บทที่ 3

ระเบียบวิธี (Methodology)

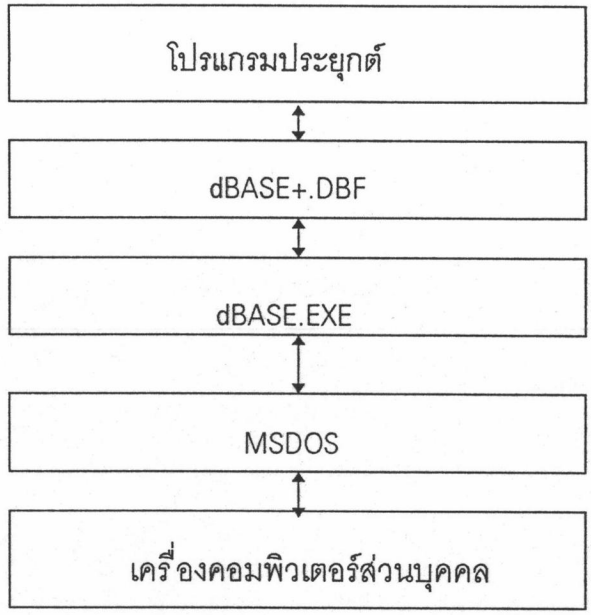
3.1 โครงสร้างแฟ้มข้อมูลบนคอมพิวเตอร์ส่วนบุคคล

เนื่องจากการใช้คอมพิวเตอร์ส่วนบุคคลเป็นที่นิยมกันมาก ทั้งในสถาบันการศึกษา, บริษัท และครอบครัว ซึ่งมีสาเหตุมาจากความก้าวหน้าของเทคโนโลยี ทำให้เครื่องคอมพิวเตอร์ส่วนบุคคลมีราคาถูก แต่มีความสามารถมากขึ้น ประกอบกับมีซอฟต์แวร์ให้เลือกใช้มากมาย ไม่ว่าจะเป็นซอฟต์แวร์ระบบ (System Software), ซอฟต์แวร์ระบบจัดการฐานข้อมูล (Database Management System) และซอฟต์แวร์ประยุกต์ (Application Software) เป็นต้น

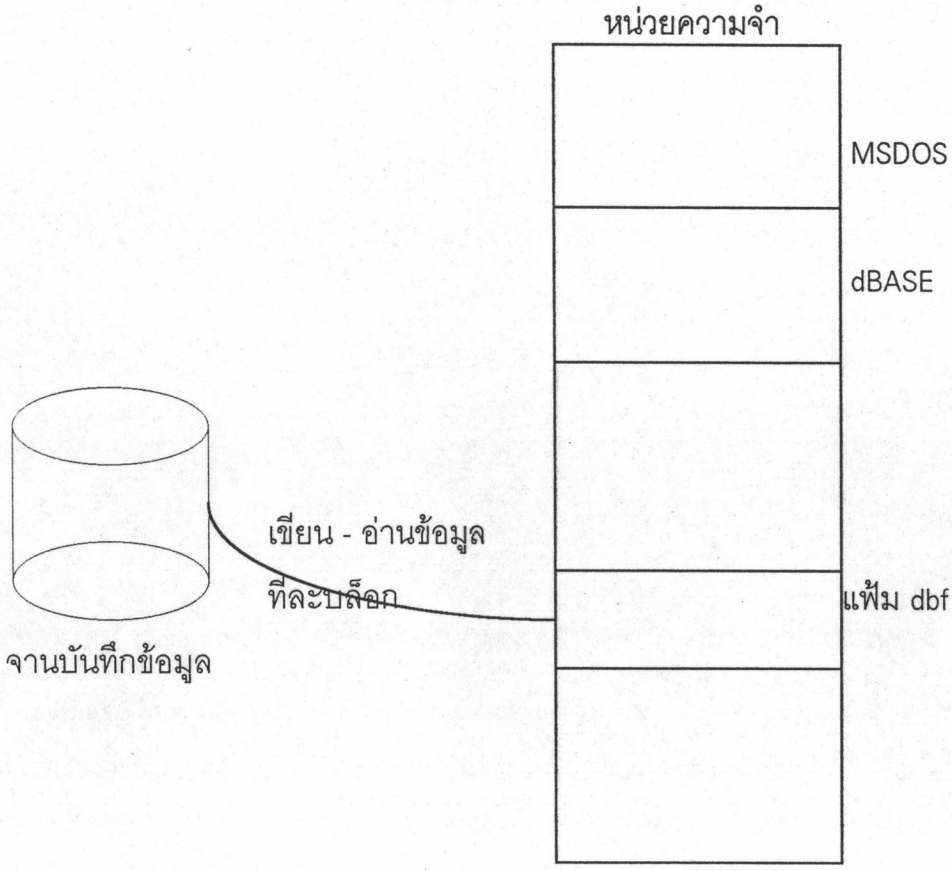
ซอฟต์แวร์ระบบปฏิบัติการที่เป็นที่นิยมใช้กันมาก ได้แก่เอ็มเอสดอส (MSDOS) ส่วนซอฟต์แวร์ระบบจัดการฐานข้อมูลมีอาทิเช่น dBASE III Plus, FOXBASE, CLIPPER เป็นต้น ทั้งนี้เนื่องจากโปรแกรมเหล่านี้มีรูปแแบบคำสั่งเข้าใจได้ง่าย และคล้ายๆกัน ทำให้ผู้ใช้ที่ถนัดการเขียนโปรแกรมหนึ่งใช้อีกโปรแกรมหนึ่งสามารถทำได้โดยง่ายรวมทั้งฐานข้อมูลก็สามารถใช้ร่วมกันได้โดยไม่ต้องแก้ไขเปลี่ยนแปลง

จากเหตุผลดังที่กล่าวมาแล้วข้างต้น ผู้วิจัยจึงได้ศึกษาโครงสร้างระบบการใช้งาน dBASE ตามรูปที่ 3.1 โดยระบบฐานข้อมูลจะประกอบด้วยแฟ้มข้อมูลจำนวนมากจัดเก็บอยู่ในหน่วยเก็บข้อมูลสำรอง การจัดเก็บข้อมูลจะอยู่ภายใต้การควบคุมของเอ็มเอสดอสดังนั้น การเรียกใช้ การสร้างแฟ้มข้อมูลและการปรับปรุงแก้ไขข้อมูลต้องให้เอ็มเอสดอสจัดการทั้งสิ้น

