บทที่ 3

การออกแบบและพัฒนาระบบ

จากการศึกษาและทำความเข้าใจเกี่ยวกับระบบฐานข้อมูล และระบบจัดการฐานข้อมูลเชิงสัมพันธ์ ซึ่งต้องมีการจัดเก็บข้อมูลในระบบจำนวนมาก งานวิทยานิพนธ์นี้เกี่ยวข้องกับส่วนสำหรับการติดต่อใช้งานฐานข้อมูลจากผู้ใช้ทั่วไป เพื่อกำหนดโครงสร้างฐานข้อมูลและชนิดข้อมูลที่จัดเก็บในฐานข้อมูล และการดำเนินงานพื้นฐานในการรับข้อมูลเข้า การลบข้อมูลออก หรือการปรับปรุงเปลี่ยนแปลงแก้ไขข้อมูลในฐานข้อมูล ในการพัฒนาระบบจึงได้วิเคราะห์และออกแบบระบบโดยคำนึงถึงการจัดเก็บข้อมูลต่างๆ ในระบบ การจัดการบัฟเฟอร์ (buffer) บนหน่วยความจำหลักเพื่อเป็นที่พักข้อมูลในระหว่างการประมวลผล และการติดต่อระหว่างผู้ใช้กับระบบเพื่อให้ผู้ใช้สามารถใช้งานฐานข้อมูลได้

3.1 การจัดการแฟ้มข้อมูล

ข้อมูลทั้งหมดในระบบฐานข้อมูลทั้งปทานุกรมข้อมูลและข้อมูลของฐานข้อมูลที่ผู้ใช้กำหนดขึ้นจะถูกจัดเก็บเป็นรีเลชันไว้ในแฟ้มข้อมูลที่ถูกสร้างขึ้นในระบบ เรียกว่า DBFILE โดยแต่ละแฟ้มข้อมูลจะใช้เนื้อที่บนหน่วยความจำสำรองหรือดิสก์ (disk) ขนาดคงที่จำนวนหนึ่งซึ่งแบ่งออกเป็นส่วน ๆ ที่มีขนาดเท่ากัน เรียกว่าบล็อก (block) การจัดสรรเนื้อที่บนดิสก์และการโอนย้ายข้อมูลระหว่างดิสก์กับหน่วยความจำหลักจะกระทำในระดับบล็อก

1. ประเภทของข้อมูลในแฟ้มข้อมูล

แฟ้มข้อมูลที่จัดเก็บในระบบเมื่อแบ่งตามประเภทของข้อมูลที่จัดเก็บในแฟ้มข้อมูล จะแบ่งได้สองประเภท ได้แก่ แฟ้มข้อมูลปทานุกรม และแฟ้มข้อมูลของฐานข้อมูล ภาพประกอบที่ 3.1 แสดงแฟ้มข้อมูลแต่ละประเภทในระบบ

แฟ้มข้อมูลปทานุกรม

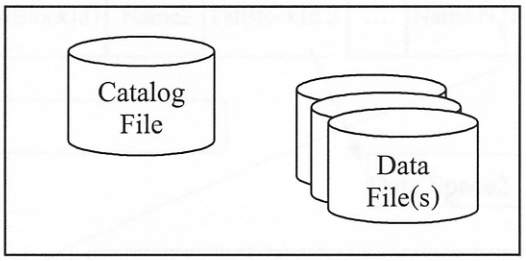
แฟ้มข้อมูลปทานุกรมเป็นแฟ้มข้อมูลสำหรับจัดเก็บรายละเอียดต่างๆ เกี่ยวกับปทานุกรมข้อมูล เรียกว่า catalog file ในระบบจะมีแฟ้มข้อมูลปทานุกรมเพียงแฟ้มข้อมูลเดียวเท่านั้น ข้อมูลที่จัดเก็บในแฟ้มข้อมูลปทานุกรมประกอบด้วยข้อมูลต่อไปนี้

- นิยามฐานข้อมูลเป็นรายละเอียดของแต่ละฐานข้อมูลทั้งหมดในระบบ
- นิยามรีเลชันเป็นรายละเอียดของรีเลชันทั้งหมดในระบบ
- นิยามแอตทริบิวเป็นรายละเอียดของแอตทริบิวทั้งหมดในระบบ

- เงื่อนไขบังคับ (constraint) เป็นรายละเอียดของเงื่อนไขบังคับทั้งหมดในระบบที่กำหนดให้กับแอตทริบิวของรีเลชัน ได้แก่ การกำหนดคีย์หลัก คีย์รอง และคีย์นอก (foreign key) ซึ่งการกำหนดคีย์นอกจะเป็นการกำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว (referential integrity) ของรีเลชัน โดยเป็นการอ้างอิงค่าแอตทริบิวที่เป็นคีย์นอกจากค่าแอตทริบิวซึ่งเป็นคีย์หลักในอีกรีเลชันหนึ่งในฐานข้อมูลเดียวกัน
- ข้อมูลดัชนีสำหรับเก็บค่าคีย์ที่ผู้ใช้กำหนดขึ้นในแต่ละรีเลชัน

แฟ้มข้อมูลของฐานข้อมูล

แฟ้มข้อมูลของฐานข้อมูลเป็นแฟ้มข้อมูลประเภทที่ใช้สำหรับจัดเก็บข้อมูลทั้งหมดในฐานข้อมูลหนึ่งๆ เรียกว่า data file ในระบบมีแฟ้มข้อมูลฐานข้อมูลได้มากกว่าหนึ่งแฟ้มข้อมูล ขึ้นอยู่กับจำนวนฐานข้อมูลที่ผู้ใช้สร้างขึ้นในระบบ โดยทุกรีเลชันของฐานข้อมูลเดียวกันจะถูกจัดเก็บไว้ในแฟ้มข้อมูลเดียวกัน



ภาพประกอบที่ 3.1 แฟ้มข้อมูลในระบบ

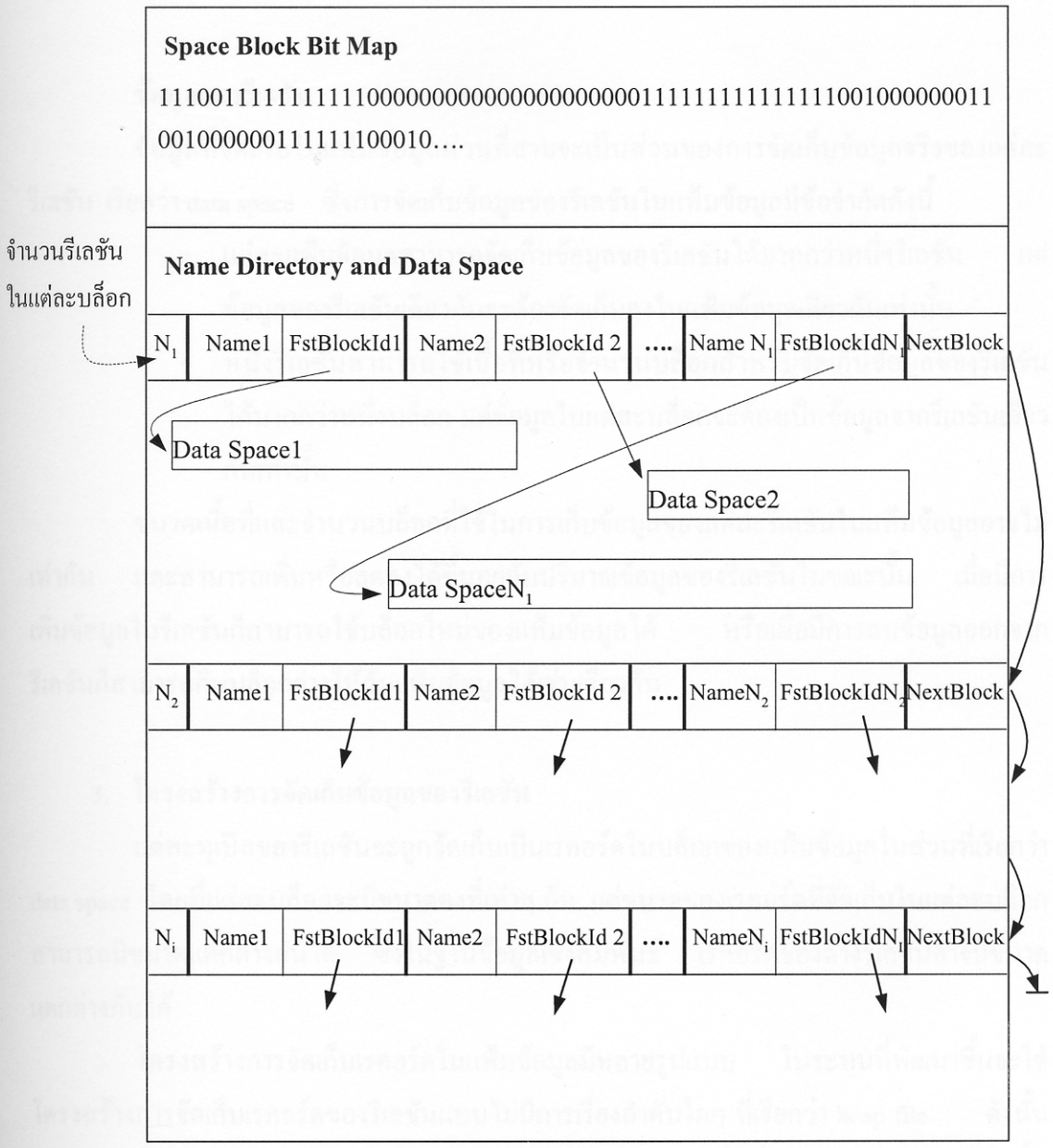
2. โครงสร้างการจัดเก็บข้อมูลของแฟ้มข้อมูลในระบบ

แต่ละแฟ้มข้อมูลของระบบถูกออกแบบให้สามารถจัดเก็บข้อมูลของรีเลชันได้มากกว่าหนึ่งรีเลชัน โดยที่ข้อมูลของรีเลชันเดียวกันจะต้องถูกจัดเก็บในแฟ้มข้อมูลเดียวกัน ดังนั้นข้อมูลที่ถูกจัดเก็บในแฟ้มข้อมูลนอกจากจะเป็นข้อมูลจริงของรีเลชันต่างๆ แล้ว จะจัดเก็บข้อมูลส่วนอื่นด้วยเพื่อให้สามารถเข้าถึงข้อมูลในแต่ละรีเลชันได้และเพิ่มประสิทธิภาพในการใช้งานแฟ้มข้อมูลที่สร้างขึ้น ภาพประกอบที่ 3.2 แสดงโครงสร้างการจัดเก็บข้อมูลของแฟ้มข้อมูลในระบบ ซึ่งข้อมูลที่จัดเก็บในแฟ้มข้อมูลของระบบแบ่งออกเป็นสามส่วน ได้แก่ สถานะของบล็อก รายชื่อรีเลชัน และข้อมูลของรีเลชันต่างๆ ที่ถูกจัดเก็บในแฟ้มข้อมูล

สถานะของบล็อก

ข้อมูลที่จัดเก็บในแฟ้มข้อมูลส่วนแรกจะเป็นส่วนที่บอกสถานะของบล็อกทั้งหมดในแฟ้มข้อมูล เรียกว่า space block bit map จะจัดเก็บลำดับบิตเพื่อระบุสถานะของบล็อกทั้งหมดใน

แฟ้มข้อมูล โดยหนึ่งบิตจะระบุสถานะของบล็อกข้อมูลหนึ่งบล็อกที่สัมพันธ์กันกับตำแหน่งบิตนั้น นั่นคือบิตตำแหน่งที่ 1 จะระบุสถานะของบล็อกที่ 1 ในแฟ้มข้อมูล และบิตตำแหน่งถัดไปก็ระบุสถานะของบล็อกถัดไปตามลำดับ ถ้าค่าบิตเป็น 1 แสดงว่าสถานะของบล็อกไม่ว่างมีข้อมูลจัดเก็บอยู่ แต่ถ้าค่าบิตเป็น 0 แสดงว่าสถานะของบล็อกว่าง ดังนั้นถ้าบล็อกใดถูกเลือกใช้เพื่อจัดเก็บข้อมูลก็จะกำหนดค่าบิตเป็น 1 และถ้ามีการคืนบล็อกให้แฟ้มข้อมูลจะกำหนดค่าบิตเป็น 0 ขนาดเนื้อที่สำหรับจัดเก็บข้อมูลส่วนนี้จะขึ้นอยู่กับจำนวนบล็อกทั้งหมดในแฟ้มข้อมูลที่สร้างขึ้น



ภาพประกอบที่ 3.2 โครงสร้างการจัดเก็บข้อมูลของแฟ้มข้อมูลในระบบ

รายชื่อรีเลชัน

ข้อมูลที่จัดเก็บในแฟ้มข้อมูลส่วนที่สองจะเป็นส่วนของการจัดเก็บรายชื่อรีเลชันทั้งหมดในแฟ้มข้อมูล เรียกว่า name directory และหมายเลขบล็อกเพื่อเป็นตำแหน่งเริ่มต้นในการเข้าถึงข้อมูลของแต่ละรีเลชันในแฟ้มข้อมูล โดยที่ชื่อรีเลชันทั้งหมดที่จัดเก็บในแฟ้มข้อมูลจะต้องไม่ซ้ำกัน ขนาดเนื้อที่และจำนวนบล็อกบนแฟ้มข้อมูลในส่วนนี้ขึ้นอยู่กับจำนวนรีเลชันที่เก็บในแฟ้มข้อมูล โครงสร้างข้อมูลที่ใช้จัดเก็บข้อมูลส่วนนี้จะเป็นโครงสร้างข้อมูลแบบลิงค์ลิสต์ ดังแสดงในภาพประกอบที่ 3.2

ข้อมูลของรีเลชัน

ข้อมูลที่จัดเก็บในแฟ้มข้อมูลส่วนที่สามจะเป็นส่วนของการจัดเก็บข้อมูลจริงของแต่ละรีเลชัน เรียกว่า data space ซึ่งการจัดเก็บข้อมูลของรีเลชันในแฟ้มข้อมูลมีข้อจำกัดดังนี้

- แต่ละแฟ้มข้อมูลสามารถจัดเก็บข้อมูลของรีเลชันได้มากกว่าหนึ่งรีเลชัน แต่ข้อมูลของรีเลชันเดียวกันจะต้องจัดเก็บลงในแฟ้มข้อมูลเดียวกันเท่านั้น
- หนึ่งรีเลชันสามารถใช้เนื้อที่หรือจำนวนบล็อกสำหรับจัดเก็บข้อมูลของรีเลชันได้มากกว่าหนึ่งบล็อก แต่ข้อมูลในแต่ละบล็อกจะต้องเป็นข้อมูลจากรีเลชันเดียวกันเท่านั้น

ขนาดเนื้อที่และจำนวนบล็อกที่ใช้ในการเก็บข้อมูลของแต่ละรีเลชันในแฟ้มข้อมูลอาจไม่เท่ากัน และสามารถเพิ่มหรือลดลงได้ขึ้นอยู่กับปริมาณข้อมูลของรีเลชันในขณะนั้น เมื่อมีการเพิ่มข้อมูลในรีเลชันก็สามารถใช้บล็อกใหม่ของแฟ้มข้อมูลได้ หรือเมื่อมีการลบข้อมูลออกจากรีเลชันก็สามารถคืนบล็อกว่างให้กับแฟ้มข้อมูลได้เช่นเดียวกัน

3. โครงสร้างการจัดเก็บข้อมูลของรีเลชัน

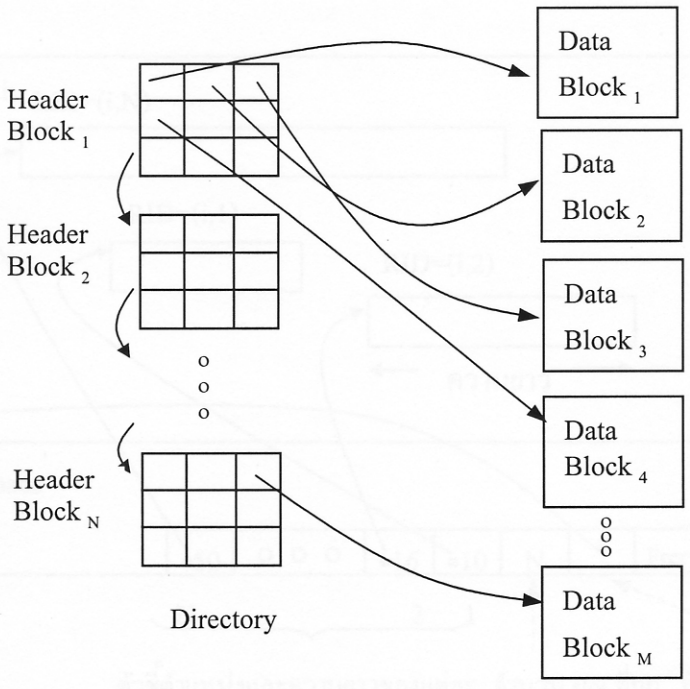
แต่ละทูเปิลของรีเลชันจะถูกจัดเก็บเป็นเรคอร์ดในบล็อกของแฟ้มข้อมูลในส่วนที่เรียกว่า data space โดยที่แต่ละบล็อกจะมีขนาดคงที่เท่าๆ กัน แต่ขนาดของเรคอร์ดที่จัดเก็บในแต่ละบล็อกสามารถมีขนาดแตกต่างกันได้ ซึ่งในฐานข้อมูลเชิงสัมพันธ์ เรคอร์ดของต่างรีเลชันอาจมีขนาดแตกต่างกันได้

โครงสร้างการจัดเก็บเรคอร์ดในแฟ้มข้อมูลมีหลายรูปแบบ ในระบบที่พัฒนาขึ้นจะใช้โครงสร้างการจัดเก็บเรคอร์ดของรีเลชันแบบไม่มีการเรียงลำดับใดๆ ที่เรียกว่า heap file ดังนั้นเรคอร์ดต่างๆจะถูกจัดเก็บไว้ในตำแหน่งใดก็ได้ของแฟ้มข้อมูลที่มีที่ว่างพอสำหรับเรคอร์ดนั้นโดยไม่คำนึงลำดับของการจัดเก็บเรคอร์ดในรีเลชัน ภาพประกอบที่ 3.3 แสดงตัวอย่างโครงสร้าง

การจัดเก็บเรคอรด์ในรีเลชันหนึ่ง โดยบล็อกของข้อมูลต่างๆ ที่จัดเก็บในรีเลชัน แบ่งเป็นสองประเภทดังนี้

- **Data block** เป็นบล็อกข้อมูลสำหรับจัดเก็บข้อมูลจริงของรีเลชัน
- **Header block** เป็นบล็อกข้อมูลสำหรับจัดเก็บตำแหน่งของ data block ที่จัดเก็บข้อมูลของรีเลชัน และขนาดเนื้อที่ว่างของ data block นั้น

โครงสร้างการจัดเก็บเรคอร์ดแบบนี้จะมีเฉพาะส่วนของ header block เท่านั้นที่จะเชื่อมโยงกันเป็นลิงค์ลิสต์แบบทิศทางเดียว เพื่อให้ง่ายต่อการจัดการพื้นที่ว่างบน data block ของรีเลชัน แต่ในกรณีที่มีการเพิ่มข้อมูลของรีเลชันและไม่มีเนื้อที่ว่างใน data block เก่าของรีเลชันจะต้องขอใช้บล็อกข้อมูลใหม่ในแฟ้มข้อมูล และกรณีที่การลบข้อมูลเก่าออกจากรีเลชันซึ่งทำให้เกิดบล็อกว่างจะต้องคืนบล็อกว่างให้กับระบบ ทั้งสองกรณีจะต้องมีการเปลี่ยนแปลงแก้ไขข้อมูลในส่วนของ header block ด้วย



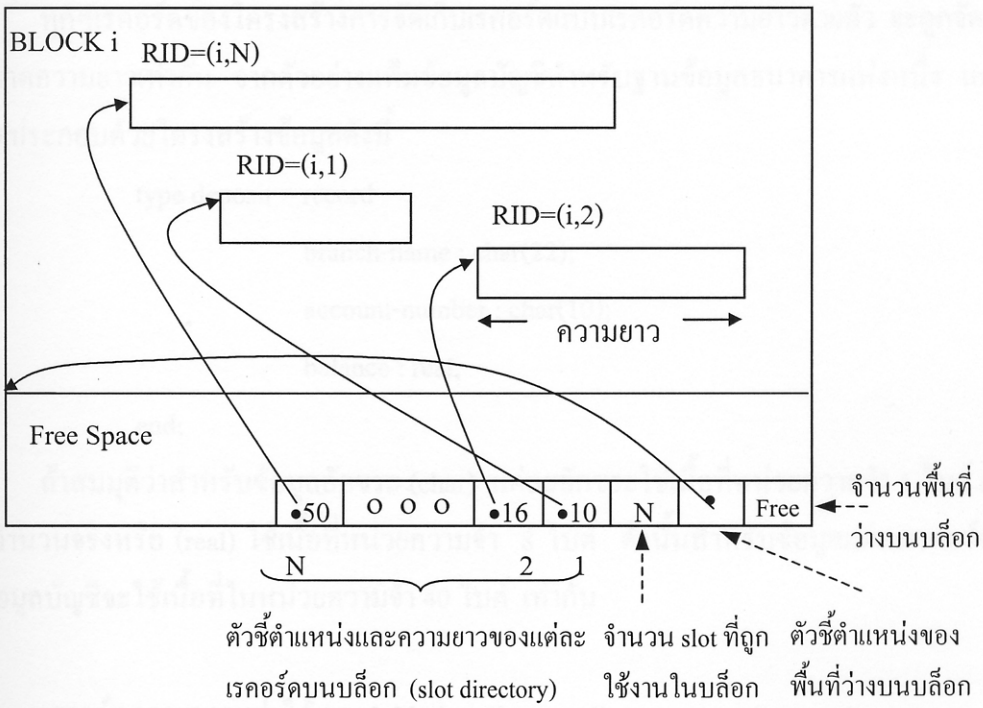
ภาพประกอบที่ 3.3 โครงสร้างการจัดเก็บข้อมูลของรีเลชัน

4. โครงสร้างของบล็อกข้อมูล (data block structure)

แฟ้มข้อมูลในระบบแบ่งออกเป็นบล็อกขนาดเท่าๆ กัน แต่ละบล็อกข้อมูลสามารถจัดเก็บเรคอร์ดของรีเลชันไว้ได้มากกว่าหนึ่งเรคอร์ด แต่ละบล็อกบนแฟ้มข้อมูลจะเก็บเรคอร์ดพร้อมทั้งรายละเอียดที่เกี่ยวข้องเพื่อความสะดวกในการจัดเก็บและเข้าถึงเรคอร์ดในบล็อก ได้แก่ ตำแหน่ง

ของพื้นที่ว่างบนบล็อก กลุ่มข้อมูลที่บอกตำแหน่งและความยาวของแต่ละเรคอร์ดในบล็อก เรียกว่า slot directory และจำนวน slot ที่ถูกใช้งานในบล็อก โดยที่แต่ละ slot จะบอกตำแหน่งและความยาวของเรคอร์ดหนึ่งๆ ในบล็อก ภาพประกอบที่ 3.4 แสดงโครงสร้างของแต่ละบล็อกในแฟ้มข้อมูลบนดิสก์

เนื่องจากแต่ละเรคอร์ดในบล็อกข้อมูลจะมีหนึ่ง slot ใน slot directory ที่เก็บตำแหน่งและความยาวของเรคอร์ด ทำให้เรคอร์ดต่าง ๆ ที่จัดเก็บในบล็อกเดียวกันมีหมายเลข slot (slot number) ต่างกันด้วย ดังนั้นแต่ละเรคอร์ดในแฟ้มข้อมูลจะมีหมายเลขเรคอร์ด (Record ID : RID) ที่แตกต่างกันเพื่อใช้ในการอ้างถึงเรคอร์ดที่ต้องการได้ ซึ่งหมายเลขเรคอร์ดจะประกอบด้วยหมายเลขบล็อก (block id) ที่เรคอร์ดถูกจัดเก็บอยู่ และหมายเลข slot ที่จัดเก็บตำแหน่งและความยาวของเรคอร์ดนั้น (RID = <block id , slot number>) ลักษณะของโครงสร้างบล็อกที่ออกแบบทำให้สามารถจัดเก็บข้อมูลได้ทั้งเรคอร์ดความยาวตายตัว และเรคอร์ดความยาวแปรได้



ภาพประกอบที่ 3.4 โครงสร้างของบล็อกข้อมูล

การเพิ่มเรคอร์ดในบล็อก

สำหรับวิธีการเพิ่มเรคอร์ดใหม่ในบล็อก ถ้าบล็อกมีเนื้อที่ว่างพอสำหรับเรคอร์ดนั้นจะทำการค้นหา slot ว่างที่มีค่าความยาวเป็น –1 หรือเพิ่ม slot ใหม่ และจัดเก็บเรคอร์ดลงในบล็อกในตำแหน่งที่ว่างบนบล็อก

