

# CH1

## 基本定義

**Database:**相關數據的集合

**Data:**具有含義可以被記錄下來的事實

**Mini-world:**現實中的某部分數據被儲存在資料庫中

**Database Management System:**方便創建和維護數據庫的系統

**Database System:**DBMS和資料庫組成的整體系統，有時包含應用

## DBMS功能

- 1.定義特定數據庫的類型、結構和約束
- 2.構建或加載初始數據庫內容至次級存儲媒體
- 3操作數據庫:檢索(查數據) 修改 通過網頁應用程式訪問數據庫
- 4.允許多個使用者和應用程式同時處理和共享數據，同時保持所有數據的一致性

## Database Approach的主要特點

### 1.資料庫系統的自描述特性

DBMS目錄存儲特定資料庫的描述信息(數據結構、類型和約束)。

這個描述被稱為meta-data(詮釋資料)。

允許DBMS軟體與不同的資料庫應用程式一起使用。

### 2.程式與資料之間的隔離

如果要加新資料不用去改程式

### 3.資料抽象

一種資料模型用於隱藏儲存細節，並向使用者呈現資料庫的概念圖。

不包括資料儲存方式或操作實作的詳細資訊。

程式將參照資料模型的概念，而不是資料儲存的細節。

允許程式與資料獨立性和程式操作獨立性。

### 4.支援多個資料視角

每位使用者可能會看到資料庫的不同視圖，該視圖僅描述該使用者感興趣的資料。

### 5.資料共享和多使用者交易處理

允許多個使用者同時存取操作資料庫，且保證資料操作不會互相干擾。具備恢復機制，確保每個已完成的事務的效果被永久地保存在資料庫中

# CH2

## Data Model

描述資料庫的結構，操作這些結構的方法，以及資料庫應遵守的特定約束條件。

結構指的是資料類型、關係，以及適用於這些資料的約束條件。

提供實現資料抽象的方法。

EX:ER圖

## **categories of data model**

**Conceptual Data Models(概念)**

提供了接近許多用戶對數據感知方式的概念

### **Physical Data Models(物理)**

描述了數據在計算機中的存儲細節

### **Implementation Data Models(實現)**

介於概念性和物理模型之間，用於許多商業數據庫管理系統的實現

### **Self-Describing Data Models(自描述)**

數據的描述與數據值結合在一起

## **Database Schema(intension)**

對資料庫的結構、資料類型和資料庫約束的詳細說明，存在DBMS

### **STUDENT**

Name	Student_number	Class	Major

## **Database State(extension)**

指在特定時間點資料庫內實際存儲的資料。這包括了資料庫中所有資料的集合。

### **COURSE**

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS

## **Three-Schema Architecture**

一種DBMS中使用的架構，資料庫被描述為三個層次

### **1.內部模式(Internal Schema)**

用於描述物理存儲結構和訪問路徑(例如索引)的層次

通常使用物理數據模型。

內部模式定義了資料庫的實際存儲方式，並描述了如何在硬盤上組織數據。

### **2.概念模式(Conceptual Schema)**

用於描述整個資料庫的結構和約束，針對用戶群體。

使用表示(或實現)數據模型，通常基於高層次數據模型中的概念模式設計。

概念模式定義了整個資料庫的邏輯結構，獨立於具體的應用程序，這也是所有外部模式的抽象。

### **3.外部模式(External Schemas)(外部層次)：**

用於描述不同用戶視圖的層次。

每個外部模式描述了特定用戶組關心的資料庫部分，並隱藏了其餘的部分。

通常使用和概念模式相同的數據模型(即表示數據模型)。

## **Logical Data Independence:**

不影響外部模式和相關應用程式的情況下，能夠修改概念模式的能力

## **Physical Data Independence**

指的是在不影響概念模式(Conceptual Schema)的情況下，能夠修改內部模式(Internal Schema)的能力。

## **CH3**

## **Types of Attributes**

### **Simple**

has a single atomic value ex:SSN or Sex

### **Composite**

composed of several components ex:Address(Street, City, State, ZipCode, Country)

## **Multi-valued**

multiple values ex: {Color} or {PreviousDegrees}

## **Stored versus derived attributes**

可以被推導而來的 ex:有幾個員工

## **名稱**

**Entity Set:**所有的值

**Entity Types:**定義

**Relationship type:**定義它的限制條件

**relationship set:**目前有哪些**relationship**

min

# **ER Model Concepts**

## **1.Entities (方框)**

是ER模型的基本概念是指具體的事物或物體, 這些事物或物體在資料庫中被表示

EX:EMPLOYEE實體可以是John Smith這個員工

## **2.Attributes (橢圓)**

用來描述一個實體的特性

## **3.Key Attribute (底線)**

對於實體類型的屬性, 如果每個實體都必須具有唯一值, 則被稱為實體類型的Key Attribute

一個Key Attribute可以是複合的

一個實體類型可以有多個Key Attribute

每個Key Attribute都會被下劃線標出

"key attribute" 是指實體中用来唯一標示該實體的屬性

## **4.Relationships (菱形)**

兩個或更多實體之間的關係

而關係也可以擁有屬性

像是工作關係的屬性為工時

在1:N情況, 如果要把屬性移到entity, 會選擇N的那邊

ER圖會有對應關係如1:N

連接的線分兩類型

完全約束(兩條線)一個entity類型的全部都必須參與關係 例如:員工跟部門的工作關係

部分約束(一條線)一個entity類型的部分參與關係 例如:員工跟部門的管理關係

## **5.Weak Entity Types (雙重方框)(Ex:訂單項目, 主題為訂單, 標示符為訂單號)**

沒有獨立的主鍵的實體且需要依賴其他實體的關係(及partial key)來確定其唯一性

弱實體識別:

1.**partial key**(虛底線):這是弱實體內部的某個屬性或屬性組合

2.與它相關的特定擁有者實體, 這是透過識別關係確定的

**Figure 3.14**  
Summary of the notation for ER diagrams.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of $E_2$ in $R$
	Cardinality Ratio 1: N for $E_1:E_2$ in $R$
	Structural Constraint (min, max) on Participation of $E$ in $R$

