

Database

What to do in case of BIG DATA!

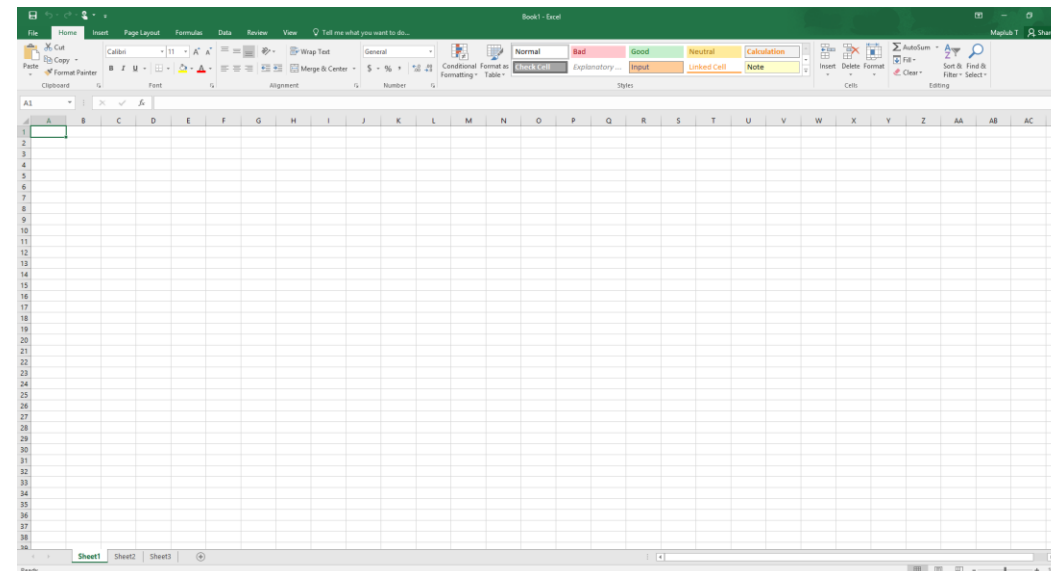
Last time

เซ็นเซอร์วัดฝุ่น PM2.5 ติดตั้งอยู่ในตำบลแห่งหนึ่งของจังหวัดน่าน มีข้อมูลวัดค่าฝุ่นดังต่อไปนี้

- วันที่ 4 ก.ย. 2023 เวลา 6:00 น. วัดได้ 15 ppm
- วันที่ 5 ก.ย. 2023 เวลา 6:00 น. วัดได้ 15 ppm
- วันที่ 6 ก.ย. 2023 เวลา 6:00 น. วัดได้ 15 ppm

ถ้ามีเซ็นเซอร์วัดฝุ่นอยู่ 200 เครื่อง แต่ละเครื่องวัดข้อมูลทุกชั่วโมง จะเก็บบันทึกข้อมูลอย่างไร

Excel ก็ไม่รอด!



BIG data

(ตอนนี้ก็ยังไม่ Big พอ)

Topic

- What is database
- Types of database
- RDBMS

Database คืออะไร?

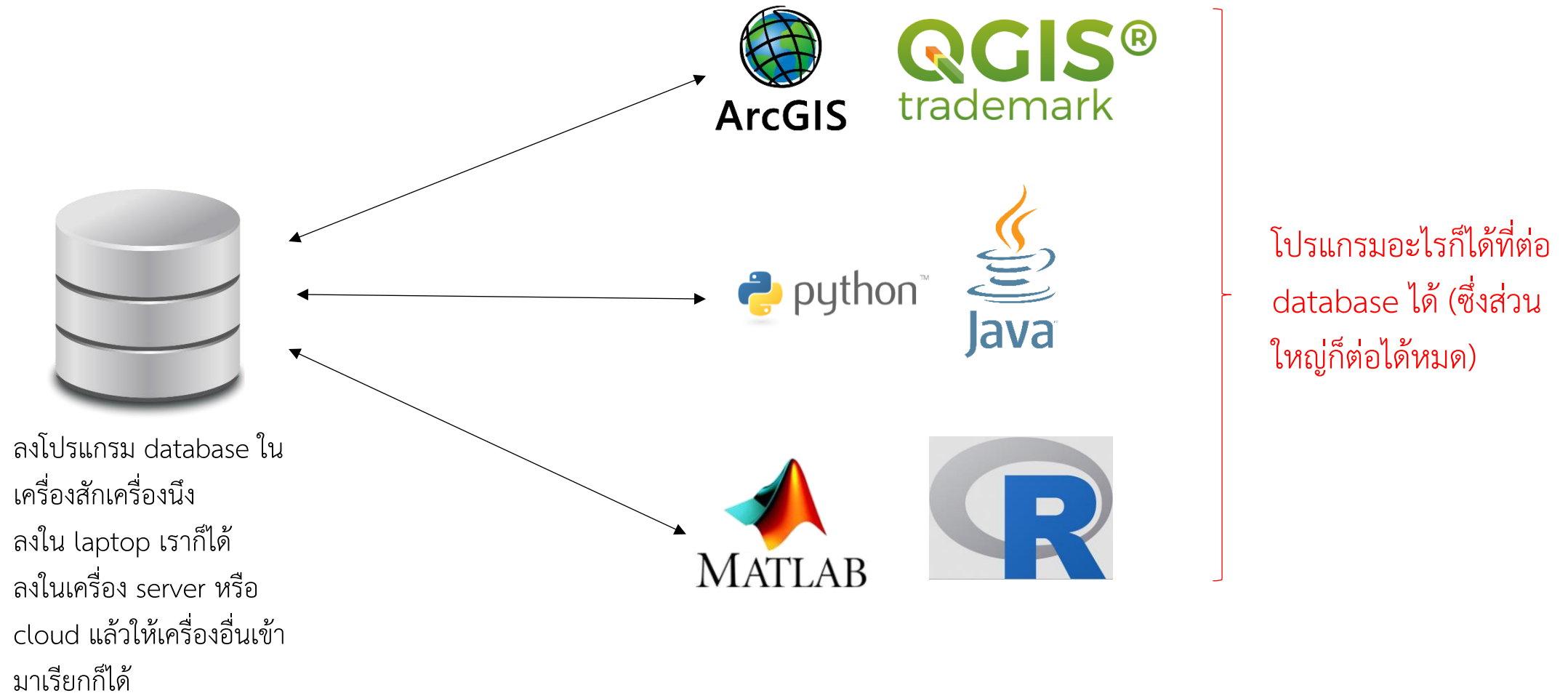
ข้อมูลคือตัวหนังสือ ตัวเลข รูปภาพ วิดีโอ
หรือแม้กระทั่ง geometry ก็ได้

