## แผนการสอนประจำบทเรียน

**รายชื่ออาจารย์ผู้จัดทำ** ผู้ช่วยศาสตราจารย์ณัฏฐพร พิมพายน

**รายละเอียดของเนื้อหา**

**ตอนที่** 6.1 **การกำหนดโครงสร้างข้อมูล**

เรื่องที่ 6.1.1 โครงสร้างของภาษาเอสคิวแอล

เรื่องที่ 6.1.2 ภาษาสำหรับนิยามข้อมูล

**ตอนที่** 6.2 **การบันทึกข้อมูล การปรับปรุงข้อมูล การลบข้อมูลและการเรียกข้อมูลอย่างง่าย**

เรื่องที่ 6.2.1 การบันทึกข้อมูล การปรับปรุงข้อมูล การลบข้อมูล

เรื่องที่ 6.2.2 การเรียกค้นข้อมูลอย่างง่าย

**ตอนที่** 6.3 **การเรียกค้นข้อมูล**

เรื่องที่ 6.3.1 ฟังก์ชัน

เรื่องที่ 6.3.2 การเรียกดูข้อมูลในรูปแบบต่าง ๆ

**แนวคิด**

1. SQL เป็นภาษามาตราฐานที่ใช้งานได้ตั้งแต่ระดับเครื่องคอมพิวเตอร์ส่วนบุคคลพีซีไปจนถึงระดับเมนเฟรม ประเภทของคำสั่งในภาษา SQL แบ่งออกเป็น 3 ประเภท คือภาษาสำหรับการนิยามข้อมูล ภาษาสำหรับการจัดการข้อมูล และภาษาควบคุม ภาษาสำหรับการนิยามข้อมูลประกอบด้วยคำสั่งที่ใช้ในการกำหนดโครงสร้างข้อมูลว่ามีคอลัมน์อะไร แต่ละคอลัมน์เก็บข้อมูลประเภทใด รวมถึงการเพิ่มคอลัมน์ การกำหนดดัชนี การกำหนดวิวของผู้ใช้
2. ภาษาสำหรับการจัดการข้อมูล ประกอบด้วยคำสั่งที่ใช้ในการเรียกใช้ข้อมูล การเปลี่ยนแปลงข้อมูล การเพิ่มหรือลบข้อมูลเป็นต้น
3. การเรียกค้นข้อมูลจากตารางเป็นการนำข้อมูลจากฐานข้อมูลโดยมีเงื่อนไขของการค้นข้อมูลหลายรูปแบบ การเรียกค้นข้อมูลสามารถค้นได้จากตารางเดียวหรือจากหลายตารางก็ได้ที่ได้มาจากการเรียกค้นข้อมูลนี้ เรียกว่าเป็นการสอบถามข้อมูล

**วัตถุประสงค์**

หลังจากศึกษาบทเรียนที่ 6 แล้ว นักศึกษาสามารถ

1. บอกถึงลักษณะภาษาสำหรับกำหนดโครงสร้างได้
2. บอกถึงลักษณะภาษาสำหรับการจัดการฐานข้อมูลได้

3. บอกถึงลักษณะภาษาสำหรับการเรียกค้นข้อมูลได้

**กิจกรรมการเรียนการสอน**

กิจกรรมที่นักศึกษาต้องทำสำหรับการเรียนการสอน ได้แก่

1. ศึกษาเอกสารการสอน
2. ปฏิบัติกิจกรรมตามที่ได้รับมอบหมายในเอกสารการสอนแต่ละตอน

**สื่อการสอน**

1. เอกสารการสอนของชุดวิชา
2. แบบฝึกปฏิบัติ
3. บทความ/ข้อมูลทางคอมพิวเตอร์
4. การให้คำปรึกษาทางโทรศัพท์
5. CD-ROM
6. Homepage ของชุดวิชาผ่านทางอินเตอร์เน็ต

**เอกสารประกอบการสอน**

1. Fundamentals of Database Systems, by Ramez Elmasri, Shamkant B. Navathe, The Second Edition, 1994
2. Database System Concepts, by Abraham Siberschaty, Henry F.Korth, S.Sudarshan, The Third Edition, 1991

**ประเมินผล**

1. ประเมินผลจากแบบฝึกหัด/ทดสอบ ในแต่ละบท
2. ประเมินผลจากการสอนประจำภาคการศึกษา

## ตอนที่ 6.1 การกำหนดโครงสร้างข้อมูล

**หัวเรื่อง**

เรื่องที่ 6.1.1 โครงสร้างของภาษาเอสคิวแอล

เรื่องที่ 6.1.2 ภาษาสำหรับนิยามข้อมูล

**แนวคิด**

1. ภาษาเอสคิวแอลหรือภาษาในการสอบถามข้อมูลที่เป็นโครงสร้าง เป็นภาษาที่เหมาะกับผู้ที่ทำงานกับฐานข้อมูล ภาษาเอสคิวแอลเป็นภาษาที่สร้างและปฏิบัติการต่อฐานข้อมูลสัมพันธ์ ซึ่งเป็นชุดข้อมูลที่เก็บในรูปของตาราง ภาษาเอสคิวแอลเป็นภาษาที่ง่ายต่อการใช้งานและมีเสรีในการใช้ภาษานี้กับคอมพิวเตอร์ชนิดต่างๆ โครงสร้างของภาษาเอสคิวแอลประกอบด้วย ภาษาสำหรับการนิยามข้อมูล ภาษาสำหรับการจัดการข้อมูลและภาษาควบคุม
2. ภาษาสำหรับนิยามข้อมูลเป็นส่วนหนึ่งของภาษาเอสคิวแอล เพื่อใช้นิยามโครงสร้างของฐานข้อมูล เพื่อทำการสร้างเปลี่ยนแปลงหรือยกเลิกโครงสร้างของฐานข้อมูลตามที่ได้ออกแบบไว้ โดยโครงสร้างของฐานข้อมูลประกอบด้วยตาราง คีย์หลักของตาราง ภาษาสำหรับนิยามข้อมูลใช้ในการสร้างตาราง สร้างคีย์หลักของตาราง และใช้ในการลบตาราง ลบคีย์หลักของตาราง

**วัตถุประสงค์**

หลังจากที่ศึกษาตอนที่ 6.1 แล้ว นักศึกษาสามารถ

1. บอกลักษณะและส่วนประกอบของภาษาเอสคิวแอลได้
2. บอกวิธีการสร้างตารางและการลบตารางได้
3. บอกวิธีการสร้างดัชนีและการลบดัชนีได้

### เรื่องที่ 6.1.1 โครงสร้างของภาษาเอสคิวแอล

ภาษา SQL (สามารถอ่านออกเสียงได้ 2 แบบ คือ “เอสคิวแอล” (SQL) หรือ “ซีเควล” (Sequel)) ย่อมาจาก Structured Query Language หรือภาษาในการสอบถามข้อมูล เป็นภาษาทางด้านฐานข้อมูล ที่สามารถสร้างและปฏิบัติการกับฐานข้อมูลแบบสัมพันธ์(relational database)โดยเฉพาะ และ เป็นภาษาที่มีลักษณะคล้ายกับภาษาอังกฤษ ภาษา SQLถูกพัฒนาขึ้นจากแนวคิดของ relational calculus และ relational algebra เป็นหลัก ภาษา SQL เริ่มพัฒนาครั้งแรกโดย almaden research center ของบริษัท IBM โดยมีชื่อเริ่มแรกว่า “ซีเควล” (Sequel) ต่อมาได้เปลี่ยนชื่อเป็น“เอสคิวแอล” (SQL) หลังจากนั้นภาษา SQLได้ถูกนำมาพัฒนาโดยผู้ผลิตซอฟแวร์ด้านระบบจัดการฐานข้อมูลเชิงสัมพันธ์จนเป็นที่นิยมกันอย่างแพร่หลายในปัจจุบัน โดยผู้ผลิตแต่ละรายก็พยายามที่จะพัฒนาระบบจัดการฐานข้อมูลของตนให้มีลักษณะเด่นเฉพาะขึ้นมา ทำให้รูปแบบการใช้คำสั่ง SQL มีรูปแบบที่แตกต่างกันไปบ้าง เช่น ORACLE ACCESS SQL Base ของ Sybase INGRES หรือ SQL Server ของ Microsoft เป็นต้น ดังนั้นในปี ค.ศ. 1986 ทางด้าน American National Standards Institute (ANSI) จึงได้กำหนดมาตรฐานของ SQL ขึ้น อย่างไรก็ดี โปรแกรมฐานข้อมูลที่ขายใน

ท้องตลาด ได้ขยาย SQL ออกไปจนเกินข้อกำหนดของ ANSI โดยเพิ่มคุณสมบัติอื่นๆ ที่คิดว่าเป็นประโยชน์เข้าไปอีกแต่โดยหลักทั่วไปแล้วก็ยังปฏิบัติตามมาตราฐานของ ANSI ในการอธิบายคำสั่งต่างๆของภาษา SQL ในหนังสือเล่มนี้จะอธิบายคำสั่งที่เป็นรูปแบบคำสั่งมาตราฐานของภาษา SQLโดยทั่วไป

1.**ประเภทของคำสั่งของภาษา** SQL

ภาษา SQL เป็นภาษาที่ใช้งานได้ตั้งแต่ระดับเครื่องคอมพิวเตอร์ส่วนบุคคลพีซีไปจนถึงระดับเมนเฟรม ประเภทของคำสั่งในภาษา SQL (The subdivision of sql) แบ่งออกเป็น 3 ประเภท คือ

1. ภาษาสำหรับการนิยามข้อมูล (Data Definition Language : DDL) ประกอบด้วยคำสั่งที่ใช้ในการกำหนดโครงสร้างข้อมูลว่ามีคอลัมน์อะไร แต่ละคอลัมน์เก็บข้อมูลประเภทใด รวมถึงการเพิ่มคอลัมน์ การกำหนดดัชนี การกำหนดวิวหรือตารางเสมือนของผู้ใช้ เป็นต้น

2. ภาษาสำหรับการจัดการข้อมูล (Data Manipulation Language : DML) ประกอบด้วยคำสั่งที่ใช้ในการเรียกใช้ข้อมูล การเปลี่ยนแปลงข้อมูล การเพิ่มหรือลบข้อมูล เป็นต้น

3. ภาษาควบคุม (Data Control Language : DCL) : ประกอบด้วยคำสั่งที่ใช้ในการควบคุม การเกิดภาวะพร้อมกัน หรือการป้องกันการเกิดเหตุการณ์ที่ผู้ใช้หลายคนเรียกใช้ข้อมูลพร้อมกัน และคำสั่งที่เกี่ยวข้องกับการควบคุมความปลอดภัยของข้อมูลด้วยการกำหนดสิทธิของผู้ใช้ที่แตกต่างกัน เป็นต้น

2. **ชนิดของข้อมูลที่ใช้ในภาษา** SQL

ในภาษา SQL การบรรจุข้อมูลลงในคอลัมน์ต่าง ๆ ของตารางจะต้องกำหนดชนิดของข้อมูล (data type) ให้แต่ละคอลัมน์ ชนิดของข้อมูลนี้จะแสดงชนิดของค่าที่อยู่ในคอลัมน์ ค่าทุกค่าในคอลัมน์ที่กำหนดจะต้องเป็นชนิดเดียวกัน เช่น ในตารางลูกค้าคอลัมน์ที่เป็นรายชื่อลูกค้า จะต้องเป็นตัวหนังสือ ในขณะที่คอลัมน์จำนวนเงินที่ลูกค้าซื้อสินค้าเป็นตัวเลข

ชนิดของข้อมูลของแต่ละคอลัมน์จะขึ้นกับลักษณะของข้อมูลแต่ละคอลัมน์ ซึ่งแบ่งได้ดังนี้ชนิดข้อมูลพื้นฐานในภาษา SQL ดังนี้

2.1 ตัวหนังสือ(character) ในภาษา SQL จะใช้

- ตัวหนังสือแบบความยาวคงที่(fixed-length character) จะใช้ char (n) หรือ character(n) แทนประเภทของข้อมูลที่เป็นตัวหนังสือใดๆที่มีความยาวของข้อมูลคงที่โดยมีความยาว n ตัวหนังสือประเภทนี้จะมีการจองเนื้อที่ตามความยาวที่คงที่ตามที่กำหนดไว้ ชนิดของข้อมูลประเภทนี้จะเก็บความยาวของข้อมูลได้มากที่สุดได้ 255 ตัวอักษร

- ตัวหนังสือแบบความยาวไม่คงที่(variable-length character) จะใช้ varchar (n) แทนประเภทของข้อมูลที่เป็นตัวหนังสือใดๆที่มีความยาวของข้อมูลไม่คงที่ โดยมีความยาว n ตัวหนังสือประเภทนี้จะมีการจองเนื้อที่ตามความยาวของข้อมูล ชนิดของข้อมูลประเภทนี้จะเก็บความยาวของข้อมูลได้มากที่สุดได้ 4000 ตัวอักษร

2.2 จำนวนเลข( numeric)

- จำนวนเลขที่มีจุดทศนิยม(decimal) ในภาษา SQL จะใช้ dec(m,n) หรือ decimal(m,n) เป็นประเภทข้อมูลที่เป็นจำนวนเลขที่มีจุดทศนิยมโดย m คือจำนวนตัวเลขทั้งหมด (รวมจุดทศนิยม) และ n คือจำนวนตัวเลขหลังจุดทศนิยม

- จำนวนเลขที่ไม่มีจุดทศนิยมในภาษา SQL จะใช้ int หรือ integer เป็นเลขจำนวนเต็มบวกหรือลบขนาดใหญ่ เป็นตัวเลข 10 หลัก ที่มีค่าตั้งแต่ –2,147,483,648 ถึง +2,147,483,647 และในภาษา SQL จะใช้ smallint เป็นประเภทข้อมูลที่เป็นเลขจำนวนเต็มบวกหรือลบขนาดเล็ก เป็นตัวเลข 5 หลัก ที่มีค่าตั้งแต่ – 32,768 ถึง + 32,767 ตัวเลขจำนวนเต็มประเภทนี้จะมีการจองเนื้อที่น้อยกว่าแบบ integer

- เลขจำนวนจริง ในภาษา SQL อาจใช้ number(n)แทนจำนวนเลขที่ไม่มีจุดทศนิยมและจำนวนเลขที่มีจุดทศนิยม

2.3 ข้อมูลในลักษณะอื่นๆ

- วันที่และเวลา(Date/Time) เป็นชนิดวันที่หรือเวลาในภาษา SQL จะใช้ date เป็นข้อมูลวันที่ ซึ่งจะมีหลายรูปแบบให้เลือกใช้ เช่น yyyy-mm-dd (1999-10-31) dd.mm.yyyy(31. 10.1999) หรือ dd/mm/yyyy (31/10/1999)

3. **ลักษณะการใช้งานของภาษา** SQL

ภาษา SQL เป็นส่วนประกอบหนึ่งของ DBMS มักพบใน DBMS เชิงสัมพันธ์หลายตัวและเป็นที่นิยมใช้ในปัจจุบัน ภาษา SQL ง่ายต่อการเรียนรู้ การใช้งานในภาษา SQLแบ่งเป็น 2 ลักษณะ คือ ภาษา SQL ที่โต้ตอบได้ (interactive SQL)และภาษา SQL ที่ฝังในโปรแกรม (embedded SQL)

3.1 **ภาษา** SQL **ที่โต้ตอบได้** ใช้เพื่อปฏิบัติงานกับฐานข้อมูลโดยตรง เป็นการใช้คำสั่งภาษา SQLสั่งงานบนจอภาพ โดยเรียกดูข้อมูลได้โดยตรงในขณะที่ทำงาน เพื่อให้ได้ผลลัพธ์ที่นำไปใช้ได้ ตัวอย่างเช่น ต้องการเรียกดูข้อมูลในคอลัมน์ SALENAME และ SALECOM จากตาราง SALESTAB จะใช้คำสั่งของภาษา SQL ดังนี้

```
SELECT  SALENAME, SALECOM
FROM SALESTAB;
```

โดยตาราง SALESTAB มีรายละเอียดของตารางดังนี้

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |
| 1003 | Ternjai | Nonthaburi | 0.10 |

ผลของคำสั่งจะแสดงผลลัพธ์ดังนี้ทันที

| SALENAME | SALECOM |
|---|---|
| Chaiwat | 0.12 |
| Mitree | 0.13 |
| Benjawan | 0.11 |
| Kanjana | 0.15 |
| Ternjai | 0.10 |

(รายละเอียดจะได้ศึกษาต่อไป)

3.2 **ภาษา** SQL **ที่ฝังในโปรแกรม** เป็นภาษา SQL ที่ประกอบด้วยคำสั่งต่าง ๆ ของ ภาษา SQL ที่ใส่ไว้ในโปรแกรมที่ส่วนมากแล้วเขียนด้วยภาษาอื่น เช่น โคบอล ปาสคาล ภาษาซี ลักษณะของคำสั่ง SQL จะแตกต่างจากภาษาอื่นๆ ในแง่ที่ว่า SQL ไม่มีคำสั่งที่เกี่ยวกับการควบคุม(control statement)เหมือนภาษาอื่น เช่น if..then...else for...do หรือ loop หรือ while ทำให้มีข้อจำกัดในการเขียนชุดคำสั่งงาน การใช้ภาษา SQL ฝังในโปรแกรมอื่นจะทำให้ภาษา SQL มีความสามารถและมีประสิทธิภาพมากยิ่งขึ้น ผลลัพธ์ของคำสั่งที่เกิดจากภาษา SQL ที่ฝังในโปรแกรมจะถูกส่งผ่านไปให้กับตัวแปรหรือพารามิเตอร์ที่ใช้ โดยโปรแกรมที่ภาษา SQL ไปฝังตัวอยู่ เช่น

```
while not end-of-file(input) do
    begin
        readin(id-num, salesperson,loc,comm);
        EXEC SQL INSERT INTO SALESTAB
        VALUES(:id-num,:salesperson,:loc,:comm);
    end;
```

จากตัวอย่างถ้าใช้คำสั่ง

```
INSERT INTO SALESTAB
VALUES (:id-num,:salesperson,: loc, :comm);
```

เพียงอย่างเดียว จะทำให้คำสั่งนี้ใส่ค่า id-num salesperson loc comm ใส่ค่าได้เพียงครั้งเดียว แต่เมื่อนำคำสั่งนี้มาใส่ไว้ในภาษาปาสคาลข้างต้นจะทำให้คำสั่งดังกล่าวมีความสามารถสูงขึ้นคือคำสั่งนี้จะสามารถทำงานซ้ำ(loop) โดยใส่ค่าต่าง ๆลงในตัวแปรเพื่อให้ทำซ้ำกันหลาย ๆครั้ง โดยจากตัวอย่างส่วนของโปรแกรมภาษาปาสคาลจะกำหนดลูปวนซึ่งจะอ่านค่าจากแฟ้มข้อมูลแล้วเก็บค่านั้นไว้ในตัวแปร id-num,

salesperson, loc, comm ของตารางSALESTAB การอ่านค่าแล้วเก็บค่าไว้ในตัวแปรจะทำซ้ำจนกระทั่งข้อมูลหมดจากแฟ้มข้อมูล

ทั้งภาษา SQL ที่โต้ตอบได้และภาษา SQL ที่ฝังในโปรแกรมจะมีลักษณะของคำสั่งที่ใช้งานเหมือนกัน จะต่างกันแต่เพียงภาษา SQL ที่ฝังในโปรแกรมจะมีวิธีการเชื่อมโยงกับภาษาอื่น ๆ

**เรื่องที่** 6.1.2 **ภาษาสำหรับนิยามข้อมูล**

คำสั่งในภาษา SQL (The subdivision of sql) แบ่งออกเป็น 3 ประเภท คือ

- คำสั่งภาษาสำหรับการนิยามข้อมูล (Data Definition Language : DDL)
- คำสั่งภาษาสำหรับการจัดการข้อมูล (Data Manipulation Language : DML)
- คำสั่งภาษาควบคุม (Data Control Language : DCL)

1. **ตารางข้อมูล**

สำหรับภาษา SQL เป็นภาษาที่ใช้สำหรับฐานข้อมูลแบบสัมพันธ์ คือประกอบด้วยตารางและในตารางหนึ่งๆมี 2 มิติได้แก่ แถว (rows) ในแนวนอน และคอลัมน์(columns) ในแนวตั้ง ฐานข้อมูลแบบสัมพันธ์ เช่น

**ตารางพนักงานขาย**(SALESTAB)

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |
| 1003 | Ternjai | Nonthaburi | 0.10 |

