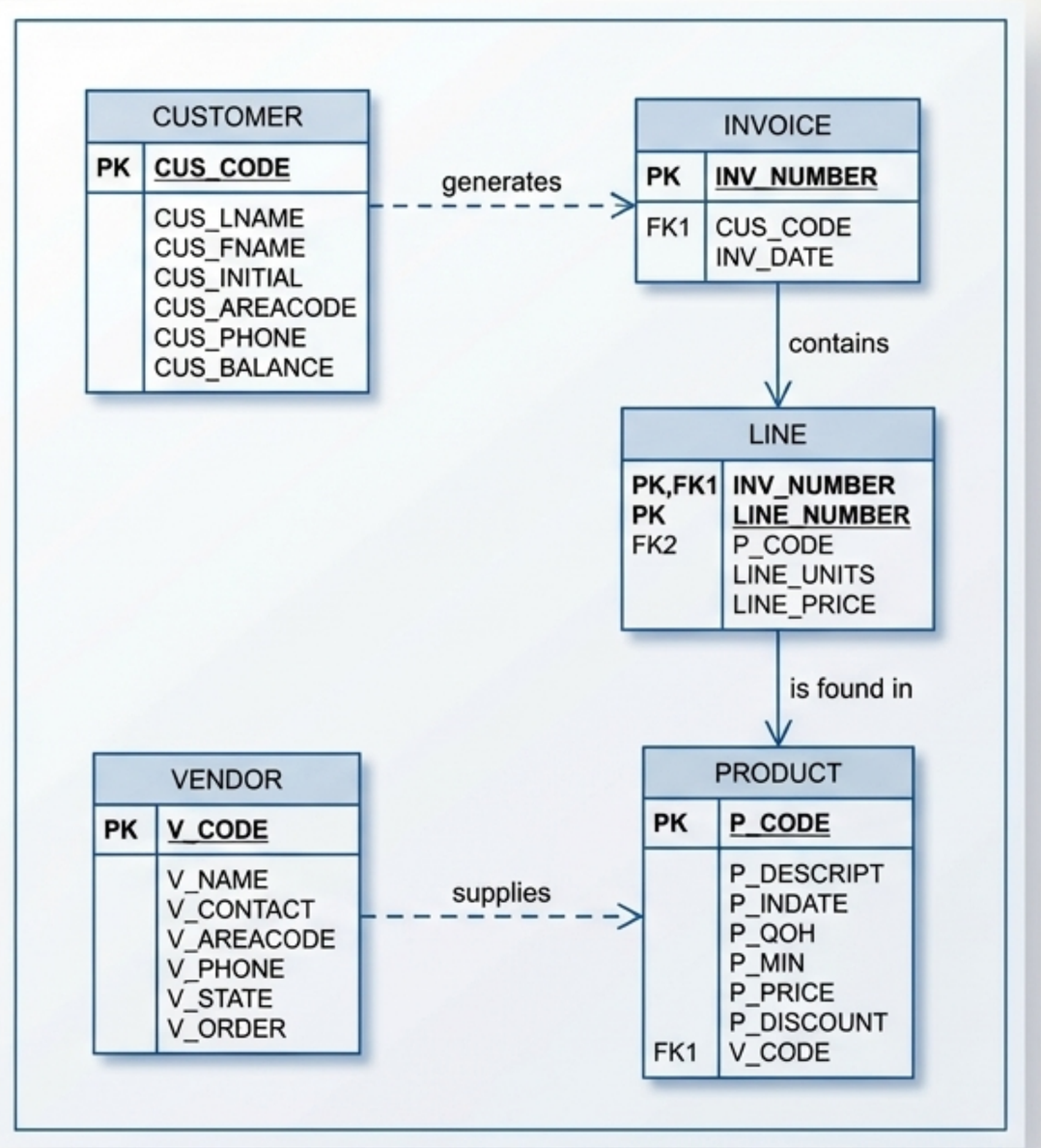
SQL: จากพิมพ์เขียวสู่ฐานข้อมูลที่มีชีวิต

คู่มือการเรียนรู้ภาษา SQL สำหรับการสร้าง
จัดการสร้าง จัดการ และสืบค้นข้อมูลอย่างมืออาชีพ

