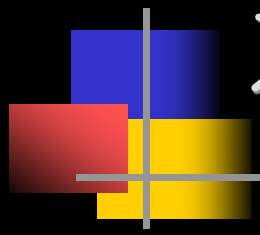
Stockholm, Sweden, by Frank S.C. Tsang
(http://www2.nkfust.edu.tw/~imfrank)

第九章　邏輯資料庫設計：關聯表的正規化



本章內容

- 9.1 簡介
- 9.2 1NF、 2NF 、 3NF 與 BCNF
- *9.3 多重值相依性與第四正規化型式 (4NF)
- *9.4 合併相依性與第五正規化型式 (5NF)
- 9.5 正規化的步驟
- 9.6 利用個體-關係模式來做邏輯資料庫設計
- 9.7 結論

簡 介

- BOB 資料庫為什麼分別放在三個關聯表中？
- 為什麼不放在四個關聯表中或兩個關聯表中？
- 到底有那些欄位要放在一個關聯表中？
- 那些有一定要分開放？
- 「邏輯資料庫設計」—正規化理論
- 邏輯資料庫設計是系統分析的重要步驟之一

為什麼要正規化？

- 關聯表的正規化做法，是將關聯表做適當分割
- 原因: 沒有正規化會造成所有資料擠在同一個關聯表, 容易有資料重覆儲存的浪費情形
  也會造成「更新時的異常現象」(Update Anomalies)
- 但是也會犧牲某些效能, 如下頁說明...

正規化的代價：魚或熊掌?

- 正規化讓資料分別放在多個關聯表其效率則會較差
- 正規化層次太高也會造成效率降低
- 可配合「反正規化技巧」(De-normalization) 提高效率
- 所以, 要不要正規化?或是正規化到何種程度?的問題實在是見仁見智, 也是設計資料庫的一項重要考量

更新異常現象說明

BO

no	name	rank	city	id	quantity
1	巨蟹書局	20	臺北市	1	30
1	巨蟹書局	20	臺北市	2	20
1	巨蟹書局	20	臺北市	3	40
1	巨蟹書局	20	臺北市	4	20
1	巨蟹書局	20	臺北市	5	10
1	巨蟹書局	20	臺北市	6	10
2	射手書局	10	高雄市	1	30
2	射手書局	10	高雄市	2	40
3	水瓶書局	30	新竹市	2	20
4	天秤書局	20	臺中市	2	20
4	天秤書局	20	臺中市	4	30
4	天秤書局	20	臺中市	5	40

更新異常現象說明

- **新增**:如果 “雙魚書局” 尚未訂購任何書籍的話，那麼新增下列資料會有困難

6	雙魚書局	40	中壢市	—	—

- 這樣會使得主鍵 (*no, id*) 中的*id* 為虛值，違反了關聯式資料模式中所規定的「個體整合限制規則」(Entity Integrity Rule)。

更新異常現象說明

- **刪除**：刪除 “水瓶書局” 訂購編號 2 號書籍的事實，由於只有單筆資料，所以會連 “水瓶書局” 的其它資訊都刪除了，造成 “水瓶書局” 是位於 “新竹市” 的資訊也失去了。

3	水瓶書局	30	新竹市	2	20

更新異常現象說明

- **更新**: 更改 “巨蟹書局” 的*city* 成為 “花蓮市” 時，就必須一次將底下的六筆記錄更動

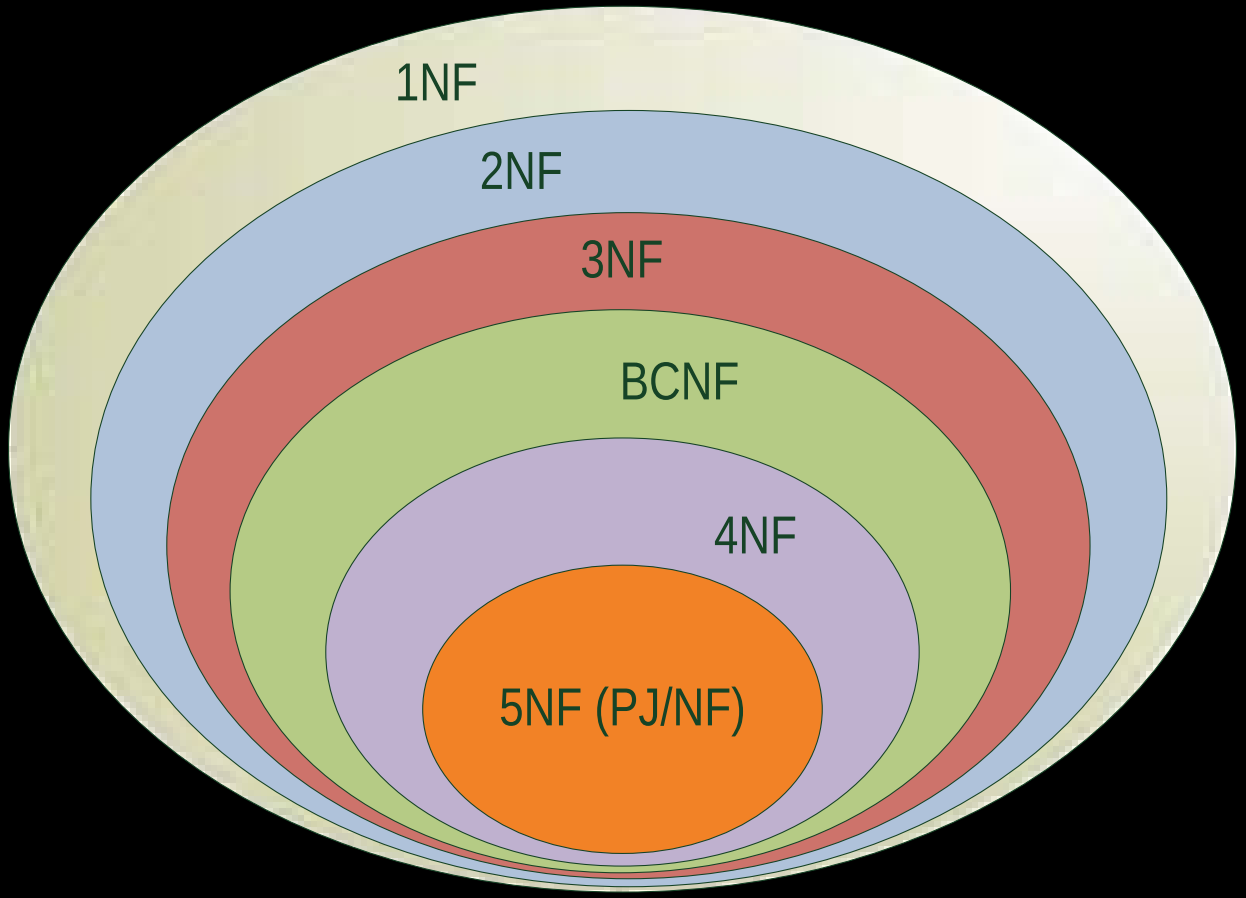
1	巨蟹書局	20	花蓮市	1	30
1	巨蟹書局	20	花蓮市	2	20
1	巨蟹書局	20	花蓮市	3	40
1	巨蟹書局	20	花蓮市	4	20
1	巨蟹書局	20	花蓮市	5	10
1	巨蟹書局	20	花蓮市	6	10

正規化型式的階層架構

- 第一正規化型式 (1NF) by E.F. Codd
- 第二正規化型式 (2NF) by E.F. Codd
- 第三正規化型式 (3NF) by E.F. Codd
- 廣義 3NF—BCNF (Boyce/Codd Normal Form)
- 第四正規化型式 (4NF) by R. Fagin
- 第五正規化型式 (5NF) by R. Fagin
  又稱為 Projection/Join Normal Form (PJ/NF)

正規化型式的階層架構圖

最外層是
完全沒有
正規化的
關聯表



功能相依性 (FD)

- Functional Dependency (FD) 引出：
  - 1NF
  - 2NF
  - 3NF
  - BCNF
- Multi-Valued Dependency (MVD) 引出 4NF
- Join Dependency (JD) 引出 5NF

功能相依性的定義

- [功能相依性] (Functional Dependency)：對於一個關聯表 R 而言，我們說
  - R 中的屬性 B 功能相依於 R 中的屬性 A (B is Functionally Dependent on A)，或稱
  - R 中的 A 在功能上決定 R 中的 B (A Functionally Determines B)，寫成 R.A→R.B，

  若且唯若在任何時候 R 中的 A 屬性值只會對應到一個 R 中的 B 屬性值。A 與 B 都可以是複合屬性 (Composite Attribute)

完全功能相依性

- 如果 R.A→R.B 而且 R.B 不功能相依於 R.A 的部份子集，則稱 R.B 完全功能相依於 R.A
- 前頁的 *rank*, *city*, *name* 並非完全功能相依於 (*no*, *id*) 而是完全功能相依於 *no*
- 通常我們稱「功能相依性」就是指「完全功能相依性」

功能相依性

- R.A → R.B　(R.B 功能相依於 R.A)
  或稱 R.A 在功能上決定 R.B
- 針對 R 中的兩筆記錄 *t* 與 *s* 來說，
  R.A→R.B 表示：　“如果 *t*.A = *s*.A 則 *t*.B = *s*.B”

R

PK	*A*	*B*	*C*	
1	a	b	m	*t*
2	a	b	n	*s*

R

PK	*A*	*B*	*C*	
1	a	b	m	*t*
2	a	c	n	*s*

功能相依性 (續)

- 針對 R 中的兩筆記錄 *t* 與 s 來說，
  R.A→R.B 表示： “如果 *t*.B ≠ s.B 則 *t*.A ≠ s.A”

R

PK	*A*	*B*	*C*	
1	a	b	m	*t*
2	x	c	n	*s*

R

PK	*A*	*B*	*C*	
1	a	b	m	*t*
2	a	c	n	*s*

功能相依性 (續)

- 若 A 是關聯表 R 的候選鍵，則 R.A→R.B
  所以，如果 A 是 R 的主鍵，則所有屬性都會功能相依於 A
- R.A→R.B 表示 A 與 B 之間的關係是 “多對一”
  Son→ Father

SF-Relationship

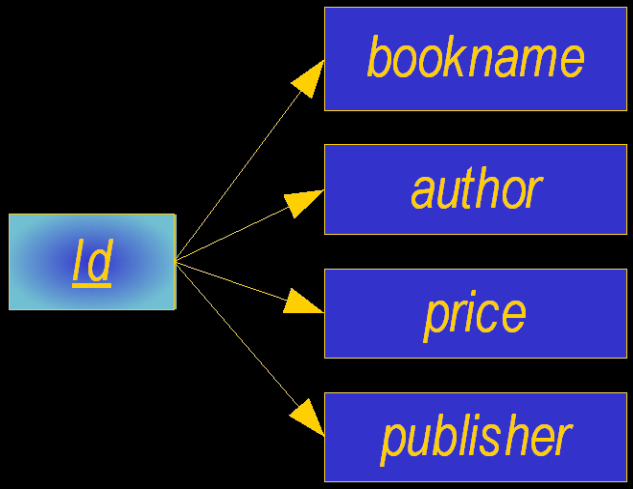
Son	*Father*
Andy	Frank
Mike	Paul

SF-Relationship

Son	*Father*
Andy	Frank
Andy	Paul

功能相依性 (續)

- 「功能相依性」可看成是一種整合限制條件（Integrity Constraint）
- 「功能相依性」只定義於單一的關聯表上
- 多個「功能相依性」的表示法可以合併成一個「功能相依圖」（FD Diagram）



功能相依性 (續)

- 我們無法從表格資料內容看出它所具備的所有「功能相依性」
- 只能靠該關聯表本身所具備的意義才可決定所有的「功能相依性」。
- 不過，我們卻可以從表格的資料中看出它“不”具有那些功能相依性。
- 請見下頁的例子...

如何從表格的資料中看出“不”具有那些功能相依性？

表格範例

A	B	C	D
X	U	X	Y
Y	X	Z	X
Z	Y	Y	Y
Y	Z	W	Z

- 不可能具備 A→B 的功能相依性（圈選處）
- 不可能具備 A→C、A→D 的功能相依性。
- 還可以看出它不具備那些功能相依性嗎？Why?