- เป็นซอฟต์แวร์ที่ใช้จัดการข้อมูล
- จัดการข้อมูลขนาดใหญ่ได้ดีกว่าการเก็บเป็นไฟล์
- ให้คนเข้ามาใช้ข้อมูลร่วมกันได้ จัดการการเข้าถึงได้แบ่งกลุ่มผู้ใช้ได้
- มีการป้องกันการเพิ่มข้อมูลซ้ำ มีการรักษาความถูกต้องของข้อมูล
- ค้นหาได้เร็วมาก มีฟังก์ชันช่วยจัดการข้อมูลให้บริการไม่ต้องนั่งเขียนโปรแกรมอ่านทีละ byte เอง
- ใช้ภาษาที่เป็นมาตรฐาน ไม่ว่ายี่ห้อใดก็คล้ายกันหมด (สำหรับ RDBMS เท่านั้น)



จริงๆแล้วหากเราเปิด database ดูโดยใช้ File Explorer ก็จะเป็นไฟล์เยอะเยอะไปหมด แต่มันไม่รู้เรื่องเลย เปิดด้วย notepad ก็จะเป็นภาษาต่างดาว การจะรู้เรื่องต้องใช้โปรแกรม database ของที่สร้างไฟล์พวกนั้นขึ้นมาเท่านั้น

เวลาใช้ database เป็นยังไง



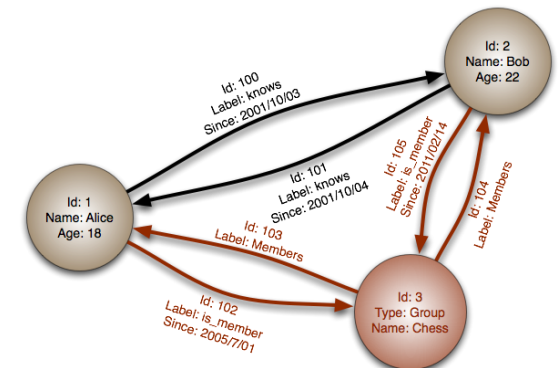
Types of Database

- Relational Database Management System (RDBMS)- Oracle, PostgreSQL, MySQL, SQLite, SQL Server, MariaDB, IBM DB2, MS Access
- NoSQL (Non SQL or Non relational) – MongoDB, Apache Cassandra
- NewSQL – ไม่รู้จักแล้วละ T_T
- อย่างอื่นก็มี เช่น Neo4J ซึ่งเก็บข้อมูลแบบ graph เหมาะกับเก็บความสัมพันธ์ของคน เป็นต้น



เป็น friend, mutual friend


รู้ไหมว่าประเภทที่กล่าวถึงนี้แบ่งตามอะไร?



รูปจาก wikipedia

RDBMS

- stands for “Relational Database Management System”
- Collect data as “relational model”


เก็บเป็นตารางนะแหละ

- Use SQL (pronounced /SEE-kwəl/) which stands for Structured Query Language
- SQL syntax- Very EASY and USEFUL!
- Introduction to SQL: <https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/v/welcome-to-sql>

PostgreSQL

- ฮิตมากในกลุ่ม GIS
- เป็น open source ใช้งานได้ฟรีไม่เสียเงิน
- Download ได้ที่ <https://www.postgresql.org/download/>
- <http://www.postgresqltutorial.com/>

SQLite

- เล็กกว่า PostgreSQL และอื่นๆ สะดวก
- อยู่ในโทรศัพท์มือถือ
- มีอยู่ใน Python, R แล้ว โหลด library ก็ใช้ได้เลย

