**TOPIC #8C**

**CHUẨN NGÔN NGỮ - THIẾT KẾ CSDL HƯỚNG ĐỐI TƯỢNG**

**Object Database Standards, Languages, and Design**

1. TỔNG QUAN VỀ MÔ HÌNH ĐỐI TƯỢNG CỦA ODMG

Mô hình đối tượng ODMG là mô hình mà ngôn ngữ định nghĩa đối tượng (ODL) và ngôn ngữ truy vấn đối tượng (OQL) là nền tảng. Mô hình này cung cấp các kiểu dữ liệu, hàm dựng kiểu và những khác niệm khác có thể được sử dụng trong ODL để xác định lược đồ CSDL đối tượng. Vì thế, mô hình ODMG là mô hình dữ liệu chuẩn cho CSDL hướng đối tượng, giống như SQL mô tả mô hình dữ liệu chuẩn cho CSDL quan hệ.

* 1. **Object và Literal**

Object và literal là các yếu tố cơ bản của mô hình đối tượng. Object có cả ID nhận biết đối tượng và giá trị, trong khi literal chỉ có giá trị mà không có ID. Giá trị của đối tượng có thể có cấu trúc phức tạp và có thể thay đổi. Tuy nhiên giá trị của literal là cố định và không thay đổi, có thể có kiểu cấu trúc.

* Object có 4 đặc điểm:
  + Identifier: là định danh duy nhất trong hệ thống (OBJECT\_ID).
  + Name: duy nhất trong một CSDL hay một chương trình. Tên của đối tượng là tùy ý. Thông thường, những tên này được sử dụng như là một entry points trong CSDL; bằng cách xác định các đối tượng này theo tên, người sử dụng có thể định vị được các đối tượng khác được tham chiếu từ những đối tượng này.
  + Lifetime: cho biết đây là *persistent object* (database object) hay *transient object* (đối tượng trong quá trình thực thi chương trình và sẽ biến mất sau khi chương trình kết thúc).
  + Structure: cho biết một đối tượng được được xây dựng bằng hàm dựng kiểu như thế nào và nó có phải là đối tượng nguyên tử hay bộ sưu tập (atomic hay collection).
* Literal:
  + Có giá trị nhưng không có ID.
  + Có ba kiểu dữ liệu:
    - Atomic: đã được định nghĩa, giá trị kiểu dữ liệu cơ bản. (short, long, float, boolean, char…)
    - Collection: một bộ sưu tập (array…) của các giá trị hay đối tượng nhưng không có Object\_id. (set<T>, bag<T>, list<T>, array<T> với T là kiểu của các đối tượng hay giá trị, Dictionary<K,V> với K là khóa, V là giá trị)
    - Structured: các giá trị được xây dựng bởi các hàm dựng kiểu (date, các biến kiểu struct …)
* Sau đây là một ví dụ đơn giản về mô hình đơn giản. Mô hình ODMG sử dụng từ khóa interface để định nghĩa một lớp. Trong thực tế, từ khóa *interface* thích hợp hơn bởi vì nó mô tả interface của các loại đối tượng, cụ thể là các thuộc tính, mối quan hệ và các phương thức. Chúng được tạo ra nhằm mục đích phục vụ cho việc định nghĩa các phương thức có thể được **kế thừa (inherited)** bởi các đối tượng do người dùng định nghĩa cho các ứng dụng riêng biệt.

interface Object {

…

boolean same\_as (in Object other\_object);

Object copy();

void delete();

};

interface Date : Object {

enum Weekday

{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

enum Month

{January, February, March, April, May, June, July, August, September, October, November, December};

unsigned short year();

unsigned short month();

unsigned short day();

…

boolean is\_equal (in Date other\_Date);

boolean is\_greater (in Date other\_Date);

…

};

Interface Collection : Object {

…

exception ElementNotFound {any element};

unsigned long cardinality();

boolean is\_empty();

…

boolean contains\_element (in any element);

void insert\_element (in any element);

void remove\_element (in any element) raises(ElementNotFound);

Iterator create\_iterator(in boolean stable);

…

};

interface Iterator {

exception NoMoreElements();

…

boolean is\_stable();

boolean at\_end();

void reset();

any get\_element() raises(NoMoreElement);

void next\_position() raises(NoMoreElement);

…

};

interface Set : Collection {

Set creation\_union(in Set other\_set);

…

boolean is\_subset\_of(in Set other\_set);

…

};

interface Bag : Collection {

unsigned long occurrences\_of(in any element);

Bag create\_union(in Bag other\_bag);

…

};

* Các phương thức được sử dụng cho các đối tượng thông qua dấu chấm.

