Structured Query Language: SQL

ไวยกรณ์โดยทั่วไป

ส่วนที่สำคัญ

Select …

ระหว่าง select กับ from คือ ส่วน columns ที่เลือกมาหลัง from หรือ join หรือ subquery

from…

ส่วนประกอบเพิ่มเติมจะมีหรือไม่มีก็ได้

join… (inner join, left join ,right join)

Where…

Group by… (ใช้ในกรณีที่มี sum count avg etc.…)

Having…

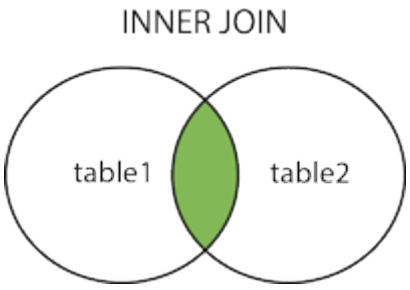
Order by…

การ Join

1. *Inner Join*

Table A	Table B	ResultAB
A	A	A A
B	B	B B
C	C	C C
D	E	
F	F	F F
G	G	G G
S	R	

ข้อมูลของการใช้งาน inner join ข้อมูลจะแสดงผลออกมาเฉพาะข้อมูลที่มีเหมือนกันใน Columns ที่ใช้ join เท่านั้น

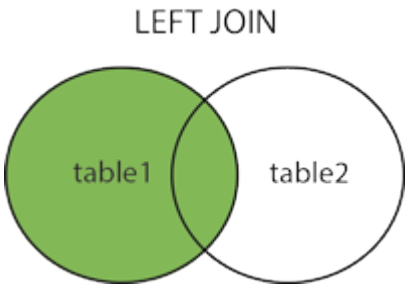


join = inner join
นอกจากนั้นสามารถใช้ where ได้ด้วยเช่นกัน

2. *Left join*

Table A	Table B	Result AB
A	A	A A
B	B	B B
C	C	C C
D	E	D *Null*
F	F	F F
G	G	G G
S	R	S *Null*

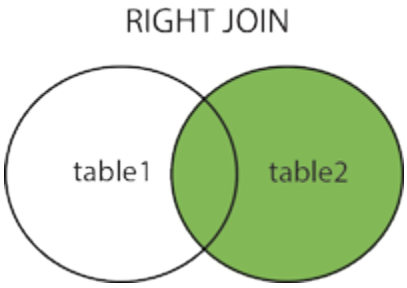
ข้อมูลของการใช้งาน **Left join** ข้อมูลทั้งหมดออกมาโดยอ้างอิงจากตารางข้อมูลด้านซ้าย โดยการแสดงผลนั้นจะแสดงผลของข้อมูลฝั่งซ้ายมีออกมาทั้งหมด และหากฝั่งขวามือข้อมูลที่ **join** ไม่มีเหมือนด้านซ้ายมือข้อมูลฝั่งขวามือจะขึ้น **Null**



3. *Right join*

Table A	Table B	Result AB
A	A	A A
B	B	B B
C	C	C C
D	E	*Null* E
F	F	F F
G	G	G G
S	R	*Null* R

ข้อมูลของการใช้งาน **Right join** ข้อมูลทั้งหมดออกมาโดยอ้างอิงจากตารางข้อมูลด้านขวาโดยการแสดงผลนั้นจะแสดงผลของข้อมูลฝั่งขวามือออกมาทั้งหมด และหากฝั่งซ้ายมือข้อมูลที่ **join** ไม่มีเหมือนด้านซ้ายมือข้อมูลฝั่งซ้ายมือจะขึ้น **Null**



ตัวอย่างคำสั่งการใช้งาน join

```
SELECT column_name(s)
        FROM table1
 JOIN table2 ON table1.column_name = table2.column_name;
```

คำสั่ง where

Where เป็นการกำหนดเงื่อนไขสำหรับการเลือกข้อมูล เช่น เอาเฉพาะ คนที่มีชื่อขึ้นต้นด้วยตัว A

Sqlite

Select * from Table_A

Where srt(Name,1,1) = 'A'

ข้อมูล string 1 ตัว

Sql sever

Select * from Table_A

Where left(Name,1) = 'A'

ข้อมูล string 1 ตัว

Other

Select * from Table_A

Where Name = 'Earth'

ข้อมูล string 1 ตัว

Other

Select * from Table_A

Where Name = 'Earth''

