

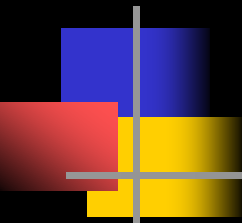
第十二章 物件導向式 資料庫系統





本章內容

- 12.1 前言
- 12.2 物件導向的概念
- 12.3 各種物件導向應用的發展
- 12.4 物件導向的優點
- 12.5 物件導向式資料庫管理系統簡介
- 12.6 物件導向式資料模式的資料結構
- 12.7 物件導向式資料庫的資料定義語言
- 12.8 物件導向式資料庫的資料處理語言
- 12.9 結論



前言

- 關聯式資料模式只適合應用在傳統對文字資料處理、運算與交換之商業應用上。
- 當代的系統則漸趨向於下列應用：
 - 多媒體的辦公室自動化環境
 - 電腦在輔助設計上的應用
 - 地理資訊系統
 - 知識庫系統、專家系統與決策支援系統
 - 電腦輔助軟體工程上的 CASE tools
 - 複雜的工程應用



複雜應用共同的需求與特性

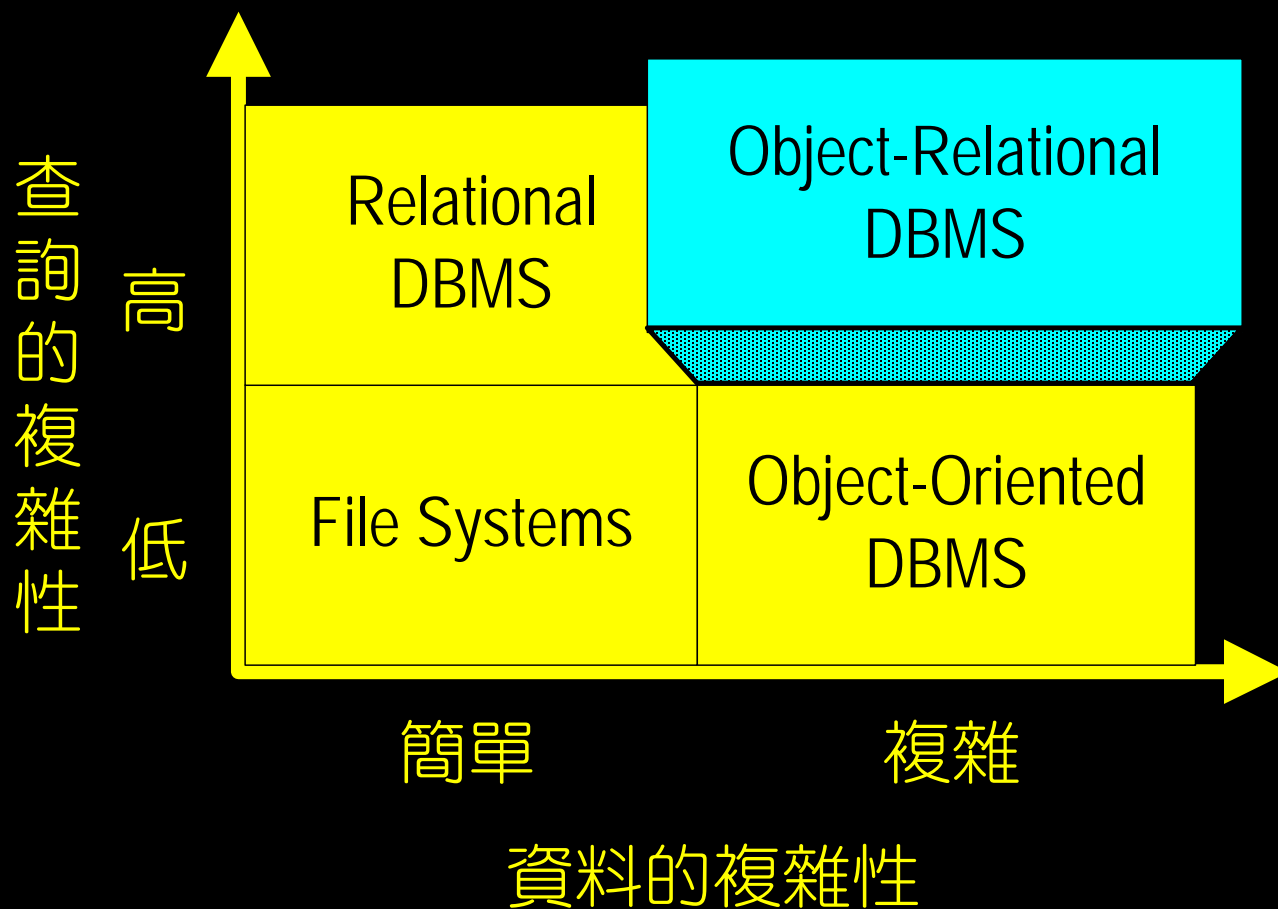
- 具有層層包覆的複雜結構關係
- 需要有多重值的屬性
- 資料具縱向分類 (Generalization/Specialization) 、橫向組合 (Aggregation) 與繼承 (Inheritance)
- 具時間 (Temporal) 與空間 (Spatial) 上的屬性與對應關係
- 具有同一文件，不同版本的需求 (Multi-Version Control)

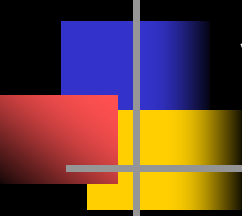


複雜應用 vs. 關聯式模式

- 上述需求與特性在關聯式資料庫中無法直接支援 (如：第一正規化型式)。
- 目前關聯式資料庫系統在大型物件的支援上，大部份是用所謂的 Binary Large Objects (BLOBs) 來儲存，需要使用者自行做處理，在查詢的處理與最佳化效能上很不理想。

資料庫管理系統的分類矩陣





1. 簡易型態及低複雜查詢的應用

- 文字編輯程式 (Text Editor) 是此類典型的應用：
 - 不需用到 SQL。
 - 唯一會用到的“查詢”動作是“開啟檔案”，
 - 唯一的“更新”動作是“寫回檔案”。
 - 很適合以檔案系統所提供的資料模式來完成，
 - 適合用在左下角象限中的資料庫管理系統，便是由作業系統 (Operating Systems) 下的檔案系統 (File Systems) 來擔任這個角色。



2. 處理簡易型態但做高複雜查詢

- 關聯式資料庫管理系統適合用來儲存商業應用資料
 - 透過 SQL 進行相當複雜的查詢。
 - 能處理的資料僅包含：整數 (Integers)、實數 (Floats)、日期 (Dates)、字串 (Character Strings) 等簡易資料型態
 - 雖然目前大多有提供 BLOB (Binary Large Objects) 或 CLOB (Character Large Object) 的型態，但使用者都要進行特殊處理
 - 這類使用 SQL-92 做商業資料儲存與查詢的應用就歸類到第二象限。



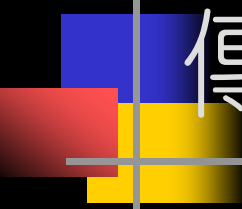
第二象限的基本需求

- 具備 SQL-89 或 SQL-92 的查詢語言能力。
- 需要前端開發工具：讓程式設計師設定表單 (Forms) 或產生報表，供使用者來輸入、顯示資料或編輯成為報表。
- 高效率的執行效能：透過“線上異動處理” (On-Line Transaction Processing) 執行「任務導向」 (Mission-Critical) 的應用。
- 錯誤的容忍度 (Fault Tolerance)：不論發生何種錯誤，都要確保使用者資料不會遺失。
- 保密性 (Security)：要確保使用者的資料不會被篡改或遭竊取。



3. 處理複雜型態但做簡單查詢

- 如：電腦輔助設計 (Electronic Computer-Aided Design, ECAD) 的應用
- 層層包覆的複雜特性，設計結果均是以複雜物件的型式存放在磁碟中，
- 程式會先將它們讀到主記憶體中、以最佳化的程序來求得結果，然後再存回磁碟中。
- 這類應用可以歸類到右下角象限的類別中



傳統檔案系統無法滿足此種需求

- 因為應用程式必須要讀取複雜物件中的層層資訊。
- 還要將資料從磁碟機中轉換到主記憶體中。
- 資料在磁碟中以一組 (Oid) 來代替，在載入到記憶體時，會被轉換成虛擬記憶體的指標。
- 由於虛擬記憶體指標是暫時的，因此當該程式後續被重新執行時，便不能繼續使用原來的指標。
- 當資料寫回磁碟機時，互相包覆的物件資訊，可能因為重新配置的動作，而導致必須進行記憶體到磁碟間的轉換。



解決方式: Persistent Type

- 在程式語言開發工具中加入「永久式資料型態」(Persistent Data Type) 的功能。
- 永久式資料型態的變數值在程式執行完後，會被系統存放到硬碟的某處，
- 下次執行時能夠再度取出應用。
- 傳統的變數宣告：`integer i;`
- 永久式資料型態的宣告：**Persistent** `integer i;`



解決方式: Persistent Type

```
main()
{
// 讀入所有的變數資料； //
// 執行所欲完成的工作；
// 將變數資料寫回檔案或資料庫；
}
```

```
main()
{
// 執行所欲完成的工作；
}
```

簡化成

輕量級 vs. 重量級運算

- 永久式程式語言 vs. 關聯式資料庫管理系統：
 - 前者的更新動作只是“輕量級” (Lightweight) 動作
 - 後者的更新動作是“重量級” (Heavy weight) 動作
- 第三象限不太需要 SQL 查詢語言，而是複雜物件的儲存與管理功能，以及執行效能。

$j = j + 1;$

輕量級

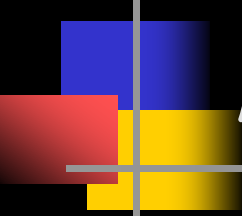
```
update Orders
set quantity = 0
where no in (select no from Bookstores
             where rank = 30)
```

-- 重量級



處理複雜資料型態也做複雜查詢

- 物件化的需求越來越高、多媒體電腦化趨勢、全球資訊網 (World Wide Web) 與資料庫的結合
 - 讓資料庫中所存放的可能包含影像 (Images)、聲音、網頁、電子文件、甚至於跨平臺的應用程式 (如：Java Applet) 等。
 - 應用需要以影像、聲音、網頁、電子文件的內容做為擷取條件 (Content-Based Retrieval) :
 - 找出含有晚霞向晚天的圖片、
 - 介於某個頻率區間的聲音、
 - 包含某段新聞稿的錄影畫面等。



處理複雜資料型態也做複雜查詢

- 從影像資料找出有晚霞向晚天的圖片，其中 SUNSET 函數是看看圖片頂端是否有橙黃色成份，其 SQL 指令如下：
select *id*
from pictures P
where **SUNSET(P.image)**
- 衍生出第四象限的需求



物件關聯式資料庫管理系統

- 具有這些能力的標準 SQL 版本是 SQL-3。
- 只提供 SQL-92 的系統，沒有能力完成此類應用工作。
- 傳統存取方法 (像 B⁺-tree 以及 hashing) 也沒有任何幫助。
- 我們需要二度空間的存取方法，像：grid file、R-tree，R⁺-Tree，以及各種類型的 Quadtree 或 K-D-B tree
- 物件關聯式資料庫管理系統
 - 具備以“物件”為主的存取方法，
 - 允許使用者或系統整合程式將存取方法加入到系統中。
 - SQL Server 2008 提供與 Common Language Runtime (CLR) 整合，允許任何 .NET 語言所開發的資料庫物件，以 EXE 或 DLL 型式儲存，並透過「組合」(Assembly) 建構機制，延伸資料庫的功能。



關聯式 vs. 物件導向式

- 資料庫管理系統已盤據了 90% 商用市場
- 即使產品都漸漸朝向物件化，還是必須提供原屬於關聯式資料庫管理系統的功能
- 這讓物件導向式資料庫管理系統與物件關聯式資料庫管理系統之間的分野已經越來越小
- 預料將來兩者應該會朝整合的路來走。
- 在本章中，我們將統一以「物件導向式資料庫管理系統」來稱呼這兩類系統。

