

第五章 關聯式資料模式的整合限制條件





本章內容

- 5.1 簡介
- 5.2 候選鍵 (Candidate Key)
- 5.3 外來鍵 (Foreign Key)
- 5.4 外來鍵參考圖 (Referential Diagram)
- 5.5 關聯式模式的兩條整合限制規則
- 5.6 外來鍵使用規則
- 5.7 在關聯表上定義額外的整合限制條件
- 5.8 利用「個體-關係模式」來規劃資料庫



簡介

- 整合限制條件 (Integrity Rule) 是用來規範資料在關聯表中的儲存、刪除或更改動作，以防止不一致或錯誤的情況發生
- 「資料庫整合性」 (Database Integrity) 應該包含兩部分 [Motro (1989b)]：
 - 正確性 (Validity)：確保錯誤資料都已經被排除在資料庫之外；
 - 完整性 (Completeness)：保證所有正確資料都已包含在資料庫中。
- 關聯式模式最初的兩條整合規則
 - 1. 個體整合限制規則 (Entity Integrity)：規範關聯表內部的限制條件
 - 2. 參考整合限制規則 (Referential Integrity)：規範關聯表之間的限制條件



延伸性整合限制條件

- 近年陸續加入以下兩類延伸性整合限制條件，擴充成為**四類規則**：
 - **3. 值域整合限制條件 (Domain Integrity)**—
 - 以 Check 條件約束與規則 (Rule) 限制資料內容/格式、
 - Create Default 規則設定預設值，以及
 - Not Null 宣告來限制屬性值的存在必要性。
 - **4. 使用者自訂整合限制條件 (User-Defined Integrity)**—
 - 使用預儲程序 (Stored Procedure)、
 - 使用者自訂函數 (User-Defined Function) 或
 - 觸發程序 (Trigger) 來撰寫規範。



簡介 (續)

- 整合限制規則，都是在基底關聯表上訂定
- 導出 (衍生) 關聯表 (Derived Relations)、視界或其它類型的關聯表，會繼承所屬基底關聯表上的整合限制規則
- 沒有完備的整合限制條件會造成資料庫內容的大亂，就如社會缺乏法律一樣，絕對不可以忽略



候選鍵 (Candidate key)

- 主鍵是一個唯一的識別值 (Unique Identifier)
- 主鍵的屬性子集需滿足「候選鍵」(Candidate Key) 的條件
- 一個關聯表的候選鍵可能有好幾個
- 主鍵由這些候選鍵中選出其一
- 沒有被選為主鍵的候選鍵稱為「替代鍵」(Alternate Key)

候選鍵應具備的條件

- 候選鍵應具備的條件

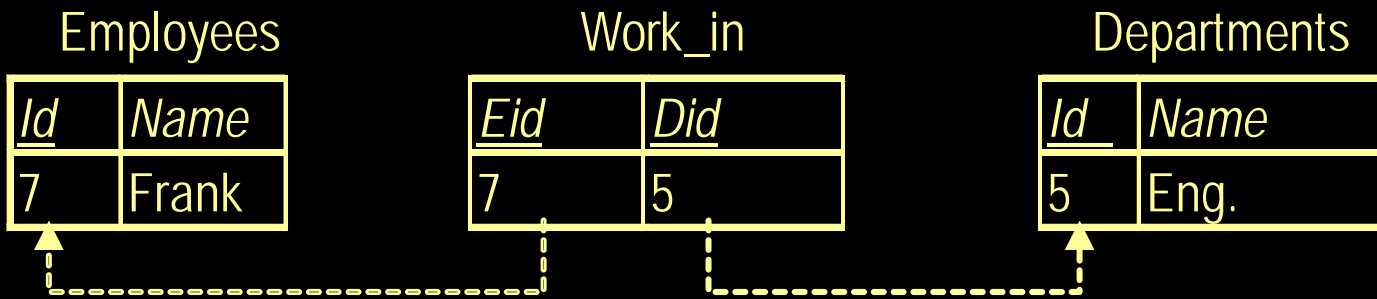
- 唯一性 (Unique Property)—沒有兩個值組的候選鍵有相同的值 (如：下表的 *no*, *cname*, *ename*, 或 (*no*, *cname*))
- 最小性 (Irreducibility Property, 或 Minimality)—該屬性子集如果去除任一個屬性時，便不再符合唯一性 (如：(*no*, *cname*) 便不具此特性)
- 符合唯一性的屬性子集在關聯表中可能會有很多個，我們統稱這些屬性子集為「超級鍵」(Super Key)，而候選鍵就是那些具有最小性的超級鍵。

Students

<i>no</i>	<i>cname</i>	<i>ename</i>	<i>age</i>
1	張三	Henry	20
6	李四	Tom	26

候選鍵 (續)

- 若候選鍵僅有一個，則它便是主鍵
- 任何關聯表一定會有一個以上的候選鍵
- 最極端的情況便是：所有屬性全部構成關聯表的候選鍵，如下圖 Work_in 主鍵為 (Eid, Did)





候選鍵 (續)

- 在集合裡要定位到每一筆值組就要靠候選鍵
- 所以學校會編學號、政府會編身份證字號、公司會編員工代號
- 實際應用不一定要有主鍵，但一定要有候選鍵
(但我們還是建議要對關聯表建立主鍵)
- 通常是另外編代號來當做主鍵
- 如何挑選最好的主鍵？(見下頁)



如何挑選主鍵？

- 選擇每一個個體都有，而且永遠不會變更其值的屬性。如：身份證字號，學號等 (地址、行動電話號碼會變更，並不適合)。
- 確保不會是虛值的屬性。
- 不要用人工才能解讀的編號鍵值。例如，原料代號 HK5838 中的 HK 代表某一倉庫位置，但倉庫卻可能常常更換
- 儘量以單一的屬性來代表整筆值組。



外來鍵 (Foreign Keys)

- 觀察 BOB 資料庫中的關聯表
 - 在 Orders 中，值組 (8, 7, 50) 不是一個合法值組。
 - Orders.no 要包含於 Bookstores.no 而且 Orders.id 要包含於 Books.id
 - 正如下頁之表格所示
 - Orders.no 中的任何屬性值都必須出現在 Bookstores.no 中 (下頁中有打 X 的表示不合法的值)
 - Orders.id 中的任何屬性值都必須出現在 Books.id 中
 - 請見下頁的圖...

外來鍵參考的意義：

no 與 *id* 為關聯表 Orders 中的外來鍵

Bookstores

<u>no</u>	name	rank	city
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市

Orders

<u>no</u>	<u>id</u>	quantity
1	1	30
1	2	20
3	3	40
1	4	20

Books

<u>id</u>	bookname	author	price	publisher
1	三國演義	羅貫中	120	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





外來鍵的定義

- 定義5.1：[外來鍵 (Foreign Key)]一個在 R_2 中的外來鍵 FK 是由 R_2 屬性集的子集合所構成，而且合乎下面兩點：
 - FK 中的每個屬性值不是全為虛值，就是全非虛值，
 - 存在某個基底關聯表 R_1 具有候選鍵 CK ，使得具有非虛值的 FK 屬性值與 R_1 中的某個值組之 CK 屬性值完全相等 (注意： R_1 與 R_2 可以是同一個關聯表)。

外來鍵的注意事項

- 關聯表 R 的外來鍵也可以參考 R 本身的屬性，稱為「父子式關聯性」(Parent-Child Relationship)

Employees

<u>ENo</u>	Name	Manager	Salary
A001	Mike	—	60k
E001	Frank	—	80k
E002	John	E001	48k
S001	Tom	—	70k

- 外來鍵不一定是關聯表中之主鍵的一部份
- 外來鍵不是所屬關聯表中之主鍵的一部份時，則該外來鍵的值也可以是一個虛值 (Null)

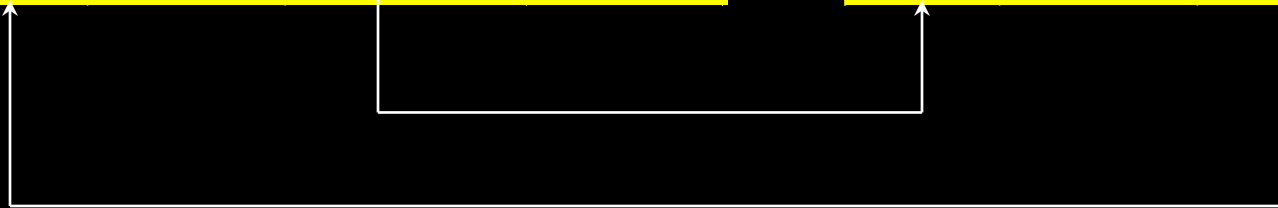
外來鍵不一定是主鍵的一部份

Departments

<u>dno</u>	Name	Manager	Budget
a1	會計	A001	40000
e1	工程	E001	50000
p1	企劃	—	48000
s1	銷售	S001	34000

Employees

<u>ENo</u>	Name	dno	Salary
A001	Mike	a1	60k
E001	Frank	e1	80k
E002	John	e1	48k
S001	Tom	s1	70k

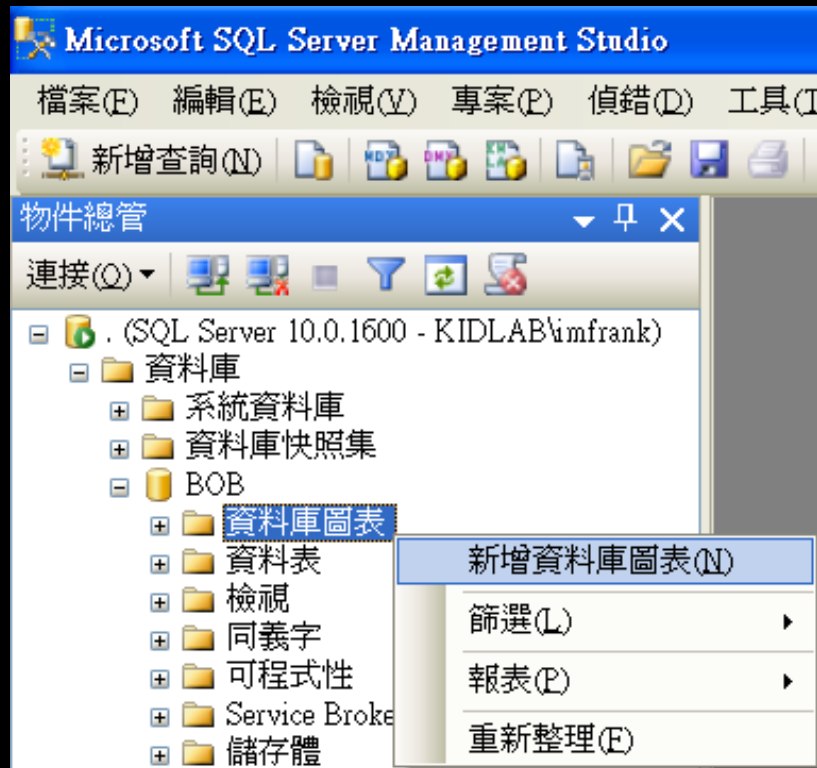


外來鍵參考圖 (Referential Diagram)

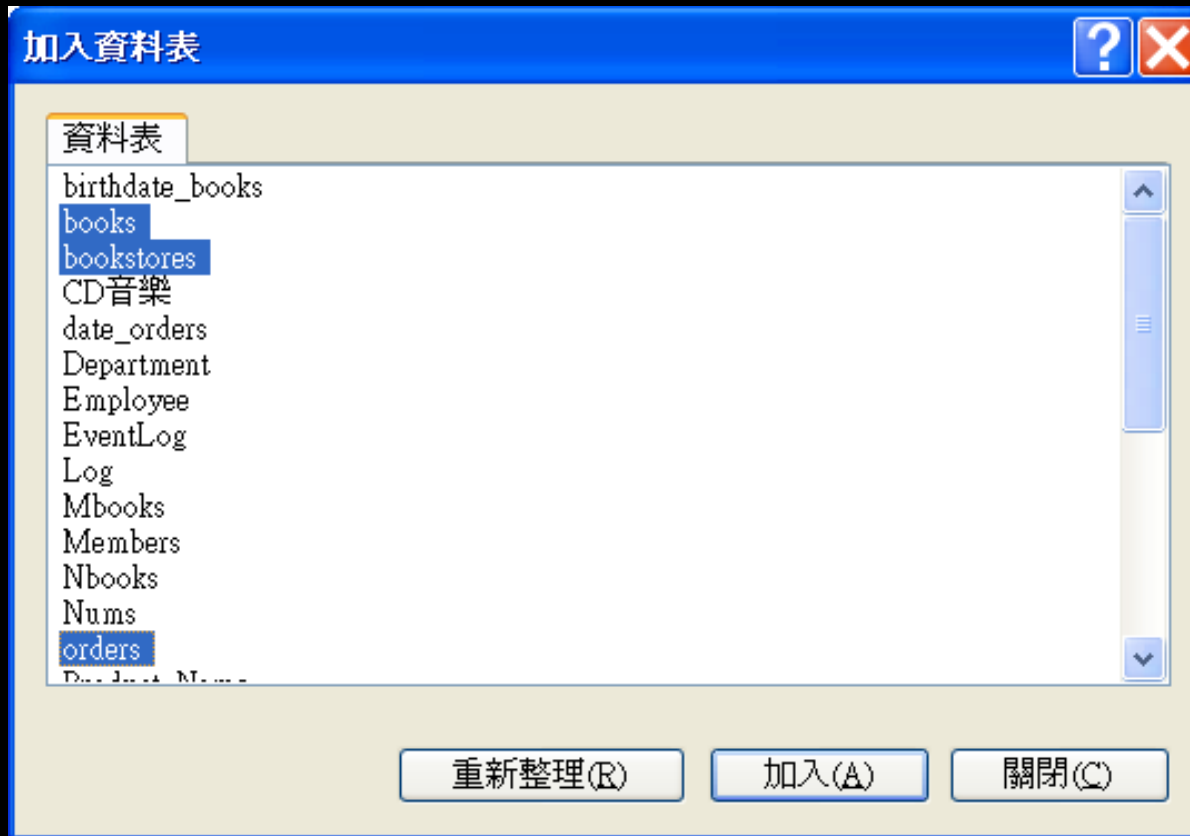
- Bookstores \xleftarrow{no} Orders \xrightarrow{id} Books
- $R_3 \rightarrow R_2 \rightarrow R_1$
- 外來鍵參考鏈 (Referential Chain)
 $R_n \rightarrow R_{n-1} \rightarrow \dots \rightarrow R_1$
- 外來鍵參考環 (Referential Cycle)
 $R_n \rightarrow R_{n-1} \rightarrow \dots \rightarrow R_1 \rightarrow R_n$ ($R_1 \rightarrow R_1$)

