資料庫管理系統

期末報告：

物件導向資料庫介紹

Introduction Object Oriented Database Management System

指導老師：黃三益 教授

學生：陳慧靜 D954020011

鄭光廷 D964020001

目　　錄

1. 物件導向資料庫基本概念

1.1 物件導向資料庫

資料庫有名的學者 Michael Stonebraker 認為物件導向式資料庫(Object-Oriented DBMS,OODBMS)和物件關連式資料庫(Object-Relational DBMS,ORDBMS),會是廿一世紀新世代資料庫科技的主流。而如今現行的資料庫市場來看，純物件導向式資料庫還是無法成功的打入市場，倒是物件關連式資料庫則漸漸的被傳統獨佔市場的關連式資料庫所接受,慢慢的引進物件的觀念進入其原本的關聯式資料庫。

從新興的網路應用程式來看，越來越多的資訊系統必須有賴物件技術來提供支援，以促進系統開發人員的維護與設計。關於物件資料庫和物件關連資料庫的核心技術，可以從其資料模式和系統架構分成四類：(1)延伸性物件資料庫(Extended Object-Oriented Database Systems)：主要是延伸關連式資料庫(Relational Database Systems)，加上封裝、繼承、和抽象資料型態而成。產品如 UniSQL、Illustra、和 OpenGres；(2)物件程式語言資料庫(Object-Oriented Database Programming Languages)：主要是延伸物件程式語言如 C++和 Smalltalk，加上儲存、回復、和同時存取資料庫功能而成。產品如 O2、ObjectStore、Objectivity、和 Ontos；(3)功能性模式物件資料庫(Functional Object-Oriented Database Systems)：主要是應用功能性模式方法，以函式呼叫來建置資料庫和使用資料庫的功能而成。產品如 Iris、Open ODB、SIM；(4)物件產生器(Object Generators)：主要是以物件產生和物件儲存為主，不提供資料庫管理和介面的功能，是最原型的物件資料庫。例如 Exodus、Genesis、LOOM..等。

以上四類都可以使用資料模式(Data Model)來定義物件、屬性、行為、關連、組合、及繼承等；而有別於關連式資料庫的資料模式，是以表格檔為主，利用主要鍵和外鍵處理彼此對應情形，而運算部份則由程式來完成。而在資料正確(Data Integrity)部分，物件資料庫或物件關連資料庫是以 OID(Object ID)為主，省去關連式資料庫的主要鍵與外鍵維護困擾，並提供多種檔案間對應和連結可能性，例如組合、繼承、複合關係，藉由欄位定義達成。在資料使用(Data Manipulation)部分，關連式資料庫在宣稱性語言與序性語言上無法銜接，須作鬆散式連結；而物件資料庫和物件關連資料庫，在這方面是定義物件介面、運作、回復、和同時存取管理，並讓資料庫存取和資料宣告及運算，在語法和語意上成為一體語言，能緊密式連接。在資料設計(Data Design)方面，物件資料庫和物件關連資料庫是定義物件資料庫的需求分析、綱要設計、交易分析與設計、和控制設計。

物件資料庫標準主要來自三個組織，分別為 ODMG(Object Data Management

Group)、ANSI/ISO 與 OMG(Object Management Group)。ODMG 成立於 1992 年，至今已提出物件模式、物件查詢語言、物件使用語言、和物件程式語言的耦合與連結標準。由於 ODMG 組成份子大多來自物件資料庫系統廠商。所以，制定和推廣產業標準較為快速而且積極，也是目前被採行最多和影響最深的物件資料庫標準。ANSI/ISO 則延續 SQL2 國際資料庫標準語言，加入程序性和物件資料模式制定 SQL3。ANSI 的著眼點為繼續延用 SQL2 影響力，保留既有資料庫管理和開發的技術投資。並在 1998 年在 SQL3 提出七項原稿標準產生，分別是架構、基礎、應用開發介面、耦合連結、程序、XA 介面、時間資料。OMG 組成份子包括電腦硬體和軟體廠商。組織目標在制定和推廣分散式和物件式系統與應用軟體的架構標準，不限資料庫。因此，物件資料庫標準只是 OMG 物件軟體標準的一部份。由於 OMG 組成成員超過四百家廠商，而其所提標準中的 CORBA(Common Object Request Borker Architecture)，已廣被主從架構和分散式運算技術所採用。使得在未來趨勢上，局部性或全部性標準整合是勢在必行的。以下用表一顯示 OMG 與 ODMG 的差異。

