

คำสั่งในการกำหนดประเภทของข้อมูล

**(Data Definition Language  
Command : DDL)**

# **(Data Definition Language Command : DDL)**

- **CREATE TABLE**
- **FORMAT:**

CREATE TABLE table\_name

( column\_name DATATYPE COLUMN\_CONSTRAINT,

column\_name DATATYPE COLUMN\_CONSTRAINT,

...

TABLE\_CONSTRAINT

);

# (Data Definition Language Command : DDL)

คำสั่ง	ความหมาย
CREATE TABLE	คำสั่งให้สร้างตาราง
table_name	กำหนดชื่อตาราง
column_name	ชื่อคอลัมน์แต่ละคอลัมน์
DATATYPE	ชนิดข้อมูลของคอลัมน์นั้น ๆ
COLUMN_CONSTRAINT	ชื่อของข้อกำหนดที่ต้องการสร้างให้กับคอลัมน์
TABLE_CONSTRAINT	ชื่อของข้อกำหนดที่ต้องการสร้างให้กับตาราง

# ประเภทของข้อมูลใน Oracle

ประเภทของข้อมูล	ความหมาย
CHAR(size)	ตัวอักษรที่มีความยาวของข้อมูลตามค่า size ค่าสูงสุดคือ 2,000 ไบต์
VARCHAR2(size)	ตัวอักษรที่มีความยาวผันเปลี่ยนไปตามความยาวของข้อมูล ค่าสูงสุดคือ 4,000 ไบต์
NCHAR(size)	ตัวอักษรเช่นเดียวกับ VARCHAR2 แต่จะเก็บข้อมูลตามค่าใน National Character Set

# ประเภทของข้อมูลใน Oracle

ประเภทของข้อมูล	ความหมาย
LONG	ตัวอักษรที่มีการกำหนดความยาวที่แน่นอนเอาไว้ มีความยาวสูงสุด 2 กิกะไบต์
NUMBER(p,s)	ตัวเลข ซึ่งแสดงจำนวนหลักด้วย <b>P</b> และจำนวนหลังทศนิยมด้วย <b>S</b> โดยที่  <b>P</b> จะมีค่าอยู่ระหว่าง 1 ถึง 38  <b>S</b> จะมีค่าอยู่ระหว่าง —84 ถึง 27

# ประเภทของข้อมูลใน Oracle

ประเภทของข้อมูล	ความหมาย
DATE	วันที่ และเวลา มีค่าได้ตั้งแต่ 1 มกราคม 4712 ก่อนคริสต์ศักราช ถึง 31 ธันวาคม 4712 หลังคริสต์ศักราช
RAW(size)	ข้อมูลไบนาเรียที่กำหนดขนาดที่แน่นอนเอาไว้ และมีความยาวสูงสุด 255 ไบต์
LONG RAW	ข้อมูลไบนาเรียที่สามารถปรับขนาดได้ และมีความยาวสูงสุด 2 กิกะไบต์

# ประเภทของข้อมูลใน Oracle

ประเภทของข้อมูล	ความหมาย
ROW ID	เลขฐานสองที่แสดงถึงตำแหน่งที่อยู่ในเรคอร์ด
CLOB	จัดเก็บข้อมูลตัวอักษรที่เป็นแบบ Large Object มีความยาวสูงสุด 4 GB
BLOB	จัดเก็บข้อมูลในรูปแบบ Large Object มีความยาวสูงสุด 4 GB

# ประเภทของข้อมูลใน Oracle

ประเภทของข้อมูล	ความหมาย
NCLOB	จัดเก็บข้อมูลตัวอักษรหลายชุดที่เป็นแบบ Large Object มีความยาวสูงสุด 4 GB
BFILE	จัดเก็บลิงค์ (pointer) ที่ใช้ในการเรียกไฟล์ที่จัดเก็บเป็น OS file นอกฐานข้อมูล

# (Data Definition Language Command : DDL)

- ตัวอย่าง 1 : การสร้างตารางแผนก ชื่อ **department** โดยจะประกอบไปด้วยข้อมูล รหัสแผนก, ชื่อแผนก

```
CREATE TABLE department(  
    D_No      CHAR(3) NOT NULL,  
    D_Name    VARCHAR2(50),  
    CONSTRAINT Dept_Pk PRIMARY KEY (D_No)  
);
```

# (Data Definition Language Command : DDL)

- ตัวอย่าง 2 : การสร้างตารางพนักงาน ชื่อ **employee** โดยจะประกอบไปด้วยรหัสพนักงาน, ชื่อ, สกุล, รหัสแผนก, ตำแหน่ง และเงินเดือน

```
CREATE TABLE employee(  
    E_No          CHAR(4) not null,  
    E_Name        VARCHAR2(30),  
    E_Last        VARCHAR2(30),  
    D_No          CHAR(3) DEFAULT 'D01',  
    Position      VARCHAR2(20),  
    Salary         NUMBER(9,2),  
    CONSTRAINT Emp_Pk PRIMARY KEY (E_No),  
    CONSTRAINT emp_Fk FOREIGN KEY (D_No)  
    REFERENCES department (D_NO)  
);
```

- R1(K1,X,Y,Z)
- R2(K2, A, B, C, **K1**)

CONSTRAINT *R2\_FK* FOREIGN KEY(**K1**)  
*REFERENCES R1(K1)*

CONSTRAINT Dept\_Fk FOREIGN KEY (D\_No)  
REFERENCES department (D\_NO));

- R1(K1,F1, X, Y, Z)
- R2 (K2, A, B, C, **K1,F1**)
- CONSTRAINT **R1\_PK** PRIMARY KEY (**K1,F1**)
- CONSTRAINT **R2\_FK** FOREIGN KEY (**K1,F1**) REFERENCES **R1 (K1,F1)**;