คำอธิบายของคอลัมน์ต่าง ๆ ในตาราง

| **คอลัมน์** | **ชนิดข้อมูล** | **รายละเอียด** |
|---|---|---|
| SALENO | Integer | เลขประจำตัวพนักงานขาย |
| SALENAME | Char(10) | ชื่อพนักงานขาย |
| ADDRESS | Char(10) | ที่อยู่ของพนักงานขาย |
| SALECOM | Decimal | ค่าคอมมิชชั่นของพนักงานขายตามคำสั่งซื้อ |

ภาษาสำหรับนิยามข้อมูล (Data Definition Language : DDL) เป็นส่วนหนึ่งของภาษา SQL โดยเป็นภาษาที่ใช้นิยามโครงสร้างของฐานข้อมูล เพื่อทำการสร้างเปลี่ยนแปลงหรือยกเลิกโครงสร้างของฐานข้อมูลตามที่ได้ออกแบบไว้ โครงสร้างของฐานข้อมูลสามารถเรียกได้อีกอย่างว่าสคีมา (sehema) ดังนั้นภาษา

สำหรับนิยามข้อมูล จึงเป็นภาษาที่ใช้ในการสร้างสคีมานั้นเอง หลังจากที่ได้มีการออกแบบฐานข้อมูลเรียบร้อยแล้วจะทำให้ทราบว่าฐานข้อมูลนั้นมีสคีมาอย่างไร และประกอบด้วยตารางใดบ้าง แต่ละตารางสัมพันธ์กันอย่างไร คีย์หลักของตารางคืออะไร เมื่อทราบถึงรายละเอียดต่างๆที่ได้จากการออกแบบฐานข้อมูลแล้วก็จะทำการสร้างตารางต่างๆที่จะใช้เป็นฐานข้อมูลลงในเครื่องคอมพิวเตอร์

**2. การสร้างตาราง**

การสร้างตารางในภาษา SQL จะใช้คำสร้าง CREATE TABLE ซึ่งเป็นคำสั่งที่ใช้ในการสร้างตารางขึ้นมาใหม่ คำสั่ง CREATE TABLE จะกำหนดชื่อตารางและกำหนดลักษณะข้อมูลเป็นคอลัมน์ต่างๆที่ตั้งขึ้นในตารางรวมไปถึงชนิดของข้อมูลของแต่ละคอลัมน์นั้น ในโครงสร้างของคำสั่งการสร้างตารางมีรูปแบบไวยากรณ์ดังต่อไปนี้

CREATE TABLE<table name>

(<column name>< >[<size>][[ constraint <constraint_name>]constraint_type]

[,<column name>data type>[<size>],.....]);

CREATE TABLE เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการสร้างตาราง

table name ชื่อตารางที่ต้องการสร้าง

column name ชื่อของคอลัมน์แต่ละคอลัมน์

data type ชนิดข้อมูลของคอลัมน์นั้นๆ

constraint ข้อกำหนดของคอลัมน์

constraint_name ชื่อของข้อกำหนดที่ต้องการสร้างให้กับคอลัมน์

constraint_type ประเภทของข้อกำหนด

ตัวอย่างที่ การสร้างตารางพนักงานขาย

```
CREATE TABLE SALESTAB
    (SALENO     integer,
    SALENAME   char (10),
    ADDRESS    char (10),
    SALECOM    decimal);
```

จากคำสั่งจะทำให้ได้ตารางพนักงานขายที่มีคอลัมน์ SALENO มีชนิดข้อมูลเป็น integer คอลัมน์ SALENAME มีชนิดข้อมูลเป็น char มีความยาว 10 ตัวอักษร คอลัมน์ ADDRESS มีชนิดข้อมูลเป็น char มีความยาว 10 ตัวอักษร และคอลัมน์ SALECOM มีชนิดข้อมูลเป็น decimal

ผลของคำสั่งการสร้างตารางจะได้ตารางพนักงานขายที่ยังไม่มีข้อมูลใดๆ เป็นเพียงแต่โครงของตารางเท่านั้นดังนี้

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|

| | | | |
|---|---|---|---|
| | | | |

3. **การสร้างตารางโดยมีการกำหนดข้อจำกัด**

การสร้างตารางสามารถกำหนดข้อจำกัด(constraints)ลงในค่าต่าง ๆที่จะป้อนลงในคอลัมน์ต่าง ๆของตารางได้ การกำหนดข้อจำกัดเป็นการควบคุมความถูกต้องสมบูรณ์(integrity)ที่จัดเก็บในฐานข้อมูลให้มีความถูกต้องตามที่ถูกกำหนดไว้หรือตามที่ควรจะเป็น การกำหนดข้อจำกัดทำให้ข้อมูลมีความเชื่อถือได้ การกำหนดข้อจำกัดจะทำให้ข้อมูลในตารางไม่สามารถรับค่าใด ๆที่ไม่ตรงกับข้อจำกัดที่กำหนดไว้ การกำหนดข้อจำกัดที่เป็นการควบคุมความถูกต้องสมบูรณ์(integrity)ได้ดังนี้

3.1 การกำหนดไม่ให้ค่าใดค่าหนึ่งเป็นค่าว่าง(NOT NULL) เป็นการกำหนดข้อมูลของคอลัมน์ใดคอลัมน์หนึ่งมีค่าว่างไม่ได้ โดยใช้คำว่า “NOT NULL” เช่น คอลัมน์ที่เป็นคีย์หลัก(primary key)ถูกระบุไม่ให้ค่าใดค่าหนึ่งเป็นค่าว่าง(NOT NULL ) หรือต้องการให้ลูกค้าทุกคนในตารางลูกค้าต้องมีข้อมูลชื่อ โดยทั่วไปการสร้างตารางถ้าในคอลัมน์ไม่ระบุคำว่า “NOT NULL” คอลัมน์นั้นจะถูกระบุให้เป็นค่า NULL โดยปริยาย(DEFAULT) นั่นคือคอลัมน์นั้นสามารถมีค่าว่างได้(NULL)

ตัวอย่าง สมมติว่าต้องการกำหนดให้ตารางพนักงานขายในคอลัมน์ SALENO และ คอลัมน์ SALENAMEไม่ให้เป็นค่าว่าง(NOT NULL ) จะสามารถสร้างตารางพนักงานด้วยคำสั่งดังนี้

```
CREATE TABLE  SALESTAB
    (SALENO          integer   NOT NULL,
    SALENAME         char(10) NOT NULL,
    ADDRESS          char(10),
    SALECOM          decimal);
```

3.2 การกำหนดไม่ให้มีค่าซ้ำกัน(UNIQUE) เป็นการสร้างตารางโดยกำหนดให้คอลัมน์นั้นทั้งตารางไม่ให้มีค่าซ้ำกัน โดยใช้คำว่า “UNIQUE” เช่น คอลัมน์รหัสพนักงานที่เป็นคีย์หลัก และไม่ต้องการให้มีค่าซ้ำจะใช้คำว่า UNIQUE เป็นการระบุข้อจำกัดนี้

ตัวอย่าง สมมติว่าต้องการกำหนดให้ตารางพนักงานขายในคอลัมน์ SALENO และ คอลัมน์ SALENAMEไม่ให้เป็นค่าว่าง(NOT NULL ) และไม่ให้มีค่าซ้ำกัน จะสามารถสร้างตารางพนักงานด้วยคำสั่งดังนี้

```
CREATE TABLE SALESPEOPLE
```

```
(SALENO          integer  NOT NULL UNIQUE,
SALENAME         char(10) NOT NULL UNIQUE,
ADDRESS          char(10),
SALECOM          decimal);
```

3.3. การกำหนดคีย์หลัก(primary key) สามารถกำหนดได้ 2 วิธีคือ

1) การกำหนดให้คอลัมน์เดียวเป็นคีย์หลัก

ตัวอย่าง สมมติว่าต้องสร้างตารางพนักงานที่กำหนดให้คอลัมน์ SALENO เป็น คีย์หลัก(primary key) โดยไม่ให้มีค่าซ้ำกัน และคอลัมน์ SALENAMEไม่ให้เป็นค่าว่าง(NOT NULL) และไม่ให้มีค่าซ้ำกัน จะสามารถสร้างตารางพนักงานด้วยคำสั่งดังนี้

```
CREATE TABLE SALESPEOPLE
    (SALENO      integer  NOT NULL UNIQUE PRIMARY KEY,
    SALENAME     char(10) NOT NULL UNIQUE,
    ADDRESS      char(10),
    SALECOM      decimal);
```

2) การกำหนดให้คอลัมน์มากกว่า 1 คอลัมน์เป็นคีย์หลัก ในบางครั้งการอ้างอิงคีย์หลัก อาจต้องใช้คอลัมน์มากกว่า 1 คอลัมน์เป็นคีย์หลัก

ตัวอย่าง สมมติว่าต้องสร้างตาราง NAMEFIELD โดยกำหนดให้คอลัมน์ FIRSTNAME และคอลัมน์ LASTNAME เป็น คีย์หลัก(primary key) จะสามารถสร้างตาราง NAMEFIELD ด้วยคำสั่งดังนี้

```
CREATE TABLE NAMEFIELD
    (FIRSTNAME   char(10) NOT NULL ,
    LASTNAME             char(10) NOT NULL UNIQUE,
    CITY         char(10),
    PRIMARY KEY (FIRSTNAME LASTNAME ));
```

3.4 การกำหนดคีย์นอก(foreign key ) คีย์นอกเป็นคอลัมน์ของตารางหนึ่งที่ใช้เชื่อมโยงหรืออ้างอิงข้อมูลกับอีกตารางหนึ่งที่มีคอลัมน์ที่มีชื่อคอลัมน์เดียวกัน เช่น ลูกค้าในตารางลูกค้า แต่ละคนมีคอลัมน์ SALENO ที่อยู่ในตารางพนักงานขาย

หากกำหนดคีย์นอกเป็นข้อจำกัดในระดับคอลัมน์จะใช้คำสั่ง REFERENCE ต่อท้ายประเภทและขนาดของคอลัมน์ที่เป็นคีย์นอก

| CUSNO | CUSNAME | SALENO |
|---|---|---|
| 2001 | Arlee | 1001 |
| 2002 | Tanachote | 1003 |
| 2003 | Tawatchai | 1002 |
| 2004 | Amporni | 1002 |
| 2006 | Surasit | 1001 |
| 2008 | Jintana | 1007 |
| 2007 | Siriwan | 1004 |

ตารางลูกค้า( CUSTOMERTAB)

| SALENO | SALENAME | SALECOM |
|---|---|---|
| 1001 | Chaiwat | 0.12 |
| 1002 | Mitree | 0.13 |
| 1004 | Benjawan | 0.11 |
| 1007 | Kanjana | 0.15 |
| 1003 | Ternjai | 0.10 |

ตารางพนักงานขาย (SALESTAB)

จากตาราง แสดงการอ้างอิงข้อมูลโดยคอลัมน์ที่มีชื่อเดียวกัน เป็นตารางลูกค้าและตารางพนักงานขายโดยในที่นี้จะไม่กล่าวถึงคอลัมน์ต่าง ๆที่ไม่จำเป็นเพื่อสะดวกในการศึกษา ลูกค้าในตารางลูกค้าแต่ละคนมีคอลัมน์ SALENO (เป็นคอลัมน์เดียวกันกับคอลัมน์ SALENO ที่อยู่ในตารางพนักงานขาย) คอลัมน์ SALENOที่อยู่ตารางลูกค้าเป็นคอลัมน์ที่แสดงพนักงานขายที่กำหนดให้กับลูกค้าแต่ละคน

ตัวอย่าง ถ้าต้องการสร้างตารางลูกค้า( CUSTOMERSTAB) โดยกำหนดให้คอลัมน์CUSNO เป็น PRIMARY KEY และคอลัมน์SALENO เป็นคีย์นอก (foreign key) ที่ใช้เชื่อมโยงหรืออ้างอิงข้อมูลกับตารางพนักงานขาย( SALESTAB) โดยใช้คำสั่งดังนี้

```
CREATE TABLE CUSTOMERSTAB
    (CUSNO      integer  NOT NULL  PRIMARY KEY,
    CUSNAME     char(10) ,
    ADDRESS     char(10),
    SALENO      integer,
    FOREIGN KEY (SALENO) REFERENCES  SALESTAB(SALENO));
```

หรือ

```
CREATE TABLE CUSTOMERSTAB
    (CUSNO      integer  NOT NULL  PRIMARY KEY,
    CUSNAME     char(10) ,
    ADDRESS     char(10),
    SALENO      integer  REFERENCES  SALESTAB(SALENO));
```

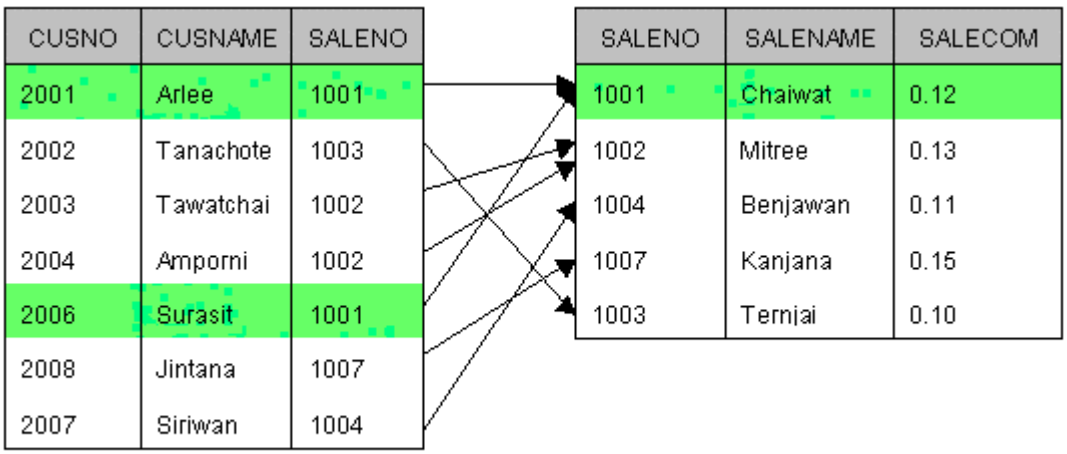
การกำหนดคีย์นอก (foreign key ) เป็นคอลัมน์ของตารางหนึ่งที่ใช้เชื่อมโยงหรืออ้างอิงข้อมูลกับอีกตารางหนึ่ง จะทำให้การปรับปรุงตารางมีผลต่อตารางอ้างอิง คำสั่งที่มีผลต่อการปรับปรุงตาราง ได้แก่คำสั่ง "CASCADES" และ "RESTRICTED" ทั้ง 2 คำสั่งนี้ใช้เพื่อควบคุมความถูกต้องครบถ้วนสมบูรณ์ในการอ้างอิงข้อมูล(referential integrity) ดังตัวอย่างต่อไปนี้

```
CREATE TABLE CUSTOMERSTAB
    (CUSNO      integer  NOT NULL  PRIMARY KEY,
    CUSNAME     char(10) ,
    ADDRESS     char(10),
    SALENO      integer  REFERENCES  SALESTAB(SALENO)
    UPDATE OF SALESTAB CASCADES,
    DELETE OF SALESTAB RESTRICTED);
```

ผลของคำสั่งนี้จะทำให้เมื่อต้องการปรับปรุงตารางที่สร้างขึ้นหรือตารางที่อ้างอิงถึง จะต้องมีเงื่อนไขในการปรับปรุง โดย

- คำสั่ง UPDATE OF SALESTAB CASCADES จะทำให้เมื่อมีการปรับปรุงคอลัมน์SALENOในตารางลูกค้าจะทำให้คอลัมน์ SALENOในตารางพนักงานขายถูกปรับปรุงไปด้วย

- คำสั่ง DELETE OF SALESTAB RESTRICTED จะทำให้เมื่อต้องการลบคอลัมน์SALENOในตารางลูกค้าจะไม่สามารถลบได้ ถ้าคอลัมน์SALENOในตารางพนักงานขายยังมีข้อมูลอยู่



| CUSNO | CUSNAME | SALENO |
|---|---|---|
| 2001 | Arlee | 1001 |
| 2002 | Tanachote | 1003 |
| 2003 | Tawatchai | 1002 |
| 2004 | Amporni | 1002 |
| 2006 | Surasit | 1001 |
| 2008 | Jintana | 1007 |
| 2007 | Siriwan | 1004 |

ตารางลูกค้า( CUSTOMERTAB)

| SALENO | SALENAME | SALECOM |
|---|---|---|
| 1001 | Chaiwat | 0.12 |
| 1002 | Mitree | 0.13 |
| 1004 | Benjawan | 0.11 |
| 1007 | Kanjana | 0.15 |
| 1003 | Ternjai | 0.10 |

ตารางพนักงานขาย (SALESTAB)

จากตัวอย่างสมมุติว่าต้องการลบ Chaiwat ออกจากตารางพนักงานขาย (SALESTAB) คำสั่งนี้ก็จะไม่เป็นที่ยอมรับ นอกเสียจากเปลี่ยนค่าคอลัมน์ SALENO ของลูกค้าชื่อ Arlee และ Surasit ไปเป็น SALENO

ของพนักงานขายผู้อื่น หรือกล่าวอีกอย่างหนึ่งว่าจะเปลี่ยนค่า SALENO ของ Chaiwat เป็น 1009 แล้ว ในตารางลูกค้า Arlee และ Surasit ก็จะเปลี่ยนค่า SALENO ของทั้งสองคนนั้นตามไปด้วยโดยอัตโนมัติ

3.5 การกำหนดการตรวจสอบ(CHECK) การตรวจสอบความถูกต้องครบถ้วนสมบูรณ์ของข้อมูล(entity integrity)โดยการระบุเงื่อนไขหรือกำหนดค่าเฉพาะของคอลัมน์ใดคอลัมน์หนึ่งขึ้น หากมีการป้อนข้อมูลที่ผิดจากเงื่อนไขที่ระบุไว้ ค่านั้นก็จะถูกปฏิเสธหรือไม่ยอมรับ การตรวจสอบจะใช้คำสั่ง CHECK <เงื่อนไข>ต่อท้ายชนิดและขนาดของข้อมูลคอลัมน์

ตัวอย่าง สมมติว่าต้องการกำหนดให้ตารางพนักงานขายในคอลัมน์ SNUM และ SNAME ไม่ให้เป็นค่าว่าง (NOT NULL ) และในคอลัมน์ SALENO และคอลัมน์ SALENAME ไม่ให้มีค่าซ้ำกันพร้อมทั้งกำหนดให้คอลัมน์ SALECOM ชนิดข้อมูลที่ป้อนลงไปจะเป็นเลขทศนิยมเท่านั้น จะสามารถสร้างตารางพนักงานด้วยคำสั่งดังนี้

```
CREATE TABLE  SALESPEOPLE
    (SALENO          integer   NOT NULL UNIQUE,
    SALENAME         char(10) NOT NULL UNIQUE,
    CITY          char(10),
    SALECOM          decimal CHECK (salecom < 1);
```

4.**ข้อคำนึงในการใช้คำสั่งสร้างตาราง**

4.1 ในการสร้างตารางแต่ละตารางอย่างน้อยที่สุดต้องกำหนดคอลัมน์ได้ 1 คอลัมน์

4.2 รายละเอียดของแต่ละคอลัมน์แยกจากกันด้วยเครื่องหมาย comma (,)

4.3 สิ้นสุดคำสั่งด้วยเครื่องหมาย semicolon (;)

*หมายเหตุ ข้อคำนึงเหล่านี้อาจมีความแตกต่างกันไปบ้าง ขึ้นอยู่กับผู้ผลิตแต่ละราย

5.**การลบโครงสร้างตารางออกจากระบบ**

เมื่อต้องการลบโครงสร้างตารางที่ถูกสร้างขึ้นจะสามารถทำได้ด้วยคำสั่ง DROP TABLE ซึ่งมีรูปแบบทั่วไปดังนี้

DROP TABLE <table name>[CASCADE CONSTRAINTS];

DROP TABLE เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการลบโครงสร้างตาราง

table name ชื่อตารางที่ต้องการลบ

CASCADE CONSTRAINTS ระบบจัดการฐานข้อมูลจะทำการลบข้อจำกัดต่าง ๆ(constraint) ที่มีการอ้างถึงตารางทิ้งไปให้ด้วยทั้งหมด

ตัวอย่าง ถ้าต้องการลบตารางพนักงานขาย (SALESTAB) จะใช้คำสั่งดังนี้

DROP TABLE SALESTAB;

ผลของคำสั่งการลบโครงสร้างตาราง จะทำให้ข้อมูลถูกลบไปด้วยจะทำให้ดัชนี( index) ทุกตัวและตารางเสมือนหรือวิวที่สร้างขึ้นสำหรับตารางSALESTABนี้ จะถูกลบไปพร้อมๆกันด้วย เมื่อมีการใช้คำสั่งลบโครงสร้างตารางเกิดขึ้นก่อนที่จะลบโครงสร้างตารางข้อมูล DBMS จะเตือนผู้ใช้ถึงผลที่เกิดจากการลบโครงสร้างตาราง

