國立中山大學資訊管理研究所

資料庫系統專題

課程講義（中文註釋版）[*]

國立中山大學資管系
黃三益教授

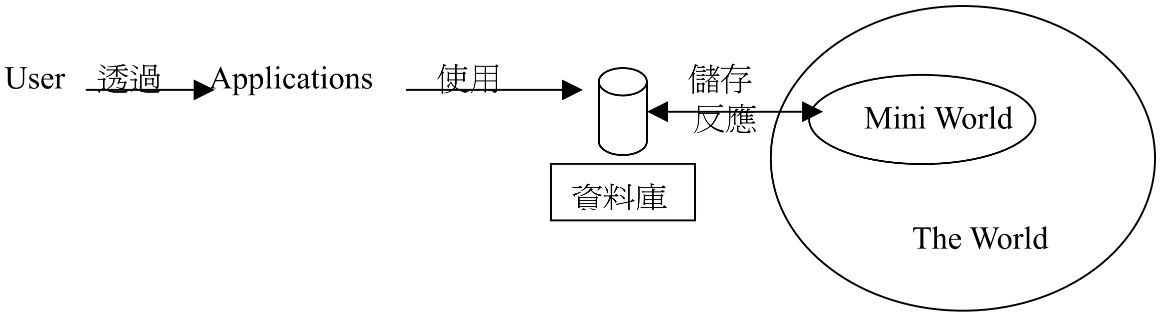
9/19/2002

[*] © Copyrights 2002, All Rights Reserved

一、Overview(Chapter 1 and 2)

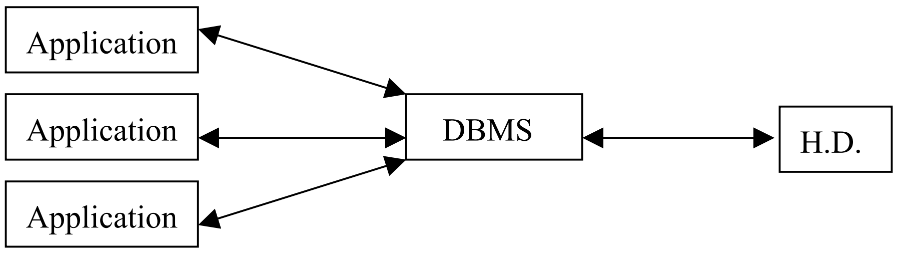
(一) What is Database?

- A database represents some aspect of the ***real world***, called the ***mini-world***. Changes to the mini-world are reflected in the database.
- The data in a database is logically coherent.
- A database has an intended group of users and some preconceived applications.



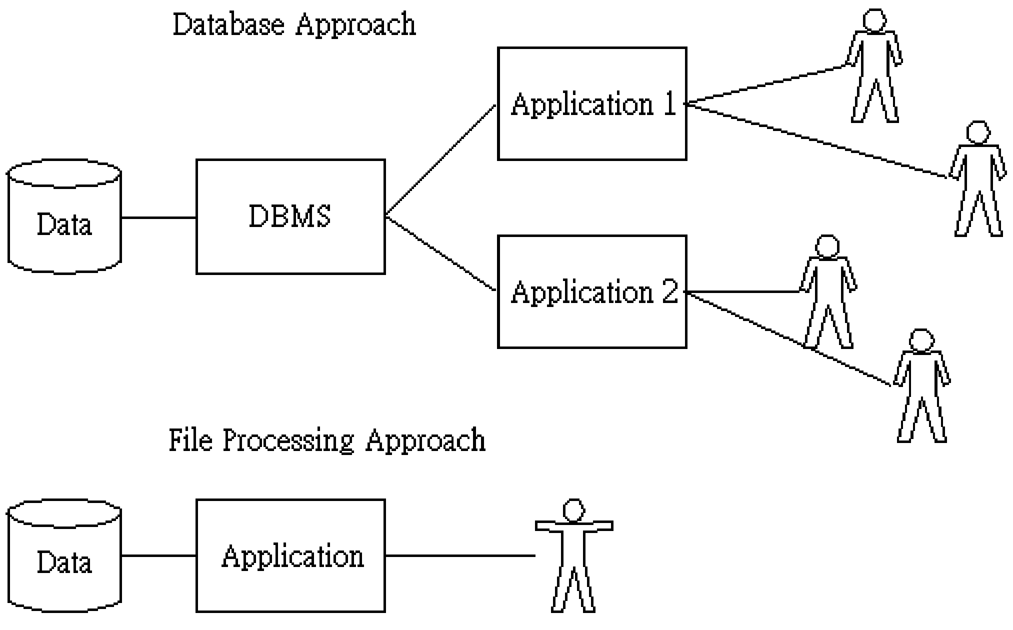
- 例如：人事資料庫，反應現實中個人狀況，將有選擇性的資料儲存到資料庫內，會有更新、刪除，
  會有一群使用者以及可供使用的程式。

(二) What is a DBMS (DataBase Management System)?



- 資料庫管理系統提供三種功能:
  - Define a Database：載明資料項目、型態及架構。(決定資料架構)
  - Populate or constructing a Database：將資料儲存於儲存媒體之過程（填入資料)。
  - Manipulate a Database：存取、查詢、更新及產生報表 (使用資料)。
- See Textbook's Figure 1.1: a simplified database system environment.

（三）What makes the database approach different from the file-processing approach?



- See here for a graphical representation.
- 資料庫包含了 Data (資料) 和 Meta Data（詮釋或目錄型資料：也就是用來解釋資料項型態和儲存型式的資料），Meta Data 又叫 catalog.
- 程式資料獨立(program-data independence)：資料內部結構的改變不會影響到應用程式。The structure of data files is stored in the DBMS catalog separately from the access programs.
- The DBMS supports multiple views of the data. The data perceived by the users may be real or derived. Views also ease the job of security enforcement. 一份資料可有多種的呈現方式
- Transaction processing：Data can be shared and also updated in a controlled manner so as to ensure that concurrent transactions operate correctly. (資料共享，具有未完成交易資料之還原功能)

（四）Who Play in The Database Environment?

- The DBA (database administrator): 管理資料庫軟硬体設備及資源，負責授權、協調、監視資料庫之使用
- The database designer/system analyst: 定義資料庫之資料項及其結構
- The system analyst/programmer: 應終端使用者需求做系統分析及撰寫程式，軟體工程師
- Sophisticated users: 使用系統所支援的查詢語言來完成資料需求
- End users/Naïve users: 使用應用程式或做查詢或更新資料

（五）When You May not Use a DBMS?

- 資料庫結構單純且很固定不變 (例如:Web log)。
- 效率要求很快速的 (例如:國防系統)。
- 不必要提供給多使用者同時存取的功能 (例如:中小企業會計系統)。

（六）DBMS Architecture and Data Independence

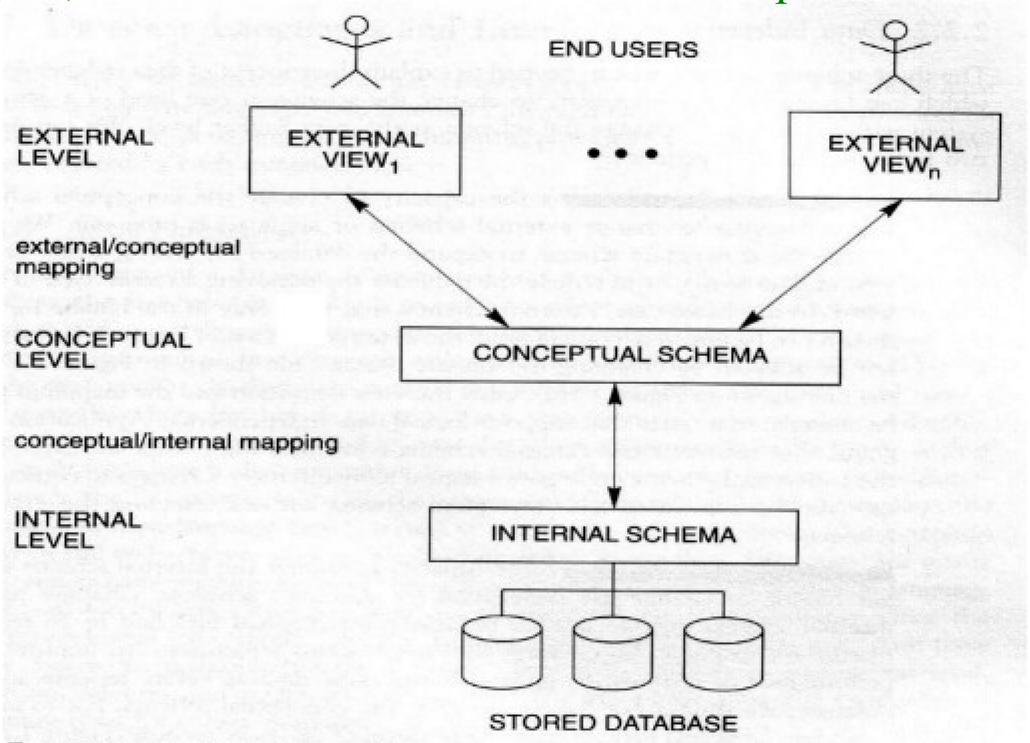


Figure 2.2 The three-schema architecture.

See here for a grpahical representation.

The three-schema architecture

Schema type	Meaning	Target
external schema	對某一 user group 描述定義要用到的資料欄位，其餘的隱藏不定義	Application programmer
conceptual schema	描述所有資料欄位之名稱，型態、資料間的關係、限制等	DBA
Physical schema	描述資料庫 Physical storage structure 例： ID：50~53 bytes NAME:5~10 bytes	DBMS designer

1.The description of a database is called the **database schema**.

2.The internal schema describes the physical storage structure of the database. It is seen by the DBMS designers.
3.The conceptual schema hides the details of the physical storage structures when describing entities in the database. It is seen by the DBA.
4.The external schema describes the part of the database in which a particular user group is interested and hides the rest. It is seen by application programmers.
5.Conceptual and External 兩個 schema 可使用相同資料模式 (例如:關聯式模式)

Data Independence

1.資料獨立：上述三層 schema 中某一層 schema 改變時不必要連帶改變其上一層之 schema。
2. Logical data independence: The change of the conceptual schema does not change external schemas or application programs.
3. Physical data independence:The change of the internal schema without having to change the conceptual (or external) schemas.

（七）Database Language

- 為了支援上述三種綱目架構，理論上需下列三種定義語言：
  - Storage definition language（SDL）：用此種語言來定義 internal schema。
  - Data definition language（DDL）: ，用 DDL 語言定義 conceptual schema。
  - V iew definition language（VDL）：用此 VDL 定義 user views。
- 此外尚需 data manipulation language(DML)：資料庫建置完成且鍵入資料後須做查詢、新增、更新及刪除等步驟，就可用此種語言。不能單獨使用，必須附屬於應用程式裏，猶如應用程式所加入的一些語法，當 DBMS 接受到這些語法後，便可依據 DDL 在資料庫中取得資料，然後再透過 DML 將資料傳遞給應用程式。
- 目前資料庫系統(DBMSs)較常採用的語言是 DDL,VDL 及 DML 搭配使用。
- DMLs 有兩種主要類別：
  1. A high-level or nonprocedural DML：可由終端機以交談方式或放在一般程式語言裡使用，記錄尋找方式為一次一組(set-at-a-time)，使用於個別交談方式時叫做查詢語言(Query language)
  2. A lower-level or procedural DML：必須放在一般程式語言裡使用，記錄尋找方式為一次一個記錄 (record-at-a-time)，目前已很少使用。
- 在一般程式語言裡若包含了 DML 指令的話，此程式語言叫做 Host language，被包含的 DML 就稱為 Data sublanguage。
- 目前所普遍使用的資料庫語言 SQL 實際上包括了 DDL, VDL, 和 DML，此外，SQL 尚有語法來管理資料使用權線和交易(transaction)的使用。

（八）The Database System Environment

- The DBMS component modules

See Figure.2.3 in the textbook for a graphical representation.

- The Database System Utilities(公用程式)
  - Loading：It is to load an existing data files—such as text files or sequential files—into the database.
  - Backup：It creates a backup copy of the database.
  - File Reorganization(檔案重整)：To reorganize a database file into a different file organization to improve performance.
  - Performance Monitoring(效能監視)：To monitors database usage and provides statistics to the DBA, so as to decide whether or not to do the file organization.（看統計資料以決定是否做檔案重整）

（九）Classification of DBMSs(資料庫的分類)

- 依使用者個數區分：.個人用(Single user) v.s. 多用戶(Multi-user).
- 依資料庫存在的地方區分： A.集中式(Centralized):目前較普遍採用 B.分散式(Distributed):因穩定性尙値探討，未獲普遍採用.
- 依資料模式區分： A.關聯式(Relational): 最普遍，易懂，目前市場大.
  B.階層式(Hierarchical): 早期使用.
  C.物件導向式(Object-Oriented):較複雜，目前市場小
  目前正將物件導向式資料庫的觀念與關聯式資料庫，成為物件關聯式資料庫(Object-relational DBMSs).
- 依價格區分：數千～佰萬.
- 依用途區分：.一般用途(General purpose) v.s. 特殊用途(Special purpose).

（十）DBMS 發展史

- 1960's: File sytems (COBOL)
- 1970: Hierarhical DBMS (IBM IMS)
- 1980: Relational DBMS for mainframes (IBM DB2)
- 1985: Relational DBMS for workstations (Oracle, Sybase, Informix)
- 1990: Object-oriented DBMS (Gemstore, Objectstore)
- 1995: Personal DBMS (MS Access, Foxpro)
- 2000: DBMS with Object and OLAP features (SQL Server 7.0, Oracle 8)
- 2002: XML databases
- IMS DBMS 銀行界採用最多，至今尙有採用者.

二、ER Model (Chapter 3)

（一）Introduction

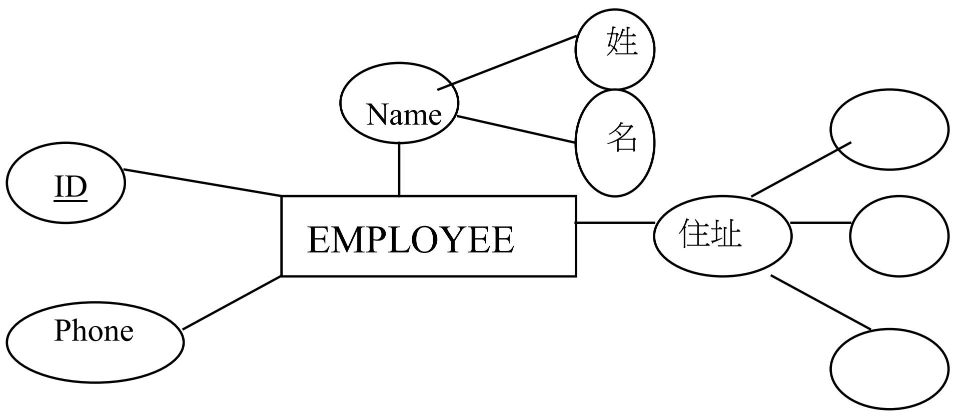
- 將系統所需資料稍微結構化的描述出來，也就是將非結構化資料以結構化方式表示，是美籍華人 Peter Chen 提出的，全名：Entity Relationship Model。
- It is used as a high level conceptual data model(概念資料模型).
- See Figure 3.1 for the phase of database design.

（二）The ER Concept

- Entities：是真實世界裡的一項事物，獨立存在的**實体**或概念，這世界上有許多的 entities.
- Attributes：一個 entity 有許多性質(properties),我們稱其每一種性質爲屬性(attribute)。
- Attribute values：每一實体的每一屬性會有值，所有實体的屬性值都儲存在資料庫裡。
- 屬性值可能是複合型 (composite) 或簡單型 (simple)。
- 屬性值也可能爲多值 (multi-valued) 或單值 (single-valued)。
- 推導屬性 (Derived attribute)：推導屬性的值是由其他屬性（stored related attribute）推導而得。
- 空値(Null value)有三種含意： A.未知(Unknown) B.不適用(N/A, not applicable), C.未知是否存在值 (The existence of value unknown)

（三）Grouping Entities

- 資料庫通常包含很多類似的實体群。
- 一個實体型態(Entity Type)定義了一組具有相同屬性(attributes)的實体。
- .某一特別實體型態的實體組成一個集合，就叫做 extension or warehouse of the entity type.
- 在 ERD(ER diagrams)內 entity type 是用一個矩形框表示，attribute 是用一個橢圓形框表示。
- Key attributes of an entity type：某實体型態的某一屬性之值可唯一表示其個別實体。
- 當一實体型態的 key 是由多個 attribute 一起構成時，這些 attribute 可組成組合屬性，它變成此實体型態的一個 key attribute，某些實体型態不只一個 Key attribute。
- 在 ERD(ER diagrams)內，每個 key attribute 是於名稱加一底線表示。
- Key attribute 的意思是表示，不管這 Entity type 的 extension 將來在任何情況下，不同 entity 的 key attribute 値都不一樣，意即其值是唯一的。
- Domains of attributes（屬性的値域）：對每一個別實体的某一屬性可指定其值的範圍（一組值）。
- 在 ERD(ER diagrams)內沒有將屬性値域表示出來，而 Multivalued attributes 用雙橢圓表示。
    - 實体組成：



- EMPLOYEE是一個Entity Type，其extension 就是所有EMPLOYEE實體所組成的集合
- Name 及住址均為組合屬性(Composite attribute)
- Phone 可能為 multi-valued (視定義而定；若是則以雙橢圓形框來表示)

- An entity type may have no key, in which case it is called a weak entity type.

- 名詞圖示：

<u>ENTITY TYPE NAME:</u>	EMPLOYEE
<u>ATTRIBUTES:</u>	Name, Ages, Salary
<u>ENTITY SET:</u> (EXTENSION)	e1. (John Smith,55,80k) e2. (Fred Brown,40,30k) e3. (Judy Clark,25,20k) • • •

（四）An exercise

- 依據課本 3.2 節例題做一 Preliminary design of entity types:
  1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and ManagerStartDate. Locations is the only multivalued attribute. We can specify that each of Name and Number is a key attribute, because each was specified to be unique.
  2. An entity type PROJECT with attributes Name, Number, Location, and ControllingDepartment. Each of Name and Number is a key attribute.
  3. An entity type EMPLOYEE with attributes Name, SSN (for social

security number), Sex, Address, Salary, BirthDate, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name-FirstName, MiddleInitial, LastName-or of Address.

4. An entity type DEPENDENT with attributes Employee, DependentName, Sex, BirthDate, and Relationship (to the employee). 請參考 Figure 3.8

- 另外參考 Figure 3.2 ER schema diagram for the COMPANY database。

（五）Relationships, Relationships Types, Roles, and Structural Constraints

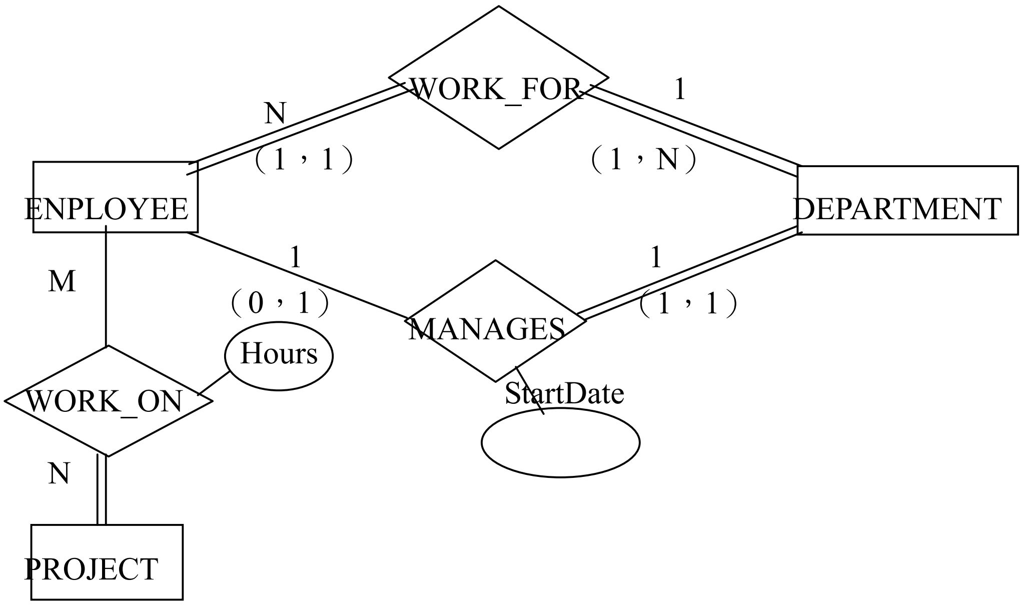
Relationship Types , Sets and Instances

- A Relationship：A relationship (instance) is an association（聯合）on entities.
- A Relationship Type ：A relationship type on *n* entity types defines a set of relationships among *n* entities.
- In ERD relationship types are displayed as diamond-shaped boxes.
- 參考課本 Figure 3.9 & 3.10，說明一些關係的 instances。
- Degree of a relationship type ：It is the number of participating entity types.（參與此關係型態的實体型態數目）
- A relationship type of degree 2 is called binary relationship type（二元關係型態）, A relationship type of degree 3 is called ternary relationship type（三元關係型態）.
- If the same entity type participates more then once in a relationship type, it may play a different role for each participation. In this case, role name can be attached. Such relationship types are called **recursive relationships**. （參考課本 Figure 3.11）

Structural Constraints on Relationship types

- Cardinality Ratio(基數比)：It specifies the number of relationships (instance) that an entity can participate.
- For binary relationship types, there are three cases:
  - 1:1 → See Figure 3.12
  - 1:N → See Figure 3.9
  - M:N→ See Figure 3.13
- Participation Constraints：
  - Total: The participation of an entity type in a relationship type is total if every entity must participate in at least a relationship.（**existence dependency**）
  - Partial: If not total.
- We will refer to the cardinality ratio and participation constrains, taken

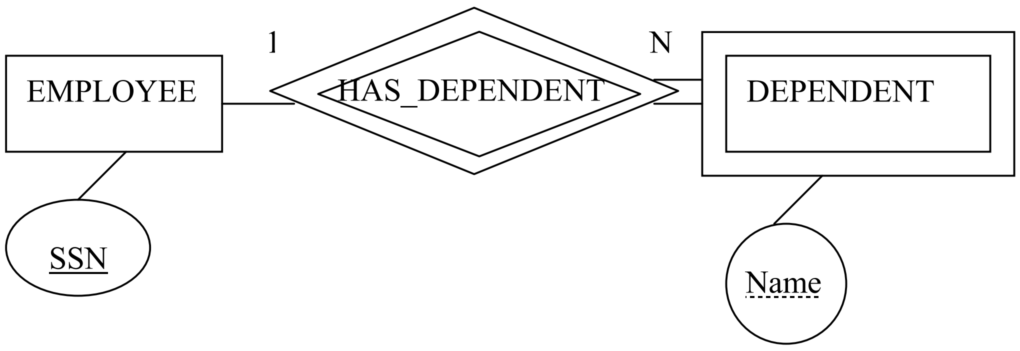
together, as the structural constraints of a relationship type.

- .關係型態也可以有屬性(Figure 3.13)，其屬性在 1:1 情況下可移到（migrated to）任一邊的 entity，在 1:N 情況下可移到 N 邊的 entity，在 M:N 情況下則不能移到任一邊的 entity，應屬於關係型態。
- (min，max)是參與情形的另一種表示法，可以精簡的提供詳細資料，(min max) 方式來表示相關的 entity type 之 structural constraint，where 0<=min<=max. It means the participating entity must participate in at least min and at most max relationship (instances)，min = 0 表示部份參與，min > 0 表示全部參與。
- 

- 參與情形：雙線表示 Total，單線表 Partial
- 員工一定要隸屬部門，部門一定要有經理
- Hours 屬性放在 EMPLOYEE 或 PROJECT 上都很難說清楚何人在何計畫做了幾小時，故放在 relationship 上較佳
- 計劃一定要有人做，但不是每個人一定要作計劃

（五）Weak Entity Types

- 某些 Entity type 沒有 key attribute，稱爲弱實体型態(Weak Entity Types)。
- Weak Entity Type 須依某種關係型態（稱爲 identifying relationship type）靠在另一實体型態上（稱爲 identifying entity type 或 owner entity type），它在此辨識關係型態上會是 total participation constraint。
- 弱實体型態的 key 可以是是其 owner entity 的 key 加上本身的某些識別屬性組合而成，這些識別的屬性組稱爲 Partial key。
- 並非每個 total participation 都會產生一個弱實体型態，例如美國的 driver liscense and person。
- 在 ERD 內，弱實体型態用矩形雙框線表示，辨識關係用菱形雙框線表示：



- 眷屬的 key 爲 Name + SSN
- 弱實体型態有時候能用複合式多值（composite，multi-valued）的屬性取代，(想想看何種情況?)。

Ex. Refine COMPANY database by taking relationships into account

1. MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation. The attribute StartDate is assigned to this relationship type.

2. WORKS_FOR, a I:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.

3. CONTROLS, a I:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users.

4. SUPERVISION, a I:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervises role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.

5. WORKS_ON, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.

6. DEPENDENTS_OF, a I:N relationship type between EMPLOYEE and DEPENDENT,which is also the identifying relationship for the weak entity type DEPENDENT.The participation of EMPLOYEE is partial, whereas that

of DEPENDENT is total.

- For solutions, please refer to Figure 3.2 or 3.15.

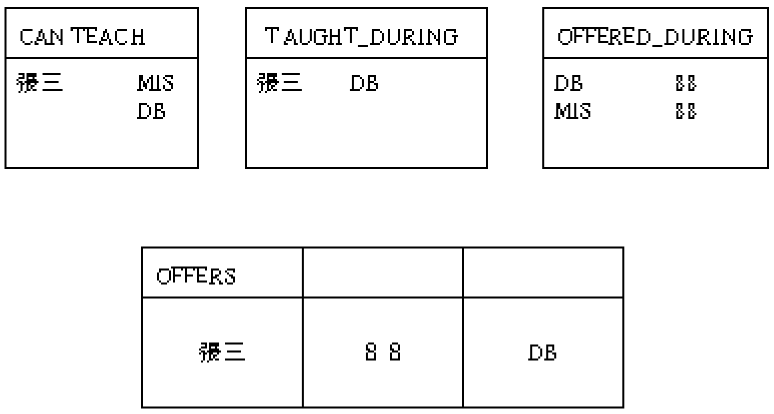
（六）Naming Conventions, and Design Issue

Proper Naming of Schema Constructs（習慣命名方式）

- Entity type 及 Relationship type 的名稱要用大寫字母（uppercase letters），Attribute 的名稱第一個字母大寫（capitalized），Role 的名稱要用小寫字母（lowercase letters）。
- Entity type 的名稱用名詞，Relationship type 的名稱用動詞，Attribute 的名稱用不同的名詞。
- 在畫 ERD 時內容文字要盡量符合人的讀書習慣，例如：A entity → relationship → B entity 的架構，要讓看的人由左而右或由上而下很像在看一句話一樣。
- 參考課本 Figure 3.14 for a summary.

