Chương 4: **Object Oriented Data Modeling**

1. **Object**: thuộc tính:
   * 1. *State*: thuộc tính kiểu & trị
     2. *Behavior*: cách mà object tác động & phản ứng lại
     3. *Identity*: mỗi object có 1 unique identity ngay cả khi tất cả các thuộc tính của chúng giống nhau. 1 object duy trì id trong suốt thời gian sống của nó.
2. **Object class**: tập hợp các đối tượng có cùng 1 cấu trúc & hành vi
3. **Class diagram**: mô tả cấu trúc tĩnh của 1 mô hình dự liệu hướng đối tượng: object classes, internal structures & relationships.
4. **Object diagram**: mô tả các thể hiện tương ứng với 1 diagram cho trước
5. **Operations**: là 1 hàm/dịch vụ mà sẽ đc cung cấp bởi tất cả các thể hiện của 1 class. Operations đc hiện thực thành các behavior.

Các kiểu của operations:

* + 1. *Constructor*: tạo ra 1 thể hiện mới của 1 class
    2. *Query*: lấy giá trị trạng thái (state) of object mà ko thay đổi nó
    3. *Update*: thay đổi trị của 1 trạng thái của object

1. **Associations**: là quan hệ giửa các object classes.
   1. ***Vai trò của associations***:
      1. Vai trò của 1 object trong association
      2. Kết thúc của 1 association là nơi mà nó kết hợp với 1 class
   2. ***Multiplicity*** (***lượng số***) số lượng các objects tham gia vào 1 association. Trong 1 class diagram, multiplicity cụ thể là:

Lowerbound … Upperbound

Vd: 0…\* 0…1 1…\* 1 \*

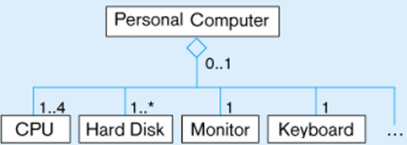
2…6 1,3,5,7

* 1. Một binary association là 2 hướng. Tên của mỗi hướng ảnh hưởng lên 1 chiều
  2. ***Biểu diễn lớp kết hợp***: 1 association có các attributes hay operations of riêng nó hay tham gia vào quan hệ với các class ≠

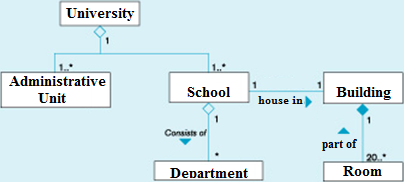
*Tên của class kết hợp*:

* + 1. Khi 1 kết hợp chỉ có các attribute nhưng ko có các operations hay ko tham gia vào các kết hợp khác => đặt tên trên đường kết hợp (association path) để nhất mạnh nó là “***association nature***”
    2. Khi 1 kết hợp có các operations của riêng nó => nên đặt tên của nó trong hình chữ nhật class để nhấn mạnh nó là “***class nature***”

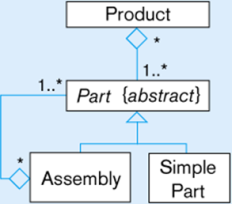
1. **Derived Attributes, Derived Associations & Derived Roles**: chúng là một & có thể nhận đc từ các attributes, associations, roles tương ứng. Phần tử kế thừa có dấu **/** phía trước tên của phần tử đó
2. ***Generalization/Specialization***:
   * 1. Subclass, superclass
     2. Common attributes, relationships & operations
     3. ***Disjoint*** >< ***Overlapping***:
        1. ***Disjoint***: nếu 1 instance of superclass là thành viên của 1 kiểu con & ko thể đồng thời là thành viên của kiểu con nào khác
        2. ***Overlapping***: nếu 1 instance of superclass có thể là thành viên của nhiều hơn 1 kiểu con.
        3. ***Thuộc tính phân biệt kiểu con***: là một thuộc tính của siêu kiểu mà trị của nó có thể xác định kiểu con
     4. ***Complete*** (total specialization) >< ***Incomplete*** (partial specialization)
        1. ***Complete***: superclass chia thành n class con, 1 instance của superclass phải là 1 trong n class con này.
        2. ***Incomplete***: superclass chia thành n class con, 1 instance của superclass có thể ko là 1 trong n class con này.
     5. ***Dynamic***: superclass chia thành n class con, 1 instance có thể thay đổi subclass theo thời gian
     6. ***Abstract class***: no direct instances
     7. ***Concrete class***: direct instances
3. **Class-level attribute**:
   * 1. Xác định 1 giá trị chung cho class, hơn là xác định 1 giá trị cụ thể cho 1 instance.
     2. Biểu diễn bằng underlining
     3. “=” là giá trị khởi tạo, giá trị mặc định
4. **Polymorphism**:
   * 1. ***Abstract Operation***: định nghĩa form hay protocol của operation, nhưng ko hiện thực nó
     2. ***Method***: hiện thực của 1 operation
     3. ***Polymorphism***: cùng operation có thể áp dụng vào nhiều class bằng nhiều cách khác nhau.
5. **Overriding Inheritance**: quá trình hiện thực các phương thức của superclass thành các hàm cụ thể của các class con
   * 1. For Extension: add code
     2. For Restriction: giới hạn phương thức
     3. For Optimization: cải thiện mã do khai thác hạn chế áp đặt bởi các lớp con
6. **Multiple Inheritance**:
   * 1. ***Multiple Classification***: 1 object là 1 thể hiện of n` hơn 1 class
     2. ***Multiple Inheritance***: 1 class kế thừa các thuộc tính từ nhiều hơn 1 supperclass
7. **Aggregation** (mối liên kết thu gộp)
   * 1. ***Aggregation***: object thu gộp quan hệ với nhiều component objects



