資訊管理學系 陳士杰老師

資料庫系統管理
Database System Management

資料庫系統概論
Database System: An Overview

國立聯合大學
NATIONAL UNITED UNIVERSITY

Outlines

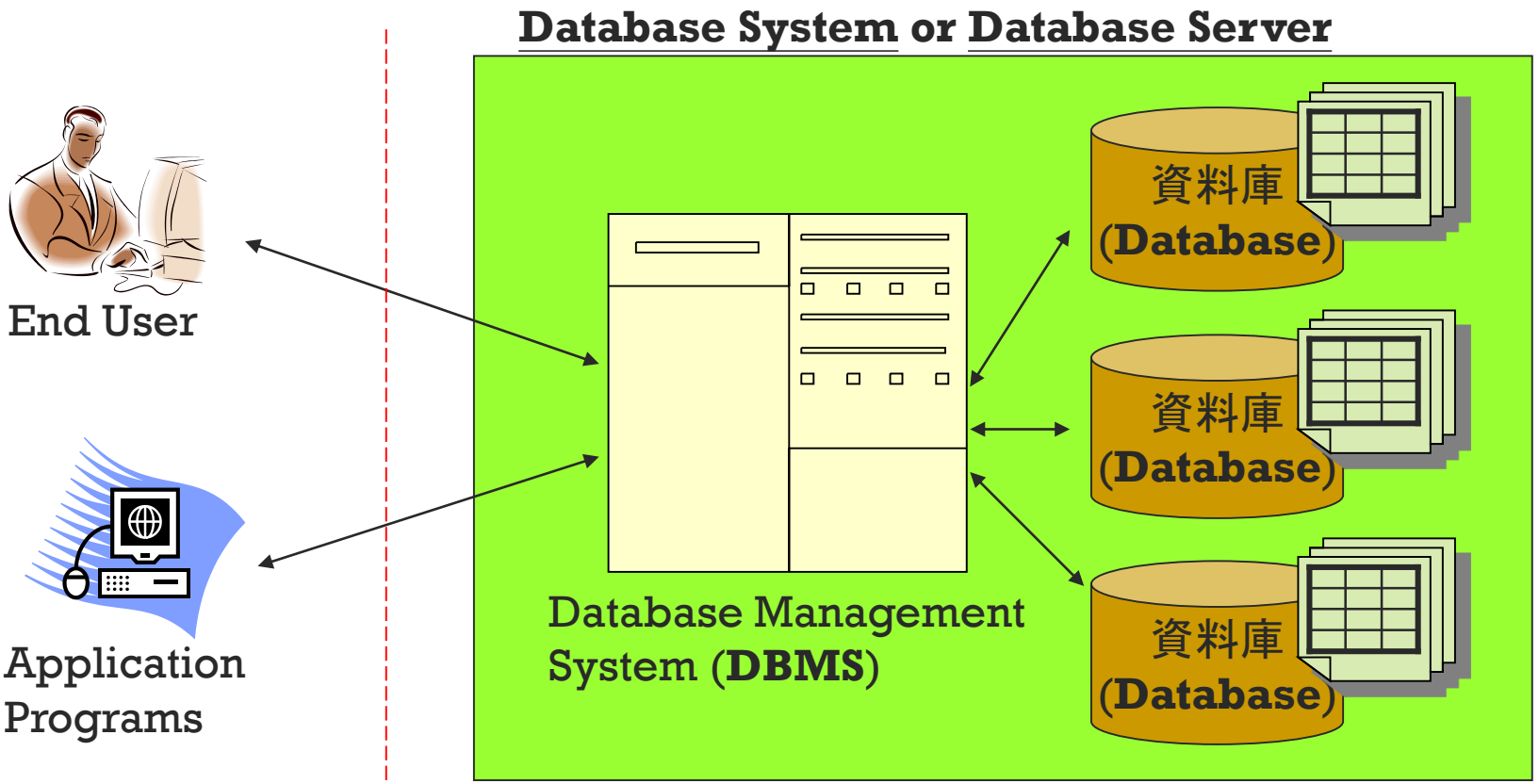
- Basic Definitions
- 使用DBMS的優點
- Database Users
- 何謂 View (觀點、視觀)？
- Meta-data (中繼資料)
- Three-Schema Architecture
- Centralized、Client/Server、Multi-tier Architectures for DBMS
- 資料庫相較於檔案系統的優點
- 資料庫的缺點

【講義：Ch. 1, Sections 1~3】

【原文：Ch. 1, Sections 1-1, 1-2, 1-3, 1-4, 1-6; Ch. 2, Section 2-1, 2-2, 2-5】

Basic Definitions

- **SQL (結構化查詢語言)**：User與AP皆透過SQL與DBMS溝通、存取DB data!!!



- **Database (資料庫; DB)**
  - A collection of related data. (一群相關資料的集合)
  - 將相關的資料以系統化的方式有效率地儲存在一起，並減少資料的重覆性
  - 一個資料庫往往是因為特定的需求或目的而被建構。因此，可以被特定的應用程式或人員所使用
  - 透過資料庫管理系統 (Database Management System; DBMS) 來管理

- **Database Management System (資料庫管理系統; DBMS)**
  - 是一群程式的集合。
  - 幫助使用者對資料庫或資料庫系統進行定義(Defining)、建構(Constructing)、操作(Manipulating)與共享(Sharing)。
    - Defining (定義)：定義那些要儲存在資料庫裡的資料，其資料型態、操作限制…等。
    - Constructing (建構)：實際建置資料表格、資料庫…等相關物件，與資料庫系統運作環境設定。
    - Manipulating (操作)：對儲存資料或資料庫物件做查詢(Querying)、更新(Updating)…等相關運作。
    - Sharing (共享)：允許多個使用者/程式同時存取資料庫。

- **Database System (資料庫系統)**
  - The DBMS software together with the database.
  - Database System = DBMS + DB
  - 例：Oracle, SQL Server, DB2, MySQL, ...