ในการเรียกใช้งานโปรแกรม dBASE จะให้เอ็มเอสดอสอ่านข้อมูลจากหน่วยเก็บสำรองมาเก็บไว้ที่หน่วยความจำหลัก ดังนั้นเอ็มเอสดอสจึงต้องมีการจัดทรัพยากรให้มีประสิทธิภาพ การอ่านหรือเขียนข้อมูลนั้นเอ็มเอสดอสจะอ่าน หรือเขียนข้อมูลเป็นบล็อกโดยให้มีหน่วยเป็นคลัสเตอร์ หรือกลุ่มของเซกเตอร์ เช่น 1 คลัสเตอร์เท่ากับ 2 เซกเตอร์ หรือ 2 x512=1024 ไบต์ ดังรูปที่ 3.2



รูปที่ 3.1 โครงสร้างระบบการใช้งาน dBASE



รูปที่ 3.2 การอ่านเขียนข้อมูลของเอ็มเอสดอสจากหน่วยเก็บข้อมูลสำรอง

สำหรับโครงสร้างภายในของแฟ้มข้อมูล dBASE จะประกอบด้วย

1. ส่วนหัว (Header) ซึ่งแสดงรายละเอียดดังตารางที่ 3.1 และตารางที่ 3.2

2. ส่วนที่ใช้เก็บข้อมูล (Data Record) จะประกอบด้วยระเบียนหลายๆ ระเบียน แต่ละระเบียนมีลักษณะดังนี้

2.1 ระเบียนปกติจะขึ้นต้นด้วยอักษรช่องว่าง (20H) ยกเว้นระเบียนที่ลบทิ้งจะใส่เครื่องหมายดอกจัน (2AH) ไว้แทน

2.2 แต่ละระเบียนไม่มีตัวคั่น (Delimiting Character) ระหว่างแต่ละเขตข้อมูลในระเบียน

2.3 ชนิดของข้อมูลจะเก็บด้วยรหัสแอสกี ดังที่แสดงในตารางที่ 3.3

2.4 dBASE จะใช้ 1AH เป็นตัวบ่งบอกถึงจุดสิ้นสุดของแฟ้มข้อมูล

ตารางที่ 3.1 โครงสร้างส่วนหัวของแฟ้มข้อมูล dBASE

เลขที่ไบต์	ความยาว	ความหมาย
0	1 ไบต์	บิต 0-2: หมายเลขเวอร์ชั่น บิต 3-5: ไว้ใช้สำหรับ SQL บิต 6-7: สำหรับใช้กับ Memo File
1-3	3 ไบต์	วันที่ที่แก้ไขครั้งสุดท้าย (YYMMDD)
4-7	32 บิต	จำนวนระเบียนภายในแฟ้มข้อมูล
8-9	16 บิต	ความยาวของโครงสร้างส่วนหัวของแฟ้มข้อมูล dBASE
10-11	16 บิต	ความยาวของระเบียน
12-31	20 ไบต์	สงวนไว้
32-n	เขตข้อมูลละ 32 ไบต์	ตารางที่ 3.2 รายละเอียดแต่ละเขตข้อมูล
n+1	1 ไบต์	ตัวจบส่วนหัวของแฟ้มข้อมูล dBASE เก็บค่า 0DH

ตารางที่ 3.2 รายละเอียดแต่ละเขตข้อมูลของแฟ้มข้อมูล dBASE

เลขที่ไบต์	ความยาว	ความหมาย
0-10	11 ไบต์	ชื่อเขตข้อมูล เติม 0 ถ้าความยาวไม่เต็ม 11 ไบต์
11	1 ไบต์	ชนิดของเขตข้อมูล (C, D, L, M หรือ N) ซึ่งแสดงไว้ใน ตารางที่ 3.3
12-15	4 ไบต์	สงวนไว้
16	1 ไบต์	ความยาวของเขตข้อมูล
17	1 ไบต์	จำนวนหลังจุดทศนิยม
18-31	14 ไบต์	สงวนไว้