ตาราง

- ทุก ๆ ตารางควรมี id ประจำแถว ซึ่งไม่ซ้ำกันแต่ละแถว ซึ่งจะเรียกว่าเป็น primary key
- ชื่อตารางและชื่อคอลัมน์เป็น case sensitive ไม่ควรมีการเว้นวรรค ถ้าอยากจะเว้นให้ใช้ underscore แทน
- แต่ละคอลัมน์ต้องกำหนดชนิดข้อมูลให้เหมาะสมและความจุ ควรกำหนดให้พอดี ไม่ขาดไม่เกิน เนื่องจากจะไม่กินพื้นที่ แล้วยังช่วยป้องกันการใส่ข้อมูลผิดอีกด้วย เช่น กำหนดให้คอลัมน์เก็บเลขประจำตัวนิสิตเป็นตัวอักษร ขนาดไม่เกิน 10 ตัว เป็นต้น

<u>groceries</u> 3 rows	
id (PK)	INTEGER
name	TEXT
quantity	INTEGER

ตาราง

- ชนิดข้อมูลในคอลัมน์จะต่างกันเล็กน้อยในแต่ละยี่ห้อ
- สำหรับ postgresql มีดังต่อไปนี้:
 - <https://www.postgresql.org/docs/current/datatype.html>

ที่ใช้บ่อย

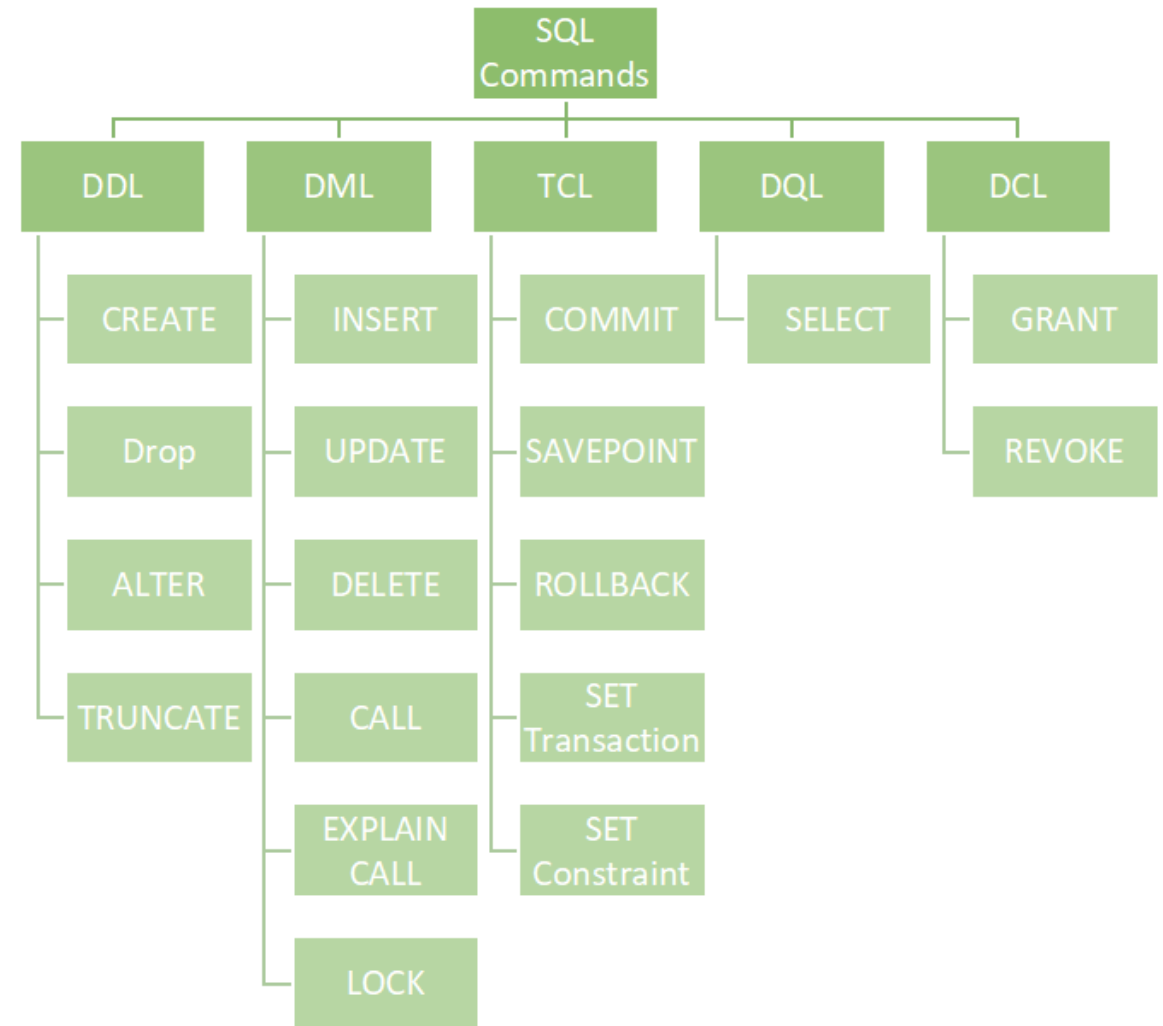
- integer
- NUMERIC(precision, scale) precision คือ จำนวนตัวเลขทั้งหมด นับรวมหลังจุดด้วย
- text ตัวหนังสือไม่จำกัดขนาด
- character(n), varchar(n)
- date, timestamp

