บทที่ 2
ทฤษฎีระบบฐานข้อมูล และอัลกอริทึม

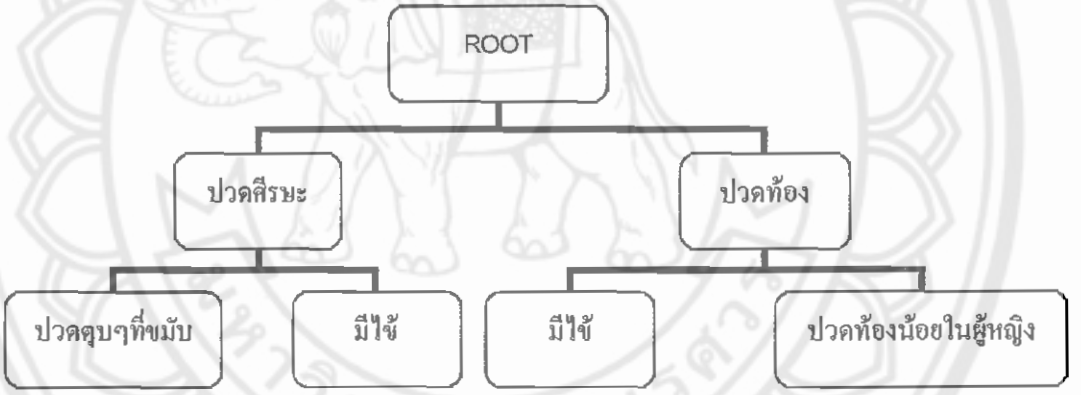
2.1 ระบบฐานข้อมูล (Database System) [4]

ฐานข้อมูล คือการรวบรวมข้อมูลต่างๆอย่างมีโครงสร้างและมีความสัมพันธ์กัน ไว้ด้วยกัน โดยที่จะสามารถจัดการกับข้อมูลนั้นได้อย่างสะดวกและรวดเร็ว โดยอาศัยการจัดการฐานข้อมูล

2.1.1 โมเดลข้อมูล (Data Model)

เพื่อให้การจัดการฐานข้อมูลอยู่ในรูปแบบการประมวลผลด้วยการเขียนโปรแกรมได้ จำเป็นจะต้องอาศัยโมเดลข้อมูลเพื่อให้สามารถ สร้าง ลบ แก้ไข เพิ่ม บันทึก กู้ข้อมูลคืน ตลอดจนสามารถทำงานต่างๆได้อย่างมีประสิทธิภาพ โดยที่เราสามารถแยกโมเดลได้เป็น 3 รูปแบบดังนี้

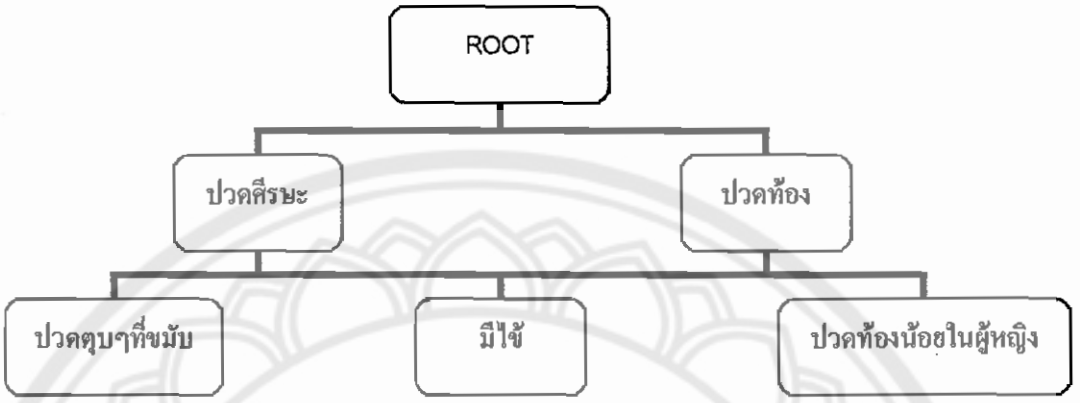
2.1.1.1 โมเดลต้นไม้ (Hierarchical Data Model) เป็นการจัดรูปแบบข้อมูลให้มี Root หรือรากอยู่บนสุดแล้วแตกแขนงออกมาเรื่อยๆ



รูปที่ 2.1 Hierarchical Data Model

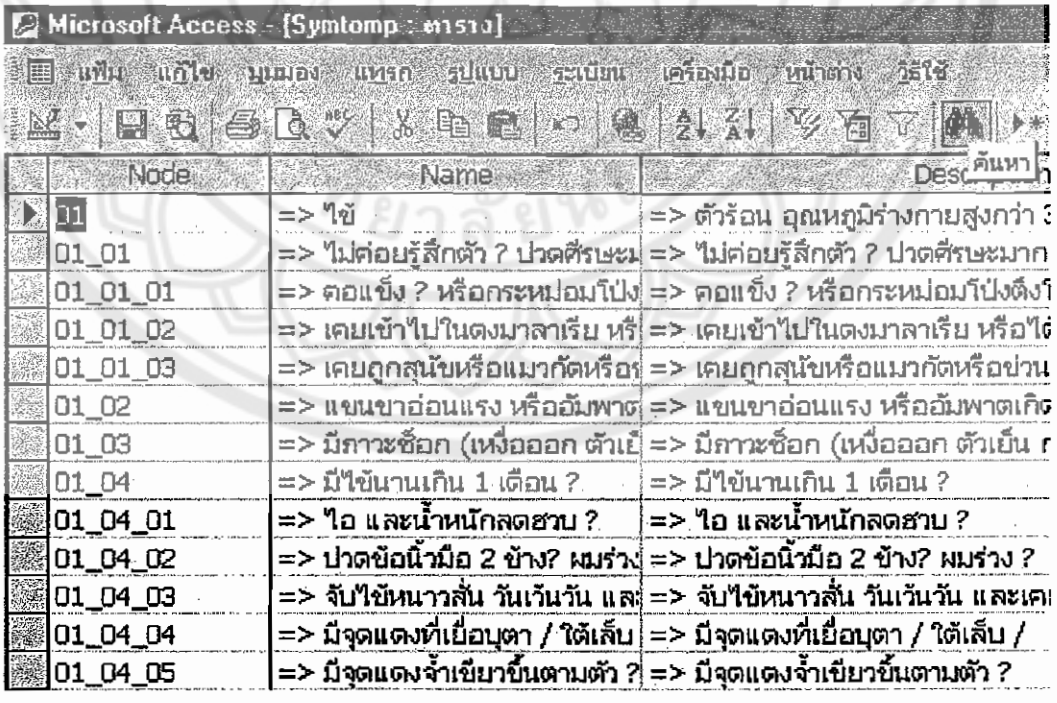
โครงสร้างแบบนี้สามารถพัฒนาได้ง่าย เนื่องจากมีโครงสร้างข้อมูลเป็นแบบพื้นฐาน แต่ยังไม่ตอบสนองความต้องการที่เพิ่มมากขึ้นและเกิดการซ้ำซ้อนของข้อมูลขึ้นได้

2.1.1.2 โมเดลแบบเครือข่าย (Network Database Model) เป็นการจัดรูปแบบข้อมูลให้สามารถเชื่อมโยงความสัมพันธ์ ซึ่งมีความยุ่งยากในการออกแบบและดูแลรักษาระบบ แต่มีประโยชน์คือลดความซ้ำซ้อนของข้อมูล



รูปที่ 2.2 Network Data Model

2.1.1.3 โมเดลประเภทความสัมพันธ์ (Relational Data Model) เป็นการจัดรูปแบบฐานข้อมูลเป็นตาราง ซึ่งบรรจุด้วยฟิลด์ต่างๆ และกำหนดค่าคีย์หลัก โดยฟิลด์ที่เหลือจะเป็นข้อมูลที่ต้องพึ่งพาคีย์หลัก



Node	Name	Desc
01	=> ไข้	=> ตัวร้อน อุณหภูมิร่างกายสูงกว่า 3
01_01	=> ไม่ค่อยรู้สึกตัว ? ปวดศีรษะม	=> ไม่ค่อยรู้สึกตัว ? ปวดศีรษะมาก
01_01_01	=> คอแข็ง ? หรือกระหม่อมโป่ง	=> คอแข็ง ? หรือกระหม่อมโป่งตึงโ
01_01_02	=> เคยเข้าไปในดงมาลาเรีย หรื	=> เคยเข้าไปในดงมาลาเรีย หรือไ
01_01_03	=> เคยถูกสุนัขหรือแมวกัดหรือข	=> เคยถูกสุนัขหรือแมวกัดหรือข่วน
01_02	=> แขนขาอ่อนแรง หรืออัมพาต	=> แขนขาอ่อนแรง หรืออัมพาตเกิด
01_03	=> มีภาวะช็อก (เหงื่อออก ตัวเย็	=> มีภาวะช็อก (เหงื่อออก ตัวเย็น ร
01_04	=> มีไข้นานเกิน 1 เดือน ?	=> มีไข้นานเกิน 1 เดือน ?
01_04_01	=> ไอ และน้ำหนักลดฮวบ ?	=> ไอ และน้ำหนักลดฮวบ ?
01_04_02	=> ปวดข้อนิ้วมือ 2 ข้าง? ผมร่วง	=> ปวดข้อนิ้วมือ 2 ข้าง? ผมร่วง ?
01_04_03	=> จับไข้หนาวสั่น วันเว้นวัน และ	=> จับไข้หนาวสั่น วันเว้นวัน และเคย
01_04_04	=> มีจุดแดงที่เยื่อบุตา / ใต้เล็บ	=> มีจุดแดงที่เยื่อบุตา / ใต้เล็บ /
01_04_05	=> มีจุดแดงจ้ำเขียวขึ้นตามตัว ?	=> มีจุดแดงจ้ำเขียวขึ้นตามตัว ?

รูปที่ 2.3 Relational Data Model

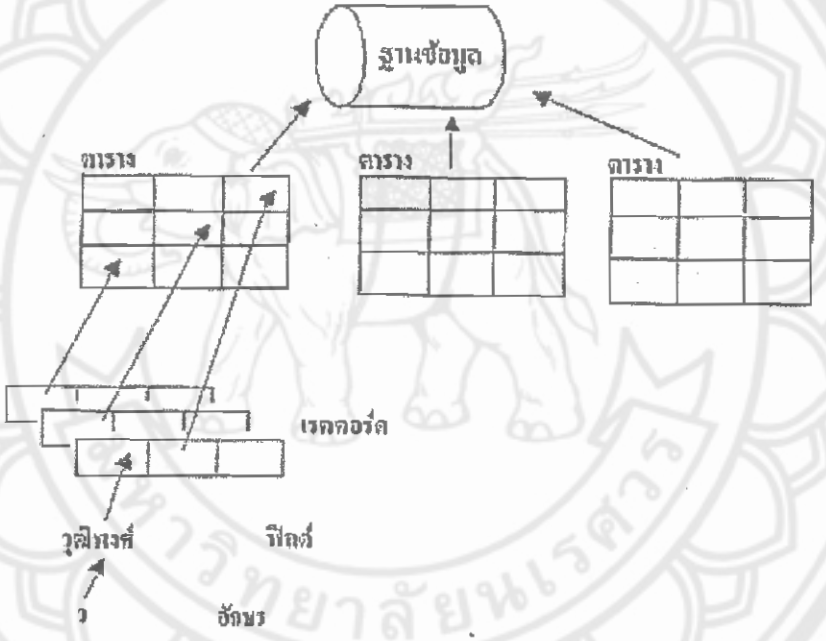
2.1.2 องค์ประกอบของระบบฐานข้อมูล

- ข้อมูล (Data) เป็นส่วนที่จำเป็นมากในระบบฐานข้อมูล

- ฮาร์ดแวร์(Hardware) เป็นหน่วยความจำของข้อมูลที่จะเก็บข้อมูลทุกอย่างไว้และประมวลผลออกมา

- ซอฟแวร์(Software) เป็นโปรแกรมที่ใช้ในการจัดการฐานข้อมูล หรือ DBMS (Database Management System)

