ทฤษฎี บทที่ 7 ระบบฐานข้อมูล MySQL

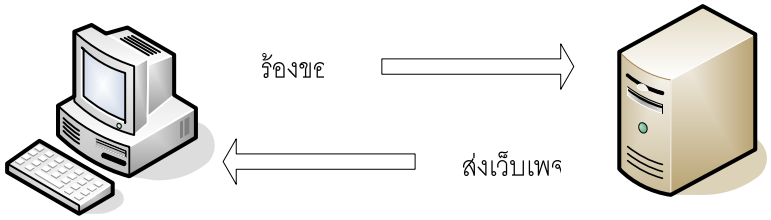
การสร้างเว็บเพจที่เรียบง่ายและไม่มีการเปลี่ยนแปลงบ่อย ๆ นั้น การใช้ภาษา HTML เพียงอย่างเดียวก็อาจจะเพียงพอแล้ว แต่ในปัจจุบันความต้องการรับรู้ข่าวสารต่าง ๆ ให้รวดเร็วและทันต่อเหตุการณ์ จำเป็นต้องนำเทคโนโลยีอื่น ๆ เข้ามาช่วยให้สามารถปรับปรุงข้อมูลได้ทันท่วงที และลดความผิดพลาดที่อาจจะเกิดขึ้นจากการที่ต้องปฏิบัติงานกับข้อมูลจำนวนมหาศาล และสิ่งหนึ่งที่ถูกนำมาใช้เพื่อลดปัญหาความซ้ำซากของข้อมูล คือ ระบบฐานข้อมูลบนเว็บ (Web Database หรือ Webbase)

และเมื่อมีความจำเป็นต้องนำระบบการจัดการฐานข้อมูลมาใช้ในเว็บไซท์ที่เขียนขึ้นด้วย PHP ระบบจัดการฐานข้อมูลที่มักจะถูกเลือกใช้เป็นลำดับต้น ๆ คือ MySQL ด้วยคุณสมบัติเด่นหลายประการ ทำให้ผู้ใช้ PHP ส่วนใหญ่เลือกใช้ MySQL เป็นระบบฐานข้อมูลสำหรับเว็บไซท์และ Web application

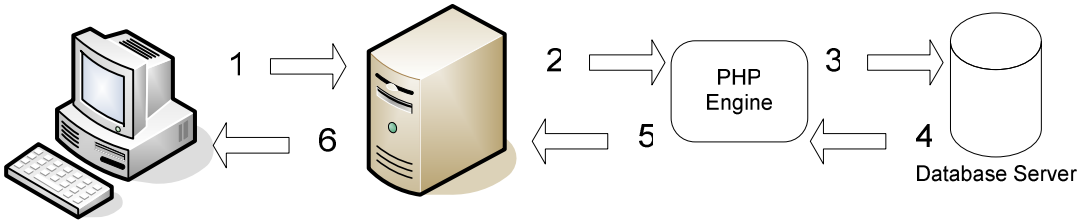
MySQL เป็นระบบฐานข้อมูลที่ถูกพัฒนาขึ้นโดยบริษัท MySQL AB ประเทศสวีเดน ซึ่งมีวัตถุประสงค์ให้ MySQL เป็นซอฟต์แวร์ที่เปิดเผยซอร์สโค้ด (Open Source)

7.1 สถาปัตยกรรมเว็บดาต้าเบส หรือ เว็บเบส

การทำงานของเว็บเซิร์ฟเวอร์โดยพื้นฐานจะเป็นการนำเว็บเพจที่ถูกเก็บอยู่ที่เซิร์ฟเวอร์ส่งผ่านไปให้แก่เว็บบราวเซอร์ในเครื่องที่ทำการร้องขอมา โดยข้อมูลส่วนใหญ่ในเว็บเพจจะอยู่ในรูปของแท็กในภาษา HTML และสคริปต์ที่ทำงานทางฝั่งไคลเอ็นต์ เช่น JavaScript



แต่สำหรับการนำระบบฐานข้อมูลเข้ามาใช้ร่วมกับเว็บเพจนั้น เมื่อเว็บบราวเซอร์ร้องขอมาที่เว็บเซิร์ฟเวอร์แล้ว PHP จะทำหน้าที่เป็นตัวกลางในการดึงข้อมูลจากฐานข้อมูล มาจัดทำเป็นสคริปต์ HTML ในรูปแบบที่เว็บบราวเซอร์สามารถเข้าใจได้ โดยมีขั้นตอนดังรูป



จากรูป อธิบายกระบวนการของเว็บแอพพลิเคชั่นที่มีการติดต่อกับฐานข้อมูลได้ ดังนี้

1. เว็บบราวเซอร์ทำการร้องขอเว็บเพจด้วยโปรโตคอล HTTP ไปยังเว็บเซิร์ฟเวอร์
2. เมื่อเว็บเซิร์ฟเวอร์ได้รับการร้องขอ จะทำการเรียกไฟล์ที่ถูกร้องขอแล้วส่งต่อให้กับ PHP Engine เพื่อทำการประมวลผล
3. ในกรณีที่สคริปต์มีคำสั่งให้ทำการติดต่อฐานข้อมูลและมีการทำคิวรี (query) เพื่ออ่านหรือประมวลผลฐานข้อมูล PHP Engine ก็จะทำการติดต่อและส่งคิวรีไปยังดาต้าเบสเซิร์ฟเวอร์ ซึ่งในบทเรียนนี้ก็คือ MySQL Server

4. ดาต้าเบสเซิร์ฟเวอร์จะส่งผลลัพธ์ของคิวรีกลับไปให้ PHP Engine
5. หลังจาก PHP Engine นำข้อมูลที่ได้รับจากดาต้าเบสเซิร์ฟเวอร์มาประมวลผลแล้ว จะทำการสร้างผลลัพธ์ในรูปแบบของ HTML แล้วส่งให้แก่เว็บเซิร์ฟเวอร์
6. เว็บเซิร์ฟเวอร์จะส่งผลลัพธ์ในรูป HTML กลับไปยังเว็บบราวเซอร์เพื่อแสดงผล

7.2 ภาษา SQL เบื้องต้น (Structure Query Laguage)

ภาษา SQL สามารถแบ่งออกได้เป็น 3 ประเภทใหญ่ ๆ คือ

- **ภาษาสำหรับการนิยามข้อมูล (Data Definition Language : DDL)**
  - คำสั่งที่ใช้กำหนดโครงสร้างข้อมูลของตาราง เช่น คำสั่ง Create Table
  - คำสั่งที่ใช้สำหรับการเพิ่มคอลัมน์ หรือปรับเปลี่ยนลักษณะของคอลัมน์ เช่น คำสั่ง Alter
  - คำสั่งสำหรับการกำหนด View หรือตารางเสมือน เช่น คำสั่ง Create View
- **ภาษาสำหรับการจัดการข้อมูล (Data Manipulation Language : DML)**
  - การเรียกใช้ข้อมูล หรือ ดูข้อมูลในตาราง เช่น คำสั่ง Select
  - การเพิ่มข้อมูลในตาราง เช่น คำสั่ง Insert
  - การลบข้อมูลในตาราง เช่น คำสั่ง Delete
  - การเปลี่ยนแปลงข้อมูลในตาราง เช่น คำสั่ง Update
- **ภาษาควบคุม (Data Control Language)**
  - ควบคุมการเกิดภาวะพร้อมกัน (Multi tasking) เช่น การจัดการ Transaction
  - ควบคุมเหตุการณ์ที่ผู้ใช้หลายคนเรียกข้อมูลเดียวกัน พร้อมกัน เช่น การ lock record
  - ควบคุมความปลอดภัยของข้อมูล เช่น การทำ data integrity
  - กำหนดสิทธิให้ผู้ใช้มีการเข้าถึงข้อมูล ต่างระดับกัน เช่น คำสั่ง Grant