使用DBMS為資料庫系統所帶來的優點

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- **Controlling Redundancy**
  - **控制重複性**
  - **整合相同的資料並集中存放，可節省儲存空間，且減少不一致的現像**
- **Restricting Unauthorized Access**
- **Providing persistent Storage for Program Objects**
- **Providing Backup and Recovery**
- **Providing Multiple User Interfaces**
- **Representing Complex Relationships among Data**
- **Enforcing Integrity Constraints**
- **Permitting Inferencing and Actions Using Rules**

- Controlling Redundancy
- **Restricting Unauthorized Access**
  - **限制未授權的存取**
  - **提供適當的安全性與認証機制，通常是使用帳號與密碼獲得相對的資料庫存取權限**
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- **Providing persistent Storage for Program Objects**
  - **提供程式物件永久的儲存空間**
  - **將複雜的程式物件或資料結構永久性地儲存在DBMS中，解決一般程式終止後，變數即被丟棄的問題。**
  - **O-O DBMS**
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- **Providing Backup and Recovery**
  - **提供備份與回復功能**
  - **確保資料庫可以回復到程式開始執行前的狀態，或確保程式能從之前中斷處繼續執行。**
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- **Providing Multiple User Interfaces**
  - **提供多重使用者介面**
  - **針對不同程度的使用者，提供不同的使用者介面，包括：圖形化使用者介面、自然語言介面、查詢語言、程式語言介面...等。**
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- **Representing Complex Relationships among Data**
  - **表示資料間的複雜關係**
  - **可以利用各資料表格之間的相關欄位 (即：外來鍵) 來表示其間的關係。**
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- **Enforcing Integrity Constraints**
  - **實施完整性限制**
  - **資料庫的運作必須遵守這些限制，以維持資料庫的運作順暢。**
  - **個體完整性限制、參考完整性限制、與其它特定資料的完整性限制。**
- Permitting Inferencing and Actions Using Rules

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing persistent Storage for Program Objects
- Providing Backup and Recovery
- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- **Permitting Inferencing and Actions Using Rules**
  - **允許使用規則來進行推論與活動**
  - **從已儲存的資料庫中推論出新的資訊，以找出原本未知的資訊。**
  - **for 知識管理、客戶關係管理、Data Mining...**

■資料庫系統相較於檔案系統的優點

- 避免資料重覆存放 (Data Redundancy)
- 避免資料不一致 (Inconsistency)
- 資料的分享 (Multi-User)
- 標準的確立
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- 資料獨立性 (Data Independence)

- **避免資料重覆存放 (Data Redundancy)**
  - 資料庫透過**資料集中化 (Data Centralized)**及**存取界面標準化**來減少資料的重覆存放，讓多個User讀取同一份資料庫內容。
  - 資料庫系統無法完全避免資料重覆存放。
- 避免資料不一致 (Inconsistency)
- 資料的分享 (Multi-User)
- 標準的確立
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- 資料獨立性 (Data Independence)

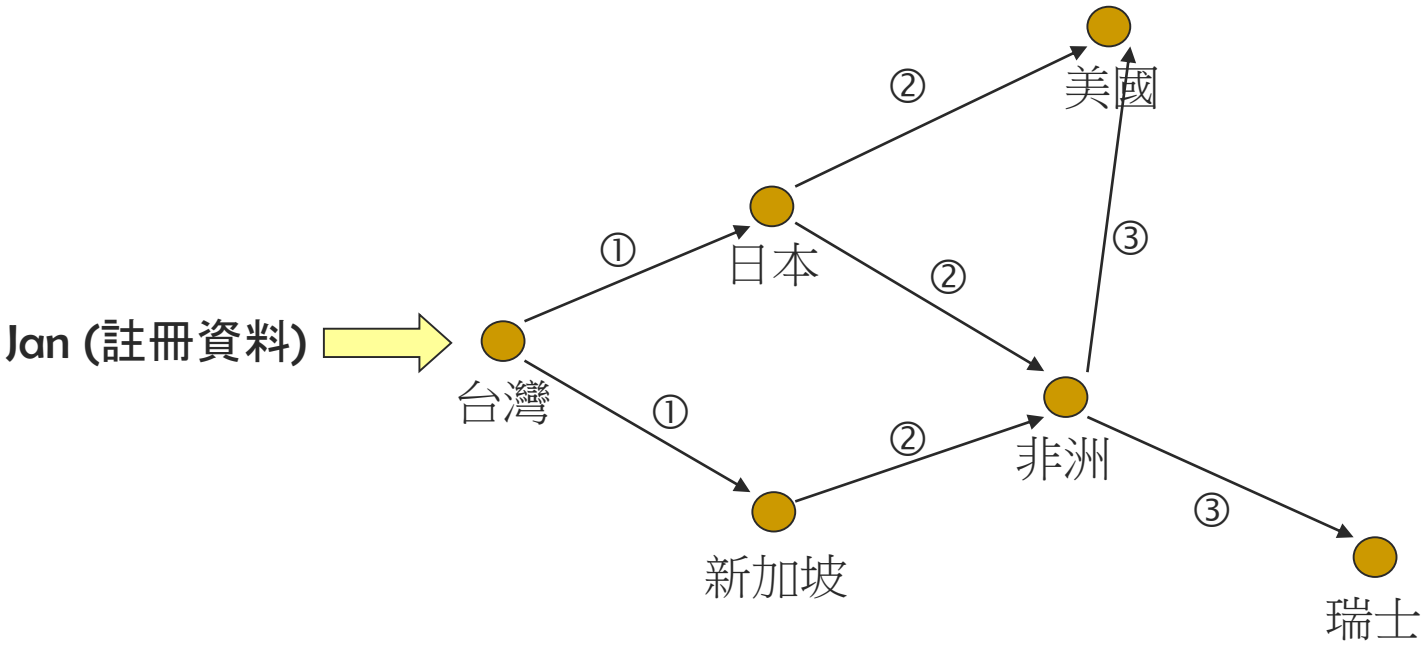
為何資料需要重覆存放?