ตารางที่ 3.3 ชนิดข้อมูลของแฟ้มข้อมูล dBASE

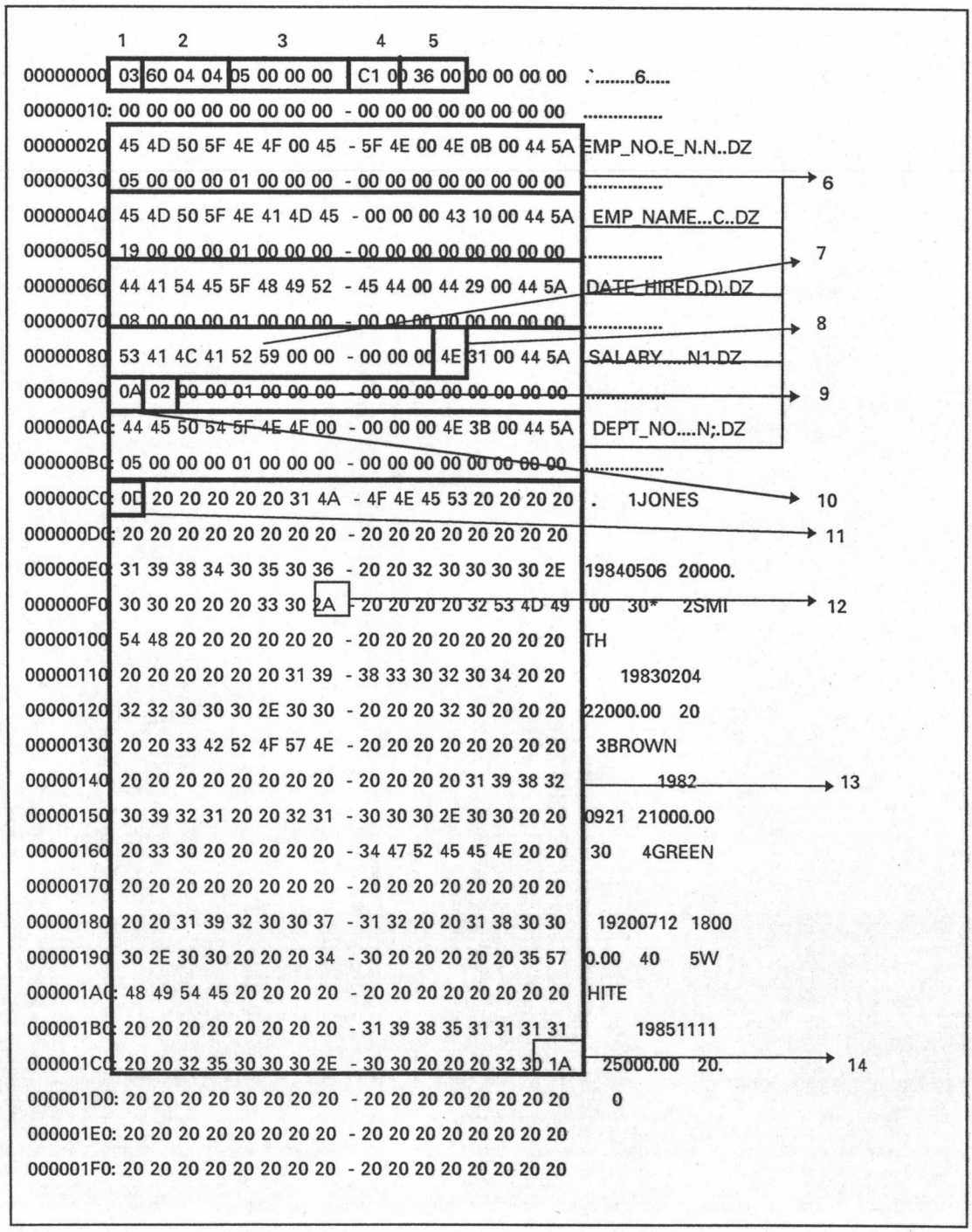
ชนิดของข้อมูล	ความยาว	รูปแบบและค่าต่างๆที่ใช้เก็บ
อักขระ (Character)	ไม่เกิน 254 ไบต์	รหัสแอสกี
ตัวเลข (Numeric)	ไม่เกิน 19 ไบต์	- . 0 1 2 3 4 5 6 7 8 9
ค่าตรรกะ (Logical)	1 ไบต์	Y y N n T t F f
บันทึก (Memo)	10 ไบต์	รหัสแอสกี
วันที่ (Date)	8 ไบต์	YYYYMMDD เช่น 19960401 สำหรับวันที่ 1 เมษายน 1996

ตัวอย่างโครงสร้างของแฟ้มข้อมูล Employee.dbf ที่อ่านได้จากคำสั่ง list stru และ list แสดงดังรูปที่ 3.3 และโครงสร้างของแฟ้มข้อมูลที่ถูกเก็บลงบนหน่วยเก็บสำรอง แสดงดังรูปที่ 3.4

```
. use employee
. list stru
Structure for database: D:employee.dbf
Number of data records:      5
Date of last update  : 04/04/96
Field  Field Name      Type        Width    Dec
    1  EMP_NO          Numeric         5
    2  EMP_NAME        Character      25
    3  DATE_HIRED      Date            8
    4  SALARY          Numeric        10      2
    5  DEPT_NO         Numeric         5
** Total **                          54

. list
Record#  EMP_NO EMP_NAME                DATE_HIRED    SALARY     DEPT_NO
      1       1 JONES                   05/06/84      20000.00      30
      2 *     2 SMITH                   02/04/83      22000.00      20
      3       3 BROWN                   09/21/82      21000.00      30
      4       4 GREEN                   07/12/20      18000.00      40
      5       5 WHITE                   11/11/85      25000.00      20
```

รูปที่ 3.3 โครงสร้างแฟ้มข้อมูล Employee.dbf ที่อ่านได้จากคำสั่ง list stru และ list



รูปที่ 3.4 โครงสร้างของแฟ้มข้อมูลที่ถูกเก็บลงบนหน่วยเก็บสำรอง

จากรูปที่ 3.4 จะพบส่วนที่ควรสนใจดัง ตารางที่ 3.4

ตารางที่ 3.4 โครงสร้างของแฟ้มข้อมูลที่ถูกเก็บลงบนหน่วยเก็บสำรอง

หมายเลข	ความหมาย	คำอธิบายเพิ่มเติม
1	เวอร์ชั่นของโปรแกรม dBASE	โดยไบต์ที่ 0 ตามรูปมีค่า 03 (ฐาน16) เท่ากับ 0000 0011 (ฐาน2) เท่ากับ 3 (ฐาน10) ซึ่งหมายถึง dBASE III
2	วันที่แก้ไขแฟ้มข้อมูลครั้งสุดท้าย	โดยไบต์ที่ 1 เก็บ ค.ศ. ตามรูปมีค่า 60 (ฐาน16) เท่ากับ 0110 0000 (ฐาน2) เท่ากับ 96 (ฐาน10) ไบต์ที่ 2 เก็บเลขลำดับของเดือน ตามรูปมีค่า 04 (ฐาน16) เท่ากับ 0000 0100 (ฐาน2) เท่ากับ 4 (ฐาน10) ไบต์ที่ 3 เก็บวันที่ ตามรูปมีค่า 04 (ฐาน16) เท่ากับ 0000 0100 (ฐาน2) เท่ากับ 4 (ฐาน10) ดังนั้น วันที่แก้ไขแฟ้มข้อมูลครั้งสุดท้าย คือวันที่ 4 เมษายน ค.ศ. 1996
3	จำนวนระเบียนข้อมูล	โดยไบต์ที่ 4 จะเก็บค่าเศษของ “จำนวนระเบียนข้อมูล/256” และไบต์ที่ 5 จะเก็บค่าจำนวนเต็มของ “จำนวนระเบียนข้อมูล/256” ซึ่งสามารถเขียนเป็นสูตรได้ว่า จำนวนระเบียนข้อมูล = (ค่าฐาน10 ของไบต์ที่ 5 X 256) + (ค่าฐาน10 ของไบต์ที่ 4) ตามรูปไบต์ที่ 4 มีค่า 05 (ฐาน16) เท่ากับ 0000 0101 (ฐาน2) เท่ากับ 5 (ฐาน 10) และไบต์ที่ 5 มีค่า 00 (ฐาน 16)