Vd: trong lớp Object để sử dụng phương thức để so sánh với đối tượng p

**o.same\_as(p)**

hay sao chép đối tượng cho đối tượng p

**p = o.copy()**

* Ngoài ra có thể sử dụng dấu [->]: **o->same\_as(p)** hay **o->copy()**
  1. **Xây dựng Interface cho tập các đối tượng**

Một đối tượng bộ sưu tập kế thừa những phương thức và đặc điểm cơ bản từ Collection interface, ví dụ như:

* O.cardinality() trả về số phần tử trong một bộ.
* O.is\_empty() trả về true nếu bộ O là rỗng.
* O.insert\_element(E) thêm một phần tử E vào bộ O.
* O.remover\_element(E) xóa một phần tử ra khỏi bộ O.
* O.contains\_element(E) trả về true nếu bộ O có chứa phần tử E.
* I = O.create\_iterator() tạo một bản sao đối tượng I từ đối tượng O.
  1. **Atomic (User-Defined) Objects**
* Trong phần này, chúng ta sẽ biết cách tạo lớp đối tượng cho các đối tượng nguyên tử (*atomic object*). Trong ODL, chúng ta sẽ sử dụng từ khóa **class.**
* Trong mô hình đối tượng, bất kì đối tượng người dùng tự định nghĩa mà không phải là đối tượng bộ sưu tập được gọi là đối tượng nguyên tử (**atomic object**). Ví dụ, trong CSDL UNIVERSITY, ta có thể định nghĩa một lớp đối tượng cho các đối tượng Student.
* Hầu hết các đối tượng là các đối tượng có cấu trúc (structured object). Ví dụ, một đối tượng Student sẽ có cấu trúc phức tạp với nhiều thuộc tính, mối quan hệt và phương thức, nhưng vẫn được xem là nguyên tử vì nó không phải là một bộ sưu tập.
* Một đối tượng nguyên tử được định nghĩa như một lớp bằng cách xác định các đặc điểm (**properties**) và các phương thức (**operations**).
* VD:

**class** Employee

( **extent** all\_employees **key** ssn)

{

**attribute** **string** name;

**attribute** **string** ssn;

**attribute** **date** birthdate;

**attribute** **enum** Gender{M, F} sex;

**attribute** **short** age;

**relationship** Department works\_for **inverse** Department::has\_emps;

void reassign\_emp(**in** **string** new\_dname) **raises**(dname, valid);

};

**class** Department

( **extent** all\_departments **key** dname, dnumber)

{

**attribute** **string** dname;

**attribute** **short** dnumber;

**attribute** **struct** Dept\_Mgr {Employee manager, **date** startdate}

mgf;

**attribute** **set**<**string**> location;

**attribute** **struct** Projs {**string** projname, **time** weekly\_hours} projs;

**relationship set**<Employee> has\_emps **inverse**

Employee::works\_for;

**void** add\_emp(**in** **string** new\_enamel **raises**(ename\_not\_valid);

**void** change\_manager(**in** **string** new\_mgr\_name; **in** **date** startdate);

}

* + Properties:
    - Thuộc tính (attributes):
      * Một thuộc tính mô tả một số khía cạnh nào đó của đối tượng. Các thuộc tính có giá trị, thường là các literal có giá trị đơn hay cấu trúc phức tạp và được lưu trong đối tượng. Tuy nhiên, các giá trị thuộc tính có thể là Object\_Ids của các đối tượng khác. Giá trị thuộc tính còn được xác định thông qua các phương thức được sử dụng để tính giá trị thuộc tính.
      * VD: trong ví dụ trên, các thuộc tính của lớp Employee là ***name***, ***ssn***, bir***t***hdate, ***sex*** và ***age***. Các thuộc tính của Department là ***dname***, ***dnumber***, ***mgr***, ***locations*** và ***projs***. Các thuộc tính ***mgr*** và ***projs*** của ***Department*** có cấu trúc phức tạp và được định nghĩa thông qua ***struct***. Vì thế giá trị của ***mgr*** trong mỗi đối tượng ***Department*** sẽ có hai phần: ***manager*** – có giá trị là ***Object***\_Id tham chiếu từ đối tượng E***m***ployee, là người quản lý ***Department***, và ***startdate*** có giá trị là ***date***. Thuộc tính vị trí của ***Department*** được định nghĩa thông qua kiểu ***set***, vì mỗi đối tượng Department có thể có nhiều vị trí.
    - Quan hệ (Relationships):
      * Một quan hệ cho biết hai đối tượng trong CSDL có quan hệ với nhau. Trong mô hình ODMG, chỉ có mối quan hệ nhị phân (mối quan hệ đơn) và mỗi mối quan hệ được biểu diễn bằng một cặp tham chiếu (**pair of inverse references**) thông qua từ khóa ***relationship***.
      * VD: trong ví dụ trên, có một một quan hệ liên quan giữa Employee với Department là nơi nhân viên làm việc – mối quan hệ ***works\_for*** của Employee. Ngược lại, mỗi Department có quan hệ với một tập các Employee làm việc trong Department - mối quan hệ ***has\_emps*** của Department.
      * Từ khóa ***inverse*** xác định mối quan hệ đơn tương ứng ngược chiều. Bằng cách xác định mối quan hệ ngược này, hệ CSDL có thể duy trì tính toàn vẹn tham chiếu của mối quan hệ một cách tự động. Ví dụ, nếu giá trị của works\_for cho Employee e tham chiếu Department d, thì giá trị của has\_emps của Department d phải chứa một tham chiếu đến e trong tập tham chiếu đến Employee. Nếu người thiết kế CSDL muốn có mối quan hệ thể hiện trong một hướng – *only one direction*, thì nó phải được tạo như một thuộc tính hay phương thức. Một ví dụ là giá trị *manager component* của thuộc tính ***mgr*** trong Department.
  + Operations:
    - Mỗi đối tượng có thể có một số phương thức, mỗi phương thức có tên, các kiểu đối số, giá trị trả về. Tên của phương thức là duy nhất trong mỗi lớp đối tượng, nhưng vẫn có thể đặt tên giống cho các phương thức trong các lớp đối tượng khác nhau. Phương thức còn có thể xác định tên của trường hợp ngoại lệ (**exception**) có thể xảy ra trong quá trình thực thi phương thức. Ví dụ, trong lớp Employee có phương thức ***reassign\_emp*** và lớp Department có hai phương thức ***add\_emp*** và ***change\_manager***.
  1. **Interfaces, Classes và Inheritance**