以上三個組織以推廣 SQL3 為主的物件程式語言和物件關連資料存取的技術，期望提升物件技術在資訊科技與資訊管理的影響力和貢獻，希望能突顯關連資料庫所不能解決的運算和儲存困難，進而有效提出物件資料庫的能力和銜接整合的保證。

表一　　OMG 與 ODMG 的差異

		OMG	ODMG
組織面	主要組織目的	分散式資訊系統的物件導向方式	資料庫管理系統的物件導向方式
	專注目標	廣泛的資訊系統	資料庫管理系統
	企業 IT 應用範圍	大	小
	組織設立的時間	先；1989	後；1991
技術面	跨語言技術	使用 OMG IDL 對映(mapping)至多種物件導向程式語言	使用 API 與三種物件導向程式語言繫結，實作 OML
	跨平台技術	CORBA 以中介軟體的架構，可以應用於既有之系統，或未來開發之系統。 UML 定義標準套用於程式塑模。 MDA 在於各種 PSM 的調整可以實作於各種平台	直接定義 ODL、OQL 及繫結的標準，各系統依據此標準
	物件模型	OMA 依據 CORBA 流程做分類	以 OMG 之物件模型修改、擴大之

概念面	抽象	CORBA：物件的跨平台溝通 UML：程式設計的一致塑模工具，易於溝通、策劃 MDA：UML 的自然延伸，而更加強 SDLC 中的需求分析模型	使物件導向資料庫管理系統有一致的介面
	實作	ORBA：如同網路封包的標頭(header)概念 UML：13 種圖表的建立 MDA： CIM→PIM→PSM→各種程式架構→程式碼	由 SQL 加入物件相關的內容形成 ODL，不定義 OML 而以 API 取代

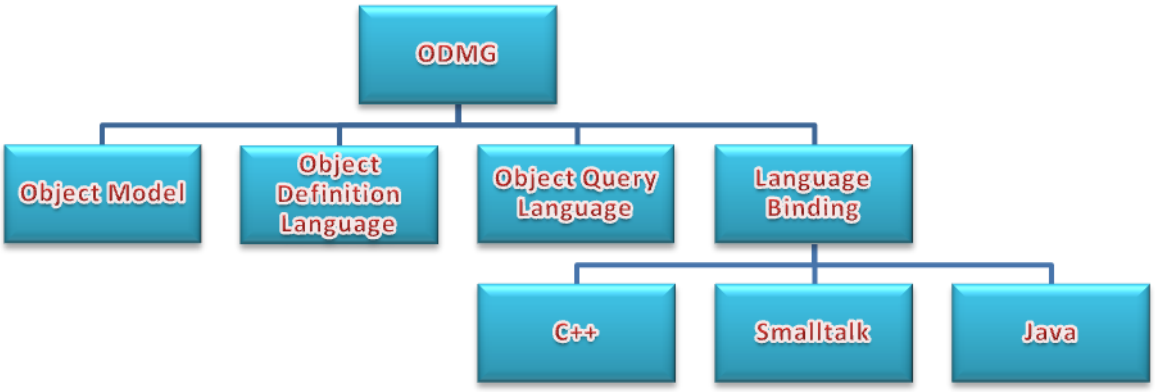
1.1.1. ODMG-物件導向資料庫新標準

早期因為缺乏一套物件資料庫的標準，無形中限制了物件觀念應用的範圍。ODMG 的主要目的是提供一套標準，讓 ODBMS 的程式設計師能夠發展可攜性的應用程式，讓一支程式可以適用於數套不同的物件式資料庫，這包含了資料綱目、程式語言、資料操作及查詢語言都必須具有可攜性。ODMG 於 1994 年修訂 ODMG-93 為 1.1 版、1995 年進一步發表 1.2 版，1996 年發展最主要核心的 2.0 版本，而於 2000 年發表 3.0 版本。3.0 版本主要是基於 2.0 版本上更增加 Java buliding，以及改善物件模式(object model)，並提高物件與關聯之間相互的對應。而在 2004 年，OMG 針對 ODMG3.0 更增加了許多 JAVA 上的應用，並也感受到 3.0 有其不足的地方，因此成立了 ODBT WG(Object Database Technology Working Group)，以發展下一世代的標準(4.0)。

