บทที่ 7 Structured Query Language (SQL)

วัตถุประสงค์ของบทเรียน

- ศึกษาเกี่ยวกับคำสั่งและฟังก์ชันการทำงานพื้นฐานของ SQL
- ศึกษาเกี่ยวกับวิธีการใช้ SQL ในการดูแลรักษาข้อมูลในแง่มุมต่างๆ
- ศึกษาเกี่ยวกับวิธีการใช้ SQL ในการจัดการข้อมูลในแง่มุมต่างๆ
- ศึกษาเกี่ยวกับวิธีการใช้ SQL เพื่อทำการค้นหาข้อมูลและสารสนเทศจากฐานข้อมูล

เนื้อหาของบทเรียน

คำสั่ง SQL ที่เกี่ยวกับ data definition คำสั่ง SQL ที่เกี่ยวกับ data manipulation คิวรีในการเรียกดูข้อมูล คำสั่งเพิ่มเติมเกี่ยวกับ data definition คำสำคัญเพิ่มเติมที่มักถูกใช้ในคำสั่ง SELECT ตารางเสมือนและการสร้างมุมมอง การเชื่อมโยงข้อมูลระหว่างตารางข้อมูลในฐานข้อมูล

กิจกรรมการเรียน-การสอน

- อธิบายพร้อมยกตัวอย่างประกอบ
- ศึกษาจากเอกสารคำสอน
- ฝึกปฏิบัติการตามที่มอบหมาย
- ทำแบบฝึกหัดท้ายบท

อุปกรณ์ที่ใช้ในการเรียน-การสอน

- เอกสารคำสอน
- เครื่องคอมพิวเตอร์
- เครื่องฉายภาพสไลด์

การวัดและประเมินผล

- การตอบคำถามระหว่างการเรียน-การสอน
- การทำแบบทดสอบย่อยท้ายบท
- การตรวจงานตามที่มอบหมาย

เนื้อหาใน 6 บทก่อนหน้าจะเน้นย้ำที่การออกแบบฐานข้อมูลที่จะประยุกต์ใช้แนวความคิดของการสร้างแบบจำลองข้อมูลเพื่อแสดงถึงโครงสร้างการจัดเก็บข้อมูลและความสัมพันธ์ระหว่างข้อมูลต่างๆ แต่อย่างไรก็ตาม การสร้างแบบจำลองข้อมูลจะเปรียบได้กับการร่างแนวความคิดของการจัดเก็บข้อมูลแต่ยังไม่ได้ดำเนินการสร้างโครงสร้างการจัดเก็บข้อมูลจริง ดังนั้น ในบทนี้เราจะทำการศึกษาเกี่ยวกับการดำเนินการสร้างการจัดเก็บข้อมูลจริงด้วยการประยุกต์ใช้ภาษา SQL (Structured Query Language) ที่ซึ่งเป็นภาษาที่ได้รับความนิยมอย่างแพร่หลายในการสร้างฐานข้อมูล

ภาษา SQL เป็นภาษาที่ใช้สำหรับสร้างฐานข้อมูลและสร้างโครงสร้างการจัดเก็บข้อมูลของตารางข้อมูลต่างๆ ใช้สำหรับการจัดการต่างๆกับข้อมูล อาทิเช่น การเพิ่ม ลบ และอัพเดทข้อมูล และยังถูกใช้สำหรับการสืบค้นข้อมูลสารสนเทศจากฐานข้อมูลด้วยเช่นกัน **ภาษา SQL จะมี 2 ฟังก์ชันการทำงานหลัก** ดังนี้

- **Data Definition Language (DDL)**—จะเป็นคำสั่ง SQL ที่เกี่ยวข้องกับ**การสร้างตารางข้อมูล ดัชนี มุมมองต่างๆ และยังรวมถึงคำสั่งสำหรับการกำหนดสิทธิ์ในการเข้าถึงตารางข้อมูลต่างๆ** ในบทนี้เราจะทำการศึกษาเกี่ยวกับคำสั่ง SQL ที่เกี่ยวข้องกับ DDL ดังแสดงในรูป 7.1

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a tables definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

รูปที่ 7.1 คำสั่ง DDL ของภาษา SQL

- **Data Manipulation Language (DML)**—จะเป็นคำสั่ง SQL ที่เกี่ยวข้องกับ**การเพิ่ม อัพเดท ลบ และสืบค้นข้อมูลจากตารางข้อมูลในฐานข้อมูล** โดยคำสั่งประเภท DML ที่เราจะทำการศึกษาในบทนี้จะแสดงได้ดังรูป 7.2

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

รูปที่ 7.2 คำสั่ง DML ของภาษา SQL

คำสั่ง SQL ทั้งสองหมวดหมู่ข้างต้นจะทำการประยุกต์ใช้คำศัพท์เฉพาะไม่ถึง 100 คำที่ซึ่งจะทำให้เราสามารถเข้าใจได้ง่าย คำสั่ง SQL จะสนใจเฉพาะเราต้องการจะทำอะไร เช่น ต้องการเรียกดูข้อมูลอะไร ต้องการเพิ่มข้อมูลอะไรในตารางข้อมูลใด ต้องการสร้างตารางข้อมูลใด เป็นต้น แต่จะไม่สนใจถึงวิธีในการดำเนินการต่างๆเพื่อที่จะได้มาซึ่งสิ่งที่ต้องการ

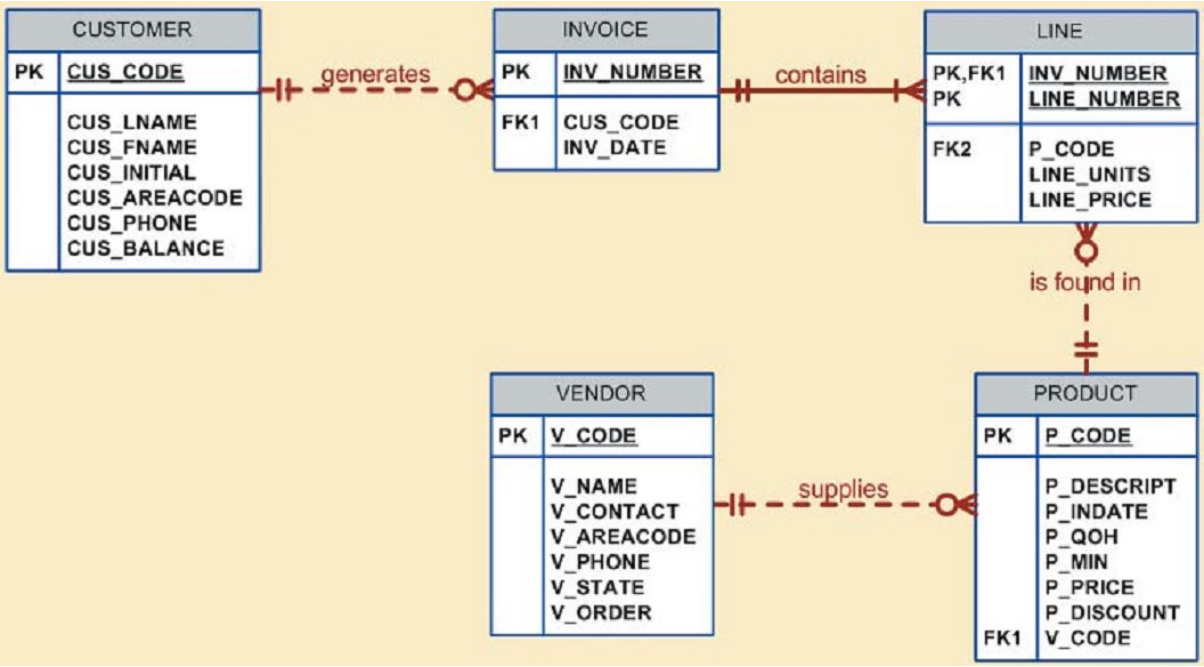
หัวใจหลักของ SQL คือ คิวรี (query) ที่ซึ่งจะครอบคลุมคำถามและการดำเนินการ เช่น รายการสินค้าใดบ้างที่มีราคามากกว่า 100$ ที่ซึ่งถูกจัดเก็บในโกดังสินค้า? และรายการสินค้าเหล่านั้นมีจำนวนเท่าใด? พนักงานที่เริ่มทำงานกับบริษัทหลังจาก 1 January 2015 เป็นจำนวนเท่าใด? เป็นต้น นอกจากคำถามข้างต้น คิวรีของ SQL ยังถูกใช้ในการเพิ่ม ลบ อัพเดทข้อมูลในแถวข้อมูลต่างๆ และยังรวมถึงการสร้างตารางข้อมูลและดัชนี แต่ก่อนที่เราจะประยุกต์ใช้คำสั่ง SQL ในการสืบค้นข้อมูลจากฐานข้อมูล เราควรที่จะต้องทำการกำหนดโครงสร้างการจัดเก็บข้อมูลด้วยการใช้คำสั่งประเภท DDL เสียก่อน

7.1 คำสั่ง SQL ที่เกี่ยวกับ data definition

ก่อนที่เราจะทำการศึกษาคำสั่ง SQL สำหรับการสร้างและการกำหนดส่วนประกอบต่างๆของตารางข้อมูล ลองพิจารณาแบบจำลองข้อมูลในรูป 7.3 เพื่อใช้เป็นตัวอย่างประกอบ โดยจากรูปจะประกอบไป

ด้วย 5 ตารางข้อมูล : CUSTOMER, INVOICE, LINE, PRODUCT และ VENDOR ที่ซึ่งจะสอดคล้องกับกฎเกณฑ์ทางธุรกิจดังต่อไปนี้

- ลูกค้าคนหนึ่งๆสามารถสั่งสินค้าที่ซึ่งจะมีใบแจ้งหนี้ได้หลายใบ และใบแจ้งหนี้หนึ่งๆจะเป็นของลูกค้าคนหนึ่งๆเท่านั้น
- ใบแจ้งหนี้หนึ่งๆจะมีข้อมูลได้หลายรายการ และแต่ละรายการจะเกี่ยวเนื่องกับใบแจ้งหนี้หนึ่งๆเท่านั้น
- แต่ละรายการในใบแจ้งหนี้หนึ่งๆจะแสดงถึงรายการสินค้าหนึ่งๆ แต่รายการสินค้าหนึ่งๆสามารถปรากฏได้หลายรายการในใบแจ้งหนี้ (อาจปรากฏในหลายใบแจ้งหนี้)
- บริษัทผู้ผลิตสินค้าจะสามารถผลิตสินค้าได้หลายรายการให้กับบริษัทของเรา แต่อาจมีบางบริษัทผู้ผลิตสินค้าที่ไม่ได้ผลิตสินค้าให้กับบริษัทของเรา (แต่บริษัทของเรามีการจัดเก็บข้อมูลของบริษัทเหล่านั้นไว้)
- บางรายการสินค้าอาจได้มาจากการผลิตเองของบริษัทเรา บางรายการสินค้าอาจได้มาจากบริษัทผู้ผลิตสินค้า โดยสินค้าที่มาจากผู้ผลิตสินค้าจะมาจากบริษัทเพียงบริษัทเดียวเท่านั้น



รูปที่ 7.3 ตัวอย่างแบบจำลองข้อมูลที่ใช้ในการศึกษาคำสั่ง SQL

จากรูป 7.3 เราจะทำการศึกษาคำสั่ง SQL เพื่อใช้ในการสร้างและกำหนดส่วนประกอบของตาราง PRODUCT และ VENDOR ที่ซึ่งจะมีตัวอย่างข้อมูลดังแสดงในรูป 7.4 ที่ซึ่งเราจะสังเกตุได้ว่า

- ตาราง VENDOR จะมีข้อมูลบริษัทผู้ผลิตสินค้าที่ไม่ได้ทำการผลิตสินค้าให้กับบริษัทของเรา ดังนั้นผู้ออกแบบฐานข้อมูลควรจะพิจารณาว่าตาราง PRODUCT จะมีความสัมพันธ์กับตาราง VENDOR

แบบ optional (ข้อมูลบริษัทสินค้าอาจปรากฏขึ้นโดยไม่มีการเชื่อมโยงความสัมพันธ์กับรายการสินค้าหนึ่งๆ)

- ข้อมูลแอทริบิว V_CODE ในตาราง PRODUCT จะต้องสอดคล้องกับแอทริบิว V_CODE ในตาราง VENDOR เพื่อที่เราจะมีความแน่ใจเกี่ยวกับ referential integrity
- ข้อมูลรายการสินค้าหนึ่งๆอาจมาจากบริษัทผู้ผลิตสินค้า หรืออาจมาจากการผลิตเองของบริษัทของเราที่ซึ่งจะทำให้รายการสินค้าหนึ่งๆไม่จำเป็นต้องมาจากบริษัทผู้ผลิตสินค้าเท่านั้น ด้วยเหตุนี้ ตาราง VENDOR จะมีความสัมพันธ์แบบ optional กับตาราง PRODUCT

Table name: VENDOR **Database name: Ch07_SaleCo**

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-10	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	18	5	119.95	0.10	25595

รูปที่ 7.4 ตัวอย่างข้อมูลในตาราง VENDOR และตาราง PRODUCT

7.1.1 การสร้างฐานข้อมูล

ก่อนที่เราจะประยุกต์ใช้ข้อมูลจาก RDBMS เราจะต้องดำเนินการสร้างโครงสร้างของฐานข้อมูล และทำการสร้างตารางข้อมูลที่ใช้ในการจัดเก็บข้อมูลของผู้ใช้ ในการที่จะดำเนินการสร้างโครงสร้างของฐานข้อมูล RDBMS จะทำการสร้างแฟ้มข้อมูลในดิสก์เพื่อใช้ในการจัดเก็บข้อมูล นอกจากนั้น RDBMS จะทำการสร้างพื้นที่ที่ใช้สำหรับจัดเก็บพจนานุกรม (data dictionary) ให้โดยอัตโนมัติ

7.1.2 Database schema

Schema ภายใต้คำสั่ง SQL จะเป็นกลุ่มของตารางข้อมูลและดัชนีข้อมูลต่างๆในฐานข้อมูลที่ซึ่งจะเป็นตารางข้อมูลและดัชนีของผู้ใช้หรือแอพพลิเคชันหนึ่งๆ โดยฐานข้อมูลหนึ่งๆอาจมีหลาย schema ตามจำนวนผู้ใช้หรือแอพพลิเคชันต่างๆ คำสั่ง SQL สำหรับสร้าง schema ของฐานข้อมูลจะมีรูปแบบเป็น

CREATE SCHEMA AUTHORIZATION {creator};

ดังนั้น ถ้าผู้ที่ต้องการสร้างคือ JOE เราจะสามารถเขียนคำสั่งได้เป็น

CREATE SCHEMA AUTHORIZATION JOE;

โดยส่วนใหญ่ของ RDBMS จะสนับสนุนคำสั่ง SQL สำหรับสร้าง schema แต่คำสั่งนี้จะถูกใช้น้อยมาก เนื่องจาก RDBMS จะทำการสร้าง schema ให้กับผู้ใช้ RDBMS คนหนึ่งๆอัตโนมัติ

7.1.3 ชนิดของข้อมูลในฐานข้อมูล

ในการเพิ่ม/กำหนดแอทริบิวต่างๆในตารางข้อมูลหนึ่งๆ เราจะต้องทำการกำหนดชนิดของข้อมูลสำหรับแอทริบิวหนึ่งๆที่มีชนิดของข้อมูลดังนี้