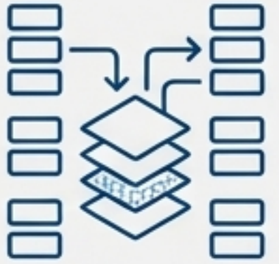
จาก 'แนวคิด' สู่ 'โครงสร้างจริง'

- ใน 6 บทก่อนหน้า เราได้เรียนรู้การออกแบบฐานข้อมูลผ่าน ‘แบบจำลองข้อมูล’ (Data Model) ซึ่งเปรียบเสมือนการร่างพิมพ์เขียวสำหรับโครงสร้างข้อมูล
- ในบทนี้ เราจะเปลี่ยนพิมพ์เขียวนั้นให้กลายเป็นโครงสร้างที่จับต้องได้ด้วย SQL (Structured Query Language) ซึ่งเป็นภาษามาตรฐานสำหรับการสร้างและจัดการฐานข้อมูลเชิงสัมพันธ์
- เราจะใช้แบบจำลองข้อมูล (ERD) นี้เป็น ‘พิมพ์เขียว’ หลักในการเรียนรู้ของเรา



เครื่องมือสองประเภทหลักของ SQL

ภาษา SQL แบ่งคำสั่งออกเป็น 2 กลุ่มหลักตามฟังก์ชันการทำงาน:



DDL (Data Definition Language): เครื่องมือของสถาปนิก

กลุ่มคำสั่งสำหรับ 'สร้าง' และ 'กำหนด' โครงสร้างของฐานข้อมูล เช่น การสร้างตาราง (Tables), ดัชนี (Indexes), และมุมมอง (Views)

COMMAND OR OPTION	DESCRIPTION
CREATE TABLE	Creates a new table in the user's database schema
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
CREATE INDEX	Creates an index for a table
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
DROP TABLE	Permanently deletes a table (and its data)
DROP VIEW	Permanently deletes a view
DROP INDEX	Permanently deletes an index

DML (Data Manipulation Language): เครื่องมือของผู้จัดการข้อมูล

กลุ่มคำสั่งสำหรับ 'จัดการ' ข้อมูลภายในตาราง เช่น การเพิ่ม (Insert), แก้ไข (Update), ลบ (Delete), และสืบค้น (Select) ข้อมูล

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
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

ขั้นที่ 1: การวางรากฐานด้วย `CREATE TABLE`

คำสั่งแรกและสำคัญที่สุดในการสร้างโครงสร้างคือ `CREATE TABLE´ เพื่อกำหนดตารางและคอลัมน์ (หรือแอทริบิว) ที่จะใช้เก็บข้อมูล เราจะใชข้อมูล เราจะเริ่มต้นด้วยการสร้างตาราง `VENDOR´ จากพิมพ์เขียวของเรา

```
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)
);
```

- กำหนดชนิดข้อมูลเป็นตัวเลขจำนวนเต็ม
- กำหนดชนิดข้อมูลเป็นข้อความที่ความยาวไม่เกิน 35 ตัวอักษร
- บังคับว่าคอลัมน์นี้ต้องมีข้อมูลเสมอ ห้ามเป็นค่าว่าง
- กำหนดให้ข้อมูลในคอลัมน์นี้ต้องไม่ซ้ำกัน
- กำหนดให้ `V_CODE` เป็นคีย์หลักของตาราง เพื่อรับประกัน Entity Integrity

การกำหนด 'กฎ' ของโครงสร้าง: Constraints

Constraints คือเงื่อนไขที่แใช้บังคับความถูกต้องและความสัมพันธ์ของข้อมูลในตาราง
เพื่อรักษาคุณสมบัติ Entity Integrity และ Referential Integrity

1. Foreign Key (คีย์นอก): สร้างความเชื่อมโยงระหว่างตาราง

```
-- จากตาราง PRODUCT
FOREIGN KEY (V_CODE) REFERENCES VENDOR
```

- **คำอธิบาย**: ทำให้แน่ใจว่า `V_CODE` ทุกค่าในตาราง `PRODUCT` จะต้องมีอยู่จริงในตาราง `VENDOR`