取得外來鍵參考圖

- 在 SQL Server 2008 中，可以用 [新增資料庫圖表] 工具自動取得外來鍵參考圖



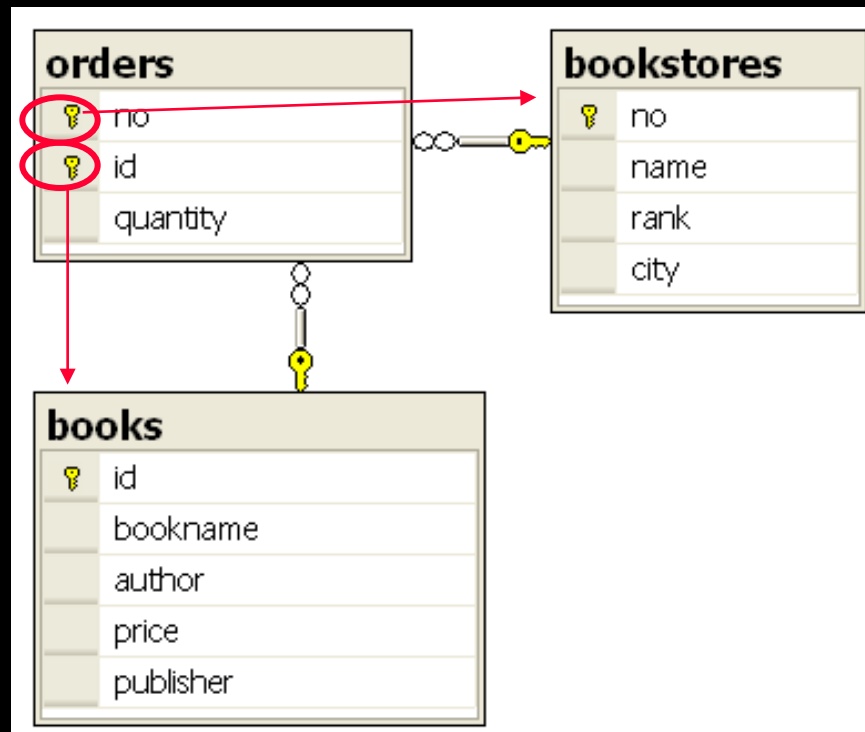
選取關聯表



在 SQL Server 建立外來鍵

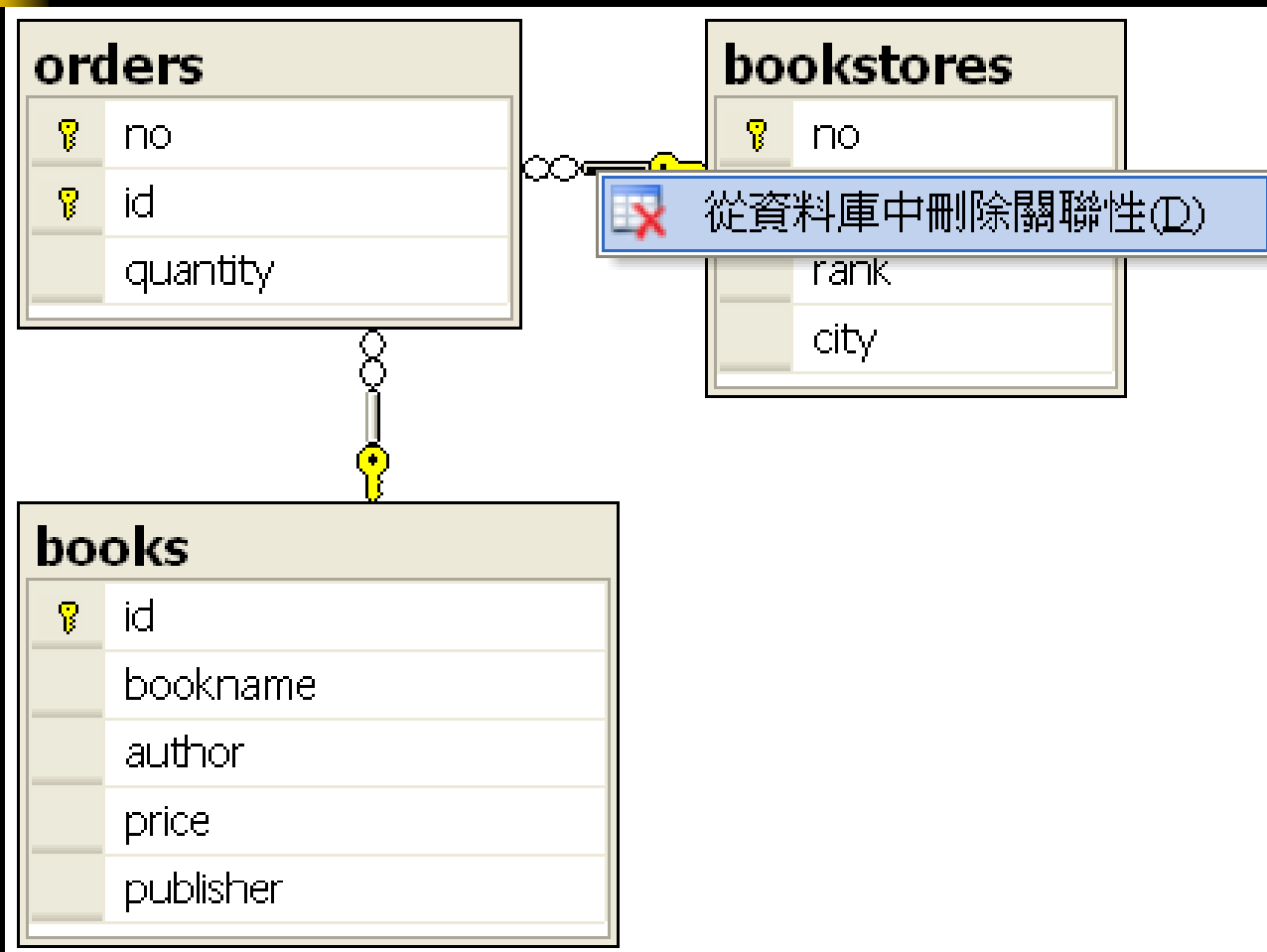
- 點選 BOB 資料庫的 Diagrams，然後按滑鼠右鍵，並選取 [新增資料庫圖表...]

以滑鼠點選
紅圈處往兩個
箭頭方向拉



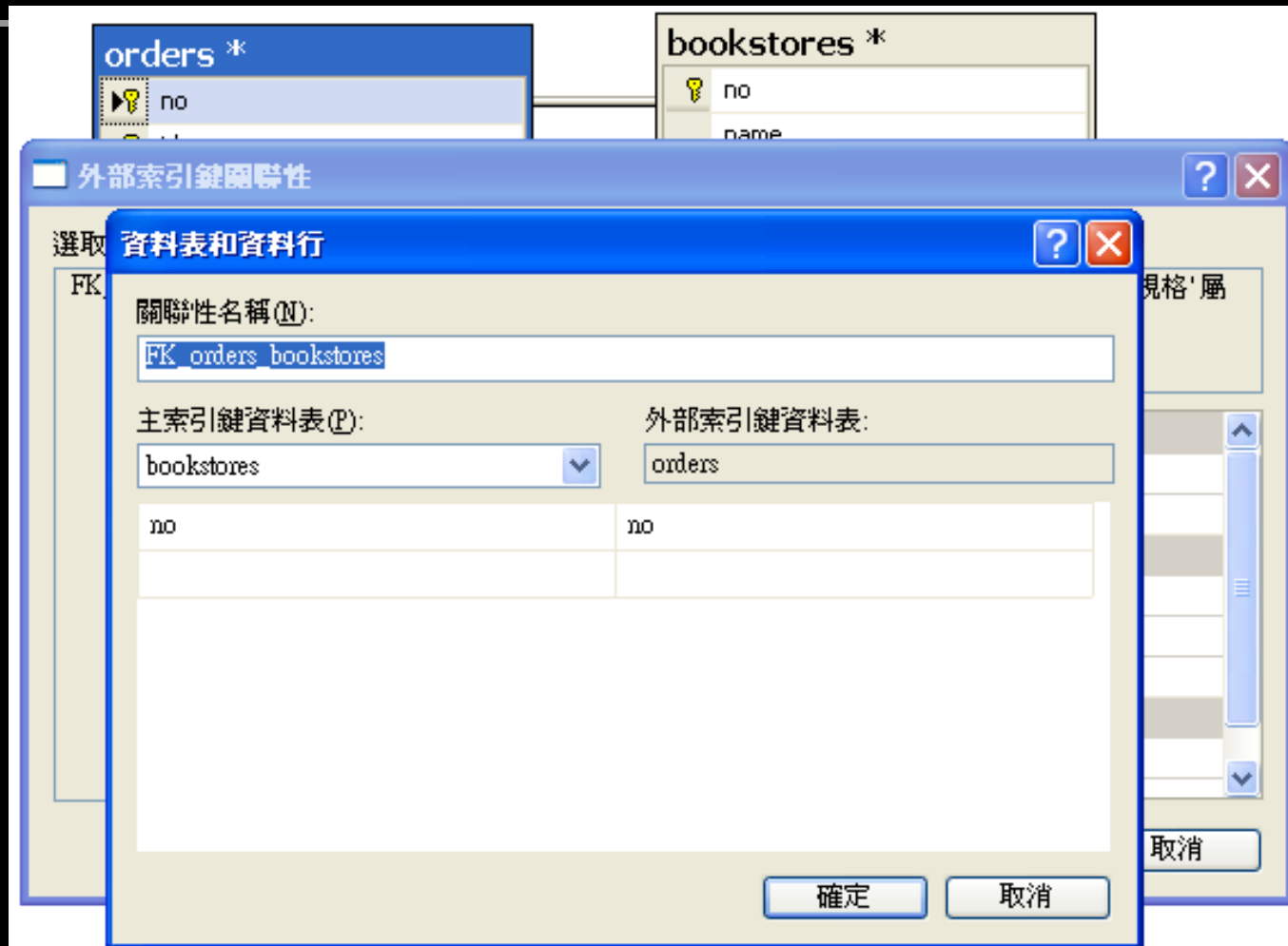
完成後記得存檔
並加以命名

刪除外來鍵關聯性的畫面



點選箭頭後
按滑鼠右鍵

建立外來鍵關聯性的畫面





關聯式模式的整合限制規則

- 個體整合限制 (Entity Integrity)—規範關聯表內部的限制條件
- 參考整合限制 (Referential Integrity)—規範關聯表與關聯表之間的限制條件

個體整合限制

- 基底關聯表主鍵的任何屬性值都不可可以是某個內定值—當然更不可可以是虛值 (Null)。
 - 每一個實體都必須是可以分辨的 (Distinguishable)
 - 主鍵為虛值表示該個體是一個完全不確定的個體
 - 查詢處理上的方便性與務實性：
請問下列的關聯表含有幾家書局的資料？(不知道)

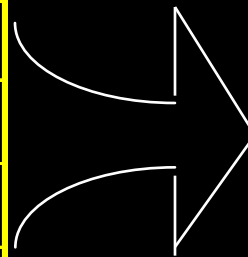
<i>no</i>	<i>name</i>	<i>rank</i>	<i>city</i>
1	巨蟹書局	20	臺北市
—	巨蟹書局	10	高雄市

主鍵的注意事項

- 若主鍵是由複合屬性所構成的話，則全部都不能是虛值
- 個體整合限制條件原則上只適用於基底關聯表

<u>no</u>	name	class	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	臺中市
5	獅子書局	30	—

查詢結果



city
臺北市
高雄市
新竹市
臺中市
—



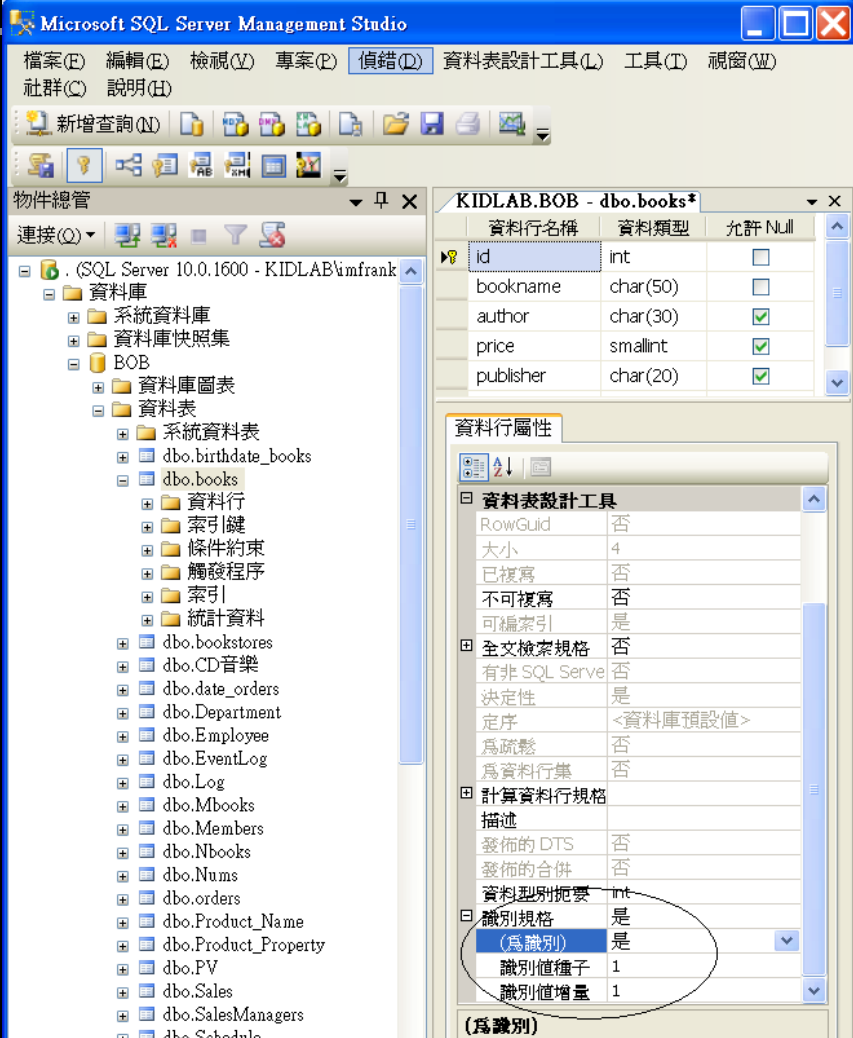
廣義的個體整合限制規則

- 更廣義的個體整合限制規則也可以寫成：
在基底關聯表中，主鍵的任何一個屬性，都不可以存放「預設值」(Default Values)，而虛值就是一種特殊的預設值。
- 主鍵中的屬性，雖然不可存放預設值 (Default Values)，但是卻可以存放預設系統函數，只要該函數每次都產生具有唯一性的值即可。