- ผู้ใช้ระบบฐานข้อมูล (U s e r) มีหลายประเภท ได้แก่ โปรแกรมเมอร์ (Programmer), ผู้ใช้บริการของระบบ (End User), ผู้ปฏิบัติการระบบ (Database Operator) และผู้บริหารระบบฐานข้อมูล (D a t a b a s e A d m i n i s t r a t o r หรือ D B A)



รูปที่ 2.4 โครงสร้างของข้อมูลที่เก็บในระบบจัดการฐานข้อมูล

2.1.3 ข้อดีของการใช้ระบบจัดการฐานข้อมูล

- ลดความซ้ำซ้อนของการเก็บข้อมูลได้ (Redundancy can be reduced)

- หลีกเลี่ยงความขัดแย้งกันของข้อมูลได้(Inconsistency can be avoided)

- สามารถใช้ข้อมูลร่วมกันได้ (The data can be share)

- สามารถจัดหาระบบความปลอดภัยที่รัดกุมได้ (Security restriction can be applied) โดยกำหนดระดับความสามารถในการเรียกใช้ข้อมูลของผู้ใช้แต่ละคน ให้แตกต่างกันตามความรับผิดชอบได้

2.1.4 โครงสร้างของระบบฐานข้อมูล

สถาปัตยกรรมของฐานข้อมูลเป็นการอธิบายถึงรูปแบบและโครงสร้างของข้อมูลภายในระบบฐานข้อมูลโดยทั่วไปในระดับแนวความคิดไม่ขึ้นอยู่กับโครงสร้างของระบบฐานข้อมูลนั้นๆ สำหรับสถาปัตยกรรมของระบบฐานข้อมูลที่นิยมใช้ได้แก่ สถาปัตยกรรม ANSI/SPARC ได้แบ่งออกเป็น 3 ระดับดังนี้

ระดับ Internal เป็นสถาปัตยกรรมในระดับที่เกี่ยวข้องกับโครงสร้างทางกายภาพในการจัดเก็บข้อมูลในฐานข้อมูลมากที่สุดเนื่องจากเป็นระดับที่กล่าวถึงการจัดเก็บข้อมูล

ระดับ **External** เป็นระดับที่เกี่ยวข้องกับผู้ใช้มากที่สุด เนื่องจากเป็นระดับที่กล่าวถึงมุมมองที่มีต่อข้อมูลของผู้ใช้แต่ละคน

ระดับ **Conceptual** เป็นสถาปัตยกรรมที่กล่าวถึงโครงสร้างของข้อมูลในระดับแนวความคิด ซึ่งเป็นภาพของโครงสร้างข้อมูลที่ใช้แทนโครงสร้างทางกายภาพของข้อมูลที่แท้จริงที่เก็บอยู่ในฐานข้อมูล

2.2 การออกแบบฐานข้อมูล [2],[8]

ฐานข้อมูลนับเป็นส่วนที่สำคัญสำหรับงานสารสนเทศที่ใช้คอมพิวเตอร์ในการประมวลผล เนื่องจากเป็นการเป็นส่วนที่ใช้จัดเก็บข้อมูลต่างๆ ซึ่งใช้เป็นอินพุทของระบบสารสนเทศจึงต้องให้ความสำคัญกับการออกแบบฐานข้อมูลเช่นเดียวกับการออกแบบส่วนประมวลผล

2.2.1 การพัฒนาระบบฐานข้อมูล (Database Life Cycle)

วงจรชีวิตของการพัฒนาระบบฐานข้อมูล เป็นขั้นตอนที่กำหนดขึ้น เพื่อใช้เป็นแนวทางในการพัฒนาระบบฐานข้อมูลขึ้นใช้งาน ประกอบด้วยขั้นตอนต่างๆ ดังนี้

Database Initial Study เป็นขั้นตอนแรกของการพัฒนาระบบฐานข้อมูล ในขั้นตอนนี้ ผู้พัฒนาระบบฐานข้อมูลจะต้องวิเคราะห์ความต้องการต่าง ๆ ของผู้ใช้ เพื่อกำหนดจุดมุ่งหมาย ปัญหา ขอบเขตและกฎระเบียบต่างๆ ของระบบฐานข้อมูลที่จะพัฒนาขึ้นเพื่อใช้เป็นแนวทางในการออกแบบฐานข้อมูลในขั้นตอนต่อไป

Database Design เป็นขั้นตอนในการเอารายละเอียดต่างๆที่ได้จากการวิเคราะห์ในขั้นตอนแรกมากำหนดเป็นแนวทางในการออกแบบ ซึ่งแบ่งเป็น 3 ระดับ คือ การออกแบบฐานข้อมูลในระดับ Conceptual, Logical และ Physical

Implementation and Loading เป็นขั้นตอนที่นำเอาโครงร่างต่างๆ ของระบบฐานข้อมูลที่ได้จากการออกแบบในขั้นตอนการออกแบบ มาสร้างเป็นข้อมูลที่จะใช้เก็บจริง

รวมทั้งแปลงข้อมูลของระบบงานเดิม ให้สามารถนำมาใช้งานในระบบฐานข้อมูลที่พัฒนาขึ้นมาใหม่

Testing and Evaluation เป็นขั้นตอนของการทดสอบระบบฐานข้อมูลที่พัฒนาขึ้น เพื่อหาข้อผิดพลาดต่างๆรวมทั้งทำการประเมินความสามารถของระบบฐานข้อมูลเพื่อนำไปเป็นแนวทางในการปรับปรุงให้รองรับความต้องการความต้องการของผู้ใช้ในด้านต่างๆได้อย่างครบถ้วนและถูกต้อง

Operation เป็นขั้นตอนที่นำอาระบบฐานข้อมูลที่พัฒนาขึ้นเรียบร้อยแล้วไปใช้งานจริง

Maintenance and Evolution เป็นขั้นตอนที่เกิดขึ้นระหว่างการใช้ฐานข้อมูลจริง เพื่อบำรุงรักษาให้ระบบฐานข้อมูลทำงานได้อย่างมีประสิทธิภาพ รวมทั้งขั้นตอนการแก้ไขและปรับปรุงระบบฐานข้อมูลในกรณีที่มีการเพิ่ม หรือเปลี่ยนแปลงความต้องการของผู้ใช้

2.2.2 ขั้นตอนการออกแบบฐานข้อมูล

การออกแบบฐานข้อมูลแบ่งออกได้เป็น 3 ขั้นตอนดังนี้

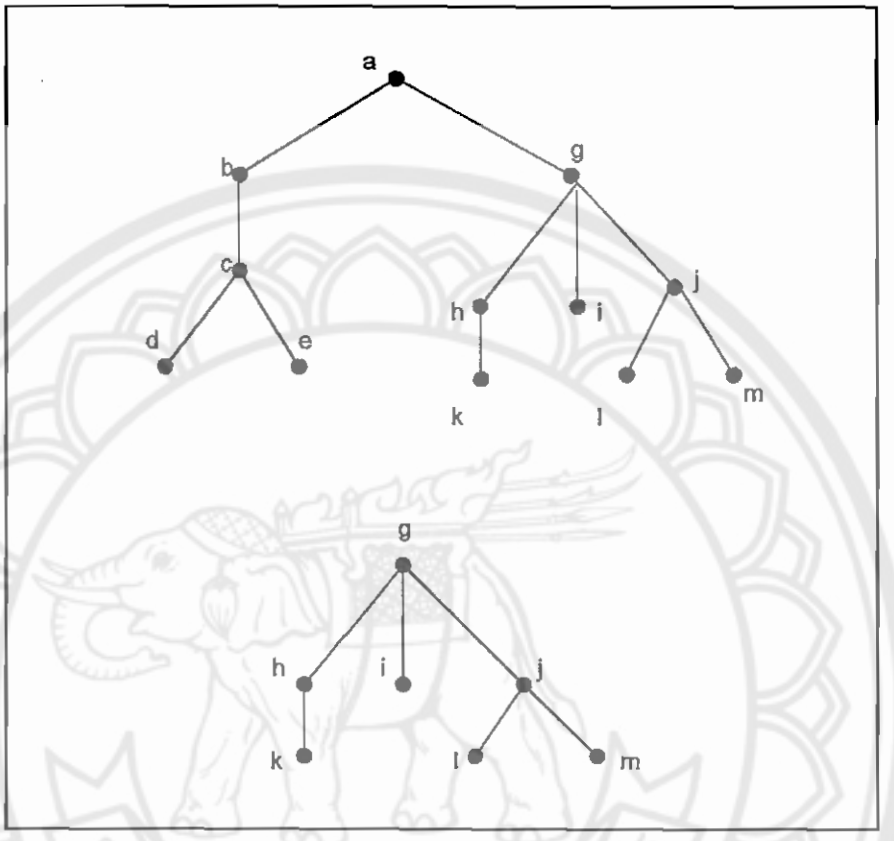
1) การออกแบบฐานข้อมูลในระดับ Conceptual การออกแบบฐานข้อมูลในระดับนี้จะเป็นการกำหนดโครงร่างเริ่มต้น ที่มีจุดมุ่งหมายเพื่ออธิบายโครงสร้างหลักๆของข้อมูลภายในระบบฐานข้อมูลโดยไม่คำนึงถึงฐานข้อมูลที่นำมาใช้ การออกแบบใน ระดับนี้มีความ สำคัญมาก เนื่องจากโครงสร้างที่ได้จากการออกแบบในขั้นตอนนี้จะถูกนำไปใช้ในขั้นตอนต่อไป โครงร่างหรือที่เรียกว่า Schema ที่ได้จากการออกแบบในขั้นตอนนี้เรียกว่า Conceptual Schema

2) การออกแบบในระดับ Logical การออกแบบในระดับนี้จะเป็นระดับที่ต่อเนื่องมาจากระดับ Conceptual กล่าวคือ การออกแบบในระดับนี้จะอาศัยโครงร่างที่จากการออกแบบในระดับ Conceptual มาปรับปรุงให้มีโครงสร้างที่เป็นไปตามโครงสร้างข้อมูลที่จะนำมาใช้งานโดยยังไม่คำนึงถึงผลิตภัณฑ์ทางด้านฐานข้อมูลที่จะนำมาใช้งาน การออกแบบในขั้นตอนนี้ต้องปรับปรุงโครงสร้างบางอย่างใน Conceptual Schema ให้สอดคล้องกับฐานข้อมูลที่จะนำมาใช้งาน การออกแบบในขั้นตอนนี้จึงต้องมีการตรวจสอบความถูกต้องของโครงร่างที่ออกแบบขึ้นกับส่วนประมวลผลต่างๆที่ออกแบบไว้รวมทั้งจะต้องแปลงโครงร่างต่างๆให้อยู่ในรูป Relation

3) การออกแบบฐานข้อมูลในระดับ Physical การออกแบบในระดับนี้ จะเป็นขั้นตอนสุดท้ายของการออกแบบฐานข้อมูลในขั้นตอนนี้ จะเป็นการปรับปรุงโครงสร้างของโครงร่างที่ออกแบบเช่นเดียวกัน แต่การปรับปรุงโครงสร้างของการออกแบบฐานข้อมูลในขั้นตอนนี้ จะเป็นการนำเอาโครงร่างที่ได้จากการออกแบบในระดับ Logical มาปรับปรุงโครงสร้างให้เป็นไปตามโครงสร้างของผลิตภัณฑ์ทางด้านฐานข้อมูลที่จะนำมาใช้งาน ผลลัพธ์ที่ได้จากการออกแบบในระดับนี้คือโครงสร้างของระบบฐานข้อมูล ที่สามารถนำไปใช้ในการสร้างตัวฐานข้อมูลจริง

2.3 การออกแบบข้อมูลให้เป็นต้นไม้และการค้นหา [7]

2.3.1 ลักษณะของข้อมูลแบบเป็นต้นไม้ (Tree)



รูปที่ 2.5 ลักษณะข้อมูลแบบต้นไม้ (Tree)

รายละเอียดของ Tree

- Parent ของจุด c คือจุด b
- children ของจุด g คือจุด h และ i,j
- siblings ของจุด h คือจุด iและ j
- ancestors ของจุด e คือจุด c,b และ a
- descendants ของจุด b คือจุด c,d และ e
- internal vertices คือจุด a,b,c,g,h และ j
- leaves คือจุด d,e,f,i,k,l และ m
- subtree rooted ที่จุด g แสดงในรูปเล็ก