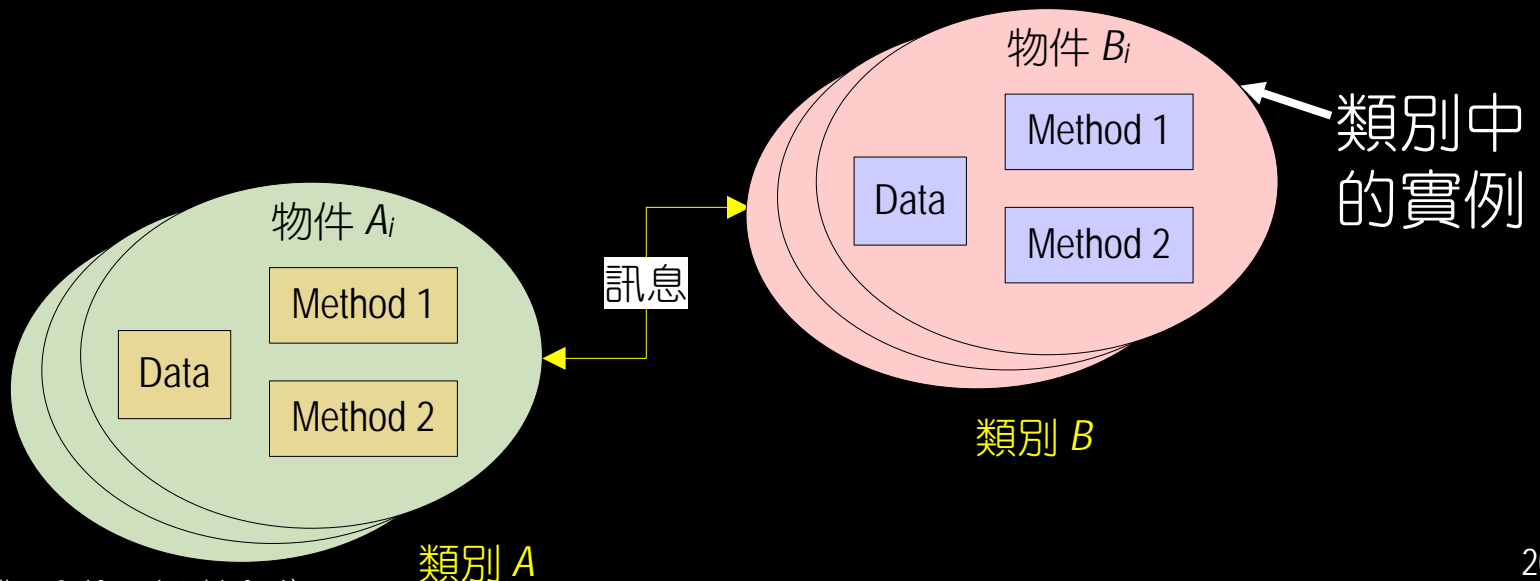


物件導向的概念

- **傳統作法**：功能導向 (Function-Oriented) — 考慮需要那些功能，然後再考慮該功能需要那些資料配合。
問題：一旦資料型態改變，用到該資料的函式都得跟著修改。
- **物件導向**：讓「功能」跟著「資料」走。
 - 如：滑鼠有「拖曳」功能，所以讓「拖曳」功能附屬在「滑鼠」上，
 - 以「物件」為主角來設計專屬的「功能」函式，稱為「物件導向」(Object-Oriented)。
 - 「抽象資料型態」(Abstract Data Types)：將資料與其運算方法結合在一起，以隱藏其內部結構的資料型態稱作。
 - 使每個物件具有自己的資料與運算方法，讓物件的資料僅能透過物件的運算方法存取，保障資料的正確性。

物件導向基本元件

- **物件**：由一組資料 (可以是基本或衍生的物件) 與「運算方法」(Methods) 組成。
- **類別**：具相同屬性以及運算方法的物件集合
- **訊息**：由一個物件傳送「訊息」給另一個物件，要求對方執行某個運算方法，完成物件的互動。

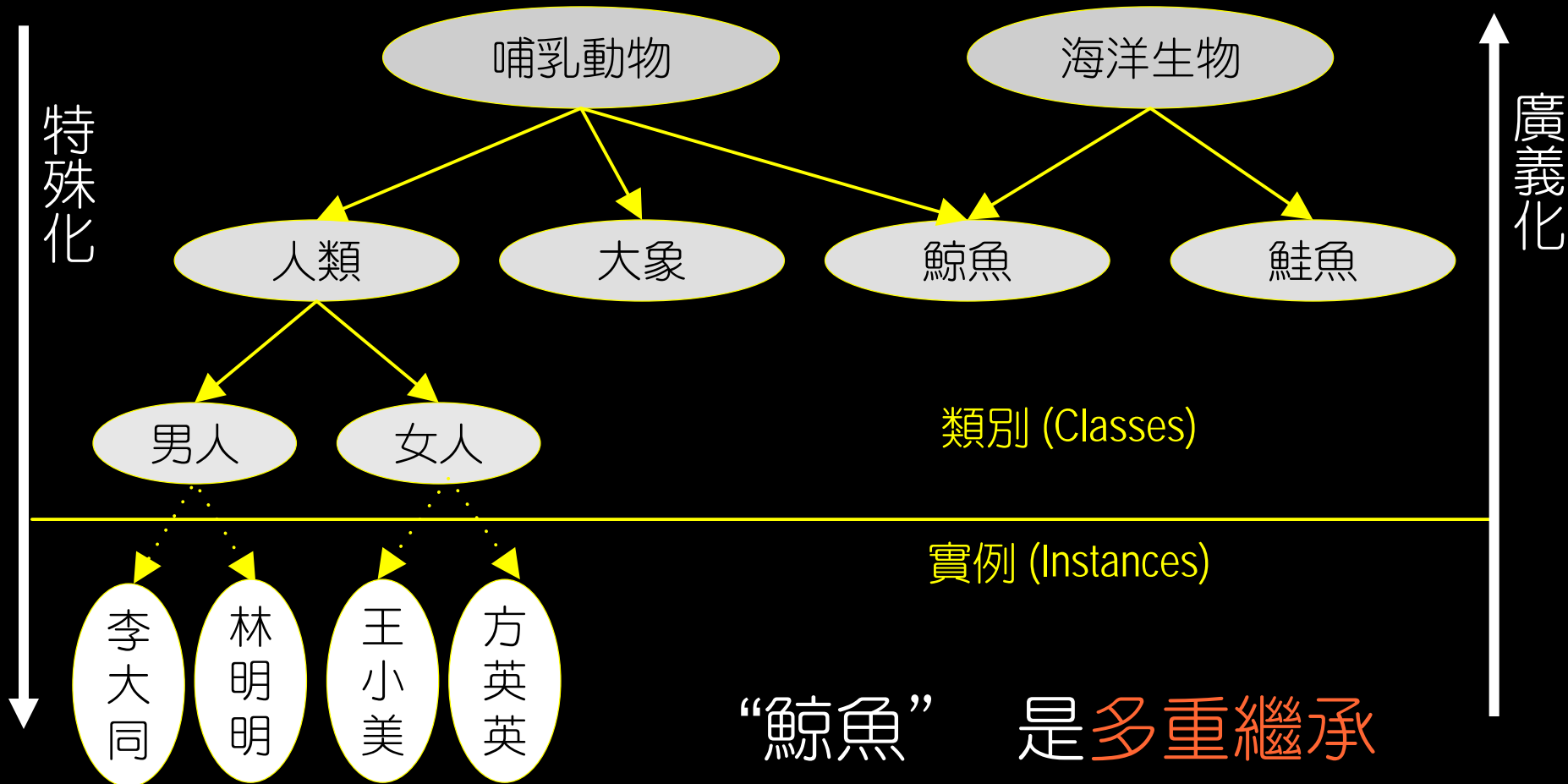




物件導向的技術

- **封裝**：抽象資料型態的整體封包與資訊隱藏
- **繼承** (Inheritance)：從既存母類別 (Superclass) 衍生子類別 (Subclass)，並讓子類別繼承母類別的屬性與操作方法，屬於**縱向關係**。
- **聚合** (Aggregation)：將某些物件組成更複雜的物件，屬於**橫向關係**。
- **多態性** (Polymorphism) — 相同的訊息，可以傳給不同類型的物件，產生不同的回應。例如 “列印” 訊息可傳給字串類別，則由印表機列印結果；若傳給 “聲音” 物件，則由喇叭播放。

繼承 (Inheritance)



聚合 (Aggregation)

- 將某些物件組成更複雜的物件，產生層層包覆的組合關係



包含

引擎
車體
電氣系統
輪胎

...



多態性 (Polymorphism)

- 允許相同的訊息 (Message) 可以送給不同類別的物件
- 每一種物件則依其類別的特性做適當的回應
- 例如：列印 (Print) 這個訊息可以送給整數、字元、實數、日期等類別的物件
- “加法” 訊息若傳給矩陣物件時，則回應矩陣的加法結果；若傳給複數物件，則回應複數的加法結果。
- 「多態性」讓類別的設計可以達成單一介面、多種使用功能的目的。



物件導向式觀念的發展

- 物件導向程式語言的發展
 - Simula-67
 - Smalltalk (1981 年由 Xerox Palo Alto Lab 所發展)
 - Objective C (1983 年由 StepStone Corp. 所發展)
 - Eiffel (1987 由 Interactive Software Eng. Inc.)
 - Object Pascal (1986 由 Apple Computer 所發展)
 - C++ (1986) — Bjarne Stroustrup
- 物件導向方法論的發展
- 物件導向式資料模式的發展

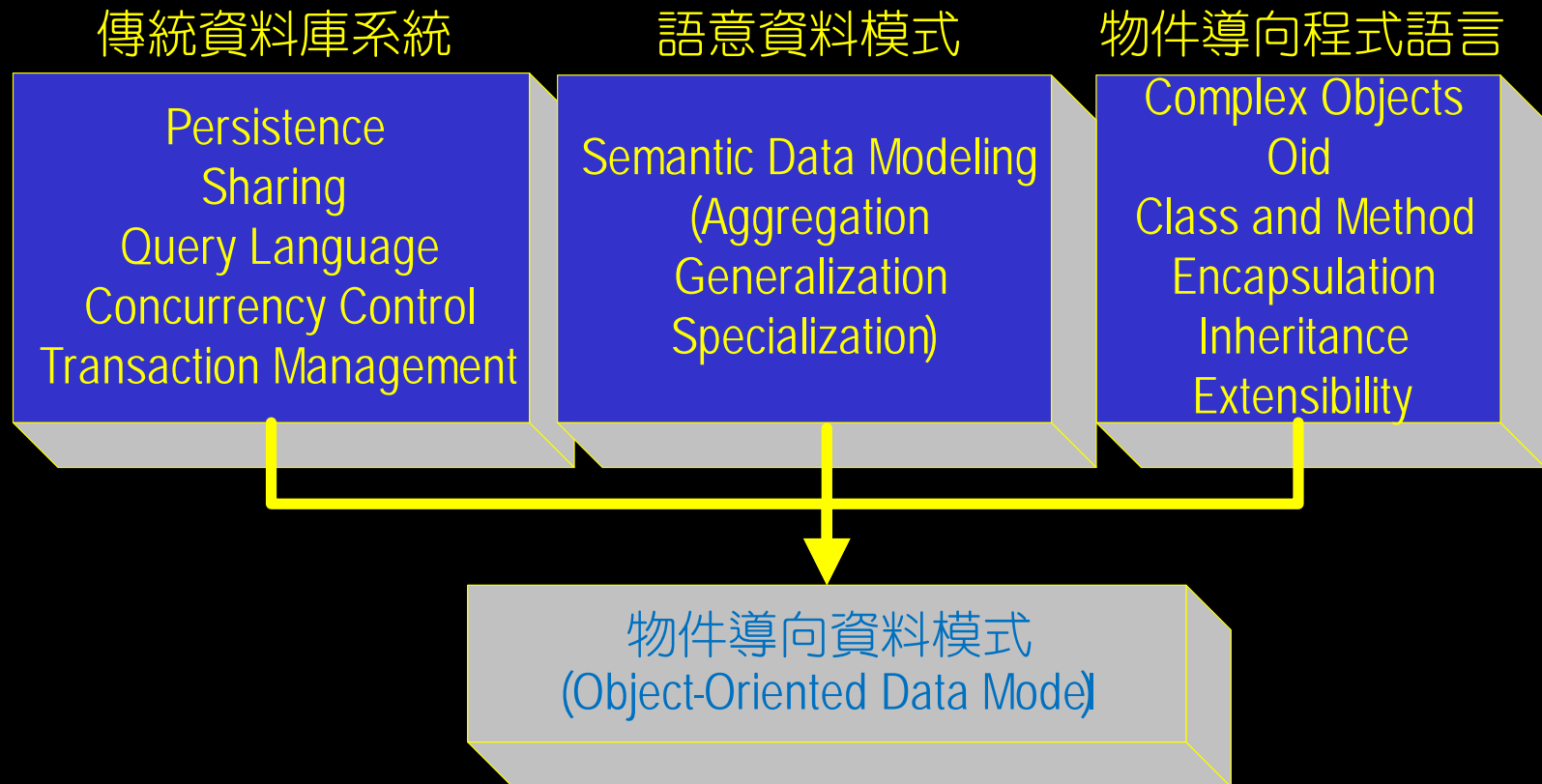


物件導向方法論的發展

- 「物件導向式系統分析」(OOA)
- 「物件導向式系統設計」(OOD)
- 「物件導向式程式設計」(OOP)
- 物件導向分析與設計的方法論 (已統一為 UML)
 - Grady Booch 的物件導向式分析設計方法論
 - James Rumbaugh 的物件模型技術 OMT
 - Ivar Jacobson 的 Use Case Driven Approach
 - Joseph Fong 探討了如何將 E²/R 轉換成 OMT 模式