# (Data Definition Language Command : DDL)

- กี๊ย์หลัก (Primary Key) คือแอตทริบิวต์ที่น้อยที่สุดที่จะใช้ในการแยกตาราง โดยอาจจะมีเพียง colum น์เดียวหรือหลาย colum น์ก็ได้ เช่น ในตาราง department จะมีกี๊ย์หลักคือ D\_No
- กี๊ร่อง(Foreign Key) คือกลุ่มของแอตทริบิวต์ที่มีน้อยที่สุดที่ใช้ในการอ้างถึงกี๊ย์หลักในตารางแม่ เช่น ในตาราง employee จะมี กี๊ร่องคือ D\_No ซึ่งจะเป็นกี๊ที่ใช้ในการอ้างถึง colum น์ D\_No ในตาราง department ซึ่งจริงๆแล้วในตาราง employee ไม่จำเป็นต้องใช้ชื่อ D\_No เหมือนในตาราง department ก็ได้

# (Data Definition Language Command : DDL)

- UNIQUE คือ ต้องมีเพียงตัวเดียวในคอลัมน์
- DEFAULT คือ การกำหนดค่าให้อัตโนมัติ ในการณ์ที่ไม่มีการส่งค่าได ๆ ลงมา
- CHECK คือ การกำหนดเงื่อนไขข้อมูลที่จะใส่เข้ามาจะต้องเป็นไปตามเงื่อนไขเท่านั้น
- NOT NULL คือ การกำหนดห้ามไม่ให้ปล่อยค่าว่างในคอลัมน์

# (Data Definition Language Command : DDL)

## ■ การสร้าง Primary Key

**Format**

**CONSTRAINT constraint\_name**

**PRIMARY KEY(column1[,column2]);**

# (Data Definition Language Command : DDL)

## ■ การสร้าง Primary Key

คำสั่ง	ความหมาย
<b>CONSTRAINT constraint_name</b>	คำสั่งใช้สร้าง <b>CONSTRAINT</b> โดย <b>constraint_name</b> คือชื่อของ <b>CONSTRAINT</b> ที่ต้องการ
<b>PRIMARY KEY (column1[,column2])</b>	กำหนดประเภทของ <b>CONSTRAINT</b> เป็น <b>PRIMARY KEY</b> และกำหนดคอลัมน์ที่ต้องการสร้างให้เป็น <b>PRIMARY KEY</b>

# (Data Definition Language Command : DDL)

## ■ การสร้าง Foreign Key

**Format**

**CONSTRAINT *constraint\_name***

**FOREIGN KEY(*column1, column2, ...*)**

**REFERENCES**

***table\_name(column1, column2, ...);***

# (Data Definition Language Command : DDL)

## ■ การสร้าง Foreign Key

คำสั่ง	ความหมาย
<b>CONSTRAINT constraint_name</b>	คำสั่งใช้สร้าง <b>CONSTRAINT</b> โดย <b>constraint_name</b> คือชื่อของ <b>CONSTRAINT</b> ที่ต้องการ
<b>FOREIGN KEY(column1,column2,...)</b>	กำหนดประเภทของ <b>CONSTRAINT</b> เป็น <b>FOREIGN KEY</b> และกำหนดคอลัมน์ที่ต้องการสร้างให้เป็น <b>FOREIGN KEY</b>

# (Data Definition Language Command : DDL)

## ■ การสร้าง Foreign Key (ต่อ)

คำสั่ง	ความหมาย
<b>REFERENCES</b> <i>table_name</i> <b>(column1,</b> <b>column2,...)</b>	กำหนดชื่อตารางและ colum ของ ตารางที่ต้องการอ้างอิงถึง

# (Data Definition Language Command : DDL)

## ■ ALTER

เป็นคำสั่งที่ใช้เพื่อเปลี่ยนแปลงสิ่งที่เราได้ทำการ CREATE ไปแล้ว ส่วนมากใช้ในการแก้ไขโครงสร้างของตารางเดิมที่มีอยู่ โดยมีรูปแบบคำสั่ง ดังนี้

```
ALTER TABLE table_name  
[ADD column_name DATATYPE  
COLUMN_CONSTRAINT]  
[MODIFY column_name DATATYPE  
COLUMN_CONSTRAINT]  
[DROP CONSTRAINT constraint_name [CASCADE]];
```

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 1 ต้องการเพิ่ม colum นี้ 2 colum นี้ในตาราง **department** จะใช้คำสั่งดังนี้

```
ALTER TABLE department
Add (
    Test_Add    NUMBER(1) not null,
    Test_Add2   VARCHAR2 (2)
);
```

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 2 ต้องการเปลี่ยน data type ของ colum นี้  
**Test\_Add2** จาก VARCHAR2 เป็น NUMBER

```
ALTER TABLE department  
MODIFY ( Test_Add2 NUMBER(2));
```

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 3 ต้องการลบคอลัมน์ Test\_Add2 ออกจากตาราง

```
ALTER TABLE department  
DROP ( Test_Add2);
```

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 4 ต้องการเพิ่ม PRIMARY KEY ให้กับคอลัมน์ E\_Name ในตาราง employee

**Format**

ALTER TABLE table\_name ADD CONSTRAINT  
ตัวแปร PRIMARY KEY (fieldname);

**ตัวอย่าง**

ALTER TABLE employee ADD CONSTRAINT  
enam\_fk PRIMARY KEY (E\_Name);

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 5 ต้องการเพิ่ม FOREIGN KEY ให้กับคอลัมน์ E\_Name ในตาราง employee

**Format**

ALTER TABLE table\_name ADD  
CONSTRAINT

ตัวแปร FOREIGN KEY (fieldname)  
REFERENCES table\_name1(fieldname);

**ตัวอย่าง**

ALTER TABLE employee ADD CONSTRAINT  
dnam\_fk FOREIGN KEY (E\_Name)  
REFERENCES department(D\_Name);

# (Data Definition Language Command : DDL)

- ตัวอย่างที่ 6 ต้องการเปลี่ยนชื่อคอลัมน์ E\_Last ให้เป็น E\_Lastname ในตาราง employee

