Database System

Structured Query Language

Introduction to SQL

- Structured Query Language (SQL)
  - เป็น Declarative language
    - ผู้ใช้ระบุเพียงข้อมูลที่ต้องการ โดยไม่จำเป็นต้องบอกวิธีการที่ได้มาซึ่งข้อมูล
  - เป็น relational database language
  - มีพื้นฐานผสมระหว่าง Relational Algebra และ Relational Calculus
  - พัฒนาครั้งแรกโดย IBM's San Jose Research Laboratory
  - ปี ค.ศ. 1986 ANSI และ ISO ได้ร่วมกันกำหนดมาตรฐานของภาษา SQL ซึ่งเรียกว่า SQL1 และต่อมาได้มีการทบทวนแก้ไขและเพิ่มเติมมาตรฐานขึ้นมาเป็น SQL2 หรือที่เรียกว่า SQL-92 (ปี ค.ศ. 1992)

SQL vs. Traditional Programming Languages

- Comparing SQL to most traditional general-purpose programming languages (3GL): C, COBOL, FORTRAN
  - 3GL: procedural and record-oriented
  - SQL: declarative and set-oriented
- Record-oriented processing
  - Each statement applies to each data item (i.e., each record)
- Set-oriented processing
  - Each statement applies to the whole set of data items (i.e., the relation(s) or the table(s))
- Note:
  - Prolog and Datalog are declarative and set-oriented
  - Relational Algebra is procedural and set-oriented

SQL Components

แบ่งเป็น 3 กลุ่ม ได้แก่

- ภาษาการนิยามข้อมูล

  DDL : Data Definition Language

- ภาษาปฏิบัติการของข้อมูล

  DML : Data Manipulation Language

- ภาษาควบคุมข้อมูล

  DCL : Data Control Language

กลุ่มคำสั่ง DML

- ใช้ในการจัดการข้อมูลในตาราง
- คำสั่งที่อยู่ในกลุ่ม DML
  - SELECT …
  - INSERT INTO…
  - UPDATE TABLE …
  - DELETE FROM …

Query

- คิวรี (Query) หมายถึง การสืบค้นข้อมูลในตารางข้อมูล ตามเงื่อนไขที่กำหนด

- รูปแบบทั่วไปของ SQL Query คือ
  - **select** <*List of Column Names*>
  - **from** <*List of Table Names*>
  - **where** <*Condition*>;

Select

- แสดงข้อมูลชื่อลูกค้าและเมืองที่ลูกค้าอยู่

```
select   Name, City
from     customer;
```

- แสดงข้อมูลทุกอย่างของลูกค้า

```
select   *
from     customer;
```

SQL Query Processing

- **SQL** เป็นภาษา **declarative**
  - ผู้ใช้ระบุเพียงข้อมูลที่ต้องการ โดยไม่จำเป็นต้องบอกวิธีการที่ได้มาซึ่งข้อมูล
  - **Query processors** อาจมีวิธีการทำงานที่แตกต่างกันเพื่อให้ได้มาซึ่งคำตอบ แต่ในท้ายที่สุดผลลัพธ์ที่ได้มาจะเหมือนกัน

SQL Query Processing

- Conceptual
  - ขั้นตอนที่ 1 **query processor** เริ่มทำงาน โดยมองหาประโยค **“from”** เพื่อใช้ในการระบุตารางที่ต้องการ
  - ขั้นตอนที่ 2 ตามด้วยการมองหาประโยค **“where”** เพื่อใช้ในการระบุแถวที่ต้องการ (จากตารางในขั้นตอนที่ 1) ตามเงื่อนไขซึ่งถูกกำหนดต่อท้ายจาก **where**
  - ขั้นตอนที่ 3 ประโยค **“select”** จะถูกประมวลผลเพื่อเลือกคอลัมน์ที่ต้องนำมาแสดง โดยชื่อคอลัมน์เหล่านั้นจะถูกกำหนดต่อท้าย **select**
- ในการทำงานจริง **query processors** จะค้นหาผลทางเลือกเตรียมไว้มากกว่า 1 วิธี

Select Distinct

- หากคำตอบที่ได้มานั้นมีข้อมูลที่เหมือนกันมากกว่า 1 แถว SQL จะไม่กำจัดข้อมูลเหล่านั้นออกไป
- **Ex.** พิมพ์ข้อมูลชื่อ-นามสกุลของพนักงานขายทั้งหมด

```
select      name, surname
from        emp;
```

  - แสดงผล (ชื่อ-นามสกุล) ของพนักงานขายทุกคน แม้จะมีพนักงานที่มีชื่อ-นามสกุลซ้ำกัน

Select Distinct (cont.)

