4.2 Un langage de Requête pour les BD Objet : OQL. Quelques Notions

– *Object Query Language*
– Notation à la SQL.
– Utilisé comme extension d'un langage de programmation hôte, comme C^{++} ou Java.

Format Général d’une requête OQL

```
SELECT liste d’expressions
FROM liste d’une ou plusieurs d\eclarations de variables.
WHERE condition C de selection
```

Une variable est déclarée en indiquant :

1. Une expression dont la valeur a un type collection, par ex. `set` ou `bag`.
2. Le nom de la variable.

L’expression (1) indique l’extension d’une classe, par exemple `Films`. Analogie avec une relation dans une requête SQL.

Notation “.”

Soit o un objet d’une classe C.

– Si p est un attribut, $o.p$ est la valeur de p pou o.
– Si p est une relation, $o.p$ est l’objet ou la collection d’objets reliés à o par p.
– Si p est une méthode (éventuellement avec paramètres $a_1, \cdots a_k$), $o.p(....)$ est le résultat de l’application de p à a.

N.B : Puisque la déclaration de la méthode `NomsActeurs` de la classe `Film` est :
`void NomsActeurs(out Set<String>);`
si `MonFilm` est un objet de la classe `Film`, l’expression `MonFilm.NomsActeurs(mesStars)` ne renvoi pas de valeur, mais, comme effet de bord, pose la valeur de la variable output `mesStars` de la méthode comme étant un ensemble de noms d’acteurs.

Exemple d’une BD à objet pur illustrer OQL

```
class Film
(extent Films key (titre, année))
{
attribute string titre;
attribute integer année;
attribute integer longueur;
attribute enum couleurs { couleur, noir&blanc } SorteFilm;
relationship Set<Star> acteurs inverse Star : :JoueDans;
relationship Studio appartient-à inverse Studio : :possède;
float longeurHeures() raises(pasLongueurTrouvée);
void NomsActeurs(out Set<String>);
void autresFilms (in Star, out Set<Film>) raises(PasActeur);
};

class Star
(extent Stars key nom)
{
attribute string nom;
attribute Struct Adr
{string rue, string ville} adresse;
relationship Set<Film> JoueDans inverse Film : :acteurs;
};

class Studio
(extent Studios key nom)
{
attribute string nom;
attribute string adresse;
relationship Set<Film> :possède inverse Film : :appartient-à;
};
```

Exemples de Requêtes OQL

```
SELECT m.année
FROM Films m
WHERE m.titre = ‘‘Autant en emporte le vent’’
```

Ici, `Films m` est une déclaration d’une variable `m` qui a sa valeur dans l’*extension* `Films` de la classe dont le nom est `Film`, i.e. : valeur(`m`) $\in$ `Films`.

Exemples de Requêtes OQL, suite

```
SELECT s.nom
FROM Films m, m.acteurs s
WHERE m.titre = ‘‘Casablanca’’
```

Dans l’évaluation, tous les couples (m,s) tels que m est un film et s est un acteur de m (selon la relation `acteurs` de `Film`) sont considérés :

FOR chaque m dans FILMS DO
FOR chaque s dans m.acteurs DO
IF m.titre = “Casablanca” THEN
ajouter s.nom au multi-ensemble résultat.

La condition après le `WHERE` limite l’espace de recherche pour `m` aux films dont le titre est “Casablanca”.

Exemples de Requêtes OQL, suite

```
SELECT DISTINCT s.nom
FROM Films m, m.acteurs s
WHERE m.apppartient-à.nom = ‘‘Disney’’
```

Comme en SQL, le mot clé `DISTINCT` élimine le répétitions dans le multi-ensemble (bag) résultat.

Exemples de Requêtes OQL, suite

Liste des noms des films du studio Disney, ordonnés par leur longueur ; si deux films ont la même longueur, trier selon l'ordre alphabétique des titres.

```
SELECT m
FROM Films m
WHERE m.appartient-à.nom = ‘‘Disney’’
ORDER BY m.longueur, m.titre
```

Exemples de Requêtes OQL, suite

On veut l'ensemble des couples de stars qui vivent à la même adresse :

```
SELECT DISTINCT Struct(star1 :s1, star2 : s2)
FROM Stars s1, Stars s2
WHERE s1.adresse = s2.adresse AND s1.nom < s2.Nom
```

Pour chaque couple qui passe le test on produit un record, avec 2 champs, nommés `star1` et `star2`. Le type de chaque champ est la classe `Star`.