**Format**

**ALTER TABLE *table\_name* RENAME *ชื่อคอลัมน์เดิม* TO *ชื่อคอลัมน์ใหม่*;**

**ตัวอย่าง**

**ALTER TABLE employee RENAME E\_Last  
TO E\_Lastname;**

# (Data Definition Language Command : DDL)

## ■ RENAME

เป็นคำสั่งเปลี่ยนชื่อตาราง

Format

**RENAME** ชื่อตารางเดิม **TO** ชื่อตารางใหม่;

## ■ DROP

เป็นคำสั่งลบตาราง

Format

**DROP TABLE** ชื่อตาราง;

คำสั่งในการควบคุมโครงสร้างของข้อมูล

**(Data Manipulation Language  
Command : DML)**

# (Data Manipulation Language Command : DML)

## ■ **INSERT**

คือการเพิ่มข้อมูลให้กับตารางโดยที่คำสั่งนี้จะสามารถเพิ่มได้ครั้งละ 1 record ถ้าหากต้องการเพิ่มข้อมูลหลาย record ก็ให้ทำซ้ำหลาย ๆ ครั้ง โดยมีรูปแบบคำสั่งดังนี้

## ■ **FORMAT**

```
INSERT INTO table_name (Column_1,  
Column2,..., Column_n)  
VALUES (Value_1, Value_2, Value_3);
```

# (Data Manipulation Language Command : DML)

ข้อมูลในตาราง department มีดังนี้

D_No	D_Name
D01	Sales
Do2	support
D03	Finance

■ ตัวอย่างการใช้คำสั่ง INSERT เพื่อเพิ่มข้อมูลแถวที่ 1

```
INSERT INTO department (D_NO, D_NAME)  
VALUES ('D01','Sales');
```

# (Data Manipulation Language Command : DML)

## ■ **INSERT** (ต่อ)

เมื่อใส่ข้อมูลแล้วอาจจะใช้คำสั่ง COMMIT เพื่อทำการยืนยันข้อมูลที่ใส่เข้าไปทั้งหมด ซึ่งโดยปกติแล้วเวลาที่ออกจากโปรแกรมจะมีการ confirm อยู่แล้วแต่เราใส่เพื่อจะได้มั่นใจแน่ๆว่าสิ่งที่เราได้ทำมานั้นได้เก็บลงไปในฐานข้อมูล แล้วเพื่อเกิดความผิดพลาดอะไรกับระบบขึ้นมาเราจะได้สามารถถูกลบข้อมูลเหล่านี้กลับมาได้

# (Data Manipulation Language Command : DML)

ข้อมูลในตาราง employee มีดังนี้

E_NO	E_NAME	E_LAST	D_NO	POSITION	SALARY
E001	Songkran	Thongsawang	D02	Programmer	18000
E002	Rujiya	Chartsakul	D01	Manager	35000
E003	Oranuch	Penkawin	D01	Sales	15000
E004	Jitlada	Kamalawat	D02	System-Analyst	25000
E005	Pikul	Monthon	D03	Co-ordinator	12000
E006	Satit	Boonyarat	D02	Manager	37000
E007	Sirinate	Rakwong	D03	Manager	32000

# (Data Manipulation Language Command : DML)

## ■ UPDATE

คือ การอ่านข้อมูล (read) ออกมายจากตารางแล้ว ทำการเขียน (write) ลงไปในตารางนั้นใหม่ โดยจะทำงานในลักษณะของ group of record ทำให้การอัพเดตใช้เวลาค่อนข้างนาน เพราะจะต้องทำงานถึง 2 รอบ

## FORMAT

UPDATE table\_name

SET column = value [,column = value]

WHERE <condition>

;

# (Data Manipulation Language Command : DML)

- **UPDATE** (ต่อ)
- โดยที่เงื่อนไข (**condition**) ที่สามารถใช้ในที่นี้ได้จะต้องมีรูปแบบดังตาราง

เงื่อนไข	ผลที่ได้เป็นจริงก็ต่อเมื่อ
$A = B$	$A$ เท่ากับ $B$
$A \neq B, A <> B$	$A$ ไม่เท่ากับ $B$
$A > B$	$A$ มากกว่า $B$
$A \geq B$	$A$ มากกว่าหรือเท่ากับ $B$
$A < B$	$A$ น้อยกว่า $B$
$A \leq B$	$A$ น้อยกว่าหรือเท่ากับ $B$
$A \text{ IN}(List\_Item\_1, \dots, List\_Item\_n)$	$A$ มีค่าเท่ากับสมาชิกที่กำหนดใน List เช่น $A \text{ IN}('test1','test2')$ หมายถึง $A$ เป็น $test1$ หรือ $test2$ ก็ได้
$A \text{ BETWEEN } X \text{ AND } Y$	ค่าของ $A$ จะอยู่ระหว่างค่า $X$ และค่า $Y$
$A \text{ LIKE } Y$	$A$ จะมีรูปแบบเหมือนกับ $Y$
$A \text{ IS NULL}$	ค่าของ $A$ ไม่ปรากฏ หรือมีค่าว่าง(null)
$\text{NOT}$	จะทำให้เงื่อนไขที่กำหนดทำงานตรงข้ามกับเงื่อนไขเดิม

# (Data Manipulation Language Command : DML)

## ■ ตัวอย่าง

การเปลี่ยน D\_Name จากชื่อ “Finance” ให้เป็น  
“Accounting”

UPDATE department

```
SET D_Name = 'Accounting'  
WHERE D_No = 'D03'  
;
```

# (Data Manipulation Language Command : DML)

## ■ **DELETE**

คือ การลบข้อมูลออกจากตาราง

## **FORMAT**

**DELETE FROM table\_name**

**WHERE <condition>**

**;**

โดยที่ <condition> ในที่นี้จะเป็นแบบเดียวกับที่ได้แสดงไว้ในเรื่อง  
ของการ “**UPDATE**” ที่ได้กล่าวไปแล้ว

# (Data Manipulation Language Command : DML)

## ■ ตัวอย่าง

การลบข้อมูลใน rekord ที่เก็บข้อมูล ‘E005’ จากตาราง employee

DELETE FROM employee

WHERE E\_No = 'E005'

;