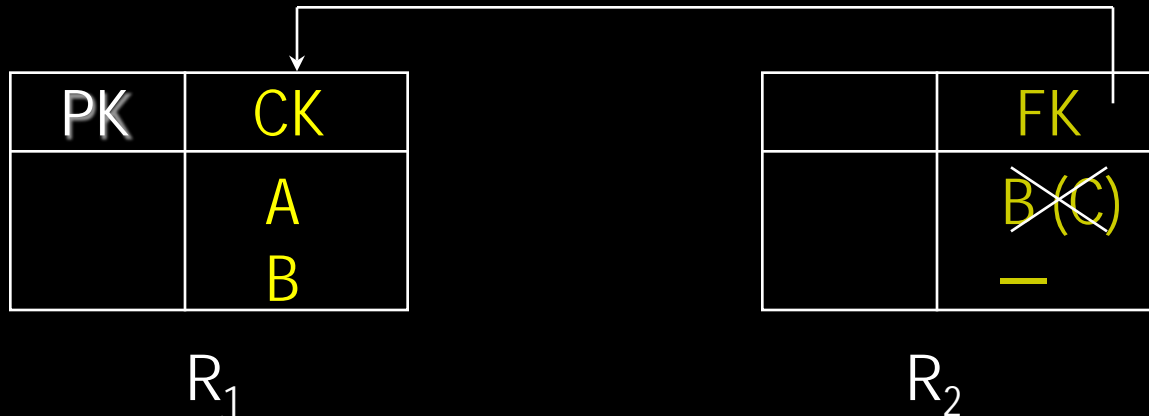
主鍵可以存放預設系統函數

- 只要該函數每次都產生具有唯一性的值即可
 - 設定屬性的「識別規格」(Is Identity) 特性為「是」：以便讓系統自動產生遞增的識別值。
 - 預設成 `getdate()` 函數直接抓取系統時間：因為 `datetime` 的精確度達到 3.33 毫秒 (ms, 即 3.33/1000 秒)，只要確保產生值組的速度不超過此時段即可。
 - 對於 `uniqueidentifier` 資料型態的屬性來說：可以預設成 `newid()` 函數，它會直接傳回一個 128 bits 的唯一編號。



參考整合限制

- R_2 中的外來鍵 FK，若參考到關聯表 R_1 中的候選鍵 CK 時，則所有 R_2 中的 FK 值一定要符合底下其中一項條件：
 - 全部屬性的值都是虛值，或
 - 等於 R_1 中某值組的候選鍵值。



回顧前面的例子

Departments

<u>dno</u>	Name	Manager	Budget
a1	會計	A001	40000
e1	工程	E001	50000
p1	企劃	—	48000
s1	銷售	S001	34000

Employees

<u>ENo</u>	Name	dno	Salary
A001	Mike	a1	60k
E001	Frank	e1	80k
E002	John	e1	48k
S001	Tom	s1	70k

- Employees.*dno* 包含於 Departments.*dno*
- Departments.*Manager* 包含於 Employees.*Eno*



外來鍵使用規則

- 對關聯表新增/刪除/修改時要符合下列規則：
 - 外來鍵的**虛值規則** (Null Rule for Foreign Key)—既是外來鍵又是主鍵的屬性不可以新增或更改為虛值。如：BOB 資料庫中的 *Orders.no* 與 *Orders.id*
 - 外來鍵**參考對象的刪除/更新規則** (Delete/Update Rule for Foreign Key Reference)—刪除或更新參考對象後，如何維持「參考整合限制」？

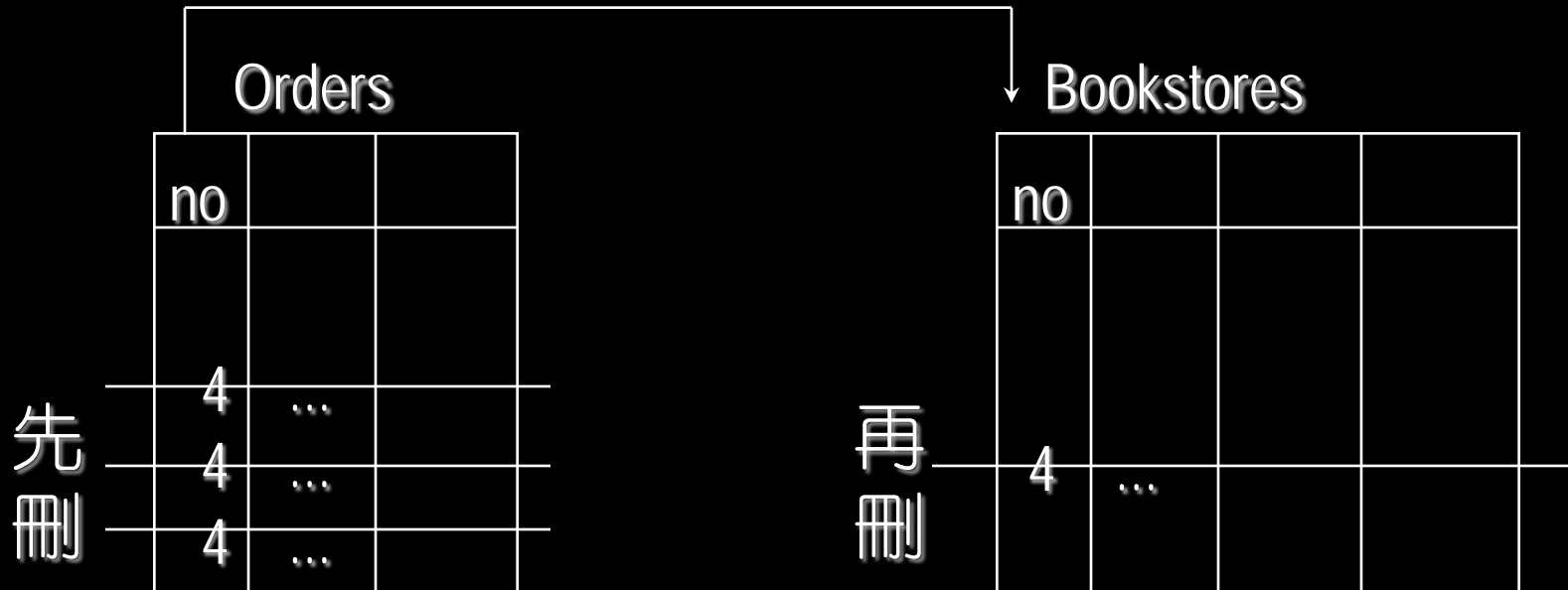


外來鍵參考對象的刪除/更新規則

- 例如：由 Bookstores 中刪除 (4, 天秤書局, ...) 後，Orders 中的 (4, ...) 便不符參考整合限制了
- 所以想刪除 Bookstores 中的 (4, 天秤書局, ...)，可以採用下列做法
 - 限制性做法 (RESTRICTED)
 - 連鎖反應的做法 (CASCADES)
 - 虛值化的做法 (NULLIFIES)

限制性做法 (RESTRICTED)

- 在沒有刪除 Orders 中所有的 (4, ...) 以前不准刪除 Bookstores 中的 (4, 天秤書局, ...)



連鎖反應做法 (CASCADES)

- 刪除 Bookstores 中的 (4, 天秤書局, ...) 之後
順便要將 Orders 中所有的 (4, ...) 也一併刪除



虛值化做法 (NULLIFIES)

- 但是 no 在 Orders 中為主鍵的一部分，所以不合法，在此情況下不能使用此一做法



可以使用虛值化做法的情況

- 刪除 (E001, Frank, —, 80k) 之前與之後的情況

Employees

<u>ENo</u>	Name	Manager	Salary
A001	Mike	—	60k
E001	Frank	—	80k
E002	John	E001	48k
S001	Tom	—	70k

刪除前

Employees

<u>ENo</u>	Name	Manager	Salary
A001	Mike	—	60k
E001	Frank	—	80k
E002	John	—	48k
S001	Tom	—	70k

刪除後並虛值化



提供參考的大原則

- 外來鍵參考對象的刪除/更新規則使用原則：
 - 若所欲刪除／更新的資料牽涉到**實體的人或物品**，則使用「**限制性作法**」。E.g., 畢業學生要離校前的離校手續會牽涉到歸還圖書，則採用限制性作法。
 - 若所欲刪除／更新的資料牽涉到**虛擬的概念**，則使用「**連鎖反應作法**」。E.g., 畢業學生離校前的電腦登入帳號，可採用連鎖反應作法。



注意事項

- 因為維持外來鍵參考整合規則必須要全部做完，不然就全部都不做，以維持一致性
- 所以，不論採用何種做法，整個一連串的動作皆要看成一個「異動」(Transaction)
- 上述三種做法可以在定義資料綱要的時候下指令規範之，放在應用程式的執行邏輯較不好，當然最好是資料庫綱要與程式邏輯都有雙重把關



SQL Server 2008 上的作法

- 在 SQL Server 2008 版本中，已經都提供了「連鎖反應作法」與「限制性作法」。
- SQL Server 2008 預設的外來鍵參考對象的刪除／更新規則是「限制性的作法」。
- 也可以透過自行撰寫的預儲程序或觸發程序 (Triggers—在第七章有更詳細的例子) 來控制外來鍵參考的整合性。



定義額外的限制規則

- 因應用範圍的不同可以訂定：

欄位是否要強制唯一 (Unique) ？

欄位的預設值 (Default Value) 為何？

欄位的範圍 (Range) 為何？

欄位的格式 (Format) 為何？



定義額外的限制規則

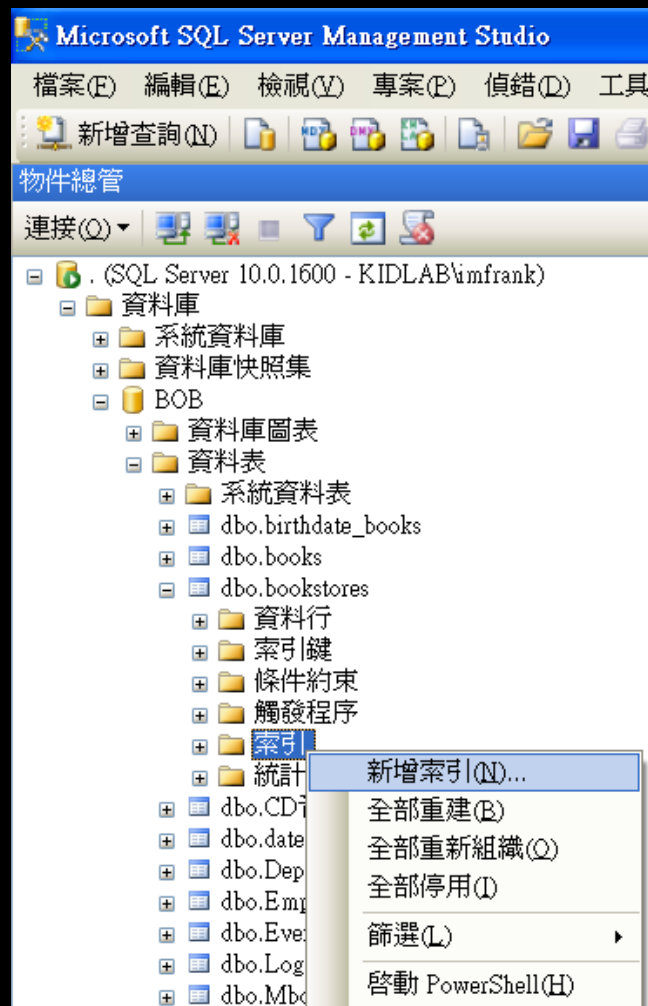
- Bookstores 中的規範
 - *no* 的屬性值必須要屬於正整數。
 - *name* 的屬性值必須是長度至少為 8 的字串
 - *rank* 的屬性值必須要屬於正整數
 - *city* 的屬性值必須要是長度至少為 6 的字串
 - 不能有兩個值組的 *no* 屬性值為相同
- 主要應由資料庫管理系統來檢查（而非應用程式）



