資訊管理學系 陳士杰老師

資料庫系統管理
Database System Management

DB系統其它議題
Advanced Topics

國立聯合大學
NATIONAL UNITED UNIVERSITY

Outlines

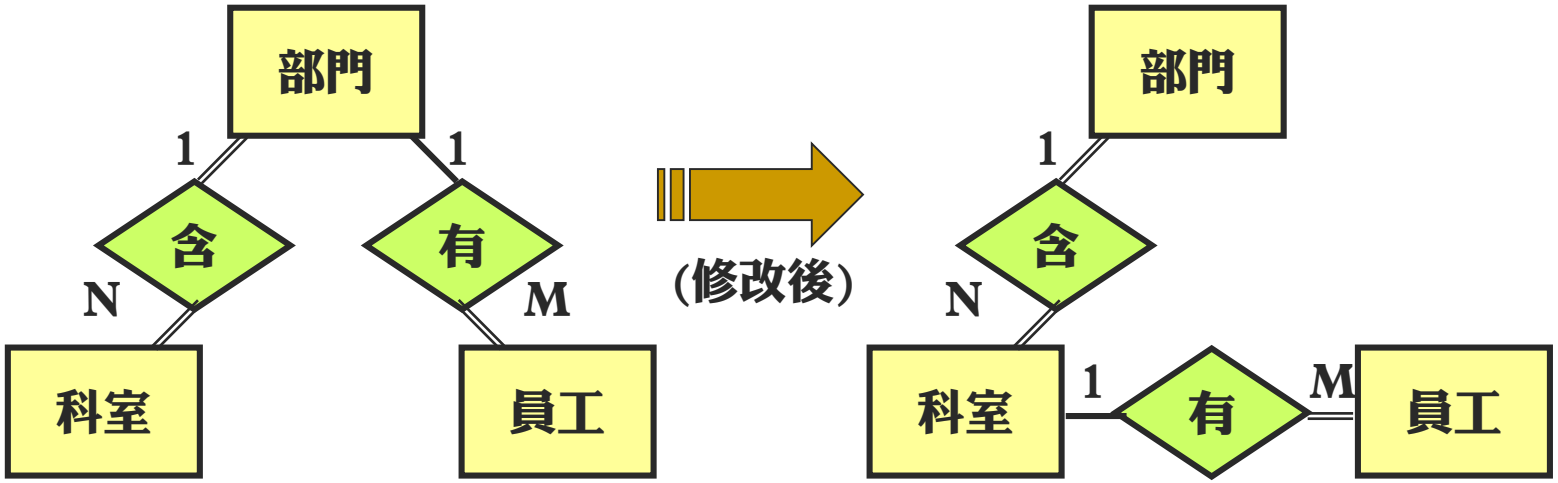
- ER Model的陷阱問題
- 查詢最佳化
- 物件導向資料庫
- 資料倉儲 (Data Warehouse)
- 線上交易處理(OLTP)與線上分析處理(OLAP)
- 資料探勘(Data Mining)
- UML(Universal Modeling Language)

【講義：Ch. 8】

【原文：Ch. 3, Ch. 15, Ch. 20, Ch. 28, Ch. 29】

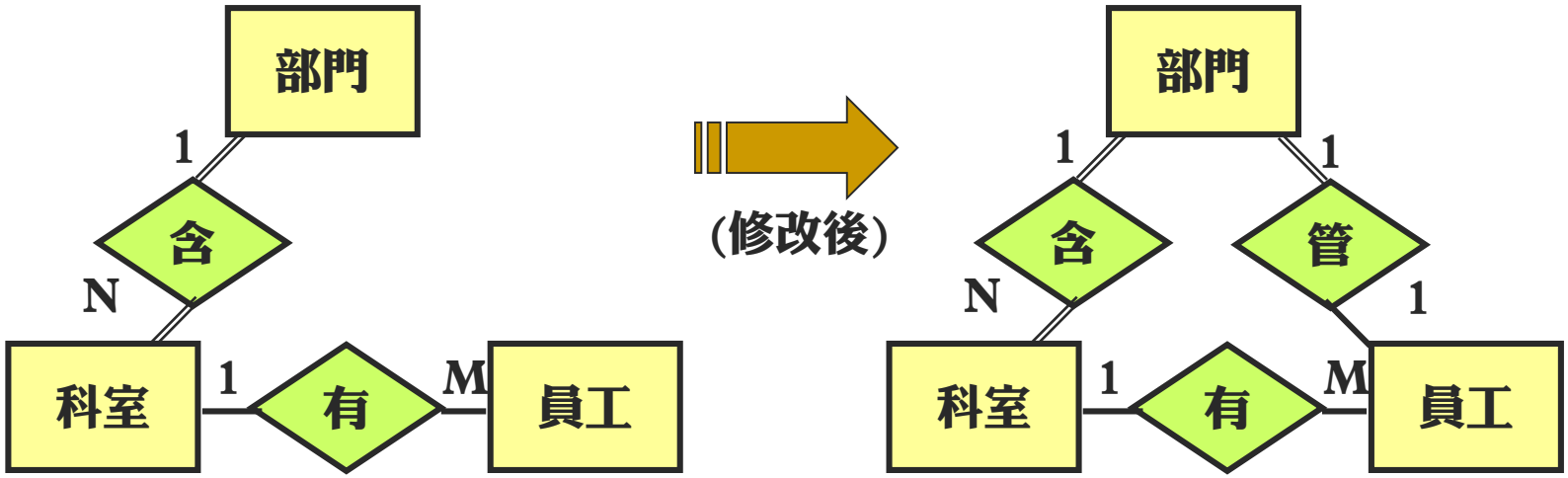
ER Model的陷阱問題

- **扇型陷阱(Fan Trap)**
  - 通常發生在兩個(以上)的一對多關係自同一個個體向外扇出時。
  - Ex:下述ER Model表示一個部門包含多個科室、一個科室僅隸屬一個部門；一個部門可有多個員工、一個員工僅隸屬一個部門：
    - 問題：找不出某員工是工作於哪一科室？此關係理論上應該是存在的!!



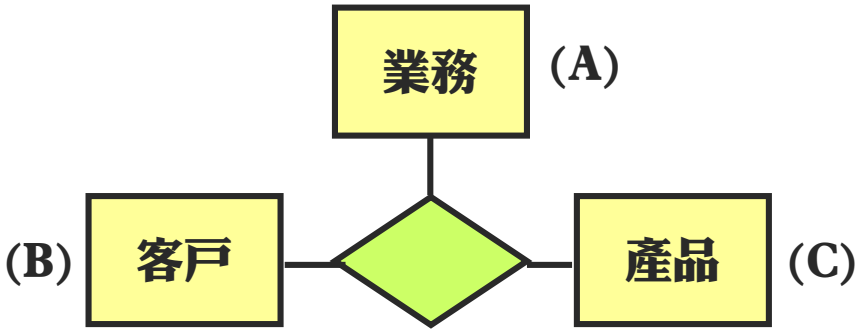
衝突陷阱(Chasm Trap)

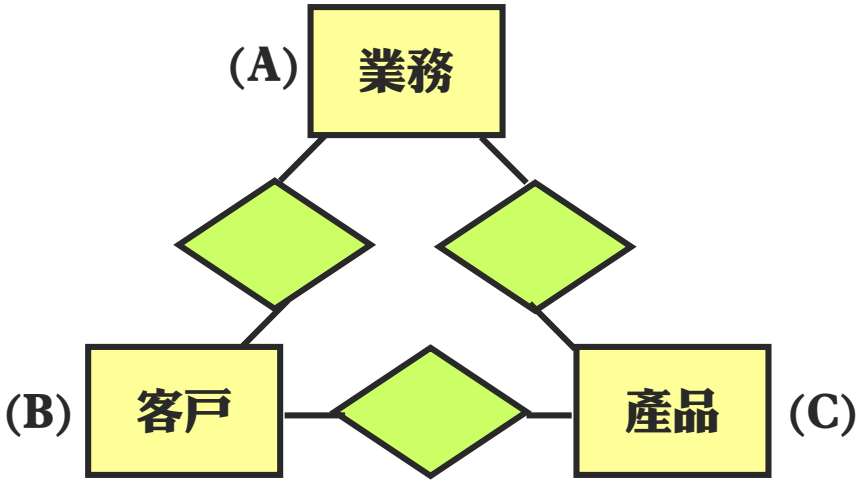
- 通常發生在關係兩端的個體有部份參與時，可能會使得較遠端個體間的關係不一定存在。
- Ex:下述ER Model表示一個部門有多個科室，每個員工隸屬一個科室，但該部門主管不屬於任何一個科室。 ：
  - 問題：找不出此主管隸屬哪個部門! 此關係理論上應該是存在的!!



一個三元關係與三個二元關係的選擇

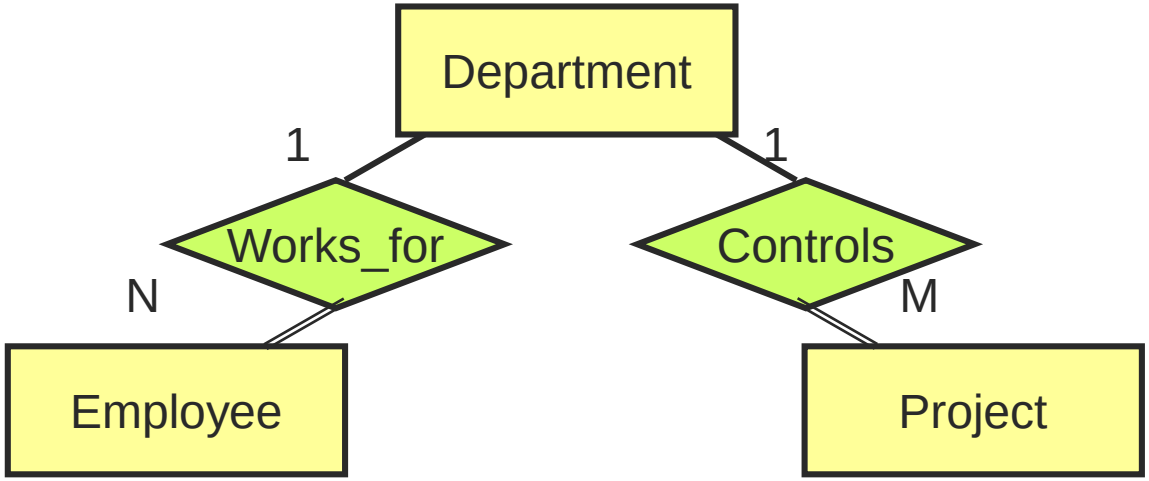
- 一個三元關係可表達的資訊：
  - 個體與個體間的**兩兩關係**。即：(A, B), (A, C), (B, C)
  - **三個個體間缺一不可**的關係。即：(A, B, C)
- 三個二元關係可表達的資訊：
  - 個體與個體間的**兩兩關係**。即：(A, B), (A, C), (B, C)





❖練習範例❖

- 假設有員工(Employee)、部門(Department)與專案(Project)三個個體，其個體關係圖(ER Model)如下圖，試問此個體關係圖可能產生何種錯誤陷阱？試舉例說明。



查詢最佳化(Query Optimization)

- 當使用者利用高階查詢語言 (如：SQL) 撰寫查詢指令，並輸入至DB系統內執行時，系統內處理指令的步驟如下：
  - 掃描 (Scanning)
    - 識別指令中的標記 (Token)。如：關鍵字、屬性名稱、關聯名稱。
  - 剖析 (Parsing)
    - 檢查指令述敘是否有依照語法規則撰寫。
  - 確認 (Validating)
    - 檢查屬性與關聯是否存在，在語意上是否有意義，以確認是否有效。
  - 查詢最佳化 (Query Optimization)
    - 一個查詢通常有許多可能的執行程序，而選擇一個最適當的程序來處理該查詢即為查詢最佳化。通常以關聯式代數來表示執行程序。
  - 產生查詢碼 (Query Code)
    - 產生執行該程序的執行碼。
  - 執行查詢

查詢最佳化的主要技術

- **啟發式查詢最佳化 (Heuristic Query Optimization)**
  - 使用關聯式代數，重新安排查詢執行程序的**操作順序**，以產生**最有效率**的執行計劃。
  - 常利用「**查詢樹 (Query Tree)**」或「**查詢圖 (Query Graph)**」。
- **成本基礎式查詢最佳化 (Cost-Based Query Optimization)**
  - 評估不同查詢執行程序的成本，找出使用哪一種關聯式代數的執行程序會**擁有最低的成本**，以此做為執行順序。
  - 一般查詢成本包括：
    - **存取 (Access) 成本**
    - **儲存 (Storage) 成本**
    - **計算 (Computation) 成本**
    - **通訊 (Communication) 成本**

    成本可能是錢或是時間。不同的資料庫系統可能會專注於不同的成本項目組合上。

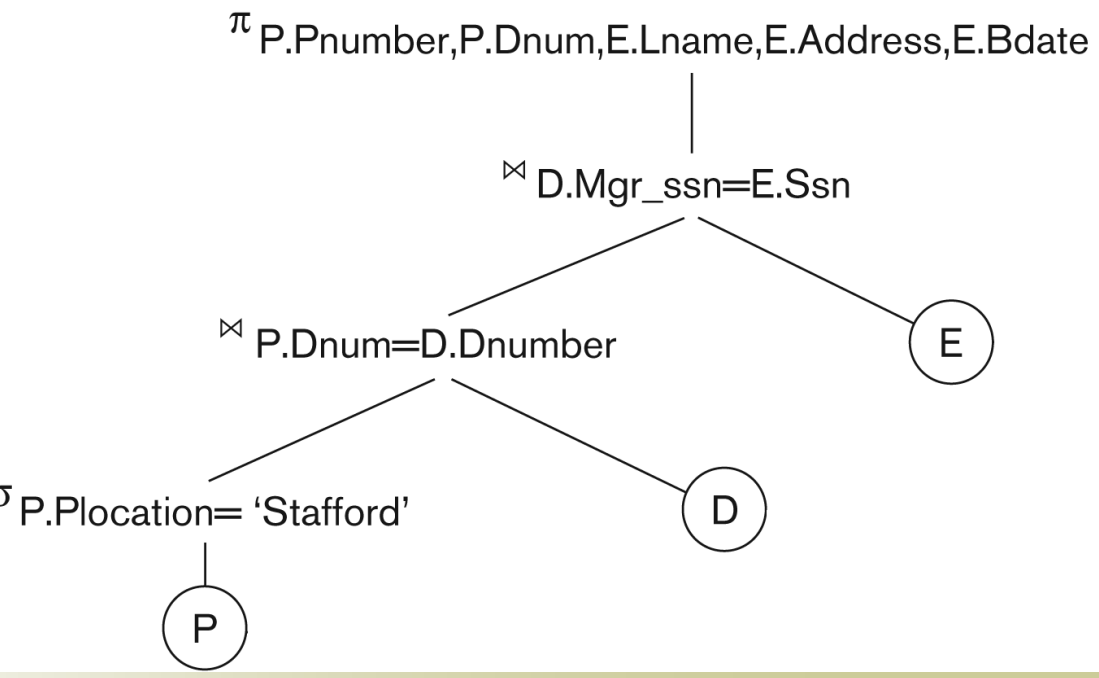
- **範例:**
  - 將每一個位於 'Stafford' 的專案，查詢出該專案的Project Number、Controlling Department Number、Department Manager's Last Name、Address 與 Birth Date.
- **SQL指令:**

```
SELECT   P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
FROM     Project AS P, Department AS D, Employee AS E
WHERE    P.Dnum=D.Dnumber AND
         D.Mgr_ssn=E.Ssn AND
         P.Plocation='Stafford';
```