2. CHECK: ตรวจสอบความถูกต้องของข้อมูล

```
-- จากตาราง CUSTOMER
CHECK(CUS_AREACODE IN('615', '713', '931'))
```

- **คำอธิบาย**: อนุญาตให้กรอกข้อมูล `CUS_AREACODE` ได้เพียง 3 ค่านี้เท่านั้น

3. DEFAULT: กำหนดค่าเริ่มต้น

```
-- จากตาราง CUSTOMER
CUS_BALANCE NUMBER(9,2) DEFAULT 0.00
```

- **คำอธิบาย**: หากไม่มีการระบุค่า `CUS_BALANCE` ระบบจะกำหนดค่าเป็น `0.00` โดยอัตโนมัติ

การเลือก 'วัสดุ' ในการสร้าง: ชนิดของข้อมูล (Data Types)

การเลือกชนิดข้อมูลที่เหมาะสมเป็นสิ่งสำคัญ เพราะส่งผลต่อการจัดเก็บ การคำนวณ และประสิทธิภาพ

ข้อมูลตัวเลข (Numeric)

- NUMBER(L,D) ตัวเลขทศนิยม (เช่น `P_PRICE NUMBER(8,2)`)
- INTEGER, SMALLINT จำนวนเต็ม (เช่น `V_CODE INTEGER`)

ข้อมูลข้อความ (Character)

- CHAR(L) ข้อความความยาวคงที่ เหมาะกับข้อมูลที่ความยาวเท่ากันเสมอ (เช่น `V_STATE CHAR(2)`)
- VARCHAR(L) ข้อความความยาวผันแปรได้ ประหยัดพื้นที่กว่า (เช่น `V_NAME VARCHAR(35)`)

ข้อมูลวันที่ (Date)

- DATE จัดเก็บวันที่ในรูปแบบ Julian เพื่อให้ง่ายต่อการคำนวณ (เช่น `P_INDATE DATE`)

ATTRIBUTE NAME	TYPE	CONTENTS
P_CODE	CHAR(10)	Product code
P_DESCRIPT	VARCHAR(35)	Product description
P_INDATE	DATE	Stocking date
P_QOH	SMALLINT	Units available
P_MIN	SMALLINT	Minimum units
P_PRICE	NUMBER(8,2)	Product price
P_DISCOUNT	NUMBER(5,2)	Discount rate
V_CODE	INTEGER	Vendor code
V_CODE	INTEGER	Vendor code
V_NAME	CHAR(35)	Vendor name
V_CONTACT	CHAR(25)	Contact person
V_AREACODE	CHAR(3)	Area code
V_PHONE	CHAR(8)	Phone number
V_STATE	CHAR(2)	State
V_ORDER	CHAR(1)	Previous order

การปรับปรุงและรื้อถอน: `ALTER` & `DROP`

หลังจากสร้างตารางแล้ว เรายังสามารถปรับเปลี่ยนโครงสร้างหรือแม้กระทั่งลบทิ้งได้ตามความจำเป็น

ALTER TABLE
ปรับเปลี่ยนโครงสร้างตาราง

การเพิ่มคอลัมน์ (Adding a Column)

```
-- เพิ่มคอลัมน์ P_SALECODE ในตาราง PRODUCT
ALTER TABLE PRODUCT
  ADD (P_SALECODE CHAR(1));
```

การแก้ไขชนิดข้อมูล (Modifying a Column)

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

DROP
ลบองค์ประกอบต่างๆ

การลบตาราง (Dropping a Table)

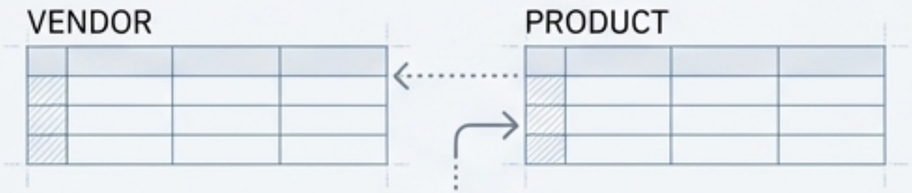
```
-- ลบตาราง PART ทั้งตาราง (ข้อมูลทั้งหมดจะหายไป)
DROP TABLE PART;
```

การลบดัชนี (Dropping an Index)

```
-- ลบดัชนีที่ชื่อว่า PROD_PRICEX
DROP INDEX PROD_PRICEX;
```