คำสั่งในการค้นหาข้อมูล

# (Data Retrieval Command)

# (Data Retrieval Command)

## ■ SELECT

เป็นคำสั่งที่ใช้ในการค้นหาข้อมูลจากฐานข้อมูล โดยโครงสร้างของคำสั่ง SELECT ประกอบด้วย SELECT, FROM, WHERE, ORDER BY และ GROUP BY

รูปแบบคำสั่งอย่างน้อยที่สุด

SELECT <cluase>  
FROM <cluase>;

ตัวอย่าง

SELECT D\_Name FROM department;

# (Data Retrieval Command)

## ■ Where Clause

จะช่วยให้เราสามารถเลือกเฉพาะเรคอร์ดที่เราต้องการเท่านั้น

**FORMAT :**

SELECT <cluase>

FROM <cluase>

WHERE <condition>

;

โดยที่เงื่อนไข <condition> ในที่นี้จะเป็นแบบเดียวกับที่ได้แสดงไว้ใน  
เรื่องของการ “**UPDATE**” ที่ได้กล่าวไปแล้ว

# (Data Retrieval Command)

## ■ ตัวอย่าง

การเลือกแสดงข้อมูลเฉพาะนักศึกษารหัส '485020147-6' ในตาราง **STUDENT**

```
SELECT *
FROM STUDENT
WHERE ST_CODE = '485020147-6'
;
```

# (Data Retrieval Command)

## ■ Boolean Operation

Operation ในที่นี้จะประกอบไปด้วยเครื่องหมาย OR และ AND โดยมีสถานการณ์ของแต่ละเหตุการณ์ ซึ่งมีดังนี้

A	B	A AND B	A OR B
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

# (Data Retrieval Command)

## ■ เครื่องหมายในการเปรียบเทียบ (comparison operator)

เครื่องหมาย	ความหมาย
=	เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่า หรือเท่ากับ
<=	น้อยกว่า หรือเท่ากับ
<> หรือ !=	ไม่เท่ากัน

## ■ เครื่องหมายในการเปรียบเทียบอื่น ๆ

เครื่องหมาย	ความหมาย
LIKE	ทดสอบเปรียบเทียบข้อความ
NOT LIKE	หาค่าที่อยู่นอกเหนือจาก LIKE
NULL	ทดสอบเปรียบเทียบค่าว่าง
NOT NULL	ทดสอบเปรียบเทียบค่าที่ไม่ใช่ค่าว่าง

# (Data Retrieval Command)

## ■ ตัวอย่างที่ 1

การเลือกแสดงทุกข้อมูลเฉพาะนักศึกษาสังกัดคณะวิทยาศาสตร์ ('F01') และ มีอาจารย์รหัส '001' เป็นอาจารย์ที่ปรึกษา ในตาราง STUDENT

```
SELECT *
FROM STUDENT
WHERE FAC_CODE = 'F01'
AND T_CODE = '001'
;
```

# (Data Retrieval Command)

## ■ ตัวอย่างที่ 2

ต้องการค้นหาว่าในร่มีผลการเรียน 3.50 ถึง 4.00 ในตาราง  
REGIST

```
SELECT *
FROM REGIST
WHERE GRADE BETWEEN 3.50 AND 4.00
;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## คำสั่ง SELECT

ใช้ในการเรียกข้อมูลจากฐานข้อมูลตามที่ผู้ใช้ต้องการ  
รูปแบบ

SELECT [\* | DISTINCT]<TARGET LIST>

FROM <TABLE NAME>[<ALIASES>]

[WHERE PREDICATE]

[GROUP BY <COLUMN LIST>] [HAVING PREDICATE]

[ORDER BY <COLUMN LIST>] [ASC | DESC];

# คำสั่ง SQL สำหรับจัดการข้อมูล

การเรียกดูข้อมูลแบบไม่มีเงื่อนไข

อาจเป็นการดูข้อมูลทั้งตารางหรือบางuetoothที่บิวต์หรือโดยการให้จัดเรียงข้อมูล

เช่น

SELECT \* FROM *DEPARTMENT*;

SELECT *ST\_CODE*, *ST\_NAME* FROM *STUDENT*;

# คำสั่ง SQL สำหรับจัดการข้อมูล

## การเรียกดูข้อมูลแบบมีเงื่อนไข

เป็นการระบุชื่อแทบทรีบิวต์ที่ต้องการระบุเป็นเงื่อนไข และข้อมูลเฉพาะในอนุประโยค WHERE โดยการระบุเงื่อนไขจะนำ operator ต่างๆ เข้ามาใช้ในการแสดงเงื่อนไข มีดังนี้

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ใช้ Logical และ Boolean Operator

Logical Operator ใช้แสดงการเปรียบเทียบค่าของข้อมูล จะแสดงอยู่ระหว่างชื่อคอลัมน์และข้อมูล เนพาะที่ต้องการแสดงเป็นเงื่อนไข Operator ดังกล่าว ประกอบด้วย  $>$ ,  $<$ ,  $=$ ,  $\geq$ ,  $\leq$ ,  $\neq$

Boolean Operator ใช้สำหรับการเรียกคุณข้อมูลที่มีเงื่อนไขมากกว่าหนึ่งเงื่อนไข เช่น AND, OR, NOT

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ต้องการดูทุกข้อมูลเฉพาะนักศึกษาที่มีผลการเรียนมากกว่า 2.50 ในตาราง REGIST

SELECT \* FROM *REGIST*

WHERE *GRADE* > 2.50;

ต้องการดูทุกข้อมูลเฉพาะนักศึกษาที่สังกัดคณะวิทยาศาสตร์ ('F01') หรืออยู่ในภาควิชาวิทยาการคอมพิวเตอร์ ('D03')

SELECT \* FROM *STUDENT*

WHERE *FAC\_CODE* = 'F01'

OR *DEP\_CODE* = 'D03';

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ใช้ Operator ของ SQL

BETWEEN...AND...

ใช้กำหนดเงื่อนไขของแอ็ททริบิวต์เป็นค่าระหว่าง 2 ค่า โดย Operator นี้จะแสดงต่อจากชื่อแอ็ททริบิวต์ที่ถูกกำหนดให้เป็นเงื่อนไข เช่นต้องการดูรหัส นศ. รหัสวิชา และเกรด จากตาราง REGIST เนพานักศึกษาที่มีผลการเรียนระหว่าง 1.50 และ 2.50

SELECT *ST\_CODE,SUB\_CODE, GRADE*

FROM *REGIST*

WHERE *GRADE* BETWEEN *1.50* AND *2.50*;

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ใช้ Operator ของ SQL

IN

ใช้กับเงื่อนไขของคอลัมน์ที่ต้องการระบุเงื่อนไขเป็นกลุ่มของข้อมูล โดย Operator นี้จะแสดงต่อจากชื่อแอთทริบิวต์ที่ถูกกำหนดให้เป็นเงื่อนไข และกลุ่มของข้อมูลที่เป็นข้อมูลเฉพาะของแอთทริบิวต์ที่เป็นเงื่อนไขนี้ จะอยู่ในวงเล็บ() และมี, คั่น

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงรายละเอียดของรหัส นศ. ชื่อ นศ. ที่มีอาจารย์  
ที่ปรึกษาห้อง 001, 002 และ 003 จากตาราง  
STUDENT