2.3.2 การ ค้นหา ตามต้นไม้

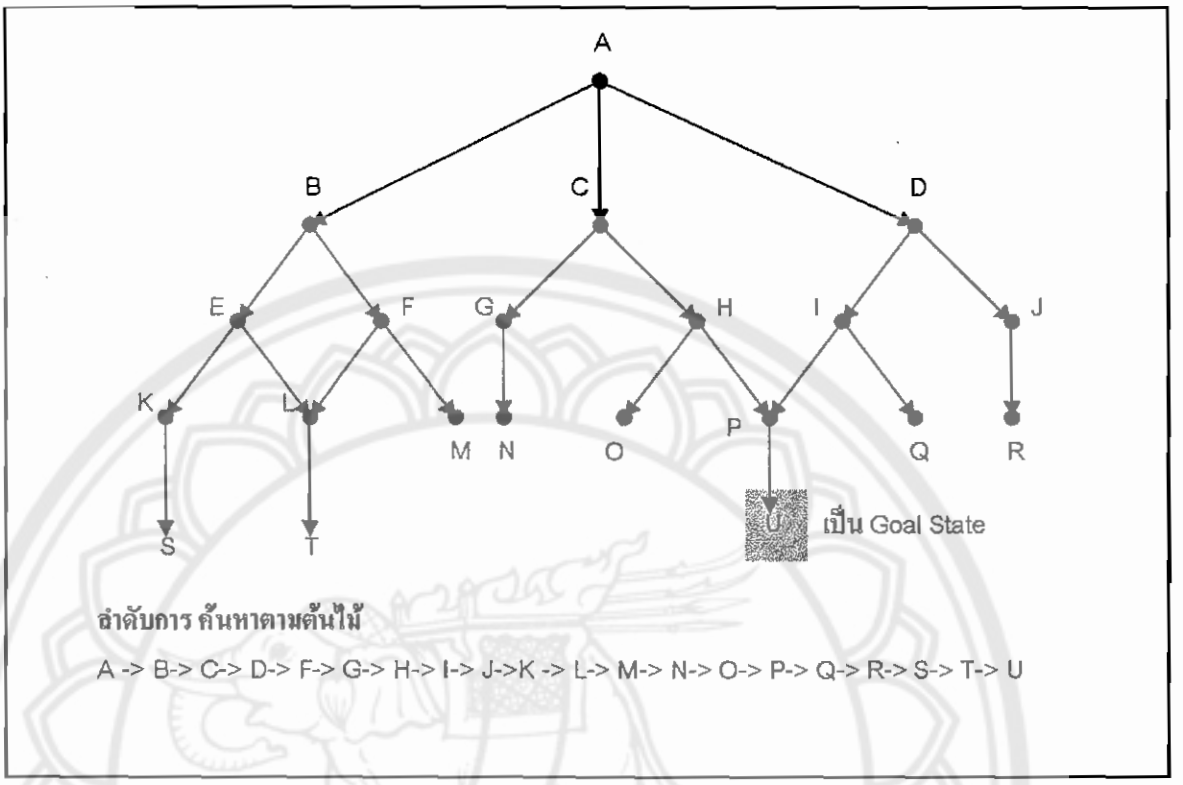
1) การ ค้นหา โดย ไม่ใช้ข้อมูลในการพิจารณาเลือกเส้นทาง (Un-Informed หรือ Blind Search) เช่น

- Breadth first search
- Depth first search

1.1) Breadth first search หรือ การ ค้นหา แนวกว้าง

```
Procedure breadth first search;
begin
        open:= [Start];
        close:= [];
        while open ≠ [] do
        begin
                remove leftmost state from open ,call it X;
                if X is a goal then return (success)
                else begin
                        generate children of X;
                        put X on closed;
                        eliminate children of X on open, or closed;
                        put remaining children on right end of open;
                end
        return (failure)  %no state left
end
```

ตัวอย่างของ Breadth first search



รูปที่ 2.6 ลักษณะการค้นหาแนวกว้าง Breadth first search

1. open =[A]; close =[]
   /* X=A */
2. open=[B,C,D]; close =[A]
   /* X=B */
3. open [C,D,E,F]; close =[B,A]
   /* X=C */
4. open [D,E,F,G,H]; close =[C,B,A]
   /* X=D */
5. open [E,F,G,H,I,J]; close =[D,C,B,A]
   /* X=E */
6. open [,F,G,H,I,J,K,L]; close =[E,D,C,B,A]
   /* X=F */
7. open [,G,H,I,J,K,L,M](as L is ready on open); close =[E,D,C,B,A]

/* X=G */

8. open [H,I,J,K,L,M,N]; close =[G,F,E,D,C,B,A]

/* X=H */

9. ทำการ ค้นหา ไปเรื่อยๆ จนกระทั่งเจอ Goal State

.

.

.

21. ในกรณีนี้ Goal State คือ โหนด U จะต้องทำการ ค้นหา ถึง 21 ครั้ง

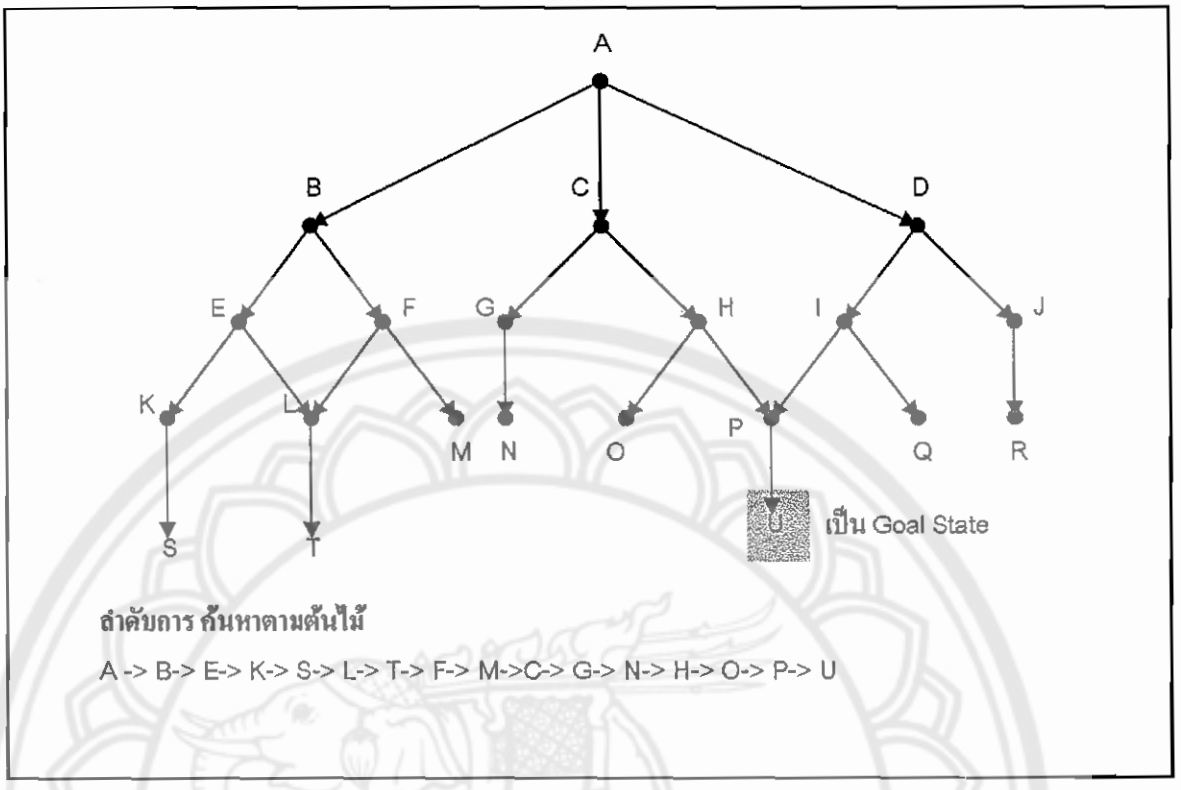
ข้อดีของ Bread first search

- ถ้ากราฟมีคำตอบอยู่จริงจะพบทางเดินที่สั้นที่สุด เป็นคำตอบแรก
- ถ้ากราฟมีความกว้างมากกว่าความลึก ให้ใช้ Breadth first search
- จะค้นหาโหนดพี่น้อง ก่อนการค้นหาโหนดลูก

1.2) Depth first search หรือ การ ค้นหา แนวลึก

```
Procedure of Depth first search;
begin
    open:= [Start];
    close:= [];
    while open ≠ [] do
    begin
        remove leftmost state from open ,call it X;
        if X is a goal then return (success)
        else begin
            generate children of X;
            put X on closed;
            eliminate children of X on open, or closed;
            put remaining children on left end of open;
        end
    return (failure)
end.
```

ตัวอย่างของ Depth first search



รูปที่ 2.7 ลักษณะการค้นหาแนวลึก (Depth first search)

1. open =[A]; close =[]
2. open=[B,C,D]; close =[A]
   /* X=A */
3. open [E,F,C,D]; close =[B,A]
   /* X=B */
4. open [K,L,F,C,D]; close =[E,B,A]
   /* X=E */
5. open [S,L,F,C,D]; close =[K,E,B,A]
   /* X=K */
6. open [L,F,C,D]; close =[S,K,E,B,A]
   /* X=S */
7. open [T.F.C.D]; close =[L,S,K,E,B,A]
   /* X=L */
8. open [F,C,D]; close =[T,L,S,K,E,B,A]

/* X=T */

9. open [M,C,D]; close =[F,T,L,S,K,E,B,A]

/* X=F */

10. open [C,D]; close =[M,F,T,L,S,K,E,B,A]

/* X=M */

11. open [G,H,D]; close =[C,M,F,T,L,S,K,E,B,A]

/* X=C */

12. ทำการค้นหาไปเรื่อยๆ จนกระทั่งเจอ Goal State

.

.

.

16. ในกรณีนี้ Goal State คือ โหนด U ซึ่งจะต้องทำการค้นหา 16 ครั้ง

ข้อดีของ Depth first search

- ใช้หน่วยความจำน้อยกว่า Breadth first search เพราะเก็บ state ใน Current path เท่านั้น
- ถ้ากราฟมีความลึกมากกว่าความกว้างให้ใช้ Depth first search
- จะค้นหาโหนดลูกก่อนการค้นหาโหนดพี่น้อง

2) การ ค้นหา โดยใช้ข้อมูลในการพิจารณาเลือกเส้นทาง (Informed หรือ Heuristics Search) เช่น

- Best first search
- Algorithm A*
- Hill Climbing search

Heuristic search

- Heuristic คือ กฎ หรือวิธีการที่ได้จากประสบการณ์ และความเชี่ยวชาญในการแก้ปัญหากฎมือขวา(rules of thumb)
- เป็นเทคนิคที่ใช้เพิ่มประสิทธิภาพของกระบวนการค้นหา โดยยอมให้ขาดความสมบูรณ์ คืออาจไม่พบคำตอบ หรือพบคำตอบที่ไม่ใช่คำตอบที่ดีที่สุด
- ใช้ Heuristics function เป็นข้อมูลเพื่อเลือก next state ที่เข้าใกล้ goal state
- Heuristics function ที่ดี ต้องไม่กระจาย state ที่ไม่จำเป็นในการนำไปสู่ goal state และไม่ทำให้การ ค้นหา หลงไปในทางที่ผิด
- ลำดับการค้นหาจะมองดูโหนดที่เป็นไปได้มากที่สุดก่อน