- 增加效率
- 備份

- 避免資料重覆存放 (Data Redundancy)
- **避免資料不一致 (Inconsistency)**
  - **若資料庫無法完全無重覆，則需透過DBMS做良好的控制，以減少產生資料不一致的現象。**
  - **資料庫具有傳播更新 (Propagating update)的特性，確保任何更新的動作會自動地更新到其它重複的資料，以減少不一致現象。**
- 資料的分享 (Multi-User)
- 標準的確立
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- 資料獨立性 (Data Independence)

何謂傳播更新?

- 從附近的點往外傳播



- 避免資料重覆存放 (Data Redundancy)
- 避免資料不一致 (Inconsistency)
- **資料的分享 (Multi-User)**
  - **不同的應用可以共享資料庫的同一份資料，不必像檔案系統，要為每一個應用建立自已的檔案。**
- 標準的確立
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- 資料獨立性 (Data Independence)

- 避免資料重覆存放 (Data Redundancy)
- 避免資料不一致 (Inconsistency)
- 資料的分享 (Multi-User)
- **標準的確立**
  - **藉由資料集中化及存取界面標準化，容易建立公司內部資料的標準，亦可幫助企業間的資料交換。**
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- 資料獨立性 (Data Independence)

- 避免資料重覆存放 (Data Redundancy)
- 避免資料不一致 (Inconsistency)
- 資料的分享 (Multi-User)
- 標準的確立
- **妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)**
  - **資料庫的集中控制讓DBA更容易從事安全性(Security)與完整性(Integrity)的限制。**
- 資料獨立性 (Data Independence)

- 避免資料重覆存放 (Data Redundancy)
- 避免資料不一致 (Inconsistency)
- 資料的分享 (Multi-User)
- 標準的確立
- 妥善的安全性與完整性控制 (Security Constraints and Integrity Constraints)
- **資料獨立性 (Data Independence)**
  - **應用程式和資料是分開的。**
  - **實體儲存方式的改變或資料庫結構的改變不會影響到上一層次的應用。**

■ 資料庫系統的缺點

- **初期成本高** (軟體、硬體、教育訓練費用...等)
- DBMS為了定義與處理資料提供了一般性，因此可能為了某些特殊需求而浪費額外資源。
- **過多的限制**：完整性限制(Integrity)、安全性控制(Security)、並行控制(Concurrency)、回復(Recovery)等。且需要額外資源來確保這些限制的正常工作。
- 資料庫設計師不常遵守完整性限制

- **何時不必採用資料庫系統：**
  - **資料及其相關應用程式是非常簡單、且不常改變的。**
  - **不會有多使用者同時存取(No multiple-user access to data)**
  - **未來擴充可能性非常小**
  - **某些應用程式有非常嚴格的即時性需求 (Stringent, real-time requirements)**
    - **Because of DBMS overhead.**

Database Users

- 對於小型個人化資料庫而言，僅需一人就可以定義、建構與操作資料庫，且此類資料庫通常不會多人共享。
- 然而，以大型組織來說，會有許多人參與一個大型資料庫的設計、使用與維護，通常會有以下的人物：
  - Database Administrators (DBA) 資料庫管理師
  - Database Designers (DBD) 資料庫設計師
  - End Users 終端使用者
  - Software Engineers (SE) 軟體工程師

- **Database administrators (DBA):**
  - **DBA負責定義、建立、維護實際的資料庫，從技術性觀點提供系統整體的控制。**
  - **DBA的任務：**
    - **決定資料庫的資訊內容 – 定義資料庫的概念層次**
    - **決定儲存結構、存取策略 – 定義資料庫的內部層次**
    - **與使用者連絡 – 定義外部層次，以及概念層次與外部層次的對應**
    - **定義安全性與完整性限制**
    - **錯誤管理– 定義資料庫的備份(Backup)、回復(Recovery)的策略**
    - **監測執行效能及應付需求的改變**
- **Database Designers (DBD):**
  - **與所有可能會去使用資料庫的使用者們溝通，了解他們的需求，以便設計出符合他們的資料與處理需求的資料庫觀點 (View; 或稱景觀)。最後再將其他使用者的觀點一起分析與整合，以滿足所有使用者的需求。**

■何謂 View (觀點、景觀)？

- 由其它實際存在於資料庫的表格所衍生出來的關聯表格。
- 不同的是，View不需要以實體的形式存在，因此可視為一個虛擬表格 (Virtual Table)。

老闆

男/女朋友

父母

姓名	修習課程	成績	工作經驗
⋮	⋮	⋮	⋮

姓名	住址	電話	興趣	男女朋友	個性
⋮	⋮	⋮	⋮	⋮	⋮

姓名	修習課程	成績	男女朋友
⋮		⋮	⋮

虛擬表格

學生資料庫

實際表格
(Base Table)

姓名	住址	電話	修習課程	成績	興趣	男女朋友	工作經驗	個性	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Meta-data

- 儲存於**系統目錄**(System Catalog)中，非資料內容本身，**用來描述資料的資料**，稱之為**中繼資料 (Meta-data)**，也有相關書籍稱其為**詮釋資料、敘述性資料**。
- 以下列的學生基本資料表為例：
  - 表頭的部份即為Meta-data
  - 主體的部份包含了三筆Data