Armstrong Axioms (阿姆斯狀公理)

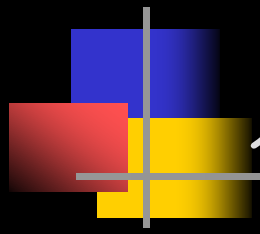
- 由已知的功能相依性推導出隱含的功能相依性
  - [反身性，Reflexivity]: 若 B 屬性是 A 屬性的子集合，則 A→B。
  - [擴充性，Augmentation]: 若 A→B 而且C 屬性是 D 屬性的子集合，則 (A, D)→(B, C)。
  - [遞移性，Transitivity]: 若 A→B 且 B→C 則 A→C。
- 由已知的功能相依性推導出隱含的功能相依性
- 阿姆斯壯公理是「健全而完整的」(Sound and Complete)：它推導出來的功能相依性不會多，也不會少，恰恰好。

1NF, 2NF 與 3NF

- 1NF 定義：所有值域僅含單元值 (Why? 見 3-13 頁)

Books_Not_1NF

id	*bookname*	*author*	*price*	*publisher*
1	{天龍八部，鹿鼎記}	金庸	{250,300}	古文出版社
2	水滸傳	施耐庵	170	中庸出版社
3	紅樓夢	曹雪芹	170	春秋出版社
4	西遊記	吳承恩	140	聊齋出版社
5	水經注	酈道元	120	易經出版社
6	道德經	老子	190	大唐出版社

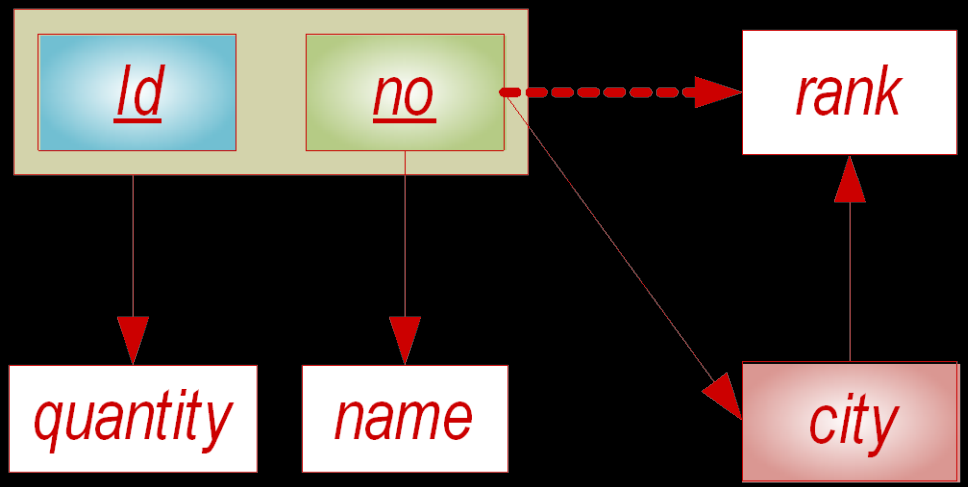


只符合 1NF 所衍生的問題

- 造成關聯表中儲存了許多多餘的資料 (Data Redundancies)，
- 甚至造成了「更新時的異常現象」。
- 上節中所提到的 BO 關聯表，就是這樣的例子，請見下面的分析

BO 的功能相依圖

- 有 Update Anomalies：原因是 BO 中包含了複合主鍵 (*no, id*)，而且其中有一些非候選鍵的屬性 (如： *name*, *rank* 與 *city*) 卻只功能相依於主鍵中的一部份 — *no*

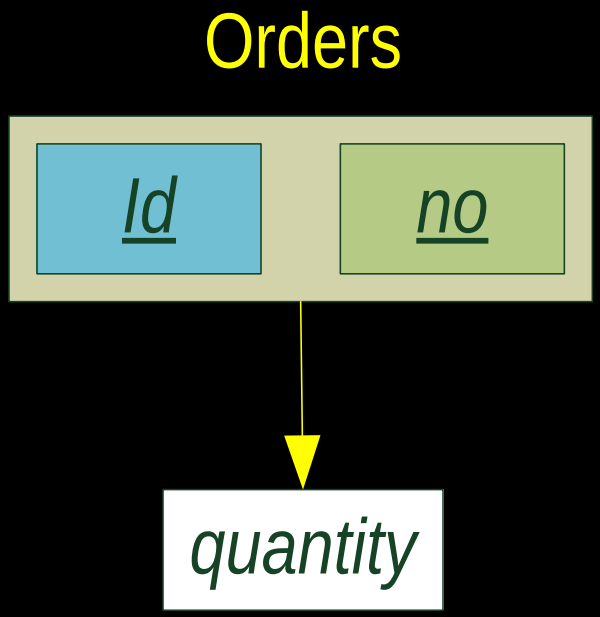


- 虛線指標是使用遞移性所推導出來的功能相依性

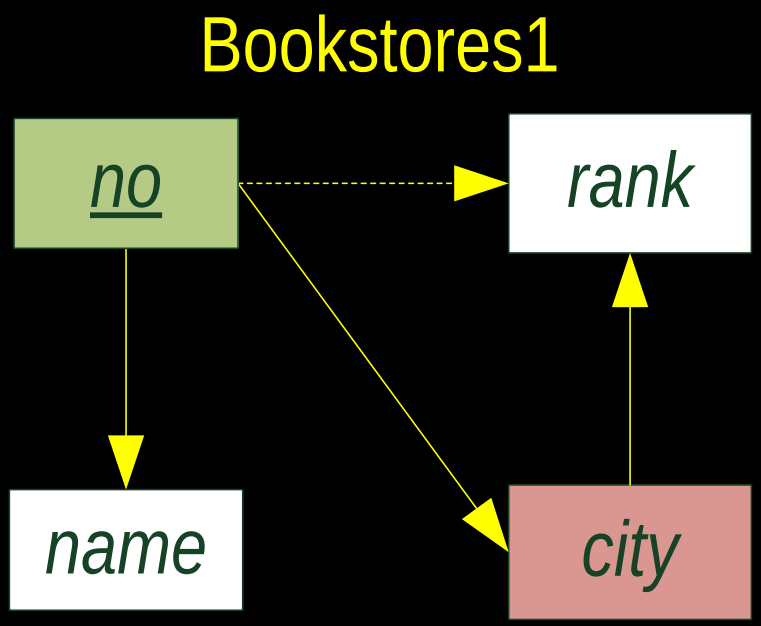
解決 BO 的異常現象

- 原因：BO中有非候選鍵的屬性（如：*name*, *rank* 與*city*）只功能相依於主鍵（*no*, *id*) 的一部份（*no*)。
- 解決方法：讓所有非候選鍵的屬性都完全功能相依於其所屬關聯表的主鍵，而非主鍵的一部份 。
- 作法：分割原關聯表來產生新的關聯表（讓所有屬性都會完全功能相依於其所屬關聯表的主鍵，而非主鍵的一部份）。
- 結果：對關聯表 BO(*no*, *id*, *name*, *rank*, *city*, *quantity*) 透過投影做適當的垂直切割後，得到 Bookstores1(*no*, *name*, *rank*, *city*) 與 Orders(*no*, *id*, *quantity*) 兩關聯表。

將 BO 拆成兩個表格的 FD 圖



Orders 的 FD 圖



Bookstores1 的 FD 圖

不失真的切割原則

- 關聯表切割後必需維持「不失真」(Non-Loss Decomposition) 的原則：原關聯表 R 切割成 R1 與 R2 後，必需要保證將 R1 與 R2 作相等合併運算 (Equijoin) 後會恢復成為原來 R 中所含的所有記錄。
- 所以，上述的 Bookstores1 與 Orders 作相等合併後應該要等於 BO 的內容，才表示不失真。
- 請同學們上臺或自行驗證之

Health 所找出的定理

- I. J. Health (1971) 証明了：“如果一個關聯表 R(A, B, C) 具有 A、B、C 三個屬性，以及 A→B 的功能相依性時，則將 R(A, B, C) 切割成 R1(A, B) 與 R2(A, C) 便一定可以維持「不失真」的原則。”

異常現象消失了！

- 新增：即使某書店尚未訂購書籍，我們仍可以新增一筆書店的資料到 Bookstores1 中。
- 刪除：若想刪除某書店 (e.g. 編號 3) 訂購某書籍 (如：編號 2) 的資料時，則只要刪除 Orders 中 (3, 2, 20) 那一筆記錄即可
- 修改：更改我們的資料庫時，則在 Bookstores1 中所只有一筆記錄需要更改

第二正規化型式

- 第二正規化型式 (2NF)：關聯表 R 屬於第二正規化型式，若且唯若它為第一正規化型式，而且所有不屬於候選鍵的屬性都完全功能相依於該關聯表的主鍵。
- 上述定義並沒有規範：不屬於候選鍵的屬性之間，不可以有任何功能相依性。
- 關聯表 Bookstores1 中所有欄位都功能相依於主鍵 *no* 以外，尚且存在 *city→rank* 的功能相依性，是屬於非候選鍵屬性之間的功能相依性，它同樣也會導致更新的異常現象發生

city→rank 所造成的異常現象

- 新增：雖然*city→rank*，但是我們無法記錄某個城市的 *rank* (例如：永和市的 *rank* 為 35)，除非真有這樣的書局存在，因為這樣會造成 *no* 的屬性值為虛值，而違反了主鍵不能為虛值的限制條件！
- 刪除：若只想刪除某書店 (例如：“獅子書局”)，但想保留它所在城市的 *rank* ，將無法達成。因為刪除 (5, “獅子書局”, 30, “臺南市”) 則 “臺南市” 的 *rank* 為 30 也一併被刪除了。

city→rank 所造成的異常現象

- 更新：如果希望更改某城市 (例如：臺北市) 的 *rank* 由 20 變成了 17，以下表為例，將會面臨有兩筆記錄要同時更改的情況。若不同時更改又會造成不相容的資料。

Bookstores1

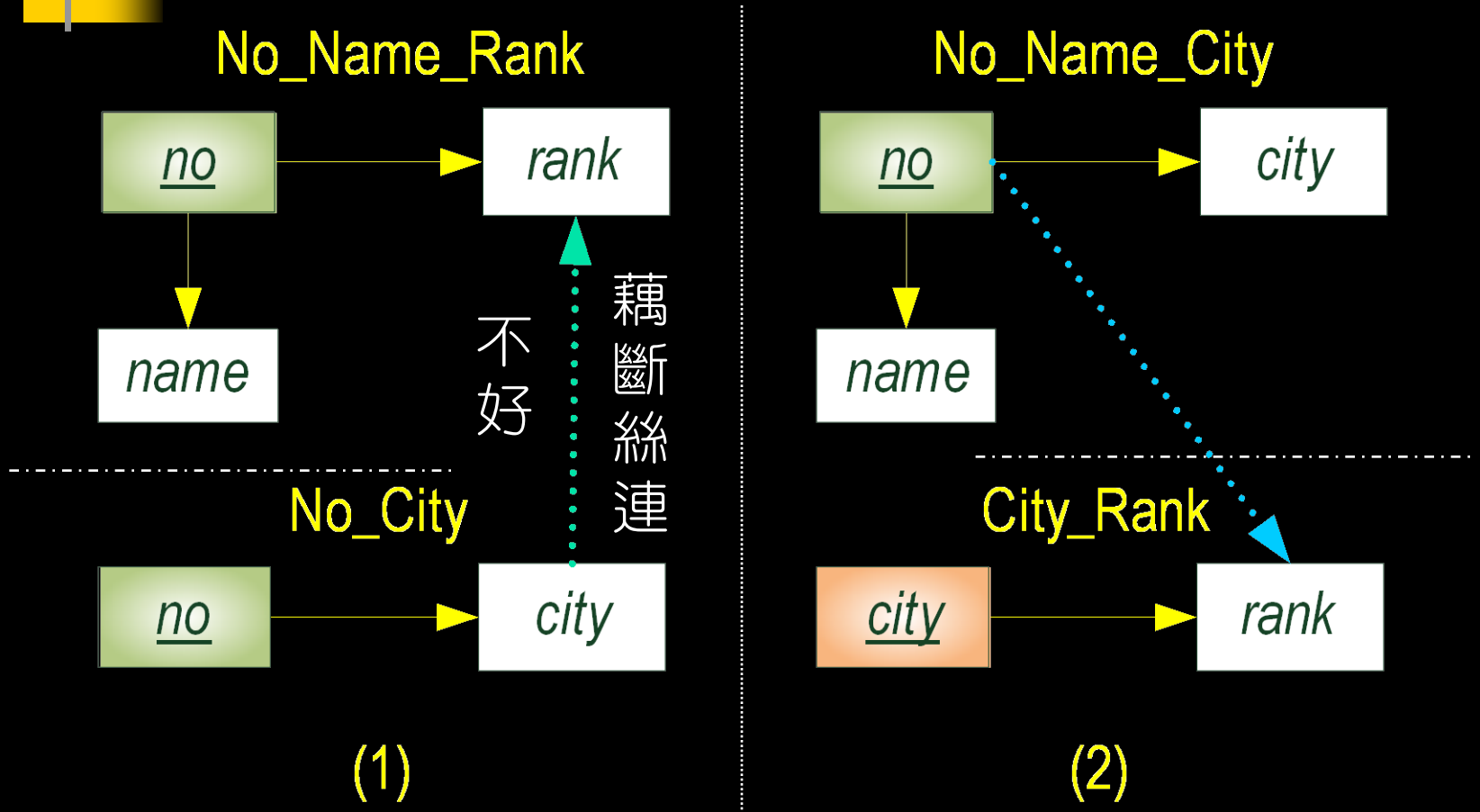
no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺北市
5	獅子書局	30	臺南市