7.2.1 ชนิดของข้อมูลที่ใช้ในภาษา SQL

ชนิดของข้อมูลอาจแบ่งได้เป็น 3 กลุ่มหลัก ๆ คือ ตัวเลข, วันที่และเวลา, สตริง สำหรับค่าตัวเลขสามารถกำหนดความยาวของตัวเลขและจำนวนตัวเลขหลังจุดทศนิยมได้ ขึ้นอยู่กับเป็นข้อมูลชนิดใด โดนในตารางด้านล่างต่อไปนี้จะแทนค่าความยาวของตัวเลขและข้อความ (สตริง) ซึ่งรวมจุดทศนิยมแล้วด้วย M และแทนจำนวนตัวเลขที่อยู่หลังจุดทศนิยมด้วย D

ข้อมูลชนิดตัวเลข (Numeric)

ตารางแสดงชนิดข้อมูลตัวเลข

ชนิดข้อมูล	*ขนาด (ไบต์)*	*ช่วงค่า*
TINYING[(M)]	1	-128 ถึง 127
SMALLINT[(M)]	2	-32768 ถึง 32767
MEDIUMINT[(M)]	3	-8388608 ถึง 8388607
INT[(M)], INTEGER[(M)]	4	-2147483648 ถึง 2147483647
BIGINT[(M)]	8	-9223372036854775808 ถึง 9223372036854775807
FLOAT[(M,D)]	4	-3.402823466E+38 ถึง -1.175494351E-38

DOUBLE[(M,D)], DOUBLE REAL[(M,D)]	8	-1.7976931348623157E+308 ถึง -2.2250738585072014E-3.8
DECIMAL[(M[,D])], DEC[(M[,D])], NUMERIC[(M[,D])]	M+2	ขึ้นอยู่กับความยาวของตัวเลข (M) เนื่องจากมีการจัดเก็บแบบ char

ข้อมูลประเภทวันที่และเวลา (Date and Time)

ข้อมูลประเภทวันที่และเวลาใน MySQL นับว่ามีความยืดหยุ่นสูง โดยจะตรวจสอบเพียงว่าเดือนอยู่ในช่วง 0-12 และวันที่อยู่ในช่วง 0-31 เท่านั้น ที่เหลือเป็นภาระของแอพพลิเคชั่นในการตรวจสอบความถูกต้องของข้อมูลประเภทวันที่ เช่นการบันทึกวันที่ '2003-02-29' (ปี ค.ศ. 2003 ไม่ใช่ปีอธิกสุรทิน เดือนกุมภาพันธ์จึงมีเพียง 28 วัน) เข้าในในฟิลด์ชนิด DATETIME นั้น MySQL จะยอมให้บันทึกเข้าไปได้

สำหรับฟิลด์ข้อมูลชนิด TIMESTAMP จะเป็นฟิลด์ที่ถูกปรับปรุงอัตโนมัติเมื่อมีการ Insert หรือ Update โดยฟิลด์ชนิดนี้จะถูกกำหนดค่าด้วยวันเวลาขณะนั้นตามรูปแบบในเทเบิล ขึ้นอยู่กับการกำหนดค่า M เป็น 14,12,8 หรือ 6

ตารางแสดงชนิดข้อมูลวันที่และเวลา

ชนิดข้อมูล	*รูปแบบ*	*ช่วงข้อมูล*
DATE	YYYY-MM-DD	1000-01-01 ถึง 9999-12-31
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ถึง 9999-12-31 23:59:59
TIMESTAMP[(M)]	YYYYMMDDHHMMSS, YMMDDHHMMSS, YYYYMMDD หรือ YYMMDD ขึ้นอยู่กับว่า M มีค่าเท่ากับ 14,12,8 หรือ 6	1970-01-01 00:00:00 ถึงปี ค.ศ. 2037
TIME	HH:MM:SS	-838:59:59 ถึง 838:59:59
YEAR(2|4)	YYYY	กรณีใช้ความยาว 2 ตำแหน่ง ค่าปี ค.ศ. ที่เก็บได้จะอยู่ระหว่าง 1970 ถึง 2069 กรณีใช้ความยาว 4 ตำแหน่ง ค่าปี ค.ศ.ที่เก็บได้จะอยู่ระหว่าง 1901 ถึง 2155

ข้อมูลชนิด CHAR และ VARCHAR

ข้อมูลสตริงชนิด CHAR และ VARCHAR ใช้เก็บข้อความขนาดสั้น ระหว่าง 1 ถึง 255 ตัวอักษร ขึ้นอยู่กับจำนวนที่กำหนดให้ ตัวอย่าง เช่น CHAR(20) เป็นการกำหนดให้เก็บข้อความขนาดความยาวไม่เกิน 20 ตัวอักษร หากพยายามใส่ค่าที่มีความยาวเกินกว่าที่กำหนดไว้ ข้อความที่เกินจะถูกตัดทิ้ง

ในการจัดเรียง (sorting) และการเปรียบเทียบข้อมูล (comparision) นั้น ทั้ง CHAR และ VARCHAR จะไม่คำนึงถึงว่าเป็นตัวอักษรพิมพ์ใหญ่ หรือพิมพ์เล็ก (case-insensitive) เว้นแต่ว่าจะมีการใช้แอตทริบิวต์ BINARY ต่อท้ายในขณะสร้างเทเบิล ซึ่งจะทำให้การจัดเรียงและการเปรียบเทียบมีการพิจารณาตัวอักษรพิมพ์ใหญ่และพิมพ์เล็กด้วย (case-sensitive)

สิ่งที่ข้อมูลทั้งสองชนิดนี้ต่างกัน ก็คือ

- ในขณะจัดเก็บข้อมูล ข้อมูลชนิด CHAR จะจัดเก็บข้อความพร้อมช่องว่างจนเต็มขนาดที่ระบุ ส่วนข้อมูลชนิด VARCHAR จะจัดเก็บเฉพาะข้อความตามที่เป็นจริงเท่านั้น
- ในขณะเรียกใช้ข้อมูล ข้อมูลชนิด CHAR จะทำการตัดช่องว่างท้ายข้อความออก
- พื้นที่จัดเก็บที่ต้องใช้สำหรับข้อมูลชนิด CHAR เท่ากับขนาดที่ระบุไว้ ส่วนข้อมูลชนิด VARCHAR จะใช้พื้นที่เท่ากับความยาวจริงของข้อความบวกด้วย 1 ไบต์เพื่อใช้สำหรับบันทึกขนาดของข้อความ ตัวอย่างเช่น

ข้อความ	***จัดเก็บแบบ char(3)***		***จัดเก็บแบบ varchar(3)***	
	ข้อความที่ถูกจัดเก็บ	***พื้นที่ที่ใช้ (ไบต์)***	***ข้อความที่ถูกจัดเก็บ***	***พื้นที่ที่ใช้ (ไบต์)***
‘’	‘ ’	3	‘’	1
‘P’	‘P ’	3	‘P’	2
‘PHP’	‘PHP’	3	‘PHP’	4
‘PHP Book’	‘PHP’	3	‘PHP’	4