Le type du résultat final de cette requête est :

`Set<Struct{star1 : Star, star2 : Star}>`.

Exemples de Requêtes OQL, suite

Utilisation de "sous-requêtes"

Une autre façon de chercher les noms des acteurs des films du studio Disney :

```
SELECT DISTINT s.nom
FROM (SELECT m
FROM Films m
WHERE m.appartient-à.nom =''Disney'') f,
f.acteurs s
```

La variable `m` prend valeur dans la collections des films du studio Disney, qui est le résultat de la sous-requête :

```
SELECT m
FROM Films m
WHERE m.appartient-à.nom =''Disney''
```

La variable `s` prend valeur dans la collections des acteurs du film de Disney `f`. **NB** : un autre `WHERE` inutile ici.

5 Bases Relationnelles-Objet

Le relationnel-objet sur Oracle

Definition de Types

Oracle permet de définir des types du style objet. Syntaxe :

```
CREATE TYPE t AS OBJECT (
    liste d'attributes et methodes
);
/
```

Le slash à la fin sert pour faire traiter par Oracle la définition de type.

Definition de Types, suite

Exemple. Définition d'un type pour représenter un point par ses coordonnées :

```
CREATE TYPE PointType AS OBJECT (
    x NUMBER,
    y NUMBER
);
/
```

Definition de Types, suite

Un type à objet peut être utilisé pour déclarer d'autres types à objet ou relations (au sense traditionnel, mais pas en FN1), par ex. on peut définir un type pour les lignes (géométriques) finies en indiquant les 2 extremités :

```
CREATE TYPE LineType AS OBJECT (
    end1 PointType,
    end2 PointType
);
/
```

Definition de Types, suite

On peut alors crér une relation (presque traditionnelle) qui est un ensemble de lignes, où chaque ligne a un OID :

```
CREATE TABLE Lines (
    lineID INT,
    line   LineType
);
```

N.B.

a) Analogie avec le `Set<LineType>` de ODL.

b) Ici, on a un attribut (`LyneType`) qui est de type non-atomique.

Elimination de Types

Pour éliminer un type comme `LineType` :

```
DROP TYPE Linetype;
```

(“to drop” en anglais : faire tomber, jeter.)

N.B. On doit d’abord effacer toutes les tables et les autres types qui utilisent ce type (dans l’exemple : la table `Lines`).

Construction d'objets (valeurs). Constructeurs pre-définis pour crér des valeurs, dont les noms sont les noms des types.

Suite de l'exemple. Pour crér une valeur de type `PointType` : le mot `PointType` (nom du type) suivi par les valeurs en ().
Construction d'une ligne de la table `Lines` ayant idéntificateur 27, qui part du point (0,0) et arrive au point (3,4) :

```
INSERT INTO Lines
    VALUES(27, LineType(
                    PointType(0.0, 0.0),
                    PointType(3.0, 4.0)
                )
    );
```

N.B. On utilise `INSERT` comme pour l'insertion dans une table standard. On a créé aussi 2 valeurs de type `PointType` et 1 valeur de type `LineType`.

Déclarations de méthodes. Une déclaration de type peut inclure la déclaration de méthodes, à l'aide de `MEMBER FUNCTION` ou `MEMBER PROCEDURE`. Comme en ODL, il faut spécifier les types des entrées et sortie.

Suite de l'exemple. On ajoute une fonction `lenght` au type `LineType`, qui produit la longueur d'une ligne et la multiplie par un facteur.

```
CREATE TYPE LineType AS OBJECT (
    end1 PointType,
    end2 PointType,
    MEMBER FUNCTION length(scale IN NUMBER) RETURN NUMBER,
    PRAGMA RESTRICT_REFERENCES(length, WNDS)
);
/
```

La dernière instruction (`PRAGMA...`) dit que la méthode ne change pas la BD. (WNDS = *write no database state*).

