Database Midterm

Edited 5
24/03/2017

Contents

Chapter 2 : Intro to Relational Model

table = file = relation

attributes = fields = columns

tuples = rows = records

Attribute Type

- *domain* คือ set ของค่าที่ value สามารถเป็นได้

- *null* ค่าพิเศษ โดยค่านี้จะเป็นสมาชิกของทุก domain

Relational Schema

- คือการเขียนโครงสร้างของ relation (เมื่อ A_1, A_2, ..., A_n เป็น attribute)

$$R = (A_1, A_2, ..., A_n)$$

ex. instructor = (ID, name, dept_name, salary)

? relation instance

Database

- ประกอบด้วย relation จำนวนมาก

?

Key

- Relation คือ set ของ Attribute, *Key* ก็คือสับเซตของ Relation

* 1 set มีหลาย subset => 1 Relation ก็มีหลาย key

* key เป็น set ของ attribute

- *Super key* คือ key ที่มีคุณสมบัติ unique (value ของ key ไม่มีวันซ้ำกัน)

ex. {std_id}, {citizen_id}, {first_name, last_name}, {std_id, first_name}

- *Candidate key* คือ super key ที่ใช้ attribute น้อยที่สุด

ex. {std_id}, {citizen_id}

- *Primary key* (PK) คือ key ที่เลือกมาจาก candidate key แล้วแต่ระบบ ส่วนมาเลือกอันที่ใช้บ่อยๆ

- *Foreign key* (FK) value ของ key นี้จะต้องมีใน relation อื่น

* FK จะเช็คข้อมูลที่ PK ของ relation ที่ reference ไป

Chapter 3 : Introduction to SQL

SQL

- ย่อมาจาก Structure Query Language
- แบ่งออกเป็น 2 ส่วน คือ DDL และ DML

DDL – Data Definition Language

- ใช้ในการสร้าง table ประกอบด้วย
- schema การกำหนด attribute ต่างๆ
- domain กำหนดขอบเขตของแต่ล่ะ attribute
- integrity constraints กำหนดความถูกต้องของข้อมูล
- authorization กำหนดสิทธิในการเข้าถึง

? indices

Domain Type

- char(n) ตัวอักษรความยาว n
- varchar(n) ตัวอักษรความยาว n (ต่างกับ char ตรงที่ว่าจะใช้ memory เท่าที่ข้อมูลมีจริงๆ)
- int จำนวนเต็ม
- numeric(p, d) จำนวนจริง p=จำนวนเลขนัยสำคัญ, d=จำนวนหลักทศนิยม (เหมาะกับข้อมูลที่ใช้คำนวณ)

Create Table

create table r (A_1 D_1, ... ,
A_n D_n,
integritry-constraint$_1$, ... ,
integritry-constraint$_k$)

Integrity Constraint

- ใช้เพื่อกำหนดความถูกต้องของข้อมูล
- *not null* กำหนดเพื่อบอกว่า attribute นี้จะมีข้อมูลที่เป็น null ไม่ได้
- *primary key(A_1, ... , A_k)* กำหนด primary key
- *foreign key(A_1, ... , A_k) references r_2* กำหนดเพื่อว่าข้อมูลของ A_1 - A_k จะต้องมีอยู่ใน r_2

* ข้อมูลที่จะเช็ค จะอ้างอิงจาก column ของ primary key ของ r_2
* attribute ที่กำหนดเป็น primary key จะมีคุณสมบัติเป็น not null ทันที

Drop and Alter Table

drop table r ลบตาราง r และข้อมูลทั้งหมด

delete from r ลบข้อมูลทั้งหมดใน r แต่ตารางยังอยู่

alter table ใช้แก้ไขโครงสร้างของตาราง

- alter table r **add** A D

- เพิ่ม attribute A ด้วย domain D
- row ที่อยู่ก่อนจะเพิ่ม จะมีค่า attribute นี้เป็น null

- alter table r **drop** A

- ลบ attribute A ออกจาก r

DML – Data Manipulation language

- ใช้ในการเรียกดูและแก้ไขข้อมูล

SELECT – FROM – WHERE

- คำสั่งพื้นฐานที่ใช้ในการเรียกดูข้อมูล

```
select A1, A2, ... , An
form r
where p
```

- A_n คือ column ที่ต้องการ (เลือก column)
- r คือ relation ที่จะนำมา (เลือก table)
- p คือเงื่อนไขของแถวที่ต้องการ (เลือก row)

SELECT Clause

- select มีอยู่ 2 แบบ
  - **select all** A_1, A_2, ... เลือกทุก row (**select** ปกติก็ถือเป็น **select all**)
  - **select distinct** A_1, A_2, ... ถ้ามี row ไหนข้อมูลซ้ำกันจะตัดออก
- สามารถใช้ * แทน attribute ทั้งหมดได้ **select ***
- ถ้าใน from มี relation มากกว่า 1 แล้วเกิดมี attribute ชื่อเหมือนกัน สามารถใช้ r.A เพื่ออ้างได้

  ex. **select** student.id, teacher.id
  from student, teacher
- ประโยค select สามารถทำการคำนวณได้ด้วย **select** salary/12