針對 ODMG 的主要架構，可由圖二詳細的了解其所包含的部份。(1)物件模式(Object Model)：ODMG 的物件模式中有許多是根據 OMG(Object Management Group)的物件模式(Object Model)而得來的，OMG 的核心 model 是被設計來處理物件的需求，並擔任物件資料庫、物件導向語言及其他應用程式的中間人。ODMG 物件模式支援了「型態」(Types)、「類別」(Classes)、「封包」(Encapsulation)、「繼承」(Inheritance)，與「同體多型態」(Polymorphism)。另外還有 Object Identifier(OID)、Collection Types-聚集型態、Transaction Model 交易模式、Relationships、Extents、Keys。(2)物件定義語言(Object Definition Language)：ODL 與任何程式語言無關，也不是被用來描述物件內部實作的細節(Implementation Detail)。它是以 OMG 的 IDL(Interface Definition Language)為基礎，再增加了 Relationships、Collection、Extents 與 Keys 等特性，並且可以將這些特性對應到現存的 ODMG 物件模式(ODMG Object Model)上。(3)物件查詢語言(Object Query Language)：OQL 已經是標準 SQL 的 Superset 了，將來可以在關聯式資料庫上所

下的任何標準 SQL 指令，若移到物件導向式資料庫上做查詢，則所表達的查詢意義與所得到的結果應該是一樣的。不過要注意 OQL 所讀取出來的是物件，而非如關聯式資料庫的表格資料。(4)語言繫結(Language Binding)：在 ODMG 標準中的 Java、C++與 Smalltalk 語言繫結是用來定義物件操作語言(OMLs)，ODMG 擴充了這些程式語言使其能支援永續性的物件，讓物件操作語言(OML)使系統發展者能夠在單一的語言環境中工作，而不需使用各別的資料庫語言。

圖二　ODMG 架構



2. 物件模式 ODMG Object Model

ODMG 物件模式是 ODMG 標準的核心，許多的物件模式是來自 OMG(Object Management Group)的物件模式，物件模式的基本模型是物件，物件可以依照型態(types)來分類，同一個型態的物件表現出共同的行為(behavior)和共同的狀態(states)。在物件型態上可以執行的一組操作定義物件的行為。整體來說物件模式支援了型態(Types)、類別(Classes)、封包(Encapsulation)、繼承(Inheritance)、同體多型態(Polymorphism)。另外還有 Object Identifier(OID)、聚集型態(Collection Types)、交易模式(Transaction Model)、Relationships、Events、Keys。

在型態(Types)方面，可以看成是一個介面，而此介面定義了性質和操作。也可以看成一組資料結構，其包含了型態實例的實際表現，或是一些方法，包含了如何操作這些資料結構，使能夠支援定義在界面中可見的狀態和行為。這其中又可分為 ODMG Primitive Types 與 ODMG Domain Types。ODMG Primitive Types 提供 ODMG 支援資料型態的實作。來自以 IDL(Interface definition Language)定義的 CORBA(Common Object Request Broker Architecture)標準，這些型態在所有平台都有相同的值域和解釋，但為達到異質性，有些 ODMG 實作會要求這些型態可以用相同的 C++型態來取代。ODMG Domain Types 則是一組類別來表示定義域(domain)的抽象，如 d_String：用來表示不定長度字串的類別、d_Interval：用來表示一段持續的時間和支援其他與時間相關類別的數學運算、d_Date：支援日期的抽象，由年、月、日組成、d_Time：表示時間。

在類別(Classes)方面，類別可算是型態的介面規格或是用來定義此型態的實作。在 ODMG 模式中，類別的使用與在 C++模式中大略一致，其標準規格的類別包含 Database，Transaction，Persistent_Object，String，Date，Time，Timestamp，Interval，Ref <T>，Ref_Any，Collection <T>，Se t <T>，Bag <T>，List <T>，Varray <T>，Iterator，A_to_B，A_to_Bs，A_to_Bp。另外 ODMGODMG 標準有提供單一資料庫存取的規格，在使用資料庫物件之前必須宣告 d_Database db，利用 open 功能來開啟欲要存取的資料庫。而在例外處理方式則沿用 C++的處理機制。

在 Object Identifier(OID)方面，建立物件實例之初，資料庫會給予每個物件實例一個唯一的物件識別符（object identifier），用以區分物件之間的差別，但資料庫實行所產生的物件識別僅僅在給定的範圍之內是唯一的，通常是一個資料庫或一組資料庫。ODMG 規定單一的資料庫存取只有一個界面，所以物件識別符在資料庫中必須是唯一，但在實行時範圍可以擴大，兩個分離的資料庫環境產生的物件識別符可以有重疊的部分。如果一個物件要參考另外一個物件，它們的物件識別符必須存在於相同的識別符範圍內。