Trong mô hình ODMG, có hai khái niệm tồn tại để xác định các kiểu đối tượng: **interface** và **class**. Ngoài ra, còn có hai kiểu của mối quan hệ kế thừa. Trong phần này, chúng ta sẽ thảo luận sự giống và khác nhau giữa những khái niệm này. Dựa vào các thuật ngữ ODMG, ta sử dụng từ **behavior** để thể hiện cho phương thức *operations*, và **state** để thể hiện cho các đặc tính *properties* (thuộc tính và mối quan hệ).

* **Interface** mô tả hành vi trừu tượng (**abstract behavior**)của một loại đối tượng, cho biết phương thức của đối tượng đó. Mặc dù interface có thể có các đặc tính như thuộc tính và phương thức, nhưng các đặc điểm này sẽ không được kế thừa. Interface còn có tính chất **noninstantiable** – có nghĩa là không thể tạo các đối tượng tương ứng từ định nghĩa interface. Được dùng chủ yếu cho việc xác định các phương thức trừu tượng có thể được kế thừa bởi các lớp và các interface khác, được gọi là **behavior inheritance** và sử dụng kí hiệu **“:”**. Vì thế trong mô hình đối tượng ODMG, behavior inheritance yêu cầu **supertype là interface**, **subtype là class hoặc interface**.
* Class mô tả các hành vi và đặc điểm trừu tượng của một kiểu đối tượng, có tính chất **instantianle –** có nghĩa là có thể tạo đối tượng riêng rẽ tương ứng từ định nghĩa class. Một loại quan hệ kế thừa khác gọi là **EXTENDS** và được xác định bằng từ khóa **extends**, được dùng để **kế thừa cả state và behavior** giữa các class. Trong sự kế thừa **EXTENDS**, cả supertype và subtype đều phải là class. Đa kế thừa thông qua EXTENDS không đượng chấp nhận. Tuy nhiên, đa kế thừa được cho phép cho kế thừa hành vi (behavior) thông qua dấu **“:”**. Vì thế, interface có thể kế thừa hành vi từ một số interface khác. Một class cũng có thể kế thừa hành vi từ một số interface khác thông qua dấu “:”, ngoài ra có thể kế thừa behavior và state từ một class khác thông qua EXTENDS.
  1. **Extents, Keys và Factory Objects**
* ***Extent:*** Trong mô hình ODMG, người thiết kế CSDL có thể thêm phần mở rộng **extent** cho bất kỳ loại đối tượng nào được định nghĩa bằng **class**. Phần mở rộng được đặt tên, và sẽ chứa tất cả các thể hiện đối tượng của lớp. Vì thế, phần mở rộng được xem là một tập các đối tượng chứa tất cả các thể hiện của lớp. Trong ví dụ phần 1.3 ta thấy các lớp ***Employee*** và ***Department*** có phần mở rộng gọi là ***all\_employees*** và ***all\_departments***. Phần mở rộng còn được sử dụng để tự động tuân thủ mối quan hệ set/subset giữa phần mở rộng của supertype và subtype của nó. Nếu hai lớp A và B có phần mở rộng là all\_A và all\_B, và lớp B là subtype của lớp A (có nghĩa là lớp B EXTENDS lớp A) thì bộ sưu tập của các đối tượng trong all\_B phải là subset của tất cả các đối tượng trong all\_A tại mọi thời điểm. Ràng buộc này tự động được tuân thủ bởi hệ CSDL.
* ***Key:*** Lớp với phần mở rộng có thể có một hoặc nhiều khóa. Một key chứa một hay nhiều thuộc tính hay mối quan hệ mà giá trị của nó chứa phải là duy nhất với mỗi đối tượng trong phần mở rộng. Trong phần mở rộng mục 1.3, lớp Employee có thuộc tính ***ssn*** là khóa, lớp Department có hai khóa: ***dname*** và ***dnumber***. Với những khóa kết hợp có nhiều thuộc tính hay mối quan hệ, các đặc tính đó được chứa trong dấu **()**.
* ***Factory Object***: một đối tượng có thể được dùng để phát sinh hay tạo các đối tượng riêng lẽ từ các phương thức của nó. Một số interface của factory object là một phần của mô hình đối tượng ODMG. Interface ***ObjectFactory*** có một phương thức đơn, ***new()***, sẽ trả về một đối tượng mới với một **Object\_Id*.*** Bằng cách kế thừa interface này, người dùng có thể tạo các factory interface cho mỗi kiểu đối tượng nguyên tử, và người lập trình có thể thực thi phương thức ***new*** một cách khác nhau cho mỗi kiểu đối tượng. Ví dụ bên dưới cho thấy một interface DataFactory, có phương thức tạo một **calendar\_date** mới, và một phương thức tạo một đối tượng mà giá trị của nó là **current\_date**… Ta có thể thấy, factory object cơ bản cung cấp phương thức dựng (**constructor operations**) cho các đối tượng mới.
* Ví dụ:

interface ObjectFactory{

Object new();