**6. การเปลี่ยนแปลงโครงสร้างตาราง**

เมื่อสร้างโครงสร้างตารางแล้ว ถ้าต้องการเปลี่ยนแปลงโครงสร้างตารางที่มีการสร้างไว้ข้างต้นใหม่ เช่น ต้องการเพิ่มหรือลบบางคอลัมน์ที่เป็นโครงสร้างหลักของตารางออก หรือต้องการเปลี่ยนประเภทข้อมูลของคอลัมน์ ซึ่งในกรณีที่ตารางมีข้อมูลและกำหนดโครงสร้างไปแล้ว การแก้ไขโครงสร้างข้อมูลอาจมีผลกระทบกับข้อมูลที่มีอยู่ แต่ในภาษา SQL สามารถใช้คำสั่งในการแก้ไขโครงสร้างข้อมูลได้ด้วยคำสั่งการเปลี่ยนแปลงโครงสร้างตาราง รูปแบบของของคำสั่ง ALTER TABLE มี 2 แบบ คือ

- ALTER TABLE ที่ใช้ในการเพิ่มคอลัมน์
- ALTER TABLE ที่ใช้ในการเปลี่ยนชื่อคอลัมน์

คำสั่ง ALTER TABLE เป็นคำสั่งที่ใช้ในการแก้ไขปรับปรุงโครงสร้างตาราง เมื่อจำเป็นที่ต้องปรับปรุงจากโครงสร้างเดิมตามที่ได้กำหนดไว้ตั้งแต่สร้างตารางในครั้งแรก คำสั่ง ALTER TABLE มีรูปแบบดังนี้

ALTER TABLE <table name>

Database update(<column_name> data type [SIZE]);

| | |
|---|---|
| ALTER TABLE | เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเปลี่ยนแปลงโครงสร้างตาราง |
| table name | ชื่อตารางที่จะเปลี่ยนแปลง |
| Database update | คำสั่งการเปลี่ยนแปลง |
| column_name | ชื่อคอลัมน์ |
| data type [SIZE] | ชนิดข้อมูลและขนาดของข้อมูล |

ตัวอย่าง ถ้าต้องการเปลี่ยนแปลงโครงสร้างตารางโดยการการเพิ่มคอลัมน์ลงไปบนโครงสร้างตารางเดิมจะใช้คำสั่งดังนี้

ALTER TABLE SALESPEOPLE ADD SALESTAB_FAX CHAR(15);

ผลของคำสั่งจะทำให้ตารางพนักงานขายมีคอลัมน์ SALESTAB_FAX ทีมีชนิดข้อมูลเป็น charมีความยาว 15 ตัวอักษรเพิ่มขึ้น

ตัวอย่าง ถ้าต้องการเปลี่ยนแปลงโครงสร้างตารางโดยการเปลี่ยนชื่อคอลัมน์จะใช้คำสั่งดังนี้

ALTER TABLE SALESPEOPLE RENAME ADDRESS TO COUNTRY;

ผลของคำสั่งจะทำให้ตารางพนักงานขาย ที่เดิมมีคอลัมน์ชื่อ ADDRESS ต้องเปลี่ยนชื่อเป็น COUNTRY แทน

**7. ข้อแตกต่างระหว่างคีย์หลักและดัชนี**

คีย์หลักได้แก่ คอลัมน์ 1 คอลัมน์หรือหลายคอลัมน์ที่ทำให้แต่ละแถวในตารางข้อมูลมีค่าของข้อมูลที่ไม่ซ้ำกัน(unique) เช่น คอลัมน์รหัสลูกค้า( CUSNO) ของตารางลูกค้า( CUSTOMERTAB) ซึ่งใช้แทนรหัสลูกค้าแต่ละคนจะมีรหัสประจำตัวไม่ซ้ำกัน คีย์หลักเป็นพื้นฐานในการเชื่อมโยงกันระหว่างตารางและควบคุมความถูกต้องครบถ้วนสมบูรณ์(integrity) ในการตรวจสอบความซ้ำกันของข้อมูลระหว่างที่ทำการป้อนข้อมูลหรือกำหนดข้อมูลใหม่ให้กับตาราง ส่วนดัชนีเป็นการสร้างโดยการเลือกคอลัมน์ใดคอลัมน์หนึ่งหรือหลายคอลัมน์จากตารางขึ้นมาเป็นดัชนี โดยในตารางหนึ่งๆ สามารถมีดัชนีได้หลายดัชนี คอลัมน์ที่จะเลือกเป็นดัชนีควรจะมีค่าของข้อมูลไม่ซ้ำกัน (unique)ในแต่ละแถว ถ้าเลือกคอลัมน์ที่เป็นดัชนีที่สามารถมีค่าว่างได้จะทำให้ DBMSไม่สามารถนำดัชนีที่เป็นค่าว่างนั้นไปค้นหาข้อมูลในฐานข้อมูลได้ การสร้างดัชนีก็เพื่อเป็นตัวนำทางในการสืบค้นข้อมูลให้เร็วขึ้น

**8. การสร้างดัชนี**

ดัชนี (Index) เป็นส่วนที่สำคัญมากต่อฐานข้อมูลเชิงสัมพันธ์ ดัชนีมีความสำคัญคือ ช่วยเพิ่มความสามารถในการค้นหาข้อมูลได้เร็วยิ่งขึ้น โดยดัชนีที่ถูกสร้างขึ้นในแต่ละแถวจะถูกเก็บเป็นตารางแยกจากตารางข้อมูล ซึ่งจะเป็นการสะดวกในการค้นหาข้อมูลในแต่ละแถว DBMS สามารถทำการค้นหาข้อมูลในตารางดัชนี เมื่อพบดัชนีที่ต้องการจะชี้นำไปยังตารางข้อมูลนั้นๆ ซึ่งถ้าตารางข้อมูลใดไม่มีการสร้างดัชนีไว้การค้นหาข้อมูลในตารางนั้นจะต้องทำการค้นหาแบบเรียงลำดับจากแถวแรกจนถึงแถวสุดท้าย นอกจากนี้ดัชนียังช่วยในการตรวจสอบและควบคุมไม่ให้มีข้อมูลเดียวกันหลายแถวซ้ำกันในตารางได้อย่างอัตโนมัติ โดยดัชนีสามารถช่วยให้ผู้ใช้หาข้อมูลแต่ละแถวตามที่กำหนดเฉพาะเจาะจงตามต้องการได้โดยอัตโนมัติ

ในการค้นหาข้อมูลใดข้อมูลหนึ่งในตารางข้อมูล ถ้าไม่มีดัชนีในการค้นหาจะทำให้เสียเวลาในการค้นหาพอสมควร ดัชนีจะเป็นตัวนำทางในการค้นหา เมื่อสร้างดัชนีจากคอลัมน์หนึ่งของตาราง DBMS จะเก็บคอลัมน์นั้นเรียงลำดับที่เหมาะสมของคอลัมน์นั้นไว้ สมมติว่าตารางลูกค้ามีข้อมูลป้อนไว้หลายรายการ และต้องการหาลูกค้าหมายเลข 2999 เนื่องจากไม่ได้เรียงลำดับแถวตามหมายเลขลูกค้าไว้ ปกติโปรแกรมจะต้องค้นหาไปที่ละแถวจนตลอดทั้งตารางเพื่อหาค่าลูกค้าหมายเลข 2999 ในคอลัมน์ CUSNO อย่างไรก็ดีถ้ามีดัชนีอยู่ในคอลัมน์ CUSNO โปรแกรมก็จะตรงไปที่หมายเลข 2999 ในดัชนีเลย การสร้างดัชนีจะทำให้การค้นหาข้อมูลเร็วขึ้น แต่การสร้างดัชนีก็เปลืองพื้นที่ในหน่วยความจำ

การสร้างดัชนีมีรูปแบบคำสั่งของการสร้างดังนี้

```
CREATE INDEX <index name>
    ON <table name>(<column>name>[,<column name>]..);
```

CREATE INDEXเป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการสร้างดัชนี

| | |
|---|---|
| index name | ชื่อดัชนี |
| table name | ชื่อตารางที่จะสร้างดัชนี |

ตัวอย่าง ถ้าตารางลูกค้าเป็นตารางที่พนักงานขายอ้างถึงบ่อยที่สุดเพื่อถามหาลูกค้าของตนเองแล้ว ก็ควรสร้างดัชนีขึ้นในคอลัมน์พนักงาน( SALENO) ของตารางลูกค้า จะใช้คำสั่งดังนี้

CREATE UNIQUE INDEX CLIENTGROUP ON CUSTOMERSTAB(SALENO);

จากคำสั่ง "UNIQUE" เป็นการระบุว่าดัชนี( index) ที่สร้างขึ้นในคอลัมน์ CLIENTGROUP ซึ่งในคอลัมน์นี้จะมีค่าที่ซ้ำกันไม่ได้ และจะใช้คอลัมน์ SALENO เป็นข้อมูลในการค้นหา ผลของคำสั่งตารางลูกค้าจะมี CLIENTGROUP เป็นดัชนี( index) ในการค้นหาข้อมูล โดยเรียงลำดับตามข้อมูลในคอลัมน์ SALENO ของตารางลูกค้า index คอลัมน์CLIENTGROUP ที่สร้างขึ้นนี้จะไม่ถูกเก็บไว้ในตารางลูกค้า แต่จะถูกเก็บไว้แยกต่างหากในหน่วยความจำของเครื่องคอมพิวเตอร์

**9. การลบดัชนีของตาราง**

เมื่อต้องการจบดัชนีที่สร้างขึ้น ก็สามารถทำได้ด้วยคำสั่ง DROP INDEX แล้วตามด้วยชื่อดัชนีที่ต้องการลบ โดยคำสั่งการลบดัชนีมีรูปแบบทั่วไปดังนี้

DROP INDEX <index name>;

DROP INDEX เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการลบดัชนี

index name ชื่อดัชนี

ในการลบดัชนีออกไปจะไม่มีผลกระทบกับรายละเอียดในคอลัมน์ต่าง ๆ ภายในตารางแต่อย่างไร เพราะดัชนีที่สร้างขึ้นไม่ได้บรรจุไว้ในตาราง

ตัวอย่าง ถ้าต้องการลบดัชนีชื่อ CLIENTGROUP ใช้คำสั่งดังนี้

DROP INDEX CLIENTGROUP ;

ผลของคำสั่งนี้จะทำให้ดัชนีชื่อ CLIENTGROUP ที่เดิมเป็นดัชนีของตารางลูกค้าได้ถูกลบออกไป ทำให้ตารางลูกค้าไม่มีดัชนีดังกล่าว

## ตอนที่ 6.2 การบันทึกข้อมูล การปรับปรุงข้อมูล การลบข้อมูลและการเรียกค้นข้อมูลอย่างง่าย

**หัวเรื่อง**

เรื่องที่ 6.2.1 การบันทึกข้อมูล การปรับปรุงข้อมูล การลบข้อมูล

เรื่องที่ 6.2.2 การเรียกค้นข้อมูลอย่างง่าย

**แนวคิด**

1. การบันทึกข้อมูล การปรับปรุงข้อมูลและการลบข้อมูลเป็นส่วนหนึ่งของภาษาสำหรับการจัดการข้อมูล ภาษาสำหรับการจัดการข้อมูลเป็นภาษาที่ช่วยในการจัดการข้อมูลภายในโครงสร้างตารางที่สร้างขึ้น การบันทึกข้อมูลตารางใดตารางหนึ่ง อาจทำได้โดยป้อนข้อมูลใหม่หรือการคัดลอกข้อมูลจากตารางหนึ่งไปอีกตารางหนึ่ง การปรับปรุงข้อมูลเป็นการเปลี่ยนค่าของคอลัมน์หรือข้อมูลของแถวในตารางฐานข้อมูล การลบข้อมูลเป็นการลบข้อมูลออกจากตารางโดยโครงสร้างของตารางยังปรากฏอยู่
2. การเรียกค้นข้อมูลจากตารางเป็นการนำข้อมูลจากฐานข้อมูลโดยมีเงื่อนไขของการค้นข้อมูลหลายรูปแบบ การเรียกค้นข้อมูลสามารถค้นได้จากตารางเดียวหรือจากหลายตารางก็ได้ที่ได้มาจากการเรียกค้นข้อมูลนี้ เรียกว่าเป็นการสอบถามข้อมูล การเรียกค้นข้อมูลสามารถเปรียบเทียบตามโอเปอร์เรเตอร์สัมพันธ์ โอเปอร์เรเตอร์บูลีน โอเปอร์เรเตอร์พิเศษ

**วัตถุประสงค์**

หลังจากศึกษาตอนที่ 6.2 แล้ว นักศึกษาสามารถ

1. บอกวิธีการบันทึกข้อมูลได้
2. บอกวิธีการปรับปรุงข้อมูลได้
3. บอกวิธีการลบข้อมูลในตารางได้

### เรื่องที่ 6.2.1การบันทึกข้อมูล การปรับปรุงข้อมูลและการลบข้อมูล

ในระบบฐานข้อมูล การบันทึกข้อมูล การปรับปรุงข้อมูลและการลบข้อมูลถือเป็นสิ่งสำคัญ ในภาษา SQL มีภาษาสำหรับการจัดการข้อมูล (Data manipulation Language : DML) ซึ่งเป็นภาษาที่ใช้ในการบันทึกข้อมูล การปรับปรุงข้อมูลและการลบข้อมูล ภาษาสำหรับการจัดการข้อมูล เป็นส่วนประกอบหนึ่งในภาษา SQL โดยภาษาสำหรับการจัดการข้อมูลใช้สำหรับจัดการข้อมูลภายในตารางของฐานข้อมูล ในการใช้คำสั่งที่เป็นภาษาสำหรับนิยามข้อมูลของภาษา SQL เช่น CREATE TABLE จะทำให้ได้โครงสร้างตารางว่างๆ ที่ยังไม่มีข้อมูลใดๆเก็บอยู่ คำสั่งในภาษาสำหรับการจัดการข้อมูลจะเป็นคำสั่งที่ช่วยในการจัดการข้อมูลภายในโครงสร้างตารางที่สร้างขึ้น ตัวอย่างของคำสั่งในภาษาสำหรับการจัดการข้อมูล จะเป็นคำสั่งการปรับปรุงข้อมูล ได้แก่ การเพิ่มข้อมูล (INSERT) การปรับปรุง (UPDATE) และ การลบข้อมูล (DELETE)ซึ่งจะกล่าวต่อไป และคำสั่งการเรียกค้นข้อมูลได้แก่คำสั่ง(SELECT)ซึ่งจะกล่าวในเรื่องต่อไป

คำสั่งที่ใช้ในการปรับปรุงข้อมูลของภาษา SQL คือ การเพิ่มข้อมูล (INSERT) การปรับปรุงข้อมูล (UPDATE) และ การลบข้อมูล (DELETE) เป็นคำสั่งในภาษาการจัดการข้อมูล เมื่อโครงสร้างหลักของตารางได้ถูกกำหนดขึ้นเรียบร้อยแล้ว ก็จะทำการบันทึกข้อมูลลงในตารางหลักหรืออาจทำการปรับปรุง หรือลบข้อมูลในภายหลัง คำสั่งทั้ง 3 นี้ เมื่อดำเนินการในภาษา SQL จะไม่แสดงผลลัพธ์ออกมาทางหน้าจอ แต่ผลของคำสั่งจะมีผลต่อข้อมูล ผู้ใช้สามารถดูผลของการใช้คำสั่งในการเพิ่มข้อมูล การปรับปรุงและการลบข้อมูล โดยใช้คำสั่งการเรียกค้นข้อมูล(SELECT)

1. **คำสั่งการเพิ่มข้อมูล**

คำสั่งการเพิ่มข้อมูลในตารางจะใช้คำสั่ง INSERT จะมีอยู่ 2 รูปแบบคือ การเพิ่มข้อมูลเข้าไปทีละแถว และ การเพิ่มข้อมูลโดยการดึงกลุ่มข้อมูลด้วยคำสั่งค้นหาข้อมูล

1.1 คำสั่งการเพิ่มข้อมูลทีละแถวโดยระบุข้อมูลที่จะ INSERTเข้าไปโดยตรง รูปแบบของคำสั่งเป็นดังนี้

INSERT INTO <tablename>[(column 1, column 2,...)]

VALUE(<value1,value2, ...>);

INSERT INTO เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเพิ่มข้อมูล

tablename ชื่อตารางที่จะเพิ่มข้อมูล

column 1, column 2,...คอลัมที่ต้องการเพิ่มข้อมูล

value1,value2, ค่าข้อมูลของแต่ละคอลัมภ์ที่ต้องการเพิ่ม

ตัวอย่าง ถ้าต้องการจะใส่ข้อมูลทุกคอลัมน์ลงในตารางลูกค้า

INSERT INTO SALESTAB

VALUES( 1001, “Chaiwat”, “Bangkok”,0.12);

ผลของคำสั่งนี้ จะมีข้อมูลปรากฏในทุกคอลัมน์ในตารางพนักงานขายดังนี้

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |

ตัวอย่าง ถ้าต้องการจะใส่ข้อมูลบางคอลัมน์ เช่น ชื่อเมือง Bangkok ชื่อลูกค้า Arlee และหมายเลขลูกค้า 2001 ลงในตารางลูกค้า ใช้คำสั่งดังนี้

INSERT INTO CUSTOMERSTAB(ADDRESS,CUSNAME,CUSNO)

VALUES( 'Bangkok','Arlee', 2001);

ผลของคำสั่งในตารางลูกค้า จะทำให้คอลัมน์ ADDRESS มีค่าเป็น Bangkok คอลัมน์ CUSNAME จะมีค่าเป็น Arlee คอลัมน์ CUSNO จะมีค่าเป็น 2001ดังนี้

| CUSNO | CUSNAME | ADDRESS | RATING | SALENO |
|---|---|---|---|---|
| 2001 | Arlee | Bangkok | | |

จะเห็นว่าไม่ได้ใส่ค่าในคอลัมน์ RATING และ SALENO ไว้ ดังนั้นทั้งสองคอลัมน์นี้จะมีค่าเป็น NULL โดยอัตโนมัติ

1.2 คำสั่งการเพิ่มข้อมูลโดยการดึงกลุ่มข้อมูลด้วยคำสั่งค้นหาข้อมูล ในภาษา SQL สามารถใช้คำสั่ง INSERT ในการนำค่าหรือหาค่าจากตารางหนึ่งแล้วไปใส่ไว้ในอีกตาราหนึ่งได้ โดยได้ค่านั้นมาจากการสอบถามข้อมูล รูปแบบเป็นดังนี้

INSERT INTO <table name>[(column 1, column 2,...)]