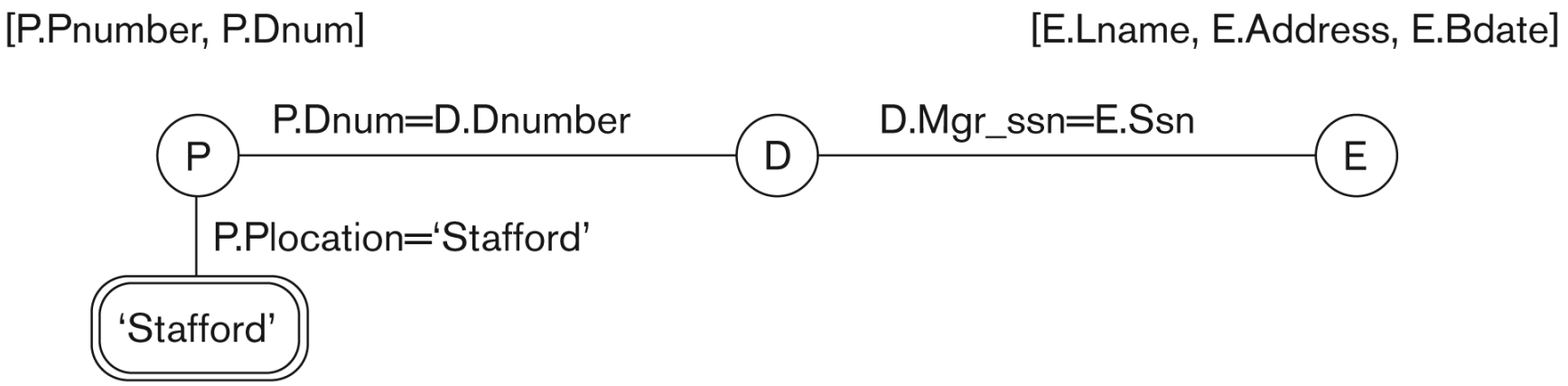
- **關聯式代數:**

$$\pi_{P.Pnumber,\ P.Dnum,\ E.Lname,\ E.Address,\ E.Bdate}(((\sigma_{P.Plocation='Stafford'}(P)) \bowtie_{P.Dnum=D.Dnumber}(D)) \bowtie_{D.Mgr\_ssn = E.Ssn}(E))$$

- **查詢樹 (Query Tree)**
  - 為關聯式代數的樹狀表示方式。查詢樹的節點規則如下：
    - 關聯表格是樹的**葉節點 (Leaf Nodes)**。
    - 關聯式代數運算子是**內部節點 (Internal Nodes)**。
- **如何還原查詢樹所表示的關聯式代數？**
  - **由下至上、由左至右**推導。
  - 直線部份**下方元件**會被**上方運算式**所包含，其餘則以**中序追蹤法**來推導。



- **查詢圖 (Query Graph)**
  - 為關聯式代數的圖形表示方式。查詢圖的節點規則如下：
    - 關聯表格 → **節點 (Nodes)**
    - 合併(Join)條件 → **邊 (Edges)**
    - 常數值 → **常數節點 (Constant Nodes; 為雙線節點)**



利用查詢樹從事查詢最佳化的步驟

① 根據關聯式代數建構查詢樹

② 限制運算子 (Select; σ) 向下移動

③ 調整樹結構

④ 以合併運算子 (Join; ⋈) 取代卡式積 (Cartesian Product; ×) 及限制運算子

⑤ 投影運算子 (Project; π) 向下移動

範例

- **假設有三個關聯表格如下所示：**

  Employee (SSN, Name, Phone, Address, Bdate, SuperSSN)

  Project (PNo, PName, Location)

  Works_on (SSN, PNo, hours)

找出參與 “Mars” 計劃，且生日在 “1960/12/31” 以後的員工姓名。其SQL指令如下：

```
SELECT   E.Name
FROM     Employee E, Works_on W, Project P
WHERE    P.PName='Mars' and
         P.PNo=W.PNo and
         E.SSN=W.SSN and
         E.Bdate>'1960/12/31';
```

- **先列出與該SQL指令相對應之關聯式代數**

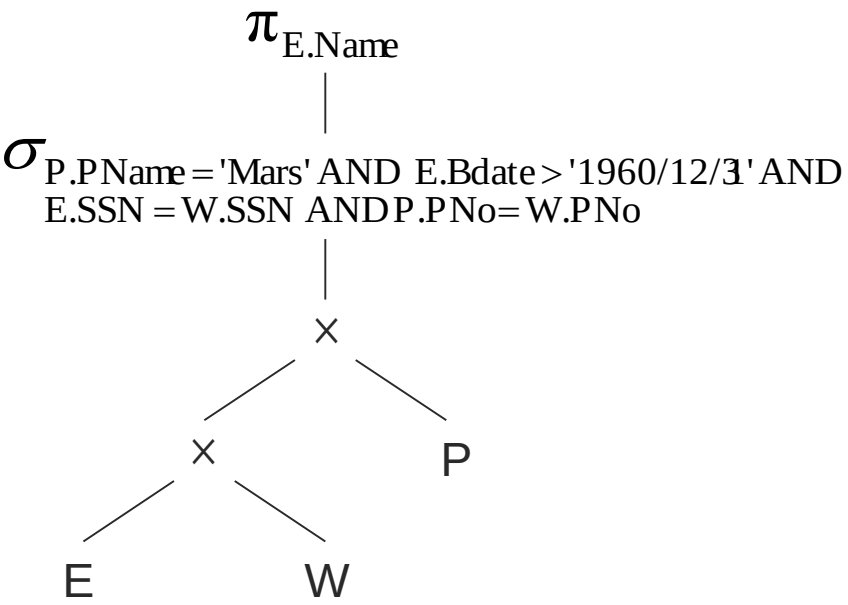
```
SELECT   E.Name
FROM     Employee E, Works_on W, Project P
WHERE    P.PName='Mars' and
         P.PNo=W.PNo and
         E.SSN=W.SSN and
         E.Bdate>'1960/12/31';
```

採最簡易的寫法即可

$$\pi_{E.Name}(\sigma_{\substack{P.PName='Mars' \text{ AND} \\ E.Bdate>'1960/12/31' \text{ AND} \\ E.SSN=W.SSN \text{ AND} \\ P.PNo=W.PNo}}(E \times W \times P))$$

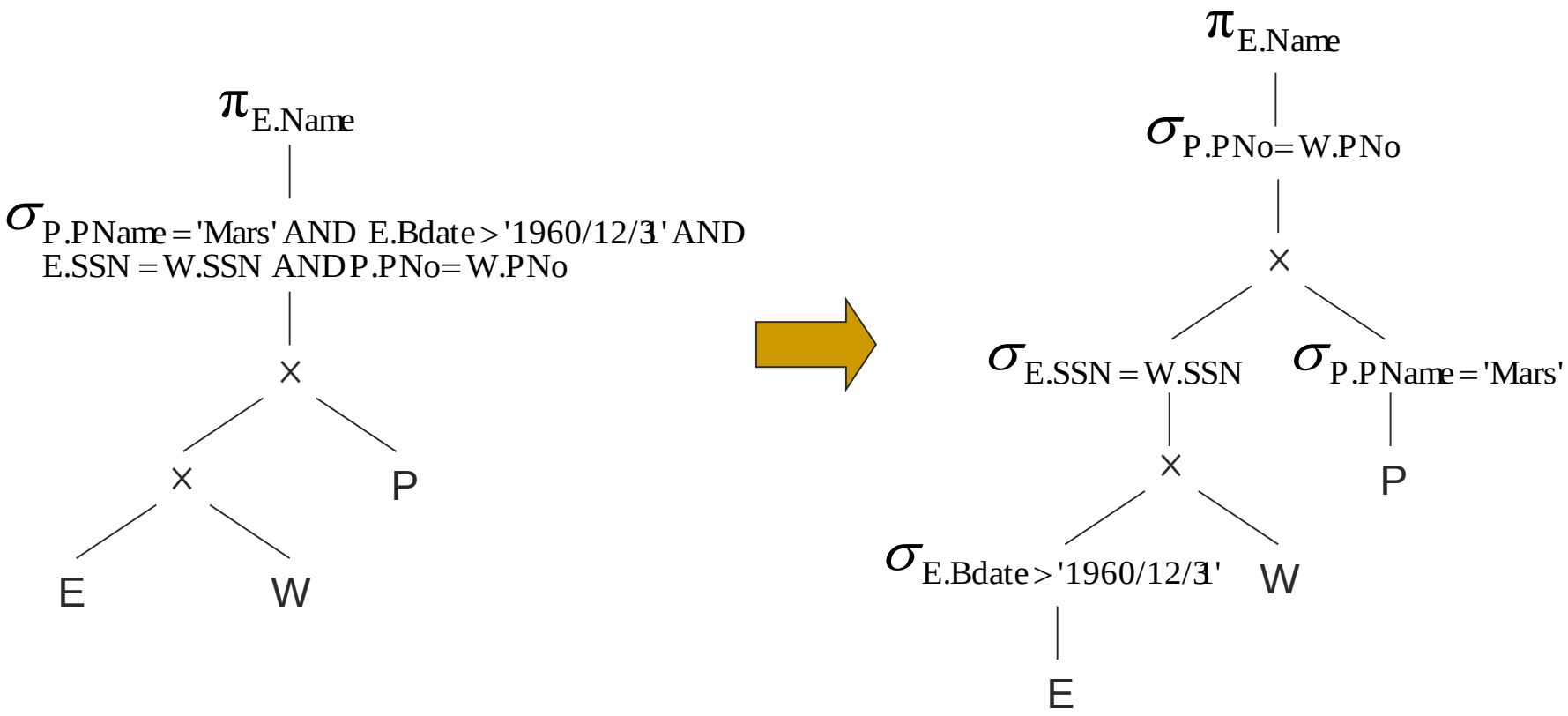
- **查詢最佳化的步驟：**
  - ① **根據關聯式代數建構查詢樹**
    - **以二元樹為主，目標投影運算子放在Root。**

$$\pi_{E.Name}(\sigma_{P.PName='Mars' \text{ AND } E.Bdate>'1960/12/31' \text{ AND } E.SSN=W.SSN \text{ AND } P.PNo=W.PNo}(E \times W \times P))$$



