Dr. Rim Samia Kaabi

Le standard de l'ODMG: OQL

1

OQL

- OQL est le standard des langages de requêtes.
- Il utilise l'ODL comme un schéma.
- Les types dans OQL sont les mêmes que ODL.
- Set(Struct) et Bag(Struct) jouent le rôle de relations.

2

Concepts nouveaux

- Expression de chemin mono-valuée
  - Séquence d'attributs ou associations mono-valués de la forme X1.X2…Xn telle que chaque Xi à l'exception du dernier contient une référence à un objet ou un litéral unique sur lequel le suivant s'applique.
  - Utilisable en place d'un attribut SQL
- Collection dépendante
  - Collection obtenu à partir d'un objet, soit parce qu'elle est imbriquée dans l'objet ou pointée par l'objet.
  - Utilisable dans le FROM

Forme des Requêtes

Select [Distinct][<type résultat>] (<expression> [, <expression>] ...)

From x in <collection> [, y in <collection>]...

Where <formule>

- Type résultat
  - automatiquement inféré par le SGBD
  - toute collection est possible (bag par défaut)
  - il est possible de créer des objets en résultat

Expressions : chemin

- Soit *x* est un objet de la classe *C*.
  1. si *a* est un attribut de *C*, alors *x.a* est la valeur de cet attribut pour l'objet x.
  2. si *r* est une relation de *C*, alors *x.r* est la valeur de ce que x pointe par *r*.

     Peut être un objet, ou un ensemble d'objets.
  3. Si *m* est une méthode de *C*, alors *x.m*(…) est le résultat de m appliquée à *x*.

5

Exemple

```
class Diplome (extent diplomes) {
    attribute string nature;
    attribute string annee;
    relationship Personne Rpersonne inverse
    Personne::Rdiplomé;
    relationship Specialite  Rspecialite inverse
    Specialite::Rdiplomeobtenu;
}
class Personne (extent Personnes) {
    attribute string nom;
    attribute string addr;
    relationship Set<Diplome> Rdipomé inverse
    Diplome::Rpersonne;
}
```

6

Exemple

```
class Specialite (extent Specialites) {
    attribute string nom;
    attribute string domaine;
    relationship Set<Diplome> RdiplomeObtenu inverse
    Diplome::Rspecialite;
}
```

7

Exemple: expression de chemin

- Soit *d* une variable de type Diplôme
  1. d.nature = la nature de l'objet d.
  2. d.Rpersonne.addr = l'adresse de la personne ayant le diplôme d.

     Note: dans ce cas, la cascade en utilisant le . est valide car d.Rpersonne est un objet et pas une collections d'objets.

8

Exemple: Utilisation illégale du .

- On ne peut pas utiliser le « . » avec une collection à gauche --- seulement avec un objet.
- Exemple (illégal), avec p une personne:

p.Rdiplome.nature

cette expression retourne un ensemble
D'objets diplomes.

9

Clause FROM

- chaque terme de la clause FROM est de la forme:

<collection> <nom d'un membre>

- la collection peut être:
  1. L'extent d'une classe.
  2. Une expression qui évalue une collection,

10

Exemple

- Liste des diplômes de Ali

```
SELECT d.nature
FROM Diplomes d
WHERE d.Rpersonne.nom = “Ali”
```

11

Astuces pour utiliser les .

- Si une expression de chemin dénote un objet, on peut utiliser les « . »
  - Exemple: d, d.nature, d.Rpersonne.nom .
- Si une expression de chemin dénote une collection d'objets, on peut l'utiliser dans la clause FROM.
  - Exemple: s.Rdiplomeobtenu .

12

Le type de résultat

- Par défaut, le type de retour d'un select-from-where est un Bag de Structs.
- Rappel Bag : {avec répétition}

13

Exemple

```
SELECT d.Rspecialite.domaine,
d.nature
FROM diplomes d, d.Rpersonne p
WHERE p.name = “Salah”
```

- type:
  Bag(Struct(domaine: string, nature: string))

14

Exemple

la recherche des noms des enseignants assurant les cours suivis par l'étudiant de numéro 136 :

Select distinct c.prof.nom
from LesEtudiants e, e.cours_suivis c
where e.n°E=136 ,

15

Renommer les champs

- Pour changer le nom d'un champs, il faut le faire précéder de deux points
- Exemple:

```
SELECT
  Diplôme:d.Rspecialite.domaine,
  d.nature
     FROM diplomes d, d.Rpersonne p
     WHERE p.name = "Salah"
```

- type:
  Bag(Struct(Diplôme: string, nature: string))