การลบเรคอร์ดในบล็อก

เมื่อมีการลบเรคอร์ดในบล็อก จะไปกำหนดความยาวเรคอร์ดที่ถูกลบใน slot เป็น –1 เพื่อระบุว่า slot ว่างและเรคอร์ดถูกลบแล้ว โดยไม่ต้องลบข้อมูลจริงในบล็อก พร้อมทั้งทำการย้ายข้อมูลในเรคอร์ดมาอยู่ติดกัน ซึ่งอาจต้องมีการเปลี่ยนแปลงตำแหน่งเรคอร์ดที่จัดเก็บใน slot ให้ถูกต้องด้วย แต่จะเห็นว่าไม่มีการเปลี่ยนแปลงหมายเลขเรคอร์ดเก่าใดๆในแฟ้มข้อมูลเลย การย้ายข้อมูลในเรคอร์ดมาอยู่ติดกันก็เพื่อทำให้พื้นที่ว่างบนบล็อกอยู่ติดกัน จะทำให้ง่ายต่อการเพิ่มเรคอร์ดใหม่ในบล็อก

5. โครงสร้างของเรคอร์ด (record structure)

ข้อมูลในระบบจะถูกจัดเก็บเป็นเรคอร์ดในบล็อกของแฟ้มข้อมูล โครงสร้างของการจัดเก็บเรคอร์ดมี 2 แบบ ได้แก่ เรคอร์ดความยาวตายตัว และเรคอร์ดความยาวแปรได้

เรคอร์ดความยาวตายตัว (fixed-length record)

ทุกๆเรคอร์ดของโครงสร้างการจัดเก็บเรคอร์ดแบบเรคอร์ดความยาวตายตัว จะถูกจัดเก็บด้วยขนาดความยาวเท่ากัน จากตัวอย่างแฟ้มข้อมูลบัญชีสำหรับฐานข้อมูลธนาคารแห่งหนึ่ง แต่ละเรคอร์ดประกอบด้วยโครงสร้างข้อมูลดังนี้

```
type deposit = record
                    branch-name : char(22);
                    account-number : char(10);
                    balance : real;
end;
```

ถ้าสมมุติว่าสำหรับข้อมูลอักขระ (char) แต่ละอักขระใช้เนื้อที่หน่วยความจำ 1 ไบต์ และข้อมูลจำนวนจริงหรือ (real) ใช้เนื้อที่หน่วยความจำ 8 ไบต์ ดังนั้นสำหรับข้อมูลแต่ละเรคอร์ดในแฟ้มข้อมูลบัญชีจะใช้เนื้อที่ในหน่วยความจำ 40 ไบต์ เท่ากัน

เรคอร์ดความยาวแปรได้ (variable-length record)

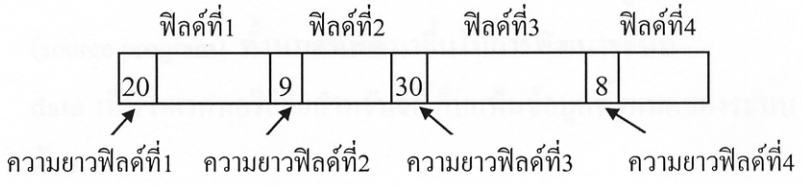
แต่ละเรคอร์ดของโครงสร้างการจัดเก็บเรคอร์ดแบบเรคอร์ดความยาวแปรได้ จะมีขนาดความยาวไม่เท่ากัน ลักษณะที่ต้องมีการจัดเก็บเรคอร์ดความยาวแปรได้อาจเนื่องมากจากเหตุผลหลายประการ เช่น เรคอร์ดมีบางฟิลด์ในเรคอร์ดที่มีชนิดข้อมูลต่างกันได้ในบางเรคอร์ด เรคอร์ดมีบางฟิลด์ในเรคอร์ดที่มีขนาดความยาวแปรได้ หรือเรคอร์ดมีบางฟิลด์ที่สามารถมีได้ค่ามากว่าหนึ่งค่า

เป็นต้น จากตัวอย่างแฟ้มข้อมูลบัญชีสำหรับฐานข้อมูลธนาคารแห่งหนึ่ง แต่ละเรคอร์ดประกอบด้วยโครงสร้างข้อมูลดังนี้

```
type account-list = record
        branch-name : char(22);
         account-info : array[1.. ∝] of
                              record
                                account-number : char(10);
                                balance : real;
                              end
end;
```

จากตัวอย่างจะเห็นว่าแต่ละเรคอร์ดในแฟ้มข้อมูลบัญชีจะใช้เนื้อที่ในหน่วยความจำไม่เท่ากันขึ้นอยู่กับจำนวนของ account-info

การจัดเก็บข้อมูลด้วยเรคอร์ดขนาดความยาวไม่คงที่สามารถทำได้หลายแบบ เช่น แต่ละเรคอร์ดจะเก็บจำนวนฟิลด์ตามด้วยข้อมูลแต่ละฟิลด์โดยมีเครื่องหมายคั่นระหว่างข้อมูลแต่ละฟิลด์ หรือแต่ละเรคอร์ดจะเก็บตำแหน่งข้อมูลแต่ละฟิลด์ในเรคอร์ดนั้น เป็นต้น สำหรับระบบที่พัฒนาขึ้นนี้ได้ใช้การจัดเก็บข้อมูลด้วยเรคอร์ดขนาดความยาวแปรได้โดยออกแบบให้แต่ละเรคอร์ดเก็บความยาวของฟิลด์ตามด้วยข้อมูลของฟิลด์ ดังตัวอย่างแสดงในภาพประกอบที่ 3.5

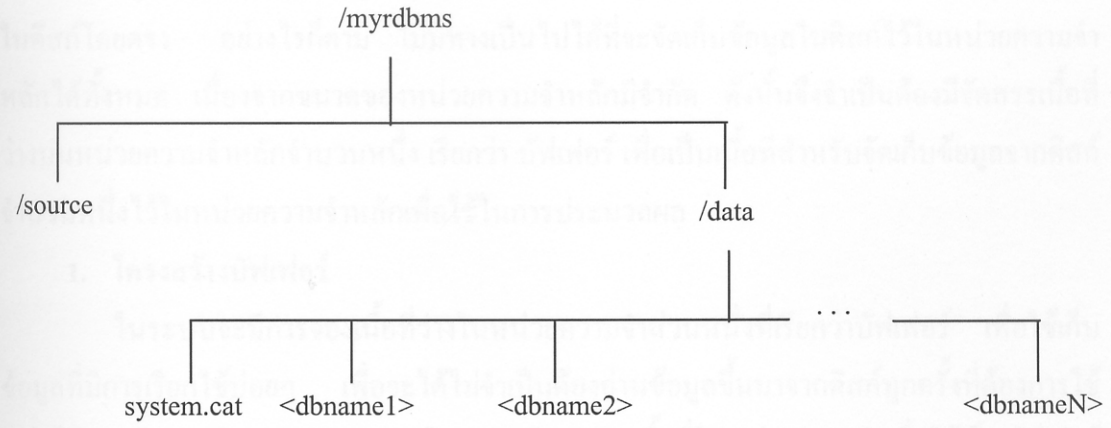


ภาพประกอบที่ 3.5 โครงสร้างของเรคอร์ดความยาวแปรได้

การสร้างและจัดการกับเรคอร์ดขนาดความยาวคงที่จะกระทำได้ง่ายกว่าเรคอร์ดขนาดความยาวไม่คงที่ ดังนั้นการจัดเก็บข้อมูลในฐานข้อมูลจะใช้โครงสร้างเรคอร์ดขนาดความยาวตายตัว ส่วนโครงสร้างเรคอร์ดความยาวแปรได้ที่ออกแบบไว้จะถูกใช้จัดเก็บเรคอร์ดของรีเลชันที่จะนำไปแสดงผลบนจอภาพในรูปของตารางที่จะกล่าวถึงในส่วนของการแสดงผล

6. การจัดเก็บแฟ้มข้อมูลในระบบ

การจัดเก็บแฟ้มข้อมูลทั้งหมดของระบบได้ออกแบบไว้ดังโครงสร้างต้นไม้ที่แสดงในภาพประกอบที่ 3.6 ลักษณะการจัดเก็บแฟ้มข้อมูลของระบบปฏิบัติการลีนุกซ์ โดยให้โหนดรากของต้นไม้ (tree) ที่จัดสร้างเป็นไดเรคทอรี (directory) ชื่อ myrdbms



ภาพประกอบที่ 3.6 การจัดเก็บแฟ้มข้อมูลในระบบ

การจัดเก็บแฟ้มข้อมูลทั้งหมดในระบบจะมี myrdbms เป็นไดเรคอทอรีที่เข้าสู่ระบบโปรแกรมที่พัฒนา ซึ่งแบ่งเป็นไดเรคทรีย่อย (subdirectory) อีกสองไดเรคทอรี ดังนี้

- **source** เป็นไดเรคทอรีย่อยสำหรับจัดเก็บแฟ้มข้อมูลที่เป็นโปรแกรมต้นฉบับ (source program) ทั้งหมดที่พัฒนาขึ้นในการพัฒนาระบบ
- **data** เป็นไดเรคทอรีย่อยสำหรับจัดเก็บแฟ้มข้อมูลทั้งหมดของระบบ ประกอบด้วย
  - **แฟ้มข้อมูลปทานุกรม** สำหรับจัดเก็บปทานุกรมข้อมูลทั้งหมดในระบบ จะมีเพียงแฟ้มข้อมูลเดียวชื่อ system.cat
  - **แฟ้มข้อมูลของฐานข้อมูล** สำหรับจัดเก็บข้อมูลทั้งหมดในฐานข้อมูลที่ผู้ใช้กำหนดขึ้น ในระบบสามารถมีแฟ้มข้อมูลประเภทนี้ได้มากกว่าหนึ่งแฟ้มข้อมูลขึ้นอยู่กับจำนวนฐานข้อมูลทั้งหมดในระบบที่ผู้ใช้สร้างขึ้น ในแฟ้มข้อมูลจะจัดเก็บเฉพาะข้อมูลที่เป็นของฐานข้อมูลเดียวกัน และมีชื่อแฟ้มข้อมูลเหมือนกับชื่อฐานข้อมูลที่ผู้ใช้สร้างขึ้น

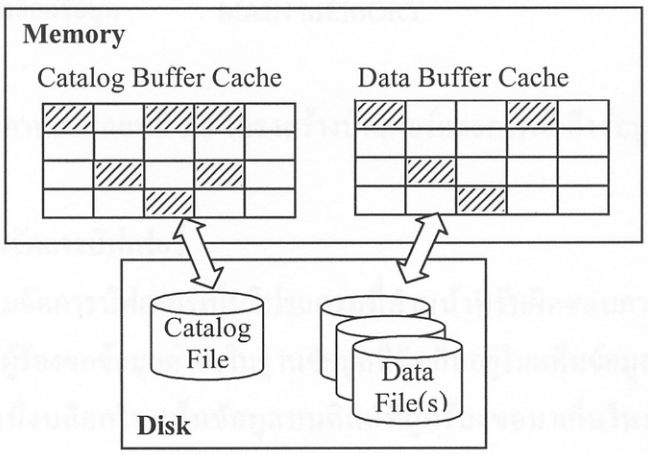
3.2 การจัดการบัฟเฟอร์

การดำเนินงานของระบบฐานข้อมูลมีเป้าหมายสำคัญอย่างหนึ่งคือการลดจำนวนการโอนย้ายข้อมูลระหว่างดิสก์กับหน่วยความจำหลัก วิธีหนึ่งที่สามารถลดจำนวนครั้งในการเข้าถึงข้อมูลในดิสก์คือการเก็บข้อมูลไว้ในหน่วยความจำหลักให้ได้จำนวนมากที่สุดเท่าที่จะเป็นไปได้ เพื่อที่โอกาสที่ข้อมูลที่ต้องการเข้าถึงอยู่ในหน่วยความจำหลักอยู่ก่อนแล้วมีมากขึ้น จึงไม่ต้องเข้าถึงข้อมูลในดิสก์โดยตรง อย่างไรก็ตาม ไม่มีทางเป็นไปได้ที่จะจัดเก็บข้อมูลในดิสก์ไว้ในหน่วยความจำหลักได้ทั้งหมด เนื่องจากขนาดของหน่วยความจำหลักมีจำกัด ดังนั้นจึงจำเป็นต้องมีจัดสรรเนื้อที่ว่างบนหน่วยความจำหลักจำนวนหนึ่ง เรียกว่า บัฟเฟอร์ เพื่อเป็นเนื้อที่สำหรับจัดเก็บข้อมูลจากดิสก์จำนวนหนึ่งไว้ในหน่วยความจำหลักเพื่อใช้ในการประมวลผล

1. โครงสร้างบัฟเฟอร์

ในระบบจะมีการจองเนื้อที่ว่างในหน่วยความจำส่วนหนึ่งที่เรียกว่าบัฟเฟอร์ เพื่อใช้เก็บข้อมูลที่มีการเรียกใช้บ่อยๆ เพื่อจะได้ไม่จำเป็นต้องอ่านข้อมูลขึ้นมาจากดิสก์ทุกครั้งที่ต้องการใช้ ทำให้ประหยัดเวลาในการทำงาน ในระบบมีการจองเนื้อที่ในหน่วยความจำเพื่อใช้เป็นบัฟเฟอร์สองส่วน ดังแสดงในภาพประกอบที่ 3.7 ซึ่งมีรายละเอียดดังนี้

- **Catalog buffer** เป็นบัฟเฟอร์สำหรับจัดเก็บข้อมูลส่วนของปทานุกรม
- **Data buffer** เป็นบัฟเฟอร์สำหรับจัดเก็บข้อมูลจากฐานข้อมูลที่กำลังใช้งานอยู่ โดยข้อมูลในบัฟเฟอร์นี้จะเป็นข้อมูลจากฐานข้อมูลเดียวกันเท่านั้น

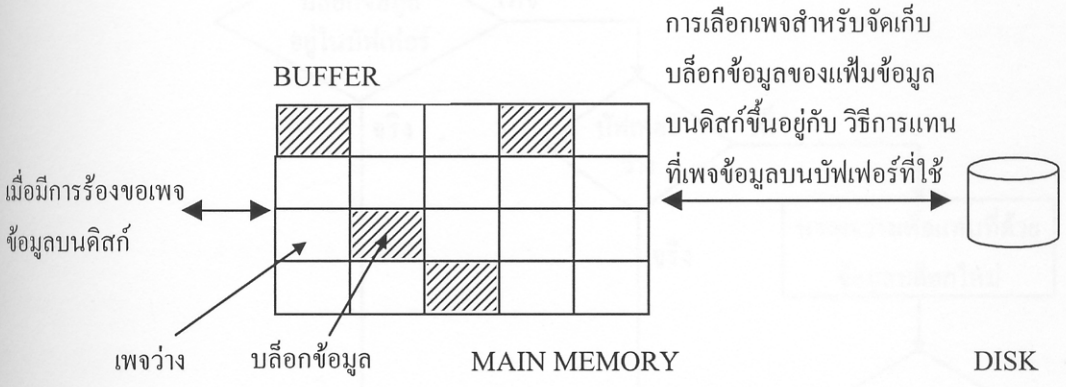


ภาพประกอบที่ 3.7 บัฟเฟอร์ที่ใช้ในระบบ

บัฟเฟอร์จะถูกแบ่งออกเป็นส่วนๆ ขนาดเท่ากัน เรียกว่า เพจ (page) แต่ละเพจจะมีขนาดเท่ากับขนาดข้อมูลหนึ่งบล็อกจากแฟ้มข้อมูลบนดิสก์ ในการโอนย้ายข้อมูลระหว่างดิสก์กับ

หน่วยความจำหลักจะเป็นการโอนย้ายข้อมูลหนึ่งบล็อกในแฟ้มข้อมูลบนดิสก์มาจัดเก็บในเพจของบัฟเฟอร์ หรือเป็นการนำข้อมูลหนึ่งเพจในบัฟเฟอร์บันทึกกลับลงในบล็อกของดิสก์

เมื่อผู้ใช้ต้องการเข้าถึงข้อมูลในแฟ้มข้อมูลบนดิสก์ ทั้งข้อมูลส่วนของปทานุกรมข้อมูลและข้อมูลจากฐานข้อมูลที่ผู้ใช้สร้างขึ้น ก็จะเข้าถึงข้อมูลโดยการค้นหาจากบัฟเฟอร์ที่จัดเตรียมไว้ก่อน หากพบข้อมูลนั้นก็จะจัดส่งข้อมูลให้กับผู้ใช้ แต่ถ้าหากไม่พบก็จะอ่านข้อมูลจากดิสก์มาบันทึกไว้ในบัฟเฟอร์ แล้วจัดส่งข้อมูลให้กับผู้ใช้ ภาพประกอบที่ 3.8 แสดงการร้องขอข้อมูลจากเพจบนบัฟเฟอร์และการติดต่อระหว่างเพจบนบัฟเฟอร์กับบล็อกบนดิสก์ ซึ่งข้อมูลเดียวกันในบัฟเฟอร์และดิสก์ ข้อมูลในบัฟเฟอร์จะเป็นข้อมูลที่ทันสมัยมากว่าข้อมูลในแฟ้มข้อมูลบนดิสก์ ดังนั้นจะต้องทำการบันทึกข้อมูลในบัฟเฟอร์กลับลงแฟ้มข้อมูลเพื่อให้ข้อมูลในดิสก์ถูกต้องทันสมัยเช่นกัน โดยจะมีโปรแกรมหนึ่งทำหน้าที่รับผิดชอบการจัดการเนื้อที่ในส่วนบัฟเฟอร์ เรียกว่า โปรแกรมจัดการบัฟเฟอร์ (buffer management)



ภาพประกอบที่ 3.8 โครงสร้างบัฟเฟอร์และการเข้าถึงข้อมูล

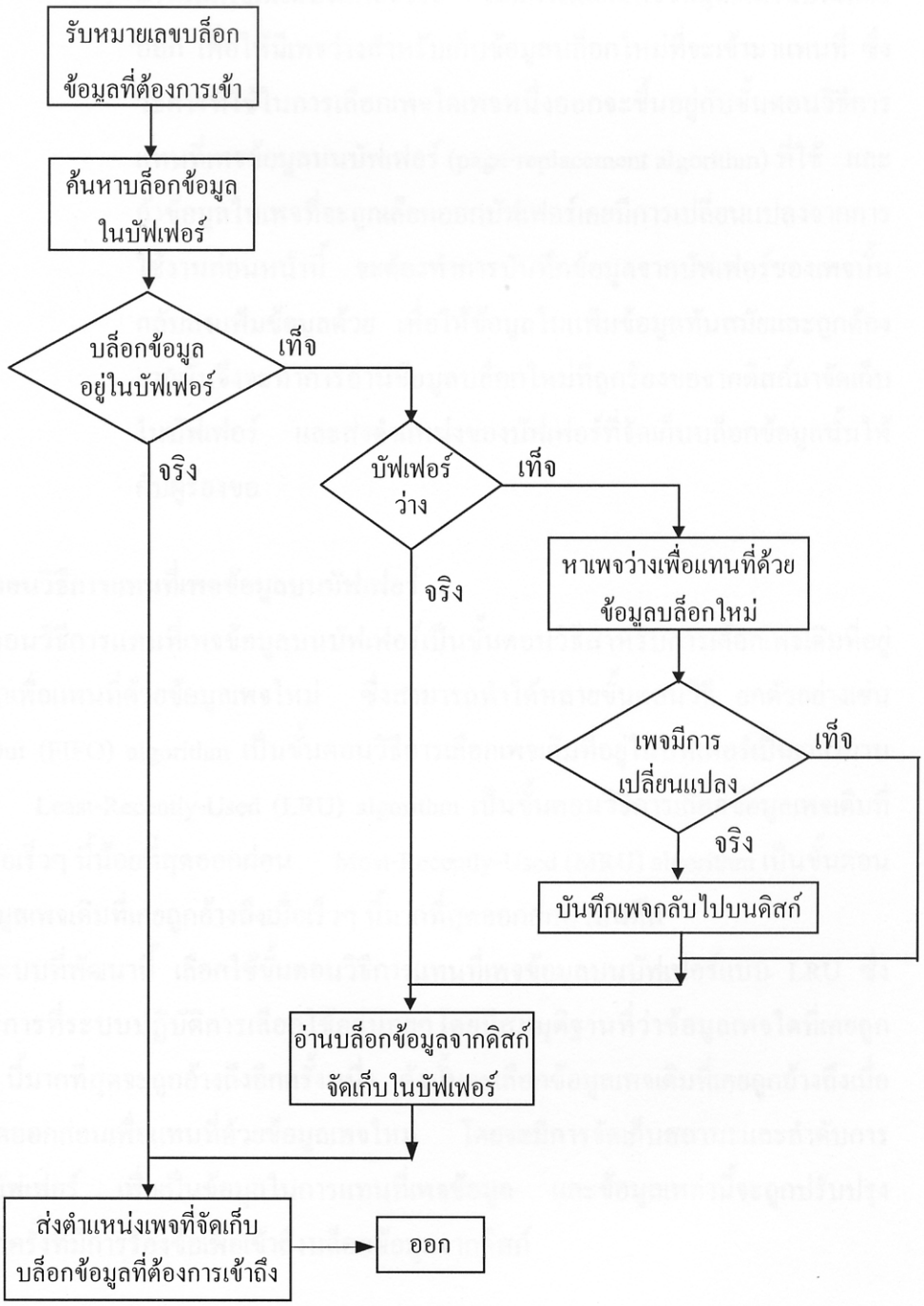
2. โปรแกรมจัดการบัฟเฟอร์

โปรแกรมจัดการบัฟเฟอร์เป็นโปรแกรมที่ทำหน้าที่รับผิดชอบการจัดสรรเนื้อที่ว่างในส่วนบัฟเฟอร์ให้กับผู้ร้องขอข้อมูลต่างๆในฐานข้อมูลที่จัดเก็บอยู่ในแฟ้มข้อมูลบนดิสก์ โดยจะทำหน้าที่อ่านข้อมูลหนึ่งบล็อกในแฟ้มข้อมูลบนดิสก์ที่ถูกร้องขอมาเก็บในหน่วยความจำในเพจว่างบนบัฟเฟอร์ และจัดเก็บรายละเอียดในการใช้งานบัฟเฟอร์ต่างๆ เช่น หมายเลขบล็อกของข้อมูลในแฟ้มข้อมูลที่อ่านขึ้นมาเก็บในเพจบนบัฟเฟอร์ หมายเลขเพจที่ถูกใช้งาน เป็นต้น เพื่อใช้เป็นข้อมูลในการจัดสรรเนื้อที่ว่างบัฟเฟอร์ และเมื่อมีการร้องขอบล็อกข้อมูลในแฟ้มข้อมูลบนดิกส์ก็จะสามารถเลือกข้อมูลในบัฟเฟอร์ให้กับผู้ร้องขอได้อย่างถูกต้อง

การเข้าถึงบล็อกข้อมูล

การเข้าถึงบล็อกข้อมูลจะค้นหาจากเพจบัฟเฟอร์ที่เตรียมไว้ก่อน ขั้นตอนในการเข้าถึงบล็อกข้อมูลสามารถแสดงได้ดังภาพประกอบที่ 3.9 ซึ่งขั้นตอนดังนี้

1) ถ้าข้อมูลเพจที่ถูกร้องขอมีถูกจัดเก็บอยู่ในบัฟเฟอร์เรียบร้อยแล้ว จะส่งตำแหน่งของบัฟเฟอร์ที่จัดเก็บข้อมูลเพจนั้นให้กับผู้ร้องขอ



ภาพประกอบที่ 3.9 ขั้นตอนการเข้าถึงบล็อกข้อมูลในแฟ้มข้อมูล

2) ถ้าข้อมูลเพจที่ถูกร้องข้อไม่ได้ถูกจัดเก็บอยู่ในบัฟเฟอร์ก็จะทำการจัดสรรเนื้อที่ว่างในบัฟเฟอร์เพื่อจัดเก็บข้อมูลเพจใหม่ที่ร้องขอ โดยแยกเป็นสองกรณีคือ