加入額外的限制規則

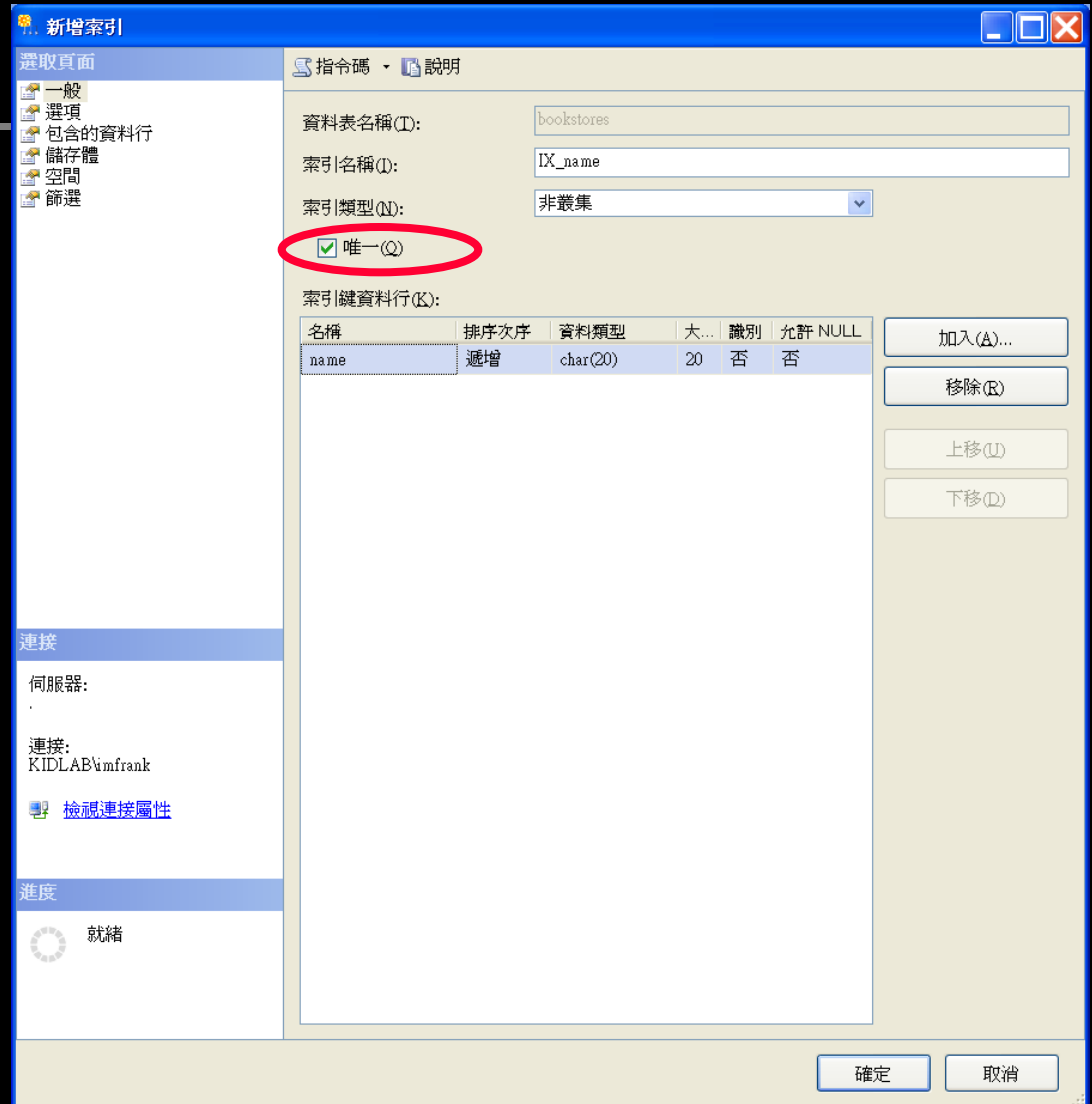
- 依資料庫的實際意義與應用，機動加入額外的整合限制條件（以下使用 Prolog 語法來說明）
- **Bookstores** (no, name, rank, city):- rank > 5, rank < 50. 可將 rank 的屬性值限制在 (5, 50) 之間
- **Employees** (id, name, age):- age >= 18.
表示本公司不雇用未滿十八歲的童工

設定 Unique Constraints



設定 Unique Constraints

選取所要設定的屬性 *name*



設定 Check Constraints

檢查條件約束

選取的 檢查條件約束(S):

CK_bookstores

正在編輯現有檢查條件約束的屬性。

☐ (一般)

運算式 rank > 5 and rank < 50

☐ 資料表設計工具

於 INSERT 及 UPDATE 時強制 是

強制複寫 是

檢查建立或重新啓用時的現 是

☐ 識別

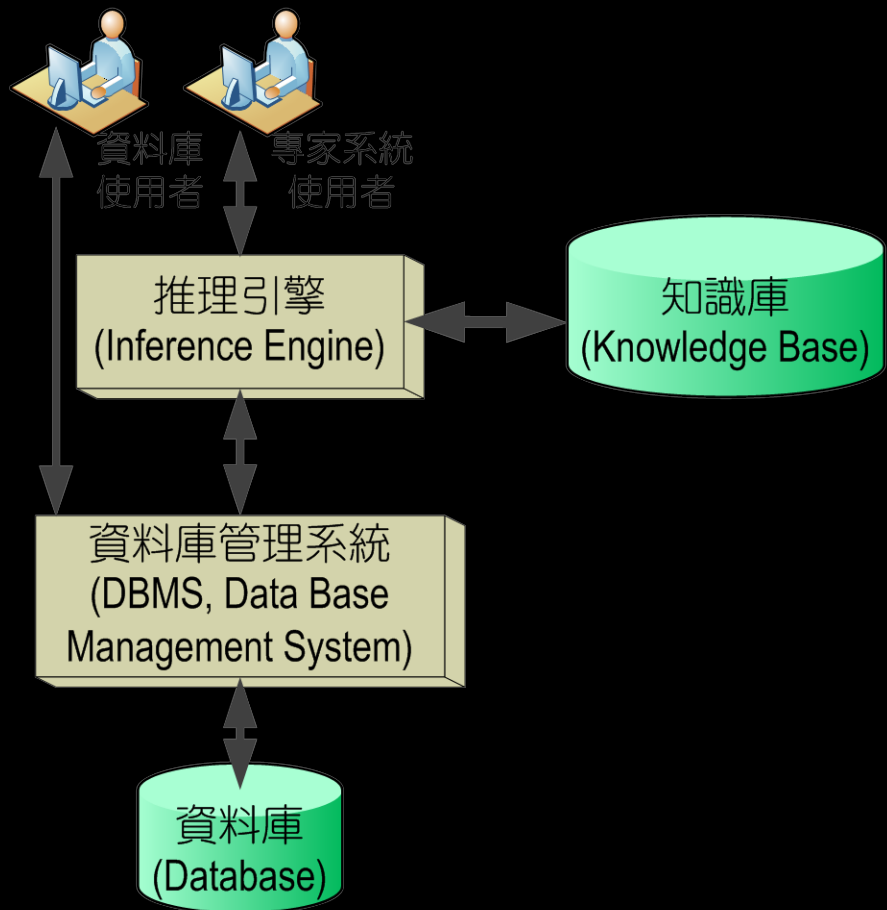
(名稱) CK_bookstores

描述

加入(A) 刪除(D) 關閉(C)

限制條件 vs. 邏輯規則

- 限制條件：關聯表名稱出現在規則左方
- 邏輯規則：關聯表名稱出現在規則右方
- 邏輯規則可以建構知識庫系統 (Knowledge-Based Systems)



邏輯規則範例

- Grandfather (X, Z) :- Father (X, Y), Father (Y, Z).

Father

<i>daddy</i>	<i>son</i>
John	Mike
Mike	Homer
Frank	Tom

Father

<i>daddy</i>	<i>son</i>
John	Mike
Mike	Homer
Frank	Tom

Grandfather

X	Z
John	Homer



整合限制規則的實現方式

- 使用 SQL 中的資料定義語言 (DDL)
- 使用觸發程序 (Triggers)
- 放在預儲程序 (Stored Procedures) 中，由應用程式呼叫之
- 放在應用程式的程式邏輯與流程控制當中
如：早期撰寫 Clipper、FoxPro、dBASE 的方式
- 越前面的方式越佳



預儲程序與觸發規則

■ 優點：

- 將程式邏輯與資料庫的存取作業分開，容易維護
- 適合用在主-從式架構或分散式架構上

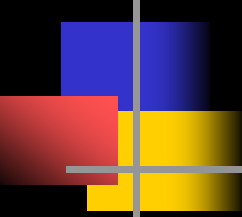
■ 缺點：

- 程式邏輯與資料庫的存取作業分開，複雜度增加
- 開發者與程式的維護者還要再學習另一種語言
- 觸發程序一多，可能會有彼此衝突的情況發生
- 不適合用來整合與跨越異質性的資料庫伺服器



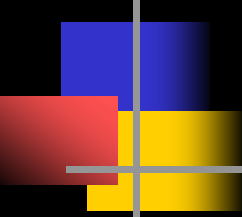
用「個體-關係模式」 做邏輯資料庫設計

- 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：分為一般個體 (Regular Entities) 與弱勢個體 (Weak Entities)；弱勢個體的存在與否，完全取決於某個一般個體。弱勢個體，如：員工的家屬、學生的家屬等，一旦員工離職 (或學生畢業、退學) 後，所有家屬資料對於公司 (或學校) 而言，已經沒有存放的必要，所以也會一併刪除。
 - 個體類型 (Entity Type)：同類型個體組成個體類型 (Entity Type)，而該類型中的每一個體，稱為實例 (Instance)。



個體-關係模式 (E-R Model) 中的概念 (續)

- 個體-關係模式 (E-R Model) 中的概念 (續)
 - 特性 (Properties)：同類型的個體有共同的特性，
 - 可分成單一特性 (Single Properties) 或複合特性 (Composite Properties)。
 - 特性也會構成鍵值 (Key)。特性的值，可以是單一值 (Single-Valued)、多重值 (Multi-Valued)，或不確定的虛值 (Unknown 或 Inapplicable)
 - 特性本身則可以是一個基本特性 (Basic Attribute) 或導出的特性 (Derived Attribute) (一般以虛線的橢圓表示)

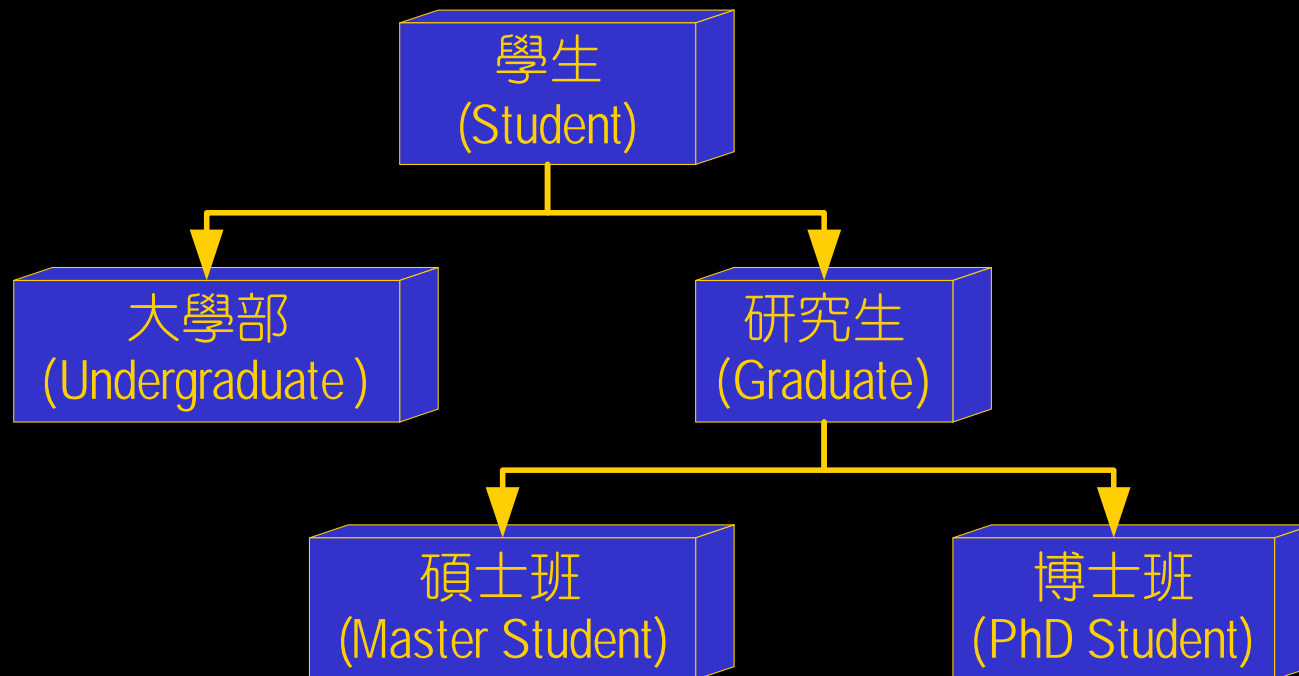


個體-關係模式 (E-R Model) 中的概念 (續)

