริ่มที่ราก ได้รูปลักษณ์ของคำถามข้อมูลที่เป็นไปได้ คือ QI1 และ QI2 ใน QI เหล่านี้ QI ที่มีค่าขอบเขตล่างน้อยที่สุดคือ QI2 ซึ่งมีกราฟที่เป็นไปได้คือ FG21 ดังรูปที่ 3.13 ซึ่ง FG นี้มีค่าขอบเขตล่างน้อยที่สุดคือ 3 จึงทำ FG21 ได้กราฟคำถาม QG211 ซึ่งมีต้นทุนคือ 6 ดังรูป 3.14 ไม่มีจุดอื่นอีกที่จะให้ต้นทุนต่ำกว่านี้ ดังนั้น QG211 จึงเป็นคำตอบ

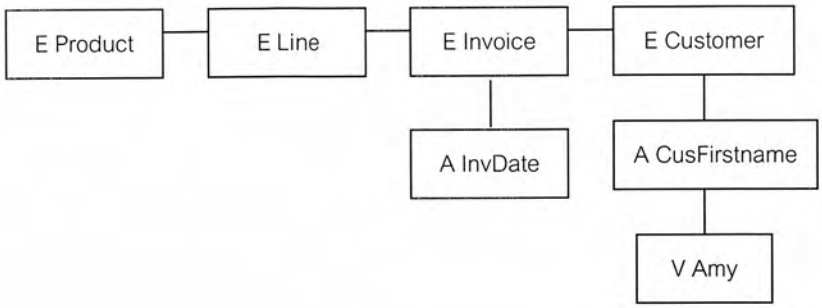


รูปที่ 3.12 กราฟการค้นหาสำหรับกรณีที่ 2

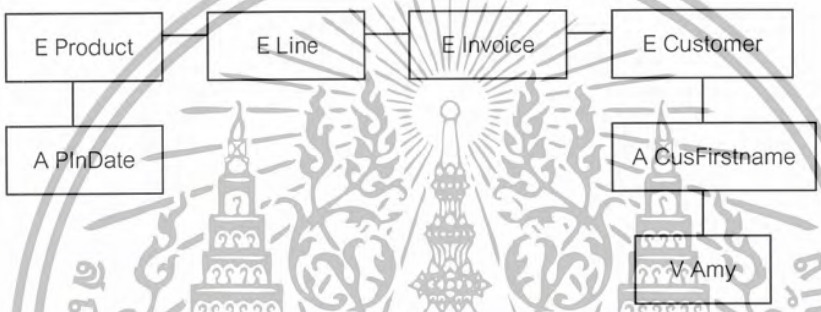
รูปที่ 3.13 กราฟที่เป็นไปได้ FG11 ของ QI1



รูปที่ 3.13 กราฟที่เป็นไปได้ FG21 ของ QI2



รูปที่ 3.14 กราฟคำถาม QG111 ของ FG11



รูปที่ 3.14 กราฟคำถาม QG211 ของ FG21

ขั้นที่ 4 สร้างคำสั่งเอสคิวแอล

คำสั่งเอสคิวแอลของ QG111

select Invoice.*, Customer.*, Customer.CusFirstname, Line.*, Product.*, Invoice.InvDate from Invoice, Customer, Line, Product where Invoice.CusCode=Customer.CusCode and (Customer.CusFirstname='amy') and Invoice.InvNo=Line.InvNo and Line.PCode=Product.PCode

คำสั่งเอสคิวแอลของ QG211

select Invoice.*, Customer.*, Customer.CusFirstname, Line.*, Product.*, Product.PInDate from Invoice, Customer, Line, Product where Invoice.CusCode=Customer.CusCode and (Customer.CusFirstname='amy') and Invoice.InvNo=Line.InvNo and Line.PCode=Product.PCode

บทที่ 4
การพัฒนาระบบ

บทนี้จะเป็นการนำเสนอการพัฒนาระบบของปัญหาพิเศษ โดยส่วนแรกจะนำเสนอถึงเครื่องมือที่ใช้ในการพัฒนาโปรแกรม ส่วนที่สองจะนำเสนอการพัฒนาฐานข้อมูล ส่วนที่สามนำเสนอการพัฒนาโปรแกรม และส่วนสุดท้ายจะนำเสนอการดำเนินงานของระบบ

4.1 เครื่องมือที่ใช้ในการพัฒนาระบบ

เครื่องมือทุกชิ้นที่ใช้ในปัญหาพิเศษนี้ทำงานภายใต้ระบบปฏิบัติการ Window XP Professional โดยเครื่องมือที่ใช้ในการพัฒนาโปรแกรมประกอบด้วย Java 2 SDK, Standard Edition รุ่น 1.4.2 ทำหน้าที่แปลและดำเนินงานโปรแกรม, JCreator Pro รุ่น 2.5 ทำหน้าที่เป็นโปรแกรมช่วยอำนวยความสะดวกในการเขียนโปรแกรม และ Java Servlet Development Kit รุ่น 2.1 เป็นเครื่องมือในการพัฒนาเซิร์ฟเลตส่วนเครื่องมือที่ใช้ในการพัฒนาฐานข้อมูล คือระบบฐานข้อมูล Oracle9i

4.2 การพัฒนาฐานข้อมูล

ฐานข้อมูลประกอบด้วย 2 ส่วน คือ ฐานข้อมูลของแบบจำลองความหมายและฐานข้อมูลตัวอย่าง โดยแบบจำลองความหมายของฐานข้อมูลที่ได้ออกแบบสามารถถูกแปลงให้อยู่ในรูปแบบตารางของฐานข้อมูลเชิงสัมพันธ์ โดยตารางนี้ถูกสร้างโดยใช้คำสั่งเอสคิวแอลด้านล่างนี้

```
create table ATTRIB
(
 AttribCode number(3) primary key,
 AttribName varchar(15),
 iformat char(1),
 ilength varchar(5)
);

create table ENTITY
(
 ERname varchar(10),
 AttribCode number(3),
```

```
 relpos number(2),
 inpkey char(1) check (inpkey in('Y','N')),
 posinpk number(2),
 constraint PK_Itemcode_ERname primary key(Attribcode,ERname),
 constraint FK_Itemcode FOREIGN KEY(Attribcode)
        REFERENCES ATTRIB
);

create table Value
(
 Vcode number(4),
 AttribCode number(3),
 data varchar(50),
 constraint PK_Vcode_Itemcode primary key(Vcode,Attribcode),
 constraint FK_LocateIn_Itemcode FOREIGN KEY(Attribcode)
        REFERENCES ATTRIB
);

create table INTEGRITY
(
 Intname varchar(20) primary key,
 inttype char(2),
 master varchar(10),
 slave varchar(10)
);

create table UserWordEntity
(
 userword varchar(50),
 word varchar(50)
);
create table UserWordAttrib
```

```
(
 userword varchar(50),
 word varchar(50)
);
create table UserWordValue
(
 userword varchar(50),
 word varchar(50)
);
```

และฐานข้อมูลตัวอย่างที่ได้ออกแบบก็สามารถถูกแปลงให้อยู่ในรูปแบบตารางของฐานข้อมูลเชิงสัมพันธ์ โดยตารางนี้ถูกสร้างโดยใช้คำสั่งเอสคิวแอลด้านล่างนี้

```
Create  Table   Customer
        (cusCode        CHAR(5),
        cusLastname     VARCHAR(15),
        cusFirstname    VARCHAR(10),
        cusAreacode     CHAR(3),
        cusTel          CHAR(7),
        cusBalance      NUMBER(6,2),
        CONSTRAINT  Pk_CusCode    PRIMARY KEY(CusCode));

Create  Table   Vendor
        (vCode          CHAR(5),
        vName           VARCHAR(20),
        vContact        VARCHAR(15),
        vAreacode       CHAR(3),
        vTel            CHAR(7),
        CONSTRAINT  Pk_VCode      PRIMARY KEY(VCode));

Create  Table   Product
        (pCode  CHAR(8),
```

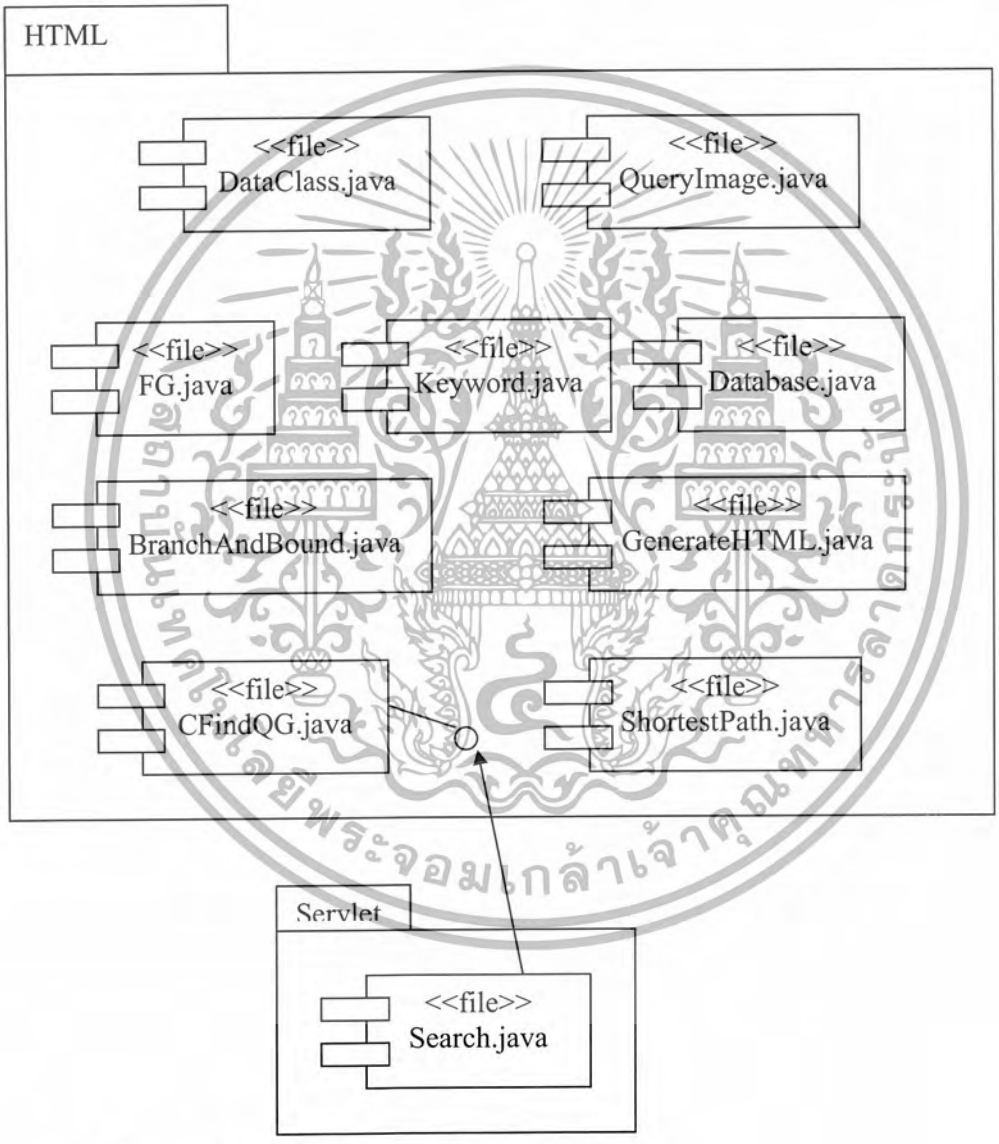
```
    pDesc   VARCHAR(45),
    pInDate DATE,
    pOnHand         NUMBER(8),
    pPrice  NUMBER(7,2),
    vCode   CHAR(5),
    CONSTRAINT  Pk_PCode        PRIMARY     KEY(PCode),
    CONSTRAINT  Fk_Product_VCode        FOREIGN     KEY(VCode)
            REFERENCES  Vendor(VCode));

Create  Table   Invoice
    (InvNo  CHAR(4),
    CusCode         CHAR(5),
    InvDate DATE,
    CONSTRAINT  Pk_InvNo        PRIMARY     KEY(InvNo),
    CONSTRAINT  Fk_Invoice_CusCode      FOREIGN     KEY(CusCode)
            REFERENCES  Customer(CusCode));

Create  Table   Line
    (InvNo  CHAR(4),
    LineNo NUMBER(2),
    PCode   CHAR(8),
    LineUnit        NUMBER(3),
    LinePrice       NUMBER(6,2),
    CONSTRAINT  Pk_InvNoLineNo      PRIMARY     KEY(InvNo,LineNo),
    CONSTRAINT  Fk_Line_PCode FOREIGN       KEY(PCode)
            REFERENCES  Product(PCode),
    CONSTRAINT  Fk_Line_InvNo FOREIGN       KEY(InvNo)
            REFERENCES  Invoice(InvNo));
```

4.3 การพัฒนาโปรแกรม

โปรแกรมที่ถูกสร้างสามารถแสดงในรูปแบบ แบบจำลองอิมพลีเมนเทชั่น (Implementation model) ได้ดังรูปที่ 4.1 โดยโปรแกรมประกอบด้วยส่วนประกอบหลัก 2 ส่วน คือ ส่วน HTML ทำ

หน้าที่สร้างเอชทีเอ็มแอล (HTML) จากกราฟคำถามซึ่งเป็นคำตอบ และส่วน Servlet ทำหน้าที่แสดงข้อมูลจาก HTML ที่ได้แก่ผู้ใช้ทางอินเทอร์เน็ต โดยใช้เซิร์ฟเลต (servlet)

ส่วน HTML จะประกอบด้วยแฟ้มข้อมูลโปรแกรมที่เขียนโดยใช้ภาษาจาวาจำนวน 9 แฟ้มข้อมูล และส่วนเซิร์ฟเลตก็ประกอบด้วยแฟ้มข้อมูลโปรแกรมที่เขียนโดยใช้ภาษาจาวาเช่นกัน จำนวน 1 แฟ้มข้อมูล และแฟ้มข้อมูล Search.java ในเซิร์ฟเลตจะทำการเรียกใช้ฟังก์ชั่นของ CFindQG.java ใน HTML



รูปที่ 4.1 แบบจำลองอิมพลีเมนเทชั่นของโปรแกรม

อัลกอริทึมค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้

```
/* ค้นหาคำสำคัญและกลุ่มของจุดที่จดจำได้
@parameter
input : คำถามข้อมูลด้วยภาษาธรรมชาติ
@return
คำสำคัญและกลุ่มของจุดที่จดจำได้*/
getKeywordVertex(String input)
{
    แบ่งข้อความ input เป็นคำ โดยใช้ช่องว่างเป็นตัวคั่น ใส่ทุกคำลงไปใน words
    for(int i=0;i<จำนวนของคำใน words;i++)
    {
        หากลุ่มของจุดที่จดจำได้ของคำจาก Metadatabse
        ถ้ามีกลุ่มของจุดที่จดจำได้ จะใส่กลุ่มของจุดที่จดจำได้นี้พร้อม keyword ลงไปใน
        kv
    }
        return kv;
}
```

อัลกอริทึมการหารูปลักษณ์ของคำถามข้อมูลจากกลุ่มของจุดที่จดจำได้

```
//หารูปลักษณ์ของคำถามข้อมูลจากกลุ่มของจุดที่จดจำได้
findQI()
{
    int nv,ne,ng,base,row,sz,sk;

    nv=numAllVertex();
    ne=nv; ng=1;
    sk=จำนวนของ keyword

    จองเนื้อที่ในหน่วยความจำให้แก่ QISet

    for(int i=0;i<sk;i++)
    {
```

```
            VS=กลุ่มของจุดที่จดจำได้กลุ่มที่ i
            sz=จำนวนจุดใน VS
            ng*=sz;
            ne/=sz;
            for(int j=0;j<ng;j++)
            {
                base=j*ne;
                for(int k=0;k<ne;k++)
                {
                    นำจุดที่ตำแหน่ง j%sz ใน VS ใส่ใน QISet ที่ตำแหน่ง j*ne+k
                }
            }
        }
        return QISet; //ผลลัพธ์ของรูปลักษณ์ของคำถามข้อมูลทั้งหมด
    }

    // หาจำนวนรูปลักษณ์ของคำถามข้อมูล
    int numAllVertex()
    {
        int num;
        num=จำนวนของจุดที่จดจำได้ของแต่ละคำถูณกัน
        return num;
    }
```