## UML class diagrams

將類別(類似於Entities)表示為具有三個部分的大圓角框：

頂端部分包括Entities(類別)名稱

第二部分包括Attributes

第三部分包括類別操作(在基本ER模型中不存在的操作)

**Relationships**用線條連接類別表示

UML術語與ER術語不同

用於數據庫設計和面向對象軟體設計

UML還有許多其他類型的軟體設計圖表。

# CH4

## Enhanced Entity-Relationship

### 1. Superclasses(父類別)

可以為任意數量Subclasses的Superclasses

EX:員工

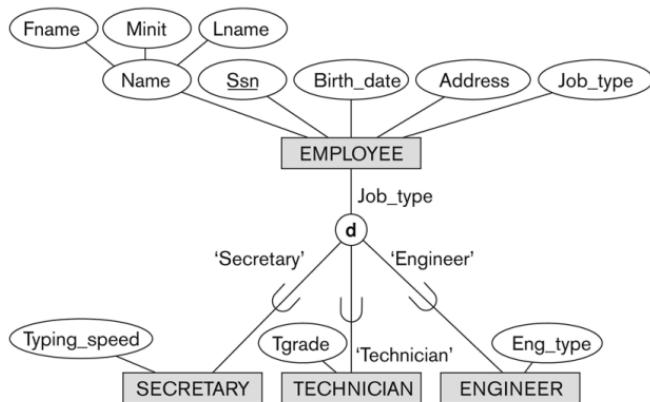
### 2. Subclasses(子類別)

不屬於任何Superclasses無法單獨存在

會繼承所有Attributes和Relationships

但仍可以保有自己的Attributes

EX:工程師 經理



## Specialization

定義Superclasses的一組Subclasses的過程

基於Superclass中實體的某些特徵去定義

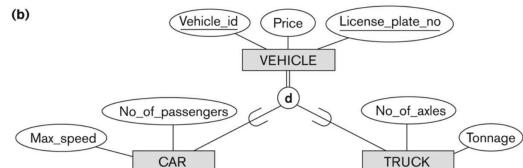
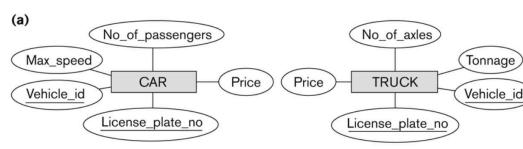
可以有多組特殊化

EX:將員工用工作類型分為不同類別

## Generalization

將具有共同特徵的多個類別泛化為一個Superclasses

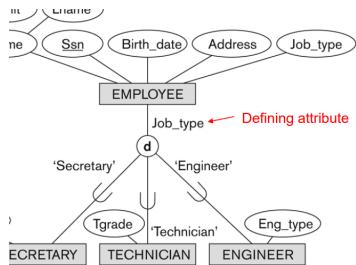
EX:轎車跟卡車都有相同屬性引擎編號、速度等等，因此定義一個車為他們的Superclasses



## Types of Specialization

Predicate-defined:用範圍來分類 ex:Age

Attribute-defined:用名子去分，像底下的secretary



**User-defined:defined by the user:>**

圖中的紅字d因為我們用Attribute去分類所以會寫defining attribute

## Disjointness Constraint:

最多為一種子類別 標記為d

如果不是disjoint則為overlapping故可以為多個子類別 標記為o

## Completeness (Exhaustiveness) Constraint:

Total:所有的entity必定為一種子類別(類似完全約束)

同樣由兩條線代表

partial:只有部分為子類別

同樣一條線表示(你可能完全不belong子類別)

## Lattices & Shared Subclasses

**Hierarchy:**子類別只有一個超類別(稱為單一繼承), 一個樹狀結構

**Lattices :**子類別可以同時是多個超類別的子類別, 這稱為多重繼承, 又稱為share subclass share subclasses:

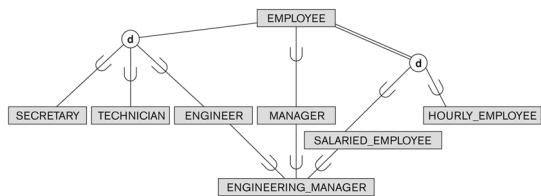


Figure 4.6  
A specialization lattice with shared subclass ENGINEERING\_MANAGER.

## Categories (UNION TYPES)

使用多個超類別來建模單一的超類別/子類別關係, 可以選擇為任意種超類別

超類別可能有不同的主鍵屬性

表示符號為U

EX:車輛的擁有者可以是個人、銀行或公司

與Shared Subclasses的區別在於:

**1. Shared Subclasses**是其超類別交集的子集合

UNION TYPES是其超類別聯集的子集合

EX:助教必須同時是學生又是員工

車輛的擁有者只需要是人或銀行或公司

**2. Shared Subclasses**的成員必須存在於其所有的超類別中

# CH5

## Relational Model

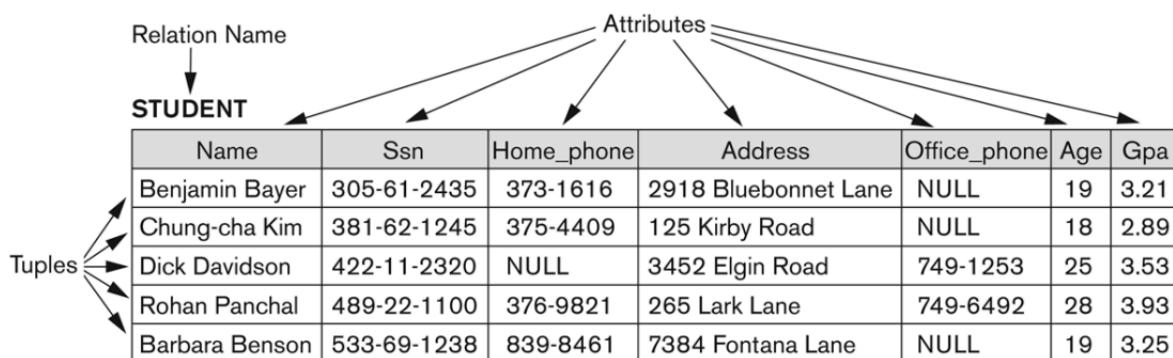
一個關聯通常包含一組row

tuple的順序可以互換。tuple的值不能再有tuple ex:<a1,a2,<a3,a4>>

tuple的值可以有NULL。如果該值不能是NULL。可以再限制裡定義

第一排的column稱為attribute

column的順序不可以互換



### Key of a Relation

每一行都具有一個數據項的值，這些項目唯一識別了該行在表格中的位置。called key 例:Ssn

### Schema (or description) of a Relation

用R(A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>)表示

1.R是關聯的名稱

2.關聯的屬性是A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>

EX:CUSTOMER(Cust-id, Cust-name, Address, Phone#)

每個屬性都有一個domain。

EX:Cust-id的域是6位數字

### r(R)

r(R) :dom (A<sub>1</sub>) X dom (A<sub>2</sub>) X ...X dom(A<sub>n</sub>)

r(R) = {t<sub>1</sub>, t<sub>2</sub>, ..., t<sub>m</sub>}, 其中每個t<sub>i</sub>是一個n-tuple

t<sub>i</sub> = <v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>n</sub>>, 其中每個v<sub>j</sub>屬於dom(A<sub>j</sub>)

dom(A<sub>1</sub>) = {0,1} dom(A<sub>2</sub>) = {a,b,c}

則可能的值為{<0,a>, <0,b>, <0,c>, <1,a>, <1,b>, <1,c> }

### Tuple

一個有排序的值集合(使用"<"">"表示) 但不考慮他的排序(可互換)

每個值來自適當的domain 且不可以分割

用t[A<sub>i</sub>] 或 t.A<sub>i</sub>來引用tuple中的值

EX:<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">

上面EX也稱為4-tuple

null值表示未知、有值但不確定、不在範圍的值

### Domain

有一個邏輯定義

也能定義的數據類型或格式

EX: "USA\_phone\_numbers" 是在美國合法的10位數字電話號碼的集合

日期格式yyyy-mm-dd, 或者dd mm,yyyy等

Cardinality: domain中的總數量

## CONSTRAINTS

### 1.Inherent or Implicit Constraints(像是淺規則限制)

EX: 要 atomic attribute value(不可分割) no duplicate tuples(不能重複的值)

### 2.Application based or semantic constraints:額外的限制條件(低修學分)

### 3.Schema-based or Explicit Constraints(main)

- Key constraints(規定識別的值)
- Entity integrity constraints
- Referential integrity constraints(Not Null constraint)

| A special **null** value is used to represent values that are unknown or not available (value exist) or inapplicable (value undefined) in certain tuples.

#### ■ INSERT may violate any of the constraints:

- **Domain constraint:** ~~110590036 insert does 格式不符~~
  - if one of the attribute values provided for the new tuple is not of the specified attribute domain
- **Key constraint:**
  - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- **Referential integrity:**
  - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- **Entity integrity:**
  - if the primary key value is null in the new tuple

#### ■ DELETE may violate only referential integrity:

- If the primary key value of the tuple being deleted is referenced from other tuples in the database
  - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
    - **RESTRICT** option: reject the deletion
    - **CASCADE** option: ~~刪就跟着刪，改就跟着改~~
      - attempt to cascade the deletion by deleting tuples that reference the tuple being deleted (**ON CASCADE DELETE**)
      - propagate the new primary key value into the foreign keys of the referencing tuples (**ON CASCADE UPDATE**) (for update operation)
    - **SET NULL** option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each *foreign key constraint*

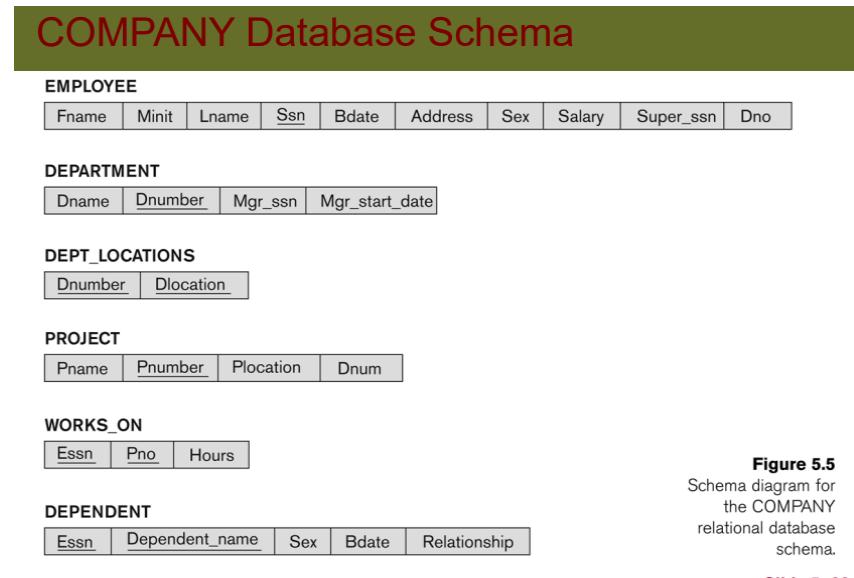
modify the  
referencing  
attributes that  
cause the  
violation

# Relational Database Schema

表示為  $S = \{R_1, R_2, \dots, R_n\}$ ,

$S$  是整個數據庫模式的名稱

$R_1, R_2, \dots, R_n$  是數據庫  $S$  內各個關聯模式的名稱



**Figure 5.5**  
Schema diagram for  
the COMPANY  
relational database  
schema.

Slide 5- 28

兩個底線是因為複合式主鍵

## Key constraints

**super key:**

任意兩個  $t_1[SK], t_2[SK]$  不會相等就滿足

superkey不一定包含 key

但 key 一定是 superkey

**key:**是最小值的 super key

CAR(State, Reg#, SerialNo, Make, Model, Year)

CAR has two keys:

- Key1 = {State, Reg#}
- Key2 = {SerialNo}

Both are also superkeys of CAR

{SerialNo, Make} is a superkey but *not a key*.