FORM Clause

- จากรูปแบบ **form** r ซึ่ง r ตัวนี้อาจเป็น table เดียวหรือเกิดจากการผสม table หลายอันก็ได้

<u>แบบ table เดียว</u>

- ex. **form** instructor

<u>แบบผสมตาราง</u>

- บางทีข้อมูลที่เราต้องการ(attribute) ทั้งหมดก็ไม่ได้อยู่ในตารางเดียวกัน

เช่น อยากได้ชื่ออาจารย์กับรายวิชาที่สอน แต่ชื่ออาจารย์อยู่ใน instructor.name และรายวิชาอยู่ใน teaches.course_id ดังนั้นเลยต้องผสมตาราง instructor และ teaches เข้าด้วยกันก่อน

Cartesian Product

Natural Join

Rename Operation

- ใช้ในการเปลี่ยนชื่อ attribute หรือ table
- นอกจากการเปลี่ยนชื่อเพื่อแสดงผลแล้ว ยังสามารถใชชื่อที่เปลี่ยนในประโยคอื่นได้ด้วย

ex. **select** i.name as 'teacher_name', t.course_id
form instructor as i, teaches as t
where i.salary > 10000

* การเปลี่ยนชื่อนี้ไม่ส่งผลต่อโครงสร้าง เปลี่ยนเพื่อใช้ชั่วคราวเท่านั้น

Where Clause

- เป็นการเลือก row ด้วยการเช็คเงื่อนไข row ที่ผ่านเงื่อนไขเท่านั้นที่จะแสดงผล

ex. **where** instructors.salary > 100000

* where ก็เหมือนกับ if ประโยคข้างหลังต้องมีค่าเป็น true false เท่านั้น
* attribute ที่ไม่ได้ select ก็สามารถใช้เป็นเงื่อนไขใน where ได้นะ

Logical operator

- สามารถใช้ **and or not** ใน where ได้

Between operator

- มี **between** ใช้เช็คช่วงของค่า

ex. **where** salary **between** 10000 **and** 20000

Comparison

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

Tuple Comparison

- สามารถเปรียบเทียบหลายข้อมูลพร้อมกันได้

ex. **where** (instructor.ID, dept_name) = (teaches.ID, 'Biology')

String Operation

- สามารถตรวจสอบ string ตามรูปแบบได้ด้วย **like** pattern

where A **like** pattern

- pattern จะใช้เครื่องหมาย 2 ตัวเพื่อสร้าง pattern

- % แทนตัวอักษรหลายตัว

- _ แทนตัวอักษร 1 ตัว

ex. 'Intro%' ข้อความที่ขึ้นต้นด้วย Intro

'_ _ _' ข้อความที่มี 3 ตัวอักษร

'_ _ _%' ข้อความที่มีอย่างน้อย 3 ตัวอักษร

Set Operation

SFW **union** SFW
SFW **intersect** SFW
SFW **except** SFW

* ขอย่อ select-form-where ว่า SFW นะ ขก.พิมพ์ล่ะ

* จำนวน column ต้องเท่ากัน

- ปกติแล้ว set operation จะกำจัด row ที่ซ้ำให้อัตโนมัติ

- ถ้าหากไม่ต้องการ ให้ใช้ **union all, intersect all, except all** แทน

Null Value

- ค่าพิเศษ **null** แสดงถึงการไม่รู้ค่านี้

- สามารถตรวจสอบค่า null ได้ด้วย **is null**

ex. **where** salary **is null**

- arithmetic operation => null 0*null = null

- comparison => null

- logical คู่อื่นจะได้เป็น null หมด ยกเว้น

- null **or** true => true

- null **and** false => false

Aggregate Function

- เป็นฟังก์ชั่นที่ใช้เพื่อหาค่าต่างๆ

- วิธีใช้ **select aggregate**(A) ; A = attribute

- **avg, min, max, sum, count**

- ปกติแล้ว aggregate function จะทำงานกับทุก row แต่สามารถแยกกลุ่มทำงานได้ โดยใช้ **group by**

ex.	**select avg**(salary)	**select** dept_name, **avg**(salary)
	from instructor	**from** instructor
		group by dept_name
	หาเงินเดือนเฉลี่ยอาจารย์ทั้งหมด	หาเงินเดือนเฉลี่ยอาจารย์แต่ล่ะแผนก