學號	姓名	系別	年級	生日	地址
001	張三	資管	2	3.18	台北
002	李四	企管	3	3.19	台中
003	王五	人管	4	3.20	台南

- 因此，資料庫系統具有**自我描述(Self-Description)**的特性。

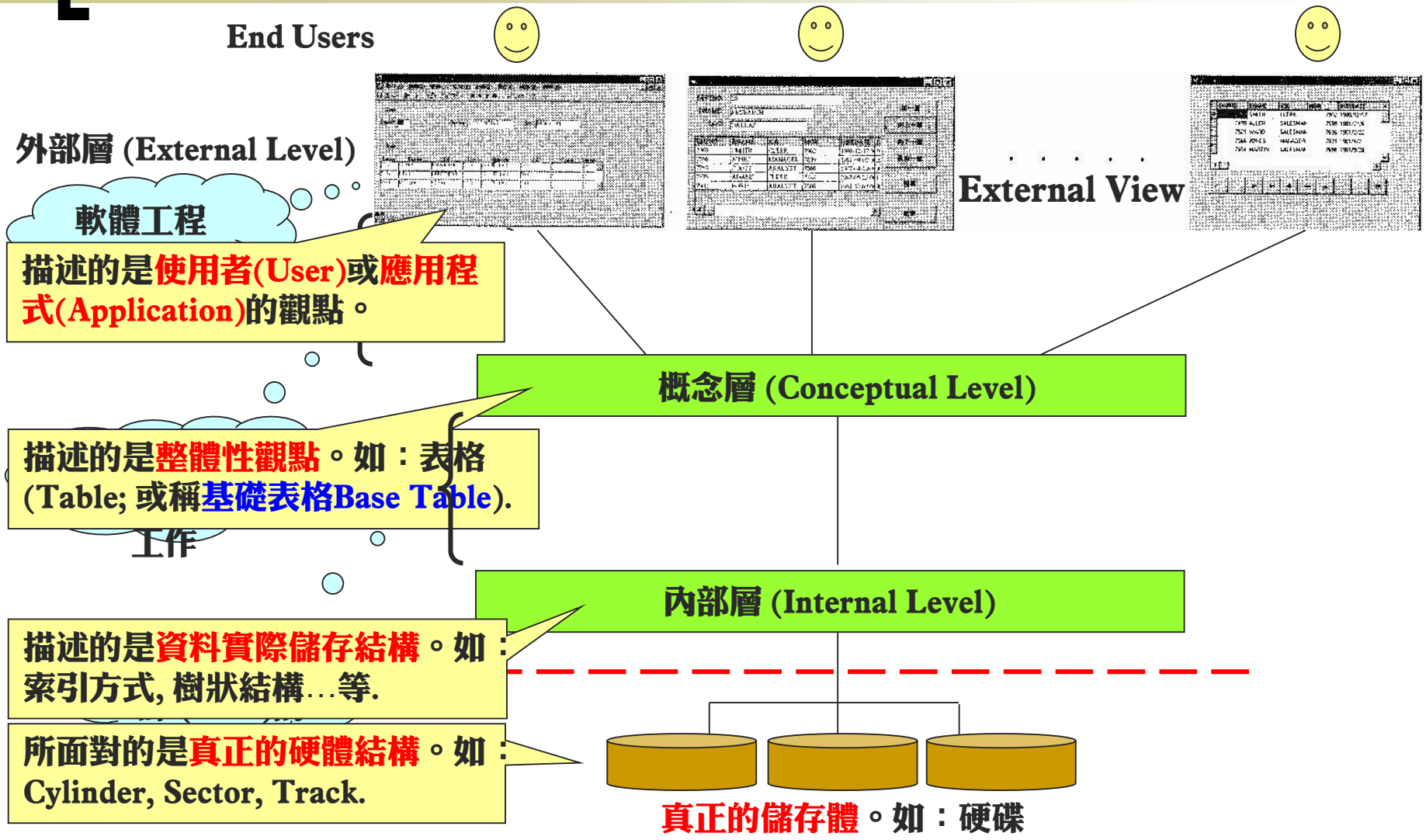
- 前例之Meta-data事實上不只有表頭資料而已，同時也包含了對於資料庫系統的定義與描述，如
  - 資料項目的型態及儲存格式，如：姓名→char(10)、薪水→int...
  - 資料間的關係及各種限制，如：員工編號不可重覆...
  - 資料庫實際儲存結構，如：循序檔、索引檔、雜湊檔...
  - ...
- 表格當中所存放的每一筆記錄，也可稱之為實例 (Instance)
  - 上述表格共有三個Instance
- Meta-data不常變動，Instance較常變動。
- Meta-data亦可稱為Schema (綱要、綱目)

- 坊間資料對Metadata與Schema的說明：
  - 【說法1】Meta-data亦可稱為Schema
  - 【說法2】資料間許多的限制與關係，在Schema中無法表現，故Schema與限制之描述方可稱為Metadata。
  - 【說法3】對資料的定義(描述)稱為Schema，定義資料之後會產生所謂的Metadata。
- 杰哥觀點…勿戰(要戰我也不理你…)
  - 區別兩者是否相同，對於學習資料庫系統管理來說，不會有太大的影響。
  - 不用費神去思辨。