SELECT statement;

INSERT INTO เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเพิ่มข้อมูล

tablename ชื่อตารางที่จะเพิ่มข้อมูล

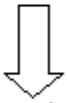
SELECT statement ประโยคคำสั่ง SELECTที่ต้องการข้อมูลอีกตารางหนึ่ง

ตัวอย่าง ถ้าต้องการใส่ข้อมูลพนักงานลงในตาราง BANGKOKSTAFF โดยข้อมูลที่จะใส่ลงไปนั้นได้มาจากตารางพนักงานขายที่อาศัยอยู่ใน “Bangkok“

```
INSERT INTO BANGKOKSTAFF
    SELECT *
    FROM SALESTAB
    WHERE ADDRESS = 'Bangkok';
```

ตารางพนักงานขาย

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |
| 1003 | Ternjai | Nonthaburi | 0.10 |



ตาราง BANGKOKSTAFF ที่พนักงานขายอยู่ในเมือง Bangkok

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1004 | Benjawan | Bangkok | 0.11 |

ผลของคำสั่งนี้จะทำให้ได้ข้อมูลพนักงานที่อยู่ในเมือง Bangkok ( ADDRESS = 'Bangkok') ทั้งหมดไปใส่ไว้ในตาราง BANGKOKSTAFF โดยตาราง BANGKOKSTAFF ได้ถูกสร้างไว้แล้วด้วยคำสั่ง CREATE TABLE ในการสร้างตาราง BANGKOKSTAFF จะต้องสร้างให้มี 4 คอลัมน์และมีชนิดข้อมูลตรงกับคอลัมน์ของตารางพนักงานขาย (โดยไม่จำเป็นต้องมีชื่อคอลัมน์เหมือนกัน)

### 2. คำสั่งปรับปรุงแถวข้อมูล

หลังจากที่ป้อนข้อมูลเข้าไปเก็บไว้ในตารางแล้ว กรณีที่ต้องการปรับปรุงแก้ไขข้อมูลสามารถทำได้ด้วยภาษา SQL การปรับปรุงแถวข้อมูลเป็นการปรับปรุงหรือแก้ไขค่าคอลัมน์ ซึ่งในคำสั่งปรับปรุงข้อมูลอาจมีมากกว่า 1 คอลัมน์ในแถวทุกแถวที่มีเงื่อนไขสอดคล้องกับที่ระบุไว้หลังคำว่า WHERE

รูปแบบของคำสั่งปรับปรุงแถวข้อมูลมีดังนี้

UPDATE <table name> SET <column 1>[, column 2,...] = <expression |sunquery>
[WHERE<condition>];

| | |
|---|---|
| UPDATE | เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการปรับปรุงข้อมูล |
| table name | ชื่อตารางที่ต้องการปรับปรุง |
| SET <column > | ชื่อคอลัมน์ที่ต้องการปรับปรุง |
| expression | ค่าข้อมูลที่ต้องการปรับปรุง |
| WHERE<condition> | เงื่อนไขในการปรับปรุง |

ตัวอย่าง ถ้าต้องการเปลี่ยนค่า RATING ของลูกค้าทั้งหมดในตารางลูกค้าให้เป็น 200 จะต้องป้อนคำสั่งดังนี้

```
UPDATE CUSTOMERSTAB
    SET RATING = 200;
```

ผลของคำสั่งจะทำให้คอลัมน์ RATING ของตารางลูกค้ามีค่าเป็น 200 ทุกแถวและเมื่อเข้าไปดูข้อมูลในตารางลูกค้าจะปรากฏข้อมูลดังนี้

**ตารางลูกค้า(CUSTOMERSTAB)**

| CUSNO | CUSNAME | ADDRESS | RATING | SALENO |
|---|---|---|---|---|
| 2001 | Arlee | Bangkok | 100 | 1001 |
| 2002 | Tanachote | Pratum | 200 | 1003 |
| 2003 | Tawatchai | Puket | 200 | 1002 |
| 2004 | Amporni | Ubon | 300 | 1002 |
| 2006 | Surasit | Bangkok | 100 | 1001 |
| 2008 | Jintana | Puket | 300 | 1007 |
| 2007 | Siriwan | Pratum | 100 | 1004 |

**ตารางลูกค้าที่มีค่า RATING =200**

| CUSNO | CUSNAME | ADDRESS | RATING | SALENO |
|---|---|---|---|---|
| 2001 | Arlee | Bangkok | 200 | 1001 |
| 2002 | Tanachote | Pratum | 200 | 1003 |
| 2003 | Tawatchai | Puket | 200 | 1002 |
| 2004 | Amporni | Ubon | 200 | 1002 |
| 2006 | Surasit | Bangkok | 200 | 1001 |
| 2008 | Jintana | Puket | 200 | 1007 |
| 2007 | Siriwan | Pratum | 200 | 1004 |

หากต้องการจะเปลี่ยนเฉพาะแถวใดแถวหนึ่งเท่านั้นก็สามารถทำได้ดังนี้

ตัวอย่าง ถ้าต้องการจะเปลี่ยนค่า RATING ให้กับลูกค้าทั้งหมด ที่มีหมายเลขประจำตัวพนักงานขาย(SALENO) เป็น 1001 ให้มีค่า RATINGเป็น 200

```
UPDATE CUSTOMERSTAB
    SET RATING = 200
    WHERE SALENO = 1001;
```

ผลของคำสั่งจะทำให้ตารางลูกค้าเดิมเปลี่ยนเป็นตารางใหม่ในตารางใหม่นี้ ลูกค้าทั้งหมดที่มีหมายเลขประจำตัวเป็น 1001 จะมีค่า RATING เป็น 200 ดังนี้

**ตารางลูกค้า(CUSTOMERSTAB)**

| CUSNO | CUSNAME | ADDRESS | RATING | SALENO |
|---|---|---|---|---|
| 2001 | Arlee | Bangkok | 100 | 1001 |
| 2002 | Tanachote | Pratum | 200 | 1003 |
| 2003 | Tawatchai | Puket | 200 | 1002 |
| 2004 | Amporni | Ubon | 300 | 1002 |
| 2006 | Surasit | Bangkok | 100 | 1001 |
| 2008 | Jintana | Puket | 300 | 1007 |
| 2007 | Siriwan | Pratum | 100 | 1004 |

**ตารางลูกค้าที่หมายเลขประจำตัวพนักงานเป็น 1001 ให้มี RATING = 200**

| CUSNO | CUSNAME | ADDRESS | RATING | SALENO |
|---|---|---|---|---|
| 2001 | Arlee | Bangkok | 200 | 1001 |
| 2002 | Tanachote | Pratum | 200 | 1003 |
| 2003 | Tawatchai | Puket | 200 | 1002 |
| 2004 | Amporni | Ubon | 300 | 1002 |
| 2006 | Surasit | Bangkok | 200 | 1001 |
| 2008 | Jintana | Puket | 300 | 1007 |
| 2007 | Siriwan | Pratum | 100 | 1004 |

### 3. คำสั่งการลบข้อมูลทั้งแถว

คำสั่งในการลบแถวข้อมูล เป็นคำสั่งที่ใช้ในการลบแถวข้อมูลทุกแถวที่มีเงื่อนไขสอดคล้องกับที่ระบุไว้หลัง WHERE คำสั่งการลบข้อมูลมีรูปแบบทั่วไปดังนี้

```
DELETE  FROM <table name>
    [WHERE<condition>];
```

DELETE FROM เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการลบข้อมูล

table name ชื่อตารางที่ต้องการลบข้อมูล

WHERE<condition> เงื่อนไขในการลบข้อมูล

ตัวอย่าง ถ้าต้องการลบแบบมีเงื่อนไข เช่น ต้องการลบพนักงานขายชื่อ Ternjai ซึ่งมีหมายเลขพนักงาน (SALENO)=1003 ออกจากตารางจะใช้คำสั่งว่า

```
DELETE  FROM SALESTAB
    WHERE SALENO = 1003;
```

ผลของคำสั่งจากตารางพนักงานขายเดิมจะทำให้ได้ตารางใหม่ดังนี้

ตารางพนักงานขาย(SALESTAB)

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |
| 1003 | Ternjai | Nonthaburi | 0.10 |

⇩

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |

โดยปกติแล้วการลบข้อมูลจะกระทำการลบเพียงบางแถวของตารางเท่านั้น การลบแถวต่าง ๆ ออกจากตารางด้วยคำสั่งในการปรับปรุงคือคำสั่ง DELETE คำสั่งนี้จะลบแถวทั้งแถวแต่ไม่สามารถลบค่าเพียงคอลัมน์ใดคอลัมน์หนึ่ง

ตัวอย่าง ถ้าต้องการลบรายละเอียดทั้งหมดของตารางพนักงานขายจะต้องป้อนคำสั่งต่อไปนี้

```
DELETE FROM SALESTAB;
```

ในตารางพนักงานขายก็จะว่างไม่มีค่าใดๆ อยู่แต่ตารางยังปรากฏอยู่ ถ้าต้องการตารางออกไปจะใช้คำสั่ง DROP TABLE

ตัวอย่าง ถ้าต้องการลบตาราง SALESTABใช้คำสั่งดังนี้

```
DROP TABLE SALESTAB;
```

**เรื่องที่6.2.2 การเรียกค้นข้อมูลอย่างง่าย**

การเรียกค้นข้อมูลเป็นการสอบถามข้อมูลหรือ "Query" โดยการนำข้อมูลจากฐานข้อมูลมาแสดงออกทางจอภาพ การสอบถามข้อมูลนี้ในภาษา SQL ใช้คำสั่ง SELECT โดยการเรียกค้นข้อมูลจะเป็นไปตามเงื่อนไขที่ผู้ใช้ข้อมูลระบุ

1. **การเรียกค้นดูทุกคอลัมน์ในตาราง**

คำสั่ง SELECT แบบง่ายมีรูปแบบดังนี้

```
SELECT *
FROM <table name>;
```

SELECT * เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูลทุกคอลัมน์

FROM เป็นการกำหนดว่าให้เรียกดูข้อมูล ได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

การเรียกดูข้อมูลสามารถเรียกดูได้มากกว่า 1 คอลัมน์ขึ้นไป โดยถ้ามีมากกว่า 1 คอลัมน์ แต่ละคอลัมน์จะต้องคั่นด้วยเครื่องหมายคอมม่า(,) และถ้าต้องการดูทุกคอลัมน์จะใช้เครื่องหมาย ดอกจัน(*) หลัง SELECT การใช้คำสั่ง SELECT จะใช้ควบคู่กับคำสั่ง FROM เสมอในการเลือกตาราง

การใช้คำสั่ง SELECT ในการเรียกค้นข้อมูลทุกคอลัมน์ในตารางจะใช้เครื่องหมายดอกจัน(*) ตามหลังคำสั่ง SELECT

ตัวอย่าง ตาราง CHECKS

| CHECK# | PAYEE | AMOUNT | REMARKS |
|---|---|---|---|
| 1. | Malee Benjanee | 150 | Have sons next time |
| 2. | Reading R.R | 24534 | Train to Chiangmai |
| 3. | Malee Benjanee | 20032 | Cellular Phone |
| 4. | Surasit Utities | 98 | Gas |
| 5. | Jintana $ Mitree | 150 | Groesries |
| 6. | Cash | 25 | Wild Night Out |
| 7. | Benjawan Gas | 251 | Gas |

ตัวอย่าง ถ้าต้องการดูทุกคอลัมน์ในตารางก็จะใช้เครื่องหมายดอกจัน(*) แทนรายการคอลัมน์ได้ทั้งหมดได้ดังนี้

```
select * from checks;
```

ผลลัพธ์

| CHECK# | PAYEE | AMOUNT | REMARKS |
|---|---|---|---|
| 1. | Malee Benjanee | 150 | Have sons next time |
| 2. | Reading R.R | 24534 | Train to Chiangmai |
| 3. | Malee Benjanee | 20032 | Cellular Phone |
| 4. | Surasit Utities | 98 | Gas |
| 5. | Jintana $ Mitree | 150 | Groesries |
| 6. | Cash | 25 | Wild Night Out |
| 7. | Benjawan Gas | 251 | Gas |

จากคำสั่ง select * จะเป็นการบอกให้นำข้อมูลทั้งจากตาราง CHECKS มา แสดง (from checks) โดยลำดับตามคอลัมน์ในฐานข้อมูล

**2. การเรียกค้นข้อมูลเฉพาะคอลัมน์ใด ๆในตารางและการเปลี่ยนลำดับคอลัมน์**

การใช้คำสั่ง SELECT ในการเรียกค้นข้อมูลเฉพาะคอลัมน์ที่สนใจทำได้โดยใส่เฉพาะคอลัมน์ที่ต้องการดูในส่วนของคำสั่ง SELECT มีรูปแบบดังนี้

SELECT <column 1, column 2,...>

FROM <table name>;

SELECT เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูล

column 1, column 2,...เป็นคอลัมน์ที่ต้องการเรียกค้น

FROM เป็นการกำหนดว่าให้เรียกดูข้อมูลได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

การเลือกบางคอลัมน์

ตัวอย่าง ถ้าต้องการแสดงข้อมูลบางคอลัมน์จะใช้ เช่น ถ้าต้องการดูคอลัมน์ CHECK# และ AMOUNTใช้คำสั่งดังนี้

SELECT CHECK#, amount from checks;

ผลลัพธ์

| CHECK# | AMOUNT |
|---|---|
| 1 | 150 |
| 2 | 24534 |
| 3 | 20032 |
| 4 | 98 |

| | |
|---|---|
| 5 | 150 |
| 6 | 25 |
| 7 | 251 |

จะเห็นได้ว่าเราสามารถใช้ทั้งอักษรตัวใหญ่และตัวเล็กปนกันในคำสั่ง ซึ่งอักษรตัวใหญ่และตัวเล็กจะไม่มีความแตกต่างกัน

ตัวอย่าง ถ้าต้องแสดงข้อมูลโดยการเปลี่ยนลำดับคอลัมน์ของข้อมูล จะใช้คำสั่งดังนี้

```
SELECT PAYEE, REMARKS, AMOUNT, CHECK#
FROM   checks;
```

ผลลัพธ์

| PAYEE | REMARKS | AMOUNT | CHECK# |
|---|---|---|---|
| Malee Benjanee | Have sons next time | 150 | 1 |
| Reading R.R. | Train to Chiangmai | 24534 | 2 |
| Malee Benjanee | Cellular Phone | 20032 | 3 |
| Surasit Utilities | Gas | 98 | 4 |
| Jintana $ Mitree | Groesries | 150 | 5 |
| Cash | Wild Night Out | 25 | 6 |
| Benjawan Gas | Gas | 251 | 7 |

**3.การเรียกค้นข้อมูลกับคำสั่ง** Distinction

จากตาราง CHECKS ถ้าต้องการดูคอลัมน์ AMOUNT เป็นดังนี้

```
select amount from checks;
```

ผลลัพธ์

| AMOUNT |
|---|
| 150 |
| 24534 |
| 20032 |
| 98 |
| 150 |

| 25 |
|---|
| 251 |

จากผลลัพธ์จะเห็นว่าในคอลัมน์ AMOUNT มีข้อมูลที่ซ้ำกันอยู่คือ 150 ถ้าใช้คำสั่ง Distinct ในคำสั่ง SELECT จะทำให้ข้อมูลที่ซ้ำกันนั้นแสดงออกมาเพียงครั้งเดียวดังนี้

```
select DISTINCT amount from checks;
```

ผลลัพธ์

| AMOUNT |
|---|
| 25 |
| 251 |
| 98 |
| 150 |
| 20032 |
| 24534 |

จะเห็นว่าจะแสดงข้อมูลออกมาเพียง 6 แถวเท่านั้น

3.**การใช้คำสั่ง** SELECT **กับ** WHERE

```
SELECT <column 1, column 2,...>
FROM <table name>
[WHERE<condition>];
```

SELECT เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูล

column 1, column 2,...คอลัมน์ที่ต้องการเรียกค้น

FROM เป็นการกำหนดว่าให้เรียกดูข้อมูล ได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

WHERE<condition> ส่วนของคำสั่งที่บอกเงื่อนไขที่จะใช้ในการค้นหาข้อมูล

การใช้ WHERE ในคำสั่ง SELECT จะช่วยให้สามารถสืบค้นข้อมูลได้อย่างเจาะจงมากกว่า เช่น ถ้าใช้เฉพาะ SELECT อย่างเดียวจะได้ข้อมูลทั้งหมด ตัวอย่างเช่น

ตัวอย่าง ตาราง BIKES

| NAME | FRAMESIZE | COMPOSITION | MILESRIDDEN | TYPE |
|---|---|---|---|---|
| TREK 2300 | 22.5 | CARBON FIBER | 3500 | RACING |

| | | | | |
|---|---|---|---|---|
| BURLEY | 22 | STEEL | 2000 | TANDEM |
| GIANT | 19 | STEEL | 1500 | COMMUTER |
| FUJI | 20 | STEEL | 500 | TOURING |
| SPECIALIZED | 16 | STEEL | 100 | MOUNTAIN |
| CANNONDALE | 22.5 | ALUMINUM | 3000 | RACING |

ถ้าต้องการจะดูเฉพาะข้อมูลของ "BURLEY" เท่านั้นเราจะต้องใช้ คำสั่ง WHERE ดังนี้

```
SELECT * FROM BIKES
WHERE NAME = 'BURLEY' ;
```

ผลลัพธ์

| NAME | FRAMESIZE | COMPOSITION | MILESRIDDEN | TYPE |
|---|---|---|---|---|
| BURLEY | 22 | STEEL | 2000 | TANDEM |

**4.โอเปอเรเตอร์**

การเรียกค้นข้อมูลอย่างมีเงื่อนไขตามหลักของภาษา SQL จะอยู่หลังคำสั่ง WHERE ซึ่งสามารถเปรียบเทียบตามโอเปอเรเตอร์ ในภาษา SQL อาจแบ่งโอเปอเรเตอร์ ได้เป็น 4 กลุ่ม คือ

1. โอเปอเรเตอร์คณิตศาสตร์(Arithmetic Operators)
2. โอเปอเรเตอร์เปรียบเทียบ(Comparison Operators)
3. โอเปอเรเตอร์อักขระ(Character Operators)
4. โอเปอเรเตอร์ตรรกะ(Logical Operators)

4.1. โอเปอเรเตอร์คณิตศาสตร์(Arithmetic Operators) ได้แก่ operators ที่เป็น plus (+) minus (-), divide (/), multiply (*), and modulo (%)

- โอเปอเรเตอร์ Plus (+) เป็นคำสั่งที่ใช้รวมค่า 2 ค่าเข้าด้วยกัน ดังตัวอย่างต่อไปนี้

จากตาราง PRICE มีรายละเอียดดังนี้

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |
| CHEESE | 89 |

| | |
|---|---|
| APPLES | 23 |

ตัวอย่าง ถ้าในคำสั่งในคอลัมน์ WHOLESALE ต้องการ บวก 15 เข้าไป ผลลัพธ์ที่ได้จะแสดงค่าของ WHOLESALE ที่บวก 15 เข้าไปโดยคอลัมน์ แต่จะแสดงเพียงชั่วคราวที่หน้าจอเท่านั้น โดยไม่มีผลต่อข้อมูลของคอลัมน์ WHOLESALE ในตาราง PRICE คอลัมน์ WHOLESALE ในตาราง PRICE จะมีค่าเหมือนเดิม

และจากคอลัมน์ WHOLESALE+15 สามารถให้แสดงผลหน้าจอเป็นชื่อคอลัมน์อื่นได้ โดยถ้าต้องการให้ WHOLESALE +15 และให้แสดงผลเป็นคอลัมน์ RETAIL จะใช้คำสั่งดังนี้

```
SELECT ITEM, WHOLESALE, (WHOLESALE + 0.15) RETAIL
FROM PRICE;
```

ผลลัพธ์

| ITEM | WHOLESALE | RETAIL |
|---|---|---|
| TOMATOES | 34 | 49 |
| POTATOES | 51 | 66 |
| BANANAS | 67 | 82 |
| TURNIPS | 45 | 60 |
| CHEESE | 89 | 104 |
| APPLES | 23 | 38 |

นอกจากนี้เราสามารถแสดงข้อมูลในคอลัมน์ให้มีชื่อใหม่ตามที่ต้องการได้ เช่น ต้องการให้แสดงข้อมูลในคอลัมน์ ITEM ในตาราง PRICE ให้แสดงออกมาทางหน้าจอเป็นชื่อคอลัมน์ PRODUCE ได้ โดยในคำสั่งยังไม่ใส่เครื่องหมายคอมม่าระหว่าง ITEM และ PRODUCE เพื่อให้ภาษาSQL เข้าใจได้ว่าจะแสดงคอลัมน์ ITEM เป็นคอลัมน์ PRODUCE ดังตัวอย่างต่อไปนี้

```
SELECT ITEM PRODUCE, WHOLESALE, WHOLESALE * 0.25 RETAIL
FROM PRICE:
```

ผลลัพธ์

| PRODUCE | WHOLESALE | RETAIL |
|---|---|---|
| TOMATOES | 34 | 59 |
| POTATOES | 51 | 76 |
| BANNANAS | 67 | 92 |
| TURNIPS | 45 | 70 |
| CHEESE | 89 | 114 |

| | | |
|---|---|---|
| APPLES | 23 | 48 |

- โอเปอร์เรเตอร์ Minus (-) คำสั่ง Minus ใช้ได้เป็น 2 กรณีคือ

1. การเปลี่ยนเครื่องหมายจากบวกเป็นลบและจากลบเป็นบวก

2. การนำข้อมูลของคอลัมน์หนึ่งไปลบออกจากข้อมูลของอีกคอลัมน์หนึ่ง

1.การเปลี่ยนเครื่องหมายจากบวกเป็นลบและจากลบเป็นบวก เช่น รายละเอียดของตาราง HILOW เป็นดังนี้

ตัวอย่างตาราง HILOW

| STATE | HIGHTEMP | LOWTEMP |
|---|---|---|
| CA | 120 | -50 |
| FL | 110 | 20 |
| LA | 101 | 15 |
| ND | 99 | -70 |
| NE | 100 | -60 |

ตัวอย่าง ถ้าต้องการให้คอลัมน์ HIGHTEMP และ LOWTEMP มีค่าจากลบเป็นบวก จากบวกเป็นลบ และแสดงหน้าจากคอลัมน์ HIGHTEMP เป็นคอลัมน์ LOWS และคอลัมน์ LOWTEMP เป็นคอลัมน์ HIGHS ใช้คำสั่งดังนี้

```
SELECT STATE, - HIGHTEMP LOWS, -LOWTEMP HIGHS
FROM HILOW;
```

ผลลัพธ์

| STATE | LOWS | HIGHS |
|---|---|---|
| CA | -120 | 50 |
| FL | -110 | -20 |
| LA | -101 | -15 |
| ND | -99 | 70 |
| NE | -100 | 60 |