- **ถ้ามีเพจใดเพจหนึ่งในบัฟเฟอร์ว่าง** จะทำการอ่านข้อมูลบล็อกใหม่ที่ถูกร้องขอจากแฟ้มข้อมูลบนดิสก์เข้าสู่บัฟเฟอร์ และส่งตำแหน่งของบัฟเฟอร์ที่จัดเก็บข้อมูลบล็อกนั้นให้กับผู้ร้องขอ
- **ถ้าไม่มีเพจใดในบัฟเฟอร์ว่าง** จะมีการเลือกเพจข้อมูลเดิมในบัฟเฟอร์ออก เพื่อให้มีเพจว่างสำหรับเก็บข้อมูลบล็อกใหม่ที่จะเข้ามาแทนที่ ซึ่งวิธีการที่ใช้ในการเลือกเพจใดเพจหนึ่งออกจะขึ้นอยู่กับขั้นตอนวิธีการแทนที่เพจข้อมูลบนบัฟเฟอร์ (page-replacement algorithm) ที่ใช้ และถ้าข้อมูลในเพจที่จะถูกเลือกออกบัฟเฟอร์เคยมีการเปลี่ยนแปลงจากการใช้งานก่อนหน้านี้ จะต้องทำการบันทึกข้อมูลจากบัฟเฟอร์ของเพจนั้นกลับลงแฟ้มข้อมูลด้วย เพื่อให้ข้อมูลในแฟ้มข้อมูลทันสมัยและถูกต้อง จากนั้นจึงจะทำการอ่านข้อมูลบล็อกใหม่ที่ถูกร้องขอจากดิสก์มาจัดเก็บในบัฟเฟอร์ และส่งตำแหน่งของบัฟเฟอร์ที่จัดเก็บบล็อกข้อมูลนั้นให้กับผู้ร้องขอ

ขั้นตอนวิธีการแทนที่เพจข้อมูลบนบัฟเฟอร์

ขั้นตอนวิธีการแทนที่เพจข้อมูลบนบัฟเฟอร์เป็นขั้นตอนวิธีสำหรับการเลือกเพจเดิมที่อยู่ในบัฟเฟอร์ออกเพื่อแทนที่ด้วยข้อมูลเพจใหม่ ซึ่งสามารถทำได้หลายขั้นตอนวิธี ยกตัวอย่างเช่น First-In-First-Out (FIFO) algorithm เป็นขั้นตอนวิธีการเลือกเพจเดิมที่อยู่ในบัฟเฟอร์เป็นเวลานานที่สุดออกก่อน Least-Recently-Used (LRU) algorithm เป็นขั้นตอนวิธีการเลือกข้อมูลเพจเดิมที่เคยถูกอ้างถึงเมื่อเร็วๆ นี้น้อยที่สุดออกก่อน Most-Recently-Used (MRU) algorithm เป็นขั้นตอนวิธีการเลือกข้อมูลเพจเดิมที่เคยถูกอ้างถึงเมื่อเร็วๆ นี้มากที่สุดออกก่อน เป็นต้น

ในระบบที่พัฒนานี้ เลือกใช้ขั้นตอนวิธีการแทนที่เพจข้อมูลบนบัฟเฟอร์แบบ LRU ซึ่งเป็นขั้นตอนวิธีการที่ระบบปฏิบัติการเลือกใช้อยู่บ่อยๆ โดยมีสมมุติฐานที่ว่าข้อมูลเพจใดที่เคยถูกอ้างถึงเมื่อเร็วๆ นี้มากที่สุดจะถูกอ้างถึงอีกครั้งหนึ่ง ดังนั้นจะเลือกข้อมูลเพจเดิมที่เคยถูกอ้างถึงเมื่อเร็วๆนี้น้อยที่สุดออกก่อนเพื่อแทนที่ด้วยข้อมูลเพจใหม่ โดยจะมีการจัดเก็บสถานะและลำดับการใช้งานเพจบนบัฟเฟอร์ เพื่อเป็นข้อมูลในการแทนที่เพจข้อมูล และข้อมูลเหล่านี้จะถูกปรับปรุงเปลี่ยนแปลงทุกครั้งที่มีการร้องข้อเพื่อเข้าถึงบล็อกข้อมูลจากดิสก์

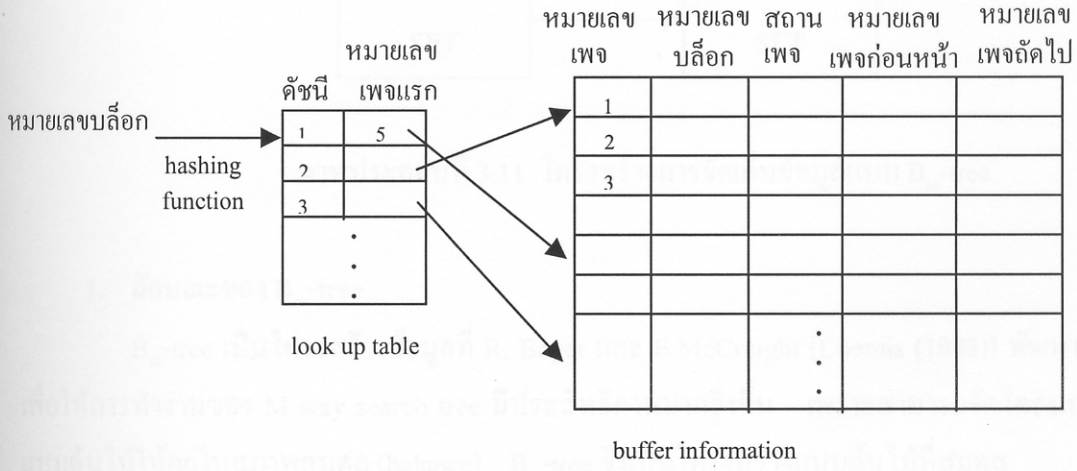
การค้นหาตำแหน่งเพจ

เนื่องจากจำนวนบล็อกข้อมูลในแฟ้มข้อมูลและจำนวนเพจในบัฟเฟอร์อาจไม่เท่ากัน และเพจในบัฟเฟอร์มีจำนวนมาก จึงต้องมีวิธีจัดการเพื่อทำให้การค้นหาเพจข้อมูลที่ต้องการสะดวกและรวดเร็วขึ้น

ในระบบที่พัฒนามีวิธีการค้นหาบล็อกข้อมูลที่ต้องการว่าถูกจัดเก็บไว้บนบัฟเฟอร์ในตำแหน่งเพจใด โดยจะนำหมายเลขบล็อกที่ต้องการไปคำนวณค่าดัชนีที่ใช้เรียกว่า hashing function เพื่อนำไปค้นหาเพจในบัฟเฟอร์ต่อไป แสดงดังภาพประกอบที่ 3.10 โดยระบบจะมีการจัดเก็บข้อมูลต่าง ๆ เพื่อเป็นประโยชน์ในการค้นหาเพจ ประกอบด้วย

- **look up table** สำหรับจัดเก็บข้อมูลค่าดัชนี และหมายเพจแรกในลิงค์ลิสต์ของกลุ่มเพจที่เก็บข้อมูลของหมายเลขบล็อกที่คำนวณด้วยวิธีการ hashing function เดียวกันแล้วให้ค่าดัชนีเดียวกัน เพื่อใช้ในการเริ่มต้นค้นหาเพจข้อมูลของบล็อกที่ต้องการ
- **buffer information** สำหรับจัดเก็บรายละเอียดของข้อมูลในเพจนั้น ได้แก่ หมายเลขบล็อกที่เป็นเจ้าของข้อมูลในเพจ สถานะของข้อมูลว่ามีการปรับปรุงแก้ไขข้อมูลในเพจหรือไม่ หมายเลขเพจก่อนหน้าและหมายเลขเพจถัดไปที่เก็บข้อมูลของบล็อกที่มีหมายเลขบล็อกที่ให้ค่าดัชนีเดียวกันเมื่อผ่านการคำนวณจาก hashing function ด้วยสมการของการหารเก็บเศษ เช่น 10 % 3 ผลลัพธ์ที่ได้คือ 1 ซึ่งสมการที่ใช้คำนวณหาค่าดัชนีในระบบที่พัฒนาขึ้นนี้คือ

ดัชนี = หมายเลขบล็อก % ขนาดของ lookup table



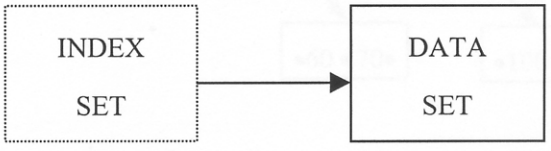
ภาพประกอบที่ 3.10 การค้นหาเพจข้อมูลในบัฟเฟอร์

ขั้นตอนในการค้นหาเพจ มีดังนี้

1) นำหมายเลขบล็อกมาคำนวณหาค่าดัชนีสำหรับการค้นหาเพจที่จัดเก็บข้อมูลของบล็อกที่ต้องการด้วยสมการ hashing function ที่ใช้ วิธีการนี้จะทำให้ไม่ต้องค้นหาทุกเพจบนบัฟเฟอร์จึงค้นหาเพจข้อมูลได้รวดเร็วขึ้น
2) นำค่าดัชนีที่ได้ไปหาตำแหน่งหมายเลขเพจแรกจาก lookup table
3) เริ่มค้นหาว่าเพจข้อมูลใดในลิงค์ลิสต์ที่เก็บข้อมูลของหมายเลขบล็อกที่ต้องการโดยใช้ข้อมูลจาก buffer information ซึ่งจะเก็บว่าเพจเก็บข้อมูลของหมายเลขบล็อกใด ถ้าค้นหาจนหมดลิงค์ลิสต์แล้วไม่พบบล็อกข้อมูลที่ต้องการแสดงว่าข้อมูลของบล็อกในหมายเลขบล็อกที่ต้องการไม่ได้จัดเก็บอยู่ในบัฟเฟอร์ จะต้องอ่านข้อมูลบล็อกใหม่มาแทนที่เพจในบัฟเฟอร์ด้วยวิธีการแทนที่เพจที่ใช้ในระบบต่อไป

3.3 การจัดการข้อมูลดัชนี

การจัดเก็บค่าคีย์ที่กำหนดในรีเลชันจะใช้โครงสร้างการจัดเก็บข้อมูลแบบ B_m-tree เพื่อความสะดวกในการเข้าถึงเรคอร์ดในรีเลชันที่มีค่าคีย์ที่สัมพันธ์กัน ภาพประกอบที่ 3.11 แสดงโครงสร้างการจัดเก็บข้อมูลแบบ B_m-tree ประกอบด้วยกลุ่มข้อมูลสองส่วน ได้แก่ กลุ่มดัชนี (index set) เป็นกลุ่มของเรคอร์ดสำหรับเก็บค่าคีย์เพื่อใช้อ้างถึงเรคอร์ดของรีเลชัน และกลุ่มข้อมูล (data set) เป็นกลุ่มของเรคอร์ดของรีเลชัน



ภาพประกอบที่ 3.11 โครงสร้างการจัดเก็บข้อมูลแบบ B_m-tree

1. ลักษณะของ B_m-tree

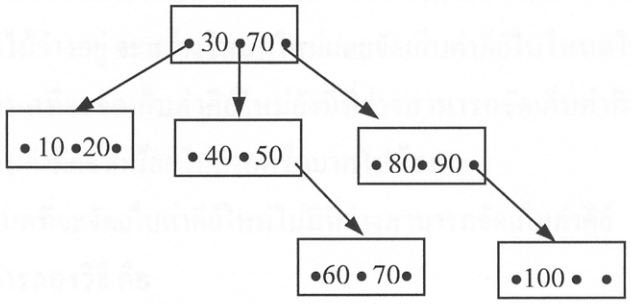
B_m-tree เป็นโครงสร้างข้อมูลที่ R. Bayer และ E.McCreight [Loomis (1983)] พัฒนาขึ้นเพื่อให้การทำงานของ M-way search tree มีประสิทธิภาพมากยิ่งขึ้น เพราะสามารถจัดโครงสร้างแบบต้นไม้ให้อยู่ในสภาพสมดุล (balance) B_m-tree จึงเป็นโครงสร้างแบบต้นไม้ที่สมดุล

โครงสร้างข้อมูลแบบต้นไม้

โครงสร้างข้อมูลแบบต้นไม้เป็นโครงสร้างข้อมูลที่ประกอบด้วยโหนดราก และโหนดถัดลงไปตั้งแต่สองโหนดขึ้นไป ถ้าจำนวนของคีย์ทั้งหมดในโครงสร้างแบบต้นไม้เท่ากับหรือน้อยกว่าจำนวนคีย์สูงสุดที่โหนดจะมีได้แล้ว ก็จะมีแต่เฉพาะโหนดรากเท่านั้น แต่ถ้าจำนวนคีย์มากกว่าจำนวนคีย์สูงสุดที่โหนดจะมีได้ โหนดรากจะถูกแตกออกเป็นสองโหนดล่าง เรียกโหนดที่อยู่ข้างบนว่าโหนดแม่ และเรียกโหนดอยู่ด้านล่างว่าโหนดลูก โดยโหนดที่อยู่ล่างสุดเรียกว่า โหนดใบ (leaf node) แต่ละโหนดจะชี้ไปยังโหนดที่อยู่ระดับต่ำกว่า ยกเว้นโหนดที่เป็นโหนดใบ แต่ละโหนดจะเก็บค่าคีย์อย่างมีลำดับจากมากไปน้อยหรือน้อยไปมากขึ้นอยู่กับขั้นตอนวิธีที่ใช้ในการจัดเก็บค่าคีย์

โครงสร้างข้อมูลแบบ M-way search tree

โครงสร้างข้อมูลแบบ M-way search tree เป็นโครงสร้างข้อมูลแบบไม่เป็นเส้นตรง (nonlinear data structure) โดยแต่ละโหนดจะมีจำนวนโหนดลูกได้ m โหนด หรือไม่มีโหนดลูกก็ได้ ดังภาพประกอบที่ 3.12



ภาพประกอบที่ 3.12 โครงสร้างของ 3-way search tree

โครงสร้างข้อมูลแบบ B_m-tree

โครงสร้างข้อมูลแบบ B_m-tree เป็นโครงสร้างข้อมูลแบบ (2m+1)-way search tree ที่ทุกโหนดยกเว้นโหนดรากจะมีจำนวนโหนดลูกได้มากที่สุด 2m+1 โหนด แต่ละโหนดจะเก็บค่าคีย์และตำแหน่งเรคอร์ดที่เก็บข้อมูลที่มีค่าคีย์ตรงกับค่าคีย์ที่เก็บในโหนดนั้นๆ ลักษณะของ B_m-tree ดังนี้

- B_m-tree จะมีค่าความสูงของต้นไม้มากกว่าหรือเท่ากับหนึ่ง หรือเป็นต้นไม้ว่าง (empty tree) ที่ไม่มีโหนด
- โหนดที่เป็นโหนดรากต้องมีจำนวนโหนดลูกอย่างน้อยสองโหนด ดังนั้นจะต้องเก็บค่าคีย์อย่างน้อยหนึ่งค่า

- ทุกโหนดที่ไม่ใช่โหนดใบ ยกเว้นโหนดราก จะต้องจำนวนโหนดลูกอย่างน้อย m+1 โหนด ดังนั้นแต่ละโหนดจะต้องเก็บค่าคีย์อย่างน้อย m ค่า
- ทุกโหนดที่เป็นโหนดใบจะอยู่ในระดับเดียวกันทั้งหมด

ตัวอย่างเช่น B_2-tree มีขนาด Order = 2 (m=2) จะต้องมีความสูงของต้นไม้อย่างน้อยเท่ากับหนึ่ง หรือว่างอยู่ โดยที่โหนดรากจะมีจำนวนโหนดลูกอย่างน้อยสองโหนด และโหนดอื่นๆ แต่ละโหนดยกเว้นโหนดใบจะต้องมีโหนดลูกอย่างน้อยสามโหนด (มีค่าคีย์อย่างน้อยสองค่า) แต่จำนวนโหนดลูกต้องไม่เกินห้าโหนด ซึ่งทุกโหนดที่เป็นโหนดใบจะอยู่ในระดับเดียวกัน

2. วิธีการเพิ่มและลบค่าคีย์ใน B_m-tree

วิธีการที่ใช้ในการเพิ่มค่าคีย์ใหม่และลบค่าคีย์เก่าในโครงสร้างข้อมูล B_m-tree จะทำให้โครงสร้างแบบต้นไม้คงสมดุลตลอดเวลา นั่นคือจำนวนระดับจากโหนดรากไปยังโหนดใบใดๆ ในโครงสร้างแบบต้นไม้จะต้องเหมือนกันไม่ว่าจะแตกไปทางไหน

การเพิ่มค่าคีย์

วิธีการเพิ่มค่าคีย์ใหม่ในโครงสร้างข้อมูล B_m-tree จะต้องค้นหาตำแหน่งโหนดที่จะจัดเก็บค่าคีย์โดยเริ่มต้นค้นหาค่าคีย์จากโหนดรากของโครงสร้างแบบต้นไม้ และจัดเก็บค่าคีย์ในโหนดนั้น โดยเรียงลำดับค่าคีย์จากน้อยไปมากหรือมากไปน้อย การเพิ่มค่าคีย์ใหม่สามารถทำได้ดังนี้

1) กรณีต้นไม้ว่างอยู่ จะสร้างโหนดใหม่และจัดเก็บค่าคีย์ในโหนดใหม่ที่สร้างขึ้น
2) กรณีโหนดที่จะจัดเก็บค่าคีย์ใหม่ยังมีที่ว่างสามารถจัดเก็บค่าคีย์ใหม่ในโหนดตามลำดับค่าคีย์จากน้อยไปมากหรือมากไปน้อย
3) กรณีโหนดที่จะจัดเก็บค่าคีย์ใหม่ไม่มีที่ว่างสามารถจัดเก็บค่าคีย์ จะมีเทคนิคในการจัดการสองวิธี คือ
   - **การแบ่งโหนด (split)** เป็นการเลือกค่าคีย์ที่มีค่าอยู่ตรงกลางระหว่างค่าคีย์ทั้งหมดในโหนดเมื่อรวมค่าคีย์ใหม่เข้าไปแล้ว และแยกกลุ่มของคีย์ที่มีค่าน้อยกว่าค่าคีย์กลางจัดเก็บไว้ที่โหนดเดิม แล้วนำกลุ่มค่าคีย์ที่มากกว่าค่าคีย์กลางไปเก็บในโหนดที่สร้างขึ้นมาใหม่ จากนั้นให้ย้ายค่าคีย์กลางไปจัดเก็บไว้ในโหนดแม่ ถ้าไม่มีโหนดแม่จะสร้างโหนดแม่ขึ้นมาใหม่และจัดเก็บค่าคีย์กลางในโหนดใหม่ที่สร้างขึ้น วิธีการนี้อาจทำให้ความสูงของโครงสร้างแบบต้นไม้เพิ่มขึ้นได้
   - **การกระจายคีย์ (redistribution)** เป็นการกระจายค่าคีย์ไปยังโหนดทางซ้ายที่อยู่ในระดับเดียวกัน (left sibling node) หรือโหนดทางขวาที่อยู่ในระดับเดียวกัน (right sibling node) และยังมีที่ว่างอยู่ ซึ่งจะมีผลกระทบต่อ

ค่าคีย์ที่จัดเก็บในโหนดแม่ด้วย เช่น กรณีเก็บค่าคีย์เรียงจากน้อยไปมากและต้องการกระจายคีย์ไปยังโหนดทางซ้าย วิธีการคือให้หาค่าน้อยที่สุดในโหนดเมื่อรวมค่าคีย์ใหม่เข้าไปแล้วเพื่อย้ายไปจัดเก็บในโหนดแม่ โดยจะต้องย้ายค่าคีย์จากโหนดแม่ที่มีค่าน้อยกว่าค่าคีย์ที่จะย้ายขึ้นมาใหม่นี้ ไปจัดเก็บในโหนดทางซ้ายซ้ายก่อน แล้วจึงจัดเก็บค่าคีย์ที่น้อยที่สุดนั้นในโหนดแม่แทนตำแหน่งค่าคีย์ที่ย้ายลงมา วิธีนี้ความสูงของโครงสร้างแบบต้นไม้จะไม่เพิ่มขึ้น

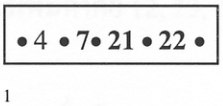
ตัวอย่างการเพิ่มค่าคีย์

ต่อไปนี้จะเป็นตัวอย่างในการเพิ่มคีย์ใหม่ในโครงสร้างข้อมูล B_2-tree ซึ่งแต่ละโหนดยกเว้นโหนดใบจะต้องมีโหนดลูกอย่างน้อยสามโหนด และมีค่าคีย์อย่างน้อยสองค่า โดยมีการจัดเก็บค่าคีย์แบบเรียงลำดับจากน้อยไปมาก การเพิ่มค่าคีย์จะเริ่มต้นต้นไม้ที่ว่างอยู่ ซึ่งสามารถแสดงลำดับการค้นหาและการเพิ่มคีย์ใหม่ได้ดังภาพประกอบที่ 3.13 ถึง ภาพประกอบที่ 3.21 ตามลำดับการเพิ่มค่าคีย์ที่จะกล่าวถึงต่อไปนี้ ค่าคีย์ที่ถูกเพิ่มใหม่จะแทนด้วยตัวอักขระตัวหนา และโหนดใหม่ที่สร้างขึ้นจะแทนด้วยเส้นประโดยมีหมายเลขกำกับที่มุมล่างซ้ายของแต่ละโหนด

- **เพิ่มค่าคีย์ 4** เริ่มต้นจากโครงสร้างข้อมูล B_2-tree ที่เป็นต้นไม้ว่าง ดังนั้นในกรณีนี้จะสร้างโหนดใหม่ (เริ่มต้นที่หมายเลข 1) แล้วใส่ค่าคีย์ใหม่จัดเก็บในโหนดใหม่ ดังภาพประกอบที่ 3.13

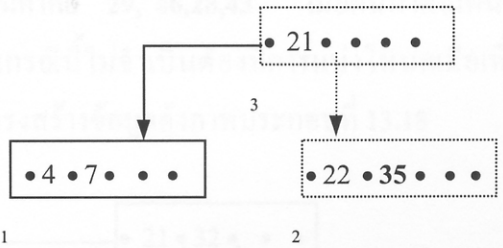
ภาพประกอบที่ 3.13 การเพิ่มค่าคีย์ใหม่ คือ 4 ในโครงสร้างข้อมูล B_2 - tree

- **เพิ่มค่าคีย์ 7 , 22 , 21** จะทำการเพิ่มที่โหนด 1 ซึ่งยังมีที่ว่างอยู่ ให้จัดเก็บค่าคีย์ใหม่เรียงตามลำดับจากน้อยไปมาก ดังภาพประกอบที่ 3.14



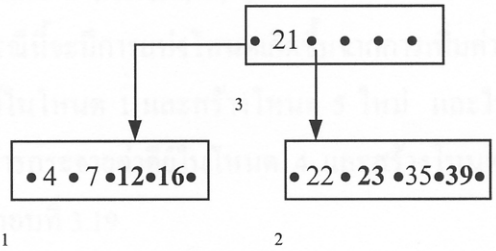
ภาพประกอบที่ 3.14 การเพิ่มค่าคีย์ 7, 22, 21 ในโครงสร้างข้อมูล B_2 – tree

- **เพิ่มค่าคีย์ 35** โหนดที่จะต้องจัดเก็บค่าคีย์คือโหนด 1 ซึ่งโหนดเต็ม ไม่มีเนื้อที่ว่างสำหรับค่าคีย์ใหม่ที่จะใส่ลงไป ในกรณีนี้จะใช้วิธีการแบ่งโหนดดังแสดงในภาพประกอบที่ 3.15 เทคนิคในการแบ่งโหนดจะเริ่มจากการเลือกคีย์อยู่ตรงกลางในโหนด 1 เมื่อรวมค่าคีย์ใหม่เข้าไปแล้วด้วย จะได้ค่าคีย์ที่อยู่ตรงกลางคือ 21 จากนั้นแยกกลุ่มของคีย์ที่มีค่าน้อยกว่าค่ากลางไปเก็บไว้ที่โหนด 1 เหมือนเดิม แล้วค่าคีย์ที่มากกว่าค่ากลางไปเก็บในโหนด 2 ที่สร้างขึ้นมาใหม่ ส่วนค่ากลางจะถูกนำขึ้นไปจัดเก็บในโหนดแม่ของโหนด 1 และ โหนด 2 ในกรณีนี้ยังไม่มีโหนดแม่อยู่ จะสร้างโหนดแม่ขึ้นมาใหม่เพื่อจัดเก็บค่าคีย์กลาง จะเห็นว่าความสูงของโครงสร้างแบบต้นไม้เพิ่มขึ้นอีกหนึ่งระดับ