在聚集型態(Collection Types)方面，Collection 是歸類其他物件的一個物件，collection 中的成員必須是同一個型態，例如：a set of students，a list of classes。Iterator 物件是對 collection 的元素作 iterate，支援向前、向後、和任意存取 collection

的所有元素。

在 Relationships 方面，又可區分為 Unidirectional 與 bidirectional。在 Unidirectional 中，d_Ref <T>表示 to-one relationships； d_Set <d_Ref <T>> & d_List <d_Ref <T>>表示無序（unordered）和有序（ordered）的 to-many relationships。在 bidirectional 中 d_Rel_Ref <T,MT>表示 to-one relationships、d_Rel_Set <T,MT>表示無序的 to-many relationships、d_Rel_List <T,MT>：表示有序的 to-many relationships。

3. 物件定義語言 Object Definition Language

物件定義語言(Object Definition Language)是一種規格描述語言，用以定義物件型態的界面，使符合 ODMG 的物件模式。ODL 主要是定義物件資料庫資料的語言，可與 OMG CORBA 標準的界面定義語言 IDL 相容。ODL 可以擴充(extensible)，不只是未來的功能，還有實體的最佳化（physical optimizations）也可以擴充的。ODL 對應用開發者而言是有價值的，當公佈規格後，ODBMS 供應商在短時期內可支援。傳統 DBMS 提供的 DDL，讓使用者定義資料型態和界面；ODL 就像物件型態的 DDL，它定義型態的特性（包括性質和操作），但只定義操作的特徵，而不是實行這些操作的方法。其優點為可攜性(portability)，可以為應用系統提供一定程度的獨立，使能因應程式語言和 ODBMS 的改變。因此 ODL 的目的主要是定義能夠在各種程式語言上可以實行的物件型態，所以 ODL 不受特定程式語言文法結構的束縛，能讓任何依循 ODMG 標準的 ODBMS 和混合語言的實作(mixed-language implementations)，都可以使用 ODL 定義的綱要。

在規格方面，又可以分為型態特性、實例性質、屬性、關連、操作等，以下用表格成列，並舉出一些範例：

規格

<interface_dcl>	::=	**Interface** <identifier> [<inheritance_spec>] <type_property_list> [:<persistence_dcl>] {[<interface_body>]};
<persistence_dcl>	::=	**persistence** | **transient**

型態特性

<inheritance_spec>	::=	<scoped_name>|<scoped_name>,<inheritance_spec>
<type_property_list>	::=	([<extent_spec>][<key_spec>])
<extent_spec>	::=	**extent**<identifier>
<key_spec>	::=	**key[s]**<key_list>
<key_list>	::=	<key>|<key>,<key_list>
<key>	::=	<property_name>|(<property_list>)
<property_list>	::=	<property_name> |<property_name>,<property_list>
<property_name>	::=	<scoped_name>
<scoped_name>	::=	<identifier> | ::<identifier> | <scoped_name> :: <identifier>

實例性質

<interface_body> ::= <export> | <export><interface_body>
<export> ::= <type_dcl>;
|<const_dcl>;
|<except_dcl>;
|<attr_dcl>;
|<rel_dcl>;
|<op_dcl>;

屬性

<attrib_dcl > ::= **[readonly] attribute**
<domain type> <identifier>
[[<positive_int_const>**]]**
<domain_type> ::= <simple_type_spec>
|<struct_type>
|<enum type>
|<attr_collection_specifier><literal>
|<attr_collection_specifier><identifier>
<attrib_collection_specifier> ::= **Set | List | Bag | Array**

關連

<rel_dcl> ::= relationship
<target_of_path><identifier>
[inverse<inverse_traversal_path>]
[{order_by<attribute_list>}]
<target_of_path> ::= <identifier>
|<rel_collection_type><<identifier>>
<inverse_ traversal_path> ::= <identifier>::<identifier>
<attribute_list> ::= <scoped_name>
|<scoped_name>,<attribute_list>