2.1) Best-First-Search

- รายการ Node ใน Open จะเรียงลำดับตามค่า Heuristic Function
- Open ใช้ Priority Queue
- Children Node ใหม่มีอยู่แล้วใน Open ให้ Update Node ที่มีค่า Heuristic Function ดีที่สุด
- Children Node ใหม่มีอยู่แล้วใน Close ที่มีค่า Heuristic Function ดีกว่า ให้ Update ในส่วน Open ในส่วน Close ให้ลบออก
- ใช้ $f(n) = g(n) + h(n)$

```
Procedure best_first_search;
Begin
    Open := [start];                                        % Initiallize
    Close := [];
    While open ; [] do                                      % state remain
    begin
            remove the left most state from open, call it X;
            if X = goal then return the path from Start to X
            else begin
                generate children of X;
                for each child of X do
                case    the child id not on open or closed;
                        begin
                            assign the child a heuristic value;
                            add the child to open
                        end;
                    the child is already on open;
                        if the child was reached by a shorter path
                        then give the state on open the shorter path
                    the child is already on close;
                    if the child was reached by a shorter path then
                        begin
                            remove the state from closed;
                            add the child to open
                        end;
                end;                                        % case
                put X on closed;
                re order state on open by heuristic merit (best leftmost)
            end;
return failure                                              % open in empty
end.
```

2.2) Algorithm A, admissibility, Algorithm A*

- Best-First Search ใช้ h(n) เรียกว่า Greedy Search
- Best-First Search ใช้ f(n) = g(n) + h(n) เรียกว่า Algorithm A
- Algorithm search ใดๆ ที่ค้นพบ minimal Path ของคำตอบแน่นอนเรียกว่าเป็น Admissibility เช่น Breadth-First Search หรือ Search ใดๆ ที่ใช้ f(n) = g(n) + h(n) (h(n) – 0 ; f(n) = g(n) + 0 คือ Breadth-First Search
- Algorithm A ที่ใช้ f(n) = g(n) + h(n) เรียกว่า Algorithm A
  - g(n) ฟังก์ชัน Cost จาก ถึง start state
  - h(n) ฟังก์ชันประมาณ (estimate) Cost จาก n ถึง goal state
  - f(n) ฟังก์ชันประมาณ Cost จาก Start State ผ่าน n ไปยัง goal state
  - $h(n) \leq h^*(n)$ เมื่อ $h^*(n)$ คือ actual cost หรือ true cost จาก n ถึง Goal state

การวิเคราะห์ Algorithm A*

ให้ n_0 คือ start state

ให้ n' คือ node ที่อยู่บน Optimal path ไปยัง goal

ให้ $h^*(n')$ คือ actual cost จาก n ถึง goal

ให้ $f^*(n')$ คือ Total Actual Cost จาก n_0 ถึง goal ผ่าน n'

$f(n') = g(n') + h(n')$ เนื่องจาก $h(n') \leq h^*(n')$ จะได้

$\leq g(n')+h^*(n')$ และเมื่อ $g(n') + h^*(n') = f^*(n')$ ดังนั้น

$\leq \underline{f^*(n')}$

เนื่องจาก Total actual Cost ที่ node ใดจะมีค่าเท่ากัน $f^*(n_0) = f^*(n')$ แล้วจะได้ว่า

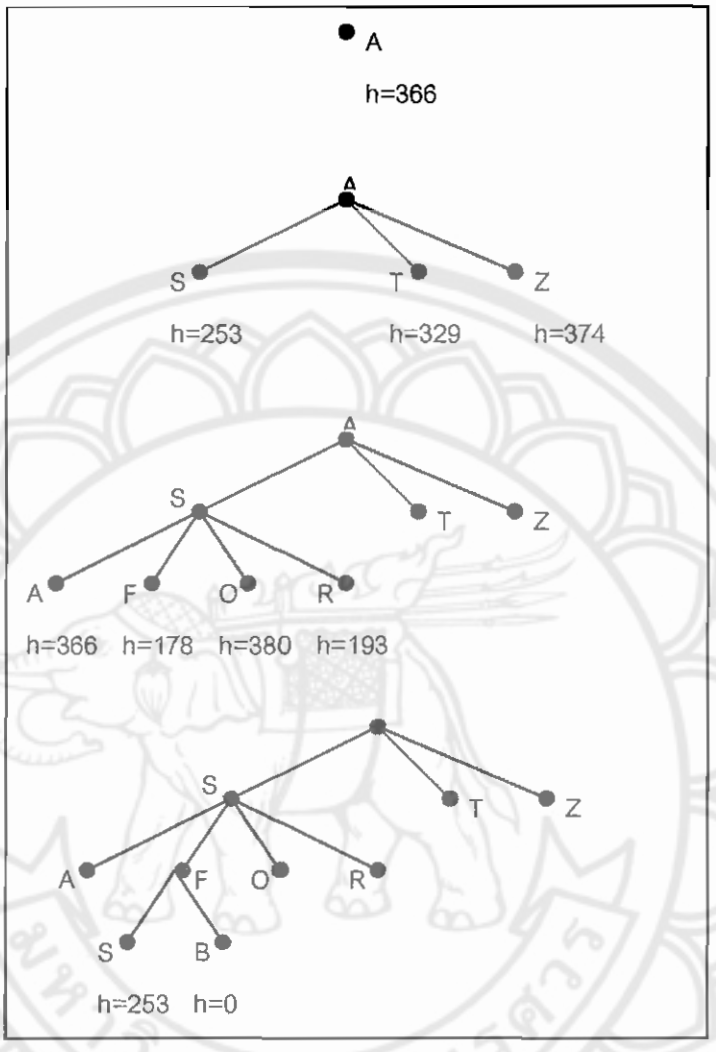
$f(n') < f^*(n_0)$

- Total Actual cost ที่ node ใดๆ จะมีค่ามากกว่า Total Cost เสมอ

มี 2 heuristics ที่เป็น Algorithm A* คือ h1 และ h2 ถ้า $h2(n) \geq h1(n)$ สำหรับทุกโหนด n ใน Search Space จะเรียกว่า h2 เป็น more informed h1 ดังนั้น search space ของ h2 จะมีขนาดเล็กกว่า search space ของ h1 (search space ของ h2 เป็น Sub Set search space ของ h1)

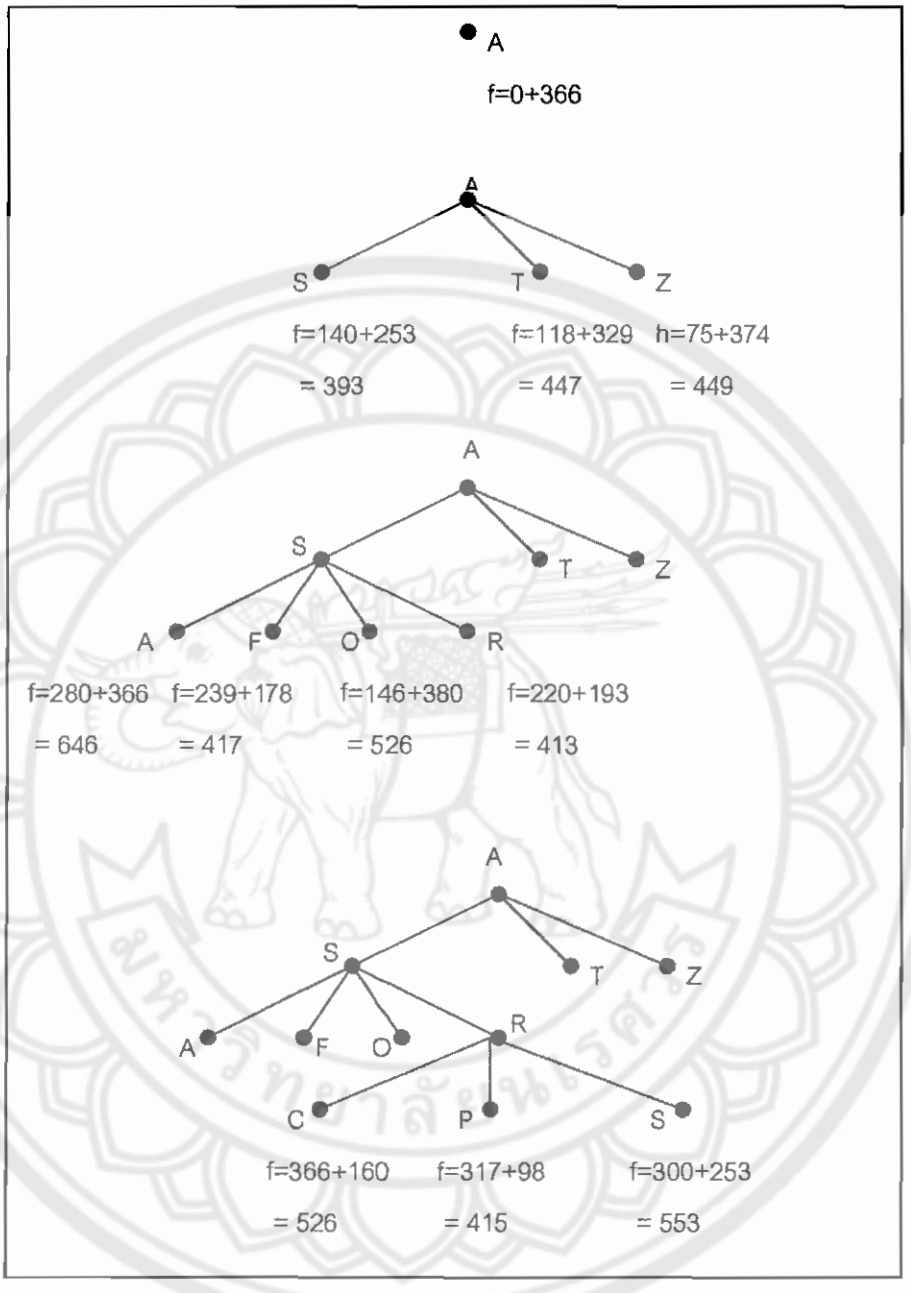
สรุปได้ว่า h(n) เป็นค่าประมาณจาก n ไป goal ยิ่งมีค่าใกล้ค่า cost ตามความจริง $h^*(n)$ เท่าไหร่ จะทำให้จำนวน node ที่ค้นหา (search Space) ยิ่งน้อยเท่านั้น

Greedy Search or Romania Map



รูปที่ 2.8 แสดงการ ค้นหา โดยใช้การเลือกเส้นทางระหว่างโหนดที่สั้นที่สุด

Algorithm A* Search on Romania Map



รูปที่ 2.9 แสดงการ ค้นหา โดยใช้ผลรวมระยะทางเลือกเส้นทางที่สั้นที่สุด

2.3) Hill-climbing search

- เกิด Loop ก็ยัง Move ได้ต่อเนื่อง
- Search ได้ทั้ง Tree และ Graph
- Heuristic Function ใช้ h(n)
- Node ที่กว่าเท่านั้นที่ถูกเลือก (Node ที่มีค่า h(n) ที่ดีกว่า h(n) ของ Node ปัจจุบัน)ถ้ามี Node ที่ดีมากกว่าหนึ่งอาจใช้การเลือกแบบซุ่ม
- ทำงานเร็วแต่ไม่ยืนยันจะได้คำตอบ

```
procedure Hill_climbing_search;
begin
    open := [start];
    close := [];
while open □ [] do
    begin
        remove the left most from open, call it X
        if X = goal then return path from Start to X
        else begin
            generate children of X
            assign the child a heurictic value
            add child more than value X to open
        end;
        put X on close
    end;
    return failure
end.
```

2.4 การใช้งานภาษา SQL [6]

SQL ย่อมาจาก Structured Query Language หมายถึงภาษามาตรฐานกลางที่ใช้ในการจัดการข้อมูลในฐานข้อมูล โดยเฉพาะอย่างยิ่งฐานข้อมูลประเภท RDBMS (Relation Database Management System) จะรู้จักภาษา SQL เป็นอย่างดี เราจะใช้ SQL เพื่อจัดการกับข้อมูลในฐานข้อมูลได้หลายอย่าง เช่น การแสดงข้อมูลจากฐานข้อมูลแบบมีเงื่อนไข, การเพิ่ม, การลบ และการนำข้อมูลจากตารางหลายๆ ตารางมาแสดงร่วมกัน ได้ เป็นต้น เป็นภาษาที่ใช้จัดการข้อมูลในฐานข้อมูลได้มีประสิทธิภาพมากที่สุด