อัลกอริทึมการค้นหากราฟคำถาม

```
    /* @parameter
    KV : กลุ่มของจุดที่จดจำได้
    @ return
    กลุ่มของกราฟคำถาม
    */
    BBSearch(DKeywordVertex KV)
    {
```

```
int current_best_LB=จำนวนเต็มที่มีค่ามาก ๆ
int min_successor_LB;
int current_LB;

ใส่ KV ลงไปใน heap
current_vertex=สมาชิกตัวแรกใน heap
นำสมาชิกตัวแรกใน heap ออกจาก heap
current_LB=ค่าขอบเขตล่างของ current_vertex
while(current_best_LB>=current_LB)
{
    min_successor_LB=จำนวนเต็มที่มีค่ามาก;
    successor_set=branch(current_vertex);
    for(int i=0;i<จำนวนของ successor ใน successor_set;i++)
    {
        successor= successor ตัวที่ I ใน successor_set
        if(ถ้าไม่เป็น terminal vertex)
        ใส่ successor ลงไปใน heap;

        if(ขอบเขตล่างของ successor<min_successor_LB)
        min_successor_LB=ค่าขอบเขตล่างของ successor

    }
    for(int i=0;i<จำนวนของ successor ใน successor_set;i++)
    {
        successor= successor ตัวที่ I ใน successor_set
        if(min_successor_LB==ค่าขอบเขตล่างของ successor)
        {
            if(ถ้าไม่เป็น terminal vertex)
            current_LB=min_successor_LB;

            else
            {
```

```
                    if(min_successor_LB<=current_best_LB)
                    {
                        if(min_successor_LB<current_best_LB)
                        {
                            ลบค่าทั้งหมดใน QGS
                        }
                        ใส่ successor ลงใน QGS
                        current_best_LB=min_successor_LB;
                    }
                }
            }
        } //end for
        if(ถ้า Heap ไม่ว่าง)
        {
            current_vertex=สมาชิกตัวแรกใน heap
            นำสมาชิกตัวแรกใน heap ออกจาก heap
        }
        else
        {
            current_LB=จำนวนเต็มที่มีค่ามาก;
        }
    }//end while

    return QGS;
}

branch(HeapElement H)
{
    if( H เป็น Root)
    {
        ResultKeyword=กลุ่มของคำที่จดจำได้ที่อยู่ใน H
        ResultQI=รูปลักษณ์ของคำถามข้อมูลทั้งหมดที่ได้มาจาก ResultKeyword
```

```
        for(int i=0;i<จำนวนของรูปลักษณ์ของคำถามข้อมูลใน ResultQI;i++)
        {
            ให้ QI เป็นรูปลักษณ์ของคำถามข้อมูลตัวที่ I ใน ResultQI
            หาค่าขอบเขตล่างของ QI
            ใส่ QI พร้อมค่าขอบเขตล่างลงไปใน HS
        }
    }
    else if ( H เป็นรูปลักษณ์ของคำถามข้อมูล)
    {
        QI=รูปลักษณ์ของคำถามข้อมูลที่อยู่ใน H
        FGS=Feasible graph ทั้งหมดที่ได้มาจาก QI
        for(int i=0;i<จำนวน Feasible graph ใน FGS;i++)
        {
            ให้ FG เป็น Feasible graph ตัวที่ I ใน FGS
            หาค่าขอบเขตล่างของ FG
            ใส่ QI พร้อมค่าขอบเขตล่างลงไปใน HS
        }
    }
    else if ( H เป็น Feasible Graph)
    {
        FG=Feasible graph ที่อยู่ใน H
        QGS=กราฟคำถามทั้งหมดที่ได้มาจาก FG
        for(int i=0;i<จำนวนกราฟคำถามใน QGS;i++)
        {
            ให้ QG เป็นกราฟคำถามตัวที่ I ใน QGS
            หาค่าขอบเขตล่างของ QG
            ใส่ QG พร้อมค่าขอบเขตล่างลงไปใน HS
        }
    }
    return HS;
}
```

อัลกอริทึมการหาค่าขอบเขตล่างของรูปลักษณ์ของคำถามข้อมูล

```
/* @parameter
QI : รูปลักษณ์ของคำถามข้อมูล
@return
ค่าขอบเขตล่างของรูปลักษณ์ของคำถามข้อมูล
*/
int determineLB(DQI QI)
{
    int count=0;

    for(int i=0;i<จำนวน vertex ใน QI;i++)
    {
        V=vertex ตัวที่ I ใน QI;
        if(V เป็น Entity)
        ใส่ V ลงไปใน ER_Set
    }

    for(int i=0;i<จำนวน vertex ใน QI;i++)
    {
        V=vertex ตัวที่ I ใน QI;
        if(ถ้า V เป็น Attribute)
        {
            ใส่ V ลงไปใน Attrib_Set
            ++count;
            ลบค่าทั้งหมดใน Attrib_ER_Set
            หา Entityทั้งหมดที่ Attribute ของมันตรงกับ V จาก metadatabase
            ใส่ Entity เหล่านี้ลงไปในAttrib_ER_Set

            if(ไม่มีสมาชิกใดของ Attrib_ER_Set ที่อยู่ใน ER_Set หรือ Counted_ER_Set)
            {
                ++count;
```

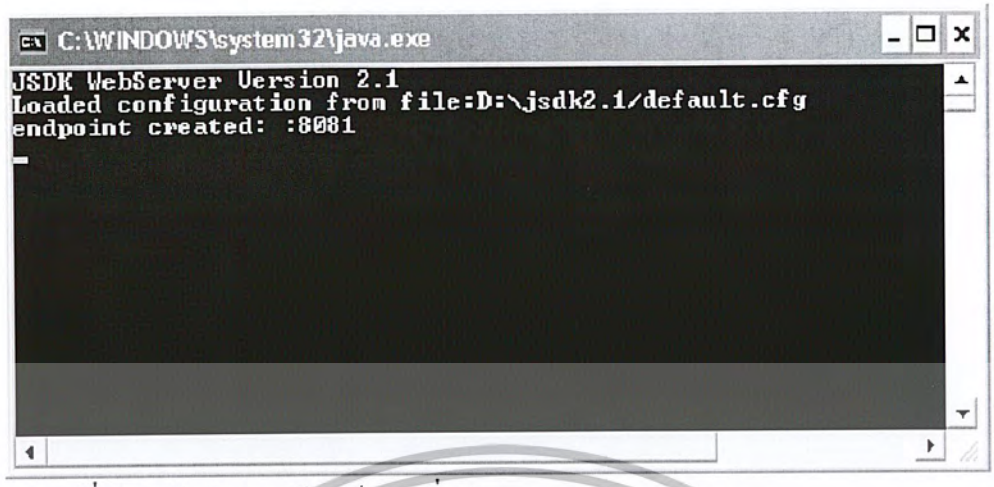
```
        if(ไม่มีสมาชิกใดของ Attrib_ER_Set ที่อยู่ใน ER_Set)
        {
            for(int j=0;j<จำนวน vertex ในAttrib_ER_Set;j++)
            {
                X=vertex ตัวที่ j ในAttrib_ER_Set
                if(X ไม่อยู่ใน Counted_ER_Set)
                ใส่ X ลงไปใน Counted_ER_Set
            }
        }
    }
}
else if(V เป็น Value)
{
    ++count;
    ลบค่าทั้งหมดใน Value_ER_Set
    หา Entity ทั้งหมดที่ Attribute ของมันมี Value ที่ตรงกับ V จาก metadatabase
    ใส่ Entity เหล่านี้ลงไปใน Value_ER_Set

    if(ไม่มีสมาชิกใดของ Value_ER_Set ที่อยู่ใน ER_Set หรือ Counted_ER_Set)
    {
        ++count;
        if(ไม่มีสมาชิกใดของ Value_ER_Set ที่อยู่ใน ER_Set)
        {
            for(int j=0;j<จำนวน vertex ในValue_ER_Set;j++)
            {
                X=vertex ตัวที่ j ใน Value_ER_Set;
                if(X ไม่อยู่ใน Counted_ER_Set)
                ใส่ X ลงไปใน Counted_ER_Set;
            }
        }
    }
```

```
                หา Attribute ทั้งหมดที่ตรงกับ V จาก metadatabase
                ใส่ Attribute เหล่านี้ลงไปใน Attrib

                if(Attrib ไม่อยู่ใน Attrib_Set)
                {
                    ++count;
                    ใส่ Attrib ลงไปใน Attrib_Set;
                }
            }
        }
        return count+|จำนวน vertex ใน ER_Set|-1;
    }
```

4.4 การดำเนินงานของระบบ

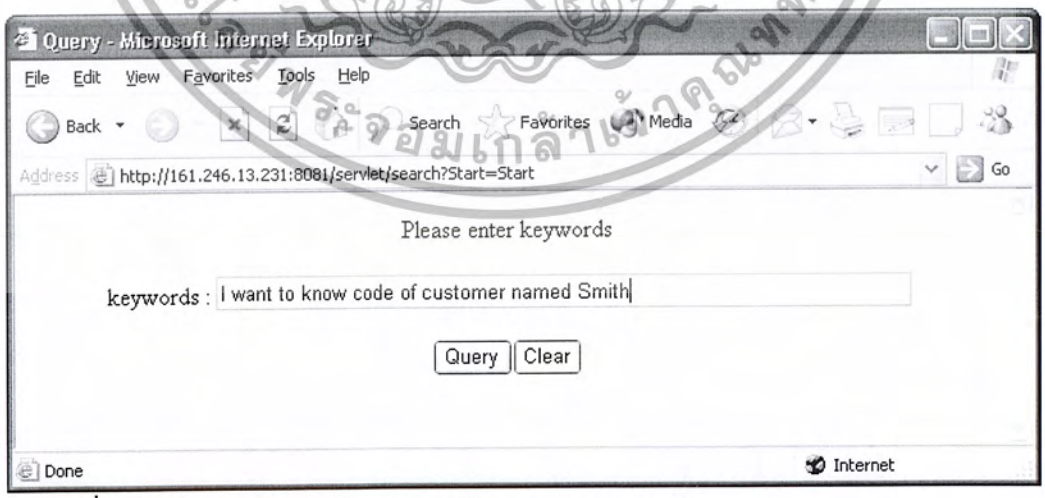
ผู้ใช้งานระบบจะแบ่งออกเป็น 2 กลุ่ม คือ ผู้ให้บริการระบบและผู้ใช้งานระบบ โดยผู้ให้บริการระบบมีหน้าที่เปิดเซอร์วิช (service) ของโปรแกรมให้ผู้ใช้บริการสามารถใช้งานได้ และผู้ใช้บริการจะทำการค้นหาข้อมูลที่ต้องการจากระบบโดยใช้บราวเซอร์ (browser) ทางอินเทอร์เน็ต

เริ่มแรกผู้ให้บริการเปิดเซอร์วิชของเซิร์ฟเลตโดยเปิดโปรแกรมชื่อ startserver.bat จะได้ผลลัพธ์ดังรูปที่ 4.2 (วิธีการปรับแต่งค่าของเซิร์ฟเลตเพื่อให้ใช้งานได้อยู่ในภาคผนวก) เมื่อเซิร์ฟเลตเริ่มทำงานแล้วผู้ใช้งานระบบจะสามารถทำการค้นหาข้อมูลที่ต้องการได้ โดยขั้นแรกผู้ใช้งานเปิดโปรแกรมบราวเซอร์ป้อนที่อยู่หน้าเว็บเพจ (URL) ของผู้ให้บริการ (ที่อยู่หน้าเว็บเพจถูกกำหนดโดยผู้ให้บริการ) แล้วบราวเซอร์จะแสดงข้อความต้อนรับดังรูปที่ 4.3 หลังจากนั้นผู้ใช้ทำการกดปุ่ม Start เพื่อเริ่มใช้งานจะได้ผลลัพธ์ดังรูปที่ 4.4 แล้วผู้ใช้ทำการป้อนคำถามข้อมูลภาษาธรรมชาติที่ต้องการลงในช่องข้อความชื่อ keywords แล้วกด Query เพื่อค้นหาข้อมูล แต่ถ้าผู้ใช้ต้องการลบข้อความในกล่องข้อความให้กดปุ่ม Clear หลังจากที่ผู้ใช้กดปุ่ม Query แล้วระบบจะทำงานเพื่อหาคำตอบและจะแสดงผลลัพธ์ที่ได้ดังรูปที่ 4.5

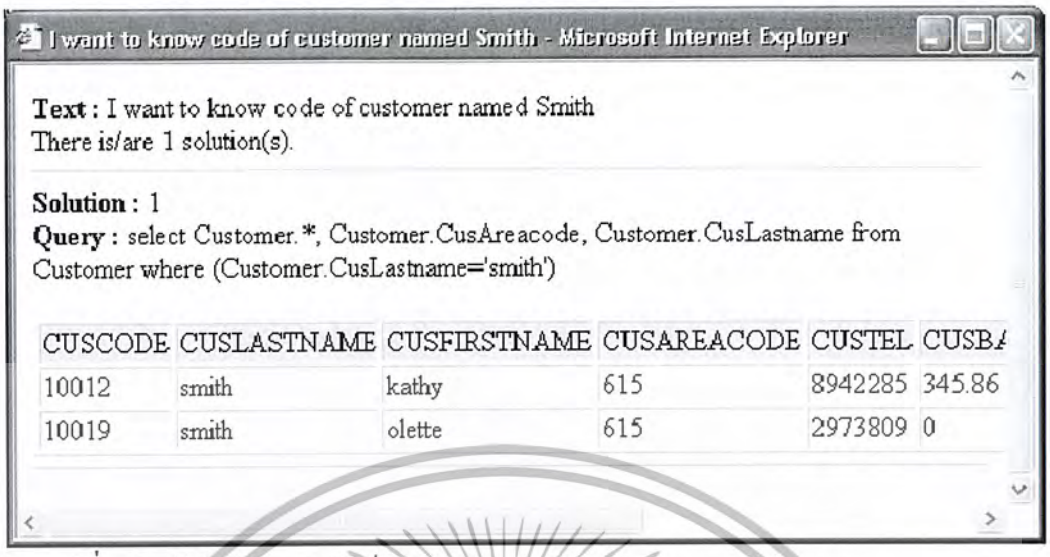


รูปที่ 4.2 หน้าต่างแสดงถึงเซิร์ฟเลตเริ่มทำงาน

รูปที่ 4.3 หน้าต่างแสดงข้อความต้อนรับ



รูปที่ 4.4 หน้าแสดงสำหรับป้อนคำถามข้อมูลภาษาธรรมชาติ



รูปที่ 4.5 หน้าต่างแสดงผลลัพธ์ของคำตอบ

4.5 การทดลอง

การทดลองมีวัตถุประสงค์เพื่อทดสอบความถูกต้องของคำตอบจากคำถามสอบถามข้อมูลจากผู้ใช้ที่มีพื้นฐานแตกต่างกัน

4.5.1 ออกแบบการทดลอง

1) ผู้ใช้

มีผู้ใช้ จำนวน 10 คนซึ่งมีความหลากหลายในด้านความรู้ โดยก่อนทำการทดลองจะอธิบายวิธีการใช้งานโปรแกรมเบื้องต้นให้ผู้ทดลองฟังประมาณ 5 นาทีและคอยให้คำแนะนำเบื้องต้น

2) ความรู้ของผู้ใช้

ความรู้ของผู้ใช้ที่ใช้ในการประเมินคือ ความรู้ในการใช้คอมพิวเตอร์ โดยจะทำการให้คะแนนเป็นตัวเลขจำนวนเต็มตั้งแต่ 0 ถึง 10 คุณภาพจะเรียงตามลำดับตัวเลข โดยเลข 0 หมายถึง ไม่มีความรู้เลย และเลข 10 หมายถึง มีความรู้อย่างดีเยี่ยม ตัวเลขที่ได้ถูกแสดงดังตารางที่ 4.1

ความรู้ \ คนที่	1	2	3	4	5	6	7	8	9	10
ความรู้คอมพิวเตอร์	7	3	8	5	7	8	6	8	4	9

ตารางที่ 4.1 การประเมินความรู้ของผู้ทดลอง

4.5.2 ผลการทดลอง

ผู้ใช้แต่ละคนทำการทดลองคนละ 5 ครั้งได้ผลดังตารางที่ 4.2

ผล \ คนที่	1	2	3	4	5	6	7	8	9	10
ถูกต้อง	5	4	5	4	5	4	5	4	5	3
ไม่ถูกต้อง	0	1	0	1	0	1	0	1	0	2

ตารางที่ 4.2 จำนวนคำตอบที่ถูกต้องและไม่ถูกต้องของผู้ทดลอง

หมายเหตุ

- ในกรณีที่ผู้ใช้มีความรู้ด้านคอมพิวเตอร์ต่ำ เมื่อให้คำแนะนำเพิ่มเติมในการปรับปรุงคำถามจะทำให้ได้คำตอบที่ต้องการมากขึ้น
- ผลลัพธ์ที่ได้จากข้อความคำถามข้อมูลอาจประกอบไปด้วยข้อมูลที่ไม่เกี่ยวข้องกับข้อความคำถาม แต่ถ้าในผลลัพธ์ที่แสดงออกมามีคำตอบที่ผู้ใช้ต้องการอยู่ด้วยจะถือว่าเป็นผลลัพธ์ที่ถูกต้อง

จากผลการทดลอง คำถามที่ตอบได้ถูกต้องมีจำนวน 44 ครั้ง

และคำถามที่ตอบได้ไม่ถูกต้องมีจำนวน 6 ครั้ง ดังนี้

ครั้งแรกคือคำถาม “give me name of customer that has lastname smith and firstname john” ซึ่งไม่มีข้อมูลนี้ในระบบแต่ระบบกลับแสดงผลลัพธ์ เป็นข้อมูลของลูกค้าซึ่งมี lastname เป็น smith ซึ่งไม่ใช่ข้อมูลที่ผู้ใช้ต้องการ

ครั้งที่2 คือคำถาม “information of pcode 1546-qq” ซึ่งไม่มีข้อมูลนี้แต่ระบบกลับแสดงข้อมูลทั้งหมดออกมา