```
SELECT ST_CODE,ST_NAME
FROM STUDENT
WHERE T_CODE IN ('001','002','003');
```

# คำสั่ง SQL สำหรับจัดการข้อมูล ใช้ Operator ของ SQL

## LIKE

ใช้ในการค้นหาข้อมูลของคอลัมน์ที่เก็บข้อมูลประเภทตัวอักษรเท่านั้น โดยที่ยังไม่ทราบค่าแน่นอนของข้อมูลทั้งหมดที่จะค้นหา หรือรู้เพียงบางตัวอักษรเท่านั้น โดย Operator นี้จะระบุต่อท้ายชื่อเอoth หรือ Wild Card ที่ถูกกำหนดให้เป็นเงื่อนไข โดยจะใช้สัญลักษณ์ที่ช่วยในการค้นหาข้อมูลเป็นตัวช่วยในการค้นหาข้อมูลที่เรียกว่า “Wild Card”

# คำสั่ง SQL สำหรับจัดการข้อมูล

สัญลักษณ์ดังกล่าวประกอบด้วย %, \_ โดยข้อมูล  
บางส่วนที่ใช้ในการค้นหาพร้อมกับสัญลักษณ์ทั้ง 2 นี้  
จะต้องมีเครื่องหมาย ‘ ’ กำกับเสมอ

% ใช้แทนจำนวนตัวอักษรได้หลายตัว

เช่น WHERE ENAME LIKE ‘N%’

\_ ใช้แทนตัวอักษรที่ไม่ทราบค่าได้ 1 ตัว

เช่น WHERE ENAME LIKE ‘N%\_\_\_\_\_’

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงทุกข้อมูล เนพาะชื่อนักศึกษาที่มีชื่อขึ้นต้น  
ด้วยอักษร ‘น’ ในตาราง STUDENT

SELECT \*

FROM *STUDENT*

WHERE *ST\_NAME* LIKE ‘*n%*’;

# คำสั่ง SQL สำหรับจัดการข้อมูล

## การเรียกคุ้ข้อมูลมากกว่า 1 Relation

หรือ Join อาจจะเรียกคุ้ข้อมูลแบบมีเงื่อนไขหรือไม่มีเงื่อนไขก็ได้

### Equi Join

แต่ละ Relation ที่จะเชื่อมโยงกันจะต้องมีแอ็พทริบิวต์ที่อ้างอิงกันได้ โดยเงื่อนไขที่ระบุจะเปรียบเทียบกันโดยใช้เครื่องหมาย =

# คำสั่ง SQL สำหรับจัดการข้อมูล

การกำหนดชื่อใหม่ให้กับตาราง

SELECT *REGIST.\**, *SUBJECT.\**

FROM *REGIST*, *SUBJECT*

WHERE *REGIST.SUB\_CODE* = *SUBJECT.SUB\_CODE*;

-----

SELECT *R.\**, *S.\**

FROM *REGIST R*, *SUBJECT S*

WHERE *R.SUB\_CODE* = *S.SUB\_CODE*;

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงข้อมูลชื่อ นศ. จากตาราง STUDENT และ  
ชื่ออาจารย์จากตาราง TEACHER เนพะ นศ. และ  
อาจารย์ที่มีที่อยู่เหมือนกัน

```
SELECT S.ST_NAME, T.T_NAME
FROM STUDENT S, TEACHER T
WHERE S.ST_ADDRESS = T.T_ADDRESS;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## Non-Equi Join

เป็นการเชื่อมโยงข้อมูล โดยเงื่อนไขที่แสดงไม่ใช่เครื่องหมาย = แต่อาจเป็น  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $\neq$  หรือ between...and... หรือ in เป็นต้น

เช่น ให้แสดงข้อมูลการลงทะเบียนของนักศึกษาและวิชาเรียน เนพาะวิชาที่ได้ผลการเรียน 3.50 ถึง 4.00

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงข้อมูลการลงทะเบียนของนักศึกษาจากตาราง REGIST และชื่อวิชาจากตาราง SUBJECT เนพารายวิชาที่ได้ผลการเรียน 3.50 ถึง 4.00

```
SELECT R.*, S.SUB_TNAME
FROM REGIST R, SUBJECT S
WHERE GRADE BETWEEN 3.50 AND 4.00
      AND R.SUB_CODE = S.SUB_CODE;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## Self-Join

เป็นการเชื่อมโยงข้อมูลโดยใช้ตารางเดียวกันและตัวเองให้ตารางเป็นชื่ออีกชื่อตารางหนึ่ง ทั้งนี้เพื่อประโยชน์ในการเชื่อมโยงข้อมูล เช่น ให้แสดงรายละเอียดของอาจารย์ที่มีที่อยู่เหมือนกัน

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงรหัส ชื่อ ที่อยู่ของอาจารย์ เนพะอาจารย์ที่มีที่อยู่เดียวกัน