操作

<op_dcl> ::= [oneway]<op_type_spec><identifier>
<parameter_dcls>[<raises_expr>][<context_expr>]
<op_type_spec> ::= <simple_type_spec> | void
<parameter_dcls> ::= ([<param_dcl_list>])
<param_dcl_list> ::= <param_dcl>
| <param_dcl>, <param_dcl_list>

```
<param_dcl>              ::= <param_attribute><simple_type_spec><declarator>
<param_attribute>        ::= in | out | inout
<raises_expr>            ::= raises(<scoped_name_list>)
<context_expr>           ::= context(<string_literal_list>)
<scoped_name_list>       ::= <scoped_name>
                             | <scoped_name>,<scoped_name_list>
<string_literal_list>    ::= <string_literal>
                             |<string_literal_list>,<string_literal_list>
```

```
interface Professor:Person
(           extent professors
keys faculty_id,soc_sec_no):persistent
{
    properties
    operations
};
```

```
interface Professor: Person
(           extent professors
            keys faculty_id,soc_sec_no): persistent
{
attribute String name;
attribute Unsigned Short faculty_id[6];
attribute Long soc_sec_no[10];
attribute Address address;
attribute Set<string> degrees;
relationships
operations
};
```

```
interface Professor: Person
(           extent professors
      keys faculty_id, soc_sec_no): persistent
{
      attribute String name;
      attribute Unsigned Short faculty_id[6];
      attribute Long soc_sec_no[10];
      attribute Address address;
```

```
        attribute Set<string> degrees;
        relationship Set<Student> advises inverse Student::advisor;
        relationship Set<TA> teaching_assistants inverse TA::works_for;
        relationship Department department    inverse Department::faculty;
        operations
};
```

4. 物件查詢語言 Object Query Language

OQL 的目的是提供陳述的查詢，存取物件資料庫中的物件，支援物件資料庫中的類別所定義的應用的物件模型。其是 ODMG 的標準物件查詢語言，可支援 ODMG 的物件模式。ODL 可與 SQL 相容，但也提供一些 SQL 不支援的功能。在 ODL 中允許功能分解，使查詢更有效率。

OQL 包含了四種結構，分別為 set、bag、list、array，分述如下：

Set

Given expressions $\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$ of type T , the expression

set（$\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$）

define a set that contains the elements defined by $\mathbf{expr_i}$ expressions.

Bag

Given expressions $\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$ of type T , the expression

bag（$\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$）

define a bag that contains the elements defined by $\mathbf{expr_i}$ expressions.

List

Given expressions $\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$ of type T , the two expressions

list（$\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$）

（$\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$）

define a list that contains the elements defined by $\mathbf{expr_i}$ expressions.

Array

Given the expressions $\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$ of type T , the expression

array（$\mathbf{expr_1,expr_2,\ldots\ldots,expr_n}$）

define an array that contains the elements defined by $\mathbf{expr_i}$ expressions.

查詢(Queries)是一個查詢由定義的查詢運算式集合所組成，此查詢運算式必須為合法之運算式，並且不可為遞迴（一個合法的運算式有可能包含其他運算式而形成遞迴）。以下範例為定義一個名字為 Tom 的學生的集合，並取得他們的 student_id。

Example:

Define tom as select distinct x from x in Students where x.name=”Tom”;

Select distinct x.student_id from x in tom

假設 q 是一個查詢的名字，而 e 是查詢運算式，則 *define q as e* 是一個查詢定義運算式(Query Definition Expressions)。以下範例敘述定義一個名為 Does 的查詢，他會傳回 bag 形態包含名字是 Doe 的所有學生。

Example:

Define Does as select x from x in Student where x.name =”Doe”

在基本運算式(Elementary Expressions)中，假如 x 是一個變數，那麼 x 是一個運算式。假如 a 是一個 atom(如數字….)，那麼 a 是一個運算式。假如 *define q as e* 是查詢定義運算式，那麼 q 是一個運算式。以下用範例解釋

Example:

27

此查詢傳回 27。

Example:

nil

此查詢傳回 nil。

Example:

Does

此查詢傳回名字是 Doe 的所有學生。

在算術運算式(Arithmetic Expressions)中，假設 e 是一個運算式，<op>是一個合法的 operation 對 e 來說，那麼 *<op> e* 是一個運算式。假設 e1 和 e2 為運算式，<op>是一 operation，那麼 *e1<op>e2* 為一運算式。以下用範例解釋

Example:

not (true)

傳回 false。

Example:

Count(Stdents) – Count(TA)