2.การนำข้อมูลของคอลัมน์หนึ่งไปลบออกจากข้อมูลของอีกคอลัมน์หนึ่ง

ตัวอย่าง ถ้าต้องการนำคอลัมน์ HIGHTEMP ลบออกจากคอลัมน์ LOWTEMPแล้วนำผลลัพธ์ที่ได้แสดงในคอลัมน์ DIFFERENCE โดยใช้คำสั่งดังนี้

```
SELECT STATE,HIGHTEM LOWS,
LOWTEMP HIGHS, (HIGHTEMP - LOWTEMP ) DIFFERENCE
FROM HILOW;
```

ผลลัพธ์

| STATE | LOWS | HIGHS | DIFFERENCE |
|---|---|---|---|
| CA | -50 | 120 | 170 |
| FL | 20 | 110 | 90 |
| LA | 15 | 99 | 84 |
| ND | -70 | 101 | 171 |
| NE | -60 | 100 | 160 |

- โอเปอร์เรเตอร์ Divide (/)เป็นคำสั่งที่ใช้ในการหารข้อมูลดังตัวอย่าง เช่น ตัวอย่างตาราง PRICE

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANNANAS | 67 |
| TURNIPS | 45 |
| CHEESE | 89 |
| APPLES | 23 |

เมื่อต้องการให้คอลัมน์ WHOLESALE ถูกหารด้วย 2 และแสดงในคอลัมน์ SALEPRICE จะใช้คำสั่งดังนี้

```
SELECT ITEM, WHOLESALE, (WHOLESALE/2) SALEPRICE
FROM PRICE;
```

ผลลัพธ์

| ITEM | WHOLESALE | SALEPRICE |
|---|---|---|
| TOMATOES | 34 | 17 |
| POTATOES | 51 | 25.5 |
| BANNANAS | 67 | 33.5 |
| TURNIPS | 45 | 22.5 |

| | | |
|---|---|---|
| CHEESE | 89 | 44.5 |
| APPLES | 23 | 11.5 |

จากตัวอย่างคอลัมน์ SALEPRICE เป็นผลลัพธ์ที่เกิดจากการนำคอลัมน์ WHOLESALE มาหารด้วย 2

- โอเปอร์เรเตอร์ Multiply (*)เป็นคำสั่งที่ใช้ในคูณค่าของข้อมูลในคอลัมน์

ตัวอย่างตาราง PRICE

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |
| CHEESE | 89 |
| APPLES | 23 |

ตัวอย่าง ถ้าต้องการคูณคอลัมน์ WHOLESALE ด้วย 0.9 ให้ได้ผลลัพธ์เป็นข้อมูลในคอลัมน์ใหม่ที่ชื่อ NEWPRICE จะใช้คำสั่งดังนี้

```
SELECT ITEM, WHOLESALE, WHOLESALE * 0.9 NEWPRICE
FROM PRICE;
```

ผลลัพธ์

| ITEM | WHOLESALE | NEWPRICE |
|---|---|---|
| TOMATOES | 34 | 30.6 |
| POTATOES | 51 | 45.9 |
| BANANAS | 67 | 60.3 |
| TURNIPS | 45 | 40.5 |
| CHEESE | 89 | 80.1 |
| APPLES | 23 | 20.7 |

- โอเปอเรเตอร์ Modulo (%)เป็นคำสั่งที่ได้ผลลัพธ์เป็นเศษที่ได้จากการหาร

ตัวอย่าง ตาราง REMAINS

| NUMERATOR | DENOMENATOR |
|---|---|
| 10 | 5 |

| | |
|---|---|
| 8 | 3 |
| 23 | 9 |
| 40 | 17 |
| 1024 | 16 |
| 85 | 34 |

ตัวอย่าง ถ้าต้องการสร้างคอลัมน์ REMAINDER ที่มีข้อมูลที่เกิดจากการนำข้อมูลในคอลัมน์ NUMERATOR หารด้วยข้อมูลในคอลัมน์ DENOMINATOR เหลือเศษในการหารเท่าไรแล้วนำค่าที่ได้ไปเก็บไว้ในคอลัมน์ REMAINDER ดังคำสั่งต่อไปนี้

```
SELECT NUMERATOR,
DENOMINATOR,
NUMERATOR%DENOMINATOR REMAINDER
FROM REMAINS;
```

ผลลัพธ์

| NUMERATOR | DENOMENATOR | REMAINDER |
|---|---|---|
| 10 | 5 | 0 |
| 8 | 3 | 2 |
| 23 | 9 | 5 |
| 40 | 17 | 6 |
| 1024 | 16 | 0 |
| 85 | 34 | 17 |

นอกจากการใช้เครื่องหมาย % ในคำสั่ง Modulo แล้วในภาษา SQL ยังใช้ฟังก์ชัน MOD แทนเครื่องหมาย %ได้ซึ่งจะให้ผลลัพธ์เช่นเดียวกันดังคำสั่งต่อไปนี้

```
SELECT NUMERATOR,
DENOMINATOR,
MOD(NUMERATOR, DENOMONATOR) REMAINDER
FROM  REMAINS;
```

4.2. โอเปอรเรเตอร์ เปรียบเทียบ (Comparison Operators เป็น Operator ที่จะให้ค่าออกมา 3 ค่า คือ ถูก (TRUE) ผิด (FALSE) ไม่รู้ (Unknow) การไม่รู้หมายถึง ถ้านำข้อมูลที่มีค่าไปเปรียบเทียบกับข้อมูลที่เป็น NULL ตัวเปรียบเทียบจะให้ค่าไม่รู้

ตัวอย่างในตาราง PRICE

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |
| CHEESE | 89 |
| APPLES | 23 |
| ORANGES | |

ตัวอย่าง ถ้าต้องการดูว่า ITEM ที่ไม่มีค่า WHOLESALE หรือค่า WHOLESALEเป็นค่าว่างจะใช้คำสั่งดังนี้

```
SELECT *
FROM PRICE
WHERE WHOLESALE IS NULL;
```

หรือ

```
SELECT *
FROM PRICE
WHERE WHOLESALE = NULL;
```

ผลลัพธ์

| ITEM | WHOLESALE |
|---|---|
| ORANGES | |

4.3 โอเปอเรเตอร์ตัวอักษร(Character Operators) ตัวโอเปอเรเตอร์ LIKE เป็นการค้นหาข้อมูลของคอลัมน์ที่เก็บข้อมูลประเภทตัวอักษรเท่านั้น โดยไม่ทราบค่าข้อมูลทั้งหมดที่จะค้นหา หรือรู้เพียงบางตัวอักษรเท่านั้น โอเปอร์เรเตอร์ LIKE จะระบุต่อท้ายชื่อคอลัมน์ที่เป็นเงื่อนไข โดยจะใช้สัญลักษณ์ที่เป็นตัวค้นหาช่วยในการค้นหาข้อมูลที่เรียกว่า วิน การ์ด (WILD Card) สัญลักษณ์ดังกล่าวประกอบด้วย % และ _(เครื่องหมายขีดเส้นใต้) โดยข้อมูลบางส่วนที่ใช้ในการค้นหาพร้อมกับสัญลักษณ์ทั้งสองนี้ จะต้องมีเครื่องหมาย ‘ ‘ กำกับเสมอ ความหมายของสัญลักษณ์ทั้งสองเป็นดังนี้คือ

- สัญลักษณ์ % ใช้แทนจำนวนอักษรได้หลายตัว เช่น พนักงานขายที่ขึ้นต้นด้วยตัว T จะเขียนเงื่อนไขว่า WHERE SALENAME LIKE ‘T%’

- สัญลักษณ์ _ ใช้แทนจำนวนที่ไม่ทราบค่า 1 ตัว เช่น พนักงานขายที่มีชื่อขึ้นต้น S และมีความยาว 7 ตัวอักษร เช่น WHERE SALENAME LIKE 'S_____'

```
SELECT PAYEE, AMOUNT, REMARKS
FROM CHECKS
WHERE PAYEE LIKE ('CA%');
```

ผลลัพธ์

| PAYEE | AMOUNT | REMARKS |
|---|---|---|
| Cash | 25 | Wild Night Out |
| Cash | 60 | Trip to Saraburi |
| Cash | 34 | Trip to Nonthaburi |

กับคำสั่ง LIKE ดังตัวอย่างข้างต้น เปรียบเทียบกับคำสั่ง WITH

```
SELECT PAYEE, AMOUNT, REMARKS
FROM CHECKS
WHERE PAYEE STARING WITH ('Ca');
```

ผลลัพธ์

| PAYEE | AMOUNT | REMARKS |
|---|---|---|
| Cash | 25 | Wild Night Out |
| Cash | 60 | Trip to Saraburi |
| Cash | 34 | Trip to Nonthaburi |

จะเห็นว่าได้ผลลัพธ์เช่นเดียวกัน

ตัวอย่าง ถ้าต้องการให้แสดงคอลัมน์ PAYEE ที่ขึ้นต้นด้วยอักษร Ca หรือ คอลัมน์ REMARKS ที่ขึ้นต้นด้วยอักษร G จะใช้คำสั่งดังนี้

```
SELECT PAYEE, AMOUNT, REMARKS
        FROM CHECKS
        WHERE PAYEE STARTING WITH('Ca')
        OR
        REMARKS LIKE 'G%';
```

ผลลัพธ์

| PAYEE | AMOUNT | REMARKS |
|---|---|---|

| | | |
|---|---|---|
| Surasit Utilities | 98 | Gas |
| Jintana $ Mitree | 150 | Groceries |
| Cash | 25 | Wild Night Out |
| Benjawan Gas | 251 | Gas |
| Cash | 60 | Trip to Saraburi |
| Cash | 34 | Trip to Nonthaburi |
| Benjawan Gas | 1575 | Gas |

4.4 โอเปอรเรเตอร์ตรรกะ (Logical Operator) เป็นตัวโอเปอเรเตอร์ที่ใช้ในการเปรียบเทียบ เชื่อมโยงค่า 2 ค่า

ตัวอย่าง ตาราง VACATION

| LASTNAME | EMPLOYEENUM | YEARS | LEAVETAKEN |
|---|---|---|---|
| ARLEE | 101 | 2 | 4 |
| AMPORNI | 104 | 5 | 23 |
| JINTANA | 107 | 8 | 45 |
| BOLIVAR | 233 | 4 | 80 |
| TANACHOTE | 210 | 15 | 100 |
| TAWATCHI | 211 | 10 | 78 |

จากตัวอย่างสมมุติว่าบริษัทให้พนักงานแต่ละคนสามารถหยุดงานได้ โดยพิจารณาจากจำนวนปีที่พนักงานทำงาน พนักงานจะหยุดงานได้ 12 วัน ในอายุการทำงานแต่ละปี ถ้าต้องการหาว่าพนักงานที่มีชื่อตัวหน้าว่า B และยังสามารถหยุดงานได้อีก 50 วัน จะใช้คำสั่ง ดังนี้

```
SELECT * LASTNAME, YEARS * 12 – LEAVETAKEN REMAINING
        FROM VACATION
        WHERE LASTNAME LIKE 'B%'
        AND
        YEARS * 12 – LEAVETAKEN > 50;
```

ผลลัพธ์

| LASTNAME | REMAINING |
|---|---|
| JINTANA | 51 |
| TANACHOTE | 80 |

จากคำสั่งเราจะใช้คำสั่ง YEARS * 12 – LEAVETAKEN เพื่อหาวันหยุดที่พนักงานยังเหลือ

- ตัวโอเปอเรเตอร์ AND เป็นตัวโอเปอเรเตอร์ ที่ใช้เชื่อมโยงค่า 2 ค่า โดยถ้าค่าหนึ่งเป็น TURE อีกค่าหนึ่งเป็น TURE จะให้ค่า TRUE ออกมา แต่ถ้าค่าหนึ่งเป็น TURE อีกค่าหนึ่งเป็น FALSE จะให้ค่าเป็น FALSE ตัวอย่างเช่น ถ้าต้องการหาว่ามีพนักงานที่ทำงานมากกว่า 5 ปี และเหลือวันหยุดมากกว่า 50% จะใช้คำสั่ง ดังนี้

```
SELECT LASTNAME WORKAHOLICS
    FROM VACATION
    WHERE YEARS >=5
    AND
    ((YEARS *12) – LEAVETAKEN)/(YEARS *12) >=0.50;
```

ผลลัพธ์

| WORKAHOLICS |
|---|
| AMPORNI |
| JINTANA |

ตัวโอเปอเรเตอร์ OR ใช้ในการเปรียบเทียบถ้าสิ่งที่นำมาเปรียบเทียบสิ่งใดสิ่งหนึ่งเป็นจริง จะได้ผลลัพธ์ออกมาเป็นจริง

```
SELECT LASTNAME WORKAHOLICS
    FROM VACATION
    WHERE YEARS >=5
    OR
    ((YEARS *12) – LEAVETAKEN)/(YEARS *12)>=0.50;
```

ผลลัพธ์

| WORKAHOLICS |
|---|
| ARLEE |
| AMPORNI |
| JINTANA |
| TANACHOTE |
| TAWATCHI |

- ตัวโอเปอเรเตอร์ NOT ในการเปรียบเทียบถ้าสิ่งที่นำมาเปรียบเทียบเป็นจริงจะได้ผลลัพธ์ออกมาเป็นเท็จ แต่ถ้าสิ่งที่นำมาเปรียบเทียบเป็นเท็จผลที่ได้ออกมาจะเป็นจริง ดังตัวอย่าง

```
SELECT *
        FORM VACATION
        WHERE LASTNAME NOT LIKE 'B%';
```

ผลลัพธ์

| LASTNAME | EMPLOYEENUM | YEARS | LEAVETAKEN |
|---|---|---|---|
| ARLEE | 101 | 2 | 4 |
| COSTALES | 211 | 10 | 78 |

- ตัวโอเปอเรเตอร์ NOT ยังใช้กับ NULL ได้ ถ้านำ not กับ null มารวมกันแล้วจะใช้สำหรับค่าที่ไม่ว่าง

ตัวอย่างตารางPRICE

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |
| CHEESE | 89 |
| APPLES | 23 |
| ORANGES | |

ตัวอย่าง ถ้าต้องการให้แสดงเฉพาะข้อมูลที่มีค่าเท่านั้นจะใช้คำสั่ง not nullมาร่วมกับ null เพื่อแสดงเฉพาะข้อมูลที่มีค่าดังคำสั่งต่อไปนี้

```
SELECT *
        FROM PRICE
        WHERE wholesale is not null;
```

ผลลัพธ์

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |

| | |
|---|---|
| TURNIPS | 45 |
| CHEESE | 89 |
| APPLES | 23 |

- โอเปอร์เรเตอร์ IN และ BETWEEN ตัวโอเปอเรเตอร์ IN เป็นการกำหนดเซ็ตของสิ่งที่ต้องการค้นหา โดยการกำหนดชื่อของสมาชิกเซ็ตลงไปในวงเล็บและแยกจากกันด้วยคอมม่า

```
SELECT *
        FROM FRIENDS
        WHERE STATE= 'Chiangmai'
        OR
        STATE = 'Bangkok'
        OR
        STATE = 'Puket';
```

ผลลัพธ์

| LASTNAME | FIRSTNAME | PHONE | ADDRESS |
|---|---|---|---|
| SIRIWAN | ARLEEWAN | 555-6666 | Bangkok |
| SURASIT | CHAIYO | 555-6767 | Puket |
| WICHAI | AMPORNWAN | 555-3116 | Chiangmai |

```
SELECT *
        FROM FRIENDS
        WHERE STATE IN ('Chiangmai  ',  'Bangkok ', 'Puket ');
```

ผลลัพธ์

| LASTNAME | FIRSTNAME | PHONE | ADDRESS |
|---|---|---|---|
| SIRIWAN | ARLEEWAN | 555-6666 | Bangkok |
| SURASIT | CHAIYO | 555-6767 | Puket |
| WICHAI | AMPORNWAN | 555-3116 | Chiangmai |

- ตัวโอเปอเรเตอร์ BETWEEN...AND... เป็นการกำหนดเงื่อนไขของคอลัมน์ระหว่างค่าสองค่า ซึ่งค่าสองที่ออยู่ระหว่างคำสั่ง BETWEEN...AND...นั้นจะมีความหมายว่าเท่ากับหรือมากกว่าและเท่ากับหรือน้อยกว่า

```
SELECT *
```

```
FROM PRICE
WHERE WHOLESALE >25
AND
WHOLESALE < 75;
```

ผลลัพธ์

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |

ถ้าใช้คำสั่ง BETWEEN จะเป็นดังนี้

```
SELECT *
FROM PRICE
WHERE WHOLESALE BETWEEN 25 AND 75;
```

ผลลัพธ์

| ITEM | WHOLESALE |
|---|---|
| TOMATOES | 34 |
| POTATOES | 51 |
| BANANAS | 67 |
| TURNIPS | 45 |

## ตอนที่ 6.3 การเรียกค้นข้อมูล

**หัวเรื่อง**

เรื่องที่ 6.3.1 ฟังก์ชัน

เรื่องที่ 6.3.2 การเรียกดูข้อมูลในรูปแบบต่างๆ

**แนวคิด**

1. ในการเรียกดูข้อมูลอาจจะทำการสรุปค่าของข้อมูล โดยการรวม การหาค่าเฉลี่ย การนับหรือการหาค่าสูงสุดหรือต่ำสุดหรือค่าอื่นๆ โดยใช้ฟังก์ชัน ฟังก์ชันสามารถนำไปใช้ในการเรียกใช้ข้อมูลยังสามารถเรียกใช้ข้อมูลแบบมีเงื่อนไขได้
2. การเรียกดูข้อมูลในรูปแบบต่างๆ สามารถเรียกดูข้อมูลได้จากหลายตาราง ซึ่งมีคำสั่งต่างๆมากมาย การเรียกดูโยการรวมตารางและการเรียกดูโดยโดยใช้ GROUP BY HAVING และอื่นๆ

**วัตถุประสงค์**

หลังจากศึกษาตอนที่ 6.3 แล้ว นักศึกษาสามารถ

1. บอกวิธีการเรียกค้นข้อมูลตามเงื่อนไขและการเรียกใช้ฟังก์ชันรวมได้
2. บอกวิธีการเรียกค้นข้อมูลจากตารางเดียวหรือจากหลายตารางได้

### เรื่องที่ 6.3.1 ฟังก์ชัน

ฟังก์ชันที่ใช้ในภาษา SQLเป็นฟังก์ชัน ซึ่งเก็บประจำไว้กับภาษา SQL ภาษา SQL มีฟังก์ชันอยู่ 6 ประเภทคือ

1.ฟังก์ชันในการรวม (Aggregate functions)

2.ฟังก์ชันวันและเวลา (Date and tune functions)

3.ฟังก์ชันคณิตศาสตร์ (Arithmetic functions)

4.ฟังก์ชันตัวอักขระ (Character functions)

5.ฟังก์ชันการแปลง (Conversion functions)

6.ฟังก์ชันอื่นๆ (Miscellaneous functions)

1.**ฟังก์ชันในการรวม**(Aggregate Functions)

เป็นกลุ่มฟังก์ชันที่ให้ผลของคำสั่งออกมาเพียง 1 คอลัมน์ ฟังก์ชันในการรวม(Aggregate Functions) เป็นกลุ่มฟังก์ชันที่ใช้กับข้อมูลที่เป็นตัวเลข ได้แก่ COUNT,SUM,AVG,MAXและ MIN

การใช้ฟังก์ชันในการรวมค่าต่าง ๆในภาษา SQL ดำเนินตามคำสั่งที่มีฟังก์ชันในการรวมค่า ผลของคำสั่งจะแสดงค่าเพียงค่าเดียว ฟังก์ชันเหล่านี้ได้แก่

COUNT เป็นคำสั่งที่สามารถใช้กับตารางหรือคอลัมน์ใด ๆ เพื่อนับจำนวนของแถวหรือคอลัมน์ซึ่งมีการใช้งาน 2 แบบดังนี้คือ

COUNT (*) เป็นคำสั่งใช้นับจำนวนแถวทั้งหมดในตารางซึ่งจะรวมจำนวนแถวที่ไม่มีค่า ( NULL) ด้วย

COUNT (DISTINCT คอลัมน์) เป็นคำสั่งใช้นับจำนวนแถวในตาราง จะไม่รวมค่าซ้ำและตำแหน่งที่ไม่มีค่า( NULL)