โดยเราจะใช้ภาษา SQL เพื่อทำคิวรี (Query) ข้อมูลที่อยู่ในตาราง ในวัตถุประสงค์ที่ต่างๆกัน เช่น อาจจะต้องการข้อมูลที่มาจากตารางเดียว หรือหลายตาราง มาแสดงด้วยกันในเวลาเดียวกัน ดังนั้น การทำคิวรี จึงเป็นการสร้างตารางเสมือนขึ้นมา ประกอบด้วยข้อมูลที่มาจากตารางเดียวหรือหลายตาราง ก็ได้ เป็นตารางที่ไม่มีอยู่จริงในฐานข้อมูล เป็นมุมมองของข้อมูลในฐานข้อมูลตามที่เราต้องการ

2.4.1 โครงสร้างของภาษา SQL

ภาษา SQL ประกอบไปด้วย 3 ส่วนใหญ่ๆคือ

1. Data Definition Language (DLL) – เป็นกลุ่มคำสั่งในภาษา SQL ที่ใช้สำหรับจัดการโครงสร้างของฐานข้อมูล เช่น การสร้างฐานข้อมูล, ปรับปรุงโครงสร้างของฐานข้อมูล เป็นต้น ตัวอย่างการใช้งานกลุ่มคำสั่ง DLL นี้ก็คือ การสร้างฐานข้อมูลด้วย MS SQL Server 7.0 ก็จะมีการใช้งานคำสั่งในกลุ่ม DLL เป็นหลัก
2. Data Manitulation Language (DML) – เป็นกลุ่มคำสั่งในภาษา SQL ที่ใช้สำหรับจัดการข้อมูลในฐานข้อมูล เช่น การแสดงข้อมูลแบบมีเงื่อนไข, การลบข้อมูล, การเพิ่มข้อมูล และการแสดงข้อมูลที่มาจากตารางหลายตาราง เป็นต้น
3. กลุ่มฟังก์ชัน Aggregate Function – เป็นฟังก์ชันพิเศษของภาษา SQL ที่ทำหน้าที่เฉพาะอย่าง เช่น หาผลรวม เร็คคอร์ด, ค่าสูงสุด, ค่าต่ำสุด เป็นต้น เป็นกลุ่มฟังก์ชันที่มีประโยชน์มาก เพราะจะช่วยลดภาระให้เราไม่ต้องเขียน code จัดการเอง

สำหรับการใช้งานภาษา SQL รวมกับ Visual Basic เพื่อจัดการข้อมูลในฐานข้อมูล จะใช้งานกลุ่มคำสั่ง DML เป็นหลัก โดยในที่นี้จะอธิบายการใช้งานกลุ่มคำสั่ง DML ร่วมกับกลุ่ม Aggregate Function และกำหนดเงื่อนไขโดยใช้ตัวดำเนินการด้านต่างๆ

คำสั่งในกลุ่มของ DML จะมีคำสั่งพื้นฐานอยู่ 4 คำสั่ง คือ

- DELETE **เป็นคำสั่งที่ใช้สำหรับลบข้อมูลหรือลบเร็คคอร์ด ใดๆในตาราง**
- INSERT **เป็นคำสั่งที่ใช้สำหรับเพิ่มข้อมูลหรือเพิ่มเร็คคอร์ด ใดๆในตาราง**
- SELECT **เป็นคำสั่งที่ใช้สำหรับเลือกข้อมูลหรือแสดงเร็คคอร์ด ใดๆที่ต้องการจากตารางอาจจะมาจากตารางเดียว หรือหลายตารางก็ได้**
- UPDATE **เป็นคำสั่งที่ใช้สำหรับแก้ไขข้อมูลหรือแก้ไขเร็คคอร์ดใดๆในตาราง**

2.4.2 ตัวดำเนินการ(Operator)

2.4.2.1 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบที่น่าสนใจได้แก่

ตารางที่ 2.1 แสดงตัวดำเนินการเปรียบเทียบในภาษา SQL

ตัวดำเนินการ	ความหมาย
=	เท่ากับ (Equal)
<>	ไม่เท่ากับ (Not Equal)
<	น้อยกว่า (Less Than)
>	มากกว่า (Greater Than)
<=	น้อยกว่าหรือเท่ากับ (Less Than or Equal To)
>=	มากกว่าหรือเท่ากับ (Greater Than or Equal To)
Like	เป็นการเปรียบเทียบโดยใช้ตัวอักษรพิเศษ (Wild Card Character)เข้ามาร่วมด้วย

2.4.2.2 ตัวดำเนินการด้านตรรกะ (Logical Operator)

ตัวดำเนินการด้านตรรกะที่นิยมมีอยู่ 3 ชนิดคือ And, Or และ Not

กลุ่มฟังก์ชัน Aggregate

กลุ่มฟังก์ชัน Aggregate **เป็นฟังก์ชันที่ใช้คำนวณทางคณิตศาสตร์ โดยช่วยให้การนำเสนอการค้นหาเป็นไปอย่างมีประสิทธิภาพ ซึ่งมีฟังก์ชันพื้นฐานต่อไปนี้**

ตารางที่ 2.2 แสดงตัวดำเนินการด้านตรรกะของภาษา SQL

ชื่อฟังก์ชัน	หน้าที่
AVG()	หาค่าเฉลี่ยของฟิลด์ จาก เร็คคอร์ด ทั้งหมด
COUNT()	นับจำนวน เร็คคอร์ด
FIRST()	หาค่าแรกในฟิลด์
LAST()	หาค่าสุดท้ายในฟิลด์
MAX()	หาค่ามากที่สุดหรือค่าสูงสุด
MIN()	หาค่าน้อยที่สุดหรือค่าต่ำสุด
SUM()	หาผลรวมทั้งหมดของฟิลด์

2.4.3 ลักษณะการใช้งานของกลุ่มคำสั่ง DML

รายละเอียดการใช้งานของกลุ่มคำสั่ง DML มีทั้ง 4 คำสั่งดังนี้

- **คำสั่ง DELETE**

เป็นคำสั่งที่ใช้สำหรับลบข้อมูล หรือลบ เร็คคอร์ด ใดๆ ออกจากตาราง มีรูปแบบการใช้งาน 2 ลักษณะใหญ่คือ

รูปแบบที่ 1

DELETE FROM ชื่อตาราง WHERE เงื่อนไข

หรือ

รูปแบบที่ 2

DELETE *FROM ชื่อตาราง (ลบข้อมูลทั้งหมดในตาราง)

ชื่อตารางในที่นี้หมายถึง ชื่อของตารางที่ต้องการลบ ส่วนเงื่อนไขจะหมายถึง เงื่อนไขในการลบข้อมูลหรือลบ เร็คคอร์ด ในตารางนั้นๆ

สำหรับเครื่องหมาย * หมายถึงข้อมูลใดๆหรือข้อมูลทุก เร็คคอร์ด

ตัวอย่างการใช้งาน

```
DELETE * FROM tblSymtomp
```

เป็นการลบ เร็คคอร์ด ทั้งหมดที่อยู่ในตาราง tblSymtomp

```
DELETE * FROM tblcustomer WHERE ordervalue<10,000
```

เป็นการลบ เร็คคอร์ด ในตารางที่ชื่อว่า tblcustomer โดยมีเงื่อนไขว่าจะลบเฉพาะ เร็คคอร์ดที่ค่าในฟิลด์มีชื่อว่า ordervalue ที่ค่า < 10,000 เท่านั้น ออกจากตาราง ดังนั้น ถ้า ในเร็คคอร์ด ใดมีค่าในฟิลด์ ordervalue > 10,000 ก็จะไม่ถูกลบออกไป

เรายังสามารถใช้ตัวดำเนินการด้านตรรกะ เช่น ANDหรือOR มาใช้ระบุเป็นเงื่อนไขได้อีกด้วย เช่น

```
DELETE * FROM tblstudent WHERE gpa<1.50 OR point<10
```

เป็นการลบ เร็คคอร์ดจากตารางที่มีชื่อว่า tblstudent โดยมีเงื่อนไขว่าจะลบเฉพาะชื่อนักศึกษาที่มีผลการเรียนต่ำกว่า 1.50 (ค่าของฟิลด์ที่ชื่อว่า gpa <1.50) หรือ มีคะแนนความประพฤติต่ำกว่า 10 คะแนน (ค่าของฟิลด์ที่ชื่อว่า point <10)

- คำสั่ง INSERT

เป็นคำสั่งที่ใช้สำหรับเพิ่มข้อมูลหรือเพิ่ม เร็คคอร์ด ในตาราง ในกรณีฟิลด์เป็นข้อมูลชนิด text จะต้องใช้เครื่องหมาย '.....' กำกับฟิลด์นั้นด้วย มีรูปแบบการใช้งาน 2 ลักษณะต่อไปนี้

```
รูปแบบที่ 1 INSERT INTO tablename(field1,field2,...) VALUE(value1,'value2',...)
รูปแบบที่ 2 INSERT INTO tablename2
        SELECT * FROM tablename1 WHERE criteria
```

กรณีนี้เราสมมติให้ฟิลด์ 2 เป็นข้อมูลชนิด text

ตัวแปร tablename หมายถึงชื่อตารางที่ต้องการเพิ่ม เร็คคอร์ด เข้าไป

ตัวแปร tablename1 หมายถึง เลือกข้อมูลจากตารางที่ชื่อว่า tablename1 ตามเงื่อนไขที่ระบุไว้ในตัวแปร criteria แล้วนำมาเพิ่มที่ตาราง tablename2

ตัวแปร field1-fieldn หมายถึงชื่อของฟิลด์ต่างๆที่อยู่ในตารางที่เราต้องการเพิ่มข้อมูล

ตัวแปร value1-valuen หมายถึง ค่าของฟิลด์ที่จะเพิ่มเข้าไปโดยที่คุณจะต้องระบุค่าให้ตรงกับฟิลด์ด้วย

ตัวแปร criteria หมายถึง เงื่อนไขในการดึงข้อมูลจากตาราง tablename1

<u>ตัวอย่างการใช้งาน</u>

```
INSERT INTO tblSymtomp (Node, Name,Description,Treatment)
    VALUES(01_05,'มีไข้เกิน7วัน','มีไข้เกิน7วันหรือหนาวสั่นมาก',รักษาภายใน 3 วัน)
```

จากตัวอย่างเป็นการเพิ่มข้อมูลเข้าไปในตารางที่ชื่อว่า tblSymtomp ประกอบไปด้วยฟิลด์ Node, Name,Description,Treatment โดยที่ให้ใส่ค่า *01_05* ลงในฟิลด์ Node, *มีไข้เกิน 7 วัน* ลงในฟิลด์ Name , *มีไข้เกิน7วันหรือหนาวสั่นมาก* ลงในฟิลด์ Description และใส่ *รักษาภายใน 3 วัน* ในฟิลด์ Treatment ฟิลด์ทั้ง 3 เป็นชนิด memo จึงต้องใส่เครื่องหมาย ' กำกับไว้

```
INSERT INTO tblSymtomp (Node, Name,Description,Treatment)
  SELECT * FROM tblSymtomp_2 WHERE Treatment='รักษาด่วน'
```

ต้องการดึงข้อมูลรายชื่อนักศึกษาจากตารางที่ชื่อว่า tblsymtomb_2 เฉพาะโรคหรืออาการที่ต้องมีการรักษาด่วน เพิ่มเข้าในตาราง tblSymtomp โดยให้ใส่ค่าตามลำดับของฟิลด์ (Node, Name,Description,Treatment

ข้อควรระวังในการใช้คำสั่ง INSERT

- กรณีที่ 1 ใส่ค่าในฟิลด์ที่เป็น Primary Key ซ้ำค่ากับเดิมที่มีอยู่แล้ว หรือฟิลด์ที่เป็น Primary Key เราไม่ได้กำหนดค่าให้มีค่าเท่ากับ Null
- กรณีที่ 2 ค่าที่ใส่เข้าไปซ้ำกับ เร็คคอร์ด ที่มีอยู่แล้วในฐานข้อมูล