（七）Relationship Type of Degree Higher Than Two

- ternary = binary + 1.
- 盡量不要用超過二元的關係型態，只有在用二元的關係型態無法充分表示實体間關係時，如: （STUDENT, COURSE, TEXT-BOOK）可用兩個二元關係來表示。
- 一個三元的關係型態不相等於三個二元的關係型態，通常一個三元的關係型態所表示的資訊多於三個二元的關係型態所表示的資訊（See Figure 4.13, 4.14, and example）。



CAN TEACH	
張三	MIS
	DB

TAUGHT_DURING	
張三	DB

OFFERED_DURING	
DB	88
MIS	88

OFFERS		
張三	8 8	DB

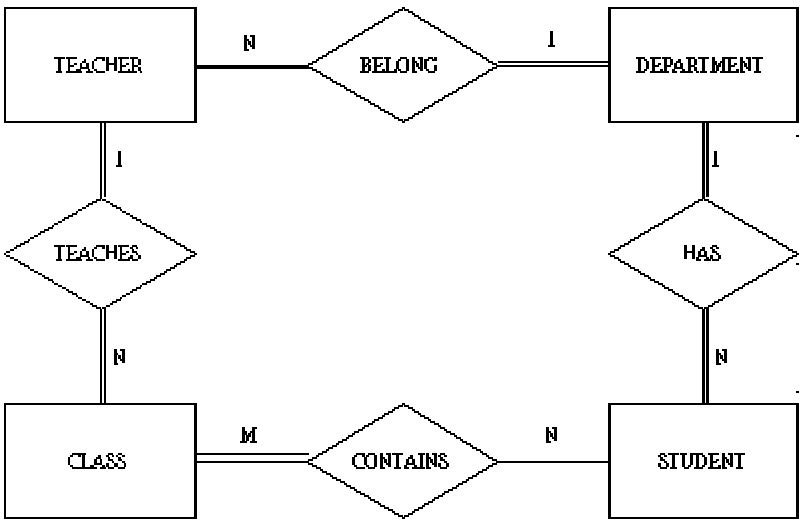
Example

（八）Exercises

練習一：Model a teaching system that includes the following entities：

1. DAPARTMENT
2. TEACHER
3. CLASS
4. STUDENT

（※考慮專任老師的情況）



此圖繪出後，藉此再找 USER 詳談，可能會引出更多需求，因而繪出更合乎實際的需求關係。

練習二 ：假設你要去 model 一個醫療系統，經過訪談，你得到以下的需求：

- 有 4 個 entity types: 醫師，住院病人，檢驗項目，病房。
- 醫師有三個 attributes: 代號，姓名，性別。其中代號是唯一的。
- 住院病人有三個 attributes: 病人代號，姓名，緊急聯絡人，其中緊急聯絡人可以有多位，且必須記載緊急聯絡人之姓名與電話。其中病人代號是唯一的。
- 檢驗項目有兩個 attributes: 項目代號，名稱。其中項目代號是唯一的。
- 病房有兩個 attributes: 房號，等級。其中房號是唯一的。
- 一位醫師至多有一位指導醫師。
- 一位醫師診療一個病人時可能會要其做零或多項檢驗，一位醫師可以診療零或多位病人，而一位病人可被一或多位醫師診療，此外，診療日期必須記載。請用 ternary relationship 來表示此關係。
- 請依以上需求，畫出其 E-R diagram。

練習三：假設你要去 model 一個醫藥百貨的銷退貨系統，經過訪談，你得到以下的需求：

- 至少要有 4 個 entity types: 客戶，銷貨單，產品，退貨單。
- 客戶有三個 attributes: 代號，姓名，性別。其中代號是唯一的。
- 產品有三個 attributes: 產品編號，產品名稱，產品種類。其中產品編號是唯一的。
- 銷貨單有三個 attributes: 銷貨單編號，發票編號，日期。其中銷貨單編號和發票編號都是唯一的。
- 退貨單有二個 attributes: 退貨單編號，日期。其中退貨單編號是唯一的。
- 一張銷貨單記載一個客戶所購買的數種產品之產品編號，產品數量及單價。
- 一張退貨單記載一個客戶所退回的數種產品之產品編號，產品數量及原銷貨單編號。

請依以上需求，畫出其 ER diagram。

Ans：

銷貨單

銷貨單編號 發票編號 日期，客戶資料		
產品編號	數量	單價

退貨單

退貨單編號 日期 客戶資料		
產品編號	數量	原銷貨單編號

練習四：假設你設計了數個網路遊戲放在你的網站上給會員使用，現在你想收集一些使用的資料以便做統計。

- 每一會員有代號，姓名，住址，性別，和出生年月日。其中代號是唯一的。
- 每一遊戲有代號，名稱，遊戲難度。其中代號是唯一的。
- 會員進站時須輸入帳號和密碼。為方便統計，你希望記載進站的時間和玩遊戲時的得分。

請依以上的需求，畫出 ERD。

思考題

1 爲何需要 Relationship Type? 爲何不用 Attribute 來表示所有的 Relationship？

答：用關係型態來表示可將各 Entity 串聯起來，共用相同的 Attribute，有資料唯一、一致的功
能，並可節省記憶媒體空間。

2.爲何需要 Weak Entity Type? 爲何不能乾脆用 Composite multi-valued attribute 來取代一個 Weak Entity Type？

答：若兩 Entity 之間有一關係存在，則 Weak Entity Type 很難變成組合式多值屬性來取代。此外，Weak Entity Type 可以表示 partial key 屬性，這也是 Composite multi-valued attribute 所無法表示的。

3.一個 Entity 可不可以出現在兩個 Entity Types 裡？

答：可以，但盡可能不要，例如下面例子，研究生可能兼具兩個 Entity 身份。解決此問題要用物件導向（Object-Oriented）的觀念，將在 EER Model 裡敘述。

三、Relational Model（Chapter 7）

Background about Relational Model（關聯式模式）

- It is introduced by Codd in 1970.
- It has a solid theoretical foundation（有很穩固的理論基礎，有很好的結構定義）.
- It is based on a simple and uniform data structure（構築在簡單、統一的資料結構上）.
- It is also the most popular data model adopted by commercial DBMSs（目前使用最多者）.

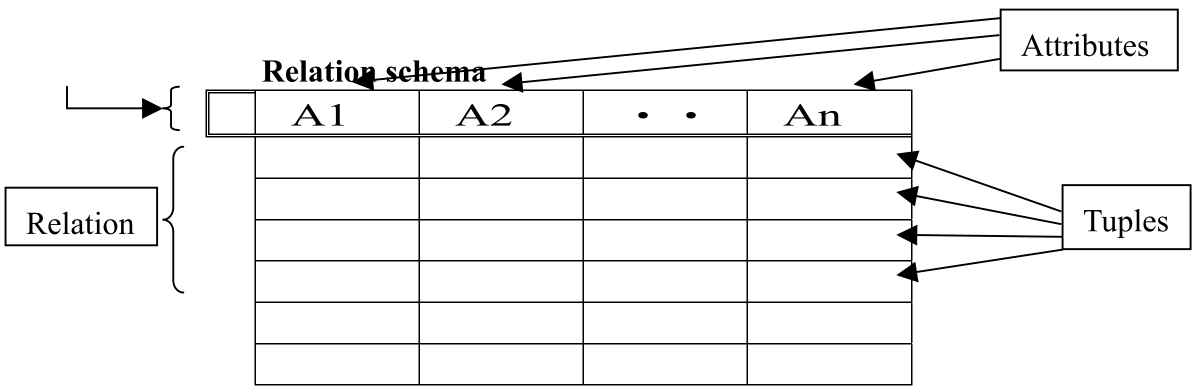
（一）Relational Model Concepts

Domains, Attributes, Tuples, and Relations

- A database is a collection of relations（一個資料庫是一堆的關聯）.
- Informally, **a relation resembles a table, each row represents a collection of related data values**. These values can be interpreted as facts describing a real-world entity or relationship.
- .In relational terminology, **a row is called a tuple, a column header is called an attribute, and the table is called a relation**.
  -

理論上稱		實務上稱
Relation	＝	Table
Tuple	＝	Row（Record）
Attribute	＝	Column（Field）

- A Relation schema R, denoted by R (A1,A2,...,An), is made up of a relation name R and a list of attributes A1,A2,…,An. The **degree** of a relation is the number of attributes of its relation schema.
  一個關聯是許多 tuples 的集合（A relation is defined as a set of tuples）
-
  -



- 身分證字號之 domain 為 A999999999，**Atomic value** 表示為一無法再分割的單一值。
- A relation (or relation state, or relation instance) **r** of the relation schema R(A1,A2,...,An),also denoted as r(R), is a set of n-tuples.
- A tuple is a set of (<attribute>,<value>) pairs.

- The meaning of attribute, extension, domain, null value in relational model is the same as that in ER model.
- Each attribute must be associated to a domain, which is a set of atomic values. (每一屬性必有一堆的單一值的值域)
- A relation (instance) at a given time reflects only the valid tuples that represent a particular state of the real world. See Figure 7.2 in page 199.
- 在 ER Model 只有 Entity 與 Relationship，在 Relational Model 也可表示此兩種事物，但並沒什麼區分，都視之為 Relations。
- A database schema ≜ A set of relation schema
- An example of a relation schema for a relation of degree 7 is the following：
  STUDENT (Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

Characteristics of Relations（關聯的特性）

- Tuples in a relation have no order.
- Ordering of values within a tuple（Ordering of attributes in a relation schema definition）is NOT important. In other words, conceptually values in a tuple have no order, but physically they do have an order.
- Composite and multi-valued attributes are not allowed in relational model.
- 某些 Relations 表示 Entities 的資料，而另某些 Relations 表示 Relationships 的資料，所以在關聯式模式裡，同樣都以 Relations 來表示 Entities 和 Relationships 的資料。

（二）Relational Model Constraints

- **Domain Constraints**：The value of each attribute A must be an atomic value from the domain dom(A) for that attribute.（每一屬性的值必定是其值域內的某單一值）
- Key Constraints：
  - By definition, no two tuples in a relation can have the same combination of values for all their
  - attributes.
  - A **superkey** is a set of attributes (subset of attributes of a relation schema) whose values are unique for each tuple.（組成唯一值的一串 attributes）
  - A **key** is a minimal superkey (i.e., no redundancy , still have the uniqueness constraint hold).
  - In general, a relation schema may have more than one key. In this case, one is chosen as the primary key,（在 Relation schema 中主要鍵之下畫一底線）while others are called candidate keys or secondary keys.
  - **Key constraint** is such that no two tuples have the same values on key attributes**.**
- Integrity Constraints（完整性限制）：
  A relational database schema is a set of relation schema and a set of integrity constraints. A relational database is a set of relations.（參考 Figure 7.5 and 7.6, 課本 204，205 頁）。
- **Entity Integrity constraint** states that no primary key value can be null.
- **Referential Integrity Constraint**：
- Informally, it says a tuple in one relation that refers to another relation must

refer to an existing tuple in that relation. (See Figure 7.7, 208 頁)

- A set of attributes of R1 is called a **foreign key** if
  - Its domain is the same as the primary key of another relation schema R2, and
  - Its value either occur as a value of the primary key of some tuple in R2 or is null.
  - A foreign key can refer to its own relation.（foreign key 也可參考到其本身） See Figure 7.7 for the graphical representation (SUPERSSN in EMPLOYEE references to SSN also in EMPLOYEE).
  - It should be enforced automatically by the DBMSs, e.g., the deletion of a department cannot occur before that of its employees.
- **Semantic** (語意的) **Integrity Constraints**：Any other semantic relationships between the values of attributes (in different tuples), e.g., salary of an employee must not exceed that of his (her) manager.

（三）Update Operations on Relations

- There are three types of update operations: insert, delete, and modify.
- Whenever update operations are applied, no integrity constraints specified should be violated.（不可違背完整性限制）
- Insert may violated all types of integrity constraints (可能違背全部的完整性限制;參考課本 209 頁 for examples). In case some integrity constraints is violated, two options are available：
  - Reject the insert.
  - Correct the reason for rejection.
- Delete may violate referential integrity constraints (可能違背參考的完整性限制;參考課本 210 頁 for examples). Three options are available for the violation of referential integrity constraints：
  - Reject the deletion.
  - Cascade the deletion.（相關的資料全部刪除）
  - Modify the referencing attribute values.（修改參考到此的屬性值）
- Modification may violate all types of integrity constraints. The issues are the same as those in insert and delete.（修改主鍵值相同於刪除一個 tuple，然後再插入一個 tuple）

（四）Basic Relational Algebra Operation

- It is a collection of operations that are used to manipulate entire relations. Each operation takes relations as input, and output is also a relation.
- Relational operations can be divided into two groups:
  - **Set operations**: UNION，INTERSECTION，DIFFERENCE，CARTESIAN（迪卡兒）PRODUCT.
  - **New operations**: SELECT，PROJECT，JOIN，aggregate functions（彙總功能）.

SELECT operation（σ ；sigma）

- Select a subset of tuples in a relation that satisfy a selection condition: E.g., $\sigma_{DNO=4}$(EMPLOYEE)，

σ **salary>25000+2000*year (EMPLOYEE)**.

- The general format is σ <selection condition>(<relation name>), where selection is a predicate（敘述句）on the attribute values in the associated relation.
- SELECT operator is unary（單運算元運算；如數學之 -A）, i.e., it is applied to single relation.
- It is applied to each tuple indivisablly.
- The fraction of tuples selected by a selection condition is called "selectivity"（選擇性）of the operation.
- SELECT operation is commutative（互換性）, i.e.
  σ <cond1>（σ <cond2>（R））= σ <cond2>（σ <cond1>（R））.

PROJECT operation（π ；pi）

- Select certain columns from the table and discard other columns, e.g.
  π **LNAME, FNAME, SALARY（EMPLOYEE）**
- The general format is π <attribute list>（<relation name>）.
- Project operator implicitly removes（自動移除）any duplicate tuples.
- The result of the PROJECT operation has only the attributes specified in <attribute list> and in the same order as they appear in the list.

Applying several relational operations to get desired results

- Several relational operations can be applied one after another to get desired results, This expression is called **relational algebra expression**, e.g., π FNAME, LNAME, SALARY（σ DNO=5（EMPLOYEE））.
- Or you can name the intermediate results：
  DEPT5_EMPS ← σ DNO=5（EMPLOYEE）
  RESULT ← π FNAME, LNAME, SALARY（DEPT5_EMPS）.

RENAME operation（ρ ；rho）

- To rename the attributes in a relation, we simply list the new attribute names in parentheses：
  TEMP ← σ DNO=5（EMPLOYEE）
  R (FIRSTNAME, LASTNAME, SALARY) ← π **FNAME, LNAME, SALARY（DEPT5_EMPS）**.
- General RENAME operation is denoted by：
  ρ **S (B1, B2,..., Bn)（R）**：renames both the relation and its attributes.
  ρ S（R）：renames the relation only.
  ρ (B1, B2,..., Bn)（R）：renames the attributes only.

Set Operations

- Set theoretic operations are binary operations.【雙運算元運算】
  - Union（∪；聯集；R∪S；包括屬於左關聯及屬於右關聯及同屬於兩關聯的全部 tuples），
  - Intersection（∩；交集；R∩S；包括同屬於兩關聯的 tuples），Difference（—；差集；R—S；包括屬於左關聯而不屬於右關聯的 tuples），
  - Cartesian Product（×；乘集；R × S；包括兩關聯的 tuples 兩兩配對後產生之所有 tuples）.
- Note that two relations that ∪，∩ or — are applied to must have the same

type of tuples, which is called union compatibility（適合聯集性；兩關聯有同數目的屬性且每一相配對的屬性有相同的值域）. See Figure 7.11 in page 218 for an example.

- Both ∪ and ∩ satisfy **commutativity**（互換性；R∪S=S∪R and R∩S=S∩R）and **associativity**（聯合性；（R∪S）∪T= R∪（S∪T）, and R∩（S∩T）=（R∩S）∩T）, Difference（—）is not commutative.
- The two relations that are applied to Cartesian Product（╳）need not to be union compatible.

k Attributes		*l* Attributes		*k+l* Attributes
R	╳	**S**	＝	**Q**
n tuples		*m* tuples		*mn* tyuples

- Suppose R and S have *n* tuples and *m* tuples respectively, R ╳ S will have *n*m* tuples, See Figure 7.12.
- Cartesian Product followed by SELECT operator can be combined by a special operator called JOIN.

JOIN operation（⋈）

- It is used to combine related tuples from two relations into single tuples. It allows us to process relationships among relations.
- The general format is **R ⋈ <join condition> S**, <join condition>(配合條件) 的通式：<條件>AND<條件>AND…AND<條件>）
- Join condition is a conjunction of Ai Θ Bj, where Ai is an attribute of R, Bj is an attribute of S, and Θ（theta）is one of the comparison operators {=,<,≦,>,≧,≠}. A join operation with such a general join condition is called a THETA join.
- The most common JOIN is that with only equality comparison operator（=）. This is called EQUIJOIN.
- The result of an EQUIJOIN operation will contain two columns of the same value.
- Suppose R and S have *n* attributes and *m* attributes respectively, R ⋈ S will have *n*+*m* attributes.
- Another join operation is called **NATURAL JOIN**—denoted by ＊—is an equijoin followed by the removal of redundant attribute. It needs the two join attributes to have the same name（此屬性稱 join attribute）.
- The result of a JOIN will have between 0 and *n*m* tuples.
- Join selectivity = (number of tuples in a join result) ÷(*n*m*)
- 沒有指定配合條件的配合就和 CARTATION PRODUCT 一樣，又稱 CROSS JOIN 或 CROSS PRODUCT，例如：
  EMPLOYEE ⋈ $_{\text{TRUE}}$ DEPARTMENT ≡ EMPLOYEE ╳ DEPARTMENT
- Three-way join：
  （（PROJECT ⋈ $_{\text{DNUM=DNUMBER}}$ DEPARTMENT）⋈ $_{\text{MGRSSN=SSN}}$ EMPLOYEE）。

Complete Set of Relational Algebra Operation

- Define any relational algebra operation can be expressed as a sequence of operations from this set.

- For example, {σ ，π ，∪，—，×} is a complete relational operation set.【其他關聯代數運算可用此集合內任一或多個運算元用一順序運算表示，例如：(R∩S) ≡ (R∪S) — ((R—S) ∪ (S—R))】。
- 例如
  A={x,y}
  B={y, z}
  A∪B={x,y,z}
  A∩B={y}
  A—B={x}
  B—A={z}
  (A∪B)—(A—B)—(B—A) = (A∩B)
- A JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation.
  R ⋈ <condition> S ≡ σ <condition>（R × S）
- A NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations.

Examples

QUERY1：參考課本 230 頁及 208 頁 Figure.7.7（找出在 'Research' 部門工作的所有員工之姓名及住址）

EMPLOYEE (FNAME, MINIT, LNAME,…, ADDRESS,…, DNO)

DEPARTMENT (DNAME, <u>DNUMBER</u>,…)

||

“Research”

<u>Ans.</u>：RESEARCH_DEPT← σ DNAME = "Research"（DEPARTMENT）

RESEARCH_EMPS←（RESEARCH_DEPT ⋈ DNUMBER=DNO EMPLOYEE）

RESULT← π FNAME,LNAME,ADDRESS（RESEARCH_EMPS）

QUERY2：參考課本 231 頁及 208 頁 Figure7.7（找出位於 'Stafford' 之每一專案並列出專案號、控管部門號碼及部門經理姓氏、住址、生日）

EMPLOYEE(…,LNAME,<u>SSN</u>,BDATE,ADDRESS,…,DNO)

DEPARTMENT(…,<u>DNUMBER</u>,MGRSSN,…)

PROJECT(…,<u>PNUMBER</u>,PLOCATION,DNUM)

||

“Stafford”

<u>Ans.</u>： STAFORD_PROJS← σ PLOCATION = "Stafford"（PROJECT）

CONTR_DEPT←（STAFORD_PROJS ⋈ DNUM=DNUMBER DEPARTMENT）

PROJ_DEPT_MGR←(CONTR_DEPT ⋈ MGRSSN=SSN EMPLOYEE）

RESULT← π PNUMBER, DNUM,LNAME, ADDRESS, BDATE（PROJ_DEPT_MGR）

QUERY4：參考課本 231 頁及 208 頁 Figure7.7（列出'Smith'有參與的專案號碼，直接或間接控管的）

EMPLOYEE(…,LNAME,SSN,…)
‖
“Smith”

WORKS_ON(ESSN,PNO,HOURS)

DEPARTMENT(…,DNUMBER,MGRSSN,…)

PROJECT(…,PNUMBER,PLOCATION,DNUM)

提示：先做 smith directly works 然後再做 smith indirectly works（manages）

Ans.： SMITH（ESSN）← π_{SSN}（$\sigma_{\text{LNAME = "Smith"}}$（EMPLOYEE））
SMITH_WORKER_PROJ← π_{PNO}（WORKS_ON ＊ SMITH）
MGRS← $\pi_{\text{LNAME, DNUMBER}}$（EMPLOYEE $\bowtie_{\text{SSN = MGRSSN}}$ DEPARTMENT）
SMITH_MANAGED_DEPTS（DNUM）← π_{DNUMBER}（$\sigma_{\text{LNAME = "Smith"}}$（MGRS））
SMITH_MGR_PROJS（PNO）← π_{PNUMBER}（SMITH_MANAGED_DEPTS＊PROJECT）
RESULT←（SMITH_WORKER_PROJ ∪ SMITH_MGR_PROJS）

The Division Operation

- .Example: Retrieve the names of employees who work on all the projects that 'John smith' works on.
  （NAMEID÷PID）.（參考課本 225 頁 Figure. 7.15）

Ans.： SMITH← $\sigma_{\text{FNAME = 'John' AND LNAME = "Smith"}}$（EMPLOYEE）
SMITH_PNOS← π_{PNO}（WORKS_ON $\bowtie_{\text{ESSN = SSN}}$ SMITH）
SSN_PNOS← $\pi_{\text{ESSN, PNO}}$（WORKS_ON）
SSNS (SSN) ← SSN_PNOS ÷ SNITH_PNOS
RESULT← $\pi_{\text{FNAME, LNAME}}$（SSNS ＊ EMPLOYEE）

- T = R（Z）÷S（X）, where Y=Z—X，X⊆Z.（此除法可以下列連續運算表示）
  T1← π_{Y}（R）
  T2← π_{Y}（(（S $\times$ T1）—R）
  T ← T1—T2

  例：**R(Y, X)** **S(X)** **T(Y)** （用 PROJECT 將 R, S 之屬性變成如下以利作除運算）

R(Y, X)		S(X)		T(Y)
y1, x1		x1		y1
y1, x2	÷	x2	＝	y2
y1, x3				
y2, x1				
y2, x2				
y3, x1				

試以上述 T ← T1—T2 之算法演算上例，R, S 如下：

T1	**T1⨯S**	**T1⨯S—R**	**T2**	**T**
y1	y1, x1	y3 x2	y3	y1
y2	y1, x2			y2
y3	y2, x1			
	y2, x2			
	y3, x1			
	y3, x2			

（五）Additional Relational Operations

Aggregate Functions and Grouping

- So far the mathematical aggregate functions（彙總功能）on collections of values from the database can not be specified by relational operations. The aggregate functions include：
  - ■ SUM，AVERAGE，MAXIMUM，MINIMUM（is used for collection of numeric values.
  - ■ COUNT（is used for counting tuples or values.
- Sometimes you need the aggregate functions to be applied in the group basis, e.g., find the average of each department.（分組再彙總功能）
- **General form：<grouping attributes>$\mathcal{F}$<function list>(<relation name>)**（$\mathcal{F}$ is pronounced "script F"）

  <grouping attributes>：a list of attributes of the <relation name>

  <function list>：a list of（<function><attribute>）pairs
- 例：R（DNO, NUMBER_OF_EMPLOYEE, AVERAGE_SAL）← $_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE SALARY}}$（EMPLOYEE）同：$\rho_{\text{R（DNO, NUMBER\_OF\_EMPLOYEE, AVERAGE\_SAL）}}$($_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE SALARY}}$（EMPLOYEE）)

R	DNO	NUMBER_OF_EMPLOYEE	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

- $\mathcal{F}_{\text{COUNT SSN, AVERAGE SALARY}}$（EMPLOYEE）產生的 table 是 R(COUNT_SSN,AVERAGE_SALARY)

COUNT_SSN	AVERAGE_SALARY
8	35125

沒有 grouping attributes，故全部統計爲一筆。

- $_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE SALARY}}$（EMPLOYEE）則為 R（DNO,COUNT_SSN,AVERAGE_SALARY）

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	32500
4	3	31000
1	1	55000

Recursive Closure Operations

- Relational algebra operations generally cannot specify recursive closure.（遞迴封閉）
- Example: Get the names of the employees who are directly or indirectly supervised by James Borg.
- Relational operations cannot specify recursive queries because they lack of looping mechanisms.
- See Figure 7.17 A two-level recursive query.
- 先找出 Borg 直屬的下屬 RESULT1，再由此與原 EMPLOYEE 配合找出其下屬的下屬 RESULT2.

$\text{BORG\_SSN} \leftarrow \pi_{\text{SSN}}(\sigma_{\text{FNAME = 'James' AND LNAME = "Borg"}}(\text{EMPLOYEE}))$
$\text{SUPERVISION（SSN1, SSN2）} \leftarrow \pi_{\text{SSN, SUPERSSN}}（\text{EMPLOYEE}）$
$\text{RESULT1} \leftarrow \pi_{\text{SSN1}}（\text{SUPERVISON} \bowtie_{\text{SSN2 = SSN}} \text{BORG\_SSN}）$
$\text{RESULT2（SSN）} \leftarrow \pi_{\text{SSN1}}（\text{SUPERVISON} \bowtie_{\text{SSN2 = SSN}} \text{RESULT1}）$
$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$

OUTER JOIN Operations

- 相對的⋈則為 inner join，inner join 只配合可匹配的部分。
- R 和 S 的 JOIN 可能會排除掉 R 或 S 中的某些 tuples，使用 OUTER JOIN 則可保留想要的 tuples.
- Example: Find a list of all employee names and also the name of the departments they manage if they happen to manage a department. 使用下列 LEFT OUTER JOIN 即可，

$\pi_{\text{FNAME, LNAME, DNAME}}（\text{EMPLOYEE} \bowtie_{\text{SSN = MGRSSN}} \text{DEPARTMENT}）$

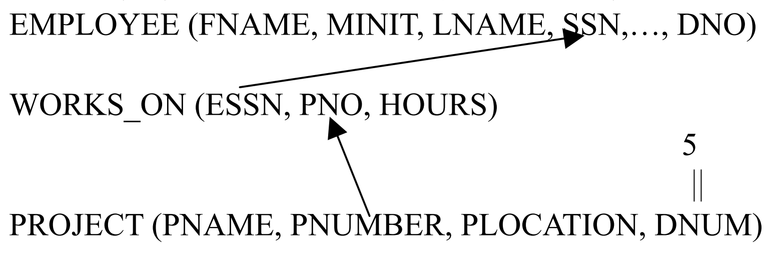
- LEFT OUTER JOIN 保留左邊關聯的所有 tuples，RIGHT OUTER JOIN（⋈）保留右邊關聯的所有 tuples，FULL OUTER JOIN（⋈）保留兩邊關聯的所有 tuples。

OUTER UNION Operations

例:兩關聯的綱目為 STUDENT（Name, SSN, Department, Advisor）及
FACULTY（Name, SSN, Department, Rank）.
則聯集後的綱目為
R（Name, SSN, Department, Advisor, Rank）
R 會包含兩關聯的所有 tuples，STUDENT 的 tuples 的 Rank 屬性為 null，FACULT 的 tuples 的 Advisor 屬性為 null，若某一 tuples 屬於此兩關聯，則 Rank 屬性及 Advisor 屬性會各有其原值。

（六）Examples

1.參考課本 231 頁 QUERY3 及 208 頁 Figure7.7，205 頁資料:
找出爲所有被第五部門控管方案工作的人員名單


EMPLOYEE (FNAME, MINIT, LNAME, SSN,…, DNO)
WORKS_ON (ESSN, PNO, HOURS)
5
PROJECT (PNAME, PNUMBER, PLOCATION, DNUM)

Ans.:

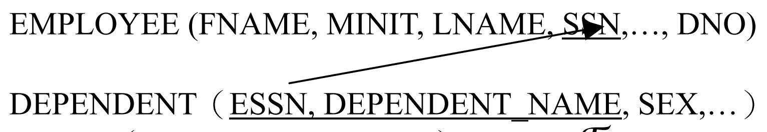
DEPT5_PROJS (PNO) $\leftarrow \pi_{\text{PNUMBER}}(\sigma_{\text{DNUM}=5}(\text{PROJECT}))$

EMP_PROJ (SSN, PNO) $\leftarrow \pi_{\text{ESSN, PNO}}(\text{WORKS\_ON})$

RESULT_EMP_SSNS $\leftarrow$ EMP_PROJ $\div$ DEPT5_PROJS （此行演算之結果爲 NULL）

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$

2.參考課本 231 頁 QUERY5 及 208 頁 Figure7.7，205 頁資料:（列出家屬有二或二個以上的員工姓名）

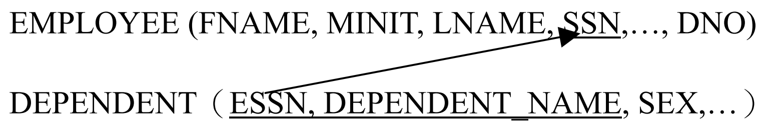

EMPLOYEE (FNAME, MINIT, LNAME, SSN,…, DNO)
DEPENDENT（ESSN, DEPENDENT_NAME, SEX,…）

Ans.: T1（SSN, NO_OF_DEPTS）$\leftarrow {}_{\text{ESSN}}\mathcal{F}_{\text{COUNT DEPENDENT\_NAME}}(\text{DEPENDENT})$

T2 $\leftarrow \sigma_{\text{NO\_OF\_DEPTS} \geqq 2}(\text{T1})$

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}(\text{T2} * \text{EMPLOYEE})$

3.參考課本 232 頁 QUERY6 及 208 頁 Figure7.7，205 頁資料: （列出沒有家屬的員工姓名）


EMPLOYEE (FNAME, MINIT, LNAME, SSN,…, DNO)
DEPENDENT（ESSN, DEPENDENT_NAME, SEX,…）

Ans.: ALL_EMPS $\leftarrow \pi_{\text{SSN}}(\text{EMPLOYEE})$

EMPS_WITH_DEPS（SSN）$\leftarrow \pi_{\text{ESSN}}(\text{DEPENDENT})$

EMPS_WITHOUT_DEPS $\leftarrow$ (ALL_EMPS — EMPS_WITH_DEPS)