- หากต้องการลบข้อมูลที่มีความเหมือนกันใช้คำสั่ง **distinct**

```
select    distinct name, surname
from      emp;
```

  - แสดงผลลัพธ์ที่เป็นเอกลักษณ์ **(unique)** ของ ชื่อ-นามสกุล
    - ชื่อเหมือนกัน นามสกุลต่าง แสดง 2 แถว
    - ชื่อต่างกัน นามสกุลเหมือนกัน แสดง 2 แถว
    - ชื่อและนามสกุลเหมือนกัน แสดง 1 แถว

Try it yourself!

- `SELECT city From customer;`
- `SELECT DISTINCT city FROM customer;`
- ได้ผลลัพธ์อย่างไร ?

SELECT Clause

- คอลัมน์ที่แสดงผล สามารถเปลี่ยนชื่อได้ (Alias name)

```
select      name as firstname, surname as lastname
from        emp;
```

แสดง 2 คอลัมน์ (firstname, lastname)

- SELECT clause สามารถบรรจุนิพจน์ทางคณิตศาสตร์ในคอลัมน์รวมถึงค่าคงที่ต่าง ๆ

- พิมพ์ชื่อและรายได้ (เงินเดือน + ค่าคอมมิสชั่น – 10%) ของพนักงานขาย

```
select      name, (sal + comm) * 0.9 as income
from        emp;
```

Try it

```
select   name as 'First name'
         , surname as lastname
from     emp;
```

WHERE Clause

Where clause ใช้เพื่อค้นหาข้อมูลที่ตรงตามเงื่อนไขระบุ

- พิมพ์ข้อมูลลูกค้าที่อยู่ในประเทศไทย

```
select   *
from     customer
where    country = 'Thailand';
```

เงื่อนไขที่เป็นตัวหนังสือเป็น case sensitive

```
Where country = 'thailand';
```

ไม่ได้ผลลัพธ์ค่าเดียวกับ

```
Where country = 'Thailand';
```

Try it yourself !

- พิมพ์เงินเดือนของพนักงานขายที่มี id = 123

```
select    sal
from      emp
where     id = 'E1';
```

ผลลัพธ์จะแสดงข้อมูลเพียงแค่คอลัมน์เงินเดือน (sal)

- แสดงพนักงานขายทุกคนที่มีเงินเดือนมากกว่า 10,000 บาท

```
select    *
from      emp
where     sal > 10000;
```

  - ผลลัพธ์อาจมีได้หลายแถว (หรือไม่มีเลย)
  - ผลลัพธ์แสดงข้อมูลทุกคอลัมน์ (select *)

Operator ที่ใช้ Where Clause

Operator	Description
=	เท่ากับ
<>	ไม่เท่ากับ (บางรุ่นของ SQL ใช้ != แทน)
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าหรือเท่ากับ
<=	น้อยกว่าหรือเท่ากับ
BETWEEN	ช่วงข้อมูล
LIKE	ค้นหาตามรูปแบบ
IN	ระบุเงื่อนไขหลากหลายที่เป็นไปได้

AND & OR Operators

Where clause สามารถบรรจุเงื่อนไขได้มากกว่า 1 เงื่อนไขเชื่อมต่อกันด้วยคำสั่ง **and , or, not**

- พิมพ์ข้อมูลพนักงานขายที่เงินเดือนมากกว่า 10,000 บาท และ เงินเดือนมากกว่าคอมมิสชั่น

```
select   *
from     emp
where    sal > 10000
and      sal > comm;
```