* + 1. ***Composition***: 1 dạng mạnh của thu gộp. Object là 1 phần của object tổng thể, nó tồn tại phụ thuộc vào object tổng thể tồn tại. Tương ứng với thực thể yếu & thực thể mạnh



* + 1. ***Recursive Aggregation***: object tổng thể đc tổng hợp từ các object tổng thể khác hay object đơn vị (object nhỏ nhất)



1. **Class Diagram => RD Schema**:
   1. Chuyển 1 class trong class diagram thành table
   2. Chuyển 1 attribute của 1 class thành table column
   3. Trong trường hợp ko có khai báo id tường minh, tạo khóa chính (thông thường kiểu integer) cho table tương ứng với class
   4. Mỗi table tương ứng với 1 subclass, dung khóa chính của superclass của nó như khóa chính của nó.
   5. Mỗi table tương ứng với associate class, tạo khóa chính của nó. Sau đó thêm khóa chính của các table tương ứng với các class tham gia kết hợp vào làm khóa ngoại
   6. Với 1 component class trong liên kết thu gộp, table của nó lấy khóa chính của table tương ứng với class thu gộp làm 1 phần của khóa chính.
   7. Chuyển kết hợp 2 ngôi hay 1 ngôi thành 1 lược đồ quan hệ mạnh giống với chuyển trong mô hình EER
   8. Chuyển các operation của 1 class thành các stored procedure hay function. Các procedure/function này chứa các lệnh SQL
   9. Ko có cách tốt nhất để biểu diễn đa kế thừa trong lược đồ quan hệ. Chúng ta fải tái cấu trúc lại class diagram để tránh trường hợp này.

Chương 5: **Object-Oriented Database Development**

1. **Defining a class**:
   1. ***Class***: từ khóa định nghĩa lớp
   2. ***Attribute***: từ khóa định nghĩa thuộc tính
      1. Giá trị của thuộc tính có thể là Object identifier hay literal
      2. *Type of literal*:
         1. ***Atomic***: hằng ko thể bị hủy trong các components
         2. ***Collection***: nhiều literal hay object
            * *Set*: tập hợp ko thứ tự, ko có giá trị trùng
            * *Bag*: tập hợp ko thứ tự, có thể chứa các giá trị trùng
            * *List*: danh sách thứ tự, tất cả có cùng kiểu
            * *Array*: mãng có thứ tự, kích thước động, có thể truy xuất bằng vị trí (index)
            * *Dictionary*: chứa tuần tự cặp giá trị key-value, ko thứ tự & ko trùng nhau
         3. ***Structure***: 1 lượng cố định các phần tử, mỗi phẩn từ có thể là literal hay object.
      3. *Attribute ranges*: cho phép 1 thuộc tính đa trị (kiểu enum)
   3. ***Operations***: trả về kiểu, tên, các tham số trong dấu ngoặc ()
   4. ***Relationship***: từ khóa để định nghĩa quan hệ
      1. Chỉ có quan hệ 1 ngôi & 2 ngôi
      2. Quan hệ là 2 hướng, hiện thực thông qua từ khóa ***inverse***.
      3. 3 loại ODL relationships:
         1. ***Relationship*** xác định 1 class là many-side
         2. ***Relationship Set*** xác định class là one-side & class còn lại là many & unordered
         3. ***Relationship List*** xác định class là one-side & class còn lại là many & ordered
      4. Khi quan hệ n-n đc biểu diễn bằng class kết hợp, chúng ta có thể chuyển quan hệ đó thành 2 quan hệ: 1-n & n-1.

***class*** Student {

(***extent*** assignments

***key*** stu\_number)

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

***attribute*** ***Date*** dateOfBirth;

***attribute*** ***set***<***Address>*** address;

***attribute enum ABC*** {1, 2, 3, 4, 5, 6, 7, 8};

***relationship*** ***set***<CourseOffering> takes ***inverse***

CourseOffering∷taken\_by;

***short*** age();

***float*** gpa();

***boolean*** register\_for(***string*** crse, ***short*** sec, ***string*** term);

};

1. **Defining Structures**: user defined type with components

***struct*** Address {

***string*** address;

***string*** city;

};

1. **Key**:
   * 1. Mỗi instance of 1 class trong OODB là unique; ko cần phải khai báo id tường minh. Tuy nhiên, key cụ thể để đảm bảo ko có 2 object of 1 class có cùng giá trị of thuộc tính key
     2. Scope của unique đc giới hạn trong phạm vi của class. Do đó, trước khi chỉ định 1 key of class, bạn phải xác định phạm vi of nó
     3. ODL cũng cho phép 1 key kết hợp, vd key{ name, address }
     4. Trong ODL, chúng ta có thể định nghĩa 1 thuộc tính đa trị bằng cách dùng từ khóa ***set***.
2. **Defining Generalization**: dùng từ khóa ***extends***.