ข้อมูล string 1 ตัว

Select * from Table

Where Soldtotal > 10,000

ข้อมูล int , float 1 ตัว

Select * from Table

Where Soldtotal > 10,000

ข้อมูล int , float 1 ตัว

Select * from Table

Where Soldtotal between 100 and 2000

ข้อมูลระหว่าง 100 ถึง 2000

Select * from Table

Where Soldtotal between 100 and 2000

ข้อมูลระหว่าง 100 ถึง 2000

Select * from Table

Where City in ('TH','ENG')

ข้อมูล string 1 ตัวขึ้นไป

Select * from Table

Where City in ('TH','ENG')

ข้อมูล string 1 ตัวขึ้นไป

Select * from Table

Where Qty in (3,4)

ข้อมูล int , float 1 ตัวขึ้นไป

Select * from Table

Where Qty in (3,4)

ข้อมูล int , float 1 ตัวขึ้นไป

คำสั่ง Group by

ใช้ในกรณีในส่วนของ Columns มีการใช้ Sum(), Count(), Avg() etc. หรือใช้ในกรณีที่มี Having เช่น

```
Select
    Product_Category,
    Sum(Soldamount)
    from Soldorder
Group by Product_Category
```

```
Select
    Product_Category,
    Count(Soldamount)
    from Soldorder
Group by Product_Category
```

```
Select
    Product_Category,
    Avg(Soldamount)
    from Soldorder
Group by Product_Category
```

การ group by ครั้งนี้เราจะ group เฉพาะในส่วนของ Columns ที่ไม่ได้ใช้คำสั่ง Sum,count,avg ในส่วนนี้ Value ข้อมูลจะถูกแบ่งตาม Product_Category ออกเป็นส่วนๆ

```
select
    Color,
    sum(Standardcost) as Standardcost
    from Product
group by Color
```

113 %

Results | Messages

	Color	Standardcost
1	Black	38636.5002
2	Blue	14746.1464
3	Grey	51.5625
4	Multi	272.2542
5	NULL	1177.5855
6	Red	32610.7661
7	Silver	20060.0483
8	Silver/Black	198.97
9	White	13.5172
10	Yellow	21507.6521

```
select
    Color,
    Size,
    sum(Standardcost) as Standardcost
    from Product
group by Color,Size
```

Results | Messages

Color	Size	Standardcost
Black	NULL	36533.6994
Blue	NULL	14661.8131
Red	NULL	32597.6798
Silver	NULL	19861.9399
Yellow	NULL	21341.3629
Black	L	106.6858
Blue	L	23.749
Multi	L	75.6132
White	L	6.7586
Yellow	L	41.5723
Black	M	106.6858
Blue	M	23.749
Multi	M	75.6132
White	M	6.7586

หากในส่วนที่แสดง columns ยิ่งมี Feature มากแค่ไหนเราก็ต้อง Group by มากเท่าจำนวน Feature ที่เราใส่เข้าไป

คำสั่ง Having

ส่วนมากแล้วจะใช้มักจะใช้เมื่อต้องการใส่เงื่อนไขที่เมื่อคำนวณโดย ต้องการให้ค่าเป็นไปในทางไหนมากกว่าหรือน้อยกว่า มีความคล้ายคลึงกับการใช้เงื่อนไข where เพียงแต่เมื่อใช้ Having แล้วต้องมีการใส่ Group by เสมอ

```
select
    Color
    from Product
group by Color
having sum(Standardcost) > 10000
```

Results | Messages

Color
Black
Blue
Red
Silver
Yellow

```
select
    Color,
    sum(Standardcost) as Standardcost
    from Product
group by Color
having sum(Standardcost) > 10000
```

Results | Messages

Color	Standardcost
Black	38636.5002
Blue	14746.1464
Red	32610.7661
Silver	20060.0483
Yellow	21507.6521

เราไม่สามารถใช้เงื่อนไข where ได้ หากใช้เงื่อนไขจะขึ้นว่า

```
select
    Color,
    sum(Standardcost) as Standardcost
    from Product
    where sum(Standardcost) > 10000
group by Color
```

```
An aggregate may not appear in the WHERE clause unless it is in a subquery contained
in a HAVING clause or a select list, and the column being aggregated is an outer
reference.
```

เราสามารถใช้เงื่อนไข where ได้แต่ต้องทำเป็น Subquery ก่อน

```
select * from
    (
    select
        Color,
        sum(Standardcost) as Standardcost
        from Product
    group by Color
    ) as Main
where Main.Standardcost > 10000
```

Results | Messages

Color	Standardcost
Black	38636.5002
Blue	14746.1464
Red	32610.7661
Silver	20060.0483
Yellow	21507.6521