如何解決此異常現象？

- 原因：Bookstores1 中有非候選鍵屬性間的功能相依性
- 解決方法：去除「非候選鍵屬性間的功能相依性」

  city→*rank* 。
- 作法：對此類關聯表做適當的垂直切割。
- 結果：有兩種切割方式：將 Bookstores1
  - 分成 No_Name_Rank (*no*, *name*, *rank*) 與 No_City (*no*, *city*)
  - 分成 No_Name_City (*no*, *name*, *city*) 與 City_Rank (*city*, *rank*)

可能的切割方式：(1) (2)


No_Name_Rank
no
rank
name
不好
藕斷絲連
No_City
no
city
(1)
No_Name_City
no
city
name
City_Rank
city
rank
(2)

為什麼 (2) 的分法較好？

- 因為 (1) 使得 *city→rank* 的關係橫跨兩個關聯表 No_Name_Rank 與 No_City，造成關聯表之間「藕斷絲連」(見上頁的圖左方虛線)，造成了：
  - 新增時的異常現象仍然存在，
  - 因為 *city* 與 *rank* 已經分別放在兩個不同的關聯表中了，所以無法由單一關聯表取得某個城市的 *rank*
  - 每次對於 (1) 中的任何關聯表作更新動作時，都還要確保 *city→rank* 的關係維持一致。

藕斷絲連的苦惱

- 因為 *city* 與 *rank* 已經分別放在兩個不同的關聯表中了，所以無法由單一關聯表取得某個城市的 *rank*

No_Name_Rank

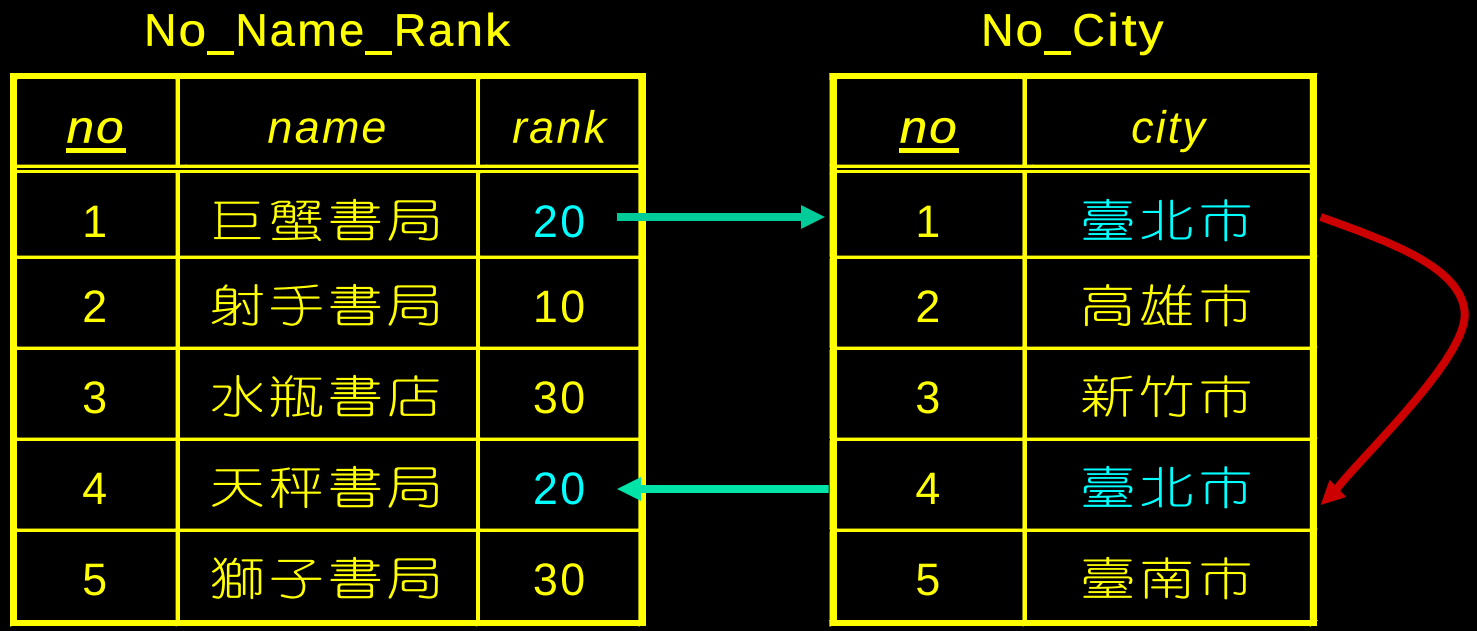
no	*name*	*rank*
1	巨蟹書局	20
2	射手書局	10
3	水瓶書店	30
4	天秤書局	20
5	獅子書局	30

No_City

no	*city*
1	臺北市
2	高雄市
3	新竹市
4	臺北市
5	臺南市

藕斷絲連的苦惱

- 每次對於 (1) 中的任何關聯表作更新動作時，都還要確保 *city→rank* 的關係維持一致。E.g., 將巨蟹書局的 rank 改成 15 那天秤書局的呢?



藕斷絲連的苦惱

- 將 “巨蟹書局” 的 *city* 由 “臺北市” 換成 “臺南市” ，則要將 No_City 中的 (1, “臺北市”) 改成 (1, “臺南市”) 之外，還須更改 No_Name_Rank 中的 (1, “巨蟹書局”, 20) 成為 (1, “巨蟹書局”, 30)

No_Name_Rank

no	*name*	*rank*
1	巨蟹書局	20→30
2	射手書局	10
3	水瓶書店	30
4	天秤書局	20
5	獅子書局	30

No_City

no	*city*
1	臺北市→臺南市
2	高雄市
3	新竹市
4	臺北市
5	臺南市

藕斷絲連的苦惱

- 每次要刪除一家書局時，都要同時對 No_Name_Rank 與 No_City 做刪除的工作，但如此做又可能會讓某個 *city* 與其 *rank* 的關係又有消失的可能了。

No_Name_Rank

no	*name*	*rank*
1	巨蟹書局	20
2	射手書局	10
3	水瓶書店	30
4	天秤書局	20
5	獅子書局	30

No_City

no	*city*
1	臺北市
2	高雄市
3	新竹市
4	臺北市
5	臺南市

No_Name_City 與 City_Rank 的結果

No_Name_City

no	*name*	*city*
1	巨蟹書局	臺北市
2	射手書局	高雄市
3	水瓶書店	新竹市
4	天秤書局	臺北市
5	獅子書局	臺南市

City_Rank

city	*rank*
臺北市	20
高雄市	10
新竹市	30
臺北市	20
臺南市	30

Rissanen 的切割原則

- J. Rissanen (1977)：做正規化時若遇有數種切割方式，則可依下面兩個原則來作：
  - 原關聯表 R 中的每一個功能相依性在切割為 R1 與 R2 後都要能由 R1 與 R2 中推導出。
  - R1 與 R2 中的共同屬性應至少是 R1 或 R2 的候選鍵。
- (1) 的分割方式只合乎第二個原則，但第 (2) 種的分割方式則完全合乎二個原則！Why？
  請同學們驗證之

異常現象又消失了！

- 新增：即使目前沒有訂購書籍的書店位於某城市中，我們仍然可以單獨輸入該城市的 *rank*
- 刪除：如果我們只想刪除某些書店的資料，但是卻仍想保有它所在城市的 *rank* 時，也可順利達成目的
- 修改：如果想要將 “臺北市” 的 *rank* 已經由 20 變成了 17，而希望更改我們的資料庫時，則只須更改一筆 City_Rank 中的記錄即可
- 請同學們參見課本，並驗證之・・・

還有漏網之魚沒有考慮到？

- 還有一種分割方式未提到，那就是將 Bookstores1 (*no*, *name*, *rank*, *city*) 分成 No_Name_Rank(*no*, *name*, *rank*) 與 City_Rank (*city*, *rank*) 。為什麼不討論此種情況呢？
- 因為根本不可行！在下面兩種情況下，會造成「失真」的現象！
  - 當兩個城市的 *rank* 一樣時
  - 當我們不知道某家書店的所在城市為何 的時候？(因為 *city*→*rank* ，所以當然也不知道其 *rank* 為何)

看看下面的例子...

- 假設 “臺南市” 與 “新竹市” 的 *rank* 一樣 (都是 30)，則分割前的 Bookstores1 如下：

Bookstores1

no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺北市
5	獅子書局	30	臺南市

看看下面的例子（續)...

- 分割成 No_Name_Rank 與 City_Rank 後的內容如下：

No_Name_Rank

no	*name*	*rank*
1	巨蟹書局	20
2	射手書局	10
3	水瓶書店	30
4	天秤書局	20
5	獅子書局	30

City_Rank

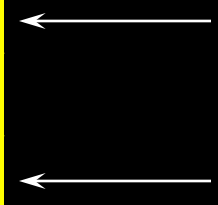
city	*rank*
臺北市	20
高雄市	10
新竹市	30
臺南市	30

看看下面的例子 (續)...

- 將上面兩者做自然合併 (Natural Join) 後發現多出了兩筆原本沒有的資料 (很奇怪)

Bookstores1

no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
3	*水瓶書店*	*30*	*臺南市*
4	天秤書局	20	臺北市
5	*獅子書局*	*30*	*新竹市*
5	獅子書局	30	臺南市



從那裡冒出來的?

再看下面的例子...

- 假設我們不知道 “獅子書局” 所在的城市為何(因為 *city→rank* ，所以當然也不知道其 *rank* 為何？) 也就是說：原關聯表如下

Bookstores1

no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺北市
5	獅子書局	(Null)	(Null)

再看下面的例子（續）...

- 分割成 No_Name_Rank 與 City_Rank 後的內容如下：

No_Name_Rank

no	*name*	*rank*
1	巨蟹書局	20
2	射手書局	10
3	水瓶書店	30
4	天秤書局	20
5	獅子書局	(Null)

City_Rank

city	*rank*
臺北市	20
高雄市	10
新竹市	30

再看下面的例子 (續)...

- 將上面兩者做自然合併 (Natural Join) 後發現“獅子書局” 的資料不見了。(更奇怪)

Bookstores1

no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺北市

原始的第三正規化型式(3NF)

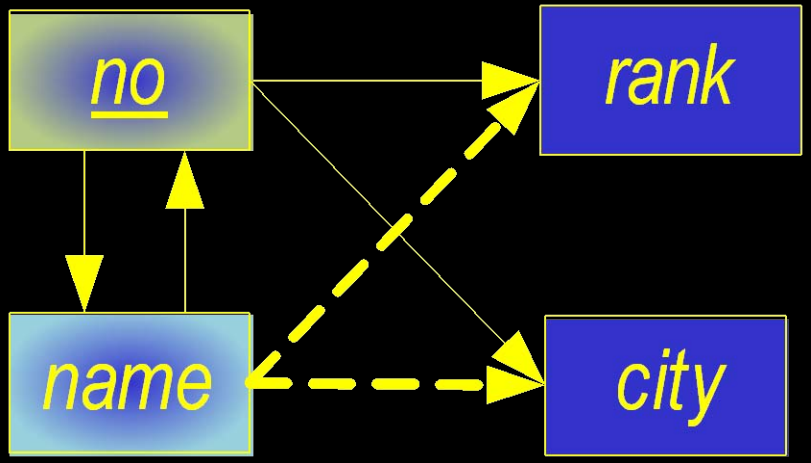
- 總結：只有將 Bookstores1 (*no*, *name*, *rank*, *city*) 切割成 No_Name_City (*no*, *name*, *city*) 與City_Rank (*city*, *rank*)，才能進一步消除異常現象。
- 由於兩個關聯表具有更進一步的特性，所以 Codd 便將類似的關聯表由「第二正規化型式」(2NF) 中劃分出來，稱之為「第三正規化型式」(3NF)
- Codd 原來對第三正規化 (3NF) 的定義：關聯表 R 屬於第三正規化型式，若且唯若它為第二正規化型式，而且所有不屬於候選鍵的屬性都不能彼此功能相依。

第三正規化有缺陷

- 我們可以很容易地驗證上一節所討論的 No_Name_City 與 City_Rank 兩個關聯表都是符合第三正規化型式。
- 不過此定義還是不夠完善，因為：如果一個關聯表有兩個複合屬性的候選鍵，其中一個候選鍵的部分屬性只功能相依於另一個候選鍵的一部分，那麼依然會造成問題。

看一個人造的例子來說明問題

- 若假設前面的 Bookstores1 沒有 *city→rank*，且書局名稱 (*name*) 也是唯一的話，將此關聯表另外稱之為 Bookstores 以示區別，其功能相依圖如



一個人造的例子來說明問題(續)