如果 relation 有不只一個 key, 我們會選一個叫做 primary key, 其餘的 key 叫做 candidate keys  
primary key 通常會選擇最小的 ex: 上面那張圖的 key2

我們會將 primary key 在表格中畫底線

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05

## Entity Integrity

primary key 不可以為空值(複合式的則皆不行)

## Referential Integrity

包含到兩個表格, 一個關聯中的元組如果參照了另一個關聯,

則它必須參照該關聯中的**現有tuple**

參照關係中的**R1**具有參照**R2**主鍵屬性**PK**的屬性**FK(Foreign key)**

如果 R1 中的 tuple1 參照了 R2 中的 tuple2，則 t1[FK] = t2[PK]。

在關聯式數據庫模式中，可以將參照完整性約束表示為從 R1.FK 指向 R2(或 R2.PK)

被參照的不能被刪除，否則要連參照的一起刪除

**FK**的值只有兩種：

1.已存在於被參照關係 R2 中對應主鍵 PK(主鍵可以指向主鍵)

2.一個空值(null)但不能是在參考自己主鍵的時候

EX:一個員工沒有部門則參照公司的部門編號時為NULL

## Semantic Integrity Constraints

### CH9

#### 目標

1.保留所有資訊

2.盡可能維持約束

3.減少NULL的值

#### 步驟 1: 繪製實體類型

建立一個table包含全部的屬性

選擇關鍵屬性做主鍵

如果選擇的key是複合鍵，則簡單鍵的集合屬性將共同構成 R 的主鍵

#### EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

#### 步驟2: 繪製弱實體

建立一個table包含全部的屬性

將識別對方的主鍵和識別關係名稱作為弱實體的主鍵

下圖的Essn和Dependent name共同組成這個弱實體的主鍵並記錄他的屬性

#### (b) DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

#### 步驟3: 處理關係

1.外鍵方法：

1對1關係：

選擇其中一個關係(比如S), 在S中加入T的主鍵作為外鍵。最好選擇在R中具有總參與度的實體類型作為S的角色。

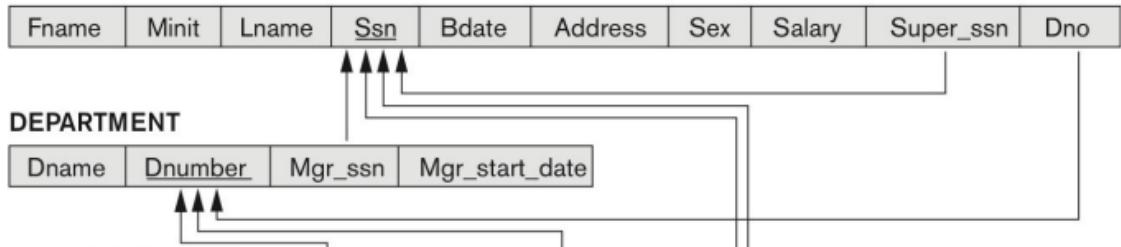
像是經營的關係因為每個部門都要經理所以將經理的ssn加入在部門中

1對多關係：

在多的關係中加入1那邊的主鍵作為外鍵在大多情況下

同理因為一個部門會有多個員工故在，多的那邊(員工)的屬性新增部門的編號

#### EMPLOYEE



#### 2.合併方法：

兩個實體類型和關係合併為一個單一的關係。

當兩者的參與度都是total時，這可能是合適的。

#### 3.交叉參照：

1對1：

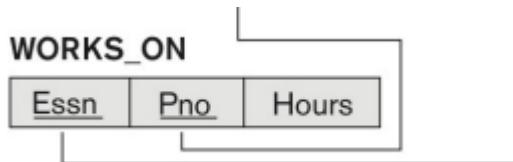
設立第三個關係R, 用於交叉參照代表這兩個實體類型的關係S和T的主鍵。

多對多關係(較常使用)：

設立第三個關係, 由參與的關係主鍵作為新的關係主鍵並記錄其屬性

因為有多個員工在不同專案的工作時數，為一個n對n關係

因此新建立一個表格將兩邊的主鍵合成作為新的主鍵

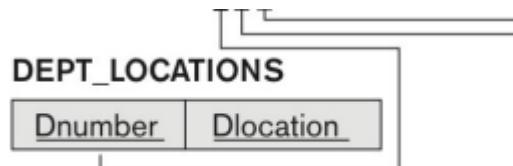


## 步驟4：處理多值屬性

創建新的表格，將自身的屬性以及對應的實體key結合成新的主鍵

部門的位置為一個複合屬性故新建立一個表格表示

並將他的屬性+原本的部門主鍵number作為新的主鍵使用



## 步驟5：處理N元關係類型

創建新的表格，將參與的全部關係類型的key結合成為新的主鍵，並記錄其屬性，和交叉參照  
類似但因為N個關係故主鍵變為N個合成

## 步驟6：處理一般化或特殊化的狀況

### 方法1：Multiple relations-Superclass and subclasses

適用於全部

當有一個超類別和多個子類別

將所有的超類別(涵蓋全部共有屬性)和子類別(分別添加各自屬性)都分別成為一個表格  
所有子類別都有超類別的key

(a) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType
-----	-------	-------	-------	-----------	---------	---------

SECRETARY

SSN	TypingSpeed
-----	-------------

TECHNICIAN

SSN	TGrade
-----	--------

ENGINEER

SSN	EngType
-----	---------

### 方法2：Multiple relations-Subclass relations only

必須所有在超類別的實體都是一種子類別(Total)的關係

只建立子類別的表格，然後會涵蓋所有從超類別繼承來的屬性及各自的屬性

(b) CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers
-----------	----------------	-------	----------	----------------

TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	Tonnage
-----------	----------------	-------	-----------	---------