ครั้งที่3 คือคำถาม “all customers details” เนื่องจากไม่มีคำสำคัญ

ครั้งที่4 คือคำถาม “company b&k” เนื่องจากไม่มีคำสำคัญ

ครั้งที่5 คือคำถาม “price > 14” เนื่องจากไม่สามารถทำการดำเนินการทางคณิตศาสตร์ได้

ครั้งที่6 คือคำถาม “pcode that amy and myron buy” เนื่องจากระบบจะแสดงข้อมูลที่ amy และ myron ซื้อ

4.5.3 สรุปผลการทดลอง

จากการวิเคราะห์ผลการทดลองพบว่าคำถามที่ระบบสามารถตอบได้อย่างถูกต้องนั้นเป็นคำถามข้อมูลแบบพื้นฐาน กล่าวคือมีคำสำคัญในข้อความคำถามครบถ้วนและเป็นการถามข้อมูลเพียงค่าเดียว ในขณะที่คำถามที่ระบบไม่สามารถให้ข้อมูลที่ถูกต้องได้นั้นมีหลายสาเหตุดังต่อไปนี้

- ไม่มีคำสำคัญในข้อความคำถาม ซึ่งมีสาเหตุเพราะไม่สามารถกำหนดคำของผู้ใช้ให้ครอบคลุมทุกๆ คำที่สามารถเป็นไปได้
- หากข้อความคำถามเป็นการถามค่าของข้อมูลจากฐานข้อมูลที่ไม่มีอยู่ ระบบจะแสดงข้อมูลทั้งหมด แทนที่จะแสดงผลลัพธ์ว่าไม่มีข้อมูล

- คำถามที่เป็นการร้องขอข้อมูลที่มีเงื่อนไขเป็นการดำเนินการ “และ” (and) จะไม่สามารถแสดงได้โดยจะให้ผลลัพธ์ของการดำเนินการ “หรือ” (or)
- หากคำถามข้อมูลประกอบด้วยการดำเนินการทางคณิตศาสตร์ เช่น >, < ระบบจะไม่สามารถหาผลลัพธ์ได้

สรุปได้ว่าหากข้อความคำถามเป็นคำถามข้อมูลแบบพื้นฐานระบบสามารถหาคำตอบได้อย่างถูกต้อง ในขณะที่คำถามข้อมูลที่ซับซ้อนหรือมีลักษณะตรงกับที่กล่าวไว้ข้างต้นระบบจะไม่สามารถให้คำตอบได้อย่างถูกต้อง โดยระดับความรู้ของผู้ทดสอบจะมีผลคือหากผู้ใช้มีความรู้ไม่สูงมักจะเกิดปัญหาจากการไม่มีคำสำคัญ ในขณะที่ผู้ใช้ที่มีความรู้สูงจะทดลองด้วยคำถามที่มีความซับซ้อนมากทำให้ได้ผลลัพธ์ไม่ถูกต้อง

บทที่ 5
สรุปผลและข้อเสนอแนะ

ในบทนี้จะนำเสนอผลจากการทดสอบการทำงานของระบบและข้อจำกัด พร้อมทั้งข้อเสนอแนะในการพัฒนาต่อไป

5.1 สรุปผล

การพัฒนาระบบการหาคำตอบที่เหมาะสมของคำถามสอบถามจากฐานข้อมูลโดยใช้เมตาเดตาเซอร์ช ได้ดำเนินตามวัตถุประสงค์และ ประโยชน์ที่วางไว้ คือ สามารถพัฒนาอัลกอริทึมในการหากราฟย่อยที่ดีที่สุดต่อคำถามสอบถามข้อมูลโดยไม่ต้องทำการแจงและประเมินกราฟย่อยที่เป็นไปได้ทั้งหมด โดยขั้นตอนการพัฒนาเริ่มจากการออกแบบโครงสร้างแบบจำลองการจัดเก็บความหมายของข้อมูลในฐานข้อมูลเพื่อใช้เป็นระบบจำลอง ศึกษาวิธีการค้นหาแบบต่าง ๆ เลือกวิธีการค้นหาโดยเลือกใช้ อัลกอริทึมบรานช์แอนด์บาวน์ นำมาทำการออกแบบและพัฒนาระบบจนสามารถใช้งานได้จริง

เมื่อทำการพัฒนาระบบจนสมบูรณ์แล้วได้ทำการทดสอบระบบโดยได้ทำการออกแบบการทดลองเพื่อทดสอบความถูกต้องของคำตอบจากคำถามสอบถามข้อมูลจากผู้ใช้ที่มีพื้นฐานแตกต่างกันเพื่อหาขีดความสามารถและข้อจำกัดของระบบ โดยได้ข้อสรุปว่าหากข้อความคำถามเป็นคำถามข้อมูลแบบพื้นฐานระบบสามารถหาคำตอบได้อย่างถูกต้อง ในขณะที่คำถามข้อมูลที่ซับซ้อนมากระบบจะไม่สามารถหาคำตอบที่ถูกต้องได้เนื่องจากระบบยังมีข้อจำกัดอยู่บ้าง

5.2 ข้อจำกัดของโปรแกรม

จากการทดลองระบบโดยให้ผู้ใช้ที่มีพื้นฐานต่างกันเป็นผู้ทดสอบพบว่าหากข้อความคำถามที่มีรูปแบบดังต่อไปนี้

- ไม่มีคำสำคัญในข้อความคำถาม
- เป็นการถามค่าของข้อมูลจากฐานข้อมูลที่ไม่มีอยู่
- คำถามที่เป็นการร้องขอข้อมูลที่มีเงื่อนไขเป็นการดำเนินการ and
- ประกอบด้วยการดำเนินการทางคณิตศาสตร์ เช่น >, <

ระบบจะไม่สามารถหาคำตอบที่ถูกต้องสำหรับข้อความคำถามได้ และสำหรับบางคำสอบถามข้อมูลพบว่าผลลัพธ์ของคำถามที่ได้จากระบบ อาจประกอบด้วยข้อมูลที่ไม่เกี่ยวข้องกับคำถามข้อมูลรวมอยู่ด้วย

จากข้อมูลข้างต้นทำให้สรุปได้ว่าระบบยังมีข้อจำกัดดังนี้

- ข้อความคำถามยังต้องมีรูปแบบเพื่อให้ระบบสามารถทำงานได้อย่างถูกต้อง ยังไม่สามารถใช้ข้อความคำถามที่เป็นภาษาธรรมชาติอย่างแท้จริงได้
- ไม่สามารถหาคำตอบสำหรับคำถามข้อมูลที่ต้องการข้อมูลเป็นช่วง ค่าเฉลี่ย หรือต้องอาศัยการดำเนินการทางคณิตศาสตร์อื่นๆ ได้
- ผลลัพธ์ของคำถามข้อมูลที่แสดงออกมายังมีความไม่สะดวกในการดูเท่าที่ควร เพราะสำหรับบางคำถามผู้ใช้อาจต้องหาข้อมูลที่ต้องการ จากข้อมูลที่ไม่เกี่ยวข้องที่แสดงออกมาทั้งหมด

5.3 ข้อเสนอแนะ

โครงงานนี้มีจุดประสงค์เพื่อพัฒนาอัลกอริทึมในการหาคำตอบ (กราฟย่อย) ที่ดีที่สุดต่อคำถามสอบถามข้อมูลด้วยภาษาธรรมชาติ โดยไม่ต้องทำการแจกแจงและประเมินกราฟย่อยที่เป็นไปได้ทั้งหมด ซึ่งสามารถนำไปพัฒนาต่อเพื่อสร้างเป็นระบบการตอบคำถามสอบถามข้อมูลด้วยภาษาธรรมชาติที่สมบูรณ์ได้ โดยจะขอนำเสนอแนวทางในการพัฒนาบางส่วนดังต่อไปนี้

- พัฒนาระบบให้สามารถใช้กับฐานข้อมูลและข้อความคำถามที่เป็นภาษาไทย
- พัฒนาระบบให้สามารถรองรับข้อความคำถามที่เป็นภาษาธรรมชาติอย่างแท้จริง
- พัฒนาระบบให้สามารถหาคำตอบสำหรับข้อความคำถามที่มีเพียงส่วนใด ส่วนหนึ่งของคำสำคัญ (ข้อความย่อย) ได้
- พัฒนาระบบให้สามารถหาคำตอบสำหรับคำถามข้อมูลที่ต้องการข้อมูลเป็นช่วง ค่าเฉลี่ย หรือต้องอาศัยการดำเนินการทางคณิตศาสตร์อื่นๆ
- พัฒนาระบบให้แสดงข้อมูลเฉพาะผลลัพธ์ของคำถามข้อมูลเท่านั้น

บรรณานุกรม

Richard E. Korf. 1996. **Artificial Intelligence Search Algorithms**. [online]. Available : http://citeseer.nj.nec.com/ korf98artificial.html.

Rob ,P. and Coronel C. **Database System: Design, Implementation, and Management.** 5th ed. Course Technology.

Veera Boonjing. 2003. "A Branch-and-Bound Algorithm for Natural Language Database Query Using Metadata Search." **KMITL Journal.** 3(1).

Veera Boonjing. 2002. "A Metadata Search Approach to Natural Language Database Query." Ph.D. Dissertation, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute.

Waiman Cheung and Cheng Hsu. 1994. **The Model-Assisted Global Query System for Multiple Databases in Distributed Enterprises.** [online]. Available : http://viu.eng.rpi.edu/publications/ magqs.pdf.

ภาคผนวก ก

ข้อมูลของฐานข้อมูลตัวอย่าง

ข้อมูลของฐานข้อมูลตัวอย่างประกอบไปข้อมูลดังนี้

CUSTOMER					
CUSCODE	CUSLASTNAME	CUSFIRSTNAME	CUSAREACODE	CUSTEL	CUSBALANCE
10010	Ramas	Alfred	615	8442573	0
10011	Dunne	Leona	713	8941238	0
10012	Smith	Kathy	615	8942285	345.86
10013	Olowski	Paul	615	8942180	536.75
10014	Orlando	Myron	615	2221672	0
10015	o'brian	Amy	713	4423381	0
10016	Brown	James	615	2971228	221.19
10017	Williams	George	615	2902556	768.93
10018	Farriss	Anne	713	3827185	216.55
10019	Smith	Olette	615	2973809	0

INVOICE		
INVNO	CUSCODE	INVDATE
1001	10014	18-Aug-99
1002	10011	18-Aug-99
1003	10012	18-Aug-99
1004	10011	19-Aug-99
1005	10018	19-Aug-99
1006	10014	19-Aug-99
1007	10015	19-Aug-99
1008	10011	19-Aug-99

LINE

INVNO	LINENO	PCODE	LINEUNIT	LINEPRICE
1001	1	13-q2/p2	1	14.99
1001	2	23109-hb	1	9.95
1002	1	54778-2t	2	4.99
1003	1	2238/qpd	1	38.95
1003	2	1546-qq2	1	39.95
1003	3	13-q2/p2	5	14.99
1004	1	54778-2t	3	4.99
1004	2	23109-hb	2	9.95
1005	1	pvc23drt	12	5.87
1006	1	sm-18277	3	6.99
1006	3	23109-hb	1	3.95
1006	4	89-wre-q	1	256.99
1007	1	13-q2/p2	2	14.99
1007	2	54778-2t	1	4.99
1008	1	pvc23drt	5	5.87
1008	2	wr3/tt3	3	119.95
1008	3	23109-hb	1	9.95
1006	2	2232/qty	1	109.92

PRODUCT					
PCODE	PDESC	PINDATE	PONHAND	PPRICE	VCODE
13-q2/p2	7.25-in pwr saw blade	12-Jul-99	32	14.99	21344
14-q1/l3	9.00-in pwr saw blade	12-Jun-99	18	17.49	21344
1546-qq2	hrd cloth, 1/4-in 2*50	14-Aug-99	15	39.95	23119
1558-qw1	hrd cloth, 1/2-in 3*50	14-Aug-99	23	43.99	23119
2232/qwe	B&D jigsaw 8-in blade	23-Jul-99	6	99.87	24288
2238/qpd	B&D cordless drill 1/2-in	19-Aug-99	12	38.95	25595
23109-hb	claw hammer	19-Aug-99	23	9.95	21225
23114-aa	sledge hammer 12 lb	1-Aug-99	8	14.4	

54778-2t	rat-tail file 1/8-in fine	14-Jul-99	43	4.99	21344
89-wre-q	hicut chain saw 16-in	6-Sep-99	11	256.99	24288
pvc23drt	PVC pipe 3.5-in 8-ft	19-Sep-99	188	5.87	
sm-18277	1.25-in metal screw, 25	28-Sep-99	172	6.99	21225
sw-23116	2.5-in wd. Screw, 50	23-Sep-99	237	8.45	21231
wr3/tt3	steel matting 4'*8'*1/6", .5"mesh	16-Aug-99	18	119.95	25595
2232/qty	B&D jigsaw 12-in blade	29-Jul-99	8	109.92	24288

VENDOR

VCODE	VNAME	VCONTACT	VAREACODE	VTEL
21225	bryson,inc	smithson	615	2233234
21226	super logo,inc	flushing	904	2158995
21231	D&E supply	singh	615	2283245
21344	gomez Bros	ortega	615	8892546
22567	dome supply	smithson	901	6781419
23119	randset Ltd	anderson	901	6783998
24004	brackman bros	browning	615	2281410
24288	ORDVA, inc	hakford	615	8981234
25443	B&K,inc	smithson	904	2270093
25501	damal suppies	smythe	615	8903529
25595	rubicon sis	orton	904	4560092

ภาคผนวก ข

ข้อมูลของแบบจำลองความหมายของฐานข้อมูลตัวอย่าง

ข้อมูลของแบบจำลองความหมายของฐานข้อมูลตัวอย่างประกอบไปด้วยข้อมูลดังนี้

ATTRIB			
ATTRIBCODE	ATTRIBNAME	IFORMAT	ILENGTH
1	CusCode	C	5
2	CusLastname	V	15
3	CusFirstname	V	10
4	CusAreacode	C	3
5	CusTel	C	7
6	CusBalance	N	6,2
7	VCode	C	5
8	VName	V	20
9	VContact	V	15
10	VAreacode	C	3
11	VTel	C	7
12	PCode	C	8
13	PDesc	V	45
14	PInDate	D	
15	POnHand	N	8
16	PPrice	N	7,2
17	InvNo	C	4
18	InvDate	D	
19	LineNo	N	2
20	LineUnit	N	3
21	LinePrice	N	6,2

ENTITY				
ERNAME	ATTRIBCODE	RELPOS	INPKEY	POSINPK
Customer	1	1	Y	1
Customer	2	2	N	
Customer	3	3	N	
Customer	4	4	N	
Customer	5	5	N	
Customer	6	6	N	
Vendor	7	1	Y	1
Vendor	8	2	N	
Vendor	9	3	N	
Vendor	10	4	N	
Vendor	11	5	N	
Product	12	1	Y	1
Product	13	2	N	
Product	14	3	N	
Product	15	4	N	
Product	16	5	N	
Product	7	6	N	
Invoice	17	1	Y	1
Invoice	1	2	N	
Invoice	18	3	N	
Line	17	1	Y	1
Line	19	2	Y	2
Line	12	3	N	
Line	20	4	N	
Line	21	5	N	

INTEGRITY			
INTNAME	INTTYPE	MASTER	SLAVE

Fk_Product_VCode	FK	Product	Vendor
Fk_Invoice_CusCode	FK	Invoice	Customer
Fk_Line_PCode	FK	Line	Product
Fk_Line_InvNo	FK	Line	Invoice

USERWORDATTRIB	
USERWORD	WORD
code	cuscode
lastname	cuslastname
surname	cuslastname
firstname	Cusfirstname
name	Cusfirstname
name	Cuslastname
area	Cusareacode
code	Cusareacode
tel.	Custel
telephone	Custel
balance	Cusbalance
no	Invno
number	Invno
date	Invdate
no	Lineno
number	Lineno
code	Pcode
unit	Lineunit
price	Lineprice
description	Pdesc
detail	Pdesc
date	Pindate
price	Pprice

code	Vcode
name	Vname
contact	vcontact
area	vareacode
code	vareacode
tel.	Vtel
telephone	Vtel

USERWORDENTITY	
USERWORD	WORD
buyer	customer
client	customer
consumer	Customer
patron	Customer
purchaser	Customer
shopper	Customer
account	Invoice
balance	Invoice
bill	Invoice
debt	Invoice
list	Invoice
reckoning	Invoice
statement	Invoice
brand	product
artifact	product
goods	product
invention	product
stock	product
seller	Vendor
dealer	Vendor

merchant	Vendor

USERWORDVALUE	
USERWORD	WORD
August	19-Aug-99
19	19-Aug-99
18	18-Aug-99
99	19-Aug-99
99	18-Aug-99
1999	19-Aug-99
1999	18-Aug-99
12	12-Jun-99
june	12-Jun-99
99	12-Jun-99
1999	12-Jun-99
12	12-Jul-99
july	12-Jul-99
99	12-Jul-99
1999	12-Jul-99
14	14-Aug-99
August	14-Aug-99
99	14-Aug-99
1999	14-Aug-99
23	23-Jul-99
july	23-Jul-99
99	23-Jul-99
1999	23-Jul-99
19	19-Aug-99
August	19-Aug-99
99	19-Aug-99