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$

（也可用 Outerjoin 方式做）

4.參考課本 232 頁 QUERY7 及 208 頁 Figure7.7，205 頁資料: （列出至少有一家屬的經理姓名）

EMPLOYEE (FNAME, MINIT, LNAME, <u>SSN</u>,…, DNO)

DEPENDENT（<u>ESSN, DEPENDENT_NAME</u>, SEX,…）

DEPARTMENT（DNAME, <u>DNUMBER</u>, MGRSSN,…）

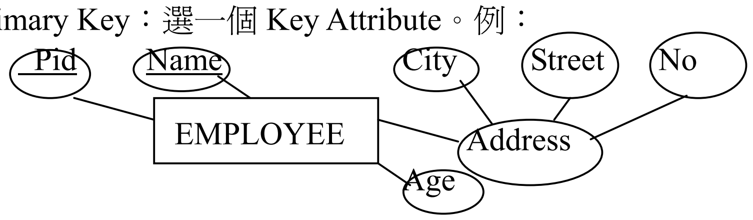
<u>Ans.</u>: MGRS（SSN） $\leftarrow \pi_{\text{MGRSSN}}$（DEPARTMENT）
EMPS_WITH_DEPS（SSN） $\leftarrow \pi_{\text{ESSN}}$（DEPENDENT）
MGRS_WITH_DEPS $\leftarrow$ （MGRS ∩ EMPS_WITH_DEPS）
RESULT $\leftarrow \pi_{\text{LNAME,FNAME}}$（MGRS_WITH_DEPS ＊ EMPLOYEE）

5.課本 239 頁 7.25 及 7.26 習題，回家作，可在網路 BBS 上討論。

（七）ERD 轉成關聯式綱目：

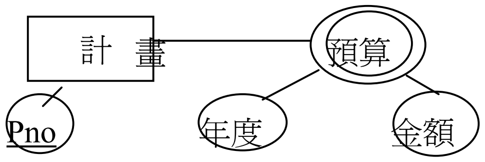
（參考課本第 290 頁 9.1.1）

1. 對每一個 Entity Type 產生一個 Table（Relational Schema），此 Table 的屬性及主鍵如下：
Attribute：所有簡單屬性（Simple Attribute）＋所有複合屬性（Composite Attribute）的展開成簡單屬性。
Primary Key：選一個 Key Attribute。例：



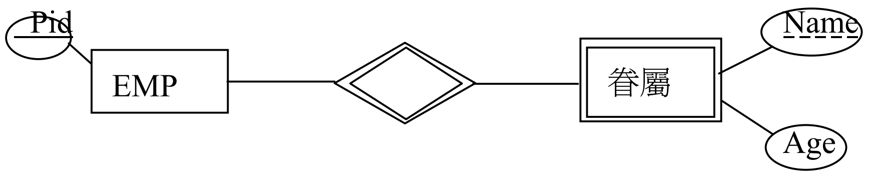
EMPLOYEE(<u>Pid</u>, <u>Name</u>, City, Street, No, Age)，另參考課本 Figure 3.2 及 7.7 中之 EMPLOYEE。

2. 對每個多值（Multi-valued）屬性，產生一個 Table（Relational Schema），此 Table 的屬性及主鍵如下：
Attribute：所有組成屬性 ＋ 原屬 Entity Type 之 Primary Key。
Primary Key：所有以上 Attributes。例：



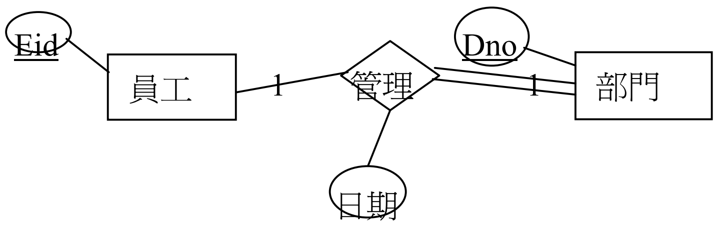
計畫本身須產生一個 Table，另預算也產生→ 預算（<u>Pno</u>, <u>年度</u>, <u>金額</u>）⇐其實金額可不為 Key.
另參考課本 Figure 3.2 中 DEPARTMENT 中之 Locations。

3.對每一個 Weak Entity Type 產生一個 Table（Relational Schema），此 Table 的屬性及主鍵如下：
Attribute：所有組成屬性（Attributes）＋ 原屬 Entity Type 之 Primary Key。
Primary Key：部分鍵屬性（Partial Key Attribute）＋ 原屬 Entity Type 之 Primary Key。例：

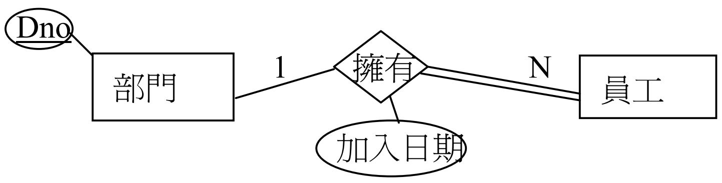


產生→眷屬（<u>DPid</u>, <u>Name</u>, Age），Dpid 與 Pid 相同，另外 EMP 本身須產生一個 Table。
另參考課本 Figure 3.2 及 7.7 中 DEPENDENT。

4.對每一個 1:1 之 Relationship Types（R 與 S）選擇一 Table（Relational Schema），假定選 R，則在 R 中加上一個外鍵（Foreign Key；參考到 S 之 Key），若此關係型態有屬性，則將這些屬性加到 R 中。在決定該那一個 Table 要加入外鍵時，以 Total Participation（必然參與）此關係型態之 Entity Type 爲優先：

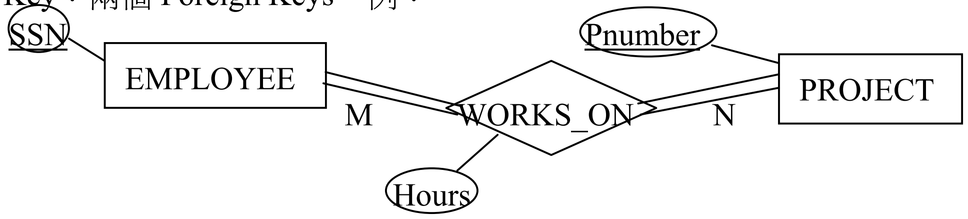


在 "部門" 產生之 Table 中加一外鍵（例如：DEid）參考到員工的 Eid，日期也加到 "部門" 中。
另參考課本 Figure 3.2 及 7.7 中 DEPARTMENT 與 EMPLOYEE 之關係 MANAGES。
5.對每一個 1:N 之 Relationship Type（R 與 S）選擇一 Table（Relational Schema），假定選 S（有 N 的關係者），則在 S 中加上一個外鍵（Foreign Key；參考到 R 之 Key），若此關係型態有屬性，則將這些屬性加到 S 中。：



在 "員工" 產生之 Table 中加一外鍵（例如：EDno）參考到部門的 Dno，加入日期也加到 "部門"中。另參考課本 Figure 3.2 及 7.7 中 DEPARTMENT 與 EMPLOYEE 之關係 WORKS_FOR。

6.對每一個 M:N 之 Relationship Type（R 與 S），產生一個 Table（Relational Schema），此 Table 的屬性及主鍵如下：
Attribute：所有 Relationship Type 的屬性 ＋ 兩個 Foreign Keys（R 與 S 兩個 Entity Type 之 Key）。
Primary Key：兩個 Foreign Keys。例：



產生一 Table → WORKS_ON（ESSN, Pno, Hours），ESSN 參考到 SSN，Pno 參考到 Pnumber。
另參考課本 Figure 3.2 及 7.7 中 DEPARTMENT 與 EMPLOYEE 之關係 WORKS_ON。

7.對每一個 n 元的 Relationship Type（n＞2），產生一個 Table（Relational Schema），此 Table 的屬性及主鍵如下：
Attribute：所有 Relationship Type 的屬性 ＋ 所有參與的 Entity Type 之 Foreign Key。
Primary Key：所有的 Foreign Keys。

8.Correspondence between ER and Relational Models：

ER Model	**Relational Model**
Entity type	"Entity" relation
1:1 or 1:N relationship type	Foreign key (or "relationship" relation)
M:N relationship type	"Relationship" relation and two foreign keys
n-ary relationship type	"Relationship" relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

四、SQL：（**Chapter 8**）

A few facts about SQL

1. 很少的 DBMS 使用關聯代數當作其查詢語言，大部分的 DBMS 會提供一些像 SQL（Structured Query Language）這種宣告樣式的語言。目前 SQL 是商用關聯式 DBMS 的標準語言，具有廣泛功能，它有資料定義、查詢、及修正等敘述。
2. Originally, SQL was called “SEQUEL”（for Structured English QUEry Language），最初是由 IBM 公司研究部門所設計的 SYSTEM R，後來由 ANSI(American National Standards Institute)及 ISO(International Standards Organization）合力訂定標準如下：
    - SQL1：SQL-86（ANSI 1986）
    - SQL2：SQL-92
    - SQL3：Still on the way，combine relational and object models.
3. 今天大部分商用關聯式 DBMS 是支援 SQL2。

A few differences between SQL relational model and theoretical relational model.

1. SQL uses the terms **table**, **row**, and **column** to indicate relation, tuple, and attribute respectively that defined in the relational data model.
2. However, a table in SQL may contain duplicated rows, while a relation does not contain duplicated tuples.
3. A table in SQL does not have to specify its primary key.

SQL consists of three types of languages

1. Data Definition Language（DDL；資料定義語言）: language syntax to add, drop, modify a table definition.
2. Data Manipulation Language（DML；資料處理使用語言）: language syntax to retrieve, insert, delete, modify rows of tables.
3. Data Control Language：（DCL；資料控制語言）: language syntax to grant/revoke the privileges(使用權限） of access on the tables, and to control the execution of transactions.

（一）Data Definition, Constraints, and Schema Changes in SQL2

Schema Concepts in SQL2

1.An **SQL schema** is identified by a schema name, and an authorization identifier to indicate the user or account who owns the schema.

2.The following statement creates a schema called COMPANY, owned by the user with authorization identifier JSMITH.
CREATE SCHEMA COMPANY **AUTHORIZATION** JSMITH;

DDL in SQL

1. **CREATE TABLE** 可定義下列事項：
   - Table name.
   - Attribute name.
   - Attribute type：
   - Numeric: INT or INTEGER, SMALLINT（整數），FLOAT, REAL, DOUBLE PRECISION（實數），DECIMAL(x, y).（含小數表示法）。
     i. Character-string: CHAR(n), VARCHAR(n), BLOB（註：BLOB →Binary Large Object，用於表示較大量資料時使用）。
     ii. Bit-string,: BIT(n), BIT VARYING(n)。
     iii. date and time: TIME (HH:MM:SS), DATE (YYYY-MM-DD)，TIMESTAMP=DATE+TIME，TIME(2)→18:50:43:20（顯示百分秒）。
   - Attribute constraints（屬性限制）：
     i. **PRIMARY KEY**（attribute list）：例:**PRIMARY KEY**（DNUMBER, DLOCATION）,
     ii. **FOREIGN KEY**（attribute list）**REFERENCES** table name（attribute list）**ON DELETE…ON UPDATE…**：
     例：**FOREIGN KEY**（SUPERSSN）**REFERENCES** EMPLOYEE（SSN）**ON DELETE** SET NULL **ON UPDATE** CASCADE;
     iii. **UNIQUE**（attribute list）：定義 alternate or secondary key。
     iv. <attribute name> <attribute type> **NOT NULL**：限制 attribute 不可為空值。
     v. <attribute name> <attribute type> **DEFAULT** <value>：限制 attribute 如沒輸入值就給予初值。
     vi. **CONSTRAINT** <constraint name>：定義要限制的名稱（在同一 schema 內須唯一），方便以後刪除或更改時可直接呼叫其名稱出來執行。

   參考課本第 247 頁 Figure 8.1(a)及 249 頁 Figure 8.1(b)。

2. **DROP SCHEMA** ：移除一個綱目
   - **DROP SCHEMA** COMPANY **CASCADE**（連鎖）：移除 COMPANY 資料庫綱目及其所有 tables, domains, and other elements.
   - **DROP SCHEMA** COMPANY **RESTRICT**（有限制）：要移除 COMPANY 資料庫綱目必須該綱目是空的，否則將不會執行。
3. **DROP TABLE**：移除一個 table
   - **DROP TABLE** DEPENDENT RESTRICT：只有當 DEPENDENT 這個 table 沒被其他 table 所 reference 到時才能刪除〔比如 referential integrity constraints or view definition〕。
   - **DROP TABLE** DEPENDENT **CASCADE**：所有 reference DEPENDENT 的 constraints 也會被連帶刪除。請參考 Figure 8.1(b)
4. **ALTER TABLE**： also called schema evolution（綱目變化），修改 table 的 SCHEMA 及限制。
   - **ALTER TABLE** COMPANY.EMPLOYEE **ADD** JOB VARCHAR(12);：增加一個屬性。

- **ALTER TABLE** COMPANY.EMPLOYEE **DROP** ADDRESS **CASCADE**;：連鎖式刪除一個屬性。All constraints and views that reference the column ADDRESS are dropped cascadingly.
- **ALTER TABLE** COMPANY.EMPLOYEE **DROP** ADDRESS **RESTRICT**;：限制式刪除一個屬性。 ADDRESS is dropped only when no constraints or views reference it.
- **ALTER TABLE** COMPANY.DEPARTMENT **ALTER** MGRSSN **DROP DEFAULT**;：刪除 DEFAULT 子句。
- **ALTER TABLE** COMPANY.DEPARTMENT **ALTER** MGRSSN **SET DEFAULT** '33344555';：重新設定 DEFAULT 子句。(將 DEFAULT 值重新設定爲'33344555')
- **ALTER TABLE** COMPANY.EMPLOYEE **DROP CONSTRAINT** EMPSUPERFK **CASCADE**;：刪除一個 CONSTRAINT。
- **ALTER TABLE** COMPANY.EMPLOYEE **ADD CONSTRAINT** EMPSUPERFK **CASCADE**;：增加一個 CONSTRAINT。

(二) Basic Queries in SQL (part of SQL DML)

SELECT-FROM-WHERE Structure of SQL Queries

1. SELECT statement, sometimes called a **mapping** or **select-from-where block**.
   Basic form：**SELECT**<attribute list>
   FROM <table list>
   WHERE <condition>
   參考課本 Figure 7.7。
2. **Q0**：Retrieve the birthdate and address of employee(s) whose name is "John B. Smith"
   Ans.：
   SELECT BDATE, ADDRESS
   FROM EMPLOYEE
   WHERE FNAME = "John" **AND** MINIT = "B"
   AND LNAME ="Smith"；
3. **Q1**：Retrieve the name and address of employees who work for the "Research" department.
   Ans.：
   SELECT FNAME, LNAME, ADDRESS
   FROM EMPLOYEE, DEPARTMENT
   WHERE DNAME = 'Research' **AND** DNUMBER = DNO；
4. **Q2**：For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.
   Ans.：
   SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
   FROM PROJECT, DEPARTMENT, EMPLOYEE
   WHERE DNUM = DNUMBER **AND** MGRSSN = SSN
   AND PLOCATION = 'Stafford'；

Dealing with Ambiguous（不確定的）Attribute Names

1. Same name in different relation, e.g., Suppose EMPLOYEE has attributes DNUMBER and NAME（rather than DNO and LNAME）, and DEPARTMENT has NAME（rather than DNAME）.
   Q1 can be revised to the following（255 頁 **Q1A**）:
   ```
   SELECT    FNAME, EMPLOYEE. NAME, ADDRESS
   FROM      EMPLOYEE, DEPARTMENT
   WHERE     DEPARTMENT. NAME = 'Research' AND
                  DEPARTMENT.DNUMBER = EMPLOYEE.DNUMBER
   ```
2. **2.** A query involves the same relation twice—use aliases,考慮以下的 query：
   - **Q8**：For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.
     Ans.：
     ```
     SELECT    E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM      EMPLOYEE AS E, EMPLOYEE AS S
     WHERE     E.SUPERSSN=S.SSN；
     ```
   - E and S are called aliases to EMPLOYEE. Alias can also be used just to shorten the relation names that prefix the attributes. For example, Q1 can be written as follows：
     Q1B：
     ```
     SELECT    E.FNAME, E.LNAME, E.ADDRESS
     FROM      EMPLOYEE AS E, DEPARTMENT AS D
     WHERE     D.DNAME = 'Research'
               AND D.DNUMBER = E.DNUMBER；
     ```

Unspecified WHERE-Clause and Use of Asterisk（＊）

1. Missing **WHERE** clause indicates no condition on row selection.
   Q9：Select the SSNs of all employee.
   Ans.：
   ```
   SELECT    SSN
   FROM      EMPLOYEE;
   ```
   Q10：Select all combinations of EMPLOYEE SSN and DEPARTMENT DNAME of.
   Ans.：
   ```
   SELECT    SSN, DNAME
   FROM      EMPLOYEE, DEPARTMENT;
   ```
   （相當於 cross Product）
2. 使用＊(Asterisk)表示要 Retrieve 全部屬性，例如：
   ```
   SELECT    ＊
   FROM      EMPLOYEE
   WHERE     DNO=5;
   ```
3. **3. Cartesian Product** can be specified by missing **WHERE** clause，例如：
   ```
   SELECT    ＊
   FROM      EMPLOYEE,DEPARTMENT;
   ```

Set Operation（集合運算） in SQL

1. **1.** SQL does not automatically eliminate duplicate rows in the results of queries. 基於下列三個原因：
   i. 要刪除重複 rows 須先排序然後才能刪除此重複，這是一種很耗費的運算。

ii. 有時候使用者可能要看這些重複資料。

iii. 如果是要作彙總運算時，刪除重複資料反而會出錯。

2. Elimination of duplicated rows is done explicitly, e.g.

SELECT DISTINCT SALARY
FROM EMPLOYEE;
（同值的 row 去掉剩一個，如此做很耗時，非必要不加 DISTINCT）

Q11：Retrieve the salary of every employee (A) and all distinct salary values (B).

Ans.：

```
(A) SELECT  ALL          SALARY
    FROM                 EMPLOYEE;
(B) SELECT  DISTINCT     SALARY
    FROM                 EMPLOYEE;
```

3. SET operations（e.g., UNION, MINUS, INTERSECT）, can be applied on "**union compatible**" relations.

假如兩關聯具有①相同數目的屬性②屬性的順序相同且同值域，則此兩關聯爲"**union compatible**"。

- **Q4**：Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
(SELECT DISTINCT  PNUMBER    [Smith 所管轄的 Project #】
 FROM       PROJECT, DEPARTMENT, EMPLOYEE
 WHERE          DNUM = DNUMBER AND MGRSSN = SSN
                    AND LNAME = 'Smith')
UNION
(SELECT DISTINCT  PNUMBER    【Smith 所做的 Project #】
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE       PNUMBER = PNO AND ESSN = SSN
            AND LNAME = 'Smith');
```

（此例之 DISTINCT 是否可以不要？）

Smith 所做的 Project #可改爲如下：

```
(SELECT     PNO AS PNUMBER     （讓此屬性名稱同前半段）
FROM        WORKS_ON, EMPLOYEE
WHERE       ESSN = SSN AND LNAME = 'Smith');
```

Substring Comparisons, Arithmetic Operators, and Ordering

1. **LIKE** qualifier can be used in clause to do substring comparison.

- **Q12**：Retrieve all employees whose address is in Houston, Texas.

Ans.：

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       ADDRESS LIKE '% Houston, TX %';
```

【**LIKE** 爲 "包含…"，"長得像…"的意思】

【"%" 表示可爲任意多個字元】

Q12A：Find all employees who were born during the 1950s.

Ans.：

```
SELECT      FNAME, LNAME
    FROM        EMPLOYEE
    WHERE       BDATE LIKE '__5_______';
```

【"_" 表示可爲任意一個字元】

2. **Simple arithmetic**（算數）can be applied on the returned values.
   - **Q13**：Show the resulting salaries if every employee working on the "ProductX" project is given a 10 percent raise.
     SELECT FNAME, LNAME, **1.1**∗SALARY
     FROM EMPLOYEE, WORKS_ON, PROJECT
     WHERE SSN = ESSN **AND** PNO = PNUMBER **AND** PNAME = 'ProductX';

3. An **interval value** can be specified as the difference between two values.
   - **Q14**：Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.
     Ans. **SELECT** *
     FROM EMPLOYEE
     WHERE (SALARY **BETWEEN** 30000 **AND** 40000) **AND** DNO = 5;

4. **ORDER BY** clause is used to specify the listing order of the query result.
   - **Q15**：Retrieve a list of employees and the projects they are working on, order by department and, within each department, ordered alphabetically by last name, first name.
     SELECT DNAME, LNAME, FNAME, PNAME
     FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
     WHERE DNUMBER = DNO **AND** SSN = ESSN **AND** PNO = PNUMBER
     ORDER BY DNAME, LNAME, FNAME;
   - ORDER BY 子句中可用 **DESC** 表示遞減排序，**ASC** 表示遞增排序，例：
     ORDER BY DNAME **DESC**, LNAME **ASC**, FNAME **ASC**;

（三）More Complex SQL Query

Nested Queries and Set Comparisons

1. The queries that have a complete SELECT-clause appears within the WHERE-clause of another query (outer query).
   - **Q4A**：Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

Ans.：

```
(SELECT    DISTINCT    PNUMBER
 FROM   PROJECT
 WHERE PNUMBER IN  【Smith 所管轄的 Project #】
     (SELECT     PNUMBER
      FROM      PROJECT, DEPARTMENT, EMPLOYEE
      WHERE     DNUM = DNUMBER
                AND MGRSSN = SSN
                AND LNAME      = 'Smith')
OR
        PNUMBER IN 【Smith 直接做的 Project #】
```

```
(SELECT PNO
  FROM    WORKS_ON, EMPLOYE
WHERE ESSN = SSN AND LNAME =
'Smith');
```

- **Comparison operator IN** 後的內部查詢值被外部查詢拿來做爲查詢條件。

2. **IN** 運算亦可作兩個屬性值的內部查詢，

- 例如：找出所有和 Smith 有相同(PNO, HOURS)的員工的 SSN

```
SELECT DISTINCT ESSN
FROM     WORKS_ON
WHERE   (PNO, HOURS) IN（SELECT PNO, HOURS FROM
WORKS_ON WHERE ESSN = '123456789'）;
```

3. 另外有 SOME 及 ANY 可配合 >, >=, <, <=, <>, = 使用。例: = (>, >=, < , <=) **ANY** (**SELECT**….)， = (>, >=, < ,<=) **SOME** (**SELECT**….)。

- 找出薪水大於部門 5 所有（任一）員工的薪水的員工姓名。

```
SELECT  LNAME, FNAME
 FROM   EMPLOYEE
 WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE
 WHERE DNO = 5);
```

4. In a nested query, an ambiguous, unqualified attribute refers to relation in the **innermost nested query**, e.g.:

- **Q16**：Retrieve the name of each employee who has a dependent with the same first name and same sex
  as the employee.

```
Ans.：  SELECT    E.FNAME, E.LNAME
          FROM    EMPLOYEE AS E
          WHERE E.ESS IN（SELECT    ESSN
                          FROM      DEPENDENT
                          WHERE     E.FNAME =
DEPENDENT_NAME AND E.SEX = SEX）;
```

E.SEX 中 E 表示從外部傳進來的，可用"別名"設定外部查詢之 Table，於同名稱屬性被引用到內部查詢時，加上別名區別之，以免與內部查詢之同名稱屬性相混淆）。

5. 有些帶有 **IN** 比較運算的巢狀查詢可用單一型（single block）查詢方式表示：

- **Q16A**：

```
SELECT    E.FNAME, E.LNAME
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE E.ESS = D.ESSN AND E.SEX = D.SEX AND
          E.FNAME = D.DEPENDENT_NAME）;
```

或

```
SELECT    FNAME, LNAME
FROM          （EMPLOYEE JOIN DEPENDENT ON SSN = ESSN）
WHERE     EMPLOYEE.SEX=DEPENDENT.SEX AND
FNAME=.DEPENDENT_NAME;
```

EXISTS Function in SQL

1. **Q16B**：**SELECT** E.FNAME, E.LNAME

```
            FROM    EMPLOYEE AS E
            WHERE   EXISTS （SELECT  *
                            FROM DEPENDENT
                            WHERE   E.SSN = ESSN
                                AND E.SEX = SEX
                            AND E.FNAME =DEPENDENT_NAME）;
```

參考右列 Relation **R**，執行下列 SELECT 敘述，結果如何？

```
SELECT    A
FROM      R
WHERE     EXISTS (SELECT      *
                  FROM      DEPARTMENT        （參考 205 頁）
                  WHERE     DNUMBER=B;
```

R

A	B
α	1
β	2
γ	3

Ans.：只有 α 。

2. **Q6**：Retrieve the name of employees who have no dependents.

```
Ans.：SELECT   FNAME, LNAME
      FROM         EMPLOYEE
      WHERE    NOT EXISTS（SELECT   *
                          FROM DEPENDENT
                          WHERE    SSN = ESSN）;
```

3. **Q7**：List the names of managers who have at least one dependent.

```
Ans.：     SELECT       FNAME, LNAME
.          FROM    EMPLOYEE
.          WHERE EXISTS        （SELECT   *
                               FROM       DEPENDENT
                               WHERE      SSN = ESSN）
                AND
                EXISTS         （SELECT   *
                               FROM       DEPARTMENT
                               WHERE      SSN = MGRSSN）;
```

Explicit Sets and NULLs in SQL

1. 在 WHERE 子句裡可用明確指定的值集合(explicit set of values)取代以巢狀查詢所得的結果，例：
   - **Q17**：Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Ans.：     SELECT      DISTINCT ESSN
           FROM        WORKS_ON
           WHERE       PNO IN (1,2,3);
```

2. 用 IS NULL 或 IS NOT NULL 來表示查詢條件裡屬性值是否爲 "NULL"。例：

   Q18：Retrieve the names of all employees who do not have supervisors.