傳回學生數量和 TA 數量的差。

在聚集運算式(Collection Expressions)中，又區分為 Universal Quantifications、Existential Quantification、Membership Testing、Select From Where、Sort-by、Unary Set Operators、Group-by。以下分別說明

- Universal Quantifications

假設 x 是一個變數名，e1 和 e2 是運算式，那麼

for all x in e1:e2

是運算式。假如對於 collection e1 所有的元素均滿足 e2，則此敘述會傳回 true。

Example:

For all x in Students: x.student_id >0

假如所有 student_id 為正值，則此敘述會傳回 true。

- Existential Quantification

假設 x 是一個變數名，e1 和 e2 是運算式，那麼

exists x in e1:e2

是運算式。假如對於 collection e1 所有的元素中至少有一個元素滿足 e2 條件式，則此敘述會傳回 true。

Example:

Exists x in Doe.takes: x.taught_by.name = “Turing”

假如 Doe 至少有一門課是被 Turing 教的話會傳回 true。

- Membership Testing

假設 e1 和 e2 為運算式，e2 是一個 collection，e1 的 type 和 e2 的元素的 type 相同，那麼 *e1 in e2* 是一個運算式。假如 e1 屬於 e2 那麼會傳回 true。

Example:

Doe in TA

假如 Doe 是 TA，則傳回 true。

- Select From Where

假設 e, e’, e1, e2, ….., en 是運算式，且 X1, X2,……,Xn 是變數名，那麼

select e from X1 in e1, X2 in e2,…..Xn in en where e’

select distinct e from X1 in e1, X2 in e2,…..Xn in en where e’

是運算式。

Example:

```
Select couple(student: x.name, professor: z.name)
    From x in Students,
        y in x.takes,
        z in y.taught_by
where z.rank = “full professor”
```

- Sort-by 操作

假設 e, e1, e2, ….., en 是運算式，那麼 *sort x in e by e1, e2, ..., en* 是運算式。

Example:

Sort x in Persons by x.age, x.name

將 Persons 用年齡排序之後，再用姓名來排序，傳回 list。

- Unary Set Operators

假設 e 是一個運算式，它代表了一個聚集(collection)，且 <op>代表｛min，max，count，sum，avg｝其中之一，那麼 *<op>(e)*為運算式。

Example:

Max(select x.salary from x in Professors)

這個敘述會傳回 Professors 中最大的薪資。

● Group-by

假設 e 是一個運算式，它代表了一個聚集(collection)，P1, P2,…..,Pn 是屬性名，e1, e2, ….., en 是 x 的運算式，P'1, P'2,…..P'm 是屬性名，e'1, e'2,…e'm 是運算式對每個分割區(partition)而言，那麼

group x in e by(P1:e1, P2: e2, …, Pn:en)

group x in e by(P1:e1, P2: e2, …, Pn:en) with (P'1:e'1, ……, P'm: e'm)

均是運算式。

對於第一個式子而言，就是將 e 依照各個條件式，將之分成數個不同的分割區，每個分割區代表一物件的集合。

Example:

```
group x in Employees
by( low: x.salary < 1000,
    medium: x.salary >= 1000 and x.salary < 10000,
    high: x.salary >= 10000)
```

這個傳回一個集合包含三個元素，每一個元素會有一個叫 partition 的屬性，其表示屬於這個元素分類的 employees 集合。所以傳回的形態如下：

set<struct(low: boolean, medium: boolean, high: boolean, partition: set<Employee>) >

第二個運算式而言，主要是加強第一個運算式，能夠對分類後的分割區的結果做運算。

Example:

```
group e in Employees
by (department: e.deptno)
with (avg_salary: avg(sellect x.salary from x in partition))
```

這個敘述傳回一個集合：包含部門的編號和這個部門員工薪資的平均。其集合的形態如下：

set<struct(department: integer, avg_salary:float)>

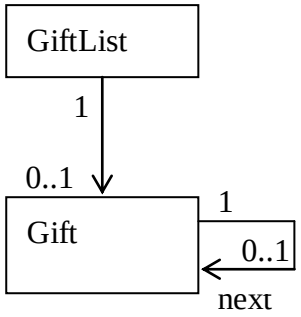
在運算元優先順序（Operate Precedence）部分，見表二

表二　　運算元優先順序

Operator（in Order of Precedence）						
（ ）	[ ]	.	－＞			
not	-（unary）	+（unary）				

in						
*	/	mod	intersect			
+	−	union	except	||		
<	>	<=	>=	<some	<any	<all
=	!=	like				
and	exist	for all				
or						
..	:					
,						
(identifier)						
order by						
having						
group by						
where						
from						
select						