Three-Schema Architecture

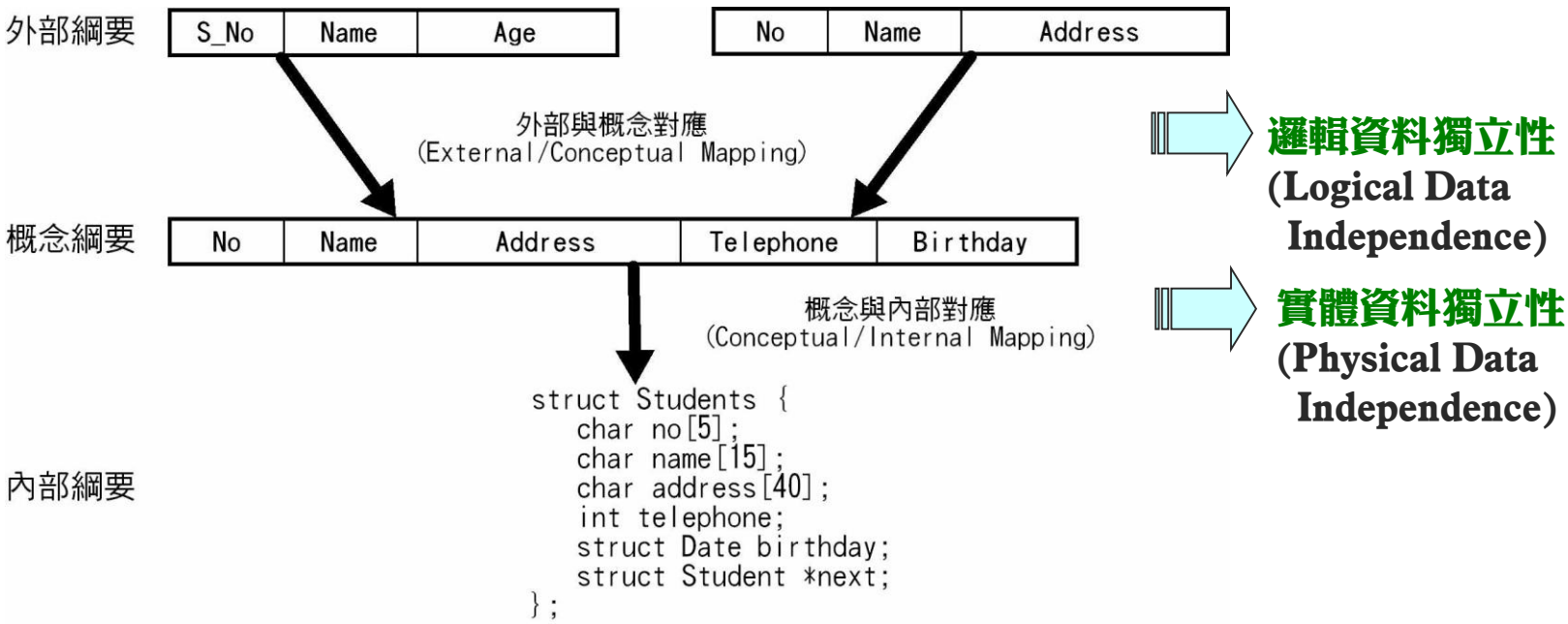
- 1970年代，ANSI與SPARC兩個研究小組提出資料庫三層架構，稱為**ANSI/SPARC架構**，以三個不同的層次(角度)來看待所架設的資料庫系統。主要分成：
  - **外部層 (External Level)**
    - 外部層所看到的**通常只是資料庫的部份資料**。
    - 是以**一般使用者**觀點所看到的資料。
    - 不同使用者有不同的View，所以看見的資料亦有所不同。
  - **概念層 (Conceptual Level)**
    - 概念層所看到的是**資料庫儲存的整體性資料**。
    - 是以**DBA**觀點所看到的完整資料庫系統。
  - **內部層 (Internal Level)**
    - 內部層所看到的是**資料實際儲存於資料庫內的資料結構或檔案組織**。

ANSI/SPARC架構



Three-Schema Architecture

- 由ANSI/SPARC三層架構的討論，可知對於資料庫系統內，用以描述資料的資料 (即：Metadata或Schema) 也分成三個不同的層次，稱為三層式綱要架構(Three-Schema Architecture)。



- **External schemas (外部綱要)**
  - At the external level
  - 描述個別使用者的觀點 (Describe the various user views)。
  - 最接近使用者的層次，資料於不同使用者有不同的呈現。
  - 隱藏不需要的部份，對個別使用者只顯示其感興趣、或有權限讀取的部份。

Student_Age_View

S_No	Name	Age

Student_Label_View

No	Name	Address

- **Conceptual schema (概念綱要)**
  - At the conceptual level
  - 呈現出資料庫全部資訊的內容
  - 以全體使用者為主，描述資料庫整體的架構、資料型態、關係與操作限制