***class*** HourlyEmployee ***extends*** Employee { … };

1. **Abstract class**: dùng từ khóa abstract trước khai báo

***abstract*** class Student {

(***extent*** students

***key*** stu\_number)

***boolean*** register\_for(string crse, short section, string term);

***abstract float*** calc\_tuition(); *//abstract operation*

};

1. **Hàm thu gộp**: count, sum, avg, max, min.
2. **Các toán tử tập hợp**: Union, Intersect, Except (difference).
3. **Querying Objects in the OODB**:
   1. *Có thể viết truy vấn trả về kết quả 1 struct*:

***select distinct struct*** (name: s.name, gpa: s.gpa)

***from*** student s s.takes x

***where*** s.name = “Mary Jones”

* 1. *Find the course code, course name & course offerings which the enrollment is less than 20*:

***select distinct struct*** (code: c.crse\_code, title: c.crse\_title,

(***select*** x ***from*** c.offers x ***where*** x.enrollment<20) )

***from*** courses c

* 1. *Find names, addresses & pga’s for those students over 30 gpa more than or equal to 3.0*:

***select*** x.name, x.address, x.gpa

***from*** (***select*** s ***from*** students ***where*** s.gpa >= 3.0) ***as*** x

***where*** x.age > 3

* 1. *Find the average salary of female employees in the company*:

***select*** avg\_salary\_female: ***avg***(e.salary) ***from*** employees e

***where*** e.gender = female

* 1. *Caculates the minimum salary for each of the two groups*:

***select min***(e.salary) ***from*** employees e ***group by*** e.gender

* 1. *Groups the projects based on their priority levels*:

***select*** \* ***from*** projects p

***group by*** low: priority = low

medium: priority = medium

high: priority = high

***having sum***(***select*** x.hours ***from*** p.has x) > 50

* 1. *Find employee-ID & names of those employees who are skilled in database design or object-oriented modeling*:

***select*** emp\_id, name ***from*** employees

***where*** “Database Design” ***in*** skills ***or*** “OO Modeling” ***in*** skills

* 1. *Find those projects that do not require C++ programming skills*:

***select*** \* ***from*** projects p

***where not***(“C++ Programming” ***in*** p.skills\_required)

* 1. *Find those employees who have been assigned to at least 1 project*:

***select*** e.emp\_id, name ***from*** employees e

***where exists*** e ***in*** (***select*** x ***from*** assignments y

y.allocated\_to x)

* 1. *Find the employees who have worked only on projects starting since the beginning of 1998*:

***select*** e.emp\_id, name ***from*** employee e e.works\_on a

***where for all*** a: a.start\_date ≥ 1/1/1998

1. **Oracle Object Types**:

***create type*** Pet\_t ***as*** ***object*** (

tag\_no ***interger***,

name ***varchar2***(60),

owner ***ref*** Person\_t, *// Tham khảo đến object owner*

***member function*** set\_tag\_no(new\_tag\_no ***in interger***) ***return*** Pet\_t

)

***create type body*** Pet\_t ***as member function*** set\_tag\_no(new\_tag\_no ***in interger***) ***return*** Pet\_t

***is***

the\_pet Pet\_t ≔ ***self***;

***begin***

the\_pet.tag\_no ≔ new\_tag\_no;

***return*** the\_pet;

***end***;

***end***;

1. **Use object types**:

***create table*** pets ***of*** Pet\_t;

***insert into*** pets ***values*** (Pet\_t(23052, ‘Mambo’));

the\_pet Pet\_t = Pet\_t(1949, ‘Gozilla’);

***insert into*** pets ***values*** (the\_pet);

1. **Đặc điểm Oracle Object Types**:
   * 1. ***REF*** chỉ định thuộc tính này lưu trủ con trỏ đến 1 đối tượng khác
     2. ***Oracle*** làm REFs “đáng tin cậy & chắc chắn” hơn khóa ngoại vì REFs ko cho phép user thay đổi value, ẩn giá trị bên trong
2. **Biểu diễn các thuộc tính đa trị dùngVarray**:
   * 1. 1 cột of 1 object table có thể chứa 1 kiểu giá trị tập hợp.
     2. Thuộc tính của 1 object/entity có thể là đa trị. Để biểu diễn thuộc tính này ta dùng Varying length array (***VARRAY***) data type hay ***Nested table***.
     3. **VARRAY** là 1 tập các phần tử cùng type, có thứ tự. Mỗi phần tử có 1 index, là số thứ tự của phần tử đó trong array.
        1. ***Count***: số lượng phần tử hiện tại
        2. ***Limit***: số lượng phần tử lớn nhất mà VARRAY có thể chứa

***create type*** BeerType ***as object*** (

name char(20),

kind char(10),

color char(10) );

***create type*** StandBeersType ***as*** ***varray***(5) ***of*** BeerType;

***create table*** BeerSellers (name char(30), beers StandBeersType);

***insert into*** BeerSellers ***values*** (‘Miller’, StandBeersType(

BeerType(‘sweet’,‘ale’,‘yellow’),BeerType(‘sour’,‘larger’,’pale’)

));

* + 1. **Nested table**: khi ~ thuộc tính của object chứa nhiều objects khác. Nested table là 1 tập các phần tử data có cùng type. Nó nằm trong 1 dòng đơn của built-in type hay object type.

***create type*** BeerTableType ***as table of*** BeerType;

***create table*** manfs (

name char(30),

address AddressType,

beers BeerTableType );

***nested table*** beers ***store as*** BeerTable;

***insert into*** manfs ***values*** (

‘Anheuser’, AddressType(‘LoopRoad’, ’Boga’, ’CA’, 56789),

BeerTableType(BeerType(‘sweet’, ’ale’, ’yellow’),

BeerType(‘sour’, ’lager’, ’pale’) )

);