SUM เป็นคำสั่งการหาผลรวมของคอลัมน์ใดคอลัมน์หนึ่ง

AVG เป็นคำสั่งการหาค่าเฉลี่ยของข้อมูลในคอลัมน์ใดคอลัมน์หนึ่งโดยในคอลัมน์ที่ไม่มีค่าใดบรรจุอยู่ (NULL VALUE)จะไม่นำมาบรรจุอยู่ในการคำนวณ การใช้ฟังก์ชั่น AVG จะนำค่าทุกตัวในคอลัมน์มาคำนวณรวมทั้งตัวที่มีค่าซ้ำกันด้วย(ถ้าไม่ต้องการนำค่านั้นมาคำนวณสามารถใช้ DISTINCT ได้เช่น AVG (DISTINCT ชื่อคอลัมน์) เพื่อหาค่าเฉลี่ยโดยไม่ต้องนำค่าซ้ำกันมาคำนวณ

MAX เป็นคำสั่งในการหาค่าสูงสุดของข้อมูลของคอลัมน์ใดคอลัมน์หนึ่ง

MIN เป็นคำสั่งในการหาค่าต่ำสุดของข้อมูลของคอลัมน์ใดคอลัมน์หนึ่ง

1.1ฟังก์ชัน COUNT (X) เป็นฟังก์ชันที่ใช้ในการนับจำนวนแถวในคอลัมน์ (X)

ตัวอย่างตารางTEAMGAME

| NAME | AB | HITS | WALKS | SINGLES | DOUBLES | TRIPLES | HR |
|---|---|---|---|---|---|---|---|
| JONES | 145 | 45 | 34 | 31 | 8 | 1 | 5 |
| DONKNOW | 175 | 65 | 50 | 50 | 10 | 1 | 4 |
| WORLEY | 157 | 49 | 35 | 35 | 8 | 3 | 3 |
| DAVID | 187 | 70 | 48 | 48 | 4 | 0 | 17 |
| HAMHOCKER | 50 | 12 | 10 | 10 | 2 | 0 | 0 |
| CASEY | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

ตัวอย่างถ้าต้องการนับจำนวนแถวทั้งหมดในตาราง TEAMGAMEโดยนับเฉพาะแถวที่ HIT หารด้วย AB แล้วมีค่าน้อยกว่า 0.35 จะใช้คำสั่งดังนี้

```
SELECT COUNT(*)
FROM TEAMGAME
WHERE HITS/AB < .35;
```

ผลของคำสั่ง จะได้ผลลัพธ์เป็นจำนวนแถวที่ HIT หารด้วย AB แล้วมีค่าน้อยกว่า 0.35

| COUNT (*) |
|---|

| 4 |
|---|

ตัวอย่าง ถ้าต้องการให้แสดงคอลัมน์ที่นับได้ชื่อ NUM_BELOW_350

```
SELECT COUNT(*) NUM_BELOW_350
FROM TEAMGAME
WHERE HITS/AB < .35;
```

ผลของคำสั่ง

| NUM_BELOW_350 |
|---|
| 4 |

ตัวอย่าง ถ้าต้องการนับจำนวนคนที่มีเงื่อนไขให้ HIT หารด้วย AB แล้วมีค่าน้อยกว่า 0.35

```
SELECT COUNT(NAME) NUM_BELOW_350
FROM TEAMGAME
WHERE HITS/AB < .35;
```

ผลของคำสั่ง

| NUM_BELOW_350 |
|---|
| 4 |

ตัวอย่าง ถ้าต้องการนับว่าข้อมูลในตาราง TEAMGAME มีจำนวนทั้งหมดกี่แถวจะใช้

```
SELECT COUNT(*)
FROM TEAMGAME;
```

ผลของคำสั่ง

| COUNT(*) |
|---|
| 6 |

1.2 ฟังก์ชัน SUM (X) เป็นฟังก์ชันที่ใช้ในการหาค่ารวมของคอลัมน์ (X) ที่เก็บข้อมูลประเภทตัวเลข

ตัวอย่าง ถ้าต้องการหาผลรวมของคอลัมน์ SINGLES จากตาราง TEAMGAME โดยให้แสดงคอลัมน์ของผลรวมที่ได้ในชื่อ TOTAL....SINGLES

```
SELECT SUM(SINGLES) TOTAL_SINGLES
FROM TEAMGAME;
```

ผลของคำสั่ง

| TOTAL_SINGLES |
|---|
| 174 |

ตัวอย่าง ถ้าต้องการหาผลรวมของคอลัมน์ SINGLES, DOUBLES, TRIPLES, HR จากตาราง TEAMGAME โดยให้แสดงคอลัมน์ของผลรวมที่ได้ในชื่อ TOTAL.SIGLES, TOTAL_DOUBLES, TOTAL_TRIPLES, TOTAL_HR ตามลำดับ

```
SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES) TOTAL_DOUBLES,
SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้จะทำการรวมคะแนนทั้งหมดในคอลัมน์ SINGLES, DOUBLES, TRIPLES, HR แล้วแสดงออกมาเป็นคอลัมน์ TOTAL_SIGLES TOTAL_DOUBLES, TOTAL_TRIPLES, TOTAL_HR ตามลำดับดังนี้

| TOTAL_SINGLES | TOTAL_DOUBLES | TOTAL_TRIPLES | TOTAL_HR |
|---|---|---|---|
| 174 | 32 | 5 | 29 |

ตัวอย่าง ถ้าต้องการหาผลรวมของคอลัมน์ SINGLES, DOUBLES, TRIPLES, HR จากตาราง TEAMGAME ที่มีเงื่อนไขว่า HITS หารด้วย AB มากกว่าหรือเท่ากับ .300 โดยให้แสดงคอลัมน์ของผลรวมที่ได้ในชื่อ TOTAL_SIGLES, TOTAL_DOUBLES, TOTAL_TRIPLES, TOTAL_HR ตามลำดับ

```
SELECT SUM(SINGLES) TOTAL_SINGLES, SUM(DOUBLES) TOTAL_DOUBLES,
SUM(TRIPLES) TOTAL_TRIPLES, SUM(HR) TOTAL_HR
FROM TEAMGAME;
WHERE HITS/AB > = .300;
```

ผลของคำสั่งที่ได้จะทำให้รวมคะแนนทั้งหมดในคอลัมน์ต่าง ๆ เฉพาะแถวที่มีค่า HITS หารด้วย AB มากกว่าหรือเท่ากับ .300 แล้วแสดงผลในชื่อ TOTAL_SIGLES, TOTAL_DOUBLES, TOTAL_TRIPLES, TOTAL_HR ตามลำดับ

| TOTAL_SINGLES | TOTAL_DOUBLES | TOTAL_TRIPLES | TOTAL_HR |
|---|---|---|---|
| 164 | 30 | 5 | 29 |

ตัวอย่าง ถ้าต้องการผลรวมในคอลัมน์ NAME

```
SELECT SUM(NAME)
FROM TEAMGAME;
```

ผลของคำสั่งจะเกิด ERROR ขึ้นเนื่องจากในคอลัมน์ NAME มีประเภทของข้อมูลเป็นตัวอักขระ ฟังก์ชัน SUM จะใช้กับตัวเลขเท่านั้น ถ้าใช้ SUM กับตัวอักษรจะเกิด ERROR ดังตัวอย่าง

```
ERROR:
ORA-01722: invalid number
```

No rows selected

1.3 ฟังก์ชัน AVG (X) เป็นฟังก์ชันที่ใช้ในการหาค่าเฉลี่ยของคอลัมน์ (X) ที่เก็บข้อมูลประเภทตัวเลข

ตัวอย่าง ถ้าต้องการหาเฉลี่ยของ HITS ให้แสดงในชื่อ HIT_AVERAGE

```
SELECT AVG(HITS) HITS_AVERAGE
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้จะแสดงค่าเฉลี่ยของ HITS ที่เกิดจากการนำค่าในแถวต่าง ๆ ในคอลัมน์ HIT มาบวกกันแล้วหารด้วยจำนวนแถว คือ 6

| HITS_AVERAGE |
|---|
| 40.166666 |

1.4 ฟังก์ชัน MAX (X) เป็นฟังก์ชันที่ใช้ในการคำนวณหาค่าสูงสุดของคอลัมน์ (X)

ตัวอย่าง ถ้าต้องการหาว่าในคอลัมน์ HITS มีค่าสูงสุดเท่าใด

```
SELECT MAX(HITS)
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้จะได้ว่า HITS มีค่าสูงสุดคือ 70 ดังนี้

| MAX(HITS) |
|---|
| 70 |

ตัวอย่าง ถ้าต้องการหาว่าใครเป็นผู้ที่ได้ HITS สูงที่สุดโดยใช้คำสั่งดังนี้

```
SELECT  NAME
FROM TEAMGAME
WHERE HITS = MAX(HITS);
```

ผลของคำสั่งจะเกิด ERROR เนื่องจากฟังก์ชันในการรวม (Aggregate function) มาใช้ในเงื่อนไขอนุประโยค WHERE ไม่ได้จะเกิด ERROR ขึ้นจากตัวอย่าง WHERE HITS = MAX(HITS); MAX(HITS) จะมาใช้ในเงื่อนไขอนุประโยค WHERE ไม่ได้ จากคำถามข้อนี้จะสามารถใช้ได้กับคำสั่ง CROUP BY และ HAVING ที่จะได้ศึกษาต่อไป

```
ERROR at line 3:
ORA – 00934: group function is not allowed here
```

ตัวอย่าง การใช้ MAX กับประเภทของข้อมูลที่เป็นตัวอักขระ

```
SELECT  MAX (NAME)
FROM TEAMGAME;
```

ผลของคำสั่งจากคำสั่งจะทำการหาชื่อของผู้ที่มีอักษร (A-Z) ตามลำดับใครที่มีอักษรลำดับมากที่สุดจะถูกแสดงออกมาเพียงแถวเดียว

| MAX(NAME) |
|---|
| WORLEY |

1.5 ฟังก์ชัน MIN (X) เป็นฟังก์ชันที่ใช้ในการหาค่าต่ำสุดของคอลัมน์ (X)

ตัวอย่าง ถ้าต้องการหาค่าต่ำสุดในคอลัมน์ AB

```
SELECT MIN(AB)
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้จะได้ค่าต่ำสุดในคอลัมน์ AB ดังนี้

| MIN (AB) |
|---|
| 1 |

ตัวอย่าง การใช้ MIN กับประเภทของข้อมูลที่เป็นตัวอักขระ

```
SELECT MIN(NAME)
FROM TEAMGAME;
```

ผลของคำสั่งจากคำสั่งจะทำการหาชื่อของผู้ที่มีอักษร (A-Z) ตามลำดับใครที่มีอักษรลำดับน้อยที่สุดจะถูกแสดงออกมาเพียงแถวเดียว

| MIN (NAME) |
|---|
| CASEY |

ตัวอย่าง ถ้าต้องการหาค่าต่ำสุดและสูงสุดในคอลัมน์ AB

```
SELECT MIN(AB), MAX(AB)
FROM TEAMGAME;
```

ผลของคำสั่งจะได้ค่าต่ำสุดและสูงสุดในคอลัมน์ AB ดังนี้

| MIN (AB) | MAX (AB) |
|---|---|
| 1 | 187 |

1.6 ฟังก์ชัน VARIANC (X) เป็นฟังก์ชันในการหาค่าส่วนเบี่ยงเบนมาตราฐานยกกำลัง2(S2) ในคอลัมน์ X

ตัวอย่าง ถ้าต้องการหาค่า VARIANC ของ คอลัมน์ HITS

```
SELECT VARIANCE(HITS)
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้จะได้ค่า VARIANCE ในคอลัมน์ HITS

| VARIANCE(HITS) |
|---|
| 802.96667 |

ตัวอย่าง ถ้าต้องการหาค่า VARIANCE ของคอลัมน์ NAME

```
SELECT VARIANCE(NAME)
FROM TEAMGAME;
```

ผลของคำสั่งจะเกิด ERRORขึ้นเนื่องจากฟังก์ชัน VARIANCE ไม่สามารถใช้กับข้อมูลที่เป็นตัวอักษรได้ดังนี้

```
ERROR:
ORA-01722: invalid number
No rows selected
```

1.7 ฟังก์ชัน STDDEV (X) หรือฟังก์ชันส่วนเบี่ยงเบนมาตราฐาน ส่วนเบี่ยงเบนมาตราฐาน คือ การหาค่ารากที่สองของผลรวมของความแตกต่างระหว่างข้อมูลดิบกับค่าเฉลี่ย ยกกำลังสอง (sum of squares ของผลต่าง) หารด้วยจำนวนข้อมูลทั้งหมดของคอลัมน์ X

ตัวอย่าง ถ้าต้องการหาส่วนเบี่ยงเบนมาตรฐานของคอลัมน์ HITS

```
SELECT STDDEV(HITS)
FROM TEAMGAME;
```

ผลของคำสั่งจะได้ส่วนเบี่ยงเบนมาตรฐานของคอลัมน์ HITS ดังนี้

| STDDEV(HITS) |
|---|
| 28.336666 |

```
SELECT  STDDEV(NAME)
FROM TEAMGAME;
```

ผลของคำสั่งจะเกิด ERROR ได้เนื่องจากฟังก์ชัน STDDEV ไม่สามารถใช้กับข้อมูลที่เป็นตัวอักษรได้

```
ERROR:
ORA-01722: invalid number
no rows selected
```

ตัวอย่าง ถ้าต้องการนับจำนวนแถวในคอลัมน์ หาค่าเฉลี่ยหาค่าสูงสุด ต่ำสุด หาค่าส่วนเบี่ยงเบนมาตรฐาน หาค่าว่าเรียน และหาผลรวมของคอลัมน์ AB

```
SELECT COUNT(AB),
    AVG(AB),
```

```
MIN(AB),
MAX(AB),
STDEV(AB),
VARIANCE(AB),
SUM(AB),
FROM TEAMGAME;
```

ผลของคำสั่งที่ได้ค่าต่าง ๆ ตามลำดับดังนี้

| COUNT(AB) | AVG(AB) | MIN(AB) | MAX(AB) | STDDEV(AB) | VARIANCE(AB) | SUM(AB) |
|---|---|---|---|---|---|---|
| 6 | 119.167 | 1 | 187 | 75.589 | 5712.97 | 715 |

2.**ฟังก์ชันวันและเวลา** (Date and tune functions)

เป็นกลุ่มฟังก์ชันที่แสดงข้อมูลออกมาเป็นวันและเวลา

ตัวอย่าง ตารางPROJECT

| TASK | STARTDATE | ENDDATE |
|---|---|---|
| KICKOFF MTG | 01-APR-2001 | 01-APR-2001 |
| TECH SURVEY | 02-APR-2001 | 01-MAY-2001 |
| USER MTGS | 15-MAY-2001 | 30-MAY-2001 |
| DESIGN WIDGET | 01-JUN-2001 | 30-JUN-2001 |
| CODE WIDGET | 01-JUL-2001 | 02-SEP-2001 |
| TESTING | 03-SEP-2001 | 17-JAN-2002 |

2.1 ฟังก์ชัน ADD_MONTHS (X,Y) เป็นฟังก์ชันที่ต้องการบวกจำนวนเดือน (Y) เข้าไปในข้อมูลคอลัมน์ X

ตัวอย่าง ถ้าต้องการให้เลื่อนเวลาในคอลัมน์ ENDDATE ให้มีกำหนดเวลาเพิ่มขึ้นอีก 2 เดือน โดยให้ผลลัพธ์แสดงคอลัมน์ TASK, STARTDATE และคอลัมน์ ENDDATE ให้แสดงเป็นคอลัมน์ ORIGINALEND ส่วนกำหนดเวลาที่บวกเพิ่มไปอีก 2 เดือน ให้แสดงในคอลัมน์ ADD_MONTH

```
SELECT TASK,STARTDATE, ENDDATE ORIGINAL_END,
```

ADD_MONTHS(ENDDATE,2)

FROM PROJECT;

ผลของคำสั่งจะได้ ADD_MONTHS เพิ่มมาอีก 1 คอลัมน์ ซึ่งเกิดจากข้อมูลในคอลัมน์ ENDDATE บวกอีก 2 เดือน

| TASK | STARTDATE | ORIGINAL | ADD_MONTH |
|---|---|---|---|
| KICKOFF MTG | 01-APR-2001 | 01-APR-2001 | 01-JUN-2001 |
| TECH SURVEY | 02-APR-2001 | 01-MAY-2001 | 01-JUN-2001 |
| USER MTGS | 15-MAY-2001 | 30-MAY-2001 | 30-JUN-2001 |
| DESIGN WIDGET | 01-JUN-2001 | 30-JUN-2001 | 31-AUG-2001 |
| CODE WIDGET | 01-JUL-2001 | 02-SEP-2001 | 02-NOV-2001 |
| TESTING | 03-SEP-2001 | 17-JAN-2002 | 17-MAR-2002 |

ตัวอย่าง ถ้าต้องการหาว่างานใดบ้างที่มีระยะการทำงานไม่เกิน 1 เดือนจะใช้คำสั่งดังนี้

SELECT TASK, TASKS_SHORTER_THAN_ONE_MONTH

FROM PROJECT

WHERE ADD_MONTHS(STARTDATE, 1) > ENDDATE;

ผลของคำสั่งจะได้คอลัมน์ TASKS_SHORTER_THAN_ONE_MONTH ที่แสดงงานที่มีระยะเวลาการทำงานไม่เกิน 1 เดือน

| TASKS_SHORTER_THAN_ONE_MONTH |
|---|
| KICKOFF MTG |
| TECH SURVEY |
| USER MTGS |
| DESIGN WIDGET |

2.2 ฟังก์ชัน LAST_DAY (X) เป็นฟังก์ชันที่แสดงวันสุดท้ายของเดือนในคอลัมน์ (X)

ตัวอย่าง เช่นต้องการแสดงวันสุดท้ายของเดือนจะใช้คำลั่งดังนี้

SELECT ENDDATE, LAST_DAY(ENDDATE)

FROM PROJECT;

ผลของคำสั่งที่ได้จะแสดงในคอลัมน์ LAST_DAY (ENDDATE) ที่แสดงวันสุดท้ายของเดือนในคอลัมน์ ENDDATE

| ENDDATE | LAST_DAY(ENDDATE) |
|---|---|

| | |
|---|---|
| 01-APR-2001 | 30-APR-2001 |
| 01-MAY-2001 | 31-MAY-2001 |
| 30-MAY-2001 | 31-MAY-2001 |
| 30-JUN-2001 | 30-JUN-2001 |
| 02-SEP-2001 | 30-SEP-2001 |
| 17-JAN-2002 | 31-JAN-2002 |

2.3 ฟังก์ชัน MONTHS_BETWEEN (X,Y) เป็นฟังก์ชันที่คำนวณค่าระหว่าง X และ Y โดยมีหน่วยเป็นเดือน

ถ้าต้องการคำนวณหาค่าระหว่างคอลัมน์ STARIDATE กับคอลัมน์ ENDDATE ว่ามีระยะเวลาห่างกันกี่เดือน

```
SELECT TASK, STARTDATE, ENDDATE, MONTHS_BETWEEN(STARTDATE, ENDDATE)
DURATION
FROM PROJECT;
```

ผลของคำสั่งที่ได้คอลัมน์ DURATION ที่ติดค่าลบเนื่องจากใช้คอลัมน์ STARTDATE ซึ่งมีค่าน้อยกว่าคอลัมน์ ENDDATE เป็นค่าเริ่มต้น

| TASK | STARTDATE | ENDDATE | DURATION |
|---|---|---|---|
| KICKOFF MTG | 01-APR-2001 | 01-APR-2001 | 0 |
| TECH SURVEY | 02-APR-2001 | 01-MAY-2001 | -.9677419 |
| USER MTGS | 15-MAY-2001 | 30-MAY-2001 | -.483871 |
| DESIGN WIDGET | 01-JUN-2001 | 30-JUN-2001 | -.9354839 |
| CODE WIDGET | 01-JUL-2001 | 02-SEP-2001 | -2.032258 |
| TESTING | 03-SEP-2001 | 17-JAN-2002 | -4.451613 |

เป็นการหาค่าเดือนเหมือนดังตัวอย่างข้างต้น แต่จะนำคอลัมน์ ENDDATE มาเป็นค่าเริ่มต้น

```
SELECT TASK, STARTDATE, ENDDATE,
MONTHS_BETWEEN(ENDDATE ,STARTDATE) DURATION
FROM PROJECT;
```

ผลของคำสั่งที่ได้คอลัมน์ DURATION เป็นบวกเพราะคอลัมน์ ENDDATE ที่เป็นค่าเริ่มต้นมีค่ามากกว่าคอลัมน์ STARIDATE

| TASK | STARTDATE | ENDDATE | DURATOPN |
|---|---|---|---|
| KICKOFF MTG | 01-APR-2001 | 01-APR-2001 | 0 |

| | | | |
|---|---|---|---|
| TECH SURVEY | 02-APR-2001 | 01-MAY-2001 | .96774194 |
| USER MTGS | 15-MAY-2001 | 30-MAY-2001 | .48387097 |
| DESIGN WIDGET | 01-JUN-2001 | 30-JUN-2001 | .93548387 |
| CODE WIDGET | 01-JUL-2001 | 02-SEP-2001 | 2.0322581 |
| TESTING | 03-SEP-2001 | 17-JAN-2002 | 4.4516129 |

ตัวอย่าง ถ้าต้องการหาว่าค่าที่เริ่มก่อนวันที่ 15 MAY 2001