### 方法3：Single relation with one type attribute

創建一個表格，表格包含全部的超類別和子類別屬性並用一個區分屬性作為分類子類別的依據，範例中為jobtype，這個方法適用在disjoint但是會容易產生多個NULL，在子類別有越多屬性時越不好

(c) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	EngType
-----	-------	-------	-------	-----------	---------	---------	-------------	--------	---------



### 方法4：Single relation with multiple type attributes

創建一個表格，表格包含全部的超類別和子類別屬性並用多個區分屬性作為分類子類別的依據，範例中為MFlag和PFlag，這個方法適用在overlapping但需要確保類型屬性的正確性

(d) PART

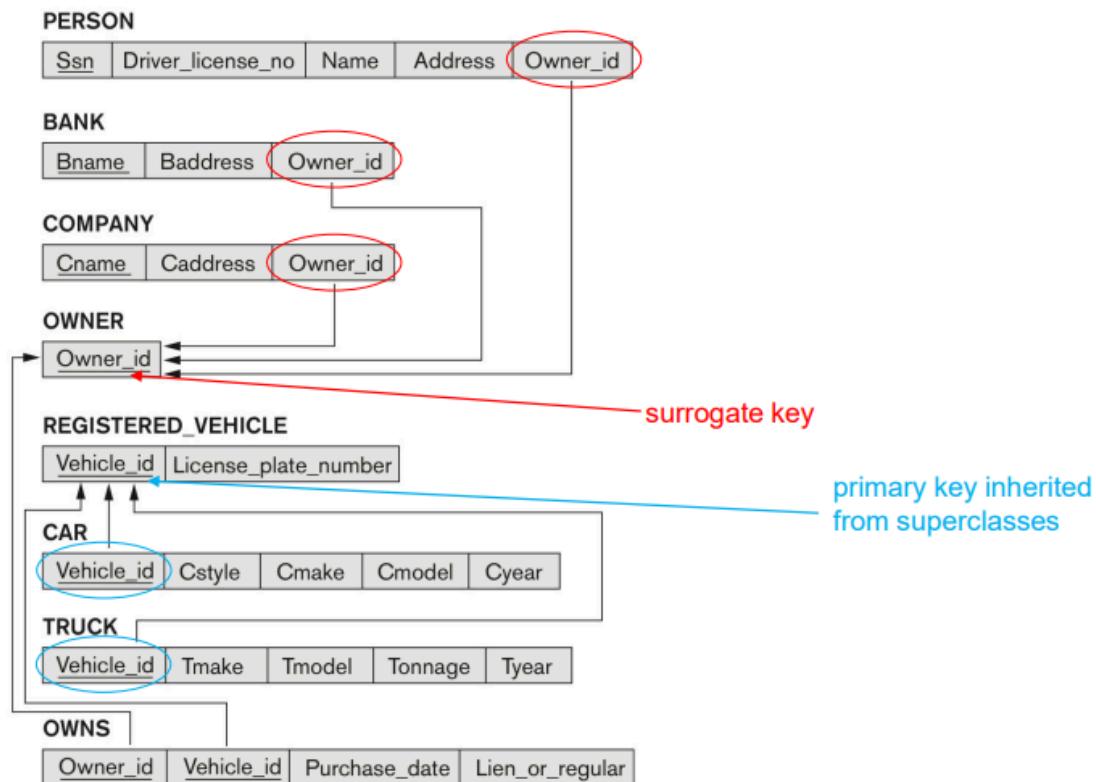
PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice
--------	-------------	-------	-----------	-----------------	---------	-------	--------------	-----------

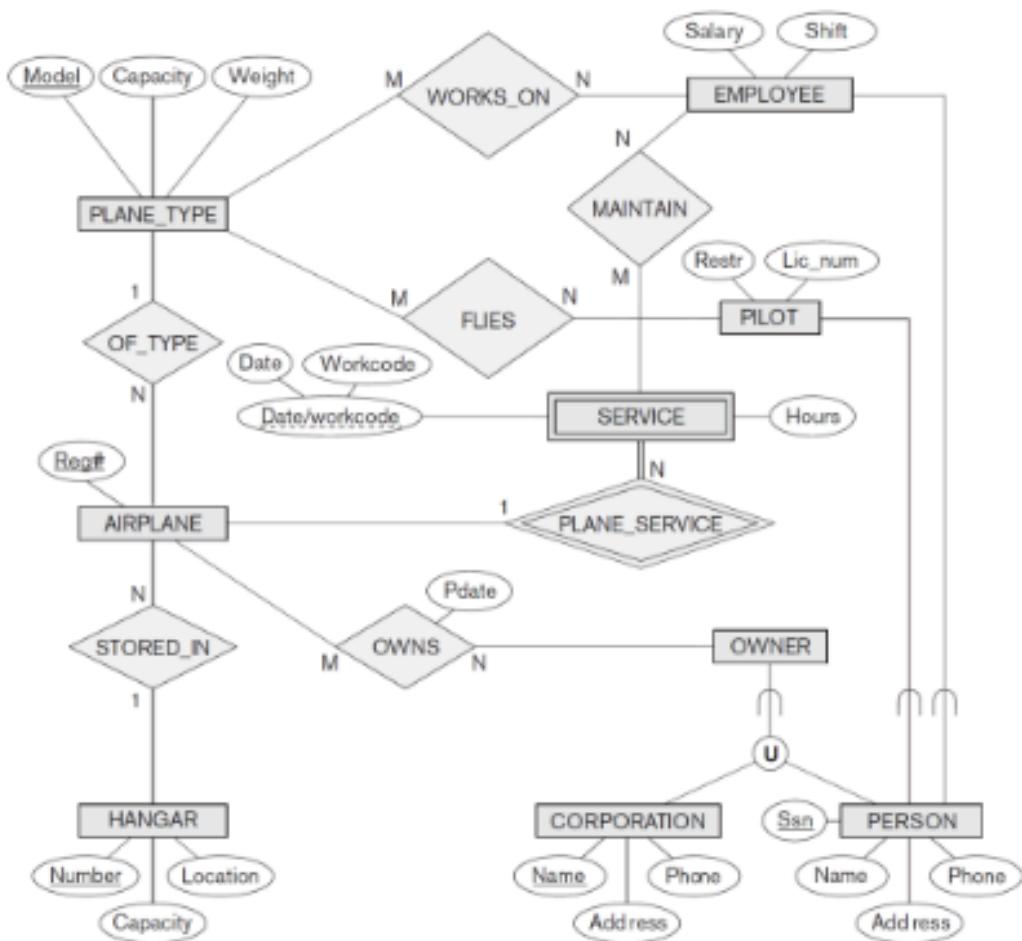
## Mapping of Shared Subclasses (Multiple Inheritance)