1999	19-Aug-99
August	1-Aug-99
1	1-Aug-99
99	1-Aug-99
1999	1-Aug-99
14	14-Jul-99
July	14-Jul-99
99	14-Jul-99
1999	14-Jul-99
6	6-Sep-99
September	6-Sep-99
99	6-Sep-99
1999	6-Sep-99
19	19-Sep-99
September	19-Sep-99
99	19-Sep-99
1999	19-Sep-99
28	28-Sep-99
September	28-Sep-99
99	28-Sep-99
1999	28-Sep-99
23	23-Sep-99
September	23-Sep-99
99	23-Sep-99
1999	23-Sep-99
16	16-Aug-99
August	16-Aug-99
99	16-Aug-99
1999	16-Aug-99
29	29-Jul-99

July	29-Jul-99
99	29-Jul-99
1999	29-Jul-99
August	18-Aug-99
jim	james
bill	williams

VALUE		
VCODE	ATTRIBCODE	DATA
1	6	0
2	14	1-Aug-99
3	14	6-Sep-99
4	19	1
5	20	1
6	13	1.25-in metal screw, 25
7	17	1001
8	1	10010
9	1	10011
10	1	10012
11	1	10013
12	1	10014
13	1	10015
14	1	10016
15	1	10017
16	1	10018
17	1	10019
18	17	1002
19	17	1003
20	17	1004
21	17	1005

22	17	1006
23	17	1007
24	17	1008
25	16	109.92
26	21	109.92
27	15	11
28	16	119.95
29	21	119.95
30	15	12
31	20	12
32	14	12-Jul-99
33	14	12-Jun-99
34	12	13-q2/p2
35	14	14-Aug-99
36	14	14-Jul-99
37	12	14-q1/l3
38	16	14.4
39	16	14.99
40	21	14.99
41	15	15
42	12	1546-qq2
43	12	1558-qw1
44	14	16-Aug-99
45	16	17.49
46	15	172
47	15	18
48	18	18-Aug-99
49	15	188
50	14	19-Aug-99
51	18	19-Aug-99

52	14	19-Sep-99
53	19	2
54	20	2
55	13	2.5-in wd. Screw, 50
56	7	21225
57	7	21226
58	7	21231
59	7	21344
60	11	2158995
61	6	216.55
62	6	221.19
63	5	2221672
64	12	2232/qty
65	12	2232/qwe
66	11	2233234
67	12	2238/qpd
68	7	22567
69	11	2270093
70	11	2281410
71	11	2283245
72	15	23
73	14	23-Jul-99
74	14	23-Sep-99
75	12	23109-hb
76	12	23114-aa
77	7	23119
78	15	237
79	7	24004
80	7	24288
81	7	25443

82	7	25501
83	7	25595
84	16	256.99
85	21	256.99
86	14	28-Sep-99
87	14	29-Jul-99
88	5	2902556
89	5	2971228
90	5	2973809
91	19	3
92	20	3
93	21	3.95
94	15	32
95	6	345.86
96	16	38.95
97	21	38.95
98	5	3827185
99	16	39.95
100	21	39.95
101	19	4
102	16	4.99
103	21	4.99
104	15	43
105	16	43.99
106	5	4423381
107	11	4560092
108	20	5
109	16	5.87
110	21	5.87
111	6	536.75

112	12	54778-2t
113	15	6
114	16	6.99
115	21	6.99
116	4	615
117	10	615
118	11	6781419
119	11	6783998
120	13	7.25-in pwr saw blade
121	4	713
122	6	768.93
123	15	8
124	16	8.45
125	5	8442573
126	11	8892546
127	12	89-wre-q
128	11	8903529
129	5	8941238
130	5	8942180
131	5	8942285
132	11	8981234
133	13	9.00-in pwr saw blade
134	16	9.95
135	21	9.95
136	10	901
137	10	904
138	16	99.87
143	8	ORDVA, inc
144	13	PVC pipe 3.5-in 8-ft
145	3	Alfred

146	3	Amy
147	9	anderson
148	3	Anne
149	8	brackman bros
150	2	Brown
151	9	browning
152	8	Bryson,inc
153	13	claw hammer
154	8	Damal suppies
155	8	dome supply
156	2	Dunne
157	2	Farriss
158	9	flushing
159	3	george
160	8	Gomez Bros
161	9	hakford
162	13	hicut chain saw 16-in
163	13	hrd cloth, 1/2-in 3*50
164	3	James
165	3	Kathy
166	3	Leona
167	3	Myron
169	3	Olette
170	2	olowski
171	2	orlando
172	9	Ortega
173	9	Orton
174	3	Paul
175	12	pvc23drt
176	2	Ramas

177	8	randset Ltd
178	13	rat-tail file 1/8-in fine
179	8	rubicon sis
180	9	Singh
181	13	sledge hammer 12 lb
182	12	sm-18277
183	2	Smith
184	9	smithson
185	9	smythe
187	8	super logo,inc
188	12	sw-23116
189	2	williams
190	12	wr3/tt3
139	13	B&D cordless drill 1/2-in
140	13	B&D jigsaw 12-in blade
141	8	B&K,inc
142	8	D&E supply
168	2	o'brian
186	13	steel matting 4'*8'*1/6", .5"mesh

ภาคผนวก ค
โปรแกรม

BranchAndBound.java

```
import java.sql.*;
import java.util.*;
import uk.ac.liv.util.BinaryHeap;

class HeapComparator implements Comparator
{
        public int compare(Object O1,Object O2)
        {
                int r=0;
                HeapElement E1,E2;

                E1=(HeapElement)O1;
                E2=(HeapElement)O2;

                r=E1.getLB()-E2.getLB();

                return r;
        }
}

class HeapElement
{
        /*
         *R=Root
         *QI=Query Image
         *FG=Feasible Graph
```

```
*QG=Query Graph
*/
String type;
int lb;
Object Element=null;

HeapElement(DKeywordVertex KV) // text
{
        type="R";
        Element=KV;
        setLB(1);
}

HeapElement(DQI QI)
{
        type="QI";
        Element=QI;
}

HeapElement(DFG FG)
{
        type="FG";
        Element=FG;
}

HeapElement(DQG QG)
{
        type="QG";
        Element=QG;
}

HeapElement(String t,Object e)
```

```
{
        type=t;
        Element=e;
}

HeapElement(String t,Object e,int l)
{
        type=t;
        Element=e;
        lb=l;
}

int getLB()
{
        return lb;
}

void setLB(int l)
{
        lb=l;
}

boolean isTerminal()
{
        boolean r;

        if(type.equals("QG")) r=true;
        else r=false;

        return r;

}
```

```
}

class HeapElementSet
{
        Vector HeapList=new Vector();

        void add(HeapElement H)
        {
                HeapList.add(H);
        }

        HeapElement get(int i)
        {
                return (HeapElement)HeapList.get(i);
        }

        int size()
        {
                return  HeapList.size();
        }
}
class CBAndB
{
        private CMetaDatabase MetaDatabase;
        private BinaryHeap Heap=new BinaryHeap(new HeapComparator());

        CBAndB(CMetaDatabase M)
        {
                MetaDatabase=M;
        }

        private int determineLB(DQG QG)
```

```
{
        int lb=0,node=0;
        DEntPath EP;
        EntTree ET;

        ++node; // root Entity
        for(int i=0;i<QG.PathList.size();i++)
        {
                ++node; // Entity connecting with the other one
                EP=(DEntPath)QG.PathList.get(i);
                for(int j=0;j<EP.Path.size();j++)
                        {
                                ET=(EntTree)EP.Path.get(j);
                                lb+=determineLBEntTree(ET);
                        }
        }
        lb+=(node-1);

        return lb;
}

private int determineLB(DQI QI) throws Exception
{
        DVertex V;
        DVertexSet ER_Set=new DVertexSet();
        DVertexSet Counted_ER_Set=new DVertexSet();
        DVertexSet Attrib_Set=new DVertexSet();
        DVertexSet Attrib_ER_Set=new DVertexSet();
        DVertexSet Value_ER_Set=new DVertexSet();
        int count=0;
        String comm;
        ResultSet result;
```

```
for(int i=0;i<QI.size();i++)
{
        V=(DVertex)QI.get(i);

        if(V.type.equals("E"))
        {
                ER_Set.add(V);
        }
}

for(int i=0;i<QI.size();i++)
{
        V=(DVertex)QI.get(i);

        if(V.type.equals("A"))
        {
                Attrib_Set.add(V);
                ++count;
                Attrib_ER_Set.clear();
                comm="select "+MetaDatabase.ENT_ERNAME+
                        " from "+MetaDatabase.ENT_TABLE+
                " where "+MetaDatabase.ENT_ATTRIBCODE+" =
"+V.code;                                       result=MetaDatabase.query(comm);

                while(result.next())
                {
                        Attrib_ER_Set.add(new
                DVertex("E",result.getString(MetaDatabase.ENT_ERNAME)))
                ;
                }
```

```
if((!DVertexSet.existElement(Attrib_ER_Set,ER_Set))&&(!DVertexSet.existElement(At
trib_ER_Set,Counted_ER_Set)))
                    {

                        ++count;
                        if(!DVertexSet.existElement(Attrib_ER_Set,ER_Set))
                        {
                            DVertex X;
                            for(int j=0;j<Attrib_ER_Set.size();j++)
                            {

                                X=Attrib_ER_Set.get(j);

if(!DVertexSet.existElement(X,Counted_ER_Set))
                                Counted_ER_Set.add(X);
                                }
                            }
                        }
                    }
                else if(V.type.equals("V"))
                {
                    ++count;
                    Value_ER_Set.clear();

                    comm="select E."+MetaDatabase.ENT_ERNAME+
                        " from "+MetaDatabase.ENT_TABLE+" E,
"+MetaDatabase.VALUE_TABLE+
                        " V where V."+MetaDatabase.VALUE_VCODE+" =
"+V.code+
                        " and V."+MetaDatabase.ATTRIB_ATTRIBCODE+" =
E."+MetaDatabase.ENT_ATTRIBCODE;
```

```
result=MetaDatabase.query(comm);
while(result.next())
{
Value_ER_Set.add(new
DVertex("E",result.getString(MetaDatabase.ENT_ERNAME)));
}

if((!DVertexSet.existElement(Value_ER_Set,ER_Set))&&(!DVertexSet.existElement(V
alue_ER_Set,Counted_ER_Set)))
{
++count;
if(!DVertexSet.existElement(Value_ER_Set,ER_Set))
{
DVertex X;
for(int j=0;j<Value_ER_Set.size();j++)
{
X=Value_ER_Set.get(j);

if(!DVertexSet.existElement(X,Counted_ER_Set))
Counted_ER_Set.add(X);
}
}
}

DVertexSet Attrib=new DVertexSet();

comm="select "+MetaDatabase.VALUE_ATTRIBCODE+
" from "+MetaDatabase.VALUE_TABLE+
" where "+MetaDatabase.VALUE_VCODE+" = "+V.code;
```

```
                result=MetaDatabase.query(comm);
                while(result.next())
                {
                        Attrib.add(new
DVertex("A",result.getString(MetaDatabase.VALUE_ATTRIBCODE)));
                }

                if(!DVertexSet.existElement(Attrib,Attrib_Set))
                {
                        ++count;
                        Attrib_Set.add(Attrib);
                }
            }
        }
        return count+Math.abs(ER_Set.size())-1;
}

/*
 * Divided QI To 3 Vertex sets (Ent, Attrib and Vertex)
 *
 * @QI : converted source QI
 *
 * Jan 7, 2004 : first created
 */


/* count number of node in feasible graph*/

private int determineLBEntTree(EntTree EN)
{
        int lb=0,node=0;
```

```
        Enumeration EAT,EV;
        AttribVertexTree AT;

        EAT=EN.AttribTree.elements();
        ++node; //entity
        while(EAT.hasMoreElements())
        {
                ++node; //attrib
                AT=(AttribVertexTree)EAT.nextElement();
                EV=AT.Vertex.elements();
                while(EV.hasMoreElements())
                {
                        ++node; //value
                        EV.nextElement();
                }
        }
        lb=(node-1);

        return  lb;
}

private int determineLB(DFG FG)
{
        Enumeration EEN;
        EntTree EN;
        int lb=0;

        EEN=FG.Root.elements();
        while(EEN.hasMoreElements())
        {
                EN=(EntTree)EEN.nextElement();
                lb+=determineLBEntTree(EN);
```

```
        }

        return lb;
}

private void determineLB(HeapElement H) throws Exception
{
        String t;
        t=H.type;
        if(t.equals("R"))
        {
                H.setLB(1);
        }
        else if (t.equals("QI"))
        {
                H.setLB(determineLB((DQI)H.Element));
        }
        else if (t.equals("FG"))
        {
                H.setLB(determineLB((DFG)H.Element));
        }
        else // QG
        {
                H.setLB(determineLB((DQG)H.Element));
        }
}

private HeapElementSet branch(HeapElement H) throws Exception
{
        HeapElementSet HS=new HeapElementSet();
        HeapElement HE;
        String t;
```

```
t=H.type;

if(t.equals("R"))
{
 DKeywordVertex ResultKeyword=(DKeywordVertex)H.Element;
 DQISet ResultQI;
 CQueryImage GetQI=new CQueryImage();
 DQI QI;

 GetQI.input(ResultKeyword);
 ResultQI=GetQI.findQI();

 for(int i=0;i<ResultQI.size();i++)
 {
        QI=(DQI)ResultQI.get(i);
        HE=new HeapElement(QI);
        HE.setLB(determineLB(QI));

        HS.add(HE);
 }
}
else if (t.equals("QI"))
{
        CFG GetFG=new CFG(MetaDatabase);
        DQI QI=(DQI)H.Element;
        DFG FG;
        DFGSet FGS;

        FGS=GetFG.createFG(QI);
        for(int i=0;i<FGS.size();i++)
 {
```

```
                FG=(DFG)FGS.get(i);
                HE=new HeapElement(FG);
                HE.setLB(determineLB(FG));

                HS.add(HE);
            }
        }
        else if (t.equals("FG"))
        {
                CSPEnt SP=new CSPEnt(MetaDatabase.getStmt());
                DFG FG=(DFG)H.Element;
                DQGSet QGS;
                DQG QG;

                QGS=SP.getSP(FG);
                for(int i=0;i<QGS.size();i++)
                {
                    QG=(DQG)QGS.get(i);
                    HE=new HeapElement(QG);
                    HE.setLB(determineLB(QG));
                    HS.add(HE);
                }
        }
        return HS;
}

public DQGSet BBSearch(DKeywordVertex KV) throws Exception
{
        int current_best_LB=Integer.MAX_VALUE;
        int min_successor_LB;
        int current_LB;
        HeapElement current_vertex,successor;
```

```
HeapElementSet successor_set;
DQGSet QGS=new DQGSet();

Heap.add(new HeapElement(KV));
current_vertex=(HeapElement)Heap.removeFirst();
current_LB=current_vertex.getLB();
while(current_best_LB>=current_LB)
{
        min_successor_LB=Integer.MAX_VALUE;
        successor_set=branch(current_vertex);
        for(int i=0;i<successor_set.size();i++)
        {
            successor=(HeapElement)successor_set.get(i);
            if(!successor.isTerminal())
            {
                Heap.add(successor);
            }
            if(successor.getLB()<min_successor_LB)
            {
                min_successor_LB=successor.getLB();
            }
        }
        for(int i=0;i<successor_set.size();i++)
        {
            successor=(HeapElement)successor_set.get(i);
            if(min_successor_LB==successor.getLB())
            {
                if(!successor.isTerminal())
                {
                    current_LB=min_successor_LB;
                }
```

```
                else
                {
                    if(min_successor_LB<=current_best_LB)
                    {

if(min_successor_LB<current_best_LB)
                        {
                            QGS.clear();
                        }
                        QGS.add((DQG)successor.Element);
                        current_best_LB=min_successor_LB;
                    }
                }
            } //end for
            if(Heap.size()!=0) // not empty
            {
                current_vertex=(HeapElement)Heap.removeFirst();
                if(current_vertex.getLB()>current_LB)
                {
                    current_LB=Integer.MAX_VALUE;
                }
            }
            else
            {
                current_LB=Integer.MAX_VALUE;
            }
        }//end while

        return QGS;
    }
}
```

CfindQG.java

```
/* Data structure collecting List of
 * Keyword with its vertex set
 */

public class CFindQG
{
        DKeywordVertex KV;

        CMetaDatabase MetaDatabase=new CMetaDatabase();

        boolean isRecognised(String freeText) throws Exception
        {

                CKeyword KeyWord=new CKeyword(MetaDatabase);
                KV=KeyWord.getKeywordVertex(freeText);
                return(KV.size()!=0);
        }
        public String getSource(String freeText) throws Exception
        {
                CMainDatabase MainDatabase=new CMainDatabase();
                String source;
                MetaDatabase.connect();

                CBAndB BB=new CBAndB(MetaDatabase);
                DQGSet QGS=null;

                if(freeText!=null)
                {
                if((freeText.length()!=0)&&(isRecognised(freeText)))
                        QGS=BB.BBSearch(KV);
```

```
            }
            MainDatabase.connect();

            CGenerateHTML GenerateHTML=new
            CGenerateHTML(MetaDatabase,MainDatabase,QGS,freeText);
            source=GenerateHTML.getHTML();

            MetaDatabase.close();
            MainDatabase.close();
            return source;
        }
}
```