ขั้นที่ 2: การบรรจุข้อมูลด้วย `INSERT`

เมื่อมีโครงสร้างตารางแล้ว เราจะเริ่มเพิ่มข้อมูล (แถวข้อมูล) เข้าไปในตารางด้วยคำสั่ง `INSERT`
สิ่งสำคัญคือต้องเพิ่มข้อมูลในตารางหลัก (ที่มี Primary Key) ก่อนตารางที่อ้างอิง (ที่มี Foreign Key) เพื่อรักษา Referential Integrity



1. เพิ่มข้อมูลในตารางหลักก่อน (Insert into Primary Table first)

VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y

```
-- เพิ่มข้อมูลบริษัทผู้ผลิตในตาราง VENDOR
INSERT INTO VENDOR
VALUES (21225, 'Bryson, Inc.', 'Smithson', '615', '223-3234', 'TN', 'Y');
```

2. เพิ่มข้อมูลในตารางที่อ้างอิง (Then, insert into Referencing Table)

PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter...	03-Nov-14	8	5	109.99	0.00	21225

```
-- เพิ่มข้อมูลสินค้าที่อ้างอิง V_CODE = 21225
INSERT INTO PRODUCT
VALUES ('11QER/31', 'Power painter...', '03-Nov-14', 8, 5, 109.99, 0.00, 21225);
```

การเพิ่มค่า NULL (Handling NULL values)

```
-- สำหรับสินค้าที่ผลิตเอง ไม่มี V_CODE
INSERT INTO PRODUCT
VALUES ('BRT-345', 'Titanium drill bit', ..., NULL);
```

การจัดการข้อมูล: `UPDATE`, `DELETE`, `COMMIT`

การจัดการข้อมูลไม่ได้มีแค่การเพิ่ม แต่ยังรวมถึงการแก้ไข ลบ และการยืนยันการเปลี่ยนแปลง

`UPDATE`: แก้ไขข้อมูลที่มีอยู่

```
-- อัปเดตราคาของสินค้า P_CODE = '13-Q2/P2'
UPDATE PRODUCT
SET P_PRICE = 17.99, P_MIN = 10
WHERE P_CODE = '13-Q2/P2';
```

⚠ **ข้อควรระวัง**: หากไม่มี WHERE จะอัปเดตทุกแถว!

`DELETE`: ลบแถวข้อมูล

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

`COMMIT` & `ROLLBACK`: บันทึกหรือยกเลิก

`COMMIT;` -- บันทึกการเปลี่ยนแปลงทั้งหมด (เพิ่ม, แก้ไข, ลบ) ลงดิสก์อย่างถาวร

`ROLLBACK;` -- ยกเลิกการเปลี่ยนแปลงทั้งหมดที่ยังไม่ได้ COMMIT

ขั้นที่ 3: การสืบค้นข้อมูลด้วย `SELECT`

หัวใจของ SQL คือการสืบค้น (Query) เพื่อดึงข้อมูลที่เราต้องการออกมาเป็นสารสนเทศ
คำสั่งพื้นฐานที่สุดคือ `SELECT` ซึ่งใช้ในการเลือกคอลัมน์จากตารางที่ต้องการ

รูปแบบพื้นฐาน:

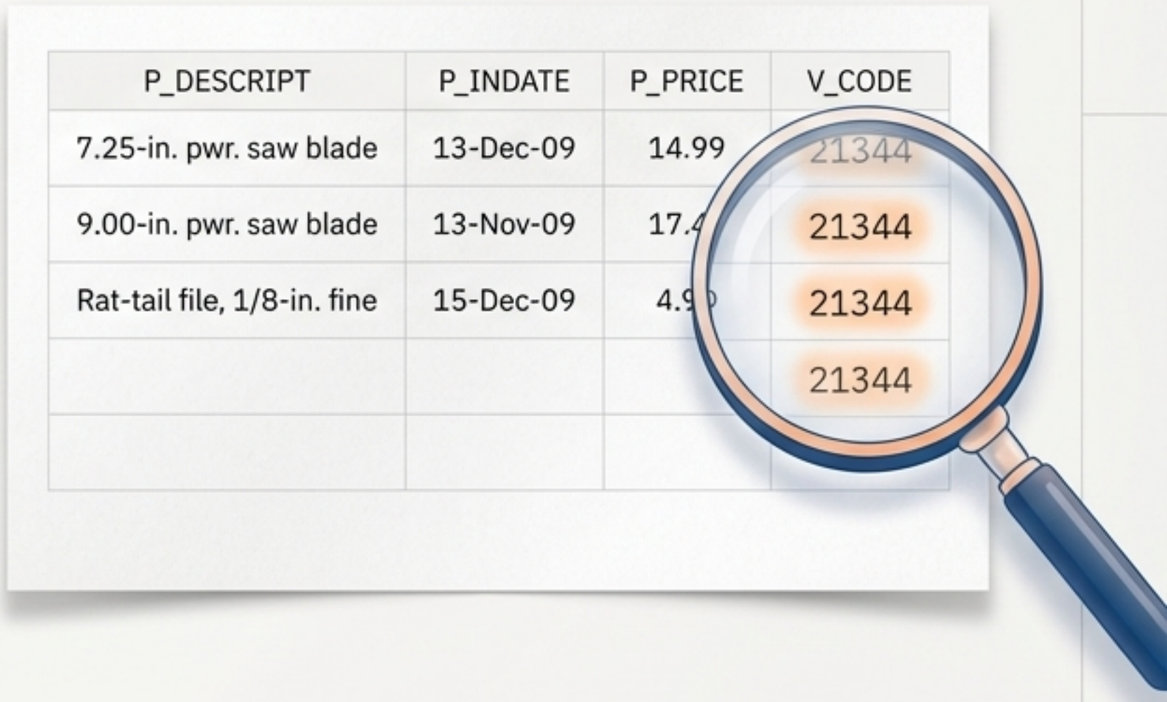
```
SELECT columnlist FROM tablename;
```

ดึงข้อมูลทุกคอลัมน์จากตาราง PRODUCT (Using wildcard `*`)

```
SELECT * FROM PRODUCT;
```

ดึงข้อมูลเฉพาะบางคอลัมน์ พร้อมเงื่อนไข `WHERE`

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



การคัดกรองข้อมูล: เจาะจงด้วย Operators

เราสามารถสร้างเงื่อนไขที่ซับซ้อนใน `WHERE` clause ได้โดยใช้ Operators ประเภทต่างๆ

Logical Operators (`AND`, `OR`, `NOT`)

```
-- สินค้าที่ราคา < 50 และ รับเข้าคลังหลัง 15-Jan-2010
WHERE P_PRICE < 50 AND P_INDATE > '15-Jan-2010'
```

Special Operators (`BETWEEN`, `IS NULL`, `LIKE`, `IN`)

```
WHERE P_PRICE BETWEEN 50.00 AND 100.00;
```
(ราคาระหว่าง 50 ถึง 100)

```
WHERE V_CODE IS NULL;
```
(สินค้าที่ไม่มีรหัสผู้ผลิต)

```
WHERE V_CONTACT LIKE 'Smith%';
```
(ผู้ติดต่อที่ขึ้นต้นด้วย 'Smith')

```
WHERE V_CODE IN (21344, 24288);
```
(สินค้าจากผู้ผลิตรหัส 21344 หรือ 24288)

การสรุปผลและวิเคราะห์: Aggregate Functions & `GROUP BY`

SQL ไม่เพียงแต่ดึงข้อมูลดิบ แต่ยังสามารถคำนวณและสรุปผลข้อมูลในภาพรวมได้

ฟังก์ชันการคำนวณ (Aggregate Functions)

`COUNT(): "นับจำนวนแถว"

`SUM(): "หาผลรวม"

`AVG(): "หาค่าเฉลี่ย"

`MAX()`/ `MIN()`: “หาค่าสูงสุด / ต่ำสุด”

ตัวอย่างการใช้งาน

นับจำนวนสินค้าทั้งหมด

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

หาราคาสินค้าสูงสุด

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

การจัดกลุ่มข้อมูลด้วย `GROUP BY`

ใช้ร่วมกับ Aggregate Functions เพื่อคำนวณค่าสรุปสำหรับข้อมูลแต่ละกลุ่ม

V_CODE	P_CODE
21344	P01
21344	P02
24288	P03
24288	P04
24288	P05

GROUP BY V_CODE

V_CODE	NUMBER_OF_PRODUCTS
21344	2
24288	3

```
-- นับจำนวนสินค้าในแต่ละ V_CODE (ผู้ผลิตแต่ละรายมีสินค้าที่ชนิด)
SELECT V_CODE, COUNT(P_CODE) AS NUMBER_OF_PRODUCTS
FROM PRODUCT
GROUP BY V_CODE;
```