ทั้ง 2 กรณีส่งผลให้ INSERT ไม่มีการเพิ่ม เร็คคอร์ต นั้นๆเข้าไปในตาราง

ตารางที่ 2.3 ตาราง Symtomp

	Node	Name	Description	Treatment
▶	01	=> ไข้	=> ตัวร้อน อุณหภูมิร่างก	*** ถ้าอาการไม่ชัดเจน (
	01_01	=> ไม่ค่อยรู้สึกตัว ?	=> ไม่ค่อยรู้สึกตัว ? ปาด	*** รักษาด่วน
	01_01_01	=> คอแข็ง ? หรือกร	=> คอแข็ง ? หรือกระหน	*** รักษาด่วน
	01_01_02	=> เคยเข้าไปในดงม	=> เคยเข้าไปในดงมาลา	*** รักษาด่วน
	01_01_03	=> เคยถูกสุนัขหรือเ	=> เคยถูกสุนัขหรือแมวก	*** รักษาด่วน
	01_02	=> แขนขาอ่อนแรง	=> แขนขาอ่อนแรง หรือ	*** รักษาภายใน 24 ชั่ว
	01_03	=> มีภาวะช็อก (เหงื่	=> มีภาวะช็อก (เหงื่อออ	*** รักษาด่วน ให้น้ำเกลื
	01_04	=> มีไข้นานเกิน 1 เ	=> มีไข้นานเกิน 1 เดือน	*** รักษาภายใน 3 วัน
	01_04_01	=> ไอ และน้ำหนักล	=> ไอ และน้ำหนักลดฮา	*** รักษาภายใน 3 วัน
	01_04_02	=> ปวดข้อนิ้วมือ 2 ข	=> ปวดข้อนิ้วมือ 2 ข้าง?	*** รักษาภายใน 3 วัน
	01_04_03	=> จับไข้หนาวสั่น วั	=> จับไข้หนาวสั่น วันเว้น	*** รักษาภายใน 3 วัน
	01_04_04	=> มีจุดแดงที่เยื่อบุต	=> มีจุดแดงที่เยื่อบุตา /	*** รักษาภายใน 3 วัน
	01_04_05	=> มีจุดแดงจ้ำเขียว	=> มีจุดแดงจ้ำเขียวขึ้นต	*** รักษาภายใน 3 วัน
	01_05	=> มีไข้เกิน 7 วัน หร	=> มีไข้เกิน 7 วัน หรือหา	*** รักษาภายใน 3 วัน (

ถ้ามีการ Insert ฟิลด์ข้อมูลเข้ามา โดยฟิลด์นั้นมี key เหมือนกับ Primarykey จะไม่สามารถ Insert ข้อมูลเข้ามาได้ เนื่องจากว่าฟิลด์ที่เป็น Primary Key จะต้องมีค่าไม่ซ้ำกัน

ค่อมาสมมติว่า ใช้คำสั่ง INSERTอีกครั้งดังนี้

```
INSERT INTO tblSymtomp (Node, Name,Description,Treatment)
  VALUES(01,'ไข้','ตัวร้อนอุณหภูมิร่างกายสูง...','ถ้าอาการไม่ชัดเจนให้ปรึกษาแพทย์')
```

จะเห็นได้ว่าฟิลด์ที่ต้องการเพิ่มเข้าไปมีอยู่แล้วในตาราง ดังนั้นคำสั่งนี้จึงไม่มีการเพิ่มเร็คคอร์ดแต่อย่างใด

- คำสั่ง **SELECT**

ใช้สำหรับเลือกหรือดึงข้อมูล (Retrieve Data) ที่เราต้องการจากตารางที่ระบุไว้ เป็นคำสั่งที่มีความยืดหยุ่นสูงมาก เพราะว่าเงื่อนไขในการนำข้อมูลออกมาจากตารางมีมากมายหลายลักษณะแต่มี**รูปแบบการใช้งานพื้นฐาน มีอยู่ 2 ลักษณะคือ**

```
SELECT * FROM ชื่อตาราง
```

หรือ

```
SELECT ฟิลด์ที่ 1, ฟิลด์ที่ 2 ,...,ฟิลด์ที่ n FROM ชื่อตาราง WHERE เงื่อนไข
```

โดย ชื่อตาราง จะหมายถึง ชื่อตารางที่ต้องการดึงข้อมูล

ส่วนตัวแปร ฟิลด์ที่ 1, ฟิลด์ที่ 2 ,...,ฟิลด์ที่ n จะหมายถึง ชื่อฟิลด์ที่ต้องการดึงข้อมูล ถ้ามีมากกว่า 1 ฟิลด์ จะใช้เครื่องหมาย , คั่นระหว่างฟิลด์

สำหรับ เงื่อนไข หมายถึง เงื่อนไขในการดึงข้อมูล อาจเป็นเงื่อนไขทางคณิตศาสตร์ หรือเป็นคำสั่ง SELECT ซ้อนอยู่ข้างในก็ได้

ตัวอย่างการใช้งาน

```
SELECT * FROM tblSymtomp
```

เป็นการเลือกข้อมูลทุกเร็คคอร์ดจากตารางที่ชื่อว่า tblSymtomp

```
SELECT Node, Name,Treatment FROM tblSymtomp
```

เป็นการเลือกข้อมูลจากฟิลด์ที่เราต้องการ ใช้ในกรณีที่เราไม่ต้องการแสดงข้อมูลทุกฟิลด์

```
SELECT * FROM tblStudent WHERE Treatment ='รักษาภายใน 3 วัน'
```

เป็นการดึงเร็คคอร์ดเฉพาะที่มีค่าในฟิลด์ Treatment มีค่าเท่ากับ *รักษาภายใน 3 วัน*

- **การใช้งานคำสั่ง SELECT แบบมีเงื่อนไข**

เป็นการใช้งานคำสั่ง SELECT ร่วมกับตัวดำเนินการอื่นๆ เพื่อเป็นเงื่อนไขในการแสดงข้อมูล รวมถึงการใช้งานร่วมกับกลุ่มฟังก์ชัน Aggregate ด้วย การใช้งานคำสั่ง SELECT แบบมีเงื่อนไขให้ดู**ที่ตาราง ชื่อว่า tblSymtomb**

<u>ตัวอย่างการใช้งาน</u> ถ้าต้องการดูเฉพาะอาการของโรคที่ที่ต้องรักษาด่วนเท่านั้น

```
SELECT  Name, Description,Salary FROM tblSymtomp WHERE Treatment = 'รักษาด่วน';
```

ผลที่ได้คือ

	Node	Name	Description	Treatment
	01_01	=> ไม่ค่อยรู้สึกตัว ?	=> ไม่ค่อยรู้สึกตัว ? ปวดศีรษะมา	*** รักษาด่วน
	01_01_01	=> คอแข็ง ? หรือกร	=> คอแข็ง ? หรือกระหม่อมโป่งตึ	*** รักษาด่วน
	01_01_02	=> เคยเข้าไปในดงม	=> เคยเข้าไปในดงมาลาเรีย หรือ	*** รักษาด่วน
	01_01_03	=> เคยถกสนับหรือเ	=> เคยถกสนับหรือแมวกัดหรือข่ว	*** รักษาด่วน

ถ้าต้องการดูเฉพาะ Treatment *รักษาด่วน* และ*คอแข็ง*....

```
SELECT Description,Treatment FROM tblSymtomp
        WHERE Treatment='รักษาด่วน' AND Description='คอแข็ง...';
```

ผลที่ได้คือ

	Node	Name	Description	Treatment
	01_01_01	=> คอแข็ง ? หรือกร	=> คอแข็ง ? หรือกระหม่อมโป่งตึ	*** รักษาด่วน

ตัวอย่างต่อไป เป็นการสร้างเงื่อนไขค้นหาที่ซับซ้อนมากยิ่งขึ้นเป็นการเปรียบเทียบจำนวน ซึ่งไม่มีในโครงงานนี้จึงขอยกตัวอย่างที่นอกเหนือจากโครงงานนี้มาอธิบายภาษา SQL เช่น ต้องการหาฟิลด์ Salary >=15000 เป็นอันดับแรกก่อน จากนั้นจึงนำเงื่อนไขที่ระบุไว้ไปจำกัดรายการเร็คคอร์ดที่ได้ให้เหลือแต่เร็คคอร์ด

```
SELECT Code,FirstName,LastName,Salary
FROM tblEmployee
WHERE Salary >= 15000 AND Dept='Teacher'OR Dept='DBA'
```

ผลลัพธ์ที่ได้คือ

ตารางที่ 2.4 ผลการค้นหาโดยใช้เงื่อนไขเปรียบเทียบจำนวนของภาษา SQL

FirstName	LastName	Salary	Dept
น.ส.ปิยพร	เพชรพง	20000	Teacher
น.ส.ธิติมา	ยุราวรรณ	17500	Teacher
นาย พิทยา	โชติแสงชัยฤกษ์	15000	Teacher
นาย ณัฐพล	กิจประชา	16000	DBA

หรือใช้คำสั่ง BETWEEN เพื่อจำกัดขอบเขตก็ได้ เช่น ต้องการรายชื่อเฉพาะที่มีเงินเดือนอยู่ระหว่าง 13000-15000 จะพบว่าเงินเดือนที่เท่ากับ 13000 และ 15000 จะถูกรวมเข้ามาด้วย

```
SELECT FirstName,LastName,Dept,Salary
FROM tblEmployee
WHERE Salary BETWEEN 13000 AND 15000;
```

ต้องการรายชื่อเฉพาะที่มีเงินเดือนอยู่ระหว่าง 13,000 – 15,000 จะพบว่ามีเงินเดือนเท่ากับ 13,000 หรือ 15,000 รวมเข้าด้วย

FirstName	LastName	Dept	Salary
น.ส. สุรัตน์	จงประสบโชค	Teacher	14000
นายอดิเทพ	ตรังคานนท์	Staff	12000
นายพิทยา	โชติแสงชัยพฤกษ์	Manager	15000
นายเอกพงศ์	สุรพันธ์พงศ์	Staff	13000
น.ส.สาวินี	ไชยทอง	DBA	15000
น.ส.นุสรา	น้อยสุพรรณ	Teacher	15000

ตารางที่ 2.5 ผลการค้นหาโดยใช้เงื่อนไข BETWEEN ของภาษา SQL

ในทางกลับกัน ถ้าไม่ต้องการให้อยู่ในช่วงที่กำหนดไว้ ให้ใช้ตัวดำเนินการ NOT เข้าช่วย

```
SELECT FirstName, LastName, Dept, Salary
From tblEmployee
WHERE Salary NOT BETWEEN 13000 AND 15000
```

ผลที่ได้ก็คือ เร็คคอร์ดที่อยู่ในช่วงที่กำหนดไว้จะถูกตัดออกทั้งหมด

ตารางที่ 2.6 ผลการค้นหาโดยใช้เงื่อนไข BETWEEN ร่วมกับ NOT ของภาษา SQL

FirstName	LastName	Dept	Salary
น.ส.ปิยพร	เพ็ชรพงศ์	Manager	20000
น.ส.ธิติมา	ยุราวรรณ	Admin	17500
น.ส.ชลธิชา	สีดา	DBA	16000
นาย ณัฐพล	กิจประชา	Admin	16000

กรณีที่ต้องการจำกัดจำนวนเร็คคอร์ดที่แสดงออกมา ให้ใช้คำสั่ง TOP

```
SELECT Top 10 * FROM tblSymtomp;
```

เป็นการเลือกทุกฟิลด์ จากตาราง tblSymtomp จำกัดให้แสดง 10 เร็คคอร์ดแรกเท่านั้น ผลที่ได้คือ

ตารางที่ 2.7 ผลการค้นหาโดยจำกัดจำนวนเร็คคอร์ดของภาษา SQL

	Node	Name	Description	Treatment
	01	=> ไข้	=> ตัวร้อน อุณหภูมิร่างกายสูงกว่า	*** ถ้าอาการไม่ชัดเจน เ
	01_01	=> ไม่ค่อยรู้สึกตัว ?	=> ไม่ค่อยรู้สึกตัว ? ปวดศีรษะมา	*** รักษาด่วน
▶	01_01_01	=> คอแข็ง ? หรือกร	=> คอแข็ง ? หรือกระหม่อมโป่งต็	*** รักษาด่วน
	01_01_02	=> เคยเข้าไปในดงม	=> เคยเข้าไปในดงมาลาเรีย หรือ	*** รักษาด่วน
	01_01_03	=> เคยถูกสุนัขหรือแ	=> เคยถูกสุนัขหรือแมวกัดหรือข่ว	*** รักษาด่วน
	01_02	=> แขนขาอ่อนแรง	=> แขนขาอ่อนแรง หรืออัมพาตเ	*** รักษาภายใน 24 ชั่ว
	01_03	=> มีภาวะช็อก (เหงื่	=> มีภาวะช็อก (เหงื่อออก ตัวเย็น	*** รักษาด่วน ให้น้ำเกลื
	01_04	=> มีไข้นานเกิน 1 เ	=> มีไข้นานเกิน 1 เดือน ?	*** รักษาภายใน 3 วัน
	01_04_01	=> ไอ และน้ำหนักล	=> ไอ และน้ำหนักลดฮวบ ?	*** รักษาภายใน 3 วัน
	01_04_02	=> ปวดข้อนิ้วมือ 2 ข	=> ปวดข้อนิ้วมือ 2 ข้าง? ผมร่วง ?	*** รักษาภายใน 3 วัน