***insert into*** manfs ***values*** (‘Dodger’, null, BeerTableType() );

*// List the beers made by Anheuser*

*//Return a single value: BeerTableType(BeerType(‘sweet’,…’sour’..)*

***select*** m.beers ***from*** manfs m ***where*** m.name = ‘Anheuser’;

*// Find the ales made by Anheuser*

*// Dùng table để lấy nested table; query trả về 1 value: ‘sweet’*

***select*** bb.name

***from table***(***select*** beers ***from*** manfs ***where*** name=’Anheuser’) bb

***where*** bb.kind = ‘ale’;

*// Tìm name & address (city) của nhà sản xuất green beer*

***select*** m.name, m.address.city from manfs m

***where exists*** (***select*** b.\* ***from table*** (

***select*** beers ***from*** manfs ***where*** name=m.name) b

***where*** b.color = ‘green’ );

*// Liệt kê tất cả tên nhà sản xuất với tất cả thong tin về beer họ sản xuất*

***select*** m.name, b.\* ***from*** manfs m, ***table***(m.beers) b;

*// hay câu query sau cũng cho kết quả tương tự*

***select*** m.name, b.\*

***from*** manfs m, ***table***(***select*** beers ***from*** manfs ***where*** name=m.name) b;

*// updating tuples in a nested table*

***update table*** (***select*** beers ***from*** manfs m ***where*** m.name=’Anheuser’) b

***set*** b.color=’blue’

* + 1. **Varray >< Nested table**:
       1. Arrays có max size, nested table thì ko
       2. Arrays luôn dày đặt, còn nested table thì thưa thớt, do đó có thể xóa 1 phần tử bất kỳ trong nested table nhưng ko đc trong array
       3. Oracle lưu trữ array data in-line (trong cùng không gian table) nhưng lưu trữ nested table data out-of-line, hệ thống sẽ tạo ra table tương ứng với nested table.
       4. Khi lưu trử trong database, array vẫn duy trì thứ tự & cận dưới của nó, nhưng nested table thì ko.

Chương 6: **Temporal Databases**

1. **2 types of temporal tables**:
   1. ***Event tables*** which hold instant timestamps
   2. ***State tables*** which hold interval timestamps. Temporal data in state table can be represented as intervals, which are bounded by start & stop timepoints.
2. **Interval-Extended relational model**:
   * 1. Khi dữ liệu hướng thời gian ở cả state tables & event tables có thể đc biểu diễn như khoảng (intervals), chúng ta có 1 phương thức nhản interval cho mô hình DB hướng thời gian. 1 quan hệ trong database như thế gọi là history.
     2. Mỗi dòng sẽ lưu trữ chiều thời gian của 1 thực thể thong qua 1 khoảng đóng (closed interval). 2 cột cần thêm vào để biểu diễn điểm đầu & cuối của khoảng
3. **Point Type of Intervals**:
   * 1. Trong mô hình dữ liệu hướng thời gian, ~ timepoint có 1 độ mịn, là mức nhỏ nhất về thời gian of ứng dụng DB.
     2. Khi chúng ta cân nhắc 1 giá trị khoảng thời gian, [d04: d10] bao gồm điểm bắt đầu d04 & kết thúc d10
     3. Khoảng thời gian bao gồm 1 tập các điểm đc sắp xếp theo thứ tự định trước
     4. Kiểu T (dùng như point type) đc định nghĩa như sau:
        1. Total ordering
        2. Phép toán ‘first’ & ‘last’ sẽ trả về giá trị nhỏ/lớn nhất of T, phụ thuộc vào thứ tự sắp xếp trước đó
        3. Phép toán ‘next’ & ‘prior’ sẽ trả về phần tử kế tiếp & trước đó
4. **Interval operations**: I1 & I2 là các khoảng thời gian
   * 1. ***Begin***(I): return lower bound of the interval
     2. ***End***(I): return upper bound of the interval
     3. I1 ***Before*** I2: I1E < I2S
     4. I1 ***After*** I2: I1S < I2E
     5. I1 ***During*** I2: (I2S<I1S I1E≤I2E) (I2S≤I1S I1E<I2E)

I1 ***Contains*** I2: (I1S<I2S I2E≤I1E) (I1S≤I2S I2E<I1E)

* + 1. I1 ***Overlaps*** I2: I1S≤I2S I2S<I1E I1E<I2E

I1 ***Overlaped\_by*** I2: I2S≤I1S I1S<I2E I2E<I1E

* + 1. I1 ***Meets*** I2: I1E=I2S

I1 ***Meeted\_by*** I2: I2E=I1S

* + 1. I1 ***Starts*** I2: I1S=I2S I1E<I2E

I1 ***Started\_by*** I2: I1S=I2S I2E<I1E

* + 1. I1 ***Finishes*** I2: I2S<I1S I1E=I2E

I1 ***Finished\_by*** I2: I1S<I2S I1E=I2E

* + 1. I1 ***Equal*** I2: I1S=I2S I1E=I2E

Chúng ta có thể gọi ***I1 Merges I2*** nếu I1 & I2 thỏa phép toán từ ***During***→***Equal***.

1. **Phép toán Fold**: (pack, coalese)
   * 1. Các phép toán Union, Dfference, Projection & Cartesian product có thể dùng ở dữ liệu hướng thời gian. Ngoài ra còn có phép toán ***fold***. Những dòng trong quan hệ thời gian gần kề hay trùng khoảng thời gian là đối tượng xử lý của ***fold***.
     2. Fold Operation: Khi 1 quan hệ n-ary R đc fold trên thuộc tính Ai (1≤ i ≤ n) tất cả dòng trong A có thể merge lại sẽ đc Merge.
2. **Chuẩn hóa thời gian**:
   1. Tập hợp những thuộc tính thời gian khác nhau (Time-varying attributes TAVs) trong quan hệ đc gọi là đồng bộ nếu mỗi TVA có thể được kết hợp giống nhau & được áp dụng trực tiếp những giá trị nhãn thời gian trong mỗi dòng của quan hệ.
   2. Chuẩn hóa thời gian tránh đc sự thừa & bất thường khi cập nhật, truy xuất dữ liệu
3. **Temporal Query Language** (**TSQL**):
   * 1. TSQL đc thiết kế để truy vấn CSDL hướng thời gian. TSQL là tập cha của SQL, giới thiệu 1 vài ngữ nghĩa & thành phần cú pháp mới
     2. TSQL thêm vào phát biể điều kiện thời gian ***When*** & truy vấn giá trị nhản thời gian; truy vấn thông tin theo thứ tự thời gian;
     3. TSQL chỉ định miền thời gian dùng mệnh đề ***Time-Slice***.
     4. Thay đổi các hàm thu gộp & phát biểu ***Group By***.