DataClass.java

```
import java.util.*;

// data structure representing vertex in database graph
class DVertex implements Cloneable
{
        String type; //E:Entity, A:Attribute and V:Value
        String data;
        String code;

        DVertex(){}

        DVertex(String t,String c)
        {
            set(t,c);
        }

        DVertex(String t,String c,String d)
        {
```

```
        set(t,c,d);
}

void set(String t,String c,String d)
{
        type=t;
        data=d;
        code=c;
}

void set(String t,String c)
{
        type=t;
        code=c;
        data=code;
}
void set(DVertex V)
{
        set(V.type,V.code,V.data);
}

String getCode()
{
        return code;
}

void show()
{
        System.out.print("("+type+":"+code+":"+data+")");
}

boolean equals(DVertex x)
```

```
        {

                if((x.code.equals(this.code)) && (x.type.equals(this.type)))
                        return true;
                else return false;

        }

        protected Object clone()
        {
                DVertex V=null;

                try
                {
                        V=(DVertex)super.clone();
                }catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }
                return V;
        }
}

class DVertexSet implements Cloneable
{
        Vector VertexSet=new Vector();

        DVertexSet()
        {
        }
```

```
DVertexSet(DVertex V)
{
        add(V);
}

void add(DVertex vertex)
{
        VertexSet.add(vertex );
}

void add(DVertexSet set)
{
        for(int i=0;i<set.size();i++)
                VertexSet.add(set.get(i));
}

int size()
{
        return VertexSet.size();
}

DVertex get(int i)
{
        return (DVertex)VertexSet.get(i);
}

void show()
{

        System.out.print("{ ");
```

```
        for(int i=0;i<size();i++)
                {
                        get(i).show();
                        System.out.print(", ");
                }
                System.out.println("} ");
}

void clear()
{
        VertexSet.clear();
}

DVertex remove(int i)
{
        return (DVertex)VertexSet.remove(i);
}

static boolean existElement(DVertexSet x,DVertexSet y) //if any elements of x in y
{
        boolean found=false;
        DVertex A,B;
        int i,j;


        i=0; j=0;
        while(i<x.size() && !found)
        {
                A=x.get(i);
                while(j<y.size() && !found)
                {
                        B=y.get(j);
```

```
                    if(A.equals(B)) found=true;
                    j++;
                }
                i++;
        }
        return found;
}

static DVertexSet searchElement(DVertexSet x,DVertexSet y) //if any elements of x in y
{
        DVertex A,B;
        DVertexSet DV=new DVertexSet();
        int i,j;

        i=0; j=0;
        while(i<x.size())
        {
                A=x.get(i);
                while(j<y.size())
                {
                        B=y.get(j);
                        if(A.equals(B)) DV.add(B);
                        j++;
                }
                i++;
        }
        return DV;
}

static boolean existElement(DVertex x,DVertexSet y) //if element x in y
{
        boolean found=false;
```

```
        DVertex A;
        int i;

        i=0;
        while(i<y.size() && !found)
        {
                A=y.get(i);

                if(A.equals(x)) found=true;

                i++;
        }
        return found;
}

protected Object clone()
{
        Enumeration E;
        DVertexSet VS=null;
        DVertex V;

        try
        {
                VS=(DVertexSet)super.clone();
                VS.VertexSet=(Vector)VertexSet.clone();


        }catch(Exception e)
        {
                System.out.println(e.getMessage());
        }
```

```
            return VS;

        }

}

class AttribVertexTree implements Cloneable
{
        DVertex Attrib;
        Hashtable Vertex=new Hashtable();

        boolean isLeaf()
        {
            return (Vertex.size()==0);
        }

        void setAttrib(DVertex Att)
        {
            Attrib=Att;
        }

        DVertex getAttrib()
        {
            return Attrib;
        }

        String getAttribCode()
        {
            return Attrib.getCode();
        }
```

```
void addVertex(DVertex Ver)
{
        if(Vertex==null) Vertex=new Hashtable();
        if(!Vertex.containsKey(Ver.getCode())) Vertex.put(Ver.getCode(),Ver);
}

void addVertex(DVertexSet VerSet)
{
        DVertex V;
        for(int i=0;i<VerSet.size();i++)
        {
                V=VerSet.get(i);
                if(!Vertex.containsKey(V.getCode())) Vertex.put(V.getCode(),V);
        }
}
void show()
{
        Enumeration E;
        System.out.print(" +-");
        Attrib.show();
        System.out.println();
        DVertex V;

        E=Vertex.elements();
        while(E.hasMoreElements())
        {
                System.out.print("  +-");
                V=(DVertex)E.nextElement();
                V.show();
                System.out.println();
        }
 }
```

```
        protected Object clone()
        {
                AttribVertexTree AVT=null;
                DVertex V;

                try{
                        AVT=(AttribVertexTree)super.clone();
                        AVT.Vertex=(Hashtable)Vertex.clone();
                        AVT.Attrib=Attrib;
                }catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }
                return AVT;
        }
}

class AttribVertexTreeRoot implements Cloneable
{
        Hashtable Root=Root=new Hashtable();

        void addAttribVertexTree(AttribVertexTree AVT)
        {
                Root.put(AVT.getAttribCode(),AVT);
        }
        void addLeafAttrib(DVertex Att)
        {
                AttribVertexTree AVT=new AttribVertexTree();
                AVT.setAttrib(Att);
```

```
        addAttribVertexTree(AVT);
}

AttribVertexTree getAttribVertexTree(String key)
{
        return (AttribVertexTree)Root.get(key);
}

boolean existAttrib(DVertex Att)
{
        return Root.containsKey(Att.getCode());
}

void addAttribVertex(DVertex Att,DVertexSet VerSet)
{
        if(Root.containsKey(Att.getCode())) // exist Attrib
        {
                AttribVertexTree target;
                target=(AttribVertexTree)Root.get(Att.getCode());
                target.addVertex(VerSet);
        }
        else // no exist Attrib so create new one
        {
                AttribVertexTree NewAttrib=new AttribVertexTree();
                NewAttrib.setAttrib(Att);
                NewAttrib.addVertex(VerSet);
                addAttribVertexTree(NewAttrib);
        }
}

void addAttribVertex(DVertexSet Att,DVertex Ver)
```

```
{
        DVertex A;
        for(int i=0;i<Att.size();i++)
        {
                A=Att.get(i);
                addAttribVertex(A,Ver);
        }
}

void addAttribVertex(DVertex Att,DVertex Ver)
{
        addAttribVertex(Att,new DVertexSet(Ver));
}

void show()
{
        Enumeration E;
        AttribVertexTree AVT;

        System.out.println("AVT root");

        E=Root.elements();

        while(E.hasMoreElements())
        {
                AVT=(AttribVertexTree)E.nextElement();
                AVT.show();
        }
}

protected Object clone()
{
```

```
            AttribVertexTree AVT;
            AttribVertexTreeRoot AVTR=null;
            try{
                    AVTR=(AttribVertexTreeRoot)super.clone();
                    AVTR.Root=(Hashtable)Root.clone();
            }catch(Exception e)
            {
                    System.out.println(e.getMessage());
            }
            return AVTR;
        }
}

class EntTree implements Cloneable
{
        DVertex Ent;
        Hashtable AttribTree=new Hashtable();

        void setEnt(DVertex E)
        {
                Ent=E;
        }

        boolean isLeaf()
        {
                return (AttribTree.size()==0);
        }

        DVertex getEnt()
        {
                return Ent;
        }
```

```
String getCode()
{
        return Ent.getCode();
}

void addAttribTree(AttribVertexTree AVT)
{
        if(!AttribTree.containsKey(AVT.getAttribCode()))
AttribTree.put(AVT.getAttribCode(),AVT);
}
void show()
{
        Enumeration E;
        AttribVertexTree V;

        Ent.show();
        System.out.println();

        E=AttribTree.elements();

        while(E.hasMoreElements())
        {
                V=(AttribVertexTree)E.nextElement();
                V.show();
        }

}

protected Object clone()
{
```

```
            AttribVertexTree AVT;
            EntTree ET=null;

            try{
                ET=(EntTree)super.clone();
                ET.AttribTree=(Hashtable)AttribTree.clone();
            }catch(Exception e)
            {
                System.out.println(e.getMessage());
            }
            return ET;
        }
}

class DFG implements Cloneable
{
        Hashtable Root=new Hashtable();

        EntTree getEntTree(String key)
        {
            return (EntTree)Root.get(key);
        }

        void addLeafEnt(DVertex Ent)
        {
            addEntTree(makeLeafEnt(Ent));
        }

        EntTree makeLeafEnt(DVertex Ent)
        {
            EntTree ET=new EntTree();
```

```
        ET.setEnt(Ent);
        return ET;
}

void addEntTree(EntTree ET)
{
        if(!Root.containsKey(ET.getCode())) Root.put(ET.getCode(),ET);
}

boolean existEnt(DVertex Ent)
{
        return Root.containsKey(Ent.getCode());
}

void addEntAttrib(DVertex Ent,AttribVertexTree AttribTree)
{
        if(Root.containsKey(Ent.getCode())) // exist Attrib
        {
                EntTree target;
                target=(EntTree)Root.get(Ent.getCode());
                target.addAttribTree(AttribTree);
        }
        else // not exists Attrib so create new one
        {
                EntTree NewDFG=new EntTree();
                NewDFG.setEnt(Ent);
                NewDFG.addAttribTree(AttribTree);
                addEntTree(NewDFG);
        }
}

void addEntAttrib(DVertexSet ES,AttribVertexTree AttribTree)
```

```
{
        DVertex E;
        for(int i=0;i<ES.size();i++)
        {
                E=(DVertex)ES.get(i);
                addEntAttrib(E,AttribTree);
        }
}

void show()
{
        Enumeration E;
        EntTree ET;

        E=Root.elements();
        while(E.hasMoreElements())
        {
                ET=(EntTree)E.nextElement();
                System.out.println("Graph");
                ET.show();
        }
}

protected Object clone()
{
        EntTree ET;

        DFG nFG=null;
        try{
                nFG=(DFG)super.clone();
                nFG.Root=(Hashtable)Root.clone();
        }catch(Exception e)
```

```
                {
                        System.out.println(e.getMessage());
                }

                return nFG;
        }
}

class DEntPath
{
        Vector Path=new Vector();

        void add(EntTree ET)
        {
                Path.add(ET);
        }

        EntTree get(int i)
        {
                return (EntTree)Path.get(i);
        }

        void show()
        {
                EntTree ET;
                for(int i=0;i<Path.size();i++)
                {
                        ET=(EntTree)Path.get(i);
                        ET.show();
                }
        }
```

```
        int size()
        {
                return Path.size();
        }
}


class DQGSet
{
        Vector QGS=new Vector();

        int size()
        {
                return QGS.size();
        }

        DQG get(int i)
        {
                return (DQG)QGS.get(i);
        }

        void show()
        {
                for(int i=0;i<size();i++)
                {
                        System.out.println("QG "+i);
                        get(i).show();
                }

        }
        void add(DQG V)
```

```
        {
                QGS.add(V);
        }

        void clear()
        {
                QGS.clear();
        }
}

class DQG
{
        EntTree mainEnt;
        Vector PathList=new Vector();

        void add(DEntPath Path)
        {
                PathList.add(Path);
        }

        void setMain(EntTree E)
        {
                mainEnt=E;
        }

        void show()
        {
                System.out.print("Root Entity : ");
                mainEnt.show();

                for(int i=0;i<PathList.size();i++)
                {
```

```
                    DEntPath Path;
                    System.out.println("path : "+i);
                    Path=(DEntPath)PathList.get(i);
                    Path.show();
                }
            }
        }

class DVertexPacket implements Cloneable
{
        DVertexSet Ent=new DVertexSet();
        DVertexSet Attrib=new DVertexSet();
        DVertexSet Value=new DVertexSet();

        void setEnt(DVertexSet E)
        {
            Ent=E;
        }

        void addAttrib(DVertex A)
        {
            Attrib.add(A);
        }

        void addValue(DVertex V)
        {
            Value.add(V);
        }
        void addEnt(DVertex E)
        {
            Ent.add(E);
        }
```

```
int sizeEnt()
{
        return Ent.size();
}
int sizeAttrib()
{
        return Attrib.size();
}
int sizeValue()
{
        return Value.size();
}

DVertex removeValue()
{
        return (DVertex)Value.remove(0);
}

DVertex removeAttrib()
{
        return (DVertex)Attrib.remove(0);
}

DVertex removeEnt()
{
        return (DVertex)Ent.remove(0);
}

protected Object clone()
{
        DVertexPacket VP=null;
```

```
            try{
                    VP=(DVertexPacket)super.clone();
                    VP.Ent=(DVertexSet)Ent.clone();
                    VP.Attrib=(DVertexSet)Attrib.clone();
                    VP.Value=(DVertexSet)Value.clone();
            }catch(Exception e)
            {
                    System.out.println(e.getMessage());
            }
            return VP;
        }
        void show()
        {
            Ent.show();
            Attrib.show();
            Value.show();
        }
}

class DFGSet
{
        Vector DFGList=new Vector();

        void add(DFG D)
        {
            DFGList.add(D);
        }

        void add(DFGSet set)
        {
            for(int i=0;i<set.size();i++)
                DFGList.add(set.get(i));
```

```
    }

    int size()
    {
        return DFGList.size();
    }

    DFG get(int i)
    {
        return (DFG)DFGList.get(i);
    }

    void show()
    {
        DFG D;

        for(int i=0;i<DFGList.size();i++)
        {
            System.out.println("*****Feasible Graph *****");
            D=(DFG)DFGList.get(i);
            D.show();
        }
    }

}

class ProcessFGPacket implements Cloneable
{
    final static int VP=1;
    final static int AP=1;
    final static int EP=1;
```

```
DFG FG=new DFG();
AttribVertexTreeRoot AttribVertex=new AttribVertexTreeRoot();
DVertexPacket VertexPacket=null;

boolean done=false;

void setVertexSet(DVertexPacket VP)
{
        VertexPacket=VP;
}

DVertex nextVertex()
{
        DVertex V;
        if(VertexPacket.sizeValue()!=0)
        {
                V=(DVertex)VertexPacket.removeValue();
        }
        else if(VertexPacket.sizeAttrib()!=0)
        {
                V=(DVertex)VertexPacket.removeAttrib();
        }
        else if(VertexPacket.sizeEnt()!=0)
        {
                V=(DVertex)VertexPacket.removeEnt();
        }
        else V=null;

        return V;
}
```

```
        protected Object clone()
        {
                ProcessFGPacket P=null;
                try{

                P=(ProcessFGPacket)super.clone();
                P.FG=(DFG)FG.clone();
                P.AttribVertex=(AttribVertexTreeRoot)AttribVertex.clone();
                P.VertexPacket=(DVertexPacket)VertexPacket.clone();
                }catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }
                return P;
        }
}

class DKeywordVertex
{
        Vector KeywordList=new Vector();
        Vector VertexList=new Vector();

        /* get Keyword from Keyword List
         * @i : integer indicating position of Keyword in Keyword List
         */
        String getKeyword(int i)
        {
                return (String)KeywordList.get(i);
        }

        /* get Vertex set of keyword
         *@i : position of keyword the Vertex set is
```

```
*/
DVertexSet getVertexSet(int i)
{
        return (DVertexSet)VertexList.get(i);
}

/* add Keyword and its Vertex set
 * @keyword : added keyword
 * @vertexSet : Vertex set corresponding keyword
 */
void add(String keyword,DVertexSet vertexSet)
{
        KeywordList.add(keyword);
        VertexList.add(vertexSet);
}

/* get number of Keyword
 */
int size()
{
        return KeywordList.size();
}

/* show data about keyword and its Vertex set
 * to output console
 */
void show()
{
        for(int i=0;i<KeywordList.size();i++)
        {
                String key;
                DVertexSet VertexSet;
```

```
                key=(String)KeywordList.get(i);
                VertexSet=(DVertexSet)VertexList.get(i);

                System.out.println("key -> "+key);
                VertexSet.show();
            }
        }
}

/* data structure of Query Image */
class DQI extends DVertexSet
{
        int LB;

}

class DQISet
{
        Vector QIS=new Vector();

        int size()
        {
                return QIS.size();
        }

        DQI get(int i)
        {
                return (DQI)QIS.get(i);
        }

        void show()
```

```
        {
                for(int i=0;i<size();i++)
                {
                        System.out.println("QI "+i);
                        get(i).show();
                }

        }
        void add(DQI V)
        {
                QIS.add(V);
        }
}
```