ถ้าต้องการเรียงลำดับข้อมูลที่จะแสดงออกมาด้วย ให้ใช้คำสั่ง ORDER BY ร่วมกับชื่อฟิลด์ที่คุณต้องการใช้เป็นเงื่อนไขในการเรียงลำดับ

ถ้าคุณไม่ระบุ จะเป็นการเรียงลำดับจากค่าน้อยไปหามาก หรือถ้าเป็นข้อความก็จะเป็นการเรียงลำดับตามตัวอักษร ถ้าถ้าในกรณีที่ต้องการเรียงลำดับจากค่ามากไปหาน้อย ให้ใช้คำสั่ง DESE กำกับไว้ด้วย

```
SELECT * FROM tblEmployee ORDER BY DateStart;
```

เป็นการเรียงลำดับจากค่าน้อนไปหามาก โดยใช้ฟิลด์ Datastart เป็นเงื่อนไข ผลที่ได้คือ

ตารางที่ 2.8 ผลการค้นหาโดยเรียงลำดับ DateStart จากน้อยไปหามาก

Code	FirstName	LastName	Salary	Dept	DateStart
007	นายณัฐพล	กิจประชา	16000	Admin	18/05/1996
003	น.ส.ธิดิมา	ยุราวรรณ	17500	Admin	16/12/1996
009	น.ส.สาวินี	ไชยทอง	15000	DBA	07/07/1998
010	น.ส.นุสรา	น้อยสุพรรณ	15000	Teacher	27/09/1998
002	น.ส. สุรัตน์	จงประสบโชค	14000	Teacher	10/03/1999
005	นายอดิเทพ	ตรังคานนท์	12000	Staff	15/04/1999
004	น.ส.ชลธิชา	สีดา	16000	DBA	24/06/1999
006	นายพิทยา	โชติแสงชัยพฤกษ์	15000	Manager	20/07/1999
008	นายเอกพงศ์	สุรพันธ์พงศ์	13000	Staff	18/02/2000
001	น.ส.ปิยพร	เพ็ชรพงศ์	20000	Manager	12/04/2000

กรณีที่ต้องการเรียงจากค่ามากไปหาน้อย

```
SELECT *  FROM tblEmployee ORDER BY DateStart DESC;
```

กรณีที่ข้อมูลแสดงออกมาซ้ำกันหลายๆ เร็คคอร์ด ถ้าคุณต้องการแสดงเพียงเร็คคอร์ดเดียว ให้ใช้คำสั่ง DISTINCE เช่น

```
SELECT Salary FROM tblEmpolyee;
```

เป็นการเลือกเฉพาะฟิลด์เงินเดือน ผลที่ได้คือ

ตารางที่ 2.9 ผลการค้นหาโดยเลือกเฉพาะบางฟิลด์ของภาษา SQL

Salary
20000
14000
17500
16000
12000
15000
16000
13000
15000
15000

จากผลที่ได้จะพบว่า มีค่าซ้ำกันอยู่คือ 15000 และ 16000 แต่ถ้าคุณใช้คำสั่ง DISTINCE

```
SELECT DISTINCE Salary FROM tblEmpolyee;
```

ผลที่ได้คือ

ตารางที่ 2.10 ผลการค้นหาโดยเลือกเฉพาะบางฟิลด์ร่วมกับคำสั่ง DISTINCE ของภาษา SQL

Salary
20000
14000
17500
16000
12000
15000
13000

กรณีที่ใช้ร่วมกับกลุ่มฟังก์ชัน Aggregate เช่น ฟังก์ชัน MAX ()

```
SELECT MAX (Salary) FROM tblEmpolyee;
```

เป็นการหาฟิลด์ Salary ที่มีค่าสูงสุด ผลที่ได้คือ

Salary
20000

ส่วนฟังก์ชัน Aggregate อื่นๆ จะมีลักษณะการใช้งานเช่นเดียวกับฟังก์ชัน MAX() ผลที่ได้จะเป็นไปตามหน้าที่ของแต่ละฟังก์ชัน

- **การใช้งานคำสั่ง SELECT ร่วมกับคำสั่ง LIKE**

คำสั่ง LIKE เป็นคำสั่งที่ใช้สำหรับกำหนดเงื่อนไขที่เกี่ยวข้องกับข้อมูลชนิดข้อความ (TEXT หรือ String) ปกติแล้วจะใช้ร่วมกับตัวอักษรพิเศษ ที่เรียกว่า Wild Card Characters เพื่อระบุเป็นเงื่อนไขให้กับคำสั่ง LIKE ในการจำกัดข้อมูลเป็นข้อความ

ตารางที่ 2.11 แสดงตัวอักษรพิเศษของภาษา SQL

ลักษณะอักษร	ความหมาย
*	ใช้แทนตัวอักษรใดๆ โดยไม่มีการจำกัดจำนวนตัวอักษร เช่น ca* หมายถึง ข้อความใดก็ตามที่มีตัวอักษร ca จะถือว่าใช่ทั้งหมด เช่น cat, catch, Canada cash, case เป็นต้น จะเห็นได้ว่า ไม่จำกัดตัวอักษรใน 1 ค่า แต่ขอให้มีตัวอักษร ca เท่านั้น
?	ใช้แทนตัวอักษรใดๆ เช่นกัน แต่จำกัดตัวอักษร เช่น ca?? หมายถึง cash, case, card เป็นต้น
%	มีความหมายเช่นเดียวกับ *
#	ใช้กับข้อมูลชนิดตัวเลข แทน 1 ตัวอักษร เช่น 12# หมายถึง 123,124,125 เป็นต้น .
[]	ใช้กำหนดขอบเขตข้อความ เช่น [a-d] หมายถึงตัวอักษร a ถึง d เท่านั้น
[!]	ความหมายตรงกับข้ามกับ [] เช่น [!a-d] หมายถึงตัวอักษร e ถึง z

ตัวอย่างการใช้งานคำสั่ง LIKE ร่วมกับ Wild Card Characters เช่น ต้องการแสดงรายละเอียดของพนักงานที่ชื่อของเขามีตัวอักษร ส. อยู่ภายใน

```
SELECT *FROM tblSymtomp
Where FirstName LIKE "%ส";
```

ผลที่ได้คือ

ตารางที่ 2.12 ผลการค้นหาโดยใช้คำสั่ง LIKE ร่วมกับ Wild Card Characters

Code	FirstName	LastName	Salary	Dept	DateStart
009	น.ส.สาวินี	ไชยทอง	15000	DBA	07/07/1998
002	น.ส. สุรัตน์	จงประสบโชค	14000	Teacher	10/03/1999
010	น.ส.นุสรา	น้อยสุพรรณ	15000	Teacher	27/09/1998

ในทางกลับกัน ถ้าไม่ต้องการให้มีชื่อ ส อยู่ด้วย ให้ใช้คำสั่งดังนี้

```
SELECT *FROM tblEmployee
Where FirstName NOT LIKE "%ส";
```

2.5 การใช้งาน Microsoft Visual Basic [6]

2.5.1 ประวัติความเป็นมาของ Visual Basic

Visual Basic เป็นภาษาคอมพิวเตอร์ (Programming Language) ที่พัฒนาโดยบริษัท Microsoft ซึ่งเป็นบริษัทยักษ์ใหญ่ที่สร้างระบบปฏิบัติการ Windows 98/Me และ Window NT/2000 ที่เราใช้กันอยู่ในปัจจุบัน โดยตัวภาษาเองมีรากฐานมาจากภาษา Basic ซึ่งย่อมาจาก Beginner's All Purpose Symbolic Instruction ถ้าแปลให้ได้ความหมายก็คือ "ชุดคำสั่งหรือภาษาคอมพิวเตอร์สำหรับผู้เริ่มต้น" ภาษา Basic จุดเด่นคือผู้ที่ไม่มีพื้นฐานการเขียนโปรแกรมเลยก็สามารถเรียนรู้และนำไปใช้งานได้อย่างง่ายดายและรวดเร็ว เมื่อเทียบกับการเรียนภาษาคอมพิวเตอร์อื่นๆ เช่น ภาษาซี (C), ปาสคาล (Pascal), ฟอร์แทรน (Fortran) หรือ แอสเซมบลี (Assembler)

Microsoft ได้พัฒนาโปรแกรมภาษา Basic มานานนับสิบปี ตั้งแต่ภาษา Mbasic (Microsoft Basic), BASICA (Basic Advanced), GWGASIC และ QuickBasic ซึ่งได้ติดตั้งมากับระบบปฏิบัติการ MS DOS ในที่สุดโดยใช้ชื่อว่า QBASIC โดยแต่ละเวอร์ชันที่ออกมานั้นได้มีการพัฒนาและเพิ่มคำสั่งต่างๆ เข้าไปโดยตลอด ในอดีตโปรแกรมภาษาเหล่านี้ล้วนทำงานใน Text Mode คือเป็นตัวอักษรล้วนๆ ไม่มีภาพกราฟิกสวยงามแบบระบบ Windows อย่างในปัจจุบัน จนกระทั่งเมื่อระบบปฏิบัติการ Windows ได้รับความนิยมสูงและเข้ามาแทนที่ DOS ไมโครซอฟท์ก็เล็งเห็นว่าโปรแกรมภาษาใน Text Mode นั้นคงถึงการที่หมดสมัย จึงได้พัฒนาปรับปรุงภาษา Basic ของตนเองออกมาใหม่เพื่อสนับสนุนการทำงานในระบบ Windows ทำให้ Visual Basic ถือกำเนิดขึ้นมาตั้งแต่บัดนั้น

Visual Basic เวอร์ชันแรกคือเวอร์ชัน 1.0 ออกสู่สายตาประชาชนตั้งแต่ปี1991 โดยในช่วงแรกนั้นยังไม่มีความสามารถต่างจากภาษา QBASIC มากนัก แต่จะเน้นเรื่องเครื่องมือที่ช่วยในการเขียนโปรแกรมบน Window ซึ่งปรากฏว่า Visual Basic ได้รับความนิยมและประสบความสำเร็จเป็นอย่างดี ไมโครซอฟท์จึงพัฒนา Visual Basic ให้ดีขึ้นเรื่อยๆ ทั้งในด้านประสิทธิภาพ ความสามารถ และเครื่องมือต่างๆ เช่น เครื่องมือตรวจสอบแก้ไขโปรแกรม (debugger) สภาพแวดล้อมของการพัฒนาโปรแกรม การเขียนโปรแกรมแบบหลายวินโดว์ย่อย (MDI) และอื่นๆอีกมากมาย

สำหรับ Visual Basic ในปัจจุบันคือเวอร์ชัน 6.0 ซึ่งออกมาในปี 1998 ได้เพิ่มความสามารถในการเขียนโปรแกรมติดต่อกับเครือข่ายอินเตอร์เน็ต การเชื่อมต่อกับระบบฐานข้อมูล รวมทั้งปรับปรุงเครื่องมือและการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) ให้สมบูรณ์ยิ่งขึ้น พร้อมทั้งเพิ่มเครื่องมือต่างๆ อีกมากมายที่ทำให้ใช้ง่ายและสะดวกขึ้นกว่าเดิม

2.5.2แนวคิดของ OOP

โอโอพี (OOP) หรือ Object Oriented Programming เป็นแนวคิดในการเขียนโปรแกรมแบบหนึ่ง ซึ่งนิยามว่าเป็น การเขียนโปรแกรมเชิงวัตถุ