ภาพประกอบที่ 3.15 การเพิ่มค่าคีย์ 35 ในโครงสร้างข้อมูล B_2 - tree

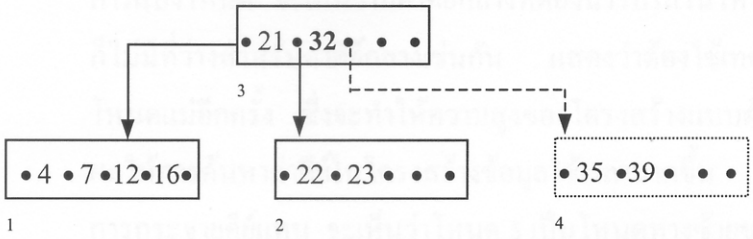
- **เพิ่มค่าคีย์ 12,23,39,16** เมื่อค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์ใหม่แต่ละค่าในกรณีนี้ไม่จำเป็นต้องมีการแบ่งโหนด ดังนั้นเมื่อเพิ่มค่าคีย์แต่ละค่าแล้วจะได้โครงสร้างข้อมูลดังภาพประกอบที่ 3.16



ภาพประกอบที่ 3.16 การเพิ่มค่าคีย์ 12, 23, 39 และ 16 ในโครงสร้างข้อมูล B_2–tree

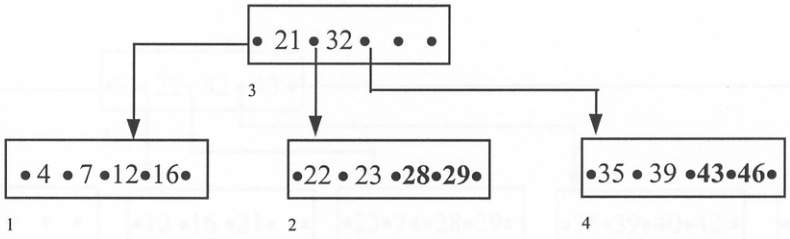
- **เพิ่มค่าคีย์ 32** เมื่อค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์ใหม่ จะได้ว่าต้องเพิ่มค่าคีย์ในโหนด 2 ซึ่งไม่มีเนื้อที่ว่างสำหรับใส่ค่าคีย์ใหม่ ในกรณีนี้จะกระจายค่าคีย์ไปจัดเก็บในโหนด 4 ที่สร้างขึ้นใหม่ และนำคีย์กลางไปจัดเก็บที่โหนดแม่

ในโหนด 3 เมื่อเพิ่มค่าคีย์ใหม่แล้วจะได้โครงสร้างข้อมูลดังภาพประกอบที่ 13.17 จะเห็นว่าในกรณีนี้ความสูงของโครงสร้างแบบต้นไม้ไม่ได้เพิ่มขึ้น



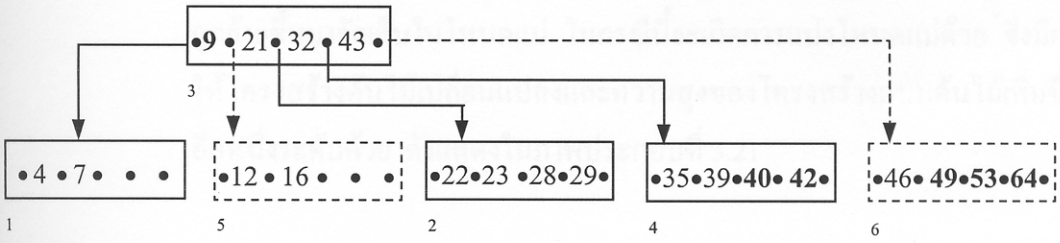
ภาพประกอบที่ 3.17 การเพิ่มค่าคีย์ 32 ในโครงสร้างข้อมูล B_5–tree

- **เพิ่มค่าคีย์ 29, 46,28,43** เมื่อค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์แต่ละค่า ในกรณีนี้ไม่จำเป็นต้องมีการแบ่งโหนดเมื่อเพิ่มค่าคีย์ใหม่แต่ละค่าแล้วจะได้โครงสร้างข้อมูลดังภาพประกอบที่ 13.18



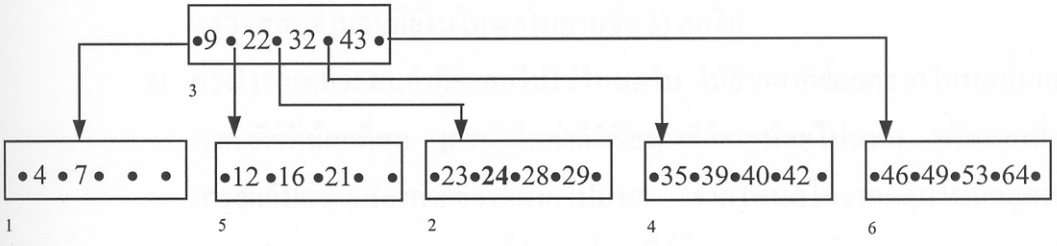
ภาพประกอบที่ 3.18 การเพิ่มค่าคีย์ 29, 46, 28, 43 ในในโครงสร้างข้อมูล B_2–tree

- **เพิ่มค่าคีย์ 64,9,49,53,40,42** เมื่อค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์แต่ละค่า ในกรณีนี้จะมีการแบ่งโหนดเกิดขึ้นจากการเพิ่มค่าคีย์ 9 ซึ่งทำให้เกิดการกระจายค่าคีย์ในโหนด 1 และสร้างโหนด 5 ใหม่ และในกรณีการเพิ่มค่าคีย์ 49 ทำให้เกิดการกระจายค่าคีย์ในโหนด 4 และสร้างโหนด 6 ใหม่ ดังแสดงในภาพประกอบที่ 3.19



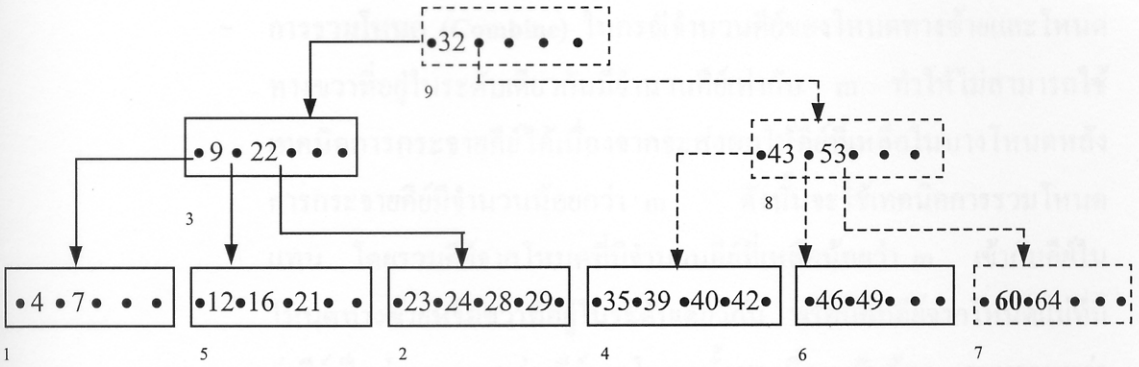
ภาพประกอบที่ 3.19 การเพิ่มค่าคีย์ 64, 9, 49, 53, 40, 42 ในโครงสร้างข้อมูล B_2–tree

- **เพิ่มค่าคีย์ 24** เมื่อค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์ใหม่ จะได้ว่าต้องเพิ่มค่าคีย์ 24 ในโหนด 2 แต่โหนด 2 ไม่มีที่ว่างสำหรับใส่ค่าคีย์ใหม่ ถ้าใช้เทคนิคการแบ่งโหนด จะเห็นว่ามีค่าคีย์กลางที่ต้องนำไปใส่ในโหนดแม่ และโหนดแม่ก็ไม่มีที่ว่างสำหรับค่าคีย์กลางเช่นกัน แสดงว่าต้องใช้เทคนิคการแบ่งโหนดที่โหนดแม่อีกครั้ง ซึ่งจะทำให้ความสูงของโครงสร้างแบบต้นไม้เพิ่มขึ้น และส่งผลให้การค้นหาค่าคีย์ในโครงสร้างข้อมูลใช้เวลามากขึ้น ดังนั้นจะใช้เทคนิคการกระจายคีย์แทน จะเห็นว่าโหนด 5 เป็นโหนดทางซ้ายของโหนด 2 และยังมีที่ว่างเหลืออยู่ ดังนั้นจะกระจายค่าคีย์ไปใส่ในโหนด 5 แทน ดังแสดงในภาพประกอบที่ 3.20 วิธีการคือให้หาค่าน้อยที่สุดในโหนด 2 เมื่อรวมค่าคีย์ใหม่เข้าไปแล้ว ค่าน้อยสุดที่ได้คือค่าคีย์ 22 โดยจะย้ายค่าคีย์จากโหนดแม่ในที่นี้คือ 21 ลงมาจัดเก็บในโหนดทางซ้ายก่อน และจึงจัดเก็บค่าคีย์ 22 ในโหนดแม่แทนตำแหน่งคีย์ 21 วิธีดังกล่าวจะไม่ทำให้ความสูงของโครงสร้างแบบต้นไม้จึงไม่เพิ่มขึ้น



ภาพประกอบที่ 3.20 การเพิ่มค่าคีย์ 24 ในโครงสร้างข้อมูล B2–tree

- **เพิ่มค่าคีย์ 60** ค้นหาตำแหน่งโหนดเพื่อจัดเก็บค่าคีย์ใหม่จะได้ว่าจะต้องเพิ่มค่าคีย์ในโหนด 6 และโหนดไม่มีที่ว่างสำหรับใส่ค่าคีย์ใหม่ และไม่สามารถใช้เทคนิคการกระจายคีย์ไปโหนดทางซ้ายหรือขวาได้ ดังนั้นจึงต้องใช้เทคนิคการแบ่งโหนด และจะเห็นโหนดแม่เป็นโหนดรากซึ่งไม่มีที่ว่างสำหรับค่าคีย์กลางที่ถูกย้ายขึ้นมาจัดเก็บในโหนดแม่ ในกรณีนี้จะเกิดการแบ่งโหนดแม่ด้วย จึงมีผลให้โครงสร้างต้นไม้เปลี่ยนแปลงและความสูงของโครงสร้างแบบต้นไม้เพิ่มขึ้นอีกหนึ่งระดับด้วย ดังแสดงในภาพประกอบที่ 3.21



ภาพประกอบที่ 3.21 การเพิ่มค่าคีย์ 60 ในโครงสร้างข้อมูล B_2–tree

การลบค่าคีย์

การลบค่าคีย์เก่าออกจากโครงสร้างข้อมูล B_m-tree จะต้องค้นหาตำแหน่งโหนดที่จะจัดเก็บค่าคีย์ที่ต้องการลบโดยเริ่มต้นค้นหาจากโหนดรากของโครงสร้างต้นไม้ วิธีการลบค่าคีย์ออกจากโหนดนั้นมีดังนี้

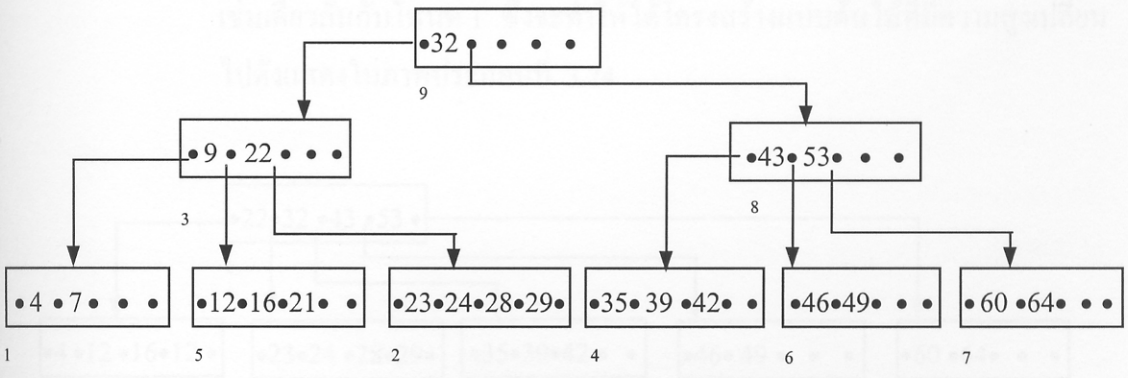
1) กรณีโหนดที่จะลบค่าคีย์ออกเป็นโหนดใบ ให้ย้ายค่าคีย์ออกจากโหนด และตรวจสอบจำนวนคีย์ในโหนดใบตามข้อ 3) ต่อไป
2) กรณีโหนดที่จะลบค่าคีย์ออกไม่ใช่โหนดใบ ให้ย้ายค่าคีย์ออกจากโหนดนั้นและหาค่าคีย์ที่น้อยที่สุด (กรณีเก็บค่าคีย์เรียงค่าคีย์จากน้อยไปมาก) หรือมากที่สุด (กรณีเก็บค่าคีย์เรียงค่าคีย์จากมากไปมาก) จากโหนดใบจากกลุ่มโหนดลูกทางขวา (successor) มาแทนที่ตำแหน่งค่าคีย์ที่ลบออกจากโหนด และตรวจสอบจำนวนคีย์ในโหนดใบตามข้อ 3) ต่อไป
3) ตรวจสอบจำนวนคีย์ที่เหลือในโหนดหลังลบคีย์ออกแล้ว ถ้าน้อยกว่า m จะมีเทคนิคการจัดการกับโหนดที่มีจำนวนคีย์ที่เหลือน้อยว่า m ได้สองแบบ คือ
   - **การกระจายคีย์** เป็นการกระจายค่าคีย์จากโหนดทางซ้ายหรือโหนดทางขวาที่อยู่ในระดับเดียวกันและจำนวนคีย์มากกว่า m ซึ่งจะมีผลกระทบต่อค่าคีย์ที่จัดเก็บในโหนดแม่ด้วย เช่น กรณีจัดเก็บคีย์เรียงจากน้อยไปมากและกระจายคีย์มาจากโหนดทางซ้ายในระดับเดียวกัน วิธีการคือให้หาค่าคีย์ที่มากสุดในโหนดทางซ้ายเพื่อย้ายไปจัดเก็บในโหนดแม่ในตำแหน่งค่าคีย์ที่มากกว่าคีย์ที่จะย้ายขึ้นมา โดยต้องย้ายค่าคีย์ที่ถูกแทนที่จากโหนดแม่มาจัดเก็บในโหนดที่มีจำนวนคีย์หน้อยกว่า m ก่อน วิธีการนี้จะไม่ทำให้ระดับความสูงของโครงสร้างต้นไม้เปลี่ยนไป

- **การรวมโหนด (Combine)** ในกรณีจำนวนคีย์ของโหนดทางซ้ายและโหนดทางขวาที่อยู่ในระดับเดียวกันมีจำนวนคีย์เท่ากับ m ทำให้ไม่สามารถใช้เทคนิคการกระจายคีย์ได้เนื่องจากจะส่งผลให้คีย์ที่เหลือในบางโหนดหลังการกระจายคีย์มีจำนวนน้อยกว่า m ดังนั้นจะใช้เทคนิคการรวมโหนดแทน โดยรวมคีย์จากโหนดที่มีจำนวนคีย์ที่เหลือน้อยว่า m เข้ากับคีย์ในโหนดทางซ้ายหรือขวาที่อยู่ในระดับเดียวกัน พร้อมกับคีย์จากโหนดแม่ที่มีค่าคีย์เป็นค่ากลางระหว่างคีย์จากโหนดทั้งสองที่รวมกันด้วย และหากพบว่าคีย์ที่เหลือในโหนดแม่มีจำนวนน้อยกว่า m จะต้องใช้วิธีการจัดการกับโหนดแม่เช่นเดียวกับที่กล่าวมาแล้วในข้อ 3)

ตัวอย่างการลบคีย์

ต่อไปนี้จะเป็นตัวอย่างในการลบคีย์เก่าจากโครงสร้างข้อมูล B_2-tree โดยเริ่มต้นโครงสร้างข้อมูลในภาพประกอบที่ 3.21 ซึ่งสามารถแสดงในลำดับการค้นหาและการลบคีย์เก่า ดังแสดงในภาพประกอบที่ 3.22 ถึงภาพประกอบที่ 3.25 ตามลำดับการลบคีย์ต่อไปนี้

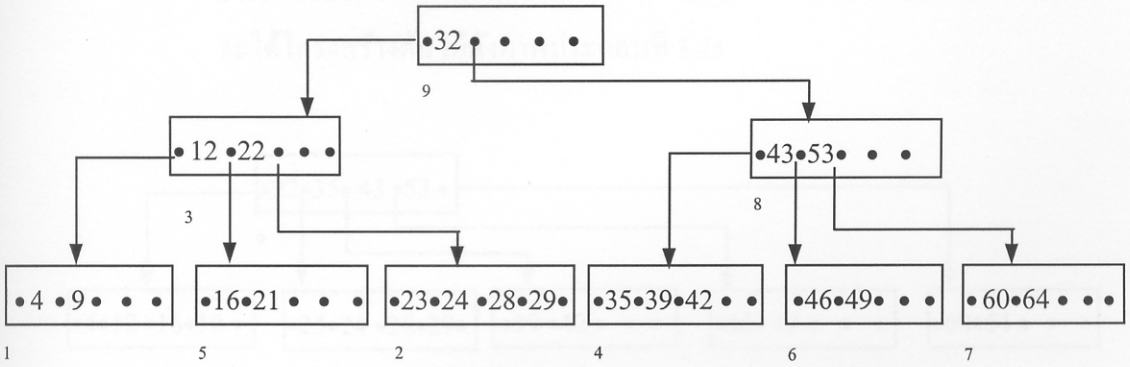
- **ลบค่าคีย์ 40** เมื่อค้นหาโหนดที่จัดเก็บค่าคีย์ที่ต้องการลบ กรณีนี้จะง่ายที่สุดเนื่องจากโหนด 4 ที่เก็บค่าคีย์ที่ต้องการลบอยู่มีจำนวนคีย์มากกว่าสอง เมื่อลบคีย์ 40 ออกจากโหนด 4 ก็จะยังไม่ทำให้จำนวนคีย์น้อยกว่าสอง ดังแสดงในภาพประกอบที่ 3.22



ภาพประกอบที่ 3.22 การลบค่าคีย์ 40 ออกจากโครงสร้างข้อมูล B_2–tree

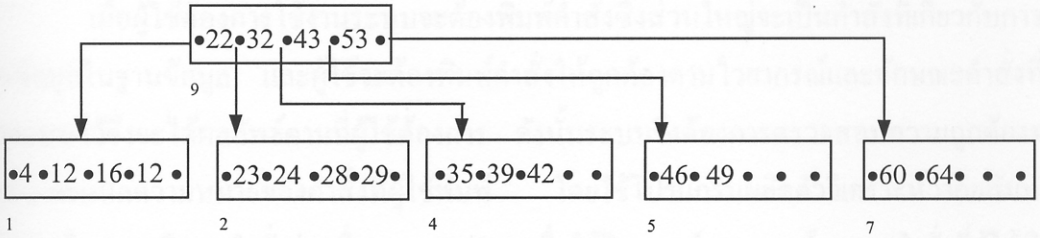
- **ลบค่าคีย์ 7** เมื่อค้นหาโหนดที่จัดเก็บค่าคีย์ที่ต้องการลบ ในกรณีนี้เมื่อลบค่าคีย์ออกจากโหนด 1 แล้วทำให้จำนวนคีย์น้อยกว่าสอง และโหนดทางขวาในระดับเดียวกันมีจำนวนคีย์มากกว่าสอง ดังนั้นจะใช้วิธีกระจายคีย์ใหม่โดยการหาค่าคีย์น้อยที่สุดในโหนด 5 ในที่นี้คือ 12 และย้ายมาจัดเก็บในโหนดแม่แทน

ที่คีย์ 9 และย้ายค่าคีย์ 9 จากโหนดแม่มาจัดเก็บในโหนด 1 จะทำให้จำนวนคีย์ของแต่ละโหนดไม่น้อยกว่าสอง ดังแสดงในภาพประกอบที่ 3.23



ภาพประกอบที่ 3.23 การลบค่าคีย์ 7 ออกจากโครงสร้างข้อมูล B_2–tree

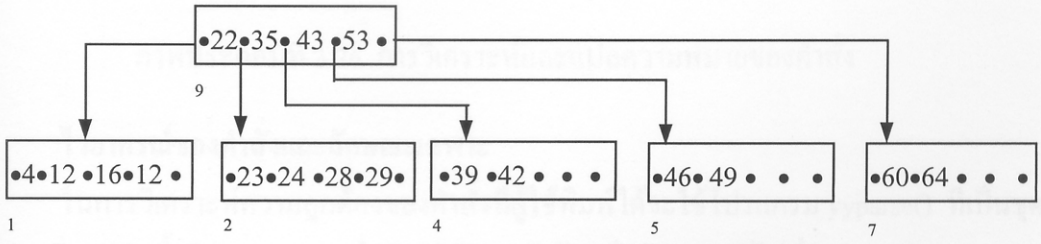
- **ลบค่าคีย์ 9** เมื่อค้นหาโหนดที่จัดเก็บค่าคีย์ที่ต้องการลบ จะได้ว่าในกรณีนี้เมื่อลบค่าคีย์ออกจากโหนด 1 จะทำให้จำนวนคีย์น้อยกว่าสอง และไม่สามารถกระจายค่าคีย์มาจากโหนดทางขวาในระดับเดียวกันได้ด้วย เพราะจะทำให้ค่าคีย์ในโหนด 5 น้อยกว่าสองเช่นกัน ดังนั้นจะใช้เทคนิคการรวมโหนดโดยรวมค่าคีย์ที่เหลือทั้งหมดในโหนด 1 และโหนด 6 พร้อมกับย้ายค่าคีย์จากโหนดแม่ในที่นี้คือ 12 จากการรวมโหนดจะเห็นว่า โหนดแม่มีจำนวนคีย์น้อยกว่าสอง เพราะเหลือค่าคีย์เดียว คือ 22 ดังนั้นจะใช้วิธีการจัดการกับโหนดแม่ในลักษณะเช่นเดียวกันกับโหนด 1 ซึ่งจะทำให้ได้โครงสร้างแบบต้นไม้ที่มีความสูงเปลี่ยนไปดังแสดงในภาพประกอบที่ 3.24



ภาพประกอบที่ 3.24 การลบค่าคีย์ 9 ออกจากโครงสร้างข้อมูล B_2–tree

- **ลบค่าคีย์ 32** เมื่อค้นหาโหนดที่จัดเก็บค่าคีย์ที่ต้องการลบ พบว่าค่าคีย์ที่ต้องการลบอยู่ในโหนด 9 และไม่ใช่โหนดใบ ในกรณีนี้จะต้องหาค่าคีย์ที่น้อยที่สุดจาก

โหนดใบที่อยู่กลุ่มโหนดลูกทางขวามาแทนที่ตำแหน่งค่าคีย์ที่ต้องการลบในโหนดที่ 9 ซึ่งจะได้ค่าคีย์ที่น้อยที่สุดในโหนดที่ 4 คือค่าคีย์ 35 ในกรณีนี้เมื่อย้ายค่าคีย์ออกจากโหนดแล้วไม่ทำให้โหนดใบมีจำนวนคีย์น้อยกว่าสอง ดังนั้นจะได้โครงสร้างต้นไม้ดังภาพประกอบที่ 3.25



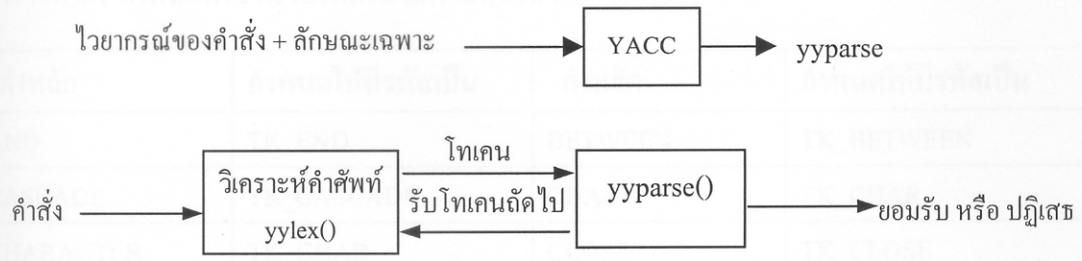
ภาพประกอบที่ 3.25 การลบค่าคีย์ 32 ออกจากโครงสร้างข้อมูล B_2–tree