};

interface DateFactory : ObjectFactory{

exception InvalidDate{};

…

Date calendar\_date ( in unsigned short year,

in unsigned short month,

in unsigned short day)

raises(InvalidDate);

…

Date current();

interface DatabaseFactory{

Database new();

};

interface Database{

void open(in string database\_name);

void close()l

void bind(in any some\_object, in string object\_name);

Object unbind(in string name);

Object lookup(in string object\_name)

raises(ElementNotFound);

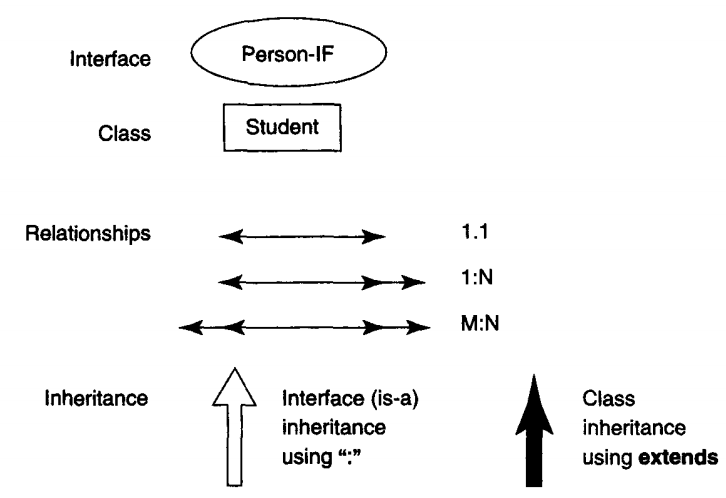
…

};

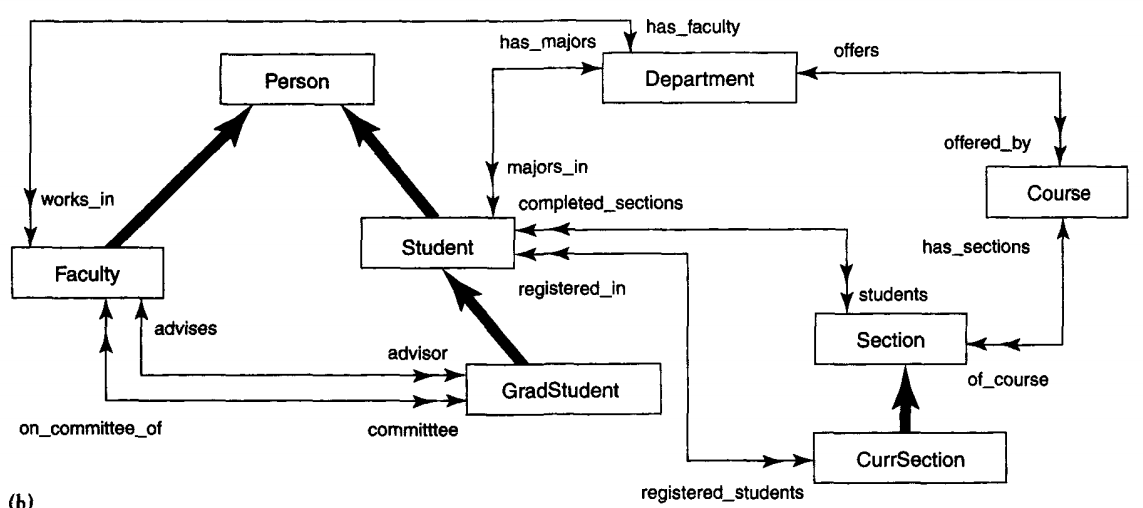
* ***Database***: bởi vì ODBMS có thể tạo nhiều CSDL khác nhau, với những lược đồ khác nhau, nên mô hình dữ liệu ODMG có các interface cho các đối tượng **DatabaseFactory** và **Database**. Mỗi CSDL có một tên CSDL riêng, phương thức **bind** có thể được dùng để gán các tên duy nhất riêng biệt vào các thể hiện đối tượng trong một CSDL cụ thể. Phương thức **lookup** trả về một đối tượng từ CSDL có tên **object\_name**, và phương thức unbind lấy tên ra khỏi các thể hiện đối tượng đã được đặt tên từ CSDL.

1. OBJECT DEFINITION LANGUAGE ODL

* ODL được thiết kế để hỗ trợ các khái niệm ngữ nghĩa của mô hình ODMG và không phụ thuộc vào bất kì ngôn ngữ lập trình nào. Mục đích chính của nó là dùng để tạo các đối tượng – class và interface. Vì thế ODL không phải là một ngôn ngữ lập trình. Người dùng có thể xác định lược đồ trong ODL mà không cần phụ thuộc vào ngôn ngữ lập trình, và sử dụng ngôn ngữ liên kết để xác định các khái niệm ODL có thể **map** với các khái niệm trong ngôn ngữ lập trình như C++, SMALLTALK và JAVA.
* Các kí hiệu trong ODL:



* Ví dụ mô hình CSDL ***UNIVERSITY***:



* Ví dụ tạo một số đối tượng bằng class trong CSDL UNIVERSITY bằng ODL:

**class** Person

(**extent** persons **key** ssn)