ถ้าเราไม่มองในแง่มุมของการเขียนโปรแกรมเพียงอย่างเดียว ให้เรามองภาพรวมมองไปในสิ่งรอบๆ ตัวเรา เราบอกได้ว่า แนวคิดของ OOP ก็คือ "ธรรมชาติของวัตถุ" หมายความว่า OOP จะมองสิ่งแต่ละสิ่งเป็น "วัตถุชิ้นหนึ่ง" มันจะยาวหรือสั้น หรือมีสีอะไรก็ตาม มันก็คือวัตถุชิ้นหนึ่งเหมือนกัน ซึ่งวัตถุแต่ละสิ่งนั้น ย่อมมีคุณสมบัติที่ต่างกัน แต่อาจมีอย่างที่เหมือนกัน เรายังสามารถคิดได้อีกว่า "วัตถุแต่ละอย่างนั้น ต่างก็มีลักษณะและวิธีการใช้เป็นตัวของมันเอง" ซึ่งหมายความว่า วัตถุแต่ละชนิดหรือแต่ละชิ้น ต่างก็มีรูปร่างลักษณะ และการใช้งาน ที่แตกต่างออกไป เราเรียกคุณลักษณะของวัตถุว่าคุณลักษณะของวัตถุ (Attribute) และเรียกการใช้งานของวัตถุว่า เมธอด (Method) เช่น

"ดินสอเป็นวัตถุที่มีลักษณะที่เรียวยาว, ภายในเป็นไส้ถ่านใช้สำหรับเขียน การใช้งานดินสอทำได้ โดยใช้มือจับและเขียนบนวัตถุรองรับ"

จากประโยคข้างต้น เราสามารถจับใจความได้ว่า คุณลักษณะของวัตถุ (Attribute) ก็คือ " ยาวเรียว, ภายในเป็นไส้ถ่าน" ส่วนการใช้งาน (Method) ก็คือ "ใช้มือจับและเขียนลงบนวัตถุรองรับ"

จากตัวอย่างข้างต้น คราวนี้เราสรุปได้แล้วว่า ถ้าเกิดวัตถุใดมีลักษณะยาวเรียว, มีไส้เป็นถ่าน เมื่อใช้งานต้องใช้มือจับและเขียนบนวัตถุรองรับ เราก็สามารถบอกได้ว่าวัตถุนั้นคือ "ดินสอ"

2.5.3 รูปแบบการติดต่อกับฐานข้อมูลด้วย Visual Basic

ในการติดต่อกับฐานข้อมูล โดยปกติแล้ว VB จะเชื่อมโยงผ่านทาง Database Engine ที่เรียกว่า JET Engine จึงอาจกล่าวได้อีกนัยหนึ่งว่า JET Engine คือ ไดรเวอร์ชนิดหนึ่ง ซึ่งทำหน้าที่เป็นตัวเชื่อมโยงให้ VB สามารถติดต่อกับฐานข้อมูลได้นั้นเอง โดยที่ฐานข้อมูลหลัก (Default) ที่ VB รู้จักเป็นอย่างดีก็คือ MS Access แต่ Visual Basic ก็สามารถติดต่อกับฐานข้อมูลได้ทุกชนิดเช่นกัน โดยอาศัยเทคโนโลยีหลายๆอย่าง

สำหรับฐานข้อมูลที่ใช้ในการทำโครงงานนี้ จะใช้ฐานข้อมูลของ MS Access 2000 Thai Edition ซึ่งการติดต่อกับฐานข้อมูลใน Visual Basic จะแยกออกเป็น 4 ประเภทใหญ่ๆ คือ

แบบที่ 1 - ติดต่อโดยอาศัยคอนโทรลด้านฐานข้อมูล

แบบที่ 2 - ติดต่อโดยใช้ออบเจ็กต์ Data Access Object (DAO)

แบบที่ 3 - ติดต่อผ่านทาง ODBC โดยตรง (ODBC Direct)

แบบที่ 4 - เข้าถึงข้อมูลโดยอาศัยเทคโนโลยี OLEDB

โดยโปรแกรมนี้ได้เลือกใช้การติดต่อฐานข้อมูลแบบที่ 2 ซึ่งมีรายละเอียดดังนี้

2.5.4 การติดต่อฐานข้อมูลโดยใช้ออบเจ็กต์ Data Access Object (DAO)

รูปแบบการติดต่อแบบนี้ จะเป็นวิธีที่ล้าสมัยแล้ว โดยมีแนวคิดในการติดต่อหรือเข้าถึงฐานข้อมูลผ่านทางองค์ประกอบต่างๆในฐานข้อมูล เช่น ฟิลด์ (Field), เร็คคอร์ด (Record), ความสัมพันธ์ระหว่างตาราง (Relation) เป็นต้น โดยจะแทนแต่ละองค์ประกอบนั้นด้วยออบเจ็กต์ (Object) และควบคุมออบเจ็กต์ต่างๆ เหล่านี้โดยการเขียนโค้ด

แม้ว่าจะทำงานได้ดีกว่า อิสระกว่า แต่มีความยุ่งยากในการเขียนโปรแกรมด้วยเช่นกัน อีกทั้งยังเป็นเทคโนโลยีที่เก่าแล้วคือ เน้นเฉพาะฐานข้อมูลที่เป็นตาราง (โดยเฉพาะ Access รุ่นเก่าๆ) แต่ว่าการเก็บข้อมูลในปัจจุบันถูกจัดเก็บอยู่ในสภาพแวดล้อมที่ต่างกันมากมาย เช่น รูปภาพ (Image), ข้อความ (Text) และรูปแบบอื่นๆ อีกมากมายทำให้ต้องสร้างออบเจ็กต์ใหม่ๆ ขึ้นมาเรื่อย
แต่นั้นไม่ใช่สิ่งที่ทำกันได้ง่ายๆ และกลายเป็นข้อจำกัดสำคัญของ DAO

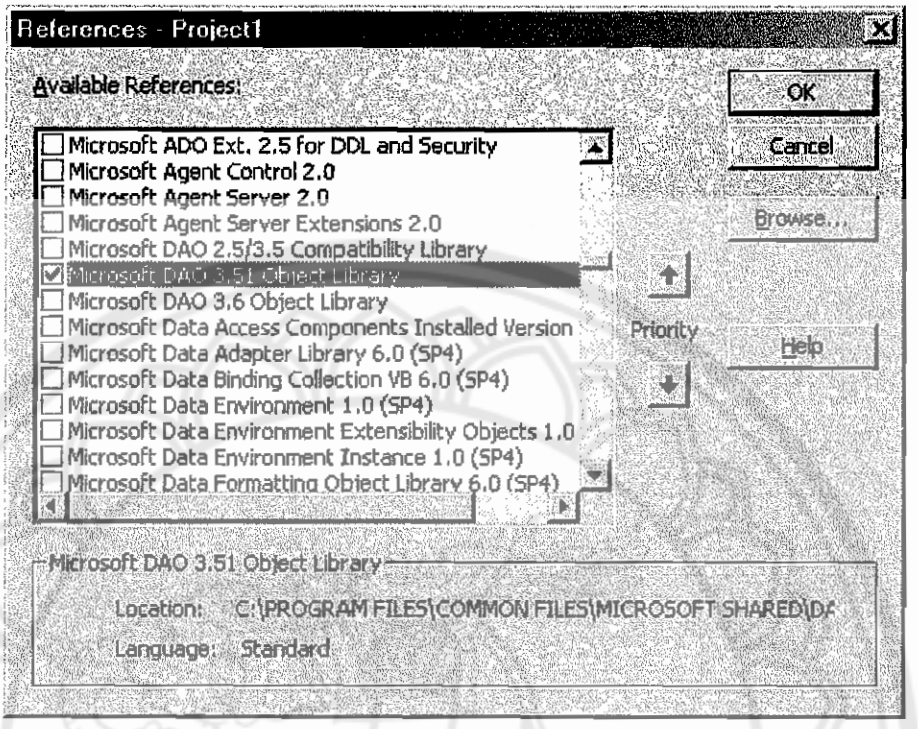
โดยการติดต่อฐานข้อมูลแบบ DAO มีองค์ประกอบดังนี้
ลักษณะของออบเจ็กต์ ในโมเดลของ DAO เบื้องต้นมีดังนี้

- ออบเจ็กต์ DBEngine (DBEngine Object) - เป็นออบเจ็กต์ระดับบนสุด เป็นตัวแทนของ JET Engine ที่ใช้สำหรับควบคุม และ/หรือจัดการโครงสร้างของฐานข้อมูลที่เราสร้างขึ้นมา ซึ่งจะมีเพียงตัวเดียวเท่านั้นใน 1 โปรเจ็กต์
- ออบเจ็กต์ Workspace (Workspace Object) – เป็นออบเจ็กต์ที่ใช้สำหรับจัดการพื้นที่สมมติขึ้นมาเพื่อเก็บฐานข้อมูลที่เราสร้างขึ้นมา
- ออบเจ็กต์ Database (Database Object) – เป็นออบเจ็กต์ที่เป็นตัวแทนของฐานข้อมูลนั้นๆ ซึ่งจะประกอบไปด้วยออบเจ็กต์ที่เป็นองค์ประกอบของฐานข้อมูล จะแบ่งออกเป็น 2 ประเภท
  กลุ่มที่ 1 : กลุ่มของออบเจ็กต์ที่ใช้สำหรับกำหนดลักษณะของฐานข้อมูลที่มีอยู่ 3 ตัว คือ
  - ออบเจ็กต์ TableDef (TableDef Object) เป็นออบเจ็กต์ที่ใช้สำหรับนิยาม หรือแทนตารางที่อยู่ในฐานข้อมูลใดๆ
  - ออบเจ็กต์ Field (Field Object) เป็นออบเจ็กต์ที่ใช้แทนฟิลด์ที่อยู่ในตารางใดๆ
  - ออบเจ็กต์ Index (Index Object) เป็นออบเจ็กต์ที่ให้แทนค่าดัชนีของตารางนั้นๆ

  กลุ่มที่ 2 : กลุ่มของออบเจ็กต์ที่ใช้สำหรับกำหนดราบละเอียดอื่นๆ ของฐานข้อมูล มีอยู่ 2 ตัว
  - ออบเจ็กต์ Relation (Relation Object) ใช้กำหนดหรือแทนความสัมพันธ์ระหว่างตาราง ในฐานข้อมูลใดๆ
  - ออบเจ็กต์ QueryDef (QueryDef Object) ใช้สำหรับสร้างชุดสตริงที่เป็นคำสั่ง SQL ที่ใช้ในฐานข้อมูลนั้นๆ

ในส่วนของการเรียกใช้งานกลุ่มออบเจ็กต์ในโมเดลของ DAO ทุกครั้ง จะต้องมีการกำหนดให้

VBIDE รู้จัก DAO เสียก่อน โดยการเลือก VBIDE ชนิด Standard EXE แล้วเลือกคำสั่ง Project > References... ดังรูปข้างล่างนี้



รูปที่ 2.10 ไดอะล็อกบ็อกซ์ References

ให้เลือกไปที่ตัวเลือก Microsoft DAO Objects 3.51/3.6 Library ก็จะทำให้สามารถเรียกใช้งานกลุ่มของออบเจ็กต์ในโมเดล DAO 3.51/3.6 ด้วย Visual Basic ได้แล้ว

2.5.5 เหตุผลของการนำ DAO มาใช้งาน

ซึ่งเหตุผลที่นำการติดต่อแบบนี้มาใช้ เนื่องจากข้อมูลส่วนใหญ่ของโครงงานนี้เป็นข้อมูลแบบข้อความ (Text) เพราะฉะนั้นจึงไม่มีความจำเป็นต้องใช้การติดต่อแบบอื่นให้ซับซ้อนมากขึ้น ซึ่งทำให้สามารถดึงข้อดีของการติดต่อแบบ DAO ซึ่งมีข้อดีคือ องค์ประกอบต่างๆ ในฐานข้อมูล จะถูก DAO มองเป็นออบเจ็กต์ทั้งหมด ซึ่งทำให้สามารถจัดการส่วนปลีกย่อยต่างๆ ในฐานข้อมูลได้เป็นอย่างดี ซึ่งจะส่งผลให้ทำงานได้ดีกว่า และอิสระกว่านั่นเอง