3.4 การจัดการการติดต่อระหว่างผู้ใช้กับระบบ

ระบบที่พัฒนาขึ้นเป็นระบบจัดการฐานข้อมูล ผู้ใช้ระบบจะต้องสามารถติดต่อใช้งานฐานข้อมูลในระบบ เพื่อกำหนดโครงสร้างฐานข้อมูลและชนิดข้อมูลที่จะจัดเก็บในฐานข้อมูล การรับข้อมูลเข้าไปจัดเก็บในฐานข้อมูล รวมทั้งการปรับปรุงเปลี่ยนแปลงแก้ไขข้อมูลในฐานข้อมูล และการค้นคืนข้อมูลในระบบ ซึ่งการติดต่อระหว่างผู้ใช้กับระบบที่พัฒนาขึ้นนี้จะเป็นแบบพิมพ์บรรทัดคำสั่ง (command line) ผู้ใช้จะติดต่อใช้งานระบบได้โดยการพิมพ์คำสั่งต่างๆ ที่ผู้พัฒนาระบบกำหนดขึ้น และแจ้งผลลัพธ์ของคำสั่งให้ผู้ใช้ทราบทางจอภาพ สามารถดูรายละเอียดคำสั่งทั้งหมดที่ใช้ในโปรแกรมได้ในภาคผนวก ก

1. การรับคำสั่งจากผู้ใช้

เมื่อผู้ใช้ต้องการใช้งานระบบจะต้องพิมพ์คำสั่งซึ่งส่วนใหญ่จะเป็นคำสั่งที่เกี่ยวกับการจัดการข้อมูลในฐานข้อมูล และผู้ใช้จะต้องพิมพ์คำสั่งให้ถูกต้องตามไวยากรณ์และลักษณะคำสั่งที่ได้ออกแบบไว้จึงจะได้ผลลัพธ์ตามที่ผู้ใช้ต้องการ ดังนั้นระบบจึงต้องการตรวจสอบความถูกต้องของคำสั่งและแปลความหมายของคำสั่งที่ผู้ใช้พิมพ์ โดยใช้โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ YACC ในการผลิตชุดคำสั่งย่อยชื่อ yyparse() เพื่อใช้วิเคราะห์ความถูกต้องของคำสั่งที่ผู้ใช้พิมพ์ โดยทำงานร่วมกับโปรแกรมวิเคราะห์คำศัพท์ yylex() ที่ผู้วิจัยพัฒนาขึ้น ดังตัวอย่างแสดงในภาพประกอบที่ 3.26 โดยคำสั่งต่างๆ จะถูกส่งให้โปรแกรม yylex() และถูกแยกออกเป็นส่วนเล็กๆ เรียกว่า โทเคน (token) เพื่อส่งให้โปรแกรม yyparse() วิเคราะห์ความถูกต้องของคำสั่งที่ผู้ใช้พิมพ์



ภาพประกอบที่ 3.26 การวิเคราะห์และแปลความหมายของคำสั่ง

ไวยากรณ์ของคำสั่งและลักษณะเฉพาะ

ในการวิเคราะห์ความถูกต้องของคำสั่งที่ผู้ใช้พิมพ์ได้จะใช้โปรแกรม yyparse() ที่เป็นชุดคำสั่งย่อยที่ถูกผลิตขึ้นโดย YACC ซึ่งต้องมีข้อมูลเข้าต้องเป็นไวยากรณ์ไม่พึ่งบริบท (context-free grammar) และลักษณะเฉพาะ (attribute) รวมกันเรียกว่า ไวยากรณ์ลักษณะเฉพาะ (attributed grammar) ดังนั้นคำสั่งต่างๆที่ใช้ในระบบจึงต้องถูกออกแบบให้อยู่ในรูปของไวยากรณ์ลักษณะเฉพาะด้วย ซึ่งสามารถดูรายละเอียดได้ในภาคผนวก ข

การกำหนดโทเคน

ในการวิเคราะห์และแปลความหมายของคำสั่งจากผู้ใช้ต้องมีโปรแกรมวิเคราะห์คำศัพท์ yylex() ทำงานร่วมกับโปรแกรม yyparse() ซึ่งโปรแกรม lex() จะทำหน้าที่แยกโทเคนต่างๆ จากคำสั่งและส่งให้ yyparse ตรวจสอบและแปลความหมายของคำสั่ง โทเคนเป็นกลุ่มคำต่างๆ ในข้อความของคำสั่ง โทเคนทั้งหมดที่กำหนดในระบบแบ่งออกเป็นประเภทต่างๆ ประกอบด้วย

การกำหนดออกแบบการทำงานของโปรแกรม yylex() ประกอบด้วย

- **คำหลัก** (keyword) เป็นคำสงวนที่ใช้ในคำสั่ง คำหลักประกอบด้วยอักขระที่อักษรตัวเล็กกับอักษรตัวใหญ่ ไม่มีความแตกต่างกัน (case insensitive) และกำหนดให้มีรหัสต่าง ๆ ดังแสดงในตารางที่ 3.1
- **ตัวชี้เฉพาะ** (indentifier) เป็นชื่อเฉพาะที่ผู้ใช้กำหนดขึ้น อาจเป็นชื่อฐานข้อมูล ชื่อรีเลชัน หรือชื่อแอตทริบิว ซึ่งผู้ใช้ต้องไม่กำหนดตัวชี้เฉพาะซ้ำกับคำหลักที่มีในระบบ ตัวชี้เฉพาะขึ้นต้นด้วยอักษรภาษาอังกฤษ และอาจตามด้วยอักษรภาษาอังกฤษ ตัวเลข หรือ "_" ตั้งแต่หนึ่งตัวขึ้นไป โดยที่อักษรตัวเล็กกับอักษรตัวใหญ่ ไม่มีความแตกต่างกัน กำหนดให้มีรหัสเป็น TK_IDENT
- **ค่าคงที่จำนวนเต็ม** ประกอบด้วยตัวเลขฐานสิบ ตั้งแต่หนึ่งตัวขึ้นไป กำหนดให้มีรหัสเป็น TK_INT

ตารางที่ 3.1 คำหลักที่ใช้ในโปรแกรมวิเคราะห์คำสั่ง

คำหลัก	กำหนดให้มีรหัสเป็น	คำหลัก	กำหนดให้มีรหัสเป็น
AND	TK_END	BETWEEN	TK_BETWEEN
CASCADE	TK_CASCADE	CHAR	TK_CHAR
CHARACTER	TK_CHAR	CLOSE	TK_CLOSE
CREATE	TK_CREATE	DATABASE	TK_DATABASE
DATABASES	TK_DATABASES	DATE	TK_DATE
DEFAULT	TK_DEFAULT	DELETE	TK_DELETE
DESC	TK_DESC	DESCRIBE	TK_DESCRIBE
DROP	TK_DROP	FOREIGN	TK_FOREIGN
FROM	TK_FROM	IN	TK_IN
INSERT	TK_INSERT	INTO	TK_INTO
KEY	TK_KEY	NOT	TK_NOT
NULL	TK_NULL	NUM	TK_NUM
ON	TK_ON	PRIMARY	TK_PRIMARY
QUIT	TK_QUIT	REFERENCES	TK_REFERENCE
SECONDARY	TK_SECONDARY	SELECT	TK_SELECT
SET	TK_SET	SHOW	TK_SHOW
TABLE	TK_TABLE	TABLES	TK_TABLES
UPDATE	TK_UPDATE	USE	TK_USE
VALUES	TK_VALUES	WHERE	TK_WHERE

- **ค่าคงที่จำนวนจริง** ประกอบด้วยเลขฐานสิบ และจุดทศนิยม โดยที่จุดทศนิยมจะอยู่หน้าสุดหรือหลังสุดก็ได้ เช่น 25.5 .9 3. เป็นต้น กำหนดให้มีรหัสเป็น TK_REAL
- **สายอักขระ** เขียนขึ้นต้นด้วยอักขระ " หรือ ' ตามด้วยอักขระใดๆ ที่ไม่ใช่ " หรือ ' ตั้งแต่หนึ่งตัวขึ้นไป ตามด้วยอักขระ " หรือ ' แต่ต้องเป็นอักขระเดียวกับกับอักขระเริ่มต้น กำหนดให้มีรหัสเป็น TK_STRING

- **ตัวดำเนินการเปรียบเทียบ** กำหนดให้มีรหัส ดังต่อไปนี้

ตัวดำเนินการ	กำหนดให้มีรหัสเป็น
=	TK_EQ
<	TK_LT
<=	TK_LE
>	TK_GT
>=	TK_GE
!=	TK_NE

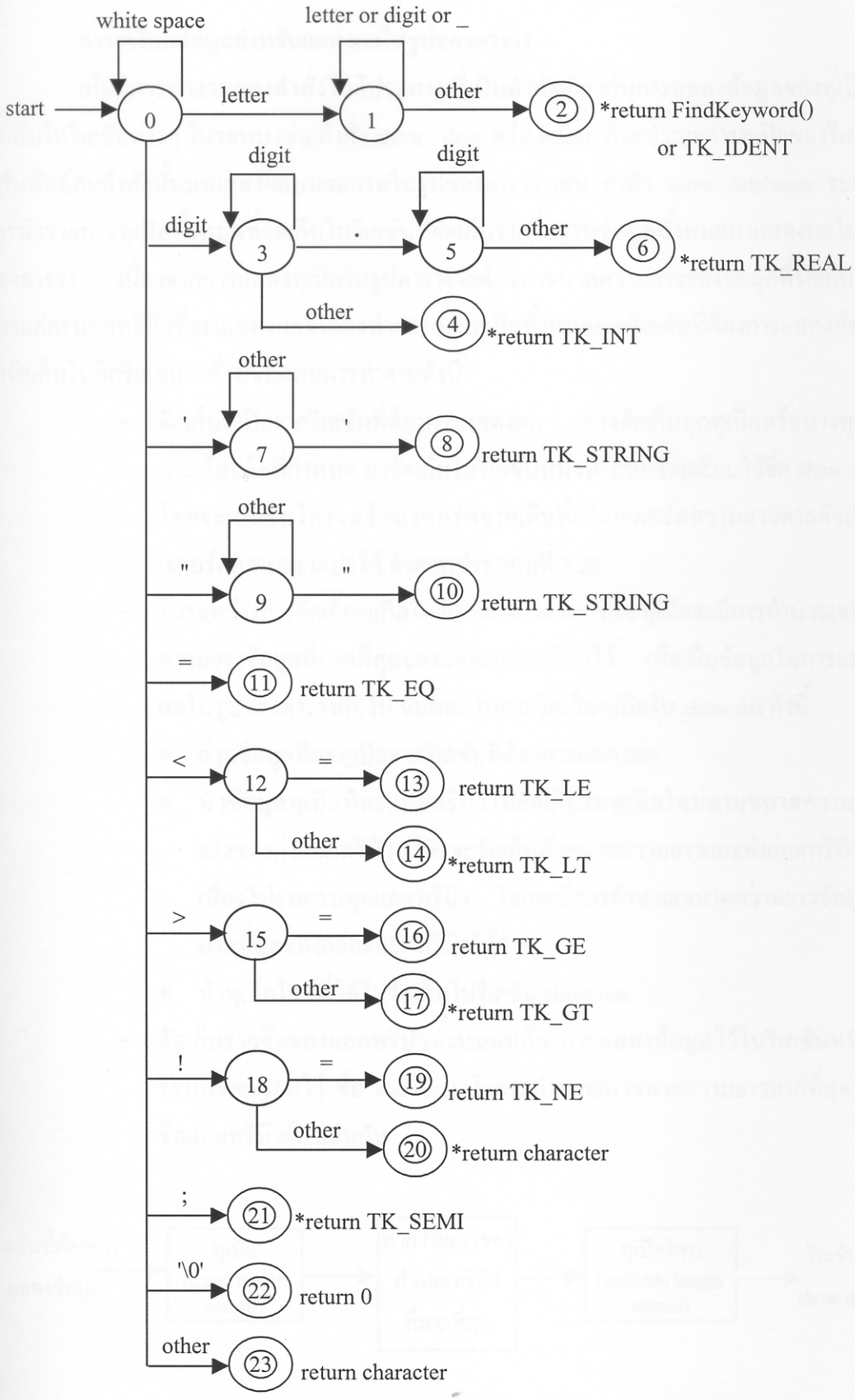
- **โทเคนอื่น ๆ** ได้แก่ ; กำหนดให้มีรหัสเป็น TK_SEMI สำหรับใช้เป็นตัวสิ้นสุดคำสั่งที่ใช้ในโปรแกรม ส่วนอักขระอื่น ๆ จะให้ค่าตามรหัสแอสกี้ของอักขระนั้นซึ่งอยู่อยู่ในช่วง 1-255

การแยกโทเคน

โปรแกรมวิเคราะห์คำศัพท์ yylex() จะทำหน้าที่แยกโทเคนจากคำสั่งที่ผู้ใช้พิมพ์เข้ามาเพื่อส่งให้โปรแกรม yyparse() ภาพประกอบที่ 3.27 แสดงวิธีการแยกโทเคนของโปรแกรม yylex() ที่พัฒนาขึ้น เพื่อรับแต่ละอักขระเข้ามาเพื่อทำการแยกโทเคน โดยเครื่องหมายวงกลมและหมายเลขจะแสดงถึงสถานะการทำงาน ซึ่งเริ่มจากสถานะการทำงานเริ่มต้นคือ start ถ้าเป็นเครื่องหมายวงกลมซ้อนกันจะหมายถึงสถานะสิ้นสุดการทำงานแยกโทเคนและสามารถทำการแยกโทเคนได้หนึ่งโทเคน ส่วนเครื่องหมาย * หมายถึง การคืน (unget) อักขระที่รับเข้ามาหนึ่งตัว นั่นคือเมื่อรับอักขระเข้ามาแล้วพบสถานะที่มีเครื่องหมาย * จะต้อง unget อักขระนั้นแล้วจึงทำการรับอักขระใหม่เพื่อเข้าสู่สถานะเริ่มต้นของการแยกโทเคนใหม่อีกครั้ง

2. การแสดงผลลัพธ์ของคำสั่ง

เมื่อผู้ใช้พิมพ์คำสั่งเพื่อใช้งานระบบแล้วจะต้องมีการแสดงผลลัพธ์ของคำสั่งที่ผู้ใช้พิมพ์เข้าไป โดยผลลัพธ์ของคำสั่งสามารถแสดงได้สองแบบ ได้แก่ ข้อความ ซึ่งอาจเป็นข้อความเพื่อแสดงข้อผิดพลาดในการพิมพ์คำสั่งของผู้ใช้ หรือเป็นผลลัพธ์จากการทำงานตามคำสั่งก็ได้ และตารางข้อมูล ซึ่งเป็นการแสดงข้อมูลของรีเลชันในรูปของตาราง



ภาพประกอบที่ 3.27 แผนภาพการแยกโทเคนของคำสั่งที่ใช้ในระบบ

การเตรียมข้อมูลสำหรับแสดงผลในรูปของตาราง

เมื่อมีการทำงานของคำสั่งในโปรแกรมที่เป็นคำสั่งเกี่ยวกับการแสดงข้อมูลของทุเปิลที่จัดเก็บในรีเลชันต่างๆ ในระบบ เช่น คำสั่ง show , desc หรือ select ก็จะนำรายการทุเปิลของรีเลชันที่สัมพันธ์กับคำสั่งนั้นมาแสดงผลบนจอภาพในรูปของตาราง เช่น คำสั่ง show databases จะเป็นการนำรายการทุเปิลทั้งหมดที่จัดเก็บในรีเลชันที่จัดเก็บรายชื่อฐานข้อมูลทั้งหมดมาแสดงผลในรูปของตาราง เนื่องจากการแสดงทุเปิลในรูปตารางจะต้องหาขนาดความยาวของข้อมูลที่จัดเก็บจริงของแต่ละแอตทริบิวที่จะแสดงผลจึงต้องทำการอ่านทุเปิลทั้งหมดจากรีเลชันที่ต้องการแสดงข้อมูลมาจัดเก็บในอีกรีเลชัน ซึ่งมีขั้นตอนการทำงานดังนี้

- จัดเก็บทุเปิลจากรีเลชันที่ต้องการแสดงผล อาจจัดเก็บทุกทุเปิลหรือบางทุเปิลตามเงื่อนไขที่กำหนด มาจัดเก็บในรีเลชันหนึ่งที่ระบบจัดเตรียมไว้ชื่อ show.data โดยจะเปลี่ยนโครงสร้างเรคอร์ดจากเดิมที่เป็นเรคอร์ดความยาวตายตัวเป็นเรคอร์ดความยาวแปรได้ ดังภาพประกอบที่ 3.28
- ในระหว่างการจัดเก็บทุเปิลในรีเลชัน show.dat ทีละทุเปิลจะมีการคำนวณขนาดความยาวข้อมูลที่มากที่สุดของแต่ละแอตทริบิวไว้ เพื่อเป็นข้อมูลในการแสดงผลในรูปของตารางต่อไป ขั้นตอนในการจัดเก็บทุเปิลใน show.dat ดังนี้
  - อ่านข้อมูลทีละทุเปิลจากรีเลชันที่ต้องการแสดงผล
  - นำข้อมูลทุเปิลทีละแอตทริบิวไปจัดเก็บในทุเปิลใหม่ตามขนาดความยาวจริงของค่าแอตทริบิว ซึ่งจะจัดเก็บทั้งขนาดความยาวและค่าแอตทริบิวต่อเนื่องไปจนครบทุกแอตทริบิว โดยจะมีการคำนวณขนาดความยาวข้อมูลที่มากที่สุดของแต่ละแอตทริบิวไว้ด้วย
  - นำทุเปิลใหม่ที่ได้ไปจัดเก็บในรีเลชัน show.dat
- จัดเก็บรายชื่อของแอตทริบิวทั้งหมดที่ต้องการแสดงข้อมูลไว้ในรีเลชันหนึ่งที่ระบบจัดเตรียมไว้ ชื่อ show.col โดยจะคำนวณหาขนาดความยาวมากที่สุดของชื่อแอตทริบิวด้วยเช่นกัน

ภาพประกอบที่ 3.28 การจัดเตรียมทุเปิลสำหรับการแสดงผลบนจอภาพในรูปของตาราง

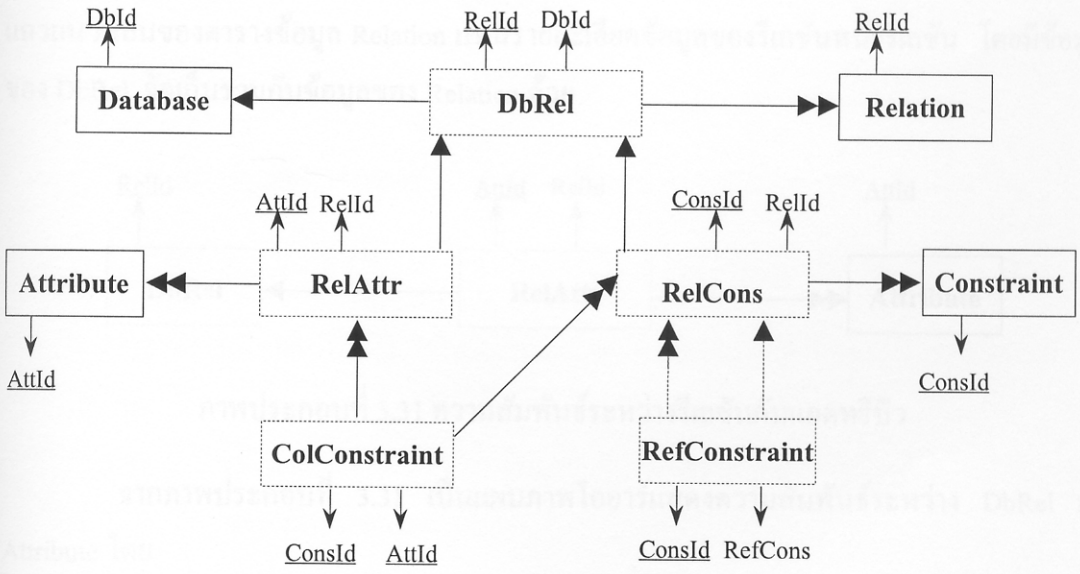
การนำข้อมูลมาแสดงผลในรูปของตาราง

เมื่อทำการเตรียมข้อมูลสำหรับแสดงผลเรียบร้อยแล้วจะนำทุเปิลทั้งหมดในรีลชัน show.col และ show.dat มาแสดงผลในรูปของตาราง โดยมีขั้นตอนการทำงานดังนี้

- อ่านทุเปิลจากรีเลชัน show.col มาแสดงผลเป็นส่วนหัวของตาราง โดยคำนวณตำแหน่งในการพิมพ์เส้นตารางไว้ด้วย
- อ่านทุเปิลจากรีเลชัน show.dat มาแสดงผล นำทุเปิลมาแยกอ่านค่าความยาวของแอตทริบิว และแสดงผลทีละแอตทริบิวจนครบทั้งทุเปิล และอ่านทุเปิลถัดไปมาแสดงผลจนครบทุกทุเปิลในรีเลชัน พร้อมกับนับจำนวนทุเปิลด้วย
- แสดงจำนวนทุเปิลที่แสดงผล

3.5 การจัดการปทานุกรมข้อมูล

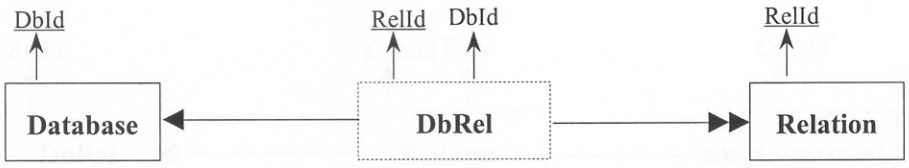
ในงานวิจัยนี้โปรแกรมที่พัฒนาใช้รูปแบบฐานข้อมูลเชิงสัมพันธ์ ดังนั้นรูปแบบข้อมูลที่ใช้กับปทานุกรมข้อมูลจะเป็นรูปแบบข้อมูลเชิงสัมพันธ์ด้วย ภาพประกอบที่ 3.29 เป็นแผนภาพโอ-อาร์แสดงปทานุกรมข้อมูลของระบบทั้งหมดที่ได้ออกแบบไว้ให้แทนเฉพาะข้อมูลที่จำเป็นต้องใช้ในงานวิจัยนี้เท่านั้น โดยแอตทริบิวที่ขีดเส้นใต้เส้นเดียว คือ คีย์หลัก และแอตทริบิวที่ขีดเส้นใต้สองเส้น คือ คีย์รอง ซึ่งสามารถดูรายละเอียดของสัญลักษณ์ต่างๆ ที่ใช้ในแผนภาพโอ-อาร์ได้ในภาคผนวก ค



ภาพประกอบที่ 3.29 ปทานุกรมข้อมูลของระบบ

1. **โครงสร้างของปทานุกรมข้อมูล**

จากแผนภาพปทานุกรมข้อมูลของระบบในภาพประกอบที่ 3.29 สามารถนำมาเขียนรายละเอียดตารางข้อมูลหลักและตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกัน ดังแสดงในภาพประกอบที่ 3.30 ถึงภาพประกอบที่ 3.34



ภาพประกอบที่ 3.30 ความสัมพันธ์ระหว่างฐานข้อมูลกับรีเลชัน

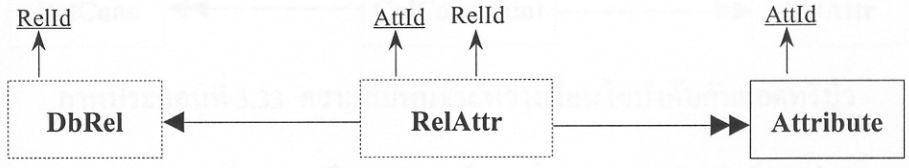
จากภาพประกอบที่ 3.30 เป็นแผนภาพโออาร์แสดงความสัมพันธ์ระหว่าง Database กับ Relation โดย

- ฐานข้อมูลหนึ่งๆ จะมีจำนวนรีเลชันได้หลายรีเลชัน
- แต่ละรีเลชันจะมีฐานข้อมูลใดฐานข้อมูลหนึ่งเป็นเจ้าของได้หนึ่งฐานข้อมูลเท่านั้น

เมื่อแปลงเป็นโครงสร้างข้อมูลจะได้ตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกันคือ

Database (DbId, DbName, DbCreate) แต่ละแถวแนวนอนของตารางข้อมูล Database แทนรายละเอียดข้อมูลของฐานข้อมูลหนึ่งฐานข้อมูล

Relation (RelId, DbId, RelName, NoAtt, TupSize, RelCreate, RelLstUpdate) แต่ละแถวแนวนอนของตารางข้อมูล Relation แทนรายละเอียดข้อมูลของรีเลชันหนึ่งรีเลชัน โดยมีข้อมูลของ DbRel จัดเก็บรวมกับข้อมูลของ Relation ด้วย



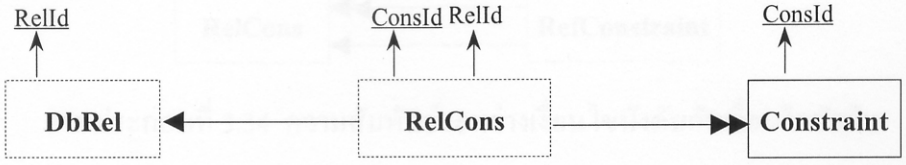
ภาพประกอบที่ 3.31 ความสัมพันธ์ระหว่างรีเลชันกับแอตทริบิว