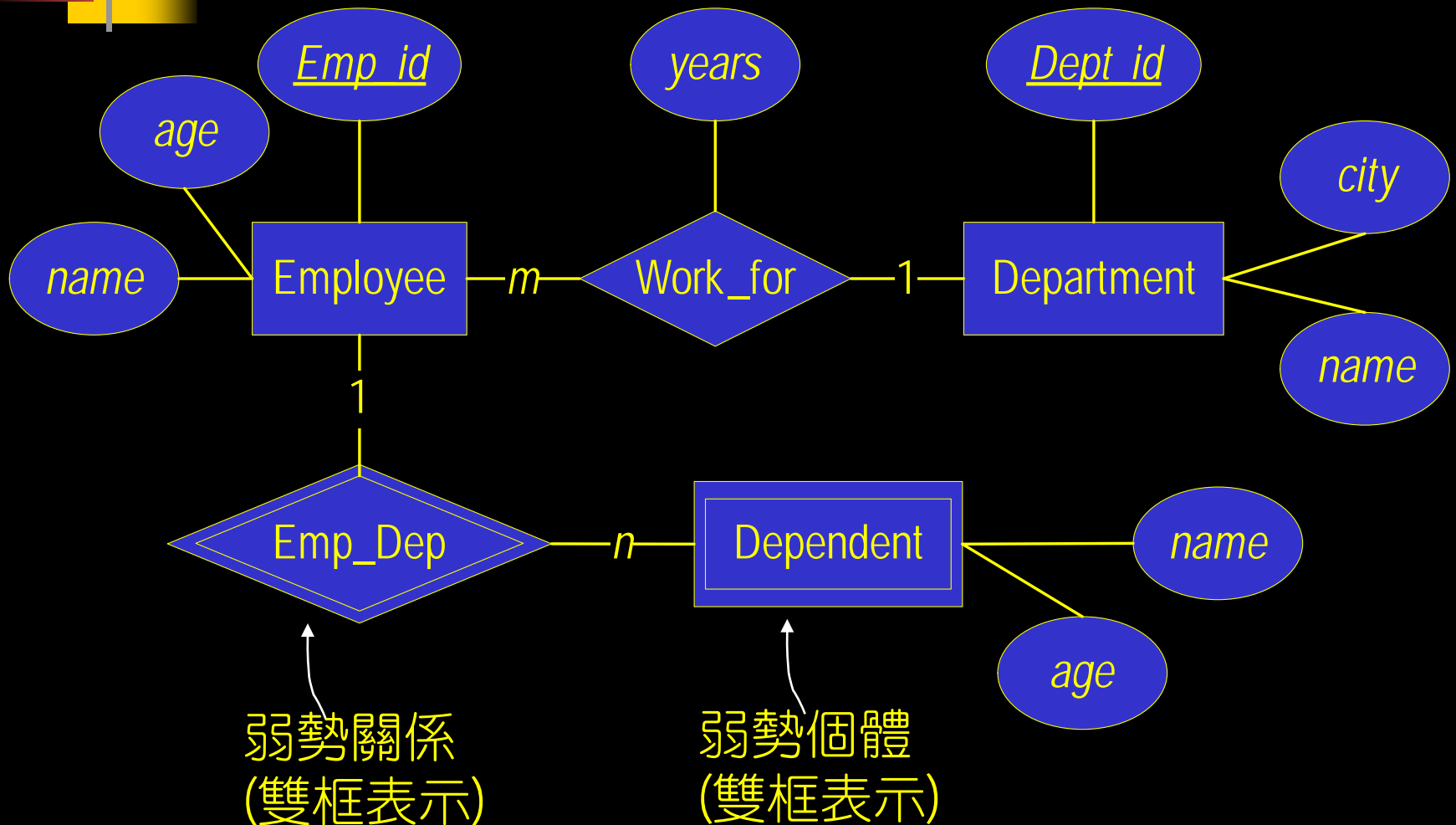
- 關係類型 (Relationships Types)：個體類型之間的關係稱作「關係類型」(Relationships Types)，
 - 關係類型所涉及的個體類型可以只有一個也可以多個，
 - 這些有關聯的個體類型我們稱為「參與者」(Participants)，
 - 參與者的數量稱為該關係類型的「等級」(Degree)。
 - 一般個體與弱勢個體之間的關係類型則稱為「弱勢關係類型」(Weak Relationship Type)。
 - 個體類型之間的關係型式可以是 1 對 1 (one-to-one)、多對 1 (many-to-one)，或多對多 (many-to-many)。
 - 有關聯的個體資料均會被放到關係類型裡，所以關係類型的每筆資料都是一個「關係」(Relationship)。

新增的個體-關係模式概念

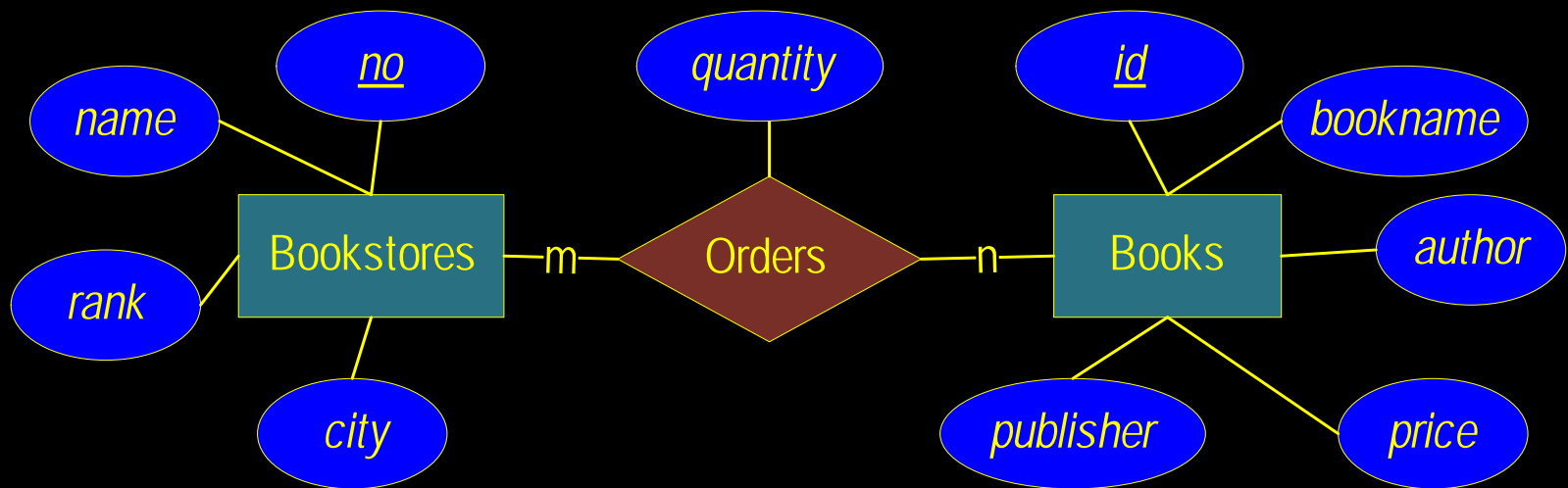
- 子類型 (Subtypes)：演變成「物件導向資料模式」(Object-Oriented Data Model) 的基礎



個體關係圖 (ERD)

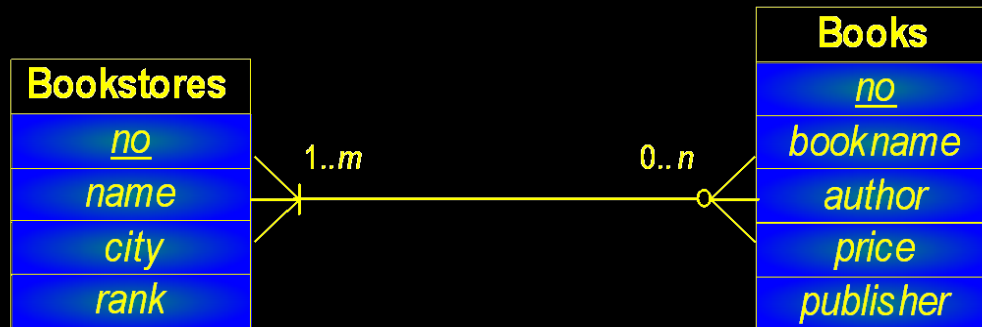


BOB 資料庫的 ERD



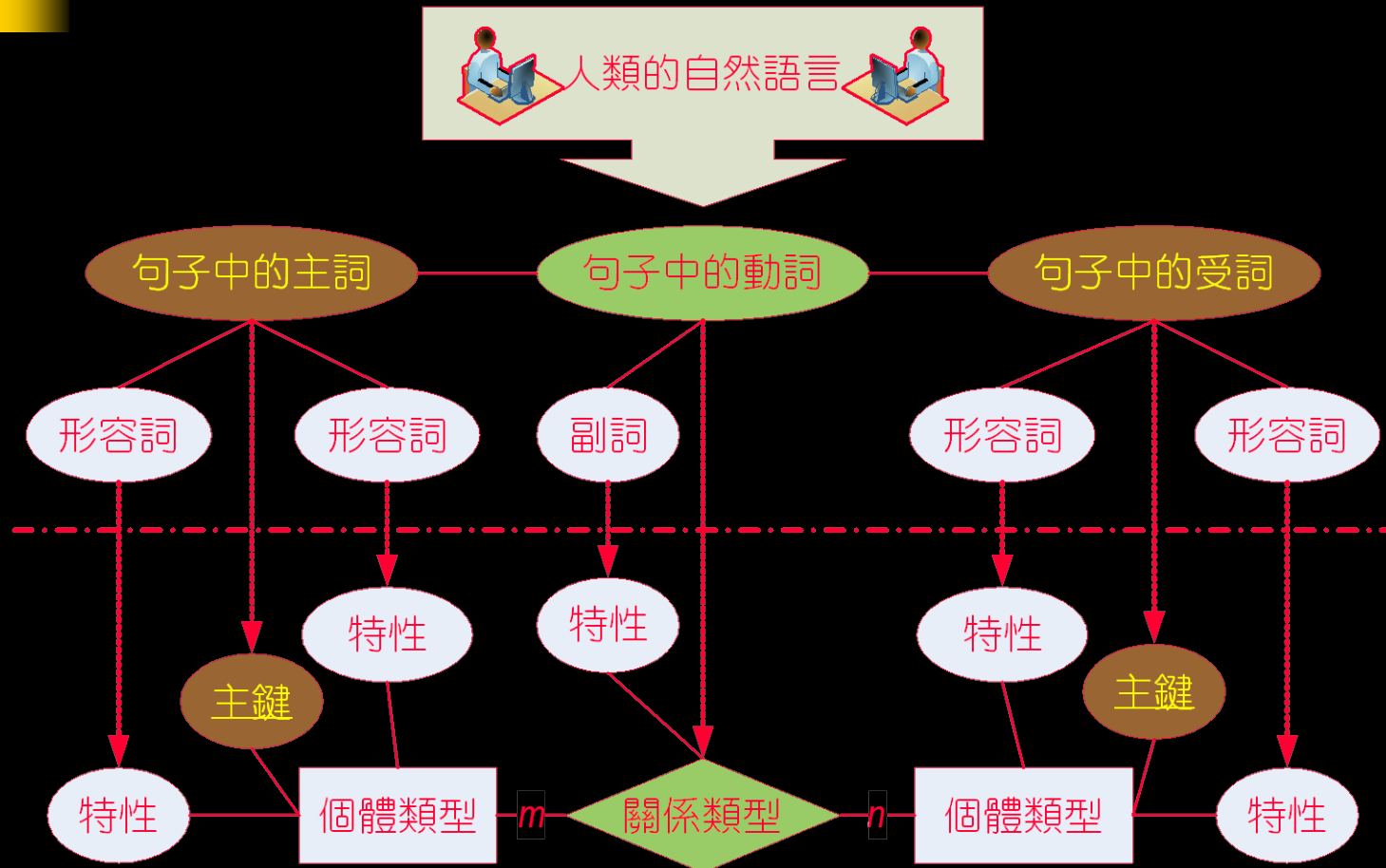
E-R Diagram 的變形

- E-R Diagram 有許多種繪法，在業界最常見的圖稱為「烏爪圖」(Crow's Feet Diagram)，主要目的大多是要節省繪圖的空間。例如，BOB 資料庫：



- 本書為忠於原作，採用 Peter P.S. Chen 原始繪法來說明

E-R 模式和自然語言的關係



請同學們舉例說明之



資料庫規劃

- 先確立應用系統需要的所有「個體類型」(Entity Type)。舉例來說，課程系統：會有「學生」(Student)、「課程」(Course) 與「教師」(Teacher) 三類個體類型。
- 找出個體類型之間的靜態「關係類型」(Relationship Type)，並探討其關係是「一對一」(one-to-one)、「一對多」(one-to-many) 或「多對多」(many-to-many)。

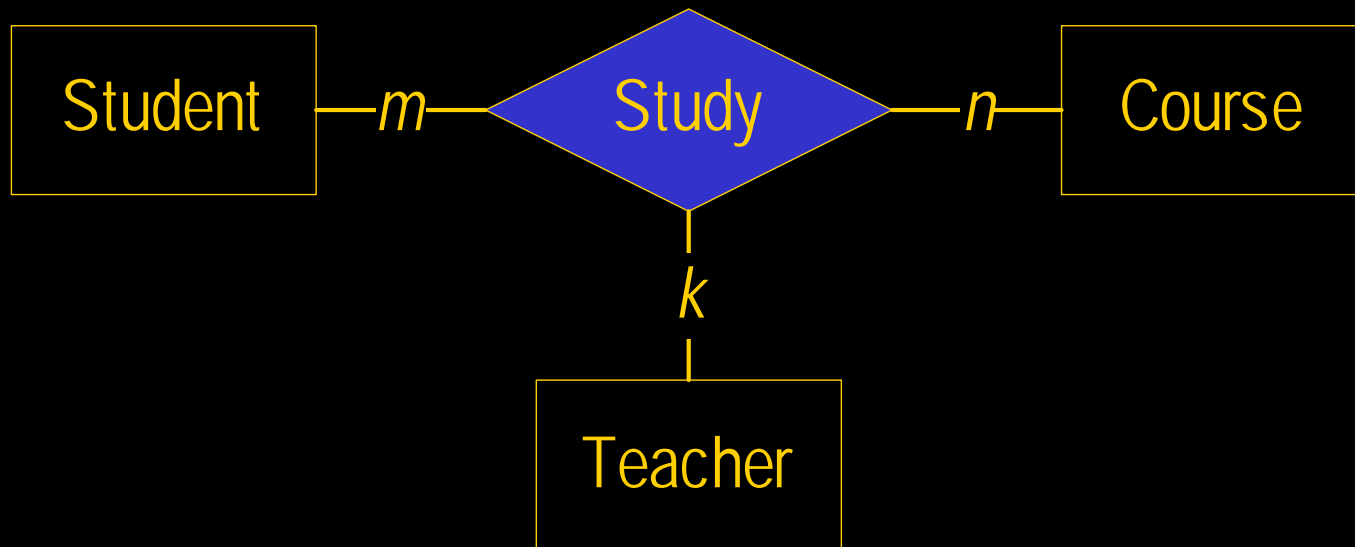


可能會有兩種情況組合

- **第一種**：學生 (Student)、課程 (Course)、教師 (Teacher) 三者形成一個「修課」(Study) 的三重關係 (Ternary Relationship)，
 - 一個學生會面對多種課程與多位教師、
 - 一個課程也會有多位學生修習或多位教師開同樣課程、
 - 一位教師也會開多種課程並面對多位學生，所以彼此之間的關係都是多對多。

資料庫規劃 (第一種情況)

- 繪出「個體-關係圖」(Entity-Relationship Diagram, ERD)