- ***ข้อมูลเชิงตัวเลข***—จะประกอบไปด้วย 4 รูปแบบดังนี้
  - NUMBER(L,D)—จะเป็นรูปแบบข้อมูลตัวเลขที่สามารถกำหนดจำนวนหลักที่ใช้ในการจัดเก็บข้อมูลได้ เช่น NUMBER(7, 2) จะเป็นตัวเลขที่จัดเก็บทศนิยม 2 ตำแหน่งและสามารถขยายได้ถึง 7 หลัก เช่น 12.32, -134.99 เป็นต้น
  - INTEGER หรือ INT—ใช้สำหรับจัดเก็บข้อมูลตัวเลขจำนวนเต็ม
  - SMALLINT—จะใช้จัดเก็บข้อมูลตัวเลขจำนวนเต็มเหมือนกัน INTEGER แต่จะมีข้อแตกต่างตรงที่ SMALLINT จะสามารถจัดเก็บข้อมูลสูงสุดได้เพียง 6 หลักเท่านั้น ด้วยเหตุนี้ SMALLINT จึงเหมาะสำหรับการจัดเก็บข้อมูลตัวเลขที่มีค่าไม่มาก
  - DECIMAL(L,D)—จะมีลักษณะคล้ายกับ NUMBER แต่ความยาวของข้อมูลที่กำหนดจะเป็นความยาวขั้นต่ำ การประยุกต์ใช้ DECIMAL จะสามารถดำเนินการได้หลายรูปแบบ อาทิเช่น DECIMAL(9,2) DECIMAL(9) และ DECIMAL ตามลำดับ
- **ข้อมูลเชิงข้อความหรือตัวอักษร**—จะประกอบไปด้วย 2 รูปแบบดังนี้
  - CHAR(L)—จะใช้ในการจัดเก็บข้อมูลข้อความที่ซึ่งจะต้องกำหนดความยาวสูงสุดที่ต้องการจัดเก็บ โดยในการกำหนดความยาวจะสามารถกำหนดให้ข้อมูลมีความยาวได้ถึง 255 ตัวอักษร แต่ในส่วนของการจัดเก็บข้อมูลจะใช้พื้นที่ในการจัดเก็บตามที่กำหนดไว้เท่านั้น ไม่ว่าข้อมูลที่ต้องการจัดเก็บจะมีจำนวนตัวอักษรน้อยกว่าจำนวนที่กำหนดไว้หรือไม่ ตัวอย่างเช่น กำหนดให้แอทริบิวหนึ่งๆในตารางข้อมูลมีรูปแบบข้อมูลเป็น CHAR(25) จากนั้นทำการจัดเก็บข้อความ “Marketing” ที่ซึ่งจะประกอบไปด้วย 9 ตัวอักษร แต่เมื่อเราทำการ

กำหนดให้แอทริบิวจัดเก็บ 25 ตัวอักษร ดังนั้น จะต้องจองพื้นที่สำหรับจัดเก็บข้อมูลเป็น 25 ตัวอักษรแต่จะเก็บข้อมูลจริงเพียง 9 ตัวอักษรเท่านั้น ที่เหลืออีก 16 ที่จะเป็นที่ว่าง

- VARCHAR(L) หรือ VARCHAR2(L)—จะเป็นรูปแบบการจัดเก็บข้อมูลตัวอักษรที่มีความยืดหยุ่นตามความยาวของข้อมูลที่ต้องการจัดเก็บ โดยในขั้นตอนเริ่มต้นเราจะต้องทำการกำหนดความยาวสูงสุดของข้อมูลที่เราต้องการจัดเก็บ จากนั้นเมื่อทำการจัดเก็บข้อมูลจะทำการจัดเก็บข้อมูลตามความยาว/จำนวนตัวอักษรของข้อมูล ตัวอย่างเช่น กำหนดให้แอทริบิวหนึ่งๆในตารางข้อมูลมีรูปแบบข้อมูลเป็น VARCHAR(25) จากนั้นทำการจัดเก็บข้อความ “Marketing” ที่ซึ่งจะประกอบไปด้วย 9 ตัวอักษร โดยในการจัดเก็บ DBMS จะทำการจองพื้นที่สำหรับจัดเก็บข้อมูลเพียง 9 ตัวอักษรตามความยาวของข้อมูลเท่านั้นที่ซึ่งจะทำให้ประหยัดพื้นที่จากการใช้ VARCHAR ได้

- **ข้อมูลวันที่**—จะถูกกำหนดด้วยการใช้รูปแบบ DATE ที่ซึ่งจะจัดเก็บข้อมูลในรูปแบบ Julian date ที่ซึ่งจะมีการจัดเก็บข้อมูลเป็น YYDDD ตัวอย่างเช่น วันที่ 1 กุมภาพันธ์ 2015 จะถูกจัดเก็บในรูปแบบ 15032 ที่ซึ่ง 2 หลักแรกจะหมายถึงสองหลักท้ายของปี ส่วน 3 หลักสุดท้ายจะหมายถึงลำดับวันที่ในปี หรืออีกตัวอย่างหนึ่ง 31 ธันวาคม 2014 จะถูกจัดเก็บข้อมูลเป็น 14365 ตามลำดับ

นอกเหนือจากชนิดของข้อมูลข้างต้นแล้ว SQL ยังมีชนิดข้อมูลอีกเป็นจำนวนมาก อาทิเช่น TIME, TIMESTAMP, REAL, DOUBLE, FLOAT และ INTERVAL ด้วย แต่ในบทนี้เราจะทำการพิจารณาเฉพาะชนิดของข้อมูลที่อธิบายข้างต้นที่ซึ่งมักถูกประยุกต์ใช้ในการกำหนดรูปแบบของข้อมูลให้กับแอทริบิวหนึ่งๆในฐานข้อมูล ในการที่จะเข้าใจเกี่ยวกับชนิดของข้อมูลที่มักถูกประยุกต์ใช้ในการจัดเก็บข้อมูลในฐานข้อมูลมากขึ้น ลองพิจารณาพจนานุกรมข้อมูลในรูป 7.5 ที่ซึ่งจะสามารถชี้ให้เห็นถึงตัวอย่างการกำหนดรูปแบบของข้อมูลได้ดังนี้

- แอทริบิว P_PRICE ในตาราง PRODCUT จะแสดงถึงจัดเก็บข้อมูลราคาสินค้าที่ซึ่งจะถูกกำหนดให้มีรูปแบบข้อมูลเป็นข้อมูลเชิงตัวเลขด้วยการประยุกต์ใช้รูปแบบ NUMBER(8,2) ที่ซึ่งจะประกอบไปด้วยทศนิยม 2 ตำแหน่ง
- แอทริบิว V_NAME ในตาราง VENDOR จะแสดงถึงการจัดเก็บข้อมูลชื่อบริษัทผู้ผลิตสินค้าที่ซึ่งจะจัดเก็บในรูปแบบ VARCHAR(35) ที่ซึ่งจะทำให้ชื่อบริษัทต่างๆที่ถูกจัดเก็บจะมีจำนวนตัวอักษรได้ไม่เกิน 35 ตัวอักษร โดยแต่ละบริษัทอาจมีความยาวของชื่อบริษัทไม่เท่ากัน ด้วยเหตุนี้จึงกำหนดให้อยู่ในรูปแบบ VARCHAR ที่ซึ่งจะจัดเก็บข้อมูลตามความยาว/จำนวนตัวอักษรที่ปรากฏในชื่อบริษัทหนึ่งๆ
- แอทริบิว V_STATE ในตารางข้อมูล VENDOR จะแสดงถึงการจัดเก็บข้อมูลที่ตั้งของบริษัทที่ซึ่งจะต้องอยู่ในรัฐหนึ่งๆโดยจัดเก็บเป็นตัวอักษรย่อของรัฐนั้นๆ เช่น รัฐนิวยอร์กจะถูกจัดเก็บเป็น NY หรือรัฐแคลิฟอร์เนียจะถูกจัดเก็บเป็น CA เป็นต้น ดังนั้นเราสามารถกำหนดให้แอทริบิว V_STATE มีรูปแบบข้อมูลเป็น CHAR(2) ที่ซึ่งจะต้องมี 2 ตัวอักษร และทุกรัฐจะต้องมี 2 ตัวอักษรทั้งสิ้น

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE*	REQUIRED	PK OR FK	FK REFERENCED TABLE
PRODUCT	P_CODE	Product code	CHAR(10)	XXXXXXXXX	NA	Y	PK	
	P_DESCRIPT	Product description	VARCHAR(35)	Xxxxxxxxxxx	NA	Y		
	P_INDATE	Stocking date	DATE	DD-MON-YYYY	NA	Y		
	P_QOH	Units available	SMALLINT	####	0-9999	Y		
	P_MIN	Minimum units	SMALLINT	####	0-9999	Y		
	P_PRICE	Product price	NUMBER(8,2)	####.##	0.00-9999.00	Y		
	P_DISCOUNT	Discount rate	NUMBER(5,2)	0.##	0.00-0.20	Y		
	V_CODE	Vendor code	INTEGER	###	100-999		FK	VENDOR
VENDOR	V_CODE	Vendor code	INTEGER	#####	1000-9999	Y	PK	
	V_NAME	Vendor name	CHAR(35)	Xxxxxxxxxxxxx	NA	Y		
	V_CONTACT	Contact person	CHAR(25)	Xxxxxxxxxxxx	NA	Y		
	V_AREACODE	Area code	CHAR(3)	999	NA	Y		
	V_PHONE	Phone number	CHAR(8)	999-9999	NA	Y		
	V_STATE	State	CHAR(2)	XX	NA	Y		
	V_ORDER	Previous order	CHAR(1)	X	Y or N	Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed character length data, 1 to 255 characters
VARCHAR = Variable character length data, 1 to 2,000 characters. VARCHAR is automatically converted to VARCHAR2 in Oracle.
NUMBER = Numeric data. NUMBER(9,2) is used to specify numbers with two decimal places and up to nine digits long, including the decimal places. Some RDBMSs permit the use of a MONEY or a CURRENCY data type.
INT = Integer values only
SMALLINT = Small integer values only
DATE formats vary. Commonly accepted formats are: 'DD-MON-YYYY', 'DD-MON-YY', 'MM/DD/YYYY', and 'MM/DD/YY'
* Not all the ranges shown here will be illustrated in this chapter. However, you can use these constraints to practice writing your own constraints.

รูปที่ 7.5 ตัวอย่างพจนานุกรมข้อมูลที่แสดงถึงการกำหนดชนิดของข้อมูล

- แอทริบิว P_INDATE ในตารางข้อมูล PRODUCT จะแสดงถึงข้อมูลวันที่บริษัทได้รับสินค้าและทำการจัดเก็บสินค้าในโกดังสินค้า โดยข้อมูลนี้จะถูกกำหนดให้มีรูปแบบเป็น DATE ที่ซึ่งจะถูกจัดเก็บในรูปแบบของ Julian date เพื่อที่จะสามารถดำเนินคำนวณต่างๆกับวันที่ได้ เช่น เรามีการจัดเก็บสินค้ามาแล้วกี่วัน เป็นต้น

การกำหนดรูปแบบของข้อมูลในแอทริบิวหนึ่งๆจะมาจากกฎเกณฑ์ทางธุรกิจและมุมมองของผู้ออกแบบฐานข้อมูลที่ซึ่งสามารถมองข้อมูลได้หลายแบบ เช่น เมื่อเราพิจารณารูปแบบของข้อมูลแอทริบิว V_CODE ที่จะแสดงถึงรหัสบริษัทผู้ผลิตสินค้าเราจะสามารถกำหนดรูปแบบให้มีรูปแบบต่างๆได้ดังนี้

- ถ้าเราต้องการให้คอมพิวเตอร์ทำการสร้างรหัสบริษัทผู้ผลิตสินค้าหนึ่งๆด้วยการใช้รหัสตัวเลขที่ซึ่งจะสามารถกำหนดได้จากการบวก 1 กับรหัสบริษัทผู้ผลิตสินค้าที่ถูกจัดเก็บล่าสุด ดังนั้นเราจะต้องทำการกำหนดให้รูปแบบของแอทริบิว V_CODE มีรูปแบบในเชิงตัวเลขโดยอาจประยุกต์ใช้รูปแบบ INTEGER หรือ SMALLINT ขึ้นอยู่กับจำนวนบริษัทผู้ผลิตสินค้าที่ต้องการจัดเก็บ
- ถ้าเราต้องการที่จะทำการเพิ่มรหัสผู้ผลิตสินค้าด้วยการกำหนดเอง (ไม่ใช้การคำนวณด้วยการบวก 1 กับรหัสบริษัทผู้ผลิตสินค้าที่ถูกจัดเก็บล่าสุด)

ในการกำหนดรูปแบบของข้อมูลหนึ่งๆ เราจะต้องพิจารณาถึงการดำเนินการกับข้อมูลนั้นๆด้วย ไม่ว่าจะเป็นการเรียงลำดับข้อมูลหรือการค้นหาข้อมูล ตัวอย่างเช่น ในธุรกิจอสังหาริมทรัพย์จะมีการจัดเก็บข้อมูลเกี่ยวกับบ้านที่ซึ่งจะมีการจัดเก็บข้อมูลจำนวนห้องน้ำในบ้านหลังหนึ่งๆที่ซึ่งเราสามารถประยุกต์ใช้รูปแบบข้อมูลเป็น CHAR(3) เนื่องจากเราไม่ต้องการที่จะทำการคำนวณต่างๆไม่ว่าจะเป็นการบวก ลบ คูณ หรือหารกับจำนวนห้องน้ำในบ้านหลังหนึ่งๆ แต่อย่างไรก็ตามการจัดเก็บข้อมูลด้วย CHAR(3) อาจก่อให้เกิดปัญหาคือ 1) อาจมีการจัดเก็บข้อมูลห้องน้ำเป็น ‘2.5’ ที่ซึ่งสามารถจัดเก็บข้อมูลในแอทริบิวได้ เนื่องจากข้อมูลมีรูปแบบตรงกับรูปแบบที่กำหนดไว้ แต่ในความเป็นจริงจะไม่มีบ้านหลังใดเลยที่มีห้องน้ำสองห้องครึ่ง หรือ 2) ถ้าเราต้องการที่จะเรียงลำดับข้อมูลบ้าน โดยเรียงลำดับจากจำนวนห้องน้ำที่มีในบ้านหลังหนึ่งๆเราจะทำให้ผลลัพธ์ที่ได้คือ บ้านที่มีห้องน้ำเท่ากับ ‘10’ จะมีค่าน้อยกว่า ‘2’ ซึ่งมันไม่ถูกต้อง ด้วยเหตุนี้ เราจึงต้องทำการพิจารณาการกำหนดรูปแบบข้อมูลอย่างละเอียดถี่ถ้วนด้วยการพิจารณาถึงการดำเนินการต่างๆกับข้อมูลเหล่านั้นด้วย

7.1.4 การกำหนดโครงสร้างของตารางข้อมูล

ณ ตอนนี้เรามีความพร้อมที่จะทำการกำหนดโครงสร้างตารางข้อมูล PRODUCT และ VENDOR ที่ปรากฏในรูป 7.3 ด้วยการประยุกต์ใช้คำสั่ง CREATE TABLE ที่ซึ่งจะมีรูปแบบคำสั่งดังนี้

```
CREATE TABLE tablename (
        column1         data type     [constraint] ,
        column2         data type     [constraint] ,
        ...
```

```
...
...
PRIMARY KEY        (column1, ...),
FOREIGN KEY        (column1, ...) REFERENCFES tablename,
CONSTRAINT         constraint );
```