จากภาพประกอบที่ 3.31 เป็นแผนภาพโออาร์แสดงความสัมพันธ์ระหว่าง DbRel กับ Attribute โดย

- รีเลชันหนึ่งๆ จะมีจำนวนแอตทริบิวได้หลายแอตทริบิว
- แต่ละแอตทริบิวจะมีรีเลชันใดรีเลชันหนึ่งเป็นเจ้าของได้หนึ่งรีเลชันเท่านั้น

เมื่อแปลงเป็นโครงสร้างข้อมูลจะได้ตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกันคือ

Attribute (AttId, RelId, AttName, AttDomain, AttPos, AttLen, AttPrecision, AttScale, AttNullable, AttDefaVal) แต่ละแถวแนวนอนของตารางข้อมูล Attribute แทนรายละเอียดข้อมูลของแอตทริบิวหนึ่งแอตทริบิว โดยมีข้อมูลของ RelAtt จัดเก็บรวมกับข้อมูลของ Attribute ด้วย



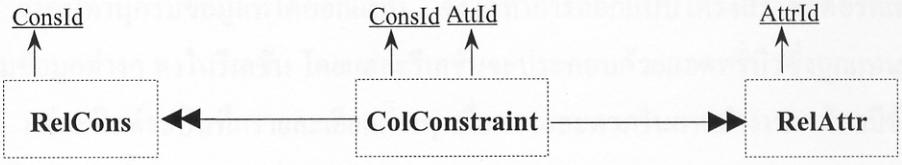
ภาพประกอบที่ 3.32 ความสัมพันธ์ระหว่างรีเลชันกับเงื่อนไขบังคับ

จากภาพประกอบที่ 3.32 เป็นแผนภาพโออาร์แสดงความสัมพันธ์ระหว่าง DbRel กับ Constraint โดย

- รีเลชันหนึ่งๆ จะมีการกำหนดเงื่อนไขบังคับได้หลายเงื่อนไขบังคับ
- แต่ละเงื่อนไขบังคับจะมีรีเลชันใดรีเลชันหนึ่งเป็นเจ้าของได้หนึ่งรีเลชันเท่านั้น

เมื่อแปลงเป็นโครงสร้างข้อมูลจะได้ตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกันคือ

Constraint (ConsId, RelId, ConsType) แต่ละแถวแนวนอนของตารางข้อมูล Constraint แทนรายละเอียดข้อมูลของการกำหนดเงื่อนไขบังคับหนึ่งเงื่อนไขบังคับ โดยมีข้อมูลของ RelCons จัดเก็บรวมกับข้อมูลของ Constraint ด้วย



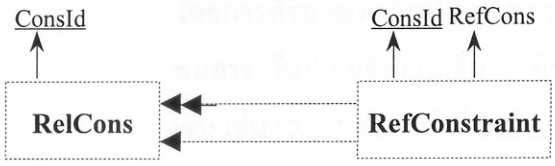
ภาพประกอบที่ 3.33 ความสัมพันธ์ระหว่างเงื่อนไขบังคับกับแอตทริบิว

จากภาพประกอบที่ 3.33 เป็นแผนภาพโออาร์แสดงความสัมพันธ์ระหว่าง RelCons กับ RelAttr โดย

- เงื่อนไขบังคับหนึ่งๆ สามารถมีแอตทริบิวที่เกี่ยวข้องกับเงื่อนไขบังคับนั้นได้มากกว่าหนึ่งแอตทริบิว
- แต่ละแอตทริบิวสามารถถูกกำหนดเงื่อนไขบังคับได้มากกว่าหนึ่งเงื่อนไขบังคับ

เมื่อแปลงเป็นโครงสร้างข้อมูลจะได้ตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกันคือ

ColConstraint (ConsId, AttId, AttPos) แต่ละแถวแนวนอนของตารางข้อมูล ColConstraint แทนรายละเอียดข้อมูลของการกำหนดเงื่อนไขบังคับของหนึ่งแอตทริบิว



ภาพประกอบที่ 3.34 ความสัมพันธ์ระหว่างเงื่อนไขบังคับกับเงื่อนไขบังคับ

จากภาพประกอบที่ 3.34 เป็นแผนภาพโออาร์แสดงความสัมพันธ์ระหว่าง RelCons กับ RelCons โดย

- เงื่อนไขบังคับหนึ่งๆ (กรณีเงื่อนไขบังคับการกำหนดคีย์นอก) จะอ้างอิงเงื่อนไขบังคับการกำหนดคีย์หลักได้เพียงหนึ่งเงื่อนไขบังคับเท่านั้น
- แต่ละเงื่อนไขบังคับ (กรณีเงื่อนไขบังคับการกำหนดคีย์หลัก) จะสามารถถูกอ้างอิงได้หลายครั้ง

เมื่อแปลงเป็นโครงสร้างข้อมูลจะได้ตารางข้อมูลเชิงสัมพันธ์ที่สมนัยกันคือ

RefConstraint (ConsId, RefCons, OnDelete) แต่ละแถวแนวนอนของตารางข้อมูล RefConstraint แทนรายละเอียดข้อมูลของเงื่อนไขบังคับการกำหนดคีย์นอกของหนึ่งเงื่อนไขบังคับ

2. รายละเอียดข้อมูลในปทานุกรมข้อมูล

จากปทานุกรมข้อมูลที่ได้ออกแบบ จึงได้ทำการออกแบบโครงสร้างเรคอร์ดเพื่อจัดเก็บปทานุกรมข้อมูลต่างๆ ลงในรีเลชัน โดยแต่ละรีเลชันจะประกอบด้วยแอตทริบิวซึ่งถูกแทนด้วยฟิลด์หนึ่งฟิลด์ แต่ละฟิลด์จะบันทึกรายละเอียดต่างๆ เพื่อความสะดวกในการทำงาน โดยมีข้อกำหนดของแต่ละรีเลชันดังนี้

- **ชื่อฟิลด์** ในรีเลชันหนึ่งๆ จะต้องมีชื่อฟิลด์ไม่ซ้ำกัน มีความยาวได้ไม่เกิน 15 ตัวอักขระ จะเป็นตัวอักษรภาษาอังกฤษ ตัวเลข หรือขีดเส้นใต้ แต่ต้องไม่มีช่องว่างและขึ้นต้นด้วยตัวอักษร ใช้ในการอ้างอิงถึงฟิลด์
- **รูปแบบข้อมูลของฟิลด์** การกำหนดชนิดข้อมูลของฟิลด์จะกำหนดเป็นค่าเป็นตัวอักษร ประกอบด้วย
  - C หมายถึง ข้อมูลอักขระ
  - I หมายถึง ข้อมูลเลขจำนวนเต็ม

- R หมายถึง ข้อมูลเลขทศนิยม
- D หมายถึง ข้อมูลวันที่ (date) มีรูปแบบ วัน/เดือน/ปี โดยปีจะต้องเป็นตัวเลข 4 ตัว และการจัดเก็บจะจัดเก็บเป็นเลขจำนวนเต็ม 4 ไบต์ โดยการคำนวณค่าวันที่เลขจำนวนเต็มที่จะใช้จัดเก็บได้จากสมการ วันที่เลขจำนวนเต็ม = ปี*10000 + เดือน*100 + วัน เช่น เช่น "2/2/2000" ค่าที่จัดเก็บ คือ 20000202
- 0 หมายถึง ข้อมูลหมายเลขเรคอร์ดในแฟ้มข้อมูล

- **ขนาดของฟิลด์** การกำหนดขนาดความยาวของข้อมูลที่จัดเก็บจริงในแต่ละฟิลด์ของ รีเลชัน จะมีหน่วยความยาวเป็นไบต์ โดยจะยาวได้ไม่เกิน 255 ไบต์

ตัวอย่างเช่น ชื่อฟิลด์ DbName รูปแบบข้อมูลของฟิลด์เป็น C ขนาดเท่ากับ 15 หมายถึง ฟิลด์ชื่อ DbName ในรีเลชันหนึ่งจัดเก็บข้อมูลในฟิลด์เป็นตัวอักษร ซึ่งมีขนาดยาวได้ไม่เกิน 15 ไบต์ ซึ่งรายละเอียดแต่ละรีเลชันของปทานุกรมข้อมูล มีดังนี้

Database

Database เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของฐานข้อมูลทั้งหมดที่มีในระบบ โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลหนึ่งฐานข้อมูล ตารางที่ 3.2 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล Database ซึ่งมีรหัสฐานข้อมูลเป็นคีย์หลัก และชื่อฐานข้อมูลเป็นคีย์รอง

ตารางที่ 3.2 รายละเอียดโครงสร้างปทานุกรมข้อมูล Database

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
DbId	P	0	8	รหัสฐานข้อมูล
DbName	S	C	15	ชื่อฐานข้อมูล
NoRel	N	I	2	จำนวนรีเลชันในฐานข้อมูล
DbCreate	N	D	4	วันที่สร้างฐานข้อมูล

Relation

Relation เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของทุกรีเลชันที่ถูกสร้างในระบบ โดยแต่ละรีเลชันในระบบต้องมีฐานข้อมูลใดฐานข้อมูลหนึ่งเป็นเจ้าของเสมอ ในฐานข้อมูลหนึ่งๆ จะมีจำนวนรีเลชันเท่าใดก็ได้ แต่ชื่อรีเลชันในฐานข้อมูลเดียวกันจะซ้ำกันไม่ได้ โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลหนึ่งรีเลชัน ตารางที่ 3.3 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล Relation ซึ่งมีรหัสรีเลชันเป็นคียหลัก

ตารางที่ 3.3 รายละเอียดโครงสร้างปทานุกรมข้อมูล Relation

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
RelId	P	0	8	รหัสรีเลชัน
DbId	N	0	8	รหัสฐานข้อมูล
RelName	N	C	15	ชื่อรีเลชัน
NoAtt	N	I	2	จำนวนแอตทริบิวที่มีในรีเลชัน
TupLen	N	I	2	ขนาดทุเปิล
RelCreate	N	D	4	วันที่สร้างรีเลชัน
RelLstUpdate	N	D	4	วันที่ปรับปรุงรีเลชันครั้งหลังสุด

Attribute

Attribute เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของทุกแอตทริบิวที่ถูกสร้างในระบบ โดยแต่ละแอตทริบิวในระบบต้องมีรีเลชันใดรีเลชันหนึ่งในฐานข้อมูลนั้นๆ เป็นเจ้าของเสมอ ในรีเลชันจะมีจำนวนแอตทริบิวได้มากกว่าหนึ่งแอตทริบิว แต่ชื่อแอตทริบิวในรีเลชันเดียวกันจะซ้ำกันไม่ได้ โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลหนึ่งแอตทริบิว ตารางที่ 3.4 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล Attribute ซึ่งมีรหัสแอตทริบิวเป็นคีย์หลัก

ตารางที่ 3.4 รายละเอียดโครงสร้างปทานุกรมข้อมูล Attribute

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
AttId	P	0	8	รหัสแอตทริบิว
RelId	N	0	8	รหัสรีเลชัน
AttName	N	C	15	ชื่อแอตทริบิว
AttDomain	N	C	1	ชนิดข้อมูลของค่าแอตทริบิว
AttPos	N	I	4	ตำแหน่งเริ่มต้นของข้อมูลในระเบียน
AttLen	N	I	4	ขนาดของข้อมูลในระเบียนซึ่งมีหน่วยเป็นไบต์
AttPrecison	N	I	1	จำนวนตัวเลขหน้าตำแหน่งทศนิยม กรณีโดเมนของแอตทริบิวเป็นชนิดตัวเลข (I หรือ R)
AttScale	N	I	1	จำนวนตัวเลขหลังตำแหน่งทศนิยม กรณีโดเมนของแอตทริบิวเป็น R
AttNullable	N	C	1	ค่าแฟลกการใส่ค่าว่าง
AttDefault	N	C	256	ค่าเริ่มต้นของข้อมูลในทุเปิล

Constraint

Constraint เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของการกำหนดเงื่อนไขบังคับได้แก่ การกำหนดคีย์หลัก คีย์รอง และคีย์นอก หรือเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวของแต่ละรีเลชันในฐานข้อมูลทั้งหมดในระบบ โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลหนึ่งเงื่อนไขบังคับ ตารางที่ 3.4 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล Constraint ซึ่งมีรหัสเงื่อนไขบังคับเป็นคีย์หลัก และค่าของประเภทของเงื่อนไขบังคับเก็บเป็นค่าอักขระประกอบด้วย

- P หมายถึง การกำหนดคีย์หลัก
- S หมายถึง การกำหนดคีย์รอง
- F หมายถึง การกำหนดคีย์นอก หรือเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว

ตารางที่ 3.5 รายละเอียดโครงสร้างปทานุกรมข้อมูล Constraint

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
ConsId	P	0	8	รหัสเงื่อนไขบังคับ
RelId	N	0	8	รหัสรีเลชัน
ConsType	N	C	1	ประเภทของเงื่อนไขบังคับ

ColConstraint

ColConstraint เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของการกำหนดเงื่อนไขบังคับของแอตทริบิวต่างๆ ในรีเลชันที่มีการกำหนดเงื่อนไขบังคับ โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลการกำหนดเงื่อนไขบังคับของหนึ่งแอตทริบิว ตารางที่ 3.6 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล ColConstraint ซึ่งมีรหัสเงื่อนไขบังคับเป็นคีย์หลัก

ตารางที่ 3.6 รายละเอียดโครงสร้างปทานุกรมข้อมูล ColConstraint

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
ConsId	P	0	8	รหัสเงื่อนไขบังคับ
AttId	P	0	8	รหัสแอตทริบิวที่มีการกำหนดเงื่อนไขบังคับ
AttPos	N	I	1	ตำแหน่งลำดับของแอตทริบิว โดยจะมีค่าเป็นศูนย์ถ้าแอตทริบิวที่สัมพันธ์กับเงื่อนไขบังคับนี้มีเพียงแอตทริบิวเดียว

RefConstraint

RefConstraint เป็นรีเลชันใช้แทนข้อมูลซึ่งเป็นรายละเอียดของการกำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวหรือการกำหนดคีย์นอก โดยแต่ละทุเปิลจะแทนรายละเอียดของข้อมูลการกำหนดคีย์นอกในหนึ่งเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว ตารางที่ 3.7 แสดงรายละเอียดโครงสร้างปทานุกรมข้อมูล RefConstraint ซึ่งมีรหัสเงื่อนไขบังคับเป็นคีย์หลัก

ตารางที่ 3.7 รายละเอียดโครงสร้างปทานุกรมข้อมูล RefConstraint

ชื่อแอตทริบิว	ชนิด	รูปแบบ	ขนาด	คำอธิบาย
ConsId	P	0	8	รหัสเงื่อนไขบังคับ
ConsRef	N	0	8	รหัสเงื่อนไขบังคับของการกำหนดคีย์หลักของรีเลชันที่ถูกอ้างถึง
OnDelete	N	C	1	ค่าแฟลกการลบทุเปิลในรีเลชันแบบต่อเนื่อง

3. ตัวอย่างปทานุกรมข้อมูลที่จัดเก็บในระบบ

จากรายละเอียดโครงสร้างปทานุกรมข้อมูลที่ออกแบบสามารถแสดงตัวอย่างการจัดเก็บข้อมูลที่เป็นไปได้ในรีเลชันต่างๆ ของปทานุกรมข้อมูลในระบบในขณะใดขณะหนึ่งได้ ดังตารางที่ 3.8 ถึงตารางที่ 3.13

ตารางที่ 3.8 ตัวอย่างข้อมูลของปทานุกรมข้อมูล Database

DbId	DbName	NoRel	DbCreate
7.0	PERSONAL	2	14/12/2000
7.1	RESEARCH	2	10/3/2001

ตารางที่ 3.9 ตัวอย่างข้อมูลของปทานุกรมข้อมูล Relation

RelId	DbId	RelName	NoAtt	TupLen	RelCreate	RelLstUpdate
9.0	7.0	DEPT	2	21	10/2/2001	10/2/2001
9.1	7.1	PERSON	5	27	2/3/2001	2/3/2001

ตารางที่ 3.10 ตัวอย่างข้อมูลของปทานุกรมข้อมูล Attribute

AttId	RelId	AttName	Att Domain	Att Pos	Att Len	Att Precision	Att Scale	Att Default	Att Nullable
12.0	9.0	ID	I	0	1	2	0	NULL	N
12.1	9.0	NAME	C	1	20	0	0	NULL	N
12.2	9.1	ID	I	0	4	5	0	NULL	N
12.3	9.1	NAME	C	4	20	0	0	NULL	N
12.4	9.1	AGE	I	24	1	2	0	NULL	Y
12.5	9.1	SEX	C	25	1	0	0	F	Y
12.6	9.1	DEPT	I	26	1	2	0	NULL	Y

ตารางที่ 3.11 ตัวอย่างข้อมูลของปทานุกรมข้อมูล Constraint

ConsId	RelId	ConsType
15.0	9.0	P
15.1	9.0	S
15.2	9.1	P
15.3	9.1	F

ตารางที่ 3.12 ตัวอย่างข้อมูลของปทานุกรมข้อมูล ColConstraint

ConsId	AttId	AttPos
15.0	12.0	0
15.1	12.1	0
15.2	12.2	0
15.3	12.6	0

ตารางที่ 3.13 ตัวอย่างข้อมูลของปทานุกรมข้อมูล RefConstraint

ConsId	RefCons	OnDelete
15.3	15.0	N

3.6 การพัฒนาระบบ

ในการพัฒนาระบบ ได้มีการกำหนดค่าคงที่และโครงสร้างข้อมูลเพื่อใช้เก็บข้อมูลที่ใช้อ้างถึงในระหว่างการทำงานของโปรแกรม ซึ่งการกำหนดค่าคงที่และโครงสร้างข้อมูลต่างๆ ในโปรแกรม อาจเขียนแตกต่างกันออกไปขึ้นอยู่ภาษาที่ใช้ในการพัฒนาโปรแกรม สำหรับระบบที่พัฒนานี้ได้กำหนดค่าคงที่และโครงสร้างข้อมูลตามลักษณะของภาษาซีที่ใช้ในการพัฒนาโปรแกรม

1. ค่าคงที่

ค่าคงที่เป็นค่าของตัวแปรต่างๆ ที่ใช้ร่วมกันในโปรแกรมสามารถแสดงได้ดังตารางที่ 3.14 ซึ่งการกำหนดค่าคงที่จะกำหนดให้กำตัวแปรเพื่อใช้ในโปรแกรมด้วยภาษาที่ใช้ในการพัฒนาโปรแกรม ตัวอย่างเช่น การกำหนดให้ตัวแปร BLOACSIZE มีค่าเท่ากับ 1024 เมื่อเขียนในโปรแกรมด้วยภาษาซีจะเขียนเป็น #define BLOCKSIZE 1024 เป็นต้น

ตารางที่ 3.14 การกำหนดค่าคงที่ต่างๆ ที่ใช้ในโปรแกรม

ชื่อตัวแปร	ค่าที่กำหนด	ชื่อตัวแปร
BLOACSIZE	1024	ขนาดบล็อกในแฟ้มข้อมูลที่สร้างขึ้นในระบบ
PAGESIZE	BLOACSIZE	ขนาดเพจของเนื้อที่ในหน่วยความจำที่จองไว้เป็นบัฟเฟอร์
MAXBLOCK	10000	จำนวนบล็อกในแฟ้มข้อมูล
MAXPAGE	1000	จำนวนเพจในบัฟเฟอร์
MAXNAME	50	ความยาวมากที่สุดของชื่อรีเลชันที่จัดเก็บในแฟ้มข้อมูล
TRUE	1	ค่าจริงที่ใช้ในการตรวจสอบเงื่อนไข
FALSE	0	ค่าเท็จที่ใช้ในการตรวจสอบเงื่อนไข
OK	1	สถานะการทำงานของโปรแกรมแสดงว่าทำงานสำเร็จตามคำสั่ง
DONE	2	สถานะการทำงานของโปรแกรมเป็นค่าที่ส่งกลับเพื่อแสดงว่าไม่สามารถทำงานสำเร็จตามคำสั่ง แต่ก็ไม่ได้พบข้อผิดพลาดใดๆ
HTSIZE	5	ค่าสำหรับใช้ในการคำนวณ hashing function เพื่อค้นหาตำแหน่งเพจในบัฟเฟอร์ที่จัดเก็บข้อมูลของบล็อกข้อมูลที่ต้องการเข้าถึง
INVALID_SLOT	-1	ค่าสำหรับตรวจสอบหมายเลข slot ที่ผิดพลาด
INVALID_BLOCK	-1	ค่าสำหรับตรวจสอบหมายเลขบล็อกที่ผิดพลาด

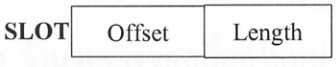
ตารางที่ 3.14 การกำหนดค่าคงที่ต่างๆ ที่ใช้ในโปรแกรม (ต่อ)

ชื่อตัวแปร	ค่าที่กำหนด	ชื่อตัวแปร
NAMELEN	15	ความยาวสูงสุดของชื่อฐานข้อมูล ชื่อรีเลชัน หรือชื่อแอตทริบิว
ATTLEN	256	ความยาวสูงสุดของค่าแอตทริบิวที่กำหนดในการสร้างรีเลชัน
COMMANDLEN	256	ความยาวของสายอักขระที่ใช้เก็บคำสั่งที่ผู้ใช้พิมพ์เข้าไป
LIBSYM	"/"	อักขระที่เป็นตัวคั่นระหว่างชื่อไดเรคทอรี
CURLIB	"."	อักขระที่ใช้ในการอ้างถึงไดเรคทอรีปัจจุบัน
DATALIB	"data"	ชื่อไดเรคทอรีที่ใช้เก็บข้อมูลทั้งหมดในระบบ
SYSFILE	"system.cat"	ชื่อแฟ้มข้อมูลสำหรับเก็บปทานุกรมของระบบ
DBNAME	"db.cat"	ชื่อรีเลชันสำหรับเก็บรายชื่อฐานข้อมูลในระบบ
RELNAME	"rel.cat"	ชื่อรีเลชันสำหรับเก็บรายชื่อรีเลชันในระบบ
ATTNAME	"att.cat"	ชื่อรีเลชันสำหรับเก็บรายชื่อแอตทริบิวในระบบ
CONSNAME	"cons.cat"	ชื่อรีเลชันสำหรับเก็บรายชื่อเงื่อนไขบังคับในระบบ
COLCONSNAME	"colcons.cat"	ชื่อรีเลชันสำหรับเก็บรายชื่อแอตทริบิวที่ถูกกำหนดเงื่อนไขบังคับ
REFCONSNAME	"refcons.cat"	ชื่อรีเลชันสำหรับเก็บรายละเอียดการกำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวในระบบ
RELINDEX	"rel.idx"	ชื่อรีเลชันสำหรับเก็บดัชนีของรายชื่อรีเลชันในระบบ
ATTINDEX	"att.idx"	ชื่อรีเลชันสำหรับเก็บดัชนีของรายแอตทริบิวในระบบ
PRIKEY	'P'	คีย์หลัก
SECKEY	'S'	คีย์รอง
FORKEY	'F'	คีย์นอก หรือเงื่อนไขบังคับอ้างอิงค่าแอตทริบิว
attCHAR	'C'	รูปแบบข้อมูลอักขระ
attINT	'I'	รูปแบบข้อมูลเลขจำนวนเต็ม
attREAL	'R'	รูปแบบข้อมูลเลขทศนิยม
attDATE	'D'	รูปแบบข้อมูลวันที่
attRID	'0'	รูปแบบข้อมูลหมายเลขเรคอร์ดในแฟ้มข้อมูล
NOT_NULL_FLAG	1	ค่าแฟลกสำหรับการกำหนด NOT NULL ให้กับแอตทริบิว
PRI_KEY_FLAG	8	ค่าแฟลกสำหรับการกำหนดคีย์หลักให้กับแอตทริบิว
SEC_KEY_FLAG	16	ค่าแฟลกสำหรับการกำหนดคีย์รองให้กับแอตทริบิว

2. โครงสร้างข้อมูล

โครงสร้างข้อมูลเป็นกำหนดการจัดเก็บข้อมูลที่ใช้ในการพัฒนาโปรแกรม ซึ่งเขียนขึ้นตามข้อกำหนดและลักษณะของภาษาซีที่ใช้ในการพัฒนาโปรแกรม เช่น โครงสร้างข้อมูลที่เก็บตำแหน่งและความยาวของแต่ละเรคอร์ด (SLOT) ประกอบด้วย Offset สำหรับเก็บตำแหน่งของเรคอร์ด และ Length สำหรับเก็บความยาวของเรคอร์ด ดังแสดงในภาพประกอบที่ 3.35 เมื่อเขียนในโปรแกรมด้วยภาษาซีจะเขียนเป็นโครงสร้างข้อมูลดังนี้

```
typedef struct slot{
        short   Offset;     /* offset of record from start of data area */
        short   Length;     /* equals -1 if slot is not in use*/
}SLOT;
```