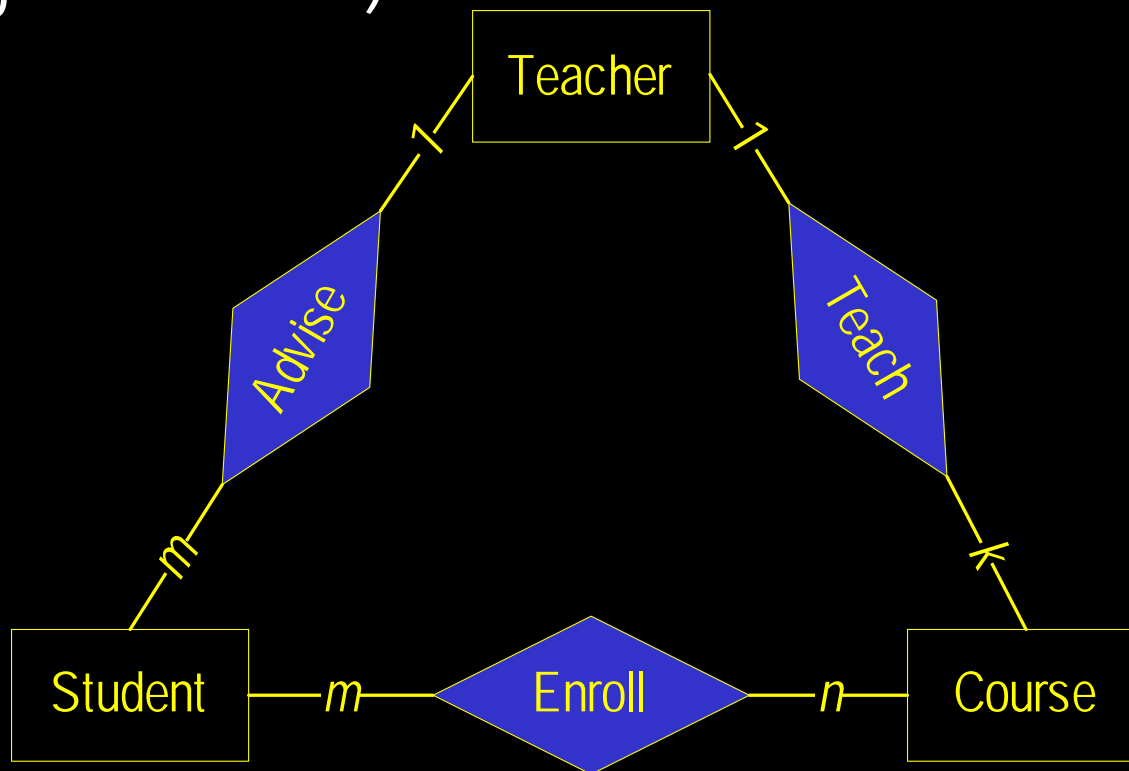


第二種可能情況組合

- **第二種**：學生 (Student)、課程 (Course)、教師 (Teacher) 兩兩之間都各有不同的關係
 - 學生與課程是「修課」(Enroll) 的多對多關係
 - 教師與學生是「指導」(Advise) 的一對多關係、
 - 教師與課程是「教授」(Teach) 的一對多關係。

資料庫規劃 (第二種情況)

- 繪出「個體-關係圖」(Entity-Relationship Diagram, ERD)



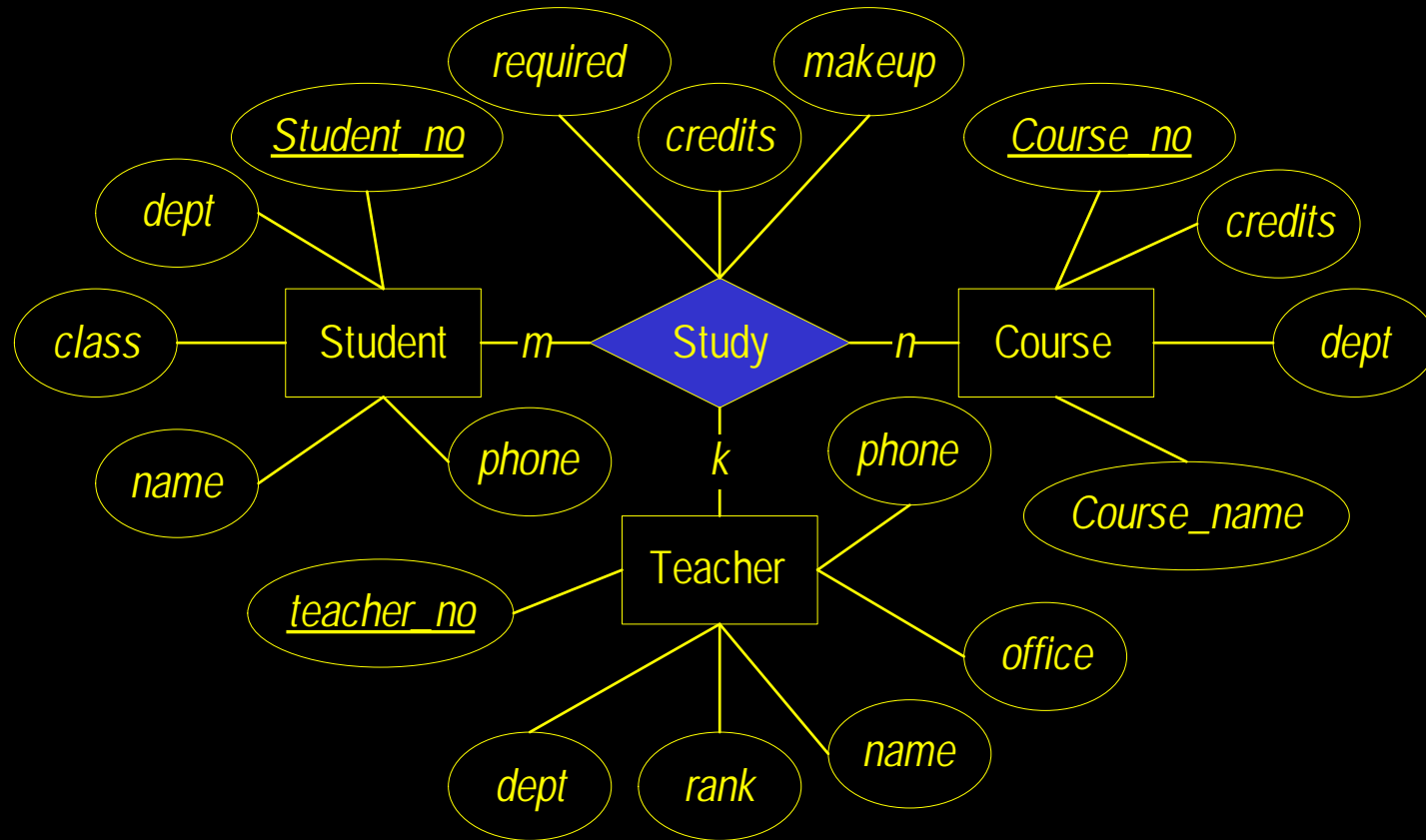


資料庫規劃 (續)

- 檢討「個體-關係圖」中每個「個體類型」所需的特性。例如：「學生」這個個體類型需要 Student_No、Dept、Class、Name、Phone 等屬性
- 檢討「個體-關係圖」中每個「關係類型」所需的屬性。
- 在「實體-關係圖」上補上橢圓形代表各個屬性。
- 針對每一個個體關係圖中的個體產生一個關聯表。

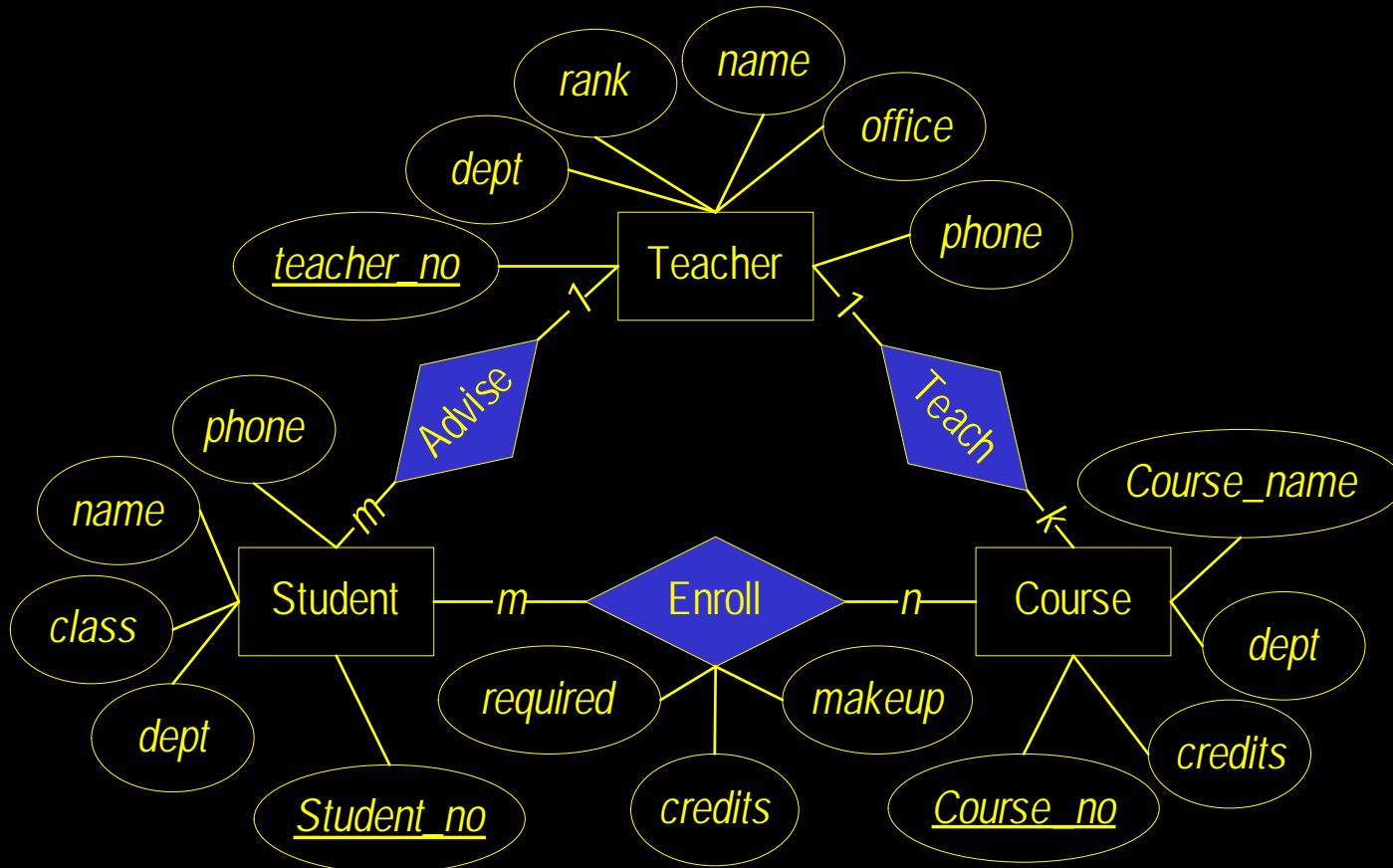
資料庫規劃 (第一種情況)


- 繪出完整的「個體-關係圖」ERD)



資料庫規劃 (第二種情況)

繪出「個體-關係圖」ERD)





個體表格與關係表 (第一種情況)

- Student(Student_No, Dept, Class, Name, Phone)
- Course(Course_No, Dept, Course_Name, Credits)
- Teacher(Teacher_No, Dept, Rank, Name, Office, Phone)
- Study(Student_No, Course_No, Teacher_No, Makeup, Required, Credits)
- 箭號表示外來鍵參考
- 注意: Course.Credits 表示課程本身的學分數, Study.Credits 表示修完課程後所獲得的學分數 (e.g., 研究生修大學部課程 (3 學分) 只能得到 0 學分)

個體表格與關係表 (第二種情況)

- Student(Student_No, Dept, Class, Name, Phone)
- Course(Course_No, Dept, Course_Name, Credits)
- Teacher(Teacher_No, Dept, Rank, Name, Office, Phone)
- Enroll(Student_no, Course_no, Makeup, Required, Credits)
- 不需產生 Advise 關聯表，只要將 Student(student_no, dept, class, name, phone) 變成 Student(student_no, dept, class, name, phone, teacher_no) 即可
- 同理也不需產生 Teach 關聯表，而是將 Course(Course_no, Dept, Course_name, Credits) 變成 Course(Course_no, Dept, Course_name, Credits, teacher_no) 即可



檢討並訂定表格的所有屬性

- 以白話文字說明該欄位的意義。
- 該欄位的類型 (Type) : Integer、Char、Date，或 image 等。
- 該欄位所佔的位元長度 (Length)。
- 該欄位的內容是否要強制唯一 (Unique)。如果是唯一的話，表示該欄位為候選鍵 (Candidate Key)，所以要確定該欄位是否要選為主鍵 (或主鍵的一部份)。