Between Operator

- เงื่อนไขเป็นช่วงของข้อมูล
- แสดงข้อมูลพนักงานซึ่งเงินเดือนอยู่ตั้งแต่ 8,000 ถึง 9,000

```
select   *
from     emp
where    sal between 8000 and 9000
```

- มีค่าเช่นเดียวกับ

```
select   *
from     emp
where    sal >= 8000 and  sal <= 9000
```

Like Operator

- SQL รองรับการจับคู่รูปแบบตัวอักษรอย่างง่าย (Pattern matching)
- การใช้ Pattern Matching ต้องใช้ควบคู่กับคำสั่ง **like**

- % แทนตัวอักษรไม่จำกัดจำนวน

  แสดงข้อมูลพนักงานขายที่ชื่อขึ้นต้นด้วย “Da”

```
select    *
from      emp
where     name like 'Da%';
```

Like Operator (cont.)

- _ (underscore)แทนตัวอักษร 1 ตัวอักษร

แสดงข้อมูลพนักงานที่ชื่อมีสามตัวอักษร

```
select *
from   emp
where  name like '_ _ _';
```

- \ ใช้เพื่อยกเลิกตัวอักษรมีความหมายเฉพาะ

แสดงพนักงานขายที่มีชื่อ ซึ่งมีเครื่องหมาย “_” รวมอยู่ด้วย

```
select *
from   emp
where  name like '%\_%';
```

IN Operator

- การค้นหาข้อมูลด้วยหลายเงื่อนไข
- แสดงข้อมูลพนักงานที่อยู่ใน **Bangkok** หรือ **London**

```
select *
from   emp
where  city IN ('Bangkok', 'London');
```

- มีค่าเช่นเดียวกับ

```
select *
from   emp
where  city = 'Bangkok'
       or city = 'London';
```

Is Null, Is Not Null

- เพื่อค้นหาข้อมูลที่ไม่มีค่าใด ๆ ปรากฏ
- Null <> Space
- แสดงข้อมูลพนักงานที่ไม่มีหัวหน้า

```
select *
from   emp
where  mgr_id is null;
```

- แสดงข้อมูลพนักงานที่มีหัวหน้า

```
select *
from   emp
where  mgr_id is not null;
```

การจัดเรียงข้อมูล (Order By)

- **ORDER BY** ใช้เพื่อเรียงลำดับแถวผลลัพธ์ที่ได้
- **ORDER BY** ทำงานหลังจากได้แถวผลลัพธ์ มาเป็นที่เรียบร้อยแล้ว และทำงานก่อน **“select”**

- ลำดับในการจัดเรียงมีด้วยกัน **2** ประเภท
  - ASC : **ascending** เรียงจากน้อยไปมาก
  - DESC : **descending** เรียงจากมากไปน้อย

  - Note : **ASC** เป็นค่าเริ่มต้น (ไม่จำเป็นต้องเขียนก็ได้)

การจัดเรียงข้อมูล (Order By) (cont.)

- “แสดงชื่อพนักงานและเงินเดือนโดยเรียงตามเงินเดือนจากน้อยไปมาก

```
select   name, sal
from     emp
order by sal asc;
```

- “แสดงชื่อ นามสกุลของลูกค้า เรียงลำดับชื่อลูกค้า จากน้อยไปมาก สำหรับลูกค้าที่มีชื่อซ้ำกัน ให้เรียงลำดับพวกเขาตามนามสกุล จากมากไปน้อย”

```
select   name, surname
from     customer
order by name asc, surname desc;
```

Aggregate Queries

- ข้อมูลหลาย ๆ แถวสามารถจัดกลุ่มรวมกันและแสดงผลในรูปแบบที่ถูกประมวลผลได้ **based on an aggregate function:**
  - Count
  - Sum
  - Average
  - Min
  - Max
- **“group by” clause** ถูกนำมาใช้เพื่อแสดงถึงการจัดกลุ่ม
- **“having” clause** ถูกนำมาใช้ (ปรากฏต่อจาก **group by**) เพื่อเลือกกลุ่มที่ตรงตามเงื่อนไข