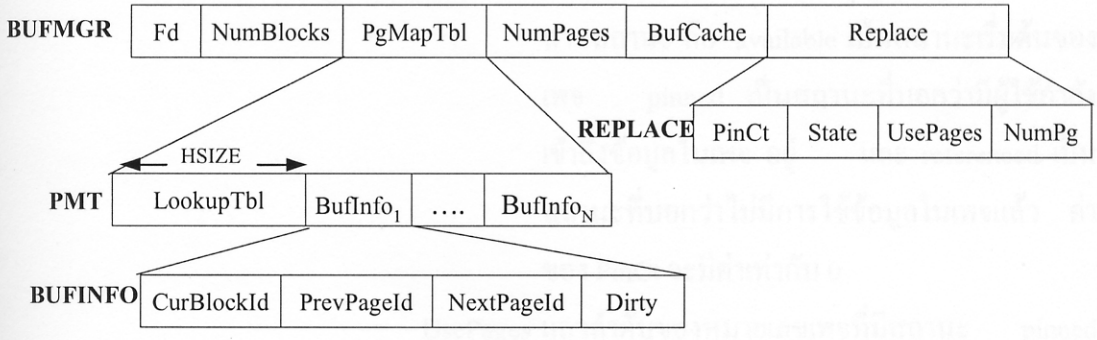


ภาพประกอบที่ 3.35 โครงสร้างข้อมูล SLOT

ในระบบที่พัฒนาประกอบด้วยโครงสร้างข้อมูลต่อไปนี้

โครงสร้างข้อมูลเกี่ยวกับบัฟเฟอร์

โครงสร้างข้อมูลสำหรับจัดการการใช้งานบัฟเฟอร์ที่จองไว้ในหน่วยความจำหลัก แสดงดังภาพประกอบที่ 3.36 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.36 โครงสร้างข้อมูลเกี่ยวกับบัฟเฟอร์

Fd คือ รหัสแฟ้มข้อมูล (file descriptor) ที่เป็นเจ้าของข้อมูลในบัฟเฟอร์

NumBlocks คือ จำนวนบล็อกมากที่สุดที่มีได้ในแฟ้มข้อมูล

PgMapTbl คือ โครงสร้างข้อมูลที่เก็บรายละเอียดการใช้บัฟเฟอร์ ประกอบด้วย

- LookupTbl ใช้ในการค้นหาเพจเป็นแถวลำดับของตำแหน่งเพจบนบัฟเฟอร์ที่จัดเก็บบล็อกข้อมูลในแฟ้มข้อมูล มีขนาดเท่ากับ HSIZE
- $BufInfo_1$ โครงสร้างข้อมูลที่เก็บข้อมูลต่อไปนี้
  - CurBlockId หมายเลขบล็อกของข้อมูลในเพจนี้
  - PrevPageId หมายเลขเพจก่อนหน้า
  - NextPageId หมายเลขเพจถัดไป
  - Dirty สถานะการเปลี่ยนแปลงข้อมูลในเพจ มีได้สองค่า คือ 0 หมายถึงข้อมูลในเพจมีการเปลี่ยนแปลง และ 1 หมายถึงข้อมูลในเพจไม่มีการเปลี่ยนแปลง

NumPages คือ จำนวนเพจของเนื้อที่ในหน่วยความจำที่จองไว้เป็นในบัฟเฟอร์

BufCache คือ เนื้อที่บนหน่วยความจำที่จองไว้จัดเก็บข้อมูลจากแฟ้มข้อมูล แบ่งออกเป็นเพจ มีจำนวนเพจทั้งหมดเท่ากับ NumPages แต่ละเพจมีขนาดเท่ากับขนาดบล็อกในแฟ้มข้อมูล

Replace คือ โครงสร้างข้อมูลที่เก็บรายละเอียดการสำหรับใช้ในการแทนที่เพจเมื่อบัฟเฟอร์เต็ม ประกอบด้วย

- PinCt นับจำนวนการขอใช้งานข้อมูลของแต่ละเพจ
- State เก็บสถานะการใช้งานของแต่ละเพจในบัฟเฟอร์ มีสามสถานะ คือ available เป็นสถานะเริ่มต้นของเพจ pinned เป็นสถานะที่บอกว่ามีผู้ใช้กำลังเข้าถึงข้อมูลในเพจ อยู่ และ referenced เป็นสถานะที่บอกว่าไม่มีการใช้ข้อมูลในเพจแล้ว ค่าของ PinCt จะมีค่าเท่ากับ 0
- UsePages แถวลำดับของหมายเลขเพจที่มีสถานะ pinned นั่นคือจะเก็บหมายเลขเพจที่มีผู้ใช้ขอใช้ข้อมูลในเพจไว้ ซึ่งหมายเลขเพจที่มีผู้ใช้ขอใช้ข้อมูลในเพจล่าสุดจะถูกเก็บไว้ท้ายสุดของแถวลำดับ ดังนั้นเมื่อมีการแทนที่เพจด้วยวิธี LRU จะเลือกหมายเลขเพจที่จัดเก็บอยู่ในลำดับต้นๆ ก่อน

NumPg คือ จำนวนเพจที่จัดเก็บใน UsePages สถานะเริ่มต้นจะเป็น 0 เนื่องจากยังไม่มีเพจที่มีผู้ใช้ขอใช้ข้อมูลในเพจ

โครงสร้างข้อมูลเกี่ยวกับแฟ้มข้อมูล

จากการออกแบบการจัดเก็บข้อมูลในแฟ้มข้อมูล จึงได้ทำการกำหนดโครงสร้างข้อมูลเพื่อความสะดวกการอ้างถึงแฟ้มข้อมูล แสดงดังภาพประกอบที่ 3.7 ซึ่งมีรายละเอียดดังนี้

ภาพประกอบที่ 3.37 โครงสร้างข้อมูลเกี่ยวกับแฟ้มข้อมูล

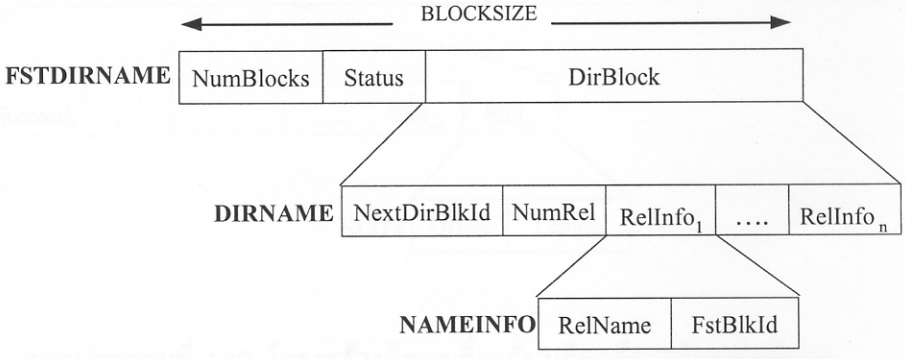
Fd คือ รหัสแฟ้มข้อมูลที่ใช้งาน

FileName คือ ชื่อแฟ้มข้อมูลที่ใช้งาน

BufMgr คือ ตัวชี้ไปยังโครงสร้างข้อมูล BUFMGR ที่จัดเก็บข้อมูลเกี่ยวกับการจัดการบัฟเฟอร์ของการใช้แฟ้มข้อมูลนี้

โครงสร้างข้อมูลเกี่ยวกับรายชื่อรีเลชันในแฟ้มข้อมูล

การจัดเก็บข้อมูลส่วนรายชื่อรีเลชันในแฟ้มข้อมูล มีลักษณะการจัดเก็บ 2 แบบคือ บล็อกแรกเป็นโครงสร้างข้อมูล FSTDIRNAME สำหรับจัดเก็บจำนวนบล็อกทั้งหมดในแฟ้มข้อมูล สถานะการเปิดใช้งานแฟ้มข้อมูล และรายชื่อรีเลชันในแฟ้มข้อมูล ส่วนบล็อกถัดไปสำหรับจัดเก็บเฉพาะส่วนของโครงสร้างข้อมูล DIRNAME ซึ่งเก็บรายชื่อรีเลชันในแฟ้มข้อมูล แต่ละแบบจะจัดเก็บในบล็อกด้วยขนาดเดียวกับขนาดบล็อกของแฟ้มข้อมูล แสดงดังภาพประกอบที่ 3.38 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.38 โครงสร้างข้อมูลเกี่ยวกับรายชื่อรีเลชันในแฟ้มข้อมูล

NumBlocks คือ จำนวนบล็อกทั้งหมดที่สามารถมีได้ในแฟ้มข้อมูล

Status คือ สถานะการเปิดใช้งานแฟ้มข้อมูล ถ้าเปิดใช้งานจะกำหนดเป็น 0 และเมื่อปิดแฟ้มข้อมูลจะเปลี่ยนกลับเป็น 1 ดังนั้นถ้ามีการออกจากระบบโดยไม่ปิดแฟ้มข้อมูลรหัสจะยังคงเป็น 0 จะไม่สามารถเปิดใช้แฟ้มข้อมูลนี้ในครั้งถัดไปได้

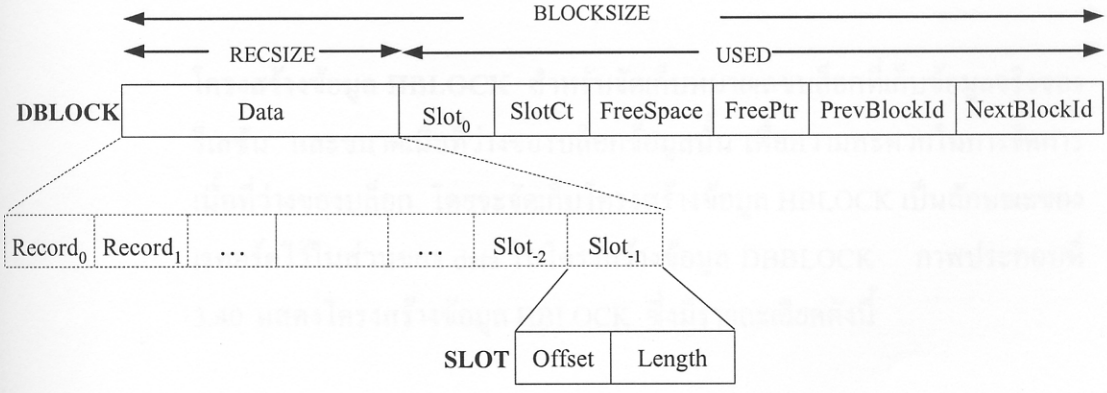
DirBlock คือ โครงสร้างข้อมูลที่เก็บรายชื่อรีเลชันในแฟ้มข้อมูล ประกอบด้วย

- NextDirBlkId หมายเลขบล็อกถัดไปที่เก็บรายชื่อรีเลชัน
- NumRel จำนวนชื่อรีเลชันที่จัดเก็บในบล็อกนี้
- RelInfo โครงสร้างข้อมูลที่เก็บข้อมูลต่อไปนี้
  - RelName ชื่อรีเลชัน
  - FstBlockId หมายเลขบล็อกแรกของข้อมูลในรีเลชัน

โครงสร้างข้อมูลเกี่ยวกับข้อมูลในแฟ้มข้อมูล

จากการออกแบบการจัดเก็บข้อมูลของฐานข้อมูลที่ผู้ใช้สร้างขึ้น ซึ่งเก็บเป็นรีเลชันไว้ในส่วนของ data space ในแฟ้มข้อมูล จึงได้ทำการกำหนดโครงสร้างข้อมูลเพื่อความสะดวกในการเข้าถึงข้อมูลของแต่ละรีเลชัน จัดเก็บเป็นสองส่วนดังนี้

- **โครงสร้างข้อมูล DBBLOCK** สำหรับเก็บข้อมูลจริงของรีเลชัน โดยจะเก็บด้วยขนาดเดียวกับขนาดบล็อกของแฟ้มข้อมูล แสดงดังภาพประกอบที่ 3.39 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.39 โครงสร้างข้อมูลเกี่ยวกับบล็อกข้อมูลในแฟ้มข้อมูล

Data คือ เก็บข้อมูล 2 แบบ ได้แก่ เรคอร์ดข้อมูลของรีเลชันเรียงต่อกันจากซ้ายไปขวา และโครงสร้างข้อมูล SLOT เก็บตำแหน่งและความยาวของเรคอร์ดในส่วน Data ส่วนของ Data ในบล็อกจะมีความยาวคงที่เท่ากับ RECSIZE ดังนั้นจำนวนเรคอร์ดและโครงสร้างข้อมูล SLOT ที่จัดเก็บใน Data จะขึ้นอยู่กับความยาวของเรคอร์ดที่จัดเก็บ การเข้าถึงข้อมูลในส่วน Data ในแต่ละบล็อกจะเข้าถึงโดยใช้ข้อมูลที่จัดเก็บในแต่ละ SLOT เมื่อมีการลบเรคอร์ดใด ๆ จะย้ายข้อมูลเรคอร์ดทางขวามาชิดกันเพื่อให้เนื้อที่ว่างในส่วน Data อยู่ติดกัน โดยจะปรับปรุงแก้ไขค่า offset ใน SLOT ให้ถูกต้องด้วย

$Slot_0$ คือ โครงสร้างข้อมูลที่เก็บตำแหน่งและความยาวของแต่ละเรคอร์ด มีรายละเอียดดังนี้

- Offset ตำแหน่งของเรคอร์ดที่จัดเก็บอยู่ใน Data
- Length ความยาวของเรคอร์ดที่จัดเก็บอยู่ใน Data ถ้า slot ว่างจะมีค่าเป็น -1

SlotCt คือ จำนวน SLOT ที่ใช้

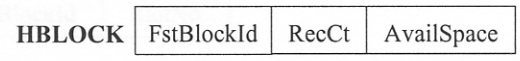
FreeSpace คือ จำนวนเนื้อที่ว่างของบล็อกในส่วน Data

FreePtr คือ ตำแหน่งที่ว่างของบล็อกในส่วน Data

PrevBlockId คือ หมายเลขบล็อกก่อนหน้า

NextBlockId คือ หมายเลขบล็อกถัดไป

- **โครงสร้างข้อมูล HBLOCK** สำหรับจัดเก็บหมายเลขบล็อกที่เก็บข้อมูลจริงของรีเลชัน และขนาดเนื้อที่ว่างของบล็อกข้อมูลนั้น เพื่อความสะดวกในการจัดการเนื้อที่ว่างของบล็อก โดยจะจัดเก็บโครงสร้างข้อมูล HBLOCK เป็นลักษณะของเรคอร์ดไว้ในส่วนของ data ในโครงสร้างข้อมูล DBBLOCK ภาพประกอบที่ 3.40 แสดงโครงสร้างข้อมูล HBLOCK ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.40 โครงสร้างข้อมูล HBLOCK

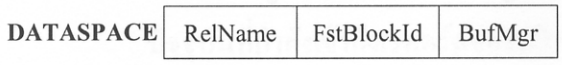
FstBlockId คือ หมายเลขบล็อกที่เก็บข้อมูลจริงของรีเลชัน

RecCt คือ จำนวนเรคอร์ดที่จัดเก็บในหมายเลขบล็อก FstBlockId เพื่อใช้ในการนับจำนวนเรคอร์ดของรีเลชัน

AvailSpace คือ จำนวนเนื้อที่ว่างของบล็อกในส่วน Data ของหมายเลขบล็อก FstBlockId

โครงสร้างข้อมูลเกี่ยวกับการอ้างถึงรีเลชัน จะจัดเก็บเป็น 2 ลักษณะคือ

- **โครงสร้างข้อมูล DATASPACE** จะถูกใช้เมื่อมีการเข้าถึงข้อมูลของรีเลชันในแฟ้มข้อมูลโดยการระบุหมายเลขเรคอร์ด เช่น การลบหรือแก้ไขข้อมูลในรีเลชัน หรือเข้าถึงข้อมูลในรีเลชันเมื่อระบุหมายเลขเรคอร์ดที่ต้องการ ภาพประกอบที่ 3.41 แสดงโครงสร้างข้อมูล DATASPACE ซึ่งมีรายละเอียดดังนี้



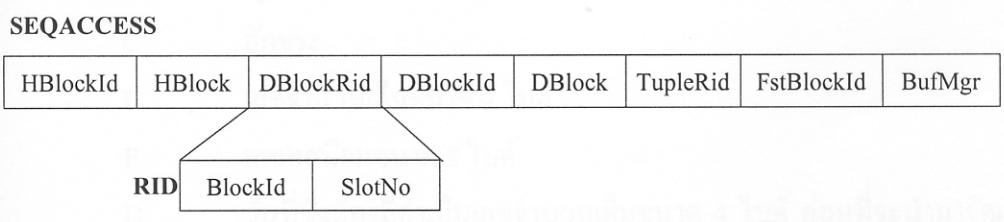
ภาพประกอบที่ 3.41 โครงสร้างข้อมูล DATASPACE

RelName คือ *ชื่อรีเลชัน*

FstBlockId คือ หมายเลขบล็อกแรกในแฟ้มข้อมูลเพื่อเข้าข้อมูลของรีเลชัน

BufMgr คือ ตัวชี้ไปยังโครงสร้างข้อมูล BUFMGR ที่จัดเก็บข้อมูลเกี่ยวกับการจัดการบัฟเฟอร์

- **โครงสร้างข้อมูล SEQACCESS** จะถูกใช้เมื่อมีการเข้าถึงข้อมูลของรีเลชันในแฟ้มข้อมูลทุกเรคอร์ดตามลำดับ ภาพประกอบที่ 3.42 แสดงโครงสร้างข้อมูล SEQACCESS ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.42 โครงสร้างข้อมูล SEQACCESS

HBlock คือ บล็อกข้อมูลของโครงสร้างข้อมูล HBLOCK บล็อกปัจจุบันที่กำลังเข้าถึงเรคอร์ดของบล็อก

HBlockId คือ หมายเลขบล็อกของ HBlock

DBlockRid คือ หมายเลขเรคอร์ดในส่วน data ของบล็อกข้อมูล HBlock ที่เก็บหมายเลขบล็อก (DblockId) ของ DBlock ปัจจุบัน และใช้เป็นข้อมูลในการเข้าถึงเรคอร์ดถัดไป สำหรับการค้นหา DBlock ถัดไป โดยหมายเลขเรคอร์ดประกอบด้วย

- BlockId หมายเลขบล็อกที่จัดเก็บเรคอร์ด
- SlotNo หมายเลข slot ที่เก็บค่าความยาว และตำแหน่งของเรคอร์ดที่จัดเก็บในส่วน data ของบล็อก

DBlock คือ บล็อกข้อมูลของโครงสร้างข้อมูล DBBLOCK บล็อกปัจจุบันที่กำลังเข้าถึงเรคอร์ดของบล็อก

DBlockId คือ หมายเลขบล็อกของ DBlock

TupleRid คือ หมายเลขเรคอร์ดในส่วน data ของบล็อกข้อมูล DBlock ที่เก็บเรคอร์ดของรีเลชันที่กำลังเข้าถึง และใช้เป็นข้อมูลในการเข้าถึงเรคอร์ดถัดไปของรีเลชัน

BufMgr คือ ตัวชี้ไปยังโครงสร้างข้อมูล BUFMGR ที่จัดเก็บข้อมูลเกี่ยวกับการจัดการบัฟเฟอร์

โครงสร้างข้อมูลเกี่ยวกับข้อมูลดัชนี

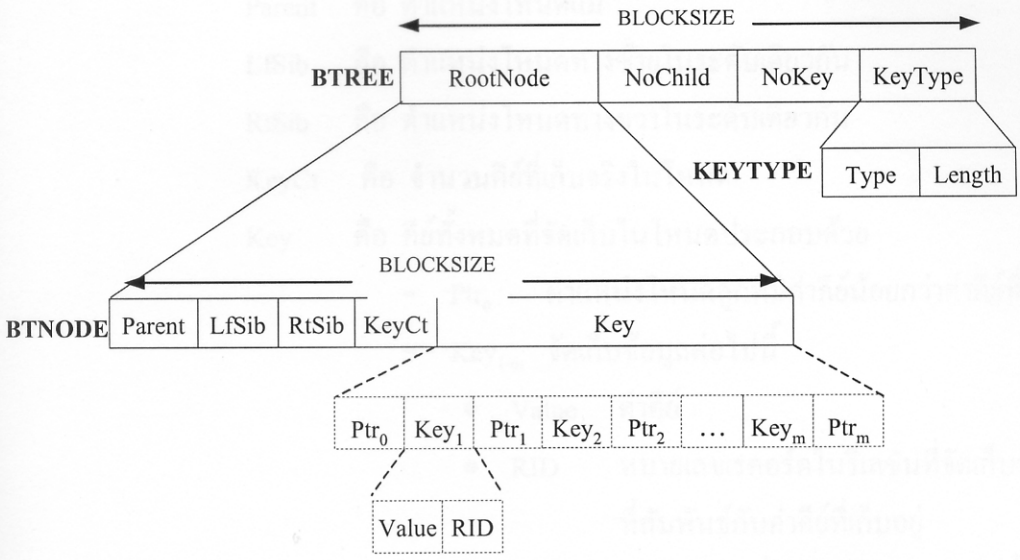
โครงสร้างข้อมูล B_m-tree ที่ออกแบบไว้เพื่อจัดเก็บข้อมูลดัชนีในโปรแกรมนี้ ได้คำนึงถึงความสะดวกในการใช้งาน โดยสามารถรองรับค่าคีย์เดียว หรือหลายคีย์ร่วมกันได้ สำหรับจำนวนโหนดลูกและจำนวนคีย์ร่วมขึ้นอยู่กับขนาดความยาวของแต่ละคีย์ และขนาดของเพจที่กำหนดในระบบ โดยกำหนดชนิดข้อมูลของค่าคีย์ที่จัดเก็บเป็นอักขระ แบ่งออกเป็นสี่ประเภทดังนี้

C อักขระ

I เลขจำนวนเต็มขนาด 4 ไบต์

F เลขทศนิยมขนาด 8 ไบต์

D วันที่ซึ่งต้องมีค่าเป็นเลขจำนวนเต็มขนาด 4 ไบต์ ก่อนที่จะนำมาจัดเก็บเป็นค่าคีย์



ภาพประกอบที่ 3.43 โครงสร้างข้อมูลเกี่ยวกับข้อมูลดัชนี

จากภาพประกอบที่ 3.43 แสดงโครงสร้างข้อมูลเกี่ยวกับข้อมูลดัชนี ซึ่งประกอบด้วยโครงสร้างข้อมูลสองส่วนคือ

- **โครงสร้างข้อมูล BTREE** เป็นโหนดสำหรับเก็บรายละเอียดเกี่ยวกับข้อมูลดัชนี ในการกำหนดคีย์ของรีเลชันแต่ละครั้งจะมีการสร้างโหนดประเภทนี้เพียงโหนดเดียว โดยจัดเก็บด้วยขนาดเท่ากับขนาดบล็อกในแฟ้มข้อมูล มีรายละเอียดดังนี้

  RootNode คือ ตำแหน่งโหนดราก ซึ่งเป็นโหนดแรกที่เก็บค่าคีย์

  NoChild คือ จำนวนโหนดลูก โดยที่แต่ละโหนดจะมีโหนดลูกมากที่สุดไม่เกิน NoChild โหนด และเก็บคีย์มากสุดไม่เกิน NoChild – 1 ค่า

  NoKey คือ จำนวนคีย์ที่ใช้รวมกันเป็นคีย์ร่วม

  KeyType คือ ชนิดและความยาวของแต่ละค่าคีย์ที่รวมกันเป็นคีย์

  - Type ชนิดของคีย์
  - Length ความยาวของคีย์

- **โครงสร้างข้อมูล BTNODE** เป็นโหนดสำหรับเก็บค่าคีย์ ตำแหน่งเรคอร์ดในรีเลชันที่มีค่าคีย์สัมพันธ์กัน และรายละเอียดที่เกี่ยวข้อง มีรายละเอียดดังนี้