須都具有相同的主鍵屬性；否則，這個共享子類別將被建模為一個類別

### 步驟7: Mapping of Union Types (Categories)

通常會在創建對應於這個類別的關聯時，指定一個新的主鍵屬性，稱為代理鍵(Surrogate Key)，如果原本就有相同的主鍵則直接使用就可以





## CH7

### JOIN

#### 1. INNER JOIN (內部聯結) :

返回兩個表格中相符的行。只有當關聯欄位在兩個表格中都有匹配值時，這些行才會包含在結果中

#### 2. LEFT JOIN (左外部聯結) :

返回左表格中所有行，以及右表格中與左表格中的行相符的行。如果右表格中沒有匹配的行，則結果中右表格的列將包含 NULL 值。

#### 3. RIGHT JOIN (右外部聯結) :

返回右表格中所有行，以及左表格中與右表格中的行相符的行。如果左表格中沒有匹配的行，則結果中左表格的列將包含 NULL 值。

#### 4. FULL JOIN (全外部聯結) :

返回左表格和右表格中所有的行，如果沒有匹配的行，對應的欄位將包含 NULL 值。

## SQL中判斷是否為空值

用is NULL 或是is NOT NULL, 不能用等於因為NULL有不同類型值

```
SELECT      Fname, Lname  
FROM        EMPLOYEE  
WHERE       Super_ssn IS NULL;
```

## IN的使用

檢查某一欄位的值是否與指定的值列表中的任何一個值相匹配

```
SELECT student_name  
FROM students  
WHERE grade IN ('A', 'B', 'C');
```

## Nested Queries的使用

1. 條件本身需要根據其他表格的資料來確定
2. 如果nested query返回的結果只有一個屬性(attribute)和一個元組(tuple), 那麼使用 = 來替代 IN 作為比較運算符

```
SELECT product_name  
FROM products  
WHERE product_id IN (SELECT product_id FROM orders);
```

## Correlated Nested Queries 使用

內部查詢中的條件或過濾條件是由外部查詢的資料提供的

```
SELECT  
    customer_id,  
    customer_name,  
    (SELECT COUNT(*)  
     FROM orders  
     WHERE orders.customer_id = customers.customer_id) AS order_count  
  FROM customers;
```