Aggregate Function

- ฟังก์ชั่นที่คำนวณค่าแบบเป็นกลุ่ม ได้แก่
  - **Count:** นับจำนวนในกลุ่ม
  - **Sum:** ผลรวมค่าในกลุ่ม
  - **Avg:** ค่าเฉลี่ยของกลุ่ม
  - **Min:** ค่าน้อยที่สุดในกลุ่ม
  - **Max:** ค่ามากที่สุดในกลุ่ม

- ถ้าแถวที่มีค่า **null** จะไม่ถูกนำคำนวณด้วย (ยกเว้น **count(*)**)

Aggregate Queries

- “แสดงเงินเดือนที่น้อยที่สุด มากที่สุด และค่าเฉลี่ย ของพนักงานขาย”

```
select    min(sal) as low,
          max(sal) as high,
          avg(sal) as average
from      emp;
```

- “แสดงจำนวนลูกค้าที่ซื้อสินค้าในร้าน”

```
select    count(distinct c_id)
from      Orderh;
```

นับเฉพาะแถวที่แตกต่าง

- “แสดงจำนวนสินค้ากลุ่ม **P**”

```
select    count(*)
from      product
where     type = 'P';
```

นับทุกแถวที่ตรงตามเงื่อนไข

Aggregate Queries (cont.)

- “แสดงชื่อของสินค้าที่มีราคาสูงที่สุด”

```
select    name,price
from      product
where     price = (
             select   max(price)
             from     product
             );
```

Common mistakes

- ความผิดพลาดโดยทั่วไป

```
select   name, max(price)
from     product ;
```

คำสั่ง select ไม่สามารถใส่ คอลัมน์ปกติ (name) และ aggregate function (max) ไว้พร้อมกันได้ (หากไม่มีคำสั่ง group by อยู่ด้วย)

```
select   name, max(price)
from     product
where    max(price);
```

Aggregate function ไม่ใช้ใน where condition

จัดกลุ่มข้อมูล (Group By)

- “แสดงกลุ่มสินค้าที่มีสินค้าซึ่งราคาสูงสุดอยู่ในกลุ่มนั้น”

```
select    type, max(price)
from      product
group by type;
```

** คอลัมน์ปกติ (type) ที่ปรากฏใน **select clause** จะต้องปรากฏใน **group by clause** ด้วย

- “แสดงประเทศและจำนวนลูกค้าในประเทศนั้น”

```
select    country, count(id)
from      customer
group by country;
```

ข้อผิดพลาดโดยทั่วไป

- ข้อผิดพลาดโดยทั่วไป

```
select  avg(max(p.price))
from    product
group by type;
```

ไม่สามารถทำ Double aggregation ได้

เงื่อนไขการจัดกลุ่ม (Having)

- “แสดงราคาต่ำสุดของสินค้าในทุกกลุ่มสินค้าซึ่งมีค่าเฉลี่ย (ของแต่ละกลุ่มสินค้า) สูงกว่า 10,000 บาท”

```
select      type, min(price)
from        product
group by    type
having avg(price) > 10000;
```

- “having” เป็นคำสั่งเงื่อนไขสำหรับการแสดงผลแบบกลุ่ม ดังนั้นโดยปกติจะต้องใช้ร่วมกับ aggregate function มีเพียงกลุ่มที่ตรงตามเงื่อนไขเท่านั้นที่จะถูกนำออกมาแสดงผล

เงื่อนไขการจัดกลุ่ม (Having)

- แสดงชื่อสาขา , จำนวนพนักงานที่ทำงานอยู่ในแต่ละสาขา และเงินเดือนรวมของพนักงานในสาขานั้นๆ โดยแสดงเฉพาะสาขาที่มีพนักงานสังกัดมากกว่า 1 คน

```
select    dept_id, count(id) as numEmp
          , sum(sal) sumsal
from      emp
group by dept_id
having count(id) > 1;
```

Computing Sequence

- ลำดับการเขียนคำสั่ง

```
select      column/aggregation
from        tables
where       row condition
group by    type;
having      group condition
order by     column/aggregation
```