* ไม่ควร select attribute อื่น นอกจาก attribute ที่ใช้ group by

* group by สามารถใช้มากกว่า 1 attribute ได้ ex. **group by** dept_name, building

- aggregate function ไม่สามารถใช้ where ปกติได้ ให้ใช้ **having** แทน

ex. **having avg**(salary) > 10000

* สามารถใช้ where ร่วมได้ แต่ต้องมาก่อน group by

select dept_name, **avg**(salary)

from instructor

where salary > 10000

group by dept_name

having avg(salary) > 20000

= หาแผนกที่เงินเดือนเฉลี่ยของ อาจารย์ที่มากกว่า10000 มีค่ามากกว่า 20000

- aggregate ทุกตัวจะมองข้าม row ที่เป็น null จะไม่ใช้คำนวณ

- ยกเว้น **count(*)** จะนับ row ที่มี null ด้วย

- ถ้าข้อมูลทั้งหมดเป็น null => count = 0, aggregate = null

Sub queries

- คือการใช้ SFW หลายชั้น

sub queries in from

- การทำงานคือจะสร้างตารางจาก SFW ข้างในก่อน แล้วค่อยนำมาสร้างตารางด้วย SFW ข้างนอกอีกที

ex. **select** dept_name

form (**select** dept_name, **avg**(salary) **as** avg_salary

from instructor

group by dept_name)

where avg_salary

sub queries in where

- ส่วนมากใช้เพื่อตรวจสอบว่า row ของ SFW ชั้นนอก ตรงกับเงื่อนไขใน SFW ข้างใน

* attribute ที่ใช้ตรวจสอบเงื่อนไข ควรเท่ากับ attribute ของ sub query

ฟังก์ชั่นที่ใช้ร่วมกับ sub query ใน where (จะได้ค่า true/false ออกมา)

ให้ A เป็น attribute หรือ set ของ Attribute (A_1, ... , A_n)

- A **in** r

เช็คว่า A อยู่ใน r => มีตอบ true

- A **not in** r

เช็คว่า A ไม่อยู่ใน r => ไม่มีตอบ true

- A <comp> **all** r

เปรียบเทียบ A กับข้อมูลทั้งหมดใน r => จริงทั้งหมด ตอบ true

- A <comp> **some** r

เปรียบเทียบ A กับข้อมูลทั้งหมดใน r => จริงตัวเดียวก็ตอบ true

- exist r

เช็คว่า r มีข้อมูลอยู่ => ถ้า r มีข้อมูลสักแถวเดียวก็ตอบ true

- **not exist** r

with clause

- เป็นการประกาศโครงสร้างชั่วคราว เพื่อใช้ในโปรแกรม

with r (A_1, ... , A_n) **as** (SFW)

ex. **with** dept_total (dept_name, value) **as**

(**select** dept_name, **sum**(salary)

from instructor

group by dept_name

- สามารถนำโครงสร้างนี้ไปใช้ใน from ของ query อื่นได้

ex. **select** dept_name

from dept_total

where value > 100000

* with ก็เหมือนกับการประกาศ function ในโปรแกรม

Scalar Sub query

- sub query ที่ได้ออกมาค่าเดียว เช่น พวก aggregate ที่ทำกับข้อมูลทั้งหมด

- สามารถนำมาเป็น attribute ได้

ex. **select** dept_name,

(**select count**(*)

from instructor **as** I

where D.dept_name = I.dept_name)

from department **as** D

- สามารถใช้เพื่อเป็นค่าในการตรวจสอบเงื่อนไขของ where ได้

select name

from instructor **as** I

where salary * 10 >

(**select** budget

from department **as** D

where D.dept_name = I.dept_name)

Modification of Database

Deletion

- ใช้เพื่อลบ row ออกจาก table

```
delete from r
where P
```

r = table ที่จะลบข้อมูล

P = เงื่อนไขในการลบข้อมูล

ex. ลบอาจารย์ที่เงินเดือนน้อยกว่า 10000

```
delete form instructor
where salary < 10000
```

* ถ้าไม่มีประโยค while ก็จะลบข้อมูลทั้งหมดใน table

* ระวังถ้าเงื่อนไขใน while ขึ้นกับ aggregate function ใน table เดียวกัน อาจผิดพลาดเพราะจะทำการคำนวณ aggregate ใหม่ทุกครั้ง ถ้าหากลบ row ไปก็จะทำให้ ค่าขง aggregate เปลี่ยน

ex. **delete from** instructor

where salary < (**select avg**(salary) **from** salary)

Insertion

- แบบไม่ระบุลำดับของ attribute value ที่ใส่เข้าไป จะแมชท์กับลำดับของ attribute ที่ประกาศเอาไว้

```
insert into r
values (v1, v2, ... , vn)
```

- แบบระบุลำดับของ attribute

```
insert into r (A1, A2, ... , An)
values (v1, v2, ... , vn)
```