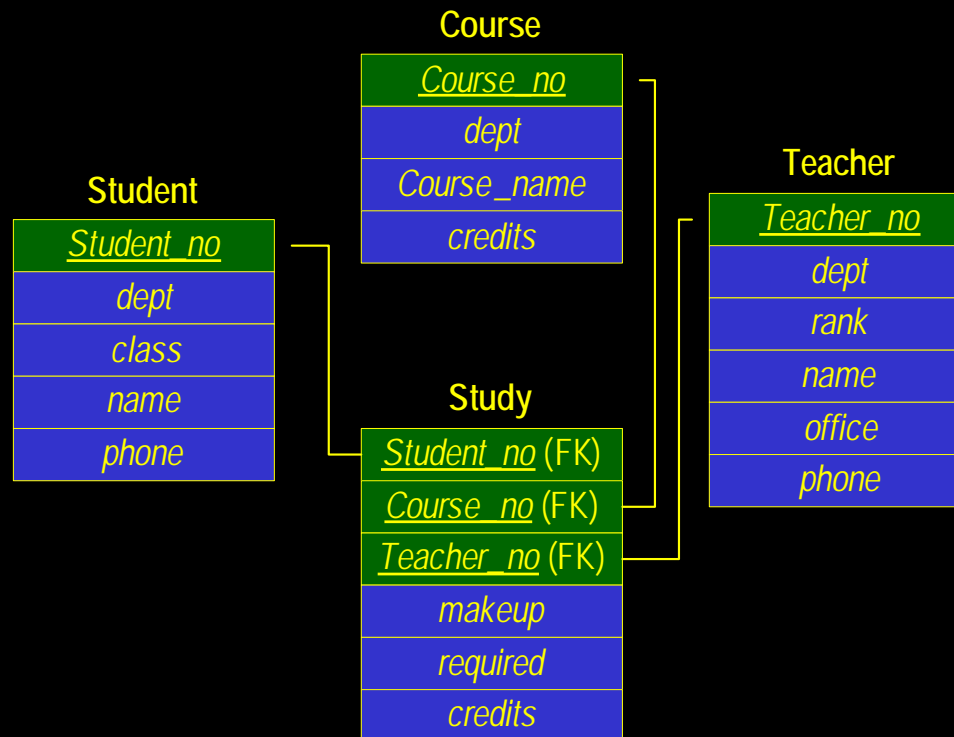


檢討並訂定表格的所有屬性

- 欄位是否有預設值 (Default Value)：例如，日期的預設值通常都是輸入資料的當日。
- 欄位是否有範圍限制 (Range) 或只能出現那些值：例如，員工年齡 (age) 的範圍只能從 18 到 65 歲之間。顏色只能有 White, Black, Blue, Red, Yellow, Purple 幾種。
- 欄位內容值之格式規範。例如，日期的格式是 mm/dd/yy；電話的格式是 (____) ____-____ 還是 ____-____？

資料庫規劃 (第一種情況)

- 繪出整體資料庫的外來鍵參考圖 (如下圖所示)
- 第二種情況的外來鍵參考圖, 請同學上臺繪出



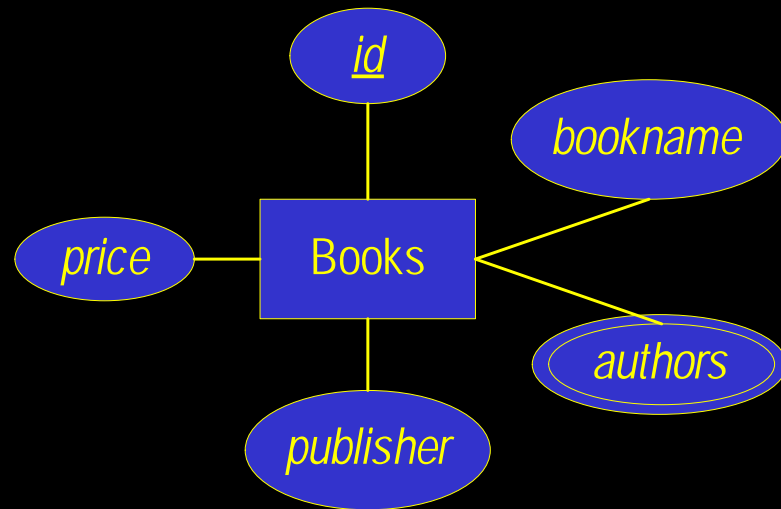


資料庫規劃 (續)

- 規劃各個功能的使用者介面 (User Interface) 與系統架構圖，讓使用者可以知道其系統的外觀為何，並了解整體系統架構。這個動作也稱為「快速原型法」(Rapid Prototyping)
- 針對各個系統的功能，說明要完成該功能的資料庫運算為何。這些運算將來可以轉成「預儲程序」(Stored Procedure) 或「觸發程序」(Trigger)

多重值屬性的處理

- 如果存在某個個體的特性值為多重值的話，就必須另外處理：將原個體的主鍵與該特性組成另一個獨立的關聯表。
- 如下圖所產生的關聯表為
 - Books (id, bookname, price, publisher)
 - Authors (id, author)



多重值屬性的處理 (續)

<u>id</u>	bookname	price	publisher	author
1	Java Programming	500	華泰	陳教授、林博士
...



<u>id</u>	bookname	price	publisher
1	Java Programming	500	華泰
...

<u>id</u>	<u>author</u>
1	陳教授
1	林博士



設計個體關係圖時常犯的錯誤

- 使用「個體-關係圖」時，常會發生錯誤的情況，
- 這些錯誤會影響後續的應用程式開發與查詢
- 設計的過程中應當仔細檢驗，並且盡量避免
- 常見的錯誤可以分成兩種， [T. Connolly, C. Begg, and A. Strachan (1999)]
 - 「扇形陷阱」 (Fan Trap) ，
 - 「斷層陷阱」 (Chasm Trap) 。

扇形陷阱 (Fan Trap)

- 以違規的駕駛人參加道路交通安全講習為例，
 - 數個「駕駛人」 (Driver) 接受同一個「講師」 (Lecturer) 輔導 (Guide)
 - 一位講師 (Lecturer) 負責 (Give) 數場講習 (Lecture)



問題討論

- 8924311 與 8924312 兩位駕駛人均接受 84310 講師的輔導。
- 84310 講師也負責了 4840 與 4780 這兩場講習的演講。
- 問題來了：如果要知道 8924311 與 8924312 兩位駕駛人所參加的講習是哪一場？是 4840 還是 4780？系統根本無法回答

Driver			Lecturer			Lecture	
<u>driver_no</u>	<u>guide_id</u>		<u>lecturer_id</u>	...		<u>lecturer_id</u>	<u>lecture_no</u>
8924311	84310	↔	84310	...	↔	84310	4840
8924312	84310	↔	86210	...	↔	84310	4780
8924315	85430	↔	85430	...	↔	85430	4968



原因與解決方案

- 原因：

- 不該把「講師」(Lecturer) 當作「駕駛人」(Driver) 與「講習場次」(Lecture) 的橋樑，因為 ERD 中的對應關係為 $n-1-1-m$

- 解決方案：

- 應該是將「講習場次」(Lecture) 拿來作為「駕駛人」(Driver) 與「講師」(Lecturer) 之間的橋樑，將 ERD 中的對應關係改成 $n-1-m-1$

修正結果

- 數個駕駛人 (Driver) 參加 (Attend) 同一場講習 (Lecture) ,
- 一位講師 (Lecturer) 負責 (Give) 數場講習 (Lecture)

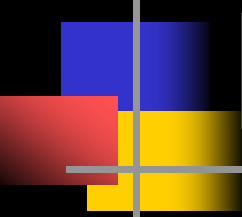


$n-1-m-1$

問題解決了

- 「駕駛人」 (Driver) 與「講師」 (Lecturer) 的關係透過「講習場次」 (Lecture) 仍然是多對一
- 駕駛人參加哪一場講習的資訊已經很清楚地存在表格中了

Driver			Lecture			Lecturer	
<u>driver_no</u>	<u>lecture_no</u>		<u>lecture_no</u>	<u>lecturer_id</u>		<u>lecturer_id</u>	...
8924311	4840	← →	4840	84310	← →	84310	...
8924312	4780	← →	4780	84310	← →	86210	...
8924315	4968	← →	4968	85430	← →	85430	...



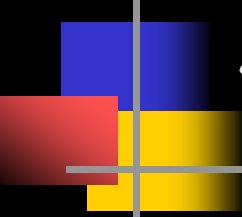
斷層陷阱 (Chasm Trap)

- 斷層陷阱 (Chasm Trap):
 - 有兩個個體類型 E_A 與 E_B ,
 - 在「個體-關係圖」中可以找到一條路徑將它們聯繫起來，
 - 但是卻存在某個屬於 E_A 中的個體無法聯繫到屬於 E_B 的任何一個個體時，
 - 那麼便是產生了「斷層陷阱」。

含 Chasm Trap 的 ERD

- 例如: 一個「科系」(Department) 中含有 (Contain) 數位「老師」(Teacher)，而一位「老師」(Teacher) 則掌管 (Manage) 許多的「財產設備」(Property)





何時會出現問題？

- 如果一個系上的任何財產設備都一定會由某位老師掌管的話，那麼這樣設計是沒有問題的，因為我們只要找出老師就可以找到所有的設備。
- 但是如果系上有某些設備並不是由老師來掌管時，那麼就會出現問題，

看看例子就知道

- '書櫃' 屬於資管系 (MIS) 的財產，但沒有老師掌管，所以用 '—' 代表掌管老師為虛值。
- 問題來了：當我們透過老師去統計系上的財產設備有哪些時，將會遺漏掉 '書櫃' 這樣財產！

Property			Teacher			Department	
<u>pname</u>	<u>manager_id</u>		<u>teacher_id</u>	<u>dept_name</u>		<u>dept_name</u>	...
投影機	84310	←→	84310	MIS	←→	MIS	...
電腦	85430	←→	85430	MIS	←→	CS	...
書櫃	—		86210	CS	←→	EE	...