***Modified*** [first|second|third|nth|last|distinct] item\_list

***From*** table\_name\_list

***When*** temporal\_comparison\_list

***Where*** search\_condition\_list

1. **Temporal Projection**: tương tự như projection, chỉ ≠ là áp dụng lên ~ thuộc tính ko thời gian. Cả 2 cột nhản thời gian ko thể đưa vào kết quả. Sau khi temporal projection, folding thực thi để các khoảng thời gian gần nhau kết hợp thành 1 khoảng thời gian trong quan hệ kết quả
2. **Temporal Comparation in When**:

***When*** a interval\_compare\_operator b

* + 1. [a, b] ***Before*** [c, d] iff (b<c)
    2. [a, b] ***After*** [c, d] iff (a>d)
    3. [a, b] ***During*** [c, d] iff (a≥c) & (b≤d)
    4. [a, b] ***Equivalent*** [c, d] iff (a=c) ^ (b=d)
    5. [a, b] ***Adjacent*** [c, d] iff (c-d=1) | (a-d=1) *// liền kề*
    6. [a, b] ***Ovelap*** [c, d] iff (a≤d) & (d≤d) *// chòng lắp*
    7. [a, b] ***Follows*** [c, d] iff (a-d=1) *// theo sau*
    8. [a, b] ***Precedes*** [c, d] iff (c-d=1) *// đi trước*

Vd:

***Select*** B.mgr, B.Time-Start ***From*** M A, M B

***Where*** A.eno = B.eno ***and*** A.eno = 23 ***and*** A.mgr = ‘Jones’

***When*** B.Interval ***Follows*** A.Interval

1. **Retrieval of Timestamps**:
   * 1. ***Time-Start*** & ***Time-End*** để lấy giá trị time-point; ***Interval*** để lấy khoảng thời gian.
     2. Phép toán ***inter*** (intersect) trên 2 khoảng thời gian (chúng phải overlap nhau) sẽ trả về 1 khoảng thời gian giao.

[a, b] ***inter*** [c, d] = [max(a, c), min(b, d)]

***Select*** M.eno,mgr, sal,(M ***inter*** S).***Time-Start***,(M **inter** S).**Time-End**

***From*** S, M

***Where*** S.eno = M.eno ***and*** salr < 40K

***When*** S.***Interval*** ***Overlap*** M.***Interval***;

*// Tìm time-start & lương of nhân viên có lương vượt quá 50K*

***Select first***(salr), Time-Start ***From*** S ***Where*** salr>50K;

*// List manager history of all employeesin the last 5 years*

***Select*** eno, mgr, Time-Start ***From*** M

***When Time-Slice year*** [Now-5, Now]

*// List all changes of salary during the years 1972-1978*

*// whose manager was Bradford*

***Select*** S.eno, salr, S.Time-Start ***From*** S, M

***Where*** S.eno = M.eno ***and*** mgr=’Bradford’

***When*** M.Interval ***Overlap*** S.Interval

***Time-Slice year*** [1972, 1978]

*// Find the period of time for which employee 45 wored under manager Jones*

***Select*** eno, ***Sum***(***Duration***) ***From*** M ***Where*** mgr=’Jones’ ***and*** eno=45

1. **Group by**: có thể Group by theo year, month, date, hour, minute của các thuộc tính Time-Start, Time-End.

*// Find the calendar years during which an employee made more foreign*

*// (other than U.S.) visits than domestic visits.*

***Select*** A.eno, A***.Time-Start.year From*** T A, T B

***Where*** A.eno=B.eno ***and*** A.country=’U.S.’ ***and*** B.country!=’U.S.’

***When*** A.***Time-Start.year*** = B.***Time-Start.year***

***Group By*** a.eno, A.***Time-Start.year***

***Having Count***(***Unique*** B.Time-Start)>Count(***Unique*** A.Time-Start)

*// For every employee & for every year, list the city that was visited most*

*// often in that year & the number of times it was visited*

***Select*** eno , Time-Start.year, city, ***Max***(***Count***(\*)) ***From*** T

***Group By*** eno, Time-Start.year, city

1. **Insertion of data**:

***Insert Into*** <table\_name> (<column\_name\_list>)

***Values*** <field\_name\_values>

* + 1. Khi 1 dòng mới đc đưa vào table thì những giá trị của các thuộc tính cụ thể phải đc cung cấp, trong đó có thuộc tính thời gian [Ts, Te].
    2. Khi 1 dòng mới đc insert vào table, phương thức ***Fold*** sẽ thực thi để merge các khoảng thời gian lại với nhau (nếu cần thiết)

1. **Modification of data**:

***Update*** <table\_name> ***Set*** <column\_name> = <value>

***When*** <valid-time> ***Where*** <condition>

* + 1. Chỉ có những dòng có khoảng thời gian giao với When clause mới bị ảnh hưởng
    2. Khi cập nhật 1 dòng có thể quan hệ sẽ sinh ra nhiều dòng kết quả

1. **Delete of data**:

***Delete From*** <table\_name>

***When*** <valid\_time> ***Where*** <condition>

* + 1. Chỉ có những dòng có khoảng thời gian giao với When clause mới bị ảnh hưởng