* ระวังถ้าใส่ value ไม่ครบตามจำนวน attribute จะเกิด error ถ้าไม่รู้ค่าของตัวไหนก็ใส่ null ไป

- สามารถใช้ query แทน values ได้

```
insert into r1
select A1, ... , An
from r2
```

* attribute ที่ select มาจะต้องเท่ากับ attribute ของ r_1

* การ insert ถ้าทำให้ primary key ซ้ำจะเกิด error ได้

Modification

- set ค่าให้กับ attribute ต่างๆ (เลือกเฉพาะบาง attribute ที่จะแก้ไข)

```
update r
set A1=v1, ... , An=vn
```

ex. **update** instructor

set salary = salary * 1.3

- เลือก row ที่จะ set ค่าใหม่โดยใช้ **where**

```
update r
set A1=v1, ... , An=vn
where P
```

- set ค่าด้วยเงื่อนไข โดยใช้ **when then else**

```
update r
set A1 = case
        when P
        then salary = v11
        else salary = v12
        end
```

chapter 4 : Intermediate SQL

Join operation

- ใช้ในการรวม 2 table กลายเป็น table ใหม่
- ส่วนมากใช้ใน **from**
- ประกอบด้วย 2 ส่วนคือ
    - join type ใช้เพื่อระบุข้อมูลที่จะแสดงออกมาหลังจากการ join
    - join condition ใช้เพื่อระบุเงื่อนไขในการ join

join type

- **cross join** A x B
- **inner join** A intersect B
- **left outer join** A union (A intersect B)
- **right outer join** B union (A intersect B)
- **full outer join** A union B

join condition

- **natural** เลือก match ด้วย column ที่ชื่อเหมือนกันทั้ง 2 ตาราง
- **on** P เลือก math โดยใช้เงื่อนไข
- **using**(A_1, A_2, ... , A_n) เลือก match โดยระบุ column ที่ชื่อเหมือน (กรณีไม่ต้องการใช้ทุก column ที่ชื่อเหมือนแบบ natural)

r_1 *join_type* r_2 **on** P
r_1 *join_type* r_2 **using**(A_1, ... , A_n)
r_1 **natural** *join_type* r_2

Views

- เหมือนกับ with ต่างกันตรงที่ view จะถูกเก็บไว้ในระบบ dbms ด้วย
- views นั้นไม่ได้เก็บตาราง แต่จะเก็บเพียง query expression (SFW)
- ทุกครั้งที่เรียกใช้ view จะต้อง execute ใหม่ทุกครั้ง

```
create view view_name as
SFW
```

- สามารถสร้าง view จาก view อีกทีได้

* view มักใช้กับการที่จะแสดงเพียงข้อมูลบางส่วนให้กับ user บางกลุ่มเท่านั้น

- ถึงจะ insert ได้แต่ก็มีข้อแม้ว่าจะต้องเป็น view แบบกระจอก ซึ่งมีสมบัติดังนี้
    - **from** มาจาก table เดียว
    - **select** มีแค่ attribute ไม่มี expression, aggregate, **distinct**
    - attribute ที่ไม่ได้ select มาจะถูก set เป็น null เมื่อ insert ค่าใหม่
    - ห้ามมี **having** และ **group by**

Transactions

Integrity Constraint

- เรื่องเก่าเหละ แต่จะมาพูดเพิ่มอีกตัวนึง
- *check (P)*
    - ใช้สำหรับตรวจสอบเงื่อนไขของ attribute แต่ล่ะตัว
    - ส่วนมากใช้เพื่อจำกัด set ของ value หรือ range ของ values

    ex. **check** (semester **in** (‘Fall’, ‘Winter’, ‘Spring’, ‘Summer’))

    ex. **check** (minutes >= 0 **and** minutes <= 60)

Index

- คือ B-tree ของข้อมูลต่างๆใน attribute ที่กำหนด ใช้เพื่อช่วยให้ค้นหาข้อมูลได้เร็วขึ้น
- primary key จะถูกสร้างเป็น index อยู่แล้วโดยอัตโนมัติ

```
create index index_name on r(A)
```

User-defined type

- เป็น type ที่สร้างขึ้นเองจาก type ที่มีอยู่

```
create type type_name as domain final
```

User-defined domain

- เป็น domain ที่สร้างขึ้นมาเอง
- ต่างจาก type ตรงที่จะมีพวก integrity constraint ได้ (not null, ...)

```
create domain type_name as domain final
```

Authorization

- การกำหนดสิทธิในการเข้าถึงหรือจัดการข้อมูล

- สิทธิเกี่ยวกับจัดการข้อมูล database
  - **Read** สามารถอ่านได้
  - **Insert** สามารถเพิ่มข้อมูลได้
  - **Update** สามารถแก้ไขข้อมูลได้
  - **Delete** สามารถลบข้อมูลได้
- สิทธิเกี่ยวกับการจัดการโครงสร้าง database
  - **Index** สามารถสร้าง/ลบ index ได้
  - **Resources** สามารถสร้าง relation ใหม่ได้
  - **Alternation** สามารถ เพิ่ม/ลบ attribute ได้
  - **Drop** สามารถลบ relation ได้