{

**attribute** **struct** Pname {**struct** fname,

**string** mname, **string** Inme} name;

**attribute** **string** ssn;

**attribute** **date** birthdate;

**attribute** **enum** Gender{M, F} sex;

…

**Short** age();

};

**class** Faculty **extends** Person

(**extent** faculty)

{

**attribute** **string** rank;

**attribute** **float** salary;

**attribute** **string** phone;

…

**relationship** Department works\_in

**inverse** Department::has\_faculty;

**relationship** **set**<GradStudent> advises

**inverse** GradStudent::advisor;

**void** give\_raise (**in** **float** raise);

**void** promote (**in** **string** new\_rank);

};

**class** Department

( **extent** departments **key** dname)

{

**attribute** **string** dname;

**attribute** **string** dphone;

**attribute** **string** doffice;

**attribute** **string** college;

**attribute** Faculty chair;

**ralatlonshlp** **set**<Faculty> has\_faculty **inverse** Faculty::works\_in;

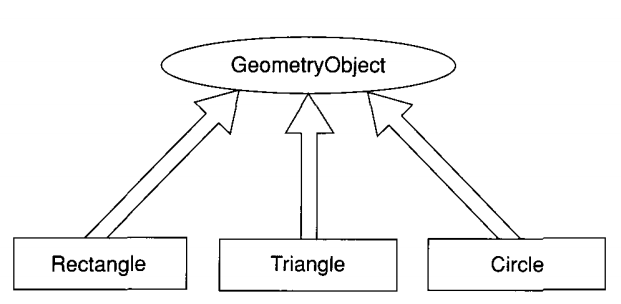
**relationship** **set**<Student> has\_majors **inverse** Student::majors\_in;

**relationship** **set**<Course> offers **inverse** Course::offered\_by;

};

…

* Ví dụ thể hiện interface và kế thừa:



* Ví dụ tạo các đối tượng trong ví dụ interface và kế thừa:

**interface** GeometryObject {

**attribute enum** Shape {Rectangle, Triangle, Circle, …} shape;

**attribute** **struct** Point{**short** x, **short** y} reference\_point;

**float** perimeter ();

**float** area();

**void** translate (**in short** x\_translation; **in short** y\_translation);

**void** rotate (**in float** angle\_of\_rotation)l

};

**class** Triangle : GeometryObject

(**extent** triangles) {

**attribute** **struct** Point{**short** x, **short** y} reference\_point;

**attribute** **short** length;

**attribute** **short** height;

attribute float orientation\_angle;

};

* Đa kế thừa (multiple inheritance) của interface bởi class được cho phép, giống như việc đa kế thừa của interface từ một interface. Tuy nhiên, với kế thừa EXTENDS (class), đa kế thừa không được sử dụng. Vì thế, một class có thể kế thừa bằng EXTNEDS từ một class.

1. OBJECT QUERY LANGUAGE OQL

OQL là ngôn ngữ truy vấn được đưa ra cho mô hình đối tượng ODMG. Nó được thiết kế để làm việc gần gũi hơn với ngôn ngữ lập trình mà liên kết ODMG đã được định nghĩa, như C++, SMALLTALK và JAVA. Vì thế, truy vấn OQL được thêm vào ngôn ngữ lập trình có thể trả về các đối tượng có kiểu thích hợp với kiểu của ngôn ngữ lập trình. Các phương thức thực thi của class có thể được viết bằng mã trong ngôn ngữ lập trình. Cú phá OQL tương tự như cú pháp của chuẩn SQL, thêm vào các khái niệm ODMG như object identity, complex object, operation, inheritance, polymorphism và relationship.

**3.1. Simple OQL Queries, database entry points, iterator variables**

* Cú pháp cơ bản của OQL là **select… from… where…**, giống như của SQL.

VD: Q0:

**SELECT** d.dname

**FROM** d **in** department

**WHERE** d.college = ‘Engineering’;

* Thông thường, entry point đến CSDL là cần thiết cho mỗi câu truy vấn, là tên đại diện cho đối tượng - ***named persistent object***. Với nhiều câu truy vấn, entry point là tên của phần **extent** của class. Tên phần mở rộng được xem là tên của một thể hiện đối tượng mà kiểu của nó là một bộ sưu tập của các đối tượng trong class. Ví dụ, tên của đối tượng **departments** là kiểu **set**<**Department**>; **persons** là kiểu **set<Person>…**
* Việc sử dụng tên mở rộng – **departments** trong Q0 – như một entry point để thể hiện một tập thể hiện của các đối tượng. Bất cứ khi nào một bộ sưu tập được tham chiếu đến trong OQL, chúng ta nên định nghĩa một biến **iterator variable – d**  trong Q0 – duyệt qua mỗi đối tượng trong bộ sưu tập. Trong nhiều trường hợp, câu truy vấn sẽ chọn các đối tượng từ bộ sưu tập, dựa trên các điều kiện đã được chỉ định trong WHERE. Trong Q0, chỉ các đối tượng d trong bộ sưu tập của **department** thỏa điều kiện **d.college = ‘Engineering’** được chọn cho kết quả. Với mỗi đối tượng được chọn d, giá trị **d.name** được gọi trong kết quả. Vì thế, kiểu của kết quả cho Q0 là **bag<string>** bởi vì kiểu của mỗi giá trị **dname** là string. Trong thực hết, kết quả của truy vấn sẽ là kiểu **bag** cho **select… from…** và kiểu **set** cho **select distinct… from…**.
* Sử dụng ví dụ trong Q0, có 3 cú pháp để định nghĩa iterator variables:
  + d **in** department
  + department d
  + department as d