```
Ans.：     SELECT       FNAME, LNAME
           FROM         EMPLOYEE
           WHERE        SUPERSSN IS NULL;
```

Renaming Attributes and Joined Tables

1. **AS** 可使用在 SELECT 子句及 WHERE 子句，用來設定屬性及關聯之別名，例：
   - **Q8A**：

```
SELECT   E.LNAME AS EMPLOYEE_NAME,
         S.LNAME AS SUPERVISOR_NAME
FROM     EMPLOYEE AS E, EMPLOYEE AS S
WHERE    E.SUPERSSN = S.SSN;
```

2. **JOIN…ON…**結構方式可用在 FROM 子句裡作爲定義 Table 之選擇及連結條件，例：
   - **Q1A**：

```
SELECT   FNAME, LNAME, ADDRESS
FROM     （EMPLOYEE JOIN DEPARTMENT
           ON DNO = DNUMBER）
WHERE    DNAME = ’Research’;
```

   (EMPLOYEE **JOIN** DEPARTMENT **ON** DNO = DNUMBER) 等同 (EMPLOYEE $\bowtie_{\text{DNO = DNUMBER}}$ DEPARTMENT)

3. 使用 **NATURE JOIN** 時常用 **AS** 更改別名，使兩關聯用同名的屬性作自然連結，例：
   - **Q1B**：

```
SELECT   FNAME, LNAME, ADDRESS
FROM     (EMPLOYEE NATURE JOIN(DEPARTMENT
AS DEPT(DNAME, DNO, MSSN, MSDATE)))
WHERE    DNAME = ’Research’;
```

   （此例是改名後，兩關聯以 DNO 作自然連結）。

4. 一般的 JOIN 是 **inner join**，若使用 **LEFT OUTER JOIN** 會把左 Table 的所有 Tuples 保留，例：
   - **Q8B**：

```
SELECT   E.LNAME AS EMPLOYEE_NAME,
         S.LNAME AS SUPERVISOR_NAME
FROM     (EMPLOYEE AS E LEFT OUTER JOIN
EMPLOYEE AS S ON E.SUPERSSN = S.SSN;
```

5. 有四種 JOIN 連結方式：INNER JOIN(同 JOIN), LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN。

Aggregate Function and Grouping

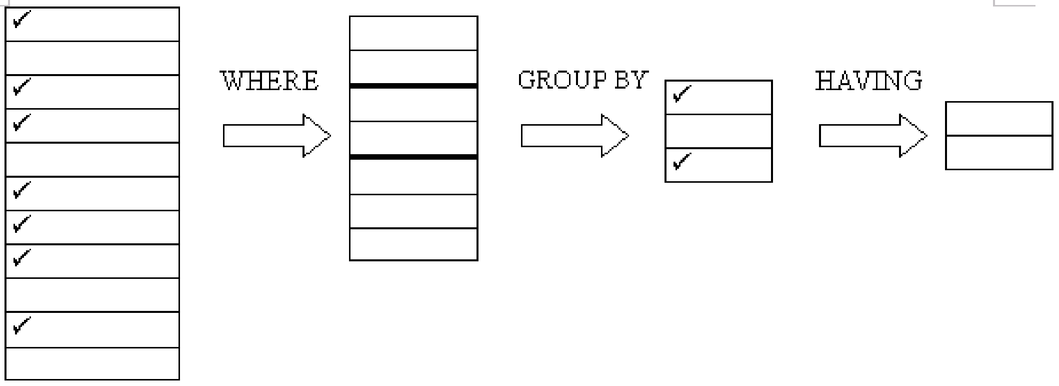
1. 此功能包含 **COUNT, SUM, MAX, MIN, AVG** 等，可使用在 **SELECT** 子句及 **HAVING** 子句。

General form：

```
SELECT     <attribute and aggregate function list>
FROM       <table list>
WHERE      <condition>
GROUP BY   <attribute list>
HAVING     <group condition >;
```

若要作分組的彙總功能運算就要加 **GROUP BY**

See the following figure for illustration.



- **Q19**：Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.
  SELECT **SUM**(SALARY)，**MAX**(SALARY)，**MIN**(SALARY)，**AVG**(SALARY)
  FROM EMPLOYEE;
- **Q20**：Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.
  SELECT **SUM**(SALARY)，**MAX**(SALARY)，**MIN**(SALARY)，**AVG**(SALARY)
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO = DNUMBER **AND** DNAME = 'Research';

2. **COUNT**（）counts the number of values in the parameter. 例：

- **Q21** and **Q22**：Retrieve the total number of employees in the company(Q21) and the number of employees in the 'Research' department(Q22).

  Ans.Q21： **SELECT** **COUNT**(＊)
  FROM EMPLOYEE
  Ans.Q22： **SELECT** **COUNT**(＊)
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO = DNUMBER **AND** DNAME = 'Research';

  上述 "＊"（Asterisk）表示以 row（tuple）來計算數目，也可用來計算單一 column，如下例：

- To count only the number of distinct values, the keyword '**DISTINCT**' has to be specified.
  （加一"DISTINCT" 表示不計入重複的值之數目）
  Q23：Count the number of distinct salary values in the database.
  Ans.： **SELECT** **COUNT**（**DISTINCT** SALARY）
  FROM EMPLOYEE;

3. **Aggregate functions** can also appear in the nested query in the **WHERE** clause.

- **Q5**：Retrieve the names of all employees who have two or more dependent.

```
Ans.：    SELECT      LNAME, FNAME
          FROM        EMPLOYEE
          WHERE       (SELECT     COUNT(∗)
                       FROM       DEPENDENT
                       WHERE      SSN = ESSN) >= 2;
```

用 SSN = ESSN 把 EMPLOYEE 及 DEPENDENT 串聯起來。

4. **Grouping attributes** 也要註明在 **SELECT** 子句內，以便彙總功能執行結果可跟隨 Grouping attributes 列出，例如：
   - **Q24**：For each department, retrieve the department number, the number of employees in the department,
     and their average salary.

     ```
     Ans.：    SELECT      DNO, COUNT(∗), AVG(SALARY)
               FROM        EMPLOYEE
               GROUP BY    DNO;       （參考 Figure 8.4(a)）
     ```
   - **Q25**：For each project, retrieve the project number, the project name, and the number of employees who work on that project.

     ```
     Ans.：    SELECT      PNUMBER, PNAME, COUNT(∗)
               FROM        PROJECT, WORKS_ON
               WHERE       PNUMBER = PNO
               GROUP BY    PNUMBER, PNAME;
     ```
5. **HAVING** provides a condition on the group of tuples associated with each value of the grouping attributes.
   - **Q26**：For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on that project.

     ```
     Ans.：    SELECT      PNUMBER, PNAME, COUNT(∗)
          （參考 Figure 8.4(b)）
               FROM        PROJECT, WORKS_ON
               WHERE       PNUMBER = PNO
               GROUP BY    PNUMBER, PNAME
               HAVING      COUNT (∗) > 2;
     ```
     （**HAVING** 是設定 **GROUP BY** 的條件）
   - **Q27**：For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on that project.

     ```
     Ans.：    SELECT      PNUMBER, PNAME, COUNT(∗)
               FROM        PROJECT, WORKS_ON,EMPLOYEE
               WHERE       PNUMBER = PNO AND SSN = ESSN AND
     DNO = 5
               GROUP BY    PNUMBER, PNAME;
     ```
   - **Q28**：For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

     ```
     Ans.：    SELECT      DNUMBER, COUNT(*)
               FROM        DEPARTMENT, EMPLOYEE
               WHERE       DNUMBER = DNO AND SALARY > 40000
     ```

```
AND
                        DNO IN   （SELECT   DNO
                                   FROM     EMPLOYEE
                                   GROUP BY DNO
                                   HAVING   COUNT(＊)＞5）
        GROUP BY  DNUMBER;
```

（先選出大於 5 個人的部門，再就此各部門計算出薪水大於$40,000 的人數）

What’s wrong with the following query:

```
SELECT    DNUMBER, COUNT(*)
FROM      DEPARTMENT, EMPLOYEE
WHERE     DNUMBER = DNO AND SALARY > 40000
GROUP BY  DNUMBER
HAVING    COUNT(＊)＞5;
```

（四）Insert，Delete，and Update Statements in SQL

The INSERT Command

1. **INSERT** 用來增加一個 tuple 到一個 relation，每個 value 的順序要按 **CREATE TABLE** 所定義的次序。
2. 標準語法：　**INSERT INTO** table-name[(attribute list)]
   VALUES (attribute-value-list);

   例：**U1:**　**INSERT INTO** EMPLOYEE
   VALUES　('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98 Oak Forest, Katy’, ‘TX’', 'M', '37000', '987654321', 4);
3. 若只增加部分 attributes 時，在括弧內的屬性名與值要對應（順序沒關係），其餘則為初值或 NULL。

   例：**U1A: INSERT INTO** EMPLOYEE (FNAME, LNAME, DNO, SSN)
   VALUES　('Richard', 'Marini', 4, '653298653');
4. 新增資料也受 SQL 所提供的完整性限制所約束（須滿足所有指定的限制時才可新增）。

   例：**U2:**　**INSERT INTO** EMPLOYEE (FNAME, LNAME, SSN, DNO)
   VALUES　('Robert', 'Hatcher', '980760540', 2);

   (U2 會失敗因參考完整性限制，DNO 沒有 2，故不能新增)

   U2A:　**INSERT INTO** EMPLOYEE (FNAME, LNAME, DNO)
   VALUES　('Robert', 'Hatcher', 2);

   （U2A 因沒有主鍵 SSN，故不能新增）
5. 一次增加多筆資料（產生一暫存 table 儲存查詢結果），例：

```
U3A:    CREATE TABLE DEPTS_INFO
                  (DEPT_NAME        VARCHAR(15),
                   NO_OF_EMPS       INTEGER,
                   TOTAL_SAL    INTEGER);
U3B:    INSERT INTO     DEPTS_INFO(DEPT_NAME,
NO_OF_EMPS, TOTAL_SAL)
```

```
            SELECT      DNAME, COUNT(＊), SUM(SALARY)
            FROM        (DEPARTMENT JOIN EMPLOYEE
ON DNUMBER = DNO)
            GROUP BY    DNAME;
```

The DELETE Command

1. **DELETE** 用來刪除一個 tuple（從一個 relation 裡），也受限於相關 relation 之參考完整性的參考機制。2.例：（參考 205 頁 EMPLOYEE）

- **U4A：**
```
DELETE FROM  EMPLOYEE
WHERE        LNAME = 'Brown';
```
（刪除 0 筆資料）

- **U4B：**
```
DELETE FROM  EMPLOYEE
WHERE        SSN = '123456789';
```
（刪除 1 筆資料）

- **U4C：**
```
DELETE FROM  EMPLOYEE
WHERE  DNO IN (SELECT    DNUMBER
               FROM      DEPARTMENT
               WHERE     DNAME = 'Research');
```
（刪除 4 筆資料）

U4D：
```
DELETE FROM  EMPLOYEE;
```
（刪除全部資料）

The UPDATE Command

1. **UPDATE** 指令用來修改 values of selected tuples，每次只能修改一個 relation 內的資料。
2. 標準語法：
```
UPDATE      table-name
SET         attribute = value, attribute = value,……
WHERE       condition;
```
3. 例：

- **U5：**
```
UPDATE      PROJECT
SET         PLOCATION = 'Bellaire', DNUM = 5
WHERE       PNUMBER = 10;
```
- **U6：**
```
UPDATE      EMPLOYEE
SET         SALARY = SALARY  ＊  1.1
WHERE       DNO IN   (SELECT        DNUMBER
                      FROM          DEPARTMENT
                      WHERE         DNAME = 'Research');
```

（五）Views（Virtual Tables）in SQL

View Concept in SQL

1. View 是虛擬的 Table，是由其他真實存在的 table 推導而來的，通常是由兩個或多個 tables（defining tables）JOIN 而來。
2. View 有兩種用途：
(1)常用的查詢把它定義成 views 方便使用者使用。
(2)某些特別的情況可定義在 views 上，形成一個外部綱目（external schema）。

Specification of Views in SQL

1. 語法：
```
CREATE VIEW   view-name
    AS        select-statement
```
例：
- V1：
```
CREATE VIEW       WORKS_ON1
AS  SELECT     FNAME, LNAME, PNAME, HOURS
    FROM       EMPLOYEE, PROJECT, WORKS_ON
    WHERE      SSN = ESSN AND PNO = PNUMBER;
```
產生的虛擬 Table 之綱目：
WORKS_ON1（FNAME, LNAME, PNAME, HOURS）
- V2：
```
CREATE VIEW  DEPT_INFO(DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
    AS  SELECT     DNAME, COUNT(＊), SUM(SALARY)
        FROM       DEPARTMENT, EMPLOYEE
        WHERE      DNUMBER = DNO
        GROUP BY   DNAME;
```
產生的虛擬 Table 之綱目：
DEPT_INFO（DEPT_NAME, NO_OF_EMPS, TOTAL_SAL）

2. 產生的 view，可以當作 base table 使用，可簡化查詢的定義敘述，如果 defining table（base table）之值有改變，則再次執行該 view 時看到的資料亦爲新的。
- QV1：
```
SELECT     FNAME, LNAME
FROM       WORKS_ON1
WHERE      PNAME = 'ProjectX';
```
3. 若要刪除一個 view 用 **DROP VIEW** 指令，例：
- V1A：`DROP VIEW     WORKS_ON1;`
- V2A：`DROP VIEW     DEPT_INFO;`

View Implementation and View Update

1. An update to a view may be ambiguous. Thus, in general, a view is updateable only when:
(1)The view is defined on a single table, and（定義在一個表單上）
(2)The view does not include aggregate functions in its definition, and（定義中不含彙總功能）
(3)The view attributes contains the primary key(or possibly some other candidate key)of the base relation.（屬性要包含所定義的表單之主鍵或副鍵）

2. 於 WORKS_ON1，更改 'John Smith' 的 PNAME 屬性，原爲 'ProductX' 改爲 'ProductY'.
UV1：
```
UPDATE     WORKS_ON1
SET        PNAME = 'ProductY'
WHERE      LNAME = 'Smith' AND FNAME = 'John' AND
PNAME = 'ProductX';
```
- 但 UV 1 爲不合法的修改，因爲將 WORKS_ON1 改成如 UV 1 至少有以下兩種方式：

```
UPDATE    WORKS_ON
SET    PNO=    (SELECT PNUMBER
                 FROM PROJECT
                 WHERE PNAME = 'ProductY')
WHERE ESSN IN (SELECT   SSN
                 FROM EMPLOYEE
                 WHERE LNAME = 'Smith'
                 AND FNAME = 'John')
       AND
       PNO IN  (SELECT PNUMBER
                 FROM PROJECT
                 WHERE PNAME = 'ProductX');
```

和

```
UPDATE    PROJECT
SET       PNAME = 'ProductY'
WHERE     PNAME = 'ProductX';
```

（六）Specifying General Constrains as Assertions

1. Constraints on multiple tables can be specified by using ASSERTION（強制規定）in SQL.
2. The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

   語法：
   ```
   CREATE ASSERTION    SALARY_CONSTRAINT
      CHECK（NOT EXISTS
           （SELECT  *
            FROM  EMPLOYEE E, EMPLOYEE M,   DEPARTMENT D
            WHERE   E.SALARY > M.SALARY AND
                 E.DNO = D.DNUMBER AND D.MGRSSN = M.SSN））；
   ```
3. 同樣的要對某些 DOMAIN 設限時：

   語法：
   ```
   CREATE DOMAIN    D_NUM AS INTEGER
   CHECK（D_NUM > 0 AND D_NUM < 21）；
   ```
4. Some active mechanism in response to some event can be specified by using TRIGGER（骨牌效應）in SQL.
   - **DEFINE TRIGGER** SALARY, **TRIGGER ON** EMPLOYEE E, EMPLOYEE M, DEPARTMENT D:
     E.SALARY > M.SALARY **AND** E.DNO = D.DNUMBER **AND** F.MGRSSN = M.SS
     ACTION_PROCEDURE INFORM_MANAGER(D.MGRSSN);
     （詳細請參考 23 章）

（七）Specifying INDEX

1. An index is a physical access structure that is specified on one or more attributes of a file. It can be used to accelerate the search of tuples based on the values of the attributes.
2. INDEX statements were dropped from SQL2 standard. Howere, almost all commercial DBMSs still include statements for specifying INDEX.

- I1: **CREATE INDEX** LNAME_INDEX
  ON EMPLOYEE(LNAME);
- I2: **CREATE INDEX** NAMES_INDEX (次序及遞增或遞減對速度很有影響)
  ON EMPLOYEE（LNAME ASC, FNAME DESC, MINIT）
- I4: **CREATE INDEX** DNO_INDEX
  ON EMPLOYEE（DNO）
  CLUSTER; （有關 Physical 的順序時）

五、Functional Dependencies and Normalization for Relational Database（CHAPTER 14, 15）

（一）Purpose

- In the mapping from ERD to relation schemas, we use some guidelines that were intuitively reasonable. However, no measure of the appropriateness or goodness of a design was introduced.（雖以直覺合理的指導原則將 ERD 轉成關聯綱目，卻未發展出衡量其設計品質、適當性、優良性的方法）。
- This chapter will identify a set of rules and theories for designing "good" relational schemas.（將訂出一套規則及理論來設計優良的關聯綱目）本章分兩個層面討論關聯綱目的"優良性"：
  A. 觀念層(conceptual/logical level)：如何解釋關聯綱目及其屬性含意。
  B. 執行層(implementation/storage level)：在一個基本關聯(base relation)內，如何儲存及更新資料。

（二）Informal Design Guidelines for Relational Schemas

- Discuss four informal（非正式的）measures of quality for relation schema design：
  1. Semantics（語意） of the attributes.
  2. Reducing the redundant values in tuples.
  3. Reducing the null values in tuples.
  4. Disallowing the possibility of generating spurious (假的) tuples.(不要產生原來沒有的 tuples)

（以上四種衡量方法有相依關係，討論如下）

Semantics of The Relational Attributes

- **Guideline 1:** Design a relation schema that is easy to explain. Do not combine attributes from multiple entity types and relationship types into a single relation.（不要把分屬多個實體型態和關係型態的屬性結合成一個關聯）。
- 參考課本 Figure 14.3.(a)，(b)：

  EMP_DEPT（ENAME, <u>SSN</u>, BDATE, ADDRESS, DNUMBER, DNAME, DNGRSSN）

  EMP_PROJ（<u>SSN, PNUMBER</u>, HOURS, ENAME, PNAME, PLOCATION）

  EMP_DEPT 混合了 employees 和 departments 的屬性，EMP_PROJ 混合了 employees 和 projects 的屬性。
- EMP_DEPT and EMP_PROJ can be used as views, but they cause problems when used as base relations.

Redundant Information in Tuples and Update Anomalies

- **Guideline 2:** Design the base relation schemas so that no update anomalies occur. Update anomalies refer to insertion, deletion, and modification anomalies（設計的基本關聯綱目，於插入、刪除、修正等動作時，要沒有異常情形發生）
- Update anomalies occur due to the redundant information stored in a relation.
- Compare the storage space needed by EMPLOYEE and DEPARTMENT in Figure 14.2 and EMP_DEPT in Figure 14.4.（EMP_DEPT 相當於 [EMPLOYEE **NATURAL JOIN** DEPARTMENT]，Figure 14.4.所佔檔案儲存空間較 Figure 14.2 爲多）。
- In addition to space waste, poor design can cause update anomalies（修改異常）:
- Insert Anomalies: Redundant information must be inserted correctly.(重複部分的資料每次輸入時要確保一致相同）
- No pure department information can exist by itself.（無法增加一沒員工的部門）
- Deletion Anomalies: Deletion of tuples may cause some information to lose forever.(若某部門僅一員工，當此員工被刪除時，則此部門之資料也同時不存在了)。
- Modification Anomalies: Change of the information about a department may involve the modification to several tuples.（若某部門經理換人時，則該部門所有員工之經理皆要修正）

Null Values in Tuples

- **Guideline 3**：As far as possible, avoid placing attributes in a base relation whose values may frequently be null.（容許少數記錄有例外狀況，不要多數都是空值）
- Null values waste space.
- Nulls cause ambiguous interpretation to aggregate operations such as COUNT or SUM.（在做彙總運算時，空值到底怎麼算呢？COUNT 時不算是一筆）
- Nulls comprise（包括）different meanings.（不適用、不知道、知道但從缺未登記）

Generation of Spurious Tuples

- **Guideline 4**：Design relation schemas so that they can be JOINed with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees no spurious tuples are generated.（以 primary keys 或 foreign keys 作 JOIN 才不致有多餘的 tuples）
- See EMP_LOCS and EMP_PROJ1 in Figure 14.5. By natural joining EMP_LOCS and EMP_PROJ1, more tuples than EMP_PROJ in Figure 14.4 are generated. These tuples are called spurious tuples.
- The design is bad because PLOCATION is neither a primary key nor a foreign key in both relations.

綜上討論指導原則有幾個問題

- 所謂"異常"就是說在做插入及修改時要做一些額外的工作，而在做刪除時會意外遺失某部分資料。
- 由於"空值"導致浪費儲存空間，同時也導致做彙總運算和配合（join）時的困擾。
- 以不適當相關的基本關聯做配合時，會產生不對及多餘的資料。

（三）Functional Dependencies：

- Functional dependency is a formal concept that can be used to define the “goodness” and “badness” of individual relation schema more precisely.（函數相依是一個正式的概念，用來更精確定義個別的關聯綱目）
- Several normal forms are based on the concept of functional dependency.
- A functional dependency, denoted as X→Y, between two sets of attributes X and Y that are subsets of R（universal relation schema）specifies for any two tuples t1 and t2 in r(R) such that t1[X] = t2[X], we must also have t1[Y]=t2[Y].（在 R 中某一個 tuple 的 Y 屬性組的值由 X 屬性組的值決定，或者說，X 屬性組的值唯一決定 Y 屬性組的值）
- With X→Y, we say X determines Y or Y is functionally dependent on X.（We also say that there is a functional dependency from X to Y）
- Note that a primary (candidate) key of a relation R determines any set of attributes in R.（主鍵可決定其所在關聯之所有屬性）
  - See Figure 14.3 (a), (b) for diagrammatic representation.（Both suffer from update anomalies）
  - A functional dependency is a semantic property（或屬性的意義）of the relation schema and cannot be inferred（推斷出）automatically from a given relation extension. See Figure 14.7, where TEXT→COURSE may be incorrectly inferred. （就目前所給的 tuples 是成立，但以後就不一定了）
  - Functional dependencies are abbreviated as **FD**（or **f.d.**）.

Inference Rules for Functional Dependencies

- More functional dependencies can be inferred from a set of given dependencies.（給一些 FDs 可推導出更多 FDs，即 $F \rightarrow F^+$），例如下列關聯，可清楚定義出 F 如下：

  EMP_DEPT(ENAME, <u>SSN</u>, BDATE, ADDRESS, DNUMBER, DNAME, DMGRSSN)

  F={ SSN→{ENAME, BDATE, ADDRESS, DNUMBER}, DNUMBER→{DNAME, DMGRSSN}}

  我們可從 F 中推導出： SSN→{DNAME, DMGRSSN}, SSN→SSN, DNUMBER→DNAME.
- The set of FDs that can be inferred from a set of given dependencies F is called the **closure of F**, **denoted as** $\mathbf{F^+}$.（在現實生活中要把所有可能的 FDs 定義出來是很不可能的，故稱爲 F 的"封閉集合"）
- We use the notation $F \models X \rightarrow Y$ to denote that the functional dependency X→Y is inferred from the set of functional dependencies F.（爲方便起見可將 FD{X,Y}→Z 簡寫爲 XY→Z，FD{X,Y,Z}→{U,V}簡寫爲 XYZ→UV.）

- **Six well-known inference rules for functional dependencies**：\
  - IR1 (reflexive rule)：If $X \supseteq Y$，then $X \rightarrow Y$.(A set of attributes always determined itself or any of its subsets)
  - IR2 (augmentation rule)：$\{X \rightarrow Y\} \models XZ \rightarrow YZ$.(Adding the same set of attributes to both the left-handand right-hand sides of a dependency result in another valid dependency.亦可寫成$\{X \rightarrow Y\} \models XZ \rightarrow Y$. 也就是說只增加左邊之屬性).
  - IR3 (transitive rule)：$\{X \rightarrow Y , Y \rightarrow Z\} \models X \rightarrow Z$.(functional dependencies are transitive；移轉規則)
  - IR4 (decomposition, or projective, rule)：$\{X \rightarrow YZ\} \models X \rightarrow Y$.(We can remove attributes from the right-hand side of a dependency. For example, repeatedly decompose the FD $X \rightarrow \{A1, A2, \ldots, An\}$ into the set of dependencies$\{X \rightarrow A1, X \rightarrow A2, \ldots, X \rightarrow An\}$.
  - IR5 (union, or additive, rule)：$\{X \rightarrow Y , X \rightarrow Z\} \models X \rightarrow YZ$.(Allows us to do the opposite for IR4)
  - IR6 (pseudotransitive rule)：$\{X \rightarrow Y , WY \rightarrow Z\} \models WX \rightarrow Z$.（假移轉規則）
- The above inference rules IR1 through IR3 are called **Armstrong's inference rules**. （IR4, IR5, IR6 是利用 IR1, IR2, IR3 推導出來的）
- When given a set of FD F, for each $X \rightarrow Y$ in F, we can use Armstrong's inference rules to determine the set of attributes that are dependent on X, denoted as X^+. X^+ is called the closure of X under F.
- 參考前段 6.(b)，其 F 定義如下：
  F={ SSN→ENAME, PNUMBER→{PNAME, PLOCATION}, {SSN, PNUMBER}→HOURS}
  利用 Algorithm 14.1（參考課本 481 頁 determining X^+.）可算出下列關於 F 的封閉集合：
  $\{SSN\}^+$ = {SSN, ENAME},
  $\{PNUMBER\}^+$ = {PNUMBER, PNAME, PLOCATION},
  $\{SSN, PNUMBER\}^+$ = {SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS}.
- Two functional dependencies E and Fare **equivalent** if $E^+ = F^+$.（Every FD in E can be inferred from F, and every FD in F can be inferred from E；that is , E is equivalent to F if both the conditions E covers F and F covers E hold.）
- A set of FD F is **minimal** if it satisfies the following conditions：
  (1) Every dependency in F has a single attribute for its right-hand side.
  (2) We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.
  (3) We can not remove any dependency from F and still have a set of dependencies that is equivalent to F.
- 求某一 FD 最小化方式請參考課本 483 頁 Algorithm 14.2.

（四）Normal Forms Based on Keys

- The normalization process was first proposed by Codd（1972a）. It takes a relation schema through a series of tests to "certify" whether it satisfies a certain **normal form.**（所謂正規化是經連續測試証明某一關聯綱目是否滿足某一正規的格式）
- 1NF,2NF, 3NF, BCNF are based on the concept of functional dependencies among the attributes of a relation. The 4NF and 5NF are based on the concepts of multivalued dependencies and join dependencies , respectively.
- Normalization of data can be seen as a process to decompose an unsatisfactory relation into smaller relations that possess the desirable properties.
- 正規化過程提供給資料庫設計者的是：
  - 分析一個關聯綱目的正規做法是以其 "key" 及其 "屬性間函數相依" 為基礎。
  - 每個關聯綱目皆可連續以正規的格式測試分割，故一關聯資料庫可正規化成某一理想數目的關聯綱目。
- Database designers need not always normalize to the highest possible normal form due to performance reasons or the complexity for checking constraints.（不需正規化到最高度，考慮執行效果的話適度即可）
- Recall the definitions about superkey, key, primary key, and candidate key.
  - superkey: 在一關聯內可決定其 tuple 值是唯一的一組屬性。
  - key: 具有最少數屬性的 superkey。
  - candidate key: 在一關聯內的 key 若多於一個時，每個 key 皆稱為 candidate key。
  - primary key: 從 candidate key 任意選出一個當 primary key。
- An attribute of R is called a **prime attribute**（主要屬性 if it is a member of any key of R.（否則稱為 **nonprime attribute**）

First Normal Form (1NF)

- **Def.(1NF)**：The domain of each attribute must be simple and single-valued.（不容許一個關聯內含有其他關聯，或某些 tuples 的 attributes 是一個關聯，屬性值必須是單一值。）
  - 1NF conforms to the definition of relations.（與關聯的定義相同，屬性不能是多值或複合屬性）
- See Figure 14.8, 14.9 for the handling of multi-valued attributes，屬性多值時解決方式：
  - ■ 把多值屬性移出，併同原關聯 primary key 另成一新關聯，新關聯的所有屬性當作其 primary key。(參考 figure 14.2)。
  - ■ 擴充 key 使包含此多值屬性。(參考 figure 14.8(c))。
  - ■ 確定此多值屬性之最多可能個數，增加該數個相同屬性（不同名稱）來存放其值。(缺點是有些 tuples 的這些屬性會產生空值)。

Second Normal Form (2NF)

- **Def**. (full functional dependency)：A FD $X \rightarrow Y$ is a **full functional dependency** if the removal of any attribute from X means the dependency does not hold any more.【That is, for any attributes $A \in X$, $(X-\{A\})$does not functionally determine Y】。

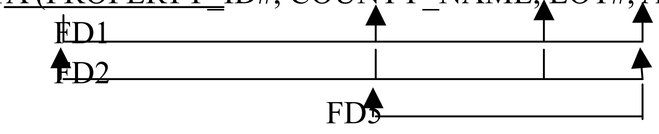
- 參考課本 Figure 14.3.
- **Def. (2NF)**：A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on every key of R.
- 參考 Figure 14.8(c) DEPARTMENT(DNAME, DNUMBER, DMGRSSN, DLOCATION)，此例中僅 DNUMBER 即可決定 DNAME, DMGRSSN 不必加上 DLOCATION 為 key。
- 參考 Figure 14.10(a)　(同上) nonprime attribute ENAME 僅由 SSN 即可決定，nonprime attribute PNAME, PLOCATION 僅由 PNUMBER 即可決定，故不符合 2NF 測試。若執行 2NF 正規化，將其分割成 EP1, EP2, EP3 即符合 2NF 測試。
- 參考 Figure 14.9(b),若選 SSN, PNUMBER 為 key，就不符合 2NF 測試，因 SSN 即可決定 ENAME。
- 參考 Figure 14.11(a). 由其 FD2 可知 COUNTY_NAME, LOT#是 secondary key, 對 TAX_RATE 來說，若沒有 LOT#，照樣可由 COUNTY_NAME 維持原函數相依(FD3:COUNTY_NAME→TAX_RATE)，故為 partial dependency，不符合 2NF 測試，分割成(b)的 LOTS1 及 LOTS2 即符合 2NF 測試。

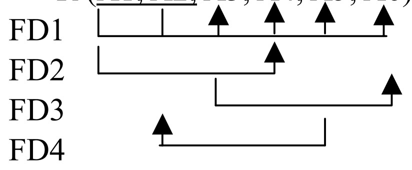
Third Normal Form (3NF)

- **Def.**(transitive dependency；遞移相依)：A FD X→Y is a **transitive dependency** if there exists a set of attributes Z that is not a subset of any key of R, and both X→Z and Z→Y hold.
- 例如參考 Figure 14.10 (b), 由 SSN→DNUMBER，再 DNUMBER→DMGRSSN，遞移結果為 SSN→DMGRSSN，這 DNUMBER 既非 EMP_DEPT 的 key，也不是其 key 的子集合，這即是 transitive dependency。
- **Def.(3NF)**：A relation schema R is in 3NF if it is in 2NF and no nonprime attribute of R is transitively dependent on some key.
- 上例之關聯是滿足 2NF(since no partial dependencies on a key exist)，但因透過 DNUMBER 產生遞移相依從 SSN 到 DMGRSSN, DNAME，故無法通過 3NF 測試。參考課本圖 14.10(b) 若將其執行 3NF 正規化分割成 ED1 及 ED2 兩個關聯，則各別都通過 3NF 測試。若執行 ED1＊ED2，將可回復原來之關聯。
- 參考 14.11(b)，由 FD4 知 AREA 是 nonprime attribute 可決定 PRICE，會產生遞移相依，若將其執行 3NF 正規化分割成 Figure 14.11(c)之 LOTS1A 及 LOTS1B 兩個關聯，則各別都通過 3NF 測試。

- .The definition of 3NF is equivalent to the following：
  For each FD X→Y in R, either
  (a) X is a superkey of R or
  (b) Y is a prime attribute of R.
  This definition can be applied directly to test whether a relation schema is in 3NF, it does not have to go through 2NF first.

（五）Boyce-Codd Normal Form (BCNF)

- **Def.(BCNF)**：A relation schema R is in **BCNF** if for each dependency X→Y in R, X must be a superkey in R.（去除 3NF 中 " Y is a prime attribute of R" 之可能性，限制更多）。
- See Figure 14.12 for an example.
  (a) LOTS1A (PROPERTY_ID#, COUNTY_NAME, LOT#, AREA)
  

  雖然 COUNTY_NAME 是 LOTS1A 的 prime attribute（因 COUNTY_NAME+LOT#爲 secondary key），但因 AERA 不是 LOTS1A 的 superkey，FD5 是滿足 3NF，但違反 BCNF，可將 LOTS1A 分割成下列兩個 BCNF 關聯:
  LOTS1AX (PROPERTY_ID#, AREA, LOT#)
  LOTS1AY (AREA, COUNTY_NAME)
  AREA, LOT# 爲 Secondary key。
- 再參考此關聯: R (A1, A2, A3, A4, A5, A6)
  

  FD2 是第二正規化要做的，FD3 是第三正規化要做的，FD4 則屬 BCNF 的範圍。
- The decomposition of schema to comply with BCNF may introduce the violation of dependency preservation, which dictate that a FD now spans two or more relations. For instance, (COUNTRY_NAME, LOT#)-> AREA now cross two relations (LOTS1AX, LOTS1AY) in the new schema. Therefore, some practitioners oppose to the decomposition for BCNF.

（六）Fourth Normal Form (4NF)

- 4NF is based on a concept called **multi-valued dependency**（**MVD**）.
- In a relation, the value of some attribute may determine a set of values in another attributes, regardless of the values in the remaining attributes. See EMP shown in Figure 15.4(a).
- **Def.** (multivalued dependency): a MVD, denoted as X↠Y specified on relation schema R, states the following condition holds:
  If two tuples t1 and t2 exist in r(R) such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties:
  t3[X] = t4[X] = t1[X] = t2[X].
  t3[Y] = t1[Y] and t4[Y] = t2[Y].
  t3[R-XY] = t2[R-XY] and t4[R-XY] = t1[R-XY].
- Whenever X↠Y, we say X multivalued determines(**multidetermines**)Y or Y is multivalued dependent on X.
- Note that according to this definition, whenever X↠Y holds in R, so does X↠(R-XY). Let R-XY be Z. Hence X↠Y implies X↠Z, and sometimes

written as $X \twoheadrightarrow Y | Z$.

- A MVD $X \twoheadrightarrow Y$ in R is called a **trivial MVD** if (a) Y is a subset of X or (b) $X \cup Y = R$.
- A **nontrivial MVD** in a relation implies the existence of redundancies. See Figure 15.5.
- Def.(**4NF**)：A relation schema R is in 4NF with respect to a set of dependencies F if, for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ in F+, X is a superkey for R.
- Note that a schema is BCNF if it is 4NF because functional dependency is a special case of multi-valued dependency.
- See Figure 15.5 again to see how 4NF saves space.
- Notice that relations containing nontrivial MVDs tend to be "**all key**" relations.
- Figure 15.4 (c) is in 4NF since it does not have nontrivial MVDs.

（七）Fifth Normal Form (5NF)

- 5NF is based on a concept called **join dependency**.
- Check whether or not SUPPLY in Figure 15.4(c) can be equally represented by the three relations R1, R2, and R3 in Figure 15.4(d).
- A join dependency (JD), denoted as JD(R1, R2, …, Rn) specified on a relation schema R, states r(R) = r(R1) * r(R2) * … *r(Rn).
- A JD(R1, R2, …, Rn) is trivial if one of the relation schema Ri is equal to R.
- Def.(**5NF**): A relation schema R is in 5NF (or called project-join normal form) with respect to a set F of dependencies if, for every nontrivial join dependency JD(R1, R2, …, Rn) in F^+, every Ri is a superkey of R.
- Discovering JDs in practical databases with hundreds of attributes is difficult; hence current practice of database design pays little attention to 5NF.

六、XML

（一）Introduction

- A XML document is a text file with XML markups. The file name usually ends with the extension .xml.
- XML stands for extensible markup languages.
- Unlike HTML, XML allows authors to create new tags.
- XML documents focus on the content of the data. The presentation of the data is not its concern. Usually, a separate .css (cascading style sheet) or .xsl (extensible style sheet language) file is used to specify the presentation of a XML document.
- Example letter.xml and usage.xml:

```
<?xml version="1.0" ?>
<!--  Fig. 5.6: letter.xml   -->
<!--  Business letter formatted with XML -->
<letter>
  <contact type="from">
      <name>Jane Doe</name>
      <address1>Box 12345</address1>
      <address2>15 Any Ave.</address2>
      <city>Othertown</city>
      <state>Otherstate</state>
      <zip>67890</zip>
      <phone>555-4321</phone>
      <flag gender="F" />
    </contact>
  <contact type="to">
      <name>Jane Doe</name>
      <address1>123 Main St.</address1>
      <address2 />
      <city>Anytown</city>
      <state>Anystate</state>
      <zip>12345</zip>
      <phone>555-1234</phone>
      <flag gender="M" />
    </contact>
    <salutation>Dear Sir:</salutation>
  <paragraph>
      It is our privilege to inform you about our new
      database managed with <bold>XML</bold>
      . This new system allows you to reduce the load
      on your inventory list server by having the client
      machine perform the work of sorting and
      filtering the data.
    </paragraph>
    <paragraph>The data in an XML element is
      normalized, so plain-text diagrams such as /---\ |
      | \---/ will become gibberish.</paragraph>
    <closing>Sincerely</closing>
    <signature >Ms. Doe</signature>
  </letter>
```

```
<?xml version = "1.0"?>
<!-- Fig. 5.5 : usage.xml                  -->
<!-- Usage of elements and attributes -->

<?xml:stylesheet type = "text/xsl" href = "usage.xsl"?>
<book isbn = "999-99999-9-X">
    <title>Deitel&apos;s XML Primer</title>
    <author>
        <firstName>Paul</firstName>
        <lastName>Deitel</lastName>
    </author>
    <chapters>
        <preface num = "1" pages = 2">Welcome</preface>
        <chapter num = "1" pages = "4">Easy
     XML</chapter>
        <chapter num = "2" pages = "2">XML
     Elements?</chapter>
        <appendix num = "1" pages =
     "9">Entities</appendix>
    </chapters>
    <media type = "CD"/>
</book>
```

（二）Parsing XML documents

- An syntactically correct XML document is called well-formed.
- There are many XML parsers freely available on the internet.

- IE 5.0 has a built-in XML parser (called msxml) that parses and renders a XML document.
- A parser can parse an XML document into a tree structure stored in main memory for further manipulation. A noted model for such tree structures is called DOM (document object model). Many parsers are written by Java, Perl, VB, etc and can be integrated into your application programs.

（三）Ingredients of XML documents

- XML documents contain all visible characters.
- There are five reserved characters: &, <, >, ‘, and “. To use these characters, you must use &amp; (for &) &lt;(for <) &gt; (>) &apos; (‘) &quot;(“) instead.
- Each element may have a value and a set of attributes.
- If a string is not to be parsed by the parser, simply put it inside a CDATA section.
- Example

```
<?xml version="1.0" ?>
<!--  Fig. 5.7 : cdata.xml      -->
<!--  CDATA section containing C++ code     -->
<book title="C++ How to Program" edition="3">
   <sample>// C++ comment if ( this->getX() < 5 &&
     value[ 0 ] != 3 ) cerr <<
     this->displayError();</sample>
   <sample>
       <![CDATA[

                   // C++ comment

                   if ( this->getX() < 5 && value[ 0 ] != 3 )

                       cerr << this->displayError();

       ]]>
   </sample>
   C++ How to Program by Deitel & Deitel
</book>
```

（四）Namespace

- To prevent naming collisions for synonyms, XML provides namespace mechanisms.
- A namespace has a prefix that can be prepended to element and attribute names to prevent potential confusion.
- Each namespace prefix is tied to a uniform resource identifier (URI) that uniquely identifies the namespace.
- A common practice is to use Universal Resource Locators (URLs) for URIs.

```
<?xml version = "1.0"?>
<!-- Fig. 5.8 : namespace.xml -->
<!-- Namespaces                    -->
<directory xmlns:text = "urn:deitel:textInfo"
               xmlns:image = "urn:deitel:imageInfo">
    <text:file filename = "book.xml">
        <text:description>A book list</text:description>
    </text:file>
    <image:file filename = "funny.jpg">
        <image:description>A funny picture</image:description>
        <image:size width = "200" height = "100"/>
    </image:file>
</directory>
```

（五）Document Type Definition (DTD) Introduction

- A DTD is used to define an XML document’s structure.
- To ensure effective B2B transactions, it is strongly recommended to use DTD to ensure document conformity.
- An XML document is considered valid if it conforms to its associated DTD.
- An example

```
<?xml version = "1.0"?>
<!-- Fig. 6.1: intro.xml        -->
<!-- Using an external subset -->
<!DOCTYPE myMessage SYSTEM "intro.dtd">
<myMessage>
    <message>Welcome to XML!</message>
```

```
</myMessage>

<!-- Fig. 6.2: intro.dtd      -->
<!-- External declarations -->
<!ELEMENT myMessage ( message )>
<!ELEMENT message ( #PCDATA )>
```

- PCDATA stands for parsable character data.
- DTD allows the definition of multi-valued, composite attributes.
  - + indicates that an element can repeat at least one time.
  - * indicates that an element can repeat at least 0 time.
  - ? indicates that an element can optionally appear. If used, appears only once.
- For example: mixed.xml

```
<?xml version = "1.0" standalone = "yes"?>
<!-- Fig. 6.5: mixed.xml                -->
<!-- Mixed content type elements -->
<!DOCTYPE format [
    <!ELEMENT format ( #PCDATA | bold | italic )*>
    <!ELEMENT bold ( #PCDATA )>
    <!ELEMENT italic ( #PCDATA )>
]>
<format>
    This is a simple formatted sentence.
    <bold>I have tried bold.</bold>
    <italic>I have tried italic.</italic>
    Now what?
</format>
```

（六）DTD Attribute Declarations

- Use ATTLIST to declare attributes.
- Attribute defaults:
  - #REQUIRED: must appear
  - #IMPLIED: If missing, any value can be used.
  - #FIXED: the value is fixed.

- Tokenized attribute type
    - ID: this attribute uniquely identifies an element.
    - IDREF: a pointer to elements with an ID attribute
    -
- For example

```
<?xml version = "1.0"?>
<!-- Fig. 6.8: IDExample.xml                          -->
<!-- Example for ID and IDREF values of attributes -->
<!DOCTYPE bookstore [
    <!ELEMENT bookstore ( shipping+, book+ )>
    <!ELEMENT shipping ( duration )>
    <!ATTLIST shipping shipID ID #REQUIRED>
    <!ELEMENT book ( #PCDATA )>
    <!ATTLIST book shippedBy IDREF #IMPLIED>
    <!ELEMENT duration ( #PCDATA )>
]>
<bookstore>
    <shipping shipID = "s1">
        <duration>2 to 4 days</duration>
    </shipping>
    <shipping shipID = "s2">
        <duration>1 day</duration>
    </shipping>
    <book shippedBy = "s2">
        Java How to Program 3rd edition.
    </book>
    <book shippedBy = "s2">
        C How to Program 3rd edition.
    </book>
    <book shippedBy = "s1">
        C++ How to Program 3rd edition.
    </book>
</bookstore>
```

- Enumerated attribute types

```
<!ATTLIST person gender (M | F) “F”>
```

```
<!ATTLIST book reference NOTATION (JAVA | C) “C”>

<!NOTATION C SYSTEM
http://www.addison-wesley.com/books/C_TOC.htm>
```

- Attribute type NMTOKEN (name token) is a value consisting of letters, digits, periods, underscores, hyphens and colon characters.
  <!ATTLIST sportsClub phone NMTOKEN #REQUIRED>
- For example

```
<?xml version = "1.0"?>
<!-- Fig. 6.14 : whitespace.xml          -->
<!-- Demonstrating whitespace parsing -->
<!DOCTYPE whitespace [
   <!ELEMENT whitespace ( hasCDATA,
      hasID, hasNMTOKEN, hasEnumeration, hasMixed )>
   <!ELEMENT hasCDATA EMPTY>
   <!ATTLIST hasCDATA cdata CDATA #REQUIRED>
   <!ELEMENT hasID EMPTY>
   <!ATTLIST hasID id ID #REQUIRED>
   <!ELEMENT hasNMTOKEN EMPTY>
   <!ATTLIST hasNMTOKEN nmtoken NMTOKEN #REQUIRED>
   <!ELEMENT hasEnumeration EMPTY>
   <!ATTLIST hasEnumeration enumeration ( true | false )
               #REQUIRED>
   <!ELEMENT hasMixed ( #PCDATA | hasCDATA )*>
]>
<whitespace>
   <hasCDATA cdata = "    simple cdata    "/>
   <hasID id = "    i20"/>
   <hasNMTOKEN nmtoken = "      hello"/>
   <hasEnumeration enumeration = "      true"/>
   <hasMixed>
      This is text.
      <hasCDATA cdata = " simple        cdata"/>
      This is some additional text.
```

```
    </hasMixed>
</whitespace>
```

- An alternative to DTDs is *schamas*. It is expected that schemas will replace DTDs as the primary means of describing document structure.
- Schema uses XML syntax.

七、Enhanced ER Model (Chapter 4)

- The EER(Enhanced-ER)model includes all the modeling concepts of the ER model. In addition, it includes the concepts of **subclass** and **superclass** and the related concepts of **specialization** and **generalization**, and the concepts of **category**（the union of objects of different entity types）. 基本上，就是比 ER model 多了物件導向觀念。

（一）Subclasses，Superclasses，and Inheritance

- Some entities in an entity type (say A) can be grouped to form another entity type (say B). We call B is a subclass of A, and A is a superclass of B. 例如 EMPLOYEE 的成員內可分成 SECRETARY, ENGINEER, MANAGER, TACHNICAN, SALARIED_EMPLOYEE, …等部分成員，各部分成員稱爲 subclass，EMPLOYEE 稱爲 superclass。此兩 class 之間有 **superclass/subclass relationship**（或簡稱 **class/subclass relationship**），有時也叫做 IS-A relationship。
- A subclass member is the same as the entity in the superclass. An entity cannot just exist in a subclass without being a member of its superclass.
- An entity can be included optionally as a member of any number of subclasses. 例如某一 engineer 會同時屬於 ENGINEER 及 SALARIED_EMPLOYEE 兩個 subclass。
- It is not necessary that every entity in a superclass be a member of some subclass. See Figure 4.2.
- A member of a subclass inherits all the attributes of the entity as a member of the superclass. The entity also inherits all relationship instances for relationship types in which the superclass participates.（**subclass** 繼承 **superclass** 的屬性和關係型態）

（二）Specialization and Generalization

- **Specialization**（特殊化；分成數個 subclass 的過程）is the process of defining *a set of subclasses* of an entity type, based on some distinguishing characteristic（依某些特性特殊化）.
- For example, {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE（依工作類別分）, while {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE} is another specialization of EMPLOYEE（依給資方式分. See Figure 4.1.
- Two main reasons for including class/subclass relationships and specialization in a data model:
  （使用 subclass 的原因）
  - Certain attributes may apply to some but not all entities of the superclass entity type.(某些屬性只用在 superclass 的部分實体中，而 subclass 就是定義有用到此屬性的實体群組)。
  - Some relationships types may be participated in only by entities that are members of the subclass. (某些關係型態只被某個 subclass 的實体所

參與)。

- Attributes that apply only to the subclass, are called **specific attributes** (or **local attributes**). A subclass can participate in **specific relationship types**.
- The specialization process involves（特殊化處理包括事項）:
  - Define a set of subclasses
  - Associate additional specific attributes with each subclass.
  - Establish additional specific relationship types between each subclass and other entity types.（要注意某些 subclass 有沒有和其他 subclass 或其他 entity 有專屬的關係）
- **Generalization** （一般化；特殊化的反向處理）refers to the process of defining a generalized entity type from the given entity types.（參考課本 Figure 4.3 Generalizing CAR and TRUCK into VEHICLE）
- See Figure 4.1 shows how we represent a specialization diagrammatically in an ERR diagram（SECRETARY 的屬性有 5 個加 3 個，即 EMPLOYEE 之所有屬性加上其 **specific attribute**， TECHNICAN 及 ENGINEER 也相同。如果 EMPLOYEE 有關係型態的話也會被繼承）

（三）Three Constraints on Specializations

本節所討論到的 constraints 也**同樣可應用到 generalization** 處理上。因係按某些屬性 specialization 的，故會有 constraints。

Constraints on Specialization

- **Defining Constraints**：
  - **predicate-defined** subclasses: a predicate on the attributes of superclass is used for determining the subclasses.(就以某些屬性形成條件設限組成一個 subclass，亦稱爲 **condition-defined** subclass）Most predicate-defined subclasses are **attribute-defined**. 意即此 predicate 是由單一屬性所構成，此屬性稱爲這特殊化的 **defining attribute**。參考 Figure 4.4.，其中 JobType 爲 defining attribute。
  - **user-defined** subclasses: Membership is specified individually for each entity by the user.（資料庫使用者需就每一 entity 個別指定其獨有的成員關係）
- **Disjointness Constraints**： Subclasses of the specialization are disjoint. That is, an entity can be a member of at most one of the subclasses of the specialization.（superclass 的成員最多屬於其中一個 subclass，不可同屬於超過一個 subclass）
  - Use 'd' symbol in the circle to denote disjointness constraint.
  - Use 'o' for **overlap (i.e. non-disjoint)** constraints.(參考課本 Figure 4.4. 及 Figure 4.5）
  - 一個 attribute-defined 的特殊化必定是有 disjointness constraint，因爲 defining attribute 是單值。
- **Completeness Constraints**：
  - A **total specialization** constraint specifies that every entity in the superclass must be a member of some subclass in the specialization. It

is denoted by a double line in the EERD.（參考課本 Figure 4.1）

- A **partial specialization** allows an entity not to belong to any of the subclasses. It is denoted by a single line in the EERD.(參考課本 Figure 4.1 及 Figure 4.4）
- **A generalization superclass usually is total**, because the superclass is derived from the subclasses and hence contains only the entities that are in the subclasses. （參考課本 Figure 4.3）

- Disjointness and completeness constraints are independent. 會有下列四種特殊化情況：
  - Disjoint, total.
  - Disjoint, partial.
  - Overlapping, total.
  - Overlapping, partial.
- Rules for deleting or inserting entities to a specialization:
  - 從 superclass 刪除一個成員(元件)，即等於從該成員所在的所有 subclass 中刪除該成員。
  - Inserting an entity in a superclass implies that the entity is mandatorily inserted in all predicate-defined (or attribute-defined) subclass for which the entity satisfies the defining predicate.
  - Inserting an entity in a superclass of a total specialization implies that the entity is mandatorily inserted in at least one of the subclass of the specialization.

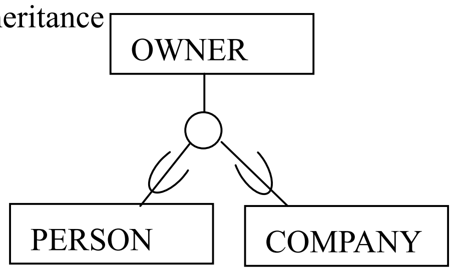
Specialization Hierarchies and Lattices

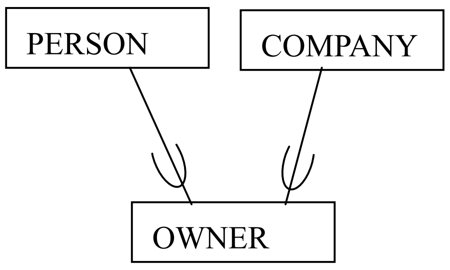
- A subclass may itself have further subclasses, forming a **hierarchy**.
- In case of **multiple inheritance**, specialization **lattices** may be formed.
- See Figure 4.6 for a lattice, in which a subclass (ENGINEERING_MANAGER) can be a subclass in more than one class/subclass relationship.
- In a specialization lattice or hierarchy, a subclass inherits all attributes of superclasses all the way to the root.（多重繼承：不只繼承直接 superclass 的屬性及關係，且繼承所有其上的 superclass 的屬性及關係，直到根節點，其中若有繼承到重複 superclass 的屬性及關係時，只繼承一次）
- See Figure 4.7. GRADUATE_STUDENT 繼承 STUDENT 及 PERSON 的屬性，共有七個屬性，RESEARCH_ASSISTANT 繼承 STUDENT_ASSISTANT, STUDENT, EMPLOYEE 及 PERSON 的屬性，共有九個屬性。
- A subclass with more than one superclass is called a **shared subclass** (例如: STUDENT_ASSISTANT). If no shared subclasses existed, we would have a hierarchy rather than a lattice.

（四）Categories

- In the subclass/superclass relationships, a subclass either has a single superclass or has multiple superclasses with *similar* properties.
- Sometimes, there is a need for grouping a subset of entities from *different* entity types. See Figure 4.8. (The OWNER is a subclass of the UNION of the three entity sets of COMPANY, BANK, and PERSON)
- A **category** is a subset of **the *union* of its superclasses**, whereas a **multiple inheritance** is a subset of the ***intersection* of its superclasses**.
- 在 EERD 中，以一個圓圈圈住 "∪" 符號，並以弧線連接 category 及其 superclasses 來表示其間之群聯集運算 (set union operation)。
- 圖解 category：

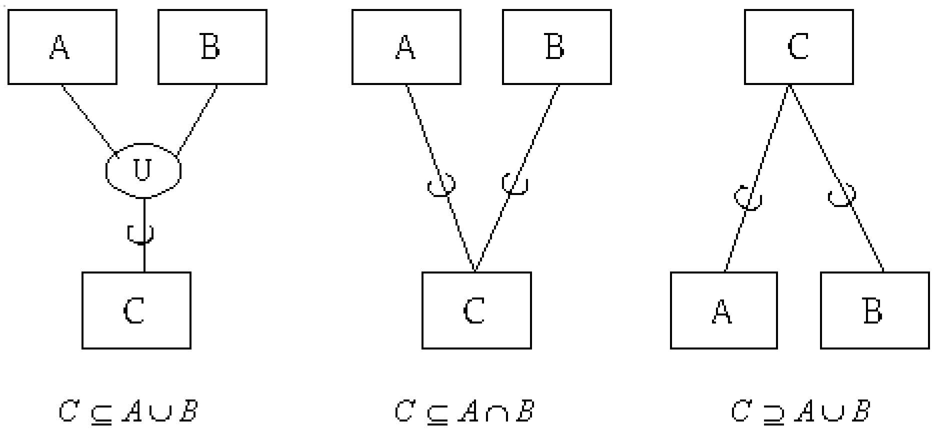
(a)hierarchy inheritance (b)multiple inheritance





(a) OWNER ⊇ (PERSON ∪ COMPANY)，(b)是 OWNER ⊆ (PERSON ∩ COMPANY)，兩者皆不能充分表示 category 的含意，應改 category 來表示 OWNER ⊆ (PERSON ∪ COMPANY) 的關係。

- The attributes of a category is the **union** of the attributes of its superclasses. However, in contrast to the multiple inheritance, no entity in a category has values for all attributes.
- Compare categorization with multiple inheritance and specialization. See the following figure:



- A category can be total or partial. See Figure 4.9.
- 假如 category 是 total 的話，就表示也可以用 specialization (如上圖（a）)，

若兩 class 具有同型態並共用很多屬性，且有相同的主要屬性，就選 specialization，否則選 categorization 較適當。

- .See Figure 4.10 for a complete example.

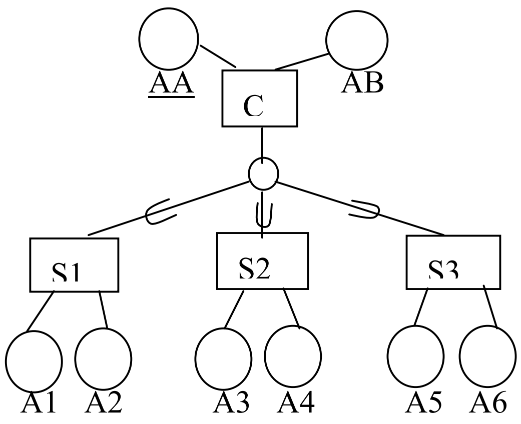
（五）Conceptual Object Modeling using UML class diagrams

- UML (Universal Modeling Language) defines several diagrams that can be used in the process of object –oriented software design.
- Among the UML diagrams, class diagrams are similar to EER diagrams in many ways. Class diagrams intend to capture the object classes and their relationships in the context of an application. Therefore, database classes, or called persistent classes (永久類別；基本類別), can also be described by EER diagrams.
- Correspondence between EER diagrams and UML class diagrams.
  - **Entity** vs. **Object**.
  - **Entity Type** vs. **Class**.
  - Each class comprises three parts：class name, attributes, and operations. (attributes 可附帶指定值域，而在 EER 內沒有 operations).
  - A composite attribute is modeled as a structured domain. See the Name attribute in EMPLOYEE class in Figure 4.11.
  - A multivalued attribute is generally modeled as a separate class. See the LOCATION class in Figure 4.11.
  - Relationship types are called *associations* in UML, and relationship instanes are called *links*.（In UML, a relationship attribute is called a *link attribute*）
  - An association may or may not have a name.
    - (min, max) notation is used to specify relationship constraints (are called **multiplicity** in UML), though the placement of multiplicity is opposite compared to ERD. (參考課本圖 3.15 與圖 4.11 中的 WORKS_FOR 關係的參與情形，兩者之(min, max)置於對調位 置，在 UML 中，用一個"＊"表示"0..＊"，一個"1"表示"1..1"，"＊"表示沒上限)
    - UML has two types of relationships： association and aggregation.
      - **Aggregation** is meant to represent a relationship between a whole object and its component parts. However, with respect to when to choose aggregation or association is not clear. (參考課本圖 4.11 中右下角的圖示)
      - Weak entities can be modelled as a **qualified association**(or qualified aggregation). See Figure 4.11
      - Dependent Name (partial key) under EMPLOYEE, is placed in a box attach to the owner class.
  - UML uses blank triangulation to indicate a disjoint specialization and filled triangulation for an overlapped specialization. See Figure 4.12.

（六）EER to Relational Mapping

- **Specialization Mapping**：{S1, S2, ..., Sn} is a specialization of C. Let k be the primary key of C. There are four approaches for mapping.
  1. A relation for C with primary key being {k} and attributes being C’s attributes. A relation for each Si with primary key being {k} and attributes being S’s attributes. (將 C 的全部屬性轉成一個關聯綱目，以 k 爲主要鍵，然後對每個 subclass，就其個別屬性加上 k 轉成個別的關聯綱目，且都以 k 爲其個別的主要鍵，參考課本圖 4.4 及圖 9.2.a)
  - ■ **This option works for any constraints on the specialization: disjoint or overlapping, total or partial.**
  2. A relation or each Si with primary key being {k} and attributes being S’s attributes union C’s attributes. (將每個 subclass，就其個別屬性加上 C 的全部屬性轉成個別的關聯綱目，且都以 k 爲其個別的主要鍵，參考課本圖 4.3 及圖 9.2.b)
  - ■ **This option is suitable only for both the disjoint and total constraints, and is not good for overlap constraint**.
  3. A single relation with key being {k} and attributes being the union of C and all Si, and a type attribute. The type attribute is used to indicate to which subclass a tuple belong. (將 C 及每個 subclass 的全部屬性，再加上辨別 tuple 所屬 subclass 的型態屬性，轉成一個關聯綱目，以 k 爲其主要鍵，參考課本圖 4.4 及圖 9.2.c，型態屬性爲 JobType，JobType 本來就是 C 的一個屬性，所以就不必再另加型態屬性)
  - ■ **This option can be used only for disjoint constraint**.
  4. A single relation with key being {k} and attributes being the union of C and all Si, and several type attributes, one for each subclass. A type attribute is used to indicate whether the tuple belongs to the associated subclass. (將 C 及每個 subclass 的全部屬性，再加上用來辨別 tuple 所屬個別 subclass 的所有個別型態屬性，轉成一個關聯綱目，以 k 爲其主要鍵，請參考課本圖 4.5 及圖 9.2.d，其型態屬性分別爲布林屬性的 MFlag 和 PFlag)
  - ■ **This option is for a specialization whose subclasses are overlapping, but will also work for a disjoint specialization**.

- 簡例:：



依上述第 1.項之四種方法的關聯綱目如下：

方法 1 之關聯綱目：

C{<u>AA</u>, AB}
S1{<u>AA</u>, A1, A2}
S2{<u>AA</u>, A3, A4}
S3{<u>AA</u>, A5, A6}

方法 2 之關聯綱目：

S1{<u>AA</u>, AB, A1, A2}
S2{<u>AA</u>, AB, A3, A4}
S3{<u>AA</u>, AB, A5, A6}

方法 3 之關聯綱目：

C{<u>AA</u>, AB, A1, A2, A3, A4, A5, A6, At}
At 爲表示 subclass type 的 type attribute.

方法 4 之關聯綱目：

C{<u>AA</u>, AB, A1, A2, A3, A4, A5, A6, At1, At2, At3}
At1, At2, At3 爲表示 subclass type 的 type attribute, 但是，是 Boolean attribute。
假設一 tuple 爲{<u>AA</u>, …, …, …, …, …, …, …, 1, 0, 1}，則表同屬 S1 和 S3。

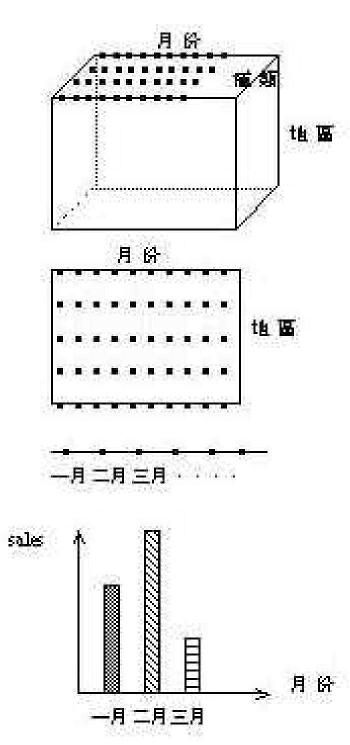
在上述方法 1 和 2 都產生多關聯(multiple relation)，而方法 3 和 4 則產生單一關聯(single relation)，在此種單一關聯中有些 tuples 的屬性會因其不屬於某些 subclass，而出現某些屬性是空値的情形。

- **Category Mapping**
  - A relation is created for a category. It is customary to specify **a new key attribute**, called **surrogate key**. （See Figure 9.4. and Figure 4.8）
  - 由不同型態的 entity type 所構成的 category 必須新定義一個 surrogate key 當作其主要鍵，然後每個 superclass 的關聯綱目都要加上此 surrogate key 當作其 foreign key（例如：PERSON, BANK, COMPANY, 及 OWNER）。但是，若 category 的 superclass 具有相同的主要鍵時，就不需要新定義一個 surrogate key 了(例如：REGISTERED_VEHICLE, CAR, TRUCK, 及 OWNS)。

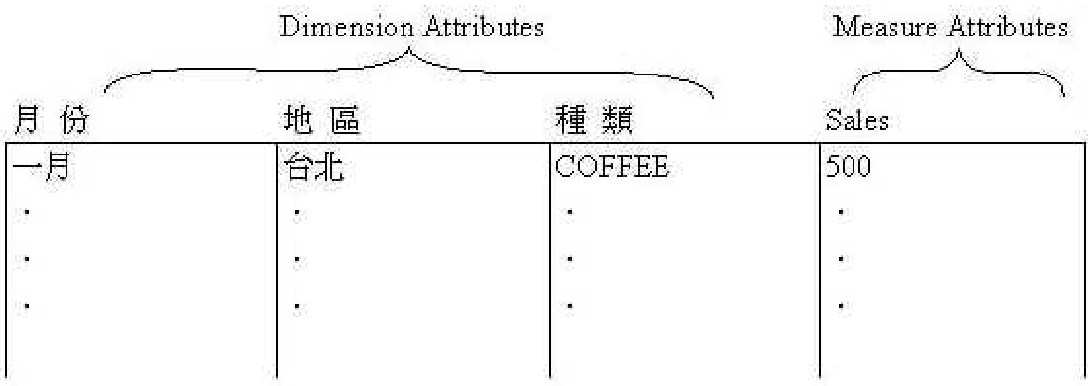
八、OLAP (supplement and Chapter 26)

（一）Introduction

- The following materials are extracted from “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals,” Data Mining and Knowledge Discovery 1, pp. 29-53, 1997.
- **OLAP** (One-Line Analytical Processing) is mainly used for data analysis applications.
- Data analysis is an iterative procedure that involves four steps（資料分析的過程）:
  - **Formulating** a query for extracting relevant data.（從一個大資料庫抓出想要的資料）
  - **Extracting** the aggregated data into a file or table.（彙總存成一個檔案或表單）
  - **Visualizing** the result in a graphical way.（將結果以圖形化方式表示出來）
  - **Analyzing** the result and formulating a new query.（分析結果後再從資料庫抓出其他想要的資料）
- MS Excel is a visualization/analysis tool.
- Data analysis tools view the dataset as an N-dimensional space. See the following figure:



- Relational systems model N-dimensional data as a relation with N-attribute domains, where some attributes are viewed as dimensions and others represents measurements.
- See Table 1 in Page 31, Time(UCT), Latitude, Longitude, and altitude are dimensions. Temp. and Pres. are measurements.
- The following figure shows dimensions and measures:


Dimension Attributes
Measure Attributes

月 份	地 區	種 類	Sales
一月	台北	COFFEE	500
.	.	.	.
.	.	.	.
.	.	.	.

- 目前資料庫的設計方法主要有兩種：
  - Multi-dimensional OLAP (簡稱 MOLAP).
  - Relational.OLAP (簡稱 ROLAP)
- 如果用 ROLAP 方式，則上例資料可存成 SALES{月份，地區，種類，銷售額}所構成的表單中，假設只考慮月份(壓縮地區及種類)，則以 SQL 可寫成如下，以得到右上圖表所需資料：

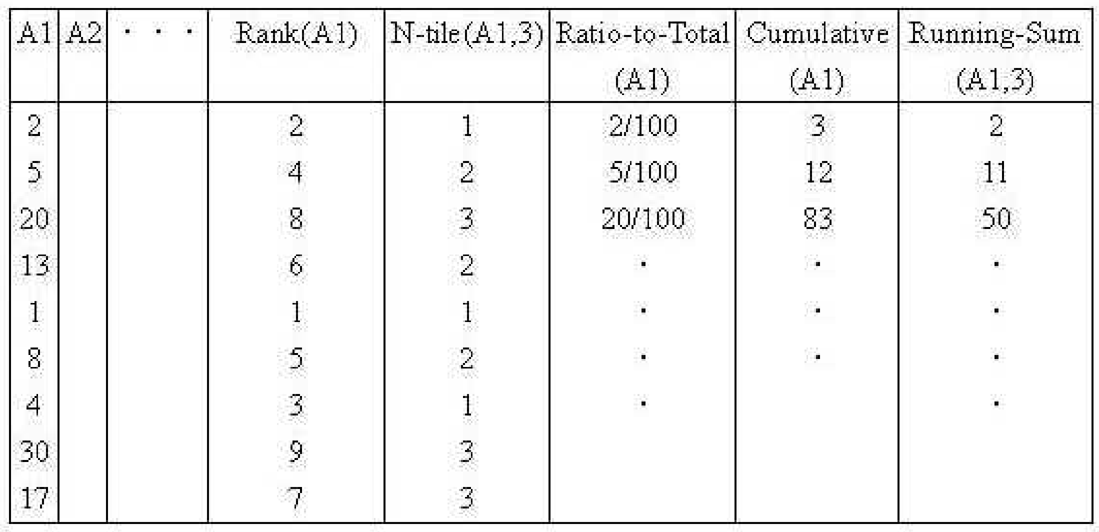
  SELECT 月份，**SUM**(銷售額)
  FROM SALES
  GROUP BY 月份;
- Data analysis tools extensively use dimensionality reduction (aggregation) for better comprehensibility. The relational systems **relies on aggregate functions and GROUP BY operator to support aggregation**. For example, the following query reports the average temperature for each reporting time and altitude：

  SELECT Time, Altitude, AVG(Temp)
  FROM Weather
  GROUP BY Time, Altitude;
- SQL's aggregation functions are very popular（愈來愈重要）. TPC-D query set has one **6D GROUP BY** and three **3D GROUP BYs**. See Table 2 for the presence of aggregate functions in SQL.
- In addition to COUNT, SUM, AVG, MIN, MAX, many systems provide extensions to aggregate functions such as median, standard deviation, center of mass, angular momentum, etc.
- Some systems allow users to add new aggregate functions in a way such as the following：(Informix Illustra system)
  1. Init (&handle): Allocates the handle and initializes the aggregate computation.（e.g., SUM = 0; COUNT=0）
  2. Iter (&handle, value): Aggregates the next value into the current

aggregate.（e.g., SUM = SUM + Sale1; COUNT=COUNT+1）
3. Value = Final (&handle): Computes and returns the resulting aggregate by using data saved in the handle.（e.g., Value = SUM/COUNT）

- Red Brick system, an UNIX OLAP vendor, add the following aggregate functions：
  - ■ Rank(attribute): Returns the rank of a tuple according to its value on “attribute”.
  - ■ N-tile(attribute, N): Divide the tupes into N categories according to their values on “attribute”. This function returns a tuple’s category rank (1…N).
    Ratio_To_Total(attribute): Return the ratio of the attribute value to the sum of attribute values of all tuples.
  - ■ Cumulative(attribute): Return the sum of all attribute values so far in an ordered list.（小於且等於某值之和）
  - ■ Running_Sum(attribute, n): Return the sum of the most recent n values in an ordered list.（依序最接近於某值的 n 個值之和）
  - ■ Running_Average(attribute, n): Return the average of the most recent n values in an ordered list.（依序最接近於某值的 n 個值之和的平均）
-

舉例：



A1	A2	· · ·	Rank(A1)	N-tile(A1,3)	Ratio-to-Total (A1)	Cumulative (A1)	Running-Sum (A1,3)
2			2	1	2/100	3	2
5			4	2	5/100	12	11
20			8	3	20/100	83	50
13			6	2	·	·	·
1			1	1	·	·	·
8			5	2	·	·	·
4			3	1	·		·
30			9	3			
17			7	3			

（二）Problems with GROUP BY

There are three main problems：(1) histograms, (2) roll-up totals and sub-totals for drill-downs, (3) cross tabulation.

Histograms

- A histogram is a figure that shows the historical change of something.
- For example, one may like to see the highest temperature of nations across days. The following query serves this purpose:

```
SELECT    day, nation, MAX(Temp)
FROM      Weather
```

```
GROUP BY   Day(Time) AS day,
           Nation(Latitude, Longitude) AS nation;
```

- However, in current standard SQL(SQL92), the above query is not valid and has to be modified to the following nested query：

```
SELECT      day, nation, MAX(Temp)
FROM        (SELECT     Day(Time) AS day,
                        Nation(Latitude, Longitude) AS nation,
                        Temp
             FROM       Weather
            ) AS foo
GROUP BY    day, nation;
```

Roll-up Totals/subtotals for drill-down

- Reports commonly aggregate data at a coarse level, and then at successively finer levels. (Roll-up：資料之加總，例如月資料加總成季或年的資料；Drill-down：資料之細分，例如年資料細分成季資料，一般在作這兩種運算時 dimension 會有一個順序)
- See Table 3a, in page 35 for data representation. The representation of Table 3a is not relational because the empty cells cannot form a key. 如把它改爲 Table 3b 的表示方式雖好，但有許多欄位的資料重複。
- .Note that Table 4, though clear, introduce too many columns. For example, when one pivots on two columns with M and N values, the resulting pivot table has NM values.
- Another simple rollup can be seen in Table 5a. Sales summary.（如下表）

- Table 5a can be formed by the complicated SQL statement shown below.（上表是一個關聯，可以如下 SQL 敘述建立，但此種方法執行效率不佳，若是 N 個 dimension 則需 N 個 UNION）

```
(SELECT      'ALL', 'ALL', 'ALL', SUM(Sales)
        FROM    Sales
        WHERE Model = 'Chevy')
 UNION
 (SELECT            Model, 'ALL', 'ALL', SUM(Sales)
        FROM        Sales
        WHERE       Model = 'Chevy'
        GROUP BY    Model)
 UNION
(SELECT          Model, Year, 'ALL', SUM(Sales)
        FROM        Sales
        WHERE       Model = 'Chevy'
        GROUP BY    Model, Year)
 UNION
(SELECT          Model, Year, Color, SUM(Sales)
        FROM        Sales
        WHERE       Model = 'Chevy'
        GROUP BY    Model, Year, Color)
```

以上 query 的結果來說，尚缺以 Color 做彙總的資料，如下表所示：

Model	Year	Color	Units
Chevy	ALL	Black	135
Chevy	ALL	White	155

於第 5 項之 SQL 敘述加上下列敘述即可增加建立上表 5b 之兩列資料：

```
UNION
(SELECT          Model, 'ALL', Color, SUM(Sales)
    FROM      Sales
    WHERE     Model = 'Chevy'
    GROUP BY  Model, Color)
```

- 如資料量很大，以上 query 的執行速度會很慢。

Cross-Tabulation (Cross tab)

- Rollup is asymmetric. The symmetric aggregation result is a table called cross-tabulation.
- A cross tabulation imposes no order on dimensions, e.g.

Ford	1994	1995	Total
Black	50	85	135
While	10	75	85
Total	60	160	220

- 使用舊的 SQL 表示 roll-up 及 cross-tab 是不適當的，以一個 6 維 cross tabulation 來說，就要以 64 次的 UNION 配合 64 個不同的 GROUP BY 運算，才能得到完整資料表。也就是說需掃描 64 次資料，並做 64 次排序，這實在太花費時間了，而且所得資料也太複雜而難以分析。
- Since rollup and cross-tab are so important, it’s better to propose new SQL operators, which enable efficient query optimization.

（三）CUBE and ROLLUP operators

- To enable efficient OLAP applications, two new operators: CUBE and ROLLUP are proposed to conduct cross tabulation and roll up respectively.
- With **CUBE** operator, the following query can be formed：

```
SELECT  day, nation, MAX(Temp)
FROM    Weather
GROUP BY CUBE
            Day(Time) AS day,
            Country(Latitude, Longitude) AS nation;
```

- If the cardinality of N attributes are C_1, C_2, …, C_n, then the cardinality of the resulting cube relation is $(C1+1)(C_2+1)\ldots(C_n+1)$.

例：CUBE

Model (5)	Year (3)	Color (3)
	ALL	
ALL		
		ALL

此 relation 會有(5+1)*(3+1)*(3+1)筆資料。

- For rollup and drill-down report, another operator ROLLUP is provided. ROLLUP suites the applications with linear functional dependencies on attributes, while CUBE is used for applications with independent attributes. For example, one may want to rollup on week, month, and year.
- ROLLUP 與 CUBE 之比較：
  以上頁第 1 項之 SQL 敘述爲例，使用 **GROUP BY ROLLUP** 及 **GROUP BY CUBE** 之不同

Day	Nation	MAX(Temp)	
ALL	•	•	
ALL	•	•	使用 CUBE 產生的資料
ALL	•	•	使用 CUBE 產生的資料
•	•	•	
•	•	•	
•	•	ALL	
•	•	•	
•	ALL	ALL	
•	•	•	
ALL	ALL	ALL	
ALL	ALL	•	使用 CUBE 產生的資料
•	ALL	•	

" • "表示有資料。

- Cumulative aggregates, like running sum or running average, work especially well with ROLLUP because the answer set is naturally sequential (linear) while the *full data cube* is naturally non-liner (multi-dimensional). ROLLUP and CUBE must be ordered for cumulative operators to apply.

Combining CUBE and ROLLUP

- **Syntax**：
  GROUP BY [<aggregation list>]
  [**ROLLUP** <aggregation list>]
  [**CUBE** <aggregation list>]
- E.g., See page 41 Figure 5. for the result of the following query.
  SELECT Manufacturer, Year, Month, Day, Color, Model, **SUM**(price) **AS** Revenue

```
FROM        Sales
GROUP BY    Manufacturer
  ROLLUP    Year(Time) AS Year
            Month(Time) AS Month,
            Day(Time) AS Day,
     CUBE   Color, Model;
```

Decoration Attributes

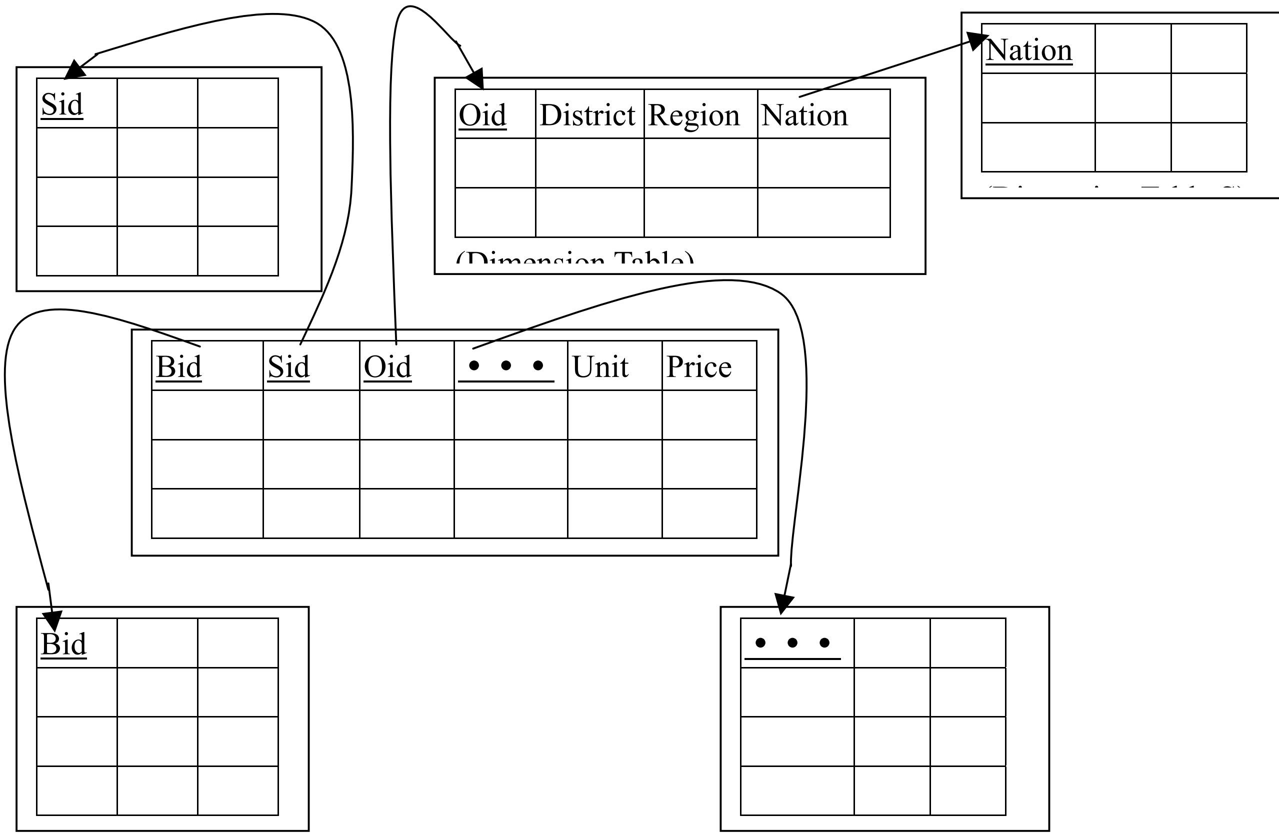
- A decoration attribute（裝飾屬性）in a query is an attribute that is functionally dependent on the aggregate columns.例如下列查詢程式中 DNAME 即爲裝飾屬性，DNAME 沒有出現在 GROUP BY 子句內：