```
SELECT T1.T_CODE,  
T1.T_NAME,T1.T_ADDRESS,T2.T_CODE,  
T2.T_NAME,T2.T_ADDRESS  
FROM TEACHER T1, TEACHER T2  
WHERE T1.T_ADDRESS = T2.T_ADDRESS  
      AND T1.T_CODE < T2.T_CODE;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## Outer-Join

เป็นการเชื่อมโยงข้อมูลในลักษณะที่แสดงข้อมูลที่ไม่เป็นไปตามเงื่อนไขที่กำหนดไว้มาก่อนมาด้วย ทั้งนี้เพื่อประโยชน์ในการดูข้อมูลที่ครบถ้วนมากขึ้น ซึ่งระบบจัดการฐานข้อมูลบางชนิดมีคำสั่ง SQL ที่สามารถทำการเชื่อมโยงเพื่อเรียกดูข้อมูลในลักษณะนี้ได้ เช่น ORACLE หรือ ACCESS เป็นต้น

# คำสั่ง SQL สำหรับจัดการข้อมูล

คำสั่งใน ORACLE ที่ทำการเชื่อมโยงเพื่อเรียกคุณข้อมูลในลักษณะนี้ จะใช้ Operator (+) โดยจะแสดงต่อท้าย colum ที่ใช้เป็นเงื่อนไขในการเชื่อมโยงตารางที่ไม่มีข้อมูล

เช่น ให้แสดงรหัสผู้ขาย ชื่อผู้ขาย รหัสสินค้า จำนวนสินค้าที่ถูกจัดส่ง รหัสและชื่อผู้ขายที่ไม่เคยจัดส่งสินค้าเลย

# คำสั่ง SQL สำหรับจัดการข้อมูล

ให้แสดงรหัส นศ. ชื่อนศ. จากตาราง STUDENT และรหัสอาจารย์ ชื่อ  
อาจารย์ จากตาราง TEACHER เพื่อแสดงรายละเอียดว่า นักศึกษาคน  
ใดมีครุเป็นอาจารย์ที่ปรึกษา พร้อมกับแสดงรายละเอียดอาจารย์ที่  
ไม่เป็นอาจารย์ที่ปรึกษาให้กับ นศ.

```
SELECT S.ST_CODE,S.ST_NAME, T.T_CODE, T.T_NAME  
FROM STUDENT S, TEACHER T  
WHERE S.T_CODE (+) = T.T_CODE;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## การเรียกข้อมูลโดยใช้ฟังก์ชันที่เกี่ยวกับการรวม (Built-in Functions)

เป็นฟังก์ชันที่สามารถประมวลผลกับข้อมูลเป็นชุด(Set) หรือที่เรียกว่า Group function หรือ Aggregate function

ฟังก์ชันที่ใช้ใน SQL ประกอบด้วย AVG, MAX, MIN, SUM, COUNT ฟังก์ชันเหล่านี้สามารถใช้ในคำสั่ง SELECT หรืออนุประโยค HAVING

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ■ AVG

AVG[\*|DISTINCT] <column name> หรือ

AVG[\*|DISTINCT] <column expression>

## ■ COUNT

COUNT[\*|DISTINCT] <column name>

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ■ MAX

MAX[\*|DISTINCT] <column name> หรือ

MAX[\*|DISTINCT] <column expression>

## ■ MIN

MIN[\*|DISTINCT] <column name> หรือ

MIN[\*|DISTINCT] <column expression>

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ■ SUM

SUM[\*|DISTINCT] <column name> หรือ

SUM[\*|DISTINCT] <column expression>

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ■ GROUP BY

การแสดงผลในลักษณะของการจัดกลุ่มข้อมูลยังสามารถใช้ GROUP BY เพื่อสั่งให้จัดกลุ่มตามแอ็ตทริบิวต์ที่ต้องการให้จัดกลุ่ม เนพะลงไป เช่นการให้แสดงยอดรวมของการส่งสินค้าตามรหัสผู้ผลิตรายได้รายหนึ่ง

# คำสั่ง SQL สำหรับจัดการข้อมูล

ในกรณีที่ใช้ GROUP BY การระบุชื่อเขตทริบิวต์ที่จะเรียกข้อมูลออกมาจะต้องเป็นข้อมูลของเขตทริบิวต์ที่ถูกระบุให้จัดกลุ่มในอนุประโยค GROUP BY

การใช้อนุประโยค GROUP BY อาจจะใช้ร่วมกับอนุประโยค HAVING เพื่อให้แสดงข้อมูลที่ได้ผ่านการจัดกลุ่มโดย GROUP BY และแสดงเพียงบางข้อมูลที่เป็นไปตามเงื่อนไขที่ระบุในอนุประโยค HAVING

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงผลรวมหน่วยกิตที่ลงทะเบียนในภาคเรียนที่ 2 ปีการศึกษา 2548 ของนักศึกษารหัส 485020147-6 จากตาราง REGIST

SELECT SUM(*CREDIT*)

FROM *REGIST,SUBJECT*

WHERE *TERM*=‘2’

AND *YEAR* = ‘2548’

AND *ST\_CODE* = ‘485020147-6’

AND *REGIST.SUB\_CODE* = *SUBJECT.SUB\_CODE*;

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ต้องการดูผลรวมหน่วยกิตที่ลงทะเบียนในภาคเรียนที่ 1 ปีการศึกษา 2549 ของนักศึกษาแต่ละรหัส จากตาราง REGIST และตาราง SUBJECT

```
SELECT ST_CODE,SUM(CREDIT)
FROM REGIST, SUBJECT
WHERE TERM = '1'
AND YEAR = '2549'
AND
REGIST.SUB_CODE=SUBJECT.SUB_CODE
GROUP BY ST_CODE;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## ■ HAVING