การมอบสิทธิ

```
grant privileges list
on relation / view
to user list
```

การถอนสิทธิ

```
revoke privilege list
on relation / view
from user list
```

privilege

- ความสามารถที่จะกำหนดให้แต่ล่ะ บุคคล (ตำแหน่ง)

    - **select , insert , update , delete , all privilege**

Roles

- เป็นการสร้างกลุ่มคนเพื่อให้ง่ายต่อการมอบสิทธิ

create role *role name*

ex. **create role** student

- ใช้ grant เพื่อมอบตำแหน่งให้คนต่างๆ

    grant student **to** Natthapach

- ใช้ role แทน user-list ในการมอบสิทธิ

    grant select **on** course **to** student

- สามารถมอบ role ให้ role ได้

    grant lib_member **to** student

* สมมติเรา log in เข้าไปใน dbms เราจะมอบสิทธิให้คนอื่นได้เฉพาะสิทธิที่เรามีเท่านั้น

chapter 5

Trigger

- คือคำสั่งที่สร้างไว้เพื่อทำงานกับเหตุการณ์บางอย่างที่จะเกิดขึ้นในอนาคต
- อารมณ์คล้ายๆ ActionListener ใน java (จำไม่ได้ก้ช่างมันเถอะ 55555)

```
create trigger trigger_name {after|before} {insert|delete|update} on relation
referencing {new|old} row as  temp_row
for each row
when (...)
begin
        ...
end;
```

- update สามารถเจาะจง attribute ที่จะเกิดเหตุการณ์ได้ด้วย

```
create trigger trigger_name {after|before} update of relation on attribute
```

- บรรทัด referencing มีไว้ใช้เพื่ออ้างอิง row ที่มีการเปลี่ยนแปลง

    referencing new row as ใช้กับ insert, update

    referencing old row as ใช้กับ delete, update

ตอนที่ไม่ควรใช้ trigger

- ขก.แปลล่ะ

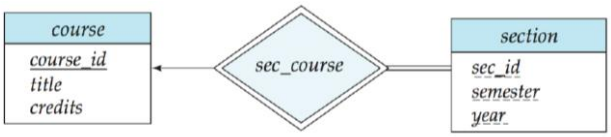
chapter 7 : Entity – Relationship Model

Entity มี 2 แบบ

- Strong Entity คือ entity ที่มี Primary Key

- Weak Entity คือ entity ที่ไม่มี Primary Key ต้องอาศัย Primary Key ของคนอื่นรวมกับตัวเอง

- Key ที่เอาไว้รวมกับ PK ของคนอื่น เรียกว่า Partial Key



* ใช้ ___ ขีดเส้นใต้ตัวที่เป็น Primary Key

* ใช้ _ _ _ ขีดเส้นใต้ตัวที่เป็น Partial Key

* relation ที่ weak entity จะไปยืม Primary key มาจะใช้เส้น 2 ชั้น (อย่าไปจำสับสนกับเส้น total)

(เดี๋ยวอธิบาย total อย่าเพิ่งถามว่าคือไร)

Attribute มี 4 แบบ (จำด้วยเด๋วต้องใช้ตอนแปลง)

- Simple Attribute attribute ปกติ

- Composite Attribute attribute ที่ประกอบด้วย attribute ย่อยอีกทีนึง (ใช้การเยื้อง บ่งบอก)

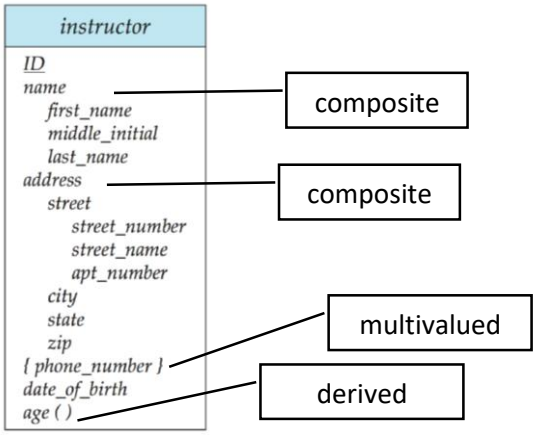
ex. name ประกอบด้วย first name, last name