物件導向式資料模式的發展

- E-R Model → Semantic Data Model → DAPLEX → Extended E-R Model → Object-Oriented Model





物件導向觀念的優點

- 反應實際世界所代表的意義。
- 軟體的可靠度(Software Reliability) 與模組化(Modularity) 結構。
- Extendibility
- 多態性—Generic Software Components, Code Reusability)。
- 繼承機制提高軟體元件的擴充性(Extensibility)。



物件導向式資料庫系統簡介

- 將資料庫以一組抽象化資料型態 (a Collection of Abstract Data Types) 表示，
- 抽象化資料型態各有其所屬的運算 (Operations)，以及屬於其該型態中的實例 (Instances)。
- 實例都得透過定義於其類別上的運算來存取。
- 類別之間有「縱向」繼承 (Generalization and Specialization)、「橫向」組合 (Aggregation) 與「繼承關係」(Inheritance)
- 物件之間透過「訊息」(Message) 溝通彼此。



物件導向資料庫系統上的概念

- 物件 (Objects)、複合物件 (Complex Objects，或稱 Composite Objects)、類別 (Classes)、訊息 (Messages) 的觀念與前述相同。
- 屬性 (Attributes)：
 - 「基本屬性」最基本的字元、整數、字串等，
 - 「複合屬性」由各種類型的屬性所構成，
 - 「集合屬性」包含許多基本屬性元素的集合，
 - 「複合集合屬性」許多複合屬性元素的集合。
- 繼承 (Inheritance)：有單一與多重繼承
- 聚合關係 (Aggregation)：將異質性的物件組合成起來，以抽象化的方式呈現。城市、街道名稱、里、鄰、巷、弄、號、樓，組合起來稱為“地址”

關聯式 vs. 物件導向資料模式

物件導向資料模式 (Object-Oriented Data Model)	關聯式資料模式 (Relational Model)
類別 (Class)	關聯表 (Relation)
物件 (Object)	值組 (Tuple)
物件識別碼 (Oid—Object Id)	主鍵加上關聯表名稱 (Primary Key and Relation Name)
基本屬性 (Primitive Attribute)	屬性 (Attribute)
複合屬性 (Complex Attribute)	以額外的關聯表與 外來鍵的連結關係來表示
集合屬性 (Set Attribute)	
複合集合屬性 (Complex Set Attribute)	
母、子類別的關係 (IS-A Relationship)	
聚合關係 (Aggregation Relationship)	無法對應
包裝 (Encapsulation)	
繼承 (Inheritance)	無法對應
方法 (Method)	預儲程序 (Stored Procedures)



物件導向資料庫系統實現方式

- 可以歸納成為以下三種：
 - 以關聯式資料庫系統為主，上面架一層物件導向的觀念一如：PostgreSQL、Starburst
 - 以物件導向程式語言為主，擴充加上資料庫的處理功能一如：GemStone、ObjectStore、Versant、Objectivity，與 O_2
 - 直接從無到有建立整個物件導向資料庫系統一如：ORION、UniSQL、OpenODB 等。



物件導向資料庫管理系統標準制定

- Object Database Management Group (ODMG)—由物件資料庫廠商所組成的聯盟體制，
- 1992 年成立，目標在訂定物件導向式資料庫管理系統的各项相關標準。
- 1993/10 公佈 ODMG-93，定義資料模式、查詢語言。
- ODMG-93 隨後為物件導向式資料庫管理系統廠商採用
- 相關資料 <http://www.odmg.org/>
- ANSI Object SQL (或稱 ANSI SQL3 委員會)
- 標準制定涵蓋面較廣，包含了各式各樣的資料庫規則與觸發條件等，進度較慢
- 目前已公佈的標準有 SQL:1998 與 SQL:2003
- 其中被 SQL Server 2008 所實現的大部份功能都已經在 7.10 節中說明了
- 後續應該還會陸續增加其他的標準。



著名的物件導向資料庫管理系統

- Alltalk—由 Eastman Kodak 公司所發展。
- GemStone—由 Servio Corp. 發展。
(<http://www.gemstone.com>)
- GENESIS—由 Univ. of Texas at Austin 發展
- IRIS—由 Hewlett-Packard Lab. 所發展。
- Itasca—由 Itasca Systems 所發展。
- Object Database—由 Object Database, Inc. 所發展
- Objectivity/DB—由 Objectivity, Inc. 公司所發展
- ObjectStore—由 Object Design Inc. 所發展

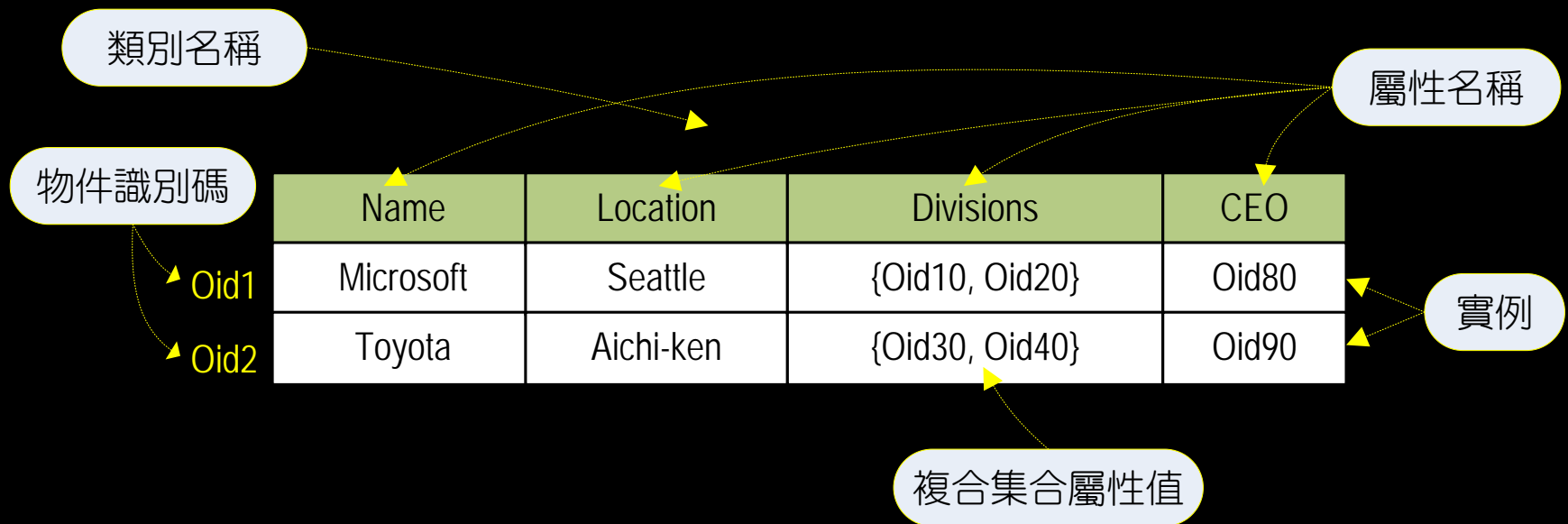


著名的物件導向資料庫管理系統

- ONTOS—由 ONTOS 公司所發展
- ORION—由 MCC 公司所發展
- PostgreSQL—由加州大學柏克萊分校 (University of California at Berkeley) M. Stonebraker 博士發展，目前屬於開放原始碼軟體 (<http://www.postgresql.org>)
- IBM DB2—IBM 已經在 DB2 上漸次加入物件導向的功能
- UniSQL—由 Won Kim 成立的 UniSQL Corp. 所發展
- Versant—由 Versant Object Technology, Inc. 公司發展
- Illustra—由 M. Stonebraker 的 Illustra 公司所發展
- Jasmine—由組合國際電腦股份公司 Computer Associates (CA) 所發展

物件導向模式的資料結構

- 類別的資料：由資料與其運算方法所組成，為了便於說明資料結構，下圖省略運算方法，僅列出資料部份





用詞與術語的對照

關聯式資料庫	物件導向式資料庫
資料型態 (Data Type)	值域 (Domain)
欄位/屬性 (Column/Attribute)	一律稱為「屬性」 (Attribute)
列/值組 (Row/Tuple)	實例 (Instance)
表格/關聯表 (Table/Relation)	類別 (Class)
視界 (View)	虛擬類別 (Virtual Class)
關聯表階層 (Table Hierarchy)	類別階層 (Class Hierarchy)
子表格 (Child Table)	子類別 (Subclass)
母表格 (Parent Table)	母類別 (Superclass)
預儲程序 (Stored Procedure)	運算方法 (Method)



物件導向式資料庫的資料結構

- 傳統關聯式資料庫提供了基本資料型態
 - 整數 (tiny int, small int, int long, ...) 、
 - 實數 (real, float, double, ...) 、
 - 日期 (date, small date, ...) 、
 - 變動 (varchar) 與固定字串 (char(*n*), ...) 等。
- 但是這些型態不足夠我們應付各種複雜應用的需求，所以必須擴充為複雜的資料型態。
請見下述三個範例的說明...

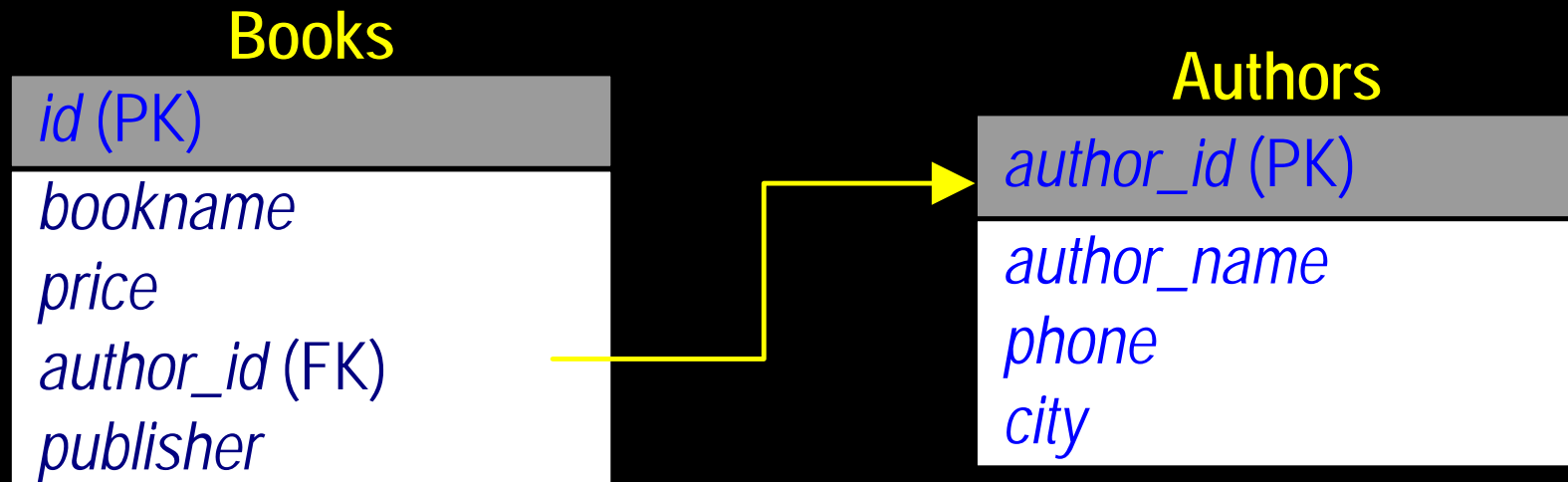
範例12.1：具有集合型態的欄位

Books

<i>id</i>	<i>bookname</i>	<i>Author</i>	<i>price</i>	<i>publisher</i>
1	UNIX系統	林明明,方英英	120	古文出版社
2	資料庫系	曾守正,周韻寰	170	中庸出版社
3	Web入門	李大同,王小華	170	春秋出版社
4	網路安全	張春嬌,林志明	140	聊齋出版社