ในการกำหนดเงื่อนไขในการสืบค้น เราจะระบุในอนุ

ประโยชน์ของ WHERE เป็นเงื่อนไขง่ายๆ ไม่มีการระบุ

ฟังก์ชันกลุ่มในเงื่อนไขนั้น แต่ถ้าหากเราต้องการให้มีการ  
เปรียบเทียบโดยใช้ฟังก์ชันกลุ่ม เราจะใช้ใน WHERE ไม่

ได้ต้องเลี่ยงไปใช้ HAVING แทน

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น รหัสอาจารย์ และชื่ออาจารย์จากตาราง TEACHER เนพะอาจารย์ที่เป็นอาจารย์ที่ปรึกษาให้กับ นศ. มากกว่า 1 คน

```
SELECT T.T_CODE, T.T_NAME
FROM TEACHER T, STUDENT S
WHERE T.T_CODE = S.T_CODE
GROUP BY T.T_CODE, T.T_NAME
HAVING COUNT (*) > 1;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ต้องการแสดงรหัส นศ. ภาคเรียน ปีการศึกษา และรวมหน่วยกิตที่ลงทะเบียน จากตาราง REGIST และ SUBJECT โดยแสดงเฉพาะภาคเรียนที่ลงทะเบียนมากกว่า 15 หน่วยกิต

```
SELECT ST_CODE, TERM, YEAR,
SUM(CREDIT) AS TOTAL
FROM REGIST R, SUBJECT S
WHERE R.SUB_CODE = S.SUB_CODE
GROUP BY ST_CODE, TERM, YEAR
HAVING SUM(CREDIT) > 15;
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## 4.5 การเรียกดูข้อมูลโดยมีคำสั่งลึบคั่นย่อ (Subquery)

เงื่อนไขในอนุประโยค WHERE สามารถใช้คำสั่ง SELECT เป็นคำสั่งลึบคั่นย่อ(Subquery) เพื่อเรียกดูข้อมูลได้ ในการมิทีผลของข้อมูลเรียกจากข้อความย่อymีค่ามากกว่า 1 ค่า ในการระบุเงื่อนไขอาจจะใช้ Operator ต่อไปนี้ในการแสดงเงื่อนไข

# คำสั่ง SQL สำหรับจัดการข้อมูล

## 1. ANY

ใช้ในการเปรียบเทียบค่าของแอ็ตทริบิวต์หนึ่งๆ ว่ามีค่าตรงกับค่าใดค่าหนึ่งของผลลัพธ์แต่ละค่าที่ได้จากคำสั่งสืบค้น ย่อ而言ที่ระบุเป็นเงื่อนไขในอนุประโยค Where หรือไม่ และจะใช้ร่วมกับ Logical Operator เช่น `=, >, <, <>`

# คำสั่ง SQL สำหรับจัดการข้อมูล

## 2. ALL

ใช้ในการเปรียบเทียบค่าของแออททริบิวต์หนึ่งๆ ว่ามีค่าตรงกับทุกค่าของผลลัพธ์ที่ได้จากการคำสั่งสืบค้นอย่างและจะใช้ร่วมกับ Logical Operator เช่น `=, >, <, <>`

# คำสั่ง SQL สำหรับจัดการข้อมูล

## 3. EXISTS

ใช้เพื่อให้แสดงว่าใช่(True) หรือ ไม่ใช่(False) อกมา หากคำสั่ง สืบค้นย่อยในอนุประ โยค Where มีค่าตรงกับค่าที่อ่านได้จาก คำสั่งสืบค้นย่อยของ SELECT ที่อยู่ด้านนอก ถ้าเป็นจริงก็แสดง ข้อมูลอกมา ถ้าไม่จริงก็จะไม่แสดงค่าของข้อมูลนั้น岀มา

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงรหัสผู้ผลิตที่ขายสินค้าหัสรหัส PN002

```
SELECT DISTINCT SNO  
FROM ORDERS  
WHERE SNO IN  
(SELECT SNO  
FROM ORDERS  
WHERE PNO = 'PN002');
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงรหัสผู้ผลิตซึ่อผู้ผลิตที่ไม่ได้ขายสินค้าหัส PN002

```
SELECT SNO,SNAME  
FROM SUPPLIER  
WHERE NOT EXISTS  
(SELECT SNO  
FROM ORDERS  
WHERE ORDERS.SNO = SUPPLIER.SNO  
AND PNO = 'PN002');
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ให้แสดงรหัสผู้ผลิต รหัสสินค้า และจำนวนสินค้าที่ขายของผู้ผลิต  
อื่นที่มีจำนวนการขายสูงกว่ารหัสผู้ผลิต SN004

```
SELECT SNO, PNO, QTY  
FROM ORDERS  
WHERE QTY > ALL  
(SELECT DISTINCT QTY  
FROM ORDERS  
WHERE SNO = 'SN004');
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

การทำคำสั่งย่อຍາຍໄໃຫ້ເຈື່ອນໄຂທີ່ກໍາທັນຄົນນີ້ມີຂໍ້ມູນວ່າ ຄໍາທີ່ສັ່ງ  
ມາເປີຍບ່າຍຈະຕ້ອງມີໜິດຂອງຂໍ້ມູນເປັນໜິດເດືອກກັບຄໍາຂອງ  
ຄອລິ້ມນີ້ທີ່ອູ່ຖານ້າຍຂອງເຄື່ອງໝາຍເປີຍບ່າຍເທື່ບ່າຍ

ດ້າເຄື່ອງໝາຍທີ່ນຳມາເປີຍບ່າຍເປັນເຄື່ອງໝາຍດັ່ງຕ່ອໄປນີ້  
ຄໍາຖານ້າມື້ອຂອງເຄື່ອງໝາຍຈະຕ້ອງເປັນຄໍາເດືອກເສມອ

=, >, <, >=, <= ,<>

# คำสั่ง SQL สำหรับจัดการข้อมูล

แต่ถ้าตัวเปรียบเทียบเป็นคำสั่งต่อไปนี้

IN, NOT IN

ค่าที่ส่งกลับมาจากการคำสั่งสืบค้นย่อยอาจมีหลายค่าได้ แต่ค่าเหล่านี้ยังคงต้องมีชนิดของข้อมูลชนิดเดียวกันกับค่าทางซ้ายของตัวเปรียบเทียบ

# คำสั่ง SQL สำหรับจัดการข้อมูล

นอกจากจะระบุคำสั่งย่อใน WHERE ได้แล้ว เรายังสามารถระบุในส่วนของ FROM ได้ด้วย

เช่น SELECT \*