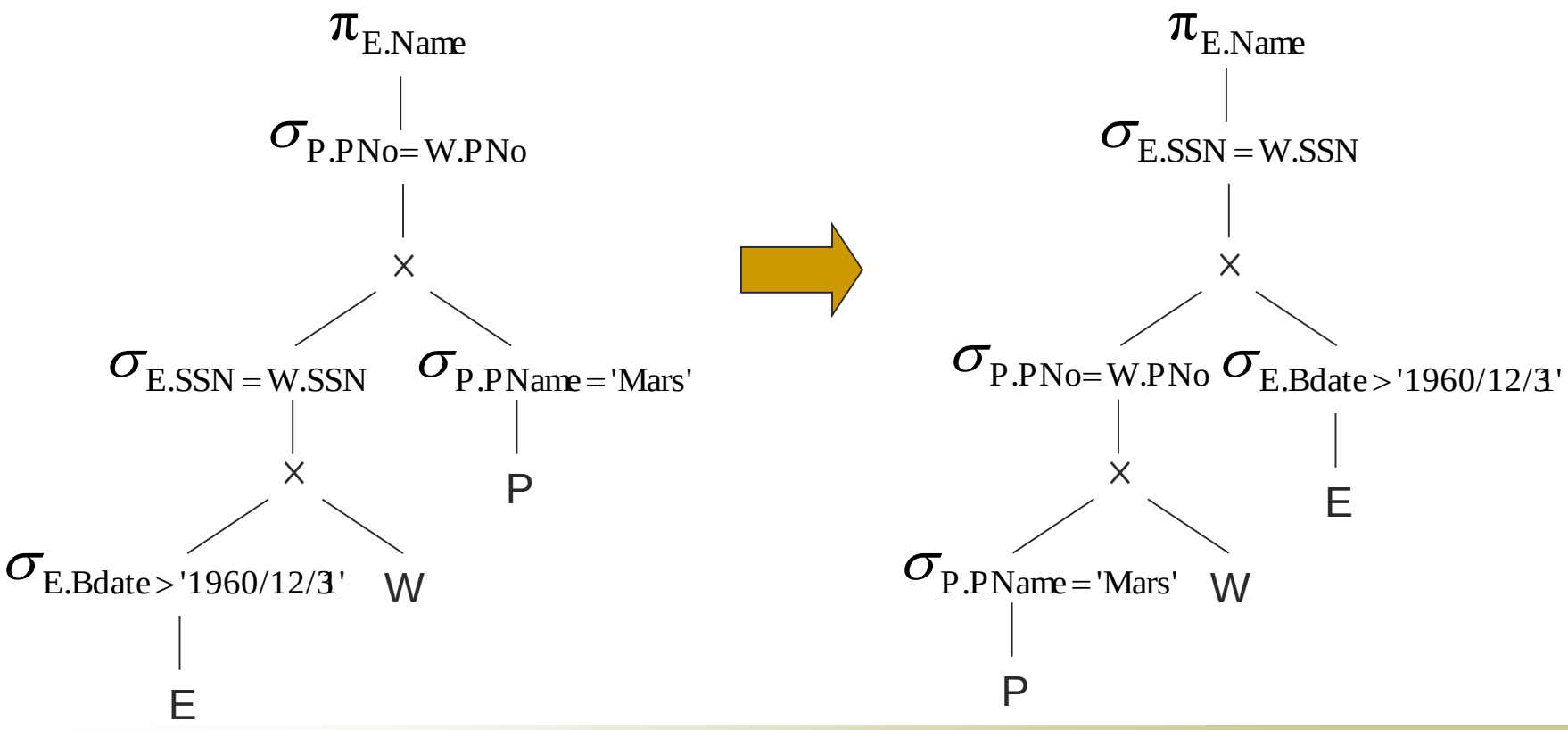
② 限制運算子 (Select; σ) 向下移動

- 限制運算子的條件全部拆解，往下放置。



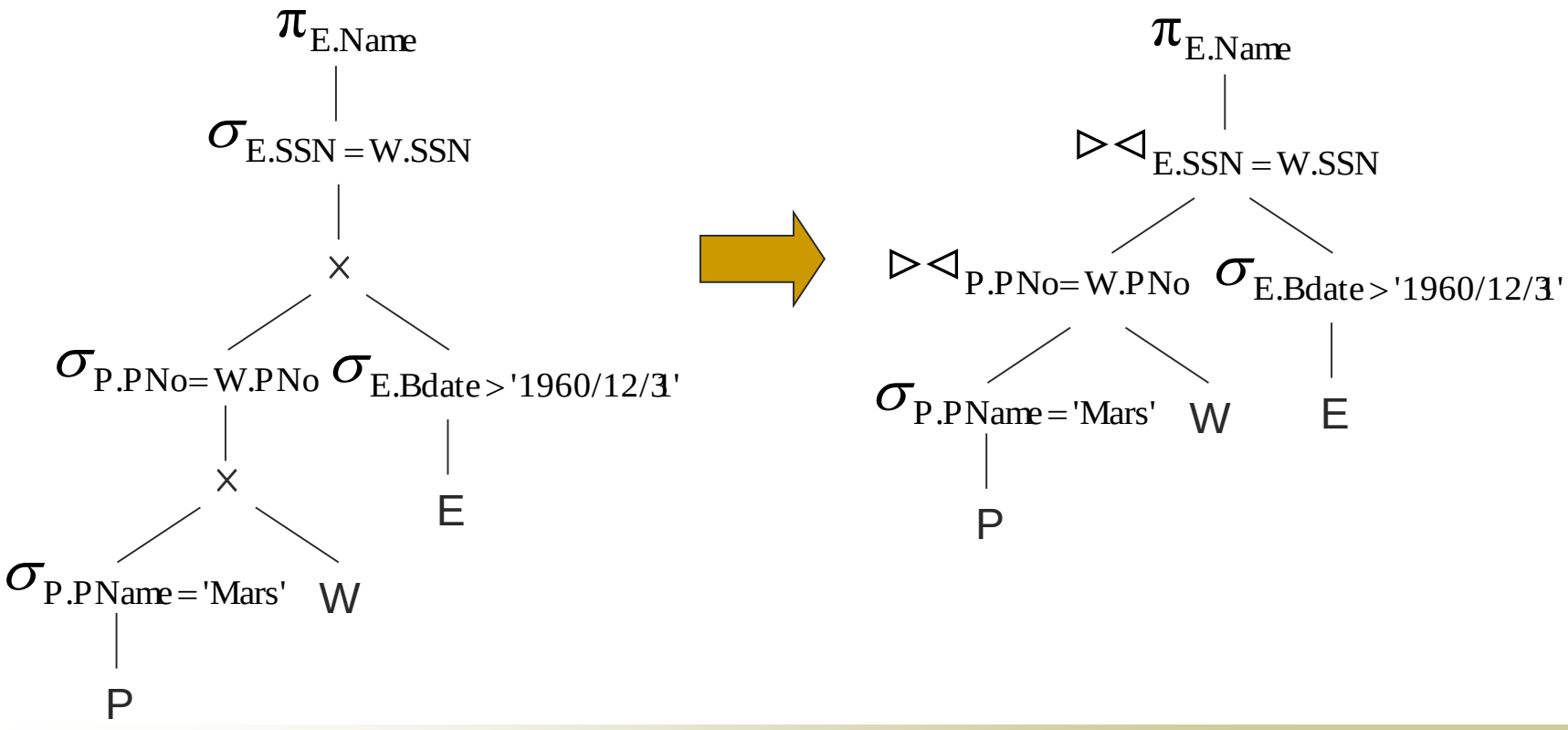
③ 調整樹結構

- 找出**最嚴格的條件** (即：能濾掉大部份資料的條件)，置換到最左下角。
- 其餘內部節點視前一置換後結果作調整。



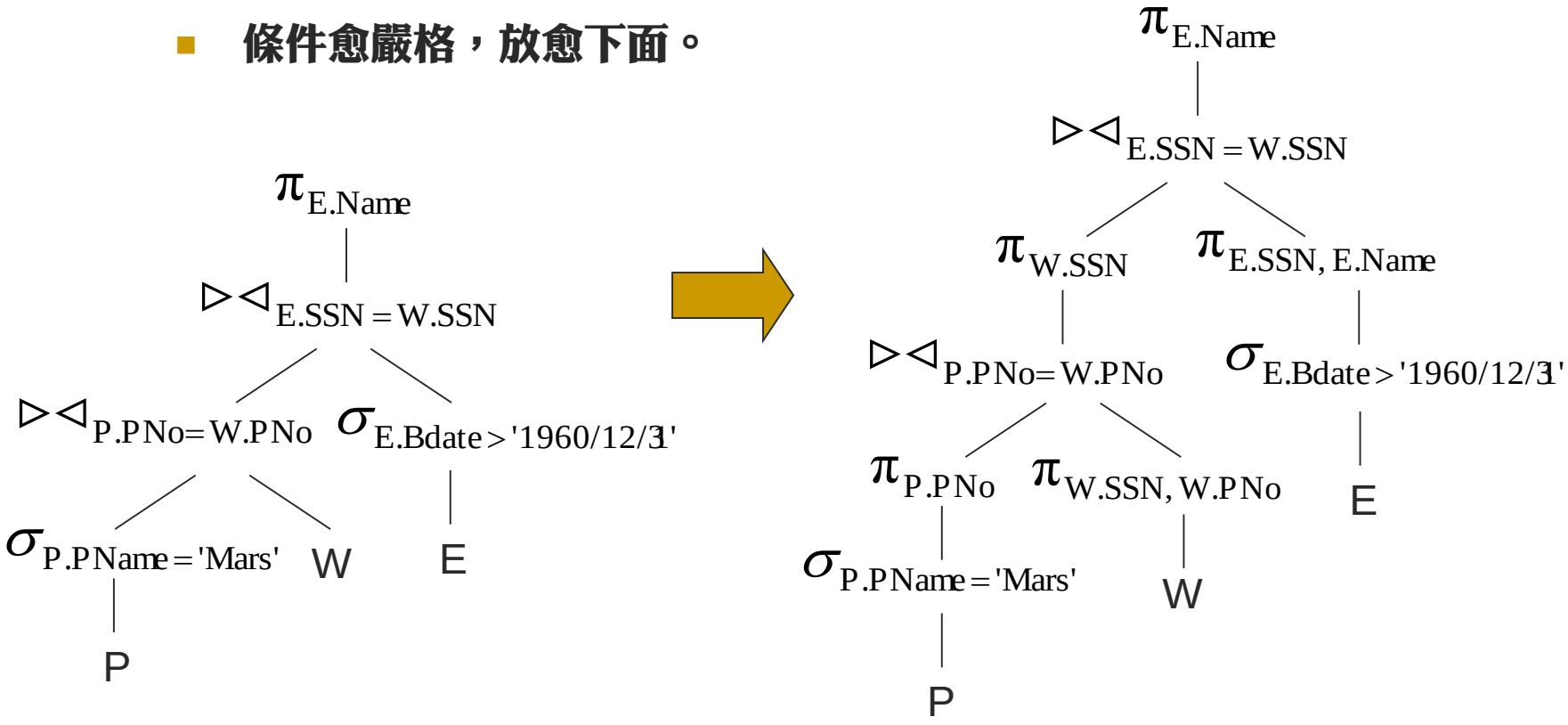
④ 以合併運算子 (Join; ⋈) 取代卡式積 (Cartesian Product; ×) 及限制運算子

- 取代**成對的卡式積與限制運算子**

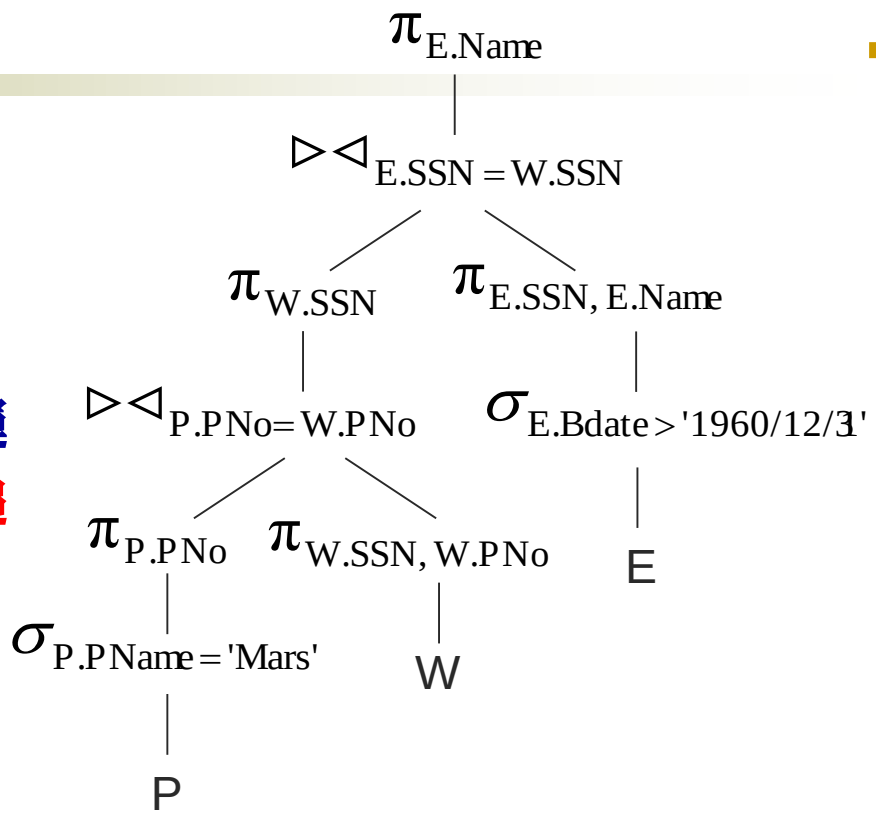


⑤ 投影運算子 (Project; π) 向下移動

- 目的：在從事合併前，就先將**不必要的屬性**，藉由投影刪除掉，僅保留需要用到的屬性。
- 條件愈嚴格，放愈下面。



- **最佳化查詢執行程序之關聯式代數結果：**
  - **由下至上、由左至右推導。**
  - **直線部份下方元件會被上方運算式所包含，其餘則以中序追蹤法來推導即可求得。**



$$\pi_{E.Name}\left\{\left[\pi_{W.SSN}\left[\pi_{P.PNo}\left(\sigma_{P.PName='Mars'}(P)\right)\triangleright\triangleleft_{P.PNo=W.PNo}\pi_{W.SSN, W.PNo}(W)\right]\right.\right.\triangleright\triangleleft_{E.SSN=W.SSN}\left.\left.\pi_{E.SSN, E.Name}\left(\sigma_{E.Bdate>'1960/12/31'}(E)\right)\right\}\right.$$

物件導向資料庫(Object-Oriented Database)

- 物件導向技術基本概念
  - Object (物件)
    - Object is an instance of a specific class (為一特定類別的實例)
    - 滿足唯一性 (Unique)，即每個Object皆是可區分的、獨立的，各Object的Attribute值 (屬性值) 之組合皆不同。
  - Class (類別)
    - 是由具有相同Attribute (屬性) 及Behavior (行為) 的物件集合而成。
    - 即：由一組Related Attributes 以及作用在這些Attributes上的Operations所定義而成。

- 由Program的角度來看O.O.：
  - Class像是一個data type
  - Object像是一個變數
  - 例：int x;
- 由Relational DB的角度來看O.O.：
  - Class像是一個 Table
  - Object像是 Table 當中的 Instance
  - 例：學生基本資料表

- Object之間的溝通方式以 “Message Passing” 為主
  - Message = Object_Name.Operation(參數)
  - 例：int x, y;

    x = y+3; → x. = (y.+(3))

    (O.O. Concepts)

    - y.+(3)⇨ y 這個物件中的 “加法”，參數為 3
    - x. = (y.+(3))⇨ x 這個物件中的 “Assign”，參數為y.+(3)的回傳結果

- Object的組成：

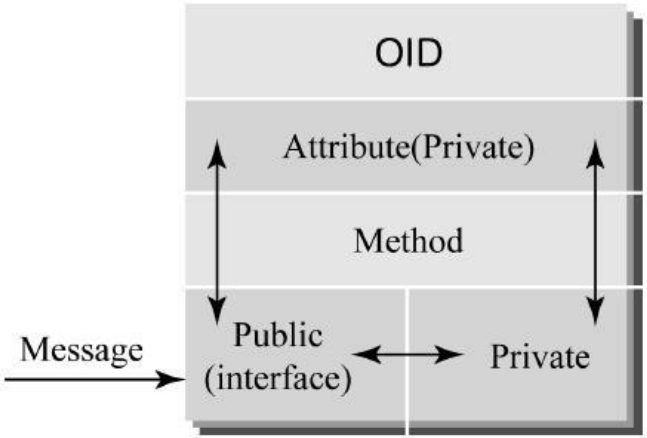


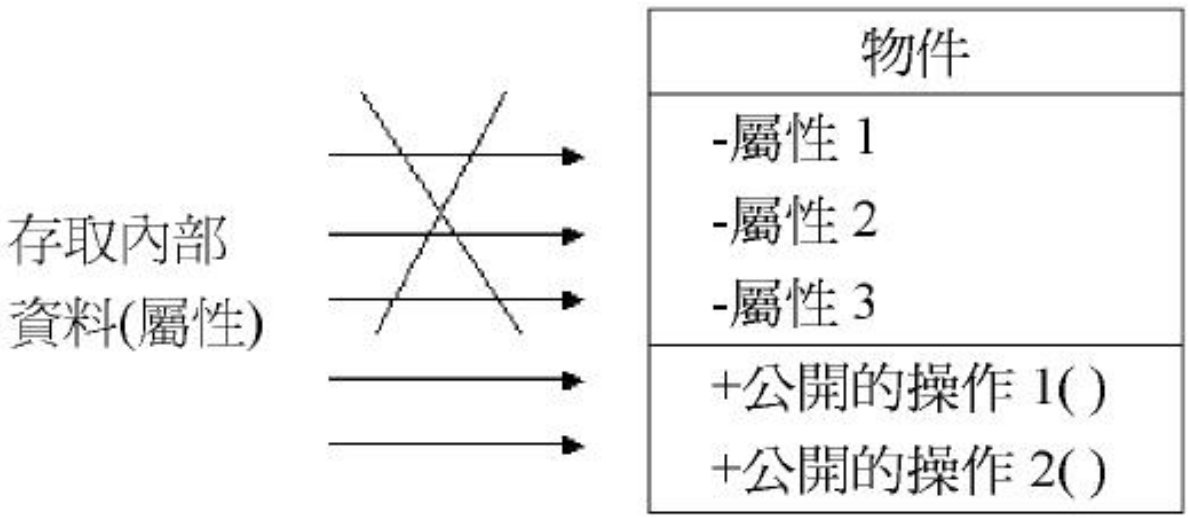
圖11-2 物件的組成

物件導向的四大特性

- Data Abstraction (資料抽象化)
- Encapsulation (封裝)
- Inheritance (繼承)
- Polymorphism (多型)