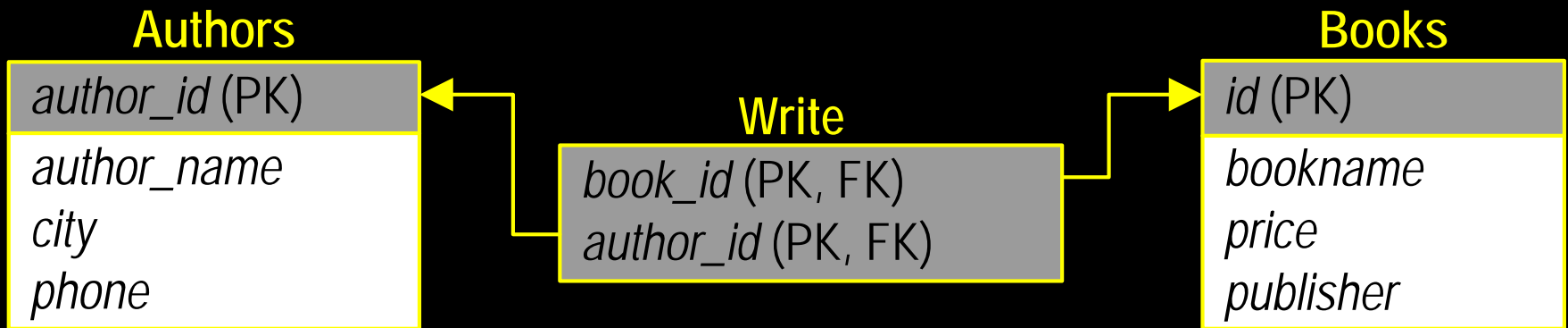
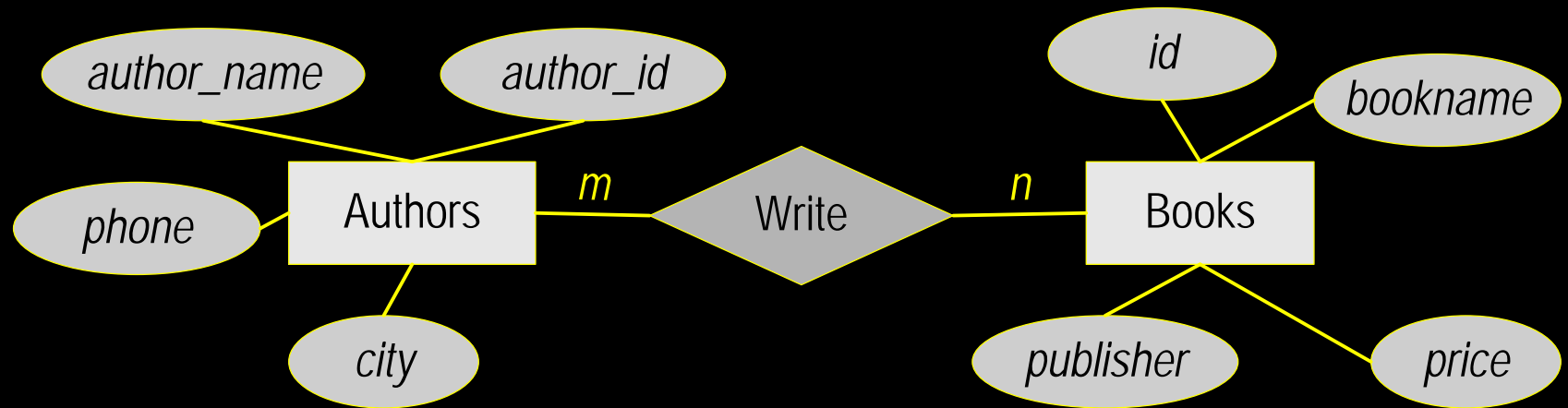
- 第一正規化規定：每個屬性值只能有一個。
怎麼辦？
- 土法煉鋼？自行寫程式以字串函數解決？

範例12.2：將屬性放大為關聯表



- 想儲存書籍作者的詳細資料，如：身份證字號、居住城市、電話等，就需要將作者獨立成另一個關聯表 **Authors**，並配合外來鍵的參考

範例12.3：綜合前述兩需求





範例12.3：綜合前述兩需求

- 既要具有集合型態、又要將某個欄位放大。
- 作法是在 **Books** 與 **Authors** 中間再加 **Write** 關聯表，存放書是由那些作者所撰寫，當然也描述了那位作者撰寫那些書的資料
- 每次都要將 **Authors**、**Write**、**Books** 三個關聯表做**明確式合併 (Explicit Join)** 才能取得所有的資料—不太自然！
- 在物件導向式的資料庫裡，就可使用效能較高的「**隱含式合併**」(**Implicit Join**) 來完成工作。



擴充式基本資料型態的優點

- 所有的東西都應該當作物件來平等看待，
- 可以將 “多位作者” 的觀念變成 “作者群” (也就是一個集合) 這個物件，
- 在物件導向式的資料庫中，可以將 **Authors** 變成一個抽象資料型態 (Abstract Data Type) —其實就是 Class
- 型態可以是簡單的字串型態—也是一個 Class，也可以複雜到如 **Authors** 類別 (以關聯表表示)



擴充式基本資料型態的優點

- 打破關聯式資料庫第一正規化的規定，讓屬性值中可內含集合 (Set)、多重集合 (Multi-set) 或序列 (List, 或 Sequence)
- 不再需要透過明確合併 **Authors**、**Write**、**Books** 三個關聯表，取得完整資料，
- 透過以 **Oid** 做為指標來追蹤資料的「隱含式合併」(Implicit Join) 來完成，
- 在效率的表現上會較佳。



物件導向式資料庫的集合型態

- *SET Domain* : 如前面所提到的 SET string 。
 - 表示該屬性中所含的資料為 *Domain* 的集合型態，
 - 集合元素不重複，彼此間也沒有任何順序關係。
 - 例如：宣告 `author SET string` 之後，
{ '林明明', '方英英' } 這個集合便可以存入一個屬性值中。
- *LIST Domain* (或 *SEQUENCE Domain*) :
 - 有序組合型態，元素之間可以不重複，順序關係必須要維持
 - 如：宣告 `author LIST string` 之後，{ '林明明', '方英英' } 這個有序串列
'林明明' → '方英英' 便可以存入 `author` 屬性中



物件導向式資料庫的集合型態

- *MULTISET Domain*：類似 *SET Domain*，但是集合元素可以重複，所以稱為「多重集合」(Multi-set)。例如：宣告 `price MULTISET int` 後便可以存 `{1, 1, 2, 3, 3}` 到屬性值中，而且重複的值不會被去除。
- *SET (Domain₁, Domain₂, ..., Domain_n)*：如 *SET* 集合型態，但是屬性值的資料來源可以是 *Domain₁, Domain₂, ..., Domain_n* 中各種型態所成的集合。例如：宣告 `note SET (string, int)` 之後，可以存 `{'林明明', '方英英', 20, 10}` 的屬性值到 `note` 欄位。



物件導向式資料庫的集合型態

- $LIST (Domain_1, Domain_2, \dots, Domain_n)$ ，類似 $LIST$ $Domain$ 宣告方式，但是屬性的資料可以是 $Domain_1, Domain_2, \dots, Domain_n$ 各種型態
- 例如：宣告 $note\ LIST (string, int)$ 之後， $\{20, '林明明', '方英英'\}$ 這個序列便可以存入屬性值 $note$ 中，而且 $20 \rightarrow '林明明' \rightarrow '方英英'$ 是有順序的。



物件導向式資料庫的集合型態

- $\text{MULTISET} (Domain_1, Domain_2, \dots, Domain_n)$ ：類似 $\text{MULTISET } Domain$ ，但是該屬性中所含的資料來源可以是 $Domain_1, Domain_2, \dots, Domain_n$ 中各種型態所成的集合，彼此間也沒有任何順序關係，集合元素可以重複。
- 例如：宣告 `note MULTISET (string, int)` 之後，`{'林明明', '方英英', 20, 20}` 這個多重集合便可以存入屬性值 `note` 中，而且重複的值不會被去除。



物件導向式資料庫的集合型態

- 如果前述的 4, 5, 6 種宣告中， $Domain_1$, $Domain_2$, ..., $Domain_n$ 都是空的，則代表E這個屬性裡面是一個可以由任意型態 (包括使用者自訂的抽象資料型態) 的資料值所構成的集合
- 例如：宣告了 `note SET ()`。則可以存入 `{1, 0.5, 'Kaohsiung', '98/01/30'}`。



集合型態間互相的轉換

- SET：轉成 MULTiset 不必透過 CAST 即可轉換 (Why ?)，但是反向或轉換成 SET 則需要透過 CAST (強制轉換)。
- LIST：可以透過 CAST 轉換成 LIST、SET 或MULTiset。
- MULTiset：可以透過 CAST 轉換成 SET 或MULTiset。



關聯式所沒有的特殊運算子

- 比較特殊的是：兩個 List 的聯集的結果是將兩者做連成一串的動作，所以還是維持原有的順序。
- 集合比較運算：SETEQ (=)、SETNEQ (\neq)、SUPERSET (\supset)、SUBSET (\subset)、SUPERSETEQ (\supseteq)，與SUBSETEQ (\subseteq)。

集合型態的運算模式

- UniSQL 將 $+$ 、 $-$ 、 $*$ 分別用來做為集合型態的聯集、差集與交集的運算子

運算元的集合型態	運算子	運算元的集合型態	運算結果的集合型態
SET	$+$, $-$, $*$	SET	SET
SET	$+$, $-$, $*$	MULTISET, LIST	MULTISET
MULTISET	$+$, $-$, $*$	SET, LIST, MULTISET	MULTISET
LIST	$+$	LIST	LIST
LIST	$+$	SET, MULTISET	MULTISET
LIST	$-$, $*$	SET, LIST, MULTISET	MULTISET



類別的運算方法

- 整數、實數、日期等類別是由系統所提供，當然其運算方法 (Method)，如：加、減、乘、除等，也是由系統提供內定的定義。
- 如果是由使用者自行定義的類別，則須由使用者自行定義其類別上的運算方法。
- 這些運算方法可以用 C、SQL 或其他語言撰寫後，在系統內註冊即可使用。
- 在 SQL 中則可以放在 where 子句中使用。



類別的運算方法

- UniSQL 中的 Method 要與其所屬的 Class 一併宣告，如果不屬於任何 Class，則將它宣告為隸屬於 **Object** 這個系統最大的類別。
- Method 在 UniSQL 中可以分成以下兩種方法：
 - **類別運算方法** (Class Method)：可透過類別物件來呼叫，通常是用來新產生一個該類別的實例時，做初始化 (Initialization) 的動作。
 - **實例運算方法** (Instance Method)：可透過該類別的實例物件來呼叫，一般的 Method 都是這類。



類別的階層與繼承

- 任何一個類別都可能是另一類別的母類別 (Superclass) 或子類別 (Subclass)。
- 所有母類別上所定義的屬性與運算方法都會由子類別繼承
- 子類別只會繼承屬性定義與運算方法，並不會繼承其資料。
- 母類別與子類別兩者所含的資料是完全沒有交集的。



類別的階層與繼承

- 子類別可以有多個母類別，並繼承不同母類別的所有屬性與運算方法，這種情況我們稱為「多重繼承」(Multiple Inheritance)。
- 在UniSQL中要宣告某個類別為多個類別的子類別時，只要用下述的語法即可達成 (under 可以換成 as subclass of)：

```
create class class_name under class_name {  
  class_name}.....
```