เพื่อที่จะทำให้คำสั่ง SQL สามารถอ่านได้ง่าย เราจะทำการกำหนด 1 แอทริบิวต่อ 1 บรรทัดที่ซึ่งจะทำให้เราสามารถอ่านคำสั่งได้โดยง่าย นอกจากนั้นเราควรที่จะต้องไม่ลืมว่า**ชื่อตารางและชื่อแอทริบิวควรที่จะต้องใช้ตัวอักษรพิมพ์ใหญ่** ด้วยเหตุนี้ เราจะสามารถสร้างโครงสร้างการจัดเก็บข้อมูลของตาราง VENDOR และ PRODUCT ได้เป็น

```
CREATE TABLE VENDOR (
        V_CODE          INTEGER         NOT NULL        UNIQUE,
        V_NAME          VARCHAR(35)     NOT NULL,
        V_CONTACT       VARCHAR(15)     NOT NULL,
        V_AREACODE      CHAR(3)         NOT NULL,
        V_PHONE         CHAR(8)         NOT NULL,
        V_STATE         CHAR(2)         NOT NULL,
        V_ORDER         CHAR(1)         NOT NULL
        PRIMARY KEY     (V_CODE));

CREATE TABLE PRODUCT (
        P_CODE          VARCHAR(10)     NOT NULL        UNIQUE,
        P_DESCRIPT      VARCHAR(35)     NOT NULL,
        P_INDATE        DATE            NOT NULL,
        P_QOH           SMALLINT        NOT NULL,
        P_MIN           SMALLINT        NOT NULL,
        P_PRICE         NUMBER(8,2)     NOT NULL,
        P_DISCOUNT      NUMBER(5,2)     NOT NULL,
        V_CODE          INTEGER,
        PRIMARY KEY     (P_CODE),
        FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```

จากการสร้างโครงสร้างตารางข้อมูลด้วยคำสั่ง SQL ข้างต้น เราจะสังเกตุรายละเอียดต่างๆได้ดังนี้

- การกำหนดเงื่อนไขในแอทริบิวต่างๆให้เป็น NOT NULL จะเป็นการทำให้แน่ใจได้ว่าการเพิ่มข้อมูลครั้งหนึ่งๆจะต้องประกอบไปด้วยค่าของข้อมูลในแอทริบิวที่มีการกำหนดเงื่อนไขให้เป็น NOT NULL การกำหนดเงื่อนไข NOT NULL ให้กับแอทริบิวจะไม่ยอมให้แอทริบิวเหล่านั้นไม่มีการปรากฏของข้อมูล
- การกำหนดเงื่อนไข UNIQUE ให้กับแอทริบิวจะเป็นการกำหนดความเป็นเอกลักษณ์ของข้อมูลให้กับแอทริบิวที่จะทำให้ค่าของข้อมูลที่ปรากฏในแอทริบิวไม่สามารถมีค่าซ้ำกันได้ (โดยส่วนมากมักทำการกำหนดเงื่อนไข UNIQUE ให้กับแอทริบิวที่ทำหน้าที่เป็น primary key)
- แอทริบิวที่ทำหน้าที่เป็น primary key จะถูกกำหนดให้มีเงื่อนไขเป็น NOT NULL และ UNIQUE ที่ซึ่งจะเป็นทำให้ตารางข้อมูลมีคุณสมบัติ entity integrity
- ข้อความ “ON UPDATE CASCADE” ในคำสั่งที่สำหรับการกำหนดให้แอทริบิวหนึ่งๆหรือกลุ่มของแอทริบิวทำหน้าที่เป็น foreign key ถูกอัพเดทอย่างถูกต้องเหมาะสม โดยจากตัวอย่างจะเป็นการกำหนดเงื่อนไขที่ซึ่งถ้าเราทำการอัพเดทข้อมูลในแอทริบิว V_CODE ในตาราง VENDOR จะทำให้ข้อมูล V_CODE ในตาราง PRODUCT ถูกอัพเดทตามไปด้วย ซึ่งการกำหนดข้อความ “ON UPDATE CASCADE” จะทำให้ตารางข้อมูลที่เชื่อมโยงความสัมพันธ์กันมีคุณสมบัติ referential integrity
- RDBMS จะทำการบังคับให้ตารางข้อมูลจะต้องมีคุณสมบัติ referential integrity แบบอัตโนมัติ ที่ซึ่งเราไม่สามารถลบข้อมูล V_CODE หนึ่งๆในตาราง VENDOR ตราบใดที่ยังมีข้อมูล V_CODE นั้นๆ ปรากฏในตาราง PRODUCT

7.1.5 การกำหนดเงื่อนไข

ในเนื้อหาบทที่ 3 เราได้เรียนรู้เกี่ยวกับการสร้างแบบจำลองข้อมูลเชิงสัมพันธ์ที่ซึ่งจะกำหนดให้ตารางข้อมูลจะต้องมี entity integrity และ referential integrity ดังนั้น ในการสร้างฐานข้อมูลจริง เราก็ควรจะดำเนินการให้ตารางข้อมูลต่างๆมี entity และ referential integrity ด้วยเช่นกัน ในการทำให้ตารางข้อมูลหนึ่งๆมีคุณสมบัติ entity integrity จะสามารถทำได้ในขั้นตอนการสร้างข้อมูลด้วยการประยุกต์ใช้คำสั่ง CREATE TABLE ที่ซึ่งจะบังคับให้ทำการกำหนดแอทริบิวหนึ่งๆหรือกลุ่มของแอทริบิวทำหน้าที่เป็น primary key และเรายังสามารถกำหนดให้แอทริบิวเหล่านั้นไม่มีค่า NULL ปรากฏขึ้นได้โดยการใช้เงื่อนไข NOT NULL แต่สำหรับในส่วนของการพิจารณาถึง referential integrity ของตารางข้อมูลต่างๆที่มีความสัมพันธ์กันจะสามารถทำได้โดยการกำหนดแอทริบิวที่ทำหน้าที่เป็น foreign key ที่ซึ่งจะสามารถทำได้ด้วยการประยุกต์ใช้คำสั่ง FOREIGN KET (attribute-name) REFERENCES tablename โดยจากตัวอย่างของการกำหนด foreign key ในตาราง PRODUCT เราจะสังเกตุได้ว่ามีการประยุกต์ใช้เงื่อนไข ON UPDATE CASCADE ที่ซึ่งจะเป็นการบังคับว่าเมื่อข้อมูล primary key ในตาราง VENDOR ที่ซึ่งทำหน้าที่เป็น foreign key ในตาราง PRODUCT มีการอัพเดท RDBMS จะต้องทำการอัพเดทค่าที่ปรากฏในแอทริบิวที่เป็น foregin key ในตาราง PRODUCT ให้โดยอัตโนมัติ นอกเหนือจากการประยุกต์ใช้เงื่อนไข ON UPDATE CASCADE

แล้ว เรายังสามารถทำการประยุกต์ใช้ ON DELETE ร่วมกัน CASCADE และยังรวมถึงการประยุกต์ใช้คำสั่ง SET NULL หรือ SET DEFAULT ด้วยเช่นกัน

นอกเหนือจากคำสั่ง PRIMARY KEY และ FOREIGN KEY ที่ใช้กำหนด primary key และ foreign key ในตารางข้อมูลแล้ว เรายังสามารถใช้คำสั่งอื่นๆเพื่อกำหนดเงื่อนไขต่างๆให้กับตารางข้อมูลได้ดังนี้

- คำสั่ง NOT NULL—จะเป็นคำสั่งที่ทำให้แน่ใจได้ว่าแอทริบิวหนึ่งๆจะไม่มีค่า NULL ปรากฏเลย
- คำสั่ง UNIQUE—จะเป็นคำสั่งที่ทำให้แน่ใจได้ว่าค่าของข้อมูลที่ปรากฏในแอทริบิวหนึ่งๆจะมีความเป็นเอกลักษณ์ (ไม่มีค่าซ้ำกัน)
- คำสั่ง DEFAULT—จะเป็นคำสั่งที่ใช้ในการกำหนดค่าโดยปริยาย ที่ซึ่ง RDBMS จะดำเนินการกำหนดค่าปริยายในแอทริบิวหนึ่งๆ เมื่อผู้ใช้ฐานข้อมูลไม่ทำการกรอกข้อมูลให้กับแอทริบิวนั้นๆ แต่ถ้าผู้ใช้ทำการกรอกข้อมูลในแอทริบิวนั้นๆ RDBMS ก็จะทำการจัดเก็บค่าของข้อมูลที่ผู้ใช้กรอก
- คำสั่ง CHECK—จะเป็นคำสั่งที่ใช้สำหรับตรวจสอบค่าของข้อมูลที่ปรากฏในแอทริบิวหนึ่งๆในขณะที่ข้อมูลถูกเพิ่มในตารางข้อมูล ตัวอย่างเช่น ค่าของข้อมูลที่ปรากฏในแอทริบิวจะต้องมีค่าไม่น้อยกว่า 10 หรือ ข้อมูลวันที่ที่จะถูกจัดเก็บในแอทริบิวหนึ่งๆจะต้องเป็นข้อมูลวันที่หลังจากวันที่ 31 December 2014 เป็นต้น

คำสั่งที่ใช้ในการกำหนดเงื่อนไขต่างๆให้กับตารางข้อมูลข้างต้นจะถูกประยุกต์ใช้ควบคู่กับคำสั่ง CREATE TABLE ที่ซึ่งเราจะสามารถทำการกำหนดเงื่อนไขได้ในสองลักษณะคือ 1) เมื่อเราทำการกำหนดแอทริบิวหนึ่งๆของตารางข้อมูลที่ซึ่งมีการกำหนดเงื่อนไขให้กับแอทริบิวนั้นๆต่อท้าย และ 2) เมื่อเราทำการประยุกต์ใช้คำสั่ง CONSTRAINT ที่ซึ่งจะเป็นการกำหนดเงื่อนไขให้กับตารางข้อมูลที่อาจส่งผลถึงแอทริบิวได้หลายแอทริบิว

ในการที่จะทำความเข้าใจเกี่ยวกับการกำหนดเงื่อนไขให้กับตารางมากขึ้น ลองพิจารณาตัวอย่างของการสร้างตาราง CUSTOMER, INVOICE และ LINE ที่ซึ่งจะสามารถสร้างได้ดังนี้

```
CREATE TABLE CUSTOMER (
        CUS_CODE        NUMBER          NOT NULL UNIQUE,
        CUS_LNAME       VARCHAR(15)     NOT NULL,
        CUS_FNAME       VARCHAR(15)     NOT NULL,
        CUS_INITIAL     VARCHAR(1),
        CUS_AREACODE    CHAR(3)         DEFAULT '615' NOT NULL
                                        CHECK(CUS_AREACODE IN('615', '713', '931')),
        CUS_PHONE       CHAR(8)         NOT NULL,
        CUS_BALANCE     NUMBER(9,2)     DEFAULT 0.00,
```

```
    PRIMARY KEY          (CUS_CODE),
    CONSTRAINT CUS_UI1 UNIQUE(CUS_LNAME, CUS_FNAME));
```

จากการสร้างตาราง CUSTOMER ข้างต้น เราจะสังเกตุได้ว่าจะมีการกำหนดค่าโดยปริยายให้กับแอทริบิว CUS_AREACODE ที่ซึ่งจะกำหนดให้มีค่า '615' เมื่อผู้ใช้ฐานข้อมูลไม่ทำการกรอกข้อมูลเกี่ยวกับรหัสของพื้นที่ที่พวกเขาพำนักอาศัยอยู่ นอกจากนั้นยังทำการกำหนดเงื่อนไข CHECK ให้กับแอทริบิว CUS_AREACODE ที่ซึ่งจะใช้ในการตรวจสอบว่าค่าของข้อมูลที่ผู้ใช้ฐานข้อมูลกรอกในขั้นตอนการเพิ่มข้อมูลให้กับตารางข้อมูลมีค่าตรงกับรหัส '615', '713' หรือ '931' หรือไม่? ถ้าผู้ใช้ฐานข้อมูลกรอกข้อมูลไม่ตรงกับรหัสทั้งสามที่อยู่ในขอบเขตของการตรวจสอบ RDBMS จะไม่ยอมรับค่าของข้อมูลที่ผู้ใช้ฐานข้อมูลกรอกและทำการแจ้งเตือนไปยังผู้ใช้ฐานข้อมูล จากการประยุกต์ใช้เงื่อนไข DEFAULT และ CHECK ข้างต้น เราจะสังเกตุได้ว่าเงื่อนไข DEFAULT จะมีการดำเนินการก็ต่อเมื่อมีข้อมูลแถวหนึ่งๆถูกเพิ่มเข้ามาใหม่ และแถวข้อมูลนั้นไม่มีค่าของข้อมูลในแอทริบิวที่กำหนดเงื่อนไข DEFAULT ไว้ แต่ในส่วนของเงื่อนไข CHECK จะดำเนินการตรวจสอบค่าข้อมูลเมื่อมีข้อมูลแถวหนึ่งๆถูกเพิ่มเข้ามาใหม่หรือข้อมูลถูกเปลี่ยนแปลงแก้ไข ท้ายสุดของการสร้างตาราง CUSTOMER จะมีการประยุกต์ใช้คำสั่ง CONSTRAINT ที่ซึ่งจะเป็นการกำหนดเงื่อนไขให้กับตาราง CUSTOMER โดยในการประยุกต์ใช้คำสั่ง CONSTRAINT จะทำการตั้งชื่อเงื่อนไขเป็น CUS_UI1 โดยจะเป็นการกำหนดเงื่อนไขที่ว่าในตารางข้อมูลจะต้องไม่มีข้อมูลแถวใดเลยที่มีค่าของข้อมูลที่ปรากฏในแอทริบิว CUS_LNAME และ CUS_FNAME ซ้ำกัน (ไม่มีลูกค้าคนใดเลยที่มีชื่อและนามสกุลซ้ำกัน)

ในส่วนของการสร้างตาราง INVOICE ด้านล่างจะมีการประยุกต์ใช้คำสั่ง DEFAULT SYSDATE กับแอทริบิว INV_DATE ที่ซึ่งจะเป็นการกำหนดข้อมูลวันที่ให้เป็นวันที่ ณ วันปัจจุบันเมื่อผู้ใช้ฐานข้อมูลไม่ทำการกรอกข้อมูลแอทริบิวนั้นๆ นอกจากนั้นกำหนดเงื่อนไขให้กับตารางข้อมูลด้วยการประยุกต์ใช้คำสั่ง CONSTRAINT ที่ถูกตั้งชื่อเป็น INV_CK1 โดยเงื่อนไขดังกล่าวจะเกี่ยวเนื่องกับเงื่อนไข CHECK ที่จะทำการตรวจสอบข้อมูลที่ปรากฏในแอทริบิว INV_DATE ที่จะต้องเป็นข้อมูลวันที่หลังจากวันที่ 1 มกราคม 2014

```
CREATE TABLE INVOICE (
    INV_NUMBER          NUMBER        NOT NULL      UNIQUE,
    CUS_CODE            NUMBER        NOT NULL,
    INV_DATE            DATE          DEFAULT SYSDATE NOT NULL
    FOREIGN KEY (CUS_CODE) REFERENCES CUSTOMER),
    CONSTRAINT INV_CK1 CHECK(INV_DATE > TO_DATE ('01-JAN-2014','DD-MON-YYYY')));
```

นอกเหนือจากข้อสังเกตุข้างต้น เราจะสามารถสังเกตุเพิ่มเติมถึงเงื่อนไขอื่นๆได้ดังนี้

- แอทริบิว CUS_CODE จะทำหน้าที่เป็น foreign key ที่ซึ่งจะเป็นแอทริบิวที่ใช้เชื่อมโยงความสัมพันธ์กับตาราง CUSTOMER ภายใต้ชื่อแอทริบิวเดียวกัน

- SYSDATE จะเป็นฟังก์ชันพิเศษที่มักจะให้ค่าวันที่ปัจจุบันเสมอ ดังนั้น ถ้าผู้ใช้ไม่กรอกข้อมูลในแอทริบิว INV_DATE จะทำให้ RDBMS ทำการกำหนดวันที่ปัจจุบันให้กับแอทริบิว INV_DATE
- TO_DATE จะเป็นฟังก์ชันที่ต้องการสองพารามิเตอร์ คือ 1) ข้อมูลวันที่ที่ใช้เป็นเงื่อนไขในการเปรียบเทียบ และ 2) รูปแบบของข้อมูลวันที่ที่จะทำการเปรียบเทียบ

ท้ายสุดจะเป็นคำสั่งการสร้างตาราง LINE ดังแสดงด้านล่างที่ซึ่งจะทำการกำหนด primary key ให้มีลักษณะเป็น composite primary key ด้วยการกำหนดให้แอทริบิว INV_NUMBER และ LINE_NUMBER ทำหน้าที่เป็น primary key และทำการกำหนดเงื่อนไข UNIQUE ให้กับแอทริบิว INV_NUMBER และ P_CODE ที่ซึ่งจะทำให้รายการสินค้าหนึ่งๆจะสามารถปรากฏได้เพียงครั้งเดียวในใบแจ้งหนี้หนึ่งๆ นอกจากนั้นยังมีการกำหนดให้ ON DELETE CASCADE กับ foreign key ที่จะเป็นการกำหนดเงื่อนไขเกี่ยวกับการลบข้อมูลแถว x หนึ่งๆในตาราง INVOICE (strong entity) จะทำให้แถวข้อมูลต่างๆในตาราง LINE (weak entity) ที่เกี่ยวข้องกับแถวข้อมูล x ถูกลบโดยอัตโนมัติ

```
CREATE TABLE LINE (
        INV_NUMBER          NUMBER          NOT NULL,
        LINE_NUMBER         NUMBER(2,0)     NOT NULL,
        P_CODE              VARCHAR(10)     NOT NULL,
        LINE_UNITS          NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
        LINE_PRICE          NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
        PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
        FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCASE,
        FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),
        CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE));
```

7.1.6 การสร้างดัชนี

แนวความคิดเกี่ยวกับการสร้างดัชนีจะช่วยให้เราสามารถเพิ่มประสิทธิภาพในการค้นหาข้อมูลและยังสามารถช่วยหลีกเลี่ยงการปรากฏซ้ำของค่าของข้อมูลในแอทริบิวที่ทำการสร้างดัชนีได้ ด้วยเหตุนี้ ภาษา SQL จึงมีคำสั่งที่สนับสนุนการสร้างดัชนีที่ซึ่งจะสามารถดำเนินการได้ด้วยการประยุกต์ใช้คำสั่ง CREATE INDEX ที่ซึ่งจะมีรูปแบบดังนี้