ตารางที่ 3.4 โครงสร้างของแฟ้มข้อมูลที่ถูกเก็บลงบนหน่วยเก็บสำรอง (ต่อ)

หมายเลข	ความหมาย	คำอธิบายเพิ่มเติม
		เท่ากับ 0000 0000 (ฐาน2) เท่ากับ 0 (ฐาน10) ดังนั้น จำนวนระเบียนข้อมูล = (0 X 256) + (5) = 5 ระเบียน
4	ความยาวของโครงสร้างส่วนหัว	ไบต์ที่ 8 ตามรูปมีค่า C1 (ฐาน16) เท่ากับ 1100 0001 (ฐาน2) เท่ากับ 193 (ฐาน10) ดังนั้น ความยาวของโครงสร้างส่วนหัว เท่ากับ 193 ไบต์
5	ความยาวของระเบียน	ไบต์ที่ 10 ตามรูปมีค่า 36 (ฐาน16) เท่ากับ 0011 0110 (ฐาน2) เท่ากับ 54 (ฐาน10) ดังนั้น ความยาวของระเบียน เท่ากับ 54 ไบต์
6	ส่วนของเขตข้อมูล	จะเริ่มที่ออฟเซต (Offset) 32, 64, 96, ...ตามลำดับ โดยแต่ละเขตข้อมูลจะใช้เนื้อที่ 32 ไบต์
7	ชื่อเขตข้อมูล	โดยจะเริ่มจากไบต์ที่ 0 ถึงไบต์ที่ 10 ของแต่ละเขตข้อมูล ซึ่งตามรูปจะมีค่า “53 41 4C 41 52 59 00 00 00 00 00” เมื่อเปลี่ยนข้อความจะได้คำว่า “SALARY” ดังนั้น ชื่อเขตข้อมูลนี้ คือ SALARY
8	ชนิดของเขตข้อมูล	โดยไบต์ที่ 11 ของแต่ละเขตข้อมูลจะใช้เก็บชนิดของเขตข้อมูล ซึ่งตาม

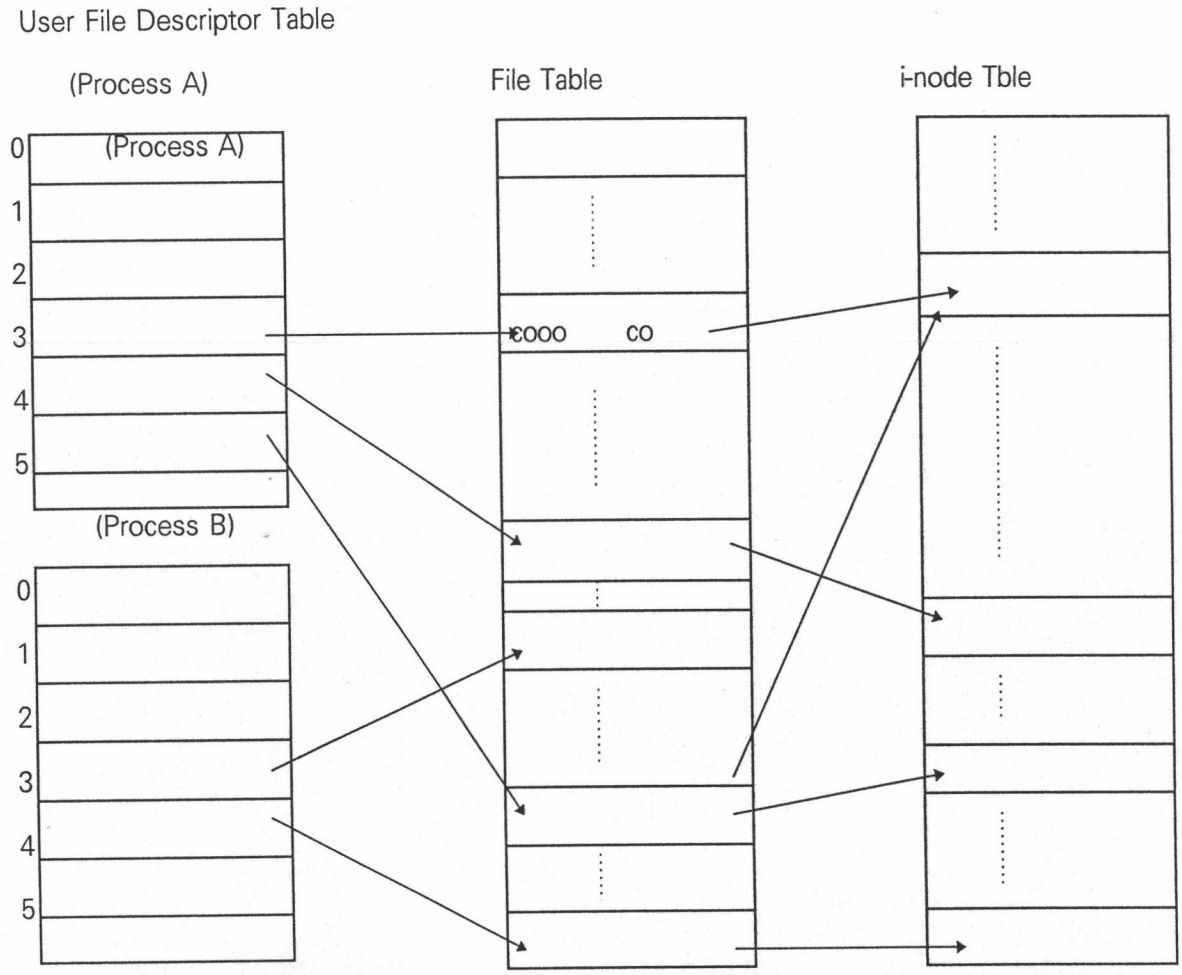
ตารางที่ 3.4 โครงสร้างของแฟ้มข้อมูลที่ถูกเก็บลงบนหน่วยเก็บสำรอง (ต่อ)