- **Data Abstraction (資料抽象化)**
  - 根據問題需求，透過抽象化的觀念可剔除物件間與問題無關的差異。
  - 定義Class/Object時，只需專注於與問題解決有關的屬性與操作，而與問題無關的部份則一律忽略。
  - 例：
    - 設計一套學校圖書館系統，其中 “讀者” 這個Class的設計，於屬性部份可能有：學號，姓名，是否停權…等，操作部份可能有：借，還，查詢…等。
    - 與問題無關的部份如：國籍，體重…等均不納入。
- Encapsulation (封裝)
- Inheritance (繼承)
- Polymorphism (多型)

- Data Abstraction (資料抽象化)
- **Encapsulation (封裝)**
  - 將Object的屬性與操作包裝在Class裡，其**製作細節**與**外界操作Class/Object時的使用方式**是獨立的。故製作細節的改變，不會影響到外界的使用。
  - 為**資訊隱藏(Information Hiding)**的概念，因外界無需知道的屬性與操作將被隱藏起來，且僅能透過某些公開的操作 (Public Operations) 做間接的使用，故具備一定程度的安全性。
  - 優點：
    - 增加程式的**可維護性**。
    - 減少**Ripple effect (漣漪效應)**。即：製作細節的修改，不會導致外界需作連動性的大幅改變。
- Inheritance (繼承)
- Polymorphism (多型)


存取內部
資料(屬性)
物件
-屬性 1
-屬性 2
-屬性 3
+公開的操作 1( )
+公開的操作 2( )

- Data Abstraction (資料抽象化)
- Encapsulation (封裝)
- **Inheritance (繼承)**
  - Class可以繼承某些**已存在Class之屬性與操作**。被繼承者稱為Parent Class (父類別)、Supper Class (超類別)；繼承者稱為Child Class (子類別)、Sub Class (次類別)。
  - 子類別也可以**自行增加**所需的屬性及操作，或是**修改**繼承而得的操作之製作細節 (**Overriding; 覆蓋**)
  - 優點：
    - 可達成**軟體再利用 (Reuse)**，節省重複開發成本。
    - 通常開放出來的元件大多是經過驗証的，直接使用元件進行開發程式，可使錯誤率下降，品質提升。
- Polymorphism (多型)

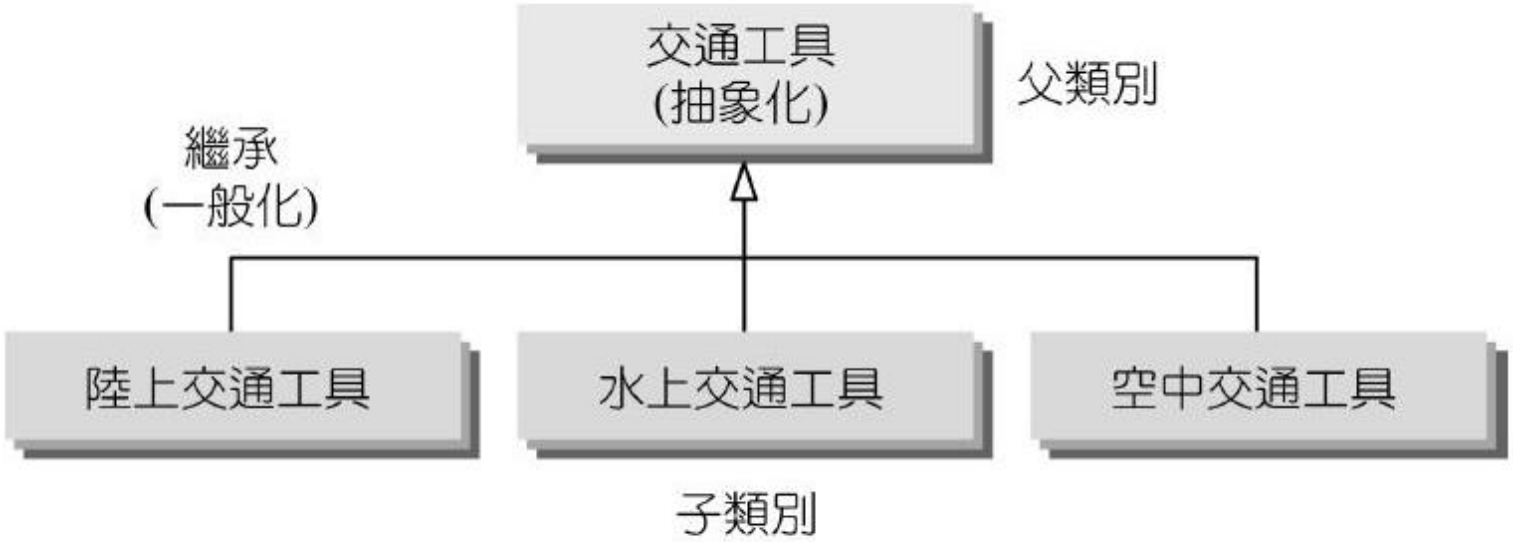
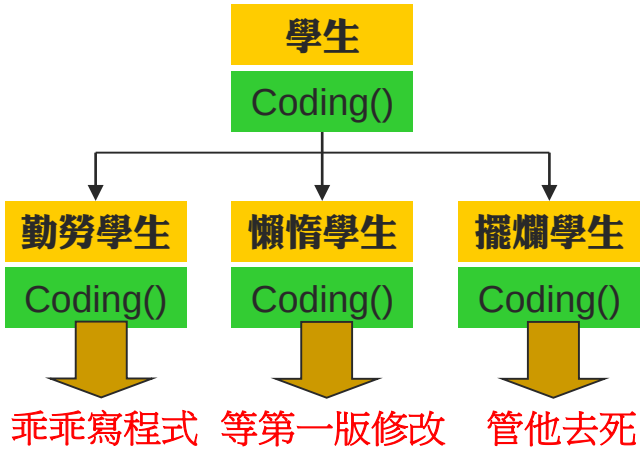


圖11-6　繼承性示意圖

- Data Abstraction (資料抽象化)
- Encapsulation (封裝)
- Inheritance (繼承)
- **Polymorphism (多型)**
  - **相同的Message送給不同的Class/Objects，可引發不同的行為模式**
  - **例：**



  - **優點：增進程式修改/維護的高度擴充彈性 (Flexibility)**

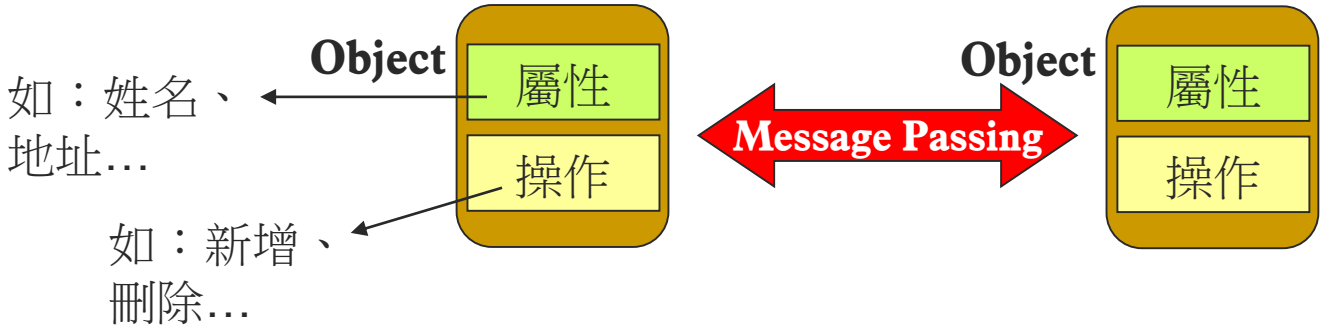
Class (Object) 之間的關係

- **Aggregation (聚合關係)**
  - 又稱：Whole-Part, Composition, Is-part-of
  - 即：物件中又有物件，又稱衍生物件
  - 例： “Computer”包含 “Motherboard”, “Mouse”, “Keyboard”, …
- **Use (使用關係)**
  - 又稱：Client-Server
  - 即：使用別人的服務
  - 例： “後勤單位” 支援 “戰鬥單位”
- **Inheritance (繼承關係)**
  - 又稱：Is-kind-of
  - 例： “研究生” 是 “學生” 的一種
- **Association (關聯關係)**
  - 即：Class/Object間的關係，因某個Operation執行後建立或消除
  - 例： “讀者” 因借 “書” 而產生兩者間的關係；因還 “書” 而使得兩者關係消除。

物件導向資料庫系統

- 將各種個體(Entity)以**物件(Object)**方式表示；而物件之間則透過**訊息(Message)**來從事溝通。
- 將整個資料庫以一組「抽象化資料型態(Abstract Data Type)」來表示。這組抽象資料型態各有其所屬的**運算(Operator)**以及屬於其型態中的「**物件實例(Object Instance)**」。這些物件實例皆透過定義於其資料型態上的運算來存取或從事適當處理。

- 資料型態間有縱向(Generalization and Specialization)、橫向(Aggregation)與繼承(Inheritance)關係。
  - 縱向(Generalization and Specialization)關係：
    - 「一般化」是用來強調各個個體間的共同性
      - 例：“卡車”、“汽車”兩個個體可以一般化 (Generalization) 為 “車子”這一個個體。
    - 「特殊化」是用來強調各個個體間的差異性
      - 例：“車子”個體中的所有實例可區分為兩大類的個體：“卡車”、“汽車”。
  - 橫向(Aggregation)關係：
    - 又稱聚合關係，所討論的是衍生物件，即物件中又有物件。



- **特性：**
  - **以物件(Object)及其特性來操作資料庫**
  - **以不變的物件識別碼(Object Identifier; OID)來唯一區別物件**
    - **OID是由系統給定，而關聯式資料庫中的表格主鍵是由人給定。**
  - **以類別(Class)來表達階層關係，且擁有衍生物件的聚合關係。**
  - **以方法(Method)來操作類別。**
    - **此方法為事先宣告好的方法!!**

RDBS v.s. OODBS

RDBS	OODBS
Table	Class
View	Virtual Class
Row of Table	Instance of Class
Column of Table	Attribute
Procedure	Operation (Method)
Child Table	Subclass
Parent Table	Superclass
Table Hierarchy (Relationship)	Class Hierarchy (Relationship)

- **關聯式資料庫系統的缺點：**
  - **缺乏豐富的資料模式化(Modeling)能力**
  - **資料模式和處理環境不一致**
    - **OOPL與OODB整合性較佳，而關聯式資料庫與前端程式則相反。**
  - **系統效率低**
    - **Join操作費時。**
  - **無法表示各種資料型態**
    - **僅能處理文字(text)或數值格式的資料**

- 物件導向資料庫系統的優點：
  - 擴充性(Extensibility)
    - 屬性、操作與外在的方法(Method)獨立，因此新增物件並不會影響系統的繼續執行
  - 操作限制
    - 操作只限於事先所定義的方法(Method)，無法進行超越權限的操作。
  - 型態定義的彈性
    - 型態定義較無限制，可利用繼承的特性，一層層定義出所需的型態。
  - 強大的模式化能力
    - 具有繼承、集合等能力，可用來從事模式化，更可詳細描述物件間真正的情況
  - 容易與物件導向程式語言整合

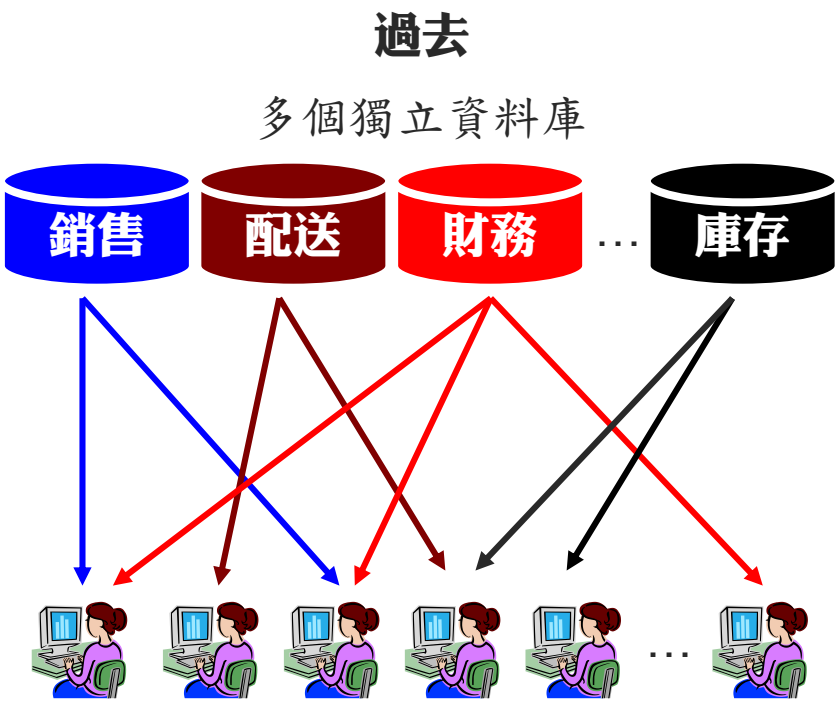
- 物件導向資料庫系統的缺點：
  - 沒有高階的查詢語言
    - 物件導向資料庫之查詢語言尚未標準化
  - 並未對關係(Association)直接支援
    - 因此仍需要關聯式資料庫系統的外來鍵(Foreign Key)
  - 多重繼承
    - 物件導向資料庫若支援多重繼承，容易產生混亂
  - 不易進行最佳化
    - 大多數的查詢都是使用者自撰的方法(Method)，因此難以進行查詢最佳化工作
  - 許多關聯式資料庫的問題，仍然出現在物件導向資料庫中
    - 在關聯式資料庫中棘手的問題依然存在
  - 實作產品不多