- Multivalue Attribute attribute ที่เป็นไปได้หลายค่า (ใช้ { } บ่งบอก)

ex. phone number คนหนึ่งคนสามารถมีหลายเบอร์ได้

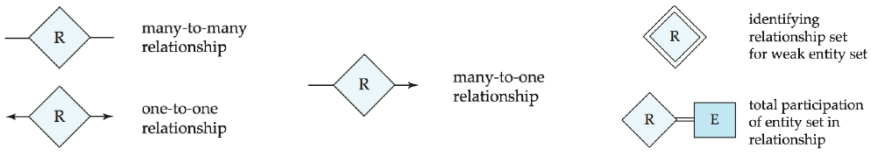
- Derived Attribute attribute ที่สามารถคำนวณได้จาก attribute อื่น (ใช้ () บ่งบอก)

ex. ถ้าใน entity นั้นเก็บ birth date แล้ว age ก็จะเป็น Derived เพราะสามารถคำนวณได้จาก birth date



Relation

- อธิบายความสัมพันธ์ของแต่ล่ะ



อิแปลง ER-Diagram to Schema

มีหลักๆ 3 ขั้น 1.แปลง entity 2. แปลง attribute 3. แปลง relation

1. แปลง entity

- strong entity แปลงเป็น schema ได้เลย attribute ก็ตามนั้นเหละ
- weak entity เก็บ PK ของ strong entity ตัวที่

2. แปลง attribute

- simple attribute ใช้ตามเดิม
- composite attribute ตัดหัวออก เหลือแค่ข้างใน
- derived attribute ตัดทิ้ง
- multivalued สร้าง schema ใหม่ – เก็บ PK ของ entity และ ข้อมูล (ใช้ทั้ง2อย่างเป็น PK ของตัวเอง)

3. แปลง relation

- many-to-many สร้าง schema ใหม่ เก็บ PK ของ entity ทั้ง 2 ฝั่ง
- many-to-one/one-to-many เก็บ PK ของฝั่ง one ไว้ที่ many ด้วย

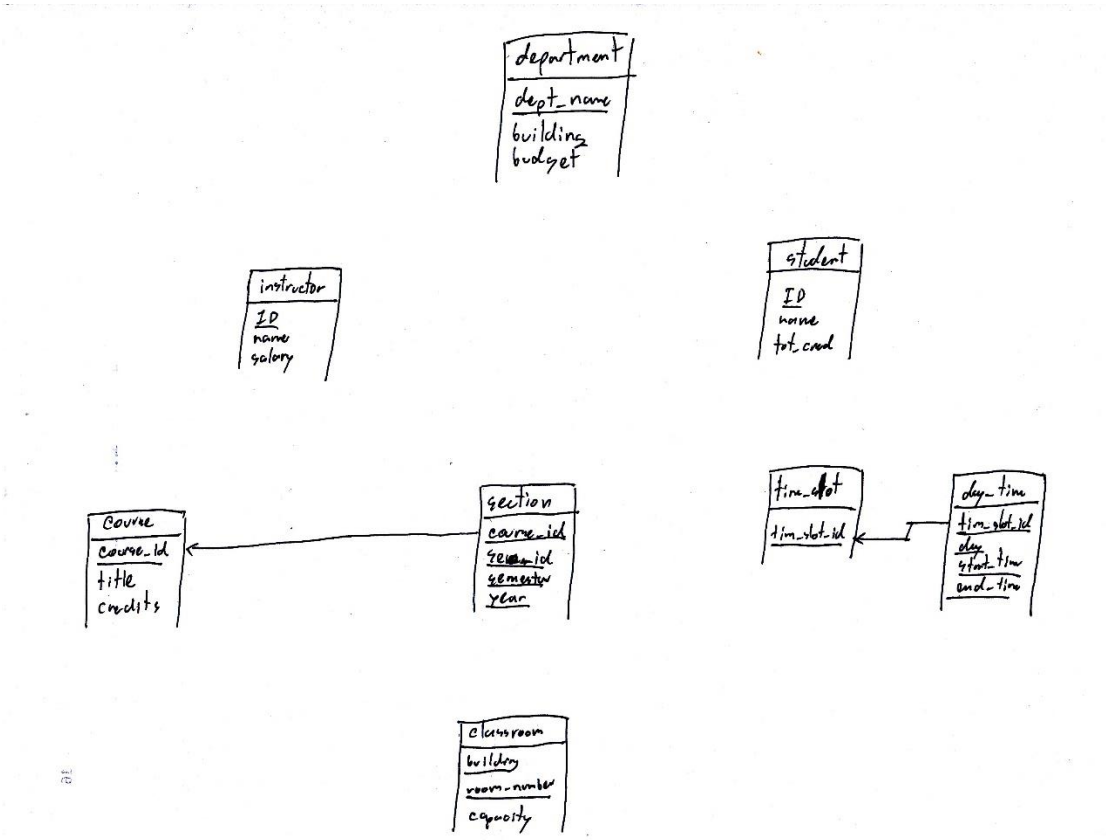
* ถ้า ฝั่ง many ไม่ใช่ total จะทำให้เกิด null value ได้