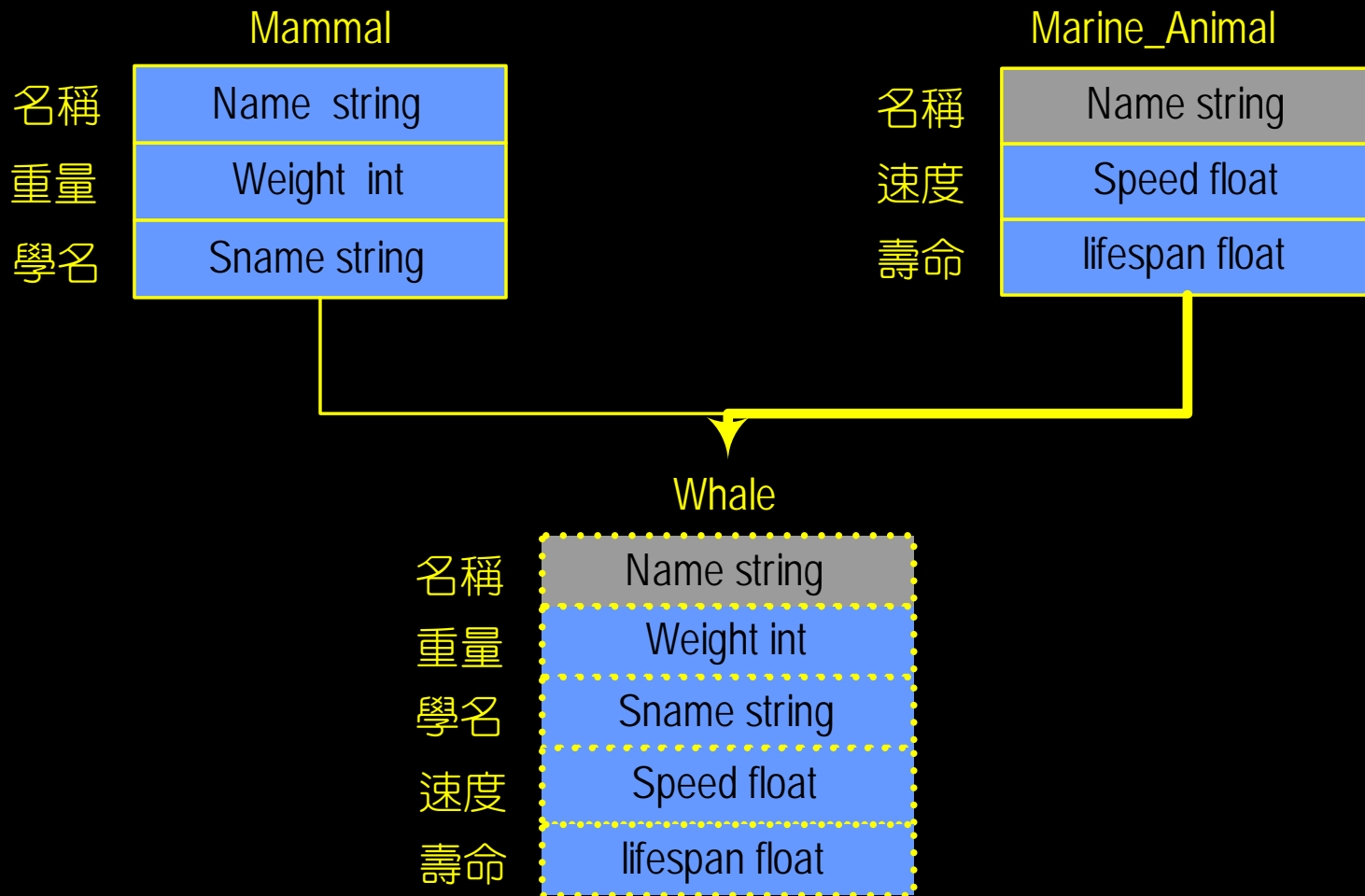
/ 如前所述的屬性宣告 */*



繼承的規則

- 子類別若有與母類別相同定義的屬性或運算方法時，則子類別上會以其屬性或運算方法去覆蓋母類別上的定義。
- 類別階層上具有相同屬性名稱的屬性都必須要有相同的資料型態。多重繼承的情況也是一樣。使用者也必須指明要從那一個類別來繼承。
- 在繼承階層中的同名 method 都要具有相同的參數型態與個數，以及相同的回傳值。

繼承的規則

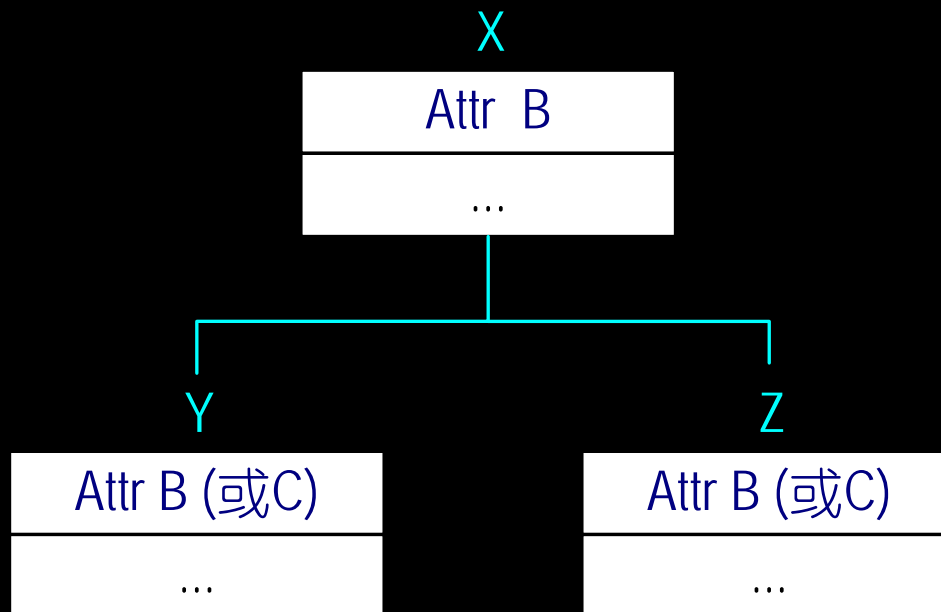
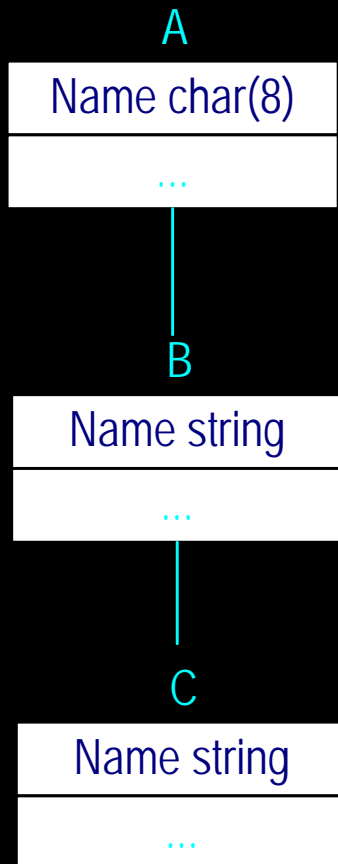




繼承的規則

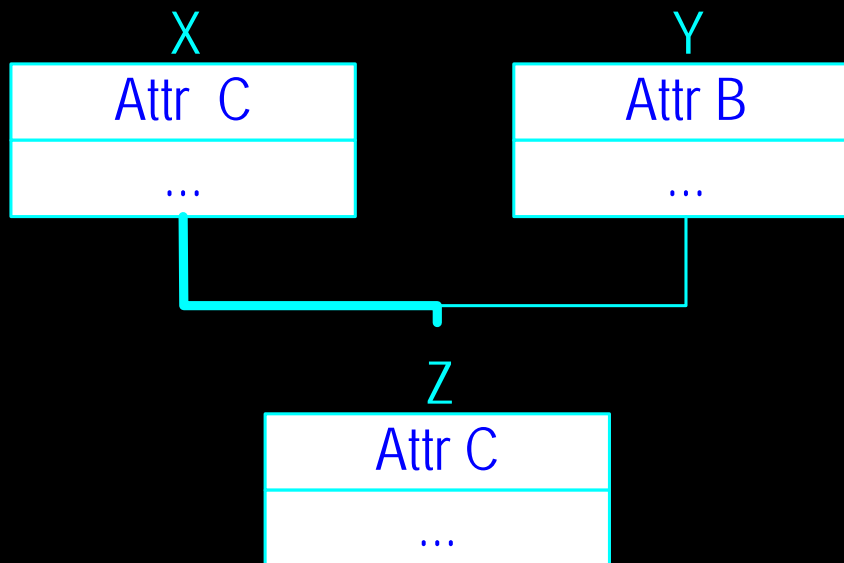
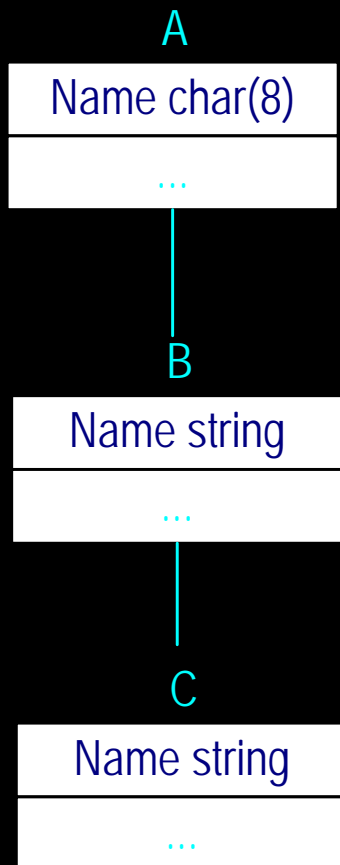
- UniSQL的語法如下：
create class Whale under Mammal, Marine_Animal
... /* 如前所述的屬性宣告 */
inherit Name of Marine_animal [as alias_name]
- 如果子類別多重繼承兩個以上的母類別，而且這些母類別中有相同的**運算方法**時，因為這些運算方法實現方式可能完全不一樣，所以宣告時也必須指定要從那一個母類別繼承。

繼承的規則



因為母類別 X 已經繼承了 B，所以 Y 與 Z 只能繼承 B 或 C，不可以繼承 A

繼承的規則



若已經有左方的繼承關係時，則 Z 只能繼承 X 的 C，不可以繼承 Y 的 B

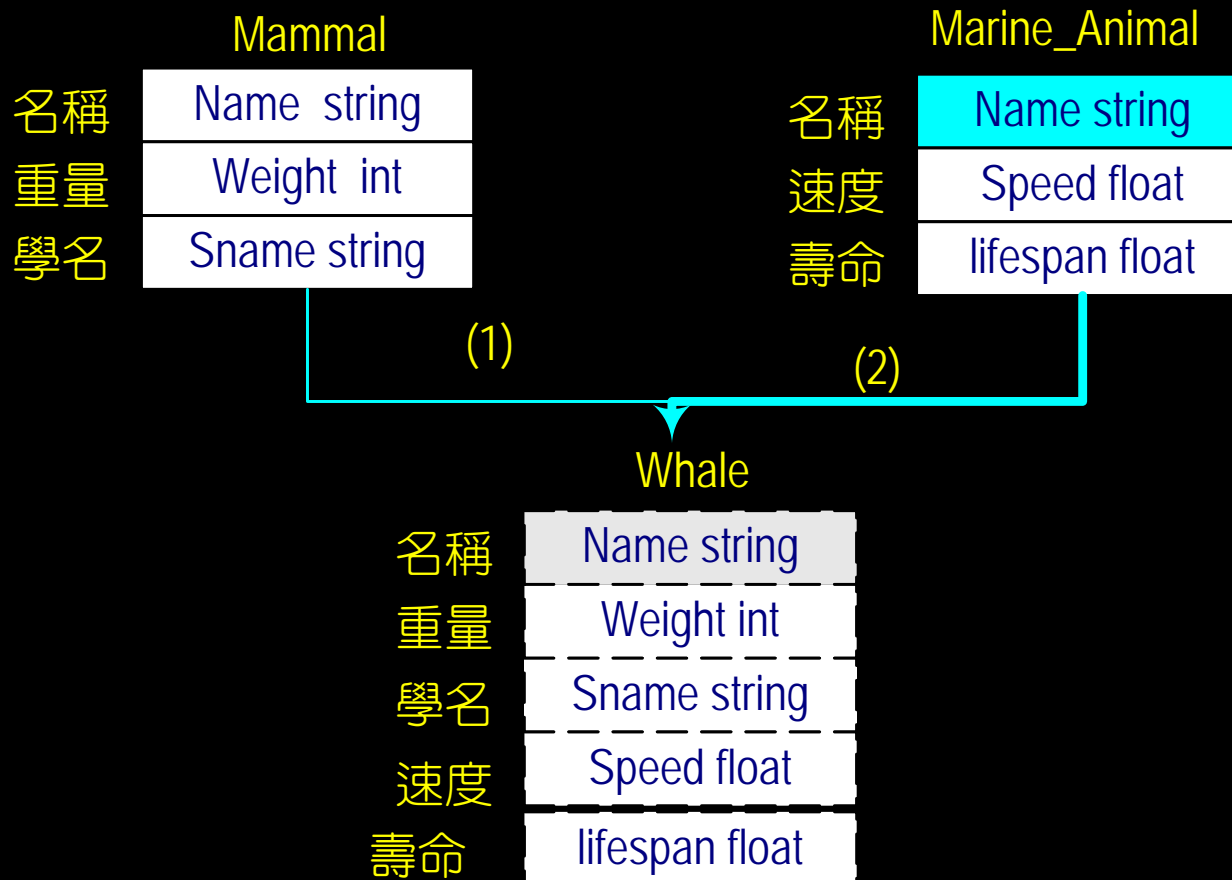


母類別與子類別間的衝突

- 會導致母類別與子類別間的衝突情況：
 - 新增屬性或運算方法。
 - 刪除屬性或運算方法。
 - 在現有類別 B 上宣告某類別 A 為其母類別。
 - 取消某個母類別與子類別間的繼承關係。
 - 刪除某個類別。