```
CREATE [UNIQUE] INDEX indexname ON tablename (colum1, colum2, ...);
```

การสร้างดัชนีจะสามารถดำเนินการได้ในหลายตารางข้อมูล ตัวอย่างเช่น การสร้างดัชนีให้กับแอทริบิว P_INDATE ที่ถูกจัดเก็บในตาราง PRODUCT ที่ซึ่งจะกำหนดชื่อดัชนีเป็น P_INDATEX จะสามารถดำเนินการได้โดย

CREATE INDEX P_INDATEX ON PRODUCT (P_INDATE);

ภายใต้**การประยุกต์ใช้ RDBMS จะมีการสร้างดัชนีให้กับตารางข้อมูลโดยอัตโนมัติที่ซึ่งจะทำการสร้างดัชนีให้กับตารางข้อมูลหนึ่งๆโดยเลือกจากแอทริบิวที่ทำหน้าที่เป็น primary key** ด้วยเหตุนี้ ถ้าเราต้องการที่จะทำการสร้างดัชนีขึ้นเอง **ภาษา SQL จะไม่ยอมให้เราทำการสร้างดัชนีซ้ำของเดิมที่ RDBMS ได้ทำการสร้างไว้แล้ว** ตัวอย่างเช่น ถ้าเราเลือกแอทริบิว P_CODE ในการสร้างดัชนีที่ซึ่งแอทริบิว P_CODE ทำหน้าที่เป็น primary key ในตาราง PRODUCT โดยเราประยุกต์ใช้คำสั่ง

CREATE UNIQUE INDEX P_CODEX ON PRODUCT(P_CODE);

จะทำให้มีข้อความฟ้องถึงความผิดพลาด “duplicate value in index” ปรากฏขึ้น ด้วยเหตุนี้ เราจึงต้องพยายามที่จะหลีกเลี่ยงการสร้างดัชนีซ้ำ โดยเราอาจทำการสร้างดัชนีกับข้อมูลด้วยการสร้างจากกลุ่มของแอทริบิว (composite index) แทนที่การสร้างดัชนีจากแอทริบิวหนึ่งๆ (unique index) การสร้างดัชนีจากกลุ่มของแอทริบิวจะเป็นวิธีการที่ได้รับความนิยมที่ซึ่งจะช่วยหลีกเลี่ยงการซ้ำกันของข้อมูลได้ ตัวอย่างเช่น เมื่อเราพิจารณาข้อมูลในรูป 7.6 ที่จะเป็นข้อมูลเกี่ยวกับคะแนนของการทดสอบพนักงานกับแบบทดสอบต่างๆที่ซึ่งพนักงานคนหนึ่งๆจะสามารถทำการทดสอบได้หลายครั้ง แต่จะทำการทดสอบในวันหนึ่งๆได้เพียงครั้งเดียว โดยโครงสร้างตารางข้อมูลในรูป 7.6 จะมีการกำหนด primary key ที่มีลักษณะเป็น composite primary key จากแอทริบิว EMP_NUM ร่วมกับแอทริบิว TEST_NUM ที่ซึ่งจะทำให้เราสามารถได้รับข้อมูลที่มีความเป็นเอกลักษณ์ ด้วยเหตุนี้เราจึงควรที่จะทำการสร้างดัชนีด้วยการสร้างเป็น composite index จากแอทริบิว EMP_NUM, TEST_CODE และ TEST_DATE ด้วยการประยุกต์ใช้คำสั่ง

CREATE UNIQUE INDEX EMP_TESTDEX ON TEST (EMP_NUM, TEST_CODE, TEST_DATE);

โดยปกติของการสร้างดัชนีจะสร้างผลลัพธ์ในลักษณะของลิสต์ของข้อมูลที่มีการเรียงลำดับจากน้อยไปมาก แต่เราสามารถปรับเปลี่ยนลำดับของข้อมูลให้เป็นจากมากไปน้อยได้ด้วยการกำหนด DESC หลังชื่อแอทริบิวที่เรากำหนดให้เป็นดัชนี ตัวอย่างเช่น ถ้าเราต้องการที่จะสร้างรายงานที่ซึ่งจะเป็นลิสต์ของรายการสินค้าโดยเรียงลำดับตามข้อมูลราคาสินค้าจากมากไปน้อย เราจะสามารถสร้างดัชนีที่มีชื่อว่า PROD_PRICEX ได้ดังนี้

CREATE INDEX PROD_PRICEX ON PRODUCT(P_PRICE DESC);

เมื่อเราทำการสร้างดัชนีแล้ว ในเวลาต่อมาเราสามารถยกเลิกการสร้างดัชนีได้โดยประยุกต์ใช้คำสั่ง DROP INDEX ที่ซึ่งจะมีรูปแบบเป็น

DROP INDEX *indexname*;

ตัวอย่างเช่น ถ้าเราต้องการยกเลิกการสร้างดัชนีที่ชื่อ PROD_PRICEX ในตาราง PRODUCT จะสามารถดำเนินได้ดังนี้

DROP INDEX PROD_PRICEX;

หลังจากทำการสร้างตารางข้อมูลและดัชนีต่างๆในตารางข้อมูลแล้ว เราจะมีความพร้อมที่จะเพิ่มข้อมูลหรือจัดเก็บข้อมูลในตารางข้อมูลได้ด้วยการประยุกต์ใช้คำสั่งประเภท DML ที่ซึ่งจะใช้สำหรับจัดการในลักษณะต่างๆกับข้อมูล

7.2 คำสั่ง SQL ที่เกี่ยวกับ data manipulation

ในส่วนนี้เราจะทำการศึกษาเกี่ยวกับคำสั่งที่ใช้ในการจัดการข้อมูลภายใต้การประยุกต์ใช้คำสั่ง INSERT, SELECT, COMMIT, UPDATE, ROLLBACK และ DELETE ตามลำดับ

7.2.1 การเพิ่มแถวของข้อมูลในตารางข้อมูล

การเพิ่มข้อมูลในตารางข้อมูลจะสามารถดำเนินการเพิ่มข้อมูลได้ทีละแถวข้อมูลด้วยการประยุกต์ใช้คำสั่ง INSERT ที่ซึ่งจะมีรูปแบบดังนี้

INSERT INTO *tablename* VALUES (*value1*, *value2*, ..., *valuen*);

โดยในการเพิ่มข้อมูลให้กับตารางข้อมูลที่มีความสัมพันธ์กับตารางข้อมูลอื่นๆ เราจะต้องเพิ่มความระมัดระวังไม่ให้ฝ่าฝืนกฎ referential integrity ที่ซึ่งแถวข้อมูลหนึ่งๆในตารางข้อมูลหนึ่งๆจะทำการอ้างถึงข้อมูลที่ไม่มีตัวตนในอีกตารางหนึ่ง ตัวอย่างเช่น ตาราง PRODUCT และตาราง VENDOR ได้ทำการประยุกต์ใช้แอทริบิว V_CODE ทำหน้าเป็น foreign key เพื่อเชื่อมโยงความสัมพันธ์ ดังนั้นอาจเกิดเหตุการณ์ที่แถวของข้อมูลหนึ่งๆในตาราง PRODUCT ทำการอ้างถึงแอทริบิว V_CODE ที่ไม่มีตัวตนหรือไม่มีข้อมูลในตาราง VENDOR ได้ ด้วยเหตุนี้ เราจึงต้องทำการกำหนดลำดับของการเพิ่มข้อมูลโดยเริ่มจากการเพิ่มแถวข้อมูลในตาราง VENDOR เสียก่อน จากนั้นจึงทำการเพิ่มข้อมูลในตาราง PRODUCT ซึ่งจะสามารถแสดงได้ดังคำสั่ง:

INSERT INTO VENDOR VALUES (21225, ‘Bryson, Inc.’, ‘Smithson’, ‘615’, ‘223-3234’, ‘TN’, ‘Y’);

INSERT INTO VENDOR VALUES (21226, ‘Superloo, Inc.’, ‘Flushing’, ‘904’, ‘215-8995’, ‘FL’, N);

INSERT INTO PRODUCT VALUES (‘11QER/31’, ‘Power painter, 15 psi., 3-nozzle’, ’03-Nov-14’, 8, 5, 109.99, 0.00, 21225);

INSERT INTO PRODUCT VALUES (’13-Q2/P2’, ‘7.25-in. pwr. saw blade’, ’13-Dec-14’, 32, 15, 14.99, 0.05, 21226);

การเพิ่มข้อมูลแถวหนึ่งๆที่มีค่า NULL ปรากฏในแอทริบิวหนึ่งๆ

ในการเพิ่มข้อมูลแถวหนึ่งๆในตารางข้อมูลหนึ่งๆ เราอาจพบเจอสถานการณ์ที่ผู้ใช้ฐานข้อมูลต้องการที่จะทำการกรอกข้อมูลที่มีค่า NULL ให้กับแอทริบิวหนึ่งๆ ตัวอย่างเช่น ถ้าเรามีรายการสินค้าหนึ่งๆที่เกิดจาก

การผลิตขึ้นเองของบริษัทหรือเราอาจยังไม่สามารถทราบรหัสที่แน่ชัดของบริษัทผู้ผลิตสินค้า เราอาจทำการเพิ่มข้อมูลโดยปล่อยให้แอทริบิว V_CODE มีค่าเป็น NULL ที่ซึ่งจะสามารถดำเนินการด้วยการประยุกต์ใช้คำสั่ง:

```
INSERT INTO PRODUCT VALUES ('BRT-345', 'Titanium drill bit', '18-Oct-09', 75, 10, 4.50, 0.06,
        NULL);
```

จากคำสั่งข้างต้น การเพิ่มค่า NULL ให้กับแอทริบิว V_CODE จะสามารถยอมรับได้ เนื่องจากแอทริบิว V_CODE ในตาราง PRODUCT จะมีลักษณะเป็นแอทริบิวที่เป็นส่วนเสริม (optional attribute) เพราะในขั้นตอนการกำหนดตารางด้วยคำสั่ง CREATE TABLE ไม่มีการประยุกต์ใช้เงื่อนไข NOT NULL กับแอทริบิว V_CODE

การเพิ่มข้อมูลในแอทริบิวที่เป็นส่วนเสริม

ตารางข้อมูลหนึ่งๆอาจมีหลายแอทริบิวที่มีลักษณะเป็นแอทริบิวส่วนเสริม ดังนั้น เราจะสามารถเพิ่มค่าของข้อมูลที่เป็น NULL ให้กับแอทริบิวเหล่านั้นได้ แต่เราสามารถหลีกเลี่ยงการเพิ่มข้อมูลที่เป็น NULL ให้กับแอทริบิวที่เป็นส่วนเสริมได้ด้วยการเพิ่มค่าของข้อมูลในแอทริบิวที่สำคัญ (required attribute) เท่านั้น ด้วยการบอกกล่าวถึงชื่อแอทริบิวที่เป็นแอทริบิวที่สำคัญ ตัวอย่างเช่น ในตาราง PRODUCT จะมีแอทริบิว P_CIDE และ P_DESCRIPT ที่เป็นแอทริบิวที่สำคัญ ดังนั้นเราสามารถเพิ่มข้อมูลเฉพาะแอทริบิวที่สำคัญได้เป็น:

```
INSERT INTO PRODUCT (P_CODE, P_DESCRIPT) VALUES ('BRT-345', 'Titanium drill bit');
```

7.2.2 การจัดเก็บข้อมูลที่มีเปลี่ยนแปลงในตารางข้อมูล

การดำเนินการกับข้อมูลในตารางหนึ่งๆไม่ว่าจะเป็นการเพิ่ม ลบ หรืออัพเดทข้อมูลจะไม่ถูกจัดเก็บในดิสก์จนกระทั่งเราทำการปิดโปรแรมที่ใช้ในการจัดการฐานข้อมูลหรือเราประยุกต์ใช้คำสั่ง COMMIT แต่อย่างไรก็ตาม ถ้าฐานข้อมูลมีผู้ใช้หลายคนหรือฐานข้อมูลอาจมีปัญหาเกี่ยวกับไฟฟ้าดับก่อนการประยุกต์ใช้คำสั่ง COMMIT ที่จะทำให้เราไม่ได้ทำการเพิ่ม ลบ หรืออัพเดทข้อมูลในดิสก์ ด้วยเหตุนี้เราจึ่งจำเป็นที่จะต้องประยุกต์ใช้คำสั่ง COMMIT อย่างสม่ำเสมอเพื่อทำให้ข้อมูลที่ถูกเพิ่ม ลบ หรืออัพเดทถูกจัดเก็บในดิสก์ โดยคำสั่ง COMMIT จะมีรูปแบบเป็น

```
COMMIT [WORK];
```

หลังจากประยุกต์ใช้คำสั่ง COMMIT แล้วจะทำให้ความเปลี่ยนแปลงที่เกิดขึ้นกับทุกตารางข้อมูลถูกจัดเก็บในดิสก์ที่ซึ่งจะช่วยให้เราแน่ใจได้ว่าตารางข้อมูลต่างๆจะมีคุณสมบัติ update integrity

7.2.3 การเรียกดูแถวข้อมูลต่างๆ

การเรียกดูข้อมูลจากตารางข้อมูลจะสามารถดำเนินการได้ผ่านคำสั่ง SELECT ที่ซึ่งจะมีรูปแบบเป็น

SELECT *column1, column2, ...*

FROM *tablename;*

ในการเรียกดูข้อมูลเราจะสามารถเรียกดูข้อมูลในแอทริบิวต่างๆได้มากกว่าหนึ่งแอทริบิว แต่ถ้าเราต้องการเรียกดูข้อมูลทุกแอทริบิวในตารางข้อมูลหนึ่งๆ เราอาจทำการกำหนดชื่อทุกแอทริบิวในคำสั่ง SELECT หรือเราทำการประยุกต์ใช้**เครื่องหมาย "*" ที่เรียกว่า "wildcard charater" ที่ซึ่งจะบ่งบอกถึงอะไรก็ได้** ตัวอย่างเช่น ถ้าเราต้องการเรียกดูข้อมูลทุกแอทริบิวในตาราง PRODUCT เราจะสามารถดำเนินได้โดย:

SELECT * FROM PRODUCT;

จากคำสั่งข้างต้นเราจะได้ผลลัพธ์ดังแสดงในรูป 7.6 ที่ซึ่งจะประกอบไปด้วยทุกรายการสินค้า (ทุกแถวข้อมูล) ที่ถูกจัดเก็บในตาราง PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-10	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	18	5	119.95	0.10	25595

รูปที่ 7.6 ตัวอย่างการแสดงผลข้อมูลในตาราง PRODUCT ด้วยคำสั่ง SELECT

7.2.4 การอัพเดทข้อมูลแถวต่างๆในตารางข้อมูล

การอัพเดทข้อมูลในตารางหนึ่งๆสามารถดำเนินการโดยประยุกต์ใช้คำสั่ง UPDATE ที่จะมีรูปแบบเป็น

UPDATE *tablename*

SET *columnname = expression [, columnname = expression]*

[WHERE *conditionlist];*

ตัวอย่างเช่น ถ้าเราต้องการอัพเดทข้อมูลในแอทริบิว P_INDATE จากเดิมที่มีค่าเป็น 13 Dec 2009 ให้กลายเป็น 18 Jan 2010 ที่ซึ่งจากตารางข้อมูลในรูป 7.6 จะเป็นข้อมูลในแถวที่สองและมีค่าในแอทริบิว P_CODE เป็น '13-Q2/P2' (แอทริบิว P_CODE ทำหน้าที่เป็น primary key ในตาราง PRODUCT) ดังนั้นเราจะสามารถประยุกต์ใช้คำสั่ง UPDATE ในการอัพเดทข้อมูลได้เป็น

```
UPDATE      PRODUCT
SET         P_INDATE = '18-JAN-2010'
WHERE       P_CODE = '13-Q2/P2'
```

แต่ถ้าเราต้องการที่จะอัพเดทข้อมูลในหลายๆแอทริบิว เราจะสามารถดำเนินการได้ด้วยการประยุกต์ใช้ ',' ในขั้นตอน SET ได้ดังนี้