SQL

คำสั่งแบ่งออกเป็นกลุ่ม ๆ ตามรูปด้านขวา

- Data Definition Language – สร้าง ลบ เปลี่ยน ตาราง ลบแถวออกหมดให้เหลือตารางเปล่าๆ
- Data Manipulation Language-จัดการข้อมูลในตาราง
- Transaction Control Language- จัดการคำสั่ง ทั้งการบันทึกคำสั่ง การย้อนกลับ และอื่น ๆ
- Data Query Language - เลือกข้อมูล
- Data Control Language – การให้สิทธิหรือ ถอนสิทธิผู้ใช้งาน

(เราจะเรียน SELECT ก่อนแล้วค่อยเรียนคำสั่งอื่น ๆ วันหลัง)



SQL

เลือกทุกคอลัมน์ ถ้าเลือกบางคอลัมน์ให้ใส่ชื่อ
คั่นด้วย comma

SELECT * **FROM** *my_table* **WHERE** *col1=100;*

เลือกข้อมูล

ชื่อ table

เงื่อนไขกรองข้อมูล

SQL

เลือกทุกคอลัมน์ ถ้าเลือกบางคอลัมน์ให้ใส่ชื่อ
คั่นด้วย comma

SELECT * **FROM** *my_table* **WHERE** *col1 IS NULL;*

เลือกข้อมูล

ชื่อ table

ดูว่ามีข้อมูลเป็น Null
หรือไม่

SQL

เลือกทุกคอลัมน์ ถ้าเลือกบางคอลัมน์ให้ใส่ชื่อ
คั่นด้วย comma

คำสั่งมหัศจรรย์
สำหรับค้นหาข้อมูลเวลาจำได้
แต่ไม่แน่ใจ

```
SELECT * FROM my_table WHERE col1 LIKE 'aus%';
```

เลือกข้อมูล

ชื่อ table

Like ต้องคู่กับสัญลักษณ์ดังต่อไปนี้

- % แทนตัวหนังสือหลายตัว
- _ แทนตัวหนังสือตัวเดียว

SQL

ชื่อ table
↓
UPDATE *my_table*
SET *column1 = value1, column2 = value2, ...*
WHERE *condition;*
↑
เงื่อนไขกรองข้อมูล

SQL

ชื่อ table

คอลัมน์ที่ต้องการใส่ค่า (หากไม่ได้กำหนดเงื่อนไข not null ในคอลัมน์ สามารถเลือกใส่บางคอลัมน์ได้)

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

ค่าในคอลัมน์ที่จะใส่ คั่นด้วย ,

SQL

ชื่อ table
↓
DELETE from *my_table*
WHERE *condition*;
↑
เงื่อนไขกรองข้อมูล

Basic Queries

- filter your columns
SELECT col1, col2, col3, ... **FROM** table1
- filter the rows
WHERE col4 = 1 **AND** col5 = 2
- aggregate the data
GROUP by ...
- limit aggregated data
HAVING count(*) > 1
- order of the results
ORDER BY col2

Useful keywords for **SELECTS**:

- DISTINCT** - return unique results
- BETWEEN** a **AND** b - limit the range, the values can be numbers, text, or dates
- LIKE** - pattern search within the column text
- IN** (a, b, c) - check if the value is contained among given.

Data Modification

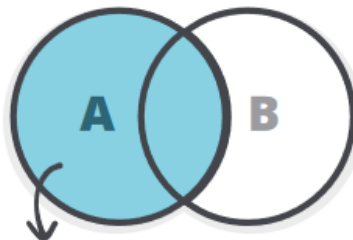
- update specific data with the **WHERE** clause
UPDATE table1 **SET** col1 = 1 **WHERE** col2 = 2
- insert values manually
INSERT INTO table1 (**ID**, **FIRST_NAME**, **LAST_NAME**)
VALUES (1, 'Rebel', 'Labs');
- or by using the results of a query
INSERT INTO table1 (**ID**, **FIRST_NAME**, **LAST_NAME**)
SELECT id, last_name, first_name **FROM** table2

Views

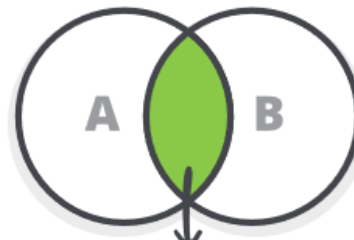
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS  
SELECT col1, col2  
FROM table1  
WHERE ...
```

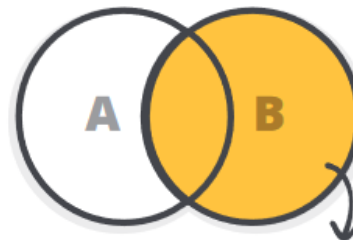
The Joy of JOINS



LEFT OUTER JOIN - all rows from table A,
even if they do not exist in table B



INNER JOIN - fetch the results that
exist in both tables



RIGHT OUTER JOIN - all rows from table B,
even if they do not exist in table A

Updates on JOINed Queries

You can use **JOINS** in your **UPDATES**

```
UPDATE t1 SET a = 1  
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id  
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINS

You can use subqueries instead of **JOINS**:

```
SELECT col1, col2 FROM table1 WHERE id IN  
(SELECT t1_id FROM table2 WHERE date >  
CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes

Avoid indexing on too many columns

Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

- convert strings to dates:
TO_DATE (Oracle, PostgreSQL), **STR_TO_DATE** (MySQL)
- return the first non-NULL argument:
COALESCE (col1, col2, "default value")
- return current time:
CURRENT_TIMESTAMP
- compute set operations on two result sets
SELECT col1, col2 **FROM** table1
UNION / EXCEPT / INTERSECT
SELECT col3, col4 **FROM** table2;

Union - returns data from both queries

Except - rows from the first query that are not present
in the second query

Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

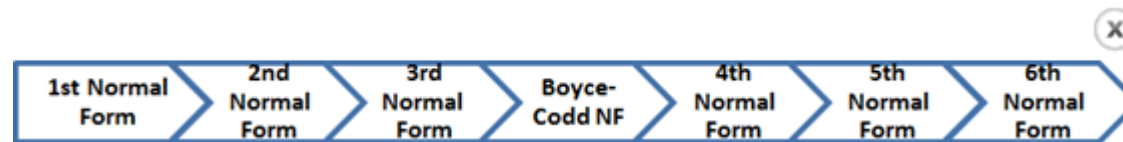
- COUNT** - return the number of rows
- SUM** - cumulate the values
- AVG** - return the average for the group
- MIN / MAX** - smallest / largest value

การออกแบบตารางเก็บข้อมูล

- เราจะจัดเก็บข้อมูลแยกเป็นตาราง โดยจะแยกข้อมูลที่เกี่ยวข้องกันเป็นกลุ่ม ๆ
- ตารางที่เกี่ยวข้องกันจะสามารถเชื่อมกันได้
- ลดความซ้ำซ้อนของข้อมูลเมื่อใส่ข้อมูล ประหยัดเนื้อที่ให้มากที่สุด
- สามารถเข้าถึงข้อมูลที่เกี่ยวข้องได้อย่างรวดเร็ว
- ช่วยให้การจัดการข้อมูลและรายงานข้อมูลให้เป็นไปอย่างสะดวกรวดเร็ว

Database Normalization

- เป็นแยกตารางเพื่อลดความซับซ้อน และลดปัญหาในการใส่ (insert) ปรับปรุง (update) หรือลบ (delete) ข้อมูลให้มากที่สุด
- มีเป็นชั้นๆ ให้ทำทีละชั้น



Problem ที่การทำ NF จะแก้ปัญหา