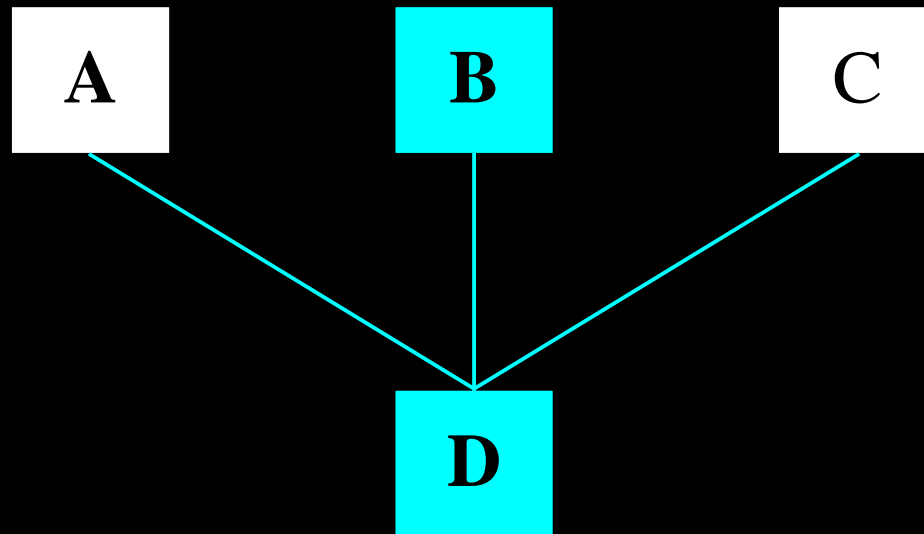
在母類別中所引發的衝突

- 新增繼承關係所產生的衝突：



在母類別中所引發的衝突

- 刪除繼承關係所產生的衝突：D 原來繼承 B 中的屬性，如果將 B 刪除則 D 要繼承 A 或 C ？

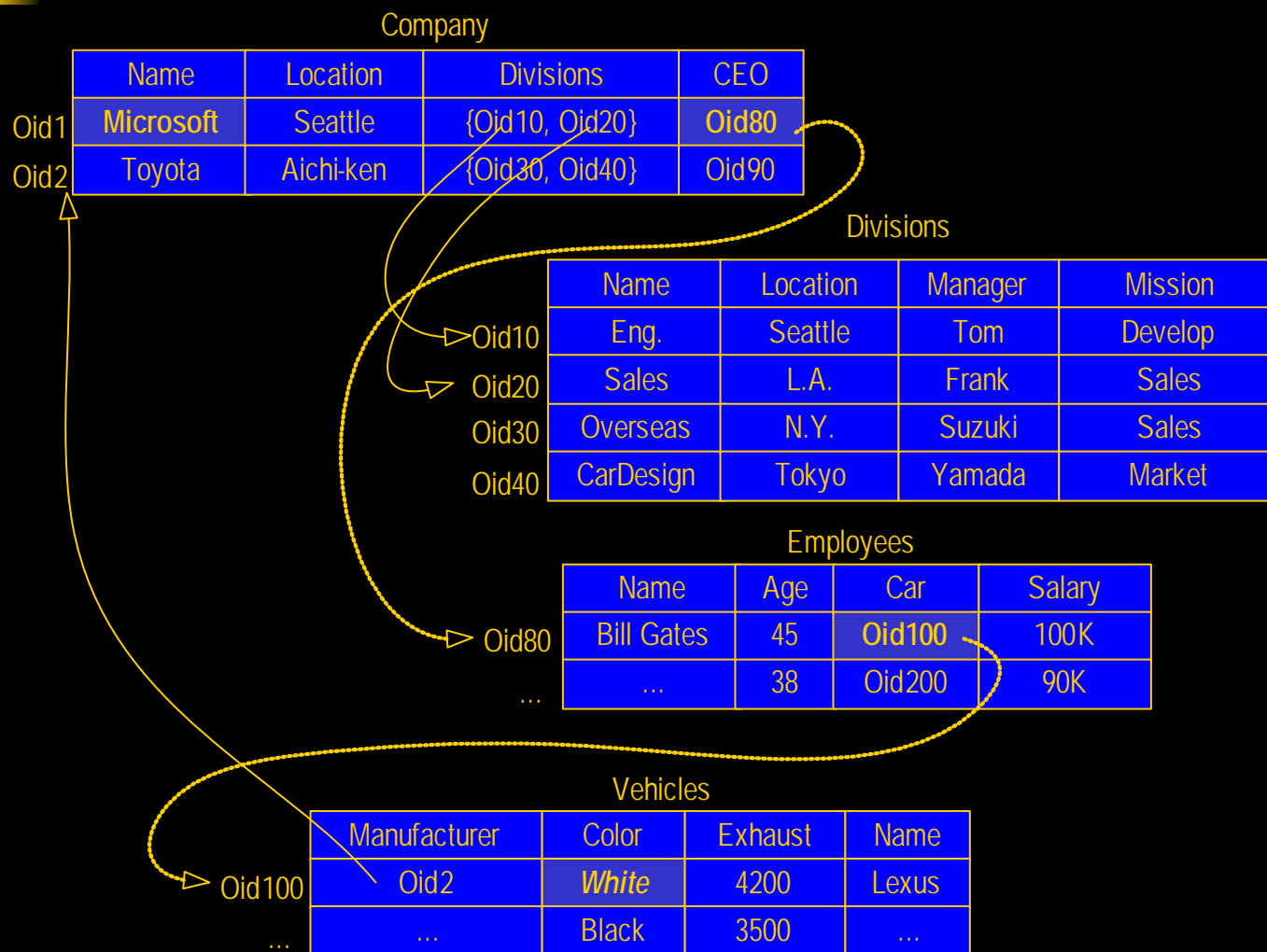




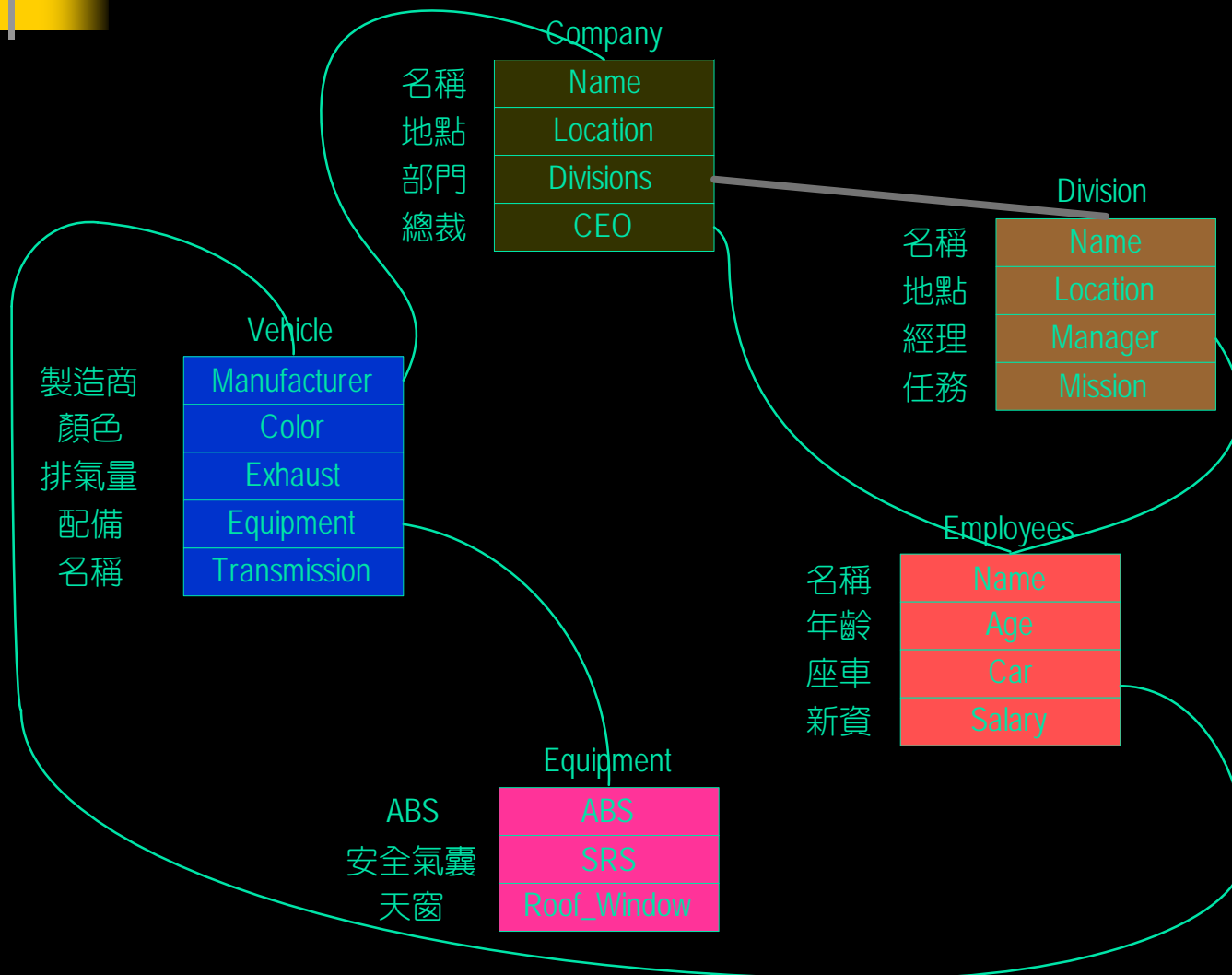
在子類別中所引發的衝突

- 新增屬性所產生的衝突：在某個類別上新增一個屬性，如果與其母類別有相同的屬性名稱時，由於先前已經繼承了該屬性了，現在又加入一個完全同名的屬性，所以衝突便會發生。
- 刪除被選為繼承對象的母類別屬性所產生的衝突：當母類別中有某個被選為繼承對象的屬性被刪除時，因子類別可能無所適從，所以產生衝突。

物件導向式資料庫範例



物件導向範例資料庫 CAM





物件導向式資料庫的 DDL

- 新增資料庫的類別

```
create class class-name(  
  attr_name type [shared [value] | default [value] ...)
```

- 以下以範例12.1 ~ 範例 12.3 說明關聯式資料模式的劣勢
- 在物件導向式資料庫裡，只要透過類別的建立，即可達成這些例子的需求。



滿足範例 12.1 的需求

- CREATE Class Books (
 id int NOT NULL UNIQUE,
 /* 可以用 Oid 取代 */
 bookname char(30) NOT NULL,
 author **SET** string NOT NULL,
 price small int NULL,
 publisher char(20) NULL
)



滿足範例 12.2 的需求

- CREATE Class Authors (
 author_id int NOT NULL UNIQUE, /* 可以省略 */
 author_name string NOT NULL,
 city char(15) NULL,
 phone char(15) NULL)
- CREATE Class Books (
 id int NOT NULL UNIQUE, /* 可以省略 */
 bookname char(30) NOT NULL,
 author **Authors** NOT NULL,
 price small int NULL,
 publisher char(20) NULL)



滿足範例 12.3 的需求

```
CREATE Class Authors (  
    author_id int NOT NULL UNIQUE, /* 可以省略 */  
    author_name string NOT NULL,  
    city char(15) NULL,  
    phone char(15) NULL)
```

```
CREATE Class Books (  
    id int NOT NULL UNIQUE, /* 可以省略 */  
    bookname char(30) NOT NULL,  
    author SET Authors NOT NULL,  
    price small int NULL,  
    publisher char(20) NULL )
```



CAM 資料庫綱要定義

- Create Class **Company** (
 - Name char (20),
 - Location char(20),
 - Divisions Set **Division**,
 - CEO **Employees**
 -)
- /* 一個公司有許多部門，
所以 Division 用 Set */

- Create Class **Division** (
 - Name char(20),
 - Location char(20),
 - Manager **Employees**,
 - Mission char(10),
 -)



CAM 資料庫綱要定義 (續)

```
Create Class Employees (  
    Name char(20),  
    Age Integer,  
    Car Vehicle,  
    Salary Integer  
)
```

```
Create Class Vehicle (  
    Manufacture Company,  
    Color Char(10),  
    Exhaust Integer,  
    Equipped Equipment,  
    Transmission Char(10)  
)
```

```
Create Class Equipment (  
    ABS Boolean,  
    Air_Bag Boolean,  
    Roof_Window Boolean)
```



屬性的宣告

- 在 UniSQL 中又將類別中的屬性再分成
 - 「實例屬性」 (Instance Attribute)、
 - 「類別屬性」 (Class Attribute)、
 - 「共享屬性」 (Shared Attribute)，與
 - 「預設屬性」 (Default Attribute) 四種
- 實例屬性 (Instance Attribute)：一個實例屬性具有某個特定實例的屬性值。例如：我們前面所定義的 *bookname*, *price*, *city* 等屬性皆是。



類別屬性 (Class Attribute)

- 類別屬性具有該類別特有的屬性值。一個類別中只有一個值而已，而且這個值與所有的實例都沒有關係。
- 通常是用來當做 meta-data，以描述該類別的特性或某個屬性的意義。宣告方式是在屬性名稱前加上 **class** 關鍵字。例如：
- CREATE Class Books (
 class *Books_meaning* string, /* 說明 Books 類別 */
 bookname char(30) NOT NULL,
 ...)