- 將 Bookstores 與 Orders 合併後將 *no*, *name*, *id*, *quantity* 投影出來後，可得如下的 NNIQ 表

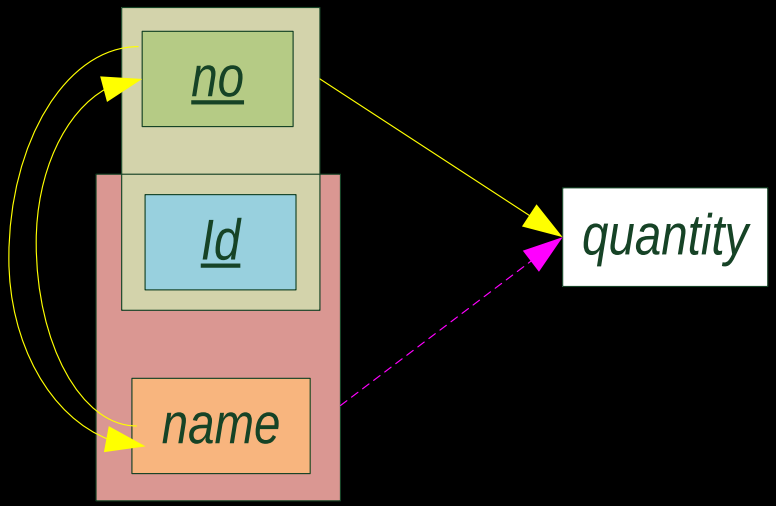
NNIQ

no	*name*	*id*	*quantity*
1	巨蟹書局	1	30
1	巨蟹書局	2	20
1	巨蟹書局	3	40
1	巨蟹書局	4	20
1	巨蟹書局	5	10
1	巨蟹書局	6	10
2	射手書局	1	30
2	射手書局	2	40
3	水瓶書局	2	20
4	天秤書局	4	30
4	天秤書局	5	40

NNIQ 的功能相依圖

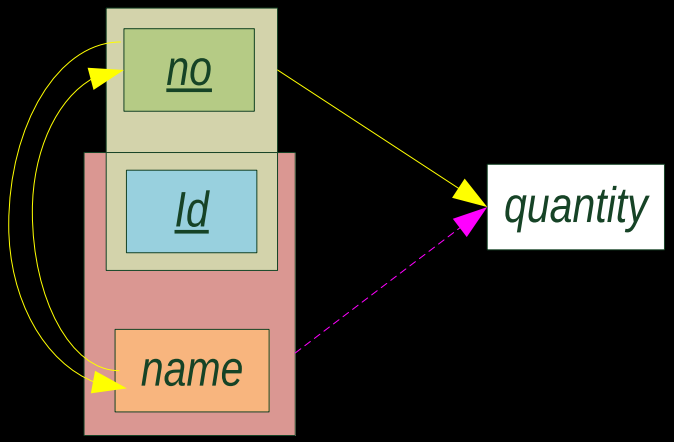
- 虛線部份可以用下列方式推導出來
  - 已知 *name*→*no*，則根據擴充性 (Augmentation) 可推得

    (*name*, *id*)→(*no*, *id*)
  - 又已知 (*no*, *id*)→*quantity*，則根據遞移性 (Transitivity)，將 (*name*, *id*)→(*no*, *id*) 與 (*no*, *id*)→*quantity* 可以推得 (*name*, *id*)→*quantity*。



NNIQ 是 3NF

- NNIQ 有兩個候選鍵 (*no*, *id*) 與 (*name*, *id*)
- 3NF 的定義：所有非候選鍵的屬性（只有quantity）都不能彼此功能相依
- 因此 NNIQ 是 3NF
- 但是仍然會有更新時的異常現象
- 請見下頁的討論...



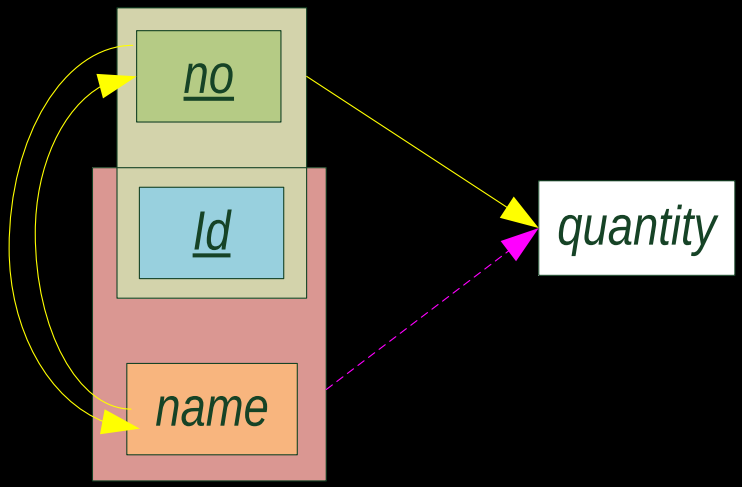
NNIQ 會產生異常現象

- 新增：(*no, id*) 構成了 NNIQ 的主鍵 (或者說 (*name, id*) 構成了 NNIQ 的主鍵)，所以：「除非某一家書店目前有訂購某些書籍，否則我們無法只新增一家書店的名稱 *name*，因為這樣會讓 *id* 的屬性值為虛值，而違反了主鍵或主鍵的一部份不能為虛值的限制條件！」
- 刪除：若想刪除某書店 (例：“水瓶書局”) 訂購某書籍 (如：書籍2) 的資料，而保留書店名稱的話，將無法達成。因為 (3, '水瓶書局', 2, 20)

NNIQ 會產生異常現象(續)

- 更新：若希望更改 “巨蟹書局” 為 “雙魚書局” 時，會面臨有六筆記錄要同時更改的情況。
- 問題癥結：候選鍵的部份屬性 (如：*no*) 只功能相依於另一個候選鍵的一部份—*name*。

- R.F. Boyce 與 E.F. Codd在發現了此一現象之後，便重新修正了第三正規化型式的定義，將其稱之為BCNF (Boyce/Codd Normal Form)



如何解決此異常現象？

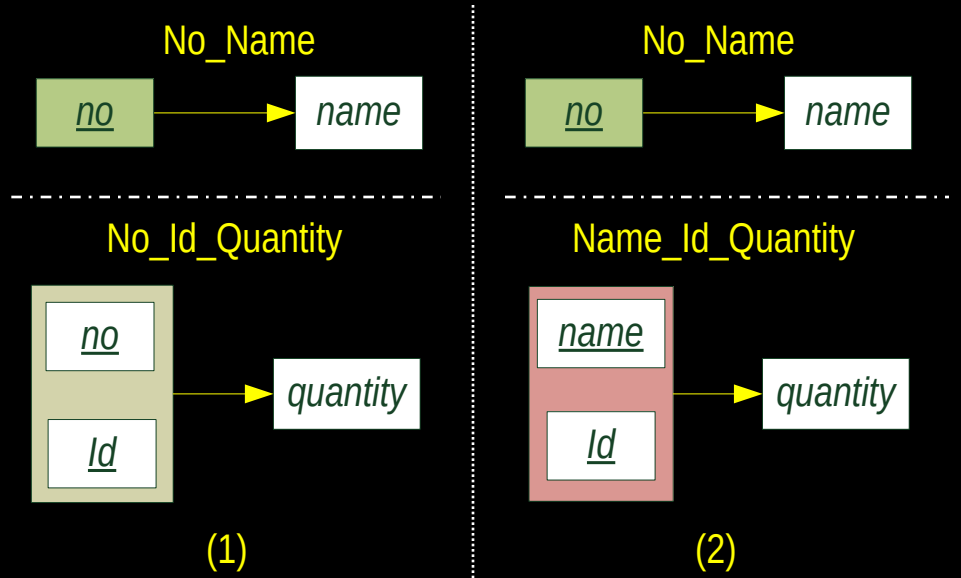
- 解決方法：將 NNIQ 分成兩個關聯表
- 作法：去除關聯表內「非完整候選鍵」的功能相依性 。分割為

1. No_Id_Quantity(*no, id, quantity*) 與 No_Name(*no, name*)

或分割為

2. No_Name(*no, name*) 與 Name_Id_Quantity(*name, id, quantity*)

- 結果：



(1) 切割成 No_Name (*no*, *name*) 與 No_Id_Quantity (*no*, *id*, *quantity*)

No_Name

no	*name*
1	巨蟹書局
2	射手書局
3	水瓶書店
4	天秤書局

No_Id_Quantity

no	*id*	*quantity*
1	1	30
1	2	20
1	3	40
1	4	20
1	5	10
1	6	10
2	1	30
2	2	40
3	2	20
4	4	30
4	5	40

(2) 切割成 No_Name (*no*, *name*) 與 Name_Id_Quantity (*name*, *id*, *quantity*)

No_Name

no	*name*
1	巨蟹書局
2	射手書局
3	水瓶書店
4	天秤書局

Name_Id_Quantity

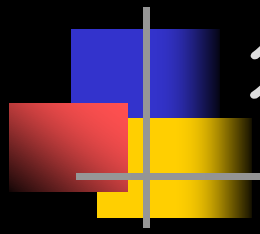
name	*id*	*quantity*
巨蟹書局	1	30
巨蟹書局	2	20
巨蟹書局	3	40
巨蟹書局	4	20
巨蟹書局	5	10
巨蟹書局	6	10
射手書局	1	30
射手書局	2	40
水瓶書局	2	20
天秤書局	4	30
天秤書局	5	40

異常現象又消失了！

- 新增：不論採用 (1) 或 (2) 的切割方式，即使某家書店目前沒有訂購書籍，仍然可以新增該書店的編號與名稱到 No_name 關聯表中，不會有*id* 為虛值的情況發生
- 刪除：如果只想刪除某書店 (例如：水瓶書局) 訂購某書籍 (如：編號 2 的書籍) 的資料時，在第 (1) 種作法下只要刪除 No_Id_Quantity 中 (3, 2, 20) 那一筆記錄即可，第 (2) 種作法也只要要刪除Name_Id_Quantity 中 ('水瓶書局', 2, 20) 那一筆。我們仍可以在 No_Name 關聯表中保有該書店本身的所有資料。

異常現象又消失了！

- 更新：假設 “巨蟹書局” 要改名為 “雙魚書局”，不論是 (1) 或 (2) 的作法，由於 No_Name 只會有一筆 “巨蟹書局” 的資料，所以當我們要更正資料時，只需更正 No_Name 的這一筆資料即可，不用擔心會造成不一致的資料。
- 「在第 (2) 種作法下，Name_Id_Quantity 不是也有 “巨蟹書局” 的資料嗎？為什麼不處理呢？」
  - 因為 “巨蟹書局” 的屬性是 *name*，而 *name* 是 Name_Id_Quantity 的外來鍵，所以關於Name_Id_Quantity 中 “巨蟹書局” 資料的更新，透過外來鍵連鎖更新的設定，系統自然會處理

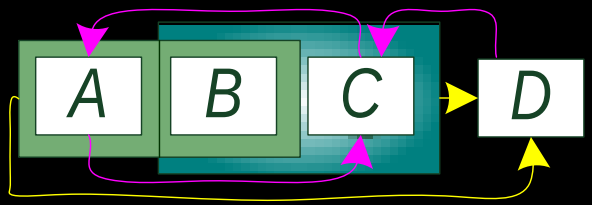


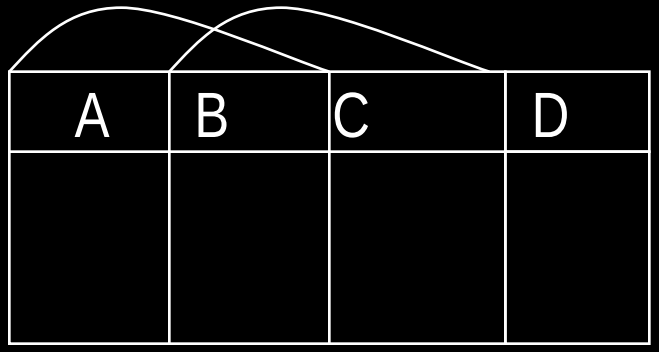
從 3NF 到 BCNF

- (1) 與 (2) 兩種切割方式都可以解決異常現象，而所產生的關聯表即是所謂的 BCNF (Boyce/Codd Normal Form)

從 3NF 到 BCNF

- 有下述幾種情況時，就必須要採行 BCNF 來取代第三正規化型式：
  - 該關聯表的候選鍵都是由數個屬性所構成的 (複合的)，
  - 這些候選鍵所含屬性的交集不是空集合，也就是說候選鍵之間有共用某些屬性
  - 候選鍵中的屬性子集功能相依於某個非候選鍵的屬性，或另一組候選鍵的屬性子集（如下圖紅色的箭頭所示）





廣義的第三正規化型式(BCNF)

- 先定義何謂「決定性屬性」(Determinant)：一個關聯表 R 中的屬性 A 為「決定性屬性」(Determinant)，此屬性可能是複合屬性，若且唯若有某些 R 中的屬性功能相依於 A。
- 從功能相依圖來看的話，那麼所有箭頭的起始屬性都是決定性屬性
- BCNF 定義: 一個關聯表 R 屬於 BCNF，若且唯若所有的決定性屬性都是該關聯表的候選鍵。

有關 BCNF 的一些說明

- 若前兩頁的三種情況不出現的話，那麼 BCNF 便是原本的 3NF
- 圖 11.3 中所提到的 BO 中的決定性屬性為 *no*、*city*、與 (*no*, *id*)，但只有 (*no*, *id*) 才是候選鍵，所以 BO 不是 BCNF
- No_Name_City 與 City_Rank 則都是 BCNF