* ถ้าแคร์เรื่อง null ให้สร้าง schema ใหม่แบบ many-to-many

- one-to-one ทำแบบ one-to-many สมมติให้ฝั่งไหนก็ได้เป็น many แล้วเก็บ PK อีกฝั่งไว้

แปลงจากชีทหน้า 7.34

step 1 แปลง entity กับ attribute ตามหลักให้เรียบร้อย



* หลักการโยงเส้น FK คือถ้าไปเอา PK ของเค้ามา ก็โยงกลับไปหาเค้าด้วยนะ

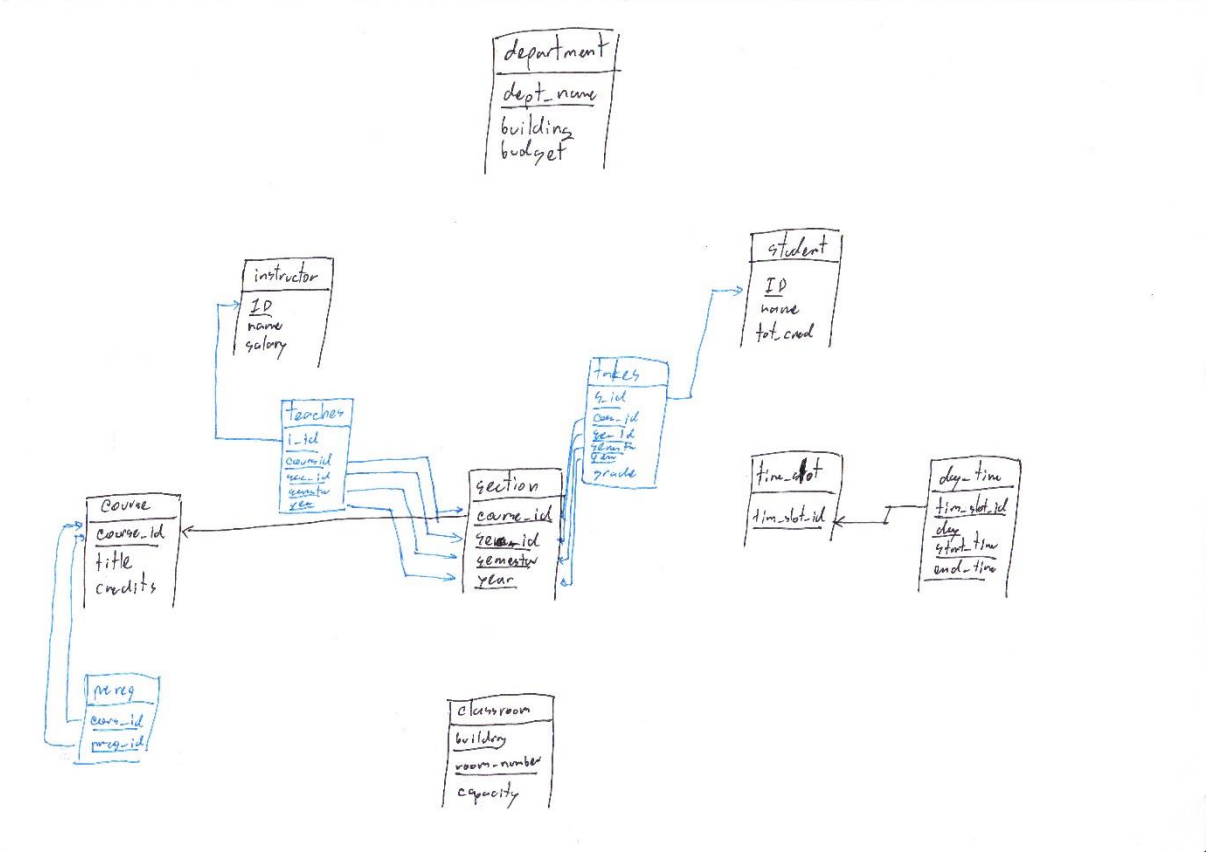
* อย่าลืมการแปลง weak entity ดูดีๆว่า double diamond อยู่ที่ฝั่งไหนถึงจะไปเอา PK ของฝั่งนั้น

ex. section เป็น weak entity และมี double diamond อยู่ทาง course เลยไปเอาของ course