- **物件導向資料庫系統所應具備的能力**
  - **具有一般資料庫系統所具有的所有外觀**
    - 應提供一般DBMS的功能，如：減少資料重複性(Redundancy)、減少不一致性(Inconsistency)、資料共享、查詢處理、交易管理、整合性與安全性…等。
  - **辨識物件**
    - 給予物件一個唯一的識別碼，保証物件的獨特性，以辨識不同之物件。
  - **資訊隱藏與封裝性**
    - 使用者或應用程式不需且不能接觸物件內部的屬性與操作，物件內部的結構被隱藏起來，透過所定義的方法呼叫物件導向資料庫進行作業。
  - **支援複雜物件**
    - 物件導向資料庫可以定義一些具有非常複雜型態的物件。如：聲音、圖形、影像、時間序列…等。

- **類別化能力**
  - 將具有相同特性的物件集合成同一類別，以抽象資料型態來表示。
- **表達繼承性(Inheritance)與多重繼承(Multi-inheritance)**
  - 表達物件間的繼承關係，同時也要支援多重繼承及處理相關模糊的問題。
- **多型能力(Polymorphism)**
  - 或稱為運算子超載(Operator Overloading)。允許物件可以在不同的情況下，賦予同一個操作不同的意義，以提高物件的再用性(Reusability).
- **版本管理(Version Management)**
  - 系統常隨需求改變而變更，因此應保存歷史的版本。

資料倉儲(Data Warehouse)

- 不同主題之資訊分析架構：



- **資料庫系統：**
  - **以單一應用為導向**
  - **在確定使用者之需求與應用後，多務求短期、即時與例行性之資料查詢與異動處理需求，故所納入的資料範疇較小，內容較詳細且同質性較高**
  - **此類系統為了因應資料即時且經常異動之特性，而採用高度正規化的資料儲存方式 (即：關聯表格正規化)，故不利進行跨資料庫之資料整合與長期之趨勢分析。**
- **資料倉儲：**
  - **以主題為導向**
  - **橫跨多個領域的主題相關資料，經由萃取、轉換及清理的資料整合工作，將長期的歷史資料載入資料倉儲中，並定期性匯入新增資料擴充其內容**
  - **可滿足例行性資料處理與較複雜的決策、查詢與長期趨勢分析需求**

- 資料倉儲可視為一個非常大型的資料庫。它可將分散各地、不同性質與功能的資料庫之資料，經由ETL的程序加以整併並儲存。
  - ETL：為資料清理 (Data Cleaning)的過程，包含萃取 (Extract)、轉置(Transform)、載入(Load)。
- 目的：
  - DW的目的在於整合及運用資料。
  - 將所有資料儲存在同一個地方，透過各種分析方法，如：線上分析處理(OLAP)、資料探勘(Data Mining) 對這些資料進行分析，以幫助決策者能從大量的資料中，分析出有價值的決策資訊，擬定策略與快速回應，幫助決策者或知識工作者從事更快及更好的決策。

資料倉儲 vs. 資料庫

	資料庫	資料倉儲
資料來源	個別儲存與應用的孤立資料	異質性的企業整合資料
時間性	目前的資料	目前資料與歷史性資料
資料內容	作業性資料	作業性資料與分析決策資料
儲存平台	可能為不同平台、分散儲存	儲存於單一平台
資料觀點	以作業性或功能性觀點	企業主題式資料觀點
資料展現	使用者不同介面	資料不同維度展現
揮發性	資料隨時揮發(更新)	資料不變性

- 資料倉儲的應用：
  - 資訊處理的支援
  - 分析性的程序
  - 資料探勘 (Data Mining)

- **DW具備以下四個特性，用以支援管理決策：**
  - **主題導向(Subject-Oriented)：**
    - **資料倉儲中存放著用以處理不同主題的資料，如：客戶、產品、銷售…等。**
  - **整合性(Integrated)：**
    - **資料可來自組織內外不同的異質性(Heterogeneous)來源，如：不同的資料庫、檔案…等，予以整合，並轉換為標準化、一致化的格式。**
  - **時間變動性(Time-variant)：**
    - **可累積歷史性(Historical)資料。依據不同時間點，反應不同時間點的資料，以分析比較不同時間點的資訊(如：5到10年)。**
  - **不可揮發性(Nonvolatile)：**
    - **DW儲存長期間的歷史資料，大多為讀取(Read)、新增(Add)資料的操作，極少修改或刪除。**

- 資料倉儲中的資料類型：
  - 整合性(Integrated)資料
  - 細目(Detailed)資料
  - 彙整(Summarized)資料
  - 歷史性(Historical)資料
  - 中繼資料(Metadata)
  - 其它

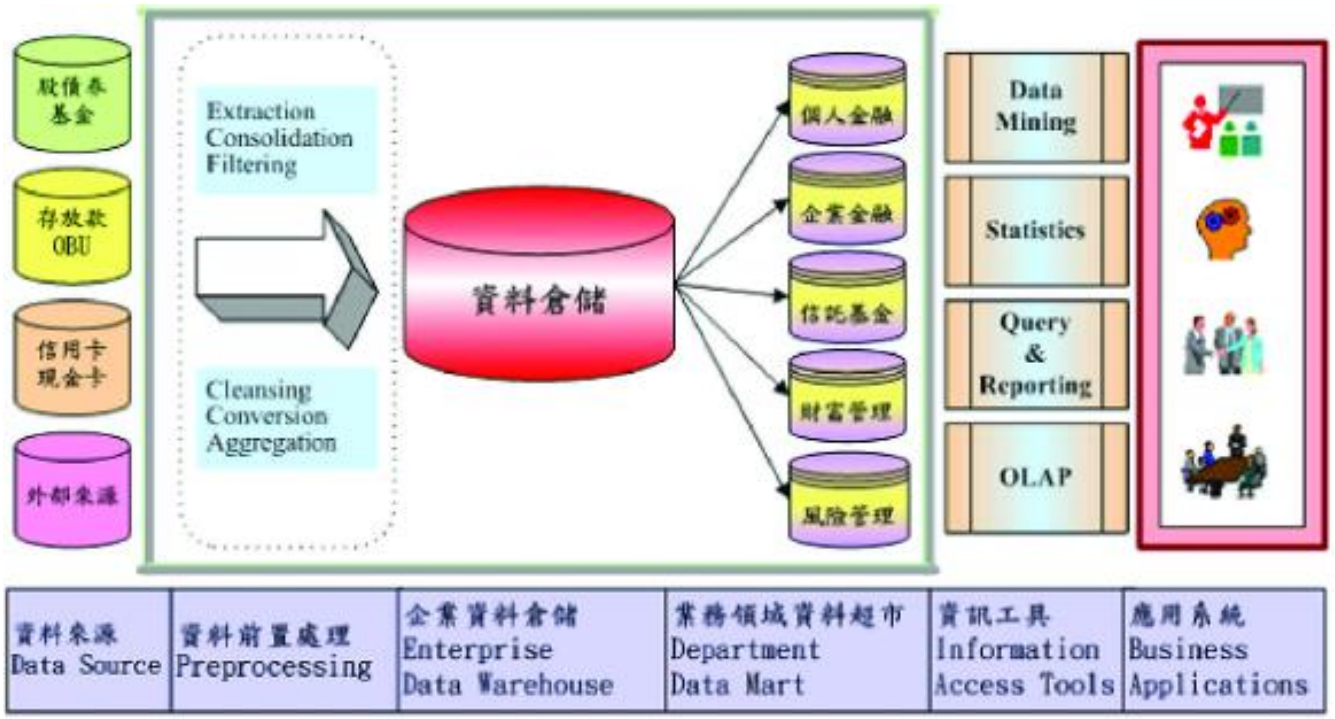
- 資料倉儲的架構：

個別階層 (Individual)	■ 資料倉儲**前端的各種查詢工具**。
部門階層 (Departmental)	■ 不同部門應用的**資料超市(Data Mart)**。 ○ Data Mart是**資料倉儲的一個子集**。 ○ 為了某個主題或應用的決策參考，擷取資料倉儲中部份資料，形成較小的資料倉儲，以提供特殊的需求。
元素階層 (Atomic)	■ 指**資料倉儲**本體。
作業階層 (Operational)	■ 指**企業內外部的資料來源**，有作業資料庫、現有資料庫、分析工具所使用的資料、企業外部資料…等。

應用

區分成

集結成


股債券
基金
存放款
OBU
信用卡
現金卡
外部來源
Extraction
Consolidation
Filtering
Cleansing
Conversion
Aggregation
資料倉儲
個人金融
企業金融
信託基金
財富管理
風險管理
Data
Mining
Statistics
Query
&
Reporting
OLAP
資料來源
Data Source
資料前置處理
Preprocessing
企業資料倉儲
Enterprise
Data Warehouse
業務領域資料超市
Department
Data Mart
資訊工具
Information
Access Tools
應用系統
Business
Applications

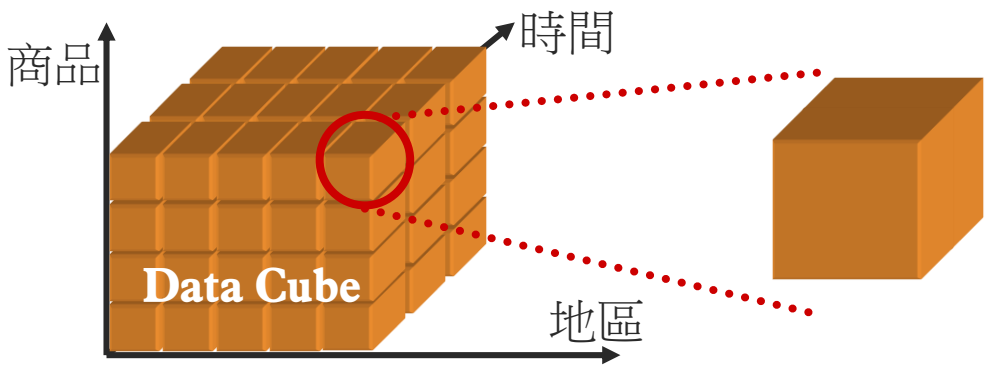
資料倉儲 vs. 資料超市

	資料倉儲	資料超市
開發時程	專案時間長，一般約12~36個月	專案時間短，一般約4~12個月
建置目的	以企業整體為考量，不針對特定應用或主題，可作為各業務之資訊分享中心	多是為特定應用或主題而建置的資料倉儲架構
成本效益比	初期建置成本較高，但隨資料量成長及架構上彈性擴充，加上可供業務單位應用，長期而言，成本效益顯著。	初期建置成本較低，短期專案效果佳。長期而言，因不斷延展，加上資訊不易分享給其它單位，代價相對昂貴。

- **資料倉儲的優點：**
  - **資料來源更廣泛**
    - 可由不同來源的資料庫擷取資料。
  - **可獲取最新的記錄，及歷史性記錄**
    - 可由資料庫中取得最新、具時效性的資料，並累積歷史性記錄，以從事時間性的分析。
  - **可滿足決策上的需求**
    - 如利用資料探勘(Data Mining)從事分析，改善決策品質。
  - **包含對資料模式化與重新模式化的能力**
    - 以不同維度重新分析資料 (Data Cube; 資料方塊)。
  - **存取資料不影響正在執行之作業程式的效率**
    - DW的資料與DB資料分開儲存與執行。
  - **降低企業成本，增加效率**
    - 降低硬體與作業成本，並有效運用組織資源，提升企業的效率與生產力。

資料方塊 (Data Cube)

- Data Cube是從資料倉儲/資料超市所建立的資料集合，由**維度 (Dimension)**與**量值(Measure)**所定義的**多維度結構**，這種架構可提供使用者快速而複雜的查詢。
  - **維度**：是**敘述性質**的資料表示方式，提供對量值資料分析的不同觀點。例如時間、地區、產品、付款方式、客戶性別等。
  - **量值**：是屬於**計量化的資料**，為資料分析中最感興趣的項目。如銷售數量、總金額、單價、利潤等。