```
UPDATE      PRODUCT
SET         P_INDATE = '18-JAN-2010', P_PRICE = 17.99, P_MIN = 10
WHERE       P_CODE = '13-Q2/P2'
```

จากข้างต้นเราจะสังเกตุได้ว่าการกำหนดเงื่อนไขด้วยการประยุกต์ใช้ 'WHERE' จะเป็นสิ่งสำคัญ ถ้าเราไม่กำหนดเงื่อนไขจะทำให้ข้อมูลในแอทริบิว P_INDATE, P_PRICE และ P_MIN ของทุกแถวข้อมูลจะถูกอัพเดททั้งหมด ด้วยเหตุนี้ ในการดำเนินการอัพเดทข้อมูล เราไม่ควรที่จะหลงลืมในการกำหนดเงื่อนไขในการอัพเดทที่ซึ่งจะทำให้มีการอัพเดทเฉพาะเงื่อนไขเท่านั้น

7.2.5 การกู้คืนข้อมูลในตารางข้อมูล

ในการจัดเก็บข้อมูล ถ้าเรายังไม่ได้ทำการประยุกต์ใช้คำสั่ง COMMIT (จะเป็นคำสั่งสำหรับจัดเก็บข้อมูลลงในดิสก์) เราจะสามารถกู้คืนข้อมูลในตารางข้อมูลได้ด้วยการประยุกต์ใช้คำสั่ง ROLLBACK แต่ถ้าเราทำการประยุกต์ใช้คำสั่ง COMMIT ไปแล้ว เราจะไม่สามารถกู้คืนข้อมูลในตารางข้อมูลก่อนการประยุกต์ใช้คำสั่ง COMMIT ได้ โดยการประยุกต์ใช้คำสั่ง ROLLBACK จะได้สามารถดำเนินการได้เป็น

```
ROLLBACK;
```

โดยปกติของการประยุกต์ใช้คำสั่ง COMMIT และ ROLLBACK จะดำเนินการกับคำสั่งประเภท DML ที่ซึ่งจะเป็นการดำเนินการเพิ่ม เปลี่ยนแปลง หรือลบข้อมูลจากตารางข้อมูล

7.2.6 การลบข้อมูลออกจากตารางข้อมูล

การลบข้อมูลออกจากตารางหนึ่งๆสามารถดำเนินการโดยประยุกต์ใช้คำสั่ง DELETE ที่จะมีรูปแบบเป็น

```
DELETE FROM     tablename
[WHERE          conditionlist];
```

ตัวอย่างเช่น ถ้าเราต้องการลบข้อมูลออกจากตาราง PRODUCT ที่ซึ่งจะต้องการลบข้อมูลรายการสินค้าที่มีรหัสเป็น 'BRT-345' เราจะสามารถดำเนินการได้ดังนี้

```
DELETE          PRODUCT
WHERE           P_CODE = 'BRT-345';
```

จากข้างต้นจะเป็นการกำหนดเงื่อนไขกับแอทริบิวที่ทำหน้าที่เป็น primary key แต่อย่างไรก็ตาม เราสามารถกำหนดเงื่อนไขกับแอทริบิวอื่นๆได้ เช่น ทำการลบข้อมูลรายการสินค้าที่มีค่า P_MIN = 5 ออกจากตารางข้อมูล PRODUCT ที่ซึ่งจะสามารถดำเนินการได้โดย

```
DELETE          PRODUCT
WHERE           P_MIN = 5;
```

7.2.7 การเพิ่มข้อมูลในตารางข้อมูลด้วยการเรียกดูข้อมูลจากตารางข้อมูลหนึ่งๆ

จากเนื้อหาในส่วน 7.2.1 เราได้ศึกษาเกี่ยวกับการเพิ่มข้อมูลในตารางข้อมูลหนึ่งๆที่ซึ่งในการดำเนินการแต่ละครั้งจะเป็นการเพิ่มข้อมูลแถวหนึ่งๆในตารางหนึ่งๆเท่านั้น แต่อย่างไรก็ตามเราสามารถทำ**การเพิ่มหลายแถวข้อมูลให้กับตารางข้อมูลด้วยการอ่านข้อมูลจากตารางข้อมูลอื่น**โดยการประยุกต์ใช้คำสั่งดังต่อไปนี้

```
INSERT INTO tablename SELECT columnlist FROM tablename;
```

จากคำสั่งข้างต้นเราจะสังเกตุได้ว่าคำสั่ง INSERT จะเป็น outer query ที่ประยุกต์ใช้คำสั่ง SELECT เป็น subquery (หรืออาจเรียกว่า nested query หรือ inner query) เป็นอินพุต โดยในการคำนวณ RDBMS จะเริ่มจากการพิจารณา subquery ก่อน จากนั้นนำผลลัพธ์ที่ได้จาก subquery ไปเป็นอินพุตของ outer query (หมายเหตุ—การกำหนด query หนึ่งๆเราสามารถกำหนดให้มี query ย่อยๆได้หลายระดับ)

7.3 คิวรีในการเรียกดูข้อมูล

ในส่วนนี้เราจะทำการศึกษาเกี่ยวกับการเพิ่มเงื่อนไขในการเรียกดูข้อมูลที่ซึ่งจะเป็นการเพิ่มเงื่อนไขในคำสั่ง SELECT เพื่อทำการเรียกดูข้อมูลแบบมีเงื่อนไขอันนำมาซึ่งการได้รับข้อมูลสารสนเทศเพื่อตอบคำถามต่างๆ เช่น “รายการสินค้าใดที่มีราคาต่ำกว่า 10$” “รายการสินค้าใดที่ถูกผลิตระหว่างวันที่ 5 Jan 2010 ถึง 20 Mar 2010” เป็นต้น

7.3.1 การเรียกดูข้อมูลด้วยการกำหนดเงื่อนไข

การเรียกดูข้อมูลด้วยการกำหนดเงื่อนไขจะสามารถดำเนินการได้ดังนี้

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist];
```

คำสั่งข้างต้นจะดำเนินการค้นหาข้อมูลและคืนค่าผลลัพธ์ที่ตรงกับเงื่อนไขที่เรากำหนด การกำหนดเงื่อนไขสามารถทำการกำหนดได้หลายเงื่อนไข หรือไม่ทำการกำหนดเงื่อนไขเลยก็ได้ แต่ถ้าทำการกำหนดเงื่อนไขในการเรียกดูข้อมูลแล้วปรากฏว่าไม่มีแถวข้อมูลใดเลยที่มีค่าของของมูลตรงกับเงื่อนไขที่กำหนดจะทำให้เราไม่ได้ผลลัพธ์จากการเรียกดูข้อมูลนั้นๆ ตัวอย่างของการเรียกดูข้อมูลแบบมีเงื่อนไขจะสามารถแสดงได้เป็น

```
SELECT      P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       V_CODE = 21344;
```

คำสั่ง SELECT ในการเรียกดูข้างต้นจะทำการเรียกดูข้อมูล 4 แอทริบิว คือ P_DESCRIPT, P_INDATE, P_PRICE และ V_CODE จากตาราง PRODUCT โดยกำหนดเงื่อนไขเป็น V_CODE = 21344 ที่ซึ่งจะหมายความว่าจะทำการเรียกดูรายการสินค้าที่ประกอบไปด้วยข้อมูลคำอธิบายสินค้า วันที่ได้รับสินค้าจากผู้ผลิต ราคาสินค้า และรหัสบริษัทผู้ผลิตสินค้าที่ซึ่งรายการสินค้าที่ถูกแสดงเป็นผลลัพธ์จะต้องเป็นรายการสินค้าที่มาจากบริษัทผู้ผลิตสินค้าที่มีรหัส 21344 ดังแสดงในรูป 7.7

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-09	14.99	21344
9.00-in. pwr. saw blade	13-Nov-09	17.49	21344
Rat-tail file, 1/8-in. fine	15-Dec-09	4.99	21344

รูปที่ 7.7 ตัวอย่างผลลัพธ์ที่ได้จากการค้นหารายการสินค้าที่มาจากบริษัทผู้ผลิตสินค้าที่มีรหัส 21344

ในการกำหนดเงื่อนไข เราสามารถใช้การเปรียบเทียบในลักษณะต่างๆที่ซึ่งจะมีการใช้สัญลักษณ์ในรูปแบบต่างๆดังแสดงในรูป 7.8

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

รูปที่ 7.8 เครื่องหมายที่ใช้ในการเปรียบเทียบ

ตัวอย่างเช่น

```
SELECT      P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       V_CODE <> 21344;
```

จะเป็นคำสั่ง SQL ในการเรียกดูข้อมูลรายการสินค้าที่ประกอบไปด้วยข้อมูลคำอธิบายสินค้า วันที่ได้รับสินค้าจากผู้ผลิต ราคาสินค้า และรหัสบริษัทผู้ผลิตสินค้าที่ซึ่งรายการสินค้าที่จะถูกแสดงเป็นผลลัพธ์จะต้องเป็นรายการสินค้าที่มาจากบริษัทผู้ผลิตสินค้าที่ไม่มีรหัสเป็น 21344

การประยุกต์ใช้ตัวดำเนินการในการเปรียบเทียบข้อมูลเชิงข้อความ

การเปรียบเทียบข้อมูลในแอทริบิวหนึ่งๆที่มีรูปแบบเป็นตัวเลขจะทำให้เราสามารถเปรียบเทียบได้โดยตรง แต่อย่างไรก็ตามเราสามารถเปรียบเทียบข้อมูลในแอทริบิวที่มูปแบบเชิงข้อความได้ ดังแสดงในคำสั่ง

```
SELECT      P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_PRICE
FROM        PRODUCT
WHERE       P_CODE < '1558-QW1';
```

จะเป็นการเรียกดูข้อมูลรายการสินค้าที่มีรหัสสินค้าน้อยกว่า '1558-QW1' โดยจะต้องการที่จะได้รับข้อมูลทั้งสิ้น 5 แอทริบิว คือ P_CODE, P_DESCRIPT, P_QOH, P_MIN และ P_PRICE ตามลำดับ โดยในการเปรียบเทียบรหัสสินค้าตามเงื่อนไข WHERE จะทำการเปรียบเทียบจากหรัส ASCII ของข้อมูลที่ซึ่งตัวอักษร A จะมีรหัส ASCII น้อยกว่าตัวอักษร B และ C ตามลำดับ ดังนั้นการดำเนินการเปรียบเทียบจะดำเนินการจากตัวอักษรตัวแรกทางซ้ายมือไปจนกระทั่งตัวอักษรตัวสุดท้ายทางขวามือตามลำดับ ตัวอย่างเช่น ในการเปรียบเทียบระหว่างข้อความ "Ardmore" และ "Aarenson" เราจะทราบได้ว่า "Ardmore" จะมีค่ามากกว่า "Aarenson"

การประยุกต์ใช้ตัวดำเนินการในการเปรียบเทียบกับข้อมูลวันที่

ภายใต้การกำหนดเงื่อนไขในคำสั่ง SELECT เราสามารถค้นหาข้อมูลที่ต้องการด้วยการเปรียบเทียบข้อมูลในแอทริบิวที่มีรูปแบบเป็นวันที่ ตัวอย่างเช่น เราต้องการรายละเอียดสินค้าที่ถูกจัดเก็บในโกดังสินค้าก่อนวันที 20 มกราคม 2010 เราจะสามารถประยุกต์ใช้การเปรียบเทียบกับข้อมูลวันที่ได้เป็น

```
SELECT      P_DESCRIPT, P_QOH, P_MIN, P_PRICE, P_INDATE
FROM        PRODUCT
WHERE       P_INDATE >= '20-Jan-2010';
```

การสร้างการคำนวณระหว่างแอทริบิวต่างๆเพื่อแสดงผลลัพธ์

ภายใต้ประยุกต์ใช้คำสั่ง SELECT ภาษา SQL จะยอมให้เราสร้างการคำนวณระหว่างแอทริบิวต่างๆเพื่อแสดงผลลัพธ์เป็นแอทริบิวหนึ่งๆ โดยในการคำนวณเราอาจประยุกต์ใช้ตัวดำเนินการพื้นฐาน เช่น +, -, * หรือ ÷ ตามลำดับ ตัวอย่างเช่น

```
SELECT      P_CODE, P_DESCRIPT, P_QOH, P_QOH * P_PRICE
FROM        PRODUCT
```

นอกเหนือจากคำสั่งข้างต้น เราสามารถทำการกำหนดชื่อให้กับแอทริบิวที่เป็นผลลัพธ์จากการคำนวณได้ ตัวอย่างเช่น

```
SELECT    P_CODE, P_DESCRIPT, P_QOH, P_QOH * P_PRICE AS TOTVALUE
FROM      PRODUCT
```

หรือ

```
SELECT    P_CODE, P_INDATE, P_INDATE + 90 AS EXPDATE
FROM      PRODUCT
```

7.3.2 การประยุกต์ใช้การดำเนินการคณิตศาสตร์

ในคิวรีหนึ่งๆเอาอาจะประยุกต์ใช้ตัวดำเนินการคณิตศาสตร์เพื่อช่วยในการค้นผลลัพธ์ที่จำเป็นต้องทำการคำนวณที่ซึ่งมักจะประยุกต์ใช้ตัวดำเนินการดังแสดงในรูป 7.9

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

รูปที่ 7.9 ตัวดำเนินการทางคณิตศาสตร์พื้นฐานที่สามารถประยุกต์ใช้กับคำสั่ง SELECT

7.3.3 การประยุกต์ใช้ตัวดำเนินการทางตรรกะ

ในการกำหนดเงื่อนไขในการเรียกดูข้อมูล เราอาจกำหนดเงื่อนไขได้มากกว่าหนึ่งเงื่อนไขที่ซึ่งเราจะสามารถประยุกต์ใช้ตัวดำเนินการทางตรรกะต่างๆเช่น AND, OR, หรือ NOT ตัวอย่างเช่น ถ้าเราต้องการที่จะเรียกดูข้อมูลรายการสินค้าที่ถูกผลิตจากบริษัทผู้ผลิตสินค้าที่มีรหัส 21344 หรือ 24288 เราสามารถประยุกต์ใช้ตัวดำเนินการ OR ดังแสดงในคำสั่ง SELECT ดังนี้

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     V_CODE = 21344 OR V_CODE = 24288;
```

หรือเราอาจต้องการที่จะค้นหารายการสินค้าที่มีราคาน้อยกว่า 50$ ที่ซึ่งเป็นถูกจัดเก็บเข้าโกดังหลังจากวันที่ 15 Jan 2010 ที่ซึ่งเราจะต้องทำการประยุกต์ใช้ตัวดำเนินการ AND ดังแสดงในคำสั่ง SELECT ดังนี้

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     P_PRICE < 50 AND P_INDATE > '15-Jan-2010';
```

หรือเราอาจต้องการที่จะค้นหารายการสินค้าที่มีราคาน้อยกว่า 50$ ที่ซึ่งถูกจัดเก็บเข้าโกดังหลังจากวันที่ 15 Jan 2010 หรือมีบริษัทผู้ผลิตสินค้าเป็น 24288 เราจะสามารถดำเนินการได้โดยใช้ตัวดำเนินการ AND และ OR ดังแสดงในคำสั่ง SELECT ดังนี้

```
SELECT      P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       (P_PRICE < 50 AND P_INDATE > '15-Jan-2010') OR V_CODE = 24288;
```

หรือเราอาจต้องการที่จะทำการค้นหารายการสินค้าที่ไม่ได้ผลิตจากบริษัทผู้ผลิตสินค้าที่มีรหัส 21344 ที่ซึ่งเราจะสามารถดำเนินการได้โดยใช้ตัวดำเนินการ NOT ดังแสดงในคำสั่ง SELECT ดังนี้

```
SELECT      *
FROM        PRODUCT
WHERE       NOT (V_CODE = 21344);
```