以下用禮物列表管理當例子，整合以上所敘述的物件查詢語言：



物件模式由兩部份組成：GiftList（一組人的模型和他們會收到的禮物，包含姓名、預算等）和 Gift（包括收禮者的姓名、禮物的描述和價格）。

1. Require Gift and GiftList to make them persistence-capable

 # include <odmg.h> *// vendor provide，contains declarations of ODMG classes*

class Gift：**public d_object**｛ *// 只有來自基本類別 d_Object 的類別才可以持久*
friend class GiftList;
friend ostream & operator <<（ostream & , const GiftList &）; // output the values of instances

```
public：
        Gift（const char *fn, const char *ln, const char *gift, unsigned long c）;
        Gift（）;
friend ostream & operator <<（ostream &, const Gift &）;
friend instream & operator >> （istream &, Gift &）;    // read initialization data
private：
        char            first_name[16];
        char            last_name[16];
        char            gift_descry[30];
        unsigned long   cost;
        d_Ref<Gift>     next;   // reference the next element on the list
};


class GiftList：public d_object｛
public：
                GiftList（const char *n, unsigned long b）;
                ~GiftList（）;
friend ostream & operator <<（ostream &, const GiftList &）;
                void addGifts（istream &）;
private：
                char            name[20];
                unsigned long   budget;
                d_Ref<Gift>     gifts;  // 在列表中第一個元素的位置
};
```

2. creates a persistent instance of GiftList and gives it a name（使用 d_Database & d_Transaction）

```
# include <odmg.h>

d_Database db;
const char * const db_name=”Gifts”;

int
main（int argc, char *argv[ ]）
 {
    db.open（db_name）;         // opens the named database
    d_Transaction tx;
```

```
    tx.begin（）;                    // begins a database transaction

    char *name = argv [1];
    unsigned long budget = atoi（argv[2]）;

    GiftList *giftlist=new（&db, “GiftList”）GiftList（name, budget）;
    db.set_object_name（giftlist, name）;

    tx.commit（）;          // commits the transaction
    db.close（）;           // closes the database
    return 0;
}
```

3. access a GiftList instance by name and print it to the standard output stream

```
# include <odmg.h>

d_Database db;
const char * const db_name=”Gifts”;
int
main（int argc, char *argv[ ]）
 {
    db.open（db_name）;        // opens the named database
    d_Transaction tx;
    tx.begin（）;              // begins a database transaction

    char *name = argv [1];
    d_Ref <GiftList > giftlist = db.lookup_object（name）;
    if（giftlist.is_null（））{
        cerr << “No gift list with the name” ;
        cerr << name << endl;
        return –1;
     }
    cout << * giftlist;   // subsequent transactions

    tx.commit（）;          // commits the transaction
    db.close（）;           // closes the database
    return 0;
}
```

5. 語言的繫結 Language Bindings

在 ODMG 標準中的 Java、C++與 Smalltalk 語言繫結是用來定義物件操作語言 (OMLs)。ODMG 擴充了這些程式語言使其能支援永續性的物件。此物件操作語言(OML)使系統發展者能夠在單一的語言環境中工作，而不需使用各別的資料庫語言。

在 ODMG 標準中的 Java 語言繫結是用來定義物件操作語言(OMLs)。ODMG 擴充了這些程式語言使其能支援永續性的物件(Persistent Objects)。支援 OQL、瀏覽(Navigation)與交易(Transactions)。物件操作語言(OML)使系統發展者能夠在單一的語言環境中工作，而不需使用各別的資料庫語言。

C++繫結提供透通語言的延伸，以使其提供建立、命名、操作與刪除物件。C++繫結允許可永續性的類別以繼承方式來建立。

Smalltalk 繫結能在 Smalltalk 中儲存、恢復與修正永續性物件的能力。包含一個用來啟動 OQL 和程序的機制，以在資料庫上交易和運作。可到達性永續的功能 被資料庫中其他永續性物件參考時，會自動變成永續，而當它不再被參考時，會被垃圾收集機制所收集。