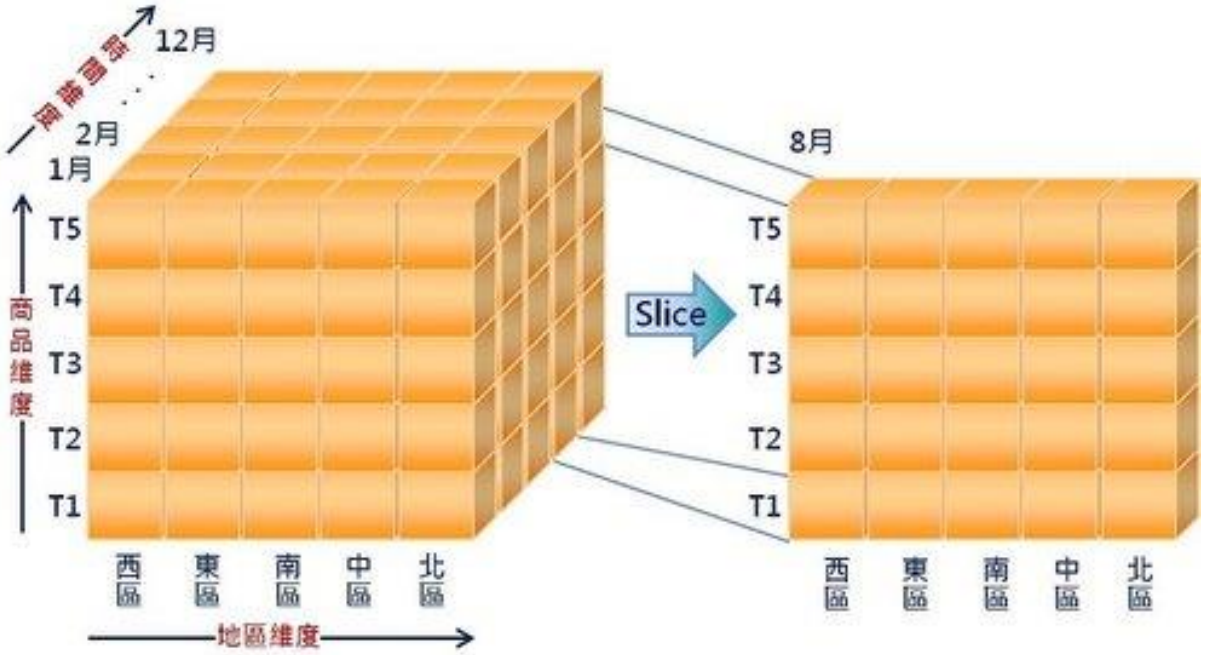


維度資料：
某一時間點，性別為男(女)的顧客，於某地區購買了A商品。

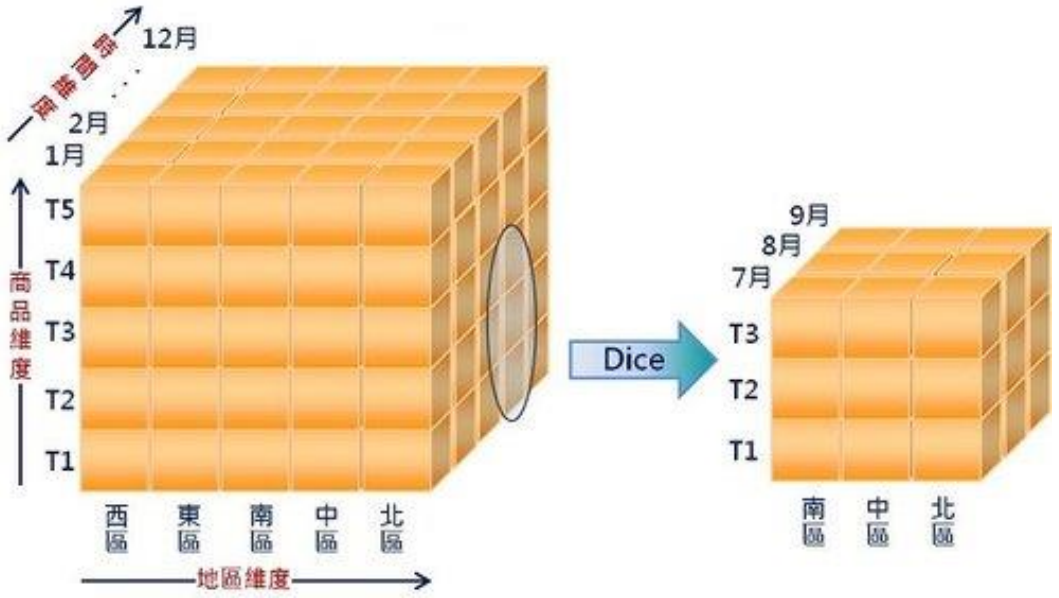
量值資料：
該交易項目之銷售數量，總金額，單價。

- Cube中的每個維度可依需求再建立數個階層，像是時間維度可以分為年、季、月、週、日等階層。如此的設計可以讓使用者用以最簡單的方式操作這些維度階層，分析各階層的數值以及查詢較複雜的問題。
- Data Cube相關操作：
  - Slice (切片)
  - Dice (切塊)
  - Drill-down (向下細分)
  - Roll-up (向上彙整)
  - Pivot (旋轉)

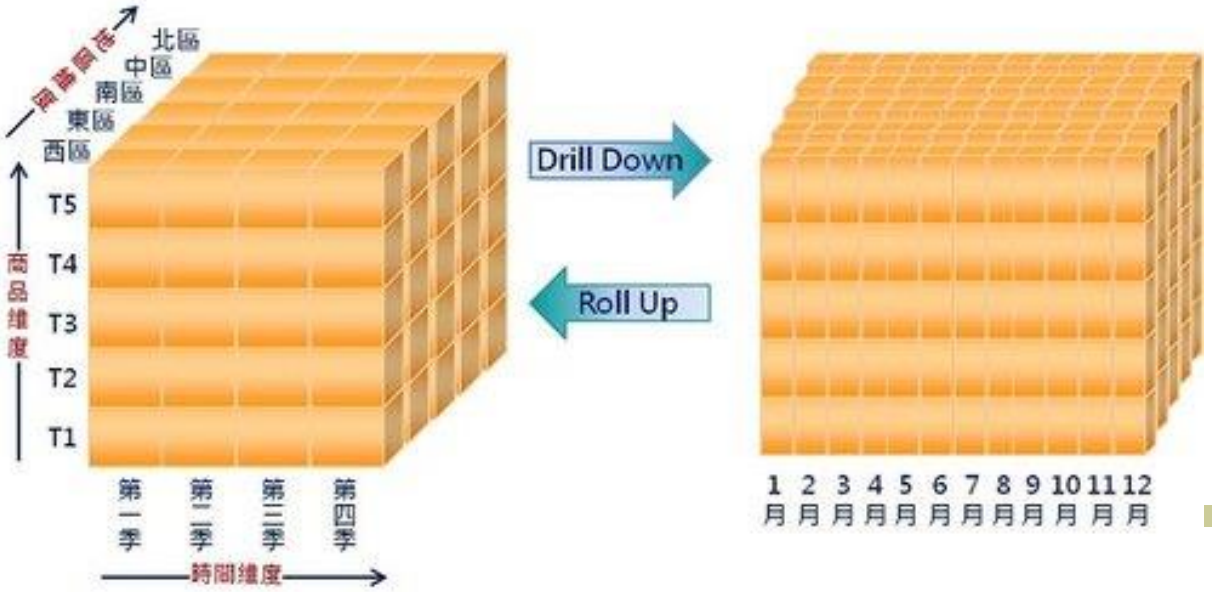
- **Slice (切片)**
  - 假設將資料方塊視為一個三維的立方體，Slice是把它切成薄薄的一片，亦即**針對某個維度限制其資料範圍**(通常是固定一個維度的值)，使原本是三維的資料切割成二維的資料。
  - 例如：全年度的銷售資料，只挑選8月份的銷售情形進行評估。



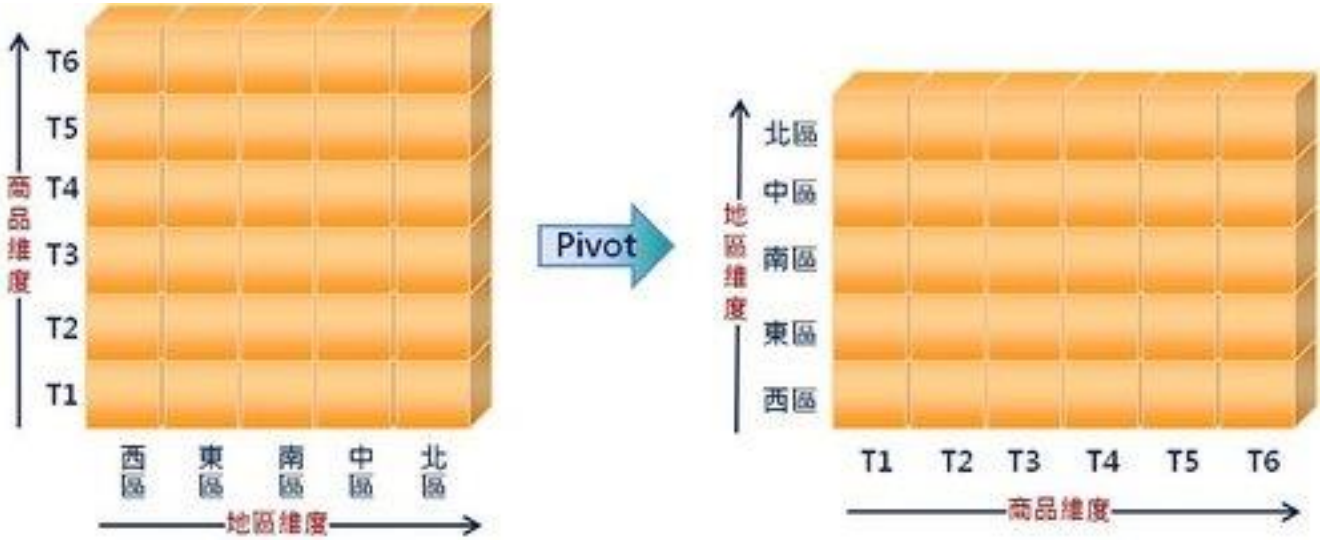
- **Dice (切塊)**
  - 選擇二個或二個以上的維度，被選中的維度皆限定一個特定資料範圍，進而切出一個範圍較小資料方塊，以進行細部分析。
  - 例如：原本的資料方塊彙總全年五個區域五項商品的交易資料。經由Dice運算，將資料範圍限制在其中三項商品，只彙總七至九月這段時間內，北中南區域的交易資料。



- **Drill-down (向下細分)、 Roll-up (向上彙整)**
  - **Drill-down**：從彙總的資料**深入到明細的數據**，以進行檢視或增加新的維度。以時間維度來說，假設目前呈現的方式是以「每季」為單位，那麼透過Drill Down功能，可以做到以「月」、「週」、「日」等較低階的明細資料。
  - **Roll-up**：將某一維度的低階明細資料**概括到高階的彙總資料**，做向上彙總運算，或是減少維度。例如以「日」為單位的表示，利用Roll-Up的功能，可以做到以「週」、「月」、「季」、「年」的彙總資料表示。



- **Pivot (旋轉)**
  - **變換維度的方向，即在表格中重新安排維度的放置（例如行列互換），讓User從不同的角度檢視資料。**
  - **例如：以時間、商品、地區三個維度所建構的銷售狀況資料方塊，可以選擇以區域維度為軸心，檢視每個區域各種商品銷售情況，或者轉換成檢視每種商品在各個區域銷售的情況。**



OLTP與OLAP

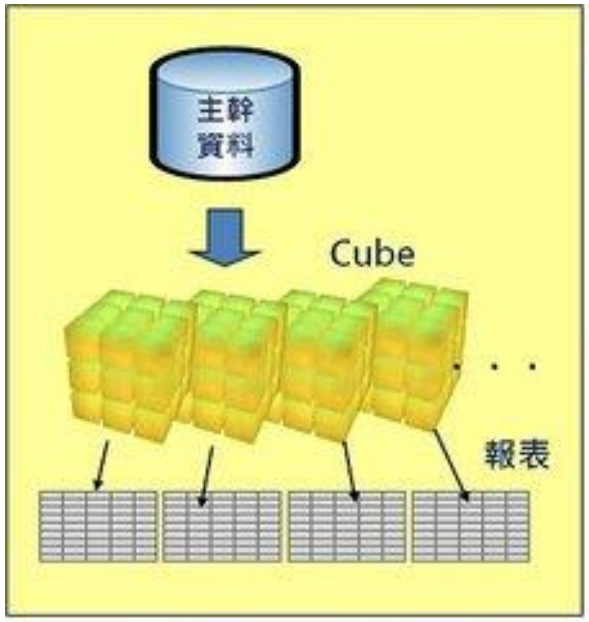
- **線上交易處理(OLTP, On-Line Transaction Processing)**
  - 利用電腦在線上直接進行**交易資料的輸入與處理作業**。
  - 使用**傳統的關聯式資料庫**，主要是**基本的、日常的交易處理**。
  - 經由資訊網路與資料庫或檔案結合，**快速處理與記錄**交易的結果，有別於傳統的批次處理。
  - 典型用在自動化的資料處理工作，其主檔案龐大、交易數量頻繁，常用於訂單輸入、銀行交易業務上，性質是**結構化且反覆性**。

- **線上分析處理(OLAP, On-Ling Analytical Processing)**
  - 讓分析師、高階管理者洞察資料的一種軟體技術。
  - 側重**決策分析**，並且提供直觀易懂視覺化的查詢結果，如數位儀表板。
  - 透過快速、一致、交談式的方式**存取多維觀點的資料**，且將原始資料轉換成為可以被使用者了解並反應企業真正構面的資訊。
  - 特性：
    - 能即時、快速的提供整合性決策資訊
    - 主要目的在支援決策資訊的分析，而非線上交易處理(OLTP) 。
    - 常需擷取非常大量的歷史資料(趨勢分析)。
    - 常需對多維度及彙整性的資訊進行複雜的分析。
    - 常需以不同時間表來比較，如月、季、年。
    - 使用者所需的資料已經事先定義並計算完成，因此查詢速度快。

	OLTP	OLAP
特色	作業性處理	資訊處理分析
導向	交易導向	分析導向
使用者	業務執行者、基層管理者、資料庫人員	知識工作者(如：高階管理者、執行長、決策人員)
功能	日常例行作業，交易活動	長期資訊分析，決策支援
資料庫設計	個體關係導向	主題、多維度導向
資料存取	由作業性資料庫存取資料，降低作業效率	不影響作業效率
資料來源	由交易產生	大多來自於資料倉儲
資料性質	即時性資料，交易明細	歷史性、彙總性資料
資料外觀	細節性、平面的關聯	彙整性、多維度
資料操作	讀寫	大多為讀取
異動頻率	資料隨時可新增、刪除與修改	歷史性彙整資料，異動機會少

OLAP的分類

- **MOLAP (Multi-dimensional OLAP)**
  - 利用特殊目的的伺服器，配合陣列式多維度儲存技術，直接對應至Data Cube陣列結構，然後依照Client端的需求，直接處理多維度資料運算並產出所要的結果。
  - 由於資料儲存於多維度的MDBS中，查詢效率較佳，但過於複雜，故資料規模限制大。



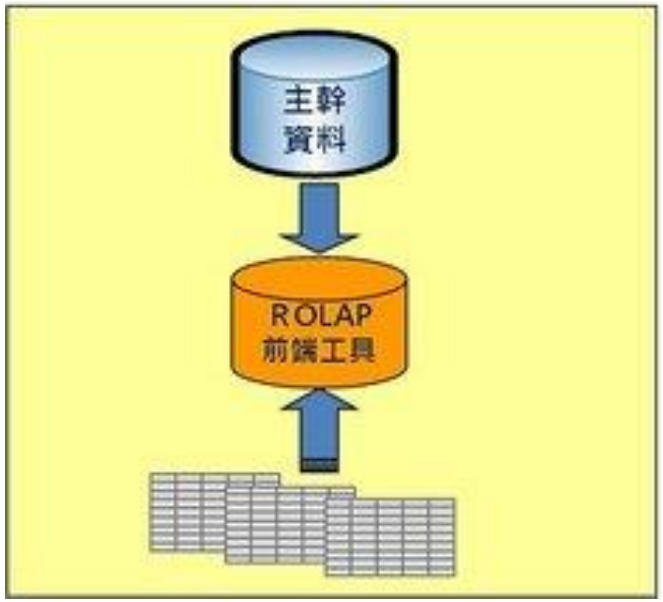
優點

- 因為在搜尋前已經做好事先準備，非常容易使用。
- 每次做搜尋時，比起在關聯式資料庫上做存取的ROLAP搜尋速度比較快。

缺點

- 因為Cube與報表有連結關係，當每次報表做變更時，Cube需要重建。
- 對於事先未能有固定的搜尋條件的特定目的之搜尋則不擅長。
- 為了產生Cube，需要花費較多的夜間批次處理時間。

- **ROLAP (Relational OLAP)**
  - **資料儲存於一般RDBS中，於RDBMS延伸功能以支援多維資料儲存與運算。ROLAP可以對Server端的關聯資料庫裡所儲存的資料做多維度資料搜尋/彙整計算，然後將結果顯示於Client端。**