```
for row in conn.execute("SELECT * FROM Record_dust"):
    print(row)
```

(1,	'Bangkok',	24.0,	'20/08/2020 20:00:00',	1)
(2,	'Bangkok',	56.0,	'20/08/2020 21:00:00',	1)
(3,	'Bangkok',	12.0,	'20/08/2020 22:00:00',	1)
(4,	'Seoul',	34.0,	'20/08/2020 23:00:00',	1)
(5,	'Seoul',	12.0,	'21/08/2020 08:00:00',	1)
(6,	'Beijing',	47.0,	'21/08/2020 09:00:00',	1)
(7,	'Tokyo',	34.0,	'20/08/2020 20:00:00',	2)
(8,	'Tokyo',	156.0,	'20/08/2020 21:00:00',	2)
(9,	'Tokyo',	89.0,	'20/08/2020 22:00:00',	2)
(10,	'Bangkok',	23.0,	'21/08/2020 09:00:00',	2)
(11,	'Bangkok',	24.0,	'21/08/2020 10:00:00',	2)
(12,	'Singapore',	19.0,	'13/08/2020 08:00:00',	3)
(13,	'Singapore',	28.0,	'14/08/2020 09:00:00',	3)
(14,	'Singapore',	27.0,	'15/08/2020 10:00:00',	3)
(15,	'Jakarta',	34.0,	'11/08/2020 12:00:00',	4)
(16,	'Jakarta',	16.0,	'12/08/2020 13:00:00',	4)
(17,	'Tashkent',	44.0,	'13/08/2020 14:00:00',	4)
(18,	'Tashkent',	32.0,	'14/08/2020 15:00:00',	4)
(19,	'Honiara',	84.0,	'11/08/2020 12:00:00',	5)
(20,	'Honiara',	84.0,	'11/08/2020 13:00:00',	5)
(21,	'Honiara',	84.0,	'11/08/2020 14:00:00',	5)
(22,	'Berlin',	84.0,	'20/08/2020 14:00:00',	6)
(23,	'Berlin',	84.0,	'21/08/2020 14:00:00',	6)
(24,	'Berlin',	84.0,	'23/08/2020 14:00:00',	6)

ขอบคุณตัวอย่างจากนิติต

- ด้านซ้ายมือเป็นการออกแบบฐานข้อมูลเซ็นเซอร์ฝุ่น ซึ่งประกอบด้วยคอลัมน์
Dust_ID, Dust_location, Temp_Celsius, Date Time, Sensor_ID
- สมมติว่าใส่ข้อมูลเพิ่ม เซ็นเซอร์ในกรุงเทพฯ จะเห็นว่า มีปัญหาในการใส่คือ จะต้องพิมพ์ Bangkok เป๊ะๆ หากเป็น bangkok (b เป็นตัวเล็ก) หรือ กทม. (ใช้ภาษาไทย) ก็จะไม่เข้าพวกทันที ทำให้เวลา select จะมีปัญหา
- วิธีแก้ไขคือแยกตารางออกไป

รู้จัก primary key กับ composite key ก่อน