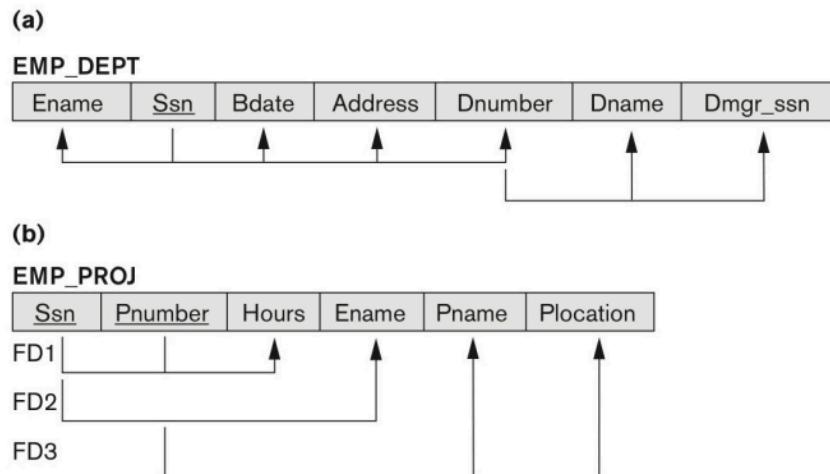
## EXIST 的使用

檢查是否返回至少一行。如果返回至少一行，  
EXISTS 的結果為 True；如果查詢沒有返回任何行，結果為 False。

第一正規化：沒有nest relation

第二正規化：左邊有沒有兩個欄位的，可不可拆？

**Figure 14.3**  
Two relation schemas  
suffering from update  
anomalies. (a)  
EMP\_DEPT and (b)  
EMP\_PROJ.



## Figure 14.11 Normalizing into 2NF and 3NF

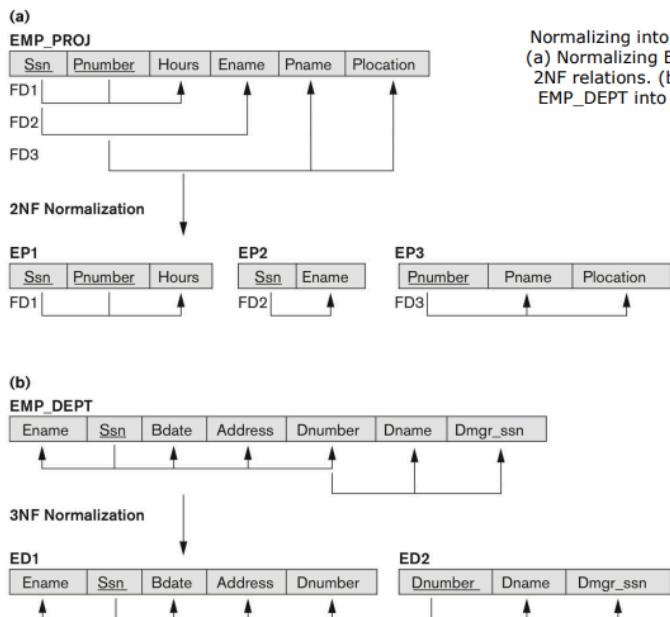
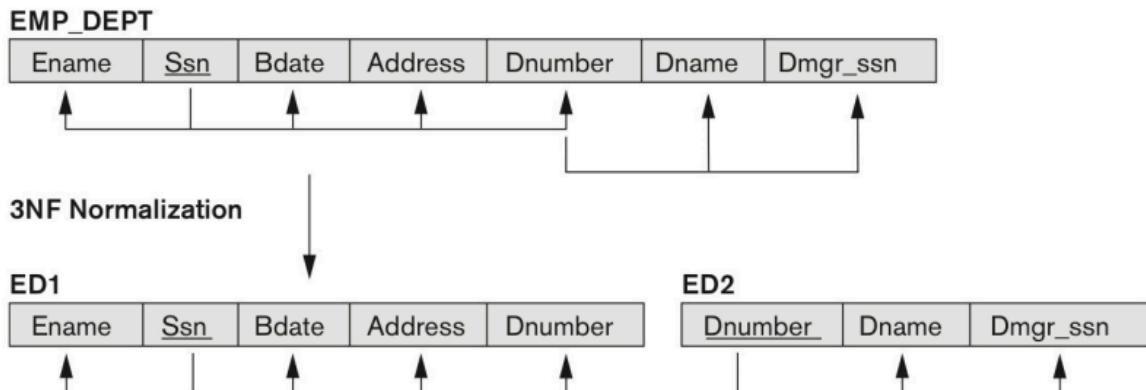


Figure 14.11  
Normalizing into 2NF and 3NF.  
(a) Normalizing EMP\_PROJ into 2NF relations. (b) Normalizing EMP\_DEPT into 3NF relations.

Copyright © 2017 Ramez Elmasri and Shamkant B. Navathe

Slide 14- 58

第三正規化：



我不知道怎麼

- A relation  $R(A, B, C, D)$  with its extension.
- Which FDs may exist in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

*May hold*

$B \rightarrow C; C \rightarrow B; \{A, B\} \rightarrow C; \{A, B\} \rightarrow D; \{C, D\} \rightarrow B$

*Do NOT hold (can be inferred from the relation state)*

$A \rightarrow B; B \rightarrow A; D \rightarrow C$

## t1[K]=t2[K])

解釋 交給其他人解釋了。

詳細影片 : 12/13的30:00開始

## 4.2 General Definition of Third Normal Form

- Definition:
  - **Superkey** of relation schema  $R$  - a set of attributes S of R that contains a key of R
  - A relation schema  $R$  is in **third normal form (3NF)** if whenever a FD  $X \rightarrow A$  holds in  $R$ , then either:
    - (a)  $X$  is a superkey of  $R$ , or
    - (b)  $A$  is a prime attribute of  $R$
- LOTS1 relation FD4 violates 3NF because  $\text{Area} \rightarrow \text{Price}$ ; and  $\text{Area}$  is not a superkey in LOTS1. (see Figure 14.12b).
- **NOTE: Boyce-Codd normal form disallows condition (b) above**
- The general definition can be **applied directly** to test whether a relation schema is in 3 NF; it does not have to go through 2 NF first

BCNF:

第四正規畫：

## Multivalued Dependency and Fourth Normal Form (cont'd.)

- (a) The EMP relation with two MVDs: ENAME  $\rightarrow\!\!\!>$  PNAME and ENAME  $\rightarrow\!\!\!>$  DNAME.
- (b) Decomposing the EMP relation into two 4NF relations EMP\_PROJECTS and EMP\_DEPENDENTS.

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) **EMP\_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y