優點

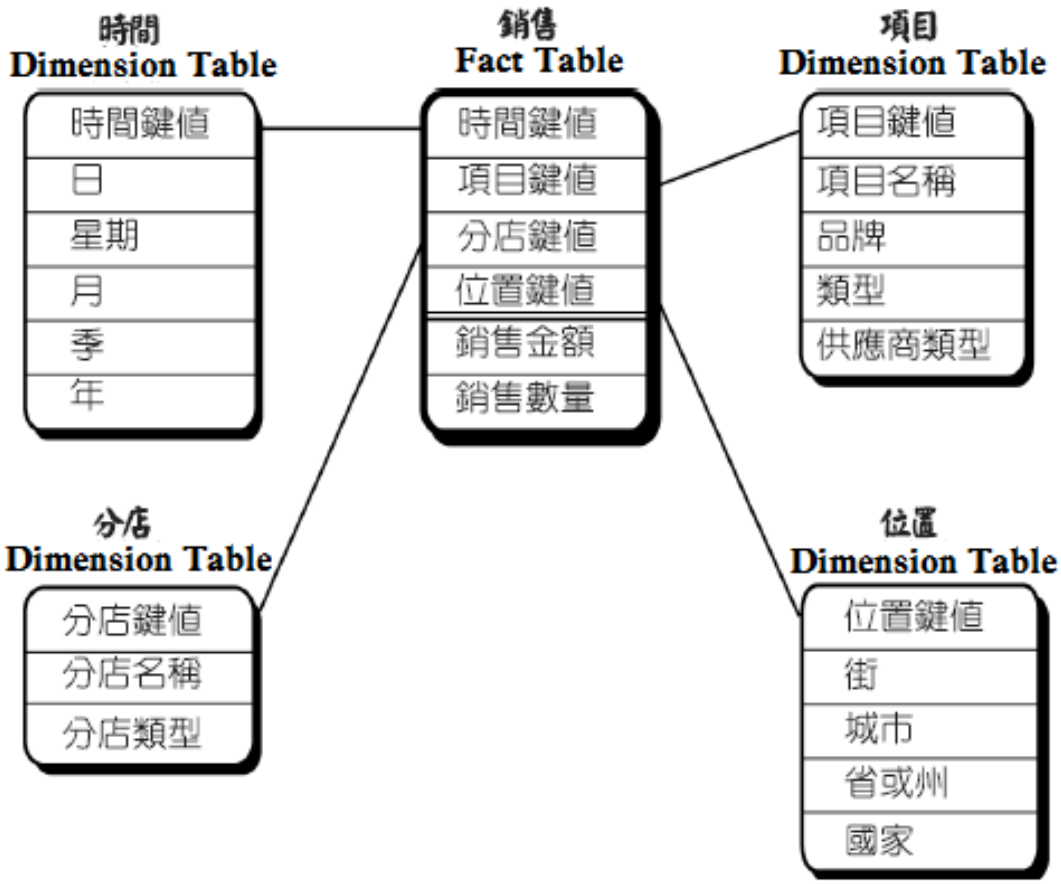
- 即使報表做變更也能夠直接利用，減少系統的維護作業。
- 能夠提供為了特別目的而臨時需要的搜尋。
- 僅就RDB做管理，因此只要少許管理工數即可，並且幾乎所有的彙總表的製作於資料載入時交由前端工具來執行，不需花費額外的彙整計算時間。

缺點

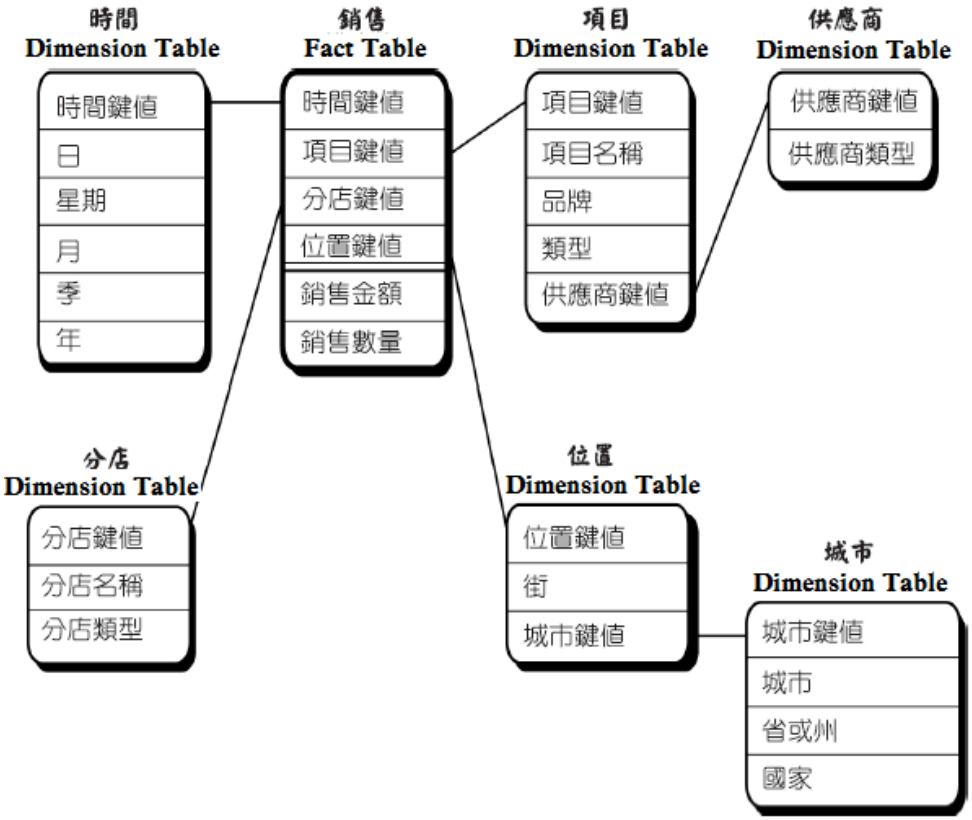
- 比起使用事先就準備好做搜尋的多維DB，需花費較多的搜尋時間。
- 對於沒有經驗者而言，要以直覺方式做資料處理門檻顯得高些。

- ROLAP將資料庫中的多維度結構劃分為兩類表格：
  - Fact Table(事實表格)
    - 儲存大量資料的大型中心表格 (Large Central Table)。存放以交易為中心、經量化後的數值資料。由於所存放的是為了達成企業所關心之主題資料，所以是資料分析的主角。
    - 事實表格是資料庫中處理得最頻繁的資料表格，所以事實資料表有時可以龐大到占據超過1TB的空間。 也由於表格資料過於龐大，為了效率考量，通常不會對它做正規化。
    - 用來儲存交易資料和維度表格的鍵值(Key Value)
  - Dimension Table(維度表格)
    - 維度表格所包含的資訊，可以幫助企業由事實表格中，取得決策者於該維度中有興趣的事實資料。
    - 簡單來說，維度表格是用來記載事實表格中所包含之資料的意義。
  - 維度表格通常很小，所以資料倉儲一般只會有少數幾個事實表格，卻會有很多個維度表格。

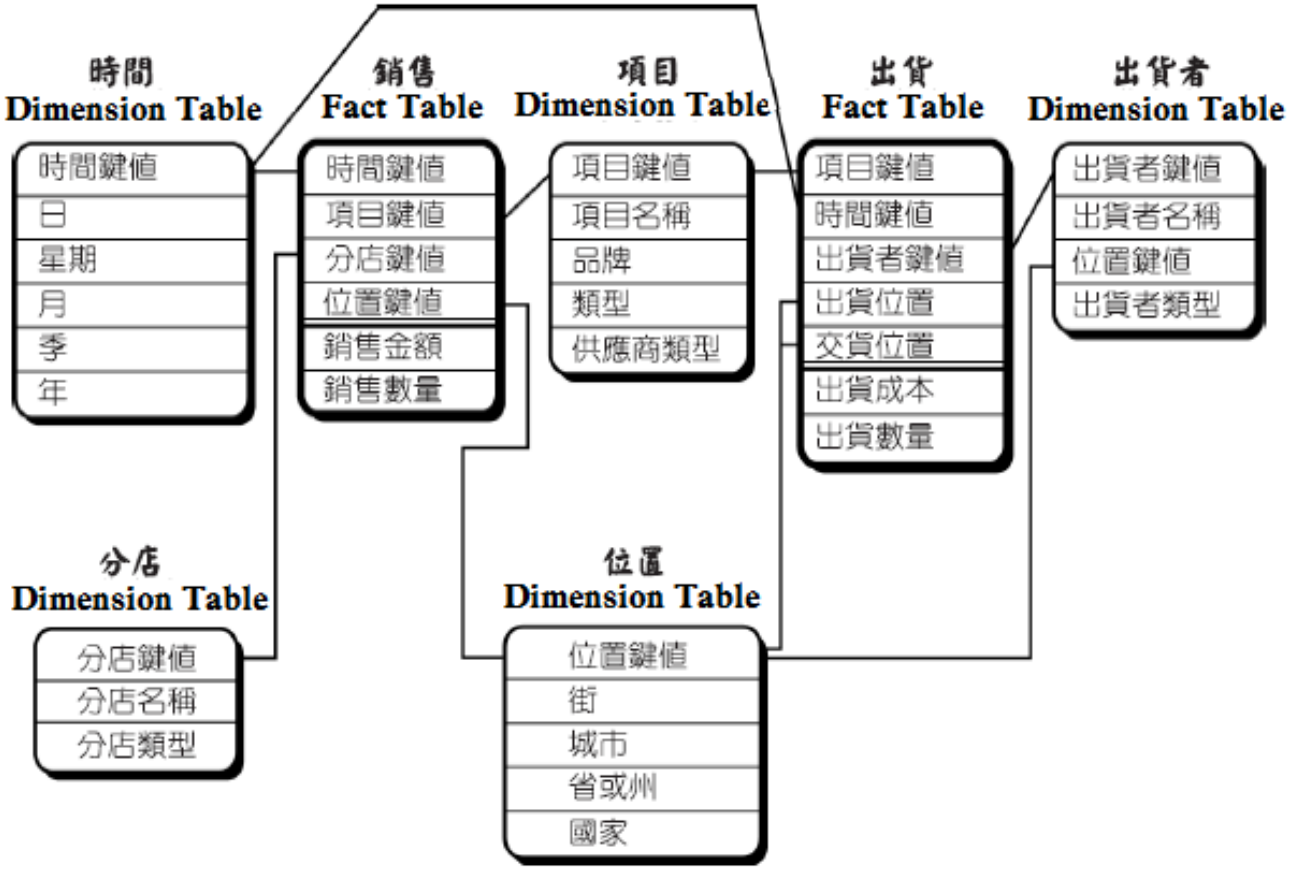
- 維度表格和事實表格之間，利用**鍵值 (Key Value)** 連繫在一起，形成了 “**星型模式 (Star Schema)**”。



- 由於維度表格內容較少，故對於**層次較複雜的維度表格**，可進一步從事**正規化**，**使用多個相關的維度表格**來描述此一維度。這種星型模式的擴展稱為“**雪花模式 (Snowflake Schema)**”。



- 若有多個事實表格共享部份維度表格，即為“事實星座模式 (Fact Constellation Schema)”。這種模式可視為星型模式的組合。



- **HOLAP (Hybrid OLAP)**
  - 結合MOLAP與ROLAP的特點。例如：低層的大量明細資料儲存於關聯式資料庫，高層的彙總資料儲存於個別的MOLAP資料庫。
  - 具有靈活性。查詢效率比ROLAP高，硬碟耗用比MOLAP低。

三種OLAP的比較

	MOLAP	ROLAP	HOLAP
特色	• Cube的內部資料與彙總資料儲存於多維度資料庫(MDBS) • 事先做彙總運算並寫入Cube內	• Cube的內部資料儲存於關聯式資料庫(RDBS) • 不事先做彙總運算	• Cube的內部資料儲存於關聯式資料庫(RDBS)，彙總資料則儲存於多維度資料庫(MDBS) • 事先做彙總運算並寫入Cube內。
優點	• 查詢效率最佳 • 可處理複雜運算功能	• 建置成本低 • 允許大範圍隨意查詢	• 建置成本與查詢效能介於MOLAP與ROLAP之間
缺點	• 耗費大量儲存容量 • 建置成本高	• 查詢效能較差 • 維護成本高	• 系統複雜度較高 • 有User數限制
代表產品	• Cognos • Hyperion Essbase • Oracle Express	• Business Objects • Micro Strategy	• Microsoft Analyzer

資料探勘(Data Mining)

- 定義：
  - 資料探勘是從大量資料中提取(Extract)或探勘(Mining)出知識的一種方法。
  - 以自動或半自動的方式對大量資料作分析，以找出有意義的關係或規則。
- 資料探勘與傳統統計方法
  - 資料探勘與傳統統計方法間具有關聯性。
  - 資料探勘必須對分析資料之屬性定義清楚，解決問題之目標需明確，提供分析演算法包括統計分析、人工智慧、決策樹、類神經網路與其它演算法，包括了定量與定性分析，可以找出多個變數間的相關性。
  - 統計方法只採用統計分析，分析者必須逐一分析變數，才能建立模式，且只用到定量分析。

- 資料探勘的技術：
  - 關聯規則(Association Rule)
    - 消費者所購買的產品間是否具有特定的關聯
    - Ex: 啤酒⇨尿布
  - 分類(Classification)
    - 依照特定條件將資料分類，可判斷資料是否隱含此族群特性之規則。
    - Ex: 會購買RV房車的族群特性
  - 聚類(Clustering)
    - 將有相同或近似特性之資料分成同一群。
    - Ex: 將公司現有客戶資料分群
  - 區別(Discrimination)
    - 比較分析族群間之差異性。
    - Ex: 年紀大且收入高的客戶，與年紀輕且收入高的客戶間的買車差異
  - 演進(Evolution)
    - 分析一段時間內資料之變化與趨勢。
    - Ex: 今天買了照相機的客戶，要過多少天才會再來買其它相關產品

分類問題 vs. 聚類問題

- **分類：**

編號	性別	年齡	婚姻	家庭所得	購買RV房車
A0001	Male	<35	未婚	高所得	否
A0002	Male	<35	未婚	小康	否
A0003	Female	≧35	已婚	高所得	是
A0004	Male	≧35	未婚	低所得	是
A0005	Female	≧35	已婚	高所得	否
A0006	Male	≧35	已婚	低所得	否
A0007	Female	≧35	未婚	小康	否
A0008	Male	≧35	已婚	高所得	是
A0009	Male	<35	已婚	低所得	是

