Le langage OQL

Choix de conception

Description du langage

OQL et ODMG

OQL par l'exemple

Pourquoi un langage de requêtes ?

- **Mode interactif (*ad hoc*) et mode programmation**
- **Optimisable par le système**
- **Dynamicité**
- **Simplicité**
- **Support pour vues, triggers, contraintes d'intégrité**

OQL : choix de conception

- Orthogonalité par rapport au système de types
- Langage fonctionnel
- Déclaratif
- Indépendance par rapport aux stratégies de nommage ou de persistance

Description du langage

OQL est fonctionnel. Une requête est une fonction dont les arguments peuvent être le résultat d'autres fonctions/requêtes.

LePresident

LePresident.cours

count(LePresident.cours)

count(LePresident.cours) > 10

Description du langage (2)

OQL est défini par un ensemble de fonctions/requêtes et la possibilité d'en définir de nouvelles grâce à la composition et à des itérateurs

```
select p.nom

from p in LesProfesseurs

where count(p.cours) > count(LePresident.cours)
```

Description du langage (3)

Entrée : tout objet ou valeur

Sortie : le langage permet

- **l'extraction de tout objet ou valeur existant dans la base**
- **la construction de toute valeur supportée par le modèle**
- **la construction de tout objet dont la classe est définie**

Au sujet de l'encapsulation

- Toutes les propriétés (attributs) d'un objet sont accessibles par la fonction "." (ou "->")
- Avantage des propriétés sur les méthodes:
  elles permettent l'optimisation
- NB : OQL ne permet pas la mise à jour des données (sauf appel de méthode)

Avertissement

Présentation de OQL 1.2

Par rapport à OQL 1.1, OQL 1.2 offre un léger maquillage SQL, le but étant d'arriver à un langage unique (travail aussi sur SQL3)

OQL 1.2 offre parfois plusieurs syntaxes,

notamment : from LesPersonnes p

from p in LesPersonnes

from LesPersonnes as p

from LesPersonnes

Accès à la base

LePresident

LesEmployes

Requêtes relationnelles

Sélection

```
select e
from e in LesEtudiants
where e.nom = "Dupont"
```

Projection

```
select e.nom
from e in LesEtudiants
```

Jointure relationnelle

```
select p.nom
from p in LesProfesseurs,
        e in LesEtudiants
where p.nom = e.nom
```

Expressions de chemin

- LePresident.nom
- LePresident.adresse.ville
- select e.adresse.ville

  from e in LesEmployes

  where e.nom = "Dupont"

Expressions de chemin

```
select e.nom
from p in LesProfesseurs,
        c in p.cours,
        e in c.etudiants
where p.nom = "Dupont"
```

Le résultat est multi-ensemble de string : {{string}}

Expression en calcul

```
{ x | exist p,e,c ( p in LesProfesseurs et
                   e in p^.cours et
                   e in c^.etudiants et
                   p^. nom = "dupont" et
                   e^.nom = x
}
```

Les méthodes

LePresident.salaire()

```
select p.nom
from  p in LesPersonnes
where p.age() > 21
```

Jointure dépendante

```
select p.nom, e.nom
from p in LesProfesseurs,
        c in p.cours,
        e in c.etudiants
```

le résultat est de type {{string x string}}

Construction de valeurs complexes

```
select struct (etudiant : e.nom,
                    prerequis : select p
                                from c in e.cursus,
                                     p in c.prerequis)
from e in LesEtudiants
```

le résultat est de type

{{ [etudiant: string, prerequis: {{Cours}}] }}

Manipulation de valeurs et objets

- **Extraction d'Information**

  LePresident.nom

  LePresident.adresse.ville

- **Construction de valeur**

  struct (rue: "grande rue", ville : "Dourdan")

Manipulation de valeurs et objets

- **Construction d'objet**

  Cours(intitule: "Maths", numero: 101)

- **Combinaison**

  Personne (nom : LePresisent.nom,
  adresse: Adresse(numero : 1,
  rue: "grande rue",
  ville: Dourdan,
  code: 78))

Requêtes nommées

- **Définition :**

```
define lesduponts as
  select p
  from p in LesPersonnes
  where p.nom = "Dupont"
```

- **Utilisation :**

```
select distinct x.adresse.ville
from x in lesduponts
```

Requêtes nommées

- **Durée de vie : une session de mode *ad hoc***

=> pas de requêtes nommées en mode programmation

- **Les requêtes nommées peuvent définir des fonctions paramétrables**

Opérations d'agrégats

- **Sur les collections on peut appliquer des opérateurs d'agrégation : min, max, count, sum, et avg.**
- **Ces opérateurs dépendent du type de la collection traitée.**
- **Si e est une expression dénotant une collection et fa un opérateur d'agrégation alors fa(e) est une expression du langage de requêtes.**
- **Exemple :**
  max (select p.salaire() from p in LesProfesseurs)

Manipulation des collections

- select struct (nom: p.nom,
  coursbd : select c
  from c in p.cours
  where c.intitule like “*BD*”)
  from p in LesProfesseurs
  where count (select c
  from c in p.cours
  where c.intitule like “*BD*”) > 0

Manipulation de collections : tri

- **Tri :**

```
select p
from p in LesPersonnes
order by p.nom, p.age desc
```

Manipulation de collections

- **Regroupement**

```
select *
from e in LesEmployes
group by bas: salaire <20000,
             moyen : salaire < 100000 and
                     salaire >= 20000,
             haut: salaire > 100000
```

- On peut utiliser une astérisque car le type du résultat est automatiquement déduit.

Regroupement (suite)

```
{{
 [ bas: true, moyen: false, haut: false, partition:{{[e:eb1], ...}}],

   [ bas: false, moyen: true, haut: false, partition:{{[e:em1],...}],

   [ bas: false, moyen: false, haut: true, partition:{{[e:eh1], ...}}]
}}
```

partition est de type bag (struct (e1 : t1, e2 : t2, ...)) où e1,e2 sont les noms des variables de la clause from et t1, t2,.. les types associés à ces variables

Regroupement (suite)

```
select dept: d,  moyenne: avg (select e.salaire
                                from e in partition)
from e in LesEmployes
group by d: e.departement
having avg (select e.salaire
            from e in partition) < 100000

struct ( d :string,
         partition : bag ( struct (e: Employe)))
```

Conversions

- element (select c
  from c in LesCours
  where c.intitule = “BD avancees”)

- first (element (select c
  from c in LesCours
  where c.intitule = “BD avancees”)
  .prerequis)

Opérations sur collections

- LesPersonnes except LesEtudiants
- LesEtudiants union set (LePresident)
- element (set (LePresident))
- list (1, 2, 3)
- flatten (select p.cours

  from p in LesProfesseurs)

Polymorphisme

- **select p.age() from p in LesPersonnes**
- **select ((Employe) p).salaire() from p in LesPersonnes**

Indication de type

- **Dans le langage de requêtes on peut forcer une variable à être d'un autre type que le type courant (cast).**
- **(Personne) LePresident**

Quantificateur universel

- **select e.nom**
  from e in LesEtudiants
  where forall c in e.cursus :
  (c.intitule like “*maths*”)

Quantificateur existentiel

- **select e.nom**
  from e in LesEtudiants
  where exists c in e.cursus :
  (exists p in c.enseignants :
  p.nom = “Delobel”)

OQL - Mode programmation

- **oql (var,**

  “select p from p in $1 where p.age < $2”,

  var2, var3)