Parent คือ ตำแหน่งโหนดแม่

LfSib คือ ตำแหน่งโหนดทางซ้ายในระดับเดียวกัน

RtSib คือ ตำแหน่งโหนดทางขวาในระดับเดียวกัน

KeyCt คือ จำนวนคีย์ที่เก็บจริงในโหนด

Key คือ คีย์ทั้งหมดที่จัดเก็บในโหนดประกอบด้วย

- Ptr_0 ตำแหน่งโหนดลูกที่มีค่าคีย์น้อยกว่าค่าคีย์ที่เก็บอยู่
- Key_{1-m} จัดเก็บข้อมูลต่อไปนี้
  - Value ค่าคีย์
  - RID หมายเลขเรคอร์ดในรีเลชันที่จัดเก็บข้อมูลที่สัมพันธ์กับค่าคีย์ที่เก็บอยู่
- Ptr_{1-m} ตำแหน่งโหนดลูกที่มีค่าคีย์มากกว่าค่าคีย์ที่เก็บอยู่

เนื่องจากแต่ละโหนดจะจัดเก็บในบล็อกข้อมูลหนึ่งบล็อกของแฟ้มข้อมูลตามที่ออกแบบไว้ ดังนั้นค่าของจำนวนคีย์ร่วม หรือ NoKey จะขึ้นความยาวคีย์แต่ละตัว โดยที่ความยาวของคีย์ร่วมทุกตัวรวมกับความยาวในการจัดเก็บข้อมูลใน RootNode NoChild และ NoKey รวมกันจะต้องไม่เกินขนาดบล็อกของแฟ้มข้อมูล และในทำนองเดียวกันค่าของจำนวนโหนดลูก หรือ NoChild จะขึ้นอยู่กับจำนวนคีย์ร่วมและความยาวของคีย์แต่ละตัวเช่นเดียวกัน ซึ่งสามารถหาค่าของ NoChild ได้ดังนี้

$$\text{NoKey} = \frac{(\text{BLOCKSIZE} - \text{ขนาดในการจัดเก็บข้อมูลของ (Parent+ LtSib+RtSib + KeyCt))}}{\text{NoKey x ขนาดในการจัดเก็บข้อมูล (Value+RID)}}$$

โครงสร้างข้อมูลเกี่ยวกับการติดต่อระหว่างผู้ใช้กับระบบ

จากการออกแบบเกี่ยวกับการติดต่อระหว่างผู้ใช้กับระบบ จึงได้กำหนดโครงสร้างข้อมูลดังต่อไปนี้

- **โครงสร้างข้อมูล LEX** จะเก็บข้อมูลที่ใช้ในการตรวจสอบคำสั่งและเก็บข้อมูลที่รับจากผู้ใช้ แสดงได้ดังภาพประกอบที่ 3.44 ซึ่งมีรายละเอียดดังนี้

LEX	sqlstr	ptr	tokstart	sqlend	command	strname	domain	length	decimal	defaval	flag	Pkey	Skey

...	Latt	Lpkey	Lskey	Lfkey	Lref	Lexpr	Lattname	Litem	ListUpd

ภาพประกอบที่ 3.44 โครงสร้างข้อมูล LEX

sqlstr คือ สายอักขระของประโยคคำสั่งที่รับจากผู้ใช้

ptr คือ ตำแหน่งอักขระในสายอักขระ sqlstr ที่กำลังตรวจสอบ

tokstart คือ ตำแหน่งอักขระเริ่มต้นของโทเคน

sqlend คือ ตำแหน่งอักขระสุดท้ายในสายอักขระ sqlstr

command คือ รหัสคำสั่งที่แปลได้จากโปรแกรม yyparse

strname คือ ชื่อฐานข้อมูล รีเลชันที่ผู้ใช้อ้างถึงในคำสั่ง

domain คือ ชนิดข้อมูลของค่าแอตทริบิว

length คือ ความยาวของค่าแอตทริบิว

decimal คือ จำนวนตำแหน่งหลังทศนิยมของค่าแอตทริบิวชนิดตัวเลข

defaval คือ ค่าสำหรับการกำหนดค่าโดยปริยายของแอตทริบิว

flag คือ ค่าแฟลกที่กำหนดให้แอตทริบิว

Pkey คือ ค่าแฟลกของการกำหนดคีย์หลัก

Skey คือ ค่าแฟลกของการกำหนดคีย์รอง

Latt คือ ลิสต์ของแอตทริบิว

Lpkey คือ ลิสต์ของชื่อแอตทริบิวที่กำหนดเป็นคีย์หลัก

Lskey คือ ลิสต์ของชื่อแอตทริบิวที่กำหนดเป็นคีย์รอง

Lfkey คือ ลิสต์ของชื่อแอตทริบิวที่กำหนดเป็นคีย์นอก

Lref คือ ลิสต์ของเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว

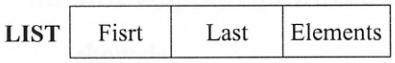
Lexpr คือ ลิสต์ของการตรวจสอบค่าแอตทริบิว

Lattname คือ ลิสต์ของชื่อแอตทริบิว

Litem คือ ลิสต์ของแอตทริบิว

Lupd คือ ลิสต์ของชื่อแอตทริบิวและลิสต์ค่าของแอตทริบิว ในคำสั่งแก้ไขปรับปรุงค่าของรีเลชัน

- **โครงสร้างข้อมูล LIST** จะเก็บลิสต์ของโครงสร้างข้อมูลประเภทต่างๆ ที่ใช้ในโปรแกรม แสดงได้ดังภาพประกอบที่ 3.45 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.45 โครงสร้างข้อมูล LIST

First คือ ตำแหน่งโหนดสุดแรกในลิสต์

Last คือ ตำแหน่งโหนดสุดท้ายในลิสต์

Elements คือ จำนวนโหนดทั้งหมดในลิสต์

- **โครงสร้างข้อมูล PRINTCOLINFO** จะเก็บรายชื่อแอตทริบิวที่จะแสดงเป็นส่วนหัวของตาราง แสดงดังภาพประกอบที่ 3.46 ซึ่งมีรายละเอียดดังนี้

PRINTCOLINFO	AttName	AttDomain	MaxLen

ภาพประกอบที่ 3.46 โครงสร้างข้อมูล PRINTCOLINFO

AttName คือ ชื่อแอตทริบิว

AttDomain คือ ชนิดข้อมูลของค่าแอตทริบิว

MaxLen คือ ขนาดความยาวสูงสุดของค่าแอตทริบิวในทุกทุเปิลของรีเลชัน

- **โครงสร้างข้อมูล DATABUFF** จะเก็บรายละเอียดเกี่ยวกับข้อมูลที่จะแสดงผลในรูปของตาราง แสดงดังภาพประกอบที่ 3.47 ซึ่งมีรายละเอียดดังนี้

DATABUFF	Col	ColCt	RowCt	DataRow	DataCol	Buffer	Pos	End	MaxLen

ภาพประกอบที่ 3.47 โครงสร้างข้อมูล DATABUFF

Col คือ ลิสต์รายชื่อแอตทริบิวที่จะแสดงเป็นส่วนหัวของตาราง

ColCt คือ จำนวนแอตทริบิว

RowCt คือ จำนวนทุเปิล

DataRow คือ ตำแหน่งข้อมูลของทุเปิลปัจจุบัน

DataCol คือ ตำแหน่งข้อมูลของแอตทริบิวปัจจุบัน

Buffer คือ ที่พักข้อมูลของทุเปิลที่จะอ่านหรือบันทึกในรีเลชัน show.dat

Pos คือ ตำแหน่งข้อมูลใน buffer ปัจจุบัน

End คือ ตำแหน่งข้อมูลสุดท้ายใน buffer

MaxLen คือ ขนาดความยาวในการจัดเก็บข้อมูลสูงสุดใน buffer

โครงสร้างข้อมูลเกี่ยวกับการจัดเก็บข้อมูลและค้นคืนข้อมูล

ในระบบได้ทำการกำหนดโครงสร้างข้อมูลสำหรับจัดเก็บปทานุกรมข้อมูล ข้อมูลของฐานข้อมูลที่ผู้ใช้สร้างขึ้น พร้อมทั้งข้อมูลต่างๆ ที่ใช้ในระหว่างการทำงานของโปรแกรม เพื่อความสะดวกในการพัฒนาโปรแกรม ประกอบด้วย

- **โครงสร้างข้อมูล DBNODE** เก็บรายละเอียดเกี่ยวกับฐานข้อมูล แสดงดังภาพประกอบที่ 3.48 ซึ่งมีรายละเอียดดังนี้

DBNODE	DbName	NoRel	DbCreate	DataFile	DbRid

ภาพประกอบที่ 3.48 โครงสร้างข้อมูล DBNODE

DbName คือ ชื่อฐานข้อมูล

NoRel คือ จำนวนรีเลชันในฐานข้อมูล

DbCreate คือ วันที่สร้างฐานข้อมูล

DataFile คือ ชื่อแฟ้มข้อมูลที่จัดเก็บข้อมูลของรีเลชันต่าง ๆ ในฐานข้อมูล

DbRid คือ หมายเลขเรคอร์ดที่เก็บรายละเอียดข้อมูลของฐานข้อมูลนี้

- **โครงสร้างข้อมูล RELNODE** เก็บรายละเอียดเกี่ยวกับรีเลชัน แสดงดังภาพประกอบที่ 3.49 ซึ่งมีรายละเอียดดังนี้

RELNODE	RelName	NoAtt	NoTup	TupSize	RelCreate	RelLstUpd	RelRid	DbNode	ListAtt	Next

ภาพประกอบที่ 3.49 โครงสร้างข้อมูล RELNODE

RelName คือ ชื่อรีเลชัน

NoAtt คือ จำนวนแอตทริบิวในรีเลชัน

NoTup คือ จำนวนทุเปิลในรีเลชัน

TupSize คือ ขนาดความยาวของแต่ละทุเปิลในรีเลชัน

RelCreate คือ วันที่สร้างรีเลชัน

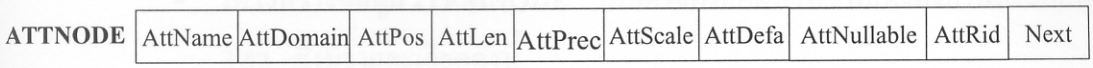
RelRid คือ หมายเลขเรคอร์ดที่จัดเก็บรายละเอียดข้อมูลรีเลชันนี้

DbNode คือ รายละเอียดของฐานข้อมูลที่เป็นเจ้าของรีเลชันนี้

ListAtt คือ ลิสต์รายชื่อแอตทริบิวของรีเลชันนี้

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล RELNODE ถัดไปในลิสต์

- **โครงสร้างข้อมูล ATTNODE** เก็บรายละเอียดเกี่ยวกับแอตทริบิว แสดงดังภาพประกอบที่ 3.50 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.50 โครงสร้างข้อมูล ATTNODE

AttName คือ ชื่อแอตทริบิว

AttDomain คือ ชนิดข้อมูลของค่าแอตทริบิว

AttPos คือ ตำแหน่งการจัดเก็บค่าแอตทริบิวในทุเปิล

AttLen คือ ความยาวค่าแอตทริบิวในทุเปิลมีหน่วยเป็นไบต์

AttPrec คือ จำนวนตำแหน่งของเลขหน้าจุดทศนิยม กรณีชนิดข้อมูลของค่าแอตทริบิวเป็นข้อมูลตัวเลข

AttScale คือ จำนวนตำแหน่งของเลขหลังจุดทศนิยม กรณีชนิดข้อมูลของค่าแอตทริบิวเป็นข้อมูลตัวเลข

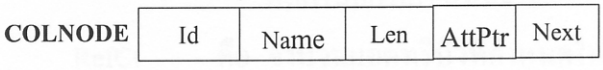
AttDefa คือ ค่าเริ่มต้นของแอตทริบิว

AttNullable คือ การกำหนดเงื่อนไขบังคับไม่อนุญาตให้ค่าแอตทริบิวเป็นค่าว่าง

AttRid คือ หมายเลขเรคอร์ดที่เก็บรายละเอียดข้อมูลของแอตทริบิวนี้

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล ATTNODE ถัดไปในลิสต์

- **โครงสร้างข้อมูล COLNODE** เก็บรายชื่อแอตทริบิว แสดงดังภาพประกอบที่ 3.51 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.51 โครงสร้างข้อมูล COLNODE

Id คือ รหัสแอตทริบิว

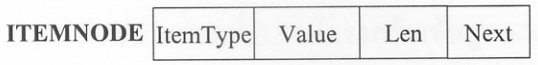
Name คือ ชื่อแอตทริบิว

Len คือ ความยาวชื่อแอตทริบิว

AttPtr คือ ตัวชี้ไปยังรายละเอียดแอตทริบิว

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล COLNODE ถัดไปในลิสต์

- **โครงสร้างข้อมูล ITEMNODE** เก็บค่าของแอตทริบิว แสดงดังภาพประกอบที่ 3.52 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.52 โครงสร้างข้อมูล ITEMNODE

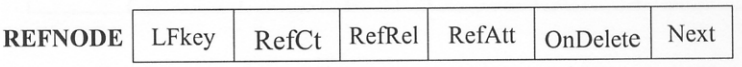
ItemType คือ ชนิดข้อมูล

Value คือ ค่าของแอตทริบิว

Len คือ ความยาวของค่าแอตทริบิว

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล ITEMNODE ถัดไปในลิสต์

- **โครงสร้างข้อมูล REFNODE** เก็บเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว แสดงดังภาพประกอบที่ 3.53 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.53 โครงสร้างข้อมูล REFNODE

LFkey คือ ชื่อแอตทริบิวที่กำหนดให้อ้างอิงค่าแอตทริบิว จากค่าคีย์หลักในอีกรีเลชันหนึ่ง

RefCt คือ จำนวนแอตทริบิวที่กำหนดให้อ้างอิงค่าแอตทริบิว จากค่าคีย์หลักในอีกรีเลชันหนึ่ง

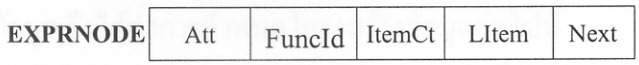
RefRel คือ ชื่อรีเลชันที่ถูกอ้างอิงค่าแอตทริบิว

RefAtt คือ ชื่อแอตทริบิวของรีเลชันที่ถูกอ้างอิงค่าแอตทริบิว

OnDelete คือ การกำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว แบบลบต่อเนื่องกัน

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล REFNODE ถัดไปในลิสต์

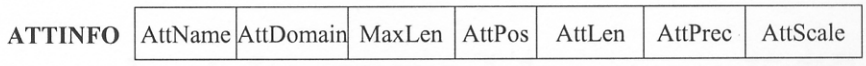
- **โครงสร้างข้อมูล EXPRNODE** เก็บการตรวจสอบค่าแอตทริบิว แสดงดังภาพประกอบที่ 3.54 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.54 โครงสร้างข้อมูล EXPRNODE

Att คือ ชื่อแอตทริบิวที่ต้องการตรวจสอบค่าแอตทริบิว
FuncId คือ รหัสฟังก์ชันที่ใช้ตรวจสอบค่าแอตทริบิว
ItemCt คือ จำนวนค่าแอตทริบิวที่จะใช้ตรวจสอบ
LItem คือ ลิสต์ของค่าแอตทริบิวที่จะใช้ตรวจสอบ
Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล EXPRNODE ถัดไปในลิสต์

- **โครงสร้างข้อมูล ATTINFO** เก็บรายละเอียดเกี่ยวกับแอตทริบิว แสดงดังภาพประกอบที่ 3.55 ซึ่งมีรายละเอียดดังนี้



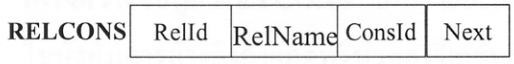
ภาพประกอบที่ 3.55 โครงสร้างข้อมูล ATTINFO

AttName คือ ชื่อแอตทริบิว
AttDomain คือ ชนิดข้อมูลของค่าแอตทริบิว
MaxLen คือ ขนาดความยาวสูงสุดของค่าแอตทริบิวในทุกทุเปิลของรีเลชัน
AttPos คือ ตำแหน่งของค่าแอตทริบิวในทุเปิล
AttLen คือ ความยาวค่าแอตทริบิวหน่วยเป็นไบต์ในทุเปิล
AttPrec คือ จำนวนตำแหน่งของเลขหน้าจุดทศนิยม กรณีกำหนดให้ค่าของแอตทริบิวเป็นข้อมูลชนิดตัวเลข
AttScale คือ จำนวนตำแหน่งของเลขหลังจุดทศนิยม กรณีกำหนดให้ค่าของแอตทริบิวเป็นข้อมูลชนิดตัวเลข

โครงสร้างข้อมูลเกี่ยวกับการลบและปรับปรุงแก้ไขข้อมูลในรีเลชัน

เมื่อจะทำการลบหรือปรับปรุงแก้ไขข้อมูลในรีเลชัน จะต้องมีการจัดเก็บรายละเอียดเกี่ยวกับการกำหนดคีย์ และเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวของรีเลชันที่ต้องการลบหรือปรับปรุงแก้ไขข้อมูล จึงได้ทำการกำหนดโครงสร้างข้อมูลต่อไปนี้

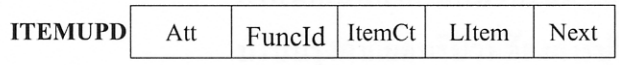
- **โครงสร้างข้อมูล RELCONS** เก็บชื่อรีเลชันและชื่อของเงื่อนไขบังคับการกำหนดคีย์ทั้งหมดของรีเลชัน แสดงดังภาพประกอบที่ 3.56 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.56 โครงสร้างข้อมูล RELCONS

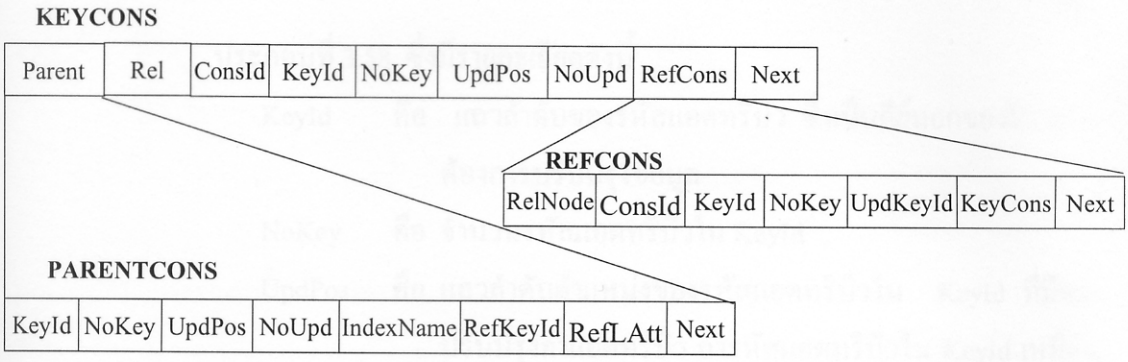
RelId คือ รหัสรีเลชัน
RelName คือ ชื่อรีเลชัน
ConsId คือ รหัสเงื่อนไขบังคับ
Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล RELCONS ถัดไปในลิสต์

- **โครงสร้างข้อมูล ITEMUPD** เก็บชื่อและค่าของแอตทริบิวที่ใช้ในการปรับปรุงข้อมูลของรีเลชัน แสดงดังภาพประกอบที่ 3.57 ซึ่งมีรายละเอียดดังนี้



ภาพประกอบที่ 3.57 โครงสร้างข้อมูล ITEMUPD

NoAtt คือ จำนวนแอตทริบิวที่จะปรับปรุงค่าแอตทริบิว
LAtt คือ ลิสต์ของชื่อแอตทริบิวที่ต้องการปรับปรุงค่าแอตทริบิว
LItem คือ ลิสต์ของค่าแอตทริบิว



ภาพประกอบที่ 3.58 โครงสร้างข้อมูล KEYCONS REFCONS และ PARENTCONS

- **โครงสร้างข้อมูล KEYCONS** เก็บรายละเอียดเกี่ยวกับการกำหนดคีย์ และเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวของรีเลชันที่ต้องการปรับปรุงข้อมูล แสดดังภาพประกอบที่ 3.58 ซึ่งมีรายละเอียดดังนี้

  Parent คือ รายละเอียดของการอ้างอิงค่าแอตทริบิวเป็นคีย์นอก

  Rel คือ ชื่อรีเลชันที่มีการกำหนดค่าคีย์หลักหรือคีย์รอง

  ConsId คือ รหัสเงื่อนไขบังคับการกำหนดค่าคีย์หลักหรือคีย์รอง

  KeyId คือ แถวลำดับของรหัสแอตทริบิวที่ถูกกำหนดเป็นคีย์หลัก หรือคีย์รองของรีเลชัน

  NoKey คือ จำนวนรหัสแอตทริบิวใน KeyId

  UpdPos คือ แถวลำดับของตำแหน่งรหัสแอตทริบิวใน KeyId ที่มีการปรับปรุงค่าแอตทริบิว ถ้ารหัสแอตทริบิวใน KeyId เหมือนกับรหัสแอตทริบิวที่ต้องการปรับปรุงค่าแอตทริบิวจะกำหนดเป็นค่าตำแหน่งหรือลำดับของรหัสแอตทริบิวที่การปรับปรุงค่าแอตทริบิวที่ต้องการปรับปรุงแก้ไข แต่ถ้าไม่เหมือนจะกำหนดเป็น 0

  NoUpd คือ จำนวนของตำแหน่งในแถวลำดับ UpdPos ที่ไม่เป็น 0

  RefCons คือ เก็บรายละเอียดของรีเลชันอื่นในฐานข้อมูลที่มีการอ้างถึงคีย์หลักของรีเลชันที่ต้องการปรับปรุงแก้ไข

  Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล KEYCONS ถัดไปในลิสต์

- **โครงสร้างข้อมูล PARENTCONS** รายละเอียดของแอตทริบิวที่ถูกกำหนดเป็นคีย์นอก เพื่อตรวจสอบว่าเมื่อปรับปรุงค่าแอตทริบิวใหม่แล้ว จะพบค่าใหม่นี้ในคีย์หลักของอีกรีเลชันที่ถูกอ้างอิงถึงค่าแอตทริบิวหรือไม่ ดังแสดงในภาพ

ประกอบที่ 3.58 ซึ่งมีรายละเอียดดังนี้

KeyId คือ แถวลำดับของรหัสแอตทริบิว ซึ่งเป็นคีย์นอกของรีเลชันที่ต้องการปรับปรุงข้อมูล

NoKey คือ จำนวนรหัสแอตทริบิวใน KeyId

UpdPos คือ แถวลำดับตำแหน่งของรหัสแอตทริบิวใน KeyId ที่มีการปรับปรุงค่าแอตทริบิว ถ้ารหัสแอตทริบิวใน KeyId เหมือนกับรหัสแอตทริบิวที่ต้องการปรับปรุงค่าแอตทริบิวจะกำหนดค่าตำแหน่งของรหัสแอตทริบิวที่ต้องการปรับปรุงค่าแอตทริบิวให้ แต่ถ้าไม่เหมือนจะกำหนดเป็น 0

NoUpd คือ จำนวนของตำแหน่งในแถวลำดับ UpdPos ที่ไม่เป็น 0

IndexName คือ ชื่อของดัชนีที่ใช้เก็บคีย์หลักของรีเลชันที่ถูกอ้างถึง

RefKeyId คือ แถวลำดับของรหัสแอตทริบิว ซึ่งเป็นคีย์หลักในอีกรีเลชันหนึ่ง ซึ่งรีเลชันที่ต้องการปรับปรุงข้อมูลอ้างอิงถึง

RefLAtt คือ ลิสต์ของแอตทริบิวในอีกรีเลชันหนึ่ง ซึ่งรีเลชันที่ต้องการปรับปรุงอ้างอิงถึง

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล KEYCONS ถัดไปในลิสต์

- **โครงสร้างข้อมูล REFCONS** เก็บรายละเอียดเกี่ยวกับเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวของรีเลชันอื่นในฐานข้อมูลที่มีการอ้างถึงคีย์หลักของรีเลชันที่ต้องการปรับปรุงข้อมูล แสดดังภาพประกอบที่ 3.58 ซึ่งมีรายละเอียดดังนี้

RelNode คือ รายละเอียดของรีเลชันที่กำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิวนี้

ConsId คือ รหัสเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว

KeyId คือ แถวลำดับของรหัสแอตทริบิวแอตทริบิวของรีเลชันอื่น ที่ถูกกำหนดเงื่อนไขบังคับการอ้างอิงค่าแอตทริบิว จากคีย์หลักของรีเลชันที่ต้องการปรับปรุงแก้ไข

NoKey คือ จำนวนรหัสแอตทริบิวใน KeyId

UpdKeyId คือ แถวลำดับของรหัสแอตทริบิวใน KeyId ที่จะต้องมีการปรับปรุงค่าแอตทริบิวในทุเปิลของรีเลชัน

KeyCons คือ รายละเอียดของการกำหนดคีย์หลักหรือคีย์รองของรีเลชัน

Next คือ ตัวชี้ไปยังโครงสร้างข้อมูล REFCONS ถัดไปในลิสต์