**3.2. Query Results and Path Expressions**

* Kết quả truy vấn có thể là bất kì kiểu gì có thể được thể hiện trong mô hình ODMG. Một câu truy vấn không nhất thiết phải theo cấu trúc select… from… where; trong trường hợp đơn giản nhất, bất kỳ tên đại diện nào cũng là câu truy vấn, kết quả được tham chiếu đến thể hiện đối tượng đó. Ví dụ:

**Q1: departments;**

* Khi entry points được sử dụng, khái niệm **path expression** có thể được sử dụng để xác định đường dẫn đến các thuộc tính đối tượng liên quan. Path expression thông thường bắt đầu tại *persistent object name* hay tại iterator variable để duyệt qua các đối tượng trong bộ sưu tập. Theo sau tên là dấu chấm (.) và tên của thuộc tính hay quan hệ có liên quan. Ví dụ:
  + **Q2: csdepartment.chair;**
  + **Q2a: csdepartment.chair.rank;**
  + **Q2b: csdepartment.has\_faculty;**
* Trong Q2 và Q2a, kết quả trả về sẽ là những giá trị đơn, truy nhiên kết quả ở Q2b lại là một bộ sưu tập có kiểu set.
* Ví dụ trả về ranks của khoa computer science:
  + **Q3: csdepartment.has\_faculty.rank;**
  + **Q3a: select f.rank**

**from f in csdepartment.has\_faculty;**

* + **Q3b: select distinct f.rank**

**from f in csdepartment.has\_faculty;**

* Q3 và Q3a trả về kiểu bag, Q3b trả về set.
* Ví dụ trả về bộ sưu tập của các sinh viên tốt nghiệp trong khoa KHMT:
  + Q4: csdepartment.chair.advises;
  + Q4a: **select struct** (name: s**t**ruct (last\_name: s.name.Iname,

first\_name:s.name.fname),

degrees: (**select** **struct** (deg: d.degree,

yr: d.year, college: d.college)

**from** d **in** s.degrees))

**from** s **in** csdepartment.chair.advises;

* Trong câu truy vấn Q4a đã có sử dụng cấu trúc phức tạp trong kết quả trả về. Cụ thể là tên và bằng cấp của sinh viên.
* **Lưu ý:** OQL trực giao với với việc xác định path expression. (OQL is *orthogonal* with respect to specifying path expressions)
  + Tên các thuộc tính, mối quan hệ và phương thức có thể được sử dụng thay thế lẫn nhau.
  + Ví dụ: gọi điểm trung bình của tất cả sinh viên năm cuối trong khoa KHMT, với kết quả được sắp xếp theo gpa.

Q5a:

**Select** **struct** (last\_name: s.name.lname, first\_name: s.name.fname, gpa: s.gpa)

**From** s **in** csdepartment.has\_majors

**Where** s.class = ‘senior’

**Order** **by** gpa **desc**, last\_name **asc**, first\_name **asc**;

Q5b:

**Select** **struct** (last\_name: s.name.lname, first\_name: s.name.fname, gpa: s.gpa)

**From** s **in** students

**Where** s.majors\_in.dname = ‘Computer Science’ **and** s.class = ‘senior’

**Order** **by** gpa **desc**, last\_name **asc**, first\_name **asc**;

**3.3. Other Features of OQL**

**3.3.1. Views as Named Queries**

- Từ khóa **define** trong OQL dùng để xác định định danh cho **named query**.

- Tên phải là duy nhất. Nếu không, kết quả hiện tại sẽ thay thế cho named query đang tồn tại.

- Khi một định nghĩa quey được tạo, nó sẽ tồn tại cho đến khi bị xóa hoặc định nghĩa lại.

- Một định nghĩa khung nhìn có thể bao gồm các đối số.

- Ví dụ: một view gồm các sinh viên trong một khoa có minor

**Define** has\_minor(dept\_name) **as**

**Select** s

**From** s **in** students

**Where** s.minor\_in.dname = dept\_name

* Trong đó, **has\_minor** là tên của view, **dept\_name** là đối số.
* has\_minor có thể được sử dụng như câu truy vấn.
* Kết quả trả về một tập có kiểu bag các sinh viên minoring trong khoa KHMT:

**has\_minors(‘Computer Science’);**

**3.3.2. Single Elements from Singleton Collections**

- Câu truy vấn OQL trả về kiểu bộ sưu tập.

- Toán tử element của OQL có thể được sử dụng để trả về các phần tử đơn từ một bộ sưu tập đơn chứa một phần tử:

Q6: **element** (**select** d

**from** d **in** departments

**where** d.dname = 'Computer Science');

- Kiểu của kết quả: **d:Department**

**-** Nếu ***d*** là rỗng hay chứa hơn một phần tử, **exception** sẽ được thực thi.

**3.3.3.** **Collection Operators (Aggregate Functions, Quantifiers)**

**-** OQL hỗ trợ một số toán tử tập hợp có thể áp dụng cho câu truy vấn.

**-** Các toán tử tập hợp và tính toán trên bộ sưu tập bao gồm: ***min, max, count, sum, avg.***

**-** **Count** trả về kết quả có kiểu interger; những toán tử khác trả về kết quả có kiểu giống với kiểu của bộ sưu tập.