  ```
  SELECT  DNUMBER, DNAME, COUNT(＊)
  FROM        EMPLOYEE
  GROUP BY    DNUMBER;
  ```

  在目前的 SQL 是不容許上面這種寫法，但本篇報告是主張應容許此種情形，也就是說假如一裝飾屬性(或屬性值)函數相依於彙總屬性，則它就可以包括在 SELECT 子句的屬性列內。
- 如果彙總 tuple 有函數定義了裝飾屬性的值，則在彙總結果 tuple 的該裝飾屬性就會有值，否則爲空值。

Snowflake/Star Schema

1. There are two kinds of tables：a fact table and a set of dimension tables.



Bid, Sid, Oid,..等爲 dimension，Unit, Price 爲 measure。
Fact Table 包含 dimensions (refer to dimension tables) and measures。

- 上圖是爲一個 **snowflake schema**，若無右上角的 Dimension Table S 則爲 **star schema**。也就是說，從 Fact Table 的任一 dimension 各只產生一個 dimension table 則爲 **star schema**，若一 dimension 產生兩個以上的 dimension table 或由 dimension table 又產生 table，則爲 **snowflake schema**。
- One can apply CUBE on the fact table and ROLLUP on some dimension tables.
- Note that the granularities of a dimension may form a lattice (e.g., date -> (week, month) -> year).

Data Warehouse

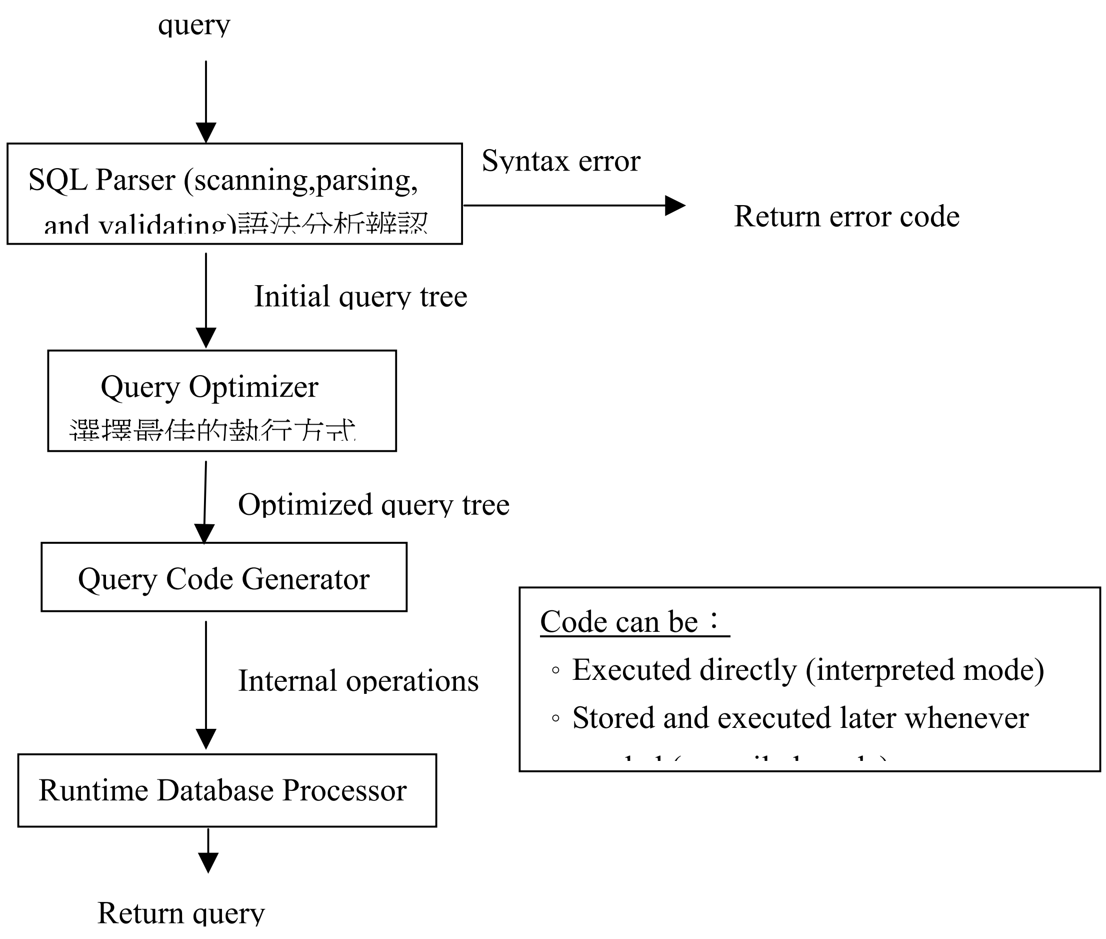
- Data warehousing（資料倉儲）can be described as “a collection of decision support technologies, aimed at enabling the knowledge worker to make better and faster decisions.（資料倉儲通常是一直累積資料，很少作更新，要把資料庫資料儲存到資料倉儲時須先做清理及改變格式，存成多維模式）
- See the overall data warehousing process in Figure 26.1 of the textbook.
- Data warehouses are generally an order of magnitude larger than the source databases, because it contains databases across time（含歷史資料）and departments. The sheer volume of data is likely to be in terabytes.

九、Query Processing and Optimization (Chapter 18 and 6)

（一）Conception

Query Processing Procedure

Typical steps when processing a high-level query（此爲 DBMS 執行的部分）:



Outline

- Basic Algorithms for Executing Query Operations (SELECT, PROJECT, JOIN, and SET operations).
- Query Optimization（查詢最佳化有兩種方式）:
  - Heuristics：經驗法則，此方法並非是最好的。
  - Cost Estimate：有系統地算出最經濟的方法執行。
- 通常在選擇查詢的執行方式時，並不是選最佳的，而是選合理有效率的方式執行。但是對特別設計的資料庫做特定的查詢時，則可以最佳化的查詢模式執行。

（二）Index Structure

- For details, see Chapter 6.（根據 Index 來搜尋，速度較快）
- There are basically two types of data structures for index, namely **tree** and **hash table**.

Tree Based Index Structure

- **Each node is a page in disk.** Thus, a node can contain multiple **<key, pointer>** pairs.
- It had better be balanced.
- Space utilization must be high, even in the presence of many deletions.
- Two commonly used tree data structures are B-tree and B+ tree. Nowadays, most DBMSs use B+ tree to implement indexes.
- For details on B-tree, please refer to the textbook.

  參考課本 173 頁 Figure 6.10，

  B-tree 結點的結構包含三種資料值：

  - Tree pointer: 指到 B-tree 其他結點的指標值（磁碟位址）。
  - Data pointer: 指到資料檔區塊的指標值（磁碟位址）。
  - search key field value: 搜尋鍵值。

B+ Tree

- The structure of B+ Tree is shown in Figure 6.11 in page 176. Also see the following figure

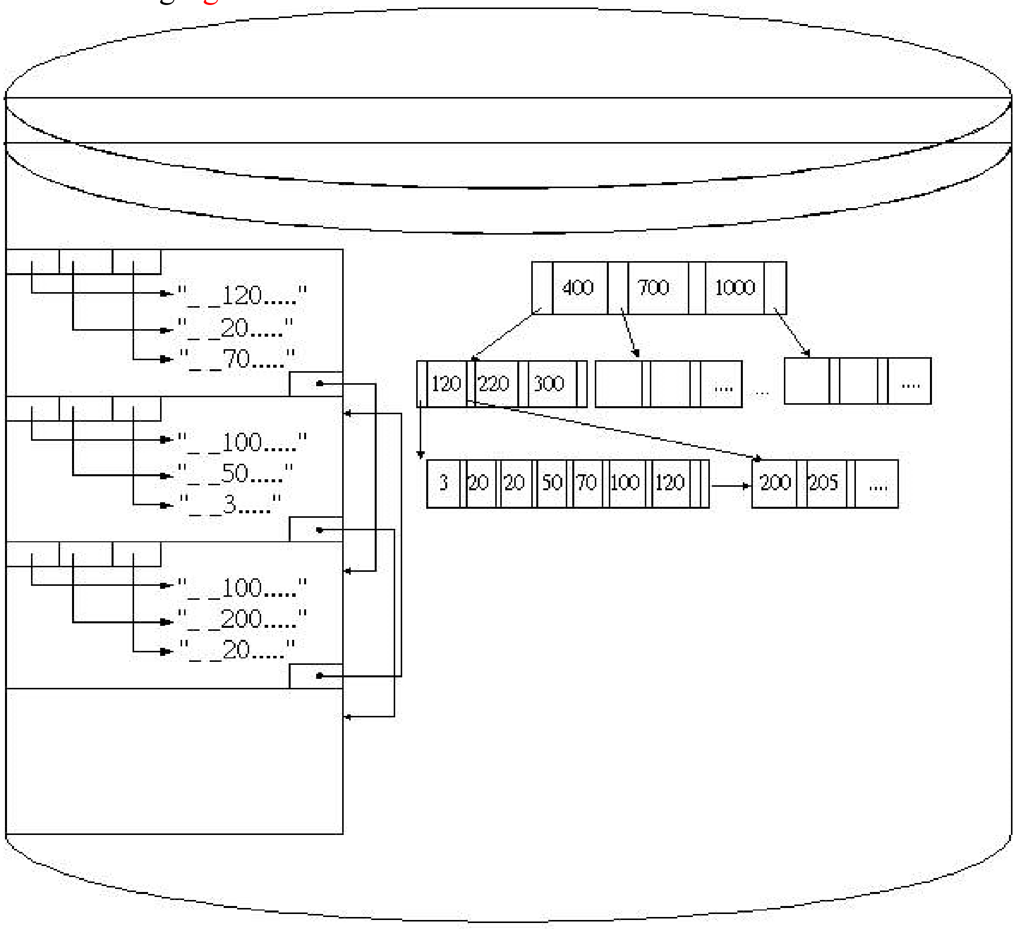


Figure 2

- B+-tree (order p) 的結構包含兩種結點：
  - Internal node: 包含二種資料值 <P1,K1,P2,K2,…,Pq-1,Kq-1>；q ≦ p
    - Tree pointer(Pi): 指到 B+-tree 其他 Internal node 或最底層 Leaf node 的指標值（磁碟位址）。
    - search key field value(Ki): 搜尋鍵值（Ki-1 < Ki）。
  - Leaf node: 包含三種資料值 <<K1,Pr1>, <K2,Pr2>,…,<Kq-1,Prq-1>, Pnext>；q ≦ p
    - Data pointer(Record pointer；Pri): 指到資料檔區塊的指標值（磁碟位址）。
    - search key field value(Ki): 搜尋鍵值（Ki-1 < Ki）。
    - Next pointer(Pnext): 指到相同 B+-tree 的下一個（右邊）Leaf node 的指標值（磁碟位址）。
- 對同一組搜尋鍵值來說，B+-tree 的中間結點(Internal nodes)比 B-tree 的結點(Tree node)可存較多值，因此用 B+-tree 所構成的樹就較為扁平，加快搜尋速度。
- Each tree pointer is a disk address (or page number).
- Each data pointer is in the form of (page #, offset).
- Let **p** be the maximum tree pointers in an internal node. Each internal node, except the root, has at least **p/2** tree pointers.
- Let **q** be the maximum data pointers in a leaf node. Each leaf node has at least **q/2** data pointers.
- All leaf nodes are at the same level.
- All leaf nodes are chained together according to the index key values.
- It can handle both equality and range search.
- See Example 6 and 7 in page 177 to get a feeling on the number of disk access a B+ tree search needs.

假設： 一個 B+ tree 的每一 Internal node 可容納 p 個 tree pointers 及 p-1 個搜尋鍵值

search key 長度 V = 9 bytes， 磁碟區塊大小(block size) B = 512 bytes
data pointer 長度 Pr = 7 bytes， tree pointer 長度 P = 6 bytes

則 $(p*P) + ((p-1)*V) \leq B$ 即為 $(p*6) + ((p-1)*9) \leq 512$

可算出每一中間結點至多可有 p=34 個 children。

假設每一 leaf node 可容納 p_{leaf} 個 data pointers 和 p_{leaf} 個 search keys, 再加上一個 tree pointer 指到下一個鄰接的 leaf node，則 $(P_{leaf}*(Pr + V)) + P \leq B$

即為 $(P_{leaf}*(7 + 9)) + 6 \leq 512$

可算出：資料指標(記錄指標) Pleaf 最多會有 31 個。

另參考課本例 7 計算當每個結點存滿率為 69%時，每一 internal node 有 34*69%=23 children, 每一 leaf node 可有 31*69%=21 data pointers，因此對一個四層的 B+ tree, 其儲存量為

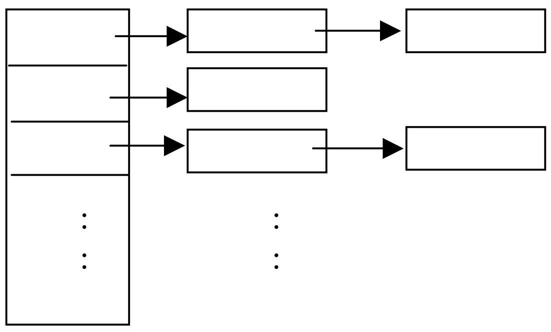
Root: 1 node	22 entries	23 pointers
Level 1: 23 nodes	506 entries	529 pointers
Level 2: 529 nodes	11638 entries	12167 pointers
Leaf level: 12167 nodes	255507 record pointers	

- For details on insertion and deletion to a B+ tree, please refer to some Data Structure book.
- 在索引之最底層(leaf)，若排成查詢所需的連續排列，效率是會最好的，例如：

  CREATE INDEX
  ON R1(A1 ASC; A2 DESC)
  產生一排序資料：(1, 100),(1, 70),(1, 20),…,(3, 120),(3, 50),(3, 20),…,(5, 90),……則下列 SQL 敘述
  SELECT ＊
  FROM R1
  WHERE A1 = 3 AND A2 < 70; 可找到(3, 50),(3, 20)，剛好相鄰，速度快，效率好。
  若把 WHERE 的條件改為 A1 > 0 AND A2 = 20 時，則可找到(1, 20),(3, 20)，兩者相隔較遠，速度會較慢，若又在不同頁的話，將會更慢。故選擇索引時，最好根據需要設定較適當的索引。

Hashing Table

- The hashing (雜湊) data structure is as follows:



- Again, each block is in the unit of disk pages. Besides, the hashing table is also in the unit of disk pages.
- Hashing based index can only handle equality search, range queries are not supported.
- This data structure is beneficial only when the number of records is huge and the index attributes is not ordered.（參考課本 139 頁 5.9 節 Hashing Techniques）

（三）Basic Algorithms for Executing Query Operations

Implementing SELECT Operation

- Examples：搜尋檔案，找出合乎條件的記錄。

$\sigma_{\text{ssn='123456789'}}$ (EMPLOYEE)	(Equality selection)
$\sigma_{\text{DNUMBER>5}}$ (DEPARTMENT)	(Range selection)
$\sigma_{\text{DNO=5 AND SALARY>30000}}$ (EMPLOYEE)	(Conjunctive selection)

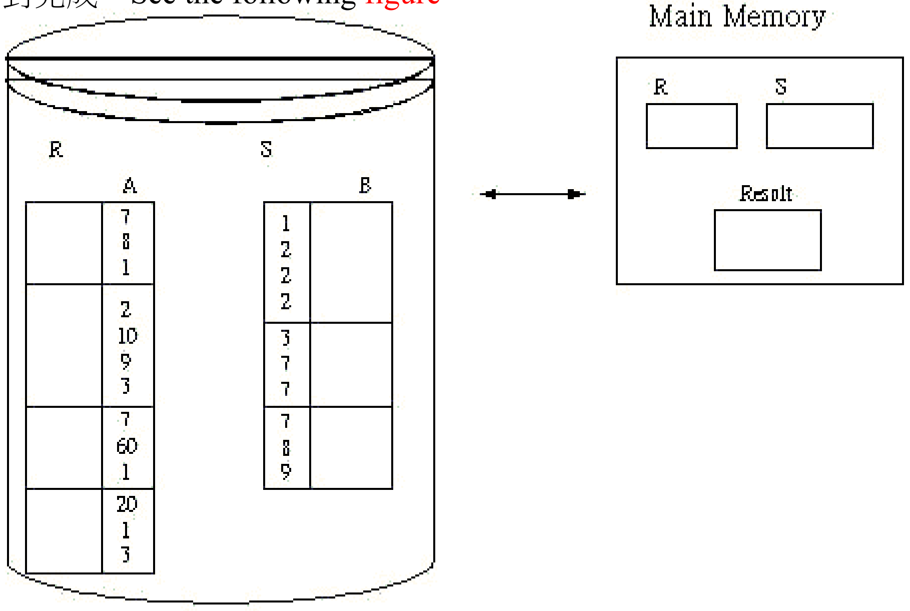
- Simple Selection：
  - (S1) Linear Search (brute force)：逐筆記錄核對，滿足條件的記錄挑出來（循序）。

- (S6) Using a secondary index (B+ tree)：在等號(=)條件的搜尋下，如索引欄是 key 的話，只會找到一筆記錄，否則會找到多筆記錄，也可使用到 ＞, >=, < , <= 條件的搜尋。

- Conjunctive Selection：
  - (S7) Using an index on a simple condition：先用索引找出合乎其中之一種條件的記錄，再拿此找出之記錄去核對是否滿足其他條件，最後剩餘記錄即是結果。
  - (S8) Using a composite index：用兩個或兩個以上的屬性做索引，所有皆相等的記錄挑出來即是，例如 B+ tree 一節裡最後的例子。
  - (S9) Using multiple indexes and do an intersection on the retrieved record pointers：依個別條件的索引，找出合乎個別條件者，再做交集，即得到結果。索引若包含記錄指標，則依此記錄指標做交集運算，所得指標所指記錄即爲合乎條件者。
- Query optimization for a SELECT operation is needed mostly for conjunctive selection.（當有多個屬性包含在選擇條件內時，SELECT 運算就必須做最佳化。）
- “**selectivity**” is an important factor to consider in choosing between multiple execution plans. For example, the estimated selectivity for an equality selection on a relation R is **1/|R| if the selection condition is on a key attribute**, and **1/i otherwise**, where i is the number of distinct values of the attribute.
  選擇性(s) = 合乎條件的記錄數／檔案記錄總數。(0 <= 選擇性 <= 1)
  選較小的 s 值的條件先做 SELECT，就比較理想有效率。
- Disjunctive selection is more difficult to optimize.
  例如：$\sigma_{\text{DNO=5 OR SALARY>30000 OR SEX = 'F'}}$ (EMPLOYEE) (各條件之間是以 OR 連接起來，是個別條件的聯集)

Implementing the JOIN Operation

- **JOIN** is one of the most time-consuming（最花時間的）operations in query processing.
- The following only deal with **EQUIJOIN**.（在查詢中較常用的是 EQUIJOIN 及 NATURAL JOIN。）
- Methods for Implementing **JOIN** R $\bowtie_{A=B}$ S（A, B 各爲 R, S 的屬性）
- (J1) Nested Loop join (Brute Force)：從 R 抓出第一筆記錄與 S 的每一筆記錄比對，合乎條件者挑出，再抓 R 的第二筆繼續比對，直到 R 全部比對完成。See the following figure



- (J2) Single-loop join：If there is an index on the attribute B of S, we can sequentially retrieve each tuple in R, and then use the value of A to search the index.（假定 S 的 B 建有索引，則循 R 的記錄順序以其 A 值在 S 中依索引迅速找出配對 B 值的記錄。）
- (J3) Sort-Merge join：If R and S are sorted (clustered) in the disk according to attribute A and B respectively, a subsequent merge can be conducted by sequentially scanning both relations.（R 及 S 兩者皆各依 A 及 B 排序，則依序抓出 A=B 者配對。若是有索引包含記錄指標者，則依此記錄指標做交集運算，所得指標所指記錄即爲合乎條件者。抓資料的次數爲 R 的頁數加上 S 的頁數。）See the following figure.

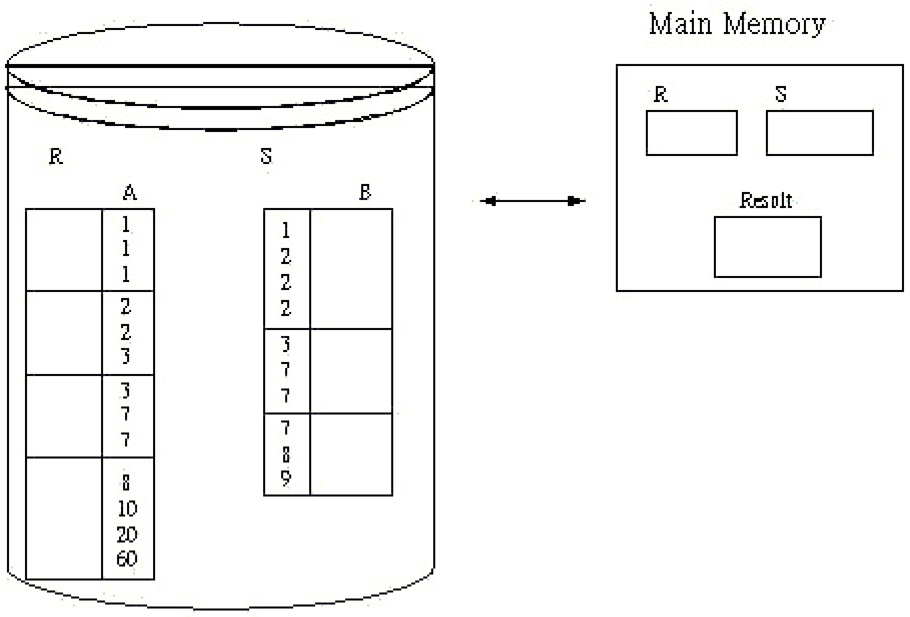


Figure 7

- (J4) Hash join: The tuples of one relation, say R, are hashed to the buckets of a hashing structure by using the values of attribute A. The tuples of S are then hashed to the appropriate buckets for matching by using the values of attribute B.（先將 R 依屬性 A 的值用雜湊法把記錄分到各 bucket 內，再將 S 依屬性 B 的值用雜湊法把記錄分到各適當的 bucket 內，最後再將各 bucket 內有 R 及 S 者配對，即爲合乎條件者。）See the following figure

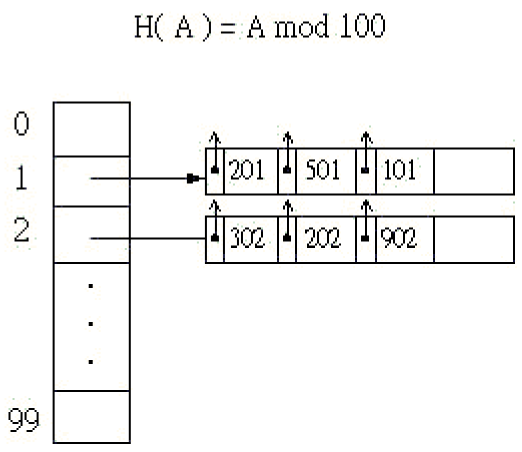


Figure 3

Implementing the PROJECT operations

- It is straightforward if no duplication is involved. (How do you know if the projected results have no duplication? 假如 PROJECT 出來的屬性列內含有 key attribute 的話，就不會有重複的記錄)
- For duplication elimination, sorting or hashing can be used.（在 SQL 內若要消除重複，則先排序再 DISTINCT。）

The Implementation of Set Operations

- Sorting can be used for ∩and∪.
- Hashing can be used for ∩and －.
- CARTESIAN PRODUCT（×）會產生 N*M 個記錄及 j+k 個屬性，這會佔用很大記憶體空間，並浪費很多時間，應盡量不用，改用其他運算法代替。
- 使用∩，∪，×時，相關 Table 須具有完全相同的屬性，且其個別屬性的值域也要相同。

Implementing Aggregate Operations

- .Aggregate operators can be computed by a table scan or an appropriate index.
- Consider the following example

```
SELECT      MAX (SALARY)
FROM        EMPLOYEE;
```

An index on EMPLOYEE (SALARY) can be naturally used to compute it without scanning the table EMPLOYEE.（若找 MAX 或 MIN 則由根結點往最右或最左以抓到所有葉結點，再由將所有索引鍵值彙總即可）

- Other aggregation operators like COUNT, AVERAGE, and SUM can be computed simply from index trees without resorting to the actual data records.
- When a GROUP BY clause is used in a query, for example：

```
SELECT      DNO, AVG (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO;
```

The data records are partitioned into several groups by grouping attribute via either sorting or hashing. Computation is then followed for the final result.（依 GROUP BY 屬性，將等值者分成同組，再就各組做彙總運算，得到結果。）

前述 SQL 敘述的作法：

依據 DNO 排序，再依 DNO 順序分組求出各組 AVG (SALARY)。

CREATE INDEX ON EMPLOYEE(DNO, SALARY)，再依索引順序求出各組 AVG (SALARY)。

Implementing Outer Join

- We can slightly modify the join algorithms to compute (left, right, full) outer join.