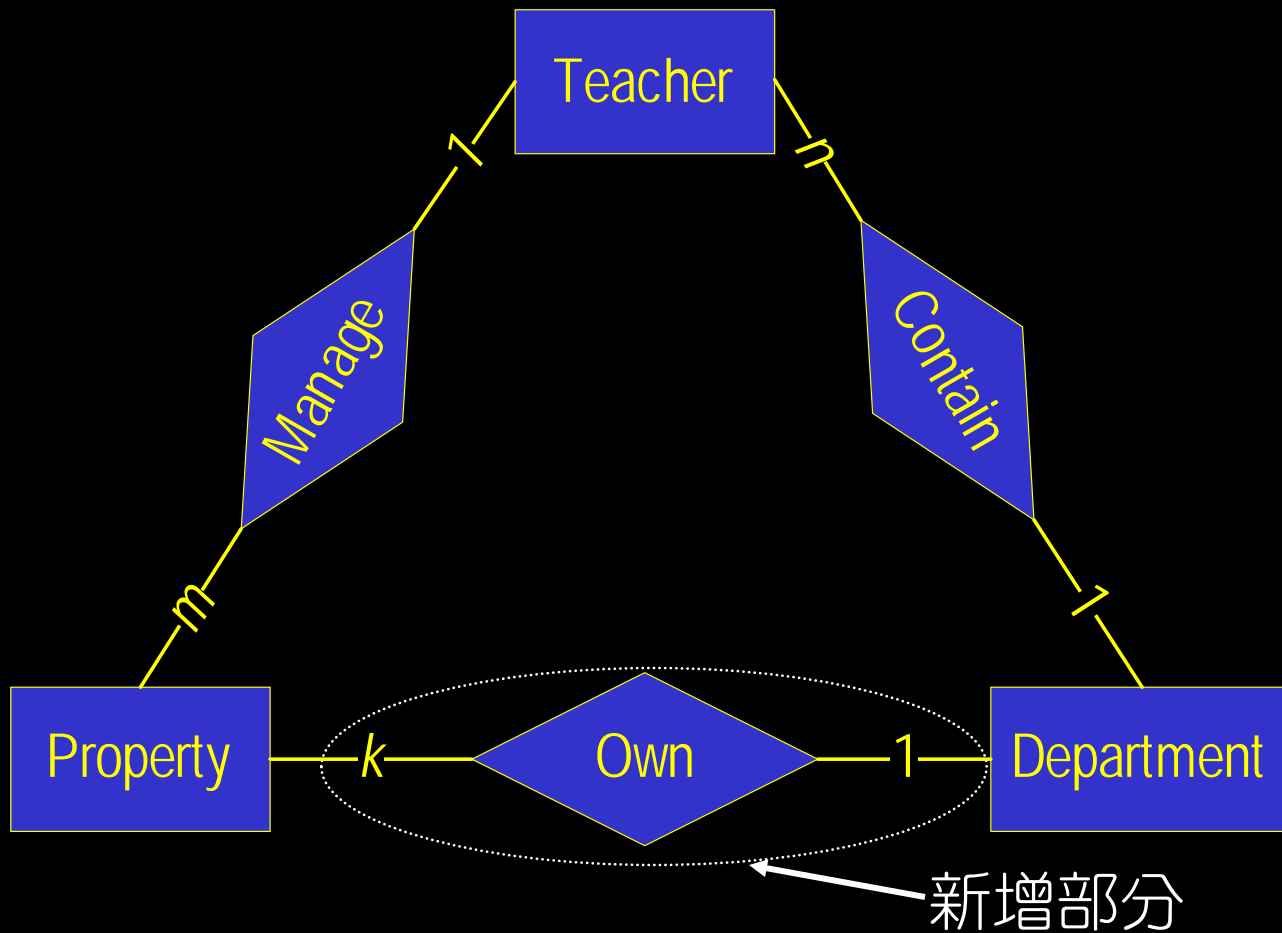


問題出在哪裡？

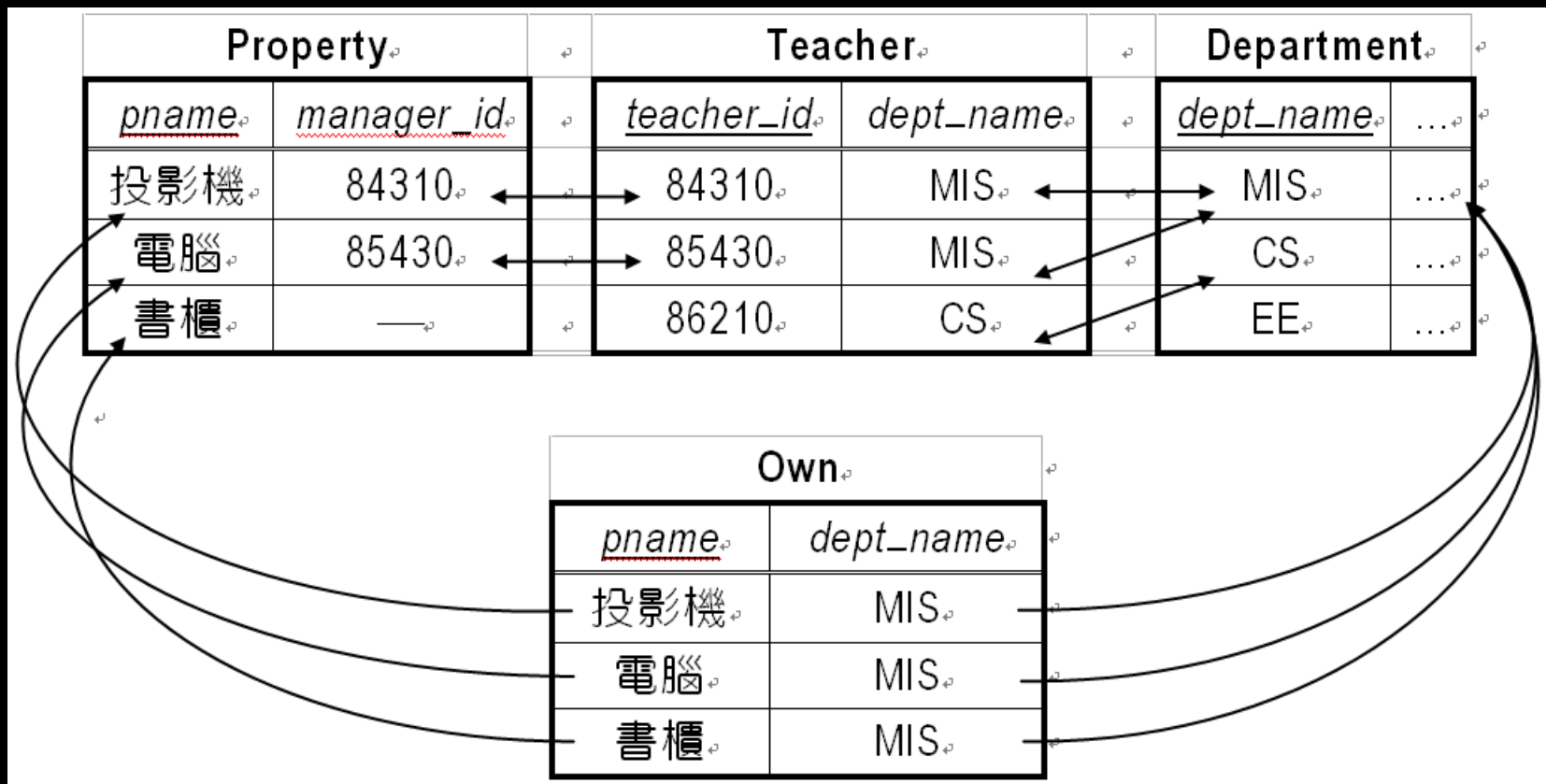
■ 原因：

- 在當有系上的設備不受「老師」(Teacher) 掌管時，我們要另外建立「科系」(Department) 與「財產設備」(Property) 之間的關係。
- 當系上存在一些沒有老師掌管的設備時，就必須為這些設備註明所屬的系別。
- 因此，我們在「科系」(Department) 與「財產設備」(Property) 之間必須加一個「擁有」(Own) 的關係。
- 如下頁所示...

修正後結果



修正後的表格內容



其實還可以再簡化...

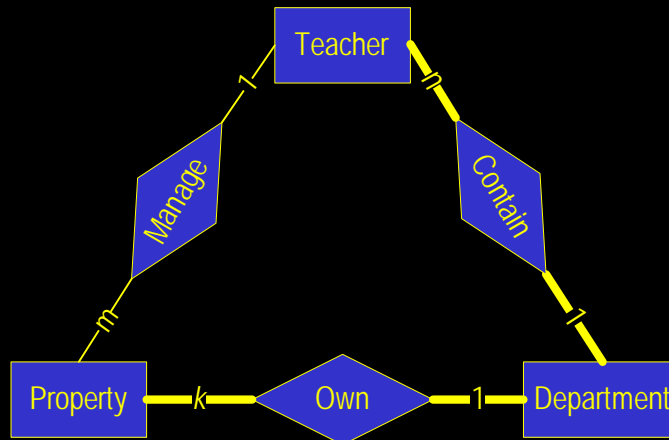
- 由於 **Own** 這個關係是一對多的關係，所以我們也可以將上面的關聯表簡化成在 **Property** 表格中新增一個欄位 *Dept_name* 用來存放該設備所屬的系別即可，當然，**簡化後 Own** 這個表格就可以省略了

Property				Teacher			Department	
<u>pname</u>	<u>manager_id</u>	<u>dept_name</u>		<u>teacher_id</u>	<u>dept_name</u>		<u>dept_name</u>	...
投影機	84310	MIS		84310	MIS		MIS	...
電腦	85430	MIS		85430	MIS		CS	...
書櫃	—	MIS		86210	CS		EE	...

加了這欄就可以省略 Own 表格

後記

- 修正後的 E-R Diagram 中好像又產生了前面所提到的「扇形陷阱」(Fan Trap)?
- 這並不會產生問題，因為
 - 每次要找老師 (Teacher) 所掌管的財產 (Property) 時，**不會走粗線的較遠路徑**，
 - 而是**直接走 [Teacher]—<Manage>—[Property]** 的最短路徑來統計。





其他資料庫規劃模式與工具

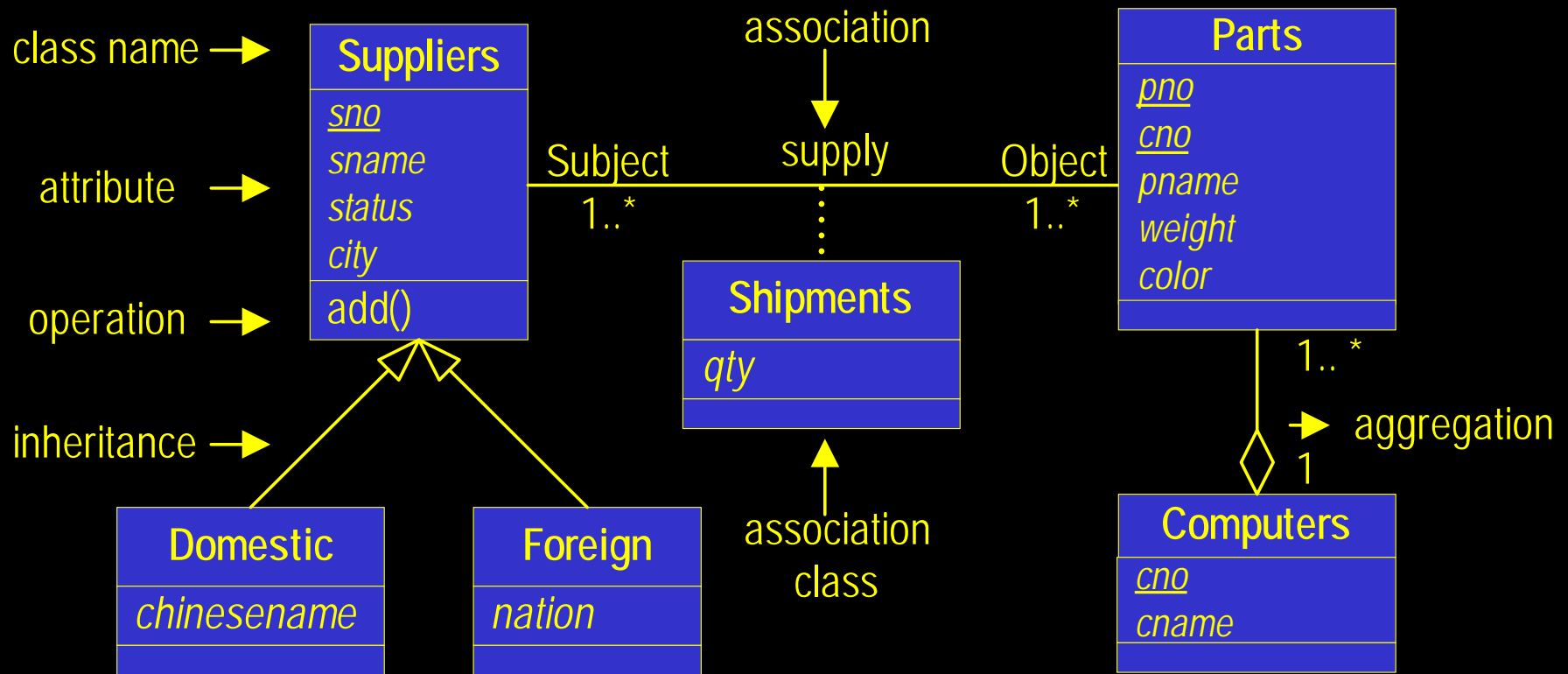
- 除了「個體-關係圖」以外，還有其他結構嚴謹的模式，提供更為廣泛的規劃元素：
 - 「統一塑模語言」(UML, Unified Modeling Language) 的「類別圖」(Class Diagram) <http://www.uml.org>
 - 「物件角色模式」(ORM, Object Role Modeling) <http://www.orm.net>



統一塑模語言 (UML)

- 由 G. Booch、J. Rumbaugh 與 I. Jacobson 整合，廣為業界採用，
- 其中「類別圖」可看成：將「個體-關係圖」物件化的延伸，
- 不但可用來做資料庫的規劃工具，同時也可以用來產生相對應的物件導向式應用程式架構，
- 通常與「電腦輔助軟體工程」(Computer Aided Software Engineering, CASE) 一併整合在一個 CASE 軟體工具 (CASE Tool) 裡。
- 市面上以 UML 為基礎的 CASE 軟體工具有：Rational Rose、ERWin、PowerDesigner 等。

UML 類別圖 (Class Diagram) 範例

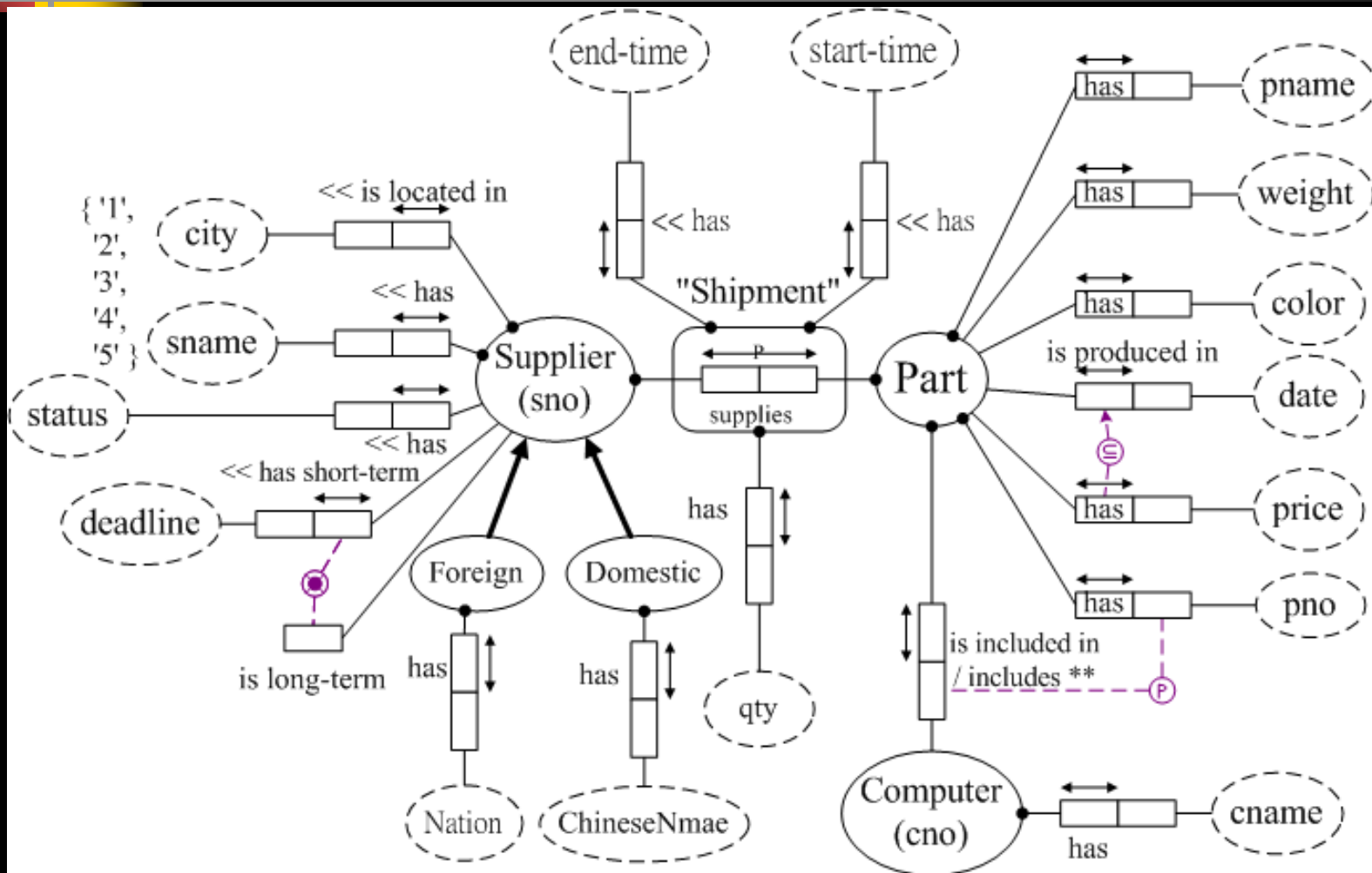


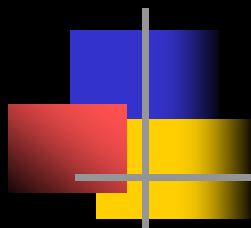


物件角色模式 (ORM)

- 由 Terry Halpin 所提出
- 受到微軟支持：最新 VISIO 企業版已內建 ORM 規劃工具
- 強調從基本的事實 (Facts) 角度出發來規劃資料庫，也稱為「事實導向式塑模」(Fact-Oriented Modeling)。
- 與「個體-關係圖」及「類別圖」最大的不同點在於：
 - 「個體-關係圖」以及「類別圖」在規劃過程常會發現必須將「屬性」變更成為「個體類型」或「物件類別」而導致整體規劃的穩定度不佳。
 - ORM 並不採用「屬性」(Attribute) 的概念，以便讓「事實」得以任意加入規劃圖中，不需做大幅的調整，提高整體規劃的穩定度。

物件角色模式 (ORM) 範例





本章結束
The End.