ข้อมูลชนิด BLOB และ TEXT

ข้อมูลชนิด BLOB (Binary Large Object) และ TEXT ใช้สำหรับเก็บข้อมูลที่มีขนาดใหญ่กว่า CHAR และ VARCHAR สิ่งที่แตกต่างระหว่าง BLOB กับ TEXT คือ ในการจัดเรียงข้อมูล (sorting) และการเปรียบเทียบ (comparision – เช่นในการใช้ WHERE Clauses) นั้น ข้อมูลชนิด BLOB จะพิจารณาตัวอักษรพิมพ์ใหญ่หรือพิมพ์เล็ก (case-sensitive) ส่วนข้อมูลชนิด TEXT1 จะไม่พิจารณา (case-insensitive) ดังนั้นหากดูในแง่นี้แล้ว ก็สามารถใช้ข้อมูลชนิด TEXT แทน VARCHAR และใช้ BLOB แทน VARCHAR BINARY ได้ แต่ยังมีสิ่งที่ต่างกันอยู่ระหว่าง TEXT กับ VARCHAR และ BLOB กับ VARCHAR BINARY ดังนี้

- BLOB และ TEXT จะไม่มีการตัดช่องว่างออกในขณะที่ทำการบันทึกข้อมูล ขณะที่ VARCHAR จะทำการตัดช่องว่างโดยบันทึกเฉพาะข้อมูลตามความยาวที่มีอยู่จริง
- ไม่สามารถกำหนดค่าเริ่มต้น (default) ให้กับ BLOB และ TEXT ได้ (หมายถึงการกำหนดในขณะสร้างเทเบิล)

ข้อมูลชนิด BLOB และ TEXT ยังแบ่งออกเป็น 4 ชนิดย่อย ตามขนาดของข้อมูลคือ TINYBLOB, BLOB, MEDIUMBLOB,LONGBLOB และ TINYTEXT, TEXT, MEDIUMTEXT,LONGTEXT

ตารางแสดงชนิดข้อมูลสตริง

ชนิดข้อมูล	***ช่วงข้อมูล***
CHAR(M)	1 ถึง 255
VARCHAR(M)	1 ถึง 255
TINYBLOB , TINYTEXT	1 ถึง 255
BLOB, TEXT	1 ถึง 65535
MEDIUMBLOB, MEDIUMTEXT	1 ถึง 16777215
LONGBLOB, LONGTEXT	1 ถึง 4294967295

ข้อมูลชนิด ENUM

ข้อมูลชนิด ENUM ใช้ในกรณีที่ต้องการจัดเก็บข้อมลสตริงที่มีอยู่ในรายการที่กำหนดไว้แล้วเท่านั้น โดยการกำหนดรายการข้อความเหล่านี้จะทำในขณะสร้างเทเบิล ซึ่งข้อความที่กำหนดไว้จะถูกแทนด้วยตัวเลขตั้งแต่ 1 (แทนข้อความแรก) เป็นต้นไป และหากข้อความที่จะเพิ่มเข้าไปไม่ตรงกับรายการที่กำหนดไว้ จะมีการเก็บค่าเป็น 0 ซึ่งหมายถึงค่าว่างเปล่า หรือเมื่อมีการเพิ่มเรคคอร์ดแต่ไม่มีการระบุค่าให้แก่ฟิลด์ชนิด ENUM ค่าที่บันทึกจะเป็น NULL ด้วยวิธีจัดเก็บแบบนี้หากมีเรคอร์ดเป็นจำนวนมาก จะช่วยลดพื้นที่การจัดเก็บลงได้อย่างมาก ตัวอย่างเช่น ในการสำรวจความคิดเห็นของประชาชนเกี่ยวกับกิจกรรมยามว่างหลักเลิกงาน โดยมีหัวข้อให้เลือกดังต่อไปนี้

เล่นกีฬา ดูโทรทัศน์ ฟังเพลง อ่านหนังสือ ทำงานบ้าน ช็อปปิ้ง ทำงานพิเศษ อื่น ๆ

แทนที่จะใช้ข้อมูลชนิด CHAR หรือ VARCHAR เพื่อจัดเก็บข้อมูลเหล่านี้ เราอาจใช้ข้อมูลชนิด ENUM ในกรณีนี้ได้ ดังตัวอย่างที่จะแสดงให้เห็นเป็นขั้นตอนต่อไปนี้

1. สร้างเทเบิล hobbyvote ขึ้นในฐานข้อมูล test โดยกำหนดให้มีฟิลด์ ID ซึ่งบอกถึงลำดับที่ของผู้ที่เข้ามาตอบแบบสอบถาม และฟิลด์ hobby ซึ่งเป็นคอลัมน์ชนิด ENUM และเก็บรายการกิจกรรมที่จะให้ผู้ตอบแบบสอบถามเลือก

```
use test ;
create table hobbyvote (ID int not null auto_increment primary key,
Hobby enum('เล่นกีฬา', 'ดูโทรทัศน์', 'ฟังเพลง', 'อ่านหนังสือ', 'ทำงานบ้าน', 'ช็อปปิ้ง', 'ทำงานพิเศษ' ,
'อื่น ๆ')) ;
```

2. เพิ่มข้อมูลเรคอร์ดแรก สมมติว่าผู้ตอบแบบสอบถามเลือกกิจกรรม 'เล่นกีฬา' (แทนค่าด้วย 1) และผู้ตอบแบบสอบถามคนที่สองเลือกกิจกรรม 'อ่านหนังสือ' (แทนค่าด้วย 4)

```
insert into hobbyvote set hobby = 1 ;
insert into hobbyvote set hobby = 4;
```

หรือเพิ่มข้อมูลโดยการระบุข้อความกิจกรรมเข้าไปเลย เช่น

```
insert into hobbyvote set hobby = 'อื่น ๆ' ;
```

3. ตรวจสอบข้อมูลที่ได้บันทึกเข้าไป

```
Select * from hobbyvote ;
```

ด้วยคุณสมบัติของข้อมูลชนิด ENUM จึงเหมาะที่จะนำมาใช้ในลักษณะของลิสต์บ็อกซ์ แต่จะสามารถเลือกได้เพียง 1 ตัวเลือกเท่านั้น หากต้องการให้ผู้ตอบแบบสอบถาม(หรือผู้ใช้ ในกรณีอื่น ๆ) สามารถเลือกตอบได้หลายข้อจะต้องใช้ข้อมูลชนิด SET แทน

ENUM สามารถเก็บรายการข้อความได้สูงสุด 65,535 ข้อความ (ตัวเลือก) ซึ่งน่าจะมากพอสำหรับการใช้งานทั่ว ๆ ไป ขณะที่ SET สามารถเก็บได้เพียง 64 ข้อความ

ข้อมูลชนิด SET

ข้อมูลชนิด SET มีคุณสมบัติที่ต่างไปจาก ENUM คือผู้ใช้สามารถเลือกข้อความ (ที่ระบุไว้แล้วในตอนสร้างเทเบิล) ได้มากกว่า 1 ข้อความ จึงเหมาะที่จะนำมาใช้ในลักษณะของเช็คบ็อกซ์

การบันทึกข้อมูลเข้าไปในฟิลด์ชนิด SET มีสองวิธี โดยบันทึกเป็นตัวเลขหรือบันทึกเป็นข้อความ เช่นเดียวกับ ENUM แต่เนื่องจากคุณลักษณะที่เก็บตัวเลือกได้มากกว่า 1 ค่า ทำให้มีรูปแบบของตัวเลขหรือข้อความที่ใช้ในการบันทึกต่างออกไปจาก ENUM ตัวอย่างเช่น การสำรวจกิจกรรมยามว่างหลักเลิกงานของกลุ่มตัวอย่าง ในตัวอย่างก่อนหน้านี้ ทดลองทำตามขั้นตอนต่อไปนี้โดยให้ฟิลด์ hobby เป็นข้อมูลชนิด SET ดังนี้

1. สร้างเทเบิล hobbiesvote ขึ้นในฐานข้อมูล test

```
use test ;
create table hobbiesvote (ID int not null auto_increment primary key,
Hobby set('เล่นกีฬา', 'ดูโทรทัศน์', 'ฟังเพลง', 'อ่านหนังสือ', 'ทำงานบ้าน', 'ช็อปปิ้ง', 'ทำงานพิเศษ' ,
'อื่น ๆ')) ;
```