7.3.4 การประยุกต์ใช้ตัวดำเนินการพิเศษ

ตัวดำเนินการพิเศษที่สามารถประยุกต์ใช้ในการกำหนดเงื่อนไขจะประกอบไปด้วย

BETWEEN—จะใช้ในการตรวจสอบค่าของข้อมูลในแอทริบิวที่จะมีค่าในช่วงที่ต้องการตรวจสอบ

IS NULL—จะใช้ในการตรวจสอบค่าของข้อมูลในแอทริบิวว่ามีค่าเป็น NULL หรือไม่

LIKE—จะใช้ในการตรวจสอบค่าของข้อมูลในแอทริบิวที่มีค่าตรงกับรูปแบบที่กำหนด

IN—จะใช้ในการตรวจสอบค่าของข้อมูลในแอทริบิวที่มีค่าในลิสต์ของค่าของข้อมูลที่กำหนด

EXISTS—จะใช้ในการตรวจสอบว่า subquery จะคืนค่าผลลัพธ์เป็นแถวของข้อมูลหรือไม่

การประยุกต์ใช้ตัวดำเนินการ BETWEEN

ตัวดำเนินการ BETWEEN มักถูกประยุกต์ใช้เพื่อตรวจสอบว่าค่าข้อมูลในแอทริบิวหนึ่งๆมีค่าอยู่ในช่วงที่กำหนด ตัวอย่างเช่น เราต้องการเรียกดูข้อมูลรายการสินค้าที่มีราคาอยู่ในช่วงระหว่าง 50$ ถึง 100$ ที่ซึ่งจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      *
FROM        PRODUCT
WHERE       P_PRICE BETWEEN 50.00 AND 100.00;
```

แต่ถ้าเราไม่ทำการประยุกต์ใช้ตัวดำเนินการ BETWEEN เราก็ยังสามารถตรวจสอบค่าข้อมูลในแอทริบิวหนึ่งๆมีค่าอยู่ในช่วงที่กำหนดได้โดย

```
SELECT      *
FROM        PRODUCT
WHERE       P_PRICE > 50.00 AND P_PRICE < 100.00;
```

การประยุกต์ใช้ตัวดำเนินการ IS NULL

ตัวดำเนินนการ IS NULL จะใช้ในการตรวจสอบว่าค่าข้อมูลที่ปรากฏในแอทริบิวหนึ่งๆมีค่าเป็น NULL ตัวอย่างเช่น ถ้าเราต้องการเรียกดูรายการสินค้าที่ถูกผลิตขึ้นเอง กล่าวคือ ไม่มีค่าของข้อมูลในแอทริบิว V_CODE ที่บ่งบอกถึงรหัสของผู้ผลิตสินค้า เราจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      P_CODE, P_DESCRIPT, V_CODE
FROM        PRODUCT
WHERE       V_CODE IS NULL;
```

จากคำสั่งข้างต้น อาจมีคำถามที่ว่าทำไมเราไม่ทำการเปรียบ “V_CODE = NULL” โดยตรง? คำตอบคือ NULL ไม่ใช่ค่าของข้อมูลที่ปรากฏในแอทริบิว แต่จะเป็นคุณสมบัติพิเศษที่บ่งบอกถึงการไม่มีค่าของข้อมูล

การประยุกต์ใช้ตัวดำเนินการ LIKE

ตัวดำเนินการ LIKE จะประยุกต์ใช้แนวความคิดอักขระแทน (wildcard) ที่ซึ่งจะมีการประยุกต์ใช้งาน 2 รูปแบบคือ การใช้เครื่องหมาย ‘%’ เพื่อแทนอักขระใดๆที่ซึ่งจะมีความยาวเท่าไรก็ได้ และ การใช้เครื่องหมาย ‘_’ เพื่อแทนอักขระหนึ่งๆ ตัวอย่างเช่น

- ‘J%’ จะหมายถึงข้อมูลใดๆที่ขึ้นต้นด้วย ‘J’ เช่น ‘J**ohnson**’, ‘J**ones**’, J**ernigan**’, ‘J**uly**’, หรือ ‘J**-2310Q**’
- ‘Jo’ จะหมายถึงข้อมูลใดๆที่ขึ้นต้นด้วย ‘Jo’ เช่น ‘Jo**hnson**’ หรือ ‘Jo**nes**’ เป็นต้น
- ‘%n’ จะหมายถึงข้อมูลใดๆที่ลงท้ายด้วย ‘n’ เช่น ‘**Johnson**’ หรือ ‘**Jernigan**’ เป็นต้น
- ‘_23-456-6789’ จะหมายถึงค่าของข้อมูลใดๆที่มี 1 ตัวอักษร ที่อยู่ก่อนหน้า ‘23-456-6789’ เช่น ‘**1**23-456-6789’, ‘**2**23-456-6789’ หรือ ‘**3**23-456-6789’
- ‘_23-456-678_’ จะหมายถึงค่าของข้อมูลใดๆที่มี 1 ตัวอักษร ที่อยู่ก่อนหน้า และ 1 ตัวอักษรที่อยู่หลัง ‘23-456-6789’ เช่น ‘**1**23-456-678**1**’ หรือ ‘**8**23-456-678**8**’
- ‘_o_es’ จะหมายถึงค่าของข้อมูลใดๆที่ขึ้นต้นด้วย 1 ตัวอักษรใดๆแล้วต้องตามด้วย ‘o’ จากนั้นจะมีอีก 1 ตัวอักษรใดๆตามหลัง ‘o’ แล้วตามด้วย ‘es’ เช่น ‘C**o**nes’, ‘C**o**kes’, ‘t**o**tes’ และ ‘r**o**les’ เป็นต้น

จากกรอบความคิดข้างต้น ถ้าเราต้องการที่จะค้นหาข้อมูลเกี่ยวกับบริษัทผู้ผลิตสินค้าที่มีผู้ติดต่อประสานงานที่ชื่อ Smith แต่ไม่ทราบนามสกุล เราจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        VENDOR
WHERE       V_CONTACT LIKE 'Smith%';
```

(หมายเหตุ หลายๆ RDBMS จะใช้ case-sensitive search ที่ซึ่งการกรอกข้อมูลตัวอักษรพิมพ์ใหญ่และพิมพ์เล็กจะให้ข้อมูลที่ไม่เหมือนกัน แต่อย่างไรก็ตามจะมีบาง RDBMS ที่ไม่ได้สนใจกรณีของ case-sensitive)

แต่สำหรับกรณีที่เราต้องการที่จะค้นหาข้อมูลเกี่ยวกับบริษัทผู้ผลิตสินค้าที่มีผู้ติดต่อประสานงานที่ชื่อไม่ได้ชื่อ Smith เราจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        VENDOR
WHERE       V_CONTACT NOT LIKE 'Smith%';
```

หรืออีกกรณีหนึ่งคือ เราต้องการที่จะค้นหาข้อมูลเกี่ยวกับบริษัทผู้ผลิตสินค้าที่มีผู้ติดต่อประสานงานชื่อ Johnson หรือ Johnsen เราจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        VENDOR
WHERE       V_CONTACT LIKE 'Johns_n';
```

การประยุกต์ใช้ตัวดำเนินการ IN

หลายๆคิวรีในการเรียกดูอาจทำการกำหนดเงื่อนไขโดยการใช้ตัวดำเนินการ OR ที่ซึ่งจะเป็นการกำหนดเงื่อนไขมากกว่าหนึ่งเงื่อนไข เช่น

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE = 21344 OR V_CODE = 24288;
```

จากคำสั่ง SELECT ข้างต้นเราสามารถปรับเปลี่ยนการใช้ตัวดำเนินการ OR ด้วยการใช้ตัวดำเนินการ IN ได้ที่ซึ่งตัวดำเนินการ IN จะมองกลุ่มของค่าข้อมูลที่เราต้องการเปรียบเทียบในรูปแบบของเซต ดังนั้นถ้าค่าของข้อมูลใดเป็นสมาชิกของเซตที่เรากำหนดไว้ในตอนเริ่มต้น เราจะถือว่าค่าของข้อมูลนั้นมีค่าตรงกับเงื่อนไขที่เรากำหนด เราสามารถประยุกต์ใช้ตัวดำเนินการ IN ได้โดยคำสั่ง SELECT เป็น

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);
```

แต่ถ้าค่าของข้อมูลที่ต้องการเปรียบเทียบอยู่ในรูปแบบตัวอักษระ เราจะต้องประยุกต์ใช้ “ ‘’ ” ครอบค่าของข้อมูลที่ต้องการเปรียบเทียบที่ซึ่งการประยุกต์ใช้ ‘’ ดังกล่าวจะทำให้ RDBMS ทราบว่าเราต้องการที่จะพิจารณาข้อมูลในลักษณะของตัวอักษร ตัวอย่างเช่น ถ้าเรากำหนดชนิดของข้อมูล V_CODE ให้อยู่ในรูปแบบ CHAR(5) เราจะสามารถดำเนินการประยุกต์ใช้ตัวดำเนินการ IN ได้เป็น

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (‘21344’, ‘24288’);
```

นอกเหนือจากการประยุกต์ใช้ตัวดำเนินการ IN เพื่อแทนที่ตัวดำเนินการ OR ดังแสดงตัวอย่างข้างต้นแล้ว เรายังสามารถประยุกต์ใช้ตัวดำเนินการ IN เพื่อเชื่อมโยงระหว่างคิวรีและ subquery ได้ ตัวอย่างเช่น ถ้าเราต้องการที่จะเรียกดูข้อมูล V_CODE และ V_NAME ของบริษัทผู้ผลิตสินค้าที่ส่งสินค้าให้กับบริษัท เราต้องสร้างคิวรีหลักและ subquery ด้วยการประยุกต์ใช้ตัวดำเนินการ IN ได้เป็น

```
SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

จากคิวรีข้างต้นจะมีการประยุกต์ใช้ subquery ที่ซึ่งจะมีขั้นตอนการทำงาน 2 ขั้นตอนคือ

1. Inner query หรือ subquery ที่ทำการค้นหา V_CODE ที่ปรากฏในตาราง PRODUCT
2. ทำการค้นหาข้อมูล V_CODE และ V_NAME จากตาราง VENDOR ที่ซึ่งมี V_CODE ตรงกับหนึ่งในเซตของ V_CODE ที่ได้จากขั้นตอนที่ 1

การประยุกต์ใช้ตัวดำเนินการ EXISTS

ตัวดำเนินการ EXISTS จะถูกประยุกต์ใช้ในการค้นหาข้อมูลจากคิวรีอื่นๆ เช่น ถ้า subquery คืนค่าผลลัพธ์เป็นเซตของแถวข้อมูล จากนั้นเราสามารถประยุกต์ใช้ตัวดำเนินการ EXISTS ในการค้นหาข้อมูลจากผลลัพธ์ที่ได้จาก subquery ได้ ตัวอย่างเช่น

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

หรือ

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN * 2);
```

7.4 คำสั่งเพิ่มเติมเกี่ยวกับ data definition

ในส่วนนี้เราจะทำการศึกษาเกี่ยวกับการปรับเปลี่ยนโครงสร้างของตารางข้อมูลที่จะเกี่ยวข้องกับการเพิ่ม-ลดแอทริบิว การเปลี่ยนชนิดของข้อมูลในแอทริบิวหนึ่งๆ นอกจากนั้น เราจะศึกษาเกี่ยวกับการคัดลอกตารางและการคัดลอกบางส่วนของตาราง และท้ายสุดจะทำการศึกษาเกี่ยวกับการลบตารางข้อมูล

การปรับเปลี่ยนโครงสร้างของตารางข้อมูลหนึ่งๆจะถูกดำเนินการภายใต้คำสั่ง ALTER TABLE ที่จะมี 3 ทางเลือกในการดำเนินการคือ **การเพิ่มแอทริบิว (ADD) การปรับเปลี่ยนคุณสมบัติของแอทริบิว (MODIFY) และ การลบแอทริบิว (DROP)** ตามลำดับ แต่อย่างไรก็ตาม RDBMS มักจะไม่อนุญาติให้ทำการลบแอทริบิวออกจากฐานข้อมูล ด้วยเหตุนี้เราจะทำการศึกษาเกี่ยวกับการเพิ่มแอทริบิวและการปรับเปลี่ยนคุณสมบัติของแอทริบิวภายใต้คำสั่ง ALTER TABLE ที่มีรูปแบบทั่วไปดังนี้

```
ATER TABLE   tablename
        {ADD | MODIFY (columnname datatype [{ADD | MODIFY} columnname datatype])};
```

นอกเหนือจากการเพิ่มแอทริบิวหนึ่งๆแล้ว เรายังสามารถประยุกต์ใช้คำสั่ง ALTER TABLE เพื่อใช้ในการเพิ่มเงื่อนไข (constraint) ให้กับตารางข้อมูลได้อีกด้วย โดยการที่จะเพิ่มเงื่อนไขหนึ่งๆให้กับตารางข้อมูลหนึ่งๆจะสามารถดำเนินการได้โดย

```
ALTER TABLE tablename
        ADD constraint [ADD constraint];
```

นอกจากข้างต้น เรายังสามารถประยุกต์ใช้คำสั่ง ALTER TABLE ในการลบแอทริบิวหรือเงื่อนไขต่างๆในคารางข้อมูลได้โดย

```
ALTER TABLE tablename
        DROP {PRIMARY KEY| COLUMN columnname | CONSTRAINT constraintname};
```

7.4.1 การปรับเปลี่ยนชนิดของข้อมูลของแอทริบิวหนึ่งๆ

ถ้าเราต้องการปรับเปลี่ยนชนิดของข้อมูลในแอทริบิวจากเชิงตัวเลขให้กลายเป็นข้อมูลเชิงตัวอักษรที่มีความยาว 5 ตัวอักษร เราสามารถประยุกต์ใช้คำสั่ง ALTER TABLE ได้เป็น

```
ALTER TABLE PRODUCT
        MODIFY (V_CODE CHAR (5));
```

ในหลาย DBMS จะไม่ยอมให้เราทำการปรับเปลี่ยนชนิดของข้อมูลของแอทริบิวหนึ่งๆ เนื่องจากแอทริบิวเหล่านั้นมีค่าของข้อมูลถูกจัดเก็บไว้ก่อนหน้าแล้ว ถ้าเราทำการปรับเปลี่ยนชนิดของข้อมูลที่ไม่เหมาะกับข้อมูลอาจทำให้เกิดปัญหาเกี่ยวกับ referential integrity ได้ ด้วยเหตุนี้ ในขั้นตอนการกำหนดชนิดของข้อมูลเรา

ควรที่จะต้องไตร่ตรองให้ดีเพื่อหลีกเลี่ยงการปรับเปลี่ยนชนิดของข้อมูลในแอทริบิวหนึ่งๆที่อาจก่อให้เกิดปัญหาตามมาได้

7.4.2 การเพิ่มแอทริบิวหนึ่งๆในตารางข้อมูล

เราสามารถประยุกต์ใช้คำสั่ง ALTER TABLE ในการเพิ่มแอทริบิวหนึ่งๆหรือหลายแอทริบิว เช่น ถ้าเราต้องการเพิ่มแอทริบิว P_SALECODE เข้าไปในตาราง PRODUCT ที่ซึ่งข้อมูลในแอทริบิว P_SALECODE จะสามารถมีได้เป็น 1, 2 หรือ 3 เท่านั้น ดังนั้นเราสามารถกำหนดชนิดข้อมูลของแอทริบิว P_SALECODE เป็น CHAR(1) และสามารถประยุกต์ใช้คำสั่ง ALTER TABLE ได้เป็น

```
ALTER TABLE PRODUCT
    ADD (P_SALECODE CHAR (1));
```

การเพิ่มแอทริบิวใหม่ให้กับตารางข้อมูล เราไม่ควรที่จะกำหนดเงื่อนไข NOT NULL ให้กับแอทริบิวนั้นๆ เนื่องจากตารางอาจมีข้อมูล (แถวข้อมูล) บรรจุอยู่ที่ซึ่งจะทำให้ค่าข้อมูลในแอทริบิวใหม่ของข้อมูลที่ถูกบรรจุอยู่มีค่าเป็น NULL จากนั้นเราจะต้องทำการอัพเดทข้อมูลในภายหลังเพื่อปรับเปลี่ยนให้ค่า NULL สำหรับแถวข้อมูลเหล่านั้นหายไป

7.4.3 การลบแอทริบิวหนึ่งๆออกจากตารางข้อมูล

ในบางกรณีเราอาจจำเป็นต้องทำการลบแอทริบิวหนึ่งๆออกจากตารางข้อมูล เช่น ถ้าเราต้องการลบแอทริบิว V_ORDER ออกจากตาราง VENDOR เราจะสามารถดำเนินการได้เป็น

```
ALTER TABLE VENDOR
    DROP COLUMN V_ORDER;
```

ในการลบแอทริบิวออกจากตารางข้อมูลภายใต้ DBMS จะมีเงื่อนไขเพิ่มเติมที่ว่าเราอาจไม่สามารถลบแอทริบิวที่ทำหน้าที่เป็น foreign key ได้ และเราอาจไม่สามารถลบแอทริบิวหนึ่งๆของตารางข้อมูลที่ซึ่งมีเพียงแอทริบิวเดียวได้