```
SELECT      LNAME, FNAME, DNAME
FROM        (EMPLOYEE LEFT OUTER JOIN DEPARTMENT ON
DNO = DNUMBER);
```

- For example, to compute a left outer join, we can modify the nested loop algorithm by using the left relation as the outer loop. However, for a given tuple in the left relation, if no matching tuple is found, the tuple is still included in the result but is padded with null values.（左關聯的 tuples 無法與右關聯的 tuples 配對者仍會保留在 RESULT 內，但其屬於右邊關聯的屬性會填入空值。）

- Alternatively, outer join can be computed by executing a combination of the following relational algebra operators:
  TEMP1 ← $\pi_{\text{LNAME, FNAME, DNAME}}$ (EMPLOYEE $\bowtie_{\text{DNO = DNUMBER}}$ DEPARTMENT)
  TEMP2 ← $\pi_{\text{LNAME, FNAME}}$ (EMPLOYEE) — $\pi_{\text{LNAME, FNAME}}$ (TEMP1)
  TEMP2 ← TEMP2 × 'null'
  RESULT ← TEMP1 ∪ TEMP2
  TEMP1 寫進寫出三次，TEMP2 寫進寫出四次，速度就慢了。在實作時可把某些順序作適當調整，可增快速度。

Implementation for operations involving the combination of multiple operators

- For example, it is quite common that selections are followed by a join.
- Creating a temporary file to hold tuples of intermediate table is not efficient.
- By combining the previous algorithms, it is possible to eliminate the overhead of temporary files.
- 設法減少暫存檔的數量及大小，則對執行效率將會有很大的改善。

（四）Using Heuristics in Query Optimization

- A **query tree** is a tree data structure that corresponds to a relational algebra expression.
  - **The leaf**：input relations.（葉結點表示輸入的關聯）
  - **The internal node**：relational algebra operators.（內部結點表示關聯代數運算）
- A query tree is commonly used as an internal representation of a query.
- A **canonical query tree**（典型的查詢樹）is typically generated for an SQL query without doing any optimization.
  - The leaves in a canonical query tree are from the '**FROM** clause', connected by cartesian product (×).
  - Selection (from **WHERE** clause；σ) are then applied as an internal node.
  - Projection (from **SELECT** clause；π) are finally applied above selection.
  - 參考課本 Figure 18.4 (b) in page 605.
- Steps in converting a query tree during heuristic optimization（經驗法則最佳化查詢樹的步驟）
- moving selection down.（愈先選出有關的資料錄，則後續運算中處理資料量愈少，減少記憶空間）
- applying the more restrictive SELECT first.（先做選出的資料錄量較少的 SELECT）
- replace × and σ with ⋈.（將前兩步驟選出的資料做 JOIN）
- moving PROJECT down.（先選出有用的屬性，減少後續資料量，減少記

憶空間）

- 產生 **final query tree**，這是最有效率的查詢方式。
  參考課本 Figure 18.5.(608-610 頁)。

（五）Using Selectivitiy Cost Estimates in Query Optimization

- Query optimization can be conducted by comparing different strategies fairly and realistically.
- The strategy with the lowest cost estimate is chosen.（選擇最低的預估成本者）
- This type of query optimization is more time-consuming and thus suitable for “**compiled query**”.（對於編譯碼查詢可做比較精細的最佳化預估，但是對於解譯碼查詢"interpreted query"就必須選較省時的最佳化預估。）
- It is also called systematic query optimization.
- It is first used by IBM System R and is now popular in most major commercial DBMSs.
- 通常我們只選幾種可行方法來做成本估算，假如每種可行方法都預估其成本的話，就反而花太多時間在做估計成本的工作了。這種方法叫做 **"cost-based query optimization"**。

Cost Components for Query Execution

執行查詢的成本包含下列各項：

- Access cost to secondary storage.（到硬碟搜尋或讀寫資料頁的成本）
- Storage cost for intermediate files.（產生中間暫時檔的儲存成本）
- Computation cost.（在記憶体做搜尋、排序、合併或欄位值計算的執行成本）
- Communication cost.（資料交換成本）
  ⑤Memory usage cost.（執行時所需緩衝記憶体成本）
- For large databases, the bottleneck is on access cost.
- For small databases, where most data are cached in main memory, the bottleneck is on computation cost.
- For distributed databases, the bottleneck is on communication cost.（網路上多個 server 時）
- In the following, the bottleneck is on access cost.（以下本章只討論抓取資料頁的成本）

Catalog Information Used in Cost Functions

- **r**：number of records. (tuples)　　　　例：$r_{EMPLOYEE}$
- **b**：number of blocks.　　　　　　　　例：$b_{EMPLOYEE}$
- **bfr**：blocking factor, i.e. number of records contained in a block.例：$r_{EMPLOYEE} / b_{EMPLOYEE} = bfr_{EMPLOYEE}$
- information regarding index structure：
  - ■ **x**：number of levels in a multilevel index structure.(INDEX 的層數；例：x_A，A 爲 INDEX 的屬性)
  - ■ $\mathbf{b_{I1}}$：number of first-level (leaf) index blocks.（INDEX 的葉結點頁數；例：$b_{I1.A}$）

    - **d**：number of distinct values of an index attribute.(例：$d_{SEX} = 2$)
- **s**：**selection cardinality**: number of resultant tuples for each equality selection on an attribute. For example, s=1 for key attribute, and **s = r/d** for non-key attribute.
- Query optimizer only needs **reasonably close values** of the required catalog information.

Cost Functions for SELECT

- S1：Linear Search (brute force；依序搜尋每個檔案頁)
    - non-key search：b
    - key search：b/2
- S6：Using a secondary index (B+ tree)
    - equality comparison：x + s (s=1 for key equality)
    - range comparison：(assuming half of the file records satisfy the condition)

      $$x + b_{I1}/2 + r/2$$

      假設一半記錄滿足所給狀況，即經 x 次的索引內部結點頁(internal block)搬動後找到索引葉結點處，然後需搬動 $b_{I1}/2$ 次葉結點頁(leaf block)找出滿足所給狀況的全部資料，最後還要憑每個資料指標(data pointer)，再搬動 r/2 次資料頁來找出資料（最壞預估，每筆資料均在不同資料頁）。
- S7：Use a simple condition as the index：the same as S6
- S8：Use a composite index：the same as S6.
- Example：The following is the catalog information about EMPLOYEE relation.
    - r = 10000, b = 2000, bfr = 5
    - clustering index on SALARY, $x_{SALARY} = 3$, s_{SALARY}=20(20 級；屬性值域的個數)
    - B+ tree index on SSN, x_{SSN}=4, s_{SSN}=1
    - B+ tree index on DNO, x_{DNO}=2, b_{I1DNO}=4, d_{DNO}=125(125 個部門), So $s_{DNO}=r/d_{DNO}$ = 80.(平均值)
    - B+ tree index on SEX, x_{SEX}=1, d_{SEX}=2, $s_{SEX} = r/d_{SEX}$=5000.(假定男女各半)

  試以上述條件計算下列問題之成本：

  (OP1)：$\sigma_{SSN=123456789}$(EMPLOYEE)
  Linear：b/2 = 1000
  Index：$x_{SSN} + 1 = 4 + 1 = 5$ (採用此法較佳)

  (OP2)：$\sigma_{DNO > 5}$ (EMPLOYEE)
  Linear：b = 2000 (採用此法較佳)
  Index：$x_{DNO} + b_{I1DNO}/2 + r / 2 = 2 + 4/2 + 10000/2 = 5004$

  (OP3)：$\sigma_{DNO=5}$(EMPLOYEE)
  Linear：b = 2000
  Index：$x_{DNO} + s_{DNO} = 2 + 80 = 82$ (採用此法較佳)

  (OP4)：$\sigma_{DNO=5 \text{ AND } SALARY>3000 \text{ AND } SEX = \text{'F'}}$ (EMPLOYEE)
  Linear：b = 2000
  Index DNO：82 (採用此法最佳)
  Index SALARY：x_{SALARY} + b/2 = 1003.（clustering index 已

排序，是循序的，由 INDEX 找到葉結點起點 30000 後就直接依葉結點找大於的部分，平均來說只找一半)

Index SEX：$x_{SEX} + s_{SEX} = 1 + 5000 = 5001$

Cost Functions for JOIN

- Join selectivity, **js**, of R ⋈ S is defined as：　（每次配對成功率）
  js = | R ⋈$_c$ S | ／ (| R | ＊ | S |)　(關聯 R 的 tuple 數目是 | R |)
- When c is ϕ（沒有連結條件）, js = 1　（與 CARTESIAN PRODUCT 一樣）
- 當沒有合連結條件（c）的記錄時，**js** = 0，通常是 $0 \leq$ **js** ≤ 1
- When c = "A = B"
  - If A is the key of R, then | R ⋈$_c$ S | ≤ | S |, so **js** ≤ (1 / | R |)
    （**js** ≤ (| S | / | R | ＊| S |)
  - If B is the key of S, then | R ⋈$_c$ S | ≤ | R |, so **js** ≤ (1 / | S |)
    （**js** ≤ (| R | / | R | ＊| S |)

Cost Functions for R ⋈$_{A=B}$ S

- (J1)：Nested Loop (Assume the memory buffer can accommodate two pages)
  $$cost = b_R + (b_R * b_S) + (js * r_R * r_S / bfr_{RS})$$
  see

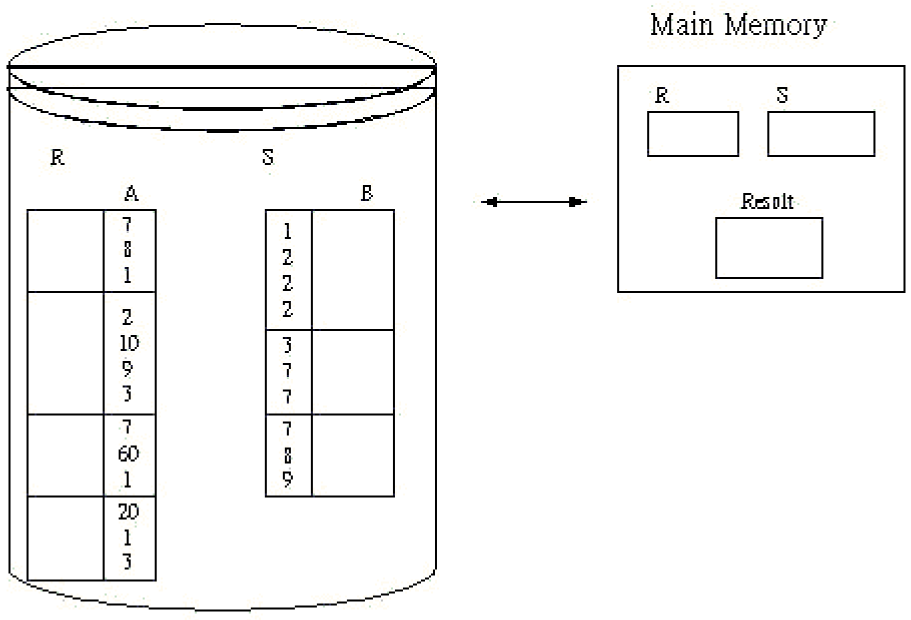


Figure 6

R 為 Outer loop，先抓第一個 block 與 S 的每一個 block 比對一次，再抓第二個 block 與 S 的每一個 block 比對一次，依序直到 R 的每一個 block 都比對完畢，合乎條件者存到 Result 內，Main Memory 的 Result buffer 滿了的話就要存回 disk 內。故 R 要抓 b_R 次，S 要抓 $b_R * b_S$ 次，
Result 要搬$(js * (r_R * r_S)) / bfr_{RS}$ 次（r_R = | R |，r_S = | S | ）。

- (J2)：Using an index, say B of S. （S 以 B 作索引，使用此索引搜尋時）
  $cost = b_R + r_R * (x_B + s_B) + js * r_R * r_S/bfr_{RS}$ （x_B = index level，s_B = selection cardinality B of S）
  If B is a clustering index：
  $cost = b_R + r_R * (x_B + (s_B/bfr_B)) + js * r_R * r_S/bfr_{RS}$（假如搜尋值都在同一頁則 $s_B/bfr_B = 1$）
  If B is the primary key：
  $cost = b_R + r_R * (x_B + 1) + js * r_R * r_S/bfr_{RS}$
- (J3)：Sort-merge.（兩個檔案都已經以 join attributes 先作了排序）
  $cost = b_R + b_S + js*r_R*r_S/bfr_{RS}$
  If sorting is needed, the sorting overhead has to be included.
  See

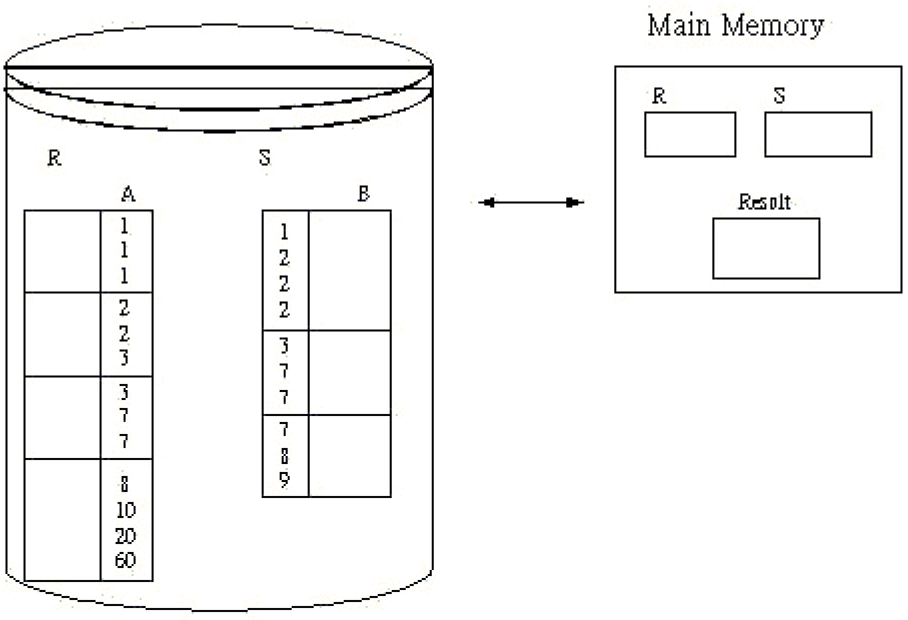


Figure 7

- Example：The following information is about the relation DEPARTMENT.
  (1) $r_D = 125$, $b_D = 13$
  (2) primary key is on DNUMBER, and $x_{DNUMBER} = 1$.
  (3) B+ tree is on MGRSSN, and $x_{MGRSSN} = 2$.
- (OP6)：EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT
  $js_{OP6} = 1/r_D = 1/125$
  Assume $bfr_{ED} = 4$
  Using method J1 (Nested Loop).
  Case 1：EMPLOYEE is the outer loop.
  $cost = b_E + b_E * b_D + js_{OP6} * r_E * r_D / bfr_{ED}$
  $= 2000 + 2000 * 13 + 1/125 * 10000 * 125/4 = 30500$
  Case 2：DEPARTMENT is the outer loop.
  $cost = b_D + b_D * b_E + js_{OP6} * r_E * r_D / bfr_{ED}$
  $= 13 + 13 * 2000 + 1/125 * 10000 * 125/4 = 28513$

Using method J2 (Access method).

Case 1：Using the index on DNUMBER of DEPARTMENT.

（EMPLOYEE 是 outer loop）

$$cost = b_E + r_E * (x_{DNUMBER} + 1) + js_{OP6} * r_E * r_D / bfr_{ED}$$

（DNUMBER 是 primary key）

$$= 2000 + 10000 * 2 + 1/125 * 10000 * 125/4 = 24500$$

Case 2：Using the index on DNO of EMPLOYEE.

（DEPARTMENT 是 outer loop）

$$cost = b_E + r_D * (x_{DNO} + s_{DNO}) + js_{OP6} * r_E * r_D / bfr_{ED}$$

$$= 13 + 125 * (2 + 80) + 1/125 * 10000 * 125/4 = 12763$$

上述四種方法裏，以 J2 之 Case 2 爲最佳。

- Example：Compute the cost of different strategies for OP7 by using J1 and J2.
  (OP7)：DEPARTMENT $X_{MGRSSN=SSN}$ EMPLOYEE　(自行練習)

十、Transaction Processing Concept (Chapter 19)

（一）Introduction to Transaction Processing

What is a transaction?

1. **Transaction**：The execution of a program that accesses or changes the contents of the database.
   (是處理資料庫時的一個單元動作，包括一個或多個的插入、刪除、更改、或存取等運算。)
2. All multi-user DBMSs support transaction processing facility.
3. From the database point of view, a transaction is a sequence of read and write operations. Please see

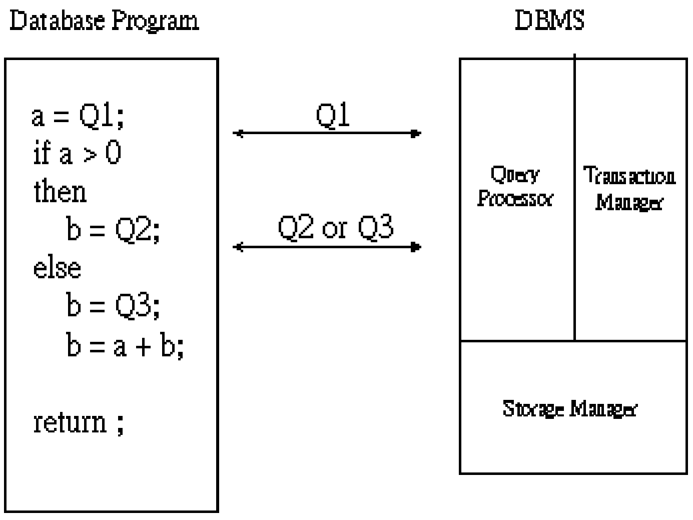


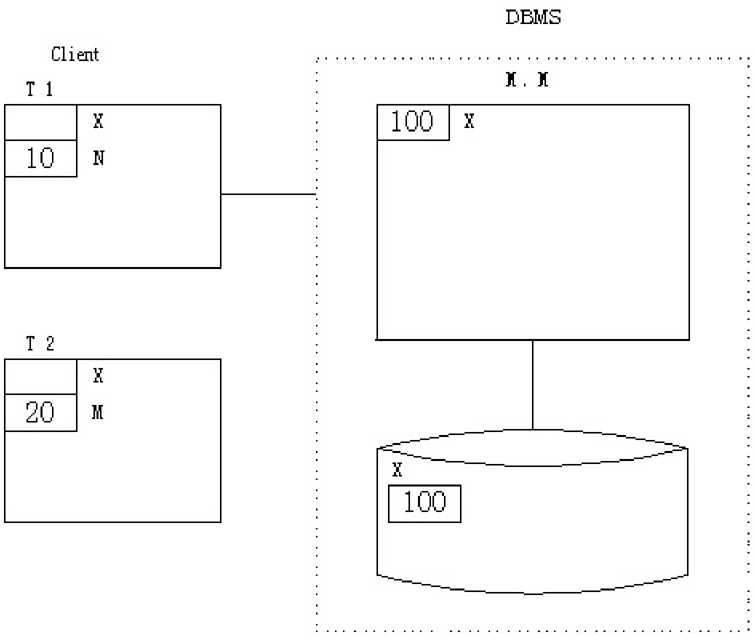
Figure 4

   Conceptually, these operations can be classified into：
   - **read-item(X)**：Read a database item X into a program variable X.
   - **write-item(X)**：Write the value of variable X into the database item named X.
   - **commit**（認定）: All the effects of the database execution so far have been installed into the database.
   - **abort**: All the effects of the database execution have been erased as if it has never happened before.
4. See Figure 19.2 in page 632 for sample transactions.

Why is concurrency control needed?

Without proper control in the presence of concurrent executions, the following anomalies may happen.

1. **The Lost Update Problem**：The update effect of one transaction is lost. See Figure 19.3(a) in page 634 for an example, and.



（T1 讀取經運算尚未寫入前 T2 就讀取，然後 T1 寫入，之後 T2 經運算又寫入，覆蓋掉 T1 的運算結果，等於沒做 T1 修改 X 的運算。）

2. **The Dirty Read Problem**：The update effect of one transaction has been read before it commits.
   See Figure 19.3(b) in page 634 for an example.（當 T1 失敗需回復原值前，T2 已經讀取了）
3. **The Incorrect Summary Problem**：The execution of a summary transaction is interleaved with the execution of an update transaction.（T1 的 Y 值更新在 T3 讀取之後，故 T3 的加總值不正確）
   See Figure 19.3(c) in page 635 for an example.
4. **The Nonrepeatable Read Problem:**
   Read of the same data item at two different places in a transaction get different values.（在 T2 兩次讀取之間，x 的值被改變，故兩次讀取之值不同）

T_1	T_2
	r(x)
w(x,5)	
commit	
	r(x)
	commit

Why Recovery is Needed?

Desired properties about transactions：**All or Nothing**.（完整的執行完畢或保持原狀都不做）

Types of Failures：

1. **User Abort**：It may occur due to any of the following reasons：
   - When an ‘ABORT’ command is explicitly issued within a transaction.
   - Because some undesired conditions are detected by the program, e.g., insufficient balance for a withdrawal transaction.

2. **System Abort**：It may occur due to any of the following reasons：
   - Hardware or software system failure.
   - Transaction or application program error, e.g. divide by 0.
   - Concurrency control enforcement, e.g. to resolve deadlock.
3. **Media Failure**，It may occur because of any of the following：
   - Disk crash.
   - Catastrophe, e.g. fire or earthquake.

Types 1 and 2 are handled by the DBMS using recovery techniques, and Type 3 is handled by backup.

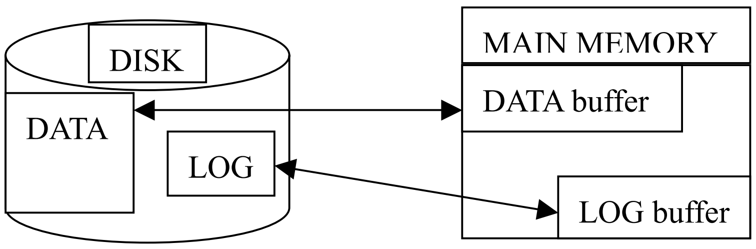
（二）Transaction and System Concepts.

Transaction States and Additional Operations

1. The recovery manager keeps track of the following operations：(爲了回復資料，系統需追蹤整個交易過程。)
   - BEGIN-TRANSACTION.
   - READ or WRITE (produced by SQL).
   - END-TRACTION：註明讀寫運算的終止。
   - COMMIT：The changes to the database will not be undone even in the presence of failure.
   - ABORT (ROLLBACK)：The effect that the transaction has applied to the database is undone.
2. The Transaction State：
   See Figure 19.4. where ABORT is generated either by the user or by the system.

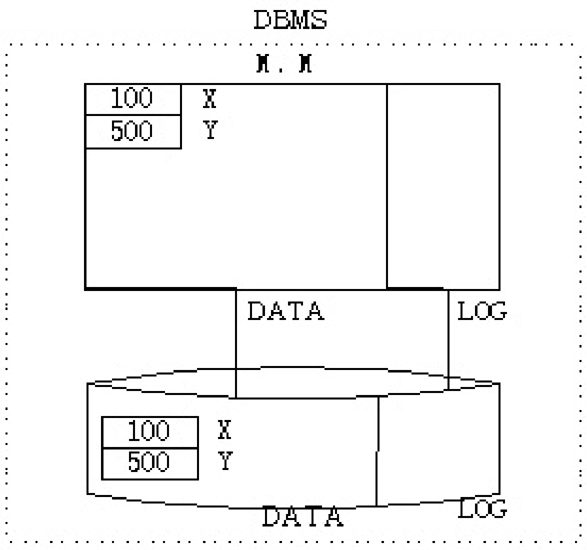
The System Log

1. To implement commit and abort correctly, the DBMS maintains a log, **also called journal**.
2. The log is a sequence of log records that are kept on disk.
3. There are five types of log records：
   - [start-transaction, T].
   - [write, T, X, old_value, new_value].（old_value 稱爲 before image[BFIM]，new_value 稱爲 after image[AFIM]。）
   - [read, T, X].
   - [commit, T].
   - [abort, T].
4. When an operation is performed, the corresponding type of log record is inserted into the log in sequential order.
5. The log is stored on disk, but there is a log buffer in main memory.
   LOG buffer 大，可減少 access 次數，提高 performance，但太大又會減少 DATA buffer 須適度調整。



Commit Point of a Transaction

1. What is **the commit point of a transaction**? when the “commit” log record has been written in the log and all the log records up to the commit record have been flushed to the disk.（交易完成要包含 log 都寫到磁碟內）
2. Note that even after the commit point, the data pages written by the transaction may not have been written to the disk.
3. Since all log records of the transaction have been written to the disk after the commit point, its effect can be redone even failure occurs.
4. When crashes occur, if ‘commit’ log record of a transaction is not found in the log disk, this transaction is pronounced to be ‘aborted’. All its before images will be applied to restore to the original database state. See



```
log:[ Start-trans, T1][read, T1, X][write, T1, X, 100, 200][Commit, T1]
       [ Start-trans, T2][read, T2, X][read, T2, Y][write, T2, Y, 500, 700][Commit, T2]
```

Checkpoints in the System Log

1. The log will occupy more and more space, and the recovery will takes longer and longer time.
2. To conquer this problem, a [checkpoint] log record is inserted to the log when all the memory buffers are flushed to the disk. **The checkpoint indicates that up to this point, the effect of all committed transactions has been reflected in the database on the disk**.
3. Redo can be conducted from the [checkpoint] log record down the log.
4. **Operations taken by a checkpoint**：
   - Suspend the execution of transactions.（暫停執行交易）
   - Force-write all update operations of committed transactions from main memory buffers to disk.（強制將已認定的交易的所有更新資料從主記憶体寫入磁碟內）
   - Write a [checkpoint] record to the log and force-write the log to disk.（將 checkpoint 記錄加到 log 內，並強制將 log 寫入磁碟內）
   - Resume executing transactions.（繼續執行新 transactions）
5. In reality, not every DBMS uses the same checkpointing technique. **The purpose of checkpointing is to reduce the amount of log records**

scanning at recovery. Any algorithm that meets this purpose can be used.（要執行 Recovery 時從最近一個 checkpoint 之後開始即可，執行 checkpoint 時會影響 on-line 的執行速度，多久執行一次 checkpoint 得視情況而定）

（三）Desirable Properties of Transactions

1. The following are the **ACID properties that a legal transaction should possess**：
   - **Atomicity**：All or nothing.（一定要完整正確作完，否則都不作）
   - **Consistency**：A transaction brings the database from one consistent state to another.（一致性）
   - **Isolation**：A transaction should not make its updates visible to other transactions until it is committed.（隔絕性）
   - **Durability**：Once a transaction commits, its effects must never be lost.（穩定存在）

（四）Schedules and Recoverability

Schedules (Histories) of Transaction

1. We want to figure out what kind of transaction execution in an interleaved fashion is acceptable.
2. A **schedule** (or called **history**) of transactions $T_{1,}$ T_2, ..., T_n is a (total) ordering of the operations in $T_{1,}$ T_2, ..., T_n such that the operations of any transaction T_i appear in the same order in which they occur in T_i. For example, a schedule of Figure 19.3 (a) in page 634 can be written as：
$r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$ 【X:=X-N 是 client 端執行的運算，故不包含於此】
3. Two operations are said to conflict（衝突）if they satisfy all of the following conditions：
   - They belong to different transactions.
   - They access the same data item.
   - At least one of them is a write operation. (thus, there are r-w, w-r, and w-w conflicts).
4. 一個完整的排程(completed schedule；S)應具有下列三種情形：
   1. 包含與其有關的所有交易（$T_{1,}$ T_2, ..., T_n）的全部運算(operation)及各交易最後的認定或取消運算。例如：$r_1(X)$, $r_2(X)$, $w_1(X)$, $r_1(Y)$, $w_2(X)$, $c_{2,}$, $w_1(Y)$, c_1（c 表示 commit, r 表示 read, w 表示 write）。
   2. 就交易的運算而言，出現在排程中的順序應與出現在各別交易中的順序相同。
   3. 就發生衝突的兩個運算而言，在排程中一定是一前一後出現。

Characterizing Schedules Based on Recoverability

1. A schedule S is **recoverable** if any transaction T in S commits after all transactions that T read data from commit.
2. If a schedule is not recoverable, some committed transaction may have to be

rolled back. For example,
$r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$

3. However, a recoverable schedule may still introduce “cascading rollback”. For example
$r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); a_1;$
4. A schedule is said to be **cascadingless** (i.e. to avoid cascading rollback) if every transaction in the schedule only read data items that were written by committed transactions.（要讀某一資料前先確定該資料已 commit）
5. However, even in a cascadingless, we cannot just place the “before-image” of a data item X when a transaction that writes X is aborted. For example,
   - $w_1(X,5); w_2(X,8); a_1;$
   - suppose the initial value of X is 9
   - The system log is
     [T1, start];[T1, w, X, 9, 5];[T2, w, X, 5, 8];[T1, ABORT];
   - 此時不能直接將 X 的値改成 before-image 9.
6. A schedule is said to be **strict** if transactions cannot read or write a data item X until the transaction that last wrote X has committed.
7. Strict schedule puts restrictions on operations with w-r or w-w conflicts.

（五）Serializability of Schedules

1. What kinds of schedules are considered ‘correct’?
   Ans.：**serial schedule**.
2. Transactions are considered “independent”. Thus the order of execution is irrelevant.
3. Problems with serial schedules： waste system resources, e.g. CPU and memory.
4. A schedule is serializable if it is equivalent to some serial schedule.
5. Types of "equivalence"：
   - Result equivalence：
     - hard to check
     - results may be the same by accident
   - View equivalence：
     - the data read by each read operation are the same
     - the data written are the same.

       e.g.：$r_1(X); w_2(X); w_1(X); w_3(X); c_1; c_2; c_3;$
6. It has been shown that there is no efficient way for testing view serializability.
7. Conflict equivalence：the order of any two conflicting operations is the same in both schedules.
8. A schedule is conflict serializable if it is conflict equivalent to some serial schedule. For example, schedule D of Figure 19.5 (c) is conflict equivalent to schedule A of Figure 19.5(a).

Testing for Conflict Serializability of a Schedule

1. Use a serialization graph (or called precedence graph) (V, E) to represent a schedule S, where V is a set of committed transactions in S and $(T_i, T_j) \in E$ if
   - one operation e_i in T_i conflicts with another operation e_j in T_j, and
   - e_i precedes e_j.
2. A schedule is **serializable** if its corresponding serialization graph contains no cycles.
   See Figure 19.7 (c) and (d).
   See Figure 19.8 (b) and (c)
3. This approach is practically of no use because of its high overhead.

(六) Transaction Support in SQL

1. With SQL, a single SQL statement is always considered to be atomic.
2. To group a sequence of SQL statements as a transaction, an explicit COMMIT or ROLLBACK is specified at the end. Note that SQL does not have explicit Begin_Transaction statement.
3. SQL2 supports a SET TRANSACTION statement, which allows users to specify the properties about transaction execution.
4. The property about transaction interleaving is called **isolation level**:
   - UNCOMMITTED: allows dirty read, nonrepeatable read, and phantom
   - READ COMMITTED: disallows dirty read, but allows nonrepeatable read and phantom
   - REPEATABLE READ: disallows dirty read and nonrepeatable read, but allows phantom
   - SERIALIZABILITY: disallows dirty read, nonrepeatable read, and phantom
5. Dirty read and nonrepeatable read are defined at the very beginning of this unit. Phontoms is defined as follows:
   - A transaction T1 may read a set of rows that satisfies a condition C from a table. Now after T1 read these rows, another transaction T2 insert a row that satisfies C. This new row is considered as a phantom because if T1 is to be re-executed, this new row will be retrieved.
6. A sample embedded SQL transaction:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
        READ WRITE
        DIAGNOSTICS SIZE 5
        ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE(FNAME, LNAME, SSN, DNO,
SALARY) VALUES (‘Robert’, ‘Smith’, ‘991004321’, 2, 35000);
```

```
EXEC SQL UPDATE EMPLOYEE SET SALARY=SALARY*1.1 WHERE
DNO=2;
EXEC SQL COMMIT;
GOTO THE-END;
UNDO: EXEC SQL ROLLBACK;
THE_END: …
```

十一、Concurrency Control Techniques (Chapter 20)

1. This chapter introduces protocols that guarantee serializability.
   - Two Phase Locking (2PL).
   - Timestamp Ordering (T/O).
   - Optimistic.
2. In practice, only (strict) 2PL is used it makes recovery easier.
3. In the following, we only describe 2PL.

（一）Locking Techniques for Concurrency Control

1. A lock is a variable associated with a data item in the database. A lock describes the status of the associated data item.

Types of Locks and System Lock Tables

1. **Binary Locks.**
   - Domain：{locked, unlocked}（or 1 and 0, for simplicity）
   - Operations：
     - lock(x)：lock state of x = locked
     - unlock(x)：lock state of x = unlocked.
   - Before accessing a data item x, a transaction must first issue lock(x). After finishing accessing x, unlock (x) can be issued.
   - See Figure 20.1 in page 663 for detailed operations of lock() and unlock().
   - Note that both operations must implemented as indivisible unit.
2. **Shared and Exclusive Locks**.
   - Domain：{shared-locked, exclusive-locked, unlocked}.
   - Operations：
     - read-lock(x)：lock state of x = shared-locked.（允許其他交易來讀取）
     - write-lock(x)：lock state of x = exclusive-locked. （不允許其他交易來讀取，專用的）
     - unlock(x)：lock state of x = unlocked.
3. See Figure 20.2 in page 665 for exact operations.
4. Again, these operations must be indivisible.
5. Each record in the lock table has four fields：

   ＜data item name, LOCK_STATE, no_of_reads, locking_transaction(s)＞
6. Rules for share/exclusive locking:
   - A read_lock(x) or write_lock(x) must be issued before any read_item(x) can be performed.
   - A write_lock(x) must be issued before any write_item(x) can be performed.
   - Unlock(x) must be issued after all read_item(x) and write_item(x) are completed in the transactions.
   - A transaction will not issue read_lock(x) if it already holds read lock or

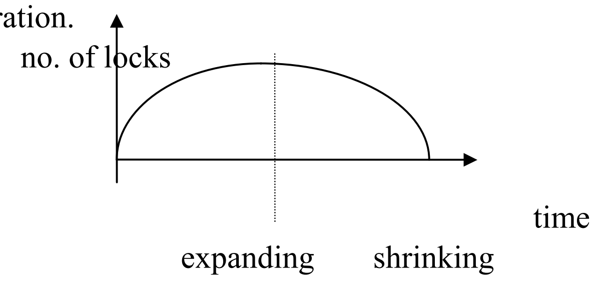
write lock on x.

- A transaction will not issue write_lock(x) if it already holds a write lock on x. However, if it already holds a read lock, write_lock(x) will try to upgrade the lock status to write_lock.

7. Two extra locking operations（Conversion of locks）：
   - lock upgrade：read-lock(x) followed by write-lock(x) in the same transaction.
   - lock downgrade：write-lock(x) followed by read-lock(x) in the same transaction.
8. Using binary locks or shared and exclusive locks does not guarantee serializability. See Figure 20.3 in page 666.

Guaranteeing Serializability by 2PL

1. **Two Phase Locking** protocol：All locking operations precede any unlock operation.



2. See Figure 20.4 for transactions following 2PL protocol.
3. Downgrading of locks must be done at the shrinking phase in order to guarantee serializability. (Why?)
4. Why does 2PL guarantee serializability? Use the Serialization Graph to prove the non-existence of cycles if 2PL is employed.
5. Variations of 2PL
   - Conservative 2PL：All locks are acquired before the transaction begins execution. *Critique*: not practical.（實作上比較難執行）
   - **Strict 2PL**：All locks are released at commit or abort. This is the most popular approach and can be easily used with recovery technique.（除非交易已經 commit 或 abort，否則不能解除 locks）

Deadlocks and Livelocks

1. See Figure 20.5 for an example.
2. Deadlock Prevention

   Each transaction is assigned a timestamp. When a transaction T_i requests a lock which is held by T_j,

   - ■ No waiting： T_i is aborted.
   - ■ Wait-die：old waits for new

     if $TS(T_i) < TS(T_j)$
     then T_i is allowed to wait
     else T_i is aborted
   - ■ Wound-wait：new waits for old

     if $TS(T_i) < TS(T_j)$

then abort T_j
else T_i waits

- Cautious waiting：
  if T_j is not blocked (not waiting for some other locked item)
  then T_i is blocked and wait
  else T_i is aborted

3. Deadlock Resolution
   - Timeout：Disadvantage：the timeout period is hard to determine.
   - Deadlock Detection and Breaking
     - Wait-for-graph: See Figure 20.5.
     - Deadlock breaking (victim selection).
4. Livelock
   - A transaction is livelocked if it is blocked for an indefinitely period of time.
   - Livelock occurs when locking protocol is not fair. For example, if a read lock request that comes later can be granted while a write lcok request that comes earlier has to wait.

（二）Granularity of Data Items

1. A database item could be chosen to be one of the following：
   - The whole database.
   - Table (A whole file).
   - Page (Adisk block).
   - Record.
   - Field.
2. LOCK 的單位的大小，影響執行效率，通常是愈小愈好，但太小也有缺點。Finer granularity allows higher concurrency while introduces more overhead：
   - More storage space.
   - More searching time for lock and unlock operations.
   - More maintenance overhead.

Degree of Isolation (theoretical description)

1. Degree 0：(Dirty Read) write-lock is acquired before each write operation and released after it.
2. Degree 1：(Browse Access) Degree 0 + write-lock is held until commit/abort. **Degree 1 can prevent lost update problem**.
3. Degree 2：(Cursor Stability) Degree 1 + read lock is acquired before each read operation and released after it.
4. Degree 3：(Repeatable Read) Degree 2 + read locks are held until commit/abort, i.e. **strict 2PL**.

十二、Database Recovery Techniques (Chapter 21)

（一）Recovery Concepts

1. **Steal/No-Steal**：If a cached page updated by a transaction **cannot** be written to disk before the transaction commits, this is called a **no-steal**. In other words, a cached page has to be **pinned** until the updating transaction commits. Of course, no-steal requires the system to allocate a very large buffer space for the updated pages in memory.
2. **Force/No-Force**：If all pages updated by a transaction are immediately written to disk when the transaction commits, this is called **force** approach. Of course, force demands more I/O operations.
3. Typical database systems employ a steal/no-force strategy.

（二）Types of Recovery

Deferred update(延後更新)

1. Each update is first recorded in the local transaction workspace.
2. At commit, all the updates are first recorded persistently in the log and then written to the database.
3. No undo is needed at failure.（直到交易認定後資料才寫入到磁碟，故萬一交易失敗也不必 UNDO）
4. 若在交易認定後資料寫入磁碟之前當機，則需根據 log 記錄重做交易運算。

Immediate Update(立即更新)

1. The database is immediately updated at each operation.
2. Before an update is flushed to the disk, the undo log must be recorded in the disk (called **write-ahead logging**；**WAL**). This can occur before commit.（在更新資料寫入磁碟之前先把之前的 undo log 寫到磁碟內）
3. At commit, the log records of a transaction must be written to the disk.
4. It needs undo/redo log.
5. If there is no cascading abort, only write operations need log records.
   - **REDO-type log record**： [write, T, x, after image]
   - **UNDO-type log record**： [write, T, x, before image]

6. In practice, all commercial DBMSs use strict 2PL. Thus, cascading abort will not occur. **Immediate update is more popular**.

Recovery for UNDO/REDO Immediate Update with Concurrent Execution

1. From the last checkpoint, get **active transaction list**.
2. Scan forward from the last checkpoint in the log, apply the redo log, and form：
   - **committed transaction list** (since the checkpoint).
   - **active transaction list** (at the failure time).

3. Undo the write operations of active transactions by scanning backward and applying the undo log record.

（三）Shadow Paging

1. A page table maintains a list of pointers to actual disk pages. It is called **shadow page table**.
2. When a transaction starts, the shadow page table is copied to the current page table in the local memory. See Figure 21.5.
3. When an update to a page occurs, a new disk page is created and pointed by the current page table.
4. When a transaction commits, all the written pages are flushed to the disk and the current page table is merged to the shadow page table.
5. When a transaction aborts, the current page is discarded.
6. Advantages：Recovery is fast since neither undo nor redo is needed.
7. Disadvantages：
   - Need more space, which in turn complicates garbage collection.
   - Committing a transaction takes more time.