เราอาจจะออกแบบตัวเลือกเป็นเช็คบ็อกซ์เพื่อให้ผู้ตอบแบบสอบถามเลือกได้มกกว่า 1 ตัวเลือก โดยแต่ละตัวเลือก (สมาชิกของเซ็ต) จะมีค่าเป็นเลขฐานสอบและเลขฐานสิบ ดังนี้

สมาชิก	*ค่าฐานสอง*	*ค่าฐานสิบ*
เล่นกีฬา	00000001	20 = 1
ดูโทรทัศน์	00000010	21 = 2
ฟังเพลง	00000100	22 = 4
อ่านหนังสือ	00001000	23 = 8
ทำงานบ้าน	00010000	24 = 16
ช็อปปิ้ง	00100000	25 = 32
ทำงานพิเศษ	01000000	26 = 64
อื่น ๆ	10000000	27 = 128

2. เพิ่มข้อมูลเรคอร์ดแรก สมมติว่าผู้ตอบแบบสอบถามเลือกกิจกรรม ('เล่นกีฬา', 'ทำงานบ้าน' และ 'ช็อปปิ้ง') ค่าที่ได้จะเป็นผลรวม (เลขฐานสิบ) ของ 1 (เล่นกีฬา), 16 (ทำงานบ้าน), 32 (ช็อปปิ้ง) เท่ากับ 49

```
insert into hobbiesvote set Hobby = 49 ;
```

และลองเพิ่มข้อมูลโดยการระบุข้อความกิจกรรมเข้าไปเลย เช่น

```
insert into hobbiesvote set Hobby = ('เล่นกีฬา, ทำงานบ้าน, ช็อปปิ้ง') ;
```

3. ตรวจสอบข้อมูลที่ได้บันทึกเข้าไป

```
Select * from hobbiesvote ;
```

7.2.2 การจัดการฐานข้อมูลและเทเบิล

ก่อนที่จะใช้คำสั่งใด ๆ ของ MySQL จะต้องทำการเชื่อมต่อกับ MySQL Server ก่อน ซึ่งสามารถเชื่อมต่อจากเครื่องที่ทำหน้าที่เป็นเซิร์ฟเวอร์เอง (localhost) หรือจากเครื่องอื่นที่อยู่ในเครือข่าย (เช่นใน LAN หรืออินเทอร์เน็ต) ก็ได้ ขึ้นอยู่กับการกำหนดสิทธิว่าได้กำหนดให้ผู้ใช้รายนั้นสามารถเชื่อมต่อเข้ามาจากเครื่องใดได้บ้าง

ตัวอย่างการล็อกอินในฐานะ root

```
mysql --user = root -p
```

จะขึ้นข้อความให้ป้อนรหัสผ่าน ดังนี้

```
Enter  password :
```

เมื่อพิมพ์รหัสผ่านถูกต้องแล้วจะขึ้นข้อความแสดงว่าการเชื่อมต่อกับ MySQL Server สำเร็จ ดังนี้

```
Welcome to the MySQL monitor, Commands end with ; or \g.
Your MySQL connection id is 4072 to server version : 3.23.49 –log
Type ‘help’ or ‘\h’ for help. Type ‘\c’ to clear the buffer.
mysql>
```

<u>การสร้างและลบฐานข้อมูล (DATABASE)</u>

การสร้างฐานข้อมูลขึ้นใหม่จะใช้คำสั่ง CREATE DATABASE ซึ่งมีรูปแบบคำสั่ง ดังนี้

รูปแบบ

```
CREATE  DATABASE  [IF NOT EXISTS]  db_name
```

จะทำให้ได้ไดเรคทอรี่ว่าง ๆ ชื่อเดียวกับฐานข้อมูลที่สร้างขึ้นมาไดเร็คทอรี่หนึ่ง เช่น การสร้างฐานข้อมูลชื่อ company_db ดังนี้

```
Create  database  company_db
```

หากไม่มีฐานข้อมูล (ไดเร็คทอรี่ย่อย) ชื่อว่า company_db อยู่ จะทำการสร้างขึ้นมาให้ แต่หากมีฐานข้อมูลนี้อยู่แล้วจะแสดงข้อผิดพลาด เว้นแต่ว่าจะใส่ IF NOT EXISTS ต่อท้ายคำสั่ง จึงจะไม่เกิดข้อผิดพลาดในกรณีมีฐานข้อมูลชื่อนั้นอยู่ก่อนแล้ว ดังนี้

```
Create database  if  not  exists  company_db
```

สำหรับการลบฐานข้อมูลจะใช้รูปแบบคำสั่ง ดังนี้

```
DROP  DATABASE  [IF  EXISTS]  db_name
```

<u>การสร้างเทเบิล (TABLE)</u>

การสร้างเทเบิลโดยปกติจะสร้างในขณะที่เปิดฐานข้อมูลใดฐานข้อมูลหนึ่งอยู่ (ใช้คำสั่ง use db_name เพื่อเปิดฐานข้อมูลที่ต้องการ) แต่นับจาก MySQL เวอร์ชั่น 3.22 เป็นต้นมา ได้อนุญาตให้สามารถสร้างเทเบิลด้วยการระบุชื่อฐานข้อมูลลงไปได้ โดยไม่จำเป็นจะต้องเปิดฐานข้อมูลเอาไว้ล่วงหน้า ดังนี้ db_name.table_name การสร้างเทเบิลมีคำสั่งปลีกย่อยมากมาย คำสั่งหลักที่ใช้ในการสร้างเทเบิลคือ CREATE TABLE ซึ่งมีรูปแบบดังนี้

รูปแบบ

```
CREATE      [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
            [(create_definition, …)]
            [table_options]
            [select_statement]
```

ความหมายของส่วนต่าง ๆ ในคำสั่งคือ

- TEMPORARY

เริ่มมีตั้งแต่ MySQL เวอร์ชั่น 3.22 ใช้ระบุเมื่อต้องการสร้างเทเบิลขึ้นมาเป็นการชั่วคราว โดยเทเบิลชั่วคราวนี้จะมองไม่เห็นด้วยคำสั่ง show tables แต่สามารถใช้คำสั่ง describe table_name เพื่อดูโครงสร้างของเทเบิลได้ เทเบิลชั่วคราวนี้สามารถนำมาใช้งานได้เหมือนกับเทเบิลทั่ว ๆ ไป แต่เมื่อการเชื่อมต่อถูกตัดขาด เทเบิลจะถูกลบโดยอัตโนมัติ

- IF NOT EXISTS

ใช้ IF NOT EXISTS เพื่อไม่ให้แสดงข้อผิดพลาดในกรณีที่มีเทเบิลชื่อเดียวกันอยู่ก่อน โดยจะไม่มีการสร้างเทเบิลขึ้นใหม่มาทับเทเบิลเดิมที่มีอยู่แล้ว

- tbl_name

ชื่อเทเบิล ตั้งตามข้อกำหนดของการตั้งชื่อไฟล์ ยกเว้นตัวอักษร / และ . โดยความยาวของชื่อต้องไม่เกิน 64 ตัวอักษร

- create_definition ประกอบด้วย 3 ส่วนหลัก คือ
  - ชื่อคอลัมน์ (ฟิลด์) ความยาวไม่เกิน 64 ตัวอักษร
  - ชนิดข้อมูลของคอลัมน์
  - ข้อกำหนดอื่น ๆ ได้แก่

```
[NOT NULL | NULL]
[DEFAULT default_value]
[AUTO_INCREMENT]
[PRIMARY KEY] [reference_definition]
        or PRIMARY KEY (index_col_name, …)
        or KEY [index_name] (index_col_name, …)
        or INDEX [index_name] (index_col_name, …)
        or UNIQUE [INDEX] [index_name] (index_col_name, …)
        or FULLTEXT [INDEX] [index_name] (index_col_name, …)
        or [CONSTRAINT symbol] FOREIGN KEY [index_name]
                (index_col_name, …) [reference_definition]
        Or CHECK (expr)
```