หมายเลข	ความหมาย	คำอธิบายเพิ่มเติม
		รูปจะมีค่า “4E” เมื่อเปลี่ยนข้อความจะได้คำว่า “N” ดังนั้น ชนิดของเขตข้อมูลนี้ คือ ตัวเลข
9	จำนวนหลังจุดทศนิยม	โดยไบต์ที่ 17 ของแต่ละเขตข้อมูลจะใช้เก็บจำนวนหลังจุดทศนิยม ซึ่งตามรูปจะมีค่า 02 (ฐาน16) เท่ากับ 0000 0010 (ฐาน2) เท่ากับ 2 (ฐาน10) ดังนั้น จำนวนหลังจุดทศนิยมจะมีได้ 2 หลัก
10	ความยาวของเขตข้อมูล	โดยไบต์ที่ 16 ของแต่ละเขตข้อมูลจะใช้เก็บความยาวของเขตข้อมูล ซึ่งตามรูปจะมีค่า 0A (ฐาน16) เท่ากับ 0000 1010 (ฐาน2) เท่ากับ 10 (ฐาน10) ดังนั้น ความยาวของเขตข้อมูลจะมีได้ 10 หลัก
11	ตัวจบส่วนหัวของแฟ้มข้อมูล dBASE	
12	ระเบียนที่ลบทิ้ง	
13	ส่วนที่ใช้เก็บข้อมูล	
14	จุดสิ้นสุดของแฟ้มข้อมูล dBASE	

3.2 โครงสร้างแฟ้มข้อมูลบนยูนิกซ์

เนื่องจากยูนิกซ์เป็นระบบปฏิบัติการในเทคโนโลยีแบบเปิด (Open System) ที่ไม่ต้องผูกติดกับระบบหรืออุปกรณ์ยี่ห้อใดยี่ห้อหนึ่งโดยเฉพาะ และมีซอฟต์แวร์สนับสนุนมากมาย รวมทั้งมีการผนวก TCP/IP เข้ากับยูนิกซ์ ทำให้ยูนิกซ์เป็นระบบปฏิบัติการที่นิยมใช้กันบนเครือข่ายอินเตอร์เน็ต

ด้วยเหตุผลตามที่กล่าวมาข้างต้น ผู้วิจัยจึงได้ศึกษาการทำงานของยูนิกซ์ เมื่อมีกระบวนการ (Process) เรียกใช้แฟ้มข้อมูล ดังรูปที่ 3.5 โดยยูนิกซ์จะค้นชื่อแฟ้มข้อมูลนั้นที่แฟ้มสารบบตามเส้นทาง (Path) ที่กระบวนการนั้นมีสิทธิ ถ้ายูนิกซ์ค้นหาชื่อแฟ้มข้อมูลพบก็จะได้หมายเลข i-node ของแฟ้มข้อมูลนั้น โดยยูนิกซ์จะควบคุมการเรียกใช้แฟ้มข้อมูลจาก File Table (FT) และ User File Descriptor Table (UFDT) โดย FT จะเป็นโครงสร้างที่ใช้ร่วมกันทุกกระบวนการ แต่ UFDT แต่ละกระบวนการจะใช้แยกจากกัน เมื่อมีกระบวนการเปิดแฟ้มข้อมูล ยูนิกซ์จะกำหนดทางเข้า (Entry) ของแต่ละตารางให้สัมพันธ์กับหมายเลข i-node โดยยูนิกซ์จะควบคุมดูแล FT ให้สามารถบอกตำแหน่งภายในแฟ้มข้อมูลที่จะอ่านหรือเขียนครั้งต่อไปของแต่ละกระบวนการ สำหรับ UFDT จะใช้เก็บตัวบ่งลักษณะแฟ้มข้อมูลที่กระบวนการนั้นเปิดใช้อยู่ทั้งหมด



รูปที่ 3.5 ความสัมพันธ์ระหว่าง User File Descriptor Table, File Table และ i-node Table

3.3 คลังชุดคำสั่งฐานข้อมูลบนยูนิกซ์

จากการศึกษาการพัฒนาคลังชุดคำสั่งฐานข้อมูล (Database Library) ของ Richard Stevens ซึ่งได้ทำการพัฒนาคลังชุดคำสั่งฐานข้อมูลเฉพาะส่วนที่ใช้ติดต่อกับฐานข้อมูลเท่านั้น ดังมีรายละเอียดต่อไปนี้

3.3.1 ฟังก์ชั่นภายในคลังชุดคำสั่งฐานข้อมูล

```
DB *db_open (const char *pathname, int oflag, int mode);
                        Return: pointer to DB structure if OK, NULL on error

void db_close(DB *db);
```

ฟังก์ชั่น db_open() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการใช้ฐานข้อมูลเป็นครั้งแรก โดยฟังก์ชั่น db_open() จะส่งตัวชี้กลับไปยังโครงสร้าง DB แก่ผู้เรียก

อาร์กิวเมนต์ที่ 1 (pathname) เป็นสายอักษรระบุชื่อฐานข้อมูล ถ้าฟังก์ชั่น db_open() ทำงานสำเร็จจะสร้างแฟ้มข้อมูล pathname.idx และ pathname.dat ขึ้นมา

อาร์กิวเมนต์ที่ 2 (oflag) เป็นรหัสเลขจำนวนเต็มซึ่งระบุจุดมุ่งหมายของการเปิดใช้ฐานข้อมูล สัญลักษณ์ที่ใช้แทนเลขจำนวนเต็ม ได้แก่

- O_RDONLY เป็นการเปิดเพื่อการอ่านเท่านั้น
- O_WRONLY เป็นการเปิดเพื่อการเขียนเท่านั้น
- O_RDWR เป็นการเปิดเพื่อการอ่านและการเขียน

อาร์กิวเมนต์ที่ 3 (mode) เป็นรหัสเลขจำนวนเต็มซึ่งระบุสิทธิ์การเข้าถึงฐานข้อมูล ในกรณีที่ใช้ฟังก์ชั่น db_open() สร้างฐานข้อมูล