- **Primary key** เป็นคอลัมน์ที่เป็นตัวแทนของแถวแต่ละแถว โดย **primary key** ต้องเป็น **unique** ห้ามซ้ำกับแถวอื่น ๆ (เช่น ในตารางเก็บข้อมูลนิสิต นิสิตทุกคนจะถูก **assign ID** เป็นของตัวเองโดยที่ไม่มีใครซ้ำกันเลย แต่นิสิตสามารถมีชื่อ-นามสกุล หรือข้อมูลอื่น ๆ ซ้ำกันได้) **ID** นี้จะใช้โยงไปข้อมูลอื่น ๆ โดย **Primary key** ที่ไปอยู่ที่ตารางอื่นจะเรียกว่า **Foreign Key**
- **Composite key** เป็น **key** ที่จะ **unique** ได้ต้องรวมคอลัมน์มากกว่า 1 ขึ้นไป เช่น ชื่อ นิสิต อาจซ้ำกันได้ แต่เมื่อรวมชื่อกับนามสกุลแล้วจะไม่ซ้ำกันเลย *

ความสัมพันธ์ (Dependencies)

- ความสัมพันธ์ระหว่างคอลัมน์ต่าง ๆ ให้ดูว่ามีความสัมพันธ์กันหรือไม่ อย่างไร

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

- EMP_ID -> EMP_NAME (EMP_NAME ขึ้นกับ EMP_ID) เนื่องจาก ID สามารถบอกได้ว่าชื่ออะไร แต่ EMP_NAME อาจจะซ้ำกันได้
- EMP_ZIP -> (EMP_STATE, EMP_CITY) เนื่องจาก ZIP จะบอกได้ว่าเป็น STATE และ CITY อะไร แต่ STATE และ CITY อาจจะซ้ำกันได้

ขั้นแรก (1NF)

- ตารางที่ 1 คอลัมน์มีหลาย attribute ให้แยกมันออกมา

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

ขั้นที่ 2 (2NF)

- แยกตารางไม่ให้มี partial dependencies

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000
2	C5		

- ให้พิจารณาคอลัมน์ที่ไม่ใช่ Key ในที่นี้คือ COURSE_FEE ซึ่งขึ้นอยู่กับว่าจะเป็นวิชาไหน (COURSE_NO) แต่ไม่ได้ขึ้นกับ STUD_NO
- เขียนได้ว่า STUD_NO->COURSE_NO และ COURSE_NO->COURSE_FEE แต่ STUD_NO เกี่ยวอะไรกับ COURSE_FEE เลย
- ถ้าหากไม่แยกตาราง แล้วต้องการ update fee 1 วิชา จะต้อง update ทุกแถวที่มี COURSE_NO ที่ต้องการแก้ไข ซึ่งอาจผิดพลาดได้
- ดังนั้นจึงต้องทำการแยก COURSE_FEE ออกไปอีกตาราง

ขั้นที่ 3 (3NF)

- แยกตารางไม่ให้ซ้ำ

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

ความสัมพันธ์ที่มีคือ

- EMP_ID -> EMP_NAME