เมื่อเทเบิลถูกสร้างขึ้นจะเกิดไฟล์ขึ้น 3 ไฟล์ ภายในไดเร็คทอรี่ของฐานข้อมูล (ไดเร็คทอรี่ที่มีชื่อเดียวกันกับฐานข้อมูล ซึ่งไฟล์ทั้งสามจะมีชื่อเหมือนกับชื่อเทเบิล แต่จะมีนามสกุลเป็น FRM, MYD และ MYI โดยไฟล์ .FRM จะเก็บข้อกำหนดหรือโครงสร้างของเทเบิล, ไฟล์ .MYD จะทำหน้าที่เก็บข้อมูล ส่วนไฟล์ .MYI จะทำหน้าที่เป็นไฟล์อินเด็กซ์ (index file)

ตัวอย่างการสร้างเทเบิล

```
create table if not exists saleorder
        (OrderNo varchar(15) primary key,
        CustomerNo varchar(20),
        OrderDate datetime,
        PromiseDate date,
        Note varchar(80)) ;
```

จากตัวอย่าง หากมีเทเบิลชื่อ saleorder อยู่ก่อนจะไม่เกิดข้อผิดพลาด เนื่องจากมีการระบุ if not exists ไว้ แต่หากไม่มีเทเบิลชื่อ saleorder อยู่ก่อนจะทำการสร้างเทเบิล saleorder ขึ้นโดยมี 4 คอลัมน์ และคอลัมน์ OrderNo เป็น primary key ซึ่งหมายความว่า ค่าในคอลัมน์ OrderNo ของแต่ละเรคอร์ดจะซ้ำกันไม่ได้ (Unique) และจะมีค่าเป็น NULL ไม่ได้ (ปล่อยว่างไม่ได้)

ในกรณีที่ต้องการกำหนดให้คอลัมน์มากกว่า 1 คอลัมน์เป็น primary key สามารถทำได้ดังนี้

```
create  table  saleorder_detail
        (OrderNo  varchar(15)  not null,
        SequenceNo  int(3)  not null,
        ItemNo  varchar(20),
        Qty  double(10,2),
        Primary  key  (OrderNo, SequenceNo)) ;
```

ตัวอย่างนี้เป็นการสร้างเทเบิลที่มีคอลัมน์ OrderNo และ SequenceNo เป็น primary key ซึ่งจะต้องกำหนดให้คอลัมน์ทั้งสองเป็น not null (ไม่รับค่า NULL) ด้วย มิฉะนั้น MySQL จะแสดงข้อผิดพลาดออกมา เทเบิล saleorder_detail ที่สร้างขึ้นนี้ ค่าของคอลัมน์ OrderNo ในแต่ละเรคอร์ดอาจซ้ำกันได้ และค่าของคอลัมน์ SequenceNo ในแต่ละเรคอร์ดอาจซ้ำกันได้เช่นกัน แต่จะไม่สามารถมีเรคอร์ดที่มีค่าซ้ำกันได้ทั้งในคอลัมน์ OrderNo และ SequenceNo ได้

ถ้าต้องการกำหนดให้คอลัมน์เพิ่มค่าขึ้นทีละ 1 โดยอัตโนมัติเมื่อมีการเพิ่มเรคอร์ด ให้กำหนด AUTO_INCREMENT และถ้าต้องการกำหนดค่าเริ่มต้นให้แก่คอลัมน์ ให้ระบุ DEFAULT ตามด้วยค่าเริ่มต้นที่ต้องการ ดังนี้

```
create  table  saleorder_detail
        (ID  int  auto_increment  primary key,
        OrderNo  varchar(15)  not null,
        SequenceNo  int(3)  not null,
        ItemNo  varchar(20),
        Qty  double(10,2),
        UnitPrice  double(14,4),
        Amount  double(14,4),
        OrderStatus  char(1)  default 'A') ;
```

เมื่อเรียกดูโครงสร้างของเทเบิลด้วยคำสั่ง

```
describe  saleorder_detail ;
```

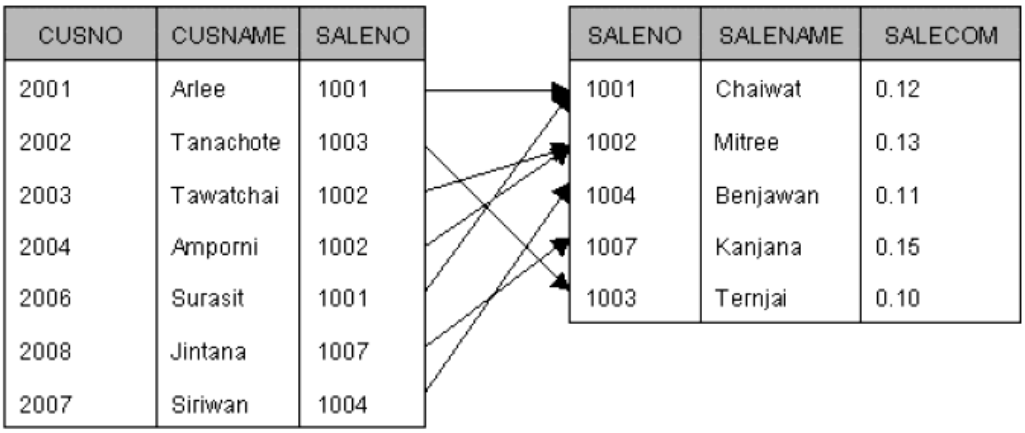
จะแสดงรายละเอียด ดังรูป

Field	Type	Null	Key	Default	Extra
ID	int(11)		PRI	NULL	auto_increment
OrderNo	varchar(15)				
SequenceNo	int(3)			0	
ItemNo	varchar(20)	YES		NULL	
Qty	double(10,2)	YES		NULL	
UnitPrice	double(14,4)	YES		NULL	
Amount	double(14,4)	YES		NULL	
OrderStatus	char(1)	YES		A	

เทเบิล saleorder_detail นี้ จะมีคอลัมน์ ID เป็น primary key และเป็นค่าตัวเลขที่เพิ่มขึ้นทีละ 1 โดยอัตโนมัติเมื่อมีการเพิ่มเรคอร์ดเข้าไป และคอลัมน์ OrderStatus จะถูกกำหนดค่าเริ่มต้นเป็น 'A' หากไม่มีการระบุค่า ในขณะที่มีการเพิ่มเรคอร์ด

นอกจากการสร้างเทเบิลโดยการกำหนด primary key แล้ว เรายังสามารถสร้างเทเบิลโดยการกำหนด คีย์นอก **(Foreign key)** ได้อีกด้วย ซึ่งประโยชน์ของการกำหนดคีย์นอกนี้ จะทำให้เราสามารถอ้างอิงการใช้งานของข้อมูล ระหว่างเทเบิล 2 เทเบิลได้

คีย์นอกเป็นคอลัมน์ของตารางหนึ่งที่ใช้เชื่อมโยงหรืออ้างอิง ข้อมูลกับอีกตารางหนึ่งที่มีคอลัมน์ซึ่ง มีชื่อคอลัมน์เดียวกัน 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 เป็นตารางลูกค้าและตารางพนักงานขายโดยในที่นี้จะไม่กล่าวถึงคอลัมน์ต่างๆที่ไม่จำเป็นเพื่อสะดวกในการศึกษา ลูกค้าในตารางลูกค้าแต่ละคนมีคอลัมน์ SALENO (เป็นคอลัมน์เดียวกันกับคอลัมน์ SALENO ที่อยู่ในตารางพนักงานขาย) คอลัมน์ SALENO ที่อยู่ตารางลูกค้าเป็นคอลัมน์ที่แสดงพนักงานขายที่กำหนดให้กับลูกค้าแต่ละคน