一個不適合正規化成 BCNF 的例子

- 是不是為了消除「更新時的異常現象」，似乎只要往下正規化就對了?! 其實並不盡然，請見下例
- 對於每一個科目 (Subject) 而言, 該科目的每一位學生 (Student) 最多只會佔用一間教室 (Classroom) 考試. 也就是說我們有一個功能相依性 (*student*, *subject*) → *classroom*
- 每一間教室只能用來考一門科目，也就是說 *classroom* → *subject*，
- 每一科目可能會使用數間教室當考場，也就是說：沒有 *subject* → *classroom* 這個功能相依性。

一個是 3NF 但非 BCNF 的例子

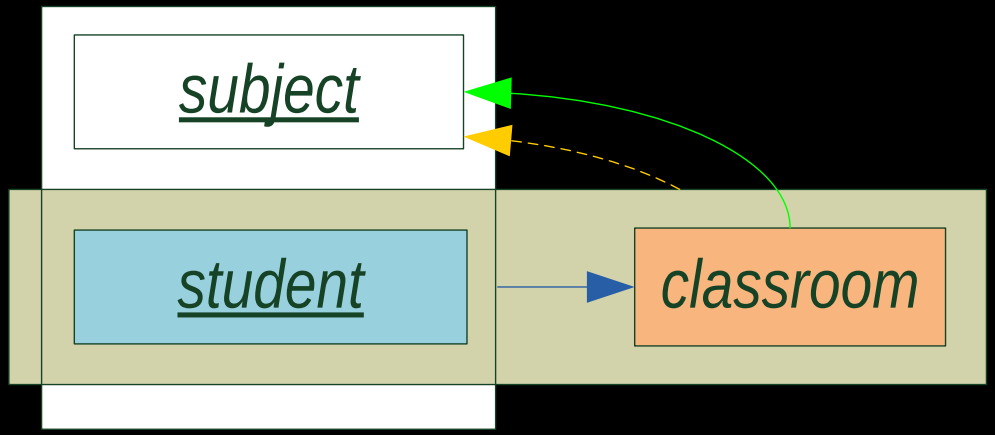
- (*student*, *subject*) → *classroom*
- *classroom* → *subject*
- 沒有 *subject* → *classroom*

SJC

student	*subject*	*classroom*
Tom	Calculus	C1301
Tom	MIS	C1307
Andy	Calculus	C1301
Andy	MIS	C1303

有關 SJC 的說明

- *classroom* → *subject*，所以使用擴充性可以導出下面的虛線部份 (*student*, *classroom*)→*subject*
- SJC 有兩個候選鍵 (*student*, *classroom*) 與 (*student*, *subject*)，
- 其功能相依圖



SJC 是3NF 但非 BCNF！

- SJC 是 3NF！因為 SJC 中沒有非候選鍵的屬性（所以他當然也是 2NF）
- 但是 SJC 卻不是 BCNF, 因為 classroom 是一個決定性屬性，但卻不是一個候選鍵。所以，SJC 不是 BCNF。當然也會發生更新時的異常現象:
  - 刪除 Andy 應考 MIS 這門科目會連 C1303 教室是用來做為 MIS 考場的事實也一併刪除

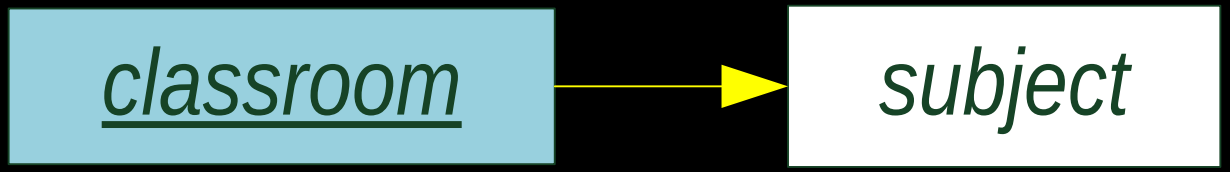
解決之道

SC

student	*classroom*
Tom	C1301
Tom	C1307
Andy	C1301
Andy	C1303

CJ

classroom	*subject*
C1301	Calculus
C1307	MIS
C1303	MIS



R(a, b) 一定是 BCNF！

- 只含兩屬性的關聯表 R(a, b) 一定是 BCNF
- 證明：
  - 如果 (a, b) 是候選鍵的話，則 R(a, b) 是 BCNF。
  - 如果 a 是候選鍵但 b 不是的話，則 a 為主鍵，所以 $a \rightarrow b$，因此 R(a, b) 是 BCNF。同理可證，如果 b 是候選鍵但 a 不是的情況。
  - 如果 a 是候選鍵，但 b 也是的話，則 $a \rightarrow b$，而且 $b \rightarrow a$，因此 R(a, b) 也是 BCNF。

仍有無法解決的遺憾！

- 將 SJC 進一步正規化成 SC (*student*, *classroom*) 與 CJ (*classroom*, *subject*) 仍有些遺憾存在！
- 由 J. Rissanen 定理我們發現：(*student*, *subject*)→ *classroom* 並沒有辦法由 SC 與 CJ 中推導出
- 造成兩個關聯表並非各自獨立，仍然 “藕斷絲連”
- 有時要將關聯表分割成兩個 BCNF 的關聯表，而又要保持互相獨立常常是不能兩全其美的

SC 與 CJ 並非各自獨立的說明

- 如果將原本在 C1307 考試的 MIS 換到 C1305 考的話，則我們必需要同時更改 SC 與 CJ 兩個關聯表

SC

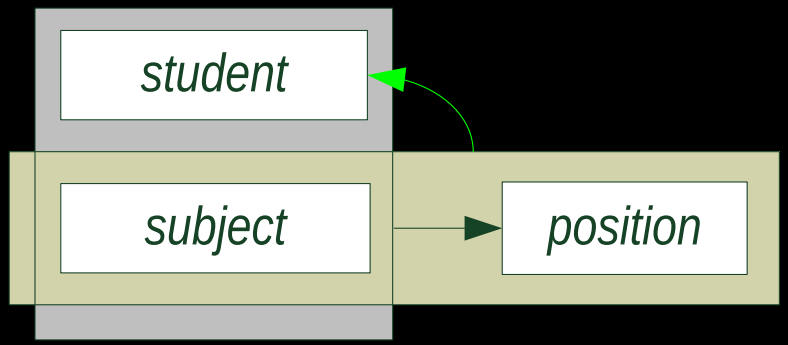
student	*classroom*
Tom	C1301
Tom	C1305
Andy	C1301
Andy	C1303

CJ

classroom	*subject*
C1301	Calculus
C1305	MIS
C1303	MIS

含有兩候選鍵的關聯表但也是 BCNF 的例子

- 下述表格含有兩個功能相依性: (*student*, *subject*) → *position* 與 (*subject*, *position*) → *student*
- EXAM 不僅是 2NF、3NF，同時也是 BCNF，因為除了沒有非候選鍵的屬性之外，所有決定性屬性都是該關聯表的候選鍵



EXAM

student	*subject*	*position*
Tom	Calculus	110101
Andy	MIS	110201
Peter	Calculus	110102
Mike	MIS	110202

多重值相依性 (MVD)

- 探討進一步正規化之前，要先將功能相依性推廣成「多重值相依性」(MVD, Multi-Valued Dependency) [R. Fagin (1977)]
- 以下是一個 BCNF 關聯表 (所有欄位構成主鍵)

University

department	*student*	*course*
MIS	Frank	Intro. to Data Structure
MIS	Frank	Intro. to Database Systems
MIS	Peter	Intro. to Data Structure
MIS	Peter	Intro. to Database Systems
EE	John	Linear Algebra
EE	John	Switching Circuit

University 上的異常現象

- Frank 與 Peter 都是 MIS 的學生，所以，如果 Frank 的必修課有 “Intro. to Data Stru.”，那麼 Peter 的必修課也要有 “Intro. to Data Stru.” (因為兩人同系)。
- 如果我們要加入一位新的學生 Mike 到電機系 (EE) 的話，那麼勢必要一次加入兩筆 (Why?)

EE	Mike	Linear Algebra
EE	Mike	Switching Circuit

多重值相依性的定義

- 多重值相依性 (MVD)： 對於關聯表 R(A, B, C) 而言，我們說 R 中的屬性 B 多重值相依於 R 中的屬性 A（或 R 中的 A 多重值決定了 B），寫成 R.A→→ R.B，若且唯若 B 屬性中的值所構成的集合在 R 中與 (A, C) 屬性中的值所構成的值組只與 A 的值有關，而與 C 的值無關。A、B、C 都可能是複合屬性。
- 見前面的 University 範例較容易了解。

關於多重值相依性的說明

- 一定是定義在三個屬性上。
- 不過嚴格上來說，如果僅含有兩個屬性的關聯表 R(X, Y) 時，如果 X 和 Y 完全沒有功能相依性的話，那麼其實也可以看成是一些「退化的多重值相依性」(Degenerate MVD)：$\Phi \rightarrow\!\!\!\rightarrow X$ 而且 $\Phi \rightarrow\!\!\!\rightarrow Y$。
- 在關聯表 R(A, B, C) 中，若 $A \rightarrow\!\!\!\rightarrow B$，則可推出 $A \rightarrow\!\!\!\rightarrow C$，所以也可以寫成 $A \rightarrow\!\!\!\rightarrow B \mid C$

關於多重值相依性的說明

- 在關聯表 R(A, B, C) 中，若 $A \rightarrow\!\!\!\rightarrow B \mid C$ 則：如果有兩個值組： (a, b1, c1) 與 (a, b2, c2) 同時出現在 R 中的話，則必然會有另外兩個值組： (a, b1, c2) 與 (a, b2, c1) 也會同時出現在 R 中。

EE	Mike	Linear Algebra
EE	John	Switching Circuit

EE	Mike	Switching Circuit
EE	John	Linear Algebra

阿姆斯壯公理在 MVD 下的推廣

- 由 C. Beeri, R. Fagin, 與 J.H. Howard 將阿姆斯壯公理推廣，並於 1977 年所提出 [C. Beeri, R. Fagin, and J.H. Howard (1977)]
- 相當複雜，所以不必強記
- 可以撰寫 Prolog 程式來推導之
- 請參考課本 9-34, 9-35 頁

第四正規化型式 (4NF)

- 第四正規化型式 (4NF)：關聯表 R 屬於第四正規化型式 (4NF)，若且唯若它所有的多重值相依性都是功能相依性。
- 將 University (*department*, *student*, *course*) 分成 DS (*department*, *student*) 與 DC (*department*, *course*) 兩個第四正規化型式可解決其異常現象
- 切記：在第四正規化的關聯表中找不到「多重值相依性」

分割為 DS 與 DC 的依據

- 分割為 DS 與 DC 的依據如下：
- 若關聯表 R(A, B, C) 具有 R.A →→R.B | R.C 若且唯若 R(A, B, C) 可以不失真地分解成 R1(A, B) 與 R2(A, C)。
- DS(*department, student*) 的所有欄位都是主鍵 ，DC(*department, course*) 亦同
- 請同學們思考一下為什麼將 University 分成 DS 與 DC 可以解決前述的異常現象？

第五正規化型式 (5NF)

- 「是不是所有的關聯表都可以不失真地切割成兩個關聯表？」答案是否定的！
- 有些關聯表必須要切割成三個以上的關聯表，才能於合併後 (將切割後所得的所有關聯表通通自然合併起來) 恢復原狀。
- 此一特性乃 A.V. Aho, C. Beeri, 與 J.D. Ullman 三人於 1977 年所提出

範例

- 將下述關聯表切成兩個關聯表都無法「不失真」地合併回來！但是切成三個則可以！

NIS

no	*id*	*sid*
1	1	2
1	2	1
2	1	1
1	1	1

將 NIS 切成三個可以不失真

- 假設切成下面三個，讓我們來驗證一下：

NI

no	*id*
1	1
1	2
2	1

IS

id	*sid*
1	2
2	1
1	1

SN

sid	*no*
2	1
1	1
1	2

驗證過程(一)

- 先將 NI 與 IS 做自然合併，合併欄位是 *id*

no	*id*
1	1
1	2
2	1

⋈id

id	*sid*
1	2
2	1
1	1

產生

NIIS

no	*id*	*sid*
1	1	2
1	1	1
1	2	1
2	1	2
2	1	1

多出了一筆待會兒會被消去

驗證過程(二)

- 再將上述結果與 SN 做自然合併，合併欄位是 (*sid*, *no*)。則可以得到原關聯表
- 雖然不同的合併順序會得到不同的中間結果，但是最後的結果卻都是一樣的。請同學驗證之

NIS

no	*id*	*sid*
1	1	2
1	2	1
2	1	1
1	1	1

造成這種現象的原因

- 上例中 “NIS 等於 NI、IS 與 SN 的合併結果” 乃是下面的限制條件所造成的：如果 (1, 1) 出現在 NI 中，而且 (1, 1) 出現在 IS 中，而且 (1, 1) 出現在 SN 中的話，則 (1, 1, 1) 便會出現在 NIS 中。
- 做進一步推導之後，我們可得：如果 (1, 1, 2) 、(2, 1, 1)、 (1, 2, 1) 都出現在 NIS 中的話，則 (1, 1, 1) 也會出現在 NIS 中。