คำสั่ง Order by

เป็นการจัดลำดับให้เรียง:

จากมากไปน้อย Desc

น้อยไปมาก Asc

```
select
    Color,
    sum(Standardcost) as Standardcost
    from Product
group by Color
order by sum(Standardcost) asc
```

Results | Messages

Color	Standardcost
White	13.5172
Grey	51.5625
Silver/Black	198.97
Multi	272.2542
NULL	1177.5855
Blue	14746.1464
Silver	20060.0483
Yellow	21507.6521
Red	32610.7661
Black	38636.5002

```
select
    Color,
    sum(Standardcost) as Standardcost
    from Product
group by Color
order by sum(Standardcost) desc
```

Results | Messages

Color	Standardcost
Black	38636.5002
Red	32610.7661
Yellow	21507.6521
Silver	20060.0483
Blue	14746.1464
NULL	1177.5855
Multi	272.2542
Silver/Black	198.97
Grey	51.5625
White	13.5172

โจทย์ SQL (จริงๆแล้วสามารถเขียนได้หลายวิธีอันนี้เป็นพียงแค่วิธีที่ผมถนัดเขียนบ่อยครับ)

1. List every postal code from address table which does not have any order has shipped to this postal code. *

```
select Address.AddressID from Address
       where Address.AddressID not in(select BillToAddressID from SalesOrderHeader)
```

ความสำคัญของตัวนี้อยู่ที่ Not in คือข้อมูลใน Address ที่ไม่มีใน subquery ตัวนี้

2. Permanently decrease list price of products in bikes category by 5%, including its sub-category. *

```
select Name,(ListPrice * 0.95) as Markdown_price from Product
       where Product.ProductCategoryID in
(select ProductCategoryID from ProductCategory  where ParentProductCategoryID = 1)
```

ความสำคัญของตัวนี้อยู่ที่ in คือข้อมูลใน Product ที่มีใน subquery ตัวนี้ select ProductCategoryID from ProductCategory where ParentProductCategoryID = 1

3. Delete address of customer name "Margaret J. Smith" from customer address table. *

ข้อนี้จำไว้ว่า FK ต้อง Link ไปที่ PK เสมอ กรณีนี้ database ไม่สมบูรณ์คือ ตาราง customer ไม่มี PK ที่ CustomerID เราจำเป็นต้องแก้ไขให้ ตาราง customer ให้มี PK ที่ CustomerID

โดยนี่คือตารางที่เราจำเป็นต้องลบ คนที่มีชื่อ Margaret ออก ซึ่ง CustomerID ของ Margaret คือ 29490

```
CREATE TABLE "CustomerAddress" (
        "CustomerID"    INTEGER,
        "AddressID"     INTEGER,
        "AddressType"   TEXT,
        "rowguid"  TEXT,
        "ModifiedDate"  TEXT,
        PRIMARY KEY("CustomerID","AddressID"),
        FOREIGN KEY("AddressID") REFERENCES "Address"("AddressID"),
        FOREIGN KEY("CustomerID") REFERENCES "Customer"("CustomerID")
)
```

จากโครงสร้างตัวที่มีปัญหาคือ FOREIGN KEY("CustomerID") REFERENCES "Customer"("CustomerID") เพราะ ตาราง Customer ไม่มี PK เป็น CustomerID

อันนี้คือโครงสร้าง ตาราง Customer จะเห็นว่าไม่มี PK ที่ CustomerID

```
CREATE TABLE "Customer" (
        "CustomerID"    INTEGER,
        "NameStyle"     INTEGER,
```

```
    "Title"     TEXT,
    "FirstName"         TEXT,
    "MiddleName"        TEXT,
    "LastName"          TEXT,
    "Suffix"    TEXT,
    "CompanyName"       TEXT,
    "SalesPerson"       TEXT,
    "EmailAddress"      TEXT,
    "Phone"     TEXT,
    "PasswordHash"      TEXT,
    "PasswordSalt"      TEXT,
    "rowguid"   TEXT,
    "ModifiedDate"      TEXT
)
```

ขั้นตอนต่อไปเราจะทำการสร้าง PK ขึ้นมาให้ตาราง Customer ณ CustomerID

1. **สร้างตาราง** dummy **ขึ้นมา**

```
CREATE TABLE dumb_Customer
(
    CustomerID   INTEGER,
    NameStyle   INTEGER,
    Title TEXT,
       FirstName TEXT,
       MiddleName TEXT,
       LastName TEXT,
       Suffix TEXT,
       CompanyName TEXT,
       SalesPerson TEXT,
       EmailAddress TEXT,
       Phone TEXT,
       PasswordHash TEXT,
       PasswordSalt TEXT,
       rowguid TEXT,
       ModifiedDate TEXT,
)
```