6. 結論

隨著企業經營對網路的應用與發展越來越依賴，傳統關聯式資料庫管理系統的不足，開啟物件導向資料庫管理系統越形重要的影響力。於 2006 年 OMG 在 Needham, MA 開會決定發展第四代的物件導向資料庫標準。為了讓物件導向資料庫的導入能夠更順暢，並受到更廣泛的應用，OMG 也成立了 ODBT 小組來主導開發更高階的技術，並由 Michael P. Card (來自 Syracuse Research Corporation) 和 Char Wales(來自 Mitre Corporation)來主持。相關的詳細內容與技術進展，可以至 ODBMS.ORG(http://www.odbms.org/odmg_ng.html)做更進一步的了解。

參考文獻：

1. Cattell, R.G.G., Object Data Management, Addison Wesly Co., 1994.
2. Cattell, R.G.G., ODMG 2.0, Morgan Kaufmann Co., 1997.
3. Cattell, R.G.G. et al, The Object Data Management Standard: ODMG 3.0. 2000
4. Dr. Dobbs, ODMG 2.0：An Overview., 1997
5. Doug Barry, Speaking of ODMG and Java., 1998
6. Doug Barry, ODMG 2.0：A Standard for Object Storage., 1998
7. Doug Barry, There's an ODMG Database in your future., 1998
8. David Jordan, C++ Object Databases-Programming with the ODMG Standard., 1998
9. Kim W., Modern Database Systems, Addison Wesly Co., 1995.
10. M.Roantree , J.B.Kennedy , P.J.Barclay, Providing views and closure for the object data management group object model., 1999
11. Stonebraker, M. Object Relational Database Systems - The Next Great Wave Morgan Kaufmann Co., 1996.

D964020001 鄭光廷的心得報告

我一直覺得資料庫與網路，是兩門資訊管理歷久不衰且非常重要的課程，尤其是資料庫更是有其不可取代性。資訊管理不外乎針對資料、資訊與知識進行有效的管理，而資料庫就是其最基礎的資訊科技。利用資料庫有效的運作，能促進其他應用程式的效能，更能發揮資訊科技的特色。這堂課我覺得除了能學習到一般資料庫所必須要了解的概念外，期中考後對於資料庫的效能部份，是這堂課醉與眾不同的地方，也是許多學習資料庫最精華的地步。

另外，在這堂課讓我印象最深刻的是要報告第三篇 paper，因為本身並不是交易管理領域，所以看第一遍的時候，真的是霧裡看花，問了學長也還是不是很懂文章中的內容，直到第二遍第三遍慢慢了解該領域的專有名詞後，才終於在報告前一天恍然大悟，還記得當剎那真的是雀躍不已，也很高興能有機會與同學分享這篇文章所要傳達的知識。

最後這邊要謝謝老師給我對於網路的訓練，每週快三小時的影音檔加上自行閱讀課本，與老師上課的補充，讓我對於資料庫的每一章節都能很清楚的了解。真的謝謝老師，雖然我的考試成績或許不是頂好，但這堂課真的學到許多東西，謝謝。

D954020011 陳慧靜的心得報告

想對資料庫管理系統有更進一步的了解，是帶領我踏入資訊管理領域的主要驅動力。當初在業界做資料庫管理系統的 coding 時，工作常常因為對資料庫管理系統的不了解而受挫。在 2001 年接觸中山網大時，老師在資料庫管理系統專題的課程中對學生的訓練，讓我受益良多，也印象非常深刻。

經過研究所的洗禮，到現在念博士班，在來上一次老師的資料庫管理系統，有很多不一樣的感受，且有更多的收穫，對資料庫的了解又更上一層。其實人偶爾也有偷懶的時後，即使到了博士班階段。老師的作業、專題、課後練習的安排，督促我隨時要找時間預習課程，並與一起上課的其它博士生討論。資料庫有很多觀念是相當抽像的，常常當我看完課本後還是只能了解部分。透過老師教學的影音檔，幫我了解了更多，在跟同學討論，及上課其它作業的討論，通常可以讓我有更詳盡的了解。受益良多。

規劃資料庫管理系統有時需要更縝密的思考以獲得更完善的規畫結果。這些也需要靠平常的練習才能更熟練，在短時間有完整的思考。在考試的時後，會覺得時間過得太快。在這樣有限的時間內，要對考試題目提出的要求座完整的思考著實不易。如果考試的時間可以再長一點，我想學生的表現可以更好一些。有時後沒有完全答對一個題目，是因為在這樣的壓力限制下，而受到影響。

一學期的資料庫管理課程，不管在理論上或是操作上都讓我受益匪淺，感謝老師辛苦的安排課程與講解，並抽空改那麼多的作業。