Chương: **Transaction**

1. **Concurrency**: Interleaved processing & Parallel processing
2. **Granularity of data** (độ mịn of dữ liệu): field, record, whole disk block (ngữ cảnh độc lập với độ mịn)
3. **Phép toán cơ bản**:
   1. ***Read\_item***(X):
      1. Tìm địa chỉ của disk block chứa item X
      2. Copy disk block vào buffer từ trong bộ nhớ chính (nếu nó vẫn chưa có ở đó)
      3. Copy item X từ buffer vào biến tên X trong chương trình
   2. ***Write\_item***(X)
      1. Tìm địa chỉ của disk block chưa item X
      2. Copy disk block này vào trong buffer từ bộ nhớ chính (nếu disk block này ko có ở trong buffer của bộ nhớ chính)
      3. Copy item X từ biến X of chương trình vào đúng vị trí trong buffer
      4. Lưu trữ block cập nhật từ buffer vào disk (ngay lập tức hay sau đó 1 thời gian)
4. **Tại sao lại cần thiết điều khiển truy xuất đồng thời**:
   * 1. ***The Lost Update Problem***: truy cập cùng lúc ~ items của database
     2. ***The Temporary Update (or Dirty Read) Problem***: khi write(X) bị lỗi 🡪 trả về giá trị X cũ, trong khi đó read(X) lại đọc giá trị tạm thời
     3. ***The Incorrect Summary Problem***: xảy ra khi thực thi đồng thời ko đc kiểm soát. Vd: khi đang count, sum, … thì có transaction khác đang update.
5. **Why recovery is needed ~ What causes a Transaction to fail**:
   * 1. ***A computer failure (system crash)***: phần cứng/phần mềm lỗi
     2. ***A transaction or system error***: vài operation trong transaction gây lỗi, như chia cho 0, tràn interger.
     3. ***Local errors or exception conditions detected by the transaction***: gây error do dữ liệu cho transaction ko đc tìm thấy; lập trình hủy transaction gây lỗi.
     4. ***Concurrency control enforcement***: điều khiển đồng thời quyết định abort transaction do có nhiều transaction đang ở trạng thái deadlock.
     5. ***Disk failure***: do disk head crash
     6. ***Physical problems & catastrophes***: do lỗi vật lý hay thảm họa
6. **Transaction**:
   1. Transaction là 1 đơn vị “nguyên tử” công việc, nó hoàn tất khi tất cả công việc đc hoàn tất.
   2. R***ecovery manager techniques use the following operations***:
      1. Begin\_transaction
      2. Read or Write
      3. End\_transaction
      4. Commit\_transaction
      5. Rollback (or Abort)
      6. Undo: giống Rollback nhưng chỉ áp dụng chó 1 operation
      7. Redo: this specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.
7. **The System Log**:
   * 1. ***Log*** or ***Journal***: log duy trì tất cả transaction operations đã có tác động đến những giá trị của các phần tử trong database.
        1. Các thông tin này cần thiết cho recovery khi transaction thất bại
        2. Log đc lưu trong disk, do đó ko bị tác động bở bất kỳ type, failure except for disk, lỗi do tai nạn
8. **Các thuộc tính ACID**:
   * 1. ***Atomicity*** (nguyên tử): là 1 đơn vị xử lý nguyên tử, nó thực đc thực thi khi tất cả hoàn tất
     2. ***Consistency preservation*** (duy trì tính nhất quán): thực thi chính xác of transaction phải đưa database từ trạng thái nhất quán này qua trạng thái nhất quán khác
     3. ***Isolation*** (sự cô lập): transaction làm việc update ẩn so với các transaction khác cho đến khi nó commit.
     4. ***Durability*** or ***Permanency*** (tính lâu bền): transaction thay đổi database & những thay đổi đc commit. Những thay đổi này phải ko bao giờ mất do các lỗi xảy ra sau đó
9. **Các đặc tính lập lịch dựa vào Recoverability**:
   * 1. ***Transaction schedule or history***: khi các transaction đang thực thi đồng thời trong sự xen kẻ, sắp xếp thực thi operations ở các dạng transactions khác nhau đc gọi là ***Transaction Schedule*** (or ***history***)
     2. ***A schedule*** (or ***history***) S của n transaction T1, T2, …, Tn
     3. ***Recoverable schedule***: khi ko có transaction cần đc roll back
     4. ***Cascadeless schedule***: khi mỗi transaction đọc chỉ những phần tử mà đã bị ghi bởi commited transactions
     5. ***Schedules requiring cascaded*** (tầng lớp) ***rollback***: lập lịch khi uncommitted transactions đọc 1 item từ 1 failed transaction phải bị rolled back.
     6. ***Strict Scheduled***: lập lịch trong tường hợp transaction ko thể đọc hay ghi 1 item X cho đến khi transaction cuối cùng ghi X commited
10. **Các đặc tính lập lịch dựa vào Serializability**:
    * 1. ***Serial schedule***: 1 schedule S là tuần tự nếu mỗi transaction T tham gia vào schedule, tất cả các operations of T đc thực thi liên tiếp trong schedule. Ngược lại thì gọi là ***nonserial schedule***.
      2. ***Serializable schedule***: schedule S là tuần tự hóa nếu nó tương đương với serial schedule của cùng n transactions. Là các tiếp cận đc dùng bởi hầu hết các DBMSs. Dùng lock với 2 phase locking
      3. ***Result equivalent***: 2 schedules đc gọi là result equivalent nếu tạ ra cùng trạng thái kết thúc of database
      4. ***Conflict equivalent***: 2 schedule là conflict equivalent nếu thứ tự of bất kỳ 2 conflicting operations là giống nhau trong cả 2 schedules
      5. ***Conflict serializable***: schedule S là conflict serializable nếu nó là conflick tương đương với serial schedule S’.
      6. Serializable ko giống với serial. Serializable là correct schedule, nó sẽ đưa database đến 1 trạng thái nhất quán. Nó xen kẽ 1 cách thích hợp & sẽ trả về trạng thái như trạng thái mà transaction đc thực thi tuần tự, nhưng nó đạt hiệu quả giống như thực thi đồng thời.
      7. ***Test for conflict serializability***:
         1. Xem xét 2 operations: read\_item(X) & write\_item(X)
         2. Tạo đồ thị ưu tiên (serialization graph) là đồ thị có hướng
         3. Cạnh đc tạo từ Ti đến Tj nếu 1 trong các operations của Ti xuất hiện trước 1 conflicting operation của Tj.
         4. Schedule là serializable iff đồ thị ko có vòng
      8. Dưới các ràng buộc ngữ nghĩa tuần tự, các schedules nếu ko conflict sẽ làm việc chính xác
11. **Transaction support in SQL2**:
    * 1. 1 phát biểu SQL đơn luôn là nguyên tốt
      2. Với SQL, ko có phát biểu Begin Transaction tường minh
      3. Mỗi transaction phải có 1 kết thúc statement tường minh, có thể là ***Commit*** hay ***Rollback***.
      4. ***Access mode***: (*Read Only* hay *Read*, *Write*) mặc định là *Read*, *Write*. Mức độ cô lập of *Read Uncommitted* đc định rõ, giống như *Read Only*.
      5. ***Diagnostic size n***: giá trị nguyên n xác định số điều kiện có thể đc nắm giử đồng thời trong diagnostic area.
      6. ***Isolation level*** <isolation>:
         1. Read Uncommited, Read Committed, Repeatable Read or Serializable.
         2. Mặc định là Serializable: Thực thi xen kẽ các transactions sẽ trung thành với khái niệm serializability.
         3. Tuy nhiên, nếu bất kỳ transaction nào thực thi ở lever thấp hơn thì tính serializability có thể bị vi phạm
      7. ***Dirty Read***: đọc giá trị mà đã đc ghi bở 1 transaction bị lỗi
      8. ***Nonrepeatable Read***:
         1. Cho phép transaction khác ghi giá trị mới khi transation khác đọc liên tiếp
         2. Transaction T1 có thể đọc 1 giá trị cho trước từ 1 table. Nếu sau đó transaction T2 update giá trị đó & T1 đọc giá trị đó 1 lần nữa, T1 sẽ thấy sự khác nhau về giá trị
      9. ***Phantoms***: những dòng mới đc đọc bằng cách dùng đọc cùng lúc với 1 điều kiện
      10. ***Possible violation of serializability***:

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation level** | **Dirty read** | **Nonrepeatable read** | **Phantom** |
| Read Uncommited | Yes | Yes | Yes |
| Read Commited | No | Yes | Yes |
| Repeatable Read | No | No | Yes |
| Serializable | No | No | No |

Chương: **Concurrency Control Techniques**

1. **Purposes of Database Concurrency Control**:
   * 1. Làm khả năng cô lập có hiệu lực (thông qua sự ngăn chặn qua lại) giữa các conflicting transaction
     2. Duy trì tính nhất quán của database thông qua việc duy trì tính nhất quán khi thực thi transaction
     3. Giải quyết những conflict xảy ra khi read-write & write-read
     4. Vd: trong quá trình thực thi nếu T1 conflict với T2 qua data item A, trình điều khiển đồng thời quyết định nếu T1 hay T2 nên lấy A & transaction còn lại sẽ roll-back hay wait.
2. **Two-Phase Locking Techniques**:
   1. Locking là 1 operation đc bảo mật:
      1. Quyền đc đọc data item cho 1 transaction
      2. Quyền đc ghi data item cho 1 transaction
   2. Unlocking là 1 operation mà sẽ loại ra quyền đọc/ghi từ data item
   3. Lock & Unlock là operation nguyên tử, hiện thực như các đơn vị độc lập, critical section trong OS
   4. ***Essential components***: shared (read) & exclusive (write)
      1. ***Lock manager***: managing locks on data items
      2. ***Lock table***: lock manager uses it to store the ID of transaction locking a data item, the data item, lock mode & pointer to next data item locked.
      3. Database yêu cầu tất cả transaction phải đúng chuẩn (well formed)
      4. Một transaction là well-formed nếu:
         1. Nó phải lock data item trước khi nó đọc/ghi item đó
         2. Nó ko đc lock data item đã bị locked & phải chờ đến khi item đó đc unlock
   5. ***Lock conversion***:
      1. ***Lock upgrade***: đang có quyền read lock, chuyển thành write lock
      2. ***Lock downgrade***: đang có quyền write lock 🡪 read lock
   6. ***2PL***: tất cả các locking operations (read\_lock, write\_lock) phải đứng trước unlock operation đầu tiên trong transaction.
      1. ***Conservative***: ngăn ngừa deadlock bằng cách lock hết tất cả data items cần thiết trước khi bắt đầu thực thi transaction
      2. ***Basic***: transaction lock data items tăng dần. Điều này có thể gây ra deadlock, do đó phải tiến hành thỏa thuận
      3. ***Strict***: Unlock write locks đc thực thi sau kết thúc transation (commit hay rolled-back). Đây là giải thuật 2PL chung nhất, nó ko deadlock-free
      4. ***Rigorous 2PL***: Strict 2PL + Read locks (cả Read & Write locks đc quan tâm)
3. ***Starvation*** xuất hiện khi 1 transaction cụ thể duy trì việc đợi hay restart & ko bao giờ thay đổi hơn nữa.
4. **Concurrency Control in Indexes**: Problem: khi 1 write\_lock đc yêu cầu, cây phải đc khóa từ góc