16

Opérateurs sur les collections

17

Quantificateur existentiel

- exists x in collection: condition(x)
- Exemple:
  - exists v in Employés.posséde : v.marque = "Renault«
  - Select e from LesEnseignants e where exists c in e.cours-assurés : c.cycle=3 ;

18

Quantificateur universel

- for all x in collection : condition(x)
  - Exemple:
  - for all b in Buveurs : b.age < 18

- **Select e from LesEnseignants e where for all c in e.cours-assurés : c.cycle=3 ;**

Test d'appartenance à une collection

- <élément> **in <collection>**
- Exemple:

Select e

from LesEnseignants e, LesCours c

where c.nomC = "BDA" and c in e.cours-assurés **;**

Agrégations

- AVG, SUM, MIN, MAX, et COUNT s'appliquent à toute collection (à condition d'avoir un sens).
- Exemple:

Count (Select e
From LesEnseignants e
where e.statut="prof") ;

21

Agrégations

• Pour chaque étudiant, donner son nom, le nombre total de ses diplômes, le nombre de diplômes obtenus en 2003, et la première année où il a obtenu un diplôme.

```
SELECT STRUCT( nom : e.nom,
        nbdiplomes: COUNT(e.études) ,
        nbdiplomes03 : COUNT( SELECT c
                              FROM  e.études c
                              WHERE c.année=2003) ,
        premièreannée : MIN( SELECT c.année
                FROM e.études c) )
FROM LesEtudiants e
```

22

Partitionnement d'une collection

- Objectif: partitionner une collection en sous ensembles et de calculer des résultats agrégés sur ces sous-ensembles → partition

Group x in <collection>

by (nom_{11}: <$expression_{11}$> , .. , nom_{1n}: <$expression_{1n}$>)

[with (nom_{21}: <$expression_{21}$> , ... , nom_{2p}: <$expression_{2p}$>)] ;

→ set (struct (nom11:domaine11,... , nom1n:domaine1n , nom21:domaine21,... nom2p:domaine2p))

où domaineij est le domaine de valeurs du résultat de <expressionij>.

Partitionnement d'une collection

```
select e
from employes e
group by (bas : e.salaire < 7000,
          moyen : e.salaire >= 7000 and
          e.salaire < 21000,
          haut : e.salaire >= 21000)
```

===>struct<bas:set(emp.),moyen:set(emp.),haut:set(emp.)>

Partitionnement d'une collection

Group c in LesCours
by (cycle : c.cycle)
with (nbcours : count(partition)) ;

→ donne en résultat un ensemble de couples (cycle:INT, nbcours:INT) qui définissent pour chaque cycle le nombre de cours de ce cycle.

Transformation d'une structure et d'une collection d'un élément

- Un struct composé d'un seul champs est automatiquement converti à la valeur de ce champs
  - Struct(*f* : *x*) converti à *x*.
- Une collection d'un seul élément peut être transformée à cet élément en utilisant l'operateur ELEMENT.
  - ELEMENT(Bag(*x*)) = *x*.

26

Expressions de Collections

- Conversion de collections

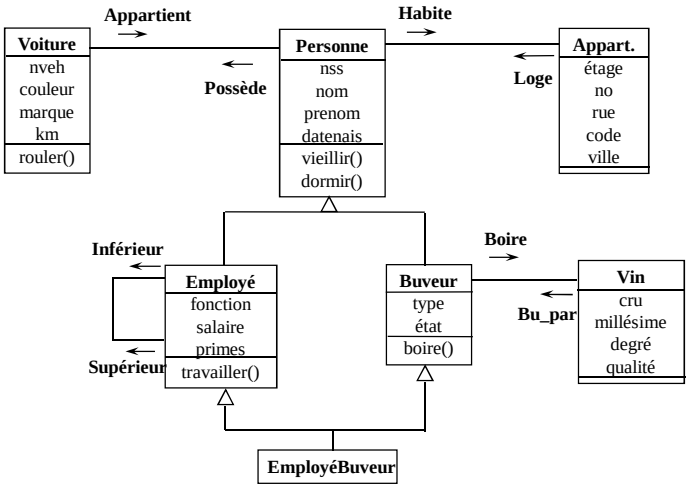
element (select v.marque
from voitures v
where v.numero = "120 abc 75")
===> string

Opérateurs ensemblistes

- Union, except, intersect
- Exemple:

(select e from LesEtudiants e, e.cours_obtenus c
where c.cours.nomC='BD')
except
(select e from LesEtudiants e, e.cours_obtenus c
where c.cours.nomC='ASD') ;

Exemple de base



Parcours d'association monovaluées

- avec sélection de structure

```
select struct (name: b.nom,
               city: b.habite.adresse.ville)
from  buveurs b
where b.type = 'gros'
```