- ลำดับการประมวลผล
  - from : เลือกตาราง
  - where : เลือกแถว
  - group by : จัดกลุ่ม
  - having : เลือกกลุ่ม
  - order by : เรียงลำดับแถวหรือกลุ่ม
  - select : เลือกคอลัมน์

FROM Clause

- ถ้า **From clause** บรรจุตารางมากกว่า 1 , **Cartesian product** จะถูกประมวลผล
- แสดงชื่อพนักงานและชื่อสาขาที่พนักงานสังกัด

```
select    emp.name as ename , dept.name as dname
from      emp, dept
where     emp.dept_id = dept.id;
```

- ชื่อตารางสามารถเปลี่ยนเพื่อให้ง่ายต่อการใช้งานด้วยคำสั่ง **as**

```
select    e.name as ename , d.name as dname
from      emp as e, dept as d
where     e.dept_id = d.id;
```

FROM Clause + Where Clause

- แสดงชื่อสินค้า ราคาสินค้าและชื่อประเภทสินค้า โดยแสดงเฉพาะสินค้าที่ราคามากกว่า 10,000 บาท

```
select      p.name as pname ,price
            , t.name as tname
from        product as p, producttype as t
where       p.ptype= t.id
and         price > 10000;
```

INNER JOIN

- เมื่อมีหลาย ๆ ตารางใน **from clause** เราสามารถใช้เงื่อนไขการ **JOIN** เพื่อใช้การเชื่อมโยงตารางเข้าด้วยกันทดแทนการใช้ **WHERE clause** ได้

- แสดงชื่อพนักงานและชื่อสาขาที่พนักงานสังกัด (ใช้ where)

```
select    e.name as ename , d.name as dname
from      emp as e, dept as d
where     e.dept_id = d.id;
```

- แสดงชื่อพนักงานและชื่อสาขาที่พนักงานสังกัด (ใช้ join)

```
select    e.name as ename , d.name as dname
from      emp as e inner join dept as d
on        e.dept_id = d.id;
```

INNER JOIN + Where Clause

- แสดงชื่อสินค้า ราคาสินค้าและชื่อประเภทสินค้า โดยแสดงเฉพาะสินค้าที่ราคามากกว่า 10,000 บาท (ใช้ where)

```
select    p.name as pname ,price
          , t.name as tname
from      product as p, producttype as t
where     p.ptype= t.id
and       price > 10000;
```

- ใช้ join

```
select    p.name as pname ,price
          , t.name as tname
from      product as p inner join producttype as t
          on p.ptype= t.id
where     price > 10000;
```

Example

- แสดงหมายเลขใบสั่งซื้อ วันที่ซื้อ รหัสสินค้าและจำนวนที่ซื้อ (ใช้ where)

```
select    h.id ,h.orderdate, d.pid, d.qty
from      orderh as h, orderd as d
where     h.id = d.oid;
```

- ใช้ join

```
select    h.id ,h.orderdate, d.pid, d.qty
from      orderh as h inner join orderd as d
On        h.id = d.oid;
```

Try it yourself !

- แสดงหมายเลขใบสั่งซื้อ, ชื่อลูกค้า (ด้วยคำสั่ง Inner join)
- แสดงหมายเลขใบสั่งซื้อ, ชื่อลูกค้า, ชื่อพนักงานขาย (ด้วยคำสั่ง Inner join)

INNER JOIN หลายตาราง

- แสดงหมายเลขใบสั่งซื้อ วันที่ซื้อ ชื่อสินค้าและจำนวนที่ซื้อ (ใช้ where)

```
select    h.id ,h.orderdate, p.name, d.qty
from      orderh as h, orderd as d, product as p
where     h.id = d.oid
and       d.pid = p.id;
```

- ใช้ join

```
select    h.id ,h.orderdate, p.name d.qty
from      (orderh as h inner join orderd as d
          on h.id = d.oid )
          inner join product as p
          on d.pid = p.id;
```