共享屬性 (Shared Attribute)

- 共享屬性是該類別中所有實例都會具有相同屬性值的屬性，
- 它的值在系統中只會存放一個而已，
- 但是每一個實例都會有此屬性值，
- 可以節省空間。
- 其宣告方式是在屬性資料型態後面加上 **shared** 這個關鍵字，並提供其屬性值於其後。



共享屬性 (Shared Attribute)

- 假定所有存放在 Books 中的書本都是中文書，而我們又不想要在每一筆資料中存放一個language 屬性值為‘中文’時，那麼可以宣告：
 - CREATE Class Books (
 bookname char(30) NOT NULL,
 author Authors NOT NULL,
 price small int NULL,
 publisher char(20) NULL,
 language char(4) **shared** ‘中文’
)



共享屬性 (Shared Attribute)

Books

<i>id</i>	<i>bookname</i>	<i>author</i>	<i>price</i>	<i>publisher</i>	<i>language</i>
1	三國演義	羅貫中	120	古文出版社	中文
2	水滸傳	施耐庵	170	中庸出版社	中文
3	紅樓夢	曹雪芹	170	春秋出版社	中文
4	西遊記	吳承恩	140	聊齋出版社	中文
5	水經注	酈道元	120	易經出版社	中文
6	道德經	老子	190	大唐出版社	中文



預設屬性 (Default Attribute)

- CREATE Class Books (
 bookname char(30) NOT NULL,
 author Authors NOT NULL,
 price small int NULL,
 publisher char(20) NULL,
 language string **default** '中文'
)
- 關聯式資料庫上已經有此功能



運算方法的宣告

- UniSQL 的格式：
- `create class class-name`
... */* 如前所述的屬性宣告 */*
[`method [class] method_name [(type1, type2, ...)] [result_type]`
[`function method_implementation_name`
[`file path_name [{, path_name}...]`];
- `method_name` 是運算方法名稱；
- `result_type` 是函數運算完後的回傳型態；
- `method_implementation_name` 是以 C 語言撰寫以實現此運算方法的函數名稱；
- `path_name` 是用來指向該 C 函數可執行檔路徑。
- 很像 SQL Server 2008 的「組合」(Assembly) 功能。

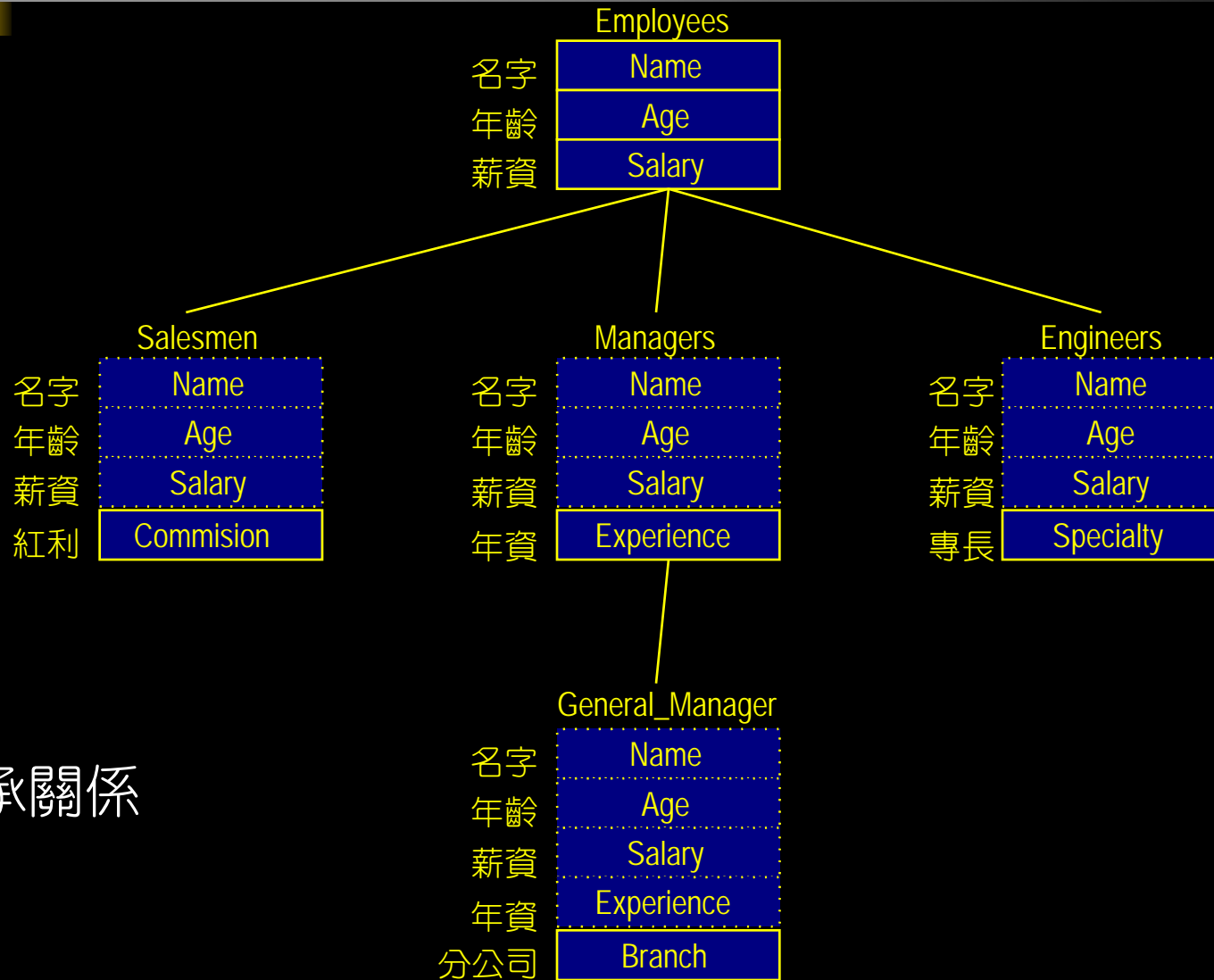


子類別的宣告

- 在 UniSQL 中，要宣告某個類別為多個類別的子類別時，用下述的語法即可達成 (under 可以換成 as subclass of)：

```
create class class_name under class_name {  
  class_name}...  
  attr_name type [shared [value] | default [value] ...)
```

資料庫綱要的繼承範例



繼承關係



各子類別的說明

- **Salesmen**：此子類別只有一個專屬於銷售員的屬性 Commission，用來儲存**銷售的紅利**。
- **Managers**：此子類別只有一個專屬於經理的屬性 Experience，用來儲存**工作的年資**。
- **Engineers**：此子類別只有一個專屬於工程師的屬性 Specialty，用來儲存其**專長**。



資料庫綱要的繼承定義

- Create Class Salemen under Employees (Commission integer)
- Create Class Managers under Employees (Experience integer)
- Create Class Engineers under Employees (Specialty string)
- Create Class General_Manager under Managers (Branch string)

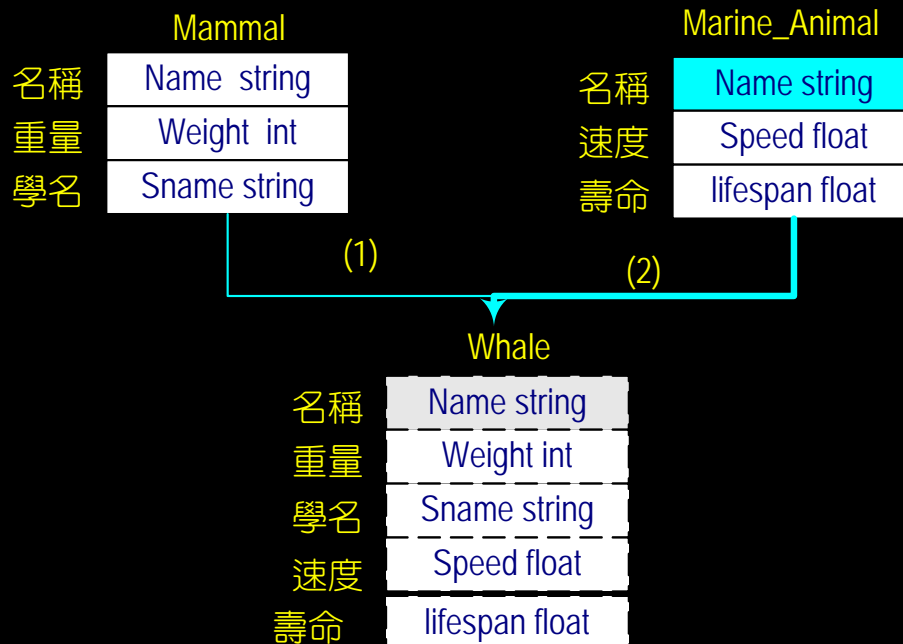
指定繼承

- 多重繼承的情況下，若有兩個母類別出現相同的屬性名稱時，必須指明要從那一個類別繼承。

create class Whale under Mammal, Marine_Animal

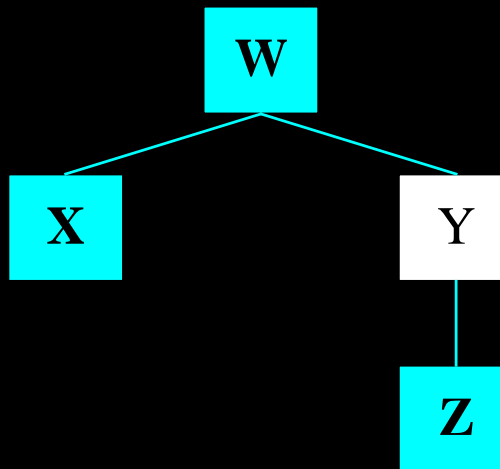
... /* 省略屬性宣告 */

inherit Name of Marine_Animal



刪除資料庫中的類別

- 在 UniSQL，若想刪除一個類別，可以使用：
`drop class [[only] class1
 | all class1 [(except [all] class2)]`
- e.g., drop all W (except Y)





刪除類別後的調整方式

- 當一個被其它類別所參考到的類別被刪除之後，除非另有指定，否則用到此類別的屬性值都會變成虛值 (Null)，請舉例說明之！
- 這些屬性的 Domain 則變成是系統中最大的類別 **Object**。
- 如果是有子類別繼承它的話，則所有繼承下來的屬性也都會被刪除。請舉例說明之！



修改類別中的屬性定義

- 在 UniSQL 中要新增一個屬性 (或類別屬性) 可以使用 alter 指令：
- `alter class classname`
`add attribute attr_name type [shared [value] | default [value]] ...;`
- `alter class classname`
`add class attribute class_attr type [shared [value] | default [value]] ...;`
- 該新增屬性也會加到所有下屬的子類別中。



修改類別中的運算方法

- **Alter class** 也可以用來新增類別的運算方法：
- **alter class** *classname*
add **method** [class] *method_name* [(*type_1*,
type_2, ...)][*result_type*]
function *method_implementation_name*
file *path_name*[{, *path_name*}...];



刪除類別中的運算方法

- 要刪除一個屬性、方法 (或類別屬性、類別方法) 也可以使用 alter 指令：
`alter class classname drop [attribute | method][class] attr_or_method,...;`
- 要改變屬性的預設值則使用：
`alter class classname
change [class] attr_name default value, ...;`



注意事項

- 修改屬性預設值只能在**原宣告類別**中做，不可以在任何其子類別中對此繼承所得的屬性做預值的修改動作。
- 改變實例屬性 (Instance Attribute) 並不會使目前的資料受到影響，**只會影響往後加入的資料**。
- 改變**類別屬性** (Class Attribute) 的預設值卻相當於對該屬性值做更新 (Update) 的動作一般。



重新命名

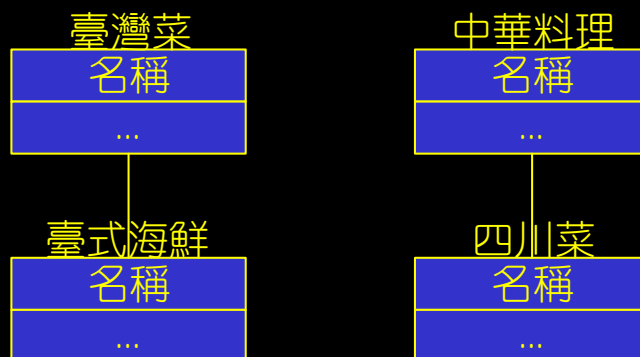
- 對屬性、運算方法（或類別屬性、類別運算方法）中的某些定義重新命名，則使用：
- `alter class classname rename rename_clause`
- `rename_clause`:
 - `[attribute | method] [class] attr_or_method as new_name`
 - `| function of [class] method_name as implementation_name`
 - `| file path_name as path_name`