Définition du code d'une méthode

Le codede la méthode est donné à part, dans une instruction `CREATE TYPE BODY`. Ici, on ne répéte pas les types des entrés et des sorties (← des procédures). La variable spéciale `SELF` dans la déf. de la méthode indique la valeur courante.

Suite de l'exemple

```
CREATE TYPE BODY LineType AS
   MEMBER FUNCTION length(scale NUMBER) RETURN NUMBER IS
       BEGIN
         RETURN scale *
            SQRT((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-SELF.
                 (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-SELF.
            );
       END;
END;
/
```

Quelques exemples de requête relationnelle-objet

Calcul du double de la longueur des chaque ligne :

```
SELECT lineID, ll.line.length(2.0)
 FROM Lines ll;
```

La requête utilise la méthode `leghth`.

Il faut utiliser un alias (ou variable) (ici, `ll`) pour acceder à un champ avec la notation "`.`". En fait, `line` tout seul et `Lines.line` ne marchent pas.

Coordonnées de la prèmiere extremité de chaque ligne :

```
SELECT ll.line.end1.x, ll.line.end1.y
     FROM Lines ll;
```

Types Références

Si t est un type, REF t est les type des références (OID) à des valeurs de type t.

Exemple : définition d'un schéma de table avec REF.

Une relation Lines2 dont les lignes sont des couples de références à de points :

```
CREATE TABLE Lines2 (
   end1 REF PointType,
   end2 REF PointType
);
```

Exemple, suite : définition du contenu dune table avec `REF`.

Supposons qu'on a déjà une table `Points` dont les valeurs sont de type `PointType`. On remplie la table `Lines2` avec des valeurs qui sont toutes les lignes dont les extremités (gauche et droite) sont des éléments de la table `Points` :

```
INSERT INTO Lines2
    SELECT REF(pp), REF(qq)
    FROM Points pp, Points qq
    WHERE pp.x < qq.x;
```

Attention : On ne peut pas inventer un objet qui n'est pas déjà dans une relation et le référencer. Par ex. ceci ne marchera pas :

```
INSERT INTO Lines2
    VALUES(REF(PointType(1,2)), REF(PointType(3,4));
```

Les objets `PointType(1,2)` et `PointType(1,2)` ne vivent dans aucune relation !

Pour suivre une référence avec le `.`, on fait comme si la valeur d'un attribut de type référence était litteralement la valeur du type référencé.

Exemple : Calcul des coordonnées x des extremité de chaque ligne élément de `Lines2` :

```
SELECT ll.end1.x, ll.end2.x
FROM Lines2 ll;
```

Relations imbriquées, dćlaration du schéma

La valeur d'un attribut d'une table peut être une relation.

Définition du schéma d'une table dont les élément sont des polygones, tout polygone étant une table (de points) lui-même.

```
CREATE TYPE PolygonType AS TABLE OF PointType;
/

CREATE TABLE Polygons (
    name   VARCHAR2(20),
    points PolygonType)
    NESTED TABLE points STORE AS PointsTable;
```

Dernière ligne : les sous-tables réprésentant les polygones ne sont pas stockées directement comme valeurs de l'attribute `points`, mais dans une autre table dont le nom est `PointsTable` (la façon de stocker est celle du modèle relationnel standard !).

Relations imbriquées, remplissage des tables

La valeur de chaque relation de niveau inférieur est représentée par une liste de valeurs du type approprié (`PolygoneType` dans notre exemple).

Exemple d'insertion d'un polygone qui est un carré :

```
INSERT INTO Polygons VALUES(
    'square', PolygonType(PointType(0.0, 0.0), PointType(0.0,
                          PointType(1.0, 0.0), PointType(1.0,
                         )
);
```

Relations imbriquées, exemples de requêtes

Les points du carré :

```
SELECT points
FROM Polygons
WHERE name = 'square';
```

Le point du carré qui sont sur la diagonale (x=y) :

```
SELECT ss.x
FROM THE(SELECT points
         FROM Polygons
         WHERE name = 'square'
        ) ss
WHERE ss.x = ss.y;
```

(sous-requête, relation du niveau inférieur dans le `FROM` avec le mot-clé `THE` et utilisation de la variable `ss` pour la nommer.)