Database.java

```
import java.sql.*;

interface CUseMetaDatabase
{
        /*
         * set the target to passed Database
         *
         * @CMD : source MetaDatabase
         **/
        public void setDatabase(CMetaDatabase CMD);
}

class CDatabase
{
        protected Connection conn;
        protected Statement stmt;
        protected boolean connected=false;
```

```
void connect(String con) throws Exception //connect to Oracle's database via JDBC
{
        Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
        conn=DriverManager.getConnection
                (con);

        //Create a statement
        stmt=conn.createStatement();
        connected=true;
}

public Statement getStmt()
{
    return stmt;
}

ResultSet query(String q) throws Exception
{
    ResultSet result=null;

    System.out.println(q);
    result=stmt.executeQuery(q);

    return result;
}

private String convertSQuote(String in)
{
    String out=new String();
    return out;
}
```

```
        boolean isConnected()
        {
                return connected;
        }

        void close() throws Exception
        {
                        stmt.close();
                        conn.close();
        }
}


// for connect to MetaData and providing proper function for MetaDatabase work
class CMetaDatabase extends CDatabase
{
        final static String VALUE_TABLE="value";
        final static String VALUE_VCODE = "vcode";
        final static String VALUE_DATA = "data";
        final static String VALUE_ATTRIBCODE = "attribcode";

        final static String ATTRIB_TABLE = "attrib";
        final static String ATTRIB_ATTRIBNAME = "attribname";
        final static String ATTRIB_ATTRIBCODE = "attribcode";


        final static String ENT_TABLE = "entity";
        final static String ENT_ERNAME = "ername";
        final static String ENT_ATTRIBCODE = "attribcode";

        void connect() throws Exception
```

```
        {
                super.connect("jdbc:oracle:thin:project/project@localhost:1521:project");
        }
}

class CMainDatabase extends CDatabase
{
        void connect() throws Exception
        {
                super.connect("jdbc:oracle:thin:company/company@localhost:1521:project");
        }
}
```

FG.java

```
import java.sql.*;

class CFG implements CUseMetaDatabase
{
        CMetaDatabase MetaDatabase;

        CFG(CMetaDatabase CMD)
        {
                setDatabase(CMD);
        }

        public void setDatabase(CMetaDatabase CMD)
        {
                MetaDatabase=CMD;
        }

        DVertexPacket QI2Vertex(DQI QI)
        {
```

```
        DVertex V;
        DVertexPacket VPack=new DVertexPacket();
        String type;
        for(int i=0;i<QI.size();i++)
        {

                V=QI.get(i);
                type=V.type;
                if(type.equals("E")) VPack.Ent.add(V);
                else if(type.equals("A")) VPack.Attrib.add(V);
                else if(type.equals("V")) VPack.Value.add(V);

        }
        return VPack;
}

/* Create FG  from QI
 * @QI : QI converted to FG
 *
 * Jan 7, 2004
 */
DFGSet createFG(DQI QI) throws Exception
{
        ProcessFGPacket FGP;
        DVertexPacket VertexSet;
        DFGSet DFGS;

        VertexSet=QI2Vertex(QI);
        FGP=new ProcessFGPacket();
        FGP.setVertexSet(VertexSet);
        DFGS=exploreFGP(FGP);
```

```
        return DFGS;

}

static int run;
static int ans=0;

DFGSet exploreFGP(ProcessFGPacket FGP) throws Exception
{
        DVertex V;
        ResultSet result;
        String comm;

        DVertexSet DV=new DVertexSet();
        DFGSet FGSet=new DFGSet();
        DFGSet exFGS;
        ProcessFGPacket exFGP;

        while((V=FGP.nextVertex())!=null)
        {
                if(V.type.equals("V"))
                {
                DVertexSet Attrib;
                Attrib=FGP.VertexPacket.Attrib;

                comm="select
"+MetaDatabase.ATTRIB_TABLE+"."+MetaDatabase.VALUE_ATTRIBCODE+

","+MetaDatabase.ATTRIB_TABLE+"."+MetaDatabase.ATTRIB_ATTRIBNAME+
                        " from "+MetaDatabase.VALUE_TABLE+
                        ","+MetaDatabase.ATTRIB_TABLE+
```

```
" where "+MetaDatabase.VALUE_VCODE+" = "+V.code+"
and "+
MetaDatabase.ATTRIB_TABLE+"."+MetaDatabase.ATTRIB_ATTRIBCODE+"="+
MetaDatabase.VALUE_TABLE+"."+MetaDatabase.VALUE_ATTRIBCODE;
                result=MetaDatabase.query(comm);
                DV.clear();
                    while(result.next())
                    {
                        DV.add(new
DVertex("A",result.getString(MetaDatabase.VALUE_ATTRIBCODE),result.getString(MetaData
base.ATTRIB_ATTRIBNAME)));
                    }
                if(DVertexSet.existElement(DV,Attrib))
                {
                    DVertexSet matchSet;
                    matchSet=DVertexSet.searchElement(DV,Attrib);
                    FGP.AttribVertex.addAttribVertex(matchSet,V);
                }
                else
                {
                    for(int i=0;i<DV.size();i++)
                    {
                     DVertex newAtt;
                     newAtt=DV.get(i);
```

```
                    exFGP=(ProcessFGPacket)FGP.clone();
                        exFGP.VertexPacket.addAttrib(newAtt);

exFGP.AttribVertex.addAttribVertex(newAtt,V);
                    exFGS=exploreFGP(exFGP);

                    FGSet.add(exFGS);
                    }
                    FGSet.size();
                    return FGSet;
                } //end else
            } // end if

            else if(V.type.equals("A"))
            {
                DVertexSet Ent;
                AttribVertexTree AVT;

                Ent=FGP.VertexPacket.Ent;

AVT=FGP.AttribVertex.getAttribVertexTree(V.getCode());

                if(AVT==null) // not yet added attrib
                {
                    AVT=new AttribVertexTree();

                    AVT.setAttrib(V);
                    FGP.AttribVertex.addAttribVertexTree(AVT);
                }

               comm="select "+MetaDatabase.ENT_ERNAME+
                    " from "+MetaDatabase.ENT_TABLE+
```

```
                    " where "+MetaDatabase.ENT_ATTRIBCODE+" =
"+V.code;

                result=MetaDatabase.query(comm);
                    DV.clear();
                    while(result.next())
                    {
                    DV.add(new
DVertex("E",result.getString(MetaDatabase.ENT_ERNAME)));
                    }

                if(DVertexSet.existElement(DV,Ent))
                {
                    DVertexSet matchSet;

                    matchSet=DVertexSet.searchElement(DV,Ent);
                    FGP.FG.addEntAttrib(matchSet,AVT);
                }
                else
                {
                    for(int i=0;i<DV.size();i++)
                    {
                    DVertex newEnt;
                    newEnt=DV.get(i);

                    exFGP=(ProcessFGPacket)FGP.clone();

                    exFGP.VertexPacket.addEnt(newEnt);

                    exFGP.FG.addEntAttrib(newEnt,AVT);
```

```
                        exFGS=exploreFGP(exFGP);
                        FGSet.add(exFGS);
                    }
                    return FGSet;
                } //end else

            } // end if

            else //Entity
            {
                EntTree ET;
                DVertexSet Ent;

                Ent=FGP.VertexPacket.Ent;
                ET=FGP.FG.getEntTree(V.getCode());

                if(ET==null)
                {
                    FGP.FG.addLeafEnt(V);
                }
            }

        } // end for

        FGSet.add(FGP.FG);
        ++ans;
        return FGSet;

    }
}
```

GenerateHTML.java

```
import java.sql.*;
```

```
import java.util.*;
import java.io.*;

class CGenerateHTML implements CUseMetaDatabase
{
        DQGSet QGS;
        String source="";
        String title;
        CMainDatabase MainDatabase;
        CMetaDatabase MetaDatabase;
        String query="",column="",table="",where="";

        CGenerateHTML(CMetaDatabase Me,CMainDatabase Ma,DQGSet Q,String t)
        {
                QGS=Q;
                title=t;
                MainDatabase=Ma;
                MetaDatabase=Me;
        }

        private String makeValue(String attribCode,String value) throws Exception
        {
                String v,format="";
                ResultSet r=null;

                        r=MetaDatabase.query("select iformat from attrib where attribcode
 ='"+attribCode+"'");
                        while(r.next()) format=r.getString("iformat");

                if(!format.equals("N")) v="'"+value+"'";
                else v=value;
```

```
        return v;
}

private void generateQuery(EntTree ET) throws Exception
{
        String cColumn="",cTable="";
        DVertex Ent;
        AttribVertexTree AT;
        DVertex V;
        Enumeration EA,EV;

        Ent=ET.Ent;
        cTable=Ent.data;
        table+=(", "+cTable);

        column+=(", "+cTable+".*");

        EA=ET.AttribTree.elements();
        while(EA.hasMoreElements()) //Attrib Vertex Tree
        {
                AT=(AttribVertexTree)EA.nextElement();
                column+=(", "+cTable+"."+AT.Attrib.data);
                if(!AT.isLeaf()) //not Leaf
                {
                        EV=AT.Vertex.elements();
                        where+=" and (";
                        while(EV.hasMoreElements())
                        {
                                V=(DVertex)EV.nextElement();

where+=(cTable+"."+AT.Attrib.data+"="+makeValue(AT.Attrib.code,V.data)+" OR ");
```

```
                    }
                    where=where.substring(0,where.length()-4);
                    where+=") ";
                }
            }
        }

        private String findJoinCondition(String Ent1,String Ent2) throws Exception
        {
            ResultSet r=null;
            String attribname="";
            String condition="";

            r=MetaDatabase.query("select attribname from attrib where attribcode in" + //get
attribname
                "(select attribcode from entity "+ // join attribcode
                " where ername='"+Ent1+"' and attribcode in "+
                "( select attribcode from entity where ername='"+Ent2+"'))");
                while(r.next())
                {

                attribname=r.getString("attribname");
                condition+=" and "+Ent1+"."+attribname+"="+Ent2+"."+attribname;
                }

            return condition;
        }

        private ResultSet queryData(DQG QG) throws Exception
        {
            ResultSet r;
            DEntPath Path;
```

```
EntTree ET,lastET=null;

//main node
ET=QG.mainEnt;
generateQuery(ET);

for(int i=0;i<QG.PathList.size();i++) //each line
{
        Path=(DEntPath)QG.PathList.get(i);
        ET=QG.mainEnt;

        for(int j=0;j<Path.size();j++) //each column
        {
            lastET=ET;
            ET=(EntTree)Path.get(j);

where+=findJoinCondition(lastET.Ent.getCode(),ET.Ent.getCode());
            generateQuery(ET);
        }
}

if(column.length()!=0)
        column=column.substring(2);
 if(where.length()!=0)
        where=where.substring(5);
 if(table.length()!=0)
        table=table.substring(2);

query="select "+column+" from "+table;
if(where.length()!=0) query+=(" where "+where);
```

```
        System.out.println("Query : "+query);
        r=MainDatabase.query(query);

        return r;
}

private String generateTable(DQG QG) throws Exception
{
        String table,table2="";
        int numCol;
        ResultSet r;
        ResultSetMetaData rm;

        r=queryData(QG);

        table="<B>Query :</B> "+query+"<BR><table border=\\\"1\\\">";

        rm=r.getMetaData();
        table+="<BR><tr bgcolor=\\\"#CCCCFF\\\">";

        numCol=rm.getColumnCount();
        for(int i=0;i<numCol;i++)
                table2+="<td>"+rm.getColumnLabel(i+1)+"</td>";

        table2+="</tr>";
                while(r.next())
                {
                        table2+="<tr>";
                        for(int i=0;i<numCol;i++)
                        {
                                table2+=("<td>"+r.getString(i+1)+"</td>");
```

```
                }
                table2+="</tr>";
            }

        table+=this.convertDQuote(table2);
        table+="</table><HR>";

        return table;
    }

    public String getHTML() throws Exception
    {
    String win;
    source+="<html>"+
                "<head>"+
                "<title>Query</title>"+
                "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-
8859-1\">";
                if(title!=null)
                {
                source+="<script language=\"JavaScript\">"+
    "QueryWin=open(\"\",\"win\",\"scrollbars=yes,resizable=yes,toolbar=no,menubar=no,wi
dth=500,height=400\");" +
                "QueryWin.document.write(\"";

                System.out.println("Text : "+title);
                win=windowQuery();
                source+=win;
                source+="\");"+
                        "</script>";
                }
                source+="</head>"+
```

```
                "<body bgcolor=\"#FFFFFF\">"+
                "<p align=\"center\">Please enter keywords </p>"+
                "<form method=\"get\"
action=\"http://161.246.13.231:8081/servlet/search\" name=\"search\">"+
                "<p align=\"center\"> keywords : "+
              "<input type=\"text\" name=\"keyword\" size=\"70\">"+
                "</p>"+
                "<p align=\"center\"> "+
              "<input type=\"submit\" name=\"Query\" value=\"Query\">"+
              "<input type=\"reset\" name=\"Clear\" value=\"Clear\">"+
                "</p>"+
                "</form>"+
                "</body>"+
                "</html>";

        return source;
}

public String windowQuery() throws Exception
{
        DQG QG;

        MainDatabase.connect();
        MetaDatabase.connect();
        //header
        String source="";
        source+="<html><head><title>"+title+"</title>"+
                        "<meta http-equiv=\\\"Content-Type\\\" content=\\\"text/html;
"+
                        "charset=iso-8859-1\\\"></head> "+
                        "<body bgcolor=\\\"#FFFFFF\\\">";
        //body
```

```
if(title.length()==0)
{
        source+="Blank text is not be allowed.";
}
else if(QGS==null)
{
        source+="There are not any recognised keywords.";
}
else
{
source+="<B>Text :</B> "+title;
source+="<BR>There is/are ";
source+=QGS.size();
source+=" solution(s).<HR>";
for(int i=0;i<QGS.size();i++)
{
        QG=(DQG)QGS.get(i);
        this.query="";this.column="";this.table="";this.where="";
        source+=("<B>Solution :</B> "+(i+1)+"<BR>");
        source+=generateTable(QG);
}
}
//tail
source+="</body></html>";

MainDatabase.close();
MetaDatabase.close();

return source;
}
```

```
    private String convertDQuote(String in)
    {
        String out=new String();
        char ch;

        for(int i=0;i<in.length();i++)
        {
            ch=in.charAt(i);

            if(ch=="") out+="\\\"";
            //else if (ch=='\') out+="\\";
            else out+=String.valueOf(ch);
        }
        return out;
    }
}
```

Keyword.java

```
import java.sql.*;

class CKeyword implements CUseMetaDatabase
{
    DVertexSet VertexSet;
    CMetaDatabase MetaDatabase;

    private String text; // source natural text

    private String words[]; // group of word after splitted by space

    CKeyword(CMetaDatabase CMD)
    {
        setDatabase(CMD);
```

```
}

public void setDatabase(CMetaDatabase CMD)
{
        MetaDatabase=CMD;
}

/* input natural text
 * @input : source nutural text
 */
void getInput(String input)
{
        text=input;
}

/* split the text to array of word
 */
private void splitText()
{
        words=text.split(" ");
}

/* determine keyword & its vertex set from the text
 * @ input : nutalral text
 */
DKeywordVertex getKeywordVertex(String input) throws Exception
{

        DVertexSet vs;
        DKeywordVertex kv=new DKeywordVertex();

        text=input;
```

```
            splitText();
            for(int i=0;i<words.length;i++)
            {
                    vs=findVertexSet(words[i]);
                    if(vs.size()!=0) kv.add(words[i],vs);
            }
            return kv;
    }

    /* find VertexSet for a keyword
     * @key : keyword
     */
    DVertexSet findVertexSet(String key) throws Exception
    {
            ResultSet result;
            DVertex Vertex;
            String comm;
            VertexSet=new DVertexSet();

            //explore in value
            key=key.replaceAll("","");
                    comm="select "+MetaDatabase.VALUE_DATA+", "+
                            MetaDatabase.VALUE_VCODE+" from
"+MetaDatabase.VALUE_TABLE+
                            " where upper('"+key+"') like
upper("+MetaDatabase.VALUE_DATA+")";

                    result=MetaDatabase.query(comm);

                            while(result.next())
                            {
```

```
                                VertexSet.add(new
DVertex("V",result.getString(MetaDatabase.VALUE_VCODE),

        result.getString(MetaDatabase.VALUE_DATA)));
                                }

            //explore value userword

                        comm="select "+MetaDatabase.VALUE_DATA+", "+
                                MetaDatabase.VALUE_VCODE+" from
"+MetaDatabase.VALUE_TABLE+
                                " where upper("+MetaDatabase.VALUE_DATA+") in ("+
                                "select upper(word) from userwordvalue where
upper('"+key+"') like upper(userword))";;

                        result=MetaDatabase.query(comm);

                                while(result.next())
                                {

                                        VertexSet.add(new
DVertex("V",result.getString(MetaDatabase.VALUE_VCODE),

        result.getString(MetaDatabase.VALUE_DATA)));
                                }

            //explore in table attrib

                        comm="select distinct "+MetaDatabase.ATTRIB_ATTRIBNAME+",
"+
                                        MetaDatabase.ATTRIB_ATTRIBCODE+" from
"+MetaDatabase.ATTRIB_TABLE+
```

```
                    " where upper('"+key+"') like
upper("+MetaDatabase.ATTRIB_ATTRIBNAME+")";
                result=MetaDatabase.query(comm);
                    while(result.next())
                    {

                        VertexSet.add(new
DVertex("A",result.getString(MetaDatabase.ATTRIB_ATTRIBCODE),

    result.getString(MetaDatabase.ATTRIB_ATTRIBNAME)));
                    }

            //explore in attrib userword

                comm="select distinct "+MetaDatabase.ATTRIB_ATTRIBNAME+",
"+
                        MetaDatabase.ATTRIB_ATTRIBCODE+" from
"+MetaDatabase.ATTRIB_TABLE+
                    " where upper("+MetaDatabase.ATTRIB_ATTRIBNAME+")
in ("+
                    "select upper(word) from userwordattrib where
upper('"+key+"') like upper(userword))";

                result=MetaDatabase.query(comm);
                    while(result.next())
                    {

                        VertexSet.add(new
DVertex("A",result.getString(MetaDatabase.ATTRIB_ATTRIBCODE),

    result.getString(MetaDatabase.ATTRIB_ATTRIBNAME)));
                    }
```

```
                //explore in table entity
                        comm="select distinct "+MetaDatabase.ENT_ERNAME+
                                " from "+MetaDatabase.ENT_TABLE+
                                " where upper('"+key+"') like
upper("+MetaDatabase.ENT_ERNAME+")";

                        result=MetaDatabase.query(comm);
                                while(result.next())
                                {
                                        VertexSet.add(new
DVertex("E",result.getString(MetaDatabase.ENT_ERNAME)));
                                }

                //explore in  entity userword
                        comm="select distinct "+MetaDatabase.ENT_ERNAME+
                                " from "+MetaDatabase.ENT_TABLE+
                                " where upper("+MetaDatabase.ENT_ERNAME+") in ("+
                                "select upper(word) from userwordentity where
upper('"+key+"') like upper(userword))";
                        result=MetaDatabase.query(comm);
                                while(result.next())
                                {
                                        VertexSet.add(new
DVertex("E",result.getString(MetaDatabase.ENT_ERNAME)));
                                }
                //end Searching

                return VertexSet;
        }
}
```