Students

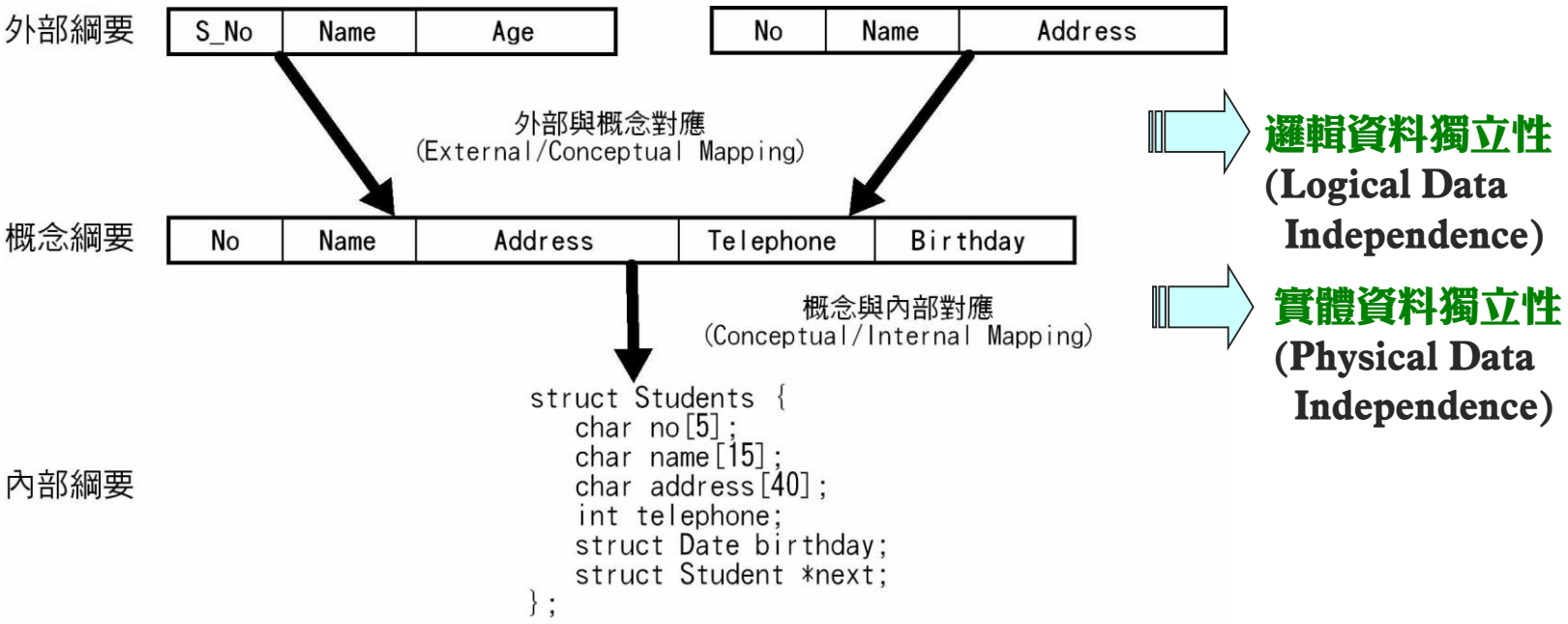
No	Name	Address	Telephone	Birthday

- **Internal schema (內部綱要)**
  - At the internal level
  - 描述有關於資料實際儲存方式，如：
    - 資料儲存結構與存取路徑的相關細節。
    - 紀錄資料儲存格式、存在哪些索引 (Index)、儲存的欄位如何呈現、與儲存紀錄的實體順序
  - 最接近機器，但不涉及實際儲存媒體的硬體結構

```
struct Students {
    char no[5];
    char name[15];
    char address[40];
    int telephone;
    struct Date birthday;
    struct Student *next;
};
```

資料庫綱要間的對映

- 各層次間的對映主要有兩種：
  - 外部與概念對映（External/Conceptual Mapping）：
    - 所有外部綱要都是對映到概念綱要，以便資料庫管理系統知道如何將外部層的資料連結到那一部分的概念綱要。
  - 概念與內部對映（Conceptual/Internal Mapping）：
    - 概念綱要對映到內部綱要的關聯，以便資料庫管理系統可以找到實際儲存裝置的記錄資料後，建立概念綱要的邏輯結構。


外部綱要
S_No
Name
Age
No
Name
Address
外部與概念對應
(External/Conceptual Mapping)
概念綱要
No
Name
Address
Telephone
Birthday
概念與內部對應
(Conceptual/Internal Mapping)
內部綱要
struct Students {
char no[5];
char name[15];
char address[40];
int telephone;
struct Date birthday;
struct Student *next;
};
邏輯資料獨立性
(Logical Data Independence)
實體資料獨立性
(Physical Data Independence)

- **Data Independence (資料獨立性)**
  - 指改變資料庫系統的某一層次綱要時，不會影響到較高層次的綱要，即：不需要跟著修改較上層的綱要。
  - 某一層綱要與其上層綱要間相互獨立，相關的資料庫系統改變只需透過對DBMS適當的管理或修改即可，不用更改資料庫的相關程式。

- Two types of data independence:
  - Logical Data Independence (邏輯資料獨立性)
    - 當改變概念綱要時，不需要跟著改變外部綱要及其應用程式。外部層與概念層之間獨立
    - 時機：如，擴大資料庫規模時
  - Physical Data Independence (實體資料獨立性)
    - 當改變內部綱要時，不需要跟著改變概念綱要和外部綱要。
    - 外部層與內部層之間獨立且概念層與內部層之間獨立
    - 時機：如，重組某些實體檔案結構

Centralized and Client/Server Architectures for DBMS

- 資料庫系統處理架構的演進：
  - 集中式處理架構 (Centralized Processing Architecture)
  - 主從式架構 (Client/Server Architecture)
  - 多層式架構 (Multi-Tier Architecture)
  - 分散式資料庫系統 (Distributed Database System)

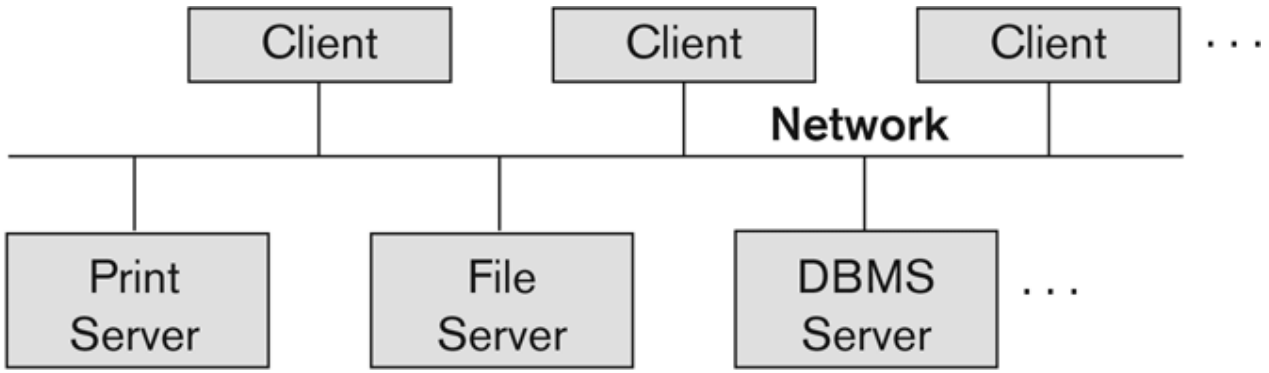
Centralized Architectures

- 是早期的系統架構，使用大型電腦主機 (Mainframe Computer) 為所有的系統運作提供主要的處理。
  - 相關運作包含：DBMS software, hardware, application programs and user interface processing software.
- 大多數使用者是透過電腦終端機 (Computer Terminal)來存取系統資料。
  - 此終端機沒有處理資料的能力，僅提供顯示的能力。
  - 因此，使用者的所有處理工作皆需經由網路傳輸到遠端的主機來處理。
- 由於是將所有的Server放在單機(Local端)上執行，因此稱之為One-tier (單一階層)架構。