- Ví dụ toán tử tập hợp: tính trung bình GPA của tất cả các chuyên ngành cao cấp trong Business:

Q8: **avg** (**select** s.gpa

**from** s **in** students

**where** s.majors\_in.dname = ‘Business’ **and** s.class = 'senior');

**- Membership và Quantification:**

* OQL hỗ trợ các toán tử membership và quantification như sau:
  + (e **in** c) là true nếu e thuộc bộ sưu tập c.
  + (**for** **all** e **in** c: b) là true nếu tất cả các phần tử e của bộ sưu tập c thỏa b.
  + (**exists** e **in** c: b) là true nếu ít nhất có một e trong bộ sưu tập c thỏa b.
* Ví dụ: gọi tên của tất cả các sinh viên đã hoàn thành khóa học có tên là ‘Database Systems’

**select** s.name.lname, s.name.fname

**from** s **in** students

**where** 'Database Systems' **in**

(**select** c.cname

**from** c **in** s.completed\_sections.section.of\_course);

* Kiểu của kết quả: **bag<struct(string, string)>**

**3.3.4. Ordered (Indexed) Collection Expressions**

**-** Các bộ sưu tập có kiểu list hoặc array cho phép gọi các phần tử đầu tiên, cuối cùng hay ở bất kì vị trí nào.

**-** OQL cung cấp các phương thức để trích một sub-collectuon hay nối hai list lại.

**-** OQL cũng cung cấp các phương thức để sắp xếp các kết quả.

**-** Ví dụ: lấy last name của thành viên có lương cao nhất:

**Q14:** **first** ( **select** **struct**(faculty: f.name.lname, salary: f.salary)

**from** f **in** faculty

**order** **by** f.salary **desc**);

- Ví dụ: lấy 3 môn trong khoa KHMT đầu tiên dựa vào giá trị gpa:

Q15: (**select struct**( last\_name: s.name.lname,

first\_name: s.name.fname, gpa: s.gpa)

**from** s **in** csdepartment.has\_majors

**order** **by** gpa **desc**) **[0:2]**;

**3.3.5. The Grouping Operator**

**-** OQL còn hỗ trợ phương thức gom nhóm **group by**.

**-** Ví dụ: lấy trung bình GPA của các môn trong mỗi khoa có hơn 100 môn

**Select** deptname., avg\_gpa: **avg** (**select** p.s.gpa **from** p **in** **partition**)

**From** s **in** students

**Group** **by** deptname: s.majors\_in.dname

**Having** **count**(**partition**) > 100;

1. TỔNG QUAN VỀ C++ LANGUAGE BINDING

* Ngôn ngữ liên kết C++ cho biết cấu trúc OQL tương thích như thế nào với cấu trúc của C++, bao gồm:
  + Thư viện lớp C++: cung cấp các lớp và các phương thức thực thi cấu trúc ODL.
  + Ngôn ngữ xử lý dữ liệu – Data Manipulation Language (ODL/OML): cho biết các đối tượng CSDL được gọi và tính toán như thế nào trong ngôn ngữ C++, là cơ sở cho cú pháp và ngữ nghĩa cho ngôn ngữ lập trình C++.
  + Một tập các cấu trúc gọi là physical pragmas (cho phép người lập trình một số điều khiển trên các lưu trữ vật lý).
* Lớp thư viện thêm C++ vào chuẩn ODMG để sử dụng tiếp đầu ngữ d\_ cho các khai báo lớp làm việc với CSDL quan niệm. Người lập trình nên biết rằng chỉ có một ngôn ngữ đang được sử dụng, không phải là hai ngôn ngữ riêng biệt. Với người lập trình làm việc với các đối tượng CSDL trong chương trình**, lớp d\_Ref<T>** được định nghĩa cho mỗi lớp cơ sở dữ liệu T trong lược đồ. Do đó, các biến chương trình kiểu d\_Ref<T> có thể tham chiếu đến các đối tượng liên tục (persistent object) hay tạm thời (transient object) của lớp T. Để sử dụng nhiều kiểu dựng trong mô hình đối tượng ODMG như collection type, nhiều template class được định nghĩa trong thư viện. Ví dụ như lớp trừu tượng d\_Object<T> xác định phương thức được kế thừa bởi tất cả các đối tượng. Tương tự như vậy, một lớp trừu tượng d\_Collection<T> xác định các phương thức của bộ sưu tập. Những lớp này không tạo đối tượng, chúng chỉ quy định các thao tác mà được kế thừa từ các đối tượng và đối tượng collection tương ứng.
* Template class: hỗ trợ các kiểu bộ sưu tập trong ODMG gồm **d\_Set<T>, d\_List<T>, d\_Bag<T>, d\_Varray<T>, d\_Dictionary<T>**. Vì thế người lập trình có thể tạo các class của các kiểu như **d\_Set<d\_Ref<Student>>** mà giá trị của nó là một tập các tham chiếu đến đối tượng **Student**.
* C++ ODL cho phép người dùng xác định các lớp của lược đồ CSDL sử dụng cấu trúc của C++ cũng như cấu trúc được hỗ trợ bởi thư viện CSDL đối tượng. Để xác định các kiểu dữ liệu của thuộc tính, các kiểu cơ sở như **d\_Short (short interger), d\_UShort (unsigned short integer), d\_Long (long integer) và d\_Float (floating point number)** đều được hỗ trợ. Ngoài ra, còn một số kiểu literal có cấu trúc cũng được hỗ trợ như **d\_String, d\_Interval, d\_Date, d\_Time và d\_Timestamp**.
* Để xác định các mối quan hệ, tiền tố Rel\_ được sử dụng với hậu tố là tên các kiểu như **d\_Rel\_Ref<Dept, has\_majors> majors\_in.** Liên kết C++ còn cho phép tạo extents thông qua sử dụng lớp thư viện d\_Extent: **d\_Extent<Person> All\_persons (DB1).**