- EMP_NAME->(EMP_STATE,EMP_CITY)

แต่มีความสัมพันธ์อีกอันที่เกิดขึ้นคือ

EMP_ID -> (EMP_STATE, EMP_CITY) ก็คือถ้าเราทราบ ID พนักงาน เราจะทราบได้ว่าพนักงานคนนั้นอยู่ STATE และ CITY ไต โดยผ่าน EMP_NAME

ปัญหาที่จะเกิดขึ้นคือ เราต้องใส่ EMP_ZIP, EMP_STATE และ EMP_CITY ให้ถูกต้องทุกครั้ง สมมติว่า EMP_ZIP = 60007 แล้วเราใส่ EMP_CITY ผิดเป็น Norwich ก็จะขัดกับ record ก่อน ๆ ทันที

ดังนั้นเราต้องกำจัดปัญหานี้โดยการแยกตารางออกไป

EMPLOYEE table:

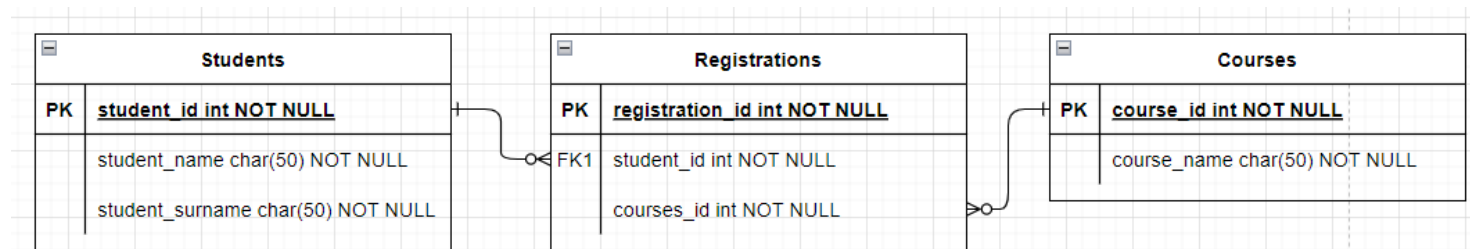
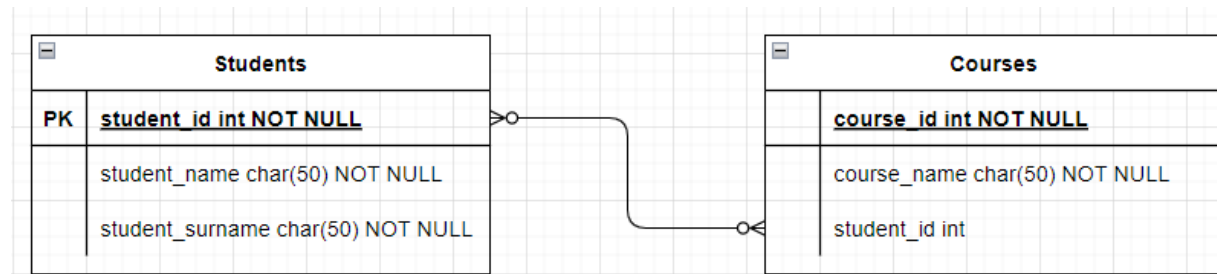
EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

การแก้ไขปัญหา many to many

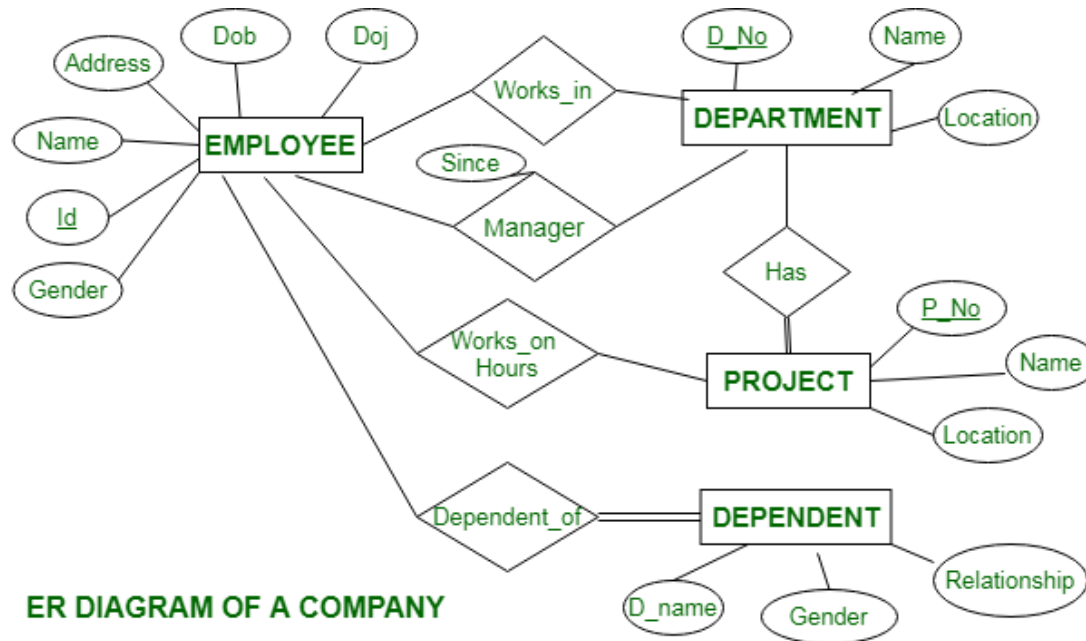
- นักเรียน 1 คน ลงทะเบียนได้หลายวิชา
1 วิชา มีนักเรียน หลายคน
- นิสิต 1 คนจองสนามกีฬาได้หลายสนาม
สนาม 1 สนาม มีคนจองได้หลายคน
- Many to many จะทำให้เกิดปัญหาความซ้ำซ้อน เพราะไม่สามารถตรวจสอบได้ เช่น
นักเรียน 1 คนลงทะเบียนได้หลายวิชา
ดังนั้น เราต้องใส่ชื่อนักเรียนคู่กับวิชาซ้ำๆ
หากสะกดชื่อนักเรียนผิด จะทำให้เกิดความผิดพลาดได้
- วิธีแก้คือมีตารางแทรกกลางระหว่าง 2 ตาราง



ER DIAGRAM

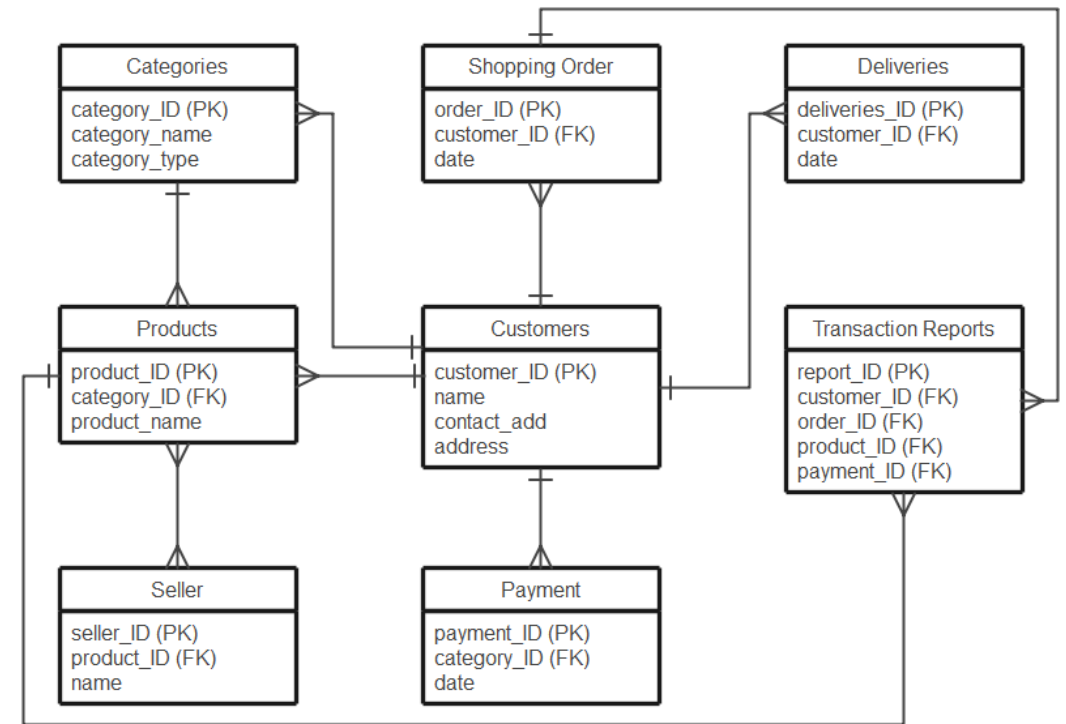
เอาแบบนี้

ERD for Online Shopping System



ER DIAGRAM OF A COMPANY

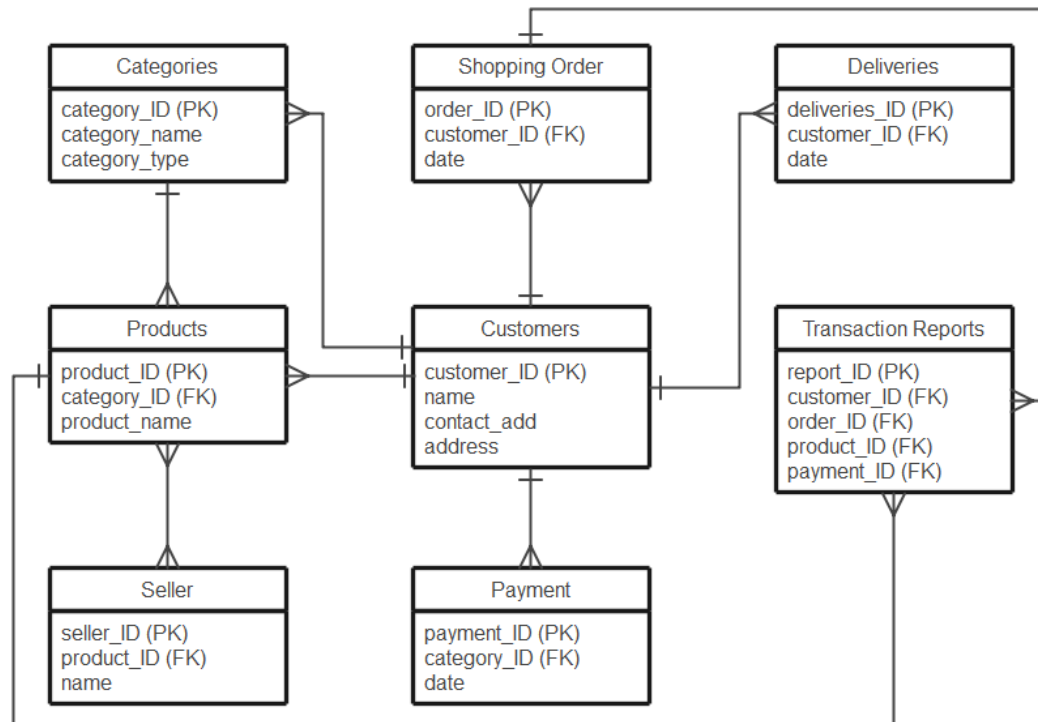
Conceptual diagram



Physical diagram







ER DIAGRAM

ERD for Online Shopping System



- ชื่อตาราง
- ชื่อคอลัมน์
- Primary Key/Foreign Key โดยจะต้องมี Foreign Key ทุกครั้งที่โยงเส้น
- 1 to Many or Many to 1

สัญลักษณ์ที่เส้น

	One
	Many
	One (and only one)
	Zero or one
	One or many
	Zero or many

เอกสารอ่านเพิ่มเติม

- <https://www.lifewire.com/transitive-dependency-1019760#:~:text=Transitive%20Dependency%20Example&text=If%20you%20know%20the%20book,t%20know%20the%20book%20name>.