**EMP\_DEPENDENTS**

ENAME	DNAME
Smith	John
Smith	Anna

Copyright © 2017 Ramez Elmasri and Shamkant B. Navathe

Slide 14- 80

第五正規化：

## Figure 14.15 Fourth and fifth normal forms.

(a) EMP			(c) SUPPLY																																					
<table border="1"> <thead> <tr> <th>Ename</th> <th>Pname</th> <th>Dname</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>X</td><td>John</td></tr> <tr><td>Smith</td><td>Y</td><td>Anna</td></tr> <tr><td>Smith</td><td>X</td><td>Anna</td></tr> <tr><td>Smith</td><td>Y</td><td>John</td></tr> </tbody> </table>	Ename	Pname	Dname	Smith	X	John	Smith	Y	Anna	Smith	X	Anna	Smith	Y	John	<table border="1"> <thead> <tr> <th>Sname</th> <th>Part_name</th> <th>Proj_name</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>Bolt</td><td>ProjX</td></tr> <tr><td>Smith</td><td>Nut</td><td>ProjY</td></tr> <tr><td>Adamsky</td><td>Bolt</td><td>ProjY</td></tr> <tr><td>Walton</td><td>Nut</td><td>ProjZ</td></tr> <tr><td>Adamsky</td><td>Nail</td><td>ProjX</td></tr> <tr><td>Adamsky</td><td>Bolt</td><td>ProjX</td></tr> <tr><td>Smith</td><td>Bolt</td><td>ProjY</td></tr> </tbody> </table>	Sname	Part_name	Proj_name	Smith	Bolt	ProjX	Smith	Nut	ProjY	Adamsky	Bolt	ProjY	Walton	Nut	ProjZ	Adamsky	Nail	ProjX	Adamsky	Bolt	ProjX	Smith	Bolt	ProjY
Ename	Pname	Dname																																						
Smith	X	John																																						
Smith	Y	Anna																																						
Smith	X	Anna																																						
Smith	Y	John																																						
Sname	Part_name	Proj_name																																						
Smith	Bolt	ProjX																																						
Smith	Nut	ProjY																																						
Adamsky	Bolt	ProjY																																						
Walton	Nut	ProjZ																																						
Adamsky	Nail	ProjX																																						
Adamsky	Bolt	ProjX																																						
Smith	Bolt	ProjY																																						
(b) EMP_PROJECTS	EMP_DEPENDENTS																																							
<table border="1"> <thead> <tr> <th>Ename</th> <th>Pname</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>X</td></tr> <tr><td>Smith</td><td>Y</td></tr> </tbody> </table>	Ename	Pname	Smith	X	Smith	Y	<table border="1"> <thead> <tr> <th>Ename</th> <th>Dname</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>John</td></tr> <tr><td>Smith</td><td>Anna</td></tr> </tbody> </table>	Ename	Dname	Smith	John	Smith	Anna																											
Ename	Pname																																							
Smith	X																																							
Smith	Y																																							
Ename	Dname																																							
Smith	John																																							
Smith	Anna																																							
(d) $R_1$	$R_2$	$R_3$																																						
<table border="1"> <thead> <tr> <th>Sname</th> <th>Part_name</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>Bolt</td></tr> <tr><td>Smith</td><td>Nut</td></tr> <tr><td>Adamsky</td><td>Bolt</td></tr> <tr><td>Walton</td><td>Nut</td></tr> <tr><td>Adamsky</td><td>Nail</td></tr> </tbody> </table>	Sname	Part_name	Smith	Bolt	Smith	Nut	Adamsky	Bolt	Walton	Nut	Adamsky	Nail	<table border="1"> <thead> <tr> <th>Sname</th> <th>Proj_name</th> </tr> </thead> <tbody> <tr><td>Smith</td><td>ProjX</td></tr> <tr><td>Smith</td><td>ProjY</td></tr> <tr><td>Adamsky</td><td>ProjY</td></tr> <tr><td>Walton</td><td>ProjZ</td></tr> <tr><td>Adamsky</td><td>ProjX</td></tr> </tbody> </table>	Sname	Proj_name	Smith	ProjX	Smith	ProjY	Adamsky	ProjY	Walton	ProjZ	Adamsky	ProjX	<table border="1"> <thead> <tr> <th>Part_name</th> <th>Proj_name</th> </tr> </thead> <tbody> <tr><td>Bolt</td><td>ProjX</td></tr> <tr><td>Nut</td><td>ProjY</td></tr> <tr><td>Bolt</td><td>ProjY</td></tr> <tr><td>Nut</td><td>ProjZ</td></tr> <tr><td>Nail</td><td>ProjX</td></tr> </tbody> </table>	Part_name	Proj_name	Bolt	ProjX	Nut	ProjY	Bolt	ProjY	Nut	ProjZ	Nail	ProjX		
Sname	Part_name																																							
Smith	Bolt																																							
Smith	Nut																																							
Adamsky	Bolt																																							
Walton	Nut																																							
Adamsky	Nail																																							
Sname	Proj_name																																							
Smith	ProjX																																							
Smith	ProjY																																							
Adamsky	ProjY																																							
Walton	ProjZ																																							
Adamsky	ProjX																																							
Part_name	Proj_name																																							
Bolt	ProjX																																							
Nut	ProjY																																							
Bolt	ProjY																																							
Nut	ProjZ																																							
Nail	ProjX																																							

Figure 14.15

Fourth and fifth normal forms. (a) The EMP relation with two MVDs: Ename  $\rightarrow\!\!\rightarrow$  Pname and Ename  $\rightarrow\!\!\rightarrow$  Dname. (b) Decomposing the EMP relation into two 4NF relations EMP\_PROJECTS and EMP\_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF but **not in 5NF** if it has the JD( $R_1, R_2, R_3$ ). (d) Decomposing the relation SUPPLY into the 5NF relations  $R_1, R_2, R_3$ .

### 期末考古題

#### 第一題：

(C) It returns the names of all customers, and any orders they might have, ordered alphabetically by customer's name.

返回所有客戶的姓名(因為left join會加入左邊表格的全部屬性), 以及他們可能擁有的任何訂單, 按客戶姓名字母順序排序。

The query is using a LEFT JOIN to retrieve all customers and their corresponding orders (if any). The result is ordered alphabetically by customer name.

#### 第二題：

(C) A view is usually realized or materialized at the time of view definition.

view是一個虛構的表格用來表示查詢的定義

In SQL, views are typically not materialized at the time of definition. Instead, a view is a virtual table that is not stored physically but is dynamically generated by the SQL engine when queried. Materialized views, on the other hand, are physical copies of the result set and are created and stored at the time of view definition.

So, the correct statement should be that views are usually not materialized at the time of view definition.

**第三題：**

(A) Designing the schema using either the top-down or bottom-up design methodology.

The quality of the design methodology itself is not typically used as a measure for the quality of the relation schema. Instead, the focus is usually on aspects like clarity of attribute semantics, reduction of redundant information, and prevention of spurious tuples. The choice between top-down and bottom-up design methodologies is a separate consideration in the design process.

(A) 設計方法論(自頂向下或自底向上)更多地涉及到設計模式的過程, 而不是模式本身的質量衡量。關係模式設計的質量通常根據清晰度、效率、無冗余以及是否符合規範化原則等因素進行評估。

**第四題：**

(B) The relation R violates BCNF.

1. (a) 32 bits  
(b) 48 bits
2. DHCP(自動分配 IP 地址和其他網路配置信息給設備, 使它們能夠在網路上通信), NAT(用於將一個網路中的私有 IP 地址轉換為另一個網路中的公共 IP 地址)
3. (1) 一個封包最多可以經過幾個路由器(ttl的用處)  
(2) 會, 因為每經過一個路由器 TTL 的值就會減一(ttl的實際應用)
4. (1) 決定封包從路由的 input port 移動到哪個 output port(DATA PLANE)  
(2) 決定封包從傳送端到目的端的路徑(CONTROLL PLANE)
5. Channel partitioning, 因為只有一個節點, 所以它可以有整個頻道進行傳輸, 不會有同一個 item slot 有兩個節點要傳輸資料的問題, 還有 channel
6.  $10_{(2)}$
7.  $RL/(L+R \times d_{poll})$
8. (1) CSMA/CD  
(2) 不會有碰撞  
(3) 不需要手動配置
- 9.