QueryImage.java

```
class CQueryImage
{
	DKeywordVertex KV;
	void input(DKeywordVertex in)
	{
		KV=in;
	}

	public int numAllVertex()
	{
		int num;

		num=1;
		for(int i=0;i<KV.size();i++)
		num*=KV.getVertexSet(i).size();
		return num;
	}

	DQISet findQI()
	{
		DQISet QISet=new DQISet();
		DQI QI;
		DVertexSet VS;
		int nv,ne,ng,base,row,sz,sk;

		nv=numAllVertex();
		ne=nv; ng=1;
		sk=KV.size();

		//allocate blank QI
		for(int i=0;i<nv;i++)
```

```
            {
                QI=new DQI();
                QISet.add(QI);
            }

            for(int i=0;i<sk;i++)
            {
                VS=KV.getVertexSet(i);
                sz=VS.size(); // number of vertex in a keyword
                ng*=sz; // number of group
                ne/=sz; // number of vertex in each group (all group in same keyword
have same ne)

                for(int j=0;j<ng;j++)
                {
                    base=j*ne;
                    for(int k=0;k<ne;k++)
                    {
                        row=base+k;
                        QISet.get(row).add(VS.get(j%sz));
                    }
                }
            }
            return QISet;
        }
}
```

ShortestPath.java

```
import java.util.*;
import java.sql.*;
```

```
/*
 * NEW SYSTEM INTEGRATED WITH LEGACY SYSTEM
*/

class CSPEnt
{
        /* interface to determineErSpSet
         * requring Entity Set
         */
        COldSPEnt OldSPEnt=new COldSPEnt();
        private Statement stmt;
        DFG FG;

        CSPEnt(Statement st)
        {
                stmt=st;
        }

        private String FG2ErSet()
        {
                String ErSet=new String();

                Enumeration E;
                EntTree ET;
                boolean first;

                E=FG.Root.elements();
                first=true;

                while(E.hasMoreElements())
                {
                        ET=(EntTree)E.nextElement();
```

```
                ErSet=ErSet+ET.Ent.code+"|";
        }

        return ErSet;
}

public DQGSet getSP(DFG F)
{
        String ErSet;
        String[] ErSpSet;
        String[] ErPair;
        String[] Er;
        DQG QG;
        DQGSet QGS=new DQGSet();

        FG=F;
        OldSPEnt.setStmt(stmt);
        ErSet=FG2ErSet();
        ErSpSet=OldSPEnt.determineErSpSet(ErSet);

        for(int i=0;i<ErSpSet.length;i++)
                {
                        QG=process(ErSpSet[i]);
                        QGS.add(QG);
                }

        return QGS;
}

private EntTree getEntTree(String Ent)
{
        EntTree ET;
```

```
        if(FG.Root.containsKey(Ent))
        {
                ET=FG.getEntTree(Ent);
        }
        else
        {
                ET=FG.makeLeafEnt(new DVertex("E",Ent,Ent));
        }
        return ET;
}

DQG process(String allPath)
{
        DQG QG=new DQG();
        StringTokenizer StPipe=new StringTokenizer(allPath,"|");
        StringTokenizer StComma;
        boolean first=true;
        String Ent,path;
        DEntPath Path=null;

        while(StPipe.hasMoreTokens()) // each path
        {
                path=StPipe.nextToken();
                StComma=new StringTokenizer(path,",");
                first=true;
                Path=new DEntPath();
                while(StComma.hasMoreTokens()) //each ent in a path
                {
                        Ent=StComma.nextToken();

                        if(first)
```

```
                {
                        QG.setMain(getEntTree(Ent));
                        first=false;
                }
                else
                {
                        Path.add(getEntTree(Ent));
                }
            }
            QG.add(Path);
        }

        return QG;
    }
}

/*
 * LEGACY SYSTEM
*/

class StringVector {

    private Vector v = new Vector();

    // add new element to vector, the size will change automatically.
    public void add(String str) {
        v.addElement(new String(str));
    }

    //public void add(int index, String str) {
    //      v.InsertElementAt( new String(str), index);
    //}
```

```
    // This method return value of the ith element of the vector
    public String get(int n) {
            return ((String) v.elementAt(n)).toString();
    }

    public void set(int n, String str) {
            v.setElementAt(new String(str), n);
    }

    // return the current size of the vector
    public int size() {
            return v.size();
    }
    public boolean isEmpty() {
            if (v.size()==0) return true;
                    else return false;
    }

    public void removeAll() {
            v.removeAllElements();
    }

    public void remove(int n) {
            v.removeElementAt(n);
    }

    public void copyInto(String[] str) {
            v.copyInto(str);
    }
}
```

```
class Message {
        String node;
        String name;
        int cost;
        String from;
        public Message(String v, String n, int c, String f){
                node = v;
                name = n;
                cost = c;
                from = f;
        }
}


class COldSPEnt
{
        private StringVector root = new StringVector();
        private Vector current_cycle = new Vector();
        private Vector next_cycle = new Vector();
        private StringVector visited_nodes = new StringVector();
        private StringVector orig_visited_nodes = new StringVector();
        private Vector receivedMessage = new Vector();
        private StringVector list_of_nodes = new StringVector();
        private int best_cost;
        private Vector newp = new Vector();
        private Vector oldp = new Vector();
        private Statement stmt;

        public void setStmt(Statement st)
        {
                stmt=st;
```

```
}

private void determineSP(String erset){

String selectStr="";
//initialize all vectors
if(!(root.isEmpty())) root.removeAll();
if (!(current_cycle.isEmpty())) current_cycle.removeAllElements();
if (!(next_cycle.isEmpty())) next_cycle.removeAllElements();
if (!(visited_nodes.isEmpty())) visited_nodes.removeAll();
if (!(orig_visited_nodes.isEmpty())) orig_visited_nodes.removeAll();
if (!(receivedMessage.isEmpty())) receivedMessage.removeAllElements();
StringTokenizer st = new StringTokenizer(erset, "|");
int nb_nodes = st.countTokens();

if (nb_nodes == 1) {
        root.add(st.nextToken().trim());
        best_cost = 0;
        return;
}

String tempToken;
while (st.hasMoreTokens()){
        tempToken = st.nextToken().trim();
        visited_nodes.add(tempToken);
        orig_visited_nodes.add(tempToken);
        receivedMessage.addElement(new Message(tempToken, tempToken, 0, ""));
        current_cycle.addElement(new Message("", tempToken, 1, tempToken));}
int current_cost = 99;
int nb_cycle = 1;
while ((current_cost > nb_nodes-1) & (nb_cycle<=current_cost-nb_nodes+2)){
        //for each message in current_cycle
```

```
for (int i=0; i<current_cycle.size(); i++){
    Message current_message = (Message)current_cycle.elementAt(i);
    StringTokenizer stt = new
StringTokenizer(current_message.from.trim(), ",");
    String[] from_node = new String[stt.countTokens()];
    int findex = 0;
    while (stt.hasMoreTokens()){
        from_node[findex] = stt.nextToken().trim();
        findex++;
    }
    /*selectStr = "select distinct ername from mdb_belongto where ername
<> '"
        +from_node[0].trim()
        +"' and ername <> '" +current_message.name.trim()
        +"' and itemcode in (select itemcode from mdb_belongto where
"+
        " ername = '"+from_node[0].trim()+"' and inpkey='1'"
        +" union select eqitemcode from mdb_equivalent where
itemcode in"
        +" (select itemcode from mdb_belongto where "
        +" ername = '"+from_node[0].trim()+"' and inpkey='1') )"
        ;
    */

    if (!(list_of_nodes.isEmpty())) list_of_nodes.removeAll();

    String previous_node = "";
    if (from_node.length > 1)  previous_node = from_node[1].trim();

    //get_related_entrel(selectStr, previous_node);
```

```
                    /*selectStr ="select master from mdb_integrity where slave =
'"+from_node[0].trim()+"'"
                                +" and master <> '" + current_message.name.trim()+
"'";*/
                    selectStr ="select master from integrity where slave =
'"+from_node[0].trim()+"'"
                                +" and master <> '" + current_message.name.trim()+
"'";
                    get_related_entrel(selectStr, previous_node);
                    /*selectStr ="select slave from mdb_integrity where master =
'"+from_node[0].trim()+"'"
                                +" and slave <> '" + current_message.name.trim()+
"'";*/
                    selectStr ="select slave from integrity where master =
'"+from_node[0].trim()+"'"
                                +" and slave <> '" + current_message.name.trim()+ "'";
                    get_related_entrel(selectStr, previous_node);

                    //for each node in list_of_nodes
                    for (int j=0; j<list_of_nodes.size(); j++){

                        if (!(searchVector(list_of_nodes.get(j), visited_nodes))){
                            //add list_of_nodes.get(j) to visited_nodes
                            visited_nodes.add(list_of_nodes.get(j));
                        }

                        //if (!(searchVector(list_of_nodes.get(j), orig_visited_nodes))){
                            int inc_cost = current_message.cost;
                            inc_cost++;
                            next_cycle.addElement(new Message("",
current_message.name, inc_cost, list_of_nodes.get(j).trim()+","+current_message.from.trim()) );
                        //      }
```

```
//find message i in node j
boolean found_message = false;
int t = 0;
while ( !(found_message) & (t< receivedMessage.size())) {
    Message work_message =
(Message)receivedMessage.elementAt(t);
    if
((list_of_nodes.get(j).trim().equalsIgnoreCase(work_message.node.trim())) &
(current_message.name.trim().equalsIgnoreCase(work_message.name.trim())) )
    { found_message = true;} else {t++;}
}
if (!(found_message)) {
    receivedMessage.addElement(new
Message(list_of_nodes.get(j).trim(),
current_message.name,current_message.cost,current_message.from));
    //Remark: Used for the next_cycle that results from
splitting from from->node in list_of_nodes
    //and (both the node and from are in original visited
node)
    if (from_node.length > 1){
        for (int ct = 0; ct < receivedMessage.size();
ct++){
            Message ct_message =
(Message)receivedMessage.elementAt(ct);
            for (int lctl= from_node.length-1; lctl
> 0; lctl--){
```

```
                                String wstr = "";
                                String laststr = "";
                                for (int ml=0; ml < lctl;
ml++){
                                    if ((lctl==1) |
(ml==lctl-1) ) {wstr = wstr+from_node[ml].trim();
                                        laststr =
from_node[ml].trim();}
                                    else {wstr =
wstr+from_node[ml].trim()+",";}
                                }
                                if (
(list_of_nodes.get(j).trim().equalsIgnoreCase(ct_message.node.trim())) &
        (ct_message.name.trim().equalsIgnoreCase(laststr.trim())) &
        (wstr.trim().equalsIgnoreCase(ct_message.from.trim()))
                                ){
        receivedMessage.setElementAt(new Message(ct_message.node, ct_message.name, 0
,ct_message.from),ct);
                                }
                            }//for lctl
                        } //for ct

                    } //if (from_node.length > 1)

                    int count_message=0;
                    for (int c=0; c < receivedMessage.size();c++){
```

```
                                        Message c_message =
(Message)receivedMessage.elementAt(c);
                                        if
(list_of_nodes.get(j).trim().equalsIgnoreCase(c_message.node.trim()))
                                                count_message++;}
                                    if ((count_message == nb_nodes) &

        (computed_cost(list_of_nodes.get(j).trim())<=current_cost )){
                                        if (!(root.isEmpty()) &

        (computed_cost(list_of_nodes.get(j).trim())<current_cost)){
                                                root.removeAll();}
                                        //Check here before insert new root
                                        if (root.isEmpty())
{root.add(list_of_nodes.get(j));}
                                        else {
                                                enumeratePaths(list_of_nodes.get(j),
newp);
                                                boolean found_root = false;
                                                int r =0;
                                                while ( r<root.size() & !(found_root)
){
                                                        enumeratePaths(root.get(r),
oldp);
                                                determineEQpath(newp,
oldp);
                                                if ( equalPath(newp, oldp) )
                                                        found_root = true;
                                                r++;
                                        }//while r
                                        if (!(found_root))
                                                root.add(list_of_nodes.get(j));
```

```
                                        }//if (root.isEmpty())

                                        current_cost =
computed_cost(list_of_nodes.get(j).trim());}
                                }//if (!(found_message))

                                //Pass message through two orig-adjacent nodes For the First
Cycle
                                if (nb_cycle == 1){
                                 if ( searchVector(list_of_nodes.get(j),orig_visited_nodes) &
                                    searchVector(from_node[0],orig_visited_nodes) ) {
                                        String t_message_name = from_node[0].trim();
                                        int t_cost = 2;
                                        String t_message_from =
list_of_nodes.get(j).trim()+"," + from_node[0].trim();
                                        next_cycle.addElement(new Message("",
t_message_name, t_cost, t_message_from) );
                                }}
                        }//for j

                }//for i
                current_cycle.removeAllElements();
                for (int z=0; z<next_cycle.size(); z++){
                        Message m = (Message)next_cycle.elementAt(z);
                        current_cycle.addElement(m);
                }
                next_cycle.removeAllElements();
                nb_cycle++;
        }//while
        best_cost = current_cost;
}
```

```
private String[] ErSpSet; //connected entrels
private String[][] ErSpElement;
private ResultSet rset;

private Vector common_path = new Vector();


public String[] determineErSpSet(String ErSet){

        determineSP(ErSet);

        ErSpSet = new String[root.size()];
        ErSpElement = new String[root.size()][];


        for (int k=0; k <root.size(); k++){
         //determine paths from root vector
         if (receivedMessage.size()==0) {
                ErSpSet[k] = root.get(0);
         } else {
                String path="";
                for (int s =0; s<receivedMessage.size(); s++){
                        Message toRoot = (Message)receivedMessage.elementAt(s);
                        if ((toRoot.node.trim().equalsIgnoreCase(root.get(k).trim())) &
(toRoot.cost != 0)) {
                                path = path+root.get(k).trim()+","+toRoot.from+"|";
                        }
                }//end for (int s =0

                //add path to pathstr
                ErSpSet[k] = path;
```

```
            }//if (receivedMessage.size()==0)
        }// end for int k

        return ErSpSet;
}

        private void get_related_entrel(String selectStr, String previous_node){
        try {
                rset = stmt.executeQuery(selectStr);
                while (rset.next()){
                        String ername = rset.getString(1);
                        if(!(searchVector(ername.trim(),list_of_nodes))){
                                if ( !(previous_node.trim().equalsIgnoreCase(ername.trim())) )
                                        list_of_nodes.add(ername.trim());
                        }
                }
        } catch (Exception e){System.out.println(e);}
}

private boolean searchVector(String searchStr, StringVector Vector) {
        boolean found = false;
        int i=0;
        while (!(found) & (i < Vector.size())) {
                if (searchStr.trim().equalsIgnoreCase(Vector.get(i).trim())) {
                        found = true;}
                i++;}
        return found;
}// end searchTermVector

private int computed_cost(String node){
        int cost = 0;
        for (int i=0; i < receivedMessage.size(); i++) {
```

```
                Message costMessage = (Message)receivedMessage.elementAt(i);
                if (node.trim().equalsIgnoreCase(costMessage.node.trim()))
                        cost = cost + costMessage.cost;


        }
        return cost;
}

private void enumeratePaths(String root, Vector pathVector) {

        StringTokenizer pfrom;
        if (!(pathVector.isEmpty())) {pathVector.removeAllElements();}
        for (int np=0; np < receivedMessage.size(); np++){
                Message np_message = (Message)receivedMessage.elementAt(np);
                String[] nodelist;
                pfrom = new StringTokenizer(np_message.from,",");
                nodelist = new String[pfrom.countTokens()+1];
                if ( (np_message.cost > 0) &
                  (root.trim().equalsIgnoreCase(np_message.node.trim())) ) {
                        nodelist[0] = np_message.node;
                        int nn=1;
                        while (pfrom.hasMoreTokens()){
                                nodelist[nn] = pfrom.nextToken();
                                nn++;
                        }//while (pfrom
                    for (int i=0; i< nodelist.length-1; i++){
                                pathVector.addElement(new Path(nodelist[i], nodelist[i+1]));
                        }
                }//if (np_message.cost > 0)
        }//for (int np
}
```

```
private Vector p2_from = new Vector();
private Vector p1_to = new Vector();
private Vector eq_path = new Vector(); //from EQ to

private void determineEQpath(Vector path1, Vector path2){

        common_path.removeAllElements();
        p1_to.removeAllElements();
        p2_from.removeAllElements();
        eq_path.removeAllElements();


        if (path1.size() == path2.size()){
                //determine common_path
                for (int i=0; i < path1.size(); i++){
                        Path p1 = (Path)path1.elementAt(i);
                        for (int j=0; j< path2.size(); j++) {
                                Path p2 = (Path)path2.elementAt(j);
                                if (p1.from.trim().equalsIgnoreCase(p2.to.trim()) &
                                        p1.to.trim().equalsIgnoreCase(p2.from.trim()) ){
                                        common_path.addElement(new
Path(p1.from.trim(),p2.to.trim()));
                                }
                        }// end j
                }// end i

                //determine p1_to
                for (int i=0; i < common_path.size(); i++){
                        Path cp = (Path)common_path.elementAt(i);
                        for (int j=0; j < path1.size(); j++){
                                Path p1 = (Path)path1.elementAt(j);
                                if (cp.from.trim().equalsIgnoreCase(p1.from.trim())){
```

```
                                p1_to.addElement(new
Path(p1.from.trim(),p1.to.trim()));
                        }
                    }
                }

                //determine p2_from
                for (int i=0; i < common_path.size(); i++){
                    Path cp = (Path)common_path.elementAt(i);

                    int k=0;
                    boolean found = false;
                    String str_from ="";
                    while (!(found) & k<path2.size()){
                        Path tp2 = (Path)path2.elementAt(k);
                        if ( cp.from.trim().equalsIgnoreCase(tp2.to.trim()) ) {
                            str_from += tp2.from;
                            found = true;} else {k++;}
                    }

                    for (int j=0; j < path2.size(); j++){
                        Path p2 = (Path)path2.elementAt(j);
                        if (str_from.trim().equalsIgnoreCase(p2.from.trim())){
                            p2_from.addElement(new Path(p2.from.trim(),
p2.to.trim()));
                        }
                    }
                }

                //determine eq_path
                for (int i=0; i < p1_to.size(); i++){
                    Path p1 = (Path)p1_to.elementAt(i);
```

```
                    for (int j=0; j < p2_from.size(); j++){
                        Path p2 = (Path)p2_from.elementAt(j);
                        if (p1.to.trim().equalsIgnoreCase(p2.to.trim())){
                            eq_path.addElement( new Path(
p1.from.trim()+p1.to.trim(), p2.from.trim()+p2.to.trim() ));
                        }
                    }
                }

        }//end path1.size()

}

private boolean equalPath(Vector path1, Vector path2) {

        boolean found = false;
        if ( (path1.size() == path2.size()) ) {
        int i=0;
        boolean all_match = true;
        while ( i < path1.size() & (all_match) ){
            Path p1 = (Path)path1.elementAt(i);
            String str1 = p1.from.trim()+p1.to.trim();
            String str2 = p1.to.trim()+p1.from.trim();

            boolean matched = false;
            int j=0;
            while ( j < path2.size() & !(matched)){
                Path p2 = (Path)path2.elementAt(j);
                String str3 = p2.from.trim()+p2.to.trim();

                boolean is_eq = false;
                int k=0;
```

```
                while ( k<eq_path.size() & !(is_eq) ){
                        Path p_eq = (Path)eq_path.elementAt(k);
                        if (str1.trim().equalsIgnoreCase(p_eq.from.trim()) &
                        str3.trim().equalsIgnoreCase(p_eq.to.trim())   ) {
                                is_eq =true;
                        } else {k++;}
                }
                if ( (is_eq) | str1.trim().equalsIgnoreCase(str3.trim()) |
                str2.trim().equalsIgnoreCase(str3.trim()) ) {
                        matched = true;
                } else {j++;}
            }//while j
            if (!(matched)) {all_match = false;} else {i++;}
        }//while i

        if (all_match) found = true;

 }//if (path1.size() = path2.size())

 return found;
}

class Path {
        String from;
        String to;
        public Path(String f, String t){
                from = f;
                to = t;
        }
}
}
```