更白話的說法

- 我們的限制條件便是：「如果 Smith 供應鋼筋，十八標工程也用了鋼筋，而且 Smith 也供應了十八標工程之原料的話，那麼 Smith 便一定提供了鋼筋給十八標工程。」
- 此種限制條件形成一個**循環現象**：Smith 供應鋼筋、鋼筋也被十八標工程所採用，而且 Smith 也供應了十八標工程之原料，則形成一個循環結果讓 Smith、鋼筋與十八標工程三者同時出現構成關聯表中的一個值組。

合併相依性

- 合併相依性(Join Dependency, 簡稱 JD)：假定 X, Y,..., Z 都是關聯表 R 的投影結果，則 R 滿足了「合併相依性」－＊(X, Y, ..., Z) ，若且唯若 R 等於 X, Y, ..., Z 的合併結果。
- 由上例可知：NIS 滿足了 ＊(NI, IS, SN) 這個合併相依性。此種具有合併相依性的關聯表會導致「更新時的異常現象」發生

NIS 的異常現象 (一)

NIS

no	*id*	*sid*
1	1	2
1	2	1

- 新增：如果原來關聯表只含兩個值組的話 (如上所示) 那麼加入 (2, 1, 1) 時，也要跟著加入 (1, 1, 1)；但是加入 (1, 1, 1) 時卻不用跟著加入 (2, 1, 1)，真是奇怪！？

NIS 的異常現象 (二)

- 如果要對原來的關聯表做刪除的話，那麼單獨刪除 (2, 1, 1) 時不會有任何副作用；但是刪除 (1, 1, 1) 時，卻要跟著刪除 (1, 1, 2)、(1, 2, 1) 或 (2, 1, 1) 其中的任何一個才行，但是要刪除那一個呢？(不知道！)

NIS

no	*id*	*sid*
1	1	2
1	2	1
~~2~~	~~1~~	~~1~~
~~1~~	~~1~~	~~1~~

可直接刪

不可直接刪

FD, MVD 與 JD 的關係

- JD 乃是我們前面所提到之 MVD 的推廣定義
- 關聯表 R(A, B, C) 中，若 A、B 與 C 具有多重值相依性 R.A →→R.B | R.C，若且唯若 R(A, B, C) 可以不失真地分解成 R1(A, B) 與 R2(A, C)。
- 套 JD 的解釋:關聯表 R(A, B, C) 中，若 R(A, B, C) 滿足合併相依性 *(AB, AC)，若且唯若 R(A, B, C) 可不失真地分解成 R1(A, B),與 R2(A, C)。
- 結論：FD 可推廣到 MVD, MVD 可推廣到 JD

解決此異常現象的作法

- 第五正規化型式 (5NF)：關聯表 R 屬於第五正規化型式 (5NF)，若且唯若 R 中所有的合併相依性 *(X, Y) 中的 X 與 Y 兩關聯表所含有的共同屬性 (也就是用來作自然合併的屬性) 是 R 的候選鍵。
- 也就是說：5NF 中是不可以存在合併相依性的
- NIS 並不是 5NF，它必須再分割成 NI(*no, id*) 、IS(*id, sid*) 與 SN(*sid, no*)，才可避免異常現象

正規化的步驟（複雜版）

- 先要求所有屬性只放 Atomic Value，產生 1NF。
- 再消除所有的「非完整候選鍵的功能相依性」產生 2NF 的關聯表。
- 再消除所有的「非候選鍵屬性之間的功能相依性」產生 3NF 關聯表。
- 再消除所有決定性屬性不為候選鍵的功能相依性 (FD) 產生 BCNF 關聯表。
- 再消除 MVD 產生 4NF
- 最後消除 JD 產生 5NF

正規化的步驟（精簡版）

- 先求所有屬性只放 Atomic Value，產生 1NF。
- 再消除所有決定性屬性不為候選鍵的功能相依性，直接產生 BCNF
- 再消除 MVD 產生 4NF
- 最後消除 JD 產生 5NF
- 關聯表切得越小，查詢效率越差，所以通常做到 BCNF 就可以了！為了效率有時甚至還要做「反正規化」(De-Normalization) 的動作

Top-Down *vs.* Bottom-Up 設計

- 開發資訊系統之前，需要對使用者做需求訪談。
- 訪談的過程中，有關資料庫設計，很重要的一點就是要找出使用者所需要用到的資料欄位、相關資料型態等，
- 而常常被忽略的一點往往就是如何找出這些欄位之間的「功能相依性」、「多重值相依性」等，
- 它們關係著要不要正規化？要如何正規化？等議題，
- 找出資料庫中所需欄位彼此之間的相依性，然後將關聯表予以正規化到某個型式其實是「由下而上」(Bottom-Up) 的作法。

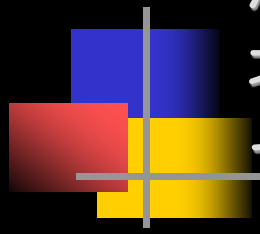
Top-Down *vs.* Bottom-Up 設計

- 在下一節裡，我們所要談的是：如何找出資料庫中的「個體」(Entities) 與彼此間的「關係」(Relationships)，並藉此設計出資料庫的結構與所需的欄位，而此種作法其實是「由上而下」(Top-Down) 的作法。兩者是可以相輔相成的

Top-Down 設計
E/R Modeling

Bottom-Up 設計
正規化過程

用「個體-關係模式」來做邏輯資料庫設計



- 利用Entity-Relationship Diagram (ERD) 做資料庫設計可以確保設計出來的是 3NF。
- 個體-關係模式 (E-R Model) 中的概念
  - 個體：一般個體 (Regular Entity) 與弱勢個體 (Weak Entity)
  - 個體類型 (Entity Type)：由同一類個體所組成
  - 特性 (Properties)：單一特性或複合特性
  - 關係：一般關係 (Regular Relationship) 與弱勢關係 (Weak Relationship) 可以有 1:1, 1:N, N:M 的關係

MVD 與 E/R 中的多對多關係

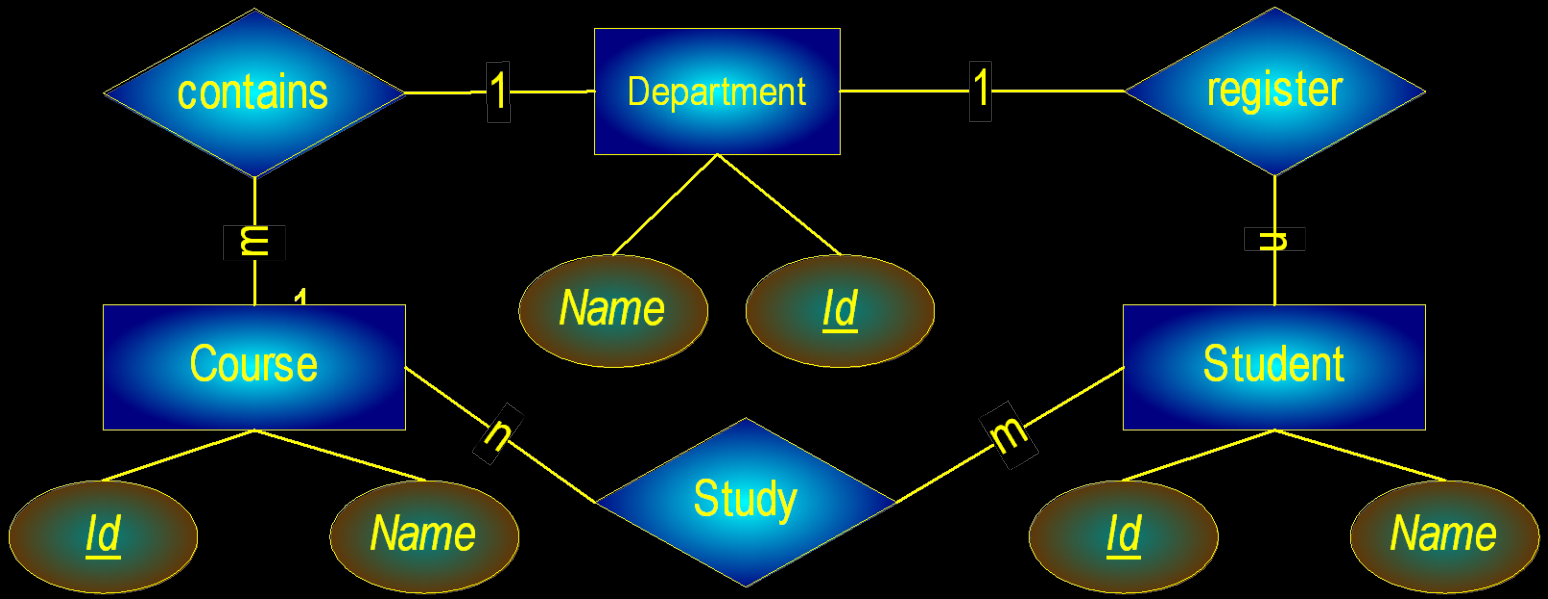
- 如果一個關聯表中包含了 MVD (Multi-Valued Dependency) 如:

University

department	*student*	*course*
MIS	Frank	Intro. to Data Structure
MIS	Frank	Intro. to Database Systems
MIS	Peter	Intro. to Data Structure
MIS	Peter	Intro. to Database Systems
EE	John	Linear Algebra
EE	John	Switching Circuit

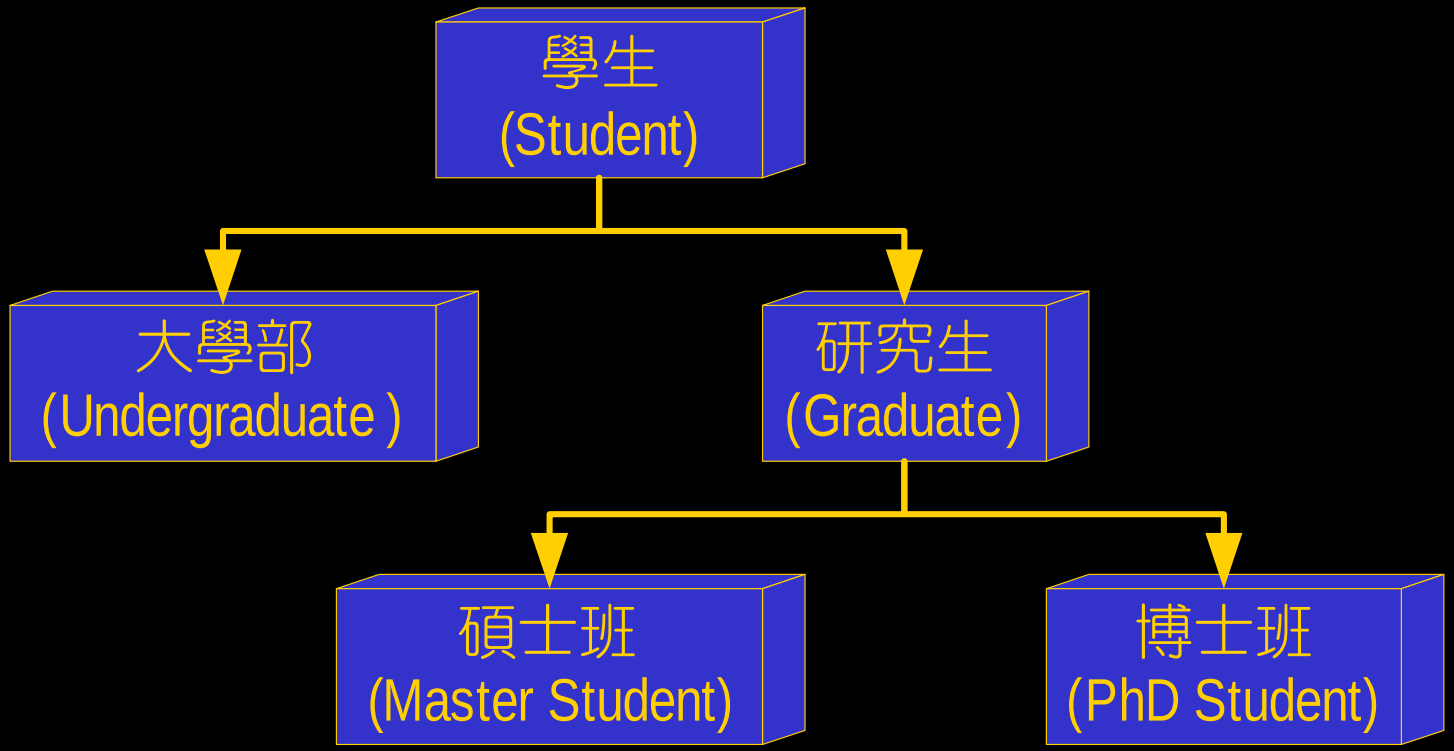
MVD 與 E/R 中的多對多關係

- 如果希望將上述欄位擴增成為關聯表的話, 那麼就會形成 Entity-Relationship Diagram 中的 “多對多” 關係 或是兩個 “1 對多” 關係