ส่วนฟังก์ชั่น db_close() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการเลิกใช้ฐานข้อมูล และปิดฐานข้อมูล

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

```
int db_store (DB *db, const char *key, const char *data, int flag);
                                    Return: 0 if OK, nonzero on error
```

ฟังก์ชั่น db_stor() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการเพิ่มหรือทำการแก้ไขข้อมูลภายในฐานข้อมูล โดยฟังก์ชั่น db_stor() จะส่งกลับเลขจำนวนเต็มแก่ผู้เรียก

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

อาร์กิวเมนต์ที่ 2 (key) เป็นสายอักษรระบุค่ากุญแจ (Key) ของข้อมูล โดยมีข้อแม้ว่าค่าของกุญแจต้องไม่ซ้ำกัน (Unique)

อาร์กิวเมนต์ที่ 3 (data) เป็นสายอักษรระบุค่าข้อมูล

อาร์กิวเมนต์ที่ 4 (flag) เป็นรหัสเลขจำนวนเต็มซึ่งใช้ระบุการดำเนินการกับฐานข้อมูล สัญลักษณ์ที่ใช้แทนเลขจำนวนเต็ม ได้แก่

- DB_INSERT เป็นการเพิ่มข้อมูล
- DB_REPLACE เป็นการแก้ไขข้อมูล

```
char *db_fetch(DB *db, const char *key);
                                    Return: pointer to data if OK, NULL if record not found
```

ฟังก์ชั่น db_fetch() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการดูข้อมูลภายในฐานข้อมูล โดยใช้กุญแจเป็นตัวค้นหา ฟังก์ชั่น db_fetch() จะส่งกลับตัวชี้ไปยังสายอักษรซึ่งระบุค่าข้อมูลแก่ผู้เรียก

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

อาร์กิวเมนต์ที่ 2 (key) เป็นสายอักษรระบุค่ากุญแจของข้อมูล

```
int db_delete(DB *db, const char *key);
                    Return: 0 if OK, -1 if record not found
```

ฟังก์ชั่น db_delete() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการลบข้อมูลภายในฐานข้อมูลทิ้ง โดยใช้กุญแจเป็นตัวระบุข้อมูล โดยฟังก์ชั่น db_delete() จะส่งกลับเลขจำนวนเต็มแก่ผู้เรียก

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

อาร์กิวเมนต์ที่ 2 (key) เป็นสายอักษรระบุค่ากุญแจของข้อมูล

```
void db_rewind(DB *db);

char *db_nextrec(DB *db, char *key);
            Return: pointer to data if OK, NULL on end of file
```

ฟังก์ชั่น db_rewind() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการกลับไปที่ข้อมูลแรกของฐานข้อมูล โดยฟังก์ชั่น db_rewind() ไม่ต้องส่งค่ากลับให้ผู้เรียก

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

ฟังก์ชั่น db_nextrec() จะถูกเรียกใช้ เมื่อผู้เรียกต้องการดูข้อมูลภายในฐานข้อมูลตามลำดับ ฟังก์ชั่น db_nextrec() จะส่งกลับตัวชี้ไปยังสายอักษรซึ่งระบุค่าข้อมูลแก่ผู้เรียก

อาร์กิวเมนต์ที่ 1 (db) เป็นตัวชี้ไปยังโครงสร้าง DB ซึ่งผู้เรียกได้ค่ากลับมาจากฟังก์ชั่น db_open()

อาร์กิวเมนต์ที่ 2 (key) เป็นสายอักษรระบุค่ากุญแจของข้อมูล

3.3.2 โครงสร้างแฟ้มข้อมูล

โครงสร้างแฟ้มข้อมูลที่ใช้ภายในคลังชุดคำสั่งฐานข้อมูล จะประกอบไปด้วย 2 แฟ้มข้อมูล คือ แฟ้มดรรชนี (Index File) และแฟ้มข้อมูล (Data File) ซึ่งจะเก็บเฉพาะข้อมูลเท่านั้น สำหรับแฟ้มดรรชนีจะเก็บค่าที่เป็นกุญแจซึ่งจะใช้โยงความสัมพันธ์ไปยังแฟ้มข้อมูล Stevens ได้เลือกเทคนิค Hashing มาใช้ในการจัดโครงสร้างแฟ้มดรรชนีด้วยเหตุผลที่ว่า โครงสร้างข้อมูล(Data Structure) ไม่ยุ่งยากเมื่อเทียบกับเทคนิคอื่นๆ เช่น B-tree และการเข้าถึงข้อมูลทำได้รวดเร็ว การที่เราจะทราบว่าแฟ้มที่เรากำลังพิจารณาเป็นแฟ้มอะไร ให้สังเกตจากคำต่อท้ายชื่อแฟ้ม ถ้าเป็นแฟ้มดรรชนีจะมีคำต่อท้ายว่า .idx แต่ถ้าเป็นแฟ้มข้อมูลจะมีคำต่อท้ายว่า .dat