Chương: **Database Security**

1. **3 khái niệm cơ bản**:
   * 1. ***Authentication***: 1 cơ chế mà xác định 1 user là ai
     2. ***Authorization***: gán quyền mà cho phép chủ thể có quyền hợp pháp truy cập vào hệ thống hay 1 hệ thống các đối tượng
     3. ***Access Control***: 1 cơ chế bảo mật (of DBMS) để giới hạn quyền truy xuất các đối tượng of hệ thống (database)
2. **Threats**:
   * 1. Bất cứ tình huống nào hay sự kiện nào (có ý hay vô ý) có tác động bất lợi đến 1 hệ thống & tổ chức
     2. Nguy cơ tác động đến các hệ thống máy tính, các databases
3. **Data security requirements**:
   1. ***Confidentiality*** (bảo mật): 1 hệ thống bảo mật đảm bảo tính bảo mật of data. Có nghĩa là nó cho phép nhìn thấy độc lập data mà họ đc phép thấy. Bảo mật có nhiều khía cạnh khác nhau:
      1. Privacy of Communications
      2. Secure Storage of Sensitive Data
      3. Authenticated Users
      4. Granular (độ mịn) Access Control
   2. ***Intergrity*** (tính toàn vẹn) 1 hệ thống bảo mật đảm bảo dữ liệu mà nó chứa phải hợp lệ. Toàn vẹn dữ liệu nghĩa là data đc bảo vệ khỏi deletion & corruption trong khi nó thay đổi database & trong khi nó truyền trên mạng. Tính toàn vẹn có những khía cạnh:
      1. Hệ thống & điều khiển các quyền trên đối tượng truy xuất các table of ứng dụng & các lệnh of hệ thống. Do đó chỉ có user đã chứng thực mới có thể thay đổi data
      2. Toàn vẹn tham khảo là khả năng duy trì các quan hệ hợp lệ giữa các giá trị trong database, tùy theo các luật đã đc định nghĩa.
      3. Database phải đc bảo vệ để chóng lại virus mà thiết kế để làm hỏng dữ liệu
      4. Sự truyền nhận trên mạng phải đc bảo vệ khỏi deletion, làm sai lệch & nghe trộm
   3. ***Availability***: 1 hệ thống bảo mật làm data sẵn sàng chứng thực user mà ko có delay. Tấn công từ chối dịch vụ là cố gắng khóa khả năng chứng thực người dùng để truy cập & dùng hệ thống khi cần thiết
   4. ***Non-repudiation*** (sự thoái thác, ko chịu trách nhiệm) ko thể cấm điều mà người nào đó làm
4. **Threats to databases**:
   * 1. Mất tính bảo mật 🡪 phải duy trì bảo mật data
     2. Mất tính toàn vẹn 🡪 phải ngăn ngừa điều chỉnh thông tin ko hợp lệ
     3. Mất tính sẳn sàng 🡪 tránh denial of service
     4. Mất tính non-repudiation 🡪 auditing & accountability (trách nhiệm)
5. **Countermeasure** (biện pháp đối phó): bảo vệ database chóng lại các dạng nguy cơ bằng 5 biện pháp đối phó có thể hiện thực:
   1. ***Access control***:
      1. ***DAC*** (***Discretionary*** (tùy ý) ***Access Control***) cấp quyền truy xuất data dựa vào đặc tính của user & quy tắc là kiểu access cụ thể lên mỗi user đc cho phép trên mỗi object trong hệ thống.
         1. Identification: 1 user cho biết anh ta là ai
         2. Authentication: xác minh tính hợp lệ của thông tin chứng thực
         3. Điểm yếu: sự phổ biến thong tin là ko đc điều khiển 🡪 nguy cơ từ Trojan Horses.
         4. Phương thức tiêu biểu of đưa DAC vào database system là dựa vào cấp & thu hồi quyền
         5. SQL chuẩn hổ trợ DAC thong qua lệnh GRANT & REVOKE.
      2. ***MAC*** (***Mandatory*** (ủy thác) ***Access Control***):
         1. Gán quyền truy xuất data dựa trên cấp bậc of user & mức độ nhạy cảm of data.
         2. 2 nguyên tắc Bell-LaPadula: no read-up & no write-down. Bảo mật data như Integrity (toàn vẹn)
         3. Rất ít hiệu quả khi áp dụng MAC
      3. ***RBAC*** (***Role-Based Access Control***)
         1. RBAC nổi lên nhanh chóng năm 1990 như là kỹ thuật đã đc chứng minh cho việc quản lý & bảo mật trong các hệ thống thương mại lớn.
         2. Khái niệm cơ bản of nó là các quyền đc kết hợp với vai trò (role) & user đc gán một role thích hợp.
         3. Role có thể đc tạo ra bằng cách dùng lệnh Create Role & Detroy Role. Giống Grant & Revoke của DAC
   2. ***Inference control***:
      1. Suy luận là quá trình đưa ra các truy vấn & suy diễn ra thông tin mới từ kết quả trả về
      2. CSDL thống kê: tập hợp kết quả truy vấn ko tiết lộ thong tin cá nhân
      3. MLS/DBMSs: nhiều cấp ủy nhiệm bảo mật DBMS
   3. ***Flow control***
   4. ***Encryption***: Mã hóa data bằng giải thuật cụ thể để trả về data ko thể đọc đc bằng bất cứ chương trình nào mà ko có khóa giải mã
   5. ***Auditing***