ตัวอย่าง ถ้าต้องการสร้างตาราง (CUSTOMERSTAB) โดยกำหนดให้คอลัมน์ CUSNO เป็น PRIMARY KEY และคอลัมน์ SALENO เป็นคีย์นอก (foreign key) ที่ ใช้เชื่อมโยงหรืออ้างอิงข้อมูลกับตารางพนักงานขาย(SALESTAB) โดยใช้คำสั่งดังนี้

```
CREATE TABLE customertab
        (Cusno int(4) NOT NULL PRIMARY KEY,
        Cusname char(10),
        Saleno int(4),
        FOREIGN KEY(Saleno) REFERENCES SALESTAB(Saleno)) ;
```

```
CREATE TABLE customertab
        (Cusno int(4) NOT NULL PRIMARY KEY,
        Cusname char(10),
        Saleno int(4) REFERENCES SALESTAB(Saleno)) ;
```

การแก้ไขโครงสร้างของเทเบิล (ALTER)

เมื่อต้องการเปลี่ยนแปลงโครงสร้างของเทเบิล เช่น เพิ่มหรือลบคอลัมน์ เปลี่ยนชื่อคอลัมน์ เปลี่ยนชนิดข้อมูลของคอลัมน์ สร้างหรือลบอินเด็กซ์ เป็นต้น จะใช้คำสั่ง ALTER TABLE ซึ่งมีรูปแบบดังนี้

รูปแบบ

```
ALTER [IGNORE] TABLE tbl_name
        alter_specification [, alter_specification …]
```

ความหมายของส่วนต่าง ๆ ในคำสั่ง ALTER TABLE มีดังนี้

tbl_name ชื่อเทเบิลที่ต้องการแก้ไขโครงสร้าง

alter_specification แบ่งเป็นกลุ่มคำสั่ง ADD, ALTER, CHANGE, MODIFY, DROP, RENAME ซึ่งมีรูปแบบและรายละเอียดดังนี้

- การเพิ่มคอลัมน์

รูปแบบ

```
ADD [COLUMN] create_definition [FIRST | AFTER column_name]
```

ใช้เพิ่มคอลัมน์ใหม่ให้แก่เทเบิล โดยที่ create_definition จะเป็นการระบุชื่อคอลัมน์ ชนิด และขนาดข้อมูลเช่นเดียวกับการใช้คำสั่ง CREATE TABLE เพื่อสร้างเทเบิล ส่วน FIRST จะเป็นการกำหนดให้คอลัมน์ที่เพิ่มขึ้นใหม่อยู่ตำแหน่งคอลัมน์แรก และ AFTER column_name เป็นการกำหนดให้คอลัมน์ที่เพิ่มขึ้นใหม่อยู่ตำแหน่งต่อจากคอลัมน์ column_name ทั้งนี้หากไม่มีการระบุตำแหน่ง คอลัมน์ที่ถูกเพิ่มขึ้นใหม่จะเป็นคอลัมน์สุดท้ายของเทเบิล

ตัวอย่าง

```
alter table table_a add field0 varchar(10) first ;
alter table table_a add field5 int after field4 ;
```

- การเพิ่มอินเด็กซ์

รูปแบบ

```
ADD INDEX [index_name] (col_name, …)
```

ใช้เพิ่มอินเด็กซ์ (index) โดยการระบุชื่อคอลัมน์ (col_name) ที่ต้องการทำอินเด็กซ์

ตัวอย่าง

```
alter table table_a add index (field0);
```

- การสร้าง primary key

รูปแบบ

```
ADD PRIMARY KEY (col_name, …)
```

ใช้สร้าง primary key โดยการระบุคอลัมน์หนึ่งหรือหลาย ๆ คอลัมน์ให้เป็น primary key

ตัวอย่าง

```
alter table table_a add primary key (field0, field1) ;
```

หากทดลองตามตัวอย่างนี้ MySQL อาจแสดงข้อผิดพลาด เนื่องจากคอลัมน์ที่จะเป็น primary key จะต้องไม่เป็นค่า NULL ซึ่งแก้ไขได้โดยเปลี่ยนแปลงคุณสมบัติของคอลัมน์ field0 และ field1 ดังนี้

```
alter table table_a modify field0 varchar(10) not null,
modify field1 int not null ;
```

แล้วจึงลองกำหนด primary key ใหม่อีกครั้งหนึ่ง

- การสร้าง unique index

รูปแบบ

```
ADD UNIQUE [index_name] (col_name, …)
```

ใช้สร้าง unique index โดยการระบุคอลัมน์หนึ่งหรือหลาย ๆ คอลัมน์

- การกำหนดหรือยกเลิกค่าเริ่มต้นของคอลัมน์

รูปแบบ

```
ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
```

ตัวอย่าง

```
alter table table_a alter field2 set default 'noname' ;
alter table table_a alter field2 drop default ;
```

- การแก้ไขคอลัมน์ (แบบที่ 1)

รูปแบบ

```
CHANGE [COLUMN] col_name create_definition
```

ใช้เปลี่ยนชื่อ ชนิด และขนาดข้อมูลของคอลัมน์ (col_name) ให้เป็นไปตามกำหนดไว้ใน create_definition

ตัวอย่าง

```
alter table table_a change field2 field_new tinyint(1) ;
```

ตามตัวอย่างนี้จะเปลี่ยนชื่อคอลัมน์ field2 เป็น field2_new พร้อมทั้งเปลี่ยนชนิดและขนาดข้อมูลเดิมให้เป็น tinyint(1) ด้วย

- การแก้ไขคอลัมน์ (แบบที่ 2)

รูปแบบ

```
MODIFY create_definition
```

มีความหมายเช่นเดียวกับ CHANGE ต่างกันที่ MODIFY จะไม่สามารถเปลี่ยนชื่อของคอลัมน์ได้

- การลบคอลัมน์

รูปแบบ

```
DROP [COLUMN] col_name
```

- การลบ primary key

รูปแบบ

```
DROP  PRIMARY  KEY
```

ใช้ลบ primary key ซึ่งหมายถึงยกเลิกคุณสมบัติการเป็น primary key ของคอลัมน์เท่านั้น ไม่ใช่การลบคอลัมน์

- การลบ index

รูปแบบ

```
DROP  INDEX  index_name
```

การเปลี่ยนชื่อเทเบิล

การเปลี่ยนชื่อเทเบิลจะใช้คำสั่ง RENAME TABLE ซึ่งสามารถเปลี่ยนชื่อได้หลายเทเบิลในคราวเดียวกัน มีรูปแบบการใช้ ดังนี้

รูปแบบ

```
RENAME  TABLE  tbl_name  TO  new_tbl_name [, tbl_name2 TO new_tabl_name2, …]
```

ตัวอย่าง

```
RENAME  TABLE  table_a  to  table_b ;
```

การลบเทเบิล

การลบเทเบิลด้วยคำสั่ง DROP TABLE มีรูปแบบการใช้งานดังนี้

รูปแบบ

```
DROP  TABLE  [IF  EXISTS]  tbl_name [, tbl_name, …]
```

คำสั่ง DROP สามารถลบเทเบิลได้หลายเทเบิลในคราวเดียวกัน ส่วนคีย์เวิร์ด IF EXISTS ซึ่งเริ่มมีให้ใช้ได้ใน MySQL 3.22 จะช่วยป้องกันไม่ให้มีการแสดงข้อผิดพลาดในกรณีที่ไม่มีเทเบิลชื่อนั้นอยู่ในฐานข้อมูล