INNER JOIN หลายตาราง

- แสดงหมายเลขใบสั่งซื้อ ชื่อลูกค้า ชื่อสินค้าและจำนวนที่ซื้อ (ใช้ where)

```
select    h.id ,c.name, p.name, d.qty
from      orderh as h, orderd as d, product as p
          ,customer as c
where     h.id = d.oid
and       d.pid = p.id
and       h.cid = c.id;
```

INNER JOIN หลายตาราง

- แสดงหมายเลขใบสั่งซื้อ ชื่อลูกค้า ชื่อสินค้าและจำนวนที่ซื้อ (ใช้ join)

```
select    h.id ,c.name, p.name d.qty
from      ((orderh as h inner join orderd as d
          on h.id = d.oid)
          inner join product as p
          on d.pid = p.id))
          inner join customer as c
          on h.cid = c.id;
```

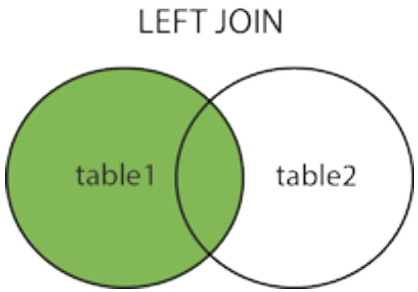
ไม่ใส่วงเล็บที่ `from clause` ก็ได้ แต่ต้องระวังเรื่องลำดับของตาราง

OUTER JOIN

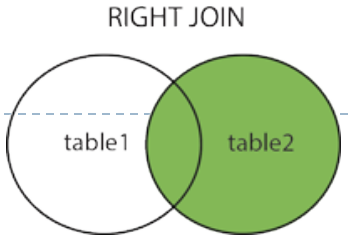
- ในกรณีที่มีการเชื่อมโยงตาราง ถ้าในตารางใดไม่มีแถวที่สามารถเชื่อมโยงกับอีกตารางหนึ่งได้ ข้อมูลจะไม่ถูกแสดงผล แต่เราสามารถใช้ คำสั่ง SQL เพื่อบังคับให้แสดงผลได้ดังนี้

- “ แสดงชื่อสินค้า ชื่อประเภทสินค้า (ถ้ามี)”

```
select   p.name , t.name
from     product as p left outer join
         producttype as t
on       p.ptype= t.id;
```



OUTER JOIN (cont.)



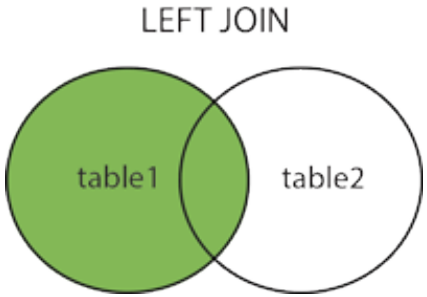
- " แสดงชื่อสินค้า (ถ้ามี) ชื่อประเภทสินค้า"

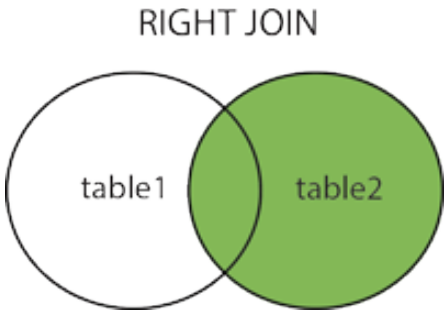
```
select      p.name, t.name
From        product as p right outer join
            producttype as t
on          p.ptype= t.id;
```