Basic Client-Server Architectures

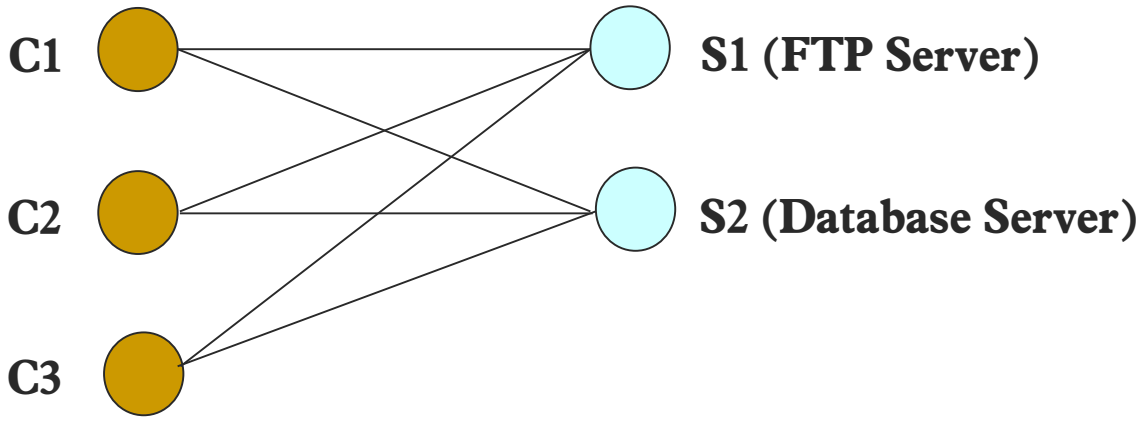
- 近期由於硬體價格下降，大多數的使用者都以個人電腦來取代原本的終端機。
- 漸漸地，DBMS開始將部份的工作轉移到User端來執行，進而發展出 client/server DBMS架構。又可稱為Two-tier (雙階層) 架構

Figure 2.5
Logical two-tier client/server architecture.



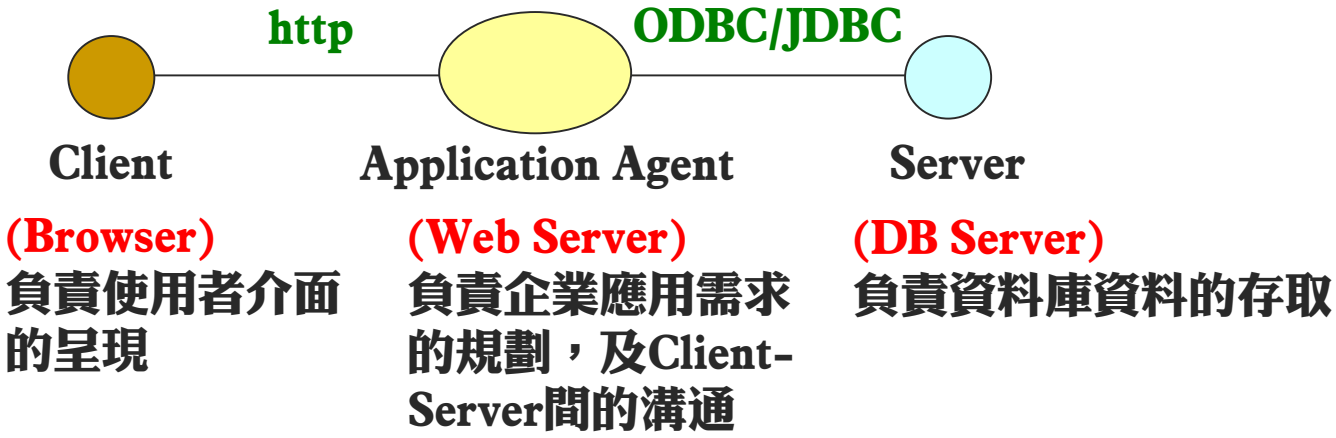
- 若Client端的使用者需要連接到DB Server時，Client端的電腦就需要有能力與Server端的DBMS做連線。
  - ODBC (Open Database Connectivity)
    - ODBC是一種由Microsoft推動、介於Server端資料庫系統與Client端應用程式之間的程式介面(Application programming interface; API)。
    - 讓Client端的應用程式可以很容易地和各種不同之關連式資料庫系統的DBMS連結並取得資料庫內的資料。換句話說，ODBC是一種讓各種資料庫都具有相同的存取資料介面的程式。
    - 因為 ODBC 是一個應用程式介面，所以要使用 ODBC 時，在Client端必須安裝ODBC 驅動程式。而在Server端的資料庫系統，如Oracle、MS SQL、MySQL等，也須有相對應的ODBC驅動程式。有了ODBC後，程式設計師在設計不同資料庫系統的程式時，就不必將重新撰寫程式，只要變動與 ODBC 所連結的介面即可，可以節省不少麻煩。
  - JDBC
    - An Java API. For the Java programming language
    - 由Sun推動

- **Client/Server (Two-tier) Architecture的問題：**
  - **當Server種類變多，Client也需因此而安裝多個介面**
    - **Client對於不同的Server可能會有不同的企業應用需求，因此需要安裝不同類型的應用程式來執行。**
    - **不同應用程式的更新非常麻煩。**
  - **當Client增多時，Server的負荷也會變重**
    - **後端Server會窮於應付，且Disk I/O也會非常頻繁，執行效率變差。**