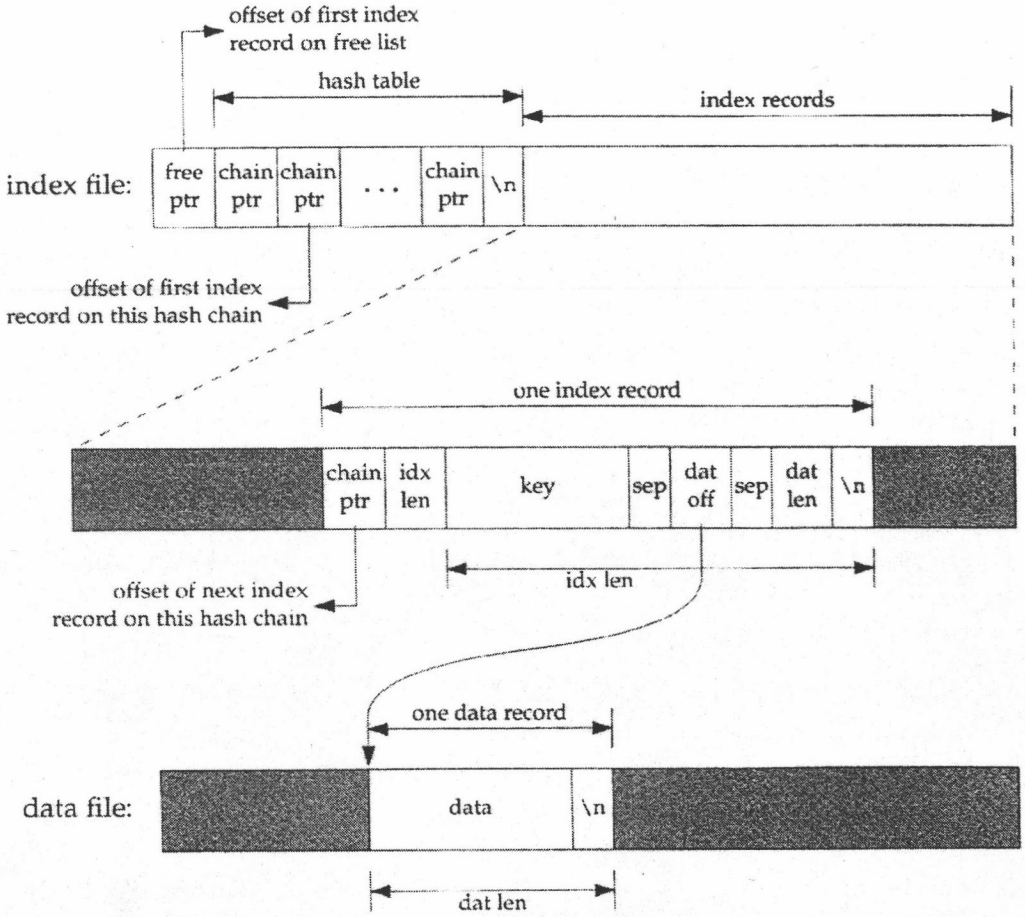
รูปที่ 3.6 จะแสดงโครงสร้างภายในแฟ้มดรรชนี และแฟ้มข้อมูล โดยแฟ้มดรรชนีจะประกอบด้วย 3 ส่วน ดังต่อไปนี้

- Free list pointer
- Hash Table
- Index record

Free list pointer จะใช้เก็บตำแหน่ง (Offset) ของระเบียนภายในแฟ้มดรรชนีที่ถูกลบทิ้ง โดยใช้ฟังก์ชั่น db_delete()

Hash Table จะใช้เก็บตำแหน่งของระเบียนภายในแฟ้มดรรชนีที่ผ่านการหาเลขที่อยู่แบบ hash เพื่อที่จะใช้โยงไปยังระเบียนอื่นๆภายในแฟ้มดรรชนีที่มีเลขที่อยู่แบบ hash ค่าเดียวกัน

Index record จะใช้เก็บตำแหน่งของระเบียนภายในแฟ้มข้อมูล, ค่ากุญแจของข้อมูล, ตำแหน่งของระเบียนภายในแฟ้มดรรชนีตัวถัดไป ที่มีเลขที่อยู่แบบ hash ค่าเดียวกัน



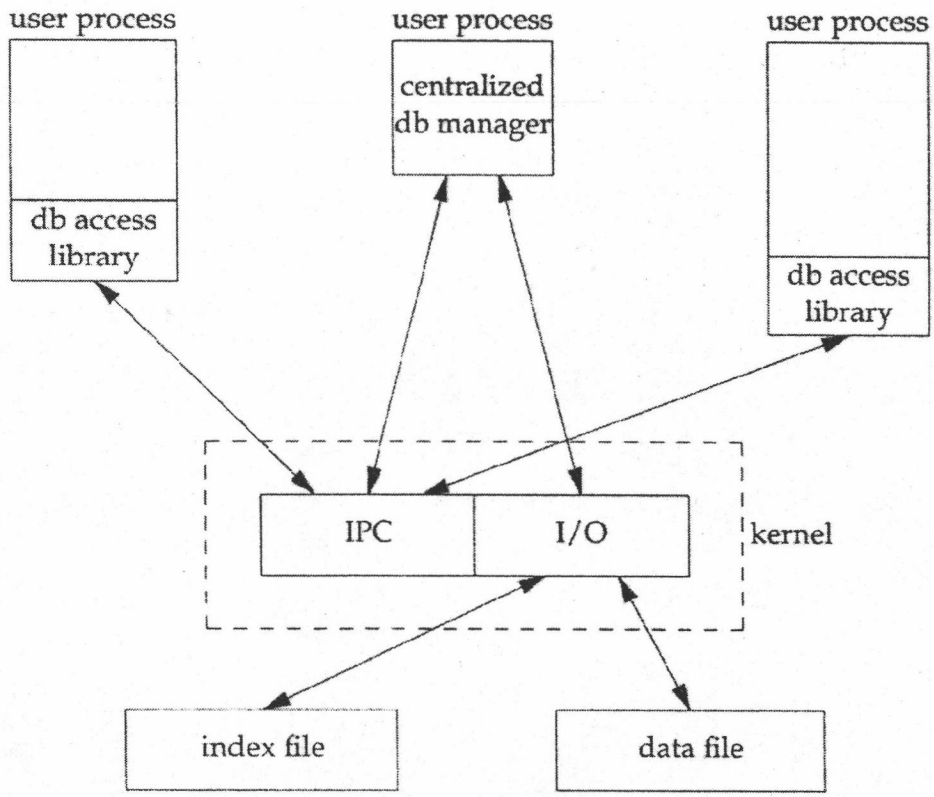
รูปที่ 3.6 การจัดของแฟ้มดรรชนี และ แฟ้มข้อมูล