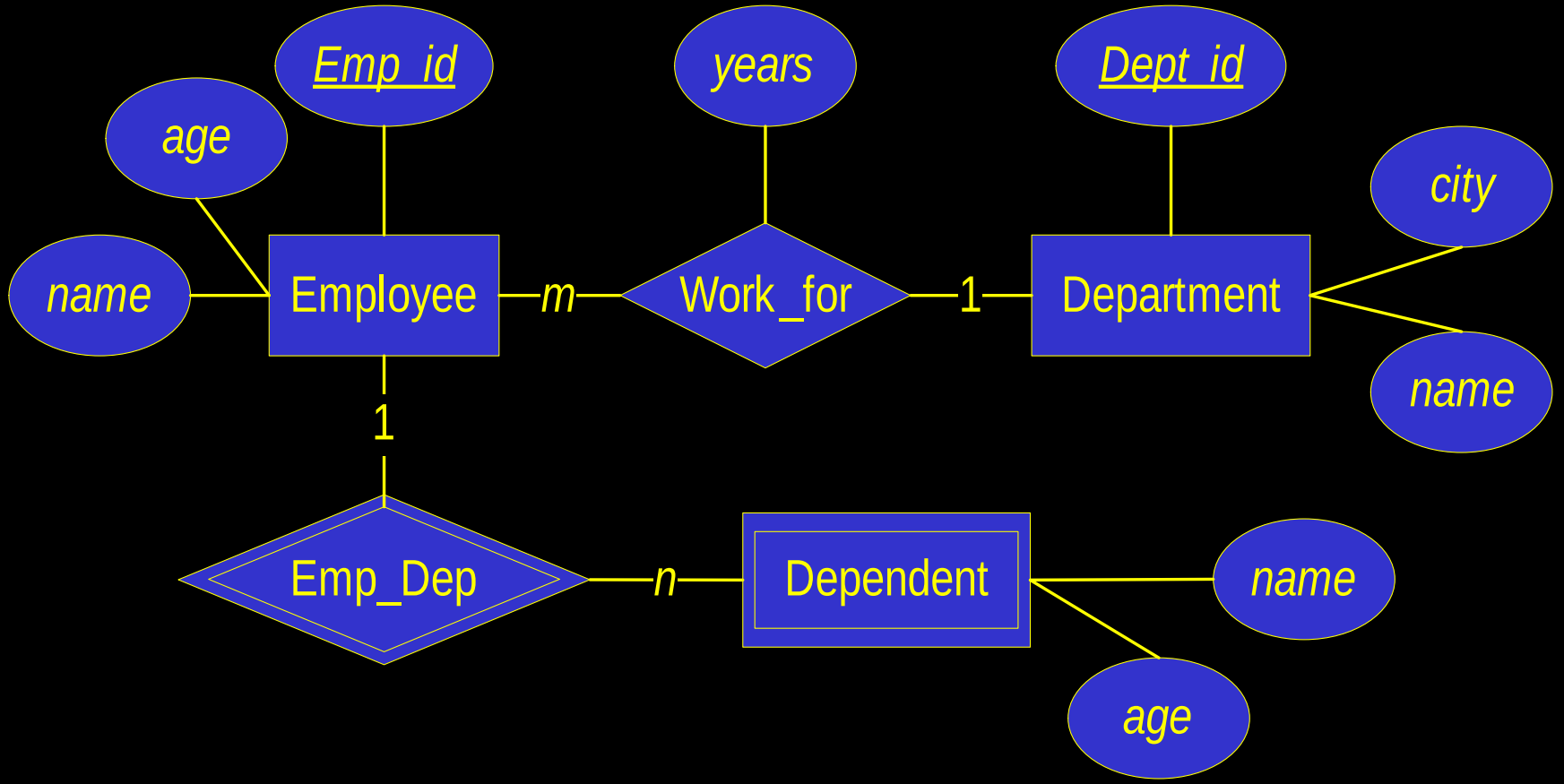


新增的個體-關係模式概念

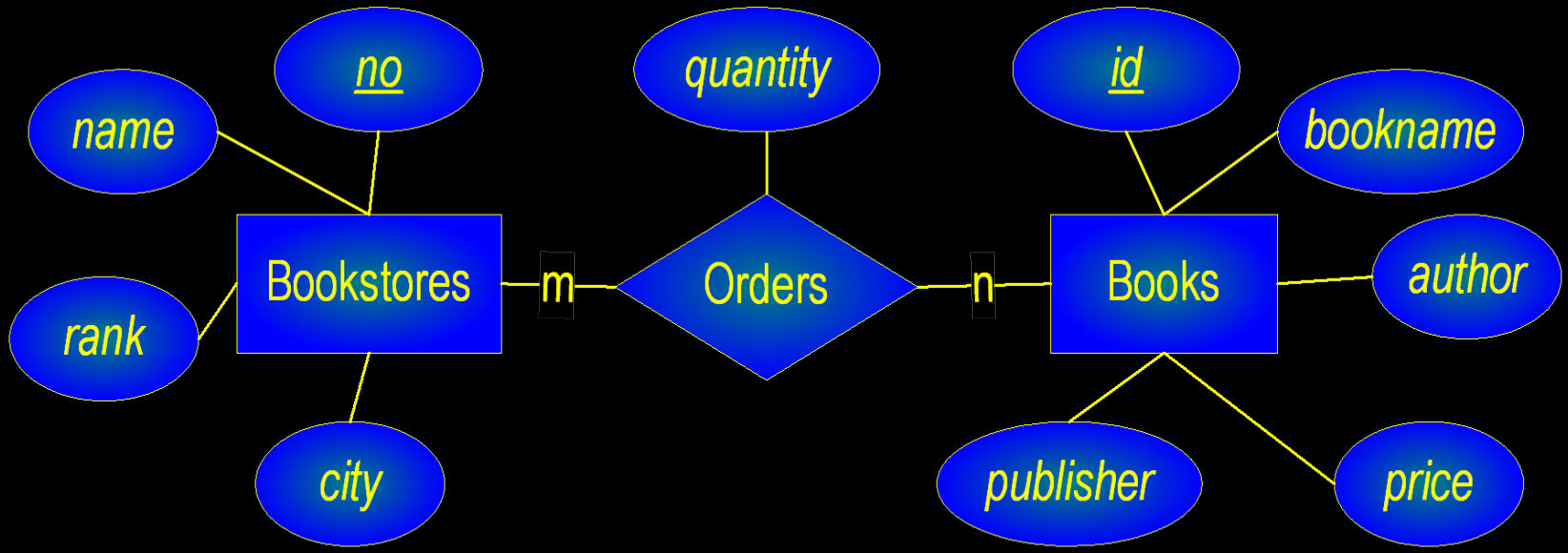
- 子類型 (Subtypes)：演變成「物件導向資料模式」(Object-Oriented Data Model) 的基礎



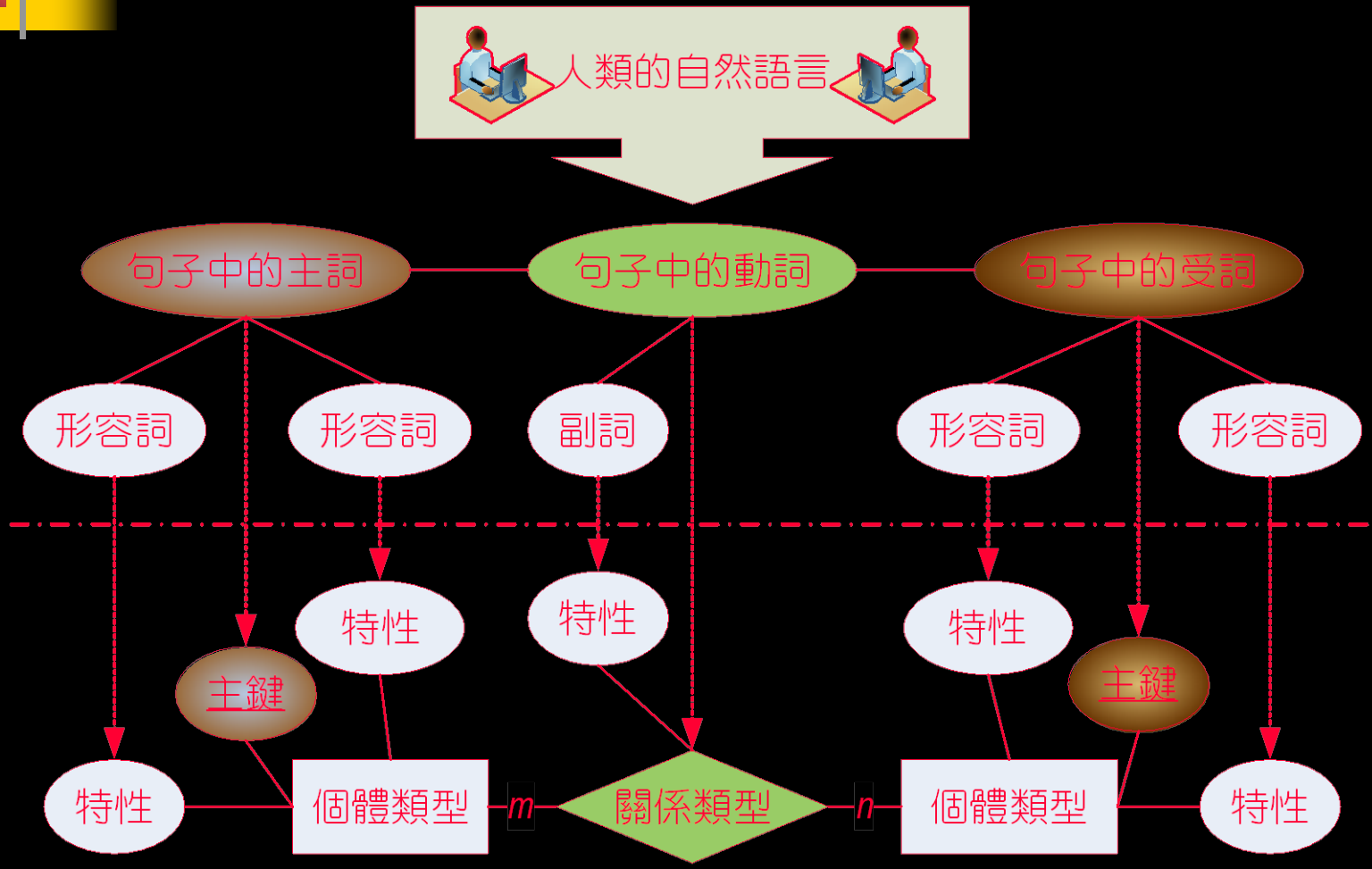
個體關係圖 (ERD)


Emp_id
years
Dept_id
age
city
name
Employee
m
Work_for
1
Department
name
1
Emp_Dep
n
Dependent
name
age

BOB 資料庫的 ERD


no
name
quantity
id
bookname
Bookstores
m
Orders
n
Books
author
rank
city
publisher
price

E-R 模式和自然語言的關係



ERD 與關聯式資料庫的對應

- 個體類型 (Entities Types)：將每個個體類型都分別以一個關聯表 E 來表示 (一般個體類型與弱勢個體類型都一樣)
  - 單一特性對應到關聯表上的一個屬性；
  - 複合特性中每一個特性對應到關聯表中的一個屬性；
  - 選定或引進一個新的屬性當做的主鍵；
  - 多重值特性則另外引進一個新的關聯表 M，M 中要增加一個屬性 FK 做為參考 E 的外來鍵；
  - 弱勢個體類型也要引進外來鍵以參考所相依之個體類型中的主鍵，而且要將此外來鍵中的 Delete 與 Update 選項都要設成 CASCADES。