* อย่าลืม multivalued แยก schema ออกมาแล้วเก็บ PK และโยงเส้น FK ด้วยนะ

Trick อย่าเพิ่งลากเส้นปิดข้างล่าง เพราะเด๋วอาจเพิ่ม attribute เข้าไปอีกตอนแปลง relation

step 2 แปลง many-to-many ก่อนเลย อันนี้ง่ายสุด



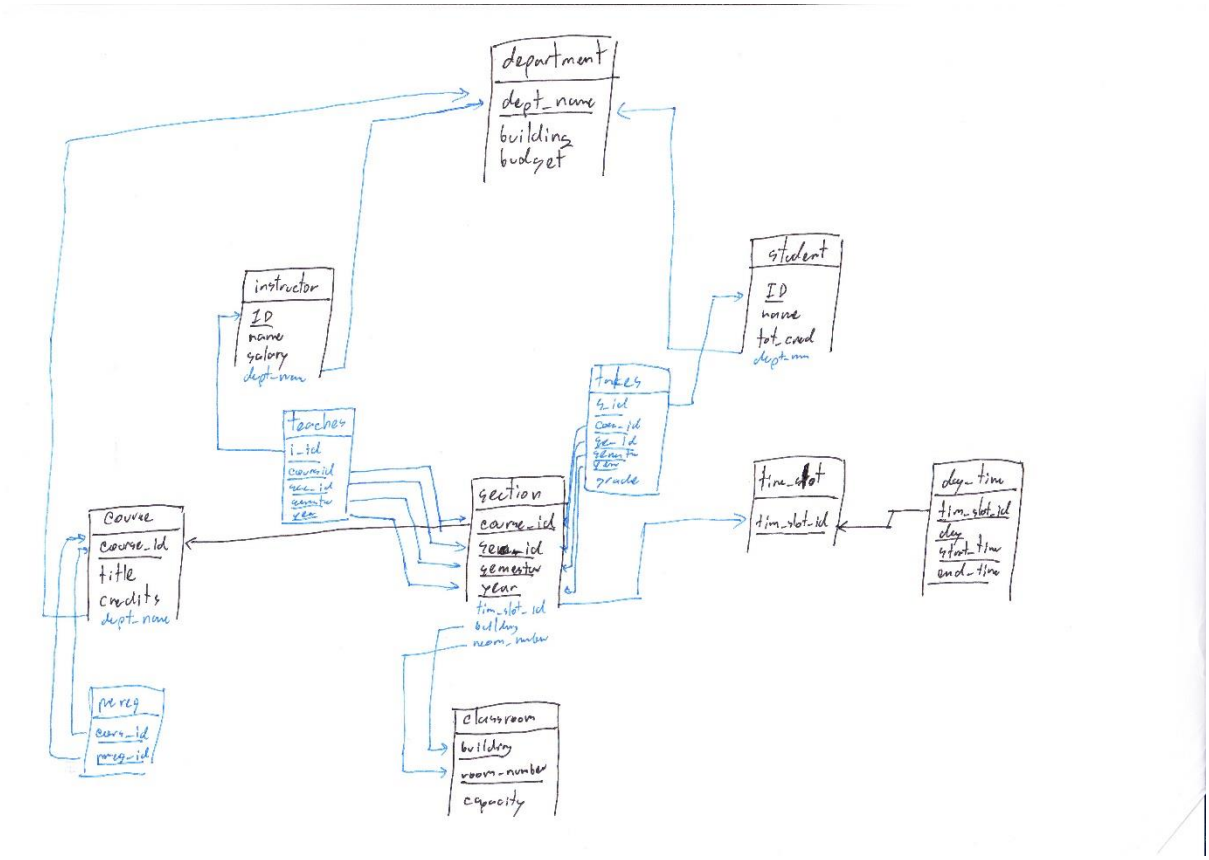
* อย่าลืมว่า schema ใหม่ที่สร้างมาใช้ PK ของ entity ทั้งสองตัวเป็น PK ของตัวเอง

* โยงเส้นด้วยนะ

* ถ้า entity ที่มี attribute อย่าลืมใส่ไป แต่ไม่ต้องให้เป็น PK นะ

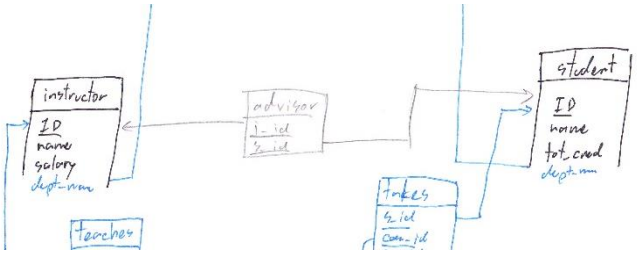
ex. takes มี attribute grade

step 3 แปลง one-to-total many / one-to-one



* เอา PK เค้ามาใส่ก็อย่าลืมโยงเส้นล่ะ

step 4 แปลง one-to-many



อันนี้บอกไปแล้วว่าทำได้ 2 แบบ แบบเก็บ PK ไว้ที่ many แบบ one-to-total many ก็ได้

หรือจะแยกออกมาแบบ many-to-many ก็ได้

เลือกเอา หรือไม่ก็ถามจารย์ว่ามี null value ได้มั๊ย

step 5

ขีดเส้นปิดข้างล่างด้วย กูขก.สแกนล่ะ