```
SELECT *
FROM PROJECT
WHERE MONTHS_BETWEEN('19 MAY 2001',STARTDATE) > 0;
```

ผลของคำสั่งที่ได้จะแสดงงานโครงการที่เริ่มก่อน วันที่ 19 MAY 2001

| TASK | STARTDATE | ENDDATE |
|---|---|---|
| KICKOFF MTG | 01-APR-2001 | 01-APR-2001 |
| TECH SURVEY | 02-APR-2001 | 01-MAY-2001 |
| USER MTGS | 15-MAY-2001 | 30-MAY-2001 |

3.**ฟังก์ชันคณิตศาสตร์** (Arithmetic functions)

เป็นกลุ่มคำสั่งที่เกี่ยวกับการคำนวณทางเลขคณิต

ตัวอย่าง ตารางNUMBERS;

| A | B |
|---|---|
| 3.1415 | 4 |
| -45 | .707 |
| 5 | 9 |
| -57.667 | 42 |
| 15 | 55 |

| | |
|---|---|
| -7.2 | 5.3 |

3.1 ฟังก์ชัน ABS(X)เป็นฟังก์ชันในการหาค่าสมบูรณ์ของ X

ต้องการหาค่าสมบูรณ์ในคอลัมน์ A

```
SELECT ABS(A) ABSOLUTE_VALUE
FROM NUMBERS;
```

ผลของคำสั่ง

| ABSOLUTE_VALUE |
|---|
| 3.1415 |
| 45 |
| 5 |
| 57.667 |
| 15 |
| 7.2 |

3.2 ฟังก์ชัน CEIL(X) and FLOOR(X)

ฟังก์ชัน CEIL (X) เป็นฟังก์ชันที่ให้ค่าตัวเลขจำนวนเต็มที่มีค่ามากว่าหรือเท่ากับค่าในคอลัมน์ (X)

ฟังก์ชัน FLOORเป็นฟังก์ชันที่ให้ค่าตัวเลขจำนวนเต็มที่พิจารณาจากค่าในคอลัมน์ X ถ้าหลังจุดทศนิยมมีค่ามากว่า 5 ก็จะให้ค่าเลขจำนวนเต็มที่มากขึ้น แต่ถ้าหลังจุดทศนิยมมีค่าน้อยกว่า 5 จะให้ค่าตัวเลขที่มีค่าน้อยลง

ตัวอย่าง ถ้าต้องการหาค่าตัวเลขจำนวนเต็มที่มีค่ามากกว่าหรือเท่ากับค่าในคอลัมน์ B

```
SELECT B, CEIL(B) CEILING
FROM NUMBERS;
```

ผลของคำสั่งจะได้คอลัมน์ CEILING ที่แสดงตัวเลขจำนวนเต็มที่มีค่ามากกว่าหรือเท่ากับค่าในคอลัมน์ B

| B | CEILING |
|---|---|
| 4 | 4 |
| .707 | 1 |
| 9 | 9 |

| | |
|---|---|
| 42 | 42 |
| 55 | 55 |
| 5.3 | 6 |

ตัวอย่าง ถ้าต้องการหาค่าตัวเลขจำนวนเต็มในคอลัมน์ A โดยถ้าหลังจุดทศนิยมมีอยู่มากกว่า 5 ก็จะให้ค่าเลขจำนวนเต็มที่มากขึ้น แต่ถ้าหลังจุดทศนิยมมีค่าน้อยกว่า 5 ก็จะให้ค่าตัวเลขที่มีค่าน้อยลง

```
SELECT A, FLOOR(A) FLOOR
FROM NUMBERS;
```

ผลของคำสั่งจะได้คอลัมน์ FLOOR ที่มีค่ามากกว่า

| A | FLOOR |
|---|---|
| 3.1415 | 3 |
| .45 | -45 |
| 5 | 5 |
| -57.667 | -58 |
| 15 | 15 |
| -7.2 | -8 |

3.3 ฟังก์ชัน COS(X), COSH(X), SIN(X), SINH(X), TAN(X), และ TANH(X) เป็นฟังก์ชันทางตรีโกณที่หาค่า cosine,hyperbolic cosine,sine,hyperbolic sine,tangent,hyperbolic tangent ที่มีค่า X เป็นองศาเรเดียน(radians,) โดย 360 degrees = 2 pile radians

ตัวอย่าง ถ้าต้องการหาค่า COS ของมุมในคอลัมน์ A

```
SELECT A, COS(A)
FROM NUMBERS;
```

ผลของคำสั่งจะได้ของ (A) ที่มีค่าดังนี้

| A | COS(A) |
|---|---|
| 3.1415 | -1 |
| -45 | .52532199 |
| 5 | .28366219 |
| -57.667 | .437183 |
| 15 | -.7596879 |
| .7.2 | .60835131 |

3.4 ฟังก์ชัน EXP (X)เป็นฟังก์ชันหาค่า e ยกกำลัง X

ตัวอย่าง ถ้าต้องการหาค่า e ยกกำลังของข้อมูลในคอลัมน์ A

```
SELECT A, EXP(A)
FROM NUMBERS;
```

ผลของคำสั่งจะได้คอลัมน์ EXP(A) ที่เป็นข้อมูลในข้อมูลคอลัมน์ A e ยกกำลังตัวเลข

| A | EXP(A) |
|---|---|
| 3.1415 | 23.138549 |
| -45 | 2.863E-20 |
| 5 | 148.41316 |
| -57.667 | 9.027E-26 |
| 15 | 3269017.4 |
| .7.2 | .00074659 |

3.5 ฟังก์ชัน LN(X) และ LOG(X)

ฟังก์ชัน LN เป็นการหาค่า natural log ของ X

ฟังก์ชัน LOG เป็นการหาค่า log ฐาน10 ของ X

ตัวอย่าง ถ้าต้องการหา natural log ของคอลัมน์ A

```
SELECT A, LN(A)
FROM NUMBERS;
```

ผลของคำสั่งจะเกิด ERROR ขึ้นเนื่องจากแถวที่ 2 และ 4 ของตาราง NUMBERS มีค่าเป็นลบ ซึ่งถ้าข้อมูลมีค่าเป็นลบจะหาค่าไม่ได้

```
ERROR:
ORA-01428: argument '-45' is out of range
```

จากตัวอย่างถ้าทำการยกกำลัง 2 ข้อมูลในคอลัมน์ A ค่าของข้อมูลที่เป็นลบอยู่เมื่อถูกยกกำลัง 2 จะกลายเป็นบวกจากนั้นจึงทำการหาค่า LN ข้อมูลในคอลัมน์ A

```
SELECT A, LN(ABS(A))
FROM NUMBERS;
```

ผลของคำสั่งในคอลัมน์ LN (ABS(A) จะได้ค่า natural log ที่เกิดจากคอลัมน์ A ยกกำลัง 2

| A | LN (ABS(A)) |
|---|---|
| 3.1415 | 1.1447004 |

| | |
|---|---|
| -45 | 3.8066625 |
| 5 | 1.6094379 |
| -57.667 | 4.0546851 |
| 15 | 2.7080502 |
| .7.2 | 1.974081 |

จะหาค่า Log ฐาน 10 ในคอลัมน์ B จากตาราง NUBMERS

```
SELECT B, LOG(B, 10)
FROM NUMBERS;
```

ผลของคำสั่งในคอลัมน์ LOG(B,10) จะให้ค่า log ฐาน 10 ของคอลัมน์ B

| B | LOG(B,10) |
|---|---|
| 4 | 1.660964 |
| .707 | -6.640962 |
| 9 | 1.0479506 |
| 42 | .61604832 |
| 55 | .57459287 |
| 5.3 | 1.3806894 |

3.6 ฟังก์ชัน MOD(X,Y) เป็นฟังก์ชันที่แสดงเศษที่เกิดข้อมูล X หารด้วย Y

ตัวอย่าง ถ้าต้องการหาเศษของ A หารด้วย B โดยแสดงคอลัมน์ A,B และคอลัมน์เศษที่เหลือ

```
SELECT A, B, MOD(A,B)
FROM NUMBERS;
```

ผลของคำสั่งจะได้คอลัมน์ MOD(A,B) เป็นคอลัมน์ที่แสดงเศษที่เกิดจากข้อมูลในคอลัมน์ A หารด้วย B

| A | B | MOD(A,B) |
|---|---|---|
| 3.1415 | 4 | 3.1415 |
| -45 | .707 | -.459 |
| 5 | 9 | 5 |
| -57.667 | 42 | -15.667 |
| 15 | 55 | 15 |
| -7.2 | 5.3 | -1.9 |

3.7 ฟังก์ชัน POWER (X,Y) เป็นฟังก์ชันในการยกกำลัง โดย X เป็นเลขฐานและ Y จะเป็นเลขยกกำลัง

```
SELECT A, B, POWER(A,B)
    FROM NUMBERS;
```

ผลของคำสั่งจะเกิด ERROR เพราะargument ในแถวที่ 2 ตัวที่เป็นเลขยกกำลังต้องมีค่าเป็นจำนวนเต็ม

```
ERROR:
ORA-01428: ARGUMENT '-45' is out of range
```

จะทำให้เลขยกกำลังมีค่าเป็นเลขจำนวนเต็มบวกโดยใช้ฟังก์ชัน CEILก่อนแล้วจึงจะนำมายกกำลัง

```
SELECT A, CEIL(B), POWER(A,CEIL(B))
FROM NUMBERS;
```

ผลของคำสั่งในคอลัมน์ CELI (B) เป็นค่าของข้อมูลในคอลัมน์ B ที่มีค่าเป็นจำนวนเต็มและเมื่อยกกำลังแล้วจะมีค่าปรากฏในคอลัมน์ POWER (A,CEIL(B)

| A | CEIL(B) | POWER(A,CEIL(B)) |
|---|---|---|
| 3.1415 | 4 | 97.3976 |
| -45 | 1 | -45 |
| 5 | 9 | 1953125 |
| -57.667 | 42 | 9.098E+73 |
| 15 | 55 | 4.842E+64 |
| -7.2 | 6 | 139314.07 |

3.8 ฟังก์ชัน SIGN (X) เป็นฟังก์ชันที่

- ให้ค่าเป็น –1 ถ้า X มีค่าน้อยกว่า 0
- ให้ค่าเป็น 0 ถ้า X มีค่าเท่ากับ 0
- ให้ค่าเป็น 1 ถ้า X มีค่ามากกว่า 0

ตัวอย่าง ถ้าต้องการหาฟังก์ชัน SIGN ในการหาค่าข้อมูลในคอลัมน์ A

```
SELECT A, SIGN(A)
FROM NUMBERS;
```

ผลของคำสั่ง

| A | SIGN (A) |
|---|---|
| 3.1415 | 1 |

| | |
|---|---|
| -45 | -1 |
| 5 | 1 |
| -57.667 | -1 |
| 15 | 1 |
| -7.2 | -1 |
| 0 | 0 |

ถ้าต้องการใช้ฟังก์ชัน SIGN ที่มีค่า 1 ในคอลัมน์ A

```
SELECT A
FROM NUMBERS
WHERE SIGN(A) =1;
```

ผลของคำสั่งจะแสดงข้อมูลในคอลัมน์ A ที่เมื่อใช้ฟังก์ชัน SIGN แล้วมีค่าเป็น 1

| A |
|---|
| 3.1415 |
| 5 |
| 15 |

3.9 ฟังก์ชัน SQRT (X) เป็นฟังก์ชันในการหาค่ารากที่ 2 ของ X

ตัวอย่าง ถ้าต้องการหารากที่ที่ 2 ของข้อมูลในคอลัมน์ A

```
SELECT A, SQRT(A)
FROM NUMBERS;
```

ผลของคำสั่งจะเกิด ERROR เนื่องจากไม่สามารถหาค่ารากที่ 2 ของตัวเลขที่มีค่าเป็นลบได้ดังนี้

ERROR:

ORA-01428: ARGUMENT '-45' is out of range

ตัวอย่าง ถ้านำข้อมูลในคอลัมน์ A มาหาค่าสมบูรณ์แล้วจึงนำไปหาค่ารากที่ 2

```
SELECT ABS(A), SQRT(ABS(A))
FROM NUMBERS;
```

ผลของคำสั่งที่ได้จะได้ค่าสมบูรณ์ของข้อมูลในคอลัมน์ A และได้ค่ารากที่ 2 ของค่าสมบูรณ์ในคอลัมน์ A

| ABS(A) | SQRT(ABS(A)) |
|---|---|
| 3.1415 | 1.7724277 |

| | |
|---|---|
| 45 | -16.7082039 |
| 5 | 12.236068 |
| 57.667 | 7.5938791 |
| 15 | 3.8729833 |
| 7.2 | 2.6832816 |
| 0 | 0 |

4.**ฟังก์ชันตัวอักขระ** (Character functions)

เป็นฟังก์ชันที่ใช้สำหรับจัดการข้อมูลอักขระ โดยที่มีตัวแปรจริงเป็นชนิดอักขระหรือชนิดตัวเลข และให้ผลการคำนวณเป็นค่าอักขระหรือค่าตัวเลข

ตัวอย่างตาราง CHARACTERS

| LASTNAME | FIRSTNAME | M | CODE |
|---|---|---|---|
| PURVIS | KELLY | A | 32 |
| TAYLOR | CHUCK | J | 67 |
| CHRISTINE | LAURA | C | 65 |
| ADAMS | FESTER | M | 87 |
| COSTALES | ARMANDO | A | 77 |
| KONG | MAJOR | G | 52 |

4.1 ฟังก์ชัน CHR เป็นฟังก์ชันสำหรับเปลี่ยนนิพจน์อักขระให้เป็นรหัส ASCII ค่าที่ได้จากฟังก์ชันนี้จะเป็นค่ารหัส ASCII

ตัวอย่าง ถ้าต้องการเปลี่ยนค่าตัวเลขในคอลัมน์ CODE ให้เป็นตัวอักษร

```
SELECT  CODE, CH(CODE)
FROM CHARACTERS;
```

ผลของคำสั่ง

| CODE | CH |
|---|---|
| 32 | |
| 67 | C |
| 65 | A |
| 87 | W |
| 77 | M |

| 52 | 4 |
|---|---|

4.2 ฟังก์ชัน CONCAT (X,Y) เป็นฟังก์ชันในการรวมอักขระ (X และ Y) เข้าด้วยกัน

ตัวอย่าง ถ้าต้องการรวมคอลัมน์ FIRSINAME กับ LASTNAME ไว้ด้วยกัน

```
SELECT  CONCAT(FIRSTNAME, LASTNAME) “FIRST AND LAST NAMES”
FROM CHARACTERS;
```

ผลของคำสั่งจะได้นำคอลัมน์ FIRSTNAME และ LASTNAME มารวมกันแสดงให้เห็นในคอลัมน์ FIRST AND LASTNAMES

| FIRST AND | LAST NAMES |
|---|---|
| KELLY | PURVIS |
| CHUCK | TAYLOR |
| LAURA | CHRISTINE |
| FESTER | ADAMS |
| ARMANDO | COSTALES |
| MAJOR | KONG |

4.3 ฟังก์ชัน INITCAP (<string>) เป็นฟังก์ชันที่เปลี่ยนค่าตัวอักขระ (string) ให้ตัวแรกเป็นอักขระตัวใหญ่แล้วตามด้วยอักขระตัวเล็ก

ถ้าต้องการเปลี่ยนให้เป็นอักษรตัวใหญ่ในคอลัมน์ FIRSTNAME

```
SELECT  FIRSTNAME BEFORE, INITCAP(FIRSTNAME) AFTER
FROM CHARACTERS;
```

**ผลของคำสั่งจะทำการเปลี่ยน**

| BEFORE | AFTER |
|---|---|
| KELLY | Kelly |
| CHUCK | Chuck |
| LAURA | Laura |
| FESTER | Fester |
| ARMANDO | Armando |
| MAJOR | Major |

4.4 ฟังก์ชัน LOWER (<string>) and UPPER (<string>)

ฟังก์ชัน LOWER (<string>) เป็นฟังก์ชัน ที่เปลี่ยนตัวอักขระ (<string>) เป็นอักขระตัวเล็ก

ฟังก์ชัน UPPER (<string>) เป็นฟังก์ชัน ที่เปลี่ยนตัวอักขระ (<string>) เป็นอักขระตัวใหญ่

ตัวอย่าง ถ้าต้องการเปลี่ยนแปลงตัวอักขระในคอลัมน์ FIRSTNAME จากอักขระตัวเล็กให้เป็นอักขระตัวใหญ่ทุกแถว ถ้าใช้คำสั่ง UPDATE ดังนี้

```
UPDATE CHARACTERS
SET FIRSTNAME = 'kelly'
WHERE FIRSTNAME = 'KELLY';
```

ผลของคำสั่งจะทำการเปลี่ยนแปลงข้อมูลได้คำสั่งละ 1 แถวเท่านั้น

```
1 rows update.
```

จากตัวอย่างถ้าใช้ฟังก์ชัน LOWER หรือ UPPER ในการเปลี่ยนแปลงตัวอักขระจะใช้คำสั่งเพียงครั้งเดียวก็สามารถเปลี่ยนแปลงข้อมูลได้ทุกแถวดังนี้

```
SELECT  FIRSTNAME, UPPER(FIRSTNAME), LOWER(FIRSTNAME)
FROM CHARACTERS;
```

ผลของคำสั่งจะทำการเปลี่ยนแปลงข้อมูลในคอลัมน์ FIRSTNAME ให้เป็นอักษรตัวใหญ่และเล็กตามลำดับดังนี้

| FIRSTNAME | UPPER(FIRSTNAME) | LOWER(FIRSTNAME} |
|---|---|---|
| Kelly | KELLY | kelly |
| CHUCK | CHUCK | chuck |
| LAURA | LAURA | laura |
| FESTER | FESTER | fester |
| ARMANDO | ARMANDO | armando |
| MAJOR | MAJOR | major |

4.7 ฟังก์ชัน REPLACE (<string>,X,Y) เป็นฟังก์ชันในการแทนค่าอักขระ X โดยการค้นหาตัวอักขระที่ต้องการแทนที แล้วแทนที่ด้วยอักขระ Y ที่ต้องการ

ตัวอย่าง ถ้าต้องการค้นหาอักขระ ST โดยไม่แทนที่ด้วยอักขระใดๆ

คำสั่งต้องการหาตัวอักษร ST ในคอลัมน์ LASTNAME โดยไม่ต้องแทนที่ด้วยตัวอักษรใด

```
SELECT  LASTNAME, REPLACE(LASTNAME, 'ST') REPLACEMENT
```

FROM CHARACTERS;

ผลของคำสั่งจะทำให้ในแถวที่ 3 และแถวที่ 6 ที่มีคอลัมน์ LASTNAME ที่มีอักษร ST อยู่จะถูกตัดทิ้งไป

| LASTNAME | REPLACEMENT |
|---|---|
| PURVIS | PURVIS |
| TAYLOR | TAYLOR |
| CHRISTINE | CHRIINE |
| ADAMS | ADAMS |
| COSTALES | COALES |
| KONG | KONG |

ตัวอย่าง ถ้าต้องการหาตัวอักษร ST ในคอลัมน์ LASTNAME แล้วแทนที่ด้วย **

SELECT LASTNAME, REPLACE(LASTNAME, 'ST', '**') REPLACEMENT

FROM CHARACTERS;

ผลของคำสั่งจะทำให้ในแถวที่ 3 และแถวที่ 6 ในคอลัมน์ LASTNAME ที่มีอักษร ST อยู่จะถูกแทนที่ด้วย **

| LASTNAME | REPLACEMENT |
|---|---|
| PURVIS | PURVIS |
| TAYLOR | TAYLOR |
| CHRISTINE | CHRI**INE |
| ADAMS | ADAMS |
| COSTALES | CO**ALES |
| KONG | KONG |

4.8 ฟังก์ชัน SUBSTR (<string>,x,y) เป็นฟังก์ชันที่นำตัวอักษร (<string>) ในตำแหน่งที่ x

ตัวอย่าง ถ้าต้องกแสดงอักษรตั้งแต่ตำแหน่งที่ 2 มาแสดง 3 ตำแหน่ง ของคอลัมน์ FIRSTNAME

SELECT FIRSTNAME, SUBSTR(FIRSTNAME,2,3)

FROM CHARACTERS;

ผลของคำสั่งจะเห็นว่าในคอลัมน์ FIRSTNAME จะแสดงอักษรออกมา 3 ตัว แม้แต่ชื่อคอลัมน์ก็จะแสดงเพียง 3 ตัวเช่นเดียวกัน

| FIRSTNAME | SUB |
|---|---|

| | |
|---|---|
| Kelly | Ell |
| CHUCK | HUC |
| LAURA | AUR |
| FESTER | EST |
| ARMANDO | RMA |
| MAJOR | AJO |

ตัวอย่าง ถ้าต้องการให้แสดงตั้งแต่ตัวอักษรในตำแหน่งที่ 3 โดยไม่จำกัดว่าให้แสดงก็ตัวอักษร