7.2.3 การดำเนินการกับข้อมูล

การดำเนินการกับข้อมูลในระบบฐานข้อมูล MySQL จะใช้คำสั่ง SQL ซึ่งประกอบด้วย การเพิ่มข้อมูล การลบข้อมูล การแก้ไขข้อมูล และการเรียกดูข้อมูล ในที่นี้จะกล่าวถึงเพียงให้สามารถใช้งานในขั้นพื้นฐานได้เท่านั้นส่วนรายละเอียดสามารถศึกษาได้จากคู่มือการใช้งาน MySQL ที่จัดทำโดย MySQL AB โดยดาวน์โหลดได้ที่ www.mysql.com/documentation

การเพิ่มข้อมูล (คำสั่ง INSERT INTO)

คำสั่งการเพิ่มข้อมูลในตารางจะใช้คำสั่ง **INSERT INTO** จะมีอยู่ 2 รูปแบบคือ การเพิ่มข้อมูลเข้าไปทีละแถว และ การเพิ่มข้อมูลโดยการดึงกลุ่มข้อมูลด้วยคำสั่งค้นหาข้อมูล

รูปแบบการใช้คำสั่ง INSERT INTO ทีละแถว

```
INSERT  INTO  table_name  SET  col_name = expression          หรือ

INSERT  INTO table_name VALUES(col_value)
```

ตัวอย่าง เพิ่มข้อมูลทุกคอลัมน์ลงในตารางลูกค้า

```
INSERT  INTO  SALESTAB  SET  SALENO = 1001, SALENAME = 'Chaiwat',
ADDRESS = 'Bangkok', SALECOM = 0.12 ;
```

ตัวอย่าง ถ้าต้องการจะใส่ข้อมูลทุกคอลัมน์ลงในตารางลูกค้า

```
INSERT  INTO  SALESTAB  VALUES(1001, 'Chaiwat', 'Bangkok',0.12 );
```

ผลของคำสั่งของการใช้ insert into สองแบบด้านบน

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);
```

ผลของคำสั่ง

CUSNO	CUSNAME	ADDRESS	RATING	SALENO
2001	Arlee	Bangkok	0	[NULL]

รูปแบบการใช้คำสั่ง INSERT INTO โดยการดึงกลุ่มข้อมูลด้วยคำสั่งค้นหาข้อมูล

```
INSERT  INTO  table_name1  SELECT  col_name FROM table_name2 WHERE
col_name = value ;
```

คำสั่งการเพิ่มข้อมูลโดยการดึงกลุ่มข้อมูลด้วยคำสั่งค้นหาข้อมูล ในภาษา SQL สามารถใช้คำสั่ง INSERT ในการนำค่าหรือหาค่าจากตารางหนึ่งแล้วไปใส่ไว้ในอีกตารางหนึ่งได้ โดยได้ค่านั้นมาจากการสอบถามข้อมูล

ตัวอย่าง ถ้าต้องการใส่ข้อมูลพนักงานลงในตาราง BANGKOKSTAFF โดยข้อมูลที่จะใส่ลงไปนั้นได้มาจากตารางพนักงานขายที่อาศัยอยู่ใน 'Bangkok'

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

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

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

<u>การปรับปรุงข้อมูล (คำสั่ง UPDATE)</u>

หลังจากที่ป้อนข้อมูลเข้าไปเก็บไว้ในตารางแล้ว กรณีที่ต้องการปรับปรุงแก้ไข ข้อมูลสามารถทำได้ด้วยภาษา SQL การปรับปรุงแถวข้อมูลเป็นการปรับปรุงหรือ แก้ไขค่าคอลัมน์ ซึ่งในคำสั่งปรับปรุงข้อมูลอาจมีมากกว่า 1 คอลัมน์ในแถวทุกแถว ที่มีเงื่อนไขสอดคล้องกับที่ระบุไว้หลังคำว่า **WHERE**

รูปแบบการใช้คำสั่ง UPDATE

```
UPDATE table_name SET col_name = expression WHERE where_definition
```

<u>ตัวอย่าง</u> ถ้าต้องการเปลี่ยนค่า RATING ของลูกค้าทั้งหมดในตารางลูกค้าให้เป็น 200 จะต้องป้อนคำสั่งดังนี้

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

ผลของคำสั่งจะทำให้คอลัมน์ **RATING** ของตารางลูกค้ามีค่าเป็น **200** ทุกแถว และเมื่อเข้าไปดูข้อมูลในตารางลูกค้าจะปรากฏข้อมูลดังนี้

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

ตัวอย่าง ถ้าต้องการจะเปลี่ยนค่า **RATING** ให้กับลูกค้าทั้งหมด ที่มีหมายเลข ประจำตัวพนักงานขาย **(SALENO)** เป็น **1001** ให้มีค่า **RATING** เป็น **200**

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

ผลของคำสั่งจะทำให้ตารางลูกค้าเดิมเปลี่ยนเป็นตารางใหม่ ในตารางใหม่นี้ 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

การลบข้อมูลทั้งแถว (DELETE FROM)

คำสั่งในการลบแถวข้อมูล เป็นคำสั่งที่ใช้ในการลบแถวข้อมูลทุกแถวที่มีเงื่อนไข สอดคล้องกับที่ระบุไว้หลัง **WHERE**

รูปแบบการใช้คำสั่ง DELETE

```
DELETE FROM table_name WHERE where_definition
```

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 ออกจากตารางด้วยคำสั่ง **DELETE FROM table_name** โดยไม่ระบุ WHERE คำสั่งนี้จะลบแถวทุกแถวที่อยู่ในเทเบิลจนหมด แต่ยังคงเหลือโครงสร้างเปล่า ๆ ของเทเบิลไว้

ตัวอย่าง ถ้าต้องการลบรายละเอียดทั้งหมดของตารางพนักงานขายจะต้องป้อนคำสั่งต่อไปนี้

```
DELETE  FROM   SALESTAB ;
```

การเรียกค้นข้อมูล (Query หรือ การใช้คำสั่ง SELECT)

รูปแบบการใช้คำสั่ง SELECT

```
SELECT  col_name  FROM  table_name
            WHERE  where_definition
             ORDER  BY  col_name ;
```

คำสั่ง SELECT ใช้สำหรับเรียกดูข้อมูลในเทเบิล โดยสามารถระบุฟิลด์ที่ต้องการเรียกดู และสามารถให้จัดเรียงเรคอร์ดตามค่าในฟิลด์ที่ต้องการได้ ซึ่งเรคอร์ดที่แสดงจะเป็นเรคอร์ดที่ตรงตามเงื่อนไขที่กำหนดไว้ในส่วน where_definition แต่หากไม่มีการระบุเงื่อนไขในส่วนนี้จะเป็นการเรียกดูข้อมูลทุก ๆ เรคอร์ดในเทเบิล

➢ **การเรียกค้นดูทุกคอลัมน์ในตาราง**

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

<u>รูปแบบ</u>

SELECT * FROM <table name>;

SELECT * เป็นคำสั่งที่ต้องมีทุกครั้งที่ต้องการเรียกค้นข้อมูลทุกคอลัมน์

FROM เป็นการกำหนดว่าให้เรียกดูข้อมูล ได้จากตารางใดบ้าง

table name ชื่อตารางที่ต้องการเรียกค้นข้อมูล

<u>ตัวอย่าง</u> ถ้าต้องการดูทุกคอลัมน์ในตารางก็จะใช้เครื่องหมายดอกจัน(*) แทนรายการคอลัมน์ได้ทั้งหมดได้ดังนี้