2. **ดรั้มข้อมูลจาก** Customer **เข้าไปใน** dumb_Customer

```
INSERT INTO dumb_Customer(CustomerID, NameStyle,
Title,FirstName,MiddleName,LastName,Suffix,CompanyName,SalesPerson,EmailAddress,Phone,
PasswordHash,PasswordSalt,rowguid,ModifiedDate)
 SELECT CustomerID, NameStyle,
Title,FirstName,MiddleName,LastName,Suffix,CompanyName,SalesPerson,EmailAddress,Phone,
PasswordHash,PasswordSalt,rowguid,ModifiedDate
 FROM Customer
```

3. **ลบตาราง** Customer **ทิ้ง**

```
DROP TABLE Customer
```

4. สร้างตาราง Customer ขึ้นมาใหม่ และสร้าง PK ให้กับ CustomerID ย้ำว่า Feature ทุกตัว Datatype และ ชื่อ Feature ต้องเหมือนกับตารางอันเก่า

```
CREATE TABLE Customer
(
    CustomerID   INTEGER,
    NameStyle   INTEGER,
    Title TEXT,
       FirstName TEXT,
       MiddleName TEXT,
       LastName TEXT,
       Suffix TEXT,
       CompanyName TEXT,
       SalesPerson TEXT,
       EmailAddress TEXT,
       Phone TEXT,
       PasswordHash TEXT,
       PasswordSalt TEXT,
       rowguid TEXT,
       ModifiedDate TEXT,
       PRIMARY KEY(CustomerID)
)
```

5. ดรั้มข้อมูลจาก dumb_Customer เข้าไปใน Customer

```
INSERT INTO Customer(CustomerID, NameStyle,
Title,FirstName,MiddleName,LastName,Suffix,CompanyName,SalesPerson,EmailAddress,Phone,
PasswordHash,PasswordSalt,rowguid,ModifiedDate)
 SELECT CustomerID, NameStyle,
Title,FirstName,MiddleName,LastName,Suffix,CompanyName,SalesPerson,EmailAddress,Phone,
PasswordHash,PasswordSalt,rowguid,ModifiedDate
 FROM dumb_Customer
```

6. ขึ้นอยู่กับเราว่าจะ ลบตารางเก่าทิ้งหรือไม่ก็ได้

```
DROP TABLE dumb_Customer
```

7. ลบข้อมูลของ ซึ่ง CustomerID ของ Margaret คือ **29490**

```
DELETE from CustomerAddress
       where CustomerID = 29490
```

4. Retrieve sales order number of an order which is billed to postal code "95603". *

ขั้นตอนแรก Check ก่อนว่าลักษณะการเชื่อมตารางเป็นยังไง

```
select * from Address
       where PostalCode = 95603

select * from CustomerAddress
       where AddressID = 1092

select * from SalesOrderHeader
       where CustomerID = 29847 (จริงๆตอบอันนี้เลยก็ได้)
```

กรณีอยากเขียนแบบ subquery

```
select *
from SalesOrderHeader
where CustomerID in
(
select CustomerAddress.CustomerID from Address
inner join CustomerAddress on Address.AddressID = CustomerAddress.AddressID
where Address.PostalCode = 95603
)
```

5. List address id of addresses which never used as billing address. *

```
select
       Address.AddressID,
       SalesOrderHeader.BillToAddressID
From  Address
left join SalesOrderHeader on  Address.AddressID = SalesOrderHeader.BillToAddressID
where SalesOrderHeader.BillToAddressID is null
```

ใช้ left join เพราะ เราต้องการให้แสดงตารางทั้งหมด โดยหาก ข้อมูลในตาราง SalesOrderHeader แสดงค่าออกมาเป็น null นั้นแปลว่า รหัส address นี้ไม่เคยมีการสั่งซื้อสินค้า

6. Change shipping address of an order id 71780 to an address of customer "Amy E. Alberts". *

เช็คว่า Customer ที่ชื่อ Amy E.Alberts มีรหัสลูกค้าอะไร

```
select * from Customer
       where FirstName like('%Amy%')
```

เช็คว่า SalesOrderHeader ที่มีรหัส 71780 มี ShipToAddressID อะไร

```
select * from SalesOrderHeader
       where SalesOrderID = 71780
```

จากนั้นทำการ Update ค่า

```
Update SalesOrderHeader
set ShipToAddressID = 29499
    where SalesOrderID = 71780
```