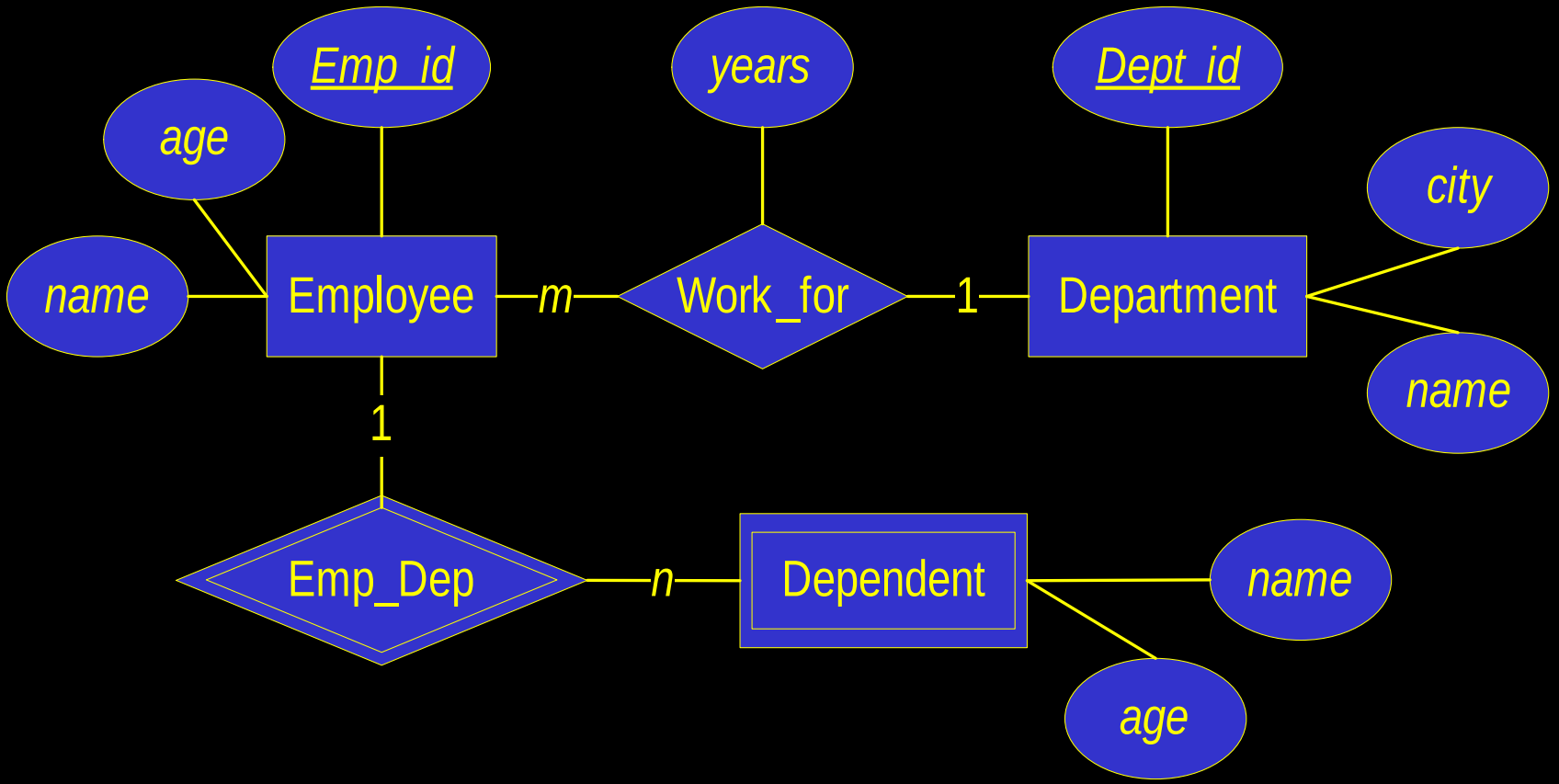
ERD 與關聯式資料庫的對應

- 關係 (Relationships)：將一般關係都分別以一個關聯表 R 來表示 (弱勢關係則不必，只要在弱勢個體中存放外來鍵即可)
  - 單一特性則對應到關聯表上的一個屬性；
  - 複合特性中每一個特性對應到關聯表中的一個屬性
  - 將所有參與者 (Participants) 關聯表 *P1* 、*P2* 、*...* 、*Pn* 的所有主鍵 *PK1* 、*PK2* 、*...* 、*PKn* 集合起來 (看做一個複合屬性 (*PK1, PK2, ..., PKn*)) 當作 R 的主鍵，每個 *PKi* 則是參考 *Pi* 的外來鍵

ERD 與關聯式資料庫的對應

- 若有多重值特性 (Multi-Valued Properties) 則要另外引進一個新的關聯表 *M*，*M* 關聯表中要多增加一個屬性 *FK* 做為 *M* 用來參考關聯表 *R* 的外來鍵，也就是說 *FK* 中儲存 *R* 的主鍵。
- 如果關係是 “一對多” (或多對一) 的話，我們也可以有些許變形：那就是只要在參與者為 “多” 的那一方之關聯表中多引進一個 “一” 那一方的關聯表之主鍵作為外來鍵，此外來鍵參考 “一” 那一方的關聯表。(見課本 9-38 頁的 Employee 例子)

請同學將此 ERD 轉成表格



子類型的實現方式

Student (學生)

no	*name*	*age*	*dept*
6502	Paul	20	CS
6506	Tom	22	CS
7501	Annie	25	CS
7602	Frank	27	MIS
8501	Jesse	30	CS

Undergraduate (大學部)

no	*name*	*age*	*dept*	***rank***
6502	Paul	20	CS	A
6506	Tom	22	CS	B

Graduate (研究生)

no	*name*	*age*	*dept*	*advisor*
7501	Annie	25	CS	Aho
7602	Frank	27	MIS	Kent
8501	Jesse	30	CS	Zadeh

陰影部分表示可不存資料

子類型的實現方式 (續)

Master (碩士班)

no	name	age	dept	advisor	type
7501	Annie	25	CS	Aho	Part Time
7602	Frank	27	MIS	Kent	Full Time

PhD (博士班)

no	name	age	aept	advisor	specialty
8501	Jesse	30	CS	Zadeh	Database

陰影部分可由母類型中用合併 (Join) 運算取得

陰影部分的取得方式

Undergraduate (大學部)

no	*rank*
6502	A
6506	B

Student (學生)

no	*name*	*age*	*dept*
6502	Paul	20	CS
6506	Tom	22	CS
7501	Annie	25	CS
7602	Frank	27	MIS
8501	Jesse	30	CS

大學部學生完整資料

Undergraduate $\bowtie_{no}$ Student

no	*name*	*age*	*dept*	*rank*
6502	Paul	20	CS	A
6506	Tom	22	CS	B

也可以透過 View 來做

- 如果要更省事的話，那麼只要將上述的各個合併運算結果定義成「視界」(Views)，例如：

Create View Undergraduate_Student (*no*, *name*, *age*, *dept*, *rank*)

As select Student.*no*, *name*, *age*, *dept*, *rank*

from Undergraduate, Student

where Undergraduate.*no* = Student.*no*;

如果沒有母類型與子類型？

- 全部都要放在 [學生] 個體類型中, 會產生許多 NULL 值

Student (學生)

no	*name*	*age*	*dept*	*rank*	*advisor*	*type*	*specialty*	*category*
6502	Paul	20	CS	A	—	—	—	Undergraduate
6506	Tom	22	CS	B	—	—	—	Undergraduate
7501	Annie	25	CS	—	Aho	Part Time	—	Graduate
7602	Frank	27	MIS	—	Kent	Full Time	—	Graduate
8501	Jesse	30	CS	—	Zadeh	—	Database	Graduate

將 E-R 圖轉成關聯表的原則

E-R 圖中的結構	關聯表表示法
一般的個體 (Entity)	建立一個具有主鍵以及相關屬性的關聯表。
弱勢個體 (Weak Entity)	建立一個具有「複合主鍵」(Composity Key) 以及相關屬性的關聯表，該複合屬性會包含本個體所依附的個體之主鍵屬性，以做為參考該個體的「外來鍵」。
一對一的一元或二元關係	將一方的主鍵放到另一方的關聯表中，或雙方皆放置對方的主鍵，以做為參考另一個體的「外來鍵」。
一對多的二元關係	將“一”方的主鍵放到“多”方的關聯表中，當做參考“一”方的「外來鍵」。

將 E-R 圖轉成關聯表的原則

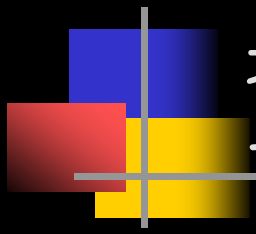
多對多的二元關係	另外為這個關係造一個關聯表，其主鍵是由有關係的兩個個體之主鍵所構成之「複合主鍵」，同時這兩個屬性為分別參考所屬原個體的「外來鍵」。
多重值特性 (Multi-Valued Properties)	另外引進一個新的關聯表 *M*，*M* 關聯表中要多增加一個屬性 *FK* 做為 *M* 用來參考關聯表 *R* 的外來鍵，也就是說 *FK* 中儲存 *R* 的主鍵。
IS-A 關係	建立一個母類型 (Superclass) 的關聯表，其中要包含所有子類型 (Subclass) 所共同具有的主鍵與非主鍵屬性，並對所有子類型建立一個單獨的關聯表，並皆以母類型的主鍵為主鍵，但是只包含該子類型所獨有的屬性。(見圖 9.14)

結論 (相依性定義與包容關係)

相依性名稱	相依性定義	例 子
功能相依性 (Functional Dependency)	對於一個關聯表 R 而言，我們說 R 中的屬性 B 功能相依於 R 中的屬性 A (或 R 中的 A 在功能上決定 R 的 B)，寫成 R.A→R.B，若且唯若在任何時候 R 中的 A 屬性值只會對應到一個 R 中的 B 屬性值。A 與 B 都可能是複合屬性 (Composite Attribute)。	各種「多對一」的關係都具有功能相依性，如：兒子與父親的關係。
多重值相依性 (Multi-valued Dependency) — 是功能相依性的進一步推廣	對於一個關聯表 R(A, B, C) 而言，我們說 R 中的屬性 B 多重值相依於 R 中的屬性 A (或 R 中的 A 多重值決定了 B)，寫成 R.A→→R.B，若且唯若 B 屬性中的值所成的集合在 R 中與 (A, C) 屬性中的值所構成的值組只與 A 的值有關，而與 C 的值無關。A、B、C 都可能是複合屬性。	例如：各學系、學生與必修課程間會有「學系」→→「學生」|「必修課程」。
合併相依性 (Join Dependency) — 多重值相依性的進一步推廣	假定 X, Y, ..., Z 都是關聯表 R 的投影結果，則 R 滿足了「合併相依性」：＊(X, Y, ..., Z)， 若且唯若 R 等於 X, Y, ..., Z 的合併結果。	如果你的應用有如 9.4 節所述之情況，便有此類相依性。

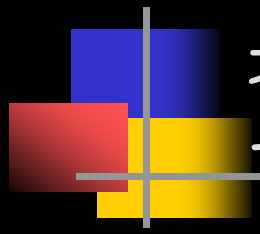
結論 (正規化型式的剖析)

正規化型式	要符合此型式的限制條件	要如此正規化的理由
第一正規化型式 (1NF)	所有的屬性值都只能是單一的值，不可以是一個集合。	簡化表示與處理方式，並且只需要一套更新運算即可。
第二正規化型式 (2NF)	要先符合 1NF，而且所有非候選鍵屬性都要完全功能相依於該關聯表的主鍵。	消除因「某些非候選鍵屬性只功能相依於主鍵的一部份」所產生的更新異常現象。
第三正規化型式 (3NF)	要先符合 2NF，而且所有的非候選鍵屬性都不能彼此功能相依。	消除因「非候選鍵屬性之間的功能相依性」所產生的更新異常現象。
廣義的第三正規化型式 (BCNF)	要先符合 2NF，而且所有的決定性屬性都是該關聯表的候選鍵。	消除因「某些候選鍵的部份屬性只功能相依於另一個候選鍵的一部份」所產生的更新異常現象。
第四正規化型式 (4NF)	要先符合 3NF 或 BCNF，而且該關聯表中不含有任何的多重值相依性。	消除因「多重值相依性」所產生的更新異常現象。
第五正規化型式 (5NF)	要先符合 4NF，且所有的合併相依性 *(X, Y) 中的 X 與 Y 的共同屬性是該關聯表的候選鍵。	消除因「合併相依性」所產生的更新異常現象。



資料庫的實務設計

- 以「個體-關係模式」(E-R Model) 為基礎，再對每一個關聯表思考是否進一步正規化，
- 考慮「查詢效率」與「更新時的異常現象」兩大因素。
- 透過 E-R Model 所產生的關聯表已經是 3NF。
- 利用商用軟體工具可以加速完成工作，例如：PowerDesigner、E-R Win、Rational Rose 等

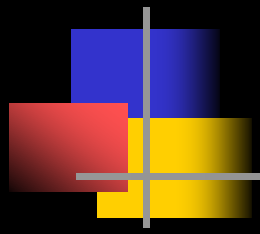


資料庫的實務設計

- 「個體-關係模式」做資料庫設計之步驟：
  - 與使用者溝通後確認需要那些屬性；
  - 找出需要那些個體，各包含那些屬性；
  - 找出個體間的關係與限制條件 (一對一、一對多，或是多對多)；
  - 轉成關聯表，並注意外來鍵參考情況下，如何設定 Delete 與 Update 選項？
  - 分別考慮每個關聯表，是否進一步正規化

反正規化 (De-normalization)

- De-normalization 可加速查詢，作法如下：
  - 將常常同時用到，卻又分別放在不同關聯表中的屬性在某個關聯表中拷貝，以增加執行效率。但是更新時要兩份拷貝一起更新，以維持一致性。
  - 若有關聯表只用於查詢，很少更新，則可以將它們正規化的結果再合併回來，以增進效率。
  - 常使用的加總計算預先算出，放在某關聯表中，以便在查詢時能直接存取。



本章結束
The End.