```
FROM (SELECT SNO  
      FROM SUPPLIER);
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

เช่น ต้องการดูชื่อพนักงานที่มีเงินเดือนมากที่สุดในแต่ละแผนก

```
SELECT DEPTNO, ENAME, SAL
```

```
FROM EMP E
```

```
WHERE SAL = (SELECT MAX(SAL)
```

```
FROM EMP
```

```
WHERE DEPTNO = E.DEPTNO);
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

## 4.6 การเรียกคุ้มข้อมูลโดยใช้ Set Operator

เราสามารถใช้เครื่องหมายในการคำนวณที่ใช้กับเซต เช่น Union, Intersection, Minus มาช่วยในการประมวลผลข้อมูลที่ได้จากคำสั่ง สืบค้นหลายๆ คำสั่งได้ เช่น

# คำสั่ง SQL สำหรับจัดการข้อมูล

```
SELECT ENAME, JOB  
FROM EMP  
WHERE ENAME LIKE 'A%'  
UNION  
SELECT ENAME, JOB  
FROM EMP  
WHERE JOB LIKE 'C%';
```

# คำสั่ง SQL สำหรับจัดการข้อมูล

SELECT ENAME, JOB

FROM EMP

WHERE ENAME LIKE ‘A%’

INTERSECT

SELECT ENAME, JOB

FROM EMP

WHERE JOB LIKE ‘C%’;

# คำสั่ง SQL สำหรับจัดการข้อมูล

SELECT ENAME, JOB

FROM EMP

WHERE ENAME LIKE ‘A%’

MINUS

SELECT ENAME, JOB

FROM EMP

WHERE JOB LIKE ‘C%’;

# ตัวอย่าง

- Get color and city for “nonParis” parts with weight greater than ten.

```
SELECT P.COLOR, P.CITY
```

```
FROM P
```

```
WHERE P.CITY <> ‘Paris’
```

```
AND P.WEIGHT > 10;
```

# ตัวอย่าง

- Get the total number of suppliers.

```
SELECT COUNT (*)
```

```
FROM S;
```

# ตัวอย่าง

- Get the maximum and minimum quantity for part P2.

```
SELECT MAX(SP.QTY), MIN(SP.QTY)
```

```
FROM SP
```

```
WHERE SP.P# = 'P2';
```

# ตัวอย่าง

- For each part supplied, get the part number and the total shipment quantity.

```
SELECT SP.P#, SUM(SP.QTY)
```

```
FROM SP
```

```
GROUP BY SP.P#;
```

หรือ

# ตัวอย่าง

```
SELECT P.P#, (SELECT SUM(SP.QTY)
```

```
    FROM SP
```

```
    WHERE SP.P# = P.P#)
```

```
FROM P;
```

# ตัวอย่าง

- Get supplier names for suppliers who supply part P2.

```
SELECT DISTINCT S.SNAME
```

```
FROM S
```

```
WHERE S.S# IN
```

```
(SELECT SP.S#
```

```
FROM SP
```

```
WHERE SP.P# = 'P2');
```

หรือ

# ตัวอย่าง

```
SELECT DISTINCT S.SNAME  
FROM   S  
WHERE S.S# IN ('S1','S2','S3','S4');
```

หรือ

# ตัวอย่าง

```
SELECT DISTINCT S.SNAME  
FROM   S, SP  
WHERE  S.S# = SP.S#  
AND    SP.P# = 'P2';
```

# ตัวอย่าง

- Get supplier names for suppliers who supply at least one red part.

```
SELECT DISTINCT S.SNAME
```

```
FROM S
```

```
WHERE S.S# IN
```

```
(SELECT SP.S#
```

```
FROM SP
```

```
WHERE SP.P# IN
```

```
(SELECT P.P#
```

```
FROM P
```

```
WHERE P.COLOR = 'Red');
```

# ตัวอย่าง

- Get supplier numbers for suppliers with status less than the current maximum status in the S table.

```
SELECT S.S#
```

```
FROM S
```

```
WHERE S.STATUS <
```

```
(SELECT MAX(S.STATUS)
```

```
FROM S);
```

# ตัวอย่าง

- Get supplier names for suppliers who do not supply part P2.

```
SELECT DISTINCT S.SNAME
```

```
FROM S
```

```
WHERE NOT EXISTS
```

```
(SELECT *
```

```
FROM SP
```

```
WHERE SP.S# = S.S#
```

```
AND SP.P# = 'P2');
```

# ตัวอย่าง

หรือ

```
SELECT DISTINCT S.SNAME
```

```
FROM S
```

```
WHERE S.S# NOT IN
```

```
(SELECT SP.S#
```

```
FROM SP
```

```
WHERE SP.P# = 'P2');
```

# ตัวอย่าง

- Get supplier names for suppliers who supply all parts.

```
SELECT DISTINCT S.SNAME  
FROM S  
WHERE NOT EXISTS  
(SELECT *  
FROM P  
WHERE NOT EXISTS  
SELECT *  
FROM SP  
WHERE SP.S# = S.S#  
AND SP.P# = P.P#));
```

# ตัวอย่าง

หรือ SELECT DISTINCT S.SNAME

FROM S

WHERE (SELECT COUNT (SP.P#)

FROM SP

WHERE SP.S# = S.S#) =

(SELECT COUNT(P.P#)

FROM P);

# ตัวอย่าง

- Get part numbers for parts that either weigh more than 16 pounds or are supplied by supplier S2, or both.

**SELECT P.P#**

**FROM P**

**WHERE P.WEIGHT >16**

**UNION**

**SELECT SP.S#**

**FROM SP**

**WHERE SP.S# = ‘S2’);**