กรณีใช้ subquery

```
Update SalesOrderHeader
set ShipToAddressID =(select CustomerID from Customer where  CustomerID = 29499)
    where SalesOrderID  =(select SalesOrderID from SalesOrderHeader where SalesOrderID
    = 71780)
```

7. Without using any join operator, list name of the products which have a total ordered number (SalesOrderDetail.OrderQty) less than 3 *

ใช้ sum เมื่อใช้ sum แล้วต้องตามด้วย Groupby เสมอนอกจากนั้นต้องใช้ subquery และการใช้เงื่อนไขที่เป็นการคำนวณ เช่น sum count avg ต้องใช้ having นอกจากว่าจะทำให้ main query เป็น subquery ถึงจะใช้ where ได้

```
select
       SalesOrderID,
       sum(OrderQty) as OrderQty
from SalesOrderDetail
group by SalesOrderID
having sum(OrderQty) < 3
```

กรณีจะใช้ where ทำให้ main query เป็น subquery

```
select * from
(
       select
              SalesOrderID,
              sum(OrderQty) as OrderQty
              from SalesOrderDetail
       group by SalesOrderID
) as Subquery
       where Subquery.OrderQty < 3
```

กรณีใช้ subquery ทั้งหมด

```
	select
		distinct
		SalesOrderID,
		(select sum(OrderQty) from SalesOrderDetail as Sub where Sub.SalesOrderID =
Main.SalesOrderID) as OrderQty
		from SalesOrderDetail as Main
		where (select sum(OrderQty) from SalesOrderDetail as Sub where
Sub.SalesOrderID = Main.SalesOrderID) < 3
```

8. List customer names whose don't have orders in the system without join operator. *

สร้าง subquery ของ salesorderheader และเลือกแต่ CustomerID ต่อท้าย (select CustomerID from SalesOrderHeader) เพราะ where เอา Feature เข้าได้ทีละตัวจะใช้ มากกว่า 1 Feature ไม่ได้ หากมากกว่านั้นต้องใช้ And , or ใช้คำสั่ง not in เพื่อที่ให้ CustomerID จากตาราง Customer ที่ไม่อยู่ใน Subqurey แสดงค่าออกมา

```
select CustomerID,
	(FirstName || ' ' || LastName) as Customer_Name
	from Customer
where CustomerID not in ( select CustomerID from SalesOrderHeader )
```

9. Retrieve customer names, total orders and the total amount they're spent in the lifetime of our system. Write a single query without joins, group by, or having operators. *

จะสังเกตได้ว่า subquery จะใช้ sum , countโดยที่ไม่ต้องใช้ group by เนื่องจากกรณีนี้ ตัว subquery ได้อยู่ในส่วนที่ใช้แสดงผล (ส่วนของ Columns) จำเป็นต้อง groub by โดยผ่านเงื่อนไขการใช้ where join ระหว่างตาราง ไม่เช่นนั้นระบบจะขึ้น Only one expression can be specified in the select list when the subquery is not introduced with EXISTS . คือ ข้อมูลใน subquery แสดงค่ามากกว่า 1 ค่า และค่าที่ถูกแสดงนั้นระบบไม่รู้ว่าต้องเอาไปเชื่อมกับข้อมูลตัวไหนใน feature ไหน แต่ในกรณีที่ Feature ตัวนั้นมีความ Unique อยู่แล้วก็สามารถใช้งานได้โดยไม่จำเป็นต้องมี sum , count ,etc.

```
select

(Customer.FirstName || ' ' || Customer.LastName) as Customer_Name,

(select sum(TotalDue) from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as TotalDue,

(select Count(*) from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as CountOrder

from Customer
```

สามารถเขียนแบบนี้ก็ได้ เพราะข้อมูล TotalDue มีความ unique อยู่แล้ว

```
select
(Customer.FirstName ||' '|| Customer.LastName) as Customer_Name,

(select TotalDue from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as TotalDue,

(select Count(*) from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as CountOrder

from Customer
```

กรณีที่อยากให้แสดงเฉพาะรายการที่ลูกค้าซื้อ สร้าง main query เป็น subquery

```
select * from
(

select
(Customer.FirstName ||' '|| Customer.LastName) as Customer_Name,
(select sum(TotalDue) from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as TotalDue,

(select Count(*) from SalesOrderHeader where Customer.CustomerID =
SalesOrderHeader.CustomerID) as CountOrder

from Customer
) as Subquery
       where Subquery.TotalDue is not null
```