```
SELECT  *  FROM  CHECKS
```

<u>ตัวอย่าง</u> 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

➢ **การเรียกค้นข้อมูลเฉพาะคอลัมน์ใดๆในตารางและการเปลี่ยนลำดับคอลัมน์**

การใช้คำสั่ง **SELECT** ในการเรียกค้นข้อมูลเฉพาะคอลัมน์ที่สนใจทำได้โดยใส่ เฉพาะคอลัมน์ที่ต้องการดูในส่วนของคำสั่ง **SELECT**

<u>ตัวอย่าง</u> ถ้าต้องการแสดงข้อมูลบางคอลัมน์ เช่น ถ้าต้องการดูคอลัมน์ CHECK และ AMOUNTใช้คำสั่งดังนี้

```
SELECT  check, amount  FROM  CHECKS ;
```

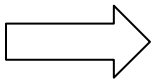
CHECK#	AMOUNT
1	150
2	24534
3	20032
4	98

- **การเรียกค้นข้อมูลด้วยคำสั่ง DISTINCT**

คำสั่ง Distinct เป็นคำสั่งที่ใช้สำหรับกรองข้อมูลของคอลัมน์ที่เราระบุในคำสั่ง Select ไม่ให้มีค่าของข้อมูลที่ซ้ำกัน

```
SELECT  DISTINCT  amount  FROM CHECKS ;
```

amount
150
24534
20032
98
150
25
251



amount
150
24534
20032
98
25
251

- **การใช้คำสั่ง SELECT กับ WHERE**

การใช้ **WHERE** ในคำสั่ง **SELECT** จะช่วยให้สามารถสืบค้นข้อมูลได้อย่างเจาะจงมากกว่า เช่น ถ้าใช้เฉพาะ **SELECT** อย่างเดียวจะได้ข้อมูลทั้งหมด ตัวอย่างเช่น

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

ตัวอย่าง ถ้าต้องการจะดูเฉพาะข้อมูลของ **"BURLEY"** เท่านั้นเราจะต้องใช้ คำสั่ง **WHERE** ดังนี้

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

ผลลัพธ์

NAME	FRAMESIZE	COMPOSITION	MILESRIDDEN	TYPE
BURLEY	22	STEEL	2000	TANDEM

ถ้าต้องการใส่เงื่อนไข หลัง คำสั่ง WHERE มากกว่า 1 คอลัมน์ ให้ใส่คำสั่ง AND หรือ OR ดังตัวอย่าง

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 ถ้าต้องการแสดงค่าของคอลัมน์ PAYEE และ AMOUNT ที่มีค่าของ AMOUNT เท่ากับ 150 **และ** PAYEE เท่ากับ Malee Benjanee สามารถเขียนได้ ดังนี้

```
SELECT  payee, amount  FROM  checks
WHERE  amount = 150  AND  payee = 'Malee Benjanee' ;
```

ตัวอย่าง ถ้าต้องการแสดงค่าของคอลัมน์ PAYEE และ AMOUNT ที่มีค่า AMOUNT = 150 **หรือ** AMOUNT = 98 จะเขียนคำสั่งได้ดังนี้

```
SELECT  payee, amount  FROM  checks
WHERE  amount = 150  OR  amount = 98 ;
```

➢ **โอเปอร์เรเตอร์คณิตศาสตร์** ได้แก่ **+, -, *, /, %**

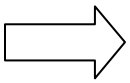
ตัวอย่าง ถ้าในคำสั่งในคอลัมน์ **WHOLESALE** ต้องการ บวก **15** เข้าไป ผลลัพธ์ที่ได้จะแสดงค่าของ **WHOLESALE** ที่บวก **15** เข้าไปโดยคอลัมน์ แต่จะแสดงเพียงชั่วคราวที่หน้าจอเท่านั้น โดยไม่มีผลต่อข้อมูลของคอลัมน์ **WHOLESALE** ในตาราง **PRICE** คอลัมน์ **WHOLESALE** ในตาราง **PRICE** จะมีค่าเหมือนเดิม

และจากคอลัมน์ **WHOLESALE+15** สามารถให้แสดงผลหน้าจอเป็นชื่อคอลัมน์อื่นได้โดยถ้าต้องการให้ **WHOLESALE +15** และให้แสดงผลเป็นคอลัมน์ **RETAIL** จะใช้คำสั่งดังนี้

```
SELECT  item, wholesale, (wholesale+0.5) RETAIL
FROM   price ;
```

ตาราง PRICE

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



ผลลัพธ์

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

➢ **โอเปเรเตอร์ตัวอักษร**

- ตัวโอเปอเรเตอร์ **LIKE** เป็นการค้นหาข้อมูลของคอลัมน์ที่เก็บข้อมูลประเภทตัวอักษร เท่านั้น โดยไม่ทราบค่าข้อมูลทั้งหมดที่จะค้นหา หรือรู้เพียงบางตัวอักษรเท่านั้น โอเปอร์เรเตอร์ **LIKE** จะระบุต่อท้ายชื่อคอลัมน์ที่เป็นเงื่อนไข โดยจะใช้สัญลักษณ์ที่เป็นตัวค้นหาช่วยในการค้นหาข้อมูลที่เรียกว่า **วิน การ์ด (WILD Card)** สัญลักษณ์ดังกล่าวประกอบด้วย **%** และ **_ (เครื่องหมายขีดเส้นใต้)** โดยข้อมูลบางส่วนที่ใช้ในการค้นหาพร้อมกับสัญลักษณ์ทั้งสองนี้จะต้องมีเครื่องหมาย ‘’ กำกับเสมอ ความหมายของสัญลักษณ์ทั้งสองเป็นดังนี้คือ
  - สัญลักษณ์ **%** ใช้แทนจำนวนอักษรได้หลายตัว เช่น พนักงานขายที่ขึ้นต้นด้วยตัว **T** จะเขียนเงื่อนไขว่า **WHERE SALENAME LIKE ‘T%’**
  - สัญลักษณ์ **_** ใช้แทนจำนวนที่ไม่ทราบค่า **1 ตัว** เช่น พนักงานขายที่มีชื่อขึ้นต้น **S** และ มีความยาว 7 ตัวอักษร เช่น **WHERE SALENAME LIKE ‘S_____’**

ตัวอย่าง ถ้าต้องการหาค่าในคอลัมน์ PAYEE จากตาราง CHECKS ที่มี คำขึ้นต้นด้วย Ca จะใช้คำสั่งดังนี้

```
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

- **การเรียกค้นข้อมูลจากหลายตาราง (Join Table)**

การจัดทำฐานข้อมูลในรูปตารางเกิดจากการที่ข้อมูลได้ออกแบบมาเพื่อลดความซ้ำซ้อน (normalization) ดังนั้นข้อมูลที่มีรายละเอียดของข้อมูลมากอาจจะถูกเก็บไว้ในหลายๆตารางแยก ออกมาต่างหาก เช่น ตารางข้อมูลที่เป็นตารางหลัก(master table) และตารางข้อมูลที่เป็นตารางเชิงรายการ(transaction table) และตารางข้อมูลที่เป็นตารางที่อยู่(address table) เป็นต้น การแยกออกเป็นตารางข้อมูล ย่อยๆนี้นอกจากลดความซ้ำซ้อน แล้วยังช่วยในการประหยัดเนื้อที่ และยัง เพิ่มประสิทธิภาพของฐานข้อมูล

ตัวอย่าง การเรียกดูข้อมูลแบบซ้อนกัน

ตาราง SALETAB

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	Temjai	Nonthaburi	0.10

ตาราง ORDERSTAB

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	109516	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

```
SELECT  orderno, amt, orderdate, cusno, orderstab.saleno
FROM  orderstab, saletab
WHERE  orderstab.saleno  =  saletab.saleno
AND  saletab.address = 'Bangkok' ;
```

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