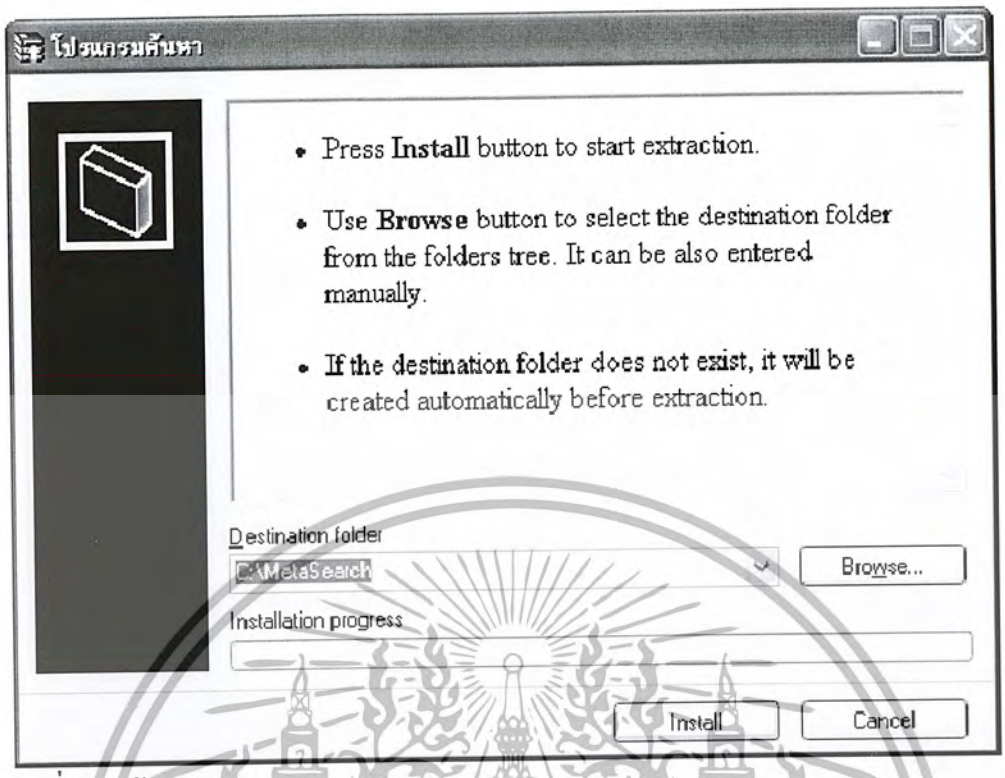
Search.java

```
import java.io.IOException;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class Search extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException
   {
      ServletOutputStream out = response.getOutputStream();
          CFindQG FindQG=new CFindQG();
          String txt,source;

          Date D=new Date();

          System.out.println("-----------------------------------");
          System.out.println("Access : "+ D.toString());
       txt=request.getParameter("keyword");

          source=FindQG.getSource(txt);
          out.println(source);
   }
}
```

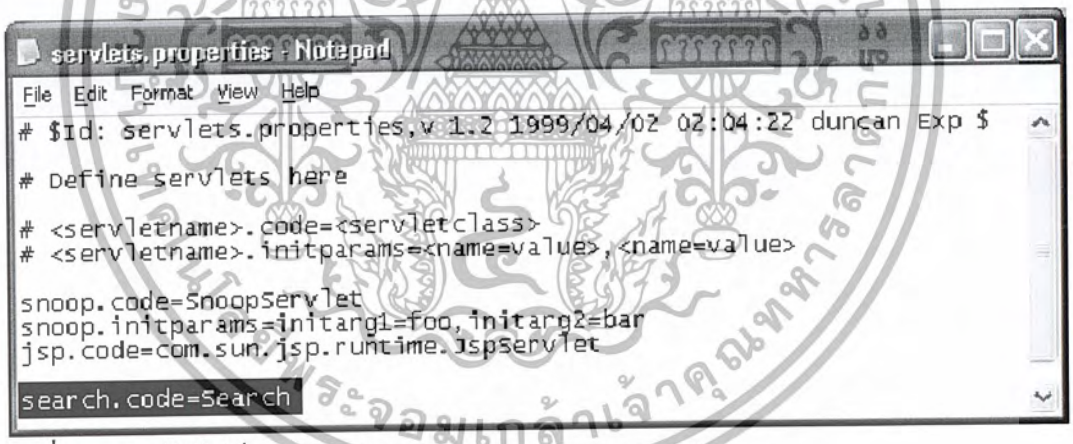
ภาคผนวก ง
การติดตั้งและปรับแต่งค่าของระบบ

วิธีการติดตั้งและปรับแต่งค่าของระบบมีดังต่อไปนี้ คือ

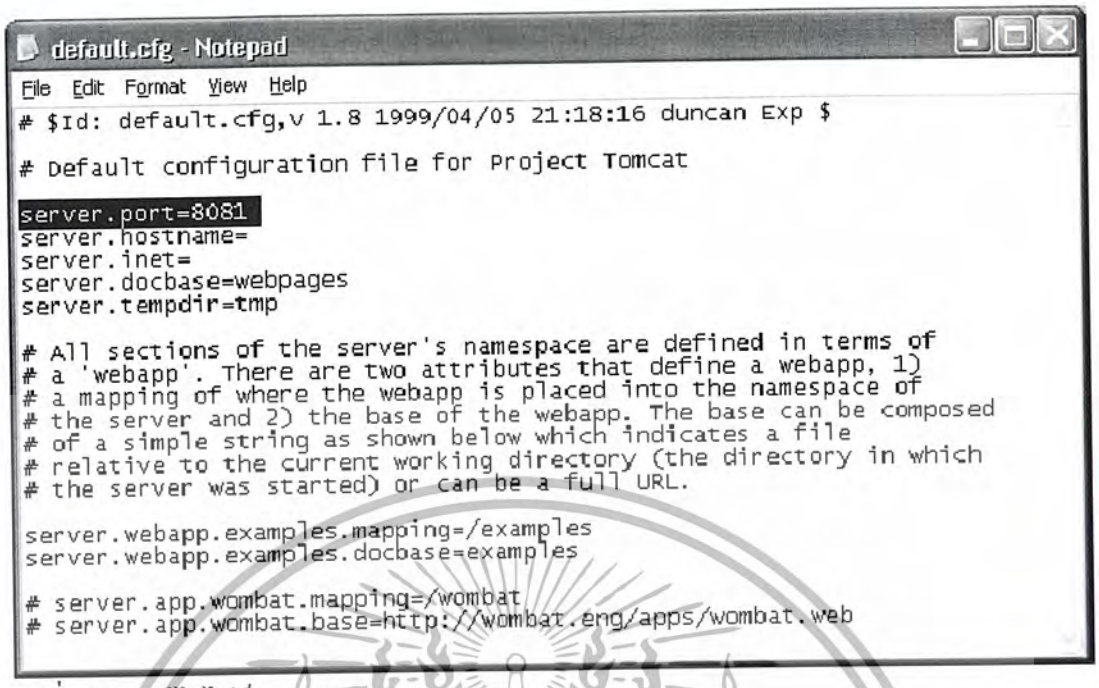
1. ทำการติดตั้งโปรแกรม Java 2 SDK, Standard Edition และระบบฐานข้อมูล Oracle
2. ทำการติดตั้ง Java Servlet Development Kit โดยทำการเปิดไฟล์ชื่อ jsdk2_1-win.exe และใส่ชื่อรากที่ต้องการ เช่น d:\ ดังรูปที่ 1
3. ทำการติดตั้งโปรแกรมค้นหา โดยทำการเปิดไฟล์ชื่อ MetaSearch.exe และใส่ชื่อรากที่ต้องการ เช่น d:\MetaSearch
4. ทำการติดตั้งโปรแกรมเซิร์ฟเลต โดยทำการคัดลอกไฟล์ชื่อ Search.class ลงไปในรากที่เก็บโปรแกรมด้านเซิร์ฟเลตของ Java Servlet Development Kit ที่ได้ติดตั้งแล้ว คือ D:\jsdk2.1\webpages\WEB-INF\servlets
5. ทำการปรับแต่งค่าของเซิร์ฟเลต โดยทำการแก้ไขไฟล์ชื่อ servlets.properties ในรากชื่อ D:\jsdk2.1\webpages\WEB-INF โดยเพิ่มข้อความ search.code=Search ลงไปในไฟล์ดังรูปที่ 2
6. เปลี่ยนพอร์ตของตัวเซิร์ฟเลต โดยทำการแก้ไขไฟล์ชื่อ default.cfg ในราก D:\jsdk2.1โดยเปลี่ยนหมายเลขตรงข้อความพอร์ตเดิม คือ 8080 ใน server.port=8080 เป็นหมายเลขพอร์ตใหม่คือ server.port=8081 ดังรูปที่ 3
7. ทำการปรับแต่งค่าเพื่อให้เซิร์ฟเลตรู้จักโปรแกรมค้นหา โดยทำการแก้ไขไฟล์ startserver.bat ในรากชื่อ D:\jsdk2.1\webpages\WEB-INF โดยทำการเพิ่มชื่อรากของโปรแกรมค้นหา โดยเพิ่มข้อความ เช่น D:\MetaSearch;D\MetaSearch\classes12.zip;D:\MetaSearch\jasa.jar ลงไปในบรรทัด set CLASSPATH ดังรูปที่ 4



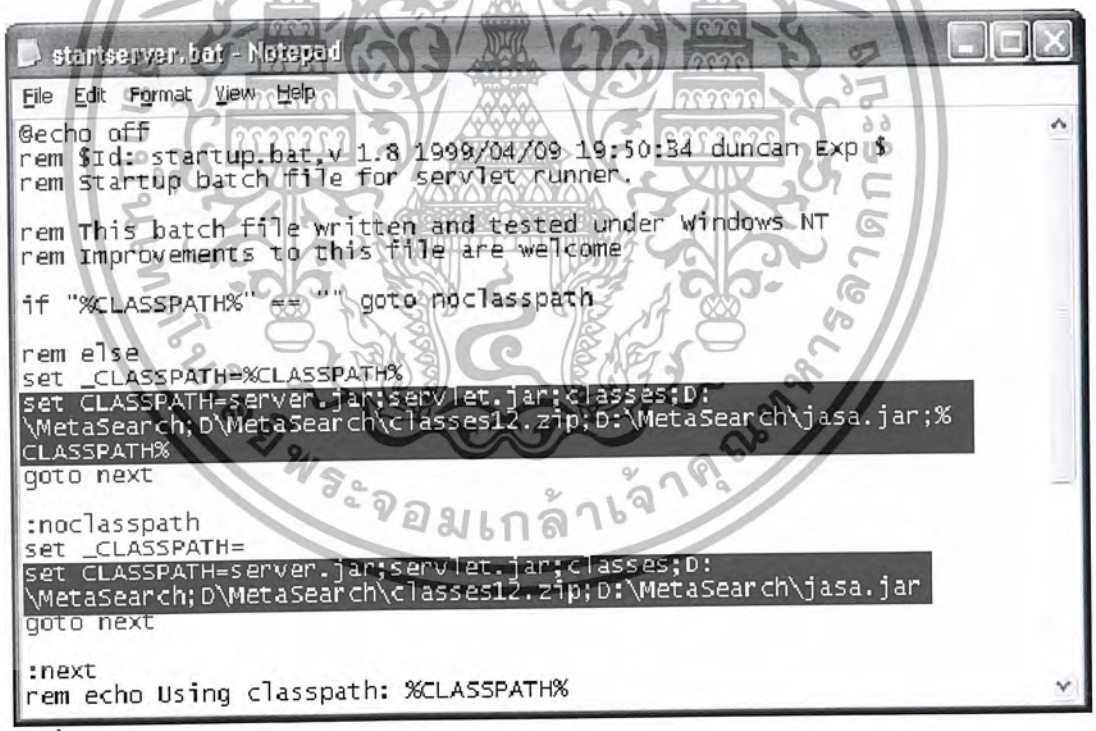
รูปที่ ผ1 หน้าจอของ jsdk2_1-win.exe



รูปที่ ผ2 การแก้ไขไฟล์ servlets.properties



รูปที่ ผ3 การแก้ไขไฟล์ default.cfg



รูปที่ ผ4 การแก้ไขไฟล์ startserver.bat