```
SELECT  FIRSTNAME, SUBSTR(FIRSTNAME,3)
FROM CHARACTERS;
```

ผลของคำสั่งจะแสดงอักษรในคอลัมน์ FIRSTNAME ตั้งแต่ตัวที่ 3 ทั้งหมด

| FIRSTNAME | SUBSTR(FIRSTN |
|---|---|
| Kelly | Lly |
| CHUCK | UCK |
| LAURA | URA |
| FESTER | STER |
| ARMANDO | MANDO |
| MAJOR | JOR |

5.**ฟังก์ชันการแปลง** (Conversion functions)

5.1ฟังก์ชัน TO_CHAR จะทำการแลง data type ที่เป็นตัวเลขให้เป็นตัวอักษร

```
SELECT  TESTNUM, TO_CHAR(TESTNUM)
FROM CONVERSIONS;
```

ผลของคำสั่ง

| TESTNUM | TO_CHAR(TESTNUM) |
|---|---|
| 95 | 95 |
| 23 | 23 |
| 68 | 68 |

```
SELECT  TESTNUM, LENGTH(TO_CHAR(TESTNUM))
FROM CONVERSIONS;
```

ผลของคำสั่งจะทำการนับความยาวของตัวเลขที่แปลงเป็นตัวอักษรแล้ว

| TESTNUM | LEGTH(TO_CHAR(TESTNUM)) |
|---|---|
| 95 | 2 |
| 23 | 2 |
| 68 | 2 |

**เรื่องที่ 6.3.3 การเรียกดูข้อมูลในรูปแบบต่างๆ**

1. การเรียกดูข้อมูลโดยใช้ฟังก์ชันในการรวม

การเรียกดูข้อมูลโดยใช้ฟังก์ชันในการรวมมีรูปแบบดังนี้

SELECT <column 1, column 2,...>

FROM <table name>

[WHERE<condition>]

[GROUP BY < grouping column>...]

[HAVING<condition>];

SELECT คำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูล

column 1, column 2,... คอลัมน์ที่ต้องการเรียกค้น

FROM การกำหนดว่าให้เรียกดูข้อมูล ได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

WHERE<condition> ส่วนของคำสั่งที่บอกเงื่อนไขที่จะใช้ในการค้นหาข้อมูล

GROUP BY < grouping column>... ส่วนของคำสั่งที่บอกเงื่อนไขการจัดกลุ่ม

HAVING<condition> ใช้ควบคู่กันกับ GROUP BY เสมอ เพื่อต้องการให้ได้ข้อมูลที่จัดกลุ่มตาม GROUP BY

การเรียกดูข้อมูลโดยใช้อนุประโยค GROUP BY อนุประโยค GROUP BY เป็นคำสั่งในการกำหนดค่าต่าง ๆ ในคอลัมน์ใดคอลัมน์หนึ่งโดยเฉพาะในรูปของอีกคอลัมน์หนึ่ง ซึ่งใช้จัดเรียงข้อมูลที่มีความสัมพันธ์กันตามเงื่อนไข ข้อมูลที่ต้องการจัดกลุ่มนั้นและ ต้องสามารถรวมกลุ่มกันได้ด้วย คำสั่ง GROUP BY เป็นคำสั่งที่ใช้ในการจัดแถวข้อมูลตามคอลัมน์ที่ระบุหลัง GROUP BY โดยข้อมูลที่เหมือนกันจะจัดให้อยู่ในหมู่เดียวกัน

ตัวอย่างตาราง CHECKS

| CHECK# | PAYEE | AMOUNT | REMARKS |
|---|---|---|---|
| 1 | Malee Benjanee | 150 | Have sons next time |

| | | | |
|---|---|---|---|
| 2 | Reading R.R. | 24534 | Train to Chiangmai |
| 3 | Malee Benjanee | 20032 | Cellular Phone |
| 4 | Surasit Utilities | 98 | Gas |
| 5 | Jintana $ Mitree | 150 | Groceries |
| 16 | Cash | 25 | Wild Night Out |
| 17 | Benjawan Gas | 251 | Gas |
| 9 | Arun Cleaners | 2435 | X-Tra Starch |
| 20 | Arun Cleaners | 105 | All Dry clean |
| 8 | Cash | 60 | Trip to Saraburi |
| 21 | Cash | 34 | Trip to Nonthaburi |
| 30 | Surasit Utilities | 875 | Water |
| 31 | Surasit Utilities | 34 | Sewer |
| 25 | Benjawan Gas | 1575 | Gas |

ตัวอย่าง ถ้าต้องการหาผลรวมในคอลัมน์ AMOUNT ของตาราง CHECKS จะใช้คำสั่งดังนี้

```
SELECT PAYEE, SUM(AMOUNT)
FROM CHECKS;
```

ถ้าไม่ใช่ GROUP BY ในการรวมคอลัมน์ AMOUNT ของแต่ละคน จะเกิด ERROR ดังนี้

ผลลัพธ์

```
Dynamic SQL Error
-SQL error code = -104
-invalid column reference
```

จากตัวอย่างข้างต้น ถ้าต้องการนับว่า PAYEE แต่ละคนมีจำนวนค่าใช้จ่ายกี่ครั้งจะใช้คำสั่งดังนี้

```
SELECT PAYEE, SUM(AMOUNT), COUNT(PAYEE)
FROM CHECKS
GROUP BY PAYEE;
```

ผลลัพธ์

| PAYEE | SUM | COUNT |
|---|---|---|
| Arun Cleaners | 2540 | 2 |

| | | |
|---|---|---|
| Cash | 119 | 3 |
| Benjawan Gas | 1826 | 2 |
| Jintana $ Mitree | 150 | 1 |
| Surasit Utilities | 1007 | 3 |
| Malee Benjanee | 10182 | 2 |
| Reading R.R. | 24534 | 1 |

ตัวอย่างตาราง ORGCHART

| NAME | TEAM | SALARY | SICKLEAVE | ANNUALLEAVE |
|---|---|---|---|---|
| ASAMS | RESEARCH | 34000.00 | 34 | 12 |
| WILKES | MARKETING | 31000.00 | 40 | 9 |
| STOKES | MARKETING | 36000.00 | 20 | 19 |
| MEZA | COLLECTIONS | 40000.00 | 30 | 27 |
| SIRIWAN | RESEARCH | 45000.00 | 20 | 17 |
| RICHARDSON | MARKETING | 42000.00 | 25 | 18 |
| FURY | COLLECTIONS | 35000.00 | 22 | 14 |
| PRECOURT | PR | 37500.00 | 24 | 24 |

ถ้าต้องการหาค่าเฉลี่ยของ SALARY ของ TEAM งานแต่ละTEAM จะใช้คำสั่งดังนี้

```
SELECT TEAM, AVG(SALARY)
FROM ORGCHART
GROUP BY TEAM;
```

ผลลัพธ์

| TEAM | AVG |
|---|---|
| COLLECTIONS | 37500.00 |
| MARKETING | 36333.33 |
| PR | 37500.00 |

| | |
|---|---|
| RESEARCH | 39500.00 |

การเรียกดูข้อมูลแบบ HAVING HAVING จะใช้ควบคู่กันกับ GROUP BY เสมอ เพื่อต้องการให้ได้ข้อมูลที่จัดกลุ่มตาม GROUP BYเพียงบางส่วนตามเงื่อนไขที่ระบุไว้ใน HAVING

จากตัวอย่างที่ผ่านมาถ้าต้องการดูเงินเดือนเฉลี่ยของทีมงานที่มีค่าน้อยกว่า 38000 จะใช้คำสั่งดังนี้

```
SELECT TEAM, AVG(SALARY)
FROM ORGCHART
GROUP BY TEAM
HAVING AVG(SALARY) < 38000;
```

ผลลัพธ์

| TEAM | AVG |
|---|---|
| PR | 37500.00 |

ตัวอย่าง การใช้ GROUP BY และ HAVING ร่วมกับคำสั่ง AND คือดูว่าทีมงานใดมีค่าเฉลี่ยของ SICKLEAVE มากกว่า 25 และ มีค่าเฉลี่ยของ ANNUALLEAVE น้อยกว่า 20 จะใช้คำสั่งดังนี้

```
SELECT TEAM, AVG(SICKLEAVE), AVG(ANNUALLEAVE)
FROM ORGCHART
GROUP BY TEAM
HAVING AVG(SICKLEAVE) >25 AND
AVG(ANNUALLEAVE) <20;
```

ผลลัพธ์

| TEAM | AVG | AVG |
|---|---|---|
| MARKETING | 28 | 15 |
| RESEARCH | 27 | 15 |

ตัวอย่าง ถ้าต้องการหาค่าเฉลี่ยของ SICKLEAVE และค่าเฉลี่ยของ ANNUALLEAVE ของจำนวนทีมที่มีจำนวน row มากกว่า 1 จะใช้คำสั่งดังนี้

```
SELECT TEAM, AVG(SICKLEAVE), AVG(ANNUALLEAVE)
FROM ORGCHART
GROUP BY TEAM
HAVING COUNT(TEAM) > 1;
```

ผลลัพธ์

| TEAM | AVG | AVG |
|---|---|---|
| COLLECTIONS | 26 | 21 |
| MARKETING | 28 | 15 |
| RESEARCH | 27 | 15 |

**2.การเรียกค้นข้อมูลจากหลายตาราง**

การจัดทำฐานข้อมูลในรูปตารางเกิดจากการที่ข้อมูลได้ออกแบบมาเพื่อลดความซ้ำซ้อน(normalization) ดังนั้นข้อมูลที่มีรายละเอียดของข้อมูลมากอาจจะถูกเก็บไว้ในหลายๆตารางแยกออกมาต่างหาก เช่น ตารางข้อมูลที่เป็นตารางหลัก(master table) และ ตารางข้อมูลที่เป็นตารางเชิงรายการ (transaction table) และตารางข้อมูลที่เป็นตารางที่อยู่(address table) เป็นต้น การแยกออกเป็นตารางข้อมูลย่อยๆนี้นอกจากลดความซ้ำซ้อน แล้วยังช่วยในการประหยัดเนื้อที่ และยังเพิ่มประสิทธิภาพของฐานข้อมูล

การเรียกค้นข้อมูลจากหลายตารางของภาษา SQL เป็นการกำหนดความสัมพันธ์ระหว่างตารางทั้งหลาย โดยสามารถเอาข้อมูลในตารางกี่ตารางก็ได้ให้มาสัมพันธ์กัน ดังนั้นจึงสามารถเชื่อมต่อข้อมูลที่แตกต่างกันได้โดยการใช้คำสั่ง WHERE คำสั่ง WHERE เป็นคำสั่งในการกำหนดเงื่อนไขในการเรียกดูข้อมูลใช้คู่กับคำสั่ง SELECT และ FROM

ตัวอย่างTABLE1

| ROW | REMARK |
|---|---|
| Row 1 | Table 1 |
| Row 2 | Table 1 |
| Row 3 | Table 1 |
| Row 4 | Table 1 |
| Row 5 | Table 1 |
| Row 6 | Table 1 |

ตัวอย่าง TABLE 2

| ROW | REMARK |
|---|---|
| Row 1 | Table 2 |
| Row 2 | Table 2 |
| Row 3 | Table 2 |
| Row 4 | Table 2 |
| Row 5 | Table 2 |

| | |
|---|---|
| Row 6 | Table 2 |

ถ้าต้องการรวมทั้ง 2 ตารางเข้าด้วยกัน

```
SELECT * FROM TABLE1, TABLES2
```

ผลลัพธ์

| ROW | REMARK | ROW | REMARK |
|---|---|---|---|
| Row 1 | Table 1 | Row 1 | Table 2 |
| Row 1 | Table 1 | Row 2 | Table 2 |
| Row 1 | Table 1 | Row 3 | Table 2 |
| Row 1 | Table 1 | Row 4 | Table 2 |
| Row 1 | Table 1 | Row 5 | Table 2 |
| Row 1 | Table 1 | Row 6 | Table 2 |
| Row 2 | Table 1 | Row 1 | Table 2 |
| Row 2 | Table 1 | Row 2 | Table 2 |
| Row 2 | Table 1 | Row 3 | Table 2 |
| Row 2 | Table 1 | Row 4 | Table 2 |
| Row 2 | Table 1 | Row 5 | Table 2 |
| Row 2 | Table 1 | Row 6 | Table 2 |
| Row 3 | Table 1 | Row 1 | Table 2 |
| Row 3 | Table 1 | Row 2 | Table 2 |
| Row 3 | Table 1 | Row 3 | Table 2 |
| Row 3 | Table 1 | Row 4 | Table 2 |
| Row 3 | Table 1 | Row 5 | Table 2 |
| Row 3 | Table 1 | Row 6 | Table 2 |
| Row 4 | Table 1 | Row 1 | Table 2 |
| Row 4 | Table 1 | Row 2 | Table 2 |
| Row 4 | Table 1 | Row 3 | Table 2 |
| Row 4 | Table 1 | Row 4 | Table 2 |
| Row 4 | Table 1 | Row 5 | Table 2 |
| Row 4 | Table 1 | Row 6 | Table 2 |

| | | | |
|---|---|---|---|
| Row 5 | Table 1 | Row 1 | Table 2 |
| Row 5 | Table 1 | Row 2 | Table 2 |
| Row 5 | Table 1 | Row 3 | Table 2 |
| Row 5 | Table 1 | Row 4 | Table 2 |
| Row 5 | Table 1 | Row 5 | Table 2 |
| Row 5 | Table 1 | Row 6 | Table 2 |
| Row 6 | Table 1 | Row 1 | Table 2 |
| Row 6 | Table 1 | Row 2 | Table 2 |
| Row 6 | Table 1 | Row 3 | Table 2 |
| Row 6 | Table 1 | Row 4 | Table 2 |
| Row 6 | Table 1 | Row 5 | Table 2 |
| Row 6 | Table 1 | Row 6 | Table 2 |

**3.การเรียกดูข้อมูลแบบซ้อนกัน**

การเรียกดูข้อมูลแบบซ้อนกัน(subqueies) เป็นการสร้างคำสั่ง SELECT ซ้อนกัน การเรียกดูข้อมูลแบบซ้อนกันมีจุดประสงค์เพื่อลดภาระในการเชื่อมตารางที่ต้องใช้ในหน่วยความจำเป็นจำนวนมาก คำสั่งย่อยนี้สามารถสร้างหลังคำสั่ง WHERE มีรูปแบบดังนี้

SELECT [*] <column 1, column 2,...>

FROM <table name>

[WHERE<column list = <Select Statement>]

SELECT คำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูล

column 1, column 2,... คอลัมน์ที่ต้องการเรียกค้น

FROM การกำหนดว่าให้เรียกดูข้อมูล ได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

WHERE<condition> ส่วนของคำสั่งที่บอกเงื่อนไขที่จะใช้ในการค้นหาข้อมูล

Select Statement ส่วนของคำสั่งที่เรียกค้นข้อมูลตามเงื่อนไข

การทำงานของคำสั่งย่อยที่ใช้ในการระบุเงื่อนไขหรือเรียกข้อมูลจะทำจากคำถามย่อยด้านในสุดผลที่ได้จะเป็นค่ากลับมาให้กับค่าที่อยู่หน้าเครื่องหมายเท่ากับ เพื่อเรียกค้นข้อมูล( SELECT )ตามต้องการ

ตัวอย่าง ถ้าต้องการหาคำสั่งซื้อของพนักงานขายที่อาศัยอยู่ใน Bangkok

SELECT *

```
FROM ORDERSTAB
WHERE  SALENO IN
(SELECT SALENO
 FROM SALETAB
 WHERE ADDRESS  = 'Bangkok');
```

ผลของคำสั่งจะได้ตารางดังนี้

ตารางคำสั่งซื้อสินค้า

| ORDERNO | AMT | ORDERDATE | CUSNO | SALENO |
|---|---|---|---|---|
| 3001 | 1869 | 6/03/2000 | 2008 | 1007 |
| 3003 | 76719 | 6/03/2000 | 2001 | 1001 |
| 3002 | 190010 | 6/03/2000 | 2007 | 1004 |
| 3005 | 516045 | 6/03/2000 | 2003 | 1002 |
| 3006 | 109816 | 6/03/2000 | 2008 | 1007 |
| 3009 | 171323 | 6/04/2000 | 2002 | 1003 |
| 3007 | 7573 | 6/04/2000 | 2004 | 1002 |
| 3008 | 472300 | 6/05/2000 | 2006 | 1001 |
| 3010 | 130995 | 6/06/2000 | 2004 | 1002 |
| 3011 | 989198 | 6/06/2000 | 2006 | 1001 |

ตารางพนักงานขาย

| SALENO | SALENAME | ADDRESS | SALECOM |
|---|---|---|---|
| 1001 | Chaiwat | Bangkok | 0.12 |
| 1002 | Mitree | Puket | 0.13 |
| 1004 | Benjawan | Bangkok | 0.11 |
| 1007 | Kanjana | Chiangmai | 0.15 |
| 1003 | Ternjai | Nonthaburi | 0.10 |

| ORDERNO | AMT | ORDERDATE | CUSNO | SALENO |
|---|---|---|---|---|
| 3003 | 76719 | 6/03/2000 | 2001 | 1001 |
| 3002 | 190010 | 6/03/2000 | 2007 | 1004 |
| 3008 | 472300 | 6/05/2000 | 2006 | 1001 |
| 3011 | 989198 | 6/06/2000 | 2006 | 1001 |

จากตัวอย่างนี้สามารถใช้คำสั่งที่ง่ายกว่าและได้ผลลัพธ์เหมือนกันคือ

```
SELECT ORDERNO AMT ORDERDATE CUSNO ORDERSTAB. SALENO
```

```
FROM ORDERSTAB, SALETAB
WHERE  ORDERSTAB.SALENO SALETAB.SALENO
AND  SALETAB.ADDRESS = 'Bangkok';
```

ตัวอย่าง ตาราง PART

| PARTNUM | DESCRIPTION | PRICE |
|---|---|---|
| 54 | PEDALS | 54.25 |
| 42 | SEATS | 24.50 |
| 46 | TIRES | 15.25 |
| 23 | MOUNTAIN BIKE | 350.45 |
| 76 | ROAD BIKE | 530.00 |
| 10 | TANDEM | 1200.00 |

ตัวอย่างตาราง ORDERS

| ORDEREDON | NAME | PARTNUM | QUANTITY | REMARKS |
|---|---|---|---|---|
| 15-MAY-2001 | TRUE WHEEL | 23 | 6 | PAID |
| 19-MAY-2001 | TRUE WHEEL | 76 | 3 | PAID |
| 2-SEP-2001 | TRUE WHEEL | 10 | 1 | PAID |
| 30-JUN-2001 | BIKE SPEC | 54 | 10 | PAID |
| 30-MAY-2001 | BIKE SPEC | 10 | 2 | PAID |
| 30-MAY-2001 | BIKE SPEC | 23 | 8 | PAID |
| 17-JAN-2001 | BIKE SPEC | 76 | 11 | PAID |
| 17-JAN-2001 | LE SHOPPE | 76 | 5 | PAID |
| 1-JUN-2001 | LE SHOPPE | 10 | 3 | PAID |
| 1-JUN-2001 | AAA BIKE | 10 | 1 | PAID |
| 1-JUN-2001 | AAA BIKE | 76 | 4 | PAID |
| 1-JUN-2001 | AAA BIKE | 46 | 14 | PAID |
| 11-JUN-2001 | JACKS BIKE | 76 | 14 | PAID |

```
SELECT *
FROM ORDERS
WHERE PARTNUM =
```

```
(SELECT PARTNUM
FROM PART
WHERE DESCRIPTION LIKE "ROAD%")
```

ผลลัพธ์

| ORDEREDON | NAME | PARTNUM | QUANTITY | REMARKS |
|---|---|---|---|---|
| 19-MAY-2001 | TRUE WHEEL | 76 | 3 | PAID |
| 17-JAN-2001 | BIKE SPEC | 76 | 11 | PAID |
| 17-JAN-2001 | LE SHOPPE | 76 | 5 | PAID |
| 1-JUN-2001 | AAA BIKE | 76 | 4 | PAID |
| 11-JUN-2001 | JACKS BIKE | 76 | 14 | PAID |