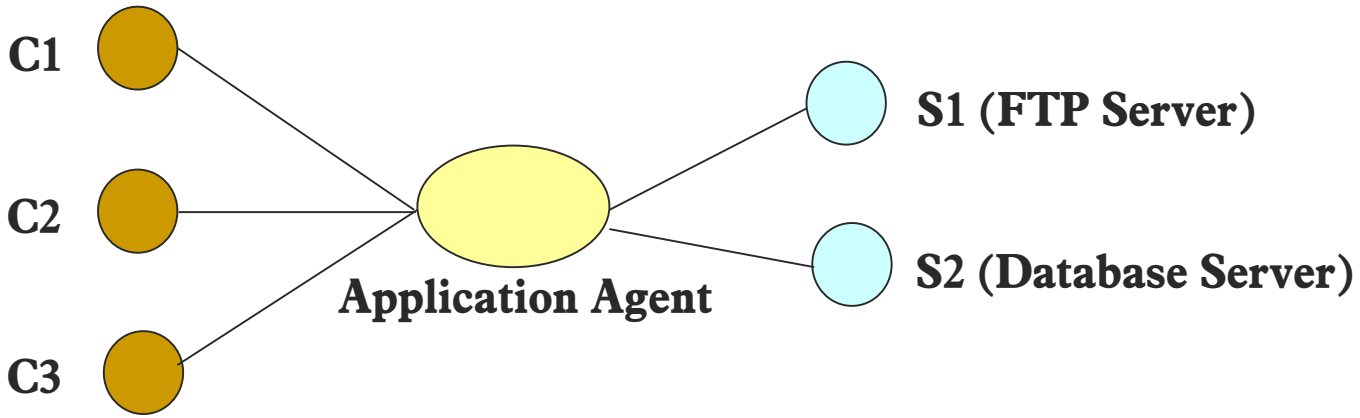


Three Tier Architecture

- 將應用程式代理者 (Application agent) 置於Client與Server中間，存放企業應用邏輯 (Business logic)，以處理Client與Server間往來的業務。
- Client較簡化，因此系統需求較低。
- 整合後端不同的Server，以統一的方式呈現內部的資料。



- **Three-Tier Architecture的問題：**
  - **Application Agent將會是瓶頸**
  - **解決方法：**
    - **中間層多設幾層 (即：Multi-tier Architecture)**

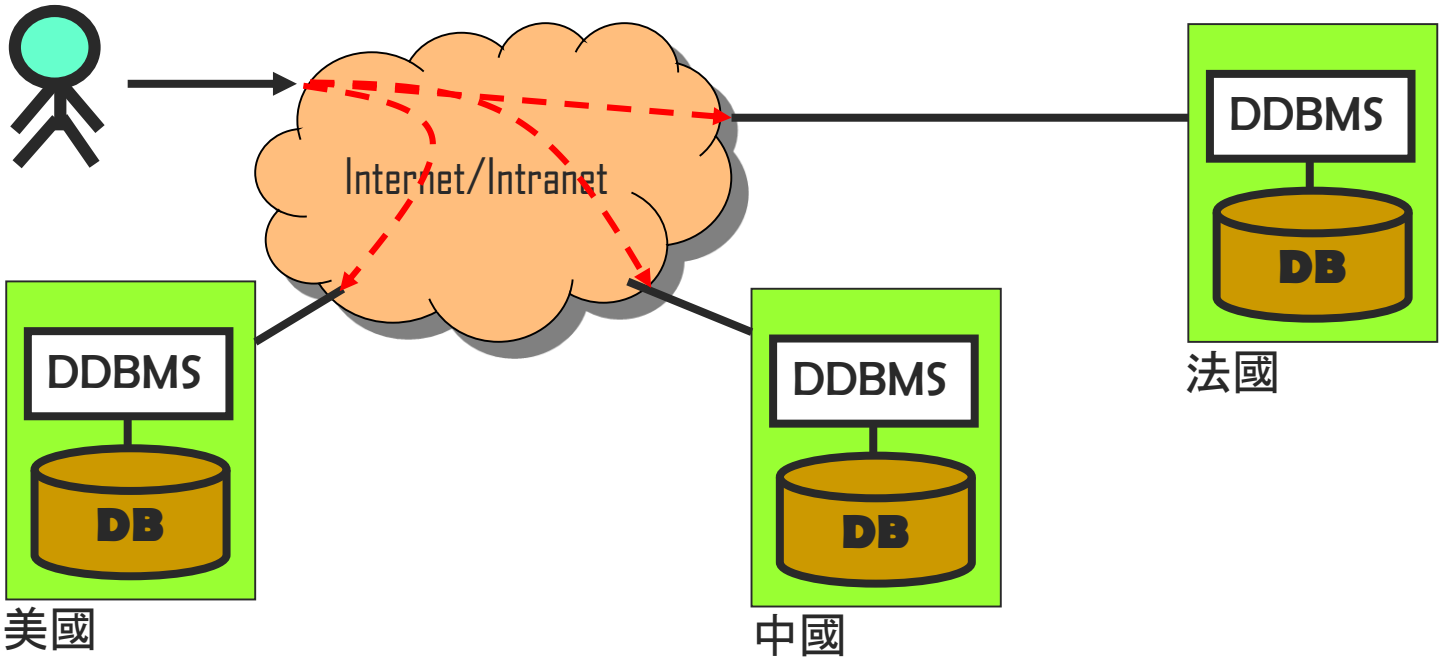


- 比較

	優點	缺點
Client/Server	■ 可藉由Client端本身的運算能力，分擔Server端處理資料的負荷。	■ 針對不同類型的Server，需要設計不同的Client端介面。 ■ Client愈多，Server的負擔越重。
3-tier architecture	■ 可由中間的Application Agent整合後端不同Server，Client端只需面對單一的Agent介面	■ 若規模不斷擴大，中間端將成為瓶頸。

分散式資料庫系統 (Distributed Database System)

- 分散式資料庫系統在邏輯上可視為**單一資料庫系統**，但實際上資料卻是分散在網路的多個地方(Database Systems)，透過分散式資料庫管理系統來從事資料的管理與控制。
  - 例：某一全球性大公司的主管，想知道目前公司有哪些客戶!!



- 分散式資料庫系統的透通性
  - 透通性 (Transparency)：不需要了解細節，即可正常運作的特性!
  - 種類：
    - 存取透通性 (Access Transparency)
    - 位置透通性 (Location Transparency)
    - 網路透通性 (Network Transparency)
    - 同步透通性 (Concurrency Transparency)
    - 重覆透通性 (Replication Transparency)
    - 錯誤透通性 (Failure Transparency)
    - 移動透通性 (Mobility Transparency)