การจัดระเบียบผลลัพธ์: `ORDER BY` & `DISTINCT`

เพื่อให้ผลลัพธ์จากการ Query อ่านง่ายและตรงตามความต้องการ
เราสามารถจัดเรียงลำดับและกำจัดข้อมูลที่ซ้ำซ้อนได้

`ORDER BY` (การจัดเรียงข้อมูล)

จัดเรียงแถวข้อมูลของผลลัพธ์ตามคอลัมน์ที่กำหนด

Syntax:

- ASC: เรียงจากน้อยไปมาก (default)
- DESC: เรียงจากมากไปน้อย


```sql
-- แสดงรายการสินค้า โดยเรียงตามราคาจากมากไปน้อย
SELECT P_DESCRIPT, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE DESC;
```

`DISTINCT` (การแสดงข้อมูลที่ไม่ซ้ำกัน)

แสดงเฉพาะค่าที่ไม่ซ้ำกันในคอลัมน์ที่เลือก


```sql
-- แสดงรหัสผู้ผลิต (V_CODE) ทั้งหมดที่มีในตาราง PRODUCT แบบไม่ซ้ำ
SELECT DISTINCT V_CODE FROM PRODUCT;
```

การสร้าง 'มุมมอง': `CREATE VIEW`

- View คือ ‘ตารางเสมือน’ (Virtual Table) ที่สร้างขึ้นจากผลลัพธ์ของคำสั่ง `SELECT`
- มันช่วยให้เราสามารถบันทึก Query ที่ซับซ้อนไว้ในชื่อที่เรียกใช้ง่าย เสมือนการสร้าง ‘หน้าต่าง’ เพื่อมองข้อมูลในมุมที่เราสนใจโดยเฉพาะ

1. สร้าง View

```
-- สร้าง View ชื่อ PRICEGT50 สำหรับสินค้าที่ราคาสูงกว่า 50
CREATE VIEW PRICEGT50 AS
  SELECT P_DESCRIPT, P_QOH, P_PRICE
  FROM PRODUCT
 WHERE P_PRICE > 50.00;
```

2. เรียกใช้งาน View (เหมือนเป็นตารางปกติ)

```
-- ดึงข้อมูลทั้งหมดจาก View ที่สร้างขึ้น
SELECT * FROM PRICEGT50;
```

ประโยชน์:

- ลดความซับซ้อนในการเขียน Query
- ควบคุมการเข้าถึงข้อมูล (ให้ผู้ใช้เห็นข้อมูลแค่บางส่วนผ่าน View)

การเชื่อมโยงทุกส่วนเข้าด้วยกัน: `JOIN`

พลังที่แท้จริงของฐานข้อมูลเชิงสัมพันธ์คือการเชื่อมโยงข้อมูลจากหลายตารางเข้าด้วยกัน คำสั่ง `JOIN` ทำหน้าที่นี้โดยการจับคู่แถวข้อมูลจากตารางต่างๆ ผ่าน Foreign Key

ตัวอย่าง `INNER JOIN`

เป้าหมาย: แสดง 'ชื่อสินค้า' และ 'ราคา' (จากตาราง `PRODUCT`) คู่กับ 'ชื่อบริษัทผู้ผลิต' (จากตาราง `VENDOR`)

```
SELECT
    P.P_DESCRIPT,
    P.P_PRICE,
    V.V_NAME
FROM PRODUCT P
JOIN VENDOR V ON P.V_CODE = V.V_CODE;
```

- `FROM PRODUCT P JOIN VENDOR V`: ระบุว่าจะเชื่อม 2 ตาราง โดยใช้ชื่อย่อ `P` และ `V`
- `ON P.V_CODE = V.V_CODE` คือเงื่อนไขการเชื่อมโยง โดยใช้คอลัมน์ `V_CODE` ที่มีร่วมกัน