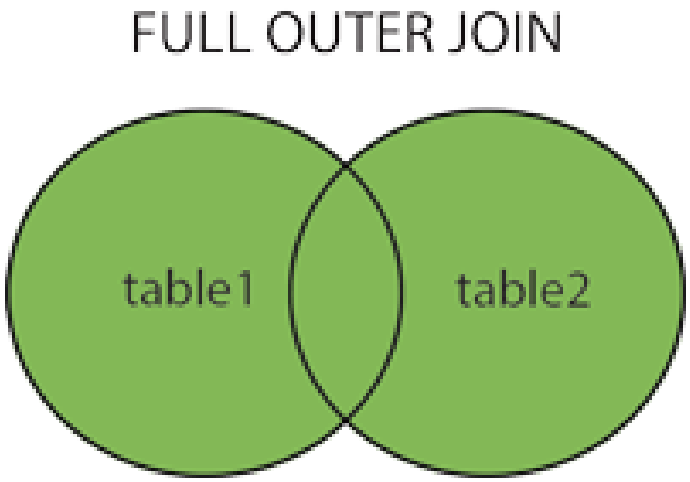
- "แสดงชื่อสินค้า ชื่อประเภทสินค้า ไม่ว่าจะมีข้อมูลอ้างอิงหรือไม่ก็ตาม"

```
Select      p.name, t.name
From        product as p full outer join
            producttype as t
on          p.ptype= t.id;
```

Outer join


LEFT JOIN
table1
table2


RIGHT JOIN
table1
table2


FULL OUTER JOIN
table1
table2

Data Modification in the Databases

- การแก้ไขข้อมูลในฐานข้อมูลมีอยู่ 3 ประเภทด้วยกัน ได้แก่
  - Insertion
    - เพิ่มข้อมูลลงในฐานข้อมูล
  - Deletion
    - ลบข้อมูล
  - Update
    - ปรับปรุงข้อมูล

Insertion Statements

- เพิ่มข้อมูลในตาราง

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Insertion Statements

- เพิ่มข้อมูลใหม่ลงในตาราง

```
Insert into product (id, name, ptype, price, stock)
    values ('P71', 'Tablet 7', 'P', 12999.99,35);
```

- หรือ ไม่ระบุคอลัมน์ แต่จะใส่ข้อมูลให้ตรงกับคอลัมน์ในตาราง **ห้ามขาด**

```
Insert into product
    values ('P72', 'Laptop', 'N', 1000.00,40);
```

- การเพิ่มข้อมูลจะสามารถทำได้ทีละ 1 แถวเท่านั้น
- ในกรณีที่ไม่มีข้อมูลในคอลัมน์นั้นใช้ **null** แทนได้

```
Insert into product
    values ('P73', 'Huawei', 'P',12999.99,null);
```

Deletion Statements

- ลบข้อมูลจากตาราง

```
DELETE FROM table_name WHERE condition;
```

Deletion Statements

- “ลบข้อมูลทุกอย่างเกี่ยวข้องกับสินค้ากลุ่ม **P4** ใน ตารางกลุ่มของสินค้า, สินค้า, การซื้อสินค้า

```
delete from Orderd
where prod_id in (
      select  id
      from  product
      where type = 'A');

delete from product
where id = 'P99';

delete from category
where type = 'M';
```

Update Statements

- ปรับปรุงข้อมูลในตาราง

```
UPDATE table_name
SET column1 = value1
    , column2 = value2, ...
WHERE condition;
```

Update Statements

- “เปลี่ยนชื่อสินค้าหมายเลข P4 เป็น ‘Pocket PC’ เปลี่ยนกลุ่มสินค้าเป็นหมายเลข M และเพิ่มราคาอีก 10%”

```
update product
set    name = 'iPhone 64GB',
       type   = 'M',
       price  = price * 1.1
where id = 'P4';
```

Complex Update Statements

- “เฉพาะสินค้าในกลุ่ม A เพิ่มราคาขายสินค้าขึ้น 10% สำหรับสินค้าที่ราคาต่ำกว่า 10000 บาท และลดราคาขายสินค้าลง 15% สำหรับสินค้าที่มีราคาสูงกว่า 15000 บาท”

```
update product
set     price =
        case
         when price < 10000 then price * 1.1
         when price > 15000 then price * 0.85
         else price
        end
where type = 'A';
```

กลุ่มคำสั่ง DCL

- ใช้ในการกำหนดสิทธิการใช้งานฐานข้อมูลให้แก่ผู้ใช้
- คำสั่งที่อยู่ในกลุ่ม DCL
  - GRANT สำหรับการให้สิทธิ์การใช้งานฐานข้อมูล
  - REVOKE สำหรับการยกเลิกสิทธิ์การใช้งานฐานข้อมูล

End of DML, DCL