7.4.4 การคัดลอกบางแอทริบิวในตารางข้อมูล

ในบางสถานการณ์เราอาจจำเป็นต้องทำการแยกตารางข้อมูลหนึ่งๆออกเป็นหลายตารางข้อมูลย่อย ที่ซึ่งจะต้องการการคัดลอกข้อมูลบางแอทริบิวจากตารางข้อมูลไปสู่ตารางข้อมูลย่อยๆหนึ่งๆ ตัวอย่างเช่น ถ้าเราต้องการคัดลอกแอทริบิว P_CODE, P_DESCRIPT, P_PRICE และ V_CODE จากตาราง PRODUCT ไปเก็บไว้ในตารางข้อมูลใหม่ที่ชื่อ PART เราจะสามารถดำเนินการโดยเริ่มจากการกำหนดโครงสร้างตารางข้อมูลด้วยการใช้คำสั่ง CREATE TABLE ดังนี้

```
CREATE TABLE        PART(
PART_CODE           CHAR(8)        NOT NULL       UNIQUE,
```

```
PART_DESCRIPT       CHAR(35),
PART_PRICE          DECIMAL(8,2),
V_CODE              INTEGER,
PRIMARY KEY (PART_CODE));
```

จากการกำหนดโครงสร้างข้างต้น เราจะสังเกตุได้ว่าชื่อแอทริบิวในตารางใหม่ที่สร้างขึ้นจะไม่เหมือนกับชื่อแอทริบิวในตาราง PRODUCT แต่อย่างไรก็ตาม การกำหนดชื่อในลักษณะดังกล่าวจะไม่ก่อให้เกิดปัญหา เนื่องจากเราสามารถทำการเชื่อมโยงความเกี่ยวข้องกันของแอทริบิวจากทั้งสองตารางข้อมูลได้ โดยในการเพิ่มข้อมูลเข้าไปในตาราง PART จะสามารถทำได้โดยการประยุกต์ใช้คำสั่ง INSERT ที่ซึ่งจะมีรูปแบบเป็น

```
INSERT INTO     target_tablename [(target_columnlist)]
SELECT          source_columnlist
FROM            source_table;
```

จากรูปแบบคำสั่งข้างต้นเราสามารถประยุกต์ใช้คำสั่งดังกล่าวเพื่อคัดลอกข้อมูลจากตาราง PRODUCT ไปยังตาราง PART ได้เป็น

```
INSERT INTO     PART (PART_CODE, PART_DESCRIPT, PART_PRICE, V_CODE)
SELECT          P_CODE, P_DESCRIPT, P_PRICE, V_CODE
FROM            PRODUCT;
```

นอกเหนือจากรูปแบบคำสั่งข้างต้น เรายังสามารถประยุกต์ใช้คำสั่งในอีกรูปแบบหนึ่งเพื่อคัดลอกข้อมูลจากตาราง PRODUCT ไปยังตาราง PART ที่ซึ่งจะสามารถรวบการสร้างตารางและการคัดลอกข้อมูลเข้าด้วยกันได้ดังนี้

```
CREATE TABLE    PART AS
SELECT          P_CODE AS PART_CODE, P_DESCRIPT AS PART_DESCRIPT, P_PRICE AS
                PART_PRICE, V_CODE
FROM            PRODUCT;
```

จากคำสั่งข้างต้น ถ้าแอทริบิวที่ใช้เก็บข้อมูลหนึ่งๆมีชื่อไม่เหมือนกันในสองตาราง เราจะต้องสร้างการเชื่อมโยงด้วยการใช้ “AS” แต่ถ้าชื่อแอทริบิวเหมือนกันเราจะสามารถกำหนดแอทริบิวนั้นๆได้โดยตรง และจากคำสั่งข้างต้นเรายังไม่ได้ทำการกำหนด primary key ให้กับตาราง PART ที่ซึ่งจะต้องทำการกำหนดในภายหลัง

7.4.5 การกำหนด primary key และ foreign key ในภายหลัง

เมื่อเราทำการสร้างข้อมูลใหม่จากการคัดลอกข้อมูลจากตารางอื่น และตารางข้อมูลใหม่ที่สร้างขึ้นยังไม่ได้ทำการกำหนด primary key ที่ซึ่งจะทำให้ตารางข้อมูลยังไม่มีคุณสมบัติ entity integrity ด้วยเหตุนี้เราจึงจำเป็นต้องกำหนด primary key และ foreign key ให้กับตารางข้อมูลดังแสดงในตัวอย่างดังนี้

```
ALTER TABLE PART
        ADD PRIMARY KEY (PART_CODE);

ALTER TABLE PART
        ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

แต่อย่างไรก็ตาม เราสามารถกำหนด primary key และ foreign key พร้อมๆกันได้ด้วยการประยุกต์ใช้คำสั่งเดียวดังนี้

```
ALTER TABLE PART
        ADD PRIMARY KEY (PART_CODE)
        ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

7.4.6 การลบตารางข้อมูลหนึ่งๆออกจากฐานข้อมูล

ตารางข้อมูลหนึ่งๆจะถูกลบออกจากฐานข้อมูลได้โดยการประยุกต์ใช้คำสั่ง DROP TABLE ตัวอย่างเช่น ถ้าเราต้องการลบตาราง PART ออกจากฐานข้อมูล เราจะสามารถดำเนินการได้ดังนี้

```
DROP TABLE  PART;
```

.ในการลบตารางข้อมูลหนึ่งๆเราจะต้องพิจารณาถึงความสัมพันธ์ของตารางข้อมูลด้วย ถ้าตารางข้อมูลที่ต้องการลบอยู่ในฝั่ง ‘1’ ของความสัมพันธ์เราจะสามารถลบตารางข้อมูลได้แต่ถ้าตารางข้อมูลอยู่ในฝั่ง ‘m’ จะทำให้เกิดปัญหาเกี่ยวกับ foreign key integrity ได้

7.5 คำสำคัญเพิ่มเติมที่มักถูกใช้ในคำสั่ง SELECT

จากเนื้อหาในส่วน 7.2 เราได้ศึกษาเกี่ยวกับการเรียกดูข้อมูลด้วยการประยุกต์ใช้คำสั่ง SELECT แต่อย่างไรก็ตาม ยังมีฟังก์ชันการทำงานอื่นๆที่สามารถประยุกต์ใช้ร่วมกับคำสั่ง SELECT ที่จะทำให้เราสามารถเรียกดูข้อมูลได้หลายแง่มุม ดังนี้

7.5.1 การจัดเรียงผลลัพธ์

การจัดเรียงผลลัพธ์ที่ได้จากการเรียกดูข้อมูลจะสามารถดำเนินการได้โดยการเพิ่มคำสั่ง ORDER BY เข้าไปในคำสั่ง SELECT ที่ซึ่งจะทำให้คำสั่ง SELECT มีรูปแบบที่เปลี่ยนไปดังนี้

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist]
[ORDER BY   columnlist [ASC | DSC]];
```

ตัวอย่างเช่น ถ้าเราต้องการเรียกดูรายการสินค้าที่ถูกจัดเก็บในตาราง PRODUCT โดยผลลัพธ์ที่ได้จะต้องเรียงลำดับตามราคาสินค้า เราจะสามารถประยุกต์ใช้คำสั่ง SELECT ได้เป็น

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE;
```

จากคำสั่ง SQL ข้างต้น เราจะสังเกตุได้ว่าคำสั่งดังกล่าวไม่มีการกำหนดลำดับว่าเป็นแบบมากไปน้อยหรือน้อยไปมาก ซึ่งโดยปกติของคำสั่ง ORDER BY จะเป็นการเรียงลำดับแบบน้อยไปมากโดยอัตโนมัติ แต่อย่างไรก็ตาม ถ้าเราต้องการเรียงลำดับผลลัพธ์ในรูปแบบมากไปน้อย เราจะต้องทำการเพิ่ม DESC ในคำสั่ง ORDER BY ดังนี้

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE DESC;
```

นอกเหนือจากการประยุกต์ใช้แอทริบิวหนึ่งๆในการเรียงลำดับข้อมูลแล้ว เรายังสามารถประยุกต์ใช้กลุ่มของแอทริบิวในการเรียงลำดับข้อมูลได้ โดยลำดับข้อมูลจะถูกเรียงตามแอทริบิวแรกที่กำหนดเป็นอันดับแรก จากนั้นจะเรียงลำดับตามแอทริบิวอื่นๆที่กำหนด ตัวอย่างเช่น

```
SELECT      EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM        EMPLOYEE
ORDER BY    EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

จากตัวอย่างข้างต้น การเรียงลำดับจะเริ่มจากการเรียงลำดับตามแอทริบิว EMP_LNAME หลังจากเรียงลำดับเสร็จแล้ว แถวข้อมูลใดที่มี EMP_LNAME เหมือนกันจะถูกเรียงลำดับตาม F_NAME และท้ายสุดแถวที่มี EMP_LNAME และ EMP_FNAME เหมือนกันจะถูกเรียงลำดับตาม EMP_INITIAL ตามลำดับ นอกจากการเรียงลำดับด้วยการประยุกต์ใช้หนึ่งหรือหลายแอทริบิวแล้ว เรายังสามารถทำการเรียงลำดับร่วมกับการกำหนดเงื่อนไขต่างๆในคิวรีได้อีกด้วย ตัวอย่างเช่น การเรียกดูข้อมูลรายการสินค้าจากตาราง PRODUCT ที่เป็นรายการสินค้าที่ถูกจัดเก็บเข้าโกดังสินค้าก่อนวันที่ 21 Jan 2010 และมีราคาไม่เกิน 50.00 โดยผลลัพธ์ที่ได้จะต้องถูกเรียงลำดับตามรหัสบริษัทผู้ผลิตสินค้าและราคาสินค้าโดยเรียงลำดับจากมากไปน้อย ที่ซึ่งจะสามารถกำหนดคำสั่ง SELECT ได้เป็น

```
SELECT      P_DESCRIPT, V_CODE, P_INDATE, P_PRICE
FROM        PRODUCT
WHERE       P_INDATE < '21-Jan-2010' AND P_PRICE <= 50.00
ORDER BY    V_CODE, P_PRICE DESC;
```

7.5.2 การเรียกดูข้อมูลที่ไม่ซ้ำกัน

ในการเรียกดูข้อมูล เราสามารถกำหนดให้ผลลัพธ์ที่ได้จะต้องไม่ซ้ำกันด้วยการประยุกต์ใช้คำสั่ง DISTINCT แทรกเข้าไปในคำสั่ง SELECT ตัวอย่างเช่น ถ้าเราต้องการเรียกดูรหัสบริษัทผู้ผลิตสินค้าที่ทำการผลิตสินค้าให้บริษัทแต่เราจะต้องการรหัสที่ไม่ซ้ำกัน เราจะสามารถดำเนินการด้วยการประยุกต์ใช้คำสั่ง SELECT ในการค้นหาข้อมูลจากตาราง PRODUCT ได้ดังนี้

```
SELECT DISTINCT     V_CODE
FROM                PRODUCT;
```

7.5.3 ฟังก์ชันการคำนวณที่ประยุกต์ใช้กับคำสั่ง SELECT

การเรียกดูครั้งหนึ่งๆอาจเกิดจากการนับจำนวนแถวข้อมูลที่สอดคล้องกับเงื่อนไขที่กำหนด การหาค่าสูงสุด/ต่ำสุดในแอทริบิวหนึ่งๆ การหาผลรวมของข้อมูลในแอทริบิวหนึ่งๆ หรือการหาค่าเฉลี่ยของข้อมูลในแอทริบิวหนึ่งที่ซึ่งจะมีการดำเนินการดังนี้

ฟังก์ชัน COUNT

ฟังก์ชัน COUNT จะถูกใช้ในการนับจำนวนแถวของข้อมูลที่เกี่ยวเนื่องกับแอทริบิวหนึ่งๆ ตัวอย่างเช่น ถ้าเราต้องการค้นหาจำนวนบริษัทผู้ผลิตสินค้าที่แตกต่างกันที่มีการผลิตสินค้าให้กับบริษัท เราจะสามารถประยุกต์ใช้ฟังก์ชัน COUNT ในคำสั่ง SELECT ได้เป็น

```
SELECT      COUNT (DISTINCT V_CODE)
FROM        PRODUCT;
```

หรือถ้าเราต้องการที่จะค้นหาจำนวนผู้ผลิตสินค้าที่แตกต่างกันที่ผลิตสินค้าให้กับบริษัทที่ซึ่งมีราคาไม่เกิน 10.00$ เราจะสามารถประยุกต์ใช้ฟังก์ชัน COUNT ในคำสั่ง SELECT ได้เป็น

```
SELECT      COUNT (DISTINCT V_CODE)
FROM        PRODUCT
WHERE       P_PRICE <= 10.00;
```

หรือถ้าเราต้องการที่จะค้นหาจำนวนสินค้าที่มีราคาไม่เกิน 5.00$ เราจะสามารถประยุกต์ใช้ฟังก์ชัน COUNT ในคำสั่ง SELECT ได้เป็น

```
SELECT      COUNT (*)
FROM        PRODUCT
WHERE       P_PRICE <= 5.00;
```

ฟังก์ชัน MAX และ MIN

ฟังก์ชัน MAX และ MIN จะถูกประยุกต์ใช้ในการค้นหาค่าสูงสุดและต่ำสุดที่ปรากฏในแอทริบิวหนึ่งๆ ตัวอย่างเช่น ถ้าเราต้องการค้นหาราคาสูงสุดของรายการสินค้า เราจะสามารถประยุกต์ใช้ฟังก์ชัน MAX ในคำสั่ง SELECT ได้เป็น

```
SELECT      MAX(P_PRICE)
FROM        PRODUCT;
```

หรือถ้าเราต้องการค้นหาราคาต่ำสุดของรายการสินค้า เราจะสามารถประยุกต์ใช้ฟังก์ชัน MIN ในคำสั่ง SELECT ได้เป็น

```
SELECT      MIN(P_PRICE)
FROM        PRODUCT;
```

หรือถ้าเราต้องการค้นหารหัสสินค้า คำอธิบายสินค้า และราคาสินค้าที่มีราคาสูงสุด เราจะสามารถประยุกต์ใช้ฟังก์ชัน MAX ในคำสั่ง SELECT ได้เป็น

```
SELECT      P_CODE, P_DESCRIPT, P_PRICE
FROM        PRODUCT
WHERE       P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);
```

ฟังก์ชัน SUM

ฟังก์ชัน SUM จะถูกประยุกต์ใช้ในการคำนวณหาผลรวมของค่าข้อมูลที่ปรากฏในแอทริบิวหนึ่งๆ ตัวอย่างเช่น ถ้าเราต้องการคำนวณผลรวมของยอดหนี้สินที่ลูกค้าทั้งหมดยังไม่ได้ทำการชำระเงินเมื่อทำการซื้อสินค้า เราจะสามารถประยุกต์ใช้ฟังก์ชัน SUM ในคำสั่ง SELECT ได้เป็น

```
SELECT      SUM(CUS_BALANCE) AS TOTBALANCE
FROM        CUSTOMER;
```

หรือถ้าเราต้องการที่จะค้นหาจำนวนราคาสินค้าทั้งหมดที่ถูกจัดเก็บอยู่ในโกดังสินค้า เราจะสามารถประยุกต์ใช้ฟังก์ชัน SUM ในคำสั่ง SELECT ได้เป็น

```
SELECT      SUM(P_QOH * P_PRICE) AS TOTVALUE
FROM        PRODUCT;
```

ฟังก์ชัน AVG

ฟังก์ชัน AVG จะมีรูปแบบคล้ายกับฟังก์ชัน MIN และ MAX ที่ซึ่งจะใช้สำหรับการคำนวณค่าเฉลี่ยของข้อมูลที่ปรากฏในแอทริบิวหนึ่งๆ ตัวอย่างเช่น ถ้าเราต้องการที่จะค้นหารายละเอียดสินค้าที่มีราคาสินค้ามากกว่าราคาสินค้าโดยเฉลี่ยของรายการสินค้าทั้งหมดและทำการเรียงลำดับข้อมูลรายการสินค้าตามราคาสินค้าโดยเรียงลำดับจากมากไปน้อย

```
SELECT      P_CODE, P_DESCRIPT, P_QOH, P_PRICE, V_CODE
FROM        PRODUCT
WHERE       P_PRICE > (SELECT AVG(P_PRICE) FROM PRODUCT)
ORDER BY    P_PRICE DESC;
```