  - ===> littéral bag <struct(Name,City)>

Parcours d'association multivaluées

- Utilisation de collections dépendantes

```
select b.nom, b.prenom
from  buveurs b, b.boire v
where v.cru = "volnay"
```

– ==> litteral bag<struct<nom:string,prenom:string>

Résultats Imbriqués

- Imbrication des select au niveau select

```
select distinct struct (nom : e.nom,
                        (inf_mieux_payes :select i
                                          from  e.inferieur i
                                          where i.salaire > e.salaire))
  from employes e
```

===> litteral de type set <struct (nom: string, inf_mieux_payes :
bag <employes>)>

- et aussi au niveau du FROM (requête collection)

Produit cartésien

- Donne le couple nom de l'étudiant et tous ses prof

```
select couple(student: x.name, professor: z.name)
from Students as x, x.takes as y, y.taught_by as z
where z.rank = "full professor"
```

→Type du résultat: bag d'objets de type couple

```
select *
from Students as x, x.takes as y, y.taught_by as z
where z.rank = "full professor"
```

- Type du résultat: ensemble de structures donnant pour chaque étudiant la section suivie et tous les prof correspondants

Invocation de méthodes

- En résultat ou dans le critère

```
select distinct e.nom,
 e.habite.adresse.ville, e.age()
from employes e
where e.salaire > 10000 and e.age() < 30
```

  - ===> litteral de type set <struct>

Invocation de méthodes

```
select max(select c.age from p.children c)
from Persons p
where p.name = "Paul"
```

```
select p.oldestChild.adrees.street
from Persons p
where p.livesIn("Paris")
```

Valeurs NULL

- 3 employés: le 1er de Paris, le 2ème de Milan et le 3ème n'a pas d'adresse

```
select e
from Employees e
where e.address.city = Paris
```

- Retourne un bag contenant les employés parisiens

```
select e.address.city
from Employees e
```

- Génère une erreur

```
select e.address.city
from Employees e
where is_defined(e.address.city)
```

- Retourne un bag contenant les villes de Paris et Milan

```
select e
from Employees e
where not(e.address.city = Paris)
```

Retourne un bag contenant les employés de Milan et ceux n'ayant pas d'adresses

Définition d'objets via Requêtes

- Define <nom> as <question>
  - permet de définir un objet de nom <nom> calculé par la question
- Exemple:
  - Define Cesars as

```
Select b
From Buveurs b
Where b.prenom = "Jules"
```

Initialisation: attribut

```
struct PhoneNumber {
  unsigned short CountryCode;
  unsigned short AreaCode;
  unsigned short PersonCode;
}
struct Address {
  string Street;
  string City;
  PhoneNumber Phone;
}
interface Person {
  attribute string Name;
  attribute Address PersonAddress
}
Sarah Person { Name "Sarah",
  PersonAdress { Street "Willow Road",
    City "Palo Alto",
    Phone { CountryCode 1,
      AreaCode 415,
      PersonCode 1234 } } }
```

Initialisation: tableau

```
interface Engineer {
  attribute unsigned long PhoneNo[3];
}

   Jane Engineer { PersonID { [0] 450123, [2] 270456} }

   Marc Engineer { PersonID { 12345, 234234 } }
```

```
   struct Point {
     float X; float Y;
   }
   interface Polygon {
     attribute array<Point> RefPoints;
   }
   P1 Polygon { RefPoints { [5] { X 7.5, Y 12.0 },
                            [12] { X 22.5, Y 23.0 } } }
```

Initialisation: tableau & collection

Tableau multi dimensionnels

```
interface PolygonSet {
  attribute array<float> PolygonRefPoints[10];
}
P2 PolygonSet { PolygonRefPoints {
 [0] { [0] 12.5 , [1] 8.98, ...}
 ...
 [1] { [0] 23.7 , [1] 3.33 } } }
```

Collections

```
interface Professor : Person {
  attribute set<string> Degrees;
}
Feynman Professor { Degrees { "Masters", "PhD" }
```

Initialisation: liens

```
interface Person {
  relationship Company Employer inverse Company::Employees;
  relationship Company Property inverse Company::Owner;
}
interface Company {
  relationship set<Person> Employees inverse Person::Employer;
  relationship Person Owner inverse Person::Property;
}
```

Liens avec les relations 1 (traités comme des attributs)

```
Jack Person { Employer McPerth }
```

Liens avec les relations m (traités comme des collections)

```
McPerth Company { Employees { Jack, Joe, Jim } }
```

Relation réflexive

```
Jack Person { Employer McPerth }
McPerth Company { Owner Jack }
```