1. OBJECT DATABASE CONCEPTUAL DESIGN

5.1. Sự khác nhau giữa thiết kế quan niệm của ODB và RDB

* Sự khác nhau cơ bản là các mối quan hệ và kế thừa. Các phương thức trong ODB được thể hiện vì chúng là một phần trong định nghĩa của class.
* Mối quan hệ trong ODB:
  + Các mối quan hệ được xây dựng bằng các thuộc tính tham chiếu bao gồm OID của các đối tượng liên quan.
  + Đối tượng đơn hay bộ đều được cho phép sử dụng để tham chiếu.
  + Mối quan hệ nhị phân có thể được thể hiện đơn chiều hay hai chiều thông qua toán tử **inverse**.
* Mối quan hệ trong RDB:
  + Mối quan hệ giữa các bộ được xác định bằng các thuộc tính có giá trị tương thích với nhau (thông quan khóa ngoại).
  + Khóa ngoại là giá trị đơn.
  + Mối quan hệ nhiều – nhiều phải được thể hiện thông qua một quan hệ riêng (bảng).
* Kế thừa:
  + Cấu trúc thừa kế được xâu dựng trong ODB (được sử dụng thông qua dấu “:” và toán tử **extends**).
  + RDB không hỗ trợ mối quan hệ kế thừa, tuy nhiên có một số lựa chọn để chuyển mối quan hệ kế thừa sang RDB.
* Một sự khác nhau chính nữa giữa ODB và RDB là các phương thức xử lý:
  + ODB: các phương thức được xác định trong quá trình thiết kế (là một phần của lớp).
  + RDB: các phương thức bị hoãn cho đến khi thực thi.

5.2. Chuyển đổi từ lược đồ EER sang ODB

* Chuyển từ lược đồ EER sang ODB rất đơn giản vì lược đồ ODB hỗ trợ cho các mối quan hệ kế thừa.
* Khi chuyển đổi được hoàn thành, các phương thức phải được thêm vào lược đồ ODB bởi vì lược đồ EER không chứa các phương thức.
* Các bước chuyển đổi:
  + B1: tạo một lớp ODL cho mỗi thực thể EER hay phân lớp.
    - Thuộc tính đa trị:
      * List: giá trị thuộc tính được sắp xếp.
      * Bag: cho phép các giá trị trùng nhau.
      * Set: các trường hợp còn lại.
    - Các thuộc tính kết hợp: sử dụng kiểu tuple.
    - Các thuộc tính khóa là khóa của phần mở rộng extent.
    - Xác định phần mở rộng cho mỗi class và các thuộc tính khóa.
  + B2: thêm mối quan hệ hay các thuộc tính tham chiếu cho các mối quan hệ nhị phân trong các lớp ODL:
    - Các kiểu mối quan hệ: giá trị đơn cho 1 – 1 và N – 1, tập giá trị cho 1 – N và quan hệ 2 chiều cho M – N.Mối quan hệ nhị phân: thêm các đặc điểm mối quan hệ hay tham chiếu vào các lớp ODL có tham gia vào MQH. Tham chiếu mối quan hệ hai chiều nếu có với các class khác.
    - Thuộc tính mối quan hệ: tạo thông qua hàm dựng bộ (tuple constructors).
  + B3: thêm các phương thức vào mỗi lớp
    - Các phương thức không có trong lược đồ EER; các yêu cầu ban đầu phải được xem xét.
    - Các hàm dựng và hủy thích hợp phải được thêm vào.
  + B4: xác định các mối quan hệ kế thừa thông qua **extends**
    - Lớp ODL tương ứng với một sub-class trong lược đồ EER kế thừa kiểu và phương thức của super-class của nó trong lược đồ ODL.
    - Các thuộc tính khác của sub-class được thêm vào như bước 1 – 3.
  + B5: nối những kiểu thực thể yếu giống như những thực thể bình thường.
    - Các thực thể yếu không tham gia vào bất kì mối quan hệ nào có thể được biểu diễn như thuộc tính đa giá trị kết hợp (composite multi-values attribute) của thực thể chủ: set<struct<. . . >> hay list<struct<. . .>>.
  + B6: nối các category (union type) vào ODL
    - Tiến trình không phức tạp.
    - Có thể theo các nối tương tự được sử dụng cho mô hình quan hệ EER:
      * Xác định class đại diện cho category.
      * Định nghĩa mối quan hệ 1:1 giữa category và mỗi super-class của nó.
    - B7:mối quan hệ nhiều nhiều
      * Mỗi mối quan hệ được chuyển thành một lớp riêng biệt với MQH tham chiếu đến các lớp tham gia.