7.5.4 การจัดกลุ่มข้อมูล

การเรียกดูข้อมูลอาจเกิดจากการจัดกลุ่มข้อมูลที่ซึ่งจะสามารถประยุกต์ใช้คำสั่ง GROUP BY ที่ซึ่งจะมีรูปแบบดังนี้

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist]
[GROUP BY   columnlist]
[HAVING     conditionlist]
[ORDER BY   columnlist [ASC | DESC]];
```

ตัวอย่างเช่น ถ้าเราต้องการนับจำนวนรายการสินค้าที่ถูกผลิตจากบริษัทผู้ผลิตสินค้าหนึ่งๆ เราจะต้องจัดกลุ่มรายการสินค้าตามบริษัทสินค้าหนึ่งๆจากนั้นจึงทำการนับจำนวนรายการสินค้าที่อยู่ในแต่ละกลุ่มที่ได้จัดกลุ่มไว้แล้ว ที่ซึ่งเราจะสามารถประยุกต์ใช้ฟังก์ชัน GROUP BY ในคำสั่ง SELECT ได้เป็น

```
SELECT      V_CODE, COUNT (DISTINCT (P_CODE))
FROM        PRODUCT
GROUP BY    V_CODE;
```

การจัดกลุ่มข้อมูลด้วยการประยุกต์ใช้ฟังก์ชัน HAVING

การประยุกต์ใช้ฟังก์ชัน GROUP BY จะสามารถประยุกต์ใช้ร่วมกับฟังก์ชัน HAVING ที่ซึ่งจะมีลักษณะคล้ายกับการกำหนดเงื่อนไข WHERE ในคำสั่ง SELECT ตัวอย่างเช่น ถ้าเราต้องการนับจำนวนรายการสินค้าที่ถูกจัดเก็บในโกดังสินค้าโดยนับจากรายการสินค้าที่ถูกผลิตจากบริษัทผู้ผลิตสินค้าเดียวกัน โดยจำนวนที่ต้องการเรียกดูจะต้องเป็นจำนวนนับของรายการสินค้าที่มีค่าเฉลี่ยของรายการสินค้าน้อยกว่า 10$ จากความ

ต้องการดังกล่าวเราจะสามารถประยุกต์ใช้ฟังก์ชัน GROUP BY ร่วมกับฟังชัน HAVING กับคำสั่ง SELECT ได้เป็น

```
SELECT    V_CODE, COUNT (DISTINCT (P_CODE)), AVG (P_PRICE)
FROM      PRODUCT
GROUP BY  V_CODE
HAVING    AVG (P_PRICE) < 10;
```

ลองพิจารณาอีกตัวอย่างหนึ่งคือ

```
SELECT    V_CODE, SUM (P_QOH * P_PRICE) AS TOTCOST
FROM      PRODUCT
GROUP BY  V_CODE
HAVING    (SUM (P_QOH * P_PRICE) > 500)
ORDER BY  SUM (P_QOH * P_PRICE) DESC;
```

จากตัวอย่างข้างต้นจะประกอบไปด้วยการดำเนินการ 3 ขั้นตอน คือ

- การรวบรวมต้นทุนของรายการสินค้าที่ถูกจัดเก็บในโกดังด้วยการจัดกลุ่มรายการสินค้าที่ถูกผลิตจากบริษัทผู้ผลิตสินค้าเดียวกัน
- การเลือกราคาต้นทุนสินค้าที่มีจำนวนเงินมากกว่า 500$
- ทำการเรียงลำดับผลลัพธ์ที่ได้จากขั้นตอนที่สองโดยเรียงลำดับตามราคาต้นทุนจากมากไปหาน้อย

7.6 ตารางเสมือนและการสร้างมุมมอง

ผลลัพธ์ที่ได้จากคำสั่ง SELECT จะเปรียบได้กับเราได้ทำการสร้างตารางข้อมูลใหม่ที่เป็นตารางเสมือน โดบารางข้อมูลใหม่จะประกอบไปด้วยข้อมูลตามเงื่อนไขที่กำหนดในคิวรี ดังนั้นเราจะสามารถกล่าวได้ว่ามุมมองหนึ่งๆ (view) จะเป็นตารางเสมือนที่เกิดขึ้นจากการประยุกต์ใช้คำสั่ง SELECT ที่ซึ่งจะประกอบไปด้วยคอลัมน์ต่างๆ และแถวของข้อมูล โดยการสร้างมุมมองหนึ่งๆเราจะต้องประยุกต์ใช้คำสั่ง CREATE VIEW ที่ซึ่งจะมีรูปแบบคำสั่งเป็น

CREATE VIEW *viewname* AS SELECT *query*;

คำสั่ง CREATE VIEW จะเป็นคำสั่งในหมวดหมู่ data definition ที่จะประกอบไปด้วย subquery ที่เป็นคำสั่ง SELECT ตัวอย่างเช่น ถ้าเราต้องการสร้างมุมมองที่มีชื่อว่า PRICEGT50 ที่จะประกอบไปด้วยข้อมูลคำอธิบายรายการสินค้า จำนวนที่ถูกจัดเก็บในโกดังสินค้า และราคาสินค้าที่มากกว่า 50$ เราจะสามารถประยุกต์ใช้คำสั่ง CREATE VIEW ที่มีรูปแบบเป็น

```
CREATE VIEW     PRICEGT50 AS
SELECT          P_DESCRIPT, P_QOH, P_PRICE
FROM            PRODUCT
WHERE           P_PRICE > 50.00;
```

จากการสร้างมุมมองข้างต้นจะทำให้เราสามารถประยุกต์ใช้มุมมองได้ในฐานะตารางข้อมูลหนึ่ง และเรายังสามารถทำการอัพเดทมุมมองได้ในทุกๆครั้งที่มีการเรียกใช้มุมมอง เมื่อเราทำการสร้างตารางข้อมูลแล้ว เราจะสามารถเรียกดูข้อมูลจากมุมมองที่สร้างขึ้นได้ โดยอาจประยุกต์ใช้คำสั่ง SELECT ดังนี้

```
SELECT          * FROM PRICEGT50;
```

อีกตัวอย่างหนึ่งคือ ถ้าเราต้องการสร้างรายงานที่แสดงถึงผลสรุปของต้นทุนในรายการสินค้าและจำนวนชิ้นสินค้าที่ถูกจัดเก็บในโกดังสินค้าที่ซึ่งถูกจัดกลุ่มโดยบริษัทผู้ผลิตสินค้าเดียวกัน เราจะสามารถสร้างมุมมองได้เป็น

```
CREATE VIEW     PROD_STATS AS
SELECT          V_CODE, SUM (P_QOH * P_PRICE) AS TOTCOST,
                MAX(P_QOH) AS MAXQTY, MIN(P_QOH) AS MINQTY,
                AVG (P_QOH) AS AVGQTY
FROM            PRODUCT
GROUP BY        V_CODE;
```

7.7 การเชื่อมโยงข้อมูลระหว่างตารางข้อมูลในฐานข้อมูล

การเชื่อมโยงข้อมูล (join) ระหว่างตารางข้อมูล เราจะสามารถดำเนินการได้ผ่านการเชื่อมโยง foreign key ที่ซึ่งจะทำหน้าที่เป็น primary key ในตารางข้อมูลหนึ่งและถูกจัดเก็บในอีกตารางข้อมูลหนึ่ง โดยการเชื่อมโยงข้อมูลจะดำเนินการก็ต่อเมื่อเราต้องการข้อมูลในตารางข้อมูลมากกว่าหนึ่งตาราง การเชื่อมโยงข้อมูลจะไม่ต้องการคำสั่งพิเศษ แต่เราจะต้องทำการเพิ่มชื่อตารางข้อมูลทั้งหมดที่ต้องทำการเรียกดูข้อมูลในส่วนของ FROM และเราจะต้องทำการกำหนดเงื่อนไขเพิ่มเติมโดยจะต้องเชื่อมโยงระหว่างแอทริบิวที่ทำหน้าที่เป็น foreign key ในส่วนของ WHERE ตัวอย่างเช่น ถ้าเราต้องการเชื่อมโยงข้อมูลระหว่างตาราง VENDOR และตาราง PRODUCT เราจะสามารถดำเนินการได้โดยกำหนดให้ส่วนของ FROM มีชื่อของตารางข้อมูลเป็น VENDOR และ PRODUCT และทำการกำหนดเงื่อนไขที่ซึ่งจะเป็นการกำหนดให้ข้อมูลที่เป็น foreign key ในทั้งสองตารางจะต้องมีค่าเท่ากัน ดังแสดงในคำสั่ง SELECT ดังนี้

```
SELECT          P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM            PRODUCT, VENDOR
WHERE           PRODUCT.V_CODE = VENDOR.V_CODE;
```

นอกเหนือจากการเชื่อมโยงข้อมูลปกติเราสามารถดำเนินการเชื่อมโยงข้อมูลกับฟังก์ชันการทำงานอื่นๆ เช่น การเรียงลำดับข้อมูล ที่ซึ่งจะสามารถดำเนินการได้ดังนี้

```
SELECT      PRODUCT.P_DESCRIPT, PRODUCT.P_PRICE, VENDOR.V_NAME,
            VENDOR.V_CONTACT, VENDOR.V_AREACODE, VENDOR.V_PHONE
FROM        PRODUCT, VENDOR
WHERE       PRODUCT.V_CODE = VENDOR.V_CODE
ORDER BY    PRODUCT.P_PRICE;
```

หรืออีกกรณีหนึ่งที่เราสามารถดำเนินการได้คือ การเพิ่มเงื่อนไขอื่นๆนอกเหนือจากการกำหนดให้ค่าของข้อมูลในแอทริบิวที่เป็น foreign key จากทั้งสองตารางข้อมูลจะต้องมีค่าเท่ากัน ที่ซึ่งจะสามารถดำเนินการได้เป็น

```
SELECT      P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        PRODUCT, VENDOR
WHERE       PRODUCT.V_CODE = VENDOR.V_CODE AND
            P_INDATE > '15-Jan-2010';
```

จากตัวอย่างข้างต้นจะเป็นการเรียกดูข้อมูลรายการสินค้าที่ถูกจัดเก็บในโกดังสินค้าหลังจากวันที่ 15 Jan 2010 โดยข้อมูลรายการสินค้าที่ต้องการเรียกดูจะประกอบไปด้วยข้อมูลคำอธิบายรายการสินค้า ราคาสินค้า ชื่อบริษัทผู้ผลิตสินค้า ชื่อผู้ติดต่อของบริษัทผู้ผลิตสินค้า ที่อยู่ของบริษัทผู้ผลิตสินค้า และเบอร์โทรศัพท์ของบริษัทผู้ผลิตสินค้า ตามลำดับ อีกตัวอย่างหนึ่งคือ ถ้าเราต้องการเรียกดูข้อมูลนามสกุลลูกค้า หมายเลขใบแจ้งหนี้ วันที่ที่ปรากฏในใบแจ้งหนี้ และคำอธิบายรายการสินค้าสำหรับทุกใบแจ้งหนี้ของลูกค้ารหัส 10014 เราจะสามารถสร้างการเชื่อมโยงข้อมูลได้เป็น

```
SELECT      CUS_LNAME, INVOICE.INV_NUMBER, INV_DATE, P_DESCRIPT
FROM        CUSTOMER, INVOICE, LINE, PRODUCT
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE AND
            INVOICE.INV_NUMBER = LINE.INV_NUMBER AND
            LINE.P_CODE = PRODUCT.P_CODE AND
            CUSTOMER.CUS_CODE = 10014
ORDER BY    INV_NUMBER;
```

ท้ายสุดเราต้องจะต้องระมัดระวังในการสร้างการเชื่อมโยงข้อมูลในลักษณะวงกลม ตัวอย่างเช่น ตาราง A เชื่อมโยงกับตาราง B, ตาราง B เชื่อมโยงกับตาราง C และตาราง C เชื่อมโยงกับตาราง A โดยเราจะต้องทำการเชื่อมโยงข้อมูลให้ไม่เกิดลักษณะวงกลมเกิดขึ้นที่ซึ่งจะทำการเชื่อมโยงตาราง A กับตาราง B และตาราง B กับตาราง C เท่านั้น

7.7.1 การเชื่อมโยงข้อมูลด้วยการใช้ชื่อแทน

ในการเรียกดูข้อมูลเราอาจทำการใช้ชื่อย่อหรือชื่อแทนตารางข้อมูล ตัวอย่างเช่น การใช้ P แทนตาราง PRODUCT และใช้ V แทนตาราง VENDOR ที่ซึ่งจะทำให้เราสามารถทำการเชื่อมโยงข้อมูลด้วยการใช้ P และ V ได้เป็น

```
SELECT      P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM        PRODUCT P, VENDOR V
WHERE       P.V_CODE = V.V_CODE
ORDER BY    P_PRICE;
```

7.7.2 การเชื่อมโยงข้อมูลแบบ recursive join

การเชื่อมโยงแบบข้อมูลแบบ recursive join จะเป็นการเชื่อมโยงข้อมูลในตารางข้อมูลเดียวเท่านั้น ตัวอย่างเช่น ถ้าเราต้องการที่จะเรียกดูข้อมูลพนักงานที่มีหัวหน้าคนเดียวกัน เราจะสามารถประยุกต์ใช้การแทนที่และการเชื่อมโยงข้อมูลแบบ recursive join ได้เป็น

```
SELECT      E.EMP_MGR, M.EMP_LNAME, M.EMP_NUM, E.EMP_LNAME
FROM        EMPLOYEE E, EMPLOYEE M
WHERE       E.EMP_MGE =M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

7.7.3 การเชื่อมโยงข้อมูลแบบ outer join

จากเนื้อหาในบทที่ 3 เราได้ศึกษาเกี่ยวกับ outer join ที่ซึ่งจะแบ่งออกเป็น left outer join และ right outer join ถ้าเราทำการพิจารณาตาราง PRODUCT และตาราง VENDOR เราจะสามารถทำการเชื่อมโยงแบบ left outer join ที่จะแสดงข้อมูลในตาราง VENDOR ที่มีข้อมูลตรงกับข้อมูลตาราง PRODUCT ได้ดังนี้

```
SELECT      P_CODE, VENDOR.V_CODE, V_NAME
FROM        VENDOR LEFT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

แต่ถ้าเราต้องการเชื่อมโยงข้อมูลแบบ right outer join ที่จะแสดงข้อมูลในตาราง PRODUCT ที่มีข้อมูลตรงกับข้อมูลในตาราง VENDOR ได้ดังนี้

```
SELECT      PRODUCT.P_CODE, VENDOR.V_CODE, V_NAME
FROM        VENDOR RIGHTJOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

คำถามท้ายบท

1. การประยุกต์ใช้คำสั่ง WHERE และ HAVING ในการกำหนดเงื่อนไขภายใต้คำสั่ง SELECT จะมีความแตกต่างกันอย่างไร
2. จงอธิบายว่าเพราะเหตุใดคำสั่ง SELECT V_CODE, SUM(P_QOH) FROM PRODUCT; ถึงมีข้อผิดพลาดเกิดขึ้นและจงแก้ไขคำสั่งดังกล่าวให้ถูกต้อง
3. เมื่อเราทำการกำหนด primary key จะทำให้ตารางข้อมูลมีความสอดคล้องกับกฎความสมบูรณ์ประเภทใด
4. เราสามารถประยุกต์ใช้คำสั่งใดในการที่จะทำให้ตารางข้อมูลคงไว้ซึ่งกฎ referential integrity
5. จากการกำหนดเงื่อนไข WHERE V_STATE IN ('TN', 'FL', 'GA') ถ้าเราต้องการกำหนดเงื่อนไขโดยไม่ใช้คำสั่ง IN เราจะสามารถกำหนดเงื่อนไขได้อย่างไร
6. จงอธิบายถึงความแตกต่างระหว่างคำสั่ง ORDER BY และ GROUP BY
7. จงอธิบายว่าเพราะเหตุใดสองคำสั่งคิวรีด้านล่างถึงให้ผลลัพธ์ไม่เหมือนกัน

   SELECT DISTINCT COUNT (V_CODE) FROM PRODUCT;

   SELECT COUNT(DISTINCT V_CODE) FROM PRODUCT;
8. จงอธิบายถึงความแตกต่างระหว่างฟังก์ชัน COUNT และ SUM
9. เพราะเหตุใดเราจึงควรที่จะกำหนดประเภทของข้อมูลวันที่ด้วยการใช้ DATE แทนที่การกำหนดข้อมูลวันที่เป็นตัวอักษร
10. จงอธิบายถึงความแตกต่างระหว่าง inner join และ outer join