修改類別間的繼承關係

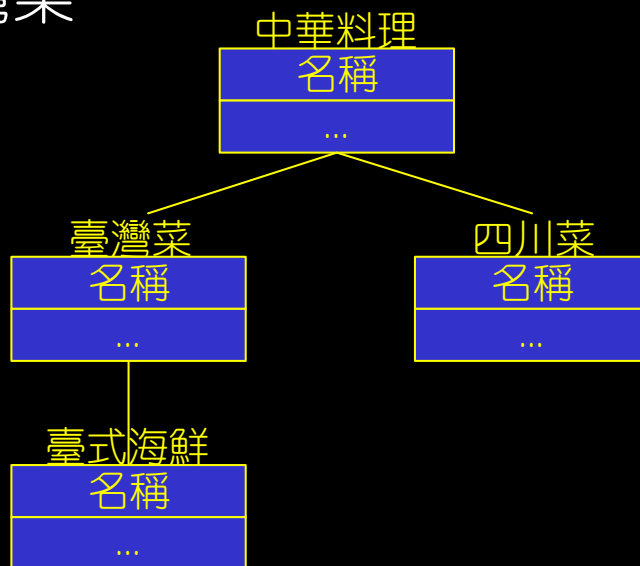
- 在現有類別之上宣告另一個類別為其母類別：
`alter class classname`
`add [或 drop 刪除] superclass class_name, ...;`
- 這樣 `classname` 還有它底下所有的子類別就會繼承所有 `class_name` 上的屬性與運算方法。
- 如圖 12.17 所示宣告前的關係要如何宣告才能變成宣告後的關係，請同學們練習一下。

修改類別的繼承關係



宣告前的情況

Alter class 臺灣菜
drop superclass
中華料理



Alter class 臺灣菜
Add superclass 中華料理

宣告後的情況



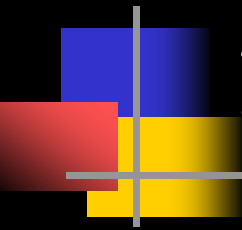
新增類別中的實例

- `insert into classname [(attr1, attr2, ..., attrn)] { values (value1, value2, ..., valuen) | select query_statement } [to :Oid_var];`
- 這個指令與關聯式資料庫的相同點是：
 - 它可以一一列出資料，以加到屬性中。
 - 也可以利用select查詢將查詢結果存入類別中。



新增類別中的實例

- 它與關聯式資料庫中最大的不同點是：
 - $value_n$ 中的值可以放置集合式的資料，以便讓使用者得以一次將集合式的資料加到集合型態的屬性中。
 - 整個 insert 指令會回傳該實例物件的 Oid，並且可以將此 Oid 放入暫存變數，以便後續可以再將此 Oid 新增到以此種類別為 Domain 的其它屬性。



最簡單的單一個實例物件

- 以範例 12.1 的例子
- insert into **Books** (*bookname*, *author*, *price*, *publisher*) values ('資料庫系統', {'曾守正', '周韻寰'}, 650, '華泰出版社');

Books

<i>bookname</i>	<i>Author</i>	<i>price</i>	<i>publisher</i>
資料庫系統	{曾守正, 周韻寰}	650	華泰出版社



新增使用者自訂類別的實例物件

- 以範例 12.3 的需求情況來說明
- 先加入作者資料到 Authors 中，並將系統回傳的 Oid 暫存在變數中：

```
insert into Authors (authorname, city, phone) values ('  
曾守正', '高雄縣', '(07) 753-3967') to :Frank;
```

```
insert into Authors (authorname, city, phone) values ('  
周韻寰', '高雄市', '(07) 571-1438') to :Annie;
```

- 再將暫存變數一次加入 Books 類別中

```
insert into Books (bookname, author, price, publisher)  
values ('資料庫系統', {:Frank, :Annie}, 650, '華泰');
```

新增完成後的狀態

Books

<i>bookname</i>	<i>author</i>	<i>price</i>	<i>publisher</i>
資料庫系統	{:Frank, :Annie}	650	華泰出版社

Authors

<i>author_name</i>	<i>city</i>	<i>phone</i>
曾守正	高雄縣	(07) 753-3967
周韻寰	高雄市	(07) 571-1438





一次完成上述狀態

- insert into **Books** (*bookname, author, price, publisher*)
values ('資料庫系統',
{
 insert into **Authors** (*authorname, city, phone*)
 values ('曾守正', '高雄縣', '(07) 753-3967'),

 insert into **Authors** (*authorname, city, phone*)
 values ('周韻寰', '高雄市', '(07) 571-1438')
},
650, '華泰出版社');



使用子查詢來新增實例物件

- 新增一本書名為 “資料庫系統理論與實務” 的書到 Books 中，其作者與 “資料庫系統應用實務” 是相同的作者
- ```
insert into Books (bookname, author, price, publisher)
values ('資料庫系統理論與實務',
set (select Books.author
 from Books
 where bookname = '資料庫系統應用實務'),
650, '華泰出版社');
```



# 修改類別的實例

---

- 最簡單的實例物件修改：
  - 將價錢小於 150 的書定價提高百分之五：  
update **Books**  
set price = price \* 1.05  
where price < 150;
- 與關聯式的作法一樣



# 修改集合屬性

---

- 將 “資料庫系統” 這本書的作者改成 “系統分析與設計” 這本書的作者 (可能會有很多位) :
  - update Books  
set *author* = (  
select *author* from Books  
where *bookname* = '系統分析與設計'  
)  
where *bookname* = '資料庫系統';



# 修改類別屬性

---

- `update class Books`  
`set Books_meaning = '用來存放本公司所販賣的書籍資料'`  
`[ where search_condition];`
- 由於類別屬性在一個類別中通常只有一個，所以 `where` 子句通常不會用到。





# 刪除類別中的實例

---

- `delete from {[only] classname | all classname [(except [all] classname)] where condition};`
- **all** 表示連 *classname* 以下的子類別實例都一併刪除。
- **Only** 是只刪除該類別，子類別不受影響。
- 在刪除時想保留繼承階層中的某一個子類別時，則可以加上 **except** 子句。
- 與 12.7.2 節中所述的很類似，只不過本節對象是刪除類別中的實例，不是刪除整個類別。



# 刪除類別中的實例

---

- 最簡單的實例物件刪除
  - 將價錢小於 150 的書刪除：

```
delete Books
where price < 150;
```

- 與關聯式的作法一樣



# 以子查詢刪除

---

- 將 “張三” 所撰寫的書都刪除：
- ```
delete from Books  
where '張三' in (  
select author.authorname  
from Books  
);
```
- 與關聯式的作法不太一樣



查詢類別資料

- **Select** [distinct] *a-list-of-attribute-names*
[**{to | Into}** *new_relation_name*]
From *a-list-of-class-names* (可加 *alias name*)
[**Where** *a-true/false-condition*]
[**Group by** *a-list-of-attribute-name*
[**Having** *a-true/false group-condition*]]
[**Order by** *a-list-of-attribute-name*]

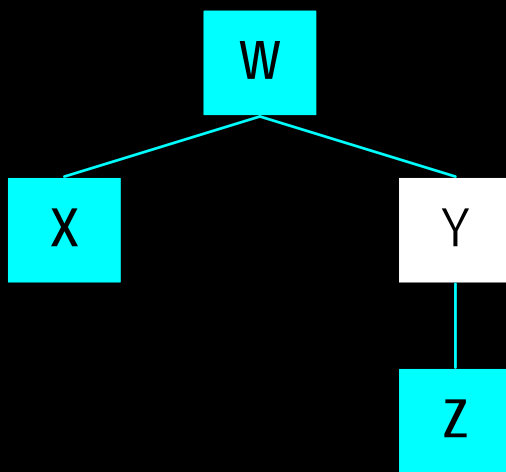


查詢類別屬性

- 從類別來查詢類別屬性：
`Select class-attribute From class class-name;`
這樣系統只會回答我們單一個答案。
- 在繼承階層上查詢部份的類別，可以在 from 子句中加上以下語法：
`from [[only] class1 | all class1 [(except [all] class2)]]`
 - All 代表：連 *class1* 以下的子類別都一併搜尋。
 - Only 意思是只查詢該類別，不搜尋其子類別。

在繼承階層上選擇性查詢

- 在 from 子句中加上如下的語法：
`from [[only] class1 | all class1 [(except [all] class2)]]`
- `select * from all W (except Y)` 的指令會使 Y 被跳過搜尋。





可以任意出現的子查詢

- 只要符合類別與資料型態的比對規則，子查詢可以出現在任何屬性、類別或常數所能出現的地方，而且可以用集合的方式來顯示：

```
select bookname, price, (select author_name from  
Authors where bookname = city)  
from Books
```
- 顯示以居住城市為書名的書籍資料，輸出其名稱、價格、與作者名稱。

子查詢回傳一個集合的情況

- 在整個子查詢的最前面加上 **set** 或 **multiset** 關鍵字即可，如果是子查詢中有 **order by** 子句的話，那麼在最前面加上 **list** 來產生一個有順序的表列結果，當然也沒有問題。例如，
- ```
select bookname, price, set (
 select author_name
 from Authors
 where bookname = city)
from Books
```

| <i>bookname</i> | <i>price</i> | <i>Author_name</i> |
|-----------------|--------------|--------------------|
| 高雄市             | 150          | {謝市長, 林局長, 王秘書}    |
| 臺北市             | 120          | {陳市長, 林局長}         |





# 子查詢會回傳整個類別的情況

- 此種子查詢也可以出現在 **from** 子句中，不過記得要將此子查詢所回傳的關聯表用 **as** 關鍵字另外命名，以方便其他子句的使用。例如：
- ```
select bookname, price, avg_price
from Books, (select bookname, avg(price), publisher
              from Books
              group by publisher) as
              tmp (bookname, avg_price, publisher)
where Books.publisher = tmp.publisher
```

查詢陣列元素的資料

- 假設業務員依照其業績，每個月薪水都不同，那要如何儲存起來？並快速查詢？
- 作法可以用陣列來完成 (PostgreSQL 已提供此功能)

Employee

<u>emp_id</u>	emp_name	phone	payment
10036	李大同	0932888588	[40k, 28k, 35k, 29k, 36k, 28k, 33k, ...]
10057	王小華	0936088588	[19k, 32k, 18k, 25k, 33k, 38k, 40k, ...]

標準陣列的查詢寫法

- 找出李大同 7 月份的薪水為何？

```
Select payment[7]  
From Employee  
Where emp_name = '李大同';
```

Employee

<u><i>Emp_id</i></u>	<i>Emp_name</i>	<i>phone</i>	<i>Payment</i>
10036	李大同	0932888588	33k

傳統的關聯式作法

Employee

<u>emp_id</u>	emp_name	phone
10036	李大同	0932888588
10057	王小華	0936088588

Salary

<u>emp_id</u>	payment	<u>month_year</u>
10036	40k	January 1999
10057	19k	January 1999
10036	28k	February 1999
10057	32k	February 1999
10036	35k	March 1999
10057	18k	March 1999
...

找出李大同 7 月份的薪水!

```
select payment
from Salary
where month = 'July' and emp_id in
(select emp_id
from Employee
where emp_name = '李大同')
```

或下述指令 (效能都不佳)

```
select payment
from Employee E, Salary S
where E.emp_id = S.emp_id
and S.month = 'July'
and E.emp_name = '李大同'
```



結論

- 物件導向式資料庫管理系統將會在下一波的進階應用中佔有一席之地。因為有下面這兩股力量在背後推動：
 - 商用資料處理上的應用會有越來越多的複雜結構必須要面對。
 - 陸續會出現許多新型態的資料庫管理系統應用，特別是多媒體與全球資訊網 (World-Wide Web) 上的應用。



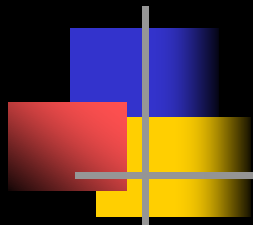
結論

- 可以預見的是：左上角象限的應用會有一些漸漸往右上角象限移動。
- 但我們不知道何時，物件關聯式資料庫將會成為主流的資料庫管理系統技術，因為畢竟其複雜的應用與結構要能推展到各個階層並不是一件短期內能做到的事。
- 大多數的關聯式廠商將會實現為人所稱道的物件導向式或物件關聯式的功能，



結論

- 關聯式系統不見得會成為新的 “古董系統” (Legacy Systems)，加入階層式與網路式系統的行列，變成夕陽工業。
- 傳統關聯式資料庫短期內仍不會被遺棄，因為我們已經找到它在「資料倉儲」(Data Warehousing) 與「資料採礦」(Data Mining) 上的許多應用，同時配合 XML 與 Web Service 將會開創另一番新局
- 透過這些應用建立「客戶關係管理」(CRM, Customer Relationship Management) 概念。



本章結束
The End.