3.3.3 รูปแบบการทำงานของฐานข้อมูล

รูปแบบการทำงานในกรณีมีหลายกระบวนการเข้าใช้ฐานข้อมูลเดียวกัน มี 2 รูปแบบ คือ

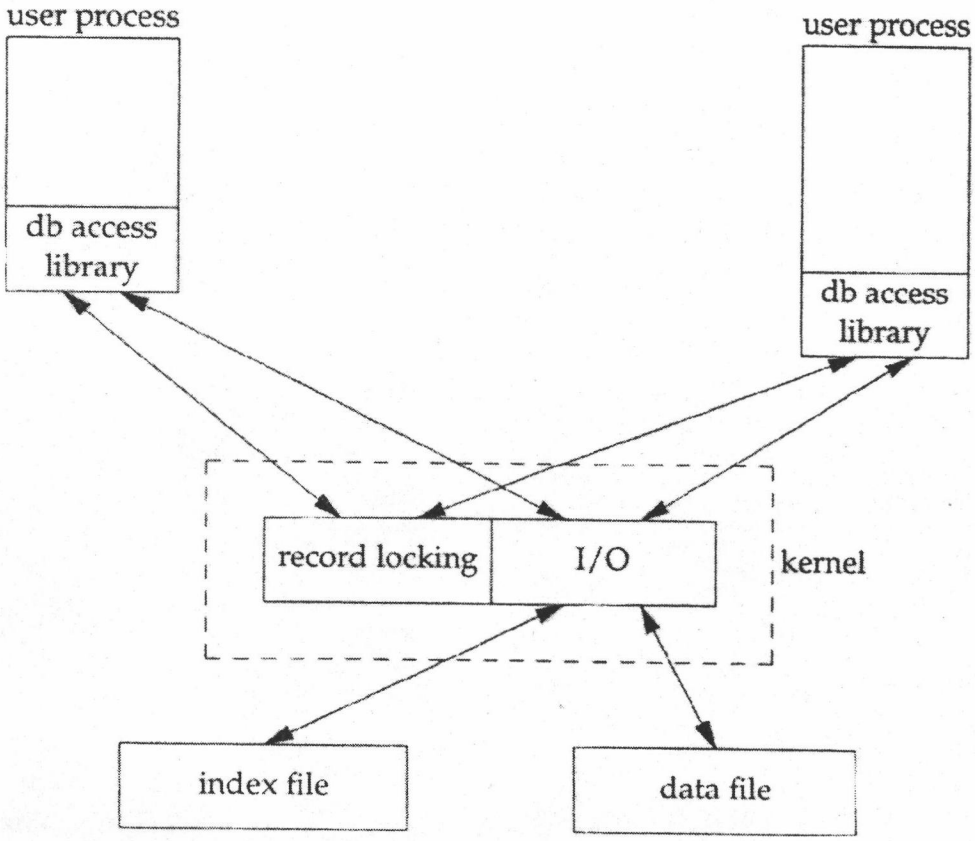
- แบบรวมศูนย์ (Centralized) มี Database Manager เป็นกระบวนการเดียวที่สามารถติดต่อกับฐานข้อมูลได้ กระบวนการอื่นจะติดต่อกับ Database Manager โดยผ่าน Interprocess Communication บนยูนิกซ์ ดังรูปที่ 3.7

- แบบแยกศูนย์ (Decentralized) ทุกกระบวนการสามารถติดต่อกับฐานข้อมูลโดยตรง ดังรูปที่ 3.8



รูปที่ 3.7 การทำงานแบบรวมศูนย์

Richard Stevens ได้เลือกเทคนิคแบบแยกศูนย์ เนื่องจากต้องการตัดการทำงานด้าน Interprocess Communication ออกไป ซึ่งจะทำให้ทำงานได้เร็วกว่าเทคนิคแบบรวมศูนย์ แต่ต้องพัฒนาฟังก์ชั่นควบคุมการเรียกใช้ข้อมูล เพื่อป้องกันไม่ให้เกิดการเขียนระเบียนเดียวกันพร้อมกันจากสองกระบวนการ



รูปที่ 3.8 การทำงานแบบแยกศูนย์

3.3.4 ระบบคอมพิวเตอร์ที่ใช้ในการพัฒนาโปรแกรมและทดสอบ

ผู้เขียนได้ใช้คอมพิวเตอร์ส่วนบุคคล หน่วยประมวลผลกลางเบอร์ 80386 ใช้ระบบปฏิบัติการยูนิกซ์ System V Release 4

3.3.5 ข้อจำกัดของคลังชุดคำสั่งฐานข้อมูล

1. ระบบไม่ได้มีชุดคำสั่งที่ใช้ในการติดต่อบนเครือข่าย เพื่อให้ใช้ข้อมูลร่วมกันได้

2. การปรับแต่ง (Tuning) การดำเนินงานของระบบฐานข้อมูลทำได้ไม่สะดวก ทั้งนี้เนื่องจากการทำงานแบบแยกศูนย์แต่ละกระบวนการเป็นอิสระจากกัน โดยกระบวนการใดจะเข้าไปประมวลผลต้องขึ้นกับระบบปฏิบัติการยูนิกซ์ตัดสินใจซึ่งผู้พัฒนาโปรแกรมไม่สามารถเข้าไปควบคุมในส่วนนี้ได้

3. ระบบควบคุมในกรณีเพิ่มข้อมูลยุ่งยาก ทั้งนี้เนื่องจากการออกแบบแฟ้มข้อมูลเป็นแบบความยาวแปรได้ (variable length) เมื่อเพิ่มข้อมูลต้องไปหาพื้นที่จากข้อมูลที่ลบแล้วและมีขนาดข้อมูลเท่ากันก่อนเพื่อลดการแตกกระจาย (Fragmentation) ซึ่งถ้าเป็นกรณีของระบบฐานข้อมูลที่มีการลบและเพิ่มข้อมูลมากๆ จะเสียเวลานานในการค้นหาพื้นที่ว่างที่เหมาะสมเพื่อทำการเพิ่มข้อมูล

4. สำหรับผู้ใช้ที่ไม่มีความรู้ในด้านการเขียนโปรแกรม จะไม่สะดวกในการเขียนโปรแกรมประยุกต์ร่วมกับคลังชุดคำสั่งฐานข้อมูล แม้ว่าคลังชุดคำสั่งจะช่วยลดคำสั่งไปมากก็ตาม

5. การจัดเก็บข้อมูลจะเป็นภาวะข้อความ (Text Mode) ทำให้เปลืองเนื้อที่ในการจัดเก็บตัวเลข