類別欄位

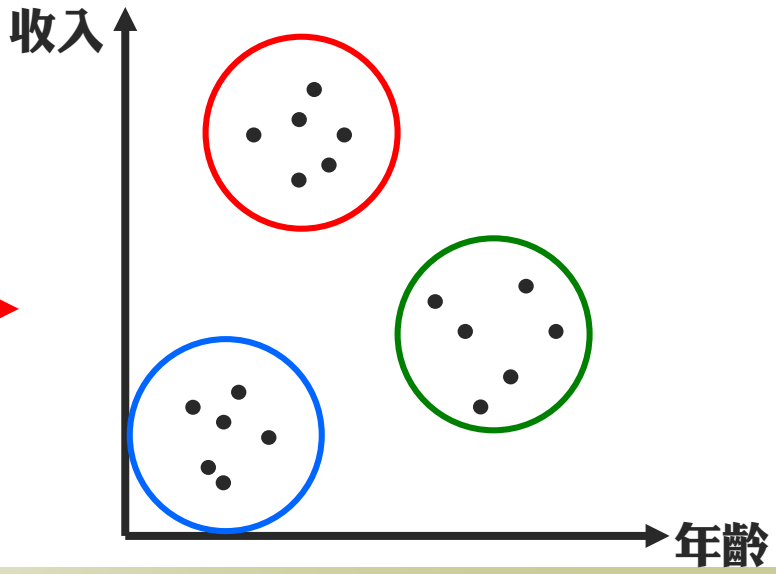
if 婚姻 = 未婚 and 年齡 < 35
then 購買RV房車 = 否

⋮

if 婚姻 = 已婚 and 家庭所得 = 高所得
then 購買RV房車 = 是

- **聚類：**

編號	性別	年齡	婚姻	家庭所得
A0001	Male	<35	未婚	高所得
A0002	Male	<35	未婚	小康
A0003	Female	≧35	已婚	高所得
A0004	Male	≧35	未婚	低所得
A0005	Female	≧35	已婚	高所得
A0006	Male	≧35	已婚	低所得
A0007	Female	≧35	未婚	小康
A0008	Male	≧35	已婚	高所得
A0009	Male	<35	已婚	低所得



UML(Unified Modeling Language)

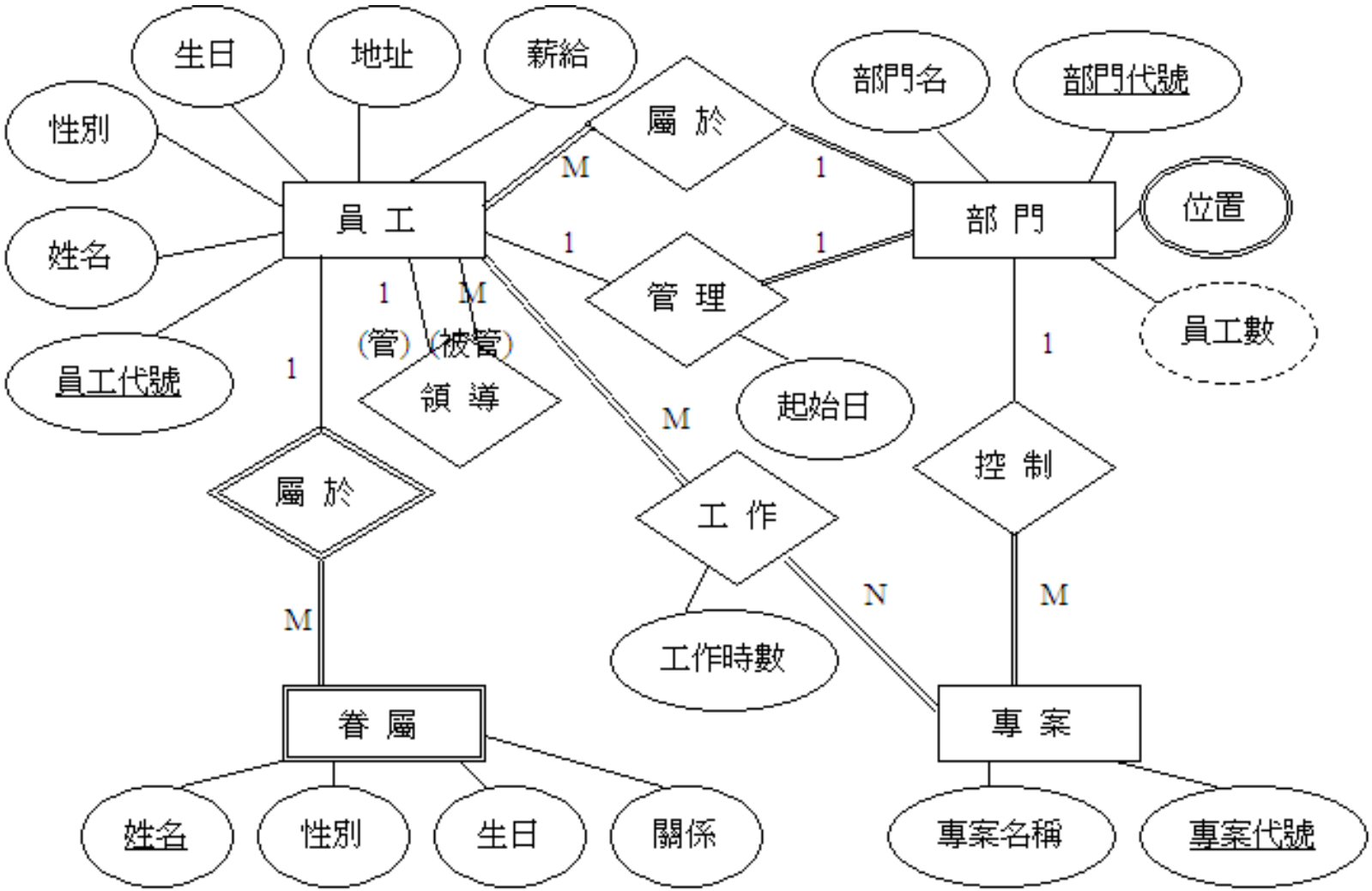
- UML為將物件導向概念模式化的技術，可用於**物件導向之軟體系統分析與設計**。
- 可採用下列幾種不同的工具(圖形)來表示：
  - **類別圖(Class Diagram)**
  - 物件圖(Object Diagram)
  - 使用案例圖(Use Case Diagram)
  - 活動圖(Activity Diagram)
  - 循序圖(Sequence Diagram)
  - 狀態圖(State Diagram)
  - 合作圖(Collaboration Diagram)
  - 元件圖(Component Diagram)
  - 開發圖 (Deployment Diagram)

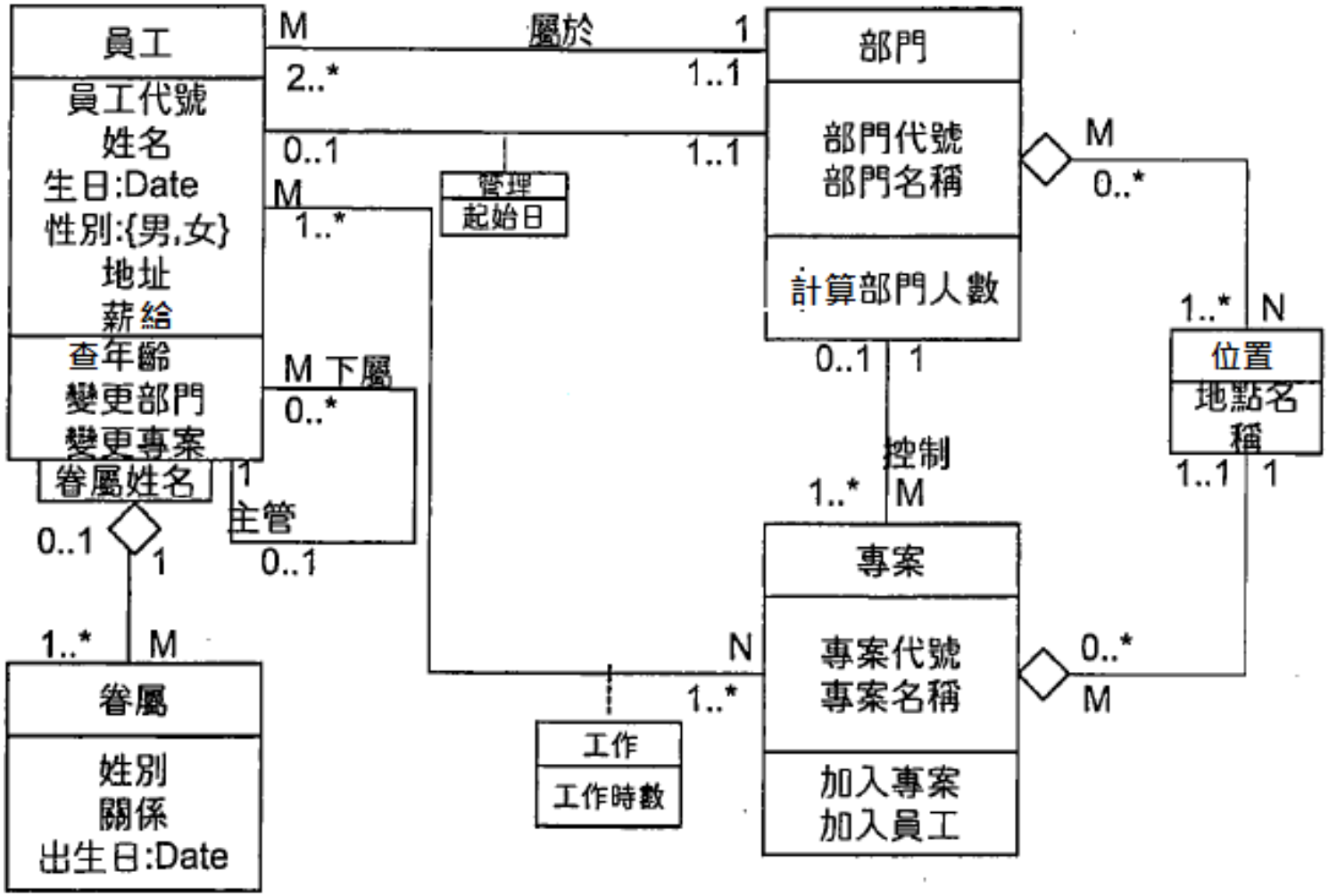
- 類別(Class)以一個方格表示，含三個部份：
  - Top Section：Class Name
  - Middle Section：類別中，個別物件的屬性
  - Last Section：這些物件的相關Operations

- UML有兩種關聯：Association及Aggregation
  - Association：物件間關聯(Relationship)類型
  - Aggregation：用來表示整個物件和它某些組成部份的關係

- **ER Model轉換為O-O Data Model之UML流程：**
  1. **對每個正常個體 (Normal Entity) 產生一個新的物件型態 (Object Type)**
  2. **對於弱個體 (Weak Entity) 在擁有者個體產生方框，並產生一個新的物件型態與聚合關係 (Aggregation; ◇—)**
  3. **針對二元 1:1，若關係上有屬性存在，產生新物件型態。**
  4. **針對二元 1:M，若關係上有屬性存在，產生新物件型態。**
  5. **針對二元 N:M，若關係上有屬性存在，產生新物件型態。**
  6. **針對多值屬性，產生新物件型態及聚合關係。此新物件型態為衍生類別，可與其它類別共享。**

舉例


生日
地址
薪給
性別
姓名
員工代號
員工
屬於
M
1
部門名
部門代號
部門
位置
員工數
1
1
1
M
(管)
(被管)
管理
領導
起始日
M
1
屬於
M
工作
工作時數
N
控制
M
眷屬
姓名
性別
生日
關係
專案
專案名稱
專案代號


員工
員工代號
姓名
生日:Date
性別:{男,女}
地址
薪給
查年齡
變更部門
變更專案
眷屬姓名
眷屬
姓別
關係
出生日:Date
屬於
M
2..*
1
1..1
0..1
1..1
管理
起始日
M
1..*
M 下屬
0..*
1
主管
0..1
0..1
1
1..*
M
部門
部門代號
部門名稱
計算部門人數
M
0..*
1..*
N
位置
地點名稱
1..1
1
0..1
1
控制
1..*
M
專案
專案代號
專案名稱
加入專案
加入員工
N
1..*
工作
工作時數
0..*
M

補 充

關聯式代數的通用轉換規則

- 以下的轉換規則可讓關聯式代數從事對等轉換：
  1. Cascade of σ: A conjunctive selection condition can be broken up into a cascade (sequence) of individual s operations:
     - $\sigma_{c1\ AND\ c2\ AND\ ...\ AND\ cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$
  2. Commutativity of σ: The σ operation is commutative:
     - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
  3. Cascade of π: In a cascade (sequence) of π operations, all but the last one can be ignored:
     - $\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) \equiv \pi_{List1}(R)$
  4. Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
     - $\pi_{A1, A2, ..., An}(\sigma_{c}(R)) \equiv \sigma_{c}(\pi_{A1, A2, ..., An}(R))$

5. Commutativity of $\bowtie$ (and $\times$): The $\bowtie$ operation is commutative as is the $\times$ operation:
   - $R \bowtie_C S \equiv S \bowtie_C R$; $R \times S \equiv S \times R$
6. Commuting σ with $\bowtie$ (or $\times$): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:
   - $\sigma_c (R \bowtie S) \equiv (\sigma_c (R)) \bowtie S$

   Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:
   - $\sigma_c (R \bowtie S) \equiv (\sigma_{c1} (R)) \bowtie (\sigma_{c2} (S))$

7. Commuting π with $\bowtie$ (or $\times$): Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:
   - $\pi_L (R \bowtie_C S) \equiv (\pi_{A1, ..., An} (R)) \bowtie_C (\pi_{B1, ..., Bm} (S))$

   If the join condition C contains additional attributes not in L, these must be added to the projection list, and a final π operation is needed.
8. Commutativity of set operations: The set operations $\cup$ and $\cap$ are commutative but "–" is not.
9. Associativity of $\bowtie$, $\times$, $\cup$, and $\cap$: These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have
   - $(R \theta S) \theta T \equiv R \theta (S \theta T)$

10. Commuting σ with set operations: The σ operation commutes with $\cup$, $\cap$, and $-$. If θ stands for any one of these three operations, we have

   - $\sigma_c(R \ \theta \ S) \equiv (\sigma_c(R)) \ \theta \ (\sigma_c(S))$

11. The π operation commutes with $\cup$.

   - $\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$

12. Converting a (σ, $\times$) sequence into $\bowtie$: If the condition c of a σ that follows a $\times$ Corresponds to a join condition, convert the (σ, $\times$) sequence into a $\bowtie$ as follows:

   - $(\sigma_C(R \times S)) \equiv (R \bowtie_C S)$

13. Other transformations.